J1a
SwapForth
Reference

James Bowman

`jamesb@excamera.com`

Excamera Labs

Pescadero

California USA

ANS Forth Compliance Label

J1a SwapForth is an ANS Forth System

Providing names from the **Core Extensions** word set
Providing names from the **Double-Number** word set
Providing names from the **Facility** word set
Providing names from the **Facility Extensions** word set
Providing names from the **String** word set
Providing names from the **Programming-Tools** word set
Providing names from the **Programming-Tools Extensions** word set

Contents

Chapter 1

Getting started

J1a SwapForth is a 16-bit version of SwapForth, intended as an interactive Forth system using very little logic and RAM. The system currently fits on a Lattice HX20 FPGA. The J1a plus peripherals take 1100 logic elements. SwapForth uses 5Kbytes of RAM, leaving about 3K for the application.

After installing the   icestorm tools, you can run on a Lattice iCEstick like this

```
git clone git@github.com:jamesbowman/swapforth.git
cd swapforth/j1a
iceprog icestorm/j1a.bin
python shell.py -h /dev/ttyUSB0
```

(where `/dev/ttyUSB0` is the appropriate port your iCEstick was assigned). You should see something like

```
Contacting... established
Loaded 196 words
>
```

And you can now try the usual Forth things, e.g.

```
1 2 + .
3   ok
```

There is a fairly complete core ANS-compatible Forth system running on the board, including a compiler.

1.1   Some demos

You can control the five on-board LEDs

```
-1 leds
  ok

0 leds
  ok
```

and to make them blink

```
: blink
  32 0 do
    i leds
    100 ms
  loop
;
blink
```

There is an Easter date calculator

```
new
#include ../demos/easter.fs
```

Now you can do

```
>2015 .easter
2015 April 5   ok
```

Or even

```
>: 20easters
+  2035 2015 do
+    cr i .easter
+  loop
+;
 ok
>20easters

2015 April 5
2016 March 27
2017 April 16
2018 April 1
2019 April 21
2020 April 12
2021 April 4
2022 April 17
2023 April 9
2024 March 31
2025 April 20
2026 April 5
2027 March 28
```

```
    2028 April 16
    2029 April 1
    2030 April 21
    2031 April 13
    2032 March 28
    2033 April 17
    2034 April 9    ok
```

1.2   Building from Scratch

After installing the icestorm tools, run

```
rm build/*
make -C icestorm
```

This will produce `j1a.bin` - but it only contains the very bare-bones system; the rest of SwapForth still needs to be compiled. To do this, load `j1a.bin` and start the shell:

```
$ iceprog icestorm/j1a.bin
$ python shell.py -h /dev/ttyUSB0 -p ../common/
Contacting... established
Loaded 127 words
>
```

Then compile the rest of SwapForth and write the finished executable with these commands:

```
#include swapforth.fs
#flash build/nuc.hex
#bye
```

Now run `make -C icestorm` again - this compiles an FPGA image with the complete code base built-in.

Chapter 2

Available Words

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 : ; < <# = > >in
>number >r ?dup @ abort abs accept align aligned allot and base
begin bl c! c, c@ cell+ cells char 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 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 are not implemented:

```
>body abort" char+ chars environment? leave
```

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.

2.2   Additional Words

The following words are not standard. Some are traditional Forth words,
others are specific to the J1a SwapForth implementation.

`.x`

`  ( n -- )`

display n as a 4-digit hex number

`-rot`

`  ( x1 x2 x3 -- x3 x1 x2 )`

rotate the top three stack entries

`bounds`

`  ( start cnt -- start+cnt start )`

prepare to loop on a range

`code@`

`  ( addr -- u )`

fetch from code memory

`cp`

`  ( -- a )`

variable: code memory current pointer

`dp`

`  ( -- a )`

variable: data memory current pointer

`forth`

```
( -- a )
```
variable: most recent dictionary entry

---

io!
```
( x a -- )
```
store x to IO port a

---

io@
```
( a -- x )
```
fetch from IO port a

---

leds
```
( x -- )
```
write x to the onboard LEDs

---

new
```
( -- )
```
restore code and data pointers to the power-up state

---

s,
```
( a u -- )
```
add the u-character string a to the data space

---

serialize
```
( -- )
```
display all of current memory in base 36

---

tth

```
( -- a )
```
variable: tethered mode

Chapter 3

The SwapForth Shell

3.1 Command reference

3.2 Tethered Mode

J1a SwapForth supports *tethered mode*, which makes the UART protocol easier to use for host programs. The SwapForth shell uses tethered mode. To enter tethered mode, write one to the variable `tth` :

```
1 tth !
```

In tethered mode, `accept` transmits byte value 30 (hex `1e`, ASCII code RS). This allows the listening program to know that the target machine is ready to accept a line of input. In addition, `accept` does not echo characters as they are typed.

Chapter 4

Memory

4.1 RAM Types

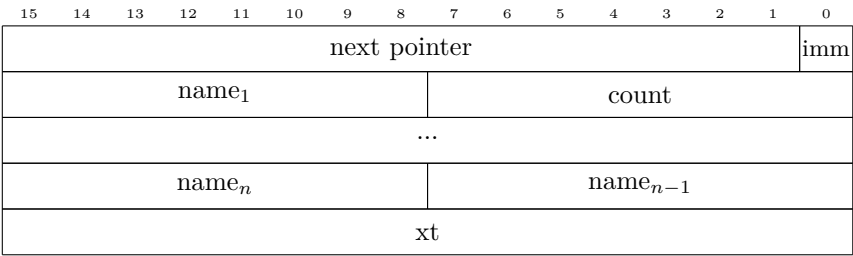The J1a implementation uses 8Kbytes of RAM in a split configuration.

The lower 4K is for code. This RAM is writable, and executable, but not (directly) readable. The variable CP (code pointer) points into this area. To read from this region, use the special word `code@` .

The upper 4K is for data. This RAM is writable and readable. The dictionary and all variables are located in this section. The variable DP points into this area.

4.2 Dictionary Layout

The SwapForth dictionary is a linked list; the variable `forth` holds the start of this list. Each dictionary entry contains:

- **next pointer** - address of the next dictionary entry, or zero for the last dictionary entry

- **imm** - immediate bit

- **count** - length of the name, in characters, 1-31

- **name$_1$ - name$_n$** - characters in name. If the length of the name is even, then a padding byte is appended

- **xt** - execution token for the word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
next pointer															imm
$name_1$								count							
...															
$name_n$								$name_{n-1}$							
xt															

Chapter 5

iCEstick Hardware interface

The J1a for iCEstick includes connections to the iCEstick peripherals:

- SPI flash

- LEDs

- IrDA tranceiver

- Pmod connector

- prototyping connectors

- UART

Access to peripherals is via the `io@` and `io!` words. Peripherals are port-mapped into a 16-bit IO address space.

Most ports are either read-only or write-only. For read-only ports, writing to the port has no effect. For write-only ports, reading from the port gives zero.

As an example of direct port access, this word blinks the on-board LEDs when a signal on IrDA is detected.

```
: x
  begin
    $2000 io@     \ read from input port
    8 and 0=      \ true if bit 3 (IrDA RXD) is 0
    $0004 io!     \ write to LEDS
  again
;
```

5.1   Port Map

5.1.1 $0001: Pmod data

Not yet implemented.

5.1.2 $0002: Pmod direction

Not yet implemented.

5.1.3 $0008: PIO output

Write-only port $0008 controls the flash and IrDA outputs.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
											IrDA SD	IrDA TXD	flash SCK	flash MOSI	flash CS

5.1.4 $0004: LEDs

The five on-board LEDS are controlled by write-only port at address $0004. Setting a bit to 1 lights the corresponding LED.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
											LED5	LED4	LED3	LED2	LED1

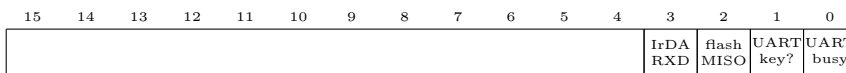Built-in word `leds` writes to this port.

5.1.5 $1000: UART data

The read-write port at address $1000 is for UART transmission or reception. Writing to the port starts transmission of a byte, reading the port returns the incoming byte.

Standard words `key` , `key?` and `emit` can be used to access this port.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								byte							
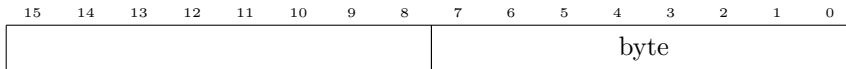
5.1.6 $2000: IrDA, flash and UART inputs

Read-only port $2000 contains the input signals from the IrDA receiver, SPI flash, and UART.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
												IrDA RXD	flash MISO	UART key?	UART busy

Index