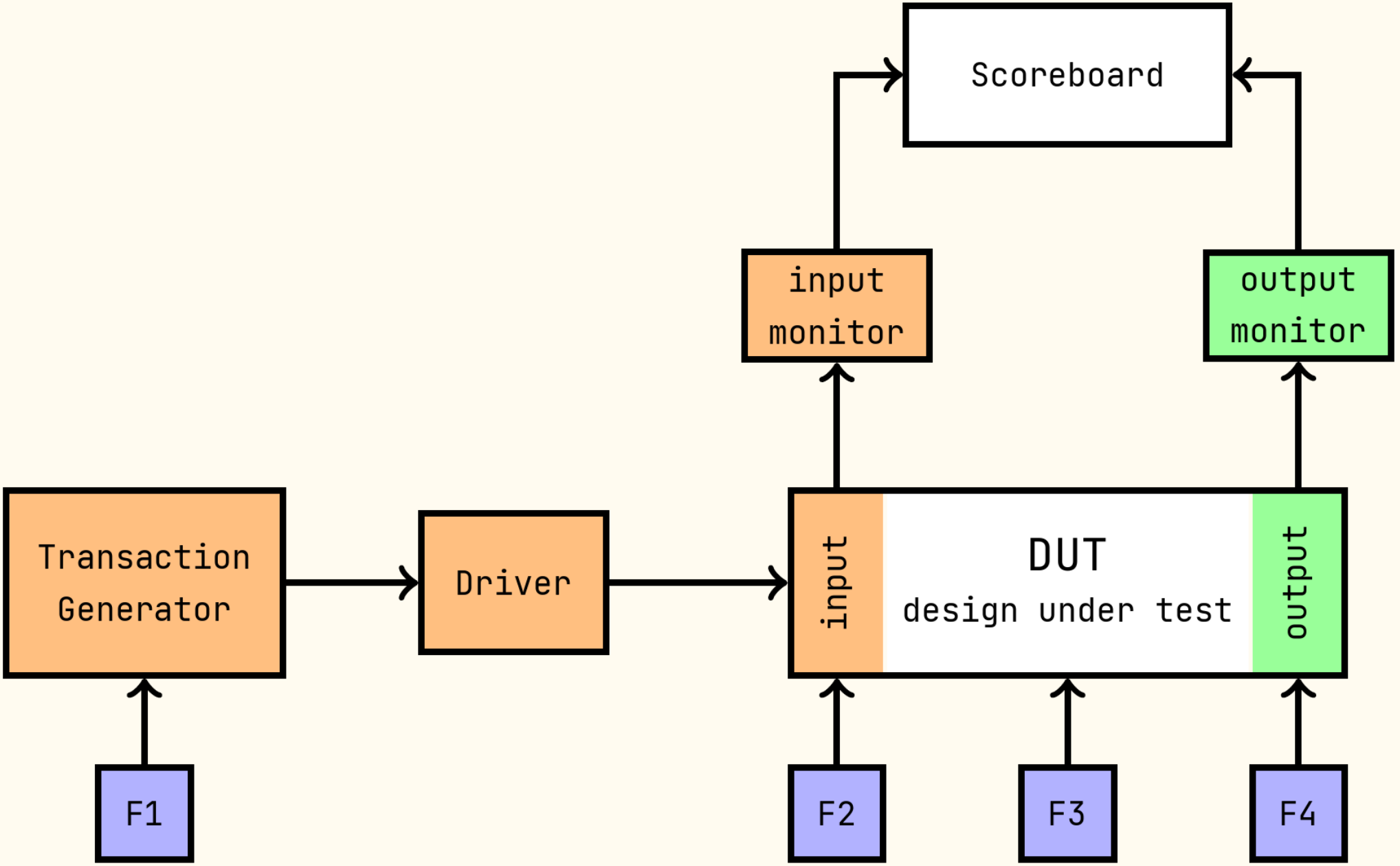
Verilator 101 (cont.)

Presented by Hai Cao Xuan
Computer Architecture 203B3

objectives

- Understand how to use basic Verification Methodology
- Learn how to write basic coverage and SVA
- **Make a Protocol-like system**

Verification Methodology


Scoreboard
input monitor
output monitor
Transaction Generator
Driver
input
DUT
design under test
output
F1
F2
F3
F4

Coverage

coverage points

1. F1: Near to the randomization and it is before actual BFM/driver to DUT.
2. F2: Inside a input monitor or input interface to sample the DUT input signals.
   → Perfect for having functional coverage on stimulus
3. F3: Separate inside DUT to monitor the internal states of DUT, like FSM states, or some registers.
4. F4: Inside a output monitor or output interface to sample the DUT output signals.
   → Perfect for having functional coverage on outputs of DUT

coverage

`Line`

Make sure all lines of codes execute

`Toggle`

Make sure all inputs and outputs toggle

`Functional`

Make sure the system reaches covered functions

cover

```
--coverage

--coverage-line

--coverage-toggle

--coverage-user
```

```
verilator_coverage --annotate logs/annotated --annotate-min 1 --
write-info logs/coverage.info logs/coverage.dat
```

cover

Cover a statement

```
always @(<clock>) begin [: cov_...]
    [cover name:] cover (@(<clock>) (<statement>));
end
```

```
[cover name:] cover property (@(<clock>) (<statement>));
```

```
Happen: cover property (@(posedge clk_i) (state_q == IDLE);
```

cover

`$past(X)`

`$past(X,N)`

Return the value of a signal in the past. It is clock-dependent.

X is a signal. N is the number of clocks that X is evaluated. If N is 1, use the first one.

```
always @(posedge i_clk)
  if (!$past(i_reset_n))
    cover(state == RESET);
always @(posedge i_clk)
  cover($past(i_data) == 4'b0011);
```

cover

How to cover a “rise”?

cover

`$rose(X)`

True if the signal rises on the clock edge. If the signal has more than one bit, it only examines the bit 0.

`$fell(X)`

True if the signal falls on the clock edge. If the signal has more than one bit, it only examines the bit 0.

little **exercise**

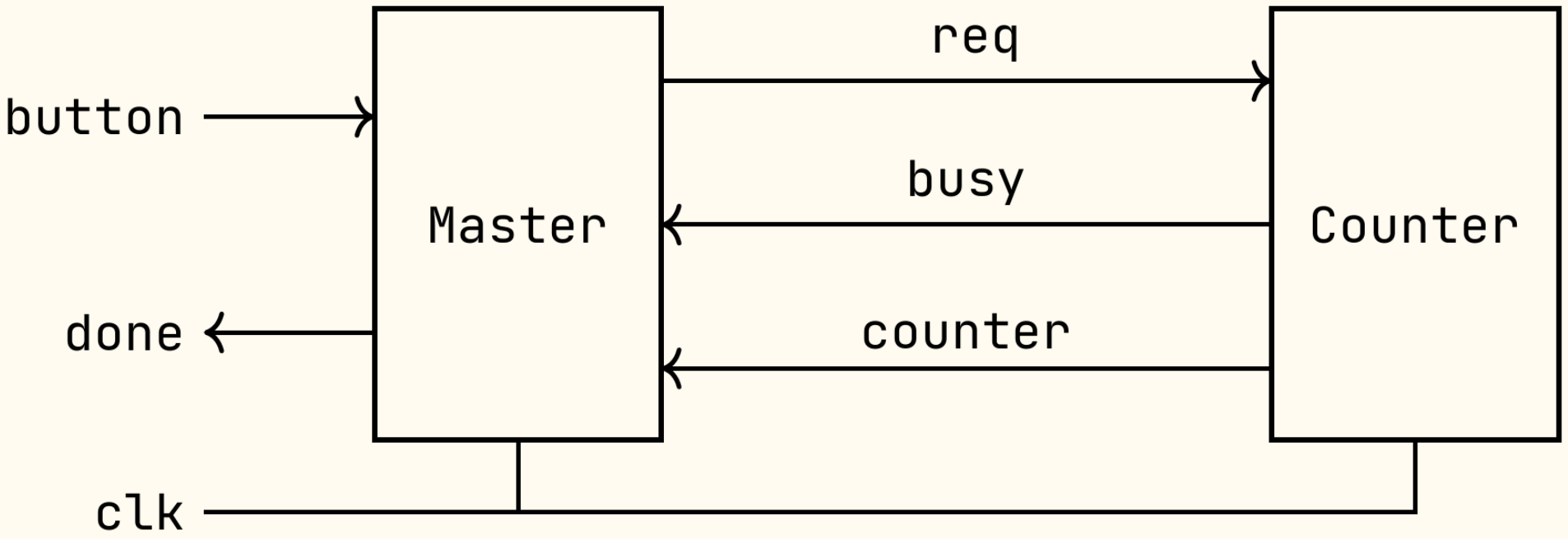
Problem

Design a Master-like and a Timer-like (Counter) to silumate a simple protocol.

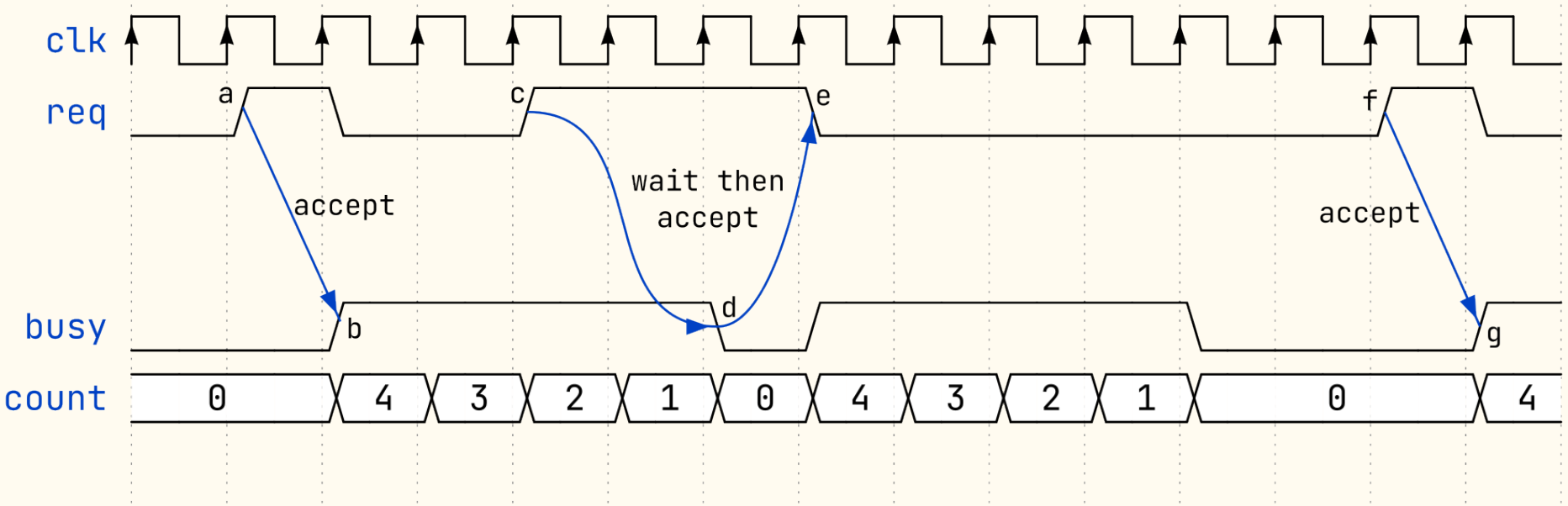
Master receives a signal from button, then requests the Counter to countdown. During the Counter-ing, if Master requests another one, it has to hold the request until the Counter is done. If the Counter is finished, the Master will raise done signal in one cycle.

Master should have button buffer to make sure the button only produce one request per multi-cycle pressing.

Block diagram

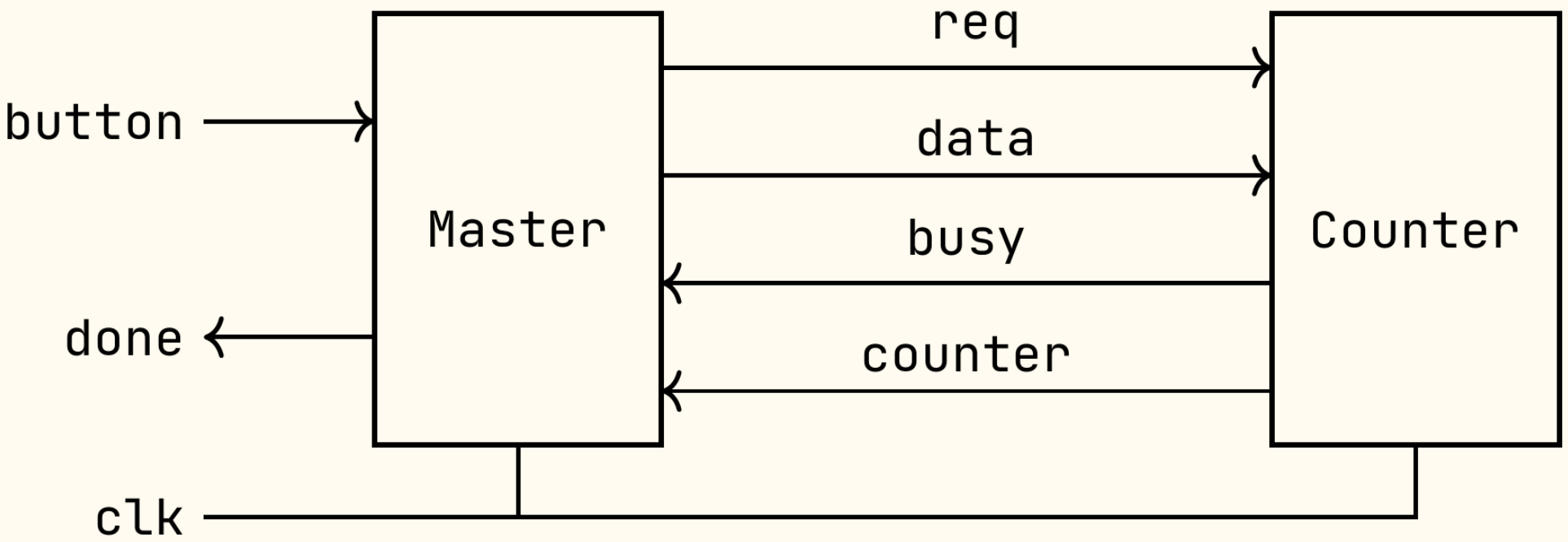


Waveform


clk
req
busy
count
a
b
c
d
e
f
g
accept
wait then
accept
accept
0
4
3
2
1
0
4
3
2
1
0
4

A little bit harder

Block diagram



Questions?

References

1. https://www.veripool.org/verilator/
2. https://verilator.org/guide/latest/
3. https://verilator.org/guide/latest/exe_verilator_coverage.html