PHASE2

Project name	EE 477 Final Project		
Document ref			
Version	1.0		
Release date	20220416		
Author	Fei Wu, Haoda Wang, Jason Yik		
Classification			
Distribution List	Fei Wu2.2, Haoda Wang3, Jason Yik2.1		
Approved by	Name	Signature	Date

Ming Hsieh Department of Electrical Engineering
University of Southern California, Los Angeles, CA 90089
Spring 2022
EE 477

Revision History

Date	Version	List of changes	Author + Signature

C A T A L O G

1 MOTIVATION 4

2 FSM 5

2.1 FSM DESIGN 5
2.2 IMPLEMENTATION 6
2.2.1 DFF 6
2.2.2 Arbiter 6
2.2.3 FSM 8

3 MINIMUM TIME PERIOD 10

1 MOTIVATION

Arbiters are widely used in various fields and products such as network applications, and SoCs, and are responsible for time shared resource management for multiple requestors. There are several techniques that ensure this scheduling like priority-based and round robin. Here we intend to implement round robin scheduling in our design.

2 FSM

The key idea is to output the value of multiplier (8bits) or Divisor (4bits of quotient concatenated with 4 bits of remainder) designed in phase 1 depending on the arbiter (grant) logic value. The grant for output is based on round robin scheduling. There are two blocks the multiplier block, and the divider block that could request access to an output link (8 bits) to send their information. The following steps will help you design the arbiter state machine. Design a FSM based on the following specifications:

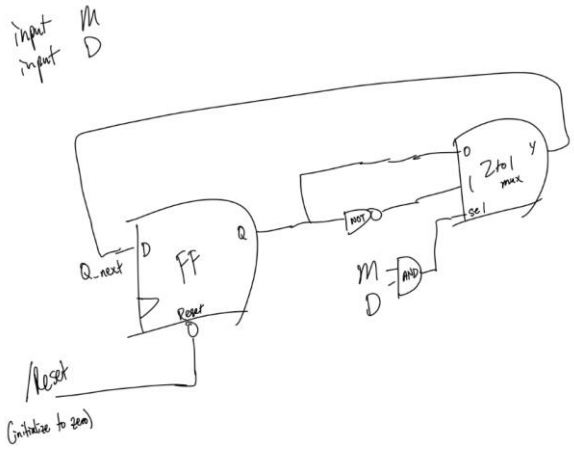
1. If only one of the logic block (multiplier or divisor) asks for grant, the arbiter should grant that block access to output and hence let the requesting block pass its results to the output. E.g., if the multiplier block is the only one that requests access in this clock, then it should be granted access either the multiplication result or divisor result (based on who asked for access).

2. If none of the two blocks asks for access, The arbiter should specify no grant, by setting an output signal STALL to 1.

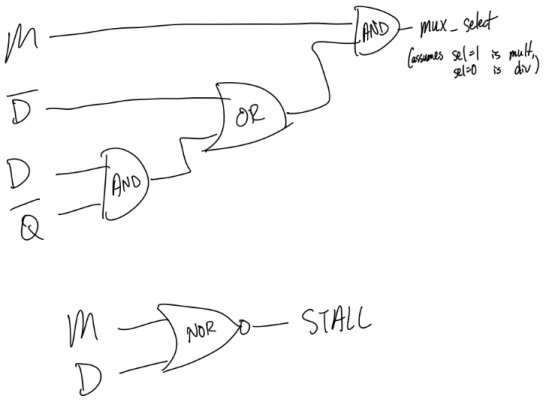
3. If both the blocks ask for access, the arbiter should check the last grant and switch that to the other. E.g., if the last grant was given to the multiplier, then now divider should be granted access. If none of the blocks was previously given access and now both ask for it, give access to multiplier.

2.1 FSM design

The FSM for the arbiter looks like the diagram, one bit of storage is needed to remember which block the last grant was given to. This bit only flips if both blocks are requesting at the same time, otherwise it is the same. So, the next-state logic looks like this:



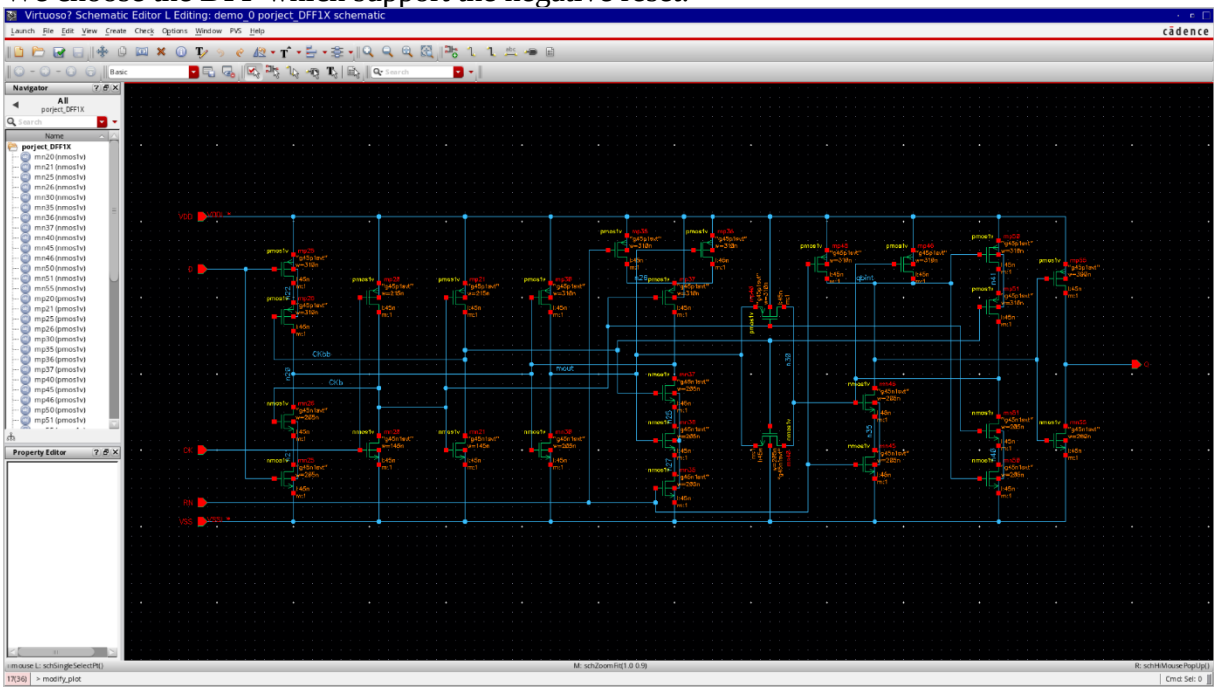
The output logic for select and stall refers to the inputs and the current state, making this a Mealy machine:



2.2 Implementation

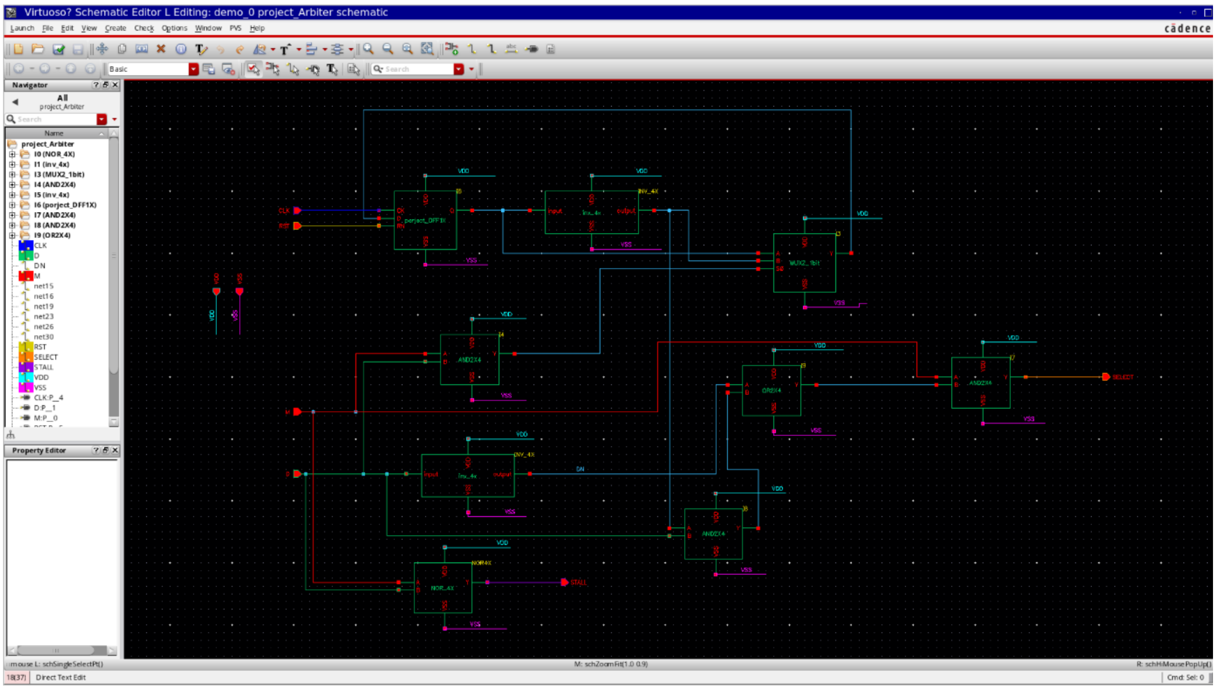
2.2.1 DFF

We choose the DFF which support the negative reset.



2.2.2 Arbiter

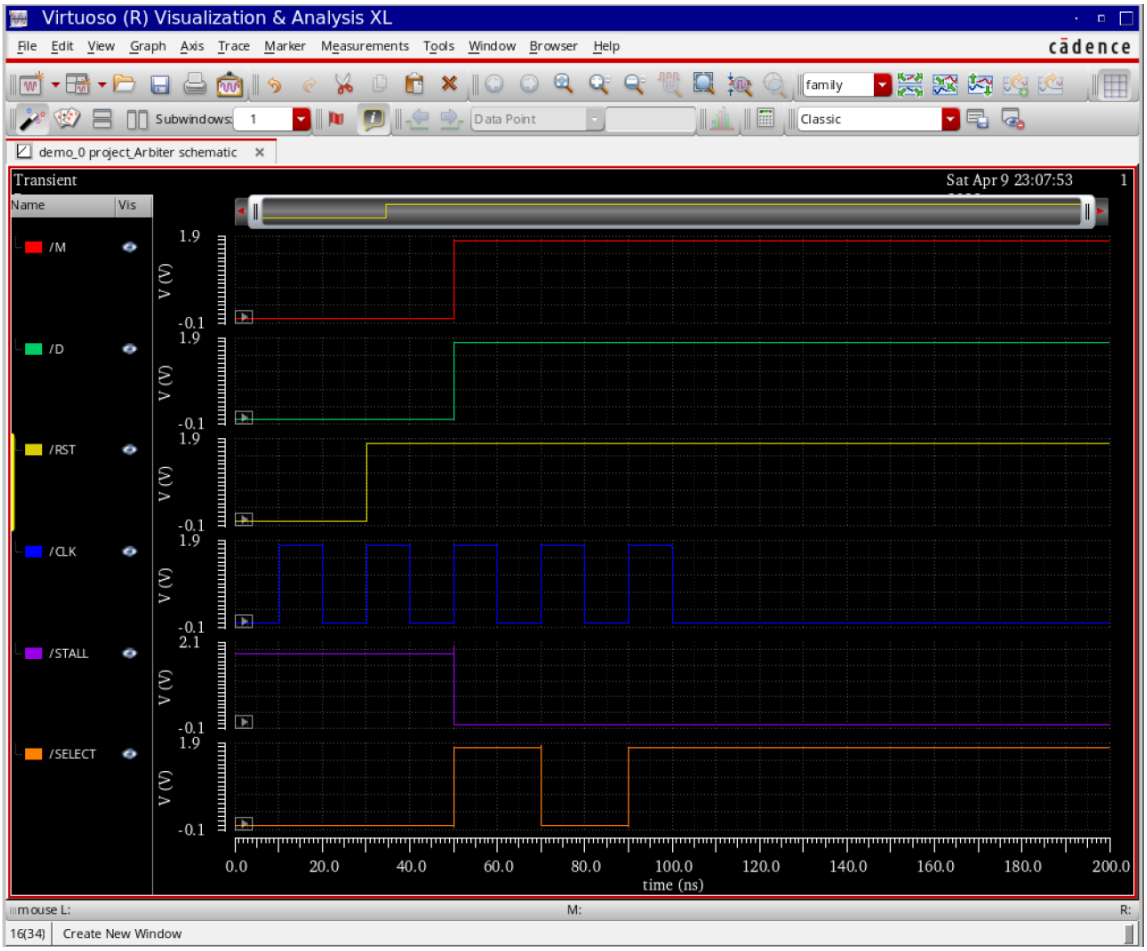
The arbiter is using to memorize the last grant and give the correct grant to the next both request situation.



Building a test wave to check the negative reset, signal request, both request, and another couple of times both request after that.

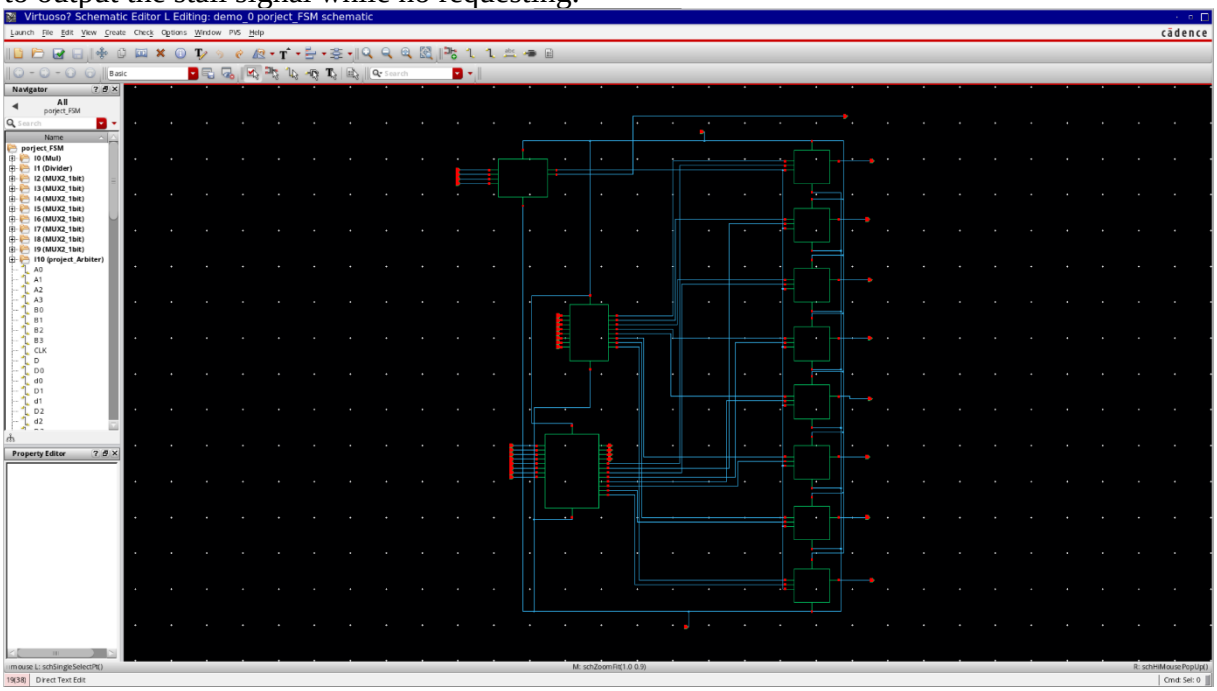
```
wufei@viterbi-scf2:project

radix 1    1    1    1
vname CLK  RST  M    D
io    i    i    i    i

tunit 10ns
vih 1.8
vil 0
voh 1.8
vol 0
trise 10p
tfall 10p

tdelay 0 0 0 0 0

;mul

0 0 0 0 0
1 1 0 0 0
2 0 0 0 0
3 1 1 0 0
4 0 1 0 0
5 1 1 1 1
6 0 1 1 1
7 1 1 1 1
8 0 1 1 1
9 1 1 1 1
10 0 1 1 1
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
                                          27,1          All
```

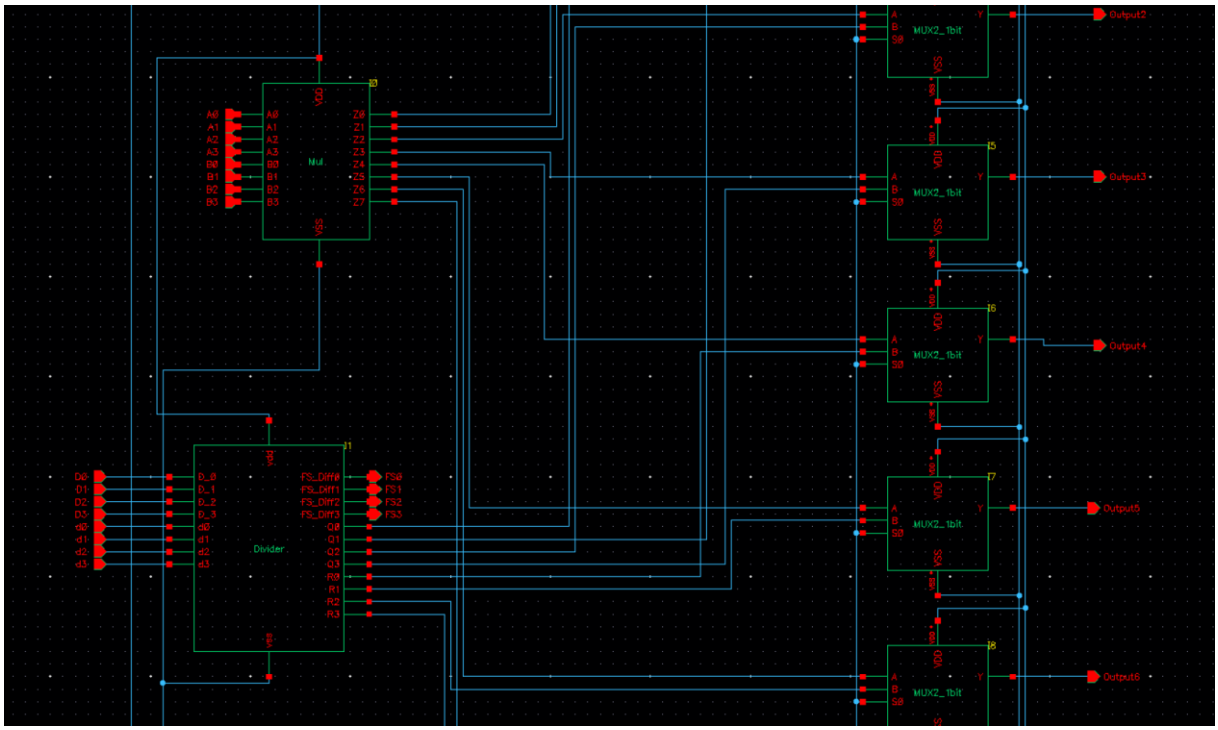
The waveform show the correct outcome of the arbitration.

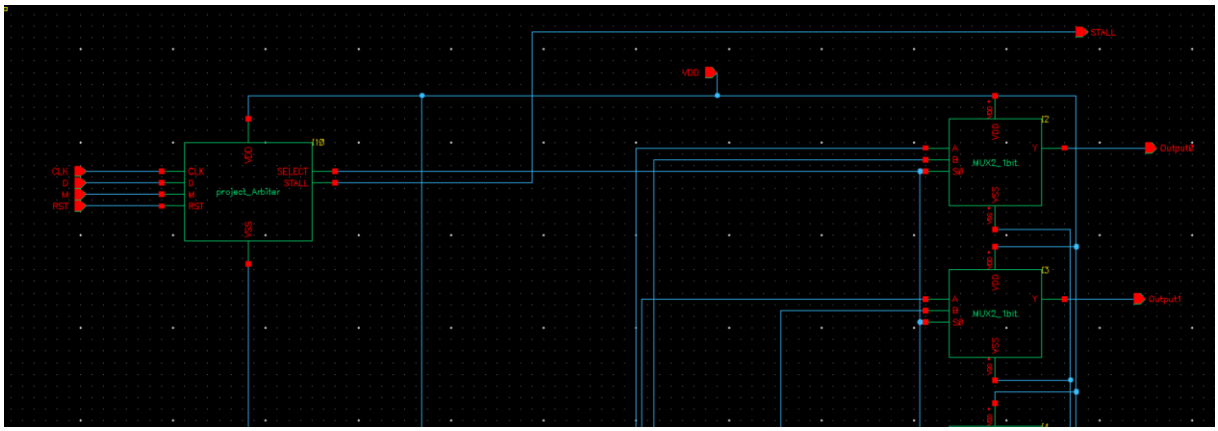


2.2.3 FSM

I connect the div and mul modules with the verified arbiter. Give the output to 8 2bits_mux and use the result of arbiter to decide which calculation unit will help the output. Also I use a wire to output the stall signal while no requesting.




Mul
Divider
MUX2_1bit
Output2
Output3
Output4
Output5
Output6

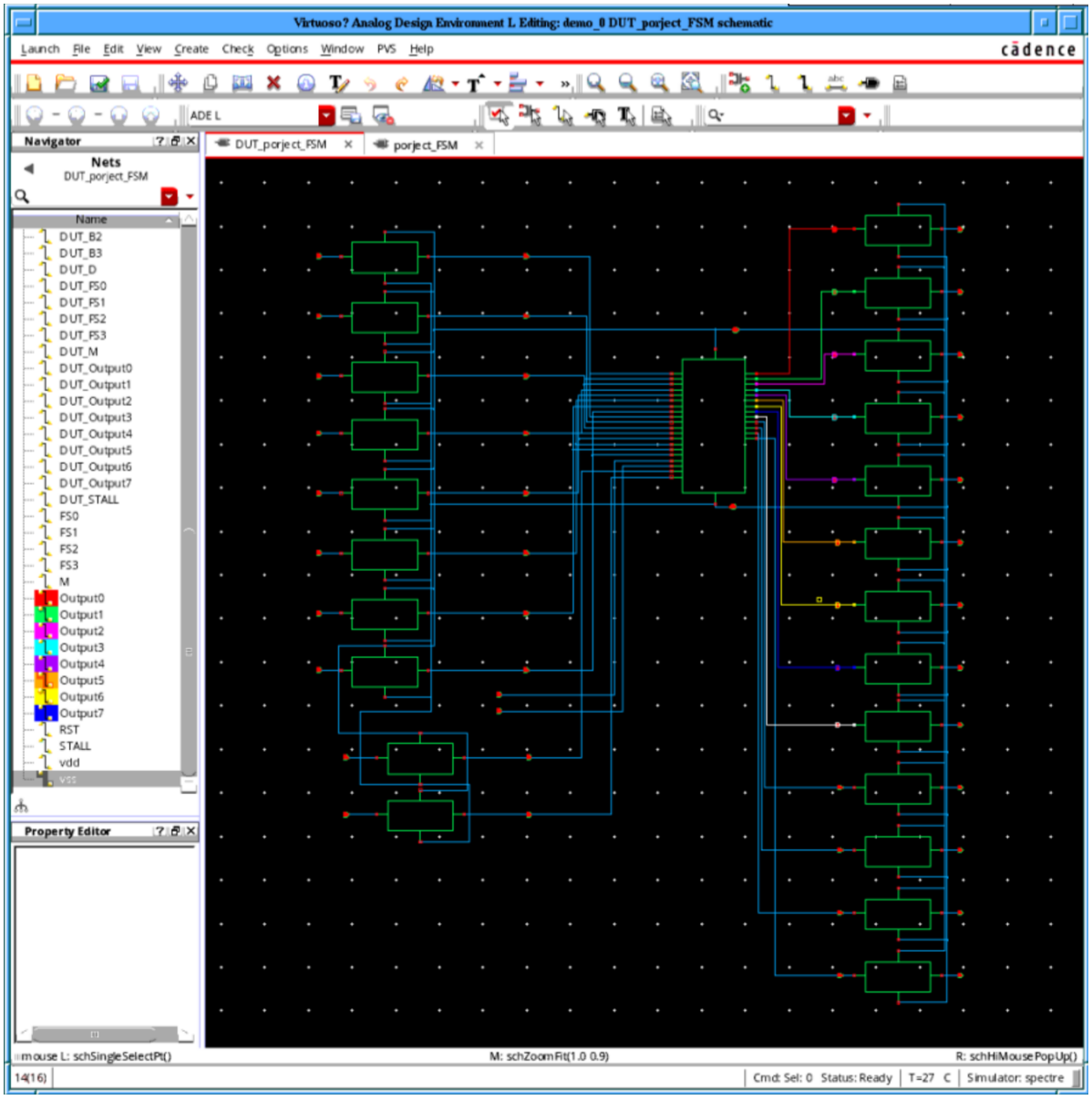

STALL
VDD
project_Arbiter
MUX2_1bit
Output0
Output1

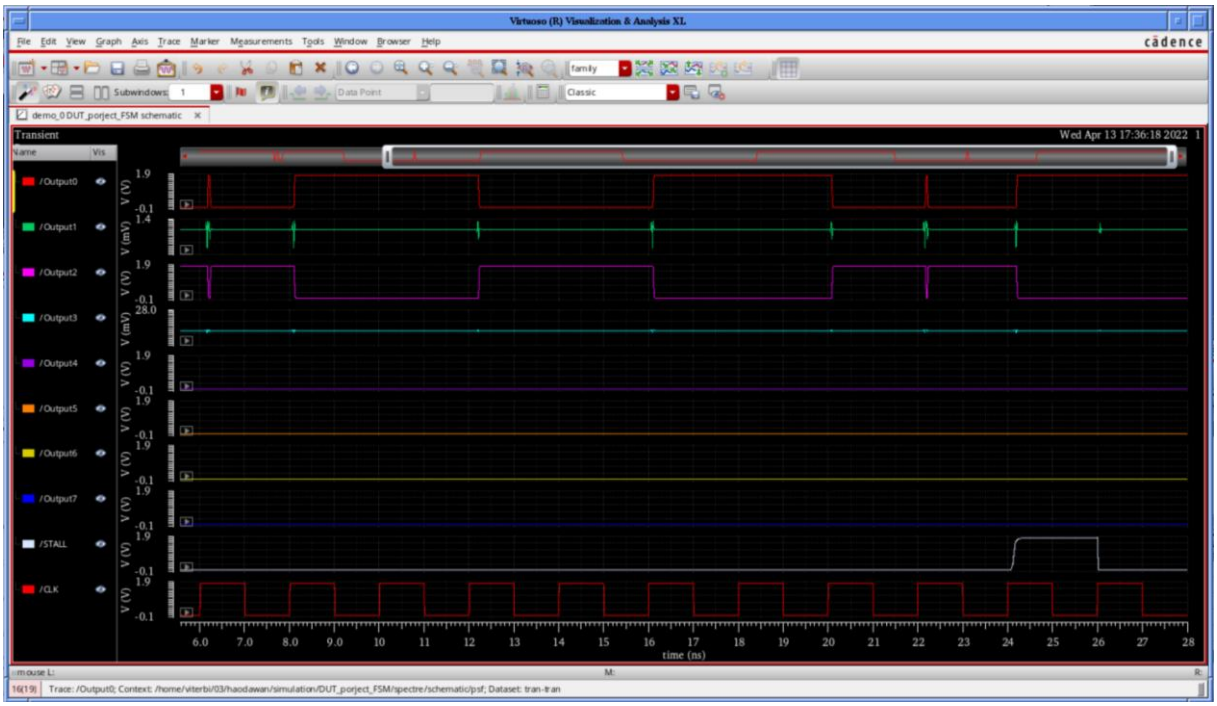
3 MINIMUM TIME PERIOD

1. You should report the minimum time period of your design. The minimum time period is the time period for which the clock of the entire system that outputs the correct results with no setup or hold violations.

Minimum time period: 0.234ns

2. Also, all functionality checks and timing report should be done using a DUT where all inputs pass through 1X inverter and outputs have a load of 4X





Time (ns)	Input A	Input B	Selection	Expected
6	0b0010	0b0010	M	0b0000100
8	0b0010	0b0010	D	0b0000001
10	0b0010	0b0010	D	0b0000001
12	0b0010	0b0010	M	0b0000100
14	0b0010	0b0010	M	0b0000100
16	0b0010	0b0010	M+D	0b0000001
18	0b0010	0b0010	D	0b0000001
20	0b0010	0b0010	M+D	0b0000100
22	0b0010	0b0010	M	0b0000100
24	0b0010	0b0010	N	STALL
26	0b0010	0b0010	M+D	0b0000001