Soumak
How rich descriptions enable early detection of hookup issues
Peter Birch and Thomas Brown, Graphcore
Overview

• Scale and complexity at Graphcore
• Constructing subsystems and chips
• Existing solutions
• Goals in developing a new solution
• Introduction to Soumak
• Shift-left of issue detection
Scale and Complexity

• Reticle-scale die with 59.4 billion transistors
  • 1,472 instances of the Tile processor
  • Numerous SERDES interfaces for Ethernet & PCIe

• Subsystems are complex and deeply hierarchical
  • Hundreds of components
  • Thousands of connections
  • Many distinct signal types
Constructing Complex Subsystems

• Infeasible in SV/VHDL
• Connectivity is horrendous
  • Thousands of connections
  • Many similarly named and sized signals
  • Verbose syntax
• Chances of an error are high
• Lint can only help so much
• Exhaustive simulation and formal proof infeasible at these scales
Abstractions

• Deeper hierarchy
  • Related modules can be grouped together to contain wiring
  • Can lead to repetitive hierarchical connections

• Use SV/VHDL interfaces
  • Grouped signals reduce complexity, lower chance of an error
  • Commercial tool support is highly variable

• Describe connectivity at a higher level
  • Use another language to describe (and automate) connectivity
Existing Solutions

• Accelera IP-XACT
  • Syntax focused on machine readability, not hand editing
  • EDA tooling required to generate RTL
  • Tool APIs and reporting limit custom flows

• Alternative HDLs like Chisel (Scala) & Amaranth (Python)
  • Partial adoption is difficult
  • Shims around SV/VHDL can be painful
Requirements

• Concise syntax for describing connectivity
• Tight integration with existing SystemVerilog design
• Support for:
  • Nested interfaces
  • Constants, typedefs, and data structures
  • Topologies such as rings, chains, and meshes
• Early-as-possible sanity checks
• Support for backends such as code generation
Workflow

- DBs
- Machine Readable Specifications

Soumak .py

Early (Declaration) Checks

Elaboration

Checkers
- Unconnected Ports
- Simple CDC
- Unique Signals

Backends
- Code Generation
- Reporting
- Design Explorer

Soumak

SV

HTML
Defining a Leaf Node

```python
import soumak
from soumak import In, Out
from soumak.signal import Clock, Reset

@soumak.block()
class Hexcpu:
    clk : In(Clock)
    rst : In(Reset)
    fault : Out(width=1, desc="Internal error occurred")
```

hexcpu.py
Defining a Leaf Node

```python
import soumak
from soumak import In, Out
from soumak.signal import Clock, Reset

@soumak.block()
class Hexcpu:
    clk : In(Clock)
    rst : In(Reset)
    fault : Out(width=1, desc="Internal error occurred")
```

- Decorators check hardware definition on declaration
- Built-in primitive signal types
- Decorator reads and checks the type annotations
Interfaces

```python
import soumak
from soumak import Request, Response
from soumak.signal import Clock, Reset

@soumak.interface()
class JTAG:
    """ JTAG bus definition ""
    tck : Request(Clock)
    trst : Request(Reset)
    tms : Request(desc="Mode select")
    tdi : Request(desc="Test data in")
    tdo : Response(desc="Test data out")
```

jtag.py
Interfaces

```
import soumak
from soumak import Request, Response
from soumak.signal import Clock, Reset

@soumak.interface()
class JTAG:
    """ JTAG bus definition """
    tck : Request(Clock)
    trst : Request(Reset)
    tms : Request(desc="Mode select")
    tdi : Request(desc="Test data in")
    tdo : Response(desc="Test data out")
```
Types & Constants

Explicit values and arithmetic is fully supported.

```
import soumak
from soumak import Constant, Typedef

@soumak.package()
class HexPackage:
    # === Constants ===
    ADDR_W : Constant(desc="Width of address bus") = 32
    DATA_W : Constant(desc="Width of the data bus") = 32
    BYTE_W : Constant(desc="Data bus width in bytes") = (DATA_W / 8)
    # === Types ===
    data : Typedef(width=DATA_W, desc="Type for carrying data")
```
Enumerations

```python
import soumak
from soumak import Constant, Enum

@soumak.enum(package=HexPackage)
class DMATransfer:
    NOP : Constant(desc="Perform no transfer")
    WRITE : Constant(desc="Write from FIFO to memory")
    READ : Constant(desc="Read from memory into FIFO")

@soumak.enum(package=HexPackage, mode=Enum.ONE_HOT)
class DMAState:
    IDLE : Constant(desc="Waiting for new DMA request")
    BUSY : Constant(desc="Busy processing memory operation")
    FAULT : Constant(desc="Error occurred while moving data")
```

hex_package.py

Implicit or explicit value assignments

Supports indexing, one-hot, and Gray coding
References can be made to enums, structs, and unions.

```python
@soumak.struct(package=HexPackage, width=32)
class DMAWriteRequest:
    mode : Instance(DMATransfer, desc="Type of transfer")
    address : Scalar(width=HexPackage.ADDR_W, desc="Where to write data")
    length : Scalar(width=16, desc="How many bytes to write")
    exclusive : Scalar(width=1, desc="Lock memory during transfer")

@soumak.struct(package=HexPackage, width=32)
class DMAReadRequest:
    mode : Instance(DMATransfer, desc="Type of transfer")
    address : Scalar(width=HexPackage.ADDR_W, desc="Where to read data from")
    length : Scalar(width=16, desc="How many bytes to read")

@soumak.union(package=HexPackage)
class DMARequest:
    write : Instance(DMAWriteRequest)
    read : Instance(DMAReadRequest)
```

hex_package.py
Interfacing with SystemVerilog

```verilog
package hex_package;

localparam ADDR_W = 32; // Width of address bus
localparam DATA_W = 32; // Width of data bus
localparam BYTE_W = 4;  // Data bus width in bytes

typedef logic [31:0] data_t; // Type for carrying data

// ...

typedef enum logic [2:0] {
    IDLE = 3'b001 // Waiting for new DMA request
    , BUSY = 3'b010 // Busy processing memory operation
    , FAULT = 3'b100 // Error occurred while moving data
} dma_state_t;

// ...
```
Subsystem Assembly

```python
from .hex_package import HexPackage
from .jtag import JTAG

@soumak.block()
class HexCore:
    clk : In(Clock)
    rst : In(Reset)
    debug : In(JTAG)
    dma_req : Out(HexPackage.DMAResult)
    dma_state : In(HexPackage.DMAState)
    # ...other signals...

@soumak.block()
class HexDMA:
    clk : In(Clock)
    rst : In(Reset)
    request : In(HexPackage.DMAResult)
    state : Out(HexPackage.DMAState)
    # ...other signals...
```

**hex_core.py**

**hex_dma.py**
Subsystem Assembly

```python
from .hex_core import HexCore
from .hex_dma import HexDMA
from .jtag import JTAG

@soymak.block()
class HexSubsystem:
    # === Ports ===
    clk : In(Clock)
    rst : In(Reset)
    debug : In(JTAG)
    # === Children ===
    core : Instance(HexCore)
    dma : Instance(HexDMA)
    # === Connectivity ===
    def connect(self):
        self.fanout(self.clk, self.all_children.clk)
        self.fanout(self.rst, self.all_children.rst)
        self.link(self.debug, self.core.debug)
        self.link(self.core.dma_req, self.dma.request)
        self.link(self.dma_state, self.core.dma_state)
```
Subsystem Assembly

from .hex_core import HexCore
from .hex_dma import HexDMA
from .jtag import JTAG

@soymak.block()
class HexSubsystem:
    # === Ports ===
    clk : In(Clock)
    rst : In(Reset)
    debug : In(JTAG)
    # === Children ===
    core : Instance(HexCore)
    dma : Instance(HexDMA)
    # === Connectivity ===
    def connect(self):
        self.fanout(self.clk, self.all_children.clk)
        self.fanout(self.rst, self.all_children.rst)
        self.link(self.debug, self.core.debug)
        self.link(self.core.dma_req, self.dma.request)
        self.link(self.dma.state, self.core.dma_state)
Subsystem Assembly

```python
from .hex_core import HexCore
from .hex_dma import HexDMA
from .jtag import JTAG

@soumak.block()
class HexSubsystem:
    # === Ports ===
    clk : In(Clock)
    rst : In(Reset)
    debug : In(JTAG)
    # === Children ===
    core : Instance(HexCore)
    dma : Instance(HexDMA)
    # === Connectivity ===
    def connect(self):
        self.fanout(self.clk, self.all_children.clk)
        self.fanout(self.rst, self.all_children.rst)
        self.link(self.debug, self.core.debug)
        self.link(self.core.dma_req, self.dma.request)
        self.link(self.dma.state, self.core.dma_state)
```

`all_children` is expanded during elaboration, allowing multiple connections to be formed with a single statement.
Topologies

- Rings, chains, and meshes topologies can be constructed using special ‘traits’
- Multiple complex connections can be formed with just a single statement
- Design tracks which connection patterns have been added

```python
from soumak import Reciprocal
from soumak.topology import Ring

@soumak.block()
class Node:
    access : Reciprocal(AccessBus)

@soumak.block(traits=[Ring])
class Network:
    nodes : 8 * Instance(Node)
    def connect(self):
        self.ring[sources=self.nodes.all.access.outbound,
                   targets=self.nodes.all.access.inbound]
```

`self.nodes.all.inbound` expands in elaboration to create a list of all inbound access ports
Benefits

• Fewer connection statements
  • Concise and easy to audit code
  • Less chance of a mistake

• Strict type checks
  • Impossible to connect incompatible signals without an explicit cast
  • Fewer lines of code to audit
  • Works for single wires and complex buses
Declaration Checks

```python
/a/work/peterb/sili/workspace_one/colossus-silicon/lib/soumak/examples/blocks/design.py:33 in <module>
  30  # ---------------------------------------------------------------------------
  31  
  32 @soumak.block()
> 33 class SingleController:
  34     """Single controller block """
  35     # Boundary I/O
  36     clk : Request(Clock, desc="Clock signal")
```

`SourakFieldError: Illegal field 'clk' of 'SingleController' - uses type 'Request', only allowed types: In, InOut, Out, Reciprocal, InBundle, InOutBundle, OutBundle, Parameter, ReciprocalBundle, Instance, Array`
Checks on Declarations

```python
@soumak.enum(package=DesignPkg, width=1)
class MessageType:
    """ Different types of messages sent over the bus """
    NOP = Constant("Perform no operation")
    WRITE = Constant("Perform a write operation")
    READ = Constant("Perform a read operation")
```

```
@/a/work/peterb/sili/workspace_one/colossus-silicon/lib/soumak/examples/packages/design.py:28 in <module>
  25 |     Token : Typedef( 4, "Access token" )
  26 | @soumak.enum(package=DesignPkg, width=1)
  27 | )
  28 | @soumak.enum(package=DesignPkg, width=1)
     | """ Different types of messages sent over the bus """
  29 |     NOP = Constant("Perform no operation")
  30 |     WRITE = Constant("Perform a write operation")

@/a/work/peterb/sili/workspace_one/colossus-silicon/lib/soumak/soumak/__init__.py:319 in do_inner
@/a/work/peterb/sili/workspace_one/colossus-silicon/lib/soumak/soumak/meta/enum.py:63 in _sk_construct

SoumakFieldError: Entry 'READ' of MessageType takes value 2 which exceeds the bit width of 1
```
Checks During Elaboration

```
   55  # Declare connectivity
   56  def connect(self):
   57      
   58  #       self.fanout(self.clk, self.ctrl.any().rst)
   59  #                        ^^^^^^^^^^^^^^^^^^^^^^^^^
   60  # Sanity checks
   61  #

SoumakConnectionPatternError: Cannot connect points
Source - 'IN' port 'nc.clk' (Unknown)
Target - 'IN' port 'nc.ctrl_0.rst' (Unknown)
As the source port is of type 'Clock' (with width of 1 bits), while target port is of type 'Reset' (with width of 1 bits).
```
Checks on the Assembled Design

[12:12:01] ERROR checkerclocking: Connection in 'Top' made between 'Top.blk_a.egress' and 'Top.blk_b.ingress' spans between two different clock domains 'Top.blk_a.clk' and 'Top.blk_b.clk'
Precise Sign-offs

```python
@soumak.block()
class Top:
    # ...
    def signoff(self):
        yield Signoff.okay("clocking",
                          extra=(self.blk_a.egress,
                                 self.blk_b.ingress)))
```
Connection Tracing
Connection Tracing
Connection Tracing
Connection Tracing
Summary

• Assembling reticle sized ASICs is a difficult task
• Soumak abstracts the assembly of subsystems
  • Shared constants and types softens boundary between tools
  • Complex interface descriptions reduces wiring verbosity
  • Python can be leveraged to automate connectivity
• Rich descriptions enable earlier checks
  • Strict type checking helps to reduce mistakes
  • Analysis flows can crawl through elaborated designs
  • Checkers can flag gross issues early in the design process