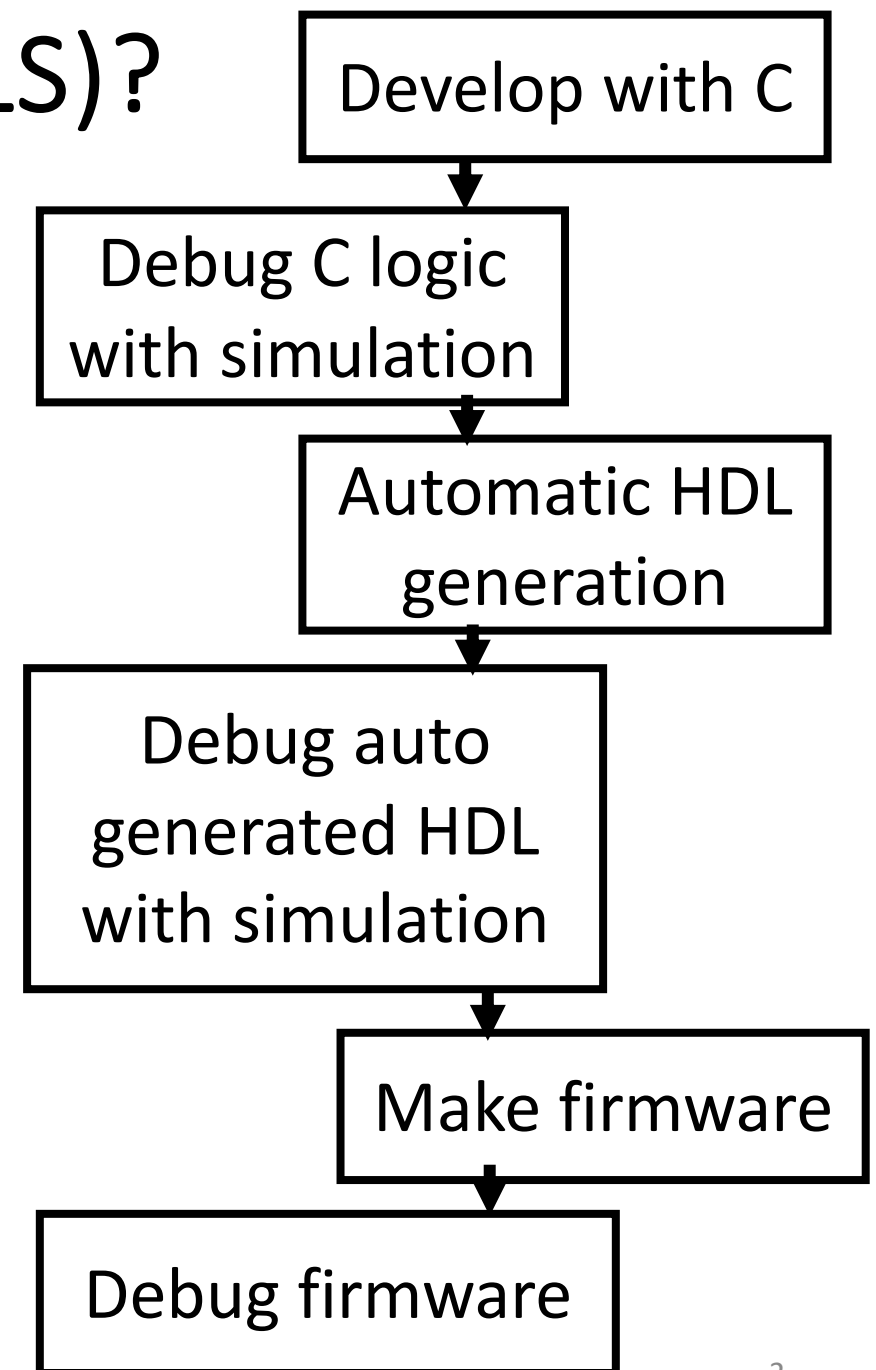


Introduction to High Level Synthesis (HLS)

What is High Level Synthesis (HLS)?

- User develops FPGA logic with C++
 - HLS automatically creates HDL version of C++
 - HDL module becomes a **IPCore**
- User generates firmware with the **IPCore**



Difficulty with HLS

- C++ was not developed for FPGAs
 - Missing features: Arbitrary bit size, parallelized logic, ...
 - Need to create new C++ syntax to be able to handle missing features. Only used in HLS.
 - ❖ `ap_int<N>` used for arbitrary bit size
 - ❖ `#pragma` used for handling parallelized logic
 - ❖ ...

Data type: Arbitrary Precision integers

- C++ only provides limit set of integer types

➤ char (8 bits), short (16 bits), int (32 bits), ...

- HLS types library —————→

➤ Has templated class

```
#include <ap_int.h>
```

```
ap_uint<9> x; // 9 bit unsigned  
ap_int<10> x; // 9 bit signed
```

➤ Variable inside <N> is **number of bits**

Data type: Arbitrary Precision integers

- `ap_fixed`<Total bits,
Integer bits, Rounding
mode, Overflow mode>

```
#include <ap_int.h>
```

```
// 9 bit unsigned, 5 bit fraction  
ap_ufixed<9,4,AP_TRN, AP_WRAP> x;
```

```
// 11 bit unsigned, 7 bit fraction  
ap_fixed<11,4,AP_TRN, AP_WRAP> y;
```

Rounding mode: `AP_TRN` (=truncation), `AP_RND`, ...

Overflow mode: What to do when number goes above number of bits.

`AP_WRAP`: Just removes MSB bit. Doesn't use resources.

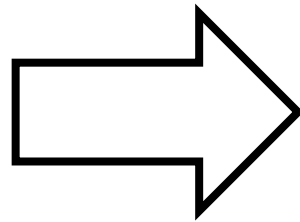
`AP_SAT`: Saturate values to min and max possible bits.

Designing multiple HDL modules with C++

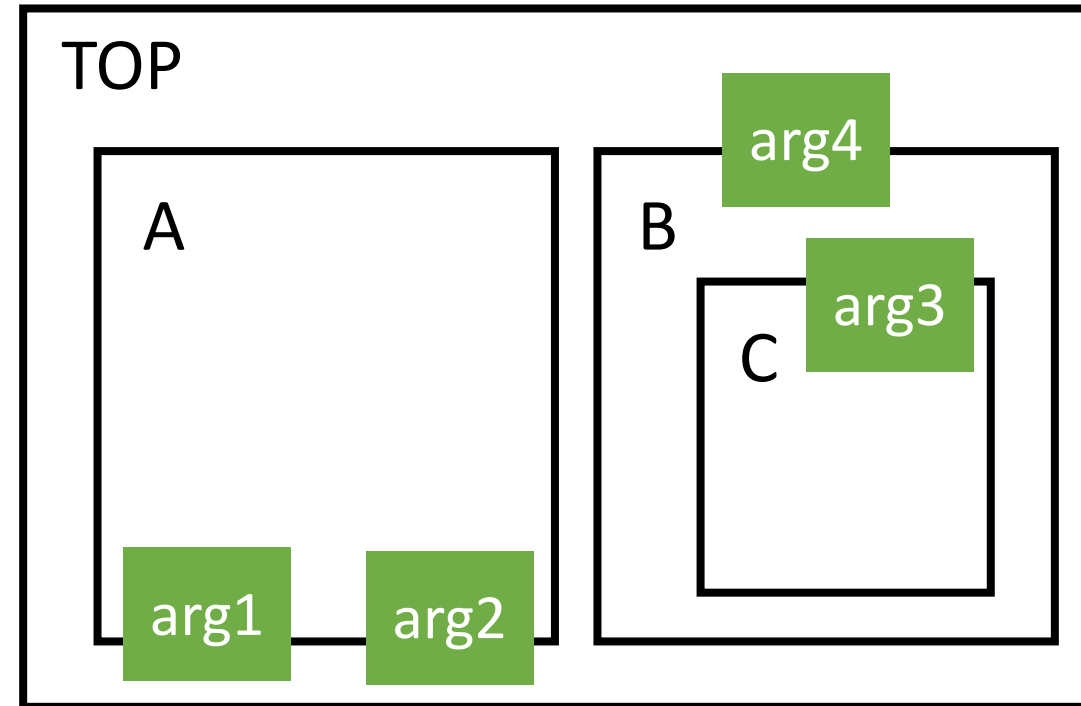
- C++ functions usually translated to HDL modules.

➤ Arguments of functions become ports of HDL module

```
void A(arg1, arg2) { ... }  
void C(arg3) { ... }  
void B(arg4) { C(...); }  
void TOP() { A(...); B(...); }
```



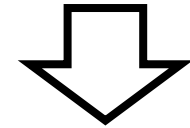
Translation
of HLS



C++ arrays in HLS

- HLS assumes that C++ arrays are a memory component.
- Default memory component is BRAM.
 - If we use only one input address ports.

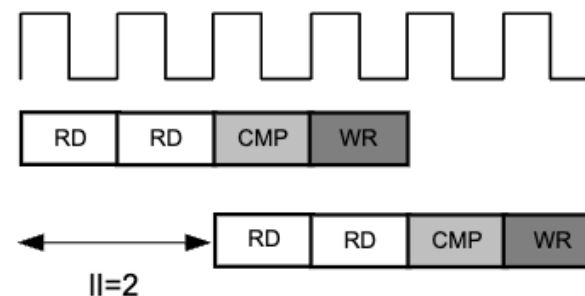
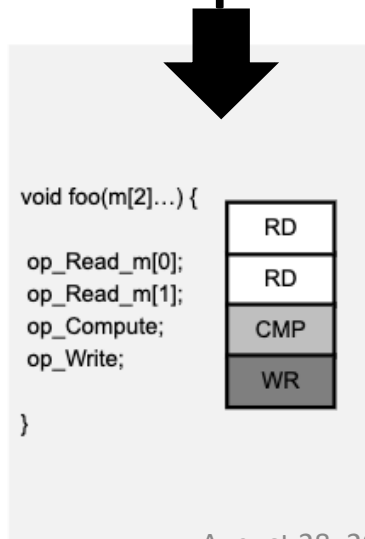
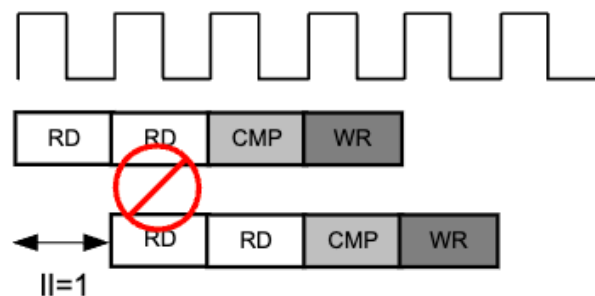
```
ap_uint<16> higgs[4]
```



0	higgs[0]
1	higgs[1]
2	higgs[2]
3	higgs[3]

Address Data
Memory

Cannot work

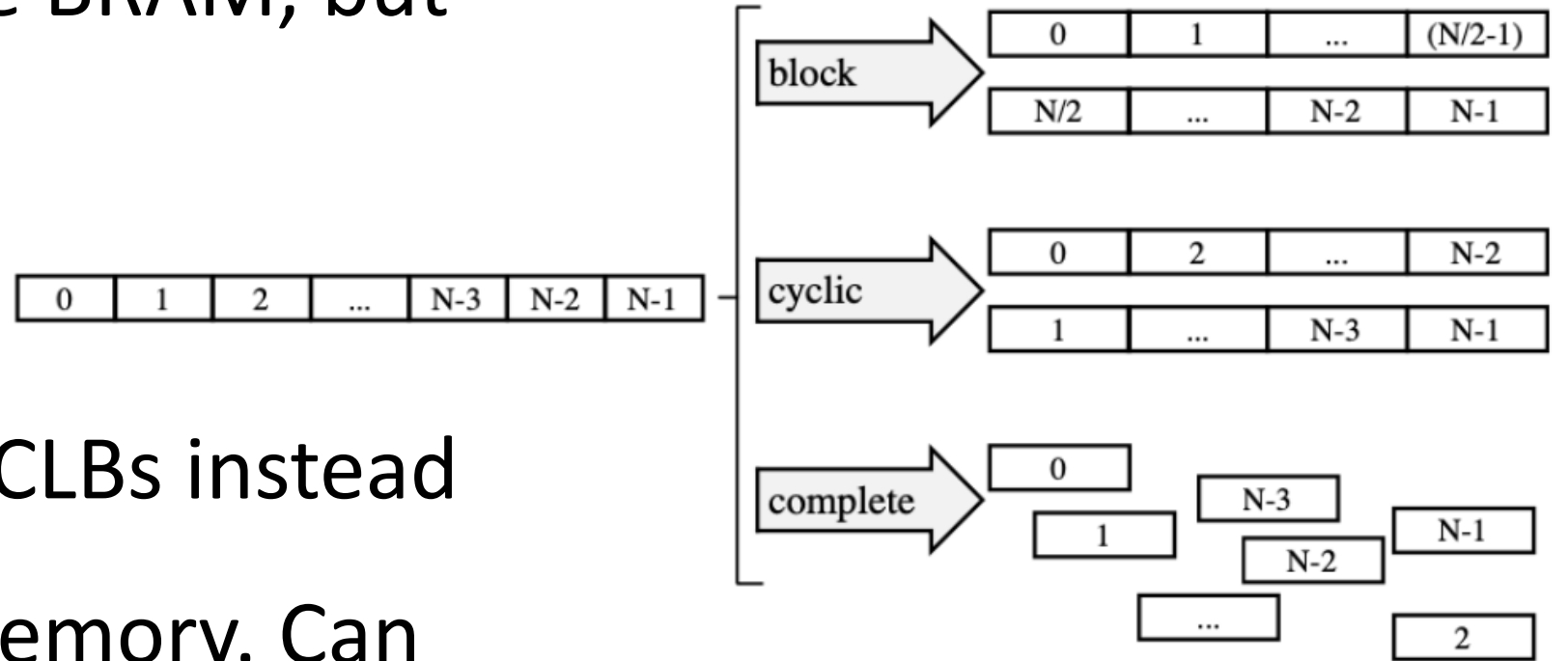


C++ arrays in HLS

- Array memory can be split up into parts.

➤ block, cycle: Use BRAM, but split array data.

Figure 57: Array Partitioning



X14251

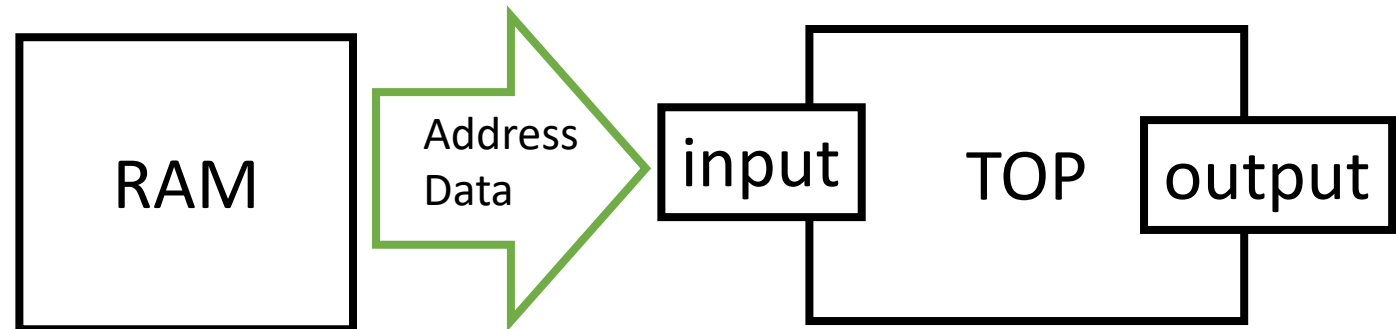
➤ complete: Use CLBs instead (LUTs) for the memory. Can access any LUT any time.

C++ arrays for interface of TOP function

- C++ TOP function can have array as argument

```
void top (ap_uint<16> input[4]; ap_uint<16> output);
```

- HLS assumes the input array is a memory chip giving data to function.



Data is put in sequentially

Loops in HLS

- A C++ loop can be translated to many

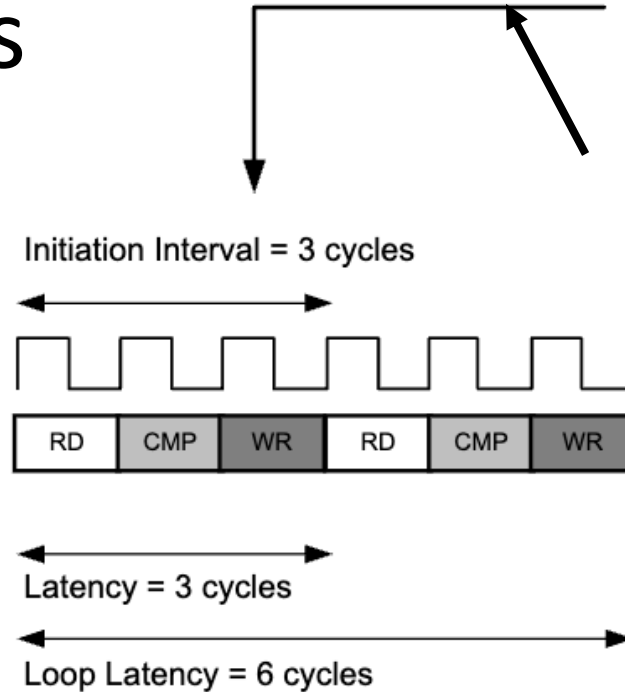
types of HLS loops

- Some loops can not be pipelined

```
while (a != b) {  
    if (a > b) a -= b;  
    else b -= a;  
}
```

- A loop with multiple statements can be pipelined.

Without Pipelining

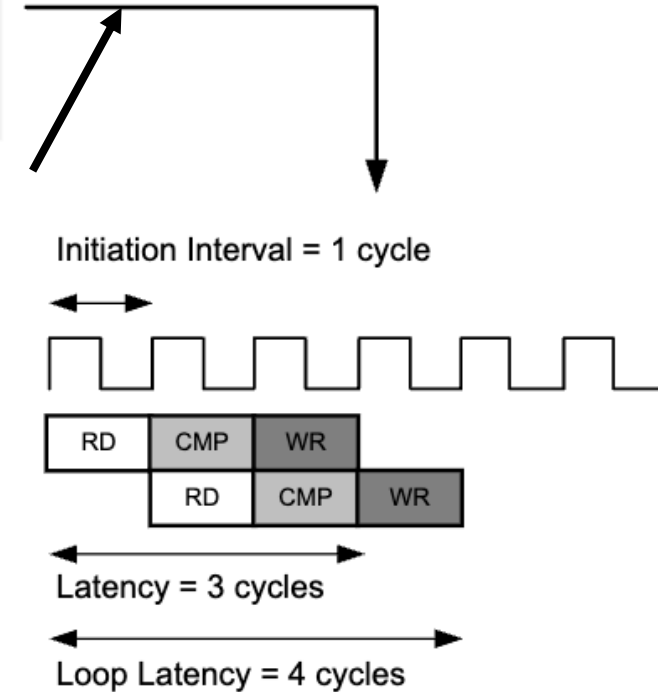


```
Loop:for(i=1;i<3;i++) {  
    op_Read;  
    op_Compute;  
    op_Write;  
}
```

Can be set with
`#pragma`



With Pipelining

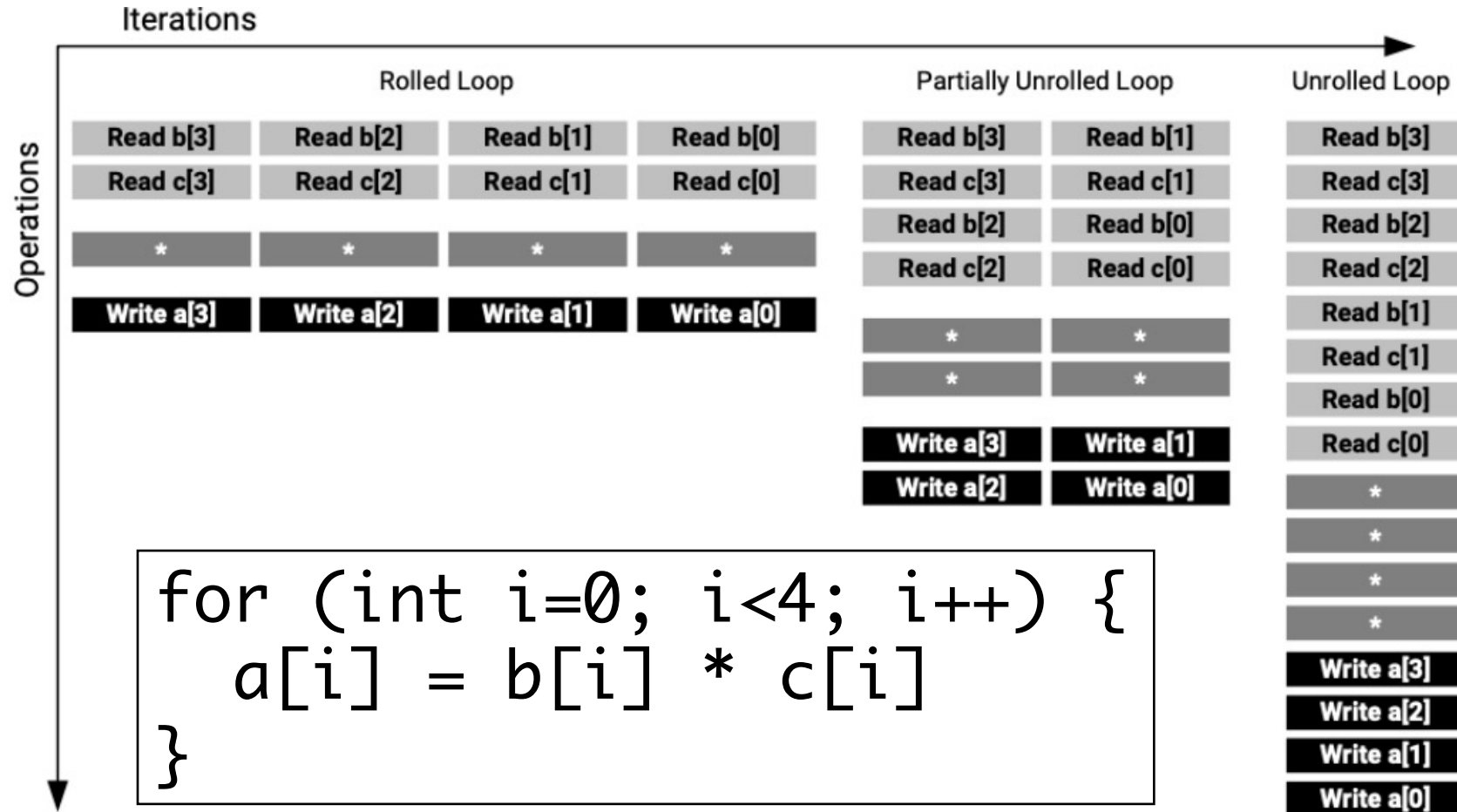


X14770-070115

Loops in HLS (Unrolling)

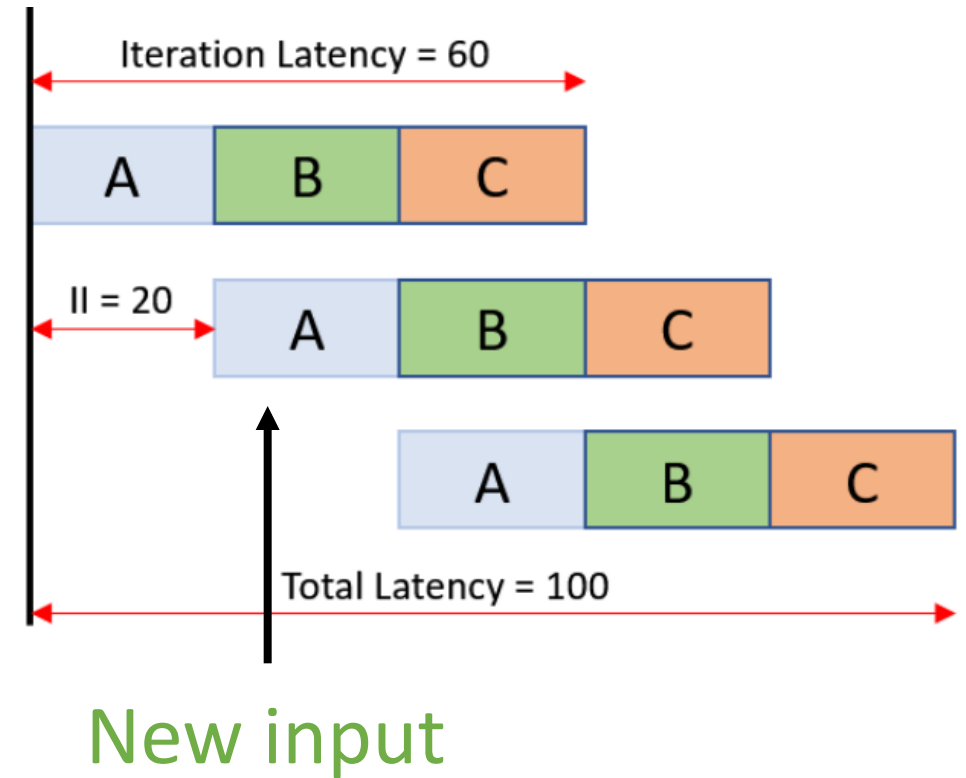
- Some loop code can be done totally in parallel.

➤ They can be “unrolled”. Default in HLS is rolled loops.



Concepts/Terms that HLS uses

- **Iteration latency:** Same as pipeline latency
- **Initiation Interval (II):** Number of clock cycles for **new input**.
- There is also **total latency**



Timing optimization in HLS

Figure 6: Sequential Execution - Two Runs

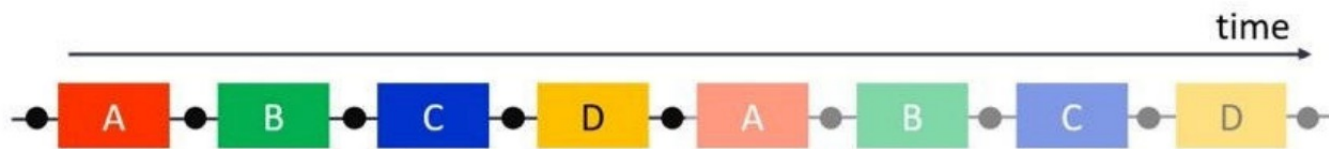


Figure 7: Task Parallelism within a Run

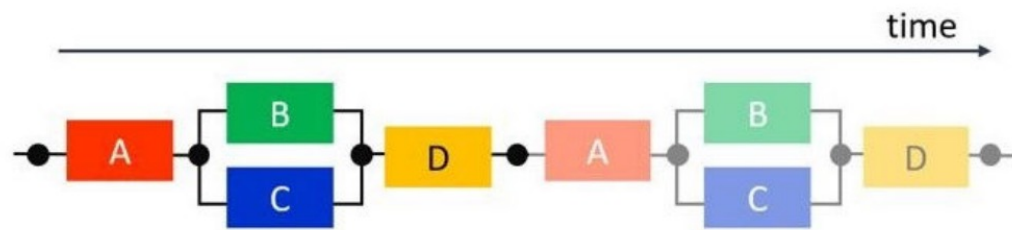
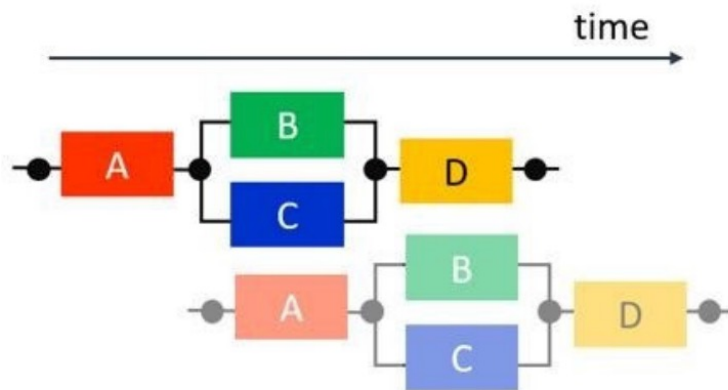


Figure 8: Task Parallelism with Pipelining



- Let's say we have below

A(in C1, out C2, out C2)

B(in C2, out C4)

C(in C3, out C5)

D(in C4, in C5, out C5)

- Many ways of implementation...
- Set using `#pragm`

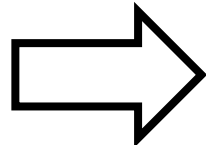
What does HLS need to make HDL?

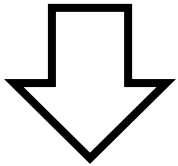
- C++ function/module source files
- C++ testbench source files
- Configuration of project: FPGA model, clock freq., ...

What can HLS do?

- C++ simulation: Using testbench C++ code, simulate C++ function code.
- C++ synthesis: Translates C++ into HDL
- C++/HDL co-simulation: Using stimuli and golden output from C++, compare with HDL simulation output.

C++ simulation

- Need to write function file.
- Need to write testbench file. 
- Simulate!



```
INFO: [vitis-run 60-791] Total elapsed time: 0h 0m 15s  
C-simulation finished successfully  
INFO: [SIM 211-210] Code Analyzer finished
```

```
#include "ffn.h"  
  
int main() {  
    // Get input  
    ap_fixed<18,4,AP_TRN,AP_SAT> x_in = 1;  
    ap_fixed<18,4,AP_TRN,AP_SAT> y_in = 2;  
    ap_fixed<18,4,AP_TRN,AP_SAT> z_out;  
  
    // Run function  
    ffn(x_in, y_in, z_out);  
  
    // Get gold ouptut  
    ap_fixed<18,4,AP_TRN,AP_SAT> gold;  
    // Could get gold values from elsewhere  
    ffn(x_in, y_in, z_out: gold);  
  
    // Compare with gold output  
    int result = 0;  
    if (gold == z_out) {  
        printf(format: "Good");  
        result = 0;  
    } else {  
        printf(format: "Error");  
        result = 1;  
    }  
    return result;  
}
```


FFN function file

- Feedforward network with 2-8-1 node structure.

```
#include "ffn.h"

void ffn(
    ap_fixed<18,4,AP_TRN,AP_SAT> x_in,
    ap_fixed<18,4,AP_TRN,AP_SAT> y_in,
    ap_fixed<18,4,AP_TRN,AP_SAT> &z_out
) {
    #pragma HLS pipeline II=1

    static const ap_fixed<18,4,AP_TRN,AP_SAT> w1_x[8] = {1.7,1.5,0.3,0.13,1.11,0.17,0.19,0.23};
    static const ap_fixed<18,4,AP_TRN,AP_SAT> w1_y[8] = {0.13,1.23,0.11,1.17,0.19,0.17,1.27,0.23};
    static const ap_fixed<18,4,AP_TRN,AP_SAT> b1[8] = {0.3,0.27,1.19,0.17,0.13,1.37,0.29,0.1};
    static const ap_fixed<18,4,AP_TRN,AP_SAT> w2[8] = {0.23,1.11,0.27,1.3,0.11,0.17,1.13,0.7};
    static const ap_fixed<18,4,AP_TRN,AP_SAT> b2 = 3;

    ap_fixed<18,4,AP_TRN,AP_SAT> s0_inputs[2];           // {x,y}
    ap_fixed<18,4,AP_TRN,AP_SAT> l1_mul_products[2][8]; // [0]=x path, [1]=y path
    ap_fixed<18,4,AP_TRN,AP_SAT> l1_pre_acts_wide[8];
    ap_fixed<18,4,AP_TRN,AP_SAT> l1_hidden_act_q[8];
    ap_fixed<18,4,AP_TRN,AP_SAT> l2_mul_products[8];
    ap_fixed<18,4,AP_TRN,AP_SAT> l2_pair_sums_stage4[4];
    ap_fixed<18,4,AP_TRN,AP_SAT> l2_group_sums_stage5[2];
    ap_fixed<18,4,AP_TRN,AP_SAT> z_reg = 0;
    ap_fixed<18,4,AP_TRN,AP_SAT> acc = 0;

    s0_inputs[0] = x_in;
    s0_inputs[1] = y_in;

    for (int i=0; i<8; ++i) {
        l1_mul_products[0][i] = s0_inputs[0] * w1_x[i];
        l1_mul_products[1][i] = s0_inputs[1] * w1_y[i];
        l1_pre_acts_wide[i] = l1_mul_products[0][i] + l1_mul_products[1][i] + b1[i];
        if (l1_pre_acts_wide[i] <= 0) l1_hidden_act_q[i] = 0;
        else l1_hidden_act_q[i] = l1_pre_acts_wide[i];

        l2_mul_products[i] = l1_hidden_act_q[i]*w2[i];
        acc += l2_mul_products[i];
    }

    z_out = acc + b2;
}
```


C++ synthesis

- Can change HLS C++ code to HDL.

Automatically generated HDL code.

```
1339 add_ln32_5_fu_1174_p2 <= std_logic_vector(unsigned(add_ln32_4_fu_1168_p2) + unsigned(ap_const_lv19_4C28));
1340 add_ln32_6_fu_1194_p2 <= std_logic_vector(signed(sext_ln32_10_fu_1191_p1) + signed(sext_ln32_9_fu_1188_p1));
1341 add_ln32_7_fu_1200_p2 <= std_logic_vector(unsigned(add_ln32_6_fu_1194_p2) + unsigned(ap_const_lv19_AE1));
1342 add_ln32_8_fu_1220_p2 <= std_logic_vector(signed(sext_ln32_13_fu_1217_p1) + signed(sext_ln32_12_fu_1214_p1));
1343 add_ln32_9_fu_1226_p2 <= std_logic_vector(unsigned(add_ln32_8_fu_1220_p2) + unsigned(ap_const_lv19_851));
1344 add_ln32_fu_1029_p2 <= std_logic_vector(signed(sext_ln32_1_fu_1026_p1) + signed(sext_ln32_fu_1023_p1));
1345 add_ln37_1_fu_2046_p2 <= std_logic_vector(signed(sext_ln37_fu_2036_p1) + signed(zext_ln37_fu_2033_p1));
1346 add_ln37_2_fu_2162_p2 <= std_logic_vector(unsigned(zext_ln37_1_fu_2154_p1) + unsigned(sext_ln37_1_fu_2151_p1));
1347 add_ln37_3_fu_2238_p2 <= std_logic_vector(signed(sext_ln37_3_fu_2228_p1) + signed(sext_ln37_2_fu_2224_p1));
1348 add_ln37_4_fu_2305_p2 <= std_logic_vector(unsigned(zext_ln37_2_fu_2296_p1) + unsigned(sext_ln37_4_fu_2292_p1));
1349 add_ln37_5_fu_2402_p2 <= std_logic_vector(unsigned(zext_ln37_3_fu_2394_p1) + unsigned(sext_ln37_5_fu_2391_p1));
1350 add_ln37_6_fu_2478_p2 <= std_logic_vector(signed(sext_ln37_7_fu_2468_p1) + signed(sext_ln37_6_fu_2464_p1));
1351 add_ln37_7_fu_2564_p2 <= std_logic_vector(unsigned(zext_ln37_4_fu_2555_p1) + unsigned(sext_ln37_8_fu_2551_p1));
1352 add_ln40_fu_2619_p2 <= std_logic_vector(signed(sext_ln40_fu_2615_p1) + signed(ap_const_lv19_C000));
1353 and_ln30_1_fu_457_p2 <= (tmp_reg_2697 and or_ln30_1_fu_451_p2);
1354 and_ln30_2_fu_524_p2 <= (xor_ln30_2_fu_519_p2 and or_ln30_3_fu_514_p2);
1355 and_ln30_3_fu_546_p2 <= (tmp_9_reg_2720 and or_ln30_4_fu_540_p2);
1356 and_ln30_4_fu_805_p2 <= (xor_ln30_4_fu_800_p2 and or_ln30_6_fu_795_p2);
1357 and_ln30_5_fu_827_p2 <= (tmp_35_reg_2789 and or_ln30_7_fu_821_p2);
1358 and_ln30_fu_435_p2 <= (xor_ln30_fu_430_p2 and or_ln30_fu_425_p2);
1359 and_ln31_1_fu_619_p2 <= (tmp_12_reg_2743 and or_ln31_1_fu_613_p2);
```

- Also prints out latency of logic and estimated resources.

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
 ffn	10	80.000	1	yes	0	23	1385	2609	0

C++/HDL co-simulation

- Compare between HDL output with golden output.

```
INFO: [Common 17-206] Exiting xsim at Thu Aug 21 13:02:01 2025...  
INFO: [COSIM 212-316] Starting C post checking ...  
GoodINFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
```

- Also shows latency

Performance & Resource Estimates							
⌵ ⌶ 🔍							
MODULES & LOOPS	AVG II	MAX II	MIN II	AVG LATENCY	MAX LATENCY	MIN LATENCY	TOTAL EXECUTION TIME
● ffn	1	1	1	10	10	10	11

Make into IPCore

```
INFO: calling package_hls_ip ip_types=vitis sysgen json_file=/home/hepdream/Work/FPGA_class/HLS_FFN/hls/hls/hls/hls_data.json
INFO: Copied 1 ipmisc file(s) to /home/hepdream/Work/FPGA_class/HLS_FFN/hls/hls/hls/impl/ip/misc
INFO: Copied 10 verilog file(s) to /home/hepdream/Work/FPGA_class/HLS_FFN/hls/hls/hls/impl/ip/hdl/verilog
INFO: Copied 10 vhdl file(s) to /home/hepdream/Work/FPGA_class/HLS_FFN/hls/hls/hls/impl/ip/hdl/vhdl
ipx::create_core: Time (s): cpu = 00:00:08 ; elapsed = 00:00:08 . Memory (MB): peak = 1642.391 ; gain = 39.836 ; free physical
INFO: Import ports from HDL: /home/hepdream/Work/FPGA_class/HLS_FFN/hls/hls/hls/impl/ip/hdl/vhdl/ffn.vhd (ffn)
INFO: Add clock interface ap_clk
INFO: Add reset interface ap_rst
INFO: Add ap_ctrl interface ap_ctrl
INFO: Add data interface x_in
INFO: Add data interface y_in
INFO: Add data interface z_out
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/opt/Xilinx/Vivado/2024.1/data/ip'.
INFO: Calling post_process_vitis to specialize IP
INFO: Calling post_process_sysgen to specialize IP
Generating sysgen info xml from json file
INFO: Created IP /home/hepdream/Work/FPGA_class/HLS_FFN/hls/hls/hls/impl/ip/component.xml
INFO: Created IP archive /home/hepdream/Work/FPGA_class/HLS_FFN/hls/hls/hls/impl/ip/xilinx_com_hls_ffn_1_0.zip
INFO: [Common 17-206] Exiting Vivado at Thu Aug 21 13:04:34 2025...
INFO: [HLS 200-802] Generated output file hls/ffn.zip
```

Make into IPCore

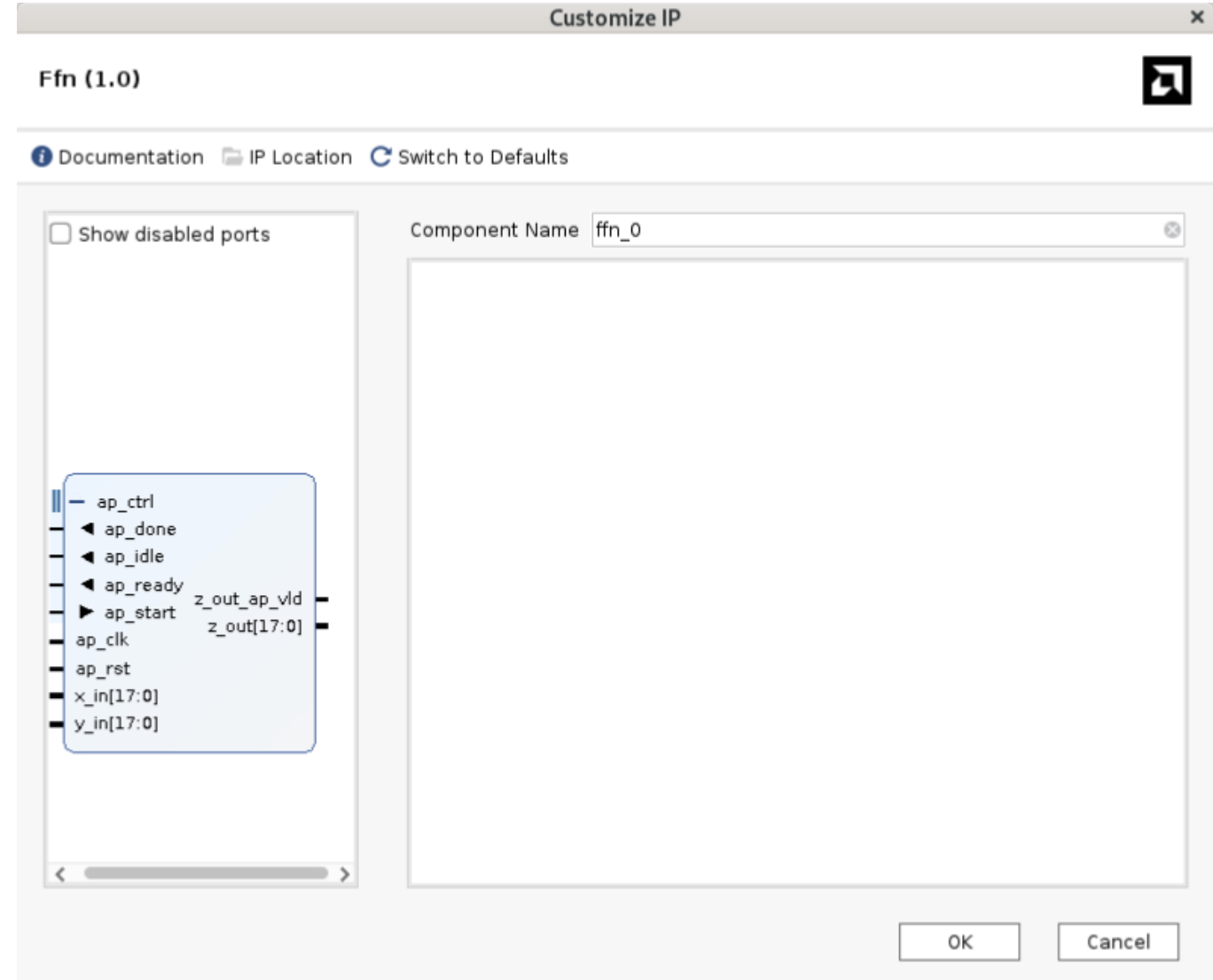
- After adding your HLS

IPCore repository to

Vivado

- You can use your IP in

Vivado!



- How much did you understand? www.kahoot.it