ZolaDOS

Last edited: 7 Jun 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		+5V	4		
		5	GPIO3		GND	6		
D0	PA0	7	GPIO4		TX	8	RX on DUART PORTA	
		9	GND		RX	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.	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	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	PB2 **DDIR**	**-->**	--	
Client Online Normally pulled high, this is taken low by the Z64 to show that it is powered & connected.	PB3 **/CO**	**-->**	GPIO13 PIN 33	Server constantly polls this line to check that client is active & available.
	PB4	**<--**	GPIO19 PIN 35	**Server Ready** When **sending**, a byte has been placed on the data bus. When **receiving**, the byte on the data bus has been received &
	PB5	**<--**	GPIO16 PIN 36	**Server Active** When **sending**, taken low to indicate the server is sending data. Goes high again when server has finished sending data.
	PB6	**<--**	GPIO26	
	PB7	**<--**	GPIO20	

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
- jsr **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

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 2	16-bit address of `.startprog` label	Address for previous JMP instruction. Machine will jump over this header info.
3	E’	Type code
4 5	16-bit address of `.header` label	Also first byte of code, so will match either `USR_PAGE` or `EXTMEM_LOC`
6 7	16-bit address of `.reset` label	Usually the same as the `.startprog` label, but not necessarily. Where to jump to when resetting.
8 9	16-bit address of `.endcode` label	Address of first byte available after end of program
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 2	Address of something to jump to – probably some kind of table	
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`	
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	LOAD - load executable .EXE file
OPCODE_DLOAD	3 - 0x03 0000	DATA LOAD - load data files - no ext added
OPCODE_XLOAD	4 - 0x04 0000	LOAD executable into extended memory - ROM images
OPCODE_LS	8 - 0x08 0000	List storage
OPCODE_OPENR	10 - 0x0A 0000	OPEN file for reading
OPCODE_OPENW	11 - 0x0B 0000	OPEN file for writing
OPCODE_CLOSE	12 - 0x0C 0000	CLOSE file
OPCODE_RBLK	30 - 0x1E 0001	Read block
OPCODE_RBYTE	31 - 0x1F 0001	Read byte
OPCODE_RSTR	32 - 0x20 0010	Read null-terminated string
OPCODE_WBLK	40 - 0x28 0010	Write block
OPCODE_WBYTE	41 - 0x29 0010	Write byte
OPCODE_WSTR	42 - 0x2A 0010	Write null-terminated string

FILE OPERATIONS		
OPCODE_DEL	72 - 0x48 0111	Delete file
OPCODE_MV	96 - 0x60 1001	Move (rename) file
BINARY DUMPS: Following will cause ZolaDOS to append '.BIN' to the filename.		
OPCODE_DUMP_CRT	128 - 0x80 1000	Save - create (no overwrite)
OPCODE_DUMP_OVR	129 - 0x81 1000	Save - create (overwrite)
OPCODE_DUMP_APP	130 - 0x82 1000	Save - append
BINARY DATA: Following will cause no extension to be appended - command must use full filename		
OPCODE_SAVE_DATC	135 - 0x87 1000	Save - create (no overwrite)
OPCODE_SAVE_DATO	136 - 0x88 1000	Save - create (overwrite)
OPCODE_SAVE_DATA	137 - 0x89 1000	Save - append
EXECUTABLE FILES: Following save modes will cause ZolaDOS to append '.EXE' to the filename		
OPCODE_SAVE_CRT	140 - 0x8A 1000	Save - create (no overwrite)
OPCODE_SAVE_OVR	141 - 0x8B 1000	Save - create (overwrite)
OPCODE_SAVE_APP	142 - 0x8C 1000	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