2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
DECEMBER 6 - 7, 2022

Soumak

How rich descriptions enable early detection of hookup issues

Peter Birch and Thomas Brown, Graphcore

GRAPHCORE

accellera SYSTEMS INITIATIVE

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

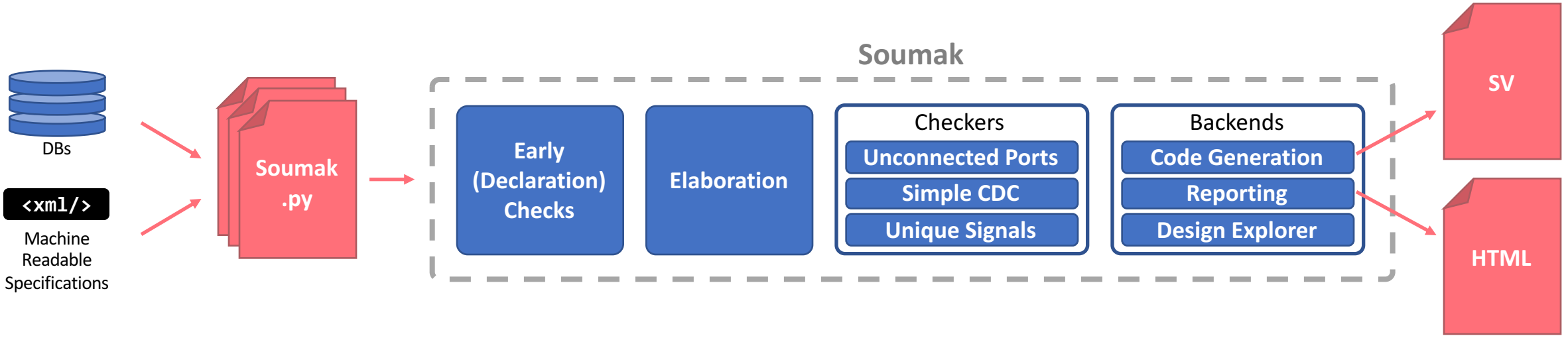
Existing Solutions

- 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

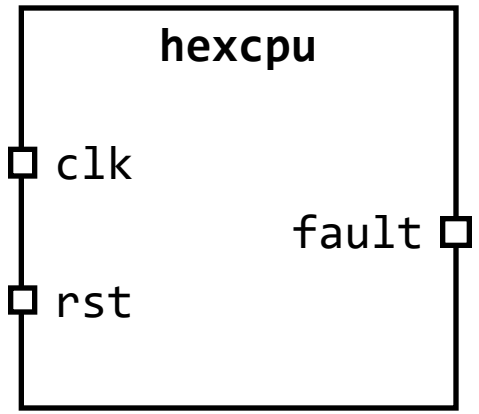
Workflow


DBs
<xml/>
Machine Readable Specifications
Soumak .py
Soumak
Early (Declaration) Checks
Elaboration
Checkers
Unconnected Ports
Simple CDC
Unique Signals
Backends
Code Generation
Reporting
Design Explorer
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

Decorators check hardware definition on declaration

```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")
```

Built-in primitive signal types

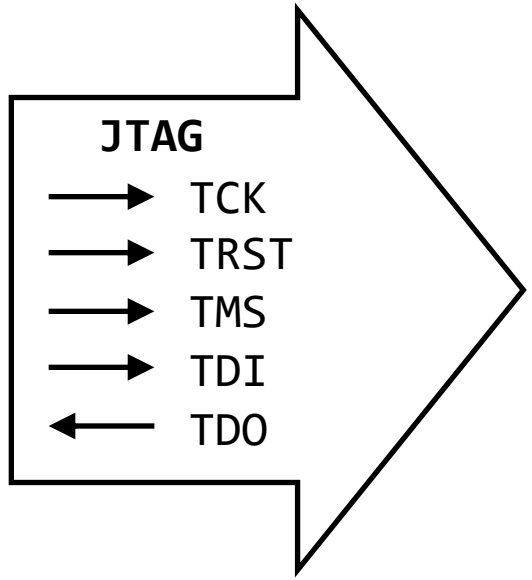
Decorator reads and checks the type annotations

hexcpu.py

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

```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")
```

Signals can travel with or against the interface

Comments are stored with each declaration

Base signal types & other interfaces can be referenced

jtag.py

Types & Constants

Explicit values and arithmetic is fully supported

```python
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")
```

Simple data types can be declared

hex_package.py

Enumerations

Implicit or explicit value assignments

```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")
```

Supports indexing, one-hot, and Gray coding

hex_package.py

Structs & Unions

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

```
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.DMARequest)
    dma_state : In(HexPackage.DMAState)
    # ...other signals...
```

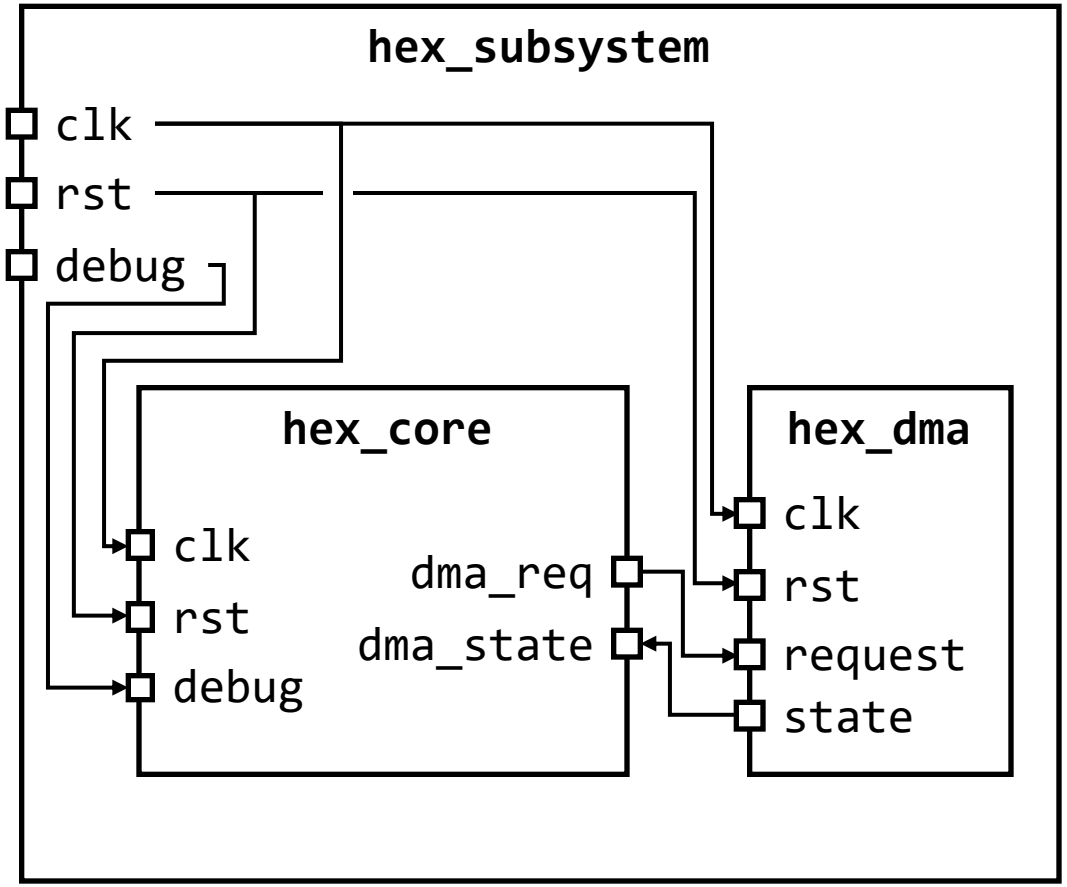
hex_core.py

```python
from .hex_package import HexPackage

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

hex_dma.py

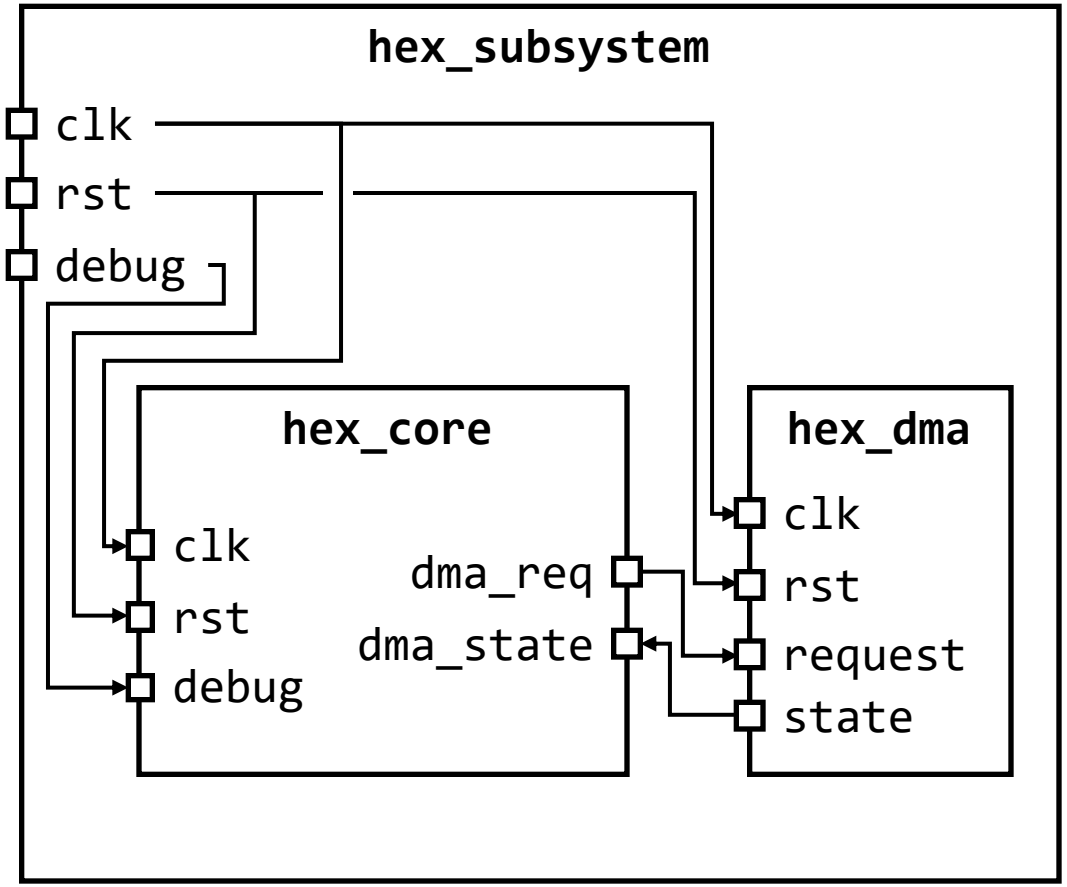
Subsystem Assembly

```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)
```



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)
```



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

```
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
```

- 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

`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

```
/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")

/a/work/peterb/sili/workspace_one/colossus-silicon/lib/soumak/soumak/__init__.py:213 in do_inner

SoumakFieldError: 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

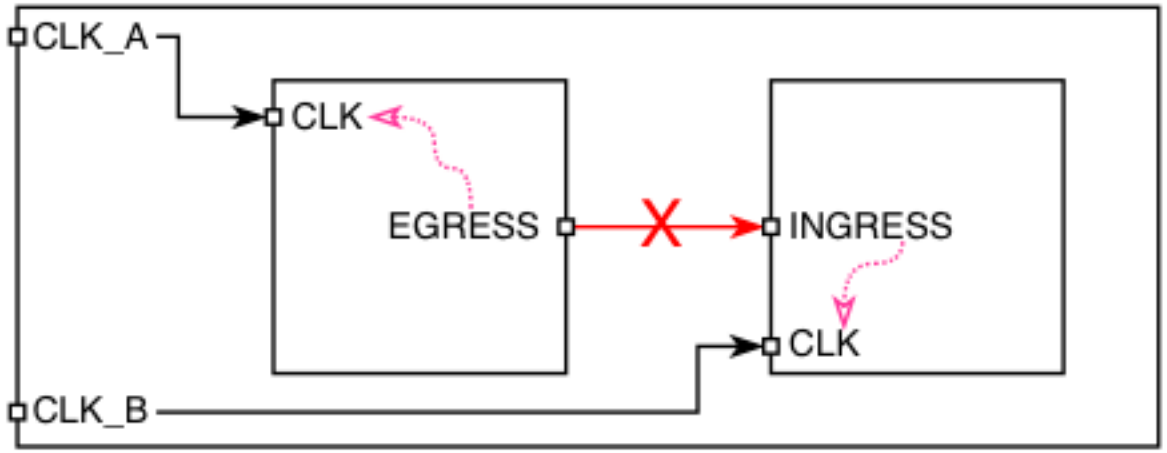
```python
@soumak.enum(package=DesignPkg, width=1)
class MessageType:
    """ Different types of messages sent over the bus """
    NOP   : Constant("Perform no operation")
    WRITE : Constant("Perform a write operation")
    READ  : Constant("Perform a read operation")
```

```
/a/work/peterb/sili/workspace_one/colossus-silicon/lib/soumak/examples/packages/design.py:28 in <module>

   25 |    Token            : Typedef(  4, "Access token"       )
   26
   27 @soumak.enum(package=DesignPkg, width=1)
❱  28 class MessageType:
   29 |    """ Different types of messages sent over the bus """
   30 |    NOP   : Constant("Perform no operation")
   31 |    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

```
/a/work/peterb/sili/workspace_one/colossus-silicon/lib/soumak/examples/blocks/design.py:58 in connect

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

/a/work/peterb/sili/workspace_one/colossus-silicon/lib/soumak/soumak/meta/block.py:514 in fanout

/a/work/peterb/sili/workspace_one/colossus-silicon/lib/soumak/soumak/meta/block.py:388 in build_links

/a/work/peterb/sili/workspace_one/colossus-silicon/lib/soumak/soumak/meta/common/connection.py:98 in check

SoumakConnectionPatternError: Cannot connect points
  Source - 'IN'  port 'mc.clk' (Unknown)
  Target - 'IN'  port 'mc.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).
```

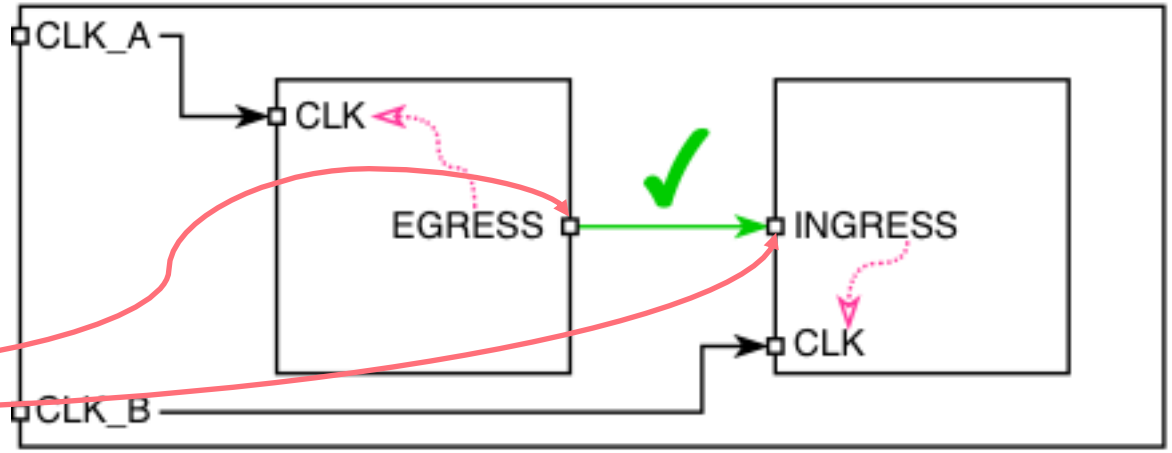
Checks on the Assembled Design



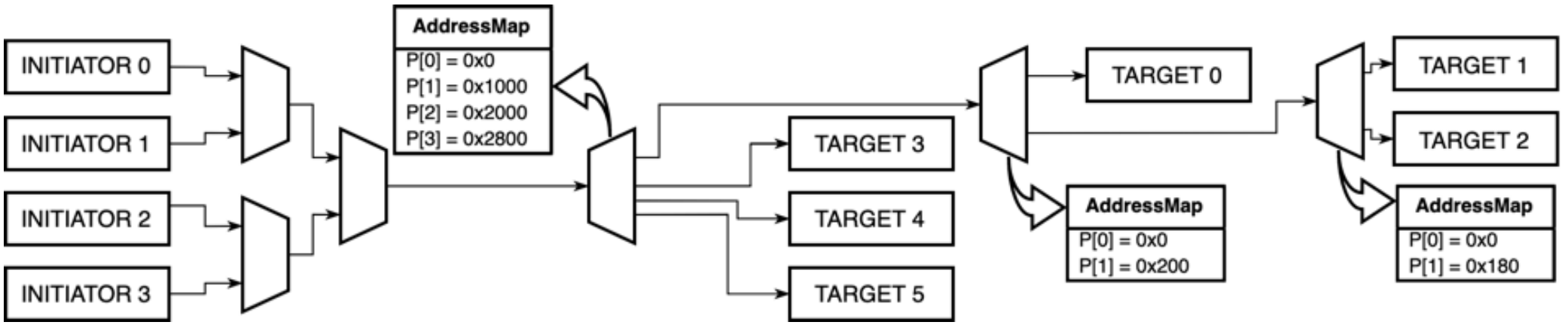

```
[12:12:01] ERROR    checker.clocking: Connection in 'Top' made between 'Top.blk_a.egress' and 'Top.blk_b.ingress' spans between    report.py:144
                    two different clock domains 'Top.blk_a.clk' and 'Top.blk_b.clk'
```

Precise Sign-offs

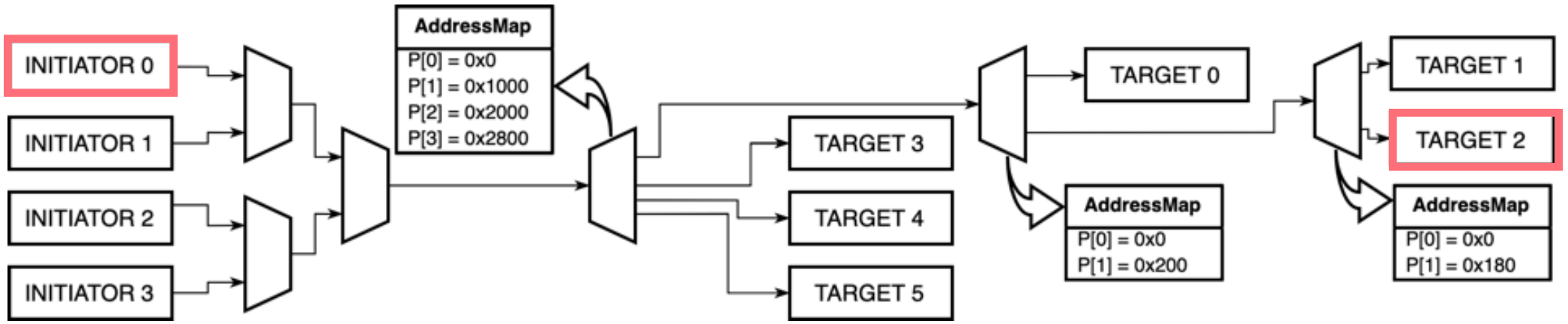
```python
@soumak.block()
class Top:

    # ...

    def signoff(self):
        yield Signoff.okay("clocking",
                           extra=(self.blk_a.egress,
                                  self.blk_b.ingress))
```



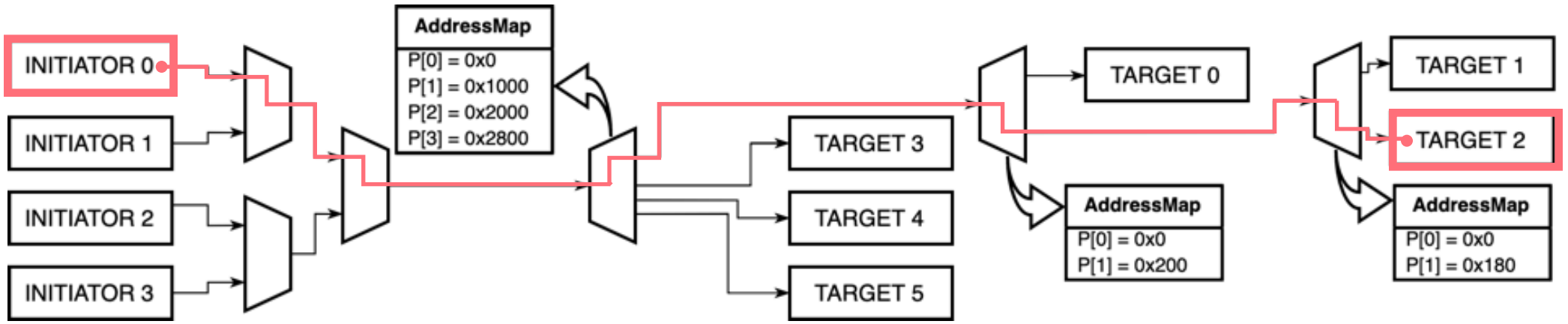
Connection Tracing



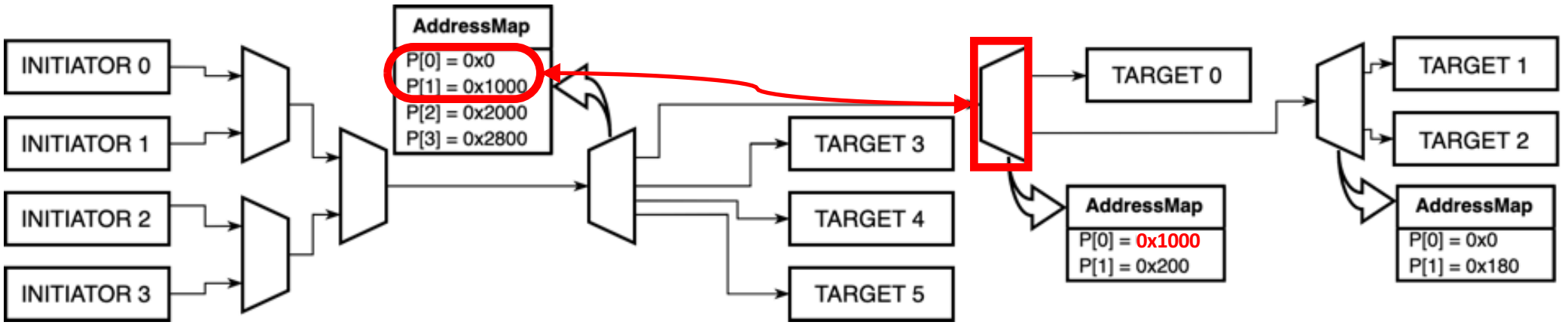
Connection Tracing



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