Paper ID #19049

First Course in VHDL Modeling and FPGA Synthesis of Digital Systems

Prof. Nozar Tabrizi, Kettering University

Dr. Nozar Tabrizi received his BS and MS degrees from the Electrical Engineering Department at Sharif University of Technology, and his PhD degree from The University of Adelaide. He is currently an associate professor of Computer Engineering at Kettering University. His research interests include Computer Microarchitecture, Computer Arithmetic, Parallel Processors and Network on Chip. He is also interested in and actively working on innovative methods of teaching.

©American Society for Engineering Education, 2017

First Course in VHDL Modeling and FPGA Synthesis of Digital Systems

Abstract

Digital Systems is a core course taken by Electrical Engineering, Computer Engineering and Computer Science students worldwide. In this class students learn the building blocks of digital systems and how to put them together to reach larger systems. For implementation purposes, students additionally learn a hardware description language such as VHDL to model their circuits, and then use FPGA chips, cutting-edge technology, to physically build and test their circuits described in VHDL. In this paper we address the challenges faced in teaching VHDL modeling and FPGA synthesis in such an introductory course, and then share our experience in teaching this part of the course. We explain the topics covered in class, we show our lecture slides as well as amount of lecture time to present them to students. Class performance has been encouraging.

Keywords: Design of Digital Systems, FPGA, Implementation of Digital Systems, VHDL

Introduction

Digital Systems is a core course taken by Electrical/Computer Engineering (ECE) as well as Computer Science (CS) students worldwide. This course is a must to understand the basics of hardware architecture of revolutionizing microprocessors that are increasingly and inevitably entering our lives especially in the era of IoT, the internet of things.

Digital Systems has 3 one-hour lecture and one 2-hour lab per week in our ECE department. Our academic terms are 10 weeks long. Number of students in this class varies each term, but 15 to 25 should be a reasonable range to describe our class size in general. In this class, "laboratory work" is an irreplaceable portion, where students learn how to *physically* build circuits. This may be done in different ways: Students place off-the-shelf chips on a breadboard, and wire them up manually. The more sophisticated the circuit is, the more chips, time and space are used. Or, students use cutting edge technology, Field Programmable Gate Arrays (FPGAs), but first they should learn a hardware description language, e.g. VHDL, to write the right code to describe their circuit. Students then use CAD tools to compile and map their code into an FPGA chip, which amazingly takes only a couple seconds! **In a 10-week academic term, our students perform 9 lab assignments out of which 7 assignments are VHDL/FPGA-based.**

Teaching a first course in VHDL to sophomores is a challenge. Unlike software programming languages, such as C, that ECE students learn in the first year of college, VHDL is a so-called *concurrent* language; students should understand what concurrency means in this context.

There is a second reason that makes it a challenge to incorporate VHDL in Digital Systems: VHDL is added on top of a course that used to be taught in one academic term by itself. Therefore, *topic scheduling* becomes more crucial especially if academic terms, such as ours, are only 10 weeks long.

VHDL is a big language; so the third challenge in teaching VHDL is to decide *what to teach*. We have crafted a 9-chapter manuscript for "Digital Systems". There are two parts in each of Chapters 3 through 9, and one part in each of Chapters 1 and 2. Students learn digital systems' theory in Chapters 1 and 2 as well as Parts I of the rest of the chapters. This is basically what we used to teach before we added the VHDL portion to the course. **VHDL modeling and FPGA synthesis of digital systems are covered in Parts II of Chapters 3 through 9**. Our paper focusses on the topics and their specific order to teach this portion. (Please note that VHDL modeling and FPGA synthesis of digital circuits is only one portion of this course. So that students get up to 26% for their lab work, up to 17% for the VHDL portion and up to 57% for the non-VHDL portion.) We teach the nine chapters in the order illustrated in the following table. Note that Chapter 5 is covered last to reach the "sequential logic", and therefore be able to do more advanced assignments as soon as possible.

Subtopics	**Slides No**
Chapter 1	Digital Circuits, Binary Numbers and Truth Tables
Chapter 2	Gates: Basic Building Blocks of Digital Circuits
Chapter 3, Part I	Switching Algebra and Analysis and Design of Digital Circuits
Chapter 3, Part II	Getting Started Computer Aided Design of Digital Circuits VHDL Modeling and FPGA Synthesis of Digital Circuits
Chapter 4, Part I	Logic Minimization Using Karnaugh Maps
Chapter 4, Part II	Hierarchical Designs and Structural Modeling
Chapter 6, Part I	Frequently Used Digital Circuits
Chapter 6, Part II	Behavioral Modeling of Digital Circuits Selected Signal Assignments and Conditional Signal Assignments
Chapter 7, Part I	Memory Cells and Analysis of Sequential Circuits
Chapter 7, Part II	Behavioral Modeling of Digital Circuits Process Constructs
Chapter 8, Part I	Design of Sequential Circuits
Chapter 8, Part II	VHDL Modeling of Finite State Machines
Chapter 9, Part I	Frequently Used Sequential Circuits
Chapter 9, Part II	Register Transfers The Backbone of Digital Systems
Chapter 5, Part I	Binary Number Systems and Binary Arithmetic

As shown in the above table, Part II of each chapter is taught after Part I of that chapter has been covered. Therefore, **the VHDL portion in our class is distributed across the whole academic term.** For this portion we spend almost ***135 minutes*** of our class time. Additionally, students spend some 15 minutes on student-oriented class activity: students are provided with a class-exercise packet; we stop lecturing at some points, and ask students to work on one or more questions pertaining to the current lecture subtopic to develop a better understanding of the lecture material. We specifically encourage them to *either teach each other or learn from each other*. We have seen firsthand how enthusiastically students participate in this teaching/learning activity. In a recent survey, we asked a class of 14 students for their opinions about the following statement:

"Class Exercises" are useful. They are a good learning aid. They also help me evaluate myself.

The survey results are shown in the following table:

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
No of students out of 14	8	4	1	1	0

We also spend some time on the pre-labs.

Our lecture topics are of course found (more or less) in other resources as well [1]-[5]. However, we believe that the sequence of materials, the way that they are presented (especially how they start and how end) and interleaved with the lab assignments, and the amount of time spent on a subtopic can make a difference. And the purpose of this paper is to share our experience with other faculty members who are new to this course, or they feel that their current teaching approach is not efficient enough. Our work is similar to many other works in the literature as pointed out in the next section. One major difference between these works and our work is that we focus more on the teaching details of VHDL rather than explaining the tools that are used in the lab and how they work, or the history of course development, etc.

We have had productive class based on our approach. Students' test results are encouraging. Moreover, our students have done excellent work (in general) on the last and challenging lab assignment to be explained in this paper.

The rest of the paper is organized as follows: Some previous work is reviewed first. We then go over our lecture materials; we also take a quick look at our lab assignments, and then will present some test results. The last section is the conclusion.

Previous work

Pang proposes an integration of online tools for digital circuit design to provide students with an active learning environment [6]. Logicly, Multisim, Modelsim and a FPGA-based design software are considered in this work, where Verilog is used as the hardware description language for FPGA synthesis. However, the topics covered to teach this language are not presented in the paper. In [7] Fida El-Din and Krad use the same CAD tool and development board as we use to add a lab project to a Computer Architecture and Organization course. This project is about modeling, simulation and FPGA synthesis of an 8-bit Arithmetic and Login Unit. However, the paper does not show the lecture materials to teach VHDL. Wang explains his VHDL teaching experience in [8]. The challenge is his work is also to teach a *minimum* subset of VHDL in an introductory course; however, the topics and therefore the order they are taught are not shown in this paper. Additionally and unlike this work, we do not teach variables in our introductory course as we believe that this concept will cause confusion while it is not necessary to know variables to perform the lab assignments of this course. In [9] Wang and Goryll describe their Online Digital Design Course. They use a CAD tool called Logisim [10]. Logisim is an educational simulator for digital circuits. It takes graphical description of hierarchical circuits through a user friendly interface. We have, however, used CAD tools that are widely used in academia. Vera et al explain the challenges they faced to set up a reconfigurable lab that was used to teach students a first course in digital design [11]. They explain the lab work, but unfortunately the lecture topics and how they are presented are not provided.

Lecture materials

In this section, we will present the sequence of topics, the slides under each subtopic and approximate amount of time spent on each subtopic to cover VHDL modeling and FPGA synthesis of digital circuits in this introductory course. Please note that before our students are exposed to VHDL modeling of digital circuits, they learn the concepts and non-VHDL design of the digital circuits. Additionally, they spend two hours (or more) per week in the lab to go over the lab assignments and do the lab assignments in which VHDL and FPGAs are used from week 3 through week 10. We also use part of our lecture time to better prepare for the lab assignments.

The following table shows the subtopics, slide numbers for each subtopic and the approximate duration of lecture for each subtopic. As shown in this table, there are 100 slides with the total lecture time of some two hours to cover VHDL modeling and FPGA synthesis of digital circuits. The 100 slides are shown at the end of this section.

Subtopics	**Slides No**	**Lecture Duration (minutes)**
Getting Started: HDL and FPGAs	1:7	9
Entity and Architecture: Simple Signal Assignments; vector and non-vector signals	8:14	9
Structural Modeling and Hierarchical Designs	15:34	30
Behavioral Modeling Selected and Conditional Signal Assignments	35:56	20
Behavioral Modeling Process Constructs: If Then Else and Case statements	57:70	22
VHDL Modeling of State Machines	71:82	20
Register Transfers: The Backbone of Digital Systems	83:100	25
Total lecture time	100 slides	135
Generate Statements and Generic Constructs (Optional reading and lab assignment)	--	--

The 100 lecture slides are presented in the appendix at the end of the paper. Please note that we have made some minor changes to the lecture PowerPoint slides to get a better fit for this paper. We have also added some text for clarification purposes.

From our experience in teaching this class for many years, we recommend that the following two points should always be taken into consideration in order to avoid common confusions:

1- Students should frequently be reminded that logic gates in FPGAs are realized using Look-Up Tables (LUTs), unlike semicustom or full-custom VLSI. See Slide 4.

2- Students should be encouraged and convinced to look at different pieces of a VHDL code as different pieces of hardware. This will significantly help them better understand the concurrency that naturally exists in this language.

As a final comment before reviewing the slides, we would like to mention that in order to further minimize the material covered in this introductory course, the following two topics may be omitted without significantly affecting students' ability to model complex digital circuits in this introductory course: Selected Signal Assignments; Case Statements.

Lab assignments

Week 1 (basic concepts on discrete and manual logic):
Truth Tables and Voltage Tables
Analysis of Simple Digital Circuits

Week 2 (basic concepts on discrete and manual logic, cont'd):
Gates:
Basic Building Blocks of Digital Circuits

Weeks 3:
Switching Algebra and Analysis and Design of Digital Circuits
Getting Started: Altera Quartus II Software, DE2 Board and
ENTITY, ARCHITECTURE, and Simple Signal Assignments in VHDL

Week 4:
Logic Minimization using Karnaugh Maps
Hierarchical Designs and Structural Modeling
Getting Started with Simulation of Digital Circuits (ModelSim)

In this lab, students structurally model, implement and test a 4-bit hierarchical full comparator.

Week 5:
Behavioral Modeling of Digital Circuits
Selected Signal Assignments and Conditional Signal Assignments

In this lab and after an introductory assignment, students build a min-max circuit. They also build the following circuit:

The circuit takes eight request lines and determines two of them that have the highest priorities among all the asserted inputs.

Fostering an Entrepreneurial Mindset through a Jigsaw-Puzzle Model
In this lab, students are provided with a library of components or *puzzle pieces* as well as the user guide of a *product* and possibly some other reading material. The user guide explains how the product works. The library contains all the necessary *puzzle pieces* to build the product. Students will go over the user guide to understand the underlying product. Then considering what they have available in the library, students will design the product by putting the puzzle pieces together. Once they come up with an initial idea and are done with their first draft of the design, students will collaborate with others who work on the same product to resolve all the possible issues and come up with the functional product. We have recently crafted a paper to report our novel idea of Jigsaw-Puzzle model, and its implementation. The paper is currently under review.

Week 6:
D-latches and D-FFs, and Analysis of Finite-State Machines
Behavioral Modeling of Digital Circuits
Process Statements

Week 7:
VHDL Modeling of Finite State Machines

Students model, implement and test a sequence detector. They also model, implement and test an LED controller that turns an LED on and off through one pushbutton. The system frequency is 50 MHz.

Weeks 8 through 10:
Fostering an Entrepreneurial Mindset through a Producer-Customer Model. In the rest of this section we will briefly go over the idea that we developed and used in this lab assignment. Interested readers may refer to our recent paper that was published based on this work [12].

Students work in groups of 3 to 5. Each team will play the role of a *customer* of a product as well as the *producer* for *another* product. There are two different types of products:

Some of the customers are provided with *defective products* each with one or more undisclosed "Implementation Deviations from the Specification", i.e. a product that does not work as it should. The customer will then critically examine the product to identify the discrepancies between the product's behavior and the product's user guide. The discrepancies will then be discussed with a producer who will understand the voice of the customer and work on the defective product to eventually locate the discrepancies and fix the product to match the user guide. The producer will also resolve the customer's *possible* misunderstandings.

The other customers each will receive either a *performance-improvable* or a *size-improvable* product, i.e., a product that can be improved to get a *faster* product or to get a *smaller* product, respectively. The customer will then critically examine the **"how it works"** of the product to see how it can be improved in the relevant domain: performance or size. The customer will then discuss their findings with a producer who will understand the voice of the customer and work on the improvable product to eventually improve it. The producer will also resolve the customer's *possible* misunderstandings.

Test results

Students are encouraged to prepare and use a double-sided cheat sheet on the tests.

Twenty students took the following test:

Look at the transition table shown below. A is the input, Y is the output and Q1 Q0 are the state variables. **Note:** Binary (not symbolic) states are used in this table.

Q1Q0 \ A	0	1	A: 0	A: 1
00	00	01	0	1
01	11	11	0	0
11	00	10	1	0
10	11	00	1	1
	Q^{n+1}		Y	

The following is an incomplete VHDL code to describe the above table. Read the code carefully and then fill in the blanks to complete the code:

```
ENTITY fsm_test IS
        PORT  (A, Clk          : IN    STD_LOGIC;           -- A is input from outside world
                     Y          : OUT  STD_LOGIC            -- Y is output to outside world
               );
END fsm_test;

ARCHITECTURE Behavior OF fsm_test IS
        -- Note: In this question, we use binary states (not symbolic states):
     SIGNAL  Current_Q, Next_Q    : STD_LOGIC_VECTOR (1 DOWNTO 0);
BEGIN
     -- Output Y is generated here: (More space provided on real test)

     -- Next states are generated here. Note: states are in binary (not symbolic):
      PROCESS (                                   )
        BEGIN
               CASE Current_Q IS    (More space provided on real test)
                      WHEN  "00" =>         -- Use an IF statement here:
                      …
                      WHEN  "01" =>
                      …
                      WHEN  "10" =>         -- You do not have to fill in the following blank

                      WHEN OTHERS =>  -- You do not have to fill in the following blank

               END CASE;
        END PROCESS;

        -- States are updated here: Fill in the blank. (More space provided on real test)
        PROCESS (Clk)
        BEGIN
        ...
               END IF;
        END PROCESS;
END Behavior;
```

The students' test scores are summarized in the following table:

Score	100%	87%	80%	weak
No of students out of 20	13	2	4	1

Fifteen students took the following test:

Question: A function table for a 4-bit counter/shifter is shown below. Write a **neat**, **complete** and **indented** VHDL code to *behaviorally* describe this counter/shifter.

R	**SE**	**CE**	**Next State**	**Mode**
1	X	X	0000	Reset
0	0	X	Shift to right by 1 bit	Shift
0	1	1	Current state + 1	Count
0	1	0	Current state	Hold

Notes:
Call the serial-in input SI (which is used in the shift mode).
Negation sign, ~, has not been appended to active-low inputs, if any.
Do NOT use time consuming names such as LEDR or KEY.
Use the signal names shown in the table.
Use the following line as it is:

IF clk'EVENT AND clk = '1' THEN

This is to help you write a more readable code in a less error-prone format!

The students' test scores are summarized in the following table:

Score	100%	98%	95%	92%	80%	weak
No of students out of 15	2	2	3	3	2	3

Twenty two students took the following test:

Question: Write a complete, legible and indented VHDL code for a counter with the following counting sequence:

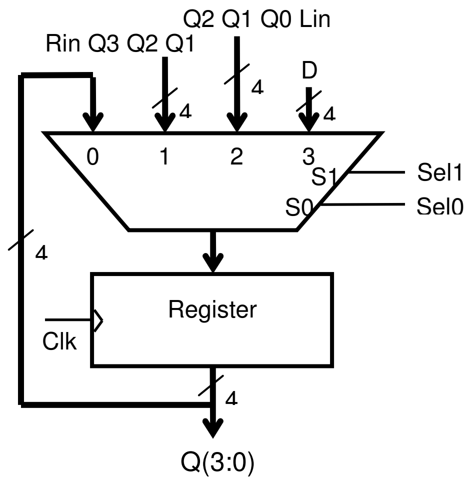
… 000, 001, 010, **011, 110**, 111, 000, 001, 010 …

The students' test scores are summarized in the following table:

Score	100%	96%	90%	80% and below
No of students out of 22	6	5	4	7

Twenty eight students took the following test:

Question: A digital circuit is shown below. Write a **neat**, **complete** and **indented** VHDL code to *behaviorally* describe this circuit.



Note:

- Do NOT use time consuming names such as LEDR or KEY; use the signal names shown in this logic diagram.
- Q, D and Sel are vectors.

ENTITY Test6 IS (*more space on the real test*)

…

END Test6;

ARCHITECTURE Behavior OF Test6 IS

BEGIN

PROCESS ( )

BEGIN

IF clk'EVENT AND clk = '1' THEN -- Leave this line as it is. (*More space on real test*)

…

END IF;

END PROCESS;

END Behavior;

The students' test scores are summarized in the following table:

Score	100%	98%	90%	80% and below
No of students out of 28	16	2	1	9

Eleven students took the following test:

Question: A logic diagram for a 4-bit counter is illustrated here. Write a **neat**, **complete** and **indented** VHDL code to *behaviorally* describe this counter.

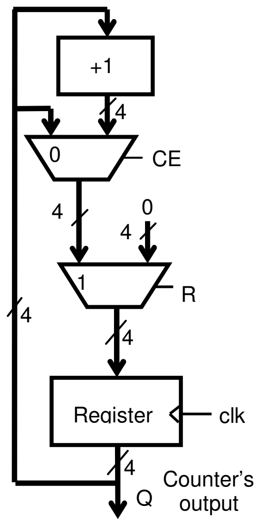
Notes:
Negation sign, ~, has not been appended to active-low inputs, if any.

Do NOT use time consuming names such as LEDR or KEY. Use the I/O names shown in the diagram.

In your VHDL code use the following line as it is:

```
IF clk'EVENT AND clk = '1' THEN
```

Where clk is the clock signal. This is to help you write a more readable code in a less error-prone format!



The students' test scores are summarized in the following table:

Score	100%	98%	weak
No of students out of 11	5	5	1

Conclusion

In this paper we shared our experience in developing a first course in VHDL modeling and FPGA synthesis of digital circuits that result in a so-called 2-dimentional course. The first dimension is the traditional design of digital circuits, in which different components are drawn on paper, and properly interconnected. This can then be transferred onto a breadboard using off-the-shelf chips wired manually. The second dimension of this course is how to use FPGAs, cutting edge technology, instead of discrete components. We addressed the three major challenges that an instructor normally faces in developing such a course. Our achievements in this course design (that can be easily used by our colleagues) are summarized as follows:

- What lecture topics to choose
- What order to choose to cover the lecture topics
- How much time (approximately) to spend on each topic

- What lab topics to choose and in what order to perform them in order to keep them synchronized with the lecture topics.

We finally presented students' test results, which are encouraging. However, we aim for continuous improvement. We see that some students are not very comfortable with preparing good cheat sheets. Some may even do not appreciate how useful a cheat sheet is to answer test questions better and faster, or they may not realize how cheat-sheet preparation by itself provides students with a deeper understanding of the concepts. A couple of students may occasionally forget to prepare and bring one. Writing a good cheat sheet is a skill, and we plan on spending some time to help students improve this skill. We believe that cheat sheet preparation is the counterpart of what we do in professional ASIC design: When we decide to develop a code, we do not normally do it from scratch; we look at the codes that we have already designed and tested, and then choose the closest one to what we need in the current project. We then make the necessary changes to tailor the (old) code to our needs. We believe that this step will greatly improve our students' performance and their test results.

References

[1] S. Brown and Z. Vranesic, Fundamentals of Digital Logic with VHDL Design, 3rd edition, McGraw Hill.

[2] A. B. Marcovitz, Introduction to Logic Design, 3rd edition, McGraw Hill.

[3] R. S. Sandige, M. L. Sandige, Fundamentals of Digital and Computer Design with VHDL, McGraw Hill.

[4] F. Vahid, Digital Design with RTL Design, VHDL, and Verilog, 2nd edition, John Wiley & Sons.

[5] J. F. Wakerly, Digital Design, Principles and Practices, 4th editon, Prentice Hall.

[6] J. Pang 2015. "Active Learning in the Introduction to Digital Logic Design Laboratory Course," *Proceddings of 2015 American Society for Engineering Education*, (Zone III).

https://www.asee.org/documents/zones/zone3/2015/Active-Learning-in-the-Introduction-to-Digital-Logic-Design-Laboratory-Course.pdf

[7] Aws Yousif Fida El-Din and Hasan Krad, "Teaching Computer Architecture and Organization using Simulation and FPGAs," International Journal of Information and Education Technology, Vol. 1, No. 3, August 2011.

[8] Guoping Wang, "Lessons and Experiences of Teaching VHDL," *Proceedings of the 2007 American Society for Engineering Education Annual Conference & Exposition*

[9] Chao Wang and Michael Goryll, "Design and Implementation of an Online Digital Design Course," *ASEE's 123rd Annual Conference & Exposition,* New Orleans, pp. June 26-29, 2016.

[10] Logisim, a graphical tool for designing and simulatgin logic circtuiw. http://www.cburch.com/logisim/

[11] Guillermo A. Vera et al, "Integrating Reconfigurable Logic in the First Digital Logic Course," pp. 10-15.9th *International Conference on Engineering Education*, July 23 – 28, 2006, San Juan, PR

[12] Nozar Tabrizi, "Fostering an Entrepreneurial Mindset in *Digital Systems Class* through a Producer-Customer Model", *2016 IEEE Frontiers in Education Conference (FIE),* Oct 12-15, 2016, Erie, USA

Appendix: Lecture Slides

Getting Started

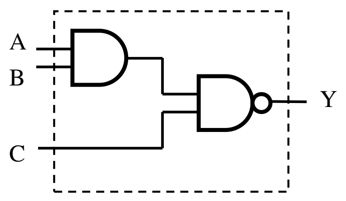
Computer Aided Design of Digital Circuits

VHDL Modeling and FPGA Synthesis of Digital Circuits

1

Manual versus Automated
Discrete versus Integrated

2 TTL chips for this circuit:

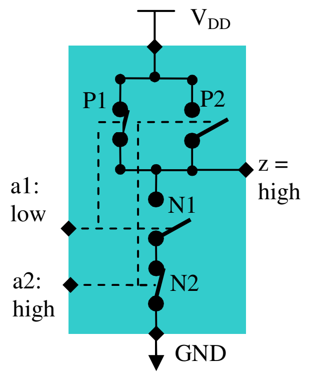


We cannot do this to design chips with 100s of millions of transistors!

2

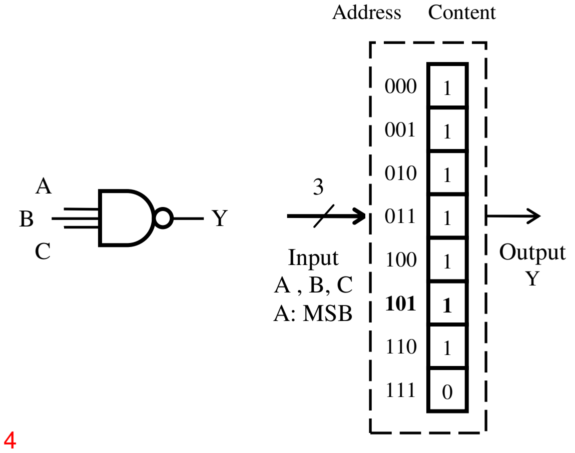
Integrated Circuits

- Full Custom
- Semicustom (ASIC)



3

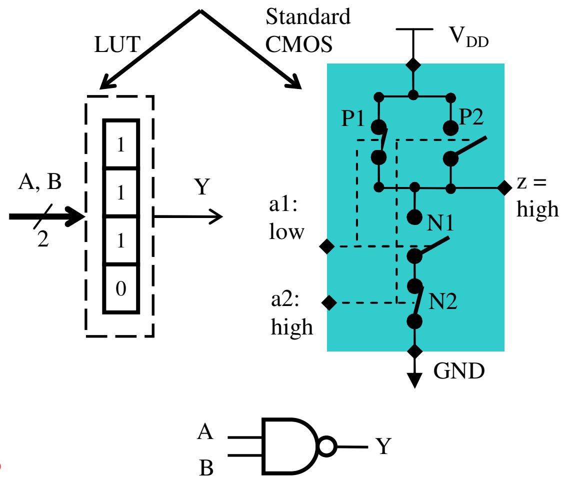
- FPGA
(Made of up LUTs)



Address	Content
000	1
001	1
010	1
011	1
100	1
101	**1**
110	1
111	0

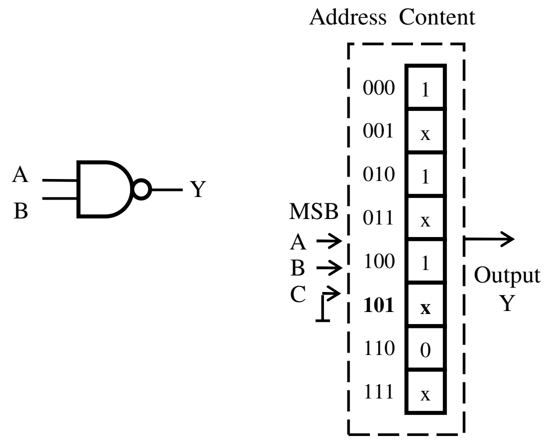
4

Y <= A NAND B



5

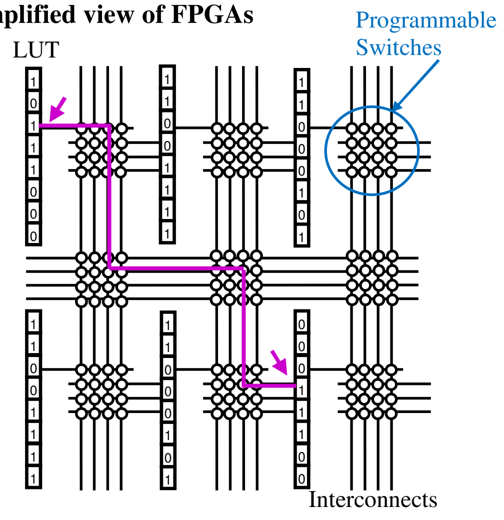
Repentant inputs



Address	Content
000	1
001	x
010	1
011	x
100	1
101	**x**
110	0
111	x

6

Simplified view of FPGAs



Entity and Architecture

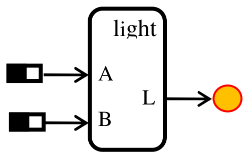
Simple Signal Assignments

Vector and non-vector Signals

Two switches control one light independently.

Row	A	B	L
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

XOR function:
1-bit partial comparator



Logic symbol or graphical entity

L = A' . B + B' . A

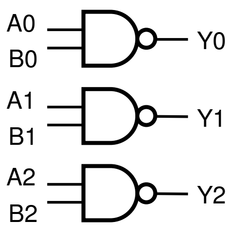
i.e., use a **Simple Signal Assignment:**

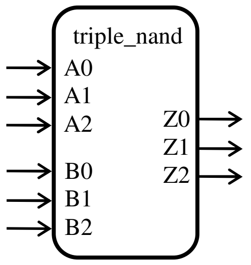
L <= (NOT A AND B) OR (NOT B AND A);

```
ENTITY light IS
PORT (A, B: IN      STD_LOGIC;
            L: OUT  STD_LOGIC
         );
END light;

ARCHITECTURE algebraic OF light IS
BEGIN
 L <= (A AND NOT B) OR (B AND NOT A);
END algebraic;
```

Vectors

Write a VHDL code to describe the following gates:

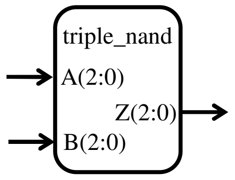




Graphical entity

```
ENTITY triple_nand IS
PORT
  (A2, A1, A0, B2, B1, B0:  IN      STD_LOGIC;
                     Z2, Z1, Z0:  OUT   STD_LOGIC
   );
END triple_nand;

ARCHITECTURE Algebraic OF triple_nand IS
BEGIN
  Z2 <=  A2 NAND B2;
  Z1 <=  A1 NAND B1;
  Z0 <=  A0 NAND B0;
END Algebraic;
```

Easier way; use the following shorthand:



```
ENTITY triple_nand IS
PORT
  (A, B:  IN     STD_LOGIC _VECTOR (2 DOWNTO 0);
      Z :  OUT STD_LOGIC _VECTOR (2 DOWNTO 0)
  );
END triple_nand;

ARCHITECTURE algebraic OF triple_nand IS
BEGIN
  Z <= A NAND B;
END algebraic;
```

13

Note:

Z <= A NAND B;

is equivalent to

```
Z(2) <= A(2) NAND B(2);
Z(1) <= A(1) NAND B(1);
Z(0) <= A(0) NAND B(0);
```

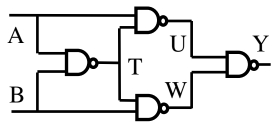
You may generalize it to longer vectors.

14

Structural Modeling: Introduction

-- 1-bit partial comparator

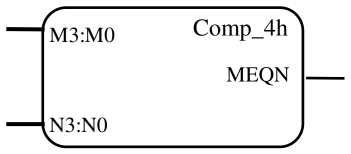
```
ENTITY comp_1 IS
PORT (A, B: IN     STD_LOGIC;
              Y: OUT  STD_LOGIC
         );
END comp_1;

ARCHITECTURE Algebraic OF comp_1 IS
  SIGNAL T, U, W: STD_LOGIC;
BEGIN
  Y <= W NAND U;
  W <= B NAND T;
  U <= A NAND T;
  T <= A NAND B;
END Algebraic;
```



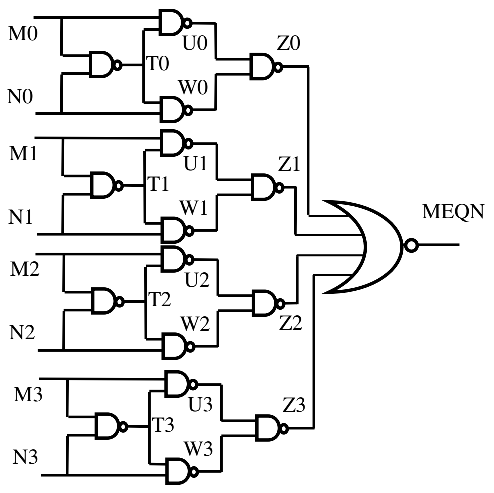
15

4-bit partial comparator

You may repeat 1-bit comparator 4 times ☹
See next slide.
Time consuming and error prone especially for large circuits.



16

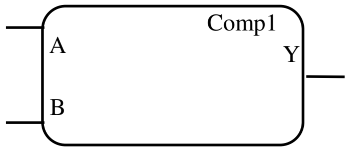


17

```
Y0 <= W0 NAND U0;
W0 <= N0 NAND T0;
U0 <= M0 NAND T0;
T0 <= M0 NAND N0;
Y1 <= W1 NAND U1;
W1 <= N1 NAND T1;
U1 <= M1 NAND T1;
T1 <= M1 NAND N1;
Y2 <= W2 NAND U2;
W2 <= N2 NAND T2;
U2 <= M2 NAND T2;
T2 <= M2 NAND N2;
Y3 <= W3 NAND U3;
W3 <= N3 NAND T3;
U3 <= M3 NAND T3;
T3 <= M3 NAND N3;
MEQN <= NOT(Z3 OR Z2 OR Z1 OR Z0)';
```

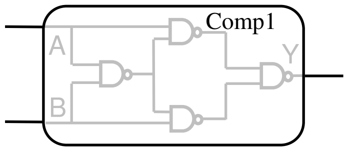
18

Graphical version of entity (logic symbol)



19

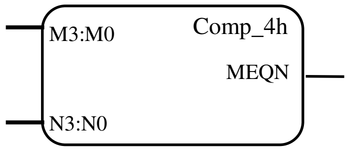
Component:



Formals: Inputs/Outputs of component
Instantiate: copy and paste component into new design
Instance: resulting copy
Actuals: Inputs/Outputs of instance (see next slide)

20

Use the 1-bit partial comparator to design a 4-bit partial comparator (Structural Modeling)



21

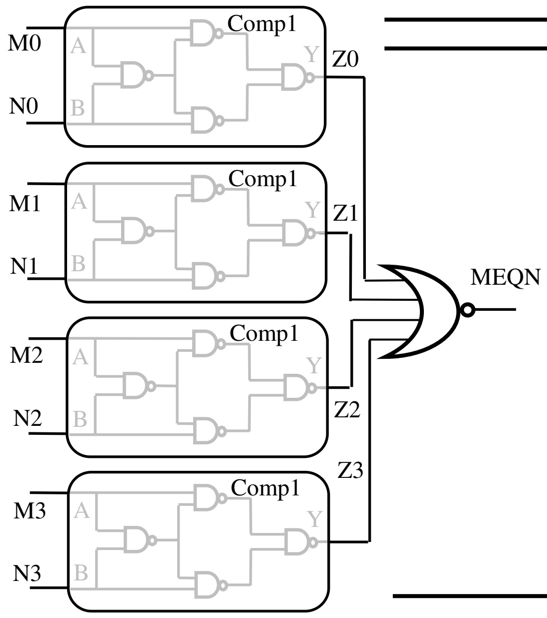
Structural Modeling

Reuse (instantiate) components as many times as needed.
You need to interconnect them properly.
See next slide.

Note:
Z0-Z3
Intermediate signals
Declare them in architecture

M0-M3, N0-N3, MEQN
Formals or Interface signals
Declare them in entity

22



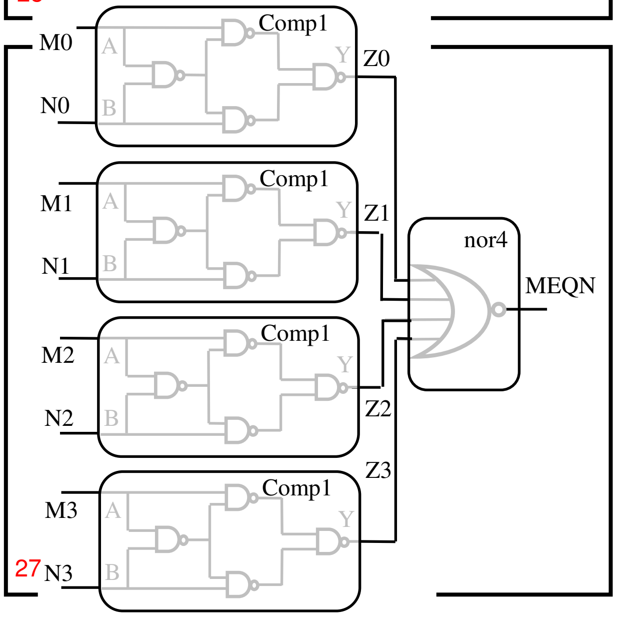
23

```
ENTITY comp_4h IS
PORT
   (M, N   : IN
                  STD_LOGIC_VECTOR (3 DOWNTO 0);
   MEQN : OUT
                  STD_LOGIC
    );
END comp_4h;
```

24

```
ARCHITECTURE Structure OF comp_4h IS
COMPONENT comp_1
  PORT (A,   B:   IN      STD_LOGIC;
               Y:   OUT  STD_LOGIC
          );
END COMPONENT;

SIGNAL Z: STD_LOGIC_VECTOR (3 DOWNTO 0);

BEGIN
  First_Comparator: comp_1
  PORT MAP   (M(0), N(0), Z(0));   -- Actuals
```
25

```
BEGIN
  First_Comparator: comp_1
  PORT MAP        (M(0),  N(0), Z(0));

  Second_Comparator: comp_1
  PORT MAP        (M(1), N(1), Z(1));

  Third_Comparator: comp_1
  PORT MAP        (M(2), N(2), Z(2));

  Fourth_Comparator: comp_1
  PORT MAP        (M(3), N(3), Z(3));

  MEQN <= NOT (Z(3) OR Z(2) OR Z(1) OR Z(0));
END Structure;
```
-- **Note:** The NOR gate is described *algebraically* and not structurally.
-- The NOR gate is described *structurally* in next slide.
26



27

This needs a new component, nor4.
nor4 will be instantiated in architecture.
28

```
ARCHITECTURE Structure OF comp_4h IS
  COMPONENT comp_1
  PORT (A, B:  IN      STD_LOGIC;
             Y:  OUT  STD_LOGIC
          );
END COMPONENT;

COMPONENT nor4
  PORT (A, B, C, D:    IN      STD_LOGIC;
                       Y:    OUT  STD_LOGIC
          );
END COMPONENT;

  SIGNAL Z: STD_LOGIC_VECTOR (3 DOWNTO 0);
```
29

```
BEGIN
  First_Comparator: comp_1
  PORT MAP   (M(0),  N(0), Z(0));

  Second_Comparator: comp_1
  PORT MAP   (M(1), N(1), Z(1));

 Third_Comparator: comp_1
  PORT MAP   (M(2), N(2), Z(2));

  Fourth_Comparator: comp_1
  PORT MAP   (M(3), N(3), Z(3));

  norGate: nor4
  PORT MAP (Z(0), Z(1), Z(2), Z(3), MEQN);
-- Note: Component nor4 is described on next slide.

END Structure;
```
30

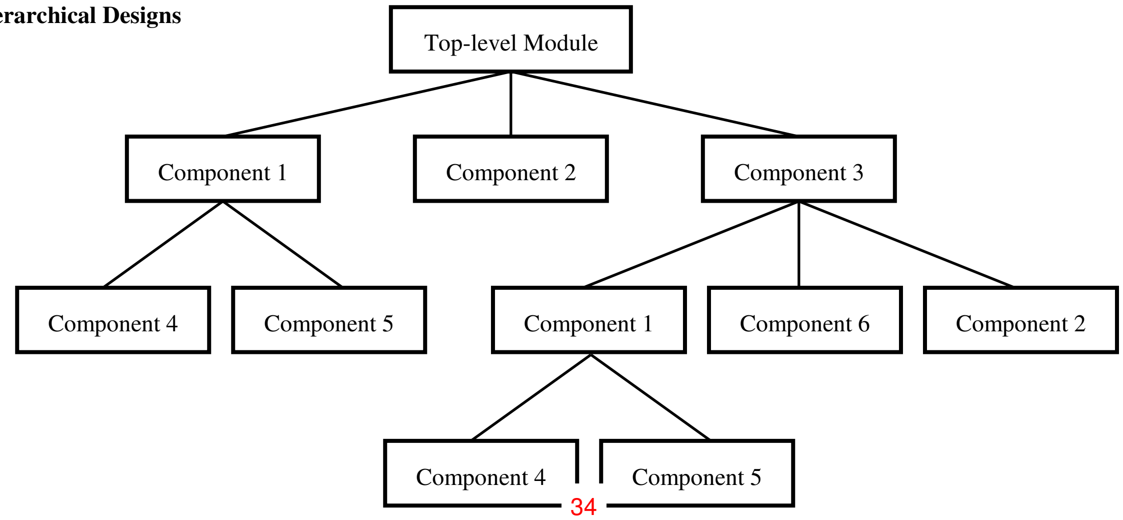
See how interconnections are made between instances, e.g., the output of first comparator is called Z(0), so is the first input of the nor gate.

31

```
--  nor4 is described here:

ENTITY nor4 IS
PORT (A, B, C, D: IN      STD_LOGIC;
                    Y: OUT  STD_LOGIC
          );
END nor4;

ARCHITECTURE algebraic OF nor4 IS
BEGIN
   Y <= NOT(A OR B OR C OR D):
END algebraic;
```

32

Hierarchical Designs



33

34

Conditional Signal Assignments

and

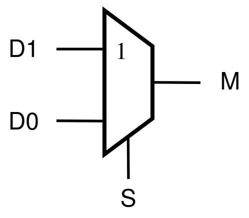
Selected Signal Assignments

35

Simple Signal Assignments work fine for simple circuits. See next slide.

36

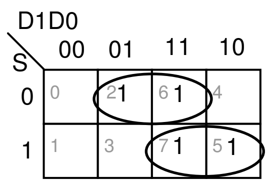
Describe a 2-input single-bit mux:



Truth table:

Row	D1	D0	S	M
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

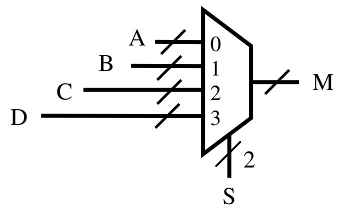
Logic function:



M = S . D1 + S' . D0

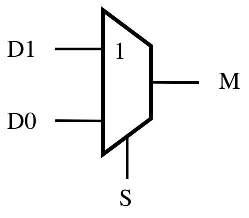
M <= (NOT S AND D0) OR (S AND D1);
Simple Signal Assignment ☺

Can we still use simple signal assignment?



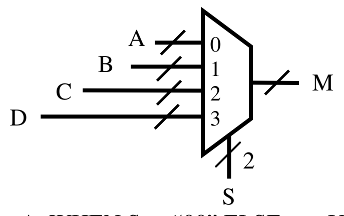
~~Simple Signal Assignment~~ ☹

Conditional Signal Assignment



M <= D1 WHEN S = '1' ELSE D0;

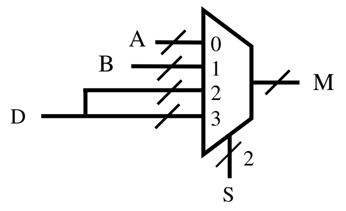
Conditional Signal Assignment



```
M <=  A  WHEN S =  "00" ELSE    -- Use " " for vectors.
      B  WHEN S =  "01" ELSE
      C  WHEN S = "10"  ELSE
      D;
```

-- S is a 2-bit vector comprised of S(1) and S(0).

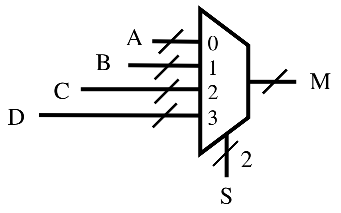
Example:



```
M <=    A WHEN  S = "00"  ELSE
        B WHEN  S =  "01" ELSE
        D;
```

43

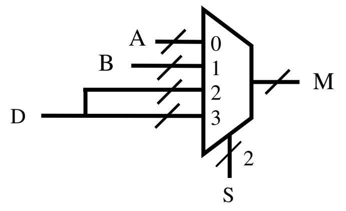
Selected Signal Assignment



```
WITH S SELECT
        M <=    A  WHEN "00",
                B  WHEN  "01",
                C  WHEN "10",
                D  WHEN  OTHERS;
```

44

Example:



```
WITH S SELECT
        M <=    A  WHEN "00",
                B  WHEN  "01",
                D  WHEN  OTHERS;
```

45

How to Describe Truth Tables?

Row	D1	D0	S	mjf
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

46

```
WITH ST SELECT

        MJF <=  '0'     WHEN    "000",
                '0'     WHEN    "001",
                '0'     WHEN    "010",
                '1'     WHEN    "011",
                '0'     WHEN    "100",
                '1'     WHEN    "101",
                '1'     WHEN    "110",
                '1'     WHEN    OTHERS;

-- Note: Order is not important.
```

47

Shorthand

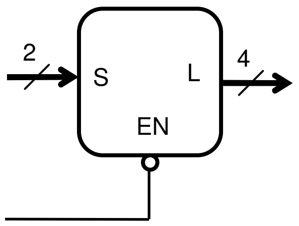
Same truth table:

```
WITH ST SELECT

  MJF <=  '0'     WHEN "000" | "001" | "010" | "100",
          '1'     WHEN OTHERS;
```

48

Binary Decoders

Example: 2 to 4 active-high binary decoder with an active-low enable



EN	S	L
0	00	0001
0	01	0010
0	10	0100
0	11	1000
1	xx	0000

49

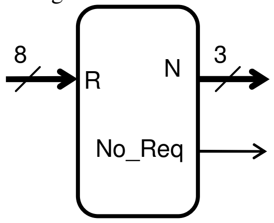
Now
order is important!

```
L <=    "0000"   WHEN  EN_N  = '1' ELSE
        "0001"   WHEN  S = "00"     ELSE
        "0010"   WHEN  S = "01"     ELSE
        "0100"   WHEN  S ="10"      ELSE
        "1000";
```

50

Priority Encoders

Example: 8 to 3 active-high



```
N <=    "111"    WHEN   R(7)  = '1'ELSE
        "110"    WHEN   R(6)  = '1' ELSE
        "101"    WHEN   R(5)  = '1' ELSE
        "100"    WHEN   R(4)  = '1' ELSE
        "011"    WHEN   R(3)  = '1' ELSE
        "010"    WHEN   R(2)  = '1' ELSE
        "001"    WHEN   R(1)  = '1' ELSE
        "000";   --  Order IS Important
```

51

R0	R1	R2	R3	R4	R5	R6	R7	N2	N1	N0	No Req
x	x	x	x	x	x	x	1	1	1	1	0
x	x	x	x	x	x	1	0	1	1	0	0
x	x	x	x	x	1	0	0	1	0	1	0
x	x	x	x	1	0	0	0	1	0	0	0
x	x	x	1	0	0	0	0	0	1	1	0
x	x	1	0	0	0	0	0	0	1	0	0
x	1	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	x	x	x	1

52

Concatenation operator, &

Example
N <= D (0) & D (3 DOWNTO 1);
D = 1010
N = 0101

Example
M <= K (4 DOWNTO 2) & L (3 DOWNTO 1);
K = 100101
L = 001001
M = 001100

53

Shorthand for repeated bits
Example
16-bit data, data_16, is sign extended, and becomes 32-bit data:
data_32 <= (31 DOWNTO 16 => data_16(15)) & data_16;

Use OTHERS to create a vector with repeated bits:
Example:
The following creates a 16-bit vector all zeros:
zero16 <= (OTHERS => '0');

54

Comparator

```
AGTB   <= '1'  WHEN  A > B   ELSE
            '0';
```

Other *relational* operators:

• Equal	=
• Not equal	/=
• Less than	<
• Less than or equal to	<=
• Greater than	>
• Greater than or equal to	>=

55

Logical and Relational operators may be combined:

Examples

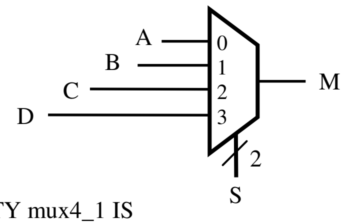
```
GT <= '1'  WHEN  A1 > B1 AND B2 < A2 ELSE  '0';

y  <= '1'  WHEN  (a AND NOT b) = '1'  ELSE '0';

z  <= '1'  WHEN  a = '1' AND b = '0'  ELSE    '0';

y  <=  '1'
        WHEN  ((a AND NOT b) = '1') AND (A1 < B1)

ELSE  '0';
```

56

Process Constructs

IF THEN ELSE
and
CASE Statements

57



```
ENTITY mux4_1 IS
PORT (A:  IN   STD_LOGIC_VECTOR(3 DOWNTO 0);
      Sel:  IN   STD_LOGIC_VECTOR (1 DOWNTO 0);
      M: OUT STD_LOGIC
      );
END mux4_1;
```

58

```
ARCHITECTURE Behavior OF mux4_1 IS
BEGIN
PROCESS (A, Sel)
BEGIN
 IF         Sel = "00" THEN M <= A(0);
   ELSIF Sel = "01" THEN M <= A(1);
   ELSIF Sel = "10" THEN M <= A(2);
   ELSE                          M <= A(3);
 END IF;
END PROCESS;
END Behavior;
```

59

```
ENTITY d_latch IS
PORT ( D, clk: IN            STD_LOGIC;
               Q: BUFFER   STD_LOGIC
        );
END d_latch;
```

60

```
ARCHITECTURE Behavior OF d_latch IS
BEGIN
 PROCESS (D, clk)
 BEGIN
   IF clk = '1' THEN  Q <= D;
   ELSE               Q <= Q;
  END IF;
 END PROCESS;
END Behavior;
```

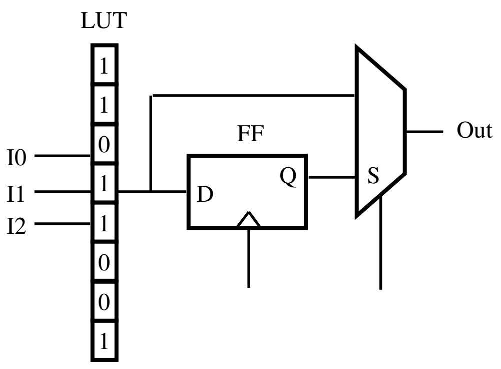
61

Note: You may leave this out:

ELSE   Q <= Q;

62

Logic Element
LUT plus a Flip Flop



63

```
ENTITY dmem IS
PORT ( D, clk     : IN      STD_LOGIC;
        Q         : OUT     STD_LOGIC
      );
END dmem;

ARCHITECTURE Behavior OF dmem IS
BEGIN
 PROCESS (clk)
 BEGIN
   IF clk'event AND clk = '1' THEN  Q <= D;

   ELSE                             Q <= Q;

 END IF;
 END PROCESS;
END Behavior;
```

64

Note: You may leave this out:

ELSE   Q <= Q;

65

```
ARCHITECTURE Behavior OF dmem IS
BEGIN
 PROCESS (clk, R)  -- R is asynchronous

 BEGIN
   IF R = '1' THEN                         Q <= '0';
    ELSIF clk'event AND clk = '1' THEN  Q <= D;
    ELSE                                   Q <= Q;
   END IF;
 END PROCESS;
END Behavior;
```

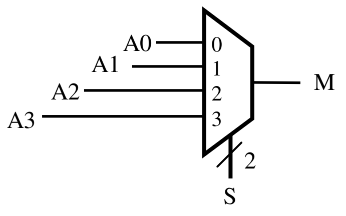
66

R, reset input is asynchronous.
So check it before Clock.
See next slide.

```
ARCHITECTURE Behavior OF dmem IS
BEGIN
 PROCESS (clk, R)  -- R is asynchronous
 BEGIN
  IF R = '1' THEN  Q <= '0';
  ELSIF  clk'EVENT AND clk = '1'  THEN
     IF C_n = '0' THEN          Q <= '0';
     ELSE                       Q <= D;
     END IF;

  ELSE                          Q <= Q;

  END IF;
        END PROCESS;
END Behavior;
```

Case Statement

Example: 4-input one-bit Multiplexer

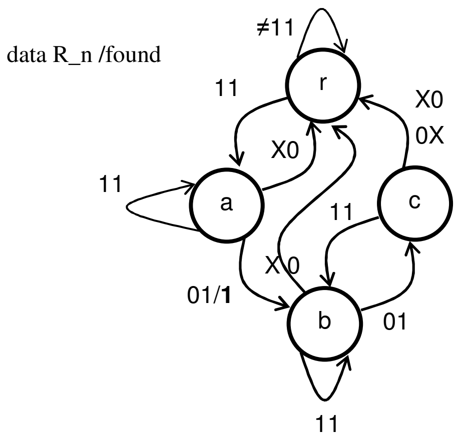


```
ARCHITECTURE Behavior OF mux4_1 IS
BEGIN
 PROCESS (A, Sel)              -- Sensitivity list
 BEGIN
  CASE  Sel IS
     WHEN        "00" => M <= A(0);
     WHEN        "01“ => M <= A(1);
     WHEN        "10" => M <= A(2);
     WHEN OTHERS => M <= A(3);
  END CASE;
 END PROCESS;
END Behavior;
```

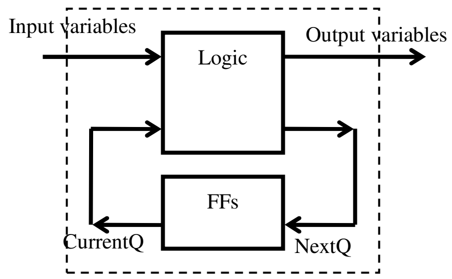
VDHL Description of Finite State Machines

Example: Synchronous bit-serial input stream is received on a single line, data. Develop a state diagram with two inputs, R_n (active-low reset) and data, and one output, found, such that found will be pulled up only with the '0' of any "10" sequence on data line provided that the reset input is deasserted. The state machine will then wait to be reset in order to resume the above cycle. It is reset by two consecutive zeros in the input stream. An asserted R_n will also send the machine to the reset state. R_n may be asserted in any state. R_n has priority over data.
A representative input bit pattern on data and the corresponding output bit pattern on found are shown below:

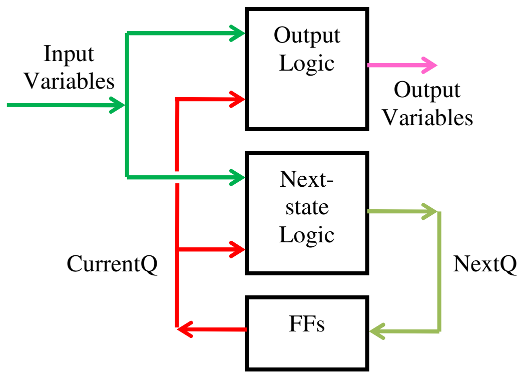
```
Data:    0 0 0 1 1 1 0 1 0 1 1 0 1 1 0 0 1 0 1 1 1 1
Found:   0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
```



Reminder

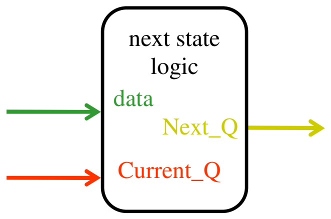


Reminder



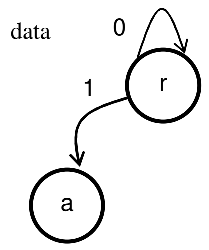
```
ENTITY fsm_design IS
  PORT (   data, R_n, Clk: IN     STD_LOGIC;
                   found: OUT   STD_LOGIC
          );
  END fsm_design;
ARCHITECTURE Behavior OF fsm_design IS
  TYPE    state_type    IS    (r, a, b, c);
  SIGNAL Current_Q, Next_Q: State_type;
BEGIN  -- Continues on next slides …
-- There are 3 parts in the architecture:
-- next state logic
-- output logic
-- memory (flip flops)
```

Next state Logic



77

```
PROCESS (Current_Q, data)
BEGIN
-- This portion is highlighted as an example:
 CASE Current_Q IS
    WHEN r => IF data = '1' THEN  Next_Q <= a;          ELSE
                                       Next_Q <= r; END IF;
```

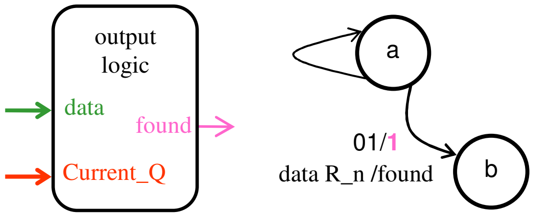


78

```
PROCESS (Current_Q, data)
BEGIN
 CASE Current_Q IS
    WHEN r => IF data = '1' THEN  Next_Q <= a;          ELSE
                                       Next_Q <= r; END IF;
   WHEN a =>  IF data = '1' THEN Next_Q <= a;         ELSE
                                     Next_Q <= b; END IF;
    WHEN b =>  IF data = '1' THEN Next_Q <= b;       ELSE
                                     Next_Q <= c; END IF;
    WHEN OTHERS =>  IF data = '1'          THEN
     Next_Q <= b;      ELSE       Next_Q <= r; END IF;
  END CASE;
 END PROCESS;
```

79

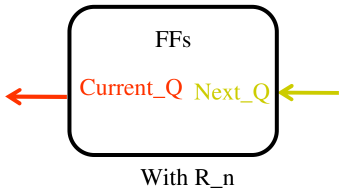
Output Logic



```
found <= '1'    --output is generated here:
WHEN Current_Q = a   AND   data = '0'     AND     R_n  =  '1'
ELSE      '0';
```

80

-- States are updated here

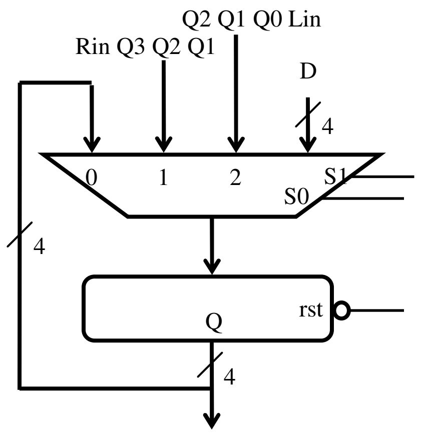


81

```
 PROCESS (Clk)
  BEGIN
    IF Clk'EVENT AND Clk = '1' THEN
       IF R_n = '0'     THEN  Current_Q <= r;  ELSE
                             Current_Q <= Next_Q;
       END IF;
     END IF;
   END PROCESS;
END Behavior;
```

82

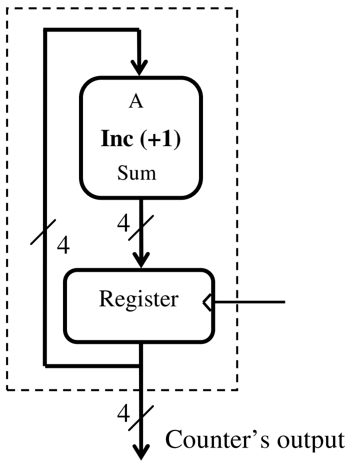
Register Transfers:
The Backbone of Digital Systems



```
ENTITY usr1 IS
PORT ( D: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
        clk, rst, Lin, Rin: IN  STD_LOGIC;
        S: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
        Q: BUFFER
             STD_LOGIC_VECTOR (3 DOWNTO 0)
         );
END usr1;
```

```
ARCHITECTURE Behavior OF usr1 IS
BEGIN
 PROCESS (clk)
 BEGIN
  IF clk'EVENT AND clk = '1' THEN
    IF rst = '0' THEN Q <= "0000"; ELSE
     CASE S IS
        WHEN "11" => Q <= D;  -- Parallel load
        WHEN "01" => Q <= Q(2 DOWNTO 0) & Lin;
        -- Shift left
        WHEN "10" => Q <= Rin & Q(3 DOWNTO 1);
        -- Shift right
```

```
        WHEN OTHERS =>  Q <= Q;
     END CASE;
    END IF;
  END PROCESS;
END Behavior;
```



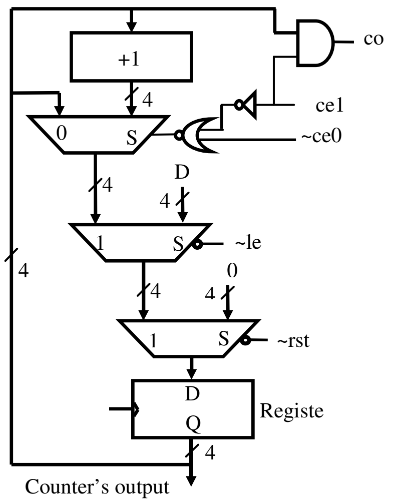
```
ENTITY cntr_4b IS
PORT ( clk: IN  STD_LOGIC;
         cntr: BUFFER
                   STD_LOGIC_VECTOR(3 DOWNTO 0)
         );
END cntr_4b;
```

89

```
ARCHITECTURE Behavior OF cntr_4b IS
BEGIN
 PROCESS (clk)
 BEGIN
  IF clk'EVENT    AND   clk = '1'   THEN
    cntr <= cntr + 1;     -- '+' is the addition operator
  END IF;
 END PROCESS;

END Behavior;
```

90

Example



91

~rst	~le	ce1	~ce0	Next State	Mode
0	X	X	X	0000	Clear
1	0	X	X	D	Load
1	1	0	X	Current State	Hold
1	1	X	1	Current State	Hold
1	1	1	0	Current State + 1	Count

92

```
ENTITY cntr_4b IS
  PORT    ( D: IN
          STD_LOGIC_VECTOR (3 DOWNTO 0);
           clk, ce0, ce1, le, rst: IN  STD_LOGIC;
           co: OUT                    STD_LOGIC;
          Q: BUFFER
           STD_LOGIC_VECTOR (3 DOWNTO 0)
             );
END cntr_4b;
```

93

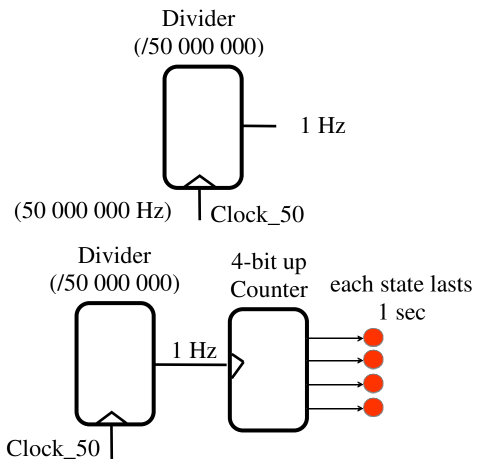
```
ARCHITECTURE Behavior OF cntr_4b IS
BEGIN
 PROCESS (clk)
 BEGIN
  IF clk'EVENT AND clk = '1' THEN
   IF rst = '0'  THEN  Q <= "0000";
   -- Active-low reset has the highest priority
   ELSIF le = '0' THEN  Q <= D;
   -- Active-low load-enable has the second priority
```

94

```
    -- 2 Count Enables: ce1 (active-high) and ce0 (active-
low). Either one can disable the counter.

     ELSIF (ce1 AND NOT ce0) = '1' THEN

       Q <= Q + 1;

      END IF;

   END IF;

 END PROCESS;

 co <= '1' WHEN Q = "1111" AND ce1 = '1' ELSE  '0';
-- Carryout (co) is not affected by ce0.

END Behavior;
```

95

Example: Divide by 50 000 000

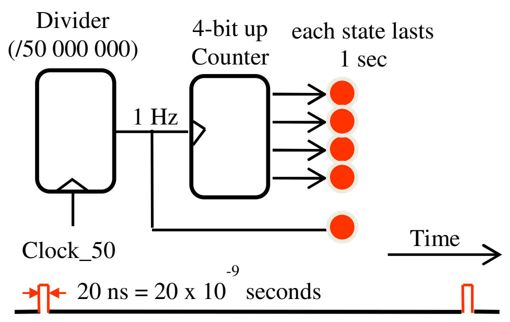


96

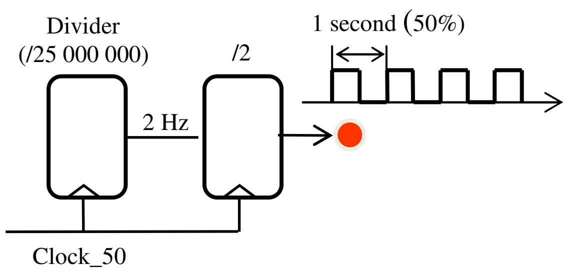
```
ENTITY tb IS
PORT (clock_27: IN                 STD_LOGIC;
             LEDR:
BUFFER   STD_LOGIC_VECTOR(3 DOWNTO 0)
          );
END tb;

ARCHITECTURE Behavior OF tb IS
SIGNAL EN   :  STD_LOGIC;

SIGNAL Q: INTEGER RANGE 0 TO 50000000;
             -- NEW!
```

97

```
BEGIN
PROCESS (clock_50) -- Divider
 BEGIN
   IF clock_50'EVENT AND clock_50 = '1' THEN
     IF EN = '1' THEN Q <= 1; ELSE   Q <= Q + 1;
     END IF;
   END IF;
 END PROCESS;

 EN  <=  '1'   WHEN   Q = 50000000   ELSE   '0';

 PROCESS (clock_50)  -- Runs @ 1 Hz
 BEGIN
   IF clock_50'EVENT AND clock_50 = '1' THEN
     IF      EN = '1'      THEN     LEDR <= LEDR + 1;
     END IF;
   END IF;
 END PROCESS;
END Behavior;
```



99



100