**Midterm Notes**

**ECE 6213 – Design of VLSI Circuits**

**Osama Yousuf**

**Lecture 1 – ASIC/FPGA & Hardware Design Overview**

* 6213 focuses on HDL + full set of IP only, 6214 will include DSP (Digital Signal Processing) + LP (Low Power) optimizations
* HDL-based design flow
  + Target-independent
    - Design specification
    - Design entry
    - Functional Verification
  + Target-dependent
    - Choose target: PLD or CPLD/FPGA or GA/Standard Cells
    - PLD:
      * Device Selected
      * Programming
      * Prototyping
    - CPLD/FPGA:
      * Device Selected
      * Synthesis & Optimization
      * Post-synthesis verification
      * Place and route
      * Timing analysis
      * Programming
      * Prototyping
    - Standard Cell
      * Cell Library Selected
      * Synthesis & Optimization
      * Post-synthesis verification
      * Place and route
      * Timing analysis
      * NO Programming
      * Prototyping
* Top-level Design
  + **Design Specific Methodology (DMR):** review technical challenges and the proposed design flow for the ASIC/FPGA
  + **Design Environment:** Includes various scripts and processes for creating block level designs and tests
  + \***RTL Top Level Design Entry:** JTAG insertion, IP instantiation, etc.
  + \***RTL Top Level Simulation & Debug:** Various methodologies may apply such as random base simulation, traditional RTL test bench, etc.
  + RTL Code Freeze: If debugging is complete, RTL code is frozen, all action items are satisfied.
  + \***Netlist Generation:** Synthesize RTL to a NETLIST.
  + **Revision Control:** All files in the design must be checkpointed and released.
  + \***Physical Verification:** Final formal verification, STA, gate-level simulations must be performed
  + **Initial Sign Off (ISO):** Final review, concludes the design
* Block level design:
  + High Level Description
  + Low level Description and Test Plan
  + RTL Coding
  + Block Verification
  + Synthesis, Implementation, Revision Control
  + Block Freeze
* The place & route operation is performed by the vendor with a back-end design tool (examples: Synopsys, Magma, Vivado (?)).
* After routing, Netlist and SDF files (standard Delay format) are generated. The SDF file is the industry standard for communicating timing information between EDA tools (Electronic Design Automation)
* **Import Design Details:**
  + DRC: Design rule checking
  + STA: Static timing analysis: After routing, a static timing analyzer determines if the design still meets all its timing goals.
  + Gate-level SIM: To verify that the device functions properly at MIN and MAX timing. Serves as a second check on the STA.

**Lecture 2 – Verilog HLS Modeling 1**

* Why HDL? To model intended operation of a piece of hardware
* Notes on history in the slides, probably not important.
* Verilog is closer to C compared to VHDL – personal preference for people with C background.
* **Verilog vs. VHDL**
  + VHDL: supports custom user-defined data types. Verilog: data types are defined by the language, not the user.
  + Behavioral level of abstraction: Verilog is closer to RTL compared to level, VHDL is closer to the system. (SystemVerilog solves this)
* Levels of Abstraction (in order from highest to lowest)
  + Any code that is synthesizable is called RTL code.
  + **Behavioral/Procedural Level:** Model the hardware behavior directly
    - May/May not be synthesizable
    - Preferred for test benches
    - How to identify? **always** blocks
  + **Dataflow Level:** Directly specify data flow
    - **\*** Stalling (# <time to stall>) is not synthesizable, it’s ignored.
    - How to identify? **assign** statements
    - Sometimes called RTL
  + **Structural/Gate Level:** Model gates, intermediate connections. Circuit is described by logical links and their timing properties.
    - Transistor-level/Switch-level
    - How to identify? Direct gates.
  + Textbooks often use confusing definitions. Anything that is not behavioral and not gate level is data flow level.
* \* For coding:
  + Always try to register all outputs
  + Outputs can be saved in wire variables, since a wire type must be **continuously** driven by a register/gate, an assign statement, or an instantiated module.
  + Inputs to modules should be wires, so that when instantiated, inputs can be passed on from registers.
  + As a consequence, assign should only be done to wires, or inside one of the following
    - Always, initial, force
  + Initial blocks start at t = 0.

**Lecture 3 – Verilog HLS Modeling 2**

* Operators in slides
  + ==, ||, !=
  + ===, !==
  + ~ bitwise negation, ! logical negation
  + Concatenation: A = {3{b}, 2’b10} (3 copies of b, followed by 2 binary bits)
* **\*** For coding:
  + assign should only be done to wires, and inside one of the following
    - Always, initial, force
  + All procedural statements (list below) should be done inside an always or initial block:
    - Begin-end
    - If-else
    - While, repeat, for, forever
  + Initial blocks start at t = 0 – synthesizable.
  + force signal = 0; release signal; (this can be used to force something to a certain value, primarily useful for test bench code)
* Traditional programming languages are sequential, Verilog (and VHDL) is concurrent
* In C, code starts in one location. In Verilog, code starts in multiple locations – wherever there’s an initial or always block. Timing gets introduced.
* fork join block – everything inside of it is concurrent (started at the same time)
* To swap values:
  + A = B, B = A
  + A <= B, B <= A
* The first case wouldn’t swap values correctly, both A and B would have the value of B, the last second would work correctly (four queues), because they get executed concurrently
  + fork #1 A = B, #1 B = A join
  + fork A = #1 B, B = #1 A join
* The first case wouldn’t swap values correctly, race condition and ambiguity, second would work properly because of the correct inter-assignment (right hand side) delay they get executed concurrently.

**Lecture 4 – Verilog HLS Modeling 3**

* Normally timing delays or stalls are accumulated in verilog code. However, they can be made absolute (no accumulation) if encapsulated within a fork-join block. This is useful for test bench coding.
* Characteristics of a good test bench:
  + Self-Checking and reporting – should report PASS/FAIL for each test case separately.
  + Timeout in case of hung
  + Progress reporting
  + Should be primarily intended to validate functionality, we shouldn’t exaggerate the clock periods to avoid timings problems. We should use STA to validate timing.
  + All clocks must be synchronous.
  + Watchdog timer, Error Monitor.
  + Tasks for basic building blocks (especially for the first reset)
* Two models for representing sequential circuits:
  + Mealy FSM
    - The output is a function of the current state as well as the input
    - The input can directly cause a change in the output, which might violate time requirements.
    - Compared to a Moore state machine, a Mealy state machine often has less total number of states.
  + Moore FSM
    - The output is a function of only the current state.
    - As a result, the output changes only after a state transition has occurred, contrary to a Mealy state machine.
    - Compared to a Mealy state machine, a Moore state machine often has higher total number of states.

**Lecture 5 – HDL Synthesis 1**

* **Important Synthesis Support Comments**
  + Initial Blocks ARE NOT synthesizable. Inside a module, we should add a reset pin inside instead.
  + Comparisons to X and Z’s are always ignored
  + Delay information is ignored
  + For, While, and Forever Loops are supported.
    - While and forever loops MUST contain a sensitivity list (@ posedge clock), etc.
* Several phases of optimization are possible in the design workflow (for better synthesis results)
* 1) Architectural Optimization – Operator Re-ordering
  + SUM <= A + B + C + D
  + If same delay for all inputs, one can optimize for speed by initiating two adders at the same time (A + B first, and C + D second in parallel, then adding the two).
  + If there’s a late input, having it happen at the end is beneficial.
* 2) Logic-level Optimization
  + Factor logical expressions
  + Extract common expressions, register outputs
* 3) Design Partitioning (speeds up compile process, simplifies constraint and script files for synthesis)
  + Keep related combinational logic together in the same module
    - Accounts for multi-cycle paths better
    - Poor partition leads to bigger I/O delays in the STA stage.
  + Merge sharable resources in the same module
    - Promotes resource reuse when possible
    - Poor partitioning leads to added glue logic, bad area/timing performance.
  + Merge user-defined resources and the logic they drive into the same module
    - Example – error resource and case statements on that error should be kept in the same block
    - Poor partitioning leads to unnecessary resource duplication
  + Separate logic with different design goals into separate blocks
    - Critical Path module should be kept away from off critical path module
    - Poor partitioning leads to bad area performance
  + Separate logic with different strategies into separate blocks
    - Error circuitry block should be separate from other random logic
  + Eliminate glue logic
    - Gates should NOT be present inside glue logic circuitry, makes efficient area performance an issue
    - Gates at leaf hierarchical levels only
  + For each block, the overall gate count should be comparable
    - Block A 1000 gates, block B 50,000 gates is acceptable
    - Block A 50 gates, Block B 100,000 gates is NOT acceptable.
  + Register all outputs (next Lecture covers this in more detail)
  + Keep point-to-point exceptions within the same module
  + Derived clock sources should be separate, and available as a port.
  + Separate negative edge and positive edge flip flops.
  + Isolate State Machines
    - Moore machines are better for timing, because they change state on clock edges only.

**Lecture 6 – HDL Synthesis 2**

* **Rise Time:** Time taken by a signal to go from low level (10%) to high level (90%) of the signal amplitude.
* **Fall Time:** Time taken by a signal to go from high level to low level (90% to 10%).
* **Pulse Width:** Defined as the width at 50% signal amplitude.
* **Note:** Rise & Fall times are 0/instantaneous in simulations, meaning that transitions are sharp. They don’t have to be the same.
* **Flip Flop Timings (assume +ve edge triggered):**
  + **Setup Time:** The time period before a triggering clock edge at which the input to a flip flop needs to be stable for correct sampling.
  + **Hold Time:** The time period after a triggering clock edge at which the input to a FF needs to be stable for correct sampling.
  + If there’s a violation, the flip flop can not guarantee correct operation.

**Diagram, schematic

Description automatically generated**

Figure 1: Hold Time Violation

Diagram, schematic

Description automatically generated

Figure 2: No violations

Diagram, schematic

Description automatically generated

Figure 3: Setup Time Violation

* + **Propagation Delay:** A flip flop requires a certain amount of time period before it can update it’s output (after seeing an input that doesn’t violate any timing reqs). This delay in the output change is called the propagation delay. Typically, there’s two different propagation delay values, low-to-high, and high-to-low.
* Flip flop set up times and propagation delays play a pivotal role in deciding the maximum clock frequency (or minimum clock period) of any sequential logic circuit.
* There are four types of paths in circuit designs that must be constrained for setup times.
  + Register to Register (Internal)
    - We only need to provide the **clock period** to the synthesis tool.
    - Defining the clock in a single-clock design constrains all timing paths between registers for a single cycle setup time analysis.
    - Uncertainty (delay difference between two clock branches), Setup time, source latency (time from clock source to destination), Sk etc. can be given to the synthesis tool to check for timing violations
    - Assume a circuit with FF1 -> Comb Logic -> FF2
      * Equation:

* + - * The sum of delays for both the cases (input to output, output to input) should be equal to or less than the time period.
  + Input to Register
    - In addition to providing clock period, latest arrival time or **input delay** is needed
    - We tell how much time is used by external logic: set\_input\_delay -max N.
  + Register to Output
    - Clock period + Output delay is needed
    - set\_output\_delay -max N
  + Input to Output directly
    - Only Input + Output Delays are needed
    - set\_input\_delay -max 2 -clock CLK [get\_ports Input2]
    - set\_output\_delay -max 2.5 -clock CLK [get\_ports Output2]
  + Useful tutorial for timings: <https://vlsitutorials.com/constraining-timing-paths-in-synthesis-svpart-1/#:~:text=For%20constraining%20register-to-register,single%20cycle%20setup%20time%20analysis>.
  + **Note:** “max” = setup time checks, “min” = hold time checks.

**Lecture 7 – Verilog Coding Style/FPGA Structure/Midterm Review**

* Two types of resets, synchronous and async. Async means that reset pin is part of the sensitivity list and the module can reset independent of the clock.
* The name of the input signal doesn’t matter, it’s the connection. For example, if reset signal is connected to the clock port of a FF, it’s acting as a clock and not reset.
* If the same port can be written into in separate places in the code, this is a **double drive** ERROR. Wouldn’t compile/synthesize.
  + Assigning to the same target from multiple sources is **WRONG**.
* To avoid unintentional latch creations by the compiler, we should always:
  + If there’s an “if” statement, always have an “else” statement. If there’s a “case” statement, always have a “default” statement.
  + Always cover assignments to the entire sensitivity list in all if/else cases or case statements.
* Code a D\_FF

Text

Description automatically generated

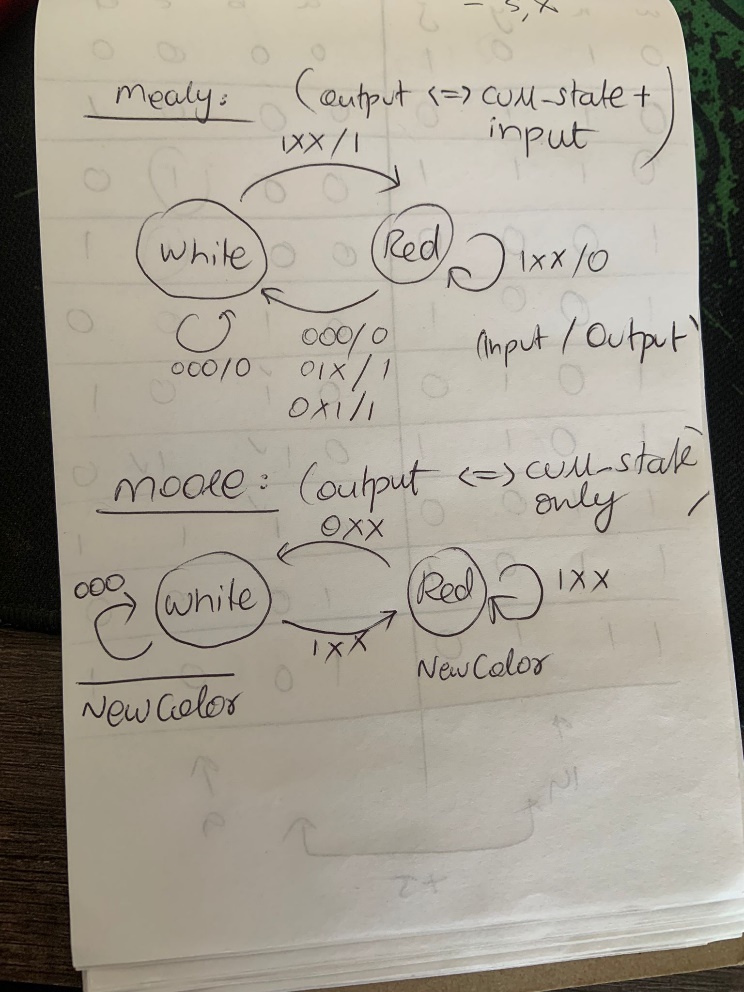
* **Note:** A n-bit ripple counter has n-clocks, a regular counter has one.
* Look up exercises on determining whether or not code will create a latch, if yes, how many, from lecture slides.

**Lecture 8 – Static Timing Analysis (STA)**

* STA is preferred over DTA because of higher coverage, also because it’s independent of input vectors.
* The tool we can use is PrimeTime (but now we will be using Design Vision for this course)
* What is STA? It’s a method of validating the timing performance of a design by checking all possible paths for timing violations
* The tool:
  + breaks the design down into a set of timing paths
  + calculates signal propagation dela along each path.
  + Checks for violation of timing constraints inside the design and at the I/O interface.
* What can the STA tool PrimeTime do?
  + Timing Checks (STA)
    - Setup and hold checks
    - Clock pulse width checks
    - Clock gating checks
  + Design Checks (DRC)
    - Unclocked registers
    - Unconstrained timing endpoints
    - Master-slave clock separation
    - Multiple clocked registers
    - Level-sensitive clocking
* PrimeTime can’t do board-level or transistor-level analysis.
* Design Vision - GUI for Design Compiler + design viewing
* PrimeTime - static timing analysis
* Refer to Timing Analysis Flow and Methodology
* Input file format can be: db, Verilog, vhdl, or edif.
* EDIF (Electronic design interchange format) is a vendor-neutral format in which to store electronic netlists and schematics.

**Notes From assignments:**

* KMAP Rules
  + Groups have to be in powers of 2
  + As long as possible
  + No group should contain a 0
  + All 1’s should be covered
  + No group is needed if all 1s have already been covered.
* FSM
  + Two always blocks
    - One for sequential state registers (current state = next state, reset)
      * Remains unchanged regardless of the type of FSM
    - Another for next state combinational logic + output logic (can be separated)
  + Mealy: Output is a function of current state + input
  + Moore: Output is a function of current state ONLY



Text

Description automatically generated