DAN64: an AVR based 8-bit Microcomputer

Juan J. Martínez jjm@usebox.net

Manual for V1.R3 - May 20, 2016

Features

- Composite video black and white output, 256 x 192 resolution, 32 x 24 characters (8 x 8 pixels font, code page 437 character set).
- PS/2 keyboard support.
- 6502 virtual machine with system call interface to native code services.
- Linear 64KB memory access from the virtual machine (256 bytes page zero, 256 bytes hardware stack, 6144 bytes of video RAM and 58880 bytes for user programs).
- External storage support via audio in/out.
- Integrated 6502 assembler and disassembler.
- Basic shell supporting peek, poke, load, run, etc.

Project page and further information: DAN64 at usebox.net

1. Operating System

DAN64 boots into a shell that reports the operating system version (v1) and the amount of RAM available for user programs:

```
DAN64 v1, 58880 bytes free
Ready
```

The prompt line is identified by a blinking square cursor.

The command interface supports backspace and the left and right cursor arrows for editing input. The screen automatically scrolls when reaching the bottom of the screen.

1.1. Shell Commands

The shell commands are case insensitive. Addresses are expected in hexadecimal format (`xxxx`, example: 1a00).

Any input that doesn't match one of the supported commands will result in a `ERR: CMD` error.

1.1.1. help

The `help` command will print the list of currently supported commands:

```
LOAD, SAVE [addr [addr]], RUN
LIST [addr], AS [addr]
PEEK [addr], POKE [addr],
CLS, HELP
```

1.1.2. cls

The `cls` command will clear the screen positioning the cursor at the left top corner of the screen.

1.1.3. load

The `load` command is used to load programs into DAN64. The programs are loaded starting at the `0x1a00` address, just after video memory.

An external audio device able to play audio files (programs) is required. Connect a 3.5mm audio cable from the audio output of your media device into DAN64 audio in jack.

The operating system will ask for confirmation by pressing the enter key, or cancellation by pressing ESC key:

```
Press <ENTER> when ready,
<ESC> to cancel...
```

Once enter is pressed, the audio file should be played in the external device. The playback volume has to be set to the right levels for DAN64. Start with a 90% and adjust from there. Check the load error codes for help.

DAN64 screen will be switched off during the loading process.

When the program has been successfully loaded, the program size will be displayed:

```
402 bytes
Ready
```

Load Errors

ERR: TIME The loading process timed out. This may happen when the audio playback doesn't start soon enough after enter key has been pressed.

You may need to adjust the output volume of your media device.

ERR: IO 01 The magic number in the audio file was incorrect. This happens when the audio file is not a DAN64 program.

You may need to adjust the output volume of your media device.

ERR: IO [02, 03, 04, 05] You may need to adjust the output volume of your media device.

ERR: IO 06 The parity byte on the data block was incorrect. This may happen when the audio playback doesn't have enough quality.

1.1.4. save

The `save` command is used to save programs from DAN64 into external storage.

Two optional parameters are supported: *start address* and *end address*. By default start address is `1a00` and the end address is `ffff`.

For example, to save the first 512 bytes of user program:

```
save 1a00 1b00
```

An external audio device able to record audio files (programs) is required. Connect a 3.5mm audio cable from the audio out jack of DAN64 into the audio input of your media device.

The operating system will ask for confirmation by pressing the enter key, or cancellation by pressing ESC key:

```
Press <ENTER> when ready,
<ESC> to cancel...
```

Start recording in your media device and then press enter. When the program has been successfully saved, the program size will be displayed:

```
512 bytes
Ready
```

DAN64 screen will be switched off during the saving process.

1.1.5. run

The `run` command is used to run current program starting at `0x1a00` memory address.

Depending on the program, it may not exit back to the operating system, but if it does, the result can be:

- **Ok**: the program finished with no error (code 0).
- **ERR: PRG xx**: the program finished with error code `xx` (hexadecimal number).
- **ERR: SYS xx**: the program generated an unsupported SYS call `xx` (hexadecimal number).

The programs are not checked for validity before running.

1.1.6. peek

The `peek` command is used to display memory contents, in both its hexadecimal representation and in printable characters, 48 bytes at a time.

One optional *address* parameter is supported. By default `peek` will start from address `0000` and increment by 48 bytes the last address (unless a new address is provided).

Example:

```
peek 1b00
 1b00: 48 65 6c 6c 6f 20 Hello
 1b06: 77 6f 72 6c 64 21 world!
 1b0c: 00 de 10 44 ff ff ...D..
 1b12: ff ff 2c 21 ca 09 ..,!..
 1b18: cc 8b 0e 44 21 83 ...D!.
 1b1e: 04 08 52 13 a1 bf ..R...
 1b24: 2f 00 00 00 00 a0 /.....
 1b2a: 04 08 22 86 04 08 .."...
```

1.1.6. poke

The `poke` command is used to set memory contents, one byte at a time. Press enter without providing a value in order to cancel the operation.

One optional *address* parameter is supported. By default `poke` will start from address `0000` and increment by one byte last address (unless a new address is provided).

1.1.7. list

The `list` command is used to disassemble memory contents.

One optional *address* parameter is supported. By default `list` will start from address `1a00` and increment by 12 instructions the last address (unless a new address is provided).

Example:

```
list
 1a00:  a9ff     LDA #$ff
 1a02:  8500     STA $00
 1a04:  a9ff     LDA #$ff
 1a06:  8501     STA $01
 1a08:  20181a   JSR $1a18
 1a0b:  20241a   JSR $1a24
 1a0e:  48       PHA
 1a0f:  20541a   JSR $1a54
 1a12:  a900     LDA #$00
 1a14:  02       SYS
 1a15:  4c151a   JMP $1a15
 1a18:  a000     LDY #$00
```

See 1.2. DAN64 Integrated Assembler and Disassembler for further details.

1.1.8. as

The `as` command is used to assemble instructions into memory.

One optional *address* parameter is supported. By default `as` will start from address `1a00` and increment by 1 instructions the last address (unless a new address is provided).

See 1.2. DAN64 Integrated Assembler and Disassembler for further details.

1.2. DAN64 Integrated Assembler and Disassembler

DAN64 includes an integrated assembler and disassembler for the 6502 CPU.

The assembler is exposed through the `as` shell command (see: 1.1.8. as) and the disassembler is used by the `list` shell command (see: 1.1.7. list).

The memory layout of the 6502 CPU is as follows:

```
0x0000 - 0x00ff : zero page
0x0100 - 0x01ff : stack
0x0200 - 0x1aff : video memory
0x1a00 - 0xffff : user program
```

The assembler supports all 6502 instructions plus the custom **SYS** instruction used by the API (see: 2. DAN64 API).

Supported instructions are: ADC, AND, ASL, BCC, BCS, BEQ, BIT, BMI, BNE, BPL, BRK, BVC, BVS, CLC, CLD, CLI, CLV, CMP, CPX, CPY, DEC, DEX, DEY, EOR, INC, INX, INY, JMP, JSR, LDA, LDX, LDY, LSR, NOP, ORA, PHA, PHP, PLA, PLP, ROL, ROR, RTI, RTS, SBC, SEC, SED, SEI, STA, STX, STY, SYS, TAX, TAY, TSX, TXA, TXS and TYA.

See supported addressing modes for an overview of the supported addressing modes.

Mode	Format
Immediate	#nn
Absolute	nnnn
Zero page	nn
Implied	
Indirect absolute	(nnnn)
Absolute indexed X	nnnn,x
Absolute indexed Y	nnnn,y
Zero page indexed X	nn,x
Zero page indexed Y	nn,y
Indexed indirect	(nn,x)
Indirect indexed	(nn),y
Relative	nnnn
Accumulator	a

Table 1: Supported addressing modes

By default the numbers are expected to be decimal and hexadecimal numbers can be specified using the $ prefix. For example:

```
as 1a00
 1a00: lda #$ff
Ok
```

An special input mode is provided to make easier to enter arbitrary data by using the `.` (dot) command, followed by a quoted string, a decimal or hexadecimal number (using the `$` prefix) or the `,` (comma) as separator.

Example (string, end of line and end of string):

```
as 1b00
 1b00: ."Hello world", $0a, 0
Ok
```

Any problem parsing input will be reported with an `ERR: SYNTAX` error.

2. DAN64 API

The 6502 virtual machine implements an extra instruction **SYS** (opcode `0x02`) that allows user programs to access DAN64 services.

Current supported services are:

- 0x00: Terminate program
- 0x01: Load data
- 0x02: Save data
- 0x10: Put character
- 0x11: Put string
- 0x12: Set cursor position
- 0x13: Fill screen
- 0x14: Write (used by CC65 C compiler, supports stdout and stderr only)
- 0x20: Get character
- 0x21: Get input
- 0x22: Read (used by CC65 C compiler, supports stdin only)
- 0x30: Put tile
- 0xa0: Get random
- 0xa1: Wait for vsync
- 0xa2: Set random seed
- 0xf0: Get version

The service is specified in the accumulator and the parameters (if any) are pushed into the stack before running the `SYS` instruction (in reverse order).

Example (terminate program, exit code 0):

```
lda #$00
pha             ; exit code 0
sys             ; A already contains the service number (00: terminate program)
```

The `SYS` instruction is not equivalent to a function call, and only the accumulator and the `pc` registers will be altered. This means that the caller will have to remove the parameters from the stack after a `SYS` call.

Example (put a character restoring the stack):

```
tsx             ; save the stack pinter
lda #64         ; '@'
```

```
pha
lda #$10        ; put character
sys
txs             ; restore the stack
```

0x00: Terminate program

Terminates the running program returning to the operating system.

- Input: exit code (byte)
- Returns (in A): -

0x01: Load data

Loads data from external storage into program memory.

- Input: address to destination (word)
- Returns (in A): 0 on success

0x02: Save data

Saves data from program memory into external storage.

- Input: address of data (word), number of bytes to save (word)
- Returns (in A): 0 on success

0x10: Put character

Displays a character in current cursor location. The cursor location won't be changed and special characters are not interpreted.

- Input: character (byte)
- Returns (in A): 0 on success

0x11: Put string

Displays a zero terminated string in current cursor location. The cursor location will be changed and special characters will be interpreted (0x0a as end of line, 0x09 as 8 spaces).

- Input: address of the zero terminated string (word)
- Returns (in A): number of displayed characters

0x12: Set cursor position

Sets cursor position to a new location.

- Input: x position (byte, 0 to 21), y position (byte, 0 to 23)
- Returns (in A): 0 on success

0x13: Fill screen

Fill the screen using a specified character (use 32 -space- for clearing the screen). The cursor is set to the top left corner of the screen.

- Input: character (byte)
- Returns (in A): 0 on success

0x14: Write

This service is used by CC65 C compiler (supports stdout and stderr only).

- Input: file descriptor (word), buffer address (word), count (word)
- Returns (in A): bytes written, 0 on error

0x20: Get character

Read a character from keyboard.

- Input: -
- Returns (in A): character, 0 if the keyboard buffer is empty

0x21: Get input

Buffered read from the keyboard. The maximum size of the input is limited to 64 characters including the zero to end the string.

- Input: address to destination buffer (word), size of the buffer (byte)
- Returns (in A): number of characters read (including the zero end)

0x22: Read

This service is used by CC65 C compiler (supports stdin only).

- Input: file descriptor (word), buffer address (word), count (word)
- Returns (in A): bytes read, 0 on error

0x30: Put tile

Displays a tile in current cursor location. The cursor location won't be changed.

The tile definition is an array of 8 bytes (1 bit per pixel).

- Input: address to tile definition (word)
- Returns (in A): 0 on success

0xa0: Get random

Returns a 8-bit random number.

- Input: -
- Returns (in A): random number

0xa1: Wait for vsync

Will return when the video vertical sync starts.

- Input: -
- Returns (in A): -

0xa2: Set random seed

Sets a 16-bit random seed for the random number generator.

- Input: random seed (word)
- Returns (in A): 0 on success

0xf0: Get version

Gets the operating system version (x.y as (x | (y << 4))).

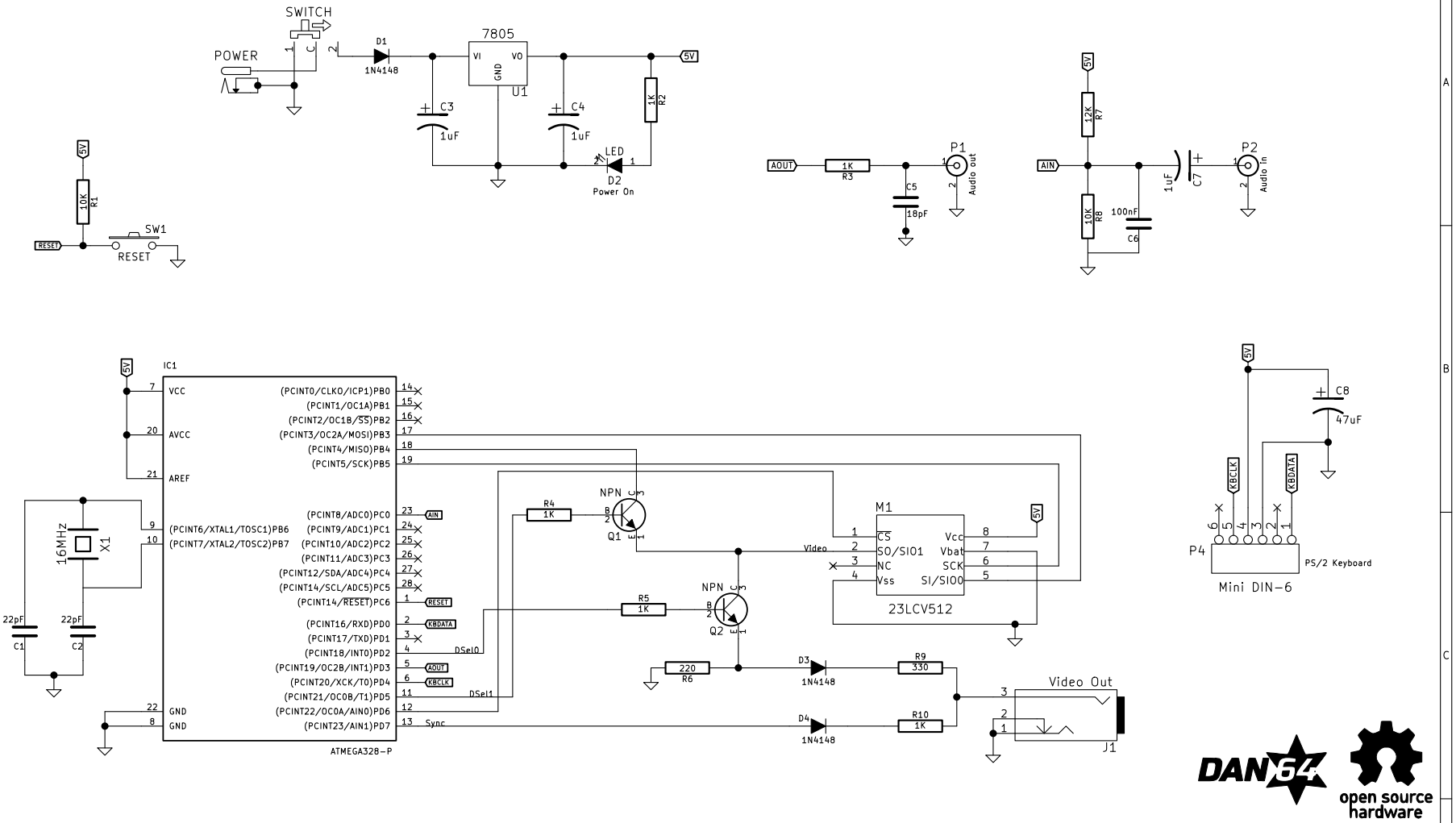
- Input: -
- Returns (in A): 1 (version 1.0)

Appendix A: Component List

Reference	Value
C1	22pF
C2	22pF
C3	1uF
C4	1uF
C5	18pF
C6	100nF
C7	1uF
C8	47uF
D1	1N4148
D2	LED
D3	1N4148

Reference	Value
D4	1N4148
IC1	ATMEGA328-P
J1	RCA Jack (CUI-RCJ-0140)
M1	23LCV512
P1	Stereo Audio Jack 3.5mm, SparkFun PRT-08032
P2	Stereo Audio Jack 3.5mm, SparkFun PRT-08032
P4	PS/2 Jack Mini DIN-6
POWER	DC Power Barrel, PRT-00119
Q1	BC548B NPN
Q2	BC548B NPN
R1	10KΩ
R2	1KΩ
R3	1KΩ
R4	1KΩ
R5	1KΩ
R6	220Ω
R7	12KΩ
R8	10KΩ
R9	330Ω
R10	1KΩ
S1	Mini Slide Switch
SW1	Momentary Push Button 2-pins
U1	LM7805
X1	16MHz Crystal Oscillator

Table 2: Component list



An AVR based 8-bit Microcomputer
Juan J. Martínez <jjm@usebox.net>

Sheet: /
File: dan64.sch

Title: DAN64

Size: A4	Date: 2016-05-20	**Rev: 3**
KiCad E.D.A. kicad 4.0.2+dfsg1-2bpo8+1-stable		Id: 1/1