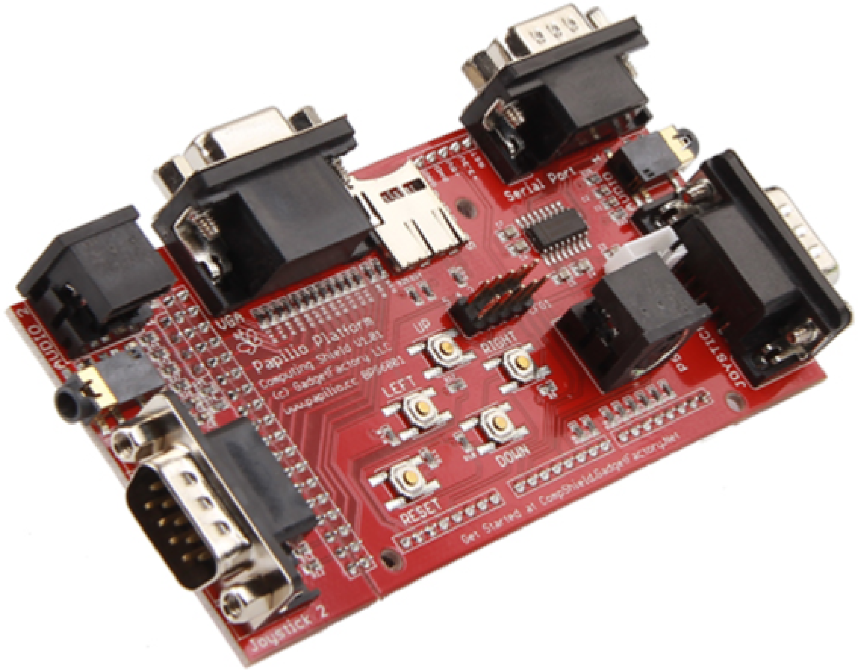
SwapForth

James Bowman `jamesb@excamera.com`

SVFIG August 22, 2015



These slides are running on a SwapForth system.

Xilinx FPGA on a Papilio Duo board

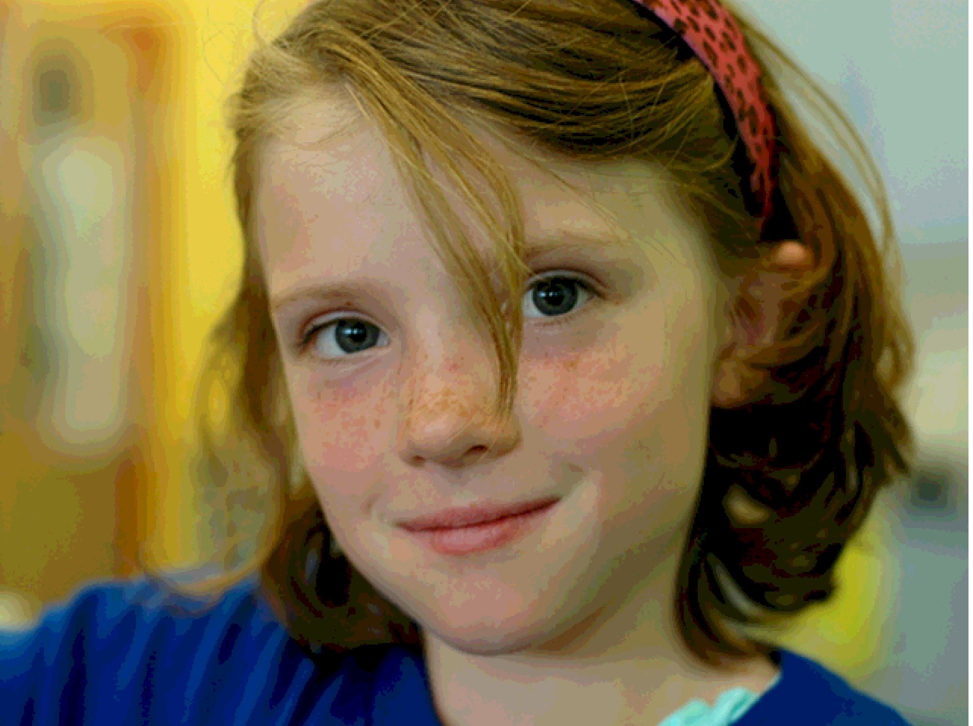
A 125MHz 32-bit J1b with 32Kbytes of RAM

SwapForth

Slideshow, which is a Forth application. It loads images from microSD into a VGA framebuffer.

Buttons work the slides, and have a PS/2 keypad hooked up.

That is it: completely standalone.



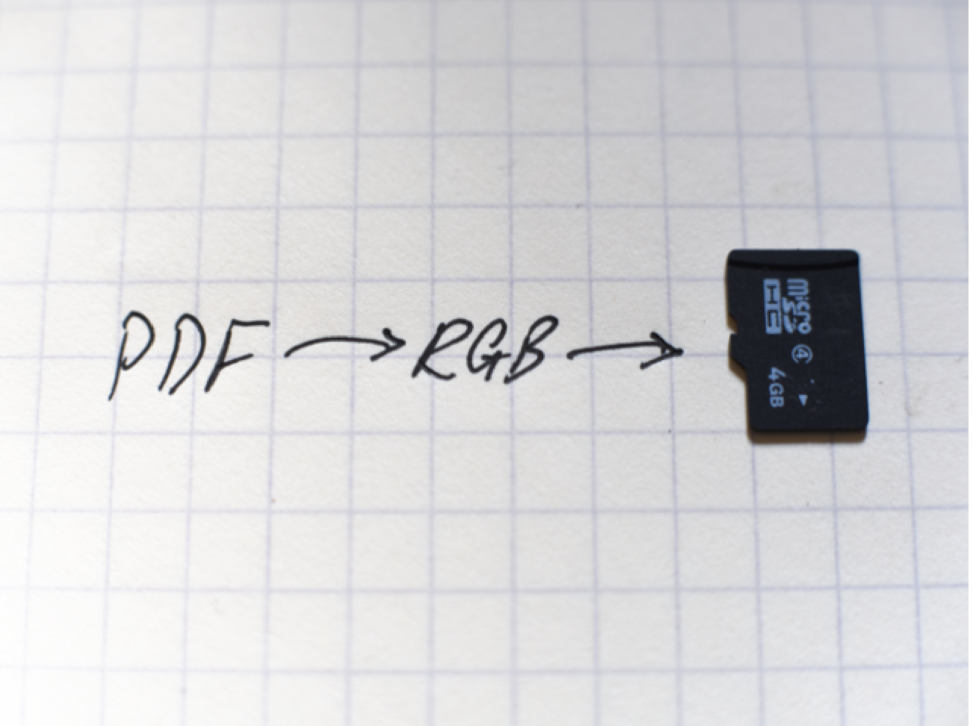
This board has an external SRAM; easy to hook it up as an RGB 640 $\times$ 480 24-bit framebuffer.

The hardware has only 4 bits for each of R, G and B. That is 16 distinct levels, so the banding is quite apparent.

`http://tinyvga.com/vga-timing/640x480@60Hz`

Just generate random numbers and add them to the 8-bit original pixels. You get noise in the image; looks like 'grain'. This is *dither* in its simplest form: overcoming a quantizing channel by adding uncorrelated noise.

http://www.excamera.com/sphinx/article-xorshift.html



Slides were made with beamer LaTeX, making a PDF.

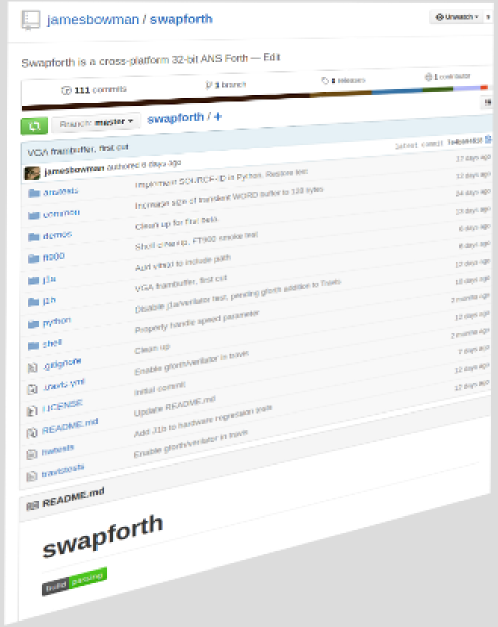
ImageMagick converts to raw RGB, then copy to a microSD

```
convert slides.pdf %02d.rgb
cat ??.rgb > /dev/sdc
```

SWAPFORTH

SwapForth is open source, BSD license

https://github.com/jamesbowman/swapforth



jamesbowman / swapforth

Unwatch

Swapforth is a cross-platform 32-bit ANS Forth — Edit

111 commits · 1 branch · 0 releases · 1 contributor

Branch: master · swapforth / +

VGA framebuffer, first cut

jamesbowman authored 8 days ago · latest commit 1a4b4482

anstests	Implement SOURCE-ID in Python. Restore test	12 days ago
common	Increase size of transient WORD buffer to 128 bytes	12 days ago
demos	Clean up for final beta	24 days ago
ft900	Shell cleanup. FT900 smoke test	13 days ago
j1a	Add vlnxil to include path	6 days ago
j1b	VGA framebuffer, first cut	8 days ago
python	Disable j1a/verilator test, pending gforth addition to Travis	12 days ago
shell	Properly handle speed parameter	18 days ago
.gitignore	Clean up	2 months ago
.travis.yml	Enable gforth/verilator in travis	12 days ago
LICENSE	Initial commit	2 months ago
README.md	Update README.md	7 days ago
hwtests	Add J1b to hardware regression tests	12 days ago
travistests	Enable gforth/verilator in travis	12 days ago

README.md

swapforth

build passing

SWAPFORTH

USEFUL!

portable: many Forths target one piece of hardware – SwapForth runs on multiple hosts. Choice is good!

It has **enough features** to be useful in real applications.

Heavy emphasis on **testing** for a solid system.

It is **efficient**, e.g. it compiles to native code: no interpreter.

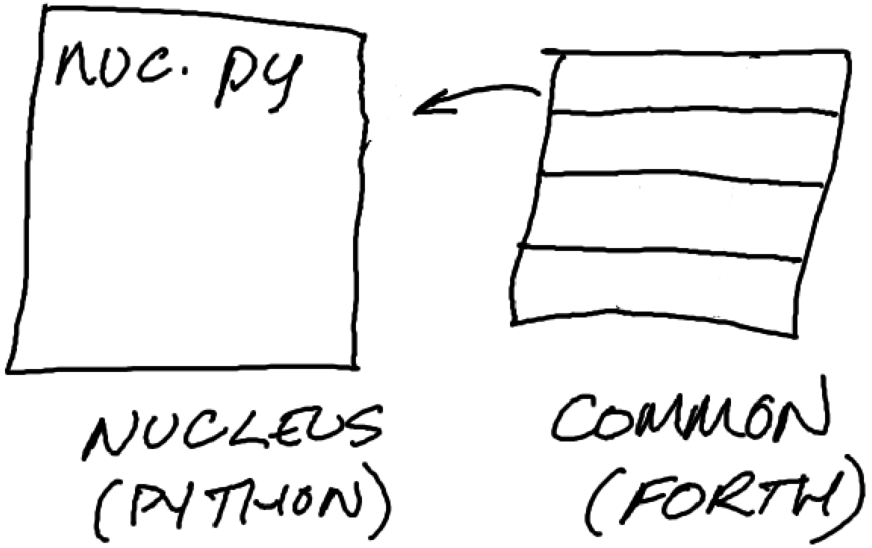
Lastly, it is a strictly standard ANS Forth: **no surprises**.

Targets

Might seem like a strange choice for a target. Forth running under Python. But Python is very widely supported, and is excellent *glue*.

16- 32- 64-bit performs OK for testing

You can select the type of target on the command line! Big- and little-endian also. Not a very efficient implementation, of course. Took a couple of hours to write. Many things are trivial: for example the stacks are lists, and the dictionary is... a dictionary.



`nuc.py` is the nucleus, written in native code, here Python. It implements just enough of Forth to be able to compile code from the console.

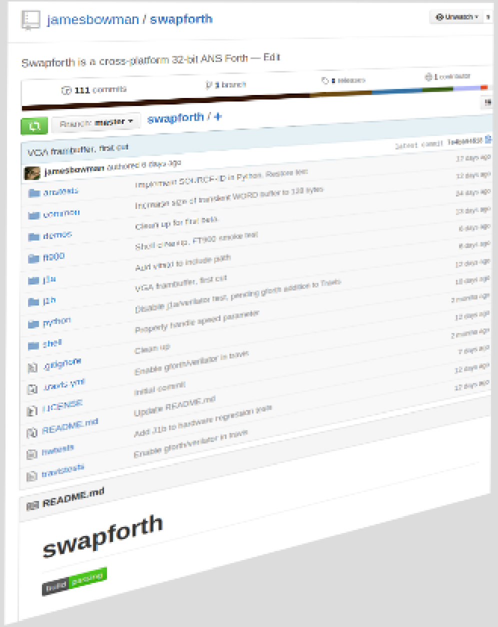
Then the rest of the system ("common") is fed to it through the console, which builds the complete Forth.

The Python build verifies that all the "common" code passes the ANS tests for 16-, 32- and 64-bit hosts.

Travis CI is a continuous integration system.

https://travis-ci.org/

They build your project for you, on an instance created just for that purpose. Is integrated with github, and triggered by every commit. Free for open-source projects.


jamesbowman / swapforth

Swapforth is a cross-platform 32-bit ANS Forth — Edit

111 commits · 1 branch · 0 releases · 1 contributor

Branch: master · swapforth / +

VGA frambuffer, first cut — latest commit 1a4b44f68

jamesbowman authored 8 days ago

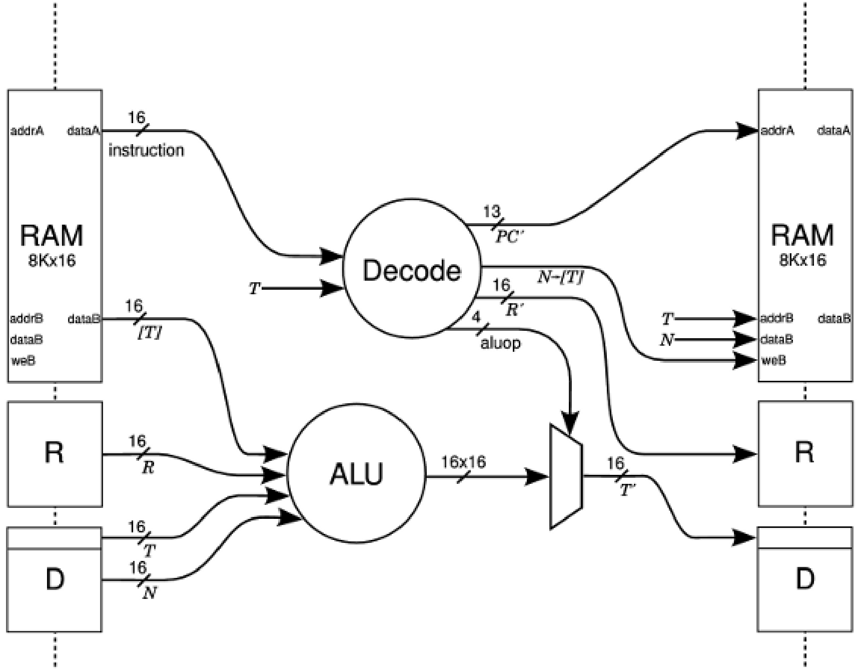
analysis	Implement SOURCE-ID in Python. Restore test	17 days ago
common	Increase size of transient WORD buffer to 128 bytes	12 days ago
demos	Clean up for final beta	24 days ago
ft900	Shell cleanup, FT900 smoke test	13 days ago
j1a	Add vlnpl to include path	6 days ago
j1b	VGA frambuffer, first cut	8 days ago
python	Disable j1a/verilator test, pending gforth addition to Travis	12 days ago
shell	Properly handle speed parameter	18 days ago
.gitignore	Clean up	2 months ago
.travis.yml	Enable gforth/verilator in travis	12 days ago
LICENSE	Initial commit	2 months ago
README.md	Update README.md	7 days ago
hwtests	Add J1b to hardware regression tests	12 days ago
travistests	Enable gforth/verilator in travis	12 days ago

README.md

swapforth

That little green button at the bottom means everything is passing the tests. The tests take about a minute to run. If something breaks, within a minute or so that badge turns red and I get an email.

J1a



The J1 is 5 years old now.

The J1a is a slightly stripped-down version. Multi-bit shift instructions are gone. It now works with simpler RAMs.

Stacks are bi-directional shift registers.

It is currently 124 lines of Verilog.

https://github.com/jamesbowman/swapforth/blob/master/j1a/verilog/j1.v

8K R/W RAM
500 LUT, 512 bits

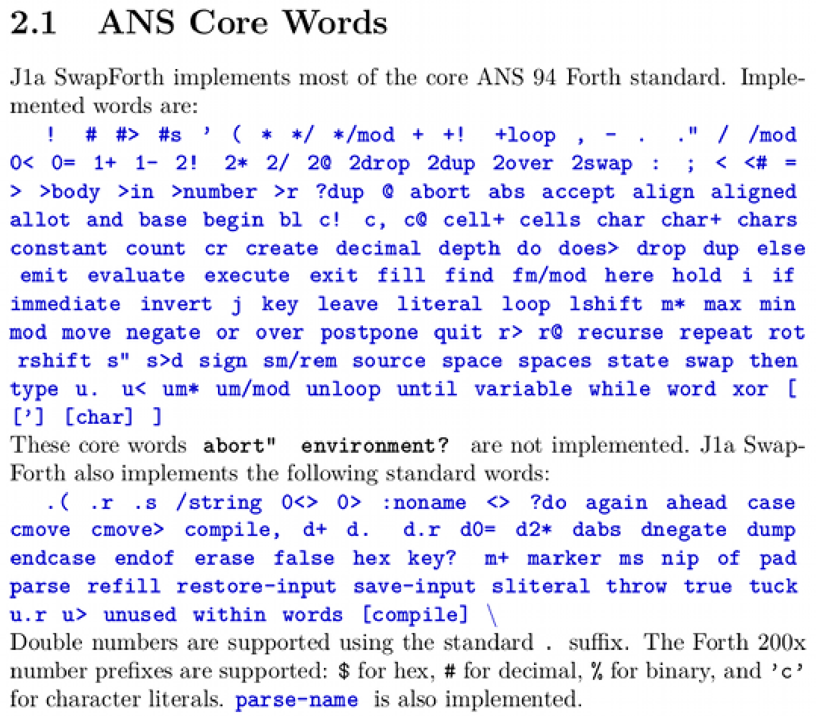
which is really, really small

R/W means one read port, one write port.

Almost all the gates are in the ALU, which is as it should be.

There are two stacks (D and R), each 16 deep by 16 bits wide.

Small enough to fit easily on almost any FPGA, and some CPLDs.



2.1 ANS Core Words

J1a SwapForth implements most of the core ANS 94 Forth standard. Implemented words are:

```
   ! # #> #s ' ( * */ */mod + +! +loop , - . ." / /mod
0< 0= 1+ 1- 2! 2* 2/ 2@ 2drop 2dup 2over 2swap : ; < <# =
> >body >in >number >r ?dup @ abort abs accept align aligned
allot and base begin bl c! c, c@ cell+ cells char char+ chars
constant count cr create decimal depth do does> drop dup else
 emit evaluate execute exit fill find fm/mod here hold i if
immediate invert j key leave literal loop lshift m* max min
mod move negate or over postpone quit r> r@ recurse repeat rot
 rshift s" s>d sign sm/rem source space spaces state swap then
type u. u< um* um/mod unloop until variable while word xor [
['] [char] ]
```

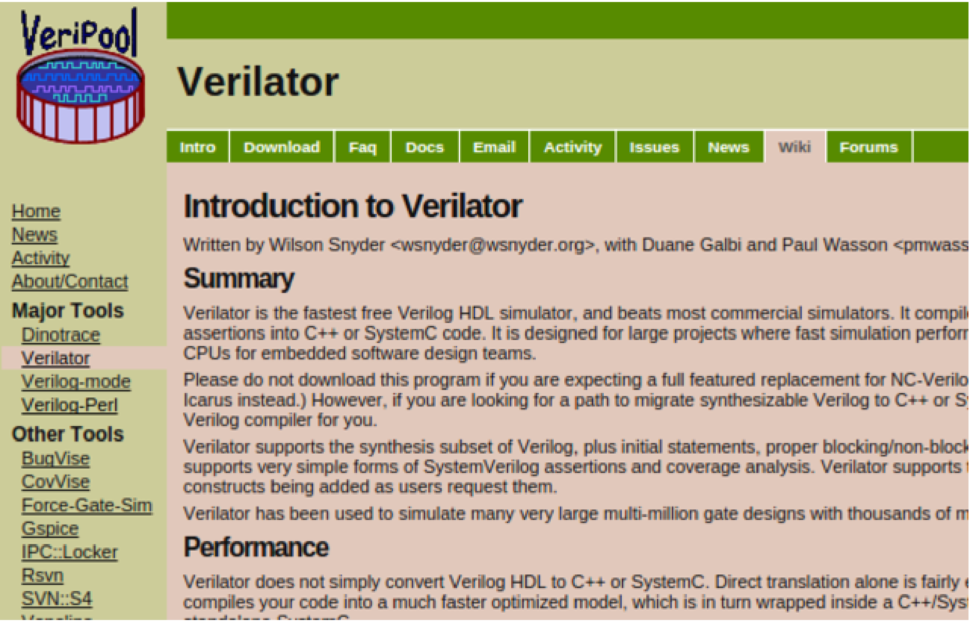
These core words `abort"` `environment?` are not implemented. J1a SwapForth also implements the following standard words:

```
   .( .r .s /string 0<> 0> :noname <> ?do again ahead case
cmove cmove> compile, d+ d. d.r d0= d2* dabs dnegate dump
endcase endof erase false hex key? m+ marker ms nip of pad
parse refill restore-input save-input sliteral throw true tuck
u.r u> unused within words [compile] \
```

Double numbers are supported using the standard `.` suffix. The Forth 200x number prefixes are supported: `$` for hex, `#` for decimal, `%` for binary, and `'c'` for character literals. `parse-name` is also implemented.

The complete Forth is currently 4.7K. This includes all of the ANS CORE words. The compiler produces native J1a code, but to save space does not do much optimization.

Everything works just as you would expect - no surprises!



Verilator

Introduction to Verilator

Written by Wilson Snyder <wsnyder@wsnyder.org>, with Duane Galbi and Paul Wasson <pmwass

Summary

Verilator is the fastest free Verilog HDL simulator, and beats most commercial simulators. It compil
assertions into C++ or SystemC code. It is designed for large projects where fast simulation perforr
CPUs for embedded software design teams.

Please do not download this program if you are expecting a full featured replacement for NC-Verilo
Icarus instead.) However, if you are looking for a path to migrate synthesizable Verilog to C++ or S
Verilog compiler for you.

Verilator supports the synthesis subset of Verilog, plus initial statements, proper blocking/non-block
supports very simple forms of SystemVerilog assertions and coverage analysis. Verilator supports
constructs being added as users request them.

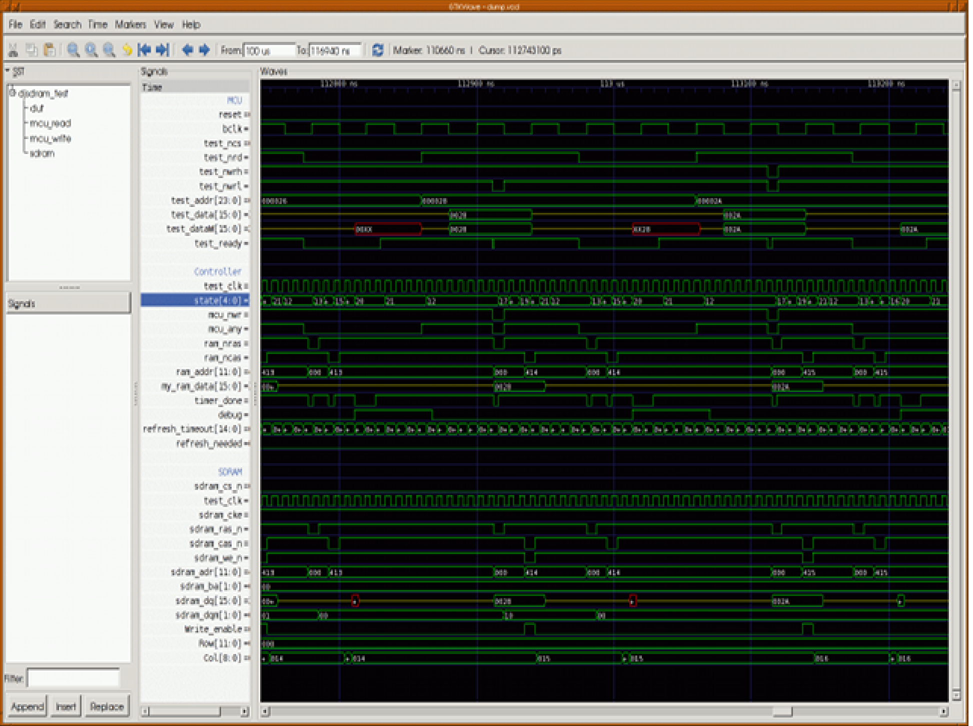
Verilator has been used to simulate many very large multi-million gate designs with thousands of m

Performance

Verilator does not simply convert Verilog HDL to C++ or SystemC. Direct translation alone is fairly
compiles your code into a much faster optimized model, which is in turn wrapped inside a C++/Sys

OK, sounds great... how do you run it?

The J1a CPU is written in Verilog, so one way is to run it on a Verilog simulator. Verilator is an open source Verilog compiler that produces C++, so you get a linkable library.

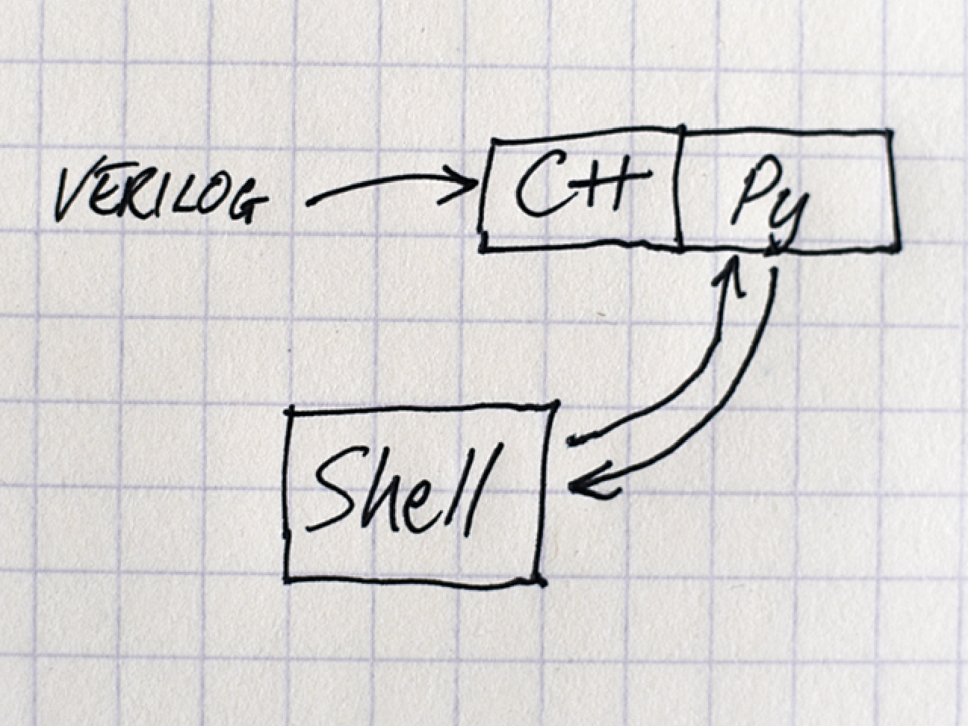


When hardware people talk about running RTL simulations they're thinking of waves like this.

And they *are* useful for debugging low-level problems.

But past a certain point the way to test out a computer is to use it.

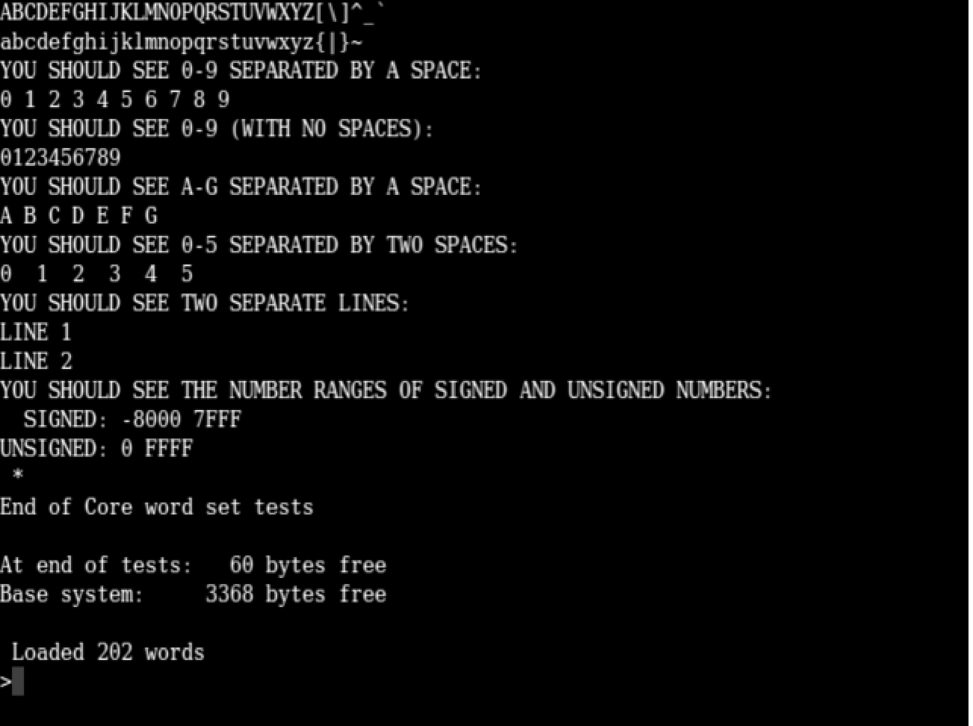
http://gtkwave.sourceforge.net/



Verilator compiles the verilog to C++.

A small piece of C/Python glue makes this look quite like a serial interface.

Then the Python shell can talk to it, just as if it were a physical UART.

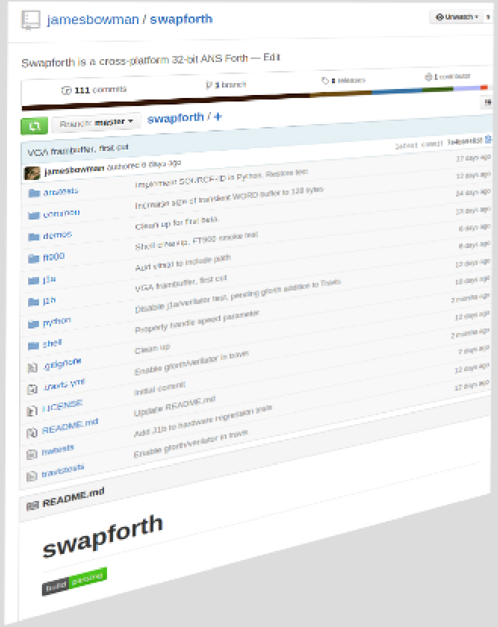


```
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`
abcdefghijklmnopqrstuvwxyz{|}~
YOU SHOULD SEE 0-9 SEPARATED BY A SPACE:
0 1 2 3 4 5 6 7 8 9
YOU SHOULD SEE 0-9 (WITH NO SPACES):
0123456789
YOU SHOULD SEE A-G SEPARATED BY A SPACE:
A B C D E F G
YOU SHOULD SEE 0-5 SEPARATED BY TWO SPACES:
0  1  2  3  4  5
YOU SHOULD SEE TWO SEPARATE LINES:
LINE 1
LINE 2
YOU SHOULD SEE THE NUMBER RANGES OF SIGNED AND UNSIGNED NUMBERS:
  SIGNED: -8000 7FFF
UNSIGNED: 0 FFFF
 *
End of Core word set tests

At end of tests:   60 bytes free
Base system:     3368 bytes free

 Loaded 202 words
>
```

Here is the J1a simulator running the standard ANS CORE tests. It takes a few seconds.

Execution speed is quite good; on a laptop it runs at about 30MHz.

Can run a simulation of the HDL at interactive speeds.


jamesbowman / swapforth

Swapforth is a cross-platform 32-bit ANS Forth — Edit

111 commits · 1 branch · 0 releases · 1 contributor

Branch: master · swapforth / +

VGA framebuffer, first cut · latest commit 1e4b4483

jamesbowman authored 8 days ago

analysis	Implement SOURCE-ID in Python. Restore test	17 days ago
common	Increase size of transient WORD buffer to 128 bytes	12 days ago
demos	Clean up for first beta	24 days ago
ft900	Shell cleanup. FT900 smoke test	13 days ago
j1a	Add vinput to include path	6 days ago
j1b	VGA framebuffer, first cut	8 days ago
python	Disable j1a/verilator test, pending gforth addition to Travis	12 days ago
shell	Properly handle speed parameter	18 days ago
.gitignore	Clean up	2 months ago
.travis.yml	Enable gforth/verilator in travis	12 days ago
LICENSE	Initial commit	2 months ago
README.md	Update README.md	7 days ago
hwtests	Add J1b to hardware regression tests	12 days ago
travistests	Enable gforth/verilator in travis	12 days ago

README.md

swapforth

build passing

This also is included in the continuous integration test. So every checkin:

builds the nucleus from `nuc.fs`

builds the CPU from the Verilog

executes the nucleus on it

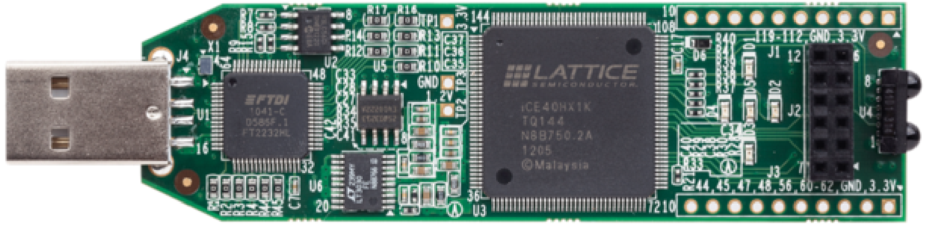
which builds the rest of SwapForth from source

which then runs the ANS CORE test

...

Takes about 3 seconds.

https://travis-ci.org/jamesbowman/swapforth/jobs/75825710



It runs on the Lattice iCEstick very well. Built with the IceStorm tools.

What use is it, apart from computing the future date of Easter?

http://excamera.com/sphinx/article-j1a-swapforth.html
http://www.clifford.at/icestorm/

This is the LabJack

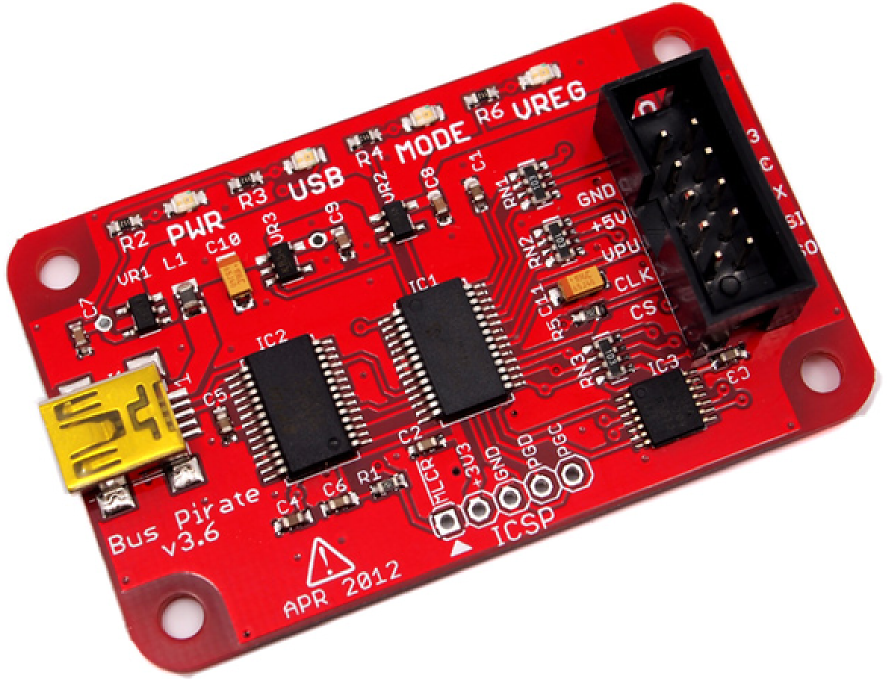
USB meets GPIOs. Has PC-side library for controlling from scripts.

https://labjack.com/

This is the FTDI MPSSE cable

USB meets GPIOs. Some protocols built-in. Has PC-side library for controlling from scripts.

http://www.ftdichip.com/Products/Cables/USBMPSSE.htm



Bus Pirate. USB meets GPIOs. A PIC runs the GPIOs, also I2C SPI, JTAG etc.

Interactive; there is a CLI running on the UART.

Protocols are fixed: need to update flash for each new protocol.

Which is how I bricked mine.

http://dangerousprototypes.com/docs/Bus_Pirate

FTDI USB
24 GPIO
programmable!

It costs $22 in singles, cheapest!

USB up to 460 Kbps

Interactive GPIO probing - the command-language is built-in!

And for protocols, just send the code as a preamble. Code is in RAM, so the next reboot puts it back to 'factory' state.

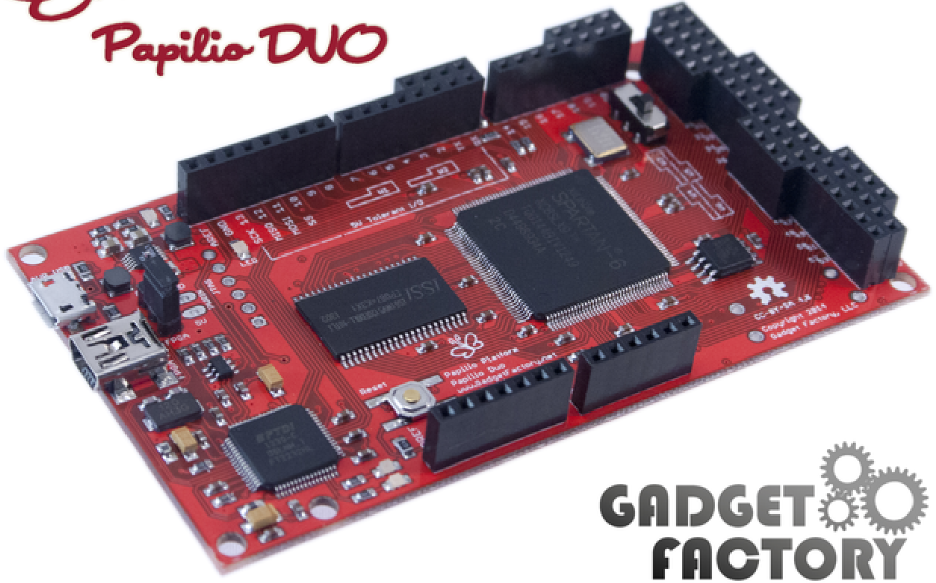
J1a aims to be the smallest useful interactive Forth machine.

J1b is larger. It is a 32-bit version, more RAM, fuller Forth.

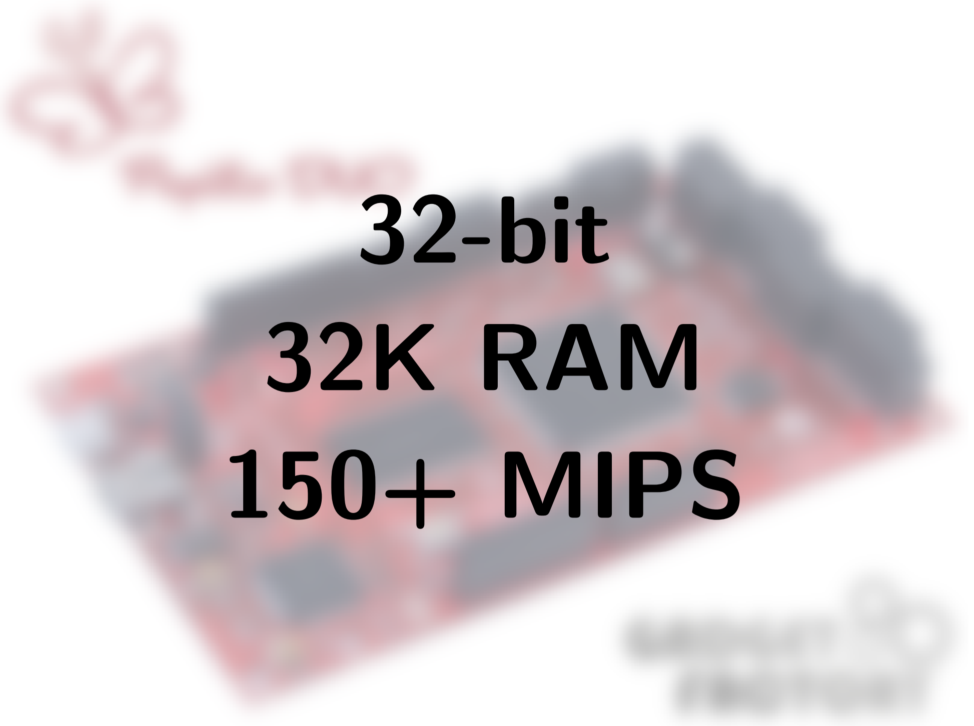
Like J1a, it builds and runs under Verilator and on an FPGA.

J1b

Papilio DUO



GADGET FACTORY


32-bit

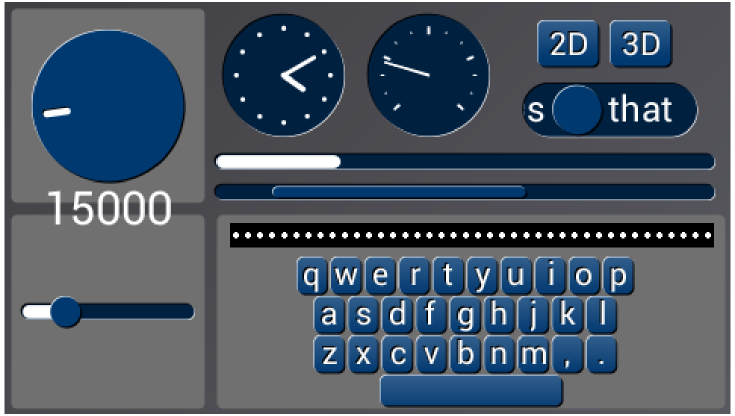
32K RAM

150+ MIPS

J1b is *still* a minimalist CPU, e.g.:

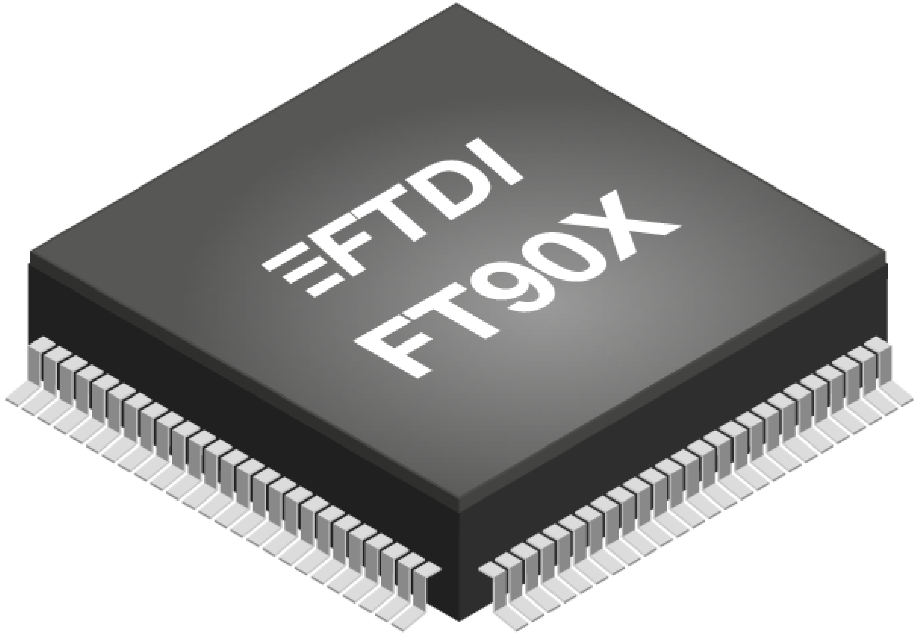
- doesn't even have subtract
- only word memory access, so `C@` and `C!` are subroutines

Compare with ZPUino which is optimized for C, runs at 100MHz.
http://www.alvie.com/zpuino/



This is a full 32-bit Forth, so it can run ANS Forth code. For example, the FT800 driver I showed here at SVFIG last time runs very well, using bit-banging GPIO.

https://github.com/jamesbowman/forth-ft800


FTDI
FT90X

	J1a	J1b	FT900
word size	16	32	32
memory	8K	32K	320K
fib2 (ms)	631	47	36
fib2 (M clocks)	7.5	5.8	3.6

Comparison of three current targets.

The benchmark comparison is of course dependent on clock speed,

`fib2` is at:
https://atariwiki.strotmann.de/wiki/Wiki.jsp?page=Forth%20Benchmarks

BYE