Recap from last lecture

- **Programmable logic**
  - Programmable logic devices (PLDs)
    - Read-only memory (ROM)
    - Programmable Logic Array (PLA)
    - Programmable Array-Logic (PAL)
  - Field programmable gate arrays (FPGAs)
    - Lookup-table (LUT) based design

- **Today’s lecture**
  - Discuss lab4 (Traffic Light Controller), verilog
  - Register Transfer Level (RTL) design
    - Datapath design
    - Control design
  - Algorithmic State Machines (ASM)
Look-up Table based FPGA

- **Look-up Table**
  - Truth table implemented in hardware
  - Can implement arbitrary function with fixed number of inputs (typically 4-5)

  ![2-Input LUT Diagram](image)

  **Example:**
  
  \[ F = x_1'x_2' + x_1x_2 \]

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

FPGA Routing

- **Programmable routing**
  - Switch-boxes (S)
  - I/O connectors (C)
Designing Large Circuits

- Most practical application use larger digital circuits
  - E.g., pocket calculator, embedded controller, microprocessors, etc.

- Sequential circuit design procedure
  - Specification of operation of every flip-flop
  - Not scalable beyond small circuits

- Modularization can abstract components in circuit
  - What is the right level of abstraction?
  - Many possible choices:
    - Flip-flops (too detailed, low level)
    - Group by functionality (e.g., decoder, adder)
    - Group by data (e.g., register)

- Register Transfer Level (RTL) notation
  - Register-level abstraction of circuit

Register Transfer Level (RTL)

- RTL specification of digital system
  - System uses a set of registers
  - Operations are performed on data stored in registers
  - Control supervises the sequence of operations

- Examples:
  - Transfer of data from one register to another: R2 ← R1
    - Value of R1 is stored into R2
    - R1 maintains prior value, R2 is over-written
  - Conditional transfer:
    - If (T1=1) then (R2 ← R1)
    - Transfer happens only if T1 has value 1
  - Multiple operations:
    - If (T1=1) then (R2 ← R1, R1 ← R2)
    - Swap of values in R1 and R2

- Note:
  - Operations always happen on clock edges (positive or negative, or both)
### RTL Operations

- **Typical operations:**
  - Transfer operations transferring data between registers
  - Arithmetic operations performing arithmetic on data in registers
  - Logic operations performing bit manipulation of non-numeric data in registers
  - Shift operations shifting data in registers

- **Examples:**
  - $R1 \leftarrow R1 + R2$
  - $R3 \leftarrow R3 + 1$
  - $R4 \leftarrow \text{shr} \ R4$
  - $R5 \leftarrow 0$

- **Most operations are intuitive**
  - Exception: logic vs. logical operations

### RTL Operations in Verilog

- **Verilog RTL logic operations:**
  - Complement: $\sim$
  - AND: $\&$
  - OR: $\mid$
  - Exclusive-OR: $^\wedge$

<table>
<thead>
<tr>
<th>bitwise or reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ (10) = 01</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

- **Logic operations performed bit-wise or reduction (single operand):**
  - $\sim (10) = 01$
  - $| (10) = 1$

- **Verilog RTL logical operations:**
  - Negation: $!$
  - AND: $&&$
  - OR: $||$

- **Logical operations performed on multiple operands**
  - “True” represented by 1, “false” represented by 0
  - $10100 && 00000 = 0$; $10100 || 00000 = 1$

- **Arithmetic operations:** $+, -, /, \*, \%$
Separation of Control and Data

- **Data processing path (Datapath)**
  - Processes data, discrete elements of information
  - Part of the system, which operates on data
    - Registers, adders, comparators, ALU, etc.
    - Well structured, design well understood
    - Common to many systems, implemented with standard components

- **Control logic**
  - Part of the system which controls datapath
    - Provides command signals
    - Much smaller than datapath
    - Unique to every ckt
    - Requires custom design

- **Control and datapath:**

```
12/04/06 Engin 112 - Intro to ECE 9
```

Algorithmic State Machines (ASM)

- **Control logic**
  - Controls sequence of operations in the datapath
    - E.g., triggers transfer of register value followed by addition
  - Sequential circuit with different control states

- **Representation of control logic:**
  - “Algorithmic State Machine” (ASM)

- **ASM flow chart describes sequence of events**
  - Contains “state box,” “decision box,” and “conditional box”

```
12/04/06 Engin 112 - Intro to ECE 10
```
ASM Block

- State boxes form state diagram:

- All operations between states happen in synch and in one clock cycle

Design Example – binary multiplier

- Design of sequential multiplier

- Block diagram:
  - How does it work?
Multiplier Operation

- **Initialization:**
  - Load Multiplicand into \( B \)
  - Load Multiplier into \( Q \)
  - Load number of bits \( n \) into \( P \)

- **Start:**
  - Multiplication begins when \( S=1 \)

- **Each step:**
  - If \( Q_0 \) (LSB in \( Q \)) is 1, then \( A \leftarrow A + B \)
  - Shift right \( CAQ \), \( C \leftarrow 0 \)

- Control logic activates all functions at the right time

Multiplier ASM

- ASM representation of multiplier?

Datapath
Homework

- Read Mano
  - 8-5 – 8-7