EE 4755—Digital Design Using Hardware Description Languages

URL: `https://www.ece.lsu.edu/v`

Instructor Information

David M. Koppelman, Room 3316R P. F. Taylor Hall

+1 225 578-5482.

`koppel@ece.lsu.edu`, `https://www.ece.lsu.edu/koppel`

Tentative office hours: M-F 14:00-15:00 (Fall 2023).

Syllabus URL

The syllabus is available via: `https://www.ece.lsu.edu/koppel/v/v.html`.

For those who prefer QR codes:

Prerequisites

Formal Prerequisite

EE 3755 (Computer Organization).

Informal Prerequisites (What you really need to know.)

C/C++ (The computer language.)

Digital hardware design: Should be able to answer these:

Is a ripple adder something to get excited about?

Does clocking a register hurt the register?

Your part of the design carries the critical path: your fault or your forté?

Digital hardware design: Should have been able to answer these:

Is a ripple adder something to get excited about?

Yes!, a ripple adder is inexpensive and it's fast enough.

No!, a ripple adder is too slow to meet specs.

Does clocking a register hurt the register?

Clocking is the operation that causes the register to remember its inputs.

That's what registers are for, so I certainly hope it doesn't hurt.

Your part of the design carries the critical path: your fault or your forté?

The critical path is the path that takes longest, so

Fault: it takes longest because I didn't do a good job. `:-(`

Forté: I'm the best, that's why they chose me to work on it. It would have taken longer with anyone else.

Graded Material

Midterm Exam, 35%

Fifty minutes, open notes.

Final Exam, 35%

Yes, it's cumulative.

Homework/Projects, 30%

Written and computer assignments.

Lowest grade or unsubmitted assignment dropped.

Electronic Design Automation (EDA) Overview

EDA Definitions

Electronic Design Automation (EDA):

The use of software to automate electronic (digital and analog) design.

Hardware Description Language:

A language used to describe hardware and to exercise, test, and verify the hardware.

Examples: SystemVerilog, Verilog, VHDL, SystemC.

Hardware Description:

Something written in an HDL with the goal of modeling or fabricating (synthesizing) hardware.

It's analogous to a procedure or program in a conventional language . . .

. . . but nothing like it at all!

Quick Examples of Hardware Descriptions.

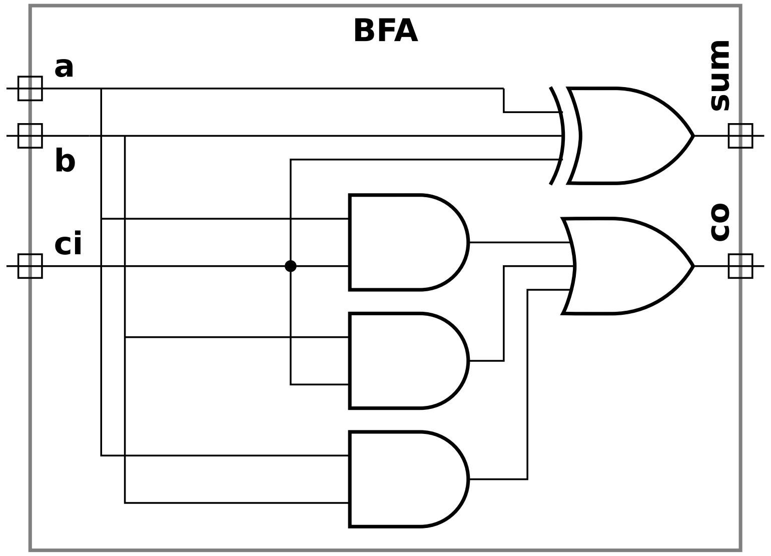
For each, will show

- A diagram.
- The description in SystemVerilog.

The Example Description Items

- *A Binary Full Adder (BFA)*
  - Adds three 1-bit quantities, `a`, `b`, `ci`, to compute `sum` and `co` (carry out).
  - Input: three 1-bit quantities, `a`, `b`, `ci`.
  - Output: one 2-bit quantity with bits `co` (MSB) and `sum` (LSB).

Description of a binary full adder:



Operation: `co, sum = a + b + ci .`

```
module bfa( output uwire sum, co, input uwire a, b, ci );
   assign  sum = a ^ b ^ ci;  // Note "^" is the exclusive or operator.
   assign  co = a & b | a & ci | b & ci;
endmodule
```

A Batcher Odd/Even Merge Module

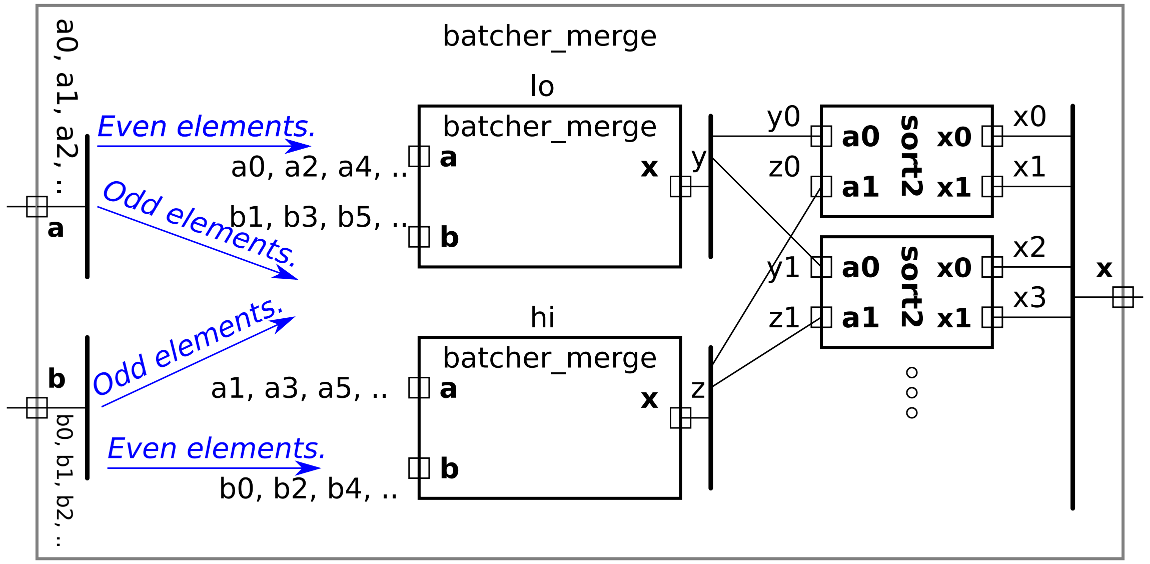
Used to merge together two sorted sequences.

A component in some advanced hardware designs.

Input: Two n-element, w-bits-per-element, sorted sequences, `a` and `b`.

Output: One $2n$-element sorted sequence, `x`.

Description of a Batcher Odd/Even merge module:



```
module batcher_merge #( int n = 4, int w = 8 )
   ( output uwire [w-1:0] x[2*n],
     input uwire [w-1:0] a[n], b[n] );
   uwire [w-1:0] xlo[n], xhi[n];
   if ( n == 1 ) begin
      assign xlo[0] = a[0], xhi[0] = b[0];
   end else begin
      localparam int nh = n/2;
      uwire [w-1:0] ae[nh], ao[nh], be[nh], bo[nh];
      for ( genvar i=0; i<nh; i++ )
        assign ae[i] = a[2*i], ao[i] = a[2*i+1], be[i] = b[2*i], bo[i] = b[2*i+1],
      batcher_merge #(nh,w) mlo( xlo, ae, bo );
      batcher_merge #(nh,w) mhi( xhi, ao, be );
   end
   for ( genvar i=0; i<n; i++ )
     sort2 #(w) s2( x[2*i], x[2*i+1], xlo[i], xhi[i] );
endmodule
```

Quick Examples' Takeaways

A reminder (or first impression) of what a Verilog module looks like.

Correspondence between a digital logic diagram and a Verilog description.

Ability to compactly describe moderate-complexity designs such as the Batcher Odd/Even Merge module.

More Definitions

Simulation:

Execution of some code which determines the state of some modeled system over time.

Most HDLs are both simulation languages and structural description languages.

Design Target:

The technology in which a design is to be implemented.

The design target is often some type of FPGA or ASIC (application-specific IC).

Synthesis [of a hardware description]:

Conversion of the description into chosen design target.

Synthesis is a multi-step process, steps include *inference* [of logic from procedural code] and *routing*.

Usage of Terminology

"I wrote a *hardware description* of my new machine learning accelerator in the *SystemVerilog hardware description language*. I then *simulated* it using the *electronic design automation tools* provided by Cadence, and it worked the first time! [Don't believe him.] Then, for a *design target* I chose an ASIC from a silicon foundry that my friend works at. She gave me a 70% discount. [Probably exaggerated.] I performed *synthesis* of my hardware description, some post-synthesis simulation to verify the timing of my design, and finally proceeded to *tape-out*. I sent the resulting design files to the foundry and got my chips back in two weeks."

Electronic Design Automation (EDA) (Longer Definition)

Electronic design in which

a hardware *description* is written in a *hardware description language*

possibly consisting of components from a vendor's *IP library*

the functionality of the description is verified by *simulation*

the formal correctness, testability, and compliance of a description is evaluated by software

and the description is converted to a manufactureable form using *synthesis tools*.

Hardware Description Languages

Hardware Description Language:

A language used for describing the structure of hardware and how the hardware should behave. Examples include SystemVerilog and VHDL.

Kinds of Digital Hardware Built Using HDL's:

Anything worth selling.

Some Popular Hardware Description Languages

Verilog and *SystemVerilog*

(Verilog was replaced by SystemVerilog in 2009.)

C-Like and C++-Like Syntax

Originated in industry in 1984, now IEEE standard.

VHDL

Ada-Like Syntax

Originated as a DoD project, now an IEEE standard.

SystemC

A C++ Class Library. Descriptions are exactly C++ syntax.

Often used with high-level synthesis tools.

History of Verilog and SystemVerilog

Verilog developed by *Gateway Design Automation* in 1984, later bought by *Cadence*.

Became an IEEE standard 1995: *IEEE 1364-1995*.

Major update in 2001: *IEEE 1364-2001*.

Minor update in 2005.

Also in 2005, related language *SystemVerilog* was standardized as *IEEE 1800-2005*.

In 2009 Verilog and SystemVerilog merged: *IEEE 1800-2009*.

Minor updates in 2013 and 2017: *IEEE 1800-2012*, *IEEE 1800-2017*.

VHSIC Hardware Description Language (VHDL)

Ada-like syntax. (Ada is a DoD-developed language for large embedded systems.)

Developed as part of U.S. Department of Defense (DoD) VHSIC program in 1983

Became IEEE standard 1076 in 1987.

Some believe that VHDL is harder to learn than Verilog.

At most one or two lectures on VHDL.

Verilog vs. VHDL

There is an advocate community for each language.

Good News: Many tools support both.

That said, SystemVerilog covered in this course.

Design Flow

Design Flow:
The steps used to produce a design, from initial design entry to the generation of the final manufacureable form. Describes which programs will be used, when they will be used, and how they will be used.

EDA tool vendors usually provide design flows that show how their products can be used.

Companies develop design flows that are used to produce their designs.

A simple design flow is described below.

Informal Design Flow

◇ Enter design. (Use your favorite text editor and HDL.)

◇ Enter *testbench* for design.

The testbench checks correctness.

This is very important—you never want to say “I thought it worked.”

- ◊ Run simulation to verify correctness.
- ◊ Use waveform viewer and other tools to find bugs. (If any.)
- ◊ Run synthesis program.

  Synthesis reports indicate area and timing.

  Not satisfied? Go to Step 1.

  Otherwise, *tape out*, download to FPGA, etc.

Informal Design Flow Demonstration

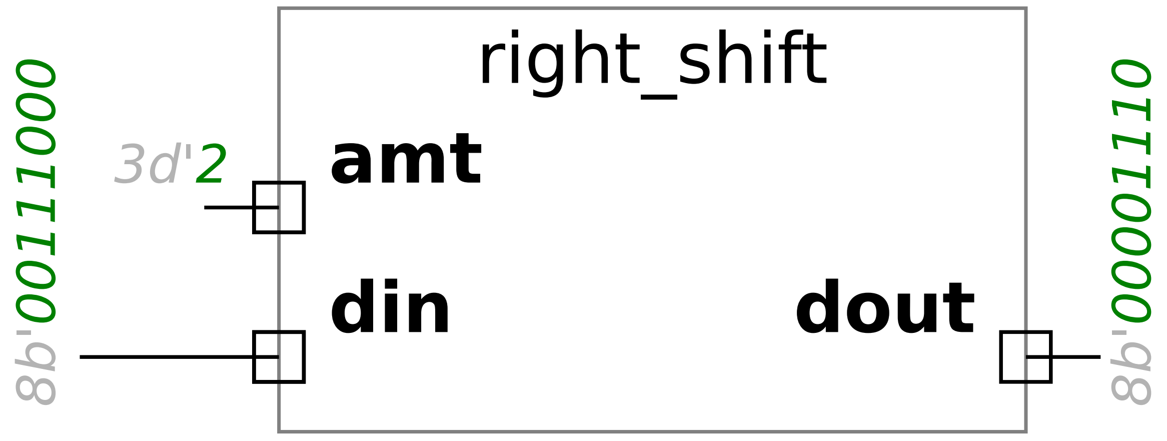
Demonstration: look at different possible logical right shifter designs.

Logical Right Shift

Inputs: amount, `amt`; unshifted value, `din`.

Output: shifted value, `dout`.

Parameter: w, number of bits.



What makes the shifter interesting.

It's used in many applications.

It's easy to describe *behaviorally* (by what it does).

A naïve implementation costs $3w^2$ gates.

A good implementation costs $6w \lg w$ gates.

Should we rely on the synthesis program to get it right?

Simple Design Flow

Three easy steps (not counting step zero).

Used to describe the major steps in a typical design flow.

List of Steps in Simple Design Flow

Simple Flow Step 0: Goal Determination.

Simple Flow Step 1: Design Capture

Simple Flow Step 2: Behavioral Verification

Simple Flow Step 3: Synthesis and Timing Verification

Simple Flow Step 0:

Start with: an idea for a new chip.

Goal: a box full of the new chips.

Simple Flow Step 1: Design Capture

Using the back of an envelope or some other suitable medium . . .
. . . develop a rough draft of the design.

Using a text editor . . .
. . . write a *Verilog description* of the design.

Using a text editor . . .
. . . write a *Verilog description* of a *testbench* used to test the design.

The testbench generates inputs for the design and verifies the design's outputs.

Simple Flow Step 2: Behavioral Verification

Using a *simulator* and *waveform viewer* . . .
. . . check if design passes testbench tests . . .
. . . and if not, debug.

Waveform viewer is sort of a virtual logic analyzer, can view signals on any part of design.

Simulator output includes messages generated by behavioral code . . .
. . . including “pass” or “fail” message produced by testbench.

Using text editor . . .
. . . fix bugs, and tune performance.

Simple Flow Step 3: Synthesis and Timing Verification

Using synthesis programs . . .
. . . generate *design database*.

Design database has information needed to fabricate the chip . . .
. . . and to perform simulations with accurate timing.

Simulate using design database to verify that timing is acceptable . . .
. . . if timing is not acceptable edit the Verilog structural description and repeat steps above.

Using the Internet, E-mail design database and credit card number to fab.

After a few weeks, get parts back in mail.

Material and Tools in This Course

Topics Covered in This Course

- Coding in SystemVerilog.
- Writing descriptions of common digital devices.
- Using simulation, waveform viewers and similar tools.
- Estimation of cost and delay.
- Writing testbench code.
- Relationship between HDL and synthesized hardware..
- Using synthesis tools.

Tools Used in This Course

Tools and Tool Vendors

EDA (Electronic Design Automation) Tools

Programs to support design automation.

These include SystemVerilog simulators, synthesis programs, design rule checkers, etc., etc., etc.

Major EDA Vendors

- Synopsis
- **Cadence Design Systems**
- Mentor Graphics

Course Will Use Products of Cadence Design Systems

ECE is a member of Cadence's University Software Program.

Software would normally cost well over 100 k$ . . .
. . . which helps explain why you can't have your own copies (unless you're rich).

For Simulation: *Cadence Xcelium Simulator* Package

A collection of programs including:

- *xmvlog* – Verilog simulator.
- *simvision* – Waveform viewer (to view sim results).
- *xrun* – Convenient front end to other programs.

For Synthesis: *Genus Synthesis*

- *genus* – Front end for synthesis tools.

Design Capture

Design Capture:
Entering a design in electronic form.

Start: Idea in engineer's head, scribbles on back of envelope.

Finish: Design in electronic form readable by some EDA tools.

Design Capture Methods

- Schematic Capture

  Enter design using GUI (graphical user interface) schematic editor.

  Fun and easy for beginners but tedious for all but small designs.

- Finite-State Machine Editors

  Programs meant for designing FSM, to be part of larger design.

- Hardware Description Languages

  Like any programming language …
  … design entered using standard or specialized text editor.

Synthesis Design Targets

Design Target:

The type of device to be manufactured or programmed. Synthesis programs generate output for a particular design target.

Design Targets

- *Application-Specific Integrated Circuit (ASIC)*

  A fully custom chip.

  Usually the fastest design target, can have the most components.

- *Field-Programmable Gate Array (FPGA)*

  A chip full of logic whose connection and function can be programmed and later re-programmed.

  Many FPGA vendors provide EDA tools for their products.

- *Programmable Logic Array (PLA)*

  Chip that can be programmed (once) to implement a logic function.

  Usually programmed at the factory.

  PLAs might be used in prototypes or when only a few parts are needed.

- *Gate Array*

  A chip full of gates manufactured in two steps:

  First generic layers containing gates are fabricated . . .
  . . . but gates are not connected to each other.

  Later, metal layers connecting gates are added.

Designer using gate arrays specifies only metal layer.

Since gates fabricated in advance time is saved.

Synthesis

Goal: Convert an HDL description into working hardware.

Start With:

Behavioral or Structural Description

Functionality has been verified by simulation.

Behavioral description (if used) follows synthesizebility rules specified for synthesis program.

Choice of Design Target

Type of target: FPGA, ASIC, etc.

Manufacturer and family.

Major Synthesis Steps (Summary)

Synthesis of technology-independent gate-level description.

Map gates and modules to technology-specific versions.

Place and route.

Major Synthesis Steps (Details)

Synthesis of *technology-independent* gate-level description.

Synthesis program infers registers and minimizes logic.

Registers aren't explicitly declared (even though it will appear otherwise) . . .
. . . so synthesis program must determine (infer) where they are needed.

Because (most) synthesis programs minimize combinational logic . . .
. . . descriptions should be written for clarity.

Output of this step is purely structural code . . .
. . . consisting of gates and standard modules (*e.g.*, for arithmetic), and library modules.

Based on output, designer might tweak design or give hints to synthesis program.

Place and Route

Placement is the determination of the physical location of a part.

Routing is the determination of paths for wires interconnecting parts.

Output of place-and-route step:

Timing information (since technology and wire lengths are known) which may be . . .
. . . *backannotated* (written into) the original behavioral description.

Behavioral descriptions re-simulated to see if they meet timing criteria.

For FPGAs, code to program the devices.

For ASICS and gate arrays, . . .
. . . a design database to *tape-out* and send to a fab.

Fabrication facilities apply additional steps, not covered here.