Imperial College London

LEAP HLS

LEAP from a user's perspective

Felix Winterstein

01 September 2015

Introduction

- State-of-the-art HLS tools can build optimized custom memory systems on-chip
- But on-chip memory capacity often not sufficient

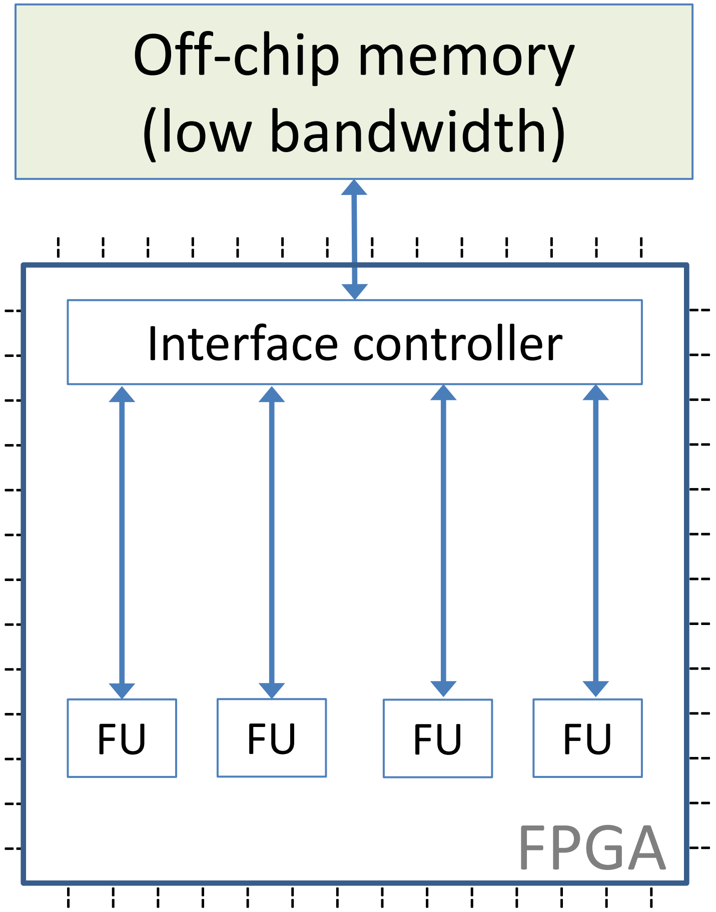
Imperial College London

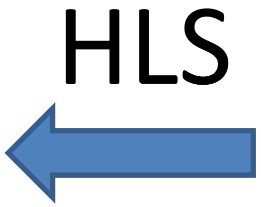
esa

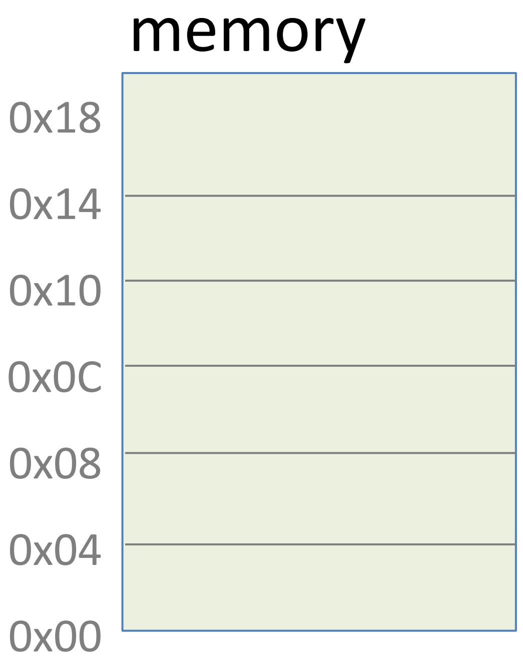
Introduction

- State-of-the-art HLS tools can build optimized custom memory systems on-chip
- But on-chip memory capacity often not sufficient
- Limited support for interfacing external memory
  - Manually connecting an HLS kernel to external memory is complicated
  - LEAP builds a high-performance memory hierarchy underneath a simple API

Recap: MATCHUP [FPGA'15]

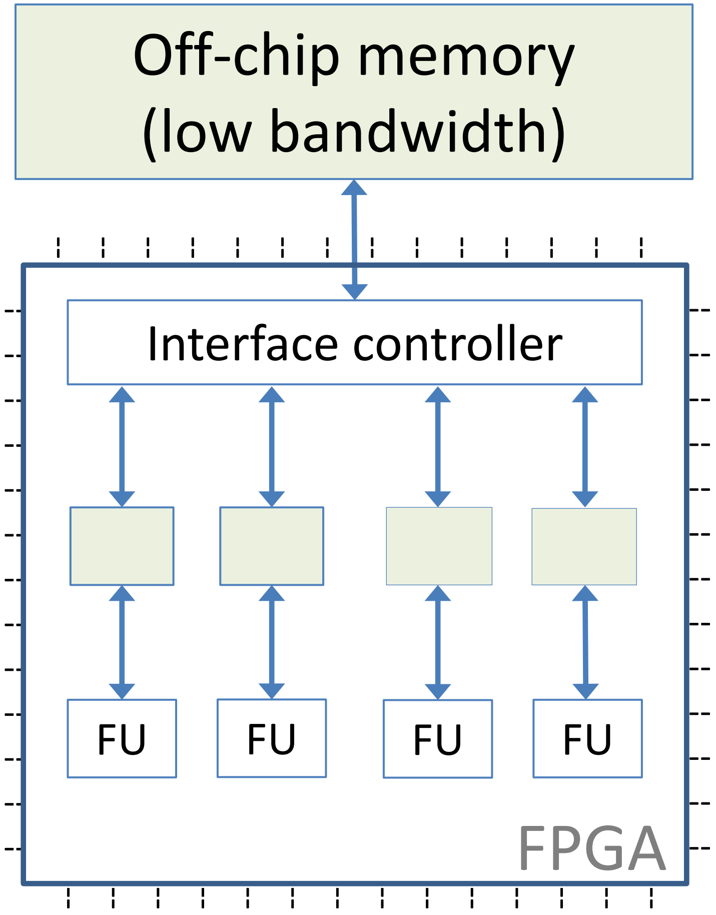


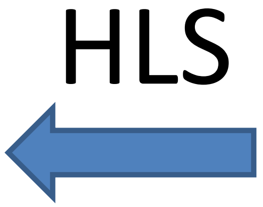


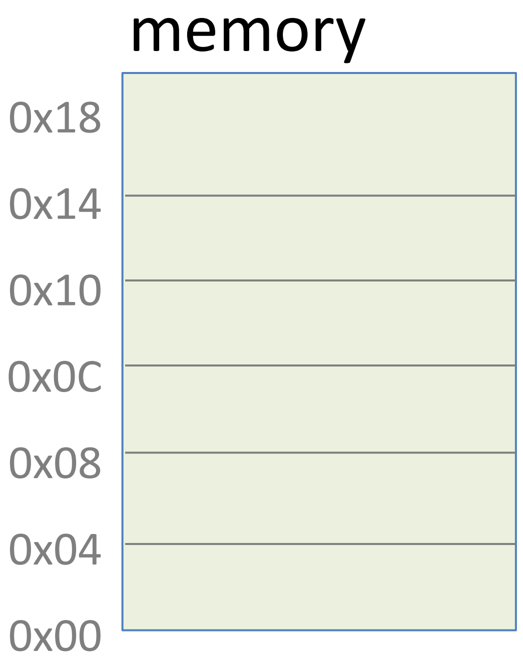


```
s = **new** stackRecord;
s->u = root;
s->n = 0;
while s!=0 do
    t = s;
    u = t->u;
    s = t->n;
```

Recap: MATCHUP [FPGA'15]

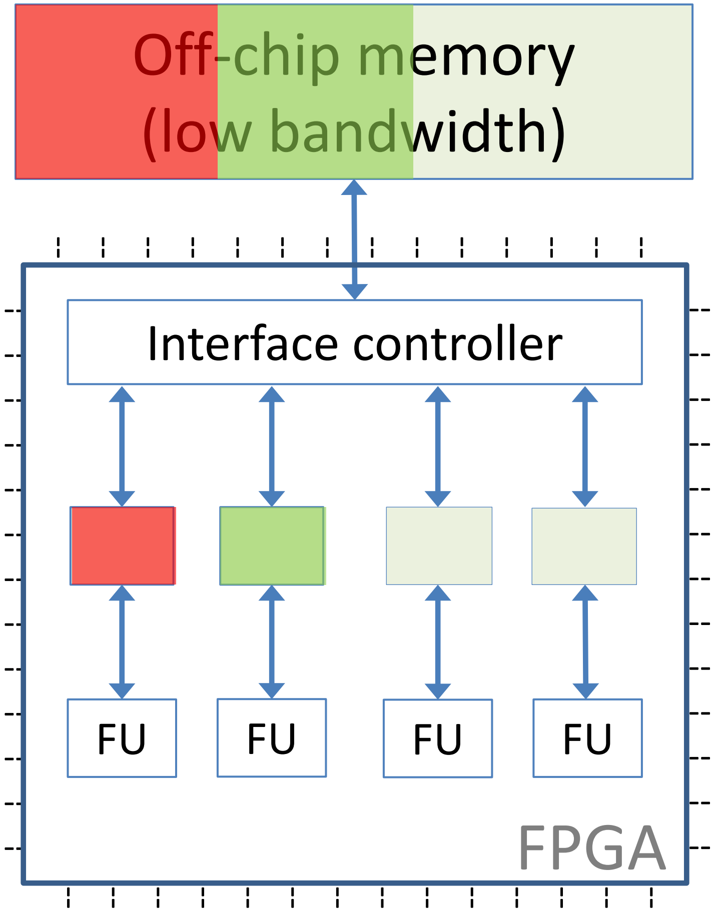


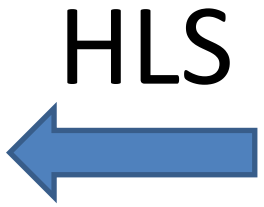


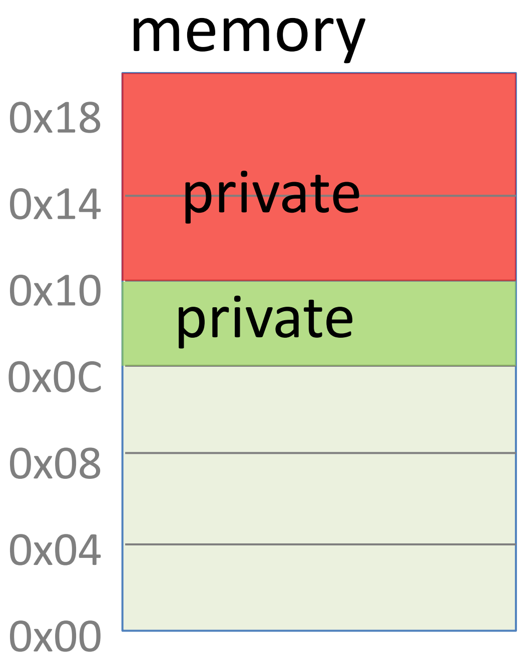


```
s = **new** stackRecord;
s->u = root;
s->n = 0;
**while** s!=0 **do**
    t = s;
    u = t->u;
    s = t->n;
```

Recap: MATCHUP [FPGA'15]

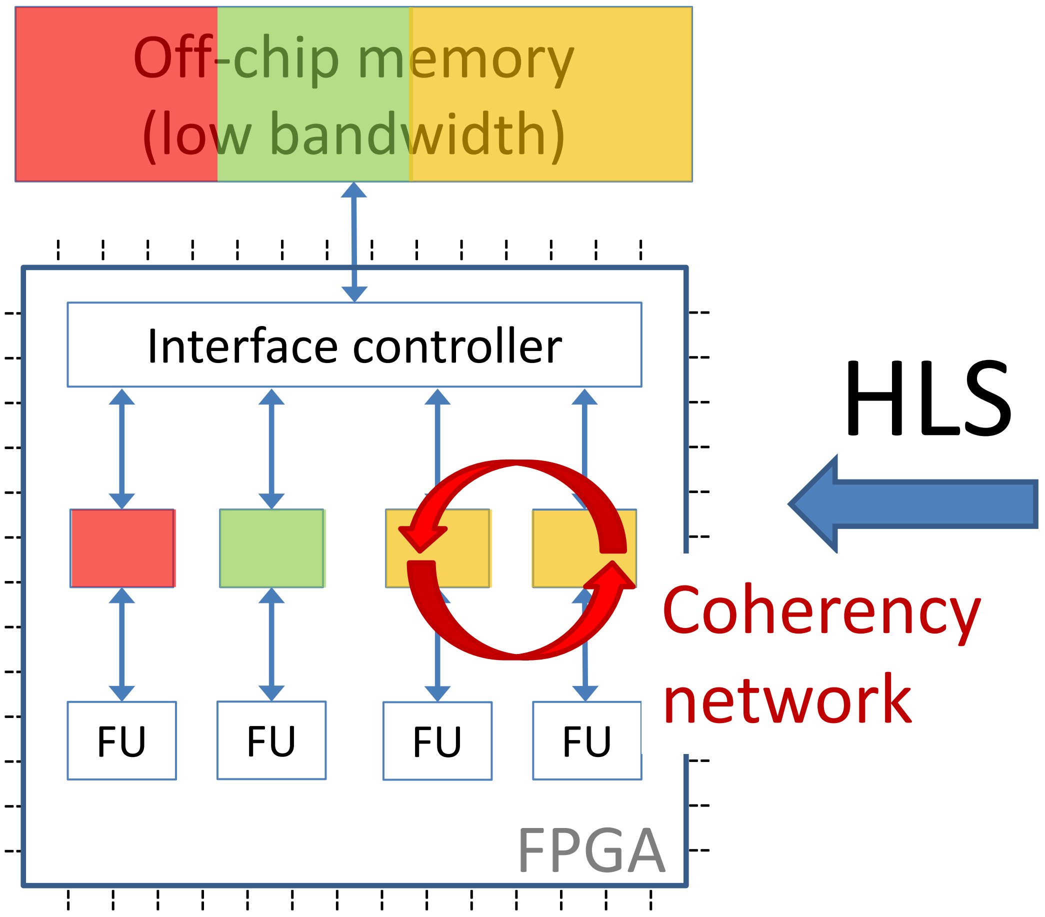


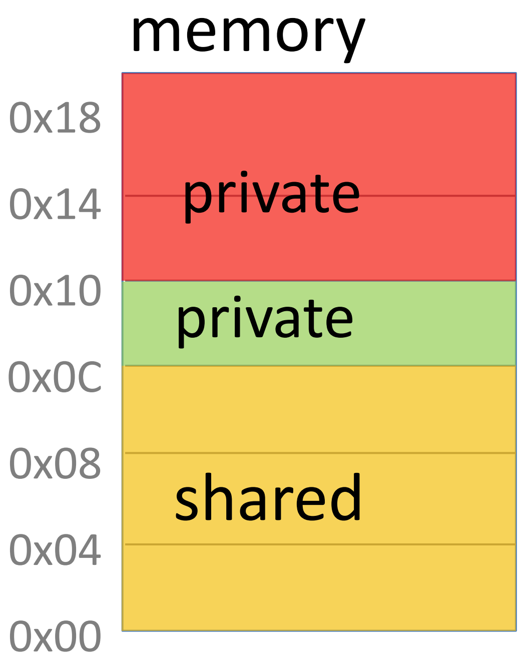




```
s = new stackRecord;
s->u = root;
s->n = 0;
while s!=0 do
    t = s;
    u = t->u;
    s = t->n;
```

Recap: MATCHUP [FPGA'15]





```
s = **new** stackRecord;
s->u = root;
s->n = 0;
while s!=0 do
    t = s;
    u = t->u;
    s = t->n;
```

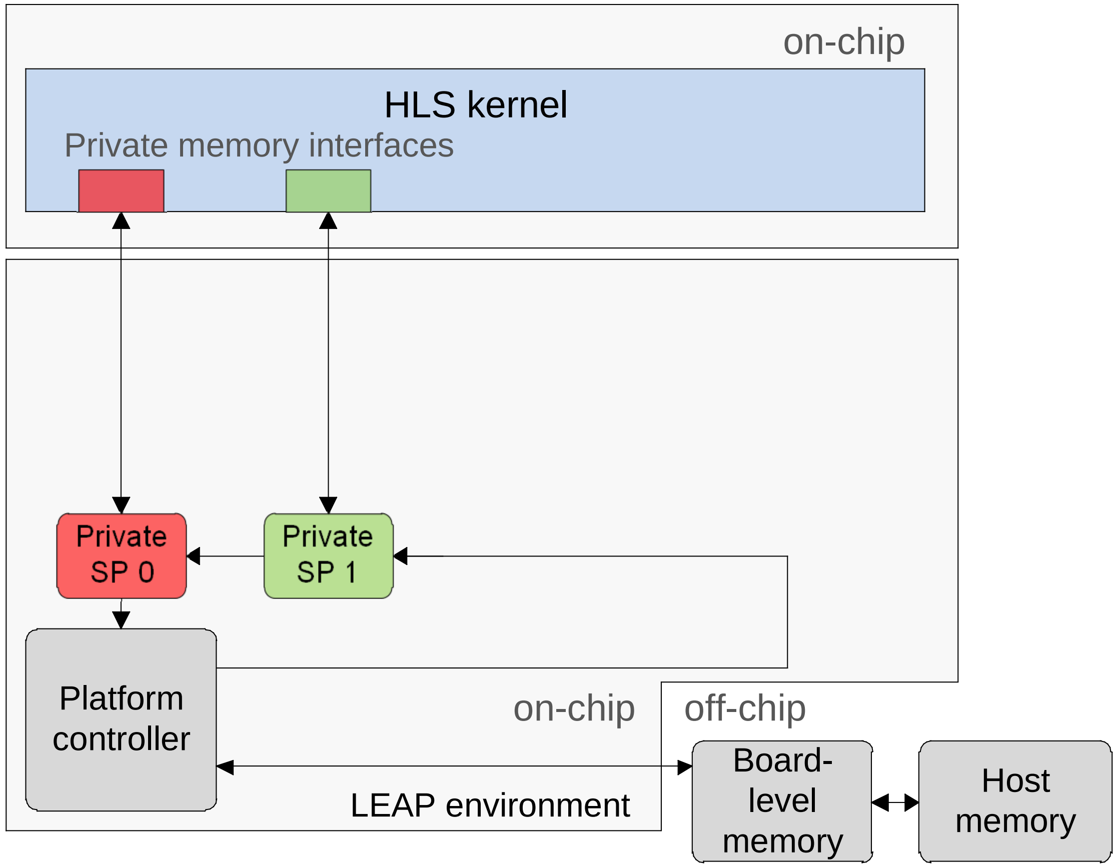
Imperial College London

LEAP's memory service

esa

LEAP scratchpads

- Interfaces to board-level and host-level memory
- Parallel cache hierarchy



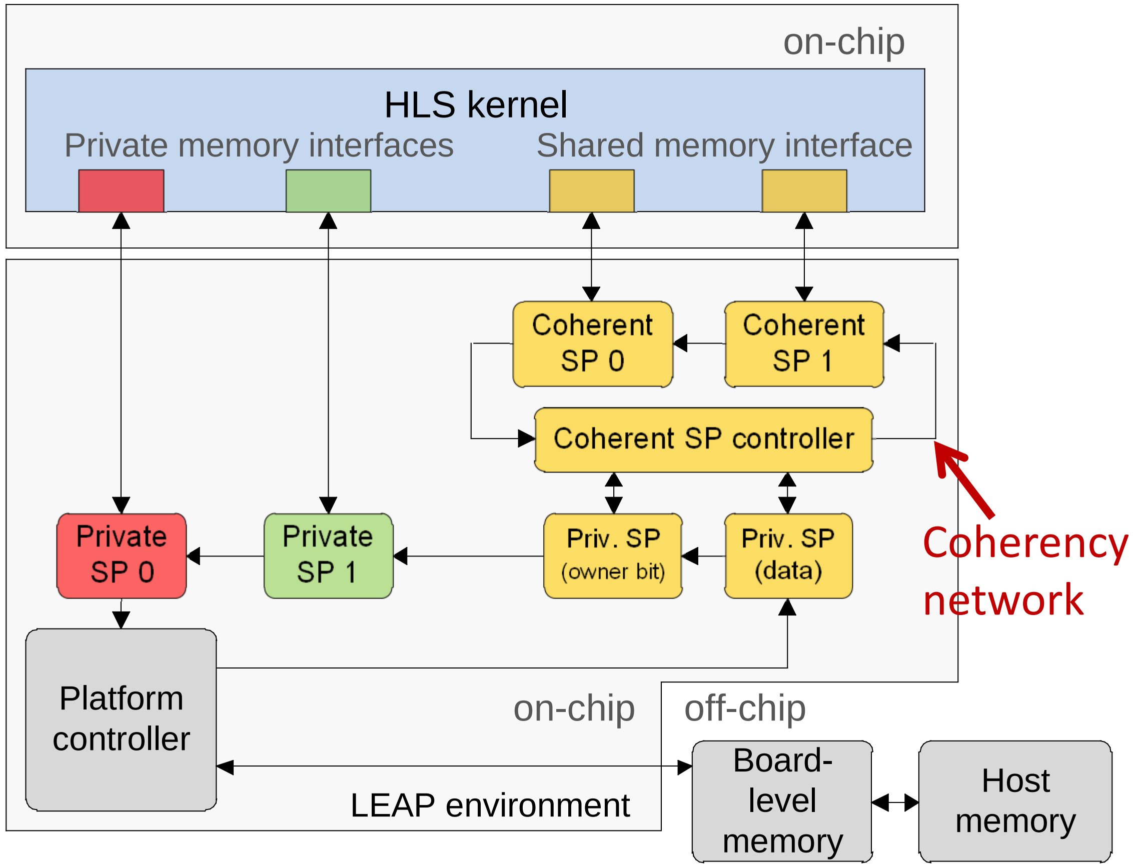
Imperial College London

LEAP's memory service

esa

LEAP scratchpads

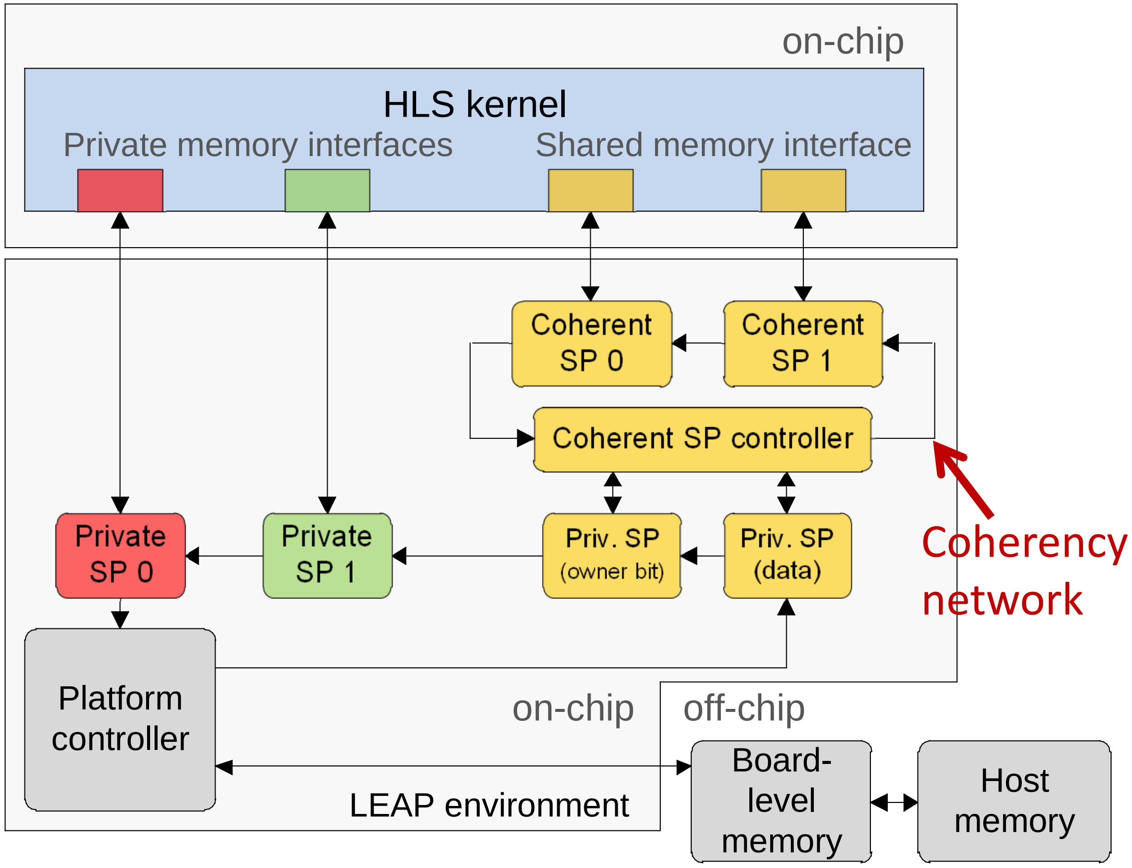
- Interfaces to board-level and host-level memory
- Parallel cache hierarchy
- Coherency networks



LEAP's memory service

LEAP scratchpads

- Interfaces to board-level and host-level memory
- Parallel cache hierarchy
- Coherency networks
- Underneath a unified **read-request, read-response, write** user interface



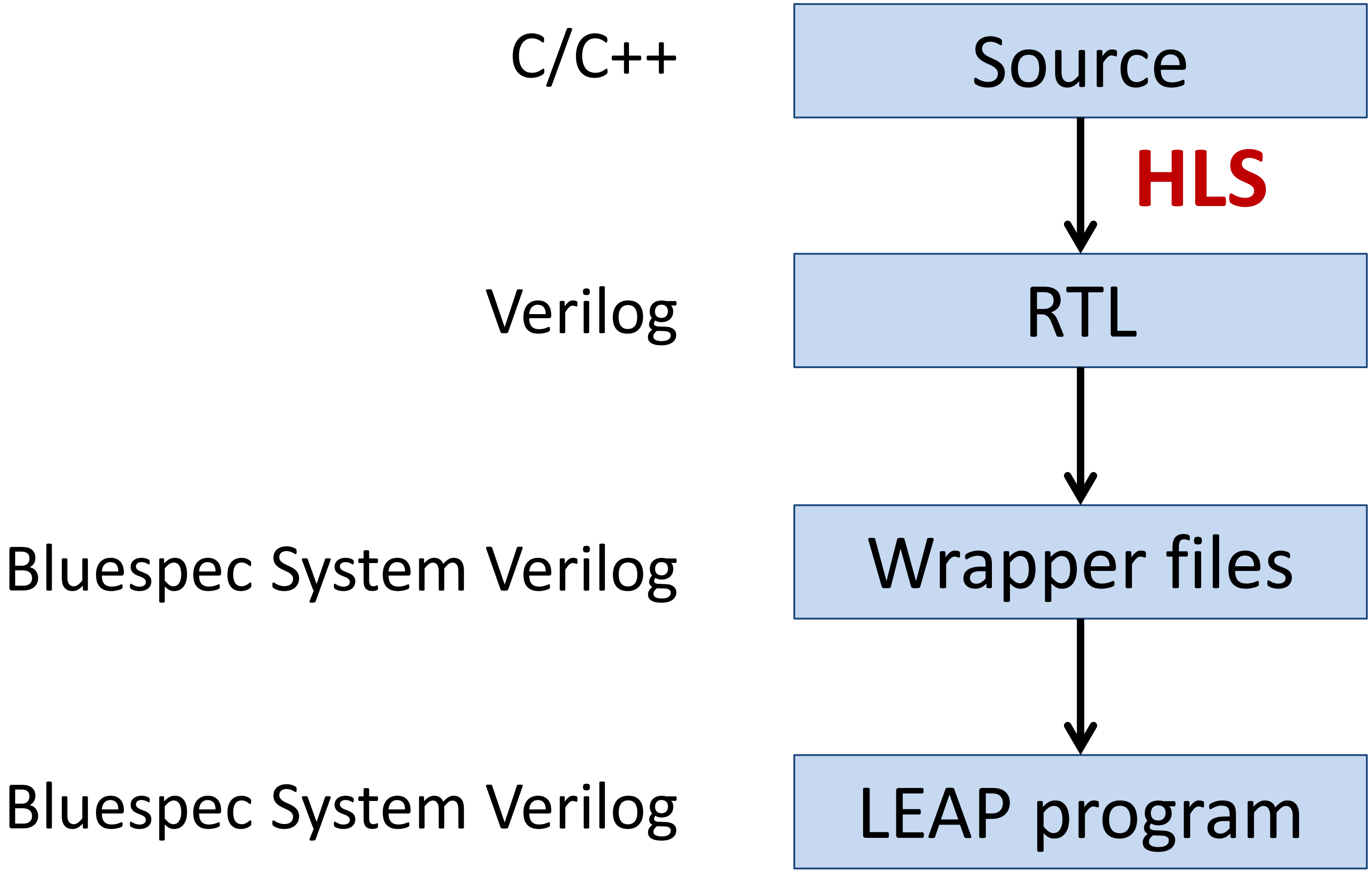
Imperial College London

What is needed?

esa

- The HLS kernel should be
  - insensitive to the memory response time,
  - compatible with a read-request, read-response, write protocol.
- Xilinx Vivado HLS can produce such designs
  - Top-level ports can be turned into bus interfaces (native ap_bus or AXI)
  - Core stalls while waiting for the bus response
  - A simple bridge between the ap_bus and scratchpad interface is required

Outline


C/C++
Source
HLS
Verilog
RTL
Bluespec System Verilog
Wrapper files
Bluespec System Verilog
LEAP program

Hello World

```
void hello_world (volatile int *bus0, volatile int *bus1)
{
        #pragma HLS INTERFACE ap_bus port=bus0
        #pragma HLS INTERFACE ap_bus port=bus1

        // internal block RAM
        int buffer[256];

        // read from bus0
        for (int i=0; i<256; i++)
                buffer[i] = bus0[i];

        // write to bus1
        for (int i=0; i<256; i++)
                bus1[i] = buffer[i]*buffer[i];
}
```

Native ap_bus interface (alternatively AXI can be specified here)

Imperial College London

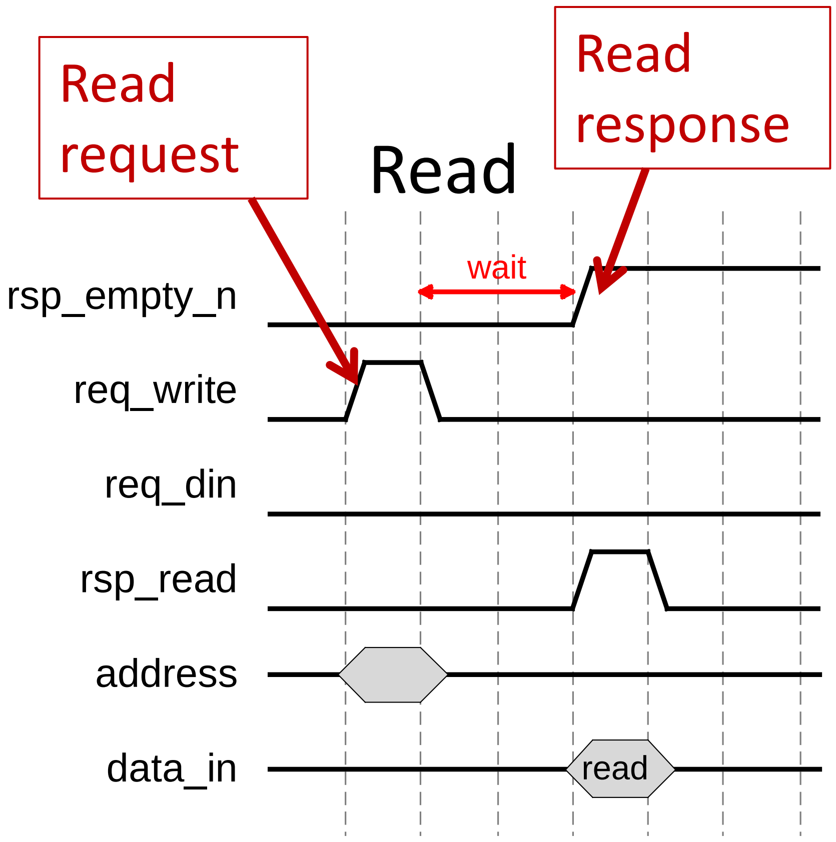
Generated RTL (top-level)

esa

```
module hello_world (
        ap_clk,
        ap_rst_n,
        ap_start,
        ap_done,
        ap_idle,
        ap_ready,
        bus0_req_din,
        bus0_req_full_n,
        bus0_req_write,
        bus0_rsp_empty_n,
        bus0_rsp_read,
        bus0_address,
        bus0_datain,
        bus0_dataout,
        bus0_size,
        …
```

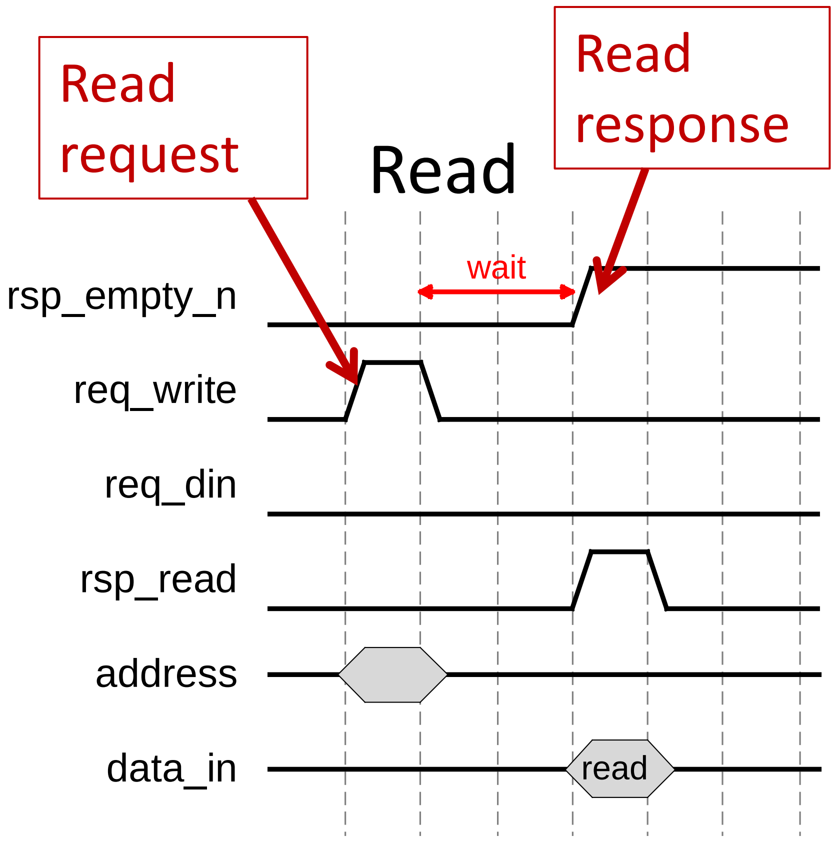
Vivado’s native ap_bus interface

Native ap_bus interface



*Vivado Design Suite User Guide, High-Level Synthesis, UG902 (v2014.3), 2014

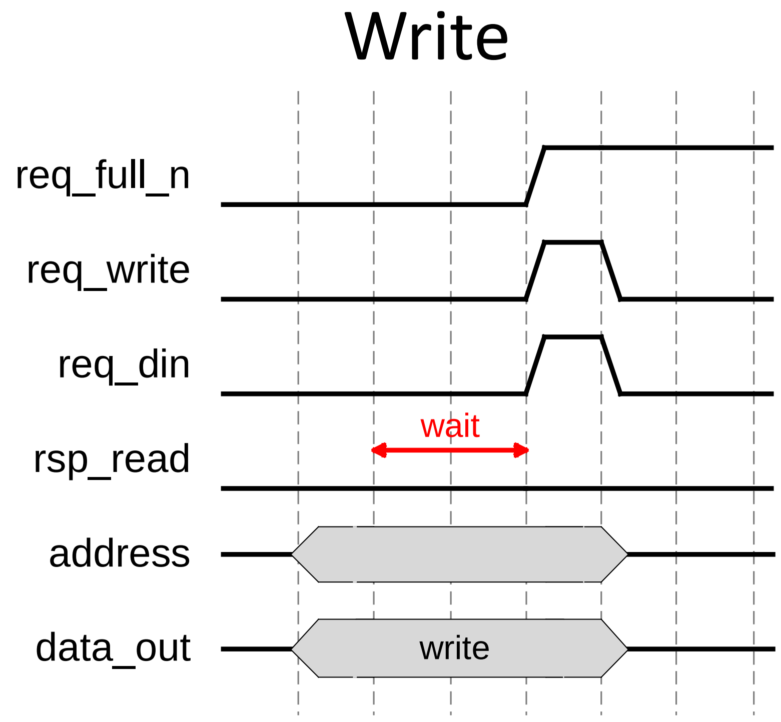
Import into Bluespec



```
import "BVI" hello_world =
   module mkMyIP( MyIP#( Bit#(n0), Bit#(n1) ) );
      // clock and reset
      …

      // bus I/O inputs
      method readRsp(data_in) enable(rsp_empty_n);


      // bus I/O outputs
      method address reqAddr() ready(req_write);


      …
endmodule
```

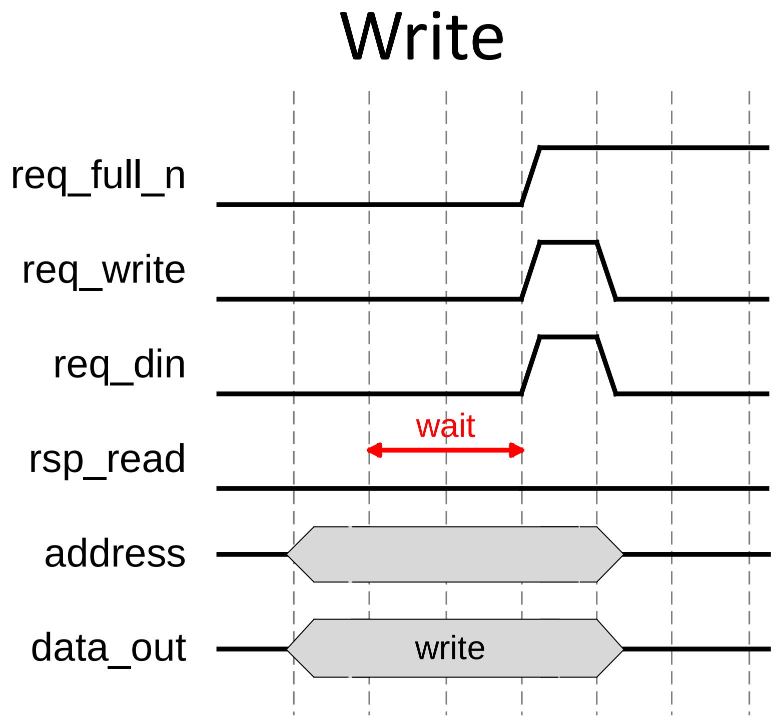
RTL signals

Import into Bluespec



```
import "BVI" hello_world =
   module mkMyIP( MyIP#( Bit#(n0), Bit#(n1) ) );
      // clock and reset
      ...

      // bus I/O inputs
      method readRsp(data_in) enable(rsp_empty_n);


      // bus I/O outputs
      method address reqAddr() ready(req_write);


      ...
endmodule
```

Import into Bluespec

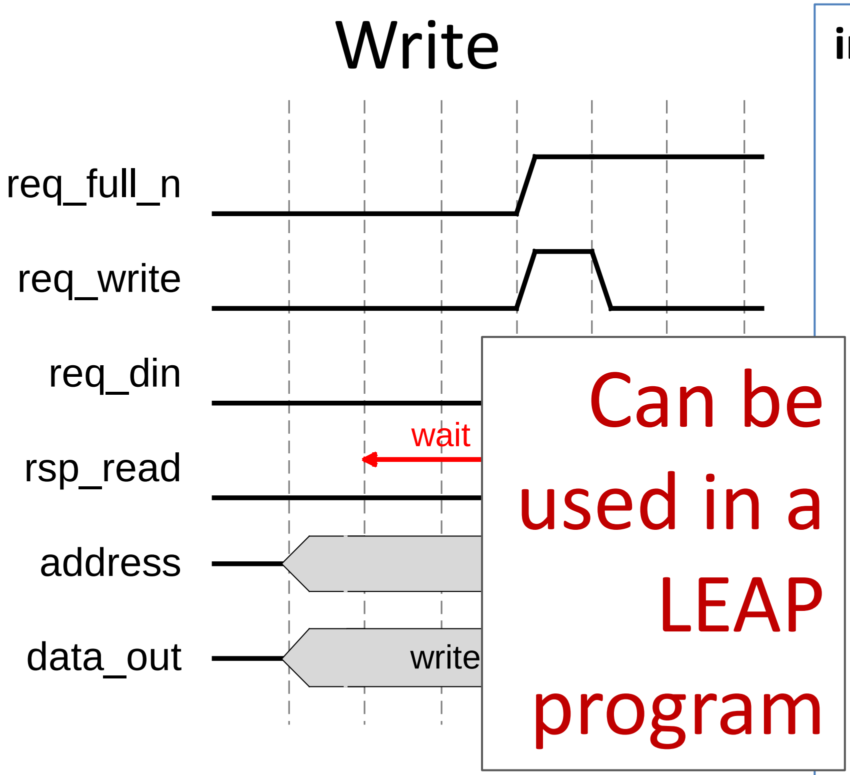


```
import "BVI" hello_world =
   module mkMyIP( MyIP#( Bit#(n0), Bit#(n1) ) );
      // clock and reset
      ...

      // bus I/O inputs
      method readRsp(data_in) enable(rsp_empty_n);
      method reqNotFull() enable(req_full_n);

      // bus I/O outputs
      method address reqAddr() ready(req_write);
      method data_out writeData() ready(req_write);
      method req_din writeReqEn() ready(req_write);
      ...
endmodule
```

Imperial College London

Import into Bluespec

esa



```
import "BVI" hello_world =
   module mkMyIP( MyIP#( Bit#(n0), Bit#(n1) ) );
      // clock and reset
      ...

      // bus I/O inputs
      method readRsp(data_in) enable(rsp_empty_n);
      method reqNotFull() enable(req_full_n);

      // bus I/O outputs
      method address reqAddr() ready(req_write);
      method data_out writeData() ready(req_write);
      method req_din writeReqEn() ready(req_write);
      ...
endmodule
```

Imperial College London

LEAP program

esa

```
rule readReq ( True );
        t_addr a = bus.reqAddr;
        requestFifo.enq(a);
endrule

rule readSPReq ( requestFifo.notEmpty );
        t_addr a = requestFifo.first;
        requestFifo.deq;

        scratchpad.readReq(a);
endrule

rule readSPResp ( True );
        t_data resp <- scratchpad.readRsp();
        bus.readRsp(resp);
endrule
```

Rule fires if the HLS core issues a read request at address *a*

Rule fires if read request pending and scratchpad able to take the request

Scratchpad responds and data is passed to the HLS core

Shared memory

```
critical_region: {
    #pragma HLS protocol fixed

    *access_critical_region = true;
    ap_wait();

     shared_bus[address] = data;
     ap_wait();

     *access_critical_region = false;
     ap_wait();
}
```

shared memory access

guard signals

Imperial College London

Shared memory

esa

```
critical_region: {
    #pragma HLS protocol fixed

    *access_critical_region = true;
    ap_wait();

     shared_bus[address] = data;
     ap_wait();

     *access_critical_region = false;
     ap_wait();
}
```

```
rule hlsAccessCriticalRegion ( True );
    Bool r = wrapper.accessCriticalRegion();
    if (r)
        lock.acquireLockReq(MY_LOCK);
    else
        lockReleaseFifo.enq(True);
endrule
```

acquire lock

release lock if no request pending

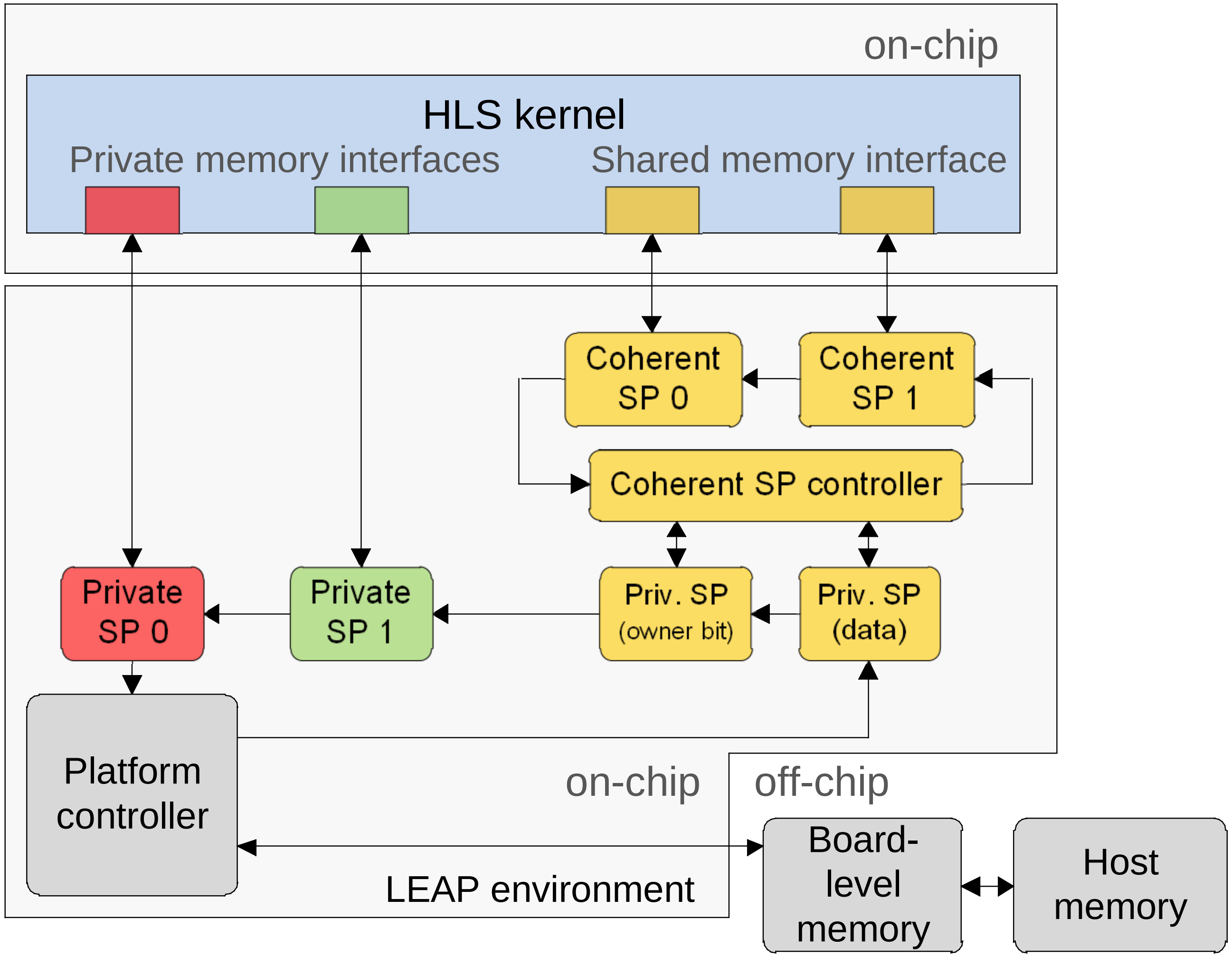
guard signals

shared memory access

Imperial College London

Recap: MATCHUP [FPGA'15]

esa



Recap: MATCHUP [FPGA’15]

		P	BRAM	Clock	Latency	
1	**Merger**					
	Parallelization (no caches)	4	62	10.0 ns	539 ms	x5
	Parallelization (with caches)	4	72	10.0 ns	115 ms	
2	**Tree deletion**					
	Parallelization (no caches)	4	91	10.0 ns	2208 us	x3
	Parallelization (with caches)	4	202	10.5 ns	711 us	
3	***K*-means clustering**					
	Parallelization (no caches)	4	125	10.0 ns	62 ms	x1.5
	Parallelization (with caches)	4	272	11.1 ns	42 ms	

Summary

- Connecting Vivado HLS native bus interfaces to LEAP scratchpads
- Private and shared address spaces
- Light-weight Bluespec wrappers enable access to board-level DRAM and host memory
- Automatic cache construction: average 3x speed-up in our benchmarks
- Code examples available at: **https://github.com/FelixWinterstein/LEAP-HLS**
- Next talk: Automatic wrapper generation

Thank you.

f.winterstein12@imperial.ac.uk

https://github.com/FelixWinterstein/LEAP-HLS