Case study: System optimization of FPGA-accelerated FIR application

Junning Fan
UNSW Sydney

Interface protocols in HLS

- The IO ports are defined with INTERFACE pragma:
  - applied on each argument of the synthesised function
    - Special case: **port =return** for the return value, as well as control signals
    - E.g. **#pragma INTERFACE s_axilite port=a**
- Generates data ports or AXI-compliant IO ports:
  - ap_none/ap_vld: data port
  - s_axi/s_axilite: AXI memory mapped ports
  - axis: AXI-stream streaming port
  - **Offer different IO throughput, latency and resource overhead.**

Burst/Streaming vs AXI-Lite

- AXI-Lite takes at least 3 clock cycles to transmit a word:
  - 1st Clock: Master send read request, read address
  - 2nd Clock: Slave presents data lines, assert read data valid
  - 3rd Clock: Master acknowledges the read by asserting data ready
- In bursting AXI-4 or streaming AXI-Stream:
  - Burst: Multiple words can be transmitted with only 1 addressing cycle
  - Streaming: No memory addressing
  - **Much higher throughput**

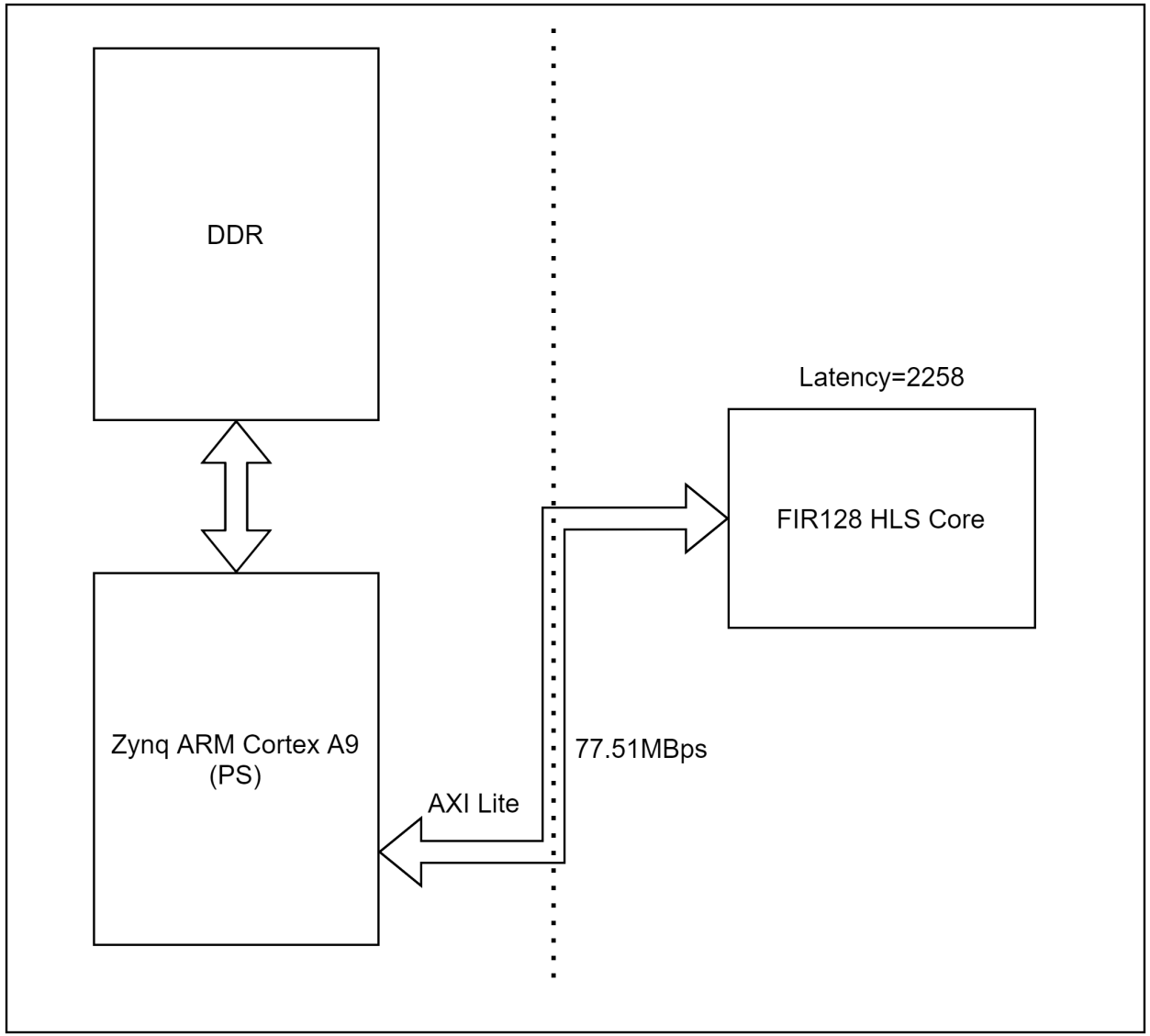
Function-wide pipelining

- PIPELINE pragma can be applied to a function body:
  - Pipeline all the operations in a function body
  - Same syntax as pipelining loops
  - Pipelining a function unrolls all loops in the function

HLS code for FIR

```
#define TAP_DEPTH 128
void fir(float* input, float* output, float
weights[TAP_DEPTH]) {
    static float shift_reg[TAP_DEPTH] = {};
    float acc = 0;
    TDL:for(int i = TAP_DEPTH-1; i >= 1; --i) {
        shift_reg[i] = shift_reg[i-1];
    }
    shift_reg[0] = *input;
    MAC:for(int i = 0; i < TAP_DEPTH; ++i) {
        acc += shift_reg[i] * weights[i];
    }
    *output = acc;
}
```

Default HLS implementation with AXI-Lite interface

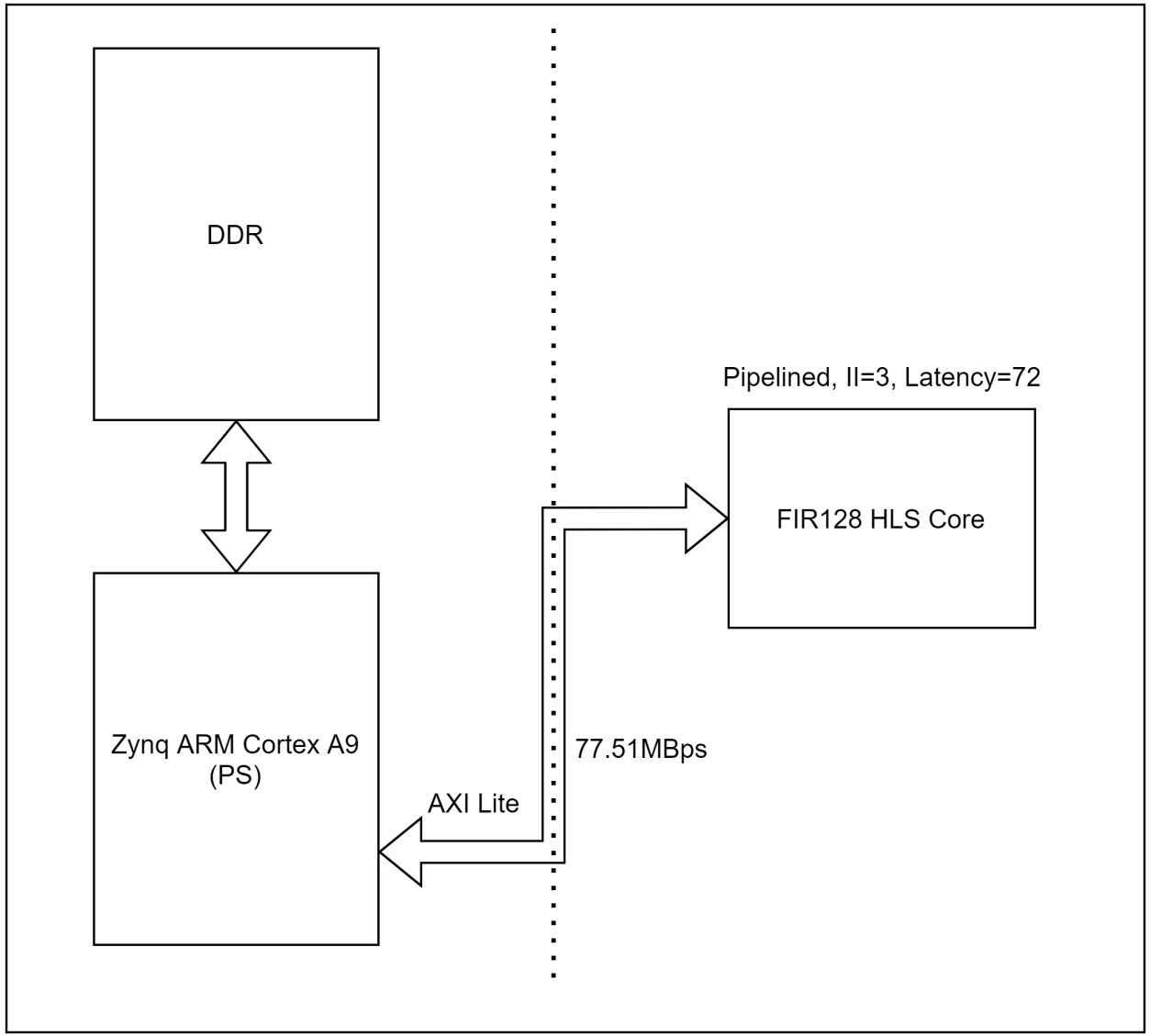


HLS pragma for baseline FIR

```
fir
    % HLS INTERFACE s_axilite port=return
    input
    % HLS INTERFACE s_axilite port=input
    output
    % HLS INTERFACE s_axilite port=output
    weights
    % HLS INTERFACE s_axilite port=weights
```

- Implement the input port as AXI-Lite (low performance without bursting)
- Implement the output port as AXI-Lite (low performance without bursting)
- Implement the interface of weights memory as AXI-Lite (low performance without bursting)

Pipelined HLS implementation with AXI-Lite interface



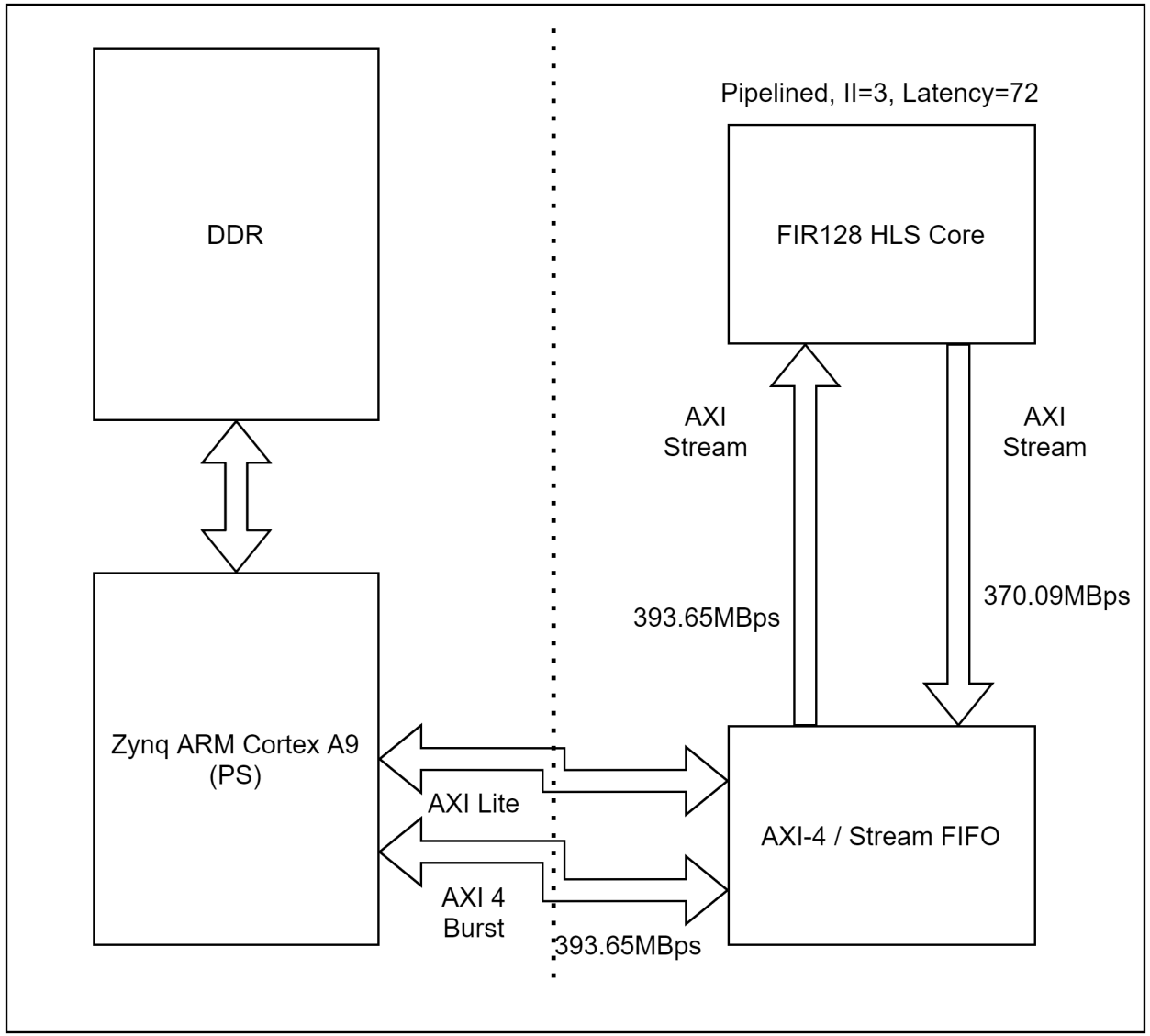
HLS pragma for pipelined FIR w/o bursting IO

```
fir
% HLS INTERFACE s_axilite port=return
% HLS PIPELINE II=3
input
% HLS INTERFACE s_axilite port=input
output
% HLS INTERFACE s_axilite port=output
weights
% HLS ARRAY_PARTITION variable=weights complete dim=1
% HLS INTERFACE s_axilite port=weights
shift_reg
% HLS ARRAY_PARTITION variable=shift_reg complete dim=1
TDL
MAC
```

Pipelines the function with initialize interval of 3

Partition the array to support simultaneous read/write in the unrolled loops

Pipelined HLS implementation with AXI-Stream interface



Special IO data for streaming in Vivado HLS

HLS code for streaming,pipelining FIR

Control signal for streaming IO type

```
typedef struct {
        fir_type_t val;
        ap_uint<1> last;}   io_type_t;

void firOptimized(hls::stream<fir_type_t>& input,
hls::stream<io_type_t>& output_stream, fir_type_t
weights[TAP_DEPTH]) {
                static fir_type_t shift_reg[TAP_DEPTH];
                fir_type_t acc = 0;
                shift_loop: for(int i = TAP_DEPTH-1; i >= 1; --i) {
                shift_reg[i] = shift_reg[i-1];
        }
        shift_reg[0] = input.read();
        mac_loop: for(int i = 0; i < TAP_DEPTH; ++i) {
                acc += shift_reg[i] * weights[i];
        }
        io_type_t output;
        output.val = acc;
        static ap_uint<7> counter = 0;
        if (counter++ == TAP_DEPTH-1) {
                output.last = 1;
        } else {
                output.last = 0;
        }
        output_stream.write(output);
}
```

HLS pragma for pipelined FIR with streaming

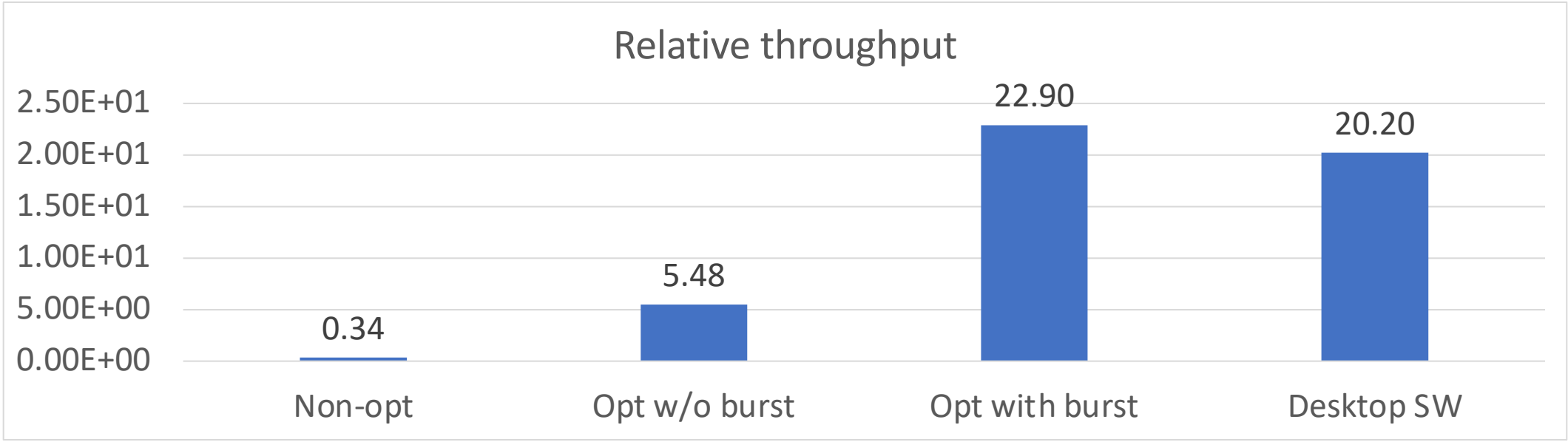
```
% HLS INTERFACE ap_ctrl_none port=return
input
% HLS INTERFACE axis register both port=input
output_stream
% HLS INTERFACE axis register both port=output_stream
weights
% HLS ARRAY_PARTITION variable=weights complete dim=1
% HLS INTERFACE s_axilite port=weights
shift_reg
% HLS ARRAY_PARTITION variable=shift_reg complete dim=1
shift_loop
% HLS UNROLL
```

AXI-stream interface

Resource Utilization

	Baseline	Pipe w/o burst	Pipe with stream
LUT	752	25321	28425
FF	613	38809	41534
DSP	5	215	215
Relative throughput	0.337	5.48	22.9

Relative throughput

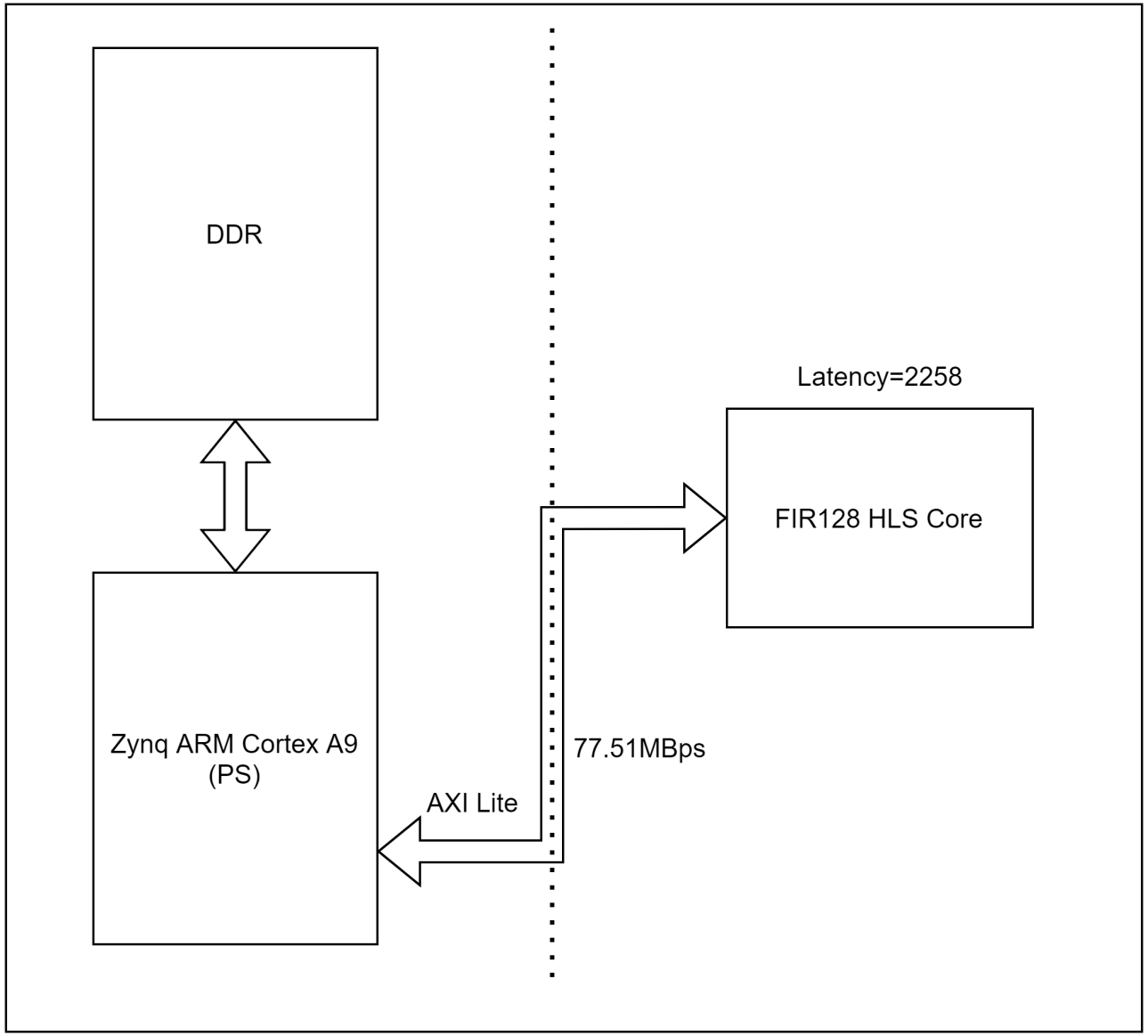


Takeaway messages 1

- Programmable logic implementation does NOT guarantee speed-up
- Pipelined/unrolled implementation can result in very large circuit

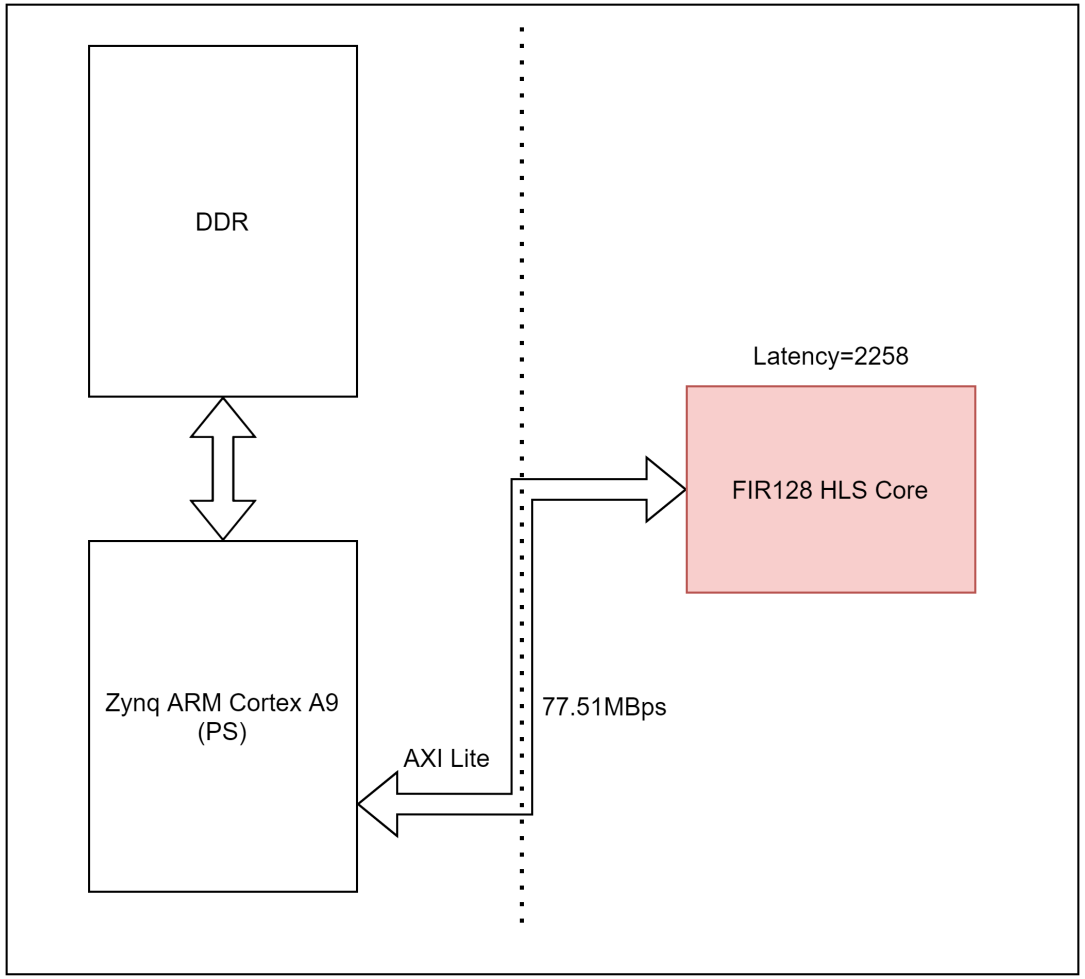
Bottleneck of each design

- Types of bottleneck?
- Compute-bound or IO bound?
- Identify bottlenecks of each design
  - Hint: compute the bandwidth needed for a design to run at full speed
- Assume:
  - Frequency of 160MHz



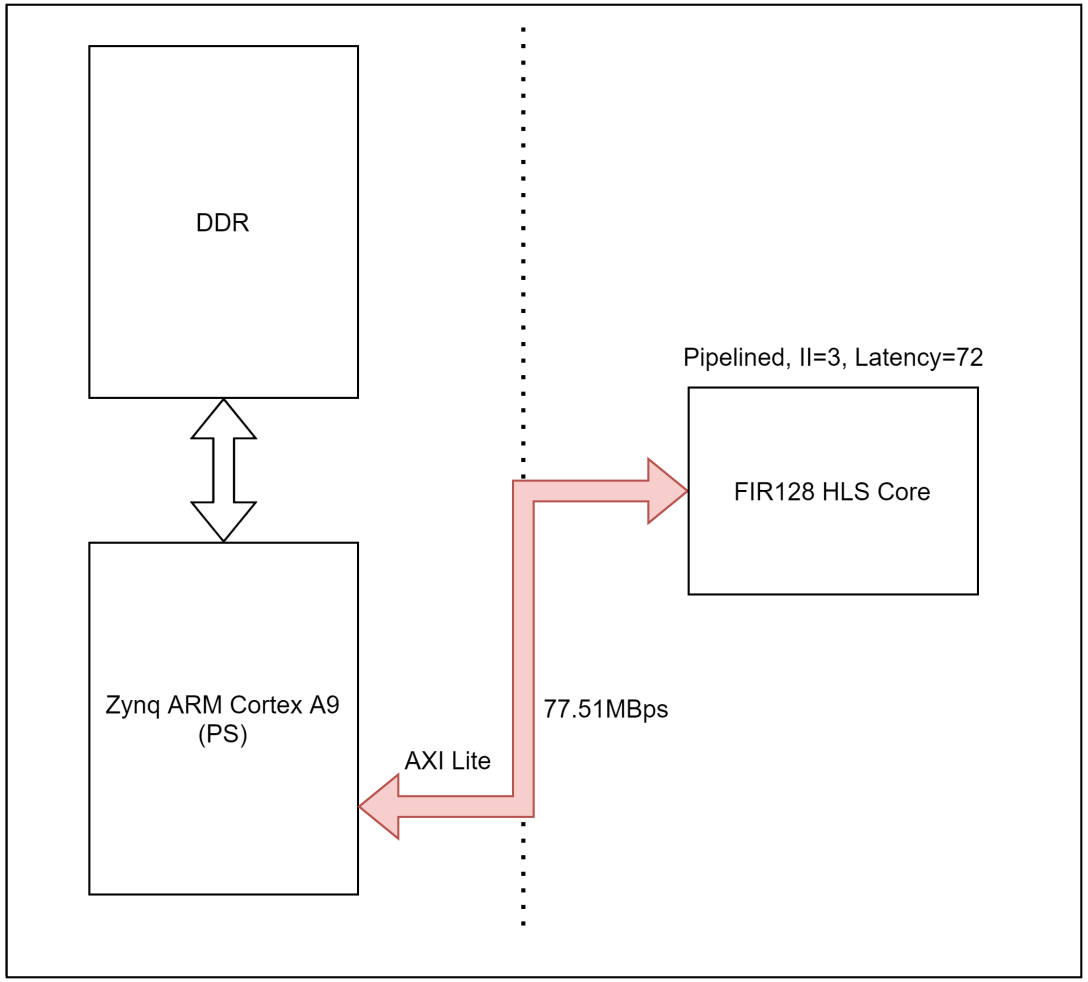
Baseline HLS

- Bandwidth needed: $\frac{4}{2256} * 160 * 10^6 =$ 283.687 KBps
- Compute-bound



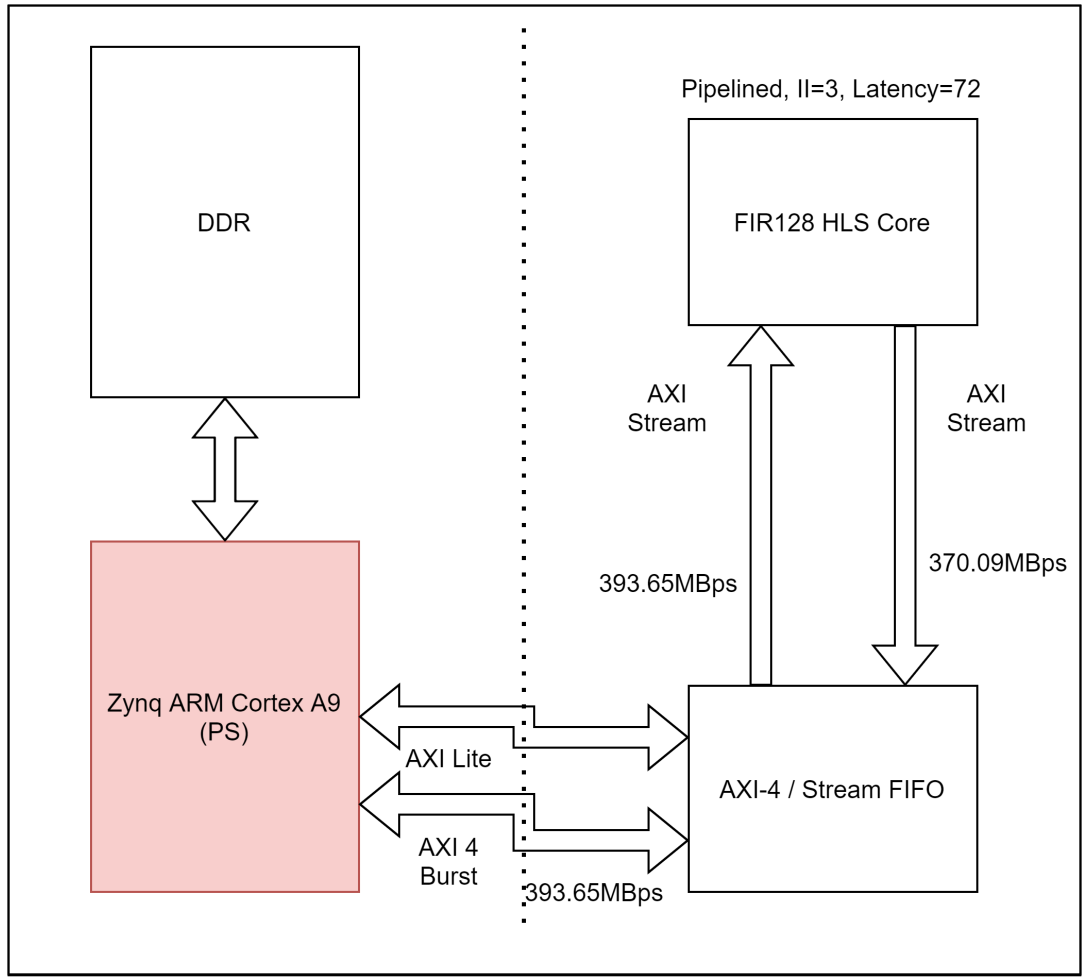
Pipelined HLS with AXI-Lite

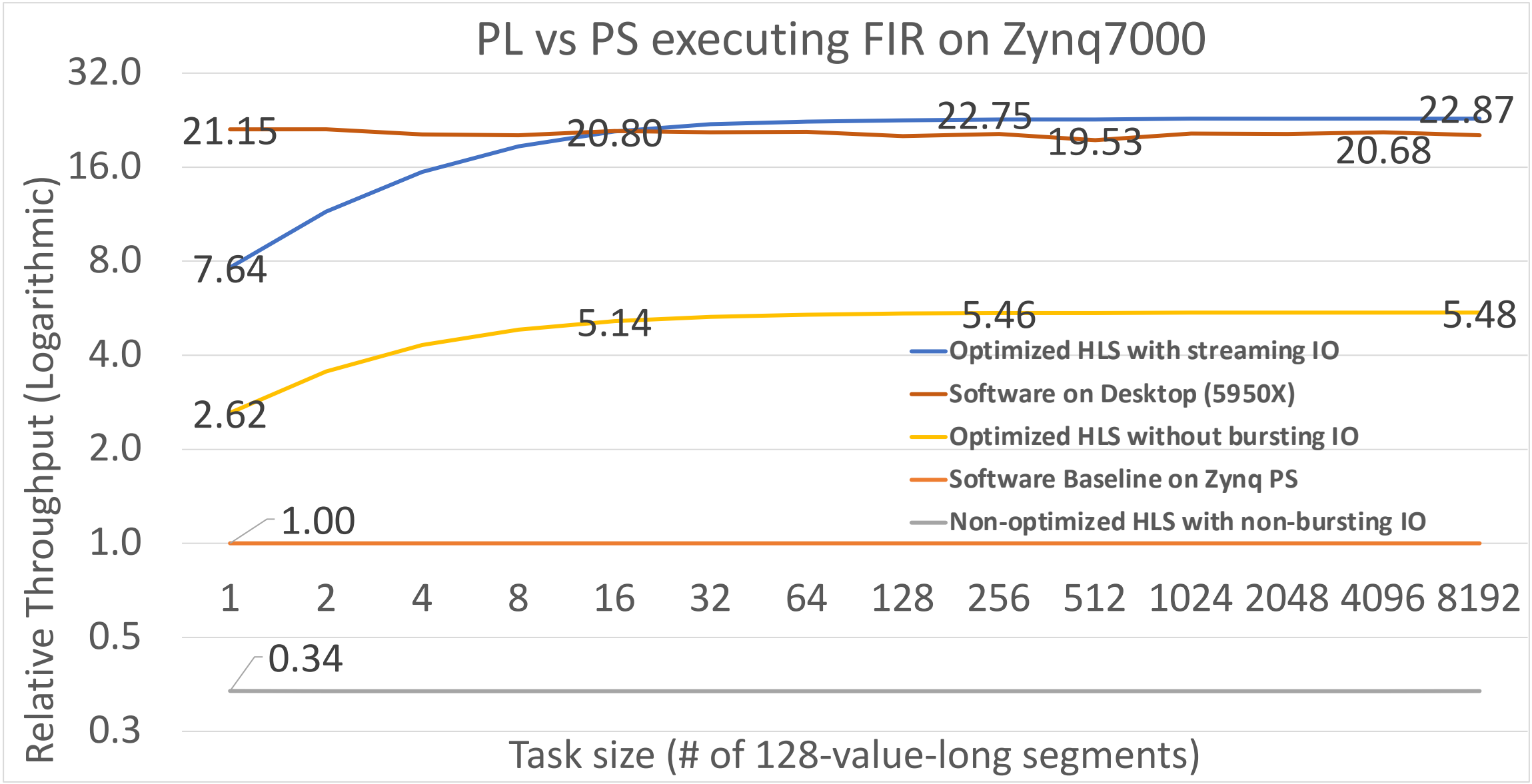
- Bandwidth needed: $\frac{4}{3} * 160 * 10^6 = 213.3\text{MBps}$
- IO-bound



Pipelined HLS with AXI-Stream

- Bandwidth needed: $\frac{4}{3} * 160 * 10^6 = 213.3\text{MBps}$
- Sub-optimal software implementation




PL vs PS executing FIR on Zynq7000
Relative Throughput (Logarithmic)
32.0
16.0
8.0
4.0
2.0
1.0
0.5
0.3
21.15
20.80
22.75
19.53
20.68
22.87
7.64
5.14
5.46
5.48
2.62
1.00
0.34
Optimized HLS with streaming IO
Software on Desktop (5950X)
Optimized HLS without bursting IO
Software Baseline on Zynq PS
Non-optimized HLS with non-bursting IO
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
Task size (# of 128-value-long segments)

Takeaway messages 2

- The IO interface should be chosen carefully to meet the bandwidth requirement of HLS designs
- PS system (software and DDR) should be able to feed data fast enough to PL system

Thank you!