Advanced VMM Transactor Development:
Tips for designing VIP you wouldn't mind reusing

Kelly D. Larson
MediaTek Wireless, Inc.
Austin Design Center

MEDIATEK

MEDIATEK
MediaTek
Wireless, Inc.

Overview

- Introduction
  - What makes something good for reuse?
- Specific tips and tricks
  - Beyond basic transactors
  - Highlights from paper
- Conclusion

MEDIATEK
MediaTek
Wireless, Inc.

SystemVerilog Adoption

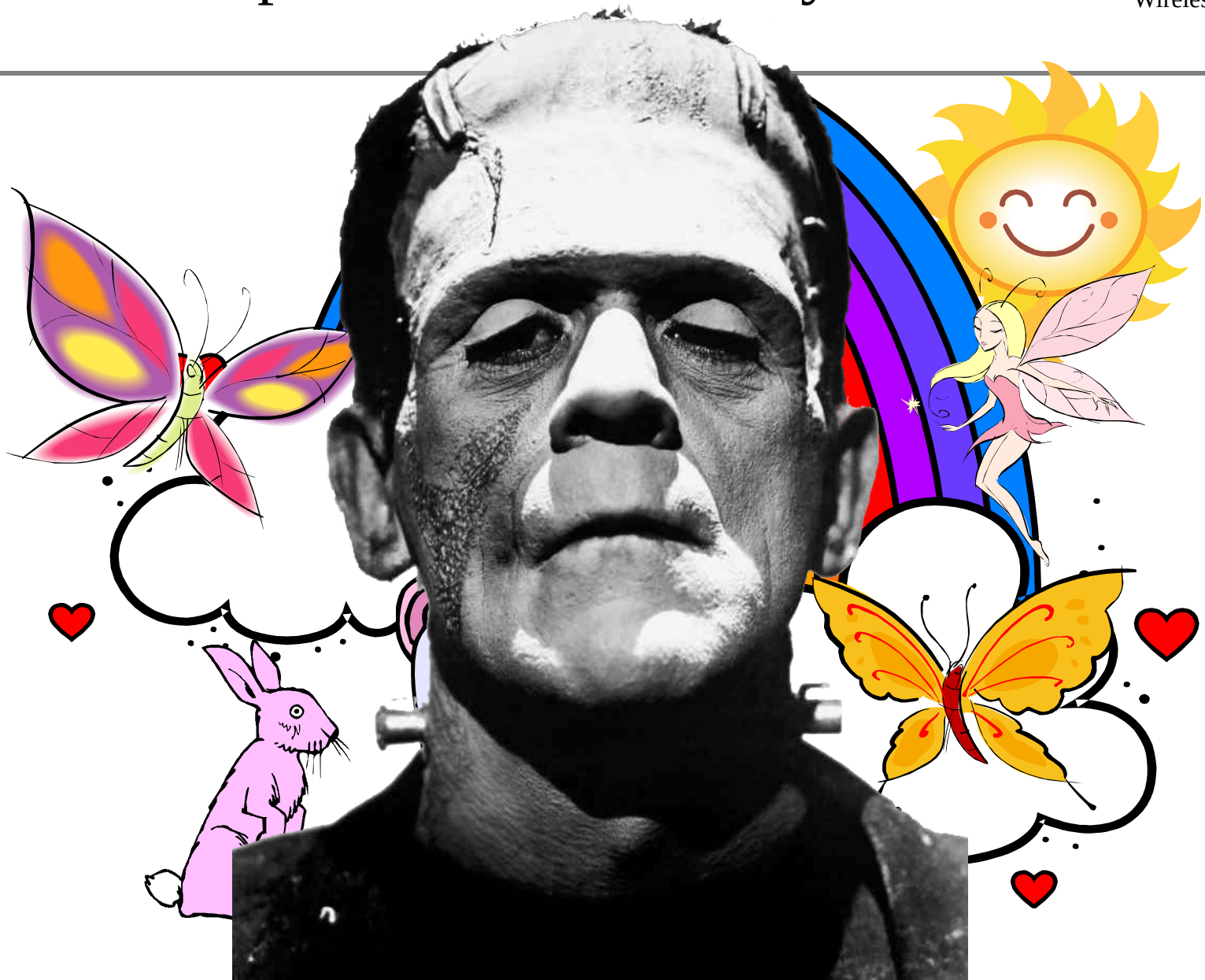
- All roads seem to be leading towards SystemVerilog
  - Great features for the DV engineer
  - A lot of talk about reuse…
- Big hope for a lot of companies is:

SystemVerilog → **Reuse** → **Productivity** →

So how's that 'reuse' thing working out for you so far???

MEDIATEK
MediaTek
Wireless, Inc.

Reuse: Expectations vs. Reality



MEDIATEK
MediaTek
Wireless, Inc.

What makes good VIP?

- Easy to Use
- Flexible

External View

- Well-designed
- Easy to maintain

Internal View

- When developing your own VIP, it's important to keep both "views" in mind.

MEDIATEK
MediaTek
Wireless, Inc.

VMM transactor?

- Extended from vmm_xactor
  - Signals → vmm_data
  - vmm_data → signals
  - vmm_data → vmm_data
- Many Uses
  - Bus Functional Models
  - Monitors & Checkers
  - Reference Models
  - Scoreboards
- This presentation will focus mainly on Master and Slave BFMs

MediaTek
Wireless, Inc.

Where to start?

- vmmgen is a nice place to start
  - Transactor configuration object
  - Reset handling
  - Basic callbacks
- But… main BFM control loop only suitable for very basic protocol, now the real work begins.

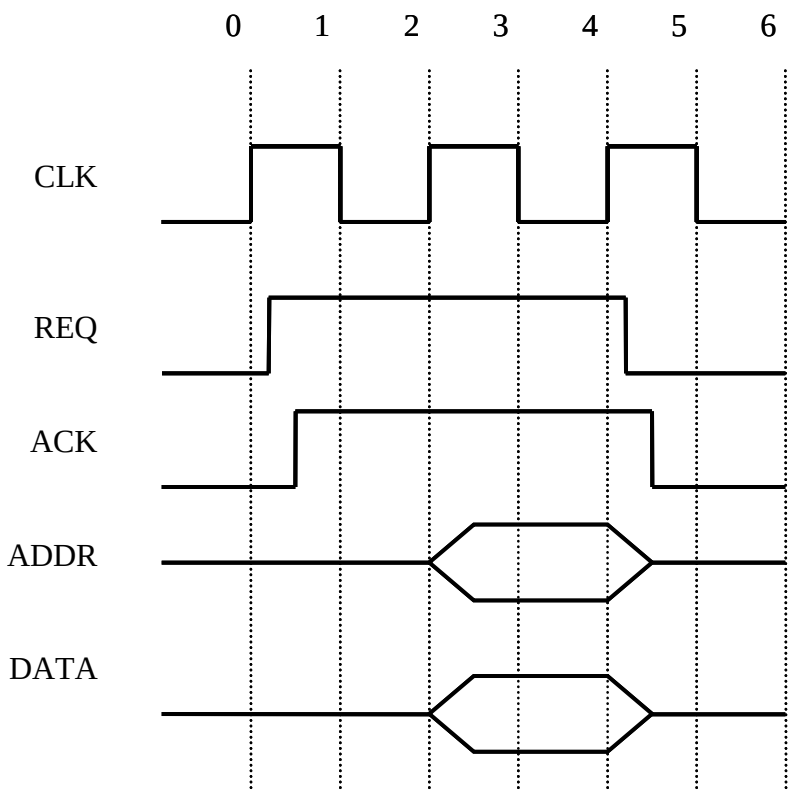
MediaTek
Wireless, Inc.

Main Control Loop

- Master Control Loop
  - Grabs transaction objects out of channel
  - Wiggles signals on the bus
  - Rinse & Repeat
- Slave Control Loop
  - Watches bus for new transactions
  - Creates a new transaction object
  - Fills the transaction object with observed bus values, responds as needed.
  - Passes completed transaction objects back to testbench environment
  - Rinse & Repeat

MediaTek
Wireless, Inc.

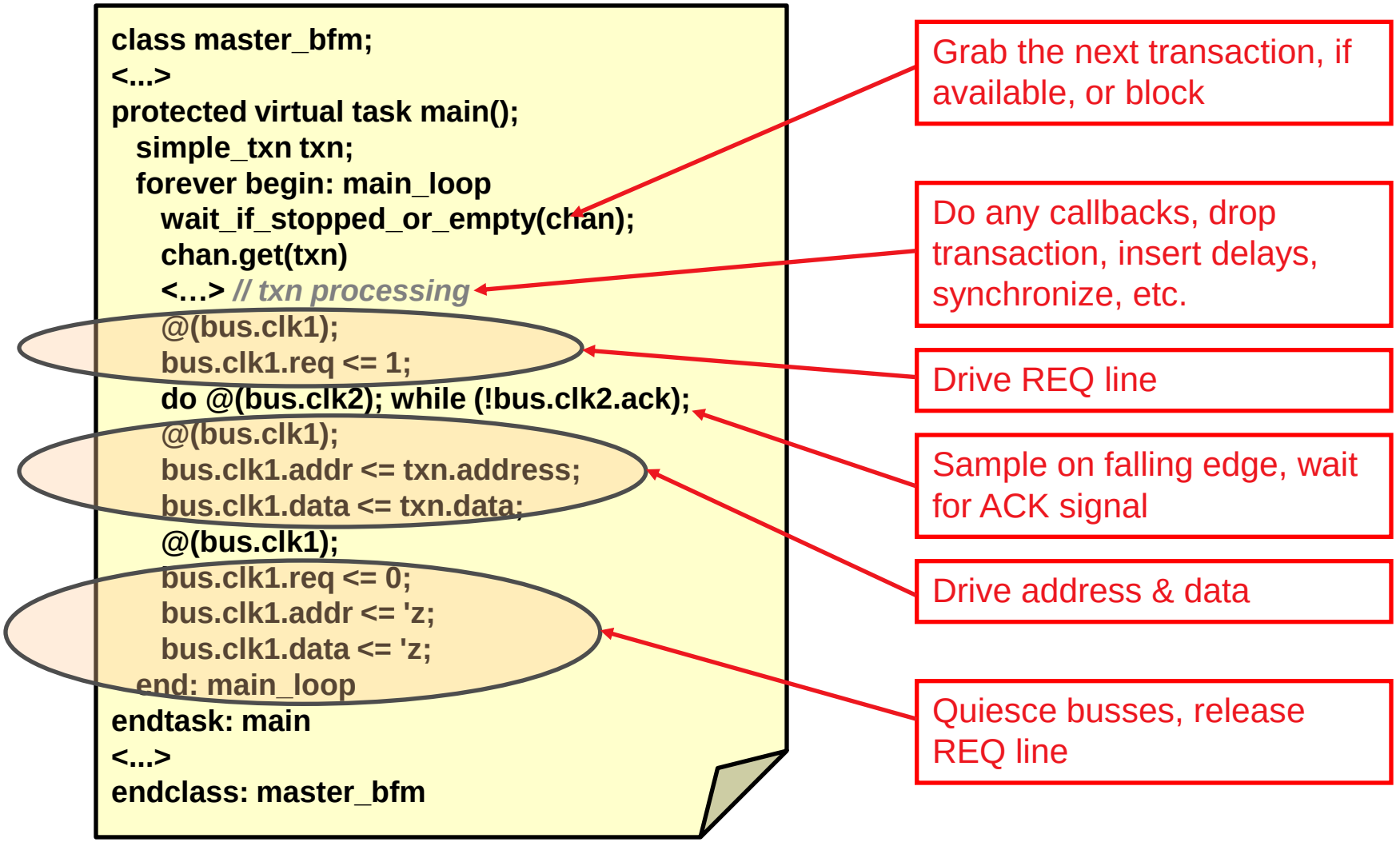
Simple Bus Transaction




```
interface simple_if (input bit clk, reset);
  wire       req;          // Request
  wire       ack;          // Acknowledge
  wire [7:0]  addr;        // Address Bus
  wire [15:0] data;        // Data Bus

  clocking clk1 @(posedge clk);
    output     #1 req, addr, data;
  endclocking

  clocking clk2 @(negedge clk);
     input     #1 ack;
  endclocking

  modport SPort (
                clocking clk1,
                clocking clk2,
                input reset
                );
endinterface: simple_if
```

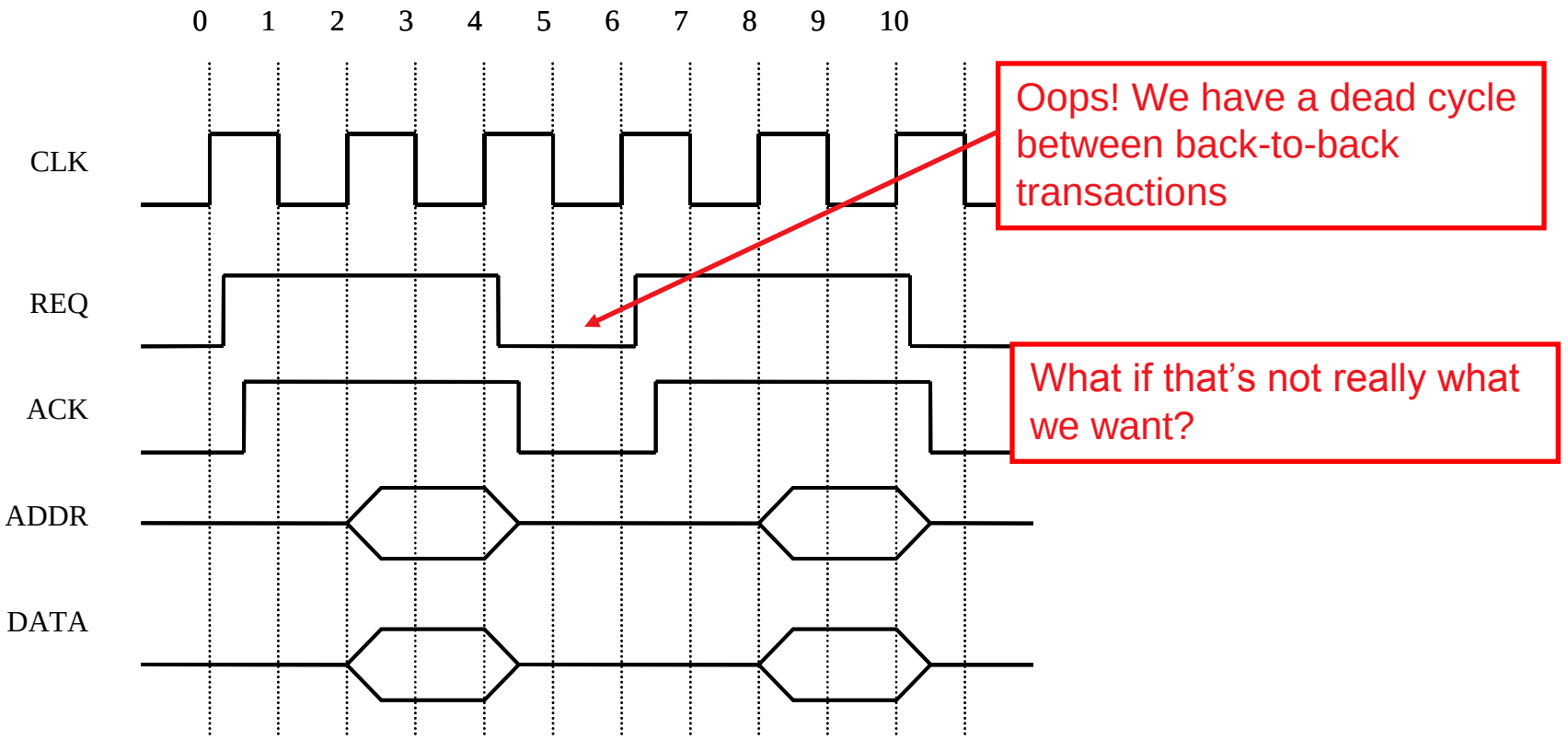
MediaTek
Wireless, Inc.

Main control thread of BFM



MediaTek
Wireless, Inc.

Back-to-back Transactions



MediaTek
Wireless, Inc.

How do we fix it?

```
class master_bfm;
<...>
protected virtual task main();
  simple_txn txn;
  forever begin: main_loop
    wait_if_stopped_or_empty(chan);
    chan.get(txn)
    <…> // txn processing
    @(bus.clk1);
    bus.clk1.req <= 1;
    do @(bus.clk2); while (!bus.clk2.ack);
    @(bus.clk1);
    bus.clk1.addr <= txn.address;
    bus.clk1.data <= txn.data;
    @(bus.clk1);
    bus.clk1.req <= 0;
    bus.clk1.addr <= 'z;
    bus.clk1.data <= 'z;
  end: main_loop
endtask: main
<...>
endclass: master_bfm
```

We could ‘peek’ ahead here, and see if there is another transaction waiting.

But what if we drop the transaction here anyway?

We could wait until the top of the loop to determine whether or not to hold the REQ line

But what if this blocks?

Any delays here might also force us to deal with the REQ line. Could get a bit messy.

MediaTek
Wireless, Inc.

Main control thread using CB Storage

```
class master_bfm;
<...>
protected virtual task main();
  simple_txn txn;
  forever begin: main_loop
    wait_if_stopped_or_empty(chan);
    chan.get(txn);
    <…> // txn processing
    bus.clk1.req <= 1;
    @(bus.clk1);
    do @(bus.clk2); while (!bus.clk2.ack);
    @(bus.clk1);
    bus.clk1.addr <= txn.address;
    bus.clk1.data <= txn.data;
    @(bus.clk2);// Wait halfway through cycle
    bus.clk1.req <= 0; // Queue possible deassert
    bus.clk1.addr <= 'z;
    bus.clk1.data <= 'z;
  end: main_loop
endtask: main
<...>
endclass: master_bfm
```

Rather than synchronize with clk1, let's stop half a cycle earlier, and queue up the quiescent bus values.

Don't worry… clocking blocks won't drive values at the wrong time.

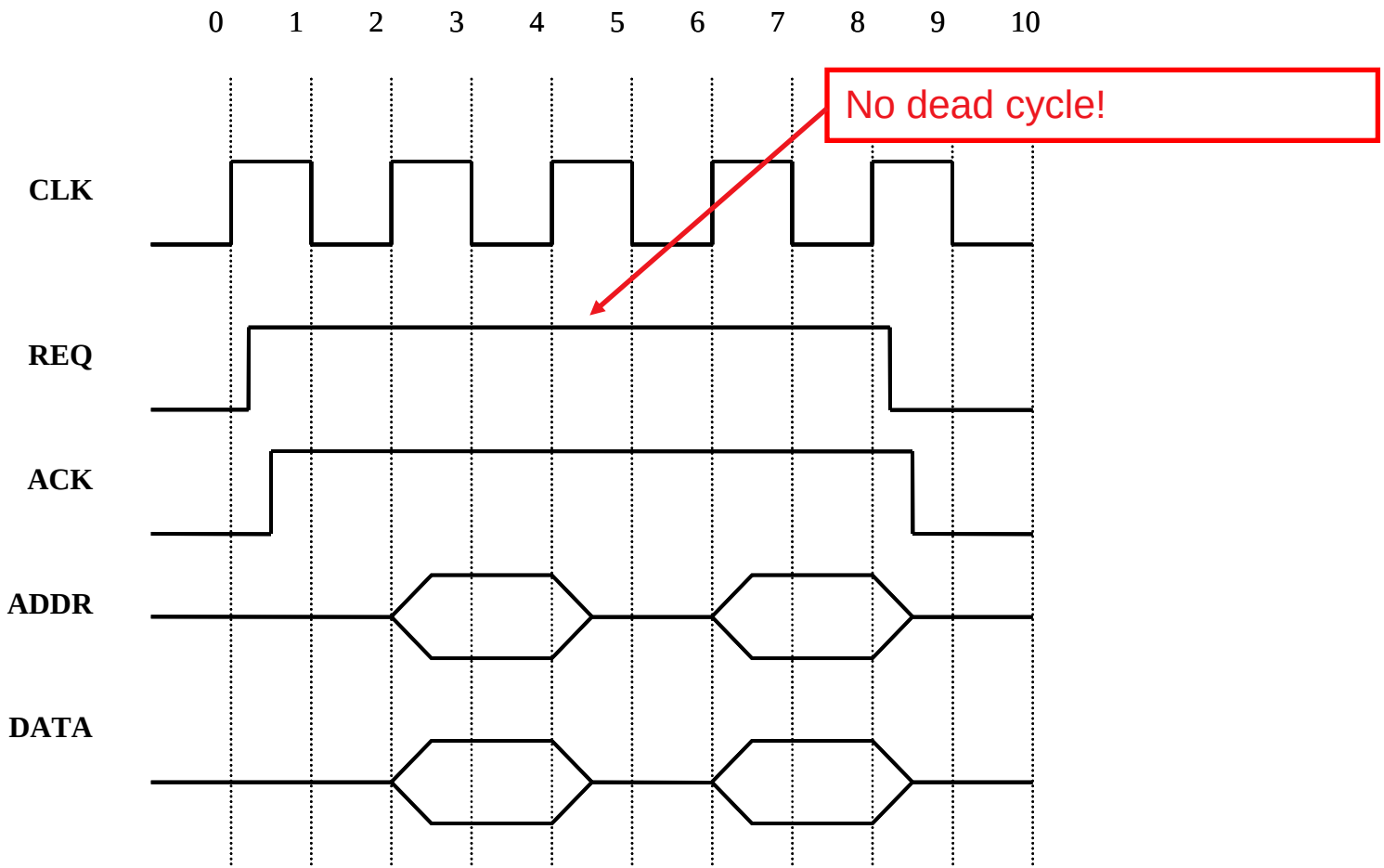
If the next transaction isn't ready, or we delay driving it for some reason, we'll deassert REQ and drive quiescent values.

If we immediately fall through to here, we'll overwrite the '0' and drive '1' for REQ

No additional lines of code!

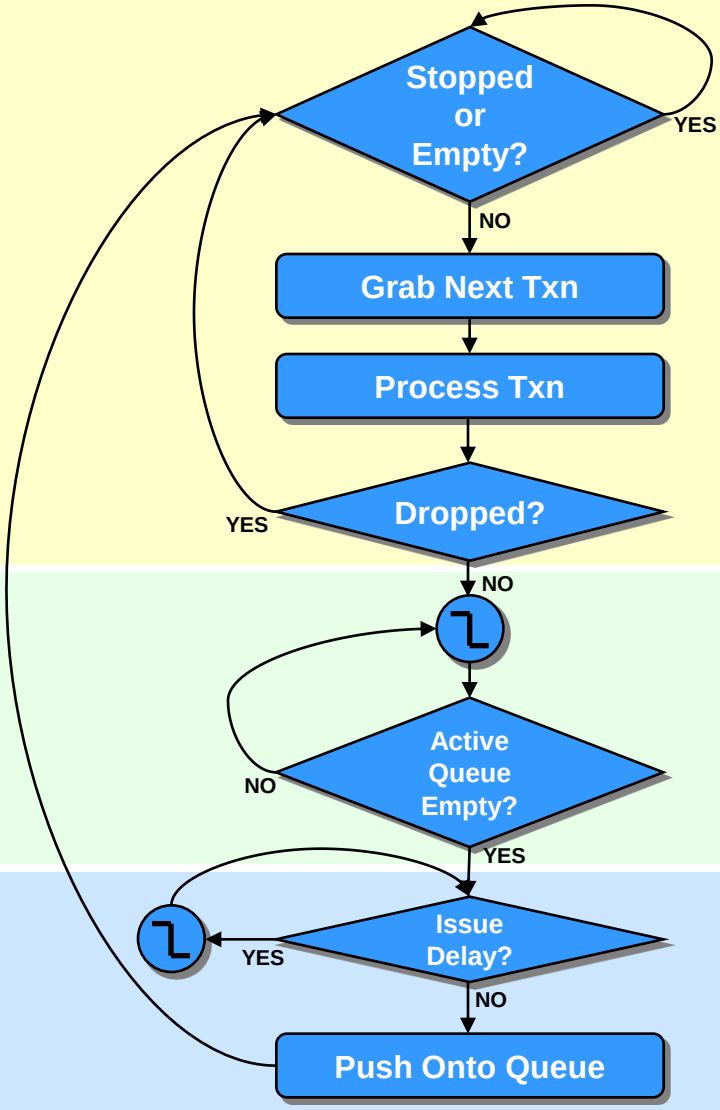
MediaTek
Wireless, Inc.

Back-to-back Transactions Without Dead Cycle



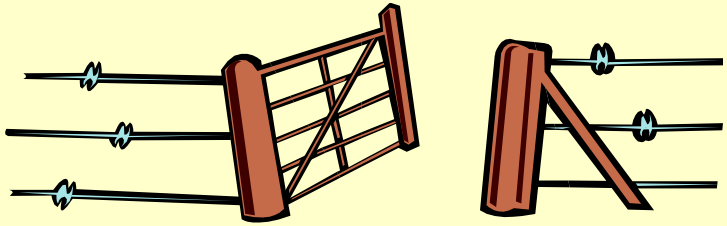
Master BFM Control Loop - Thread #1

MEDIATEK
MediaTek
Wireless, Inc.



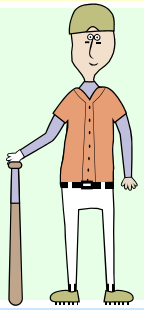
STEP 1: Qualification

"The Front Gate"



STEP 2: Transaction Pending

"The Batter's Box"



STEP 3: Issue Delay

MEDIATEK
MediaTek
Wireless, Inc.

Master BFM Control Loop - Thread #1 Code

```
forever begin: channel_loop
   wait_if_stopped_or_empty(in_chan);
   in_chan.get(next_txn);

   drop = 0;
   `vmm_callback(dspbus_master_callbacks,
                 pre_trans(this, next_txn, drop));
   if (drop) continue;

   do @(dspbus.clk_mf);
   while (txn_queue.size()>0);

   if (!cfg.ignore_issue_delay)
       repeat (next_txn.delay_issue)
           @(dspbus.clk_mf);

   txn_queue.push_back(next_txn);
end: channel_loop
```

Grab transaction from channel.

Do callback. Check to see if dropped.

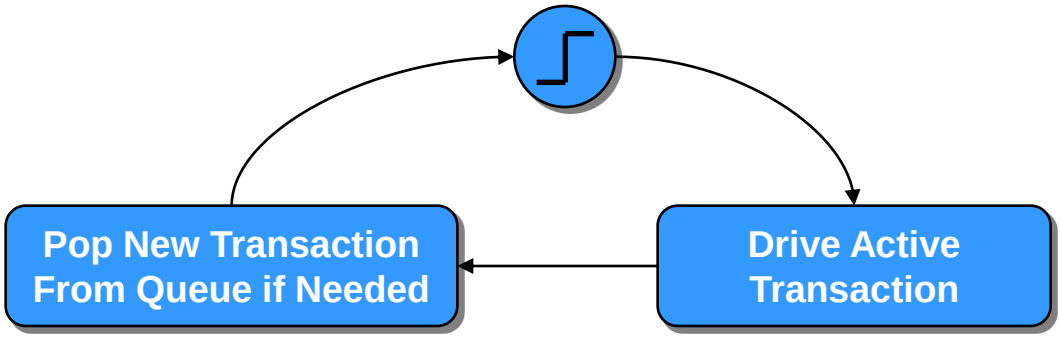
Check to see if transactor is ready for the next transaction.

Even if we're ready for the next transaction, we still need the opportunity to insert issue delay.

Push onto queue.

MediaTek
Wireless, Inc.

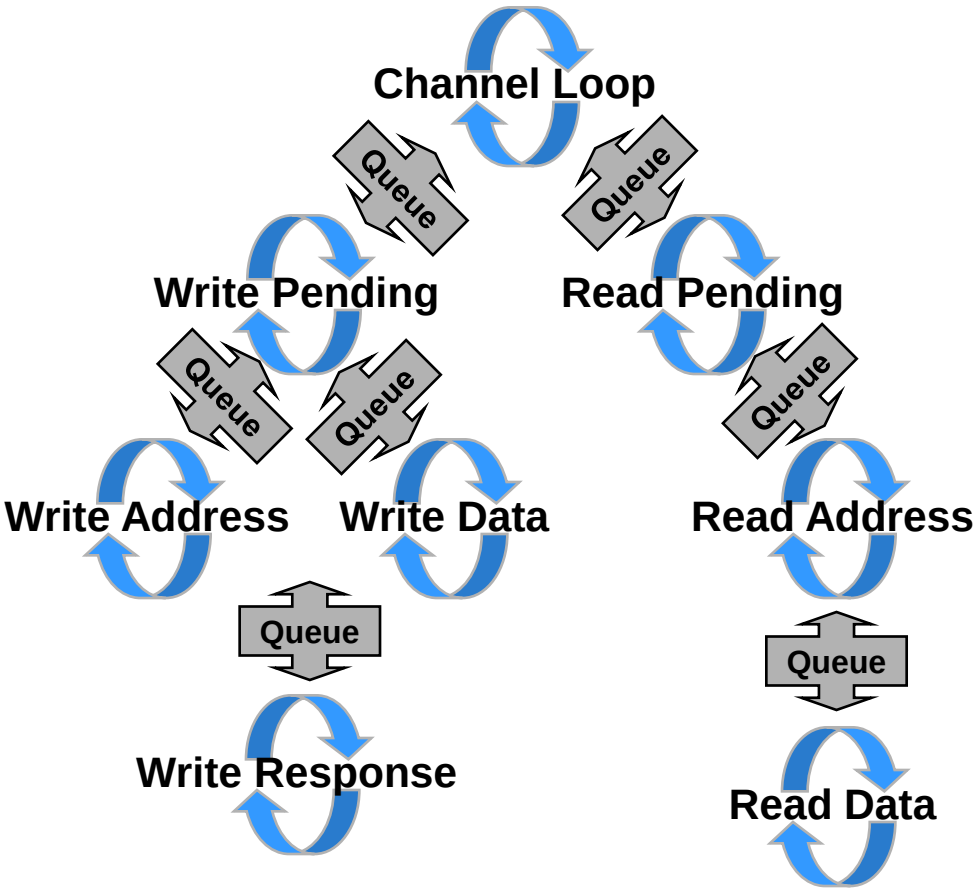
Master BFM Control Loop – Thread #2



- Use different edge when pulling transactions out of queue
  - Avoid races
- Queues more flexible than pointer for communication between threads
  - Example: Early Burst Termination for AHB causes transaction "morphing" within Thread #2

MediaTek
Wireless, Inc.

Extending Threads & Queues

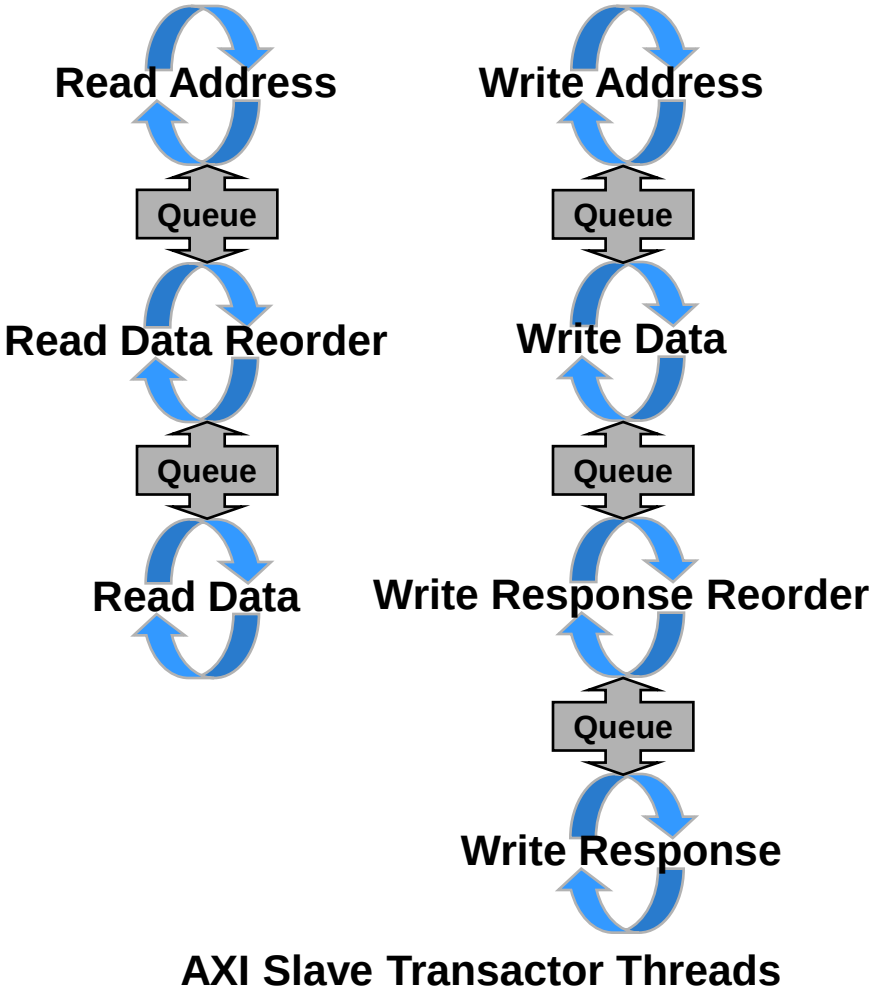


AXI Master Transactor Threads

- Thread/Queue mechanism can be extended beyond two threads if needed
- Threads don't have to synchronize to bus clocks
  - "Channel Loop" can issue multiple transactions at a time
- Queues used to manage delay between threads
  - Ex: write address and write data can start in any order

MEDIATEK
MediaTek
Wireless, Inc.

Extending Threads & Queues



- Additional threads used here to reorder transactions
- Queues can allow more than one transaction to occupy same phase
  - Ex: Data interleaving

MediaTek
Wireless, Inc.

Serializing Threads

```
task write_address_channel();
<...>
  forever begin: forever_loop
    @(posedge sigs.aclk);
    <...> // Process write address channel
    write_address_channel_complete.put(1);
  end: forever_loop
endtask: write_address_channel

task write_data_channel();
<...>
  forever begin: forever_loop
    @(posedge sigs.aclk);
     write_address_channel_complete.get();
    <...> // Process write data channel
   end: forever_loop
endtask: write_data_channel
```

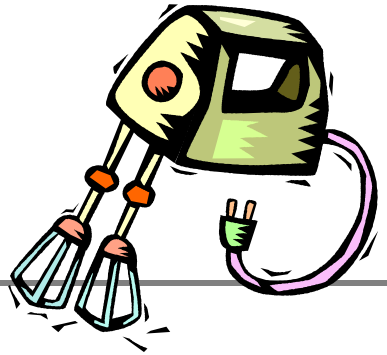
At the end of the loop, put a key into the semaphore 'bucket'.

Block here until the above loop completes and releases a key.

In this example, 'write_address_channel' will always be processed before 'write_data_channel.'

MediaTek
Wireless, Inc.

Mixing things up with randomization



- Transactor should be able to vary any aspect of a transaction that is variable
  - Bus spec should be examined carefully
  - Ex: If request line doesn't need to be held for entire burst, it should be dropped at a random point in the burst.
  - Wait states, busy states, handshaking
- Slave Response Delays
  - Randomization through transactor configuration
  - Randomization through callbacks
  - Randomization through transaction data class members

MEDIATEK
MediaTek
Wireless, Inc.

Randomizing Response Delay

```
function int StartOfBeat(AHB_txn curtrans, int curbeat);
  int hready_delay;

  // Calculate default beat delay
  if (curtrans.trans_type == AHB_txn::READ)
    hready_delay = $urandom_range(cfg.min_rd_delay,
                                  cfg.max_rd_delay);
  else
    hready_delay = $urandom_range(cfg.min_wr_delay,
                                  cfg.max_wr_delay);
  // pre_beat callback
  `vmm_callback(AHBSlaveBFM_callbacks,
                pre_beat(this,curtrans,
                         curbeat,hready_delay));

  return(hready_delay);

endfunction: StartOfBeat
```

- Methods for randomization of delay do not have to be mutually exclusive

Here we calculate delays for a beat of data based on configuration parameters.

But then we let the callback override these delays with something else if it wants to.

MediaTek
Wireless, Inc.

Randomization of delay between transactions

- In addition to randomization of delays *within* the transaction, randomization of delay *between* transactions is important
  - Don’t force the testbench environment to deal with this
  - This can be handled easily by the master transactor

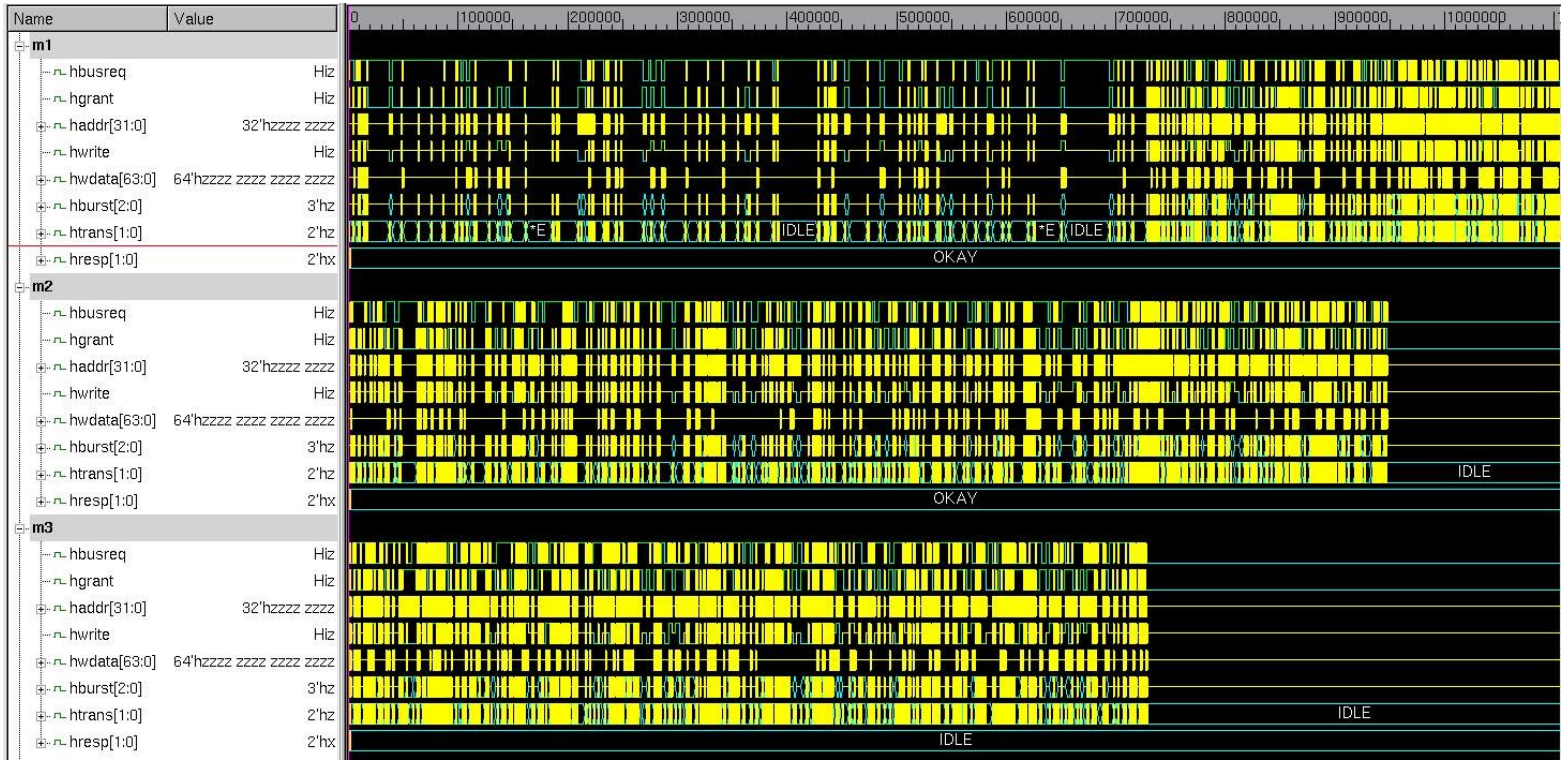
MediaTek
Wireless, Inc.

Issue delay randomization

- Here multiple masters are generating randomized transactions.
  - Transaction timings fully randomized.
  - No randomized delay between transactions.
- Interesting corner case, but is this really what you want?

MEDIATEK
MediaTek
Wireless, Inc.

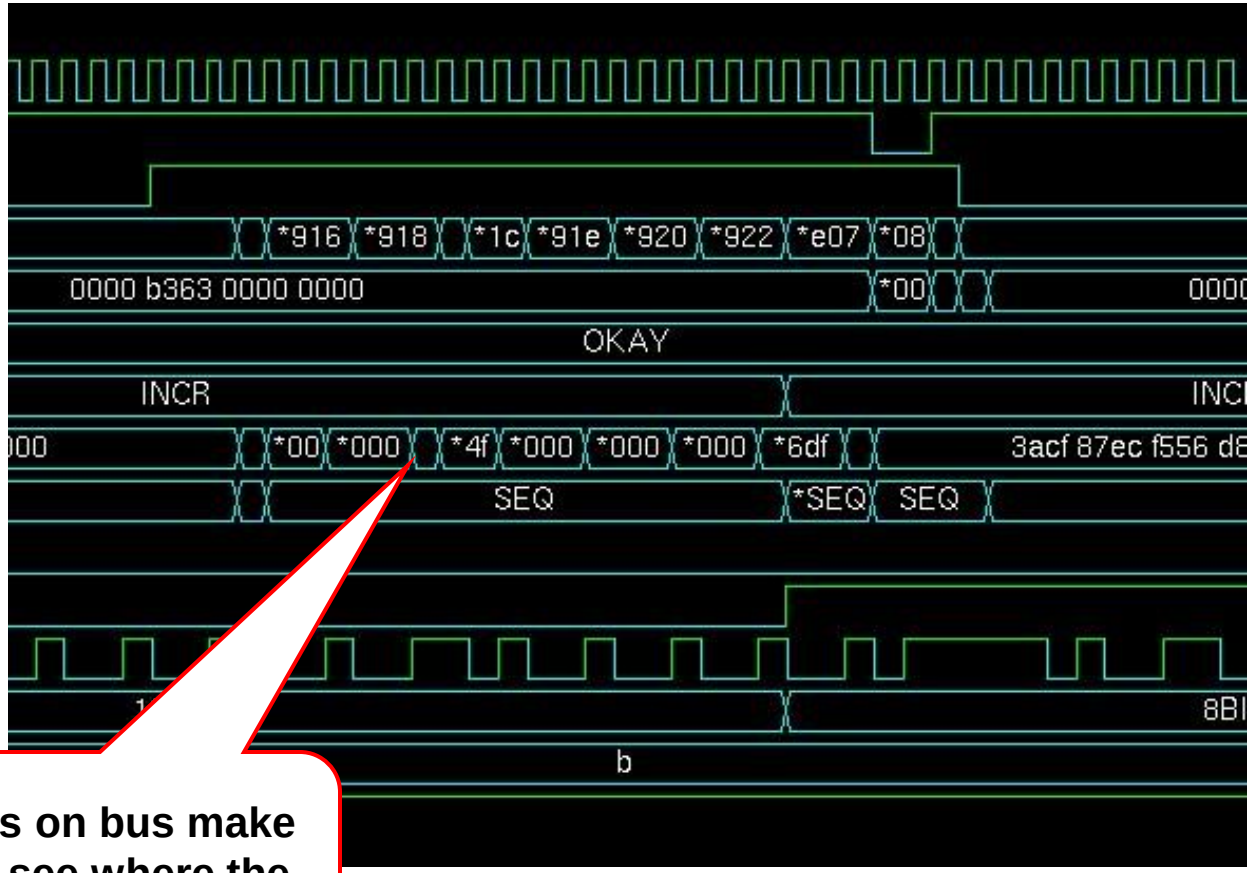
Issue delay randomization



- Here transactions are issued with random spacing.
  - Done inside the master transactor.
- Lower priority transactions can now sneak through.
  - Probably more interesting for system testing.

MediaTek
Wireless, Inc.

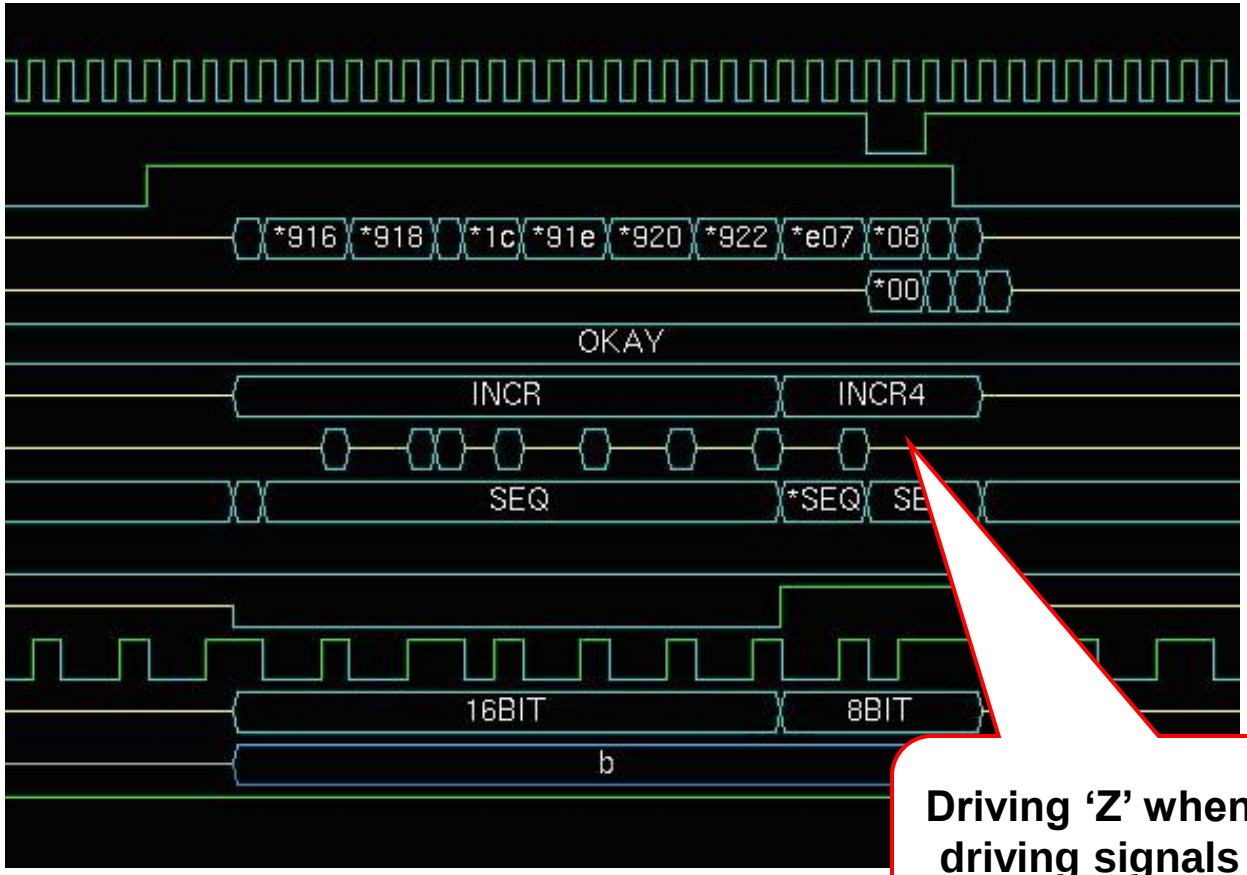
Debugging waveforms



Old values on bus make it hard to see where the transaction is.

MEDIATEK
MediaTek
Wireless, Inc.

Debugging waveforms



Driving ‘Z’ when not actively driving signals helps clean up the waveform.

MediaTek
Wireless, Inc.

Debug Features

- Have option to drive a ‘Z’ whenever not actively driving a signal
  - Cleans up waveforms, helps debug
- Not just esthetic, can catch RTL ‘looking’ at the bus when it’s not supposed to
- Additional option should be provided to randomize the bus signals when not actively driving a signal

MediaTek
Wireless, Inc.

Ready “out of the box”

- Transaction objects should come with a set of “reasonable” constraints
  - Puts upper bounds on delays
  - Prevents ‘illegal’ transactions
  - Should be able to .randomize() your vmm_data object and use it

MEDIATEK
MediaTek
Wireless, Inc.

No Hacks!

Top five ways to keep people from needing to “hack” on your VIP…

1. Callbacks
2. Use factory patterns for new transactions
3. Notifications
4. Debug/Trace information
5. Full randomization of transactions

MediaTek
Wireless, Inc.

Conclusion

- Developing verification IP is a lot of work, and can consume much of your DV resources.
  - Need to do it right
- Reuse is still a goal (unicorns and rainbows?)
- For people to want to reuse, it must meet the following criteria:

1. Easy to use
   - Minimize effort ‘out of the box’
2. Flexible
   - Must meet both foreseen and unforeseen requirements.
3. Well Designed
4. Easy to Maintain

MediaTek
Wireless, Inc.

Questions?

Thank You!

Email: kelly.larson@mediatek.com