



CSE140L: Components and Design Techniques for Digital Systems Lab

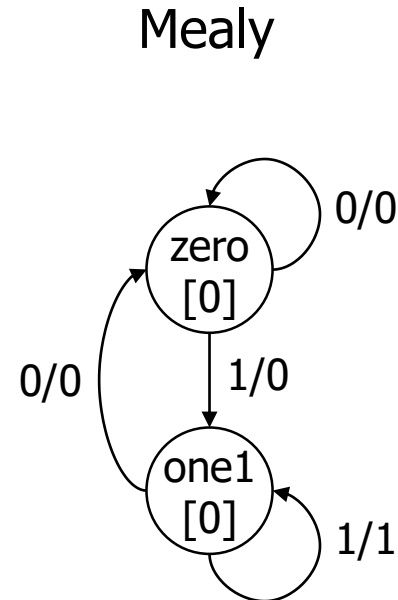
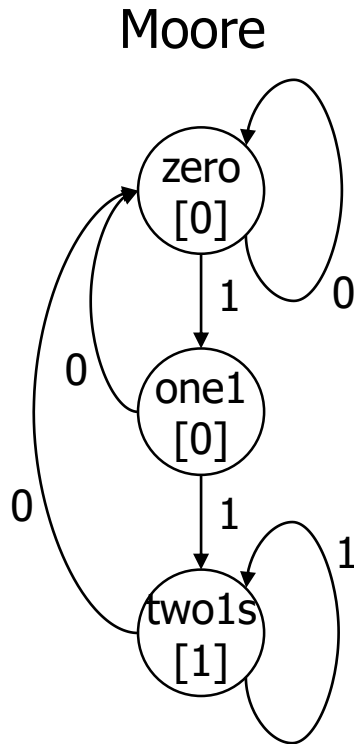
FSMs

Instructor: Mohsen Imani

Slides from Tajana Simunic Rosing

FSM design example – Moore vs. Mealy

- Remove one 1 from every string of 1s on the input

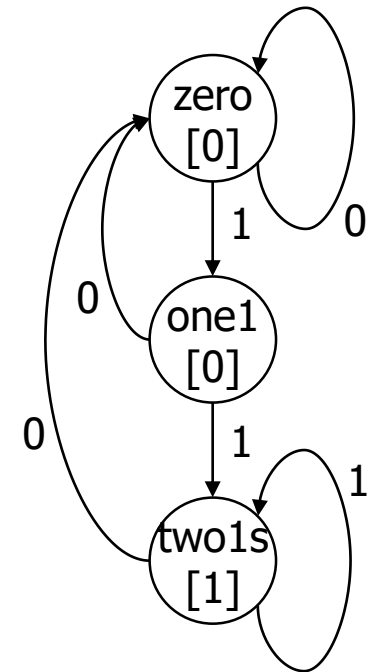


Verilog FSM - Reduce 1s example

- Moore machine

```
module reduce (clk, reset, in, out);  
  input clk, reset, in;  
  output out;  
  
  parameter zero    = 2'b00;  
  parameter one1    = 2'b01;  
  parameter twos    = 2'b10;  
  
  reg out;  
  reg [1:0] state;    // state variables  
  reg [1:0] next_state;  
  
  always @(posedge clk) begin  
    if (reset) state = zero;  
    else      state = next_state;  
  end
```

state assignment
(easy to change,
if in one place)



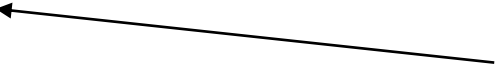
Always include a reset signal !!!

Moore Verilog FSM (cont'd)

```
always @(in or state)
begin

case (state)
  zero:
    // last input was a zero
    begin
      if (in) next_state = one1;
      else   next_state = zero;
    end
  one1:
    // we've seen one 1
    begin
      if (in) next_state = twols;
      else   next_state = zero;
    end
  twols:
    // we've seen at least 2 ones
    begin
      if (in) next_state = twols;
      else   next_state = zero;
    end
endcase
end
```

crucial to include
all signals that are
input to state determination



note that output
depends only on state



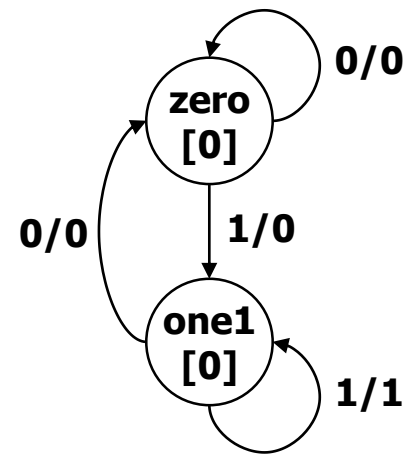
```
always @(state) begin

  case (state)
    zero: out = 0;
    one1: out = 0;
    twols: out = 1;
  endcase
end
endmodule
```

Mealy Verilog FSM

```
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  reg out;
  reg state; // state variables
  reg next_state;

  always @(posedge clk)begin
    if (reset) state = zero;
    else      state = next_state;
  end
  always @(in or state)
    case (state)
      zero:          // last input was a zero
        begin
          out = 0;
          if (in) next_state = one;
          else   next_state = zero;
        end
      one:           // we've seen one 1
        if (in) begin
          next_state = one; out = 1;
        end else begin
          next_state = zero; out = 0;
        end
    endcase
end
endmodule
```

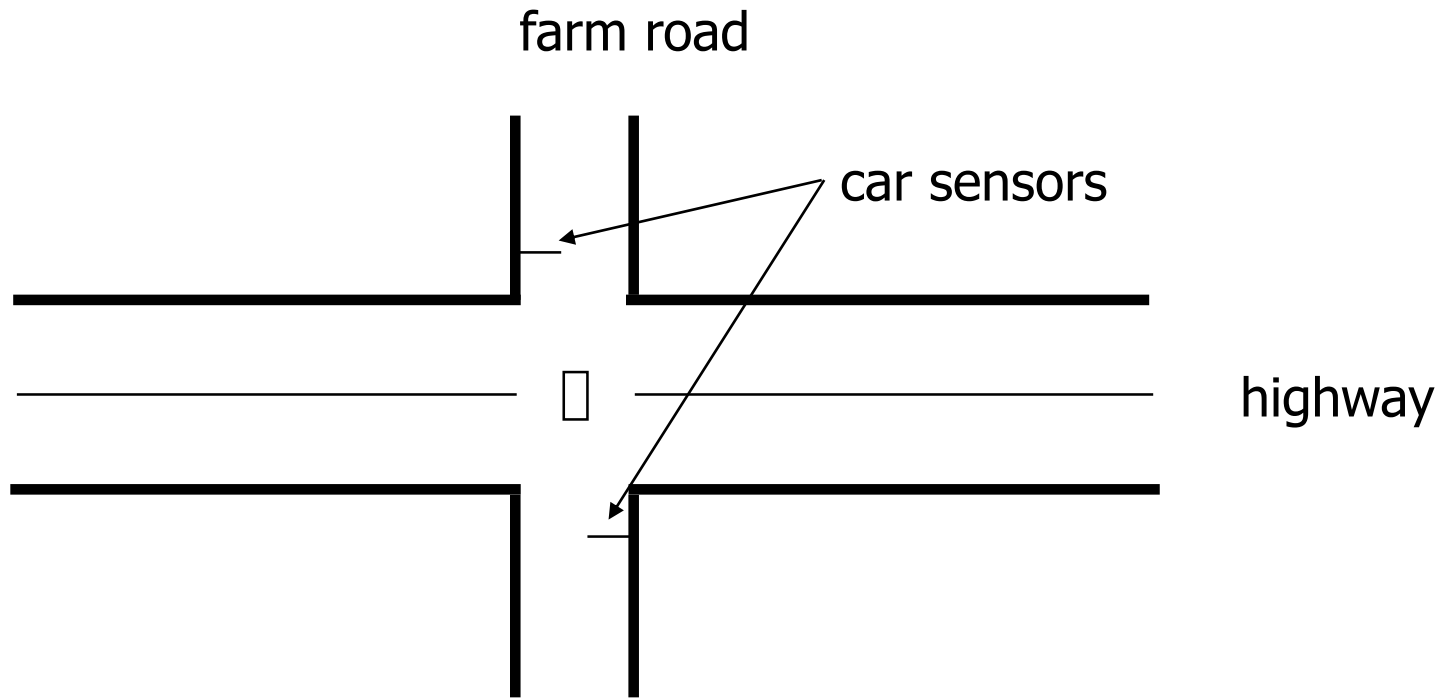


Mealy/Moore Summary

- Mealy machines tend to have fewer states
 - different outputs on arcs ($i \cdot n$) rather than states (n)
- Mealy machines react faster to inputs
 - react in same cycle – don't need to wait for clock
 - delay to output depends on arrival of input
- Moore machines are generally safer to use
 - outputs change at clock edge (always one cycle later)
 - in Mealy machines, input change can cause output change as soon as logic is done – a big problem when two machines are interconnected – asynchronous feedback

Example: Traffic light controller

- Highway/farm road intersection

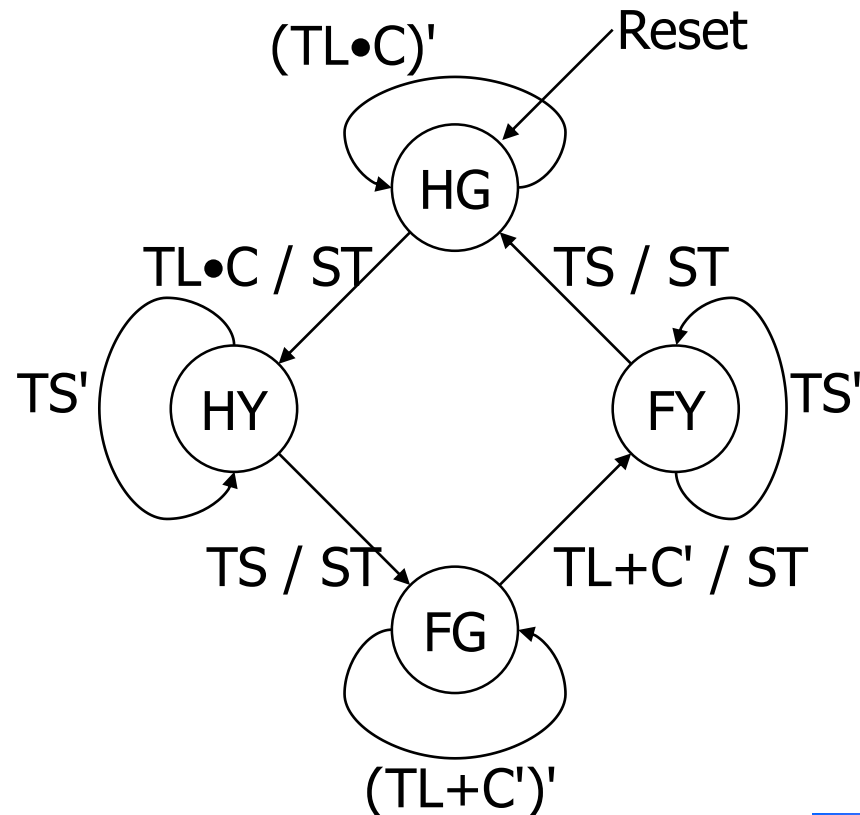


Traffic light controller (cont.)

- Detectors C sense the presence of cars waiting on the farm road
 - with no car on farm road, light remain green in highway direction
 - if vehicle on farm road, highway lights go from Green to Yellow to Red, allowing the farm road lights to become green
 - these stay green only as long as a farm road car is detected but never longer than a set interval; after the interval expires, farm lights transition from Green to Yellow to Red, allowing highway to return to green
 - even if farm road vehicles are waiting, highway gets at least a set interval of green
- Assume you have an interval timer that generates:
 - a short time pulse (TS) and
 - a long time pulse (TL),
 - in response to a set (ST) signal.
 - TS is to be used for timing yellow lights and TL for green lights

Traffic light controller (cont.)

- | inputs | description | outputs | description |
|--------|---------------------------------|------------|--|
| reset | place FSM in initial state | HG, HY, HR | assert green/yellow/red highway lights |
| C | detect vehicle on the farm road | FG, FY, FR | assert green/yellow/red highway lights |
| TS | short time interval expired | ST | start timing a short or long interval |
| TL | long time interval expired | | |
- | state | description |
|-------|--------------------------------|
| HG | highway green (farm road red) |
| HY | highway yellow (farm road red) |
| FG | farm road green (highway red) |
| FY | farm road yellow (highway red) |



Traffic light controller (cont.)

- Generate state table with symbolic states
- Consider state assignments

output encoding – similar problem to state assignment
(Green = 00, Yellow = 01, Red = 10)

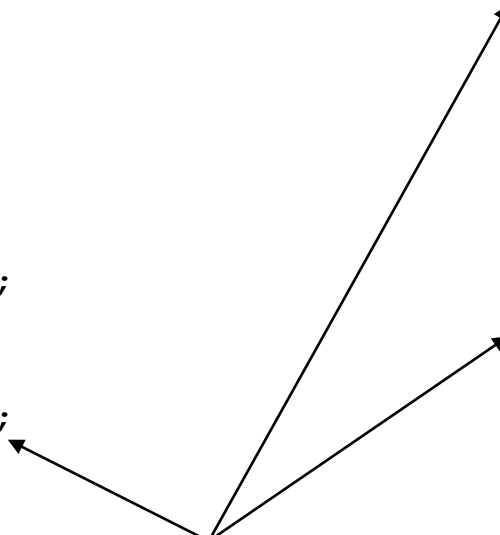
Inputs			Present State	Next State	Outputs		
C	TL	TS			ST	H	F
0	–	–	HG	HG	0	Green	Red
–	0	–	HG	HG	0	Green	Red
1	1	–	HG	HY	1	Green	Red
–	–	0	HY	HY	0	Yellow	Red
–	–	1	HY	FG	1	Yellow	Red
1	0	–	FG	FG	0	Red	Green
0	–	–	FG	FY	1	Red	Green
–	1	–	FG	FY	1	Red	Green
–	–	0	FY	FY	0	Red	Yellow
–	–	1	FY	HG	1	Red	Yellow

SA1:	HG = 00	HY = 01	FG = 11	FY = 10	
SA2:	HG = 00	HY = 10	FG = 01	FY = 11	
SA3:	HG = 0001	HY = 0010	FG = 0100	FY = 1000	(one-hot)

Traffic light controller FSM

- Specification of inputs, outputs, and state elements

```
module FSM(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, Clk);  
    output    HR;  
    output    HY;  
    output    HG;  
    output    FR;  
    output    FY;  
    output    FG;  
    output    ST;  
    input     TS;  
    input     TL;  
    input     C;  
    input     reset;  
    input     Clk;  
  
    reg [6:1] state;  
    reg      ST;  
  
    parameter highwaygreen  = 6'b001100;  
    parameter highwayyellow = 6'b010100;  
    parameter farmroadgreen  = 6'b100001;  
    parameter farmroadyellow = 6'b100010;  
  
    assign HR = state[6];  
    assign HY = state[5];  
    assign HG = state[4];  
    assign FR = state[3];  
    assign FY = state[2];  
    assign FG = state[1];
```



The diagram shows three arrows originating from the 'state' variable declaration 'reg [6:1] state;' and pointing to the assignment lines for HR, HY, and HG. This indicates that the state variable is used to determine the values of these outputs.

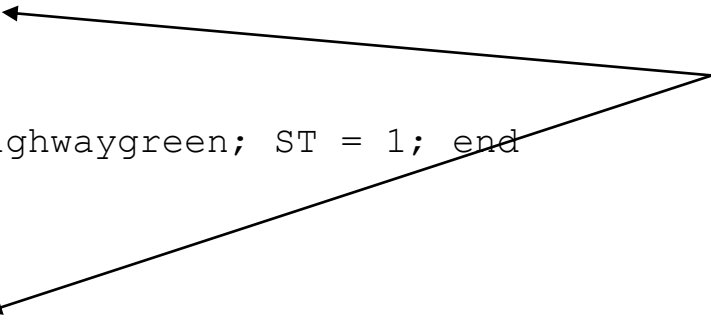
specify state bits and codes
for each state as well as
connections to outputs

Traffic light controller FSM

```
initial begin state = highwaygreen; ST = 0; end

always @(posedge Clk)
begin
    if (reset)
        begin state = highwaygreen; ST = 1; end
    else
        begin
            ST = 0;
            case (state)
                highwaygreen:
                    if (TL & C) begin state = highwayyellow; ST = 1; end
                highwayyellow:
                    if (TS) begin state = farmroadgreen; ST = 1; end
                farmroadgreen:
                    if (TL | !C) begin state = farmroadyellow; ST = 1; end
                farmroadyellow:
                    if (TS) begin state = highwaygreen; ST = 1; end
            endcase
        end
    end
end
endmodule
```

case statement
triggerred by
clock edge



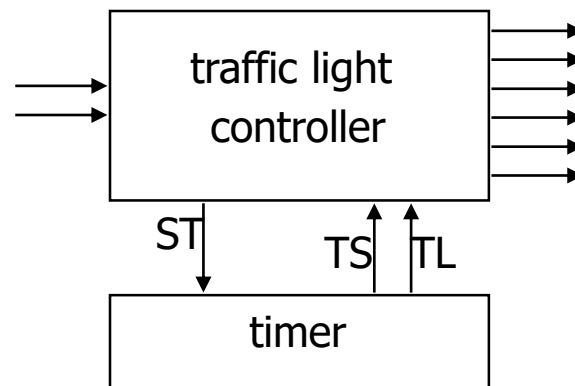
Timer FSM for traffic light controller

```
module Timer(TS, TL, ST, Clk);  
    output TS;  
    output TL;  
    input    ST;  
    input    Clk;  
    integer  value;  
  
    assign TS = (value >= 4); // 5 cycles after reset  
    assign TL = (value >= 14); // 15 cycles after reset  
  
    always @(posedge ST) value = 0; // async reset  
  
    always @(posedge Clk) value = value + 1;  
  
endmodule
```

Complete traffic light controller

- Tying it all together (FSM + timer) with structural Verilog (same as a schematic drawing)

```
module main(HR, HY, HG, FR, FY, FG, reset, C, Clk);  
  output HR, HY, HG, FR, FY, FG;  
  input  reset, C, Clk;  
  
  Timer part1(TS, TL, ST, Clk);  
  FSM   part2(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, Clk);  
endmodule
```



Finite state machines summary



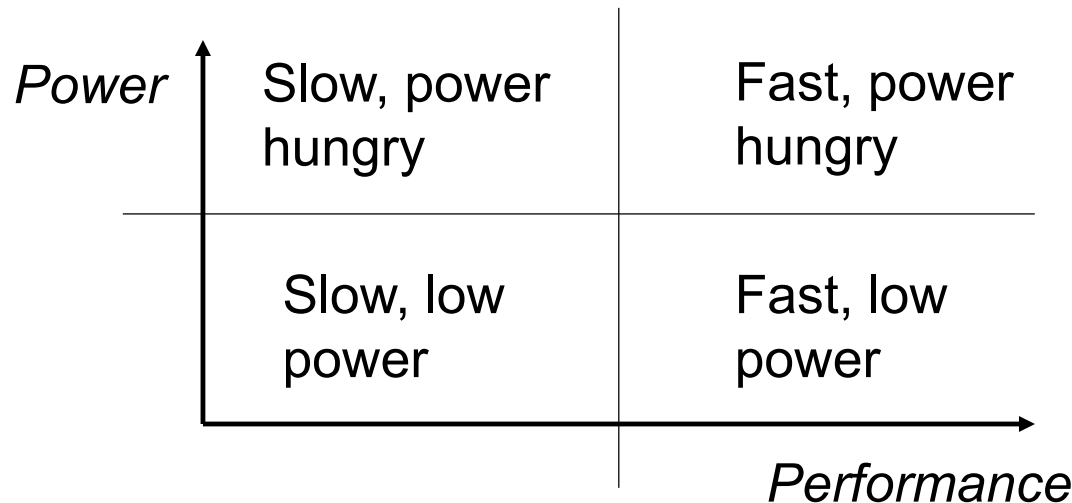
- Models for representing sequential circuits
 - abstraction of sequential elements
 - finite state machines and their state diagrams
 - inputs/outputs
 - Mealy, Moore, and synchronous Mealy machines
- Finite state machine design procedure
 - deriving state diagram
 - deriving state transition table
 - determining next state and output functions
 - implementing combinational logic
- Hardware description languages
 - Use good coding style
 - Communicating FSMs



Design space of digital circuits

When designing circuits, we want to achieve a desired functionality while looking for tradeoffs between the following:

- **Performance** (e.g. timing, delay, clock frequency)
- **Power consumption**



Power and performance are closely related. In general, you cannot decrease one without increasing the other

Your design might have a number of additional constraints:

- Area
- Accuracy

What is power?

In physics: Power is the rate of doing work (i.e. the rate of consuming Energy)

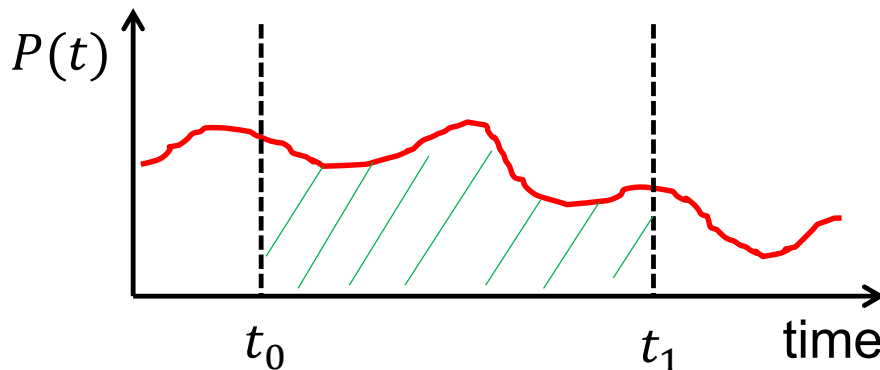
$$P = \frac{E}{t}$$

Units of measure:

- Power: *Watt*
- Energy: *Joule*

$$1 \text{ Watt} = 1 \text{ Joule} / 1 \text{ second}$$

Power is a function of time, energy is not!



Energy consumed in a time interval $[t_0, t_1]$:

$$E = \int_{t_0}^{t_1} P(t) dt$$

Power consumption of circuits

- The definition of “work done per unit time” is still valid
- We need to investigate more into details what the “work done” is in electrical circuits

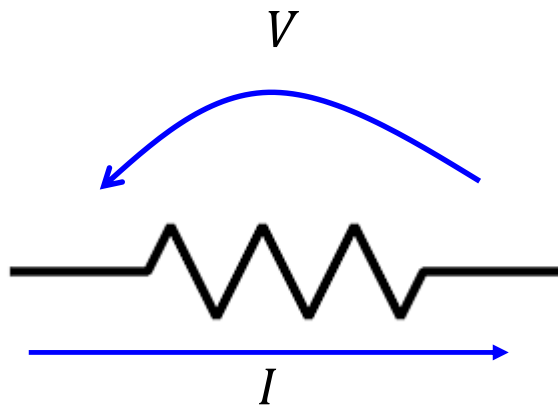
$$\text{Work done} = E = V Q$$

V = voltage

Q = charge

$$P = \text{work done per unit time} = \frac{E}{t} = \frac{VQ}{t} = V I$$

Example: Resistor



Conservation of energy:
energy cannot be created or
destroyed, but can be altered from
one form to another

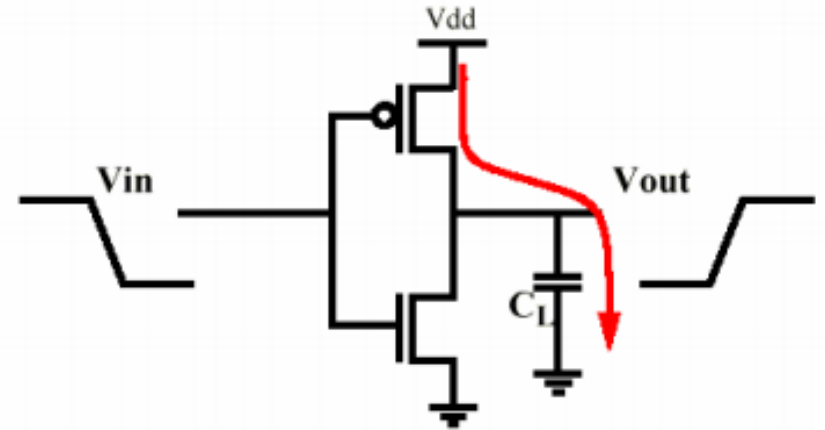
Electrical energy dissipated on a
resistor turns into heat

Example: CMOS inverter

There is power consumed every time there is a current flowing (I) subject to a difference of electric potential (V).

Remember:

- Transistors have an intrinsic resistance
- We model the output connection of gates with a “load capacitance”



When is that the inverter is consuming electric power?

- When the output is changing its values (and transistors are switching)
- Also, when transistor are OFF, they are still “leaking” some current

Where is this power going to?

- Dissipated as heat
- Spent for “charging” the load capacitor

Power consumption

- Power dissipation in CMOS circuits comes from two components:
 - **Dynamic Power**
 - Takes place when transistors are switching
 - Charging and discharging (switching) of the load capacitance
 - “Short-Circuit” current while both pMOS and nMOS networks are partially ON
 - **Static Power**
 - Given by “leakage currents”
 - Subthreshold conduction
 - Tunneling current
 - Leakage through reverse biased diodes

Dynamic power

Dynamic power can be modeled by a relatively simple mathematical model:

$$P_{dynamic} = A C V^2 f$$

V : Operating voltage of the circuit

f : Operating frequency (i.e. clock) of the circuit

C : Capacitance

- Equivalent capacitance of the circuit
- Once the circuit is built, this is a fixed property of the circuit
- It is a function of number and dimension of wires and transistors

A : Activity factor

- It is a term that accounts for “how much” the transistors are switching
- It is a property of the “workload” of the circuit (for example, the application you are executing on your computer)

Static power

Static power can be expressed by the product of voltage times leakage current:

$$P_{static} = V I_{leakage}$$

- The leakage current $I_{leakage}$ is a rather complicated term, which is itself the sum of different contributions (depending on the physical origin of the leak).
 - Subthreshold leakage
 - Gate leakage
 - Junction leakage
 - Contention current
- Such contributions have much more complicated equations, which depend on many technological and physical parameters of transistors

Problems related to power consumption

- Data centers:
 - Electricity bill \$\$\$



- Mobile devices:
 - Battery



- Common problem: Higher temperature
 - Temperature increases linearly with power.
 - Data centers: fans, cooling systems, AC → even higher electricity bill !
 - Mobiles: Overheating, discomfort for the user, risk of damaging the device.
 - Higher temperature → higher static power consumption!

How to reduce dynamic power consumption?

Dynamic power reduction:

$$P_{dynamic} = A C V^2 f$$

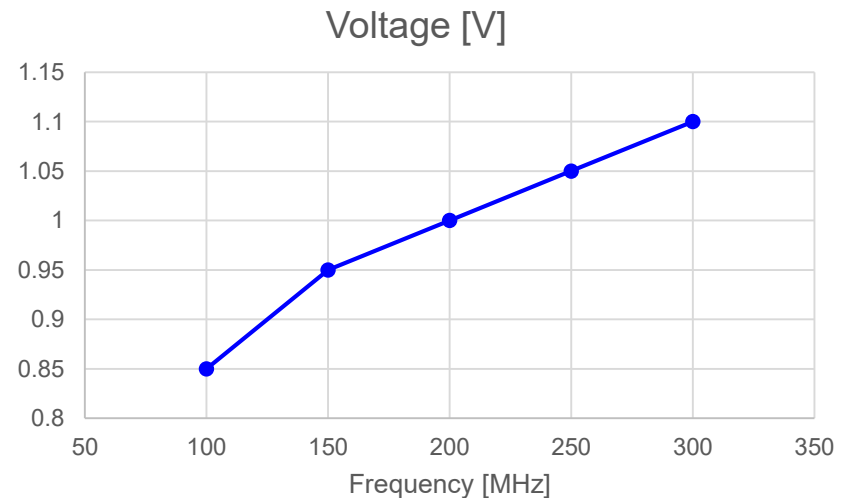
- **Decrease activity factor**
 - Selective clock gating
 - Drawback: if the system transitions rapidly from an idle mode to a fully active mode a large di/dt spike will occur
- **Decrease switching capacitance**
 - Small transistors
 - Careful floor planning to reduce interconnect
- **Decrease power supply**
 - Adjust voltage depending on the operating mode
- **Decrease operating frequency**
 - Modern OS and processors support Dynamic Voltage Frequency Scaling (DVFS)

Example 1: GPU, power and FPS

Your operating system can control the operating frequency and voltage of your GPU while playing 3D games. This would also impact the quality of the game, referred to as Frames per Second (FPS). For the game to be playable, the FPS should be at least 60.

Assume that FPS increases linearly with frequency: $FPS = b * f$
Where $b = 0.5$

Assume the GPU has a range of frequency [100 300] *Mhz*, and can switch only between fixed Voltage-frequency pairs



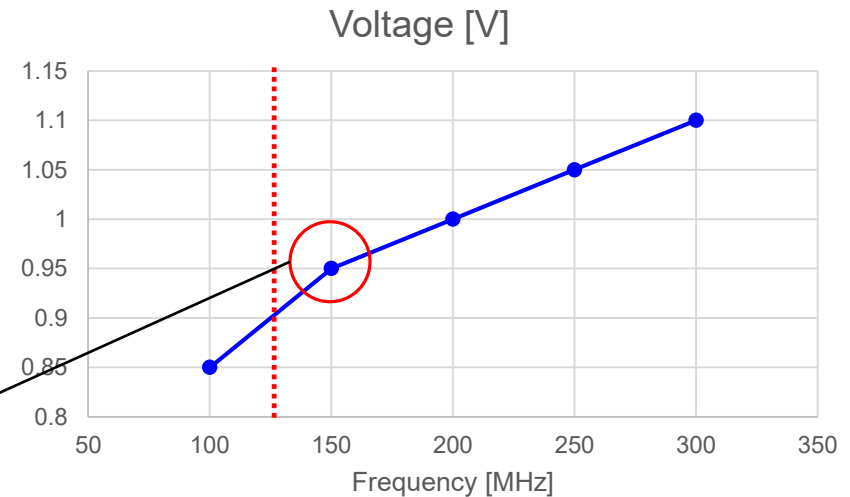
Example 1: GPU, power and FPS

$$FPS_{target} = b * f_{target}$$

$$f_{target} = \frac{FPS_{target}}{b} = \frac{60}{0.5} = 120MHz$$

$$f_{selected} = 150MHz$$
$$V_{selected} = 0.95V$$

$$FPS = b * f_{selected} = 75 > FPS_{target}$$



Assuming that $A = 0.8$, $C = 120pF$, and that the static power is constant and equal to $5W$, calculate the total power consumption

$$P_{dynamic} = ACV^2f = 0.8 * 120 * 10^{-9} * (0.95)^2 * 150 * 10^6 \cong 13W$$

$$P_{total} = P_{dynamic} + P_{static} = 13W + 5W = 18W$$

Example 2: Smartphone under the sun

If your phone is under the sun, the temperature of the processor is 70 C.
When it is under the shade, the temperature is 40 C.
Assume that the static power is described by:

$$P_{static} = a e^{bT}$$

Where $a = 1[W]$ and $b = \frac{1}{50} \left[\frac{1}{C} \right]$

Assuming that the battery has 2000J of residual capacity, how long do you increase the battery lifetime by keeping it on the shade? (assume that the dynamic power is zero and that the power consumption of other components is negligible)

$$t(40\text{ C}) = \frac{E}{P_{static}(40\text{ C})} = \frac{2000J}{2.22W} \cong 900\text{ s}$$
$$t(70\text{ C}) = \frac{E}{P_{static}(70\text{ C})} \cong 500s$$