EGC 455
SOC Design & Verification

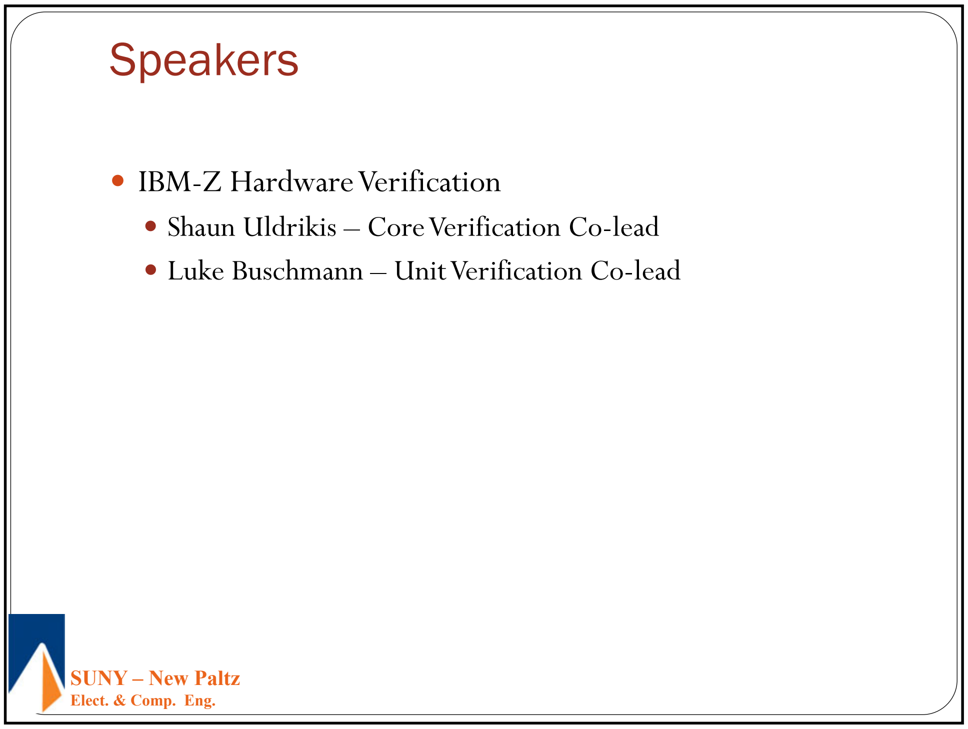
Functional Verification of Hardware

As presented by IBM

New Paltz
STATE UNIVERSITY OF NEW YORK

Baback Izadi
Division of Engineering Programs
bai@engr.newpaltz.edu

1

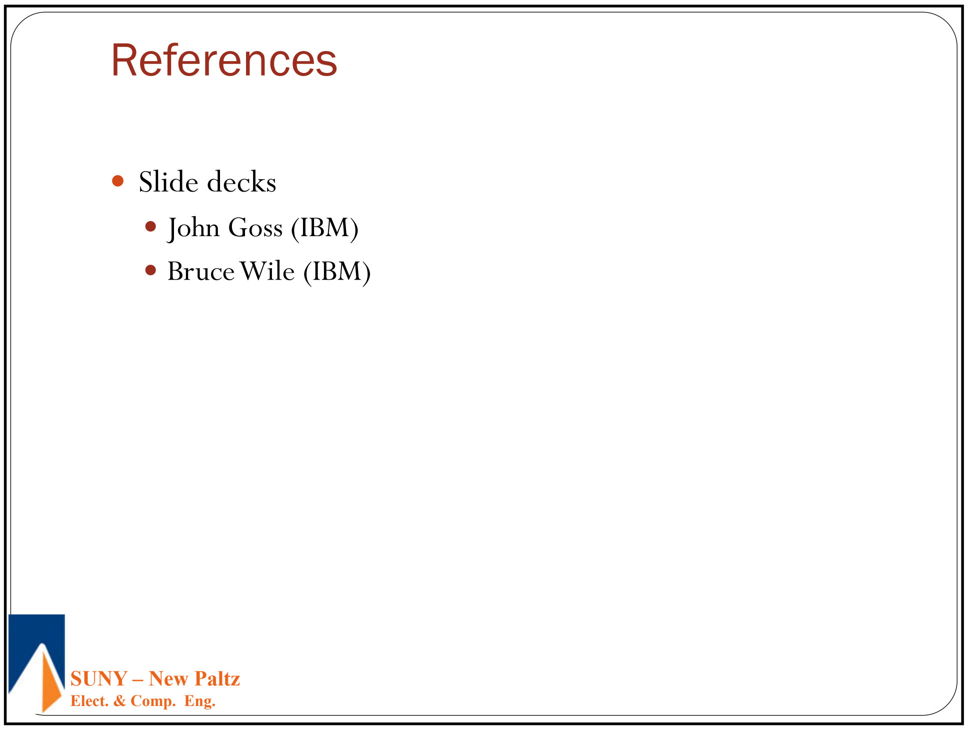


Speakers

- IBM-Z Hardware Verification
  - Shaun Uldrikis – Core Verification Co-lead
  - Luke Buschmann – Unit Verification Co-lead

SUNY – New Paltz
Elect. & Comp. Eng.

2

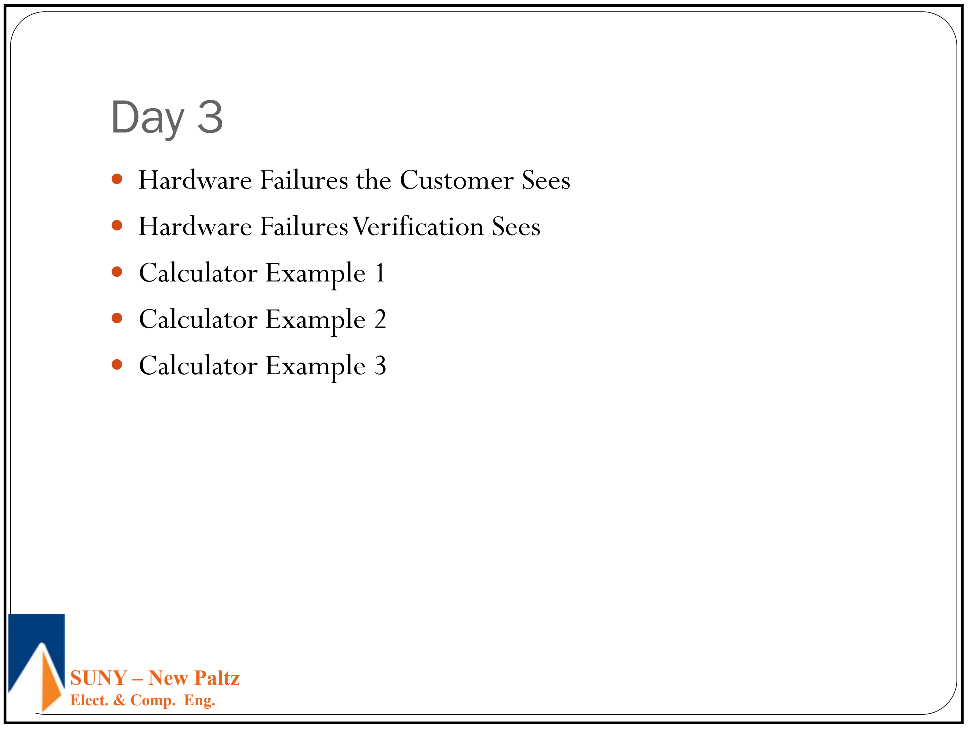


References

- Slide decks
  - John Goss (IBM)
  - Bruce Wile (IBM)

SUNY – New Paltz
Elect. & Comp. Eng.

3

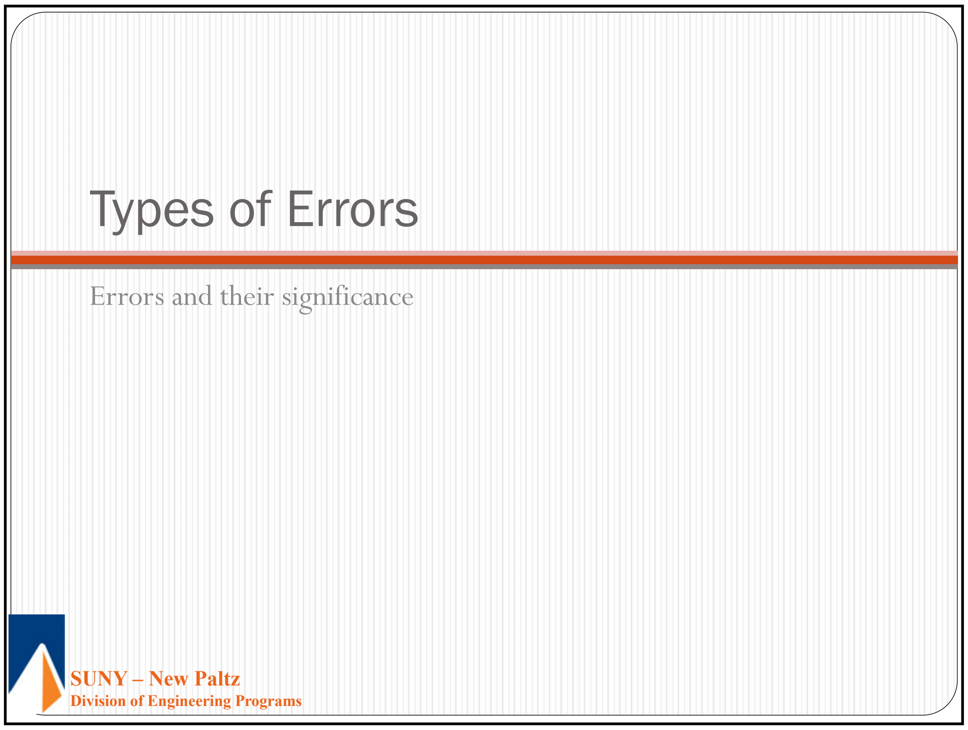


Day 3

- Hardware Failures the Customer Sees
- Hardware Failures Verification Sees
- Calculator Example 1
- Calculator Example 2
- Calculator Example 3

SUNY – New Paltz
Elect. & Comp. Eng.

4

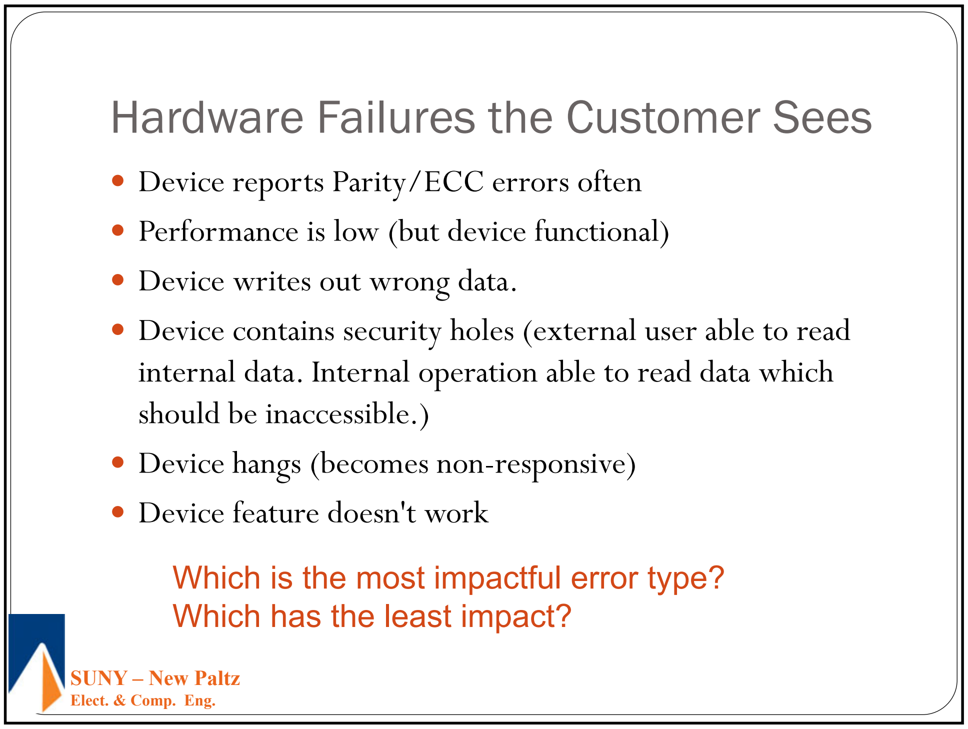


Types of Errors

Errors and their significance

SUNY – New Paltz
Division of Engineering Programs

5



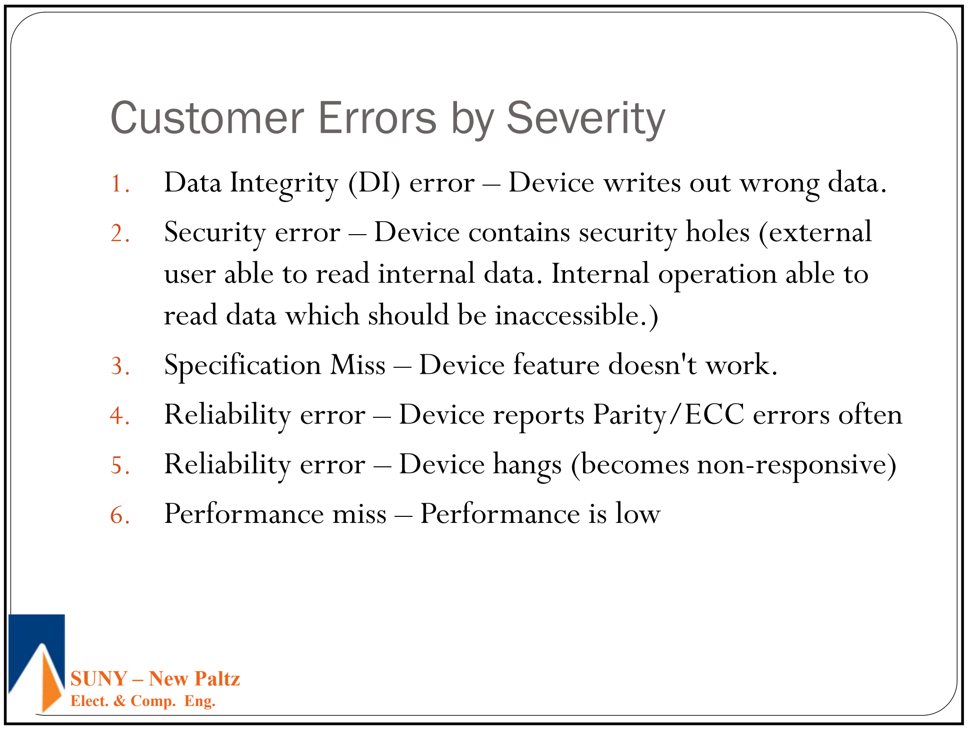
Hardware Failures the Customer Sees

- Device reports Parity/ECC errors often
- Performance is low (but device functional)
- Device writes out wrong data.
- Device contains security holes (external user able to read internal data. Internal operation able to read data which should be inaccessible.)
- Device hangs (becomes non-responsive)
- Device feature doesn't work

Which is the most impactful error type?
Which has the least impact?

SUNY – New Paltz
Elect. & Comp. Eng.

6

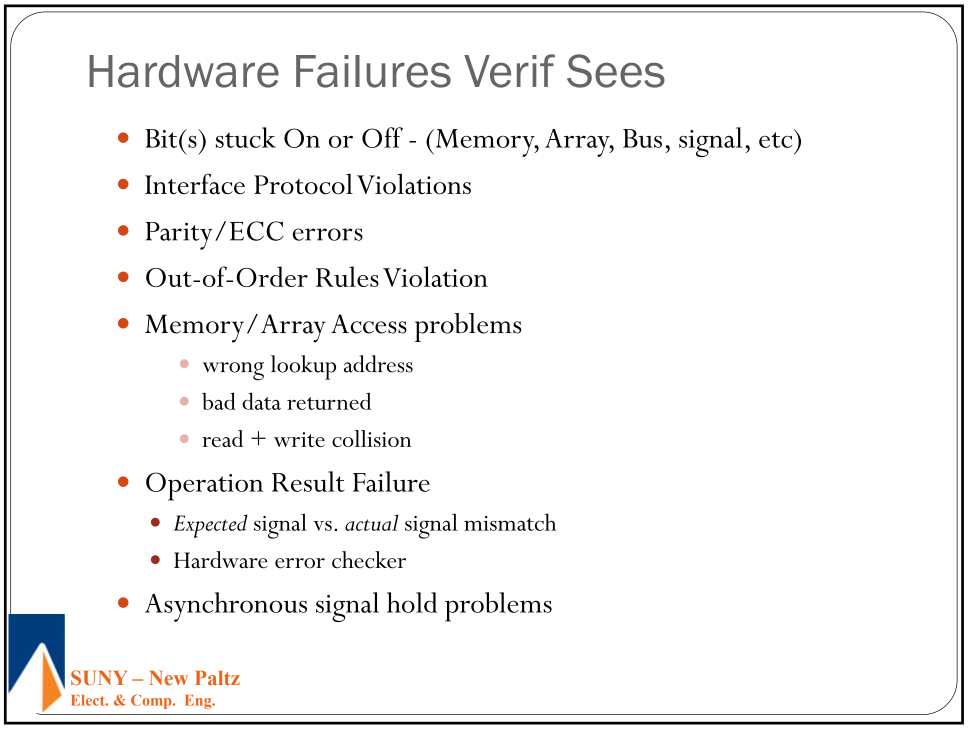


Customer Errors by Severity

1. Data Integrity (DI) error – Device writes out wrong data.
2. Security error – Device contains security holes (external user able to read internal data. Internal operation able to read data which should be inaccessible.)
3. Specification Miss – Device feature doesn't work.
4. Reliability error – Device reports Parity/ECC errors often
5. Reliability error – Device hangs (becomes non-responsive)
6. Performance miss – Performance is low

SUNY – New Paltz
Elect. & Comp. Eng.

7

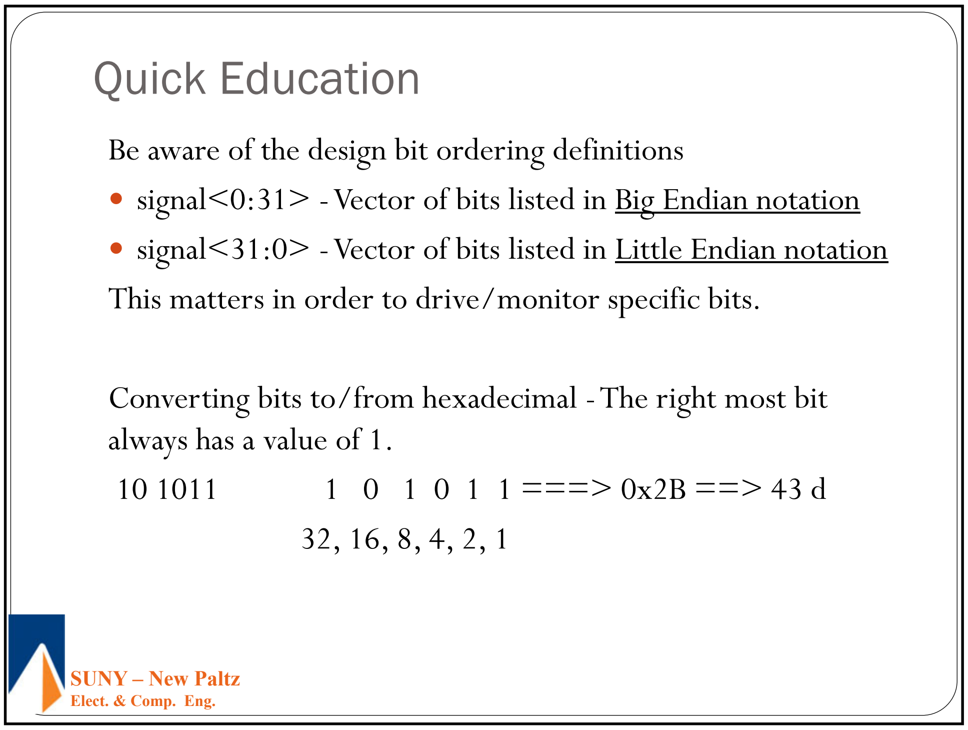


Hardware Failures Verif Sees

- Bit(s) stuck On or Off - (Memory, Array, Bus, signal, etc)
- Interface Protocol Violations
- Parity/ECC errors
- Out-of-Order Rules Violation
- Memory/Array Access problems
  - wrong lookup address
  - bad data returned
  - read + write collision
- Operation Result Failure
  - *Expected* signal vs. *actual* signal mismatch
  - Hardware error checker
- Asynchronous signal hold problems

SUNY – New Paltz
Elect. & Comp. Eng.

8



Quick Education

Be aware of the design bit ordering definitions

- signal<0:31> - Vector of bits listed in Big Endian notation
- signal<31:0> - Vector of bits listed in Little Endian notation

This matters in order to drive/monitor specific bits.

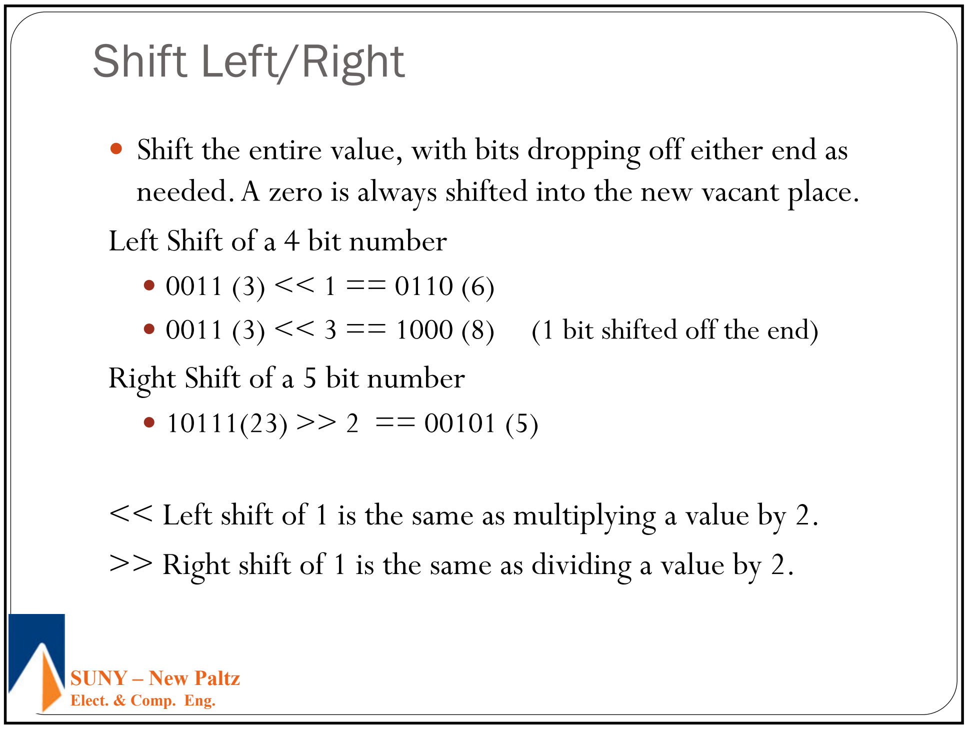
Converting bits to/from hexadecimal - The right most bit always has a value of 1.

10 1011 1 0 1 0 1 1 ===> 0x2B ==> 43 d

32, 16, 8, 4, 2, 1

SUNY – New Paltz
Elect. & Comp. Eng.

9



Shift Left/Right

- Shift the entire value, with bits dropping off either end as needed. A zero is always shifted into the new vacant place.

Left Shift of a 4 bit number

- 0011 (3) << 1 == 0110 (6)
- 0011 (3) << 3 == 1000 (8) (1 bit shifted off the end)

Right Shift of a 5 bit number

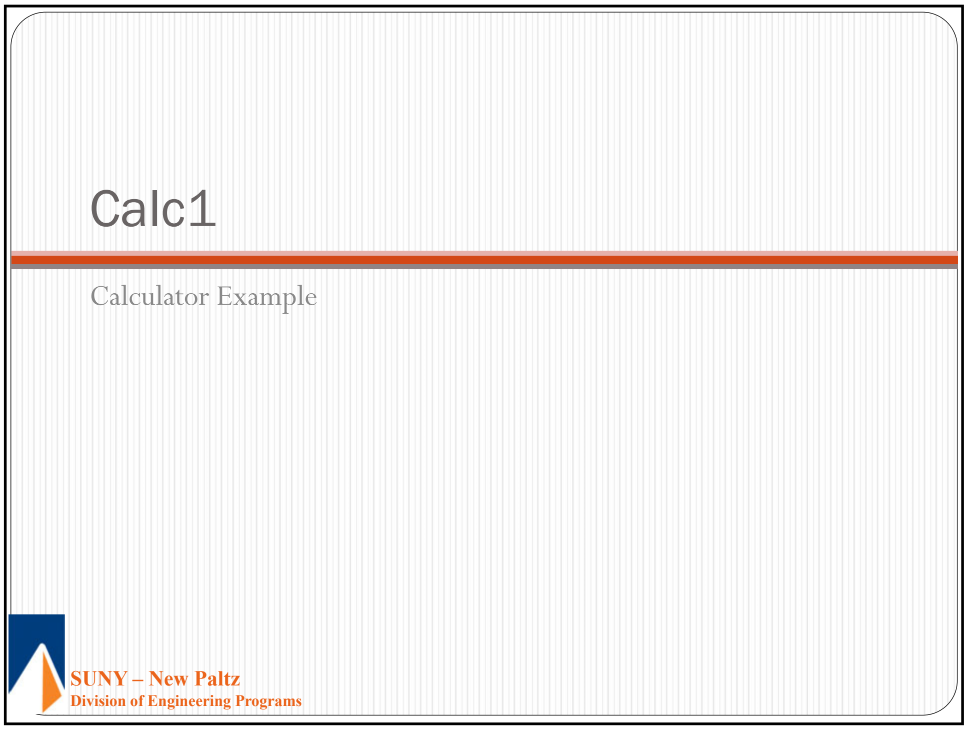
- 10111(23) >> 2 == 00101 (5)

<< Left shift of 1 is the same as multiplying a value by 2.

>> Right shift of 1 is the same as dividing a value by 2.

SUNY – New Paltz
Elect. & Comp. Eng.

10

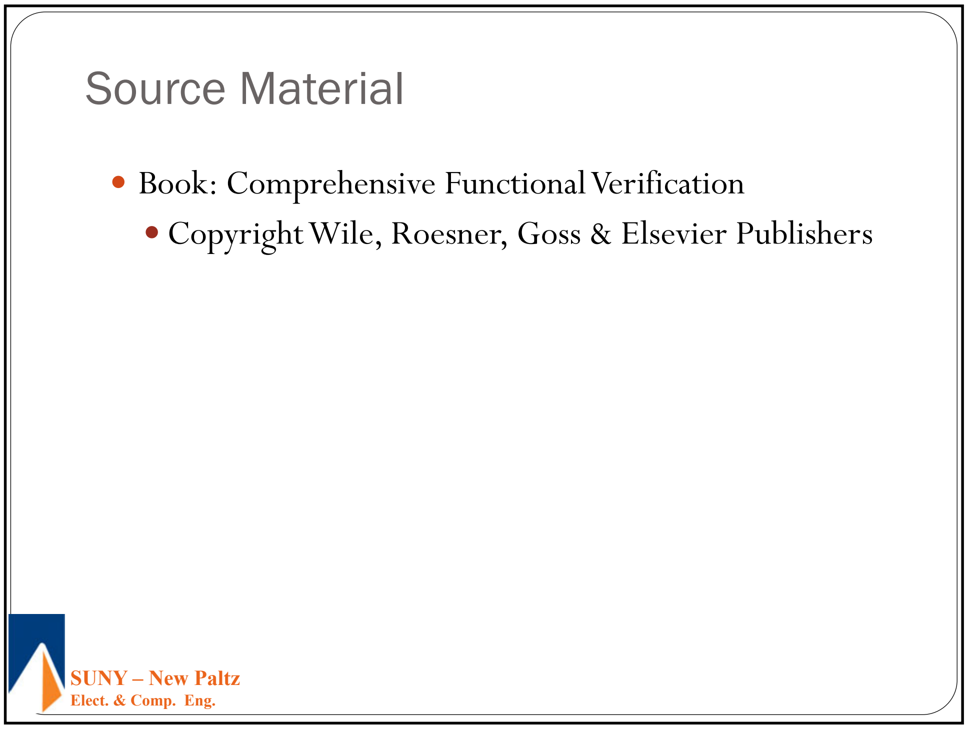


Calc1

Calculator Example

SUNY – New Paltz
Division of Engineering Programs

11

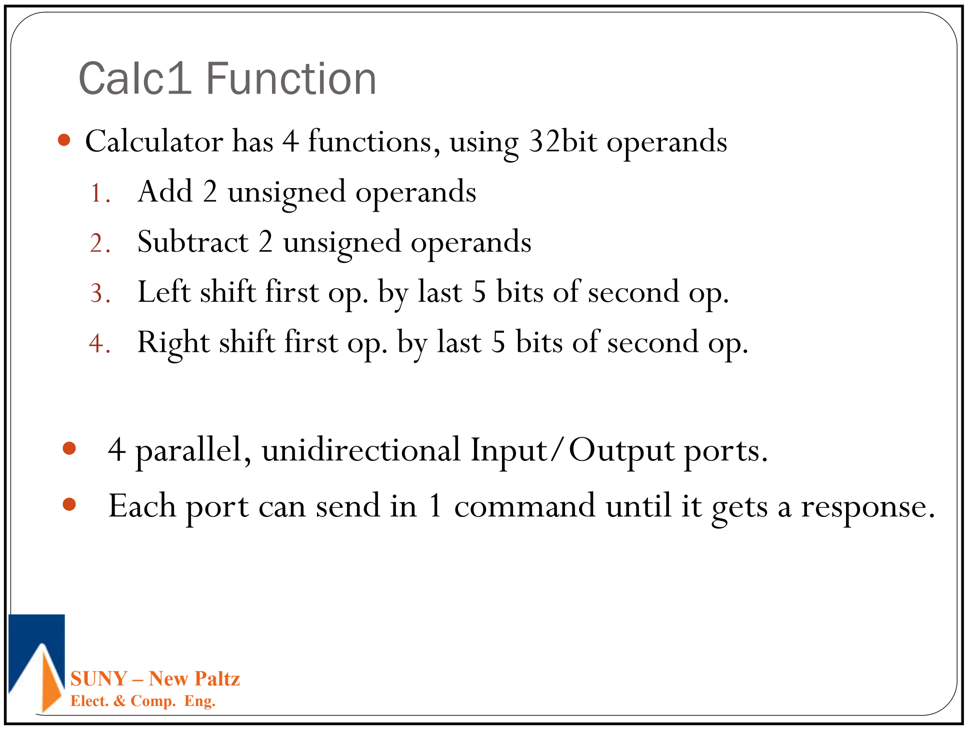


Source Material

- Book: Comprehensive Functional Verification
  - Copyright Wile, Roesner, Goss & Elsevier Publishers

SUNY – New Paltz
Elect. & Comp. Eng.

12



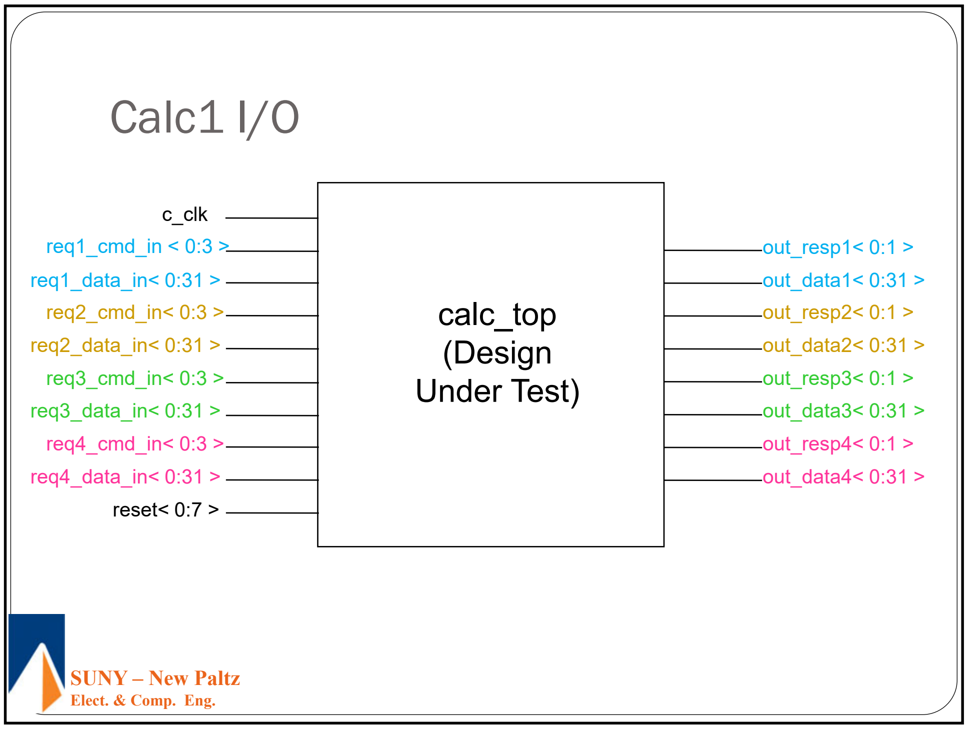
Calc1 Function

- Calculator has 4 functions, using 32bit operands
  1. Add 2 unsigned operands
  2. Subtract 2 unsigned operands
  3. Left shift first op. by last 5 bits of second op.
  4. Right shift first op. by last 5 bits of second op.

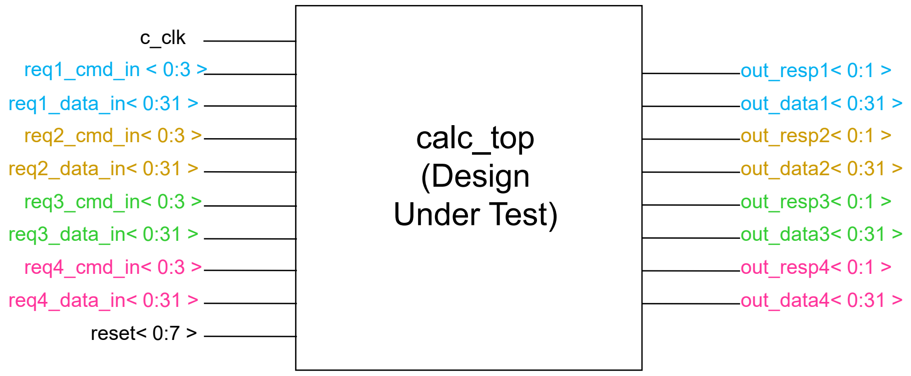
- 4 parallel, unidirectional Input/Output ports.
- Each port can send in 1 command until it gets a response.

SUNY – New Paltz
Elect. & Comp. Eng.

13



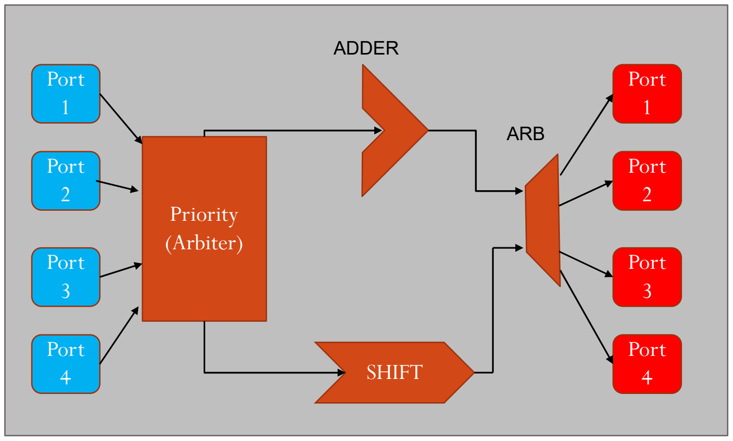
Calc1 I/O



SUNY – New Paltz
Elect. & Comp. Eng.

14

Calc1 Block Diagram



SUNY – New Paltz
Elect. & Comp. Eng.

15

Calc1 Details - Requests

- The opcodes for commands on cmd_in<0..3> are shown below.
- Operand1 enters the calculator on data_in<0..31> during the same cycle that the command is issued.
  Operand2 follows one cycle after operand1.

Encoding of Commands

code	operation
0	no operation
1	add operand1 to operand2
2	subtract operand2 from operand1
5	shift operand1 to the left by operand2 places.
6	shift operand1 to the right by operand2 places.

SUNY – New Paltz
Elect. & Comp. Eng.

16

Calc1 Details - Responses

- The opcodes for responses on out_resp<0..1> are below.
- Valid result data exits the calculator on out_data<0..31> during the same cycle as the response. (example timing diagram on following page)

Encoding of Responses

code	operation
0	no response
1	operation completed successfully
2	op. unsuccessful (invalid opcode, overflow or underflow)
3	internal error encountered

SUNY – New Paltz
Elect. & Comp. Eng.

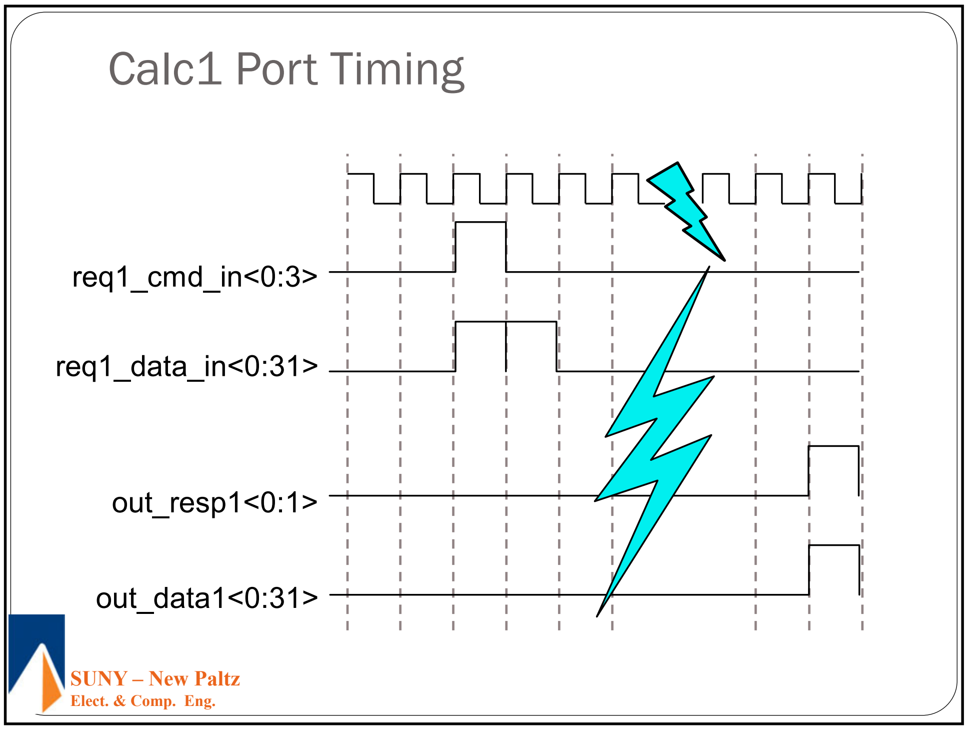
17

Calc1 Function

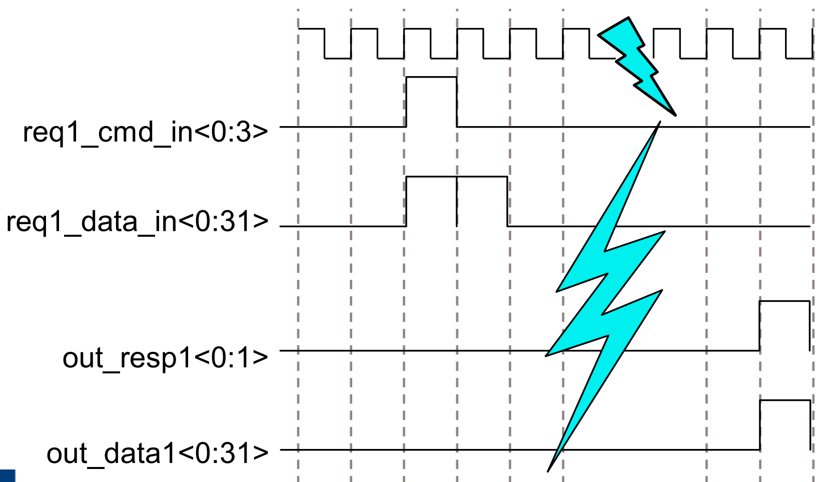
- Calculator can handle 4 requests in parallel
  1. All requestors have equal priority
  2. Priority logic works on first come first serve algorithm
  3. Priority logic allows for 1 add or subtract at a time and one shift operation at a time

SUNY – New Paltz
Elect. & Comp. Eng.

18

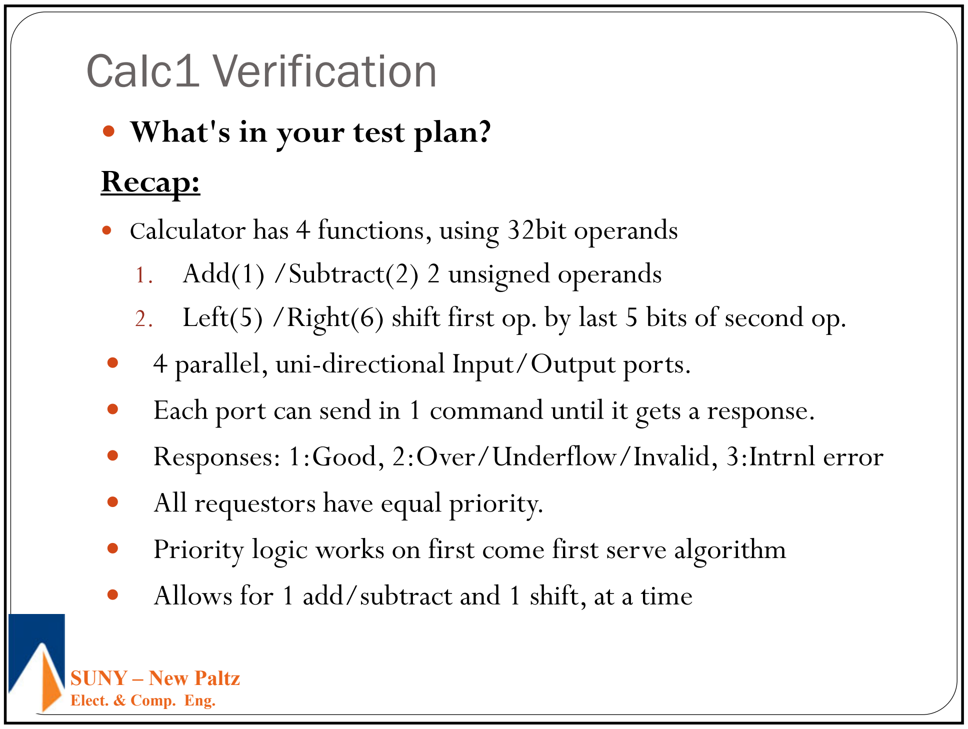


Calc1 Port Timing



SUNY – New Paltz
Elect. & Comp. Eng.

19



Calc1 Verification

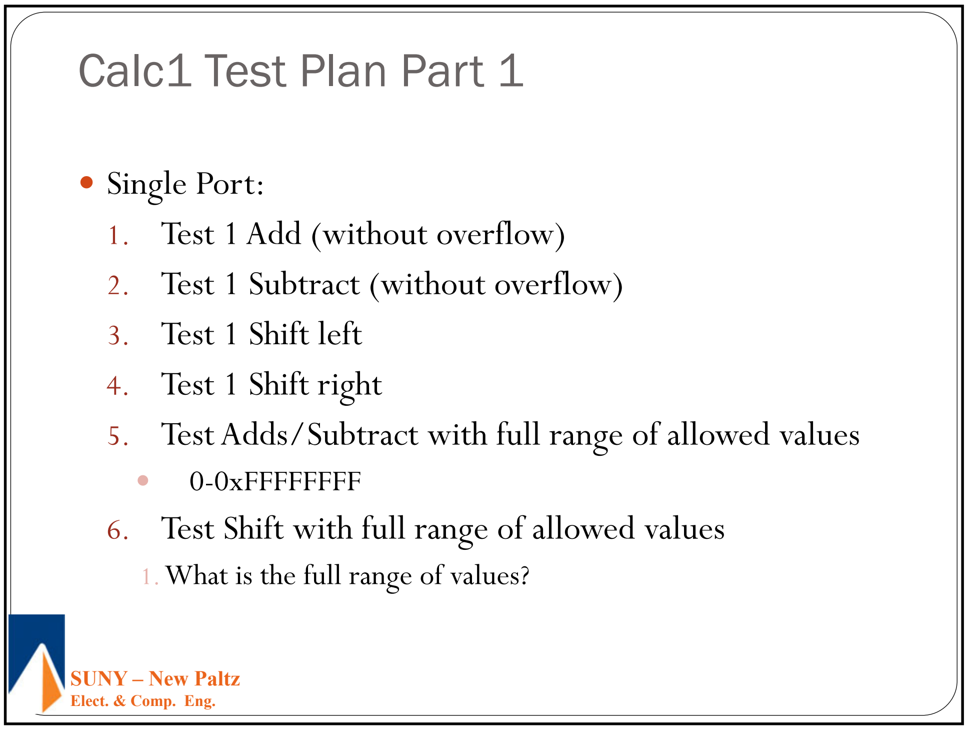
- **What's in your test plan?**

<u>Recap:</u>

- Calculator has 4 functions, using 32bit operands
  1. Add(1) /Subtract(2) 2 unsigned operands
  2. Left(5) /Right(6) shift first op. by last 5 bits of second op.
- 4 parallel, uni-directional Input/Output ports.
- Each port can send in 1 command until it gets a response.
- Responses: 1:Good, 2:Over/Underflow/Invalid, 3:Intrnl error
- All requestors have equal priority.
- Priority logic works on first come first serve algorithm
- Allows for 1 add/subtract and 1 shift, at a time

SUNY – New Paltz
Elect. & Comp. Eng.

20



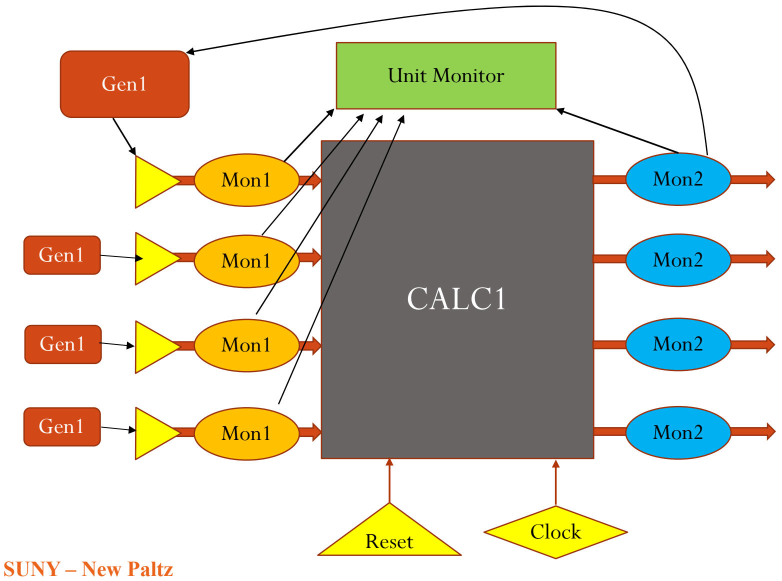
Calc1 Test Plan Part 1

- Single Port:
  1. Test 1 Add (without overflow)
  2. Test 1 Subtract (without overflow)
  3. Test 1 Shift left
  4. Test 1 Shift right
  5. Test Adds/Subtract with full range of allowed values
     - 0-0xFFFFFFFF
  6. Test Shift with full range of allowed values
     1. What is the full range of values?

SUNY – New Paltz
Elect. & Comp. Eng.

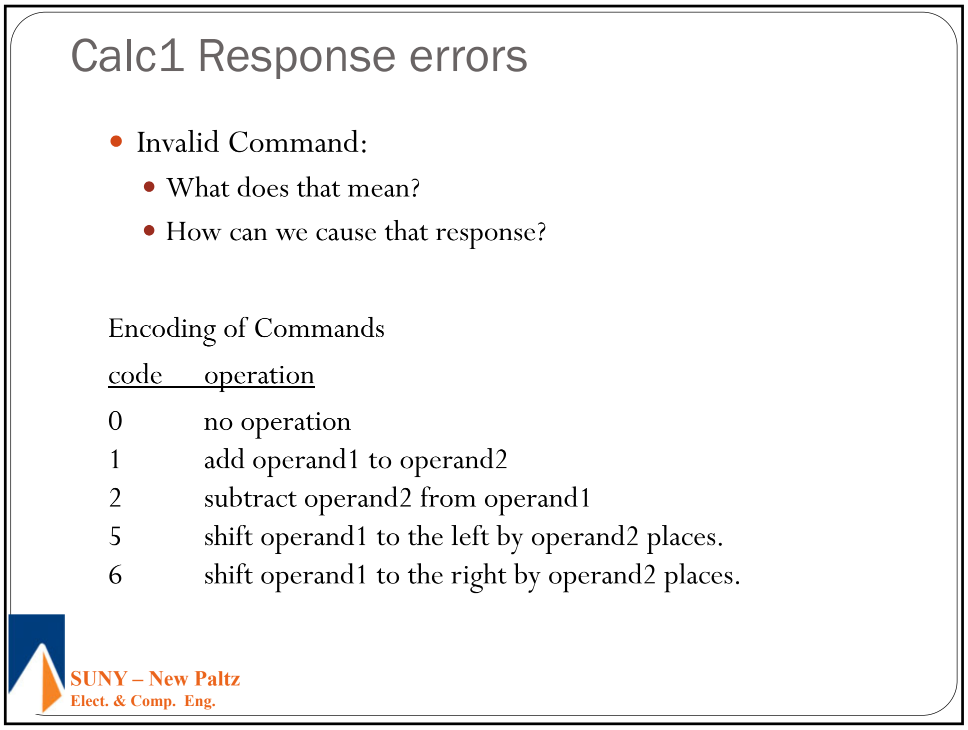
21

Calc1 Environment



SUNY – New Paltz
Elect. & Comp. Eng.

22



Calc1 Response errors

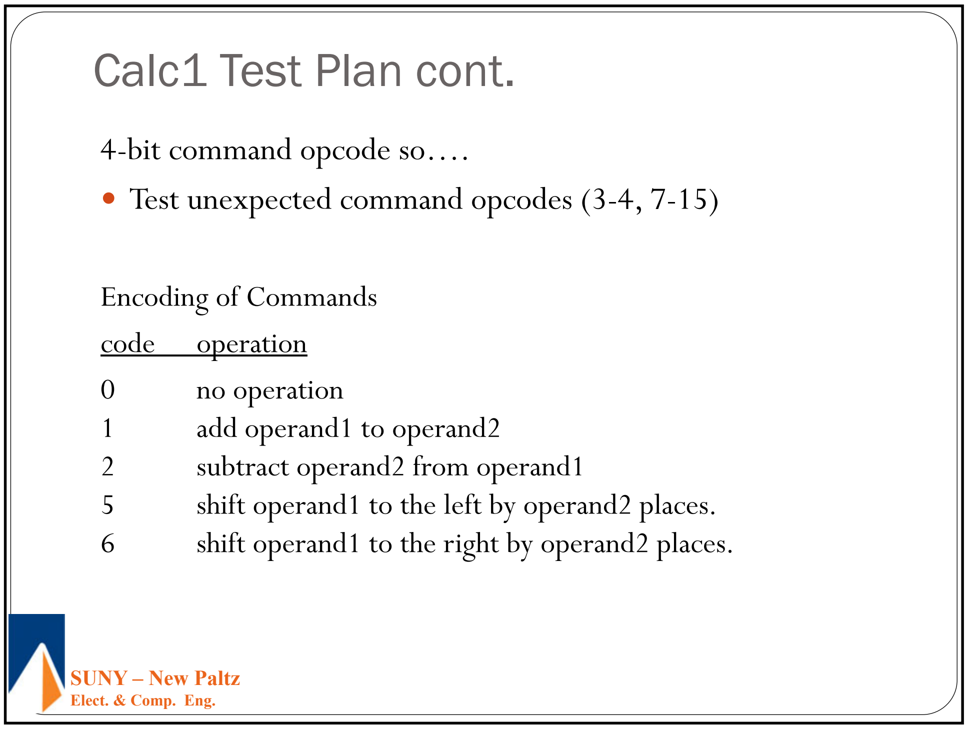
- Invalid Command:
  - What does that mean?
  - How can we cause that response?

Encoding of Commands

code	operation
0	no operation
1	add operand1 to operand2
2	subtract operand2 from operand1
5	shift operand1 to the left by operand2 places.
6	shift operand1 to the right by operand2 places.

SUNY – New Paltz
Elect. & Comp. Eng.

23



Calc1 Test Plan cont.

4-bit command opcode so….

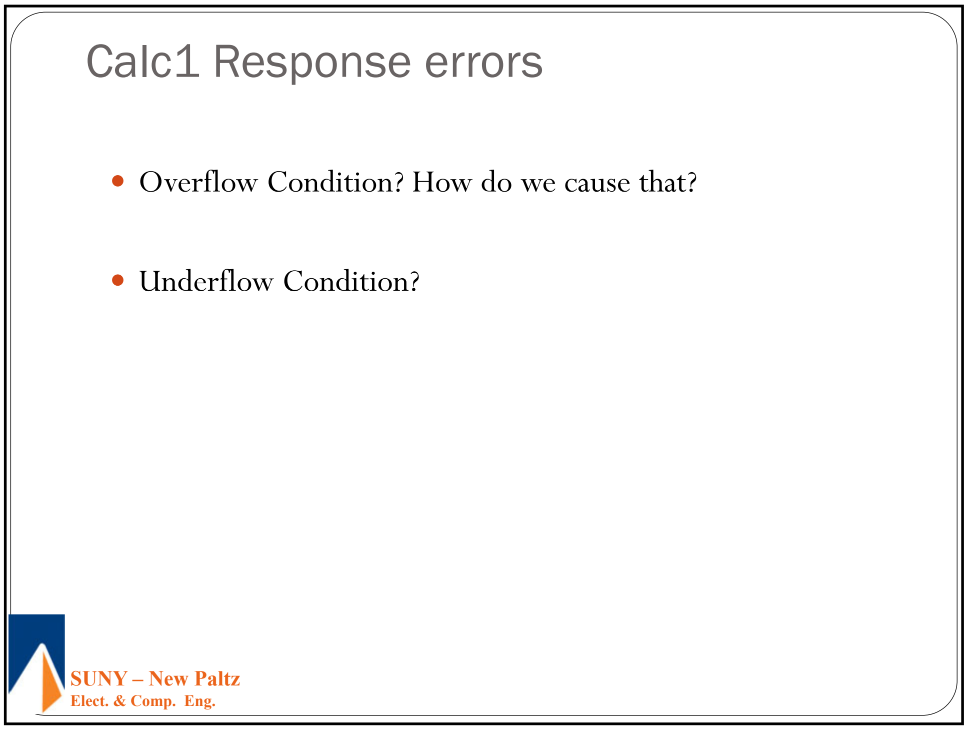
- Test unexpected command opcodes (3-4, 7-15)

Encoding of Commands

code	operation
0	no operation
1	add operand1 to operand2
2	subtract operand2 from operand1
5	shift operand1 to the left by operand2 places.
6	shift operand1 to the right by operand2 places.

SUNY – New Paltz
Elect. & Comp. Eng.

24

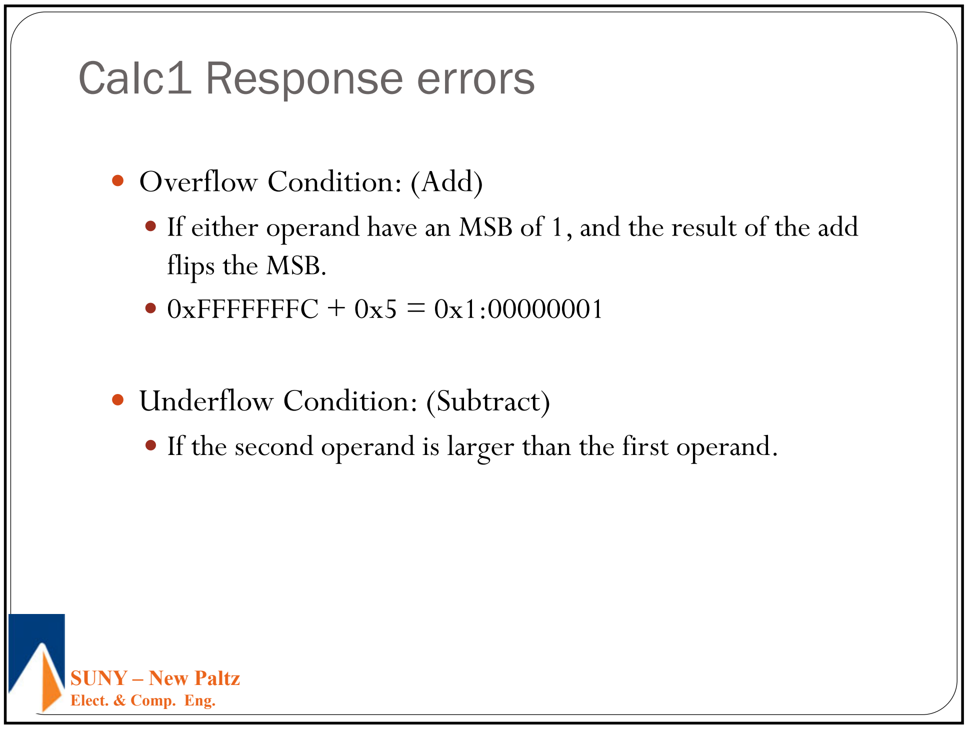


Calc1 Response errors

- Overflow Condition? How do we cause that?
- Underflow Condition?

SUNY – New Paltz
Elect. & Comp. Eng.

25

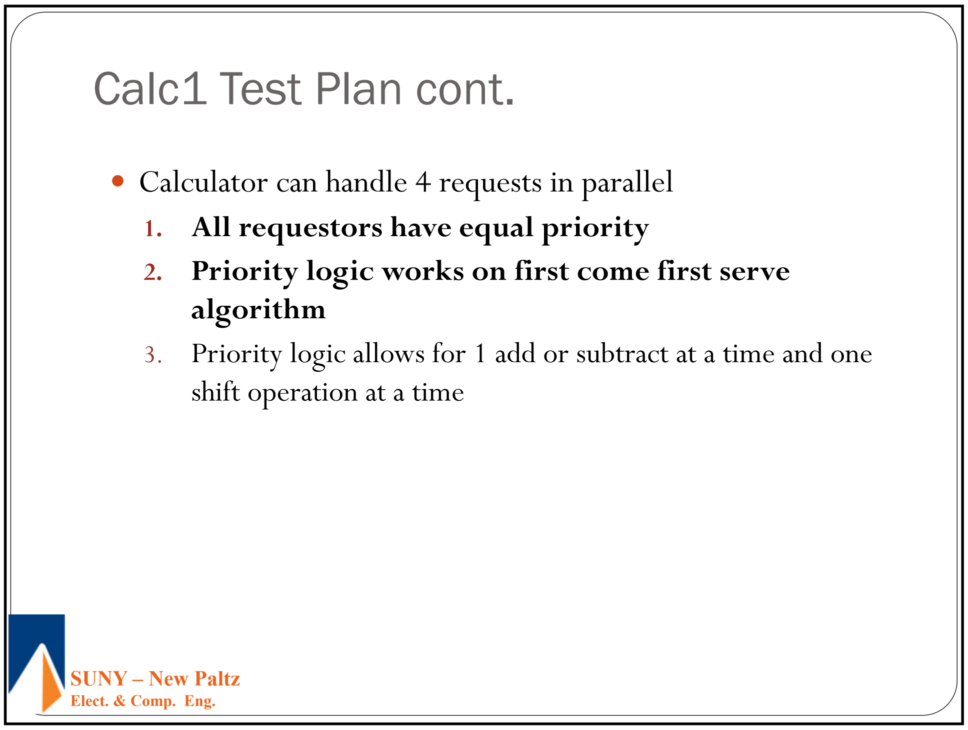


Calc1 Response errors

- Overflow Condition: (Add)
  - If either operand have an MSB of 1, and the result of the add flips the MSB.
  - 0xFFFFFFFC + 0x5 = 0x1:00000001
- Underflow Condition: (Subtract)
  - If the second operand is larger than the first operand.

SUNY – New Paltz
Elect. & Comp. Eng.

26

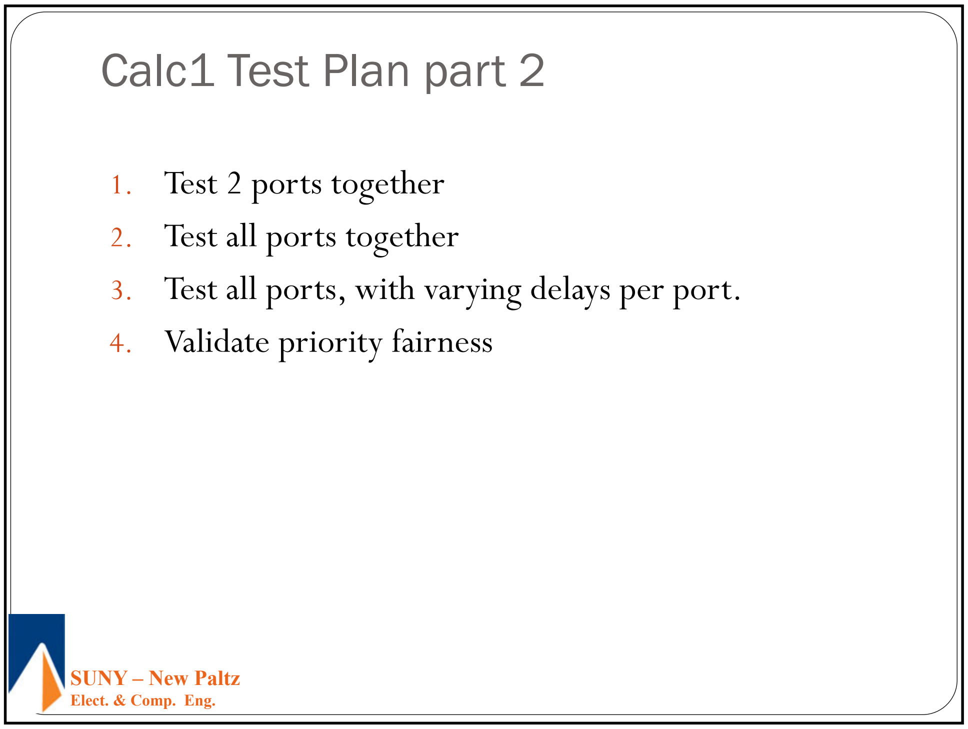


Calc1 Test Plan cont.

- Calculator can handle 4 requests in parallel
  1. **All requestors have equal priority**
  2. **Priority logic works on first come first serve algorithm**
  3. Priority logic allows for 1 add or subtract at a time and one shift operation at a time

SUNY – New Paltz
Elect. & Comp. Eng.

27

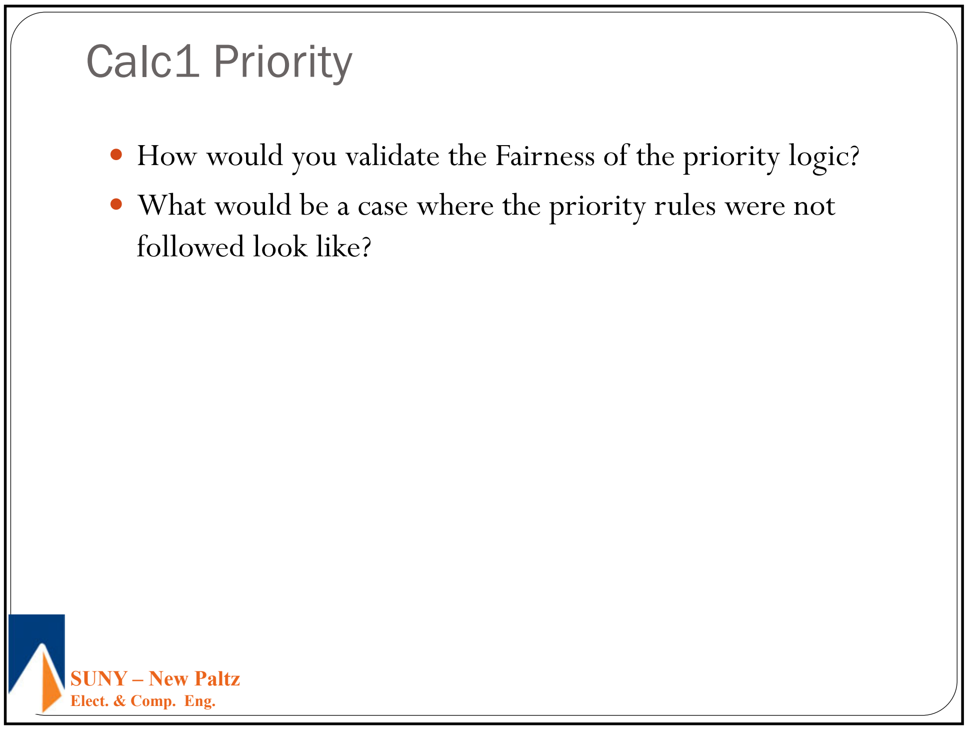


Calc1 Test Plan part 2

1. Test 2 ports together
2. Test all ports together
3. Test all ports, with varying delays per port.
4. Validate priority fairness

SUNY – New Paltz
Elect. & Comp. Eng.

28

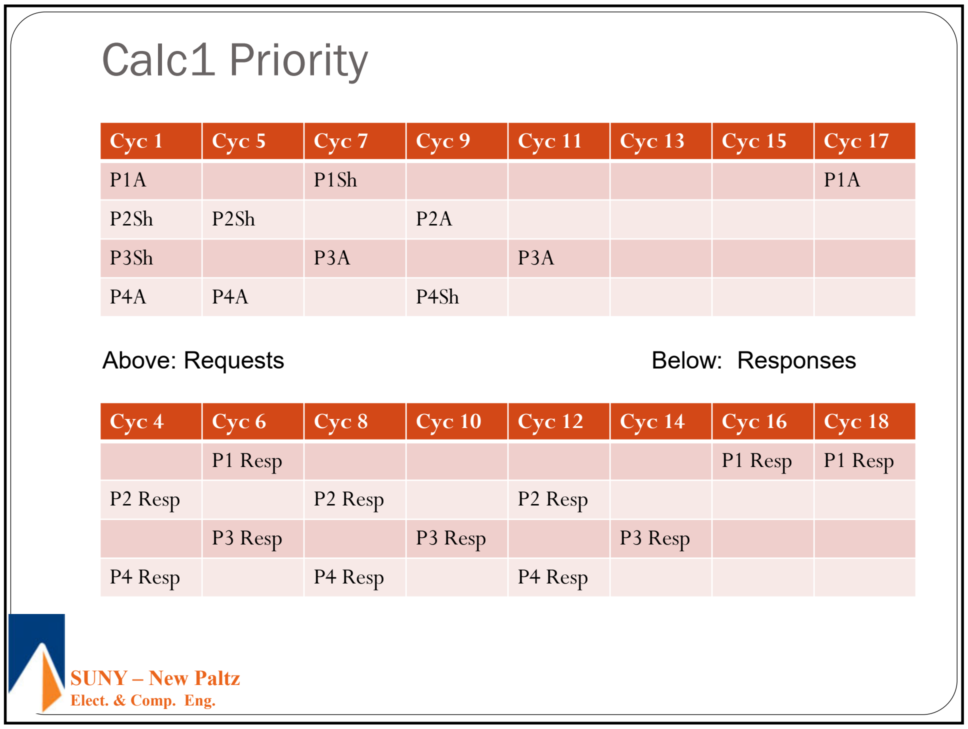


Calc1 Priority

- How would you validate the Fairness of the priority logic?
- What would be a case where the priority rules were not followed look like?

SUNY – New Paltz
Elect. & Comp. Eng.

29



Calc1 Priority

Cyc 1	Cyc 5	Cyc 7	Cyc 9	Cyc 11	Cyc 13	Cyc 15	Cyc 17
P1A		P1Sh					P1A
P2Sh	P2Sh		P2A				
P3Sh		P3A		P3A			
P4A	P4A		P4Sh				

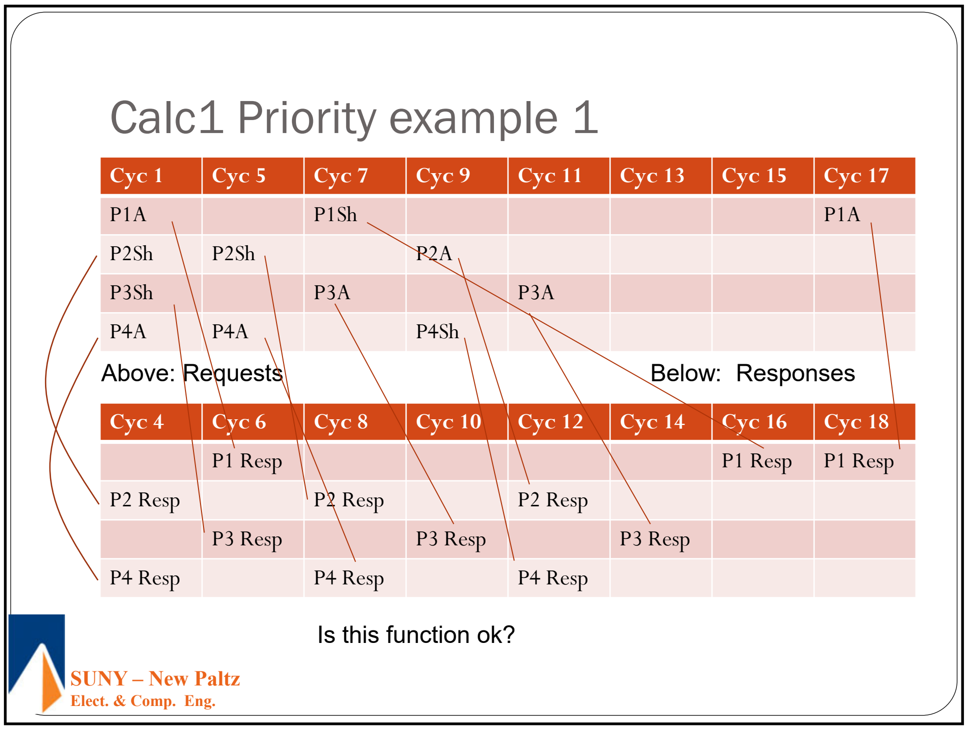
Above: Requests

Below: Responses

Cyc 4	Cyc 6	Cyc 8	Cyc 10	Cyc 12	Cyc 14	Cyc 16	Cyc 18
	P1 Resp					P1 Resp	P1 Resp
P2 Resp		P2 Resp		P2 Resp			
	P3 Resp		P3 Resp		P3 Resp		
P4 Resp		P4 Resp		P4 Resp			

SUNY – New Paltz
Elect. & Comp. Eng.

30



Calc1 Priority example 1

Cyc 1	Cyc 5	Cyc 7	Cyc 9	Cyc 11	Cyc 13	Cyc 15	Cyc 17
P1A		P1Sh					P1A
P2Sh	P2Sh		P2A				
P3Sh		P3A		P3A			
P4A	P4A		P4Sh				

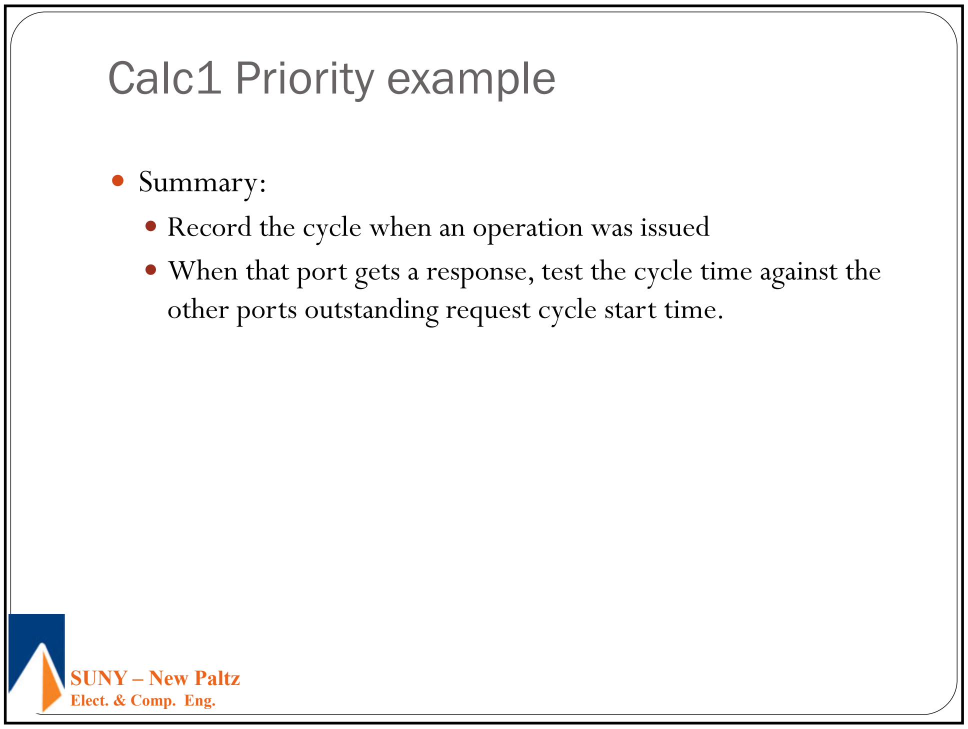
Above: Requests Below: Responses

Cyc 4	Cyc 6	Cyc 8	Cyc 10	Cyc 12	Cyc 14	Cyc 16	Cyc 18
	P1 Resp					P1 Resp	P1 Resp
P2 Resp		P2 Resp		P2 Resp			
	P3 Resp		P3 Resp		P3 Resp		
P4 Resp		P4 Resp		P4 Resp			

Is this function ok?

SUNY – New Paltz
Elect. & Comp. Eng.

31



Calc1 Priority example

- Summary:
  - Record the cycle when an operation was issued
  - When that port gets a response, test the cycle time against the other ports outstanding request cycle start time.

SUNY – New Paltz
Elect. & Comp. Eng.

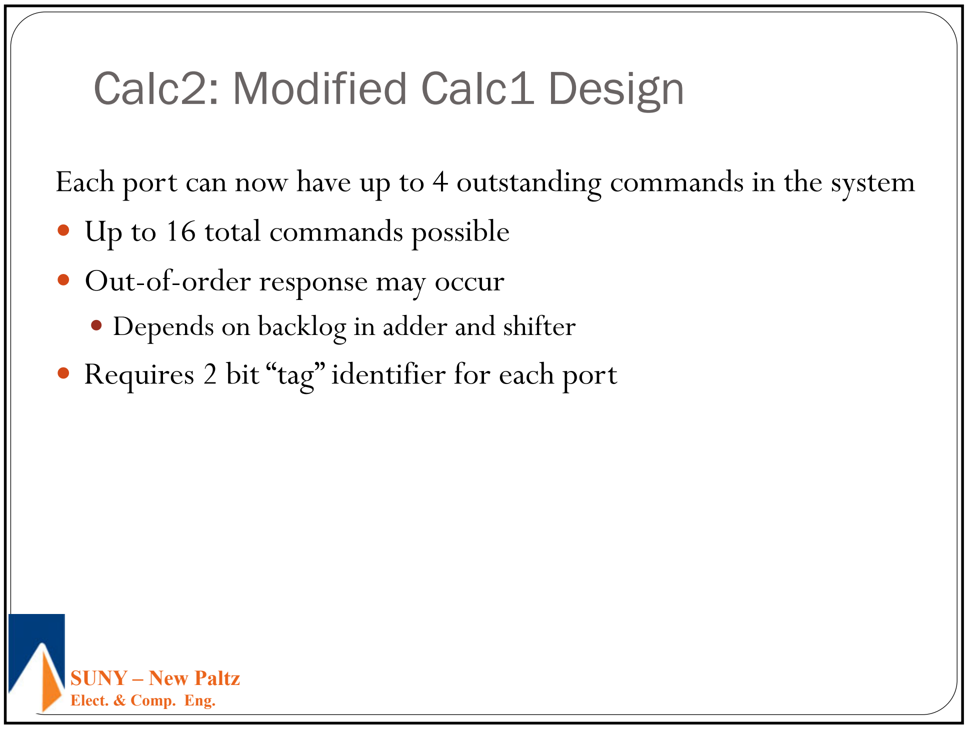
32

Calc2

Enhancement

SUNY – New Paltz
Division of Engineering Programs

33



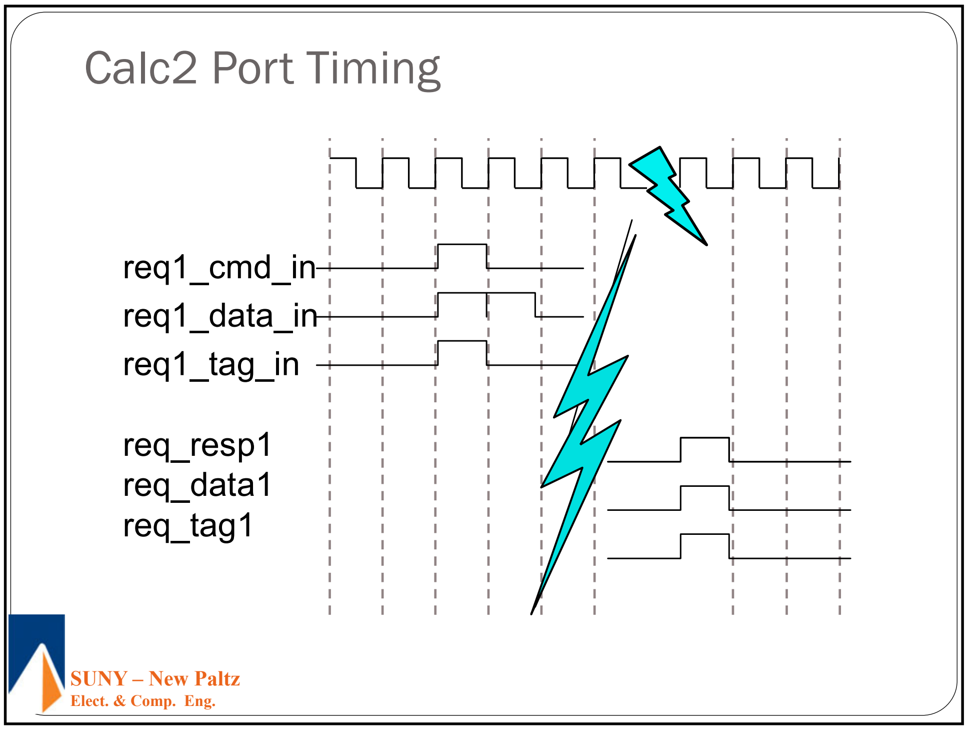
Calc2: Modified Calc1 Design

Each port can now have up to 4 outstanding commands in the system

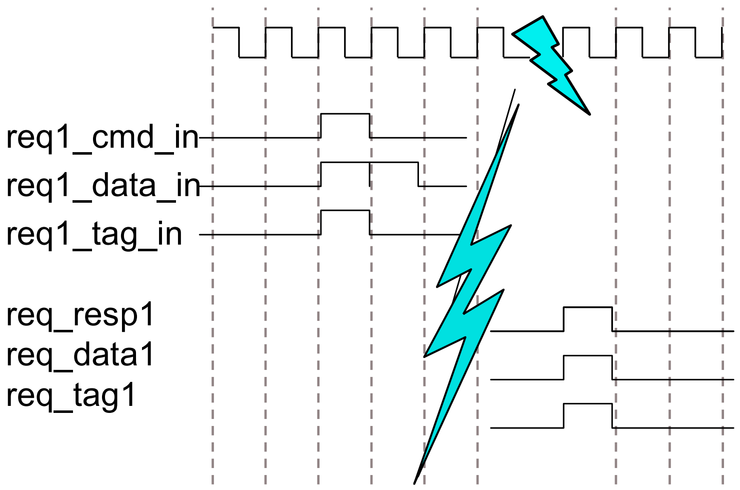
- Up to 16 total commands possible
- Out-of-order response may occur
  - Depends on backlog in adder and shifter
- Requires 2 bit "tag" identifier for each port

SUNY – New Paltz
Elect. & Comp. Eng.

34

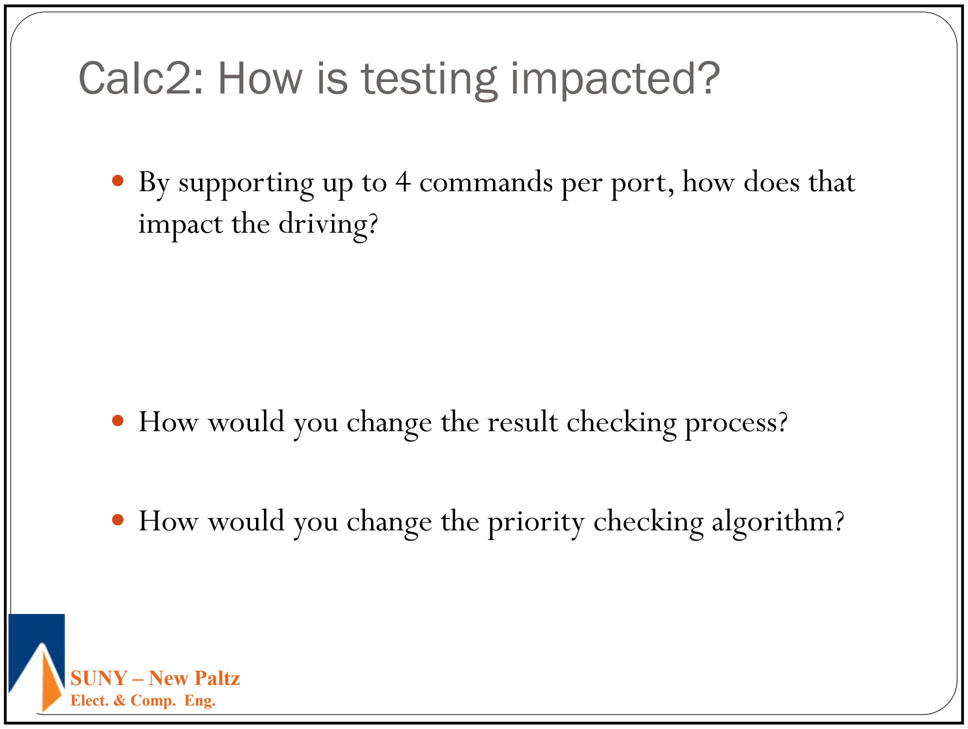


Calc2 Port Timing



SUNY – New Paltz
Elect. & Comp. Eng.

35

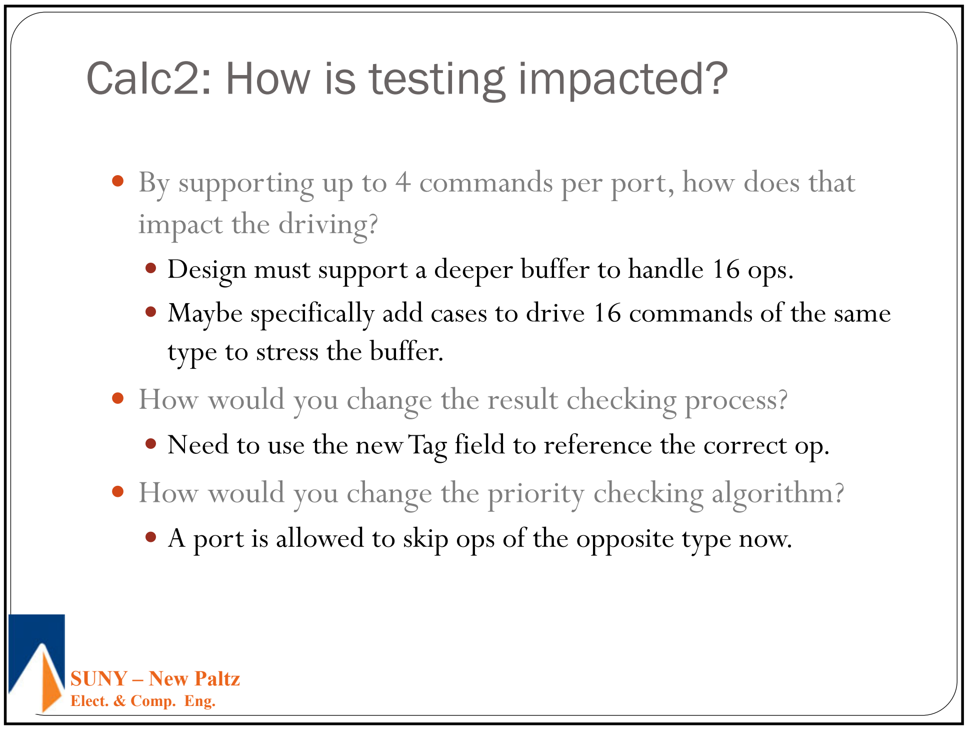


Calc2: How is testing impacted?

- By supporting up to 4 commands per port, how does that impact the driving?
- How would you change the result checking process?
- How would you change the priority checking algorithm?

SUNY – New Paltz
Elect. & Comp. Eng.

36

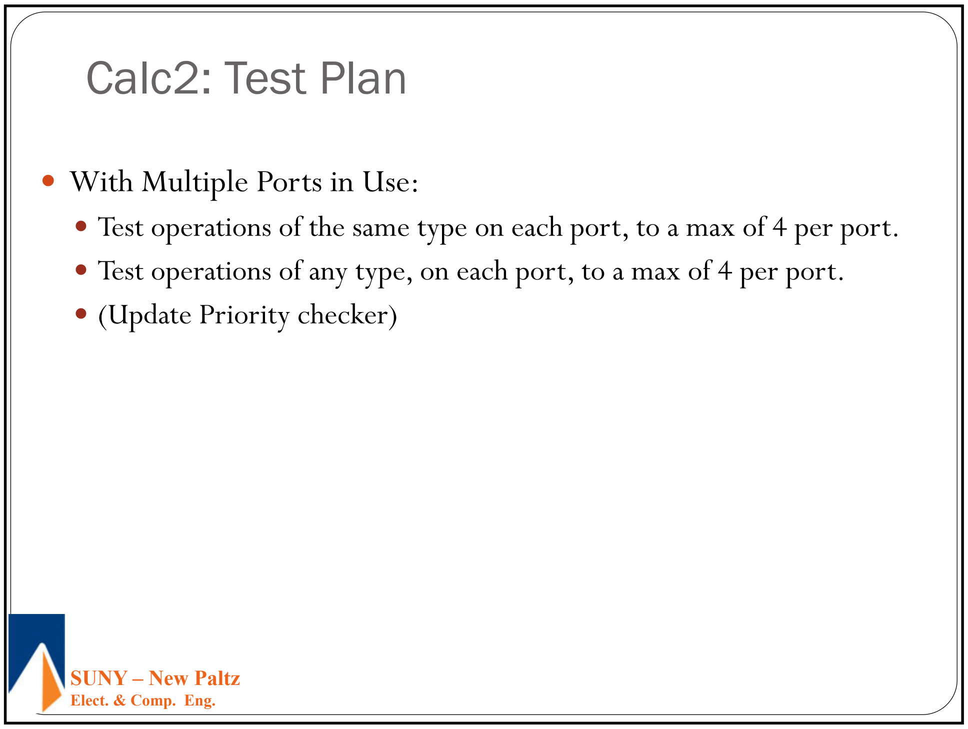


Calc2: How is testing impacted?

- By supporting up to 4 commands per port, how does that impact the driving?
  - Design must support a deeper buffer to handle 16 ops.
  - Maybe specifically add cases to drive 16 commands of the same type to stress the buffer.
- How would you change the result checking process?
  - Need to use the new Tag field to reference the correct op.
- How would you change the priority checking algorithm?
  - A port is allowed to skip ops of the opposite type now.

SUNY – New Paltz
Elect. & Comp. Eng.

37

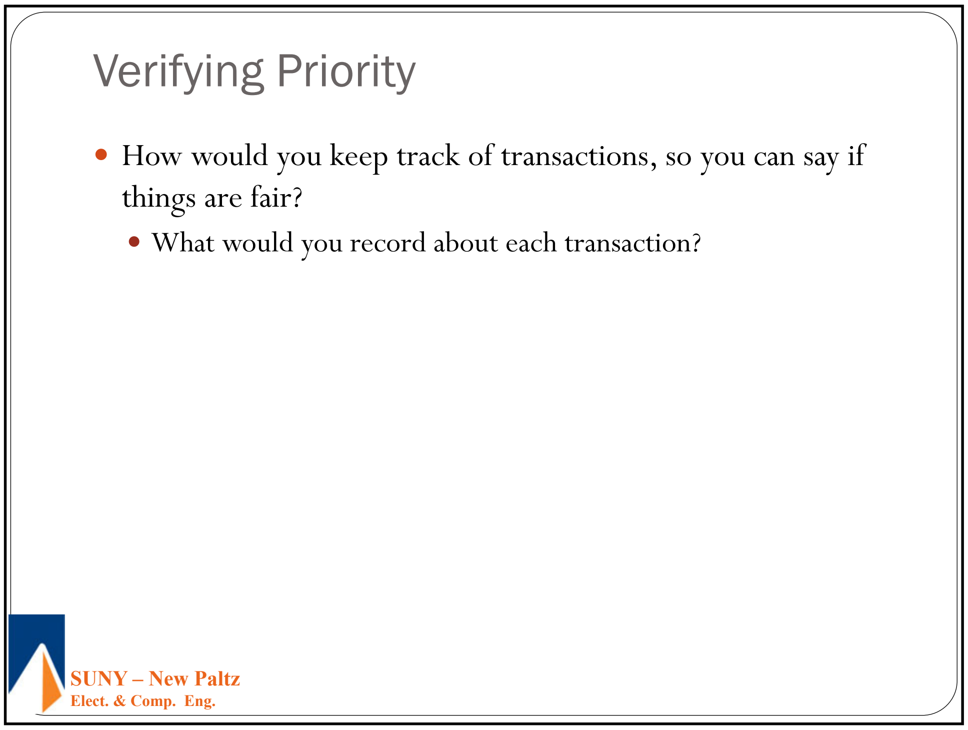


Calc2: Test Plan

- With Multiple Ports in Use:
  - Test operations of the same type on each port, to a max of 4 per port.
  - Test operations of any type, on each port, to a max of 4 per port.
  - (Update Priority checker)

SUNY – New Paltz
Elect. & Comp. Eng.

38

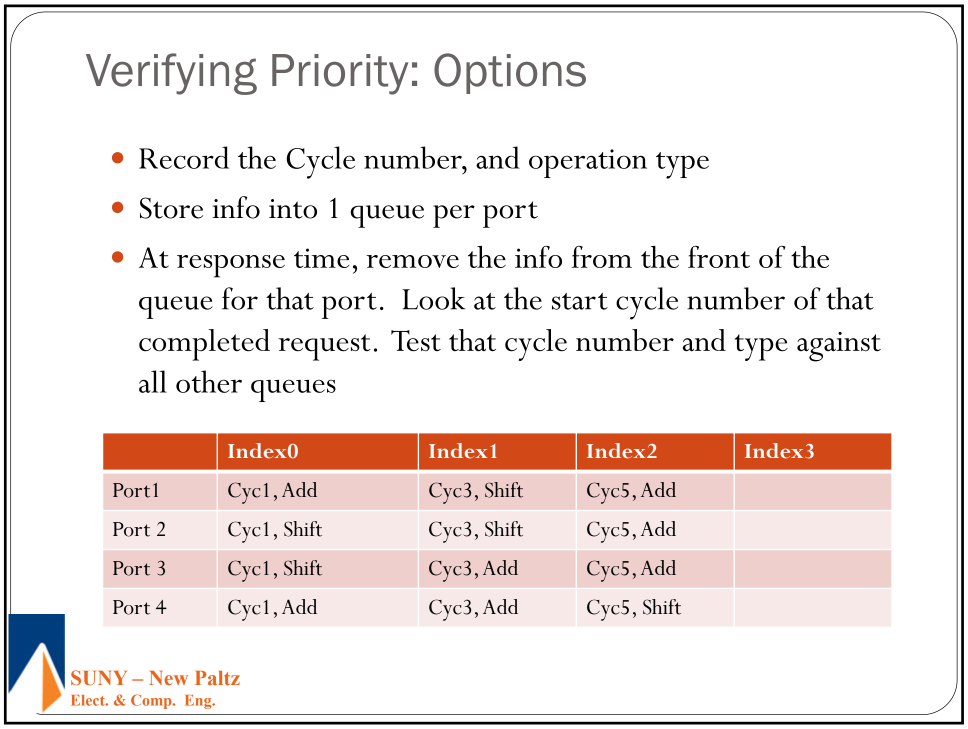


Verifying Priority

- How would you keep track of transactions, so you can say if things are fair?
  - What would you record about each transaction?

SUNY – New Paltz
Elect. & Comp. Eng.

39



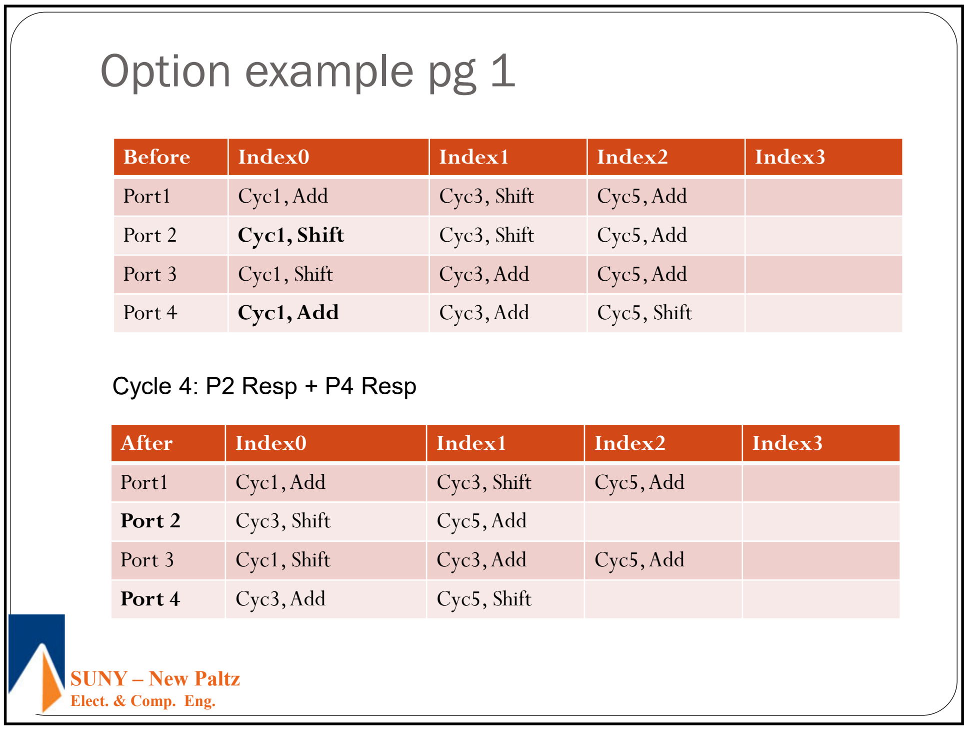
Verifying Priority: Options

- Record the Cycle number, and operation type
- Store info into 1 queue per port
- At response time, remove the info from the front of the queue for that port. Look at the start cycle number of that completed request. Test that cycle number and type against all other queues

	Index0	Index1	Index2	Index3
Port1	Cyc1, Add	Cyc3, Shift	Cyc5, Add	
Port 2	Cyc1, Shift	Cyc3, Shift	Cyc5, Add	
Port 3	Cyc1, Shift	Cyc3, Add	Cyc5, Add	
Port 4	Cyc1, Add	Cyc3, Add	Cyc5, Shift	

SUNY – New Paltz
Elect. & Comp. Eng.

40



Option example pg 1

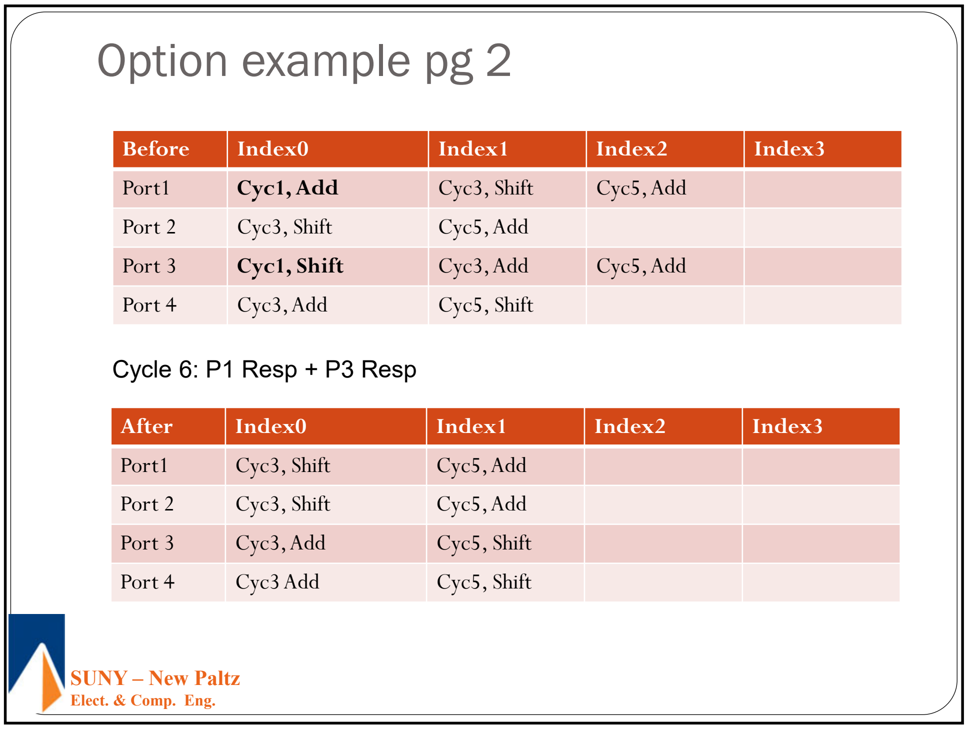
Before	Index0	Index1	Index2	Index3
Port1	Cyc1, Add	Cyc3, Shift	Cyc5, Add	
Port 2	**Cyc1, Shift**	Cyc3, Shift	Cyc5, Add	
Port 3	Cyc1, Shift	Cyc3, Add	Cyc5, Add	
Port 4	**Cyc1, Add**	Cyc3, Add	Cyc5, Shift	

Cycle 4: P2 Resp + P4 Resp

After	Index0	Index1	Index2	Index3
Port1	Cyc1, Add	Cyc3, Shift	Cyc5, Add	
Port 2	Cyc3, Shift	Cyc5, Add		
Port 3	Cyc1, Shift	Cyc3, Add	Cyc5, Add	
Port 4	Cyc3, Add	Cyc5, Shift		

SUNY – New Paltz
Elect. & Comp. Eng.

41



Option example pg 2

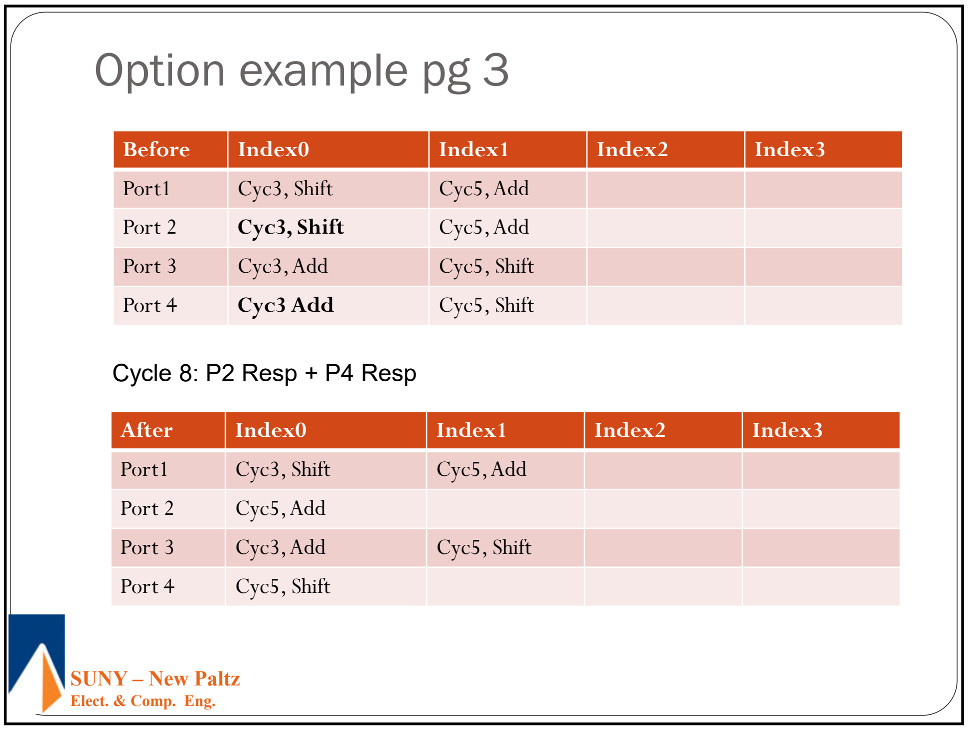
Before	Index0	Index1	Index2	Index3
Port1	**Cyc1, Add**	Cyc3, Shift	Cyc5, Add	
Port 2	Cyc3, Shift	Cyc5, Add		
Port 3	**Cyc1, Shift**	Cyc3, Add	Cyc5, Add	
Port 4	Cyc3, Add	Cyc5, Shift		

Cycle 6: P1 Resp + P3 Resp

After	Index0	Index1	Index2	Index3
Port1	Cyc3, Shift	Cyc5, Add		
Port 2	Cyc3, Shift	Cyc5, Add		
Port 3	Cyc3, Add	Cyc5, Shift		
Port 4	Cyc3 Add	Cyc5, Shift		

SUNY – New Paltz
Elect. & Comp. Eng.

42



Option example pg 3

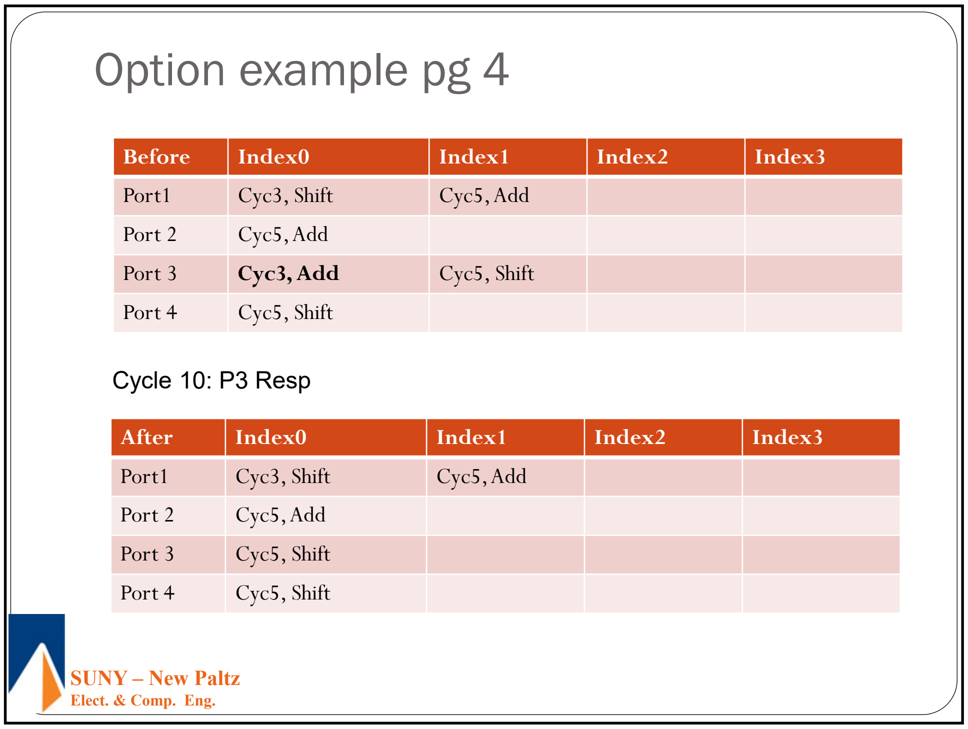
Before	Index0	Index1	Index2	Index3
Port1	Cyc3, Shift	Cyc5, Add		
Port 2	**Cyc3, Shift**	Cyc5, Add		
Port 3	Cyc3, Add	Cyc5, Shift		
Port 4	**Cyc3 Add**	Cyc5, Shift		

Cycle 8: P2 Resp + P4 Resp

After	Index0	Index1	Index2	Index3
Port1	Cyc3, Shift	Cyc5, Add		
Port 2	Cyc5, Add			
Port 3	Cyc3, Add	Cyc5, Shift		
Port 4	Cyc5, Shift			

SUNY – New Paltz
Elect. & Comp. Eng.

43



Option example pg 4

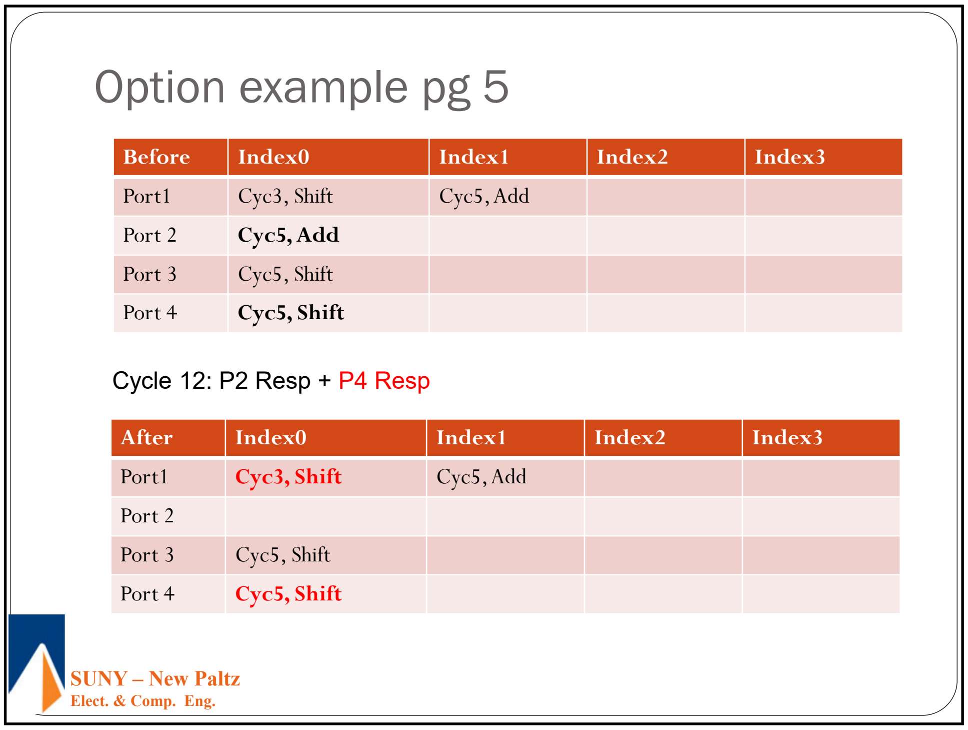
Before	Index0	Index1	Index2	Index3
Port1	Cyc3, Shift	Cyc5, Add		
Port 2	Cyc5, Add			
Port 3	**Cyc3, Add**	Cyc5, Shift		
Port 4	Cyc5, Shift			

Cycle 10: P3 Resp

After	Index0	Index1	Index2	Index3
Port1	Cyc3, Shift	Cyc5, Add		
Port 2	Cyc5, Add			
Port 3	Cyc5, Shift			
Port 4	Cyc5, Shift			

SUNY – New Paltz
Elect. & Comp. Eng.

44



Option example pg 5

Before	Index0	Index1	Index2	Index3
Port1	Cyc3, Shift	Cyc5, Add		
Port 2	**Cyc5, Add**			
Port 3	Cyc5, Shift			
Port 4	**Cyc5, Shift**			

Cycle 12: P2 Resp + P4 Resp

After	Index0	Index1	Index2	Index3
Port1	**Cyc3, Shift**	Cyc5, Add		
Port 2				
Port 3	Cyc5, Shift			
Port 4	**Cyc5, Shift**			

SUNY – New Paltz
Elect. & Comp. Eng.

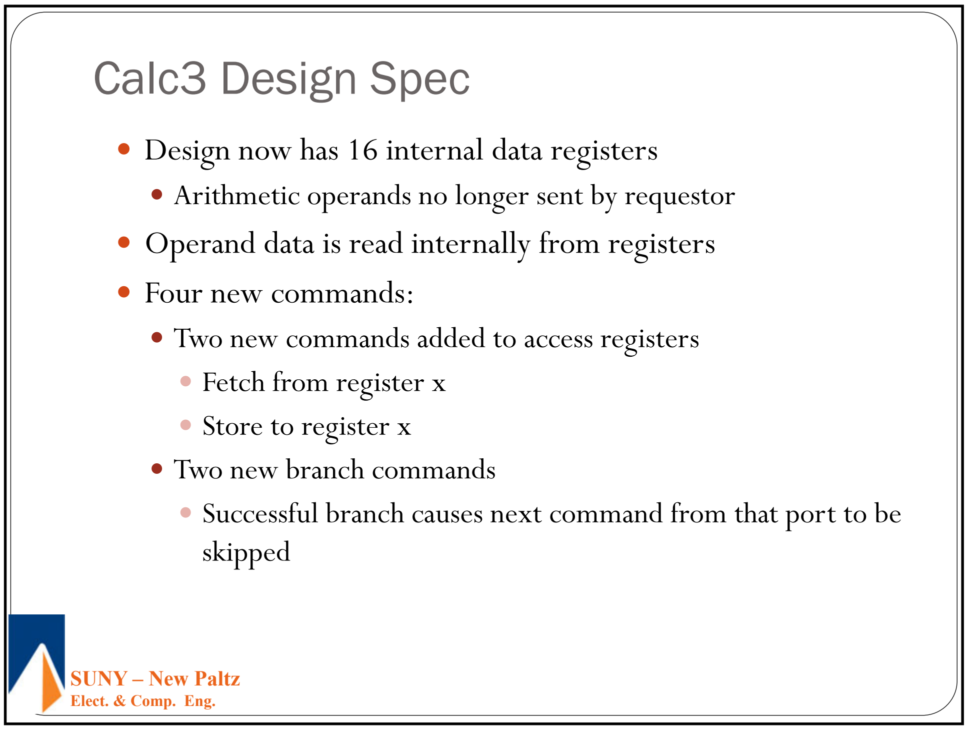
45

Calc3

Now it acts like a CPU

SUNY – New Paltz
Division of Engineering Programs

46

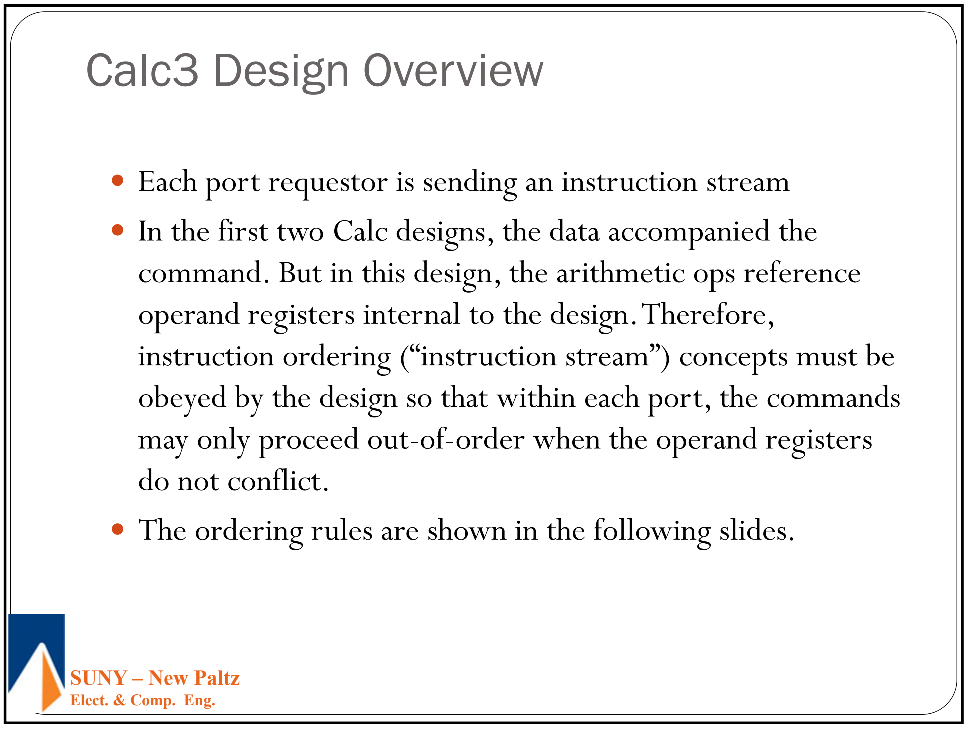


Calc3 Design Spec

- Design now has 16 internal data registers
  - Arithmetic operands no longer sent by requestor
- Operand data is read internally from registers
- Four new commands:
  - Two new commands added to access registers
    - Fetch from register x
    - Store to register x
  - Two new branch commands
    - Successful branch causes next command from that port to be skipped

SUNY – New Paltz
Elect. & Comp. Eng.

47



Calc3 Design Overview

- Each port requestor is sending an instruction stream
- In the first two Calc designs, the data accompanied the command. But in this design, the arithmetic ops reference operand registers internal to the design. Therefore, instruction ordering ("instruction stream") concepts must be obeyed by the design so that within each port, the commands may only proceed out-of-order when the operand registers do not conflict.
- The ordering rules are shown in the following slides.

SUNY – New Paltz
Elect. & Comp. Eng.

48

I/O Specification

For each requestor X:
Inputs:

reqX_cmd(0:3)
- 0001 add: adds contents of d1 to d2 and stores in r1
- 0010 subtract: subtracts contents of d2 from d1 and stores in r1
- 0101 shift left: shifts contents of d1 to the left d2(27:31) places and stores in r1
- 0110 shift right: shifts contents of d1 to the right d2(27:31) places and stores in r1
- 1001 store: stores reqX_data(0:31) into r1
- 1010 fetch: fetches contents of d1 and outputs it on out_dataX(0:31)
- 1100 branch if zero: skip next valid command if contents of d1 are 0
- 1101 branch if equal: skip next valid command if contents of d1 and d2 are equal

reqX_d1(0:3) - operand register to read
reqX_d2(0:3) - operand register to read
reqX_r1(0:3) - operand register to write
reqX_tag(0:1)
reqX_data(0:31)

Outputs:
outX_resp(0:1)
- 00: No Response
- 01: Successful completion
- 10: overflow/underflow error
- 11 : Command skipped due to branch

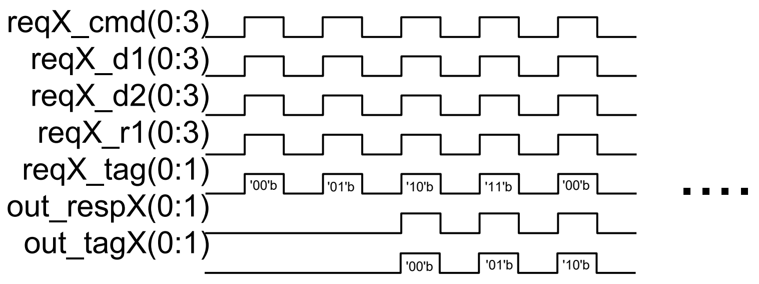
outX_tag(0:1)
outX_data(0:31)

SUNY – New Paltz
Elect. & Comp. Eng.

49

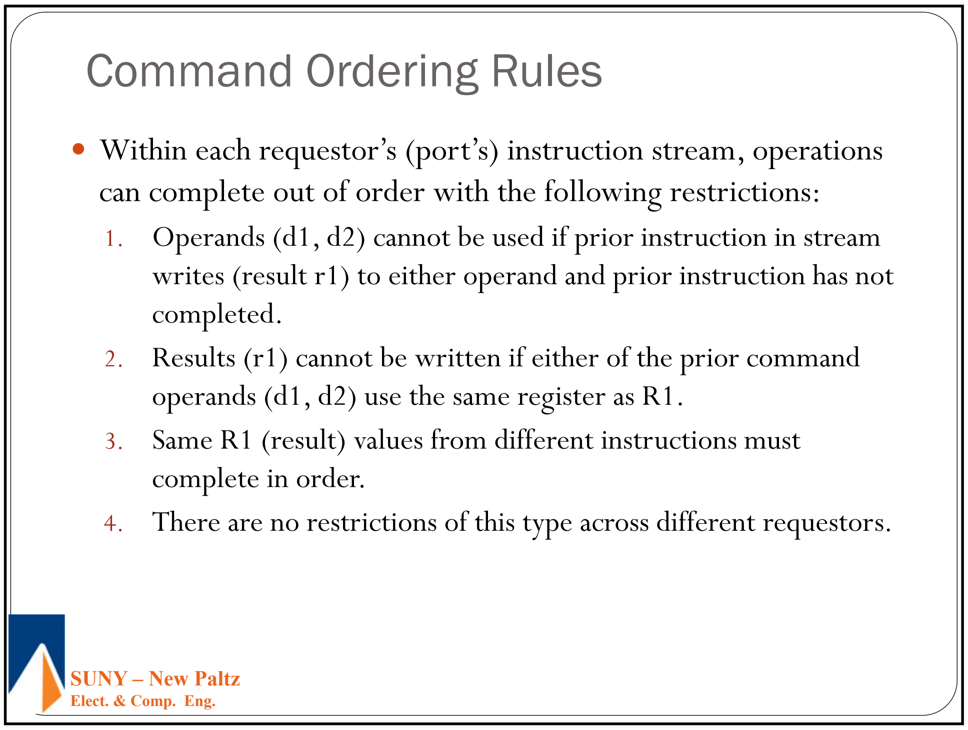
I/O Timing

- Fastest multiple command (any cmd type) timing (example is if only one requestor is sending commands):



SUNY – New Paltz
Elect. & Comp. Eng.

50

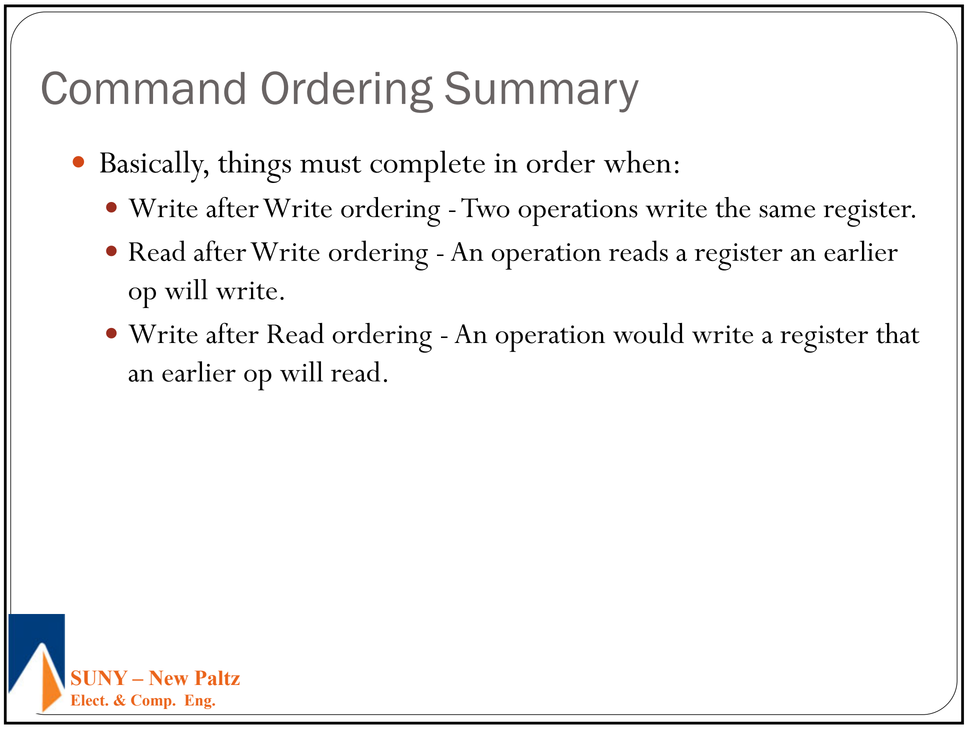


Command Ordering Rules

- Within each requestor's (port's) instruction stream, operations can complete out of order with the following restrictions:
  1. Operands (d1, d2) cannot be used if prior instruction in stream writes (result r1) to either operand and prior instruction has not completed.
  2. Results (r1) cannot be written if either of the prior command operands (d1, d2) use the same register as R1.
  3. Same R1 (result) values from different instructions must complete in order.
  4. There are no restrictions of this type across different requestors.

SUNY – New Paltz
Elect. & Comp. Eng.

51



Command Ordering Summary

- Basically, things must complete in order when:
  - Write after Write ordering - Two operations write the same register.
  - Read after Write ordering - An operation reads a register an earlier op will write.
  - Write after Read ordering - An operation would write a register that an earlier op will read.

SUNY – New Paltz
Elect. & Comp. Eng.

52

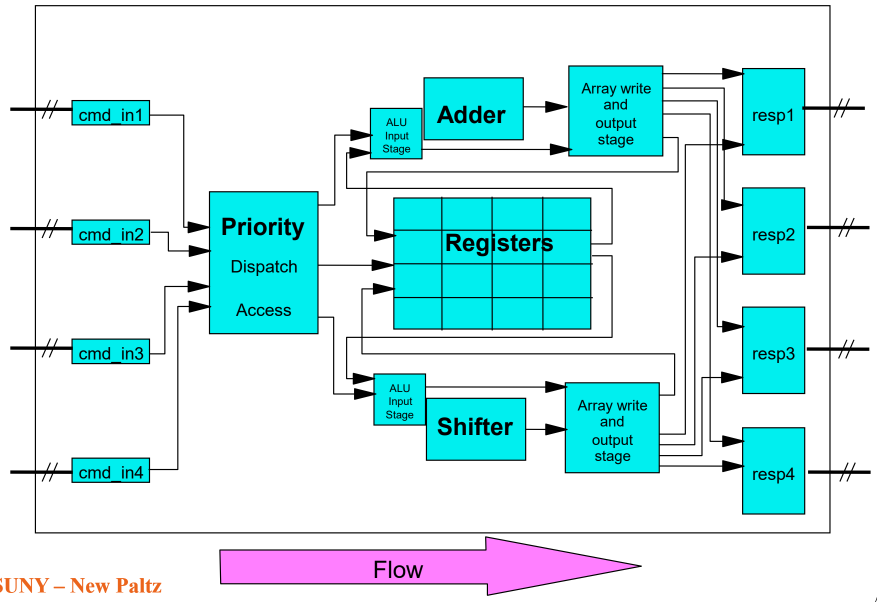
Commands that Follow a Branch

- Any valid command can follow a branch.
- If the branch evaluates true, the following command will be "skipped":
  - Add/Sub/SL/SR will not write to array
  - Store will not write to array
  - Fetch will not return data
  - Branch will evaluate to false (case of branch followed by branch)
- Response code of '11'b for follower indicating above action has occurred.
- Invalid OP codes are ignored and are NOT considered to "command that follows a branch."

SUNY – New Paltz
Elect. & Comp. Eng.

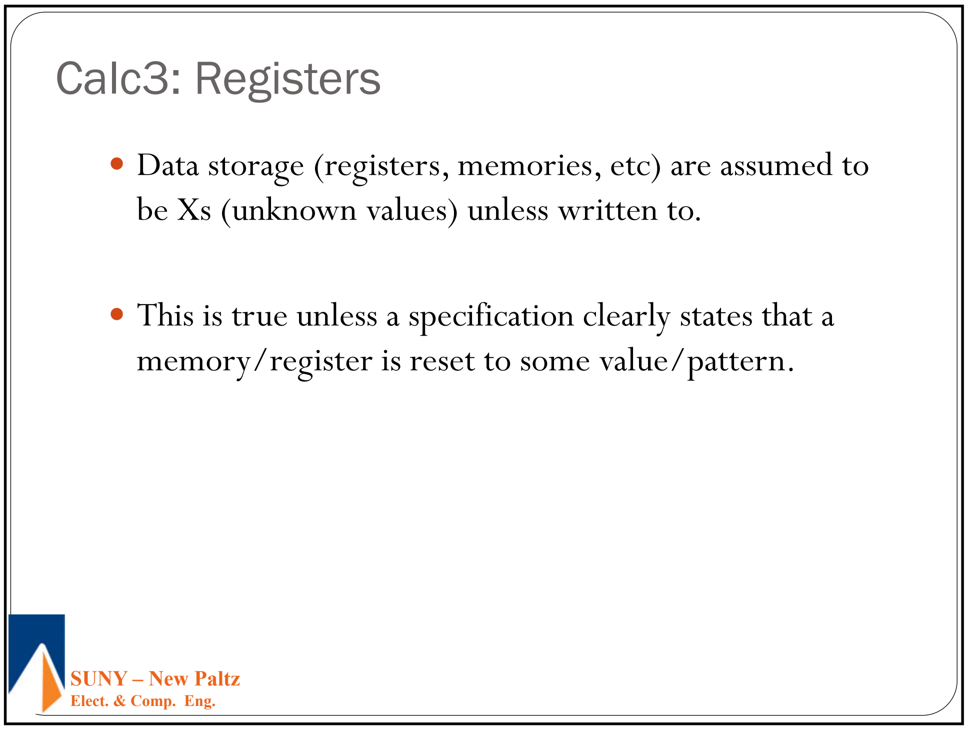
53

Calc3 Block Diagram



SUNY – New Paltz
Elect. & Comp. Eng.

54

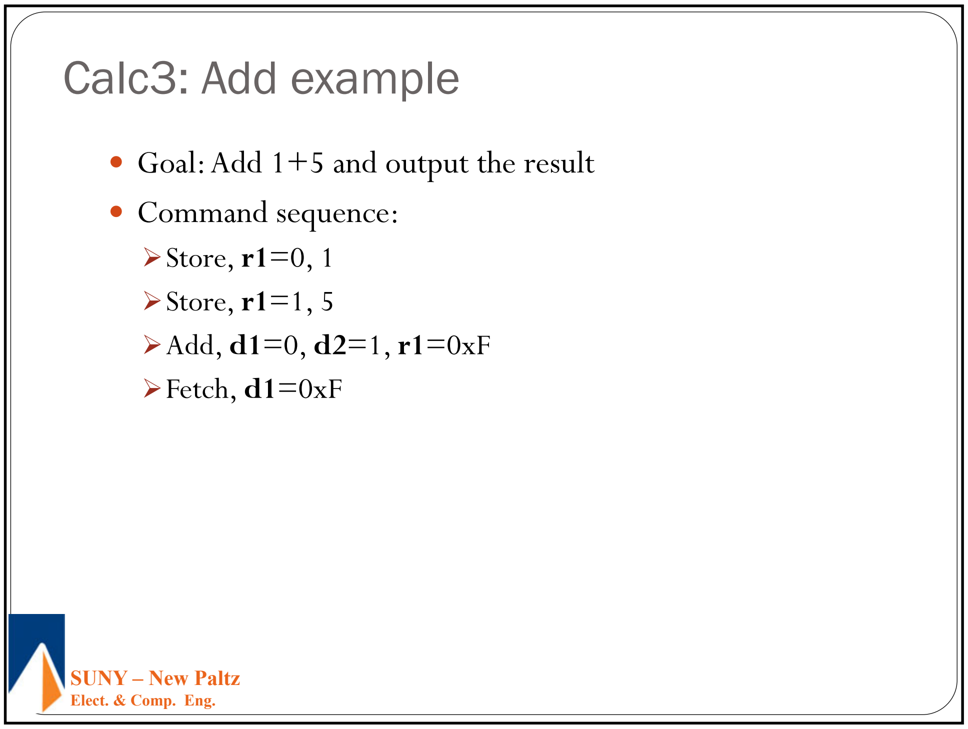


Calc3: Registers

- Data storage (registers, memories, etc) are assumed to be Xs (unknown values) unless written to.
- This is true unless a specification clearly states that a memory/register is reset to some value/pattern.

SUNY – New Paltz
Elect. & Comp. Eng.

55

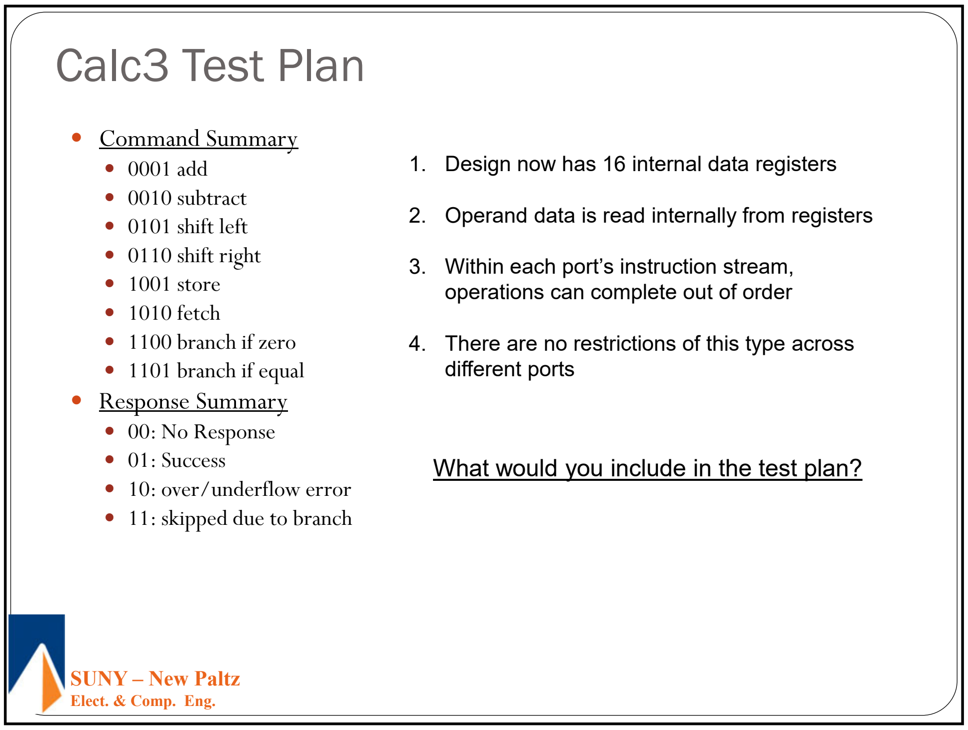


Calc3: Add example

- Goal: Add 1+5 and output the result
- Command sequence:
  - Store, **r1**=0, 1
  - Store, **r1**=1, 5
  - Add, **d1**=0, **d2**=1, **r1**=0xF
  - Fetch, **d1**=0xF

SUNY – New Paltz
Elect. & Comp. Eng.

56



Calc3 Test Plan

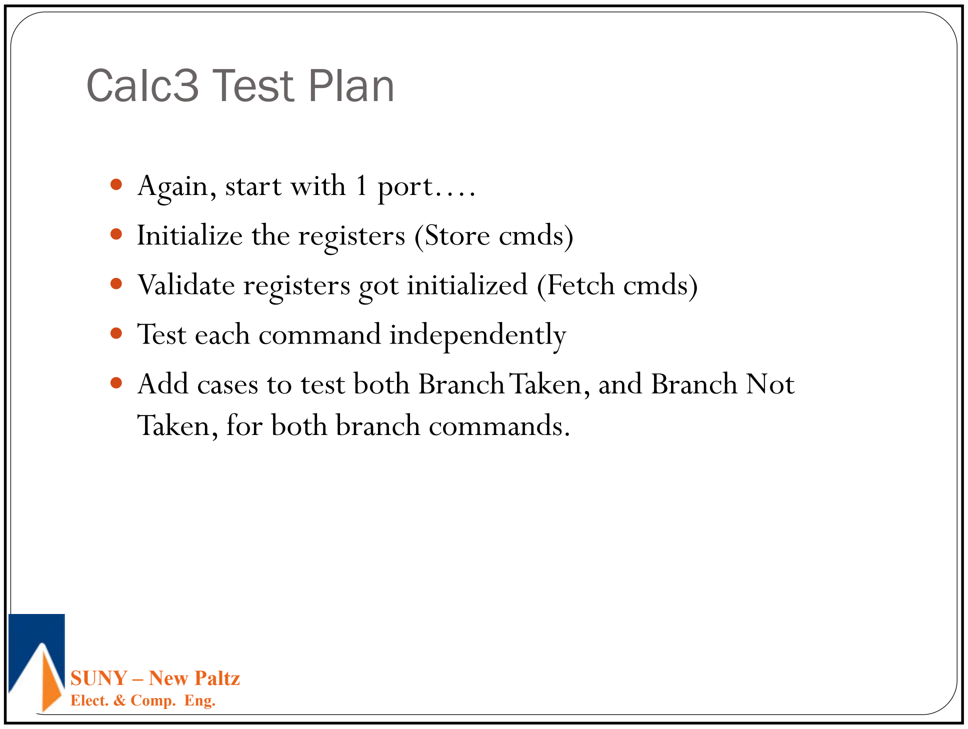
- Command Summary
  - 0001 add
  - 0010 subtract
  - 0101 shift left
  - 0110 shift right
  - 1001 store
  - 1010 fetch
  - 1100 branch if zero
  - 1101 branch if equal
- Response Summary
  - 00: No Response
  - 01: Success
  - 10: over/underflow error
  - 11: skipped due to branch

1. Design now has 16 internal data registers
2. Operand data is read internally from registers
3. Within each port's instruction stream, operations can complete out of order
4. There are no restrictions of this type across different ports

What would you include in the test plan?

SUNY – New Paltz
Elect. & Comp. Eng.

57



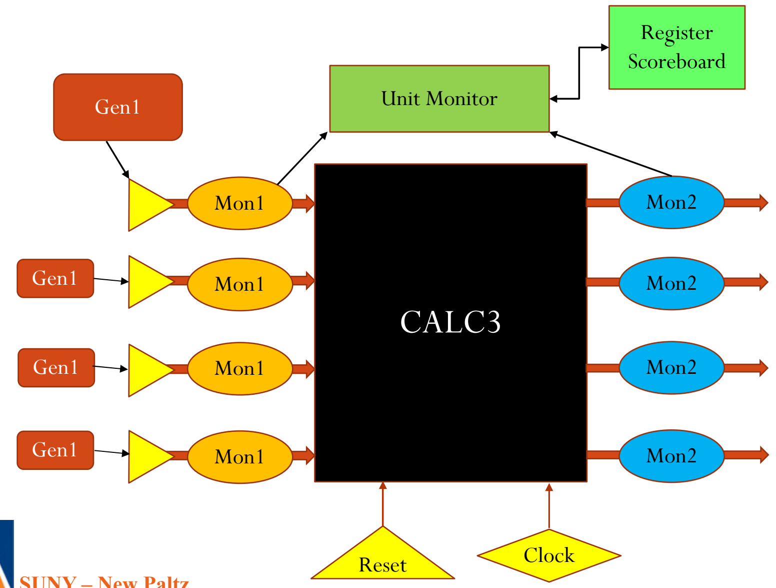
Calc3 Test Plan

- Again, start with 1 port….
- Initialize the registers (Store cmds)
- Validate registers got initialized (Fetch cmds)
- Test each command independently
- Add cases to test both Branch Taken, and Branch Not Taken, for both branch commands.

SUNY – New Paltz
Elect. & Comp. Eng.

58

Calc3 Environment



SUNY – New Paltz
Elect. & Comp. Eng.

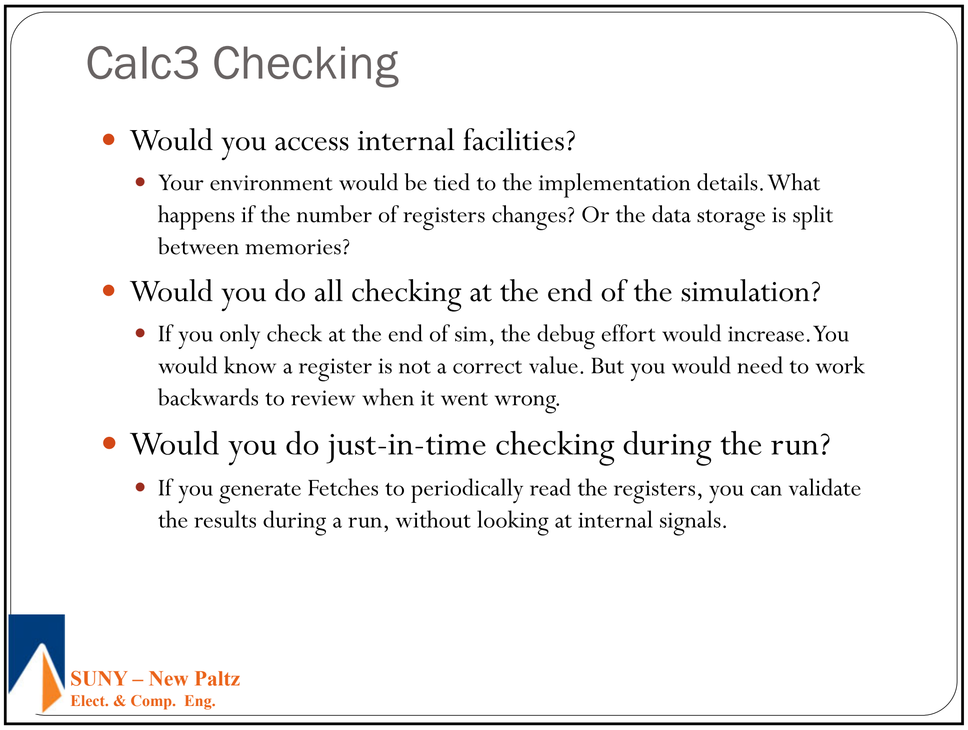
59

Calc3 Checking

- How would you check the result of an operation?
  - Would you access internal facilities?
  - Would you do all checking at the end of the simulation?
  - Would you do just-in-time checking during the run?

SUNY – New Paltz
Elect. & Comp. Eng.

60

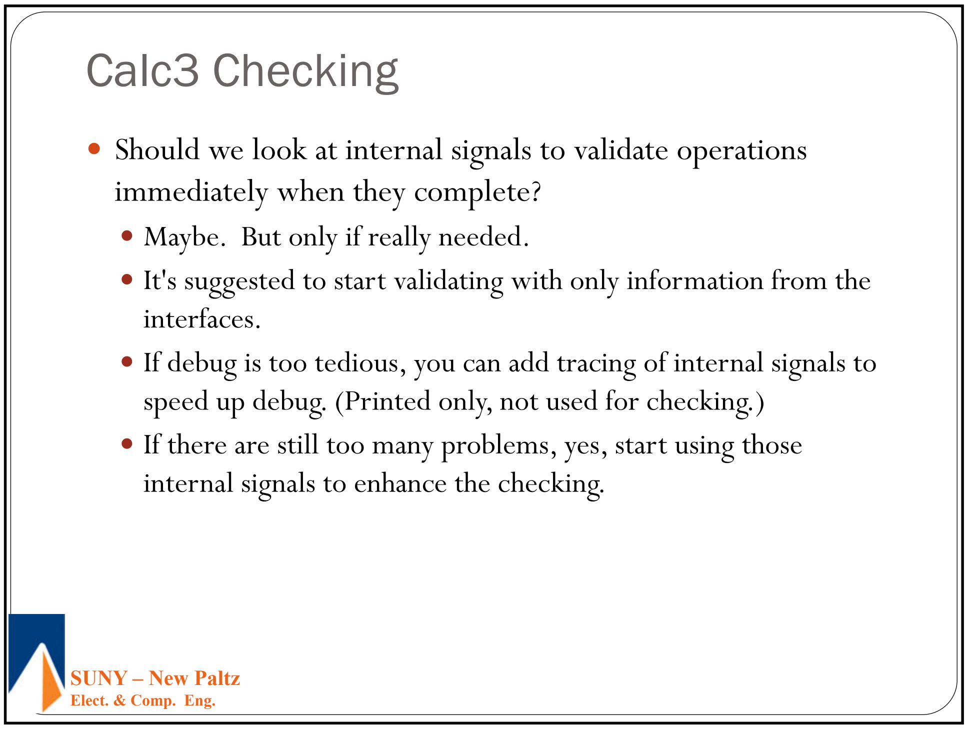


Calc3 Checking

- Would you access internal facilities?
  - Your environment would be tied to the implementation details. What happens if the number of registers changes? Or the data storage is split between memories?
- Would you do all checking at the end of the simulation?
  - If you only check at the end of sim, the debug effort would increase. You would know a register is not a correct value. But you would need to work backwards to review when it went wrong.
- Would you do just-in-time checking during the run?
  - If you generate Fetches to periodically read the registers, you can validate the results during a run, without looking at internal signals.

SUNY – New Paltz
Elect. & Comp. Eng.

61



Calc3 Checking

- Should we look at internal signals to validate operations immediately when they complete?
  - Maybe. But only if really needed.
  - It's suggested to start validating with only information from the interfaces.
  - If debug is too tedious, you can add tracing of internal signals to speed up debug. (Printed only, not used for checking.)
  - If there are still too many problems, yes, start using those internal signals to enhance the checking.

SUNY – New Paltz
Elect. & Comp. Eng.

62

Calc3 Checking: Ordering Rules

- What do you need to store to validate the ordering rules?
  - Pretty much everything: Cycle, Cmd, d1, d2, r1, tag

	Index0	Index1	Index2
Port1	Cyc1, Add, 0, 1, 2, T0	Cyc3, Shift, 0,1, 3, T1	Cyc5, Add, 0, 4, 5, T2
Port 2	Cyc3, Store, 15, 0xFC, T1	Cyc5, Fetch, 15, T0	
Port 3	Cyc1, Add, 12, 13, 14, T0		
Port 4	Cyc1, Add, 4, 5, 6, T1	Cyc3, Add, 6, 7, 8, T3	Cyc5, Shift, 9, 10, 11, T0

SUNY – New Paltz
Elect. & Comp. Eng.

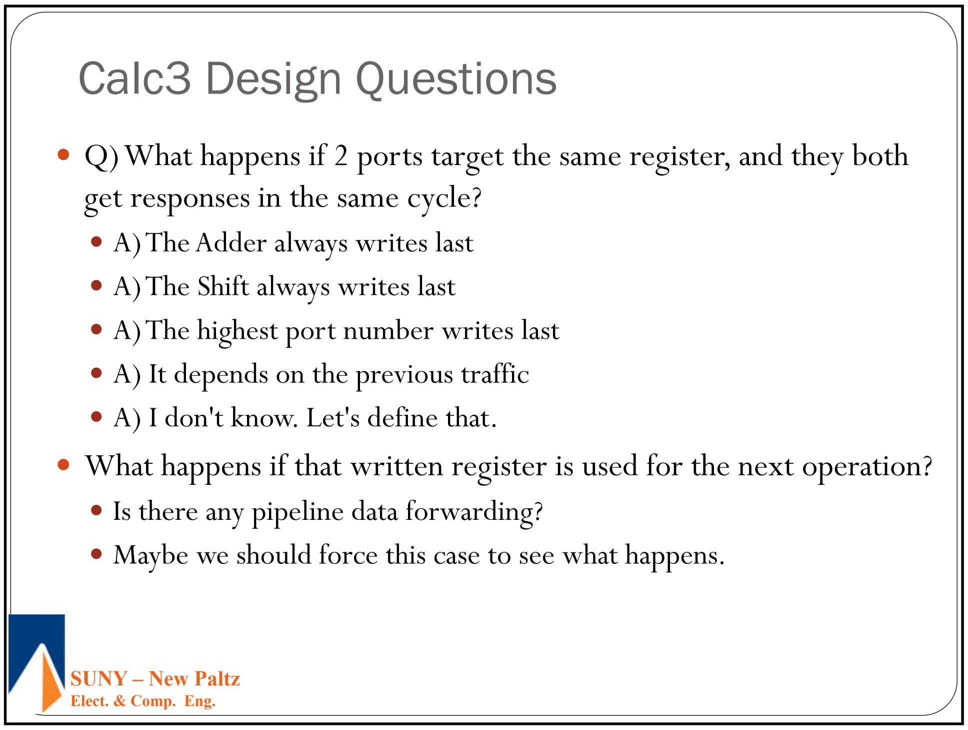
63

Calc3 Checking: Branches

- How can you handle checking the branch command function?
  - Commands after a branch Taken, will be skipped.
  - Skipped commands get a response of '11'
- What if no command is in stream for that port when the branch completes?

SUNY – New Paltz
Elect. & Comp. Eng.

64

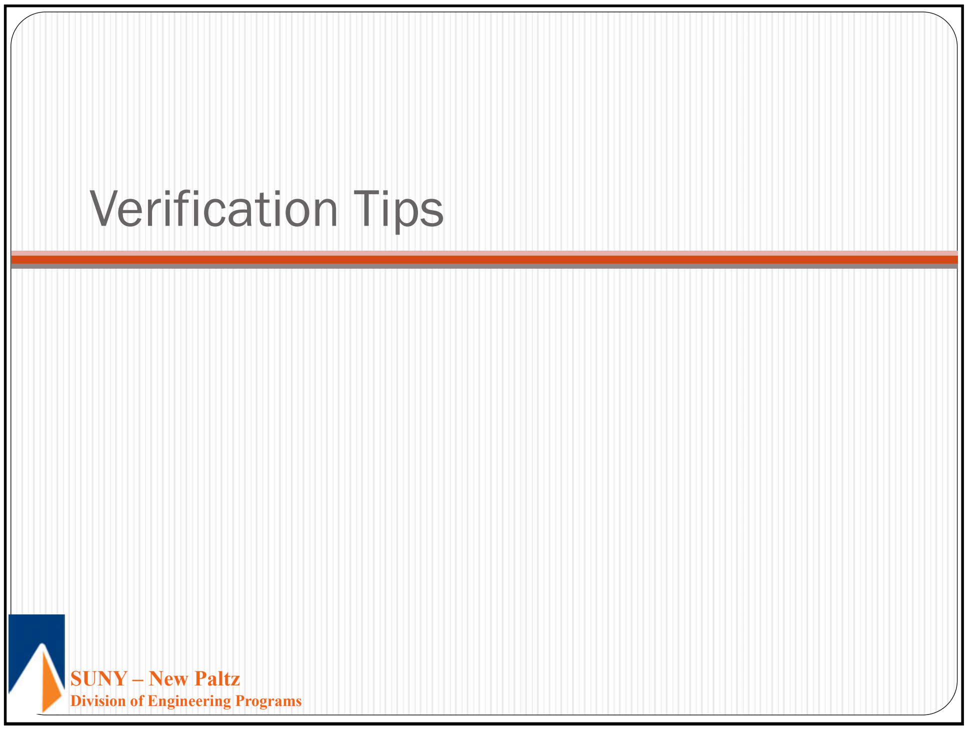


Calc3 Design Questions

- Q) What happens if 2 ports target the same register, and they both get responses in the same cycle?
  - A) The Adder always writes last
  - A) The Shift always writes last
  - A) The highest port number writes last
  - A) It depends on the previous traffic
  - A) I don't know. Let's define that.
- What happens if that written register is used for the next operation?
  - Is there any pipeline data forwarding?
  - Maybe we should force this case to see what happens.

SUNY – New Paltz
Elect. & Comp. Eng.

65



Verification Tips

SUNY – New Paltz
Division of Engineering Programs

66

Question the Specification

- Push back on the specification writer if needed
  - Get clarity on ANYTHING ambiguous
  - Are un-used signals required to be 0 while other parts of the bus are valid?
  - Are signals required to be 0 between valid operations?
- If there is a failure due to ambiguity in the spec, it's largely a Verification Miss.
- Question Everything.

SUNY – New Paltz
Elect. & Comp. Eng.

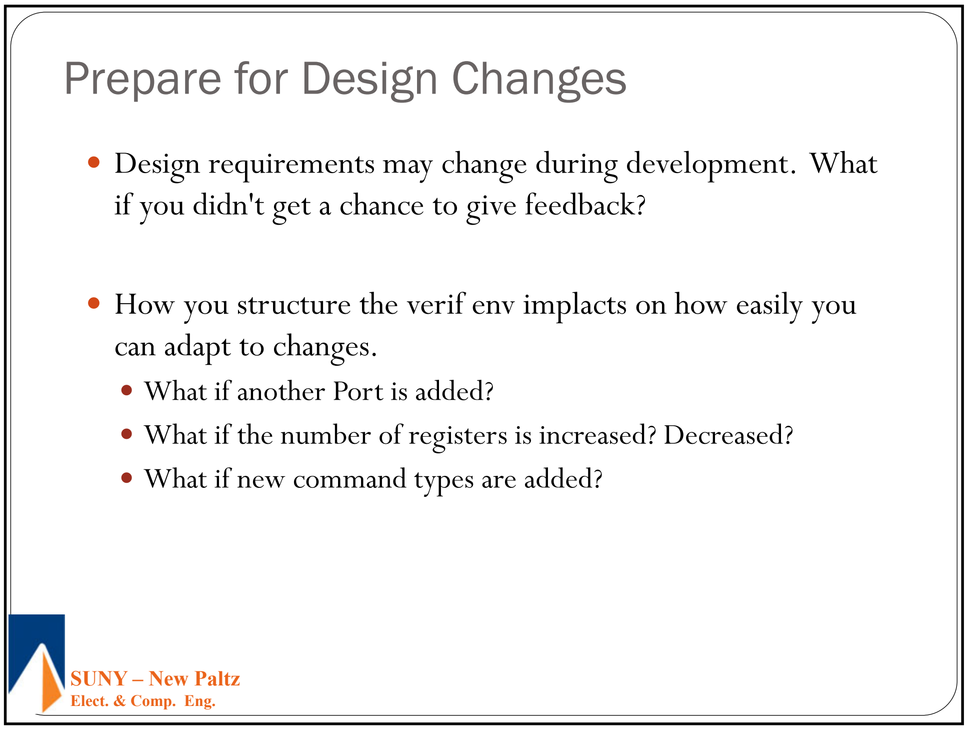
67

Feedback to Designers

- Verification should be involved in design decisions.
- "Design For Verification"
- A small design change may have an enormous impact on the verif env. Giving that feedback after the change is implemented is too late.

SUNY – New Paltz
Elect. & Comp. Eng.

68



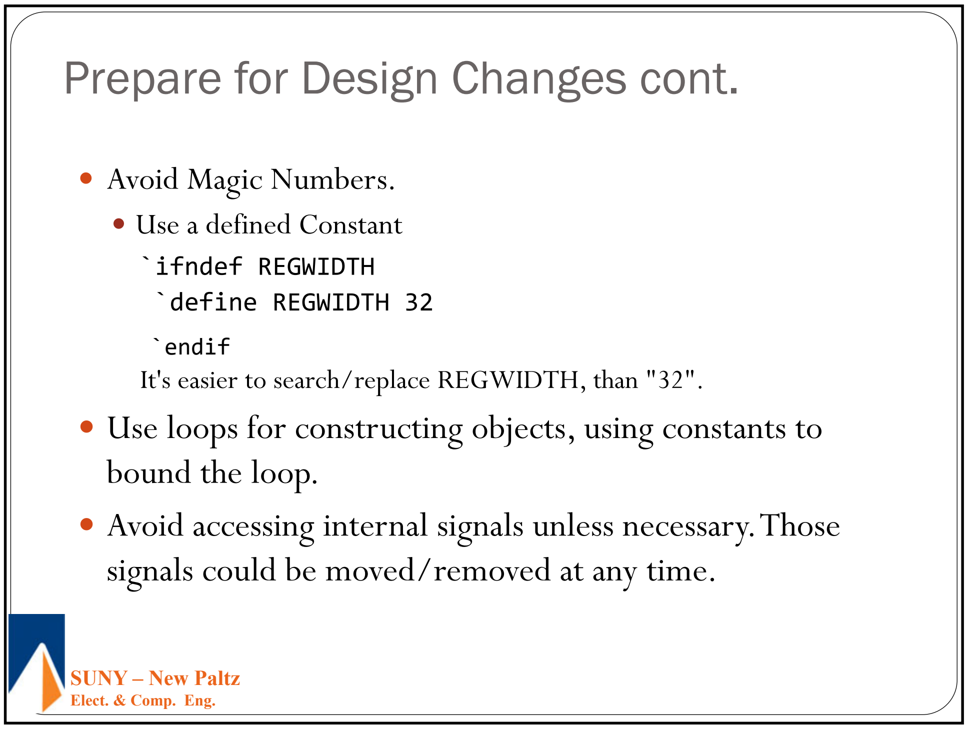
Prepare for Design Changes

- Design requirements may change during development. What if you didn't get a chance to give feedback?

- How you structure the verif env implacts on how easily you can adapt to changes.
  - What if another Port is added?
  - What if the number of registers is increased? Decreased?
  - What if new command types are added?

SUNY – New Paltz
Elect. & Comp. Eng.

69



Prepare for Design Changes cont.

- Avoid Magic Numbers.
  - Use a defined Constant

```
`ifndef REGWIDTH
  `define REGWIDTH 32

 `endif
```

It's easier to search/replace REGWIDTH, than "32".

- Use loops for constructing objects, using constants to bound the loop.
- Avoid accessing internal signals unless necessary. Those signals could be moved/removed at any time.

SUNY – New Paltz
Elect. & Comp. Eng.

70

Asynchronous signals

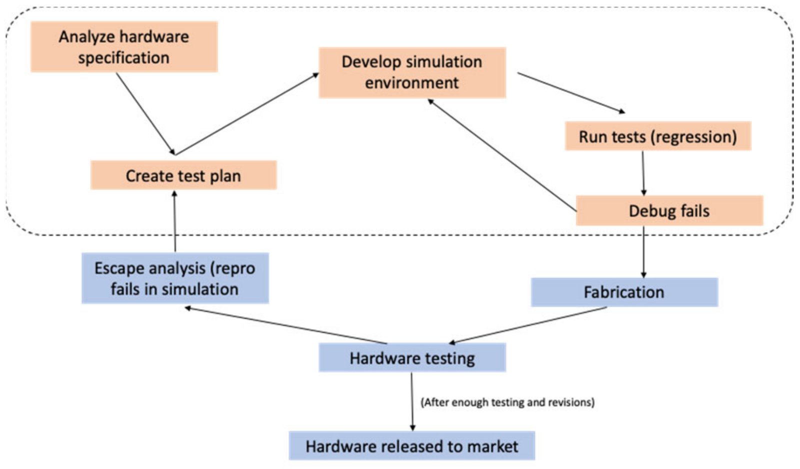
- A design which has signals crossing clock domains (different source oscillators)
- A design with inputs which are not governed by any clock

- All signals of this type should be clearly identified.
- Special testing is required to model the shifting alignment the async signals may have in the final product
- This may even require a different simulator.

SUNY – New Paltz
Elect. & Comp. Eng.

71

Verification cycle


Analyze hardware specification
Develop simulation environment
Run tests (regression)
Create test plan
Debug fails
Escape analysis (repro fails in simulation
Fabrication
Hardware testing
(After enough testing and revisions)
Hardware released to market

SUNY – New Paltz
Elect. & Comp. Eng.

72