Seven Segment Library

2014-09-12

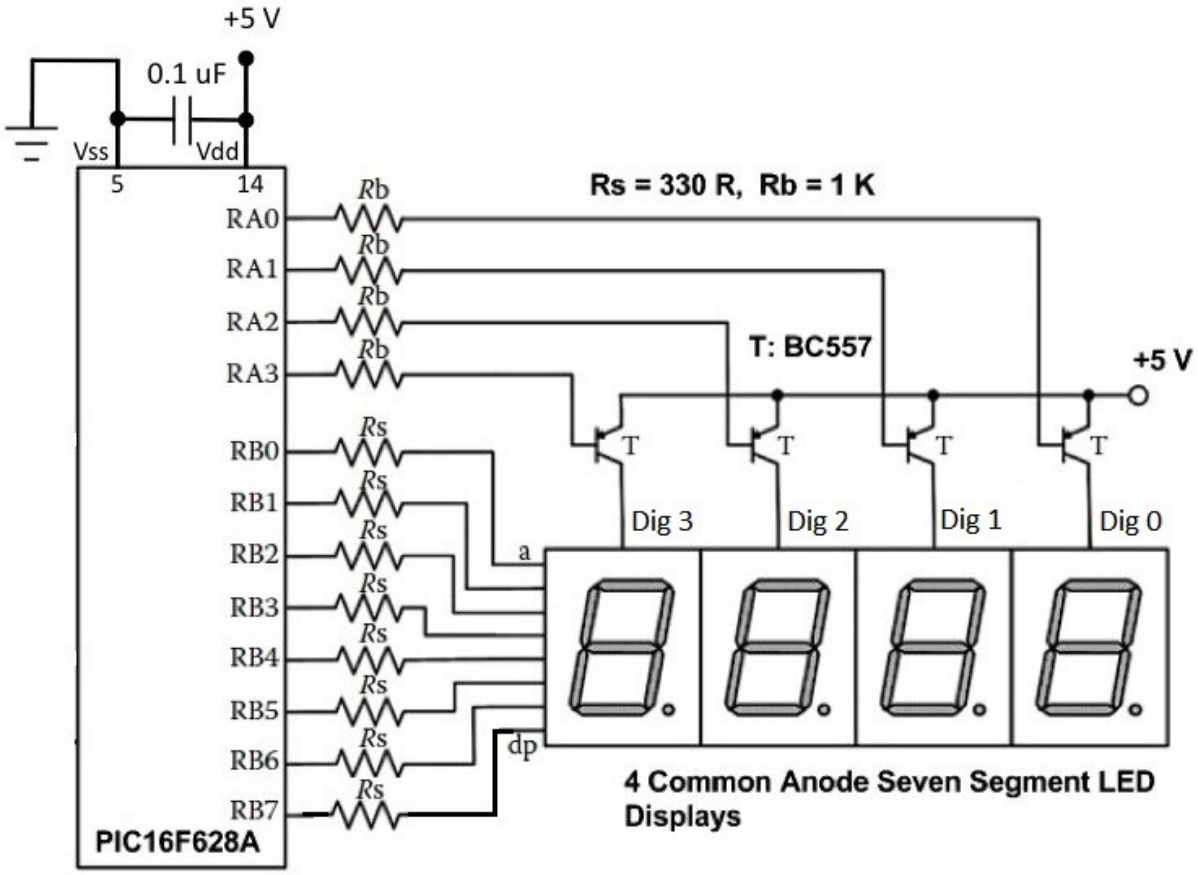
1 Contents

2 Characteristics

- 4 seven segment digits
- common anode 7 segment displays
- can display digits in range 0..9, A..F (so also hex values)
- can also display minus sign and a space
- sequential drive (scanning) of the 4 digits
- extra hardware (besides the PIC) needed: 4 transistors, 12 resistors

3 Circuit Diagram

The diagram shown below is for the “Seven_Segment.mpas” example provided together with the library:



As you can see, the digits (the common anodes) are switched on/off (sequentially) by the 4 transistors T, driven by PortA0.. PortA3, while the segments (the cathodes) are driven by PortB0..B7. All corresponding segments of the digits are connected with each other.

The values of Rs define the brightness of the 7 segments displays: the lower the brighter. Make sore to not exceed the maximum segment current or the maximum PIC output current.

4 The library usage

The following things have to be done in the program to use the library:

- The definition of “connections” variables (section Variables used by the library)
- The library initialisation (section Initialisation)
- The calling of the “Display” routine regularly (section The actual display routine)
- The definition of what each digit has to display either using
    - The raw library variables (section Defining the digit’s content), or
    - Using the easy high level routines (section High level Routines)

4.1 Variables used by the library

The library requires in the using program the definitions of the variables defining the PIC port connections to the 7 segment digits:

```
// Declaration of the bits that drive each Digit
var SSD_Dig0: sbit at PortA.0; // Use LATx.y for P18F etc... PIC's here
    SSD_Dig1: sbit at PortA.1;
    SSD_Dig2: sbit at PortA.2;
    SSD_Dig3: sbit at PortA.3;

    SSD_Dig0_Direction: sbit at TrisA.0;
    SSD_Dig1_Direction: sbit at TrisA.1;
    SSD_Dig2_Direction: sbit at TrisA.2;
    SSD_Dig3_Direction: sbit at TrisA.3;

// declaration of the port driving the segments
var SSD_Segments:           byte at PortB; // Use LATx.y for P18F etc... PIC's
    SSD_Segments_Direction: byte at TrisB;
```

See the circuit diagram: “SSD_Digx” define the connections (via a 1K resistor) to the base of the transistors (common anode drive), “SSD_Segments” defines the segments driving port.

The “SSD_Digx” connections do not have to reside on the same port, they can be defined on different ports. The “SSD_Segments” connections are always on the same port, they can not be spread over different ports.

4.2 Initialisation

Before usage the library has to be initialised with

```
SSD_Init;
```

This will “clear” all the digits and all decimal points (nothing lights up).

4.3 The actual display routine

Due to the "sequential" scanning nature of the 7 segment digits there is a tendency to flicker if the refresh frequency is too low. The overall scanning cycle should be no more than 20 millisecs, which means 5 milliseconds per digit here.

To achieve this the actual display routine has to every 5 milliseconds. The easiest way is to do this in a Timer interrupt routine.

This is the routine "SSD_Display".

Example of usage:

```
procedure interrupt[1];
begin
  if (TMR0IF_bit)[1] then
  begin
    TMR0IF_bit[1] := 0;
    TMR0[1]       := 100;  // 5 millisec

    //Enter your code here

    SSD_Display; // <--- Library routine call every 5 millisecs

  end;
end;
```

4.4 Defining a digit's content

To control what is displayed on the 4 seven segment digits a number of variables is available in the library:

```
var SSD_DigVals: array[0..3] of byte;
```

and

```
var SSD_DP: array[0..3] of boolean;
```

The contents of "SSD_DigVals" defines what digits are to be shown. It accepts the values 0..9 and a..f (the latter to display hex values), and also the values "SSD_SPACE" and "SSD_MINUS".
The contents of "SSD_DP" defines if a decimal point should light up. The values it accepts are false (off) and true (on).

[1] The routine is made for usage with Timer0 of the PIC16F628a at a 4 Mhz internal clock. To be adapted of course to the actual timer and clock speed used.

Example:

```
SSD_DigVals[0]:= 0; // will display "3210" provided
SSD_DigVals[1]:= 1; // digit 3 is the left one of course
SSD_DigVals[2]:= 2;
SSD_DigVals[3]:= 3;

SSD_DP[2] := not SSD_DP[2]; // will blink the decimal point of digit 2
```

As you can see this method and library interface is a very “raw” one. The user defines the contents of each digit and of each decimal point.
To make it more easy to display e.g. words values a number of routines have been added to fill SSD_DigVals, see next section.

4.5 High level Routines

These routines will fill “SDD_DigVals” for you to e.g. represent a word or a BCD value.

They are:

```
procedure SSD_Display_Word(Value: word; SuppressLeadingZeroes: boolean);
// Fill SSD_DigVals, Value is a 16 bits binary value (0..9999)

procedure SSD_Display_BCD_word(Value: word; SuppressLeadingZeroes: boolean);
// Fill SSD_DigVals, Value is a 16 bits BCD value (0..9999, each Dig 0..9)

procedure SSD_Display_Hex_word(Value: word);
// Fill SSD_DigVals, Value is a 16 bits binary value ($0..$ffff), output is hex

procedure SSD_Display_Short(Value: short; SuppressLeadingZeroes: boolean);
// Fill SSD_DigVals, Value is a "short" (-128..+127)
```

The comments above speaks for itself.
There are no high level routines to control the decimal points.

5 Library Interface

```
unit Seven_Segment_4D_CA;

// Routines to drive 4 Digits of 7 segments display, common anode

// All "segments" of the 4 digits (the cathodes) should be coupled together, and
each of those couplings should be connected to
// the PIC port "SDD_Segments" via a current limiting resistor (e.g. 270 Ohm).
// The anodes of the 4 digits should be connected to the collector of a BC557
transistor, of which the emitter is connected to Vcc.
// The base of the 4 transistors should be connected via a 4.7K Ohm resistor to the
"SSD_Digx" PIC outputs.


// -------------------------------------------------------------------------
// --------- Variables to be defined in the using program ------------------
// -------------------------------------------------------------------------

// Declaration of the bits that drive each Digit
var SSD_Dig0: sbit; sfr; external;
    SSD_Dig1: sbit; sfr; external;
    SSD_Dig2: sbit; sfr; external;
    SSD_Dig3: sbit; sfr; external;

    SSD_Dig0_Direction: sbit; sfr; external;
    SSD_Dig1_Direction: sbit; sfr; external;
    SSD_Dig2_Direction: sbit; sfr; external;
    SSD_Dig3_Direction: sbit; sfr; external;

// declaration of the port driving the segments
var SSD_Segments:           byte; sfr; external;
    SSD_Segments_Direction: byte; sfr; external;

const SSD_MINUS: byte = 16;
      SSD_SPACE: byte = 17;

// -------------------------------------------------------------------------
// ---------------- Basic Library Variables ------------------------------
// -------------------------------------------------------------------------

var SSD_DigVals: array[0..3] of byte;  // raw values to show (0..9, A..F, 16 (for'-
') or 17 for(' '))
// One can fill these variables with raw values or use one of the "Display"
routines below
// These "raw" values are translated to appropriate output on "SSD_Segments" by
"SSD_Display".

var SSD_DP: array[0..3] of boolean; // values (false = off / true = on) for the
decimal points


// -------------------------------------------------------------------------
// ---------------- Basic Library Routines -------------------------------
// -------------------------------------------------------------------------
procedure SSD_Init;
// Initialises the Seven Segment Display digits to ' ' (spaces)
```

```
procedure SSD_Display;
// Translates the "raw" values in "SSD_DigVals" to the correct output on
"SSD_Segments" (common anode codes)
// and does this for all digits (only one digit per call, each time the next digit
with wrap around)
// To be called -> regularly, preferably in an interrupt routine <- at least every
5 millisecs


// ---------------------------------------------------------------------------
// ---------------- Additional Library Routines ------------------------------
// ---------------------------------------------------------------------------

procedure SSD_Display_Word(Value: word; SuppressLeadingZeroes: boolean);
// Fill SSD_DigVals, Value is a 16 bits binary value (0..9999)

procedure SSD_Display_BCD_word(Value: word; SuppressLeadingZeroes: boolean);
// Fill SSD_DigVals, Value is a 16 bits BCD value (0..9999, each Dig 0..9)

procedure SSD_Display_Hex_word(Value: word);
// Fill SSD_DigVals, Value is a 16 bits binary value ($0..$ffff), output is hex

procedure SSD_Display_Short(Value: short; SuppressLeadingZeroes: boolean);
// Fill SSD_DigVals, Value is a "short" (-128..+127)


implementation
```

6 Examples of Usage

Two examples are provided together with the library (examples map):

- “Seven_Segment” shows how to use the raw library functions and the high level functions,
- “RTC_7_Segments” shows the display of an RTC (DS1307) hours and minutes, together with a blinking decimal point.

The circuit diagram (see Circuit Diagram) is for both examples but only shows the connections to the 4 seven segment digits.

[end of document]