Ronobir Mookherjee

Design Assignment 5 Part 2

Description:

Similar to Part 1, but here the data from the temperature sensor is transferred to a 4 digit 7 Segment LED display driven from an LED driver chip, here the max 7221.

Flowchart:

Code:

*/\**

*\* CpE301DesignAssignment5Part2.c*

*\**

*\* Submission for part 1*

*Part 1 and 2 were combined into one project, and are being separated and submitted individually.*

*The purpose of part 2 is to get the value of the Temperature using ADC once per second and outputting to a 7 segment display*

*The four primary components used include the Temperature sensor LM34 from TI, the Atmega328p, the Maxim 7221cng, and a 4 Digit 7 Segment display*

*The transmission protocol used here is SPI*

*The MAX 7221 chip was employed because of its specially properties as listed: BCD decoder, Multilpex scan, SRAM array, and you only need 1 external resistor to provide the necessary power to the 4 digit 7 Segment display. The Maxim 7221 is known as an 8 digit LED display Driver.*

*\* Author: Ronobir Mookherjee*

*\*/*

*/\* Below is the standard setup for an AVR project, with preprocessor directives\*/*

#include <stdlib.h>

#include <alloca.h>

#include <assert.h>

#include <ctype.h>

#include <errno.h>

#include <inttypes.h>

#include <math.h>

#include <setjmp.h>

#include <stdint.h>

#include <stdio.h>

#include <string.h>

#include <avr/boot.h>

#include <avr/io.h>

#include <avr/cpufunc.h>

#include <avr/eeprom.h>

#include <avr/fuse.h>

#include <avr/interrupt.h>

#include <avr/lock.h>

#include <avr/power.h>

#include <avr/pgmspace.h>

#include <avr/sfr\_defs.h>

#include <avr/sleep.h>

#include <avr/version.h>

#include <util/crc16.h>

#include <util/parity.h>

volatile int TempRead;

volatile char strBuffer[4];

volatile char binZero = 0b00111111;

volatile char binOne = 0b00000110;

volatile char binTwo = 0b01011011;

volatile char binThree = 0b01001111;

volatile char binFour = 0b01100110;

volatile char binFive = 0b01101101;

volatile char binSix = 0b01111101;

volatile char binSeven = 0b00000111;

volatile char binEight = 0b01111111;

volatile char binNine = 0b01101111;

*//Above is for debugging purposes, to send data each segment individually, each represents a number on the 7 segment display*

void InitialzeIO(void){

*// Debug ports to check if all functions are working*

DDRC = (0 << DDC0); *//Input Port for ADC Temp Sensor*

*/\*PORTC |= (1<<PORTC0);\*/*

DDRD |= (1<<DDD2)|(1<<DDD6); *// Debug Ports*

}

void InitTimer(void) *//Timer used for ADC conversion, set to 1s*

{

OCR1A = 8000; *//1s value at 8mhz clock rate*

TIMSK1 |= (1 << OCIE1A); *//enables CTC mode interrupts*

TCCR1B |= (1 << WGM12); *//sets to Mode 4 or CTC mode*

}

void startTimer(void) *//function call that starts timer*

{

TCCR1B |= (1 << CS12) | (1 << CS10); *//clock prescaler that divides clock 1024 for the system clock*

}

void stopTimer(void) *//stops timer*

{

TCCR1B &= ~((1 << CS12) | (1 << CS10));

TIMSK1 &=~(1<<OCIE0A);

}

void InitADC(){ *//initializes ADC*

*//select Vref = AVCC and left adjust result setting to 8 bits*

ADMUX = (0<<REFS1)|(0<<REFS0)|(1<<ADLAR);

*//set prescaler, enable autoriggering, enable adc interrupt, and enable adc*

ADCSRA |= (1<<ADPS1)|(1<<ADPS2)|(1<<ADPS0)|(1<<ADATE)|(1<<ADIE)|(1<<ADEN);

ADCSRB |= (0<<ADTS1)|(1<<ADTS0)|(1<<ADTS2); *//Set clock division value to 256*

}

void SetADCChannel(int ADCchannel){ *//function to select which ADC channel to pickup from for output going to the ADCMUX*

if (ADCchannel == 5)

{

ADMUX = (0<<MUX1)|(0<<MUX3)|(1<<MUX2)|(1<<MUX0);

}

else if (ADCchannel == 0)

{

ADMUX = (0<<MUX1)|(0<<MUX3)|(0<<MUX2)|(0<<MUX0);

}

else

{

ADMUX = (0<<MUX1)|(1<<MUX3)|(0<<MUX2)|(0<<MUX0);

}

}

void startADC(void) *//Starts ADC transmission*

{

ADCSRA |= (1<<ADSC);

}

void StopADC(void)

{

ADCSRA &= ~((1<<ADEN)|(1<<ADIE));

}

void delayforLED(void) *// use Timer0 to generate 2.5 ms delay for 4 Digit 7 Segment Display*

{

*// Set the Timer Mode to CTC*

TCCR0A |= (1 << WGM01);

*// Set the value that you want to count to*

OCR0A = 125;

*// start the timer*

TCCR0B |= (0 << CS01) | (0 << CS00)| (1 << CS02);

*// set prescaler to 256 and start the timer*

while ( (TIFR0 & (1 << TOV0) ) > 0) *// wait for the overflow event*

{

}

TIFR0 &= ~(1 << TOV0);

*// reset the overflow flag*

}

void initializeSPI(void) *//SPI communication used to communicate with the LED Driver Chip, Max 7221*

{

DDRB |= (1<<2)|(1<<3)|(1<<5); *// SCK, MOSI and SS as outputs*

DDRB &= ~(1<<4); *// MISO as input*

SPCR |= (1<<MSTR); *// Set as Master*

SPCR |= (1<<SPR0)|(1<<SPR1); *// divided clock by 128*

}

void startSPI(void) *//Enables the start bit on the SPI register*

{

SPCR |= (1<<SPE); *// Enable SPI*

}

void sendDataSPI(unsigned char cmd, unsigned char data) *//function to send data, first with the command op code, and then the data values to be displayed*

{

PORTB &= ~(1<<2); *// turn off slave select for transmission so there isn't any noise picked up*

SPDR = cmd; *// Command data, used by the Max 7221 for operation, such as selecting which digits, or how many digits*

while(!(SPSR & (1<<SPIF))); *// wait until transmission is complete*

SPDR = data; *// send the data*

while(!(SPSR & (1<<SPIF))); *// wait until transmission is complete*

PORTB |= (1<<2); *//enables slave select and finishes transmission*

}

int binary\_decimal(int n) */\* Function to convert binary to decimal. used for debugging\*/*

{

int decimal=0, i=0, rem;

while (n!=0)

{

rem = n%10;

n/=10;

decimal += rem\*pow(2,i);

++i;

}

return decimal;

}

void stopSPI(void)

{

SPCR |= ~(1<<SPE); *//Disable SPI*

DDRB &= ~((1<<2)|(1<<3)|(1<<5)); *// SCK, MOSI and SS as inputs*

DDRB |= (1<<4); *// MISO as output*

SPCR &= ~(1<<MSTR); *// Set as slave*

}

void initialize7Seg(void) *//initializes the 4 digit 7 segment display and the Maxim chip*

{

sendDataSPI(0x09, 0b00001111); *//Scans for 4 digits of the 7 segment display*

sendDataSPI(0x0B, 0x04); *// enables operation for 4 digits*

sendDataSPI(0x0C, 0x01); *// turns on the display*

}

void updateDisplay(int Temperature)

{

*//\*\*\*\*\*\*\* Remember that the Max 7221 chip has a built in BCD \*\*\*\*\*\*\*\*//*

*//for the first digit*

sendDataSPI(0x01, Temperature/1000); *//shifts first digit of data*

sendDataSPI(0x02, ((Temperature % 1000)/ 100)); *//shifts next digit, using mod of 10^3*

sendDataSPI(0x03, ((Temperature%100)/10)); *// shifts third digit*

sendDataSPI(0x04, ((Temperature%100)%10)); *//shifts final digit*

*// With the Max 7221 chip you don't need to refresh the display quickly to display all characters, the LED Driver takes care of this*

}

int main(void) *//setups up the chip, with the function calls to enable communication using SPI and ADC conversion*

{

InitialzeIO();

InitADC();

InitTimer();

startTimer();

startADC();

initializeSPI();

initialize7Seg();

startSPI();

while(1)

{

*//main loop*

}

}

ISR(ADC\_vect) *// ADC to pull data from the temperature sensor every 1 second*

{

while(ADIF == 0){

};*//check to see if ADCH and ADCL have been populated before proceeding*

TempRead = ADCH;*// is the value of the temperature sensor captured and stored in the variable*

TIFR1 = (0<<OCF1A)|(0<<OCF1B);*//disables flag to reset interrupt and start again*

StopADC();*//stops ADC to restart back to main.*

itoa(binary\_decimal(TempRead), strBuffer, 10); *//used for debugging, the itoa function takes the integer and stores in a string buffer (array) that can also be used to send data, digit by digit*

updateDisplay(TempRead); *//outputs to the 4 digit 7 segment display using SPI communication*

stopSPI();

}

Schematics:







