Tools and Tribulations:
Python, Vivado and just a little less MATLAB

Jack Hickish

Cavendish Laboratory, University of Cambridge
I think we should start seeing other people

The CASPER toolflow?

A "Plan"

A Python-based toolflow implementation

Current Status
I don’t like MATLAB.
It’s not you, MATLAB, it’s me

>> myarray(0)
Subscript indices must either be real positive integers or logicals.
Mostly...

```
jack@centosvm:~$ /usr/libexec/devel$ ./startsg

License checkout failed.
MATLAB is unable to connect to the license server. 
Check that the license manager has been started, and that the MATLAB client machine can communicate with the license server.

Troubleshoot this issue by visiting:
http://www.mathworks.com/support/lme/R2012b/15

Diagnostic Information:
Feature: MATLAB
License path: /home/jackh/.matlab/R2012b_licenses:/opt/MATLAB/R2012b/licenses/license.dat:/opt/MATLAB/R2012b/licenses/network.lic
Licensing error: -15,570. System Error: 115
```

Actually, sometimes it is you
But I like the CASPER toolflow

- One-click compile
- Shared memory management
- Parameterized libraries
What is the CASPER toolflow?

- Simulink frontend – User creation
- Matlab middleware – yellow block creation, EDK project creation
- EDK project generation
- ISE backend
What I don’t like about the CASPER toolflow

- Simulink frontend.
  - Few people know how to make yellow blocks. I blame MATLAB OOP.
  - Deprecated code / standards (a small irritation)
  - EDK isn’t well utilised, and adds a layer of code wrapping.
  - I want to use Vivado!
What I don’t like about the CASPER toolflow

- Simulink frontend.
- Few people know how to make yellow blocks. I blame MATLAB OOP.
- Deprecated code / standards (a small irritation)
- EDK isn’t well utilised, and adds a layer of code wrapping.
- I want to use Vivado!
What I don’t like about the CASPER toolflow

- Simulink frontend.
- Few people know how to make yellow blocks. I blame MATLAB OOP.
- Deprecated code / standards (a small irritation)
- EDK isn’t well utilised, and adds a layer of code wrapping.
- I want to use Vivado!
What I don’t like about the CASPER toolflow

- Simulink frontend.
- Few people know how to make yellow blocks. I blame MATLAB OOP.
- Deprecated code / standards (a small irritation)
- EDK isn’t well utilised, and adds a layer of code wrapping.
- I want to use Vivado!
What I don’t like about the CASPER toolflow

- Simulink frontend.
- Few people know how to make yellow blocks. I blame MATLAB OOP.
- Deprecated code / standards (a small irritation)
- EDK isn’t well utilised, and adds a layer of code wrapping.
- I want to use Vivado!
A "Plan"

- Leave the Simulink frontend (for now)
- Distinct, separate toolflow middleware
- Generate plain verilog
- Compile with whatever you want (Vivado for 7 series)
A "Plan"

- Leave the Simulink frontend (for now)
- Distinct, separate toolflow middleware
  - Generate plain verilog
  - Compile with whatever you want (Vivado for 7 series)
A "Plan"

- Leave the Simulink frontend (for now)
- Distinct, separate toolflow middleware
- Generate plain verilog
- Compile with whatever you want (Vivado for 7 series)
A "Plan"

- Leave the Simulink frontend (for now)
- Distinct, separate toolflow middleware
- Generate plain verilog
- Compile with whatever you want (Vivado for 7 series)
A "Plan"

All python, launches MATLAB/Vivado when necessary
Implementation

- Frontend (Simulink) separate from rest of flow

```plaintext
BEGINBLOCK test/sw6
Name=sw6
FullPath=test/sw6
Tag=xps:sw_reg
mode=one value
io_dir=From Processor
io_delay=0
sample_period=1
names=reg
bitwidths=32
arith_types=0
bin_pts=0
sim_port=on
show_format=off
ENDBLOCK test/sw6
```
Implementation

- XPSblock-like objects, which add modules / signals / ports / wishbone interfaces to the top-level verilog template.

```python
from yellow_block import YellowBlock
from verilog import VerilogInstance

class sw_reg(YellowBlock):
    def set_dependencies(self):
        self.add_requirement('wb_clk')

    def modify_top(self, top):
        if self.blk['io_dir'] == 'To Processor':
            module = 'wb_register_simulink2ppc'
            inst = VerilogInstance(entity=module, name=self.fullname, comment=self.fullname)
            inst.add_wb_interface(nbytes=4)
            inst.add_port('user_clk', 'userclk')
            inst.add_port('user_data_in', '%s_user_data_in' % self.fullname)
            top.add_instance(inst)
            top.add_signal('%s_user_data_in' % self.fullname, width=32, comment='sw_reg_hookup')
        elif self.blk['io_dir'] == 'From Processor':
            module = 'wb_register_ppc2simulink'
            inst = VerilogInstance(entity=module, name=self.fullname, comment=self.fullname)
            inst.add_wb_interface(nbytes=4)
            inst.add_port('user_clk', 'userclk')
            inst.add_port('user_data_out', '%s_user_data_out' % self.fullname)
            top.add_instance(inst)
            top.add_signal('%s_user_data_out' % self.fullname, width=32, comment='sw_reg_hookup')
```
Implementation

- End product – complete top-level verilog file with automated, parameterized wishbone interfaces.
Where I’m at

Current status
- Overall flow complete
- Skeletal base packages for KC705 and SNAP boards
- Wishbone infrastructure complete
- software registers, brams, gpio & adc16 blocks done or nearly done

Still to do
- All the other yellow blocks!
- core_info.tab generation and some sort of bof-like bitstream wrapping
Isn’t this just mibuild?

I used the base-package model, would it be better to generate all code from scratch?

Single verilog library, or one per platform like the current toolflow – add files on demand?

What about the frontend?
Lessons learned / Questions posed

- Isn’t this just mibuild?
- I used the base-package model, would it be better to generate all code from scratch?
- Single verilog library, or one per platform like the current toolflow – add files on demand?
- What about the frontend?
Lessons learned / Questions posed

▶ Isn’t this just mibuild?
▶ I used the base-package model, would it be better to generate all code from scratch?
▶ Single verilog library, or one per platform like the current toolflow – add files on demand?
▶ What about the frontend?
Lessons learned / Questions posed

- Isn’t this just mibuild?
- I used the base-package model, would it be better to generate all code from scratch?
- Single verilog library, or one per platform like the current toolflow – add files on demand?
- What about the frontend?
Questions?
Thoughts?
Advice?
Criticism?