CMPE 415: FPGA Development
Prof. Ryan Robucci

Illustrations snapshots are from the Design Warrior’s Guide to FPGAs by Clive Maxfield, Elsevier
Vocab Lesson

- CAD: Computer Aided Design
- CAE: Computer Aided Engineering
- EDA: Electronic Design Automation
- Historically,
  - CAE refereed to front end tools like design capture and simulation
  - CAD refereed to backend tools like layout, place, and route
  - EDA was accepted as the merger of all topics
As gate counts and designs became large, HDL replaced schematics since there was a need for a more, compact, more manageable description.

<table>
<thead>
<tr>
<th>Level</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithmic</td>
<td>Functional</td>
<td>High level functional description loops, processes</td>
</tr>
<tr>
<td>Behavioral</td>
<td>Functional</td>
<td>Description of registers and logic between them. Boolean Expressions and Registers implied by storage, This includes simple if structors</td>
</tr>
<tr>
<td>RTL</td>
<td>Functional</td>
<td>Instantiateations of primitive gates and registers</td>
</tr>
<tr>
<td>Gate</td>
<td>Structural</td>
<td>Instantiateations of Transistors Also called transistor level</td>
</tr>
<tr>
<td>Switch</td>
<td>Structural</td>
<td>Instantiateations of Transistors Also called transistor level</td>
</tr>
</tbody>
</table>
Of key importance are the length in time of the design cycles.
Of key importance are the length in time of the design cycles.
Synthesis – Mapping

- is the process of associating entities such as gate-level functions in the gate-level netlist with the physical LUT-level functions available on the FPGA

Portion of gate-level netlist

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The Design Warrior's Guide to FPGAs
Synthesis – Packing

- Grouping of LUT and registers into CLBs

Collection of LUTs

LUT

LUT

LUT

\[\ldots\]

LUT

Collection of CLBs

\[\ldots\]

\[\ldots\]
Place and Route

- Placement of CLBs

Collection of CLBs

CLB

LUT
LUT

CLB

LUT
LUT

CLB

LUT
LUT

CLB Placement
Interdependence

- Mapping, Packing, and Place and Route sometimes have dependencies.
- A global optimum solution if difficult to find
- Heuristics are used in each step that generally lead to a good outcome
Timing Analysis and Post-Place-and-Route simulation

- After Place and route, we have a fully routed physical design and a **timing analysis** tool can extract timing and check for any timing violations (setup, hold, etc...) associated with any of the internal registers.

- These are more accurate than load estimates that would be used before place and route.

- A new netlist can be generated that includes accurate delays in a **standard delay format** SDF file associated with the *post place and route netlist* (can't push delays directly back to original description as lots of stuff has moved around or changed).
FPGA vs ASIC Tools

- FPGAs vs ASICs: FPGAs are regular structures and represent a constrained design space for analysis tools (and synthesis tools).
- Many designs will emerge from one underlying physical hardware design.
- This underlying hardware can be heavily characterized in the fabricated IC. This for allows tools that can perform accurate general design analysis for anyone using the same underlying hardware. In ASICs, each design can be very different, meaning it is more difficult for tools to accurately predict timing for every possible design. In ASIC Design, SPICE-Level simulation is sometimes used.
Static Timing Analysis

- Static timing analysis refers to using delays extracted from physical implementation to analyze timing directly rather than through simulation.
  - Place-and-routed delays are extracted from place and routed design.
- Static timing analysis does not involve driving inputs into the system and analyzing resulting waveforms.
- Pre-place-and-route estimates delays and can drive synthesis, timing-driven synthesis.
- Typically pessimistic delay assumptions are made to arrive at a worst-case model – a data-driven simulation may reveal what delays are really significant.
- Static Timing Analysis is often fast and may be part of an automation tool’s optimization process to test and evaluate design option trade-offs.
Timing Analysis Concepts

• User Constraints File
• Clock Domains
• Clock jitter
• Async Reset
• Multiple Clock Cycle Allowance (an Exception)
Clock Period Constraint

- For Xilinx Tools, a PERIOD constraint should be supplied for every clock

```plaintext
NET "CLK_50MHZ" PERIOD = 20.0ns HIGH 40%;
```

- or

```plaintext
NET "CLK_50MHZ" TNM_NET="TNM_clk50"
TIMESPEC “TS_clk50” = PERIOD “TNM_clk50” 20.0ns HIGH 40%;
```
Clock Domain is Defined by Sequential Elements

• Sequential Elements Include
  – Flip Flops
  – Latches
  – Distributed RAM/ROM
  – Block RAM/ROM
  – FIFOs
  – I/O Hardware with Clock Input (e.g. I/O SerDes)
  – Hardware bocks with Clock Input (e.g. Xilinx MULT18/18)

• The combinatorial paths between sequential elements in the same clock domain are constrained and must be analyzed
Setup and Hold Time Times

- Setup and Hold times define a window around a clock edge during which data inputs to a register should not transition.
- Setup Time defines the time before a clock edge that a signal must settle. A violation occurs with a path delay is too large. (It so happens that negative setup times are common)
- Hold Time defines the time after a clock edge that a signal must not begin a transition. A violation occurs when a path delay is too small.
- Setup and Hold Time Slack quantify how much “room to spare” before and error occurs. A negative slack indicates a violation.
Clock Skew

- Clocks are buffered through a clock routing network. Clock signals are delayed with respect to the original clock.
- Paths are defined starting at a source register and terminating at a destination register.
- Path delay ($T_{PD} = T_{clk-to-q} + T_{comb.\ path\ delay}$) is defined with respect to the source register clock edge.
- Clock Skew is the difference in arrival time of clock edges at destination registers and source registers.

Clocks are buffered through a clock routing network. Clock signals are delayed with respect to the original clock.
Positive Clock Skew

- Setup and hold time windows are defined with respect to the destination register clock edge.
- If the destination clock is more delayed than the source clock, it represents positive clock skew with respect to that path. This gives more time for a path to settle and thus avoid a setup time violation. It unfortunately also delays the cutoff time for holding a data signal.
Negative Clock Skew

- If the destination clock is less delayed than the source clock, it represents negative clock skew with respect to that path. This gives less time for a path to settle and advances the cutoff for when a signal must hold.

Clock

Available Path Delay Without clock skew
Min Allowed $T_{PD}$ w/o clock Skew
Min Allowed $T_{PD}$ with clock Skew (better)

Available Path Delay with clock Skew

Destination Clock

Source Clock
Static Timing Analysis is a structural analysis based on previous characterizations. A key parameter from the analysis is the **setup timing slack**

\[ T_{CLK\_TO\_Q} \]: Delay from clock edge to sequential gates update

\[ T_{CPD} \]: Delay through combinatorial gates and routing

\[ T_{PD} \]: Path Delay \( T_{CLK\_TO\_Q} + T_{CPD} \)

**Critical Path Timing requirement:** \( T_{clk} + T_{skew} > T_{PD} + T_{setup} \)

\( T_{skew} \): Time from when clk edge occurs at an source flip-flop to when the edge occurs at a destination (e.g. clock delay 1 - clock delay 2) As defined here, positive clock skew with respect to a critical path increases setup slack, while negative skew reduces it.
Setup Time Analysis (slack)

- For each path there should be some slack to the timing. A positive slack value refers to how much extra delay could be added or how much faster a clock rate could be.

\[ T_{suSlack} = T_{clk} - (T_{PD} + T_{setup}) + T_{skew} \] - Jitter or Uncertainty

\( T_{suSlack} \): positive slack indicates the timing requirement is met for a defined clock period while a negative slack means it has not been met.

All paths must be analyzed. The paths with the longest delay are important, but the analysis is a combination of delay and skew, not just path delay. A short delay path could still be a problem.
Hold Time Analysis and Slack

- Hold Time Analysis Avoids Race Conditions

\[ T_{\text{slack(hold)}} = T_{PD} - T_{\text{hold}} - T_{\text{skew}} \]

- Most problematic are “short” combinatorial paths and high clock skew (e.g. back-to-back registers far from each other on the clock network)

- Can fix by slowing path (adding several slow buffers in series)
  - In VLSI, tend to route clock in opposite direction of data whenever creating shift register chains.
What is (not) Included? Input Offsets and Output requirements

- Assume CLKA and CLKB are independently constrained – **Unrelated Clock Domains**

- By default, the input and output paths, regardless if there is only one or more clock domains, are not analyzed
  - Can add a specification for time after a clock edge allowed to reach output pad
  - Can add a specification for the transition window for input to analyze setup and hold time

- **Cross domain paths** are paths originating from the output of a sequential element in one clock domain and ending at the input of sequential elements in another clock domain

- By default, if multiple clock domains are present, cross domain paths are not constrained and not analyzed.

- If the clocks of two clock domains are related in some limited ways, the paths can be analyzed if the relationship is specified...

Source: Xilinx Timing Constraints User Guide

*Figure 3-12: Unrelated clock domains*
Following is an example of the PERIOD constraint syntax. The TS_Period_2 constraint value is a multiple of the TS_Period_1 TIMESPEC.

```
TIMESPEC TS_Period_1 = PERIOD "clk1_in_grp" 20 ns HIGH 50%;
TIMESPEC TS_Period_2 = PERIOD "clk2_in_grp" TS_Period_1 * 2;
```

Note: If the two PERIOD constraints are not related in this method, the cross clock domain data paths is not covered or analyzed by any PERIOD constraint.
Additional Constraints

- Can exclude paths were timing is not important (false paths)
- Can override delay on some paths
  - Such as providing a series of multipliers two clock cycles instead of one to complete
Asynchronous Signals
(e.g. Async Clear)

• Note that asynchronous signals are not easily analyzed for timing and are sensitive to glitches.

• Wherever sensitivity to glitches exits, use registered output logic to generate the control signal.
Dynamic Timing Analysis

- Whereas static timing analysis processes timing equations based on the circuit structure, dynamic timing analysis uses the results of a temporal simulation.

- The generated/saved waveforms (i.e. transient waveforms) are analyzed for timing violations and glitches.

- A complexity is that suitable input sequences (input vectors) must be created to test characteristics of the circuit at internal points.
Dynamic Timing Analysis using Event-Driven Simulation

- Dynamic Timing Analysis uses an event driven simulation, just like functional simulation i.e. dynamic timing is a gate level simulation with timing information.

- Event driven simulation exploits the fact that most signals are quiescent at any given point in time.
  - In event driven simulation, no computational effort is expended on quiescent signals, i.e., their values are not recomputed at each time step.
  - Rather, the simulator waits for an event to occur, i.e., for a signal to undergo a change in value, and ONLY the values of those signals are recomputed.
  - Decide time step size dynamically ("on the fly")
Review of Timing in Verilog: Inertial vs Transport Delays

- Verilog tools include several methods for annotating timing including use of
  - specify blocks

- Timing Data augmentation using a sidecar files e.g. Standard Delay Format Files

- Delay parameters:
  - Review Starting at Slide 21 Lecture 05, especially Inertial vs Transport Delays
Dynamic vs Static Timing Analysis

Functional vs Timing Verification

- **Functional simulation and verification** refers to verifying logical descriptions, including Boolean expressions and register transfers.

- **Timing verification** refers to making sure that all timing requirements, external and internal (setup, hold, etc..) are satisfied.

- **Dynamic Timing Analysis** refers to analysis of timing by simulating the system with inputs and examining the resulting waveforms.

- **Static timing analysis** uses no data and does not use a simulation – is just analyzes the structure.

- A timing analysis tool stores delays in a separate file: Standard Delay Format (**SDF**) file along side the post place and route netlist.
Review Static vs Dynamic

- Dynamic timing analysis and functional simulation each require well-selected input sequences. These are called test-vectors. Static Timing analysis uses no test vectors.

- Dynamic timing analysis is sometimes combined with functional simulation while static timing analysis can not.

- Though Dynamic Timing and Functional Analysis use an event driven simulation which is much faster than SPICE-level “analog” circuit simulation, Static Timing Analysis is much faster and is used during place and route iterations.

- Static timing analysis is more straightforward with one clock domain, though can be extended to handle multiple clock domains. Dynamic timing analysis can automatically handle multiple clock domains.
In-Line Verification Approach

HDL Testbench

- Stimulus
- RTL
  - Functional Model
    - Possibly Non-synthesizable
    - Behavioral Code
  - Per Clock Cycle Compare
Post Simulation Analysis
Verification Approach

- Input Vector File
- Expected Output Vector File
- C/Perl/Python Waveform Comparison
- Saved Output Vector

RTL → Simulation → Expected Output Vector File → C/Perl/Python Waveform Comparison → Saved Output Vector
Code coverage metrics

- Basic: % of lines verified
- Branch: % of branches covered
- Expression coverage: % of input combinations to expressions covered
- State Coverage & Path Coverage: % of states and/or transition paths taken in a state machine
- Functional Coverage: % of functions of circuit covered: memory read/write, multiplication, find max etc...
- Assertion/property Coverage: % of coded assertions coverage
Techniques to improve simulation time

- Use of Reduced functionality
- Use of Reduced explicit delay
- Use of Mixed-language simulation environments
FPGA vendors role in EDA

- On ASIC side, tools are expensive
- FPGA companies focused on selling FPGAs provided tools for cheap in order to sell the FPGAs.
- Tools started to be produced by third parties
VHDL vs Verilog

Many HDLs exist, VHDL and Verilog are the most popular.

- Relatively easy to learn syntax
- Fixed data types
- Interpreted constructs
- Good gate-level timing
- Limited design reusability
- Limited design management

- Relatively difficult to learn syntax
- Abstract data types
- Compiled constructs
- Less good gate-level timing
- Good design reusability
- Good design management

The Design Warrior's Guide to FPGAs
Verilog vs VHDL

- VHDL popular in government-related work and high-level design
  - Syntax is longer, self-documenting
- Verilog, initially targeted as a simulation language, used by IC designers and people wanting to do faster simulations
  - Verilog is compiled into simulation-platform bytecode and run
- Other HDLs include SystemC, SystemVerilog
Coding without understanding hardware

• As I advised, behavioral code should be a shorthand hardware description. You should not code without having an understanding of the hardware you are implying.

• You should not arbitrarily code algorithms without any idea of the hardware you are describing. Your code organization and approach should somewhat correlate to the structure of the hardware you would design.

• Synthesizers are getting better, but there will be cases where good hardware insights should drive implementation.
Hardware Implications in Coding

- Hardware resources $A^9$ vs $A^{<<3} + A$
- Experiment with various implementations and check the RTL to learn what to and how much to expect from your synthesizer.
Hardware Implications in coding

- **Resource sharing**

  ```
  if (B > C)
    then Y = A + B;
  else Y = A + C;
  end if;
  ```

  Resource Sharing = ON

- Key resource sharing hint is assignment to the same named value
- Xilinx Recommends
  
  \[ Y = (B > C) \ A + B : A + C \]

- If in a statemachine, assign to the same named reg from different states.
  
  S1: temp = A+B;
  S2: temp = B+C;

The Design Warrior's Guide to FPGAs
Priority structures, Parallel Muxes
Latch/memory inferences

case SEL of;
   "00": Y = A;
   "01": Y = B;
   "10": Y = C;
   otherwise: Y = D;
end case;

if SEL == "00" then Y = A;
elseif SEL == "01" then Y = B;
elseif SEL == "10" then Y = C;
else
end if;

The Design Warrior's Guide to FPGAs
RAM/ROM Options

- Distributed RAM: uses SRAM-Based LUTs as memory
- Block RAM: Used Dedicated RAM Blocks. More dense storage
- External RAM: Typically need to implement a memory controller. Newer FPGA offer this as Hard IP
- RAM without a write access can essentially implement a ROM – just need an initialization process.
ROM Examples

- Here values are initialized in the verilog code
  http://www.asic-world.com/code/hdl_models/rom_using_case.v

- Tools will also support combining a presynthesized design configuration file with a “initial memory contents” file to create a final bitstream
  https://www.xilinx.com/itp/xilinx10/isehelp/cgn_r_coe_file_syntax.htm
IP Centric Design

- Review early slides for examples of common IP cores.
  - Building block approach tasks a designer with understanding existing micro and macro blocks and augmenting with minimal custom design

- Hard IP Cores vs Soft IP Cores
  - Hard IP Cores – hardware built into FPGA
  - Soft IP Cores – Typically not Pre-synthesized Module to reside in the FPGA fabric, often highly-customizable

- Firm IP Cores
  - Soft IP Typically Presynthesized,
  - potentially already partial mapping, placement, contrains set, etc., less ability to customize

- Encrypted IP Unplaced and unrouted
  - encrypted netlist provided
  - Functional simulation model provided
  - Placed and routed
    - encrypted netlist with placement provided already mapped to LUTs
    - Design is already tweaked and may include placement of pins, hard cores
  - Functional simulation model provided (not to be used for synthesis)
One of the problems with HDL, specifically RTL, is that the design time is very long. Updating the code can be time consuming. Verifying the code can be time consuming. This limits design iterations and therefore the ability to make high level optimizations or explore different designs.

High level languages aim to solve this problem, but hiding the concepts of parallelism and timing is difficult. Hiding the issue or having users unaware of them is problematic.
Sequential Code (Software) Synthesis and Simulation

- Functional simulation is an inherent feature of sequential code approaches and typically runs very fast.
- To produce hardware from software either a to-HDL or direct-to-hardware path can be taken.
Embedded Processor-Centric Design Flow

- If processor running sequential code makes sense, it can be added to a design which allows directly integrating software in an FPGA Design

- Hard vs soft processor cores
  - Today ARM processors are commonly found in FPGAs – hard processor
  - Customizable processors can also be synthesized – soft processors

- Hardware/Software Partitioning
  - Control vs processing
  - Processing speed
  - Lots of high speed parallel timing required vs multiplexing hardware
  - Ease of coding
Hardware Software Codesign

System architects partition design into hardware and software functional blocks (may be part of system/algorithmic-level environment or done by hand)

Original Concept

Hardware design entry (RTL)
- Synthesize
- Place-and-Route
- Configuration file
- Download to development board

Software design entry (C/C++ or A)
- Compile and/or Assemble
- Link etc.
- Executable image

The Design Warrior's Guide to FPGAs
Simulation Environments for Processors

Need to simulate Processor in HDL environment

• Options:
  – RTL model – slow
  – C/C++/SystemC model – high-level
  – Specialized simulators the perform cycle-based simulation (not quite the level of detail of timing)
DSP Design Flows

- Centered around Commonly used blocks and techniques
- Emphasize use of
  - dedicated hardware macro blocks, FFT, DSP cores
  - dedicated low-level hardware features (e.g. carry add-logic)
- Data Flow Programming: Specialized description tools (DSP-block-level diagramming provided in Labview, Simulink, etc...) e.g. draw pictures
- Floating-point versus fixed-point often becomes topic in DSP hardware implementation

Silicon Virtual Prototyping

- FPGA design can be a useful part of and ASIC Design Flow

- FPGAs as prototyping:
  - Ability to implement hardware design in system and perform in-system design test

- Cost of FPGA tools is much less
  - Ability to design with cheaper tools first

- Iterations can be over hours vs days or weeks or even months for a silicon rerun
Board and multi-FPGA development

• Some modern tools handle synthesis to multiple FPGAs and handle modeling of components in a variety of languages including C

• Working with multiple clocks, data serialization and deserialization, delay modeling, and constraints become more critical
Concluding Points

- As synthesizers improve, less dependence on the coding style may be possible.
  - General conversion of C-like software without regard to hardware or how synthesizers work might still lead to an optimal solution

- Reliance on IP blocks (predesigned blocks) is becoming heavier. Goal of some high-level design tools and synthesizers is to allow designer to use them.

- Hardware isn’t best for everything, can use general purpose processors.
  - Hardware software design is becoming increasingly important with many soft and hard processor cores available as options