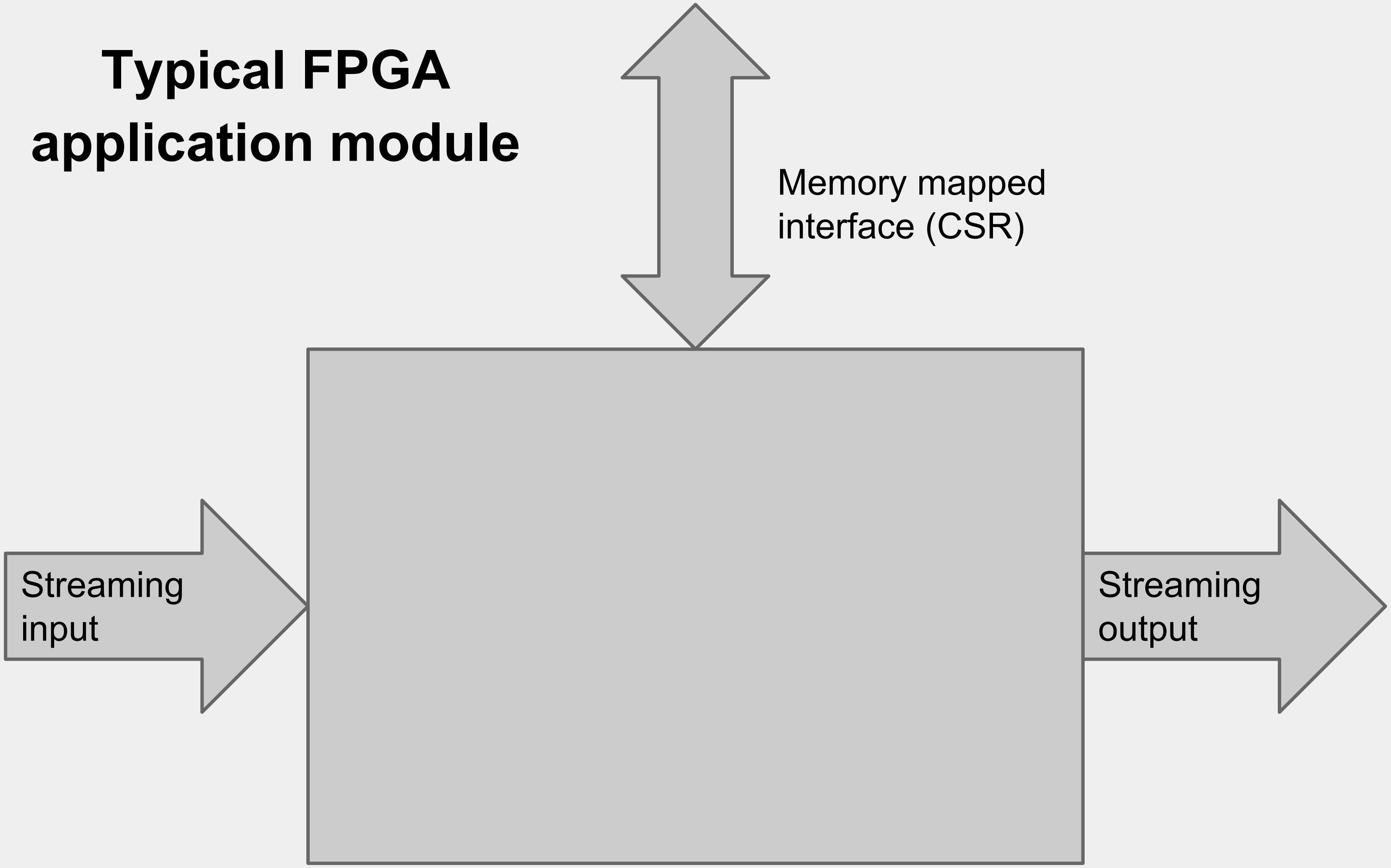
Enhancing Hardware Design Flows with MyHDL

Keerthan Jaic, Melissa Smith
{kjaic,smithmc}@clemson.edu

Typical FPGA application module



Declaring the module in verilog

```
module module_name
  #(
  parameter SYMBOL_WIDTH = 8,
  parameter SYMBOLS_PER_BEAT = 8
  ) (
  input clk,
  input reset_n,

  input [SYMBOLS_PER_BEAT*SYMBOL_WIDTH-1:0] stream_in_data,
  input [$clog2(SYMBOLS_PER_BEAT)-1:0] stream_in_empty,
  input stream_in_valid,
  input stream_in_startofpacket,
  input stream_in_endofpacket,
  output reg stream_in_ready,

  output reg [SYMBOLS_PER_BEAT*SYMBOL_WIDTH-1:0] stream_out_data,
  output reg [$clog2(SYMBOLS_PER_BEAT)-1:0] stream_out_empty,
  output reg stream_out_valid,
  output reg stream_out_startofpacket,
  output reg stream_out_endofpacket,
  input stream_out_ready,

  input [1:0] csr_address,
  output reg [31:0] csr_readdata,
  output reg csr_readdatavalid,
  input csr_read,
  input csr_write,
  output reg csr_waitrequest,
  input [31:0] csr_writedata);
```

Problems

- Too much boilerplate code is required to declare or instantiate modules
- Modules and testbenches end up becoming unnecessarily verbose
- It takes longer for new programmers to grok the design

Goals

- Making hardware description concise
- Reduce development time
- Simplify the process of writing testbenches
- Facilitate sharing of data structures across HW and SW code

What is Python?

- Dynamic high level language
- Clean syntax and extensive standard library
- Enables rapid prototyping and experimentation.

Example:

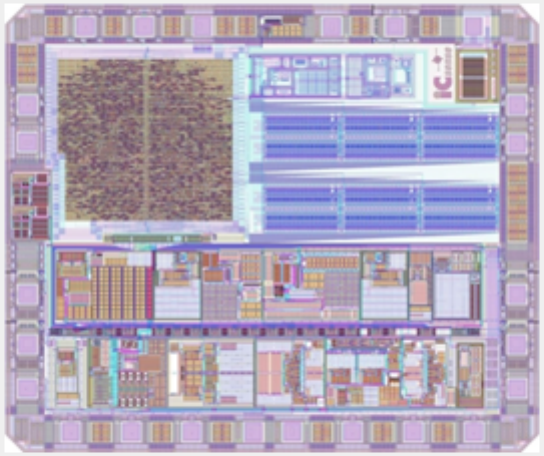
```
a = [1,2,3]
def function():
    for i in a:
        print(a)

>>> function()
1
2
3
```

What is MyHDL?

- MyHDL is a FOSS package for using python as a HDL/HVL.
- The goal of the MyHDL project is to empower hardware designers with the elegance and simplicity of the Python language.
- MyHDL designs can be converted to Verilog or VHDL automatically.
- Built in simulator with VCD tracing support
- Cosimulation with icarus/GHDL/ModelSim/cver

Short History of MyHDL

- Jan Decaluwe
  - Creator of MyHDL
  - Founder & Board Member Sigasi, Easic
- First Release on SourceForge Sep 30, 2003



MyHDL ASIC
www.jandecaluwe.com/hdldesign/digmac.html

Python: Functions vs. Generators

Function:

```
def function():
    for i in range(5):
        return i


>>> function()

0
```

Generator:

```
def generator():
    for i in range(5):
        yield i


>>> g = generator()

>>>g.next()

0

>>>g.next()

1

...
```

Python: Decorators

- Syntactic sugar for modifying functions

Example:

```
@decorator
def func(...):
    …..
```

is equivalent to:

```
def func(...):
    …..
func = decorator(func)
```

MyHDL: Example

```
def mux():
    while True:              #run forever
        yield a, b, sel     # wait till one of them changes
        if sel:
            out.next = a
        else:
            out.next = b
```

MyHDL: Decorators

- Decorators are used to abstract away details from HDL

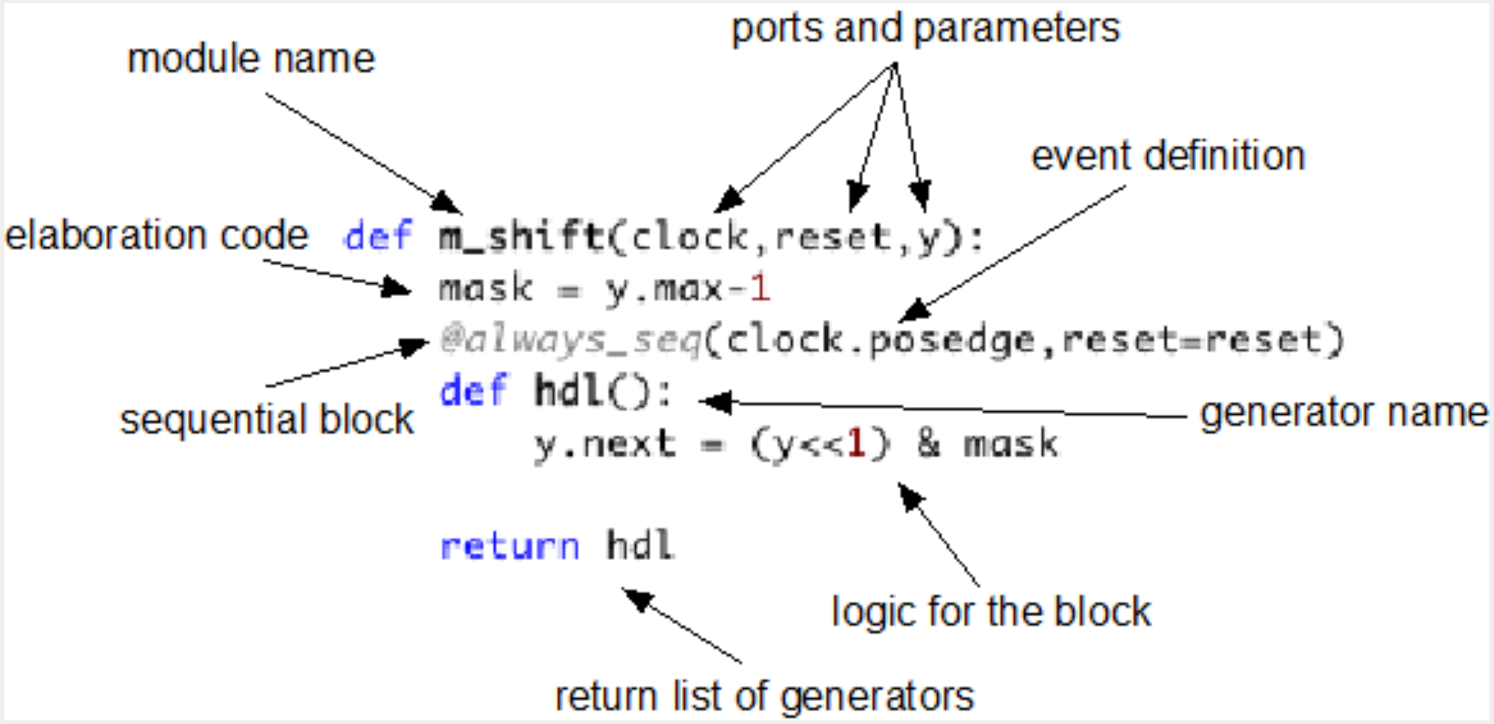
```
@always(a, b, sel)
def mux():
    if sel:
        out.next = a
    else:
        out.next = b
```

```
@always_comb
def mux():
    if sel:
        out.next = a
    else:
        out.next = b
```

MyHDL: Decorators

```
def dff(clk, rst, d, q):
    q = Signal(0)

    @always_seq(clk.posedge, rst)
    def logic():
        q.next = d

    return logic
```

Anatomy of a MyHDL Module

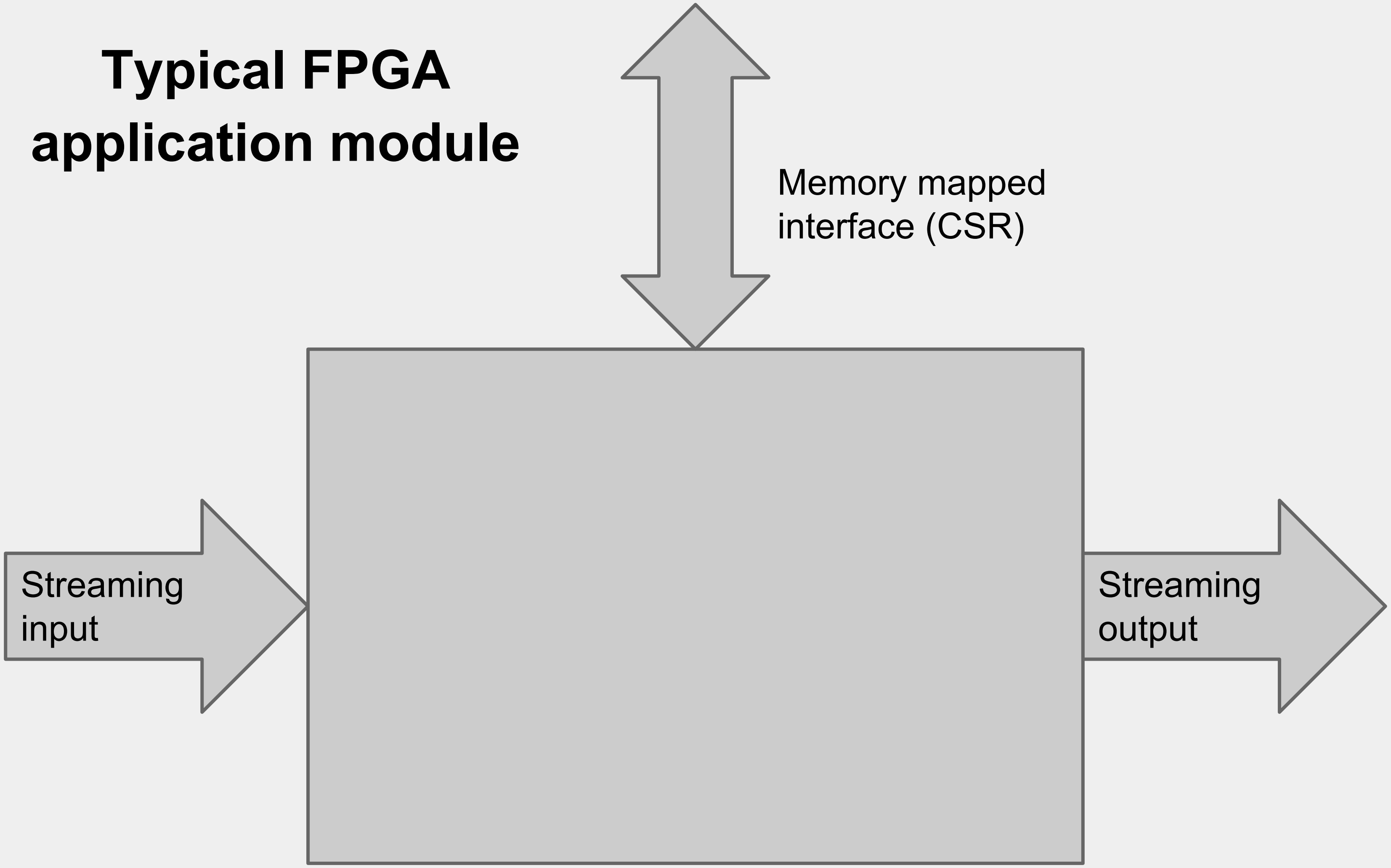


```
def m_shift(clock,reset,y):
    mask = y.max-1
    @always_seq(clock.posedge,reset=reset)
    def hdl():
        y.next = (y<<1) & mask

    return hdl
```

MyHDL v0.8 conversion limitations

- MyHDL conversion works by parsing the AST
- MyHDL could not convert code which included attributes
- This prevents using advanced abstraction techniques such as interfaces
- We added attribute conversion support (0.9)

Typical FPGA application module



Abstracting interfaces using MyHDL

- Python classes can be used to represent interfaces
- Parameterization and error checking can be performed in python

```
class AvalonST(object):
    def __init__(self, symbolwidth=8, symbols_per_beat=1):
        if not 1 <= symbolwidth <= 512:
            raise ValueError("symbolwidth must be between 1-512")

        if not 1 <= symbols_per_beat <= 32
            raise ValueError("symbolsPerBeat must be between 1-
32")

        self.valid = Sig(False)
        self.ready = Sig(False)
        self.data = Sig(symbolwidth*symbols_per_beat)
```

Abstracting interfaces using MyHDL

- Class inheritance can be used to extend interfaces without repeating code

```
class AvalonSTPkts(AvalonST):
    def __init__(self, symbolwidth=8, symbols_per_beat=1):
        super(AvalonSTPkts, self).__init__(symbolwidth, symbolwidth)
        self.startofpacket = Sig(False)
        self.endofpacket = Sig(False)

        if symbols_per_beat > 1:
            self.empty = Sig(max=symbols_per_beat)
```

Abstracting interfaces using MyHDL

```
def module_name(clk, rst, asi, aso, csr):
    ...
    ...
    …
    st = Enum(‘IDLE’, ‘IN_PACKET’,
‘WAITING’)
    @always_seq(clk.posedge, rst):
    def logic():
        if asi.valid and state = st.IDLE:
            state.next = st.IN_PACKET
             aso.data.next = asi.data
    ...
    ...
```

- Modules can be concisely declared using interfaces
- Dot notation can be used to access signals inside interfaces

Abstracting interfaces using MyHDL

```
class AvalonSTPkts(AvalonST):
    ...
    ...
    def tb_send(self, data):
        firstword = True
        for word in data:
            # Wait till sink is ready
            while True:
                yield clk.negedge
                self.valid.next = 0
                if self.ready:
                    break
            self.valid.next = 1
            self.data.next = word
    ...
```

- Bus functional models for testbenches can be included in the interface classes
- Testbench code can use arbitrary python modules such as pcap parsing or image processing

Abstracting interfaces using MyHDL

```
mon = aso.tb_mon(dest=result)
@instance
def stimulus():
    yield rst
    for pkt in data
        expected = golden_model(pkt)
        yield asi.tb_send(pkt)
        assert result == expected
        raise StopSimulation
```

- This simplifies testbenches and makes it simpler to write bus agnostic testbenches

Advanced Examples

- Sharing data structures across SW algorithm code and HW logic
- Single register/bit field specification in HW, SW code

MyHDL Benefits

- Allows algorithm development, HDL and scripting in the same environment.
- Makes python libraries available for testing HDL
- Avoid repetitive tasks. Input/Output port declaration, conversions for signed arithmetic, reset behaviour declaration, etc.
- Code reuse makes it easier to maintain code

Get in touch

- www.myhdl.org
- MyHDL is under active development
- Mailing list is a good place to get support