EECS 373
Introduction to Embedded System Design

Robert Dick
University of Michigan

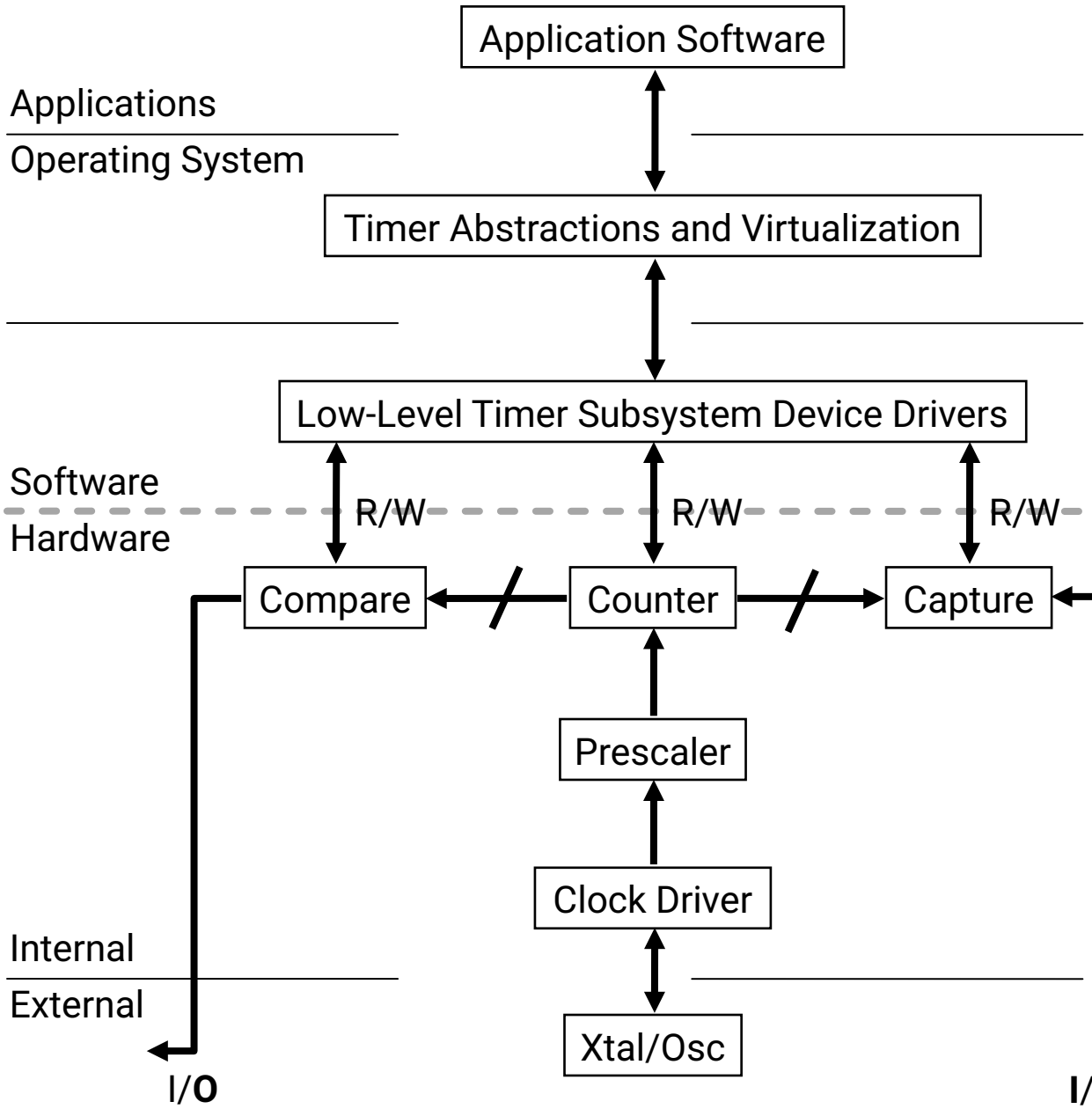
Lecture 8: Hazards, Setup and Hold

13 February 2023

Review

- Timers
  - Uses.
  - HW/SW implementation.
- Subpreemption priorities

Anatomy of a timer system




```
...
timer_t timerX;
initTimer();
...
startTimerOneShot(timerX, 1024);
...
stopTimer(timerX);
```

```
typedef struct timer {
  timer_handler_t handler;
  uint32_t time;
  uint8_t mode;
  timer_t* next_timer;
} timer_t;
```

```
timer_tick:
  ldr r0, count;
  add r0, r0, #1
  ...
```

```
module timer(clr, ena, clk, alrm);
  input clr, ena, clk;
  output alrm;
  reg alrm;
  reg [3:0] count;

  always @(posedge clk) begin
    alrm <= 0;
    if (clr) count <= 0;
    else count <= count+1;
  end
endmodule
```

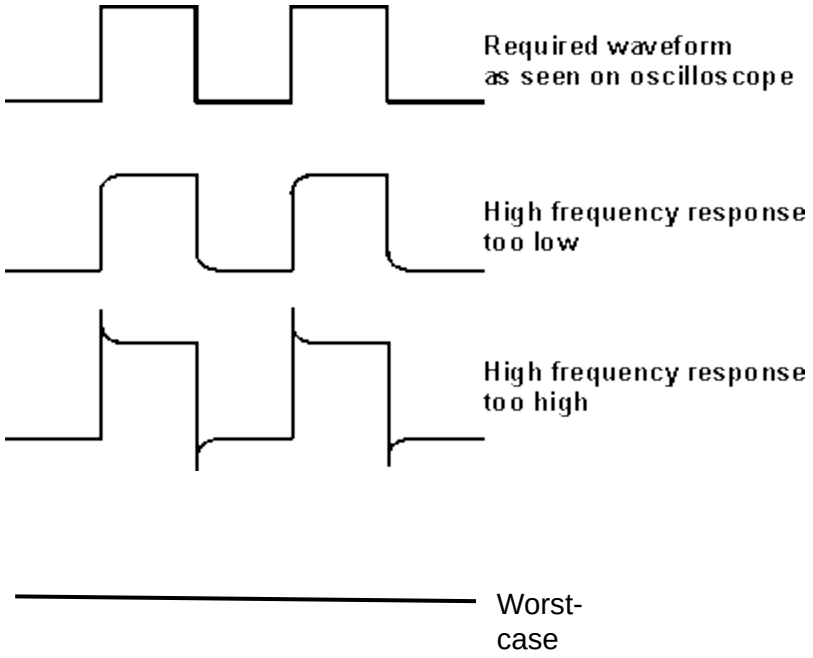
Position in course

- Introduction
- Arch, assem, ABI, debugger
- Project, MMIO
- Assem, MMIO, APB
- APB
- Interrupts
- Timers (You Are Here)
- Serial
- ADCs, DACs, datasheets
- Analog and PCB
- Power
- Filters and amps
- **Project**

Main topics covered before midterm exam

- Embedded system definition and market
- Technology trends
- Embedded applications
- ARM architecture, assembly, and ABI
- MMIO
- Debugging
- APB
- Build process
- Aspects of ANSI C related to embedded systems
- Interrupts
- Timers

Lab/project comment: compensated probes



Outline

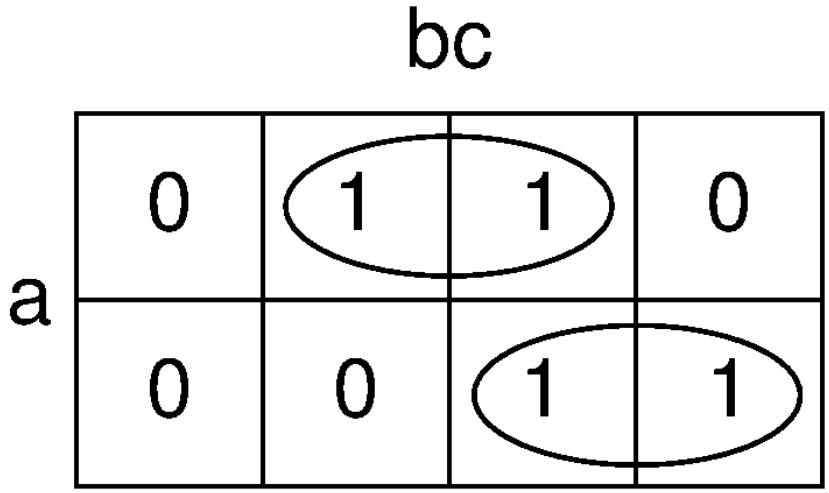
- **Hazards**
- Setup and hold times

Hazards

- Race between variable transitions.
- May, not must, produce a glitch.
- Glitch
  - Static glitch: transient pulse of incorrect value when output should be stable.
  - Dynamic glitch: transient pulse of incorrect value when output should be changing.
- Consider a minimal implementation of
  - f(a, b, c) = a'b'c + a'bc + abc + abc'

Hazards

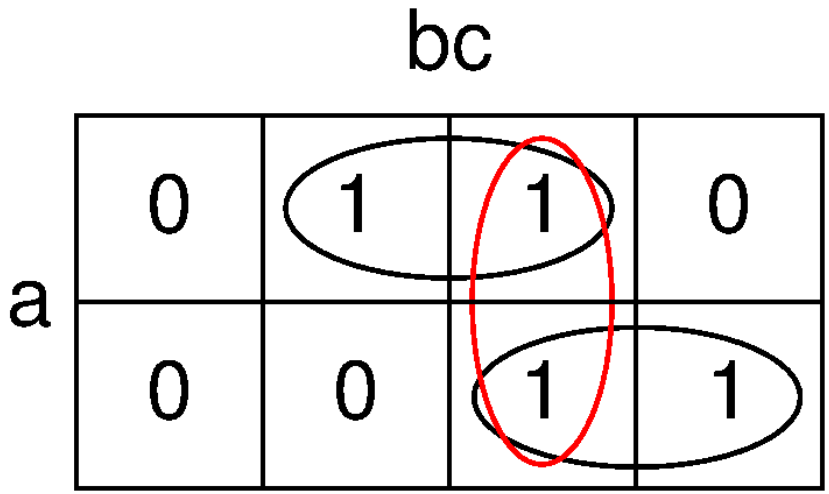
- Consider a minimal implementation of
  - f(a, b, c) = a'b'c + a'bc + abc + abc'



- f(a, b, c) = a'c + ab
- What if b=1, c=1?

Hazards

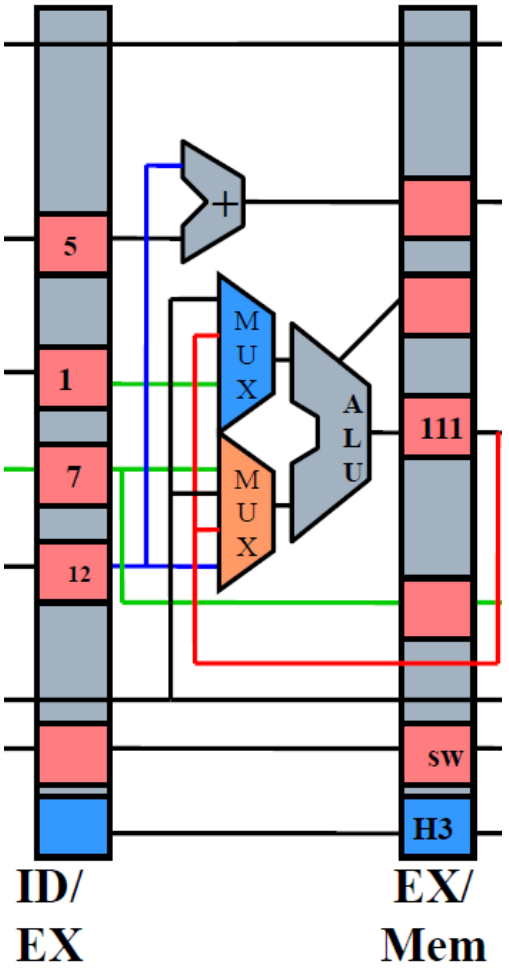
- How to eliminate
  - Limit logic to two levels
  - Cover all transitions



- f(a, b, c) = a'c + ab + bc
- What if b=1, c=1?

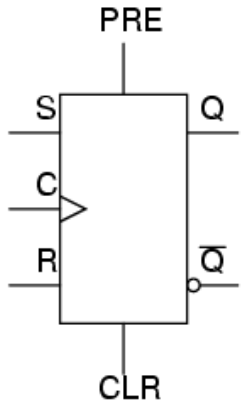
Effect of hazards

- Can sometimes ignore in synchronous systems.
- Only sampling on clock edges.
- Make clocks slow enough so glitching done before next edge.
- Wastes energy.
- Causes major problems in asynchronous systems.
- Different design style required.

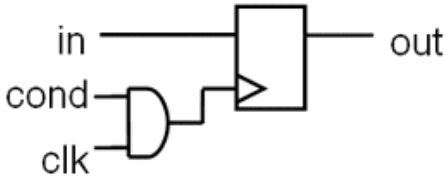


When hazards need special attention

- Asynchronous resets
  - No glitching on (p)reset.
  - Could use flip-flop on the input.
  - Instead, use synchronous reset.
- Clocks
  - Hazards can produce spurious edges.



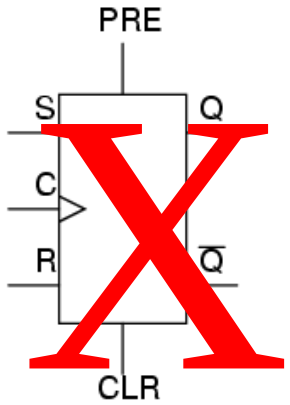
Traditionally, CLR is used to indicate async reset. "R" or "reset" for sync. reset.

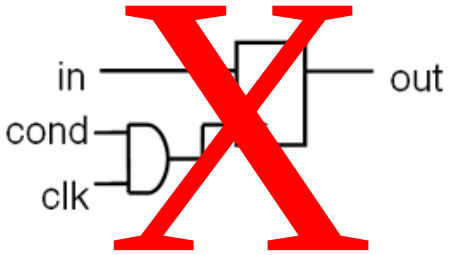


If clk is high and cond glitches, you get extra edges!

Simple design rules

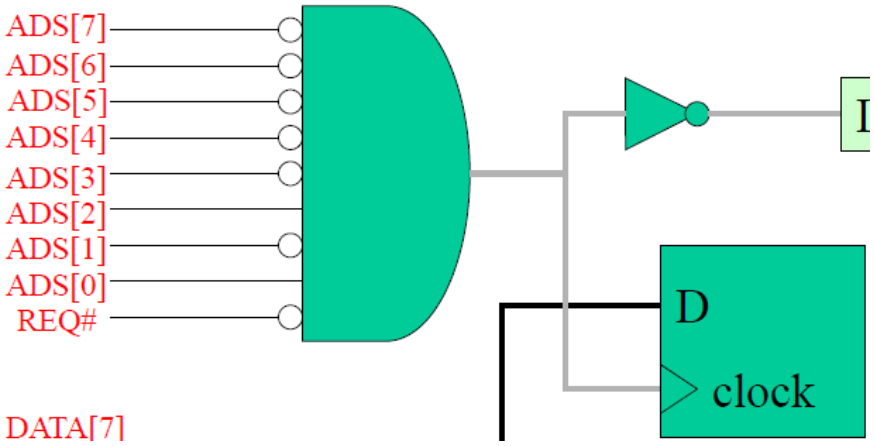
- Understand implications of asynchronous resets.
- Don't drive clock with hazardous logic.
- Hazard-free guarantee.
  - Only two levels.
  - Cover transitions.
  - Literal or complement, not both.
  - Example.





Glitches

- Async clock used in our bus example.
- Safe: REQ drops after glitches done.
- Might be safe in other circumstances.
- Need detailed analysis to know.

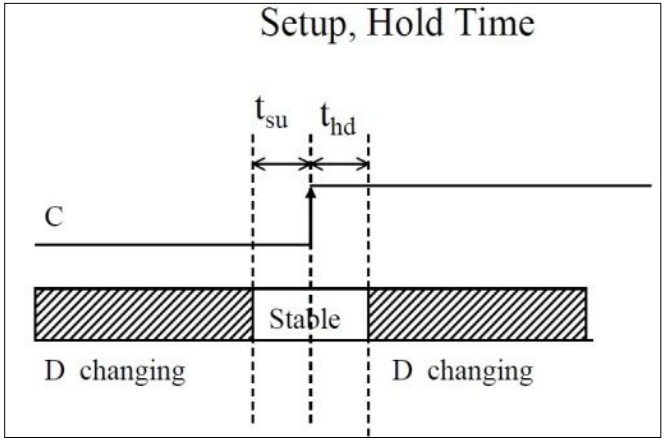


Outline

- Hazards
- **Setup and hold times**

Setup and hold time

- What if clock and data change at same time?
- Data latched is unclear.
- Often worse for registers than single flip-flops.
  - Inconsistent state.
- Use temporal guard band around clock edge.
- Setup time.
- Hold time.



So what happens if we violate set-up or hold time?

- Often, get one of the two values.
  - Consider getting a button press from the user.
  - Fine in this case.
- Can be harmful.
  - Flip-flop may not settle to a “0” or a “1” quickly.
  - Could cause setup time violations for later gates.
  - Different fanout gates may see different outputs.
  - May see mix of old and new values on different bits.

Example

- A common thing to do is reset a state machine using a button.
  - User can “reset” the system.
- Assume setup/hold time violation.
- State machine bits reset in different cycles.

Methods of synchronizing

- Dealing asynchronous inputs complex.
  - Can violate setup/hold times if not done properly.
- Can synchronize with circuit.
  - First flip-flop might have problems.
  - Second should be fine.
- Or do it right: explicitly design an AFSM.
- I have been criticized by the department chair (at another univeristy) for teaching AFSM design to undergrads, but may do so if there is interest.

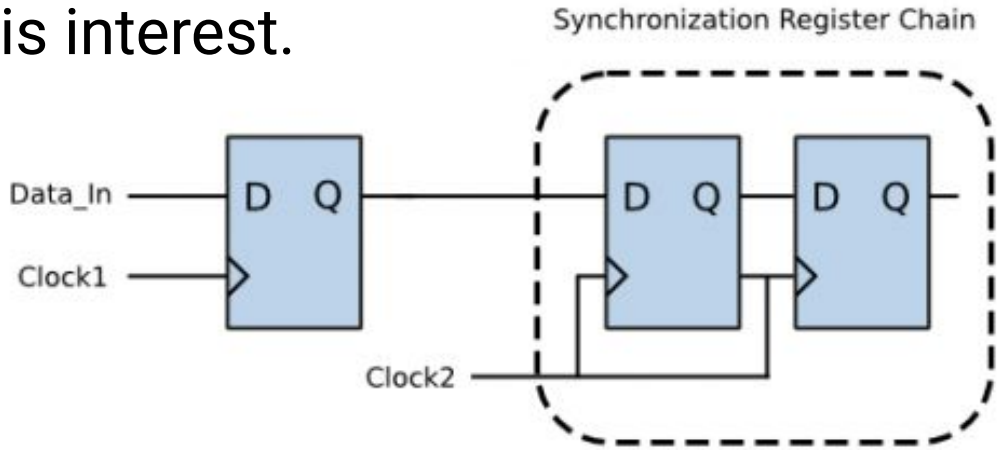
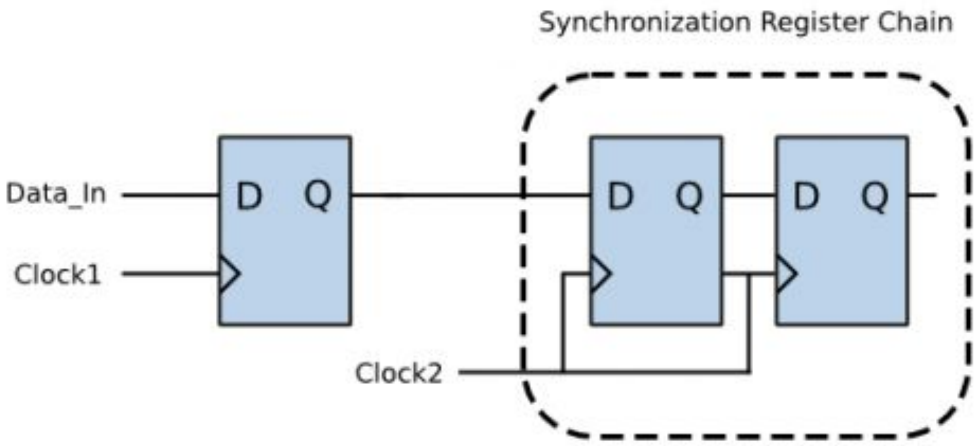


Figure from http://www.eeweb.com/electronics-quiz/solving-metastability-design-issues, we use the same thing to deal with external inputs too!

Example synchronization circuit




```
/* Synchonization of Asynchronous switch input */

always@(posedge clk)
begin
 sw0_pulse[0] <= sw_port[0];
 sw0_pulse[1] <= sw0_pulse[0];
 sw0_pulse[2] <= sw0_pulse[1];
end
```

```
always @(posedge clk) SSELr <= {SSELr[1:0], SSEL};
```

Example synchronization circuit

- Embedded system definition and market
- Technology trends
- Embedded applications
- ARM architecture, assembly, and ABI
- MMIO
- Debugging
- APB
- Build process
- Aspects of ANSI C related to embedded systems
- Interrupts
- Timers

Done.