6.375 Tutorial 3
Scheduling, Sce-Mi & FPGA Tools

Ming Liu

Overview

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

Packet Switching

- UCLA
- USC
- MIT
- Harvard

Demo
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);

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
Synthesis Boundary

Handling Polymorphism

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

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

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

Synthesis Boundary

Guard Logic (Example)

- Synthesis boundaries simplifies guard logic

```verilog
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`
Sce-Mi

Standard Co-Emulation Modeling Interface

A software/hardware communication framework
Sce-Mi Abstraction

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

TCP (simulation) or PCIe (HW)

---

Sce-Mi Example: Calculator

---

Testbench | ServerXactor | Calculator (DUT)
Sce-Mi: Hardware
Calculator DUT

```haskell
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

---

Sce-Mi: Hardware
SceMiLayer

```haskell
(* 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)
Sce-Mi: Software

```c
#include "bsv_scemi.h"
#include "SceMiHeaders.h"

int main(int argc, char * argv[]){
    // Initialize scemi ...
    SceMi auto-generates the Command class
    ImportProxyT<Command> inport("", "scemi_calcxactor_reg_inport", sceMi);
    // Initialize the SceMi output
    OutportQueueT<Value> outport("", "scemi_calcxactor_resp_outport", sceMi);
    Type of port

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

    inport.sendMessage(cmd);

    std::cout << outport.getMessage() << std::endl;
    // Get results!
```
Sce-Mi: Transactors

Many transactor available. See documentation

![Diagram showing transactors and messages](image)

Figure 21: Corresponding Software Proxies and Basic Transactors

---

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"

---
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.

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`
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

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

- Can specify clocks/reset explicitly

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

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

```haskell
//Scemi clock to user clock (dut) sync fifo
SyncFIFOIfc#(Sample) toApSyncQ <- mkSyncFIFOPromCC(2, clk_usr);
//User clock to Scemi clock sync fifo
SyncFIFOIfc#(Sample) fromApSyncQ <- mkSyncFIFOToCC(2, clk_usr, rst_usr);
```
Running Design on FPGA

FPGA Hardware Setup

PCIe

Xilinx VC707 (Virtex 7) Development Board
FPGA Architecture

Brief

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

Tool Flow for FPGA

SceMi build file invokes all tools

BSV

Bluespec Compiler

Verilog

Xilinx Vivado

Synthesis

Constraints (.xdc)

Map

Place and Route

Bit File

C++/SceMi

g++

Software TB Binary
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

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
Examples of Long Paths

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

```verilog
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);
```

Examples of Long Paths

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

```verilog
Vector#(N, Reg#(Bit#(32)))
outVectR <= replicateM(mkReg(0))

rule doSel;
  outR[sel] <= ..
endrule

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

```n
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

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.
Lab 4 Code