SEE3243 Digital System

Lecturers :
Muhammad Mun'im Ahmad Zabidi
Muhammad Nadzir Marsono
Kamal Khalil

Week 1: Introduction

"Teacher only opens the door for you, you have to enter by yourself"
Old Chinese Proverb

Why this course?

- This course is an extension to SEE 1223. It is about digital logic **design**.
- Both types of circuits (digital and analogue) are used in practice, but digital circuits are much more prevalent than analogue circuits. Why?
- Analogue data is more precise than digital data because digital data (discrete) is an approximation of analogue data (continuous).
- Digital systems are usually more accurate than analogue systems because they are less vulnerable to noise.

Syllabus

Topic	Week
1. Introduction: Hierarchical design, CAD software.	1
2. Logic simplification: SOP/POS logic, De-Morgan Theorem, Entered-variable Karnaugh map, **Introduction to hazard & glitches**.	1
3. Logic design using MSI components and PLD: multiplexor, decoder, ROM, PLA, PAL, GAL, tristate, introduction to FPGA & CPLD.	2
4. Arithmetic circuits: half-adder, full adder, ripple-carry adder, subtractor, CLA adder, ALU, combinational multiplier, Design Trade-off.	2
5. Sequential circuits: synchronous & asynchronous circuits, latches & flip-flops, characteristic equations, metastability.	1

Syllabus (cont'd)

Topic	Week
6. **Registers & Counters**: Registers File, shift registers, counters, state diagrams, synthesis of synchronous counters.	1
7. **Finite State Machines (FSM):** State diagrams for FSM, Moore & Mealy models, design of sequence detectors, state encoding.	2
8. **Advanced FSM Realization**: Design of up/down counter without and with enable. Design of vending machine. Design of traffic light controller.	2
9. **Case Studies:** Datapath and control units	2

Book(s)

- Textbook:
  - Randy H. Katz and Gaetano Borriello**,** *Contemporary Logic Design.* 2nd ed. Upper Saddle River, NJ: Pearson Education, Inc., 2006.
- Other references
  - Donald D. Givone**,** *Digital Principles and Design***.** International ed. Singapore: McGraw-Hill, 2003.
  - Stephen Brown and Zvonko Vranesic, *Fundamentals of Digital Logic with VHDL Design.* 2nd ed. Singapore: McGraw-Hill, 2005.
  - Alan B. Marcovitz, *Introduction to Logic Design***.** 2nd ed. New York, NY: McGraw-Hill, 2005.

Misc – CAD skills

- We will use Quartus II - won’t be taught in class.
- Familiarize yourself with this EDA.
- You have to learn it yourself. Tutorials will be given to assist you.
- Lecture notes will be made available at UTM E-Learning Server <http://elearning.utm.my>

Major Topics To Be Discussed

- Fundamental digital design skills (data types, Boolean algebra, minimization techniques)
- Combinational circuits (circuits *without* memory)
- Introduction to arithmetic circuitry
- Sequential circuits (circuits *with* memory)
- Finite/Algorithmic State Machines. Some case studies.

Assumptions

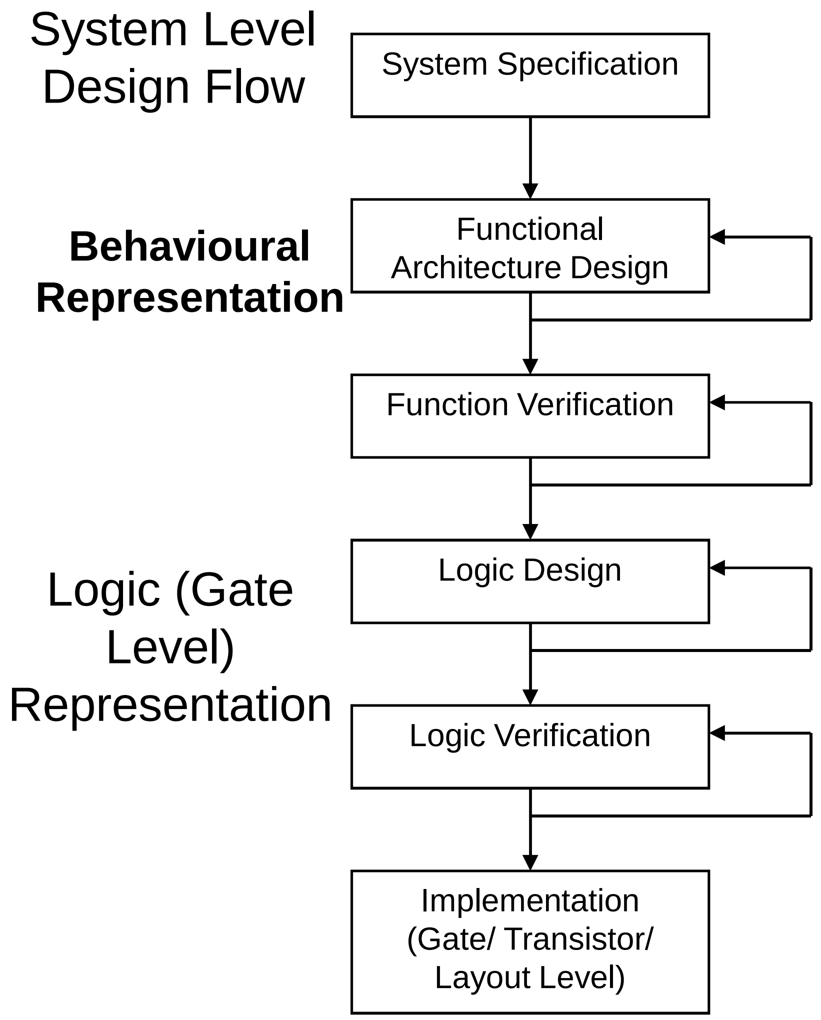
- You’re all well versed in
  - Data and number representation and operations
  - Boolean algebra
  - Logic gates
  - Simple minimization techniques (up to 4-variable Karnaugh maps)
- All these topics you’d learnt in SEE 1223.

What you can expect

- Exercises, for you to do on your own
- Usage of Quartus II CAD

Hierarchical Design

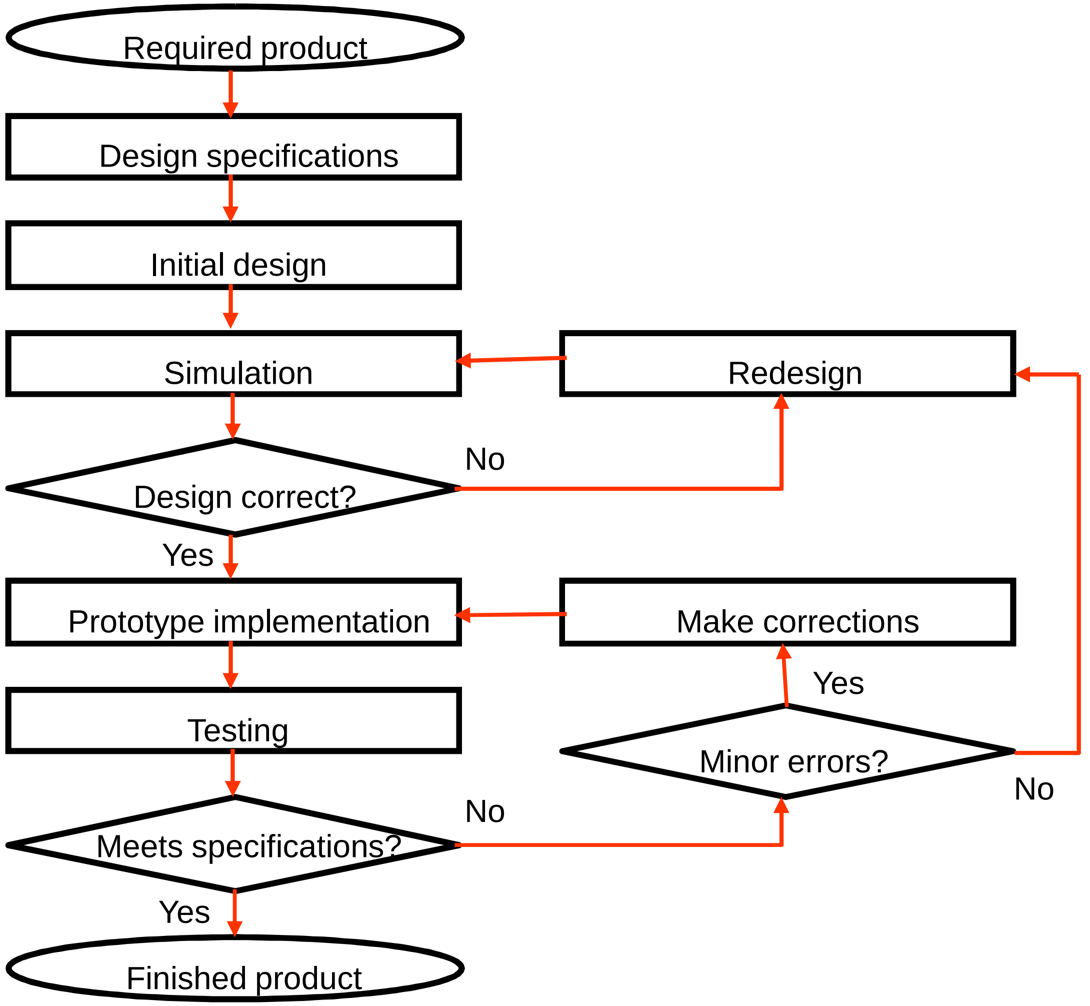
- **Definition:** Hierarchy, or *"divide and conquer"* .
- Dividing a module into sub-modules and then repeating this operation on the sub-modules until the complexity of the smaller parts becomes manageable.


System Level Design Flow
System Specification
Behavioural Representation
Functional Architecture Design
Function Verification
Logic (Gate Level) Representation
Logic Design
Logic Verification
Implementation (Gate/ Transistor/ Layout Level)

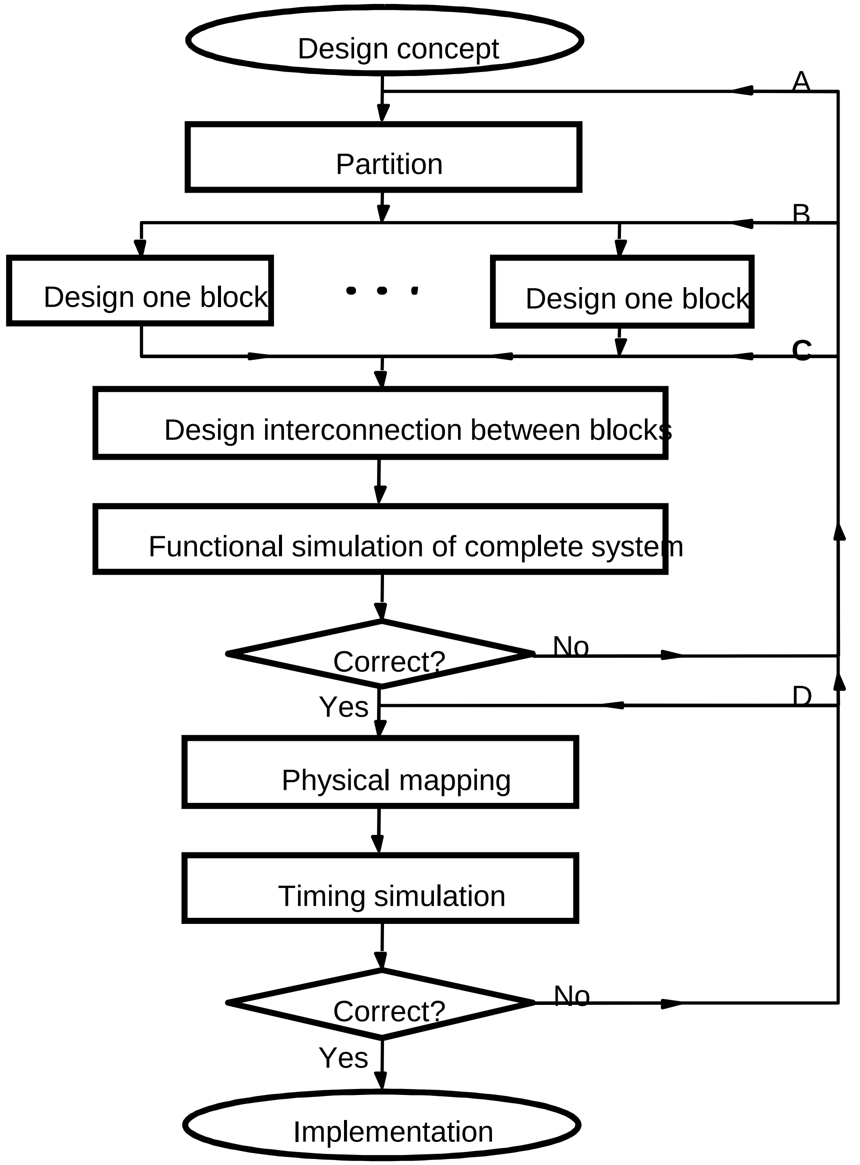
Top-down vs Bottom-up

- Top-down design flow provides an excellent design process control.
- In reality, there is no truly unidirectional approach.
- Both top-down and bottom-up approaches have to be combined. In system level design, in order to fit the system into the allowable constraint (area, speed, power consumption) some functions may have to be removed and the design process must be repeated (may require significant modifications).

Development Process



Design Flow for Logic Circuits



Concept of Regularity, Modularity and Locality

- **Regularity**
  - Means that the hierarchical decomposition of a large system must be simple and similar as much as possible. It must exists at all levels of abstraction.
  - Eg: At the logic level, identical gate structures can be used, etc. If the designer has a small library of well-defined and well-characterized basic building blocks, a number of different functions can be constructed by using this principle.

Concept of Regularity, Modularity and Locality

- **Modularity**
  - Hierarchical functional blocks must be well-defined – functionality and interfaces.
  - Each block can be designed independently (relatively from each other).
  - All of the blocks can be combined with ease to form the large system.
  - Enables the parallelization of the design process.

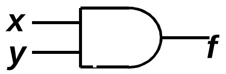
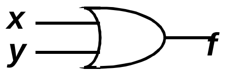
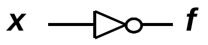
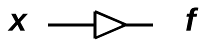
Concept of Regularity, Modularity and Locality

- **Locality**
  - The well-characterized definition of interfaces for each module in the system stays at the local level.
  - Thus, the internals of each module become unimportant to the exterior modules.
  - Connections are mostly between neighbouring modules, avoiding long-distance connections as much as possible to avoid interconnect delay. Time-critical operations should be performed locally.

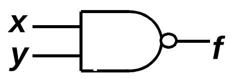
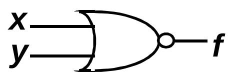
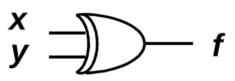
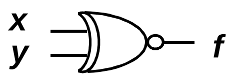
Where Are You Now?

- Assumption: you know basics of logic theory, Boolean algebra, Karnaugh map and stuffs from SEE 2222.
- This is only for review. Get a book!
- Lets take a sneak review…
- ***Slides 1-19 to 1-35*** are only for review. You can skip these slides.

Logic Gates: Revisited

Name	Graphical Symbol	Algebraic Function	Truth Table
AND		***f* = *x.y***	*x y* | *f* 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		***f* = *x* + *y***	*x y* | *f* 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
INVERTER		***f* = *x’***	*x* | *f* 0 | 1 1 | 0
BUFFER		***f* = *x***	*x* | *f* 0 | 0 1 | 1

Logic Gates: Revisited

Name	Graphical Symbol	Algebraic Function	Truth Table
NAND		***f = (x.y)’***	*x* *y* | *f* 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		***f = (x + y)’***	*x* *y* | *f* 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
EX-OR		***f = x’y + xy’***	*x* *y* | *f* 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
EX-NOR		***f = x’y’ + xy***	*x* *y* | *f* 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Boolean Expression

- Boolean expressions are a much better form for representing digital circuits because it is much easier to manipulate and simplify.
- A Boolean expression is an expression formed with:
  - binary variables
  - the binary operators **OR** and **AND**
  - the unary operator **NOT**
  - parentheses
  - an equal sign
- For example,
  - $F = x'y + z$ ***F*** is **1** when $z = 1$ **OR** when $x = 0$ **AND** $y = 1$.

Operator Precedence

- The precedence of operations is as follows:
  - parentheses
  - NOT
  - AND
  - OR

Boolean Algebra

- **Definition**: Theorems that are used at design time to manipulate and simplify Boolean expressions for easier and less expensive implementation.
- Any Boolean expression can be represented using only **AND, OR**, and **NOT** operations.
- May need to use Boolean algebra to change the form of a Boolean expression to better utilize the types of gates provided by the component library being used.
- A Boolean variable, x, can have two values, typically 1 and 0 (on and off)

Properties of Boolean Algebra

- Identity Elements
  - X+0=X
  - X • 1=X (Dual of previous)
- Commutative property
  - A+B=B+A
  - A• B=B• A (Dual of previous)
- Associative property
  - A + (B+C) = (A+B)+C
  - A• (B•C) = (A•B)•C

- Distributive of '+ over •' and '• over +'
  - A+(B•C) = (A+B) • (A+C)
  - A• (B+C) = (A•B)+(A•C)
- Existence of the complement
  - A+A' = 1
  - A•A' = 0

Duality

- Every Boolean expression has a dual
- If the expression is valid, then the dual is valid
- To obtain the dual:
  - Replace all + with · and all · with +
    - A+(BC) = (A+B)(A+C)
    - A(B+C) = AB+AC
  - Keep parenthesis order
  - Replace ‘1’ with ‘0’ and vice versa
- Duality can be used to prove theorems and allow simple transformation of Boolean functions
- Also makes it easy to find other forms of a theorem

Variable Theorems

- *Idempotency*
  - A+A = A
  - A·A = A
- *Null elements* for + and · operators
  - A+1 = 1
  - A·0 = 0
- *Involution*
  - (A')' = A

- *Absorption*
  - A+AB = A
  - A(A'+B) = AB
  - AB+AB' = A
  - (A+B)(A+B') = A
  - AB+AB'C = AB+AC
  - (A+B)(A+B'+C) = (A+B)(A+C)
- *DeMorgan's*
  - (A+B)'= A'·B'
  - (A·B)'= A'+B'
- *Consensus*
  - AB+A'C+BC=AB+A'C

Some Definitions

- ***Literal*** - a variable or complement of the variable in terms
- ***Product term*** - single literal or product (·) of two or more literals,
  - e.g.: ABC
- ***Sum term*** - single literal or sum (+) of two or more literals,
  - e.g.: A+B+C
- ***minterm*** – normal product term of *n* literals that is 1 for exactly one set of input values
  - 2^n unique ***n***-variable minterms
  - 4-variable minterm – A’B’C’D’, A’B’C’D …. ABDC (16 possible terms)
- ***maxterm*** – normal sum term of *n* literals , expression that is 0 for exactly one set of input values
  - 2^n unique ***n***-variable maxterms
  - 4-variable maxterm – A+B+C+D, ….. A’+B’+C’+D’ (16 possible terms)

Minterm – Maxterm relationship

- $M_i = m_i'$
- Proof
  - At row 5,

    $m_5 = AB'C$

    $m_5' = (AB'C)'$

    $= A' + B + C'$

    $= M_5$

Decimal Number	ABC	Minterm	Maxterm
0	000	$A'B'C'=m_0$	$A+B+C=M_0$
1	001	$A'B'C=m_1$	$A+B+C'=M_1$
2	010	$A'BC'=m_2$	$A+B'+C=M_2$
3	011	$A'BC=m_3$	$A+B'+C'=M_3$
4	100	$AB'C'=m_4$	$A'+B+C=M_4$
5	101	$AB'C=m_5$	$A'+B+C'=M_5$
6	110	$ABC'=m_6$	$A'+B'+C=M_6$
7	111	$ABC=m_7$	$A'+B'+C'=M_7$

Forms of Boolean Expression

Complement

- Use DeMorgan's theorem
- DeMorgan's theorem states:
  - (X + Y)' = X' * Y'
- DeMorgan's theorem can be extended to 3 or more variables.
- Example
  - Given **(X + Y + Z)'** Let A = Y + Z
  - (X + A)' = (X' * A')
  - Substituting back in Y + Z
  - = (X' * (Y + Z)')
  - = X' * Y' * Z'

- The compliment of a function can be obtained by interchanging **AND**'s and **OR**'s and complementing each literal.
- Parenthesis may need to be included to keep the order of the evaluation.
- Remember, that in the absence of parenthesis, **AND** has precedence over **OR** operation.

Canonical SOP and POS

- *Canonical SOP Of A Function*
  - a function represented as a sum of minterms
  - F(A,B,C) = A'BC'+ABC'+A'BC+ABC
- *Canonical POS Of A Function*
  - a function represented as a product of maxterms
  - F = (A+B'+C)(A+B'+C')(A'+B+C')
- Any function can be represented as a canonical POS or SOP form, which is either an two-level ***AND-OR tree*** or a ***OR-AND tree***

Example

- In SOP
  - x’y’z+ x’yz’+ x’yz+ xyz
  - Σm(1,2,3,7)
- In POS
  - (x+y+z) (x’+y+z) (x’+y+z’) (x’+y’+z)
  - ΠM(0,4,5,6)
- For the same function F
  - Σm(1,2,3,7)= ΠM(0,4,5,6)

Row	xyz	Minterm	Maxterm	F
0	000	x’y’z’	x+y+z	0
1	001	x’y’z	x+y+z’	1
2	010	x’yz’	x+y’+z	1
3	011	x’yz	x+y’+z’	1
4	100	xy’z’	x’+y+z	0
5	101	xy’z	x’+y+z’	0
6	110	xyz’	x’+y’+z	0
7	111	xyz	x’+y’+z’	1

Don’t Cares

- Very often, the specification of a function is incomplete
- Output state is unimportant for that particular set of inputs or input state never occurs
- Any input combination whose state is unimportant is a “don’t care” state (*d* in SOP and *D* in POS)
- Useful feature for minimization of states
- Example, with minterms AB’C (101) and ABC’(110) are don’t cares
  - Minterm – $F(A,B,C) = \Sigma m(0,1,2) + \Sigma d(5,6)$
  - Maxterm – $F(A,B,C) = \Pi M(3,4,7) \cdot \Pi D(5,6)$

Limitation of Boolean Algebra

- There is no algorithm you can follow that is guaranteed to lead to the simplest form of the expression
- Given any intermediate result there is no way to tell if it is in fact the simplest form of the expression

DIY Example

- Given the following SOP expression, minimize it:
  - $F(x,y,z) = x'y'z' + x'y'z + xy'z' + xy'z + xyz'$

- Minimization via the application of Boolean algebra is error prone, especially if there are large equations.