

# 6.375 Tutorial 3

## Scheduling, Sce-Mi & FPGA Tools

Ming Liu

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-1

## Overview

- ◆ Scheduling Example
- ◆ Synthesis Boundaries
- ◆ Sce-Mi
- ◆ FPGA Architecture/Tools
- ◆ Timing Analysis

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-2

# Scheduling Recap

1. Intra-rule: What makes a legal rule?

- No double write
- No combinational cycles

2. Inter-rule: When can rules fire in the same cycle?

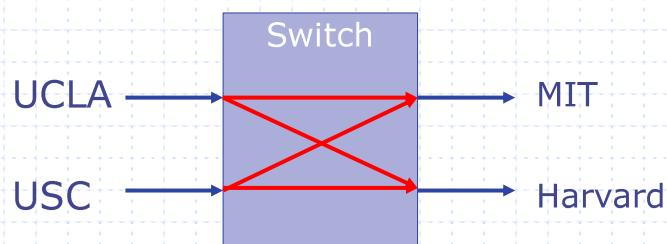
- Parallel execution of rules *appears* as if rules executed in sequential one-rule-at-a-time order

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-3

# Packet Switching



Demo

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-4

# Synthesis Boundary

- ◆ By default BSV compiles your design into a *single flat* Verilog module
  - All methods/rules in-lined
- ◆ Use synthesis pragma to generate separate modules (.v files)

```
(*synthesize*)
module mkMultiplier (Multiplier);
```

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-5

# Synthesis Boundary

- ◆ Benefits:
  - Faster compile times. Modules are reused
  - Modularization at the circuit level
  - Better for synthesis and reporting
- ◆ Drawbacks:
  - Polymorphic modules not supported
  - More conservative guard lifting

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-6

# Synthesis Boundary

## Handling Polymorphism

- ◆ Verilog does not support polymorphic interfaces
- ◆ Wrap polymorphic modules with specific instantiations

```
FFT#(N, ComplexData) fft <- mkFFT();
```

```
(* synthesize *)
module mkSynthesizableFFT(FFT#(N, ComplexData));
    FFT#(N, ComplexData) fft <- mkFFT();
    return fft;
endmodule
```

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-7

# Synthesis Boundary

## Guard Logic (Example)

- ◆ Synthesis boundaries simplifies guard logic

```
method Action doAction( Bool x );
    if( x ) begin
        aQ.enq(a);
    end else begin
        bQ.enq(b);
    end
endmethod
```

- Lifted guard without synthesis boundary:
  - ◆  $(x \&& aQ.notFull) || (!x \&& bQ.notFull)$
- Lifted guard with synthesis boundary:
  - ◆  $aQ.notFull \&& bQ.notFull$

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-8

# Sce-Mi

Feb 26, 2016

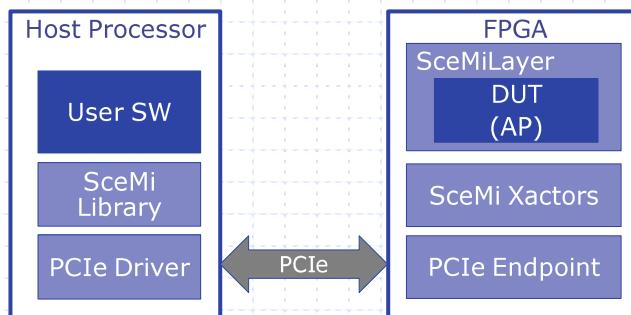
<http://csg.csail.mit.edu/6.375>

T03-9

# Sce-Mi

Standard Co-Emulation Modeling Interface

- ◆ A software/hardware communication framework



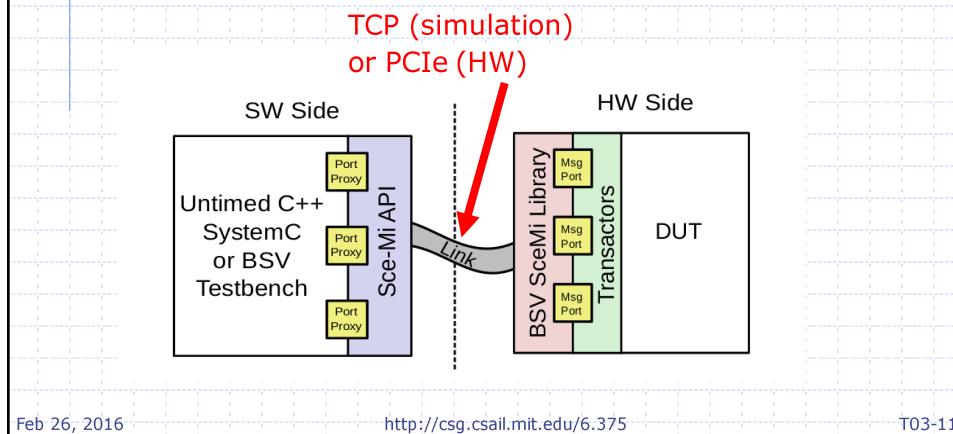
Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

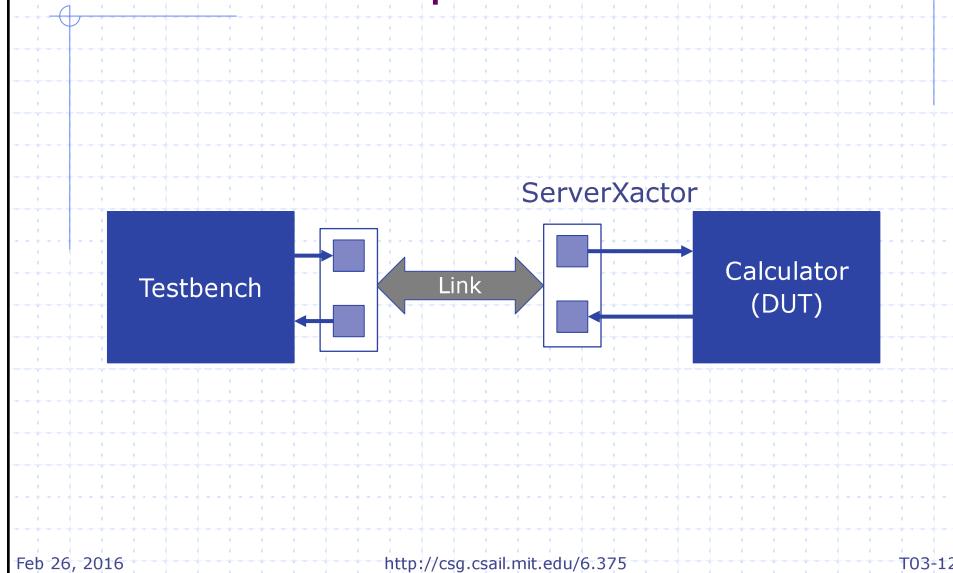
T03-10

# Sce-Mi Abstraction

- ◆ Ports and proxies form a **FIFO abstraction** over a link



## Sce-Mi Example: Calculator



## Sce-Mi: Hardware Calculator DUT

```
typedef Bit#(32) Value;

typedef enum { ADD, SUB, MUL} Operation deriving (Bits, Eq);
typedef union tagged {
    void Clear;
    struct {
        Operation op;
        Value val;
    } Operate;
} Command deriving (Bits, Eq);

typedef Server#(Command, Value) Calculator;

module mkCalculator (Calculator);
...
endmodule
```

Server interface:  
Put commands  
Get results

Feb 26, 2016 http://csg.csail.mit.edu/6.375 T03-13

## Sce-Mi: Hardware SceMiLayer

```
(* synthesize *)
module [Module] mkDutWrapper (Calculator);
    Calculator calc <- mkCalculator();
    return calc;
endmodule

module [SceMiModule] mkSceMiLayer();
    SceMiClockConfiguration conf = defaultValue;
    SceMiClockPortIfc clk_port = mkSceMiClockPort(conf);
    Calculator dut <- buildDut(mkDutWrapper, clk_port);
    Empty calcxactor <- mkServerXactor(dut, clk_port);
    Empty shutdown <- mkShutdownXactor();
endmodule
```

Wrap DUT in  
synthesis boundary

Instantiate the DUT

Create a Server  
transaction on the  
Calculator dut interface

Note: DutWrapper in Lab 4 is slightly  
more complex (discussed later)

Feb 26, 2016 http://csg.csail.mit.edu/6.375 T03-14

# Sce-Mi: Software

```
#include "bsv_scemi.h"
#include "SceMiHeaders.h" ← Auto-generated classes based on your BSV types

int main(int argc, char* argv[]){
    //Initialize scemi ...
    ImportProxyT<Command> import ("",
        "scemi_calcxactor_req_inport", sceMi);

    // Initialize the SceMi outport
    OutportQueueT<Value> outport ("",
        "scemi_calcxactor_resp_outport", sceMi);
}

Type of port
```

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-15

# Sce-Mi: Software

## Sending/Receiving

```
Command cmd; ← Construct the cmd
cmd.the_tag = Command::tag_Operate;
cmd.m_Operate.m_val = 100;
cmd.m_Operate.m_op = Operation(Operation::e_ADD)

import.sendMessage(cmd); ← Send the cmd to hardware (blocking)

std::cout << outport.getMessage() << std::endl;
```

Get results!

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-16

# Sce-Mi: Transactors

- ◆ Many transactor available. See documentation

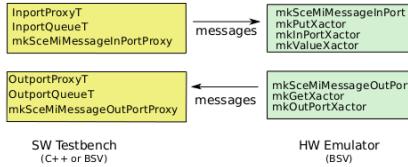


Figure 21: Corresponding Software Proxies and Basic Transactors

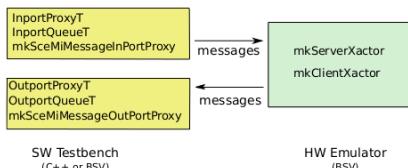


Figure 22: Corresponding Software Proxies and Transactors

Feb 26, 2016

T03-17

# Sce-Mi: Build Process

- ◆ SceMi has its own build tool: build
  - Lab 4: project.bld
- ◆ Always **simulate** first:
  1. Wrap DUT in SceMiLayer in hardware
  2. Run build: "build -v"
    - ◆ This creates scemi.params and C++ headers
  3. Write/update your software C++ code
  4. Re-run "build -v"

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-18

## Sce-Mi: Simulation

- ◆ SceMi simulation is done over TCP sockets
- ◆ “build -v” will generate two binaries
  - bsim\_dut: simulated hardware
  - tb: the host software
- ◆ Run both binaries. They will communicate with each other.

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-19

## Sce-Mi: Building Hardware

- ◆ Run Build:
  - `vivado_setup build -v`
    - ◆ Runs synthesis tool (for a long time) to create FPGA bitfile
    - ◆ Compiles software tb
- ◆ Log into a server with FPGA attached
  - `programfpga <path_to_bitfile>`
- ◆ Run software
  - `runtb ./tb 2.0`

Demo

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-20

# Clocks and Resets in BSV

- ◆ All BSV modules have implicit clock and reset interfaces
  - Compiled Verilog files will have these ports
- ◆ Modules can also have additional clocks/resets as parameters

```
module mkDut(DutIfc); //One default clk/rst  
//Two clks, one default rst  
module mkDut2Clk#(Clock clk_usr) (DutIfc);
```

- ◆ Can specify clocks/reset explicitly

```
DutIfc dut <- mkDut(clocked_by clk, reset_by rst);  
DutIfc dut <- mkDut2Clk(clk_usr, clocked_by clk, reset_by rst);
```

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-21

# SceMi Clocks

- ◆ SceMi transactors run at 50MHz
- ◆ But our DUT runs faster in a different clock domain
- ◆ Thus need clock domain crossing primitives at the DUT interface
  - Synchronization FIFOs/Registers
  - Bluespec provides these

```
//Scemi clock to user clock (dut) sync fifo  
SyncFIFOIfc#(Sample) toApSyncQ <- mkSyncFIFOFromCC(2, clk_usr);  
//User clock to Scemi clock sync fifo  
SyncFIFOIfc#(Sample) fromApSyncQ <- mkSyncFIFOToCC(2, clk_usr,  
rst_usr);
```

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-22

# Running Design on FPGA

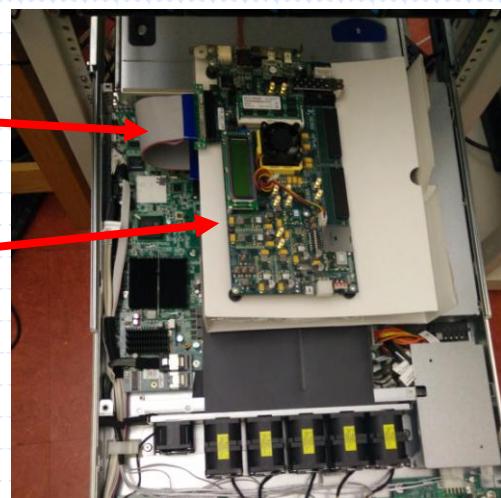
Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-23

## FPGA Hardware Setup

PCIe  
Xilinx VC707  
(Virtex 7)  
Development Board



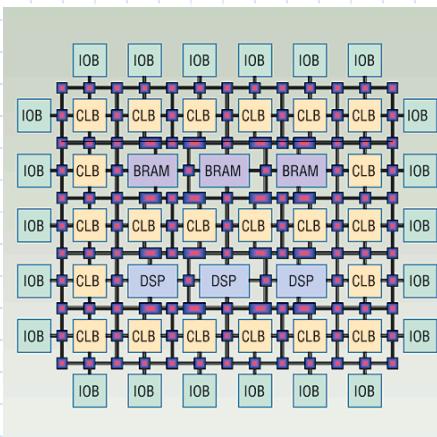
Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-24

# FPGA Architecture Brief

- ◆ CLB: configurable logic block
- ◆ Routing switches
- ◆ BRAMs (~1-8MB total)
- ◆ DSP
- ◆ IOB: I/O buffers
- ◆ Hard IPs: PCIe, DRAM
- ◆ Clocking network



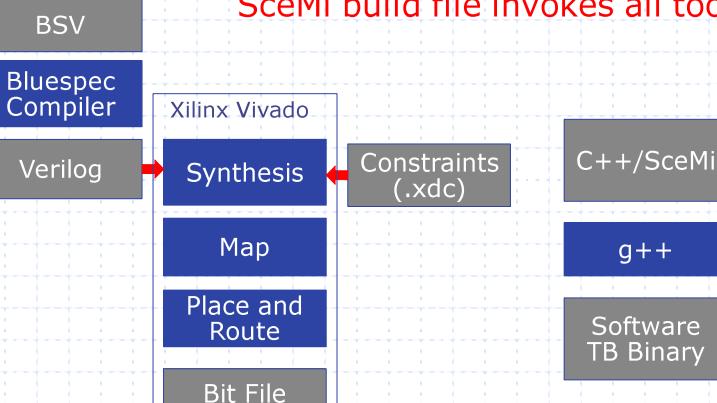
Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-25

## Tool Flow for FPGA

SceMi build file invokes all tools



Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-26

# Reading Timing Reports

- ◆ Focus on Max Delay Path (setup)
- ◆ Slack: Timing margin
- ◆ Location: Physical location of the slice
- ◆ Delay Type: which resource in the slice the path goes through
  - FDRE: register, LUT: look up table, CARRY4: carry chain
  - Net: routing delay
- ◆ Incr (ns): Additional delay added
- ◆ Path (ns): Total delay

**Demo**

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-27

# Debugging Timing Errors

- ◆ Search for “VIOLATED” in the timing report
- ◆ Look at which module the error is from
  - Source and destination
- ◆ What logic is generally on the path
- ◆ Remember that you are looking at one particular path – when there are timing failures on one path, there are typically many paths failing in the same way
  - TNS Failing Endpoints tells you exactly how many
- ◆ Resolve long combinational paths by pipelining/folding your design

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-28

## Examples of Long Paths

- ◆ Multipliers/adders generally expensive
  - Can be cheap for constants
  - Proportional to width

```
rule doMult;
    outR <= a * b;
endrule
```

```
Reg#(Bit#(32)) a <- mkReg(0); //value set dynamically
Reg#(Bit#(32)) b <- mkReg(0); //value set dynamically
method setInputs (Bit#(32) a, Bit#(32) b) ...
```

```
Reg#(Bit#(4)) a <- mkReg(0);
Reg#(Bit#(4)) b <- mkReg(0);
```

```
Bit#(32) a = 2;
Reg#(Bit#(32)) b <- mkReg(0);
```

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-29

## Examples of Long Paths

```
Vector#(N, Reg#(Bit#(32)))
    outVectR <- replicateM(mkReg(0))
rule doSel;
    outR[sel] <= ...
endrule
```

```
Reg#(Bit#(TLog#(N))) sel <- mkReg(0); //dynamically set
```

- ◆ Dynamic selection creates muxes
- ◆ Many input muxes creates long comb paths

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-30

## Examples of Long Paths

```
rule doPipe;  
    for (Integer sel=0; sel< n; sel=sel+1) begin  
        outFIFO[sel].enq( inpFIFO[sel].first );  
        inpFIFO[sel].deq;  
    end  
endrule
```

Likely OK

```
for (Integer sel=0; sel< n; sel=sel+1) begin  
    rule doPipe;  
        outOneFIFO.enq( inpFIFO[sel].first );  
        inpFIFO[sel].deq;  
    endrule  
end
```

Many conflicting  
rules -> large mux

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-31

## A Few Notes

- ◆ Please don't program the FPGA until design meets timing
  - Send me an email if having issues with FPGA
- ◆ Servers with FPGAs attached:
  - bdbm[12-14].csail.mit.edu
  - Do not use these for long compiles
- ◆ Additional servers for compiling
  - bdbm[10-11].csail.mit.edu
- ◆ Sharing
  - Use "w" and "top" to check who is using machine
  - Only one person can use the FPGA at a time
  - Change TCP port in project.bld in sim
- ◆ Start early! Synthesis takes >30 minutes.

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-32

# Lab 4 Code

Feb 26, 2016

<http://csg.csail.mit.edu/6.375>

T03-33