Summary of Combinational Logic

- Combinational device/circuit: any circuit built using the basic gates
- Expressed as
  - Truth table
  - Digital circuit
  - Boolean function
- Any boolean function can be expressed as two level function
- Minimization procedure: Karnaugh Map
  - Try to minimize the number of gates, and inputs to gates, in a two level circuit

Combinational vs. Sequential

- **Combinational Circuit**
  - always gives the same output for a given set of inputs
    - ex: adder always generates sum and carry, regardless of previous inputs
- **Sequential Circuit**
  - stores information
  - output depends on stored information (state) plus input
    - so a given input might produce different outputs, depending on the stored information
  - example: vending machine
    - Current total increases when you insert coins
    - output depends on previous state
  - useful for building "memory" elements and "state machines"

Sequential Logic

- Build a device, using combinational logic devices, to store a value
  - RS Latch (also called SR Latch)
  - concept of memory
- Methodology behind design of sequential logic circuits
  - Finite State Machines
  - Example of Vending machine
- Combine sequential and combinational logic devices to “assemble” a simple processor!
Sequential Circuits – Storage Elements

- We need to design a device capable of storing information
  - Store a bit value
- Build it using the devices we have thus far
  - Use “feedback” circuit
- To be useful, sequential device needs mechanism for setting its state
- R-S Latch:
  - Output = previous value
  - Or Set output to new value (0 or 1), and hold this new value till next “write” into the device

Feedback Circuits

- Stable circuit
  - Output point of circuit retains value indefinitely
- Unstable circuit
  - State that remains constant only for a duration of a few gate delays

Feedback Circuits

- To retain their state values, sequential circuits rely on feedback.
- Feedback in digital circuits occurs when an output is looped back to the input.
- A simple example of this concept is shown below.
  - If Q is 0 it will always be 0, if it is 1, it will always be 1. Why?

Feedback Circuits

- What if we had three gates in the circuit?
**R-S Latch: Simple Storage Element**

- **R** is used to "reset" or "clear" the element – set it to zero.
- **S** is used to "set" the element – set it to one.

If both **R** and **S** are one, out could be either zero or one.
- "quiescent" state – holds its previous value
- note: If **a** is 1, **b** is 0, and vice versa

**R-S Latch Summary**

- **R = S = 1**
  - hold current value in latch
- **S = 0, R=1**
  - set value to 1
- **R = 0, S = 1**
  - set value to 0
- **R = S = 0**
  - both outputs equal one
  - final state determined by electrical properties of gates
  - Don’t do it!

**Clocked Flip-Flops/Circuits**

- Subsystem in a computer consists of a large number of combinational and sequential devices
  - Each sequential device is like an SR latch which is in one of two states
  - As machine executes its cycle, the states of all sequential devices change with time
- To control large collection of devices in an orderly (synchronized) fashion, machine maintains a clock
  - Requires all devices to change their states at the same time
  - Clock generates sequence of pulses

**Clock**

- As the name implies, sequential logic circuits require a means by which events can be sequenced.
- State changes are controlled by clocks.
  - A "clock" is a special circuit that sends electrical pulses through a circuit.
- Clocks produce electrical waveforms such as:
Clocked Circuits

- State changes occur in sequential circuits only when the clock ticks.
- Circuits can change state on (a) the rising edge or falling edge – edge triggered circuits, or (b) when the clock pulse reaches its highest voltage level – level triggered.
  - Time between pulses is the period of the clock

Clocked RS Latch – Flip Flop

- Every sequential device has a Clock (CK) input in addition to its other inputs
  - Device designed to respond to inputs only during a clock pulse
- Where do we add clock to the RS Latch?
  - Clocked RS Latch = RS Flip Flop
- Shield NAND gates (latch) from effect of S,R except when clock is high
  - When clk is low the inputs to the NAND gates are both 1 regardless of S,R
  - No change in latch output value
  - When clk is high, values of R and S pass to the NAND gates – i.e., latch

Clocked Latches – Flip Flop

SR Flip Flop: Reverse of RS

- More common latch is the SR latch
  - Have 11 input as undefined
  - 00 holds state
  - Set S=1 to set latch to 1
  - Set R=1 to set latch to 0
- Replace NAND gates by NOR gates!
SR Latch/Flip-flop

- The most basic sequential logic components, the SR flip-flop.
  - The “SR” stands for set/reset.

SR Latch Behaviour

- The behavior of an SR flip-flop is described by a characteristic table.
- Q(t) means the value of the output at time t. Q(t+1) is the value of Q after the next clock pulse.

<table>
<thead>
<tr>
<th>S R</th>
<th>Q(t) (no change)</th>
<th>0 1</th>
<th>D (reset to 0)</th>
<th>1 0</th>
<th>1 (set to 1)</th>
<th>1 1</th>
<th>undefined</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Q(t)</td>
<td>0 1</td>
<td>0</td>
<td>1 0</td>
<td>1</td>
<td>1 1</td>
<td>undefined</td>
</tr>
</tbody>
</table>

SR Flip Flop

- The SR flip-flop actually has three inputs: S, R, and its current output, Q.
- Thus, we can construct a truth table for this circuit, as shown to the right.
- Notice the two undefined values. When both S and R are 1, the SR flip-flop is unstable.

Concept of Memory and Write to Memory

- We want ONE input to the latch
- We want to control when to “update” value in the latch (output)
- When write is enabled, we want Output = Value of Input
  - Else output = old value of output
**Gated D-Latch**

- Two inputs: D (data) and WE (write enable)
  - When \( WE = 1 \), latch is set to value of D
    - \( S = \text{NOT}(D) \), \( R = D \)
  - When \( WE = 0 \), latch holds previous value
    - \( S = R = 1 \)

- 

**D Flip Flop Behaviour**

- Typical modeling of D flip flop simplifies things
  - Input \( D = 1 \) then at clock pulse, output of flip flop becomes 1
  - Input \( D = 0 \) then at clock pulse, output becomes 0

**D Flip Flop**

- The D flip-flop is the fundamental circuit of computer memory.
  - D flip-flops are usually illustrated using the block diagram shown below.
- Modify latch to make it a clocked flip flop
- The characteristic table for the D flip-flop is shown at the right.

**Next... Storage Devices**

- Temporary storage in a computer?
  - Where are variables stored before being sent to the arithmetic unit for operations on them?
- Register
  - Can we build an n-bit register using latches?
- What about “main” memory
- Disk
  - Later...
**Register**

- A register stores a multi-bit value.
  - We use a collection of D-latches, all controlled by a common WE.
  - When WE = 1, n-bit value D is written to register.

**Representing Multi-bit Values**

- Number bits from right (0) to left (n-1)
  - just a convention -- could be left to right, but must be **consistent**
- Use brackets to denote range:
  - D[l:r] denotes bit l to bit r, from *left to right*

\[ A = 0101001101010101 \]

\[ A[14:9] = 101001 \]

\[ A[2:0] = 101 \]

*May also see A<14:9>, especially in hardware block diagrams.*

**Memory**

- We know how to store m-bit number in a register
- How about many m-bit numbers?
  - Bank of registers?
- How to fetch a specific m-bit number?
  - addressing

**Memory**

- Now that we know how to store bits, we can build a memory – a logical \( k \times m \) array of stored bits.

- **Address Space**:
  - number of locations
  - (usually a power of 2)

- **Addressability**:
  - number of bits per location
  - (e.g., byte-addressable)
Looking from the outside, what do we need?

### Address Space

- A large number of addressable fixed size locations.
  - n bits allow the addressing of $2^n$ memory locations.
    - Example: 24 bits can address $2^{24} = 16,777,216$ locations (i.e. 16M locations).
    - If each location holds 1 byte (= 8 bits) then the memory is 16MB.
    - If each location holds one word (32 bits = 4 bytes) then it is 64 MB.

### Addressability

- Computers are either byte or word addressable - i.e. each memory location holds either 8 bits (1 byte), or a full standard word for that computer (16 bits for the LC-3, more typically 32 bits, though now many machines use 64 bit words).
  - Normally, a whole word is written and read at a time:
    - If the computer is word addressable, this is simply a single address location.
    - If the computer is byte addressable, and uses a multi-byte word, then the word address is conventionally either that of its most significant byte (big endian machines) or of its least significant byte (little endian machines).
Memory

- Given address, fetch contents at that address
  - Select or enable one of many locations

**Memory Example**

A 2^2 by 3 bits memory:
- two address lines: A[1:0]
- three data lines: D[2:0]
- one control line: WE

Building a Memory

- Each bit
  - is a gated D-latch
- Each location
  - consists of w bits (here w = 1)
  - w = 8 if the memory is byte addressable
- Addressing
  - n locations means log_2 n address bits (here 2 bits => 4 locations)
  - decoder circuit translates address into 1 of n locations

Reading a location in memory

One gated D-latch
More Memory Details

- This is no longer the way actual memory is implemented.
  - Fewer transistors, much more dense,
    relies on electrical properties
- But the logical structure is very similar.
  - Address decoder
  - Word select line
  - Word write enable
- Two basic kinds of RAM (Random Access Memory)
  - Static RAM (SRAM)
    - Fast, maintains data as long as power applied
  - Dynamic RAM (DRAM)
    - Slower but denser, bit storage decays – must be periodically refreshed

Also, non-volatile memories: ROM, PROM, flash, ...

Course Admin-trivia

- Lab this week and next is Project 1 completion
  - Project 1 code
    - Dlc 'code checker' tells you if rules violated and if there is a syntax error
    - Review sequential logic
  - HW3, Teamwork 3, Teamwork 4 posted
    - Solutions to HW1,2 and Quiz 1,2 posted
    - Work on Teamwork4 in class next class
- Quiz 3 next Tuesday
- Exam 1: October 7
- Reading:
  - You should have completed reading Chapters 1,2,3
  - Look up Univ. Maryland notes linked from lectures webpage

Design of Sequential Logic Circuits

Summary of Digital Logic

- Combinational logic
- Storage elements
  - R-S Latches and D-latch
  - Concept of memory: address space and addressability
- Sequential circuits – next!
  - Recall the vending machine example
    - Progress from state (current total) to new state (new total)
**Compare and Contrast**

- **Combinational Logic Circuits**
  - Make decisions
  - Same inputs always produce same output
  - Depends on what is happening now

- **Sequential Logic Circuits**
  - Make decisions and store information
  - Output depends on inputs AND state
  - Depends on what has happened in the past as well as what is happening now

**A Vending Machine**

- **Input valid coins:**
  - Q (25 cents) D (10) or N (5)

- **Keep track of current total**
  - Is it 75 cents or more?

- **When it reaches 75 or more:**
  - Generate output

- **States of the machine?**

**Finite State Machines**

- The behavior of sequential circuits can be expressed using characteristic tables or finite state machines (FSMs).
  - FSMs consist of a set of nodes that hold the states of the machine and a set of arcs that connect the states.
  - Directed graph to represent a FSM

- Moore and Mealy machines are two types of FSMs that are equivalent.
  - They differ only in how they express the outputs of the machine.
  - Moore machines place outputs on each node/state
  - Mealy machines present their outputs on the transitions.

**State Machine**

- **Type of sequential circuit**
  - Combines combinational logic with storage
  - "Remembers" state, and changes output (and state) based on inputs and current state

![State Machine Diagram]

- **Inputs**
- **Outputs**
Sequential Logic Circuits - State

- The concept of state
  - The state of a system is a “snapshot” of all relevant elements at a moment in time.
  - A given system will often have only a finite number of possible states.
  - For many systems, we can define the rule which determines under what conditions a system can move from one state to another.

Example

- The game of tic-tac-toe has only a certain number of possible dispositions of Xs and Os on the 3x3 grid.

- A given game of tic-tac-toe will progress through a subset of these possible states (until someone wins) - i.e. it traverses a specific path through “state space”, one move at a time.

Finite State Machines

- Many systems meet the following five conditions:
  - A finite number of states
  - A finite number of external inputs
  - A finite number of external outputs
  - An explicit specification of all allowed state transitions
  - An explicit specification of the rules for each external output value
- A microprocessor is a perfect candidate for description as a FSM.
The Clock

- Frequently, a clock circuit triggers transition from one state to the next.

At the beginning of each clock cycle, state machine makes a transition, based on the current state and the external inputs.
  > Not always required.

Storage: Master-Slave Flipflop

- A pair of gated D-latches, to isolate next state from current state.

During 1st phase (clock=1), previously-computed state becomes current state and is sent to the logic circuit.

During 2nd phase (clock=0), next state, computed by logic circuit, is stored in Latch A.

Storage

- Each master-slave flip-flop stores one state bit.

- The number of storage elements (flip-flops) needed is determined by the number of states (and the representation of each state).
  > Each bit can be 0 or 1 = 2 states
  > N bits can represent $2^N$ states

Example: Blinking Traffic Sign

- Example: If a FSM has 12 states, then the circuit needs $\log_2 12 = 4$ storage elements.
  > Fewer the states, less hardware needed
  > Concept of Minimization of States for a given FSM

Example: Blinking Traffic Sign

- Each bit can be 0 or 1 = 2 states
- N bits can represent $2^N$ states
- Example: If a FSM has 12 states, then the circuit needs $\log_2 12 = 4$ storage elements.
  > Fewer the states, less hardware needed
  > Concept of Minimization of States for a given FSM
DANGER
MOVE RIGHT

1
3
5
2
4

DANGER
MOVE RIGHT

1
3
5
2
4

DANGER
MOVE RIGHT

1
3
5
2
4

DANGER
MOVE RIGHT

1
3
5
2
4
Example: Traffic Sign

- A blinking traffic sign: How many states
  - 4 states
    - No lights on
    - 1 & 2 on
    - 1, 2, 3, & 4 on
    - 1, 2, 3, 4, & 5 on
      (repeat as long as switch is turned on)
- How many bits to represent the 4 states
  - \( S_1S_0 \)
    - With \( S_1S_0 \) values: 00, 01, 10, 11

Traffic Sign State Diagram

- Note we really have 3 groups of lights to be controlled = 3 control lines \( X, Y, Z \)
  - Group 1: Lights 1 and 2; controlled by \( Z \)
    - If \( Z=1 \) then Group 1 lights (1 and 2) are switched on
  - Group 2: lights 3 & 4; controlled by \( Y \)
  - Group 3: Light 5; controlled by \( X \)
Finite State Machine Example - 2

- When is group 1 on?
  - in states 01, 10 and 11 - but only when the switch IN is on!

- Logic expressions for X,Y,Z
  - Depends on S0 and S1, and Input is on
    - if input is off then X,Y,Z are at 0

- can you come up with a logic expression for next state values of S0 and S1?
  - Depends on current values of S0 and S1, and Input is on
    - Input off than both bits are set to 0 since next state is 00
    - Next state value of S0 denoted S’0 = 1 if current state is 00 or current state 10 and In=1

- When do we switch to the next state?
  - the two bits of S[1:0] are updated at every clock cycle
  - we have to make sure that the new state does not propagate to the combinational circuit input until the next clock cycle.

Traffic Sign Truth Tables

<table>
<thead>
<tr>
<th>S1</th>
<th>S0</th>
<th>Z</th>
<th>Y</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Next State: S1’S0’
(depend on state and input)

<table>
<thead>
<tr>
<th>Switch</th>
<th>S1</th>
<th>S0</th>
<th>S1’</th>
<th>S0’</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Whenever In=0, next state is 00.

Traffic Sign Logic

- Z = (S1S0 + S1S0 + S1S0).In
- Y = (S1S0 + S1S0).In
- X = (S1S0).In
The data path of a computer is all the logic used to process information. 
- Eg. data path of the LC-3.

**Combination Logic**
- Decoders -- convert instructions into control signals
- Multiplexers -- select inputs and outputs
- ALU (Arithmetic and Logic Unit) -- operations on data

**Sequential Logic**
- State machine -- coordinate control signals and data movement
- Registers and latches -- storage elements

**What Next?**
- Next topic: The von Neumann model of computer architecture
  - Basic components
  - How instructions are processed
  - The LC3 computer and instruction set
Recall: what are Computers meant to do?

- We will be solving problems that are describable in English (or Greek or French or Hindi or Chinese or ...) and using a box filled with electrons and magnetism to accomplish the task.
  - This is accomplished using a system of well defined (sometimes) transformations that have been developed over the last 50+ years.

Problem Transformation
- levels of abstraction

- The desired behavior: the application
  - Natural Language
  - Algorithm
  - Program
  - Machine Architecture
  - Micro-architecture
  - Logic Circuits
  - Devices

- The building blocks: electronic devices

Putting it all together

- The goal:
  - Turn a theoretical device - Turing’s Universal Computational Machine - into an actual computer ...
  - ... interacting with data and instructions from the outside world, and producing output data.

- Smart building blocks:
  - We have at our disposal a powerful collection of combinational and sequential logic devices.

- Now we need a master plan ...