Patrick Austin

CPE 301 - 1104, Fall 2016

Homework 7

10/31/2016

8.1. -Use pre-scale division of 16

Set ADSCRA bit 0 to 0, bit 1 to 0, bit 2 to 1.

-Use the internal 5V VREF

Set ADMUX bit 6 to 0, bit 7 to 1.

-Use right justification

Set ADMUX bit 5 to 0.

-Use ADC channel 5.

Set ADMUX bit 0 to 1, bit 1 to 0, bit 2 to 1, bit 3 to 0.

These values were determined from reading textbook chapter 8 and looking at the register descriptions.

8.2

This function is included in problem 8.4 and is shown compiling there.

void myADCInit() //Enable ADC channel 5 with properties described below

{

\*myADSCRA = 0b10010100; //enable but do not start conversion on the ADC.

//disable auto trigger, clear enable flag

//disable interrupt, select 16 pre-scaler

\*myADSCRB = 0b01000001; //prevent ADC switch off, set analog comparator

//set reserved values low

\*myADMUX = 0b01000101; //select 5V internal Vref, right justify, use

//ADC channel 5. set reserved value low

\*myDIDR0 = 0b00111111; //set reserved values low, use ADC channel 5

//write other values high to reduce power consumption

}

8.4

//Patrick Austin

//CPE 301 HW 7 Problem 8.4

//Revision Number 1

//Revision date: 10/30/2016

//hardware pointer declarations

//output

volatile unsigned char\* myPortDDRK = (unsigned char\*) 0x107;

volatile unsigned char\* myPortK = (unsigned char\*) 0x108;

//timer

volatile unsigned char\* myTIFR1 = (unsigned char\*) 0x36;

volatile unsigned char\* myTCC1A = (unsigned char\*) 0x80;

volatile unsigned char\* myTCC1B = (unsigned char\*) 0x81;

volatile unsigned char\* myTCC1C = (unsigned char\*) 0x82;

volatile unsigned int\* myTCNT1 = (unsigned int\*) 0x84;

volatile unsigned char\* myTIMSK1 = (unsigned char\*) 0x6F;

//ADC

volatile unsigned char\* myADSCRA = (unsigned char\*) 0x7A;

volatile unsigned char\* myADSCRB = (unsigned char\*) 0x7B;

volatile unsigned char\* myADMUX = (unsigned char\*) 0x7C;

volatile unsigned char\* myDIDR0 = (unsigned char\*) 0x7E;

volatile unsigned int\* myADCDR = (unsigned int\*) 0x78;

/\* Lookup table for seven segment values: such that

segmentvalues[x] contains the segments to light

to display x. convention: bit 0 (rightmost) specifies

whether to light segment a, bit 1 b, ..., bit 6 f. Bit 7

is for the decimal point and is unused in this program,

so it is always set to 0. \*/

static const unsigned char segmentValues[] =

{ 0b00111111, 0b00000110, 0b01011011, 0b01001111,

0b01100110, 0b01101101, 0b01111101, 0b00000111,

0b01111111, 0b01100111, 0b01110111, 0b01111100,

0b00111001, 0b01011110, 0b01111001, 0b01110001 };

//function prototypes

void myADCInit();

unsigned int readADC();

void myDelay(unsigned long int mSeconds);

//function implementations

void setup() //set up output on portK, enable timer 1, and initialize the ADC

{

\*myPortDDRK = 0xFF; //enable all bits on port K for output to seven segment LED

\*myTCC1A = 0; //enable timer 1

\*myTCC1B = 0;

\*myTCC1C = 0;

\*myTIMSK1 = 0;

myADCInit(); //enable ADC

}

void loop() //take a voltage reading, convert it to a time to delay, show a LED value

{ //for a period proportional to the voltage reading (higher V -> more time), repeat

//variables

unsigned int timeToDelay;

static unsigned char currentHex = 0x00; //static for persistence between loops

unsigned int input;

input = readADC(); //get a value from the ADC

timeToDelay = ( (input \* (500/1023) ) + 100); //scale the ADC's reading to an int between 0 and 500

//where 500 corresponds to a 5V reading, 250 to

//a 2.5V reading, etc. Then add 100. This will be

//the number of ms that the segments will display for.

//ie if 5V wait 600 ms, 2.5V wait 350 ms, 0V wait 100ms

if ( currentHex > 0x0F ) //check if currentHex has exceeded value 0x0F

{ currentHex = 0x00; } //if so, set it back to 0x00

\*myPortK = segmentValues[currentHex]; //send the appropriate segments to the LED

currentHex++; //increment currentHex to the next hex value

myDelay(timeToDelay); //and then delay as appropriate for the voltage read

} //then repeat for a new voltage reading...

//function implementations

void myADCInit() //Enable ADC channel 5 with properties described below. Also used in problem 8.2

{

\*myADSCRA = 0b10010100; //enable but do not start conversion on the ADC.

//disable auto trigger, clear enable flag

//disable interrupt, select 16 pre-scaler

\*myADSCRB = 0b01000001; //prevent ADC switch off, set analog comparator

//set reserved values low

\*myADMUX = 0b01000101; //select 5V internal Vref, right justify, use

//ADC channel 5. set reserved value low

\*myDIDR0 = 0b00111111; //set reserved values low, use ADC channel 5

//write other values high to reduce power consumption

}

unsigned int readADC() //return the value from an ADC conversion, in form on an unsigned int between 0 and 1023

{

unsigned int result;

\*myADSCRA |= 0x40; //start ADC conversion

while ( (\*myADSCRA & 0x10) == 0 ) //wait for conversion complete flag to be raised

{}

result = \*myADCDR & 0x03FF; //get ADC value, masking off bits not used in the data register

\*myADSCRA = 0b10010111; //turn off ADC by restoring initial ADSCRA value

return result;

}

void myDelay(unsigned long int mSeconds) //use timer 1 to delay for mSeconds ms.

{

\*myTCC1B = \*myTCC1B & 0xF8; //set timer to off

//calculate preload value for the timer. need preload value such that the time needed for the

//timer to go from the preload value to raising the overflow flag will take 'mSeconds' ms.

//this implementation uses the 1024 prescaler.

\*myTCNT1 = (unsigned int) (65536 - (long) (15.625 \* mSeconds)); //get preload value

\*myTCC1B = \*myTCC1B | 0x05; //enable timer with 1024 prescaler

while ( (\*myTIFR1 & 0x01) == 0 ) //until the overflow flag is raised, do nothing

{}

\*myTCC1B = 0; //delay complete, turn off the timer

\*myTIFR1 = \*myTIFR1 | 0x01; //reset the oveflow flag by writing a 1, finished

}

Shown compiling here:

