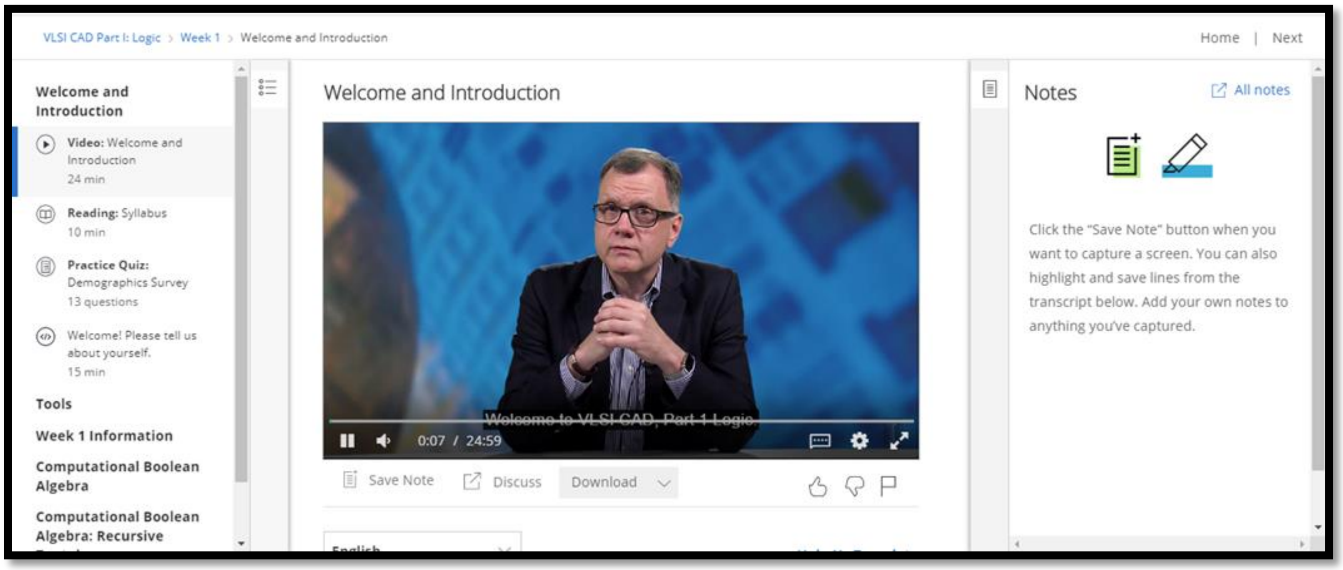
# DAILY ASSESSMENT REPORT

| | | | |
|---|---|---|---|
| **Date:** | **03-August-2020** | **Name:** | **Swastik R Gowda** |
| **Course:** | **VLSI CAD Part-1** | **USN:** | **4AL17EC091** |
| **Topic:** | **Week 1** | **Semester & Section:** | **6th Sem 'B' Sec** |
| **Github Repository:** | **swastik-gowda** | | |

## FORENOON SESSION DETAILS

**Image of session**


VLSI CAD Part I: Logic > Week 1 > Welcome and Introduction

Home | Next

Welcome and Introduction

Video: Welcome and Introduction 24 min

Reading: Syllabus 10 min

Practice Quiz: Demographics Survey 13 questions

Welcome! Please tell us about yourself. 15 min

Tools

Week 1 Information

Computational Boolean Algebra

Computational Boolean Algebra: Recursive

Welcome and Introduction

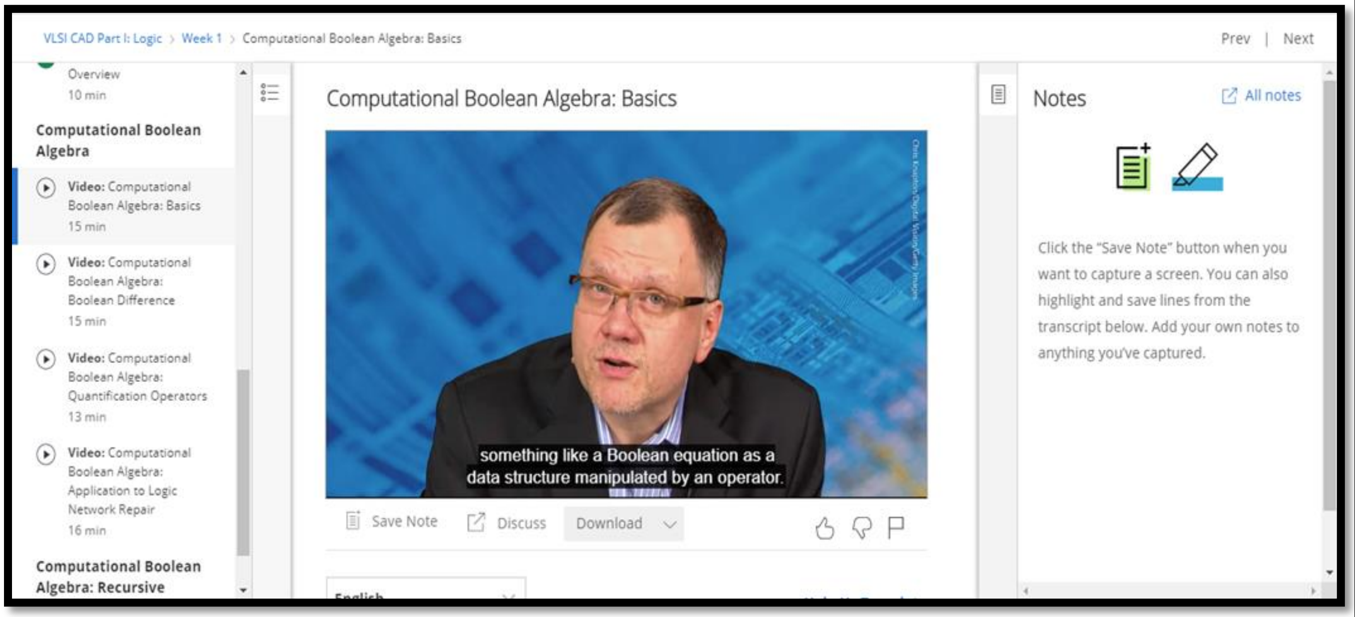
Welcome to VLSI CAD, Part 1 Logic.

0:07 / 24:59

Save Note Discuss Download

Notes All notes

Click the "Save Note" button when you want to capture a screen. You can also highlight and save lines from the transcript below. Add your own notes to anything you've captured.


VLSI CAD Part I: Logic > Week 1 > Computational Boolean Algebra: Basics

Prev | Next

Overview 10 min

Computational Boolean Algebra

Video: Computational Boolean Algebra: Basics 15 min

Video: Computational Boolean Algebra: Boolean Difference 15 min

Video: Computational Boolean Algebra: Quantification Operators 13 min

Video: Computational Boolean Algebra: Application to Logic Network Repair 16 min

Computational Boolean Algebra: Recursive

Computational Boolean Algebra: Basics

something like a Boolean equation as a data structure manipulated by an operator.

Save Note Discuss Download

Notes All notes

Click the "Save Note" button when you want to capture a screen. You can also highlight and save lines from the transcript below. Add your own notes to anything you've captured.

**Report – Report can be typed or hand written for up to two pages.**

## Computational Boolean Algebra: Basics

- This is where we're going to start talking about Boolean algebra from a computational angle. Computational Boolean algebra basics.
- So what do I mean by computational Boolean algebra? Look, by way of background, I'm assuming that you've done some Boolean algebra. You've done hand manipulations of these algebraic equations.
- You've done Karnaugh maps to simplify little Boolean equations. I've got 1 showing on the right. But, look, this is just not sufficient for real designs and is a, is a concrete way of just, you know, illustrating that.
- Let's suppose that I actually told you that I have 40 variables, and I'd like you to optimize that with a Karnaugh map.
- Do the math; it's got about a trillion squares. You could in fact fit that on an American style football field, which is 100 by 50 yards by the way.
- But every map in the Karnaugh map would be about 60 by 60 microns, so that's 60 millionths of a meter. That is just really not happening, here has just got to be a better way.
- So what we need is, we need a computational approach, we need an algorithmic, computational set of strategies for Boolean stuff.
- We've got to be able to think of Boolean objects as data structures and operators and that's just a really new thing. So what are we going to study? We're going to study three things; we're going to study decomposition strategies, because the way you do something that's really complicated is by taking it apart into simpler pieces.
- Solving what you can on the simple pieces and then putting them back together. So there's a set of advanced concepts and terms you need to be able to do this.
- We're also just going to look at some computational strategies, ways to think about Boolean functions, that let them be, manipulated by programs, that's a big thing.
- And then, we're going to look at a couple of small, but really interesting applications, because when you have new tools, there are some new and interesting things to do.
- And I find that it's actually, and this may be a little bit surprising to you, it's interesting to go back to calculus as an analogy here.
- One of the things you probably learned somewhere in high school is that you can take complex functions, for example, something like e to the x.
- Alright, so there's e to the x, and you could actually express e to the x as simpler pieces. Alright, so suppose that you have you know, powers of x.
- You know, you have 1, x, x squared; x cubed and stuff like that. You can glue those things together and you discover that e to the x is 1 plus x plus x squared over 2 factorial plus x cubed over 3 factorial and on you go.

## Step-by-step plan and confidence interval:

- A recursion, where, if we cannot answer the question as to whether the function is a tautology, we can pick a variable, use Shannon factoring, split things, turn them into simpler pieces and build the outlines of a recursive implementation.
- However, there's a whole bunch of more interesting properties of Boolean algebra and Boolean equation that we can use to make this efficient from an Engineering point of view.
- So, in this lecture, we're going to show what those interesting tricks are. And we're going to put it all together in something called URP, the Unite Recursive Paradigm, which is an incredible, famous, powerful set of methods for doing Interesting things on complicated Boolean objects.
- So, we think we have a recursive strategy that will do it. And if we can't calculate whether a function is a tautology just by looking at its positional cube notation list, we shall select a variable, split it into positive and negative cofactors.
- These are simpler objects. They have smaller data structures, we hope, and we should answer the question recursively only if they are both one tautology, can their original function be one.
- And so, this brings up a bunch of mechanical algorithm questions like what are the selection rules, like how should I pick the x, what are the termination rules, how do I know when I can actually stop and not recurse and just, you know, mechanically, how do I do all this stuff.
- Let's start with the mechanics because they're actually pretty easy. Now, I'll admit that this is a bit of a complex looking slide but what I will just say is that this is just a recipe, this is just all the mechanics written down on one slide.
- How do you do the cofactors? How do you actually take the positional cube notation list and take a variable and either cofactor it by setting the variable equal to 1 or setting the variable equal to 0? This is what you do, all you do is you look at every cube in the list, if you're attempting to set the x to be a 1 to compute the positive cofactor.
- You look, if the slot for x has a 1 0, well that's an x bar, you make the cube go away, you remove it. If the slot has a 0 1, well, that's actually got an x.
- When you set x to 1, you set it to a don't care, because that, that variable also went away. And if the cubed didn't have an x in it in the first place, well, you just leave it alone.
- You don't do anything to it. And the rules sort of flip if you're attempting to compute the x equals 0 cofactor.
- If there's a 0 1 in that slot, well, you should just remove the cube, because there's an x there.
- If there's a 1 0 there, well, you should just put a don't care there. And if there's a 1, 1 there, well there's no x there in the first place, you don't have to do anything. It makes a lot more sense to just go look at an example.
- So here's a cube abd and here's a cube BC bar. What if you want to do f of a? And so, we're going to, we're going to set, set the value of a to be a 1, well, in the, in this particular case, if you set the value of a, right, to be a 1, well, in this cube, that becomes a don't care because the, the a goes away, and everything else in the cube stays the same, right? So, this is just a BC cube now.