

Università degli Studi di Palermo
Computer Engineering

Embedded Systems

IR Receiver Project

By Davide Sferrazza
January 29, 2023

Contents

1	Introduction	1
2	Hardware	1
2.1	Raspberry Pi 4 Model B	1
2.2	FTDI Adapter FT232RL	2
2.3	KY-022 Infrared receiver module	3
2.4	Samsung AA59-00584A	4
2.5	LCD1602	5
2.6	IIC PCF8574AT Interface	7
2.6.1	Writing mechanism	8
2.6.2	PCF8574AT to LCD connection	10
2.7	Schematics	11
3	Environment	12
3.1	pijFORTHos	12
3.2	Minicom and Picocom	12
3.3	Script	13

1 Introduction

The proposed project consists of using an IR receiver to capture commands sent by a remote controller and displaying them on an LCD display, along with the name of the pressed buttons.

This document is organized as follows:

- Firstly, I will discuss the hardware used for the implementation, by describing the physical architecture where the software will run and the external necessary hardware components that will interact with it;
- Then, I will explain the environment used for project development and how to obtain the parameters needed for the project to function properly;
- Lastly, I will illustrate how the code was designed and organized.

On the last pages you can find the entire code. It can also be found online using GitHub (insert link).

2 Hardware

2.1 Raspberry Pi 4 Model B

The target of this project is a general-purpose Single-Board Computer (SBC): the Raspberry Pi 4 Model B.

It is a member of a series of products, which are developed in the United Kingdom by the Raspberry Pi Foundation in association with Broadcom.

It was released in June 2019[1] and replaced the well-known **Raspberry Pi 3 Model B** and **Raspberry Pi 3 Model B+**.

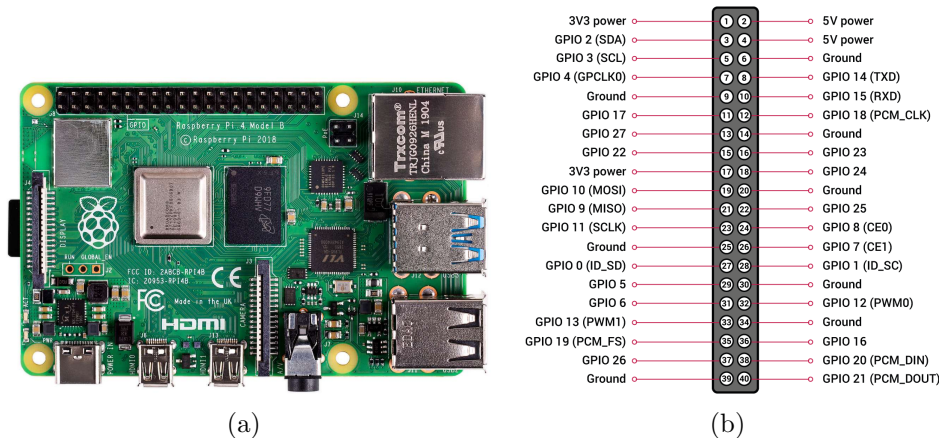


Figure 1: Raspberry Pi 4 Model B

It is built with the following specifications[2]:

- 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 IEEE 802.11ac wireless, Bluetooth 5.0, BLE;
- Gigabit Ethernet;
- 2 USB 3.0 ports; 2 USB 2.0 ports;
- Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards);
- $2 \times$ 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);
- H264 (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 degrees C ambient.

The model I used comes with 4GB of RAM.

2.2 FTDI Adapter FT232RL

Since modern computers do not expose serial ports to program the SBC, a UART (Universal Asynchronous Receiver-Transmitter) serial adapter is required.

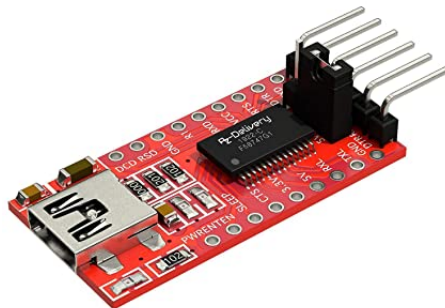


Figure 2: FTDI Adapter FT232RL from Mini-USB to TTL

I adopted the Mini-USB to TTL serial adapter provided by AZ-Delivery[4]. To avoid voltage problems power is supplied from the terminal used to develop the project and is shared between the Pi 4 and the FT232RL. The connection is made in the following way:

- FT232RL RX pin is connected to GPIO 14 (TXD);
- FT232RL TX pin is connected to GPIO 15 (RXD);
- FT232RL GND pin is connected to the GND of the Pi 4.

2.3 KY-022 Infrared receiver module

IR detectors[5] are little microchips with a photocell that are tuned to listen to infrared light. They are almost always used for remote control detection - every TV and DVD player has one of these in the front to listen for the IR signal from the clicker. Inside the remote control is a matching IR LED, which emits IR pulses to tell the TV to turn on, off or change channels. IR light is not visible to the human eye, which means it takes a little more work to test a setup.



Figure 3: KY-022

The module is able to detect frequencies ranging from about 35 KHz to 41 KHz, but the peak frequency detection is at 38 KHz.

The IR receiver comes with 3 pins: the digital signal output pin (S) used to read the value of infrared light, the power pin (+) and the ground pin (-).

When it detects a 38KHz IR signal, the output is low. When it detects nothing, the output is high.

It requires a supply voltage in [2.7, 5.5]V, so it is powered by the Pi 4 using one of the 3V3 pins. Its GND pin is connected to the GND of the Pi 4.

Its output pin is connected to GPIO 25.

2.4 Samsung AA59-00584A

The TV remote controller I used is produced by Samsung.



Figure 4: Samsung AA59-00584A

After several timing trials, I observed that the implemented protocol conforms to the timings found online on the page cited in the references [9].

For convenience, I report the picture of the timings here:

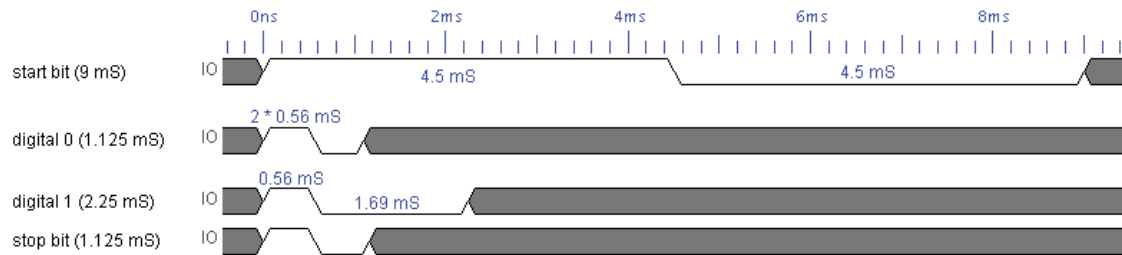


Figure 5: Samsung protocol

2.5 LCD1602

The LCD1602[6], [7] is an industrial character LCD that can display 16×2 or 32 characters at the same time, with a display font of 5×8 dots. The principle of the LCD1602 liquid crystal display is to use the physical characteristics of the liquid crystal to control the display area by voltage, that is, the graphic can be displayed.

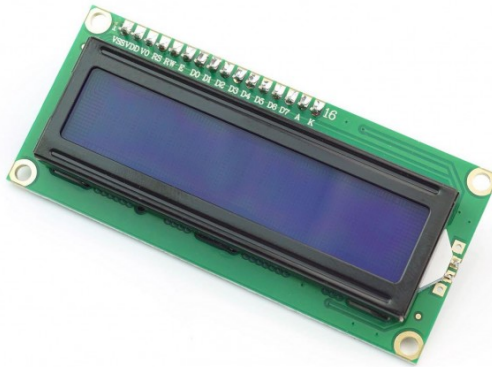


Figure 6: LCD 1602

It is controlled through a parallel interface with:

- 8-bit/4-bit data bus;
- 3 control signals.

The interface signals reach the two controller chips that drive the LCD panel:

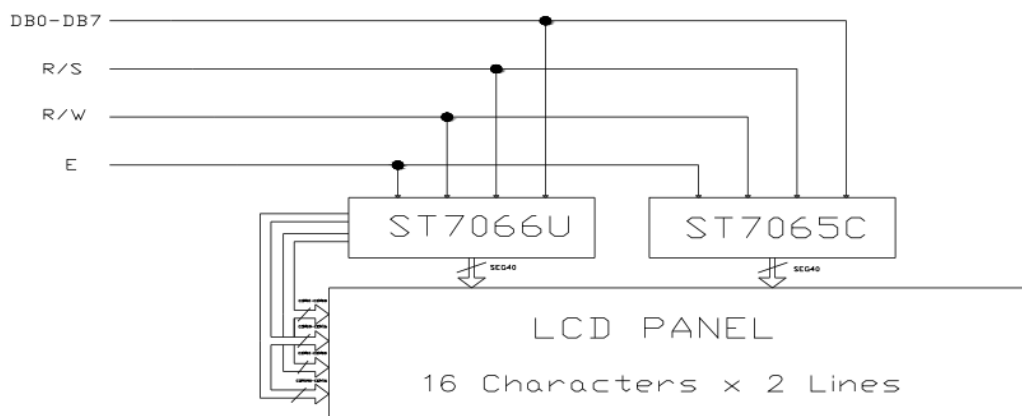


Figure 7: Block Diagram

Pin assignments are summarized in this table:

No.	Symbol	Level	Function	
1	Vss	--	0V	Power Supply
2	Vdd	--	+5V	
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 +5V	
16	BLK	--	BLACKLIGHT 0V-	

Figure 8: LCD pin assignments

To properly read or write data, some timing constraints must be observed. Independently of whether we are reading or writing, the Enable signal must start its falling edge while DB0-DB7 are stable. If it is not, then erroneous data are sampled. The read or write mode is chosen only by the RW signal, so only one case of these modes can be represented.

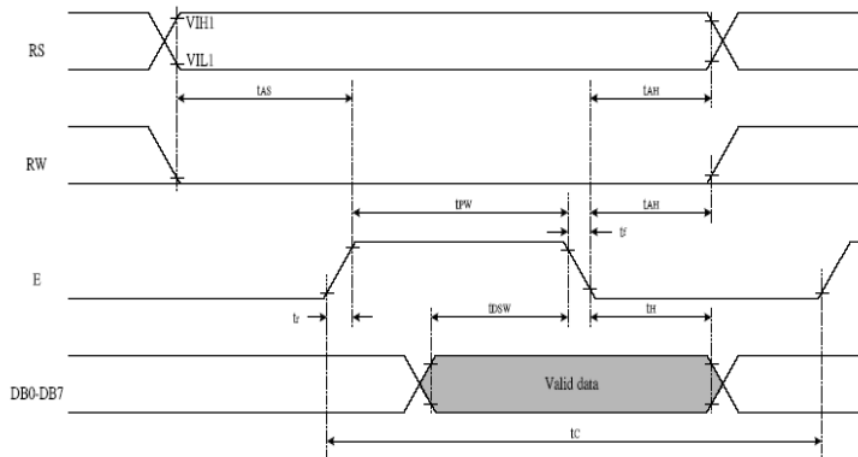


Figure 9: Writing timing characteristics

There are four categories of instructions that:

- set display format, data length, cursor move direction, display shift etc.;
- set internal RAM addresses;
- perform data transfer from/to internal RAM;

- others.

A detailed description of how they can be realized is as follows:

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	AC6	AC5	AC4	AC3	AC2	AC1	AC0	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

Figure 10: Instruction Table

2.6 IIC PCF8574AT Interface

To ease the communication to the LCD display a serial I²C module[7], [8] has been used. The interface connects its serial input and parallel output to the LCD, so only 4 lines can be used to do the job.

The I²C bus was invented by Philips Semiconductor (now NXP Semiconductors). It can be described as:

- synchronous;
- multi-master;
- multi-slave;

- packet switched;
- single-ended.

Each device connected to the bus is software-addressable by a unique address.

Two wires carry data (SDA - Serial DATA) and clock signals (SCL - Serial CLock), with the bus clock generated by the master

It makes use of open-drain connections for bidirectional communication which allows us to transmit a logic low simply by activating a pull-down FET, which shorts the line to ground. To transmit a logic high, the line is left floating, and the pull-up resistor pulls the voltage up to the voltage rail.

The PCF8574AT I/O expander for I²C-bus contains:

- 8-bit remote I/O pins (indicated as P0, P1, ..., P7) used to transfer data;
- 3 address pins (indicated as A0, A1, A2) used to address the slave.

In this project, it is not necessary to read data from I²C LCD, so only the writing mechanism will be described.

2.6.1 Writing mechanism

To allow a master to send data to a slave device:

1. the master, which acts as the transmitter, initiates communication by sending a START condition and addressing the slave, which acts as the receiver;
2. the master sends data to the slave-receiver;
3. the master terminates the transfer by sending a STOP condition.

A high-to-low transition on the SDA line while the SCL is high is interpreted as a START condition.

In a reverse manner, a low-to-high transition on the SDA line while the SCL is high is interpreted as a STOP condition.

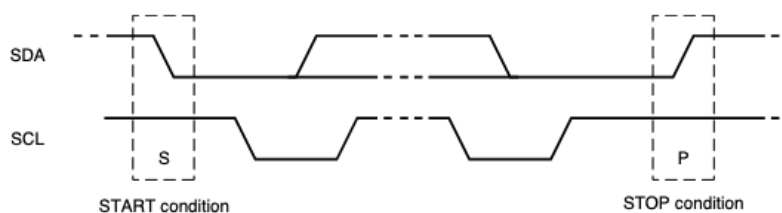


Figure 11: START and STOP conditions

A byte may either be a device address, register address, or data written a slave.

One data bit is transferred during each clock pulse. 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.

Any number of data bytes can be transferred from the master to slave between the

START and STOP conditions.

Data is transferred starting from the MSB (Most Significant Bit).

After each byte of data is transmitted, the master releases the SDA line to allow the slave-receiver to signal a successful transfer with an ACK (Acknowledge) or a failed transfer with a NACK (Not Acknowledge).

The receiver sends an ACK bit if it pulls down the SDA line during the low phase of period 9 of SCL. If the SDA line remains high, a NACK is sent.

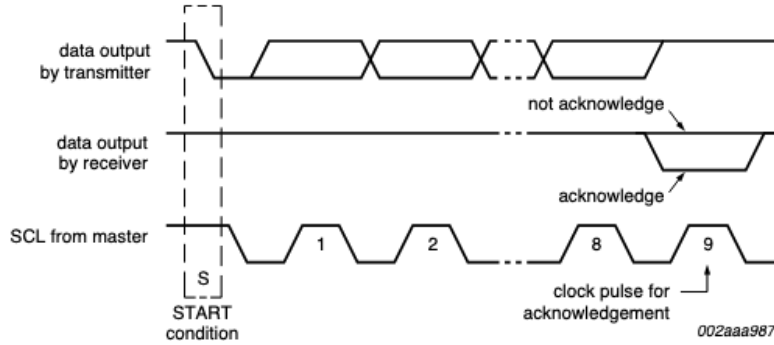


Figure 12: Acknowledgements

Thus, there are 6 steps for the writing mechanism:

1. the master sends the START condition and slave address setting the last bit of the address byte to logic 0 for the write mode;
2. the slave sends an ACK bit;
3. the master sends the register address of the register it wishes to write to;
4. the slave possibly acknowledges again;
5. the master starts sending data;
6. the master terminates the transmission with a STOP condition.

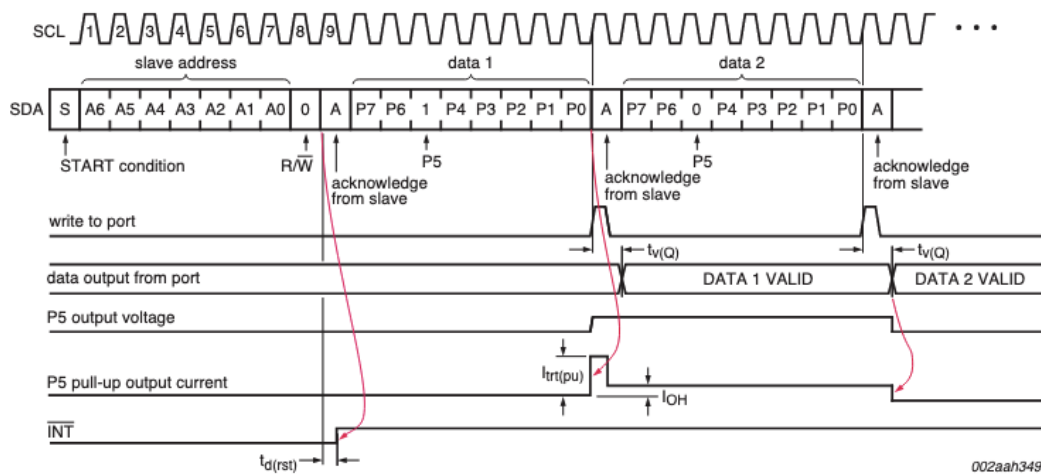


Figure 13: Write mode (output)

All this work is done by the PCF8574AT interface. All we need to do is to send the data byte for pins P7 to P0.

2.6.2 PCF8574AT to LCD connection

The PCF8574AT expander is soldered to the LCD pins according to the following scheme:

PCF8574AT pins	LCD pins
P0	RS
P1	R/W
P2	E
P3	Backlight
P4	D4
P5	D5
P6	D6
P7	D7

Table 1: PCF8574AT to LCD pin connections

Hence, the first time the LCD is turned on, it must be set to operate in 4 bit mode by sending the sequence:

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

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

Table 2: FUNCTION SET for 4 bit mode

After this setting, instructions can be sent by transmitting the most significant nibble first.

As an example, suppose you want to display an E, whose ASCII code is 0x45. To accomplish this task, the sequence to be sent is as follows:

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

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

Table 3: Sending an ASCII E

which means that the sequence to transmit is: 0x4D, 0x49, 0x5D, 0x59.

Command instructions are sent in the same way, except for the RS bit, which must be set to zero.

2.7 Schematics

All the hardware components shown in the previous sections have been connected with a breadboard to the Pi 4 as follows:

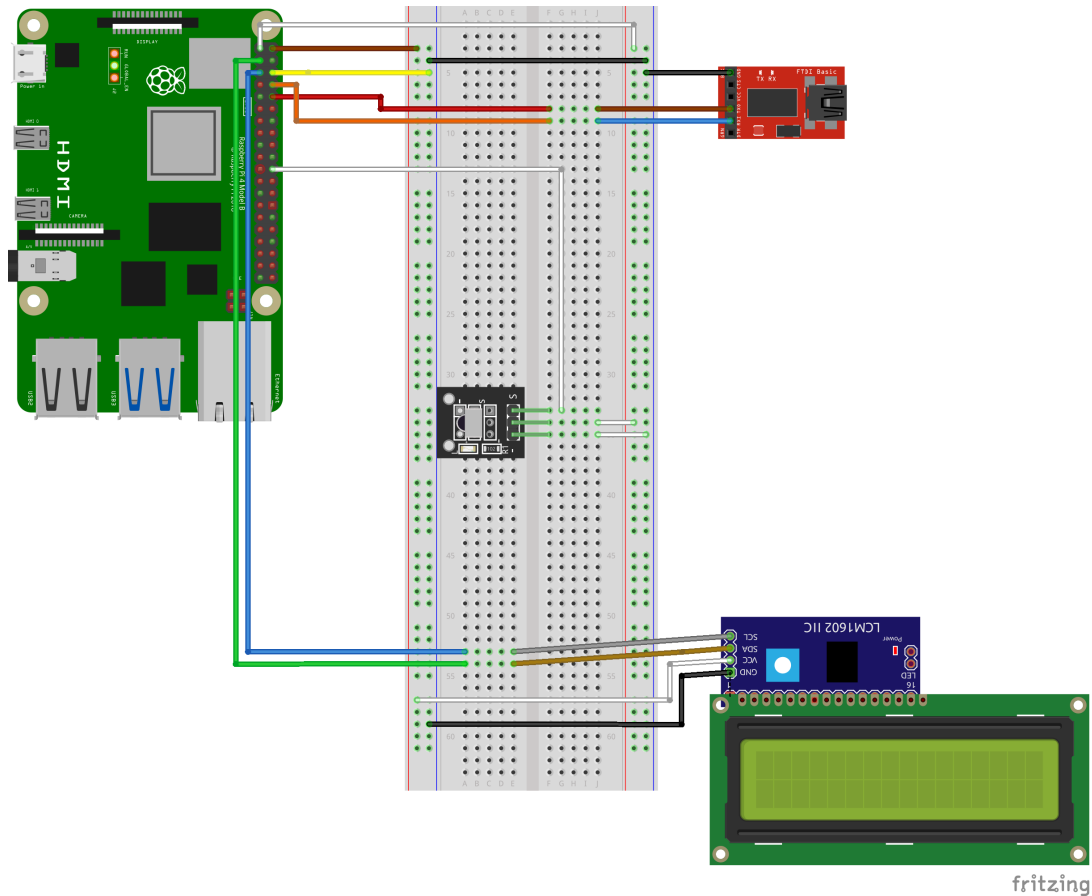


Figure 14: Schematics

The following table summarizes all the connections between them and the Pi:

Pi 4 Model B	KY-022	FT232RL	IIC PCF8574T
GPIO 25	S		
3V3 power	+		
GPIO 14 (TXD)		RX	
GPIO 15 (RXD)		TX	
5V power			VCC
GPIO 2 (SDA)			SDA
GPIO 3 (SCL)			SCL
Ground	-	GND	GND

3 Environment

3.1 pijFORTHos

Forth is a programming language designed by *Charles H. "Chuck" Moore* which can be implemented easily for resource-constrained machines due to its simplicity. It is a procedural language heavily based on the use of the stack.

There are several dialects, each with its own definitions. The development involves the definition of new words, which added to the pre-existing vocabulary, can create the final application incrementally and naturally.

pijFORTHos[11] is a bare-metal FORTH interpreter for the Raspberry Pi, based on Jonesforth-ARM[10].

Jonesforth-ARM is an ARM port of x86 JonesForth, which is a Linux-hosted FORTH presented in a Literate Programming style by *Richard W.M. Jones*.

pijFORTHos represents the base on which the application is built.

3.2 Minicom and Picocom

Minicom[12] is a terminal emulation program for Unix-like operating systems, used for communication and typically to set up a remote serial console.

Picocom[13] is, in principle, very much like minicom. It was designed to serve as a simple, manual, modem configuration, testing, and debugging tool.

ASCII-XFR[14] transfers files in ASCII mode. It is used to send the source file to the Raspberry because it allows a delay between each character and line sent. Given that the UART is asynchronous, this is a needed feature because it avoids overrun errors and lost characters if the receiver is busy while executing or compiling FORTH words.

Picocom is launched on the development machine through the command:

```
picocom --b 115200 /dev/cu.usbserial-A50285BI --send "ascii-xfr
↩ -sv -l100 -c10" --imap delbs
```

It is launched with the same VCP (Virtual COM Port) parameters as the Pi UART:

- `--b 115200`: 9600 bit/s bit rate;
- `/dev/cu.usbserial-A50285BI`: the serial UART device;
- `--send "ascii-xfr -sv -l100 -c10"`:

specifies `ascii-xfr` as the external command to use for transmitting files.

The options used are:

- `-sv`: verbose send mode;

- -l100: sets a 100 milliseconds delay after each line is sent, this usually gives enough time to run a command and be ready for the next line in time
- -c10: waits 10 milliseconds between each character sent.
- --imap delbs: : allows the use of backspace to delete a character.

3.3 Script

Since the serial UART has a limited operating speed, one way to speed up the loading of the entire project is to delete all comments and merge the files into a single file.

As not everyone has a Unix-like system operating, the first part of the script (`create_program.sh`) is made for Zsh, but can be easily changed to other shells because it simply prints the files to the stdout.

```

1  #!/bin/zsh
2
3  rm program.f 2> /dev/null
4  cat jonesforth.f      \
5      se-ans.f          \
6      utils.f           \
7      timer.f           \
8      i2c.f             \
9      lcd.f             \
10     ir_receiver.f     \
11     lookup_table.f    \
12     main.f            \
13     | ./unify_and_uncomment.py

```

The second part (`unify_and_uncomment.py`) is done in Python, because today almost everyone has a Python interpreter in their system. It reads from stdin, deleting comments and joining all words by separating them with a single space.

```

1  #!/usr/bin/python3
2
3  import sys
4  import re
5
6  with open('program.f', 'a') as f:
7      for line in sys.stdin:
8          # delete leading and trailing extra spaces
9          line = line.strip()
10         # skip comment lines
11         if line and line[0] != '\\':
12             # delete inline comments such as ( i1 i2 ... -- o1 o2 ... )
13             reg = re.search("\\s+\\(\\s+\\[~\\]*--\\[~\\]*\\s+\\)$", line)
14             if hasattr(reg, 'start') and hasattr(reg, 'end'):
15                 idx1 = reg.start()
16                 idx2 = reg.end()
17                 line = line[:idx1] + line[idx2:]
18             # delete inline comments such as \ ...
19             line = line[: line.find('\\') - 1 ] if line.find('\\') != -1 else line

```

```

20         # unify line by deleting extra spaces and skipping
21         # string containing only whitespaces
22         skip_whitespaces = lambda x: not x.isspace()
23         line = ' '.join( filter( skip_whitespaces, re.split('\s+', line) ) )
24         print(line, file=f, end=' ')

```

\ There are 8 Broadcom Serial Control (BSC) controllers, numbered from 0 to 7
 \ Only 6 of these masters can be used, because BSC masters 2 and 7 are not
 ↪ user-accessible.
 \ Since GPIO pins 2 and 3 are used, BSC1 is the reference master.
 804000 CONSTANT BSC1

\ There are 8 I2C registers, each of which is at an address obtained by applying
 ↪ an
 \ offset to BSC1 (this process is the same for all BSCs).
 \ I2C registers are:

\ - C register	The control register is used to enable interrupts, clear
↪ the FIFO,	
\	define a read or write operation and start a transfer;
\ - S register	The status register is used to record activity status,
↪ errors	
\	and interrupt requests;
\ - DLEN register	The data length register defines the number of bytes of
↪ data	
\	to transmit or receive in the I2C transfer. Reading the
\	register gives the number of bytes remaining in the
↪ current	
\	transfer;
\ - A register	The slave address register specifies the slave address
↪ and cycle type.	
\	The address register can be left across multiple
↪ transfers;	
\ - FIFO register	The Data FIFO register is used to access the FIFO. Write
↪ cycles	
\	to this address place data in the 16-byte FIFO, ready to
\	transmit on the BSC bus. Read cycles access data received
↪ from	
\	the bus;
\ - DIV register	The clock divider register is used to define the clock
↪ speed of the	
\	BSC peripheral;
\ - DEL register	The data delay register provides fine control over the
\	sampling/launch point of the data;
\ - CLKT Register	The clock stretch timeout register provides a timeout on
↪ how long the	
\	master waits for the slave to stretch the clock before
↪ deciding that	
\	the slave has hung.

\ The following constants are defined to point to the registers above.
 \ DIV, DEL and CLKT can be left whitout changes.

BSC1 PERI_BASE +	CONSTANT CTRL
BSC1 PERI_BASE + 04 +	CONSTANT STATUS
BSC1 PERI_BASE + 08 +	CONSTANT DLEN
BSC1 PERI_BASE + 0C +	CONSTANT SLAVE
BSC1 PERI_BASE + 10 +	CONSTANT FIFO


```

BSC1 PERI_BASE + 14 +      CONSTANT DIV
BSC1 PERI_BASE + 18 +      CONSTANT DEL
BSC1 PERI_BASE + 1C +      CONSTANT CLKT

\ Set slave address.
\ It can be left across multiple transfers.
: SET_SLAVE ( slave_address -- )
    SLAVE ! ;

\ Set number of data bytes to transfer.
: SET_DLEN ( length -- )
    DLEN ! ;

\ Place 8 bits at a time in FIFO in order to transmit them on the BSC bus.
: APPEND ( 8_bit_data -- )
    FIFO ! ;

\ Reset the control register without touching the reserved bits.
\ Reserved bits are in positions: 31:16, 14:11, 6 and 3:1.
\ Interrupts are disabled.
: RESET_CTRL ( -- )
    CTRL @ 87B1 BIC CTRL ! ;

\ Reset status for subsequent transfers without touching the reserved bits.
\ Only CLKT (9), ERR (8) and DONE (1) can be cleared (W1C type), all other flags
↪ are read-only (RO).
\ Reserved bits are in positions: 31:10.
: RESET_STATUS ( -- )
    STATUS @ 302 OR STATUS ! ;

\ Clear FIFO without touching the reserved bits.
\ - CLEAR (5:4) set to X1 or 1X in order to clear the FIFO before the new frame
↪ is started.
\ Interrupts are disabled.
: CLEAR_FIFO ( -- )
    CTRL @ 10 OR CTRL ! ;

\ Modify control register to trigger a transfer.
\ To start a new transfer, all bits are zero except for:
\ - I2CEN (15) set to 1 to enable the BSC controller;
\ - ST (7) set to 1 to start a new transfer (one-shot operation).
\ Interrupts are disabled.
: TRANSFER ( -- )
    CTRL @ 8080 OR CTRL ! ;

\ Data transfer through the I2C bus interface.
\ Since communication is established to the LCD panel, 8 bits at a time are sent.
: >I2C
    RESET_STATUS
    RESET_CTRL
    CLEAR_FIFO
    01 SET_DLEN
    APPEND
    TRANSFER ;

\ Setup the I2C bus interface and the slave address.

```

```

\ Configure GPIO pin 2 for Serial Data Line.
\ Configure GPIO pin 3 for Serial Clock Line.
\ Set the slave address to 0x27.
: INIT_I2C
    02 ALTO CONFIGURE
    03 ALTO CONFIGURE
    27 SET_SLAVE ;

\ Consider the following structure of data transfer:
\      D7 D6 D5 D4 Backlight Enable Read/Write Register-Select
\ equivalently:
\      D7 D6 D5 D4 BL EN RW RS
\ In order to send a byte, it must be decomposed in two nibbles, upper nibble and
↪ lower
\ nibble. Each nibble represented by D7 D6 D5 D4 must be followed by a
↪ combination of
\ BL EN RW RS.
\ For each data or command to transfer RW = 0.
\ Given a byte B = HIGH LOW (upper-nibble lower-nibble), if it is part of a
↪ command,
\ then transfer is obtained by sending:
\   HIGH 1 1 0 0 -> HIGH 1 0 0 0 -> LOW 1 1 0 0 -> LOW 1 0 0 0
\ If it part of a data transfer then:
\   HIGH 1 1 0 1 -> HIGH 1 0 0 1 -> LOW 1 1 0 1 -> LOW 1 0 0 1
\ RS is equal to 0 for instruction input and it is equal to 1 for data input.

\ Returns setting parts to be sent based on a truth value that indicates whether
↪ the input
\ is part of a command or data.
: SETTINGS ( truth_value -- first_setting second_setting )
    IF
        0C 08
    ELSE
        0D 09
    THEN ;

\ Returns a nibble aggregated with the first setting part and the second setting
↪ part.
: AGGREGATE ( settings byte -- nibble_second_setting nibble_first_setting )
    04 LSHIFT DUP ROT OR -ROT OR ;

\ Divides a byte into two nibbles.
: BYTE>NIBBLES ( byte -- lower_nibble upper_nibble )
    DUP OF AND SWAP 04 RSHIFT OF AND ;

\ Send a nibble to LCD aggregated with settings.
: SEND_NIBBLE ( nibble truth_value -- )
    SETTINGS ROT
    AGGREGATE
    >I2C 1000 DELAY
    >I2C 1000 DELAY ;

\ Transmits input to LCD given an instruction or data.
: >LCD ( input -- )
    DUP 08 WORD>BIT >R

```

```
BYTE>NIBBLES R@  
SEND_NIBBLE R>  
SEND_NIBBLE ;
```

References

- [1] E. Upton. “Raspberry Pi 4 on sale now from \$35.” (2019), [Online]. Available: <https://www.raspberrypi.org/blog/raspberry-pi-4-on-sale-now-from-35/>.
- [2] *Raspberry pi 4 product brief*, <https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf>.
- [3] *Raspberry pi 4 gpio*, <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html>.
- [4] *Ftdi adapter ft232rl*, <https://www.az-delivery.de/it/products/ftdi-adapter-ft232rl>.
- [5] *Elegoo documentation*, https://drive.google.com/file/d/1jflQ6lDgM_gSl604r_zW7n1IvWrFUKN9/view.
- [6] *Lcd 1602*, <https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf>.
- [7] *I2c lcd 1602*, <https://wiki.52pi.com/index.php?title=Z-0234>.
- [8] *Pcf8574*, https://www.nxp.com/docs/en/data-sheet/PCF8574_PCF8574A.pdf.
- [9] *Samsung protocol*, https://www.techdesign.be/projects/011/011_waves.htm.
- [10] *Jonesforth-arm*, <https://github.com/M2IHP13-admin/JonesForth-arm>.
- [11] *Pijforthos*, <https://github.com/organix/pijFORTHos>.
- [12] *Minicom*, <https://linux.die.net/man/1/minicom>.
- [13] *Picocom*, <https://linux.die.net/man/8/picocom>.
- [14] *Ascii-xfr*, <https://man7.org/linux/man-pages/man1/ascii-xfr.1.html>.