The N.I.G.E. Machine

# 1. Installation and set-up

## Introduction

The N.I.G.E. Machine is a user-expandable micro-computer system that runs on an FPGA development board and 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 Nexys2 (1200K gate) and Nexys4 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>

## 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:
  + Nexys2 (1200K gate) v2.0
  + Nexys4 v3.0 (default branch)

## 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
  + Nexys2: E:\N.I.G.E.-Machine\board\_nexys2\_1200\_v2.0.bit
  + Nexys4: E:\N.I.G.E.-Machine\board\_nexys4\_v3.0.bit
* The N.I.G.E. Machine is now running as a FORTH microcomputer
  + See chapter 2 for further guidance

## 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

## Optional SD card interface

* Plug a Digilent SD card PMOD device into port A of either the Nexys2 or Nexys4
  + <http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,401,513&Prod=PMOD-SD>
* Format an SD card as FAT32
  + Full-size SD cards are strongly recommended over micro-SD cards for compatibility
  + Both normal (<2GB) and high capacity (>2GB) SD cards are acceptable
* 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
* See chapter 2 for further guidance on using the SD card in the PMOD slot as a file system

# 2. Using the N.I.G.E. machine as a forth microcomputer

# 3. Customizing the system software

# 4. Customizing the system hardware

# Appendix 1. N.I.G.E. Machine specifications

|  |  |  |
| --- | --- | --- |
| Version | v2.0 | v3.0 |
| Circuit board | Digilent Nexys2 (1200K gate) | Digilent Nexys4 |
| 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 | [] | [] |
| External PSDRAM | 16 MB | 16 MB |
| VGA modes | 640 x 480  800 x 600 | 640 x 480  800 x 600  1024 x 768 |
| Display format | Character graphics (8 x 8 pixel) and pixel graphics | Character graphics (8 x 8 pixel) 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 1. CPU instruction set reference

|  |  |  |  |
| --- | --- | --- | --- |
| **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) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **RSP@** | 1 byte | 0x0B | 1 cycle |
| Load the parameter stack with the current value of the return 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 | | ( -- RSP) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **PSP!** | 1 byte | 0x0C | 1 cycle |
| Save the top item from the stack as the current parameter stack pointer | | ( PSP --) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **RSP!** | 1 byte | 0x0D | 1 cycle |
| Save the top item from the stack as the current return stack pointer | | ( RSP --) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **+** | 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 | | ( x1 -- x2) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **1-** | 1 byte | 0x12 | 1 cycle |
| Subtract 1 | | ( x1 -- x2) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **2/** | 1 byte | 0x13 | 1 cycle |
| Arithmetic shift right | | ( 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. | | ( 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. | | ( 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=** | 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 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 | | ( x1 -- x2) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **LSR** | 1 byte | 0x26 | 1 cycle |
| Logical shift right | | ( 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, highest part 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, highest part 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 -- u-rem u-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) | | (x1 x2 -- rem 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) |

|  |  |  |  |
| --- | --- | --- | --- |
| **RTS\_TRAP** | 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 unblocks further interrupts | | ( --) | (addr --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **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 2a. N.I.G.E. Machine memory map - v2.0, Nexys2

#### 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 Nexys 2 board | W | 0xF830 | 63536 |
| SWITCHES | 8 switch inputs on the Nexys 2 board | R | 0xF834 | 63540 |
| SPIDATA | SPI data baye. 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 3. N.I.G.E. Machine cross-assembler Appendix 2. N.I.G.E. MACHINE FORTH ANSI COMPLIANCE

The following reference is organized according to the categories of the ANSI FORTH documentation. Status indicates whether the word has been implemented on the N.I.G.E. Machine. Y indicates yes, N no, and that Y\* that the word is implemented by with some limitation or difference as compared with ANSI FORTH.

### 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 |  |
| +LOOP | Y |  |
| , | Y | See also W, M, and $, in N.I.G.E. Machine specific world list |
| - | 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 |  |
| ALIGN | Y | A 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 |  |
| BEGIN | Y |  |
| BL | Y |  |
| C! | Y | See also W! |
| C, | Y | See also W, |
| C@ | Y | See also W@ |
| CELL+ | Y |  |
| CELLS | Y |  |
| CHARS | N | Would be a no operation in the N.I.G.E. Machine FORTH environment. |
| 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 |  |
| 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 | 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

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

### 6.3.3 FACILITY words

|  |  |  |
| --- | --- | --- |
| KEY? | Y | See also KKEY?, SKEY? And SKEY? |

### FILE ACCESS words

|  |  |  |
| --- | --- | --- |
| INCLUDED | Y |  |

### PROGRAMMING TOOLS words

|  |  |  |
| --- | --- | --- |
| .S | Y |  |
| ? | Y |  |
| DUMP | Y |  |
| WORDS | Y |  |
| STATE | Y |  |

### STRING words

|  |  |  |
| --- | --- | --- |
| COMPARE | Y |  |
| SLITERAL | Y | See also CLITERAL |

# APPENDIX 3. N.I.G.E. Machine specific words - user level

#### VGA display

|  |  |  |
| --- | --- | --- |
| BACKGROUND | (x --) | Set the current screen background color to the specified value. The color is specified as a 12 bit value in the form RRRGGGBBB |
| 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 also adjusted accordingly |
| VGA | ( n --) | Sets the VGA mode:  0 - 640 \* 480  1 - 800 \* 600 (Nexys2 default)  2 - 1024 \* 768 (Nexys4 default)  The number of screen rows (ROWS) and columns (COLUMNS) are also adjusted accordingly |
| COLORMODE | ( n --) | Sets the color mode:  0 - 16/16  1 - 128/0 (default) |
| CLS | ( --) | Clear the screen |
| TAB | ( -- addr) | VARIABLE pointing to the current size of tab stops. The default is 3 |

#### RS232 port

|  |  |  |
| --- | --- | --- |
| **Word** | **Stack effect** | **Description** |
| 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 |

#### I/O redirection

|  |  |  |
| --- | --- | --- |
| >REMOTE | ( --) | Redirect FORTH environment output to the RS232 |
| >LOCAL | ( --) | Redirect FORTH environment output to the screen |
| <REMOTE | ( --) | Receive FORTH environment input from the RS232 |
| <LOCAL | ( --) | Receive FORTH environment input from the keyboard |

#### 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 |
| INCLUDE | ( “FILEPATH, --) | Include the file at name and location FILEPATH |

#### System

|  |  |  |
| --- | --- | --- |
| MS | (n --) | Wait for n milliseconds |
| RESET | ( --) | Reset the N.I.G.E. Machine to power on configuration but otherwise preserve memory contents |

#### 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, much 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 |

#### 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. |
| ERROR | ( n --) | Print “ERROR n” and ABORT |

# APPENDIX 4. N.I.G.E. Machine specific words - system level

#### VGA 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 | ( -- addr) | VARIABLE pointing to the current column position of the cursor |
| CSR-Y | ( -- addr) | VARIABLE pointing to 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 | ( -- addr) | Byte-length VARIABLE that holds the current number of screen rows. Access with C@ |
| COLS | ( -- addr) | Byte-length VARIABLE that holds the current number of screen columns. Access with C@ |
| SCRSET | ( --) | Set the ROWS and COLS 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 |

#### 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 raw scan code key code |
| 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.) |

#### 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 |

#### Timeout interrupt

|  |  |  |
| --- | --- | --- |
| TIMEOUT | ( n --) | Set the timeout counter for n milliseconds. If n=0, clear the timeout counter. If the timeout counter is allowed to expire without being cleared it will issue QUIT. |