#include "xc.h"

#include "my header.h"

#include "pins.h"

#include "functions.h"

#include "config.h"

#define VersionNmbr 100

#define SubVersion ' '

#define autoReInitialize

\_2Bytes integer;

volatile FLAGS\_BYTE myflags, switch\_flags, lcd\_flags, flags0;

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#define \_TMR4IF PIR5bits.TMR4IF

#define \_TMR4ON T4CONbits.TMR4ON

#define \_ADIF PIR1bits.ADIF

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////// FOR LCD FUNCTIONS ///////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#define FunSet 0x28

#define EntryMode 0x06

#define ClearDisplay 0x01

#define DisplayON 0x0c

#define DisplayOFF 0x08

#define DisplayGap 200 // determines the gap between the display of 2 parameters

#define displayFull lcd\_flags.B2

#define lcdInitialised lcd\_flags.B3

#define normalRun lcd\_flags.B4

#define messageEnd lcd\_flags.B5

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#define flag10mS myflags.B0

#define setupMode myflags.B1

#define flag1Sec myflags.B2

#define testflag myflags.B3

#define timeOut myflags.B4

#define \_sensorPin1 myflags.B5

#define \_sensorPin2 myflags.B6

#define countChange myflags.B7

#define \_countChange flags0.B0

//volatile bit testflag, flag1Sec, setupMode, flag10mS;

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////// FOR SETUP SWITCHES ///////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

FlagConv switchBits;

#define switch1 switch\_flags.B0

#define switch2 switch\_flags.B1

#define switch3 switch\_flags.B2

#define upSwitch switch3

#define enterSwitch switch2

#define dnSwitch switch1

#define \_switch1 switchBits.flagBits.B0

#define \_switch2 switchBits.flagBits.B1

#define \_switch3 switchBits.flagBits.B2

FlagConvchar \_7SegData;

#define \_7SegBit0 \_7SegData.flagBits1.B0

#define \_7SegBit1 \_7SegData.flagBits1.B1

#define \_7SegBit2 \_7SegData.flagBits1.B2

#define \_7SegBit3 \_7SegData.flagBits1.B3

#define \_7SegBit4 \_7SegData.flagBits1.B4

#define \_7SegBit5 \_7SegData.flagBits1.B5

#define \_7SegBit6 \_7SegData.flagBits1.B6

#define \_7SegBit7 \_7SegData.flagBits1.B7

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////// VARIABLE DECLARATIONS///////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

unsigned int switchTable[] = { (506+545)/2, (545+598)/2, (598+653)/2, (653+704)/2, (704+781)/2, (781+894)/2, (894+1023)/2 };

unsigned char publicString[10],charBuffer[20];

unsigned char dispTask, displayLine,lcdDataCount,lcdData,whichChannel,AccumulateCount,totalLinesLCD;

unsigned char disp7SegTask, whichseg;

unsigned int adcResult,switchV,\_switchV, testV, \_testV, \_whichDisplay, keyStrength, \_1secCount;

volatile unsigned int count55uS;

volatile unsigned int outCount, inCount;

const unsigned char companyString[] = "Wait...";

const unsigned char inString[] = "In ";

const unsigned char outString[] = "Out ";

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void init\_clock(void) ///////////////////////////////////////////////////////////////////////////////////////////////

{ //

OSCCONbits.IRCF0 = 0; // 8 MHz (INTOSC drives clock directly)

OSCCONbits.IRCF1 = 1; //

OSCCONbits.IRCF2 = 1; //

//

OSCCONbits.IDLEN = 0; // Device enters Sleep mode when a SLEEP instruction is executed

//OSCCONbits.SCS1 = 1; // Internal oscillator block

//

OSCCONbits.SCS1 = 0; // Primary oscillator

OSCCONbits.SCS0 = 0; // Primary oscillator

//

while(OSCCONbits.IOFS == 0){} // INTOSC Frequency Stable bit

//

OSCTUNEbits.PLLEN = 1; //

// PLL enabled for INTOSC (4 MHz and 8 MHz only)

while(OSCCONbits.IOFS == 0){} // INTOSC Frequency Stable bit

} ///////////////////////////////////////////////////////////////////////////////////////////////

void initialise\_Ports(void) ///////////////////////////////////////////////////////////////////////////////////////////////

{ //

ANSELA = 0x03; //

ANSELB = 0x00; //

ANSELC = 0x00; //

//

TRISA = SetupA; //

TRISB = SetupB; //

TRISC = SetupC; //

//

PORTA = 0x00; //

PORTB = 0x00; //

PORTC = 0x00; //

LATC = 0x00; //

LATB = 0x00; //

LATC = 0x00; //

} ///////////////////////////////////////////////////////////////////////////////////////////////

void initialise\_timer4(void) ///////////////////////////////////////////////////////////////////////////////////////////////

{ // designed for 55 us

T4CON = 0x00; //

TMR4 = 0x00; //

INTCON = 0x00; //

PIE5 = 0x00; //

PIR5 = 0x00; //

//

RCONbits.IPEN = 1; // enable Interrupt priority levels

INTCONbits.GIE = 1; // global interrupt enabled

INTCONbits.PEIE = 1; // peripharal interrupt enabled

IPR5bits.TMR4IP = 1; //

PIE5bits.TMR4IE = 1; //

//

T4CON = 0x08; //

PR4 = 220; //

\_TMR4ON = 1; //

\_TMR4IF = 0; //

//

} ///////////////////////////////////////////////////////////////////////////////////////////////

void init\_adc(void) ///////////////////////////////////////////////////////////////////////////////////////////////

{ // //

ADCON0=0b00000001; // 00000000 CHS=AN0 AD/OFF

ADCON1=0b00000000; // 00000000 TRIGSEL=ECCP1 VCFG=AVDD VNCFG=AVSS

ADCON2=0b10000010; // 10101100 Right Just 12 TAD Fosc/32

IPR1bits.ADIP = 1;

PIE1bits.ADIE = 1;

}

//-------------------------------------------------------------------------------------------------------------------------------//

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////// LCD FUNCTIONS //////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void showSimpleMessage(unsigned char \*string) //-----------------------------------------------

{

dispTask = -1;

if(setupMode)

{

displayLine = 0; //

if(\_whichDisplay && !setupMode) //

charBuffer[++dispTask] = '\*'; //

} //

while(++dispTask < 16) //

{ //

charBuffer[dispTask] = \*string; //

if(!\*string++) // Note: Simply calling showSimpleMessage() will not display anything on LCD.

break; // This function prepare the charBuffer[] and other variables for display.

} // After calling showSimpleMessage(), we should call showMessage() several

dispTask = 0; // times to make the message really appear on LCD.

}

//-------------------------------------------------------------------------------------------------------------------------------//

void decimal2String(unsigned int number) //-----------------------------------------------

{ // This function convert the number (passed to this function) to string and

unsigned int count; // store the result in publicString[]. For example if 23768 is passed then

count = 5; // on completion publicString[0] = '2', publicString[1] = '3' ...

while(count--) // publicString[4] = '8'

{ //

publicString[count] = number%10+'0'; //

number /= 10; //

} //

} //-----------------------------------------------

//-------------------------------------------------------------------------------------------------------------------------------//<<<

void showMixedMessage(unsigned char \*string, unsigned int number, unsigned int format, unsigned char lastChar)

{ //

unsigned int count1, count2, count3; // This is a tested OK function to display messages mixed with number.

// For example to display 'Input Volt = 213.8V' this function can be used.

dispTask = -1;

if(setupMode)

{

displayLine = 0; //

if(\_whichDisplay && !setupMode) //

charBuffer[++dispTask] = '\*'; //

} //

while(++dispTask < 16) // The calling convention:

{ // To display 'Input Volt = 213.8V' call the function as:

if(!\*string) //

break; // showMixedMessage("Input Volt = ", volt, 0x0304, 'V')

charBuffer[dispTask] = \*string++; // -or-

} // showMixedMessage(string, volt, 0x0304, 'V')

charBuffer[dispTask++] = ' '; //

decimal2String(number); //

integer.\_word = format; // Where string is defined as a char string 'Input Volt = 'in prog memory.

count1 = (unsigned int) integer.\_byte.\_msb; //

count2 = count1; // The argument 'format':

count1 /= 16; // This argument decide how the number will displayed. For example if volt

count2 %= 16; // = 21386 and it is to be displayed as 213.8 the formatting is done as:

count3 = (unsigned int) integer.\_byte.\_lsb; //

while(count1 < 5) // (1) Char to be hidden at left side: 0

{ // (2) Digits before decimal point: 3

if(!count2--) // (3) Number of digits to be displayed: 4

{ //

charBuffer[dispTask++] = '.'; // With this the foratting = 0x0304. Note that 4 bits are allotted for first

continue; // 2 parameters but 8 bits are allotted for the 3rd parameter.

} //

if(!count3--) //

break; // Note: Simply calling showMixedMessage() will not display anything on LCD.

charBuffer[dispTask++] = publicString[count1++]; // This function prepare the charBuffer[] and other variables for display.

} // After calling showMixedMessage(), we should call showMessage() several

charBuffer[dispTask++] = lastChar; // times to make the message really appear on LCD. Also note that each call

charBuffer[dispTask] = 0; // to showMessage() should be seperated by a time delay. See displayTask()

dispTask = 0;

// to see how it is managed.

}

//-------------------------------------------------------------------------------------------------------------------------------//

void writeData(unsigned char value) //------------------------------------------------------------------

{ //

while(lcdDataCount); //

lcdData = value; //

lcdDataCount = 6; //

}

//-------------------------------------------------------------------------------------------------------------------------------//

void showMessage(void) //-------------------------------------------------------

{ // This function is a basic LCD function because it includes LCD

#ifdef autoReInitialize // initialisation

static char refreshCount = 0; //

#endif //

char element; //

//

regSelect = 0; // This is a function exactly similar to the one we have

if(lcdInitialised) // written in 'sinewave without EE16'.

{ //

if(dispTask < 22) // Writting a message to LCD need several tasks like:

{ // setting address 0, writting 8 char, setting address

switch(dispTask) // 0x28, writting remaining char etc. The gap between

{ // each write to the LCD should be at least 40uS.

case 0 : // So as we usually do, this function should be repetedly

#ifdef autoReInitialize // called until displTask = 128.

if(!displayLine) //

{ //

if(++refreshCount > 8) // When first calling dispTask = 0 and charBuffer[] should

{ // be filled with the message to be shown. After the first

refreshCount = 0; // call simply call the function again and again until

lcdInitialised = 0; // dispTask becomes 128.

break; //

} // This function will take care of long messages (only the

} // first 16 char will be shown) and short message (space

#endif // will be filled at the end of message)

if(!displayLine) //

writeData(0x80); //

if(displayLine == 1) //

writeData(0xC0); //

if(displayLine == 2) //

writeData(0x94); //

if(displayLine == 3) //

writeData(0xD4); //

if(++displayLine >= totalLinesLCD) //

{ //

displayFull = 1; //

displayLine = 0; // At beginning, when the LCD is just powered and not

} // initialised calling this function again and again will

else // do the initialisation process step-by-step. Flag

displayFull = 0; // 'lcdInitialised' = 0 means LCD is not initialised In

dispTask++; // that case the initialisation process are done before

messageEnd = 0; // doing the message display.

break; // 31/08/2007: Added auto initialization feature for LCD.

case 9 : // The entire program rely on this function for displaying

if(totalLinesLCD == 1) // something on LCD. After displaying few messages the

writeData(0x28+0x80); // function itself load the flag lcdInitialised <- 0 so that

dispTask++; // LCD is automatically re-initialized.

break; //

default : // The strategy is: When dispTask == 0 the job of this function

regSelect = 1; // is to write 0x80 with regSelect <- 0 (make address = 0x00).

if(dispTask < 9) // This is the first job for starting a new message. At that

element = charBuffer[dispTask-1]; // time function check whether ++refreshCount > 8.

else // If so function reload refreshCount <- 0, lcdInitialised <- 0

element = charBuffer[dispTask-2]; // and return without doing usual job. From the next

if(element && !messageEnd) // call onwards, since lcdInitialised == 0, the function

writeData(element); // will start LCD re-initialization. Note that function

else // is checking ++refreshCount > 8. Instead of 8 it can be 9, 10

{ // or any other number. Checking ++refreshCount > 8 means the

writeData(' '); // function do the re-initialization after every 8 messages.

messageEnd = 1; // Seems to be OK.

} // \*\*\*\* NOTE

if(++dispTask > 21) // The main controller that send UART string to this controller

dispTask = 128; // can periodically send char 0x80 to re-initialize the LCD.

break; // When UART interrupt receive 0x80 it make lcdInitialised <- 0.

} // So auto re-initialization is not needed in this code. This can

} // be done with conditional compiling with 'autoReInitialize'

else //

dispTask = 128; // Even if the LCD is already initialised there is no problem

} // to initialise it again.

else //-------------------

{ // Control reach here if LCD is not initialized

switch(dispTask++) // (lcdInitialised != 1). In that case using the variable

{ // dispTask as an index this function do one small task at a

case 0: // time. Within 10 steps the initialisation will be complete

case 1: // and when it is complete the flag lcdInitialised <- 1.

case 2: //

case 3: // Actually 10 steps for initialisation of LCD is little over.

lcdData4 = 1; // No need of a time gap between these steps.

if(dispTask == 4)

lcdData4 = 0;

regSelect = 0;

lcdData7 = 0;

lcdData6 = 0;

lcdData5 = 1;

while(lcdDataCount); //

lcdDataCount = 2; //

break;

case 4:

writeData(FunSet);

break;

case 5:

writeData(DisplayOFF);

break;

case 6:

writeData(DisplayON);

break;

case 7:

writeData(EntryMode);

break;

case 8:

writeData(ClearDisplay);

break;

case 9:

lcdInitialised = 1;

dispTask = 0;

displayFull = 0;

displayLine = 0;

break;

}

}

}

//-------------------------------------------------------------------------------------------------------------------------------//

unsigned int calcSwitchBits(void) //-----------------------------------------------------------

{ // From the switch voltage this function calculate which switches are pressed.

unsigned int count = 0; //

while(switchV > 64\*switchTable[count]) // Note that switch voltage is sampled by the ADC section and 64 samples are

if(++count > 6) // added together and stored in switchV. Note that since ADC is 10bit adding 64

break; // nos will not cause an overflow. It will give a filtering also.

return(count); //

} //-----------------------------------------------------------

//-------------------------------------------------------------------------------------------------------------------------------//

void readSwitchBits(void) //----------------------------------------------------------

{ // For proper working this function should be called frequently, say @10mS.

static unsigned int oldSwitch, switchCount = 10;//

unsigned int value; // For the internal use this function keeps two variables oldSwitch and

// switchCount. If the switch bits read using calcSwitchBits() is different

value = calcSwitchBits(); // from the old value then switchCount <- 10. When switchCount is decremented

if(value != oldSwitch) // down to zero the present switch bits is moved to the integer portion

switchCount = 10; // of switchBits.

oldSwitch = value; //

if(switchCount > 10) // Note that bit0, bit1 and bit2 of switchBits is defined as switch1,

switchCount = 10; // switch2 and switch3. So we can know whether these bits are set

if(!--switchCount) // or cleared. When a switch is pressed the coresponding flag is cleared.

{ // This should be taken care. You can think that when the switch is pressed

switchCount++; // the pin voltage become 0 and so the flag also become 0.

switchBits.intValue = value; //

} //

} //----------------------------------------------------------

//-------------------------------------------------------------------------------------------------------------------------------//

void checkKeyboard(void) //----- This function should be called 20 or 10 mS ----------------

{ // This is a tested OK function to read the switch press. There are

static unsigned int oldSwitch, debounce = 10; // 3 flags: \_switch1, \_switch2, \_switch3 and similar to that other

static unsigned int keyRepeatCount; // 3 flags switch1, switch2, switch3. When switch1 is pressed \_switch1

// will become 0 and it will stay low as long as the switch is kept

readSwitchBits(); // pressed. When the switch is released \_switch1 will become 1.

switch1 = 0; //

switch2 = 0; //

switch3 = 0; // But the flag 'switch1' will be set by this function when the switch

if(!--debounce) // is pressed and function will set it again and again when the switch

{ // is kept pressed (key repeat action). When the switch is released and

++debounce; // pressed again the flag will be set again.

if(oldSwitch != switchBits.intValue) //

{ // The key repeat facility also has accelerated key repeatation. This is

debounce = 10; // achieved with a simple trick. On the first key press the global

if( (oldSwitch & 0x0001) && !\_switch1) // variable keyStrength will be 1. If the user keep press the key then key

switch1 = 1; // repeat action will start and on every repeat action keyStrength will be

if( (oldSwitch & 0x0002) && !\_switch2) // incremented. (from this the calling function can distinguish whether the

switch2 = 1; // key press is first press or auto-repeat action. In auto-repeat action

if( (oldSwitch & 0x0004) && !\_switch3) // keyStrength != 1).

switch3 = 1; //

oldSwitch = switchBits.intValue; //

keyStrength = 1; //

keyRepeatCount = 200; //

} //

else //

{ //

if(setupMode) //

{ //

if( !(--keyRepeatCount) ) //

{ //

if(keyStrength < 3000) //

++keyStrength; //

keyRepeatCount = 20; //

if(!\_switch1) //

switch1 = 1; //

if(!\_switch2) //

switch2 = 1; //

if(!\_switch3) //

switch3 = 1; //

} //

} //

} //

} //

}

//-------------------------------------------------------------------------------------------------------------------------------//

void load7segData(unsigned char data, unsigned char whichdisp)

{

if(disp7SegTask)

return;

\_7SegData.charValue = data;

whichseg = whichdisp;

disp7SegTask = 1;

}

//-------------------------------------------------------------------------------------------------------------------------------//

void displayTask(void)

{

if(dispTask < 127)

{

showMessage();

return;

}

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////// main ///////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void main(void)

{

init\_clock();

initialise\_Ports();

init\_adc();

initialise\_timer4();

lcdInitialised = 0;

displayLine = 0;

showSimpleMessage( (unsigned char \*) companyString);

count55uS = 00; //

timeOut = 0;

while(!timeOut);

inCount = 0;

outCount = 0;

while(1)

{

if(countChange)

\_countChange = 1;

else

\_countChange = 0;

countChange = 0;

count55uS = 20000;

timeOut = 0;

while(!timeOut)

{

if(\_countChange)

{

if(count55uS > 19000)

buzzer = 1;

else

buzzer = 0;

}

if(flag10mS)

{

flag10mS = 0;

displayTask();

checkKeyboard();

}

if(count55uS == 19900)

{

count55uS--;

displayLine = 0;

showMixedMessage(inString, inCount, 0x0605, ' ');

}

if(count55uS == 10000)

{

count55uS--;

displayLine = 1;

showMixedMessage(outString, outCount, 0x0605, ' ');

}

}

}

Led1 = 0;

Led2 = 0;

Led3 = 0;

Led4 = 1;

buzzer = 0;

load7segData(0x06, 1); // 1

while(1)

{

if(flag10mS)

{

flag10mS = 0;

displayTask();

checkKeyboard();

}

if(!\_switch3)

Led1 = 1;

else

Led1 = 0;

if(!\_switch2)

Led2 = 1;

else

Led2 = 0;

if(!\_switch1)

{

Led3 = 1;

buzzer = 1;

}

else

{

buzzer = 0;

Led3 = 0;

}

if(flag1Sec)

{

flag1Sec = 0;

Led4 = ~Led4;

testflag = ~ testflag;

if(testflag)

load7segData(Pattern0, 2); // 0

else

load7segData(Pattern2, 1); // 2

}

}

return;

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////// INTERRUPT //////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void \_\_interrupt() high\_isr (void)

{

if(\_TMR4IF) // 55us second overflow

timer4Isr();

if(\_ADIF)

adcIsr();

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#define NoSensorWait 20000

void timer4Isr(void)

{

static unsigned char \_55us\_count;

static unsigned char countSensor1, countSensor2, wait4NoSignal;

static unsigned int longWait;

if(!(--count55uS)) //

timeOut = 1;

ADCON0bits.GO = 1; // Start ADC conversion

if(++\_55us\_count >= 180)

{

\_55us\_count = 0;

flag10mS = 1;

if(++\_1secCount >= 100)

{

\_1secCount = 0;

flag1Sec = 1;

}

}

//--- Code related with sensors -------------------------------

if(!sensorPin1)

\_sensorPin1 = 1;

else

\_sensorPin1 = 0;

if(!sensorPin2)

\_sensorPin2 = 1;

else

\_sensorPin2 = 0;

if(\_sensorPin1)

{

longWait = NoSensorWait;

if(countSensor1 < 200)

++countSensor1;

else

{

if(countSensor2 > 199)

{

if(!wait4NoSignal)

{

++outCount;

countChange = 1;

}

wait4NoSignal = 10;

}

}

}

if(\_sensorPin2)

{

longWait = NoSensorWait;

if(countSensor2 < 200)

++countSensor2;

else

{

if(countSensor1 > 199)

{

if(!wait4NoSignal)

{

++inCount;

countChange = 1;

}

wait4NoSignal = 10;

}

}

}

if(!\_sensorPin1 && !\_sensorPin2)

{

if(wait4NoSignal)

--wait4NoSignal;

if(longWait)

--longWait;

else

{

countSensor1 = 0;

countSensor2 = 0;

}

}

//--- End of Code related with sensors -------------------------------

if(lcdDataCount)

{

switch(lcdDataCount--)

{

case 6:

lcdData4 = 0; // be noted. This is explained at the end of this file.

lcdData6 = 0; //

lcdData7 = 0; // <---- order of clearing data4, 5, 6 and 7 is very important. Since data4 and data5

lcdData5 = 0;

if(lcdData & 0x0010) // setting/clearing data4.

lcdData4 = 1; //

if(lcdData & 0x0020)

lcdData5 = 1;

if(lcdData & 0x0040)

lcdData6 = 1;

if(lcdData & 0x0080)

lcdData7 = 1;

break;

case 5:

lcdClock = 1;

break;

case 4:

lcdClock = 0; //for lcd function on timer

break;

case 3:

lcdData4 = 0;

lcdData6 = 0;

lcdData7 = 0;

lcdData5 = 0;

if(lcdData & 0x0001)

lcdData4 = 1;

if(lcdData & 0x0002)

lcdData5 = 1;

if(lcdData & 0x0004)

lcdData6 = 1;

if(lcdData & 0x0008)

lcdData7 = 1;

break;

case 2:

lcdClock = 1;

break;

case 1:

default:

lcdClock = 0;

lcdDataCount = 0;

break;

}

}

switch(disp7SegTask)

{

case 1:

data4094 = \_7SegBit7;

disp7SegTask++;

break;

case 2:

clock4094 = 1;

clock4094 = 0;

disp7SegTask++;

break;

case 3:

data4094 =\_7SegBit6;

disp7SegTask++;

break;

case 4:

clock4094 = 1;

clock4094 = 0;

disp7SegTask++;

break;

case 5:

data4094 = \_7SegBit5;

disp7SegTask++;

break;

case 6:

clock4094 = 1;

clock4094 = 0;

disp7SegTask++;

break;

case 7:

data4094 = \_7SegBit4;

disp7SegTask++;

break;

case 8:

clock4094 = 1;

clock4094 = 0;

disp7SegTask++;

break;

case 9:

data4094 =\_7SegBit3;

disp7SegTask++;

break;

case 10:

clock4094 = 1;

clock4094 = 0;

disp7SegTask++;

break;

case 11:

data4094 = \_7SegBit2;

disp7SegTask++;

break;

case 12:

clock4094 = 1;

clock4094 = 0;

disp7SegTask++;

break;

case 13:

data4094 =\_7SegBit1;

disp7SegTask++;

break;

case 14:

clock4094 = 1;

clock4094 = 0;

disp7SegTask++;

break;

case 15:

data4094 = \_7SegBit0;

disp7SegTask++;

break;

case 16:

clock4094 = 1;

clock4094 = 0;

disp7SegTask++;

break;

case 17:

Disp1CS = 0;

Disp2CS = 0;

latch4094 = 1;

latch4094 = 0;

if(whichseg == 0x01)

Disp1CS = 1;

else

Disp2CS = 1;

disp7SegTask = 0;

break;

default:

disp7SegTask = 0;

break;

}

\_TMR4IF = 0;

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void adcIsr(void) //

{ //

\_ADIF = 0; // <- Clear the interrupt flag

adcResult = ADRESH\*256 + ADRESL; // ADC result is in ADRESL and ADRESH (LSB and MSB)

switch(whichChannel++) // ADC conversion can be done one channel at a time only. Which channel is converted

{ // now is indicated by the variable whichChannel. At present we have only 2 channels

case 1: // now. So whichChannel take only two values 1 and 2.

\_testV += adcResult; // reading analog channel 0.

ADCON0 = 0b00000101; // After reading the first result load ADCON0 for the next channel. The channel is now

break; // connected to the ADC. When ADCON0bits.GO bit is set by timer4 the conversion will start.

case 2: //

\_switchV += adcResult; // reading analog channel 4

ADCON0 = 0b00000001; // After reading the result load ADCON0 for the next channel.

whichChannel = 1;

if(++AccumulateCount >= 64)

{

AccumulateCount = 0;

testV = \_testV;

\_testV = 0;

switchV = \_switchV;

\_switchV = 0;

}

break;

default:

ADCON0 = 0b00000001; // Other than 1 and 2 there is no other possible values for whichChannel. But we are

whichChannel = 1; // taking precaution to handle that possibility also.

break;

}

}