The N.I.G.E. Machine

Reference Manual

Andrew Read (andrew81244@outlook.com)

Document last updated: August 2015

CONTENTS

1. INSTALLATION AND SET-UP

1.1 INTRODUCTION

The N.I.G.E. Machine is a user-expandable micro-computer system that runs on an FPGA development board. It has been designed specifically for the rapid prototyping of experimental scientific hardware or other devices. The key components of the system include a stack-based softcore CPU optimized for embedded control, a FORTH software environment, and a flexible digital logic layer that interfaces the micro-computer components with the external environment.

The N.I.G.E Machine is presently available for both the Digilent Nexys 2 (1200K gate) and Nexys 4 FPGA boards.

Further information on the N.I.G.E. Machine design is available in two papers presented at EuroFORTH 2012 and 2013:

- http://www.complang.tuwien.ac.at/anton/euroforth/ef12/papers/
- http://www.complang.tuwien.ac.at/anton/euroforth/ef13/papers/

Two short video introductions are also available:

- http://www.youtube.com/watch?v=0v-HuVLRoUc
- http://www.youtube.com/watch?v=0Kj5EMdnkMk

1.2 SET-UP PRELIMINARIES

- Install Xilinx ISE version 14.6
  - The N.I.G.E. Machine is being developed on ISE 14.6, so using this version will avoid compatibility issues
- Install a suitable GIT repository manager, e.g.:
  - http://www.syntevo.com/smartgithg
- Clone the open-source N.I.G.E. Machine repository from GitHub to your local machine
  - Remote repository location:
    - https://github.com/Anding/N.I.G.E.-Machine
  - To preserve absolute file references used by ISE, the local directory for the repository must be exactly as follows:
    - E:\N.I.G.E.-Machine
- Within the local repository, switch to the appropriate branch for your Nexys board:
  - Nexys 2 (1200K gate) v2.0
  - Nexys 4 v4.0 (default branch)

1.3 QUICK START

- Attach a VGA monitor, keyboard (PS/2 - Nexys 2 or USB - Nexys 4), and 5V power supply

- Set the FPGA board jumper wires according to the Nexys reference manuals
- Configure the FPGA board with the pre-compiled N.I.G.E. Machine bit file
    - Nexys 2: E:\N.I.G.E.-Machine\board_Nexys 2_1200_v2.0.bit
    - Nexys 4: E:\N.I.G.E.-Machine\board_Nexys 4_v4.0.bit
- The N.I.G.E. Machine is now running as a FORTH microcomputer
    - See chapter 2 for further guidance

1.4 FULL START

- Unzip the file Xilinx_ISE.zip to the local repository folder
    - E:\N.I.G.E.-Machine
- The Xilinx project files should now be found in
    - E:\N.I.G.E.-Machine\Xilinx_ISE
    - Watch out for inadvertent folder duplication ("\Xilinx_ISE\Xilinx_ISE") caused by the unzip process or absolute project file references used by ISE will be invalid
- Double click on the ISE project file
    - E:\N.I.G.E.-Machine\Xilinx_ISE\NIGE_Machine.xise
- The N.I.G.E. Machine design files are now open in Xilinx ISE
- Synthesize the design files
- Configure the Nexys board with the newly compiled N.I.G.E. Machine bit file
    - E:\N.I.G.E.-Machine\Xilinx_ISE\[xxx].bit

1.5 OPTIONAL SD CARD INTERFACE

- Nexys 2: utilize a full-size SD card and a Digilent SD card PMOD device plugged into port A
    - http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,401,513&Prod=PMOD-SD
- Nexys 4: utilize a micro-SD card and the micro-SD slot on the Nexys board
- Format the SD card as FAT32
    - Both normal (<2GB) and high capacity (>2GB) SD cards are acceptable
    - The card must be formatted as FAT32 irrespective of capacity
- Copy all of the files from the following folder onto the SD card
    - E:\N.I.G.E.-Machine\Software
- Insert the SD card into the slot on the PMOD (Nexys 2) or the directly into board (Nexys 4)
- See chapter 2 for further guidance on using the SD card as a file system

2. USING THE N.I.G.E. MACHINE AS A FORTH MICROCOMPUTER

2.1 ANSI FORTH

At power-on the N.I.G.E. Machine is a self-contained microcomputer with FORTH system software. As far as possible the system software has been designed to be compliant with ANSI FORTH. Some minor ANSI deviations were allowed given the constraints of an embedded system. See appendix 6 for a list of the ANSI FORTH words that are available on the N.I.G.E. Machine and appendix 7 for a list of N.I.G.E. Machine specific control words.

2.2. FILE SYSTEM

2.2.1 SD card usage and file system navigation

The N.I.G.E. Machine uses SD cards for file system external storage. See Chapter 1 for SD card set-up details. At power-on the only available file access word is INCLUDE. Prepare and insert an SD card, then issue the following commands to make available additional ANSI FORTH words from the file access and other word sets.

```
MOUNT              \ Mount an SD card and initialize FAT32 data
INCLUDE SYSTEM.F   \ SYSTEM.F extends the FORTH system software
```

The N.I.G,E. Machine reads and writes filenames in 8+3 format only (e.g. “FILENAME.EXT”), but directory structures are supported. The file path directory separator character is either forward slash “/” or back slash “\”, and these may be used interchangeably. A leading slash or a double slash (“//” or “\\”) within a file path are interpreted as go-up-one-directory-level. No special character is used to specify the root folder.

Examples:

```
S” PROJECT/TEST.F”  \ specify the file TEST.F in the PROJECT directory beneath
                    \ the current directory

S” \B\DATA.TXT”     \ go up one level from the current directory and down
                    \ again into folder B to specify the file DATA.TXT
```

2.2.2 New-line character

The system software recognizes LF as the line terminator disregards CR characters. The WRITE-LINE word terminates lines with CRLF. The N.I.G.E. Machine can therefore read and process both Windows and LINUX format files, but files are written in Windows format.

2.3 MEMORY ADDRESS REGIONS

The N.I.G.E. Machine address space is spanned in bytes. It comprises three separate memory regions:

1. System memory is available for storage of the FORTH dictionary and other data. This is referred to as SRAM in the N.I.G.E. Machine documentation. System memory is comprised of FPGA BLOCK RAM and is the only memory from which the CPU can fetch and execute code.

2. Memory mapped hardware registers that control the operation of the N.I.G.E. Machine or connect to external pins on the FPGA. These registers are implemented in FPGA fabric logic and route directly into the N.I.G.E. Machine's own functional modules.

3. The external pseudo-static dynamic RAM chip included on the Nexys boards. This memory is referred to as PSDRAM. The N.I.G.E. Machine CPU can fetch and store data from PSDRAM, but cannot execute code there. PSDRAM is also used to hold the VGA screen display buffer. Access to PSDRAM is arbitrated by a direct memory access controller module within the N.I.G.E. Machine.

In terms of the choice of RAM for data storage, SRAM has the advantage that it is fast (1 clock cycle access) and deterministic. However total capacity is limited to tens of kilobytes. PSDRAM is more plentiful (16 megabytes), but access is slower and subject to arbitration with the VGA display.

The CPU has separate instructions for byte, word and long-word memory access and the corresponding FORTH words C!, C@, W!, W@, !, @ are available in system software.

Access to SRAM does not need to be aligned. Both words and long-words can successfully be read/written to odd address boundaries without penalty. PSDRAM access on the other hand must be aligned: words can only be accessed on even address boundaries and long-words accessed on address boundaries divisible by four.

The N.I.G.E. Machine is big-endian format. Memory access for words and long-words is such that the highest value byte is placed in the lowest memory address.

Appendices 3 and 4 provide complete memory maps for the N.I.G.E. Machine v2.0 and v4.0

2.4 VGA DISPLAY

The N.I.G.E. Machine outputs standard VGA signals through the VGA D-sub connector on the Nexys boards.

This section of the user manual describes the display technical characteristics, while appendix 7 documents a set of N.I.G.E. Machine specific FORTH display command words, and appendices 2 and 3 document the hardware registers that may be accessed directly for controlling the display.

2.4.1 Display organization

The N.I.G.E. Machine display is character graphics based. There are 256 different character codes and each character is 8 pixels high by 8 pixels wide. The first 128 characters are drawn from the ANSI character set and there are 128 additional custom characters. All of these characters are soft-

programmable by writing to the 2KB CHARACTER RAM that resides within the SRAM address space.

Each character position on screen comprises a single word in memory. The top left character on screen corresponds to the first address of the screen buffer in memory. Within each word, the high byte (stored at the lower memory address) contains the color information while the low byte (stored at the higher memory address) references the character code.

2.4.2 Interlace mode

The N.I.G.E. Machine VGA display is able to interlace two additional scan lines between each row of character graphics to assist the readability of text. This is known as interlace mode and soft-selectable on or off.

```
2 INTERLACE  \ Sets interlace mode with two scanlines between each row

0 INTERLACE  \ Sets non-interlaced mode
```

2.4.3 Display resolution

The N.I.G.E. Machine display output is soft-selectable between the following VGA modes:

VGA mode (binary)	VGA pixel resolution	Character resolution (non-interlace mode)	Character resolution (interlace mode)
000	VGA display off	n/a	n/a
001	640*480	80*60	80*48
010	800*600	100*75	100*60
011*	1024*768	128*96	128*77
100*	1920*1080	240*135	240*108

*VGA resolutions only available on the N.I.G.E. machine v4.0

```
2 VGA        \ Sets 800*600 VGA mode
```

2.4.4 Screen buffer

The display screen buffer is held in PSDRAM memory. The location of the start of the screen buffer is soft-programmable and can be redirected by writing to the SCREENPLACE hardware register. (Or equivalently, via the SCREENPLACE FORTH variable). The default location of the start of the screen buffer can be read from the SCREENBASE FORTH constant.

In normal input and output the FORTH system software automatically moves the start location of the display buffer through a fixed range of memory to achieve faster screen refreshes when

scrolling the display. The screen buffer location is returned to its default location when the screen is cleared through a CLS command, or when repeated screen scrolls have brought the screen buffer to the end of its allocated range.

User application may allocate their own block of PSDRAM for display purposes and relocate the screen buffer there for the duration of their execution. When the location of the screen buffer is restored to its prior value the display will revert accordingly.

2.4.5 Color modes

The N.I.G.E. Machine has two color modes. These define the way that the color byte for each character is interpreted.

- In 16/16 color mode the highest 4 bits of the color byte define the background color and the lowest 4 bits define the foreground color of that character.

```
0 COLORMODE        \ Set 16/16 color mode
```

- In 256/0 color mode all 8 bits of the color byte define the foreground color of that character. In this mode the background color for the whole screen is determined by the value of the BACKGROUND hardware register. 256/0 color mode is the power-on default.

```
1 COLORMODE        \ Set 256/0 color mode
```

Interpretation of the foreground and background colors differs between the N.I.G.E. Machine v2.0 and v4.0.

2.4.6. v2.0 color output

The Nexys 2 board has a color output range of 8 bits. In 256/0 color mode an 8 bit color code is directly interpreted as an RGB value (RRRGGGBB format). The BACKGROUND hardware register is 8 bits wide.

In 16/16 color mode, the 4 bits are also directly interpreted as an RGB value that is hard-wired extrapolated to 8 bits.

2.4.7. v4.0 color output

The Nexys 4 board has a color output range of 12 bits (4096 colors). In 256/0 color mode, the foreground color value (0-255) indexes a word in 16 bit PALETTE RAM from which the actual 12 bit RGB color code is read (RRRRGGGGBBBB format). PALETTE RAM is pre-defined with a range of colors (see Appendix 5) but is also soft-programmable. The BACKGROUND hardware register is 16 bits wide (of which the lowest 12 bits are used).

In 16/16 color mode, the foreground color value (0-15) indexes the first 16 words in PALETTE RAM as above but the 16 background colors are pre-defined in hardware.

2.5 OTHER INPUT/OUTPUT

In the default configuration the N.I.G.E. Machine includes a single RS232 port.

In the N.I.G.E. Machine v2.0 the RS232 port is connected directly to the Nexys 2 board's RS232 D-sub connector. However the Nexys 4 board does not include a D-sub connector. For the N.I.G.E. Machine v4.0 a PMOD RS232 expansion should be connected to PMOD socket C, lower pin row. Alternatively it is possible to re-route the RS232 port to the Nexys 4 board's RS232/USB interface. Comment/uncomment the relevant NET definitions in the FPGA constraints file to make this change:

1. E:\N.I.G.E.-Machine\Software\Board_Nexys 4.ucf

RS232 communication over an RS232/USB adapter may be less reliable due to buffer/latency issues in the adapter or PC driver. Appendix 7 documents N.I.G.E. Machine specific RS232 input/output words.

The UART adapter is hardwired to 8 bits, 1 stop bit, no parity, no handshaking. The baud rate is user configurable. Default settings are as follows:

v2.0, Nexys 2: 9,600 baud

v4.0, Nexys 4: 57,600 baud

2.6 EXCEPTION HANDLING

The N.I.G.E. Machine incorporates CATCH and THROW as single machine language instructions. QUIT operates using a master CATCH statement that calls the main user interaction system loop. If an error is thrown that is not trapped by user code, then it will be caught by the CATCH statement within QUIT.

QUIT assumes that any error codes it receives are counted strings, which it will display as an error message. A straightforward way for user code to abort execution and return an error message is as follows

```
c" My error message" THROW       \ Abort execution and return to QUIT
```

2.7 MULTITASKING

The N.I.G.E. Machine has hardware multitasking capabilities built into the soft-core CPU (see Appendix 12). The FORTH system software includes various support words for launching and controlling tasks which are described in Appendix 7.

The hardware multitasking core has the flexibility to operate whichever model for taking sequencing is best suited for the application at hand. The default model provided by the FORTH system software is cooperative, round-robin sequencing.

3. CUSTOMIZING THE SYSTEM SOFTWARE

The system software is written in assembly language. A two-pass cross-assembler is available to prepare the necessary binary files from the assembly language source.

The CPU instruction set is documented in Appendix 2. Because the N.I.G.E. Machine CPU is designed as a FORTH processor, the instruction set is essentially a sub-set of the most primitive FORTH words. The cross assembler provides additional macros corresponding to higher-level FORTH word, mostly for flow control. Appendix 9 documents the cross assembler macros and directives.

3.1 RUNNING THE CROSS-ASSEMBLER

The cross assembler should be run in an ANSI FORTH environment such as a PC running VFX FORTH. The command to run the cross assembler is ASMX. The N.I.G.E. Machine system software source file is E:\N.I.G.E.-Machine\System\FORTH.ASM.

```
INCLUDE ASMX.F   \ include the cross assembler

S" E:\N.I.G.E.-Machine\System\FORTH.ASM" ASMX
                 \ cross-assemble the system software
```

The three output files are as follows, and **they will always be placed in the folder E:\N.I.G.E.-Machine\System** to maintain integrity of the absolute file references used by Xilinx ISE.

- SRAM.BIN
  - Binary file suitable for transfer to the N.I.G.E. Machine by boot-loader
- SRAM.TXT
  - Text file that is used during simulation with the Xilinx ISE simulator
- SRAM.COM
  - Text file that is used when Xilinx ISE generates the SRAM memory module. Note that the SRAM module must be explicitly regenerated in ISE each time the system software source is reassembled

All of the output files are placed in this folder "E:\N.I.G.E.-Machine\System", so it is important that the exact directory structure of the source file depository is maintained.

3.2 STRUCTURE OF THE SYSTEM SOFTWARE

The system software comprises three regions:

- Interrupt vector table
- Interrupt handlers
- FORTH dictionary

3.2.1 Interrupt vector table

Valid interrupts redirect program counter execution to the appropriate location in the interrupt vector table. Each table is two bytes long and should be constructed as follows.

1. A two byte branch instruction (BRA) to an interrupt handler, for an active interrupt
2. An RTI NOP (return from interrupt, no operation) combination, for an inactive interrupt

The interrupt vector table is documented in appendix 11.

3.2.1 Interrupt handlers

Interrupt handler should be placed within the range of a branch instruction (i.e. in the first 8Kb of system memory). All interrupt handlers must be terminated with an RTI instruction rather than an RTS instruction, as this signals the interrupt controller than new interrupts may be processed.

See chapter 4 for more details on the operation of the interrupt controller and interrupt priority.

3.2.2 FORTH dictionary

The structure of the FORTH system is documented in appendix 10. Note that if the dictionary is extended then the dictionary word LAST (which is also the last definition in the source file FORTH.ASM) must be updated accordingly.

3.2.3 Exception handling

The N.I.G.E. Machine incorporates CATCH and THROW as single machine language instructions. The N.I.G.E. Machine implementing of the FORTH words CATCH and THROW operate as usual as per their ANSI definitions. However there is a slight usage difference for the machine language instruction CATCH as follows: In an assembly language program the machine language instruction CATCH must be followed by the machine language instruction ZERO in all cases.

This is to ensure the correct behavior when a word that is called by CATCH returns via RTS rather than THROW. In the case of return via RTS the return address is the instruction immediately following CATCH, and so the execution thread will proceed to the instruction ZERO, thus showing the correct ANSI FORTH behavior. In the case of return via THROW the return address is the instruction immediately following CATCH plus one, so the execution thread will step over the instruction ZERO but retain the THROW code on the top of stack.

For example:

```
#.l          subroutine          \ load the subroutine address on stack
CATCH                            \ call the subroutine
ZERO                             \ RTS returns here
...                              \ n THROW (n != 0) returns here
```

3.3 UPDATING THE SYSTEM SOFTWARE

The most straightforward way to test the system software is to transfer the completed binary machine code file to the FPGA board on a temporary basis using the boot loader. However the file transferred by boot loader will be lost when the board is powered off or the FPGA is

reprogrammed. To incorporate the revised system software on a permanent basis the FPGA configuration .bit file needs to be regenerated with the revised machine code file.

In either case the first step is to re-assemble the updated system software and confirm that a new binary file has been created:

- E:\N.I.G.E.-Machine\System\SRAM.bin

Note that this **exact filepath and filename** must be used to ensure compatibility with the absolute file references used by Xilinx ISE.

3.3.1 Using the boat loader to update the system software until power off

Nexys 2, v2.0

The Digilent Adept application should be used to transfer the file SRAM.bin to via the Epp interface into register 0xFF. The system will automatically reset when the transfer is started and reboot with the new system software.

Nexys 4, v4.0

The Digilent Adept application is not compatible with the Nexys 4 board and so the boot loader utilizes transfer via the RS232 port.

- Establish an RS232 connection between the PC and the Nexys 4 board. (See section 2.5)
- Press the CPU reset button on the Nexys 4 board. The board will reset for 4 seconds
- Immediately transfer the file SRAM.bin to the Nexys 4 board via the RS232 interface at the UART default settings of 57,600 baud, 8 bits, 1 stop bit, no parity, no handshaking. A blue LED on the Nexys 4 board will light during transfer. Ensure that the binary file is transferred without modification (e.g. no CR/LF substitutions)
- At the end of the 4 second reset reboot will occur with the new system software. This will be retained in memory until either the board is switched off or the FPGA is reprogrammed.

3.3.2 Updating the FPGA configuration file for a permanent system software update

The procedure for permanently updating the FPGA configuration bit file is as follows:

- Open the SRAM core in Xilinx ISE
    - inst_SYS_RAM
- Generate the core and the regenerate the programming file in ISE. **Note that the filepath to the Xilinx memory core initialization module must be exactly as follows:**
    - E:\N.I.G.E.-Machine\System\SRAM.coe
- Transfer the FPGA configuration .bit file to the Nexys board in the usual manner

4. CUSTOMIZING THE SYSTEM HARDWARE

The N.I.G.E. Machine is designed at the outset to be extended for specific applications through customized hardware that control or interact with external apparatus. The overall scheme for doing this is as follows:

- Customized hardware is developed as VHDL (or Verilog) modules within the N.I.G.E. Machine design project
- The custom modules interface with the outside world either through the electronic components available on the Nexys board (e.g. LED's, microphone, etc.) or through circuits attached to the PMOD expansion ports. Digilent supply a range of ready-made PMOD circuits that may be suitable for a variety of purposes
    - http://www.digilentinc.com/Products/Catalog.cfm?NavPath=2,401&Cat=9
- The custom modules interface with the N.I.G.E. Machine through either or both of two channels:
    - Additional user-defined hardware registers (that extend the list documented in appendices 3 and 4). These hardware registers may be readable, writable or both. The CPU access them through normal memory fetch and store instructions, while the bit level signals are routed directly into the design of the custom hardware module
    - Additional user-defined system interrupts (that extend the list documented in appendix 11). After creating additional system interrupts appropriate entries will need to be created for them in the interrupt vector table and appropriate interrupt handlers will need to be developed. This will involve customizing the system software in assembly language (see chapter 3)
- Extending the hardware registers and interrupts involves making changes to the following modules in the project design files. In general this should be as straightforward as duplicating the existing logic for each new item using the design file comments as a guide:
    - Inst_HW_Registers
    - Inst_Interrupt
- The FORTH system software is used to debug, test, and develop the necessary control applications for the customized hardware. Because FORTH can be used as an interpreted language, and because the N.I.G.E. Machine includes native keyboard and video display interfaces, the benefits of both rapid prototyping and fast execution speed are available to enhance software development compared with a traditional embedded development using a remote editor and compiler

In addition the N.I.G.E. Machine design is released under a dual license with open source rights for non-commercial use (please see the cover page of this manual for further details.) The design of the system itself can be remade.

This manual cannot fully describe the process of custom hardware development in VHDL or Verilog, but some further suggestions in respect of the N.I.G.E. Machine are made as follows:

- Testing design changes in the electronic simulator (Xilinx ISM) is essential before testing in hardware. For this purpose a faster-to-simulate SRAM module (RAM_for_Testbench) is provided. Comment out the instance of SYS_RAM in Board_Nexys4 and comment in RAM_for_Testbench. The principal advantage is that this module can read a revised SRAM.bin file directly without being regenerated, thus saving time
- Regression testing of CPU functionality is also essential after modifications are made. Several regression test programs are provided in the folder E:\N.I.G.E.-Machine\System to test the CPU instructions set, memory access and other functions. They can be run in both the electronic simulator and in hardware. Each test in the sequence is announced via the seven-segment display. If a test fails then its sequence number will remain on display. If the complete sequence of tests completes successfully then the value 0xFF is displayed
- It is critical to ensure that the new design meets timing. ISE SmartXplorer is very helpful for obtaining the best place and route for a given design and optimizing timing

APPENDIX 1. SYSTEM SPECIFICATIONS

Version	**v2.0**	**v4.0**
Circuit board	Digilent Nexys 2 (1200K gate)	Digilent Nexys 4
FPGA family	Xilinx Spartan-3E	Xilinx Artix-7
CPU data format	32 bits	32 bits
CPU pipeline	3 stage	3 stage
CPU throughput	1 instruction per clock-cycle for most instructions	1 instruction per clock-cycle for most instructions
Embedded control optimizations	Deterministic execution CPU Low latency interrupts PC branch in 2 clock cycles	Deterministic execution CPU Low latency interrupts PC branch in 2 clock cycles
System clock frequency	50 MHz	100 MHz
On-chip static RAM	50 KB	136 KB
External PSDRAM	16 MB	16 MB
Interrupts	Yes	Yes
VGA modes	640 x 480 800 x 600	As left plus 1024 x 768 1920 x 1080
Display format	Character graphics (8 x 8 pixel) and pixel graphics	Character graphics (16 x 16 pixel, adjustable) only
Display color depth	256 colors (8 bit)	256 colors on screen from a palette of 4096 colors (12 bit)
Other input/output ports	PS/2 keyboard RS232 SPI port for SD card interface PMOD ports for user expansion	USB keyboard RS232 SPI port for SD card interface PMOD ports for user expansion

APPENDIX 2. CPU INSTRUCTION SET

Assembler mnemonic	**Instruction length (bytes)**	**Encoding**	**Duration (cycles)**
Description		**Parameter stack effect (before -- after)** **3rd 2nd 1st on stack**	**Return stack effect**

NOP	1 byte	0x00	1 cycle
No operation		(--)	(--)

DROP	1 byte	0x01	1 cycle
Remove top item from parameter stack		(x --)	(--)

DUP	1 byte	0x02	1 cycle
Duplicate the top stack item		(x -- x x)	(--)

SWAP	1 byte	0x03	1 cycle
Exchange the two top stack items		(x y -- y x)	(--)

OVER	1 byte	0x04	1 cycle
Make a copy of the second item on the stack		(x y -- x y x)	(--)

NIP	1 byte	0x05	1 cycle
Dispose of the second item on the stack		(x y -- y)	(--)

ROT	1 byte	0x06	1 cycle
Rotate the top three stack times so that the second item becomes top		(x y z -- z x y)	(--)

>R	1 byte	0x07	1 cycle
Remove the top item from the parameter stack and place it on the return stack		(x --)	(-- x)

R@	1 byte	0x08	1 cycle
Copy the top item from the return stack to the parameter stack		(-- x)	(x -- x)

R>	1 byte	0x09	1 cycle
Remove the top item from the return stack and place it on the parameter stack		(-- x)	(x --)

PSP@	1 byte	0x0A	1 cycle
Load the parameter stack with the current value of the parameter stack pointer. The stack pointer is the count of items currently on the stack and also directs the CPU datapath to the first stack item held in SRAM		(-- PSP)	(--)

CATCH	1 byte	0x0B	2 cycles
Branch to the subroutine at address n. Advance the exception, subroutine and return stacks accordingly. The CATCH machine language instruction must be followed by the ZERO machine language instruction in all cases.		(n --)	(--)

RESETSP	1 byte	0x0C	1 cycle
Reset the parameter, return, subroutine and exception stacks to zero.		(--)	(--)

THROW	1 byte	0x0D	1 cycle / 3 cycles
If n=0 then drop the value from the parameter stack and do nothing (1 cycle). If n != 0 then return execution to the address immediately following the calling CATCH plus one, retain the value on the top of the parameter stack, and pop the subroutine and returns stacks accordingly.		(n – n | --)	(--)

+	1 byte	0x0E	1 cycle
Add two 32 bit integer numbers. x3 = x1 + x2		(x1 x2 -- x3)	(--)

-	1 byte	0x0F	1 cycle
Subtract two 32 bit integer numbers. x3 = x1 - x2		(x1 x2 -- x3)	(--)

NEGATE	1 byte	0x10	1 cycle
Negate a 32 bit integer in two's complement format		(x1 -- x2)	(--)

1+	1 byte	0x11	1 cycle
Add 1. x2 = x1 + 1		(x1 -- x2)	(--)

1-	1 byte	0x12	1 cycle
Subtract 1. x2 = x1 - 1		(x1 -- x2)	(--)

2/	1 byte	0x13	1 cycle
Arithmetic shift right. Rotate bits 31 - 1 to bits 30 - 0, and maintain the value of bit 31		(x1 -- x2)	(--)

ADDX	1 byte	0x14	1 cycle
Add two integers with extend flag as carry. The extend flag resides within the datapath and is not otherwise accessible to software. The flag is only affected by arithmetic instructions. See the D+ FORTH word		(x1 x2 -- x3)	(--)

SUBX	1 byte	0x15	1 cycle
Subtraction with extend flag as borrow. The extend flag resides within the datapath and is not otherwise accessible to software. The flag is only affected by arithmetic instructions. See the D- FORTH word		(x1 x2 -- x3)	(--)

=	1 byte	0x16	1 cycle
Returns -1 (true) if x1 = x2		(x1 x2 -- flag)	(--)

<>	1 byte	0x17	1 cycle
Returns -1 (true) if x1 <> x2		(x1 x2 -- flag)	(--)

<	1 byte	0x18	1 cycle
Returns -1 (true) if x1 < x2		(x1 x2 -- flag)	(--)

>	1 byte	0x19	1 cycle
Returns -1 (true) if x1 > x2		(x1 x2 -- flag)	(--)

U<	1 byte	0x1A	1 cycle
Returns -1 (true) if u1 < u2, where u is unsigned		(u1 u2 -- flag)	(--)

U>	1 byte	0x1B	1 cycle
Returns -1 (true) if u1 > u2, where u is unsigned		(u1 u2 -- flag)	(--)

0= or NOT	1 byte	0x1C	1 cycle
Returns -1 (true) if x1 = 0. Equivalent to Boolean NOT		(x1 -- flag)	(--)

0<>	1 byte	0x1D	1 cycle
Returns -1 (true) if x1 <> 0		(x1 -- flag)	(--)

0<	1 byte	0x1E	1 cycle
Returns -1 (true) if x1 < 0		(x1 -- flag)	(--)

0>	1 byte	0x1F	1 cycle
Returns -1 (true) if x1 > 0		(x1 -- flag)	(--)

ZERO or FLASE	1 byte	0x20	1 cycle
Place zero (false) on the stack. Equivalent to ZERO		(-- 0)	(--)

AND	1 byte	0x21	1 cycle
Bitwise AND		(x1 x2 -- x3)	(--)

OR	1 byte	0x22	1 cycle
Bitwise OR		(x1 x2 -- x3)	(--)

INVERT	1 byte	0x23	1 cycle
Bitwise INVERT (also known as bitwise NOT)		(x1 -- x2)	(--)

XOR	1 byte	0x24	1 cycle
Bitwise XOR		(x1 x2 -- x3)	(--)

LSL or 2*	1 byte	0x25	1 cycle
Logical shift left (equivalent to multiply by 2). Bit 0 is set to 0		(x1 -- x2)	(--)

LSR	1 byte	0x26	1 cycle
Logical shift right. Bit 31 is set to 0		(x1 -- x2)	(--)

XBYTE	1 byte	0x27	1 cycle
Sign extend a byte to 32 bits		(x1 -- x2)	(--)

XWORD	1 byte	0x28	1 cycle
Sign extend a word to 32 bits		(x1 -- x2)	(--)

MULTS	1 byte	0x29	5 cycles
Multiply two signed 32 bit integers to produce a 64-bit integer that is held in the top two stack positions, higher word is top of stack		(x1 x2 -- d3)	(--)

MULTU	1 byte	0x2A	1 cycle
Multiply two unsigned 32 bit integers to produce a 64-bit integer that is held in the top two stack positions, higher word is top of stack		(u1 u2 -- ud3)	(--)

DIVS	1 byte	0x2B	42 cycles
Divide two 32-bit signed numbers to produce a 32-bit quotient (top of stack) and a 32-bit remainder (next on stack)		(x1 x2 -- rem quot)	(--)

DIVU	1 byte	0x2C	41 cycles
Divide two 32-bit unsigned numbers to produce a 32-bit quotient (top of stack) and a 32-bit remainder (next on stack)		(u1 u2 – u-rem u-quot)	(--)

FETCH.L	1 byte	0x2D	2 cycles in SRAM
Fetch a longword from memory, big endian		(addr -- n)	(--)

STORE.L	1 byte	0x2E	2 cycles in SRAM
Store a longword in memory, big endian		(n addr --)	(--)

FETCH.W	1 byte	0x2F	2 cycles in SRAM
Fetch a word from memory, big endian		(addr -- n)	(--)

STORE.W	1 byte	0x30	2 cycles in SRAM
Store a word in memory, big endian		(n addr --)	(--)

FETCH.B	1 byte	0x31	2 cycles in SRAM
Fetch a byte from memory		(addr -- n)	(--)

STORE.B	1 byte	0x32	2 cycles in SRAM
Store a byte in memory		(n addr --)	(--)

?DUP	1 byte	0x33	2 cycles
Duplicate the top stack item only if non zero		(x -- x x | x)	(--)

LOAD.B or #.B	2 bytes	0x34, x1	2 cycles
Fetch inline byte to stack and zero extend		(-- x)	(--)

LOAD.W or #.W	3 bytes	0x35, x2, x1	2 cycles
Fetch inline word to stack and zero extend. High byte first		(-- x)	(--)

LOAD.L or #.L	5 bytes	0x36, x4, x3, x2, x1	2 cycles
Fetch inline longword to stack. Highest byte first		(-- x)	(--)

JMP	1 byte	0x37	2 cycles
Redirect program execution to the address on the parameter stack		(addr --)	(--)

JSL	4 bytes	0x38, x3, x2, x1	2 cycles
Redirect program execution to the address specified by the following 3 bytes (highest byte first). Place the original next following instruction address on the return stack		(--)	(-- addr)

JSR	1 byte	0x39	2 cycles
Redirect program execution to the address on the parameter stack and place the original next following instruction address on the return stack		(addr --)	(-- addr)

TRAP	1 byte	0x3A	2 cycles
Jump to the trap vector (address 0x02) and place the original next following instruction address on the return stack. Used for breakpoint debugging		(--)	(-- addr)

RETRAP	1 byte	0x3B	2 cycles
Return from subroutine, execute one program instruction and trap again. Used for single step debugging		(--)	(addr --) (-- addr)

RTI	1 byte	0x3C	2 cycles
Return from an interrupt routine. Similar to RTS but also changes the interrupt controller state to allow further interrupts		(--)	(addr --)

PAUSE	1 byte	0x3D	2 cycles
Task switch. Yield execution of the current task and switch execution to the next-to-execute task.		(--)	(--)

RTS	1 byte	0x40	2 cycles
Return from a subroutine that was entered via a JSR or BSR instruction		(--)	(addr --)

,RTS	1 byte	(0x40 OR opcode)	1 cycle
As RTS but is a compound for any single-cycle instruction that does not itself reference or impact the return stack. The compound instruction saves one cycle and one byte on each subroutine return (e.g. DROP,RTS).		(--)	(addr --)

BEQ	2 bytes	(0x80 OR hi), lo	3 cycles
Branch if the top of stack item is zero. The top 6 bits of the branch offset are in the first instruction byte, the bottom 8 bits of the branch address follow in a second instruction byte. The branch offset is calculated from the address of the second byte		(flag --)	(--)

BRA	2 bytes	(0xC0 OR hi), lo	3 cycles

Branch. The top 6 bits of the branch offset are in the first instruction byte, the bottom 8 bits of the branch address follow in a second instruction byte. The branch offset is calculated from the address of the second byte	(--)	(--)

APPENDIX 3. MEMORY MAP (V2.0, NEXYS 2)

Overall memory map

Region	Implementation	Size	Bottom	Top
System memory	FPGA block RAM	50 KB	0x000000	0x00F7FF
Hardware registers	FPGA logic	2 KB	0x00F800	0x00FFFF
External memory	PSDRAM chip	16 MB	0x010000	0xFFFFFF

System memory

Region	Size	Bottom	Top
Forth system software and user applications	44 KB	0x000000	0x00AFFF
Parameter stack	2 KB	0x00E000	0x00E7FF
Return stack	2 KB	0x00E800	0x00EFFF
Character RAM	2 KB	0x00F000	0x00F7FF

Hardware registers

Name	Function	R/W	Hex	Dec
SCREENPLACE	Pointer to the start of the character/text screen buffer in external memory	R/W	0xF800	63488
GFXPLACE	Pointer to the start of the pixel graphics screen buffer in external memory	R/W	0xF804	63496
BACKGROUND	Screen background color	R/W	0xF808	63496
MODE	Graphics mode - see below for bit level	R/W	0xF80C	63500
RS232DIN	RS232 data in	R	0xF810	63504
RS232DOUT	RS232 data out. Writing to this register triggers the RS232 port to output the byte	W	0xF814	63508
RS232UBRR	UBRR = 50,000,000 / (baud +1) / 16		0xF818	63512
RS232STAT	RS232 port status - see below for bit level		0xF81C	63516
PS2DIN	PS/2 port data in	R	0xF820	63520
SYSCOUNT	Unsigned 32 bit counter clocked at 50 MHz	R	0xF824	63524
MSCOUNT	Unsigned 32 bit counter clocked at 1 KHz	R	0xF828	63528
IRQMASK	Interrupt request mask - see below	R/W	0xF82C	63532
SEVENSEG	4 character seven segment output on the	W	0xF830	63536

	Nexys 2 board			
SWITCHES	8 switch inputs on the Nexys 2 board	R	0xF834	63540
SPIDATA	SPI data byte. Writing to this register triggers the SPI transmit/receive cycle	R/W	0xF838	63544
SPICONTROL	Control of SPI port - see below for bit level	R/W	0xF83C	63548
SPISTATUS	Status of SPI port - see below for bit level	R	0xF840	63552
SPICLKDIV	SPI clock = 50,000,000 / SPICLKDIV	W	0xF844	63556
VBLANK	Bit 0 is set during the VGA vertical blank interval and cleared otherwise	R	0xF848	63560

APPENDIX 4. MEMORY MAP (V4.0, NEXYS 4)

Overall memory map

Region	Implementation	Size	Bottom	Top
System memory	FPGA block RAM	136 KB	0x000000	0x03F7FF
Hardware registers	FPGA logic	2 KB	0x03F800	0x03FFFF
External memory	PSDRAM chip	16 MB	0x040000	0xFFFFFF

System memory

Region	Size	Bottom	Top
Forth system software and user applications	128 KB	0x000000	0x01FFFF
Not used - reserved for expansion	108 KB	0x020000	0x03D7FF
Character RAM	4 KB	0x03C000	0x03D7FF
Palette RAM	2 KB	0x03D000	0x03DFFF
Subroutine and exception local variables	2 KB	0x03D800	0x03E7FF
USER data area	2 KB	0x03E000	0x03EFFF
Multitasking control	2 KB	0x03F000	0x03F7FF

Hardware registers

Name	Function	R/W	Hex	Dec
SCREENPLACE	Pointer to the start of the character/text screen buffer in external memory	R/W	0x03F800	260096
BACKGROUND	Screen background color	R/W	0x03F808	260104
MODE	Graphics mode – see below for bit level	R/W	0x03F80C	260108
RS232DIN	RS232 data in	R	0x03F810	260112
RS232DOUT	RS232 data out. Writing to this register triggers the RS232 port to output the byte	W	0x03F814	260116
RS232DIVIDE	DIVIDE = 100,000,000 / baud rate	W	0x03F818	260120
RS232STATUS	RS232 port status – see below for bit level	R	0x03F81C	260124
PS2DIN	PS/2 port data in	R	0x03F820	260128
SYSCOUNT	Unsigned 32 bit counter clocked at 100 MHz	R	0x03F824	260132
MSCOUNT	Unsigned 32 bit counter clocked at 1 KHz	R	0x03F828	260136
IRQMASK	Interrupt request mask – see below	R/W	0x03F82C	260140

SEVENSEG	8 character Nexys4 seven segment display	W	0x03F830	260144
SWITCHES	16 switch inputs on the Nexys 2 board	R	0x03F834	260148
SPIDATA	SPI data byte. Writing to this register triggers the SPI transmit/receive cycle	R/W	0x03F838	260152
SPICONTROL	Control of SPI port – see below for bit level	R/W	0x03F83C	260156
SPISTATUS	Status of SPI port – see below for bit level	R	0x03F840	260160
SPICLKDIV	SPI clock = 100,000,000 / SPICLKDIV	W	0x03F844	260164
VBLANK	Bit 0 is set during the VGA vertical blank interval and cleared otherwise	R	0x03F848	260168
INTERLACE	Number of interlace scanlines	R/W	0x03F84C	260172
CHARWIDTH	Width of each character in pixels	R/W	0x03F850	260176
CHARHEIGHT	Height of each character in pixels	R/W	0x03F854	260180
VGAROWS	Number of complete character rows	R/W	0x03F858	260184
VGACOLS	Number of complete character columns	R/W	0x03F85C	260188

Hardware registers - bit level

Register/bit	**4**	**3**	**2**	**1**	**0**
MODE	n/a	0 = 16/16 color mode 1= 256/0 color mode	000 = Display off 001 = 640 * 480 010 = 800 * 600 010 = 1024 * 768 100 = 1902 * 1080		
RS232STATUS	n/a	n/a	n/a	Transfer bus enable	Read data available (strobe)
IRQMASK	MS	PS/2	RS232 TBE	RS232 RDA	n/a
SPI CONTROL	n/a	0 = clock rests lo 1 = clock rests hi	0 = latch then shift 1= shift then latch	0 = MOSI default lo 1 = MOSI default hi	0 = CS lo 1 = CS hi
SPI STATUS	n/a	MISO	Write protect	Chip detect	SD transfer bus ready

APPENDIX 5. PALETTE RAM COLOR TABLE

Color #	Color	RGB	Hue	Saturation	Luminescence
0	Black	0x000	0	0%	0%
1	Grey	0x888	0	0%	50%
2	Silver	0xBBB	0	0%	75%
3	White	0xFFF	0	0%	100%
4	Red	0xF00	0	100%	50%
5	Yellow	0xFF0	60	100%	50%
6	Green	0x0F0	120	100%	50%
7	Cyan	0x0FF	180	100%	50%
8	Blue	0x00F	240	100%	50%
9	Magenta	0xF0F	300	100%	50%
10	Maroon	0x800	0	100%	25%
11	Olive	0x880	60	100%	25%
12	Dark green	0x080	120	100%	25%
13	Teal	0x088	180	100%	25%
14	Navy	0x008	240	100%	25%
15	Purple	0x808	300	100%	25%
16-255	various				

APPENDIX 6. IMPLEMENTATION OF ANSI FORTH WORDS

Thus appendix is organized in alignment with the word sets of the ANSI FORTH standard and documents the availability of these words on the N.I.G.E. Machine.

- For the ANSI CORE word set, "status" indicates whether the word has been implemented on the N.I.G.E. Machine (Y/N). An asterisk indicates that the word is implemented but with some limitation as compared with the ANSI FORTH specification.

- For the optional ANSI word sets, only those words implemented on the N.I.G.E. Machine are listed. Where a filename (e.g. "SYSTEM.F") is indicated, the word must be included from that file on an appropriately mounted SD card.

CORE words

Word	Status	Notes
!	Y	See also W!
#	N	Use U# instead. Division with a 64-bit dividend is not implemented on the N.I.G.E. Machine.
#>	N	Use U#> instead. See #.
#S	N	Use U#S instead. See #.
‘	Y	
(	Y	
*	Y	
*/	Y*	The intermediate value is single (32-bit) precision only
*/MOD	Y*	The intermediate value is single (32-bit) precision only
+	Y	
+!	Y	See also W+! and C+!
+LOOP	Y	
,	Y	See also W, M, and $,
-	Y	
.	Y	
."	Y	
/	Y	
/MOD	Y	
0<	Y	
0=	Y	
1+	Y	
1-	Y	
2!	N	Less suitable for a big-endian processor
2*	Y	
2/	Y	
2@	N	Less suitable for a big-endian processor
2DROP	Y	
2DUP	Y	
2OVER	Y	
2SWAP	Y	
:	Y	

;	Y	
<	Y	
<#	Y	
=	Y	
>	Y	
>BODY	N	Would be a no operation in the N.I.G.E. Machine FORTH environment. Will not be implemented for space and efficiency reasons
>IN	Y	
>NUMBER	Y	
>R	Y	
?DUP	Y	
@	Y	
ABORT	Y	
ABORT"	N	Will not be implemented for space and efficiency reasons
ABS	Y	
ACCEPT	Y	Tab characters are interpreted as white space
ALIGN	Y	No-operation on the N.I.G.E. Machine
ALIGNED	Y	Alignment is taken to the next highest longword boundary
ALLOT	Y	
AND	Y	
BASE	Y	The variable BASE is held on the exception stack and upon EXIT or THROW its value is restored the value it held before CATCH
BEGIN	Y	
BL	Y	
C!	Y	See also W!
C,	Y	See also W,
C@	Y	See also W@
CELL+	Y	
CELLS	Y	
CHARS	Y	No-operation on the N.I.G.E. Machine
CONSTANT	Y	
COUNT	Y	
CR	Y	
CREATE	Y	
DECIMAL	Y	
DEPTH	Y	
DO	Y	
DOES>	Y	
DROP	Y	
DUP	Y	
ELSE	Y	
EMIT	Y	
ENVIRONMENT?	N	Will not be implemented for space and efficiency reasons
EVALUATE	Y	
EXECUTE	Y	Tab and new-line characters are interpreted as white spaces
EXIT	Y	

FILL	Y	See also FILL.W
FIND	Y	
FM/M	N	Will not be implemented for space and efficiency reasons
HERE	Y	
HOLD	Y	
I	Y	
IF	Y	
IMMEDIATE	Y	
INVERT	Y	
J	Y	
KEY	Y	
LEAVE	Y	
LOOP	Y	
LSHIFT	Y	
M*	Y	
MAX	Y	
MIN	Y	
MOD	Y	
MOVE	Y	
NEGATE	Y	
OR	Y	
OVER	Y	
POSTPONE	Y	
QUIT	Y	
R>	Y	
R@	Y	
RECURSE	Y	
REPEAT	Y	
ROT	Y	
RSHIFT	Y	
S"	Y	Multiple buffers are provided in interpret mode. See also C" and ,"
S>D	N	Equivalent to FALSE on the N.I.G.E. Machine.
SIGN	Y	
SM/REM	N	Division with a 64-bit dividend is not supported
SOURCE	Y	
SPACE	Y	
SPACES	Y	
STATE	Y	
SWAP	Y	
THEN	Y	
U.	Y	
U<	Y	
UM*	Y	
UM/MOD	N	Division with a 64-bit dividend is not supported
UNLOOP	Y	
UNTIL	Y	

VARIABLE	Y	
WHILE	Y	
WORD	Y	
XOR	Y	
[	Y	
[']	Y	
[CHAR]	Y	
]	Y	

CORE EXTENSION words

Word	File	Notes
.(		
.R		
0<>		
0>		
<>		
?DO		
AGAIN		
BUFFER:		Allocates space in PSDRAM. Suitable for larger data blocks. See also SBUFFER: which allocates space in SRAM
C"		In the interpretation state C" will copy the text until the following " to the PAD as a counted string and return its address
CASE		
COMPILE,		
DEFER		
ENDCASE		
ENDOF		
FALSE		Returns 0
HEX		
INTERPRET		Interpret a line from the input buffer
IS		
MARKER		
NIP		
OF		
PAD		
PARSE		
PICK		
RESTORE-INPUT		SAVE-INPUT and RESTORE-INPUT use internal variables to store the input source specification. RESTORE-INPUT does not return a flag
SAVE-INPUT		See RESTORE-INPUT
TRUE		
U.R		
U>		
UNUSED		
WITHIN		

\		

DOUBLE-NUMBER words

D+		
D-		

EXCEPTION words

CATCH		The N.I.G.E. Machine system software has a top-level CATCH that assumes throw codes are counted strings which it will display as part of an error message
THROW		Use c" My error message" THROW to abort execution and return to QUIT with an error message

FACILITY words

KEY?		See also KKEY?, SKEY? and SKEY?
MS		

FILE ACCESS words

CLOSE-FILE	SYSTEM.F	
CREATE-FILE	SYSTEM.F	
DELETE-FILE	SYSTEM.F	
FILE-POSITION	SYSTEM.F	
FILE-SIZE	SYSTEM.F	
OPEN-FILE	SYSTEM.F	
R/O	SYSTEM.F	
R/W	SYSTEM.F	
READ-FILE	SYSTEM.F	
READ-LINE	SYSTEM.F	
REPOSITION-FILE	SYSTEM.F	
RESIZE-FILE	SYSTEM.F	
W/O	SYSTEM.F	
WRITE-FILE	SYSTEM.F	
WRITE-LINE	SYSTEM.F	
FLUSH-FILE	SYSTEM.F	
INCLUDE		
RENAME-FILE	SYSTEM.F	

MEMORY words

ALLOCATE	SYSTEM.F	
FREE	SYSTEM.F	
RESIZE	SYSTEM.F	

PROGRAMMING TOOLS words

.S		
?		
DUMP		
SEE	TOOLS.F	
WORDS		
STATE		

STRING words

COMPARE		See also $=
SLITERAL		See also CLITERAL

SEARCH-ORDER words

DEFINITIONS		
FIND		
FORTH-WORDLIST		
GET-CURRENT		
GET-ORDER		
SEARCH-WORDLIST		
SET-CURRENT		
SET-ORDER		
WORDLIST		

APPENDIX 7. SYSTEM SPECIFIC FORTH WORDS

This appendix lists N.I.G.E. Machine specific FORTH words in addition to those included in the ANSI word sets. The words are organized by function. Where a filename is indicated in parenthesis (e.g. SYSTEM.F), that file must be included from an appropriately mounted SD card. This list focuses on commonly used words. Appendix 8 lists further N.I.G.E. Machine specific words more relevant to customization of the system software.

Local variables

Local variable syntax is based on VFX FORTH and this documentation is adapted from the VFX FORTH documentation. A maximum of 16 local variables are permitted on the N.I.G.E. Machine.

`{: ni1 ni2 ... | lv1 lv2 ... -- o1 o2 :}` defines named inputs, local variables, and outputs. The items between {: and | are named inputs. The items between | and -- are local variables. The items between -- and :} are comments. The named inputs are automatically copied from the data stack on entry. Named inputs and local variables can be referenced by name within the word during compilation. The output names are dummies to allow a complete stack comment to be included.

{ is accepted in place of {: and } in place of :}. Named inputs and locals return their values when referenced, and must be preceded by -> or TO in order to perform a store. VALUE types are not implemented on the N.I.G.E. Machine therefore -> and TO are only applicable to local variables.

Word	Stack effect	Description
{: or {	(--)	Begin a list of named inputs and local variables
|	(--)	Separator between named inputs and local variables
--	(--)	Terminate a list of named inputs and local variables
:} or }	(--)	Terminate a list of named inputs and local variables

Multitasking

The N.I.G.E. Machine operates a round-robin multitasker. Task switching may be preemptive or cooperative.

Word	Stack effect	Description
USER	(offset <name> --)	Create a user variable with name, <name>, at offset bytes from the start of the user area. The FORTH word <name> will be accessible by all tasks, but each task has access only to its local copy. The first available slot for application specific user variables is at n = 44, and 980 free storage bytes are available from that point.

+USER	(size <name> --)	Create a user variable with name, <name>, of size bytes at the next available offset in the user variable area
RUN	(... pn n XT -- VM# true | false)	Find and initialize a new task to take n stack parameters (... pn) and execute task XT. Return the number of the task allocated to this task (VM) and true if successful, or false if all tasks are currently otherwise allocated. The newly created task will be positioned in the round-robin sequence immediately after the current task. Tasks are numbered 0 through 31. Note that XT must either be an infinite loop or contain code to self-abort the task
PAUSE	(--)	Task switch. Yield CPU execution of the current task and switch CPU execution to the next-to-execute task
SINGLE	(--)	Disable multitasking. PAUSE instructions will be treated as a NOP. By default multitasking is enabled at power-on on the N.I.G.E. Machine
MULTI	(--)	Enable multitasking. Note that if there is only a single active task then PAUSE will be treated as NOP
SLEEP	(VM --)	Put task VM to sleep by removing it from the list of executing tasks. VM remains allocated and can be woken at a later time
WAKE	(VM --)	Wake task VM by inserting it into the list of executing tasks immediately following the current task
STOP	(VM --)	Deallocate task VM and remove it from the list of executing tasks. Task VM may now be recycled by RUN
VIRQ	(XT VM --)	Virtual interrupt. Cause task VM to branch to the subroutine at XT and then return to its prior point of execution. The virtual interrupt will occur at the time when task VM is next scheduled to execute. (VIRQ does not cause execution to pass to task VM early/out of sequence)
THIS-VM	(-- VM)	Return the number of the currently executing task. Tasks are numbered 0 through 31. A power-on task 0 will be executing the FORTH system software
THIS-SLEEP	(--)	Put the currently executing task to sleep by taking it out of the list of executing tasks. The task may be woken by another task. Note that THIS-SLEEP should be used instead of THIS-VM SLEEP to ensure correct task switching behavior
THIS-STOP	(--)	Deallocate the currently executing task to sleep and take it out of the list of executing tasks. The task may now be recycled by RUN. Note that THIS-STOP should be used instead of THIS-VM STOP to ensure correct task switching behavior

ACQUIRE	(sem --)	Acquire the binary semaphore (sem) or wait in a busy loop until it becomes free. Note that a semaphore can be any FORTH variable with global scope. Semaphores are minimum single byte in length (word or longword length variables may also be used). A semaphore contains the number of the latest successfully acquiring task XOR 255, or 0 if not acquired.
RELEASE	(sem --)	Release the binary semaphore (sem). See also ACQUIRE
PREEMPTIVE	(n --)	Enable preemptive multitasking with period of n instructions between task switches. If n = 0 then preemptive multitasking is disabled.

System

Word	Stack effect	Description
RESET	(--)	Reset the N.I.G.E. Machine to power on configuration but otherwise preserve memory contents
COUNTER	(-- ms)	Current count of the rolling 32 bit millisecond counter
SEVENSEG	(n --)	Display a 32 bit value on the Nexys 4 eight character, seven segment, LED display

Screen display

BACKGROUND	(x --)	Set the current screen background color to the specified value. The color is specified as a 8 bit (v2.0) or 12 bit (v 3.0) value in the form RRRGGGBB or RRRRGGGGBBBB
INK	(addr --)	**Byte length** FORTH variable holding the foreground color to be used by EMIT. Access with C!
INTERLACE	(flag --)	Sets or unsets interlace mode. In interlace mode there are 2 blank (background color) scan lines between each row of 8 bit high characters. The number of screen rows (ROWS) is adjusted accordingly
VGA	(n --)	Sets the VGA mode: 0 - display off 1 - 640 * 480 2 - 800 * 600 (default) 3 - 1024 * 768 4 - 1920 * 1080 The number of screen rows (ROWS) and columns (COLUMNS) are also adjusted accordingly
COLORMODE	(n --)	Sets the color mode (see section 2):

		0 - 16/16 1 - 256/0 (default)
CLS	(--)	Clear the screen
HOME	(--)	Position the cursor at the top left screen position without clearing the screen
TAB	(-- addr)	VARIABLE pointing to the current size of tab stops. The default is 3

SD card and FAT file system

MOUNT	(--)	Mount an SD card and initialize the FAT32 data structures. Call MOUNT after inserting or replacing an SD card

SD card and FAT file system (SYSTEM.F)

DIR	(--)	List the current directory
CD	"FILEPATH"	Set the current directory to FILEPATH
DELETE	"FILEPATH"	Delete the directory given as FILEPATH

RS232 port

SEMIT	(x --)	Emit a character to the RS232
SKEY?	(-- flag)	Check if a character is waiting to be read from the 256 byte circular buffer maintained for the RS232 port
SKEY	(-- n)	Wait for and read the next character available at the RS232 port
STYPE	(c-addr n --)	Type a string to the RS232 (asynchronous operation)
SEMIT	(x --)	Emit a character to the RS232
SZERO	(--)	Abandon all waiting characters in the RS232 input buffer and reset the buffer pointer to zero
BAUD	(n --)	Set the baud rate to n - CHECK THIS FOR NEW CLOCK RATE
>REMOTE	(--)	Redirect FORTH environment output to the RS232. The redirection vectors are held on the exception stack and reset upon EXIT or THROW to their values as at before CATCH.
>LOCAL	(--)	Redirect FORTH environment output to the screen. See >REMOTE.
<REMOTE	(--)	Receive FORTH environment input from the RS232. See >REMOTE.

<LOCAL	(--)	Receive FORTH environment input from the keyboard. See >REMOTE.

Memory (SYSTEM.F)

MEM.SIZE	(addr -- n)	Return the size of an allocated memory block
AVAIL	(--)	Show free memory block list
UNAVAIL	(--)	Show used memory block list

Compiler extensions

HERE1	(-- addr)	VARIABLE pointing to the dictionary pointer for the PSDRAM dictionary space. Only used by BUFFER:
INLINESIZE	(-- addr)	VARIABLE pointing to the maximum code-length in bytes that the compiler will compile inline rather than as a subroutine call. The default value is 10 and the minimum allowable is 9 since certain code, such as LOOP code, must be compiled inline
W,	(w --)	Allocate 2 bytes in the dictionary and store a word from the stack
M,	(addr u --)	Allocate and store u bytes from addr into the dictionary. u is not saved in the dictionary. Compiles a string or other block of data from memory
$,	(addr u --)	Allocate and store u bytes from addr into the dictionary. u is is compiled as the first byte. Compiles a counted string.
LITERAL	(n --)	Compile a literal to the dictionary
CLITERAL	(addr u --)	Compile to the dictionary a string literal as an executable that will be re-presented at run time as a counted string c-addr

Programming tools (TOOLS.F)

DASM	(addr n --)	Disassemble n bytes starting at address addr. Note that DASM does not identify literal strings within word definitions and so disassembly will become unreliable when they are encountered
SIZEOF	(xt -- n)	Return the size of an executable
.’	(addr -- c-addr n true | false)	If addr points to an executable FORTH word, provide the name of the word and return true. Otherwise return false

Editor (EDITOR.F)

The editor is a simple keyboard based file text editor primarily intended for editing FORTH language source files on the SD card file system. The FORTH words to launch the editor and keyboard commands are listed below.

EDIT	"FILENAME"	Open an existing file and launch the editor
EDITNEW	"FILENAME"	Create a new file on the SD card and launch the editor
Page up, page down, cursor keys		Cursor movement
Home		Move cursor to first line of file
End		Move cursor to last line of file
Backspace, delete		Character deletion
Esc		Exit without saving changes
F1		Save changes and exit
F2		Save changes and continue

Other supporting words

BINARY	(--)	Set BASE = 2
NOT	(n – n)	Equivalent to 0=
XBYTE	(n – n)	Sign extend a byte on the stack to 32 bits
XWORD	(n –n)	Sign extend a word on the stack to 32 bits
W@	(addr – n)	Fetch a word from memory
W!	(n addr --)	Store a word in memory
FILL.W	(addr n w --)	Fill a region of memory with n words w. FILL.W utilizes the STORE.W machine language instruction and is faster than FILL in accessing PSDRAM
UPPER	(x -- X)	Convert one ASCII character to uppercase
DIGIT	(char base -- n true | char false)	Convert a single ASCII character to a number in the given base
NUMBER?	(c-addr u - false | n true ,)	Convert an ASCII string to a number and return with a success or failure flag
COMP	(n1 n2 – n)	Return -1 if n1<n2, +1 if n1>2, 0 if n1=n2
$=	(c-addr1 u1 c-addr2 u2 -- flag)	Test two strings for equality. Case insensitive
MASK@	(addr mask -- u)	Fetch the longword at address addr bitwise though the read-enable mask. Equivalent to @ followed by OR
MASK!	(u addr mask --)	Store the longword u at address addr bitwise through the write-enable mask

APPENDIX 8. FURTHER SYSTEM SPECIFIC WORDS

This appendix documents the remainder of N.I.G.E. Machine specific words likely to be relevant to customization of the FORTH system software.

Screen display

VEMIT	(n --)	Emit a character to the VDU and process any screen-codes (e.g. CR or BACKSPACE) accordingly. Move the current screen cursor position forward
VTYPE	(c-addr n --)	Type a string to the VDU and process any screen-codes (e.g. CR or BACKSPACE) accordingly. Move the current screen cursor position forward
EMITRAW	(n --)	Emit a character to the VDU without processing any screen-codes. Move the current screen cursor position forward
TYPERAW	(c-addr n --)	Type a string to the VDU without processing any screen-codes. Move the current screen cursor position forward
CSR-PLOT	(x --)	Plot the specified ASCII character at the current cursor position. Does not change the cursor position.
CSR-ADDR	(-- addr)	Return the memory address of the current cursor position (as held by CSR-X and CSR-Y) within the screen buffer in PSDRAM
CSR-X	(-- col)	Return the current column position of the cursor
CSR-Y	(-- row)	Return the current row position of the cursor
CSR-ON	(--)	Plot the cursor symbol at the current cursor position. The character at that position is saved in an internal variable. (Used by ACCEPT)
CSR-OFF	(--)	Unplot the cursor symbol from the current cursor position and restore the character which was previously there. (Used by ACCEPT)
CSR-FWD	(--)	Advance the cursor by one character
CSR-BACK	(--)	Move back the cursor by one character
CSR-TAB	(--)	Advance the cursor to the next tab stop
NEWLINE	(--)	Scroll the screen downwards by one line of text and return the cursor to the first column of the blank line below
SCROLL	(n -- flag)	Scroll the screen fwd or back n lines within the 120 line frame buffer. Returns true if out of range or false otherwise

ROWS	(-- rows)	Return the current number of screen rows.
COLS	(-- cols)	Return the current number of screen columns.
SCRSET	(--)	Set the ROWS and COLS internal variables according to the current screen configuration. Used by INTERLACE and SCREENMODE
SCREENBASE	(-- addr)	CONSTANT address of the pre-allocated screen buffer
SCREENPLACE	(-- addr)	VARIABLE holding the current address of the screen buffer. This variable address is a memory-mapped hardware register. Default is SCREENBASE
VWAIT	(--)	Wait for the VGA vertical blank interval. Used prior to writing to or moving the screen buffer

SD card and FAT file system

SD.init	(--)	Reset the SD card, check the SD version number and initialize the card
SD.sector-code	(n -- b4 b3 b2 b1)	Take a sector number n, scale according to the SD card version and split into bytes in preparation for a SD care read or write sector command
SD.select&check	(--)	Asset SD card chip select and wait for the card to signal ready
SD.read-sector	(addr n --)	Read 512 bytes from sector n into a buffer at addr
SD.write-sector	(addr n --)	Write 512 byte to sector n from addr
FAT.read-long	(addr n -- x)	Get a little endian longword (x) from the buffer at address (addr) and position (n)
FAT.write-long	(x addr n --)	Write a little endian longword (x) to the buffer at address (addr) and position (n)
FAT.read-word	(addr n -- x)	Get a little endian word (x) from the buffer at address (addr) and position (n)
FAT.write-word	(x addr n --)	Write a litte endian word to the buffer at address (addr) and position (n)
FAT.UpdateFSInfo	(--)	Update the FAT32 FSInfo sector with next free cluster
FAT.clus2sec	(n -- n)	Given a valid cluster number return the number of the first sector in that cluster
FAT.get-fat	(cluster -- value)	Return the FAT entry (value) for the given cluster

FAT.put-fat	(value cluster --)	Place value in the FAT location for the given cluster
FAT.string2filename	(addr n -- addr)	Convert an ordinary string to a short FAT filename
FAT.find-file	(addr n -- dirSector dirOffset firstCluster size flags TRUE | FALSE)	Find a file with filename (addr n) in the current directory. Return FALSE if not found or TRUE and file system parameters otherwise
FAT.load-file	(addr firstCluster --)	Load a file to memory at address addr, specifying the file by the number of its first cluster

SD card and FAT file system (SYSTEM.F)

FAT.FindFreeCluster	(-- n)	Return the first cluster on the disk
FAT.save-file	(addr size firstCluster)	Save a file to disk assuming size <> 0
FAT.FindFreeEntry	(dirCluster -- dirSector dirOffset TRUE | FALSE)	Find the first available entry in a directory
FAT.size2space	(size -- space)	Rule for the space to allocate to a file
FAT.new-file	(dirSector dirOffset firstCluster size fam -- fileid ior)	
FAT.copynonblank	(out-addr in-addr -- out-addr+1)	copy a non-space character from in-out
FAT.Filename2String	(addr -- addr n)	convert a short FAT filename to an ordinary string

PS/2 keyboard

KKEY?	(-- flag)	Check if a character is waiting to be read from the 256 byte circular buffer maintained for the PS/2 keyboard
KKEY	(-- n)	Wait for and read the next character available from the PS/2 keyboard. Returns the ASCII code for the key
PS2DECODE	(n -- n)	Decode a PS/2 scan code into ASCII. Returns 0 if there is no valid ASCII match. (PS2DECODE is called directly by the PS/2 interrupt routine during normal operation.)

APPENDIX 9. CROSS-ASSEMBLER

Assembler lines consist of up to four fields separate by white spaces in the following format. White spaces are the SPACE and TAB characters.

```
LABEL       INSTRUCTION       VALUE       COMMENT
```

1. LABEL

Any legitimate FORTH identifier can be used as an optional label. The identifier will become a CONSTANT in the hosting FORTH environment. The value assigned to the label is the current program counter address.

2. INSTRUCTION

An instruction is one of the following:

- the assembler mnemonic for a CPU instruction (documented in Appendix 1)
- an assembler macro command (documented below)
- an assembler directive (documented below)

Assembler macro commands

IF	Flow command. Identical operation to the corresponding FORTH word
WHILE	
THEN	
ELSE	
BEGIN	
AGAIN	
REPEAT	
UNTIL	
DO	
LOOP	
+LOOP	Note that +LOOP is only implemented to support positive increments
UNLOOP	
J	Note: use R@ for I

Assembler directives

CMD	Usage: `CMD .” Hello World”` Execute the remainder of the line as a FORTH expression in the hosting FORTH environment

EQU	Usage: `label EQU value` Assign the value to the label. Note that label will become a FORTH word in the hosting FORTH environment. Value may be a numerical value or a FORTH expression. Precede a hexadecimal value with HEX. Note that the cross assembler automatically resets to DECIMAL at the start of each new line of code
DC.S	Usage: `DC.S some text` Include the following text (until end of line) as a string of characters in memory. Note that the text is not counted or terminated. Take care of (unintended) spaces and tabs between the text and the end of line as these will be included by the assembler
DC.L	Usage: `DC.L 65536 356 24` Define one or more constant longword. Include the following values as 4 byte long-word constants in memory. **The values are given in reverse order, with the last value in the list placed in the lowest memory location.**
DC.W	Usage: `DC.L HEX 0CD FF` Define one or more constant words. Include the following values as 2 byte word constants in memory. **The values are given in reverse order, with the last value in the list placed in the lowest memory location**
DC.B	Usage: `DC.B 255` Define constant byte. Include the following values as a byte constants in memory. **The values are given in reverse order, with the last value in the list placed in the lowest memory location**
DS.L	Usage: `DS.L 12` Define storage longwords. Reserve the following value number of longwords in memory and initialize to zero. In this example 48 bytes will be reserved and initialized zero.
DS.W	Usage: `DS.W 8` Define storage words. Reserve the following value number of longwords in memory and initialize to zero. In this example 16 bytes will be reserved
DS.B	Usage: `DS.B 1` Define storage words. Reserve the following value number of longwords in memory and initialize to zero. In this example 1 byte will be reserved

3. VALUE

Value is applicable to certain CPU instructions and assembler directives. The value field is evaluated in the hosting FORTH environment. It may therefore be simple number or a FORTH expression. Precede a hexadecimal values with HEX. Note that the cross assembler automatically resets to DECIMAL at the start of each new line of code.

Supporting FORTH words that may be useful in the VALUE field

REL	Defined by the assembler as `: rel 1+ - ;`
DEL	Defined by the assembler as `: del 1- - ;`

4. COMMENT

The comment characters are ";" and "(". The assembler ignores all text from the comment character to end of line. An optional ")" character may be used for presentation purposes with "(" but is not required.

APPENDIX 10. FORTH SYSTEM DICTIONARY STRUCTURE

The FORTH system software uses a dictionary structure as follows

Field	**Length**	**Description**
LINK field	4 bytes	Points to the NAME field of the previous FORTH word. Set to zero for the last word in the dictionary.
NAME field (first byte)	1 byte	The first byte of the NAME field is organized as follows bit 7: PRECEDENCE bit. Always set to 1 bit 6: IMMEDIATE bit. Set for immediate words only bit 5: SMUDGE bit. Set to hide a word in the dictionary bits 4-0: LENGTH of the name in bytes (0 - 32 bytes)
NAME field (remaining bytes)	[LENGTH] bytes	NAME in ASCII characters. Length as given by the LENGTH field in byte 1 of the field. The FORTH interpreter/compiler is case insensitive.
SIZE field	2 bytes	The SIZE field is organized as follows bit 15: MUSTINLINE. Set to force inline compilation bit 14: NOINLINE. Set to prevent inline compilation bits 13-0: SIZE of the executable in bytes
CODE field	[x] bytes	Executable code. Size as given by the SIZE field.

There is no explicit parameter field in this dictionary structure because the FORTH system software is subroutine threaded. Words created with CREATE are allocated a parameter field immediately after the in-line executable code.

APPENDIX 11. INTERRUPT VECTOR TABLE

The default CPU interrupt vector table is as follows:

System memory location	**Interrupt name**	**Interrupt description**
0x00	RESET	Power-on and system reset execution address
0x02	TRAP	Vector for TRAP and RETRAP instructions. Available for program debugging.
0x04	RS232RDA	Triggered by Read Data Available strobe on the RS232 port. The native interrupt handler places incoming data in a 256 byte buffer that is accessed via the SKEY? And SKEY words.
0x06	RS232TBE	Triggered by Transfer Bus Enable on the RS232 port. Used by the STYPE word for asynchronous RS232 output.
0x08	PS2	Triggered by data available strobe on the PS/2 port. The native interrupt handler places incoming data in a 256 byte buffer that is accessed via the KKEY? And KKEY words. Note that on the Nexys 4 board, USB keyboard inputs are converted to PS/2 scan codes by the Nexys 4 interface.
0xA0	MS	Milliseconds interrupt. Used by the TIMEOUT word.
0xC0 -	various	Available for user expansion

APPENDIX 12. HARDWARE MULTITASKING CONTROL

The N.I.G.E. Machine has hardware multitasking capabilities built into the soft-core CPU. At any one time the CPU will be executing a single task upon the current VM. When a task switch occurs, the state of the currently executing VM is saved to the hardware multitasking core, and the state of the next-to-execute VM is concurrently loaded from the hardware multitasking core into the CPU. Such a task switch occurs in 2 clock cycles and requires no support from a task monitor in software.

Multitasking control is achieved through configuring a set of memory mapped registers and by inclusion of the machine language instruction PAUSE in application program code. The FORTH system software contains words to handle all aspects of hardware multitasking control in the default mode of round-robin, cooperative multitasking (see Appendix 7). This appendix contains further details of hardware multitasking control at the hardware level that would be relevant to customizing the FORTH system software or developing an alternative multitasking model.

12.1 Simple registers

The following memory mapped registers are concerned with multitasking at the overall level

Name	Function	R/W	Hex	Dec
SINGLEMULTI	Enable ('1') or disable ('0') multitasking. If a PAUSE machine language instruction instruction is encountered with multitasking disabled then it will be treated as a NOP	R/W	0x03F000	258048
CURRENTVM	The number of the currently executing task. Tasks are numbered 0 through 31. A power-on task 0 will be executing the FORTH system software	R	0x03F004	258052
INTERVAL	The interval for pre-emptive multitasking, in clock cycles. If INTERVAL = 0 then pre-emptive multitasking is off.	R/W	0x03F008	258056

12.2 Task Control

There are 32 task control registers, one corresponding to each task. The registers are 16 bits wide and are memory mapped with read and write access. The lower 5 bits of each register are interpreted directly by the hardware multitasking core as the number of the next-to-execute task. Thus if TASKCONTROL0 has the lowest 5 bits set as "000010", then when task #0 yields the CPU at a pause instruction then CPU execution will pass to task #2. The remaining 11 bits of each register are not interpreted by the hardware multitasking core and may be used to assist with the implementation of various multi-tasking models.

In the FORTH system software, bit 15 of each task control register indicates whether that VM has already been assigned ('1') or not ('0'), and bits 5 to 9 are a backwards pointer to the previously

executing task. The scheduler utilizes the linked list structure thus created between executing tasks to efficiently handle the insertion or removal of new tasks.

Name	Function	R/W	Hex	Dec
TASKCONTROL0	Task control of task 0	R/W	0x03F200	258560
TASKCONTROL1	Task control of task 1	R/W	0x03F204	258564
Etc.				

12.3 PC Override registers

There are 32 PC override registers, each 20 bits wide. By default, when a task resumes execution its program counter is restored to the next-to-execute instruction within that task at the point where it previously yielded. However if a non-zero instruction address is waiting in the PC override register of a particular task when it resumes execution then program flow will be redirected to that address. The PC override register is automatically zero'd after this occurs so that on the following occasion a yield and resume cycle will proceed as normal.

The FORTH system software uses the PC override register to direct the execution flow of a newly assigned task to the common startup code for a new task. The common startup code proceeds to reset all stack pointers to zero, initialize system user variables and exception stack variables, copy stack parameters passed from the initiating task and then jump to the execution token provided by the initiating task.

Name	Function	R/W	Hex	Dec
PCOVERRIDE0	Task control of task 0	R/W	0x03F400	259072
PCOVERRIDE 1	Task control of task 1	R/W	0x03F404	259076
Etc.				

12.4 Virtual Interrupt registers

There are 32 virtual interrupt registers, each 20 bits wide. If a non-zero instruction address is waiting in the virtual interrupt of a particular task when it resumes execution then (a) the saved value of the program counter that would have been otherwise restored is placed onto the return (and subroutine) stacks, and (b) program flow will be redirected to the virtual interrupt address. The virtual interrupt override register is automatically zero'd after this occurs so that on the following occasion a yield and resume cycle will proceed as normal.

Access to the virtual interrupt functionality is provided in the FORTH system software by the VIRQ word.

Name	Function	R/W	Hex	Dec
VIRQ0	Virtual interrupt of task 0	R/W	0x03F600	259584

VIRQ 1	Virtual interrupt of task 1	R/W	0x03F604	259586
Etc.				

12.5 Hardware interrupts

There is no interaction between hardware multitasking and hardware interrupts. Interrupts are handled by whichever task is executing when the interrupt occurs. However interrupts should not themselves include a PAUSE instruction since if an interrupt does not exit properly through its own RTI instruction then the machine will remain blocked to all further interrupts.

12.6 Alternative multitasking models

A straightforward method for implementing a priority scheduling scheme for multitasking would be to allocate a single task as the scheduling task and write to the task control register such that each other task returned to the scheduling task when yielding execution. The scheduling task would run code to determine which task to run next and then update the lower 5 bits of its own task control register accordingly