MSOC Lab #1

Yu-Sheng Lin

Po-Chen Wu

Example #0
Hello world

In Example #0, you don’t need to write code

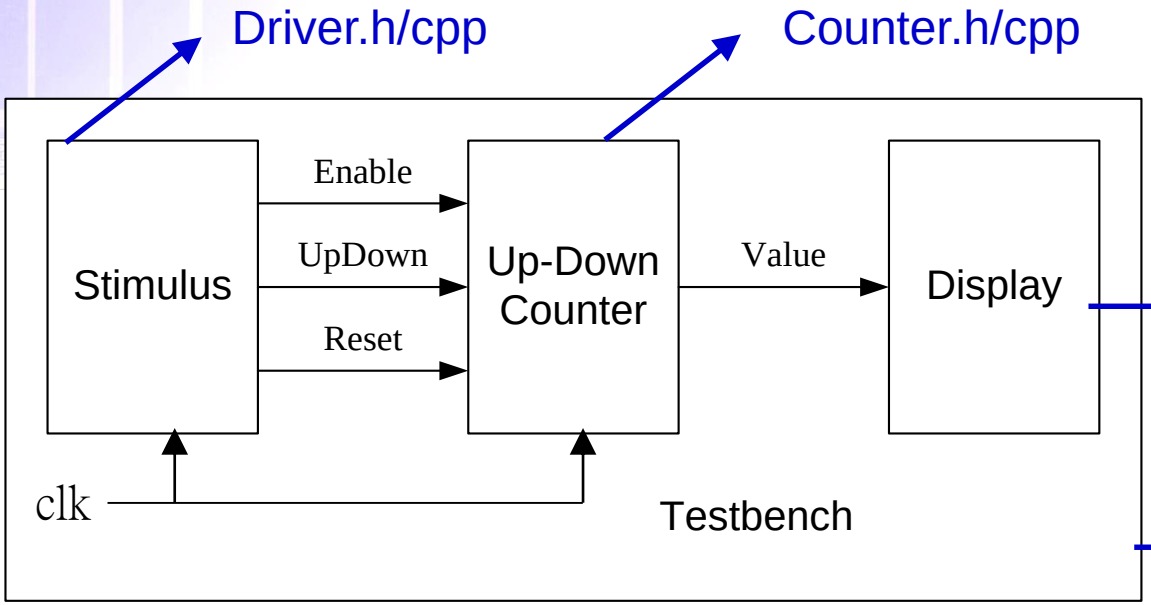
You just need to build and run the code, please refer to the other slide

Example #1
Up-Down Counter

Up-Down Counter

Counter Spec.

Value range: [0,15] ,
with saturation
(up to 15 and stop at 15;
down to 0 and stop at 0)



Enable	1: enable updown
UpDown	1: value <= value+1; 0: value <= value-1
Reset	1: value <= 0, highest priority
Value	Value of counter

Counter

```
SC_MODULE(Counter) {
    sc_in_clk   clk_;
    sc_in<bool> rst_;
    sc_in<bool> enable_;
    sc_in<bool> up_down_;
    sc_out<int> value_;
```

```
module Counter(
    input clk_,
    input rst_,
    input enable_,
    input up_down_,
    output [31:0] value_
```

Driver (Testbench)

```
enable_.write(false);
rst_.write(false);
up_down_.write(false);
wait();
rst_.write(true);
wait();
rst_.write(false);
wait();
enable_.write(true);
up_down_.write(true);
```

```
enable = 0;
rst = 0;
up_down = 0;
@(posedge clk)
rst = 1;
@(posedge clk)
rst = 0;
@(posedge clk)
enable = 0;
up_down = 0;
```

Running the Code…

- You get an assertion error!
- Before debugging the Counter, we observe the Counter first!

Our Display Module

```
SC_MODULE(Display) {
    sc_in<int> value_; // module Display(input [31:0] value_);
```

- Add the value change to the sensitive list
  - `sensitive << value_;`
- OK, you can fix the bug now

Example #2
FIR Filter

Outline

- In Example #2, we demo an advanced data type FIFO
  - `sc_fifo` ~ `sc_signal` (Channel)
    `sc_fifo_in_if` ~ `sc_signal_in_if` (Interface)
    `sc_fifo_out_if` ~ `sc_signal_out_if` (Interface)
- Unlike signals, FIFO has timing information

FIFO v.s. Signal (1)

- If you FIFO.write() once, then you can FIFO.read() once

- If you Signal.write() once, then Signal.read() will always yield the same value

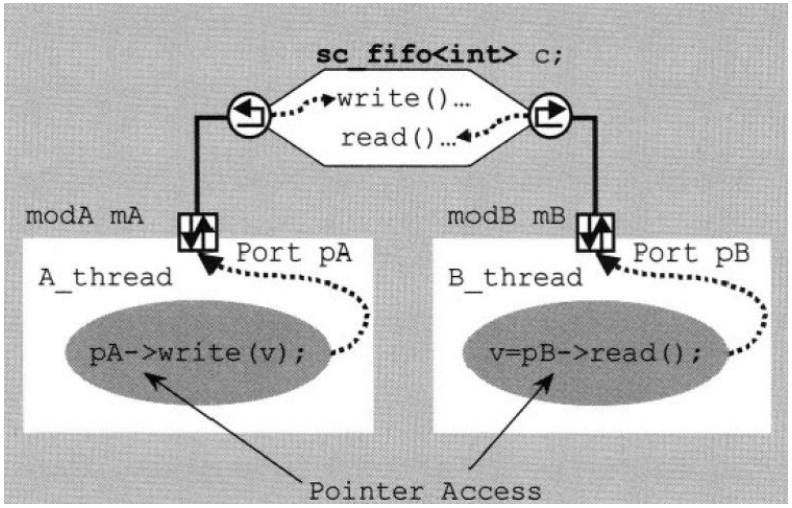
FIFO v.s. Signal (2)

- If FIFO.write() is called when the FIFO is full, it waits until someone FIFO.read() something to it, and vice versa

- Signal.write() and Signal.read() effects immediately, that is, they never block

Definition

- Interface : Abstract class
- Channel : Implementation class
- Port : Interface pointer



(Recap)

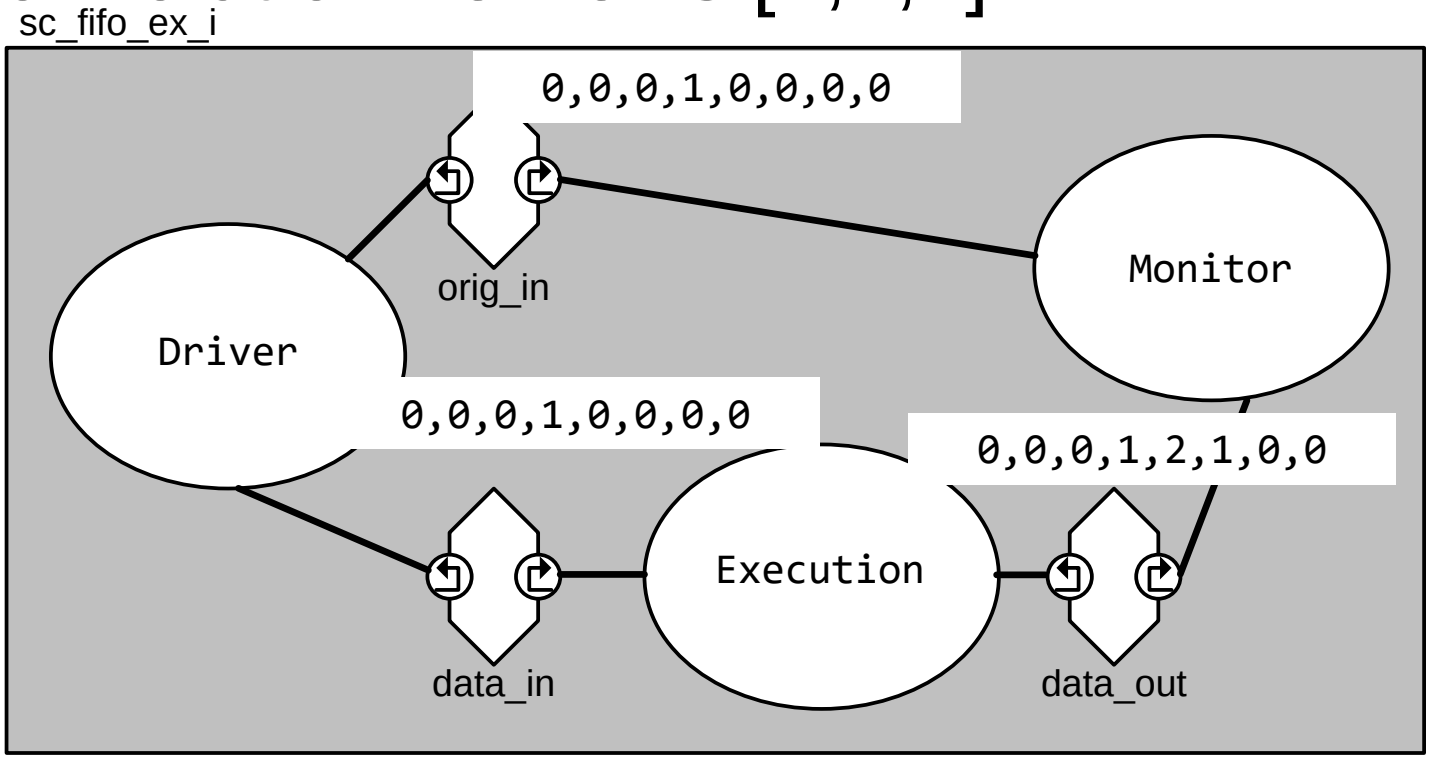
- A ***channel*** is a ***module*** that implements specific ***interfaces***
- Use ***ports*** to connect ***channels***

Convenient Classes

- sc_signal_in_if
  - sc_in = sc_port<sc_signal_in_if>
- sc_signal_out_if
  - sc_out = sc_port<sc_signal_out_if>
- sc_signal_inout_if
  - sc_inout = sc_port<sc_signal_inout_if>
- The same applies for FIFOs

Code Structure

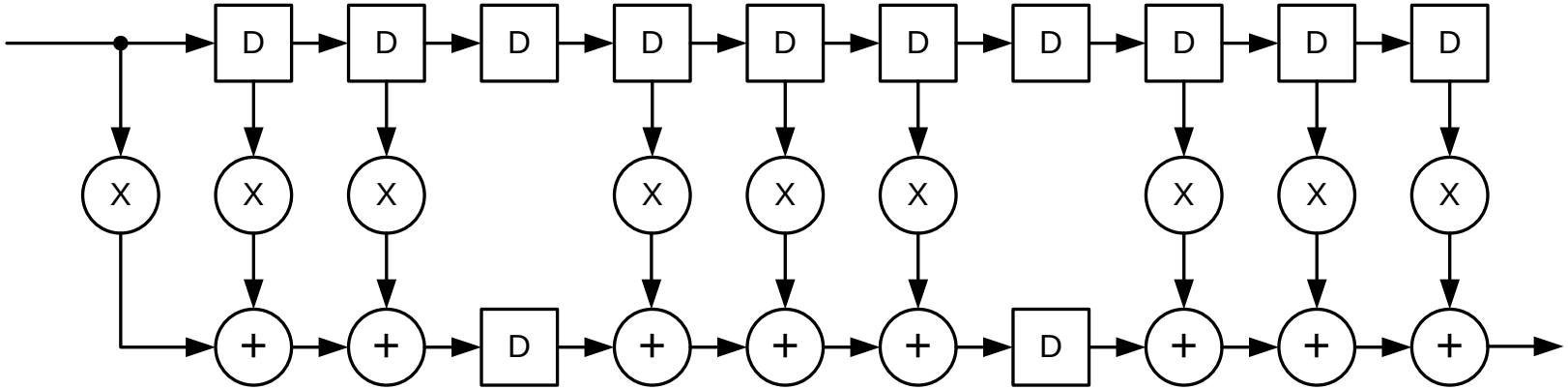
- Convolution kernel is [1,2,1]

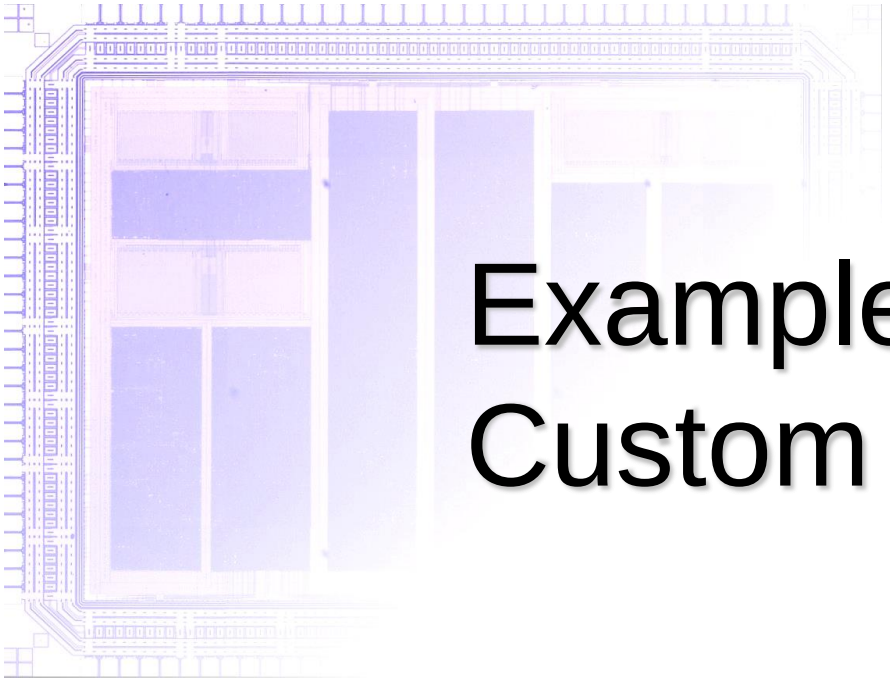


Code Structure in Text

```
SC_MODULE(FIR) {
    sc_fifo<int> orig_in_;
    sc_fifo<int> data_in_;
    sc_fifo<int> data_out_;
    void Driver();
    void FirExecution();
    void Monitor();
```

A Possible FIR Implementation



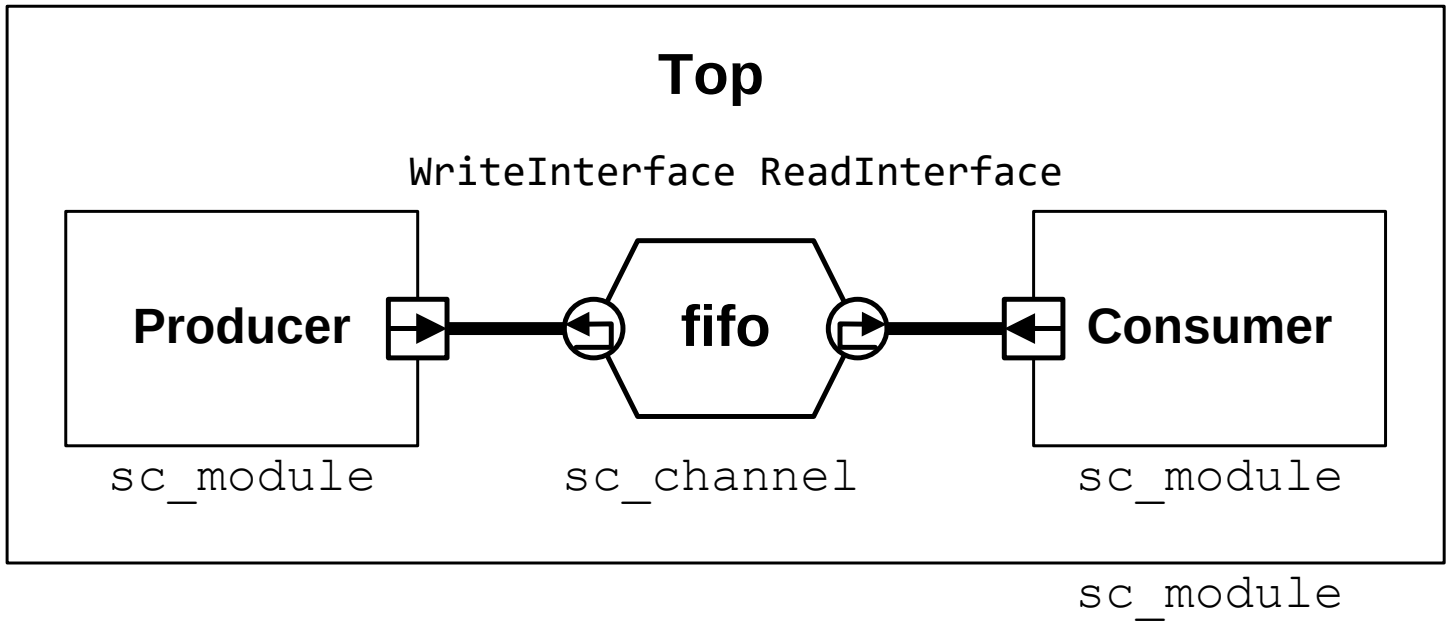


Example #3
Custom Interface

Re-invent the FIFO

- In this part, we practice interface/channel by implementing the FIFO again
  - `sc_fifo -> Fifo`
    `sc_fifo_in_if -> WriteInterface`
    `sc_fifo_out_if -> ReadInterface`

Simple FIFO


Top
WriteInterface ReadInterface
Producer
fifo
Consumer
sc_module
sc_channel
sc_module
sc_module

Implement Read Interface

```
class ReadInterface:
    virtual public sc_interface {
public:
    virtual void Read(char&) = 0;
    virtual int NumAvailable() = 0;
    virtual bool Empty() = 0;
};
```

Implement FIFO Channel By Inheriting Interface(s)

```
class Fifo:
        public sc_channel,
        public ReadInterface,
        public WriteInterface {
public:
        void Write(char c);
        void Read(char &c);
        void Reset();
        int NumAvailable();
        bool Full();
        bool Empty();
```

Consumer

```
if(in_->Empty()) {
    in_->Read(c);
}
```

Actually, we can ignore the empty check, why?

Blocking and Non-blocking

```
const bool ALWAYS_BLOCK = true;
if(ALWAYS_BLOCK and in_->Empty()) {
    cout << "Read: Blocking from " << sc_time_stamp();
    in_->Read(c);
    cout << " to " << sc_time_stamp() << "." << endl;
} else {
    cout << "Read: Nonblocking from " << sc_time_stamp();
    in_->Read(c);
    cout << " to " << sc_time_stamp() << "." << endl;
}
wait(100, SC_NS);
```

Play with the boolean flag and observe it. There is no coding here. (Hint: notice the timing before and after the read)

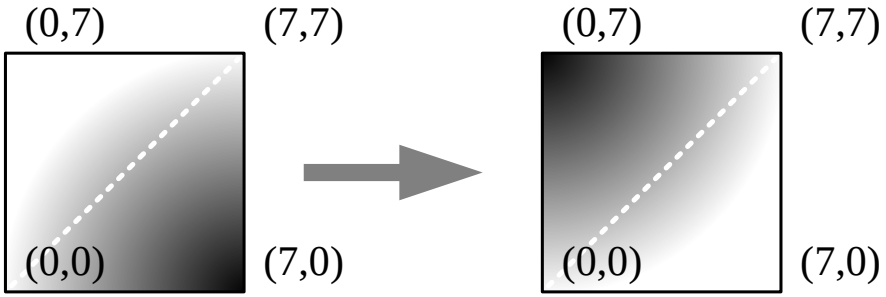
Example #4
Transaction Level Modeling

Transaction Level Modeling

- Majorly used for functional modeling, platform modeling, and testbench constructing.
- Testbenchs are written with channel such as buses and FIFO.
- Allow incremental refining
- Emphasize MORE on data transfer, LESS on signals

A Matrix Transpose Example

- In example #5-1, #5-2 we show a 8-by-8 memory transpose unit implemented in different ways
- First, we have two interfaces
  - `virtual void direct_read(int** block) = 0;`
  - `virtual void word_read(unsigned x, unsigned y, int& d) = 0;`



Model of Different Accuracy

```
mport->direct_read(reg);
for(int j = 0; j < 8; j++) {
    for(int i = 0; i < j; i++) {
        swap(reg[j][i], reg[i][j]);
    }
}
mport->direct_write(reg);
```

Model a large read/write transaction (much useless, mainly for correctness check)

```
for(int j = 0; j < 8; j++) {
    for(int i = 0; i < j; i++) {
        mport->direct_read(
            i, j, reg[i][j]
        );
    }
}
…
for(int j = 0; j < 8; j++) {
    for(int i = 0; i < j; i++) {
        mport->direct_write(
            i, j, reg[i][j]
        );
    }
}
```

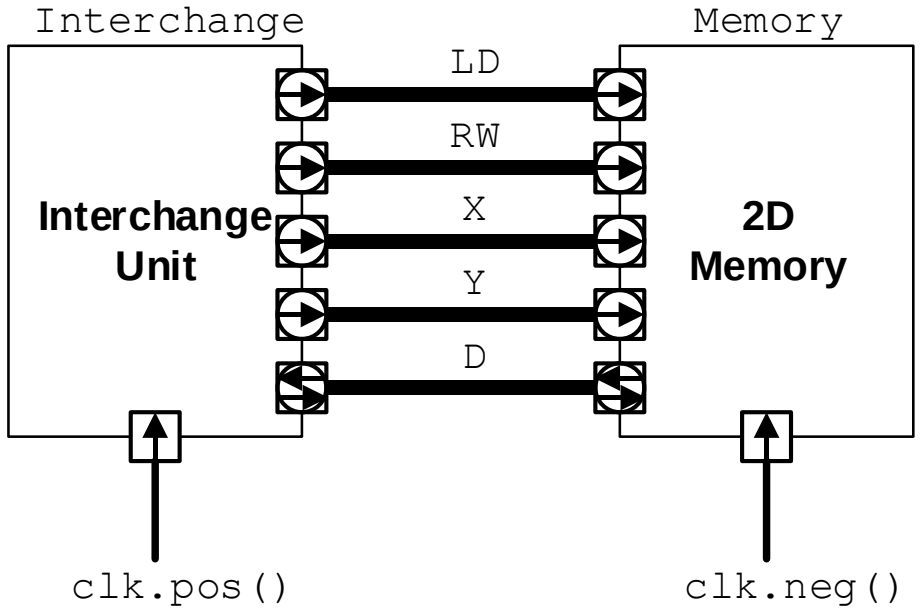
Model a data read/write at once (no timing)

Cycle Accurate Model

- In Example #5-3, we model a more detailed signal level memory with cycle.

The Cycle Accurate Code

```
auto Z = sc_lv<32>(SC_LOGIC_Z);
LD->write(true);
RW->write(READ);
X->write(i);
Y->write(j);
D->write(Z);
wait();
LD->write(false);
```



But It's Inconvenient

- Verification team never want to know the details HW team

They think their testbench should look like this.

```
mport->direct_write(
    i, j, reg[i][j]
);
```

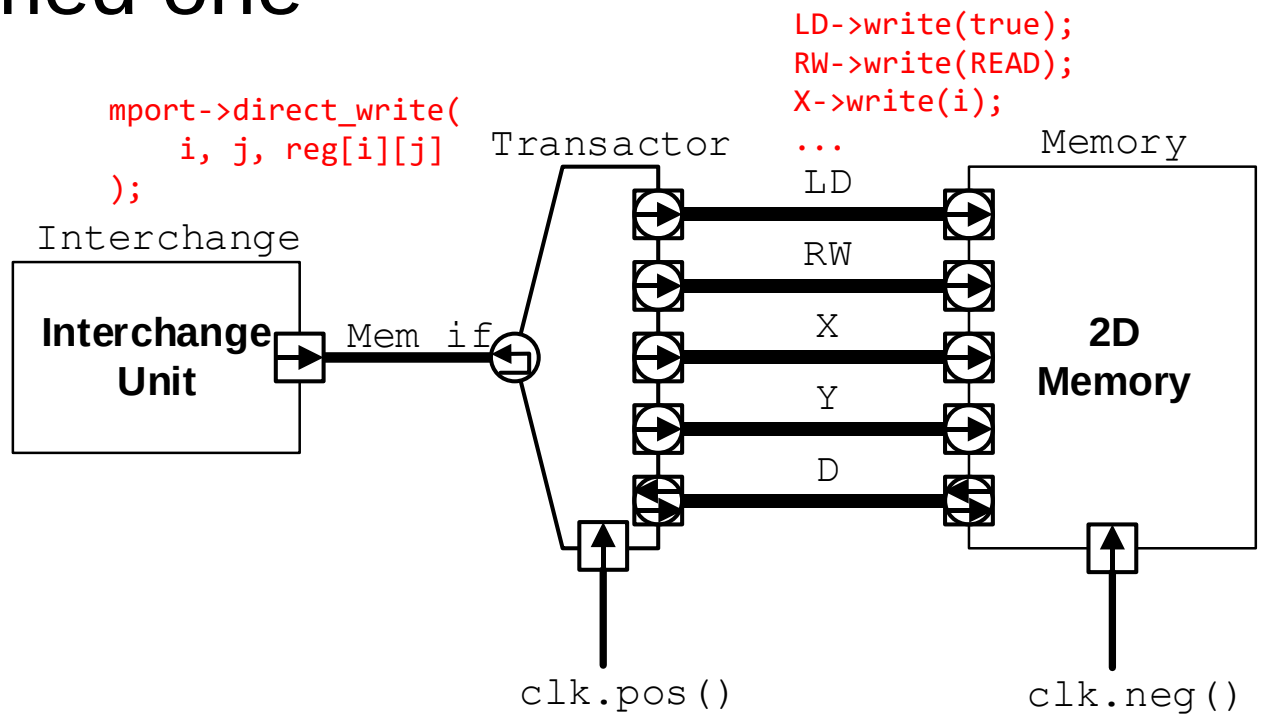
Not this

```
auto Z = sc_lv<32>(SC_LOGIC_Z);
LD->write(true);
RW->write(READ);
X->write(i);
Y->write(j);
D->write(Z);
wait();
LD->write(false);
```

BTW, verification team members commonly come from senior HW engineers

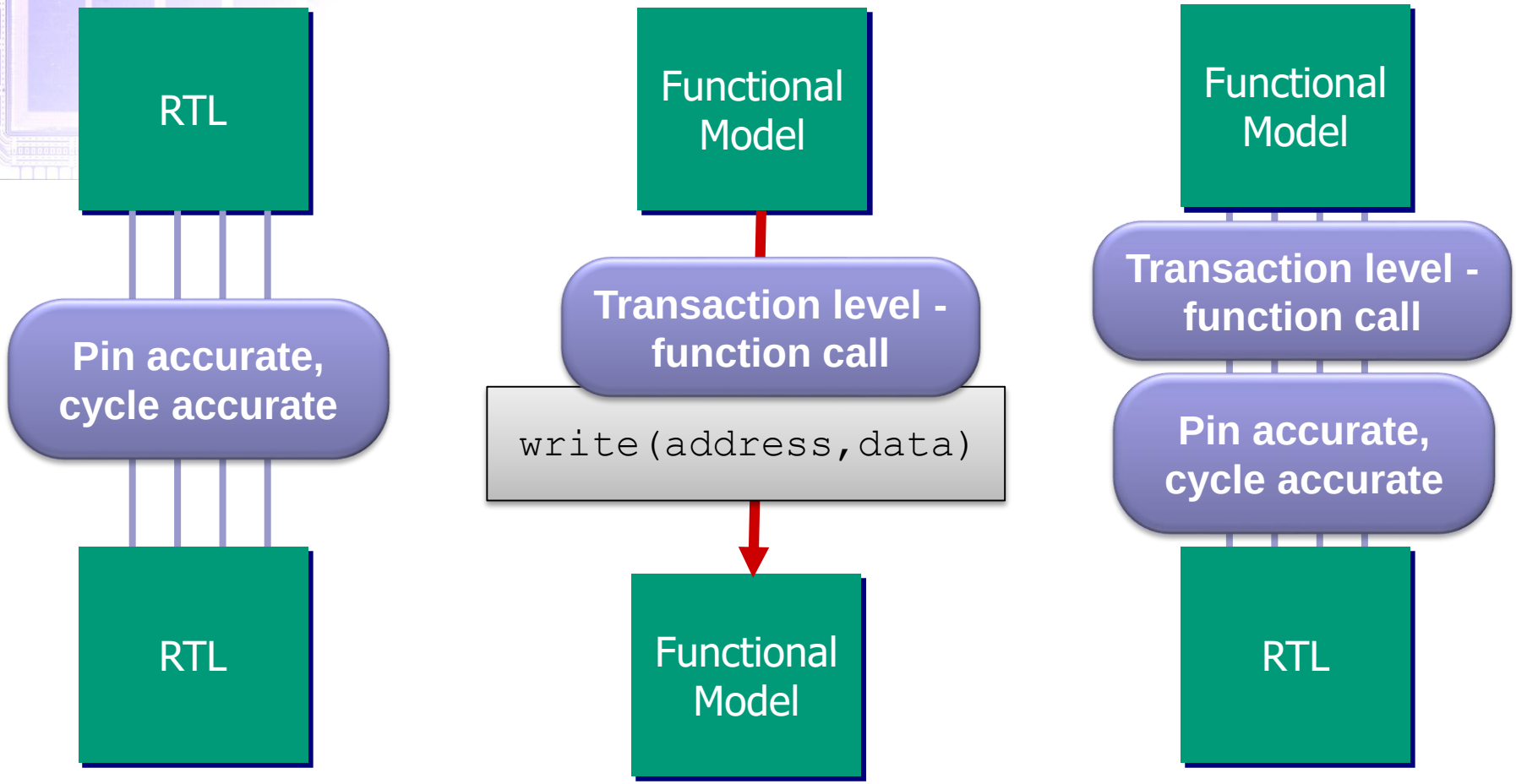
So We Need A Transactor

- Transactor convert a untimed model to a timed one



Appendix
Introduction of Transaction Level Modeling (TLM) Standard

What/Why TLM?



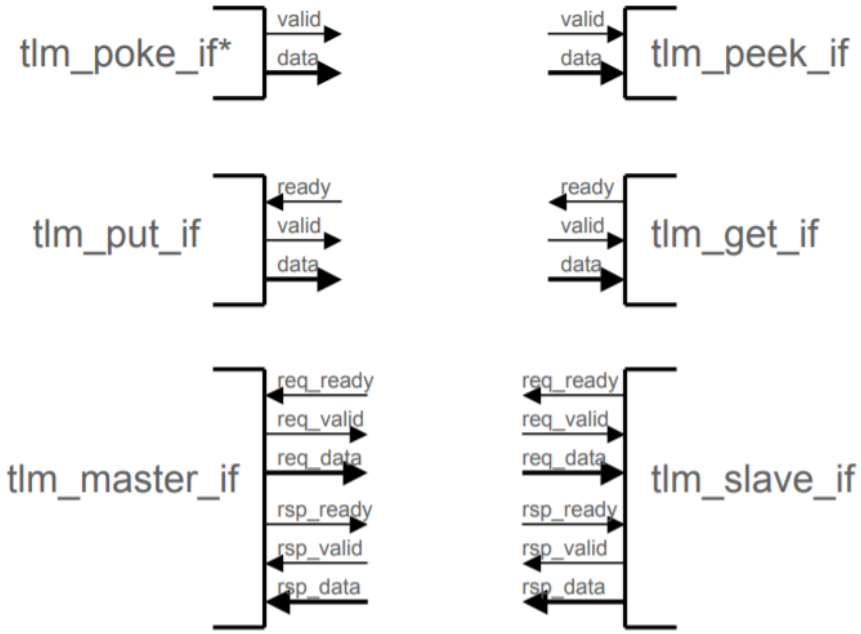
Simulate every event 100-10,000 X faster simulation Easy testing

Why TLM (cont'd)?

- **Fast**
- Integrate HW and SW models
- Early platform for SW development, easy to distribute
- **Early system exploration and verification**
- Verification reuse

TLM 1.0

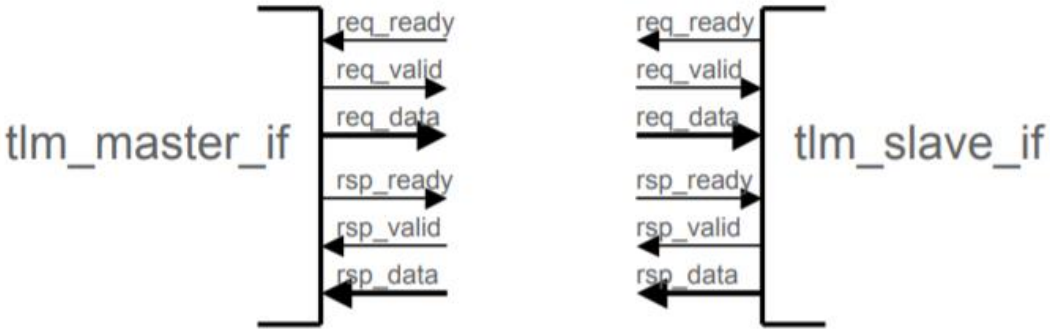
- TLM 1.0 models common low-level module protocols as interfaces



http://www.ti.uni-tuebingen.de/uploads/media/Presentation-13-OSCI_2_swan.pdf

SRAM Example #4 Revisit

- SRAM is a special case of TLM.
  - M → S: RW, address, (data)
  - S → M: (data)



http://www.ti.uni-tuebingen.de/uploads/media/Presentation-13-OSCI_2_swan.pdf

TLM-1.0 → TLM-2.0

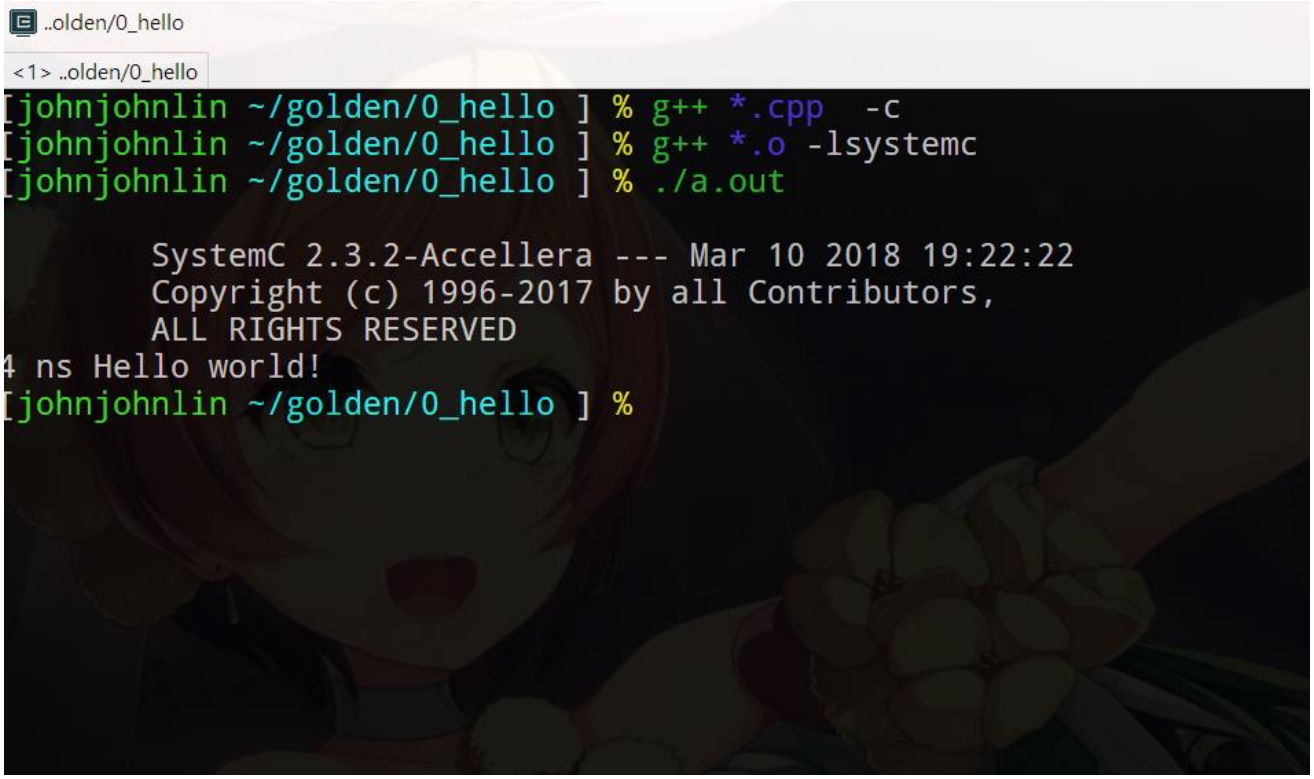
- TLM-2.0 is the new standard for interoperability between **memory mapped bus models**
  - It's for higher level
- TLM-1.0 is not deprecated
  - TLM-1.0 is included within TLM-2.0

Assignment #1

Requirements

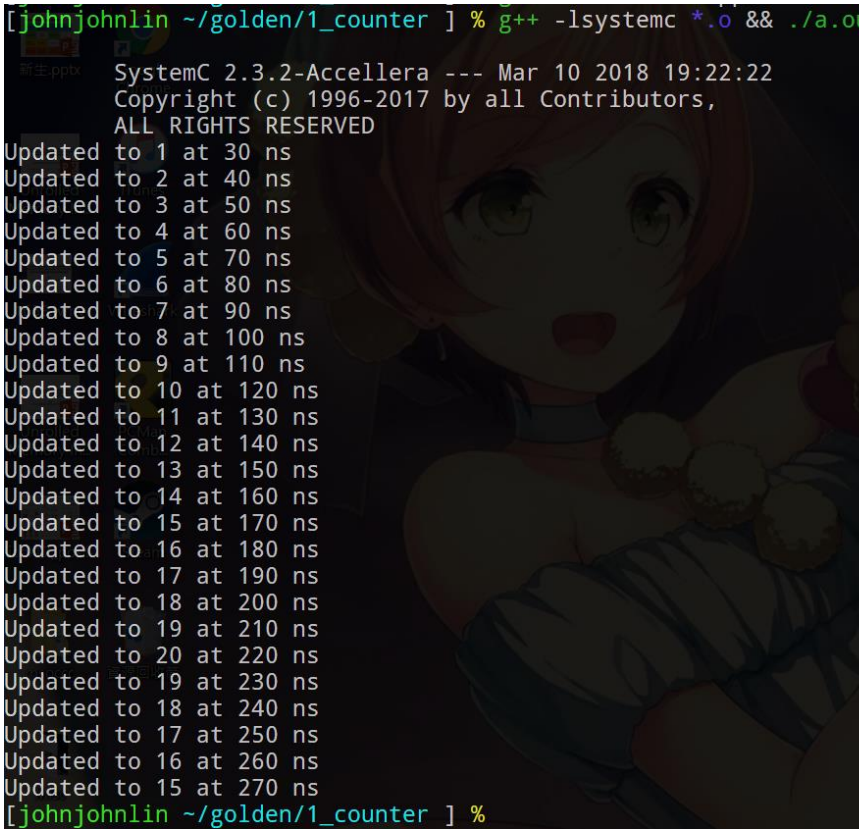
Lab Requirement (Example #0)

- Build and run it, an ArchLinux example is shown here.



Lab Requirement (Example #1)

- Modify Display class to print the value change
- Fix the bug in the counter
- Also, test decrementing the counter
- Submission:
  - 1_counter/Display.cpp
  - 1_counter/Counter.cpp
  - 1_counter/main.cpp
  - 1_counter/TestDriver.cpp

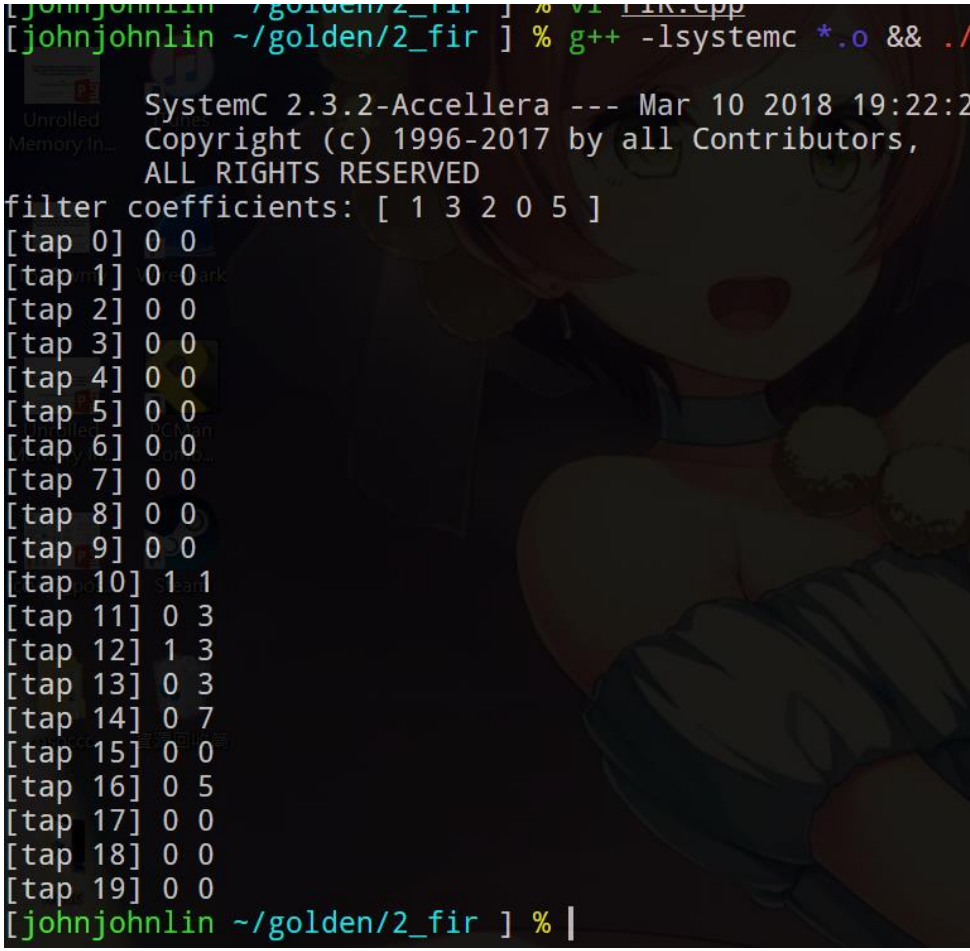


Lab Requirement (Example #1)

- Discuss in report.pdf
  - The else branch is never executed, why?

Take-Home HW (Example #2)

- Implement FirExecution such that it yields the correct results.
- Submission:
  - 2_fir/FIR.cpp



Take-Home HW (Example #3)

- Implement
  - Fifo::Empty, Fifo::Full, Fifo::Write
- Submission:
  - 3_fifo/Fifo.cpp

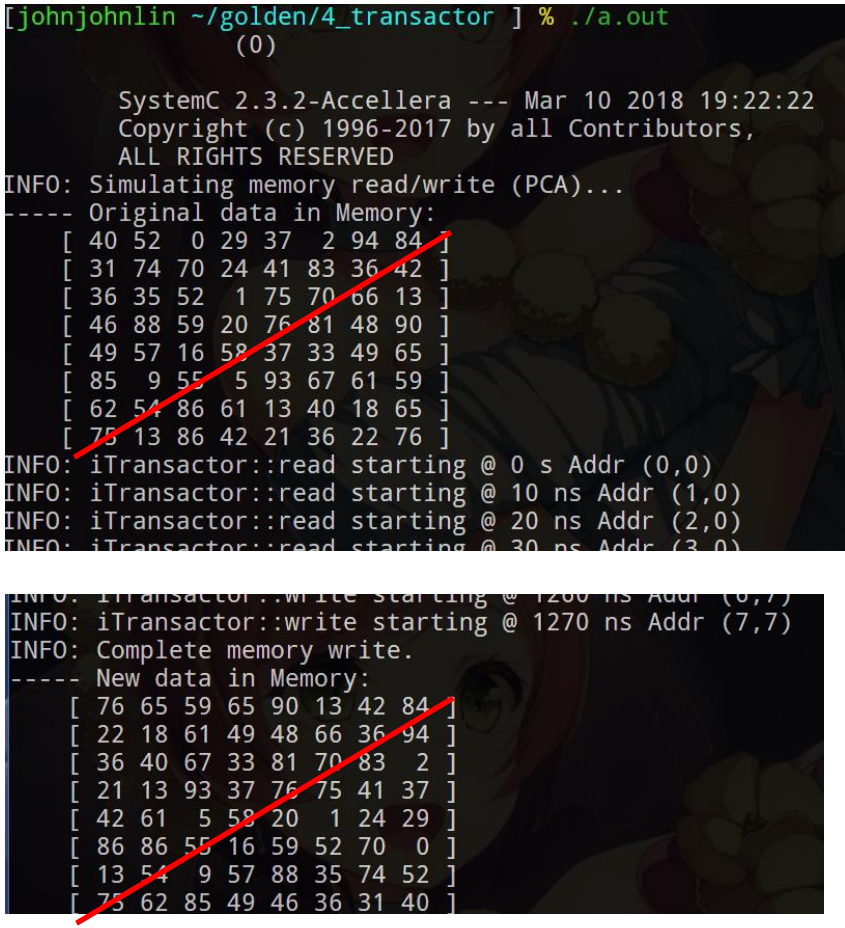
```
[johnjohnlin ~/golden/3_fifo ] % ./a.out

        SystemC 2.3.2-Accellera --- Mar 10 2018 19:22:22
        Copyright (c) 1996-2017 by all Contributors,
        ALL RIGHTS RESERVED
Read: Nonblocking from 0 s to 0 s(H).
Read: Blocking from 100 ns to 1 us(H).
Read: Nonblocking from 1100 ns to 1100 ns(i).
Read: Nonblocking from 1200 ns to 1200 ns(,).
Read: Blocking from 1300 ns to 2 us(H).
Read: Nonblocking from 2100 ns to 2100 ns(i).
Read: Nonblocking from 2200 ns to 2200 ns(,).
Read: Nonblocking from 2300 ns to 2300 ns(_).
Read: Nonblocking from 2400 ns to 2400 ns(M).
Read: Nonblocking from 2500 ns to 2500 ns(o).
Read: Nonblocking from 2600 ns to 2600 ns(n).
Read: Nonblocking from 2700 ns to 2700 ns(i).
```

Take-Home HW (Example #3)

- Discuss in report.pdf
  - Did you notice that some parts are unchanged/changed if we change the ALWAYS_BLOCK flag? Please explain why they are changed/unchanged

Take-Home HW (Example #4)

- (Bonus part!!)
- Implement the transactor class
  - (Numbers might change)
- Submission:
  - 4_transactor/4_transactor/Transactor.cpp



Grading Rule

- Example #0 (10%)
- Example #1 (25/10%)
- Example #2 (20%)
- Example #3 (20/15%)
- Example #4 (20%)