Lab 5

September 17, 2021
Delivery date: September 17, 2021

I2C Interfacing with C

Javier Mondragon Martin del Campo

A01365137

Prof. Matías Vázquez Piñón
Tecnológico de Monterrey

1 Terminal commands to obtain data from the RTC on part I.61

To set the components in different commands the following lines were used:

```
sudo ./i2c -s104 -dw -c2500 -ib 2 0 0
sudo ./i2c -s104 -dw -c2500 -ib 2 1 85
sudo ./i2c -s104 -dw -c2500 -ib 2 2 0
sudo ./i2c -s104 -dw -c2500 -ib 2 3 5
sudo ./i2c -s104 -dw -c2500 -ib 2 3 59
sudo ./i2c -s104 -dw -c2500 -ib 2 5 9
sudo ./i2c -s104 -dw -c2500 -ib 2 6 33
```

Setting the RTC with one command (tested but not used initially) the following command could be used:

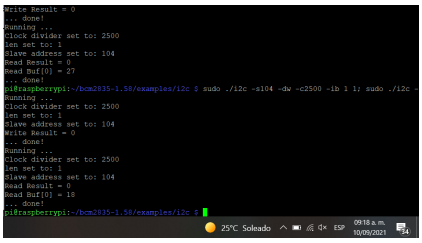
```
sudo ./i2c -s104 -dw -c2500 -ib 8 0 0 85 0 5 59 9 33
```

For reading the register (being the register X), the command used was:

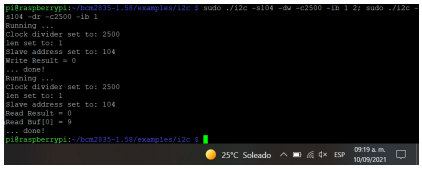
```
sudo ./i2c -s104 -dw -c2500 -ib 1 X; sudo ./i2c -s104 -dr -c2500 -ib 1
```

2 Demostration

(a) Second



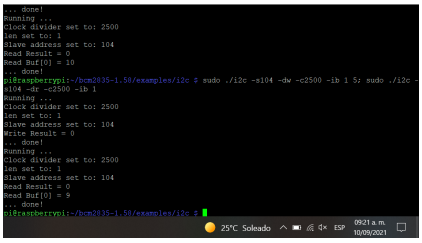
(b) Minute



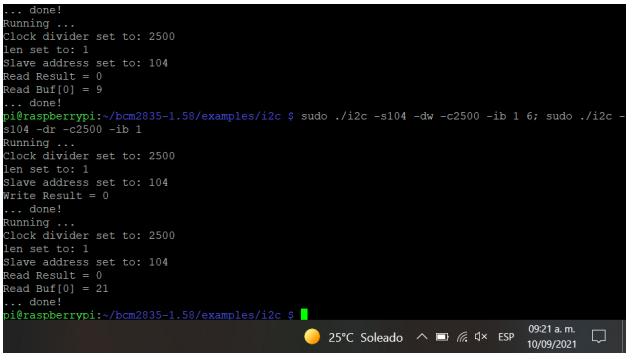
(c) Hour

(d) Day of the week

(e) Day of the month



(f) Hour



(g) Year

Figure 1: Output from Part 1

3 Terminal commands to build and execute the program in C, writen on Part II.1.1

```
/************************************************************************
*
*   i2c.c
*
*   Copyright (c) 2013 Shahrooz Shahparnia (sshahrooz@gmail.com)
*
*   Description:
*   i2c is a command-line utility for executing i2c commands with the
*   Broadcom bcm2835.  It was developed and tested on a Raspberry Pi s
*   computer model B.  The utility is based on the bcm2835 C library d
*   by Mike McCauley of Open System Consultants, http://www.open.com.a
*
*   Invoking spincl results in a read or write I2C transfer.
Options include the
*   the I2C clock frequency, read/write, address, and port initializat
*   procedures.  The command usage and command-line parameters are des
*   in the showusage function, which prints the usage if no command-li
*   are included or if there are any command-line parameter errors.
Invoking i2c
*   requires root privilege.
*
*   This file contains the main function as well as functions for disp
*   usage and for parsing the command line.
*
*   Open Source Licensing GNU GPLv3
*
*   Building:
* After installing bcm2835, you can build this
* with something like:
* gcc -o i2c i2c.c -l bcm2835
* sudo ./i2c
*
* Or you can test it before installing with:
* gcc -o i2c -I ../../src ../../src/bcm2835.c i2c.c
```

```
* sudo ./i2c
*
*   History:
*   11/05     VERSION 1.0.0: Original
*
*       User input parsing (comparse) and showusage\
*       have been adapted from: http://ipsolutionscorp.com/raspberry-pi
*       mostly to keep consistence with the spincl tool usage.
*
*       Compile with: gcc -o i2c i2c.c bcm2835.c
*
*       Examples:
*
*            Set up ADC (Arduino: ADC1015)
*            sudo ./i2c -s72 -dw -ib 3 0x01 0x44 0x00 (select config re
*            sudo ./i2c -s72 -dw -ib 1 0x00 (select ADC data register)
*
*            Bias DAC (Arduino: MCP4725) at some voltage
*            sudo ./i2c -s99 -dw -ib 3 0x60 0x7F 0xF0 (FS output is wit
*            Read ADC convergence result
*            sudo ./i2c -s72 -dr -ib 2 (FS output is 0x7FF0 with PGA1 =
*
*       In a DAC to ADC loop back typical results are:
*
*       DAC     VOUT    ADC
*       7FFh    1.6V    677h                    Note ratio is FS_ADC*PGA_
*       5FFh    1.2V    4DCh
*       8F0h    1.8V    745h
*       9D0h    2V      7EAh
*       000h    10mV    004h
*
**************************************************************************

#include <bcm2835.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
```

```
#include <time.h>

#define MODE_READ 0
#define MODE_WRITE 1

#define MAX_LEN 32

#define RTC_ADDR 104
#define TC74_ADDR 77

#define CLK_DIV 2500

char wbuf[MAX_LEN];

// 01/01/01 Mon 12:00:00 AM
uint8_t rtc_config[8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x01, 0x0
// DATA: SEC, MIN, HOUR, WEEKDAY, DAY, MONTH, YEAR
uint8_t rtc_buff[7];

uint8_t tc74_buff[1];

typedef enum {
    NO_ACTION,
    I2C_BEGIN,
    I2C_END
} i2c_init;

uint8_t  init = NO_ACTION;
uint16_t clk_div = BCM2835_I2C_CLOCK_DIVIDER_148;
uint8_t slave_address = 0x00;
uint32_t len = 0;
uint8_t  mode = MODE_READ;

char buf[MAX_LEN];
int i;
uint8_t data;
int n_records;
char file_buff[3][42];
```

```
char file_buff_t[3][28];

int RTC_init();
int RTC_get();
int TC74_get();
void delay_ms();
char* weekday(int);
void logFile();

int main(int argc, char **argv) {

  printf("Running ... \n");

  RTC_init();
  while(1){
    RTC_get();
    TC74_get();
    sprintf(file_buff_t[n_records%3], "RECEIVER> Temperature: %d°C\n",
    printf("RECEIVER> Temperature: %d°C\n", tc74_buff[0]);
    sprintf(file_buff[n_records%3], "RECEIVER> Record %d: %02X/%02X/%0
    logFile();
    n_records++;
    delay_ms();
  }
    return 0;
}

int RTC_init(){

  if (!bcm2835_init())
  {
    printf("bcm2835_init failed. Are you running as root??\n");
    return 1;
  }

  if (!bcm2835_i2c_begin())
  {
    printf("bcm2835_i2c_begin failed. Are you running as root??\n");
```

```
    return 1;
  }

  bcm2835_i2c_setSlaveAddress(RTC_ADDR);
  bcm2835_i2c_setClockDivider(CLK_DIV);

  data = bcm2835_i2c_write(rtc_config, 8);

  bcm2835_i2c_end();
  bcm2835_close();

  return data;

}

int RTC_get(){
  if (!bcm2835_init())
  {
    printf("bcm2835_init failed. Are you running as root??\n");
    return 1;
  }

  if (!bcm2835_i2c_begin())
  {
    printf("bcm2835_i2c_begin failed. Are you running as root??\n");
    return 1;
  }

  bcm2835_i2c_setSlaveAddress(RTC_ADDR);
  bcm2835_i2c_setClockDivider(CLK_DIV);

  data = bcm2835_i2c_write(&rtc_config[0], 1);

  data = bcm2835_i2c_read(rtc_buff, 7);

  bcm2835_i2c_end();
  bcm2835_close();
  printf("RECEIVER> Record %d: %02X/%02X/%02X %s %02X:%02X:%02X\n", n_
```

```
  return data;
}

int TC74_get(){
  if (!bcm2835_init())
  {
    printf("bcm2835_init failed. Are you running as root??\n");
    return 1;
  }

  if (!bcm2835_i2c_begin())
  {
    printf("bcm2835_i2c_begin failed. Are you running as root??\n");
    return 1;
  }

  bcm2835_i2c_setSlaveAddress(TC74_ADDR);
  bcm2835_i2c_setClockDivider(CLK_DIV);

  data = bcm2835_i2c_read(tc74_buff, 1);

  bcm2835_i2c_end();
  bcm2835_close();

  return tc74_buff[0];
}

void delay_ms()
{
    int milli_seconds = 1000 * 10000;
    clock_t start_time = clock();
    while (clock() < start_time + milli_seconds){
      if(TC74_get()>30)
        break;
    }
}

char* weekday(int day){
```

```
  switch (day)
  {
  case 0:
    return "Mon";
  case 1:
    return "Tue";
  case 2:
    return "Wed";
  case 3:
    return "Thu";
  case 4:
    return "Fri";
  case 5:
    return "Sat";
  case 6:
    return "Sun";
  default:
    return "ERR";
  }
}

char date_log[20], temp_log[28];

void logFile(){
  FILE *fp;
  fp = fopen("output.txt","w");
  if(fp){
    for(int i=0; i<3 && i<= n_records; i++){
      snprintf(date_log, 42, "%s", &file_buff[i]);
      snprintf(temp_log, 28, "%s", &file_buff_t[i]);
      fprintf(fp, "%s\n", temp_log);
      fprintf(fp, "%s\n", date_log);
    }
    fclose(fp);
  }
}
```

Output from the program:

```
RECEIVER> Temperature: 15°
RECEIVER> Record 3: 01/01/01 Mon 00:00:30
RECEIVER> Temperature: 19°
RECEIVER> Record 4: 01/01/01 Mon 00:00:40
RECEIVER> Temperature: 21°
RECEIVER> Record 5: 01/01/01 Mon 00:00:0A
```

(The image of the console directly was deleted from my computer by Visual Studio Code and I could not recover it)

4 Link to your GitHub repository containing the codes for Lab work, including parts I and II

`https://github.com/javiermomc/Sistemas_Embebidos/tree/main/L05`

5 Conclusions

The most challenging part was to find out how the i2c program work and figure out how to give the parameters like the clock divider or the values which have to be in decimal instead of hexadecimal. An application I propose this circuit may be useful for a greenhouse, recording the temperature with this embedded system and giving an alert if it surpasses the 30 degrees. A common chore that can be replaced with an embedded system may be the coffe machine, at certain time it start making the coffee and it may require a mosfet to start the warming process and directly a pump with a i2c module to start the brewing.

6 Bibliography

1. https://github.com/matias-vazquez/SistemasEmbebidos