ESE381 Embedded Microprocessor Systems Design II

Spring 2024, K. Short
revised April 4, 2024 3:03 pm

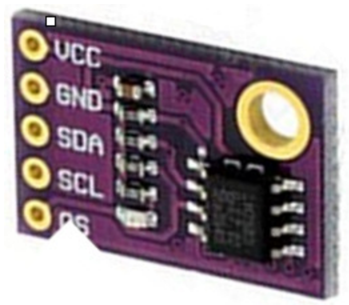
Laboratory 09: LM75 I2C Interface and Basic I2C Transactions

To be performed the week starting April 7th.

Prerequisite Reading
1 AVR128DB48 Data Sheet Section 29, TWI - Two-Wire Interface
2. UM10204 I2C-Bus Specification And User Manual
3. LM75 Digital Temperature Sensor and Thermal Watchdog with 2-Wire Interface Data Sheet.
4. LabLec 09: LM75 I2C Interface and Basic I2C Transactions
5. Print Floating Point Numbers in AVR C with Atmel Studio
https://startingelectronics.org/articles/atmel-AVR-8-bit/print-float-atmel-studio-7/

Overview

The LM75 temperature sensor includes a silicon band gap temperature sensor, a delta-sigma analog-to-digital converter, and a digital overtemperature detector. A microcontroller can query the LM75 through its I2C interface to read the sensor's temperature at any time. You will use an LM75 mounted on a breakout board, as shown below.



The breakout board has been hardwired so that the least significant three bits of the LM75's slave address are 000. The slave address control byte is shown below.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
1	0	0	1	A2	A1	A0	R/$\overline{W}$

The LM75 has four registers.

REGISTER NAME	ADDRESS (hex)	POR STATE (hex)	POR STATE (binary)	POR STATE (°C)	READ/ WRITE
Temperature	00	000X	0000 0000 0XXX XXXX	—	Read only
Configuration	01	00	0000 0000	—	R/W
T_{HYST}	02	4B0X	0100 1011 0XXX XXXX	75	R/W
T_{OS}	03	500X	0101 0000 0XXX XXXX	80	R/W

X = Don't care.

To, keep this laboratory as simple as possible, we will only be using the Temperature register. Registers are accessed using a pointer to the register. This pointer value defaults to point to the Temperature register when the LM75 is reset. The most significant nine bits of the 16-bit Temperature register hold the measured temperature as a 2's complement number.

UPPER BYTE								LOWER BYTE							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Sign bit 1= Negative 0 = Positive	MSB 64°C	32°C	16°C	8°C	4°C	2°C	1°C	LSB 0.5°C	X	X	X	X	X	X	X

X = Don't care.

The purpose of this laboratory is to design and debug an I2C interface to the LM75 and write a some simple driver functions to configure the AVR128DB48s TWI0 module and to read the LM75's temperature. These tasks will help you become familiar with using the AVR128DB48's TWI module to implement a simple I2C protocol interface.

Design Tasks

Design Task 1: Using TWI0 Module to Provide an I2C Interface to the LM75.

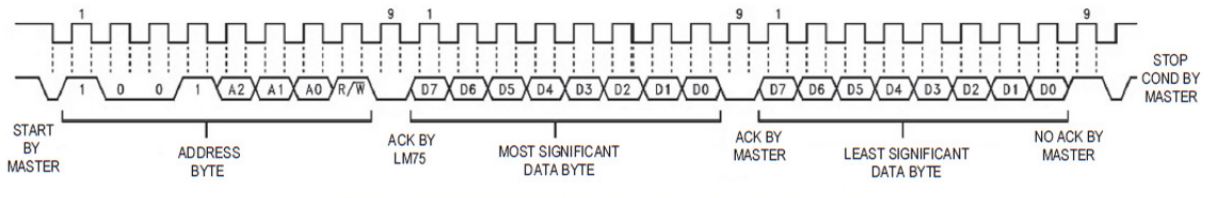
Study the data sheet for the LM75. This device has only an I2C interface. The AVR128DB48's TWI0 module must be configured to use this interface to communicate with the LM75. SDA must be driven by PA2 and SCL must be driven by PA3. PA2 and PA3 are the default SDA and SCL signals, respectively, for TWI0.

Draw a schematic diagram showing the wiring of the LM75 and its interface to the AVR128DB48. Compute and verify the voltage and current compatibility between the two devices using the checklist used before. Submit these items as part of your prelab.

Design Task 2: Program to Read the Temperature from an LM75

Write a function named `TWI0_LM75_init` that is passed no arguments and returns no value. This function initializes the AVR128DB48's TWI0 module to communicate with the LM75. The bit transfer rate between the AVR128DB48 and the LM75 must be as fast as possible, but less than or equal to 400 kb/s.

The overall objective of this task is just to read the temperature measured by the LM75. The I2C waveform to perform a two Data byte read of the LM75 follows.



(a) TYPICAL 2-BYTE READ FROM PRESET POINTER LOCATION SUCH AS TEMP, T_{HIGH}, T_{LOW}.

Write a function named `TWIO_LM75_read` that is passed one `uint8_t` argument. This argument's value is the LM75's 7-bit slave address and the R/$\overline{W}$ bit. This function returns a `uint16_t` that is the contents of the Temperature register with the don't care bits forced to 0s.

Write a program named `read_LM75_temp`. Your program must use the previously defined functions to read the Temperature register and compute the temperature. Use the following global variables to store the values read from the LM75's Temperature register and computed temperature.

```
uint8_t temp_reg_high;  //high byte of LM75 Temperature register
uint8_t temp_reg_low;   //low byte of LM75 Temperature register
uint16_t LM75_temp_reg = 0; //LM75 Temperature register contents
int16_t LM75_temp = 0;  //right adjusted 2's complement LM75 temp
```

Submit your program listing as part of your prelab. Make sure that your program includes program headers, function headers, and clear comments throughout.

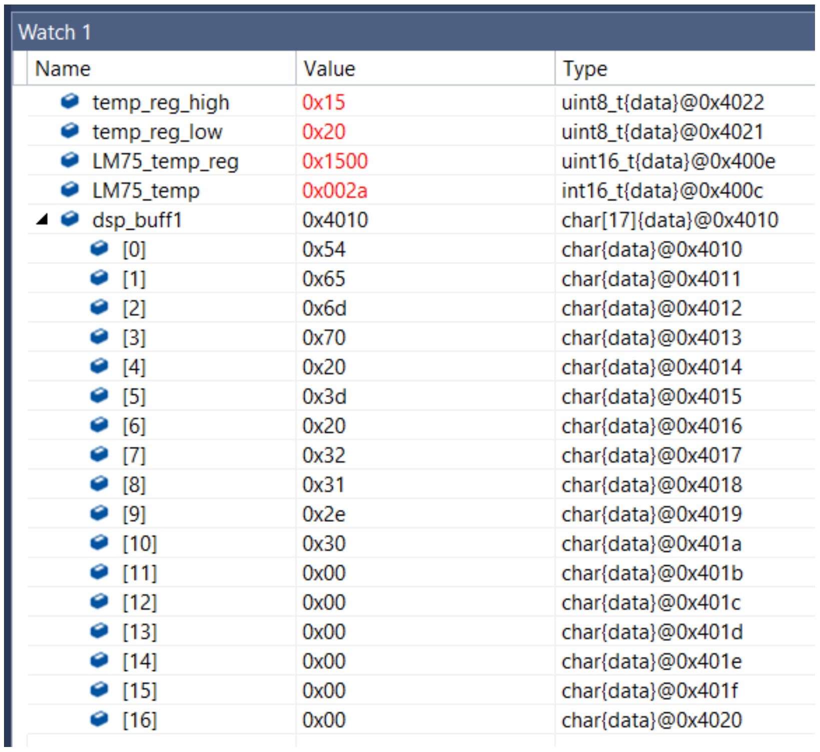
Design Task 3: Program to Read and Display the Temperature from an LM75

Write a program named `display_LM75_temp`. This program is an extension of the program of Task 2. It continually displays the LM75's temperature as a signed decimal value on line 1 of the LCD. Provide an appropriate prefix for the value displayed on the LCD. The program should display the temperature with a resolution of +/- 0.5 °C.

The temperature range of the LM75 is +125 to -55 °C. You can use the `sprintf` function to put a formatted string that displays the temperature into the LCD SRAM buffer for line 1 of the display. However, if you want to use floating point specifiers in the format string of your `sprintf`, you must change the Studio 7 linker settings in your project. The default settings in Atmel Studio 7 disable floating point for `sprintf` / `printf` type functions to save microcontroller flash memory. The prerequisite reading document *Print Floating Point Numbers in AVR C with Atmel Studio 7* tells you how to properly change Atmel Studio 7's linker settings.

Use the watch window to verify that your buffer contents are correct.

```
uint8_t temp_reg_high;  //high byte of LM75 Temperature register
uint8_t temp_reg_low;   //low byte of LM75 Temperature register
uint16_t LM75_temp_reg = 0; //LM75 Temperature register contents
int16_t LM75_temp = 0;  //right adjusted 2's complement LM75 temp
char dsp_buff1[17];     //buffer for line 1 of LCD image
```



Watch 1

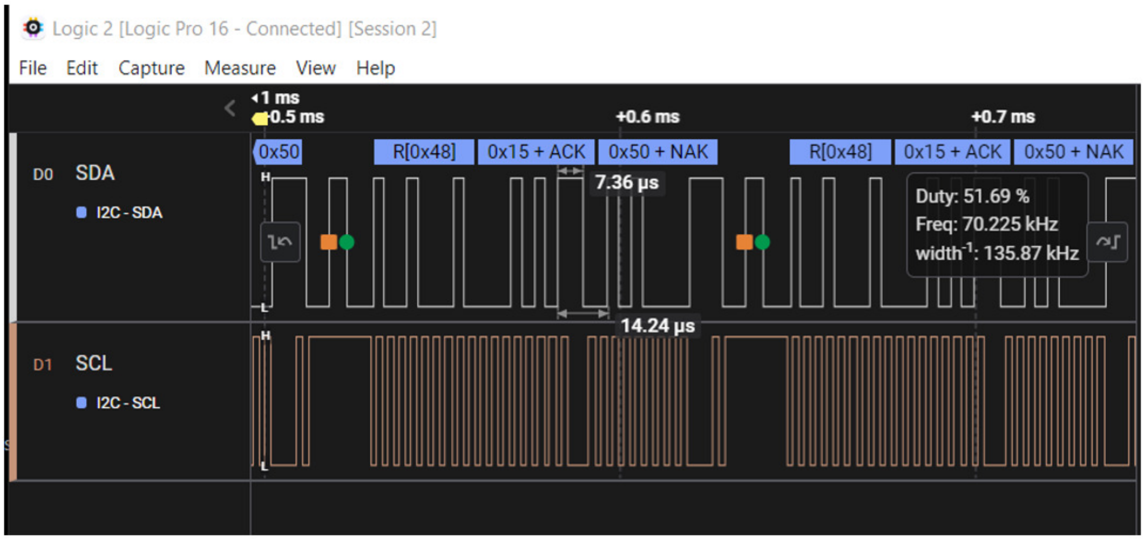
Name	Value	Type
temp_reg_high	0x15	uint8_t{data}@0x4022
temp_reg_low	0x20	uint8_t{data}@0x4021
LM75_temp_reg	0x1500	uint16_t{data}@0x400e
LM75_temp	0x002a	int16_t{data}@0x400c
dsp_buff1	0x4010	char[17]{data}@0x4010
[0]	0x54	char{data}@0x4010
[1]	0x65	char{data}@0x4011
[2]	0x6d	char{data}@0x4012
[3]	0x70	char{data}@0x4013
[4]	0x20	char{data}@0x4014
[5]	0x3d	char{data}@0x4015
[6]	0x20	char{data}@0x4016
[7]	0x32	char{data}@0x4017
[8]	0x31	char{data}@0x4018
[9]	0x2e	char{data}@0x4019
[10]	0x30	char{data}@0x401a
[11]	0x00	char{data}@0x401b
[12]	0x00	char{data}@0x401c
[13]	0x00	char{data}@0x401d
[14]	0x00	char{data}@0x401e
[15]	0x00	char{data}@0x401f
[16]	0x00	char{data}@0x4020

Laboratory Activities

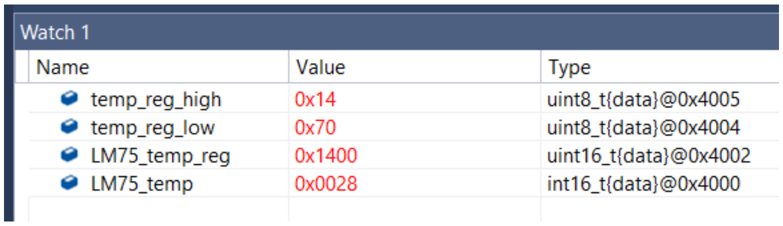
Laboratory Task 1: Using TWI0 Module to Provide an I2C Interface to the LM75
Wire the LM75 into your system.

Have a TA verify your wiring before applying power and get the TA's signature.

Laboratory Task 2: Program to Read the Temperature from an LM75
Create a project named `read_LM75_temp`. Add your program `read_LM75_temp` and compile it. Connect and configure the Saleae Logic Analyzer so that you can view the transactions on the I2C bus.



Run your program and verify that it operates properly. Verify that your readings of the LM75's Temperature register and your computation of the temperature are correct by displaying the appropriate variables in the Watch window.



Watch 1

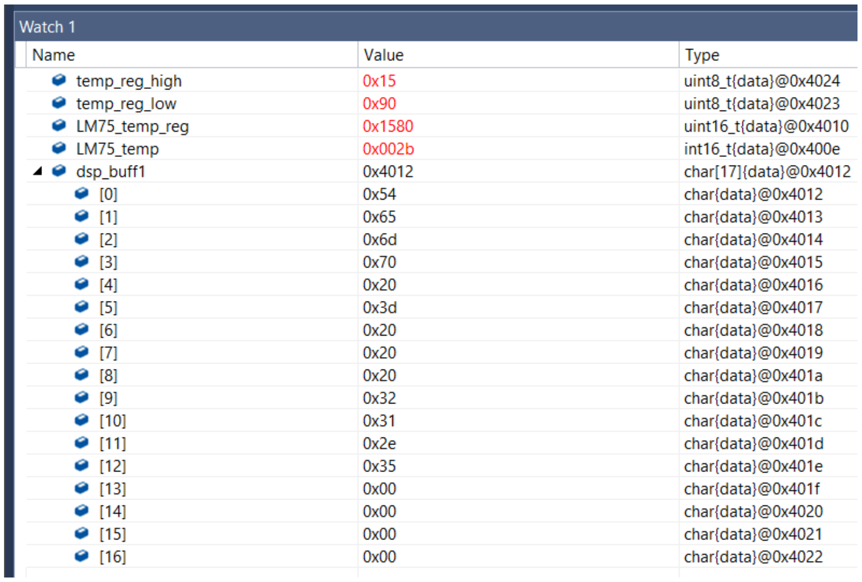
Name	Value	Type
temp_reg_high	0x14	uint8_t{data}@0x4005
temp_reg_low	0x70	uint8_t{data}@0x4004
LM75_temp_reg	0x1400	uint16_t{data}@0x4002
LM75_temp	0x0028	int16_t{data}@0x4000

If there are problems, identify their sources and correct them. Carefully use a "cold can" and heat gun to provide a range of temperatures to your LM75.

Have a TA verify the correct operation of your system and the correct waveforms on the Saleae Logic Analyzer and get the TA's signature.

Laboratory Task 3: Program to Read and Display the Temperature from an LM75
Create a project named `display_LM75_temp`. Add your program `display_LM75_temp` program and compile it. Run your program and verify that it operates properly. Your program

must display both positive and negative temperatures correctly on the LCD. Show a TA your Watch window showing the appropriate values for the variables and line 1 LCD buffer. Show that what is displayed on the LCD is consistent with the variable values.


Watch 1

Name	Value	Type
temp_reg_high	0x15	uint8_t{data}@0x4024
temp_reg_low	0x90	uint8_t{data}@0x4023
LM75_temp_reg	0x1580	uint16_t{data}@0x4010
LM75_temp	0x002b	int16_t{data}@0x400e
dsp_buff1	0x4012	char[17]{data}@0x4012
[0]	0x54	char{data}@0x4012
[1]	0x65	char{data}@0x4013
[2]	0x6d	char{data}@0x4014
[3]	0x70	char{data}@0x4015
[4]	0x20	char{data}@0x4016
[5]	0x3d	char{data}@0x4017
[6]	0x20	char{data}@0x4018
[7]	0x20	char{data}@0x4019
[8]	0x20	char{data}@0x401a
[9]	0x32	char{data}@0x401b
[10]	0x31	char{data}@0x401c
[11]	0x2e	char{data}@0x401d
[12]	0x35	char{data}@0x401e
[13]	0x00	char{data}@0x401f
[14]	0x00	char{data}@0x4020
[15]	0x00	char{data}@0x4021
[16]	0x00	char{data}@0x4022

Have a TA verify the correct operation of your system and get the TA's signature.

Leave the circuit you have constructed on your breadboard insert, it may be used in later laboratories.

Questions

1. What is the maximum `SCL` clock frequency for the LM75? Explain how you determined it. At what `SCL` frequency are you operating your LM75, explain how you know that.

2. What is the value to be loaded into the MBAUD register to get the desired maximum `SCL` clock frequency. Show your calculations.

3. What value should the DBGRUN bit have in the DBGCTRL register when you are debugging your software and why?

4. When viewing SDA and SCL on the Saleae Logic Analyzer, what do the red square and green circle represent?

5. In one short sentence, explain whether the master or the slave should be providing the acknowledge after a byte transfer.