ECE260C
Spring 2022

VLSI Advanced Topics

Introductions

Introductions

- **Prof**
  - John Eldon jeldon@eng.ucsd.edu
- **TA**
  - Xinyue Wei xiwei@ucsd.edu

John A. Eldon, DEnv
Continuing lecturer, CSE, ECE, MAS

- Graduate student intern, then research scientist, Data Sciences Division of Technology Service Corporation, doing classification and regression on "Big Data" back when it was small; doctoral thesis: oxidant-precursor (smog) modeling
- Semiconductor industry (1980-2015): applications engineer, manager of advance development, consultant/contract engineer
  - VLSI-DSP / computer arithmetic
  - digital video processing
  - mixed signal analog/digital
  - high speed communications
  - embedded hardware/software systems

John Eldon, continued

- Taught Verilog at UCSD Extension since 1993, adding analog design, mixed signal design, UVM-verification, and signals & systems.
  - Currently teaching: mixed signal; signals and systems/DSP
- Encore career at UCSD, teaching various courses in
  - CSE: 30, 140, 140L, 141, 141L
  - ECE: 5, 30, 111, 165, 260A, 260C
  - Master of Advanced Study/Wireless Embedded Systems:
    - 237B: Software for embedded
    - 269: Hardware implementation of embedded
    - 237A: Embedded projects (PYNQ)

(Virtual) Office Hours

- Google calendar link on Piazza or Canvas
- Prof office hours on Zoom
  - Right after class
    - M, W, F, 1000-1100
  - Plus by appointment, also on Zoom
    - Best times to catch me are weekday and weekend afternoons
    - email me or post a request on Piazza

Housekeeping

- All sections are delivered on Zoom
- Lecture M, W, F 0900
- Real time Zoom attendance is optional, but highly encouraged.
- If you miss a class, be sure to catch up by watching the recording.

Course Logistics

- Lab exercises (individual or small group)
- Project (larger lab exercise)
  - ***OPTIONAL***: choose your own term project instead
- No exams (unless you beg for one ☺)
- Content:
  - SystemVerilog-based simulation
  - ASIC and FPGA logic synthesis
  - Universal Verification Method (UVM)

Course Outline

Part 1: SystemVerilog-based Design and Verification

1. Intro

    Terminology

    - SoC, FPGA, ASIC, HDL, Design, Synthesis, Simulator

    Full custom chip designing process and challenges

    Role of hardware description languages

    SystemVerilog Capabilities and Syntax

    Applications of FPGAs

2. ASIC, FPGA, Logic Synthesis

    Introduction to ASIC and FPGA based prototyping

    ASIC and FPGA design flow

    Introduction to FPGA Architecture and its internal components

    Logic Synthesis, Simulation, Simulator, Waveform

    ASIC versus FPGA differences

    Matlab/Simulink high level synthesis

Course Outline, Continued

- 3. SystemVerilog Modeling Abstraction
- Combinational vs. Sequential
- Behavioral vs. Register Transfer Level Data Flow Modeling
- 4. Anatomy of a SystemVerilog Module
- specification format
- ports, parameters
- hierarchies
- 5. Data Types, Continuous Assignments, Conditional Operator
- nets, wires, logics, regs, wireand (WAND)
- synthesizable, non-synthesizable variables
- signed (two’s comp.), unsigned

Course Outline, Continued

- 6. Blocking, nonblocking assignments
- RTL event scheduling flow
- inter-delay
- shift registers, Johnson counters, barrel shifters
- 7. Procedural Blocks
- initial, always
- latch, flip flop, combinational
- race conditions
- LFSR / pseudonoise generator
- 8. RTL Programming Statements

Course Outline, Continued

- 9. Finite State Machines
- Mealy v. Moore
- state diagrams to Verilog
- parity and error detection
- UARTs and vending machines
- 10. Packed, unpacked arrays
- memories
- 11. Test benches
- tasks and functions
- $stop and $finish
- $write, $display, $strobe, $monitor

Course Outline, Continued

- Part 2: Universal Verification Method

- 1. Interfaces and Bus Functional Models
- 2. Advanced OOP
- 3. Classes & Extension
- 4. Polymorphism
- 5. Static Methods
- 6. Factory Pattern
- 7. Object-Oriented Testbench Example
- 8. UVM Tests, Components, Environments
- 9. Test Bench Analysis Ports
- 10. Interthread Communication
- 11. Class Hierarchies and Deep Operations
- 12. UVM Transactions, Agents, Sequences

Part 3: Wireless Embedded Systems

- Definition
- Examples
- Design & Implementation
- Resource Allocation (Hardware Accelerators vs. Software in Microprocessor Core)
- Software Defined Radio

Lecture/demo

- SystemVerilog syntax
- SystemVerilog state machine
- Simulation: Questa demo
- FPGA Synthesis: Mentor Precision Demo
- FPGA Synthesis: Quartus demo
- Vivado demo
- Simulation & High-Level FPGA Synthesis: Simulink Demo

Tools Access

- Mathworks (Matlab, Simulink, HDL Coder, Comms & DSP Toolboxes):
  - You may download and install on your own computer
  - Use university license (VPN in to do so)

Questa Simulator (replaces ModelSim)

- Option 1: download & install on your own computer
  - https://community.intel.com/t5/Intel-FPGA-Software-Installation/Intel-Starter-Edition-2021-2-requires-License/td-p/1280757
  - Follow the instructions in Zawani_M Intel's response
  - Free, but requires license
- Option 2: hit the playground
  - Create a free account on Doulos' edaplayground.com
  - Need to add $dumpvars command to your testbench to create a signal waveform:
    - declare **$dumpfile**("*your_desired_name*.vcd");
    - in **initial begin** block: **$dumpvars**;   *// dumps everything!*

Quartus Prime Lite (free synthesizer)

- https://www.intel.com/content/www/us/en/software-kit/684216/intel-quartus-prime-lite-edition-design-software-version-21-1-for-windows.html?
- Free, no license required

- Option 2: Use Mentor Precision on edaplayground
  - Requires a run.do file with your test bench
  - (shows Codewright logo on my computer)

run.do

run.do (for Xilinx FPGA)

- setup_design -manufacturer Xilinx -family Artix-7 -part 7A100TCSG324
- foreach arg $::argv {
-   add_input_file $arg
- }
- compile
- synthesize
- auto_write precision.v
- report_output_file_list
- report_area
- report_timing
- exec cat precision.v

Application Specific Processors

- At 0.13mm, Pentium 4 die size can fit ~50 ARM9 cores, 80 at 0.10mm
- At 0.13mm at 250MHz clock, ARM9 dissipates ~0.1W. 50x = 5W
- At 0.13mm 150M transistors single chip, 250M at 0.10mm

SoC

System-on-Chip = Boards-on-Chip?

• Instead of Core5 + PCI + PCI cards + DRAM,
System-on-Chip = CPU cores (e.g. ARM)+ on-chip networks (e.g. AHB, AMBA bus) + ASIC blocks + SRAM

• Many issues are similar: e.g. need standardization process for components supplied by 3rd party to interoperate

• What's different? Components come as "Intellectual Property" (i.e. "Designs") rather than IC's

• So why not make the components customizable?

Need to fab specific instantiations

Why App Specific Processors?

- The promise: Customization enables better performance and/or better fit
  - Examples of customization
  - – Add additional “data paths”
  - – Add application-specific “data paths” and/or application specific instructions
  - – Vary processor parameters (e.g. #registers, cache sizes)
  - – Add application-specific accelerators (as “peripherals”)
    - $$$ ==> suited for high-volume production only
    - FPGAs are increasingly popular even for released products
      - weigh unit costs vs. design & development costs

Example: Tensilica's XTensa Processor

- Basic architecture
- Comparable performance of 32-bit RISC
- $0.7mm^2$ in 0.18mm
- 0.4mw/MHz

Additional Data Paths

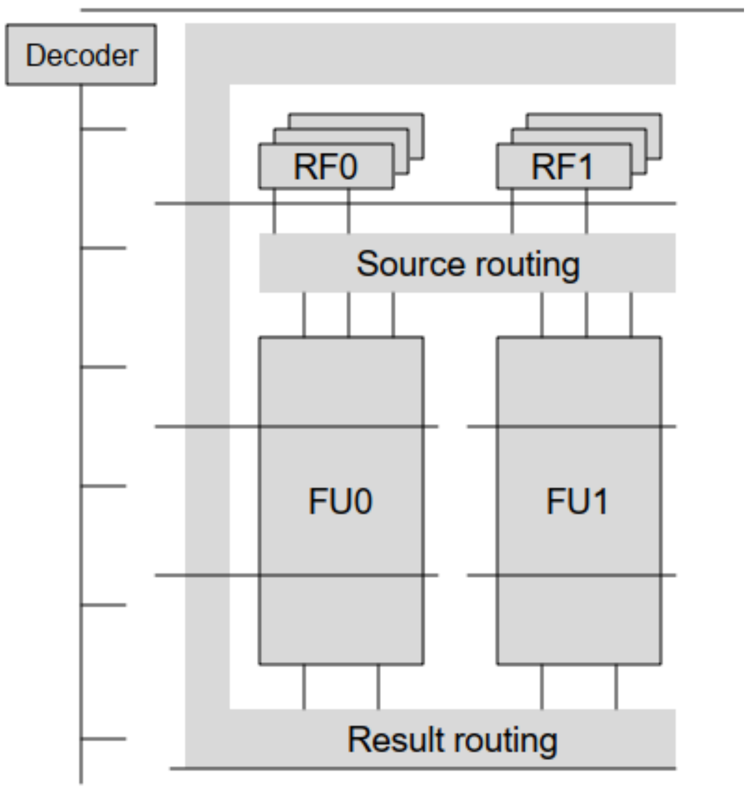
Challenge: effective use of parallel processors and register files

Note the source & result routing

Big Crossbar Switches

FU = "functional unit"

RF = "register file" / cache memory

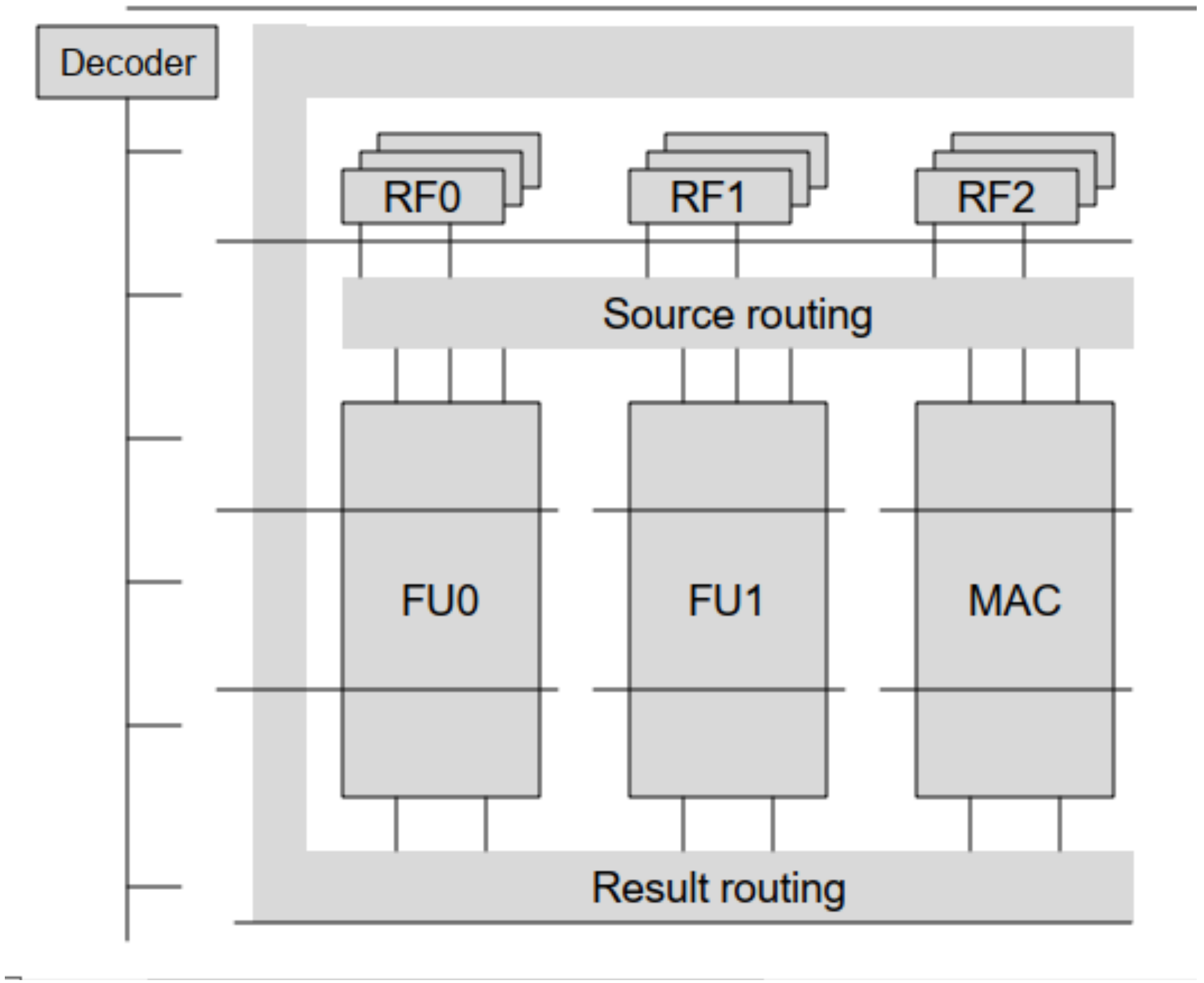


Specialized Data Paths

MAC = macaroni and cheese
... multiplier/accumulator, actually

Typical hardware accelerator for DSP

Popular as building blocks in FPGAs
Popular as IP in custom ASICS

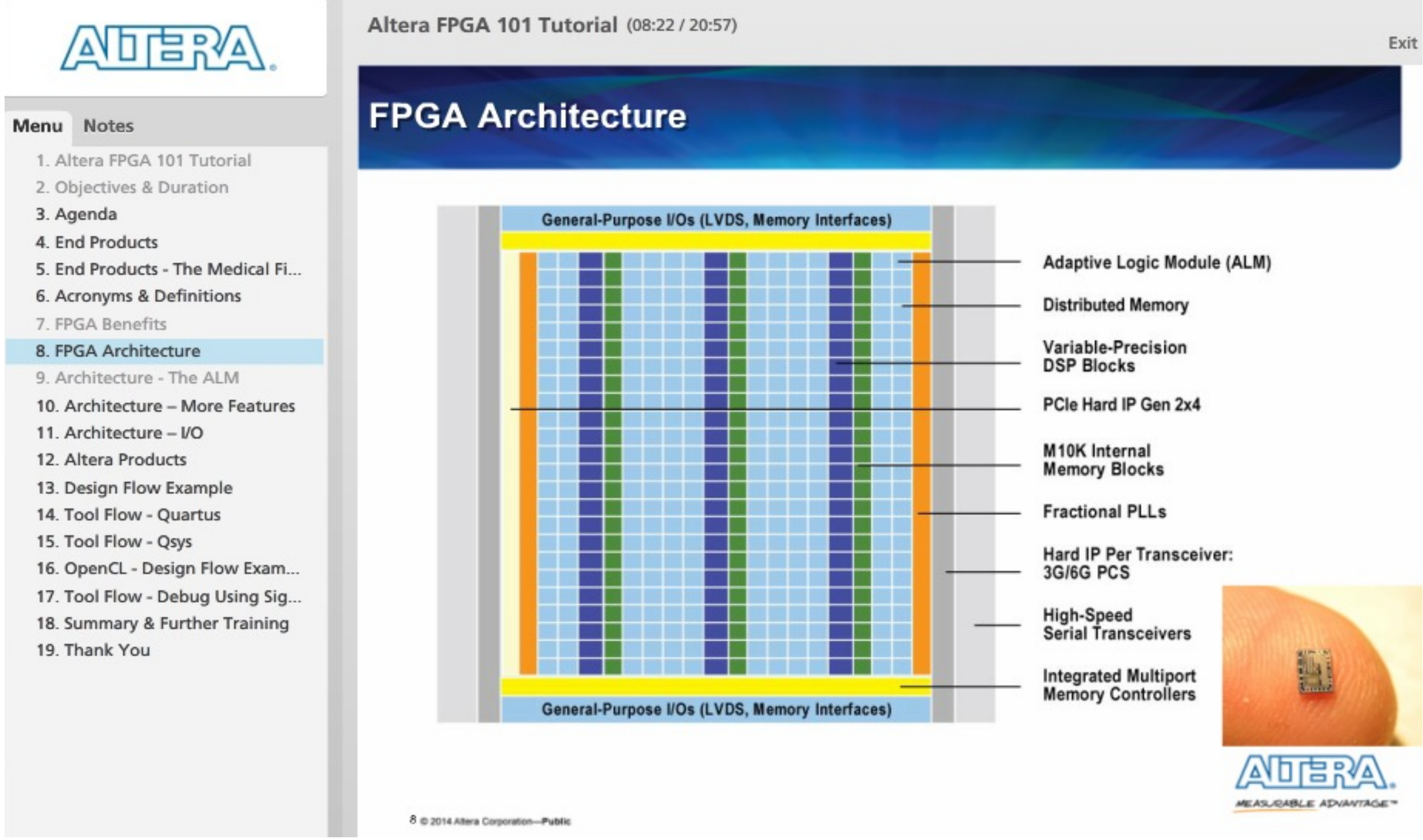


FPGA

- Field Progammable Gate Array
- Macro Architecture
  - array of logic cells, memory
- Micro Architecture
  - anatomy of a logic cell

Intel FPGA Macro Architecture

Cool Intel Trick: splittable multipliers
one 18x18 or two 9x9



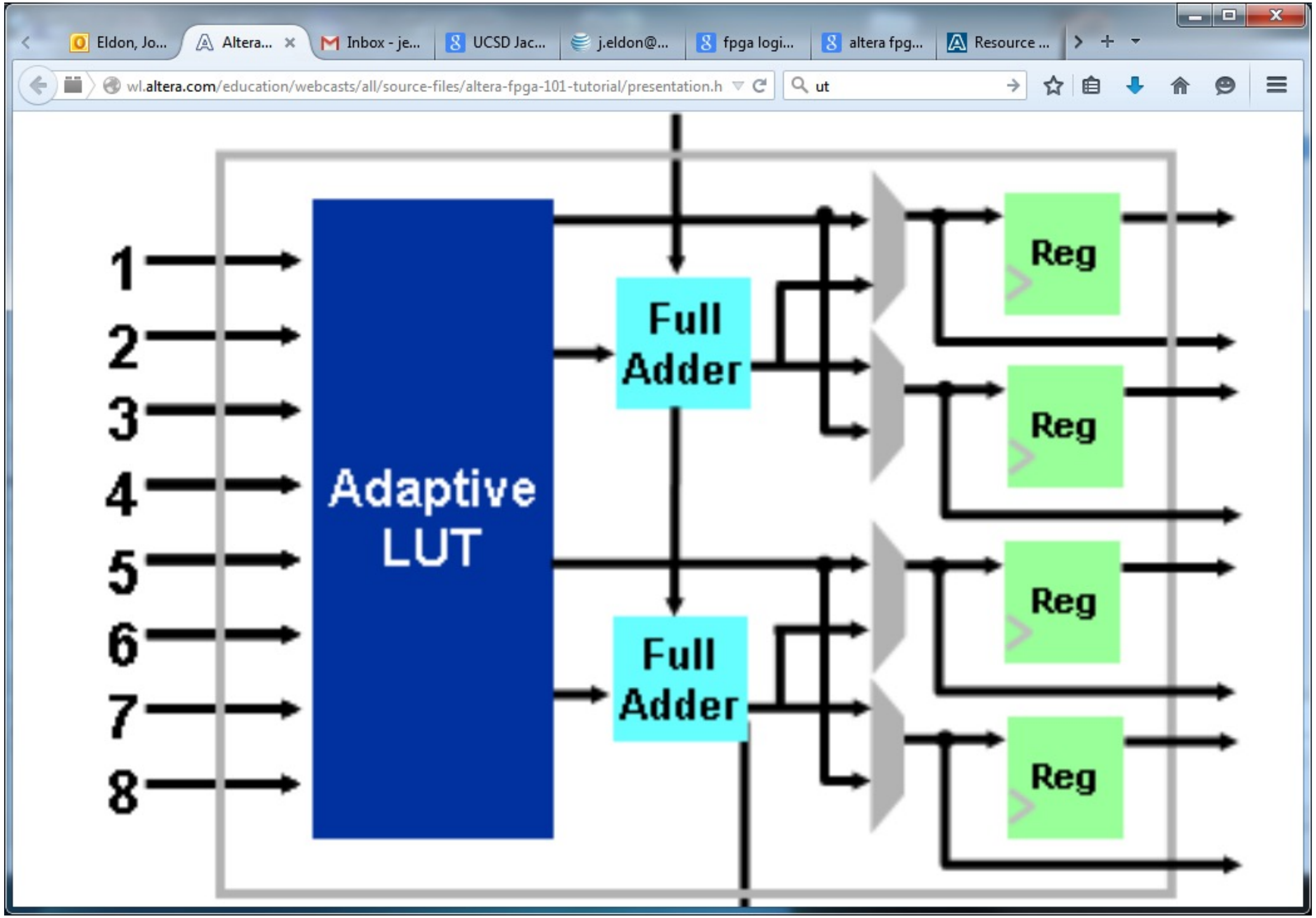
Anatomy of a Cell

Lookup Table
each out = f(all ins)

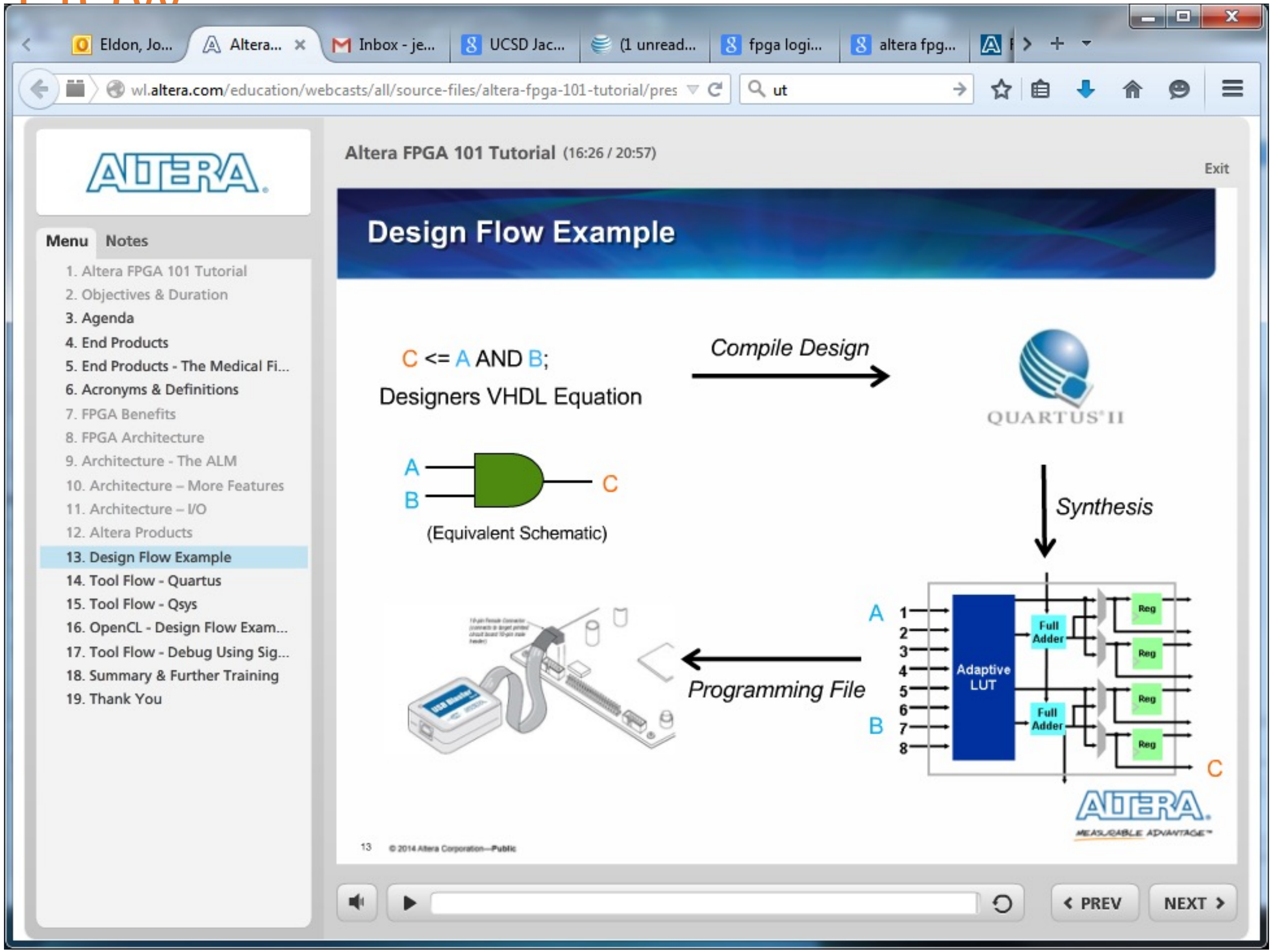
Carry Chain For Adders

Muxes For Bypass

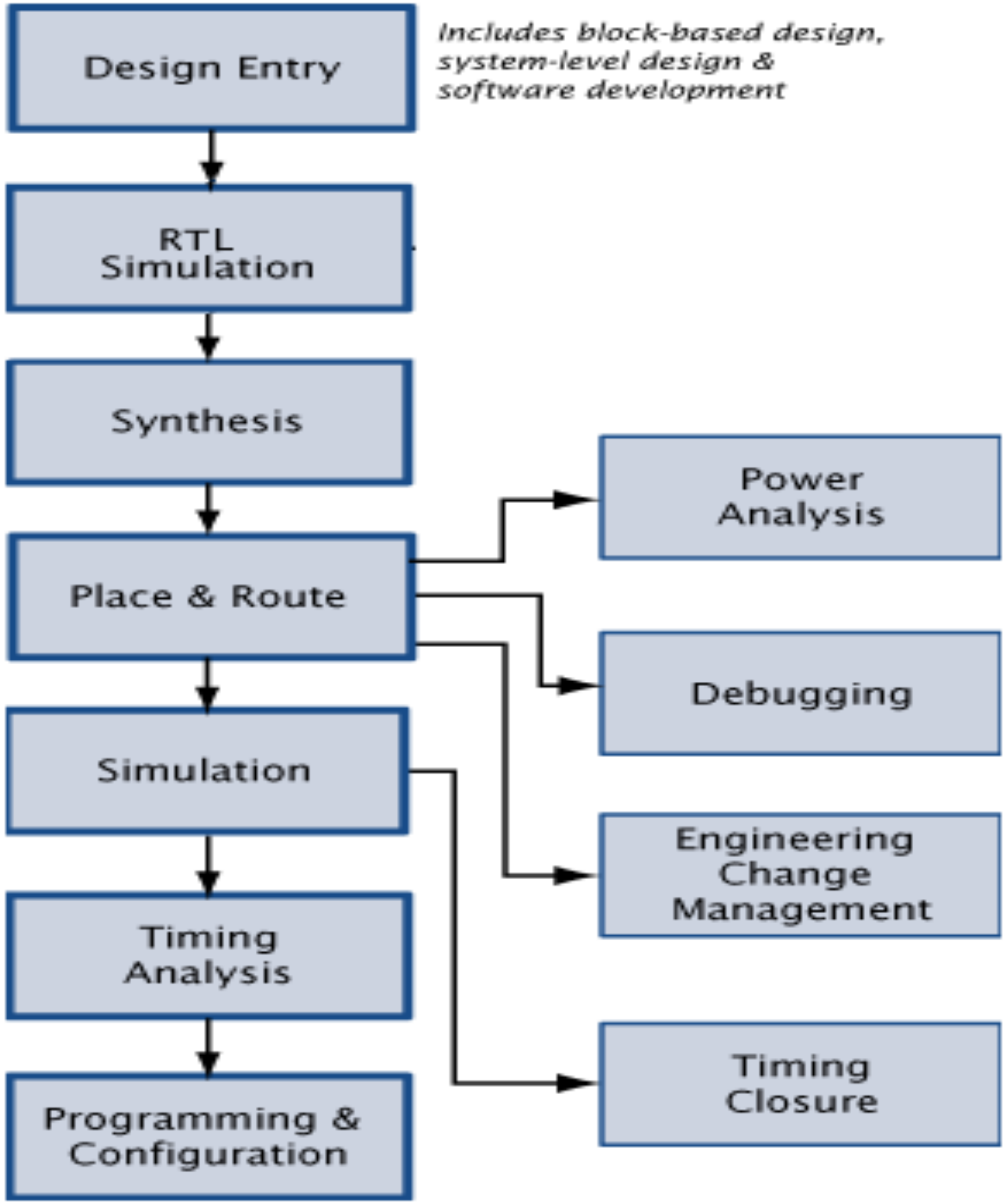
Combinational & Registered Outputs



Design Flow


Eldon, Jo...
Altera...
Inbox - je...
UCSD Jac...
(1 unread...
fpga logi...
altera fpg...
wl.altera.com/education/webcasts/all/source-files/altera-fpga-101-tutorial/pres
ut
ALTERA
Menu
Notes
1. Altera FPGA 101 Tutorial
2. Objectives & Duration
3. Agenda
4. End Products
5. End Products - The Medical Fi...
6. Acronyms & Definitions
7. FPGA Benefits
8. FPGA Architecture
9. Architecture - The ALM
10. Architecture – More Features
11. Architecture – I/O
12. Altera Products
13. Design Flow Example
14. Tool Flow - Quartus
15. Tool Flow - Qsys
16. OpenCL - Design Flow Exam...
17. Tool Flow - Debug Using Sig...
18. Summary & Further Training
19. Thank You
Altera FPGA 101 Tutorial (16:26 / 20:57)
Exit
Design Flow Example
C <= A AND B;
Designers VHDL Equation
Compile Design
QUARTUS II
A
B
C
(Equivalent Schematic)
Synthesis
A
B
1
2
3
4
5
6
7
8
Adaptive LUT
Full Adder
Full Adder
Reg
Reg
Reg
Reg
C
Programming File
ALTERA
MEASURABLE ADVANTAGE
13
© 2014 Altera Corporation—Public
PREV
NEXT

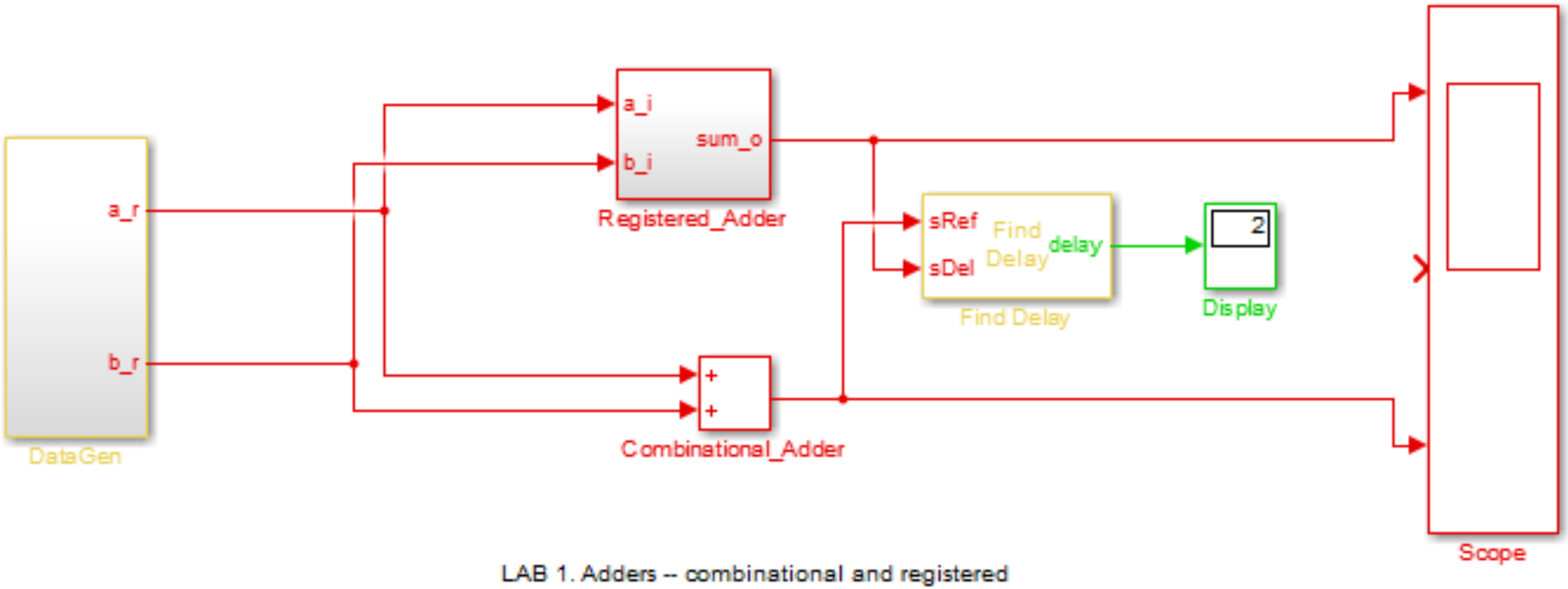
Typical Design Flow



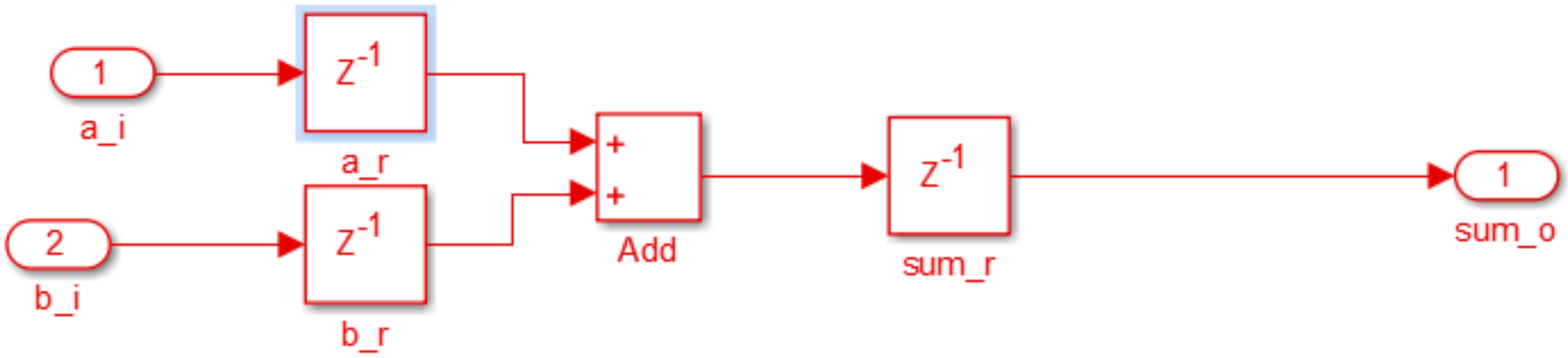
Simulation vs. Synthesis

- We **simulate** the testbench and the device under test (DUT), using ModelSim logic simulator, to VERIfy LOGic
- We **synthesize** just the DUT, never the bench, using Quartus, to build real hardware
  - In this class we used this to make sure our design is realizable in hardware and to estimate how fast it could be clocked

Test Bench and DUT



Inside a DUT (Registered Half Adder)



Z^{-1} = clocked logic (D flip flop) -- denotes sample time of delay

Verilog

- "Verify Logic“
- 1984, Phil Moorby
- Shift from schematic to text based design

SystemVerilog

- "Verify Logic" enhanced
- logic
- typedef struct
- typedef enum
- always_ff, always_comb
- array notation
- a**n and $clog2
- classes -- great for modular, reusable test benches

Questa (Siemens / Mentor Graphics)

- Simulation tool
- Behavior modeling of .sv files
  - no real sense of timing
  - verifies logical correctness only
- Post-synthesis timing simulation
  - uses .svo and .sdo files from Quartus
  - verifies logic & timing
  - beyond scope of this class ☺

Quartus (IntelFPGA)

- logic synthesis tool
- turns Verilog (.sv) into netlist (.svo)
- We need Quartus for three reasons:
  - 1. Verify that our design can be built in REAL hardware
  - 2. Obtain an estimate of how fast said hardware can be clocked/run (Lab 4)
  - 3. Estimate how much of our FPGA is needed