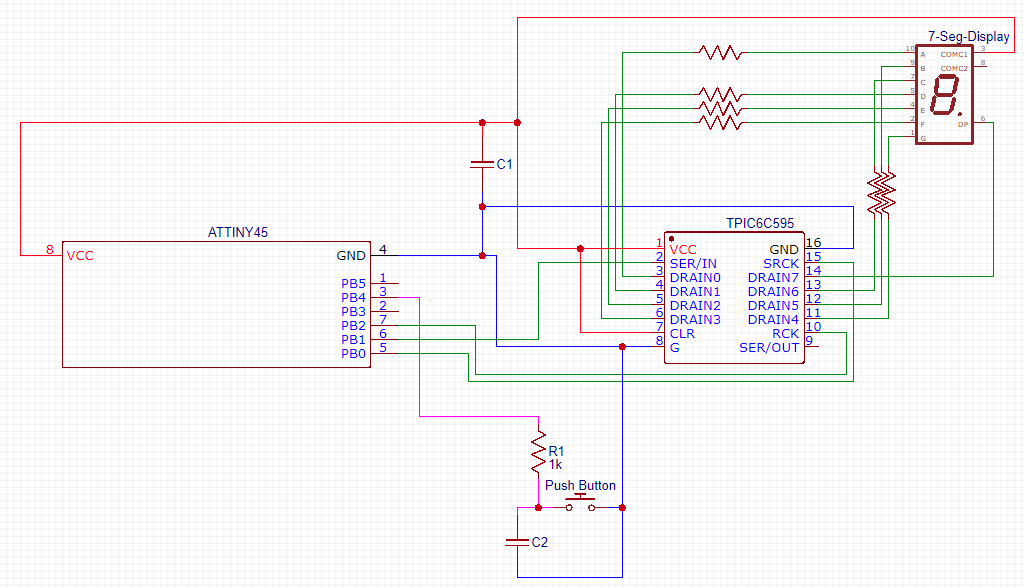
**Introduction**

The goal of this lab is to build a hexadecimal up/down counter using the ATtiny45 microcontroller, a 8-bit shift register, a 7-segment display, and a pushbutton. The counter will have 3 basic functions, count up or down, switch modes, and reset based on how long the pushbutton was held down for.

**Schematic**

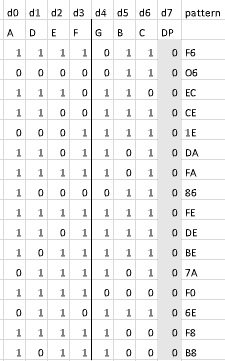


**Discussion**

The planning for this lab consisted of two phases. Phase 1 consisted of wiring the microcontroller and board according to the schematic labeled above. Afterwards, the board was connected to a PC and programmed to change the display of the LED by pressing the button. Phase 2 consisted of finishing the functionality required for the lab.

This final phase consisted of adding new modes, button functionality, and the ability to display all 16 hexadecimal values and a decimal point. To make the display work, we need to use appropriate current values. When looking at the datasheet for the Seven Segment Display, we calculated that the resistor value needed is around 300 Ohms.

The first new feature included adding features such as an increment and decrement mode. To access each mode, the user must press and hold down the button for one to two seconds. This was implemented this in the software program by creating a main loop which continuously checks to see if the button is pressed. From here the program awaits a button press to perform the necessary functionality required to complete the lab.

The functionality for the lab When the button is pressed the stack jumps to a function which creates another loop. In the loop we call a delay function which runs once every ten seconds. A separate register is incremented every after each delay function call. The value of the register is then compared to 10, and 20 to determine the functionality of the circuit. If the value is less than twenty and greater than ten a function titled “findPattern” is called and, dependent on which mode it is in, returns the next value. If it is greater than 20, then the LED light resets to 0.

To display each value the board we relied on using a hexadecimal pattern dependent on the drains and LED lights required to replicate each number. Because there is not enough I/O lines into the ATtiny45, we used a 8-bit shift register to expand the I/O of the microcontroller. By shifting in each value one at a time we can control the 7 Segment Display with fewer I/O pins. We shifted into the register all 8 bits required for the display according to the patterns on the left. We decided to create patterns with the decimal point off, then if we are in decrement mode we add 1 to the hex pattern to insure the decimal point turns on.

**Conclusion**

Our project can be described in three phases. In phase 1 we created the circuit schematic and implemented it onto the circuit board. Then the board was connected to a computer and programmed to change the LED display with a press of the button. Phase 2 included implementing the functionality required to complete the lab. This included adding new modes, button functionality, and the ability to display all 16 hexadecimal values and a decimal point.

This lab was useful to learn more about how to program in Assembly and work with complex circuitry. In this lab we learned how to connect a AtTiny45 microcontroller to an integrated circuit. This connects to a 7-segment LED display and push button. Later on, we learned more about the complexity of working with an Assembly program.

**Source Code**

;

; EmbeddedLab2.asm

;

; Created: 2/8/2019 10:27:17 AM

; Author : wnbrg

;

;R19 hold 0 when counting up and 1 when counting down

;R18 holds current counter

;R17 holds loop counter for SR

;R16 holds the pattern to display

; put code here to configure I/O lines

; connected to TPIC6C595 as output

sbi DDRB,0; SRCK

sbi DDRB,1; SERIN

sbi DDRB,2; RCK

cbi DDRB,3; Push Button

; start main program

ldi R18, 0

rcall resetTo0

mainloop:

sbis PINB,3

rcall push\_down

rjmp mainloop

push\_down:

.Def timer = r20

ldi timer, 0

timerLoop:

rcall delay

inc timer

sbic PINB, 3

rjmp timer\_break

rjmp timerLoop

timer\_break:

cpi timer, 10

brlo incOrDec

cpi timer, 20

brlo switch\_mode

brsh resetTo0

ret

switch\_mode:

tst R19

breq if\_start4

ldi R19, 0

rjmp end\_if4

if\_start4:

ldi R19, 1

end\_if4:

rcall findPattern

ret

incOrDec:

tst R19

breq if\_start5

rcall count\_down

rjmp end\_if5

if\_start5:

rcall count\_up

end\_if5:

ret

count\_up:

cpi R18,0x0F

breq if\_start2

inc R18

rjmp if\_end2

if\_start2:

rcall resetTo0

if\_end2:

rcall findPattern

ret

count\_down:

cpi R18,0x00

breq if\_start3

dec R18

rjmp if\_end3

if\_start3:

rcall resetToF

if\_end3:

rcall findPattern

ret

resetTo0:

ldi r19, 0

ldi r18, 0

rcall findPattern

ret

resetToF:

ldi r18, 0x0F

rcall findPattern

ret

display:

; see if you should add one to display D.P. (currently decrementing)

tst R19

brne if\_start1

rjmp end\_if1

if\_start1:

inc R16

end\_if1:

; backup used registers on stack

push R16

push R17

in R17, SREG

push R17

ldi R17, 8 ; loop --> test all 8 bits

loop:

rol R16 ; rotate left trough Carry

BRCS set\_ser\_in\_1 ; branch if Carry set

; put code here to set SER\_IN to 0

cbi PORTB,1

rjmp end

set\_ser\_in\_1:

; put code here to set SER\_IN to 1

sbi PORTB,1

end:

; put code here to generate SRCK pulse

sbi PORTB,0

nop

nop

nop

cbi PORTB,0

dec R17

brne loop

; put code here to generate RCK pulse

sbi PORTB,2

nop

nop

nop

cbi PORTB,2

; restore registers from stack

pop R17

out SREG, R17

pop R17

pop R16

ret

disp0:

ldi R16, 0xF6 ; load pattern to display

rcall display ; call display subroutine

ret

disp1:

ldi R16, 0x06 ; load pattern to display

rcall display ; call display subroutine

ret

disp2:

ldi R16, 0xEC ; load pattern to display

rcall display ; call display subroutine

ret

disp3:

ldi R16, 0xCE ; load pattern to display

rcall display ; call display subroutine

ret

disp4:

ldi R16, 0x1E ; load pattern to display

rcall display ; call display subroutine

ret

disp5:

ldi R16, 0xDA ; load pattern to display

rcall display ; call display subroutine

ret

disp6:

ldi R16, 0xFA ; load pattern to display

rcall display ; call display subroutine

ret

disp7:

ldi R16, 0x86 ; load pattern to display

rcall display ; call display subroutine

ret

disp8:

ldi R16, 0xFE ; load pattern to display

rcall display ; call display subroutine

ret

disp9:

ldi R16, 0xDE ; load pattern to display

rcall display ; call display subroutine

ret

dispA:

ldi R16, 0xBE ; load pattern to display

rcall display ; call display subroutine

ret

dispB:

ldi R16, 0x7A ; load pattern to display

rcall display ; call display subroutine

ret

dispC:

ldi R16, 0xF0 ; load pattern to display

rcall display ; call display subroutine

ret

dispD:

ldi R16, 0x6E ; load pattern to display

rcall display ; call display subroutine

ret

dispE:

ldi R16, 0xF8 ; load pattern to display

rcall display ; call display subroutine

ret

dispF:

ldi R16, 0xB8 ; load pattern to display

rcall display ; call display subroutine

ret

findPattern: ;R15 holds the number we need to display, find the pattern

cpi R18,0

breq disp0

cpi R18,1

breq disp1

cpi R18,2

breq disp2

cpi R18,3

breq disp3

cpi R18,4

breq disp4

cpi R18,5

breq disp5

cpi R18,6

breq disp6

cpi R18,7

breq disp7

cpi R18,8

breq disp8

cpi R18,9

breq disp9

cpi R18,10

breq dispA

cpi R18,11

breq dispB

cpi R18,12

breq dispC

cpi R18,13

breq dispD

cpi R18,14

breq dispE

cpi R18,15

breq dispF

delay: ;Delays for 1/10 of a sec

ldi r23,4 ; r23 <-- Counter for outer loop

d1: ldi r24,93 ; r24 <-- Counter for level 2 loop

d2: ldi r25,53 ; r25 <-- Counter for inner loop

d3: dec r25

nop ; no operation

brne d3

dec r24

brne d2

dec r23

brne d1

ret