Michael Ghisilieri

CPE301 – SPRING 2016

Design Assignment 3

**DO NOT REMOVE THIS PAGE DURING SUBMISSION:**

The student understands that all required components should be submitted in complete for grading of this assignment.

|  |  |  |  |
| --- | --- | --- | --- |
| **NO** | **SUBMISSION ITEM** | **COMPLETED (Y/N)** | **MARKS**  **(/MAX)** |
| 0. | COMPONENTS LIST AND FLOW CHART |  |  |
| 1. | INITIAL CODE OF TASK 1 |  |  |
| 2. | SCHEMATICS |  |  |
| 3. | SCREENSHOTS OF EACH TASK OUTPUT |  |  |
| 4. | SCREENSHOT OF EACH DEMO |  |  |
| 5. | GITHUB LINK OF THE DA |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 0. | COMPONENTS LIST AND FLOW CHART |  |  |

**COMPONENTS:**

ATmega328P Chip x1

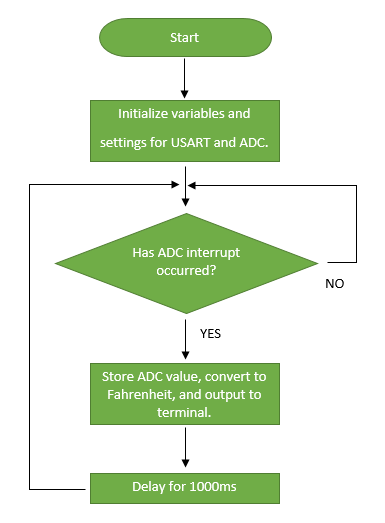
LM34 Temperature Sensor x1

SparkFun FTDI Chip x1

Mini USB cable x1

120 nF capacitor x2

**FLOWCHART:**



|  |  |  |  |
| --- | --- | --- | --- |
| 1. | INITIAL CODE OF TASK 1 |  |  |

/\*

\* DA3\_Task1.c

\*

\* Created: 3/17/2016 2:42:22 PM

\* Author : Michael

\*/

// This code waits for a character and transmits the character back (with interrupts)

#define *F\_CPU* 8000000UL // Clock Speed

#include <avr/io.h>

#include <stdint.h> // needed for uint8\_t

#include <avr/interrupt.h>

#include <util/delay.h> // include library for \_delay\_ms function

void USART0SendByte(char);

void delay (); // function declaration for delay

// 32 bit integer to hold ADC value during conversion to Fahrenheit value

volatile *uint32\_t* temperature;

volatile *uint16\_t* ADCvalue; // Global variable, set to volatile if used with ISR

volatile char temp\_out; // current digit of temperature being transmitted

int main(void)

{

// UBRR = (F\_CPU / (16 \* 9600)) - 1

// UBRR = (8 MHz / (16 \* 9600)) - 1 = 51 or 0x33

UBRR0L = 0x33; // set based on equation using baud rate of 9600

UCSR0B = (1 << TXEN0); // Enable transmitter

UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); // 8 bit data

ADMUX = 0; // use ADC0

ADMUX |= (1 << REFS0); // use AVcc as the reference

ADCSRA |= (1 << ADPS2) | (1 << ADPS1); // 64 prescale for 16Mhz

ADCSRA |= (1 << ADATE); // Set ADC Auto Trigger Enable

ADCSRB = 0; // 0 for free running mode

ADCSRA |= (1 << ADEN); // Enable the ADC

ADCSRA |= (1 << ADIE); // Enable Interrupts

ADCSRA |= (1 << ADSC); // Start the ADC conversion

sei();

while(1) // loop forever and wait for ADC interrupt

{

}

}

ISR(ADC\_vect)

{

ADCvalue = ADC; // read all bits of ADC (10) and store into ADCvalue

temperature = ADCvalue; // copy value into new variable in case ADC overwrites it

// equation used to convert ADC value is:

// (ADC register \* 5 \* 100) / 1024

// which is the ADC multiplied by the reference voltage multiplied by 100

// since the LM34 sensor gives its value in mV and then divide by 1024 since

// a ten bit register will have a max value of 2^10 or 1024

temperature = temperature\*5;

temperature = temperature\*100;

temperature = temperature/1024;

temp\_out = temperature / 10; // dividing 2 digit temp (i.e. 75 or 80) will give tens digit

USART0SendByte((temp\_out+'0')); // output tens digit converted to its matching char

temp\_out = temperature % 10; // % 10 will give the ones digit of a 2 digit temp

USART0SendByte((temp\_out+'0')); // output ones digit converted to its matching char

USART0SendByte('\n'); // send a line feed to the terminal

USART0SendByte('\r'); // send return to the terminal to move cursor of terminal

delay(); // delay for 1000 ms or 1 second

}

void USART0SendByte(char u8Data)

{

//wait while previous byte is completed

while(!(UCSR0A&(1<<UDRE0))){};

// Transmit data

UDR0 = u8Data;

}

void delay () {

int i; // declare i for counter

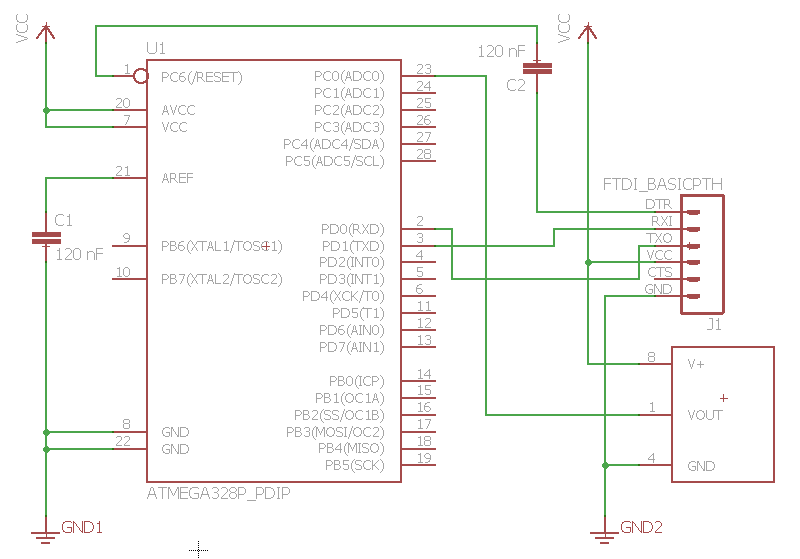
for(i = 0; i < 100; i++) { // loop 100 times

*\_delay\_ms*(10); // delay 10 ms

} // total delay = 100\*10ms = 1000ms

}

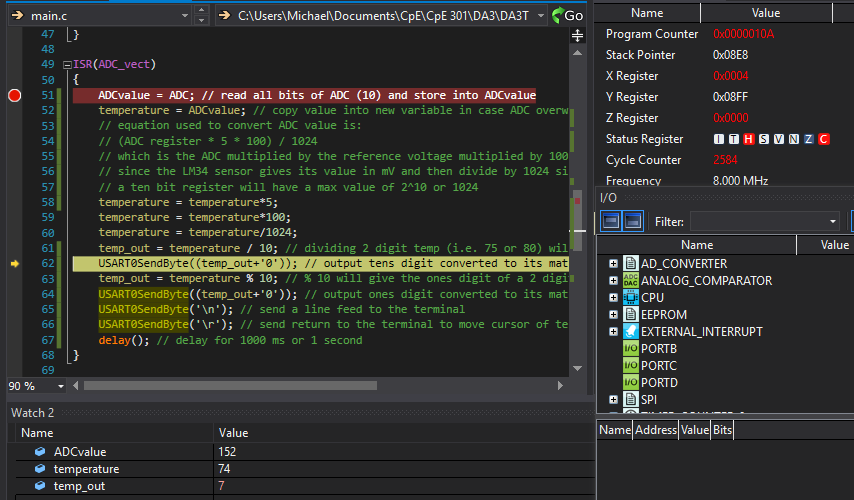
|  |  |  |  |
| --- | --- | --- | --- |
| 2. | SCHEMATICS |  |  |



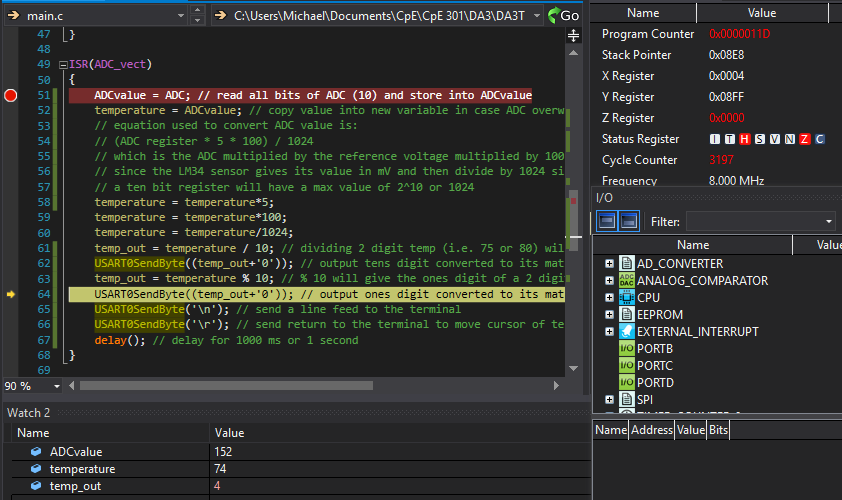
|  |  |  |  |
| --- | --- | --- | --- |
| 3. | SCREENSHOTS OF EACH TASK OUTPUT |  |  |

TASK 1:

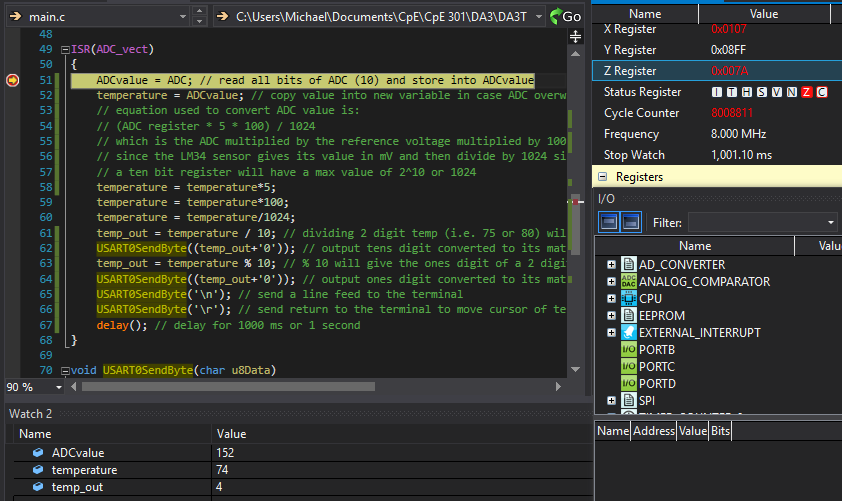
Giving the ADCvalue in simulation a value of 152, which is very similar to what the ADC would actually get at room temperature, will convert the temperature correction to Fahrenheit. It is shown that a temperature of 74 is calculated and the first digit being transmitted to the terminal ‘7’ is the tens digit of the temperature.



The second digit being transmitted is a ‘4’ which is achieved with %10 of the temperature transmitting the ones digit of the temperature.

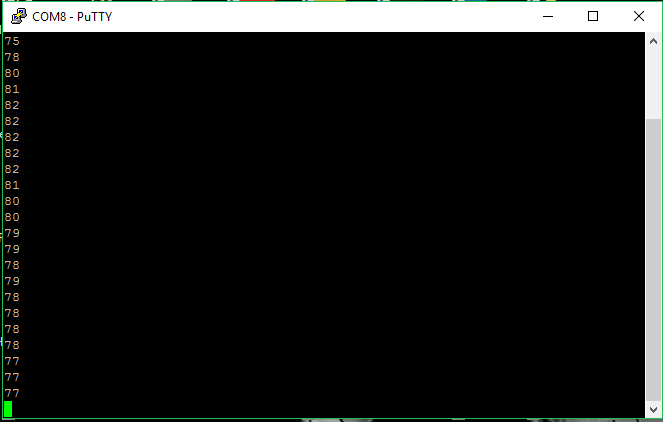


This simulation shot shows that it roughly takes 1,001.10 ms to get back to the ADC interrupt after the delay function is called and the interrupt occurs again. This is very close to the expected 1 second that should be transmitted to the terminal at **1.0011** seconds.



|  |  |  |  |
| --- | --- | --- | --- |
| 4. | SCREENSHOT OF EACH DEMO |  |  |

TASK 1: Screenshot that shows rising and falling temperature from touching sensor with finger



|  |  |  |  |
| --- | --- | --- | --- |
| 5. | GITHUB LINK OF THE DA |  |  |
| https://github.com/michael-ghisilieri/CpE301\_DAs.git | | | |

**Student Academic Misconduct Policy**

<http://studentconduct.unlv.edu/misconduct/policy.html>

“This assignment submission is my own, original work”.

Michael Ghisilieri