LEAP HLS
LEAP from a user’s perspective

Felix Winterstein
01 September 2015
• State-of-the-art HLS tools can build optimized custom memory systems on-chip
• But on-chip memory capacity often not sufficient
• 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]

Off-chip memory (low bandwidth)

Interface controller

HLS

FPGA

memory

0x18
0x14
0x10
0x0C
0x08
0x04
0x00

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]

Off-chip memory (low bandwidth)

Interface controller

HLS

memory

0x18
0x14
0x10
0x0C
0x08
0x04
0x00

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]

Off-chip memory (low bandwidth)

Interface controller

HLS

memory

0x18
0x14
0x10
0x0c
0x08
0x04
0x00

private

private


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]

Off-chip memory (low bandwidth)

Interface controller

HLS

Coherency network

memory

0x18
0x14
0x10
0x0C
0x08
0x04
0x00

private
private
shared

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

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

![Diagram of LEAP’s memory service]

- HLS kernel
- Private memory interfaces
- Platform controller
- off-chip
  - Board-level memory
  - Host memory
- on-chip
  - LEAP environment
LEAP's memory service

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
What is needed?

• 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

Source

HLS

RTL

C/C++

Verilog

Wrapper files

Bluespec System Verilog

LEAP program

Bluespec System Verilog
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];
}
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

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 "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 "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
Import into Bluespec

Can be used in a LEAP program

```
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
```
LEAP program

**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 access

```
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
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
Recap: MATCHUP [FPGA’15]

- HLS kernel
  - Private memory interfaces
  - Shared memory interface

- Platform controller
- Coherent SP 0
- Coherent SP 1
- Coherent SP controller
- Priv. SP (owner bit)
- Priv. SP (data)

- LEAP environment
- Board-level memory
- Host memory

- On-chip connections
- Off-chip connections
Recap: MATCHUP [FPGA’15]

<table>
<thead>
<tr>
<th>Step</th>
<th>Function</th>
<th>P</th>
<th>BRAM</th>
<th>Clock</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Merger</td>
<td>4</td>
<td>62</td>
<td>10.0 ns</td>
<td>539 ms</td>
</tr>
<tr>
<td></td>
<td>Parallelization (no caches)</td>
<td>4</td>
<td>72</td>
<td>10.0 ns</td>
<td>115 ms</td>
</tr>
<tr>
<td>2</td>
<td>Tree deletion</td>
<td>4</td>
<td>91</td>
<td>10.0 ns</td>
<td>2208 us</td>
</tr>
<tr>
<td></td>
<td>Parallelization (no caches)</td>
<td>4</td>
<td>202</td>
<td>10.5 ns</td>
<td>711 us</td>
</tr>
<tr>
<td></td>
<td>Parallelization (with caches)</td>
<td>4</td>
<td>202</td>
<td>10.5 ns</td>
<td>711 us</td>
</tr>
<tr>
<td>3</td>
<td>K-means clustering</td>
<td>4</td>
<td>125</td>
<td>10.0 ns</td>
<td>62 ms</td>
</tr>
<tr>
<td></td>
<td>Parallelization (no caches)</td>
<td>4</td>
<td>272</td>
<td>11.1 ns</td>
<td>42 ms</td>
</tr>
</tbody>
</table>
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