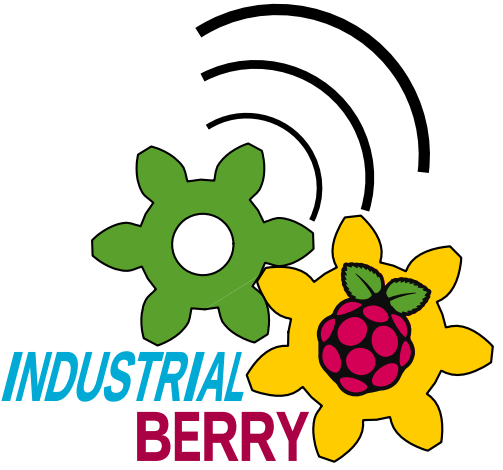
CanBarry PI V 1.0



www.industrialberry.com

August 2013

Contents

1 License **1**

2 Introduction **3**

3 Hardware implementation **5**

4 Software implementation **9**

4.1 Real Time Clock 9

4.1.1 RTC with shell 9

4.1.2 RTC with a compiled C code 11

5 Components list **15**

Bibliography **17**

List of Figures

2.1 CanBerry PI on Raspberry 4

3.1 Electric diagram of CanBus block 6
3.2 Electric diagram of RTC block 7
3.3 CanBerry Connector 7

List of Tables

5.1 CanBerry Pi V 1.0 16

Chapter 1

License

Open-source hardware shares much of the principles and approach of free and open-source software. In particular, we believe that people should be able to study our hardware to understand how it works, make changes to it, and share those changes. To facilitate this, we release all of the original design files (Eagle CAD) for the IndustrialBerry hardware. These files are licensed under a Creative Commons Attribution Share-Alike license, which allows for both personal and commercial derivative works, as long as they credit IndustrialBerry and release their designs under the same license. The IndustrialBerry software/firmware is also open-source.

Chapter 2

Introduction

CanBerry Pi V 1.0 is an extension board for RaspBerry Pi. It is an Open Hardware Design. It has two functionalities: a can bus module and an onboard Real Time clock powered by a 12 mm battery. In fig 2.1 is shown the Board on Raspberry PI.

The CanBus is based on MCP2515 [1] SPI controller and the MCP2551 [2] tranceiver. All functionalities are full integrated in standard linux kernel, so, they can be avaible on fly, or at last recompiling linux kernel to add canbus functionalities. The real time clock is based on PCF8523 [3] I2C controller. It is full compatible with linux too. Using I2C Kernel module, and standard kernel functions, date and hour can be set/get by simple commands. On the bottom side is located an on board battery to guarantee a data autonomy more than 20 years. In chapter hardware there are all informations on principal components, schematics to rebuild and modify RaspBerry PI board. In chapter Software is reported how all hardware can be used: as recompile kernel, build simple user space function to set and get I2C data, etc... In chapter application is reported a typical example of how to use the board.

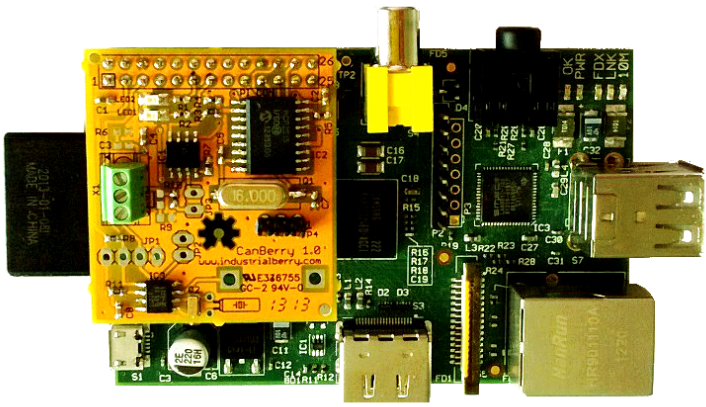


Figure 2.1: CanBerry PI on Raspberry

Chapter 3

Hardware implementation

CanBarry PI 1.0 is composed bu two blocks: a CanBus Module shown in Fig. 3.1 and a Real Time Clock shown in 3.2

MCP2515 is a stand alone SPI canbus controller full integrated in linux kernel. At the start, the driver was implemented as a block device. Recently it is assumed to be a network module into the kernel. It is supplied by 3.3V from raspberry connector (fig 3.3). MCP2551 is supplied by 5V from Raspberry connector insted. So, to match voltage physical level between the two chips, a voltage matching made by R3 and R4 has been used.

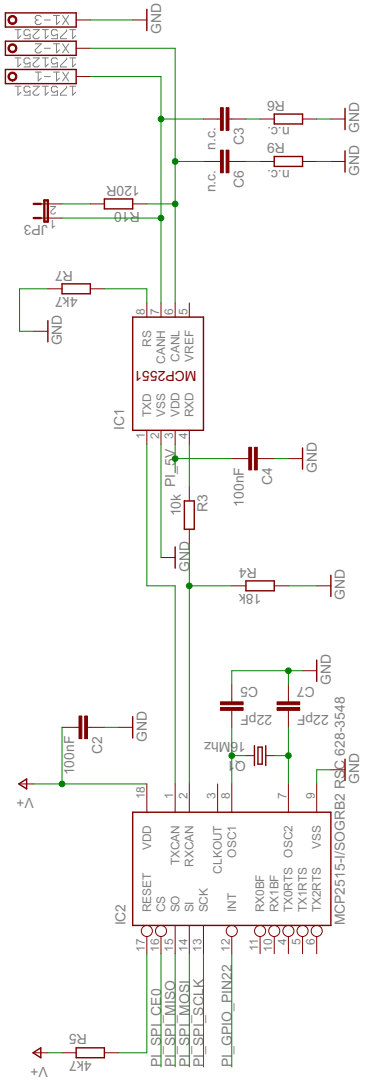


Figure 3.1: Electric diagram of CanBus block

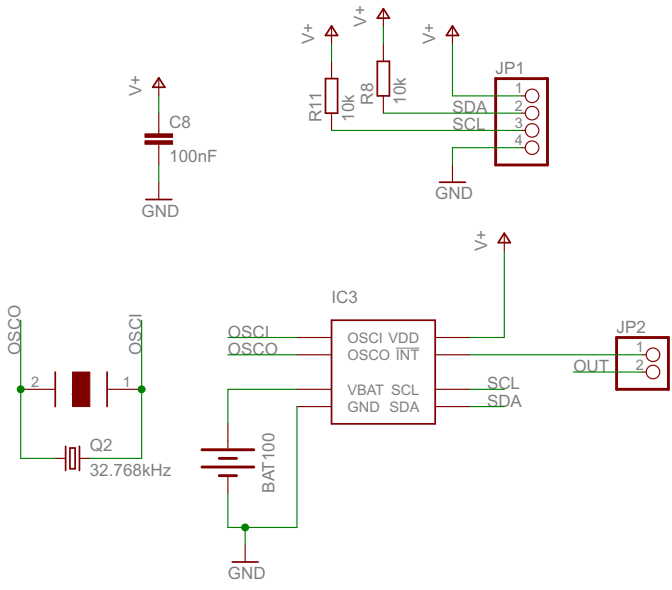


Figure 3.2: Electric diagram of RTC block

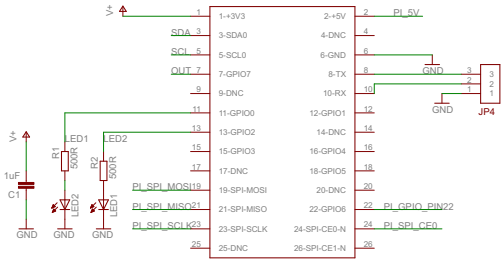


Figure 3.3: CanBerry Connector

Chapter 4

Software implementation

4.1 Real Time Clock

It is possible use the RTC IC with the terminal or with a compiled program. The PCF8523 is a device I2C, and then we must install **i2c-tool**

```
sudo aptitude install i2c-tool
```

and **libi2c-dev** before use it.

```
sudo aptitude install libi2c-dev
```

4.1.1 RTC with shell

The following code allow the management of the RTC with the i2c-tool directly from the shell.

```
### BEGIN INIT INFO
#RTC Set
### END INIT INFO

#! /bin/sh
# /record.sh

USER=pi
HOME=/pi

export USER HOME

case "$1" in
 set)
   echo "Setting time on PCF8523"
   #Set Control Reg 1
   i2cset -y 0 0x68 0x00 0x10 b
```

```
  #Set Control Reg 3
  i2cset -y 0 0x68 0x02 0x20 b
  echo "Year"
  i2cset -y 0 0x68 0x09 13 b
  echo "Month"
  i2cset -y 0 0x68 0x08 3 b
  echo "Day"
  i2cset -y 0 0x68 0x06 16 b
  echo "Hour"
  i2cset -y 0 0x68 0x05 22 b
  echo "Min"
  i2cset -y 0 0x68 0x04 50 b
  echo "Second"
  i2cset -y 0 0x68 0x03 25 b



  ;;

get)
  echo "Getting register value"
  echo "Year"
  i2cget -y 0 0x68 0x09
  echo "Month"
  i2cget -y 0 0x68 0x08
  echo "Day"
  i2cget -y 0 0x68 0x06
  echo "Hour"
  i2cget -y 0 0x68 0x05
  echo "Min"
  i2cget -y 0 0x68 0x04
  echo "Second"
  i2cget -y 0 0x68 0x03
  ;;

*)
  echo "Usage: record.sh {set|get}"
  exit 1
  ;;
esac

exit 0
```

4.1.2 RTC with a compiled C code

The following code allow the management of the RTC with the libi2c-dev from a C program.

```
/* pcf8523.c
   pcf8523_i2c_rtc.c - example of accessing a PCF8563 via
   the BSC0 (I2C) peripheral on a BCM2835 (Raspberry Pi)
 */
#include <linux/i2c-dev.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <errno.h>
#include <string.h>

#define ADDR  0x68
int fd;
int year;
int month;
int day;
int hour;
int min;
int sec;


void open_i2c(){

        fd = open( "/dev/i2c-0", O_RDWR );
        if( ioctl( fd, I2C_SLAVE, ADDR ) < 0 )
        {
                fprintf( stderr, "Failed to set slave address: %m\n" );
        }
}

void set_time(){

        if( i2c_smbus_write_byte_data( fd, 0x09, year ) < 0 )
        fprintf( stderr, "Failed to write YEAR to I2C device: %m\n" );
        if( i2c_smbus_write_byte_data( fd, 0x08, month ) < 0 )
        fprintf( stderr, "Failed to write MONTH to I2C device: %m\n" );
        if( i2c_smbus_write_byte_data( fd, 0x06, day ) < 0 )
        fprintf( stderr, "Failed to write DAY to I2C device: %m\n" );
```

```
        if( i2c_smbus_write_byte_data( fd, 0x05, hour ) < 0 )
        fprintf( stderr, "Failed to write HOUR to I2C device: %m\n" );
        if( i2c_smbus_write_byte_data( fd, 0x04, min ) < 0 )
        fprintf( stderr, "Failed to write MIN to I2C device: %m\n" );
        if( i2c_smbus_write_byte_data( fd, 0x03, sec ) < 0 )
        fprintf( stderr, "Failed to write SEC to I2C device: %m\n" );
        if( i2c_smbus_write_byte_data( fd, 0x02, 0x20 ) < 0 )
        fprintf( stderr, "Failed to write Reg3 to I2C device: %m\n" );
        if( i2c_smbus_write_byte_data( fd, 0x00, 0x10 ) < 0 )
        fprintf( stderr, "Failed to write Reg1 to I2C device: %m\n" );
        printf("RTC pcf8523 setted \n");

}

void get_time(void){
        year = i2c_smbus_read_byte_data(fd, 0x09);
        month = i2c_smbus_read_byte_data(fd, 0x08);
        day = i2c_smbus_read_byte_data(fd, 0x06);
        hour = i2c_smbus_read_byte_data(fd, 0x05);
        min = i2c_smbus_read_byte_data(fd, 0x04);
        sec = i2c_smbus_read_byte_data(fd, 0x03);
        printf("PCF8523 Time ");
        if (hour <10)
                printf("0%i:", hour);
        else
            printf("%i:", hour);

    if (min <10)
                printf("0%i:", min);
    else
            printf("%i:", min);

        if (sec <10)
                printf("0%x", sec);
        else
            printf("%x", sec);

        if (day <10)
                printf(" Date 0%i/", day);
        else
            printf(" Date %i/", day);

    if (month <10)
                printf("0%i/", month);
        else
```

```
            printf("%i/", month);

        printf("20%i\n", year);

}

void get_data(void){

        printf ("Input Day: ");
        while (!(scanf("%d", &day) != 0 && day <= 31 && day > 0))
        {
                while (getchar() != '\n');
                printf ("Try again: ");
        }
        printf ("Input Month: ");
        while (!(scanf("%d", &month) != 0 && month <= 12 && month > 0))
        {
                while (getchar() != '\n');
                printf ("Try again: ");
        }
        printf ("Input Year: ");
        while (!(scanf("%d", &year) != 0 && year <= 99 && year > 0))
        {
                while (getchar() != '\n');
                printf ("Try again: ");
        }

        printf ("Input Hour: ");
        while (!(scanf("%d", &hour) != 0 && hour <= 24 && hour >= 0))
        {
                while (getchar() != '\n');
                printf ("Try again: ");
        }

        printf ("Input Min: ");
        while (!(scanf("%d", &min) != 0 && min <= 60 && min >= 0))
        {
                while (getchar() != '\n');
                printf ("Try again: ");
        }

        printf ("Input Sec: ");
        while (!(scanf("%x", &sec) != 0 && sec <= 60 && sec >= 0))
        {
                while (getchar() != '\n');
```

```
                printf ("Try again: ");
        }
}


int main(int argc, char *argv[]) {
        if (argc < 2){
                open_i2c(); // Open i2c
                get_time();
                printf("Type pcf8523 -s for set RTC\n");
                }
        else if (argc == 2){
                if (!strcmp (argv[1],"-s")){
                        open_i2c(); // Open i2c
                        get_time();
                        printf("Setting RTC pcf8523 \n");
                        get_data();
                        set_time();
                        }
                else printf("Argument error\n");
                }
        else printf("Too many arguments\n");
    // Done!
}
```

The program pcf8523 allows simple operations on the RTC, for example, we can set and read the time and the date. We can see how the PCF8523 works:

```
pi@raspberrypi ~/compiler_test $ ./pcf8523
PCF8523 Time 17:24:29 Date 28/04/2013
Type pcf8523 -s for set RTC
pi@raspberrypi ~/compiler_test $ ./pcf8523 -s
PCF8523 Time 17:25:13 Date 28/04/2013
Setting RTC pcf8523
Input Day: 30
Input Month: 4
Input Year: 13
Input Hour: 18
Input Min: 10
Input Sec: 12
RTC pcf8523 setted
pi@raspberrypi ~/compiler_test $ ./pcf8523
PCF8523 Time 18:10:16 Date 30/04/2013
Type pcf8523 -s for set RTC
```

Chapter 5

Components list

In the table 5.1 we can see the Bill of Material for the board, all the components are available on-line. For simplicity, every component has a DigiKey order code (www.digikey.com).

Quantity	Value	Package	Parts	Digikey-cod	Unit Price $	
2	Yellow	1206	LED1, LED2	754-1144-1-ND	0,21	0,42
1	120 Ω	0603	R10	RMCF0603JT120RCT-ND	0,02	0,02
2	499 Ω	0603	R1, R2	RMCF0603FT499RCT-ND	0,04	0,08
2	4.7 kΩ	0603	R5, R7	RMCF0603JT4K70CT-ND	0,02	0,04
3	10 kΩ	0603	R3, R8, R11	P10KGCT-ND	0,10	0,30
1	18 kΩ	0603	R4	P18KGCT-ND	0,10	0,10
2	22pf	0603	C5, C7	445-1273-1-ND	0,10	0,20
3	100nf	0603	C2, C4, C8	445-1316-1-ND	0,10	0,30
1	1uf	0603	C1	445-1322-1-ND	0,10	0,10
1	32.768kHz		Q2	X801-ND	0,49	0,49
1	16MHz		Q1	535-10226-1-ND	0,41	0,41
1	PCF8523	SOIC8	IC3	568-5306-1-ND	1,36	1,36
1	MCP2515	SOIC18	IC2	MCP2515-I/SO-ND	1,98	1,98
1	MCP2551	SOIC8	IC1	MCP2551-I/SN-ND	1,12	1,12
1	RETAINER COIN 12MM		BAT100	BK-890-ND	0,32	0,32
1	TERMINAL-3-PCB	3 X 3.5mm	X1	277-5749-ND	1,51	1,51
1	Header 26 pos	2 X 13		SAM1086-13-ND	2,97	2,97
1				PCB	2,50	2,50
				Total		14,22

Table 5.1: CanBerry Pi V 1.0

Bibliography

[1] Microchip. *MCP2515 Datasheet.*

[2] Microchip. *MCP2551 Datasheet.*

[3] NXP. *PCF8523 Datasheet.*