Bus Centric Synchronous Message Exchange for Hardware Designs

Brian Vinter
Niels Bohr Institute

Motivation

When we model inherently synchronous systems that don't use external choice CSP becomes an inconvinient approach

We'd like

- Busses
- Implicit communication

Needed properties

What is needed:

Broadcasting channels

Hidden clock

Implicit latches

What helps us do it simpler:

Globally synchronous in nature

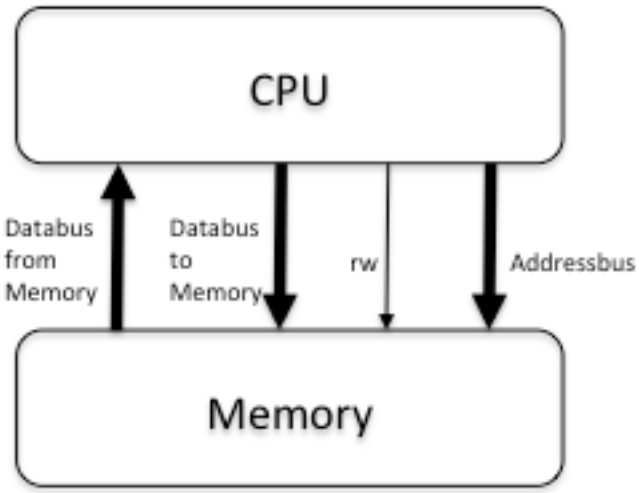
- Turns out we can relax this easily

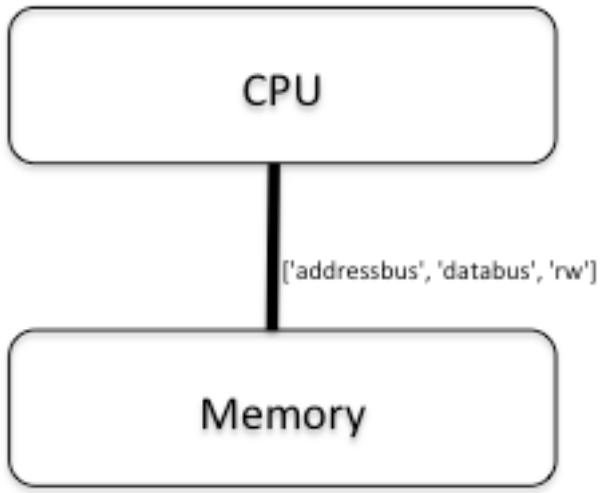
Shared nothing

'Reading ends' are guaranteed to be ready at down clock

- So external choice is not needed

Simple setup





Changes from the original SME approach

Busses are bi-directional

A bus may consist of a set of signals

Busses are now active objects that are accessed by the components

Components within other components may run at a higher clock-rate

All signals may be logged

Networks may new be rendered graphically for debugging

Elements

Bus – now an active component

Component – think of it as a process

Function – equivalent to a Componenet but not targeted for HW

Network – describes the wiring of a set of elements

The bus

Hosts a set of signals

Components access bus values directly

Reading a field that was not written raises an exception

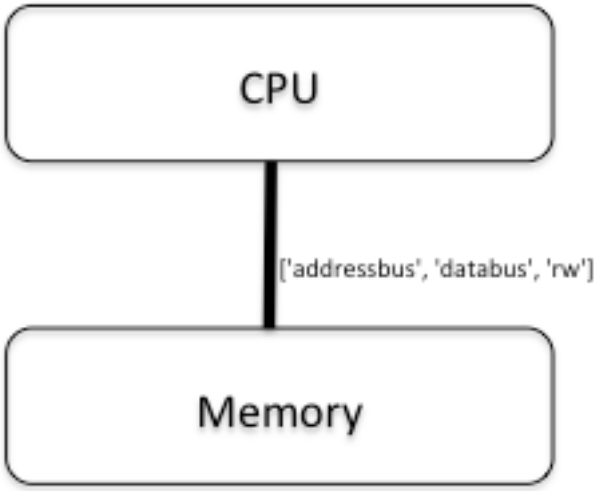
Writing to a field that was already written within the same clock-cycle raises an exception

Fields are typed, explicitly or implicitly, writing a value of a non compatible type raises an exception

The bus

```
class Bus(dict):
    def __init__(self, name, keys):
        if type(keys) != list:
            keys = [keys]
        self.name = name
        self.datatype = None

        self.readvalues = dict.fromkeys(keys)
        self.writevalues = dict.fromkeys(keys)

        self.readvalues['__SME_name__'] = self.name
        self.writevalues['__SME_name__'] = self.name

        self.log = {}
        for key in self.fields():
            self.log[key] = []

    def graph(self, prefix):
        self.graphname = prefix+self.name
        graph = self.graphname+'[label='+self.name+', shape=Mrecord, style=filled, color=grey];\n'
        return graph

    def __getitem__(self, key):
        if self.readvalues[key] == None:
            raise ReadConfict(self.name,key)
        return self.readvalues[key]

    def __setitem__(self, key, value):
        if self.writevalues[key] != None:
            raise WriteConfict(self.name,key)
        self.writevalues[key] = value

    def fields(self):
        result = self.readvalues.keys()
        result.remove('__SME_name__')
        return result

    def clock(self):
        self.readvalues, self.writevalues = self.writevalues, self.readvalues
        self.writevalues = dict.fromkeys(self.readvalues)
```

Simple setup



Simple setup

CPUMEM.py — UNREGISTERED

HFT.py | CPUMEM.py | linedetect.py | SME.py

```
1   from SME import Bus, Network, External
2
3   class cpu(External):
4       def setup(self, args):
5           self.bus = args
6           self.state = 4
7           self.bus['rwp'] = 3
8           self.bus['addressbus'] = 0
9
10      def run(self): ...
30
31  class memory(External):
32      def setup(self, args):
33          self.bus = args
34          self.memory = [0] * 1024
35
36      def run(self): ...
42
43  class system(Network):
44      def wire(self, args):
45          self.bus = Bus('PM', ['addressbus', 'databus', 'rwp'])
46          self.memory = memory('Mem', self.bus)
47          self.cpu = cpu('CPU', self.bus)
48
49  test = system('Membus')
50  print test.plot()
51  #test.clock(13)
52
```

.* Aa “” 50 Find Find Prev Find All

Line 36, Column 3 Spaces: 2 Python

Total Physical Source Lines of Code (SLOC) = 44

One motivation ☺

One motivation ☺

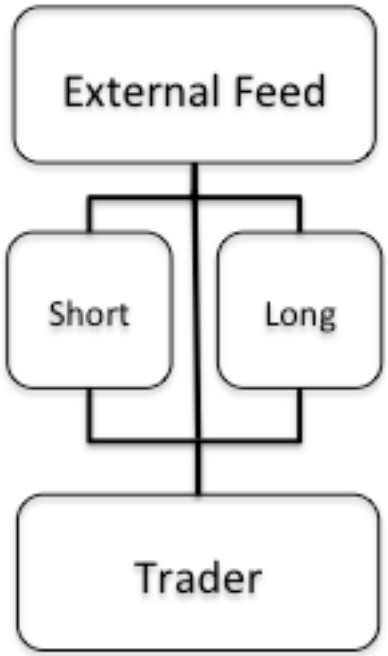
CNN Money Business Markets Tech Luxury stock tickers

Investing Guide

Trading was halted 1,200 times Monday

One flash-crash every 24 sec

Small example: Toy HFT chip



Small example: Toy HFT chip

HFT.py

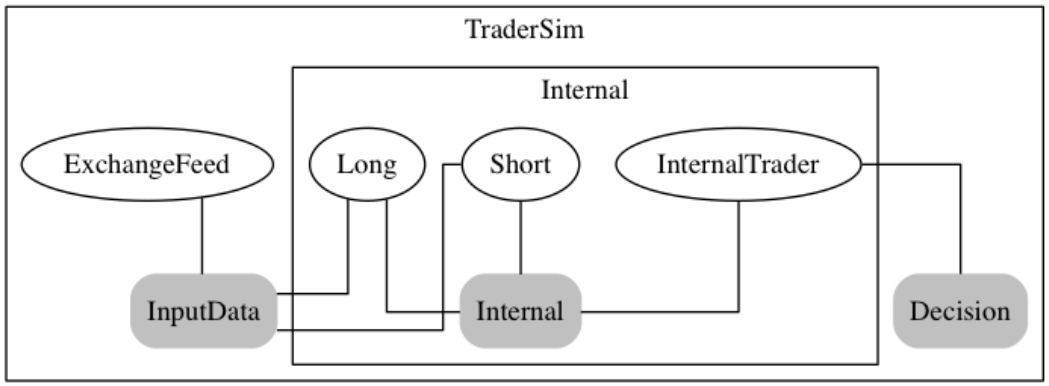
HFT.py | linedetect.py | SME.py

```
from SME import Bus, Network, Function, External
import random

class StockInput(External): #This truely is a simulated stock market
    def setup(self, args):
        self.bus = args
        self.papers = [random.random()*100.0 for _ in xrange(100)]
        self.bus['id'] = 0
        self.bus['value'] = self.papers[0]

    def run(self):
        target = random.randint(0,len(self.papers)-1)
        self.papers[target] += random.random() - 0.5
        self.bus['id'] = target
        self.bus['value'] = self.papers[target]

class avg(Function):
    def setup(self, args):
        self.input, self.output, self.id, self.oname, self.window, self.wsize, self.avg = args
        self.next = 0
        self.output[self.oname] = self.avg

    def run(self):
        if self.input['id'] == self.id:
            self.avg -= self.window[self.next]
            self.window[self.next] = self.input['value'] / self.wsize
            self.avg += self.window[self.next]
        self.output[self.oname] = self.avg

class trader(Function):
    def setup(self, args):
        self.input, self.output = args
        self.state = 0

        self.output['internal'] = 0

    def run(self):
        result = 0 #Assume do nothing
        if self.state == 0 or self.state == 1:
            if self.input['short'] < self.input['long']:
                result = -1 #Sell!
        if self.state == 0 or self.state == -1:
            if self.input['short'] > self.input['long']:
                result = 1 #Buy!

        self.output['internal'] = result

class internal(Network):
    def wire(self, args):
        self.input, self.output, self.id, shortdata, shortlen, shortavg, longdata, longlen, longavg = args
        self.data = Bus('Internal',['short', 'long'])
        self.short = avg('Short', [self.input, self.data, self.id, 'short', shortdata, shortlen, shortavg])
        self.long = avg('Long', [self.input, self.data, self.id, 'long', longdata, longlen, longavg])
        self.trader = trader('InternalTrader', [self.data, self.output])

class system(Network):
    def wire(self, args):
        self.id, shortdata, shortlen, shortavg, longdata, longlen, longavg = args

        self.input = Bus('InputData', ['id', 'value'])
        self.output = Bus('Decision', ['internal','index'])

        self.internal = internal('Internal',[self.input, self.output, self.id, shortdata, shortlen, shortavg, longdata, longlen, longavg])
        self.sim = StockInput('ExchangeFeed', self.input)
```

Line 62, Column 1 | Tab Size: 4 | Python

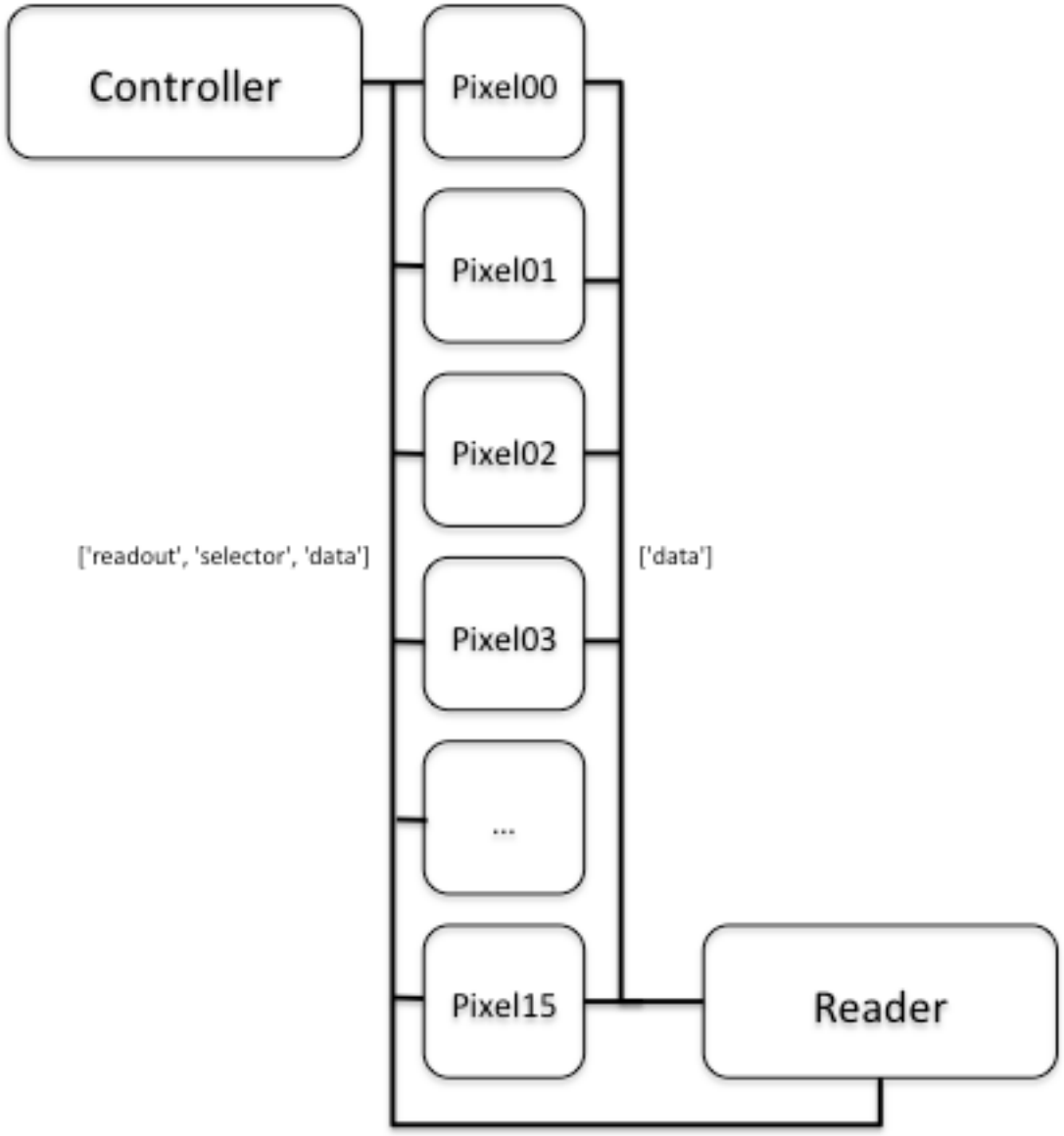
Total Physical Source Lines of Code (SLOC) = 64

Small example: Toy HFT chip




```
Decision.index, Decision.internal, InputData.id, InputData.value, Internal.short, Internal.long,
-1, 0, 0, 50.0317660611, 50.1584571119, 50.0199398988,
-1, 1, 19, 50.1576112478, 50.1584571119, 50.0199398988,
0, 0, 14, 49.5694055881, 50.1584571119, 50.0199398988,
19, 0, 13, 49.9659242821, 50.1584571119, 50.0199398988,
14, 0, 48, 49.4135076139, 50.1584571119, 50.0199398988,
13, 0, 2, 49.7612525624, 50.1584571119, 50.0199398988,
48, 0, 79, 50.6306782514, 50.1584571119, 50.0199398988,
2, 0, 91, 49.9335783041, 50.1584571119, 50.0199398988,
79, 0, 58, 49.5852791995, 50.1584571119, 50.0199398988,
91, 0, 17, 49.5831442915, 50.1584571119, 50.0199398988,
```

An example: Xray camera driver

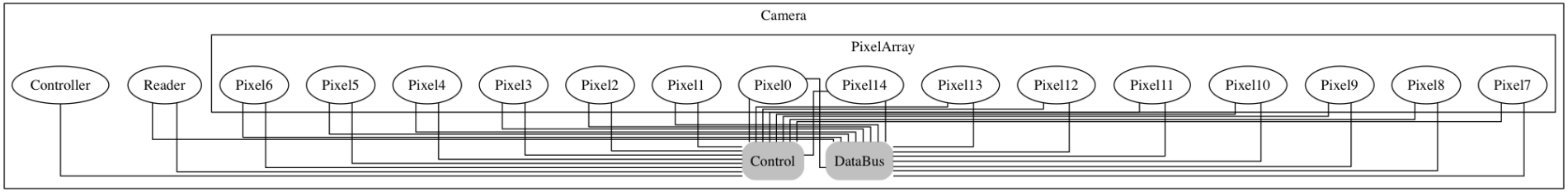


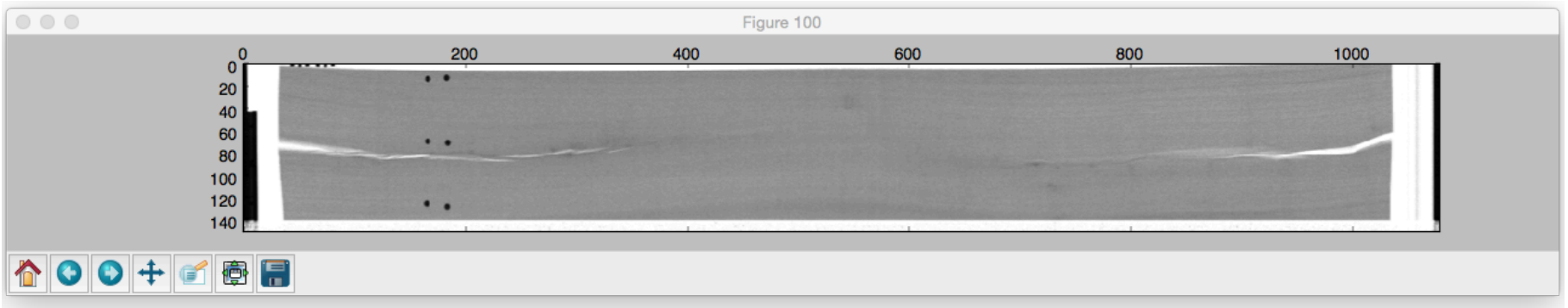
An example: Xray camera driver

```
class System(Network):
    def wire(self, args):
        self.pixels, self.buffer, self.rate, self.simdata = args
        self.controlbus = Bus('Control', ['readout', 'selector', 'data'])
        self.databus = Bus('DataBus', 'data')

        self.controller = Controller('Controller', [self.controlbus, self.rate, self.pixels])
        self.reader = Reader('Reader', [self.databus, self.controlbus, self.buffer])
        self.elements = Array('PixelArray', [self.controlbus, self.databus, self.pixels, self.simdata])
```

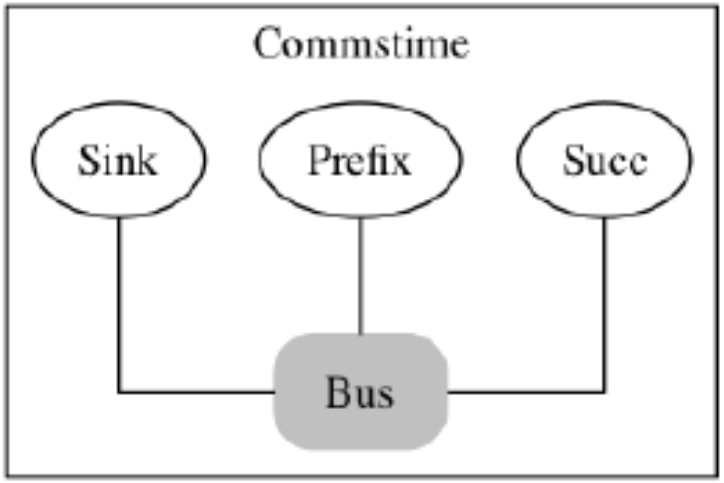
Total Physical Source Lines of Code (SLOC) = 86

An example: Xray camera driver





"Commstime"




```
class prefix(Function):
  def setup(self, args):
    self.bus, self.input, self.output, self.value = args
    self.bus[self.output] = self.value

  def run(self):
     self.bus[self.output] = self.bus[self.input]

class succ(Function):
  def setup(self, args):
    self.bus, self.input, self.output = args
    self.bus[self.output] = 0

  def run(self):
     self.bus[self.output] = self.bus[self.input] + 1


class sink(Function):
  def setup(self, args):
    self.bus, self.input = args

  def run(self):
    pass

class network(Network):
 def wire(self, args):
   self.bus = Bus('Bus',['a','c'])

   self.prefix = prefix('Prefix', [self.bus, 'c', 'a', 0])
   self.succ = succ('Succ',[self.bus, 'a', 'c'])
   self.sink = sink('Sink', [self.bus, 'a'])
```

Total Physical Source Lines of Code (SLOC) = 46

Performance

SME	SMEv2	SMEv2 no log
1.616	1.967	1.704

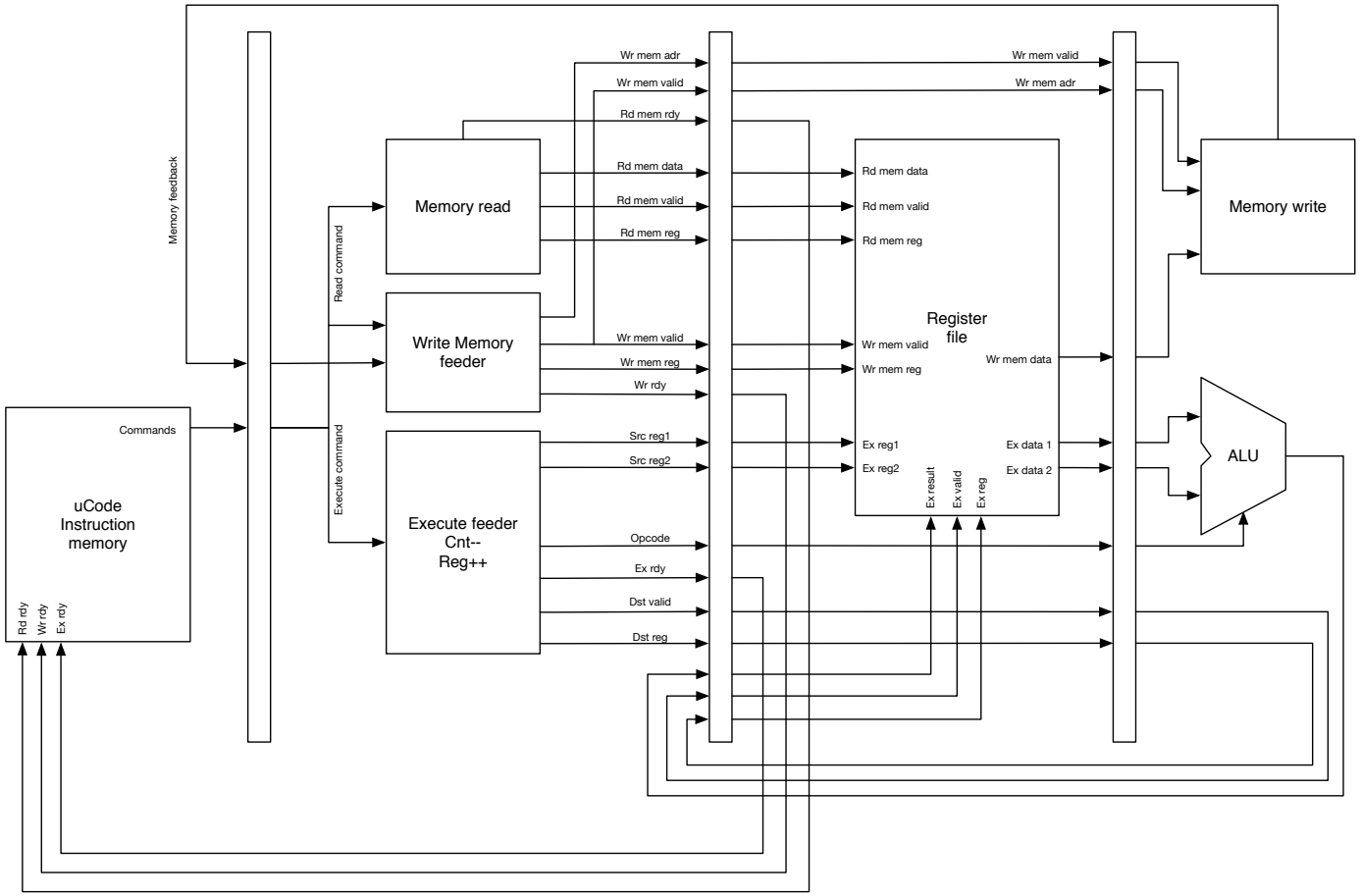
What's not in the paper

We actually have SMEv2 implementations in

- Python
- C++
- C#

They are not quite identical in API – a little cleanup required before we release the code

A larger example



C++ sloccount

```
Creating filelist for src
Categorizing files.
Finding a working MD5 command....
Found a working MD5 command.
Computing results.


SLOC    Directory       SLOC-by-Language (Sorted)
666     src             cpp=666


Totals grouped by language (dominant language first):
cpp:            666 (100.00%)
```

C++ Performance

1.9 GHz notebook – using one core

Simulated target frequency 1GHz

Rate 1:980

C# stats

```
SLOC   Directory       SLOC-by-Language (Sorted)
820     top_dir         cs=820
11      Properties      cs=11
1       obj             cs=1
0       bin             (none)


Totals grouped by language (dominant language first):
cs:              832 (100.00%)
```

C# Performance

2.8 GHz notebook – using one core

Simulated target frequency 1GHz

Rate 1:1.400.000

Note that the simulated models are not at all identical
the C# model is much more detailed

Summary

Much simpler wireing now

Type checking

Graphic rendering

Logs for VHDL simulation input

Clock multiplier support