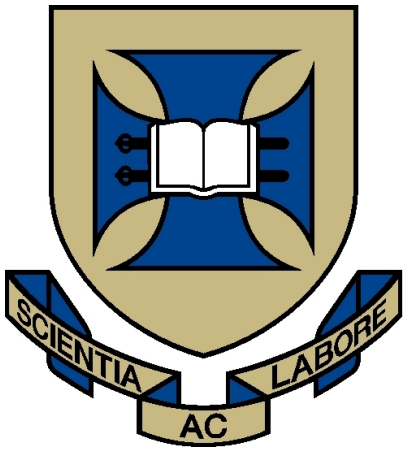
Daniel **Fitz**
(43961229)



University Of Queensland

CSSE4010 – Digital System Design

ENGG2800 Lecture Notes

Table of Contents

List of Tables

List of Figures

What this subject covers

- Combinational Circuit Design
  - Minimisation by manipulation of Boolean expressions
  - Two level and multilevel implementation of logic
  - MSI building blocks: multiplexers, decoders, ROM's and PLA's, parity blocks, etc
  - Arithmetic circuits
  - Timing problems in combinational circuits, hazards
- Sequential Circuit Design
  - Revised basic latches and flip-flops (FF), latch / FF timing and triggering
  - Sequential (Finite State Machine) advanced design methods
  - MSI sequential parts: registers, shift registers, counters, memories, special functions
  - Principles of bus design
  - Controller-data path design methodology
  - Design for testability
  - Computer synthesis and optimization tools
- VHDL – VHSIC Hardware Description Language
  - A Language for describing digital systems
  - Concepts of digital circuit simulation and synthesis
  - Skills with VHDL in simulation, synthesis and test → (Register Transfer Level) RTL design in VHDL
- Technology
  - Field Programmable Gate Arrays

Introduction to moden design methodology

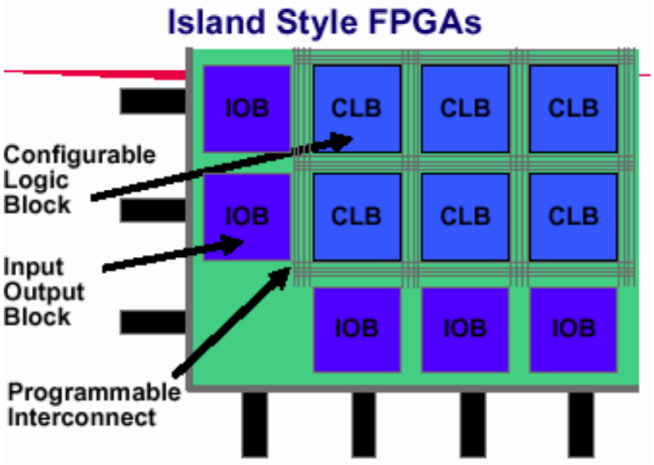


Figure 1: FPGA

Modern FPGAs

- Millions of usable gates
  - Memory blocks on chip
  - DSP blocks
  - Processors (hard and soft cores)
- Integrated development
  - From 'C' or VHDL
  - From Matlab and Simulink
- Ready to compete with ASICs (Application Specific Integreated Circuits)

How to succeed?

- Structured design methodology

- Start at high level of abstraction
- Start simulation as early as possible
- Make prototypes on FPGAs
- Integrated CAD tools
  - Simulation, synthesis, test, emulation, documentation
- Knowledgeable designers

Logic Minimization with Karnaugh Maps

Circuit Minimization

- Algebraic manipulations can be used to simplify Boolean expressions
  - As we've seen, this process is not always easy
- Karnaugh maps (K-maps) provide an easy and visual technique for finding the minimum cost SOP (or POS) form for a Boolean expression
  - This technique has limitations, i.e. works for number inputs less than 7
  - Not good for CAD tools, but good for teaching the idea of simplification

Truth Tables, Minterms

Consider majority function - output D=1, if at least 2 inputs are 1

A	B	C	D	Minterm	Minterm number
0	0	0	0	A'B'C'	m_0
0	0	1	0	A'B'C	m_1
0	1	0	0	A'BC'	m_2
0	1	1	1	A'BC	m_3
1	0	0	0	AB'C'	m_4
1	0	1	1	AB'C	m_5
1	1	0	1	ABC'	m_6
1	1	1	1	ABC	m_7

$$D = \Sigma m(3,5,6,7) = A'BCAB'C + ABC' + ABC$$

The Combining Property

- Recall the combining property

$$xyxy' = x(y + y') = x$$

- Example:

$$f = x1'x2'x3'x1'x2x3' + x1x2'x3' + x1x2'x3$$
$$= m0 + m2 + m4 + m5$$

  - Minterms m0 and m2 differ in only one variable (x2)
    - m0 and m2 can be combined to get x1'x3'
    - Reduced fan in and reduced number of gates
  - Hence, f = x1'x3' + x1x2' (still SOP but not canonical)

Visualizing the Combining Property

$$f(x1, x2) = \sigma m(0,1,3)$$

x1\x2	0	1
0	1	1
1	0	1

Minimum form: f=x1'+x2

3 Variable Map

$$f(A,B,C) = \Sigma m(1,2,6,7)$$

A\BC	00	01	11	10
0	0	1	0	(lightblue) 1
1	0	0	1	(lightblue) 1

Minimum SOP is:

$$f = ABBC' + A'B'C$$

> Gray Code: any two consecutive numbers differ in only a single bit

A\BC	00	01	11	10
0	0	0	1	1
1	1	1	0	0

$$f = A'B + AB'(A \oplus B)$$

A\BC	00	01	11	10
0	0	1	1	0
1	1	0	0	1

The formal Karnaugh Map Method

1) Choose a 1 element
2) Find all maximal groups of 1's adjacent to that element
   - Note: "box" must be a power of 2 in size

1) Repeat steps 1-2 for all 1 elements
2) Select all boxes for which a 1 is "covered" by only that box
   - These boxes are essential!

1) For all 1's not covered by the essential boxes, select the smallest number of other boxes that cover them
   - In case of a choice, select the largest box!

4 Variable Maps

$$f(A,B,C,D) = \Sigma m(0,1,2,3,6,8,9,11,13,14)$$

AB\CD	00	01	11	10
00	1	1	1	1
01	0	0	0	1
11	0	1	0	1
10	1	1	1	0

$$f = AC'D + BCD' + B'C' + B'D + A'B'$$

Terminology

- An **implicant** is a product term in an SOP expression (or a sum term in POS expression)
  - Implicants are always rectangular in shape and the number of 1's covered is a power of 2
- A **prime implicant** is an implicant that is not fully contained in some other larger implicant

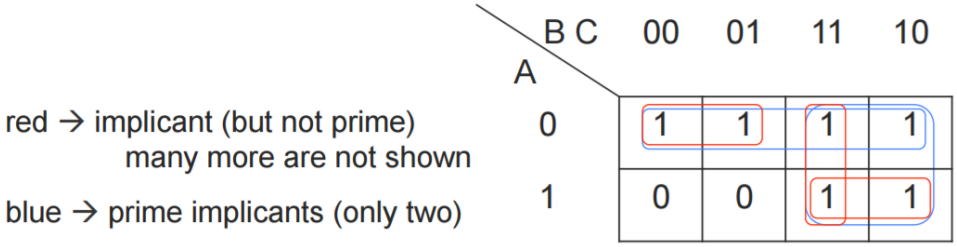


Figure 1: Prime Implicant

Essential Prime Implicants

- An **essential prime implicant** is a prime implicant that contains a 1 not included in any other prime implicant
  - The minimum Boolean expression *must* use this term
- A **cover** is a collection of implicants that accounts for all valuations in which the function is "on" (e.g. 1)

Dont Care Conditions

- Many times there are incompletely specified conditions
  - Valuations that can never occur, or for which we "don't care what the device does"
- Modeling such a device requires us to specify don't care conditions in those instances
  - Use X as a value to indicate we don't care what happens
- Don't care situations are often called **incompletely specified functions**

$$f(A,B,C,D) = \Sigma m(1,5,8,9,10)d(3,7,11,15)$$

AB\CD	00	01	11	10
00	0	1	X	0
01	0	1	X	0
11	0	0	X	0
10	1	1	X	1

$$f = AB' + A'D$$

Karnaugh Map Method Restated

1) Choose an element from the "on" set

2) Find all maximal groups (prime implicants) of "on" elements and X elements adjacent to that element

> Note 1: prime implicants are always a power of 2 in size
> Note 2: do not feel compelled to include X's – use them only when they provide a larger implicant

1) Repeat steps 1-2 for all elements in the "on" set

2) Select all *essential* prime implicants

3) For all elements of the "on" set not covered by the essential prime implicants, select the smallest number of prime implicants that cover them

Combinational Building Blocks

- Set of basic combinational building blocks:
  - Multiplexers (MUXs) and demultiplexers (DMUXs)
    - As a result, define ACTIVE HIGH and ACTIVE LOW terminology
  - Encoders
  - Decoders
  - ROMs

MUXs and DMUXs

- A multiplexer (MUX) or data selector is a combinational digital device that performs the function of selecting one of many data input lines for transmission to some destination
- It uses *n data select* lines to control 2^n *data input* lines

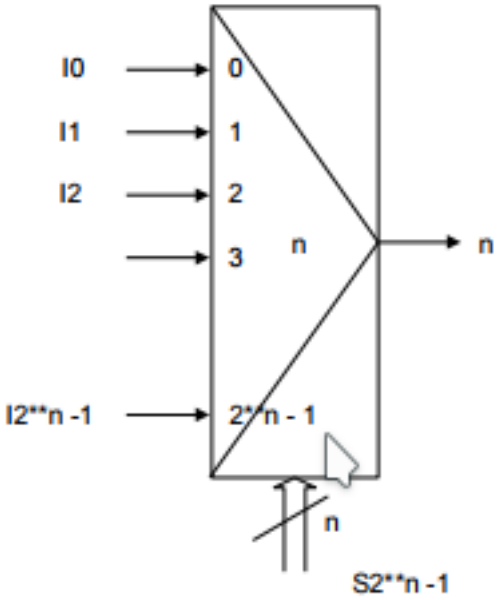


Figure 1: Mux

- The enable pin is active low, with the intention of activiting / deactivating the output
- We need to define this terminology, and to define a set of standards we will follow when designing ciruits – IEEE standards

Documentation Standards

Signal Names

- Each input and output in a logic circuit should have a distinctive alphanumeric label, the signal's name
- We have used single character signal names for many circuits in the past because the circuits didn't do much
- In a real system, well-chosen signal names convey information to someone reading the logic diagram the same way that variable names in software program do

Active Levels for Pins

- Active levels may be associated with input and output pins of gates and larger-scale logic elements
- We use an inversion bubble to indicate an active-low pin and the absence of a bubble to indicate an active-high pin

Why Active Low?

- When Vin turns the transitor ON, the collector voltage drops to ~0.7V, and current flows from Vcc into the ground, dissipating head in R_L
- The power dissipated is approximately Vcc^2/R_L
- When Vin turns the transistor OFF, Vout rises to Vcc, the current through R_L is negligible, and no power is dissipated in R_L

- When, a circuit is not to be used for long periods of time, leave it in the Vout high (logic 1) state, and assert the logic 0 state for the short time needed to activate it

Decoder

- A decoder is an n-input / 2^n-output combinational logic device which has the function of activating one of its 2^n outputs for every unique input pattern (or WORD) of n bits
- Each output is identified by the MINTERM CODE, m_i, of the input WORD pattern it represents

Designing with Decoders

- Since each output is identified by the MINTERM CODE, combinational circuits can be implemented by ORing the appropriate outputs together

$$F = \Sigma m(0, 2, 5, 7)$$

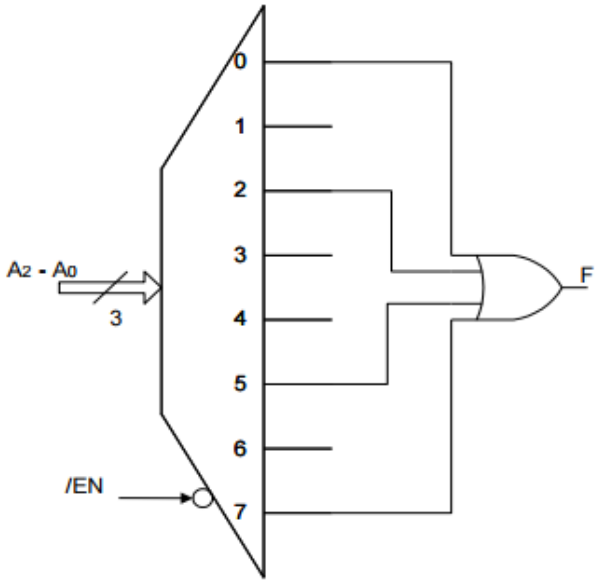


Figure 1: Decoder Example

Demultiplexers (DMUXs)

- A demultiplexer is a device with
  - 1 data input
  - *n* data select lines
  - 2{{*n*}} output lines
  - which can route data from the single input source to one of several destinations

Read-Only Memory (ROM)

- ROMs belong to an important class of general-purpose IC logic array devices which find extensive use in combinational logic design

ROM Logic Structure

- A ROM is in *n*-input / *m*-output device, composed of a decoder stage and a "memory" array
- Bit combinations of *n* input variables are called *addresses*
- There are 2{{*n*}} possible addresses each representing a coded MINTERM on the output side of the decoder stage
- To program, the fusible links are *blown* to remove the connection
- In mask programmable ROMs the fusible links are manufactured in the silicon process, and cannot be field programmable
- In Erasable Programmable ROMs (EPROMs) the fusible links are replaced by special nonvolatile charge-storage mechanisms
- Fusible link based ROMs are called Programmable ROMs (PROMs)

Structural Diagram of a ROM

- Use this symbol whenever you need to draw a ROM in your logic diagram. It is far more expressive than a rectangular box with the name ROM (or DECODER or MUX) plastered somewhere near the box

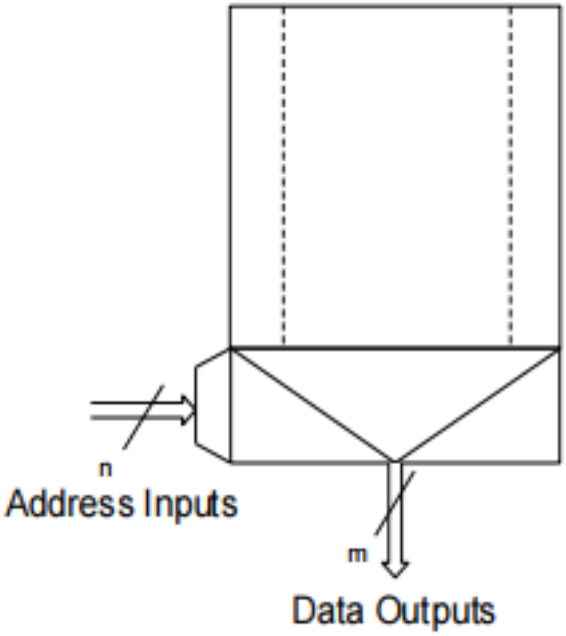


Figure 1: ROM Diagram

VHDL - Intro

- HDL based design technology
- Basic concepts in HDL simulation and synthesis
- VHDL basics
  - Language strategy
  - Language structure
  - Basic contracts and their use

HDL-based design flow

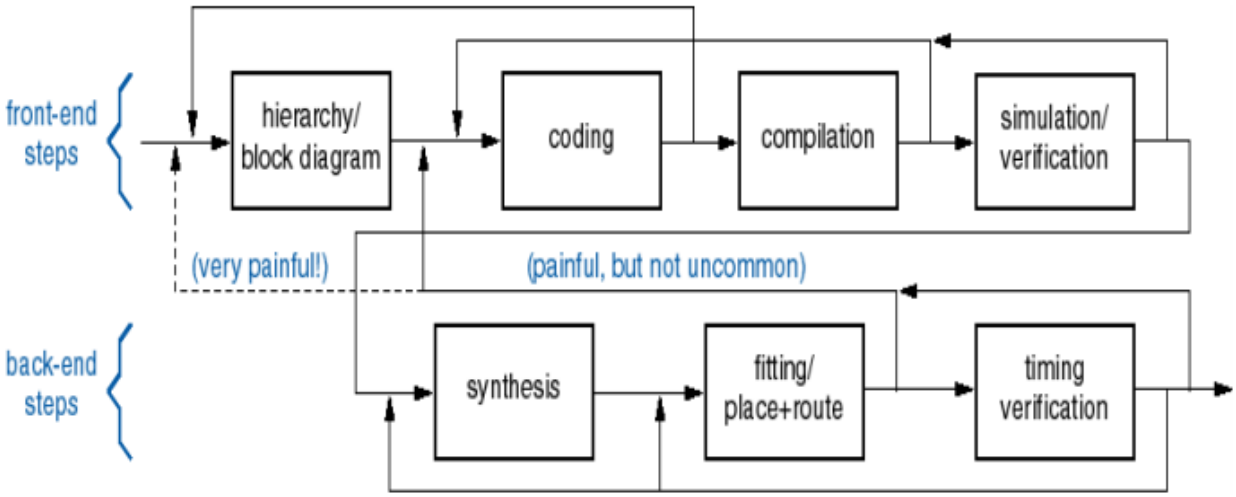


Figure 1: HDL design flow

Applicable to FPGA, ASIC and VLSI

VHDL origins

- Developed in the mid-1980s under DoD sponsorship
- Used for design description, simulation, and synthesis

- Only a subset of the language can be synthesized

VHDL hierarchy

- System is a collection of modules
- **Architecture**: detailed description of the internal structure or behaviour of a module
- **Entity**: a "wrapper" for the architecture that exposes only its external interfaces, hiding the internal details

VHDL coding styles

- **Behavioral** – within the process construct
  - Write a sequential algorithm that describes the circuit's behaviour
  - Flexible like programming but may not be synthesizable or may lead to a very large circuit
  - Very useful for simulation but also synthesis with modern behavioural synthesis tools
- **Dataflow** – in the architecture construct
  - Assign concurrent expressions to signals
  - Includes "when" and "select" (case) conditional statements
- **Structural** – in the architecture construct
  - Define explicit components and the connections between them (netlist).concurrent
  - Textual equivalent of drawing a schematic

Entity and architecture definitions for different modules can be in different files (but entity analyzed first). Compiler maintains "work" library and keeps track of definitions using entity and architecture names.

Testbench in VHDL

- Entity
  - No inputs or outputs
- Architecture
  - Declares component to test, declares signals
  - Instantiates component, connects to signals
  - Process writes input signals, checks output signal
    - Waits a small amount of time after writing input signals
    - Checks for correct output value using "assert" statement

Designed by committee

- Tries to be all things to all people
  - Result – very general, but also very complex
- Standard logic values and elements are not built-in
- Standard logic defined by a "package", `IEEE 1164 STD_LOGIC`
  - Must be explicitly "used" by program

Standard logic values

- Need additional values for simulation three-state logic, pull-ups, etc
- "U" – Uninitialized
- "X" – Forcing Unknown
- "0" – Forcing 0
- "1" – Forcing 1
- "Z" – High Impedance
- "W" – Weak Unknown
- "L" – Weak 0
- "H" – Weak 1
- "-" – Don't care

Predefined VHDL Operators

Grouped into classed:

- Binary logical operators: **and or nand nor xor xnor**
- Relational operators: **= /= < <= > >=**
- Shift operators: **ssl srl sla sra rol ror**
- Adding operators: **+ - &" (concatenation)
- Unary sign operators: **+ -**
- Multiplying operators: **/ mod rem**

VHDL strong typing

- Every signal, variable, function parameter, and function result has a "type"
  - A few built-in types, plus user defined types
- In assignment statements, comparisons, and function calls, types must match
- Commonly used IEEE-1164 types:
  - STD_LOGIC (one bit)
  - STDLOGICVECTOR(range)(multibit vector)
  - INTEGER (built-in integer type)
- Any element used must be declared first

VHDL predefined data types

bit	'0' or '1'
boolean	FALSE or TRUE
integer	an integer in the range -(2^{31} - 1) to (2^{31} - 1) (some implementations support a wider range)
real	floating-point number in the range -1.0E38 to +1.0E38
character	any legal VHDL character including upper and lowercase letters, digits, and special characters (each printable character must be enclosed in single quotes; e.g. 'd', '7', '+')
time	an integer with units fs, ps, ns, us, ms, sec, min, or hr

User-defined Data Types

- Enumeration type
  - **type** state_type **is** (S0, S1, S2, S3, S4, S5);
  - **signal** state : state_type := S1;
    - define a signal called state that can have any one of the values S0, S1, S2, S3, S4, or S5, and that is initialized to S1
    - state <= S5;
    - state <= S10;

VHDL signals

- **Signals** carry data between concurrent blocks, they are **abstract 'wires'**. Can be as simple as a single bit or as complicated as a struct.

Tutes

Tute 1

		Inputs	Outputs
---	---	---	---
M	In_0	In_1	Out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$M \setminus I_1 I_0$	00	01	11	10
0	0	1	1	0
1	0	0	1	1