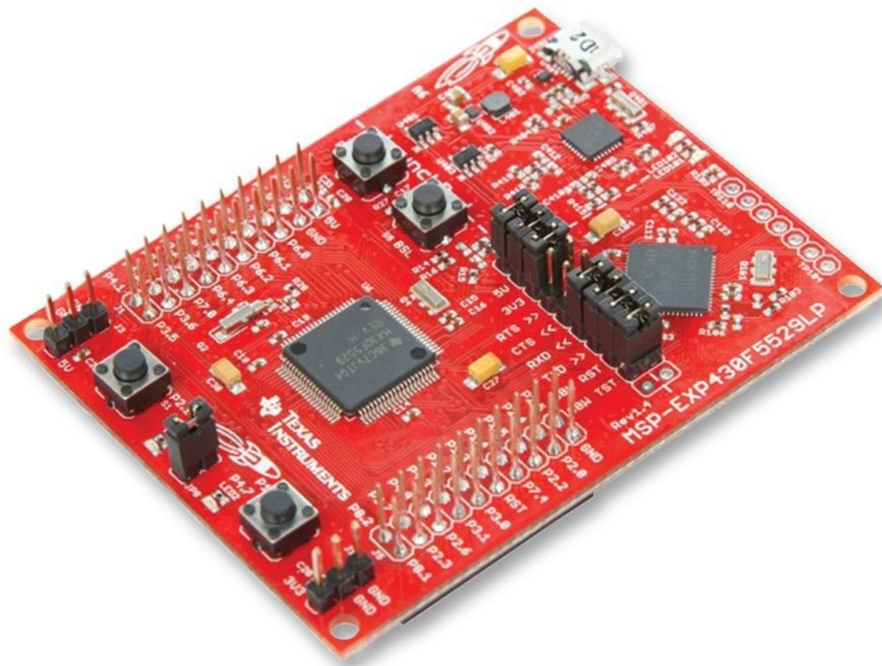
Tinkering TI MSP430F5529

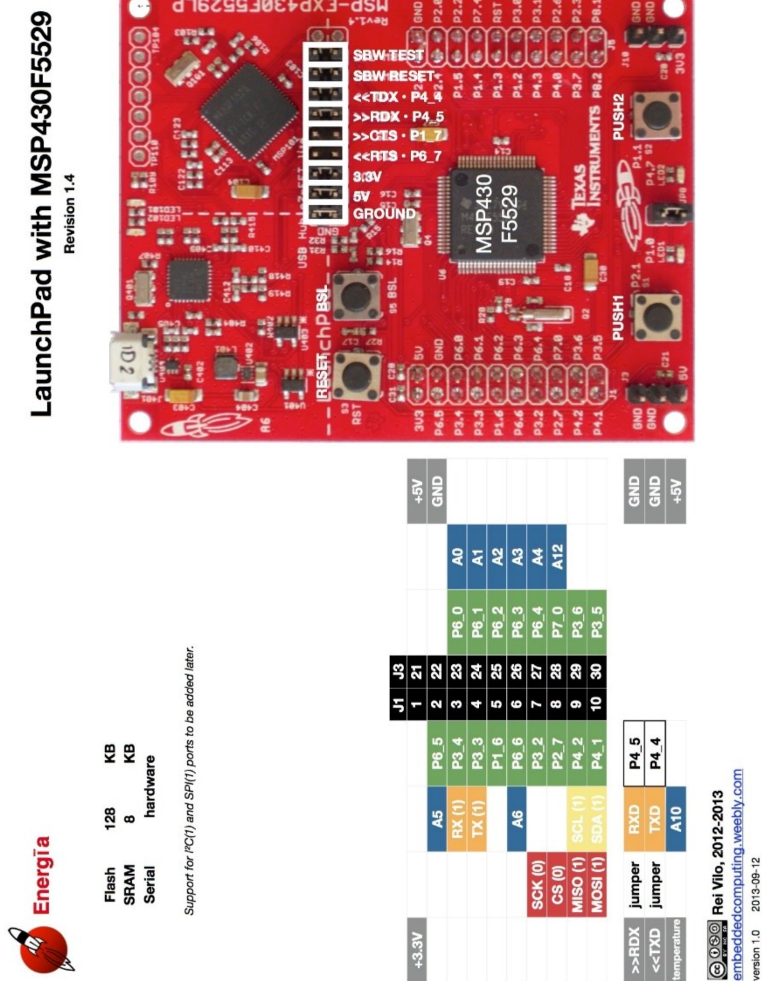
In my past tutorials on MSP430s, I demonstrated how to get started with MSP430 general purpose microcontrollers from ***Texas Instruments (TI)***. Those tutorials covered most aspects of low and mid-end MSP430G2xxx series microcontrollers. For those tutorials, TI's official software suite – ***Code Composer Studio (CCS)*** – an Eclipse-based IDE and ***GRACE*** – a graphical peripheral initialization and configuration tool similar to STM32CubeMX were used. To me, those low and mid-end TIs chips are cool and offer best resources one can expect at affordable prices and small physical form-factors. I also briefly discussed about advanced MSP430 microcontrollers and the software resources needed to use them effectively. Given these factors, now it is high time that we start exploring an advanced 16-bit TI MSP430 microcontroller using a combination of past experiences and advanced tools. MSP430F5529 is such a robust high-end device and luckily it also comes with an affordable Launchpad board dedicated for it.



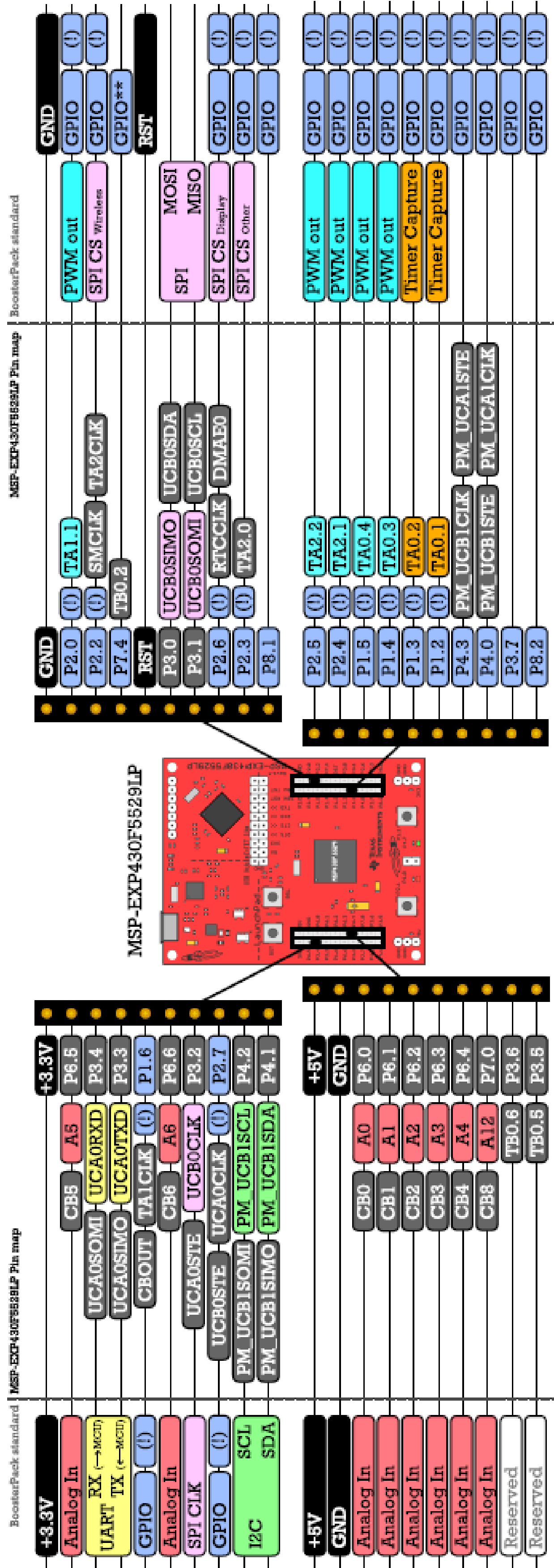
First of all, MSP430F5529 is a monster microcontroller because it offers large memory spaces– 128kB of flash and 8kB of RAM. Secondly, it is a 16-bit RISC microcontroller that host several advanced features like USB, DMA, 12-bit SAR ADC, analogue comparator, a real time clock (RTC), several sophisticated timers with multiple capture-compare I/O channels, etc. There is an on-chip hardware multiplier apart from several bidirectional general-purpose digital input-output (DIO/GPIO) pins, multipurpose communication peripherals (USCIs) and a highly complex clock and power system that can be tweaked to optimize the speed-power performances. The MSP430F5529LP microcontroller featured on-board MSP430F5529 Launchpad is a low power energy-efficient variant of MSP430F5529 and hence the *LP* designation at the end of the part naming. It can run at low voltage levels like 1.8V. In short, it is a hobbyist wildest dream come true.

MSP-EXP430F5529LP Launchpad Board

The MSP-EXP430F5529LP Launchpad board is just like other Launchpad boards, having similar form-factor and layout. Unlike the previously seen MSP-EXP430G2 Launchpads, its header pins are brought out using dual-sided male- female rail-connectors. These allow us to stack ***BoosterPacks*** on both sides.

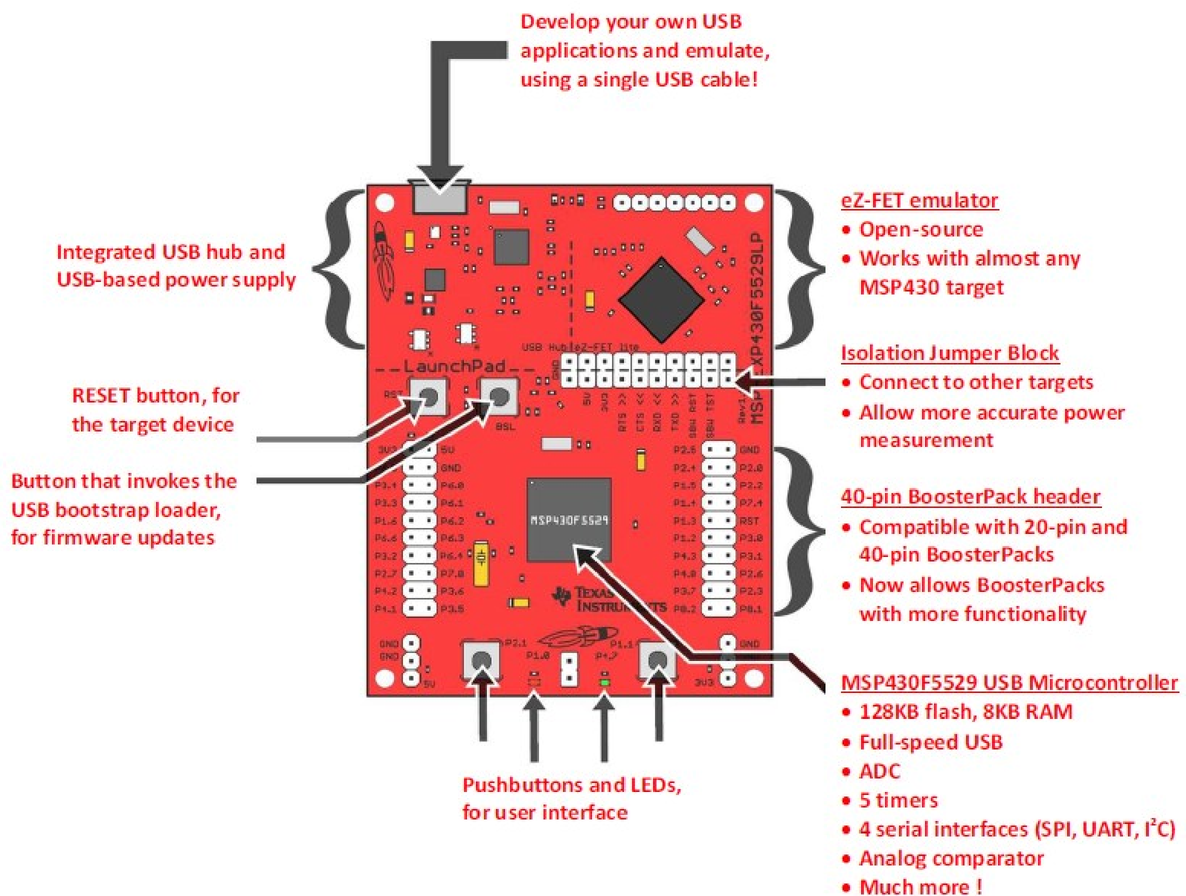


Shown above and below are the pin maps of this Launchpad board.



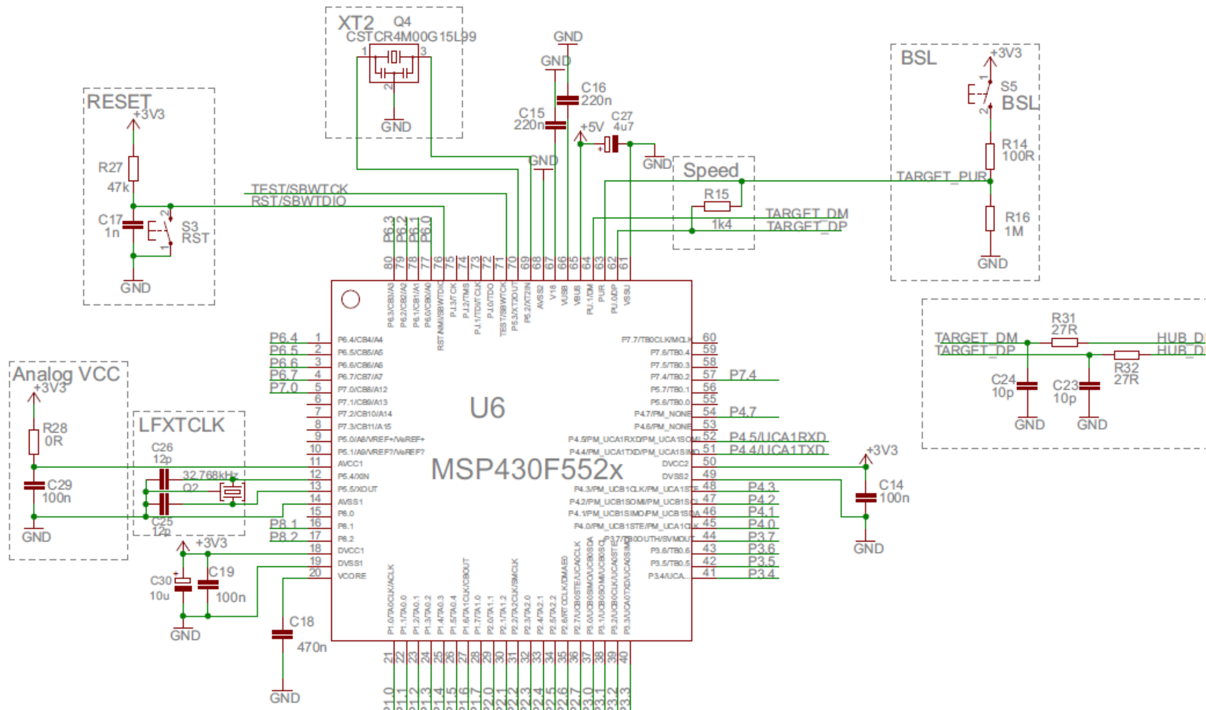
The first pin map is useful for ***Energia*** users and the second one is useful for CCS users. I use a combination of both.

Like other Launchpad boards, this Launchpad comes with a separable MSP-FET USB programmer and debugger interface. Since MSP430F5529 micro has USB hardware embedded in it, the on-board micro USB port can act as a physical USB port when developing USB-based hardware. The micro USB port isn't, however, directly connected with the chip. Instead of that, the same USB port is connected to the on-board FET programmer and the MSP430F5529 chip via an on-board USB hub. The micro USB port also provides power to the board. The on-board MSP430 chip is powered by 3.3V LDO regulator. This LDO is not meant for driving large loads. There are sufficient power rail pins to hook-up low-power external devices like sensors, LCDs, etc without the need of power rail extenders.



MSP430F5529 is not a low pin count micro and it has a sophisticated clock system that can be feed with both internal and external clock sources. Thus, unlike MSP430G2xxx Launchpads, this Launchpad comes with two external crystals – a 4MHz and a 32.768kHz crystal that are physically connected with GPIO pins. There are two user buttons and two LEDs also. Additionally, there are dedicated buttons for USB bootstrap loader and hardware reset. There are few jumpers that can be for measurements, debugging, programming, etc.

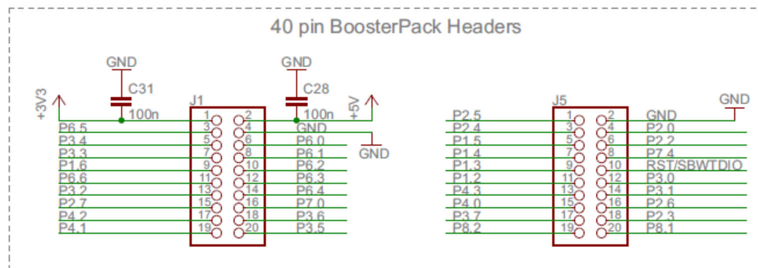
Shown below is the schematic of MSP430F5529LP Launchpad board:

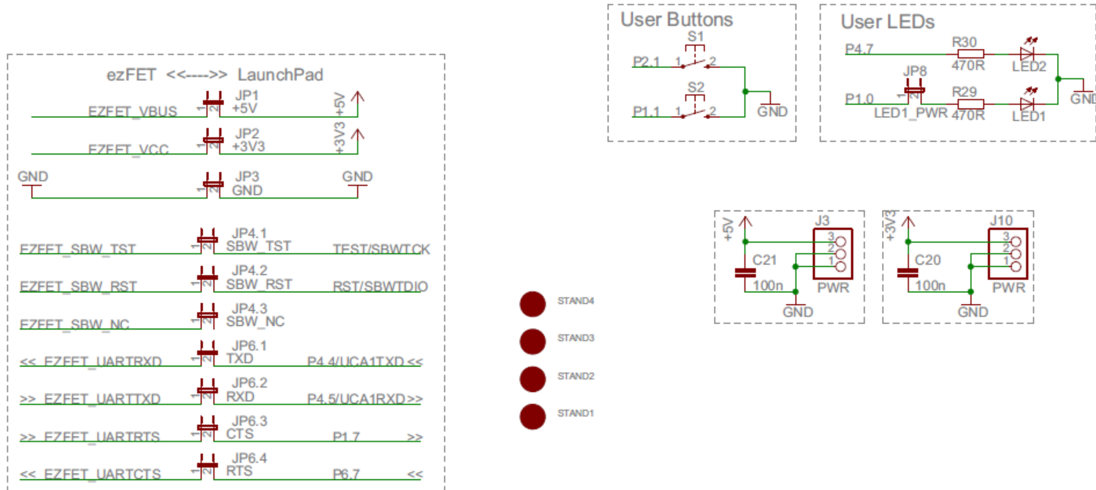


Design Notes:

Remove R28 and inject AVCC directly if needed

Free IO pins: 4.6, 5.0, 5.1, 5.6, 5.7, 7.1, 7.2, 7.3, 7.5, 7.6, 7.7, 8.0, PJ.x (4)





TI MSP430Ware Driver Library and Other Stuffs

Like with other advanced microcontrollers from TI, there is no GRACE-like support that can be found for MSP430F5529 and similar devices. In fact, GRACE only supports few low-end and mid-end MSP430 microcontrollers and for some reason TI stopped updating this great tool since 2013. One probably reason may be the difficulty in developing a GUI-based application that can be used across all computer platforms (Windows, Linux, Mac OS, etc) while maintaining support for all ever-growing families of MSP430 devices.

Instead of upgrading GRACE, TI invested on something called ***MSP430Ware Driver Library*** or simply ***Driverlib***. Driverlib replaces traditional register-level coding with a set of dedicated hardware libraries that reduces learning and implementation curves. These libraries consist of functions, variables and constants that have meaningful names. A sample code is shown below:

```
Timer_A_outputPWMParam outputPWMParam = {0};

outputPWMParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
outputPWMParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_2;
outputPWMParam.timerPeriod = 20000;
outputPWMParam.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
outputPWMParam.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
outputPWMParam.dutyCycle = 0;

Timer_A_outputPWM(TIMER_A2_BASE, &outputPWMParam);
```

Note that there is no register-level coding involved and that's the beauty. Here all the settings of a timer in PWM mode are being set. Each parameter is named with a meaningful name instead of some meaningless magic numbers and coding jargon.

Initially, I was doubtful and hesitant about using Driverlib because it hides away all the good-old register-level coding and at the same time it lacks good documentations. Comparing Driverlib documentation with the documentation of other manufacturers like STMicroelectronics or Microchip, I would say that Driverlib documentation is somewhat incomplete, clumsy and difficult to perceive. Adding to this is the fact that TI while developing and updating Driverlib made some changes to some function and definition names. This creates further confusion unless you are using the right version of documentation alongside the right Driverlib version.

Here's one example. Both are correct and the compiler won't throw any unexpected error but the first one is seen in older documentations while the second is used more often now.

```
Timer_A_startCounter(__MSP430_BASEADDRESS_T0A5__, TIMER_A_CONTINUOUS_MODE);

                                 or

Timer_A_startCounter(TIMER_A0_BASE, TIMER_A_CONTINUOUS_MODE);
```

Issues like this one discussed above may not always be that simple and when it comes to older codes/examples, thing may become very ugly.

Another disadvantage that comes along with stuffs like Driverlib is resource management. Indeed, Driverlib reduces coding efforts a lot but at the expense of valuable and limited memory resources. Performance is also a bit compromised as execution speed is reduced due to additional hidden coding.

Despite these facts, I decided to go along with Driverlib because like other coders I didn't want to spend time going through register-by-registers. In present day's embedded system arena, people like the easy, effective and quick paths. It happened to me that after few trials I was able to master Driverlib and could also pinpoint issues with documentation and even practically get rid of the issues. As of this moment, I am enjoying it a lot and this whole tutorial is based on it.

Use ***Resource Explorer*** of Code Composer Studio (CCS) IDE or visit the following link:

http://www.ti.com/tool/MSPWARE

to get access to MSP430Ware. MSP430Ware contains lot example codes, libraries and other stuffs.

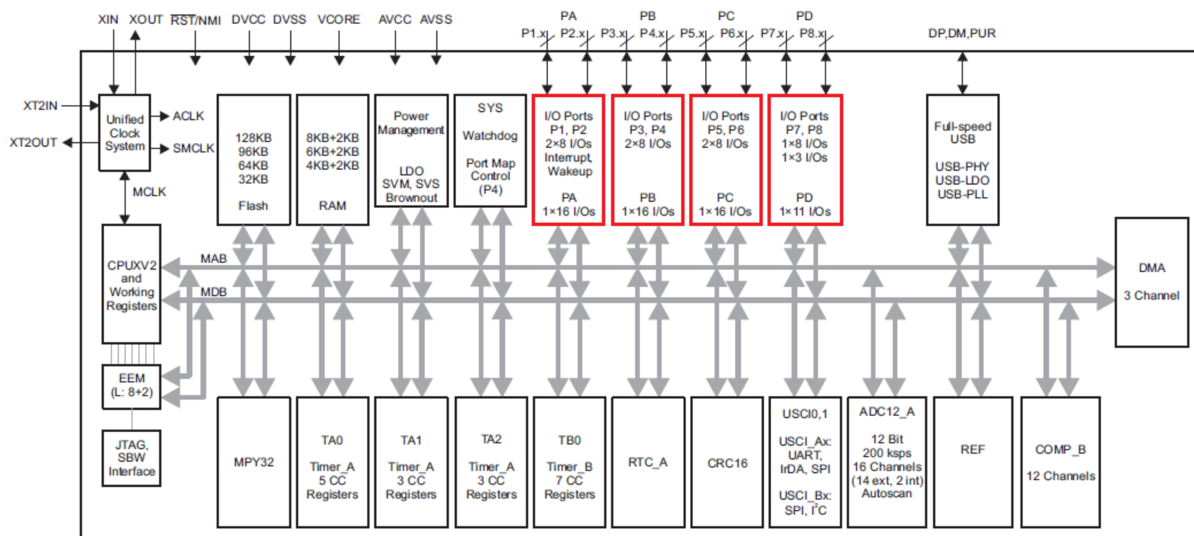
The rest of the software suite and hardware tools will be same as the ones used before. Grace support is unavailable and so it won't be used.

Please also download MSP430F5529 documentations and Launchpad board resources.

Digital Input-Output - DIO

Digital Input-Output (DIO) *a.k.a* General-Purpose Input-Output (GPIO) coding is the basic and simplest requirement for any microcontroller. This is one area where a microcontroller (MCU) differs from a microprocessor (MPU).

MSP430F5529LP Launchpad comes with 40-pin dual-in-line headers. Of these 40 pins, 35 pin headers are connected to DIO pins. There seven DIO ports but they are not evenly divided, i.e. not all ports are 8-bit wide as one would typically expect. Most of the DIOs have more than one functionality, i.e. timer, ADC, communication, etc. DIO pins can be individually programmed as either inputs or outputs. There are independent input-output data registers. In input mode, DIOs can be pulled up or pulled down using internal pull resistors just like other MSP430s. To take care of Electromagnetic Interference (EMI) issues, output drive strength can be altered in output mode, adding incredible robustness. Additionally, pins can operate at high frequency (25MHz) conditions.



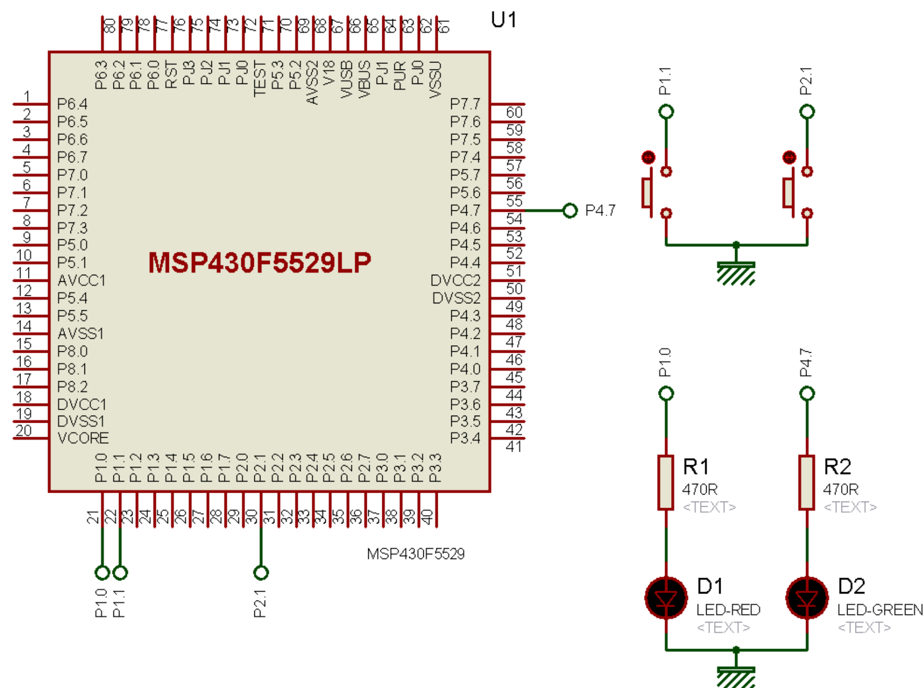
Note in the diagram above that ports are grouped as ***PA***, ***PB***, ***PC*** and ***PD***. Ports are usually 8-bit wide but after adjacently grouping them as such, they become 16-bit wide. PA port pins additionally have interrupt capability unlike other ports. PA port consists of P1 and P2, PB port consists of P3 and P4 and so forth. Port grouping is helpful when we need to drive parallel port interfaces like TFT displays.

It is advised not to exceed output drive current or input voltage ranges/polarity. Best practices are to avoid driving loads directly via output pins and using buffer ICs or isolation for input pins. We must check that whether we are not overloading the microcontroller pins any how because this may have undesired consequences. Try to keep MCU's total current consumption as low as possible. Any device is more stable at low power and optimum temperature conditions than otherwise. Remember MSP430s are low-power device.

Code Example

```c
#include "driverlib.h"


void GPIO_init(void);


void main(void)
{
    WDT_A_hold(WDT_A_BASE);

    GPIO_init();

    while(1)
    {
        if(GPIO_getInputPinValue(GPIO_PORT_P1,
                                 GPIO_PIN1) == 0)
        {
            GPIO_setOutputHighOnPin(GPIO_PORT_P4,
                                    GPIO_PIN7);

            GPIO_setOutputLowOnPin(GPIO_PORT_P1,
                                   GPIO_PIN0);
        }

        if(GPIO_getInputPinValue(GPIO_PORT_P2,
                                 GPIO_PIN1) == 0)
        {
            GPIO_setOutputHighOnPin(GPIO_PORT_P1,
                                    GPIO_PIN0);

            GPIO_setOutputLowOnPin(GPIO_PORT_P4,
                                   GPIO_PIN7);
        }
    };
}


void GPIO_init(void)
{
    GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1,
                                         GPIO_PIN1);

    GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P2,
                                         GPIO_PIN1);

    GPIO_setAsOutputPin(GPIO_PORT_P1,
                        GPIO_PIN0);

    GPIO_setDriveStrength(GPIO_PORT_P1,
                          GPIO_PIN0,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setDriveStrength(GPIO_PORT_P4,
                          GPIO_PIN7,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);
}
```

Hardware Setup



Explanation

The demo here alternatively turns on/off the pair of on-board LEDs with on-board button presses. Since this is the first example, I did not configure clocks and so the default clock settings are used. There is nothing dependent on timing.

Inputs are configured as inputs with internal pull-up resistor. This is so because the on-board buttons are directly connected the GPIO pins without any pull resistor.

```
GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1, GPIO_PIN1);
```

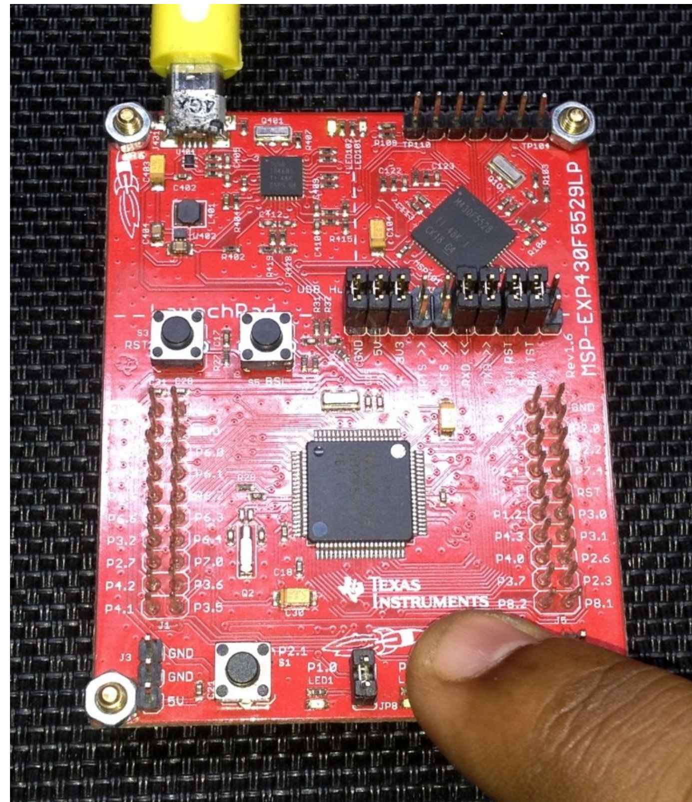
Outputs are configured full output drive output pins.

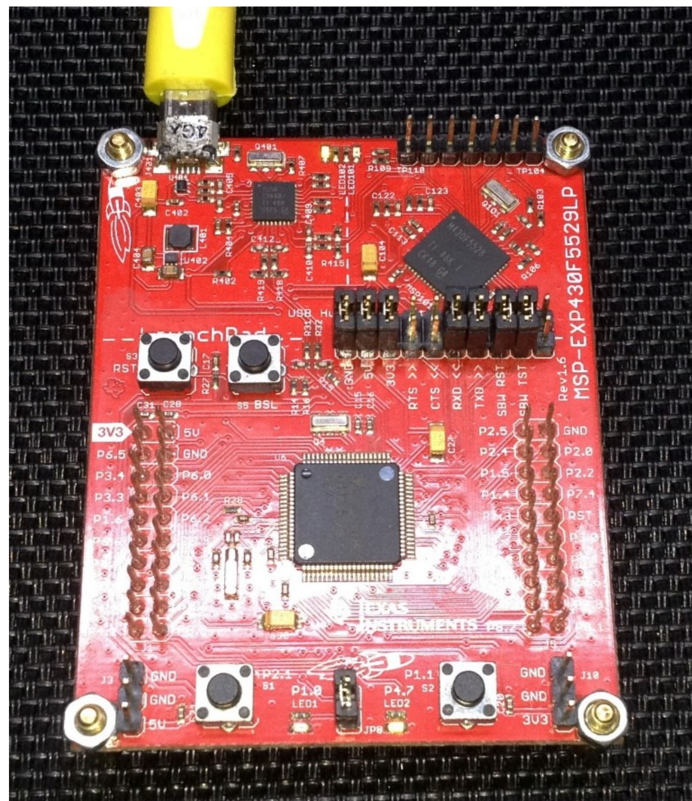
```
GPIO_setAsOutputPin(GPIO_PORT_P4, GPIO_PIN7);

GPIO_setDriveStrength(GPIO_PORT_P4, GPIO_PIN7, GPIO_FULL_OUTPUT_DRIVE_STRENGTH);
```

In the main loop, the LED pins are toggle alternatively with button presses.

```
if(GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1) == 0)
{
    GPIO_setOutputHighOnPin(GPIO_PORT_P4, GPIO_PIN7);
    GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
}

if(GPIO_getInputPinValue(GPIO_PORT_P2, GPIO_PIN1) == 0)
{
    GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
    GPIO_setOutputLowOnPin(GPIO_PORT_P4, GPIO_PIN7);
}
```

Demo

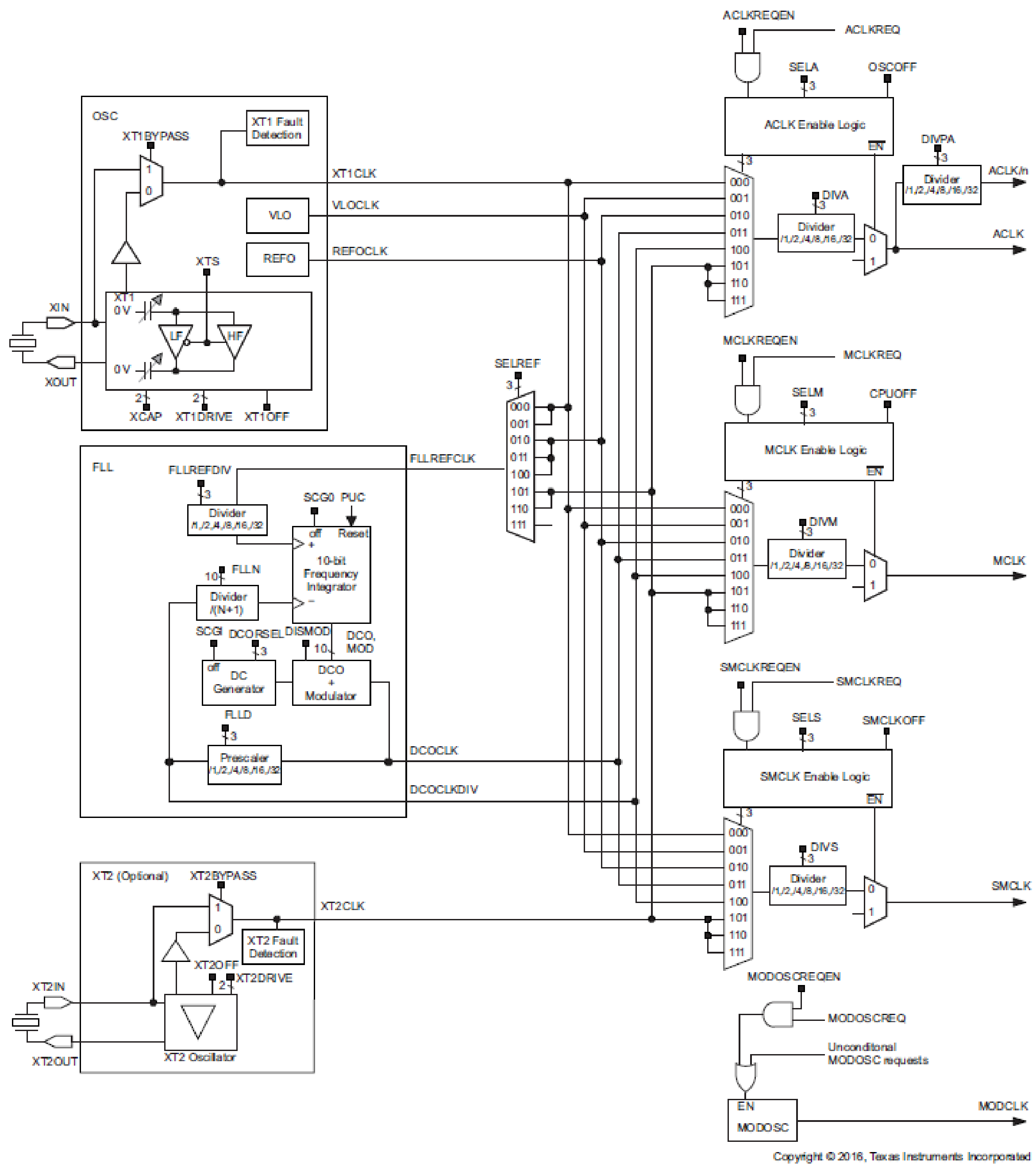




Video demo: https://youtu.be/6b1a15OIddg

Unified Clock System - UCS

MSP430F5529 just like other MSP430s have a very sophisticated clock system called Unified Clock System (UCS). This is a modern trend for most microcontrollers. The goal of such sophistications is to allow users to balance between performance and power optimization with minute compromises. Shown below is the block diagram of UCS. Though it may appear complicated, it has two main parts - clocks signals and their sources.



In a MSP430F5529 micro, there are three clock signals or simply clocks that drive internal hardware peripheral modules and the CPU. These clocks can be feed with different sources independently and have independent divider/scalars. Different clocks have different primary uses. These clocks are as follows:

- ***Master Clock – MCLK***
    - Used primarily for CPU.
    - Usually very fast and usually fastest amongst other clocks.
- ***Sub-Master Clock – SMCLK***
    - Used for hardware peripheral modules like timers, ADC, USCI, etc.
    - Can be as fast as MCLK.
- ***Auxiliary Clock – ACLK***
    - Can be used for hardware peripheral modules specially the low power ones.
    - Typically, slower than other clocks and acts as a tertiary clock.

The following clock sources can independently feed the aforementioned clock signals:

- ***Very Low Oscillator Clock – VLOCLK***
    - On-chip very low frequency RC oscillator (about 10kHz).
    - Not very accurate.
    - Intended for idle/sleep modes and timing insensitive tasks.
- ***Reference Oscillator Clock – REFOCLK***
    - On-chip reference oscillator (32.768kHz).
    - Moderately accurate.
- ***Digitally Controlled Oscillator – DCO***
    - Digitally-controlled fast oscillator source.
    - Stabilized Frequency-Locked-Loop (FLL).
- ***External Oscillator 1 – XT1CLK***
    - In a Launchpad board, this is connected to an external onboard 32.768kHz crystal.
    - Useful for driving the RTC module.
    - Can be feed with other external crystals or external clock sources from 4 – 32MHz.
- ***External Oscillator 2 – XT2CLK***
    - In a Launchpad board, this is connected to an external onboard 4 MHz crystal.
    - Can be feed with crystals or external clock sources from 4 – 32MHz.

All of these sources and clock signals can be applied in a vast number of combinations, allowing us to set, fine tune and run the ***Digitally Controlled Oscillator (DCO)*** at an astonishing speed of 25MHz.

Code Example

delay.h

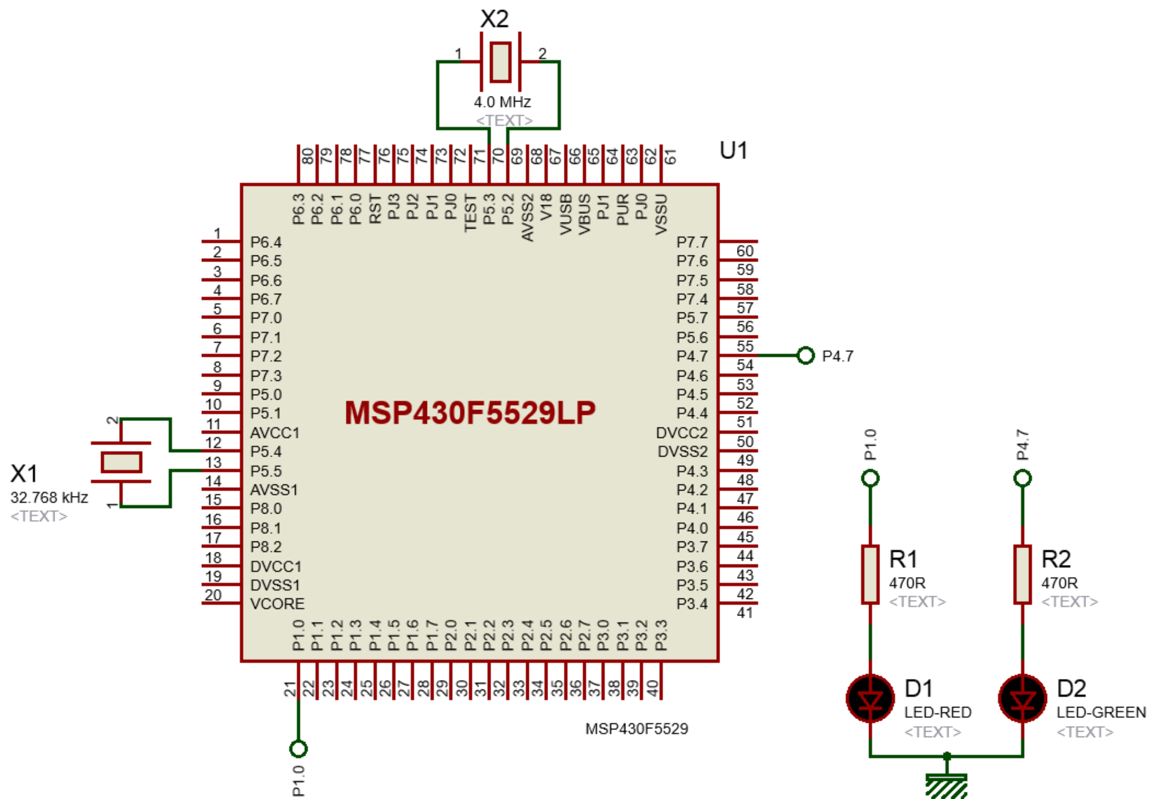
```
#define XT1_FREQ          32768
#define XT2_FREQ          4000000
#define MCLK_FREQ         25000000

#define XT1_KHZ           (XT1_FREQ / 1000)
#define XT2_KHZ           (XT2_FREQ / 1000)
#define MCLK_KHZ          (MCLK_FREQ / 1000)

#define scale_factor       4

#define MCLK_FLLREF_RATIO (MCLK_KHZ / (XT2_KHZ / scale_factor))

#define CPU_F             ((double)MCLK_FREQ)

#define delay_us(delay) __delay_cycles((long)(CPU_F*(((double)delay)/1000000.0)))
#define delay_ms(delay) __delay_cycles((long)(CPU_F*(((double)delay)/1000.0)))
```

main.c

```
#include "driverlib.h"
#include "delay.h"


void indicate(void)
{
    GPIO_setOutputHighOnPin(GPIO_PORT_P4, GPIO_PIN7);

    delay_ms(20);

    GPIO_setOutputLowOnPin(GPIO_PORT_P4, GPIO_PIN7);

    delay_ms(20);
}


void toggle_LED(void)
{
    signed char i = 2;

    while(i > -1)
    {
        GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);

        delay_ms(40);
        i--;
    };

    GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
}


void main(void)
{
    WDT_A_hold(WDT_A_BASE);

    GPIO_setAsOutputPin(GPIO_PORT_P1,
                        GPIO_PIN0);
```

```
    GPIO_setDriveStrength(GPIO_PORT_P1,
                          GPIO_PIN0,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setDriveStrength(GPIO_PORT_P4,
                          GPIO_PIN7,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    indicate();
    toggle_LED();

    UCS_initClockSignal(UCS_MCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    indicate();
    toggle_LED();

    UCS_initClockSignal(UCS_MCLK,
                        UCS_REFOCLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    indicate();
    toggle_LED();

    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_REFOCLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    indicate();

    while (1)
    {
        toggle_LED();
    };
}
```

Hardware Setup



Explanation

Unless altered or set by coder, the default clock speeds of the three clocks are as follows:

- **MCLK ≈ 1.0 MHz**
- **SMCLK ≈ 1.0 MHz**
- **ACLK = 32.768 kHz**

However, we don't want to use these defaults as it is pointless to use so when we have better options. We can alter clock settings and tune the ***Digitally Controlled Oscillator (DCO)*** to run at 25MHz which is, by the way, the max clock speed of MSP430F5529 micro.

The demo here demonstrates how clock speed affects LED blinking over the same amount of delay. Different clocks are initiated with different settings and LED blinking is monitored. The onboard green LED (*indicate function*) indicates a change in clock settings while the onboard red LED (*toggle function*) is blinked as an indicator of speed. This is possibly the simplest arrangement.

```
void indicate(void)
{
    GPIO_setOutputHighOnPin(GPIO_PORT_P4, GPIO_PIN7);
    delay_ms(20);

    GPIO_setOutputLowOnPin(GPIO_PORT_P4, GPIO_PIN7);
    delay_ms(20);
}


void toggle_LED(void)
{
    signed char i = 2;

    while(i > -1)
    {
        GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
        delay_ms(40);
        i--;
    };

    GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
}
```

At first, no clock settings are applied and so the clocks operate at their respective default speeds. Note that software delays are dependent on MCLK frequency and so LED flashing will differ with different MCLK speeds. Please refer to delay library header file. In the code, the delay amounts are not adjusted with clock speeds.

```
#define XT1_FREQ          32768
#define XT2_FREQ          4000000
#define MCLK_FREQ         25000000

#define XT1_KHZ           (XT1_FREQ / 1000)
#define XT2_KHZ           (XT2_FREQ / 1000)
#define MCLK_KHZ          (MCLK_FREQ / 1000)

#define scale_factor       4

#define MCLK_FLLREF_RATIO (MCLK_KHZ / (XT2_KHZ / scale_factor))

#define CPU_F             ((double)MCLK_FREQ)

#define delay_us(delay)    __delay_cycles((long)(CPU_F*(((double)delay)/1000000.0)))
#define delay_ms(delay)    __delay_cycles((long)(CPU_F*(((double)delay)/1000.0)))
```

In the code, you'll notice that only MCLK is altered for the aforementioned reason.

```
UCS_initClockSignal(UCS_MCLK, UCS_XT2CLK_SELECT, UCS_CLOCK_DIVIDER_1);
....
UCS_initClockSignal(UCS_MCLK, UCS_REFOCLK_SELECT, UCS_CLOCK_DIVIDER_1);
```

Up to this point only internal sources have been employed but we do have options for external and high frequency clocks. To use on-board external crystals, we have to initialize their respective pins.

```
GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5, (GPIO_PIN4 | GPIO_PIN2));

GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5, (GPIO_PIN5 | GPIO_PIN3));
```

These pins should be initiated as alternative/secondary function pins. Port Mapping Controller (PMC) does this job in background. If these pins are not initialized properly as such, the crystals may not function at all and we may end up with an inert lifeless microcontroller.

As per onboard crystal placement, keep in mind that

- XT1 = 32.768 kHz
- XT2 = 4.0 MHz

With the pins initialized, we are ready to set the external sources as per their respective frequencies.

```
UCS_setExternalClockSource(XT1_FREQ, XT2_FREQ);
```

We can directly use these sources for clocks:

```
UCS_initClockSignal(UCS_ACLK, UCS_XT1CLK_SELECT, UCS_CLOCK_DIVIDER_1);
```

We can also use these sources to tune the DCO. To tune the DCO, we must initialize the ***Frequency Locked Loop (FLL)*** using either internal or external sources. First, the FLL's reference clock source is initialized.

```
UCS_initClockSignal(UCS_FLLREF, UCS_XT2CLK_SELECT, UCS_CLOCK_DIVIDER_4);
```

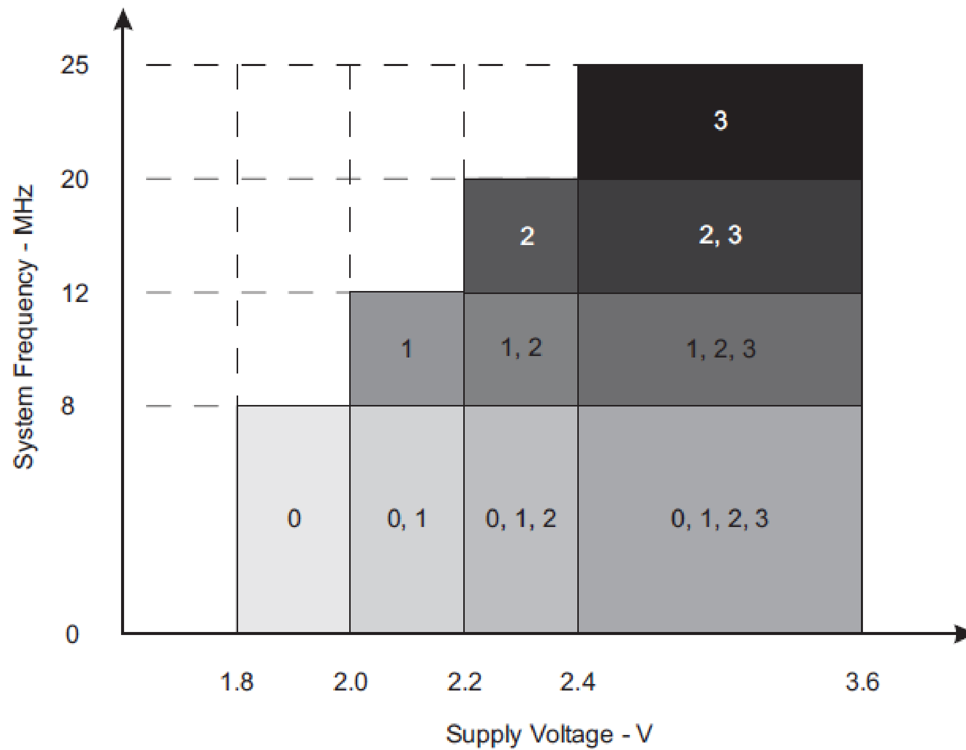
After this, the FLL is allowed to stabilize or settle.

```
UCS_initFLLSettle(MCLK_KHZ, MCLK_FLLREF_RATIO);
```

The following lines in the delay header file are important because their values alter MCLK frequency.

```
#define MCLK_KHZ          (MCLK_FREQ / 1000)

#define scale_factor       4

#define MCLK_FLLREF_RATIO (MCLK_KHZ / (XT2_KHZ / scale_factor))
```

Another thing that is very important to note is the relation between system frequency and core power mode. The graph below shows this relation.

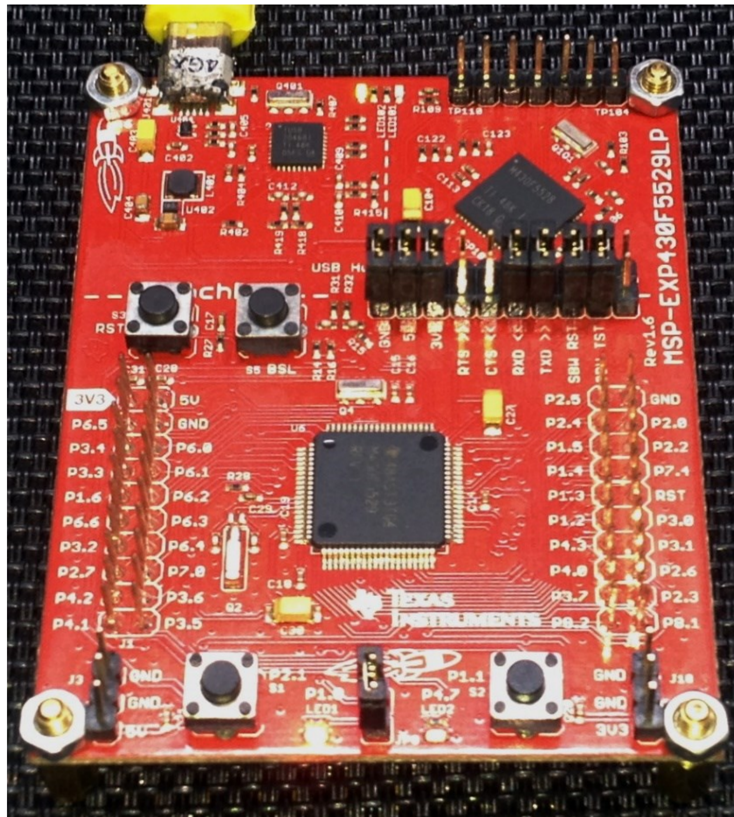


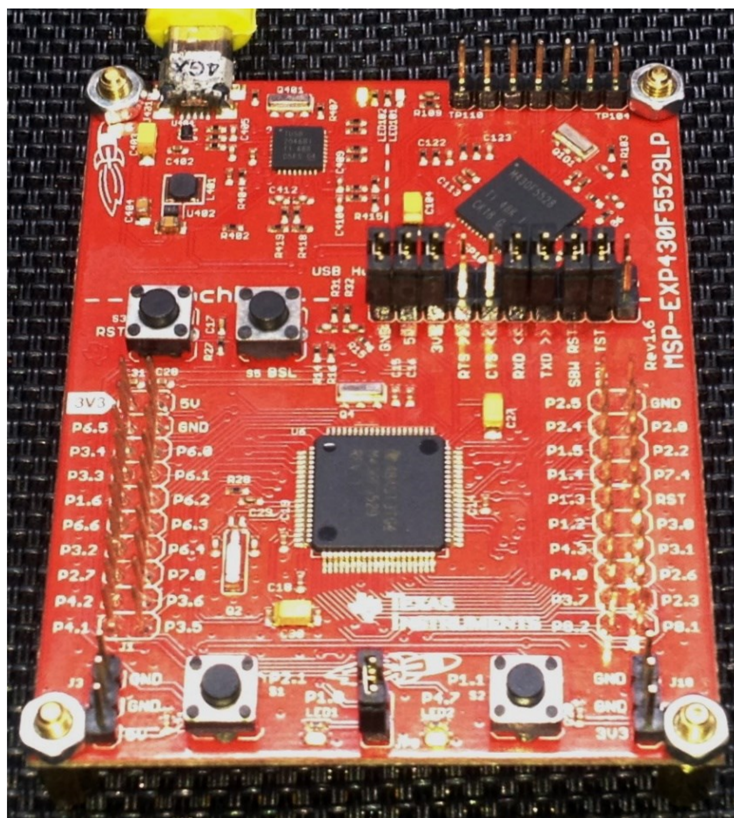
From the graph, we can see that higher the system clock frequency, the higher is the required supply voltage, i.e. core power. By default, the core power mode is 0 and, in this mode, the maximum system clock frequency is 8MHz. My example code goes to maximum operating frequency of 25MHz at the ending and so core power mode 3 is applied before setting the FLL.

```
PMM_setVCore(PMM_CORE_LEVEL_3);
```

The rest of the clocks (ACLK and SMCLK) are set after setting the MCLK.

Demo





Demo video: https://youtu.be/o32kFY6SymA

Alphanumerical LCD

Alphanumerical LCDs are probably the easiest devices to quickly project information visually. They are easy to use and cheap. To be able to use them, we do not need any additional hardware feature of a micro or any special communication hardware like I2C and SPI. Digital I/Os are what we need to use these displays. Just like my other tutorials, this tutorial includes three LCD examples – one with software SPI communication, one with software I2C communication and one with direct DIO-based 4-bit LCD interface.

Be sure to use 3.3V compatible LCD because most of them don't operate at 3.3V power level. Most of the LCD are designed for 5V operation. It is okay to use 5V power supply for powering LCD if there is no other option. LCDs accept 0V - 3.3V logic level. However, there should be no way with which a 5V power supply/device can get connected with MSP430's 3.3V power bus.

Code Example

lcd.h

```
#include "driverlib.h"
#include "delay.h"


#define LCD_RS_PORT             GPIO_PORT_P8
#define LCD_RS_PIN              GPIO_PIN1

#define LCD_EN_PORT             GPIO_PORT_P8
#define LCD_EN_PIN              GPIO_PIN2

#define LCD_D4_PORT             GPIO_PORT_P2
#define LCD_D4_PIN              GPIO_PIN3

#define LCD_D5_PORT             GPIO_PORT_P3
#define LCD_D5_PIN              GPIO_PIN7

#define LCD_D6_PORT             GPIO_PORT_P2
#define LCD_D6_PIN              GPIO_PIN6

#define LCD_D7_PORT             GPIO_PORT_P4
#define LCD_D7_PIN              GPIO_PIN0


#define LCD_RS_HIGH             GPIO_setOutputHighOnPin(LCD_RS_PORT, LCD_RS_PIN)
#define LCD_RS_LOW              GPIO_setOutputLowOnPin(LCD_RS_PORT, LCD_RS_PIN)

#define LCD_EN_HIGH             GPIO_setOutputHighOnPin(LCD_EN_PORT, LCD_EN_PIN)
#define LCD_EN_LOW              GPIO_setOutputLowOnPin(LCD_EN_PORT, LCD_EN_PIN)

#define LCD_DB4_HIGH            GPIO_setOutputHighOnPin(LCD_D4_PORT, LCD_D4_PIN)
#define LCD_DB4_LOW             GPIO_setOutputLowOnPin(LCD_D4_PORT, LCD_D4_PIN)

#define LCD_DB5_HIGH            GPIO_setOutputHighOnPin(LCD_D5_PORT, LCD_D5_PIN)
#define LCD_DB5_LOW             GPIO_setOutputLowOnPin(LCD_D5_PORT, LCD_D5_PIN)

#define LCD_DB6_HIGH            GPIO_setOutputHighOnPin(LCD_D6_PORT, LCD_D6_PIN)
#define LCD_DB6_LOW             GPIO_setOutputLowOnPin(LCD_D6_PORT, LCD_D6_PIN)

#define LCD_DB7_HIGH            GPIO_setOutputHighOnPin(LCD_D7_PORT, LCD_D7_PIN)
#define LCD_DB7_LOW             GPIO_setOutputLowOnPin(LCD_D7_PORT, LCD_D7_PIN)

#define clear_display           0x01
#define goto_home               0x02

#define cursor_direction_inc    (0x04 | 0x02)
#define cursor_direction_dec    (0x04 | 0x00)
#define display_shift           (0x04 | 0x01)
#define display_no_shift        (0x04 | 0x00)

#define display_on              (0x08 | 0x04)
#define display_off             (0x08 | 0x02)
#define cursor_on               (0x08 | 0x02)
#define cursor_off              (0x08 | 0x00)
#define blink_on                (0x08 | 0x01)
#define blink_off               (0x08 | 0x00)

#define _8_pin_interface        (0x20 | 0x10)
#define _4_pin_interface        (0x20 | 0x00)
#define _2_row_display          (0x20 | 0x08)
#define _1_row_display          (0x20 | 0x00)
```

```c
#define _5x10_dots              (0x20 | 0x40)
#define _5x7_dots               (0x20 | 0x00)

#define DAT                     1
#define CMD                     0

#define dly                     2


void LCD_DIO_init(void);
void LCD_init(void);
void LCD_send(unsigned char value, unsigned char mode);
void LCD_4bit_send(unsigned char lcd_data);
void LCD_putstr(char *lcd_string);
void LCD_putchar(char char_data);
void LCD_clear_home(void);
void LCD_goto(unsigned char x_pos, unsigned char y_pos);
void toggle_EN_pin(void);
void toggle_io(unsigned char lcd_data, unsigned char bit_pos, unsigned int PORT, unsigned i
nt PIN);
```

lcd.c

```c
#include "lcd.h"


void LCD_DIO_init(void)
{
    GPIO_setAsOutputPin(LCD_RS_PORT, LCD_RS_PIN);
    GPIO_setAsOutputPin(LCD_EN_PORT, LCD_EN_PIN);
    GPIO_setAsOutputPin(LCD_D4_PORT, LCD_D4_PIN);
    GPIO_setAsOutputPin(LCD_D5_PORT, LCD_D5_PIN);
    GPIO_setAsOutputPin(LCD_D6_PORT, LCD_D6_PIN);
    GPIO_setAsOutputPin(LCD_D7_PORT, LCD_D7_PIN);

    _delay_cycles(100);
}


void LCD_init(void)
{
    LCD_DIO_init();

    LCD_RS_LOW;
    delay_ms(dly);
    toggle_EN_pin();

    LCD_send(0x33, CMD);
    LCD_send(0x32, CMD);

    LCD_send((_4_pin_interface | _2_row_display | _5x7_dots), CMD);
    LCD_send((display_on | cursor_off | blink_off), CMD);
    LCD_send((clear_display), CMD);
    LCD_send((cursor_direction_inc | display_no_shift), CMD);
}

void LCD_send(unsigned char value, unsigned char type)

{
    switch(type)
    {
      case DAT:
      {
          LCD_RS_HIGH;
```

```
            break;
        }
        default:
        {
            LCD_RS_LOW;
            break;
        }
    }

    LCD_4bit_send(value);
}


void LCD_4bit_send(unsigned char lcd_data)
{
    toggle_io(lcd_data, 7, LCD_D7_PORT, LCD_D7_PIN);
    toggle_io(lcd_data, 6, LCD_D6_PORT, LCD_D6_PIN);
    toggle_io(lcd_data, 5, LCD_D5_PORT, LCD_D5_PIN);
    toggle_io(lcd_data, 4, LCD_D4_PORT, LCD_D4_PIN);

    toggle_EN_pin();

    toggle_io(lcd_data, 3, LCD_D7_PORT, LCD_D7_PIN);
    toggle_io(lcd_data, 2, LCD_D6_PORT, LCD_D6_PIN);
    toggle_io(lcd_data, 1, LCD_D5_PORT, LCD_D5_PIN);
    toggle_io(lcd_data, 0, LCD_D4_PORT, LCD_D4_PIN);

    toggle_EN_pin();
}


void LCD_putstr(char *lcd_string)
{
    do
    {
        LCD_send(*lcd_string++, DAT);
    }while(*lcd_string != '\0');
}


void LCD_putchar(char char_data)
{
    LCD_send(char_data, DAT);
}


void LCD_clear_home(void)
{
    LCD_send(clear_display, CMD);
    LCD_send(goto_home, CMD);
}


void LCD_goto(unsigned char x_pos, unsigned char y_pos)
{
    if(y_pos == 0)
    {
        LCD_send((0x80 | x_pos), CMD);
    }
    else
    {
        LCD_send((0x80 | 0x40 | x_pos), CMD);
    }
}
```

```c
void toggle_EN_pin(void)
{
    LCD_EN_HIGH;
    delay_ms(dly);
    LCD_EN_LOW;
    delay_ms(dly);
}


void toggle_io(unsigned char lcd_data, unsigned char bit_pos, unsigned int PORT, unsigned i
nt PIN)
{
    unsigned char temp = 0x00;

    temp = (0x01 & (lcd_data >> bit_pos));

    switch(temp)
    {
        case 0:
        {
            GPIO_setOutputLowOnPin(PORT, PIN);
            break;
        }

        default:
        {
            GPIO_setOutputHighOnPin(PORT, PIN);
            break;
        }
    }
}
```

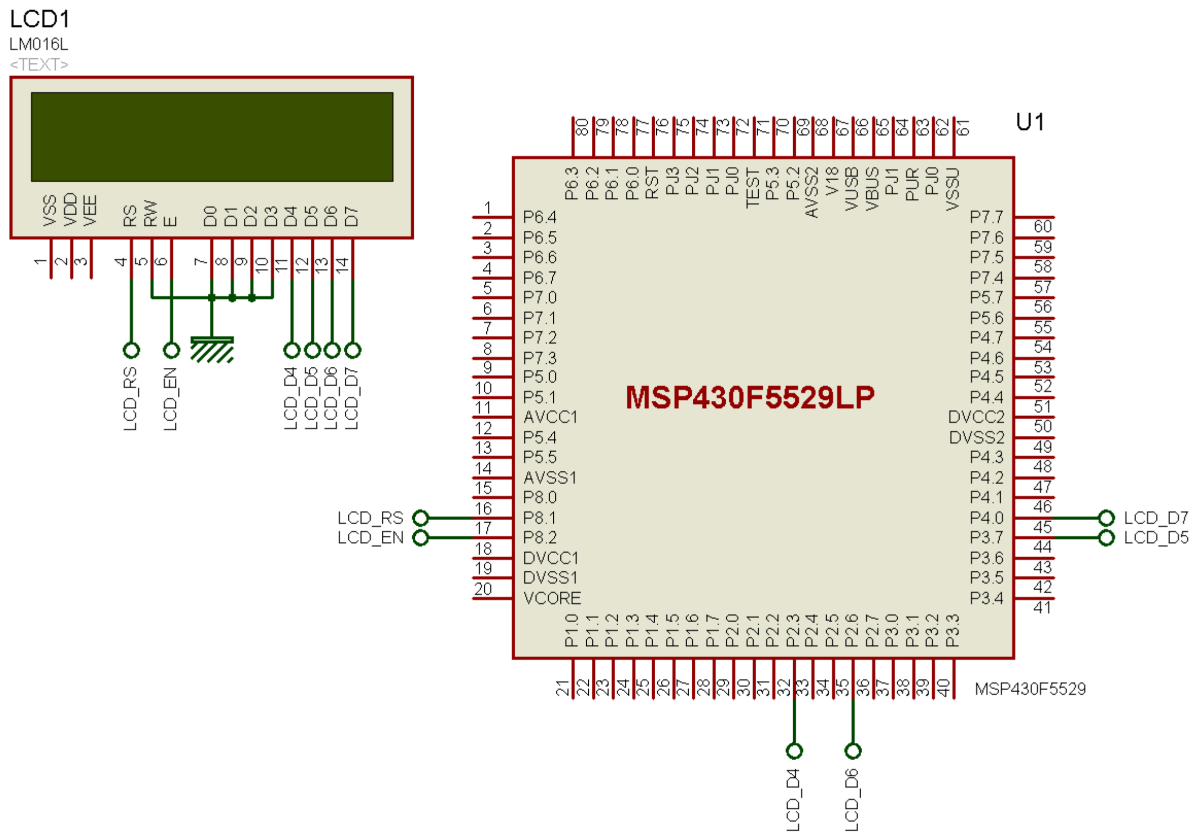
main.c

```c
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"


void clock_init(void);
void show_value(unsigned char value);


void main(void)
{
    unsigned char s = 0x00;

    char txt1[] = {"MICROARENA"};
    char txt2[] = {"SShahryiar"};
    char txt3[] = {"MSP430F5529LP"};
    char txt4[] = {"Launchpad!"};

    WDT_A_hold(WDT_A_BASE);

    clock_init();

    LCD_init();

    LCD_clear_home();

    LCD_goto(3, 0);
    LCD_putstr(txt1);
    LCD_goto(3, 1);
    LCD_putstr(txt2);
```

```
    delay_ms(4000);

    LCD_clear_home();

    for(s = 0; s < 13; s++)
    {
        LCD_goto((1 + s), 0);
        LCD_putchar(txt3[s]);
        delay_ms(60);
    }
    for(s = 0; s < 10; s++)
    {
        LCD_goto((3 + s), 1);
        LCD_putchar(txt4[s]);
        delay_ms(60);
    }
    delay_ms(4000);

    s = 0;
    LCD_clear_home();

    LCD_goto(3, 0);
    LCD_putstr(txt1);

    while(1)
    {
        show_value(s);
        s++;
        delay_ms(400);
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_REFOCLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}
```

```
void show_value(unsigned char value)
{
   unsigned char ch = 0x00;

   ch = ((value / 100) + 0x30);
   LCD_goto(6, 1);
   LCD_putchar(ch);

   ch = (((value / 10) % 10) + 0x30);
   LCD_goto(7, 1);
   LCD_putchar(ch);

   ch = ((value % 10) + 0x30);
   LCD_goto(8, 1);
   LCD_putchar(ch);
}
```

Hardware Setup



Explanation

Since alphanumerical displays utilize GPIOs, there's hardly anything to explain here. Try not to use GPIOs that have alternate roles because you may never know when would you need their alternative functions.

Most popularly alphanumerical LCDs are driven in 4-bit data mode and this is what that has been done here. First let's see some typical LCD instructions.

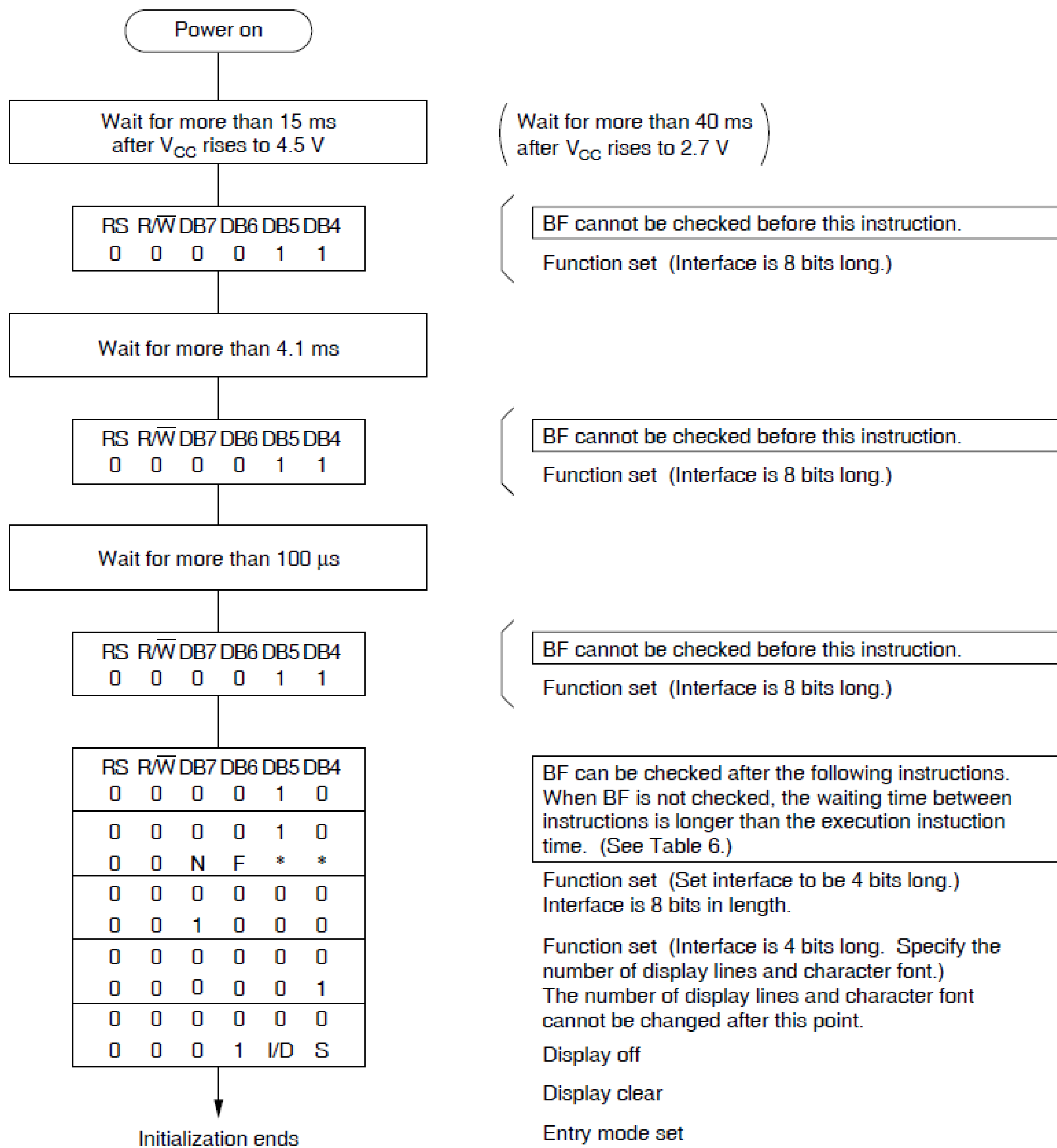
No.	Instructions	Code										Function
		RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
1	Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20h" to DDRAM and set DDRAM address (AC) to "00h"
2	Return Home	0	0	0	0	0	0	0	0	1	x	Set DDRAM address (AC) to "00h" and return cursor to its original position if shifted (DDRAM contents are not change)
3	Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Set cursor moving direction and specify display shift, during data read and write of DDRAM and CGRAM. S=1, screen shifting; S=0, no screen shifting I/D=1, AC=AC+1 and if S=1, screen shift left I/D=0, AC=AC-1 and if S=0, screen shift right
4	Display ON/OFF	0	0	0	0	0	0	1	D	C	B	D=1, display on; D=0, display off C=1, cursor on; C=0, cursor off B=1, cursor blinking on; B=0, cursor blinking off
5	Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	x	x	Move the cursor or shift the display, where DDRAM contents. S/C=1, shift screen; S/C=0, shift cursor R/L=1, to right-side; R/L=0, to left side (if S/C=1, AC will not be changed)
6	Function Set	0	0	0	0	1	DL	N	F	x	x	DL=1, 8-bit interface; DL=0, 4-bit interface N=1, 2-line display; N=0, 1-line display F=1, 5x11 dots font; F=0, 5x8 dots font
7	Set CGRAM address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter
8	Set DDRAM address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter
9	Read Busy flag & address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Check the system status and get the address counter content (AC6~AC0). BF=1, busy; BF=0, ready
10	Write data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write the data into internal RAM, where the address counter pointing at.
11	Read data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read the data from internal RAM, where the address counter pointing at.

Note that these instructions are sent from host micro to LCD module by changing GPIO states only. Only the following GPIO pins are used.

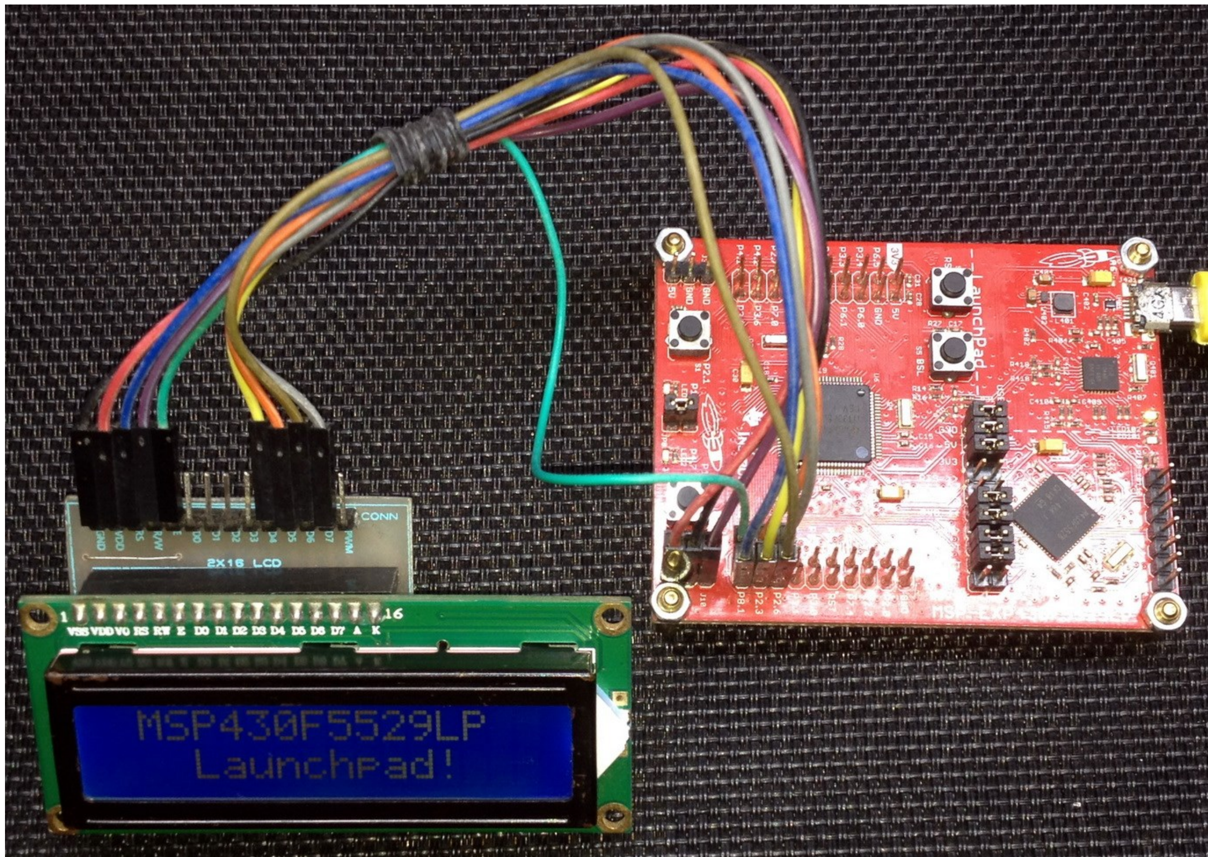
```
#define LCD_RS_PORT            GPIO_PORT_P8
#define LCD_RS_PIN             GPIO_PIN1

#define LCD_EN_PORT            GPIO_PORT_P8
#define LCD_EN_PIN             GPIO_PIN2

#define LCD_D4_PORT            GPIO_PORT_P2
#define LCD_D4_PIN             GPIO_PIN3

#define LCD_D5_PORT            GPIO_PORT_P3
#define LCD_D5_PIN             GPIO_PIN7

#define LCD_D6_PORT            GPIO_PORT_P2
#define LCD_D6_PIN             GPIO_PIN6

#define LCD_D7_PORT            GPIO_PORT_P4
#define LCD_D7_PIN             GPIO_PIN0
```

This declaration in the LCD header file is very important as it states GPIO pin purposes.

Initialization of the LCD is done by going through the steps in the following program flow graph and manipulating the LCD pins accordingly.



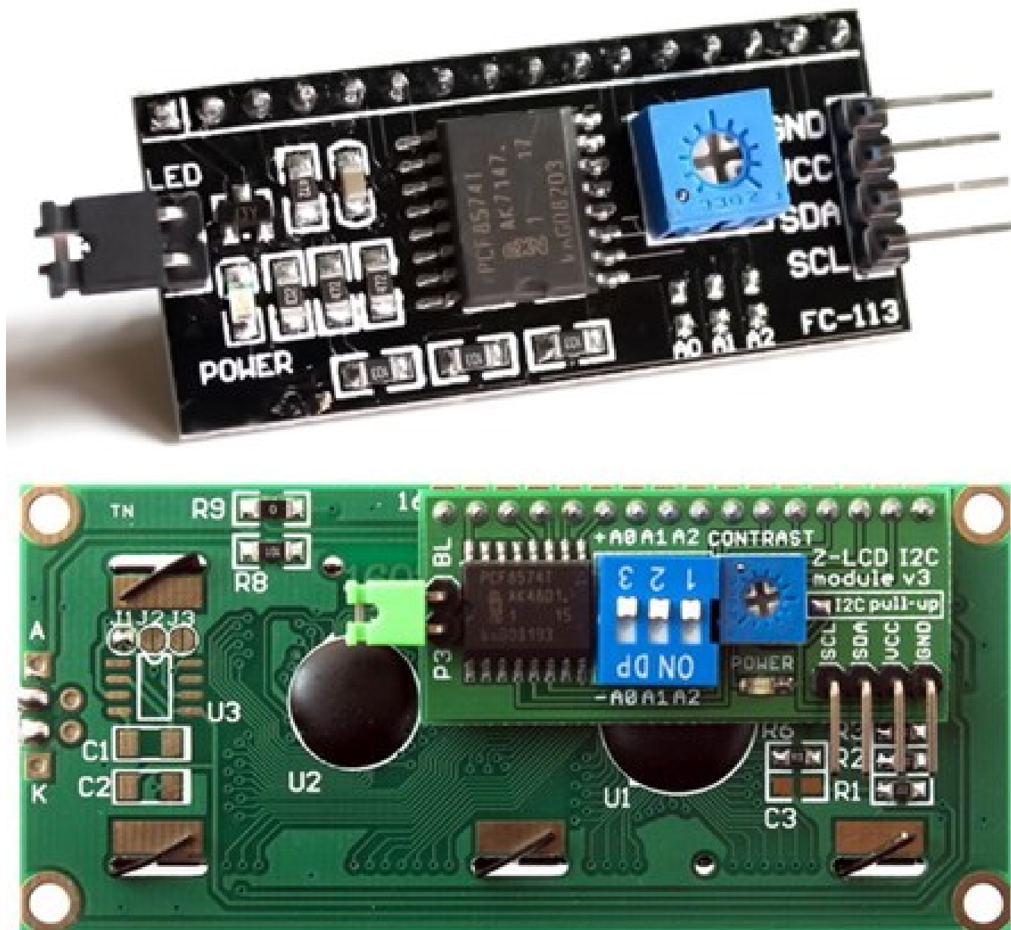
Demo



Demo video: https://youtu.be/eydLbB-tdIc

2 Wire (Software I2C) LCD

Alphanumerical LCDs need lot of pins (6 pins at least) for interfacing and in some cases, this is an expensive requirement. In such cases, hardware designs become complex and every pin is precious for their secondary roles. We can avoid this by using software-based I2C or SPI LCD drivers.



We can use I2C-based port expanders like PCF8574 or MCP23017 for this purpose.

Code Example

SW_I2C.h

```
#include "driverlib.h"
#include "delay.h"


#define SW_I2C_SDA_PORT GPIO_PORT_P8
#define SW_I2C_SCL_PORT GPIO_PORT_P8

#define SDA_pin         GPIO_PIN1
#define SCL_pin         GPIO_PIN2

#define SDA_DIR_OUT()   GPIO_setAsOutputPin(SW_I2C_SDA_PORT, SDA_pin)
#define SDA_DIR_IN()    GPIO_setAsInputPin(SW_I2C_SDA_PORT, SDA_pin)
#define SCL_DIR_OUT()   GPIO_setAsOutputPin(SW_I2C_SCL_PORT, SCL_pin)
#define SCL_DIR_IN()    GPIO_setAsInputPin(SW_I2C_SCL_PORT, SCL_pin)

#define SDA_HIGH()      GPIO_setOutputHighOnPin(SW_I2C_SDA_PORT, SDA_pin)
#define SDA_LOW()       GPIO_setOutputLowOnPin(SW_I2C_SDA_PORT, SDA_pin)
#define SCL_HIGH()      GPIO_setOutputHighOnPin(SW_I2C_SCL_PORT, SCL_pin)
#define SCL_LOW()       GPIO_setOutputLowOnPin(SW_I2C_SCL_PORT, SCL_pin)

#define SDA_IN()        GPIO_getInputPinValue(SW_I2C_SDA_PORT, SDA_pin)

#define I2C_ACK         0xFF
#define I2C_NACK        0x00

#define I2C_timeout     1000


void SW_I2C_init(void);
void SW_I2C_start(void);
void SW_I2C_stop(void);
unsigned char SW_I2C_read(unsigned char ack);
void SW_I2C_write(unsigned char value);
void SW_I2C_ACK_NACK(unsigned char mode);
unsigned char SW_I2C_wait_ACK(void);
```

SW_I2C.c

```
#include "SW_I2C.h"


void SW_I2C_init(void)
{
    SDA_DIR_OUT();
    SCL_DIR_OUT();
    delay_ms(1);
    SDA_HIGH();
    SCL_HIGH();
}


void SW_I2C_start(void)
{
    SDA_DIR_OUT();
    SDA_HIGH();
    SCL_HIGH();
    delay_us(4);
    SDA_LOW();
```

```
    delay_us(4);
    SCL_LOW();
}


void SW_I2C_stop(void)
{
    SDA_DIR_OUT();
    SDA_LOW();
    SCL_LOW();
    delay_us(4);
    SDA_HIGH();
    SCL_HIGH();
    delay_us(4);
}


unsigned char SW_I2C_read(unsigned char ack)
{
    unsigned char i = 8;
    unsigned char j = 0;

    SDA_DIR_IN();

    while(i > 0)
    {
        SCL_LOW();
        delay_us(2);
        SCL_HIGH();
        delay_us(2);
        j <<= 1;

        if(SDA_IN() != 0x00)
        {
            j++;
        }

        delay_us(1);
        i--;
    };

    switch(ack)
    {
        case I2C_ACK:
        {
            SW_I2C_ACK_NACK(I2C_ACK);;
            break;
        }
        default:
        {
            SW_I2C_ACK_NACK(I2C_NACK);;
            break;
        }
    }

    return j;
}


void SW_I2C_write(unsigned char value)
{
    unsigned char i = 8;

    SDA_DIR_OUT();
    SCL_LOW();
```

```
    while(i > 0)
    {
        if(((value & 0x80) >> 7) != 0x00)
        {
            SDA_HIGH();
        }
        else
        {
            SDA_LOW();
        }


        value <<= 1;
        delay_us(2);
        SCL_HIGH();
        delay_us(2);
        SCL_LOW();
        delay_us(2);
        i--;
    };
}


void SW_I2C_ACK_NACK(unsigned char mode)
{
    SCL_LOW();
    SDA_DIR_OUT();

    switch(mode)
    {
        case I2C_ACK:
        {
            SDA_LOW();
            break;
        }
        default:
        {
            SDA_HIGH();
            break;
        }
    }

    delay_us(2);
    SCL_HIGH();
    delay_us(2);
    SCL_LOW();
}


unsigned char SW_I2C_wait_ACK(void)
{
    signed int timeout = 0;

    SDA_DIR_IN();

    SDA_HIGH();
    delay_us(1);
    SCL_HIGH();
    delay_us(1);

    while(SDA_IN() != 0x00)
    {
        timeout++;
```

```
        if(timeout > I2C_timeout)
        {
            SW_I2C_stop();
            return 1;
        }
    };

    SCL_LOW();
    return 0;
}
```

PCF8574.h

```
#include "SW_I2C.h"


#define PCF8574_address                  0x4E

#define PCF8574_write_cmd                PCF8574_address
#define PCF8574_read_cmd                 (PCF8574_address + 1)


void PCF8574_init(void);
unsigned char PCF8574_read(void);
void PCF8574_write(unsigned char data_byte);
```

PCF8574.c

```
#include "PCF8574.h"


void PCF8574_init(void)
{
    SW_I2C_init();
    delay_ms(100);
}


unsigned char PCF8574_read(void)
{
    unsigned char port_byte = 0;

    SW_I2C_start();
    SW_I2C_write(PCF8574_read_cmd);
    port_byte = SW_I2C_read(I2C_NACK);
    SW_I2C_stop();

    return port_byte;
}


void PCF8574_write(unsigned char data_byte)
{
    SW_I2C_start();
    SW_I2C_write(PCF8574_write_cmd);
    SW_I2C_ACK_NACK(I2C_ACK);
    SW_I2C_write(data_byte);
    SW_I2C_ACK_NACK(I2C_ACK);
    SW_I2C_stop();
}
```

lcd.h

```
#include "PCF8574.h"


#define clear_display                                0x01
#define goto_home                                    0x02

#define cursor_direction_inc                         (0x04 | 0x02)
#define cursor_direction_dec                         (0x04 | 0x00)
#define display_shift                                (0x04 | 0x01)
#define display_no_shift                             (0x04 | 0x00)

#define display_on                                   (0x08 | 0x04)
#define display_off                                  (0x08 | 0x02)
#define cursor_on                                    (0x08 | 0x02)
#define cursor_off                                   (0x08 | 0x00)
#define blink_on                                     (0x08 | 0x01)
#define blink_off                                    (0x08 | 0x00)

#define _8_pin_interface                             (0x20 | 0x10)
#define _4_pin_interface                             (0x20 | 0x00)
#define _2_row_display                               (0x20 | 0x08)
#define _1_row_display                               (0x20 | 0x00)
#define _5x10_dots                                   (0x20 | 0x40)
#define _5x7_dots                                    (0x20 | 0x00)

#define BL_ON                                        1
#define BL_OFF                                       0

#define dly                                          2

#define DAT                                          1
#define CMD                                          0


void LCD_init(void);
void LCD_toggle_EN(void);
void LCD_send(unsigned char value, unsigned char mode);
void LCD_4bit_send(unsigned char lcd_data);
void LCD_putstr(char *lcd_string);
void LCD_putchar(char char_data);
void LCD_clear_home(void);
void LCD_goto(unsigned char x_pos, unsigned char y_pos);
```

lcd.c

```
#include "lcd.h"


static unsigned char bl_state;
static unsigned char data_value;


void LCD_init(void)
{
  PCF8574_init();
  delay_ms(100);

  bl_state = BL_ON;
  data_value = 0x04;
  PCF8574_write(data_value);
```

```
  delay_ms(10);

  LCD_send(0x33, CMD);
  LCD_send(0x32, CMD);

  LCD_send((_4_pin_interface | _2_row_display | _5x7_dots), CMD);
  LCD_send((display_on | cursor_off | blink_off), CMD);
  LCD_send((clear_display), CMD);
  LCD_send((cursor_direction_inc | display_no_shift), CMD);
}


void LCD_toggle_EN(void)
{
  data_value |= 0x04;
  PCF8574_write(data_value);
  delay_ms(dly);
  data_value &= 0xF9;
  PCF8574_write(data_value);
  delay_ms(dly);
}


void LCD_send(unsigned char value, unsigned char mode)
{
  switch(mode)
  {
     case CMD:
     {
        data_value &= 0xF4;
        break;
     }
     case DAT:
     {
        data_value |= 0x01;
        break;
     }
  }

  switch(bl_state)
  {
     case BL_ON:
     {
        data_value |= 0x08;
        break;
     }
     case BL_OFF:
     {
        data_value &= 0xF7;
        break;
     }
  }

  PCF8574_write(data_value);
  LCD_4bit_send(value);
  delay_ms(dly);
}


void LCD_4bit_send(unsigned char lcd_data)
{
  unsigned char temp = 0x00;

  temp = (lcd_data & 0xF0);
  data_value &= 0x0F;
  data_value |= temp;
```

```
  PCF8574_write(data_value);
  LCD_toggle_EN();

  temp = (lcd_data & 0x0F);
  temp <<= 0x04;
  data_value &= 0x0F;
  data_value |= temp;
  PCF8574_write(data_value);
  LCD_toggle_EN();
}


void LCD_putstr(char *lcd_string)
{
  do
  {
    LCD_putchar(*lcd_string++);
  }while(*lcd_string != '\0') ;
}


void LCD_putchar(char char_data)
{
  if((char_data >= 0x20) && (char_data <= 0x7F))
  {
    LCD_send(char_data, DAT);
  }
}


void LCD_clear_home(void)
{
  LCD_send(clear_display, CMD);
  LCD_send(goto_home, CMD);
}


void LCD_goto(unsigned char x_pos,unsigned char y_pos)
{
  if(y_pos == 0)
  {
    LCD_send((0x80 | x_pos), CMD);
  }
  else
  {
    LCD_send((0x80 | 0x40 | x_pos), CMD);
  }
}
```

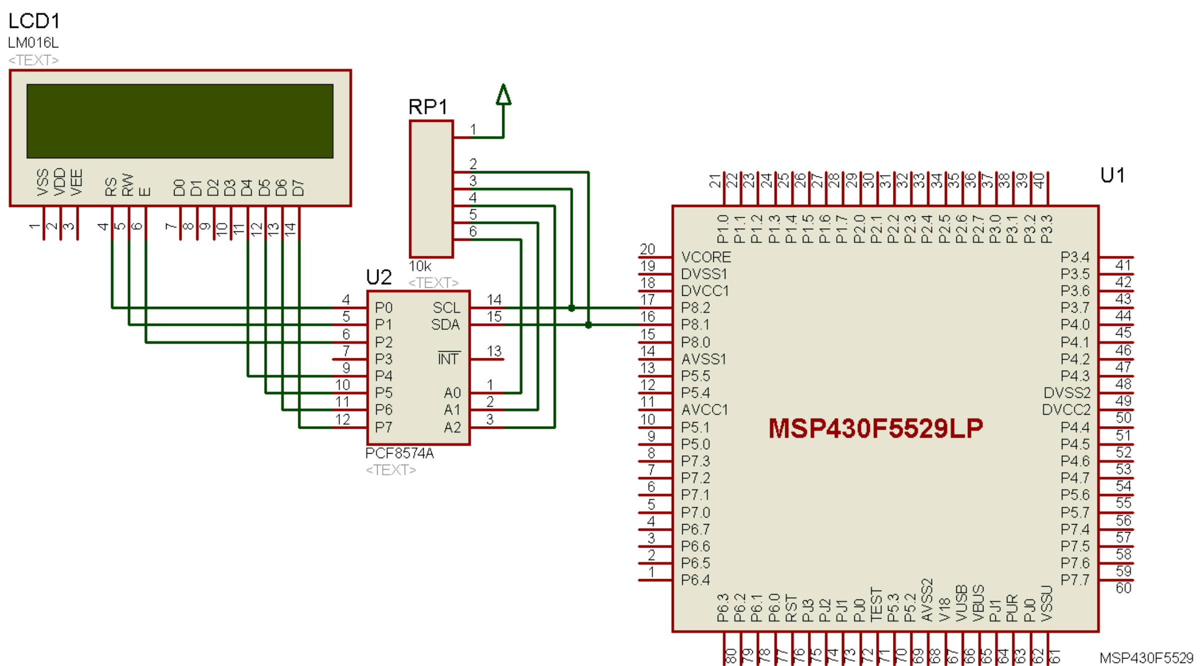
main.c

```
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"


void clock_init(void);
void show_value(unsigned char value);


void main(void)
{
    unsigned char s = 0x00;
```

```
    char txt1[] = {"MICROARENA"};
    char txt2[] = {"SShahryiar"};
    char txt3[] = {"MSP430F5529LP"};
    char txt4[] = {"Launchpad!"};

    WDT_A_hold(WDT_A_BASE);

    clock_init();

    LCD_init();

    LCD_clear_home();

    LCD_goto(3, 0);
    LCD_putstr(txt1);
    LCD_goto(3, 1);
    LCD_putstr(txt2);
    delay_ms(4000);

    LCD_clear_home();

    for(s = 0; s < 13; s++)
    {
        LCD_goto((1 + s), 0);
        LCD_putchar(txt3[s]);
        delay_ms(60);
    }
    for(s = 0; s < 10; s++)
    {
        LCD_goto((3 + s), 1);
        LCD_putchar(txt4[s]);
        delay_ms(60);
    }
    delay_ms(4000);

    s = 0;
    LCD_clear_home();

    LCD_goto(3, 0);
    LCD_putstr(txt1);

    while(1)
    {
        show_value(s);
        s++;
        delay_ms(400);
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
```

```
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_REFOCLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void show_value(unsigned char value)
{
   unsigned char ch = 0x00;

   ch = ((value / 100) + 0x30);
   LCD_goto(6, 1);
   LCD_putchar(ch);

   ch = (((value / 10) % 10) + 0x30);
   LCD_goto(7, 1);
   LCD_putchar(ch);

   ch = ((value % 10) + 0x30);
   LCD_goto(8, 1);
   LCD_putchar(ch);
}
```

Hardware Setup



Explanation

The initialization and working procedure of the LCD module is nothing different from the first LCD example. The only exception is the fact that a PCF8574 port expander is used to manipulate the LCD I/Os instead of direct host micro connection.

PCF8574 from NXP is an I2C-based 8-bit port expander. With just two GPIO pins or I2C pins we can read and write a complete external 8-bit GPIO port. This is what makes PCF8574 a GPIO expander.

Since we are yet to check out the hardware I2C option of MSP430F5529 micro, we will be using software-based I2C. The software I2C driver is coded in *SW_I2C* header and source files. The lines of code there are self-explanatory. All of the I2C operations are simulated by altering two GPIO pin states and using software delays. The important part, however, is the GPIO declarations and it is found in the following part of the *SW_I2C* header file.

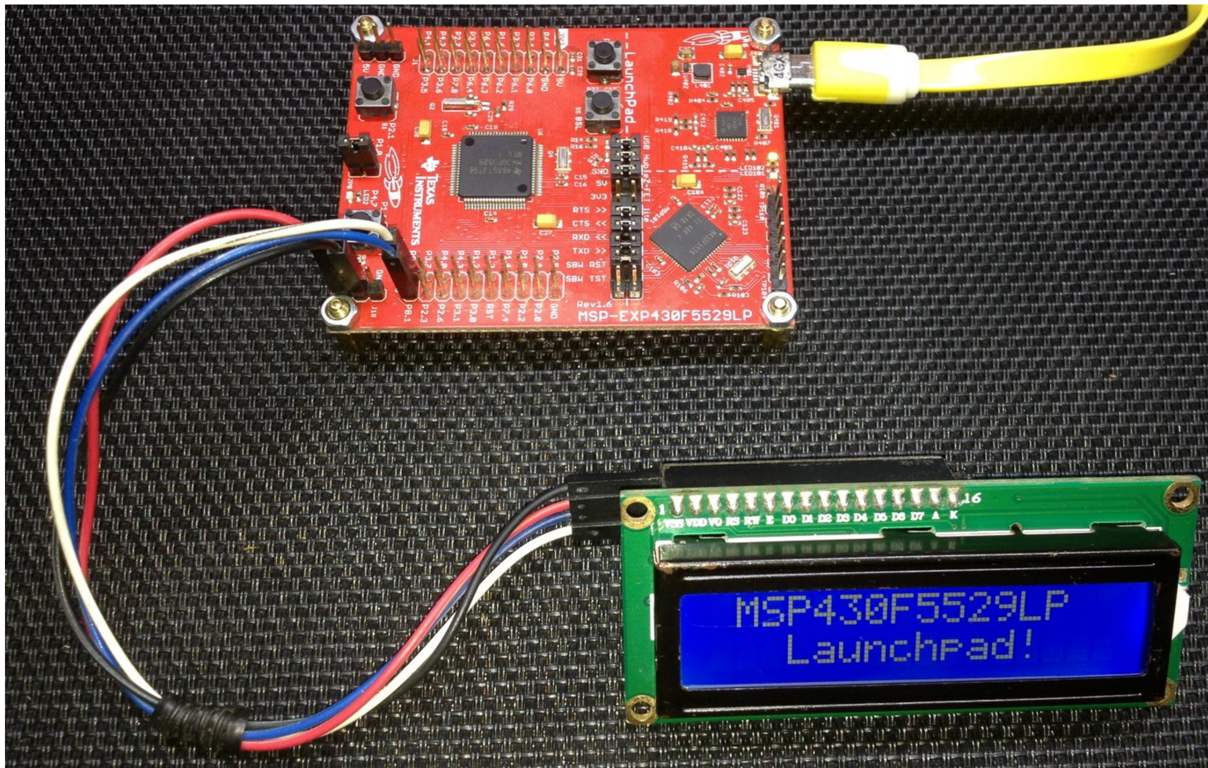
```
#define SW_I2C_SDA_PORT                GPIO_PORT_P8
#define SW_I2C_SCL_PORT                GPIO_PORT_P8

#define SDA_pin                        GPIO_PIN1
#define SCL_pin                        GPIO_PIN2
```

We are not going to read our LCD module and so we don't need *PCF8574_read* function of PCF8574 library. We'll just need the write function.

```
void PCF8574_write(unsigned char data_byte)
{
    SW_I2C_start();
    SW_I2C_write(PCF8574_write_cmd);
    SW_I2C_ACK_NACK(I2C_ACK);
    SW_I2C_write(data_byte);
    SW_I2C_ACK_NACK(I2C_ACK);
    SW_I2C_stop();
}
```

We send the address and command of the PCF8574 module first and then we send the port states. Note that the port states are sent just like the first LCD example.

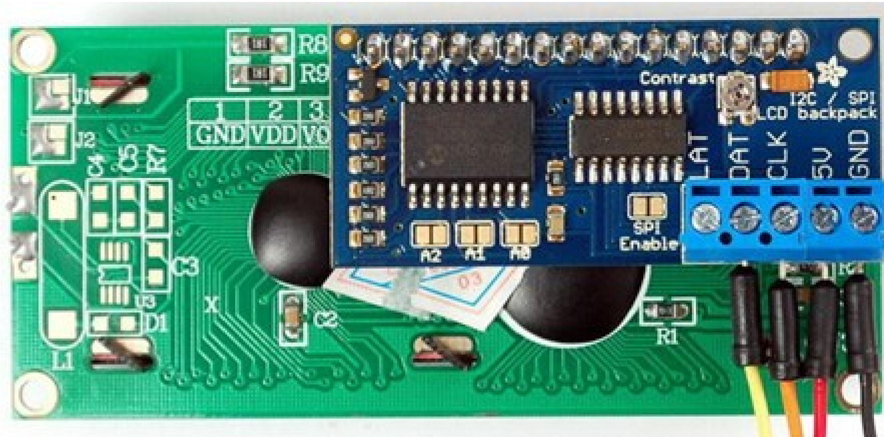
Demo



Demo video: https://youtu.be/jbVioP6Uh30

3 Wire (Software SPI) LCD

Another way of driving alphanumerical LCDs is by using software SPI. Software SPI uses the same concept as in software I2C. The only exceptions here are an additional GPIO pin and the mode of communication. We can use shift registers like CD4094B or 74HC595 for this purpose. Both are equally fine. We can also use dedicated SPI-based port expander like Microchip's MCP23S17.



Software SPI, as in this example, can be used to read/write other SPI-compatible hardware. This is often a requirement when hardware SPI or its pins are inaccessible.

Code Example

lcd.h

```
#include "driverlib.h"
#include "delay.h"


#define SDO_PORT                    GPIO_PORT_P8
#define SCK_PORT                    GPIO_PORT_P8
#define STB_PORT                    GPIO_PORT_P3

#define SDO_pin                     GPIO_PIN1
#define SCK_pin                     GPIO_PIN2
#define STB_pin                     GPIO_PIN7

#define SDO_DIR_OUT                 GPIO_setAsOutputPin(SDO_PORT, SDO_pin)
#define SCK_DIR_OUT                 GPIO_setAsOutputPin(SCK_PORT, SCK_pin)
#define STB_DIR_OUT                 GPIO_setAsOutputPin(STB_PORT, STB_pin)

#define SDO_HIGH                    GPIO_setOutputHighOnPin(SDO_PORT, SDO_pin)
#define SDO_LOW                     GPIO_setOutputLowOnPin(SDO_PORT, SDO_pin)
#define SCK_HIGH                    GPIO_setOutputHighOnPin(SCK_PORT, SCK_pin)
#define SCK_LOW                     GPIO_setOutputLowOnPin(SCK_PORT, SCK_pin)
#define STB_HIGH                    GPIO_setOutputHighOnPin(STB_PORT, STB_pin)
#define STB_LOW                     GPIO_setOutputLowOnPin(STB_PORT, STB_pin)

#define clear_display               0x01
#define goto_home                   0x02

#define cursor_direction_inc        (0x04 | 0x02)
#define cursor_direction_dec        (0x04 | 0x00)
```

```
#define display_shift                 (0x04 | 0x01)
#define display_no_shift              (0x04 | 0x00)

#define display_on                    (0x08 | 0x04)
#define display_off                   (0x08 | 0x02)
#define cursor_on                     (0x08 | 0x02)
#define cursor_off                    (0x08 | 0x00)
#define blink_on                      (0x08 | 0x01)
#define blink_off                     (0x08 | 0x00)

#define _8_pin_interface              (0x20 | 0x10)
#define _4_pin_interface              (0x20 | 0x00)
#define _2_row_display                (0x20 | 0x08)
#define _1_row_display                (0x20 | 0x00)
#define _5x10_dots                    (0x20 | 0x40)
#define _5x7_dots                     (0x20 | 0x00)

#define dly                           1


void SIPO(void);
void LCD_init(void);
void LCD_command(unsigned char value);
void LCD_send_data(unsigned char value);
void LCD_4bit_send(unsigned char lcd_data);
void LCD_putstr(char *lcd_string);
void LCD_putchar(char char_data);
void LCD_clear_home(void);
void LCD_goto(unsigned char x_pos, unsigned char y_pos);
```

lcd.c

```
#include "lcd.h"


static unsigned char data_value;


void SIPO(void)
{
    unsigned char bit = 0x00;
    unsigned char clk = 0x08;
    unsigned char temp = 0x00;

    temp = data_value;
    STB_LOW;

    while(clk > 0)
    {
        bit = ((temp & 0x80) >> 0x07);
        bit &= 0x01;

        switch(bit)
        {
            case 0:
            {
                SDO_LOW;
                break;
            }
            default:
            {
                SDO_HIGH;
                break;
            }
```

```
        }

        SCK_HIGH;

        temp <<= 1;
        clk--;

        SCK_LOW;
    }

    STB_HIGH;
}


void LCD_init(void)
{
    SDO_DIR_OUT;
    SCK_DIR_OUT;
    STB_DIR_OUT;

    data_value = 0x08;
    SIPO();
    delay_ms(10);

    data_value = 0x30;
    SIPO();

    data_value |= 0x08;
    SIPO();
    delay_ms(dly);
    data_value &= 0xF7;
    SIPO();
    delay_ms(dly);

    data_value = 0x30;
    SIPO();

    data_value |= 0x08;
    SIPO();
    delay_ms(dly);
    data_value &= 0xF7;
    SIPO();
    delay_ms(dly);

    data_value = 0x30;
    SIPO();

    data_value |= 0x08;
    SIPO();
    delay_ms(dly);
    data_value &= 0xF7;
    SIPO();
    delay_ms(dly);

    data_value = 0x20;
    SIPO();

    data_value |= 0x08;
    SIPO();
    delay_ms(dly);
    data_value &= 0xF7;
    SIPO();
    delay_ms(dly);

    LCD_command(_4_pin_interface | _2_row_display | _5x7_dots);
```

```
    LCD_command(display_on | cursor_off | blink_off);
    LCD_command(clear_display);
    LCD_command(cursor_direction_inc | display_no_shift);
}


void LCD_command(unsigned char value)
{
    data_value &= 0xFB;
    SIPO();
    LCD_4bit_send(value);
}


void LCD_send_data(unsigned char value)
{
    data_value |= 0x04;
    SIPO();
    LCD_4bit_send(value);
}


void LCD_4bit_send(unsigned char lcd_data)
{
    unsigned char temp = 0x00;

    temp = (lcd_data & 0xF0);
    data_value &= 0x0F;
    data_value |= temp;
    SIPO();

    data_value |= 0x08;
    SIPO();
    delay_ms(dly);
    data_value &= 0xF7;
    SIPO();
    delay_ms(dly);

    temp = (lcd_data & 0x0F);
    temp <<= 0x04;
    data_value &= 0x0F;
    data_value |= temp;
    SIPO();

    data_value |= 0x08;
    SIPO();
    delay_ms(dly);
    data_value &= 0xF7;
    SIPO();
    delay_ms(dly);
}


void LCD_putstr(char *lcd_string)
{
    while (*lcd_string != '\0')
    {
        LCD_send_data(*lcd_string);
        lcd_string++;
    };
}


void LCD_putchar(char char_data)
{
```

```
    LCD_send_data(char_data);
}


void LCD_clear_home(void)
{
    LCD_command(clear_display);
    LCD_command(goto_home);
}


void LCD_goto(unsigned char x_pos,unsigned char y_pos)
{
    if(y_pos == 0)
    {
        LCD_command(0x80 | x_pos);
    }
    else
    {
        LCD_command(0x80 | 0x40 | x_pos);
    }
}
```

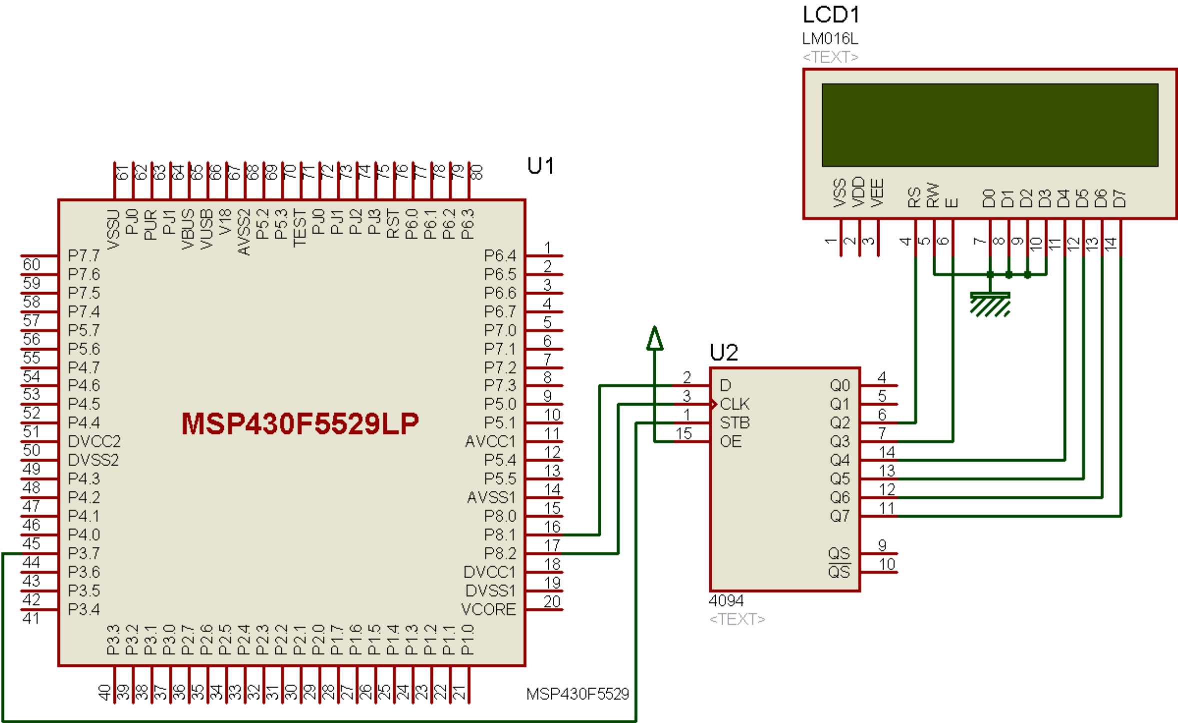
main.c

```
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"


void clock_init(void);
void show_value(unsigned char value);


void main(void)
{
    unsigned char s = 0x00;

    char txt1[] = {"MICROARENA"};
    char txt2[] = {"SShahryiar"};
    char txt3[] = {"MSP430F5529LP"};
    char txt4[] = {"Launchpad!"};

    WDT_A_hold(WDT_A_BASE);

    clock_init();

    LCD_init();

    LCD_clear_home();

    LCD_goto(3, 0);
    LCD_putstr(txt1);
    LCD_goto(3, 1);
    LCD_putstr(txt2);
    delay_ms(4000);

    LCD_clear_home();

    for(s = 0; s < 13; s++)
    {
        LCD_goto((1 + s), 0);
        LCD_putchar(txt3[s]);
        delay_ms(60);
```

```
    }
    for(s = 0; s < 10; s++)
    {
        LCD_goto((3 + s), 1);
        LCD_putchar(txt4[s]);
        delay_ms(60);
    }
    delay_ms(4000);

    s = 0;
    LCD_clear_home();

    LCD_goto(3, 0);
    LCD_putstr(txt1);

    while(1)
    {
        show_value(s);
        s++;
        delay_ms(400);
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_REFOCLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void show_value(unsigned char value)
{
   unsigned char ch = 0x00;

   ch = ((value / 100) + 0x30);
   LCD_goto(6, 1);
```

```
    LCD_putchar(ch);

    ch = (((value / 10) % 10) + 0x30);
    LCD_goto(7, 1);
    LCD_putchar(ch);

    ch = ((value % 10) + 0x30);
    LCD_goto(8, 1);
    LCD_putchar(ch);
}
```

Hardware Setup



Explanation

As with the past example, GPIO pins will be used to emulate software SPI. Thus, we need to declare them first. SDO pin is serial data out pin, SCK is SPI clock pin and lastly, STB is SPI strobe/chip enable (CE)/slave select (SS) pin. Though 74HC595 and CD4094B are functionally same, my favourite is CD4094B due to its wide voltage range. This is what I have used here and the SPI pins are named in its respect.

```
#define SDO_PORT                    GPIO_PORT_P8
#define SCK_PORT                    GPIO_PORT_P8
#define STB_PORT                    GPIO_PORT_P3

#define SDO_pin                     GPIO_PIN1
#define SCK_pin                     GPIO_PIN2
#define STB_pin                     GPIO_PIN7


#define SDO_DIR_OUT                 GPIO_setAsOutputPin(SDO_PORT, SDO_pin)
#define SCK_DIR_OUT                 GPIO_setAsOutputPin(SCK_PORT, SCK_pin)
#define STB_DIR_OUT                 GPIO_setAsOutputPin(STB_PORT, STB_pin)

#define SDO_HIGH                    GPIO_setOutputHighOnPin(SDO_PORT, SDO_pin)
#define SDO_LOW                     GPIO_setOutputLowOnPin(SDO_PORT, SDO_pin)
#define SCK_HIGH                    GPIO_setOutputHighOnPin(SCK_PORT, SCK_pin)
#define SCK_LOW                     GPIO_setOutputLowOnPin(SCK_PORT, SCK_pin)
#define STB_HIGH                    GPIO_setOutputHighOnPin(STB_PORT, STB_pin)
#define STB_LOW                     GPIO_setOutputLowOnPin(STB_PORT, STB_pin)
```

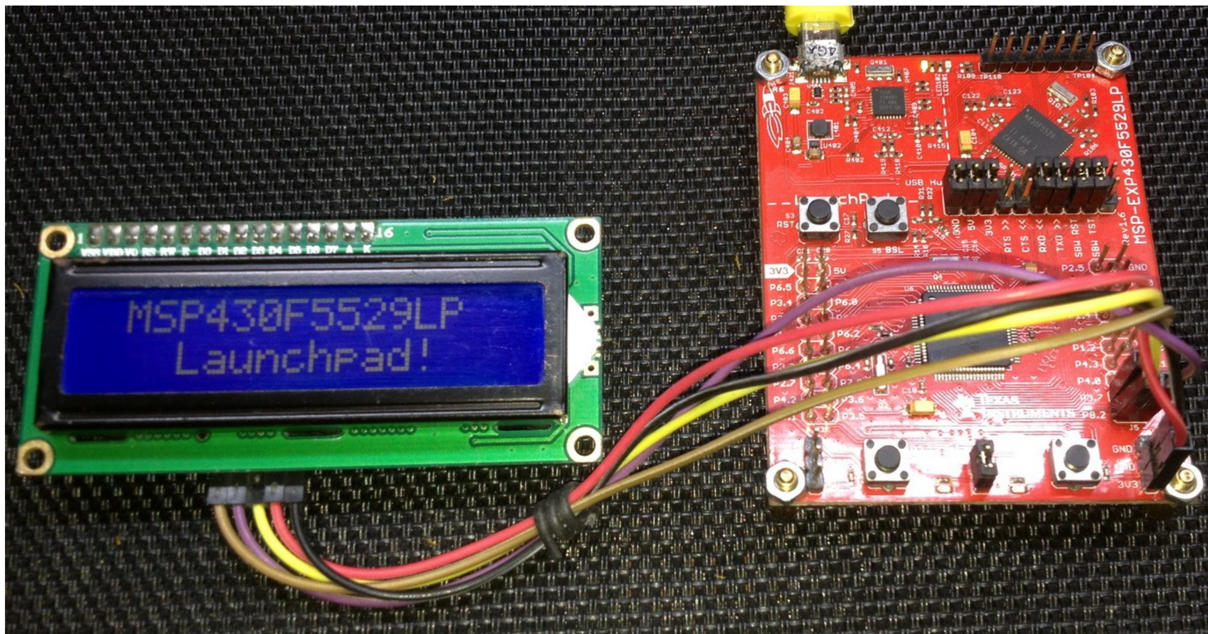
We will not be modifying any function of the LCD itself. It is same as like other LCD examples. This is why we will just be going through the software function responsible for port expansion. Function named *SPIO* is responsible for serially taking data from host micro and providing 8-bit parallel output, hence the name SIPO.

Strobe pin is held low before serially shifting data one bit at a time. Data is shifted on every high to low transition of clock signal for 8 clock transitions, i.e. 8-bit of data is transferred. After shifting out 8 bits of data, the strobe pin is held high and at that moment the 8 output pins of CD4094B shift-register are updated.

```
void SIPO(void)
{
    unsigned char bit = 0x00;
    unsigned char clk = 0x08;
    unsigned char temp = 0x00;

    temp = data_value;
    STB_LOW;

    while(clk > 0)
    {
        bit = ((temp & 0x80) >> 0x07);
        bit &= 0x01;

        switch(bit)
        {
            case 0:
            {
                SDO_LOW;
                break;
```

```
            }
            default:
            {
                SDO_HIGH;
                break;
            }
        }

        SCK_HIGH;

        temp <<= 1;
        clk--;

        SCK_LOW;
    }

    STB_HIGH;
}
```

Demo

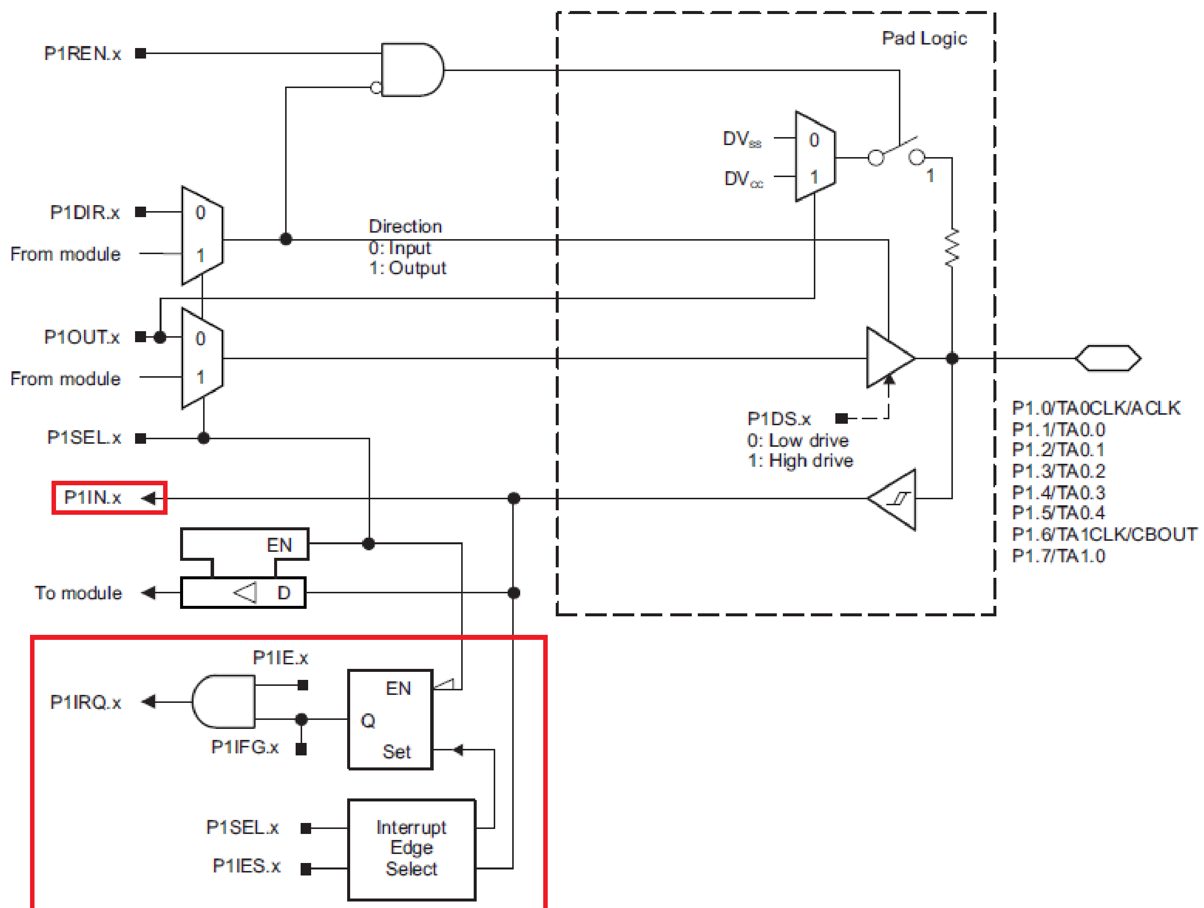


Demo video: https://youtu.be/UPakLxBDb0E

Low Power Modes (LPM) and External Interrupt

At present era, the energy consumption of a device is a very important part of its design. Low power consumption along with optimum performance and long endurance are highly preferable aspects one can think of while crafting a new electronic product. With such in mind, MSP430s were developed with low power consumption feature. In terms of power consumption, MSP430s offer six modes of operation with five being low power or sleep modes of different consumption levels. In TI docs, low power modes are often simply referred as ***LPMs***.

External interrupt is an important DIO input mode feature. It can break the continuity of a regular program flow, i.e. the main routine, and perform certain higher priority secondary tasks separately. Interrupts can be also used to wake up micros from low power sleep modes. Combinations of LPMs and external interrupts have many uses. A good example of such combinations is a wireless mouse. Unless moved or its buttons pressed, the mouse remains in dormant/sleep state to conserve precious battery energy. The internal electronics of the mouse fires up otherwise as if it has been triggered by interrupts caused by motion or button presses. Port A (PA) pins of MSP430F5529 micro are all external interrupt capable pins. Shown below is the structure of a P1 port pin with red regions highlighting external interrupt resources:



Apart from external interrupts, other interrupts like timer interrupts can also wake up a MSP430F5529 from LPM states.

There are 5 LPM modes and an Active Mode of operation. The intent of these modes is to change power consumption by suspending certain hardware and clocks. These are as follows:

• Active mode (AM)

- All clocks are active.

• Low-power mode 0 (LPM0)

- CPU and MCLK are disabled.
- FLL loop control, ACLK and SMCLK remain active.

• Low-power mode 1 (LPM1)

- CPU, MCLK and FLL loop control are disabled.
- ACLK and SMCLK remain active.

• Low-power mode 2 (LPM2)

- CPU, MCLK, FLL loop control and DCOCLK are disabled.
- DC generator of the DCO remains enabled.
- ACLK remains active

• Low-power mode 3 (LPM3)

- CPU, MCLK, FLL loop control, DC generator of the DCO and DCOCLK are disabled.
- ACLK remains active.

• Low-power mode 4 (LPM4)

- CPU, ACLK, MCLK, FLL loop control, DC generator of the DCO and DCOCLK are disabled.
- Crystal oscillator is stopped.
- Complete data retention.

• Low-power mode 4.5 (LPM4.5)

- Internal regulator disabled.
- No data retention.
- Wake-up signal from RST or NMI, P1, and P2.

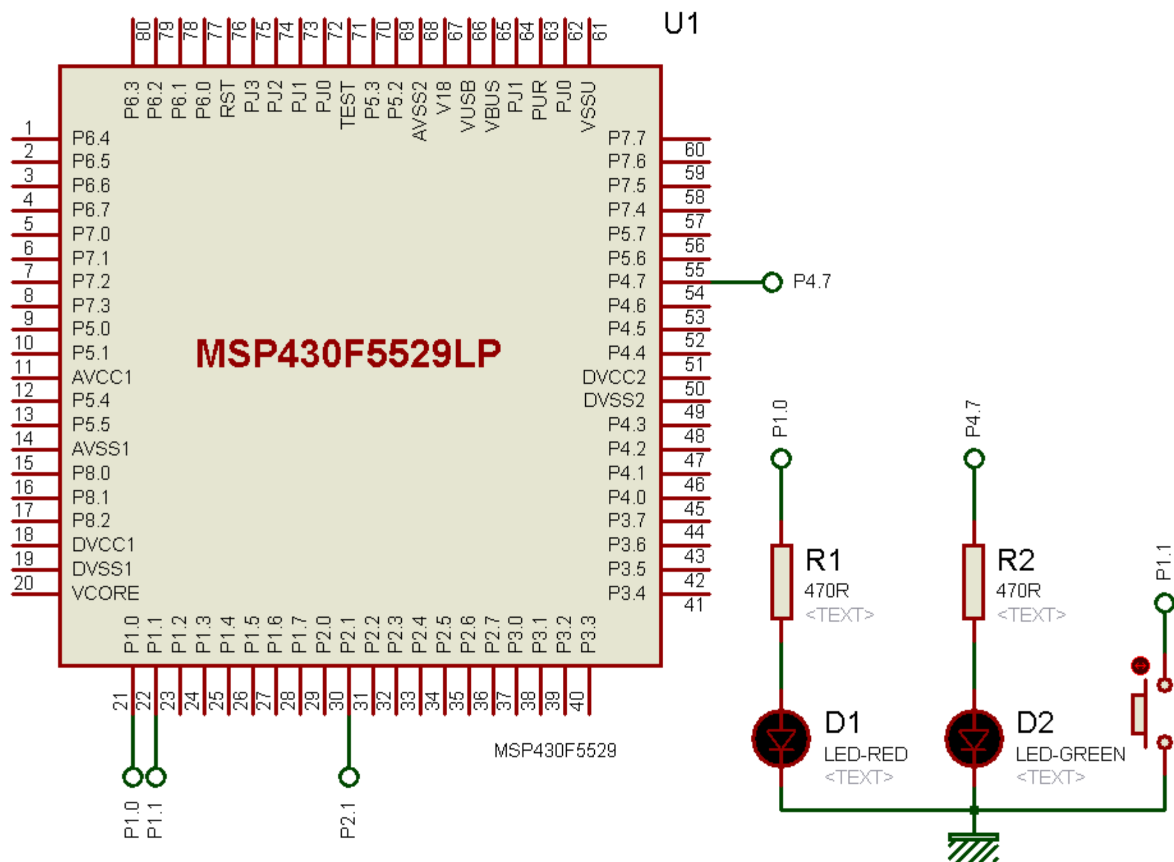
Wake up times from various LPM states also varies but these variations are so tiny in terms of time that they are rarely observed visually.

Code Example

```
#include "driverlib.h"


unsigned char toggle = 0;


void GPIO_init(void);


#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR(void)
{
    LPM3_EXIT;

    if(GPIO_getInterruptStatus(GPIO_PORT_P1,
                               GPIO_PIN1) != 0)
    {
        GPIO_clearInterrupt(GPIO_PORT_P1,
                            GPIO_PIN1);

        toggle = 1;
        _nop();
    }
}


void main(void)
{
    unsigned char i = 0;
    unsigned char j = 0;

    WDT_A_hold(WDT_A_BASE);

    GPIO_init();

    while(1)
    {
        for(j = 0; j < 10; j++)
        {
            GPIO_toggleOutputOnPin(GPIO_PORT_P1,
                                   GPIO_PIN0);

            __delay_cycles(1000000);
        }

        if(toggle)
        {
            for(i = 0; i <= 9; i++)
            {
                GPIO_toggleOutputOnPin(GPIO_PORT_P4,
                                       GPIO_PIN7);

                __delay_cycles(100000);
            }

            toggle = 0;
        }

        LPM3;
    };
}
```

```c
void GPIO_init(void)
{
    GPIO_setAsOutputPin(GPIO_PORT_P1,
                        GPIO_PIN0);

    GPIO_setDriveStrength(GPIO_PORT_P1,
                          GPIO_PIN0,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setDriveStrength(GPIO_PORT_P4,
                          GPIO_PIN7,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1,
                                         GPIO_PIN1);

    GPIO_clearInterrupt(GPIO_PORT_P1,
                        GPIO_PIN1);

    GPIO_selectInterruptEdge(GPIO_PORT_P1,
                             GPIO_PIN1,
                             GPIO_HIGH_TO_LOW_TRANSITION);

    GPIO_enableInterrupt(GPIO_PORT_P1,
                         GPIO_PIN1);

    __enable_interrupt();
}
```

Hardware Setup



Explanation

For this demo, onboard P1.1 button, P1.0 and P4.7 LEDs are used. The LEDs are set as in the GPIO example, i.e. as outputs.

P1.1 button pin is set as input pin with pull-up resistor. This pin is also used as the external interrupt pin. At first, any pending interrupt is cleared. We, then, have to set interrupt edge, i.e. polarity sensitivity. Finally, we enable interrupt on the interrupt pin and enable global interrupt.

```
GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1, GPIO_PIN1);

GPIO_clearInterrupt(GPIO_PORT_P1, GPIO_PIN1);

GPIO_selectInterruptEdge(GPIO_PORT_P1, GPIO_PIN1, GPIO_HIGH_TO_LOW_TRANSITION);

GPIO_enableInterrupt(GPIO_PORT_P1, GPIO_PIN1);

__enable_interrupt();
```

Now the button pin is ready to behave like an external interrupt pin rather than a regular input pin. Remember that P1 belongs to PA port and only PA port pins are interrupt capable.

Let's look at the main loop now. Here, we see that P1.0 LED will be toggled 10 times first. The *if-condition* will not be initially executed since it is linked with the external interrupt. Thus, the conditional statement will be skipped and LPM3 condition will be set.

Once LPM is applied, everything seems to be suspended. P1.0 LED is supposed to keep flashing but that doesn't happen.

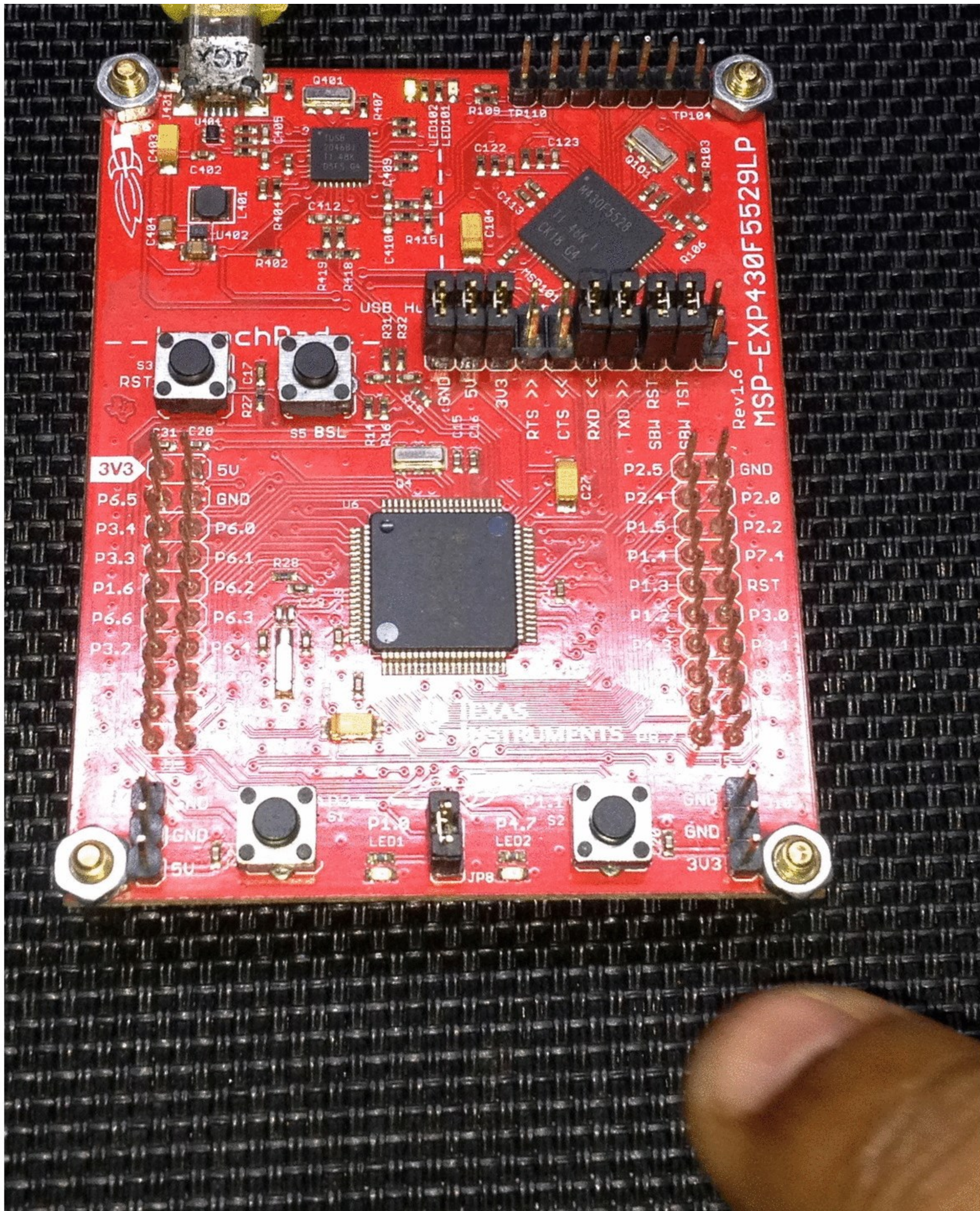
```
for(j = 0; j < 10; j++)
{
    GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);

    __delay_cycles(1000000);
}

if(toggle)
{
    for(i = 0; i <= 9; i++)
    {
        GPIO_toggleOutputOnPin(GPIO_PORT_P4, GPIO_PIN7);

        __delay_cycles(100000);
    }

    toggle = 0;
}

LPM3;
```

When the button is pressed, external interrupt kicks in, LPM3 is exited, interrupt flag is cleared and variable *toggle* is set. Since there is one interrupt vector for the entire P1 port pins, it is needed to be sure which pin caused the interrupt. This is why the state of P1.1 needs to be checked right after interrupt.

```
#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR(void)
{
    LPM3_EXIT;

    if(GPIO_getInterruptStatus(GPIO_PORT_P1, GPIO_PIN1) != 0)
    {
        GPIO_clearInterrupt(GPIO_PORT_P1, GPIO_PIN1);

        toggle = 1;
        _nop();
    }
}
```

Since now the *toggle* variable is set, P1.0 LED flashes and then P4.7 LED flashes, indicating that an external interrupt has been processed.

Demo



Demo video: https://youtu.be/R9Wmk0dMdl4

Internal Flash Memory and Cyclic Redundancy Check (CRC) Module

There are many microcontrollers in today's market that don't come with embedded EEPROM memory. Most ARM microcontrollers, for example, don't have EEPROM memories. EEPROM is typically used for the storage of non-volatile data like configurations, calibrations, etc that are usually not changed frequently and need to be retained even after reset/power failure. EEPROM memories are slow and have lower endurance. Due to these limitations, it is better to use segments of flash memory that will not be used to hold application code. Flash memories have comparatively higher life cycles, high reliability and are faster. The only problem with them is writing. We can't write to individual locations. We have to write/erase a whole segment of multiple byte areas at a time. No matter which memory class we are dealing with, frequent writing/erasing of memory locations also leads to wear and permanent damage. These issues can be avoided by a number of software-based means like wear-levelling. Data reading has no issues as data can be read individually or as a whole. MSP430F5529 has four flash memory areas called information memory (***A*** to ***D***). Information memory is where we can store non-volatile data and these locations have no conflict with application/main memory locations.

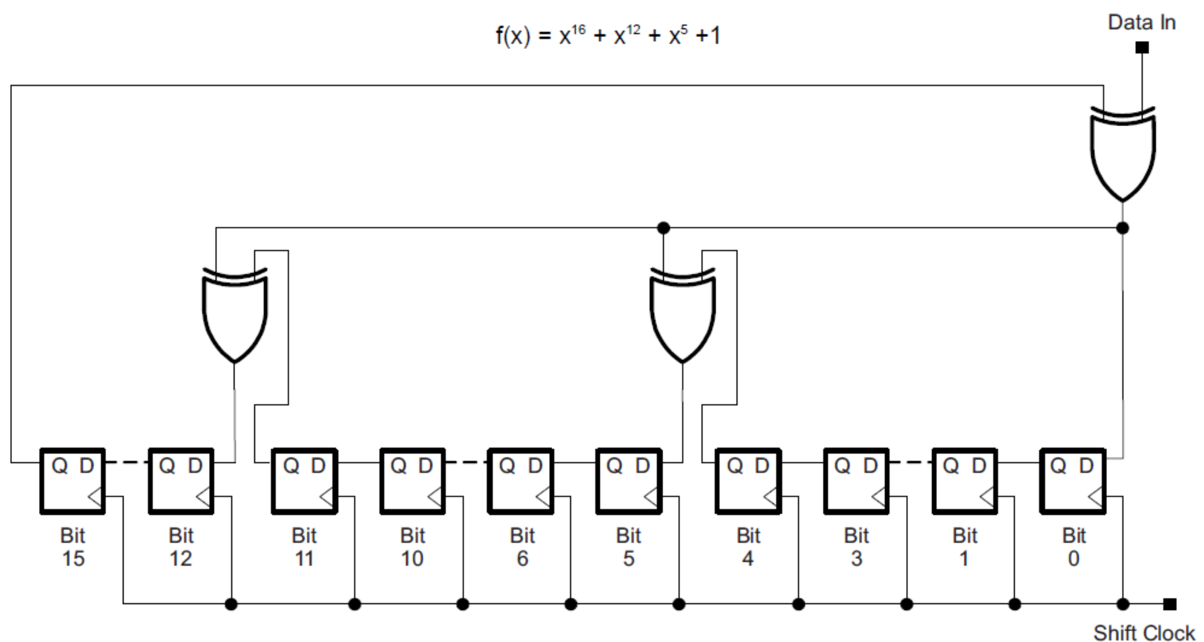
		MSP430F5529 MSP430F5528 MSP430F5519
Memory (flash) Main: interrupt vector	Total Size	128KB 00FFFFh to 00FF80h
Main: code memory	Bank D	32KB 0243FFh to 01C400h
	Bank C	32KB 01C3FFh to 014400h
	Bank B	32KB 0143FFh to 00C400h
	Bank A	32KB 00C3FFh to 004400h
RAM	Sector 3	2KB 0043FFh to 003C00h
	Sector 2	2KB 003BFFh to 003400h
	Sector 1	2KB 0033FFh to 002C00h
	Sector 0	2KB 002BFFh to 002400h
USB RAM	Sector 7	2KB 0023FFh to 001C00h
Information memory (flash)	Info A	128 B 0019FFh to 001980h
	Info B	128 B 00197Fh to 001900h
	Info C	128 B 0018FFh to 001880h
	Info D	128 B 00187Fh to 001800h
Bootloader (BSL) memory (flash)	BSL 3	512 B 0017FFh to 001600h
	BSL 2	512 B 0015FFh to 001400h
	BSL 1	512 B 0013FFh to 001200h
	BSL 0	512 B 0011FFh to 001000h
Peripherals	Size	4KB 000FFFh to 0h

In MSP430F5529 micro, there is a Cyclic Redundancy Check (CRC) hardware. This hardware has several uses. Most commonly it is used to ensure or check data integrity against data corruption. In communications and data storage, CRC is highly needed since there is no other good and simple mean to cross-check data integrity efficiently and quickly.

CRC is like an agreement/encryption that two/more parties agree upon. As an example, consider two people, ***A*** and ***B*** sharing a common safe. ***A*** leaves a message in the safe for ***B***. ***B*** has to open the safe with the same kind of key that ***A*** used to lock the safe or else ***B*** won't get the message. Here the keys form the agreement that both of these persons agreed upon and the safe is the method to encapsulate the message from unwanted people.

Modern communication and data storage use CRC just like the example detailed above. CRC is an algorithm that tells dictates this encryption. Data is sent/stored along with a CRC code. When the sent/stored data is received/read along with the CRC content of the data, CRC is recalculated and verified against the sent/stored CRC. If both sent/stored CRC and received/read CRC match then it is safe to assume that no data corruption has occurred.

Shown below is the CRC module block diagram of MSP430F5529. The CRC module produces a signature for a given sequence of data values. The signature is generated through a feedback path from data bits 0, 4, 11, and 15. The CRC signature is based on the polynomial given in the CRC-CCITT-BR polynomial.



Code Example

```
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"
#include "lcd_print.h"


#define INFOA_START     0x1980
#define INFOA_END       0x19FF

#define INFOB_START     0x1900
#define INFOB_END       0x197F

#define INFOC_START     0x1880
#define INFOC_END       0x18FF

#define INFOD_START     0x1800
#define INFOD_END       0x187F


void clock_init(void);
void GPIO_init(void);
char Flash_Read_Char(unsigned int address);
unsigned int Flash_Read_Word(unsigned int address);


void main(void)
{
    unsigned char s = 0x00;
    unsigned char temp = 0x00;
    unsigned char value[2] = {0x00, 0x00};
    unsigned char set_values[4] = {33, 64, 99, 16};
    unsigned int status = 0x0000;

    WDT_A_hold(WDT_A_BASE);

    clock_init();
    GPIO_init();

    LCD_init();
    LCD_clear_home();

    CRC_setSeed(CRC_BASE,
                0xABCD);

    CRC_set8BitData(CRC_BASE,
                    Flash_Read_Char(INFOB_START));

    temp = CRC_getResult(CRC_BASE);

    LCD_goto(0, 0);
    LCD_putstr("Checking CRC");

    LCD_goto(0, 1);
    LCD_putstr("STO");

    LCD_goto(8, 1);
    LCD_putstr("CRC");

    print_C(3, 1, Flash_Read_Char(INFOB_START + 1));
    print_C(11, 1, temp);

    delay_ms(1600);
```

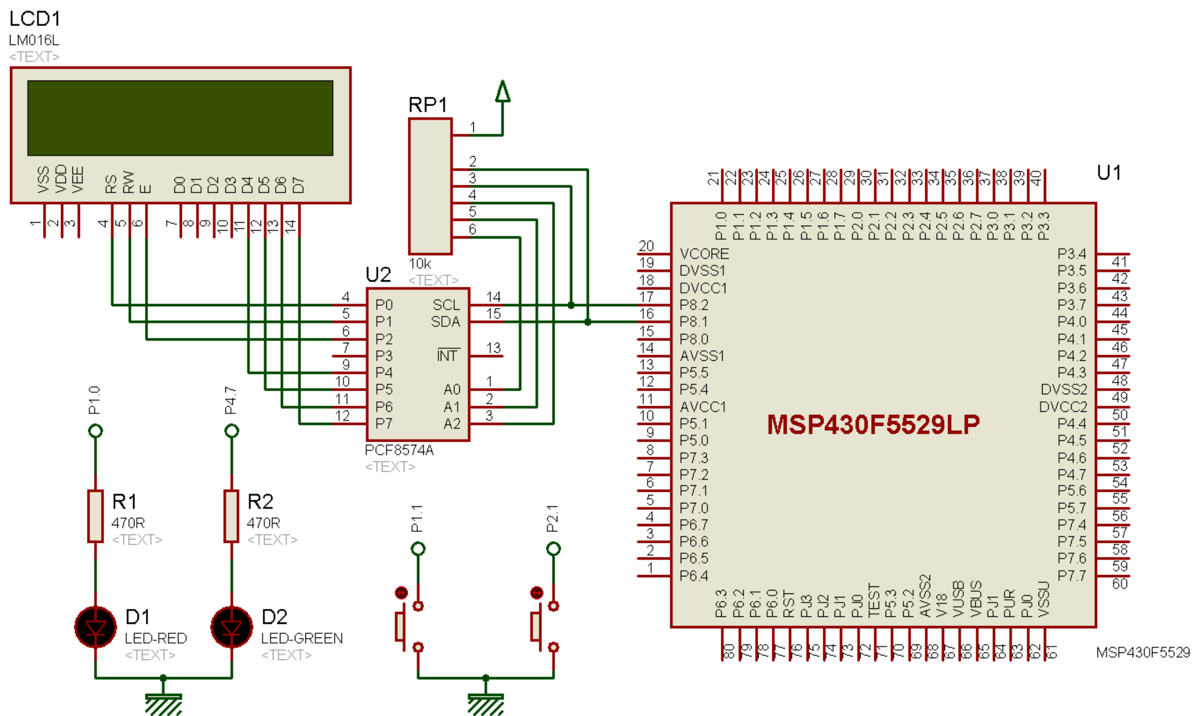
```
LCD_clear_home();

LCD_goto(0, 0);
LCD_putstr("INFO A Seg.");

if(temp != Flash_Read_Char(INFOB_START + 1))
{
    do
    {
        FlashCtl_eraseSegment((unsigned char *)INFOB_START);

        status = FlashCtl_performEraseCheck((unsigned char *)INFOB_START,
                                            128);
    }while (status == STATUS_FAIL);
}

if((GPIO_getInputPinValue(GPIO_PORT_P2,
                          GPIO_PIN1) == false))
{
    GPIO_setOutputHighOnPin(GPIO_PORT_P4,
                            GPIO_PIN7);

    FlashCtl_unlockInfoA();

    do
    {
        FlashCtl_eraseSegment((unsigned char *)INFOA_START);

        status = FlashCtl_performEraseCheck((unsigned char *)INFOA_START,
                                            128);
    }while (status == STATUS_FAIL);

    FlashCtl_write8(set_values,
                    (unsigned char *)INFOA_START,
                    4);

    FlashCtl_lockInfoA();

    LCD_goto(0, 1);
    LCD_putstr("Set: 33 64 99 16");

    delay_ms(3000);

    GPIO_setOutputLowOnPin(GPIO_PORT_P4,
                           GPIO_PIN7);
}

else
{
    LCD_goto(0, 1);
    LCD_putstr("Read");

    for(s = 0; s < 4; s++)
    {
        temp = Flash_Read_Char(INFOA_START + s);
        print_C(5, 1, s);
        print_C(13, 1, temp);
        delay_ms(2000);
    }
}


LCD_clear_home();

LCD_goto(0, 0);
```

```
    LCD_putstr("INFO B Seg.");


    temp = Flash_Read_Char(INFOB_START);
    LCD_goto(0, 1);
    LCD_putstr("WR: ---");
    LCD_goto(9, 1);
    LCD_putstr("RD:");
    print_C(12, 1, temp);
    delay_ms(2000);

    temp = 0;

    while(1)
    {
        if(GPIO_getInputPinValue(GPIO_PORT_P1,
                                 GPIO_PIN1) == false)
        {
            while(GPIO_getInputPinValue(GPIO_PORT_P1,
                                        GPIO_PIN1) == false);

            print_C(3, 1, value[0]);

            do
            {
                FlashCtl_eraseSegment((unsigned char *)INFOB_START);

                status = FlashCtl_performEraseCheck((unsigned char *)INFOB_START,
                                                    128);
            }while (status == STATUS_FAIL);

            CRC_setSeed(CRC_BASE,
                        0xABCD);

            CRC_set8BitData(CRC_BASE,
                            value[0]);

            value[1] = CRC_getResult(CRC_BASE);

            FlashCtl_write8(value,
                            (unsigned char *)INFOB_START,
                            2);

            GPIO_setOutputHighOnPin(GPIO_PORT_P4,
                                    GPIO_PIN7);

            delay_ms(1000);

            GPIO_setOutputLowOnPin(GPIO_PORT_P4,
                                   GPIO_PIN7);

        }

        delay_ms(20);

        print_C(3, 1, value[0]);
        value[0] = temp++;

        delay_ms(200);
    };
}


void clock_init(void)
{
```

```
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_3,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_2);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1,
                                         GPIO_PIN1);

    GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P2,
                                         GPIO_PIN1);

    GPIO_setAsOutputPin(GPIO_PORT_P1,
                        GPIO_PIN0);

    GPIO_setDriveStrength(GPIO_PORT_P1,
                          GPIO_PIN0,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setDriveStrength(GPIO_PORT_P4,
                          GPIO_PIN7,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);
}


char Flash_Read_Char(unsigned int address)
{
    char value = 0x00;
    char *FlashPtr = (char *)address;

    value = *FlashPtr;

    return value;
```

```
}

unsigned int Flash_Read_Word(unsigned int address)
{
    unsigned int value = 0x0000;
    unsigned int *FlashPtr = (unsigned int *)address;

    value = *FlashPtr;

    return value;
}
```

Hardware Setup



Explanation

The idea here is to use internal flash memory for storing non-volatile user data and CRC module for verification of data integrity.

MSP430F5529's flash memory is partitioned into main, information, and BSL memory sections. The main flash memory is used for storing application code, BSL memory is used for storing bootloader and the information memory is used for storing non-volatile user data. Check the memory map shown below. Here two out of three types of flash memories are shown only. Different sizes of data can be read/written in flash memory but when erasing only segment-sized erase is possible. The segment size of main and BSL memories is 512 bytes and the segment size of information memories is 128 bytes. Since information memory segments are smaller than other flash memories, it is best suited for user data storage for obvious memory segment size advantage. Lesser size means faster operation, lesser wear and efficient memory management. As mentioned before, frequent writes/erases wear

memories and so it is a good practice not do either frequently. Complex software-based wear-levelling techniques and using RAM-backed buffers can be applied to reduce memory wears.

Segment A is a very important segment as it can be used to stores important/critical internal calibration data, configurations and other important set points. This is why, it is protected and locked separately. It is wise to leave it and use the other three segments of information memory space to store user data that frequently change.

		MSP430F5529 MSP430F5528 MSP430F5519
Memory (flash) Main: interrupt vector	Total Size	128KB 00FFFFh to 00FF80h
Main: code memory	Bank D	32KB 0243FFh to 01C400h
	Bank C	32KB 01C3FFh to 014400h
	Bank B	32KB 0143FFh to 00C400h
	Bank A	32KB 00C3FFh to 004400h
Information memory (flash)	Info A	128 B 0019FFh to 001980h
	Info B	128 B 00197Fh to 001900h
	Info C	128 B 0018FFh to 001880h
	Info D	128 B 00187Fh to 001800h

Now let's see how the code works here.

Firstly, let us start by defining the location boundaries of the information memory segments.

```
#define INFOA_START     0x1980
#define INFOA_END       0x19FF

#define INFOB_START     0x1900
#define INFOB_END       0x197F

#define INFOC_START     0x1880
#define INFOC_END       0x18FF

#define INFOD_START     0x1800
#define INFOD_END       0x187F
```

CRC check is done in the beginning of the code although at first run CRC check is invalid as no user data is initially stored. However, this comes to use when we reset or power up after saving our data.

For CRC module to work, we need to give a seed, i.e. a number in form of a key. Here it is arbitrarily set as *0XABCD*. CRC module uses this seed and given data to compute CRC result. After this we visually check if the stored CRC matches with the calculated CRC. The check is done in code too. The CRC values are stored in two locations of the B segment of information memory.

```
CRC_setSeed(CRC_BASE, 0xABCD);

CRC_set8BitData(CRC_BASE, Flash_Read_Char(INFOB_START));

temp = CRC_getResult(CRC_BASE);

LCD_goto(0, 0);
LCD_putstr("Checking CRC");

LCD_goto(0, 1);
LCD_putstr("STO");

LCD_goto(8, 1);
LCD_putstr("CRC");

print_C(3, 1, Flash_Read_Char(INFOB_START + 1));
print_C(11, 1, temp);


delay_ms(1600);

LCD_clear_home();

LCD_goto(0, 0);
LCD_putstr("INFO A Seg.");

if(temp != Flash_Read_Char(INFOB_START + 1))
{
    do
    {
        FlashCtl_eraseSegment((unsigned char *)INFOB_START);

        status = FlashCtl_performEraseCheck((unsigned char *)INFOB_START,
                                             128);
     }while(status == STATUS_FAIL);
}
```

If the contents of both locations of Info B memory are different then it is likely that the memories are corrupt/empty and so an erase cycle is necessary to wipe out inconsistent data. The entire Info B segment is erased. If it is otherwise then the erase is not performed.

Now let us put some simulated fixed data as if these data are set points of some process or hardware. To set these data in Info A memory we have to push P2.1 button. First, the segment is unlocked and then an erase is performed to get rid of old data. After the erase cycle, data (array named "*set_values*") is stored and the flash segment is locked. The locking ensures that no other operation is possible with this memory unless unlocked.

If the P2.1 button is not pressed then stored data is simply read.

Note no CRC check is performed for this segment.

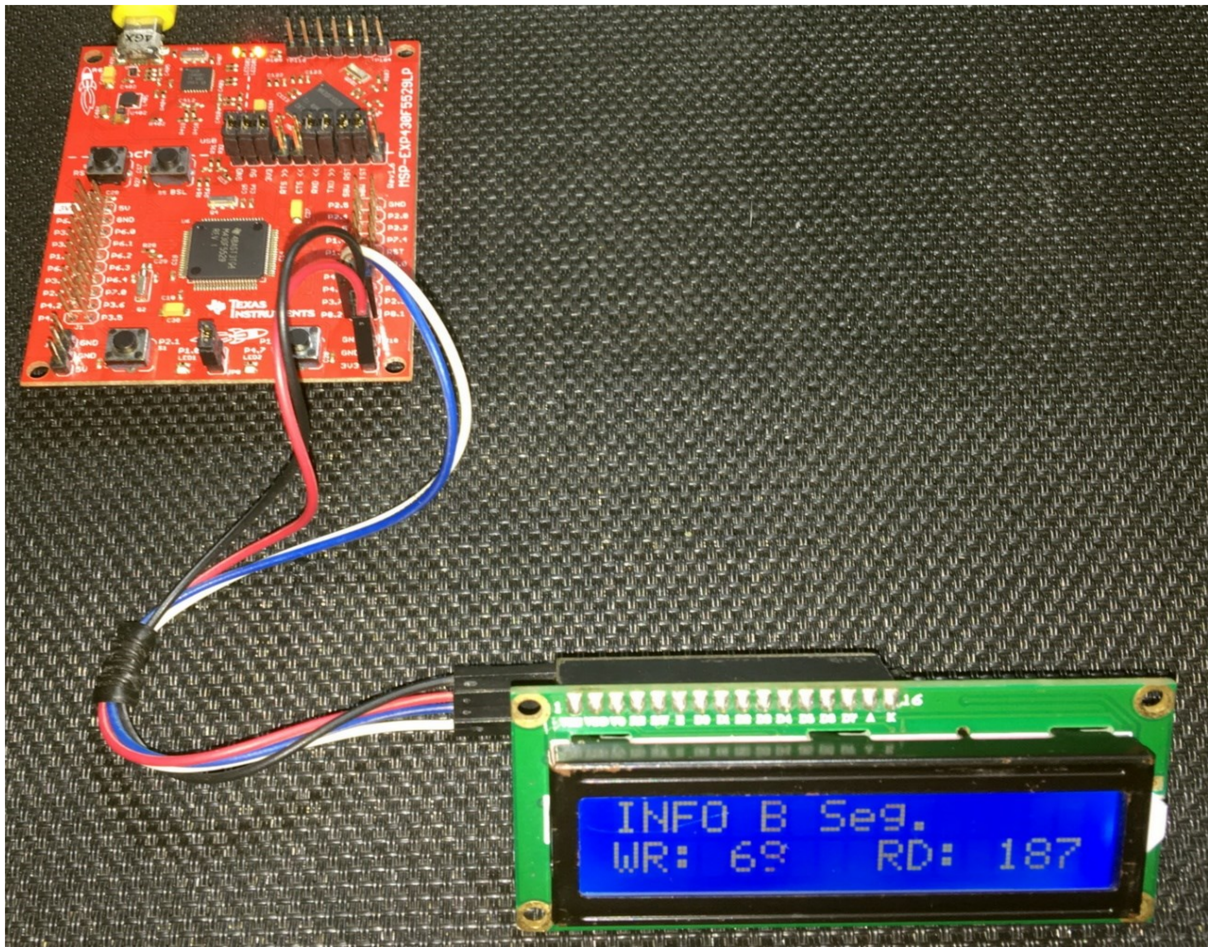
```
if((GPIO_getInputPinValue(GPIO_PORT_P2, GPIO_PIN1) == false))
{
    GPIO_setOutputHighOnPin(GPIO_PORT_P4, GPIO_PIN7);

    FlashCtl_unlockInfoA();

    do
    {
       FlashCtl_eraseSegment((unsigned char *)INFOA_START);

       status = FlashCtl_performEraseCheck((unsigned char *)INFOA_START, 128);
     }while (status == STATUS_FAIL);

     FlashCtl_write8(set_values, (unsigned char *)INFOA_START, 4);

     FlashCtl_lockInfoA();

     LCD_goto(0, 1);
     LCD_putstr("Set: 33 64 99 16");

     delay_ms(2600);

     GPIO_setOutputLowOnPin(GPIO_PORT_P4, GPIO_PIN7);
    }

else
{
    LCD_goto(0, 1);
    LCD_putstr("Read");

    for(s = 0; s < 4; s++)
    {
        temp = Flash_Read_Char(INFOA_START + s);
        print_C(5, 1, s);
        print_C(13, 1, temp);
        delay_ms(2000);
    }
}
```

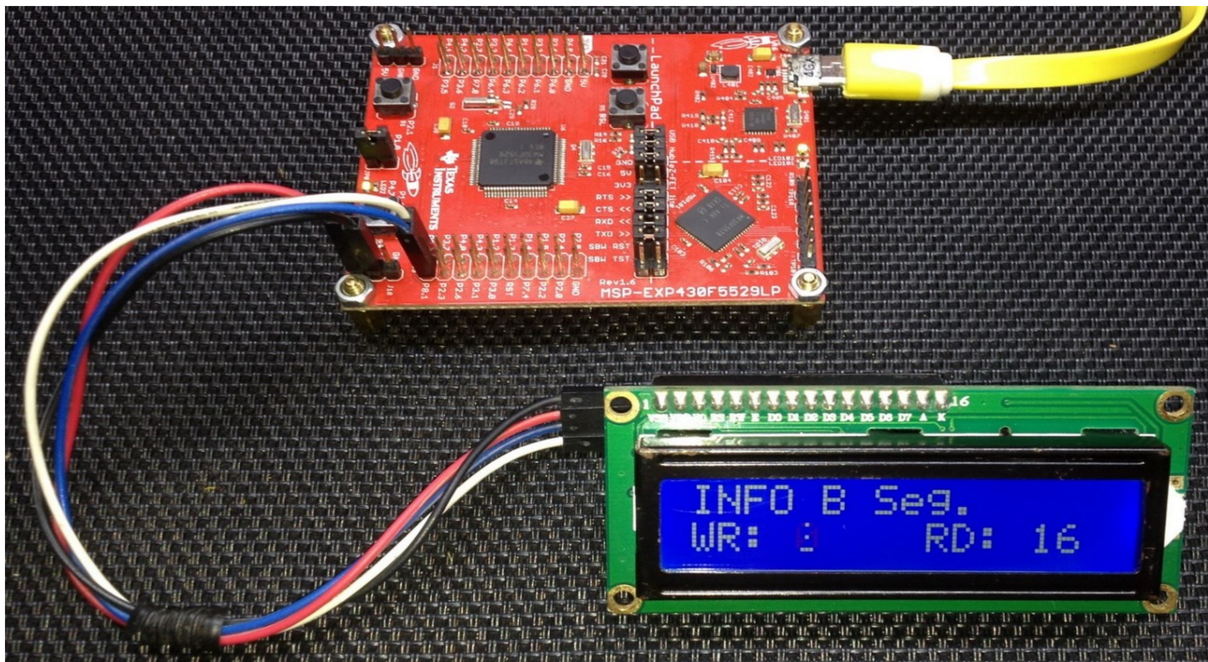
All of the tasks mentioned so far are done before the main loop. In the main loop a value named *"temp"* is incremented. When P1.1 button is pressed, the current value of *temp* is stored in Info B segment. Since we cannot rewrite or overwrite a flash memory, we must perform an erase operation before storing new data. When a new data is stored, a CRC is also performed with *0xABCD* as seed as in the beginning of the code. The new data and CRC are saved in two different locations. These two values will be used for checking data integrity when the MSP430 micro resets or powers up next. This is what we saw at the beginning of the code.

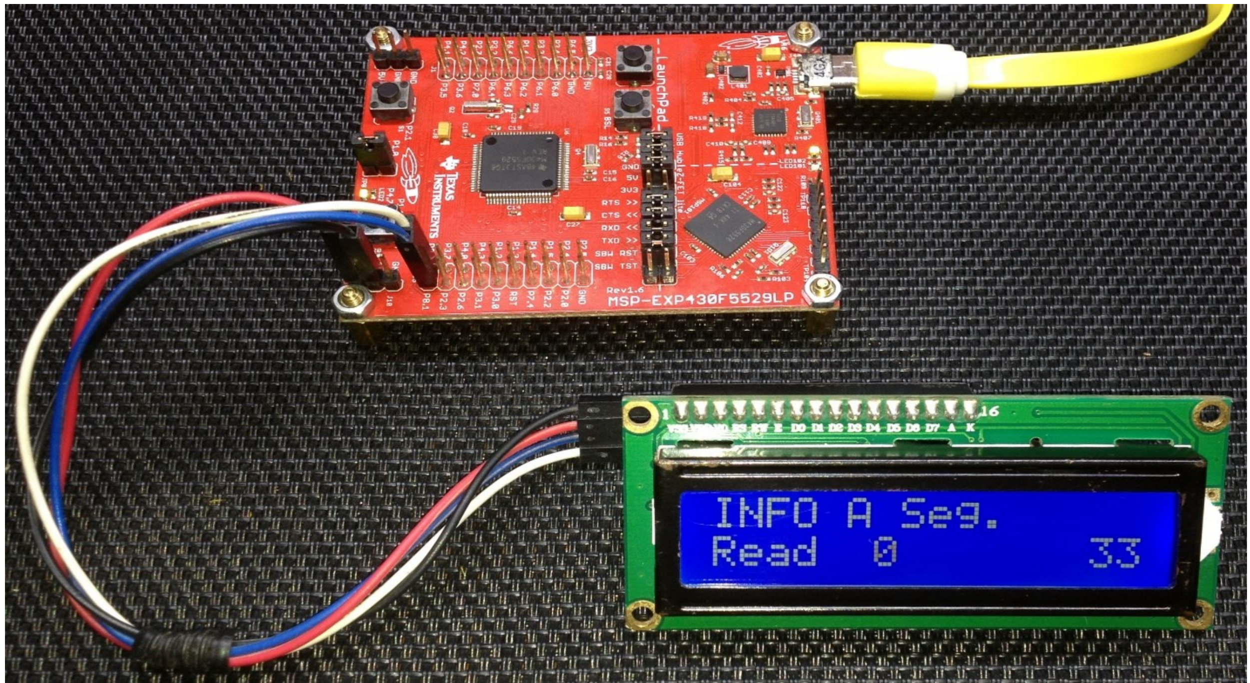
Note that there is no Info B locking unlocking feature.

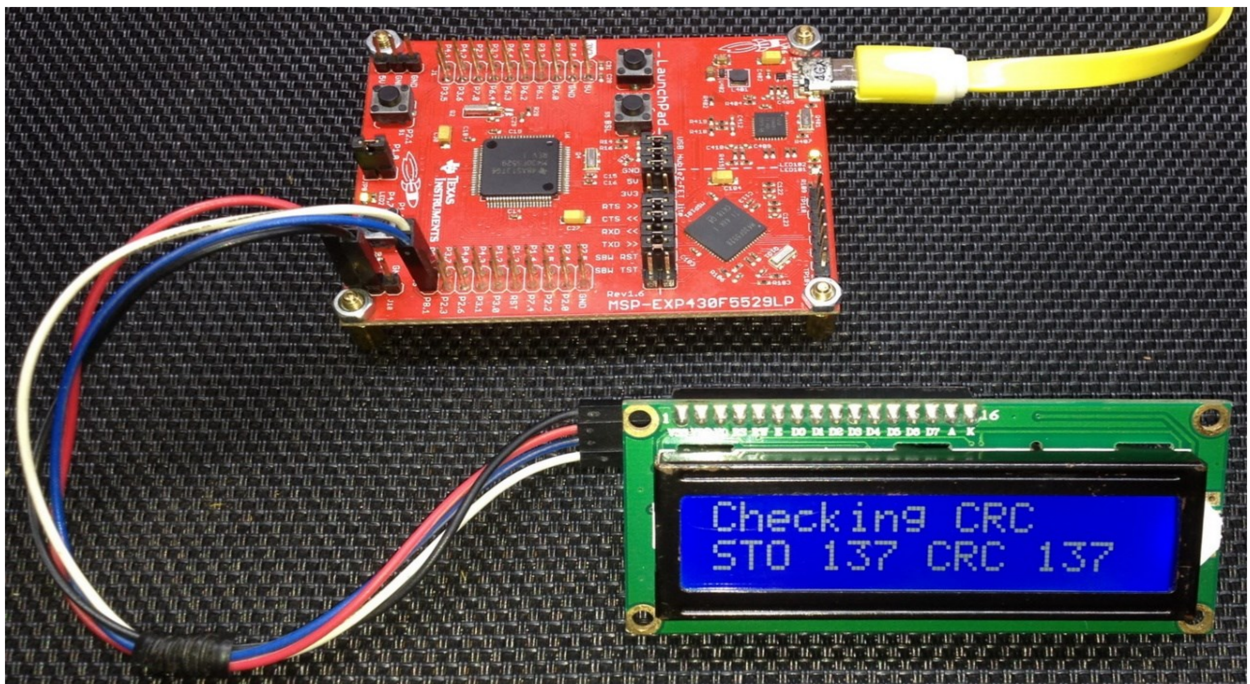
```
if(GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1) == false)
{
   while(GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1) == false);

   print_C(3, 1, value[0]);

   do
   {
       FlashCtl_eraseSegment((unsigned char *)INFOB_START);

       status = FlashCtl_performEraseCheck((unsigned char *)INFOB_START, 128);
    }while (status == STATUS_FAIL);

    CRC_setSeed(CRC_BASE, 0xABCD);

    CRC_set8BitData(CRC_BASE, value[0]);

    value[1] = CRC_getResult(CRC_BASE);

    FlashCtl_write8(value, (unsigned char *)INFOB_START, 2);

    GPIO_setOutputHighOnPin(GPIO_PORT_P4, GPIO_PIN7);

    delay_ms(1000);

    GPIO_setOutputLowOnPin(GPIO_PORT_P4, GPIO_PIN7);
}

delay_ms(20);

print_C(3, 1, value[0]);
value[0] = temp++;

delay_ms(200);
```

Demo









Demo video: https://youtu.be/EeHk4eZ3WEw

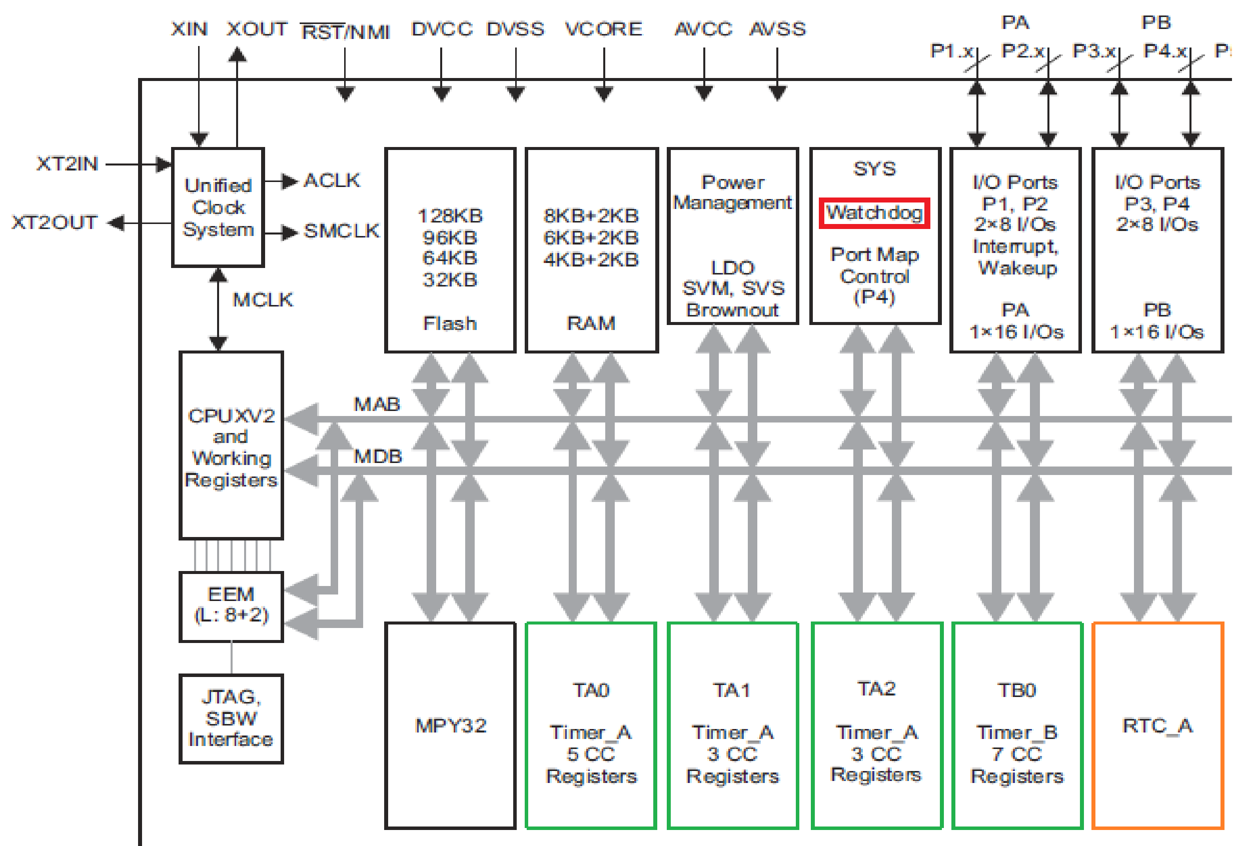
Timer Overview

MSP430F5529 microcontroller has six timers divided in three categories – General Purpose Timers like Timer A and Timer B, Watchdog Timer (WDT) and Real Time Clock (RTC).

There are

- 3 Timer As – T0A5, T1A3 and T2A3
- 1 Timer B – T0B7
- 1 WDT_A
- 1 RTC_A

The cut-away block diagram of MSP430F5529's internal peripherals shown below highlights these timers.



Timer A and Timer B are almost same with minor differences. Timer B offers better PWM options. As their names suggest, RTC_A and WDTA_A are specialized timers for specific or special purposes. They can be alternatively used like regular timers with some limitations.

These timers can be fed with bewildering number of clock combinations offered by the UCS. There are options for external sources in some cases too. The structures and the operations of all of these timers is similar to what we have seen in my previous MSP430 tutorial. The only exception is the RTC_A module as this timer was not available then. Before trying to use any timer, it is my recommendation to go through the driverlib files of the respective timer. This will enable visualizing what can and cannot

be done with a given timer module. This also reduces time to read technical documents and datasheet. However, I will still recommend reading these docs should one want to master the timers.

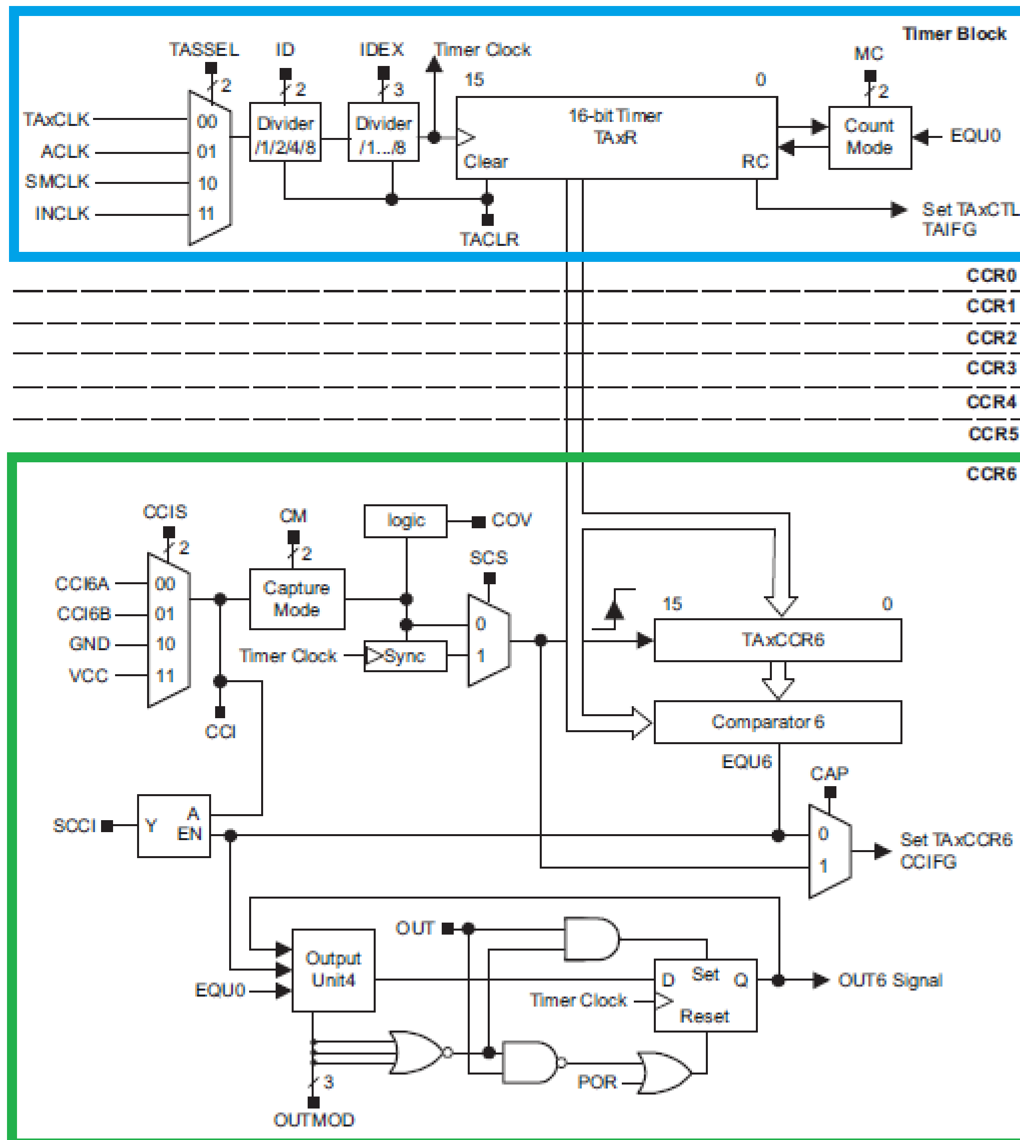
Timer A and B are 16-bit general-purpose timers with capture-compare (CC) modules. The CC modules can be used for either capturing waveforms or PWM/waveform generations. If CC modules are not needed, these timers can be used for timing events and keeping track of time. In TI's literature, we will see that timers are named as ***TxYn*** where ***Y*** represents timer class/category, i.e. Timer A, Timer B, etc, ***x*** represents timer number or instantiation and ***n*** represents number of CC channels. Therefore, ***T0A5*** means 0th instantiation of Timer A having 5 CC channels or simply 0th A-type timer having 5 CC channels. Sometimes, the CC channels are ignored and the timers are simply named like ***TA2***. This naming states 2nd instantiation of Timer A or simply 2nd A-type Timer. Thus, there is a short naming format and a long naming format for general purpose timers.

MSP430x5xxx and MSP430x6xxx family of microcontrollers have three types of RTCs and they vary slightly in some features. MSP430F5529 has one RTC_A module which is the basic of all three types. RTC_A lacks battery-backup option unlike other RTCs. RTC_A is a special timer with a 32-bit counter that is intended for time-keeping applications, i.e., clock and calendar. Since time-keeping needs to be very accurate, RTC has special hardware arrangements to take care of so. It is my personal recommendation to use dedicated 32.768 kHz temperature-compensated crystal oscillator (TCXO) as the source of RTC_A clock. This will ensure better accuracy and precision. It is a common problem for most RTCs to either lead ahead or lack behind actual time. RTC_A can also be used as a 32-bit timer if time-keeping is not needed. However, there are no hardware CC channels with RTC_A.

WDT_A is similar but more advanced than the WDT+ seen in my previous MSP430 tutorial. It has 8 timer intervals rather than 4. This special timer is primarily intended for avoiding erratic/irresponsive behaviour in the event of an electromagnetic interference (EMI) or conditions that leave a micro into an unforeseen loop or other software issue. Again, this timer can be alternatively used like a regular timer if watchdog feature is not needed. However, this timer too doesn't have any CC channel.

Free-Running Timer – TA0

A free-running timer is a timer that is left to run on its own. It has limited uses in regular life as it is based on polling. Such timers are useful for generating random numbers, timer-based delay functions, etc. Any timer of a MSP430 microcontroller can be used for making such a timer.

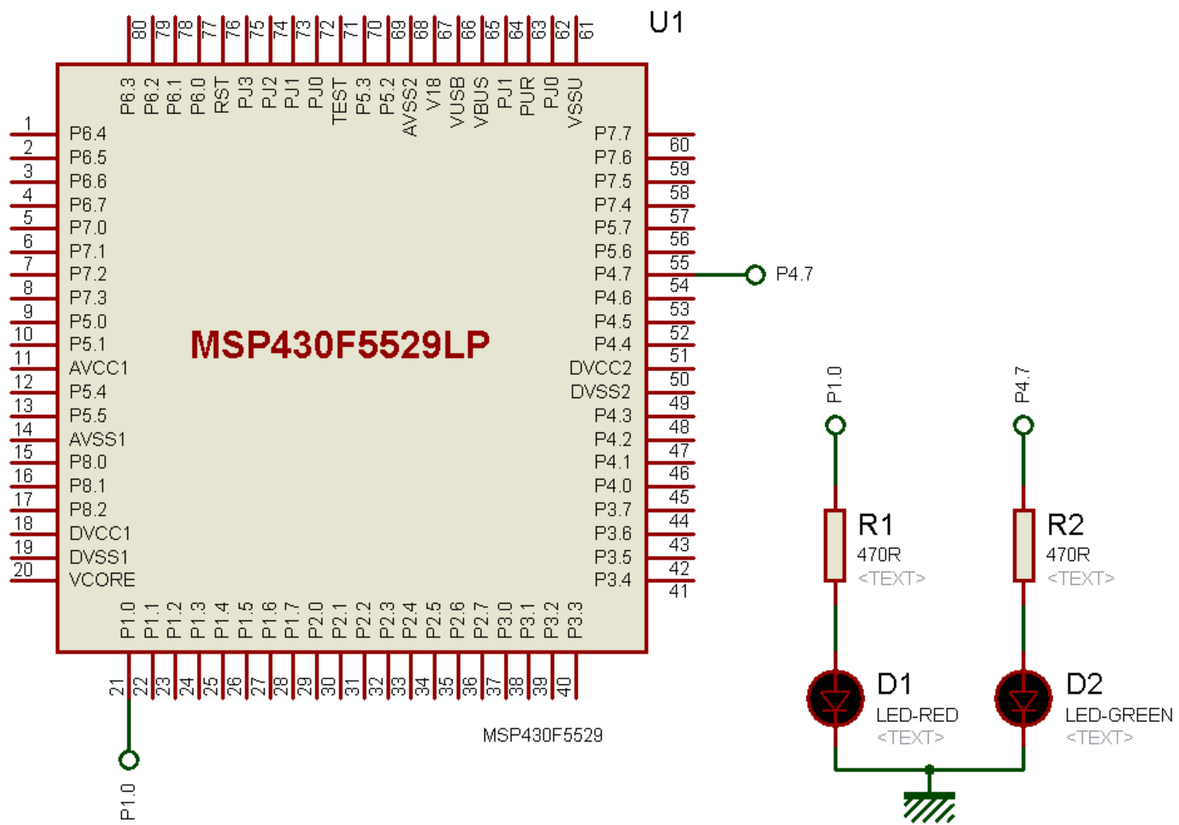


Code Example

```
#include "driverlib.h"
#include "delay.h"


void clock_init(void);
void GPIO_init(void);
void timer_T0A5_init(void);


void main(void)
{
    WDT_A_hold(WDT_A_BASE);

    clock_init();
    GPIO_init();
    timer_T0A5_init();

    while(1)
    {
        if(Timer_A_getCounterValue(__MSP430_BASEADDRESS_T0A5__) >= 32768)
        {
            GPIO_setOutputLowOnPin(GPIO_PORT_P1,
                                   GPIO_PIN0);

            GPIO_setOutputHighOnPin(GPIO_PORT_P4,
                                    GPIO_PIN7);
        }
        else
        {
            GPIO_setOutputHighOnPin(GPIO_PORT_P1,
                                    GPIO_PIN0);

            GPIO_setOutputLowOnPin(GPIO_PORT_P4,
                                   GPIO_PIN7);
        }
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_2);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_MCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_SMCLK,
```

```
                        UCS_REFOCLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsOutputPin(GPIO_PORT_P1,
                        GPIO_PIN0);

    GPIO_setDriveStrength(GPIO_PORT_P1,
                          GPIO_PIN0,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setDriveStrength(GPIO_PORT_P4,
                          GPIO_PIN7,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);
}


void timer_T0A5_init(void)
{
    Timer_A_initContinuousModeParam ContinuousModeParam =
    {
         TIMER_A_CLOCKSOURCE_SMCLK,
         TIMER_A_CLOCKSOURCE_DIVIDER_1,
         TIMER_A_TAIE_INTERRUPT_DISABLE,
         TIMER_A_DO_CLEAR,
         false
    };

    Timer_A_stop(__MSP430_BASEADDRESS_T0A5__);

    Timer_A_clearTimerInterrupt(__MSP430_BASEADDRESS_T0A5__);

    Timer_A_initContinuousMode(__MSP430_BASEADDRESS_T0A5__,
                               &ContinuousModeParam);

    Timer_A_startCounter(__MSP430_BASEADDRESS_T0A5__,
                         TIMER_A_CONTINUOUS_MODE);
}
```

Hardware Setup



Explanation

Onboard LED GPIOs are setup as outputs. It is same as with other examples. The interesting part for us here is the timer setup.

```
void timer_T0A5_init(void)
{
    Timer_A_initContinuousModeParam ContinuousModeParam =
    {
         TIMER_A_CLOCKSOURCE_SMCLK,
         TIMER_A_CLOCKSOURCE_DIVIDER_1,
         TIMER_A_TAIE_INTERRUPT_DISABLE,
         TIMER_A_DO_CLEAR,
         false
    };
    Timer_A_stop(__MSP430_BASEADDRESS_T0A5__);
    Timer_A_clearTimerInterrupt(__MSP430_BASEADDRESS_T0A5__);
    Timer_A_initContinuousMode(__MSP430_BASEADDRESS_T0A5__, &ContinuousModeParam);
    Timer_A_startCounter(__MSP430_BASEADDRESS_T0A5__, TIMER_A_CONTINUOUS_MODE);
}
```

For demoing free-running timer, i.e. timer without interrupt or CC channels, TA0 is used. Note it is driven by SMCLK. SMCLK’s clock source is the internal 32 kHz REFOCLK oscillator.

```
UCS_initClockSignal(UCS_SMCLK, UCS_REFOCLK_SELECT, UCS_CLOCK_DIVIDER_1);
```

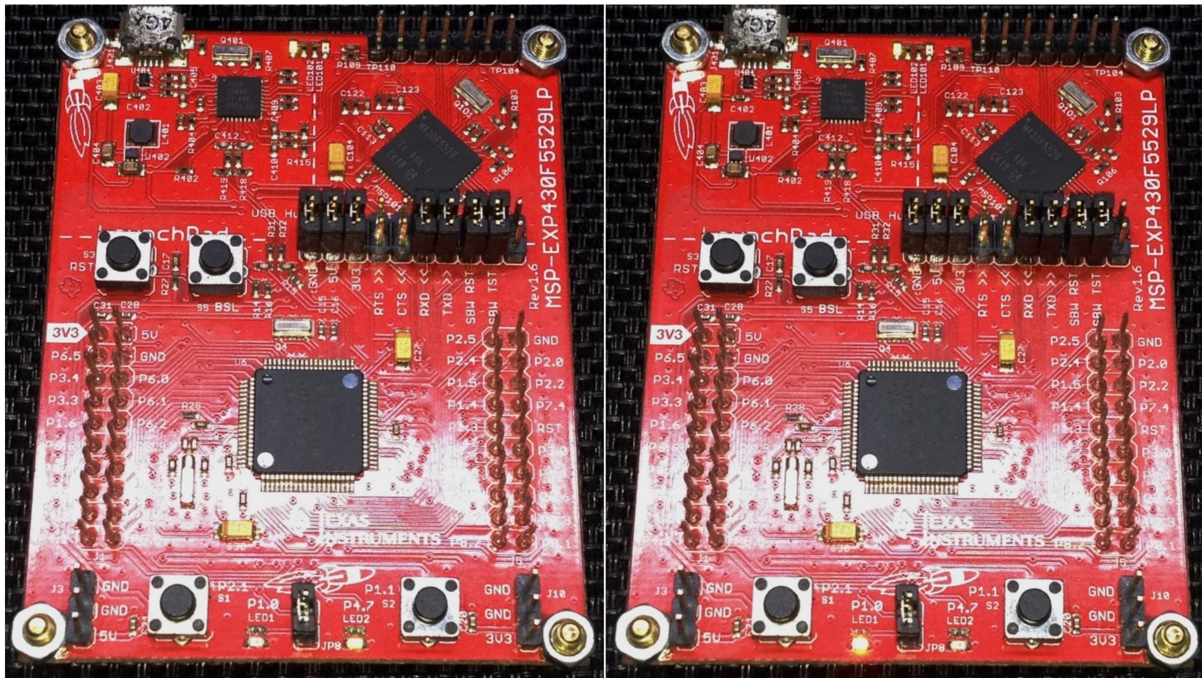
No divider is used and so we can roughly expect 32000 ticks per second. We would also want to clear past settings and disable timer interrupt since it is a no-interrupt timer example. Initially the timer is halted because we want to start it right after setting the timer up. Note that here the timer is set in continuous mode of operation. In continuous mode, the timer counts up from 0 to top value of 0xFFFF or 65535 and then rolls over to zero. Shown below are modes of operation of Timer A.

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh.
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero.

We want to have the onboard LEDs flash equally and alternatively. Since the timer is running at approximately 32 kHz, it will overflow in about 2 second time. To get the LEDs alter state every second, we have to divide 65535 timer count into two equal fractions and check the timer count on every main loop passes. It is checked if the timer count is 32678 or more, i.e. half of 65535. For one count fraction, i.e. from 0 to 32767, the red LED will be lit while the green LED will be off. The same goes for the other fraction, i.e. from 32768 to 65535 count, but the LEDs will be in different logic states.

```
if(Timer_A_getCounterValue(__MSP430_BASEADDRESS_T0A5__) >= 32768)
{
   GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
   GPIO_setOutputHighOnPin(GPIO_PORT_P4, GPIO_PIN7);
}

else
{
    GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
    GPIO_setOutputLowOnPin(GPIO_PORT_P4, GPIO_PIN7);
}
```

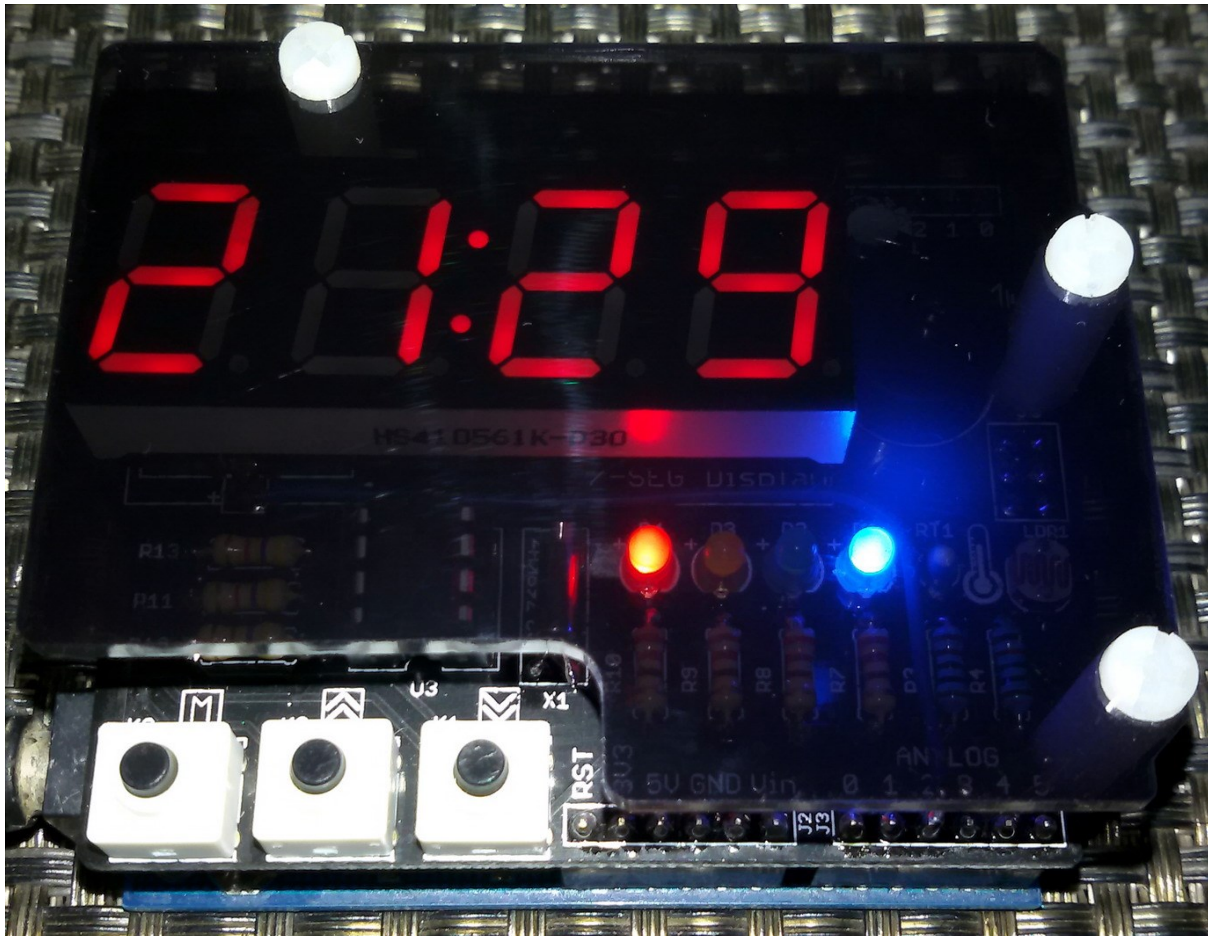
Demo



Demo video: https://youtu.be/ZVoiIhzBDVg

Timer Interrupt – TA1

Timers become more useful when used with interrupts. At fixed intervals, interrupts occur and this process allows us to perform tasks that need periodic update. One common use of timer interrupt is to drive multiple seven segment displays. A timer is coded in such a way that its overflow interrupt triggers every millisecond. During this time, one seven segment display is updated. Since one millisecond interval is a very short duration for human vision to notice, we can update several seven segment displays in a matter of few milliseconds and our eyes will see that all displays have been simultaneously updated.



Code Example

```
#include "driverlib.h"
#include "delay.h"


#define GATE_HIGH                    GPIO_setOutputHighOnPin(GPIO_PORT_P8, GPIO_PIN2)
#define GATE_LOW                     GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN2)

#define CLK_HIGH                     GPIO_setOutputHighOnPin(GPIO_PORT_P8, GPIO_PIN1)
#define CLK_LOW                      GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN1)

#define A_HIGH                       GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN6)
#define A_LOW                        GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6)

#define B_HIGH                       GPIO_setOutputHighOnPin(GPIO_PORT_P4, GPIO_PIN0)
#define B_LOW                        GPIO_setOutputLowOnPin(GPIO_PORT_P4, GPIO_PIN0)

#define C_HIGH                       GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN3)
#define C_LOW                        GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN3)

#define D_HIGH                       GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7)
#define D_LOW                        GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7)

#define SW                           GPIO_getInputPinValue(GPIO_PORT_P3, GPIO_PIN1)

#define top_seg                      4
#define bot_seg                      0

#define HIGH                         true
#define LOW                          false


const unsigned char num[0x0A] = {0xED, 0x21, 0x8F, 0xAB, 0x63, 0xEA, 0xEE, 0xA1, 0xEF,
0xEB};
unsigned char data_values[0x09] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};


unsigned char n = 0x00;
unsigned char SW_in = 0x00;


void clock_init(void);
void GPIO_init(void);
void timer_T1A1_init(void);
void write_74HC164(register unsigned char value);
void write_74HC145(register unsigned char channel);
void show_LEDs(unsigned char LED1_state, unsigned char LED2_state, unsigned char
LED3_state, unsigned char LED4_state);
void show_numbers(signed int value, unsigned char pos);


#pragma vector = TIMER1_A1_VECTOR
__interrupt void Timer_A_ISR(void)
{
    Timer_A_clearTimerInterrupt(TIMER_A1_BASE);

    write_74HC164(data_values[n]);
    write_74HC145(n);

    n++;

    if(n > 9)
    {
```

```
        n = 0;
    }
}


void main(void)
{
    signed int i = 0;
    signed int j = 9999;

    WDT_A_hold(WDT_A_BASE);

    clock_init();
    GPIO_init();
    timer_T1A1_init();

    while(true)
    {
        switch(SW_in)
        {
             case 1:
             {
                 show_LEDs(1, 0, 0, 0);
                 break;
             }

             case 2:
             {
                 show_LEDs(0, 1, 0, 0);
                 break;
             }

             case 3:
             {
                 show_LEDs(0, 0, 1, 0);
                 break;
             }

             case 4:
             {
                 show_LEDs(0, 0, 0, 1);
                 break;
             }
        }

        SW_in = 0x00;

        i++;
        j--;

        if(i > 9999)
        {
            i = 0;
            j = 9999;
        }

        show_numbers(i, bot_seg);
        show_numbers(j, top_seg);

        delay_ms(100);
        show_LEDs(0, 0, 0, 0);
    };
}
```

```
void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_2);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_MCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsInputPin(GPIO_PORT_P3,
                       GPIO_PIN1);

    GPIO_setAsOutputPin(GPIO_PORT_P2,
                        GPIO_PIN3);
    GPIO_setDriveStrength(GPIO_PORT_P2,
                          GPIO_PIN3,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P2,
                        GPIO_PIN6);

    GPIO_setDriveStrength(GPIO_PORT_P2,
                          GPIO_PIN6,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P3,
                        GPIO_PIN7);

    GPIO_setDriveStrength(GPIO_PORT_P3,
                          GPIO_PIN7,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN0);

    GPIO_setDriveStrength(GPIO_PORT_P4,
                          GPIO_PIN0,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P8,
                        GPIO_PIN1);
```

```
    GPIO_setDriveStrength(GPIO_PORT_P8,
                          GPIO_PIN1,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P8,
                        GPIO_PIN2);

    GPIO_setDriveStrength(GPIO_PORT_P8,
                          GPIO_PIN2,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);
}


void timer_T1A1_init(void)
{
    Timer_A_initUpModeParam UpModeParam = {0};

    UpModeParam.clockSource = TIMER_A_CLOCKSOURCE_ACLK;
    UpModeParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_1;
    UpModeParam.timerPeriod = 9999;
    UpModeParam.timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_ENABLE;
    UpModeParam.captureCompareInterruptEnable_CCR0_CCIE =
                                          TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE;
    UpModeParam.timerClear = TIMER_A_DO_CLEAR;
    UpModeParam.startTimer = true;

    Timer_A_initUpMode(TIMER_A1_BASE,
                       &UpModeParam);

    Timer_A_startCounter(TIMER_A1_BASE,
                         TIMER_A_UP_MODE);

    __enable_interrupt();
}


void write_74HC164(register unsigned char value)
{
  register unsigned char s = 0x08;

  while(s > 0)
  {
        if((value & 0x80) != 0x00)
        {
            GATE_HIGH;
        }
        else
        {
            GATE_LOW;
        }

        CLK_HIGH;
        CLK_LOW;

        value <<= 1;
        s--;
  };
}


void write_74HC145(register unsigned char channel)
{
    A_LOW;
    B_LOW;
```

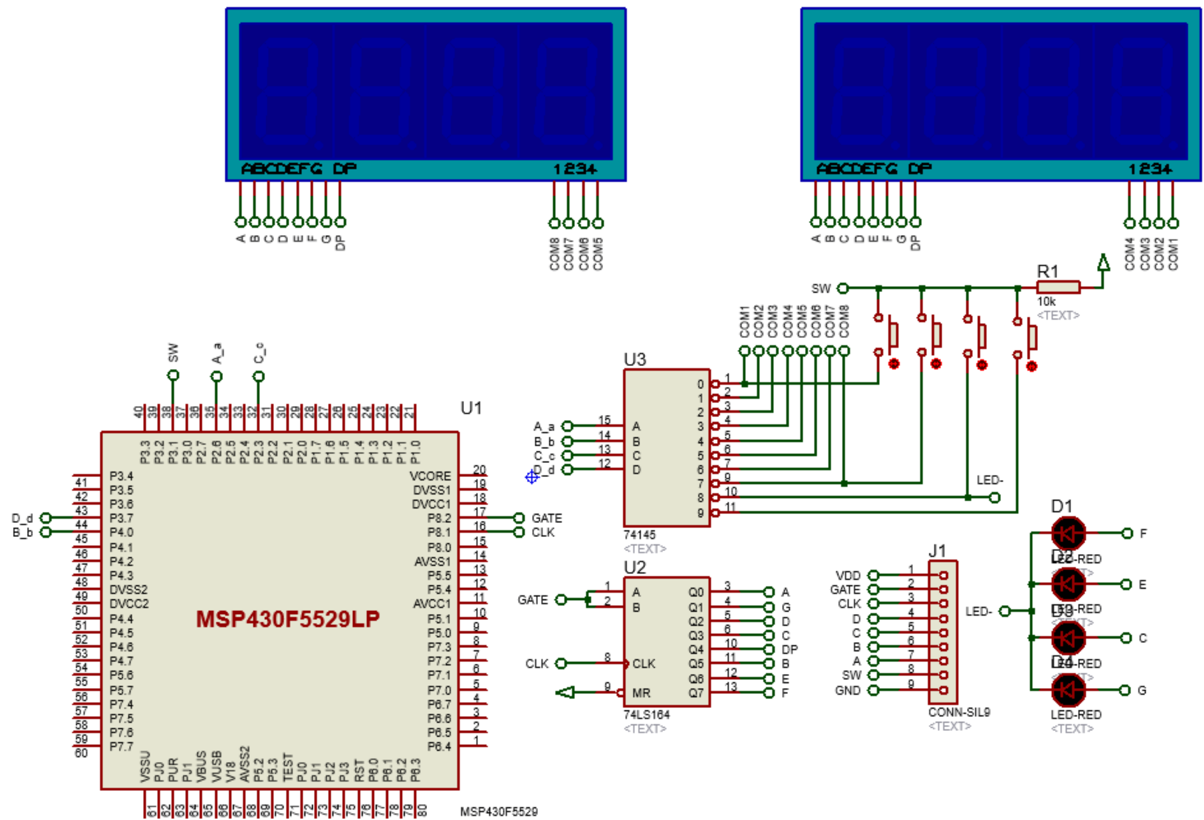
```
    C_LOW;
    D_LOW;

    switch(channel)
    {
        case 0:
        {
            if(SW == LOW)
            {
                SW_in = 1;
            }

            break;
        }

        case 1:
        {
            A_HIGH;
            B_LOW;
            C_LOW;
            D_LOW;

            break;
        }

        case 2:
        {
            A_LOW;
            B_HIGH;
            C_LOW;
            D_LOW;

            break;
        }

        case 3:
        {
            A_HIGH;
            B_HIGH;
            C_LOW;
            D_LOW;

            break;
        }

        case 4:
        {
            A_LOW;
            B_LOW;
            C_HIGH;
            D_LOW;

            break;
        }

        case 5:
        {
            A_HIGH;
            B_LOW;
            C_HIGH;
            D_LOW;

            break;
        }

        case 6:
```

```
        {
            A_LOW;
            B_HIGH;
            C_HIGH;
            D_LOW;

            break;
        }

        case 7:
        {
            A_HIGH;
            B_HIGH;
            C_HIGH;
            D_LOW;

            if(SW == LOW)
            {
                SW_in = 2;
            }

            break;
        }

        case 8:
        {
            A_LOW;
            B_LOW;
            C_LOW;
            D_HIGH;

            if(SW == LOW)
            {
                SW_in = 3;
            }

            break;
        }

        case 9:
        {
            A_HIGH;
            B_LOW;
            C_LOW;
            D_HIGH;

            if(SW == LOW)
            {
                SW_in = 4;
            }

            break;
        }
    }
}


void show_LEDs(unsigned char LED1_state, unsigned char LED2_state, unsigned char
LED3_state, unsigned char LED4_state)
{
  switch(LED1_state)
  {
    case HIGH:
    {
        data_values[8] |= 0x80;
        break;
```

```
    }
    case LOW:
    {
        data_values[8] &= 0x7F;
        break;
    }
  }

  switch(LED2_state)
  {
    case HIGH:
    {
      data_values[8] |= 0x40;
      break;
    }
    case LOW:
    {
      data_values[8] &= 0xBF;
      break;
    }
  }

  switch(LED3_state)
  {
    case HIGH:
    {
      data_values[8] |= 0x08;
      break;
    }
    case LOW:
    {
      data_values[8] &= 0xF7;
      break;
    }
  }

  switch(LED4_state)
  {
    case HIGH:
    {
      data_values[8] |= 0x02;
      break;
    }
    case LOW:
    {
      data_values[8] &= 0xFD;
      break;
    }
  }
}


void show_numbers(signed int value, unsigned char pos)
{
  register unsigned char ch = 0x00;

  if((value >= 0) && (value <= 9))
  {
    ch = (value % 10);
    data_values[(0 + pos)] = num[ch];
    data_values[(1 + pos)] = 0x00;
    data_values[(2 + pos)] = 0x00;
    data_values[(3 + pos)] = 0x00;
  }
  else if((value > 9) && (value <= 99))
  {
```

```
    ch = (value % 10);
    data_values[(0 + pos)] = num[ch];
    ch = ((value / 10) % 10);
    data_values[(1 + pos)] = num[ch];
    data_values[(2 + pos)] = 0x00;
    data_values[(3 + pos)] = 0x00;
  }
  else if((value > 99) && (value <= 999))
  {
    ch = (value % 10);
    data_values[(0 + pos)] = num[ch];
    ch = ((value / 10) % 10);
    data_values[(1 + pos)] = num[ch];
    ch = ((value / 100) % 10);
    data_values[(2 + pos)] = num[ch];
    data_values[(3 + pos)] = 0x00;
  }
  else if((value > 999) && (value <= 9999))
  {
    ch = (value % 10);
    data_values[(0 + pos)] = num[ch];
    ch = ((value / 10) % 10);
    data_values[(1 + pos)] = num[ch];
    ch = ((value / 100) % 10);
    data_values[(2 + pos)] = num[ch];
    ch = (value / 1000);
    data_values[(3 + pos)] = num[ch];
  }
}
```

Hardware Setup



Explanation

Timer interrupt is best understood by using it to scan multiple seven segment displays and buttons. This time timer TA1 is used.

```
void timer_T1A1_init(void)
{
    Timer_A_initUpModeParam UpModeParam = {0};

    UpModeParam.clockSource = TIMER_A_CLOCKSOURCE_ACLK;
    UpModeParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_1;
    UpModeParam.timerPeriod = 9999;
    UpModeParam.timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_ENABLE;
    UpModeParam.captureCompareInterruptEnable_CCR0_CCIE =
                                          TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE;
    UpModeParam.timerClear = TIMER_A_DO_CLEAR;
    UpModeParam.startTimer = true;

    Timer_A_initUpMode(TIMER_A1_BASE, &UpModeParam);

    Timer_A_startCounter(TIMER_A1_BASE, TIMER_A_UP_MODE);

    __enable_interrupt();
}
```

Timer TA1 is clocked with ACLK which in turn is being feed with 4MHz XT2CLK source. No divider is used at any stage and so, the timer will tick every 0.25µs.

```
UCS_initClockSignal(UCS_ACLK, UCS_XT2CLK_SELECT, UCS_CLOCK_DIVIDER_1);
```

We wish to have a time period of 2.5 ms and thus, here we will be using the timer in up mode with a top value of 9999, i.e. 10000 counts. Up mode is similar to previously seen continuous mode. The only difference is the fact that in continuous mode the top value of a timer is fixed at 65535 count while in up mode, the top value can be set to anything between 0 to 65535. The timer will count from 0 to 9999, i.e. there are 10000 ticks before timer overflow. Since one tick is 0.25 µs, 10000 ticks equal 2.5ms (0.25 µs x 10000 = 2.5 ms).

We will only need timer interrupt. We won't be using any capture-compare interrupt and so we have to disable it. As with previous examples we have to clear any past setting although there is none.

After setting up the timer's settings, we will just start it and enable global interrupt as to begin time count.

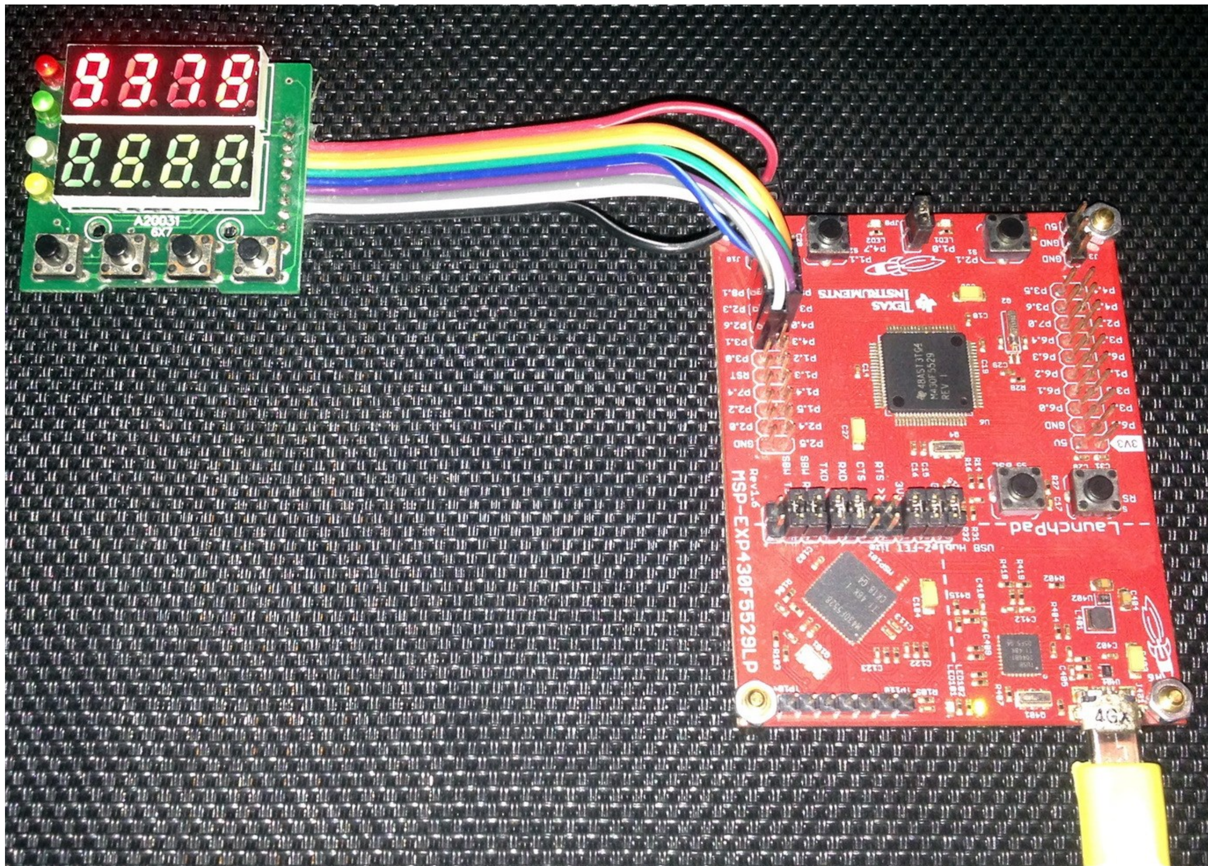
Now let's see what's being done inside the timer interrupt? According to the schematic and objective of this project, we need to update all seven segment displays and read the 4-bit keypad so quickly as if everything appears to work in real time and without any lag.

```
#pragma vector = TIMER1_A1_VECTOR
__interrupt void Timer_A_ISR(void)
{
    Timer_A_clearTimerInterrupt(TIMER_A1_BASE);

    write_74HC164(data_values[n]);
    write_74HC145(n);

    n++;

    if(n > 9)
    {
      n = 0;
    }
}
```

Inside timer interrupt subroutine, both logic ICs are updated. Firstly, the number to be displayed is sent to the 74HC164 IC and then the seven-segment display to show the number is updated by writing the 74HC145 IC. At every interrupt, one seven segment display is updated. There are 8 such displays and so it takes about 20 ms to update all of these displays. During this time the keypad is also scanned in the main loop. With different key presses, different LEDs light up.

```
switch(SW_in)
{
       case 1:
        {
            show_LEDs(1, 0, 0, 0);
            break;
        }
        case 2:
        {
             show_LEDs(0, 1, 0, 0);
             break;
        }
         case 3:
        {
             show_LEDs(0, 0, 1, 0);
             break;
        }
         case 4:
         {
             show_LEDs(0, 0, 0, 1);
             break;
         }
}

SW_in = 0x00;
```

Demo



Video demo: https://youtu.be/SXanUMbreIs

Single Pulse Width Module (PWM) – TA2

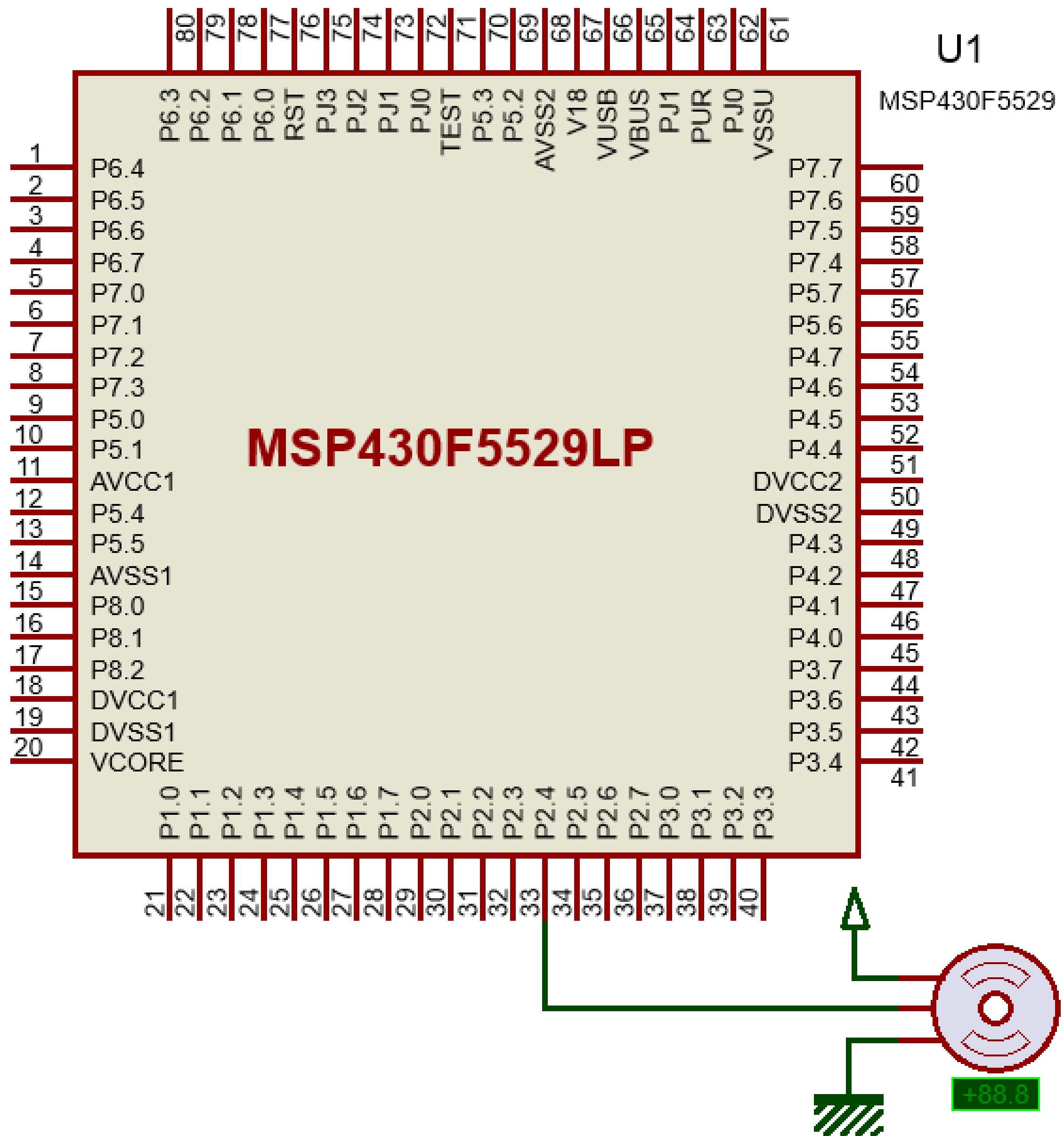
PWM is an output feature of timers' CC channels. Pulse width modulation (PWM) is needed in crafting switch-mode power supplies, inverters, motor controllers, servos and many other devices. PWM can also be used to generate analogue output similar to a Digital-to-Analogue Converter (DAC). In this example, we will see how we can use a single PWM channel to drive a servo motor.

Code Example

```
#include "driverlib.h"
#include "delay.h"


#define INC         1
#define DEC         0


void clock_init(void);
void GPIO_init(void);
void timer_T2A3_init(void);


void main(void)
{
    unsigned char dir = INC;
    unsigned int r = 0;

    WDT_A_hold(WDT_A_BASE);

    clock_init();
    GPIO_init();
    timer_T2A3_init();

    while(true)
    {
        if((r < 1500) && (dir == INC))
        {
            r++;
        }

        if((r == 1500) && (dir == INC))
        {
            dir = DEC;
        }

        if((r > 0) && (dir == DEC))
        {
            r--;
        }

        if((r == 0) && (dir == DEC))
        {
            dir = INC;
        }

        Timer_A_setCompareValue(TIMER_A2_BASE,
                                TIMER_A_CAPTURECOMPARE_REGISTER_1,
                                (1000 + r));

        delay_ms(2);
    };
```

```
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_MCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_2);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P2,
                                                GPIO_PIN4);
}


void timer_T2A3_init(void)
{
    Timer_A_outputPWMParam outputPWMParam = {0};

    outputPWMParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
    outputPWMParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_2;
    outputPWMParam.timerPeriod = 20000;
    outputPWMParam.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
    outputPWMParam.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
    outputPWMParam.dutyCycle = 0;

    Timer_A_outputPWM(TIMER_A2_BASE,
                      &outputPWMParam);

    Timer_A_setCompareValue(TIMER_A2_BASE,
                            TIMER_A_CAPTURECOMPARE_REGISTER_1,
                            1000);
}
```

Hardware Setup

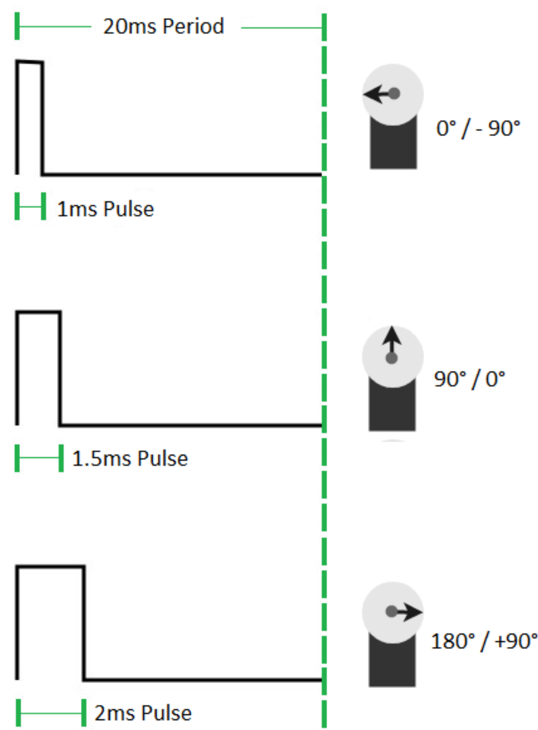


Explanation

As mentioned, the goal of this demo is to drive a regular servo motor using a single PWM pin of MSP430F5529. Firstly, it must be noted that PWM or Capture-Compare function is a secondary function of GPIO pins. Thus, to use a PWM pin, it must be declared as Output Peripheral Module Function pin.

```
GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P2, GPIO_PIN4);
```

Typical servos have a timing as shown below:



Note that every pulse has a period of 20ms. However, the high time varies from 5 - 10%, i.e. the duty cycle of the PWM should be between 5 – 10% in order to rotate the servo.

This time TA2 is used. We could have used Timer B as it is better suited for PWM. This won't matter much though for this example.

```
void timer_T2A3_init(void)
{
    Timer_A_outputPWMParam outputPWMParam = {0};

    outputPWMParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
    outputPWMParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_2;
    outputPWMParam.timerPeriod = 20000;
    outputPWMParam.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
    outputPWMParam.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
    outputPWMParam.dutyCycle = 0;

    Timer_A_outputPWM(TIMER_A2_BASE, &outputPWMParam);

    Timer_A_setCompareValue(TIMER_A2_BASE,TIMER_A_CAPTURECOMPARE_REGISTER_1, 1000);
}
```

SMCLK is again used to clock the timer but this time it is scaled by a factor of 2. SMCLK will thus have a frequency of 2MHz as it is feed with 4MHz XT2_CLK.

```
UCS_initClockSignal(UCS_SMCLK, UCS_XT2CLK_SELECT, UCS_CLOCK_DIVIDER_2);
```

Inside the timer, this incoming clock signal is further divided by 2 and so the timer will tick at every 1µs. Like up mode, the top value or max PWM duty cycle is set to 20000. This means that the period of the PWM will be 20000 µs or 20 ms. Initially, the duty cycle is set to 0ms and the output mode is set to reset-set mode. Different output modes of MSP430's PWM are shown below.

OUTMODx	Mode	Description
000	Output	The output signal OUTn is defined by the OUT bit. The OUTn signal updates immediately when OUT is updated.
001	Set	The output is set when the timer *counts* to the TAxCCRn value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer *counts* to the TAxCCRn value. It is reset when the timer *counts* to the TAxCCR0 value.
011	Set/Reset	The output is set when the timer *counts* to the TAxCCRn value. It is reset when the timer *counts* to the TAxCCR0 value.
100	Toggle	The output is toggled when the timer *counts* to the TAxCCRn value. The output period is double the timer period.
101	Reset	The output is reset when the timer *counts* to the TAxCCRn value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer *counts* to the TAxCCRn value. It is set when the timer *counts* to the TAxCCR0 value.
111	Reset/Set	The output is reset when the timer *counts* to the TAxCCRn value. It is set when the timer *counts* to the TAxCCR0 value.

After setting the PWM parameters, the timer and its PWM are started. Initially the pulse high time is set to 1ms, i.e. the servo will at 0° (-90°) position. We want the servo to rotate back and forth, i.e. from 0° (-90°) to 180° (+90°). To do this, we have to increase and decrease the pulse high time in the main loop.

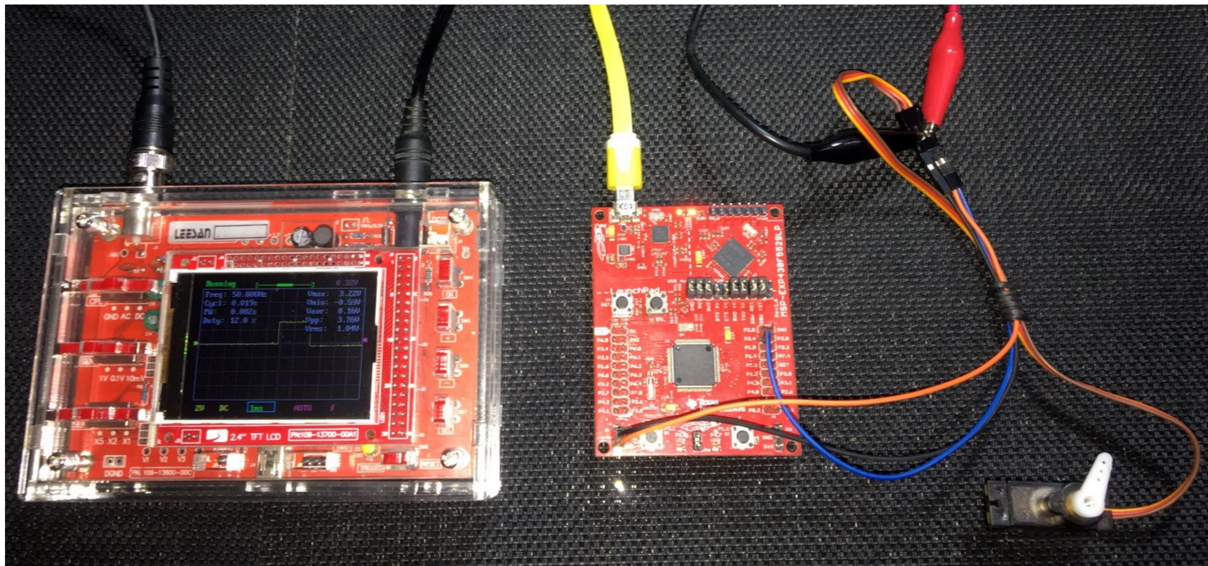
```
if((r < 1500) && (dir == INC))
{
   r++;
}
if((r == 1500) && (dir == INC))
{
   dir = DEC;
}
 if((r > 0) && (dir == DEC))
{
    r--;
}
if((r == 0) && (dir == DEC))
{
   dir = INC;
}
Timer_A_setCompareValue(TIMER_A2_BASE, TIMER_A_CAPTURECOMPARE_REGISTER_1, (1000 + r));

delay_ms(2);
```

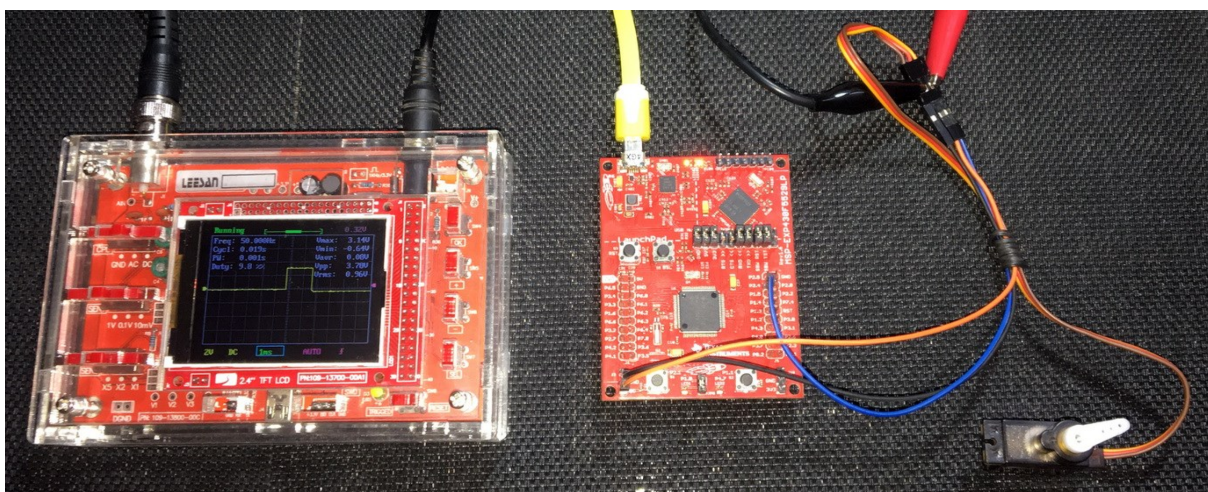
The line below is responsible of duty cycle variation:

```
Timer_A_setCompareValue(TIMER_A2_BASE, TIMER_A_CAPTURECOMPARE_REGISTER_1, (1000 + r));
```

It states which timer it is, which channel it is and the duty cycle of that channel.

Demo





Demo video: https://youtu.be/eDl7YJNfCO0

Multiple Pulse Width Module (PWM) – TA0

Single PWM is useful for simple purposes but in many applications, we need multiple PWMs. In case of inverters, MOSFET/IGBT drives and bridges, there is no alternative other than to use multi-channel PWM. In this example, we will see how we can use multiple PWM to drive an RGB LED.

Code Example

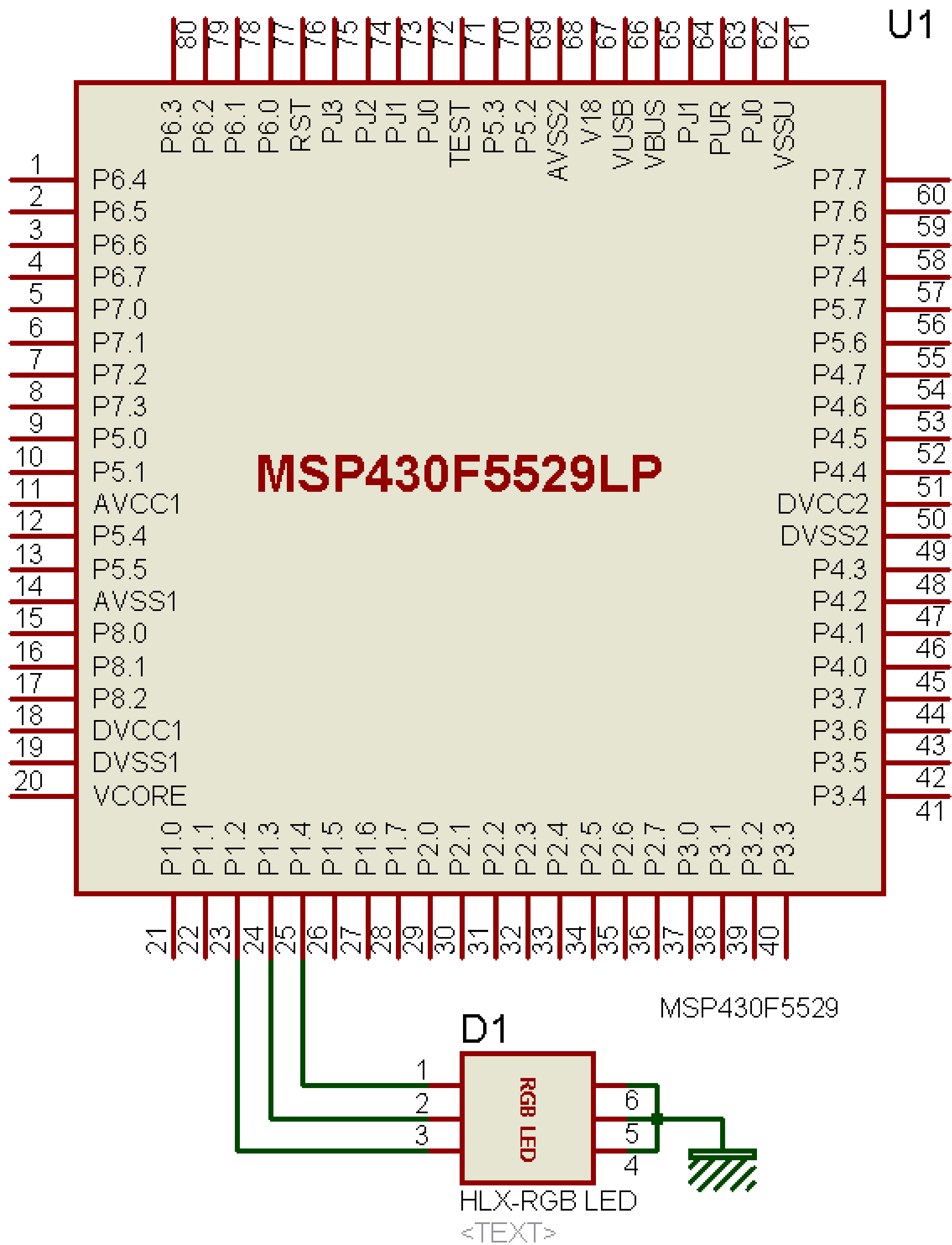
```
#include "driverlib.h"
#include "delay.h"


#define steps 32


void clock_init(void);
void GPIO_init(void);
void timer_T0A5_init(void);


void main(void)
{
    unsigned char i = 0x00;

    const unsigned int duty1_value[steps] = {0,
                                             6424,
                                             12786,
                                             19026,
                                             25082,
                                             30896,
                                             36413,
                                             41579,
                                             46344,
                                             50664,
                                             54494,
                                             57801,
                                             60550,
                                             62716,
                                             64278,
                                             65220,
                                             65534,
                                             65218,
                                             64272,
                                             62708,
                                             60540,
                                             57788,
                                             54480,
                                             50647,
                                             46326,
                                             41558,
                                             36391,
                                             30873,
                                             25057,
                                             19000,
                                             12760,
                                             6397};

    const unsigned int duty2_value[steps] = {57801,
                                             60550,
                                             62716,
                                             64278,
```

```
                                            65220,
                                            65534,
                                            65218,
                                            64272,
                                            62708,
                                            60540,
                                            57788,
                                            54480,
                                            50647,
                                            46326,
                                            41558,
                                            36391,
                                            30873,
                                            25057,
                                            19000,
                                            12760,
                                            6397,
                                            0,
                                            6424,
                                            12786,
                                            19026,
                                            25082,
                                             30896,
                                             36413,
                                             41579,
                                             46344,
                                            50664,
                                            54494};

    const unsigned int duty3_value[steps] = {57788,
                                            54480,
                                            50647,
                                            46326,
                                            41558,
                                            36391,
                                            30873,
                                            25057,
                                            19000,
                                            12760,
                                            6397,
                                            0,
                                            6424,
                                            12786,
                                            19026,
                                            25082,
                                             30896,
                                             36413,
                                             41579,
                                             46344,
                                            50664,
                                            54494,
                                            57801,
                                            60550,
                                            62716,
                                            64278,
                                            65220,
                                            65534,
                                            65218,
                                            64272,
                                            62708,
                                            60540};

    WDT_A_hold(WDT_A_BASE);

    clock_init();
    GPIO_init();
```

```
    timer_T0A5_init();

    while(true)
    {
        for(i = 0; i < steps; i++)
        {
            Timer_A_setCompareValue(TIMER_A0_BASE,
                                    TIMER_A_CAPTURECOMPARE_REGISTER_1,
                                    duty1_value[i]);

            Timer_A_setCompareValue(TIMER_A0_BASE,
                                    TIMER_A_CAPTURECOMPARE_REGISTER_2,
                                    duty2_value[i]);

            Timer_A_setCompareValue(TIMER_A0_BASE,
                                    TIMER_A_CAPTURECOMPARE_REGISTER_3,
                                    duty3_value[i]);

            delay_ms(250);
        }
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_MCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P1,
                                                (GPIO_PIN2 + GPIO_PIN3 + GPIO_PIN4));
}


void timer_T0A5_init(void)
{
```

```
    Timer_A_initCompareModeParam CompareModeParam1 = {0};
    Timer_A_initCompareModeParam CompareModeParam2 = {0};
    Timer_A_initCompareModeParam CompareModeParam3 = {0};

    Timer_A_initContinuousModeParam ContinuousModeParam = {0};

    ContinuousModeParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
    ContinuousModeParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_1;
    ContinuousModeParam.timerClear = TIMER_A_DO_CLEAR;
    ContinuousModeParam.timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_DISABLE;
    ContinuousModeParam.startTimer = false;

    CompareModeParam1.compareInterruptEnable = TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE;
    CompareModeParam1.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
    CompareModeParam1.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
    CompareModeParam1.compareValue = 0;

    CompareModeParam2.compareInterruptEnable = TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE;
    CompareModeParam2.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
    CompareModeParam2.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_2;
    CompareModeParam2.compareValue = 0;

    CompareModeParam3.compareInterruptEnable = TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE;
    CompareModeParam3.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
    CompareModeParam3.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_3;
    CompareModeParam3.compareValue = 0;

    Timer_A_initCompareMode(TIMER_A0_BASE,
                            &CompareModeParam1);

    Timer_A_initCompareMode(TIMER_A0_BASE,
                            &CompareModeParam2);

    Timer_A_initCompareMode(TIMER_A0_BASE,
                            &CompareModeParam3);

    Timer_A_initContinuousMode(TIMER_A0_BASE,
                               &ContinuousModeParam);

    Timer_A_startCounter(TIMER_A0_BASE,
                         TIMER_A_CONTINUOUS_MODE);
}
```

Hardware Setup



Explanation

Multi-PWM example here uses the same concepts as in the single PWM example. This time timer TA0 is used and the following pins are set for PWM output:

```
GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P1,
                                            (GPIO_PIN2 + GPIO_PIN3 + GPIO_PIN4));
```

Note that this timer has 5 CC channels associated with it but we are using only three of them since we have connected an RGB LED to these pins.

4MHz XT2CLK source feeds SMCLK which in turn is used to run timer TA0. Thus, timer TA0 is operating at 4MHz speed.

```
UCS_initClockSignal(UCS_SMCLK, UCS_XT2CLK_SELECT, UCS_CLOCK_DIVIDER_1);
```

Timer PWM setup is not very much different. The only differences are timer mode, the number of channels and their individual settings.

```
void timer_T0A5_init(void)
{
    Timer_A_initCompareModeParam CompareModeParam1 = {0};
    Timer_A_initCompareModeParam CompareModeParam2 = {0};
    Timer_A_initCompareModeParam CompareModeParam3 = {0};

    Timer_A_initContinuousModeParam ContinuousModeParam = {0};

    ContinuousModeParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
    ContinuousModeParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_1;
    ContinuousModeParam.timerClear = TIMER_A_DO_CLEAR;
    ContinuousModeParam.timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_DISABLE;
    ContinuousModeParam.startTimer = false;

    CompareModeParam1.compareInterruptEnable = TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE;
    CompareModeParam1.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
    CompareModeParam1.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
    CompareModeParam1.compareValue = 0;

    CompareModeParam2.compareInterruptEnable = TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE;
    CompareModeParam2.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
    CompareModeParam2.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_2;
    CompareModeParam2.compareValue = 0;

    CompareModeParam3.compareInterruptEnable = TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE;
    CompareModeParam3.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
    CompareModeParam3.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_3;
    CompareModeParam3.compareValue = 0;

    Timer_A_initCompareMode(TIMER_A0_BASE, &CompareModeParam1);

    Timer_A_initCompareMode(TIMER_A0_BASE, &CompareModeParam2);

    Timer_A_initCompareMode(TIMER_A0_BASE, &CompareModeParam3);

    Timer_A_initContinuousMode(TIMER_A0_BASE, &ContinuousModeParam);

    Timer_A_startCounter(TIMER_A0_BASE, TIMER_A_CONTINUOUS_MODE);
}
```

unlike in the previous PWM example timer TA0 is set for continuous mode of operation and so the top PWM count value is 65535. Thus, the period of the PWMs is about 16ms. No interrupt is used. We just have to setup each channel individually and then run the timer in continuous mode.

In the main loop, PWMs of each channel are altered and the effect is visible in the form of RGB LED's changing colour.

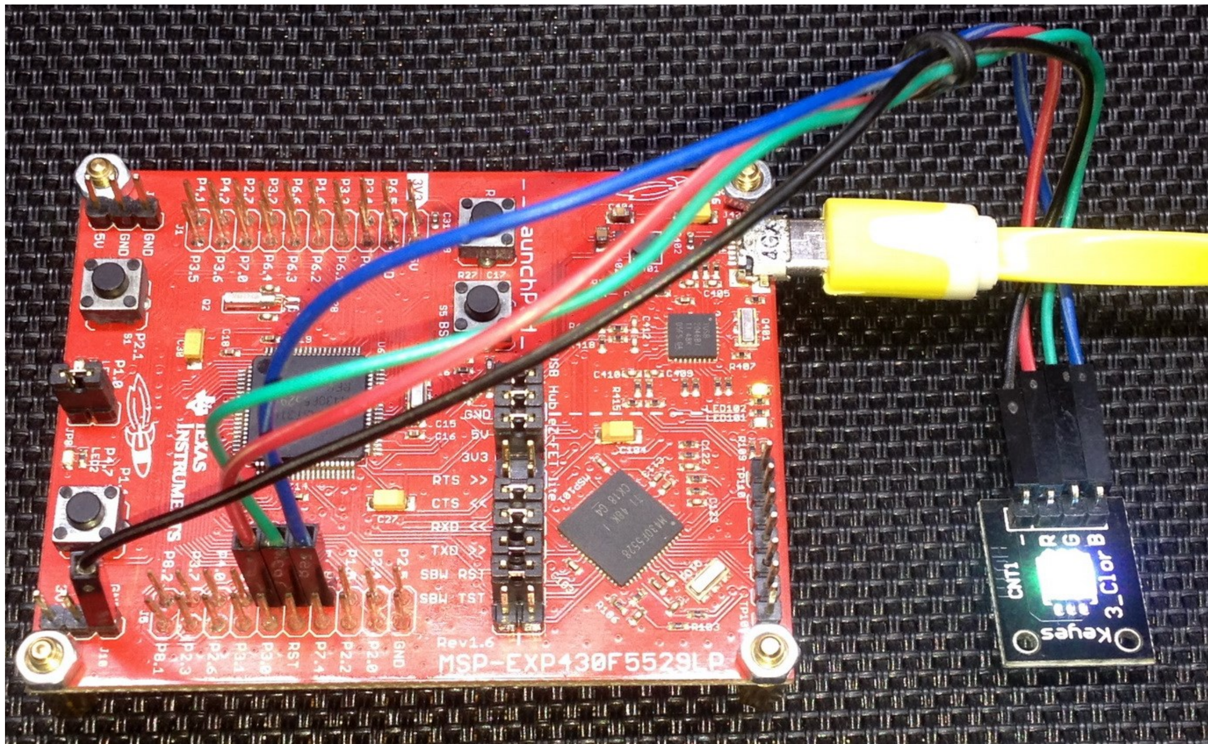
```
for(i = 0; i < steps; i++)
{
     Timer_A_setCompareValue(TIMER_A0_BASE,
                             TIMER_A_CAPTURECOMPARE_REGISTER_1,
                             duty1_value[i]);

     Timer_A_setCompareValue(TIMER_A0_BASE,
                             TIMER_A_CAPTURECOMPARE_REGISTER_2,
                             duty2_value[i]);

     Timer_A_setCompareValue(TIMER_A0_BASE,
                             TIMER_A_CAPTURECOMPARE_REGISTER_3,
                             duty3_value[i]);

     delay_ms(250);
}
```

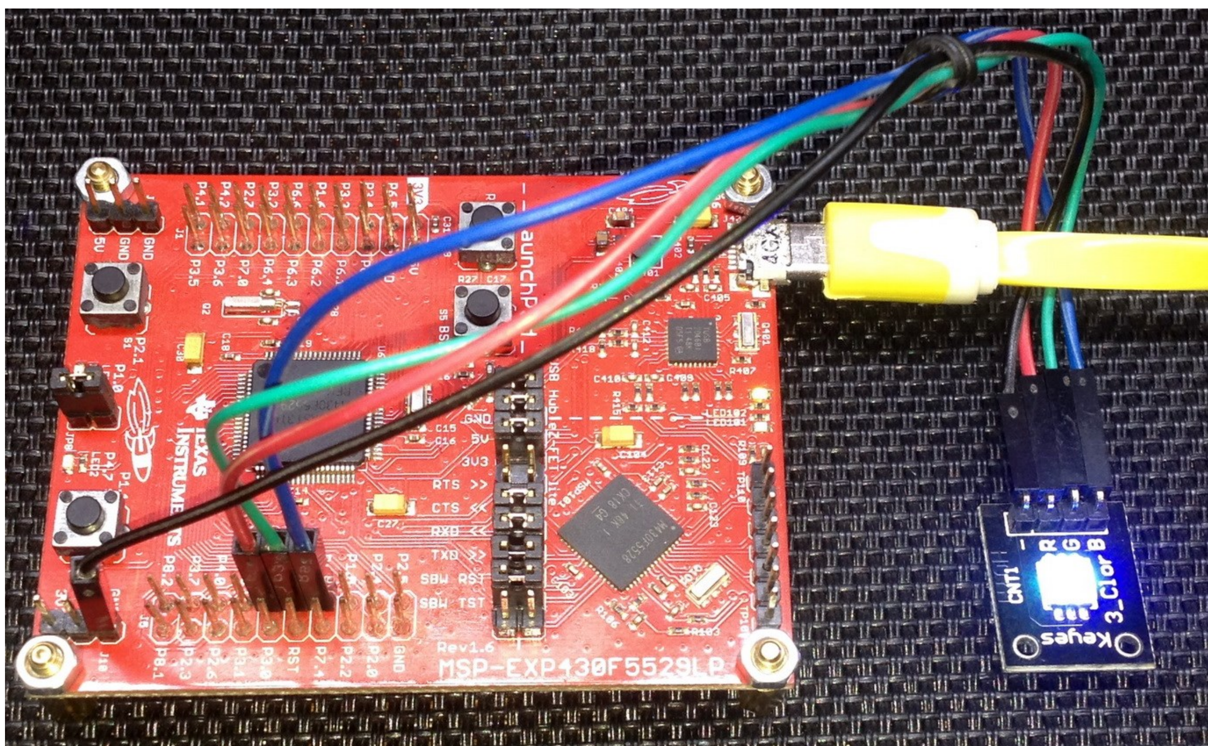
The duty cycle of each PWM channel is coded to in a half-sine wave pattern but each of them has a different phase shift from the other.

```
const unsigned int duty3_value[steps] = {57788,
                                         54480,
                                         50647,
                                         46326,
                                         41558,
                                         36391,
                                         30873,
                                         25057,
                                         19000,
                                         12760,
                                         6397,
                                         0,
                                         6424,
                                         12786,
                                         19026,
                                         25082,
                                          30896,
                                          36413,
                                          41579,
                                          46344,
                                         50664,
                                         54494,
                                         57801,
                                         60550,
                                         62716,
                                         64278,
                                         65220,
                                         65534,
                                         65218,
                                         64272,
                                         62708,
                                         60540};
```

When combined with the three LEDs of the RGB, this creates a beautiful rhythmic colour pattern.

Demo





Video demo: https://youtu.be/aJrqjpIhGx8

Timer as Counter – TA2

When the clock source of a timer is both regular and repetitive, the timer can be used for tracking/keeping time. When the same timer is fed with an irregular source, the timer acts like a counter. In both cases, pulses are counted – one in synchronous while the other is asynchronous. We can make software-based counters by incrementing/decrementing variables. However, we can do this on hardware level if we use timers as counters and feed them with external inputs.

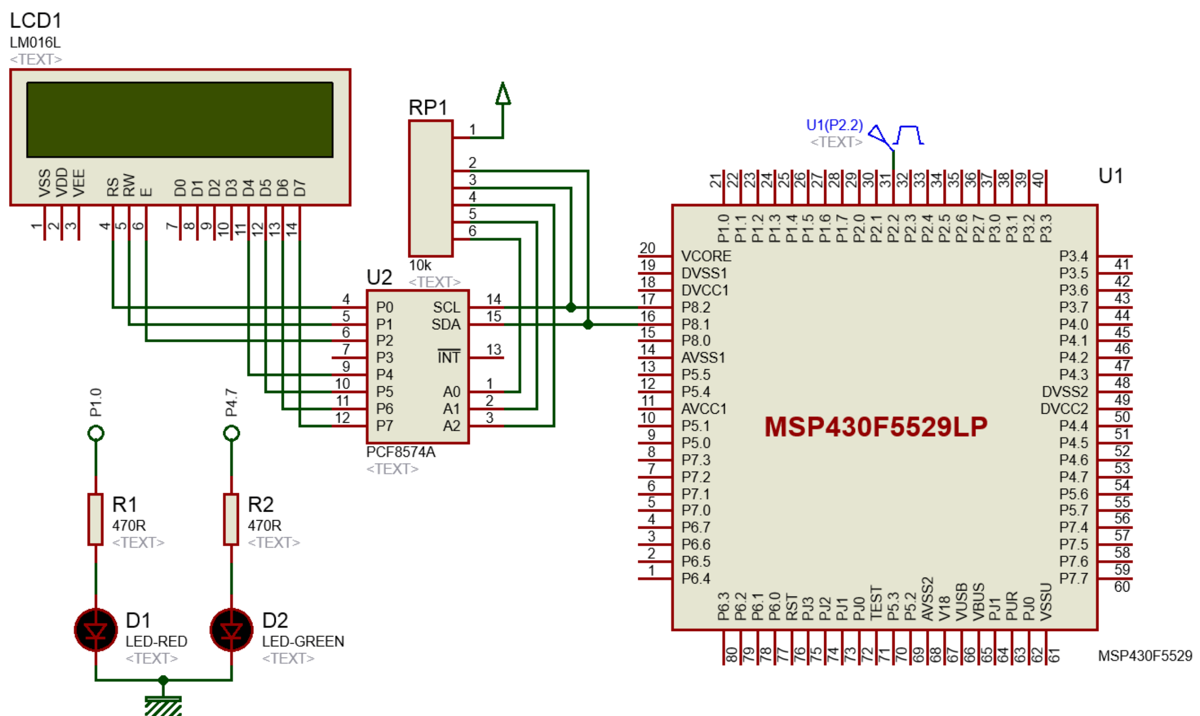
Code Example

```
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"
#include "lcd_print.h"


void clock_init(void);
void GPIO_init(void);
void timer_T2A3_init(void);


void main(void)
{
    signed int value = 0;

    WDT_A_hold(WDT_A_BASE);

    clock_init();
    GPIO_init();
    timer_T2A3_init();
    LCD_init();

    LCD_clear_home();

    LCD_goto(1, 0);
    LCD_putstr("TA2 as Counter");

    LCD_goto(0, 1);
    LCD_putstr("Count:");


    while(true)
    {
        value = Timer_A_getCounterValue(TIMER_A2_BASE);
        print_C(12, 1, value);

        if((value > 20) && (value < 40))
        {
            GPIO_setOutputHighOnPin(GPIO_PORT_P1,
                                    GPIO_PIN0);

            GPIO_setOutputLowOnPin(GPIO_PORT_P4,
                                   GPIO_PIN7);
        }

        else if((value > 40) && (value < 60))
        {
            GPIO_setOutputHighOnPin(GPIO_PORT_P4,
                                    GPIO_PIN7);
```

```
            GPIO_setOutputLowOnPin(GPIO_PORT_P1,
                                   GPIO_PIN0);
        }

        else if((value > 60) && (value < 120))
        {
            GPIO_setOutputHighOnPin(GPIO_PORT_P1,
                                    GPIO_PIN0);

            GPIO_setOutputHighOnPin(GPIO_PORT_P4,
                                    GPIO_PIN7);
        }

        else
        {
            GPIO_setOutputLowOnPin(GPIO_PORT_P1,
                                   GPIO_PIN0);

            GPIO_setOutputLowOnPin(GPIO_PORT_P4,
                                   GPIO_PIN7);
        }
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_MCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P2,
                                               GPIO_PIN2);

    GPIO_setAsOutputPin(GPIO_PORT_P1,
                        GPIO_PIN0);
```

```
    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);
}


void timer_T2A3_init(void)
{
    Timer_A_initUpModeParam UpModeParam = {0};

    UpModeParam.clockSource = TIMER_A_CLOCKSOURCE_EXTERNAL_TXCLK;
    UpModeParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_1;
    UpModeParam.timerPeriod = 120;
    UpModeParam.timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_DISABLE;
    UpModeParam.captureCompareInterruptEnable_CCR0_CCIE =
TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE;
    UpModeParam.timerClear = TIMER_A_SKIP_CLEAR;
    UpModeParam.startTimer = true;

    Timer_A_stop(TIMER_A2_BASE);

    Timer_A_clearTimerInterrupt(TIMER_A2_BASE);

    Timer_A_initUpMode(TIMER_A2_BASE,
                       &UpModeParam);
}
```

Hardware Setup



Explanation

Here timer T2A3 is used to count incoming pulses. Since counting pulses involves timer's external input pin, this input pin must be declared as an Input Peripheral Module Function pin.

```
GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P2, GPIO_PIN2);
```

Timer TA2 is setup as follows:

```
void timer_T2A3_init(void)
{
    Timer_A_initUpModeParam UpModeParam = {0};

    UpModeParam.clockSource = TIMER_A_CLOCKSOURCE_EXTERNAL_TXCLK;
    UpModeParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_1;
    UpModeParam.timerPeriod = 120;
    UpModeParam.timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_DISABLE;
    UpModeParam.captureCompareInterruptEnable_CCR0_CCIE =
TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE;
    UpModeParam.timerClear = TIMER_A_SKIP_CLEAR;
    UpModeParam.startTimer = true;

    Timer_A_stop(TIMER_A2_BASE);

    Timer_A_clearTimerInterrupt(TIMER_A2_BASE);

    Timer_A_initUpMode(TIMER_A2_BASE, &UpModeParam);
}
```

Clock source is the very first thing to note in this example. ACLK or SMCLK is not used. Externally clock pulses are applied via timer input pin. Note however timer input pin is also not a CC channel pin.

Up mode counting is used with a top value of 120 and so the timer will count from 0 to 120 and then roll over.

Interrupts are avoided and so polling method is used.

Initially, the timer is kept in halt state and timer settings are not cleared because no setting was applied beforehand.

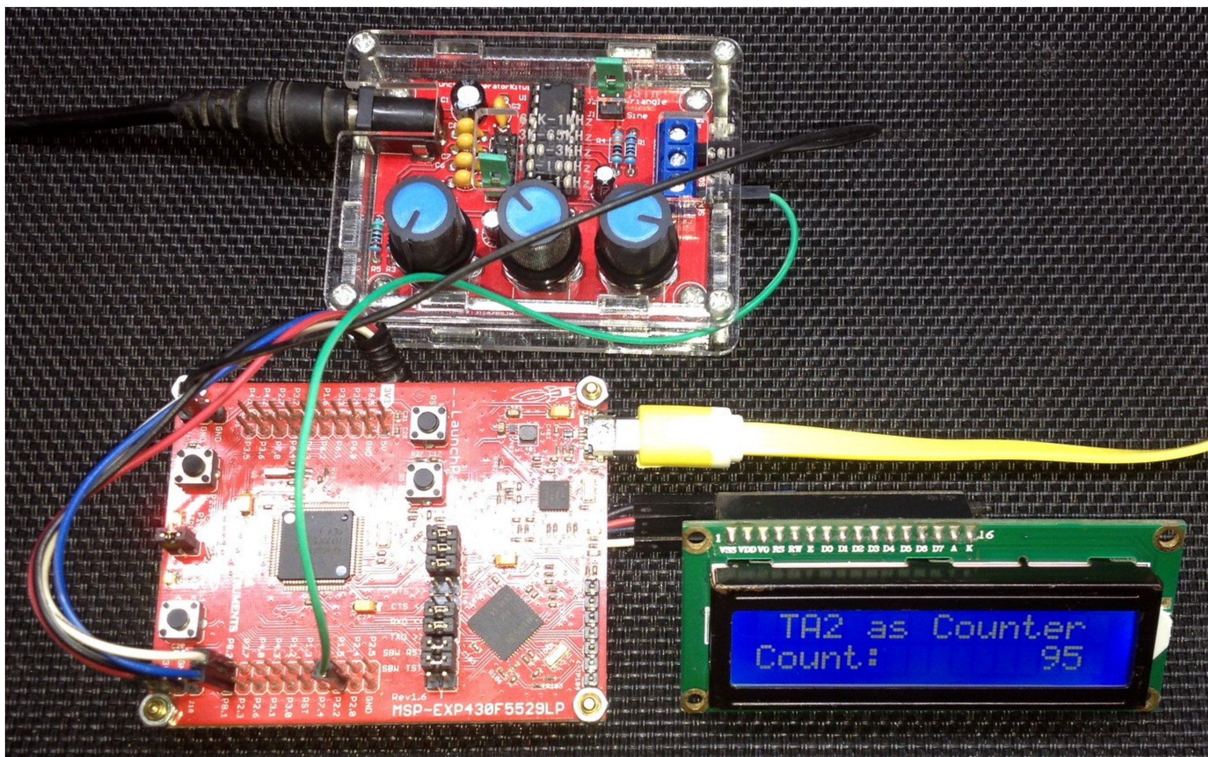
After setting up the timer, the timer is started but counting doesn't start until there is a logic transition in the timer's external input pin. Each incoming pulse increases the timer's count.

In the main loop, timer's count is checked or polled on each loop passes. The value of timer count is displayed on an LCD and onboard LEDs are flashed according to count values.

```
value = Timer_A_getCounterValue(TIMER_A2_BASE);
print_C(12, 1, value);

if((value > 20) && (value < 40))
{
    GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);

    GPIO_setOutputLowOnPin(GPIO_PORT_P4, GPIO_PIN7);
}
```

```
else if((value > 40) && (value < 60))
{
    GPIO_setOutputHighOnPin(GPIO_PORT_P4, GPIO_PIN7);

    GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
}

else if((value > 60) && (value < 120))
{
    GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);

    GPIO_setOutputHighOnPin(GPIO_PORT_P4, GPIO_PIN7);
}

else
{
    GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);

    GPIO_setOutputLowOnPin(GPIO_PORT_P4, GPIO_PIN7);
}
```

Demo



Video demo: https://youtu.be/ZHtPsZM9G2Y

Timer Input Capture Mode – TA0

Just like PWM, timer input capture is another important feature of a timer. This is an input-side feature unlike PWM. Input capture is needed for measurement of pulse widths, frequency and time period of an incoming waveform.

Code Example

```
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"
#include "lcd_print.h"


unsigned int pulse_ticks = 0;
unsigned int start_time = 0;
unsigned int end_time = 0;


void clock_init(void);
void GPIO_init(void);
void timer_T0A5_init(void);
void timer_T2A3_init(void);


#pragma vector=TIMER0_A1_VECTOR
__interrupt void TIMER0_A1_ISR(void)
{
  switch(__even_in_range(TA0IV, 10))
  {
       case 0x00: break;              // None
       case 0x02:                     // CCR1 IFG
       {
           end_time = Timer_A_getCaptureCompareCount(TIMER_A0_BASE,
                                                     TIMER_A_CAPTURECOMPARE_REGISTER_1);

           pulse_ticks = (end_time - start_time);

           start_time = end_time;

           Timer_A_clearCaptureCompareInterrupt(TIMER_A0_BASE,
                                                TIMER_A_CAPTURECOMPARE_REGISTER_1);

           break;
       }
       case 0x04: break;              // CCR2 IFG
       case 0x06: break;              // CCR3 IFG
       case 0x08: break;              // CCR4 IFG
       case 0x0A: break;              // CCR5 IFG
       case 0x0C: break;              // CCR6 IFG
       case 0x0E:                     // TA0IFG
       {
           GPIO_toggleOutputOnPin(GPIO_PORT_P1,
                                  GPIO_PIN0);

           break;
       }
       default:   _never_executed();
  }
}
```

```c
void main(void)
{
    unsigned char i = 0;
    unsigned char settings_changed = false;
    unsigned long timer_clock_frequency = 0;

    WDT_A_hold(WDT_A_BASE);

    clock_init();
    GPIO_init();
    timer_T0A5_init();
    timer_T2A3_init();

    LCD_init();
    LCD_clear_home();

    LCD_goto(0, 0);
    LCD_putstr("PWM./Hz:");

    LCD_goto(0, 1);
    LCD_putstr("Cap./Hz:");

    timer_clock_frequency = UCS_getSMCLK();

    while(true)
    {
        if(GPIO_getInputPinValue(GPIO_PORT_P2,
                                 GPIO_PIN1) == false)
        {
            while(GPIO_getInputPinValue(GPIO_PORT_P2,
                                        GPIO_PIN1) == false);

            i++;

            GPIO_setOutputHighOnPin(GPIO_PORT_P4,
                                    GPIO_PIN7);

            delay_ms(100);

            GPIO_setOutputLowOnPin(GPIO_PORT_P4,
                                   GPIO_PIN7);

            if(i > 5)
            {
                i = 0;
            }

            settings_changed = false;

        }

        switch(settings_changed)
        {
            case true:
            {
                print_I(8, 1, (timer_clock_frequency / pulse_ticks));
                delay_ms(100);
                break;
            }

            default:
            {
                switch(i)
                {
                    case 1:
```

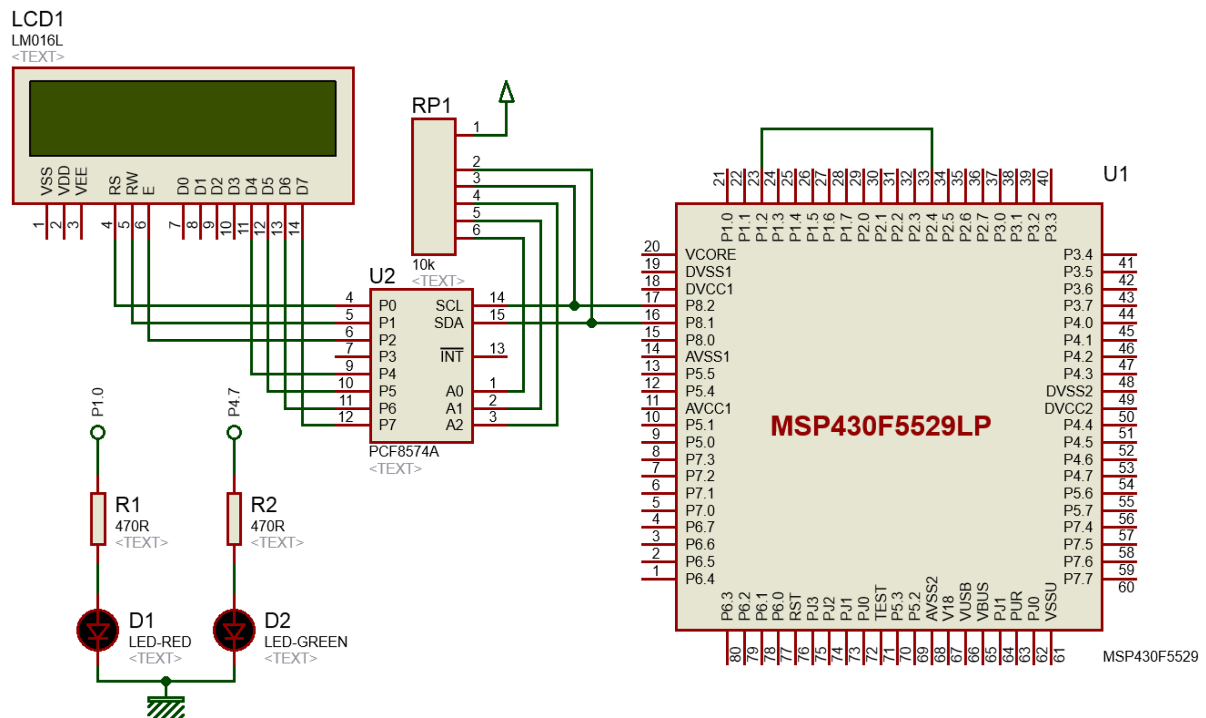
```
                {
                    LCD_goto(9, 0);
                    LCD_putstr("200  ");

                    Timer_A_outputPWMParam outputPWMParam = {0};

                    outputPWMParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
                    outputPWMParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_4;
                    outputPWMParam.timerPeriod = 5000;
                    outputPWMParam.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
                    outputPWMParam.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
                    outputPWMParam.dutyCycle = 0;

                    Timer_A_outputPWM(TIMER_A2_BASE,
                                      &outputPWMParam);

                    Timer_A_setCompareValue(TIMER_A2_BASE,
                                            TIMER_A_CAPTURECOMPARE_REGISTER_1,
                                            2000);

                    break;
                }

                case 2:
                {
                    LCD_goto(9, 0);
                    LCD_putstr("125  ");

                    Timer_A_outputPWMParam outputPWMParam = {0};

                    outputPWMParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
                    outputPWMParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_4;
                    outputPWMParam.timerPeriod = 8000;
                    outputPWMParam.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
                    outputPWMParam.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
                    outputPWMParam.dutyCycle = 0;

                    Timer_A_outputPWM(TIMER_A2_BASE,
                                      &outputPWMParam);

                    Timer_A_setCompareValue(TIMER_A2_BASE,
                                            TIMER_A_CAPTURECOMPARE_REGISTER_1,
                                            4000);

                    break;
                }

                case 3:
                {
                    LCD_goto(9, 0);
                    LCD_putstr("1000 ");

                    Timer_A_outputPWMParam outputPWMParam = {0};

                    outputPWMParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
                    outputPWMParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_4;
                    outputPWMParam.timerPeriod = 1000;
                    outputPWMParam.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
                    outputPWMParam.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
                    outputPWMParam.dutyCycle = 0;

                    Timer_A_outputPWM(TIMER_A2_BASE,
                                      &outputPWMParam);

                    Timer_A_setCompareValue(TIMER_A2_BASE,
                                            TIMER_A_CAPTURECOMPARE_REGISTER_1,
```

```
                                600);

                break;
            }

            case 4:
            {
                LCD_goto(9, 0);
                LCD_putstr("8000 ");

                Timer_A_outputPWMParam outputPWMParam = {0};

                outputPWMParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
                outputPWMParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_4;
                outputPWMParam.timerPeriod = 125;
                outputPWMParam.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
                outputPWMParam.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
                outputPWMParam.dutyCycle = 0;

                Timer_A_outputPWM(TIMER_A2_BASE,
                                  &outputPWMParam);

                Timer_A_setCompareValue(TIMER_A2_BASE,
                                        TIMER_A_CAPTURECOMPARE_REGISTER_1,
                                        60);

                break;
            }

            case 5:
            {
                LCD_goto(9, 0);
                LCD_putstr("2000 ");

                Timer_A_outputPWMParam outputPWMParam = {0};

                outputPWMParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
                outputPWMParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_4;
                outputPWMParam.timerPeriod = 500;
                outputPWMParam.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
                outputPWMParam.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
                outputPWMParam.dutyCycle = 0;

                Timer_A_outputPWM(TIMER_A2_BASE,
                                  &outputPWMParam);

                Timer_A_setCompareValue(TIMER_A2_BASE,
                                        TIMER_A_CAPTURECOMPARE_REGISTER_1,
                                        250);

                break;
            }

            default:
            {

                LCD_goto(9, 0);
                LCD_putstr("333.3");

                Timer_A_outputPWMParam outputPWMParam = {0};

                outputPWMParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
                outputPWMParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_4;
                outputPWMParam.timerPeriod = 3000;
                outputPWMParam.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
                outputPWMParam.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
```

```
                        outputPWMParam.dutyCycle = 0;

                        Timer_A_outputPWM(TIMER_A2_BASE,
                                          &outputPWMParam);

                        Timer_A_setCompareValue(TIMER_A2_BASE,
                                                TIMER_A_CAPTURECOMPARE_REGISTER_1,
                                                1500);

                        break;
                    }

                }

                settings_changed = true;

                break;
            }
        }
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_MCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P2,
                                         GPIO_PIN1);

    GPIO_setAsOutputPin(GPIO_PORT_P1,
                        GPIO_PIN0);

    GPIO_setDriveStrength(GPIO_PORT_P1,
                          GPIO_PIN0,
```

```
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setDriveStrength(GPIO_PORT_P4,
                          GPIO_PIN7,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P1,
                                               GPIO_PIN2);

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P2,
                                                GPIO_PIN4);
}


void timer_T0A5_init(void)
{
    Timer_A_initCaptureModeParam CaptureModeParam = {0};
    Timer_A_initContinuousModeParam ContinuousModeParam = {0};

    Timer_A_stop(TIMER_A0_BASE);
    Timer_A_clearTimerInterrupt(TIMER_A0_BASE);

    ContinuousModeParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
    ContinuousModeParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_1;
    ContinuousModeParam.timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_ENABLE;
    ContinuousModeParam.timerClear = TIMER_A_SKIP_CLEAR;
    ContinuousModeParam.startTimer = true;

    CaptureModeParam.captureRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
    CaptureModeParam.captureMode = TIMER_A_CAPTUREMODE_RISING_EDGE;
    CaptureModeParam.captureInputSelect = TIMER_A_CAPTURE_INPUTSELECT_CCIxA;
    CaptureModeParam.synchronizeCaptureSource = TIMER_A_CAPTURE_ASYNCHRONOUS;
    CaptureModeParam.captureInterruptEnable = TIMER_A_CAPTURECOMPARE_INTERRUPT_ENABLE;

    Timer_A_initContinuousMode(TIMER_A0_BASE,
                               &ContinuousModeParam);

    Timer_A_initCaptureMode(TIMER_A0_BASE,
                            &CaptureModeParam);

    __enable_interrupt();
}


void timer_T2A3_init(void)
{
    Timer_A_outputPWMParam outputPWMParam = {0};

    outputPWMParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
    outputPWMParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_4;
    outputPWMParam.timerPeriod = 2000;
    outputPWMParam.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
    outputPWMParam.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
    outputPWMParam.dutyCycle = 0;

    Timer_A_outputPWM(TIMER_A2_BASE,
                      &outputPWMParam);

    Timer_A_setCompareValue(TIMER_A2_BASE,
                            TIMER_A_CAPTURECOMPARE_REGISTER_1,
                            1000);
}
```

Hardware Setup



Explanation

In this example, two timers are used. One is set to provide variable frequency PWM while the other is set to measure PWM frequency.

First, let's see how the PWM is configured. Timer TA2 is used to provide PWM output

```
void clock_init(void)
{
....
    UCS_initClockSignal(UCS_SMCLK, UCS_XT2CLK_SELECT, UCS_CLOCK_DIVIDER_1);
....
}


void GPIO_init(void)
{
....
    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P2, GPIO_PIN4);
}


void timer_T2A3_init(void)
{
    Timer_A_outputPWMParam outputPWMParam = {0};

    outputPWMParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
    outputPWMParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_4;
    outputPWMParam.timerPeriod = 2000;
    outputPWMParam.compareRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
    outputPWMParam.compareOutputMode = TIMER_A_OUTPUTMODE_RESET_SET;
    outputPWMParam.dutyCycle = 0;
```

```
    Timer_A_outputPWM(TIMER_A2_BASE,
                      &outputPWMParam);

    Timer_A_setCompareValue(TIMER_A2_BASE,
                            TIMER_A_CAPTURECOMPARE_REGISTER_1,
                            1000);
}
```

One CC channel of timer T2A3 is used to generate a PWM output of 500Hz with these settings. I believe, by now, the code is not difficult to understand.

On P2.1 button press, the time period and duty cycle of this timer are altered and square waves of different frequencies are generated.

Now let's see how the capture timer is setup. Capture hardware is made with timer TA0. CC channel 1 is used and therefore P1.2's secondary function is enabled.

```
void clock_init(void)
{
....
    UCS_initClockSignal(UCS_SMCLK, UCS_XT2CLK_SELECT, UCS_CLOCK_DIVIDER_1);
....
}


void GPIO_init(void)
{
....
    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P1, GPIO_PIN2);
....
}


void timer_T0A5_init(void)
{
    Timer_A_initCaptureModeParam CaptureModeParam = {0};
    Timer_A_initContinuousModeParam ContinuousModeParam = {0};

    Timer_A_stop(TIMER_A0_BASE);
    Timer_A_clearTimerInterrupt(TIMER_A0_BASE);

    ContinuousModeParam.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
    ContinuousModeParam.clockSourceDivider = TIMER_A_CLOCKSOURCE_DIVIDER_1;
    ContinuousModeParam.timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_ENABLE;
    ContinuousModeParam.timerClear = TIMER_A_SKIP_CLEAR;
    ContinuousModeParam.startTimer = true;

    CaptureModeParam.captureRegister = TIMER_A_CAPTURECOMPARE_REGISTER_1;
    CaptureModeParam.captureMode = TIMER_A_CAPTUREMODE_RISING_EDGE;
    CaptureModeParam.captureInputSelect = TIMER_A_CAPTURE_INPUTSELECT_CCIxA;
    CaptureModeParam.synchronizeCaptureSource = TIMER_A_CAPTURE_ASYNCHRONOUS;
    CaptureModeParam.captureInterruptEnable = TIMER_A_CAPTURECOMPARE_INTERRUPT_ENABLE;

    Timer_A_initContinuousMode(TIMER_A0_BASE, &ContinuousModeParam);

    Timer_A_initCaptureMode(TIMER_A0_BASE, &CaptureModeParam);

    __enable_interrupt();
}
```

Capture hardware consists of two parts – CC input capture and continuous mode timer.

Firstly, the continuous mode timer is setup. This timer is fed with 4MHz SMCLK. It ticks at every 250ns and approximately overflows every 16ms.

Secondly, the capture part is setup using CC channel 1. It is set to look for rising edges of incoming waveform.

In the interrupt, respective flags are checked. Note both flags are under same interrupt vector. When timer overflow occurs, P1.0 LED is only toggled. However, the interesting part happens inside the CCR1 flag. It is at this stage, time period of capture waveform is captured and calculated.

```
#pragma vector=TIMER0_A1_VECTOR
__interrupt void TIMER0_A1_ISR(void)
{
  switch(__even_in_range(TA0IV, 10))
  {
       case 0x00: break;             // None
       case 0x02:                    // CCR1 IFG
       {
           end_time = Timer_A_getCaptureCompareCount(TIMER_A0_BASE,
TIMER_A_CAPTURECOMPARE_REGISTER_1);

           pulse_ticks = (end_time - start_time);

           start_time = end_time;

           Timer_A_clearCaptureCompareInterrupt(TIMER_A0_BASE,
TIMER_A_CAPTURECOMPARE_REGISTER_1);

           break;
       }
       case 0x04: break;             // CCR2 IFG
       case 0x06: break;             // CCR3 IFG
       case 0x08: break;             // CCR4 IFG
       case 0x0A: break;             // CCR5 IFG
       case 0x0C: break;             // CCR6 IFG
       case 0x0E:                    // TA0IFG
       {
           GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);

           break;
       }
       default:   _never_executed();
  }
}
```

When a rising-edge is detected, timer's current count is immediately stored. When another rising-edge is detected, the timer count of that instance is also stored. Subtracting these two counts results in a count difference. Since we know timer's tick period, we can use the count difference to compute period/frequency of the incoming waveform.

SMCLK's frequency is the timer's operating speed and it is found out by the code as shown below:

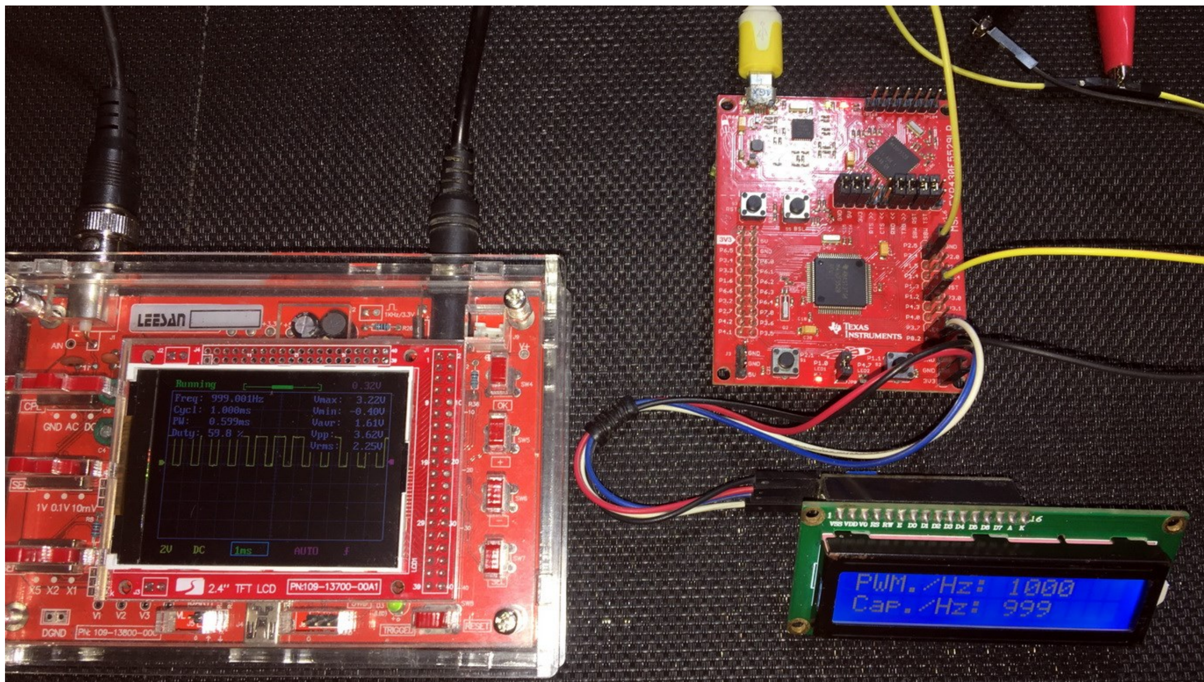
```
timer_clock_frequency = UCS_getSMCLK();
```

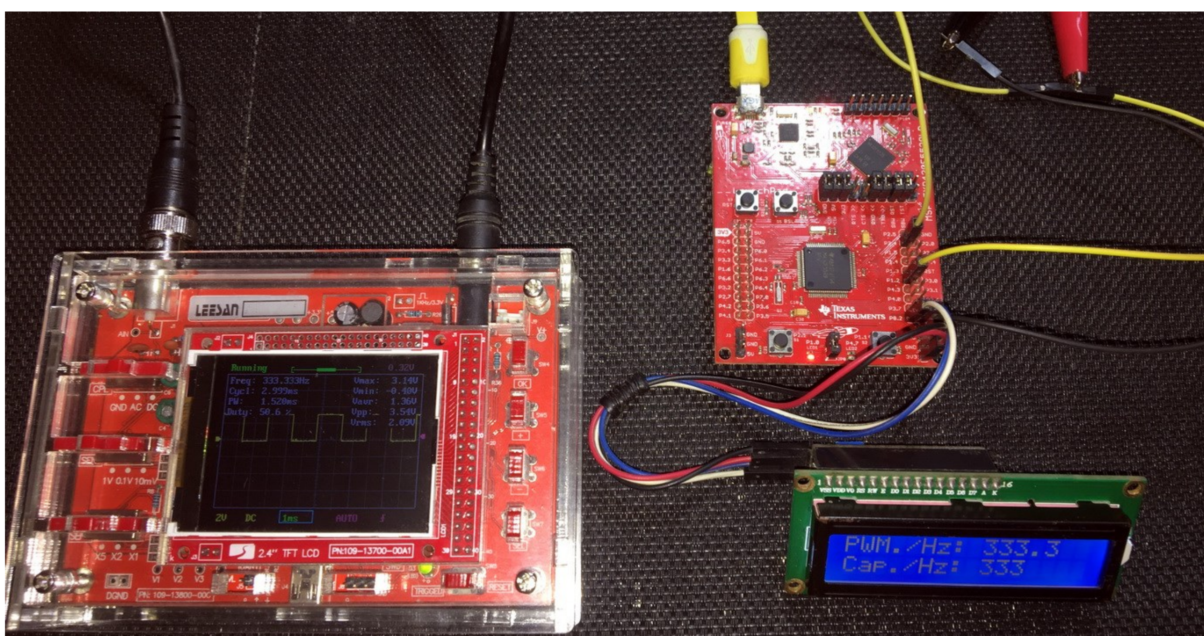
Dividing timer clock frequency (4MHZ) with time count differences, i.e. *pulse_ticks*, gives us the frequency of capture waveform.

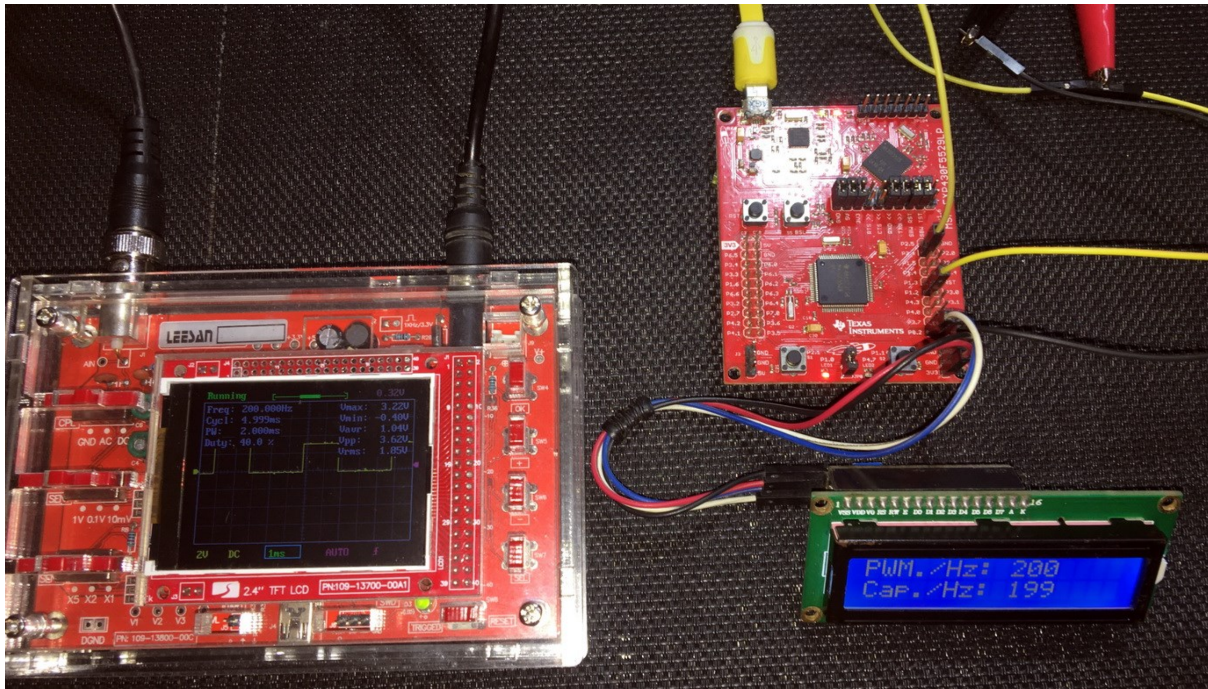
```
print_I(8, 1, (timer_clock_frequency / pulse_ticks));
delay_ms(100);
```

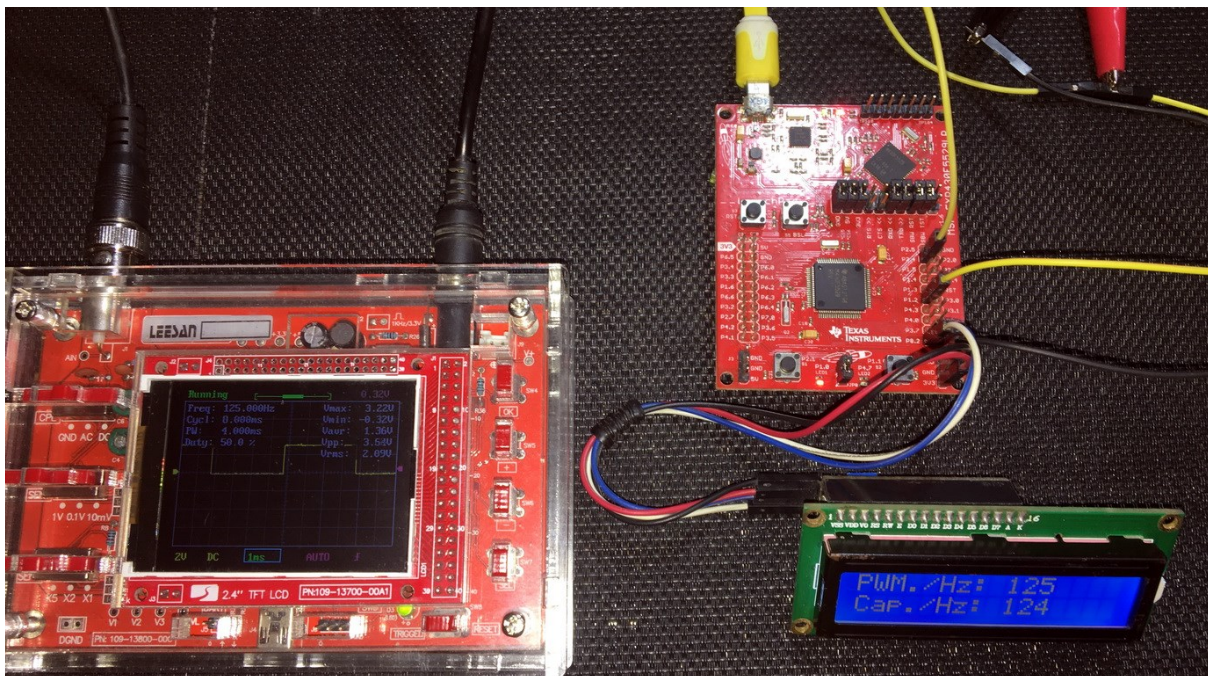
The limitation of this code is inaccuracy of measurement at high frequencies and that's because at high frequencies, capture resolution decreases. To capture high frequencies, we can use faster clocks or more sampling.

Demo





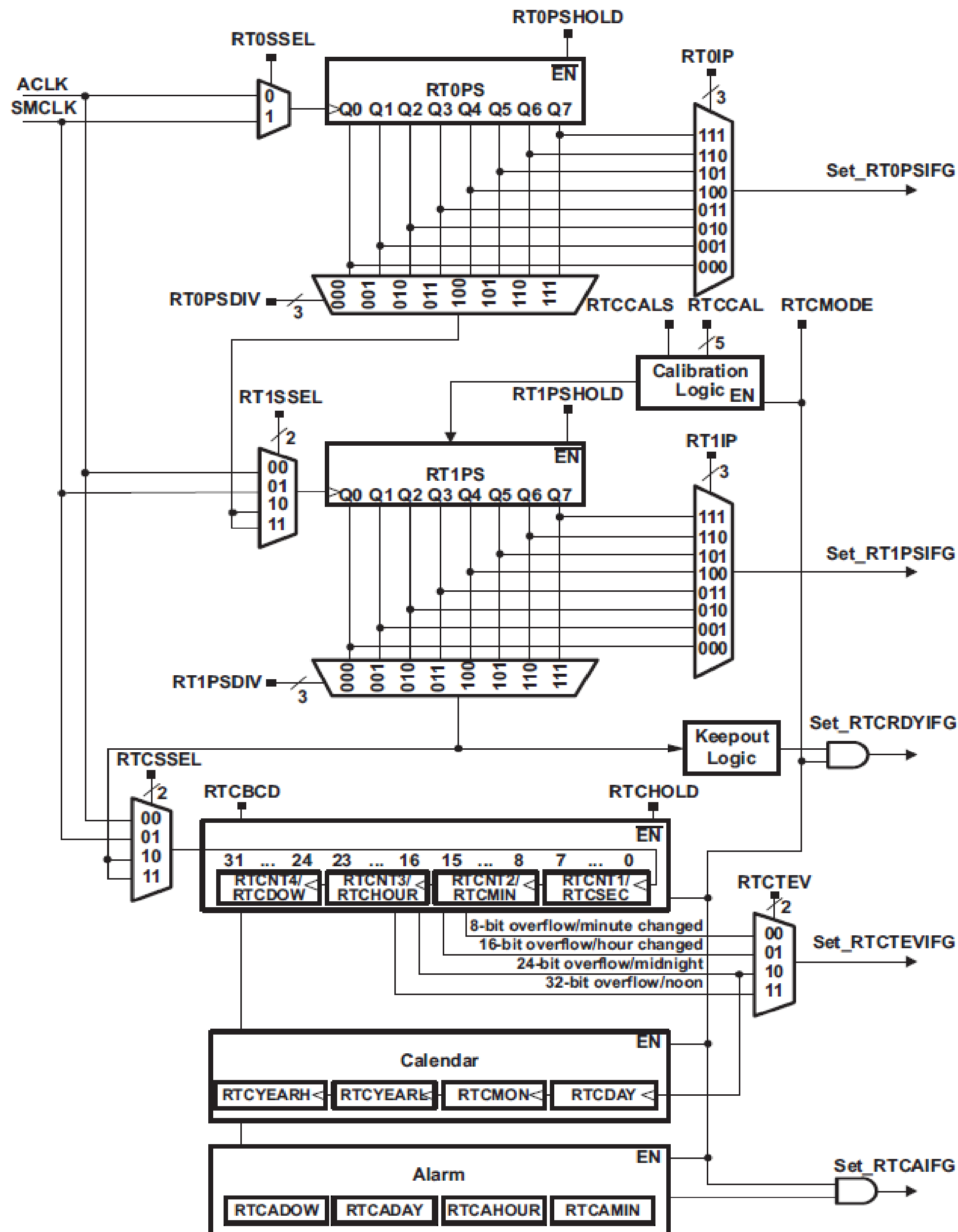




Video demo: https://youtu.be/SbxmKmFN6Pw

Real Time Clock – RTC_A

As stated before, RTC_A is mainly intended for time keeping and, in this demo, we will see how make a real-time clock with the RTC_A module.



Let's see some insights of RTC_A module. Firstly, we can see that the RTC_A module can be feed with either SMCLK or ACLK. Since SMCLK has other important uses, it is better to use ACLK for RTC_A. ACLK should have a frequency of 32.768 kHz and to achieve that it is better to use dedicated external crystal or TCXO. The RTC's counter *RTCBCD* can be then clocked directly by clock signals or via *RTxPS* registers. The desired tick period for RTC is obviously 1 Hz. Selectable either in BCD or hexadecimal format, the RTC_A calendar block gives time and date. A cool feature is the fact that internal algorithm takes care of leap-years from the year 1901 AD to 2099 AD. It is also possible to set calendar alarms but coder has to be careful because out-of-range or invalid date-time settings may result in unpredictable behaviour. Lastly, there are several interrupts for various timer events.

Code Example

```c
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"
#include "lcd_print.h"


Calendar current_Time;
unsigned char update_time = false;


void clock_init(void);
void GPIO_init(void);
void RTC_init(void);
void display_year(unsigned char x_pos, unsigned char y_pos, unsigned int value);
void display_time_date(unsigned char x_pos, unsigned char y_pos, unsigned char value);
unsigned int change_value(unsigned char x_pos, unsigned char y_pos, signed int value,
signed int value_min, signed int value_max, unsigned char value_type);
void set_RTC(void);


#pragma vector = RTC_VECTOR
__interrupt void RTC_A_ISR (void)
{
    switch(__even_in_range(RTCIV, 16))
    {
        case 0: break;  //No interrupts
        case 2:         //RTCRDYIFG
        {
            GPIO_toggleOutputOnPin(GPIO_PORT_P4,
                                   GPIO_PIN7);

            current_Time = RTC_A_getCalendarTime(RTC_A_BASE);

            update_time = true;

            break;
        }
        case 4: break;  //RTCEVIFG
        case 6: break;  //RTCAIFG
        case 8: break;  //RT0PSIFG
        case 10: break; //RT1PSIFG
        case 12: break; //Reserved
        case 14: break; //Reserved
        case 16: break; //Reserved
        default: break;
    }
}


void main(void)
{
    WDT_A_hold(WDT_A_BASE);

    clock_init();
    GPIO_init();
    RTC_init();
    LCD_init();

    LCD_clear_home();

    LCD_goto(6, 0);
    LCD_putstr(":  :");
```

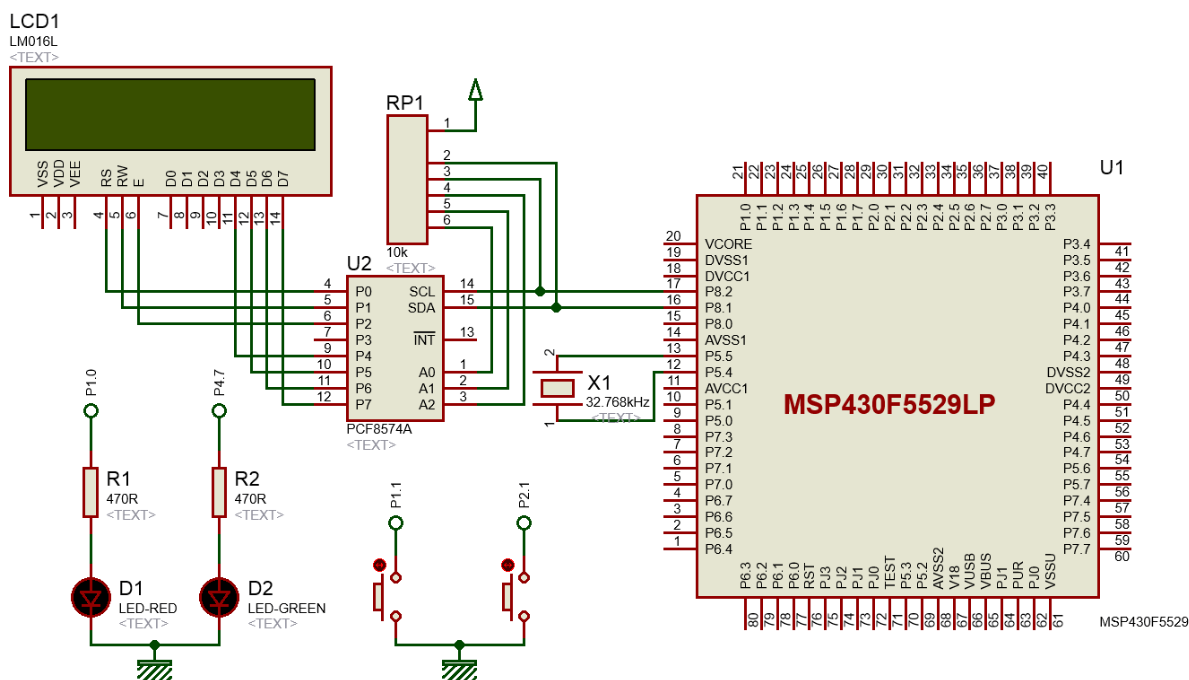
```
    LCD_goto(5, 1);
    LCD_putstr("/  /");

    while(1)
    {

        set_RTC();

         if(update_time)
         {
             display_time_date(4, 0, current_Time.Hours);
             display_time_date(7, 0, current_Time.Minutes);
             display_time_date(10, 0, current_Time.Seconds);
             display_time_date(3, 1, current_Time.DayOfMonth);
             display_time_date(6, 1, current_Time.Month);
             display_year(9, 1, current_Time.Year);
             update_time = 0;
         }
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_3,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_2);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1,
                                         GPIO_PIN1);

    GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P2,
                                         GPIO_PIN1);
```

```
    GPIO_setAsOutputPin(GPIO_PORT_P1,
                        GPIO_PIN0);

    GPIO_setDriveStrength(GPIO_PORT_P1,
                          GPIO_PIN0,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setDriveStrength(GPIO_PORT_P4,
                          GPIO_PIN7,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);
}


void RTC_init(void)
{
    current_Time.Seconds = 30;
    current_Time.Minutes = 10;
    current_Time.Hours = 10;
    current_Time.DayOfWeek = 1;
    current_Time.DayOfMonth = 1;
    current_Time.Month = 1;
    current_Time.Year = 2000;

    RTC_A_initCalendar(RTC_A_BASE,
                       &current_Time,
                       RTC_A_FORMAT_BINARY);

    RTC_A_setCalendarEvent(RTC_A_BASE,
                           RTC_A_CALENDAREVENT_MINUTECHANGE);

    RTC_A_clearInterrupt(RTC_A_BASE,
                         (RTCRDYIFG | RTCTEVIFG | RTCAIFG));

    RTC_A_enableInterrupt(RTC_A_BASE,
                          (RTCRDYIE | RTCTEVIE | RTCAIE));

    RTC_A_startClock(RTC_A_BASE);

    __enable_interrupt();
}


void display_year(unsigned char x_pos, unsigned char y_pos, unsigned int value)
{
    unsigned char tmp = 0;

    tmp = (value / 100);
    display_time_date(x_pos, y_pos, tmp);
    tmp = (value % 100);
    display_time_date((x_pos + 2), y_pos, tmp);
}


void display_time_date(unsigned char x_pos, unsigned char y_pos, unsigned char value)
{
    unsigned char ch = 0;

    ch = (value / 10);
    LCD_goto(x_pos, y_pos);
    LCD_putchar((ch + 0x30));
```

```
    ch = (value % 10);
    LCD_goto((1 + x_pos), y_pos);
    LCD_putchar((ch + 0x30));
}


unsigned int change_value(unsigned char x_pos, unsigned char y_pos, signed int value,
signed int value_min, signed int value_max, unsigned char value_type)
{
    while(1)
    {
        LCD_goto(x_pos, y_pos);

        switch(value_type)
        {
            case 1:
            {
                LCD_putstr("    ");
                break;
            }
            default:
            {
                LCD_putstr("  ");
                break;
            }
        }

        delay_ms(60);

        if(GPIO_getInputPinValue(GPIO_PORT_P1,
                                 GPIO_PIN1) == false)
        {
            value++;
        }

        if(value > value_max)
        {
            value = value_min;
        }

        switch(value_type)
        {
            case 1:
            {
                display_year(x_pos, y_pos, ((unsigned int)value));
                break;
            }
            default:
            {
                display_time_date(x_pos, y_pos, ((unsigned char)value));
                break;
            }
        }
        delay_ms(90);

        if(GPIO_getInputPinValue(GPIO_PORT_P2,
                                 GPIO_PIN1) == false)
        {
            while(GPIO_getInputPinValue(GPIO_PORT_P2,
                                        GPIO_PIN1) == false);

            delay_ms(200);
            return value;
        }
    };
}
```

```
void set_RTC(void)
{
    if(GPIO_getInputPinValue(GPIO_PORT_P2,
                             GPIO_PIN1) == false)
    {
        GPIO_setOutputHighOnPin(GPIO_PORT_P1,
                                GPIO_PIN0);

        while(GPIO_getInputPinValue(GPIO_PORT_P2,
                                    GPIO_PIN1) == false);

        RTC_A_disableInterrupt(RTC_A_BASE,
                               (RTCRDYIE | RTCTEVIE | RTCAIE));

        RTC_A_holdClock(RTC_A_BASE);

        __disable_interrupt();

        update_time = false;

        current_Time.Hours = change_value(4, 0, current_Time.Hours, 0, 23, 0);

        current_Time.Minutes = change_value(7, 0, current_Time.Minutes, 0, 59, 0);

        current_Time.Seconds = change_value(10, 0, current_Time.Seconds, 0, 59, 0);

        current_Time.DayOfMonth = change_value(3, 1, current_Time.DayOfMonth, 1, 31, 0);

        current_Time.Month = change_value(6, 1, current_Time.Month, 1, 12, 0);

        current_Time.Year = change_value(9, 1, current_Time.Year, 1970, 2099, 1);

        GPIO_setOutputLowOnPin(GPIO_PORT_P1,
                               GPIO_PIN0);

        RTC_A_initCalendar(RTC_A_BASE,
                           &current_Time,
                           RTC_A_FORMAT_BINARY);

        RTC_A_clearInterrupt(RTC_A_BASE,
                             (RTCRDYIFG | RTCTEVIFG | RTCAIFG));

        RTC_A_enableInterrupt(RTC_A_BASE,
                              (RTCRDYIE | RTCTEVIE | RTCAIE));

        RTC_A_startClock(RTC_A_BASE);

        __enable_interrupt();
    }
}
```

Hardware Setup



Explanation

Earlier I mentioned why we should use ACLK for RTC and so for that ground I used ACLK to drive the RTC_A module. On board externa 32.768kHzl crystal is used for ACLK as it is better than any other source.

```
UCS_initClockSignal(UCS_ACLK, UCS_XT1CLK_SELECT, UCS_CLOCK_DIVIDER_1);
```

Onboard LEDs and buttons are also used. P4.7 LED is used as second ticker while P1.0 LED is used as setting mode indicator. P1.1 button is used as a modifier button while P2.1 button acts as a set/enter button.

```
GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1, GPIO_PIN1);
GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P2, GPIO_PIN1);

GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
GPIO_setAsOutputPin(GPIO_PORT_P4, GPIO_PIN7);
```

Now let’s see how the RTC is setup.

```
void RTC_init(void)
{
    current_Time.Seconds = 30;
    current_Time.Minutes = 10;
    current_Time.Hours = 10;
    current_Time.DayOfWeek = 1;
    current_Time.DayOfMonth = 1;
    current_Time.Month = 1;
    current_Time.Year = 2000;

    RTC_A_initCalendar(RTC_A_BASE, &current_Time, RTC_A_FORMAT_BINARY);
```

```
    RTC_A_setCalendarEvent(RTC_A_BASE, RTC_A_CALENDAREVENT_MINUTECHANGE);

    RTC_A_clearInterrupt(RTC_A_BASE, (RTCRDYIFG | RTCTEVIFG | RTCAIFG));

    RTC_A_enableInterrupt(RTC_A_BASE, (RTCRDYIE | RTCTEVIE | RTCAIE));

    RTC_A_startClock(RTC_A_BASE);

    __enable_interrupt();
}
```

We start by initializing and loading date-time parameters of our choosing in binary format. We chose binary format because in that way we can avoid unnecessary BCD-to-binary conversions. Finally, we enable required interrupts after clearing them and start the RTC.

Inside the interrupt vector, we just check the *RTCRDY* interrupt flag and that's because from here we will extract current RTC time info. Other interrupts, although enabled, are not needed as we are not using alarm and event features.

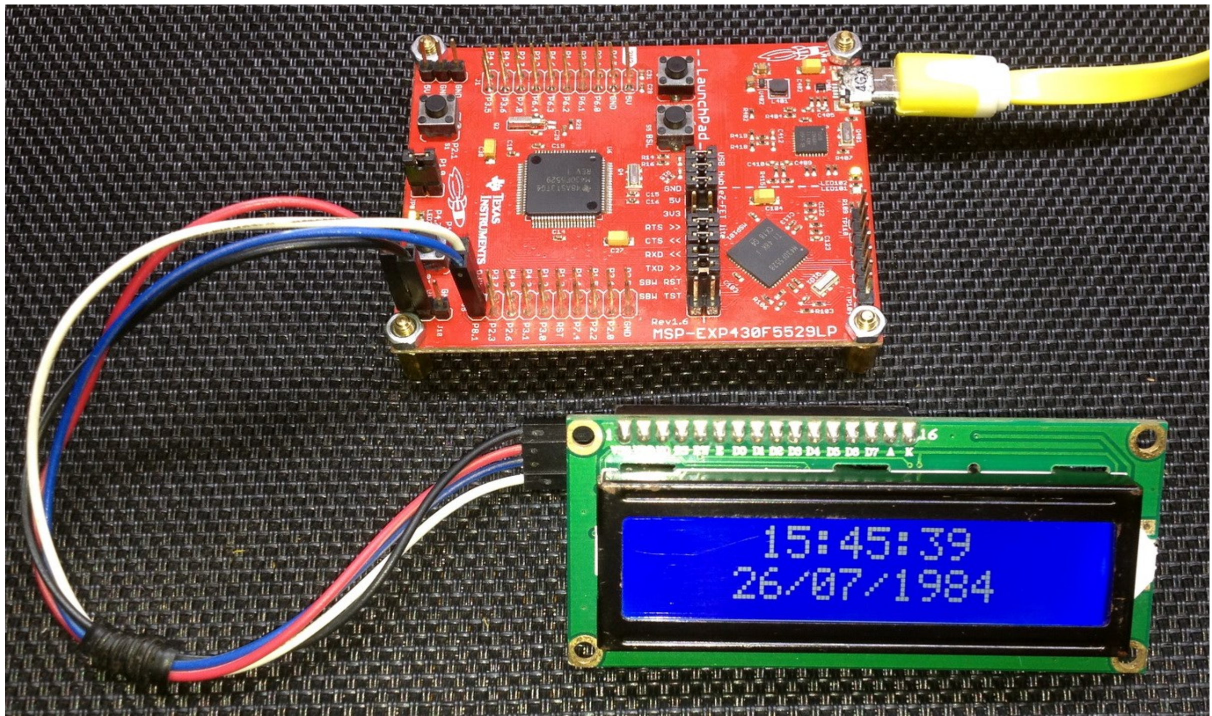
```
#pragma vector = RTC_VECTOR
__interrupt void RTC_A_ISR (void)
{
    switch(__even_in_range(RTCIV, 16))
    {
        case 0: break;  //No interrupts
        case 2:         //RTCRDYIFG
        {
            GPIO_toggleOutputOnPin(GPIO_PORT_P4,
                                   GPIO_PIN7);

            current_Time = RTC_A_getCalendarTime(RTC_A_BASE);

            update_time = true;

            break;
        }
        case 4: break;  //RTCEVIFG
        case 6: break;  //RTCAIFG
        case 8: break;  //RT0PSIFG
        case 10: break; //RT1PSIFG
        case 12: break; //Reserved
        case 14: break; //Reserved
        case 16: break; //Reserved
        default: break;
    }
}
```

In the main loop, we can do two things, either we can set time or watch it. When setting date-time, the display only shows the date-time parameters we are setting. After setup up, the RTC module is reinitialized with new date and time.

```
set_RTC();

if(update_time)
{
   display_time_date(4, 0, current_Time.Hours);
   display_time_date(7, 0, current_Time.Minutes);
   display_time_date(10, 0, current_Time.Seconds);
   display_time_date(3, 1, current_Time.DayOfMonth);
```

```
    display_time_date(6, 1, current_Time.Month);
    display_year(9, 1, current_Time.Year);
    update_time = 0;
}
```

Demo



Video demo: https://youtu.be/TKQeRMbaxKs

RTC_A as Counter

As with other timers, counter mode is a secondary feature of RTC_A module. In this mode, the calendar feature of RTC_A is not used. Instead of being an RTC, RTC_A in this mode behaves like a regular timer. However, unlike other regular timers, we cannot get timer count. This does not limit us much from using it for time bases as there are two counters that can be cascaded and, individually and independently prescaled. RTC_A is rarely used in this role because it is only used as such when we have run out of regular timers and have no need for a RTC.

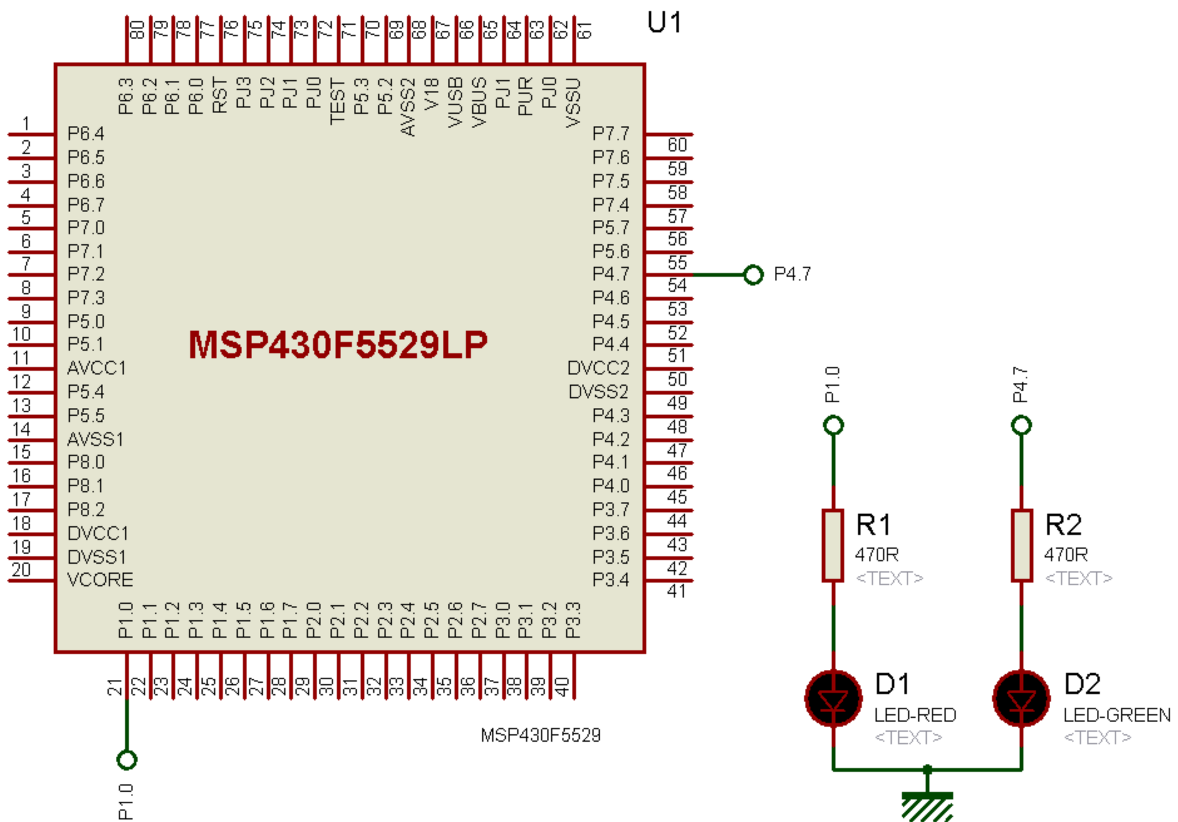
Code Example

```
#include "driverlib.h"
#include "delay.h"


void clock_init(void);
void GPIO_init(void);
void RTC_init(void);


#pragma vector=RTC_VECTOR
__interrupt void RTC_A_ISR(void)
{
    switch (__even_in_range(RTCIV, 16))
    {
        case 0: break;  //No interrupts
        case 2: break;  //RTCRDYIFG
        case 4:         //RTCEVIFG
        {
            GPIO_toggleOutputOnPin(GPIO_PORT_P1,
                                   GPIO_PIN0);

            GPIO_toggleOutputOnPin(GPIO_PORT_P4,
                                   GPIO_PIN7);

            break;
        }
        case 6: break;  //RTCAIFG
        case 8: break;  //RT0PSIFG
        case 10: break; //RT1PSIFG
        default: break;
    }
}


void main(void)
{
    WDT_A_hold(WDT_A_BASE);

    clock_init();
    GPIO_init();
    RTC_init();

    while(1)
    {
    };
}


void clock_init(void)
```

```
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_3,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsOutputPin(GPIO_PORT_P1,
                        GPIO_PIN0);

    GPIO_setDriveStrength(GPIO_PORT_P1,
                          GPIO_PIN0,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setDriveStrength(GPIO_PORT_P4,
                          GPIO_PIN7,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setOutputLowOnPin(GPIO_PORT_P1,
                           GPIO_PIN0);

    GPIO_setOutputHighOnPin(GPIO_PORT_P4,
                            GPIO_PIN7);
}


void RTC_init(void)
{
    RTC_A_initCounter(RTC_A_BASE,
                      RTC_A_CLOCKSELECT_RT1PS,
                      RTC_A_COUNTERSIZE_16BIT);

    RTC_A_initCounterPrescale(RTC_A_BASE,
```

```
                              RTC_A_PRESCALE_0,
                              RTC_A_PSCLOCKSELECT_SMCLK,
                              RTC_A_PSDIVIDER_4);

    RTC_A_initCounterPrescale(RTC_A_BASE,
                              RTC_A_PRESCALE_1,
                              RTC_A_PSCLOCKSELECT_RT0PS,
                              RTC_A_PSDIVIDER_16);

    RTC_A_setCounterValue(RTC_A_BASE,
                          0);

    RTC_A_clearInterrupt(RTC_A_BASE,
                         RTC_A_TIME_EVENT_INTERRUPT);

    RTC_A_clearInterrupt(RTC_A_BASE,
                         RTC_A_PRESCALE_TIMER1_INTERRUPT);

    RTC_A_enableInterrupt(RTC_A_BASE,
                          RTC_A_TIME_EVENT_INTERRUPT);

    RTC_A_enableInterrupt(RTC_A_BASE,
                          RTC_A_PRESCALE_TIMER1_INTERRUPT);

    RTC_A_startClock(RTC_A_BASE);

    __enable_interrupt();
}
```

Hardware Setup



Explanation

For simplest-possible demo, we will again make a LED toggler.

Obviously, counter mode is different from regular calendar mode and so we can use different clock settings. Here, we will be using SMCLK and it will be running at 4MHz.

```
UCS_initClockSignal(UCS_SMCLK, UCS_XT2CLK_SELECT, UCS_CLOCK_DIVIDER_1);
```

Both onboard LEDs are used but they are initialized at different logic states.

```
GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);

GPIO_setAsOutputPin(GPIO_PORT_P4, GPIO_PIN7);

GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);

GPIO_setOutputHighOnPin(GPIO_PORT_P4, GPIO_PIN7);
```

RTC_A module's setup this time is obviously different from the last RTC_A example.

```
void RTC_init(void)
{
    RTC_A_initCounter(RTC_A_BASE,
                      RTC_A_CLOCKSELECT_RT1PS,
                      RTC_A_COUNTERSIZE_16BIT);

    RTC_A_initCounterPrescale(RTC_A_BASE,
                              RTC_A_PRESCALE_0,
                              RTC_A_PSCLOCKSELECT_SMCLK,
                              RTC_A_PSDIVIDER_4);

    RTC_A_initCounterPrescale(RTC_A_BASE,
                              RTC_A_PRESCALE_1,
                              RTC_A_PSCLOCKSELECT_RT0PS,
                              RTC_A_PSDIVIDER_16);

    RTC_A_setCounterValue(RTC_A_BASE, 0);

    RTC_A_clearInterrupt(RTC_A_BASE, RTC_A_TIME_EVENT_INTERRUPT);

    RTC_A_clearInterrupt(RTC_A_BASE, RTC_A_PRESCALE_TIMER1_INTERRUPT);

    RTC_A_enableInterrupt(RTC_A_BASE, RTC_A_TIME_EVENT_INTERRUPT);

    RTC_A_enableInterrupt(RTC_A_BASE, RTC_A_PRESCALE_TIMER1_INTERRUPT);

    RTC_A_startClock(RTC_A_BASE);

    __enable_interrupt();
}
```

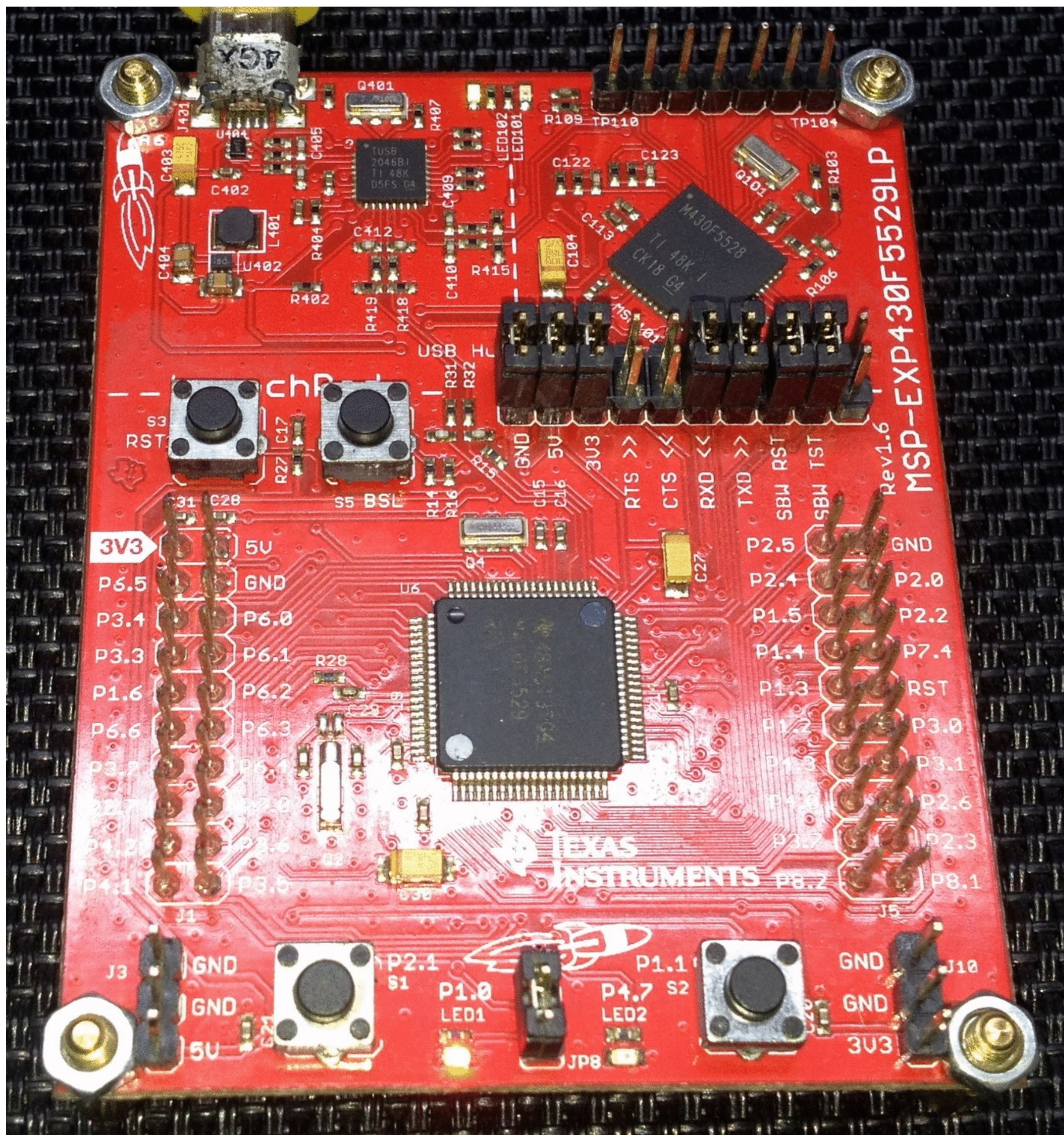
Firstly, the size of the main RTC counter and its clock source are set. Here the clock source is *RT1PS*. *RT0PS* and *RT1PS* are sub-counters. *RT0PS* is clocked with either SMCLK or ACLK while *RT1PS* is clocked with SMCLK, ACLK or prescaled *RT0PS*. Finally, the RTC module's main counter can be fed with SMCLK, ACLK or prescaled *RT1PS*. Thus, here the latter codes involve setting these parameters. *RT0PS* is fed with 4MHz SMCLK and then a prescalar of 4 is applied. The resultant frequency is a 1MHz which is used to feed *RT1PS*. Now *RT1PS* is further prescaled by 16, resulting in 62.5kHz clock. This is what that

feeds the main counter. Only 16 bits of the 32-bit main counter is used. Thus, the main counter overflows roughly about every second.

Every overflow event triggers an interrupt. Inside the interrupt onboard LEDs are toggled.

```
#pragma vector=RTC_VECTOR
__interrupt void RTC_A_ISR(void)
{
    switch (__even_in_range(RTCIV, 16))
    {
        case 0: break;  //No interrupts
        case 2: break;  //RTCRDYIFG
        case 4:         //RTCEVIFG
        {
            GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);

            GPIO_toggleOutputOnPin(GPIO_PORT_P4, GPIO_PIN7);

            break;
        }
        case 6: break;  //RTCAIFG
        case 8: break;  //RT0PSIFG
        case 10: break; //RT1PSIFG
        default: break;
    }
}
```

Demo

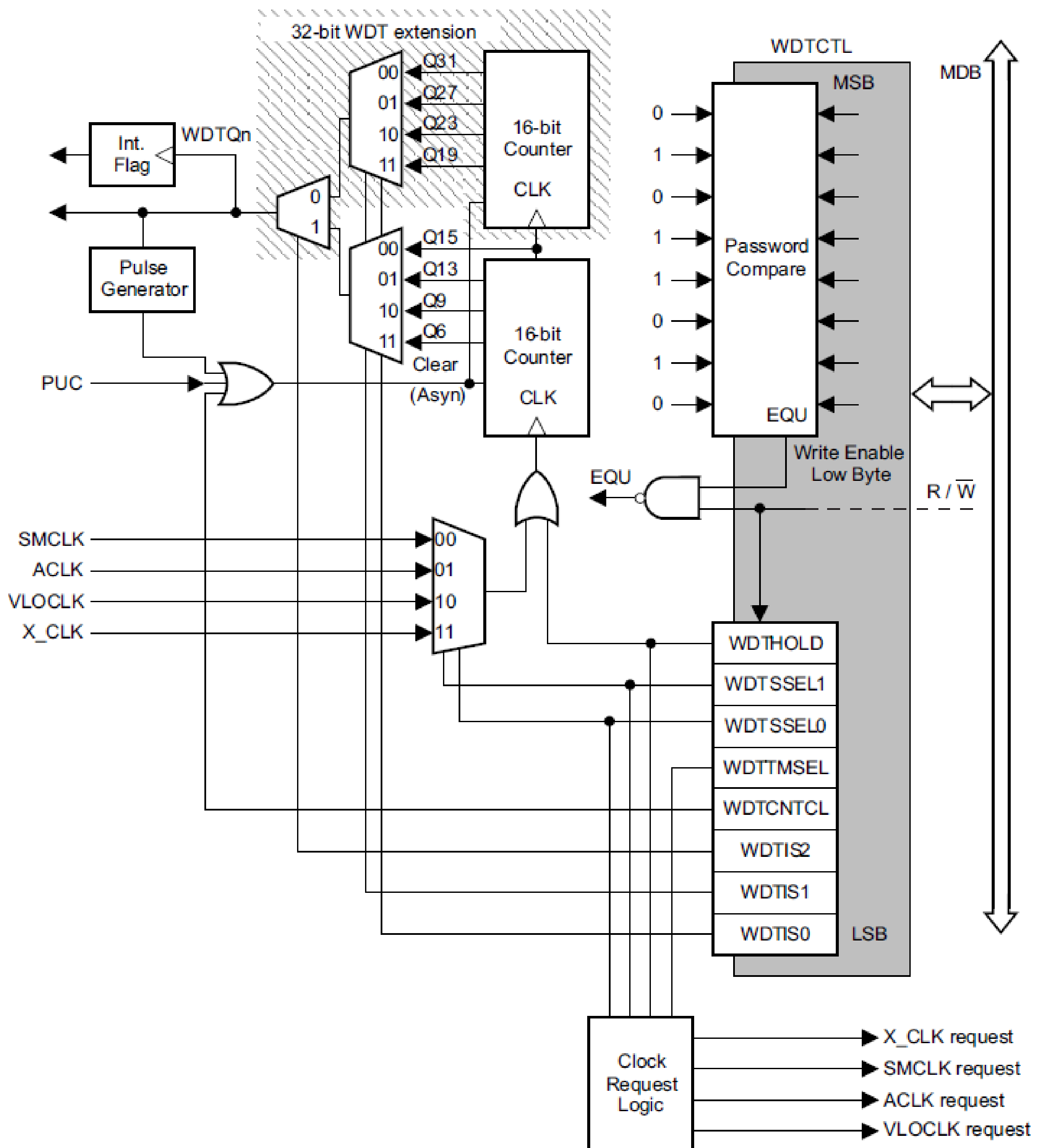


Demo video: https://youtu.be/caOQTGAZXiQ

`

Watchdog Timer - WDTA

The prime task of a watchdog timer is to reset a microcontroller should there be any software irresponsiveness. MSP430F5529's watchdog timer, *WDTA* can be used like a regular watchdog timer as well as an interval timer. WDTA is similar to the previously seen WDT+ but it is a bit advanced in some areas like 32-bit counter instead of 16-bit counter. Shown below is the block diagram of WDTA:



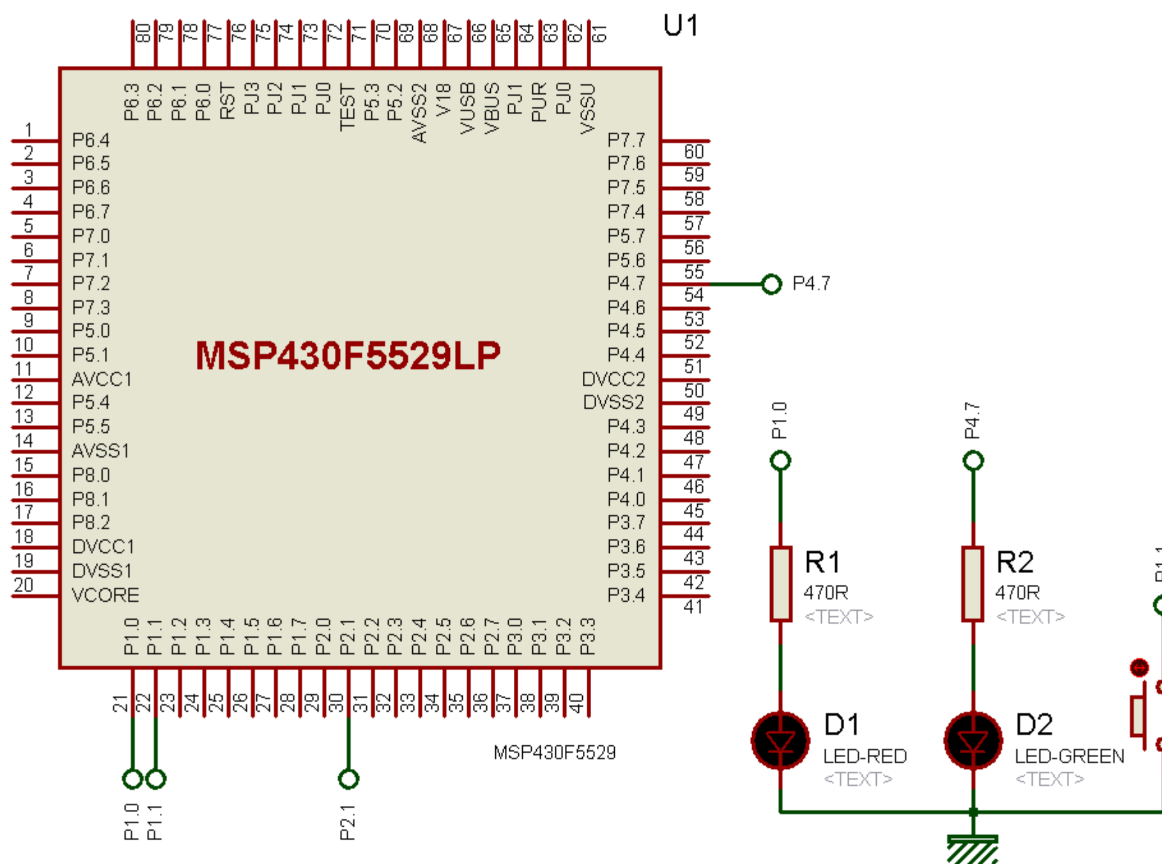
Unlike the watchdogs of many other microcontrollers, WDTA is password protected and it means that to read/write it, a special code (*0x05A*) needs to be set first.

Code Example

```
#include "driverlib.h"
#include "delay.h"


void clock_init(void);
void GPIO_init(void);
void WDTA_init(void);


void main(void)
{
    WDT_A_hold(WDT_A_BASE);

    clock_init();
    GPIO_init();
    WDTA_init();

    while(1)
    {
        GPIO_toggleOutputOnPin(GPIO_PORT_P1,
                               GPIO_PIN0);

        delay_ms(600);

        WDT_A_resetTimer(WDT_A_BASE);

        if(GPIO_getInputPinValue(GPIO_PORT_P1,
                                 GPIO_PIN1) == false)
        {
            GPIO_setOutputHighOnPin(GPIO_PORT_P4,
                                    GPIO_PIN7);

            while(1);
        }
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_3,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);
```

```
    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_2);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1,
                                         GPIO_PIN1);

    GPIO_setAsOutputPin(GPIO_PORT_P1,
                        GPIO_PIN0);

    GPIO_setDriveStrength(GPIO_PORT_P1,
                          GPIO_PIN0,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);
    GPIO_setDriveStrength(GPIO_PORT_P4,
                          GPIO_PIN7,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setOutputLowOnPin(GPIO_PORT_P1,
                           GPIO_PIN0);

    GPIO_setOutputLowOnPin(GPIO_PORT_P4,
                           GPIO_PIN7);
}


void WDTA_init(void)
{
    WDT_A_initWatchdogTimer(WDT_A_BASE,
                            WDT_A_CLOCKSOURCE_VLOCLK,
                            WDT_A_CLOCKDIVIDER_32K);

    WDT_A_start(WDT_A_BASE);

    WDT_A_resetTimer(WDT_A_BASE);
}
```

Hardware Setup



Explanation

After power-on reset, WDTA is configured in watchdog mode with an approximate initial 32ms reset interval. Unless commanded to stop immediately, it will keep resetting MCU. This is why at the start of every program the following code is placed:

```
WDT_A_hold(WDT_A_BASE);
```

This piece of code disables WDTA until reconfigured and reused.

To setup WDTA in watchdog mode, we need to specify its source of clock signal. In this example, it is VLOCLK. We also need to decide the WDTA clock divider in order to achieve the required amount of period.

```
void WDTA_init(void)
{
    WDT_A_initWatchdogTimer(WDT_A_BASE, WDT_A_CLOCKSOURCE_VLOCLK, WDT_A_CLOCKDIVIDER_32K);

    WDT_A_start(WDT_A_BASE);

    WDT_A_resetTimer(WDT_A_BASE);
}
```

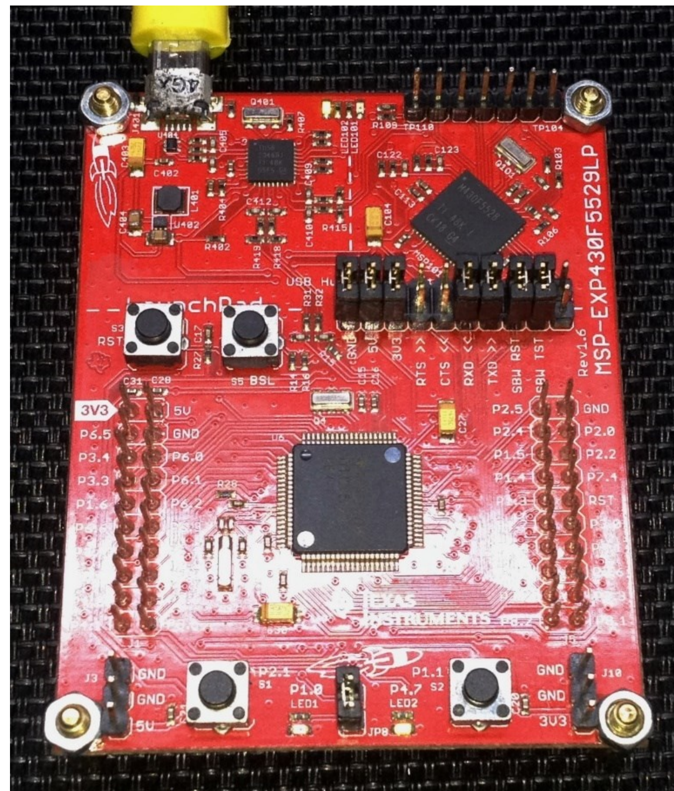
Here the time period according to the settings is about 3 seconds (32000 / 10 kHz). If WDTA is not reset within this time window, a reset will be triggered.

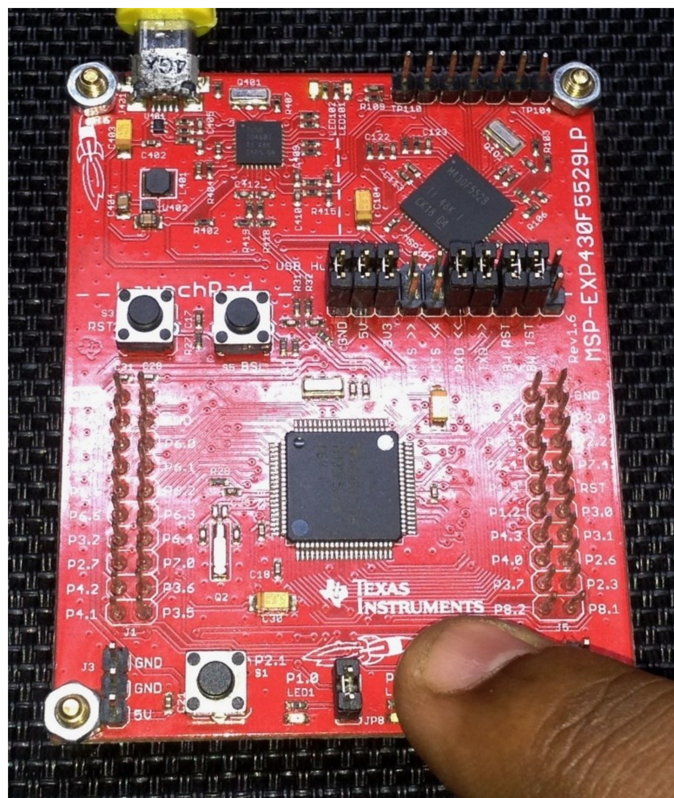
In the main, P1.0 LED is toggled every 600ms and WDTA is reset. Thus, WDTA counter is reset 5 times earlier than reset interval. This goes on until P1.1 push button is pressed. When the button is pressed, P4.7 LED is lit and a predefined intentional software loop is entered. Since WDTA counter is no longer reset in the loop, WDTA counter keeps ticking and a reset occurs after 3 seconds, starting everything from the beginning.

```
GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);

delay_ms(600);

WDT_A_resetTimer(WDT_A_BASE);

if(GPIO_getInputPinValue(GPIO_PORT_P1, GPIO_PIN1) == false)
{
   GPIO_setOutputHighOnPin(GPIO_PORT_P4, GPIO_PIN7);

   while(1);
}
```

Note that after reset, RAM data are usually not erased unless modified or reinitialized. However, in real life applications, RAM data may be corrupted because something unusual caused the reset to occur. Therefore, we can use CRC or other means to check RAM data integrity if past data are needed.

Demo





Demo video: https://youtu.be/1EhM42_W_GY

WDTA as an Interval Timer

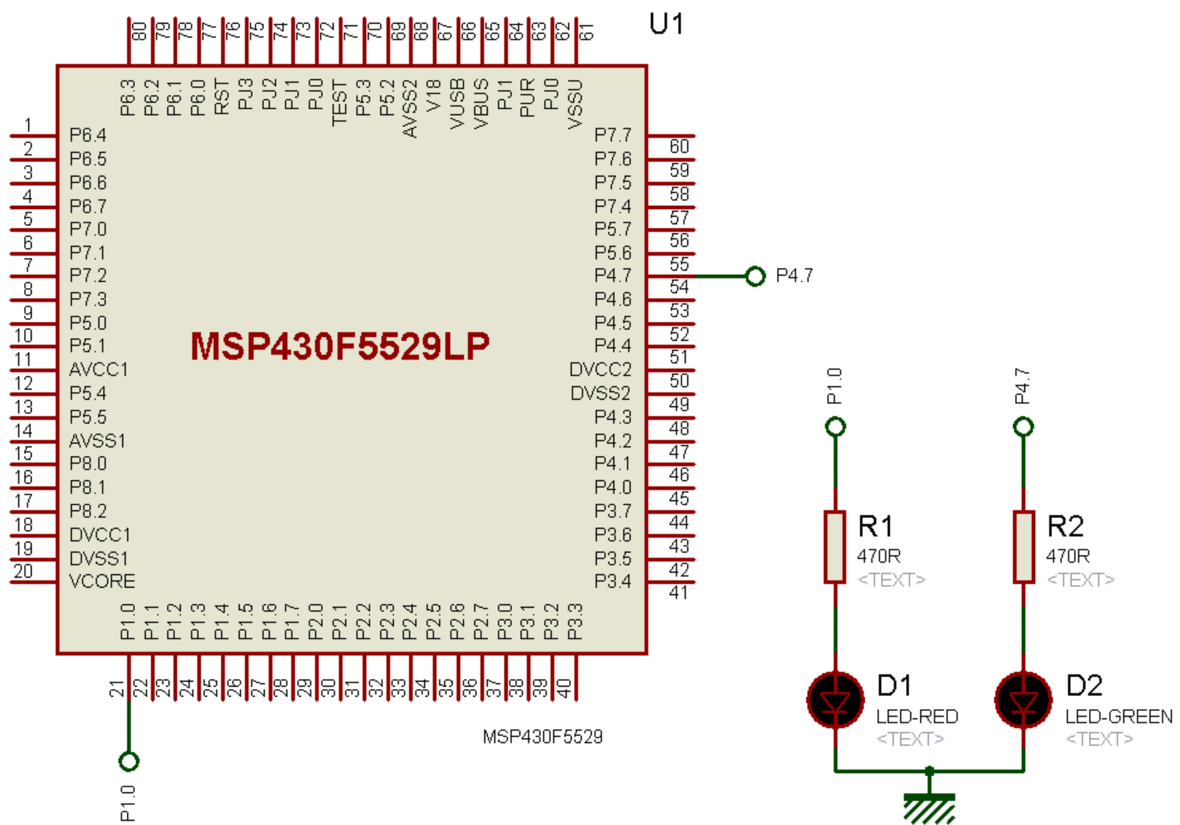
As mentioned before, WDTA can be used like an ordinary interval timer if watchdog function is not needed. This is strictly a secondary property of WDTA and there are some limitations of it. Still I would thank TI for this dexterity because sometimes in big projects people tend to run out of timers.

Code Example

```
#include "driverlib.h"


void GPIO_init(void);
void WDTA_init(void);


#pragma vector = WDT_VECTOR
__interrupt void WDT_A_ISR (void)
{
    GPIO_toggleOutputOnPin( GPIO_PORT_P1,
                            GPIO_PIN0);

    GPIO_toggleOutputOnPin( GPIO_PORT_P4,
                            GPIO_PIN7);
}


void main(void)
{
    WDT_A_hold(WDT_A_BASE);

    GPIO_init();
    WDTA_init();

    while(1)
    {
    };
}


void GPIO_init(void)
{
    GPIO_setAsOutputPin(GPIO_PORT_P1,
                        GPIO_PIN0);

    GPIO_setDriveStrength(GPIO_PORT_P1,
                          GPIO_PIN0,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setDriveStrength(GPIO_PORT_P4,
                          GPIO_PIN7,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setOutputHighOnPin(GPIO_PORT_P1,
                            GPIO_PIN0);

    GPIO_setOutputLowOnPin(GPIO_PORT_P4,
                           GPIO_PIN7);
}
```

```
void WDTA_init(void)
{
    WDT_A_initIntervalTimer(WDT_A_BASE,
                            WDT_A_CLOCKSOURCE_SMCLK,
                            WDT_A_CLOCKDIVIDER_512K);

    SFR_clearInterrupt(SFR_WATCHDOG_INTERVAL_TIMER_INTERRUPT);

    SFR_enableInterrupt(SFR_WATCHDOG_INTERVAL_TIMER_INTERRUPT);

    WDT_A_start(WDT_A_BASE);

    __enable_interrupt();
}
```

Hardware Setup



Explanation

The demo here is a simple LED toggler. Onboard LEDs are used. UCS clock settings are untouched and so all clocks are in their defaults.

Firstly, WDTA is stopped to prevent a software reset.

```
WDT_A_hold(WDT_A_BASE);
```

Configuration of WDTA in interval mode is nothing different from WDTA in watchdog mode. We have to specify WDTA clock source and its divider. Here the clock source is SMCLK and the divider is 512000. Since we are using default clock setting, SMCLK is running at about 1MHz speed. Thus, dividing this speed with 512000 results in approximately 500ms interval. We cannot access WDTA counter and so we will have to use interrupt method when using WDTA in interval mode. Thus, at every 500ms an interrupt is triggered.

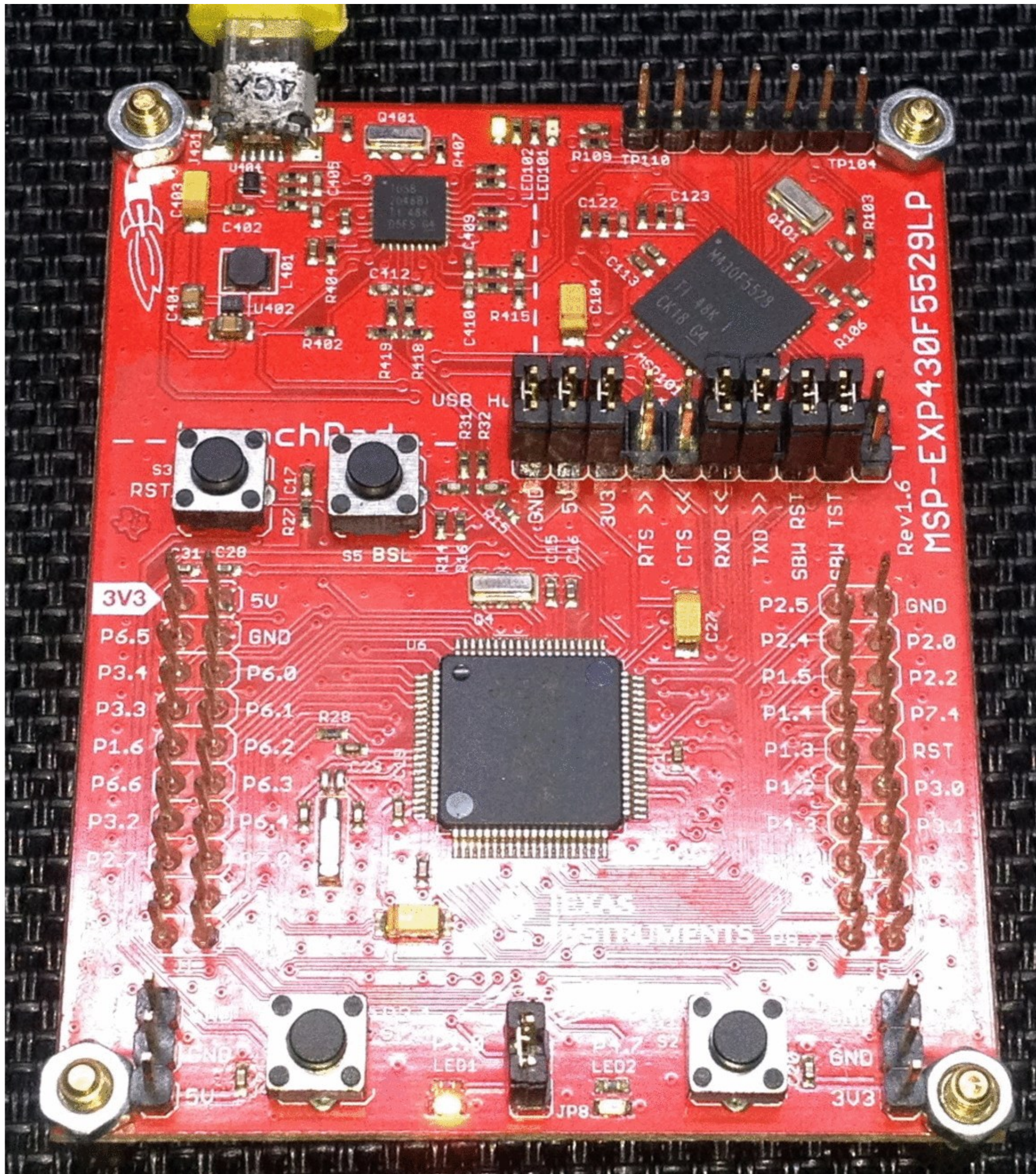
```
void WDTA_init(void)
{
    WDT_A_initIntervalTimer(WDT_A_BASE, WDT_A_CLOCKSOURCE_SMCLK, WDT_A_CLOCKDIVIDER_512K);

    SFR_clearInterrupt(SFR_WATCHDOG_INTERVAL_TIMER_INTERRUPT);

    SFR_enableInterrupt(SFR_WATCHDOG_INTERVAL_TIMER_INTERRUPT);

    WDT_A_start(WDT_A_BASE);

    __enable_interrupt();
}
```

Inside the interrupt, onboard LED states are toggled.

```
#pragma vector = WDT_VECTOR
__interrupt void WDT_A_ISR (void)
{
    GPIO_toggleOutputOnPin( GPIO_PORT_P1, GPIO_PIN0);

    GPIO_toggleOutputOnPin( GPIO_PORT_P4, GPIO_PIN7);
}
```

There is nothing done in the main loop and so it is empty.

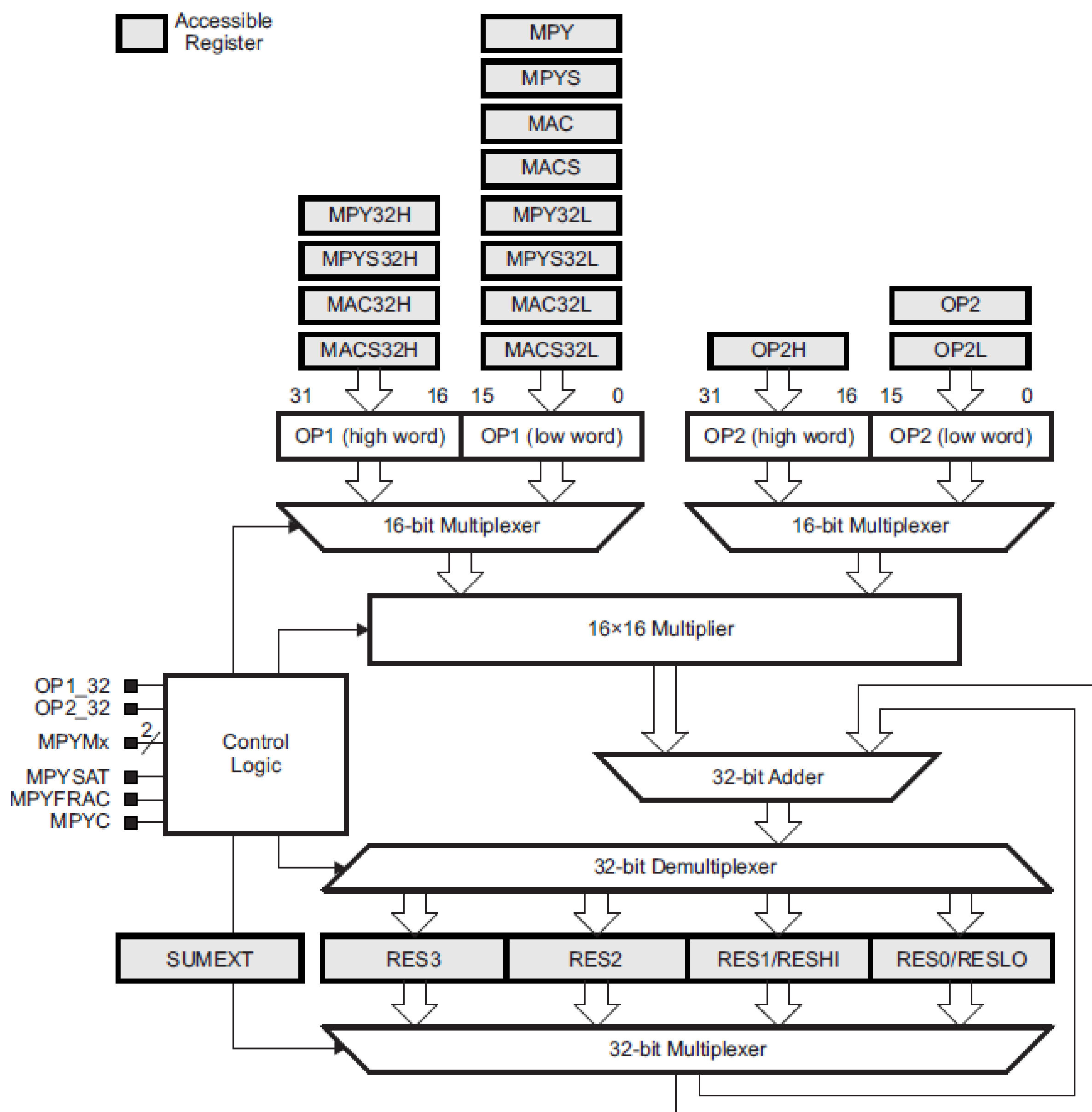
Demo



Demo video: https://youtu.be/vfGtHR4QTi0

Hardware Multiplier Module – MPY32

Many high-end microcontrollers consist of internal hardware multiplier. So, what does this hardware do? Well a multiplier performs multiplication and simply that's it. Why is it important to have a hardware multiplier when we are coding in high-level C language and simple multiplication instruction is enough to do the job? Without hardware multiplier, multiplication is made possible by complex coding, i.e. by hidden software methods. Unlike dedicated hardware, software method takes time and uses resources since it is an emulated task. It is more like hardware rendering vs software rendering. We tend to like good graphics while playing games. A general-purpose computer without a dedicated graphics card may not properly run a game that has good graphical details. A gaming computer, on the other hand, has a dedicated graphics card and it can run the game with best performance. Just like the graphics card, a hardware multiplier is such a necessary hardware renderer that is often needed in time-limited complex computations and digital signal processing (DSP).



MSP430F5529 has a 32-bit hardware multiplier named ***MPY32***. Like DMA, it is not a part of the CPU and so it won't affect the operations of the CPU when used. However, the CPU is needed to load and extract data from the multiplier. It supports the following multiplications:

• Unsigned multiply
• Signed multiply
• Unsigned multiply accumulate
• Signed multiply accumulate
• 8-bit, 16-bit, 24-bit, and 32-bit operands
• Saturation
• Fractional numbers
• 8-bit and 16-bit operation compatible with 16-bit hardware multiplier
• 8-bit and 24-bit multiplications without requiring a "sign extend" instruction

Software-based multiplication utilizes CPU and keeps it busy. Software multiplication is like repetitive addition. We don't see it happening because the whole process is done in machine/assembly language level while we are coding in high-level C language.

Code Example

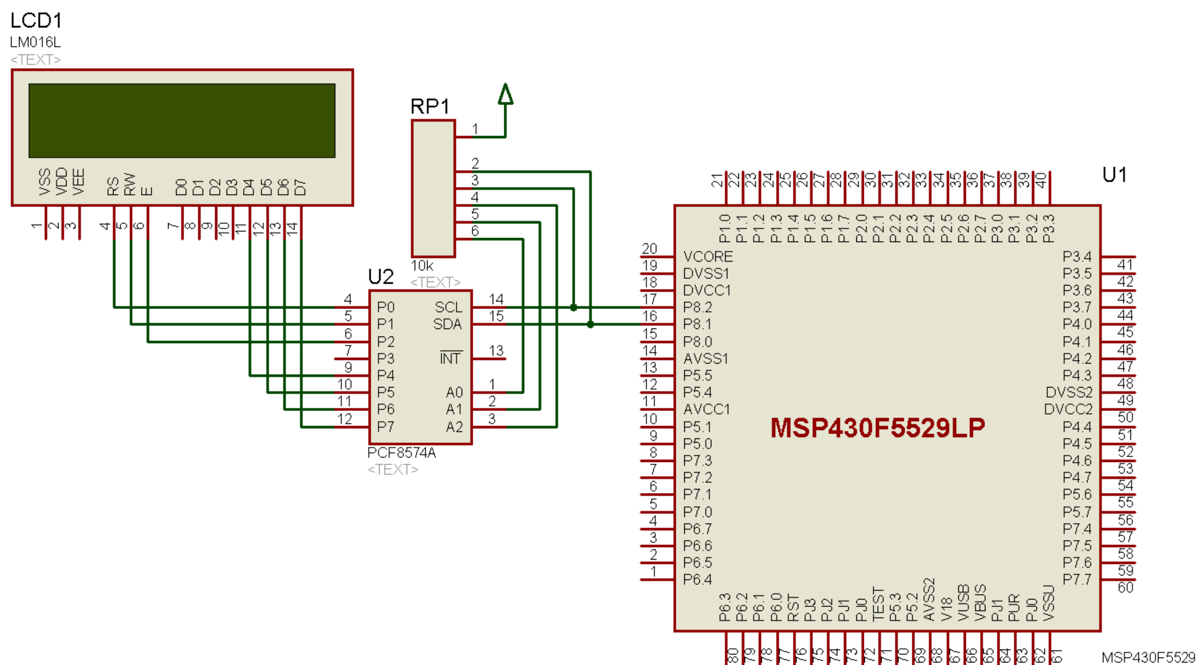
```
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"
#include "lcd_print.h"


void clock_init(void);
void timer_T0A5_init(void);


void main(void)
{
    unsigned int i = 0;
    unsigned int j = 0;

    signed long res = 0;
    signed int num1 = 263;
    signed int num2 = 249;
    unsigned int timer_count = 0;

    WDT_A_hold(WDT_A_BASE);

    clock_init();
    timer_T0A5_init();

    LCD_init();
    LCD_clear_home();

    LCD_goto(4, 0);
    LCD_putstr("Software");
    LCD_goto(1, 1);
    LCD_putstr("Multiplication");

    delay_ms(2000);
    LCD_clear_home();

    LCD_goto(0, 0);
    LCD_putstr("263(x)249=");
    LCD_goto(0, 1);
    LCD_putstr("T.CNT = ");

    Timer_A_startCounter(__MSP430_BASEADDRESS_T0A5__,
                         TIMER_A_CONTINUOUS_MODE);

    for(i = 0; i < num1; i++)
    {
        for(j = 0; j < num2; j++)
        {
            res++;
        }
    }

    timer_count = Timer_A_getCounterValue(__MSP430_BASEADDRESS_T0A5__);

    Timer_A_stop(__MSP430_BASEADDRESS_T0A5__);

    print_I(10, 0, res);
    print_I(10, 1, timer_count);

    delay_ms(4000);
    LCD_clear_home();
```

```
    res = 0;
    timer_count = 0;
    Timer_A_clear(__MSP430_BASEADDRESS_T0A5__);

    LCD_goto(4, 0);
    LCD_putstr("Hardware");
    LCD_goto(1, 1);
    LCD_putstr("Multiplication");

    delay_ms(2000);
    LCD_clear_home();

    LCD_goto(0, 0);
    LCD_putstr("263(x)249=");
    LCD_goto(0, 1);
    LCD_putstr("T.CNT = ");

    Timer_A_startCounter(__MSP430_BASEADDRESS_T0A5__,
                         TIMER_A_CONTINUOUS_MODE);

    MPY32_setOperandOne16Bit(MPY32_MULTIPLY_UNSIGNED,
                             num1);

    MPY32_setOperandTwo16Bit(num2);

    res = MPY32_getResult();

    timer_count = Timer_A_getCounterValue(__MSP430_BASEADDRESS_T0A5__);

    Timer_A_stop(__MSP430_BASEADDRESS_T0A5__);

    print_I(10, 0, res);
    print_I(10, 1, timer_count);

    delay_ms(4000);


    num1 = -99;
    num2 = 660;

    LCD_clear_home();

    LCD_goto(0, 0);
    LCD_putstr("Signed Software");
    LCD_goto(1, 1);
    LCD_putstr("Multiplication");

    delay_ms(2000);
    LCD_clear_home();

    LCD_goto(0, 0);
    LCD_putstr("-99(x)660=");
    LCD_goto(0, 1);
    LCD_putstr("T.CNT = ");

    Timer_A_startCounter(__MSP430_BASEADDRESS_T0A5__,
                         TIMER_A_CONTINUOUS_MODE);

    res = (((signed long)num1) * ((signed long)num2));

    timer_count = Timer_A_getCounterValue(__MSP430_BASEADDRESS_T0A5__);

    Timer_A_stop(__MSP430_BASEADDRESS_T0A5__);

    print_I(10, 0, res);
    print_I(10, 1, timer_count);
```

```
    delay_ms(4000);
    LCD_clear_home();


    res = 0;
    timer_count = 0;

    Timer_A_clear(__MSP430_BASEADDRESS_T0A5__);

    LCD_goto(0, 0);
    LCD_putstr("Signed Hardware");
    LCD_goto(1, 1);
    LCD_putstr("Multiplication");

    delay_ms(2000);
    LCD_clear_home();

    LCD_goto(0, 0);
    LCD_putstr("-99(x)660=");
    LCD_goto(0, 1);
    LCD_putstr("T.CNT = ");

    Timer_A_startCounter(__MSP430_BASEADDRESS_T0A5__,
                         TIMER_A_CONTINUOUS_MODE);

    MPY32_setOperandOne16Bit(MPY32_MULTIPLY_SIGNED,
                             num1);

    MPY32_setOperandTwo16Bit(num2);

    res = MPY32_getResult();

    timer_count = Timer_A_getCounterValue(__MSP430_BASEADDRESS_T0A5__);

    Timer_A_stop(__MSP430_BASEADDRESS_T0A5__);

    print_I(10, 0, res);
    print_I(10, 1, timer_count);

    while(1)
    {
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_3,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
```

```
                              UCS_XT2CLK_SELECT,
                              UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void timer_T0A5_init(void)
{
    Timer_A_initContinuousModeParam ContinuousModeParam =
    {
        TIMER_A_CLOCKSOURCE_SMCLK,
        TIMER_A_CLOCKSOURCE_DIVIDER_1,
        TIMER_A_TAIE_INTERRUPT_DISABLE,
        TIMER_A_DO_CLEAR,
        false
    };

    Timer_A_stop(__MSP430_BASEADDRESS_T0A5__);

    Timer_A_initContinuousMode(__MSP430_BASEADDRESS_T0A5__,
                               &ContinuousModeParam);
}
```

Hardware Setup



Explanation

For demoing the merits of having hardware multiplier, I coded a simple program that simply performs certain calculations and keep record of time count as to determine how long it took to perform the calculations. Two sorts of multiplications are performed both using hardware module and by software means. Same numbers are used in both cases.

Timer TA0 is set with SMCLK having XT2_CLK source and no division. Thus, it is running at 4MHz speed. No interrupt is used to keep things simple. The measure of time is taken as timer ticks and not as actual time in seconds or other units. The larger the value of ticks the longer is the time it took to perform a given calculation. The timer is started right before performing a calculation and stopped immediately after completing it.

```
void timer_T0A5_init(void)
{
    Timer_A_initContinuousModeParam ContinuousModeParam =
    {
         TIMER_A_CLOCKSOURCE_SMCLK,
         TIMER_A_CLOCKSOURCE_DIVIDER_1,
         TIMER_A_TAIE_INTERRUPT_DISABLE,
         TIMER_A_DO_CLEAR,
         false
    };

    Timer_A_stop(__MSP430_BASEADDRESS_T0A5__);

    Timer_A_initContinuousMode(__MSP430_BASEADDRESS_T0A5__,
                               &ContinuousModeParam);
}
```

The first calculation is an unsigned multiplication of numbers *263* and *249*. As I stated before software multiplication is simply repetitive addition and the code below does exactly that:

```
for(i = 0; i < num1; i++)
{
    for(j = 0; j < num2; j++)
    {
        res++;
    }
}
```

The numbers to be multiplied form loops and a variable called *res* is incremented on each loop passes. This, in effect, behaves like a rudimentary multiplication. The result of this multiplication is *65487*. It is revealed that this calculation takes about 13500 counts or about 3ms.

The timer is cleared and restarted. This time however the hardware multiplier is used to multiply the same numbers again.

```
MPY32_setOperandOne16Bit(MPY32_MULTIPLY_UNSIGNED,
                         num1);

MPY32_setOperandTwo16Bit(num2);

res = MPY32_getResult();
```

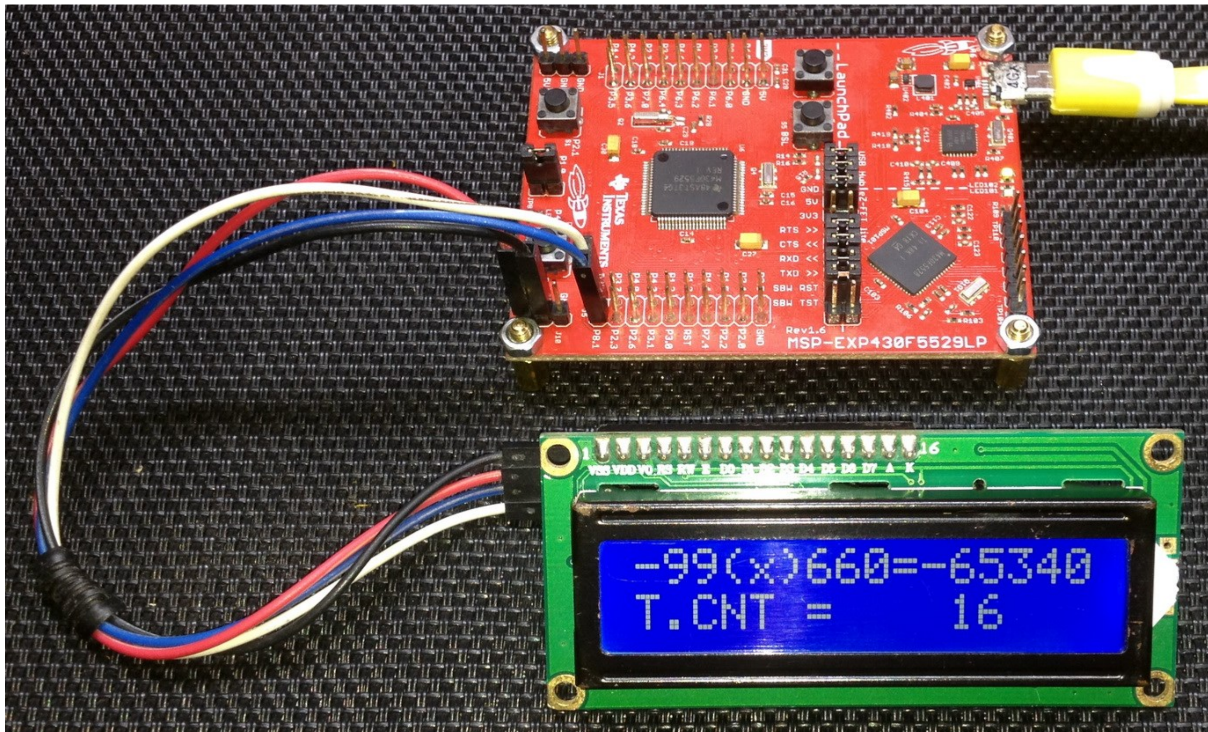
It is found that this time the same calculation takes just 17 ticks or about 4µs. This shows that hardware multiplication is roughly 750 times faster than software multiplication.

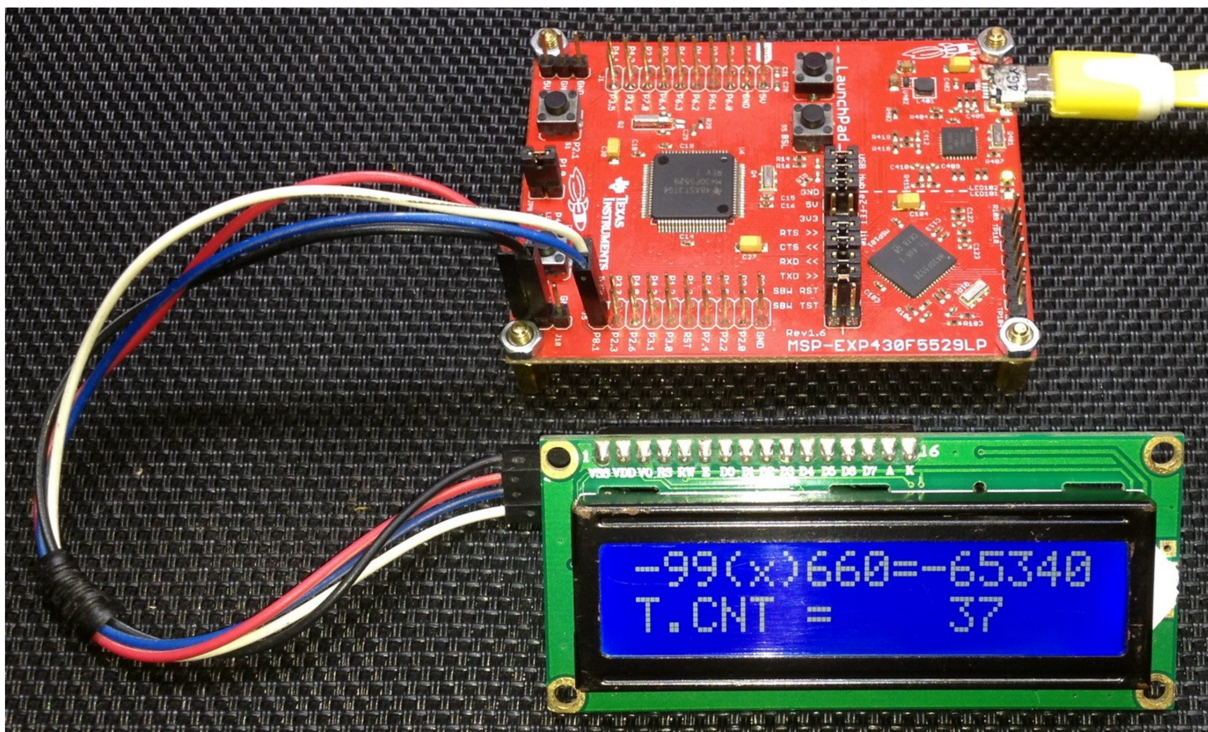
The same is done once more but this time using signed software multiplication of numbers *-99* and *660*. This time however compiler's multiplication operator is used. The result of this calculation is -*65340*. It is found that it takes 36 timer ticks or 9 µs.

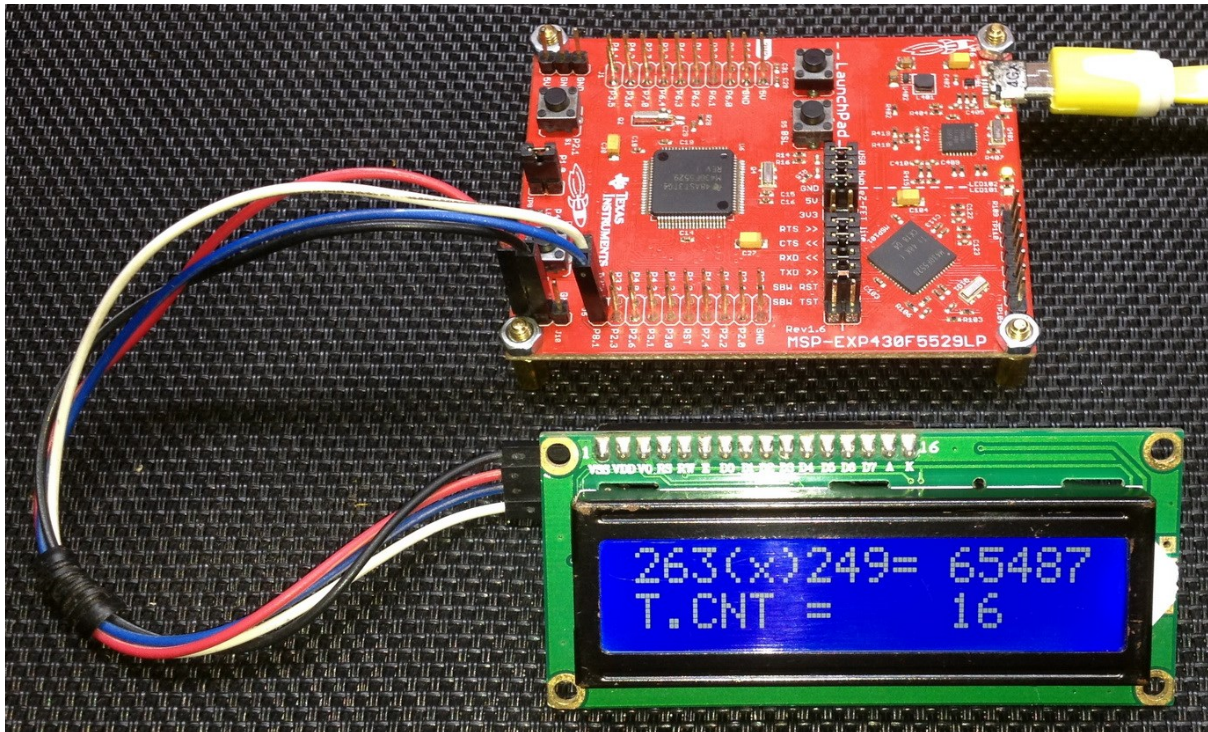
```
res = (((signed long)num1) * ((signed long)num2));
```

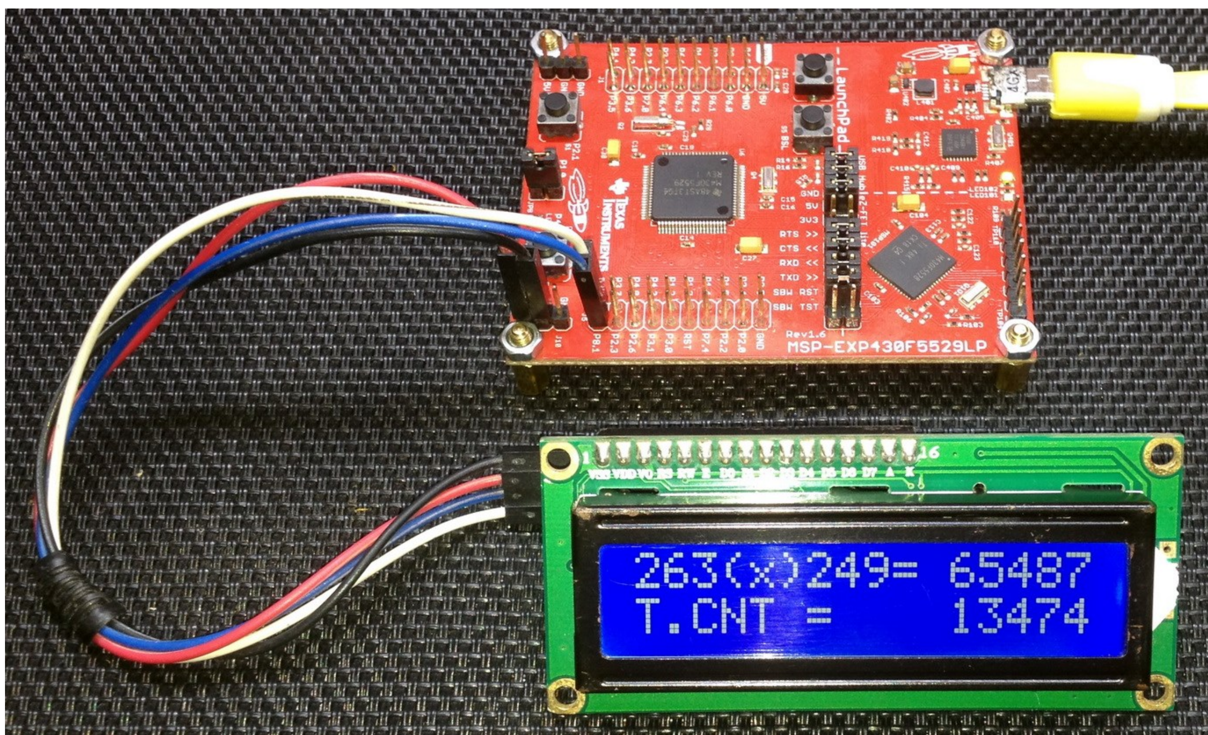
Again, the hardware multiplier is used for computing the same calculation and again it took 17 ticks or about 4µs. Unlike the first example with unsigned numbers, the time difference is small this time because when the compiler sees multiplication operator, it performs multiplication in assembly/ machine language level. Thus, time is reduced significantly but still it is not as fast as hardware-based multiplication.

Demo







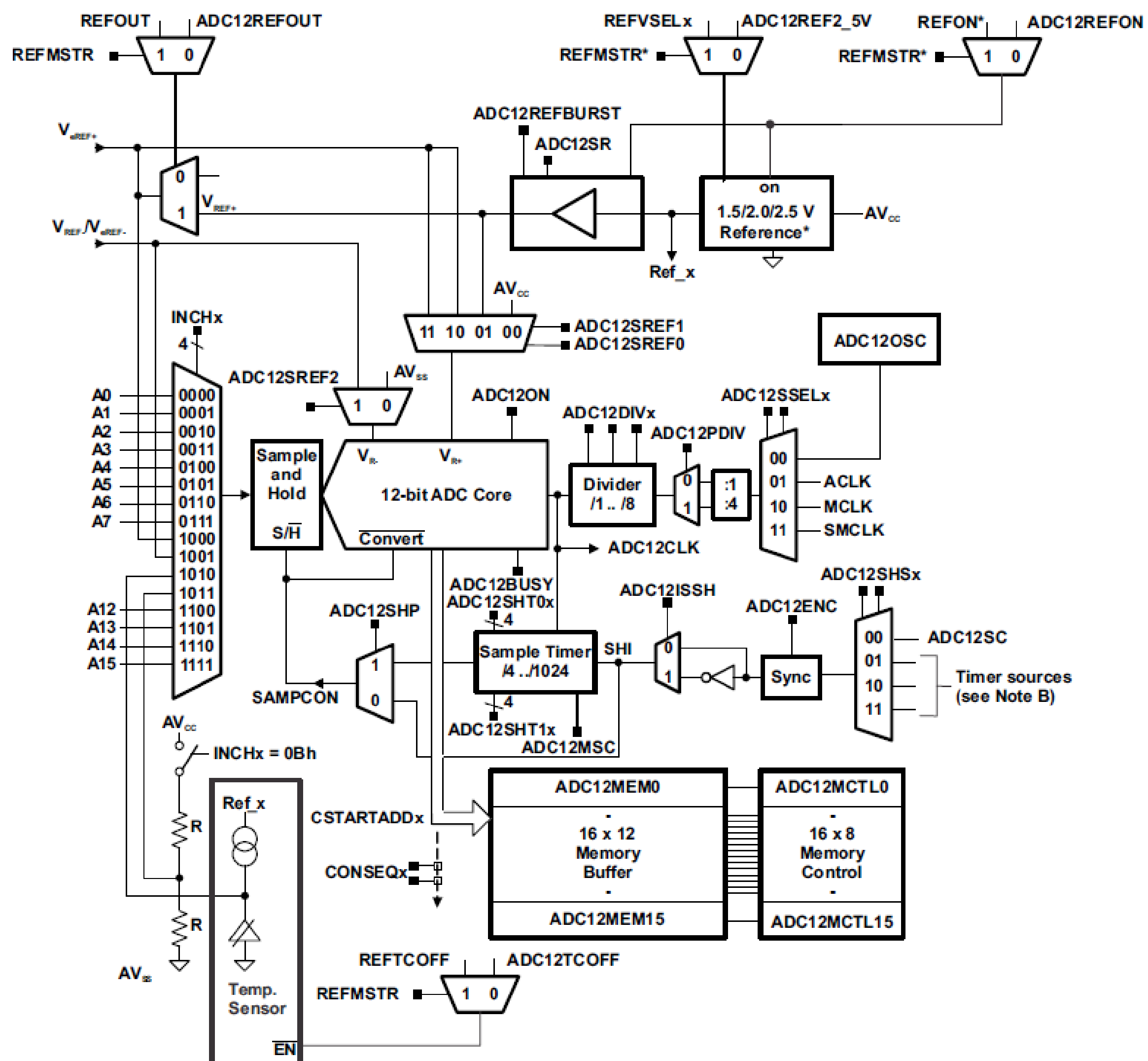


Demo video: https://youtu.be/Mjdlla198Q8

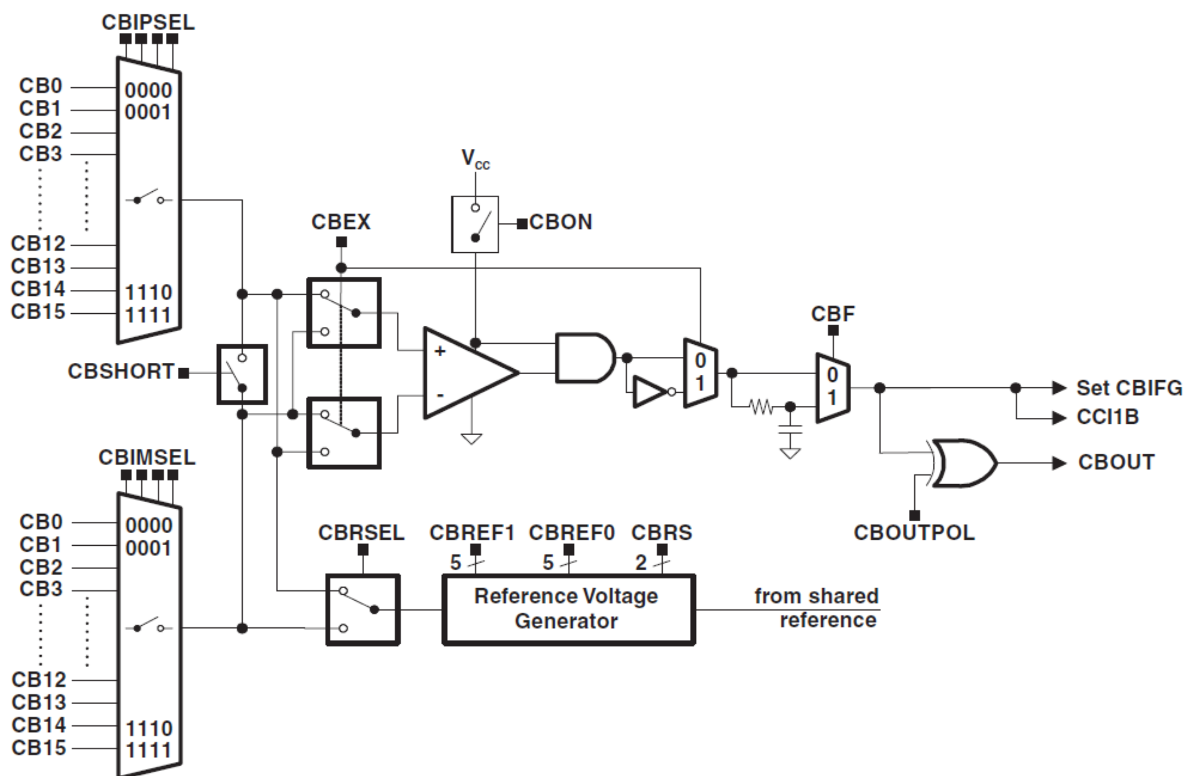
Analogue Hardware Overview

Except ***Digital-to-Analog Converter (DAC)***, MSP430F5529 microcontroller offers two major analogue hardware peripherals – ***ADC12_A*** and ***Comp_B+***. Additionally, there is an on-chip reference voltage generator called ***REF*** module that can generate pretty accurate voltage references - 1.5V, 2.0V and 2.5V reference voltages.

As the name suggests ADC12_A is a fast 12-bit ***Analogue-to-Digital Converter (ADC)***. Since it has higher resolution compared to other typical 8-bit and 10-bit ADCs of other microcontrollers, it much more precise when it comes to reading analogue voltages. MSP430s usually pack SAR ADCs and the ADC12_A of MSP430F559 is a 200+ ksps ***Successive Approximation (SAR)*** ADC. ADC12_A has an internal temperature sensor and REF module's reference voltages can be used with ADC12_A for more precise measurements. It is also possible to use external sources. Positive and negative references can be independently selected. Sample-and-hold circuitry offers programmable sampling periods via timers or software. In MSP430F5529, there are 16 individually configurable input channels - 12 external input channels and 4 internal input channels. There also 16 buffer registers to store AD conversion data. Since MSP430F5529 hosts a ***Direct-Memory-Access (DMA)*** peripheral, it can be combined with ADC12_A and we can get some more interesting effects.



A comparator compares two analogue voltage levels. This comparison results in an indication of which signal is at a higher/lower voltage level than the other. In simple terms, it is a one-bit ADC. Though it may look that a comparator is unnecessary when we have a good built-in ADC, it is otherwise. A comparator is a very important analogue building block. It is helpful in places where knowing voltage levels is more important the voltage itself. This makes it faster than an ADC in such cases. Owing to this a whole lot of electronics is based on it. Examples of such electronics include oscillators, level sensing, VU meters, capacitive touch sensing, measurement devices, etc. A LC meter is a perfect example. A LC meter is usually based on an oscillator. This oscillator uses a comparator. Its frequency varies with the L and C components, oscillating at a fixed frequency with known L and C values. Measuring frequency shifts as a result of changing L/C values leads us to measure unknown L/C effectively.



Another typical application of comparators is low battery detection. In fact, brownout detection circuit of microcontrollers uses a comparator to detect low voltage brownout level.

REF Module and ADC12 in Polling Mode

Good ADC performance and accurate readings depend on good reference sources. The stable these reference sources are with respect to changes in temperature and input voltage fluctuations, the better the ADC readings. MSP430F5529 is equipped with a built-in reference source generator module called ***REF*** module that is able to generate 1.5V, 2.0V and 2.5V with good precision. REF module has good temperature coefficients and power-saving features. REF module doesn't restrict us from using external reference sources but since they are available internally and factory trimmed, it is better to use them to reduce external parts count and avoid unnecessary hassle. The highlighted region in ADC12's block diagram shown below shows the components of MSP430F5529's REF module. A quick view reveals how they are connected with the ADC.

ADC12 is perhaps simply understood without the use of DMA and interrupts. ADC12 is configured and started. Readings of ADC channels are stored in ADC memory locations after conversions. Reading ADC memories give out last conversion results. In this example, we will see exactly that and we will be reading MSP430's internal temperature sensor with the ADC.

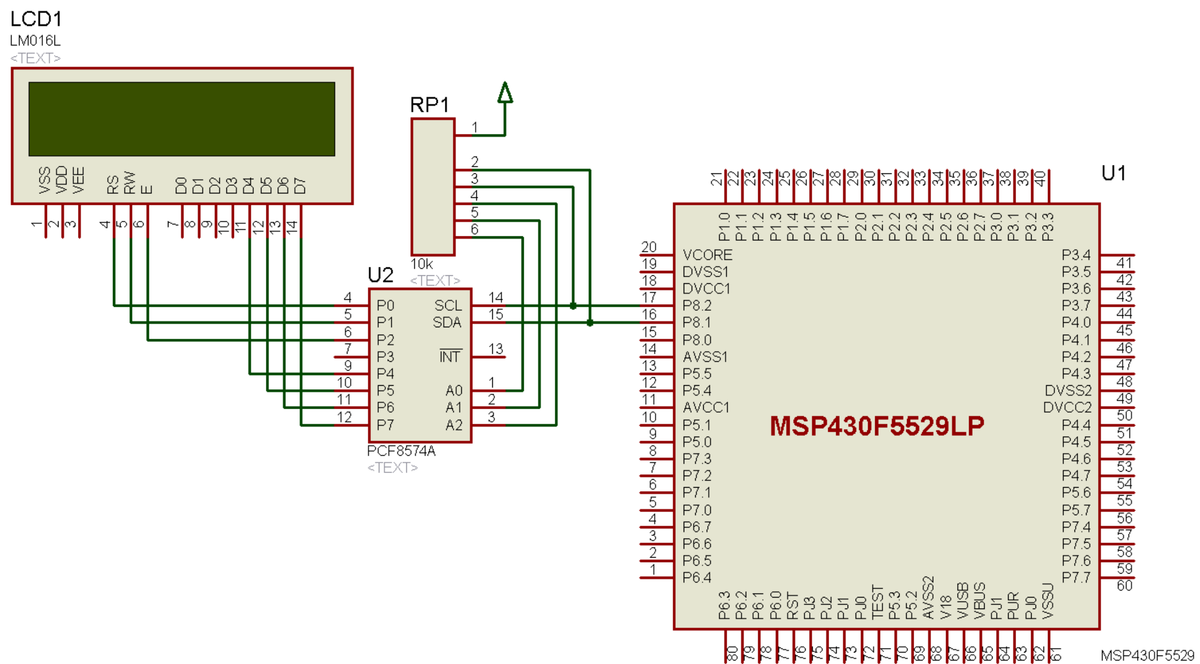
Code Example

```
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"
#include "lcd_print.h"


void clock_init(void);
void GPIO_init(void);
void ADC12_init(void);
void REF_init(void);


void main(void)
{
    signed int ADC_Count = 0;
    signed long temp = 0;

    WDT_A_hold(WDT_A_BASE);

    clock_init();
    GPIO_init();
    REF_init();
    ADC12_init();

    LCD_init();
    LCD_clear_home();
    load_custom_symbol();

    LCD_goto(0, 0);
    LCD_putstr("ADC :");

    LCD_goto(0, 1);
    LCD_putstr("T/ C:");
    print_symbol(2, 1, 0);

    while(1)
    {
        ADC_Count = ADC12_A_getResults(ADC12_A_BASE,
                                       ADC12_A_MEMORY_0);

        temp = ((((signed long)ADC_Count - 1855) * 667) / 4096);

        print_I(11, 0, ADC_Count);
        print_I(13, 1, temp);

        delay_ms(200);
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
```

```
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_2);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);
}



void ADC12_init(void)
{
    ADC12_A_configureMemoryParam configureMemoryParam = {0};

    ADC12_A_init(ADC12_A_BASE,
                 ADC12_A_SAMPLEHOLDSOURCE_SC,
                 ADC12_A_CLOCKSOURCE_ACLK,
                 ADC12_A_CLOCKDIVIDER_1);

    ADC12_A_setupSamplingTimer(ADC12_A_BASE,
                               ADC12_A_CYCLEHOLD_768_CYCLES,
                               ADC12_A_CYCLEHOLD_4_CYCLES,
                               ADC12_A_MULTIPLESAMPLESENABLE);

    ADC12_A_setResolution(ADC12_A_BASE,
                          ADC12_A_RESOLUTION_12BIT);

    configureMemoryParam.memoryBufferControlIndex = ADC12_A_MEMORY_0;
    configureMemoryParam.inputSourceSelect = ADC12_A_INPUT_TEMPSENSOR;
    configureMemoryParam.positiveRefVoltageSourceSelect = ADC12_A_VREFPOS_INT;
    configureMemoryParam.negativeRefVoltageSourceSelect = ADC12_A_VREFNEG_AVSS;
    configureMemoryParam.endOfSequence = ADC12_A_NOTENDOFSEQUENCE;

    ADC12_A_configureMemory(ADC12_A_BASE,
                            &configureMemoryParam);

    ADC12_A_enable(ADC12_A_BASE);

    ADC12_A_startConversion(ADC12_A_BASE,
                            ADC12_A_MEMORY_0,
                            ADC12_A_REPEATED_SINGLECHANNEL);
}
```

```
void REF_init(void)
{
    while(REF_ACTIVE == Ref_isRefGenBusy(REF_BASE));

    Ref_setReferenceVoltage(REF_BASE,
                            REF_VREF1_5V);

    Ref_enableReferenceVoltage(REF_BASE);

    __delay_cycles(100);
}
```

Hardware Setup



Explanation

Let's see first how the REF module is configured. REF module is not clock dependent as like other hardware and doesn't require use of any external pin.

```
void REF_init(void)
{
    while(REF_ACTIVE == Ref_isRefGenBusy(REF_BASE));
    Ref_setReferenceVoltage(REF_BASE, REF_VREF1_5V);
    Ref_enableReferenceVoltage(REF_BASE);
    __delay_cycles(100);
}
```

Using REF module is very easy. We first check its state. We set our desired reference voltage. Here we need the 1.5V reference. After selection, we enable it and wait for some time to get it settled.

```
void ADC12_init(void)
{
    ADC12_A_configureMemoryParam configureMemoryParam = {0};

    ADC12_A_init(ADC12_A_BASE,
                 ADC12_A_SAMPLEHOLDSOURCE_SC,
                 ADC12_A_CLOCKSOURCE_ACLK,
                 ADC12_A_CLOCKDIVIDER_1);

    ADC12_A_setupSamplingTimer(ADC12_A_BASE,
                               ADC12_A_CYCLEHOLD_768_CYCLES,
                               ADC12_A_CYCLEHOLD_4_CYCLES,
                               ADC12_A_MULTIPLESAMPLESENABLE);

    ADC12_A_setResolution(ADC12_A_BASE, ADC12_A_RESOLUTION_12BIT);

    configureMemoryParam.memoryBufferControlIndex = ADC12_A_MEMORY_0;
    configureMemoryParam.inputSourceSelect = ADC12_A_INPUT_TEMPSENSOR;
    configureMemoryParam.positiveRefVoltageSourceSelect = ADC12_A_VREFPOS_INT;
    configureMemoryParam.negativeRefVoltageSourceSelect =
ADC12_A_VREFNEG_AVSS;
    configureMemoryParam.endOfSequence = ADC12_A_NOTENDOFSEQUENCE;

    ADC12_A_configureMemory(ADC12_A_BASE, &configureMemoryParam);

    ADC12_A_enable(ADC12_A_BASE);

    ADC12_A_startConversion(ADC12_A_BASE,
                            ADC12_A_MEMORY_0,
                            ADC12_A_REPEATED_SINGLECHANNEL);
}
```

We are not using any external channel and so there is no special pin setup for ADC12.

As with any hardware, the source of ADC12's clock is set first. Here the clock source is unscaled ACLK.

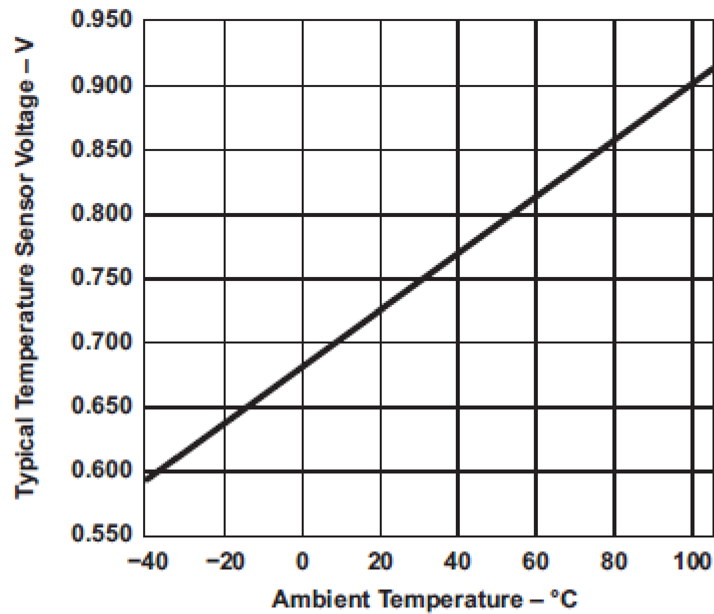
```
UCS_initClockSignal(UCS_ACLK, UCS_XT1CLK_SELECT, UCS_CLOCK_DIVIDER_1);
```

ACLK is fed with an external 32.768 kHz XT1 crystal running at 32.768 kHz.

Next, we set the sampling timer by defining the sample-hold times. We also set the ADC's resolution to 12 bits. We can also set 8 and 10 resolution if required.

We also have to let the ADC know which ADC channel to read, where to store the readings and what the reference voltages are for the ADC block. Note that the reference voltage source is the 1.5V source.

We, finally, start the ADC and leave it in repeated single channel conversion mode. In this way, the ADC will continuously measure our desired channel and store readings in an ADC memory location (here Memory 0).

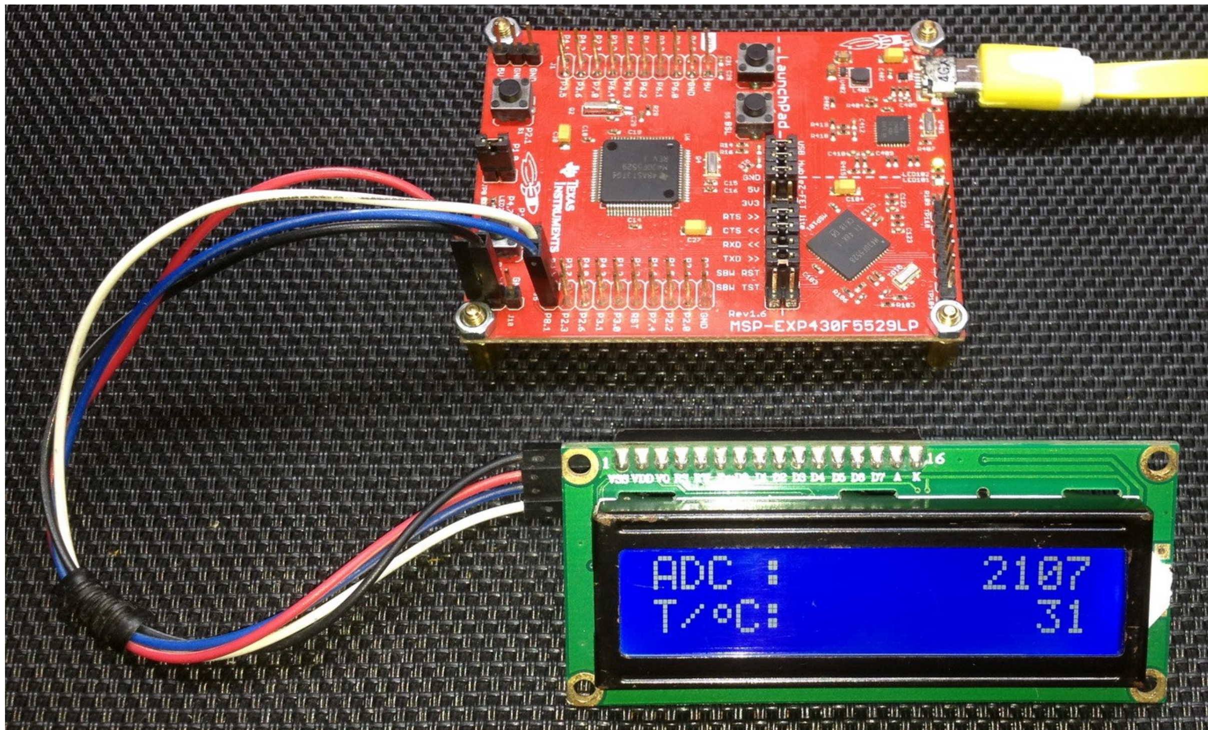


From the graph shown above, we can see that the voltage (ultimately ADC count) vs temperature relationship of the internal temperature sensor is a perfect straight-line and so a linear equation will be needed to describe this relationship. Now if we know the ADC count of the internal temperature sensor, we can determine MSP430's core temperature.

In the main loop, as I stated earlier, ADC memory 0 location is read and the reading is converted to temperature. The ADC count and the temperature are then shown on an LCD display.

```
ADC_Count = ADC12_A_getResults(ADC12_A_BASE, ADC12_A_MEMORY_0);

temp = ((((signed long)ADC_Count - 1855) * 667) / 4096);
```

Demo



Demo video: https://youtu.be/yJVYkq0gBxI

ADC12 Interrupt

ADC interrupt is as important as timer and communication interrupts. We can do other tasks while ADC is performing a conversion.

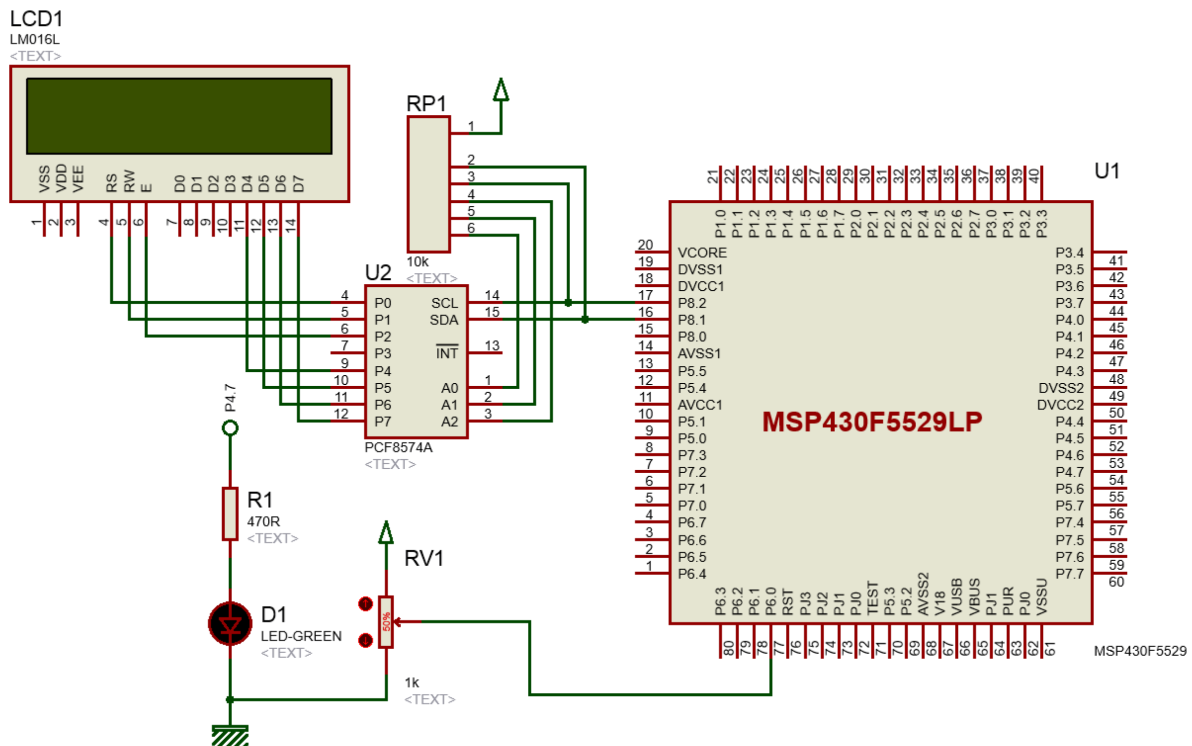
Code Example

```
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"
#include "lcd_print.h"


unsigned long cnt = 0;


void clock_init(void);
void GPIO_init(void);
void ADC12_init(void);


#pragma vector = ADC12_VECTOR
__interrupt void ADC12ISR (void)
{
    switch (__even_in_range(ADC12IV, 34))
    {
        case  0: break;   //Vector  0:  No interrupt
        case  2: break;   //Vector  2:  ADC overflow
        case  4: break;   //Vector  4:  ADC timing overflow
        case  6:          //Vector  6:  ADC12IFG0
        {
            cnt = ADC12_A_getResults(ADC12_A_BASE,
                                     ADC12_A_MEMORY_0);
            break;
        }
        case  8: break;   //Vector  8:  ADC12IFG1
        case 10: break;   //Vector 10:  ADC12IFG2
        case 12: break;   //Vector 12:  ADC12IFG3
        case 14: break;   //Vector 14:  ADC12IFG4
        case 16: break;   //Vector 16:  ADC12IFG5
        case 18: break;   //Vector 18:  ADC12IFG6
        case 20: break;   //Vector 20:  ADC12IFG7
        case 22: break;   //Vector 22:  ADC12IFG8
        case 24: break;   //Vector 24:  ADC12IFG9
        case 26: break;   //Vector 26:  ADC12IFG10
        case 28: break;   //Vector 28:  ADC12IFG11
        case 30: break;   //Vector 30:  ADC12IFG12
        case 32: break;   //Vector 32:  ADC12IFG13
        case 34: break;   //Vector 34:  ADC12IFG14
        default: break;
    }
}


void main(void)
{
    unsigned long volts = 0;

    WDT_A_hold(WDT_A_BASE);

    clock_init();
```

```
    GPIO_init();
    ADC12_init();

    LCD_init();
    LCD_clear_home();

    LCD_goto(0, 0);
    LCD_putstr("ADC Count:");

    LCD_goto(0, 1);
    LCD_putstr("Volts/mV :");

    while(1)
    {
        volts = ((cnt * 3300) / 4095);
        print_I(11, 0, cnt);
        print_I(11, 1, volts);
        delay_ms(100);
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_2);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P6,
                                               GPIO_PIN0);
}
```

```
void ADC12_init(void)
{
    ADC12_A_configureMemoryParam configureMemoryParam = {0};

    ADC12_A_init(ADC12_A_BASE,
                 ADC12_A_SAMPLEHOLDSOURCE_SC,
                 ADC12_A_CLOCKSOURCE_ACLK,
                 ADC12_A_CLOCKDIVIDER_1);

    ADC12_A_setupSamplingTimer(ADC12_A_BASE,
                               ADC12_A_CYCLEHOLD_768_CYCLES,
                               ADC12_A_CYCLEHOLD_4_CYCLES,
                               ADC12_A_MULTIPLESAMPLESENABLE);

    ADC12_A_setResolution(ADC12_A_BASE,
                          ADC12_A_RESOLUTION_12BIT);

    configureMemoryParam.memoryBufferControlIndex = ADC12_A_MEMORY_0;
    configureMemoryParam.inputSourceSelect = ADC12_A_INPUT_A0;
    configureMemoryParam.positiveRefVoltageSourceSelect = ADC12_A_VREFPOS_AVCC;
    configureMemoryParam.negativeRefVoltageSourceSelect = ADC12_A_VREFNEG_AVSS;
    configureMemoryParam.endOfSequence = ADC12_A_NOTENDOFSEQUENCE;

    ADC12_A_configureMemory(ADC12_A_BASE,
                            &configureMemoryParam);

    ADC12_A_clearInterrupt(ADC12_A_BASE,
                           ADC12IFG0);

    ADC12_A_enableInterrupt(ADC12_A_BASE,
                            ADC12IE0);

    __enable_interrupt();

    ADC12_A_enable(ADC12_A_BASE);

    ADC12_A_startConversion(ADC12_A_BASE,
                            ADC12_A_MEMORY_0,
                            ADC12_A_REPEATED_SINGLECHANNEL);
}
```

Hardware Setup



Explanation

This example is similar to the last one in many aspects. I won't be going through all settings. I'll only highlight the key differences.

This time an external ADC channel is used and so we have to initialize the alternative role of its pin.

```
GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P6, GPIO_PIN0);
```

In the ADC12 initialization, the external input pin or the ADC channel's physical pin is set after its GPIO initialization. Unlike the last ADC example in which we used internal 1.5V reference source, the positive reference source for ADC this time is set to *AVCC*. *AVSS* is the negative voltage source for both examples. *AVCC* is same as the system bus voltage, i.e. 3.3V.

```
configureMemoryParam.inputSourceSelect = ADC12_A_INPUT_A0;
configureMemoryParam.positiveRefVoltageSourceSelect = ADC12_A_VREFPOS_AVCC;
configureMemoryParam.negativeRefVoltageSourceSelect = ADC12_A_VREFNEG_AVSS;
```

Since we are using interrupt method, relevant interrupt flag is cleared first and then enabled.

```
ADC12_A_clearInterrupt(ADC12_A_BASE, ADC12IFG0);
ADC12_A_enableInterrupt(ADC12_A_BASE, ADC12IE0);
__enable_interrupt();
```

In this case the 0th ADC interrupt flag is used since the channel to be read is the 0th channel of the ADC. Don't confuse interrupt vector number with interrupt flag number. Interrupt vector number for 0th ADC interrupt flag is 6.

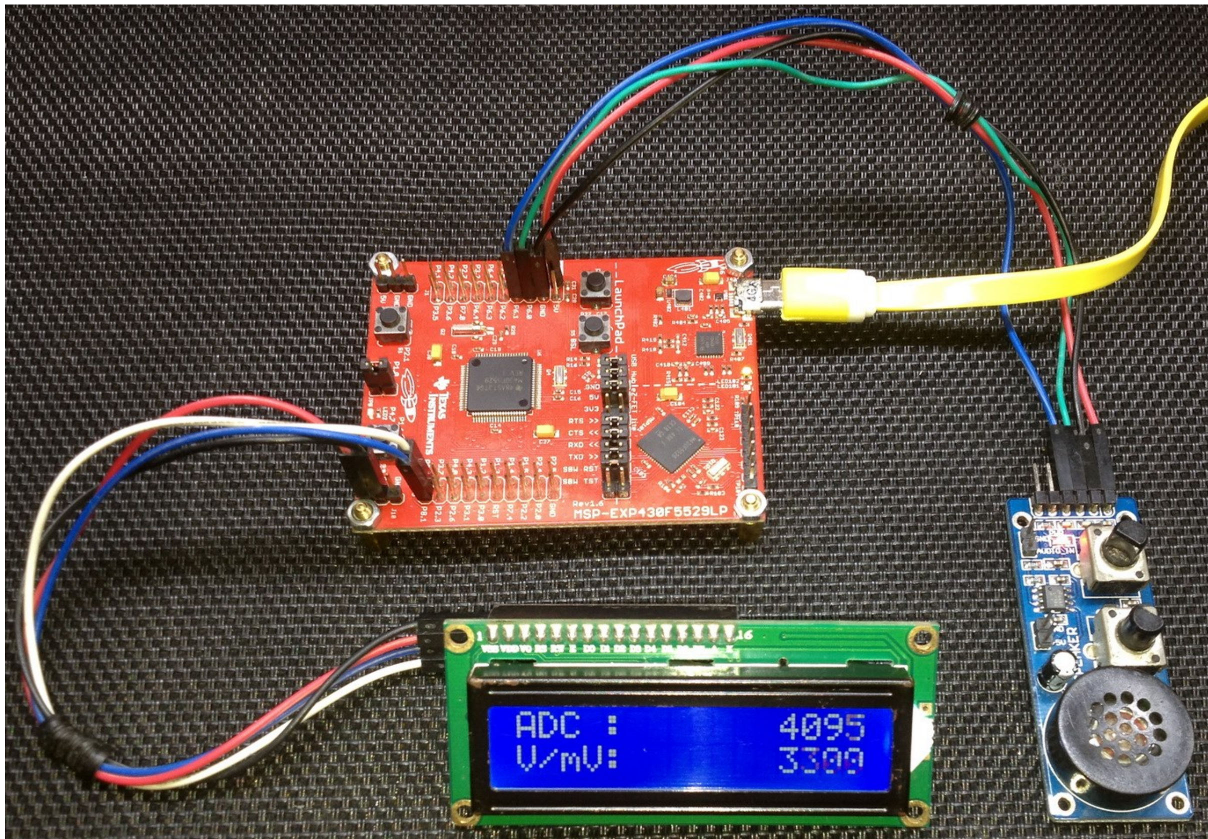
```
#pragma vector = ADC12_VECTOR
__interrupt void ADC12ISR (void)
{
    switch (__even_in_range(ADC12IV, 34))
    {
        case  0: break;   //Vector  0:  No interrupt
        case  2: break;   //Vector  2:  ADC overflow
        case  4: break;   //Vector  4:  ADC timing overflow
        case  6:          //Vector  6:  ADC12IFG0
        {
            cnt = ADC12_A_getResults(ADC12_A_BASE,
                                     ADC12_A_MEMORY_0);
            break;
        }
        case  8: break;   //Vector  8:  ADC12IFG1
        case 10: break;   //Vector 10:  ADC12IFG2
        case 12: break;   //Vector 12:  ADC12IFG3
        case 14: break;   //Vector 14:  ADC12IFG4
        case 16: break;   //Vector 16:  ADC12IFG5
        case 18: break;   //Vector 18:  ADC12IFG6
        case 20: break;   //Vector 20:  ADC12IFG7
        case 22: break;   //Vector 22:  ADC12IFG8
        case 24: break;   //Vector 24:  ADC12IFG9
        case 26: break;   //Vector 26:  ADC12IFG10
        case 28: break;   //Vector 28:  ADC12IFG11
        case 30: break;   //Vector 30:  ADC12IFG12
        case 32: break;   //Vector 32:  ADC12IFG13
        case 34: break;   //Vector 34:  ADC12IFG14
        default: break;
    }
}
```

ADC interrupt triggers when a new result is loaded in ADC memory location. Inside the interrupt ADC memory location 0 is read to get the conversion result, i.e. ADC count. When the ADC memory is accessed, its interrupt flag is automatically cleared.

In the main loop, the ADC count derived from the ADC interrupt is converted to voltage. Both the voltage and ADC count are displayed on an alphanumerical LCD.

```
volts = ((cnt * 3300) / 4095);
print_I(11, 0, cnt);
print_I(11, 1, volts);
delay_ms(100);
```

Demo



Demo video: https://youtu.be/iAb3Fth1dXM

ADC12 Sampling Multiple Input

Till now, we have seen only one ADC channel in action. We have seen examples of single internal and external channels only. However, in real-life and in many occasions, we may need the use of more than one ADC channel. For example, we will need multiple ADC channels when crafting an Uninterrupted Power Supply (UPS). In an UPS, we will be needing to measure battery voltage, battery charging and discharging current, AC output voltage and current, internal temperature and so on. In such cases, multiple ADC channels are musts. This example demonstrates how to use multiple ADC channels of a MSP430F5529 microcontroller.

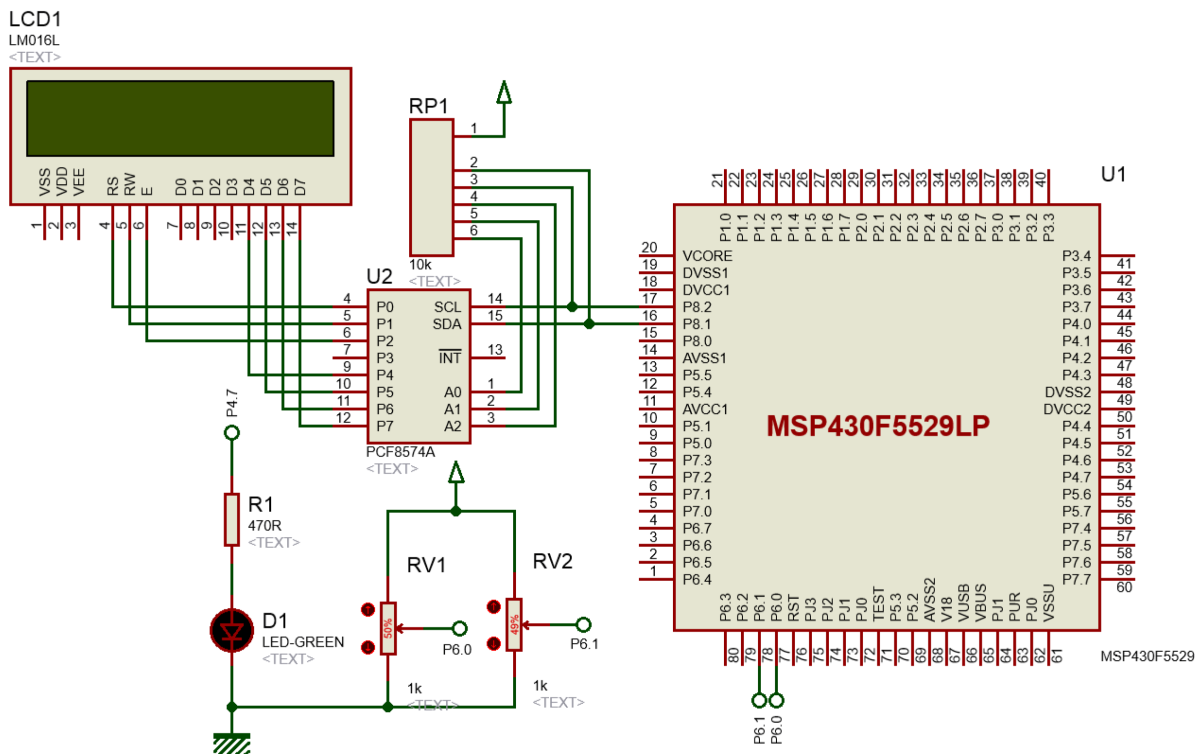
Code Example

```
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"
#include "lcd_print.h"


unsigned long res[2] = {0, 0};


void clock_init(void);
void GPIO_init(void);
void ADC12_init(void);


#pragma vector = ADC12_VECTOR
__interrupt void ADC12ISR (void)
{
    switch (__even_in_range(ADC12IV, 34))
    {
        case  0: break;   //Vector  0:  No interrupt
        case  2: break;   //Vector  2:  ADC overflow
        case  4: break;   //Vector  4:  ADC timing overflow
        case  6: break;
         {
              res[0] = ADC12_A_getResults(ADC12_A_BASE,
                                          ADC12_A_MEMORY_0);
            break;
         }                    //Vector  6:  ADC12IFG0
        case  8:
        {
            res[1] = ADC12_A_getResults(ADC12_A_BASE,
                                        ADC12_A_MEMORY_1);
            break;
        }                 //Vector  8:  ADC12IFG1
        case 10: break;   //Vector 10:  ADC12IFG2
        case 12: break;   //Vector 12:  ADC12IFG3
        case 14: break;   //Vector 14:  ADC12IFG4
        case 16: break;   //Vector 16:  ADC12IFG5
        case 18: break;   //Vector 18:  ADC12IFG6
        case 20: break;   //Vector 20:  ADC12IFG7
        case 22: break;   //Vector 22:  ADC12IFG8
        case 24: break;   //Vector 24:  ADC12IFG9
        case 26: break;   //Vector 26:  ADC12IFG10
        case 28: break;   //Vector 28:  ADC12IFG11
        case 30: break;   //Vector 30:  ADC12IFG12
        case 32: break;   //Vector 32:  ADC12IFG13
        case 34: break;   //Vector 34:  ADC12IFG14
        default: break;
```

```
    }
}


void main(void)
{
    WDT_A_hold(WDT_A_BASE);

    clock_init();
    GPIO_init();
    ADC12_init();

    LCD_init();
    LCD_clear_home();

    LCD_goto(0, 0);
    LCD_putstr("CH0:");

    LCD_goto(0, 1);
    LCD_putstr("CH1:");

    while(1)
    {

        print_I(11, 0, res[0]);
        print_I(11, 1, res[1]);
        delay_ms(100);
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_2);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}
```

```
void GPIO_init(void)
{
    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P6,
                                               (GPIO_PIN0 | GPIO_PIN1));
}

void ADC12_init(void)

{
    ADC12_A_configureMemoryParam CH0_configureMemoryParam = {0};
    ADC12_A_configureMemoryParam CH1_configureMemoryParam = {0};

    CH0_configureMemoryParam.memoryBufferControlIndex = ADC12_A_MEMORY_0;
    CH0_configureMemoryParam.inputSourceSelect = ADC12_A_INPUT_A0;
    CH0_configureMemoryParam.positiveRefVoltageSourceSelect = ADC12_A_VREFPOS_AVCC;
    CH0_configureMemoryParam.negativeRefVoltageSourceSelect = ADC12_A_VREFNEG_AVSS;
    CH0_configureMemoryParam.endOfSequence = ADC12_A_NOTENDOFSEQUENCE;

    CH1_configureMemoryParam.memoryBufferControlIndex = ADC12_A_MEMORY_1;
    CH1_configureMemoryParam.inputSourceSelect = ADC12_A_INPUT_A1;
    CH1_configureMemoryParam.positiveRefVoltageSourceSelect = ADC12_A_VREFPOS_AVCC;
    CH1_configureMemoryParam.negativeRefVoltageSourceSelect = ADC12_A_VREFNEG_AVSS;
    CH1_configureMemoryParam.endOfSequence = ADC12_A_ENDOFSEQUENCE;

    ADC12_A_init(ADC12_A_BASE,
                 ADC12_A_SAMPLEHOLDSOURCE_SC,
                 ADC12_A_CLOCKSOURCE_ACLK,
                 ADC12_A_CLOCKDIVIDER_1);

    ADC12_A_setupSamplingTimer(ADC12_A_BASE,
                               ADC12_A_CYCLEHOLD_256_CYCLES,
                               ADC12_A_CYCLEHOLD_4_CYCLES,
                               ADC12_A_MULTIPLESAMPLESENABLE);

    ADC12_A_setResolution(ADC12_A_BASE,
                          ADC12_A_RESOLUTION_12BIT);

    ADC12_A_configureMemory(ADC12_A_BASE,
                            &CH0_configureMemoryParam);

    ADC12_A_configureMemory(ADC12_A_BASE,
                            &CH1_configureMemoryParam);

    ADC12_A_clearInterrupt(ADC12_A_BASE,
                           ADC12IFG0);

    ADC12_A_enableInterrupt(ADC12_A_BASE,
                            ADC12IE0);

    ADC12_A_clearInterrupt(ADC12_A_BASE,
                           ADC12IFG1);

    ADC12_A_enableInterrupt(ADC12_A_BASE,
                            ADC12IE1);

    __enable_interrupt();

    ADC12_A_enable(ADC12_A_BASE);

    ADC12_A_startConversion(ADC12_A_BASE,
                            ADC12_A_MEMORY_0,
                            ADC12_A_REPEATED_SEQOFCHANNELS);
}
```

Hardware Setup



Explanation

ADC12's settings have two sections. One is common and the other is channel dependent.

Common settings include ADC's clock setup, resolution and sample-hold timer settings. These will be applicable for all channels.

```
ADC12_A_init(ADC12_A_BASE,
             ADC12_A_SAMPLEHOLDSOURCE_SC,
             ADC12_A_CLOCKSOURCE_ACLK,
             ADC12_A_CLOCKDIVIDER_1);

ADC12_A_setupSamplingTimer(ADC12_A_BASE,
                           ADC12_A_CYCLEHOLD_256_CYCLES,
                           ADC12_A_CYCLEHOLD_4_CYCLES,
                           ADC12_A_MULTIPLESAMPLESENABLE);

ADC12_A_setResolution(ADC12_A_BASE, ADC12_A_RESOLUTION_12BIT);
```

Channel parameters define ADC memory location, references and sequence info. Additionally, if interrupts are used, they need to applied separately.

```
CH0_configureMemoryParam.memoryBufferControlIndex = ADC12_A_MEMORY_0;
CH0_configureMemoryParam.inputSourceSelect = ADC12_A_INPUT_A0;
CH0_configureMemoryParam.positiveRefVoltageSourceSelect = ADC12_A_VREFPOS_AVCC;
CH0_configureMemoryParam.negativeRefVoltageSourceSelect = ADC12_A_VREFNEG_AVSS;
CH0_configureMemoryParam.endOfSequence = ADC12_A_NOTENDOFSEQUENCE;

CH1_configureMemoryParam.memoryBufferControlIndex = ADC12_A_MEMORY_1;
CH1_configureMemoryParam.inputSourceSelect = ADC12_A_INPUT_A1;
```

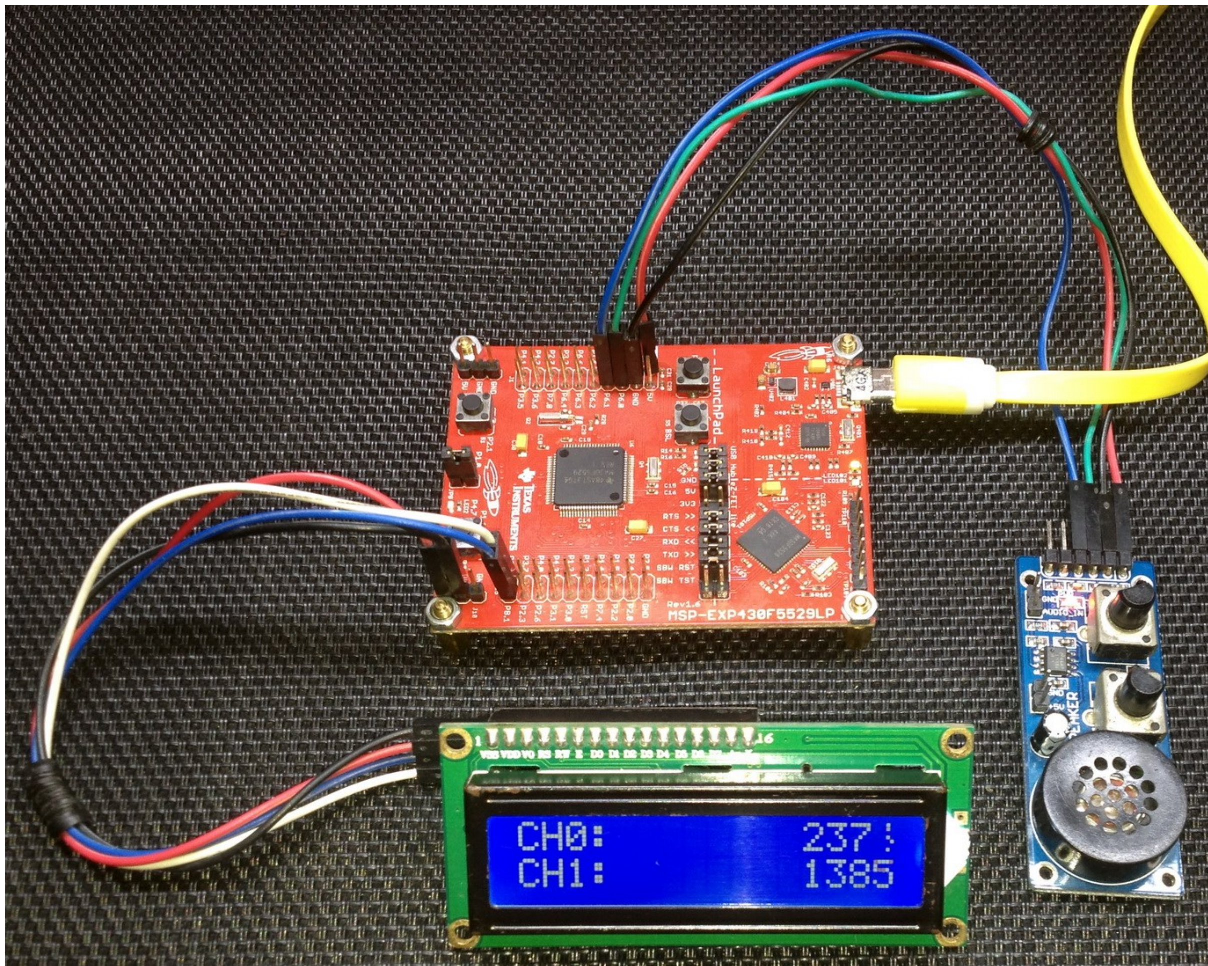
```
CH1_configureMemoryParam.positiveRefVoltageSourceSelect = ADC12_A_VREFPOS_AVCC;
CH1_configureMemoryParam.negativeRefVoltageSourceSelect = ADC12_A_VREFNEG_AVSS;
CH1_configureMemoryParam.endOfSequence = ADC12_A_ENDOFSEQUENCE;

....

ADC12_A_clearInterrupt(ADC12_A_BASE, ADC12IFG0);
ADC12_A_enableInterrupt(ADC12_A_BASE, ADC12IE0);

ADC12_A_configureMemory(ADC12_A_BASE, &CH0_configureMemoryParam);

ADC12_A_clearInterrupt(ADC12_A_BASE, ADC12IFG1);
ADC12_A_enableInterrupt(ADC12_A_BASE, ADC12IE1);

ADC12_A_configureMemory(ADC12_A_BASE, &CH1_configureMemoryParam);
```

At this point, I would like to highlight the concept of sequence. When multiple ADC channels are used, ADC conversions are done sequentially, i.e. one after another. In our code, we have to specify one channel as the end of sequence while denoting other channels as no end of sequence. In this way, channels are systematically queued.

ADC reading process is same as the one we saw in the ADC interrupt example. The only exception is the usage of two vectors as two channels are in different memory planes.

```
#pragma vector = ADC12_VECTOR
__interrupt void ADC12ISR (void)
{
    switch (__even_in_range(ADC12IV, 34))
    {
        case  0: break;   //Vector  0:  No interrupt
        case  2: break;   //Vector  2:  ADC overflow
        case  4: break;   //Vector  4:  ADC timing overflow
        case  6: break;
         {
             res[0] = ADC12_A_getResults(ADC12_A_BASE,
                                        ADC12_A_MEMORY_0);
           break;
         }                  //Vector  6:  ADC12IFG0
        case  8:
        {
           res[1] = ADC12_A_getResults(ADC12_A_BASE,
                                        ADC12_A_MEMORY_1);
           break;
        }                 //Vector  8:  ADC12IFG1
        case 10: break;   //Vector 10:  ADC12IFG2
        case 12: break;   //Vector 12:  ADC12IFG3
        case 14: break;   //Vector 14:  ADC12IFG4
        case 16: break;   //Vector 16:  ADC12IFG5
        case 18: break;   //Vector 18:  ADC12IFG6
        case 20: break;   //Vector 20:  ADC12IFG7
        case 22: break;   //Vector 22:  ADC12IFG8
        case 24: break;   //Vector 24:  ADC12IFG9
        case 26: break;   //Vector 26:  ADC12IFG10
        case 28: break;   //Vector 28:  ADC12IFG11
        case 30: break;   //Vector 30:  ADC12IFG12
        case 32: break;   //Vector 32:  ADC12IFG13
        case 34: break;   //Vector 34:  ADC12IFG14
        default: break;
    }
}
```

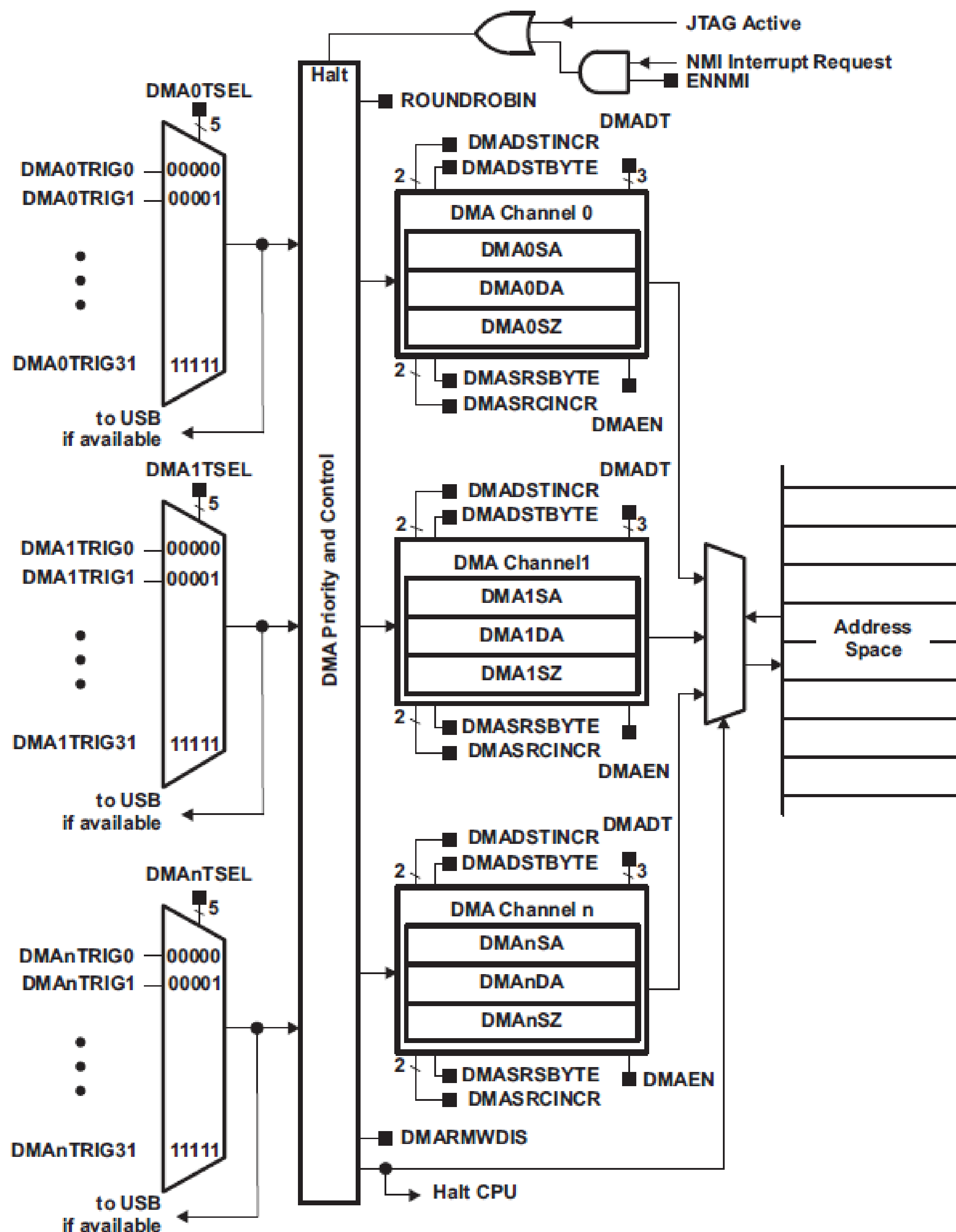
Demo



Video demo: https://youtu.be/03TEEj95lM8

ADC12 and Direct Memory Access (DMA) Module

Apart from interrupt method, there is another way to doing AD conversions without polling/waiting. This involves DMA-backed AD conversion. DMA hardware is not a part of the ADC. In simple terms, it is rather more like a separate data pipeline bus that can be used without much interaction from the CPU. DMA-based AD conversions are in that way very silent. DMA can send AD conversion readings directly to user-defined memory locations automatically without any intervention. DMA can also be used for other hardware like SPI and UART. Shown below is the DMA block diagram of MSP430F5529:



Code Example

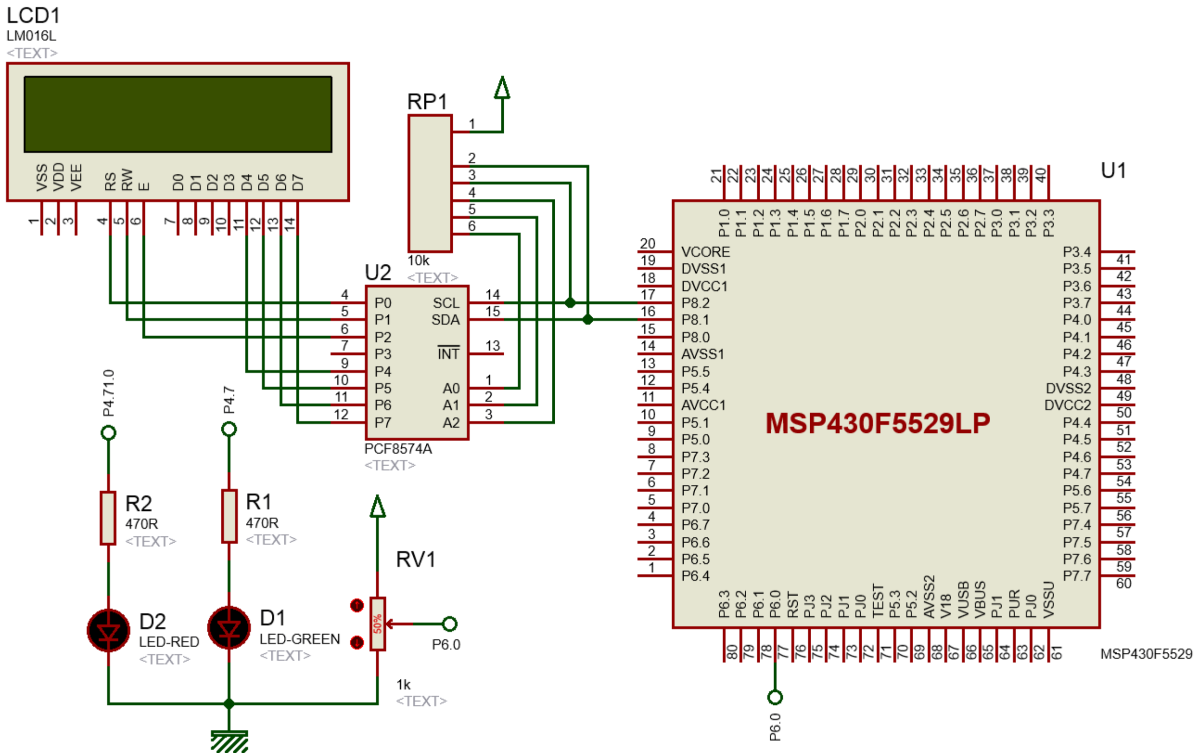
```
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"
#include "lcd_print.h"


void clock_init(void);
void GPIO_init(void);
void ADC12_init(void);
void DMA_HW_init(void);


uint16_t memory_location = 0x0000;


#pragma vector = DMA_VECTOR
__interrupt void DMA_ISR (void)
{
    switch (__even_in_range(DMAIV, 16))
    {
        case 0: break;  //None
        case 2:         //DMA0IFG = DMA Channel 0
        {
            GPIO_toggleOutputOnPin(GPIO_PORT_P1,
                                   GPIO_PIN0);

            break;
        }
        case 4: break;  //DMA1IFG = DMA Channel 1
        case 6: break;  //DMA2IFG = DMA Channel 2
        case 8: break;  //DMA3IFG = DMA Channel 3
        case 10: break; //DMA4IFG = DMA Channel 4
        case 12: break; //DMA5IFG = DMA Channel 5
        case 14: break; //DMA6IFG = DMA Channel 6
        case 16: break; //DMA7IFG = DMA Channel 7
        default: break;
    }
}


void main(void)
{
    unsigned int volt = 0;

    WDT_A_hold(WDT_A_BASE);

    clock_init();
    GPIO_init();
    ADC12_init();
    DMA_HW_init();

    LCD_init();
    LCD_clear_home();

    LCD_goto(0, 0);
    LCD_putstr("ADC :");

    LCD_goto(0, 1);
    LCD_putstr("V/mV:");

    while(1)
    {
```

```c
        volt = ((memory_location * 3300.0) / 4095.0);

        print_I(11, 0, memory_location);
        print_I(11, 1, volt);

        GPIO_toggleOutputOnPin(GPIO_PORT_P4,
                               GPIO_PIN7);
        delay_ms(200);
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_2);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P6,
                                               GPIO_PIN0);

    GPIO_setAsOutputPin(GPIO_PORT_P1,
                        GPIO_PIN0);

    GPIO_setDriveStrength(GPIO_PORT_P1,
                          GPIO_PIN0,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setDriveStrength(GPIO_PORT_P4,
                          GPIO_PIN7,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);
```

```
}


void ADC12_init(void)
{
    ADC12_A_configureMemoryParam configureMemoryParam = {0};

    ADC12_A_init(ADC12_A_BASE,
                 ADC12_A_SAMPLEHOLDSOURCE_SC,
                 ADC12_A_CLOCKSOURCE_ACLK,
                 ADC12_A_CLOCKDIVIDER_1);

    ADC12_A_setupSamplingTimer(ADC12_A_BASE,
                               ADC12_A_CYCLEHOLD_768_CYCLES,
                               ADC12_A_CYCLEHOLD_4_CYCLES,
                               ADC12_A_MULTIPLESAMPLESENABLE);

    ADC12_A_setResolution(ADC12_A_BASE,
                          ADC12_A_RESOLUTION_12BIT);

    configureMemoryParam.memoryBufferControlIndex = ADC12_A_MEMORY_0;
    configureMemoryParam.inputSourceSelect = ADC12_A_INPUT_A0;
    configureMemoryParam.positiveRefVoltageSourceSelect = ADC12_A_VREFPOS_AVCC;
    configureMemoryParam.negativeRefVoltageSourceSelect = ADC12_A_VREFNEG_AVSS;
    configureMemoryParam.endOfSequence = ADC12_A_NOTENDOFSEQUENCE;

    ADC12_A_configureMemory(ADC12_A_BASE,
                            &configureMemoryParam);

    ADC12_A_enable(ADC12_A_BASE);

    ADC12_A_startConversion(ADC12_A_BASE,
                            ADC12_A_MEMORY_0,
                            ADC12_A_REPEATED_SINGLECHANNEL);
}


void DMA_HW_init(void)
{
    DMA_initParam DMA_init_Param = {0};

    DMA_disableTransferDuringReadModifyWrite();

    DMA_init_Param.channelSelect = DMA_CHANNEL_0;
    DMA_init_Param.transferModeSelect = DMA_TRANSFER_REPEATED_SINGLE;
    DMA_init_Param.transferSize = 1;
    DMA_init_Param.triggerSourceSelect = DMA_TRIGGERSOURCE_24;
    DMA_init_Param.transferUnitSelect = DMA_SIZE_SRCWORD_DSTWORD;
    DMA_init_Param.triggerTypeSelect = DMA_TRIGGER_RISINGEDGE;

    DMA_init(&DMA_init_Param);

    DMA_setSrcAddress(DMA_CHANNEL_0,
                      ADC12_A_getMemoryAddressForDMA(ADC12_A_BASE,ADC12_A_MEMORY_0),
                      DMA_DIRECTION_UNCHANGED);

    DMA_setDstAddress(DMA_CHANNEL_0,
                      (uint32_t)(uintptr_t)&memory_location,
                      DMA_DIRECTION_INCREMENT);

    DMA_clearInterrupt(DMA_CHANNEL_0);

    DMA_enableInterrupt(DMA_CHANNEL_0);
```

```
    DMA_enableTransfers(DMA_CHANNEL_0);

    __enable_interrupt();
}
```

Hardware Setup



Explanation

ADC12 is configured just as we already did in past examples but no ADC12 interrupt is used. The new changes are related to DMA configuration.

We wish to transfer ADC data to our designated memory location with DMA hardware. Thus, we have to declare a word-sized global variable. We chose word size, i.e. 16-bit because ADC12 readings are typically 12-bit in size. This memory location will be used with DMA and that makes it important.

```
uint16_t memory_location = 0x0000;
```

DMA initialization requires a few parameters to be set in order to define the its characteristics.

```
void DMA_HW_init(void)
{
    DMA_initParam DMA_init_Param = {0};

    DMA_disableTransferDuringReadModifyWrite();

    DMA_init_Param.channelSelect = DMA_CHANNEL_0;
    DMA_init_Param.transferModeSelect = DMA_TRANSFER_REPEATED_SINGLE;
    DMA_init_Param.transferSize = 1;
    DMA_init_Param.triggerSourceSelect = DMA_TRIGGERSOURCE_24;
    DMA_init_Param.transferUnitSelect = DMA_SIZE_SRCWORD_DSTWORD;
    DMA_init_Param.triggerTypeSelect = DMA_TRIGGER_RISINGEDGE;

    DMA_init(&DMA_init_Param);

    DMA_setSrcAddress(DMA_CHANNEL_0,
                      ADC12_A_getMemoryAddressForDMA(ADC12_A_BASE, ADC12_A_MEMORY_0),
                      DMA_DIRECTION_UNCHANGED);

    DMA_setDstAddress(DMA_CHANNEL_0,
                      (uint32_t)(uintptr_t)&memory_location,
                      DMA_DIRECTION_INCREMENT);

    DMA_clearInterrupt(DMA_CHANNEL_0);

    DMA_enableInterrupt(DMA_CHANNEL_0);

    DMA_enableTransfers(DMA_CHANNEL_0);

    __enable_interrupt();
}
```

The operation of DMA peripheral is similar to that of a water pipeline. In a water pipeline, we have to define source of water and its destination along with flow direction and frequency of flow. We also have to define the pipe size (diameter) in order to avoid unwanted fluid pressure, backflow and water loss.

In similar terms, for a DMA peripheral, we have to define source and destination of data and the size of data. The source of data in this case is ADC12's 0th memory location and the destination is the word-sized global variable we declared in the beginning of the code. We have to let the compiler know the physical addresses, i.e. pointer of the source and the destination.

MSP430's DMA can support the following transfer modes:

DMADT	Transfer Mode	Description
000	Single transfer	Each transfer requires a trigger. DMAEN is automatically cleared when DMAxSZ transfers have been made.
001	Block transfer	A complete block is transferred with one trigger. DMAEN is automatically cleared at the end of the block transfer.
010, 011	Burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN is automatically cleared at the end of the burst-block transfer.
100	Repeated single transfer	Each transfer requires a trigger. DMAEN remains enabled.
101	Repeated block transfer	A complete block is transferred with one trigger. DMAEN remains enabled.
110, 111	Repeated burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN remains enabled.

We will be using 0th DMA channel in repeated single transfer mode since the ADC will be converting data repeatedly. The transfers will be in words and why it is so has already been explained.

Since we are using repeated single transfer mode, a trigger source is needed that will initiate the transfer. In this code, the DMA trigger source is numbered 24.

TRIGGER	CHANNEL		
	0	1	2
0	DMAREQ	DMAREQ	DMAREQ
1	TA0CCR0 CCIFG	TA0CCR0 CCIFG	TA0CCR0 CCIFG
2	TA0CCR2 CCIFG	TA0CCR2 CCIFG	TA0CCR2 CCIFG
3	TA1CCR0 CCIFG	TA1CCR0 CCIFG	TA1CCR0 CCIFG
4	TA1CCR2 CCIFG	TA1CCR2 CCIFG	TA1CCR2 CCIFG
5	TA2CCR0 CCIFG	TA2CCR0 CCIFG	TA2CCR0 CCIFG
6	TA2CCR2 CCIFG	TA2CCR2 CCIFG	TA2CCR2 CCIFG
7	TB0CCR0 CCIFG	TB0CCR0 CCIFG	TB0CCR0 CCIFG
8	TB0CCR2 CCIFG	TB0CCR2 CCIFG	TB0CCR2 CCIFG
9	Reserved	Reserved	Reserved
10	Reserved	Reserved	Reserved
11	Reserved	Reserved	Reserved
12	Reserved	Reserved	Reserved
13	Reserved	Reserved	Reserved
14	Reserved	Reserved	Reserved
15	Reserved	Reserved	Reserved
16	UCA0RXIFG	UCA0RXIFG	UCA0RXIFG
17	UCA0TXIFG	UCA0TXIFG	UCA0TXIFG
18	UCB0RXIFG	UCB0RXIFG	UCB0RXIFG
19	UCB0TXIFG	UCB0TXIFG	UCB0TXIFG
20	UCA1RXIFG	UCA1RXIFG	UCA1RXIFG
21	UCA1TXIFG	UCA1TXIFG	UCA1TXIFG
22	UCB1RXIFG	UCB1RXIFG	UCB1RXIFG
23	UCB1TXIFG	UCB1TXIFG	UCB1TXIFG
24	ADC12IFGx	ADC12IFGx	ADC12IFGx
25	Reserved	Reserved	Reserved
26	Reserved	Reserved	Reserved
27	USB FNRXD	USB FNRXD	USB FNRXD
28	USB ready	USB ready	USB ready
29	MPY ready	MPY ready	MPY ready
30	DMA2IFG	DMA0IFG	DMA1IFG
31	DMAE0	DMAE0	DMAE0

The 24th DMA trigger source is selected because we want the transfer to occur when an AD conversion is ready to be read.

Though DMA interrupt has been used in the code, it does nothing rather than toggling P1.0 LED. This is merely an indicator of data transfer completion.

```
#pragma vector = DMA_VECTOR
__interrupt void DMA_ISR (void)
{
    switch (__even_in_range(DMAIV, 16))
    {
        case 0: break;   //None
        case 2:          //DMA0IFG = DMA Channel 0
        {
            GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);

            break;
        }
        case 4: break;   //DMA1IFG = DMA Channel 1
        case 6: break;   //DMA2IFG = DMA Channel 2
        case 8: break;   //DMA3IFG = DMA Channel 3
        case 10: break;  //DMA4IFG = DMA Channel 4
        case 12: break;  //DMA5IFG = DMA Channel 5
        case 14: break;  //DMA6IFG = DMA Channel 6
        case 16: break;  //DMA7IFG = DMA Channel 7
        default: break;
    }
}
```

In the main loop, the destination memory location is read. Both voltage and ADC count are displayed on an LCD. P4.7 LED is toggled to indicated that LCD data update and memory read have been done.

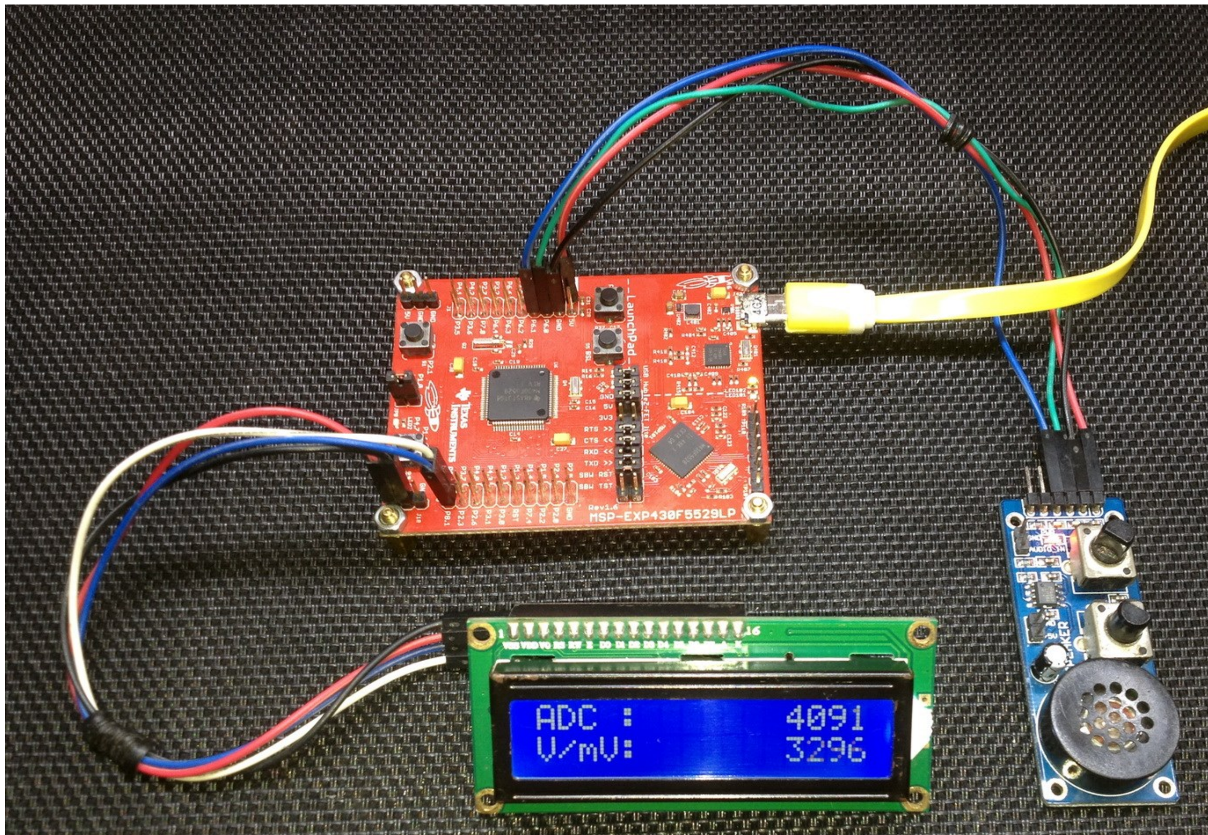
```
volt = ((memory_location * 3300.0) / 4095.0);

print_I(11, 0, memory_location);
print_I(11, 1, volt);

GPIO_toggleOutputOnPin(GPIO_PORT_P4, GPIO_PIN7);
delay_ms(200);
```

Note that in the code, we didn't use any ADC12 function like the one shown below for reading the ADC.

```
ADC12_A_getResults(ADC12_A_BASE, ADC12_A_MEMORY_0);
```

We read the ADC indirectly using the DMA peripheral.

Demo



Video demo: https://youtu.be/ardoI3NE3H8

Analogue Comparator - Comp_B+ Module

Although MSP430F5529 sports a good 12-bit ADC, it also packs a sophisticated analogue comparator called ***COMP_B+*** that can be employed in a number of ways. As with any other embedded analogue comparator, it can be feed with external inputs as well as internal reference voltage generator – REF module. Additionally, this comparator has certain advanced features like software selectable RC filter, voltage hysteresis generator and timer input capture interlink. Positive and negative inputs can be shorted internally to remove any stray charge accumulation. Lastly, another attractive feature of COMP_B+ is its ultra-low power consumption.

Analogue comparators can be used in many cases. Some typical uses include

- voltage level monitor,
- measurement of capacitance,
- measurement of inductance,
- measurement of resistance,
- RC oscillators,
- noise remover,
- waveform converter,
- RC-based ADC, etc.

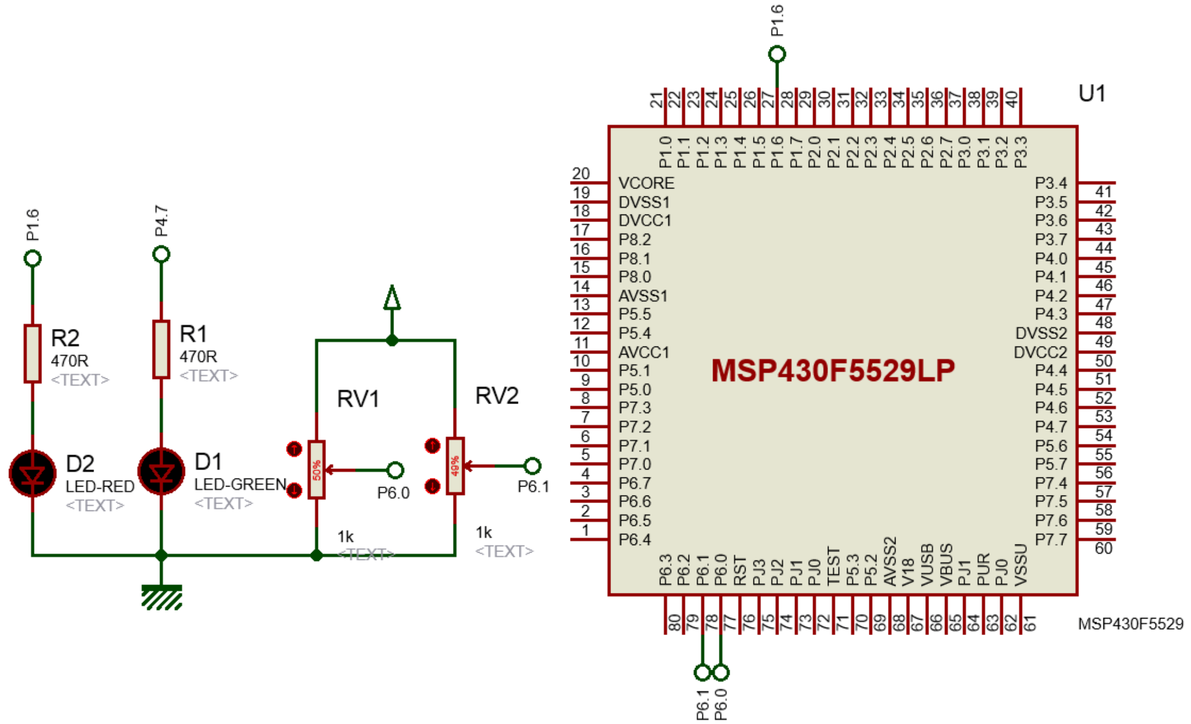
Code Example

```
#include "driverlib.h"
#include "delay.h"


void clock_init(void);
void GPIO_init(void);
void Comp_B_Plus(void);


#pragma vector = COMP_B_VECTOR
__interrupt void Comp_B_ISR(void)
{
    if(Comp_B_getInterruptStatus(COMP_B_BASE,
                                 COMP_B_OUTPUT_FLAG))
    {
        GPIO_toggleOutputOnPin(GPIO_PORT_P4,
                               GPIO_PIN7);

        Comp_B_clearInterrupt(COMP_B_BASE,
                              COMP_B_OUTPUT_FLAG);
    }
}


void main(void)
{
    WDT_A_hold(WDT_A_BASE);

    clock_init();
    GPIO_init();
    Comp_B_Plus();
```

```
    while(true)
    {
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_MCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P6,
                                               (GPIO_PIN0 + GPIO_PIN1));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P1,
                                                GPIO_PIN6);
}


void Comp_B_Plus(void)
{
    Comp_B_initParam Comp_B_Param = {0};

    Comp_B_Param.invertedOutputPolarity = COMP_B_NORMALOUTPUTPOLARITY;
    Comp_B_Param.positiveTerminalInput = COMP_B_INPUT0;
    Comp_B_Param.negativeTerminalInput = COMP_B_INPUT1;
    Comp_B_Param.outputFilterEnableAndDelayLevel = COMP_B_FILTEROUTPUT_DLYLVL3;
    Comp_B_Param.powerModeSelect = COMP_B_POWERMODE_NORMALMODE;

    Comp_B_init(COMP_B_BASE,
                &Comp_B_Param);

    Comp_B_enableInterrupt(COMP_B_BASE,
```

```
                             COMP_B_OUTPUT_INT);

    Comp_B_setInterruptEdgeDirection(COMP_B_BASE,
                                        COMP_B_RISINGEDGE);

    Comp_B_enable(COMP_B_BASE);

    __enable_interrupt();
}
```

Hardware Setup



Explanation

Here two voltage levels are compared and the result of comparison is visually displayed with a LED.

As with other cases, comparator pins are secondary functions of some GPIO pins and are needed to be declared before using COMP_B+.

```
GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P6, (GPIO_PIN0 + GPIO_PIN1));

GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P1, GPIO_PIN6);
```

Now we are ready to setup the comparator module. Comparator parameters state output polarity, inputs to comparator inputs, RC filter level and power consumption mode.

```c
void Comp_B_Plus(void)
{
  Comp_B_initParam Comp_B_Param = {0};

  Comp_B_Param.invertedOutputPolarity = COMP_B_NORMALOUTPUTPOLARITY;
  Comp_B_Param.positiveTerminalInput = COMP_B_INPUT0;
  Comp_B_Param.negativeTerminalInput = COMP_B_INPUT1;
  Comp_B_Param.outputFilterEnableAndDelayLevel = COMP_B_FILTEROUTPUT_DLYLVL3;
  Comp_B_Param.powerModeSelect = COMP_B_POWERMODE_NORMALMODE;

  Comp_B_init(COMP_B_BASE, &Comp_B_Param);

  Comp_B_enableInterrupt(COMP_B_BASE, COMP_B_OUTPUT_INT);

  Comp_B_setInterruptEdgeDirection(COMP_B_BASE, COMP_B_RISINGEDGE);

  Comp_B_enable(COMP_B_BASE);

  __enable_interrupt();
}
```

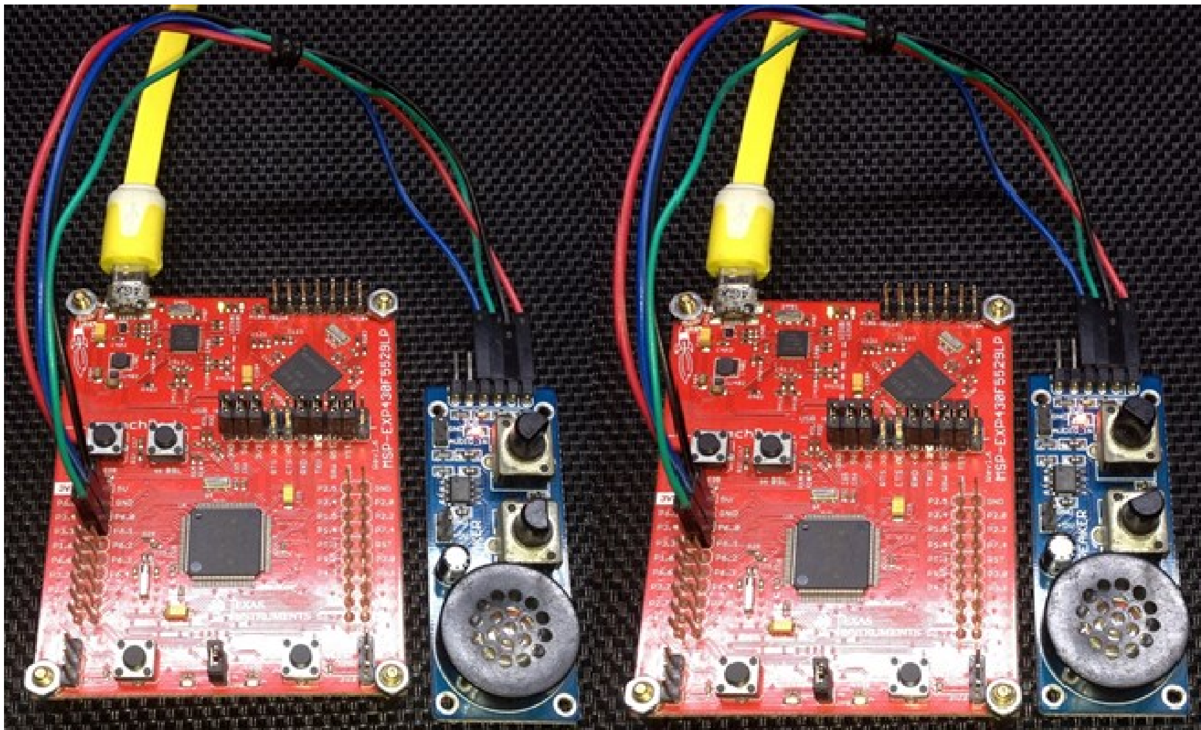
It is my advice to use comparator interrupt because a comparator event occurs to signify a major event. This is why rising edge interrupt is used here. This type of interrupt will occur only when positive input is greater than negative input.

There is no code inside the main loop but inside the interrupt subroutine P4.7 LED is toggled when rising edge comparator interrupt occurs.

```c
#pragma vector = COMP_B_VECTOR
__interrupt void Comp_B_ISR(void)
{
    if(Comp_B_getInterruptStatus(COMP_B_BASE, COMP_B_OUTPUT_FLAG))
    {
        GPIO_toggleOutputOnPin(GPIO_PORT_P4, GPIO_PIN7);

        Comp_B_clearInterrupt(COMP_B_BASE, COMP_B_OUTPUT_FLAG);
    }
}
```

When interrupt occurs, we need to check relevant interrupt flag and clear it after processing the interrupt.

Demo



Demo video: https://youtu.be/G7bc9IN55GM

Communication Hardware Overview

The charm of MSP430F5529 microcontroller is its USB hardware. This is typically not found in most 8-bit and 16-bit general purpose microcontrollers. Apart from USB hardware module, MSP430F5529 has other communication modules that are must-haves in any modern-era microcontroller. Except CAN, MSP430F5529 supports all sorts of communications listed below.

Comm.	*Description*	*I/O*	*Max. Speed*	*Max. Distance*	*Max. Possible Number of Devices in a Bus*
UART	Asynchronous serial point-to-point communication	2	115.2kbps	15m	2 (Point-to-Point)
SPI	Short-range synchronous master-slave serial communication	3/4	4Mbps	0.1m	Virtually unlimited
I2C	Short-range synchronous master-slave serial communication using one data and one clock line	2	1Mbps	0.5m	127
RS-485	Asynchronous differential two wire serial communication with one master	2	115.2kbps	1.2km	Several
CAN	Industrial differential two wire communication with more than one master support	2	1Mbps	5km	Several
LIN	Asynchronous two wire serial communication similar to UART	2	20kbps	40m	Several
IrDA	Wireless serial communication using infrared medium	2	115.2kbps	<1m	2 (Point-to-Point)
USB	Asynchronous full-duplex high-speed serial communication using two data lines/pins	2	480Mbps	<5m	127

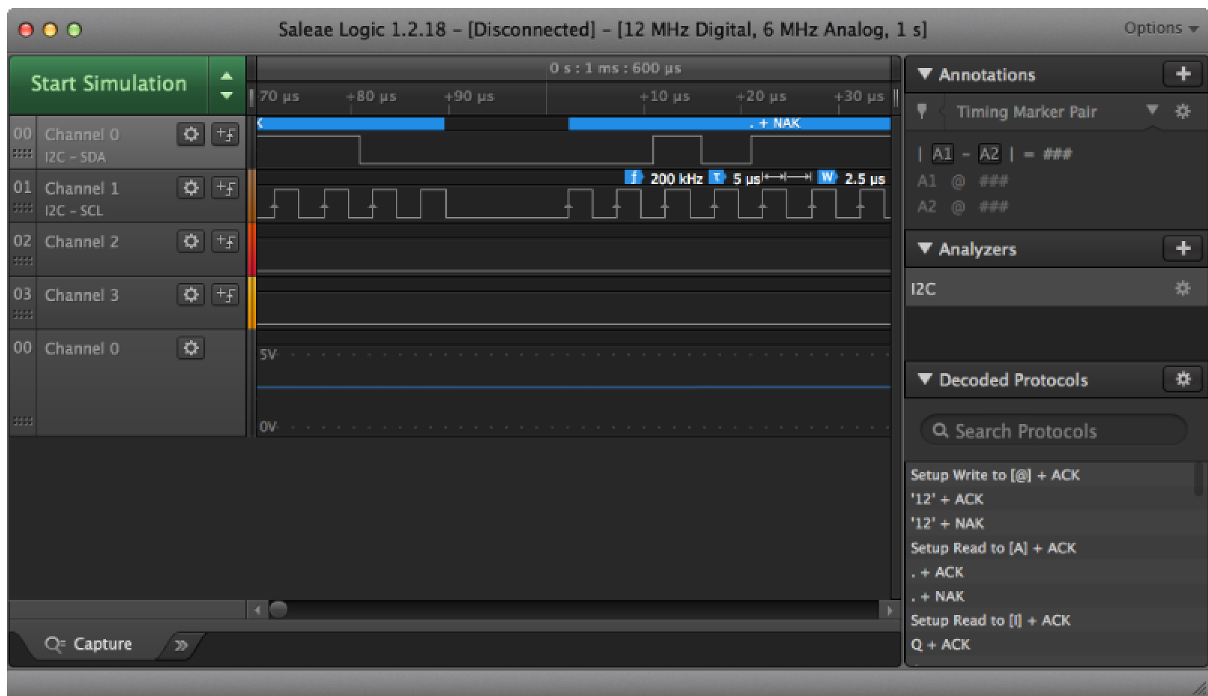
In my previous MSP430 tutorial, I elaborated the difference between ***Universal Serial Interface (USI)*** and ***Universal Serial Communication Interface (USCI)*** modules. In MSP430F5529 micro, there is no USI module as USCI is more robust and advanced than USI. There are 2 USCI_Ax - USCI_A0 and USCI_A1 and 2 USCI_Bx - USCI_B0 and USCI_B1 modules in MSP430F5529. Thus, the total USCI module count in MSP430F5529 is four.

USCI_As are better-suited for asynchronous communications like SPI, UART, LIN and IrDA communications while USCI_Bs are optimized for synchronous communications like for I2C and SPI communications.

With USCI we can expand communication and connectivity. We can then use CAN controllers like MCP2515, ethernet controllers like W5100, WiFi modules like ESP8266 and many others that are not typically available in any microcontroller.

Apart from these hardware interfaces, we can emulate software-based communications like software I2C, software SPI and software UART. One-wire or other non-standard format of communications like that of WS2812 neopixel LEDs, DHT11 relative-humidity & temperature sensor, etc can be created

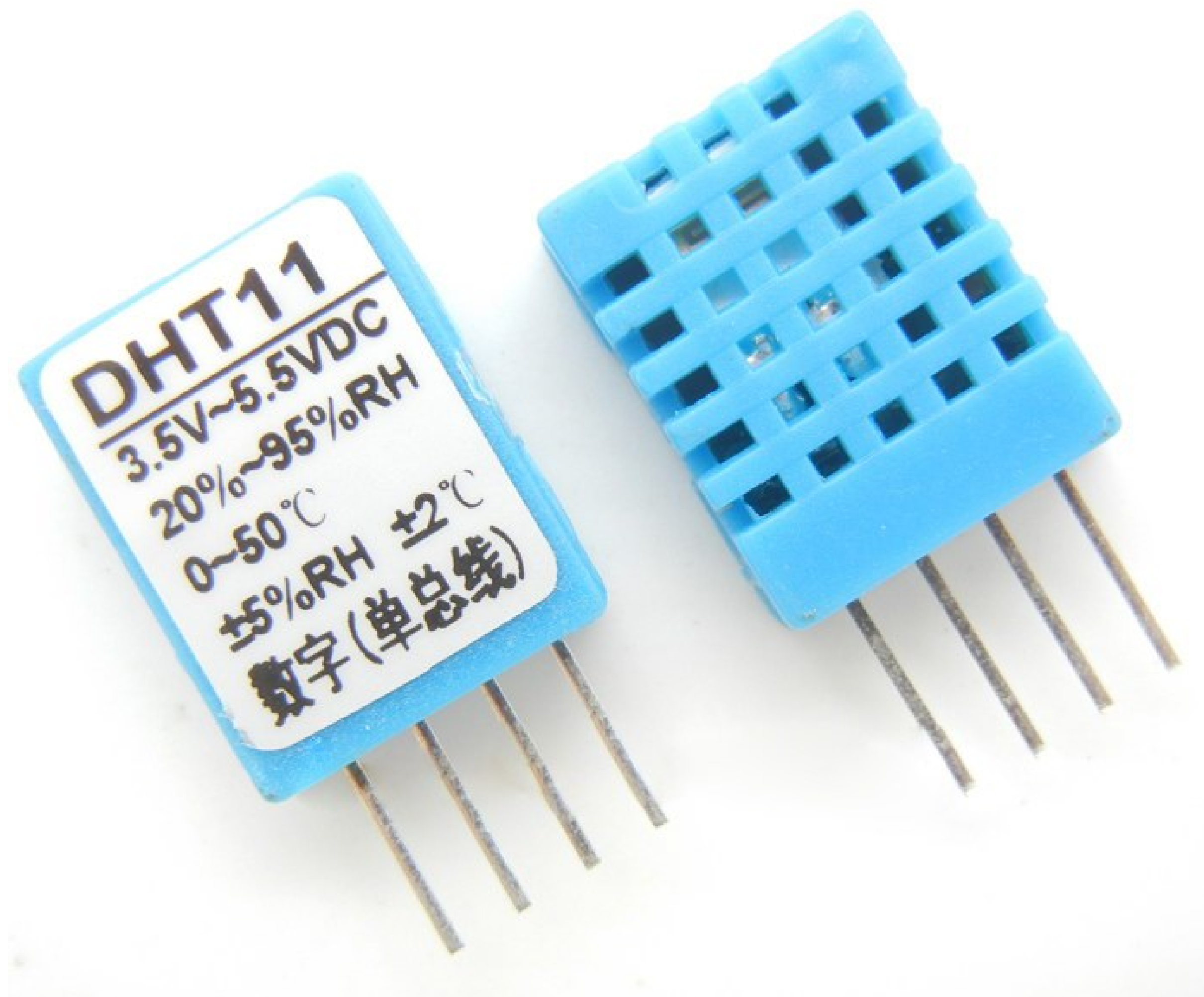
easily using software delays, timers, GPIOs or other hardware by deciphering communication protocols and replicating them.



It is my recommendation that we must invest in good signal/logic analysers and oscilloscopes to fully understand and appreciate the workings of different communication protocols. Timing diagrams and protocol rules are other areas that we must study deeply.

One-Wire Communication with DHT11

DHT11 relative humidity & temperature sensor can be interfaced with a microcontroller via one wire. However, one-wire communication is not standard format of communication like UART, I2C and SPI. Every device has its own protocol that differs a lot with others. However, the basic trick in one-wire protocol is time-slotting mechanism. Ones and zeros are represented by pulses of different pulse widths or duty cycle.



DHT11 can be used to measure temperatures nominally from 0°C to 50°C and relative humidity from 20% to 95%. Though it is not very precise, it can be used for general uses.

A big advantage of this sensor is its ability to operate at 3.3V although it is recommended to apply supply voltages between 3.5V – 5.5V.

Code Example

DHT11.h

```
#include "driverlib.h"
#include "delay.h"


#define HIGH                        1
#define LOW                         0

#define DHT11_PORT                  GPIO_PORT_P3

#define DHT11_pin                   GPIO_PIN7

#define DHT11_DIR_OUT()             GPIO_setAsOutputPin(DHT11_PORT, DHT11_pin)
#define DHT11_DIR_IN()              GPIO_setAsInputPin(DHT11_PORT, DHT11_pin)

#define DHT11_pin_HIGH()            GPIO_setOutputHighOnPin(DHT11_PORT, DHT11_pin)
#define DHT11_pin_LOW()             GPIO_setOutputLowOnPin(DHT11_PORT, DHT11_pin)

#define DHT11_pin_IN()              GPIO_getInputPinValue(DHT11_PORT, DHT11_pin)


void DHT11_init(void);
unsigned char get_byte(void);
unsigned char get_data(void);
```

DHT11.c

```
#include "DHT11.h"


unsigned char values[0x05];


void DHT11_init(void)
{
   DHT11_DIR_IN();
   delay_ms(1000);
}


unsigned char get_byte(void)
{
   unsigned char s = 0;
   unsigned char value = 0;

   DHT11_DIR_IN();

   for(s = 0; s < 8; s++)
   {
      value <<= 1;
      while(DHT11_pin_IN() == LOW);
      delay_us(30);

      if(DHT11_pin_IN() == HIGH)
      {
          value |= 1;
      }

      while(DHT11_pin_IN() == HIGH);
```

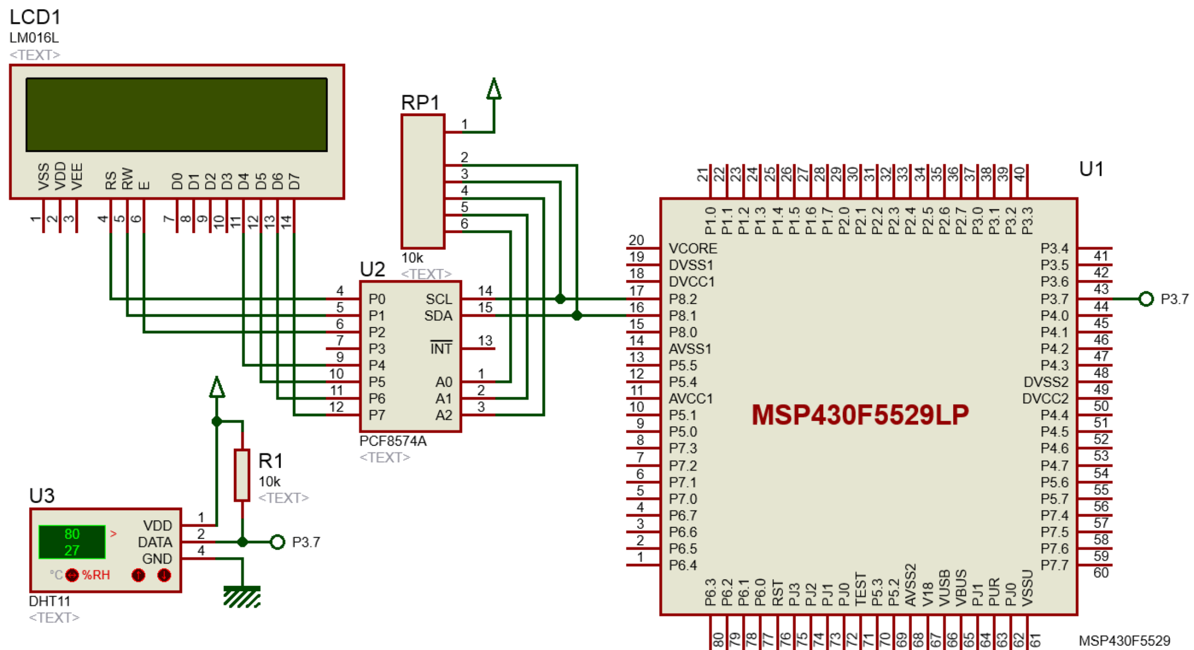
```
    }
    return value;
}


unsigned char get_data(void)
{
    short chk = 0;
    unsigned char s = 0;
    unsigned char check_sum = 0;

    DHT11_DIR_OUT();

    DHT11_pin_HIGH();
    DHT11_pin_LOW();
    delay_ms(18);
    DHT11_pin_HIGH();
    delay_us(26);

    DHT11_DIR_IN();

    chk = DHT11_pin_IN();

    if(chk)
    {
        return 1;
    }
    delay_us(80);

    chk = DHT11_pin_IN();

    if(!chk)
    {
        return 2;
    }
    delay_us(80);

    for(s = 0; s <= 4; s += 1)
    {
        values[s] = get_byte();
    }

    DHT11_DIR_OUT();

    DHT11_pin_HIGH();

    DHT11_DIR_IN();

    for(s = 0; s < 4; s += 1)
    {
        check_sum += values[s];
    }

    if(check_sum != values[4])
    {
        return 3;
    }
    else
    {
        return 0;
    }
}
```

main.c

```c
#include "driverlib.h"
#include "delay.h"
#include "DHT11.h"
#include "lcd.h"
#include "lcd_print.h"


extern unsigned char values[0x05];


void clock_init(void);


void main(void)
{
    unsigned char state = 0;

    WDT_A_hold(WDT_A_BASE);

    clock_init();

    LCD_init();
    load_custom_symbol();

    DHT11_init();

    while(1)
    {
        state = get_data();

        switch(state)
        {
            case 1:
            {
            }
            case 2:
            {
                 LCD_clear_home();
                 LCD_putstr("No Sensor Found!");
                 break;
            }
            case 3:
            {
                 LCD_clear_home();
                 LCD_putstr("Checksum Error!");
                 break;
            }
            default:
            {
                 LCD_goto(0, 0);
                 LCD_putstr("R.H/ %:       ");

                 print_C(13, 0, values[0]);

                 LCD_goto(0, 1);
                 LCD_putstr("Tmp/");
                 print_symbol(4, 1, 0);
                 LCD_goto(5, 1);
                 LCD_putstr("C:");

                 if((values[2] & 0x80) == 1)
                 {
                     LCD_goto(13, 1);
```

```
                    LCD_putstr("-");
                }
                else
                {
                    LCD_goto(13, 1);
                    LCD_putstr(" ");
                }

                print_C(13, 1, values[2]);
                break;
            }
        }

        delay_ms(1000);
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_MCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_REFOCLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}
```

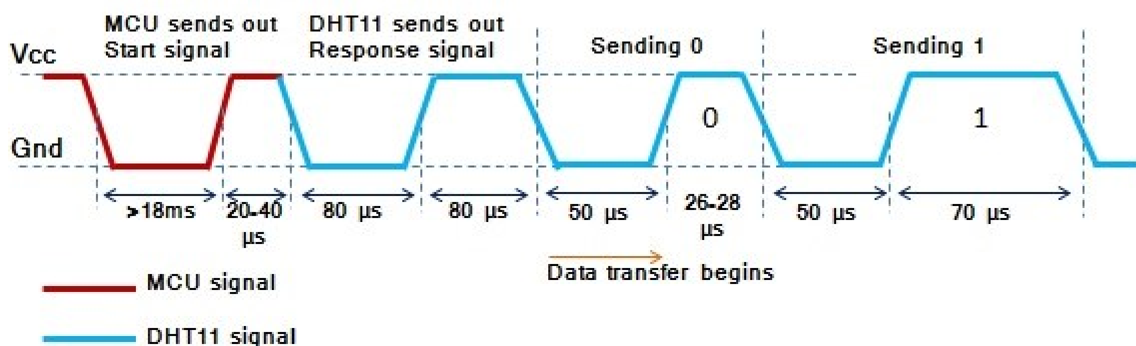
Hardware Setup



Explanation

One-wire sensors and devices use time-slotting mechanism to communicate with host micro. Time-slotting and communication methodology differ from device-to-device and have no standards like I2C, SPI, UART, etc. Despite this fact, communicating with one-wire devices is not very difficult. All we have to do is to generate timed pulse or read pulse transitions.

Shown below is the timing diagram for DHT11 relative humidity & temperature sensor. We have to keep the one-wire data line floating unless there is any communication. A floating pin is basically an externally pulled-up input pin unless used otherwise. The pull-up is achieved using an external resistor connected between the host micro's pin and power supply's positive line.



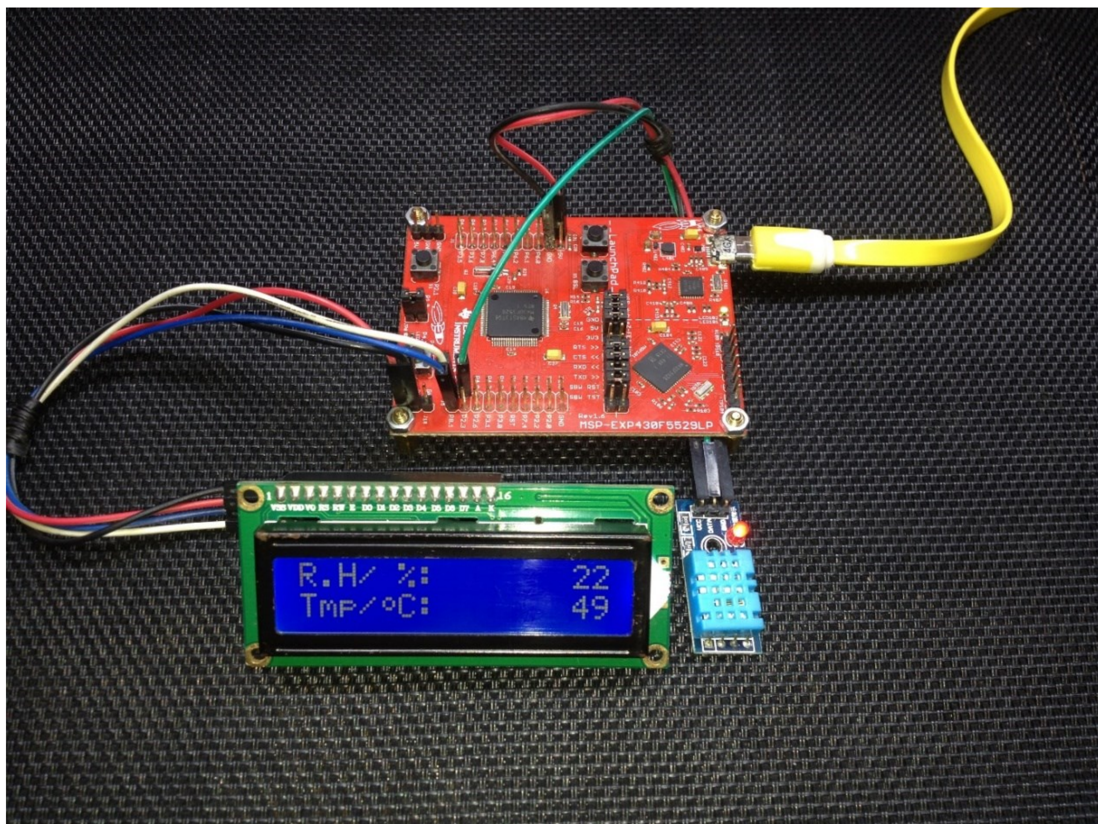
A host micro has to pull-down the floating communication low for about 18ms and then keep it high for about 30µs to command DHT11 to send out its temperature and humidity readings. This is done as follows in the code below.

```
DHT11_pin_HIGH();
DHT11_pin_LOW();
delay_ms(18);
DHT11_pin_HIGH();
delay_us(26);
```

It is very straight forward. DHT11 upon receiving these logic-level transitions starts to serially send out its data readings one-by-one. During reads, we have to check timings in order to determine if the sent pulse is denoting a one or zero. If an incoming pulse is high for more than 30µs, it is treated as a one or else it is treated otherwise. Again, it is pretty simple.

```
value <<= 1;
while(DHT11_pin_IN() == LOW);
delay_us(30);

if(DHT11_pin_IN() == HIGH)
{
   value |= 1;
}

while(DHT11_pin_IN() == HIGH);
```

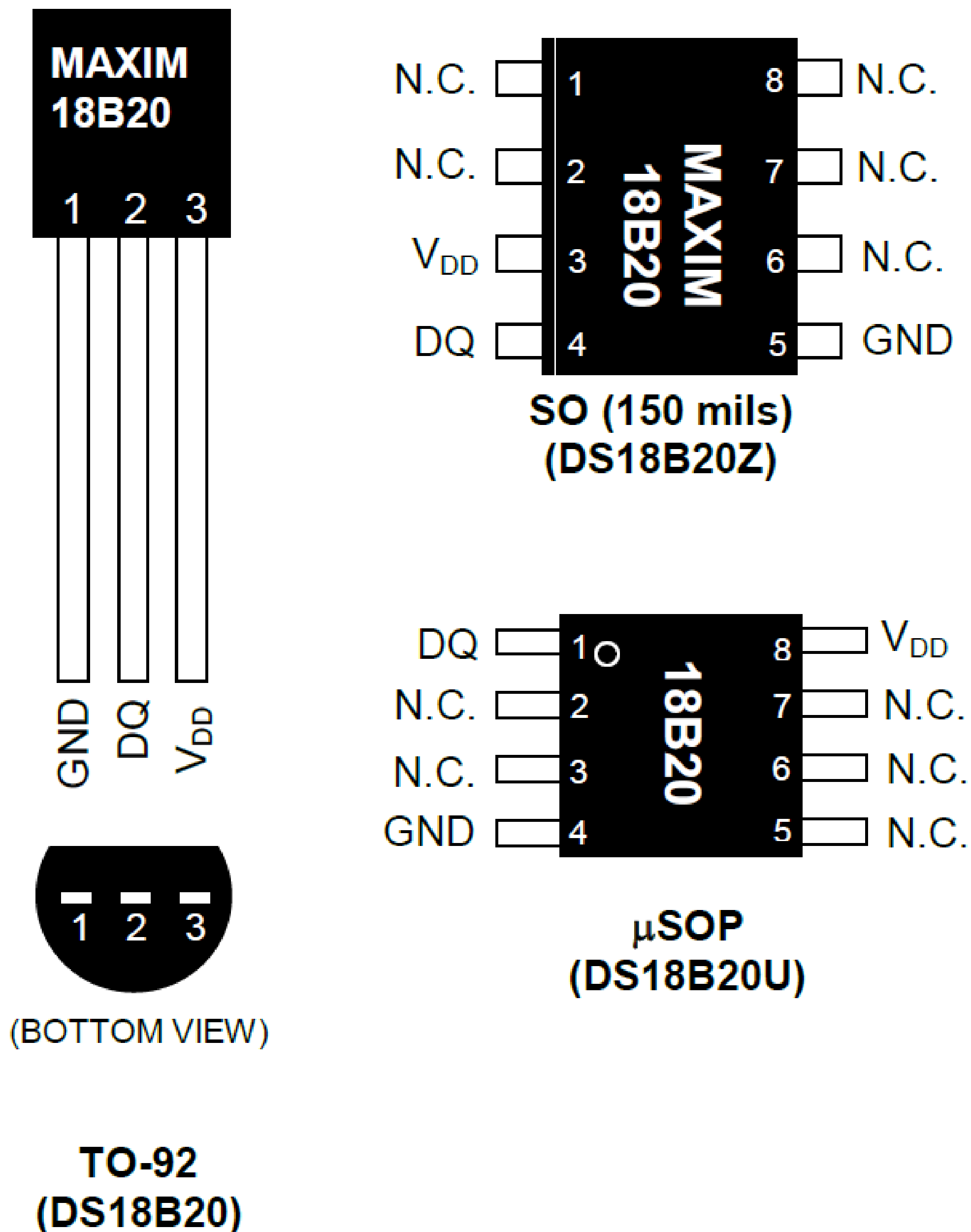
Demo



Demo video: https://youtu.be/p8dCG94tzxo

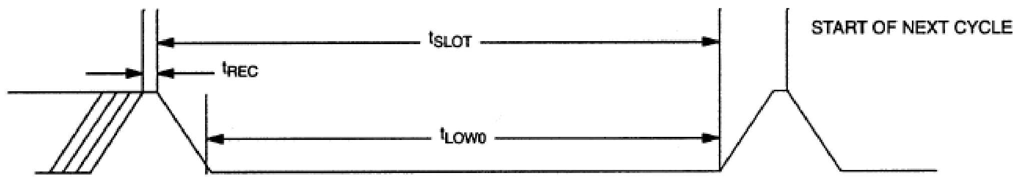
One-Wire Communication with DS18B20

Dallas DS18B20 is another one-wire device but as said earlier, it is different from DHT11 or other one-wire devices. DS18B20 is a high-resolution digital temperature sensor. This sensor also uses time-slotting mechanism to send and receive data from a host micro.

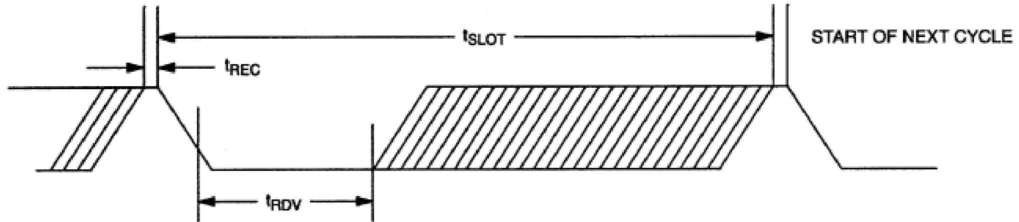


One advantage of time-slotting mechanism is the fact that no special hardware and therefore no dedicated pin of a host microcontroller is needed. Any physical pin can be used and only software delays along with polling methodology are what required. It is also possible to use internal timers instead of wasteful software delays.

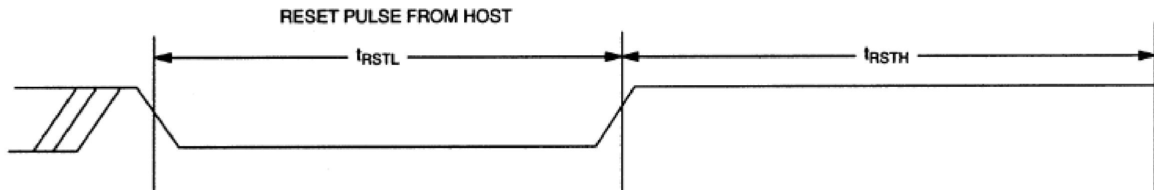
1–WIRE WRITE ZERO TIME SLOT



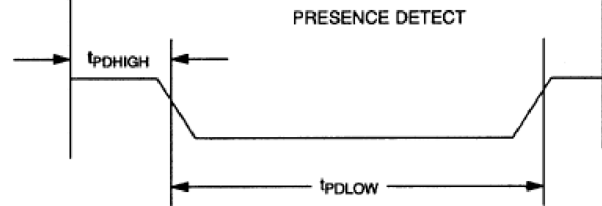
1–WIRE READ ZERO TIME SLOT



1–WIRE RESET PULSE



1–WIRE PRESENCE DETECT



Time-slotting technique is also used in infrared remote controllers.

Code Example

one_wire.h

```
#include "driverlib.h"
#include "delay.h"


#define DS18B20_PORT        GPIO_PORT_P3

#define DS18B20_PIN         GPIO_PIN7

#define DS18B20_OUTPUT()    GPIO_setAsOutputPin(DS18B20_PORT, DS18B20_PIN)
#define DS18B20_INPUT()     GPIO_setAsInputPin(DS18B20_PORT, DS18B20_PIN)

#define DS18B20_IN()        GPIO_getInputPinValue(DS18B20_PORT, DS18B20_PIN)

#define DS18B20_OUT_HIGH()  GPIO_setOutputHighOnPin(DS18B20_PORT, DS18B20_PIN)
#define DS18B20_OUT_LOW()   GPIO_setOutputLowOnPin(DS18B20_PORT, DS18B20_PIN)

#define TRUE                1
#define FALSE               0


unsigned char onewire_reset(void) ;
void onewire_write_bit(unsigned char bit_value);
unsigned char onewire_read_bit(void);
void onewire_write(unsigned char value);
unsigned char onewire_read(void);
```

one_wire.c

```
#include "one_wire.h"


unsigned char onewire_reset(void)
{
     unsigned char res = FALSE;

     DS18B20_OUTPUT();
     DS18B20_OUT_LOW();
     delay_us(480);
     DS18B20_OUT_HIGH();
     delay_us(60);

     DS18B20_INPUT();
     res = DS18B20_IN();
     delay_us(480);

     return res;
}


void onewire_write_bit(unsigned char bit_value)
{
    DS18B20_OUTPUT();
    DS18B20_OUT_LOW();

    if(bit_value)
    {
        delay_us(104);
        DS18B20_OUT_HIGH();
```

```
    }
}


unsigned char onewire_read_bit(void)
{
    DS18B20_OUTPUT();
    DS18B20_OUT_LOW();
    DS18B20_OUT_HIGH();
    delay_us(15);
    DS18B20_INPUT();

    return(DS18B20_IN());
}


void onewire_write(unsigned char value)
{
     unsigned char s = 0;

     DS18B20_OUTPUT();

     while(s < 8)
     {
          if((value & (1 << s)))
          {
              DS18B20_OUT_LOW();
              _delay_cycles(1);
              DS18B20_OUT_HIGH();
              delay_us(60);
          }

          else
          {
              DS18B20_OUT_LOW();
              delay_us(60);
              DS18B20_OUT_HIGH();
              _delay_cycles(1);
          }

          s++;
     }
}


unsigned char onewire_read(void)
{
     unsigned char s = 0x00;
     unsigned char value = 0x00;

     while(s < 8)
     {
          DS18B20_OUTPUT();

          DS18B20_OUT_LOW();
          _delay_cycles(1);
          DS18B20_OUT_HIGH();

          DS18B20_INPUT();
          if(DS18B20_IN())
          {
              value |=  (1 << s);
          }

          delay_us(60);
```

```
            s++;
    }

    return value;
}
```

DS18B20.h

```
#include "one_wire.h"


#define convert_T                       0x44
#define read_scratchpad                 0xBE
#define write_scratchpad                0x4E
#define copy_scratchpad                 0x48
#define recall_E2                       0xB8
#define read_power_supply               0xB4
#define skip_ROM                        0xCC

#define resolution                      12


void DS18B20_init(void);
float DS18B20_get_temperature(void);
```

DS18B20.c

```
#include "DS18B20.h"


void DS18B20_init(void)
{
    onewire_reset();
    delay_ms(100);
}


float DS18B20_get_temperature(void)
{
    unsigned char msb = 0x00;
    unsigned char lsb = 0x00;
    register float temp = 0.0;

    onewire_reset();
    onewire_write(skip_ROM);
    onewire_write(convert_T);

    switch(resolution)
    {
        case 12:
        {
            delay_ms(750);
            break;
        }
        case 11:
        {
            delay_ms(375);
            break;
        }
        case 10:
        {
            delay_ms(188);
```

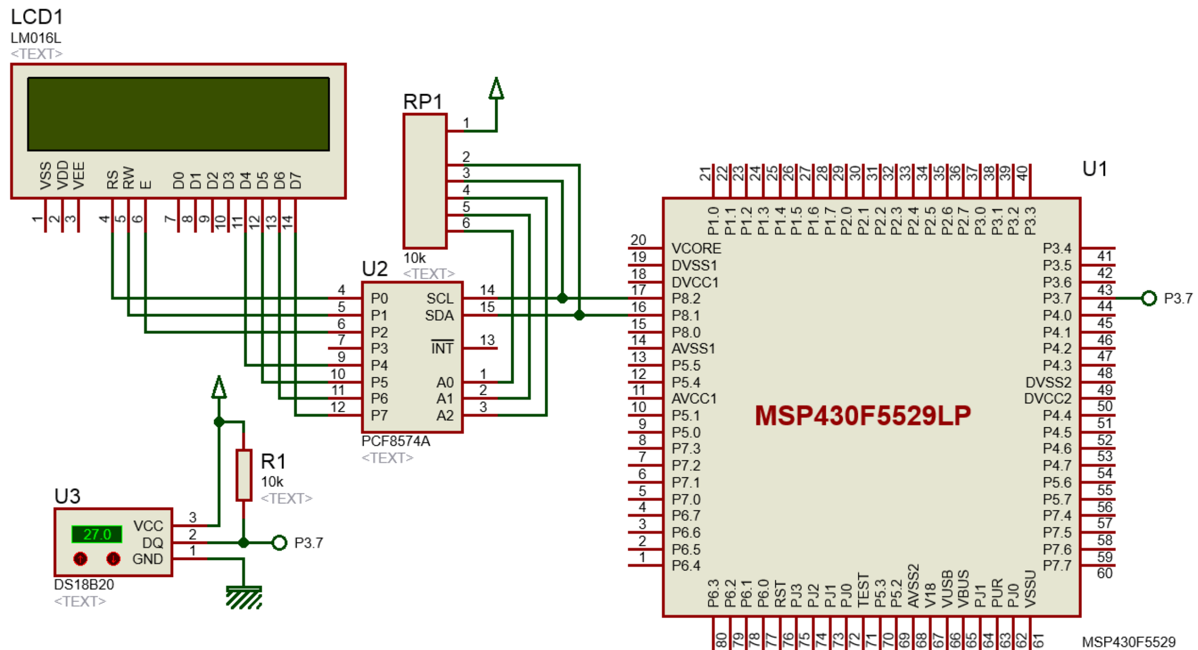
```
            break;
        }
        case 9:
        {
            delay_ms(94);
            break;
        }
    }

    onewire_reset();

    onewire_write(skip_ROM);
    onewire_write(read_scratchpad);

    lsb = onewire_read();
    msb = onewire_read();

    temp = msb;
    temp *= 256.0;
    temp += lsb;


    switch(resolution)
    {
        case 12:
        {
            temp *= 0.0625;
            break;
        }
        case 11:
        {
            temp *= 0.125;
            break;
        }
        case 10:
        {
            temp *= 0.25;
            break;
        }
        case 9:
        {
            temp *= 0.5;
            break;
        }
    }

    delay_ms(40);

    return (temp);
}
```

main.c

```
#include "driverlib.h"
#include "delay.h"
#include "one_wire.h"
#include "DS18B20.h"
#include "lcd.h"
#include "lcd_print.h"


void clock_init(void);
```

```c
void main(void)
{
    float t = 0.0;

    WDT_A_hold(WDT_A_BASE);

    clock_init();

    DS18B20_init();

    LCD_init();
    load_custom_symbol();

    LCD_goto(1, 0);
    LCD_putstr("MSP430 DS18B20");

    LCD_goto(0, 1);
    LCD_putstr("T/ C");
    print_symbol(2, 1, 0);

    while(1)
    {
        t = DS18B20_get_temperature();
        print_F(9, 1, t, 3);
        delay_ms(1000);
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_0,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_MCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_REFOCLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}
```

Hardware Setup



Explanation

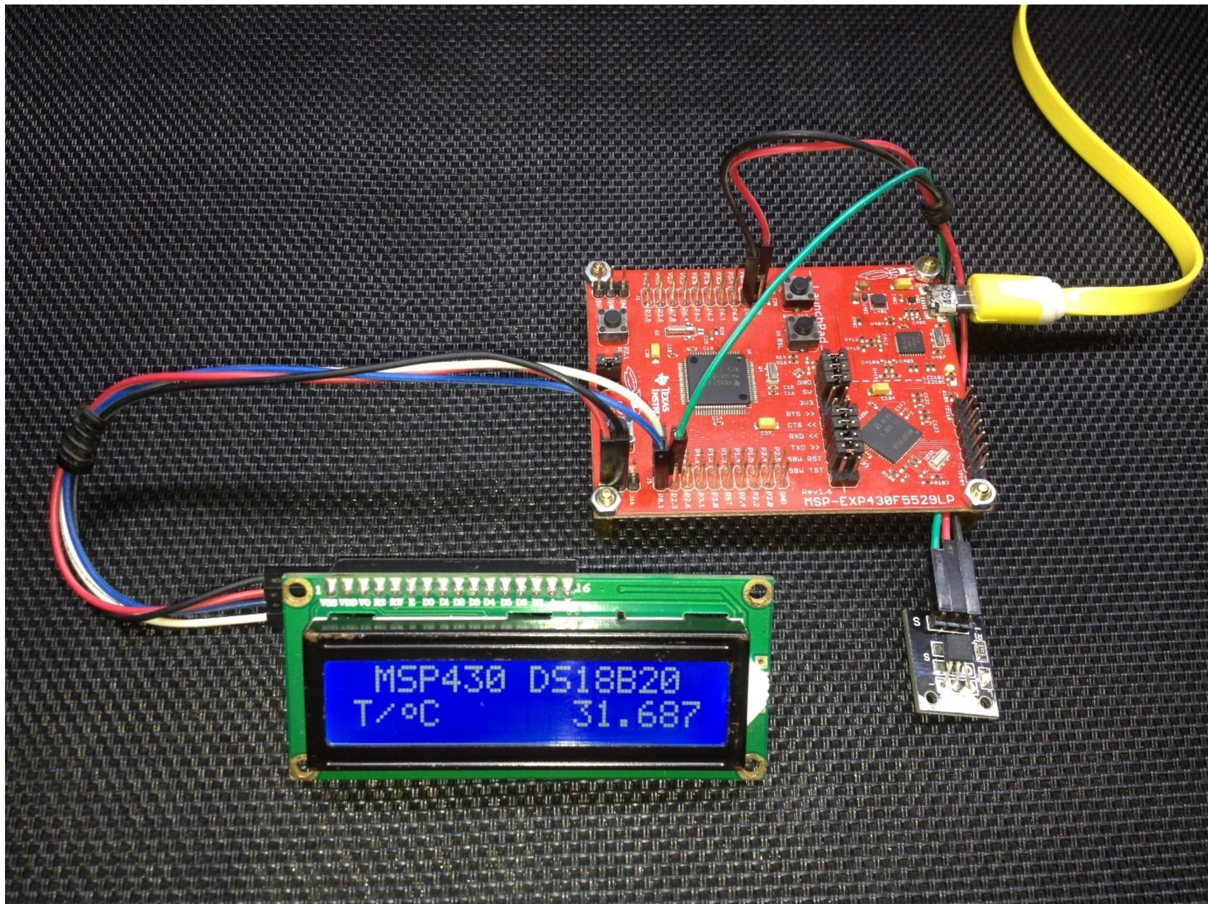
One wire communication is detailed in these application notes from Maxim:

https://www.maximintegrated.com/en/app-notes/index.mvp/id/126

https://www.maximintegrated.com/en/app-notes/index.mvp/id/162

These notes are all that are needed for implementing the one wire communication interface for DS18B20. Please go through these notes for details. The codes are self-explanatory and are implemented from the code examples in these app notes.

Demo



Demo video: https://youtu.be/dih8Zm6u9j0

USCI – SPI – I2C Example 1

In this first USCI example, we will see how we can use USCI modules to read a MMA7455L accelerometer in I2C mode and project its readings on a SPI-based ST7735 TFT display. This is an over simplified example and a rudimentary one although it may appear that lot of coding has been done here. Here, the accelerometer's axes readings are plotted on the TFT display.

Code Example

MMA7455L.h

```
#include "driverlib.h"
#include "delay.h"


#define I2C_port                            GPIO_PORT_P4

#define I2C_SDA_pin                         GPIO_PIN1
#define I2C_SCL_pin                         GPIO_PIN2

#define MMA7455L_address                    0x1D

#define MMA7455L_X_out_L                    0x00
#define MMA7455L_X_out_H                    0x01
#define MMA7455L_Y_out_L                    0x02
#define MMA7455L_Y_out_H                    0x03
#define MMA7455L_Z_out_L                    0x04
#define MMA7455L_Z_out_H                    0x05
#define MMA7455L_X_out                      0x06
#define MMA7455L_Y_out                      0x07
#define MMA7455L_Z_out                      0x08
#define MMA7455L_STATUS                     0x09
#define MMA7455L_DETSRC                     0x0A
#define MMA7455L_TOUT                       0x0B
#define MMA7455L_RSV1                       0x0C
#define MMA7455L_I2C_Address                0x0D
#define MMA7455L_User_Info                  0x0E
#define MMA7455L_Who_Am_I                   0x0F
#define MMA7455L_X_offset_L                 0x10
#define MMA7455L_X_offset_H                 0x11
#define MMA7455L_Y_offset_L                 0x12
#define MMA7455L_Y_offset_H                 0x13
#define MMA7455L_Z_offset_L                 0x14
#define MMA7455L_Z_offset_H                 0x15
#define MMA7455L_MCTL                       0x16
#define MMA7455L_INTRST                     0x17
#define MMA7455L_CTL1                       0x18
#define MMA7455L_CTL2                       0x19
#define MMA7455L_LDTH                       0x1A
#define MMA7455L_PDTH                       0x1B
#define MMA7455L_PW                         0x1C
#define MMA7455L_LT                         0x1D
#define MMA7455L_TW                         0x1E
#define MMA7455L_RSV2                       0x1F

#define MMA_7455_2G_MODE                    0x05
#define MMA_7455_4G_MODE                    0x09
#define MMA_7455_8G_MODE                    0x01
```

```c
void I2C_DIO_init(void);
void USCI_I2C_init(void);
void MMA7455L_init(void);
void MMA7455L_write_byte(unsigned char address, unsigned char value);
void MMA7455L_write_word(unsigned char address, unsigned int value);
unsigned char MMA7455L_read_byte(unsigned char address);
unsigned int MMA7455L_read_word(unsigned char address);
signed char MMA7455L_read_axis_8(unsigned char axis);
signed int MMA7455L_read_axis_10(unsigned char axis);
```

MMA7455L.c

```c
#include "MMA7455L.h"


void I2C_DIO_init(void)
{
    GPIO_setAsPeripheralModuleFunctionOutputPin(I2C_port, (I2C_SDA_pin | I2C_SCL_pin));
}


void USCI_I2C_init(void)
{
    USCI_B_I2C_initMasterParam I2C_param = {0};

    I2C_DIO_init();

    I2C_param.selectClockSource = USCI_B_I2C_CLOCKSOURCE_SMCLK;
    I2C_param.i2cClk = UCS_getSMCLK();
    I2C_param.dataRate = USCI_B_I2C_SET_DATA_RATE_100KBPS;

    USCI_B_I2C_initMaster(USCI_B1_BASE, &I2C_param);

    USCI_B_I2C_enable(USCI_B1_BASE);
}


void MMA7455L_init(void)
{
    USCI_I2C_init();
    MMA7455L_write_byte(MMA7455L_CTL1, 0x80);
    MMA7455L_write_byte(MMA7455L_MCTL, MMA_7455_2G_MODE);
}


void MMA7455L_write_byte(unsigned char address, unsigned char value)
{
    USCI_B_I2C_setSlaveAddress(USCI_B1_BASE, MMA7455L_address);
    USCI_B_I2C_setMode(USCI_B1_BASE, USCI_B_I2C_TRANSMIT_MODE);

    USCI_B_I2C_masterSendMultiByteStart(USCI_B1_BASE, address);
    while(!USCI_B_I2C_masterIsStartSent(USCI_B1_BASE));
    USCI_B_I2C_masterSendMultiByteFinish(USCI_B1_BASE, value);

    while(USCI_B_I2C_isBusBusy(USCI_B1_BASE));
}


void MMA7455L_write_word(unsigned char address, unsigned int value)
{
    unsigned char lb = 0x00;
    unsigned char hb = 0x00;

    lb = ((unsigned char)value);
```

```c
    value >>= 8;
    hb = ((unsigned char)value);

    MMA7455L_write_byte(address, lb);
    MMA7455L_write_byte((address + 1), hb);
}


unsigned char MMA7455L_read_byte(unsigned char address)
{
    unsigned char value = 0x00;

    USCI_B_I2C_setSlaveAddress(USCI_B1_BASE, MMA7455L_address);
    USCI_B_I2C_setMode(USCI_B1_BASE, USCI_B_I2C_TRANSMIT_MODE);

    USCI_B_I2C_masterSendStart(USCI_B1_BASE);
    USCI_B_I2C_masterSendSingleByte(USCI_B1_BASE, address);

    USCI_B_I2C_setSlaveAddress(USCI_B1_BASE, MMA7455L_address);
    USCI_B_I2C_setMode(USCI_B1_BASE, USCI_B_I2C_RECEIVE_MODE);

    USCI_B_I2C_masterReceiveSingleStart(USCI_B1_BASE);
    value = USCI_B_I2C_masterReceiveSingle(USCI_B1_BASE);

    while(USCI_B_I2C_isBusBusy(USCI_B1_BASE));

    return value;
}


unsigned int MMA7455L_read_word(unsigned char address)
{
    unsigned char lb = 0x00;
    unsigned int hb = 0x0000;

    USCI_B_I2C_setSlaveAddress(USCI_B1_BASE, MMA7455L_address);
    USCI_B_I2C_setMode(USCI_B1_BASE, USCI_B_I2C_TRANSMIT_MODE);

    USCI_B_I2C_masterSendStart(USCI_B1_BASE);
    USCI_B_I2C_masterSendSingleByte(USCI_B1_BASE, address);

    USCI_B_I2C_setSlaveAddress(USCI_B1_BASE, MMA7455L_address);
    USCI_B_I2C_setMode(USCI_B1_BASE, USCI_B_I2C_RECEIVE_MODE);

    USCI_B_I2C_masterReceiveMultiByteStart(USCI_B1_BASE);
    lb = USCI_B_I2C_masterReceiveMultiByteNext(USCI_B1_BASE);
    hb = USCI_B_I2C_masterReceiveMultiByteFinish(USCI_B1_BASE);

    USCI_B_I2C_masterReceiveMultiByteStop(USCI_B1_BASE);

    while(USCI_B_I2C_isBusBusy(USCI_B1_BASE));

    hb <<= 8;
    hb |= lb;

    return hb;
}


signed char MMA7455L_read_axis_8(unsigned char axis)
{
    return ((signed char)MMA7455L_read_byte(axis));
}
```

```
signed int MMA7455L_read_axis_10(unsigned char axis)
{
    return ((signed int)MMA7455L_read_word(axis) & 0x03FF);
}
```

ST7735.h

```
#include "driverlib.h"
#include "delay.h"


#define MOSI_port                               GPIO_PORT_P3
#define MISO_port                               GPIO_PORT_P3
#define CLK_port                                GPIO_PORT_P3
#define RST_port                                GPIO_PORT_P3
#define RS_port                                 GPIO_PORT_P3
#define CS_port                                 GPIO_PORT_P3

#define MOSI_pin                                GPIO_PIN0
#define MISO_pin                                GPIO_PIN1
#define CLK_pin                                 GPIO_PIN2
#define RST_pin                                 GPIO_PIN3
#define RS_pin                                  GPIO_PIN4
#define CS_pin                                  GPIO_PIN5


#define MOSI_pin_high() GPIO_setOutputHighOnPin(MOSI_port, MOSI_pin)
#define MOSI_pin_low()  GPIO_setOutputLowOnPin(MOSI_port, MOSI_pin)

#define get_MISO_pin()  GPIO_getInputPinValue(MISO_port, MISO_pin)

#define CLK_pin_high()  GPIO_setOutputHighOnPin(CLK_port, CLK_pin)
#define CLK_pin_low()   GPIO_setOutputLowOnPin(CLK_port, CLK_pin)

#define RST_pin_high()  GPIO_setOutputHighOnPin(RST_port, RST_pin)
#define RST_pin_low()   GPIO_setOutputLowOnPin(RST_port, RST_pin)

#define RS_pin_high()   GPIO_setOutputHighOnPin(RS_port, RS_pin)
#define RS_pin_low()    GPIO_setOutputLowOnPin(RS_port, RS_pin)

#define CS_pin_high()   GPIO_setOutputHighOnPin(CS_port, CS_pin)
#define CS_pin_low()    GPIO_setOutputLowOnPin(CS_port, CS_pin)


#define ST7735_NOP                              0x00
#define ST7735_SWRESET                          0x01
#define ST7735_RDDID                            0x04
#define ST7735_RDDST                            0x09
#define ST7735_RDDPM                            0x0A
#define ST7735_RDD_MADCTL                       0x0B
#define STT7735_RDD_COLMOD                      0x0C
#define ST7735_RDDIM                            0x0D
#define ST7735_RDDSM                            0x0E

#define ST7735_SLPIN                            0x10
#define ST7735_SLPOUT                           0x11
#define ST7735_PTLON                            0x12
#define ST7735_NORON                            0x13

#define ST7735_INVOFF                           0x20
#define ST7735_INVON                            0x21
#define ST7735_GAMSET                           0x26
#define ST7735_DISPOFF                          0x28
#define ST7735_DISPON                           0x29
```

```c
#define ST7735_CASET                        0x2A
#define ST7735_RASET                        0x2B
#define ST7735_RAMWR                        0x2C
#define ST7735_RAMRD                        0x2E

#define ST7735_PTLAR                        0x30
#define ST7735_TEOFF                        0x34
#define ST7735_TEON                         0x35
#define ST7735_MADCTL                       0x36
#define ST7735_IDMOFF                       0x38
#define ST7735_IDMON                        0x39
#define ST7735_COLMOD                       0x3A

#define ST7735_RDID1                        0xDA
#define ST7735_RDID2                        0xDB
#define ST7735_RDID3                        0xDC
#define ST7735_RDID4                        0xDD

#define ST7735_FRMCTR1                      0xB1
#define ST7735_FRMCTR2                      0xB2
#define ST7735_FRMCTR3                      0xB3
#define ST7735_INVCTR                       0xB4
#define ST7735_DISSET5                      0xB6

#define ST7735_PWCTR1                       0xC0
#define ST7735_PWCTR2                       0xC1
#define ST7735_PWCTR3                       0xC2
#define ST7735_PWCTR4                       0xC3
#define ST7735_PWCTR5                       0xC4
#define ST7735_VMCTR1                       0xC5

#define ST7735_RDID1                        0xDA
#define ST7735_RDID2                        0xDB
#define ST7735_RDID3                        0xDC
#define ST7735_RDID4                        0xDD

#define ST7735_PWCTR6                       0xFC

#define ST7735_GMCTRP1                      0xE0
#define ST7735_GMCTRN1                      0xE1

#define BLACK                               0x0000
#define BLUE                                0x001F
#define RED                                 0xF800
#define GREEN                               0x07E0
#define CYAN                                0x07FF
#define MAGENTA                             0xF81F
#define YELLOW                              0xFFE0
#define WHITE                               0xFFFF

#define MADCTL_MY                           0x80
#define MADCTL_MX                           0x40
#define MADCTL_MV                           0x20
#define MADCTL_ML                           0x10
#define MADCTL_RGB                          0x08
#define MADCTL_MH                           0x04

#define ST7735_TFTWIDTH                     128
#define ST7735_TFTLENGTH                    160

#define CMD                                 0
#define DAT                                 1

#define SQUARE                              0
#define ROUND                               1
```

```
#define NO                                      0
#define YES                                     1


void SPI_DIO_init(void);
void USCI_SPI_init(void);
void ST7735_init(void);
void ST7735_Write(unsigned char value, unsigned char mode);
void ST7735_Reset(void);
void ST7735_Word_Write(unsigned int value);
void ST7735_Data_Write(unsigned char DataH, unsigned char DataL);
void ST7735_Data_Write_4k(unsigned char DataH, unsigned char DataM, unsigned char DataL);
void ST7735_Set_Addr_Window(unsigned char xs, unsigned char ys, unsigned char xe, unsigned char ye);
void ST7735_RAM_Address_Set(void);
void ST7735_Invert_Display(unsigned char i);
unsigned int ST7735_Swap_Colour(unsigned int colour);
unsigned int ST7735_Color565(unsigned char r, unsigned char g, unsigned char b);
void ST7735_Set_Rotation(unsigned char m);
void TFT_fill(unsigned int colour);
void Draw_Pixel(unsigned char x_pos, unsigned char y_pos, unsigned int colour);
void Draw_Line(signed int x1, signed int y1, signed int x2, signed int y2, unsigned int colour);
void Draw_Rectangle(signed int x1, signed int y1, signed int x2, signed int y2, unsigned char fill, unsigned char type, unsigned int colour, unsigned int back_colour);
void Draw_Circle(signed int xc, signed int yc, signed int radius, unsigned char fill, unsigned int colour);
void Draw_Font_Pixel(unsigned char x_pos, unsigned char y_pos, unsigned int colour, unsigned char pixel_size);
void print_char(unsigned char x_pos, unsigned char y_pos, unsigned char font_size, unsigned int colour, unsigned int back_colour, unsigned char ch);
void print_str(unsigned char x_pos, unsigned char y_pos, unsigned char font_size, unsigned int colour, unsigned int back_colour, unsigned char *ch);
void print_C(unsigned char x_pos, unsigned char y_pos, unsigned char font_size, unsigned int colour, unsigned int back_colour, signed int value);
void print_I(unsigned char x_pos, unsigned char y_pos, unsigned char font_size, unsigned int colour, unsigned int back_colour, signed int value);
void print_D(unsigned char x_pos, unsigned char y_pos, unsigned char font_size, unsigned int colour, unsigned int back_colour, unsigned int value, unsigned char points);
void print_F(unsigned char x_pos, unsigned char y_pos, unsigned char font_size, unsigned int colour, unsigned int back_colour, float value, unsigned char points);
```

ST7735.c

```
#include "ST7735.h"


static const unsigned char fonts[96][5] =
{
     {0x00, 0x00, 0x00, 0x00, 0x00} // 20
    ,{0x00, 0x00, 0x5f, 0x00, 0x00} // 21 !
    ,{0x00, 0x07, 0x00, 0x07, 0x00} // 22 "
    ,{0x14, 0x7f, 0x14, 0x7f, 0x14} // 23 #
    ,{0x24, 0x2a, 0x7f, 0x2a, 0x12} // 24 $
    ,{0x23, 0x13, 0x08, 0x64, 0x62} // 25 %
    ,{0x36, 0x49, 0x55, 0x22, 0x50} // 26 &
    ,{0x00, 0x05, 0x03, 0x00, 0x00} // 27 '
    ,{0x00, 0x1c, 0x22, 0x41, 0x00} // 28 (
    ,{0x00, 0x41, 0x22, 0x1c, 0x00} // 29 )
    ,{0x14, 0x08, 0x3e, 0x08, 0x14} // 2a *
    ,{0x08, 0x08, 0x3e, 0x08, 0x08} // 2b +
    ,{0x00, 0x50, 0x30, 0x00, 0x00} // 2c ,
    ,{0x08, 0x08, 0x08, 0x08, 0x08} // 2d -
    ,{0x00, 0x60, 0x60, 0x00, 0x00} // 2e .
```

```
    ,{0x20, 0x10, 0x08, 0x04, 0x02} // 2f /
    ,{0x3e, 0x51, 0x49, 0x45, 0x3e} // 30 0
    ,{0x00, 0x42, 0x7f, 0x40, 0x00} // 31 1
    ,{0x42, 0x61, 0x51, 0x49, 0x46} // 32 2
    ,{0x21, 0x41, 0x45, 0x4b, 0x31} // 33 3
    ,{0x18, 0x14, 0x12, 0x7f, 0x10} // 34 4
    ,{0x27, 0x45, 0x45, 0x45, 0x39} // 35 5
    ,{0x3c, 0x4a, 0x49, 0x49, 0x30} // 36 6
    ,{0x01, 0x71, 0x09, 0x05, 0x03} // 37 7
    ,{0x36, 0x49, 0x49, 0x49, 0x36} // 38 8
    ,{0x06, 0x49, 0x49, 0x29, 0x1e} // 39 9
    ,{0x00, 0x36, 0x36, 0x00, 0x00} // 3a :
    ,{0x00, 0x56, 0x36, 0x00, 0x00} // 3b ;
    ,{0x08, 0x14, 0x22, 0x41, 0x00} // 3c <
    ,{0x14, 0x14, 0x14, 0x14, 0x14} // 3d =
    ,{0x00, 0x41, 0x22, 0x14, 0x08} // 3e >
    ,{0x02, 0x01, 0x51, 0x09, 0x06} // 3f ?
    ,{0x32, 0x49, 0x79, 0x41, 0x3e} // 40 @
    ,{0x7e, 0x11, 0x11, 0x11, 0x7e} // 41 A
    ,{0x7f, 0x49, 0x49, 0x49, 0x36} // 42 B
    ,{0x3e, 0x41, 0x41, 0x41, 0x22} // 43 C
    ,{0x7f, 0x41, 0x41, 0x22, 0x1c} // 44 D
    ,{0x7f, 0x49, 0x49, 0x49, 0x41} // 45 E
    ,{0x7f, 0x09, 0x09, 0x09, 0x01} // 46 F
    ,{0x3e, 0x41, 0x49, 0x49, 0x7a} // 47 G
    ,{0x7f, 0x08, 0x08, 0x08, 0x7f} // 48 H
    ,{0x00, 0x41, 0x7f, 0x41, 0x00} // 49 I
    ,{0x20, 0x40, 0x41, 0x3f, 0x01} // 4a J
    ,{0x7f, 0x08, 0x14, 0x22, 0x41} // 4b K
    ,{0x7f, 0x40, 0x40, 0x40, 0x40} // 4c L
    ,{0x7f, 0x02, 0x0c, 0x02, 0x7f} // 4d M
    ,{0x7f, 0x04, 0x08, 0x10, 0x7f} // 4e N
    ,{0x3e, 0x41, 0x41, 0x41, 0x3e} // 4f O
    ,{0x7f, 0x09, 0x09, 0x09, 0x06} // 50 P
    ,{0x3e, 0x41, 0x51, 0x21, 0x5e} // 51 Q
    ,{0x7f, 0x09, 0x19, 0x29, 0x46} // 52 R
    ,{0x46, 0x49, 0x49, 0x49, 0x31} // 53 S
    ,{0x01, 0x01, 0x7f, 0x01, 0x01} // 54 T
    ,{0x3f, 0x40, 0x40, 0x40, 0x3f} // 55 U
    ,{0x1f, 0x20, 0x40, 0x20, 0x1f} // 56 V
    ,{0x3f, 0x40, 0x38, 0x40, 0x3f} // 57 W
    ,{0x63, 0x14, 0x08, 0x14, 0x63} // 58 X
    ,{0x07, 0x08, 0x70, 0x08, 0x07} // 59 Y
    ,{0x61, 0x51, 0x49, 0x45, 0x43} // 5a Z
    ,{0x00, 0x7f, 0x41, 0x41, 0x00} // 5b [
    ,{0x02, 0x04, 0x08, 0x10, 0x20} // 5c ?
    ,{0x00, 0x41, 0x41, 0x7f, 0x00} // 5d ]
    ,{0x04, 0x02, 0x01, 0x02, 0x04} // 5e ^
    ,{0x40, 0x40, 0x40, 0x40, 0x40} // 5f _
    ,{0x00, 0x01, 0x02, 0x04, 0x00} // 60 `
    ,{0x20, 0x54, 0x54, 0x54, 0x78} // 61 a
    ,{0x7f, 0x48, 0x44, 0x44, 0x38} // 62 b
    ,{0x38, 0x44, 0x44, 0x44, 0x20} // 63 c
    ,{0x38, 0x44, 0x44, 0x48, 0x7f} // 64 d
    ,{0x38, 0x54, 0x54, 0x54, 0x18} // 65 e
    ,{0x08, 0x7e, 0x09, 0x01, 0x02} // 66 f
    ,{0x0c, 0x52, 0x52, 0x52, 0x3e} // 67 g
    ,{0x7f, 0x08, 0x04, 0x04, 0x78} // 68 h
    ,{0x00, 0x44, 0x7d, 0x40, 0x00} // 69 i
    ,{0x20, 0x40, 0x44, 0x3d, 0x00} // 6a j
    ,{0x7f, 0x10, 0x28, 0x44, 0x00} // 6b k
    ,{0x00, 0x41, 0x7f, 0x40, 0x00} // 6c l
    ,{0x7c, 0x04, 0x18, 0x04, 0x78} // 6d m
    ,{0x7c, 0x08, 0x04, 0x04, 0x78} // 6e n
    ,{0x38, 0x44, 0x44, 0x44, 0x38} // 6f o
    ,{0x7c, 0x14, 0x14, 0x14, 0x08} // 70 p
```

```
    ,{0x08, 0x14, 0x14, 0x18, 0x7c} // 71 q
    ,{0x7c, 0x08, 0x04, 0x04, 0x08} // 72 r
    ,{0x48, 0x54, 0x54, 0x54, 0x20} // 73 s
    ,{0x04, 0x3f, 0x44, 0x40, 0x20} // 74 t
    ,{0x3c, 0x40, 0x40, 0x20, 0x7c} // 75 u
    ,{0x1c, 0x20, 0x40, 0x20, 0x1c} // 76 v
    ,{0x3c, 0x40, 0x30, 0x40, 0x3c} // 77 w
    ,{0x44, 0x28, 0x10, 0x28, 0x44} // 78 x
    ,{0x0c, 0x50, 0x50, 0x50, 0x3c} // 79 y
    ,{0x44, 0x64, 0x54, 0x4c, 0x44} // 7a z
    ,{0x00, 0x08, 0x36, 0x41, 0x00} // 7b {
    ,{0x00, 0x00, 0x7f, 0x00, 0x00} // 7c |
    ,{0x00, 0x41, 0x36, 0x08, 0x00} // 7d }
    ,{0x10, 0x08, 0x08, 0x10, 0x08} // 7e ?
    ,{0x78, 0x46, 0x41, 0x46, 0x78} // 7f ?
};


static unsigned char width = ST7735_TFTWIDTH;
static unsigned char length = ST7735_TFTLENGTH;



void SPI_DIO_init(void)
{
    GPIO_setAsPeripheralModuleFunctionInputPin(MISO_port, MISO_pin);

    GPIO_setAsPeripheralModuleFunctionOutputPin(MOSI_port, MOSI_pin);
    GPIO_setAsPeripheralModuleFunctionOutputPin(CLK_port, CLK_pin);

    GPIO_setAsOutputPin(RST_port, RST_pin);
    GPIO_setDriveStrength(RST_port, RST_pin, GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(RS_port, RS_pin);
    GPIO_setDriveStrength(RS_port, RS_pin, GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(CS_port, CS_pin);
    GPIO_setDriveStrength(CS_port, CS_pin, GPIO_FULL_OUTPUT_DRIVE_STRENGTH);
}


void USCI_SPI_init(void)
{
    USCI_B_SPI_initMasterParam SPI_param = {0};

    SPI_DIO_init();

    SPI_param.selectClockSource = USCI_B_SPI_CLOCKSOURCE_SMCLK;
    SPI_param.clockSourceFrequency = UCS_getSMCLK();
    SPI_param.desiredSpiClock = 2000000;
    SPI_param.msbFirst = USCI_B_SPI_MSB_FIRST;
    SPI_param.clockPhase = USCI_B_SPI_PHASE_DATA_CAPTURED_ONFIRST_CHANGED_ON_NEXT;
    SPI_param.clockPolarity = USCI_B_SPI_CLOCKPOLARITY_INACTIVITY_HIGH;

    USCI_B_SPI_initMaster(USCI_B0_BASE, &SPI_param);

    USCI_B_SPI_enable(USCI_B0_BASE);
}


void ST7735_init(void)
{
    USCI_SPI_init();

    ST7735_Reset();
```

```
    ST7735_Write(ST7735_SWRESET, CMD);
    delay_us(150);
    ST7735_Write(ST7735_SLPOUT, CMD);
    delay_us(150);

    ST7735_Write(ST7735_FRMCTR1, CMD);
    ST7735_Write(0x01, DAT);
    ST7735_Write(0x2C, DAT);
    ST7735_Write(0x2D, DAT);

    ST7735_Write(ST7735_FRMCTR2, CMD);
    ST7735_Write(0x01, DAT);
    ST7735_Write(0x2C, DAT);
    ST7735_Write(0x2D, DAT);

    ST7735_Write(ST7735_FRMCTR3, CMD);
    ST7735_Write(0x01, DAT);
    ST7735_Write(0x2C, DAT);
    ST7735_Write(0x2D, DAT);
    ST7735_Write(0x01, DAT);
    ST7735_Write(0x2C, DAT);
    ST7735_Write(0x2D, DAT);

    ST7735_Write(ST7735_INVCTR, CMD);
    ST7735_Write(0x07, DAT);

    ST7735_Write(ST7735_PWCTR1, CMD);
    ST7735_Write(0xA2, DAT);
    ST7735_Write(0x02, DAT);
    ST7735_Write(0x84, DAT);

    ST7735_Write(ST7735_PWCTR1, CMD);
    ST7735_Write(0xC5, DAT);

    ST7735_Write(ST7735_PWCTR2, CMD);
    ST7735_Write(0x0A, DAT);
    ST7735_Write(0x00, DAT);

    ST7735_Write(ST7735_PWCTR3, CMD);
    ST7735_Write(0x8A, DAT);
    ST7735_Write(0x2A, DAT);

    ST7735_Write(ST7735_PWCTR4, CMD);
    ST7735_Write(0x8A, DAT);
    ST7735_Write(0xEE, DAT);

    ST7735_Write(ST7735_PWCTR5, CMD);
    ST7735_Write(0x0E, DAT);

    ST7735_Write(ST7735_VMCTR1, CMD);
    ST7735_Write(0x00, DAT);

    ST7735_Write(ST7735_COLMOD, CMD);
    ST7735_Write(0x05, DAT);

    ST7735_Write(ST7735_MADCTL, CMD);
    ST7735_Write(0xC8, DAT);

    ST7735_RAM_Address_Set();

    ST7735_Write(ST7735_GMCTRP1, CMD);
    ST7735_Write(0x02, DAT);
    ST7735_Write(0x1C, DAT);
    ST7735_Write(0x07, DAT);
    ST7735_Write(0x12, DAT);
```

```
    ST7735_Write(0x37, DAT);
    ST7735_Write(0x32, DAT);
    ST7735_Write(0x29, DAT);
    ST7735_Write(0x2D, DAT);
    ST7735_Write(0x29, DAT);
    ST7735_Write(0x25, DAT);
    ST7735_Write(0x2B, DAT);
    ST7735_Write(0x39, DAT);
    ST7735_Write(0x00, DAT);
    ST7735_Write(0x01, DAT);
    ST7735_Write(0x03, DAT);
    ST7735_Write(0x10, DAT);

    ST7735_Write(ST7735_GMCTRN1, CMD);
    ST7735_Write(0x03, DAT);
    ST7735_Write(0x1D, DAT);
    ST7735_Write(0x07, DAT);
    ST7735_Write(0x06, DAT);
    ST7735_Write(0x2E, DAT);
    ST7735_Write(0x2C, DAT);
    ST7735_Write(0x29, DAT);
    ST7735_Write(0x2D, DAT);
    ST7735_Write(0x2E, DAT);
    ST7735_Write(0x2E, DAT);
    ST7735_Write(0x37, DAT);
    ST7735_Write(0x3F, DAT);
    ST7735_Write(0x00, DAT);
    ST7735_Write(0x00, DAT);
    ST7735_Write(0x02, DAT);
    ST7735_Write(0x10, DAT);

    ST7735_Write(ST7735_NORON, CMD);
    delay_ms(10);

    ST7735_Write(ST7735_DISPON, CMD);
    delay_ms(100);

    ST7735_Write(ST7735_RAMWR, CMD);
    delay_ms(100);
}


void ST7735_Write(unsigned char value, unsigned char mode)
{
    CS_pin_low();

    if(mode != 0)
    {
        RS_pin_high();
    }
    else
    {
        RS_pin_low();
    }

    USCI_B_SPI_transmitData(USCI_B0_BASE, value);
    while(USCI_B_SPI_isBusy(USCI_B0_BASE));

    CS_pin_high();
}


void ST7735_Reset(void)
{
    RST_pin_low();
```

```
    delay_ms(2);
    RST_pin_high();
    delay_ms(2);
}


void ST7735_Word_Write(unsigned int value)
{
    ST7735_Write(((value & 0xFF00) >> 0x08), DAT);
    ST7735_Write((value & 0x00FF), DAT);
}


void ST7735_Data_Write(unsigned char DataH, unsigned char DataL)
{
    ST7735_Write(DataH, DAT);
    ST7735_Write(DataL, DAT);
}


void ST7735_Data_Write_4k(unsigned char DataH, unsigned char DataM, unsigned char DataL)
{
    ST7735_Write(DataH, DAT);
    ST7735_Write(DataM, DAT);
    ST7735_Write(DataL, DAT);
}


void ST7735_Set_Addr_Window(unsigned char xs, unsigned char ys, unsigned char xe, unsigned
char ye)
{
    ST7735_Write(ST7735_CASET, CMD);
    ST7735_Write(0x00, DAT);
    ST7735_Write(xs, DAT);
    ST7735_Write(0x00, DAT);
    ST7735_Write(xe, DAT);

    ST7735_Write(ST7735_RASET, CMD);
    ST7735_Write(0x00, DAT);
    ST7735_Write(ys, DAT);
    ST7735_Write(0x00, DAT);
    ST7735_Write(ye, DAT);

    ST7735_Write(ST7735_RAMWR, CMD);
}


void ST7735_RAM_Address_Set(void)
{
    ST7735_Set_Addr_Window(0x00, 0x00, 0x7F, 0x9F);
}


void ST7735_Invert_Display(unsigned char i)
{
     if(i == ST7735_INVON)
     {
         ST7735_Write(ST7735_INVON, CMD);
     }
     else
     {
         ST7735_Write(ST7735_INVOFF, CMD);
     }
}
```

```
unsigned int ST7735_Swap_Colour(unsigned int colour)
{
    return ((colour << 0x000B) | (colour & 0x07E0) | (colour >> 0x000B));
}


unsigned int ST7735_Color565(unsigned char r, unsigned char g, unsigned char b)
{
    return (((r & 0xF8) << 0x08) | ((g & 0xFC) << 0x03) | (b >> 0x03));
}


void ST7735_Set_Rotation(unsigned char m)
{
    unsigned char rotation = 0x00;

    ST7735_Write(ST7735_MADCTL, CMD);
    rotation = (m % 4);

    switch(rotation)
    {
        case 0:
        {
            ST7735_Write((MADCTL_MX | MADCTL_MY | MADCTL_RGB), DAT);
            width = ST7735_TFTWIDTH;
            length = ST7735_TFTLENGTH;
            break;
        }
        case 1:
        {
            ST7735_Write((MADCTL_MY | MADCTL_MV | MADCTL_RGB), DAT);
            width = ST7735_TFTLENGTH;
            length = ST7735_TFTWIDTH;
            break;
        }
        case 2:
        {
            ST7735_Write((MADCTL_RGB), DAT);
            width = ST7735_TFTWIDTH;
            length = ST7735_TFTLENGTH;
            break;
        }
        case 3:
        {
            ST7735_Write((MADCTL_MX | MADCTL_MV | MADCTL_RGB), DAT);
            width = ST7735_TFTLENGTH;
            length = ST7735_TFTWIDTH;
            break;
        }
    }
}


void TFT_fill(unsigned int colour)
{
    unsigned char i = 0x00;
    unsigned char j = 0x00;

    ST7735_Set_Addr_Window(0, 0, (width - 1), (length - 1));

    for(j = length; j > 0; j--)
    {
        for(i = width; i > 0; i--)
        {
            ST7735_Word_Write(colour);
```

```
        }
    }
}


void Draw_Pixel(unsigned char x_pos, unsigned char y_pos, unsigned int colour)
{
    ST7735_Set_Addr_Window(x_pos, y_pos, (1 + x_pos), (1 + y_pos));
     ST7735_Word_Write(colour);
}


void Draw_Line(signed int x1, signed int y1, signed int x2, signed int y2, unsigned int col
our)
{
    signed int dx = 0x0000;
    signed int dy = 0x0000;
    signed int stepx = 0x0000;
    signed int stepy = 0x0000;
    signed int fraction = 0x0000;

    dy = (y2 - y1);
    dx = (x2 - x1);

    if (dy < 0)
    {
        dy = -dy;
        stepy = -1;
    }
    else
    {
        stepy = 1;
    }

    if (dx < 0)
    {
        dx = -dx;
        stepx = -1;
    }
    else
    {
        stepx = 1;
    }

    dx <<= 0x01;
    dy <<= 0x01;

    Draw_Pixel(x1, y1, colour);

    if (dx > dy)
    {
        fraction = (dy - (dx >> 1));
        while (x1 != x2)
        {
            if (fraction >= 0)
            {
                y1 += stepy;
                fraction -= dx;
            }
            x1 += stepx;
            fraction += dy;

            Draw_Pixel(x1, y1, colour);
        }
    }
```

```
    else
    {
        fraction = (dx - (dy >> 1));

        while (y1 != y2)
        {
            if (fraction >= 0)
            {
                x1 += stepx;
                fraction -= dy;
            }
            y1 += stepy;
            fraction += dx;
            Draw_Pixel(x1, y1, colour);
        }
    }
}


void Draw_Rectangle(signed int x1, signed int y1, signed int x2, signed int y2, unsigned ch
ar fill, unsigned char type, unsigned int colour, unsigned int back_colour)
{
     unsigned char i = 0x00;
     unsigned char xmin = 0x00;
     unsigned char xmax = 0x00;
     unsigned char ymin = 0x00;
     unsigned char ymax = 0x00;

     if(fill != NO)
     {
        if(x1 < x2)
        {
           xmin = x1;
           xmax = x2;
        }
        else
        {
           xmin = x2;
           xmax = x1;
        }

        if(y1 < y2)
        {
           ymin = y1;
           ymax = y2;
        }
        else
        {
           ymin = y2;
           ymax = y1;
        }

        for(; xmin <= xmax; ++xmin)
        {
             for(i = ymin; i <= ymax; ++i)
             {
                 Draw_Pixel(xmin, i, colour);
             }
         }
     }

     else
     {
        Draw_Line(x1, y1, x2, y1, colour);
        Draw_Line(x1, y2, x2, y2, colour);
        Draw_Line(x1, y1, x1, y2, colour);
```

```
        Draw_Line(x2, y1, x2, y2, colour);
     }

     if(type != SQUARE)
     {
         Draw_Pixel(x1, y1, back_colour);
         Draw_Pixel(x1, y2, back_colour);
         Draw_Pixel(x2, y1, back_colour);
         Draw_Pixel(x2, y2, back_colour);
     }
}


void Draw_Circle(signed int xc, signed int yc, signed int radius, unsigned char fill, unsig
ned int colour)
{
   signed int a = 0x0000;
   signed int b = 0x0000;
   signed int p = 0x0000;

   b = radius;
   p = (1 - b);

   do
   {
        if(fill != NO)
        {
            Draw_Line((xc - a), (yc + b), (xc + a), (yc + b), colour);
            Draw_Line((xc - a), (yc - b), (xc + a), (yc - b), colour);
            Draw_Line((xc - b), (yc + a), (xc + b), (yc + a), colour);
            Draw_Line((xc - b), (yc - a), (xc + b), (yc - a), colour);
        }
        else
        {
            Draw_Pixel((xc + a), (yc + b), colour);
            Draw_Pixel((xc + b), (yc + a), colour);
            Draw_Pixel((xc - a), (yc + b), colour);
            Draw_Pixel((xc - b), (yc + a), colour);
            Draw_Pixel((xc + b), (yc - a), colour);
            Draw_Pixel((xc + a), (yc - b), colour);
            Draw_Pixel((xc - a), (yc - b), colour);
            Draw_Pixel((xc - b), (yc - a), colour);
        }

        if(p < 0)
        {
            p += (0x03 + (0x02 * a++));
        }
        else
        {
            p += (0x05 + (0x02 * ((a++) - (b--))));
        }
    }while(a <= b);
}


void Draw_Font_Pixel(unsigned char x_pos, unsigned char y_pos, unsigned int colour, unsigne
d char pixel_size)
{
     unsigned char i = 0x00;

     ST7735_Set_Addr_Window(x_pos, y_pos, (x_pos + pixel_size - 1), (y_pos + pixel_size - 1
));

     for(i = 0x00; i < (pixel_size * pixel_size); i++)
```

```
    {
        ST7735_Word_Write(colour);
    }
}


void print_char(unsigned char x_pos, unsigned char y_pos, unsigned char font_size, unsigned
 int colour, unsigned int back_colour, unsigned char ch)
{
    unsigned char i = 0x00;
    unsigned char j = 0x00;

    unsigned char value = 0x00;

    if(font_size < 1)
    {
        font_size = 1;
    }

    if(x_pos < font_size)
    {
        x_pos = font_size;
    }

    for (i = 0x00; i < 0x05; i++)
    {
        for (j = 0x00; j < 0x08; j++)
        {
            value = 0x00;
            value = ((fonts[ch - 0x20][i]));

            if((value >> j)  & 0x01)
            {
                Draw_Font_Pixel(x_pos, y_pos, colour, font_size);
            }
            else
            {
                Draw_Font_Pixel(x_pos, y_pos, back_colour, font_size);
            }

            y_pos = y_pos + font_size;
         }
         y_pos -= (font_size << 0x03);
         x_pos += font_size;

     }
     x_pos += font_size;

     if(x_pos > width)
     {
         x_pos = (font_size + 0x01);
         y_pos += (font_size << 0x03);
     }
}


void print_str(unsigned char x_pos, unsigned char y_pos, unsigned char font_size, unsigned
int colour, unsigned int back_colour, unsigned char *ch)
{
    do
    {
        print_char(x_pos, y_pos, font_size, colour, back_colour, *ch++);
        x_pos += (font_size * 0x06);
    }while((*ch >= 0x20) && (*ch <= 0x7F));
}
```

```
void print_C(unsigned char x_pos, unsigned char y_pos, unsigned char font_size, unsigned in
t colour, unsigned int back_colour, signed int value)
{
    unsigned char ch[5] = {0x20, 0x20, 0x20, 0x20, 0x20};

     if(value < 0x00)
     {

        ch[0] = 0x2D;
        value = -value;
     }
     else
     {
        ch[0] = 0x20;
     }

     if((value > 99) && (value <= 999))
     {
         ch[1] = ((value / 100) + 0x30);
         ch[2] = (((value % 100) / 10) + 0x30);
         ch[3] = ((value % 10) + 0x30);
     }
     else if((value > 9) && (value <= 99))
     {
         ch[1] = (((value % 100) / 10) + 0x30);
         ch[2] = ((value % 10) + 0x30);
         ch[3] = 0x20;
     }
     else if((value >= 0) && (value <= 9))
     {
         ch[1] = ((value % 10) + 0x30);
         ch[2] = 0x20;
         ch[3] = 0x20;
     }

     print_str(x_pos, y_pos, font_size, colour, back_colour, ch);
}


void print_I(unsigned char x_pos, unsigned char y_pos, unsigned char font_size, unsigned in
t colour, unsigned int back_colour, signed int value)
{
    unsigned char ch[7] = {0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20};

    if(value < 0)
    {
        ch[0] = 0x2D;
        value = -value;
    }
    else
    {
        ch[0] = 0x20;
    }

    if(value > 9999)
    {
        ch[1] = ((value / 10000) + 0x30);
        ch[2] = (((value % 10000)/ 1000) + 0x30);
        ch[3] = (((value % 1000) / 100) + 0x30);
        ch[4] = (((value % 100) / 10) + 0x30);
        ch[5] = ((value % 10) + 0x30);
    }
```

```
    else if((value > 999) && (value <= 9999))
    {
        ch[1] = (((value % 10000)/ 1000) + 0x30);
        ch[2] = (((value % 1000) / 100) + 0x30);
        ch[3] = (((value % 100) / 10) + 0x30);
        ch[4] = ((value % 10) + 0x30);
        ch[5] = 0x20;
    }
    else if((value > 99) && (value <= 999))
    {
        ch[1] = (((value % 1000) / 100) + 0x30);
        ch[2] = (((value % 100) / 10) + 0x30);
        ch[3] = ((value % 10) + 0x30);
        ch[4] = 0x20;
        ch[5] = 0x20;
    }
    else if((value > 9) && (value <= 99))
    {
        ch[1] = (((value % 100) / 10) + 0x30);
        ch[2] = ((value % 10) + 0x30);
        ch[3] = 0x20;
        ch[4] = 0x20;
        ch[5] = 0x20;
    }
    else
    {
        ch[1] = ((value % 10) + 0x30);
        ch[2] = 0x20;
        ch[3] = 0x20;
        ch[4] = 0x20;
        ch[5] = 0x20;
    }

    print_str(x_pos, y_pos, font_size, colour, back_colour, ch);
}


void print_D(unsigned char x_pos, unsigned char y_pos, unsigned char font_size, unsigned in
t colour, unsigned int back_colour, unsigned int value, unsigned char points)
{
    unsigned char ch[6] = {0x2E, 0x20, 0x20, 0x20, 0x20, 0x20};

    ch[1] = ((value / 1000) + 0x30);

    if(points > 1)
    {
        ch[2] = (((value % 1000) / 100) + 0x30);

        if(points > 2)
        {
            ch[3] = (((value % 100) / 10) + 0x30);

            if(points > 3)
            {
                ch[4] = ((value % 10) + 0x30);
            }
        }
    }

    print_str(x_pos, y_pos, font_size, colour, back_colour, ch);
}


void print_F(unsigned char x_pos, unsigned char y_pos, unsigned char font_size, unsigned in
t colour, unsigned int back_colour, float value, unsigned char points)
```

```
{
    signed long tmp = 0x0000;

    tmp = (signed long)value;
    print_I(x_pos, y_pos, font_size, colour, back_colour, tmp);
    tmp = ((signed long)((value - (float)tmp) * 10000));

    if(tmp < 0)
    {
       tmp = -tmp;
    }

    if((value >= 10000) && (value < 100000))
    {
        print_D((x_pos + (0x24 * font_size)), y_pos, font_size, colour, back_colour, tmp, p
oints);
    }
    else if((value >= 1000) && (value < 10000))
    {
        print_D((x_pos + (0x1E * font_size)), y_pos, font_size, colour, back_colour, tmp, p
oints);
    }
    else if((value >= 100) && (value < 1000))
    {
        print_D((x_pos + (0x18 * font_size)), y_pos, font_size, colour, back_colour, tmp, p
oints);
    }
    else if((value >= 10) && (value < 100))
    {
        print_D((x_pos + (0x12 * font_size)), y_pos, font_size, colour, back_colour, tmp, p
oints);
    }
    else if(value < 10)
    {
        print_D((x_pos + (0x0C * font_size)), y_pos, font_size, colour, back_colour, tmp, p
oints);

        if((value) < 0)
        {
            print_char(x_pos, y_pos, font_size, colour, back_colour, 0x2D);
        }
        else
        {
            print_char(x_pos, y_pos, font_size, colour, back_colour, 0x20);
        }
    }
}
```

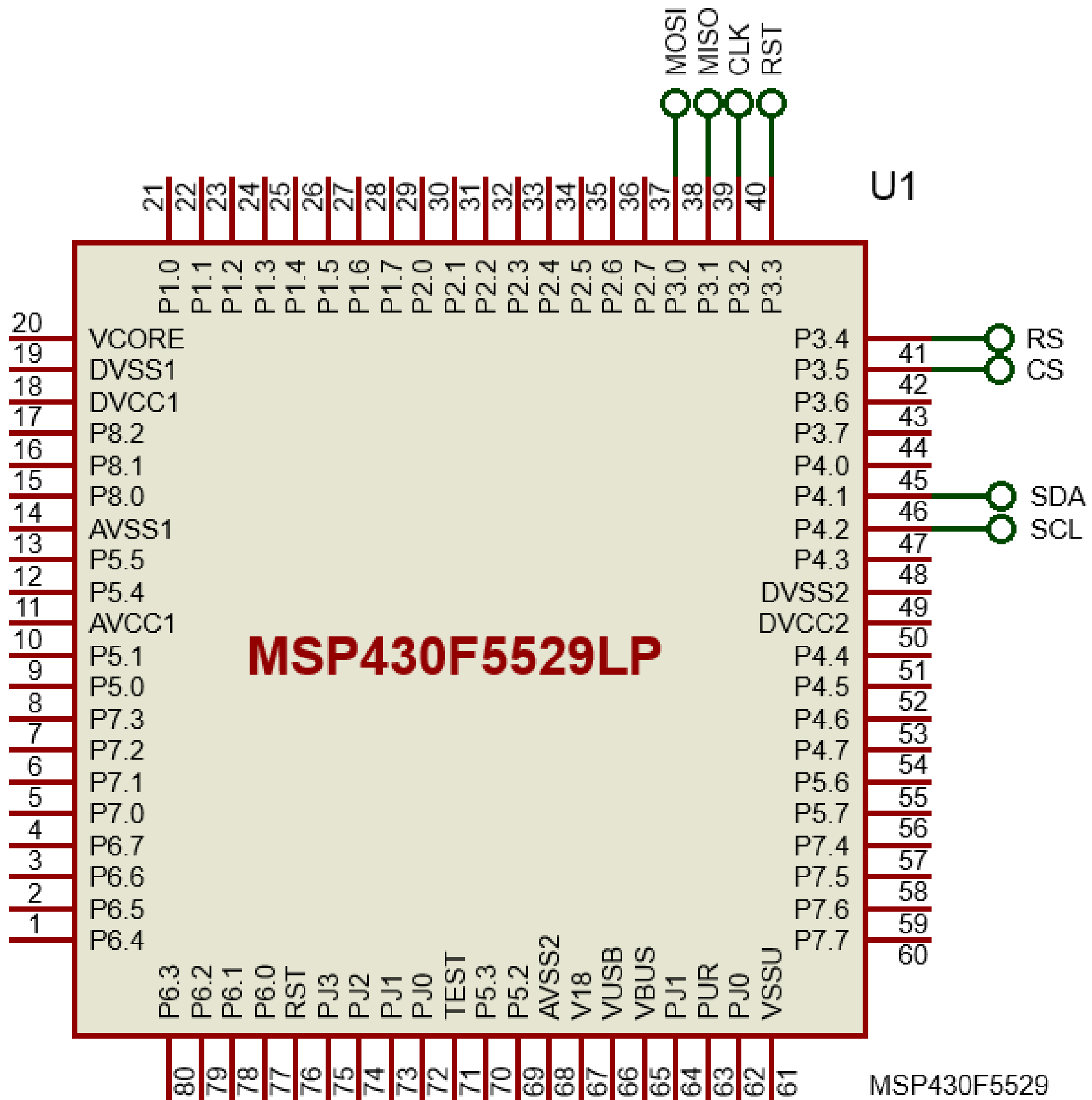
main.c

```
#include "driverlib.h"
#include "delay.h"
#include "ST7735.h"
#include "MMA7455L.h"


void clock_init(void);


void main(void)
{
    signed char x_axis_8 = 0;
    signed char y_axis_8 = 0;
    signed char z_axis_8 = 0;
```

```
    WDT_A_hold(WDT_A_BASE);

    clock_init();
    ST7735_init();
    MMA7455L_init();

    ST7735_Set_Rotation(0x02);
    TFT_fill(BLACK);

    print_str(1, 100, 1, GREEN, BLACK, "X8:");
    print_str(1, 110, 1, GREEN, BLACK, "Y8:");
    print_str(1, 120, 1, GREEN, BLACK, "Z8:");



    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setDriveStrength(GPIO_PORT_P4,
                          GPIO_PIN7,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    while(1)
    {
        x_axis_8 = MMA7455L_read_axis_8(MMA7455L_X_out);
        y_axis_8 = MMA7455L_read_axis_8(MMA7455L_Y_out);
        z_axis_8 = MMA7455L_read_axis_8(MMA7455L_Z_out);

        print_C(45, 100, 1, GREEN, BLACK, x_axis_8);
        print_C(45, 110, 1, GREEN, BLACK, y_axis_8);
        print_C(45, 120, 1, GREEN, BLACK, z_axis_8);

        Draw_Circle(x_axis_8, x_axis_8, 2, YES, CYAN);

        GPIO_toggleOutputOnPin(GPIO_PORT_P4,
                               GPIO_PIN7);

        delay_ms(100);

        Draw_Circle(x_axis_8, x_axis_8, 2, YES, BLACK);
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_3,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
```

```
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}
```

Hardware Setup



Explanation

Let's check the I2C functions of USCI first. USCI B1 is used in I2C mode here. We start by setting alternative roles of I2C GPIO pins as shown below:

```
GPIO_setAsPeripheralModuleFunctionOutputPin(I2C_port, (I2C_SDA_pin | I2C_SCL_pin));
```

After setting the I2C pins, the I2C hardware is ready to be initialized.

```
void USCI_I2C_init(void)
{
    USCI_B_I2C_initMasterParam I2C_param = {0};

    I2C_DIO_init();

    I2C_param.selectClockSource = USCI_B_I2C_CLOCKSOURCE_SMCLK;
    I2C_param.i2cClk = UCS_getSMCLK();
    I2C_param.dataRate = USCI_B_I2C_SET_DATA_RATE_100KBPS;

    USCI_B_I2C_initMaster(USCI_B1_BASE, &I2C_param);

    USCI_B_I2C_enable(USCI_B1_BASE);
}
```

For initialization, we need to set I2C module's clock source first. In this case, the clock source is SMCLK. SMCLK, in this example, is running at 1 MHz as it is being sourced by prescaled XT2 clock source.

```
UCS_initClockSignal(UCS_SMCLK, UCS_XT2CLK_SELECT, UCS_CLOCK_DIVIDER_4);
```

We have to let the USCI I2C module know the speed of SMCLK and decide our I2C clock rate. Internal mechanism scales the I2C clock source to match the desired I2C data clock rate. Here we set the clock rate to 100 kbps.

After setting all these, we enable the USCI I2C module.

Note that no interrupt has been used as polling is good enough for basic I2C communication.

Now let's check I2C write operations. Shown below are two types of I2C write operation functions. The first one is coded with driverlib and the other is what we usually do in other microcontrollers and in other software platforms. These are shown to highlight key differences.

```
void MMA7455L_write_byte(unsigned char address, unsigned char value)
{
    USCI_B_I2C_setSlaveAddress(USCI_B1_BASE, MMA7455L_address);
    USCI_B_I2C_setMode(USCI_B1_BASE, USCI_B_I2C_TRANSMIT_MODE);

    USCI_B_I2C_masterSendMultiByteStart(USCI_B1_BASE, address);
    while(!USCI_B_I2C_masterIsStartSent(USCI_B1_BASE));
    USCI_B_I2C_masterSendMultiByteFinish(USCI_B1_BASE, value);

    while(USCI_B_I2C_isBusBusy(USCI_B1_BASE));
}
```

The driverlib-based I2C write operation hides several stuffs in contrast to the operation shown below:

```
void MMA7455L_write_byte(unsigned char address, unsigned char value)
{
    I2C_Start();
    I2C_Write(MMA7455L_write_address);
    I2C_Write(address);
    I2C_Write(value);
    I2C_Stop();
    delay_ms(250);
}
```

Firstly, there is no I2C start and stop command in driverlib as these are hidden and done internally under the hood of driverlib functions. Secondly, we have to mention in our code if the I2C mode of data transaction is a transmission or a reception. Lastly, the I2C bus is polled for I2C bus status. The polling loops till the bus is free. I believe these differences are clear now.

Now let's see the I2C read operations. Just like the write operations, we will again check differences between driverlib and conventional I2C bus reading method.

```
unsigned char MMA7455L_read_byte(unsigned char address)
{
    unsigned char value = 0x00;

    USCI_B_I2C_setSlaveAddress(USCI_B1_BASE, MMA7455L_address);
    USCI_B_I2C_setMode(USCI_B1_BASE, USCI_B_I2C_TRANSMIT_MODE);

    USCI_B_I2C_masterSendStart(USCI_B1_BASE);
    USCI_B_I2C_masterSendSingleByte(USCI_B1_BASE, address);

    USCI_B_I2C_setSlaveAddress(USCI_B1_BASE, MMA7455L_address);
    USCI_B_I2C_setMode(USCI_B1_BASE, USCI_B_I2C_RECEIVE_MODE);

    USCI_B_I2C_masterReceiveSingleStart(USCI_B1_BASE);
    value = USCI_B_I2C_masterReceiveSingle(USCI_B1_BASE);

    while(USCI_B_I2C_isBusBusy(USCI_B1_BASE));

    return value;
}
```

The code snippet below shows conventional I2C bus read:

```
unsigned char MMA7455L_read_byte(unsigned char address)
{
    unsigned char value = 0;

    I2C_Start();
    I2C_Write(MMA7455L_write_address);
    I2C_Write(address);

    I2C_Restart();
    I2C_Write(MMA7455L_read_address);
    value = I2C_Read(0);
    I2C_Stop();

    return value;
}
```

The key differences between these two code snippets above are similar as the ones we already noticed in I2C write codes. I stated similar because there are some new additions to these already existing differences. Firstly, note that there is no I2C restart or start function in driverlib as sending out slave address incorporates that in secret. Secondly, during an I2C read, there is a transmission and a reception session and so there are two USCI modes in a bus read operation. Lastly, note that read acknowledgement (ACK/NACK) is not manually sent in driverlib as again it is done under the hood.

I believe, by now, the driverlib concepts for I2C are clear.

Since we are using a TFT display as an external SPI device in this example and not reading it, we will just be focusing on SPI write operation only.

Firstly, we declare all necessary GPIO pins.

```
#define MOSI_port                   GPIO_PORT_P3
#define MISO_port                   GPIO_PORT_P3
#define CLK_port                    GPIO_PORT_P3
#define RST_port                    GPIO_PORT_P3
#define RS_port                     GPIO_PORT_P3
#define CS_port                     GPIO_PORT_P3

#define MOSI_pin                    GPIO_PIN0
#define MISO_pin                    GPIO_PIN1
#define CLK_pin                     GPIO_PIN2
#define RST_pin                     GPIO_PIN3
#define RS_pin                      GPIO_PIN4
#define CS_pin                      GPIO_PIN5


#define MOSI_pin_high()             GPIO_setOutputHighOnPin(MOSI_port, MOSI_pin)
#define MOSI_pin_low()              GPIO_setOutputLowOnPin(MOSI_port, MOSI_pin)

#define get_MISO_pin()              GPIO_getInputPinValue(MISO_port, MISO_pin)

#define CLK_pin_high()              GPIO_setOutputHighOnPin(CLK_port, CLK_pin)
#define CLK_pin_low()               GPIO_setOutputLowOnPin(CLK_port, CLK_pin)

#define RST_pin_high()              GPIO_setOutputHighOnPin(RST_port, RST_pin)
#define RST_pin_low()               GPIO_setOutputLowOnPin(RST_port, RST_pin)

#define RS_pin_high()               GPIO_setOutputHighOnPin(RS_port, RS_pin)
#define RS_pin_low()                GPIO_setOutputLowOnPin(RS_port, RS_pin)

#define CS_pin_high()               GPIO_setOutputHighOnPin(CS_port, CS_pin)
#define CS_pin_low()                GPIO_setOutputLowOnPin(CS_port, CS_pin)
```

We, then, initialize the GPIOs as per pin characteristics. SPI pins, just like I2C pins, are initialized as secondary function pins while the rest are initialized as ordinary GPIOs.

```
void SPI_DIO_init(void)
{
    GPIO_setAsPeripheralModuleFunctionInputPin(MISO_port, MISO_pin);

    GPIO_setAsPeripheralModuleFunctionOutputPin(MOSI_port, MOSI_pin);
    GPIO_setAsPeripheralModuleFunctionOutputPin(CLK_port, CLK_pin);

    GPIO_setAsOutputPin(RST_port, RST_pin);
    GPIO_setDriveStrength(RST_port, RST_pin, GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(RS_port, RS_pin);
    GPIO_setDriveStrength(RS_port, RS_pin, GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(CS_port, CS_pin);
    GPIO_setDriveStrength(CS_port, CS_pin, GPIO_FULL_OUTPUT_DRIVE_STRENGTH);
}
```

Next, we initialize the USCI B0 module in SPI mode.

```
void USCI_SPI_init(void)
{
    USCI_B_SPI_initMasterParam SPI_param = {0};

    SPI_DIO_init();

    SPI_param.selectClockSource = USCI_B_SPI_CLOCKSOURCE_SMCLK;
    SPI_param.clockSourceFrequency = UCS_getSMCLK();
    SPI_param.desiredSpiClock = 2000000;
    SPI_param.msbFirst = USCI_B_SPI_MSB_FIRST;
    SPI_param.clockPhase = USCI_B_SPI_PHASE_DATA_CAPTURED_ONFIRST_CHANGED_ON_NEXT;
    SPI_param.clockPolarity = USCI_B_SPI_CLOCKPOLARITY_INACTIVITY_HIGH;

    USCI_B_SPI_initMaster(USCI_B0_BASE, &SPI_param);

    USCI_B_SPI_enable(USCI_B0_BASE);
}
```

The first part SPI module's initialization is the initialization of required GPIOs.

Like I2C, we have to declare the source of SPI clock (here SMCLK), its frequency and the SPI clock speed. SPI needs additional info like clock polarity, clock phase and data orientation. These are also needed to be set.

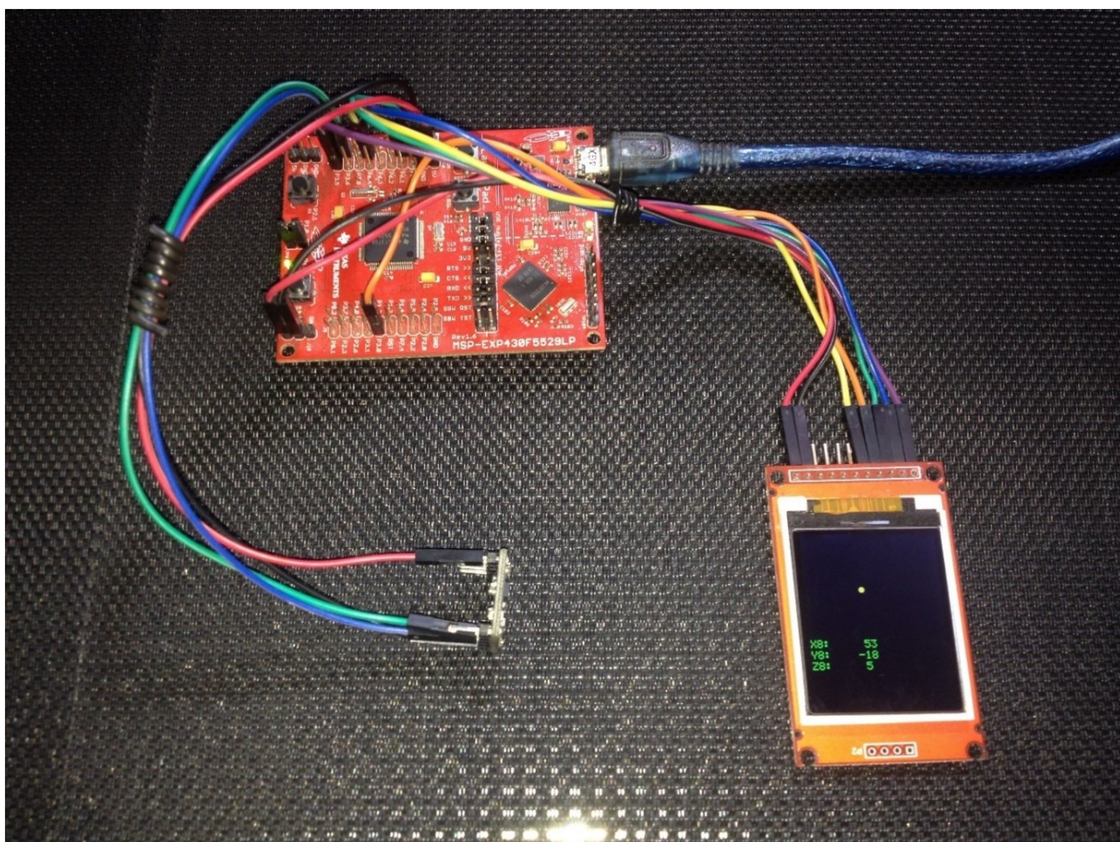
After setting all these, we have our USCI SPI module ready to rock-&-roll.

SPI write is very simple and straight forward. We just have to write desired value(s) and wait for the SPI bus to get free. As we all know, for most SPI devices, chip select pin needs to be low prior to a read/write operation and high after completion of read/write. This is done here too.

```
void ST7735_Write(unsigned char value, unsigned char mode)
{
    CS_pin_low();

    if(mode != 0)
    {
        RS_pin_high();
    }
    else
    {
        RS_pin_low();
    }

    USCI_B_SPI_transmitData(USCI_B0_BASE, value);
    while(USCI_B_SPI_isBusy(USCI_B0_BASE));

    CS_pin_high();
}
```

The demo here reads a MMA7455L accelerometer and plots a circle on a ST7735 TFT display. The coordinates of the circle are based on the X-Y values of the accelerometer. The accelerometer’s axes values are also displayed.

Demo



Demo video: https://youtu.be/SZvO0WUhLM0

USCI – SPI – I2C Example 2

We covered most of the stuffs that are needed to be understood and applied for using USCI in SPI and I2C modes. However, we didn't see any SPI read operation in our last example. This example is meant to eliminate that missing stuff while allowing us to explore and practice USCI modules further.

It is yet another over-simplified example. A MAX6675 SPI-based thermocouple-digital converter is read using a MSP430's USCI in SPI mode while an I2C-based SSD1306 OLED display is used to display the temperature read from the thermocouple.

Code Example

MAX6675.h

```
#include "driverlib.h"
#include "delay.h"


#define MOSI_port                   GPIO_PORT_P3
#define MISO_port                   GPIO_PORT_P3
#define CLK_port                    GPIO_PORT_P2
#define CS_port                     GPIO_PORT_P3

#define MOSI_pin                    GPIO_PIN3
#define MISO_pin                    GPIO_PIN4
#define CLK_pin                     GPIO_PIN7
#define CS_pin                      GPIO_PIN2

#define MOSI_pin_high()             GPIO_setOutputHighOnPin(MOSI_port, MOSI_pin)
#define MOSI_pin_low()              GPIO_setOutputLowOnPin(MOSI_port, MOSI_pin)

#define get_MISO_pin()              GPIO_getInputPinValue(MISO_port, MISO_pin)

#define CLK_pin_high()              GPIO_setOutputHighOnPin(CLK_port, CLK_pin)
#define CLK_pin_low()               GPIO_setOutputLowOnPin(CLK_port, CLK_pin)

#define CS_pin_high()               GPIO_setOutputHighOnPin(CS_port, CS_pin)
#define CS_pin_low()                GPIO_setOutputLowOnPin(CS_port, CS_pin)

#define T_min                       0
#define T_max                       1024

#define count_max                   4096

#define no_of_pulses                16

#define deg_C                       0
#define deg_F                       1
#define tmp_K                       2

#define open_contact                0x04
#define close_contact               0x00

#define scalar_deg_C                0.25
#define scalar_deg_F_1              1.8
#define scalar_deg_F_2              32.0
#define scalar_tmp_K                273.0

#define no_of_samples               16
```

```
void SPI_DIO_init(void);
void USCI_SPI_init(void);
void MAX6675_init(void);
unsigned char MAX6675_get_ADC(unsigned int *ADC_data);
float MAX6675_get_T(unsigned int ADC_value, unsigned char T_unit);
```

MAX6675.c

```
#include "MAX6675.h"


void SPI_DIO_init(void)
{
    GPIO_setAsPeripheralModuleFunctionInputPin(MISO_port, MISO_pin);

    GPIO_setAsPeripheralModuleFunctionOutputPin(MOSI_port, MOSI_pin);
    GPIO_setAsPeripheralModuleFunctionOutputPin(CLK_port, CLK_pin);

    GPIO_setAsOutputPin(CS_port, CS_pin);
    GPIO_setDriveStrength(CS_port, CS_pin, GPIO_FULL_OUTPUT_DRIVE_STRENGTH);
}


void USCI_SPI_init(void)
{
    USCI_A_SPI_initMasterParam SPI_param = {0};

    SPI_DIO_init();

    SPI_param.selectClockSource = USCI_A_SPI_CLOCKSOURCE_SMCLK;
    SPI_param.clockSourceFrequency = UCS_getSMCLK();
    SPI_param.desiredSpiClock = 1000000;
    SPI_param.msbFirst = USCI_A_SPI_MSB_FIRST;
    SPI_param.clockPhase = USCI_A_SPI_PHASE_DATA_CAPTURED_ONFIRST_CHANGED_ON_NEXT;
    SPI_param.clockPolarity = USCI_A_SPI_CLOCKPOLARITY_INACTIVITY_LOW;

    USCI_A_SPI_initMaster(USCI_A0_BASE, &SPI_param);

    USCI_A_SPI_enable(USCI_A0_BASE);
}


void MAX6675_init(void)
{
    USCI_SPI_init();
}


unsigned char MAX6675_get_ADC(unsigned int *ADC_data)
{
    unsigned char lb = 0;
    unsigned char hb = 0;
    unsigned char samples = no_of_samples;
    unsigned int temp_data = 0;
    unsigned long avg_value = 0;


    while(samples > 0)
    {
        CS_pin_low();

        USCI_A_SPI_transmitData(USCI_A0_BASE, 0x00);
```

```c
        while(USCI_A_SPI_isBusy(USCI_A0_BASE));
        hb = USCI_A_SPI_receiveData(USCI_A0_BASE);
        while(USCI_A_SPI_isBusy(USCI_A0_BASE));

        USCI_B_SPI_transmitData(USCI_A0_BASE, 0x00);
        while(USCI_A_SPI_isBusy(USCI_A0_BASE));
        lb = USCI_A_SPI_receiveData(USCI_A0_BASE);
        while(USCI_A_SPI_isBusy(USCI_A0_BASE));

         CS_pin_high();

         temp_data = hb;
         temp_data <<= 8;
         temp_data |= lb;
         temp_data &= 0x7FFF;

         avg_value += (unsigned long)temp_data;

         samples--;
         delay_ms(10);
    };

    temp_data = (avg_value >> 4);

    if((temp_data & 0x04) == close_contact)
    {
        *ADC_data = (temp_data >> 3);
        return close_contact;
    }
    else
    {
        *ADC_data = (count_max + 1);
        return open_contact;
    }
}


float MAX6675_get_T(unsigned int ADC_value, unsigned char T_unit)
{
    float tmp = 0.0;

    tmp = (((float)ADC_value) * scalar_deg_C);

    switch(T_unit)
    {
        case deg_F:
        {
             tmp *= scalar_deg_F_1;
             tmp += scalar_deg_F_2;
             break;
        }
        case tmp_K:
        {
            tmp += scalar_tmp_K;
            break;
        }
        default:
        {
            break;
        }
    }

    return tmp;
}
```

SSD1306.h

```
#include "driverlib.h"
#include "delay.h"


#define I2C_port                                  GPIO_PORT_P3

#define I2C_SDA_pin                               GPIO_PIN0
#define I2C_SCL_pin                               GPIO_PIN1

#define SSD1306_I2C_Address                       0x3C

#define DAT                                       0x60
#define CMD                                       0x00

#define Set_Lower_Column_Start_Address_CMD        0x00
#define Set_Higher_Column_Start_Address_CMD       0x10
#define Set_Memory_Addressing_Mode_CMD            0x20
#define Set_Column_Address_CMD                    0x21
#define Set_Page_Address_CMD                      0x22
#define Set_Display_Start_Line_CMD                0x40
#define Set_Contrast_Control_CMD                  0x81
#define Set_Charge_Pump_CMD                       0x8D
#define Set_Segment_Remap_CMD                     0xA0
#define Set_Entire_Display_ON_CMD                 0xA4
#define Set_Normal_or_Inverse_Display_CMD         0xA6
#define Set_Multiplex_Ratio_CMD                   0xA8
#define Set_Display_ON_or_OFF_CMD                 0xAE
#define Set_Page_Start_Address_CMD                0xB0
#define Set_COM_Output_Scan_Direction_CMD         0xC0
#define Set_Display_Offset_CMD                    0xD3
#define Set_Display_Clock_CMD                     0xD5
#define Set_Pre_charge_Period_CMD                 0xD9
#define Set_Common_HW_Config_CMD                  0xDA
#define Set_VCOMH_Level_CMD                       0xDB
#define Set_NOP_CMD                               0xE3

#define Horizontal_Addressing_Mode                0x00
#define Vertical_Addressing_Mode                  0x01
#define Page_Addressing_Mode                      0x02

#define Disable_Charge_Pump                       0x00
#define Enable_Charge_Pump                        0x04

#define Column_Address_0_Mapped_to_SEG0           0x00
#define Column_Address_0_Mapped_to_SEG127         0x01

#define Normal_Display                            0x00
#define Entire_Display_ON                         0x01

#define Non_Inverted_Display                      0x00
#define Inverted_Display                          0x01

#define Display_OFF                               0x00
#define Display_ON                                0x01

#define Scan_from_COM0_to_63                      0x00
#define Scan_from_COM63_to_0                      0x08

#define x_size                                    128
#define x_max                                     x_size
#define x_min                                     0
#define y_size                                    32
#define y_max                                     8
```

```c
#define y_min                             0

#define ON                                1
#define OFF                               0

#define YES                               1
#define NO                                0

#define ROUND                             1
#define SQUARE                            0

#define NUM                               1
#define CHR                               0

#define buffer_size                       512//(x_max * y_max)


unsigned char buffer[buffer_size];


void swap(signed int *a, signed int *b);
void I2C_DIO_init(void);
void USCI_I2C_init(void);
void OLED_init(void);
void OLED_write(unsigned char value, unsigned char control_byte);
void OLED_gotoxy(unsigned char x_pos, unsigned char y_pos);
void OLED_fill(unsigned char bmp_data);
void OLED_clear_screen(void);
void OLED_clear_buffer(void);
void OLED_cursor(unsigned char x_pos, unsigned char y_pos);
void print_char(unsigned char x_pos, unsigned char y_pos, unsigned char num_reg_state, unsi
gned char ch);
void print_string(unsigned char x_pos, unsigned char y_pos, unsigned char num_reg_state, un
signed char *ch);
void print_chr(unsigned char x_pos, unsigned char y_pos, unsigned char num_reg_state, signe
d int value);
void print_int(unsigned char x_pos, unsigned char y_pos, unsigned char num_reg_state, signe
d long value);
void print_decimal(unsigned char x_pos, unsigned char y_pos, unsigned char num_reg_state, u
nsigned int value, unsigned char points);
void print_float(unsigned char x_pos, unsigned char y_pos, unsigned char num_reg_state, flo
at value, unsigned char points);
void draw_bitmap(unsigned char xb, unsigned char yb, unsigned char xe, unsigned char ye, un
signed char *bmp_img);
void draw_pixel(unsigned char x_pos, unsigned char y_pos, unsigned char colour);
void draw_line(signed int x1, signed int y1, signed int x2, signed int y2, unsigned char co
lour);
void draw_V_line(signed int x1, signed int y1, signed int y2, unsigned colour);
void draw_H_line(signed int x1, signed int x2, signed int y1, unsigned colour);
void draw_triangle(signed int x1, signed int y1, signed int x2, signed int y2, signed int x
3, signed int y3, unsigned char fill, unsigned int colour);
void draw_rectangle(signed int x1, signed int y1, signed int x2, signed int y2, unsigned ch
ar fill, unsigned char colour, unsigned char type);
void draw_rectangle(signed int x1, signed int y1, signed int x2, signed int y2, unsigned ch
ar fill, unsigned char colour, unsigned char type);
void draw_circle(signed int xc, signed int yc, signed int radius, unsigned char fill, unsig
ned char colour);
```

SSD1306.c

```c
#include "SSD1306.h"


static const unsigned char font_regular[92][6] =
```

```
{
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00,    // sp
    0x00, 0x00, 0x00, 0x2f, 0x00, 0x00,    // !
    0x00, 0x00, 0x07, 0x00, 0x07, 0x00,    // "
    0x00, 0x14, 0x7f, 0x14, 0x7f, 0x14,    // #
    0x00, 0x24, 0x2a, 0x7f, 0x2a, 0x12,    // $
    0x00, 0x62, 0x64, 0x08, 0x13, 0x23,    // %
    0x00, 0x36, 0x49, 0x55, 0x22, 0x50,    // &
    0x00, 0x00, 0x05, 0x03, 0x00, 0x00,    // '
    0x00, 0x00, 0x1c, 0x22, 0x41, 0x00,    // (
    0x00, 0x00, 0x41, 0x22, 0x1c, 0x00,    // )
    0x00, 0x14, 0x08, 0x3E, 0x08, 0x14,    // *
    0x00, 0x08, 0x08, 0x3E, 0x08, 0x08,    // +
    0x00, 0x00, 0x00, 0xA0, 0x60, 0x00,    // ,
    0x00, 0x08, 0x08, 0x08, 0x08, 0x08,    // -
    0x00, 0x00, 0x60, 0x60, 0x00, 0x00,    // .
    0x00, 0x20, 0x10, 0x08, 0x04, 0x02,    // /
    0x00, 0x3E, 0x51, 0x49, 0x45, 0x3E,    // 0
    0x00, 0x00, 0x42, 0x7F, 0x40, 0x00,    // 1
    0x00, 0x42, 0x61, 0x51, 0x49, 0x46,    // 2
    0x00, 0x21, 0x41, 0x45, 0x4B, 0x31,    // 3
    0x00, 0x18, 0x14, 0x12, 0x7F, 0x10,    // 4
    0x00, 0x27, 0x45, 0x45, 0x45, 0x39,    // 5
    0x00, 0x3C, 0x4A, 0x49, 0x49, 0x30,    // 6
    0x00, 0x01, 0x71, 0x09, 0x05, 0x03,    // 7
    0x00, 0x36, 0x49, 0x49, 0x49, 0x36,    // 8
    0x00, 0x06, 0x49, 0x49, 0x29, 0x1E,    // 9
    0x00, 0x00, 0x36, 0x36, 0x00, 0x00,    // :
    0x00, 0x00, 0x56, 0x36, 0x00, 0x00,    // ;
    0x00, 0x08, 0x14, 0x22, 0x41, 0x00,    // <
    0x00, 0x14, 0x14, 0x14, 0x14, 0x14,    // =
    0x00, 0x00, 0x41, 0x22, 0x14, 0x08,    // >
    0x00, 0x02, 0x01, 0x51, 0x09, 0x06,    // ?
    0x00, 0x32, 0x49, 0x59, 0x51, 0x3E,    // @
    0x00, 0x7C, 0x12, 0x11, 0x12, 0x7C,    // A
    0x00, 0x7F, 0x49, 0x49, 0x49, 0x36,    // B
    0x00, 0x3E, 0x41, 0x41, 0x41, 0x22,    // C
    0x00, 0x7F, 0x41, 0x41, 0x22, 0x1C,    // D
    0x00, 0x7F, 0x49, 0x49, 0x49, 0x41,    // E
    0x00, 0x7F, 0x09, 0x09, 0x09, 0x01,    // F
    0x00, 0x3E, 0x41, 0x49, 0x49, 0x7A,    // G
    0x00, 0x7F, 0x08, 0x08, 0x08, 0x7F,    // H
    0x00, 0x00, 0x41, 0x7F, 0x41, 0x00,    // I
    0x00, 0x20, 0x40, 0x41, 0x3F, 0x01,    // J
    0x00, 0x7F, 0x08, 0x14, 0x22, 0x41,    // K
    0x00, 0x7F, 0x40, 0x40, 0x40, 0x40,    // L
    0x00, 0x7F, 0x02, 0x0C, 0x02, 0x7F,    // M
    0x00, 0x7F, 0x04, 0x08, 0x10, 0x7F,    // N
    0x00, 0x3E, 0x41, 0x41, 0x41, 0x3E,    // O
    0x00, 0x7F, 0x09, 0x09, 0x09, 0x06,    // P
    0x00, 0x3E, 0x41, 0x51, 0x21, 0x5E,    // Q
    0x00, 0x7F, 0x09, 0x19, 0x29, 0x46,    // R
    0x00, 0x46, 0x49, 0x49, 0x49, 0x31,    // S
    0x00, 0x01, 0x01, 0x7F, 0x01, 0x01,    // T
    0x00, 0x3F, 0x40, 0x40, 0x40, 0x3F,    // U
    0x00, 0x1F, 0x20, 0x40, 0x20, 0x1F,    // V
    0x00, 0x3F, 0x40, 0x38, 0x40, 0x3F,    // W
    0x00, 0x63, 0x14, 0x08, 0x14, 0x63,    // X
    0x00, 0x07, 0x08, 0x70, 0x08, 0x07,    // Y
    0x00, 0x61, 0x51, 0x49, 0x45, 0x43,    // Z
    0x00, 0x00, 0x7F, 0x41, 0x41, 0x00,    // [
    0x00, 0x02, 0x04, 0x08, 0x10, 0x20,    // \92
    0x00, 0x00, 0x41, 0x41, 0x7F, 0x00,    // ]
    0x00, 0x04, 0x02, 0x01, 0x02, 0x04,    // ^
    0x00, 0x40, 0x40, 0x40, 0x40, 0x40,    // _
    0x00, 0x00, 0x01, 0x02, 0x04, 0x00,    // '
```

```
    0x00, 0x20, 0x54, 0x54, 0x54, 0x78,   // a
    0x00, 0x7F, 0x48, 0x44, 0x44, 0x38,   // b
    0x00, 0x38, 0x44, 0x44, 0x44, 0x20,   // c
    0x00, 0x38, 0x44, 0x44, 0x48, 0x7F,   // d
    0x00, 0x38, 0x54, 0x54, 0x54, 0x18,   // e
    0x00, 0x08, 0x7E, 0x09, 0x01, 0x02,   // f
    0x00, 0x18, 0xA4, 0xA4, 0xA4, 0x7C,   // g
    0x00, 0x7F, 0x08, 0x04, 0x04, 0x78,   // h
    0x00, 0x00, 0x44, 0x7D, 0x40, 0x00,   // i
    0x00, 0x40, 0x80, 0x84, 0x7D, 0x00,   // j
    0x00, 0x7F, 0x10, 0x28, 0x44, 0x00,   // k
    0x00, 0x00, 0x41, 0x7F, 0x40, 0x00,   // l
    0x00, 0x7C, 0x04, 0x18, 0x04, 0x78,   // m
    0x00, 0x7C, 0x08, 0x04, 0x04, 0x78,   // n
    0x00, 0x38, 0x44, 0x44, 0x44, 0x38,   // o
    0x00, 0xFC, 0x24, 0x24, 0x24, 0x18,   // p
    0x00, 0x18, 0x24, 0x24, 0x18, 0xFC,   // q
    0x00, 0x7C, 0x08, 0x04, 0x04, 0x08,   // r
    0x00, 0x48, 0x54, 0x54, 0x54, 0x20,   // s
    0x00, 0x04, 0x3F, 0x44, 0x40, 0x20,   // t
    0x00, 0x3C, 0x40, 0x40, 0x20, 0x7C,   // u
    0x00, 0x1C, 0x20, 0x40, 0x20, 0x1C,   // v
    0x00, 0x3C, 0x40, 0x30, 0x40, 0x3C,   // w
    0x00, 0x44, 0x28, 0x10, 0x28, 0x44,   // x
    0x00, 0x1C, 0xA0, 0xA0, 0xA0, 0x7C,   // y
    0x00, 0x44, 0x64, 0x54, 0x4C, 0x44,   // z
    0x14, 0x14, 0x14, 0x14, 0x14, 0x14    // horiz lines
};


static const unsigned char Number_Font[11][6] =
{
   0x00, 0x7F, 0x41, 0x41, 0x41, 0x7F, //0
   0x00, 0x00, 0x00, 0x41, 0x7F, 0x40, //1
   0x00, 0x79, 0x49, 0x49, 0x49, 0x4F, //2
   0x00, 0x41, 0x49, 0x49, 0x49, 0x7F, //3
   0x00, 0x0F, 0x08, 0x08, 0x08, 0x7F, //4
   0x00, 0x4F, 0x49, 0x49, 0x49, 0x79, //5
   0x00, 0x7F, 0x48, 0x48, 0x48, 0x78, //6
   0x00, 0x00, 0x01, 0x01, 0x01, 0x7F, //7
   0x00, 0x7F, 0x49, 0x49, 0x49, 0x7F, //8
   0x00, 0x0F, 0x09, 0x09, 0x09, 0x7F, //9
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00  //spc
};


void swap(signed int *a, signed int *b)
{
    signed int temp = 0x0000;

    temp = *b;
    *b = *a;
    *a = temp;
}


void I2C_DIO_init(void)
{
    GPIO_setAsPeripheralModuleFunctionOutputPin(I2C_port, (I2C_SDA_pin | I2C_SCL_pin));
}


void USCI_I2C_init(void)
{
    USCI_B_I2C_initMasterParam I2C_param = {0};
```

```
    I2C_DIO_init();

    I2C_param.selectClockSource = USCI_B_I2C_CLOCKSOURCE_SMCLK;
    I2C_param.i2cClk = UCS_getSMCLK();
    I2C_param.dataRate = USCI_B_I2C_SET_DATA_RATE_400KBPS;

    USCI_B_I2C_initMaster(USCI_B0_BASE, &I2C_param);

    USCI_B_I2C_enable(USCI_B0_BASE);
}


void OLED_init(void)
{
     USCI_I2C_init();
     delay_ms(100);

     OLED_write((Set_Display_ON_or_OFF_CMD | Display_OFF), CMD);
     OLED_write(Set_Multiplex_Ratio_CMD, CMD);
     OLED_write(0x1F, CMD);
     OLED_write(Set_Display_Offset_CMD, CMD);
     OLED_write(0x00, CMD);
     OLED_write(Set_Display_Start_Line_CMD, CMD);
     OLED_write((Set_Segment_Remap_CMD | Column_Address_0_Mapped_to_SEG127), CMD);
     OLED_write((Set_COM_Output_Scan_Direction_CMD | Scan_from_COM63_to_0), CMD);
     OLED_write(Set_Common_HW_Config_CMD, CMD);
     OLED_write(0x02, CMD);
     OLED_write(Set_Contrast_Control_CMD, CMD);
     OLED_write(0x8F, CMD);
     OLED_write(Set_Entire_Display_ON_CMD, CMD);
     OLED_write(Set_Normal_or_Inverse_Display_CMD, CMD);
     OLED_write(Set_Display_Clock_CMD, CMD);
     OLED_write(0x80, CMD);
     OLED_write(Set_Pre_charge_Period_CMD, CMD);
     OLED_write(0x25, CMD);
     OLED_write(Set_VCOMH_Level_CMD, CMD);
     OLED_write(0x20, CMD);
     OLED_write(Set_Page_Address_CMD, CMD);
     OLED_write(0x00, CMD);
     OLED_write(0x03, CMD);
     OLED_write(Set_Page_Start_Address_CMD , CMD);
     OLED_write(Set_Higher_Column_Start_Address_CMD, CMD);
     OLED_write(Set_Lower_Column_Start_Address_CMD, CMD);
     OLED_write(Set_Memory_Addressing_Mode_CMD, CMD);
     OLED_write(0x02, CMD);
     OLED_write(Set_Charge_Pump_CMD, CMD);
     OLED_write(0x14, CMD);
     OLED_write((Set_Display_ON_or_OFF_CMD | Display_ON), CMD);

     delay_ms(100);

     OLED_clear_buffer();
     OLED_clear_screen();
}


void OLED_write(unsigned char value, unsigned char control_byte)
{
    USCI_B_I2C_setSlaveAddress(USCI_B0_BASE, SSD1306_I2C_Address);
    USCI_B_I2C_setMode(USCI_B0_BASE, USCI_B_I2C_TRANSMIT_MODE);

    USCI_B_I2C_masterSendMultiByteStart(USCI_B0_BASE, control_byte);
    while(!USCI_B_I2C_masterIsStartSent(USCI_B0_BASE));
    USCI_B_I2C_masterSendMultiByteFinish(USCI_B0_BASE, value);
```

```
    while(USCI_B_I2C_isBusBusy(USCI_B0_BASE));
}


void OLED_gotoxy(unsigned char x_pos, unsigned char y_pos)
{
     OLED_write((Set_Page_Start_Address_CMD + y_pos), CMD);
     OLED_write(((x_pos & 0x0F) | Set_Lower_Column_Start_Address_CMD), CMD);
     OLED_write((((x_pos & 0xF0) >> 0x04) | Set_Higher_Column_Start_Address_CMD), CMD);
}


void OLED_fill(unsigned char bmp_data)
{
    unsigned char x_pos = 0x00;
    unsigned char page = 0x00;

    for(page = 0; page < y_max; page++)
    {
        OLED_gotoxy(x_min, page);

        USCI_B_I2C_setSlaveAddress(USCI_B0_BASE, SSD1306_I2C_Address);
        USCI_B_I2C_setMode(USCI_B0_BASE, USCI_B_I2C_TRANSMIT_MODE);

        USCI_B_I2C_masterSendMultiByteStart(USCI_B0_BASE, DAT);
        while(!USCI_B_I2C_masterIsStartSent(USCI_B0_BASE));

        for(x_pos = x_min; x_pos < x_max; x_pos++)
        {
            USCI_B_I2C_masterSendMultiByteNext(USCI_B0_BASE, bmp_data);
        }

        USCI_B_I2C_masterSendMultiByteStop(USCI_B0_BASE);

        while(USCI_B_I2C_isBusBusy(USCI_B0_BASE));
    }
}


void OLED_clear_screen()
{
    OLED_fill(0x00);
}


void OLED_clear_buffer()
{
     unsigned int s = 0x0000;

     for(s = 0; s < buffer_size; s++)
     {
          buffer[s] = 0x00;
     }
}


void OLED_cursor(unsigned char x_pos, unsigned char y_pos)
{
    unsigned char s = 0x00;

    if(y_pos != 0x00)
    {
        if(x_pos == 1)
        {
```

```
            OLED_gotoxy(0x00, (y_pos + 0x02));
        }
        else
        {
            OLED_gotoxy((0x50 + ((x_pos - 0x02) * 0x06)), (y_pos + 0x02));
        }

        for(s = 0x00; s < 0x06; s++)
        {
            OLED_write(0xFF, DAT);
        }
    }
}


void draw_bitmap(unsigned char xb, unsigned char yb, unsigned char xe, unsigned char ye, un
signed char *bmp_img)
{
    unsigned int s = 0x0000;
    unsigned char x_pos = 0x00;
    unsigned char y_pos = 0x00;

    for(y_pos = yb; y_pos <= ye; y_pos++)
    {
        OLED_gotoxy(xb, y_pos);
        for(x_pos = xb; x_pos < xe; x_pos++)
        {
            OLED_write(bmp_img[s++], DAT);
        }
    }
}


void print_char(unsigned char x_pos, unsigned char y_pos, unsigned char num_reg_state, unsi
gned char ch)
{
    unsigned char chr = 0x00;
    unsigned char s = 0x00;

    switch(num_reg_state)
    {
        case NUM:
        {
            if((ch >= 48) && (ch <= 57))
            {
                chr = (ch - 48);
            }
            else
            {
                chr = 10;
            }
            break;
        }
        default:
        {
            chr = (ch - 0x20);
            break;
        }
    }

    if(x_pos > (x_max - 0x06))
    {
        x_pos = 0x00;
        y_pos++;
    }
```

```
    OLED_gotoxy(x_pos, y_pos);

    for(s = 0x00; s < 0x06; s++)
    {
        switch(num_reg_state)
        {
            case NUM:
            {
                OLED_write(Number_Font[chr][s], DAT);
                break;
            }
            default:
            {
                OLED_write(font_regular[chr][s], DAT);
                break;
            }
        }
    }
}


void print_string(unsigned char x_pos, unsigned char y_pos, unsigned char num_reg_state, un
signed char *ch)
{
    unsigned char s = 0x00;

    do
    {
        print_char(x_pos, y_pos, num_reg_state, ch[s++]);
        x_pos += 0x06;
     }while((ch[s] >= 0x20) && (ch[s] <= 0x7F));
}


void print_chr(unsigned char x_pos, unsigned char y_pos, unsigned char num_reg_state, signe
d int value)
{
    unsigned char ch = 0x00;

    if(value < 0x00)
    {
        print_char(x_pos, y_pos, CHR, '-');
        value = -value;
    }
    else
    {
        print_char(x_pos, y_pos, CHR, ' ');
    }

     if((value >= 100) && (value < 1000))
     {
         ch = (value / 100);
         print_char((x_pos + 6), y_pos, num_reg_state,  (48 + ch));
         ch = ((value % 100) / 10);
         print_char((x_pos + 12), y_pos, num_reg_state, (48 + ch));
         ch = (value % 10);
         print_char((x_pos + 18), y_pos, num_reg_state, (48 + ch));
     }
     else if((value >= 10) && (value < 100))
     {
         ch = ((value % 100) / 10);
         print_char((x_pos + 6), y_pos, num_reg_state, (48 + ch));
         ch = (value % 10);
         print_char((x_pos + 12), y_pos, num_reg_state, (48 + ch));
         print_char((x_pos + 18), y_pos, num_reg_state, 0x20);
```

```
     }
     else if((value >= 0) && (value < 10))
     {
         ch = (value % 10);
         print_char((x_pos + 6), y_pos, num_reg_state, (48 + ch));
         print_char((x_pos + 12), y_pos, num_reg_state, 0x20);
         print_char((x_pos + 18), y_pos, num_reg_state, 0x20);
     }
}


void print_int(unsigned char x_pos, unsigned char y_pos, unsigned char num_reg_state, signe
d long value)
{
    unsigned char ch = 0x00;

    if(value < 0)
    {
        print_char(x_pos, y_pos, CHR, '-');
        value = -value;
    }
    else
    {
        print_char(x_pos, y_pos, CHR, ' ');
    }

    if((value >= 10000) && ((value < 100000)))
    {
        ch = (value / 10000);
        print_char((x_pos + 6), y_pos, num_reg_state, (48 + ch));

        ch = ((value % 10000)/ 1000);
        print_char((x_pos + 12), y_pos, num_reg_state, (48 + ch));

        ch = ((value % 1000) / 100);
        print_char((x_pos + 18), y_pos, num_reg_state, (48 + ch));

        ch = ((value % 100) / 10);
        print_char((x_pos + 24), y_pos, num_reg_state, (48 + ch));

        ch = (value % 10);
        print_char((x_pos + 30), y_pos, num_reg_state, (48 + ch));
    }

    else if((value >= 1000) && (value < 10000))
    {
        ch = ((value % 10000)/ 1000);
        print_char((x_pos + 6), y_pos, num_reg_state, (48 + ch));

        ch = ((value % 1000) / 100);
        print_char((x_pos + 12), y_pos, num_reg_state, (48 + ch));

        ch = ((value % 100) / 10);
        print_char((x_pos + 18), y_pos, num_reg_state, (48 + ch));

        ch = (value % 10);
        print_char((x_pos + 24), y_pos, num_reg_state, (48 + ch));
        print_char((x_pos + 30), y_pos, num_reg_state, 0x20);
    }
    else if((value >= 100) && (value < 1000))
    {
        ch = ((value % 1000) / 100);
        print_char((x_pos + 6), y_pos, num_reg_state, (48 + ch));

        ch = ((value % 100) / 10);
        print_char((x_pos + 12), y_pos, num_reg_state, (48 + ch));
```

```
        ch = (value % 10);
        print_char((x_pos + 18), y_pos, num_reg_state, (48 + ch));
        print_char((x_pos + 24), y_pos, num_reg_state, 0x20);
        print_char((x_pos + 30), y_pos, num_reg_state, 0x20);
    }
    else if((value >= 10) && (value < 100))
    {
        ch = ((value % 100) / 10);
        print_char((x_pos + 6), y_pos, num_reg_state, (48 + ch));

        ch = (value % 10);
        print_char((x_pos + 12), y_pos, num_reg_state, (48 + ch));

        print_char((x_pos + 18), y_pos, num_reg_state, 0x20);
        print_char((x_pos + 24), y_pos, num_reg_state, 0x20);
        print_char((x_pos + 30), y_pos, num_reg_state, 0x20);
    }
    else if((value >= 0) && (value < 10))
    {
        ch = (value % 10);
        print_char((x_pos + 6), y_pos, num_reg_state, (48 + ch));
        print_char((x_pos + 12), y_pos, num_reg_state, 0x20);
        print_char((x_pos + 18), y_pos, num_reg_state, 0x20);
        print_char((x_pos + 24), y_pos, num_reg_state, 0x20);
        print_char((x_pos + 30), y_pos, num_reg_state, 0x20);
    }
}


void print_decimal(unsigned char x_pos, unsigned char y_pos, unsigned char num_reg_state, u
nsigned int value, unsigned char points)
{
    unsigned char ch = 0x00;

    print_char(x_pos, y_pos, CHR, '.');

    ch = (value / 1000);
    print_char((x_pos + 6), y_pos, num_reg_state, (48 + ch));

    if(points > 1)
    {
        ch = ((value % 1000) / 100);
        print_char((x_pos + 12), y_pos, num_reg_state, (48 + ch));


        if(points > 2)
        {
            ch = ((value % 100) / 10);
            print_char((x_pos + 18), y_pos, num_reg_state, (48 + ch));

            if(points > 3)
            {
                ch = (value % 10);
                print_char((x_pos + 24), y_pos, num_reg_state, (48 + ch));
            }
        }
    }
}


void print_float(unsigned char x_pos, unsigned char y_pos, unsigned char num_reg_state, flo
at value, unsigned char points)
{
    signed long tmp = 0x00;
```

```
    tmp = value;
    print_int(x_pos, y_pos, num_reg_state, tmp);
    tmp = ((value - tmp) * 10000);

    if(tmp < 0)
    {
       tmp = -tmp;
    }

    if((value >= 10000) && (value < 100000))
    {
        print_decimal((x_pos + 36), y_pos, num_reg_state, tmp, points);
    }
    else if((value >= 1000) && (value < 10000))
    {
        print_decimal((x_pos + 30), y_pos, num_reg_state, tmp, points);
    }
    else if((value >= 100) && (value < 1000))
    {
        print_decimal((x_pos + 24), y_pos, num_reg_state, tmp, points);
    }
    else if((value >= 10) && (value < 100))
    {
        print_decimal((x_pos + 18), y_pos, num_reg_state, tmp, points);
    }
    else if(value < 10)
    {
        print_decimal((x_pos + 12), y_pos, num_reg_state, tmp, points);
        if((value) < 0)
        {
            print_char(x_pos, y_pos, CHR, '-');
        }
        else
        {
            print_char(x_pos, y_pos, CHR, ' ');
        }
    }
}


void draw_pixel(unsigned char x_pos, unsigned char y_pos, unsigned char colour)
{
    unsigned char value = 0x00;
    unsigned char page = 0x00;
    unsigned char bit_pos = 0x00;

    page = (y_pos / y_max);
    bit_pos = (y_pos - (page * y_max));
    value = buffer[((page * x_max) + x_pos)];

    if((colour & YES) != NO)
    {
        value |= (1 << bit_pos);
    }
    else
    {
        value &= (~(1 << bit_pos));
    }

    buffer[((page * x_max) + x_pos)] = value;
    OLED_gotoxy(x_pos, page);
    OLED_write(value, DAT);
}
```

```
void draw_line(signed int x1, signed int y1, signed int x2, signed int y2, unsigned char co
lour)
{
    signed int dx = 0x0000;
    signed int dy = 0x0000;
    signed int stepx = 0x0000;
    signed int stepy = 0x0000;
    signed int fraction = 0x0000;

    dy = (y2 - y1);
    dx = (x2 - x1);

    if (dy < 0)
    {
        dy = -dy;
        stepy = -1;
    }
    else
    {
        stepy = 1;
    }

    if (dx < 0)
    {
        dx = -dx;
        stepx = -1;
    }
    else
    {
        stepx = 1;
    }

    dx <<= 1;
    dy <<= 1;

    draw_pixel(x1, y1, colour);

    if(dx > dy)
    {
        fraction = (dy - (dx >> 1));
        while (x1 != x2)
        {
            if(fraction >= 0)
            {
                y1 += stepy;
                fraction -= dx;
            }

            x1 += stepx;
            fraction += dy;

            draw_pixel(x1, y1, colour);
        }
    }
    else
    {
        fraction = (dx - (dy >> 1));
        while (y1 != y2)
        {
            if (fraction >= 0)
            {
                x1 += stepx;
                fraction -= dy;
            }

            y1 += stepy;
```

```
            fraction += dx;

            draw_pixel(x1, y1, colour);
        }
    }
}


void draw_V_line(signed int x1, signed int y1, signed int y2, unsigned colour)
{
    if(y1 > y2)
    {
       swap(&y1, &y2);
    }

    while(y2 > (y1 - 1))
    {
        draw_pixel(x1, y2, colour);
        y2--;
    }
}


void draw_H_line(signed int x1, signed int x2, signed int y1, unsigned colour)
{
    if(x1 > x2)
    {
       swap(&x1, &x2);
    }

    while(x2 > (x1 - 1))
    {
        draw_pixel(x2, y1, colour);
        x2--;
    }
}


void draw_triangle(signed int x1, signed int y1, signed int x2, signed int y2, signed int x
3, signed int y3, unsigned char fill, unsigned int colour)
{
    signed int a = 0;
    signed int b = 0;
    signed int sa = 0;
    signed int sb = 0;
    signed int yp = 0;
    signed int last = 0;
    signed int dx12 = 0;
    signed int dx23 = 0;
    signed int dx13 = 0;
    signed int dy12 = 0;
    signed int dy23 = 0;
    signed int dy13 = 0;

    switch(fill)
    {
        case YES:
        {
            if(y1 > y2)
            {
                swap(&y1, &y2);
                swap(&x1, &x2);
            }
            if(y2 > y3)
            {
```

```
        swap(&y3, &y2);
        swap(&x3, &x2);
    }
    if(y1 > y2)
    {
        swap(&y1, &y2);
        swap(&x1, &x2);
    }

    if(y1 == y3)
    {
        a = b = x1;

        if(x2 < a)
        {
            a = x2;
        }
        else if(x2 > b)
        {
            b = x2;
        }
        if(x2 < a)
        {
            a = x3;
        }
        else if(x3 > b)
        {
            b = x3;
        }

        draw_H_line(a, (a + (b - (a + 1))), y1, colour);
        return;
    }

    dx12 = (x2 - x1);
    dy12 = (y2 - y1);
    dx13 = (x3 - x1);
    dy13 = (y3 - y1);
    dx23 = (x3 - x2);
    dy23 = (y3 - y2);
    sa = 0,
    sb = 0;

    if(y2 == y3)
    {
        last = y2;
    }
    else
    {
        last = (y2 - 1);
    }

    for(yp = y1; yp <= last; yp++)
    {
        a = (x1 + (sa / dy12));
        b = (x1 + (sb / dy13));
        sa += dx12;
        sb += dx13;
        if(a > b)
        {
            swap(&a, &b);
        }
        draw_H_line(a, (a + (b - (a + 1))), yp, colour);
    }

    sa = (dx23 * (yp - y2));
```

```
            sb = (dx13 * (yp - y1));
            for(; yp <= y3; yp++)
            {
                a = (x2 + (sa / dy23));
                b = (x1 + (sb / dy13));
                sa += dx23;
                sb += dx13;

                if(a > b)
                {
                    swap(&a, &b);
                }
                draw_H_line(a, (a + (b - (a + 1))), yp, colour);
            }


            break;
        }
        default:
        {
            draw_line(x1, y1, x2, y2, colour);
            draw_line(x2, y2, x3, y3, colour);
            draw_line(x3, y3, x1, y1, colour);
            break;
        }
    }
}


void draw_rectangle(signed int x1, signed int y1, signed int x2, signed int y2, unsigned ch
ar fill, unsigned char colour, unsigned char type)
{
     unsigned short i = 0x00;
     unsigned short xmin = 0x00;
     unsigned short xmax = 0x00;
     unsigned short ymin = 0x00;
     unsigned short ymax = 0x00;

     if(fill != NO)
     {
        if(x1 < x2)
        {
           xmin = x1;
           xmax = x2;
        }
        else
        {
           xmin = x2;
           xmax = x1;
        }

        if(y1 < y2)
        {
           ymin = y1;
           ymax = y2;
        }
        else
        {
           ymin = y2;
           ymax = y1;
        }

        for(; xmin <= xmax; ++xmin)
        {
           for(i = ymin; i <= ymax; ++i)
```

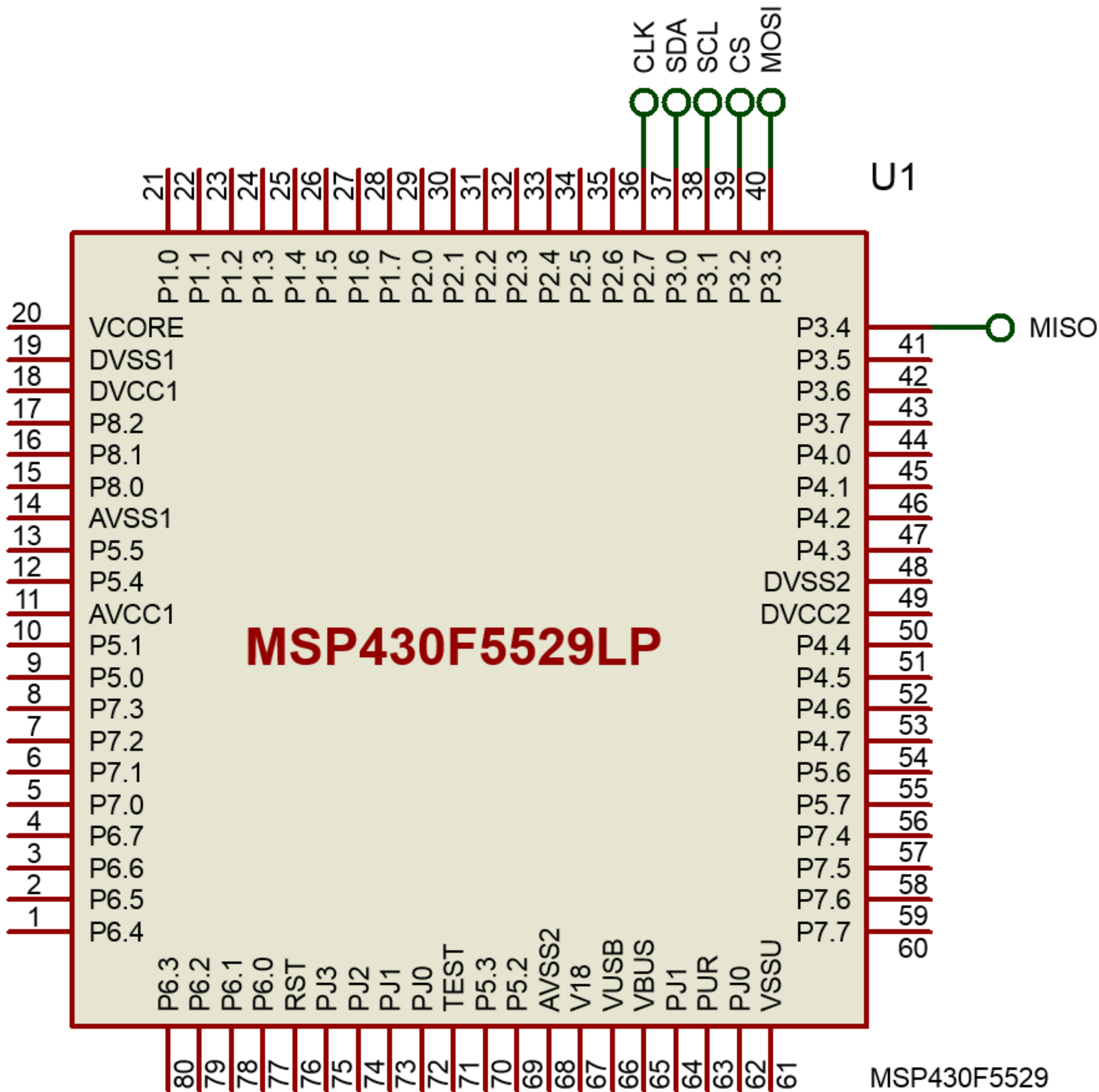
```
           {
               draw_pixel(xmin, i, colour);
           }
         }
     }

     else
     {
        draw_line(x1, y1, x2, y1, colour);
        draw_line(x1, y2, x2, y2, colour);
        draw_line(x1, y1, x1, y2, colour);
        draw_line(x2, y1, x2, y2, colour);
     }

     if(type != SQUARE)
     {
         draw_pixel(x1, y1, ~colour);
         draw_pixel(x1, y2, ~colour);
         draw_pixel(x2, y1, ~colour);
         draw_pixel(x2, y2, ~colour);
     }
}


void draw_circle(signed int xc, signed int yc, signed int radius, unsigned char fill, unsig
ned char colour)
{
   signed int a = 0x0000;
   signed int b = 0x0000;
   signed int P = 0x0000;

   b = radius;
   P = (1 - b);

   do
   {
        if(fill != NO)
        {
           draw_line((xc - a), (yc + b), (xc + a), (yc + b), colour);
           draw_line((xc - a), (yc - b), (xc + a), (yc - b), colour);
           draw_line((xc - b), (yc + a), (xc + b), (yc + a), colour);
           draw_line((xc - b), (yc - a), (xc + b), (yc - a), colour);
        }
        else
        {
           draw_pixel((xc + a), (yc + b), colour);
           draw_pixel((xc + b), (yc + a), colour);
           draw_pixel((xc - a), (yc + b), colour);
           draw_pixel((xc - b), (yc + a), colour);
           draw_pixel((xc + b), (yc - a), colour);
           draw_pixel((xc + a), (yc - b), colour);
           draw_pixel((xc - a), (yc - b), colour);
           draw_pixel((xc - b), (yc - a), colour);
        }

        if(P < 0)
        {
           P += (3 + (2 * a++));
        }
        else
        {
           P += (5 + (2 * ((a++) - (b--))));
        }
    }while(a <= b);
}
```

main.c

```
#include "driverlib.h"
#include "delay.h"
#include "SSD1306.h"
#include "MAX6675.h"


extern unsigned char buffer[buffer_size];


void clock_init(void);


void main(void)
{
    float t = 0;
    unsigned int ti = 0;

    WDT_A_hold(WDT_A_BASE);

    clock_init();
    OLED_init();
    MAX6675_init();

    while(1)
    {
        MAX6675_get_ADC(&ti);
        t = MAX6675_get_T(ti, deg_C);

        print_float(40, 10, NUM, t, 1);

        delay_ms(400);
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_3,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF, UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ, MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);
```

```
    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}
```

Hardware Setup



Explanation

I won't be going through hardware initialization and other common stuffs that have been covered earlier. The focus here is toward the SPI read operation.

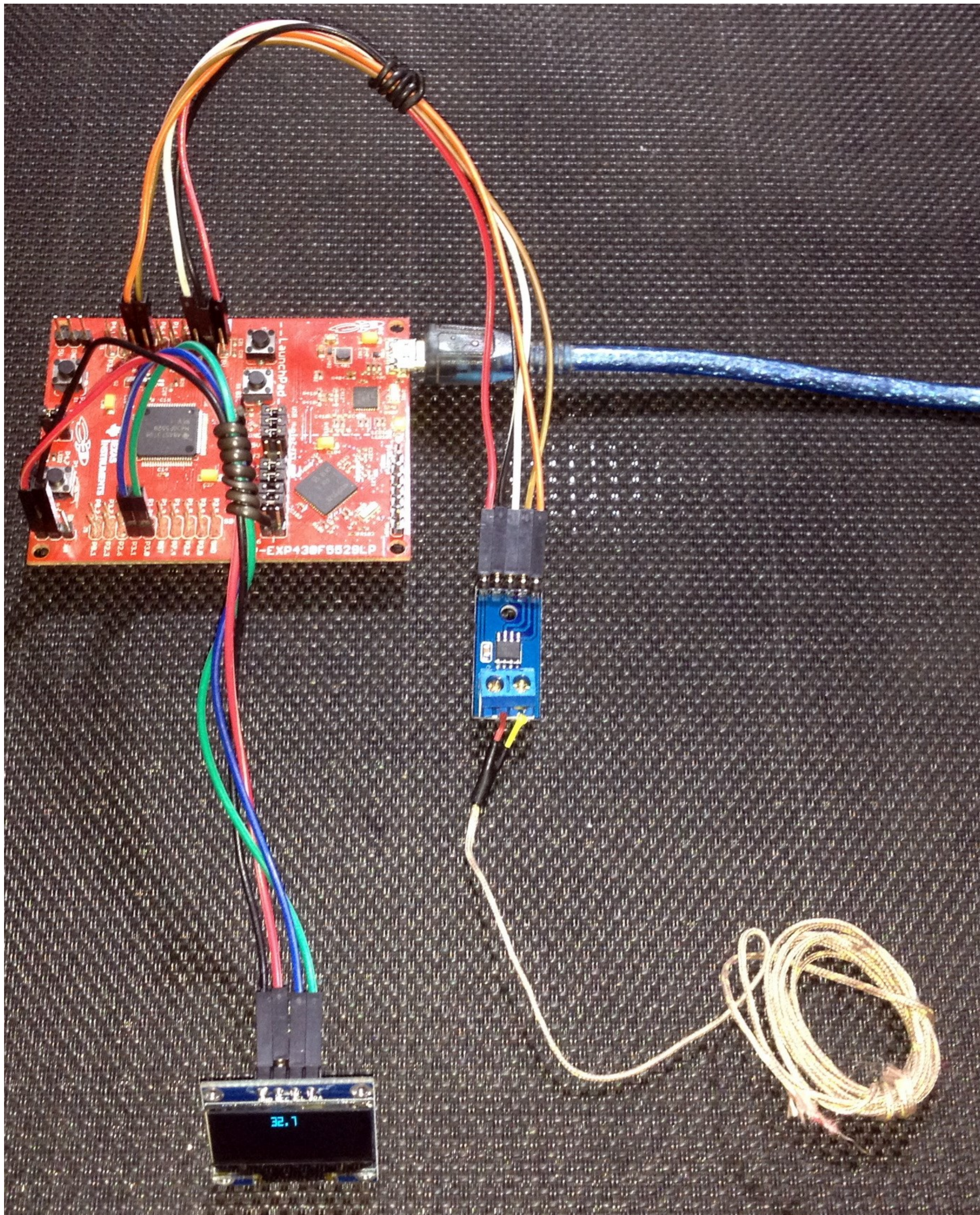
In the *MAX6675_get_ADC* function, the SPI read operation can be seen.

```
CS_pin_low();

USCI_A_SPI_transmitData(USCI_A0_BASE, 0x00);
while(USCI_A_SPI_isBusy(USCI_A0_BASE));
hb = USCI_A_SPI_receiveData(USCI_A0_BASE);
while(USCI_A_SPI_isBusy(USCI_A0_BASE));

USCI_B_SPI_transmitData(USCI_A0_BASE, 0x00);
while(USCI_A_SPI_isBusy(USCI_A0_BASE));
lb = USCI_A_SPI_receiveData(USCI_A0_BASE);
while(USCI_A_SPI_isBusy(USCI_A0_BASE));

CS_pin_high();
```

Although no SPI write operation is possible for MAX6675 due absence of slave-in pin in it, 0s are written prior to every SPI bus read. It is not mandatory write 0s and we can use any other value since it has no real use other than SPI clock generation. This is so because SPI bus is like a ring buffer. In SPI bus, data is simultaneously read and written with SPI clock pulses. After every complete transfer, the SPI bus is polled to check if it is free.

The demo here reads temperature with a thermocouple connected to a MAX6675 and displays the read temperature on a SSD1306 OLED display every 400 ms.

Demo



Demo video: https://youtu.be/jsL-HjWkYGA

USCI – SPI – I2C Example 3

Two examples SPI-I2C would have been enough but went for this final third one because sometimes it is difficult for grasping I2C/SPI operations with driverlib alone. This is especially the case when people are migrating from low-end or mid-range MSP430s that have no driverlib support and this is what happened with me at the beginning. It was then I decided what if I can use the good old I2C and SPI libraries that TI supplied for their low and mid-range devices. This example demonstrates exactly that. Here both the SPI and I2C hardware of MSP430F5529 were tested against the older libraries. These libraries do not apply the use of driverlib APIs except for pin declarations. I would also like readers to check my past SPI and I2C tutorials on MSP430G2xxx series for better reference and understanding. Here's the link: http://embedded-lab.com/blog/more-on-ti-msp430s/ for those tutorials.

Note that unlike MSP430G2xxx devices, MSP430F5529 doesn't have any USI hardware and instead of it we will have to stick to USCI modules - USCI_B modules for I2C and USCI_A or USCI_B modules for SPI communications.

If still things are still messy with USCI, we can use software-based I2C/SPI libraries but those won't perform like the hardware-based USCI ones. In terms of communication speed, software-based I2C/SPI are slow and in terms of coding, software-based solutions are resource-consuming. Thus, it is best to learn and use USCI modules.

Here in this demo, an I2C-based BH1750 light sensor is read and its readings are displayed on a SPI-based Nokia GLCD (PCD8544).

Code Example

USCI_I2C.h

```
#include <msp430.h>
#include "driverlib.h"


#define USCI_B0_I2C_port                                          GPIO_PORT_P3

#define USCI_B0_I2C_SDA_pin                                       GPIO_PIN0
#define USCI_B0_I2C_SCL_pin                                       GPIO_PIN1

#define USCI_B1_I2C_port                                          GPIO_PORT_P4

#define USCI_B1_I2C_SDA_pin                                       GPIO_PIN1
#define USCI_B1_I2C_SCL_pin                                       GPIO_PIN2


void I2C_USCI_B0_init(unsigned char address);
void I2C_USCI_B0_set_address(unsigned char address);
unsigned char I2C_USCI_B0_read_byte(unsigned char address);
unsigned char I2C_USCI_B0_read_word(unsigned char address,unsigned char *value, unsigned ch
ar length);
unsigned char I2C_USCI_B0_write_byte(unsigned char address, unsigned char value);

void I2C_USCI_B1_init(unsigned char address);
void I2C_USCI_B1_set_address(unsigned char address);
unsigned char I2C_USCI_B1_read_byte(unsigned char address);
```

```
unsigned char I2C_USCI_B1_read_word(unsigned char address,unsigned char *value, unsigned ch
ar length);
unsigned char I2C_USCI_B1_write_byte(unsigned char address, unsigned char value);
```

USCI_I2C.c

```
#include "USCI_I2C.h"


void I2C_USCI_B0_init(unsigned char address)
{
    GPIO_setAsPeripheralModuleFunctionOutputPin(USCI_B0_I2C_port,
                                                (USCI_B0_I2C_SDA_pin
                                                | USCI_B0_I2C_SCL_pin));

    UCB0CTL1 |= UCSWRST;
    UCB0CTL0 = (UCMST | UCMODE_3 | UCSYNC);
    UCB0CTL1 = (UCSSEL_2 | UCSWRST);
    UCB0BR0 = 50;
    UCB0I2CSA = address;
    UCB0CTL1 &= ~UCSWRST;
}


void I2C_USCI_B0_set_address(unsigned char address)
{
    UCB0CTL1 |= UCSWRST;
    UCB0I2CSA = address;
    UCB0CTL1 &= ~UCSWRST;
}


unsigned char I2C_USCI_B0_read_byte(unsigned char address)
{
    while(UCB0CTL1 & UCTXSTP);
    UCB0CTL1 |= (UCTR | UCTXSTT);

    while(!(UCB0IFG & UCTXIFG));
    UCB0TXBUF = address;

    while(!(UCB0IFG & UCTXIFG));
    UCB0CTL1 &= ~UCTR;
    UCB0CTL1 |= UCTXSTT;
    UCB0IFG &= ~UCTXIFG;

    while(UCB0CTL1 & UCTXSTT);
    UCB0CTL1 |= UCTXSTP;

    return UCB0RXBUF;
}


unsigned char I2C_USCI_B0_read_word(unsigned char address,unsigned char *value, unsigned ch
ar length)
{
    unsigned char i = 0;

    while(UCB0CTL1 & UCTXSTP);

    UCB0CTL1 |= (UCTR | UCTXSTT);

    while(!(UCB0IFG & UCTXIFG));

    UCB0IFG &= ~UCTXIFG;
```

```
    if(UCB0IFG & UCNACKIFG)
    {
        return UCB0STAT;
    }

    UCB0TXBUF = address;

    while(!(UCB0IFG & UCTXIFG));

    if(UCB0IFG & UCNACKIFG)
    {
        return UCB0STAT;
    }

    UCB0CTL1 &= ~UCTR;
    UCB0CTL1 |= UCTXSTT;
    UCB0IFG &= ~UCTXIFG;

    while(UCB0CTL1 & UCTXSTT);

    do
    {
        while (!(UCB0IFG & UCRXIFG));
        UCB0IFG &= ~UCTXIFG;
        value[i] = UCB0RXBUF;
        i++;
    }while(i < (length - 1));

    while (!(UCB0IFG & UCRXIFG));

    UCB0IFG &= ~UCTXIFG;
    UCB0CTL1 |= UCTXSTP;
    value[length - 1] = UCB0RXBUF;
    UCB0IFG &= ~UCTXIFG;

    return 0;
}


unsigned char I2C_USCI_B0_write_byte(unsigned char address, unsigned char value)
{
    while(UCB0CTL1 & UCTXSTP);

    UCB0CTL1 |= (UCTR | UCTXSTT);

    while(!(UCB0IFG & UCTXIFG));

    if(UCB0IFG & UCNACKIFG)
    {
        return UCB0STAT;
    }

    UCB0TXBUF = address;

    while(!(UCB0IFG & UCTXIFG));

    if(UCB0IFG & UCNACKIFG)
    {
        return UCB0STAT;
    }

    UCB0TXBUF = value;

    while(!(UCB0IFG & UCTXIFG));
```

```
    if(UCB0IFG & UCNACKIFG)
    {
        return UCB0STAT;
    }

    UCB0CTL1 |= UCTXSTP;
    UCB0IFG &= ~UCTXIFG;

    return 0;
}


void I2C_USCI_B1_init(unsigned char address)
{
    GPIO_setAsPeripheralModuleFunctionOutputPin(USCI_B1_I2C_port,
                                                (USCI_B1_I2C_SDA_pin
                                                | USCI_B1_I2C_SCL_pin));

    UCB1CTL1 |= UCSWRST;
    UCB1CTL0 = (UCMST | UCMODE_3 | UCSYNC);
    UCB1CTL1 = (UCSSEL_2 | UCSWRST);
    UCB1BR0 = 10;
    UCB1I2CSA = address;
    UCB1CTL1 &= ~UCSWRST;
}


void I2C_USCI_B1_set_address(unsigned char address)
{
    UCB1CTL1 |= UCSWRST;
    UCB1I2CSA = address;
    UCB1CTL1 &= ~UCSWRST;
}


unsigned char I2C_USCI_B1_read_byte(unsigned char address)
{
    while(UCB1CTL1 & UCTXSTP);
    UCB1CTL1 |= (UCTR | UCTXSTT);

    while(!(UCB1IFG & UCTXIFG));
    UCB1TXBUF = address;

    while(!(UCB1IFG & UCTXIFG));
    UCB1CTL1 &= ~UCTR;
    UCB1CTL1 |= UCTXSTT;
    UCB1IFG &= ~UCTXIFG;

    while(UCB1CTL1 & UCTXSTT);
    UCB1CTL1 |= UCTXSTP;

    return UCB1RXBUF;
}


unsigned char I2C_USCI_B1_read_word(unsigned char address,unsigned char *value, unsigned ch
ar length)
{
    unsigned char i = 0;

    while(UCB1CTL1 & UCTXSTP);

    UCB1CTL1 |= (UCTR | UCTXSTT);

    while(!(UCB1IFG & UCTXIFG));
```

```
    UCB1IFG &= ~UCTXIFG;

    if(UCB1IFG & UCNACKIFG)
    {
        return UCB1STAT;
    }

    UCB1TXBUF = address;

    while(!(UCB1IFG & UCTXIFG));

    if(UCB1IFG & UCNACKIFG)
    {
        return UCB1STAT;
    }

    UCB1CTL1 &= ~UCTR;
    UCB1CTL1 |= UCTXSTT;
    UCB1IFG &= ~UCTXIFG;

    while(UCB1CTL1 & UCTXSTT);

    do
    {
        while (!(UCB1IFG & UCRXIFG));
        UCB1IFG &= ~UCTXIFG;
        value[i] = UCB1RXBUF;
        i++;
    }while(i < (length - 1));

    while (!(UCB1IFG & UCRXIFG));

    UCB1IFG &= ~UCTXIFG;
    UCB1CTL1 |= UCTXSTP;
    value[length - 1] = UCB1RXBUF;
    UCB1IFG &= ~UCTXIFG;

    return 0;
}


unsigned char I2C_USCI_B1_write_byte(unsigned char address, unsigned char value)
{
    while(UCB1CTL1 & UCTXSTP);

    UCB1CTL1 |= (UCTR | UCTXSTT);

    while(!(UCB1IFG & UCTXIFG));

    if(UCB1IFG & UCNACKIFG)
    {
        return UCB1STAT;
    }

    UCB1TXBUF = address;

    while(!(UCB1IFG & UCTXIFG));

    if(UCB1IFG & UCNACKIFG)
    {
        return UCB1STAT;
    }

    UCB1TXBUF = value;
```

```
    while(!(UCB1IFG & UCTXIFG));

    if(UCB1IFG & UCNACKIFG)
    {
        return UCB1STAT;
    }

    UCB1CTL1 |= UCTXSTP;
    UCB1IFG &= ~UCTXIFG;

    return 0;
}
```

BH1750.h

```
#include "driverlib.h"
#include "delay.h"
#include "USCI_I2C.h"


#define  BH1750_addr                        0x23

#define  power_down                         0x00
#define  power_up                           0x01
#define  reset                              0x07
#define  cont_H_res_mode1                   0x10
#define  cont_H_res_mode2                   0x11
#define  cont_L_res_mode                    0x13
#define  one_time_H_res_mode1               0x20
#define  one_time_H_res_mode2               0x21
#define  one_time_L_res_mode                0x23


void BH1750_init(void);
void BH1750_write(unsigned char cmd);
unsigned int BH1750_read_word(void);
unsigned int get_lux_value(unsigned char mode, unsigned int delay_time);
```

BH1750.c

```
#include "BH1750.h"


void BH1750_init(void)
{
   I2C_USCI_B0_init(BH1750_addr);
   delay_ms(10);
   BH1750_write(power_down);
}


void BH1750_write(unsigned char cmd)
{
    I2C_USCI_B0_write_byte(BH1750_addr, cmd);
}


unsigned int BH1750_read_word(void)
{
  unsigned long value = 0x0000;
  unsigned char bytes[2] = {0x00, 0x00};

  I2C_USCI_B0_read_word(0x11, bytes, 2);
```

```
  value = ((bytes[1] << 8) | bytes[0]);

  return value;
}


unsigned int get_lux_value(unsigned char mode, unsigned int delay_time)
{
  unsigned long lux_value = 0x00;
  unsigned char dly = 0x00;
  unsigned char s = 0x08;

  while(s)
  {
     BH1750_write(power_up);
     BH1750_write(mode);

     lux_value += BH1750_read_word();

     for(dly = 0; dly < delay_time; dly += 1)
     {
         delay_ms(1);
     }

     BH1750_write(power_down);

     s--;
  }

  lux_value >>= 3;

  return ((unsigned int)lux_value);
}
```

USCI_SPI.h

```
#include <msp430.h>
#include "driverlib.h"


#define USCI_A0_MOSI_port                    GPIO_PORT_P3
#define USCI_A0_MISO_port                    GPIO_PORT_P3
#define USCI_A0_CLK_port                     GPIO_PORT_P2

#define USCI_A0_MOSI_pin                     GPIO_PIN3
#define USCI_A0_MISO_pin                     GPIO_PIN4
#define USCI_A0_CLK_pin                      GPIO_PIN7

#define USCI_A1_MOSI_port                    GPIO_PORT_P4
#define USCI_A1_MISO_port                    GPIO_PORT_P4
#define USCI_A1_CLK_port                     GPIO_PORT_P4

#define USCI_A1_MOSI_pin                     GPIO_PIN4
#define USCI_A1_MISO_pin                     GPIO_PIN5
#define USCI_A1_CLK_pin                      GPIO_PIN0

#define USCI_B0_MOSI_port                    GPIO_PORT_P3
#define USCI_B0_MISO_port                    GPIO_PORT_P3
#define USCI_B0_CLK_port                     GPIO_PORT_P3

#define USCI_B0_MOSI_pin                     GPIO_PIN0
#define USCI_B0_MISO_pin                     GPIO_PIN1
#define USCI_B0_CLK_pin                      GPIO_PIN2
```

```
#define USCI_B1_MOSI_port                   GPIO_PORT_P4
#define USCI_B1_MISO_port                   GPIO_PORT_P4
#define USCI_B1_CLK_port                    GPIO_PORT_P4

#define USCI_B1_MOSI_pin                    GPIO_PIN1
#define USCI_B1_MISO_pin                    GPIO_PIN2
#define USCI_B1_CLK_pin                     GPIO_PIN3


void SPI_USCI_A0_init(void);
void SPI_USCI_A0_write(unsigned char tx_data);
unsigned char SPI_USCI_A0_read(void);
unsigned char SPI_USCI_A0_transfer(unsigned char tx_data);

void SPI_USCI_A1_init(void);
void SPI_USCI_A1_write(unsigned char tx_data);
unsigned char SPI_USCI_A1_read(void);
unsigned char SPI_USCI_A1_transfer(unsigned char tx_data);

void SPI_USCI_B0_init(void);
void SPI_USCI_B0_write(unsigned char tx_data);
unsigned char SPI_USCI_B0_read(void);
unsigned char SPI_USCI_B0_transfer(unsigned char tx_data);

void SPI_USCI_B1_init(void);
void SPI_USCI_B1_write(unsigned char tx_data);
unsigned char SPI_USCI_B1_read(void);
unsigned char SPI_USCI_B1_transfer(unsigned char tx_data);
```

USCI_SPI.c

```
#include "USCI_SPI.h"


void SPI_USCI_A0_init(void)
{
    GPIO_setAsPeripheralModuleFunctionInputPin(USCI_A0_MISO_port,
                                               USCI_A0_MISO_pin);

    GPIO_setAsPeripheralModuleFunctionOutputPin(USCI_A0_MOSI_port,
                                                USCI_A0_MOSI_pin);

    GPIO_setAsPeripheralModuleFunctionOutputPin(USCI_A0_CLK_port,
                                                USCI_A0_CLK_pin);

    UCA0CTL1 |= UCSWRST;
    UCA0CTL0 = UCCKPH | UCMSB | UCMST | UCMODE_1 | UCSYNC;
    UCA0CTL1 = UCSSEL_2;
    UCA0BR0 = 4;
    UCA0BR1 = 0;
    UCA0CTL1 &= ~UCSWRST;
}


void SPI_USCI_A0_write(unsigned char tx_data)
{
    while(!(UCA0IFG & UCTXIFG));
    UCA0TXBUF = tx_data;
    while(UCA0STAT & UCBUSY);
}


unsigned char SPI_USCI_A0_read(void)
```

```
{
    unsigned char rx_data = 0;

    while(!(UCA0IFG & UCRXIFG));
    rx_data = UCA0RXBUF;
    while(UCA0STAT & UCBUSY);

    return rx_data;
}


unsigned char SPI_USCI_A0_transfer(unsigned char tx_data)
{
    unsigned char rx_data = 0;

    while(!(UCA0IFG & UCTXIFG));
    UCA0TXBUF = tx_data;
    while(UCA0STAT & UCBUSY);

    while(!(UCA0IFG & UCRXIFG));
    rx_data = UCA0RXBUF;
    while(UCA0STAT & UCBUSY);

    return rx_data;
}


void SPI_USCI_A1_init(void)
{
    GPIO_setAsPeripheralModuleFunctionInputPin(USCI_A1_MISO_port,
                                               USCI_A1_MISO_pin);

    GPIO_setAsPeripheralModuleFunctionOutputPin(USCI_A1_MOSI_port,
                                                USCI_A1_MOSI_pin);

    GPIO_setAsPeripheralModuleFunctionOutputPin(USCI_A1_CLK_port,
                                                USCI_A1_CLK_pin);

    UCA1CTL1 |= UCSWRST;
    UCA1CTL0 = UCCKPH | UCMSB | UCMST | UCMODE_1 | UCSYNC;
    UCA1CTL1 = UCSSEL_2;
    UCA1BR0 = 8;
    UCA1BR1 = 0;
    UCA1CTL1 &= ~UCSWRST;
}


void SPI_USCI_A1_write(unsigned char tx_data)
{
    while(!(UCA1IFG & UCTXIFG));
    UCA1TXBUF = tx_data;
    while(UCA1STAT & UCBUSY);
}


unsigned char SPI_USCI_A1_read(void)
{
    unsigned char rx_data = 0;

    while(!(UCA1IFG & UCRXIFG));
    rx_data = UCA1RXBUF;
    while(UCA1STAT & UCBUSY);

    return rx_data;
}
```

```
unsigned char SPI_USCI_A1_transfer(unsigned char tx_data)
{
    unsigned char rx_data = 0;

    while(!(UCA1IFG & UCTXIFG));
    UCA1TXBUF = tx_data;
    while(UCA1STAT & UCBUSY);

    while(!(UCA1IFG & UCRXIFG));
    rx_data = UCA1RXBUF;
    while(UCA1STAT & UCBUSY);

    return rx_data;
}


void SPI_USCI_B0_init(void)
{
    GPIO_setAsPeripheralModuleFunctionInputPin(USCI_B0_MISO_port,
                                               USCI_B0_MISO_pin);

    GPIO_setAsPeripheralModuleFunctionOutputPin(USCI_B0_MOSI_port,
                                                USCI_B0_MOSI_pin);

    GPIO_setAsPeripheralModuleFunctionOutputPin(USCI_B0_CLK_port,
                                                USCI_B0_CLK_pin);

    UCB0CTL1 |= UCSWRST;
    UCB0CTL0 = UCCKPH | UCMSB | UCMST | UCMODE_1 | UCSYNC;
    UCB0CTL1 = UCSSEL_2;
    UCB0BR0 = 8;
    UCB0BR1 = 0;
    UCB0CTL1 &= ~UCSWRST;
}


void SPI_USCI_B0_write(unsigned char tx_data)
{
    while(!(UCB0IFG & UCTXIFG));
    UCB0TXBUF = tx_data;
    while(UCB0STAT & UCBUSY);
}


unsigned char SPI_USCI_B0_read(void)
{
    unsigned char rx_data = 0;

    while(!(UCB0IFG & UCRXIFG));
    rx_data = UCB0RXBUF;
    while(UCB0STAT & UCBUSY);

    return rx_data;
}


unsigned char SPI_USCI_B0_transfer(unsigned char tx_data)
{
    unsigned char rx_data = 0;

    while(!(UCB0IFG & UCTXIFG));
    UCB0TXBUF = tx_data;
    while(UCB0STAT & UCBUSY);
```

```
    while(!(UCB0IFG & UCRXIFG));
    rx_data = UCB0RXBUF;
    while(UCB0STAT & UCBUSY);

    return rx_data;
}


void SPI_USCI_B1_init(void)
{
    GPIO_setAsPeripheralModuleFunctionInputPin(USCI_B1_MISO_port,
                                               USCI_B1_MISO_pin);

    GPIO_setAsPeripheralModuleFunctionOutputPin(USCI_B1_MOSI_port,
                                                USCI_B1_MOSI_pin);

    GPIO_setAsPeripheralModuleFunctionOutputPin(USCI_B1_CLK_port,
                                                USCI_B1_CLK_pin);

    UCB1CTL1 |= UCSWRST;
    UCB1CTL0 = UCCKPH | UCMSB | UCMST | UCMODE_1 | UCSYNC;
    UCB1CTL1 = UCSSEL_2;
    UCB1BR0 = 8;
    UCB1BR1 = 0;
    UCB1CTL1 &= ~UCSWRST;
}


void SPI_USCI_B1_write(unsigned char tx_data)
{
    while(!(UCB1IFG & UCTXIFG));
    UCB1TXBUF = tx_data;
    while(UCB1STAT & UCBUSY);
}


unsigned char SPI_USCI_B1_read(void)
{
    unsigned char rx_data = 0;

    while(!(UCB1IFG & UCRXIFG));
    rx_data = UCB1RXBUF;
    while(UCB1STAT & UCBUSY);

    return rx_data;
}


unsigned char SPI_USCI_B1_transfer(unsigned char tx_data)
{
    unsigned char rx_data = 0;

    while(!(UCB1IFG & UCTXIFG));
    UCB1TXBUF = tx_data;
    while(UCB1STAT & UCBUSY);

    while(!(UCB1IFG & UCRXIFG));
    rx_data = UCB1RXBUF;
    while(UCB1STAT & UCBUSY);

    return rx_data;
}
```

PCD8544.h

```
#include "driverlib.h"
#include "delay.h"
#include "USCI_SPI.h"


#define RST_port                    GPIO_PORT_P6
#define CS_port                     GPIO_PORT_P6
#define DC_port                     GPIO_PORT_P6

#define RST_pin                     GPIO_PIN0
#define CS_pin                      GPIO_PIN1
#define DC_pin                      GPIO_PIN2

#define RST_pin_high()              GPIO_setOutputHighOnPin(RST_port, RST_pin)
#define RST_pin_low()               GPIO_setOutputLowOnPin(RST_port, RST_pin)

#define CS_pin_high()               GPIO_setOutputHighOnPin(CS_port, CS_pin)
#define CS_pin_low()                GPIO_setOutputLowOnPin(CS_port, CS_pin)

#define DC_pin_high()               GPIO_setOutputHighOnPin(DC_port, DC_pin)
#define DC_pin_low()                GPIO_setOutputLowOnPin(DC_port, DC_pin)

#define PCD8544_set_Y_addr            0x40
#define PCD8544_set_X_addr            0x80

#define PCD8544_set_temp              0x04
#define PCD8544_set_bias              0x10
#define PCD8544_set_VOP             0x80

#define PCD8544_power_down          0x04
#define PCD8544_entry_mode          0x02
#define PCD8544_extended_instruction 0x01

#define PCD8544_display_blank       0x00
#define PCD8544_display_normal      0x04
#define PCD8544_display_all_on      0x01
#define PCD8544_display_inverted    0x05

#define PCD8544_function_set        0x20
#define PCD8544_display_control     0x08

#define CMD                         0
#define DAT                         1

#define X_max                       84
#define Y_max                       48
#define Rows                        6

#define BLACK                       0
#define WHITE                       1
#define PIXEL_XOR                   2

#define ON                          1
#define OFF                         0

#define NO                          0
#define YES                         1

#define buffer_size                  504


void PCD8544_init(void);
void PCD8544_reset(void);
```

```
void PCD8544_write(unsigned char type, unsigned char value);
void PCD8544_set_contrast(unsigned char value);
void PCD8544_set_cursor(unsigned char x_pos, unsigned char y_pos);
void PCD8544_print_char(unsigned char ch, unsigned char colour);
void PCD8544_print_custom_char(unsigned char *map);
void PCD8544_fill(unsigned char bufr);
void PCD8544_clear_buffer(unsigned char colour);
void PCD8544_clear_screen(unsigned char colour);
void print_image(const unsigned char *bmp);
void print_char(unsigned char x_pos, unsigned char y_pos, unsigned char ch, unsigned char c
olour);
void print_string(unsigned char x_pos, unsigned char y_pos, unsigned char *ch, unsigned cha
r colour);
void print_chr(unsigned char x_pos, unsigned char y_pos, signed int value, unsigned char co
lour);
void print_int(unsigned char x_pos, unsigned char y_pos, signed long value, unsigned char c
olour);
void print_decimal(unsigned char x_pos, unsigned char y_pos, unsigned int value, unsigned c
har points, unsigned char colour);
void print_float(unsigned char x_pos, unsigned char y_pos, float value, unsigned char point
s, unsigned char colour);
void Draw_Pixel(unsigned char x_pos, unsigned char y_pos, unsigned char colour);
void Draw_Line(signed int x1, signed int y1, signed int x2, signed int y2, unsigned char co
lour);
void Draw_Rectangle(signed int x1, signed int y1, signed int x2, signed int y2, unsigned ch
ar fill, unsigned char colour);
void Draw_Circle(signed int xc, signed int yc, signed int radius, unsigned char fill, unsig
ned char colour);
```

PCD8544.c

```
#include "PCD8544.h"


static unsigned char PCD8544_buffer[X_max][Rows];


static const unsigned char font[96][5] =
{
     {0x00, 0x00, 0x00, 0x00, 0x00} // 20
    ,{0x00, 0x00, 0x5f, 0x00, 0x00} // 21 !
    ,{0x00, 0x07, 0x00, 0x07, 0x00} // 22 "
    ,{0x14, 0x7f, 0x14, 0x7f, 0x14} // 23 #
    ,{0x24, 0x2a, 0x7f, 0x2a, 0x12} // 24 $
    ,{0x23, 0x13, 0x08, 0x64, 0x62} // 25 %
    ,{0x36, 0x49, 0x55, 0x22, 0x50} // 26 &
    ,{0x00, 0x05, 0x03, 0x00, 0x00} // 27 '
    ,{0x00, 0x1c, 0x22, 0x41, 0x00} // 28 (
    ,{0x00, 0x41, 0x22, 0x1c, 0x00} // 29 )
    ,{0x14, 0x08, 0x3e, 0x08, 0x14} // 2a *
    ,{0x08, 0x08, 0x3e, 0x08, 0x08} // 2b +
    ,{0x00, 0x50, 0x30, 0x00, 0x00} // 2c ,
    ,{0x08, 0x08, 0x08, 0x08, 0x08} // 2d -
    ,{0x00, 0x60, 0x60, 0x00, 0x00} // 2e .
    ,{0x20, 0x10, 0x08, 0x04, 0x02} // 2f /
    ,{0x3e, 0x51, 0x49, 0x45, 0x3e} // 30 0
    ,{0x00, 0x42, 0x7f, 0x40, 0x00} // 31 1
    ,{0x42, 0x61, 0x51, 0x49, 0x46} // 32 2
    ,{0x21, 0x41, 0x45, 0x4b, 0x31} // 33 3
    ,{0x18, 0x14, 0x12, 0x7f, 0x10} // 34 4
    ,{0x27, 0x45, 0x45, 0x45, 0x39} // 35 5
    ,{0x3c, 0x4a, 0x49, 0x49, 0x30} // 36 6
    ,{0x01, 0x71, 0x09, 0x05, 0x03} // 37 7
    ,{0x36, 0x49, 0x49, 0x49, 0x36} // 38 8
```

```
    ,{0x06, 0x49, 0x49, 0x29, 0x1e} // 39 9
    ,{0x00, 0x36, 0x36, 0x00, 0x00} // 3a :
    ,{0x00, 0x56, 0x36, 0x00, 0x00} // 3b ;
    ,{0x08, 0x14, 0x22, 0x41, 0x00} // 3c <
    ,{0x14, 0x14, 0x14, 0x14, 0x14} // 3d =
    ,{0x00, 0x41, 0x22, 0x14, 0x08} // 3e >
    ,{0x02, 0x01, 0x51, 0x09, 0x06} // 3f ?
    ,{0x32, 0x49, 0x79, 0x41, 0x3e} // 40 @
    ,{0x7e, 0x11, 0x11, 0x11, 0x7e} // 41 A
    ,{0x7f, 0x49, 0x49, 0x49, 0x36} // 42 B
    ,{0x3e, 0x41, 0x41, 0x41, 0x22} // 43 C
    ,{0x7f, 0x41, 0x41, 0x22, 0x1c} // 44 D
    ,{0x7f, 0x49, 0x49, 0x49, 0x41} // 45 E
    ,{0x7f, 0x09, 0x09, 0x09, 0x01} // 46 F
    ,{0x3e, 0x41, 0x49, 0x49, 0x7a} // 47 G
    ,{0x7f, 0x08, 0x08, 0x08, 0x7f} // 48 H
    ,{0x00, 0x41, 0x7f, 0x41, 0x00} // 49 I
    ,{0x20, 0x40, 0x41, 0x3f, 0x01} // 4a J
    ,{0x7f, 0x08, 0x14, 0x22, 0x41} // 4b K
    ,{0x7f, 0x40, 0x40, 0x40, 0x40} // 4c L
    ,{0x7f, 0x02, 0x0c, 0x02, 0x7f} // 4d M
    ,{0x7f, 0x04, 0x08, 0x10, 0x7f} // 4e N
    ,{0x3e, 0x41, 0x41, 0x41, 0x3e} // 4f O
    ,{0x7f, 0x09, 0x09, 0x09, 0x06} // 50 P
    ,{0x3e, 0x41, 0x51, 0x21, 0x5e} // 51 Q
    ,{0x7f, 0x09, 0x19, 0x29, 0x46} // 52 R
    ,{0x46, 0x49, 0x49, 0x49, 0x31} // 53 S
    ,{0x01, 0x01, 0x7f, 0x01, 0x01} // 54 T
    ,{0x3f, 0x40, 0x40, 0x40, 0x3f} // 55 U
    ,{0x1f, 0x20, 0x40, 0x20, 0x1f} // 56 V
    ,{0x3f, 0x40, 0x38, 0x40, 0x3f} // 57 W
    ,{0x63, 0x14, 0x08, 0x14, 0x63} // 58 X
    ,{0x07, 0x08, 0x70, 0x08, 0x07} // 59 Y
    ,{0x61, 0x51, 0x49, 0x45, 0x43} // 5a Z
    ,{0x00, 0x7f, 0x41, 0x41, 0x00} // 5b [
    ,{0x02, 0x04, 0x08, 0x10, 0x20} // 5c ?
    ,{0x00, 0x41, 0x41, 0x7f, 0x00} // 5d ]
    ,{0x04, 0x02, 0x01, 0x02, 0x04} // 5e ^
    ,{0x40, 0x40, 0x40, 0x40, 0x40} // 5f _
    ,{0x00, 0x01, 0x02, 0x04, 0x00} // 60 `
    ,{0x20, 0x54, 0x54, 0x54, 0x78} // 61 a
    ,{0x7f, 0x48, 0x44, 0x44, 0x38} // 62 b
    ,{0x38, 0x44, 0x44, 0x44, 0x20} // 63 c
    ,{0x38, 0x44, 0x44, 0x48, 0x7f} // 64 d
    ,{0x38, 0x54, 0x54, 0x54, 0x18} // 65 e
    ,{0x08, 0x7e, 0x09, 0x01, 0x02} // 66 f
    ,{0x0c, 0x52, 0x52, 0x52, 0x3e} // 67 g
    ,{0x7f, 0x08, 0x04, 0x04, 0x78} // 68 h
    ,{0x00, 0x44, 0x7d, 0x40, 0x00} // 69 i
    ,{0x20, 0x40, 0x44, 0x3d, 0x00} // 6a j
    ,{0x7f, 0x10, 0x28, 0x44, 0x00} // 6b k
    ,{0x00, 0x41, 0x7f, 0x40, 0x00} // 6c l
    ,{0x7c, 0x04, 0x18, 0x04, 0x78} // 6d m
    ,{0x7c, 0x08, 0x04, 0x04, 0x78} // 6e n
    ,{0x38, 0x44, 0x44, 0x44, 0x38} // 6f o
    ,{0x7c, 0x14, 0x14, 0x14, 0x08} // 70 p
    ,{0x08, 0x14, 0x14, 0x18, 0x7c} // 71 q
    ,{0x7c, 0x08, 0x04, 0x04, 0x08} // 72 r
    ,{0x48, 0x54, 0x54, 0x54, 0x20} // 73 s
    ,{0x04, 0x3f, 0x44, 0x40, 0x20} // 74 t
    ,{0x3c, 0x40, 0x40, 0x20, 0x7c} // 75 u
    ,{0x1c, 0x20, 0x40, 0x20, 0x1c} // 76 v
    ,{0x3c, 0x40, 0x30, 0x40, 0x3c} // 77 w
    ,{0x44, 0x28, 0x10, 0x28, 0x44} // 78 x
    ,{0x0c, 0x50, 0x50, 0x50, 0x3c} // 79 y
    ,{0x44, 0x64, 0x54, 0x4c, 0x44} // 7a z
```

```
    ,{0x00, 0x08, 0x36, 0x41, 0x00} // 7b {
    ,{0x00, 0x00, 0x7f, 0x00, 0x00} // 7c |
    ,{0x00, 0x41, 0x36, 0x08, 0x00} // 7d }
    ,{0x10, 0x08, 0x08, 0x10, 0x08} // 7e ?
    ,{0x78, 0x46, 0x41, 0x46, 0x78} // 7f ?
};


void PCD8544_init()
{
    GPIO_setAsOutputPin(RST_port, RST_pin);
    GPIO_setDriveStrength(RST_port, RST_pin, GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(CS_port, CS_pin);
    GPIO_setDriveStrength(CS_port, CS_pin, GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(DC_port, DC_pin);
    GPIO_setDriveStrength(DC_port, DC_pin, GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    SPI_USCI_A0_init();

    PCD8544_reset();
    PCD8544_write(CMD, (PCD8544_extended_instruction | PCD8544_function_set));
    PCD8544_write(CMD, (PCD8544_set_bias | 0x02));
    PCD8544_set_contrast(0x39);
    PCD8544_write(CMD, PCD8544_set_temp);
    PCD8544_write(CMD, (PCD8544_display_normal | PCD8544_display_control));
    PCD8544_write(CMD, PCD8544_function_set);
    PCD8544_write(CMD, PCD8544_display_all_on);
    PCD8544_write(CMD, PCD8544_display_normal);

    PCD8544_clear_buffer(OFF);
    PCD8544_clear_screen(ON);
}


void PCD8544_reset(void)
{
     RST_pin_low();
     delay_us(100);
     RST_pin_high();
}


void PCD8544_write(unsigned char type, unsigned char value)
{
    switch(type)
    {
        case 1:
        {
            DC_pin_high();
            break;
        }

        default:
        {
            DC_pin_low();
            break;
        }
    }

    CS_pin_low();
    SPI_USCI_A0_write(value);
    CS_pin_high();
}
```

```
void PCD8544_set_contrast(unsigned char value)
{
    if(value >= 0x7F)
    {
       value = 0x7F;
    }

    PCD8544_write(CMD, (PCD8544_extended_instruction | PCD8544_function_set));
    PCD8544_write(CMD, (PCD8544_set_VOP | value));
    PCD8544_write(CMD, PCD8544_function_set);
}


void PCD8544_set_cursor(unsigned char x_pos, unsigned char y_pos)
{
    PCD8544_write(CMD, (PCD8544_set_X_addr | x_pos));
    PCD8544_write(CMD, (PCD8544_set_Y_addr | y_pos));
}


void PCD8544_print_char(unsigned char ch, unsigned char colour)
{
     unsigned char s = 0;
     unsigned char chr = 0;

     for(s = 0; s <= 4; s++)
     {
           chr = font[(ch - 0x20)][s];
           if(colour == BLACK)
           {
               chr = ~chr;
           }
           PCD8544_write(DAT, chr);
     }
}


void PCD8544_print_custom_char(unsigned char *map)
{
    unsigned char s = 0;

    for(s = 0; s <= 4; s++)
    {
        PCD8544_write(DAT, *map++);
    }
}


void PCD8544_fill(unsigned char bufr)
{
    unsigned int s = 0;

    PCD8544_set_cursor(0, 0);

    for(s = 0; s < buffer_size; s++)
    {
        PCD8544_write(DAT, bufr);
    }
}


void PCD8544_clear_buffer(unsigned char colour)
{
```

```
    signed char x_pos = (X_max - 1);
    signed char y_pos = (Rows - 1);

    while(x_pos > -1)
    {
        while(y_pos > -1)
        {
            PCD8544_buffer[x_pos][y_pos] = colour;
            y_pos--;
        }

        x_pos--;
    }
}


void PCD8544_clear_screen(unsigned char colour)
{
    unsigned char x_pos = 0;
    unsigned char y_pos = 0;

    for(y_pos = 0; y_pos < Rows; y_pos++)
    {
        for(x_pos = 0; x_pos < X_max; x_pos++)
        {
            print_string(x_pos, y_pos, " ", colour);
        }
    }
}


void print_image(const unsigned char *bmp)
{
    unsigned int s = 0;

    PCD8544_set_cursor(0, 0);

    for(s = 0; s < buffer_size; s++)
    {
        PCD8544_write(DAT, bmp[s]);
    }
}


void print_string(unsigned char x_pos, unsigned char y_pos, unsigned char *ch, unsigned cha
r colour)
{
    PCD8544_set_cursor(x_pos, y_pos);

    do
    {
       PCD8544_print_char(*ch++, colour);
    }while((*ch >= 0x20) && (*ch <= 0x7F));
}


void print_chr(unsigned char x_pos, unsigned char y_pos, signed int value, unsigned char co
lour)
{
    unsigned char ch = 0x00;

    if(value < 0)
    {
        PCD8544_set_cursor(x_pos, y_pos);
        PCD8544_print_char(0x2D, colour);
```

```
        value = -value;
    }
    else
    {
        PCD8544_set_cursor(x_pos, y_pos);
        PCD8544_print_char(0x20, colour);
    }

     if((value > 99) && (value <= 999))
     {
         ch = (value / 100);
         PCD8544_set_cursor((x_pos + 6), y_pos);
         PCD8544_print_char((48 + ch), colour);

         ch = ((value % 100) / 10);
         PCD8544_set_cursor((x_pos + 12), y_pos);
         PCD8544_print_char((48 + ch), colour);

         ch = (value % 10);
         PCD8544_set_cursor((x_pos + 18), y_pos);
         PCD8544_print_char((48 + ch), colour);
     }
     else if((value > 9) && (value <= 99))
     {
         ch = ((value % 100) / 10);
         PCD8544_set_cursor((x_pos + 6), y_pos);
         PCD8544_print_char((48 + ch), colour);

         ch = (value % 10);
         PCD8544_set_cursor((x_pos + 12), y_pos);
         PCD8544_print_char((48 + ch), colour);

         PCD8544_set_cursor((x_pos + 18), y_pos);
         PCD8544_print_char(0x20, colour);
     }
     else if((value >= 0) && (value <= 9))
     {
         ch = (value % 10);
         PCD8544_set_cursor((x_pos + 6), y_pos);
         PCD8544_print_char((48 + ch), colour);

         PCD8544_set_cursor((x_pos + 12), y_pos);
         PCD8544_print_char(0x20, colour);

         PCD8544_set_cursor((x_pos + 18), y_pos);
         PCD8544_print_char(0x20, colour);
     }
}


void print_int(unsigned char x_pos, unsigned char y_pos, signed long value, unsigned char c
olour)
{
    unsigned char ch = 0x00;

    if(value < 0)
    {
        PCD8544_set_cursor(x_pos, y_pos);
        PCD8544_print_char(0x2D, colour);
        value = -value;
    }
    else
    {
        PCD8544_set_cursor(x_pos, y_pos);
        PCD8544_print_char(0x20, colour);
    }
```

```
    if(value > 9999)
    {
        ch = (value / 10000);
        PCD8544_set_cursor((x_pos + 6), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = ((value % 10000)/ 1000);
        PCD8544_set_cursor((x_pos + 12), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = ((value % 1000) / 100);
        PCD8544_set_cursor((x_pos + 18), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = ((value % 100) / 10);
        PCD8544_set_cursor((x_pos + 24), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = (value % 10);
        PCD8544_set_cursor((x_pos + 30), y_pos);
        PCD8544_print_char((48 + ch), colour);
    }

    else if((value > 999) && (value <= 9999))
    {
        ch = ((value % 10000)/ 1000);
        PCD8544_set_cursor((x_pos + 6), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = ((value % 1000) / 100);
        PCD8544_set_cursor((x_pos + 12), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = ((value % 100) / 10);
        PCD8544_set_cursor((x_pos + 18), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = (value % 10);
        PCD8544_set_cursor((x_pos + 24), y_pos);
        PCD8544_print_char((48 + ch), colour);

        PCD8544_set_cursor((x_pos + 30), y_pos);
        PCD8544_print_char(0x20, colour);
    }
    else if((value > 99) && (value <= 999))
    {
        ch = ((value % 1000) / 100);
        PCD8544_set_cursor((x_pos + 6), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = ((value % 100) / 10);
        PCD8544_set_cursor((x_pos + 12), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = (value % 10);
        PCD8544_set_cursor((x_pos + 18), y_pos);
        PCD8544_print_char((48 + ch), colour);

        PCD8544_set_cursor((x_pos + 24), y_pos);
        PCD8544_print_char(0x20, colour);

        PCD8544_set_cursor((x_pos + 30), y_pos);
        PCD8544_print_char(0x20, colour);
    }
    else if((value > 9) && (value <= 99))
```

```
    {
        ch = ((value % 100) / 10);
        PCD8544_set_cursor((x_pos + 6), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = (value % 10);
        PCD8544_set_cursor((x_pos + 12), y_pos);
        PCD8544_print_char((48 + ch), colour);

        PCD8544_set_cursor((x_pos + 18), y_pos);
        PCD8544_print_char(0x20, colour);

        PCD8544_set_cursor((x_pos + 24), y_pos);
        PCD8544_print_char(0x20, colour);

        PCD8544_set_cursor((x_pos + 30), y_pos);
        PCD8544_print_char(0x20, colour);
    }
    else
    {
        ch = (value % 10);
        PCD8544_set_cursor((x_pos + 6), y_pos);
        PCD8544_print_char((48 + ch), colour);

        PCD8544_set_cursor((x_pos + 12), y_pos);
        PCD8544_print_char(0x20, colour);

        PCD8544_set_cursor((x_pos + 18), y_pos);
        PCD8544_print_char(0x20, colour);

        PCD8544_set_cursor((x_pos + 24), y_pos);
        PCD8544_print_char(0x20, colour);

        PCD8544_set_cursor((x_pos + 30), y_pos);
        PCD8544_print_char(0x20, colour);
    }
}


void print_decimal(unsigned char x_pos, unsigned char y_pos, unsigned int value, unsigned c
har points, unsigned char colour)
{
    unsigned char ch = 0x00;

    PCD8544_set_cursor(x_pos, y_pos);
    PCD8544_print_char(0x2E, colour);

    ch = (value / 1000);
    PCD8544_set_cursor((x_pos + 6), y_pos);
    PCD8544_print_char((48 + ch), colour);

    if(points > 1)
    {
        ch = ((value % 1000) / 100);
        PCD8544_set_cursor((x_pos + 12), y_pos);
        PCD8544_print_char((48 + ch), colour);


        if(points > 2)
        {
            ch = ((value % 100) / 10);
            PCD8544_set_cursor((x_pos + 18), y_pos);
            PCD8544_print_char((48 + ch), colour);

            if(points > 3)
```

```
            {
                ch = (value % 10);
                PCD8544_set_cursor((x_pos + 24), y_pos);
                PCD8544_print_char((48 + ch), colour);;
            }
        }
    }
}


void print_float(unsigned char x_pos, unsigned char y_pos, float value, unsigned char point
s, unsigned char colour)
{
    signed long tmp = 0x00;

    tmp = ((signed long)value);
    print_int(x_pos, y_pos, tmp, colour);
    tmp = ((value - tmp) * 10000);

    if(tmp < 0)
    {
       tmp = -tmp;
    }

    if((value >= 9999) && (value < 99999))
    {
        print_decimal((x_pos + 36), y_pos, tmp, points, colour);
    }
    else if((value >= 999) && (value < 9999))
    {
        print_decimal((x_pos + 30), y_pos, tmp, points, colour);
    }
    else if((value >= 99) && (value < 999))
    {
        print_decimal((x_pos + 24), y_pos, tmp, points, colour);
    }
    else if((value >= 9) && (value < 99))
    {
        print_decimal((x_pos + 18), y_pos, tmp, points, colour);
    }
    else if(value < 9)
    {
        print_decimal((x_pos + 12), y_pos, tmp, points, colour);
        if((value) < 0)
        {
            PCD8544_set_cursor(x_pos, y_pos);
            PCD8544_print_char(0x2D, colour);
        }
        else
        {
            PCD8544_set_cursor(x_pos, y_pos);
            PCD8544_print_char(0x20, colour);
        }
    }
}


void Draw_Pixel(unsigned char x_pos, unsigned char y_pos, unsigned char colour)
{
    unsigned char row = 0;
    unsigned char value = 0;

    if((x_pos >= X_max) || (y_pos >= Y_max))
    {
        return;
```

```
    }

    row = (y_pos >> 3);

    value = PCD8544_buffer[x_pos][row];

    if(colour == BLACK)
    {
        value |= (1 << (y_pos % 8));
    }
    else if(colour == WHITE)
    {
        value &= (~(1 << (y_pos % 8)));
    }
    else if(colour == PIXEL_XOR)
    {
        value ^= (1 << (y_pos % 8));
    }

    PCD8544_buffer[x_pos][row] = value;

    PCD8544_set_cursor(x_pos, row);
    PCD8544_write(DAT, value);
}


void Draw_Line(signed int x1, signed int y1, signed int x2, signed int y2, unsigned char co
lour)
{
    signed int dx = 0x0000;
    signed int dy = 0x0000;
    signed int stepx = 0x0000;
    signed int stepy = 0x0000;
    signed int fraction = 0x0000;

    dy = (y2 - y1);
    dx = (x2 - x1);

    if (dy < 0)
    {
        dy = -dy;
        stepy = -1;
    }
    else
    {
        stepy = 1;
    }

    if (dx < 0)
    {
        dx = -dx;
        stepx = -1;
    }
    else
    {
        stepx = 1;
    }

    dx <<= 0x01;
    dy <<= 0x01;

    Draw_Pixel(x1, y1, colour);

    if (dx > dy)
    {
        fraction = (dy - (dx >> 1));
```

```
        while (x1 != x2)
        {
            if (fraction >= 0)
            {
                y1 += stepy;
                fraction -= dx;
            }
            x1 += stepx;
            fraction += dy;

            Draw_Pixel(x1, y1, colour);
        }
    }
    else
    {
        fraction = (dx - (dy >> 1));

        while (y1 != y2)
        {
            if (fraction >= 0)
            {
                x1 += stepx;
                fraction -= dy;
            }
            y1 += stepy;
            fraction += dx;
            Draw_Pixel(x1, y1, colour);
        }
    }
}


void Draw_Rectangle(signed int x1, signed int y1, signed int x2, signed int y2, unsigned ch
ar fill, unsigned char colour)
{
     unsigned char i = 0x00;
     unsigned char xmin = 0x00;
     unsigned char xmax = 0x00;
     unsigned char ymin = 0x00;
     unsigned char ymax = 0x00;

     if(fill != NO)
     {
        if(x1 < x2)
        {
           xmin = x1;
           xmax = x2;
        }
        else
        {
           xmin = x2;
           xmax = x1;
        }

        if(y1 < y2)
        {
           ymin = y1;
           ymax = y2;
        }
        else
        {
           ymin = y2;
           ymax = y1;
        }

        for(; xmin <= xmax; ++xmin)
```

```
        {
             for(i = ymin; i <= ymax; ++i)
             {
                 Draw_Pixel(xmin, i, colour);
             }
         }
     }

     else
     {
        Draw_Line(x1, y1, x2, y1, colour);
        Draw_Line(x1, y2, x2, y2, colour);
        Draw_Line(x1, y1, x1, y2, colour);
        Draw_Line(x2, y1, x2, y2, colour);
     }
}


void Draw_Circle(signed int xc, signed int yc, signed int radius, unsigned char fill, unsig
ned char colour)
{
    signed int a = 0x0000;
    signed int b = 0x0000;
    signed int p = 0x0000;

    b = radius;
    p = (1 - b);

    do
    {
        if(fill != NO)
        {
            Draw_Line((xc - a), (yc + b), (xc + a), (yc + b), colour);
            Draw_Line((xc - a), (yc - b), (xc + a), (yc - b), colour);
            Draw_Line((xc - b), (yc + a), (xc + b), (yc + a), colour);
            Draw_Line((xc - b), (yc - a), (xc + b), (yc - a), colour);
        }
        else
        {
            Draw_Pixel((xc + a), (yc + b), colour);
            Draw_Pixel((xc + b), (yc + a), colour);
            Draw_Pixel((xc - a), (yc + b), colour);
            Draw_Pixel((xc - b), (yc + a), colour);
            Draw_Pixel((xc + b), (yc - a), colour);
            Draw_Pixel((xc + a), (yc - b), colour);
            Draw_Pixel((xc - a), (yc - b), colour);
            Draw_Pixel((xc - b), (yc - a), colour);
        }

        if(p < 0)
        {
            p += (0x03 + (0x02 * a++));
        }
        else
        {
            p += (0x05 + (0x02 * ((a++) - (b--))));
        }
    }while(a <= b);
}
```

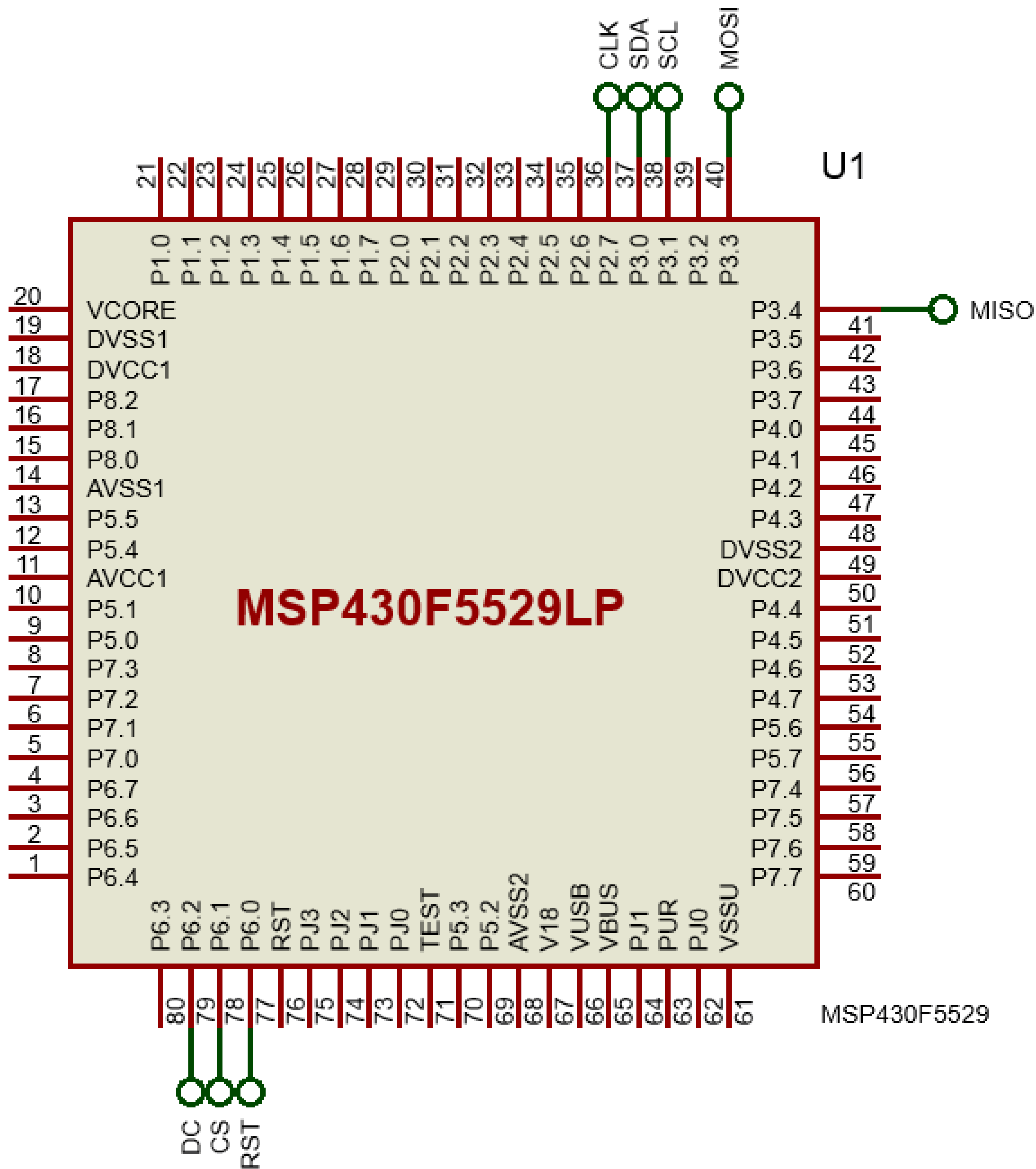
main.c

```
#include <msp430.h>
#include "driverlib.h"
```

```
#include "delay.h"
#include "USCI_I2C.h"
#include "USCI_SPI.h"
#include "PCD8544.h"
#include "BH1750.h"


void clock_init(void);


void main(void)
{
    unsigned int LX = 0x0000;
    unsigned int tmp = 0x0000;

    WDT_A_hold(WDT_A_BASE);

    clock_init();
    PCD8544_init();
    BH1750_init();

    print_string(1, 3, "Lux:", ON);

    while(1)
    {
        tmp = get_lux_value(cont_H_res_mode1, 20);

        if(tmp > 10)
        {
            LX = tmp;
        }
        else
        {
            LX = get_lux_value(cont_H_res_mode1, 140);
        }

        print_int(35, 3, LX, ON);

        delay_ms(200);
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_3,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);
```

```
    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}
```

Hardware Setup



Explanation

The USCI libraries for SPI and I2C here are mostly coded at raw level and without much driverlib help. These are based on my past tutorials. I actually copied from those tutorial examples and made some minor changes. These libraries are already proven and so there is nothing new here. Basically, this USCI example is mere a proof-of-concept of the fact that we can use USCI without driverlib.

We start by declaring I/O pins as per device's pin mapping. Let's see the I2C library first.

```
#include <msp430.h>
#include "driverlib.h"


#define USCI_B0_I2C_port                                          GPIO_PORT_P3

#define USCI_B0_I2C_SDA_pin                                       GPIO_PIN0
#define USCI_B0_I2C_SCL_pin                                       GPIO_PIN1

#define USCI_B1_I2C_port                                          GPIO_PORT_P4

#define USCI_B1_I2C_SDA_pin                                       GPIO_PIN1
#define USCI_B1_I2C_SCL_pin                                       GPIO_PIN2
```

GPIOs to be used need to set for alternative roles.

```
GPIO_setAsPeripheralModuleFunctionOutputPin(USCI_B0_I2C_port,
                                            (USCI_B0_I2C_SDA_pin
                                            | USCI_B0_I2C_SCL_pin));
```

The rest of the code is same as like the ones in the HW_I2C files of my past tutorial.

The demo here uses USCI B0 I2C module to operate a BH1750 light sensor as shown below:

```
void BH1750_write(unsigned char cmd)
{
    I2C_USCI_B0_write_byte(BH1750_addr, cmd);
}


unsigned int BH1750_read_word(void)
{
  unsigned long value = 0x0000;
  unsigned char bytes[2] = {0x00, 0x00};

  I2C_USCI_B0_read_word(0x11, bytes, 2);

  value = ((bytes[1] << 8) | bytes[0]);

  return value;
}
```

The same goes for the USCI SPI library.

```
#include <msp430.h>
#include "driverlib.h"
```

```c
#define USCI_A0_MOSI_port                   GPIO_PORT_P3
#define USCI_A0_MISO_port                   GPIO_PORT_P3
#define USCI_A0_CLK_port                    GPIO_PORT_P2

#define USCI_A0_MOSI_pin                    GPIO_PIN3
#define USCI_A0_MISO_pin                    GPIO_PIN4
#define USCI_A0_CLK_pin                     GPIO_PIN7

#define USCI_A1_MOSI_port                   GPIO_PORT_P4
#define USCI_A1_MISO_port                   GPIO_PORT_P4
#define USCI_A1_CLK_port                    GPIO_PORT_P4

#define USCI_A1_MOSI_pin                    GPIO_PIN4
#define USCI_A1_MISO_pin                    GPIO_PIN5
#define USCI_A1_CLK_pin                     GPIO_PIN0

#define USCI_B0_MOSI_port                   GPIO_PORT_P3
#define USCI_B0_MISO_port                   GPIO_PORT_P3
#define USCI_B0_CLK_port                    GPIO_PORT_P3

#define USCI_B0_MOSI_pin                    GPIO_PIN0
#define USCI_B0_MISO_pin                    GPIO_PIN1
#define USCI_B0_CLK_pin                     GPIO_PIN2

#define USCI_B1_MOSI_port                   GPIO_PORT_P4
#define USCI_B1_MISO_port                   GPIO_PORT_P4
#define USCI_B1_CLK_port                    GPIO_PORT_P4

#define USCI_B1_MOSI_pin                    GPIO_PIN1
#define USCI_B1_MISO_pin                    GPIO_PIN2
#define USCI_B1_CLK_pin                     GPIO_PIN3
```

SPI has separate input and output pins and so their declaration need care.

```c
GPIO_setAsPeripheralModuleFunctionInputPin(USCI_A0_MISO_port, USCI_A0_MISO_pin);

GPIO_setAsPeripheralModuleFunctionOutputPin(USCI_A0_MOSI_port, USCI_A0_MOSI_pin);

GPIO_setAsPeripheralModuleFunctionOutputPin(USCI_A0_CLK_port, USCI_A0_CLK_pin);
```

Many SPI devices need additional control pins and those can be declared as ordinary GPIOs. These pins can be any pins of our choice. In this case, our Nokia GLCD has a reset, chip select and data-command pin apart from SPI communication pins.

```c
#define RST_port                   GPIO_PORT_P6
#define CS_port                    GPIO_PORT_P6
#define DC_port                    GPIO_PORT_P6

#define RST_pin                    GPIO_PIN0
#define CS_pin                     GPIO_PIN1
#define DC_pin                     GPIO_PIN2

#define RST_pin_high()             GPIO_setOutputHighOnPin(RST_port, RST_pin)
#define RST_pin_low()              GPIO_setOutputLowOnPin(RST_port, RST_pin)

#define CS_pin_high()              GPIO_setOutputHighOnPin(CS_port, CS_pin)
#define CS_pin_low()               GPIO_setOutputLowOnPin(CS_port, CS_pin)

#define DC_pin_high()              GPIO_setOutputHighOnPin(DC_port, DC_pin)
#define DC_pin_low()               GPIO_setOutputLowOnPin(DC_port, DC_pin)

....
```

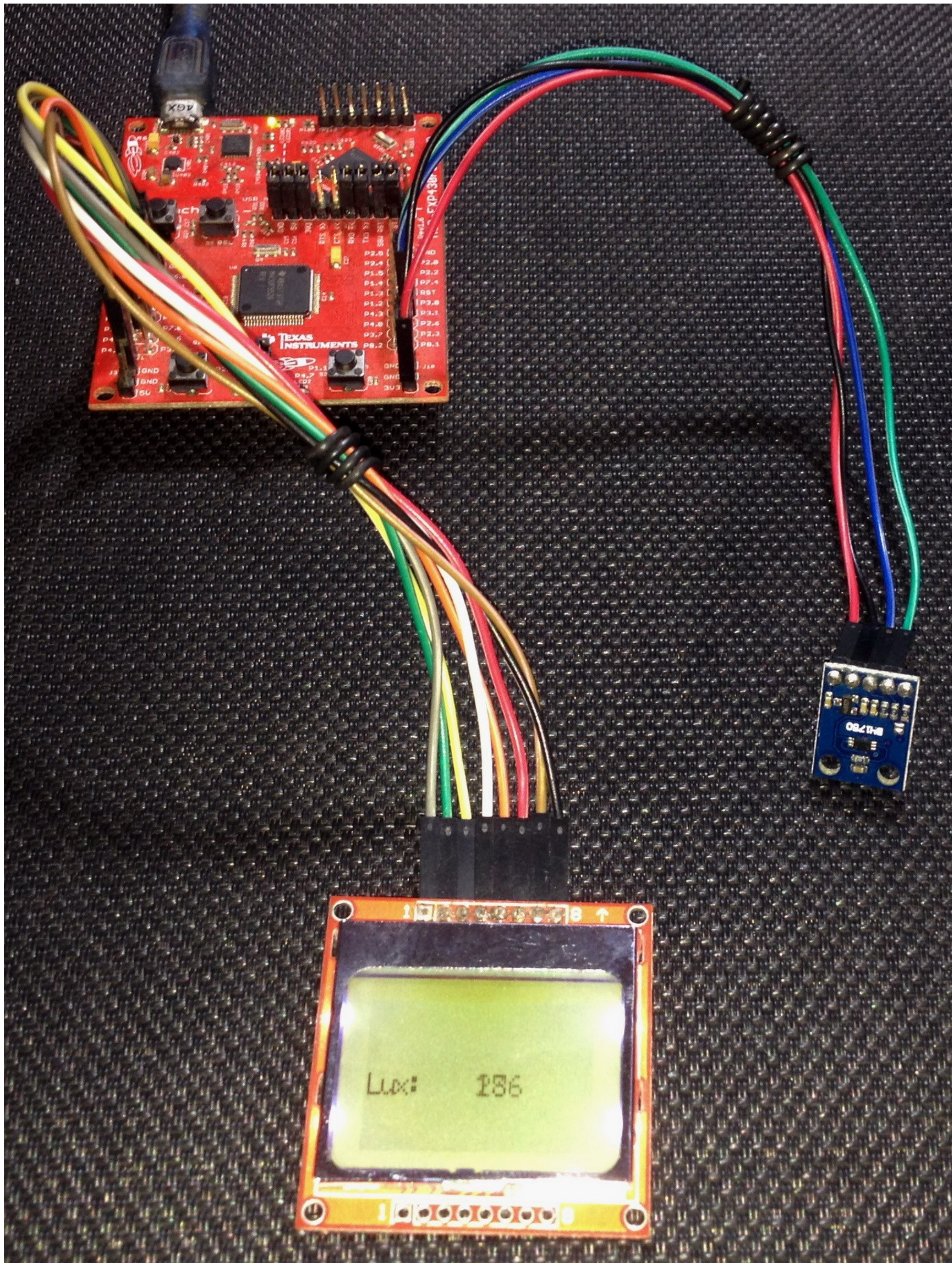
```
GPIO_setAsOutputPin(RST_port, RST_pin);
GPIO_setDriveStrength(RST_port, RST_pin, GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

GPIO_setAsOutputPin(CS_port, CS_pin);
GPIO_setDriveStrength(CS_port, CS_pin, GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

GPIO_setAsOutputPin(DC_port, DC_pin);
GPIO_setDriveStrength(DC_port, DC_pin, GPIO_FULL_OUTPUT_DRIVE_STRENGTH);
```

The code below shows how we used the USCI_A0 in SPI mode to operate the Nokia GLCD.

```
void PCD8544_write(unsigned char type, unsigned char value)
{
    switch(type)
    {
        case 1:
        {
            DC_pin_high();
            break;
        }

        default:
        {
            DC_pin_low();
            break;
        }
    }

    CS_pin_low();
    SPI_USCI_A0_write(value);
    CS_pin_high();
}
```

Writing the display is same as the TFT write example. There is no read back from the GLCD.

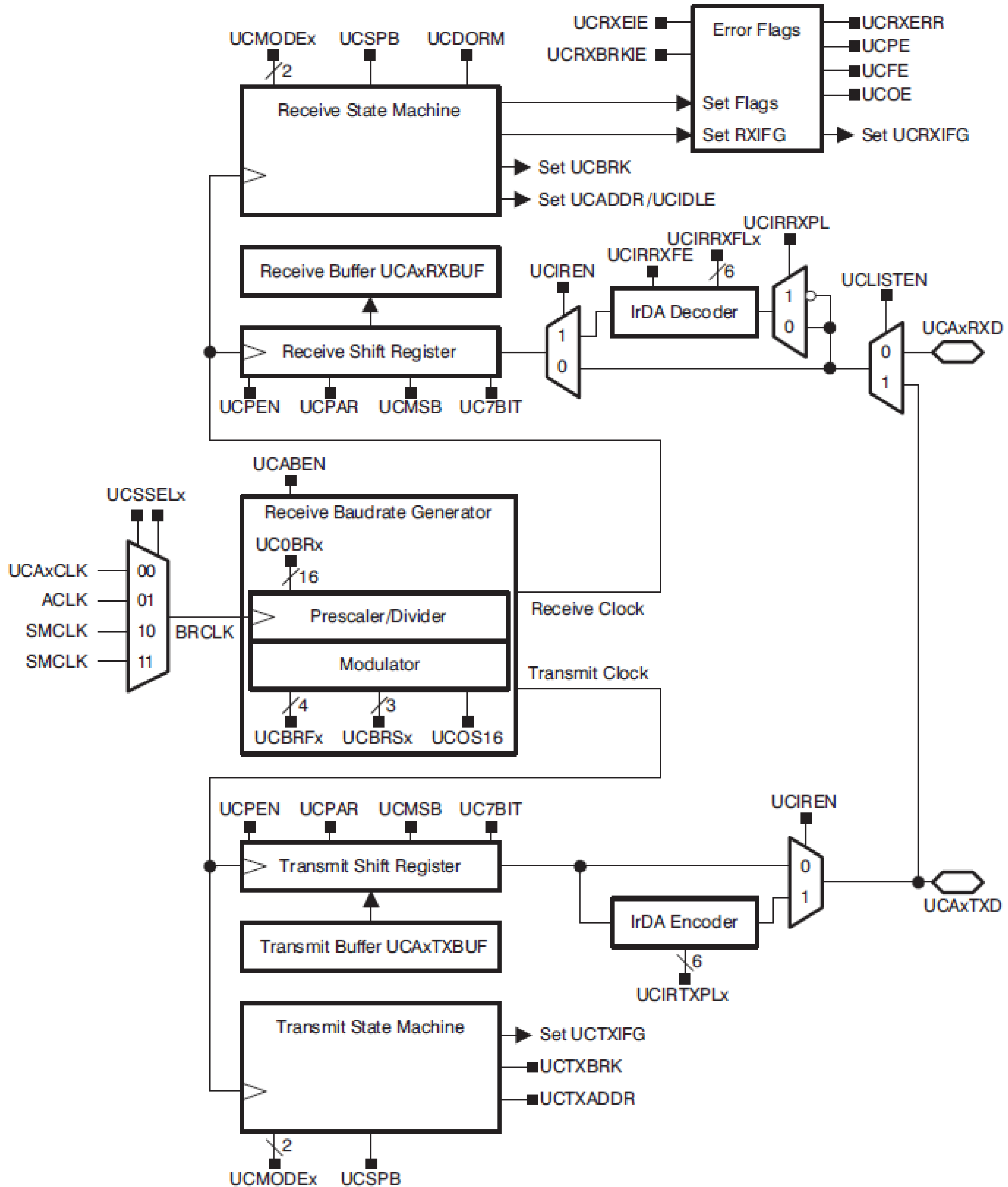
Demo



Demo video: https://youtu.be/bomb4JiuQAM

USCI – UART

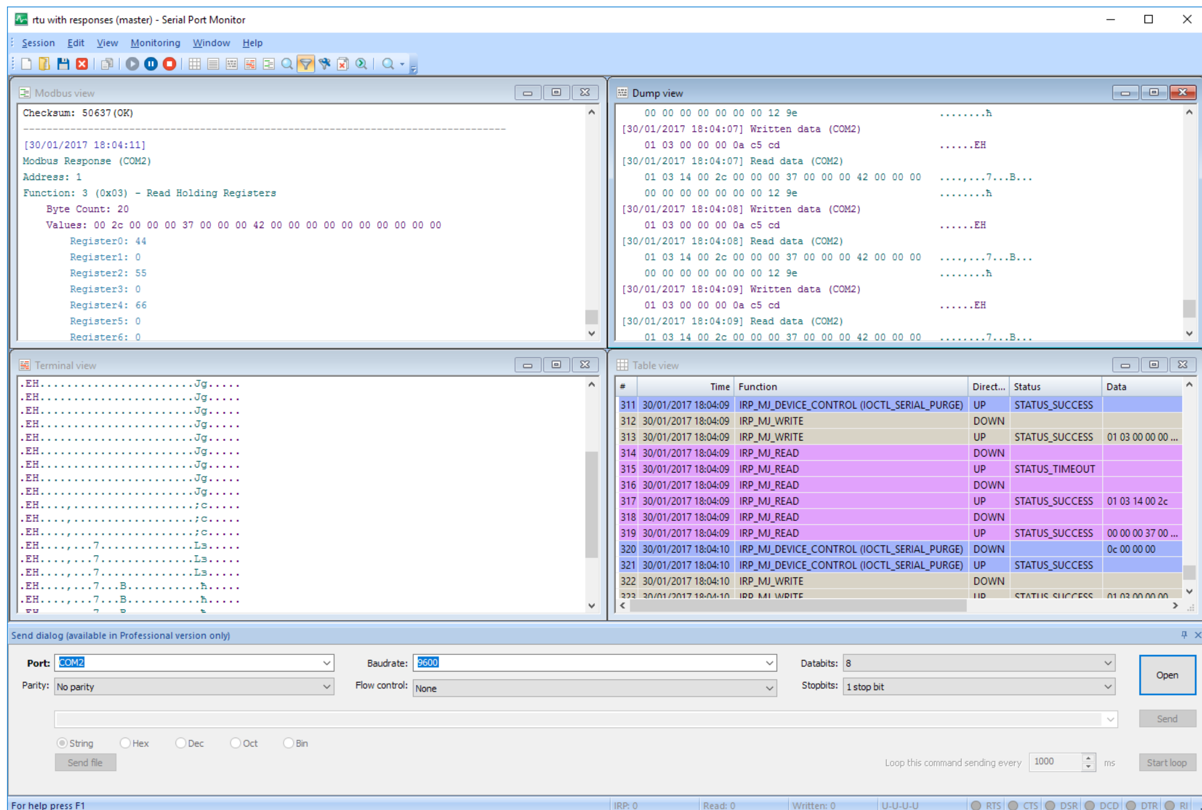
UART/serial communication in MSP430s is achieved using USCI modules. In MSP430F5529, USCI A modules can be used for both SPI and UART.



The block diagram of USCI in UART mode is shown above. As can be seen, UART module can be clocked with four clock sources. These clock sources can be fine-tuned for desired baud rates. There are separate buffers and state machines for transmission and reception parts. The UART module can use used for IrDA communication and there are optional internal encoder and decoder for so. Lastly, there are flags for events and errors at various points.

Eltima Soft

Eltima software is a US-based company that make some useful computer communication software interfaces. Eltima has been kind enough to supply me their Serial Port Monitor.



In embedded-system world, we all use some kind of serial port monitor but Eltima's serial port monitor has the following features that are not usually available in other software:

- COM data logging in real-time and saving for later uses
- Multi-port monitoring in one session
- Five different visualization modes
- Emulation of port
- Sniffing of MODBUS RTU and ASCII data
- Comparison of sessions
- Easy to use user-interface
- Changing of serial port parameters on-the-fly
- Digitally signed drivers
- Advanced filtering

Of the many features, I particularly liked the logging and MOBUS features. I have no other similar software to compare with it. I highly recommend readers to use it and see for themselves.

In this tutorial post, I have exclusively used this software.

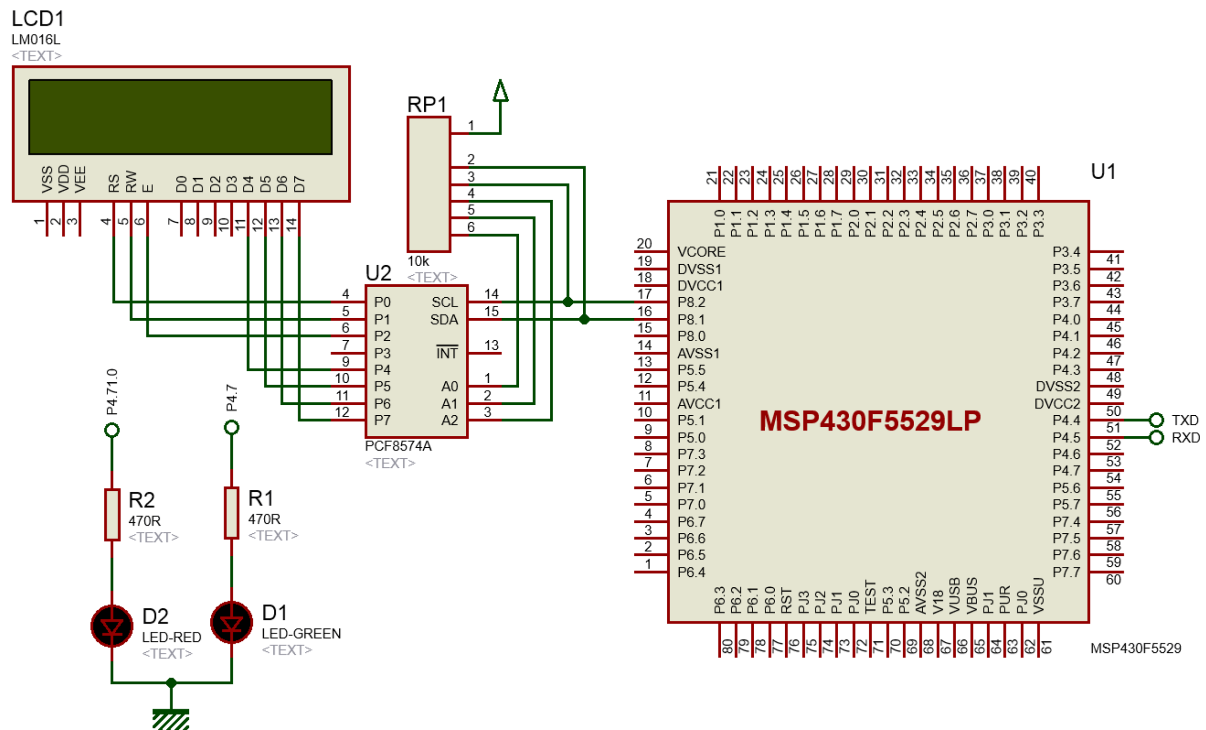
Code Example

```
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"
#include "lcd_print.h"


char RX_Data = 0;
char TX_Data = 0;


void clock_init(void);
void GPIO_init(void);
void USCI_UART_init(void);


#pragma vector = USCI_A1_VECTOR
__interrupt void UART_ISR(void)
{
    switch(__even_in_range(UCA1IV, 4))
    {
        case 0x00:      // None
        {
            break;
        }

        case 0x02:      //Data RX
        {
            RX_Data = USCI_A_UART_receiveData(USCI_A1_BASE);

            GPIO_toggleOutputOnPin(GPIO_PORT_P4,
                                   GPIO_PIN7);

            break;
        }

        case 0x04:      //TX Buffer Empty
        {
            break;
        }
    }
}


void main(void)
{
    WDT_A_hold(WDT_A_BASE);

    clock_init();

    USCI_UART_init();

    GPIO_init();

    LCD_init();
    LCD_clear_home();

    LCD_goto(0, 0);
    LCD_putstr("MSP430 USCI UART");
    LCD_goto(0, 1);
    LCD_putstr("TXD: ");
    LCD_goto(10, 1);
    LCD_putstr("RXD: ");
```

```
    while(1)
    {
        TX_Data = RX_Data;

        GPIO_toggleOutputOnPin(GPIO_PORT_P1,
                               GPIO_PIN0);

        USCI_A_UART_transmitData(USCI_A1_BASE,
                                 TX_Data);

        while(USCI_A_UART_getInterruptStatus(USCI_A1_BASE,
USCI_A_UART_TRANSMIT_INTERRUPT_FLAG) == 0);

        LCD_goto(5, 1);
        LCD_putchar(TX_Data);

        LCD_goto(15, 1);
        LCD_putchar(RX_Data);

        delay_ms(900);
    };
}


void clock_init(void)
{
    PMM_setVCore(PMM_CORE_LEVEL_3);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
                                               (GPIO_PIN4 | GPIO_PIN2));

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P5,
                                                (GPIO_PIN5 | GPIO_PIN3));

    UCS_setExternalClockSource(XT1_FREQ,
                               XT2_FREQ);

    UCS_turnOnXT2(UCS_XT2_DRIVE_4MHZ_8MHZ);

    UCS_turnOnLFXT1(UCS_XT1_DRIVE_3,
                    UCS_XCAP_3);

    UCS_initClockSignal(UCS_FLLREF,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initFLLSettle(MCLK_KHZ,
                      MCLK_FLLREF_RATIO);

    UCS_initClockSignal(UCS_SMCLK,
                        UCS_XT2CLK_SELECT,
                        UCS_CLOCK_DIVIDER_4);

    UCS_initClockSignal(UCS_ACLK,
                        UCS_XT1CLK_SELECT,
                        UCS_CLOCK_DIVIDER_1);
}


void GPIO_init(void)
{
    GPIO_setAsOutputPin(GPIO_PORT_P1,
                        GPIO_PIN0);
```

```
    GPIO_setDriveStrength(GPIO_PORT_P1,
                          GPIO_PIN0,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsOutputPin(GPIO_PORT_P4,
                        GPIO_PIN7);

    GPIO_setDriveStrength(GPIO_PORT_P4,
                          GPIO_PIN7,
                          GPIO_FULL_OUTPUT_DRIVE_STRENGTH);

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P4,
                                               GPIO_PIN5);

    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P4,
                                                GPIO_PIN4);
}


void USCI_UART_init(void)
{
    USCI_A_UART_initParam UART_Param = {0};

    UART_Param.selectClockSource = USCI_A_UART_CLOCKSOURCE_ACLK;
    UART_Param.clockPrescalar = 3;
    UART_Param.firstModReg = 0;
    UART_Param.secondModReg = 3;
    UART_Param.msborLsbFirst = USCI_A_UART_LSB_FIRST;
    UART_Param.parity = USCI_A_UART_NO_PARITY;
    UART_Param.numberofStopBits = USCI_A_UART_ONE_STOP_BIT;
    UART_Param.uartMode = USCI_A_UART_MODE;
    UART_Param.overSampling = USCI_A_UART_LOW_FREQUENCY_BAUDRATE_GENERATION;

    USCI_A_UART_init(USCI_A1_BASE,
                     &UART_Param);

    USCI_A_UART_resetDormant(USCI_A1_BASE);

    USCI_A_UART_enable(USCI_A1_BASE);

    USCI_A_UART_enableInterrupt(USCI_A1_BASE,
                                USCI_A_UART_RECEIVE_INTERRUPT);

    __enable_interrupt();
}
```

Hardware Setup



Explanation

Firstly, GPIO pins are initialized as secondary function pins.

```
GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P4, GPIO_PIN5);

GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P4, GPIO_PIN4);
```

The following function initializes USCI A1 module in UART mode.

```
void USCI_UART_init(void)
{
    USCI_A_UART_initParam UART_Param = {0};

    UART_Param.selectClockSource = USCI_A_UART_CLOCKSOURCE_ACLK;
    UART_Param.clockPrescalar = 3;
    UART_Param.firstModReg = 0;
    UART_Param.secondModReg = 3;
    UART_Param.msborLsbFirst = USCI_A_UART_LSB_FIRST;
    UART_Param.parity = USCI_A_UART_NO_PARITY;
    UART_Param.numberofStopBits = USCI_A_UART_ONE_STOP_BIT;
    UART_Param.uartMode = USCI_A_UART_MODE;
    UART_Param.overSampling = USCI_A_UART_LOW_FREQUENCY_BAUDRATE_GENERATION;

    USCI_A_UART_init(USCI_A1_BASE, &UART_Param);
    USCI_A_UART_resetDormant(USCI_A1_BASE);
    USCI_A_UART_enable(USCI_A1_BASE);
    USCI_A_UART_enableInterrupt(USCI_A1_BASE, USCI_A_UART_RECEIVE_INTERRUPT);
    __enable_interrupt();
}
```

In serial communication, baud rate selection is very important because two serial communication devices must negotiate under a common data transaction rate or else data is unrecognized or treated as garbage. Since baud rate clock-sensitive, we have to carefully chose clock and prescalar settings and this is reflected in the first setting values – *clockPrescalar*, *firstModReg* and *secondModReg*.

Now let's see how we came to these values. Refer to section ***36.3.13 Typical Baud Rates and Errors*** of ***MSP430x5xx and MSP430x6xx Family User's Guide***.

BRCLK Frequency (Hz)	Baud Rate (baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)		Maximum RX Error (%)	
32 768	1200	27	2	0	-2.8	1.4	-5.9	2.0
32 768	2400	13	6	0	-4.8	6.0	-9.7	8.3
32 768	4800	6	7	0	-12.1	5.7	-13.4	19.0
32 768	9600	3	3	0	-21.1	15.2	-44.3	21.3
1 000 000	9600	104	1	0	-0.5	0.6	-0.9	1.2
1 000 000	19200	52	0	0	-1.8	0	-2.6	0.9
1 000 000	38400	26	0	0	-1.8	0	-3.6	1.8
1 000 000	57600	17	3	0	-2.1	4.8	-6.8	5.8
1 000 000	115200	8	6	0	-7.8	6.4	-9.7	16.1

We have used ACLK as the clock source for UART. ACLK happens to be derived from XT1 oscillator and so its frequency is 32.768 kHz.

```
UCS_initClockSignal(UCS_ACLK, UCS_XT1CLK_SELECT, UCS_CLOCK_DIVIDER_1);
```

Therefore, to achieve a baud rate of 9600, we have to put the values as shown in the highlighted section of the table above. With these settings the TX-RX errors are high but still we will go with these just to see that even after such high error rates, the serial communication remains smooth and steady.

The rest of the settings describe other properties of our MSP430's serial port. These include number of stop bits, parity, USCI mode and others.

We will be using reception interrupt and so it is also needed to be enabled.

```
USCI_A_UART_enableInterrupt(USCI_A1_BASE, USCI_A_UART_RECEIVE_INTERRUPT);
```

Sending data is very easy. It is just like what we did in SPI examples. We write the data to be sent and wait for the transmit buffer to get empty.

```
USCI_A_UART_transmitData(USCI_A1_BASE, TX_Data);
while(USCI_A_UART_getInterruptStatus(USCI_A1_BASE,
                                     USCI_A_UART_TRANSMIT_INTERRUPT_FLAG) == 0);
```

Data is received in the interrupt. It is a very simple process. Once a reception interrupt is triggered, we just have to read the reception buffer. Interrupt flag is automatically cleared.

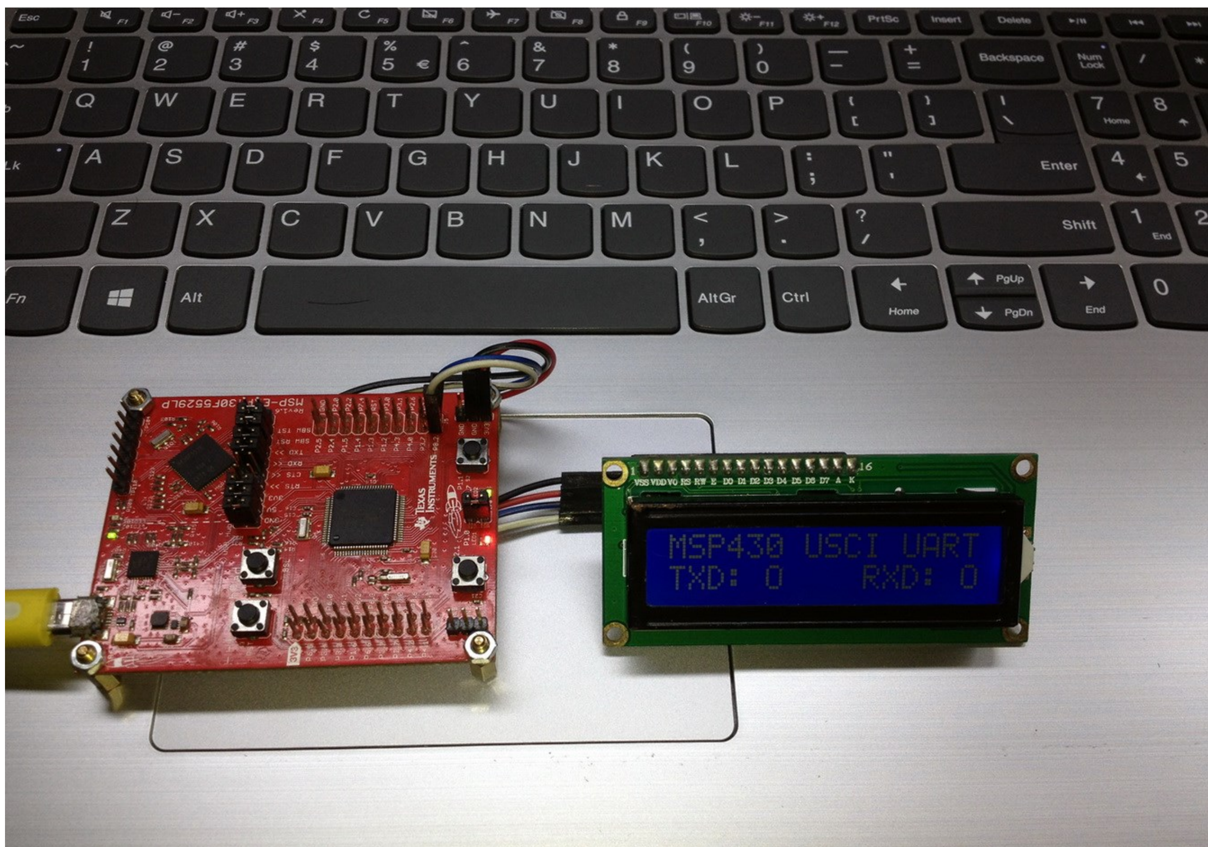
```
#pragma vector = USCI_A1_VECTOR
__interrupt void UART_ISR(void)
{
    switch(__even_in_range(UCA1IV, 4))
```

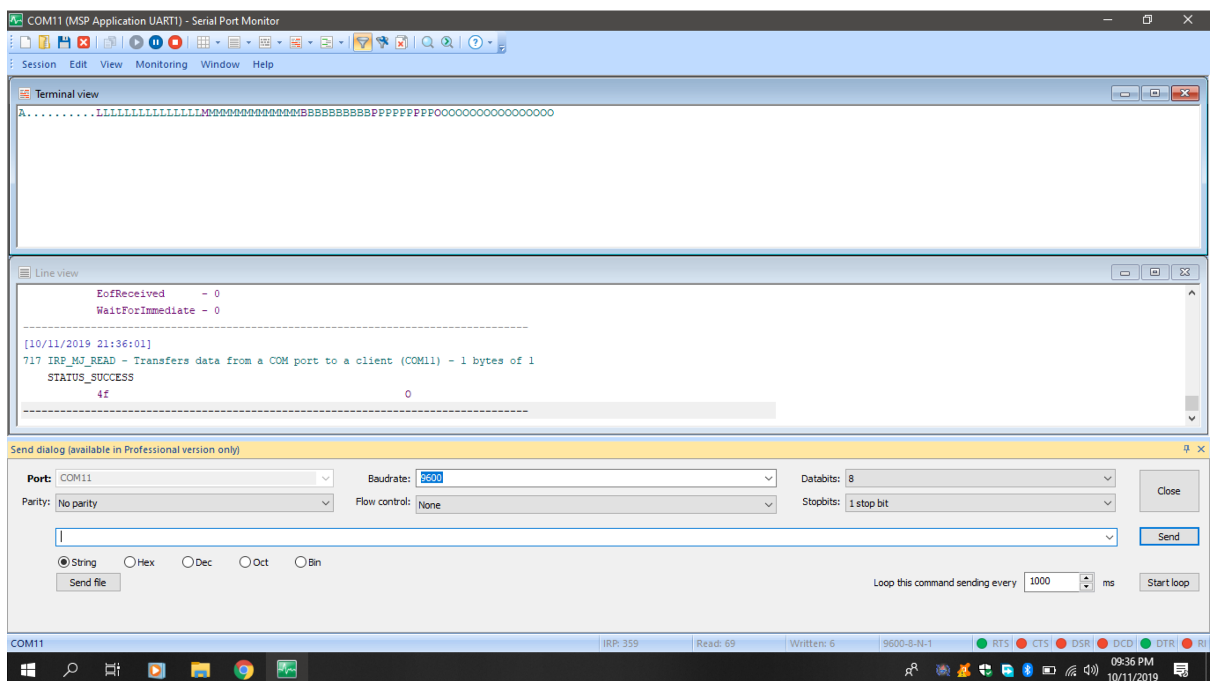
```
    {
        case 0x00:      // None
        {
            break;
        }

        case 0x02:      //Data RX
        {
            RX_Data = USCI_A_UART_receiveData(USCI_A1_BASE);

            GPIO_toggleOutputOnPin(GPIO_PORT_P4, GPIO_PIN7);

            break;
        }

        case 0x04:      //TX Buffer Empty
        {
            break;
        }
    }
}
```

We chose interrupt method for reception because we don't know for sure when a data will be sent to the MSP430 and as such so we don't want to miss any data sent to the micro.

The demo here simply echoes any character data that has been sent to it from a host PC.

Demo





Demo video: https://youtu.be/0gZMtbxT7Sw

USB Module Overview

The USB module of MSP430F5529 is not very difficult to use. TI has provided some working deployable examples in their 430ware software suite. However, some explanation is needed at first to get started. We will go through two basic USB module examples – USB CDC and USB HID.

In simple terms, USB CDC is a USB mode with which we can emulate virtual serial COM ports and USB HID is another common USB mode with which we can make devices like USB keyboard, mouse, input devices, printer, scanner, etc as well as other similar I/O devices. MSP430F5529 also supports mass storage class. In all cases, we create a way to make our MSP430 micro communicate with a host device.

Now let's see what we need to get started with USB coding. Apart from driverlib files and personal libraries we will need additional libraries for USB software development. Note the following folders:

- *USB API* Folder
- *USB_app* Folder
- *USB_config* Folder

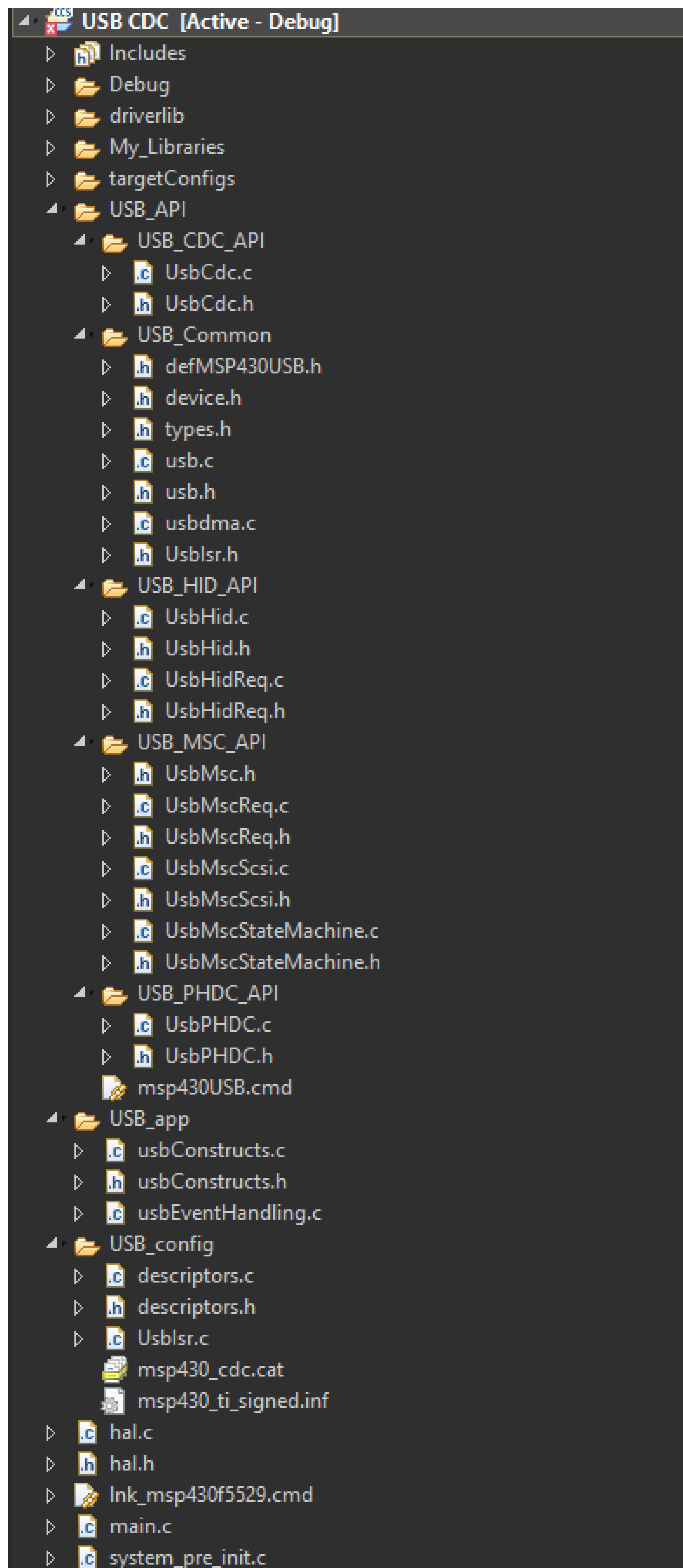
Apart from these we'll need some important files too:

- *Hal* header and source files
- *system_pre_init* source file

Fortunately for us we don't have to change anything in these files/folders. We just have to add them in our USB projects and the process of adding doesn't need any special attention. Alternatively, we can simply copy example USB projects from MSP430ware and have them customized as per need.

Very briefly speaking, *USB API folder*, sub-folders and files in them are what describe a specific USB mode's working. They list relevant functions for a given mode. *USB_app* folder and associated files further elaborate these operations. USB devices need descriptors and *USB_config* folder and files contain them. *Ha*l source and header files contain clock and GPIO setups and *system_pre_init* basically contains nothing except watchdog setup.

Shown below is the folder tree of a typical USB project. All I stated so far is what boils down to this folder tree.


USB CDC [Active - Debug]
Includes
Debug
driverlib
My_Libraries
targetConfigs
USB_API
USB_CDC_API
UsbCdc.c
UsbCdc.h
USB_Common
defMSP430USB.h
device.h
types.h
usb.c
usb.h
usbdma.c
UsbIsr.h
USB_HID_API
UsbHid.c
UsbHid.h
UsbHidReq.c
UsbHidReq.h
USB_MSC_API
UsbMsc.h
UsbMscReq.c
UsbMscReq.h
UsbMscScsi.c
UsbMscScsi.h
UsbMscStateMachine.c
UsbMscStateMachine.h
USB_PHDC_API
UsbPHDC.c
UsbPHDC.h
msp430USB.cmd
USB_app
usbConstructs.c
usbConstructs.h
usbEventHandling.c
USB_config
descriptors.c
descriptors.h
UsbIsr.c
msp430_cdc.cat
msp430_ti_signed.inf
hal.c
hal.h
lnk_msp430f5529.cmd
main.c
system_pre_init.c

In the main, all of the aforementioned files are included as shown below:

```
#include "driverlib.h"

#include "USB_config/descriptors.h"
#include "USB_API/USB_Common/device.h"
#include "USB_API/USB_Common/usb.h"
#include "USB_API/USB_CDC_API/UsbCdc.h"
#include "USB_app/usbConstructs.h"

#include "hal.h"
```

Pay attention to these functions shown below:

```
PMM_setVCore(PMM_CORE_LEVEL_2);
USBHAL_initPorts();
USBHAL_initClocks(8000000);
USB_setup(TRUE, TRUE);
```

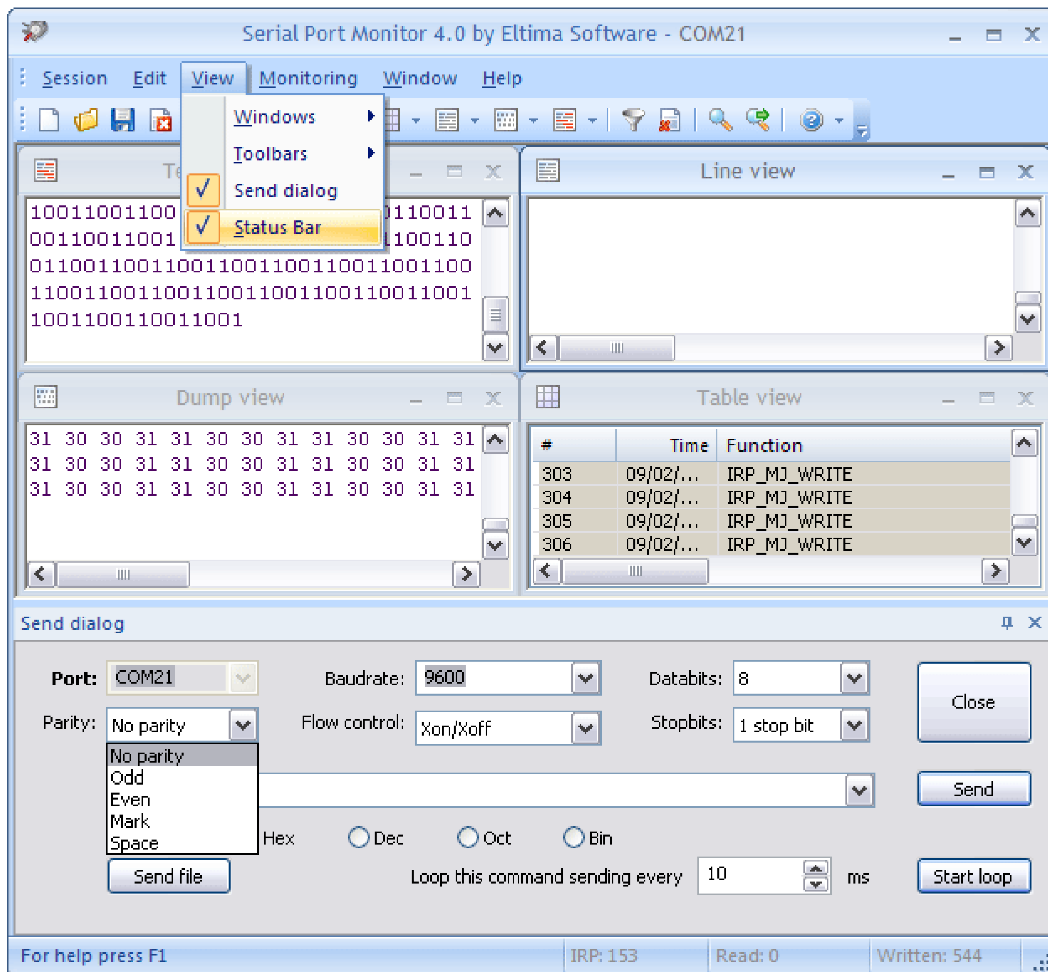
Clock setting is very important because USB hardware is clock sensitive. FLL is used for both MCLK and USB clock. It is also necessary to set the right level for PMM core. Usually level 2 is preferred as the clock speed is 8MHz. On start-up, all GPIO ports are driven low to avoid floating input and unwanted EMI-related issues. This causes some additional power consumption. Thus, it is recommended to setup GPIOs for other tasks after initial USB port initialization. Please refer to TI's official docs for additional info.

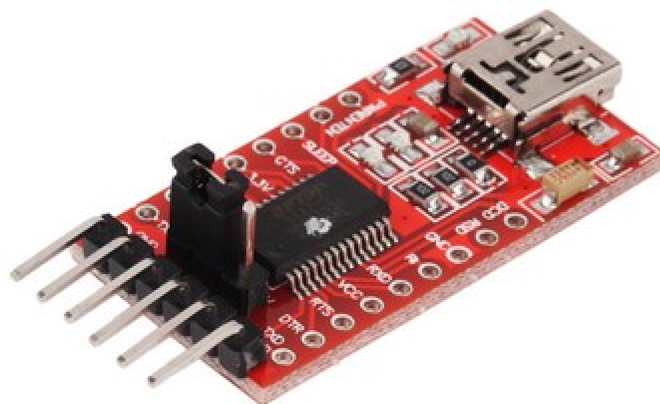
If you are not using official TI boards or equivalents and designing stuffs on your own then pay attention to USB hardware design considerations. EMC considerations are also needed to focused more alongside good PCB layout practices.

In addition to these, we can develop our own customized PC/mobile applications with Visual C Sharp (C#), Visual Basic or some other similar computer programming languages. This part is beyond the scope of this tutorial and won't be discussed.

USB CDC

With USB CDC mode we can directly transfer data to and from a PC/mobile device without needing an external USB-to-Serial converter like FT232, CP2102, PL2303, etc. We'll simply make a virtual COM port. The idea in this demo is to send out on-chip temperature sensor's temperature reading upon request from a host PC. The request is nothing special but just an ASCII character. The key that was pressed in the host PC's keyboard is then displayed on an LCD connected with the MSP430.





Code Example

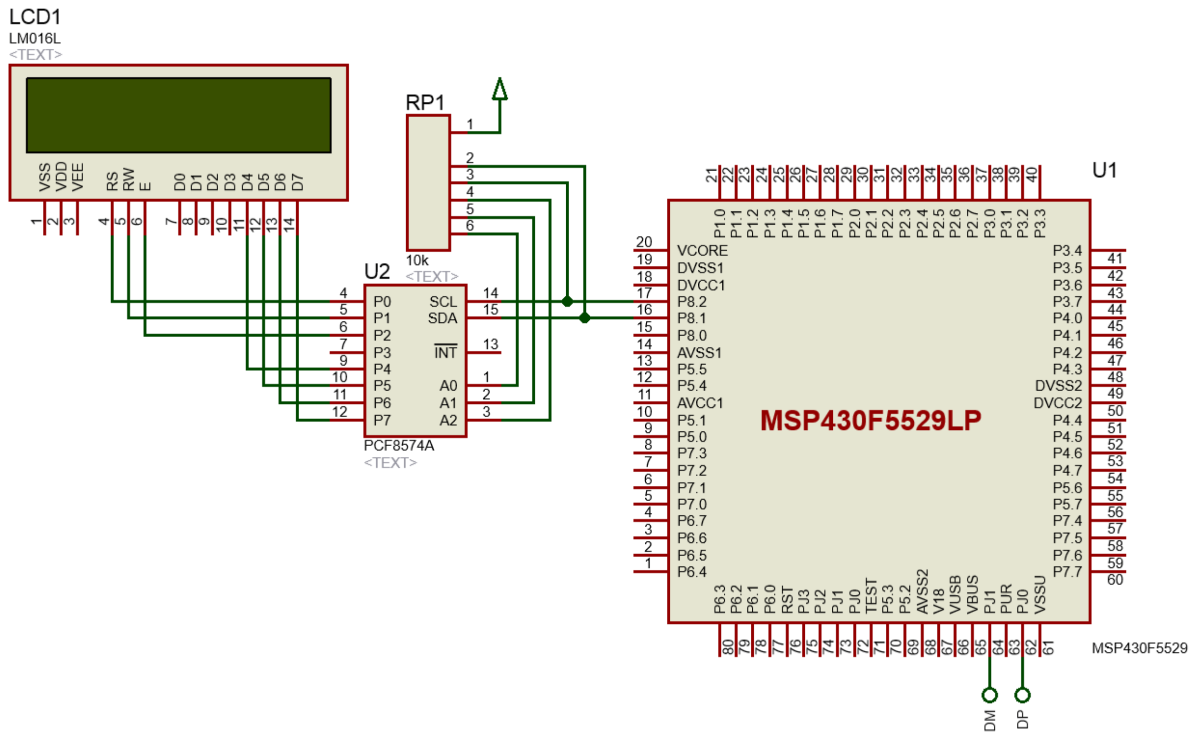
```
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"
#include "lcd_print.h"

#include "USB_config/descriptors.h"
#include "USB_API/USB_Common/device.h"
#include "USB_API/USB_Common/usb.h"
#include "USB_API/USB_CDC_API/UsbCdc.h"
#include "USB_app/usbConstructs.h"

#include "hal.h"


volatile uint8_t bCDCDataReceived_event = FALSE;


#define RX_BUFFER_SIZE  1
#define TX_BUFFER_SIZE  4

char dataBuffer[RX_BUFFER_SIZE] = "";
char nl[2] = "\n";
uint16_t count;


void ADC12_init(void);
void REF_init(void);


#pragma vector = UNMI_VECTOR
__interrupt void UNMI_ISR (void)
{
    switch (__even_in_range(SYSUNIV, SYSUNIV_BUSIFG ))
    {
        case SYSUNIV_NONE:
            __no_operation();
            break;
        case SYSUNIV_NMIIFG:
            __no_operation();
            break;
        case SYSUNIV_OFIFG:
            UCS_clearFaultFlag(UCS_XT2OFFG);
            UCS_clearFaultFlag(UCS_DCOFFG);
            SFR_clearInterrupt(SFR_OSCILLATOR_FAULT_INTERRUPT);
            break;
        case SYSUNIV_ACCVIFG:
            __no_operation();
            break;
        case SYSUNIV_BUSIFG:
            SYSBERRIV = 0;
            USB_disable();
    }
}


void main (void)
{
    float tmp = 0.0;
    signed int ADC_Count = 0;
    signed long temp = 0;
    uint8_t tc[TX_BUFFER_SIZE] = {0x00, 0x00, '\r', '\n'};
```

```
    WDT_A_hold(WDT_A_BASE);

    PMM_setVCore(PMM_CORE_LEVEL_2);

    REF_init();

    ADC12_init();

    LCD_init();
    LCD_clear_home();

    LCD_goto(1, 0);
    LCD_putstr("MSP430 USB CDC");

    LCD_goto(0, 1);
    LCD_putstr("T/'C:");

    LCD_goto(11, 1);
    LCD_putstr("RX:");

    USBHAL_initPorts();
    USBHAL_initClocks(8000000);
    USB_setup(TRUE, TRUE);

    __enable_interrupt();

    while (1)
    {
        uint8_t ReceiveError = 0;
        uint8_t SendError = 0;

        ADC_Count = ADC12_A_getResults(ADC12_A_BASE,
                                       ADC12_A_MEMORY_0);

        tmp = (((float)ADC_Count * 1.5) / 4095.0);
        temp = ((tmp - 0.688) / 0.00252);

        print_C(6, 1, temp);

        switch (USB_getConnectionState())
        {
            case ST_ENUM_ACTIVE:
            {
                __disable_interrupt();
                if (!USBCDC_getBytesInUSBBuffer(CDC0_INTFNUM))
                {
                    __bis_SR_register(LPM0_bits + GIE);
                }

                __enable_interrupt();


                if (bCDCDataReceived_event)
                {
                    bCDCDataReceived_event = FALSE;

                    count = USBCDC_receiveDataInBuffer((uint8_t*)dataBuffer,
                                                       RX_BUFFER_SIZE,
                                                       CDC0_INTFNUM);
                    LCD_goto(15, 1);
                    LCD_putstr(dataBuffer);
                }

                tc[0] = ((temp / 10) + 0x30);
                tc[1] = ((temp % 10) + 0x30);
```

```
                if(USBCDC_sendDataInBackground((uint8_t*)tc,
                                               TX_BUFFER_SIZE,
                                               CDC0_INTFNUM,
                                               1))
                {
                    SendError = 0x01;
                    break;
                }

                break;

            }
            case ST_PHYS_DISCONNECTED:
            case ST_ENUM_SUSPENDED:
            case ST_PHYS_CONNECTED_NOENUM_SUSP:
                __bis_SR_register(LPM3_bits + GIE);
                _NOP();
                break;

            case ST_ENUM_IN_PROGRESS:
            default:;
        }

        if (ReceiveError || SendError)
        {
        }
    }
}


void ADC12_init(void)
{
    ADC12_A_configureMemoryParam configureMemoryParam = {0};

    ADC12_A_init(ADC12_A_BASE,
                 ADC12_A_SAMPLEHOLDSOURCE_SC,
                 ADC12_A_CLOCKSOURCE_ACLK,
                 ADC12_A_CLOCKDIVIDER_1);

    ADC12_A_setupSamplingTimer(ADC12_A_BASE,
                               ADC12_A_CYCLEHOLD_768_CYCLES,
                               ADC12_A_CYCLEHOLD_4_CYCLES,
                               ADC12_A_MULTIPLESAMPLESENABLE);

    ADC12_A_setResolution(ADC12_A_BASE,
                          ADC12_A_RESOLUTION_12BIT);

    configureMemoryParam.memoryBufferControlIndex = ADC12_A_MEMORY_0;
    configureMemoryParam.inputSourceSelect = ADC12_A_INPUT_TEMPSENSOR;
    configureMemoryParam.positiveRefVoltageSourceSelect = ADC12_A_VREFPOS_INT;
    configureMemoryParam.negativeRefVoltageSourceSelect = ADC12_A_VREFNEG_AVSS;
    configureMemoryParam.endOfSequence = ADC12_A_NOTENDOFSEQUENCE;

    ADC12_A_configureMemory(ADC12_A_BASE,
                            &configureMemoryParam);

    ADC12_A_enable(ADC12_A_BASE);

    ADC12_A_startConversion(ADC12_A_BASE,
                            ADC12_A_MEMORY_0,
                            ADC12_A_REPEATED_SINGLECHANNEL);
}


void REF_init(void)
```

```
{
    while(REF_ACTIVE == Ref_isRefGenBusy(REF_BASE));

    Ref_setReferenceVoltage(REF_BASE,
                            REF_VREF1_5V);

    Ref_enableReferenceVoltage(REF_BASE);

    __delay_cycles(100);
}
```

Hardware Setup



Explanation

The setup for using the ADC and on-chip temperature sensor have already been discussed and demoed previously. Likewise, the basic USB setup has been discussed in the USB module overview section. Therefore, I'll skip these parts. The main loop is the part that is of our interest. As can be seen, temperature read from ADC is shown in the display first. The *switch-case* statement acts according to the state of the USB module's connection state. Of these states, the first one (*ST_ENUM_ACTIVE*) is of most importance. In this state, data is transferred by a method called enumeration. The rest of the states have other importance but these won't be needed here. The whole system is interrupt driven and so it is necessary to disable interrupts upon a reception event and reenable interrupts after processing them.

```
while (1)
{
    uint8_t ReceiveError = 0;
    uint8_t SendError = 0;

    ADC_Count = ADC12_A_getResults(ADC12_A_BASE, ADC12_A_MEMORY_0);

    tmp = (((float)ADC_Count * 1.5) / 4095.0);
```

```
    temp = ((tmp - 0.688) / 0.00252);

    print_C(6, 1, temp);

    switch (USB_getConnectionState())
    {
        case ST_ENUM_ACTIVE:
        {
            __disable_interrupt();
            if (!USBCDC_getBytesInUSBBuffer(CDC0_INTFNUM))
            {
                __bis_SR_register(LPM0_bits + GIE);
            }

            __enable_interrupt();


            if (bCDCDataReceived_event)
            {
                bCDCDataReceived_event = FALSE;

                count = USBCDC_receiveDataInBuffer((uint8_t*)dataBuffer,
                                                    RX_BUFFER_SIZE,
                                                    CDC0_INTFNUM);
                LCD_goto(15, 1);
                LCD_putstr(dataBuffer);
            }

            tc[0] = ((temp / 10) + 0x30);
            tc[1] = ((temp % 10) + 0x30);

            if(USBCDC_sendDataInBackground((uint8_t*)tc,
                                            TX_BUFFER_SIZE,
                                            CDC0_INTFNUM,
                                            1))
            {
                SendError = 0x01;
                break;
            }

            break;

        }
        case ST_PHYS_DISCONNECTED:
        case ST_ENUM_SUSPENDED:
        case ST_PHYS_CONNECTED_NOENUM_SUSP:
             __bis_SR_register(LPM3_bits + GIE);
             NOP();
             break;

        case ST_ENUM_IN_PROGRESS:
        default:;
    }

    if (ReceiveError || SendError)
    {
    }
}
```

Out of the all the jargon, the following codes are what that we need to receive and transmit data respectively.

```
USBCDC_receiveDataInBuffer((uint8_t*)dataBuffer, RX_BUFFER_SIZE, CDC0_INTFNUM);

USBCDC_sendDataInBackground((uint8_t*)tc, TX_BUFFER_SIZE, CDC0_INTFNUM, 1))
```

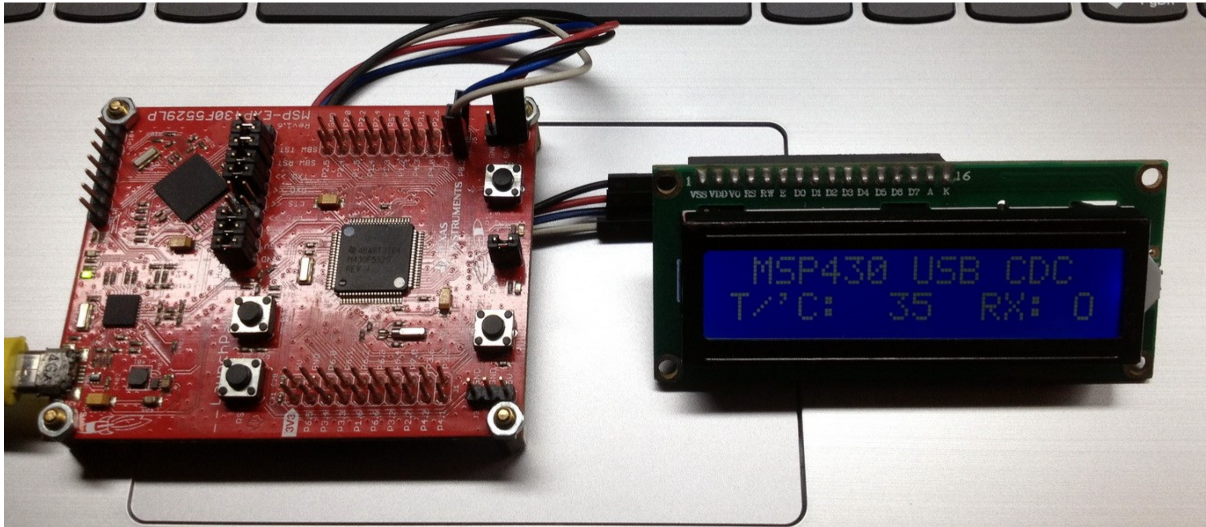
The receive function reads received data from *CDC0_INTFNUM* buffer. This is simply a buffer location that acts bidirectionally. We have to mention the array to which received data needs to be transferred. We also have to mention the size of this array. The transmit function is almost the same as the receive function. Note that all data transactions are done in chunks of bytes. For the sake of simplicity, errors and other USB states are not used. However, when developing a good real-life USB device these should not be ignored.

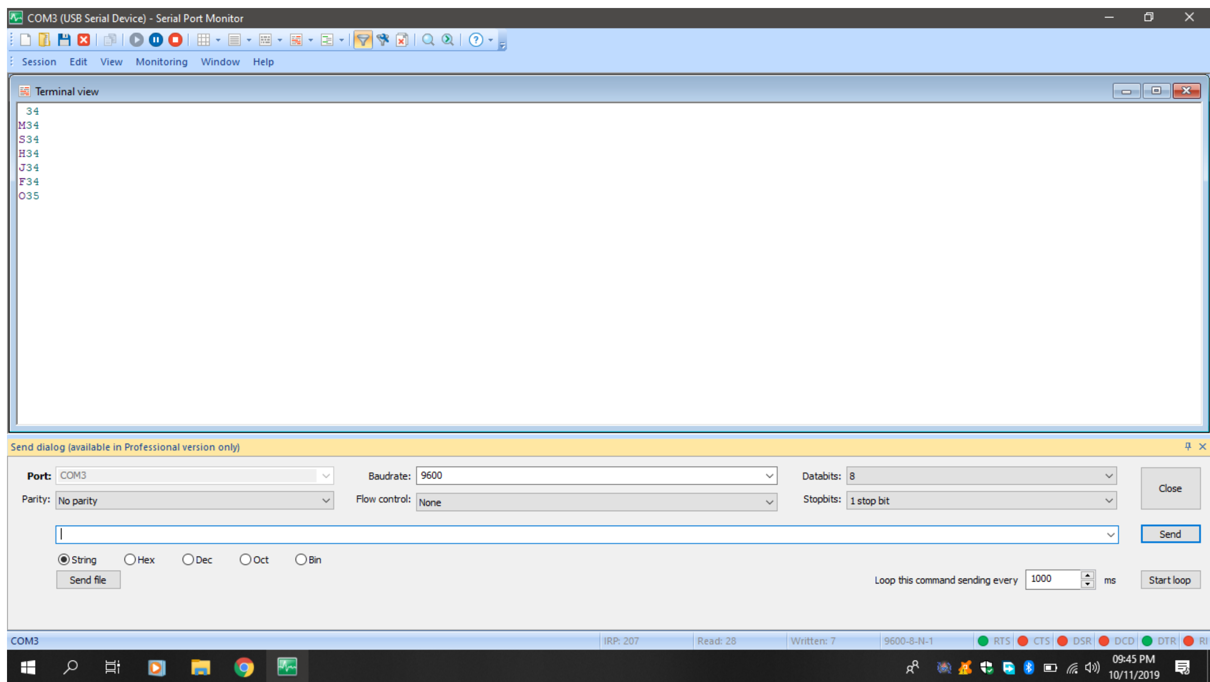
Since it is a USB example, our MSP430 launchpad board needs to be connected with a host PC. Note that no driver installation would be needed for this COM port device as it is a virtual/non-physical COM port and the port number can be any port number.

We'll need a port monitoring terminal software like the Eltima Soft's Serial Port Monitor.

The demo here works by sending out on-chip temperature sensor's reading when a keyboard key of the host PC is pressed. The key that was pressed is echoed back.

Demo





Demo video: https://youtu.be/0HfwllD4ig0

USB HID

Most of the common devices like keyboards, mice, printers, scanners, barcode readers, gamepads, etc that we use with our PCs and laptops in regular life are USB HID devices. We can use our MSP430F5529 microcontroller to make USB HID devices and in this example, we will use our MSP430 microcontroller to make a rudimentary mouse. Unlike conventional mice, our mouse will be using a keypad joystick instead of optoelectronics or mechanical trackball.

Code Example

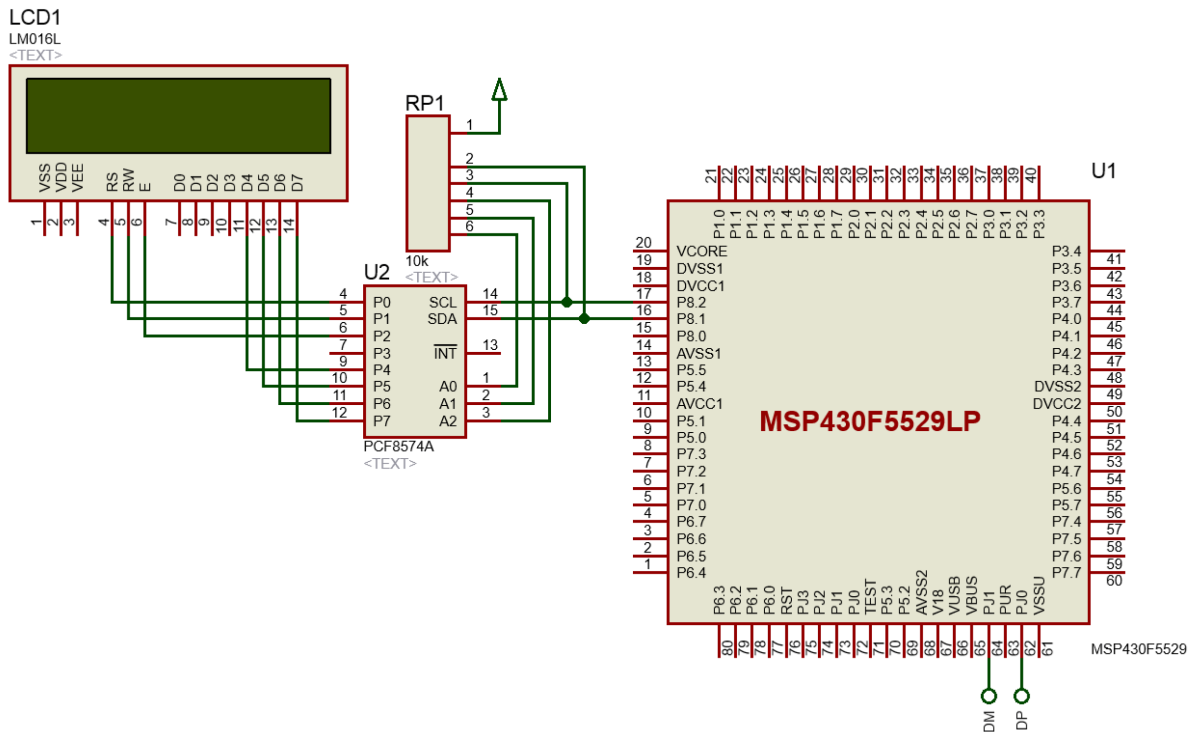
```
#include "driverlib.h"
#include "delay.h"
#include "lcd.h"
#include "lcd_print.h"

#include "USB_config/descriptors.h"
#include "USB_API/USB_Common/device.h"
#include "USB_API/USB_Common/usb.h"
#include "USB_API/USB_HID_API/UsbHid.h"


#include "hal.h"


#define LED_PORT            GPIO_PORT_P4
#define LED_PIN             GPIO_PIN7

#define MOUSE_PORT          GPIO_PORT_P6
#define MOUSE_UP_PIN        GPIO_PIN0
#define MOUSE_DOWN_PIN      GPIO_PIN1
#define MOUSE_LEFT_PIN      GPIO_PIN2
```

```
#define MOUSE_RIGHT_PIN      GPIO_PIN3
#define MOUSE_BUTTON_PIN     GPIO_PIN4

#define change_amount        1


typedef struct
{
    int8_t buttons;
    int8_t dX;
    int8_t dY;
    int8_t dZ;
}MOUSE_REPORT;


MOUSE_REPORT mouseReport = {0x00, 0x00, 0x00, 0x00};


void DIO_init(void);


#pragma vector = UNMI_VECTOR
__interrupt void UNMI_ISR (void)
{
    switch (__even_in_range(SYSUNIV, SYSUNIV_BUSIFG ))
    {
        case SYSUNIV_NONE:
        {
            __no_operation();
            break;
        }

        case SYSUNIV_NMIIFG:
        {
            __no_operation();
            break;
        }

        case SYSUNIV_OFIFG:
        {
            UCS_clearFaultFlag(UCS_XT2OFFG);
            UCS_clearFaultFlag(UCS_DCOFFG);
            SFR_clearInterrupt(SFR_OSCILLATOR_FAULT_INTERRUPT);
            break;
        }

        case SYSUNIV_ACCVIFG:
        {
            __no_operation();
            break;
        }
        case SYSUNIV_BUSIFG:
        {
            SYSBERRIV = 0;
            USB_disable();
        }
    }
}


void main (void)
{
    unsigned char t = 0x0000;

    WDT_A_hold(WDT_A_BASE);
```

```
    LCD_init();
    LCD_clear_home();

    LCD_goto(1, 0);
    LCD_putstr("MSP430 USB HID");

    LCD_goto(0, 1);
    LCD_putstr("X: ");

    LCD_goto(6, 1);
    LCD_putstr("Y:");

    LCD_goto(12, 1);
    LCD_putstr("B:");

    PMM_setVCore(PMM_CORE_LEVEL_2);
    USBHAL_initPorts();
    USBHAL_initClocks(8000000);
    USB_setup(TRUE, TRUE);

    DIO_init();

    __enable_interrupt();

    while (1)
    {
        if(GPIO_getInputPinValue(MOUSE_PORT,
                                 MOUSE_UP_PIN) == false)
        {
            mouseReport.dY += change_amount;
        }

        if(GPIO_getInputPinValue(MOUSE_PORT,
                                 MOUSE_DOWN_PIN) == false)
        {
            mouseReport.dY -= change_amount;
        }

        if(mouseReport.dY > 127)
        {
            mouseReport.dY = 127;
        }

        if(mouseReport.dY < -127)
        {
            mouseReport.dY = -127;
        }

        if(GPIO_getInputPinValue(MOUSE_PORT,
                                 MOUSE_LEFT_PIN) == false)
        {
            mouseReport.dX -= change_amount;
        }

        if(GPIO_getInputPinValue(MOUSE_PORT,
                                 MOUSE_RIGHT_PIN) == false)
        {
            mouseReport.dX += change_amount;
        }

        if(mouseReport.dX > 127)
        {
            mouseReport.dX = 127;
        }
```

```
        if(mouseReport.dX < -127)
        {
            mouseReport.dX = -127;
        }

        if(GPIO_getInputPinValue(MOUSE_PORT,
                                 MOUSE_BUTTON_PIN) == false)
        {
            mouseReport.buttons ^= 0x01;
        }

        switch (USB_getConnectionState())
        {
            case ST_ENUM_ACTIVE:
            {
                USBHID_sendReport((void *)&mouseReport,
                                  HID0_INTFNUM);

                print_C(2, 1, mouseReport.dX);
                print_C(8, 1, mouseReport.dY);
                print_C(14, 1, mouseReport.buttons);

                GPIO_toggleOutputOnPin(LED_PORT,
                                       LED_PIN);

                break;
            }

            case ST_PHYS_DISCONNECTED:
            case ST_ENUM_SUSPENDED:
            case ST_PHYS_CONNECTED_NOENUM_SUSP:
                break;

            case ST_ENUM_IN_PROGRESS:
            default:;
        }

        t++;
        if(t > 40)
        {
            t = 0;
            mouseReport.dX = 0;
            mouseReport.dY = 0;
            mouseReport.buttons = 0;
        }
    }
}


void DIO_init(void)
{
    GPIO_setAsInputPinWithPullUpResistor(MOUSE_PORT,
                                         MOUSE_UP_PIN);

    GPIO_setAsInputPinWithPullUpResistor(MOUSE_PORT,
                                         MOUSE_DOWN_PIN);

    GPIO_setAsInputPinWithPullUpResistor(MOUSE_PORT,
                                         MOUSE_LEFT_PIN);

    GPIO_setAsInputPinWithPullUpResistor(MOUSE_PORT,
                                         MOUSE_RIGHT_PIN);

    GPIO_setAsInputPinWithPullUpResistor(MOUSE_PORT,
                                         MOUSE_BUTTON_PIN);
}
```

Hardware Setup



Explanation

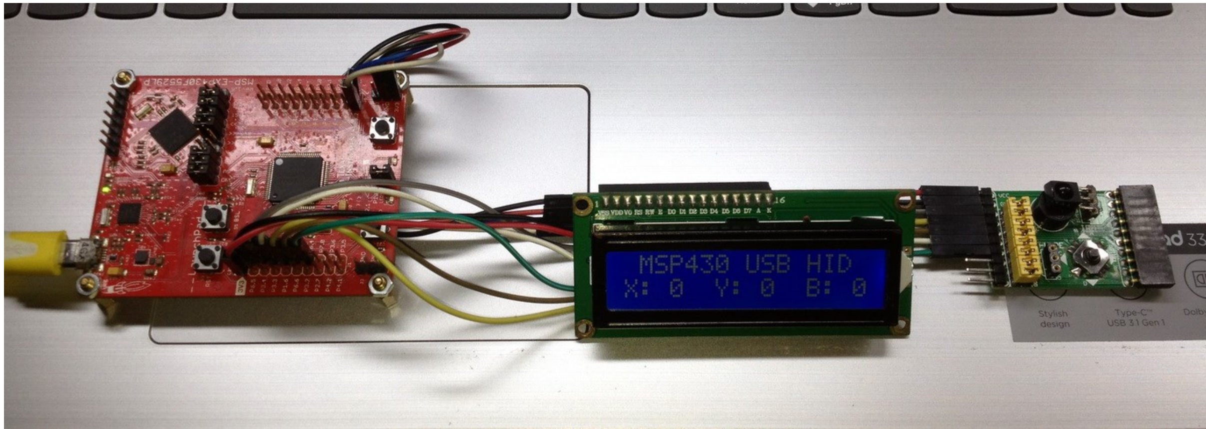
Just like the past example, again we are interested in the main loop as everything is done here. On each press of joystick buttons *x* and *y* coordinates are changed by 1 unit. We are not using mouse coordinate acceleration as with any regular mouse and that is why the change amount is 1 unit only. On PC screen, the mouse cursor will appear to move slowly as if the PC is busy doing some other works. We are also assuming that the *x* and *y* coordinate boundaries range from -127 to +127 units. In the code, we have to take care that these extremes are not exceeded. We are not using typical mouse click buttons although I coded the joystick middle button for clicking. This is not very important here as mouse cursor movement is all that is needed for USB HID demo. In the *ST_ENUM_ACTIVE* case, the location of cursor and button state are reported to host PC. The PC then uses these to change the position of the mouse cursor.

```
while (1)
{
    if(GPIO_getInputPinValue(MOUSE_PORT, MOUSE_UP_PIN) == false)
    {
        mouseReport.dY += change_amount;
    }

    if(GPIO_getInputPinValue(MOUSE_PORT, MOUSE_DOWN_PIN) == false)
    {
        mouseReport.dY -= change_amount;
    }

    if(mouseReport.dY > 127)
    {
        mouseReport.dY = 127;
```

```
    }

    if(mouseReport.dY < -127)
    {
        mouseReport.dY = -127;
    }

    if(GPIO_getInputPinValue(MOUSE_PORT, MOUSE_LEFT_PIN) == false)
    {
        mouseReport.dX -= change_amount;
    }

    if(GPIO_getInputPinValue(MOUSE_PORT, MOUSE_RIGHT_PIN) == false)
    {
        mouseReport.dX += change_amount;
    }

    if(mouseReport.dX > 127)
    {
        mouseReport.dX = 127;
    }

    if(mouseReport.dX < -127)
    {
        mouseReport.dX = -127;
    }

    if(GPIO_getInputPinValue(MOUSE_PORT, MOUSE_BUTTON_PIN) == false)
    {
        mouseReport.buttons ^= 0x01;
    }

    switch (USB_getConnectionState())
    {
        case ST_ENUM_ACTIVE:
        {
            USBHID_sendReport((void *)&mouseReport, HID0_INTFNUM);

            print_C(2, 1, mouseReport.dX);
            print_C(8, 1, mouseReport.dY);
            print_C(14, 1, mouseReport.buttons);

            GPIO_toggleOutputOnPin(LED_PORT, LED_PIN);

            break;
         }

         case ST_PHYS_DISCONNECTED:
         case ST_ENUM_SUSPENDED:
         case ST_PHYS_CONNECTED_NOENUM_SUSP:
             break;

         case ST_ENUM_IN_PROGRESS:
         default:;
     }

     t++;
     if(t > 40)
     {
         t = 0;
         mouseReport.dX = 0;
         mouseReport.dY = 0;
         mouseReport.buttons = 0;
      }
}
```

For 40 times, the mouse's coordinate changes and button states are not reset automatically and this is so because we don't want to keep holding the joystick buttons for movement once our desired speed is achieved. Remember no acceleration is used.

Demo



Demo video: https://youtu.be/JdS_LfKBhjk

Epilogue

My journey with MSP430F5529 has been a mixture of success and failure. It is successful in terms of being able to apply driverlib successfully while being able to master a mid-range high performance TI microcontroller. Most of the hardware is similar to other MSP430 micros and so I had little or no issues understanding them. This is for the first time I got to get myself introduced with a TI micro that has USB hardware as well as other advanced hardware like DMA, RTC, hardware multiplier, etc that are typically found in ARM microcontrollers. Previously, I have seen similar microcontrollers like PIC18F4550 and ATMega32U4 but none of them were as rich as MSP430F5529 when it comes to both hardware and software suites.

The failure part is the time it took for me to get all these things together because on the internet I didn't find much support for most of the stuffs I needed to know quickly. I had to read lot of docs and try out several things before I could make a solid conclusion. However, after completing this tutorial doc, I believe no one will have issues as I had. I also believe that it will pave way for developing stuffs with TI's Tiva C ARM microcontrollers as working with Tiva micros need knowledge of TivaWare – a software suite from TI that is similar to MSP430Ware.

To summarize, this MSP430 beast packs a good punch that can help us avoid investing on an ARM overkill or on other expensive platforms. It has everything to make devices like a datalogger, a USB power meter, an advanced MPPT solar charge controller, a balancing robot, etc. In fact, in my professional career, I have seen several control systems that host TI microcontrollers like the MSP430F5529. MSP430F5529 is like an ARM micro in 16-bit disguise. I would strongly urge readers to explore this microcontroller family and beyond more. Last but not least, TI microcontroller portfolio has microcontrollers for various application-specific purposed but compared to other manufacturers TI's microcontrollers are smart and well-designed in terms of hardware management. TI also manufactures other digital and analogue components. This allows us to design a complete board with all-TI components while making TI a good component partner.

The code examples of this document can be found here.

Happy coding.

Author: Shawon M. Shahryiar
https://www.youtube.com/user/sshahryiar
https://www.facebook.com/groups/microarena
https://www.facebook.com/MicroArena

29.01.2020