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 inconvenient 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 + '\n' + 'graph = graphname + '+str(value)+', shape=Mrecord, style=filled, color=grey];\n' + 'return graph

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

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

```python
from SME import Bus, Network, External

class cpu(External):
    def setup(self, args):
        self.bus = args
        self.state = 4
        self.bus['rwp'] = 3
        self.bus['addressbus'] = 0

    def run(self):

class memory(External):
    def setup(self, args):
        self.bus = args
        self.memory = [0] * 1024

    def run(self):

class system(Network):
    def wire(self, args):
        self.bus = Bus('PM', ['addressbus', 'databus', 'rwp'])
        self.memory = memory('Mem', self.bus)
        self.cpu = cpu('CPU', self.bus)

test = system('Membus')
print(test.plot())
#test.clock(13)
```

Total Physical Source Lines of Code (SLOC) = 44
Trading was halted 1,200 times Monday
One motivation 😊

Trading was halted 1,200 times Monday

One flash-crash every 24 sec
Small example: Toy HFT chip
Small example: Toy HFT chip

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"

```python
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', [self.bus, 'a', 'c'])

    def __init__(self):
        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

<table>
<thead>
<tr>
<th></th>
<th>SME</th>
<th>SMEv2</th>
<th>SMEv2 no log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>1.616</td>
<td>1.967</td>
<td>1.704</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>SLOC</th>
<th>Directory</th>
<th>SLOC-by-Language (Sorted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>666</td>
<td>src</td>
<td>cpp=666</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>SLOC</th>
<th>Directory</th>
<th>SLOC-by-Language (Sorted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>820</td>
<td>top_dir</td>
<td>cs=820</td>
</tr>
<tr>
<td>11</td>
<td>Properties</td>
<td>cs=11</td>
</tr>
<tr>
<td>1</td>
<td>obj</td>
<td>cs=1</td>
</tr>
<tr>
<td>0</td>
<td>bin</td>
<td>(none)</td>
</tr>
</tbody>
</table>

**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 wiring now

Type checking

Graphic rendering

Logs for VHDL simulation input

Clock multiplier support