Terence Henriod

HW 6

CPE 301: Microprocessors System Design

October 8, 2013

1. Answer Question 6.1 as modified to read as follows:
   1. Using the ATmega328P pin configuration table and the Arduino Duemilanove Schematic in Appendix A, identify in a new Table all of the pins on the three ports (PB, PC, and PD) that are already dedicated to an alternative function via pin-muxing.
   2. Make an equivalent Table for the ATmega2560 used in the Arduino Mega SBC for the equivalent alternative functions (for example, we found that the LED is connected to PB5 on the Arduino Duemilanove and PB7 on the Arduino Mega).
   3. Hint: all the pins in the table correspond to pins on the schematic; the pins to be identified will go to specific circuitry other than the headers. The schematic for the Arduino Mega is posted on WebCampus.

See attached Excel sheets for this “solution.”

1. Write a C function that will take in a char variable and output an appropriate 7-segment set of signals to display the associated alphabet (both upper and lower case) and numeric character. (This includes the setup() and loop() functions as specified in the book)

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

This function's intended use is to accept a byte (represented as an int) that

will represent hex values 0-F, and the switch statement will configure output

port K such that it will appropriately light a 7-segment display to display

the given hex digit.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

void drive7Segment( volatile unsigned int input );

void setup()

{

// variables

volatile unsigned char\* portDDRB;

portDDRB = (unsigned char\*) 0x24;

volatile unsigned char\* portDDRK;

portDDRK = (unsigned char\*) 0x107;

volatile unsigned char\* portK;

portK = (unsigned char\*) 0x108;

// set DDRs

// using portB bits 0-3 as input (0-3 low)

\*portDDRB &= 0x00;

// using portK bits 0-7 as output (0-7 high)

\*portDDRK |= 0x7F;

// set output (portK 0-7) all high

\*portK |= 0x7F;

}

void loop()

{

// variables

volatile unsigned int\* portB;

portB = (unsigned int\*) 0x25;

volatile unsigned int\* portK;

portK = (unsigned int\*) 0x108;

volatile unsigned int previous;

volatile unsigned char charValue = (unsigned char) 0x00;

// loop through the hex digits to display functionality

for(charValue = (unsigned char) 0x00; charValue <= (unsigned char) 0x0F; charValue++)

{

// display the char on the 7 segment display

drive7Segment( charValue );

// let digit display for a while

myDelay( HALF\_SEC );

}

// if digit F is reached, restart at 0

if( charValue > (unsigned char)0x0F )

{

// reset charValue to 0

charValue = (unsigned char) 0x00;

}

// no return - void - continue loop operation

}

void drive7Segment( volatile unsigned char input )

{

// variables

volatile unsigned int\* portK;

portK = (unsigned int\*) 0x108;

// given a character, display different behavior

switch( input )

{

case 0x00:

// display a 0

// set segment a high

// set segment b high

// set segment c high

// set segment d high

// set segment e high

// set segment f high

// (0011 1111)

\*portK |= 0x3F;

// set segment g low

// (1011 1111)

\*portK &= 0xBF;

break;

case 0x01:

// display a 1

// set segment b high

// set segment c high

// (0000 0110)

\*portK |= 0x06;

// set segment a low

// set segment d low

// set segment e low

// set segment f low

// set segment g low

// (1000 0110)

\*portK &= 0x86;

break;

case 0x02:

// display a 2

// set segment a high

// set segment b high

// set segment d high

// set segment e high

// set segment g high

// (0101 1011)

\*portK |= 0x5B;

// set segment c low

// set segment f low

// (1101 1011)

\*portK &= 0xDB;

break;

case 0x03:

// display a 3

// set segment a high

// set segment b high

// set segment c high

// set segment d high

// set segment g high

// (0100 1111)

\*portK |= 0x4F;

// set segment e low

// set segment f low

// (1100 1111)

\*portK &= 0xCF;

break;

case 0x04:

// display a 4

// set segment b high

// set segment c high

// set segment f high

// set segment g high

// (0110 0110)

\*portK |= 0x66;

// set segment a low

// set segment d low

// set segment e low

// (1110 0110)

\*portK &= 0xE6;

break;

case 0x05:

// display a 5

// set segment a high

// set segment c high

// set segment d high

// set segment f high

// set segment g high

// (0110 1101)

\*portK |= 0x6D;

// set segment b low

// set segment e low

// (1110 1101)

\*portK &= 0xED;

break;

case 0x06:

// display a 6

// set segment a high

// set segment c high

// set segment d high

// set segment e high

// set segment f high

// set segment g high

// (0111 1101)

\*portK |= 0x7D;

// set segment b low

// (1111 1101)

\*portK &= 0xFD; break;

case 0x07:

// display a 7

// set segment a high

// set segment b high

// set segment c high

// (0000 0111)

\*portK |= 0x07;

// set segment d low

// set segment e low

// set segment f low

// set segment g low

// (1000 0111)

\*portK &= 0x87;

break;

case 0x08:

// display an 8

// set segment a high

// set segment b high

// set segment c high

// set segment d high

// set segment e high

// set segment f high

// set segment g high

// (0111 1111)

\*portK |= 0x7F;

break;

case 0x09:

// display a 9

// set segment a high

// set segment b high

// set segment c high

// set segment f high

// set segment g high

// (0110 0111)

\*portK |= 0x67;

// set segment d low

// set segment e low

// (1110 0111)

\*portK &= 0xE7;

break;

case 0x0A:

// display an A

// set segment a high

// set segment b high

// set segment c high

// set segment e high

// set segment f high

// set segment g high

// (0111 0111)

\*portK |= 0x77;

// set segment d low

// (1111 0111)

\*portK &= 0xF7;

break;

case 0x0B:

// display a b

// set segment c high

// set segment d high

// set segment e high

// set segment f high

// set segment g high

// (0111 1100)

\*portK |= 0x7C;

// set segment a low

// set segment b low

// (1111 1100)

\*portK &= 0xFC;

break;

case 0x0C:

// display a C

// set segment a high

// set segment d high

// set segment e high

// set segment f high

// (0011 1001)

\*portK |= 0x39;

// set segment b low

// set segment c low

// set segment g low

// (1011 1001)

\*portK &= 0xB9;

break;

case 0x0D:

// display a d

// set segment b high

// set segment c high

// set segment d high

// set segment e high

// set segment g high

// (0101 1110)

\*portK |= 0x5E;

// set segment a low

// set segment f low

// (1101 1110)

\*portK &= 0xDE;

break;

case 0x0E:

// display an E

// set segment a high

// set segment d high

// set segment e high

// set segment f high

// set segment g high

// (0111 1001)

\*portK |= 0x79;

// set segment b low

// set segment c low

// (1111 1001)

\*portK &= 0xF9;

break;

case 0x0F:

// display a F

// set segment a high

// set segment e high

// set segment f high

// set segment g high

// (0111 0001)

\*portK |= 0x71;

// set segment b low

// set segment c low

// set segment d low

// (1111 0001)

\*portK &= 0xF1;

break;

}

// no return - void

}

1. Because many microcontrollers operate in noisy environments, the first detection of a signal-transition cannot always be trusted (i.e., it might be a glitch on the line). Additionally, many mechanical switches are notorious for *bouncing* when they are first pressed or released. So, you are to write a function that *debounces* an input pin. Your function should be called whenever the first signal-event occurs on an input pin, and it should debounce for a reasonable amount of time (e.g., 20 msec). At the end of the debounce time, the input should be re-sampled and compared to the original state of the line; if they are different, you should declare that a valid transition has occured.

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

This function's intended use is to be called once a change in a port signal is

detected. The function then waits for 20 msec before re-polling, and if the

differing inout is still present, the previous value is updated and returned

by reference.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

unsigned int myDebounce( volatile unsigned int\* inPort,

volatile unsigned int previousVal );

unsigned int myDebounce( volatile unsigned int\* inPort,

volatile unsigned int& previousVal )

{

// delay to let any bounce settle

myDelay( DEBOUNCE );

// check to see if the signal is still different from before

if( \*inPort != previousVal )

{

// if it is, accept the value, update the \*now\* previous value

previousVal = \*inPort;

}

// return the new previous value for use

return previousVal;

}

1. Write a program that monitors and debounces an external push-button device. Do something interesting when the user presses a button (e.g., write a message to the terminal using the serial library or use GPIO output pins to light up an LED, etc.). Your program should continuously monitor the push-button switch and respond whenever it is pressed or released after debouncing, of course.

void loop()

{

// variables

volatile unsigned int\* portB;

portB = (unsigned int\*) 0x25;

volatile unsigned int\* portK;

portK = (unsigned int\*) 0x108;

volatile unsigned int previous;

volatile unsigned char charValue = (unsigned char) 0x00;

// loop through the hex digits to display functionality

for(charValue = (unsigned char) 0x00; charValue <= (unsigned char) 0x0F; charValue++)

{

/\*

This addition is more or less equivalent to myDebounce, but gives more freedom

to add functionality to the button press detection (were it not for the

specification of 6.3, I would have written myDebounce in this manner). Interestingly,

the “counting” reset will occur with either the push OR release of the button.

\*/

// check for push button input

if( \*portB != previous )

{

// delay to let the bouncing taper

myDelay( DEBOUNCE );

// check again for the changed input

if( \*portB != previous )

{

// update the \*now\* previous value

previous = \*portB;

charValue = (unsigned char) 0x00;

}

}

// display the char on the 7 segment display

drive7Segment( charValue );

// let digit display for a while

myDelay( HALF\_SEC );

}

// if digit F is reached, restart at 0

if( charValue > (unsigned char)0x0F )

{

// reset charValue to 0

charValue = (unsigned char) 0x00;

}

// no return - void - continue loop operation

}