Technologisches Gewerbe Museum

I2C

SYT IndInf

Siegel Hannah

January 15, 2015

Contents

1 Task Description

A project should be done, where the ADXL345 Sensor is sending Informations to the Controller using I2C. The 3 axes should be printed out using UART.

1.1 Team

We (Wolfang Mair, Christian Janeczeck and Andreas Vogt) met on Saturday, 10.01.2015 as a Team in the School.
We did the example together, because we were not able to do it by ourselves and we only had one Sensor (ADXL345). Therefore we started the example together. We have done it in this team, and the Code is not a copy from anybody else from our class. We have also all understood I2C. The code is not exactly the same, but it may be quite the same.
We thought, that it would be fine if we would form a team, and we hope that this is not a problem. We are happy to show the program working on each our stations if required to proove that we have all done it equally. Every Person from the Team has commented the source code by themselves, done the protocol by their own and finalized to code.

2 Working time

Estimated working time

Task	**Time in hours**
Setting up the working environments	1
Running any example	1
UART	1
I2C Implementation	4
Debuging & Testing	10
Documentation	1
Total	17 hours

Final working time

Task	**Time in hours**
Setting up the working environments	1
Running any example	0.5
UART	3
I2C Implementation	4
Debuging & Testing	3
Documentation	2
Total	13 hours

3 Working protocol

3.1 How to connect the ADXL345

Voltage Supply
"I2C and SPI digital communications are available. In both cases, the ADXL345 operates as a slave. I2C mode is enabled if the CS pin is tied high to VDD I/O. [..] ",[2, page 15]

Voltage Supply
" With CS tied high to VDD I/O, the ADXL345 is in I2 C mode, requiring a simple 2-wire connection, as shown in Figure 5 ",[2, page 18]. I connected the Clock with PB2 and the Data with PB3. I also connected the Voltage Supply with the +3.3V and the Ground to the controllers GND port.

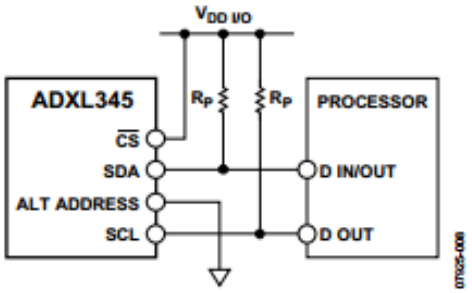

Figure 1: I2C Layout [2, page 18]

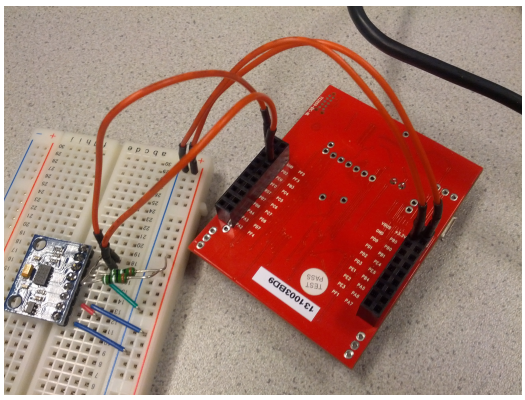

Figure 2: How to connect the wires

3.2 Setting up the working environement

I used the DebianVM we got in VMWare-Workstation. The Makefile could be found in the tiva-template [1]. At first I copied it into the github-repo. I also copied the `TM4C123GH6PM.ld` File.
I than copied the Files from the `/home/schueler/opt/tivaware/examples/peripherals/i2c` into the repository and changed the Makefile. I set the PROJ simply to src: `PROJ := src`.

3.3 UART

I than added only the UART. I wrote a method called `UARTConf`. In it, the necessary configurations for UART were done.

```
extern void UARTConf(void) {
   // Enable Ports for UART
   ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
   ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

   // Enable A0 and A1 for UART pins
   ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
   ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
   ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

   // Set Clock Source
   UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);

   // Config UART
   UARTStdioConfig(0, 115200, 16000000);
}
```

I than commented all the I2C parts out, only to test UART. I used `cutecom` to read the UART output. Somehow the UART didn't work.
I tried for a long time and finally we found an example in tivaware-template [1], `buttons_isr`. I changed the Makefile in the tiva-template so that this class will be executed. And here the UART worked fine. So I decided to simply use the `buttons_isr` as a scaffold. So everything that has something to do with the buttons was removed. The UART worked fine.

Later, while testing the I2C already, it stopped suddenly working, without that I had changed something. A restart of the system, the `cutecom` and plugging in and out the micro controller finally did work.

3.4 I2C

3.4.1 Addresses

"An alternate I2C address of 0x53", [2, page 18]. This means that the slave address can be found on 0x53. All the other addresses can be found on [2, page 22]:

```
0x00 0 DEVID R 11100101 Device ID
0x2D 45 POWER_CTL R/W 00000000 Power-saving features control
0x31 49 DATA_FORMAT R/W 00000000 Data format control
0x32 50 DATAX0 R 00000000 X-Axis Data 0
0x33 51 DATAX1 R 00000000 X-Axis Data 1
0x34 52 DATAY0 R 00000000 Y-Axis Data 0
0x35 53 DATAY1 R 00000000 Y-Axis Data 1
0x36 54 DATAZ0 R 00000000 Z-Axis Data 0
0x37 55 DATAZ1 R 00000000 Z-Axis Data 1
```

3.4.2 Basic I2C Communication

On page 315 and following it is explained how the I2C must be implemented. [3, page 315-334]

1. The code for setting up the InitI2C was taken from the `master_slave_loopback.c` example. I did not copy the `HWREG` line, because it was not working. Also all the loopback settings were not copied.
   Vennesa then told me, that I had to add the command `GPIOPinTypeI2CSCL` as well, so I did.

```
extern void I2CConf(void){
  // Enable the Ports
  SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

  // Configure Port B2 onto the Clock
  GPIOPinConfigure(GPIO_PB2_I2C0SCL);

  // Configure Port B3 onto the Data
  GPIOPinConfigure(GPIO_PB3_I2C0SDA);

  // Configure Port B
  GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3);

  // Configure Port B for the Clock
  GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);

  // Set up the I2C Clock
  I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), NORMAL_CLK);
}
```

2. "the user must first initialize the I2C master module with a call to `I2CMasterInitExpClk()`",[3, page 315]

```
I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);
```

[3, page 343]

3. "Data is transferred by first setting the slave address using `I2CMasterSlaveAddrSet()`. ",[3, page 315] I set the Slave Adress to 0x53, because this was what I have found out previously. "That function is also used to define whether the transfer is a send (`write = false, read = true`)",[3, page 315]

```
I2CMasterSlaveAddrSet(I2C0_BASE, 0x53, false);
```

4. "Then, if connected to an I2C bus that has multiple masters, the Tiva I2C master must first call `I2CMasterBusBusy()` before attempting to initiate the desired transaction.",[3, page 315] But because I only have one master this is not needed.

```
while(I2CMasterBusy(I2C0_BASE)) {}
```

5. "After determining that the bus is not busy, if trying to send data, the user must call the `I2CMasterDataPut()` function. ",[3, page 315] First I wanted to read the slave ID.

```
I2CMasterDataPut(I2C0_BASE, 0x00);
```

6. "After determining that the bus is not busy, if trying to send data, the user must call the `I2CMasterDataPut()` function. ",[3, page 315] First I wanted to read the slave ID.

```
I2CMasterDataPut(I2C0_BASE, 0x00);
```

7. "The transaction can then be initiated on the bus by calling the `I2CMasterControl()` function with any of the following commands:
`I2C_MASTER_CMD_SINGLE_SEND`
`I2C_MASTER_CMD_SINGLE_RECEIVE`
`I2C_MASTER_CMD_BURST_SEND_START`
`I2C_MASTER_CMD_BURST_RECEIVE_START`." ,[3, page 315]
I used the `I2C_MASTER_CMD_SINGLE_SEND` command.

```
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
```

8. "For the single send and receive cases, the polling method involves looping on the return from `I2CMasterBusy()`. ",[3, page 316]

```
while(I2CMasterBusy(I2C0_BASE)) {}
```

9. " Once that function indicates that the I2C master is no longer busy, the bus trans- action has been completed and can be checked for errors using `I2CMasterErr()`. ",[3, page 316]

```
int8_t error = I2CMasterErr(I2C0_BASE);
```

10. " If there are no errors, then the data has been sent or is ready to be read using `I2CMasterDataGet()`. ",[3, page 316]

```
data = (int8_t) I2CMasterDataGet(I2C0_BASE);
```

So the complete code is to read a value is:

```
extern int8_t read(uint32_t slaveAddress, uint32_t registerAddress){
    // Set the Slave Address and write
    I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, WRITE);

    // Set the Register / Data to get
    I2CMasterDataPut(I2C0_BASE, registerAddress);

    // Sending
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);

    // Waiting until the Master is done
    while(I2CMasterBusy(I2C0_BASE)) {}

    // Get error code, if there is one
    int8_t error = I2CMasterErr(I2C0_BASE);
    if (error != I2C_MASTER_ERR_NONE){
        UARTprintf("\n error-code : %i \n",error);
    }

    // Set the Slave Address and read
    I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, READ);

    // Reiciving Data
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);

    // Waiting until the Master is done
    while(I2CMasterBusy(I2C0_BASE)) {}

    // Get error code, if there is one
    error = I2CMasterErr(I2C0_BASE);
    if (error != I2C_MASTER_ERR_NONE){
        UARTprintf("\n error-code : %i \n",error);
    }

    // Fetching the Data out of the register
    int8_t data = (int8_t) I2CMasterDataGet(I2C0_BASE);

    // Return the data
    return data;
}
```

With this code, only the Slave's ID was read. (Using 0x00).

For reading the values, the read method could be used, only with different Addresses.

3.4.3 Power Mode

I wrote a function for the Power Mode. It first writes a Register (The Register for the Power mode) and then it writes a second value which turned the Power Mode on.

" After VS is applied, the device enters standby mode, where power consumption is minimized and the device waits for VDD I/O to be applied and for the command to enter measurement mode to be received. (This command can be initiated by setting the measure bit (Bit D3) in the POWER_CTL register (Address 0x2D).)"[2, page 13].
Therefore I set the 0x2D to 0x3F.
Information about the Register 0x2D—POWER_CTL can also be seen on page 25.

```
extern void writePowermode(uint32_t slaveAddress,uint32_t registerAddress,uint32_t
    data){
    // Set the Slave Address and write
    I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, WRITE);

    // Send the Register
    I2CMasterDataPut(I2C0_BASE, registerAddress);

    // Sending
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);

    // Waiting until the Master is done
    while(I2CMasterBusy(I2C0_BASE)) {}

    // Get error code, if there is one
    int8_t error = I2CMasterErr(I2C0_BASE);
    if (error != I2C_MASTER_ERR_NONE)
              UARTprintf("\n error-code : %i \n",error);

    // Set the Slave Address and write
    I2CMasterSlaveAddrSet(I2C0_BASE, slaveAddress, WRITE);

    // Send the Power Mode Command
    I2CMasterDataPut(I2C0_BASE, data);

    // Sending
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISHED);

    // Waiting until the Master is done
    while(I2CMasterBusy(I2C0_BASE)) {}

    // Get error code
    error = I2CMasterErr(I2C0_BASE);
    if (error != I2C_MASTER_ERR_NONE)
              UARTprintf("\n error-code : %i \n",error);
}
```

3.4.4 Reading the Values

Reading the Values were easy. I used the read method. After having set the power mode, this has not been any problem. I looped in the main method:

```
while(1){
      // Print the x Values
      UARTprintf("Slave said: X %i \n", shift(read(SLAVE_ADDRESS,
          DATAX0),read(SLAVE_ADDRESS, DATAX1)));

      // Print the y Values
      UARTprintf("Slave said: Y %i \n", shift(read(SLAVE_ADDRESS,
          DATAY0),read(SLAVE_ADDRESS, DATAY1)));

      // Print the z Values
      UARTprintf("Slave said: Z %i \n \n ", shift(read(SLAVE_ADDRESS,
          DATAZ0),read(SLAVE_ADDRESS, DATAZ1)));

      // Sleep
      ROM_SysCtlDelay(ROM_SysCtlClockGet() / 2);
   }
```

The values ware Bitshifted using the shift method: Markus Klein helped me with this.

```
extern int16_t shift(int8_t axis1,int8_t axis2){
  return((int16_t) axis2 << 8) | ((int16_t) axis1);
}
```

3.5 Testing

```
Starting the program ...
Slave ID: -27

Slave said: X 10
Slave said: Y 0
Slave said: Z -40

 Slave said: X 10
Slave said: Y 0
Slave said: Z -30
```

Figure 3: Startup

```
 Slave said: X -10
Slave said: Y -36
Slave said: Z -104

 Slave said: X -256
Slave said: Y -78
Slave said: Z -46

 Slave said: X -68
Slave said: Y 94
Slave said: Z 92
```

Figure 4: Some Values

```
    error-code : 12

    error-code : 8
   Slave said: X 0
   Slave said: Y 2
   Slave said: Z -30
```

Figure 5: Errors (I plugged out some wires)

4 Problems

4.1 Build Failed

Sometimes I received this error message: I was not able to solve it, but it was during using the Loopback example.

```
build/slave_receive_int.o:/home/schueler/repositories/tiva-template/src/i2c2/slave_receive_int.
    first defined here
arm-none-eabi-ld: warning: cannot find entry symbol ResetISR; defaulting to 00000000
Makefile:67: recipe for target 'build/main' failed
make: *** [build/main] Error 1
```

4.2 Flashing didn't work

Sometimes I got the following error, even trough the cutecom was able to use the device and the lsusb was showing it. It was solved when plugging out the phone which was charging in the meantime...

```
lm4flash build/main.bin
Unable to get device serial number: LIBUSB_ERROR_TIMEOUT
Unable to find any ICDI devices
```

4.3 While loop is not stopping

I have had the problem that the while-loop within the read() function was an infinite loop, which means that the Master is always busy. The problem was, that the Voltage cable was not in the port anymore, it was getting out unintended.

4.4 Are the values correct?

I had no way to verify the values, and they seemed a little bit odd too, because they were not 0 even trough the sensor was not moving. This problem has not been solved.
Also, many students from the class were multiplication the output by 4.
I have found the following section in the Datasheet concerning this issue, but to be honest I didnt understand what this precisely means, so I just let the values as they were.

"Self-test change is defined as the output (g) when the `SELF_TEST` bit = 1 (in the `DATA_FORMAT` register, Address 0x31) minus the output (g) when the `SELF_TEST` bit = 0. Due to device filtering, the output reaches its final value after 4 × t when enabling or disabling self-test, where t = 1/(data rate). The part must be in normal power operation (`LOW_POWER` bit = 0 in the `BW_RATE` register, Address 0x2C) for self-test to operate correctly. "[2, page 5]

4.5 Controller stopped working the next day

When I tried out the example again the next day, there were no values any more. Vennesa told me, that this was because the power mode was not set any more, which means, that the power mode function was not writing as it should have in the first place (because it got set by another code). The power mode is therefore set for a longer time, which means that the code actually didnt set it and therefore he code was wrong.
I solved this bug when I wrote with a `BURST SEND` instead of the `SINGLE SEND`.

References

[1] **Tiva Template Github**, Michael Borko
https://github.com/mborko/tiva-template
last used: 11.01.2015, 10:34

[2] **Data Sheet ADXL345**
http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf
last used: 10.01.2015, 10:36

[3] **TivaWare™ Peripheral Driver Library**
https://github.com/mborko/tiva-template/blob/master/docs/SW-TM4C-DRL-UG-2.0.1.11577.pdf
last used: 11.01.2015, 10:33