Analysis and Optimization of the Implicit Broadcasts in FPGA HLS to improve Maximum Frequency

Licheng Guo*, Jason Lau*, Yuze Chi, Jie Wang, Cody Hao Yu, Zhe Chen,

Zhiru Zhang and Jason Cong

University of California Los Angeles, Cornell University

* indicates co-first authors

https://github.com/Licheng-Guo/vivado-hls-broadcast-optimization

Outline

- Introduction
- Problem Classification
- Solution
- Experiments

RTL Verilog vs. Untimed C/C++

- Much higher developing efficiency
- Less achievable frequency compared to RTL designs
- Hard to debug the critical path

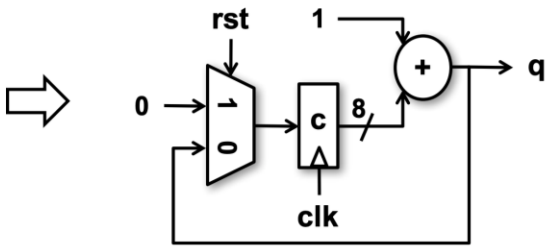
```
module dut(rst, clk, q);
  input rst;
  input clk;
  output q;
  reg [7:0] c;

  always @ (posedge clk)
  begin
    if (rst == 1b'1) begin
      c <= 8'b00000000;
  end
  else begin
    c <= c + 1;
  end

  assign q = c;
endmodule
```

RTL Verilog

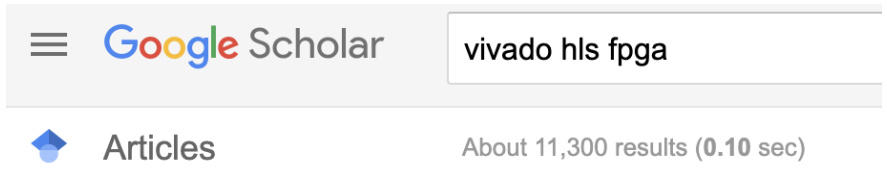
vs.

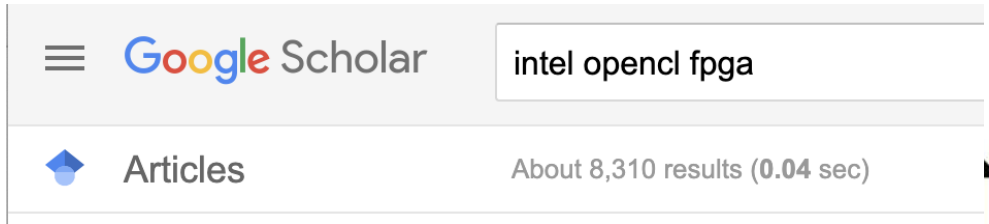
```
uint8 dut() {
  static uint8 c;
  c+=1;
}
```

High-Level Synthesis



An 8-bit counter





We Analyze the Timing Issues of Complex Designs

We Analyze the Timing Issues of Complex Designs

- Most critical paths are related to broadcasts
  - Some are hidden in user codes
  - Some are inferred by the HLS compiler
  - Lead to high-fanout interconnects and bad timing quality

We Analyze the Timing Issues of Complex Designs

- Most critical paths are related to broadcasts
  - Some are hidden in user codes
  - Some are inferred by the HLS compiler
  - Lead to high-fanout interconnects and bad timing quality
- We categorize common types of broadcasts in HLS-based designs.

We Analyze the Timing Issues of Complex Designs

- Most critical paths are related to broadcasts
  - Some are hidden in user codes
  - Some are inferred by the HLS compiler
  - Lead to high-fanout interconnects and bad timing quality
- We categorize common types of broadcasts in HLS-based designs.
- We analyze the inherent limitations of current HLS tools exposed by the broadcast problem

We Analyze the Timing Issues of Complex Designs

- Most critical paths are related to broadcasts
  - Some are hidden in user codes
  - Some are inferred by the HLS compiler
  - Lead to high-fanout interconnects and bad timing quality
- We categorize common types of broadcasts in HLS-based designs.
- We analyze the inherent limitations of current HLS tools exposed by the broadcast problem
- Our lightweight solutions bring significant frequency boost on real-world HLS designs

Outline

- Introduction
- Problem Classification
- Solution
- Experiments

Classification of Broadcasts

- Data Broadcast
  - Originate from the source code
  - High fan-out signals in the datapath
  - Can be mapped back to certain lines in the source code

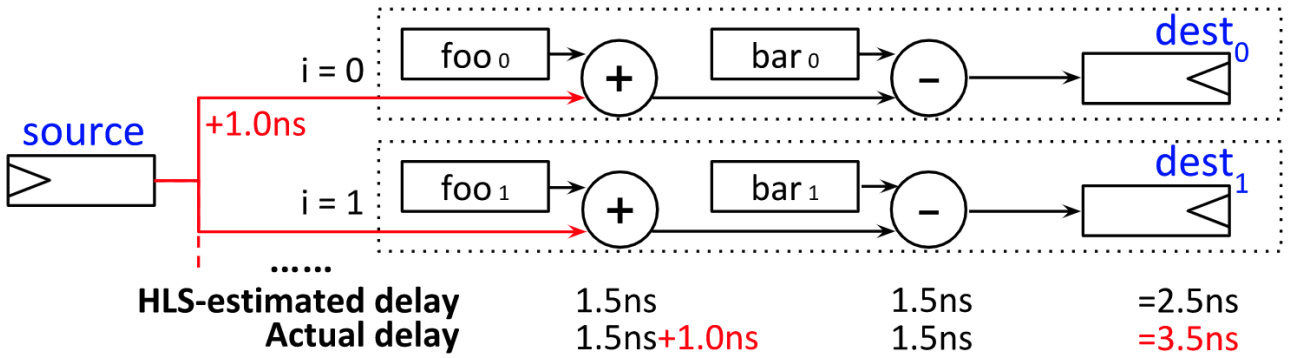
Classification of Broadcasts

- Data Broadcast
  - Originate from the source code
  - High fan-out signals in the datapath
  - Can be mapped back to certain lines in the source code
- Control Broadcast
  - Originate from the compiler
  - High fan-out signals from control logic
  - Completely transparent to users

Data Broadcast

- Scenario 1: unrolled loop

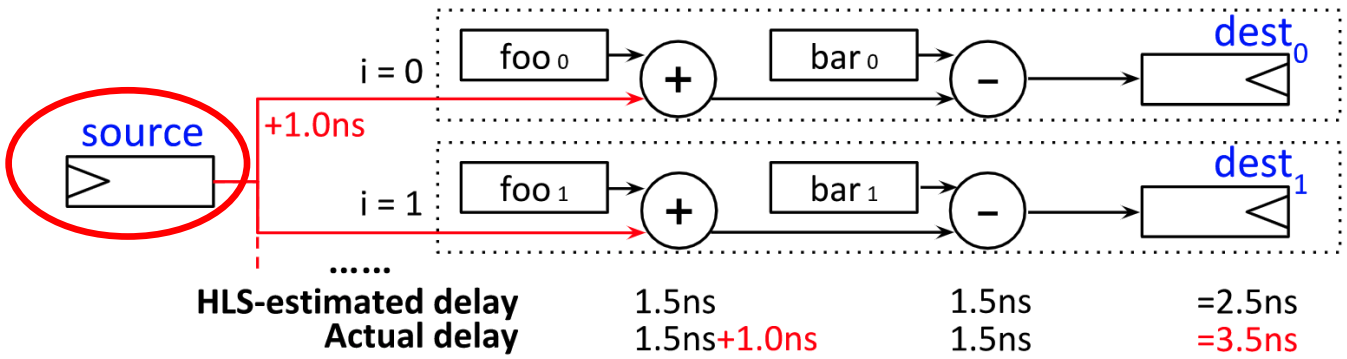
```
1 data_t source = ...;          // loop-invariant variable
2 for (size_t i = 0; i < 1024; i++) {
3 #pragma HLS unroll
4   foo = ...i...; bar = ...i...;  // loop-dependent
5   dest[i] = source + foo - bar; /* ... */ }
```



Data Broadcast

- Scenario 1: unrolled loop

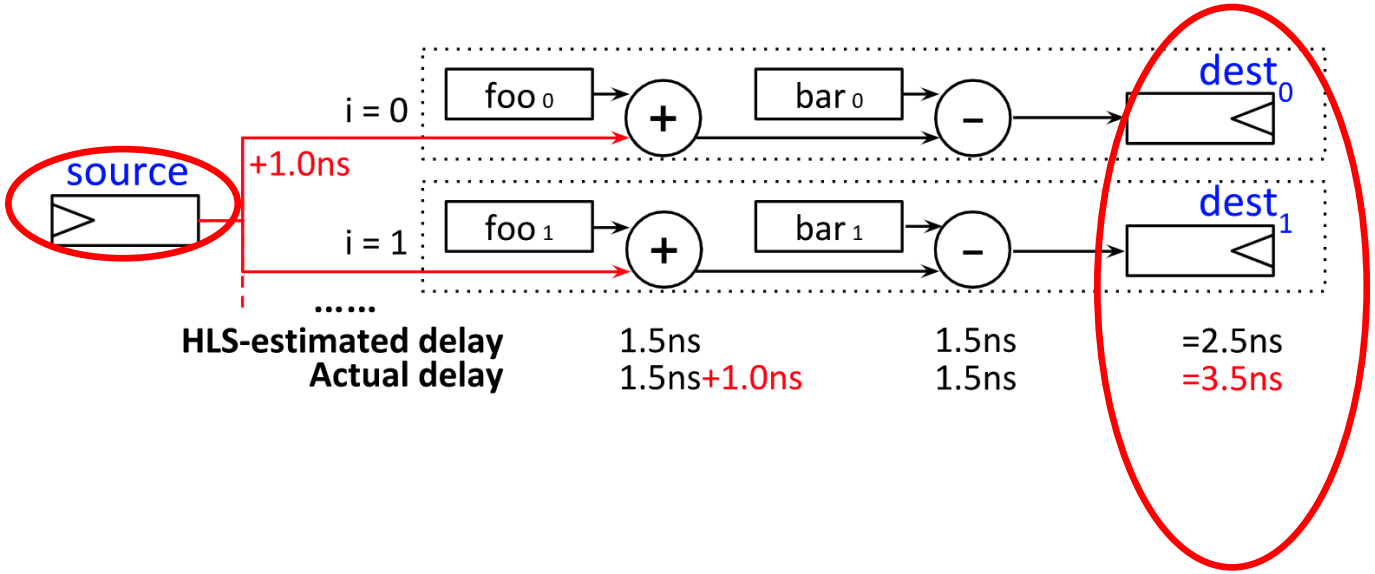
```
1 data_t source = ...;          // loop-invariant variable
2 for (size_t i = 0; i < 1024; i++) {
3 #pragma HLS unroll
4   foo = ...i...; bar = ...i...;  // loop-dependent
5   dest[i] = source + foo - bar; /* ... */ }
```



Data Broadcast

- Scenario 1: unrolled loop

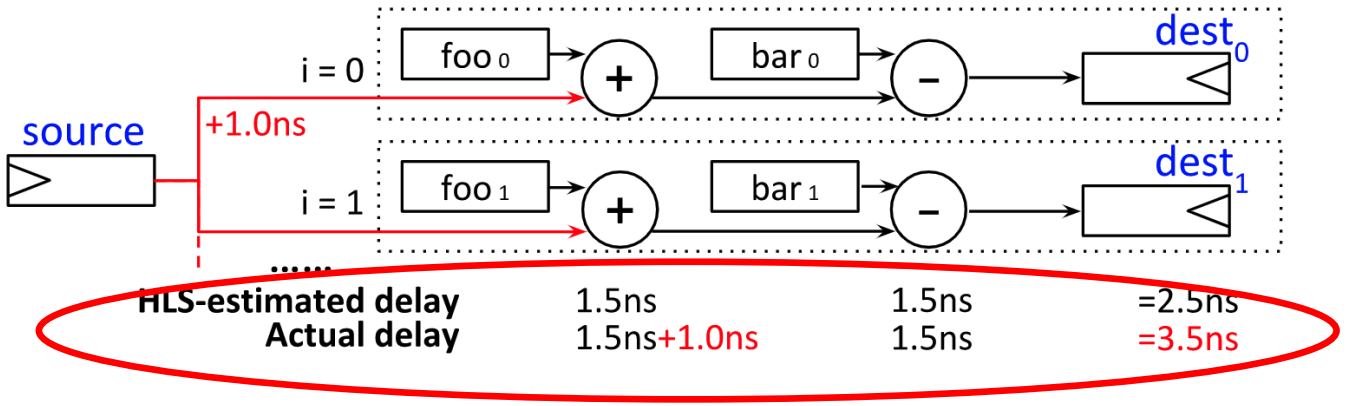
```
1 data_t source = ...;          // loop-invariant variable
2 for (size_t i = 0; i < 1024; i++) {
3 #pragma HLS unroll
4   foo = ...i...; bar = ...i...;  // loop-dependent
5   dest[i] = source + foo - bar; /* ... */ }
```



Data Broadcast

- Scenario 1: unrolled loop

```
1 data_t source = ...;          // loop-invariant variable
2 for (size_t i = 0; i < 1024; i++) {
3 #pragma HLS unroll
4   foo = ...i...; bar = ...i...;  // loop-dependent
5   dest[i] = source + foo - bar; /* ... */ }
```

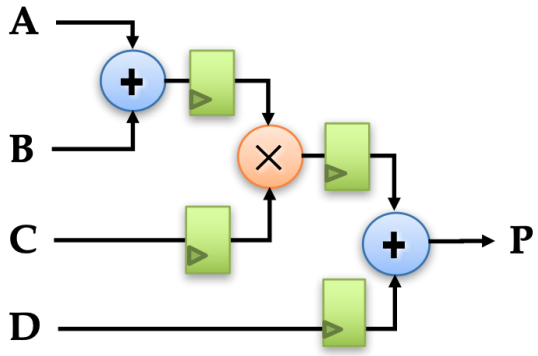


Problem: current HLS delay model does not consider the additional net delay

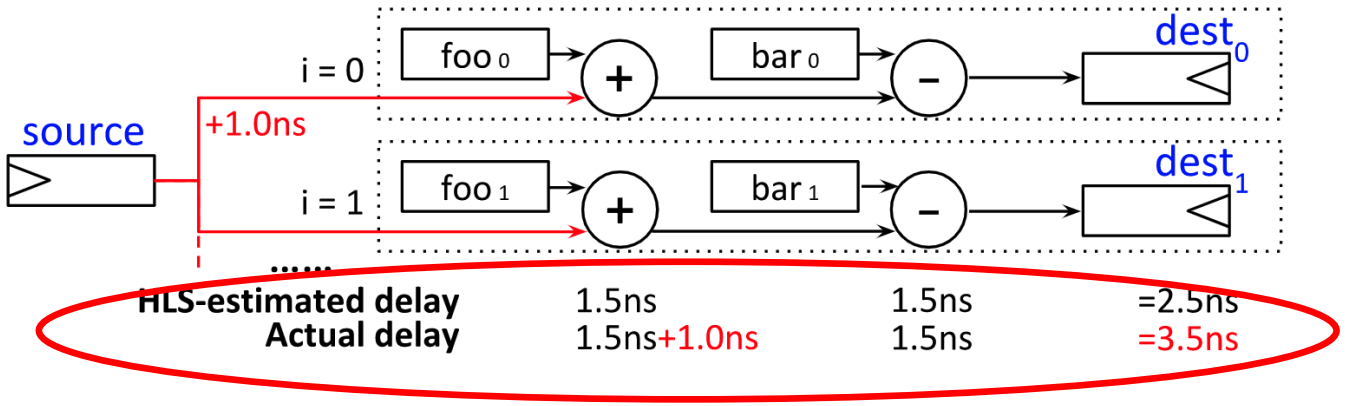
Data Broadcast

- Scenario 1: unrolled loop

```
char A, B, C, D,
int P;

P = (A+B)*C+D
```




```
1 data_t source = ...;          // loop-invariant variable
2 for (size_t i = 0; i < 1024; i++) {
3 #pragma HLS unroll
4   foo = ...i...; bar = ...i...;  // loop-dependent
5   dest[i] = source + foo - bar; /* ... */ }
```

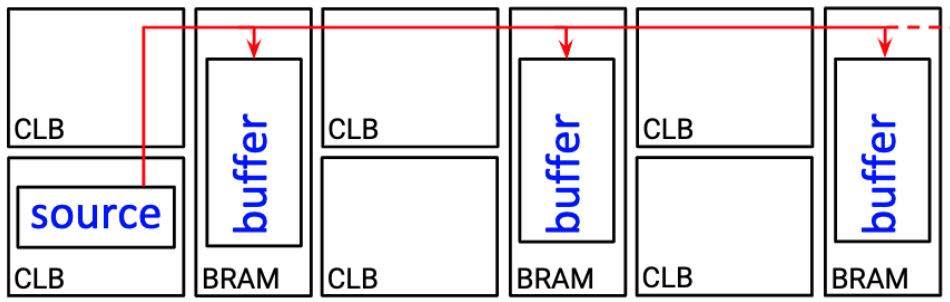


underestimated delay --> inadequate registering

Data Broadcast

- Scenario 2: Large buffer

```
1 data_t buffer[737280]; // mapped to multiple BRAM units
2 buffer[idx] = source; // `source` connects to every BRAM unit
```




```
store
  i32 %source
, i32* %b_addr
```

HLS-estimated	1.2ns
Actual	1.2ns +3.0ns

Control Broadcast

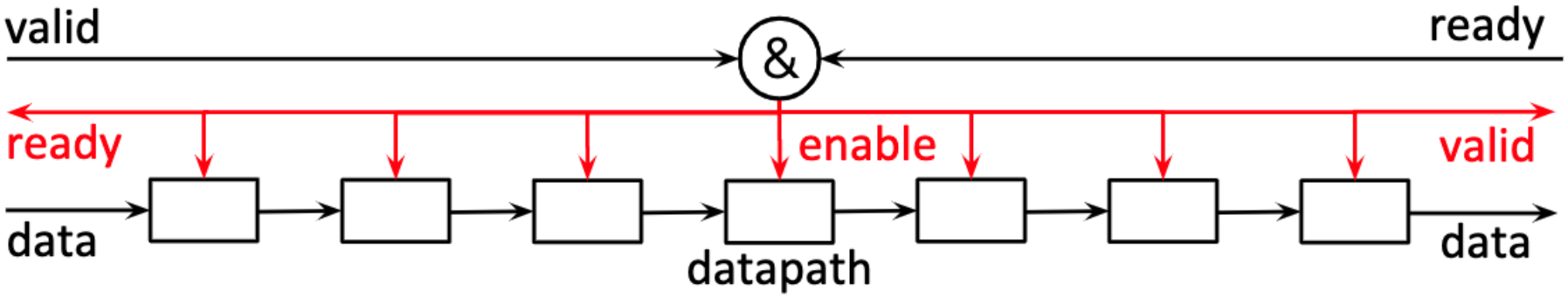
- Scenario 1: Pipeline backpressure

```
1 for (int i = 0; i < ITER; i++) {
2 #pragma HLS pipeline
3   input_fifo.read(&a); /* implicit "empty"-based stall */
4   b = inlined_datapath_foo(a);
5   output_fifo.write(b); /* implicit "full"-based stall */ }
```

Control Broadcast

- Scenario 1: Pipeline backpressure

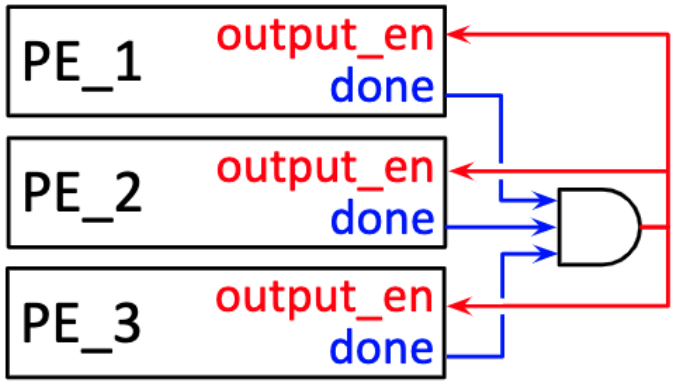
```
1 for (int i = 0; i < ITER; i++) {
2 #pragma HLS pipeline
3  input_fifo.read(&a); /* implicit "empty"-based stall */
4   b = inlined_datapath_foo(a);
5  output_fifo.write(b); /* implicit "full"-based stall */ }
```



Control Broadcast

- Scenario 2: Synchronization of parallel logics
- The compiler infers parallelism from sequential code
- Insert synchronization logic to guarantee correctness

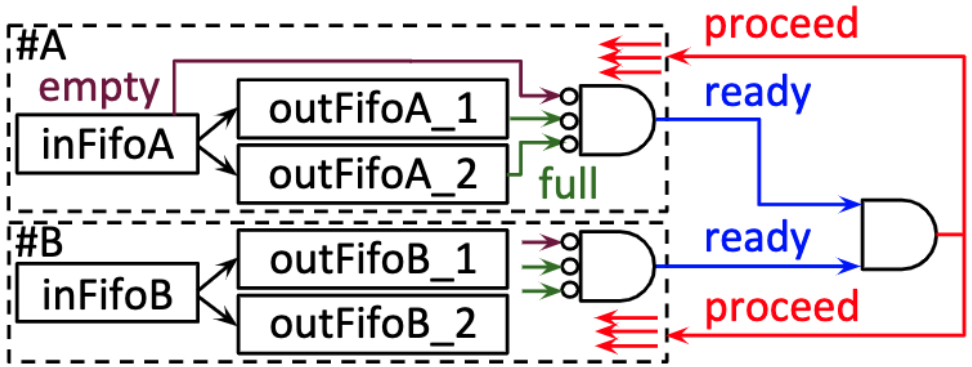
```
1 data_t kernel( ..... ) {
2   /* --- inferred parallelization --- */
3   aOut = PE_1(aIn); bOut = PE_2(bIn); cOut = PE_3(cIn); // ...
4   /* --- inferred synchronization --- */
5   return aOut + bOut + cOut /* ... */; }
```



Control Broadcast

- Scenario 2: Synchronization of parallel logics
- The compiler infers parallelism from sequential code
- Insert synchronization logic to guarantee correctness

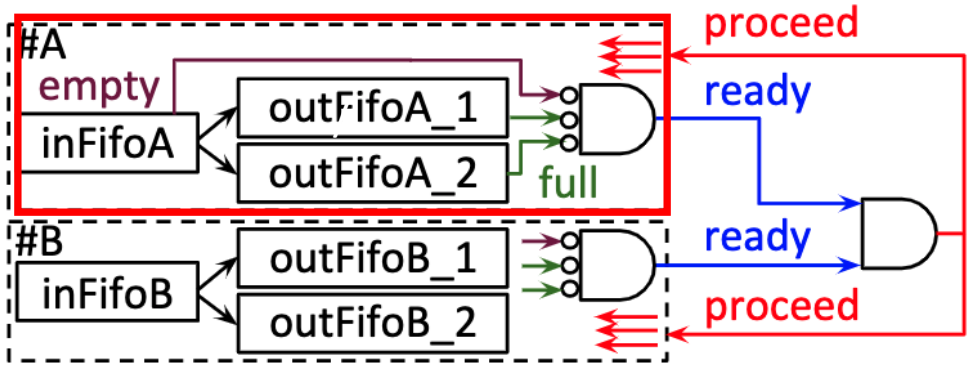
```
1 #pragma HLS dataflow
2 while (1) {
3   /* --- inferred parallelization --- */
4   inFifoA.read(&a);
5     outFifoA1.write(a.foo); outFifoA2.write(a.bar); // #A
6   inFifoB.read(&b);
7     outFifoB1.write(b.foo); outFifoB2.write(b.bar); // #B
8   /* --- HLS infers excessive synchronization --- */ }
```



Control Broadcast

- Scenario 2: Synchronization of parallel logics
- The compiler infers parallelism from sequential code
- Insert synchronization logic to guarantee correctness

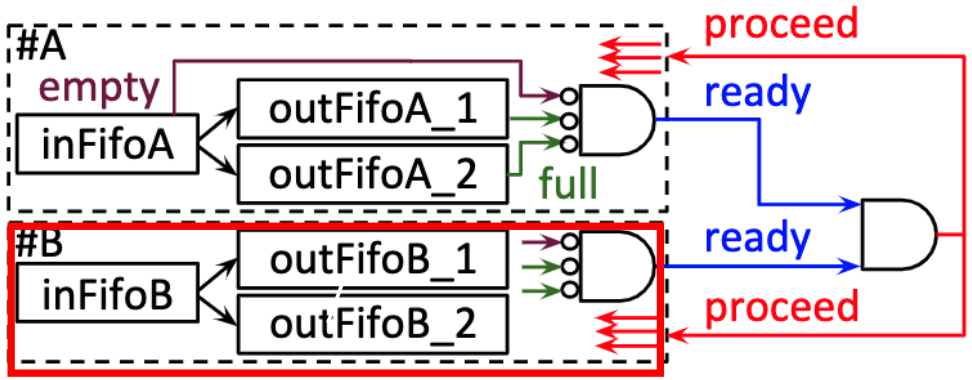
```
1 #pragma HLS dataflow
2 while (1) {
3   /* --- inferred parallelization --- */
4   inFifoA.read(&a);
5     outFifoA1.write(a.foo); outFifoA2.write(a.bar); // #A
6   inFifoB.read(&b);
7     outFifoB1.write(b.foo); outFifoB2.write(b.bar); // #B
8   /* --- HLS infers excessive synchronization --- */ }
```



Control Broadcast

- Scenario 2: Synchronization of parallel logics
- The compiler infers parallelism from sequential code
- Insert synchronization logic to guarantee correctness

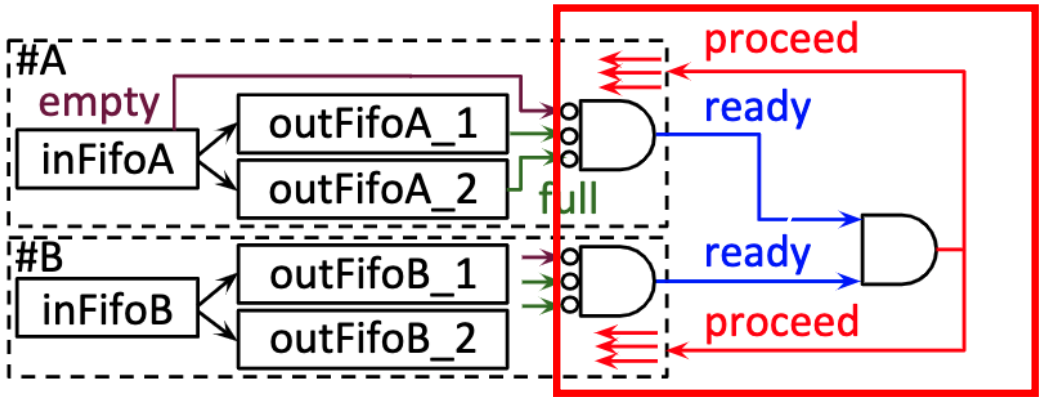
```
1 #pragma HLS dataflow
2 while (1) {
3   /* --- inferred parallelization --- */
4   inFifoA.read(&a);
5     outFifoA1.write(a.foo); outFifoA2.write(a.bar); // #A
6   inFifoB.read(&b);
7     outFifoB1.write(b.foo); outFifoB2.write(b.bar); // #B
8   /* --- HLS infers excessive synchronization --- */ }
```



Control Broadcast

- Scenario 2: Synchronization of parallel logics
- The compiler infers parallelism from sequential code
- Insert synchronization logic to guarantee correctness

```
1 #pragma HLS dataflow
2 while (1) {
3   /* --- inferred parallelization --- */
4   inFifoA.read(&a);
5     outFifoA1.write(a.foo); outFifoA2.write(a.bar); // #A
6   inFifoB.read(&b);
7     outFifoB1.write(b.foo); outFifoB2.write(b.bar); // #B
8   /* --- HLS infers excessive synchronization --- */ }
```



reduce-then-broadcast

Summary of Broadcast Types

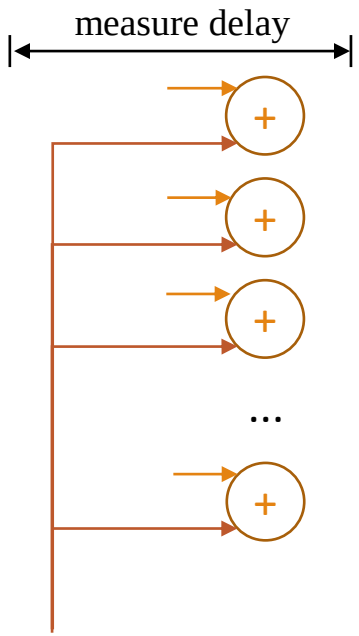
- Data Broadcast
  - Loop unrolling: loop-invariants variables will be broadcast
  - Large buffer: logical buffer entity will become scattered memory units
  - Lead to incorrect delay prediction -> bad clock insertion
- Control Broadcast
  - Pipeline control: backpressure signals are broadcast to the whole datapath
  - Synchronization control: guarantee the correctness of concurrent execution
  - Unscalable broadcast of control signals -> not working for large designs

Outline

- Introduction
- Problem Classification
- Solution
- Experiments

Broadcast-Aware Scheduling

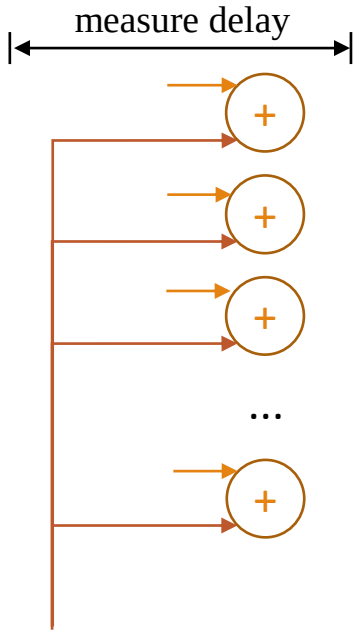
- Isolate the broadcast skeletons and measure the delay



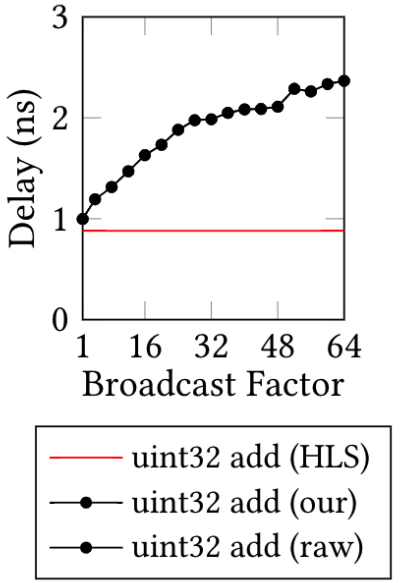
a broadcast skeleton

Broadcast-Aware Scheduling

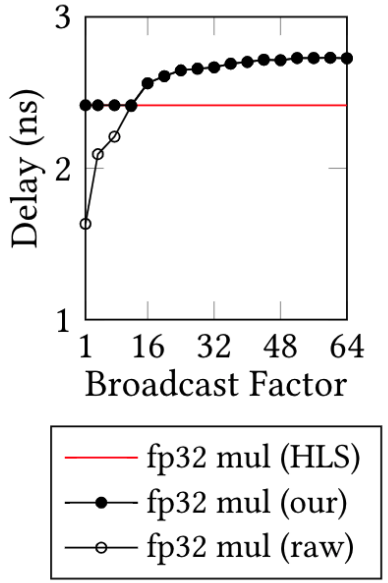
- Isolate the broadcast skeletons and measure the delay
- The additional delay serve as a conservative calibration



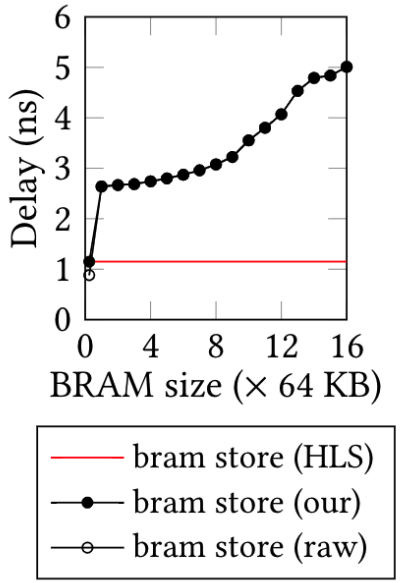
A broadcast skeleton



(a) 32 bits integers adder



(b) 32 bits FP multiplier



(c) BRAM buffer access

Broadcast-Aware Scheduling

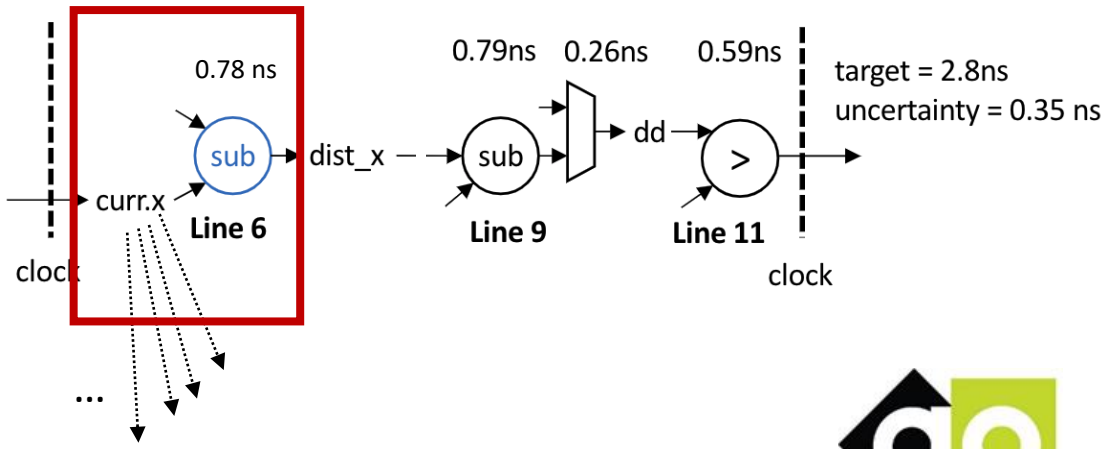
- Example: a genome sequencing accelerator design
- Broadcast elements to 64 datapaths

```
#pragma HLS pipeline
#define UNROLL_FACTOR 64
// ......
for (int j = 0; j < UNROLL_FACTOR; j++) {
#pragma HLS unroll
    dist_x = prev[j].x - curr.x;
    dist_y = prev[j].y - curr.y;

    dd = dist_x > dist_y ? dist_x - dist_y : dist_y - dist_x;
    min_d = dist_y < dist_x ? dist_y : dist_x;
    log_dd = log2(dd); // a series of if-else
    temp = min_d > prev[j].w ? prev[j].w : min_d;

    dp_score[j]= temp - dd * avg_qspan - (log_dd>>1)
    if((dist_x == 0 || dist_x > max_dist_x )||
        (dist_y > max_dist_y || dist_y <= 0) ||
        (dd > bw) || (curr.tag != prev[j].tag) ){
        dp_score[j] = NEG_INF_SCORE;
}   } ......
```

Broadcast-Aware Scheduling

- Example: a genome sequencing accelerator design
- Broadcast elements to 64 datapaths

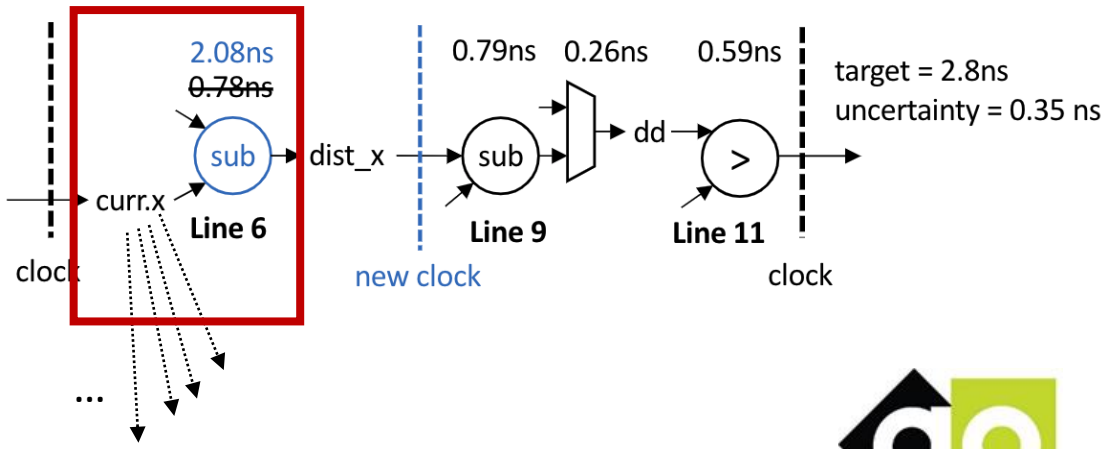
```
1  #pragma HLS pipeline
2  #define UNROLL_FACTOR 64
3  // ......
4  for (int j = 0; j < UNROLL_FACTOR; j++) {
5  #pragma HLS unroll
6      dist_x = prev[j].x - curr.x;
7      dist_y = prev[j].y - curr.y;
8
9      dd = dist_x > dist_y ? dist_x - dist_y : dist_y - dist_x;
10     min_d = dist_y < dist_x ? dist_y : dist_x;
11     log_dd = log2( dd ); // a series of if-else
12     temp = min_d > prev[j].w ? prev[j].w : min_d;
13
14     dp_score[j]= temp - dd * avg_qspan - (log_dd>>1)
15     if((dist_x == 0 || dist_x > max_dist_x )||
16         (dist_y > max_dist_y || dist_y <= 0) ||
17         (dd > bw) || (curr.tag != prev[j].tag) ){
18         dp_score[j] = NEG_INF_SCORE;
19 }   } ......
```



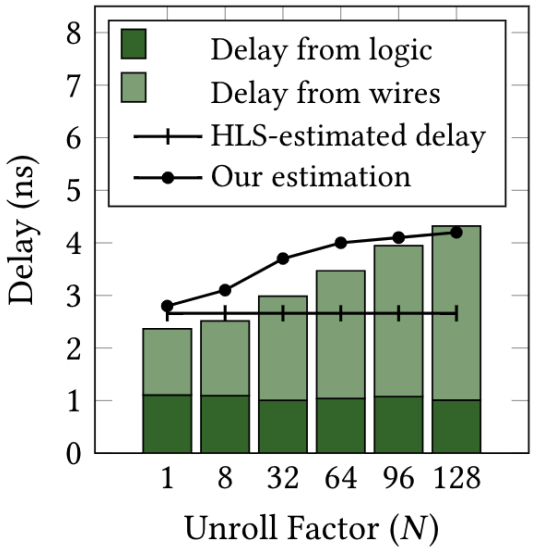
Broadcast-Aware Scheduling

- Example: a genome sequencing accelerator design
- Broadcast elements to 64 datapaths

```
1  #pragma HLS pipeline
2  #define UNROLL_FACTOR 64
3  // ......
4  for (int j = 0; j < UNROLL_FACTOR; j++) {
5  #pragma HLS unroll
6      dist_x = prev[j].x - curr.x;
7      dist_y = prev[j].y - curr.y;
8
9      dd = dist_x > dist_y ? dist_x - dist_y : dist_y - dist_x;
10     min_d = dist_y < dist_x ? dist_y : dist_x;
11     log_dd = log2( dd ); // a series of if-else
12     temp = min_d > prev[j].w ? prev[j].w : min_d;
13
14     dp_score[j]= temp - dd * avg_qspan - (log_dd>>1)
15     if((dist_x == 0 || dist_x > max_dist_x )||
16         (dist_y > max_dist_y || dist_y <= 0) ||
17         (dd > bw) || (curr.tag != prev[j].tag) ){
18         dp_score[j] = NEG_INF_SCORE;
19 }   } ......
```

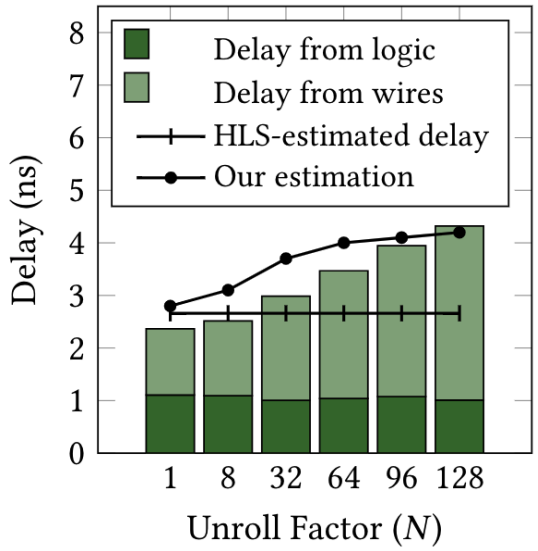


Broadcast-Aware Scheduling

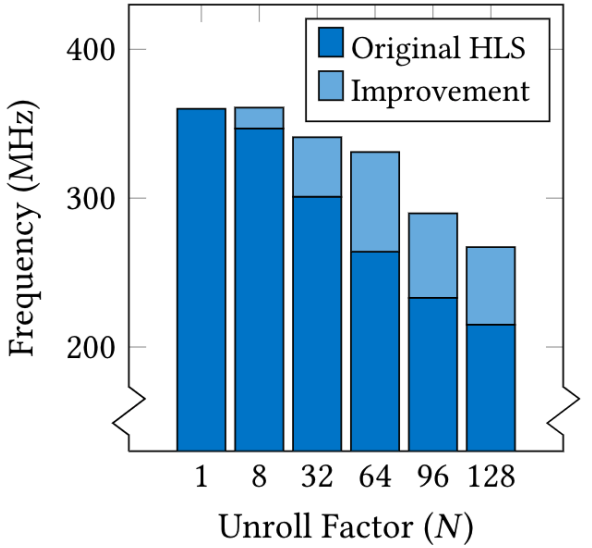


Delay of the aforementioned path

Broadcast-Aware Scheduling



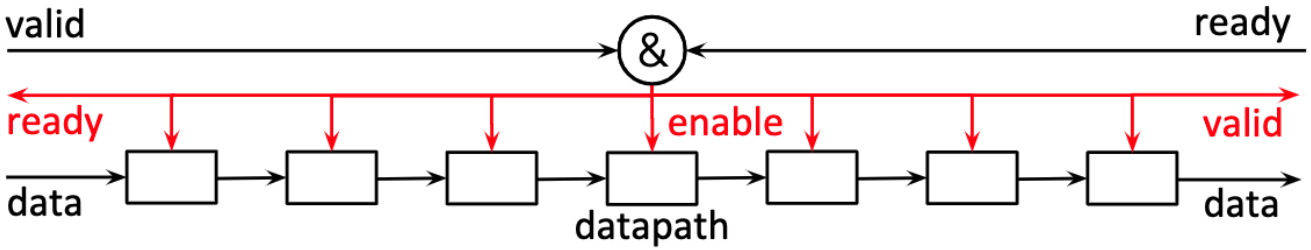
Delay of the aforementioned path

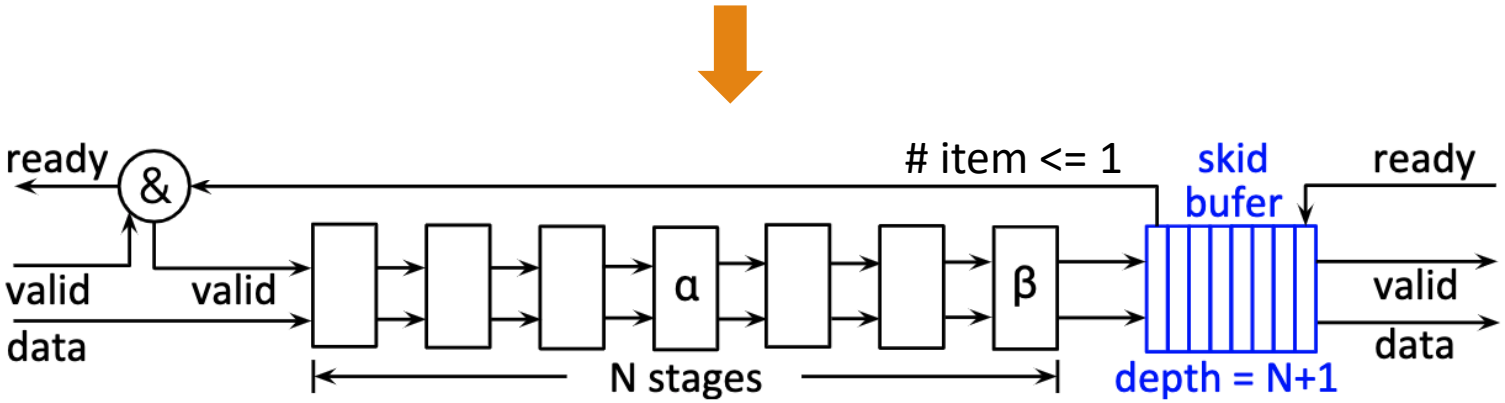


Overrall frequency improvements

Skid-Buffer-Based Pipeline Control

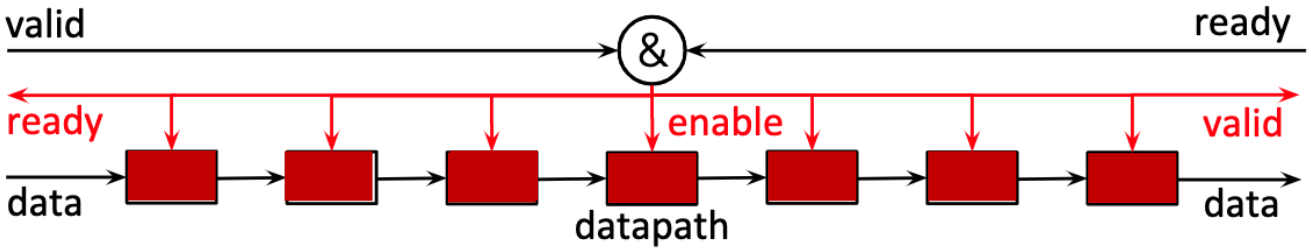
- Adopt skid buffer for flow control

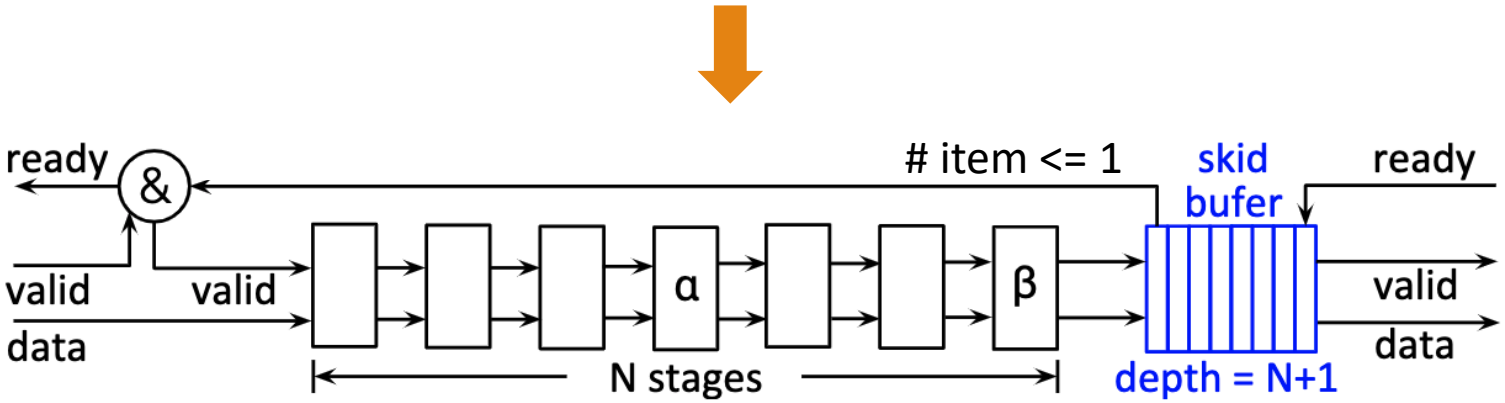




Skid-Buffer-Based Pipeline Control

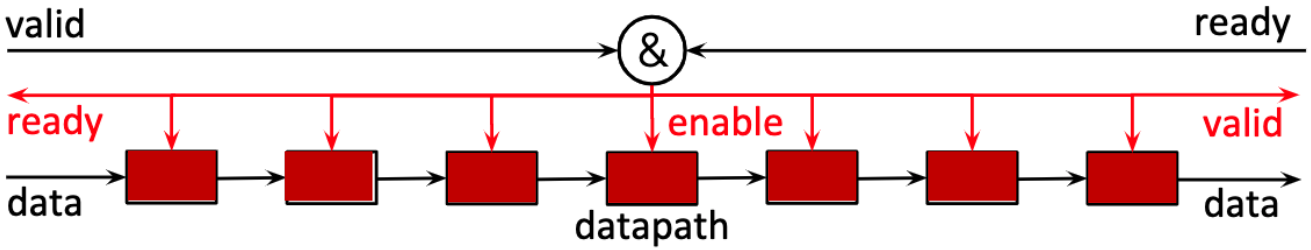
- Adopt skid buffer for flow control

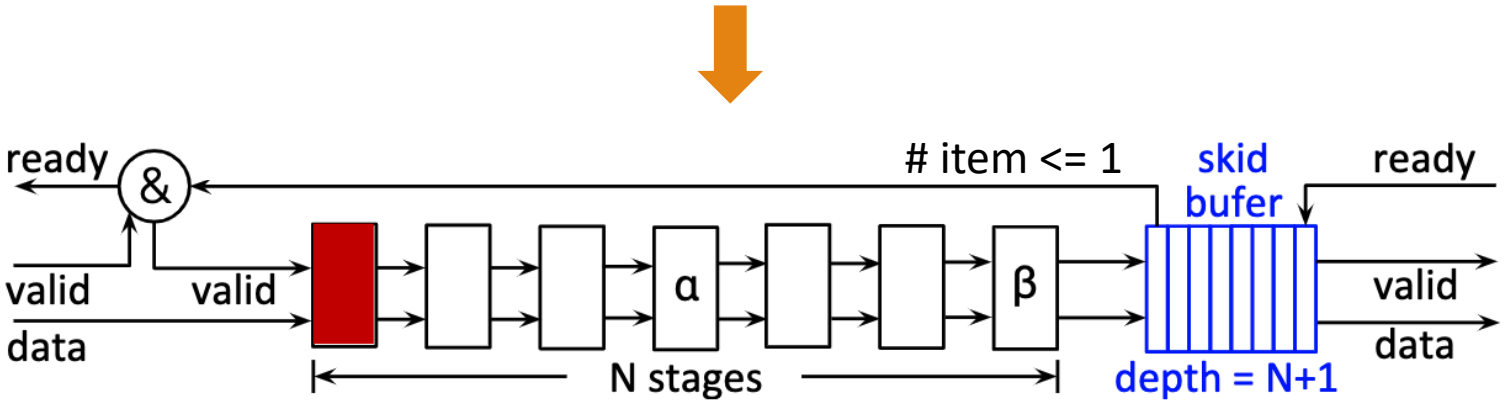




Skid-Buffer-Based Pipeline Control

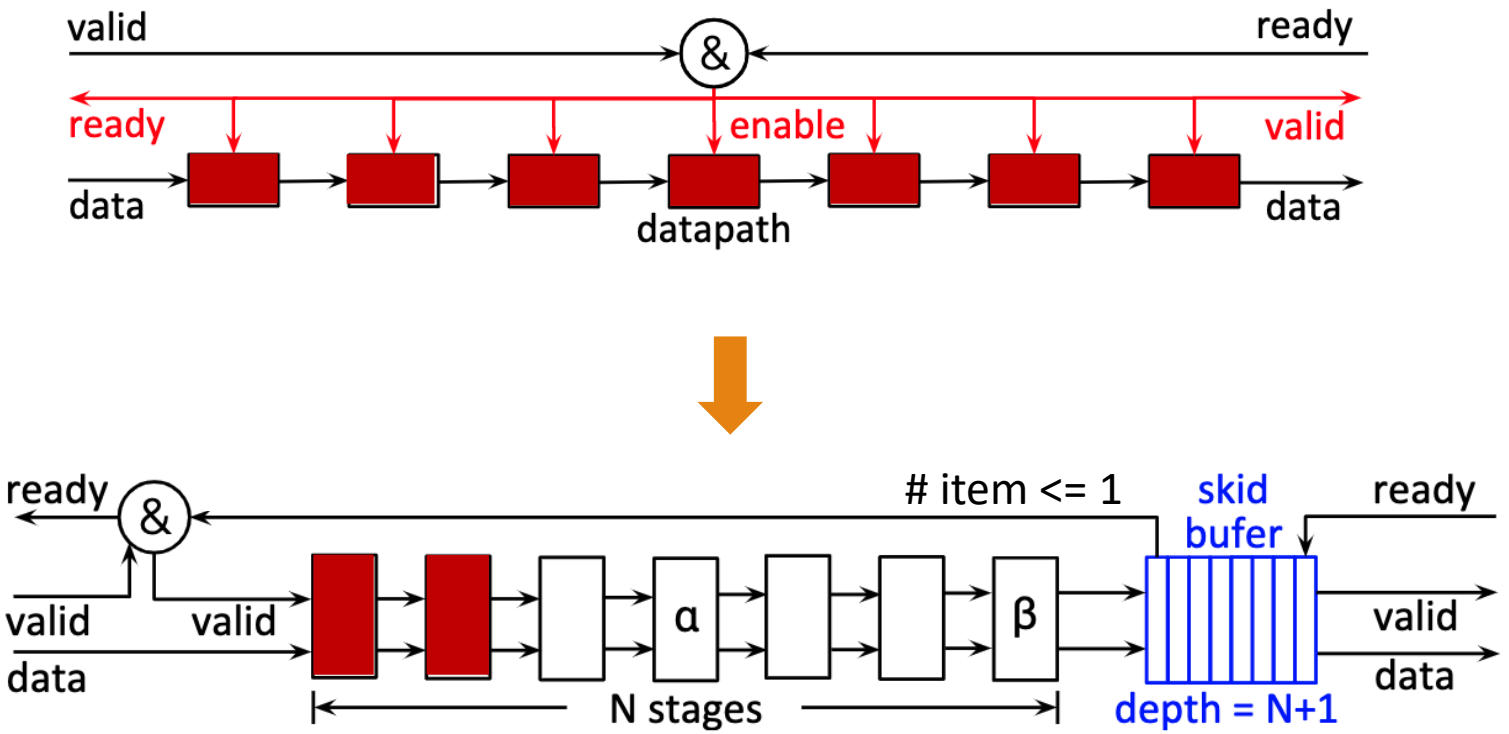
- Adopt skid buffer for flow control





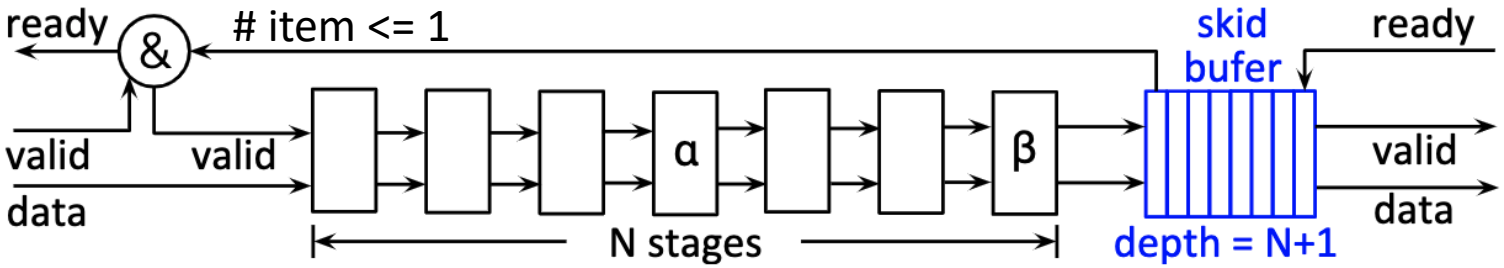
Skid-Buffer-Based Pipeline Control

- Adopt skid buffer for flow control



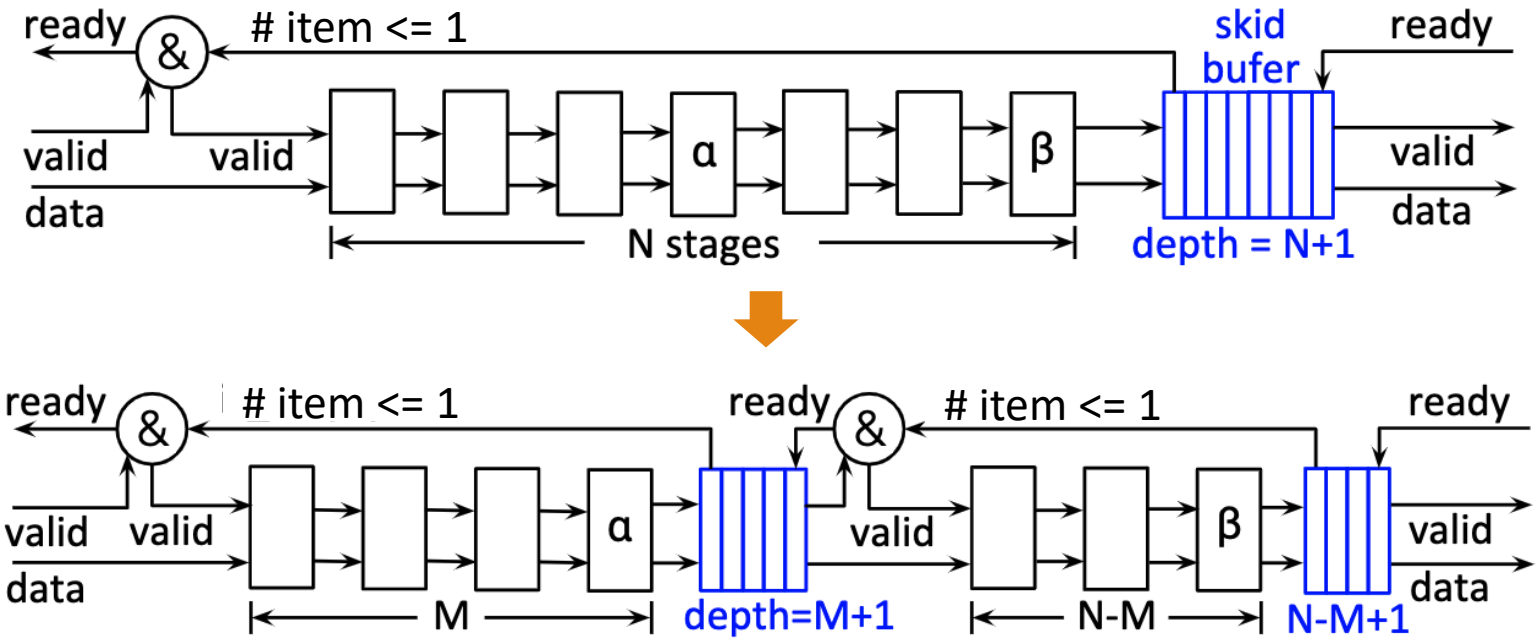
Skid-Buffer-Based Pipeline Control

- Buffer width equals that of the pipeline output
- Different pipeline stages have different output width



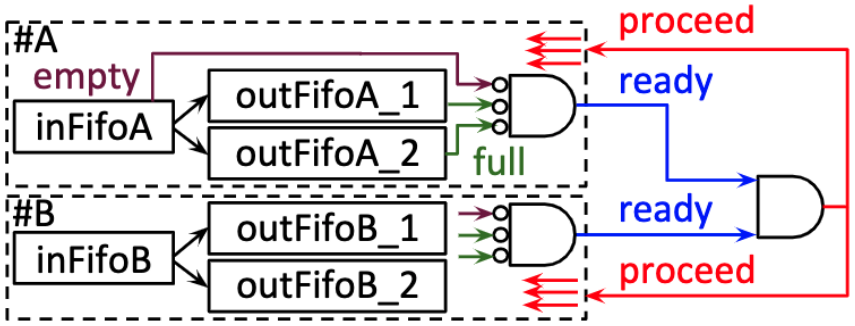
Skid-Buffer-Based Pipeline Control

- Buffer width equals that of the pipeline output
- Different pipeline stages have different output width
- Dynamic programming to optimize the area overhead

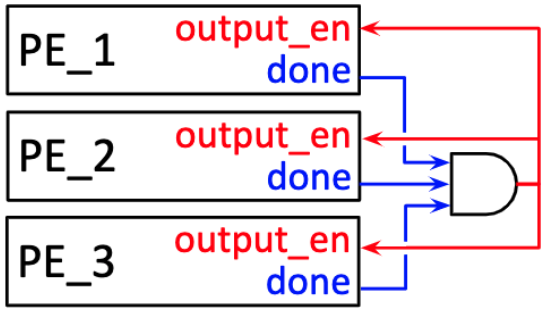


Synchronization Logic Pruning

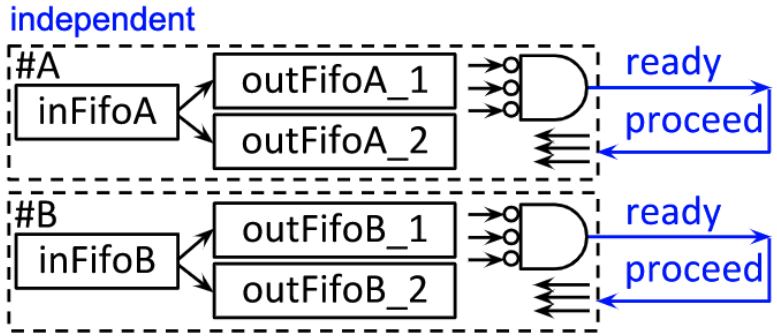
- Prune away redundant synchronization logic



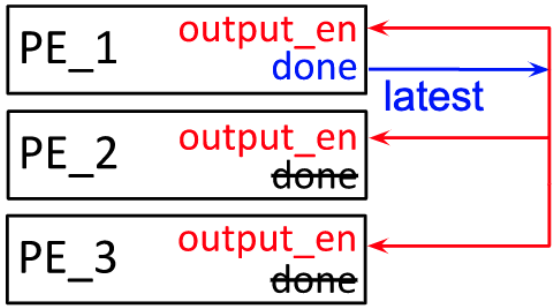
(a) Architecture of Example #1



(b) Architecture of Example #2



(a) Architecture of Example #1



(b) Architecture of Example #2

Experiment Results

- > 50% improvement on our benchmarks
- For more details please check our paper :)
- https://github.com/Licheng-Guo/vivado-hls-broadcast-optimization

Application	Broadcast type	Target FPGA	LUT (%)		FF (%)		BRAM (%)		DSP (%)		Freq (MHz)		
			Orig	Opt	Orig	Opt	Orig	Opt	Orig	Opt	Orig	Opt	Diff
Genome Sequencing [9]	Data	UltraScale+	22	22	11	12	6	6	8	8	264	341	29%
LSTM Network [10]	Data	UltraScale+	8	9	6	6	2	2	14	14	285	325	14%
Face Detection [1]	Data	Kintex-7	21	22	14	15	16	16	9	9	220	273	24%
Matrix Multiply [4]	Pipe. Ctrl. & Data	UltraScale+	23	23	24	27	25	25	74	74	202	299	48%
Stream Buffer	Pipe. Ctrl. & Data	UltraScale+	1	1	1	1	95	95	0	0	154	281	82%
Stencil [3]	Pipe. Ctrl.	UltraScale+	40	40	41	41	30	29	83	83	120	253	111%
Vector Arithmetic	Pipe. Ctrl. & Sync.	UltraScale+	17	17	16	15	0	<1	60	60	195	301	54%
HBM-Based Stencil [3]	Pipe. Ctrl. & Sync.	UltraScale+ HBM	21	23	23	23	34	31	37	37	191	324	70%
Pattern Matching [4]	Data & Sync.	Virtex-7	17	17	5	7	9	9	0	0	187	278	49%

Analysis and Optimization of the Implicit Broadcasts in FPGA HLS to improve Maximum Frequency

- We classify and analyze the common types of broadcasts in HLS
- We propose methods:
  - delay model calibration to optimize the data broadcast
  - min-area skid-buffer to optimize pipeline control
  - synchronization pruning to optimize synchronization broadcast
- We bring over 50% of frequency gain to well-optimized designs.
- https://github.com/Licheng-Guo/vivado-hls-broadcast-optimization

Licheng Guo, Jason Lau*, Yuze Chi, Jie Wang, Cody Hao Yu, Zhe Chen, Zhiru Zhang and Jason Cong*

University of California Los Angeles, Cornell University