Implementing the OTTER with PyMTL3

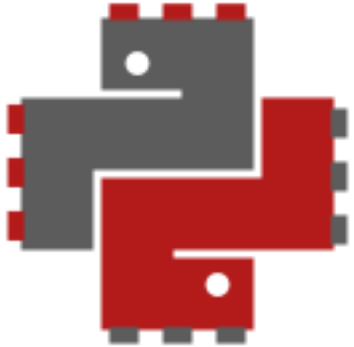
Curtis Bucher

CPE333

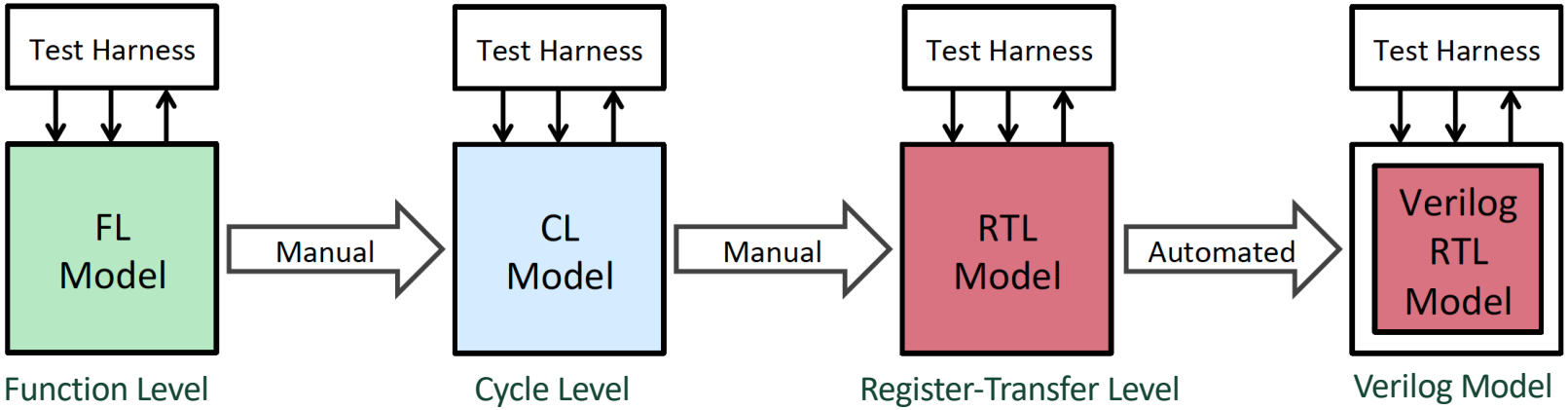
CAL POLY

PYMTL3

- An open-source Python framework for hardware **modelling**, **generation**, **simulation** and **verification**.
- Lightweight and cross platform, note requiring Vivado for development



PyMTL: Design Process



- FL modeling allows for the rapid creation of a working model.
- This design is *manually refined* into a PyMTL CL model that includes timing.
- Promising architectures can again be *manually refined* into PyMTL RTL to accurately model resources.
- Verilog generated from PyMTL RTL can be passed to an EDA toolflow for accurate area, energy, and timing estimates.

PyMTL3 Example: Checksum

```python
#-------------------------------------
# Checksum FL
#-------------------------------------

def checksum( words ):

  sum1 = b16(0)
  sum2 = b16(0)
  for word in words:
    sum1 = ( sum1 + word ) & 0xffff
    sum2 = ( sum2 + sum1 ) & 0xffff

  return concat( sum2, sum1 )
```

Functional

- Traditional python code
- Includes non-synthesizable python

Non-Synthesizable Python

- use of common Python data structures: set, dict, list of non-translatable constructs, etc.
- arbitrary function calls

```python
#-----------------------------------------------------
# ChecksumCL
#-----------------------------------------------------

class ChecksumCL( Component ):

  def construct( s ):

    s.recv = CalleeIfcCL( Type=Bits128 )
    s.send = CallerIfcCL( Type=Bits32  )

    s.in_q = PipeQueueCL( num_entries=2 )
    s.in_q.enq //= s.recv

    @update_once
    def up_checksum_cl():
      if s.in_q.deq.rdy() and s.send.rdy():
        bits = s.in_q.deq()
        words = b128_to_words( bits )

        result = checksum( words )
        s.send( result )
```

Cycle-Accurate

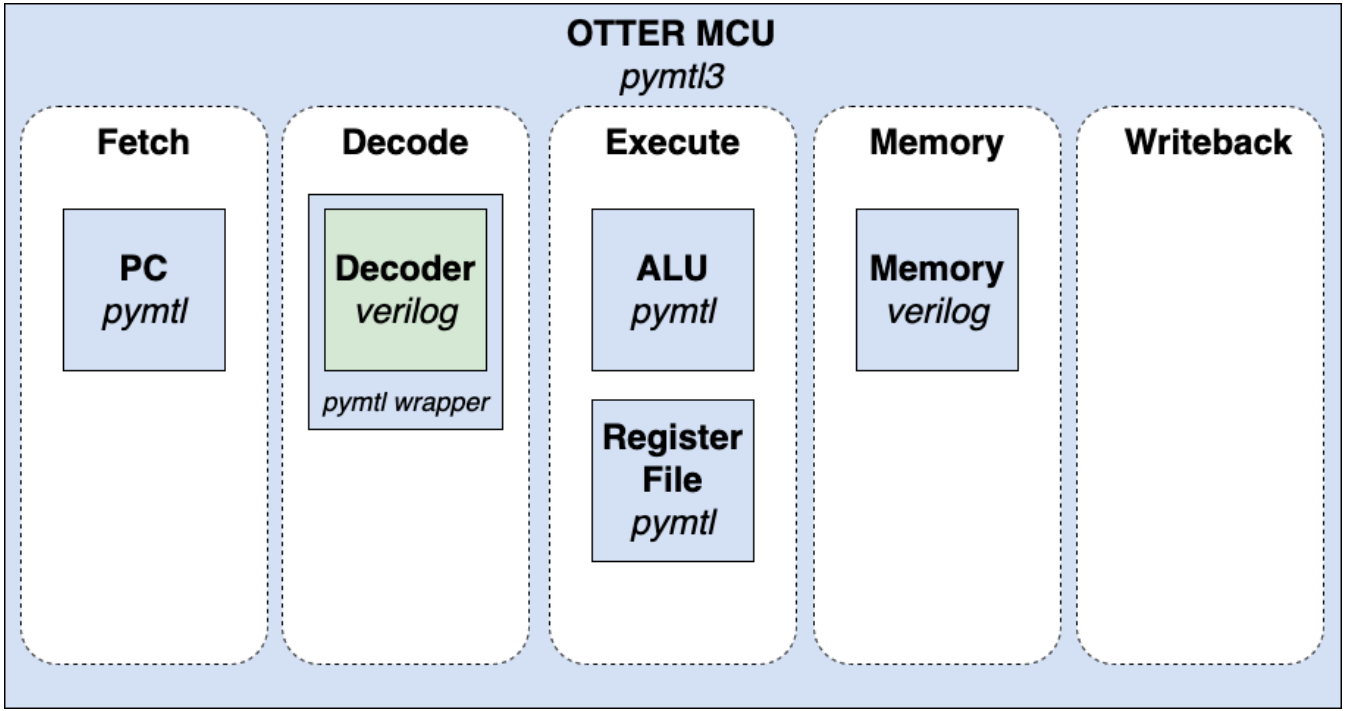
- Physical circuit represented in Python code.
- Interconnected components
- Includes non-synthesizable python

```python
#-----------------------------------------------------
# ChecksumRTL
#-----------------------------------------------------

class ChecksumRTL( Component ):

  def construct( s ):

    # Interface

    s.recv  = RecvIfcRTL( Bits128 )
    s.send  = SendIfcRTL( Bits32  )

    # Component

    s.words = [ Wire( Bits16 ) for _ in range( 8 ) ]
    s.sum1  = Wire( Bits32 )
    s.sum2  = Wire( Bits32 )

    # ... much more connecting code ... #

    @update
    def up_rtl_send():
      go = s.in_q.deq.rdy & s.send.rdy
      s.send.en     @= go
      s.in_q.deq.en @= go

    @update
    def up_rtl_sum():
      s.send.msg @= ( s.sum2 << 16 ) | s.sum1
```

Register-Transfer

- Translated into Verilog
- Only synthesizable PyMTL

Current Progress



Open-Source Tools

- **Jupyter Notebooks**

  Powerful software tool can be used with PyMTL for simple, effective experimenting, simulation, visualization etc.

- **Unittest / Pytest**

  Simple, powerful python testing frameworks allow for advanced hardware testing at every level of design (FL, CL, RTL)

- **Hypothesis**

  Powerful python framework for automatically generating testcases and extracting the simplest failing cases.

- **Numpy + python packages**

  Can be used for experimenting with FL, CL models before final RTL model.

- **Verilator**

  Powerful, lightweight open-source Verilog simulation and testing solution. Can be tightly integrated with project structure.

- **GTKwave**

  Open-source waveform viewer.

Hypothesis used to find the minimum falsifying example

```
_____ TestMemory.test_rw_word _____

self = <test_memory.TestMemory testMethod=test_rw_word>

    @given(
>       st.integers(min_value=-1, max_value=2**addr_width - dpw - 1),
        st.integers(min_value=0, max_value=2**word_width - 1),
    )

tests/test_memory.py:23:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
tests/test_memory.py:30: in test_rw_word
    self.mem.write_word(addr, data)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

s = s, addr = Bits8(0xff), data = Bits32(0x00000000)

    def write_word(s, addr: int, data: int) -> Bits:
        dpw = s.word_width // s.data_width
>       s.mem[addr : addr + dpw] = [
            data[i * s.data_width : (i + 1) * s.data_width]
            for i in range(dpw - 1, -1, -1)
        ]
E       ValueError: could not broadcast input array from shape (4,) into shape (0,)

src/cl/memory.py:25: ValueError
---------- Hypothesis ----------
Falsifying example: test_rw_word(
    addr=-1, data=0, self=<test_memory.TestMemory testMethod=test_rw_word>,
)
```

GTKWave can read .vcd files generated by PyMTL