mcx16

# Introduction

mcx16 is a 16-bit microcontroller written in VHDL. It was inspired by PicoBlaze, an 8-bit microcontroller designed by Ken Chapman at Xilinx. mcx16 uses an instruction set and architecture similar to PicoBlaze, so if you've already learned one, using the other should be straight-forward. The mcx16 code is tested on Xilinx FPGAs.

Some highlights of mcx16:

- 16-bit addresses, data, port addresses and ports
- Can use 1 (non-pipelined) or 2 (pipelined) clocks per instruction
- 74MHz (74 MIPS) on a Spartan 3 XC3S200AN-4 non-pipelined, and 119 MHz when pipelined (59 MIPS), using 110/114 slices
  (area optimized, 16 registers, 32 deep stack, 512 word program, distributed stack)
- 107 MHz (107 MIPS) on a Spartan 6 XC6SLX9-2 non-pipelined, and 166 MHz when pipelined (83 MIPS), using 46/41 slices
- Pipelining, register count (1 – 256), stack depth (1 – 65536), and program/stack type (BRAM/distributed) are all user-set generics; program size up to 65536 words
- Cross-platform assembler
- Optional JTAG loader using Kris Chaplin's technique and cross-platform Xilinx tools

 © 2011 Gregory Bredthauer 

# CONTENTS


| | | | | | |
|---|---|---|---|---|---|
| Source Files | 3 | Addresses | 28 | INPUT | 52 |
| Instantiation | 4 | Op-Codes | 29 | OUTPUT | 53 |
| Basic Usage | 5 | Conditionals | 30 | JUMP | 54 |
| Optional Usage | 6 | NOP | 31 | CALL | 55 |
| Architecture | 7 | LOAD | 32 | RETURN | 56 |
| Command Set | 8 | AND | 33 | LOADRETURN | 57 |
| Basic Operation | 9 | OR | 34 | TEST | 58 |
| Reset | 10 | XOR | 35 | TESTCY | 59 |
| Clocking | 11 | ADD | 36 | COMPARE | 60 |
| Clock Enable | 12 | ADDCY | 37 | COMPARECY | 61 |
| Input Port | 13 | SUB | 38 | SETCY | 62 |
| Output Port | 14 | SUBCY | 39 | CLEARCY | 63 |
| Single Cycle I/O | 15 | SL0 | 40 | STRING | 64 |
| Two Cycle I/O | 16 | SL1 | 41 | WSTRING | 65 |
| IN_BIT | 17 | SLX | 42 | Escape Codes | 66 |
| LOAD Ports | 18 | SLA | 43 | mcx16loader | 67 |
| Unused Inputs | 19 | SLB | 44 | JTAG Generics | 68 |
| Generic Parameters | 20 | RL | 45 | JTAG Loading | 69 |
| Stack | 21 | SR0 | 46 | JTAG Chain | 70 |
| Assembler | 22 | SR1 | 47 | JTAG Examples | 71 |
| Assembly Language | 24 | SRX | 48 | mcx16rom | 72 |
| Instructions | 25 | SRA | 49 | mcx16uart | 74 |
| Registers | 26 | SRB | 50 | Conclusion | 75 |
| Constants | 27 | RR | 51 | | |


 © 2011 Gregory Bredthauer 

# Source Files

- mcx16.vhd: Microcontroller VHDL
- mcx16rom.vhd: ROM VHDL template
- mcx16loader.vhd: JTAG loader VHDL
- mcx16uart.vhd: UART VHDL
- mcx16util.vhd: Utility library VHDL
- mcx16asm.cpp: Assembler
- mcx16asm.pro: Qt project file for assembler

The above are all distributed under the simplified BSD license. The assembler is built on the Qt GUI Toolkit, which is available under the LGPL.


© 2011 Gregory Bredthauer

mcx16

# INSTANTIATION

## mcx16 declaration:

```
component mcx16 is
	generic (
		REGISTER_COUNT: integer;
		STACK_DEPTH: integer;
		SINGLE_CYCLE: boolean;
		STACK_BRAM: boolean
	);
	port (
		RST: in std_logic;
		CLK: in std_logic;
		CE: in std_logic;
		READ_STROBE: out std_logic;
		IN_PORT_ID: out std_logic_vector(15 downto 0);
		IN_PORT: in std_logic_vector(15 downto 0);
		WRITE_STROBE: out std_logic;
		OUT_PORT_ID: out std_logic_vector(15 downto 0);
		OUT_PORT: out std_logic_vector(15 downto 0);
		IN_BIT: in std_logic;
		ROM_CE: out std_logic;
		ADDR: out std_logic_vector(15 downto 0);
		CODE: in std_logic_vector(35 downto 0)
	);
end component;
```

## mcx16 instance:

```
inst_mcx16: mcx16
	generic map (
		REGISTER_COUNT => 16,
		STACK_DEPTH => 32,
		SINGLE_CYCLE => true,
		STACK_BRAM => false
	)
	port map (
		RST => RST,
		CLK => CLK,
		CE => CE,
		READ_STROBE => READ_STROBE,
		IN_PORT_ID => IN_PORT_ID,
		IN_PORT => IN_PORT,
		WRITE_STROBE => WRITE_STROBE,
		OUT_PORT_ID => OUT_PORT_ID,
		OUT_PORT => OUT_PORT,
		IN_BIT => IN_BIT,
		ROM_CE => ROM_CE,
		ADDR => ADDR,
		CODE => CODE
	);
```


© 2011 Gregory Bredthauer

# Basic Usage

- You write an mcx16 assembly program ("myprogram.txt").
- You assemble it, and output an mcx16rom module initialized with your code:
  - `mcx16asm myprogram.txt -o`
- You add mcx16.vhd, mcx16util.vhd and myprogram.vhd to your FPGA project, and connect the mcx16 and mcx16rom signals to the rest of your design appropriately.
- Run it on your FPGA!

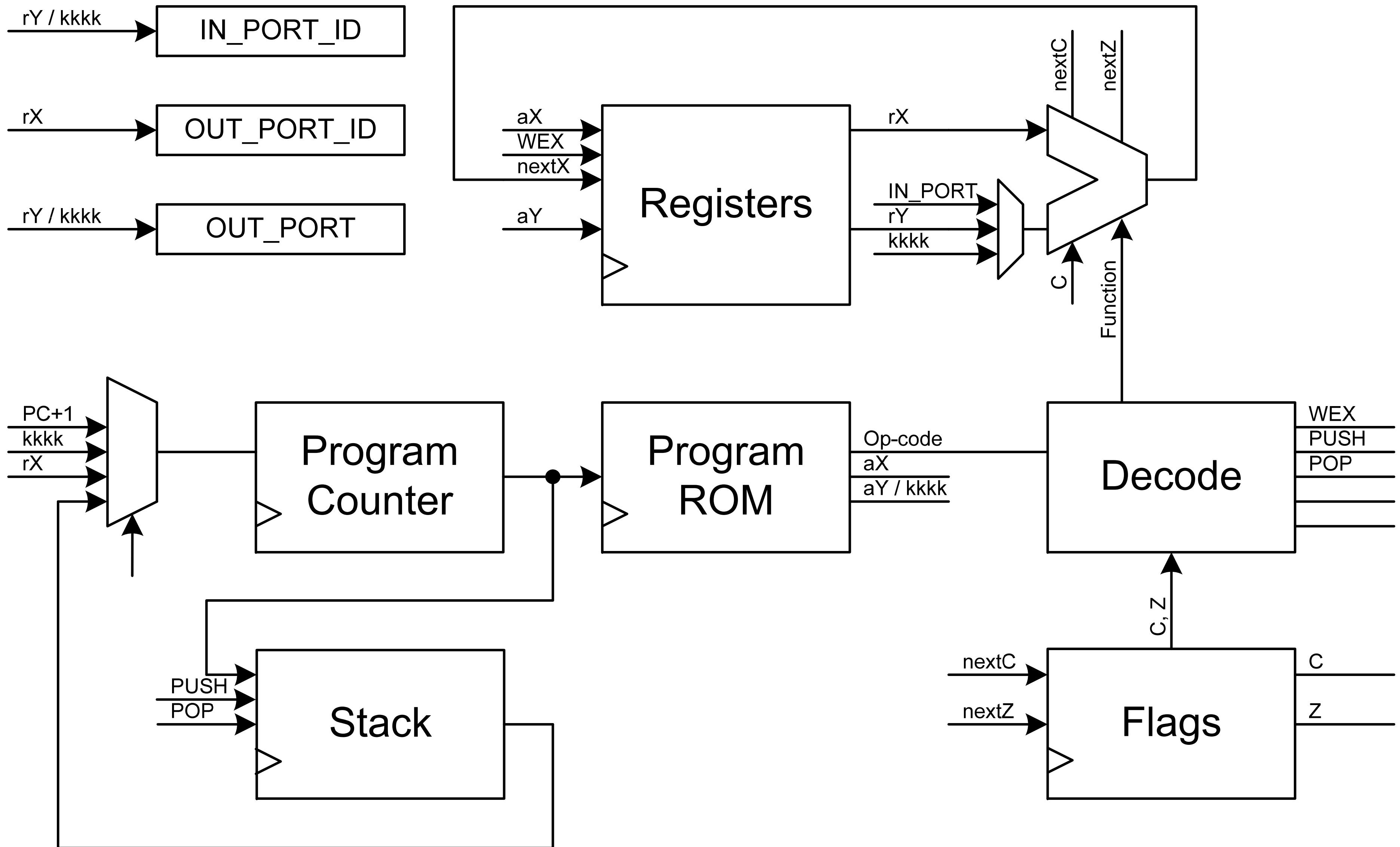
 © 2011 Gregory Bredthauer 

# Optional Usage

- Add mcx16loader.vhd to your project, and connect it to the LOAD ports on mcx16rom and RST port on mcx16.
- After you've compiled your design and your FPGA is up and running with an embedded mcx16 (and is connected via JTAG to your system), load a new program into mcx16:
  - `mcx16asm mybetterprogram.txt -l`
- This requires running in an environment with access to iMPACT (ISE Design Suite Command Prompt)

 © 2011 Gregory Bredthauer 

# Architecture




 © 2011 Gregory Bredthauer 

mcx16

# Command Set

<u>Logical</u>
LOAD rX, rY / kkkk
AND rX, rY / kkkk
OR rX, rY / kkkk
XOR rX, rY / kkkk

<u>Arithmetic</u>
ADD rX, rY / kkkk
ADDCY rX, rY / kkkk
SUB rX, rY / kkkk
SUBCY rX, rY / kkkk

<u>Shifting</u>
SL0/SL1/SLX/SLA/SLB/RL rX
SR0/SR1/SRX/SRA/SRB/RR rX

<u>I/O</u>
INPUT rX, rY / kkkk
OUTPUT rX, rY / kkkk

<u>Branching</u>
JUMP rX / kkkk
CALL rX / kkkk
RETURN
LOADRETURN rX, kkkk

<u>Flags and Comparisons</u>
TEST rX, rY / kkkk
TESTCY rX, rY / kkkk
COMPARE rX, rY / kkkk
COMPARECY rX, rY / kkkk
SETCY
CLEARCY

<u>Conditionals</u>
All commands can be prefixed with:
IF C / NC / Z / NZ / B / NB
    If the condition specified is true, the command is executed. If not, it behaves as a NOP.

<u>Assembler Macros</u>
STRING “abcdef\n”
WSTRING “abcdef\n”
    Translated into a sequence of 8 or 16-bit LOADRETURNs.
NOP
    Translated into LOAD r0, r0


© 2011 Gregory Bredthauer

# Basic Operation

- mcx16 is fundamentally a state machine. Its present state is defined by the contents of the program counter, registers, stack, and flags. The next state is a function of the current state, the current instruction word (from the ROM), and the input port/bit.
- Instruction words are 36-bits wide, and are fetched from a ROM. The ROM contents are filled by the assembler, either at compile time in VHDL or at run time via a JTAG interface.


© 2011 Gregory Bredthauer

# Reset

- Setting RST high resets the program counter, stack pointer, and flags. READ_STROBE and WRITE_STROBE are also driven low. Normal operation begins a few clock cycles after RST goes low.
- The RST input is optional, since mcx16 self-resets after FPGA configuration is complete. Tie RST to zero if resets during normal operation are unnecessary.

 © 2011 Gregory Bredthauer 

# CLOCKING

- mcx16 is a fully synchronous design. All registers are updated on the rising edge of the clock.
- mcx16 can operate single-cycle by setting the top level generic SINGLE_CYCLE to true. In this mode, a single instruction is executed every clock cycle.
- To support higher system clock frequencies, set SINGLE_CYCLE to false. This causes pipeline registers to be inserted in the design. In this mode, a single instruction is executed every two clock cycles.


© 2011 Gregory Bredthauer

mcx16

# Clock Enable

- The top level CE input can be driven low to pause mcx16 for an arbitrary number of clock cycles. Operation will resume from wherever mcx16 was stopped.
- Use CE to save power when mcx16 is unused, or to delay execution if extra time is needed during input/output operations (to compensate for pipeline stages in input port muxes, for example).


© 2011 Gregory Bredthauer

# INPUT PORT

- The INPUT rX, kkkk and INPUT rX, rY instructions set IN_PORT_ID to kkkk or rY, respectively, and set READ_STROBE to 1. Your logic must decode IN_PORT_ID and connect the requested signal to IN_PORT. The data at IN_PORT must be valid before the rising clock edge while READ_STROBE is high.
- For two-cycle operation, IN_PORT_ID will be valid for both cycles, while READ_STROBE will only be high for the second cycle.

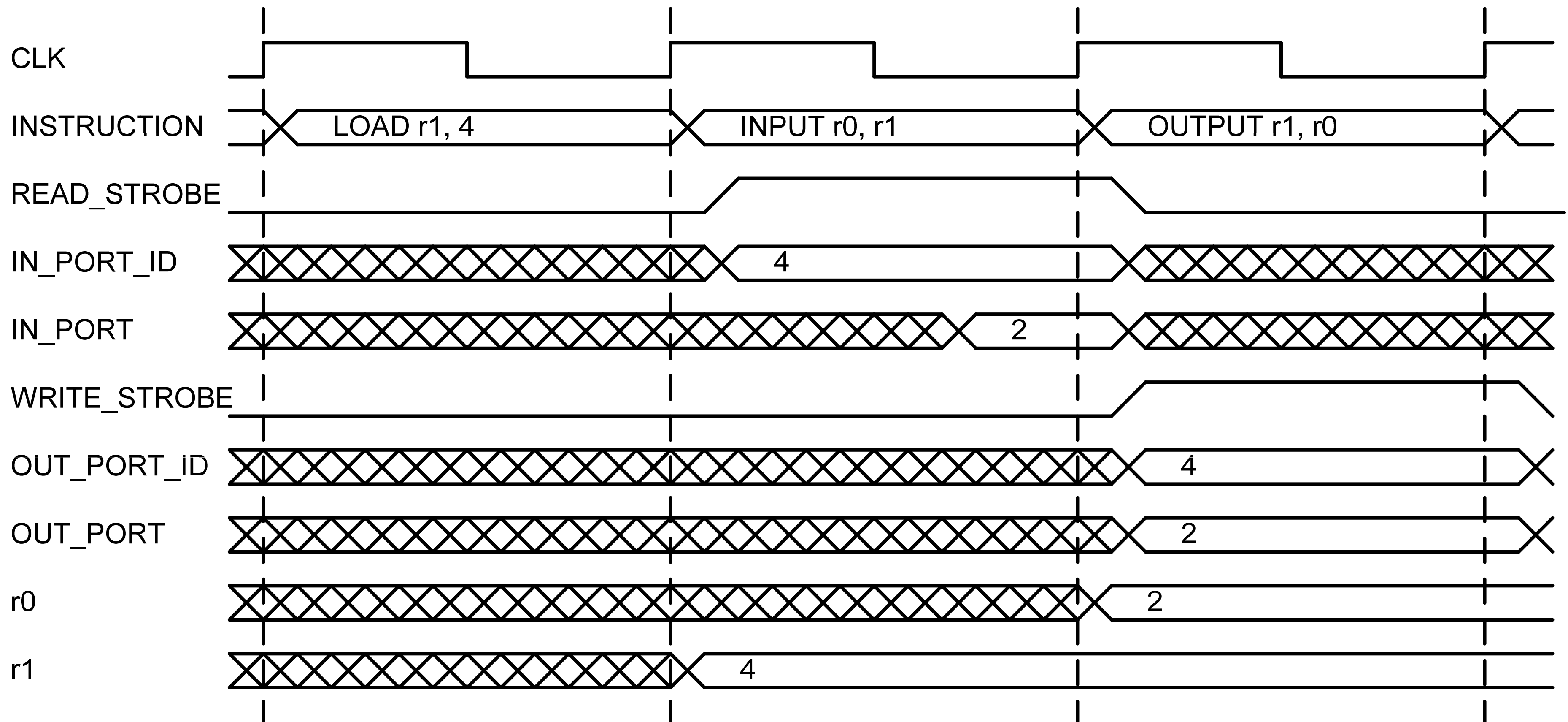

© 2011 Gregory Bredthauer

# Output Port

- The OUTPUT rX, kkkk and OUTPUT rX, rY instructions set OUT_PORT_ID to kkkk or rY, respectively, and set WRITE_STROBE to 1. Your logic must decode OUT_PORT_ID and connect OUT_PORT to the desired destination (typically a flip-flop with its CE controlled by OUT_PORT_ID).
- For two-cycle operation, OUT_PORT_ID will be valid for both cycles, while WRITE_STROBE will only be high for the second cycle.

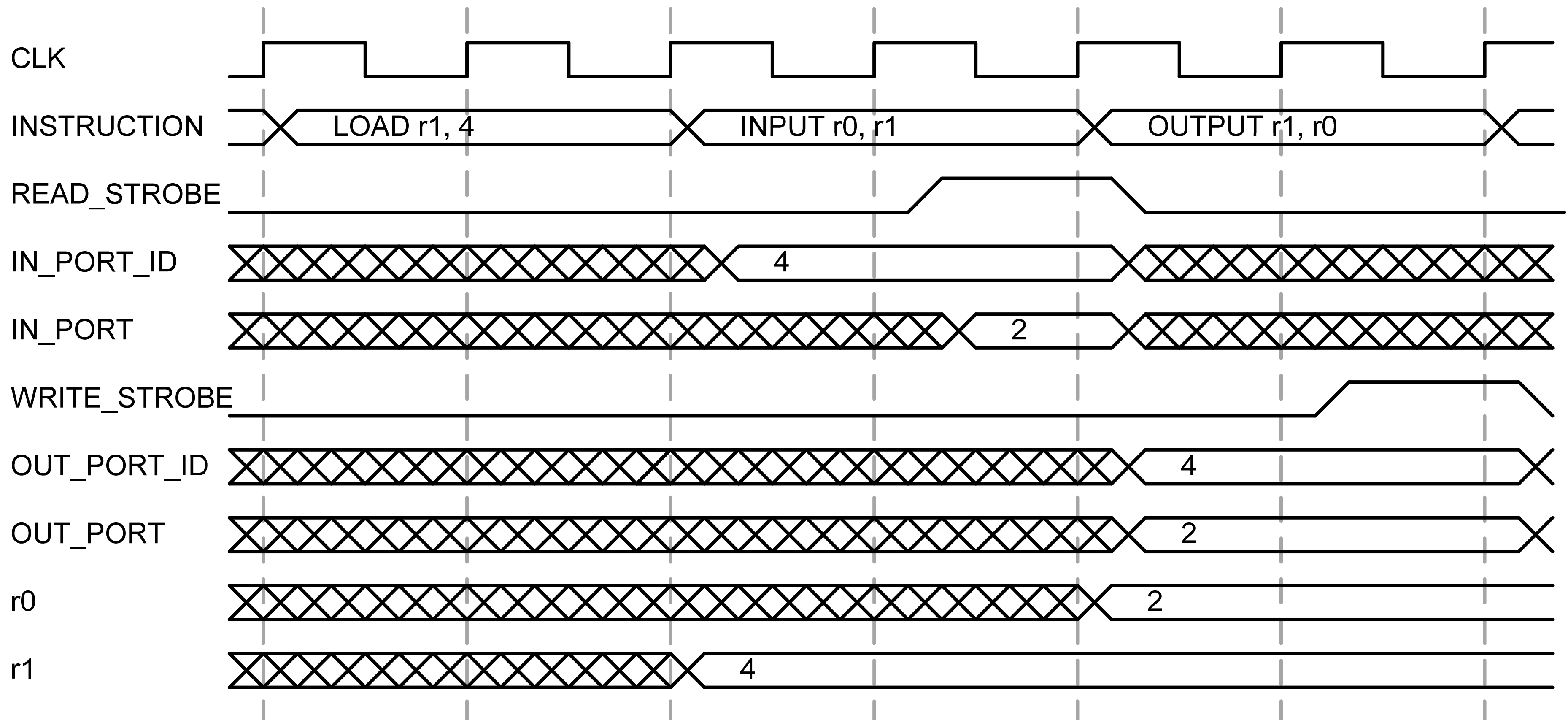
 © 2011 Gregory Bredthauer 

# Single Cycle I/O




 © 2011 Gregory Bredthauer 

# Two Cycle I/O




 © 2011 Gregory Bredthauer 

# IN_BIT

- mcx16 provides an IN_BIT input. This input can be used in conditional statements (IF B JUMP bit_set) or shift instructions (SLB serial_to_parallel_reg).
- If there's only a single bit in your design that you'll use with this input, tie it directly to IN_BIT (unless pipeline registers are needed for timing).
- For selecting among multiple bits, use an OUTPUT command to write to a register that controls a mux for the IN_BIT input.

 © 2011 Gregory Bredthauer 

# LOAD PORTS

- The mcx16rom program can be modified using the LOADCLK / LOAD / LOADADDR / LOADDATA inputs. LOADCLK can be asynchronous to CLK. The instruction at LOADADDR will change to LOADDATA on any LOADCLK rising edge when LOAD is high.
- To avoid unpredictable operation, RST should be held high during ROM modifications.
- mcx16loader can be connected to the LOAD ports to allow JTAG updating of the program ROM.


© 2011 Gregory Bredthauer

# UNUSED INPUTS

- Tie unused input ports (IN_BIT, IN_PORT, LOADs) to zero. This will typically cause warnings about unused signals during implementation, but the design will usually come out smaller and faster.
- If the LOAD ports are unused, tie LOADCLK to CLK (and the remaining LOAD inputs to zero) to avoid BRAM implementation errors.

 © 2011 Gregory Bredthauer 

# Generic Parameters

- Use the top-level generics REGISTER_COUNT, SINGLE_CYCLE, STACK_DEPTH and STACK_BRAM to adjust the capabilities of an mcx16.
- Certain values fit most naturally into a particular FPGA. For example, program lengths up to 512 words fit into 1 BRAM in Spartan FPGAs, while 1024 words fit into 1 BRAM in Virtex FPGAs. 16 registers fit best in a Spartan 3, while 32 registers fit best in Spartan 6 and Virtex FPGAs. Feel free to experiment!

 © 2011 Gregory Bredthauer 

# Stack

- The number of CALL instructions that can be nested is controlled by the STACK_DEPTH generic. A depth of 32 is best for Spartan 3 FPGAs, and 64 for Spartan 6 and Virtex devices.
- To save logic, the stack can use a BRAM by setting STACK_BRAM to true. This is only possible when SINGLE_CYCLE is false.
- Together, the mcx16 generics allow you to trade size, speed, and functionality to fit your needs.

 © 2011 Gregory Bredthauer 

mcx16

# ASSEMBLER

- The mcx16asm assembler is provided as C++ source code and as a Windows executable. To compile it yourself, first download and install the Qt SDK for your platform:
  - http://qt.nokia.com/downloads
- Then, from a command prompt with Qt in the path, compile mcx16asm:
  - `qmake`
  - `make`

# ASSEMBLER

- For a list of assembler options, execute:
  - `mcx16asm -h`
- To check the syntax of a program, execute:
  - `mcx16asm myprogram.txt`
- To assemble a program and generate an mcx16rom module with the program embedded:
  - `mcx16asm myprogram.txt -o`
- Note that the mcx16rom.vhd template file must be available to use the -o option. Use the -t option to specify a different template file.

 © 2011 Gregory Bredthauer 

# Assembly Language

- Blank lines are ignored.
- Lines that start with a semicolon are comments, and are ignored.
- Instructions can be followed by a semicolon to have comments on the same line.
- Labels must be alone on a line, and are an alphanumeric string followed by a colon.
- Programs start at address zero.
- The assembler doesn't know how many registers your mcx16 instance has – you are responsible for staying within the hardware limits.


© 2011 Gregory Bredthauer

mcx16

# Instructions

- Instructions are of the form:
  [conditional] instruction [destination][, source]
- As an example,
  `IF Z LOAD r0, 7`
  will load the source value 7 into the destination register r0 if the zero flag is set.
- Registers are referred to as r[hex value], where [hex value] can go from 0 to [number of registers – 1], so r0 to r1F for 32 registers.


© 2011 Gregory Bredthauer

# REGISTERS

- mcx16 has REGISTER_COUNT identical 16-bit general purpose registers.
- You can give a register a more meaningful label with the ALIAS assembler directive:

```
ALIAS r0 MyCounter
ALIAS r0 MyCaptain
LOAD r0, 8            ; Sets r0 to 8
LOAD MyCounter, 8     ; Also sets r0 to 8
LOAD MyCaptain, 8     ; Also also sets r0 to 8
```

- Register aliases, constant names, and labels must all be unique.

 © 2011 Gregory Bredthauer 

# CONSTANTS

- When an instruction uses a constant (“kkkk”), you can specify it in decimal, hexadecimal, ASCII, or as a named constant/label.
- Examples:

```
MyLabel:
MyConstant = 100
LOAD r0, 100              ; Sets r0 to 100
LOAD r0, 0x64             ; Also sets r0 to 100
LOAD r0, ‘d’              ; Also also sets r0 to 100
LOAD r0, ‘\n’             ; Sets r0 to newline (0xA)
LOAD r0, MyLabel          ; Sets r0 to the address of MyLabel
LOAD r0, MyConstant       ; Back to setting r0 to 100
```

- Everything but ASCII/strings are case-insensitive.

 © 2011 Gregory Bredthauer 

# Addresses

- By default, the assembler begins generating instructions at address 0. To change the address of the next assembled instruction, use the command:
  `ADDRESS 100`
- The assembler reports the highest address used, which defines how large the ROM must be.
- JUMP/CALL instructions can use constants, labels, or register values as a destination, which enables the use of jump tables.

 © 2011 Gregory Bredthauer 

# Op-Codes

- Each mcx16 op-code is 36 bits wide.
- Bits 35 – 33 identify conditional instructions.
- Bits 32 – 25 specify the instruction.
- Bit 24 selects rY or kkkk as the source.
- Bits 23 – 16 identify the destination register rX.
- Bits 15 – 8 identify the source register rY.
- Bits 15 – 0 specify the 16 bit constant kkkk, and are also used to select amongst the shift operations.

 © 2011 Gregory Bredthauer 

# Conditionals

- All instructions can be prefixed with a conditional (bits 35 – 33, ccc).
- If the condition is true, the instruction is executed, if false, it behaves as a NOP.

| 35 | 34 | 33 | Condition |
|---|---|---|---|
| 0 | 0 | 0 | Unconditional |
| 0 | 1 | 0 | Z = 1 |
| 0 | 1 | 1 | Z = 0 |
| 1 | 0 | 0 | C = 1 |
| 1 | 0 | 1 | C = 0 |
| 1 | 1 | 0 | IN_BIT = 1 |
| 1 | 1 | 1 | IN_BIT = 0 |

 © 2011 Gregory Bredthauer 

# NOP

NOP

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX unchanged
Z unchanged
C unchanged

PC ← PC + 1
Stack unchanged
WRITE_STROBE ← 0
READ_STROBE ← 0

 © 2011 Gregory Bredthauer 

# LOAD

LOAD rX, rY

| c | c | c | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

LOAD rX, kkkk

| c | c | c | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX ← rY /

rX ← kkkk

Z unchanged

C unchanged

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0


© 2011 Gregory Bredthauer

mcx16

# AND

AND rX, rY

| c | c | c | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | x | x | x | x | x | x | x | x | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

AND rX, kkkk

| c | c | c | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX ← rX and rY /

rX ← rX and kkkk

Z ← 1 if result is 0

C ← 0

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0


© 2011 Gregory Bredthauer

# mcx16 OR

OR rX, rY

| c | c | c | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | x | x | x | x | x | x | x | x | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

OR rX, kkkk

| c | c | c | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX ← rX or rY /

rX ← rX or kkkk

Z ← 1 if result is 0

C ← 0

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0


© 2011 Gregory Bredthauer

mcx16

# XOR

XOR rX, rY

| c | c | c | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | x | x | x | x | x | x | x | x | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

XOR rX, kkkk

| c | c | c | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX ← rX xor rY /

rX ← rX xor kkkk

Z ← 1 if result is 0

C ← 0

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0


© 2011 Gregory Bredthauer

mcx16

# ADD

ADD rX, rY

| c | c | c | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

ADD rX, kkkk

| c | c | c | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX ← rX + rY /

rX ← rX + kkkk

Z ← 1 if result is 0

C ← 1 if result > FFFF

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0

 © 2011 Gregory Bredthauer 

mcx16

# ADDCY

ADDCY rX, rY

| c | c | c | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | x | x | x | x | x | x | x | x | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

ADDCY rX, kkkk

| c | c | c | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX ← rX + rY + C /

rX ← rX + kkkk + C

Z ← 1 if result is 0

and last Z = 1

C ← 1 if result > FFFF

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0

# SUB

SUB rX, rY

| c | c | c | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | x | x | x | x | x | x | x | x | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

SUB rX, kkkk

| c | c | c | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX ← rX - rY /

rX ← rX - kkkk

Z ← 1 if result is 0

C ← 1 if result < 0

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0

 © 2011 Gregory Bredthauer 

# SUBCY

SUBCY rX, rY

| c | c | c | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | x | x | x | x | x | x | x | x | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

SUBCY rX, kkkk

| c | c | c | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX ← rX - rY - C /

rX ← rX - kkkk - C

Z ← 1 if result is 0

and last Z = 1

C ← 1 if result < 0

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0

 © 2011 Gregory Bredthauer 

# SLO

SLO rX

| c | c | c | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX(0) ← 0
rX(15-1) ← rX(14-0)
Z ← 1 if result is 0
C ← rX(15)

PC ← PC + 1
Stack unchanged
WRITE_STROBE ← 0
READ_STROBE ← 0

 © 2011 Gregory Bredthauer 

# SL1

SL1 rX

| c | c | c | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX(0) ← 1
rX(15-1) ← rX(14-0)
Z ← 0
C ← rX(15)

PC ← PC + 1
Stack unchanged
WRITE_STROBE ← 0
READ_STROBE ← 0

 © 2011 Gregory Bredthauer 

# SLX

SLX rX

| c | c | c | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX(0) ← rX(0)

rX(15-1) ← rX(14-0)

Z ← 1 if result is 0

C ← rX(15)

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0

 © 2011 Gregory Bredthauer 

# SLA

SLA rX

| c | c | c | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX(0) ← C
rX(15-1) ← rX(14-0)
Z ← 1 if result is 0
C ← rX(15)

PC ← PC + 1
Stack unchanged
WRITE_STROBE ← 0
READ_STROBE ← 0

 © 2011 Gregory Bredthauer 

mcx16

# SLB

SLB rX

| c | c | c | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX(0) ← IN_BIT
rX(15-1) ← rX(14-0)
Z ← 1 if result is 0
C ← rX(15)

PC ← PC + 1
Stack unchanged
WRITE_STROBE ← 0
READ_STROBE ← 0


© 2011 Gregory Bredthauer

mcx16

# RL

RL rX

| c | c | c | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX(0) ← rX(15)
rX(15-1) ← rX(14-0)
Z ← 1 if result is 0
C ← rX(15)

PC ← PC + 1
Stack unchanged
WRITE_STROBE ← 0
READ_STROBE ← 0

mcx16

# SR0

SR0 rX

| c | c | c | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX(15) ← 0

rX(14-0) ← rX(15-1)

Z ← 1 if result is 0

C ← rX(0)

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0


© 2011 Gregory Bredthauer

# SR1

SR1 rX

| c | c | c | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX(15) ← 1
rX(14-0) ← rX(15-1)
Z ← 0
C ← rX(0)

PC ← PC + 1
Stack unchanged
WRITE_STROBE ← 0
READ_STROBE ← 0

 © 2011 Gregory Bredthauer 

mcx16

# SRX

SRX rX

| c | c | c | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX(15) ← rX(15)

rX(14-0) ← rX(15-1)

Z ← 1 if result is 0

C ← rX(0)

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0


© 2011 Gregory Bredthauer

# SRA

SRA rX

| c | c | c | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX(15) ← C
rX(14-0) ← rX(15-1)
Z ← 1 if result is 0
C ← rX(0)

PC ← PC + 1
Stack unchanged
WRITE_STROBE ← 0
READ_STROBE ← 0

 © 2011 Gregory Bredthauer 

mcx16

# SRB

SRB rX

| c | c | c | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX(15) ← IN_BIT
rX(14-0) ← rX(15-1)
Z ← 1 if result is 0
C ← rX(0)

PC ← PC + 1
Stack unchanged
WRITE_STROBE ← 0
READ_STROBE ← 0


© 2011 Gregory Bredthauer

# RR

RR rX

| c | c | c | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX(15) ← rX(0)
rX(14-0) ← rX(15-1)
Z ← 1 if result is 0
C ← rX(0)

PC ← PC + 1
Stack unchanged
WRITE_STROBE ← 0
READ_STROBE ← 0

 © 2011 Gregory Bredthauer 

# mcx16 INPUT

INPUT rX, rY

| c | c | c | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

INPUT rX, kkkk

| c | c | c | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

IN_PORT_ID ← rY /

IN_PORT_ID ← kkkk

rX ← IN_PORT

Z unchanged

C unchanged

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 1


© 2011 Gregory Bredthauer

mcx16

# OUTPUT

OUTPUT rX, rY

| c | c | c | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

OUTPUT rX, kkkk

| c | c | c | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

OUT_PORT_ID ← rX

OUT_PORT ← rY /

OUT_PORT ← kkkk

Z unchanged

C unchanged

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 1

READ_STROBE ← 0


© 2011 Gregory Bredthauer

# JUMP

JUMP rY

| c | c | c | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

JUMP kkkk

| c | c | c | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX unchanged

Z unchanged

C unchanged

PC ← rY / kkkk

Stack unchanged

WRITE_STROBE ← 0

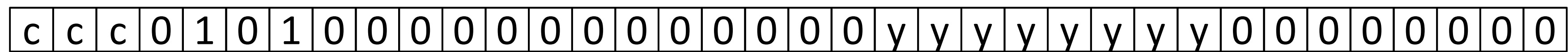
READ_STROBE ← 0


© 2011 Gregory Bredthauer

mcx16

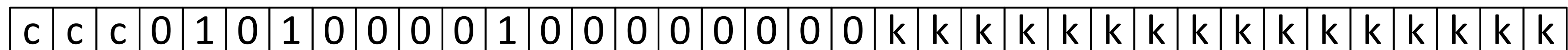
# CALL

CALL rY



| c | c | c | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

CALL kkkk



| c | c | c | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX unchanged

Z unchanged

C unchanged

PC ← rY / kkkk

Stack ← PC

WRITE_STROBE ← 0

READ_STROBE ← 0

 © 2011 Gregory Bredthauer 

mcx16

# RETURN

RETURN

| c | c | c | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX unchanged

Z unchanged

C unchanged

PC ← stack + 1

WRITE_STROBE ← 0

READ_STROBE ← 0


© 2011 Gregory Bredthauer

# LOADRETURN

LOADRETURN rX, kkkk

| c | c | c | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX ← kkkk

Z unchanged

C unchanged

PC ← stack + 1

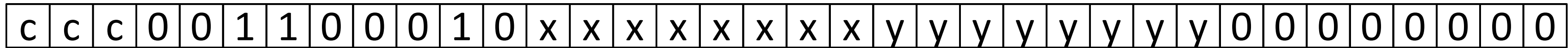
WRITE_STROBE ← 0

READ_STROBE ← 0

 © 2011 Gregory Bredthauer 

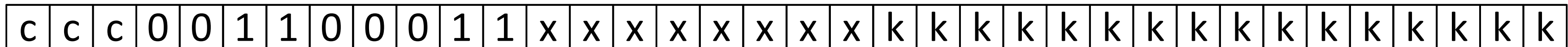
# mcx16 TEST

TEST rX, rY



| c | c | c | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | x | x | x | x | x | x | x | x | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

TEST rX, kkkk



| c | c | c | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX unchanged

result ← rX and rY /
rX and kkkk

Z ← 1 if result is 0

C ← 1 if result has an odd number of bits set

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0

 © 2011 Gregory Bredthauer 

# TESTCY

TESTCY rX, rY

| c | c | c | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | x | x | x | x | x | x | x | x | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

TESTCY rX, kkkk

| c | c | c | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX unchanged

result ← rX and rY /
rX and kkkk

Z ← 1 if result is 0 and
last Z = 1

C ← 1 if result with C
has an odd number
of bits set

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0


© 2011 Gregory Bredthauer

# mcx16 COMPARE

COMPARE rX, rY

| c | c | c | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | x | x | x | x | x | x | x | x | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

COMPARE rX, kkkk

| c | c | c | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX unchanged

result ← rX - rY /
rX - kkkk

Z ← 1 if result is 0

C ← 1 if result < 0

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0


© 2011 Gregory Bredthauer

# COMPARECY

COMPARECY rX, rY

| c | c | c | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | x | x | x | x | x | x | x | x | y | y | y | y | y | y | y | y | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

COMPARECY rX, kkkk

| c | c | c | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | x | x | x | x | x | x | x | x | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k | k |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX unchanged

result ← rX - rY - C /
rX - kkkk - C

Z ← 1 if result is 0 and
last Z = 1

C ← 1 if result < 0

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0


© 2011 Gregory Bredthauer

# mcx16

# SETCY

SETCY

| c | c | c | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX unchanged

Z unchanged

C ← 1

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0


© 2011 Gregory Bredthauer

# mcx16 CLEARCY

CLEARCY

| c | c | c | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

rX unchanged

Z unchanged

C ← 0

PC ← PC + 1

Stack unchanged

WRITE_STROBE ← 0

READ_STROBE ← 0


© 2011 Gregory Bredthauer

# STRING

- STRING is an assembler directive which expands into a series of LOADRETURN instructions.

```
   LOAD r0, MyString           ; Put the address of MyString in r0
StringLoop:
   CALL r0                     ; Read the next char of MyString
   TEST r1, 0xFF               ; Each char is stored in r1
   IF Z JUMP StringDone        ; We’re done if the char is zero
   CALL WriteCharacter         ; Call a subroutine to transmit r1
   ADD r0, 1                   ; Move to the next character
   JUMP StringLoop
MyString:
   ; Specify a register to be loaded and then the string
   STRING r1, “This string will get sent to the world!\0”
StringDone:
   ; The rest of the program follows
```

# WSTRING

- WSTRING operates similarly to STRING, but two characters are stored per LOADRETURN. WSTRING r2, “Yikes” is translated into:

```
LOADRETURN r2, 0x5969   ; ‘Y’ ‘i’
LOADRETURN r2, 0x6B65   ; ‘k’ ‘e’
LOADRETURN r2, 0x7300   ; ‘s’ 0
```

- This allows memory savings for long strings, at the expense of more complicated decoding routines.

 © 2011 Gregory Bredthauer 

# Escape Codes

- Strings and character constants can use the following set of escape codes (similar to C):

| | |
|---|---|
| \\ : Backslash | (0x5C) |
| \r : Carriage return | (0x0D) |
| \n : Newline | (0x0A) |
| \t : Tab | (0x09) |
| \" : Double quote | (0x22) |
| \' : Single quote | (0x27) |
| \0 : Null terminator | (0x00) |


© 2011 Gregory Bredthauer

# MCX16LOADER

- To facilitate faster debugging of mcx16 programs, mcx16loader.vhd is provided to allow JTAG access to an mcx16 program running in an FPGA.
- Connect the RST and LOADx signals to the same ports on the mcx16 and mcx16rom.

mcx16loader declaration:

```
component mcx16loader is
	generic (
		FPGA: string;
		JTAG_CHAIN: integer
	);
	port (
		RST: out std_logic;
		LOADCLK: out std_logic;
		LOAD: out std_logic;
		LOADADDR: out std_logic_vector(15 downto 0);
		LOADDATA: out std_logic_vector(35 downto 0)
	);
end component;
```

mcx16loader instance:

```
inst_mcx16loader: mcx16loader
	generic map (
		FPGA => "S3",
		JTAG_CHAIN => 1
	)
	port map (
		RST => RST,
		LOADCLK => LOADCLK,
		LOAD => LOAD,
		LOADADDR => LOADADDR,
		LOADDATA => LOADDATA
	);
```

 © 2011 Gregory Bredthauer 

mcx16

# JTAG Generics

- Set the mcx16loader generics as desired. FPGA can be "S3" (Spartan 3), "S3A" (Spartan 3A), "S6" (Spartan 6), "V4" (Virtex 4), "V5" (Virtex 5), or "V6" (Virtex 6).
- Set JTAG_CHAIN to 1 or 2 (corresponding to USER1/USER2) for Spartan 3/3As, and 1 through 4 (USER1-USER4) for Spartan 6s and Virtex 4s/5s/6s.
- The mcx16asm -u <chain> option should be set to match the value of JTAG_CHAIN.

 © 2011 Gregory Bredthauer 

# JTAG Loading

- To load a program via JTAG into an mcx16 embedded in a configured FPGA, run:

```
mcx16asm myprogram.txt -l
```

- If you're using a Spartan 3/3A alone on a JTAG chain, and set to JTAG_CHAIN=1 (USER1), that's all that's necessary. Add a "-v" to see the action.
- For more complex scenarios, you must give mcx16asm additional information:
- Pass "-f s3", "-f s6", "-f v4", "-f v5" or "-f v6" to specify the target FPGA family.


© 2011 Gregory Bredthauer

mcx16

# JTAG Chain

- Pass “-bi <bits>”, “-bd <bits>”, “-ai <bits>”, and “-ad <bits>” to specify the IR and DR register lengths of all devices before and after the target FPGA. The DR lengths will just be the number of devices before (-bd) or after (-ad) the FPGA. The IR lengths will be the sum of the lengths of the JTAG instruction registers of all devices before (-bi) and after (-ai) the FPGA. These lengths can be found in the BSDL documentation for each device.

 © 2011 Gregory Bredthauer 

# JTAG Examples

- If your JTAG chain is TDI → XCF01S → V5 → TDO and JTAG_CHAIN is 3, the command would be:

```
mcx16asm prog.txt -l -f v5 -u 3 -bi 8 -bd 1
```

- If your JTAG chain is TDI → S6 → XCF01S → TDO and JTAG_CHAIN is 1, the command would be:

```
mcx16asm prog.txt -l -f s6 -u 1 -ai 8 -ad 1
```

- Creating a batch file to save typing this repeatedly is recommended!

 © 2011 Gregory Bredthauer 

# MCX16ROM

- The mcx16rom.vhd file is the template mcx16asm uses to generate a ROM prepopulated with the user's program.
- The mcx16rom component will have been given a name matching the assembly code filename:

mcx16rom declaration:

```
component mymcx16rom is
	generic (
		ROM_TYPE: string
	);
	port (
		LOADCLK: in std_logic;
		LOAD: in std_logic;
		LOADADDR: in std_logic_vector(15 downto 0);
		LOADDATA: in std_logic_vector(35 downto 0);
		CLK: in std_logic;
		CE: in std_logic;
		ADDR: in std_logic_vector(15 downto 0);
		DATA: out std_logic_vector(35 downto 0)
	);
	end component;
```

mcx16loader instance:

```
inst_mymcx16rom: mymcx16rom
	generic map (
		ROM_TYPE => “block"
	)
	port map (
		LOADCLK => LOADCLK,
		LOAD => LOAD,
		LOADADDR => LOADADDR,
		LOADDATA => LOADDATA,
		CLK => CLK,
		CE => CE,
		ADDR => ADDR,
		DATA => DATA
	);
```


© 2011 Gregory Bredthauer

# ROM OPTIONS

- Pass mcx16asm the –m option to set a minimum ROM size (defaults to 512 words), for example:

  ```
  mcx16asm prog.txt –o –m 1024
  ```

  This will make the ROM at least 1024 words big (larger if the prog.txt exceeds that size): useful during development as your program may grow, and you wish to avoid recompiling the design.
- Set the ROM_TYPE generic to “block” to force BRAM usage or “distributed” to use distributed RAM. mcx16 can use zero BRAMs if the stack and ROM are both set to use distributed memory.

 © 2011 Gregory Bredthauer 

mcx16

# MCX16UART

- mcx16uart is a simple N81 UART that can be used with mcx16 (see the mcx16s6mb Spartan-6 LX9 MicroBoard example)

mcx16uart declaration:

```
component mcx16uart is
	generic (
		CLOCK_RATE: real;
		BAUD_RATE: real;
		MAX_BAUD_ERROR: real;
		FIFO_DEPTH: integer
	);
	port (
		CLK: in std_logic;
		TX_RST: in std_logic;
		TX_WRITE: in std_logic;
		TX_DATA: in std_logic_vector(7 downto 0);
		TX: out std_logic;
		TX_EMPTY: out std_logic;
		TX_FULL: out std_logic;
		RX_RST: in std_logic;
		RX_READ: in std_logic;
		RX: in std_logic;
		RX_DATA: out std_logic_vector(7 downto 0);
		RX_EMPTY: out std_logic;
		RX_FULL: out std_logic
	);
end component;
```

mcx16uart instance:

```
inst_mcx16uart: mcx16uart
	generic map (
		CLOCK_RATE => CLOCK_RATE,
		BAUD_RATE => BAUD_RATE,
		MAX_BAUD_ERROR => MAX_BAUD_ERROR,
		FIFO_DEPTH => FIFO_DEPTH
	)
	port map (
		CLK => CLK,
		TX_RST => TX_RST,
		TX_WRITE => TX_WRITE,
		TX_DATA => TX_DATA,
		TX => TX,
		TX_EMPTY => TX_EMPTY,
		TX_FULL => TX_FULL,
		RX_RST => RX_RST,
		RX_READ => RX_READ,
		RX => RX,
		RX_DATA => RX_DATA,
		RX_EMPTY => RX_EMPTY,
		RX_FULL => RX_FULL
	);
```

# CONCLUSION

- Thanks for taking a look at mcx16! This is free software, so I can't guarantee any support. However, you can post questions, comments, bugs, improvements, etc. on the mcx16 forum here:

  http://www.bitpond.com/forums/forum/mcx16-forum/

- Enjoy!

  \- Greg Bredthauer

 © 2011 Gregory Bredthauer 