J1a
SwapForth
Reference

James Bowman
jamesb@excamera.com
Excamera Labs
Pescadero
California USA

ANS Forth Compliance Label

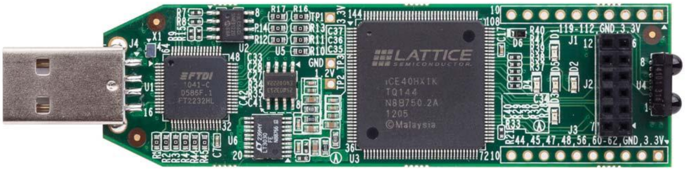
J1a SwapForth is an ANS Forth System

Providing names from the **Core Extensions** word set

Contents

Chapter 1

Getting started



Connect to the SwapForth board using a terminal program of your choice. Set the serial parameters to:

- 115200 baud
- 8 data bits, no parity, no stop bit (often called “8N1”, and often the default)

```
----------------------------------------------------------
swapForth v0.1
```

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:

```
   ahead  dump  .s  /string  parse-name  sliteral  throw  words
```

2.2 Additional Words

The following words are not part of the standard, and are specific to J1a SwapForth. 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 `u`-character the 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 Notes on Tethered Mode

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

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="15">next pointer</td><td>imm</td></tr>
<tr><td colspan="8">name_1</td><td colspan="8">count</td></tr>
<tr><td colspan="16">...</td></tr>
<tr><td colspan="8">name_n</td><td colspan="8">name_{n-1}</td></tr>
<tr><td colspan="16">xt</td></tr>
</table>

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

5.1.5 $1000: UART data

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