Discussion

Fri Nov 20
outline

● **RTL**
  ○ HLSM
  ○ DATA PATH
  ○ HLSM -> FSM
  ○ Connecting datapath & control path

● **TIMING**
  ○ RTL timing
  ○ common mistakes

● **FSM (sequence detector)**
  ○ state diagram
  ○ excitation table
  ○ Implementation
RTL DESIGN

Design a system that outputs the average of the most recent two data input samples,

- input: I 8bit unsigned
- input: S 1bit, sample the input when S goes from 0 to 1.
- output: avg 8bit unsigned

what are the difficulties?

➢ capture the edge of S: need to wait when it stays 1 or 0 (need two wait srate)
➢ Distinguish the initial state
➢ How to get two samples from 1 input lines?! REGISTERS!
STEP 1: HLSM

Ireg: saves the last value
Prevreg: saves the previous value
avgreg: holds the result
init: initial to zero, we should wait for input S to become zero.
wait: we wait here until we see rising edge on S
Sample: rising edge detected, shift the registers, calculate the average
waitlow: wait till S falls.
STEP 2: DATA PATH

- Shift registers to hold the two most recent data
- output of the adder is 9 bit to take care of overflow
- Shift to left instead of division
- ld & clr are coming from the controller,
- we can use the same ld&clr for all registers in this question
- Dont forget the output register!
STEP3: connecting datapath & control path

Be careful about the directions!

Load & Clear signals for all the registers are going to the Datapath unit from the controller.
STEP 4: CONTROLLER FSM

- get rid of any arithmetic operation which exist in the HLSM,
- registers’ load and signals should be defined as the output of the FSM
- for simplicity define the output only when it is equal to one
what is the longest path?

we need to look at the path between each pair of registers:

from Ireg to avgreg = adder + shifter

from prevreg to avgreg = adder + shifter

\[ T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew} \]

longest (tpd)= T-adder + T-shifter

from Ireg to Prevreg = zero combinational delay-> shortest (Tcd)

you may need to calculate delay of the components!
TIMING

Let's look at an abstract circuit,
The rectangles are some combinational logic.
what is the longest path?
we need to look at each fully combinational path from output of FFi to input of FFj.
fully combinational means there should not be any FF in the middle!
from FF1 to FF2: path A
from FF2 to FF1: path C

There is NO fully combinational path from A+C.

there is no fully combinational path from output of FF1 back to itself.

from FF1 to FF3: path (A+B)

from FF4 to FF4: path D

YES, you should care about the feedback loop from a FF to itself!

If two FFs are directly connected to each other, that path will be the shortest path.
TIMING

\[ T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew} \quad t_{ccq} + t_{cd} > t_{hold} + t_{skew} \]

➢ find all the path (previous slides)
➢ find the longest one and the shortest one based on the given numerical values
➢ in these formula T_setup, T_hold, T_ccq, T_pcq corresponds to a single FF. You should NOT multiply it by the number of the FFs. The reason is that we are looking at each path separately (look back at the slides to see how the formula was derived) common mistake in your midterm 2 :(
FSM, SEQUENCE DETECTOR

Design a FSM which detects pattern 0011, example:

input: 10011010011
output: 00001000001

First, let's figure out what states we need:

➢ detect the first zero “S0”
➢ detect the second (or more zero) “S1”
➢ detect one 1 after the two zeros “S2”
➢ detect the second one “S3”
S0: wait till we get the first 0,
S1: we need another 0 to detect 00:
    if 0 is received go to S2, 00 detected
    if 1 received, we need to reset
S2: wait to get 1
S3: if zero received go to S1, if 1 received
    pattern is detected
POTENTIAL MISTAKES

- when 1 is received at S3, there is no need for a new state, we can go back to initial state
- when 0 is received at S3, it's not like reset so we need to go to S1 not S0
- make sure to reuse the states, and don't add extra states.
Excitation table

<table>
<thead>
<tr>
<th>CS</th>
<th>inp</th>
<th>NS</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>00</td>
<td>1</td>
</tr>
</tbody>
</table>
IMPLEMENTATION

D1 = \text{in}'Q1'Q0 + Q1Q0'

\begin{tabular}{c|cccc}
\text{in\backslash Q1Q0} & 00 & 01 & 11 & 10 \\
\hline
0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 \\
\end{tabular}

D0 = \text{in}'Q0'Q1 + \text{in}'Q0Q1 + \text{in}Q1Q0'

\begin{tabular}{c|cccc}
\text{in\backslash Q1Q0} & 00 & 01 & 11 & 10 \\
\hline
0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 \\
\end{tabular}

out = Q1Q0in

\textbf{COMMON MISTAKES}

- DONT forget the FFS!
- use K-maps/dont cares to simplify.
- just implement what the question is asking for : MSB bit, LSB bit, output? DONT implement the whole circuit if you are not asked to do so!