TDD For Embedded Systems...
All The Way Down To The Hardware

Neil Johnson
XtremeEDA
What Do I Mean By Hardware

- **ASIC**
  - Application Specific Integrated Circuit
  - Static structure
  - Digital or mixed signal
  - High NRE/Low cost
- **FPGA**
  - Field Programmable Gate Array
  - Reprogrammable structure
  - Primarily digital
  - No NRE/High cost
- **SoC**
  - Either of the above + embedded processor(s) + software
SoC Development Basics

- Typical SoC design flow
  - Specification
  - Design
  - Verification
  - Physical design
  - Fabrication
  - Validation
  - Integration

Pre-silicon

Production

Software

- Documentation
- Code
- “Stuff”
- Chip
- Board
- System
- OS
- Drivers
- Application
Hardware Design and Test in 2 Easy Steps

6-12 months
Hardware Design and Test in 2 Easy Steps

8-24 months
Design and Verification

- Throw it over the wall dynamic
  - Code then test
  - Very little (no) unit testing
// the purpose of this component is to raise an objections
// at pre_reset so that other components registered to the
// same domain never proceed beyond that point

class svunit_idle_uvm_component extends uvm_component;

`uvm_component_utils(svunit_idle_uvm_component)

function new(string name = "", uvm_component parent = null);
    super.new(name, parent);
endfunction

function void phase_started(uvm_phase phase);
    if (phase.get_name() == "pre_reset") begin
        phase.raise_objective(null);
    end
endfunction

endclass
Language Basics (Design)

```haskell
// calculate the waddr to the memory based on the // frame position flags
always @(negedge rst_n or posedge clk) begin
    if (rst_n) begin
        next_waddr <= 0;
    end
    else begin
        if (calcs_strobe && first_row_flag && first_column_flag) begin
            next_waddr <= EFFECTIVE_WIDTH;
        end
        else if (calcs_strobe && last_row_flag && first_column_flag) begin
            next_waddr <= next_waddr + EFFECTIVE_WIDTH;
        end
        else if (calcs_strobe && first_row_flag && !first_column_flag ||
                 calcs_strobe && last_row_flag && !first_column_flag ||
                 strobe_3_of_4) begin
            next_waddr <= next_waddr - EFFECTIVE_WIDTH;
        end
        else if (strobe_2_of_4 || strobe_2_of_4 || strobe_4_of_4) begin
            next_waddr <= next_waddr + EFFECTIVE_WIDTH + 1;
        end
        else if (next_wr) begin
            if (next_waddr < 6 * EFFECTIVE_WIDTH-1) next_waddr <= next_waddr + 1;
            else next_waddr <= 0;
        end
    end
end
```
Hardware Simulation

```java
// simulate the wads to the memory based on the
// frame position file

// ...

Chips Freepage con:0 or generate clock: begin
if (Clock_A = 1)
    begin
        new_result = 0;
    end
else
    begin
        new_result = 0;
    end
end

// ...
```

```
class avuni_idle_vcmp_extends uvm_component;

function new(string name = "", uvm_component parent = null);
    super.new(name, parent);
endfunction

function void phase_startd(uvm_phase phase);
    if (phase.get_name() == "pre_reset") begin
        phase.raise_objectection(null);
    end
endfunction

endclass
```
Development Timeline

- Block Level design
- Block Level Testbench
- Block Level Testing
- Top Level Testbench
- Top Level Testing
- Start
- Done
Effort Spent In Verification

Mean time verification engineers spends in different task

Non-FPGA Designs

Wilson Research Group and Mentor Graphics
2010 Functional Verification Study, Used with permission

© 2011 Mentor Graphics Corp. Company
Effort and Results
Mean time Non-FPGA verification engineers spends in different tasks

- 36% Test Planning
- 16% Testbench Development
- 23% Creating and Running Test
- 22% Debug
- 4% Other

More time spent in debug than any other task!

Wilson Research Group and Mentor Graphics, 2012 Functional Verification Study, Used with permission
Where FPGA Verification Engineers Spend Their Time

- **Test Planning**: 43%
- **Testbench Development**: 19%
- **Creating Test and Running Simulation**: 20%
- **Debug**: 13%
- **Other**: 4%

TDD is a great idea that can save us a lot of money by helping us avoid writing buggy code.
TDD Mechanics

1. Write a test
2. Run the test (must fail)
3. Implement your design
4. Run the test (must pass)
5. Repeat
Each module in a design has a list of dedicated unit tests
  – tests exhaustively cover functionality of the module
    (“exhaustively” within reason)
  – unit tests are automated
    • anytime the code changes for a given module, the corresponding tests are run to make sure it isn’t broken
Unit tests informally increase quality, not red tape
  – tests are planned/written while building a component
  – documentation/tracking requirements are very light
“Use-this” modules are integrated when their unit tests pass
Unit testing happens before anything else
  – then block and/or top level testing
  – Unit testing does not replace current verification practices
    • change... perhaps. replace... no.
We're too busy to save money.

This guy is an idiot.

It's lunch time. Why are you telling us this at lunch time?
My Commitment to Agile Hardware development is to...
Build a platform to showcase TDD as a credible technique for doing hw/sw co-development. This is why I'm making this commitment... code quality and synchronization between hw developers are of major importance in embedded systems development. I think this platform could be a great tool for addressing both issues; TDD being applied in both domains on a platform that encourages and enables collaboration.
This is how I intend to meet my commitment... we'll use an FPGA board to implement some simple yet visual application. The app will be split between SW running on an ARM Cortex A9 and logic on the FPGA. We'll build the code on both sides w/TDD and likely Googletest for the C++ and SystemC unit on the software side. For people that use the platform, they'll have access to all the code so they can add/change/move features in or between sw/hw domains. Then they can rebuild/deploplay and watch it go. Ease of use is top priority. We want people to be comfortable playing w/it. This is how I'll know if I've achieved my commitment...
I'll take it to Agile 2014 and set it up in the coaching area. If I get 10 people sitting down, changing code, re-ship and re-deploy then I'm happy. less than 10 will be a miserable failure!!
I plan to meet my Commitment by...
Agile 2014 conference is July 28th. Done and polished before then.

PS: "We" is me and Soheil. This is a 2-man commitment! — Scott Johnson

Sohil Salihian April 16/2014

“...showcase TDD as a credible technique for doing hw/sw co-development.”

“code quality and synchronization between hw/sw developers are of major importance in embedded systems development.”

“we'll use an FPGA board to implement some simple yet visual application.”
AgileSoc.com

Agile Hardware w/TDD

Programming Steps

I
- ARM
- Program FPGA
- Run SW
- Erase

II
- ARM
- Cloud
- New Life

Application

< 2 neighbours
- Underpopulated

2-3 neighbours
- Sustained
- Overpopulated

> 4 neighbours
- 3 neighbours
Hardware Deliveries

Step 0 – Wires

Step 1 – FIFO

Step 2 – Frame Processor

C++
  * Google test
  * Google mock
  * Mock/isolate class interaction

C/C++
  * Google test
  * Google mock
  * Mock/isolate driver interaction

Verilog
  * SV Unit
  * Verilog
  * IP
Hardware Deliveries

- Step 0 – Wires
  - placeholder for future development

- Step 1 – FIFO
  - ingress/egress functions to a qpram

- Step 2 – Frame Processor
  - add a shadow around living cells
  - non-trivial hardware block built with TTDD

TDD'ing Hardware

C++
- *GoogleTest
- *GoogleMock
- *Mock/isolate class interaction

C/C++
- *GoogleTest
- *GoogleMock
- *Mock/isolate driver interaction

Verilog
- *SV Unit
- *Verilogs
- *IP

Agile HW

Next page
Hardware Deliveries

• Step 0 – Wires
  – placeholder for future development

• Step 1 – FIFO
  – ingress/egress functions to a qpram

• Step 2 – Frame Processor
  – add a shadow around living cells
  – non-trivial hardware block built w/TDD

Agile HW 0.1 (wires) Last year

Agile HW 0.1 (fifo) QPRAM (refactored)

QPRAM
  - logic
  - fifo

Agile HW 0.2 (shadow)

Pull
  → calc
  → QPRAM

This year

* shadow calculation
* handshake
* back pressure
* flow control

* meets TAPI / functional limitation
* too slow
* didn’t exist
Problem: Low Level Hardware Interactions
Problem: Low Level Hardware Interactions

```vhdl
`SVTEST(ingress_write_1_pixel)
 @(posedge clk);

iVALID = 1;
iDATA = 'haa55bb;
iUSER = 1;
iKEEP = 'hb;
iLAST = 0;

@(posedge clk);

#1;

`FAIL_UNLESS(wdata == {1, 'hb', 0, 'haa55bb});
`FAIL_UNLESS(waddr == 0);
`FAIL_UNLESS(wr == 1);
`SVTEST_END
```
Solution: Create Higher Level API

```vhdl
```

```vhdl`
svtest(ingress_write_1_pixel)
setIngressPixel('haa55bb);
step();
expectRamWrite(0, 'haa55bb);
```
vtest
```vhdl`
svtest(ingress_write_1_pixel)
@(posedge clk);

iVALID = 1;
iDATA = 'haa55bb;
iUSER = 1;
iKEEP = 'hb;
iLAST = 0;

@(posedge clk);

#1;

\`FAIL UNLESS(wdata === \{ 1, 'hb, 0, 'haa55bb \});
\`FAIL UNLESS(waddr == 0);
\`FAIL UNLESS(wr === 1);
```
vtest
```
```
Problem: Not Structured for Unit Testing

Memory

ingress() -> FIFO -> expect

egress()

test path
Problem: Not Structured for Unit Testing

A unit test should fail for exactly 1 reason

ingress() -> FIFO -> expect egress()
Solution: Restructure/Isolate Functionality

A unit test should fail for exactly 1 reason.
Problem: It’s Not Over With TDD

ingress() → FIFO → Memory → egress()

expecting
Solution: Integration Tests For Integration
Problem: Hardware is Multi-Threaded

```
SVTEST(test_with_3_threads)
// synchronize to clk thread
@(negedge clk);

fork
  begin
    // some action on input thread
  end

begin
  // wait for response thread
  `FAIL_IF(someCondition);
end
join
`SVTEST_END
```
Solution: Maintain a Single Thread

```plaintext
SVTEST(ingress_write_1_pixel)
setIngressPixel('haa55bb);
step();
expectRamWrite(0, 'haa55bb);
SVTEST_END
```
Problem: Design Doesn’t Meet Timing

ingress() → FIFO → Memory → expect egress()
O PRAM (refactored)

* didn't exist

* too slow

* meets timing

/ functional limitation

<table>
<thead>
<tr>
<th>PLU</th>
<th>dpram</th>
<th>dpram</th>
<th>dpram</th>
</tr>
</thead>
</table>

f0

ram
Problem: No xUnit Framework

- Software developers have a lot of options
- Unit testing isn’t a common hardware technique
  – we had no xUnit framework
Solution: SVUnit Testing Framework

Perl scripts and Makefiles

Test Runner Object

Test Suite Objects

Unit Test Objects

Test Methods

write code here

test_<a>()
test_<b>()
Where Verification Engineers Spend Their Time

- 20% Being Productive
- ~15% Debug
- 65% Riding Bikes
Vlad... Glenn... take a look at this guy's blog. His name is Neil Johnson and his blog is called AgileSoC.com.

Applying agile software practices in hardware development?

What?

This is tragic.

It reads like a cry for help.

Thanks, eh.

AgileSoC.com
Agile Hardware is real... and it's partly my fault.
nosnhojn@gmail.com
@nosnhojn