Lab 6 I2C

EE138

Colin Chen & Daniel Graham

5/12/2015

Introduction

In the I2C lab, we will be using the I2C communication protocol to complete two different tasks.

- Communicate with a Digital to Analog chip to power an LED
- Communicate with a digital display to have it display lettered text

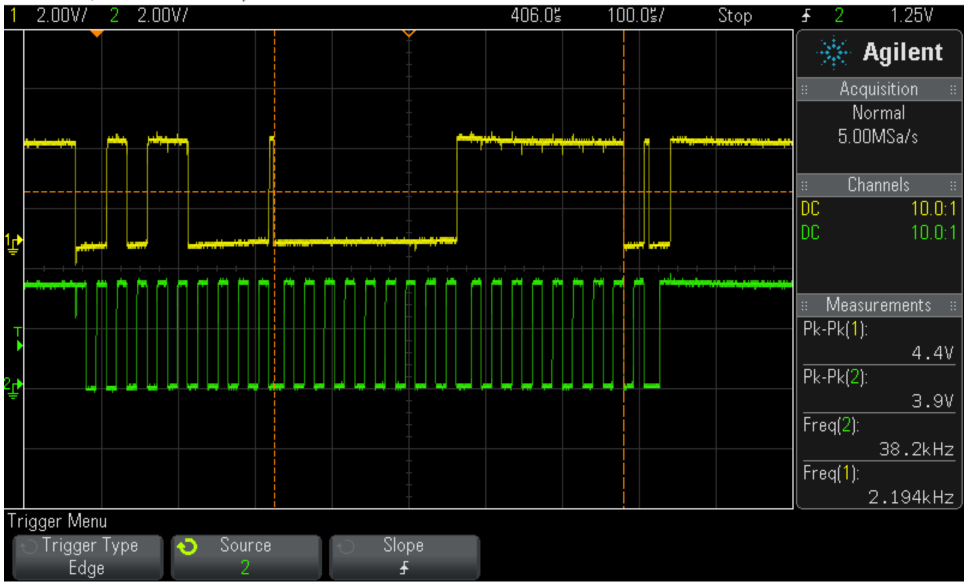
Many electrical applications and peripherals that are in close proximity to the main controller use I2C as a form of communication. I2C heavily reduces the amount of connections needed to communicate with multiple peripherals as it only requires two wires. In addition, the limitation of the number of peripherals is also quite large due to the availability of 8 bits worth of addressability.

Procedure

I2C is a serial communication protocol that relies only on two lines of connection. The two connections lines, Serial Data Line (SDA) and Serial Clock Line (SCL) allows for one master (or in some cases several) to connect to as many peripherals as the master can address with its 8 bit register. The I2C protocol for writing, as our lab only has the master writing to the slave, consists of starting with a start bit, followed by address, command, data, and then a stop bit. Using this method, we can communicate with both the DAC peripheral and the lettered display. For the SAMD20, we will be using the SERCOM I2C interrupt to handle all I2C communication. The start bit will be initialized by writing of the address and the interrupt will also be triggered by the start bit as well. Using the smart mode of the SAMD20, the acknowledge statements from the slave will automatically be registered allowing for simplified coding. After the address is sent, the command bit is sent followed by all the data required by the slave peripheral to operate. For the DAC, this data will be a value between 0x0 and 0xFFFFFFFF with 0 being the lowest value and 0xFFFFFFFF the max. The text display is a little more complex in setting up

Results

DSO-X 2022A, MY52010608: Tue May 12 02:43:46 2015



Both objectives of the lab were successfully completed. The above waveform shows the address, acknowledge, command, and data being sent in respect to the clock line. For the DAC, we were able to turn the LED on, signifying that data was being written to the DAC and that it was executing the data properly. We were able to properly modify the intensity of the LED by changing the value of the 8 bit data being sent. For the display, the text of our names were split up by letter and we mapped out the corresponding hex values into an array. We then sent these values as 8 bit data into the display similar to how we sent data in the DAC. The display code was a little more complex due to the display needing to receive more data but other than setting up the display and sending the corresponding data, this lab was not too difficult.

Code

```
/**

 * \file

 *

 * Lab #6 EE138

 *

 */

#include <asf.h>

#include <clock.h>

#include <conf_clocks.h>

Sercom *I2C_ptr = 0x42000800;
```

```
int global_array[50] =
{0x44,0x61,0x6E,0x69,0x65,0x6C,0x02,0x47,0x72,0x61,0x68,0x61,0x6D,0x02,0x02,0x02,0x02,0x02,0x02,
0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0
x02,0x02,0x43,0x6F,0x6C,0x69,0x6E,0x02,0x43,0x68,0x65,0x6E};

volatile int state=0;

volatile int init_state=0;

volatile int global_state = 0;

volatile bool hasRun = false;

void checkIt(void){

Port *ports = PORT_INSTS;

PortGroup *porA = &(ports->Group[0]);

porA->OUTTGL.reg = PORT_PA13;

}



void init_ports(void){

Port *ports = PORT_INSTS;

PortGroup *porA = &(ports->Group[0]);

PortGroup *porB = &(ports->Group[1]);

porA->PMUX[4].bit.PMUXO = 0x2;

porA->PINCFG[9].bit.PMUXEN = 0x1;

porA->PMUX[4].bit.PMUXE = 0x2;

porA->PINCFG[8].bit.PMUXEN = 0x1;

porB->DIRSET.reg = PORT_PB30;

porA->DIRSET.reg = PORT_PA13;

porB->OUTSET.reg = PORT_PB30;

}



void init_clock(void){
```

```
PM->APBCMASK.reg |=0x1 << 2;

uint32_t temp=0x0D; // ID for ________ is __________ (see table 14-2)

temp |= 0<<8; // Selection Generic clock generator 0

GCLK->CLKCTRL.reg=temp; // Setup in the CLKCTRL register

GCLK->CLKCTRL.reg |= 0x1u << 14; // enable it.

}

void init_I2C(void){

I2C_ptr ->I2CM.CTRLA.reg = 0x0 <<1;

I2C_ptr ->I2CM.CTRLA.reg |= 0x05 << 2;

I2C_ptr ->I2CM.BAUD.bit.BAUD = 100;

I2C_ptr ->I2CM.CTRLA.bit.SDAHOLD = 0x2;

// I2C_ptr ->I2CM.CTRLB.bit.CMD = 0x2;

I2C_ptr ->I2CM.INTENSET.bit.MB = 0x1;

I2C_ptr ->I2CM.CTRLB.bit.SMEN = 0x1;

while(I2C_ptr -> I2CM.STATUS.reg & SERCOM_I2CM_STATUS_SYNCBUSY){}

I2C_ptr ->I2CM.CTRLA.reg |= 0x1 << 1;

while(I2C_ptr -> I2CM.STATUS.reg & SERCOM_I2CM_STATUS_SYNCBUSY){}

I2C_ptr ->I2CM.STATUS.bit.BUSSTATE = 0x1;

NVIC_EnableIRQ(SERCOM0_IRQn);

}


void init_LCD(){

switch(init_state){

case 1:

I2C_ptr ->I2CM.DATA.reg = 0x00;

break;

case 2:

while(I2C_ptr -> I2CM.STATUS.reg & SERCOM_I2CM_STATUS_SYNCBUSY){}
```

```
I2C_ptr ->I2CM.DATA.reg = 0x38;

break;

case 3:

while(I2C_ptr -> I2CM.STATUS.reg & SERCOM_I2CM_STATUS_SYNCBUSY){}

I2C_ptr ->I2CM.DATA.reg = 0x39;

break;

case 4:

while(I2C_ptr -> I2CM.STATUS.reg & SERCOM_I2CM_STATUS_SYNCBUSY){}

I2C_ptr ->I2CM.DATA.reg = 0x14;

break;

case 5:

while(I2C_ptr -> I2CM.STATUS.reg & SERCOM_I2CM_STATUS_SYNCBUSY){}

I2C_ptr ->I2CM.DATA.reg = 0x78;

break;

case 6:

while(I2C_ptr -> I2CM.STATUS.reg & SERCOM_I2CM_STATUS_SYNCBUSY){}

I2C_ptr ->I2CM.DATA.reg = 0x5E;

break;

case 7:

while(I2C_ptr -> I2CM.STATUS.reg & SERCOM_I2CM_STATUS_SYNCBUSY){}

I2C_ptr ->I2CM.DATA.reg = 0x6D;

break;

case 8:

while(I2C_ptr -> I2CM.STATUS.reg & SERCOM_I2CM_STATUS_SYNCBUSY){}

I2C_ptr ->I2CM.DATA.reg = 0x0F;

break;

case 9:

while(I2C_ptr -> I2CM.STATUS.reg & SERCOM_I2CM_STATUS_SYNCBUSY){}

I2C_ptr ->I2CM.DATA.reg = 0x01;
```

```
break;

case 10:

while(I2C_ptr -> I2CM.STATUS.reg & SERCOM_I2CM_STATUS_SYNCBUSY){}

I2C_ptr ->I2CM.DATA.reg = 0x06;

break;

case 11:

I2C_ptr ->I2CM.CTRLB.bit.CMD = 0x03;

hasRun = true;

break;

}

init_state++;

}




SERCOM0_Handler(void){

  /*******************

  LCD DISPLAY INITIALIZATION

  ********************/


if(global_state == 0){

init_LCD();

}

if(global_state == 1){

if(state ==0){

while(I2C_ptr -> I2CM.STATUS.reg & SERCOM_I2CM_STATUS_SYNCBUSY){}

I2C_ptr ->I2CM.DATA.reg = 0x40;

state++;
}
if(state<=51){
    while(I2C_ptr -> I2CM.STATUS.reg & SERCOM_I2CM_STATUS_SYNCBUSY){}
    I2C_ptr ->I2CM.DATA.reg = global_array[state-2];
//   I2C_ptr ->I2CM.DATA.reg = 0x41;
}


if(state==52){
I2C_ptr ->I2CM.CTRLB.bit.CMD = 0x03;
}
    state++;



}



}



int main (void)
{
system_clock_init();
init_ports();
init_clock();
init_I2C();
init_state = 0;
global_state = 0;
I2C_ptr ->I2CM.ADDR.reg = 0x3C<<1;
while(!hasRun){};
```

```
global_state++;

I2C_ptr ->I2CM.ADDR.reg = 0x3C<<1;

checkIt();

while(1){

}

// Insert application code here, after the board has been initialized.

}
```