TSTE12 Design of Digital Systems Lecture 10

Kent Palmkvist

LINKÖPING UNIVERSITY

Agenda

- Code style
- Intellectual Property (IP)
- Alternatives to VHDL
  - System-C
  - SystemVerilog

TSTE12 Deadlines Y,D,ED

- Weekly meetings should have started
  - Internal weekly meeting with transcript sent to supervisor
- Project completion
  - Friday 14 October
  - Presentation
  - Project report

TSTE12 Deadlines MELE, Erasmus

- Weekly meetings should have started
  - Internal weekly meeting with transcript sent to supervisor
- Project completion
  - Friday 28 October
  - Presentation
  - Project report

TSTE12 Presentation/demonstration

- 15-20 minutes/group
- All group members should participate
- Available times will be announced later
- At least two groups at the same time (2 groups audience)
- Projector, computer and DE2-115 board available
- See web page for guidelines of presentation
  - Want a selling presentation (but do not overdo this)
  - Point out what is different from everyone else designs
- Present both technical and administrative results

TSTE12 Project Documents

- Project report
  - Use the general LIPS template document
- Afterstudy report
  - Use the special afterstudy report template
  - Fill in and submit individually
- Delivery
  - Clean out unrelated stuff from the project directories
  - Put a README.TXT at the top level of the project directory
    - Describe where, what name, how to use designs

Handins

- Handin set 2 available today 26 September
  - Deadline 3 October 23:30
- Handin set 3
  - Start 10 October, deadline 17 October 23:30
- Handin set 4
  - Start 24 October, deadline 31 October 23:30
  - Not necessary if you got atleast 9 correct theory and atleast 9 correct coding tasks
- INDIVIDUAL work, no cooperation on handins

Last lecture today for Y/D/ED

- Lecture 11
  - Microprogramming
  - Lab 3
- Lecture 12
  - Low level programming,
  - Assembly language, C
  - Computer Peripheral (I/O)

Resource sharing

- Chose one of two sums
  - Mux + add => 51 area, 8.47 delay
  - Add + mux => 73 area, 7.09 delay

```
if sel='1' then
  tmp_1 := a;
  tmp_2 := b;
else
  tmp_1 := c;
  tmp_2 := d;
end if;
sum <= tmp_1 + tmp_2;
```

```
If sel='1' then
  sum <= a + b;
else
  sum <= c + d;
```

```
sum_1 := a + b;
sum_2 := c + d;
if sel='1' then
  sum <= sum_1;
else
  Sum <= sum_2;
end if;
```


Resource sharing

- Chose one of two sums. May add both or chose inputs first
  - Mux+add => 51 area, 8.47 delay
  - Add+mux => 73 area, 7.09 delay
- Flattening and structure. (logic level, not hierarchy)
- Logic can be flattened to e.g., two levels instead of three. Different results of area and logic

How is timing requirements defined?

- Often derived from a symbolic clock
- Signals are defined from edges of the clock
  - Fix setup and hold time. Include clock skew
- Usually defined as maximum delay
  - Expensive to guarantee minimum delay
  - Delay pin to flipflop, flipflop to pin
  - Time from flipflop to flipflop
- Possible to specify multi cycle delay
- False paths

Results

- Time reports
  - Generated by analysis of netlist/layout
  - Critical path reports
- Area reports
- Resource reports
  - Routing, flipflops, LUT, multipliers etc.
- VHDL simulation models
  - Post synthesis, post layout
- Layout possible to modify (edit at bit level)

Synthesis operation

- Synthesis is based on different types of pattern matching
  - Support most constructs
  - Behavour may still be different
  - Often adds complicated patterns that are then simplified
    - E.g., full flipflop with asynchronous reset and set with fixed inputs.
- Example: Generally generates a single flipflop, but timing of Qinvers differs between simulation of VHDL and synthesized design.

```
Process dflipflop(clk)
begin
  if rising_edge(clk) then
    Q <= D;
  end if;
  Qinvers <= not Q;
end process
```


Recommended patterns

- Style guide exists (patterns)
  - Specific to the synthesis tools
- Specify patterns that are allowed and recommended
  - Important to produce efficient implementations
  - Example units: counters, memories, tristate buffers
- These manuals are available online

Common rules/hints for synthesis

- Do not assign initial values to signals and variables in declarations
  - Not supported for synthesis
  - Use explicit reset instead
- Counter design
  - Use loadable down counters if not power of 2 counting
    - Avoids comparison operation, use carry result instead
- Always use limited number ranges
  - May get 32 bit arithmetic if not limited
- Allow synthesis tool to select state coding

Storage elements

- RAM block synthesis
  - Ordinary array implementation (if not recognized as RAM): flip-flops!
  - Altera Cyclone IV 2C115: 3.9 Mbit RAM
- Large number of available types
  - single/multi port
  - Synchronous/asynchronous
- RAM areas may be initialized (when FPGA is configured)
- ROM areas sometimes implemented as initialized RAM Areas
  - Described as case statements or array of constants

Single port memory model

- Some hardware require clocking

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity sync_ram_singleport is
  generic (data_width : natural := 8;
           addr_width : natural := 8);
  port(
    clk   : in std_logic;
    we    : in std_logic;
    addr : in std_logic_vector(addr_width-1 downto 0);
    data_in : in std_logic_vector(data_width-1 downto 0);
    data_out : out std_logic_vector(data_width-1 downto 0));
end entity;
```

```
architecture rtl of sync_ram_singleport is
  type mem_type is array (2**addr_width-1 downto 0) of
     std_logic_vector(data_width-1 downto 0);
  signal mem : mem_type;
  signal addr_reg : std_logic_vector(addr_width-1 downto 0);
begin
  singleport : process(clk)
    begin
      if rising_edge(clk) then
        if (we = '1') then
          mem(conv_integer(addr)) <= data_in;
        end if;
        addr_reg <= addr;
      end if;
    end process singleport;
    data_out <= mem(conv_integer(addr_reg);
end rtl;
```


If synthesis does not achieve the optimization goals

- Rerun synthesis
  - Optimization often based on probabilistic algorithms
  - Different results in every run
- Try another optimization algorithm in the tool
  - Usually possible to optimize for area or speed
  - Combination of optimizations may give better results
- Change the state coding

  Select different state coding algorithms

Achieving the optimization goals, cont.

- Rewrite the VHDL code
  - Manually define the state coding
  - Rewrite with same functionality
    - Known as retiming
    - Register balancing
    - If vs case
- Rewrite giving different functionality
  - Pipelining
- Rewrite operations manually
  - Supplied adder and multiplier structures may not be optimal in all situations

Achieving the optimization goals, cont.

- Increase the minimum power supply voltage
- Reduce the temperature range
- Change technology
- Reduce error coverage
  - Reduce ability to test manufactured chips
- Change to a better synthesis tool
- Optimize logic manually
- Change optimization goals

IP-blocks (Intellectual Property)

- IP-blocks are predesigned function blocks of different types
- Soft cores
  - synthesizable code, often technology independent
  - E.g., HDL designer moduleware
- Hard cores
  - Pure layout for a specific technology/process
- Firm cores
  - In the middle, for example parameterized gates

How do they work

- May consist of a FA cell or gate up to a complete microprocessor or dedicated systems such as a modem.
  - Examples: microprocessors (ARM, powerpc etc.), memories, peripherals (usb, ethernet, etc.)
- Requires a high-level model that can be used for behavioural simulation (usually not possible to synthesize)
- Behavioural model is replaced with optimized netlist or layout at synthesis (sometimes not available to the user!)

Advantages of IP

- Reuse of code and designs
- Tested (hopefully)
- Fast path to final design
- Do not need to be an expert on every subsystem
  - Example: fast multiplier structures
  - Still get high performance designs

Drawbacks with IP

- Interface problems
  - Clock rates, bus protocols, number systems, wordlengths, etc.
- Risks at purchase
  - Functionality
  - Documentation
- Support
- Verification
  - Require lot of testbenches
  - Missing models

Models at different abstraction levels

- Model type, development environment, need, usage
- ISA
  - C,C++
  - Microprocessor based designs, HW/SW
  - High-speed simulation, application run
- Behavioural
  - C, C++, HDL
  - Non-microprocessor designs
  - High-speed simulation, application run

Models, cont.

- Bus functional
  - C, C++, HDL
  - System simulation, internal behavior of the core
  - Simulation of bus protocols and transactions
- Fully functional
  - HDL
  - System verification
  - Simulation of cycle-by-cycle behavior

Models, cont.

- Emulation
  - Synthesized HDL
  - High-speed system verification
  - Simulation of cycle-by-cycle behavior
- Timing
  - Stamp, Synopsis do, SDF
  - Required by firm and hard cores
  - Timing verification

Models, cont.

- Floor plan/area
  - LEF-format
  - Required by hard cores only
  - SoC-level integration and physical design

IP blocks

- Been around for a long time
  - Standard cell libraries (hard IP)
  - Produced by process vendors (chip manufacturers)
  - Memory layouts
- Increasing number of 3rd party IP producers
  - FPGA vendor specific tools to help integrate IP into design flow

Design flow using IP

- Top-down design flow difficult
  - Must get a match between subsystem and IP block
- Best to use a meet-in-the-middle approach
  - Identify functionality to be put in IP
  - Perform top-down partitioning until meet IP

Soft CPU

- Common trend to include soft CPU support
- CPU structure defined as IP
- Peripherals added using configuration files / GUI
- Software drivers automatically included
- GNU based development systems
- Custom instruction support

Alternative HDL languages

- Verilog
  - IEEE standard 1995, revised 2001, merged into Systemverilog
- Systemverilog
  - IEEE standard 2005
- System-C
  - IEEE standard 2005
- Handel-C, Catapult C
- Mobius, JHDL (Java HDL)
- Chisel

Verilog language background

- First designed 1985
- Designed for logic simulation (not synthesis)
- Owned by Cadence until 1990
  - Released to public 1990
- IEEE standard 1995
- Inspired by C
- Possible to mix simulation of blocks in Verilog and VHDL

Verilog, general aspects

- C-like syntax
  - Operators
  - bitfields
- Case sensitive identifiers
- Include files (.h)
  - Corresponds to VHDL packages
  - Share common definitions
- Not strongly typed
- No pointers and access types

General structure

- Module
  - Corresponds to VHDL entity
  - Specifies interface and function

```
module small_block (a, b, c, o1, o2);
  input a, b, c;
  output o1, o2;

  wire s;
  assign o1 = s || c ;
  assign s = a && b ;
  assign o2 = s ^ c ;
endmodule
```


Data types

- Signal values [0 1 x z]
  - Predefined
- Values specifies bitwidth
  - 334 32 bits wide decimal number
  - 3'b11 3 bits wide binary number (ie, 011)
  - 20'h'f_ffff 20 bits wide hexadecimal number
  - 10'bZ 10 bits wide all tri-state

Nets and registers

- Multiple Net types
  - wire, tri, wor, wand, supply0, supply1
- Net use
  - Corresponds to a signal connecting elements
  - Only possible to assign using continuous assignments (not in always blocks)
- Reg
  - Store a value
- Reg and nets can be bitvectors and arrays

Process equivelent construct

- Initial block
  - Only run once
- Always block
  - corresponds to process statement
  - Sensitivity list
  - Sequential and/or parallel code inside

Examples: latch and register

- Sensitivity list is a boolean expression

```
...
input input_foo, ena ;
reg output_foo ;
...
always @ (ena or input_foo)
  if (ena)
    output_foo = input_foo ;
...
```

```
....
input input_foo, clk ;
reg output_foo ;
....
always @ (posedge clk)
  output_foo = input_foo ;
....
```


Generic template for clocked circuit with asynchronous and synchronous

```
always @(<edge of clock> or <edge_of_asynchronous_signals> )
  if (<asynchronous_signal>)
     <asynchronous signal_assignments>
  else if (<asynchronous_signal>)
    <asynchronous signal_assignments>
    ...
  else
    <synchronous signal_assignments>
```

Timing model

- Specify a minimum time resolution
  - Not completely separated from “standard time”
  - Possible to define delay of assignment
    - #1 test = a; // assign test after 1 time unit
- May have a data slip
  - Always @(posedge clk) Q1 = D;
  - Always @(posedge clk) Q2 = Q1; // Data slip
- Data slip solved by added delay
  - Always @(posedge clk) Q1 = #1 D;
  - Always @(posedge clk) Q2 = #1 Q1;

Dynamic processes

- Support named events
  - Declare an event: event event7;
  - Trigger event: -> event7;
  - Code triggered by event:

```
@ (event7) begin
<Some procedural code>
end
```

- Support fork/join
  - Difficult to translate to hardware

SystemC

- www.systemc.org
  - Possible to download C++ library (free)
- Language based on standard C++
- Initially used for system level simulation and verification
  - Include IP block functionality
  - Add timing and hardware limitations
- Same timing model as VHDL (macro and micro timing)

SystemVerilog

- The following presentation found at
  - http://compas.cs.stonybrook.edu/~nhonarmand/courses/sp15/cse502/res/date04_systemverilog.pdf
  - Presented at Design Automation Conference (DATE) in 2004

Short summary

- VHDL
- Use of VHDL
  - High Abstraction Level
  - RTL
  - Gate Level
- Synthesis
  - RTL -> Gate

LINKÖPING UNIVERSITY