ZolaDOS

|  |  |
| --- | --- |
| Last edited: | 14 Sep 2025 |

ZolaDOS is being run as a systemd service on imp and zd0. The unit file is: /etc/systemd/system/zolados.service.

To update the software, run goup on the dev machine. This won't upload because the target is set to 'local'.

On imp/zd0, run the getz script.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Zolatron | | RPi | |  | RPi | | Zolatron | |
|  |  | 1 | +3.3V |  | +5V | 2 |  |  |
|  |  | 3 | GPIO2 SDA |  | +5V | 4 |  |  |
|  |  | 5 | GPIO3 SCL |  | GND | 6 |  |  |
| D0 | PA0 | 7 | GPIO4 |  | TX GPIO14 | 8 | RX on DUART PORTA | |
|  |  | 9 | GND |  | RX GPIO15 | 10 | TX on DUART PORTA | |
| D1 | PA1 | 11 | GPIO17 |  | GPIO18 | 12 | PA2 | D2 |
| D3 | PA3 | 13 | GPIO27 |  | GND | 14 |  |  |
| D4 | PA4 | 15 | GPIO22 |  | GPIO23 | 16 | PA5 | D5 |
|  |  | 17 | +3.3V |  | GPIO24 | 18 | PA6 | D6 |
|  |  | 19 | GPIO10 |  | GND | 20 |  |  |
|  |  | 21 | GPIO9 |  | GPIO25 | 22 | PA7 | D7 |
|  |  | 23 | GPIO11 |  | GPIO8 | 24 |  | Ext. LED |
|  |  | 25 | GND |  | GPIO7 | 26 |  | /IRQ |
|  |  | 27 | X |  | X | 28 |  |  |
| /CA | PB0 | 29 | GPIO5 |  | GND | 30 |  |  |
| /CR | PB1 | 31 | GPIO6 |  | GPIO12 | 32 | PB2 | /CO |
|  |  | 33 | GPIO13 |  | GND | 34 |  |  |
| /SR | PB4 | 35 | GPIO19 |  | GPIO16 | 36 | PB5 | /SA |
| S4 | PB6 | 37 | GPIO26 |  | GPIO20 | 38 | PB7 | INTSEL |
|  |  | 39 | GND |  | GPIO21 | 40 |  | /RST |

PB3 on 65C22 goes to DDIR pin on level shifter.

# Control Signals

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Client (Z64)** |  | dir |  | **Server (RPi)** |
| **Client Active**  Taken low by the Z64 to indicate that it wishes to initiate an action. It's held low by the Z64 until everything is done.  When **sending**, taken high again to indicate that everything has been sent. | PB0    **/CA** | **-->** | GPIO5  PIN 29 | Server polls this line. When it goes low, that initiates a Zolados process. |
| **Client Ready**  When **idle**, the client wishes to initiate a read or write operation.  When **sending**, a byte has been placed on the data bus.  When **receiving**, the byte on the data bus has been received & processed. | PB1    **/CR** | **-->** | GPIO6  PIN 31 | After /CA has gone low, server monitors this line. When it goes low, server strobes /SR low to acknowledge. |
| **Data Direction**  Sets the direction of the data bus. Connects to pin 2 of the 74LVC4245.  HIGH = Z64 to Server  LOW = Server to Z64 | PB3    **DDIR** | **-->** | -- |  |
| **Client Online**  Normally pulled high, this is taken low by the Z64 to show that it is powered & connected. | PB2    **/CO** | **-->** | GPIO13  PIN 33 | Server constantly polls this line to check that client is active & available. |
|  | PB4 | **<--** | GPIO19  PIN 35    **/SR** | **Server Ready**  When **sending**, a byte has been placed on the data bus.  When **receiving**, the byte on the data bus has been received & processed. |
|  | PB5 | **<--** | GPIO16  PIN 36    **/SA** | **Server Active**  When **sending**, taken low to indicate the server is sending data. Goes high again when server has finished sending data. |
|  | PB6 | **<--** | GPIO26  PIN 37 |  |
|  | PB7 | **<--** | GPIO20  PIN 38 |  |

# LOAD/ SAVE PROCESS

**LOAD functions**

**cmdprcLOAD**

* Set FILE\_ADDR to USR\_START
* jsr **zd\_getfile**
  + jsr **read\_filename**
  + jsr **zd\_loadfile**
    - jsr **zd\_handshake**
      * jsr **zd\_init\_process**
        + jsr **zd\_signalDelay** (x2)
        + jsr **zd\_waitForSR** (starts, checks and stops timer)
      * jsr **zd\_send\_strbuf**
        + jsr **zd\_waitForSRoff** (starts, checks and stops timer)
        + jsr **zd\_signalDelay**
        + jsr **zd\_waitForSR**
      * jsr **zd\_svr\_resp**
        + jsr **zd\_waitForSA** (starts, checks and stops timer)
        + sr **zd\_waitForSR** (starts, checks and stops timer)
    - jsr **zd\_waitForSAoff** (starts, checks and stops timer)
    - jsr **zd\_rcv\_data**
      * jsr **zd\_waitForSA** (starts, checks and stops timer)
      * loop:
        + jsr **zd\_waitForSR** (starts, checks and stops timer)
        + jsr **zd\_waitForSRoff** (starts, checks and stops timer)
        + jsr **zd\_strobeDelay** (timerloop)
      * jsr **zd\_strobeDelay** (timer loop)
      * jsr **zd\_waitForSAoff** (starts, checks and stops timer)

**cmdprcXLOAD**

* set A to ZD\_OPCODE\_XLOAD
* jsr xload\_file
  + Read filename from STDIN\_BUF to STR\_BUF
  + Check currently selected RAM bank is available
  + Set FILE\_ADDR to EXTMEM\_START
  + **jsr OSZDLOAD** (zd\_loadfile)

**cmdprcXOPEN**

* set A to ZD\_OPCODE\_DLOAD
* jsr xload\_file
  + Read filename from STDIN\_BUF to STR\_BUF
  + Check currently selected RAM bank is available
  + Set FILE\_ADDR to EXTMEM\_START
  + **jsr OSZDLOAD** (zd\_loadfile)

**SAVE functions**

**cmdprocDUMP**

* Read hex address pair
* Read filename from STDIN\_BUF to STR\_BUF
* j**sr OSZDSAVE (zd\_save\_data)**

**cmdprcSAVE**

* Check executable code is currently loaded
* Read filename from STDIN\_BUF to STR\_BUF
* Set TEMP\_ADDR\_A to USR\_START
* Set TEMP\_ADDR\_B to LOMEM
* **jsr OSZDSAVE (zd\_save\_data)**

**Common functions**

**OSZDLOAD – funcs\_ZolaDOS.asm :: .zd\_loadfile**

* zd\_handshake
  + zd\_init\_process - send opcode, get response
  + zd\_send\_strbuf - send filename
  + zd\_svr\_resp
* zd\_rcv\_data – loads to address specified by FILE\_ADDR/+1

**OSZDSAVE – funcs\_ZolaDOS.asm :: .zd\_save\_data**

(Might replace first three steps with zd\_handshake)

* zd\_handshake
  + zd\_init\_process - send opcode, get response
  + zd\_send\_strbuf - send filename
  + zd\_svr\_resp
* zd\_send\_data

**zd\_send\_data**

# FUNCTIONS

| Z64 -> SERVER (Z64 -> SERVER) | |
| --- | --- |
| ZOLATRON | SERVER |
|  |  |
|  |  |
|  |  |
|  |  |

| INIT PROCESS (Z64 -> SERVER) | |
| --- | --- |
| ZOLATRON | SERVER |
|  | (IDLE) - /SA and /SR inactive, DATA port = input |
| (.zd\_init\_process) |  |
| Set DDR to OUTPUT |  |
| Sets /CA - LOW | Watching for /CA to be ACTIVE |
|  | Watch for /CR |
| Load opcode on DATA port |  |
| Sets /CR - LOW |  |
| Wait for /SR to be LOW |  |
|  | Read DATA port for code |
|  | Strobe /SR |
| Set /CR - HIGH |  |
| Set /CA - HIGH |  |

| HANDSHAKE (Z64 -> SERVER) | |
| --- | --- |
| ZOLATRON | SERVER |
| (.zd\_handshake) |  |
| INIT PROCESS | |
| SEND STRBUF | |
| Set DDR to INPUT |  |

| SEND STRBUF (Z64 -> SERVER) | |
| --- | --- |
| ZOLATRON | SERVER |
| (.zd\_send\_strbuf) |  |
|  | Wait for /CA |
| Set /CA - LOW |  |
|  |  |
|  | — GET TEXT loop |
|  | Wait for /CR - ACTIVE |
| Wait for /SR to be HIGH |  |
| Put byte on DATA port |  |
| Set /CR - LOW |  |
| Wait for /SR to be LOW |  |
|  | Read char from data port |
|  | Check if valid character |
|  | Strobe /SR |
| Set /CR - HIGH |  |

| FILE DATA  Data from server to Z64 | |
| --- | --- |
| ZOLATRON | SERVER |
| (.zd\_recv\_data) | Set /SR to INACTIVE |
| Wait for /SA to be ACTIVE |  |
|  | Get list of files |
|  | Set /SA to ACTIVE |
|  | Set DATA port to OUTPUT |
| — Loop for each file | |
| — Loop for each filename | |
|  | SENDBYTE (char) |
| — end filename loop | |
|  | SENDBYTE (0) |
| — end file loop | |
|  | SENDBYTE (DataEndCode = 255) |
|  | Set /SA to INACTIVE |

| SERVER SENDBYTE (SERVER -> Z64) | |
| --- | --- |
| ZOLATRON | SERVER |
| Wait for /SR to be ACTIVE |  |
|  | Set DATA port value |
|  | Strobe /SR |
|  | Wait for /CR - ACTIVE |
| Read byte |  |
| Set /CR - LOW |  |
| Set /CR - HIGH |  |
|  | Wait for /CR - INACTIVE |

# PROCESSES

After each process, set DATA port to INPUT.

## LOAD FILE

| ZOLATRON | SERVER |
| --- | --- |
| (.zd\_handshake) |  |
|  |  |
|  | Load file |
| SEND RESPONSE CODE | |
|  | Wait for /CA to be INACTIVE |
|  | Set DATA port to OUTPUT |
|  | Set /SA to ACTIVE |
|  | Wait for /CR |
|  | Set code on DATA port |
|  | Strobe /SR |
|  | Set /SA to INACTIVE |
|  | Delay - 100µs |
|  | Set /SA to ACTIVE |
|  |  |
|  | Loop while !EOF |
|  | Read next byte from file data |
| SEND BYTE | |
|  | Set DATA port value |
|  | Strobe /SR |
|  | Wait for /CR - ACTIVE |
|  | Wait for /CR - INACTIVE |
|  | Set /SA to INACTIVE |

## LS

| ZOLATRON | SERVER |
| --- | --- |
| Load opcode on DATA port |  |
| INIT PROCESS | |
| FILE DATA | |

**NOT SURE WHAT THIS IS**

|  |  |
| --- | --- |
| **Zolatron** | **RPi** |
| sets **CR** to **OFF** | set **SR** to **ON** |
| sets data direction to OUTPUT | waiting for **CA** to go **ON** |
| sets **CA** to **ON** |  |
| waits for server **SR** to go **OFF** |  |
| ***DATA LOOP*** |  |
| — puts byte on data bus | waits for **CR** to go **ON** |
| — sets **CR** to **ON** |  |
| — strobe delay | reads byte |
| — sets **CR** to **OFF** |  |
| — waits for **SR** to go **ON** |  |
|  | strobes **SR** **ON** then **OFF** |
| — waits for **SR** to go **OFF** | waits for **CR** to go **OFF erroring here** |
| — increments address pointer |  |
| — loops back | checks **CA** == **ON** otherwise ends loop |
| sets **CA** to **OFF** |  |

# File Headers

**File type codes**

These live at byte $03 in the header.

|  |  |  |  |
| --- | --- | --- | --- |
| **CODE** | **TYPE** | **HEX** | **NOTE** |
| D | Data | 44 | Pure data files. |
| E | Executable | 45 | Executable programs that run either at USR\_START or EXTMEM\_START. |
| O | Overlay | 4F | Code designed to be swapped in & out to complement a program in main memory. |
| X | OS extension | 58 | OS extensions designed to be run from the command line. |

**Data files**

These might be (eg) lookup tables.

|  |  |  |
| --- | --- | --- |
| **BYTE** | **CONTENTS** | **NOTE** |
| 0 | $FF |  |
| 1 | $00 or load address LSB | These bytes can be zero or the 16-bit address of the usual load location for the data. |
| 2 | $00 or load address MSB |
| 3 | D’, ‘E’, ‘O’ or ‘X’ | Type code |

**Executable files**

Programs designed to be run in main memory (at USR\_START) or extended memory (at EXTMEM\_START).

Most files – and especially executable files – should start with a header.

|  |  |  |
| --- | --- | --- |
| **BYTE** | **CONTENTS** | **NOTE** |
| 0 | $4C | JMP instruction - jmp .startprog |
| 1 | 16-bit address of  .startprog label | Address for previous JMP instruction. Machine will jump over this header info. |
| 2 |
| 3 | E’ | Type code |
| 4 | 16-bit address of  .header label | Also first byte of code, so will match either USR\_PAGE or EXTMEM\_LOC |
| 5 |
| 6 | 16-bit address of  .reset label | Usually the same as the .startprog label, but not necessarily. Where to jump to when resetting. |
| 7 |
| 8 | 16-bit address of  .endcode label | Address of first byte available after end of program |
| 9 |
| A | -- reserved -- | (For future expansion) |
| B | -- reserved -- | (For future expansion) |
| C | -- reserved -- | (For future expansion) |
| D | .prog\_name | Nul-terminated string containing short version of program name |
| varies | .version\_string | Nul-terminated version string |

**Overlays**

Not sure if/how these are going to work, so reserved for future expansion.

|  |  |  |
| --- | --- | --- |
| **BYTE** | **CONTENTS** | **NOTE** |
| 0 | $4C | Jump to ... something. |
| 1 | Address of something to jump to – probably some kind of table |  |
| 2 |
| 3 | O’ | Type code |

**OS Extensions**

-- WORK IN PROGRESS --

OS extension files are designed to contain additional OS features. They load into extended memory.

Each package will be limited to 8KB. The package should start with a header as usual. In this case, the header will include a command table that lists the commands provided by the package followed by the address of the related function.

.command\_table

equs "FOO",0

equw foo\_function

equs "BAR",0

equw bar\_function

equb EOTBL\_MKR

.foo\_function

...

rts

.bar\_function

...

rts

The OS can poll through the extended memory banks. In each one it:

* 1. Checks that the type code is 'X'. If not, go on to next bank.
* 2. Read the address of the command\_table from bytes 2 & 3 of the header.
* 3. Reads the next string in the command\_table and matches it against the entered command.
  + 4. If it matches, read the address bytes. Done.
  + 5. If it doesn't match:
    - 6. Skip two bytes (ie, skip over the address)
    - 7. Check to see if next byte is = EOTBL\_MKR.
      * 8. If it is, done with this bank – go to next one.
      * 9. If it isn't, loop back to step 3.

The code should also contain a routine to print a list of available commands. This code starts at .startprog.

|  |  |  |
| --- | --- | --- |
| **BYTE** | **CONTENTS** | **NOTE** |
| 0 | $4C | JMP instruction - jmp .startprog |
| 1 | 16-bit address of  .startprog label | Address for previous JMP instruction. Machine will jump over this header info. |
| 2 |
| 3 | X' | Type code |
| 4 | 16-bit address of  .command\_table label |  |
| 5 |
| 6 | -- reserved -- |  |
| 7 | -- reserved -- |  |
| 8 | -- reserved -- |  |
| 9 | -- reserved -- |  |
| A | .package\_name | Nul-terminated string containing short version of package name |
| varies | .version\_string | Nul-terminated version string |
|  |  |  |

# OPCODES & CONSTANTS

 The actual numerical value of these may change, so here we'll just list the constant names used.

These are defined in:

* 6502 code: **cfg\_ZolaDOS.asm**
* Go code: **zdlib/zdconsts.go**

**The names given are for the ZolaDOS (Go) code, but the 6502 constants are the same, but prefixed with ‘ZD\_’.**

|  |  |  |
| --- | --- | --- |
| **NAME** | **Code** | **Function** |
| OPCODE\_LOAD | 2 - 0x02  0000 0010 | LOAD - load executable .EXE file |
| OPCODE\_DLOAD | 3 - 0x03  0000 0011 | DATA LOAD - load data files - no ext added |
| OPCODE\_XLOAD | 4 - 0x04  0000 0100 | LOAD executable into extended memory - ROM images |
| OPCODE\_LS | 8 - 0x08  0000 1000 | List storage |
| OPCODE\_OPENR | 10 - 0x0A  0000 1010 | OPEN file for reading |
| OPCODE\_OPENW | 11 - 0x0B  0000 1011 | OPEN file for writing |
| OPCODE\_CLOSE | 12 - 0x0C  0000 1100 | CLOSE file |
| OPCODE\_RBLK | 30 - 0x1E  0001 1110 | Read block |
| OPCODE\_RBYTE | 31 - 0x1F  0001 1111 | Read byte |
| OPCODE\_RSTR | 32 - 0x20  0010 0000 | Read null-terminated string |
| OPCODE\_WBLK | 40 - 0x28  0010 1000 | Write block |
| OPCODE\_WBYTE | 41 - 0x29  0010 1001 | Write byte |
| OPCODE\_WSTR | 42 - 0x2A  0010 1010 | Write null-terminated string |
| FILE OPERATIONS | | |
| OPCODE\_DEL | 72 - 0x48  0111 0010 | Delete file |
| OPCODE\_MV | 96 - 0x60  1001 0110 | Move (rename) file |
| BINARY DUMPS: Following will cause ZolaDOS to append '.BIN' to the filename. | | |
| OPCODE\_DUMP\_CRT | 128 - 0x80  1000 0000 | Save - create (no overwrite) |
| OPCODE\_DUMP\_OVR | 129 - 0x81  1000 0001 | Save - create (overwrite) |
| OPCODE\_DUMP\_APP | 130 - 0x82  1000 0010 | Save - append |
| BINARY DATA: Following will cause no extension to be appended - command must use full filename | | |
| OPCODE\_SAVE\_DATC | 135 - 0x87  1000 0111 | Save - create (no overwrite) |
| OPCODE\_SAVE\_DATO | 136 - 0x88  1000 1000 | Save - create (overwrite) |
| OPCODE\_SAVE\_DATA | 137 - 0x89  1000 1001 | Save - append |
| EXECUTABLE FILES: Following save modes will cause ZolaDOS to append '.EXE' to the filename. | | |
| OPCODE\_SAVE\_CRT | 140 - 0x8A  1000 1010 | Save - create (no overwrite) |
| OPCODE\_SAVE\_OVR | 141 - 0x8B  1000 1011 | Save - create (overwrite) |
| OPCODE\_SAVE\_APP | 142 - 0x8C  1000 1100 | Save - append |
|  |  |  |
|  |  |  |

**ERROR CODES**

These are defined in:

* 6502 code: **cfg\_main.asm**
* Go code: **zdlib/zdconsts.go**

|  |  |
| --- | --- |
| **NAME** | **Value (dec)** |
| ERR\_FILE\_READ | 7 |
| ERR\_FILE\_LIST | 13 |
| ERR\_FILE\_EXISTS | 22 |
| ERR\_FILE\_OPEN | 23 |
| ERR\_FILE\_DEL | 24 |
| ERR\_FILENOTFOUND | 25 |
| ERR\_FILE\_BOUNDS | 26 |
|  |  |
|  |  |
|  |  |