Laboratory log book Robotics

Joris Stork: 6185320 Lucas Swartsenburg: 6174388
Sander van Veen: 6167969

January 18, 2011

Contents

1 I^{2}C and the CSS Compiler

1.1 Objective

Our objective is to implement an I^{2}C bus using: a 3-state word generator, an I/O expander and a microcontroller (these are described below). The I^{2}C bus is a two-wire serial bus used for data transport between integrated circuits.

1.2 Method

Our implementation of the I^{2}C bus involves transporting 8 parallel bits of data - generated by a human operator via the interface of a word generator - over from the I/O expander using the 2 bit I^{2}C protocol to the microcontroller, and back again. We programmed the microcontroller in a C-like language and used the provided C API (functions `i2c_read` and `i2c_write`) to act as the master on the I^{2}C bus, with the I/O expander as slave (see the appendix for code). We connected the microcontroller's C4 and C3 pins with the I/O expander's SDA (serial data line) and SCL (serial clock line) pins respectively. The I/O expander's eight left-hand side pins were connected to the corresponding eight pins of the word generator. The program loaded onto the microcontroller initiates the transfer on the bus, first setting the SDA signal to "low", then indicating the transmission direction (from slave to master) and the address of the slave (corresponding to the left-hand pins of the I/O expander) from which to send the data. Once the microcontroller had received the data, the program initiated a second transfer in the opposite direction, sending the same data it had received back to the I/O expander for display on the I/O expander's right-hand LEDs.

1.2.1 Hardware

- PCF8574 I^{2}C I/O expander integrated circuit (IC). Each of this I/O expander's eight pins can be used as an input or output. In order to use a pin as input, a 1 has to be written to that pin's register.
- 16F876 microcontroller: this is an 8 bit microcontroller that we load with programs written in mplab ...
- 74LS244 word (8 bit) generator. Buttons toggle each of the eight pins representing the 8 bits of output.

1.2.2 Software

The follows software runs on an i86 Windows XP machine which is connected to the microcontroller using a serial port cable.

- Tera Term Web 3.1 : used as a terminal console for the microcontroller.
- MPLAB IDE v6.30
- PIDAC 876 programmer: Loads the compiled program onto the microcontroller using the serial port.

1.3 results

The experiment worked as expected. The `0b11000110` was transmitted and returned successfully, as shown in the image below:

Figure 1: Photo set-upI^2C

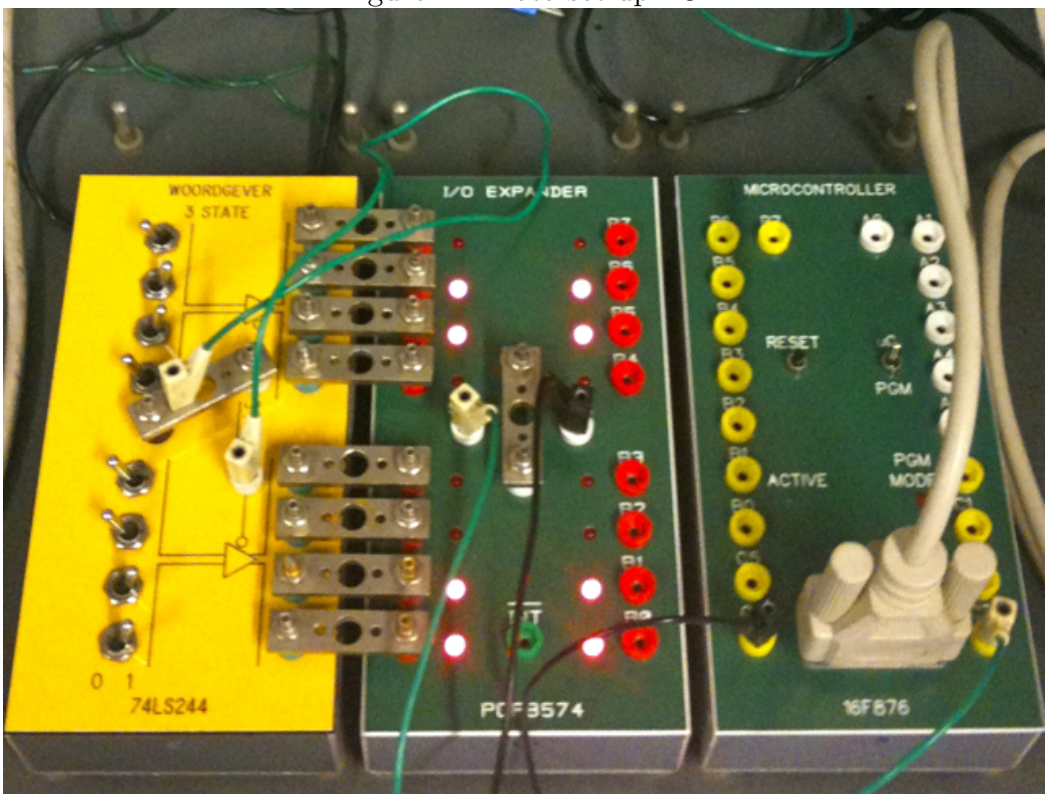
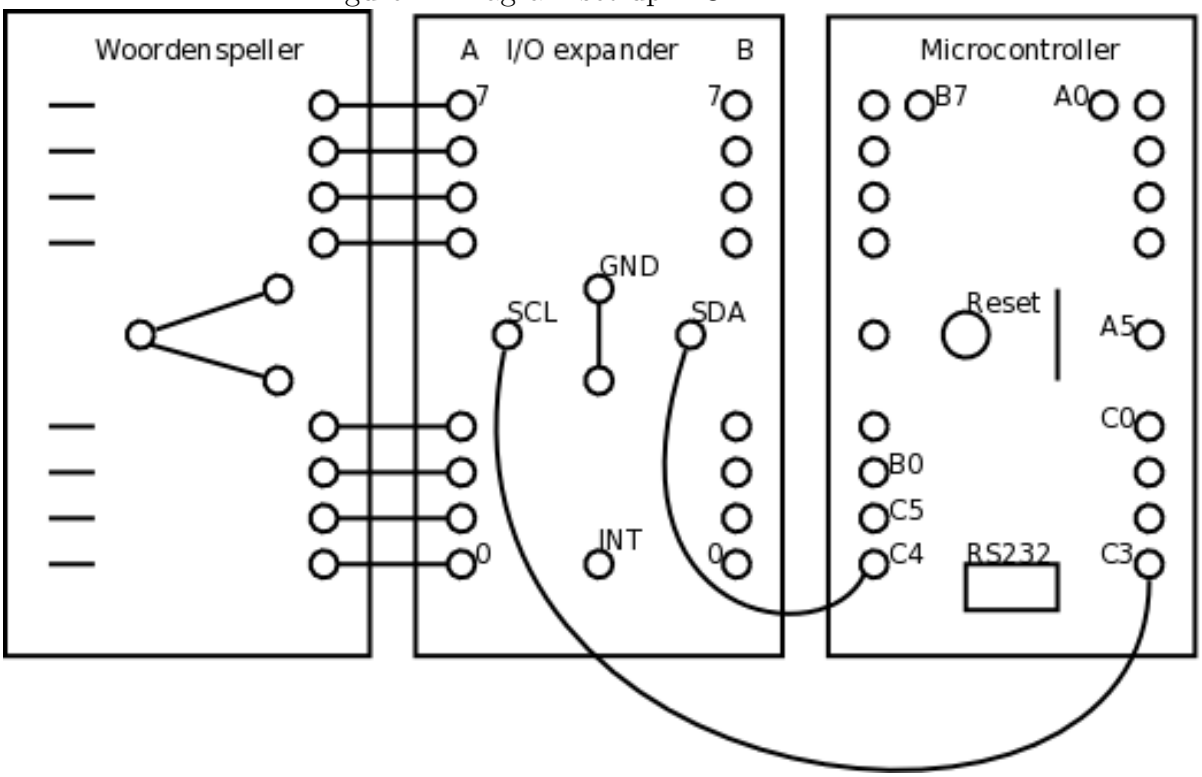


Figure 2: Diagram set-up I^2C



1.4 assumptions

We didn't make any assumptions, so nothing useful to mention here.

2 SAR ADC

2.1 Objective

Our objective is to convert a analog signal to a digital signal and to test how accurate and what the range of our equipment is. We also wan't to find out what the relation is between the voltage and the digital value that is produced. This experiment is in preparation of the next one.

In this experiment we assume that the relation between the voltage and the digital value is 0.40v/bit because that is what is written on the outside of the equipment. assume t

2.2 Method

We attach a SAR (successive approximation register) to the comperator. The SAR will conduct the comparator so that the current voltage can be found with a binary search. The results of the comparing operation will be written to a I/O expander. It wasn't part of this assignment, but (like shown in the previous assignment) it is possible to attach the I/O expander using a I^2C bus to a microcontroller. This way we can read out the values on the computer screen. The potentiometer has 10 positions. We shall test the voltage and digital value on each position.

2.3 Equipment

2.3.1 Hardware

- PCF8574 I^2C I/O expander integrated circuit (IC).
- 4 bit Value generator.
- Pulse generator (clock). We used one wich had different frequencies, but we liked using the 1Mhz clock.
- SAR (successive approximation register). A SAR is a IC that conducts succesive approximation using a comperator.
- Comperator. The comperator also has a digital to analog converter on board. The SAR gives a digital signal to the comperator and the comperator produces a currancy. This currancy is compared to the currancy on the input of the comperator. If the analog input currancy

is higher than the produced currancy, the comperator will return a 1 to the SAR. Otherwise, a 0.

- Potentiometer. On the potentiometer there are 11 spots, marked from 0 to 10. On each of these spots the potentiometer will give a certain currance. This currancy will grow exponetially going from 0 to 10.
- A voltage meter. We used a voltage meter to measure the currancy coming the potentiometer. Its purpuse in this setup is to check the relation between the digital signal from the SAR and the voltage.
- (Optional) 16F876 microcontroller: this is an 8 bit microcontroller that we load with programs written in mplab ...

2.3.2 Software

For this assignment we don't need to use any software. Only when we want to read out the digital values on a computer screen, we can use the code of the previous assignment (I^2C).

- Tera Term Web 3.1 : used as a terminal console for the microcontroller.
- MPLAB IDE v6.30
- PIDAC 876 programmer: Loads the compiled program onto the microcontroller using the serial port.

2.4 Raw results

Potentiometer	Voltage (Volt)	Binary	Decimal	Value / Voltage
0	0.00	0000 0000	0	0.0000
1	0.02	0000 0001	1	0.0200
2	0.09	0000 0010	2	0.0450
3	0.29	0000 0011	3	0.0966
4	0.52	0000 0110	6	0.0866
5	0.74	0000 1111	15	0.0493
6	0.98	0001 1000	24	0.0408
7	1.38	0001 1111	31	0.0445
8	2.19	0011 0011	51	0.0429
9	3.42	0100 1111	79	0.0432
10	4.77	0111 0001	113	0.0422

Table 1: Potentiometer's value compared the measured voltage and binary value.

2.5 Results

From the table above we can conclude that the relation is 0.40v/bit. At the lower voltages we can see some inaccurate results. But when we come up to 1.00 Volt, we can clearly see the 0.40v/bit relation.

The SAR has 256 bits, and we just showed that there is a 0.40v/bit relation, so the range of the SAR is about: $256 * 0.04V = 10.24V$.

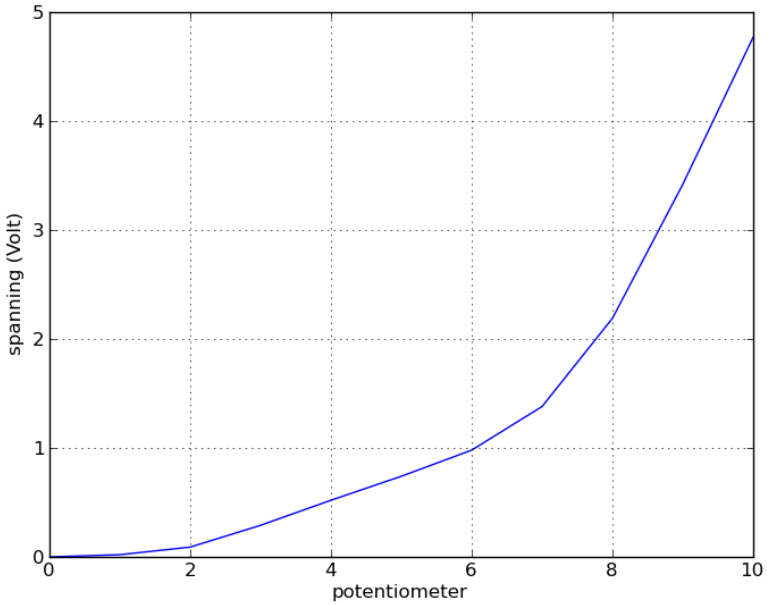
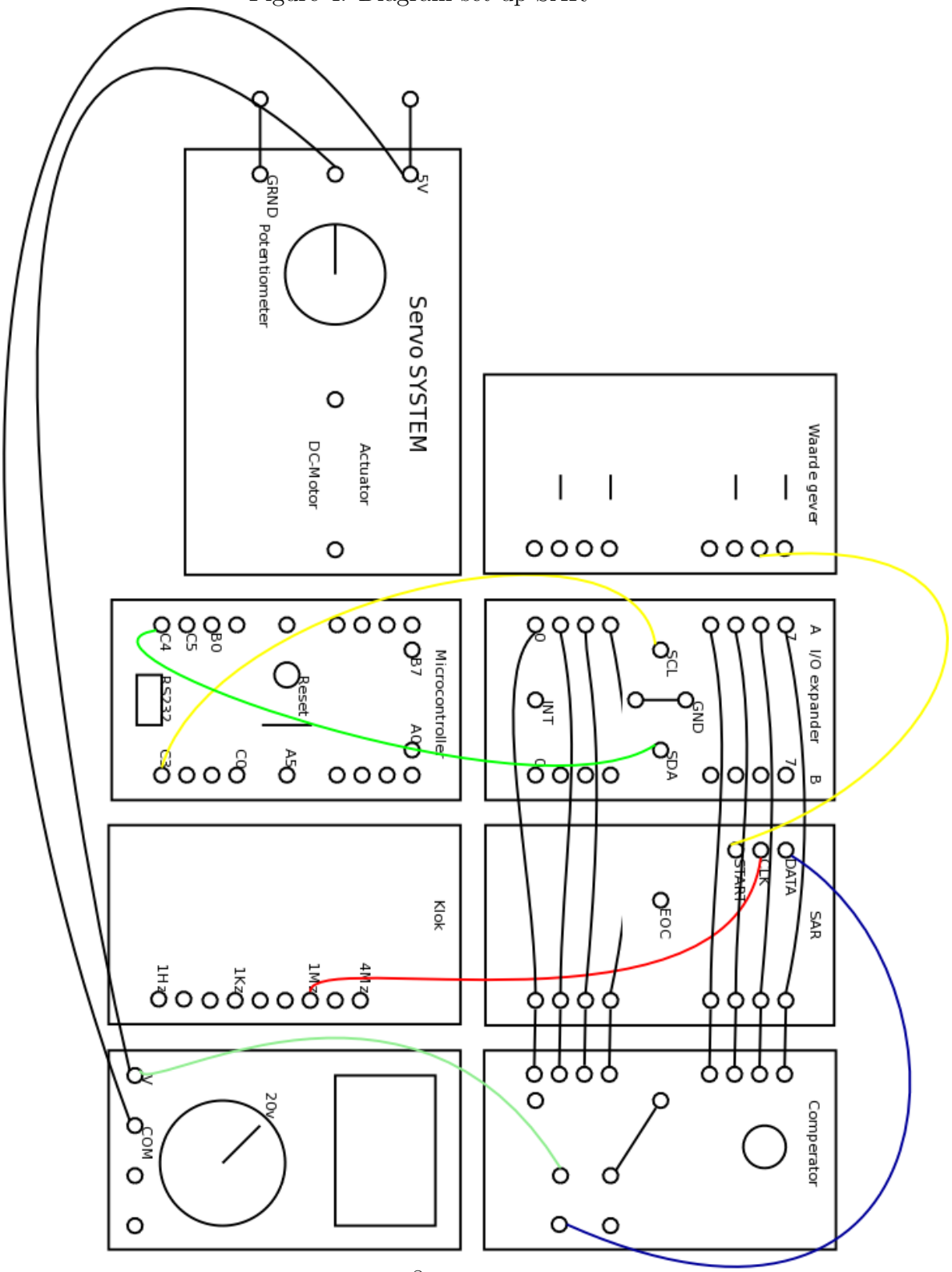


Figure 3: Values of the potentiometer compared the corresponding voltage.

Figure 4: Diagram set-up SAR



2.6 Assumptions

Our assumption turned out the be almost right. After doing some tests we found out that especially with a higher voltage, the relation is 0.40v/bit.

3 Servo System

3.1 Objective

Our objective is to build a system that controls the position of a servo. The system also needs to detect errors (when the servo didn't reach its destination due to some reason) and needed a abort button.

Doing this assignment will learn us how to handle and program hardware and information coming from hardware.

We also started working on a GUI that controls the servo, but due to lack of time we weren't able to finish this.

3.2 Method

We first start by modifying the setup of the previous assingment. We need a Pidac module to power the server and we need to add the microcontroller to control the system. We hook up the Pidac module to the servo and the microcontroller to the Pidac module. The microcontroller also needs to be connected to the I/O expander via a I^2C bus and it needs to be connected to the start port of the SAR. We use a word generator to control to which position the servo needs to go and to put the Pidac module and the servo on and off.

After taking care of the harware bit, we program the microcontroller to receive a wanted position and the current position and to turn the servo in a way that the servo will reach the wanted position.

3.3 Equipment

3.3.1 Hardware

- PCF8574 I^2C I/O expander integrated circuit (IC).
- Pulse generator (clock). We used one wich had different frequencies, but we liked using the 1Mhz clock.
- SAR (successive approximation register). A SAR is a IC that conducts succesive approximation using a comperator.
- Comperator. The comperator also has a digital to analog converter on board. The SAR gives a digital signal to the comperator and the

comperator produces a currancy. This currancy is compared to the currancy on the input of the comperator. If the analog input currancy is higher than the produced currancy, the comperator will return a 1 to the SAR. Otherwise, a 0.

- Potentiometer. On the potentiometer there are 11 spots, marked from 0 to 10. On each of these spots the potentiometer will give a certain currance. This currancy will grow exponetially going from 0 to 10.
- Servo. A Servo that turns around the nob of the potentiometer.
- A voltage meter. We used a voltage meter to measure the currancy coming the potentiometer.
- 16F876 microcontroller: this is an 8 bit microcontroller that we load with programs written in mplab ...
- 74LS244 word (8 bit) generator. Buttons toggle each of the eight pins representing the 8 bits of output.
- L293D PIDAC-module. The Pidac module powers the servo.

3.3.2 Software

The follows software runs on an i86 Windows XP machine which is connected to the microcontroller using a serial port cable.

- Tera Term Web 3.1 : used as a terminal console for the microcontroller.
- MPLAB IDE v6.30
- PIDAC 876 programmer: Loads the compiled program onto the microcontroller using the serial port.

3.4 Raw results

After implementing the hardware and programming the micocontroller, the servo was able to stop on 4 given positions and give an error when the motor got stuck.

3.5 Results

The function to let the motor stop on 4 positions is implemented.

Figure 5: Diagram set-up Servo SYSTEM

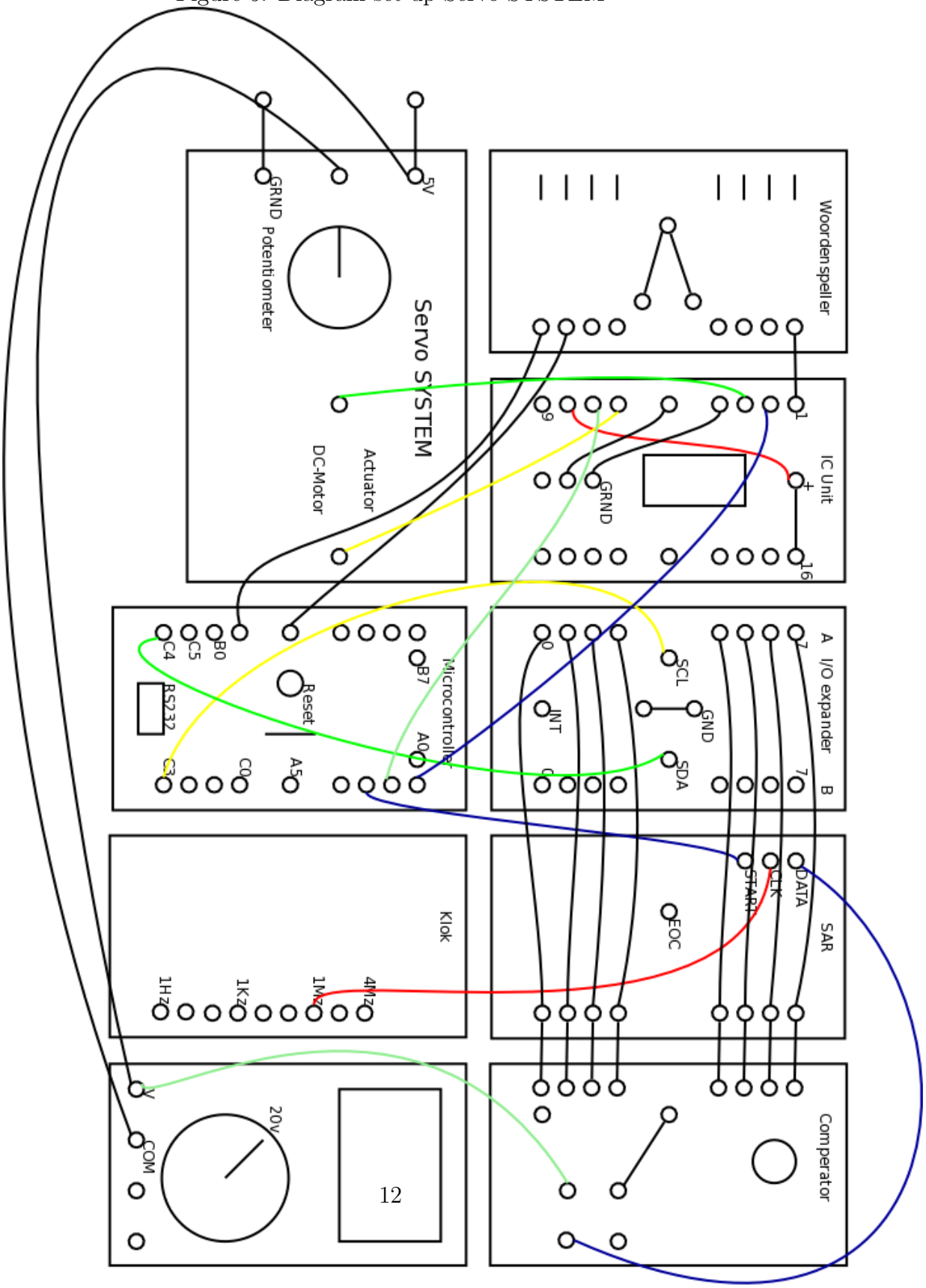


Figure 6: Part one of setup

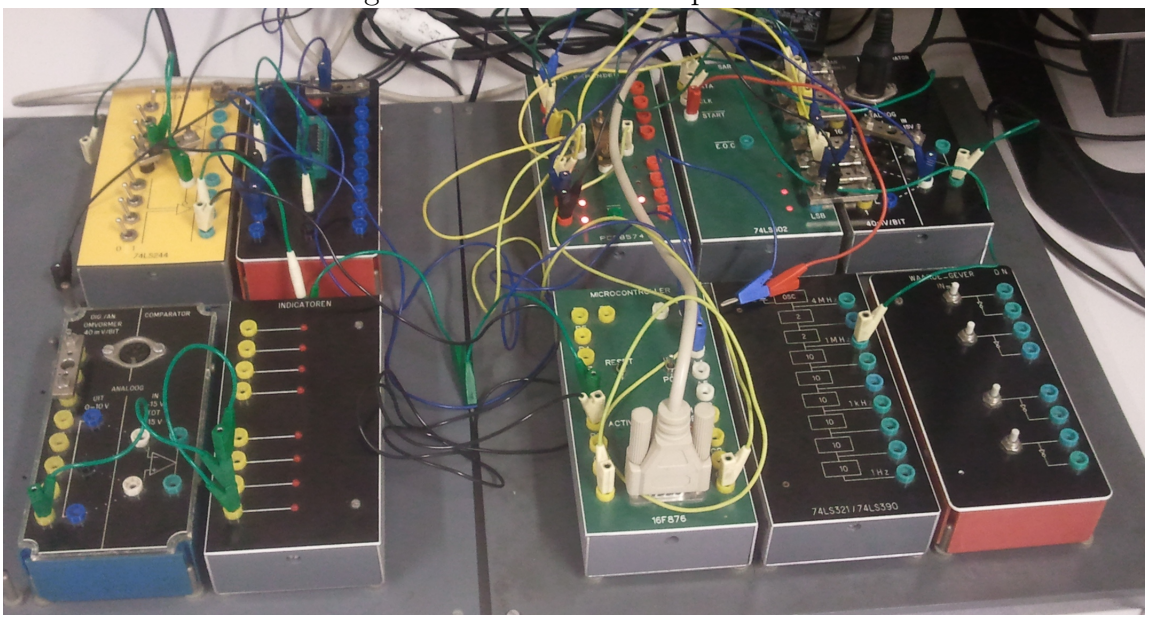
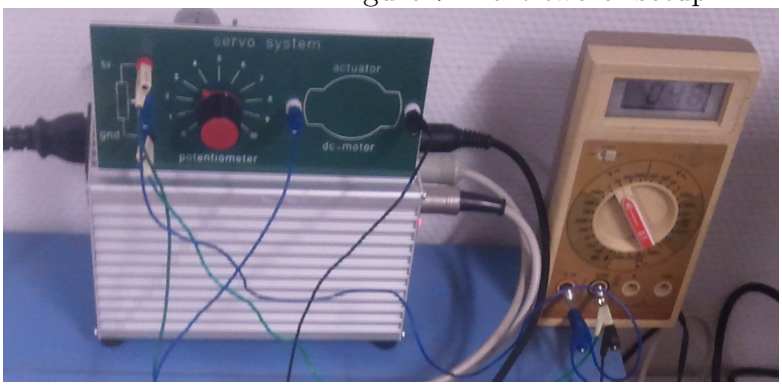


Figure 7: Part two of setup



3.6 Assumptions

There were no assumptions.

A Source: I^{2}C

```
////////////////////////////////////////////////////////////////////////
//
// Filename      :     Test876.c
// Revision      :     1.0
// Created       :     19-3-2001
// Revised     :      26-11-2003 by Benb

// Project       :     Pidac876
// Device         :     PIC16F876
// Development     :     MPLAB / CCS PCM
// Author         :     E. Steffens
// Department     :     Faculty of science
// Copyright     :     Universiteit van Amsterdam
//Description     :     Testing serial connection with PC
////////////////////////////////////////////////////////////////////////

#include <C:\Program Files\PICC\Devices\16F876.H>
#include <C:\Program Files\PICC\Drivers\CTYPE.H>

// Inform the compiler the clock frequency is 8 MHz
#use delay(clock=8000000)

// Setup the RS232 communication
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7,  bits=8)

//Setup the I2C bus
#use I2C (Master, SDA=PIN_C4, SCL=PIN_C3, SLOW)

int main(){
    int Getal = 0;
    int Getal_1 = 0;
    printf("Assignment 4 \n\r");

    //Voorbeeld: Byte van Master ? Slave met adres 1

    while(1){
        //Voorbeeld: Byte van Slave met adres 0 ? Master
        i2c_start();
```

```
        i2c_write( 0x41 );//write address (01000001)
        Getal = i2c_read(0); // acknowledge
        i2c_stop();

        i2c_start();
        i2c_write( 0x42 );//write address (01000010)
        i2c_write( Getal );     //write byte
        i2c_stop();
        if(Getal!=Getal_1){
                printf("%d \n\r", Getal);
                Getal_1 = Getal;
        }
    }
    return 0;
}
```

B Source: Servo System

```
////////////////////////////////////////////////////////////////////////
//
// Filename      :     Test876.c
// Revision      :     1.0
// Created       :     19-3-2001
// Revised     :      26-11-2003 by Benb

// Project       :     Pidac876
// Device         :     PIC16F876
// Development     :     MPLAB / CCS PCM
// Author         :     E. Steffens
// Department     :     Faculty of science
// Copyright     :     Universiteit van Amsterdam
//Description     :     Testing serial connection with PC
////////////////////////////////////////////////////////////////////////

#include <C:\Program Files\PICC\Devices\16F876.H>
#include <C:\Program Files\PICC\Drivers\CTYPE.H>

// Inform the compiler the clock frequency is 8 MHz
#use delay(clock=8000000)

// Setup the RS232 communication
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7,  bits=8)

//Setup the I2C bus
#use I2C (Master, SDA=PIN_C4, SCL=PIN_C3, SLOW)

int main(){
    int Getal = 0, success =0, state = 0;
    unsigned long int count = 0;
    unsigned long int check_count = 0;
    printf("Servo \n\r");
    while(1){
        check_count++;
        count++;

        // Receives the data (current position) from the I2C.
```

```
if((count >= 0) && (count <= 4)){
    // Start the SAR
    OUTPUT_HIGH(PIN_A3);
}
else if (count >= 7){
    count = 0;
    i2c_start();
    i2c_write( 0x41 );//write address (01000001)
    Getal = i2c_read(0); // acknowledge
    i2c_stop();

    i2c_start();
    i2c_write( 0x42 );//write address (01000010)
    i2c_write( Getal );     //write byte
    i2c_stop();

    OUTPUT_LOW(PIN_A3);
}else {
    i2c_start();
    i2c_write( 0x41 );//write address (01000001)
    Getal = i2c_read(0); // acknowledge
    i2c_stop();

    i2c_start();
    i2c_write( 0x42 );//write address (01000010)
    i2c_write( Getal );     //write byte
    i2c_stop();
}

// DEBUG: Print the count so that we can find out what is happening.
printf("count %ld \n\r", count);

// Case position 0 (B1: 0 and B2: 0)
if (INPUT(PIN_B1)==0 && INPUT(PIN_B2)==0){
    if(state!=0){
        check_count = 0;
        state = 0;
    }
    success = 0;
    // Postion has been reached, turn off the servo
    if (count > 5 && Getal > 0 && Getal <=2){
```

```
            success = 1;
            check_count = 0;
            OUTPUT_LOW(PIN_A1);
            OUTPUT_LOW(PIN_A2);
        }
        // If "Getal" is smaller than this position requires, make sure
        // servo turns right.
        else if((count > 5) && (Getal < 1)){
            OUTPUT_LOW(PIN_A1);
            OUTPUT_HIGH(PIN_A2);
        }
        // If "Getal" is bigger than this position requires, make sure
        // servo turns left.
        else if ((count > 5) && (Getal > 1)){
            success = 2;
            OUTPUT_LOW(PIN_A2);
            OUTPUT_HIGH(PIN_A1);
        }

    }
    // Case position 1 (B1: 1 and B2: 0)
    if (INPUT(PIN_B1)==1 && INPUT(PIN_B2)==0){
        if(state!=1){
            check_count = 0;
            state = 1;
        }
        success = 0;
        // Postion has been reached, turn off the servo
        if (count > 5 && Getal > 8 && Getal <=12){
            success = 1;
            check_count = 0;
            OUTPUT_LOW(PIN_A1);
            OUTPUT_LOW(PIN_A2);
        }
        // If "Getal" is smaller than this position requires, make sure
        // servo turns right.
        else if((count > 5) && (Getal < 10)){
            OUTPUT_LOW(PIN_A1);
            OUTPUT_HIGH(PIN_A2);
        }
        // If "Getal" is bigger than this position requires, make sure
```

```
    // servo turns left.
    else if ((count > 5) && (Getal > 10)){
        OUTPUT_LOW(PIN_A2);
        OUTPUT_HIGH(PIN_A1);
    }
}
// Case position 2 (B1: 0 and B2: 1)
if (INPUT(PIN_B1)==0 && INPUT(PIN_B2)==1){
    if(state!=2){
        check_count = 0;
        state = 2;
    }

    success = 0;
    // Postion has been reached, turn off the servo
    if (count > 5 && Getal > 35 && Getal <=55){
        success = 1;
        check_count = 0;
        OUTPUT_LOW(PIN_A1);
        OUTPUT_LOW(PIN_A2);
    }
    // If "Getal" is smaller than this position requires, make sure
    // servo turns right.
    else if((count > 5) && (Getal < 40)){
        OUTPUT_LOW(PIN_A1);
        OUTPUT_HIGH(PIN_A2);
    }
    // If "Getal" is bigger than this position requires, make sure
    // servo turns left.
    else if ((count > 5) && (Getal > 40)){
        check_count = 0;
        OUTPUT_LOW(PIN_A2);
        OUTPUT_HIGH(PIN_A1);
    }
}
// Case position 3 (B1: 1 and B2: 1)
if (INPUT(PIN_B1)==1 && INPUT(PIN_B2)==1){
    if(state!=3){
        check_count = 0;
        state = 3;
    }
```

```
            success = 0;
            // Postion has been reached, turn off the servo
            if (count > 5 && Getal > 80 && Getal <=110){
                success = 1;
                check_count = 0;
                printf("%ld \n\r", check_count);
                OUTPUT_LOW(PIN_A1);
                OUTPUT_LOW(PIN_A2);
            }
            // If "Getal" is smaller than this position requires, make sure
            // servo turns right.
            else if((count > 5) && (Getal < 100)){
                OUTPUT_LOW(PIN_A1);
                OUTPUT_HIGH(PIN_A2);
            }
            // If "Getal" is bigger than this position requires, make sure
            // servo turns left.
            else if ((count > 5) && (Getal > 100)){
                OUTPUT_LOW(PIN_A2);
                OUTPUT_HIGH(PIN_A1);
            }
        }

        /*
         * If the state hasn't changed or if a destination position hasn't
         * been reached in withing 600 counts, shut the program down.
         */
        if(check_count > 600){
            printf("ERROR: motor stuck\n\r");
           OUTPUT_LOW(PIN_A1);
           OUTPUT_LOW(PIN_A2);
           break;
        }
        printf("Getal %d \n\r", Getal);
    }
    return 0;
}
```