MSX Pi Interface v. 0.8.2
Ronivon Candido Costa
old@retro-cpu.run

Technical Reference Manual

Table of Contents

1 MSXPi Overview

MSXPi is a solution to allow MSX computers user Raspberry Pi devices as generic peripherals. I say it is a solution because it is a set of software and hardware components put together to allow such functionality.

MSXPi hardware is composed of a MSX compatible interface to connect to the slot cartridge of any MSX 1,2,2+ and Turbo R. Due to some specific hardware implementation of some MSX models, the interface may not work on all MSX models.

MSXPi software is composed of different components running on MSX and on Raspberry Pi. On the Raspberry Pi side, there is the msxpi-server component which listen for GPIO events, decode serial data using a few pins and transform that data into control commands and parameters.

On the MSX side, there are different software components depending on how the user wants to use MSXPi.

- MSX-DOS 1 is stored in the MSXPi EPROM. This custom version has been modified to use MSXPi drivers to allow boot from disk images stored on Raspberry Pi. The EPROM also contain a set of CALL commands to user from BASIC.
- MSXPi Cloud Client, which is invoked by “call msxpiload” and is then download from Rasperry Pi. This client has a command line interface (somewhat like MSX-DOS command line), and allows to run some commands to interact with Raspberry Pi (deprecated).
- MSX-DOS (1,2,Nextor) commands. These are “.com” commands that implement a series of functionalities on MSX-DOS allowing greater experience with MSXPi. This should be the main development line since it covers most of the users and allows for greater flexibility.
- Nestor File System (NSF), only available for MSX-DOS2 and Nextor (This is currently under development). This method creates a virtual drive under MSX DOS, and all access to that virtual drive will actually be accessing Raspberry Pi resources. These resources can be the file system or a network resource.
- MEXPIEXT.BIN with CALL commands to extend BASIC to access MSXPi without the need of the ROM. This program adds a set of CALL commands in the RAM area in $4000, allowing new commands to be installed from DISK runnning “msxiext.bin”.

In the next sections we will discuss the resources currently available for MSXPi, and how to use them.

1.1 What's new in version 0.8.2

This versions adds two main changes to MSXPi, but still allowing compatibility with all previous versions.

- CALL commands instalable from disk (msxpiext.bin)

  MSXPi commands are being ported to allow compatibility with BASIC when called using CALL. Not all commands will be compatible, but some are (such as PRUN) and allow exhange of data with Pi in BASIC. New commands will be created specifically for BASIC.

- New server in Python

  The msxpi-server has been ported to Python. This improves development time, at the same time allowing same level of transfer rate since the transfer of blocks of data are still using the C function.

2 Users Guide for Interface v0.7 with Software v0.8.2

This release comes with an EPROM containing MSXPi-DOS 1.04. This is a custom DOS, modified from MSX-DOS 1.03.

The EPROM software must match the Raspberry Pi server component, or some commands will fail. A newer version of the server component can be used, but not an older version.

MSXPi boots into MSXPi-DOS 1 from drive A: which is mapped to the disk image in /home/pi/msxpi/disks/msxpiboot.dsk.
Drive B: is mapped to /home/pi/msxpi/disks/msxpitools.dsk which contains all currently available DOS commands to use with MSXPi and some other tools.

Both disk images are provided in the MSXPi distribution.

<u>Note:</u> The MSXPi-DOS in MSXPi ROM can be skipped pressing “P” before MSXPi connection test starts. This allow skipping the ROM contents, and boot into another DOS cartridge such as ATA-IDE or MFR. Unfortunately this does not work on all MSX models.

MSXPi specific commands starts with “P”, such as “PDIR”, “PDATE”, “PCOPY” and so on. There are a few exceptions though, such as the command “PRUN” to run commands directly on Rapsberry Pi.

From MSXPi-DOS, you can use all resources available to MSX-DOS1, such as game loaders and other programs.

When booting from MSXPi, you can also have a second disk drive interface attached (such as ATA-IDE or MegaFlashRom SD). These drives will be accessible from MSX-DOS1 and you can exchange files between MSXPi disk images and those drives.

The MSXPi tools and usage will be covered in a later section.

2.1 MSXPi ROM

The EPROM contain also a set of CALL commands available to the user from BASIC prompt:

- CALL MSXPIVER
  Display the ROM version and available commands

- CALL MSXPISTATUS
  Test connection with Raspberry Pi and display status

- CALL MSXPILOAD
  Load the MSXPi Client from Raspberry Pi, and execute. The next section provide information on how to use this client.

2.2 MSXPIEXT.BIN

MSXPIEXT.BIN is an extension for BASIC. This command should be loaded and executed to add new CALL commands to MSX-BASIC.

To install the extension, run the following command from BASIC:

bload”msxpiext.bin”,r

A short help is diplayed.

These commands use a buffer to exchange data with Rpi. This buffer has the following format:

Return code	Size (lsb)	Size (msb)	Data (this area size=lsb+256*msb)

The followind commands are available after installing the extension:

CALL MSXPI(<”***output,buffer, command***”>)

This command does not currently use the return code and size areas, and the data starts at the address passed.

output:

0 : do not print output

1: print output to screen (this is the default)

2: store output in buffer

buffer:

4 digit hex memory address. For example, “C000”

command:

Any supported command, such as “pdir”, “pcd”, “prun”, etc.

CALL MSXPISEND(“<***buffer***>”)

Send contents of buffer to RPi (it uses the SENDDATABLOCK function from msxpi_bios.asm). This command uses the buffer format as specified previously. The First byte should be left unused, and the next two bytes should contain the size of the data to transfer.

CALL MSXPIRECV(“<***buffer***>”)

Read data from RPi and store in the buffer (it uses the RECVDATABLOCK function from msxpi_bios.asm). After completing the transfer, the first byte will contain the return code and the next two bytes will contain the number of bytes received.

2.3 Examples for MSXPi Extension

2.3.1 CALL MSXPI

```
call msxpi(“pdir”)

call msxpi(“0,pdir”)

call msxpi(“1,pdir”)

call msxpi(“2,D000,pdir”)
```

Note: in this last example, the output of the command is saved in the buffer starting at address #D000.

2.3.2 CALL MSXPISEND

In this example, we will send data (a simple 'TST command) to Rpi and verify the return code received.

```
10 gosub 100 ' Send a command to RPi
20 rc=peek(&hD000): ? hex$(rc) ' Get Return Code
50 end
99 ' Send data to RPi – a text command in this case
100 POKE &HD000,0 ' This is the area reserved for the return code
110 POKE &HD001,3:POKE &HD002,0 ' This is the buffer size (LSB,MSB)
120 POKE &HD003,ASC('T')
130 POKE &HD004,ASC('S')
140 POKE &HD005,ASC('T')
150 CALL MSXPISEND(“D000”) ' Send command TST to Rpi
160 RETURN
```

2.3.3 CALL MSXPIRECV

This example receive data from Rpi and print on screen. For this reason, is is convenient that the data is ascii in a valid range so it will be correcly displayed on screen. This same routine works with any binary data.

```
10 gosub 100 ' Receive data from RPi
20 rc=peek(&hD000): ? hex$(rc) ' Get Return Code
30 if rc=&hE0 then gosub 200   ' If RC=RC_SUCCESS then print data
50 end
99 ' Receive soma data from RPi
100 POKE &HD000,0 ' This is the area reserved for the return code
110 POKE &HD001,0:POKE &HD002,0 ' This is the area reserved for buffer
size
120 CALL MSXPIRECV(“D000”) ' Received data from RPi
```

```
130 RETURN
200 S=PEEK(&HD001)+256*PEEK(&HD002) ' Get size of buffer
210 FOR M = 0 TO S-1 ' Will read all bytes iin the buffer...
220 PRINT CHR$(PEEK(&HD003)+M); ' and print on screen.
230 NEXT M:RETURN
```

2.4 MSXPi Client Commands

Note: This client is deprecated. Please use the commands from MSX-DOS or CALL from BASIC.

After starting the client with command “call msxpiload”, the following is displayed in the screen.

Note: List of available commands may vary depending on the version you are using, since MSXPi is under constant development.

```
MSXPi Hardware Interface v0.7
MSXPi Cloud OS (Client) v0.8.1
TYPE HELP for available commands
CMD:
```

The available commands can be viewed with the HELP command:

```
CMD:HELP
BASIC CHKPICONN CLS PDIR  HELP PLOADBIN PLOADROM PMORE #(Pi Command) PRESET
```

CHKPICONN: will verify if Pi is responding. Result will be printed on screen, depending on status.
DIR: will show files in current path. Dir will work for files on the Raspberry Pi filesystem, and also remote

PLOADBIN/PLOADROM: Load a binary program into memory. Currently supported formats are binary files to run from BASIC (PLOADBIN), and .rom (PLOADROM for roms up to 32KB. The loader is very basic and cannot load all rom types.
File is loaded into the address extracted from the header, even for a ROM file. In this case, inter-slot calls are made to write the rom into page 1 (0x4000), which makes the load process slower.

RESET: Sent reset command to both MSXPi interface and MSXPi Server application.

# (PRUN on MSX-DOS): This command passes all commands directly to the Raspberry Pi for execution. The command is executed, and the output printed in the MSX screen. It does not support input to the command, such as prompts. The command must execute and terminate, returning control to the MSXPi.
Examples of usage:
#ls

```
#pwd
#hostname
#unzip msxgame.zip
```

2.5 For ROM-less MSXPi

MSXPi prototypes that was released without EPROM should be with the “P” commands from MSX-DOS or loading the MSXPIEXTENSION.
Optionally, and only for tests purposes, it can also be used with a MegaFlashRom (MFR) cartridge. The file MSXPi-DOS_16K.ROM should be written to the Megaram or MFR and MSXPi will work as a regular MSXPi with integrated EPROM.
This method can also be used to test new ROM releases before writing it to the EPROM.

3 MSXPi Architecture and Behavioural Description

Following picture illustrate the high level design of the MSXPi interface.

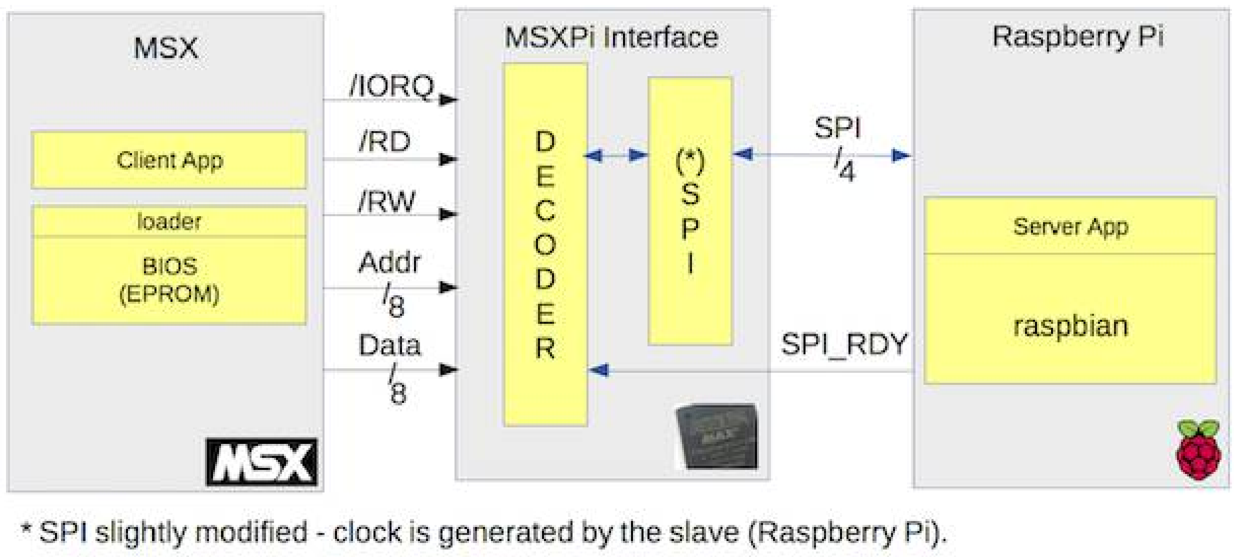


Figure 1: Architecture

1. MSX starts a transfer with Pi, by one of two means:
1a. MSX send a read signal to Pi, by writing 0 to port 0x56;
1b. MSX send a write signal to Pi, by writing a value to port 0x5A.
2. CPLD decodes addresses and buffers the MSX byte.
3. SPI sends a signal to Pi to start a new transfer, lowering SPI_CS.
4. Pi send back to the Interface the busy signal, lowering SPI_RDY.
5. Pi starts generating clock pulses into SPI_SCLK pin.
6. Bit shifter inside CPLD convert PI serial bits to a byte, and MSX byte to serial bits. Transfer is done in both ways (full-duplex). Pi byte is buffered into the CPLD.
7. MSX read port 0×5A to receive the byte send by Pi.

The interface uses serial transfer between CPLD and Raspberry Pi. Because of this technical feature, it is not ideal for some applications that require high throughput, such as graphics applications. Currently the benchmark transfer rate is approximately 5KB per second (Kilo Bytes / s).

The interface between MSX and Pi is made through a 5 volt CPLD tolerant, the EPM3064 from Altera. In this CPLD is implemented a logic transfer a byte between MSX and Raspberry Pi in serial full duplex mode using a variant of the SPI protocol with a small change in the generation of the clock, which is being generated by the pi (slave) instead of being Generated by MSX or by the interface itself. This change allows the Pi to specify the transfer speed.
The client application was developed in the Assembly Z80 language, and the server application in C and also Python.
On the Raspberry side, the application runs on the Raspbian, and is automatically initialized every time the Pi is connected by the systemd service.

By using a standard linux system, the solution allows a wide range of applications to be easily developed through the use of existing tools in Raspberry, both for programming and for accessing Pi GPIOs.

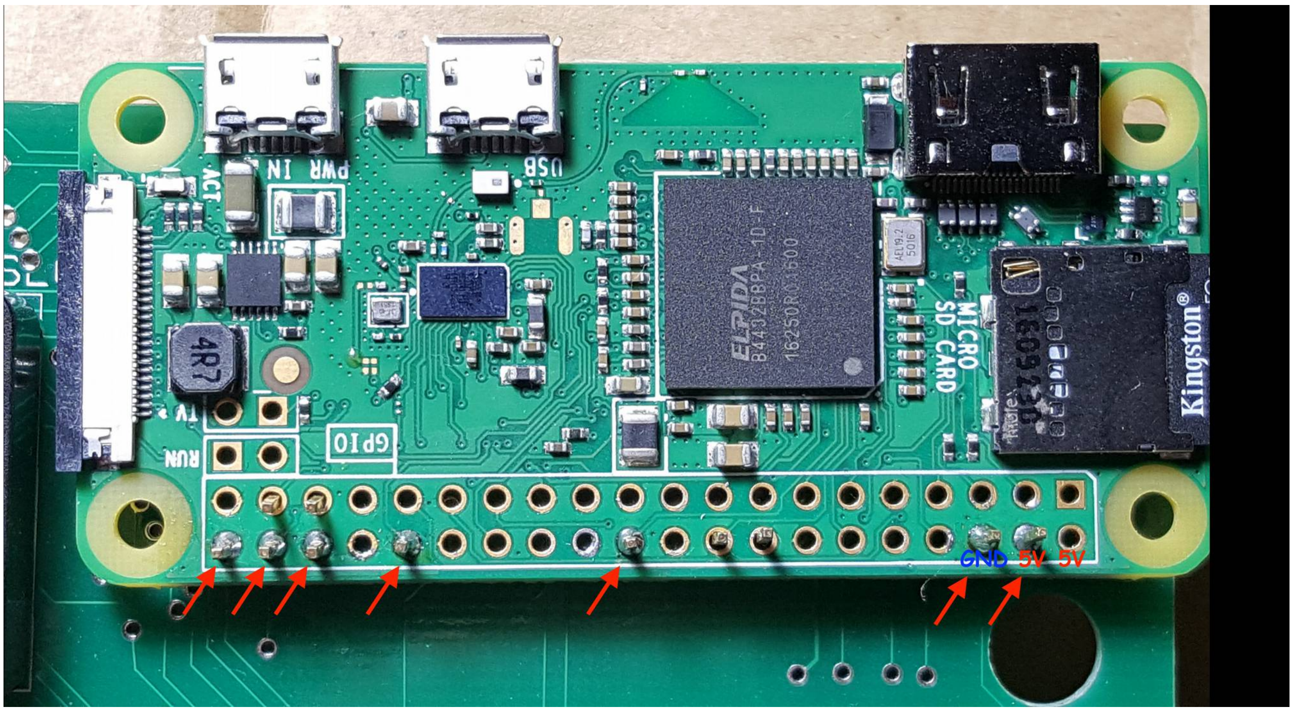
4 Connecting MSXPi to the Raspberry Pi

The interface is compatible with any model of Raspberry Pi, although it was developed for the "Zero" model, so that it is fully incorporated into the cartridge. The only exposed parts will be SD, USB and LED card slots.
The GPIO pins (see Appendix 1) of the Pi used by the interface are:

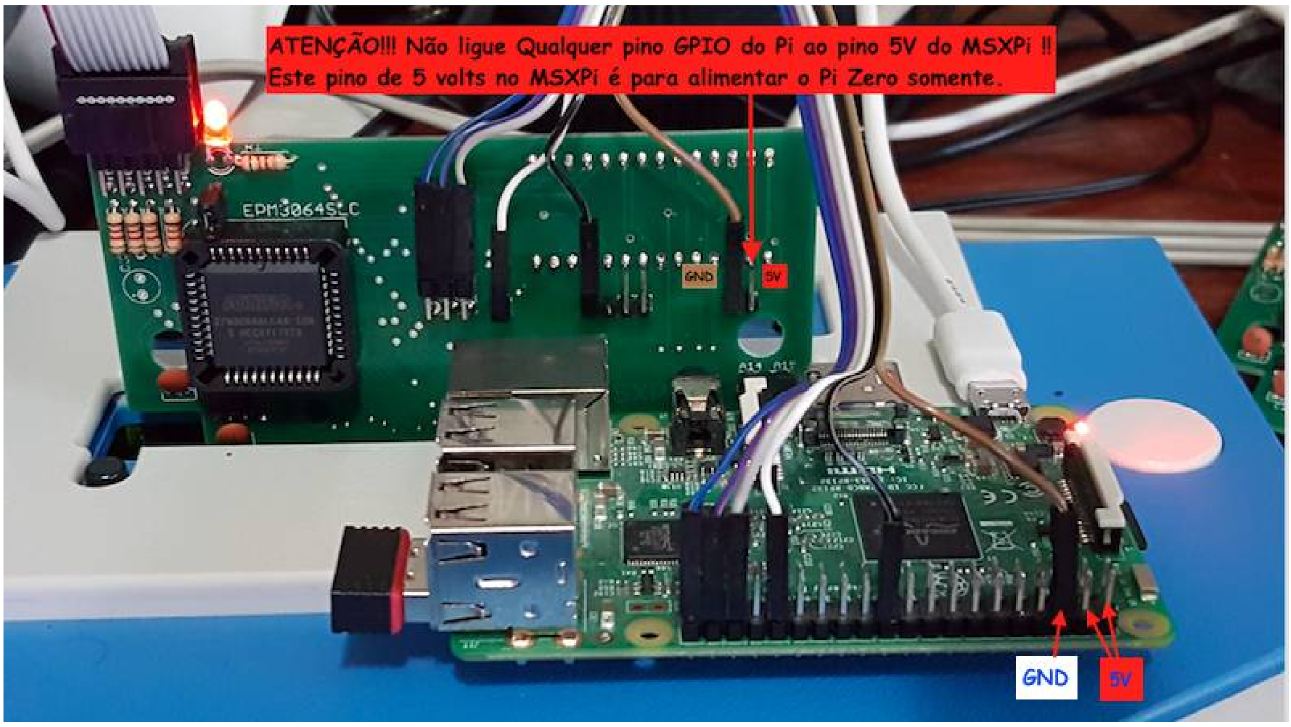
GPIO	Purpose	Direction	Enabled	CPLD Pin
21	/CS (SPI Chip Select)	Input	0	28
20	SCLK (SPI Clock)	Output	N/A	34
16	MOSI (Interface out)	Input	N/A	5
12	MISO (Interface in)	Output	N/A	41
25	PI_Ready	Output	1	12
	GND (Pin 4)	N/A	N/A	
	VCC (Pin 6)	N/A	N/A	

When using RPi Zero, power is supplied by MSX.
When using RPi 1,2 or 3, RPi must be connected to an external power supply.

To attach Raspberry Pi Zero to the interface, use the photo in illustration 2 as a reference. As for software v0.8.1, only 7 PINs are used on RPi 0 as indicated with the red arrows. Soldering is only required in the signaled pins.



To attach a Raspberry Pi model B +, 2 and 3 to the interface, use the picture in illustration 3 as a reference.


ATENÇÃO!!! Não ligue Qualquer pino GPIO do Pi ao pino 5V do MSXPi !!
Este pino de 5 volts no MSXPi é para alimentar o Pi Zero somente.
EPM3064SLC
GND
5V
GND
5V

<u>Note</u>: It is essential that the MSXPi GND (brown wire) is connected to the Raspberry Pi. Without this connection, the signals on the interface pins will not be transmitted correctly.

5 MSXPi Programmer's Reference

This section describes the MSXPi specific IO library. This library is hardware-dependent and should be always be used instead of trying to access the hardware directly. This will allow for greater compatibility with future releases of the hardware.

The MSXPi software library consist of a set of Z80 assembly routines on the MSX side, and a C and Python framework on the Raspberry Pi side. In this section we focus on the MSX-side library.

The MSXPi project has the following structure:

/asm-common/include/include.asm → contain constant definitions and MSX BIOS labels

/asm-common/include/basic_stdio.asm → contain the PUTCHAR call for programs in BASIC environment and cartridge programs (such as MSX-DOS driver)

/asm-common/include/msxdos_stdio.asm → contain the PUTCHAR call for MSX-DOS programs, and also a MSX-DOS wrapper for the SENDPICMD bios routine.

/asm-common/include/msxpi_bios.asm → Contain all data transfer and other bios functions (hardware-independent) used by MSXPi programs.

/asm-common/include/msxpi_io.asm → This file contain the hardware-dependent I/O functions for MSXPi. All functions in this library access MSXPi hardware ports, and are called by functions in the libraries mentioned previously.

/asm-common/include/msxpi_api.asm → contain extra-routines that might be useful.

Next section will describe the bios functions only.

5.1 MSXPi I/O (MSXPI_IO.ASM)

CHKSPIRDY

Label: CHKPIRDY

Purpose	: Read the MSXPi Interface status register (port 0x56)to determine the status of the Interface and Pi Server App. This function should return zero when Pi is responding, and 1 when not listening for commands.
Input	: None
Output	: Flag C: Set when Pi not responding.
Registers	: AF is modified

Usage notes: Pi is determined as responding when I/O control port 56h returns zero. This routine loop 65535 times before it will return with error (zero not detected on that port).

PIEXCHANGEBYTE
Label: PIEXCHANGEBYTE

Purpose : Send a byte to Pi (write to data port 5AH) and read a byte back. Usually, the byte read at this cycle is meaningless because Pi send whatever is available on its internal register during any transfer. If you send a byte expecting to receive a valid response, than you must call this function a second time (this is because Pi needs time to process your byte and execute whatever command it need to produce the expected answer).
Input : Register A: byte to send.
Output : Flag C: Set if there was an error
Register A: byte received
Registers : AF is modified

PIREADBYTE
Label: PIREADBYTE
Purpose : Read a byte from the MSXInterface. This command send value 0 to control port 56h, then call CHKPIRDY to know when a byte is available to be read. In the sequence, the routine read data port 5Ah and return the byte in register A.
Input : None.
Output : Flag C: Set if there was an error
Register A: byte received
Registers : AF is modified

PIWRITEBYTE
Label: PIWRITEBYTE
Purpose : Send a byte to Pi. This function call CHKPIRDY to know when Pi is available to receive data, and in the sequence write the data to the data port 5Ah.
Input : Register A: byte to send to Pi
Output : None
Registers : No registers are modified

SENDIFCMD
Label: SENDIFCMD
Purpose : Send a single-byte command to the MSXInterface (port 56h).
Input : Register A: Byte/Command to send.
Output : None.
Registers : None.

Usage notes: Current implemented commands are:

- Reset (0FFH) – Sending 0FF will force a RESET of the MSXPi interface internal state.

- Status (00h) – Reading this port will report the MSXPi state, being 0 = available.

5.2 MSXPi BIOS (MSXPI_BIOS.ASM)

SYNCH
Label: SYNCH

Purpose: Enforce a reset of MSXPi interface, and try to communicate with Pi.
Input: None
Output: Flag C: Set if connection failed.
        Register A: Value returned by Pi if was able to connect, or error code.

Usage note: This function sends command “SYN” to Pi. It does nothing on Pi, but function a successful execution assure Pi is responding and waiting a command.

SENDPICMD
Label: SENDPICMD
Purpose: Send a command to Raspberry Pi
Input:
  DE = should contain the command string
  BC = number of bytes in the command string
Output:
  Flag C set if there was a communication error
Modifies: AF, BC, DE, HL

RECVDATABLOCK
Label: RECVDATABLOCK
Purpose: Receive a block of data from PI. Calculate CRC using simple XOR of bytes received, and exchange with Pi at the end of the transfer.
Input:
  DE = memory address to write the received data
Output:
  Flag C set if error
  A = error code
  DE = Original address if routine finished in error,
  DE = Next current address to read if finished successfully
Modifies: AF, BC, DE, HL

SENDDATABLOCK
Label: SENDDATABLOCK
Purpose: Send a number of bytes to Pi. Calculate CRC using simple XOR of bytes received, and exchange with Pi at the end of the transfer.
Input:
  BC = number of bytes to send
  DE = memory to start reading data
Output:
  Flag C set if error
  A = error code
  DE = Original address if routine finished in error,
  DE = Next current address to read if finished successfully
Modifies: AF, BC, DE, HL

SECRECVDATA
Label: SECRECVDATA

Purpose: Read data from Pi, 512 bytes at a time, and retry a number of times defined in GLOBALRETRIES contant.
Input:
  DE = memory address to start storing data
Output:
  Flag C set if error
Modifies: AF, BC, DE, HL

SECSENDDATA
Label: SECSENDDATA
Purpose: Send data to Pi, 512 bytes at a time and retry a number of times defined in GLOBALRETRIES contant.
Input:
  BC = number of bytes to send
  DE = memory to start reading data
Output:
  Flag C set if error

DOWNLOADDATA
Label: DOWNLOADDATA
Purpose: Load data using configurable block size. Every call will read next block until data ends.
Input:
  A = 1 to show dots for every 256 bytes
  BC = block size to transfer
  DE = Buffer to store data
Output:
```
  Flag C: Set if occurred and error during transfer, such as CRC
       Z: Set if end of data
          Unset if there is still data
       A: Error code
          A = error code, or
          A = RC_SUCCESS - block transferred, there is more data
          A = ENDTRANSFER - end of transfer, no more data.
```
Modifies: AF, BC, DE, HL

5.3 BIOS Functions Address Table in ROM

This table show the actual addresses for the MSXPi bios functions in the ROM. To use these functions, the slot where MSXPi is connected must be first identified, and then an interslot call can be made to the required address.

Note that the functions are not aware of slot switching and ram banks, which means that any buffer provided to the function must be directly accessible by the function.

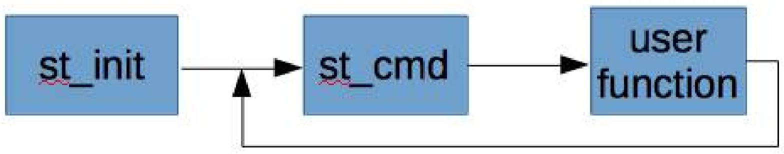
Address	Function
7602	MSXPIVER
7608	MSXPISTATUS
7630	MSXPILOAD
7886	RECVDATABLOCK
78B5	SENDDATABLOCK
78EA	SECRECVDATA
791A	SECSENDDATA
7946	READDATASIZE
7953	SENDDATASIZE
7AAC	CHKPIRDY
7ABD	PIREADBYTE
7ACB	PIWRITEBYTE
7AD3	PIEXCHANGEBYTE
7AA9	SENDIFCMD
7882	SENDPICMD
7A12	CHECKBUSY
7A2C	PRINT
7A4A	PRINTNLINE
7A55	PRINTNUMBER
7A6A	PRINTDIGIT
7A79	PRINTPISTDOUT
7858	SYNCH
795C	DOWNLOADDATA
79A9	UPLOADDATA

To identify the MSXPi slot, you need a search routine (not currently provided). Different methods can be used to find where MSXPi is connected, but you can use the following addresses and string patterns:

Address	String
76F2	MSXPi Hardware Interface v0.7
7711	MSXPi ROM v0.8.1
772F	<14 characters in this position contain the build id>

5.4 MSXPi server in C

The server application that runs on Raspberry Pi is implemented using a state machine and functions implementing the commands, as shown in the following diagram.



The change of states occurs as commands are received by Pi. Each command should be parsed inside the “st_cmd” state, and call a function that implements the actions required by the command. For the benefit of productivity, the re-implementation of the full server application is discouraged.

The low-level functions that communicate with MSXPi Interface are implementing using a bit-bang SPI-like protocol, thus it is serial. Due to this, performance is not impressive, but it is an easy implementation using simple and cheap hardware.

The main state of the application is "st_cmd", which validates incoming commands, initializes all attributes necessary to execute each command, and call the specific function for the command.

The code structure that decodes a command and prepares the call to the function is shown below.

```
} else if((strncmp(msxcommand,"PDIR",4)==0) ||
          (strncmp(msxcommand,"pdir",4)==0)) {
              printf("PDIR\n");
              if (pdir(msxcommand)!=RC_SUCCESS)
                 printf("!!!!! Error !!!!!\n");
              appstate = st_cmd;
              break;
```

For the given example above, the functions is implemented as show next:

```
int pdir(unsigned char * msxcommand) {
   memcpy(msxcommand,"ls  ",4);
   return runpicmd(msxcommand);
}
```

5.5 MSXPi Server in Python

Soon.

6 SPI Implementation in the MSXPi Interface

Communication between MSX and Raspberry Pi is facilitated by the MSXPi Interface, which implements a serial protocol using CPLD technology. The protocol uses five GPIO pins and full duplex transfers.
The GPIO pins implements the signals:

- CS (Enable signal from MSX, tells Pi that transfer should start)
- SCLK (Clock signal from Pi, tells MSX when to drive GPIO signals)
- MOSI (Data from MSX to Pi)
- MISO (Data from Pi to MSX)
- SPI_RDY (Ready signal, tells MSX when Pi is read to start a byte transfer)

When Pi is ready to process a transfer, it drive the SPI_RDY signal high.
When MSX wants to start a byte transfer, it checks SPI_RDY. If it is high, MSX drive CS low.
When CS toggle states, Pi jumps to an interrupt function to process the transfer if the signal went low. In this case, it also drive SPI_RDY low to tell MSX that it cannot send another byte.
PI starting generating the clock signal SCLK to synchronize the transfer, and send the 8 bits to MSX. MSX will receive each bit and use bit shift to store in a buffer.
When 8 bits are transferred, PI bring SPI_RDY high again to tell MSX that it can receive more data. The MSXInterface will keep the byte in its buffer. MSX can now send a read command to port 0x5A to get the byte.

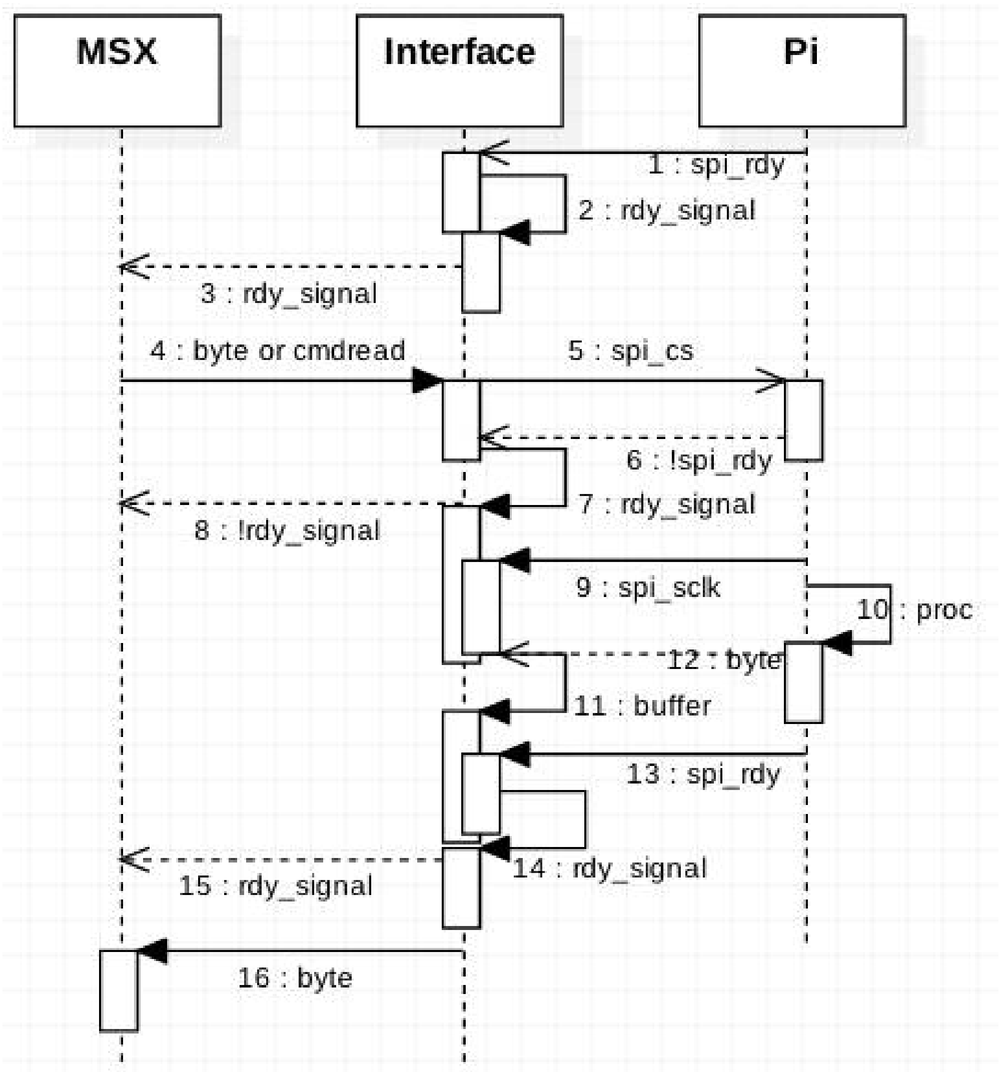


Illustration 2: Diagram Sequence for the Interface

7 Appendix 1: Development Tutorial

This appendix will be expanded (hopefully some day) with some useful contents.
As for now, please refer to the Documents/DevTemplate directory in the git repository for a template to use for development.
Also refer to the source codes available, specially under Client directory, for examples of implementation.

8 Appendix 2: Pi - GPIO Pin Numbering

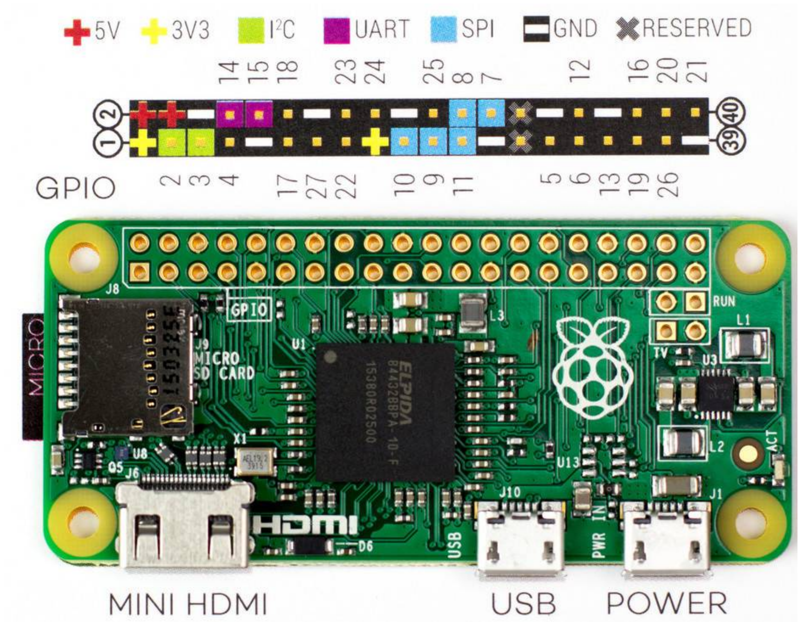


Illustration 4: GPIO models B +, 2, 3 and Zero.

The GPIO numbering of illustration 4 is valid for B +, 2, 3 and Zero models.

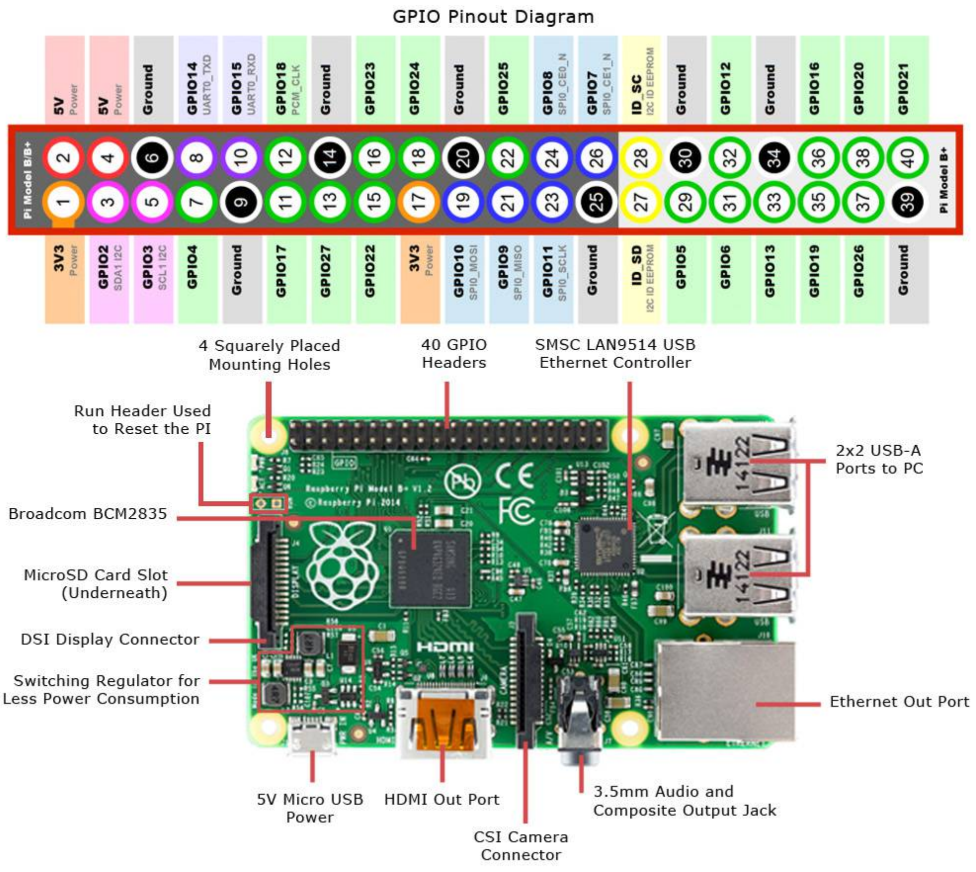


Illustration 5: GPIO models B +, 2, 3 and Zero.