ECE 337 | Lecture 1

RTL Design

- Diagrams used
  - Schematics (FFs + gates) - for small detailed design
  - Simple RTL Block Diagrams (one level) - for basic seq. functions.
  - Hierarchical RTL Block Diagrams (multi-level) - entire system designs.
  - Functional Block Diagram (one level) - top level view of system
- RTL = Register Transfer Level.
- Simple RTL Diagrams - can include gates/FF as needed, but usually no.
  - Consists of 2 parts
    - X Registers - has clk & maybe rst
    - X Comb Logic - no clk, no rst, no feedback
  - Helps planning verilog code in a module.
  - Use arrows
- Hierarchical RTL Block Diagrams - can use gates/FF as needed, but usually no.
  - Has 3 parts
    - X Registers
    - X Comb Logic
    - X Blocks for subsystems
  - Represents your entire sys.
  - Clk & rst are external inputs
  - Need to add lower level diagrams for blocks
- Func. Block Diagram - Top level & omit clk & rst
- Diagram Process
  - Identify needed functions
  - Identify I/O
  - Describe seq. of op. (Pseudocode, flowcharts etc.)
  - Identify Data & state to be saved
  - Adapt from similar diagram if possible.

ECE 337 | Lecture 2

Verilog Review

- Tricky Concepts
  - Blocking vs Non-blocking assignment
    - x Blocking - immediately assigned value (comb blocks) (=)
    - x Non-blocking - executed at end of block (for FFs) (<=)
  - Reg vs Wire
    - x Wires - relate to wires, can only be given values from a port or assign
    - x Reg - can only be modified in 1 block and 1 procedural block reps. storage elements.
  - Data Types
    - x Logic: 0, 1, X, Z (X: unk., Z: high imp.)
    - x Wire: defaults to logic, driven by mult sources, X > 0/1 > Z (precedent)
    - x Logic (SV) - can replace wire/reg & prevents multiple drivers
    - x X - can set if mult. drivers, not reset or no store val, latch or metastable, gate level: timing hazards, transition 0 & 1, min/max prop delay
    - x Z - can set if no driver, explicit assign, gate level: output of tristate buff/open drain buff.
  - Latches (Inferred)
    - x Bad Design, has times where vars are undef. in code → latches onto values.
    - x Values should only update by clock or always (comb)
- Sequential Logic (uses FF & latches (Avoid latches))
  - Don't make comb feedback loops
  - Using always_FF @ ( ) and always_comb (maybe always_latch)
  - Storage needs a reset (check reset val and mod in Verilog)
  - Initial values for vars, wires, in always/initial doesn't exist in hardware
- Test benches
  - How to decide what to test
    - x Study Design specification (i/os, feature, funct., timing req.)
      - tb should test all regs.
    - x Study Code & State Diagram (branch conds, i/o, regs, etc.)
      - tb should take every branch and toggle a wide range of reg vals, including edge cases.
    - x Cover simple regs., unknown cases etc.
    - x Try to break it even over simple things like a hacker.
    - x Manage timing for testing & use coverage reports.

ECE 337 | Lecture 2 cont'd

Verilog Review cont'd

- Test Benches cont'd
  - We can draw seq. diagrams to see if we're testing right.
    - ex. Tb stim → DUT → checker → check (t=0)

- Unpacked Arrays vs Packed
  - Unpacked = elems are not next to each other
  - Packed = elems are adj. in mem.

- Coding Test Benches
  - Skeleton Code

```
`timescale 1ns / 10ps
module <name> ();
    <Parameters>

    <Var Declarations>

    <Clock Generation>
    always begin
        tb_clk = 1'b0;
        #(CLK_Period/2.0);
        tb_clk = 1'b1;
    end #(CLK_PERIOD/2.0);

    <Tasks>
    <Module> #(<Parameters>) DUT (<Signals>);
    initial begin
        <Variable Init>
        <Tests>
        $finish
    end
endmodule
```

- Usually tb_[name] → Decl. w/ localparam or used for params and such like Clk Period
- tb_vars
- Creates tb_clk signals
- Funcs to help test inst tested module give all tb vars a value
- the tests
- can use $stop but usually for err, and $finish is for success

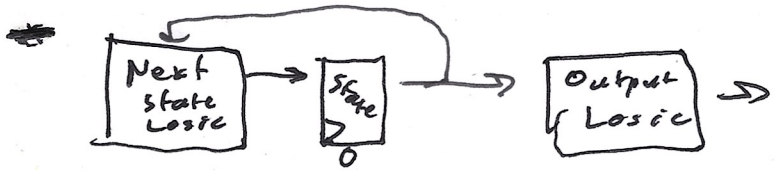
- May have waveform output (used for some simulators)

```
initial begin
    $dumpfile("<waveform.vcd>");
    $dumpvars;
end
```

ECE 337 | Lecture 3

State Machines

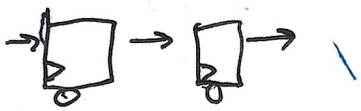
- Use Moore Machines ✱
  - Mealy produce long delays & can cause comb. feedback loops.
  - Moore is easier to test & debug
  - Mealy also can prop. glitches from i → o, slowers max clk speed)
- Can use typedef enum logic [N:0] {...} State to make it easier. Use these blocks:



  - Two comb blocks & always_FF is common.
  - Cases can be useful.
  - default state prevents latches.

Controllers

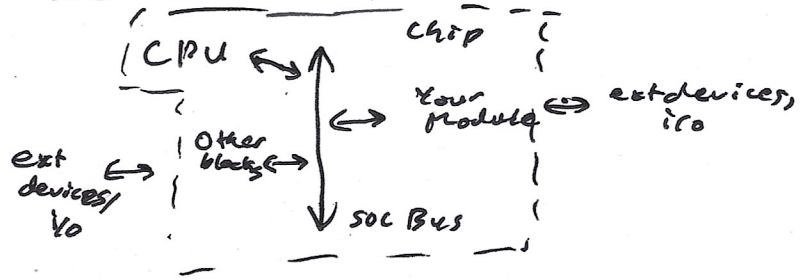
- Usually use FSMs, Synchs, Clock Dividers, Timers, etc.
- Synchs. Prevent metastable propagation by send values on the clk.



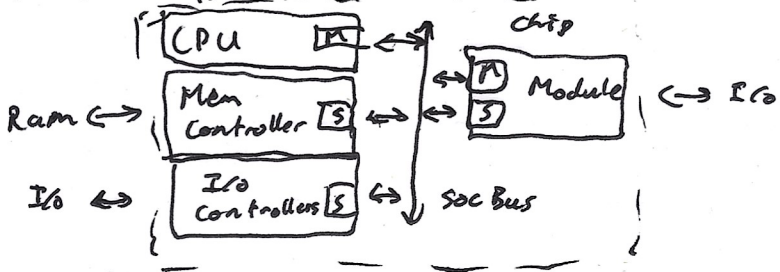
- Clock Dividers - slows down operations for processing
  - Just count and toggle output for a clkdiv (can just pulse for ez detect)
  - All blocks should use SysClk, and use clkdiv as just an input.
- Tips
  - Pulse for timers & clkdiv to prev edge detect needs.
  - Chain clkdivs for more & better division.

SoC Design

- Concept: SoC - System on Chip



- Minimal Architecture:



  - CPU - all SoCs have a processor.
    - ✗ All i/o to CPU goes through bus except clk, reset, + interrupts
    - ✗ CPU manages mem + i/o through bus
    - ✗ IRL might have mult. clks so they stay fast everywhere.
  - SoC Bus - basic is just a bunch of wires for signals + data
    - ✗ Sometimes includes address decoding logic + multiplexing logic to avoid tri-state wires
  - Bus Manager (sometimes master) - module used/built into components to issue commands to the Bus
    - ✗ The simplest SoC only have one manager
  - Bus Subordinate (Sometimes slave) - module used/built into comps. to respond to commands from the Bus.
    - ✗ Most modules fall here. In a SoC.
  - Your Module - will usually at least need sub. interface.
    - ✗ Can have manager too.
  - Memory controller - used to connect bus to RAM
  - I/O controller - used to connect i/o to bus. (ex. GPIO, SPI, I^2C, etc.)
- SoC Bus - primary source of communication on a SoC
  - Many standards, we cover AHB-lite + APB.
    - ✗ AHB-lite (Advanced High performance Bus) - simplest high speed bus standards for ARM processors.
    - ✗ APB (Advanced Peripheral Bus) - even more simple ARM standard, but can only be used for low speed I/O.
    - ✗ Uses M/S interfacing, address space, mem. map, multicycle vs pipeline bus.

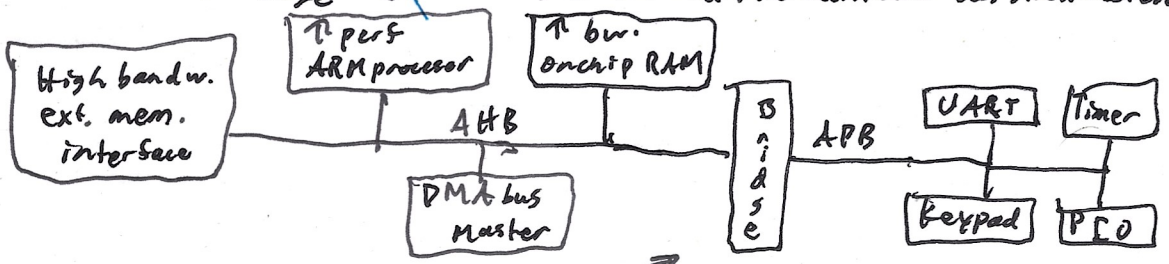
SoC Design cont'd

- M/S interfaces
  - Major signals on SoC bus - clk, address, wdata, rdata, w enable, renable, wait.
  - write trans.
    - ✕ M writes address, w.e., and w data
    - ✕ S recognizes address & accepts data / assert wait till ready.
  - Read Trans.
    - ✕ M writes address, r.e.
    - ✕ S recognizes address & writes data / assert wait till ready.
- SoC uses address to direct data - often part of it is the dest. and the other is loc in dest. (sometimes ↑ and ↓ respectively)
- Multi cycle SOC bus vs Pipelined SoC bus
  - Multicycle SoC bus - only one trans. can be in process at a time
    - ✕ Takes 2+ clk cycles, one for master & one for slave.
      - More if slave sets wait high.
    - ✕ Depending on bus edge sens (↑ or ↓) or level sens (this & or low) will vary.
  - Pipelined SoC bus - supports multiple types of transactions.
    - ✕ Has two phases like multicycle - address phase & data phase
    - ✕ Data phase overlaps w/ next address phase
    - ✕ wait extended current phase (data+addy) until set low.

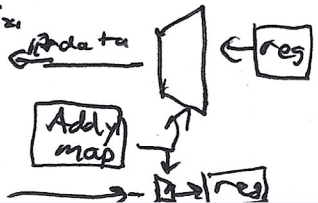
Simple slow Similar to APB

High Performance Similar to AHB/AHB-Lite

- Example of bus usage (AMBA 2.0) (Advanced Microcontroller Bus Architecture)

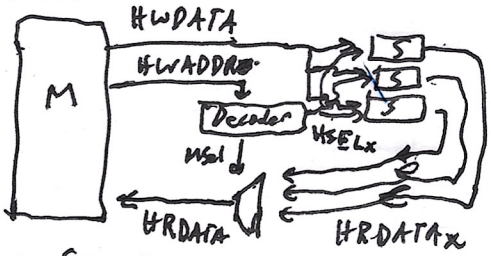


- APB
  - Signals (APB bridge)
    - ✕ PCLK - from clk source - is clk
    - ✕ PRESETn - reset - from sys. bus equivalent
    - ✕ PADDR - the address for the transaction - from APB bridge (up to 32b)
    - ✕ PSELx - select signal - generated by APB bridge (1 for each "s")
    - ✕ PENABLE - indicates second half of APB transfer - from APB bridge.
    - ✕ PWRITE - high for a write, low for a read - from APB bridge
    - ✕ PWDATA - write data - from APB bridge (up to 32b)
    - ✕ PREADY - wait signal when low - from sub interface.
    - ✕ PRDATA - read data - from sub interface (up to 32b)
    - ✕ PSLVERR - transfer failure - from sub interface (not req., tie low if unused)
  - Ex:



SoC Design cont'd

- AHB-lite
  - Signals
    - X HCLK - from clk source - is clk
    - X HRESETn - reset - from reset controller
    - X HADDR - address for the transaction - from master (32b) (modifiable)
    - X HBURST - burst type to indicate if its part of a burst (length etc) - from master (3b)
    - X HSIZE - size of the transfer - from master (3b) (modifiable)
      - usually byte, halfword, & word.
    - X HTRANZ - transaction type (IDLE, Busy, NONSEQ, SEQ) - from master (2b)
    - X HWDATA - write data - from master (32b) (modifiable)
    - X HWRITE - high for writes, low for read (stays the same for entire burst) - from master
    - X HRDATA - read data - from 's' (32b)
    - X HREADYOUT - ↑ trans ready, ↓ to stall - from 's'
    - X HRESP - ↑ for trans err. - from 's'
    - X HSELx - 's' select sig (one for each 's') from decoder
    - X HMASTLOCK - locks trans indicator. - from master - for multi m. systems
    - X HPROT - indicates protection levels - from 'm' (4b)
  - Model:

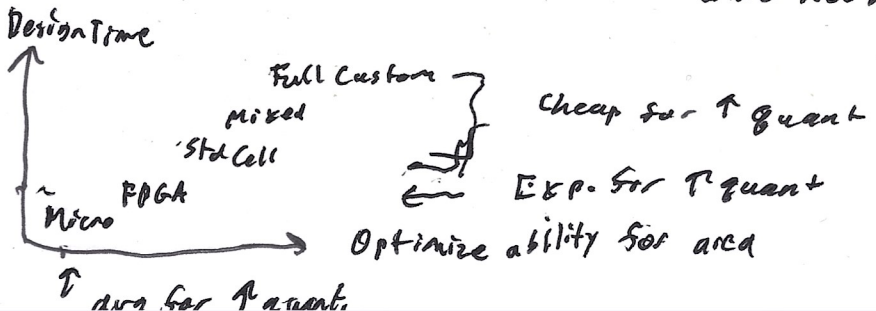


For multiple 'm's use a 'multilayer interconnect' that implements left but more complicated for more devices

  - Ex: Similar to APB but reg between addy map+reg and hwdata out. also add multiplex w/ addy map (select) & regs.

ECE 337 | Lecture 5

ASIC Implementation Options

- Options: Standard Cell ASIC, Full Custom ASIC, Gate Array, Field Programmable Gate Array (FPGA), Microcontroller.
- Standard Cell ASIC
  - Library of pre-designed logic "cells"
  - place+route software positions + connects cells
  - Pros: Cheap in ↑ quantities, Design is easier than Full Custom, Easier to reuse IP vs Full Custom
  - Cons: Expensive in ↓ quantities, expensive to change chips, very inefficient for some functions.
- Full Custom ASIC
  - Draw transistor schematics, layout by hand.
  - Pros: Cheap in HUGE quantities, has best area/speed optimization
  - Cons: ↑ time + ↑ cost to design, harder to reuse for other tech.
- Mixed Custom + Standard Cell
  - Used Custom for analog blocks (or option) + blocks where Speed/size is critical.
  - Option is common for SoCs.
- Gate Array - not used that much
  - Transistors/Logic gates on silicon
  - We design interconnect by adding metal
  - compromise between ASIC + FPGA
- Reconfigurable Devices (FPGA/ Complex Programmable Logic Device (CPLD))
  - Layout of Logic + wires is fixed
  - Synthesis + mapping software programs how wires are connected + logic
  - Pros: Very low time to market, quick + easy to prototype/change
  - Cons: Slower + larger than ASIC, ↑ expensive in large quants.
- Microcontrollers
  - Used mostly by software for small complexity tasks
  - Used for embedded systems w/ arch. based on SoC.
  - Pros: More flexible, easier + quicker than FPGA, quick + easier to prototype/change, smaller than FPGA
  - Cons: Slower than everything, doesn't exploit parallelism + has fetch + decode. Some glue logic + interface hardware needed.
- 

ECE 337 | Lecture 6

Timing Analysis and Critical Paths

- Important terms
  - Timing/delay path - logic propagation path from i/o of comb block
  - Critical Path - timing path w/ longest/shortest delay
  - Static timing analysis - analyze delay by tracing timing paths
  - Dynamic timing analysis - analyze delay by simulation + test vectors
- We need analysis to decide clock speed & timing constraints
  - slowest path constrained by clk speed, t_{pss}, t_{setup} ← time before clk data valid
  - fastest path constrained by clk skew, t_{pss}, t_{hold} ← time after clk data valid
    - t_{pss} is prop. delay of FF
    - ↑ diff in clk due to path len
  - FF out can be used for i and FF in can be used for o.
  - Worry abt t_{hold} when comb delay + t_{pss} < t_{hold} + clk skew
- Timing reports can be generated, but not fixed till placement & route
  - Includes start, end, & crit path delay
  - Try to analyze rtl and verify w/ report