Calc Project Overview

Design Specs

Copyright Wile, Roesner, Goss

Overview of Calc desgins

- A series of three designs:
  - Calc1
  - Calc2
  - Calc3
- These designs were provided by IBM: Thanks to: Bruce Wile, John C. Goss, Wolfgang Roesner

Copyright Wile, Roesner, Goss

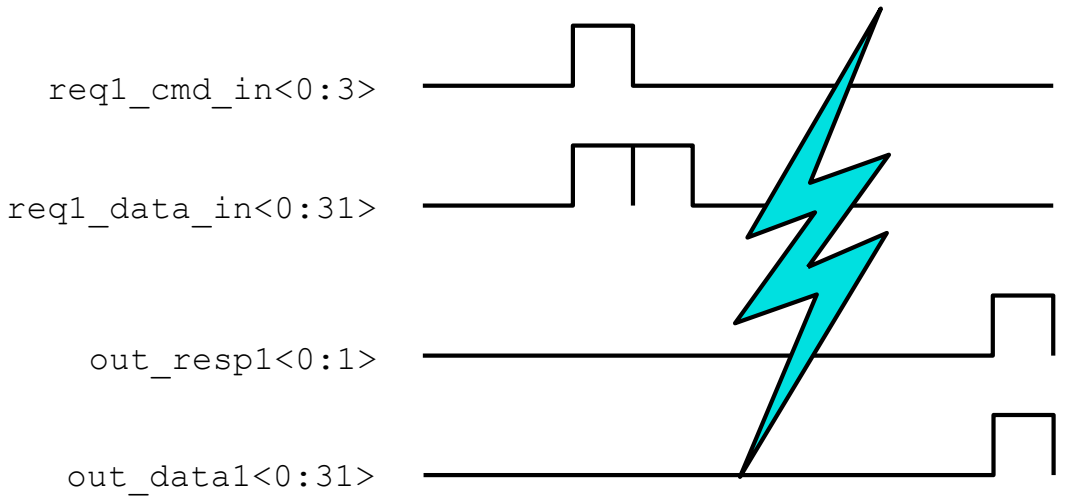
Calc1

- Calc1 is a simple 4-port calculator.
- Each port can send in 1 command at a time.
- The commands consist of add, subtract, shift left, and shift right.
- The operand data accompanies the commands.
- Since there are 4 ports, there can be up to 4 outstanding commands at once.

Copyright Wile, Roesner, Goss

Calc1

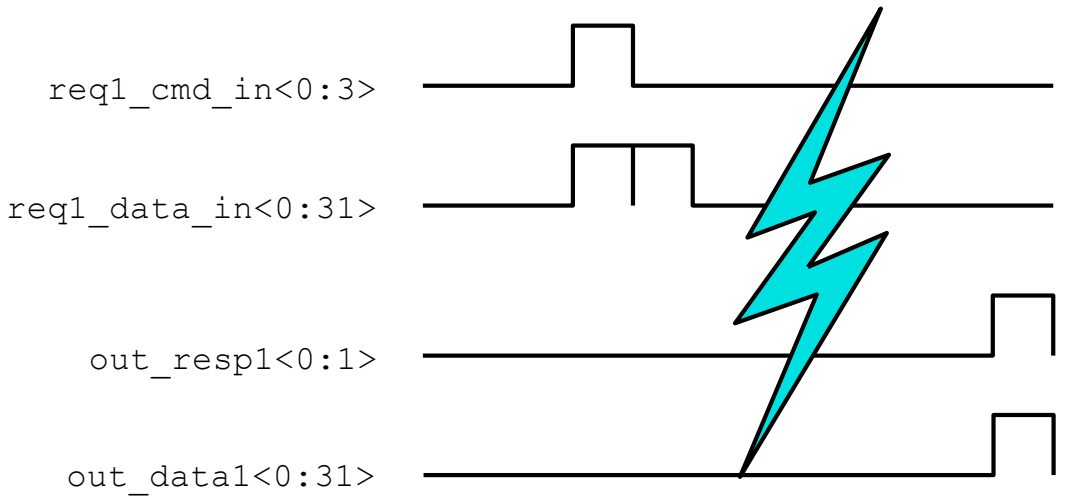
- The input/Output timing on a single port is as follows:



Copyright Wile, Roesner, Goss

Calc1

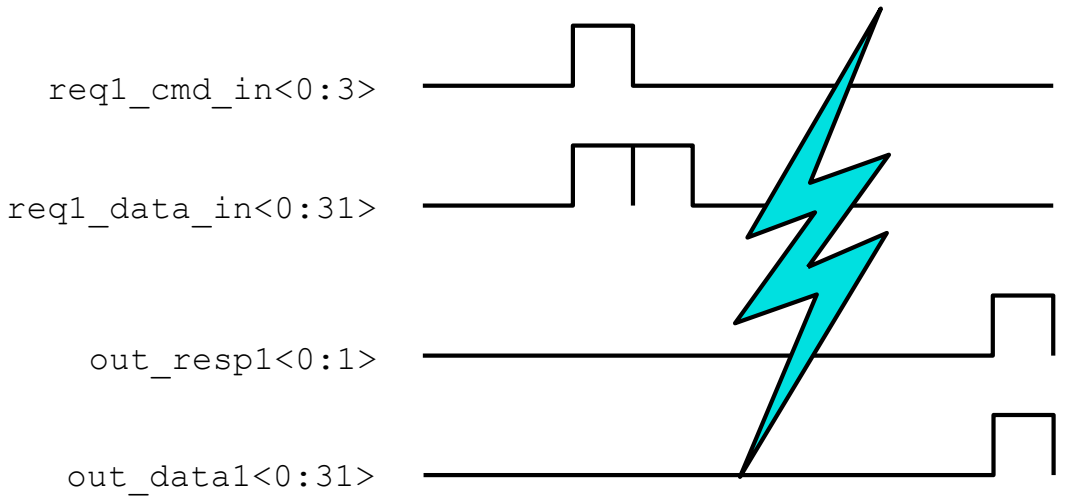
- The lightning bolt represents “some number of cycles passing.”



Copyright Wile, Roesner, Goss

Calc1

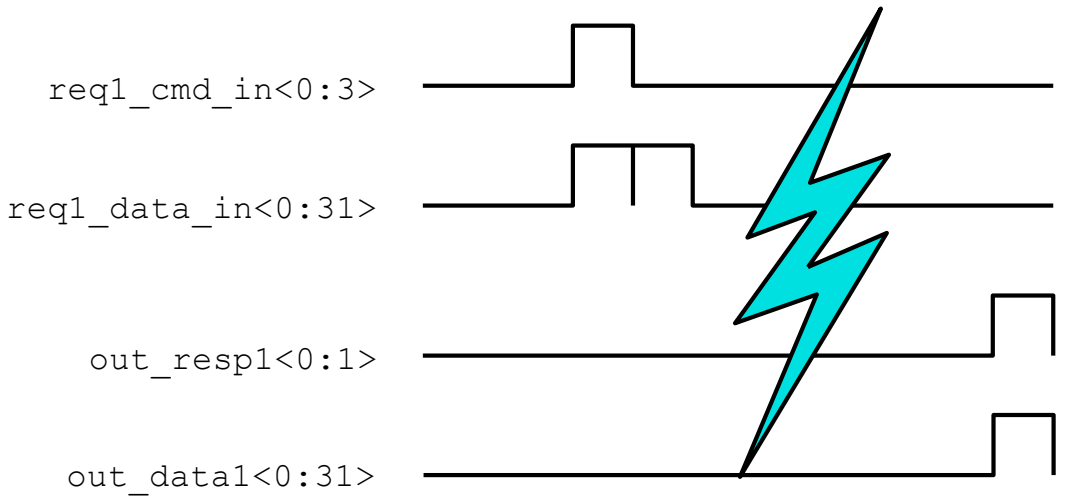
- A second request from the same port is prohibited until the response from the first command is received.



Copyright Wile, Roesner, Goss

Calc1

- Internally, there are two ALUs: one that processes all add commands, and the second for all shift commands. Priority logic dispatches the individual commands to the ALUs.



Copyright Wile, Roesner, Goss

Calc2

- The second design, Calc2, is much like the first, except that each port may now have up to 4 outstanding commands at a time.

Copyright Wile, Roesner, Goss

Calc2

- Each command from a port is sent in serially (one at a time), but the calc log may respond “out of order,” depending upon the internal state of the queues.

Copyright Wile, Roesner, Goss

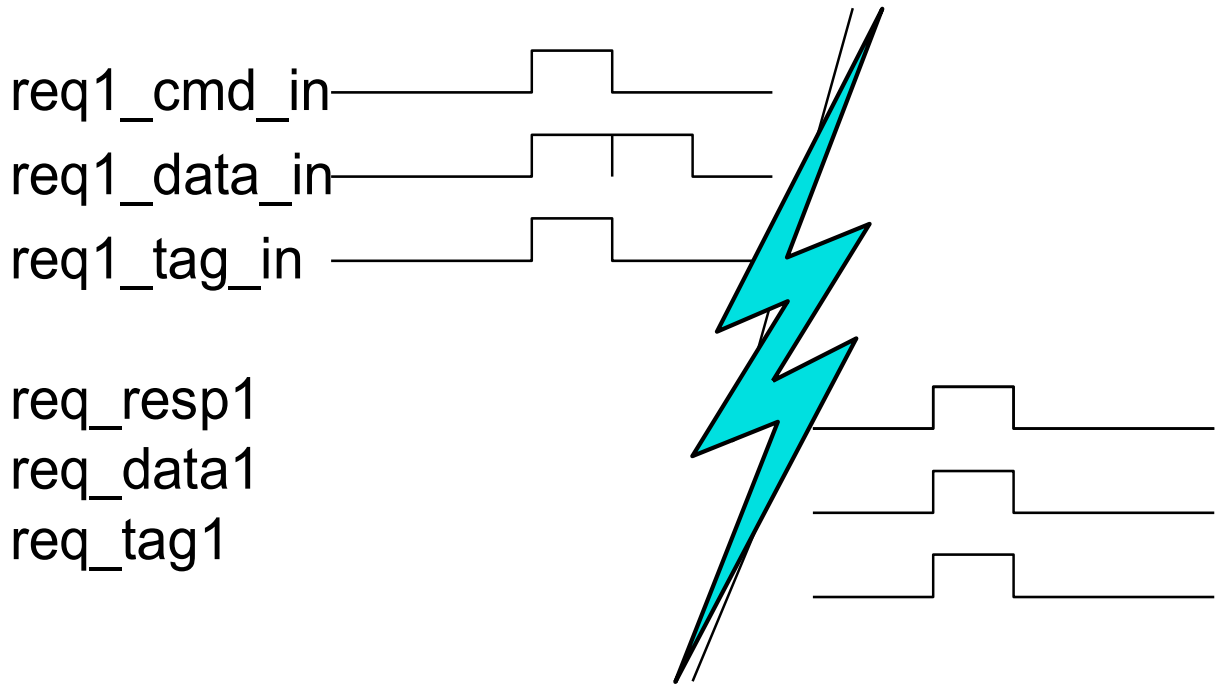
Calc2

- As a result, each command is now accompanied by a 2-bit tag, which identifies the command when the response is received.

Copyright Wile, Roesner, Goss

Calc2

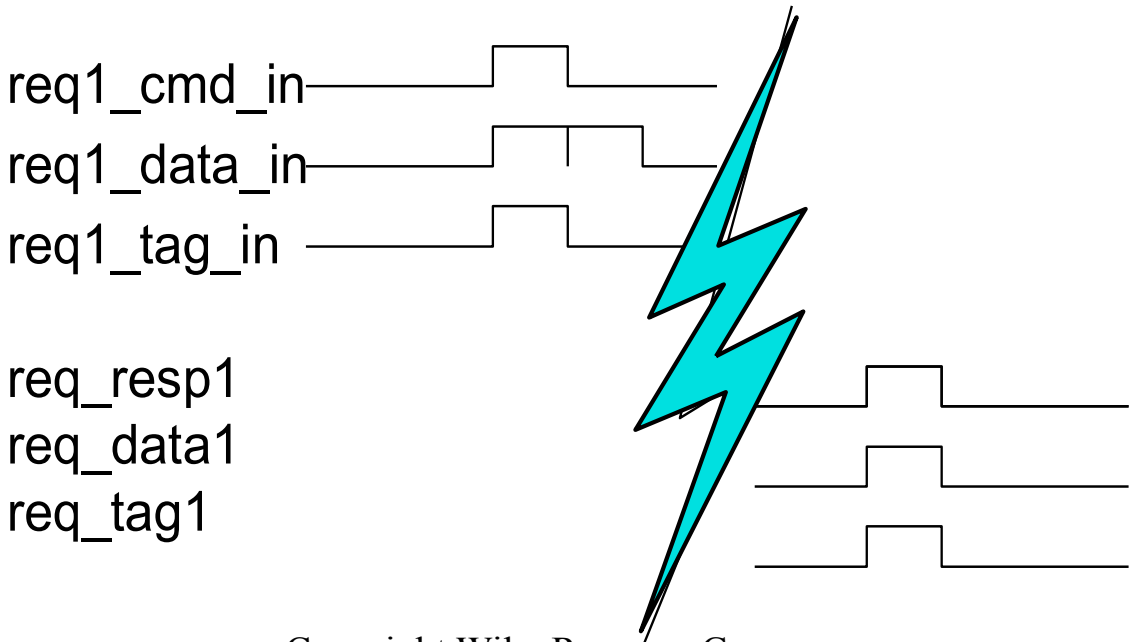
- A possible timing diagram:



Copyright Wile, Roesner, Goss

Calc2

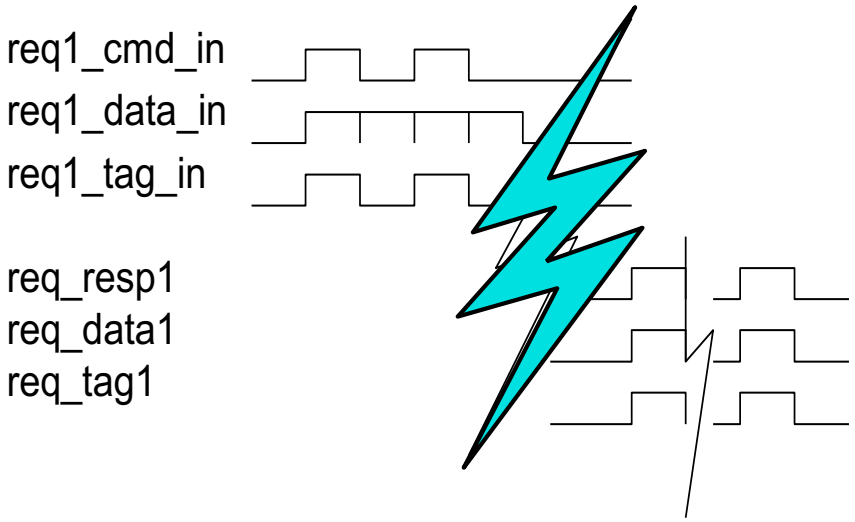
- The lightning bolt represents “some number of cycles passing.” The data accompanies a successful response signal.



Copyright Wile, Roesner, Goss

Calc2

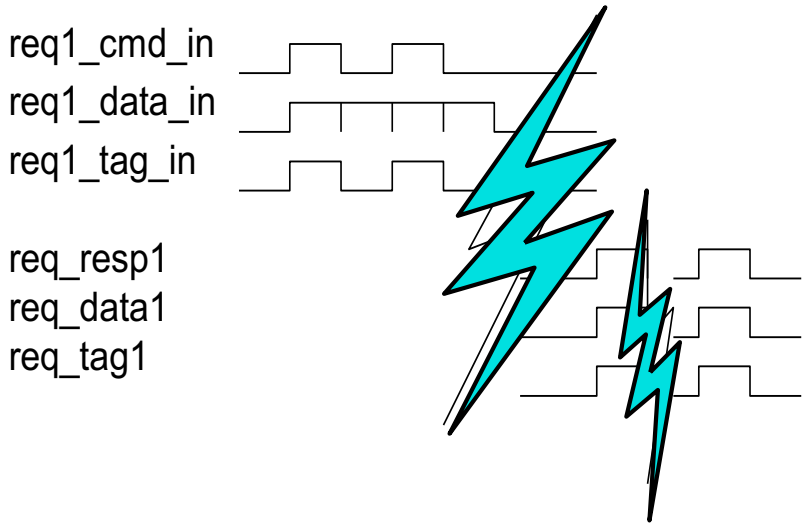
- A timing diagram of back-to-back commands might look like:



Copyright Wile, Roesner, Goss

Calc2

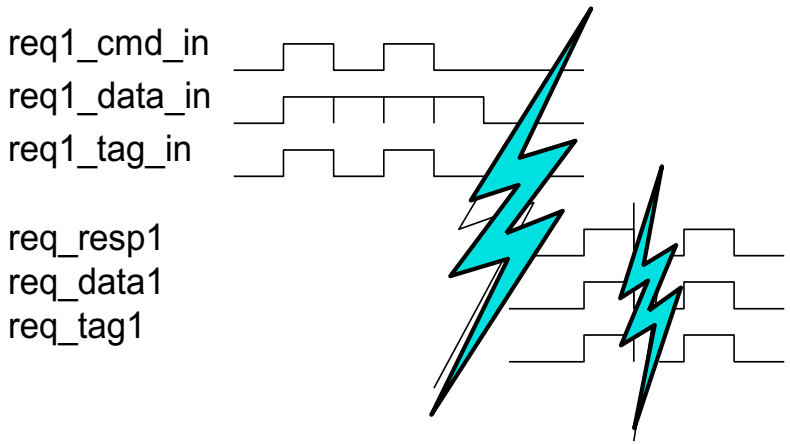
- The lightning bolt represents “some number of cycles passing.”



Copyright Wile, Roesner, Goss

Calc2

- The data accompanies a successful response signal. Responses may or may not be in order, but the tags will identify which command the response is for.



Copyright Wile, Roesner, Goss

Calc2

- There may be up to 16 commands outstanding at once (4 ports, 4 commands each).
- As in Calc1, the commands are add, subtract, shift left, and shift right.

Copyright Wile, Roesner, Goss

Calc3 Overview

- Design now has 16 internal data registers
  - Arithmetic operands no longer sent by requestor
  - Operand data is read internally from registers

Copyright Wile, Roesner, Goss

Calc3 Overview

- Two new commands added to access registers
  - Fetch from register x
  - Store to register x

Copyright Wile, Roesner, Goss

Calc3 Overview

- Two new branch commands
  - Successful branch causes next command from port to be skipped

Copyright Wile, Roesner, Goss

Calc3 Overview

- Each requestor can still send in up to 4 commands
  - 2-bit tag on request
  - Using same tag simultaneously is not supported

Copyright Wile, Roesner, Goss

Calc3 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 in 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.

Copyright Wile, Roesner, Goss

I/O Protocols

For each requestor X:

Inputs:

- reqX_cmd(0:3)
  - add: 0001 adds contents of d1 to d2 and stores in r1
  - subtract: 0010 subtracts contents of d2 from d1 and stores in r1
  - shift left: 0101 shifts contents of d1 to the left d2(27:31) places and stores in r1
  - shift right:0110 shifts contents of d1 to the right d2(27:31) places and stores in r1
  - store: 1001 stores reqX_data(0:31) into r1
  - fetch:1010 fetches contents of d1 and outputs it on out_dataX(0:31)
  - branch if zero: 1100 skip next valid command if contents of d1 are 0
  - branch if equal: 1101 skip next valid command if contents of d1 and d2 are equal
- reqX_d1(0:3)
- reqX_d2(0:3)
- reqX_r1(0:3)
- reqX_tag(0:1)
- reqX_data(0:31)

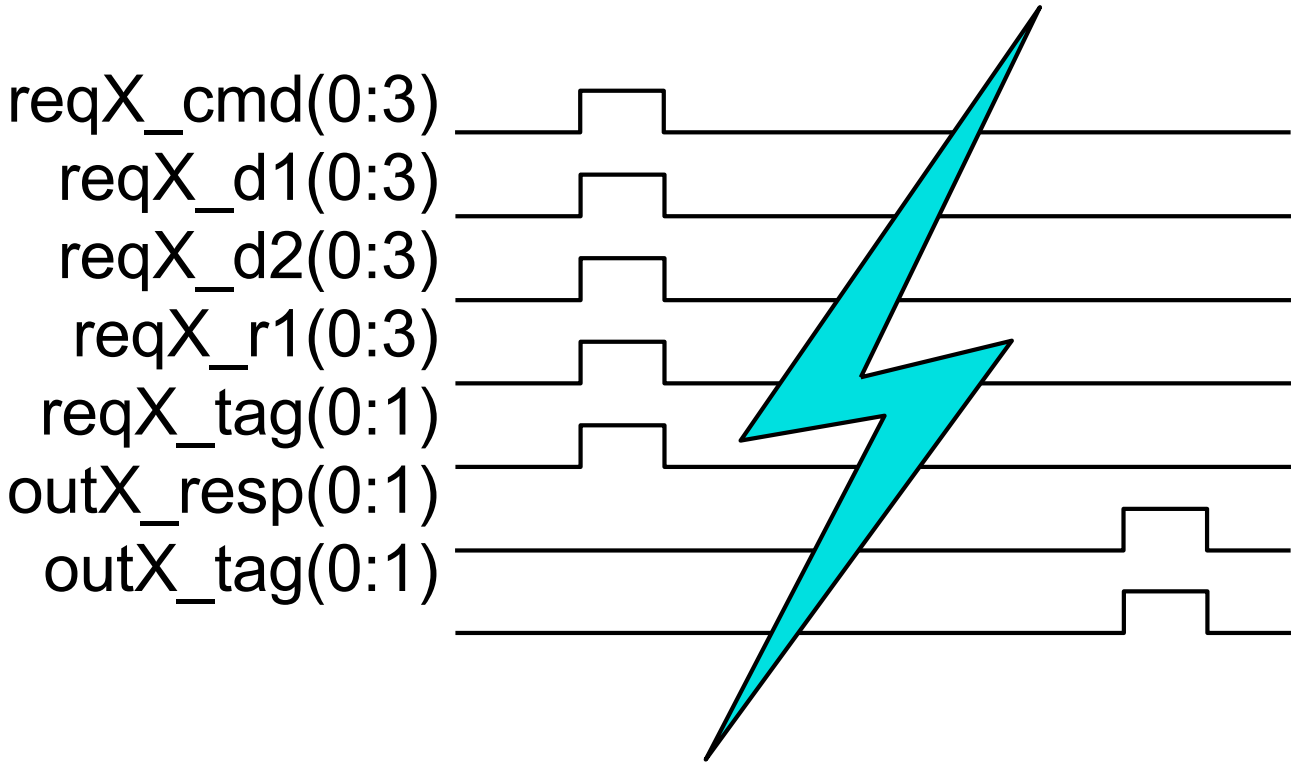
Outputs:

- outX_resp(0:1)
  - Successful completion: 01
  - overflow/underflow error:10
  - Command skipped due to branch: 11
- outX_tag(0:1)
- outX_data(0:31)

Copyright Wile, Roesner, Goss

I/O Timing

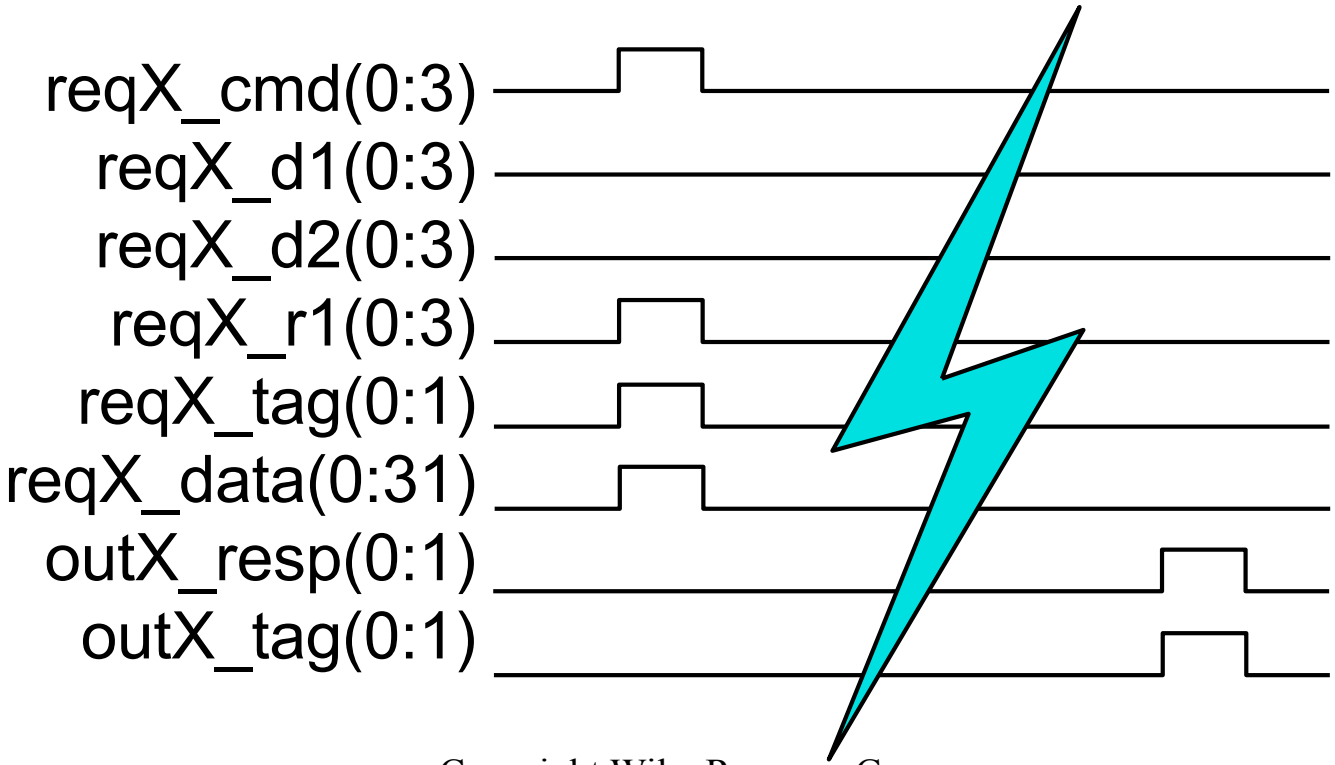
- Basic timing of a single arithmetic command:



Copyright Wile, Roesner, Goss

I/O Timing

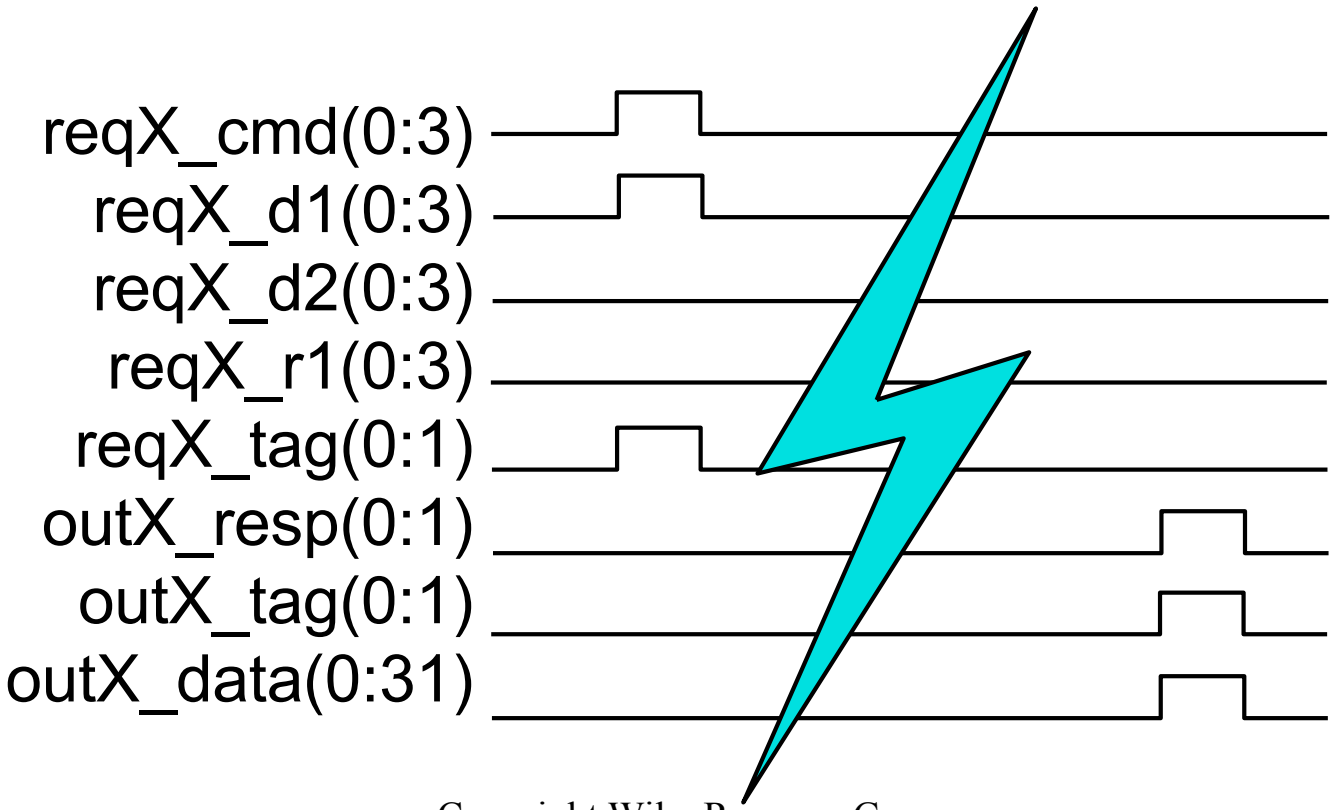
- Basic timing of a store command:



Copyright Wile, Roesner, Goss

I/O Timing

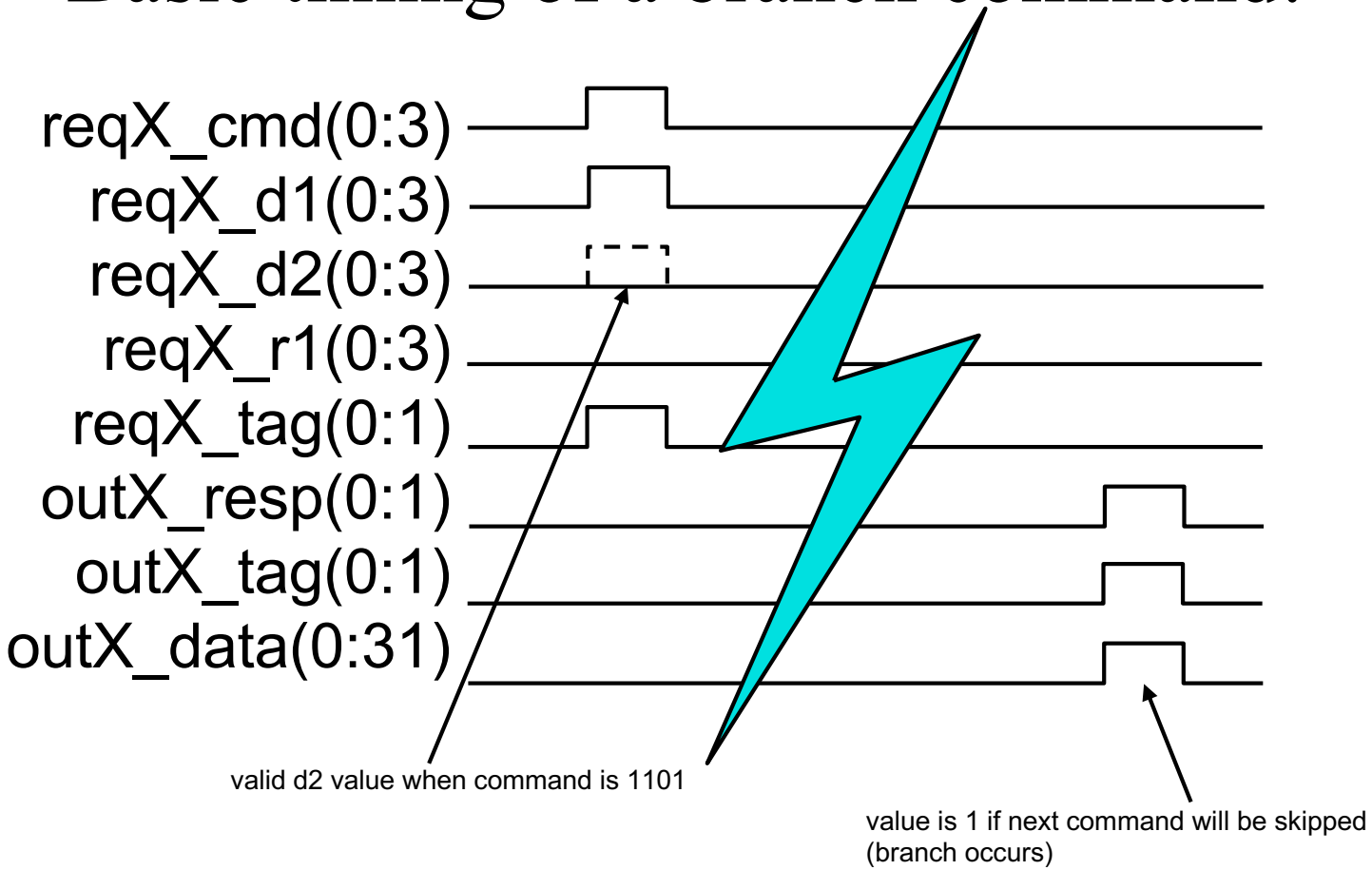
- Basic timing of a fetch command:



Copyright Wile, Roesner, Goss

I/O Timing

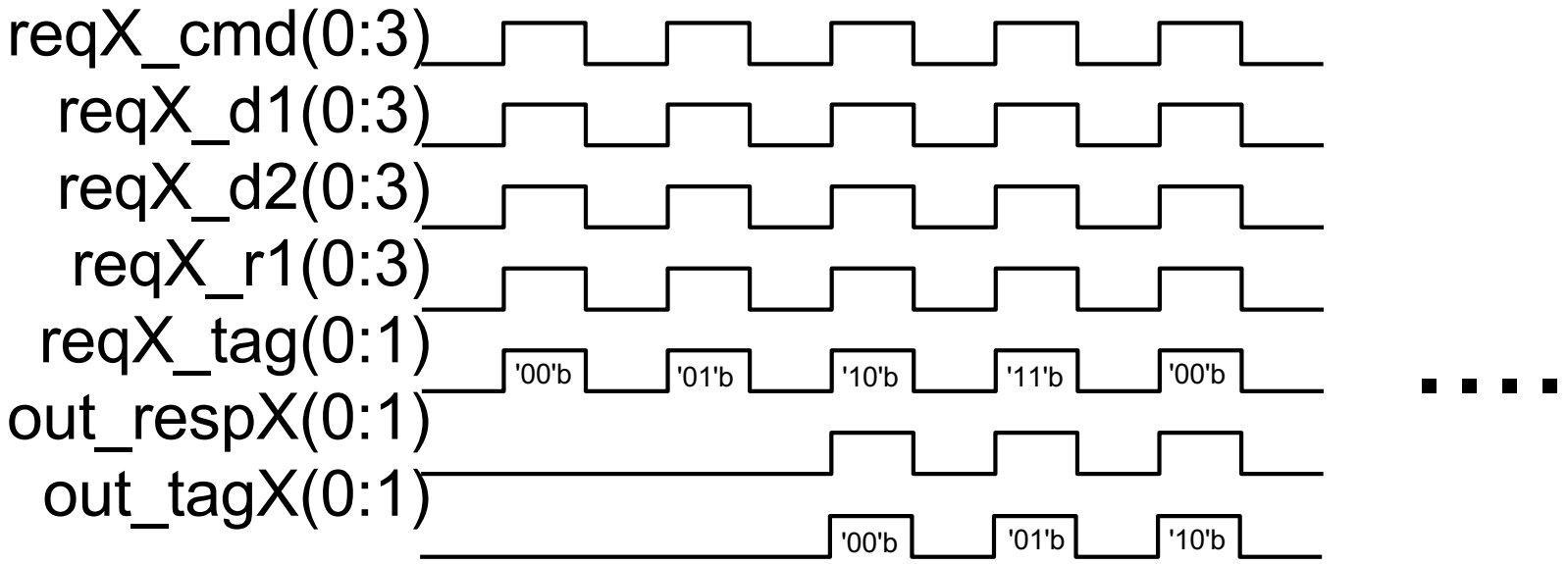
- Basic timing of a branch command:



Copyright Wile, Roesner, Goss

I/O Timing

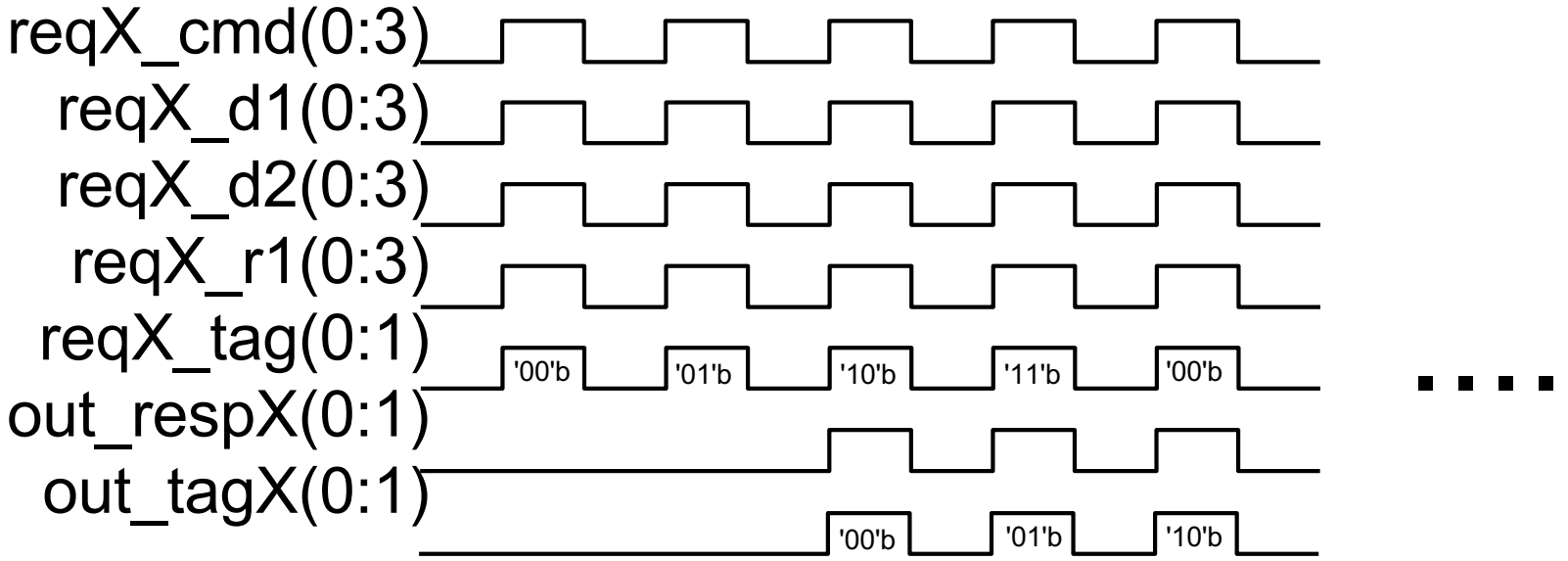
- Fastest multiple command (any cmd type) timing (example is if only one requestor is sending commands):



Copyright Wile, Roesner, Goss

I/O Timing

- Requestor must leave dead cycle between commands.
- Requestor may only have 4 commands outstanding at a time, each with a unique tag.



Copyright Wile, Roesner, Goss

Command That Follows 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
  - Brach will evaluate to false (case of branch followed by branch)

Copyright Wile, Roesner, Goss

Command That Follows a 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.”

Copyright Wile, Roesner, Goss

Command and Ordering Rules

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

Copyright Wile, Roesner, Goss

Functional Verification

- Build a verification environment for
- Calc1 ➔ Done (Directed testcases)
- Calc2,Calc3.
- Test Plan
- Random verification
- Use SystemVerilog to build your verification environment.
- You need to report all the bugs founds in Calc2 and Calc3

Copyright Wile, Roesner, Goss

Functional verification

- Report for all students
- Presentation for grade students only.