

UNIVERSITÀ DEGLI STUDI DI PALERMO

EMBEDDED SYSTEMS

Door Alarm System on Raspberry Pi 4

Angelo Barbera 2023

Table Of Contents

1	Introduction	1
2	Hardware	1
2.1	Raspberry PI 4	1
2.2	FT232-AZ USB to TTL serial UART adapter	2
2.3	KY-003 Hall sensor	3
2.4	KY-012 Buzzer	3
2.5	LEDs	3
2.6	Push Button	4
2.7	Resistors	4
2.8	LCD 1602	4
2.9	PCF8574AT 8-bit I/O expander for I2C bus	7
2.9.1	The I2C bus	7
2.9.2	LCD connection	9
2.10	GPIO wiring diagram	10
3	Environment	11
3.1	pijForthos	11
3.2	Minicom, Picocom	11
4	Software	12
4.1	Environment setup	12
4.2	Code structure	12
4.2.1	se-ans.f	12
4.2.2	utils.f	13
4.2.3	i2c.f	15
4.2.4	lcd.f	16
4.2.5	peripherals.f	18
4.2.6	main.f	19
5	References	20

1 Introduction

The project is a Door Alarm System. The System is able to monitor the open or closed status of a door using a hall sensor to detect the presence of a magnet; if the latter is far from the sensor, it is emitted a sound alert with a buzzer. Using two LEDs and a LCD display 16x2 the status of the door is shown. In addition there is a button that can be used to turn off the alarm when the door is closed.

In the following sections is described in detail the hardware and the software used to implement the project.

2 Hardware

2.1 Raspberry PI 4

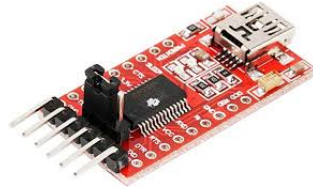


The chosen target for this project is the Raspberry Pi 4 Model B, a single board computer developed by the Raspberry Pi Foundation and released in 2019. The tech specs include:

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5Ghz
- 1GB, 2GB, 4GB, or 8GB LPDDR4-3200 SDRAM (depending on model)
- 2.4 GHz and 5.0 Ghz 802.11ac wireless
- Gigabit Ethernet
- Bluetooth 5.0, BLE
- 2 USB 3.0 ports, 2 USB 2.0 ports
- Raspberry Pi standard 40 pin GPIO header

- 2 micro-HDMI ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H.264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.1, Vulkan 1.0
- Micro-SD card slot for loading operating system and data storage
- 5V DC via USB-C connector (minimum 3A)
- 5V DC via GPIO header (minimum 3A)
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)
- Operating temperature 0 - 50 °C ambient

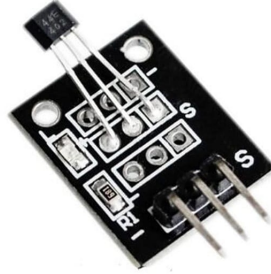
2.2 FT232-AZ USB to TTL serial UART adapter



The FT232-AZ USB to TTL serial UART adapter is used to connect the PC used during the development of the project to the target in order to send and receive data between the PC and the Raspberry Pi 4. The PC is connected through a USB port, the target is connected through GPIO pins according to the following table.

GPIO	Function	UART adapter
14 (Tx)	Output	Rx
15 (Rx)	Input	Tx
Ground	Ground	Ground

2.3 KY-003 Hall sensor



The KY-003 hall sensor allows to detect a magnetic field. When the magnetic field at the Hall sensor exceeds the operate point threshold (BOP) the output of the device switches low. When the magnetic field is reduced to below the release point threshold (BRP) the device output switches high. BOP and BRP may vary respectively from 1 mT to 33 mT and from 5 mT to 35 mT at operating temperature $T = 25^{\circ}\text{C}$ depending on the sensor model. This sensor is used to trigger the alarm when the magnet is far from the sensor.

2.4 KY-012 Buzzer



The KY-012 Buzzer is an active piezoelectric buzzer, it generates a sound of approximately 2.5kHz when input signal (S) is high. The Buzzer is activated when the Hall sensor does not detect the magnet.

2.5 LEDs



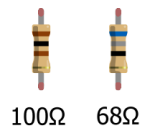
The LEDs are used to show the alarm status. When the Hall sensor does not detects the magnet the green LED turns off and the red LED turns on. When the Hall sensor detect the magnet and the push button is pressed, the green LED turns on and the red LED turns off.

2.6 Push Button



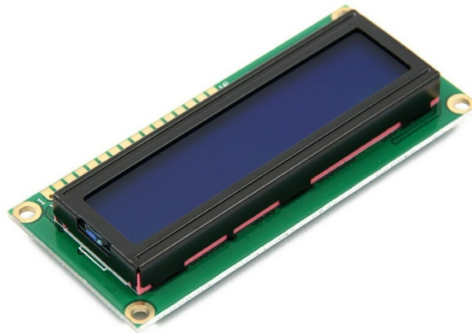
The push button can be used to turn off the alarm when the hall sensor detects the magnet.

2.7 Resistors

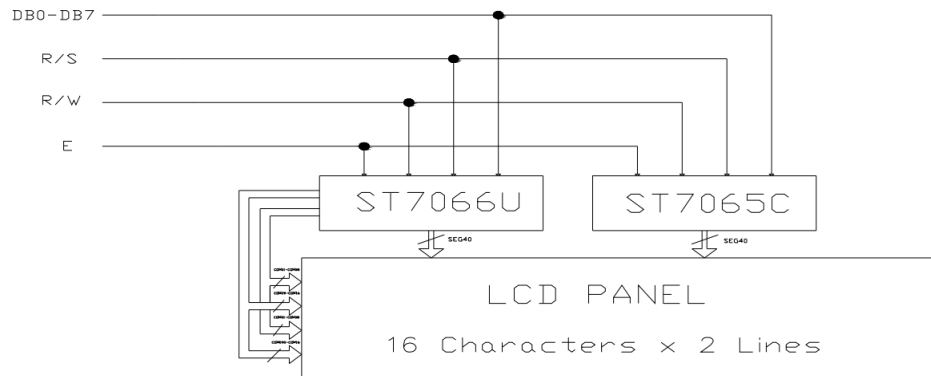


The resistors are connected in series to the LEDs to limit the current flowing through the LED and to ensure that the supplied voltage does not exceeds the maximum voltage of the LED. The 100 Ω resistor is connected to the red LED, the 68 Ω resistor is connected to the green LED.

2.8 LCD 1602



The LCD 1602 is an industrial character LCD that can display 16x02 or 32 characters at the same time. The LCD 1602 is controlled through a parallel interface with 8-bit / 4-bit data bus and 3 control signals. The interface signals reach the two controller chips that drive the LCD panel as shown in the following block diagram.



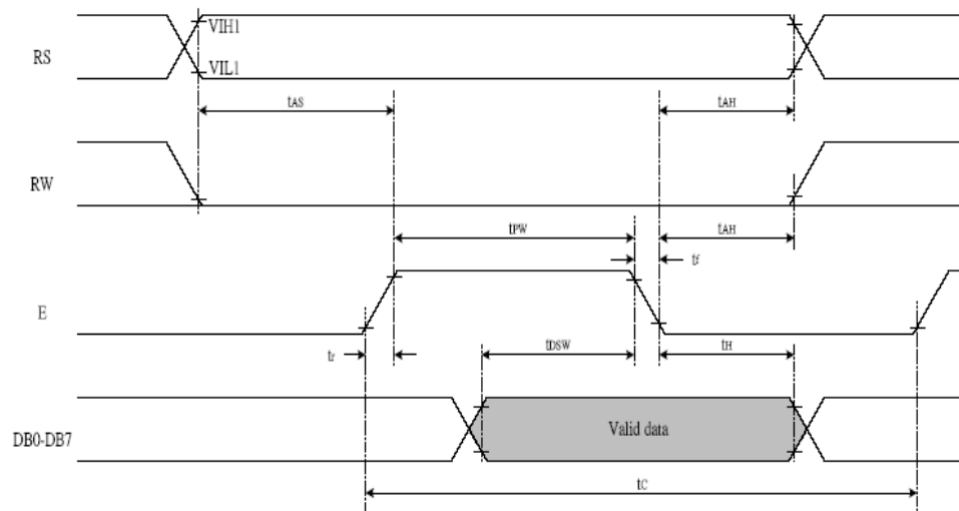
The following table describes the pin assignment

No.	Symbol	Level	Function
1	Vss	--	0V
2	Vdd	--	+3.3V
3	V0	--	for LCD
4	RS	H/L	Register Select: H:Data Input L:Instruction Input
5	R/W	H/L	H--Read L--Write
6	E	H,H-L	Enable Signal
7	DB0	H/L	Data bus used in 8 bit transfer
8	DB1	H/L	
9	DB2	H/L	
10	DB3	H/L	
11	DB4	H/L	Data bus for both 4 and 8 bit transfer
12	DB5	H/L	
13	DB6	H/L	
14	DB7	H/L	
15	BLA	--	BLACKLIGHT +3.3V
16	BLK	--	BLACKLIGHT 0V-

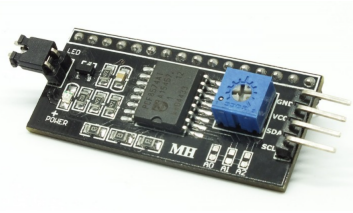
The LCD module is controlled through instructions to set display format, data length, scrolling modality, internal RAM address, to perform data transfer from/to internal RAM and to access status flag. The following table describes the instructions.

Instruction	Instruction Code										Description	Description Time (270KHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM, and set DDRAM address to "00H" from AC	1.52 ms
Return Home	0	0	0	0	0	0	0	0	1	x	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.52 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 us
Display ON/OFF	0	0	0	0	0	0	1	D	C	B	D=1:entire display on C=1:cursor on B=1:cursor position on	37 us
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	x	x	Set cursor moving and display shift control bit, and the direction, without changing DDRAM data.	37 us
Function Set	0	0	0	0	1	DL	N	F	x	x	DL:interface data is 8/4 bits N:number of line is 2/1 F:font size is 5x11/5x8	37 us
Set CGRAM address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter	37 us
Set DDRAM address	0	0	1	AD6	AD5	AD4	AD3	AD2	AD1	AD0	Set DDRAM address in address counter	37 us
Read Busy flag and address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0 us
Write data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM)	37 us
Read data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM)	37 us

In order to write to the controller chips we need to set properly the control bits: the R/W bit must be set to 0 for writing, the RS bit must be set to 1 for data input and to 0 for instruction input, the E bit must be set to 1 before the start of the data transmission and must be set to 0 before the end of the data transmission as shown in the following figure



2.9 PCF8574AT 8-bit I/O expander for I2C bus



The PCF8574AT provides general-purpose remote I/O expansion via the two-wire bidirectional I^2C bus. It is used to connect the Raspberry Pi to the LCD 1602 using the I^2C bus instead of the parallel interface.

2.9.1 The I2C bus

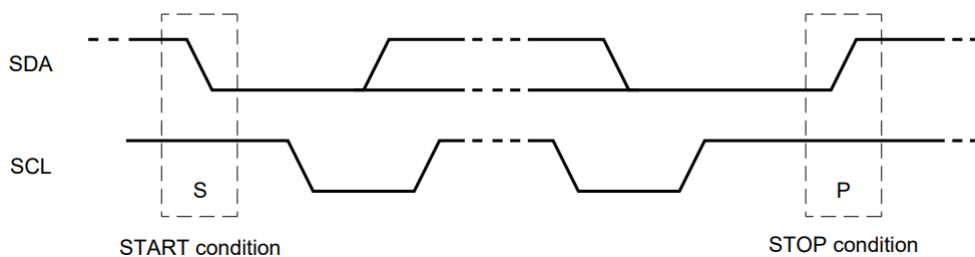
The I^2C bus is a synchronous, multi-master, multi-slave, packet switched, single-ended, serial computer bus invented by Philips Semiconductor. It is used to connect lower speed peripheral integrated circuits to processors and microcontrollers in short distance. Each device connected to the bus is software addressable by a unique address to differentiate between other devices that are on the same bus. Two wires carry data (SDA) and clock signal (SCL).

Open drain connection allows for bidirectional communication: to transmit low the logic activate the pull-down FET (so the line is pulled low), to transmit high the logic release the bus turning off the pull-down FET (so the pull-up resistor pulls up the line).

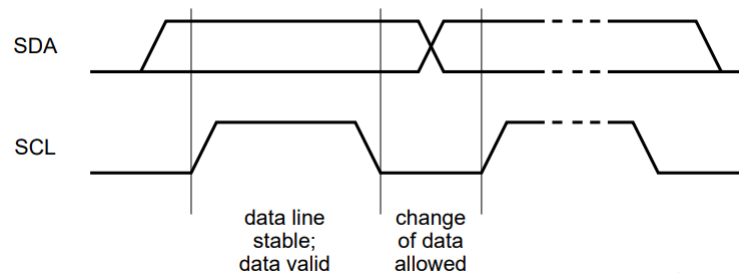
The procedure for a master to send data to a slave is the following:

- Master-transmitter sends a START condition and addresses the slave-receiver
- Master-transmitter sends data to slave-receiver
- Master-transmitter terminates the transfer with a STOP condition

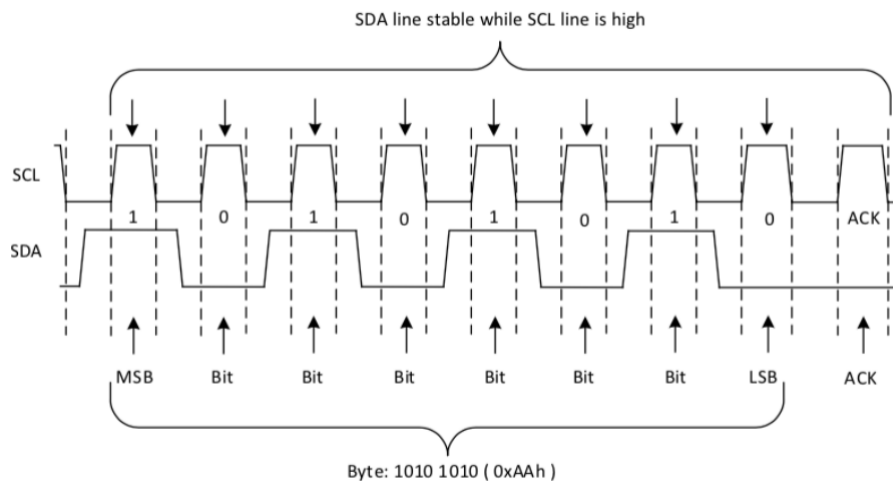
The START condition is defined by a high-to-low transition on the SDA line while the SCL is high. The STOP condition is defined by a low-to-high transition on the SDA line while the SCL is high.



One data bit is transferred during each clock pulse of the SCL, the data on the SDA line must remain stable during the high period of the clock pulse as changes in the data line at this time will be interpreted as control signals (START or STOP). Any number of data bytes can be transferred between the START and STOP conditions. Data is transferred Most Significant Bit first.

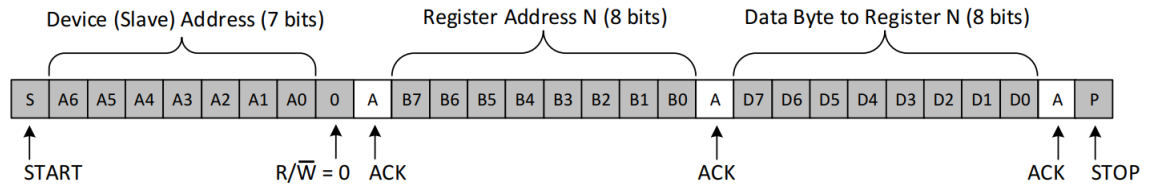


Each byte of data is followed by one ACK (acknowledge) bit from the receiver to signal that the byte was successfully received. Before the receiver can send an ACK the transmitter must release the SDA line. To send an ACK bit the receiver pulls down the SDA after receiving the last bit so the line is stable low during the high phase of the ACK clock period. When SDA line remains high after receiving the last bit, this is interpreted as a NACK (not acknowledge)



The procedure to write on the bus is the following:

- The master sends a START condition with the slave's address followed by the R/W bit set to 0
- The slave sends the acknowledge bit
- The master sends the register address of the register it wishes to write to
- The slave acknowledges the register address
- The master starts sending the register data to the slave
- The master terminates the transmission with a STOP condition



2.9.2 LCD connection

The following table describes the connections between the LCD 1602 and the PCF8574AT

PCF8574AT	LCD 1602
P0	RS
P1	RW
P2	E
P3	Backlight
P4	D4
P5	D5
P6	D6
P7	D7

In order to set the LCD 1602 to 4 bit mode it is necessary to send the following command

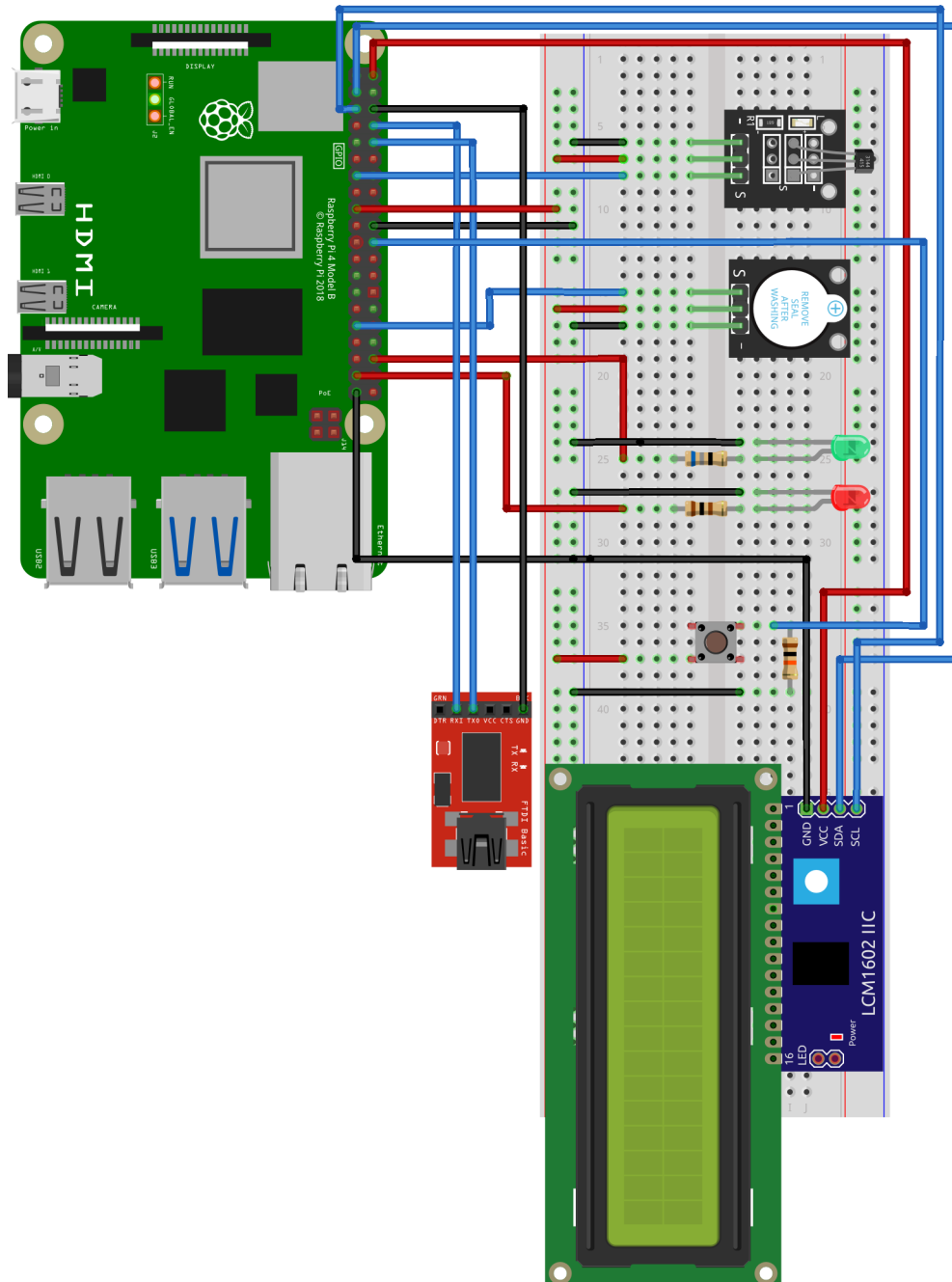
	D7	D6	D5	D4	Backlight	E	R/W	RS
2C	0	0	1	0	1	1	0	0
28	0	0	1	0	1	0	0	0

Now is possible to send a command signal or a data signal according to the following tables

	D7	D6	D5	D4	Backlight	E	R/W	RS
MSB_CMD C	B7	B6	B5	B4	1	1	0	0
MSB_CMD 8	B7	B6	B5	B4	1	0	0	0
LSB_CMD C	B3	B2	B1	B0	1	1	0	0
LSB_CMD 8	B3	B2	B1	B0	1	0	0	0

	D7	D6	D5	D4	Backlight	E	R/W	RS
MSB_DATA D	B7	B6	B5	B4	1	1	0	1
MSB_DATA 9	B7	B6	B5	B4	1	0	0	1
LSB_DATA D	B3	B2	B1	B0	1	1	0	1
LSB_DATA 9	B3	B2	B1	B0	1	0	0	1

2.10 GPIO wiring diagram



The following table is the GPIO wiring diagram.

GPIO	Function	Connection
2	SDA	SDA (I/O expander)
3	SCL	SCL (I/O expander)
6	Output	S (Buzzer)
16	Output	Anode (Green LED)
25	Input	Button
26	Output	Anode (Red LED)
27	Input	S (Hall sensor)
5V	Power	VCC (I/O expander)
3V3	Power	Breadboard
Ground	Ground	Breadboard

3 Environment

3.1 pijForthos

Forth is a procedural, stack-oriented programming language and interactive environment designed by Charles H. "Chuck" Moore. Forth combines a compiler with an integrated command shell, where the user interacts via subroutine called words

pijForthos is a bare-metal operating system for Raspberry Pi based on Jonesforth-ARM. Jonesforth-ARM is an ARM port of x86 JonesForth by Richard W.M. Jones, a Forth interpreter developed for ARM. If you have pijFORTHos on a Micro SD card in the Raspberry Pi, you can connect it to development machine using the USB to TTL serial UART adapter. Using the programs described below you can have access to the Forth console.

3.2 Minicom, Picocom

Minicom is a text-based terminal emulator program for Unix-like operating systems, it is used to set up a remote serial console. Picocom is similar to Minicom and it was designed to serve as a simple, manual, modem configuration, testing and debugging tool.

ASCII-XFR transfers files in ASCII mode and it is used to send the source file to the Raspberry Pi allowing to set a delay between each character and line sent. It is useful because, since the UART connection is asynchronous, if the receiver is busy compiling or executing Forth words and the transmitter is sending data, it is possible to lose them.

The following command is used to set up the communication with the Raspberry Pi :

```
picocom --b 115200 /dev/ttyUSB0 --imap delbs -s "ascii-xfr -sv -l100 -c10"
```

- `--b 115200` defines the baud rate
- `/dev/ttyUSB0` specifies the UART port
- `--imap delbs` maps `del` to backspace
- `-s "ascii-xfr -sv -l100 -c10"` specifies the external program that will be used to transmitting files
- `-sv` specifies verbose mode
- `-l100` specifies the delay (100 milliseconds) after transmitting each line
- `-c10` specifies the delay (10 milliseconds) after transmitting each character

4 Software

4.1 Environment setup

To setup the environment it is necessary to format the micro SD card using FAT32 as file system. Then the following files must be copied in the micro SD card:

- `bootcode.bin` contains the bootloader code
- `config.txt` contains the configuration parameters
- `start4.elf` contains the second-stage bootloader
- `fixup.dat` contains Video Core code and initialization data
- `bcm2711-rpi-4-b.dtb` describes the hardware present in the Raspberry Pi
- `kernel7.img` contains the piFORTHos code

The `config.txt` file must be edited specifying the following settings:

- `dtparam=i2c_arm=on` enables I^2C
- `enable_uart=1` enables UART

4.2 Code structure

The code is divided into the following files

- `se-ans.f` contains some definition for ANS compliance
- `utils.f` contains utility words
- `i2c.f` contains the words used to set up i2c
- `lcd.f` contains the words used to control the lcd
- `peripherals.f` contains the words used to manage the peripherals
- `main.f` contains the words used to run the system

The previous files must be sent to the Raspberry Pi using `picocom` in the specified order.

4.2.1 `se-ans.f`

```
\ Sistemi Embedded 18/19
\ Daniele Peri - Universita' degli Studi di Palermo
\
\ Some definitions for ANS compliance
\
\ v. 20181215

: JF-HERE    HERE ;
: JF-CREATE  CREATE ;
: JF-FIND    FIND ;
: JF-WORD    WORD ;
```

```

: HERE    JF-HERE @ ;
: ALLOT   HERE + JF-HERE ! ;

: [']     ' LIT , ; IMMEDIATE
: '       JF-WORD JF-FIND >CFA ;

: CELL+   4 + ;

: ALIGNED 3 + 3 INVERT AND ;
: ALIGN JF-HERE @ ALIGNED JF-HERE ! ;

: CREATE  JF-WORD JF-CREATE DOCREATE , ;
: (DODOES-INT) ALIGN JF-HERE @ LATEST @ >CFA ! DODOES> ['] LIT ,
  LATEST @ >DFA , ;
: (DODOES-COMP) (DODOES-INT) ['] LIT , , ['] FIP! , ;
: DOES>COMP    ['] LIT , HERE 3 CELLS + , ['] (DODOES-COMP) , ['] EXIT , ;
: DOES>INT      (DODOES-INT) LATEST @ HIDDEN ] ;
: DOES>         STATE @ 0= IF DOES>INT ELSE DOES>COMP THEN ; IMMEDIATE

```

4.2.2 utils.f

HEX

```

FE200000 CONSTANT PERI_BASE    \ Base address of peripherals
1 CONSTANT OUTPUT
0 CONSTANT INPUT

\ Creates a busy loop
: DELAY ( delay_value -- )
  BEGIN 1 - DUP 0 = UNTIL DROP
;

\ Returns the GPFSEL address of the specified fsel number
: FSEL>ADDRESS ( fsel_number -- gpfsel_register_address )
  A / 4 * PERI_BASE +
;

\ Returns the GPFSEL value of the specified fsel number
: FSEL>VALUE ( fsel_number -- gpfsel_register_value )
  FSEL>ADDRESS @
;

\ Creates bit mask to safely modify GPIO registers value
: MASK ( fsel_number value -- bit_mask )
  SWAP A MOD 3 * LSHIFT
;

\ Returns the new_value setting to 0 the specified bit in mask

```

```

: BIC ( value mask - new_value)
  INVERT AND
;

\ Creates bit mask to use bic (bit clear)
: BIC_MASK ( fsel_number -- bic_mask )
  7 MASK
;

\ Gets only the first 4 MSB from a byte
: MSB ( byte -- MSB)
  4 RSHIFT
;

\ Gets only the first 4 LSB from a byte
: LSB ( byte -- LSB)
  OF AND
;

\ Copies the top of the return stack without affecting it
: R@ ( -- TORS )
  R> R> TUCK >R >R
;

\ Sets GPFSEL register to the specified value
: SET_GPFSEL ( fsel_number value -- )
  OVER \ ( fsel_number value fsel_number )
  BIC_MASK \ ( fsel_number value bic_mask )
  ROT ROT SWAP DUP ROT \ ( bic_mask fsel_number fsel_number value )
  MASK \ ( bic_mask fsel_number bit_mask )
  SWAP \ ( bic_mask bit_mask fsel_number )
  DUP \ ( bic_mask bit_mask fsel_number fsel_number )
  FSEL>VALUE \ ( bic_mask bit_mask fsel_number original_value )
  >R ROT R> SWAP \ ( bit_mask fsel_number original_value bic_mask )
  BIC \ ( bit_mask fsel_number bic_value )
  ROT \ ( fsel_number bic_value bit_mask )
  OR \ ( fsel_number new_value )
  SWAP \ ( new_value fsel_number )
  FSEL>ADDRESS \ ( new_value fsel_address )
  !
;

\ Returns the mask used to set or clear GPIO register
: SET_CLR_MASK ( gpio_pin -- mask )
  DUP 20 < IF 1 SWAP LSHIFT \ gpio_pin < 32
  ELSE DUP 1F > IF 1 SWAP 20 MOD LSHIFT \ gpio_pin > 31
  THEN THEN
;

```



```

\ Sets GPIO register
: ON ( gpio_pin -- )
  DUP SET_CLR_MASK SWAP
  DUP 20 < IF DROP 1C PERI_BASE + !           \ gpio_pin < 32
  ELSE DUP 1F > IF DROP 20 PERI_BASE + !       \ gpio_pin > 31
  THEN THEN
;

\ Clears GPIO register
: OFF ( gpio_pin -- )
  DUP SET_CLR_MASK SWAP
  DUP 20 < IF DROP 28 PERI_BASE + !           \ gpio_pin < 32
  ELSE DUP 1F > IF DROP 2C PERI_BASE + !       \ gpio_pin > 31
  THEN THEN
;

\ Returns the value of the specified pin
: READ ( gpio_pin -- value )
  DUP 20 < IF 34 PERI_BASE + @ SWAP RSHIFT 1 AND
  ELSE DUP 1F > IF 38 PERI_BASE + @ SWAP RSHIFT 1 AND
  THEN THEN
;

```

4.2.3 i2c.f

HEX

```

FE804000 CONSTANT BSC1           \ Base address of BSC1 register
BSC1 CONSTANT CONTROL           \ Control register address
4 BSC1 + CONSTANT STATUS        \ Status register address
8 BSC1 + CONSTANT DATA_LENGTH  \ Data Length register address
0C BSC1 + CONSTANT SLAVE_ADDRESS \ Slave Address register address
10 BSC1 + CONSTANT DATA_FIFO   \ Data FIFO register address

\ Sets GPIO2 and GPIO3 to ALTO
: CONFIG_I2C_GPIO ( -- )
  2 4 SET_GPFSEL      \ Sets FSEL2 to 100
  3 4 SET_GPFSEL      \ Sets FSEL3 to 100
;

\ Resets the Status Register setting:
\ bit 1 (Transfer Done) to 1,
\ bit 8 (Ack Error) to 1,
\ bit 9 (Clock Stretch Timeout) to 1
: RESET_STATUS ( -- )
  STATUS @ 302 OR STATUS !
;

\ Resets FIFO setting bit 4 (FIFO Clear) to 1

```

```

: RESET_FIFO ( -- )
    CONTROL @ 10 OR CONTROL !
;

\ Sets the number of bytes of data to transmit or receive to 1
: SET_DATA_LENGTH ( -- )
    DATA_LENGTH @ 1 OR DATA_LENGTH !
;

\ Sets the slave address
: SET_SLAVE ( -- )
    SLAVE_ADDRESS @ 7F BIC 27 OR SLAVE_ADDRESS !
;

\ Stores data in Data FIFO register
: STORE_FIFO ( data -- )
    DATA_FIFO @ FF BIC OR DATA_FIFO !
;

\ Starts a new transfer setting
\ bit 0 (Read Transfer) to 0
\ bit 7 (Start Transfer) to 1
\ bit 15 (I2C enable) to 1
: START_TRANSFER ( -- )
    CONTROL @ 1 BIC 8080 OR CONTROL !
;

\ Sends 8 bit using i2c
: SEND ( data -- )
    RESET_STATUS
    RESET_FIFO
    SET_DATA_LENGTH
    SET_SLAVE
    STORE_FIFO
    START_TRANSFER
;

```

4.2.4 lcd.f

```

    HEX

02 CONSTANT FUNCTION_SET      \ Sets 4 bit mode
01 CONSTANT CLEAR             \ Clear lcd

: DOOR ( -- reversed_ascii_code length)
    20 52 4F 4F 44 5
;

```

```

: CLOSED ( -- reversed_ascii_code length)
  20 44 45 53 4F 4C 43 7
;

: OPEN ( -- reversed_ascii_code length)
  20 4E 45 50 4F 5
;

\ Sends a nibble to lcd
: SEND_NIBBLE ( LSB settings -- )
  SWAP 4 LSHIFT OR
  SEND
  1000 DELAY
;

\ Sends command to lcd
\ It is necessary to send: D7, D6, D5, D4,
\ Backlight, Enable, Read (1)/Write (0), Register Select (1-data 0-instruction)
\ C = 1100, 8 = 1000
\ 4 MSB|C
\ 4 MSB|8
\ 4 LSB|C
\ 4 LBS|8
: SEND_COMMAND ( command -- )
  DUP MSB SWAP LSB SWAP DUP          \ ( LSB MSB MSB )
  OC SEND_NIBBLE
  08 SEND_NIBBLE
  DUP
  OC SEND_NIBBLE
  08 SEND_NIBBLE
;

\ Sends data to lcd
\ It is necessary to send: D7, D6, D5, D4,
\ Backlight, Enable, Read (1)/Write (0), Register Select (1-data 0-instruction)
\ D = 1101, 9 = 1001
\ 4 MSB|D
\ 4 MSB|9
\ 4 LSB|D
\ 4 LBS|9
: SEND_DATA ( data -- )
  DUP MSB SWAP LSB SWAP DUP          \ ( LSB MSB MSB )
  OD SEND_NIBBLE
  09 SEND_NIBBLE
  DUP
  OD SEND_NIBBLE
  09 SEND_NIBBLE
;

```

```

\ Initializes lcd
: INITIALIZE_LCD ( -- )
    CONFIG_I2C_GPIO
    FUNCTION_SET SEND_COMMAND
;

\ Send a word to lcd
: SEND_WORD ( reversed_ascii_code length --)
    >R
    BEGIN R> 1 - >R SEND_DATA R@ 0 = UNTIL
    R> DROP
;

```

4.2.5 peripherals.f

HEX

1A CONSTANT RED_LED	\ The red LED is connected to GPIO 26
10 CONSTANT GREEN_LED	\ The green LED is connected to GPIO 16
6 CONSTANT BUZZER	\ The Buzzer is connected to GPIO 6
19 CONSTANT BUTTON	\ The Button is connected to GPIO 25
1B CONSTANT HALL_SENSOR	\ The Hall sensor is connected to GPIO 27

```

\ Initializes peripherals
: INITIALIZE_PERIPHERALS ( -- )
    RED_LED OUTPUT SET_GPFSEL      \ Sets gpfsel 26 to output
    GREEN_LED OUTPUT SET_GPFSEL    \ Sets gpfsel 16 to output
    BUZZER OUTPUT SET_GPFSEL       \ Sets gpfsel 6 to output
    BUTTON INPUT SET_GPFSEL        \ Sets gpfsel 25 to input
    HALL_SENSOR INPUT SET_GPFSEL   \ Sets gpfsel 27 to input
;

\ Returns the status of the button (1 or 0)
: BUTTON_STATUS ( -- button_status)
    BUTTON READ
;

\ Returns the status of the hall sensor (1 or 0)
: HALL_SENSOR_STATUS ( -- hall_sensor_status)
    HALL_SENSOR READ
;

\ Returns the status of the buzzer (1 or 0)
: BUZZER_STATUS ( -- buzzer_status )
    BUZZER READ
;

```

4.2.6 main.f

HEX

\ Executes the program

: RUN (--)

INITIALIZE_LCD

CLEAR SEND_COMMAND

INITIALIZE_PERIPHERALS

GREEN_LED ON

BEGIN

HALL_SENSOR_STATUS 1 = BUZZER_STATUS 0 = AND IF

GREEN_LED OFF

RED_LED ON

BUZZER ON

CLEAR SEND_COMMAND

DOOR OPEN SEND_WORD SEND_WORD

THEN

BUTTON_STATUS 1 = HALL_SENSOR_STATUS 0 = AND IF

RED_LED OFF

GREEN_LED ON

BUZZER OFF

CLEAR SEND_COMMAND

DOOR CLOSED SEND_WORD SEND_WORD

THEN

0 UNTIL

;

5 References

- raspberrypi.com/products/raspberry-pi-4-model-b/specifications
- cdn.shopify.com/s/files/1/1509/1638/files/FT232-AZ_Adapter_Datenblatt_AZ-Delivery-Vertriebs_GmbH.pdf
- cdn.shopify.com/s/files/1/1509/1638/files/Hall_Sensor_Modul_Digital_Datenblatt.pdf
- cdn.shopify.com/s/files/1/1509/1638/files/KY-012_Buzzer_Modul_Aktiv_Datenblatt-AZ-Delivery-Vertriebs_GmbH.pdf
- wiki.52pi.com/index.php?title=Z-0234
- msb-co.ir/wp-content/uploads/2021/10/eone-1602a1.pdf
- nxp.com/docs/en/data-sheet/PCF8574_PCF8574A.pdf
- ti.com/lit/an/slva704/slva704.pdf
- github.com/organix/pijFORTHos
- linux.die.net/man/1/minicom
- linux.die.net/man/8/picocom
- linux.die.net/man/1/ascii-xfr
- manpages.ubuntu.com/manpages/trusty/man8/picocom.8.html
- unix.com/man-page/redhat/1/ascii-xfr/