Terence Henriod

HW 10

CPE 301: Microprocessors System Design

November 5, 2013

1. 8.1: Determine the appropriate bit settings for ADCSRA, ADSCRB, ADMUX, and DIDR0 to read an analog input using the following specific details. Explain how you determined the register values.
   1. use a pre-scale division of 16,
   2. use the internal 5V VREF,
   3. use right justification,
   4. use ADC channel 5.

/\* See Exercise 2 for the appropriate bit settings as described in the documentation.

All values and addresses were determined using the textbook and the given data

sheets using careful selection of appropriate values.

\*/

1. 8.2: Create a function that initializes the ADC sub-system based on the values determined in problem 8.1.

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

Global Variables (Yes, these are global!)

Mostly used for register pointers that can't be declared constant.

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

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\* myADC\_HI = (unsigned char\*) 0x79;

volatile unsigned int\* myADC\_LO = (unsigned char\*) 0x78;

void initializeADC()

{

// variables

// none

// initialize the registers to their appropriate values

\*myADSCRA = 0b10010100;

// 10010 to enable, not start, disable auto trigger, clear the enable

// flag (1), and disable the interrupt

// 100 to select a pre-scalar of 16

\*myADSCRB = 0b01000000;

// 1 to prevent switching the ADC off

// 001 to set the analog comparator

\*myADMUX = 0b01000101;

// 01 to select the internal 5V Vref

// 0 to right justify

// 0 reserved

// 0101 to select ADC channel 5

\*myDIDR0 = 0b00100000;

// 00 to ensure future compatibility (these are reserved)

// 111111 to disable the ADC channel 5 digital (and all others to

// reduce power consumption). Bit 5 is for ADC 5

// no return - void

};

1. 8.4: Write a program that polls ADC channel 5 for the analog value of the sensor set up in problem 8.3. Based on the value, create a series of LED patterns on a 7 segment display, and adjust the rate at which the pattern changes based on the analog input value.

NOTE: My interpretation of the 7 segment display changes which are requested in 8.4 is: a) pick some pattern of segments (for example 0 through F) and b) cycle through these patterns and as the analog voltage-in increases cycle faster and faster (or slower and slower) - your choice.

8.3: Use trim-pot (or any other reasonable analog sensing device) as the input to ADC channel 5. For a simple example, see Figure 8.6.

/\*=============================================================================

Creator: Terence Henriod

Course: CPE301

Section: 1101

Program Name: HW10 ADC

Description: This program demonstrates the use of the Arduino ADC by changing

the rate at which hex digits are cycled on a 7-segment display.

Revision #: v0.01

Date: 11/5/2013

THIS CODE MAY BE ASSUMED TO BE WRITTEN UNDER THE GNU PUBLIC LISCENCE AND IS

THEREFORE FREE. FREE DOES NOT NECESSARILY MEAN FREE OF COST, BUT FREE FOR

DISTRIBUTION AND MODIFICATION. HOWEVER, THE AUTHOR OF THIS CODE MUST BE CITED

WHEN THIS CODE IS USED, DISTRIBUTED, OR MODIFIED.

==============================================================================\*/

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

Global Variables (Yes, these are global!)

Mostly used for register pointers that can't be declared constant.

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

// output register

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

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

// timer registers

volatile unsigned char\* tcc1A = (unsigned char\*) 0x80; // timer/counter1 control A

volatile unsigned char\* tcc1B = (unsigned char\*) 0x81; // timer/counter1 control B

volatile unsigned char\* tcc1C = (unsigned char\*) 0x82; // timer/counter1 control C

// tcRegisters replaces the two above at the same time

volatile unsigned int\* tcRegisters = (unsigned int\*) 0x84; // uns int assumed 16 bit

volatile unsigned char\* tcnt1H = (unsigned char\*) 0x85; // more significant, 15:8

volatile unsigned char\* tcnt1L = (unsigned char\*) 0x84; // 7:0

volatile unsigned char\* timsk1 = (unsigned char\*) 0x6F; // interrupt mask register

volatile unsigned char\* tc1Flag = (unsigned char\*) 0x36; // timer/counter interrupt flag register

// ADC registers

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 char\* myADC\_HI = (unsigned char\*) 0x79;

volatile unsigned char\* myADC\_LO = (unsigned char\*) 0x78;

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

Constants

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

const int INPUT\_CLASS = 10;

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

Function Prototypes

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

void initializeADC5();

void initializeTimer1();

volatile unsigned int pollADC();

void drive7Segment( volatile unsigned int input );

void MyDelay( unsigned long int mSeconds );

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

Setup

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

void setup()

{

// setup the output for he 7 segment display

\*portDDRK = 0x00; // start at 0

\*portDDRK = 0xFF; // all bits high for output

// initialize the Analog to Digital Converter channel 5

initializeADC5();

// intitialize timer 1 for normal mode and to be appropriate for millisecond

// timing

initializeTimer1();

// no return - void

}

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

Loop

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

void loop()

{

// variables

volatile unsigned int inputVal;

static volatile unsigned int oldInput;

static volatile unsigned int wait = 500;

volatile unsigned char hexDigit = 0x00;

// poll for data

inputVal = pollADC();

// convert the data to an appropriate wait time if necessary

// I did the if just for "funsies"

if( inputVal != oldInput )

{

// divides input into 10 classes and makes them "easy" fractions of seconds

wait = ( ( inputVal / INPUT\_CLASS ) + 100 );

}

// loop through the hex digits at the set speed

for( hexDigit = 0; hexDigit <= 0x0F; hexDigit++ )

{

// output the hex digit

drive7Segment( hexDigit );

// let digit display for specified time

MyDelay( wait );

}

// end of loop function - restart from main

}

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

Function Implementations

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

void initializeADC5()

{

// variables

// none

// initialize the registers to their appropriate values

\*myADSCRA = 0b10010100;

// 10010 to enable, not start, disable auto trigger, clear the enable

// flag (1), and disable the interrupt

// 100 to select a pre-scalar of 16

\*myADSCRB = 0b01000000;

// 1 to prevent switching the ADC off

// 001 to set the analog comparator

\*myADMUX = 0b01000101;

// 01 to select the internal 5V Vref

// 0 to right justify

// 0 reserved

// 0101 to select ADC channel 5

\*myDIDR0 = 0b00100000;

// 00 to ensure future compatibility (these are reserved)

// 111111 to disable the ADC channel 5 digital (and all others to

// reduce power consumption). Bit 5 is for ADC 5

// no return - void

};

void initializeTimer1()

{

// variables

// none

// set timer to normal mode

\*tcc1A = 0; // all bits 0, especially WGM11 and WGM10

// continue to set timer to normal mode

\*tcc1B = 0; // all bits to 0, especially WGM13 and WGM11

// and CS2:0 for clock stop

\*tcc1C = 0; // ensure register flags are cleared

\*timsk1 = 0; // ensure these are cleared too

\*tc1Flag |= 0x01; // write 1 to clear flag (seriously)

// no return - void

}

volatile unsigned int pollADC()

{

// variables

volatile unsigned int result = 0;

volatile unsigned char lowReg;

volatile unsigned char highReg;

// start the ADC conversion

\*myADSCRA |= 0x40; // write 1 to bit 6

// wait for the conversion to complete

while( ( \*myADSCRA & 0x10 ) == 0 ) // bit 4 is conversion complete flag

{};

// sample the conversion value, sample the high register first or the next

// conversion will start

lowReg = \*myADC\_LO;

highReg = \*myADC\_HI;

// place the values in the result

result |= highReg;

result <<= 8;

result |= highReg;

// turn the ADC off and clear the conversion complete flag

\*myADSCRA = 0b10010100; // same value as in initialize function

// return result

return result;

}

void drive7Segment( volatile unsigned char input )

{

// variables

// none

// 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 an 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

}

void MyDelay( unsigned long int mSeconds )

{

// variables

volatile unsigned int timerStartVal = 0;

volatile unsigned int countupVal = 0;

// ensure the clock is stopped

// set the clock select bits for clock stopping

// i.e. CS02, CS01, CS00 to (000)

\*tcc1B &= 0xF8; // (1111 1000)

// calculate the number of ticks we need to wait

/\* This means we need to figure out how many ticks are necessary.

Assuming the frequency of the clock in the Arduino is 16.000 MHz,

this means that we get 16,000,000 ticks per second, 16 ticks

per millisecond.

If we use a pre-scale factor of 1024, then we perceive the timer

to oscillate at 15625 Hz. Invert this to find that every tick is then

perceived to occur every 6.4 E-5 seconds.

Now, we need to find a constant that will be correspond to the number

of ticks per millisecond. This is found by dividing 1 miliisecond

by the time it takes for a tick. This is 15.625. I will "ceiling" this

number (if necessary) and say that there are 16 ticks per millisecond. (10h)

\*/

// get countup value

countupVal = (unsigned int) ( mSeconds \* 15.625);

// calculate timer start value (MAX + 1 - countupVal)

timerStartVal = (unsigned int) (65536 - (long) countupVal);

// setup for timing

// store the calculated timer start value

//TODO: Use the 16 bit type defined in the header file

\*tcRegisters = timerStartVal; // 124? WTF?

// clear the TOV flag

\*tc1Flag |= 0x01;

// start timing

// start the timer to count in the 1024 pre-scale

// (set CS2, CS1, CS0 to 101 (XXXX X101) )

\*tcc1B = 0x05;

// let TCNT1 count up to max (that is, TOV1 flag gets set)

while( (\*tc1Flag & 0x01) == 0x00 )

{

}

// clean up - stop counter

// set the clock select bits for clock stopping

// i.e. CS02, CS01, CS00 to (000)

\*tcc1B = 0; // (1111 1000)

// clear the TOV1 flag

\*tc1Flag |= 0x01;

// no return - void

}