February 2016

*** N.I.G.E. MACHINE ***

FPGA BASED MICROCOMPUTER SYSTEM

GENERAL DESCRIPTION

The N.I.G.E. Machine is a complete micro-computer system hosted within an FPGA development board, the Digilent Nexys 4

FEATURES

- CUSTOM 32 BIT SOFTCORE CPU
  - Stack machine design
  - Microcode based
  - 100 MHz clock frequency
- HIGH INSTRUCTION THROUGHPUT
  - 3 stage CPU pipeline
  - One instruction per clock cycle throughput for most instructions
- DETERMINISTIC EXECUTION
  - All instructions, including conditional branches, jumps and local memory accesses complete in constant time
- HIGH CODE DENSITY
  - Single byte instruction encoding
- VERY LOW INTERRUPT LATENCY
  - Interrupt routine begins 4 clock cycles following the interrupt signal
- FAST BRANCH INSTRUCTIONS
  - Conditional and unconditional branches execute in 2 clock cycles
- HARDWARE SUPPORTED EXCEPTION HANDLING
  - FORTH words CATCH and THROW are atomic machine language instructions
  - Full exception procedure completes in 3 clock cycles via hardware support
- HARDWARE SUPPORTED MULTITASKING
  - Hardware support for 32 tasks
  - Full task switch completes in 2 clock cycles via hardware support
  - Configurable task order
  - 4 KiB private task memory areas
  - Virtual interrupts
- LOCAL AND EXTERNAL MEMORY
  - 128 KiB local FPGA memory
  - 32 bit memory databus
  - Fetch/store longword, word or byte
  - Non-aligned fetch/store supported
  - 16 MiB external dynamic RAM
- FORTH SYSTEM SOFTWARE
  - ANSI FORTH (with minor exceptions)
  - Interpretive and compiled environment
  - No external tool chain required
- FULL SET OF BUILT-IN PERIPHERALS
  - VGA output (640*480 to 1920*1080)
  - 256 on screen colors , 12 bit color range
  - User-definable character based graphics
  - RS232, keyboard, SPI, P-MOD ports
- SD CARD STORAGE
  - Onboard microSD card slot
  - FAT file system provided in FORTH
- EXTENDABLE DESIGN
  - User hardware interfaces via memory-mapped registers and interrupts
  - AXI4 format external memory interface
  - PC based cross assembler for the FORTH system software included
- OPEN SOURCE VHDL DEIGN FILES
  - Dual GPL/commercial license

APPLICATIONS

- Rapid prototyping of experimental scientific apparatus
- Projects involving the dual development of custom hardware and custom software
- Any microcomputer application

CONTENTS

N.I.G.E. Machine copyright, license and disclaimer

The N.I.G.E machine, its design and its source code are Copyright (C) 2012-2016 by Andrew Read and dual licensed. (1) For commercial or proprietary use you must obtain a commercial license agreement with Andrew Richard Read (andrew81244@outlook.com) (2) You can redistribute the N.I.G.E. Machine, its design and its source code and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. The N.I.G.E Machine is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this repository. If not, see http://www.gnu.org/licenses

1. OVERVIEW

At the hardware level N.I.G.E. Machine is organized as a series of modules centered on a 32 bit custom softcore CPU (Figure 1. The N.I.G.E. Machine system diagram).

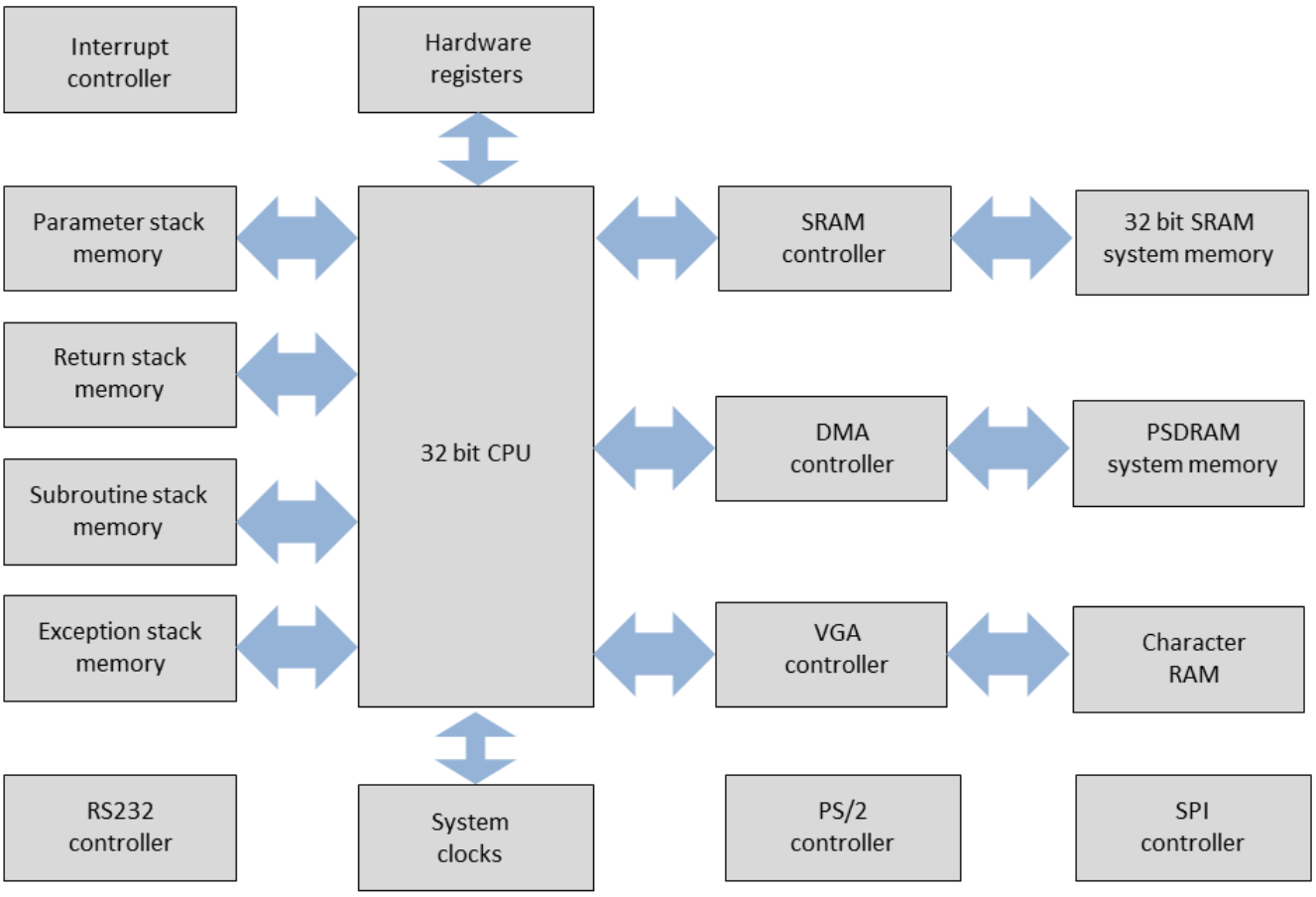

FIGURE 1. THE N.I.G.E. MACHINE SYSTEM DIAGRAM

The CPU interfaces with a number of distinct blocks of memory via separate databuses. Local memory ("SRAM") is comprised of BLOCK RAM units within the FPGA. External memory ("PSDRAM") is comprised of a separate dynamic RAM chip located outside of the FPGA on the Nexys 4 development board. These memories are accessed via separate memory controllers that provide specific functionality to the N.I.G.E. Machine. In addition, there are four memory blocks that are organized as stacks (the parameter stack, the return stack, the subroutine stack and the return stack).

The CPU also interfaces with a hardware register module fabricated from FPGA logic elements. The module provides a set of registers that are memory mapped to the CPU address space. These registers serve a number of functions such as input/output ports and system control settings.

The interrupt controller is responsible for accepting and prioritizing external interrupt request and passing them to the CPU. The system clocks module provides a variety of clock signals to other modules, including the main 100 MHz system clock. The other input/output modules are the RS232 controller, the PS/2 keyboard controller, the SPI controller, and the VGA controller.

1.1. SOFTCORE CPU

The softcore CPU is a 32 bit stack machine optimized for the FORTH programming language (many of the machine language instructions provided by the CPU are FORTH language primitives). The CPU has been designed with a number of features to enhance its suitability for embedded applications.

High instruction throughput

The CPU features a three-stage execution pipeline that delivers throughput of one instruction per clock cycles for most instructions.

Deterministic execution

All instructions, including conditional branches and local memory accesses, execute in a non-varying number of cycles. In addition the execution pipeline has been designed so that there are no conflict states that could result in missed cycles. This provides deterministic execution and enables periodic signals to be generated without jitter.

High code density

The CPU uses microcode rather than a hardwired decoder. This enables a relatively large number of control lines to be driven by a relatively small opcode. As a result all machine language instructions are encoded in a single byte, excluding any literal data.

Fast interrupt response time

Since the CPU is a stack machine there is no requirement to save and restore a register when entering or leaving interrupt service routines. The N.I.G.E. Machine's typical interrupt response time is only 4 clock cycles: 2 cycles to branch to the interrupt vector table plus by 2 cycles to branch through the vector to the interrupt routine itself.

Fast branch performance

On the N.I.G.E. Machine conditional and unconditional branches (BEQ and BRA) execute in 2 clock cycles.

Fast exception handling

The N.I.G.E. Machine includes dedicated hardware stacks for subroutines and exception frames and implements CATCH and THROW as atomic machine language instructions that execute in 2 and 3 cycles respectively. The subroutine and exception stacks are hidden from the FORTH environment.

Fast multitasking

The N.I.G.E. Machine features hardware multitasking. 32 tasks are available. Full task switches execute in 2 clock cycles. Both cooperative and pre-emptive multitasking modes are supported.

1.2. LOCAL MEMORY ("SRAM")

Local memory comprises 128KiB of general memory space available to the CPU. This memory space is byte addressed. It may contain both code and data without restriction. The CPU stores words and longwords in big endian format. SRAM memory space has been optimized for the FORTH language and the needs of embedded systems as follows:

Fast memory access

SRAM is implemented with FPGA BLOCK RAM. All load and store instructions to/from SRAM execute deterministically in 2 clock cycles

Flexible memory access

The databus between the CPU and SRAM is 32 bits wide. The CPU features separate machine language instructions for the load/store of byte, word and long word data.

Misaligned access is supported via the SRAM controller module: longword and word data may be written to or read from any byte address with no clock cycle penalty.

1.3. EXTERNAL MEMORY ("PSDRAM")

External memory comprises 16MiB of storage within a separate pseudo-static dynamic RAM (PSDRAM) module on the Nexys 4 development board. The CPU is able to load/store longwords, words and bytes to/from PSDRAM using the same machine language instructions as SRAM. However the CPU is unable to execute code from PSDRAM.

The PSDRAM memory space is byte addressed but word and longword access must be aligned. The CPU stores words and longwords in big endian format. CPU access to PSDRAM is mediated by the DMA controller that also mediates access by the VGA controller to the screen buffer. As a result of this arrangement CPU access to PSDRAM is not deterministic.

1.4. HARDWARE REGISTERS

The hardware register module provides access to all of the memory mapped I/O registers and registers that control the system configuration. Registers may be from 1 to 32 bits in width, but are all longword aligned. Registers may be readable, writable or both. Execution of a write instruction addressed to certain registers (e.g. RS232DOUT) will trigger the relevant hardware modules to function.

Additional, user-defined registers may be added in order to interface bespoke VHDL components with the N.I.G.E. Machine.

1.5. INTERRUPTS

The interrupt module manages system interrupts. 15 interrupt request lines are available of which 4 are used for built in functions (RS232 RDA, RS232 TBE, PS2, millisecond tick). The remaining 11 may be user-defined in order to interface bespoke VHDL components with the N.I.G.E. Machine.

Interrupts on the N.I.G.E. Machine are prioritized: if two interrupt requests are signaled on the same clock cycle then the lowest numbered interrupt will have priority.

Interrupts on the N.I.G.E. Machine are blocking. Whilst an interrupt service routine is in progress no further interrupts will be actioned. Blocked interrupts are saved by the interrupt module and actioned in priority order once the blocking interrupt routine has completed. However multiple requests for the same interrupt are not counted – the relevant interrupt will be actioned only once the block has been released.

For each interrupt line, the relevant interrupt service routine is accessed via a user definable branch instruction located within the system interrupt vector table. Due to the capability of the branch instruction, all service routines must be located within the lowest 8 KiB of system memory.

All interrupt lines are software maskable via a memory-mapped,16 bit interrupt mask register.

1.6. VIDEO OUTPUT

The N.I.G.E. Machine has a built-in video controller for direct output of a VGA monitor signal. The VGA signal is software selectable between the following modes

Pixel resolution	640 x 480	800 x600	1024 x 768	1920 x 1080

The graphics model is memory mapped character graphics. The character size is user definable between 8 and 16 pixels in width and height. 256 characters are available. The character bitmaps are stored individually in 8 KiB of user-definable RAM.

The color range of the VGA output is 12 bits, allowing a total range of 4096 colors. Due to the capability of the memory mapped graphics, 256 colors may be on screen at any one time. A user-definable PALLETTE RAM specifies the 12 bit color definitions of each of the 256 on-screen colors.

Two modes of color display are available. Either 256 separately selectable foreground colors for each character plus one background color for the whole screen, or 16 separately selectable foreground colors and 16 separately selectable background colors for each character.

The video controller has the capability to interlace between 0 and 15 blank scan lines between each row of characters. Interlace lines may be adjusted enhance the readability of on screen text or removed altogether for a character based block graphics display.

1.7. SD CARD INTERFACE

The Digilent Nexys 4 development board incorporates a microSD card slot. The N.I.G.E. Machine accesses the SD card via an SPI I/O module. The FORTH system software implements a restricted version of the FAT file system. Files on SD card are compatible with a PC and Microsoft Windows. SD cards must be in FAT32 format (regardless of capacity). Filenames are limited to the 8+3 character length format. Directories are supported. There is no restriction on the type of microSD card - most SD cards are suitable.

1.8. USB (PS/2) KEYBOARD

The Digilent Nexys 4 development board incorporates a USB keyboard port with an on-board USB to PS/2 converter chip. The USB keyboard port is compatible with the majority (but not all) USB keyboards. The N.I.G.E. Machine accesses the keyboard via PS/2 signals.

1.9. FORTH SYSTEM SOFTWARE

The N.I.G.E. Machine provides a 32 bit, multitasking FORTH environment at power-on. FORTH is an interpreted and compiled computer language with more than 45 years of use in embedded applications. The N.I.G.E. Machine FORTH environment is ANSI FORTH compatible with minor limitations. The implementation includes: a standard FORTH programming environment, FAT file system access to a microSD card, a suite of extension word for using the N.I.G.E. Machine facilities, and a built-in editor for text files (including FORTH program files) stored on microSD card.

The FORTH system software is tightly integrated with the N.I.G.E. Machine softcore CPU. Many CPU instructions are in fact FORTH primitives. In addition, specialized hardware extensions within the softcore CPU have been developed to enhance the performance and reliability of the FORTH environment. In particular, exception handling and task switching are handled at the hardware level as atomic CPU instructions.

1.10. PMOD EXPANSION PORTS

The Diligent Nexys 4 board incorporates 5 expansion ports ("PMODS") that may be used for connecting off-the-shelf or custom made accessories such as transducers, I/O devices or special hardware modules such as GPS receivers.

2. SUMMARY SPECIFICATIONS

N.I.G.E. Machine version	**v4.0**
Circuit board	Digilent Nexys 4
FPGA family	Xilinx Artix-7
CPU format	32 bit stack machine
CPU pipeline	3 stage
CPU throughput	1 instruction per clock-cycle for most instructions
CPU embedded control optimizations	Deterministic execution CPU Low latency interrupts PC branch in 2 clock cycles
CPU FORTH hardware extensions	Hardware exception handling Hardware multitasking
System clock frequency	100 MHz
FPGA system memory	128 KiB
External PSDRAM memory	16 MiB
Interrupts	Yes, with user-expandable IRQ lines
VGA modes	640 x 480 800 x 600 1024 x 768 1920 x 1080
Display format	Character graphics (16 x 16 pixel, adjustable)
Display color depth	256 colors on screen from a 12-bit palette of 4096 colors
Other input/output ports	USB keyboard RS232 microSD card interface PMOD ports for user expansion

3. CPU INSTRUCTION SET

3.1. INSTRUCTION SIZE AND ENCODING

The default instruction size is a single byte encoded as follows: bit 7 identifies whether the instruction is a branch or an ordinary instruction. If the instruction is a branch then bit 6 specifies if the branch is conditional or unconditional. For branch instructions bits 5 – 0 of the instruction are read as the high part (bits 13 – 8) of the branch address, with a following byte holding the low part (bits 7 – 0) of the branch address.

If the instruction is not a branch then bit 6 specifies whether a return from subroutine is to be taken along with the execution of the instruction ("compound RTS"). Bits 5 – 0 are read as an integer that identifies the instruction in question (“opcode”).

Bit 7	Bit 6	Bits 5 - 0	Interpretation
1	1	High part of branch address	Unconditional branch (BRA)
1	0		Conditional branch (BEQ)
0	1	Opcode	Ordinary instruction with return from subroutine
0	0		Ordinary instruction

FIGURE 2: INSTRUCTION ENCODING

Where literal data is required as part of an instruction it follows in the succeeding bytes in big-endian format (i.e. highest data byte first)

Instruction	Byte 1 (bits 5 -0)	Byte 2	Byte 3	Byte 4	Byte 5
Branch	14 bit branch address		-	-	-
Load.L	Opcode	32-bit literal			
Load.W	Opcode	16-bit literal		-	-
Load.B	Opcode	8-bit literal	-	-	-
JSL	Opcode	24-bit subroutine address			-

FIGURE 3: MULTI-BYTE INSTRUCTIONS WITH LITERAL DATA

3.2. INSTRUCTION SET SUMMARY

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 / 4 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 (4 cycles)		(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)	(--)

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)	(--)

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)	(--)

=	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.		(-- 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, high word is top of stack		(x1 x2 -- d3)	(--)

MULTU	1 byte	0x2A	5 cycles
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	2 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		(--)	(--)

4. SYSTEM MEMORY MAP

Overall memory map

Region	Implementation	Size	Bottom	Top
System memory	FPGA block RAM	As below	0x000000	0x03F7FF
Hardware registers	FPGA logic	As below	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	0x03AFFF
Character RAM	8 KB	0x03B000	0x03CFFF
Palette RAM	2 KB	0x03D000	0x03D7FF
Subroutine and exception local variables	2 KB	0x03D800	0x03DFFF
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	R/W	0x03F838	260152

	triggers the SPI transmit/receive cycle			
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 less 1	R/W	0x03F850	260176
CHARHEIGHT	Height of each character in pixels less 1	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

5. VGA OUTPUT

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

5.1. DISPLAY ORGANIZATION

The N.I.G.E. Machine display is character graphics based. There are 256 different character codes and each character is between 8 and 16 pixels high and between 8 and 16 pixels wide.

In the default character set 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 8KB CHARACTER RAM that resides within the system memory 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.

5.2. INTERLACE MODE

The N.I.G.E. Machine VGA display is able to interlace between 0 and 15 additional scan lines between each row of character graphics to assist the readability of text.

5.3. 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 (no interlace rows)	Character resolution (2 interlace rows)
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

5.4. SCREEN BUFFER

The display screen buffer is held in PSDRAM memory. The location of the start of the screen buffer is soft-programmable

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.

5.5. 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.

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.

The VGA output 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 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.

5.6. 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	‘’Amiga’’ blue	0x05A	148	100%	33%
15	Purple	0x808	300	100%	25%
16-255	various				

6. IMPLEMENTATION OF ANSI FORTH WORDS

Thus section 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.

6.1. ANSI CORE WORDS IMPLEMENTED ACCORDING TO FORTH STANDARD 200X

!	>NUMBER	ELSE	RECURSE
‘	>R	EMIT	REPEAT
(	?DUP	EVALUATE	ROT
*	@	EXECUTE	RSHIFT
+	ABORT	EXIT	S”
+!	ABS	FILL	S>D
+LOOP	ACCEPT	FIND	SIGN
,	ALIGN	HERE	SM/REM
-	ALIGNED	HOLD	SOURCE
.	ALLOT	I	SPACE
.”	AND	IF	SPACES
/	BASE	IMMEDIATE	STATE
/MOD	BEGIN	INVERT	SWAP
0<	BL	J	THEN
0=	C!	KEY	U.
1+	C,	LEAVE	U<
1-	C@	LOOP	UM*
2*	CELL+	LSHIFT	UNLOOP
2/	CELLS	M*	UNTIL
2DROP	CHARS	MAX	VARIABLE
2DUP	CONSTANT	MIN	WHILE
2OVER	COUNT	MOD	WORD
2SWAP	CR	MOVE	XOR
:	CREATE	NEGATE	[
;	DECIMAL	OR	[‘]
<	DEPTH	OVER	[CHAR]
<#	DO	POSTPONE	]
=	DOES>	QUIT	
>	DROP	R>	
>IN	DUP	R@	

6.2. ANSI CORE WORDS IMPLEMENTED WITH MODIFICATIONS

*/	The intermediate value is single (32-bit) precision only
*/MOD	The intermediate value is single (32-bit) precision only

6.3. ANSI CORE WORDS NOT IMPLEMENTED

#	Use U# instead. Division with a 64-bit dividend is not supported by hardware
#>	Use U#> instead.
#S	Use U#S instead.
2!	Little-endian format inappropriate on a big-endian processor
2@	Little-endian format inappropriate on a big-endian processor
>BODY	Would be a no operation
ABORT”	Will not be implemented for space and efficiency reasons
ENVIRONMENT?	Will not be implemented for space and efficiency reasons
FM/M	Will not be implemented for space and efficiency reasons
S>D	Equivalent to FALSE on the N.I.G.E. Machine.
SM/REM	Division with a 64-bit dividend is not supported by hardware
UM/MOD	Division with a 64-bit dividend is not supported by hardware

6.4. CORE EXTENSION WORDS IMPLEMENTED

.(	CASE	MARKER	TRUE
.R	COMPILE,	NIP	U.R
0<>	DEFER	OF	U>
0>	ENDCASE	PAD	UNUSED
<>	ENDOF	PARSE	WITHIN
?DO	FALSE	PICK	\
AGAIN	HEX	RESTORE-INPUT	
BUFFER:	INTERPRET	SAVE-INPUT	
C”	IS		

6.5. DOUBLE-NUMBER WORDS IMPLEMENTED

D+
D-

6.6. EXCEPTION WORDS IMPLEMENTED

CATCH
THROW

6.7. FACILITY WORDS IMPLEMENTED

KEY?
MS

6.8. STRING WORDS IMPLEMENTED

COMPARE
SLITERAL

6.9. FILE ACCESS WORDS IMPLEMENTED

INCLUDE these words from SYSTEM.F

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

6.10. MEMORY WORDS IMPLEMENTED

INCLUDE these words from SYSTEM.F

ALLOCATE
FREE
RESIZE

6.11. PROGRAMMING TOOLS WORDS IMPLEMENTED

.S
?
DUMP
SEE *(INCLUDE this word from TOOLS.F)*
WORDS
STATE

6.12. SEARCH-ORDER WORDS IMPLEMENTED

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

7. SYSTEM FEATURES AND RELATED FORTH WORDS

This section 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. This list focuses on commonly used words. Section 8 lists further N.I.G.E. Machine specific words more relevant to customization of the system software.

7.1. 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

7.2. VGA OUTPUT

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	(n --)	Sets the number of interlace scanlines between each row of characters. The number of screen rows (ROWS) is adjusted accordingly. The minimum number of interlace rows is 0 and the maximum is 15. The default is 2 interlace rows.
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.

7.3. SD CARD AND FAT FILE SYSTEM

The N.I.G,E. Machine reads and writes filenames in 8+3 format only (e.g. “FILENAME.EXT”). 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.

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

INCLUDE the following words from SYSTEM.F

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

7.4. VISUAL PREFERENCES

INCLUDE the following words from PREFS.F

SETFONT	“FILENAME”	Load and extract an Amiga bitmap font file into character memory. The font should be fixed width and not more than 16 pixels in width.
AMIGACOLORS	(--)	Set the display colors to match the blue, white, orange and black format of AmigaDOS 1.3
DEFULTFONT	(--)	Restore the default system font
DEFAULTCOLORS	(--)	Restore the default system colors

7.4. EDITOR

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.

INCLUDE the following words from EDITOR.F

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

7.5. LOCAL VARIABLES

Local variable syntax is based on VFX. A maximum of 10 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

7.6. 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 at completion**
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 yield 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.

7.7. 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.

7.8. MEMORY

INCLUDE the following words from 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

7.9. 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

7.10. PROGRAMMING TOOLS

INCLUDE the following words from 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

7.11. OTHER 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

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.

8.1. 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

8.2. 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

8.3. ADDITIONAL SD CARD AND FAT FILE SYSTEM

INCLUDE these words from 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)	Allocate memory, open a file structure and load data
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

8.4. 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.)

9. INTERRUPTS

9.1. INTERRUPT VECTORS

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

9.2. 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.

9.3. INTERRUPT VECTOR TABLE

The default CPU interrupt vector table is as follows:

Address	**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. This register is masked by default.
0xC0 -	various	Available for user expansion

10. EXCEPTION HANDLING

The N.I.G.E. Machine implements the ANSI FORTH words CATCH and THROW as atomic machine language instructions.

10.1. THE EXCEPTION STACK

The N.I.G.E. Machine maintains an internal exception stack that is traversed with CATCH and THROW instructions. A segment of the exception stack is memory mapped to fixed addresses in the system memory space. This space may be used to hold global variables that will be atomically restored to their prior values following an exception. For example, with reference to Anton Ertl's "hex.-helper" problem [1]:

```
: hex.-helper
      HEX   \ the variable BASE is located on the exception stack
      U.
;

: hex. ( n --, print the top of stack in hexadecimal)
      ['] hex.-helper catch throw
      \ BASE will be restored to its prior value regardless of any occurrence
;
```

In addition to BASE, STDOUT, and STDIN used by the FORTH system software, there are three longword slots on the exception stack available for user definition as follows:

Variable	**Address**
BASE	0x3B080
STDOUT	0x3B084
STDIN	0x3B08C
User definable 1	0X3b094
User definable 2	0X3b098
User definable 3	0X3b09C

[1] M. Anton Ertl, “Cleaning up after yourself”, in EuroFORTH, 2008

10.2. ASSEMBLY LANGUAGE USAGE

There is a slight usage difference for the machine language instruction CATCH as compared with the FORTH word 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, In the case of return via THROW the return address is the instruction immediately following CATCH plus one.

For example:

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

11. MULTITASKING HARDWARE

The N.I.G.E. Machine has hardware multitasking capabilities built into the soft-core CPU. These facilities are made available in the FORTH system software through a set of FORTH words largely in common with PolyFORTH. See section 7 for a description of the multitasking word set.

This section documents the hardware multitasking control registers for the benefit of developers wishing to customize the FORTH system software multitasking model.

11.1. CONTROL SETTINGS 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

11.2 . 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.				

11.3. 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.				

11.4. 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.				

11.5. 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. 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
```

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.

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	
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

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- - ;`

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.

13. 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.