Chapter 8: Hierarchical Design

Team 11: 鄭昊昕 游子慶 林稜凱 游翊

Outline

- Arrays Shared Between Blocks

- Blocks with Common Interface Control Variables

- Reconvergence: Balancing the Latency Between Blocks

Outline

- Arrays Shared Between Blocks

- Blocks with Common Interface Control Variables

- Reconvergence: Balancing the Latency Between Blocks

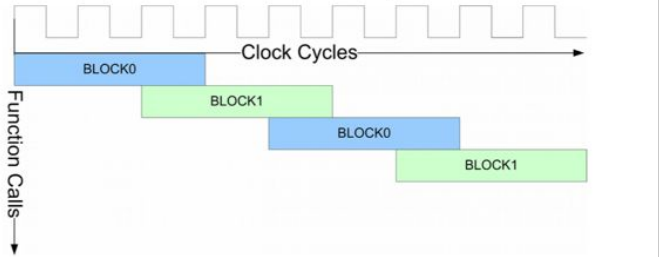
Two Blocks in Cascade but Access in Reverse Order

Example 129 Out of Order Array Access

```
 1 #include "ac_int.h"
 2
 3 void BLOCK0 (ac_int<4,false> din[3],
 4               ac_int<4,false> dout[3]) {
 5
 6   WRITE:for(ac_int<3,false> i=0; i <3; i++) {
 7     dout[i] = din[i]+1;
 8   }
 9 }
10
11 void BLOCK1 (ac_int<4,false> din[3],
12               ac_int<4,false> dout[3]) {
13
14   WRITE:for(ac_int<3,true> i=2; i >= 0; i--) {
15     dout[i] = din[i]-1;
16   }
17 }
18
19 #pragma hls_design top
20 void top (ac_int<4,false> din[3],
21           ac_int<4,false> dout[3]) {
22
23     ac_int<4,false> tmp[3];
24     BLOCK0(din,tmp);
25     BLOCK1(tmp,dout);
26 }
```

← ascending order

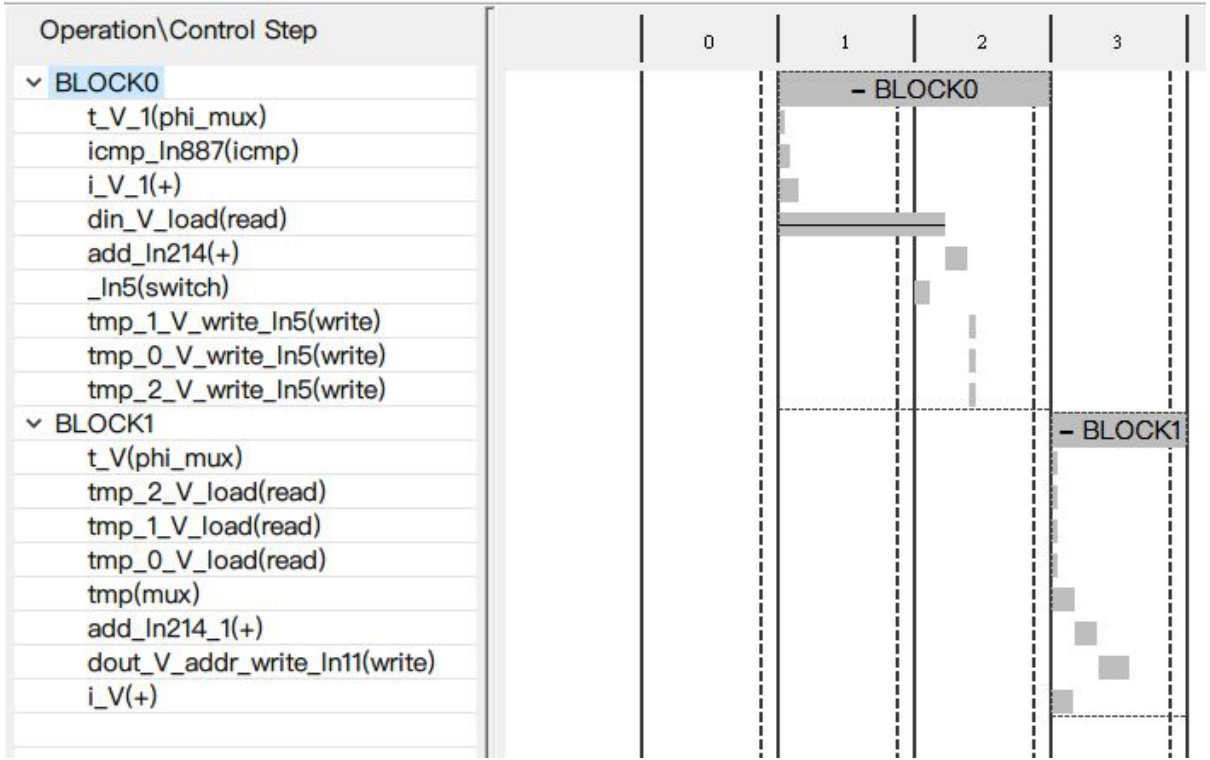
← descending order



Out of Order Array Access

```
void BLOCK0(ap_uint<4> din[3], ap_uint<4> dout[3]) {
    WRITE:for (ap_uint<3> i = 0; i < 3; i++) {
#pragma HLS PIPELINE II = 1
        dout[i] = din[i] + 1;
    }
}

void BLOCK1(ap_uint<4> din[3], ap_uint<4> dout[3]) {
#pragma HLS PIPELINE II = 1
    WRITE:for (ap_int<3> i = 2; i >= 0; i--) {
        dout[i] = din[i] - 1;
    }
}

void top(ap_uint<4> din[3], ap_uint<4> dout[3]) {
    ap_uint<4> tmp[3];
    BLOCK0(din, tmp);
    BLOCK1(tmp, dout);
}
```

Out of Order Array Access - Scheduling



Out of Order Array Access - Testbench

```
#define len 3

int main(void) {
    ap_uint<4> din[3] = {0, 1, 2}, dout[3];
    int pass = 1;

    top(din, dout);

    cout << ">> Start Checking" << endl;
    cout << "------------------------" << endl;
    for (int i = 0; i < len; i++) {
        cout << "Output: " << dout[i] << "\tExpect: " << i << endl;
        if (dout[i] != i) pass = 0;
    }

    cout << "------------------------" << endl;
    if (!pass) {
        cout << "Failed" << endl;
        return 1;
    }
    cout << "Passed Congrats" << endl;
    return 0;
}
```

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	5.369 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
7	7	70.000 ns	70.000 ns	4	4	dataflow

Detail

- Instance
- Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	0	-	5	17	-
Instance	-	-	33	224	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	0	0	38	243	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	~0	~0	0

Detail

- Instance
- DSP48E
- Memory
- FIFO
- Expression
- Multiplexer
- Register

Interface

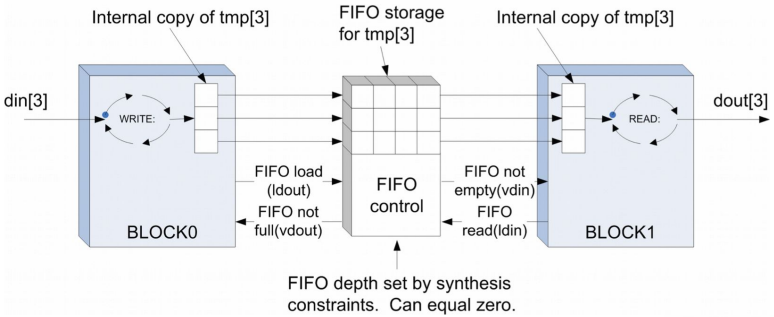
Summary

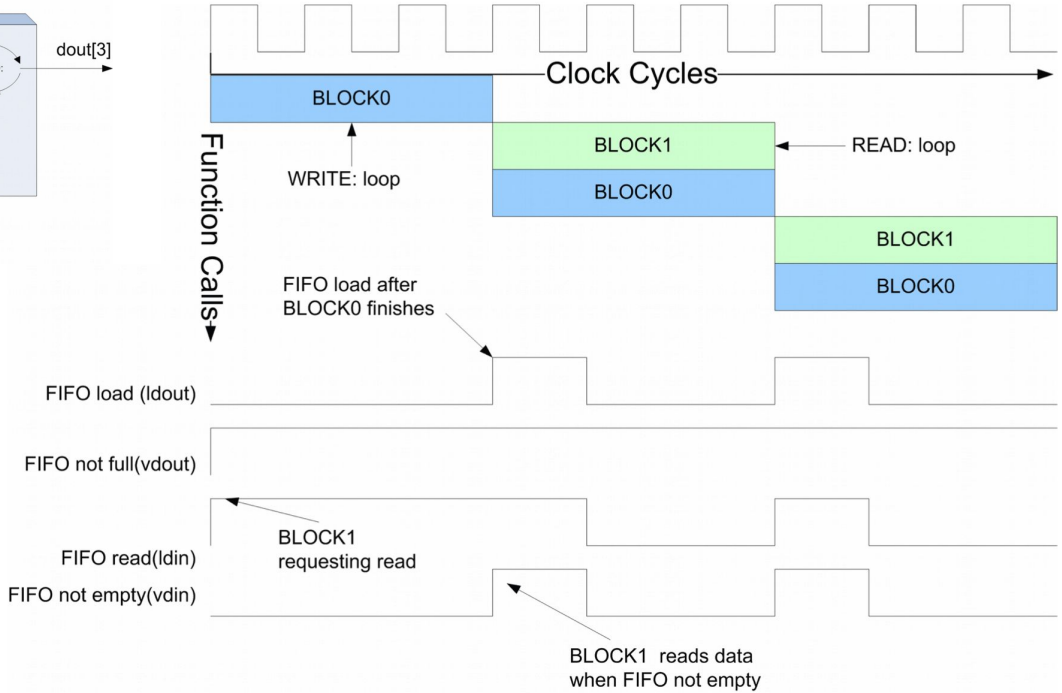
RTL Ports	Dir	Bits	Protocol	Source Object	C Type
in_V_V_dout	in	4	ap_fifo	in_V_V	pointer
in_V_V_empty_n	in	1	ap_fifo	in_V_V	pointer
in_V_V_read	out	1	ap_fifo	in_V_V	pointer
out_V_V_din	out	5	ap_fifo	out_V_V	pointer
out_V_V_full_n	in	1	ap_fifo	out_V_V	pointer
out_V_V_write	out	1	ap_fifo	out_V_V	pointer
ap_clk	in	1	ap_ctrl_hs	top	return value
ap_rst	in	1	ap_ctrl_hs	top	return value
ap_start	in	1	ap_ctrl_hs	top	return value
ap_done	out	1	ap_ctrl_hs	top	return value
ap_ready	out	1	ap_ctrl_hs	top	return value
ap_idle	out	1	ap_ctrl_hs	top	return value

Arrays Mapped to Registers

```
class simple_chanstruct {
    hls::stream<chanStruct> tmp;

    void BLOCK0(hls::stream<uint4>& din, hls::stream<chanStruct>& dout) {
        chanStruct reg;
#pragma HLS ARRAY_PARTITION variable=reg complete dim=1
        WRITE:for (ap_uint<3> i = 0; i < 3; i++) {
#pragma HLS PIPELINE II = 1
            reg.data[i] = din.read();
        }
        dout.write(reg);
    }

    void BLOCK1(hls::stream<chanStruct>& din, hls::stream<uint4>& dout) {
        chanStruct reg;
#pragma HLS ARRAY_PARTITION variable=reg complete dim=1
        reg = din.read();
        READ:for (ap_int<4> i = 2; i >= 0; i--) {
#pragma HLS PIPELINE II = 1
            dout.write(reg.data[i]);
        }
    }

   public:
    simple_chanstruct() {}

    void run(hls::stream<uint4>& din, hls::stream<uint4>& dout) {
#pragma HLS ARRAY_PARTITION variable=tmp complete dim=1
#pragma HLS STREAM variable=tmp depth=3
        BLOCK0(din, tmp);
        BLOCK1(tmp, dout);
    }
};
```

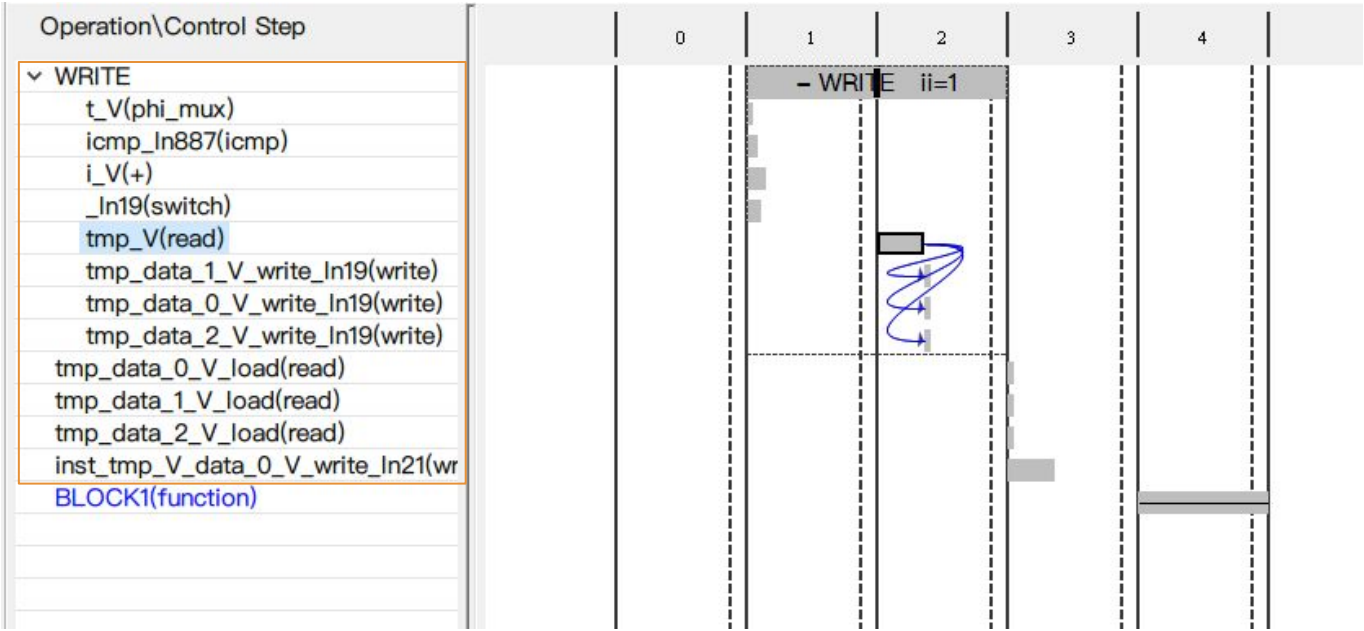
Arrays Mapped to Registers - Block Diagram & Scheduling





Arrays Mapped to Registers - Scheduling

Block 0



Arrays Mapped to Registers - Testbench

```cpp
#define len 9

using namespace std;

int main(void) {
    hls::stream<uint4> din, dout;
    uint4 output;
    int pass = 1;

    for (int i = 0; i < len; i++) {
        din.write(i);
    }

    while (!din.empty()) {
        top(din, dout);
    }

    cout << ">> Start Checking" << endl;
    cout << "-------------------------" << endl;
    for (int i = 0; i < len; i++) {
        output = dout.read();
        uint4 expect = (i / 3) * 3 + 2 - i % 3;
        cout << "Output: " << output << "\tExpect: " << expect << endl;
        if (output != expect) pass = 0;
    }
    cout << "-------------------------" << endl;

    if (!pass) {
        cout << "Failed" << endl;
        return 1;
    }

    cout << "Passed Congrats" << endl;
    return 0;
}
```

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	4.375 ns	0.62 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
5	525	25.000 ns	25.000 ns	5	5	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	0	-	15	48	-
Instance	-	-	22	145	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	128	-
Register	-	-	7	-	-
Total	0	0	44	323	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	~0	~0	0

Interface

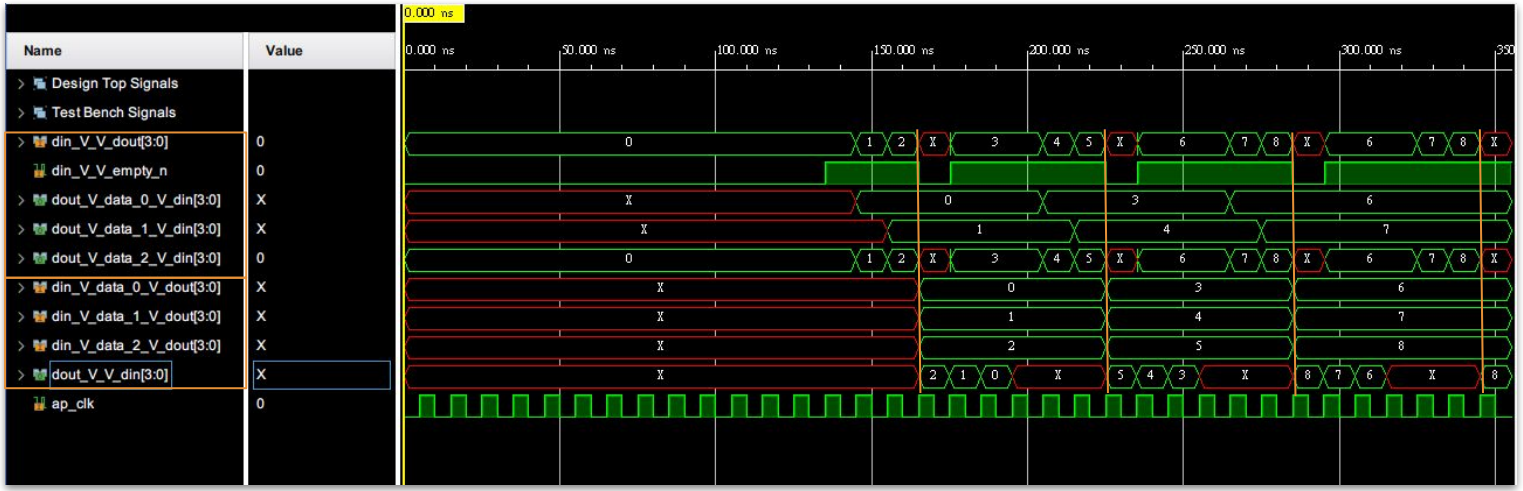
Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	top	return value
ap_rst	in	1	ap_ctrl_hs	top	return value
ap_start	in	1	ap_ctrl_hs	top	return value
ap_done	out	1	ap_ctrl_hs	top	return value
ap_idle	out	1	ap_ctrl_hs	top	return value
ap_ready	out	1	ap_ctrl_hs	top	return value
din_V_V_dout	in	4	ap_fifo	din_V_V	pointer
din_V_V_empty_n	in	1	ap_fifo	din_V_V	pointer
din_V_V_read	out	1	ap_fifo	din_V_V	pointer
dout_V_V_din	out	4	ap_fifo	dout_V_V	pointer
dout_V_V_full_n	in	1	ap_fifo	dout_V_V	pointer
dout_V_V_write	out	1	ap_fifo	dout_V_V	pointer

Waveform

Stream

Struct



Outline

- Arrays Shared Between Blocks

- Blocks with Common Interface Control Variables

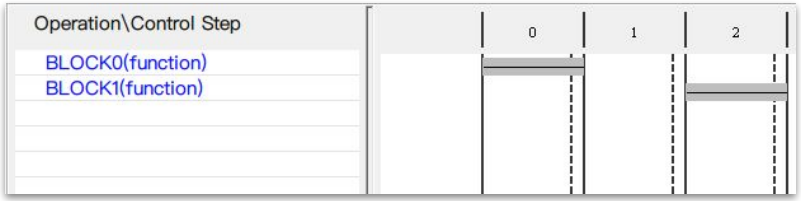
- Reconvergence: Balancing the Latency Between Blocks

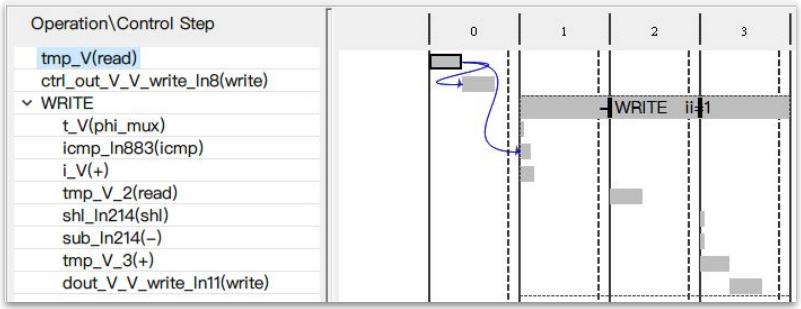
Passing Control Variables Between Blocks

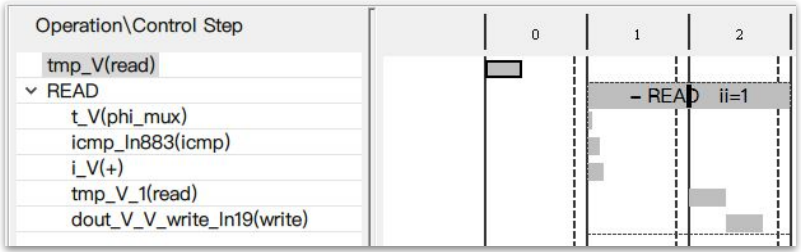
```
3  void BLOCK0(hls::stream<uint4>& din,
4              hls::stream<uint4>& dout,
5              hls::stream<ctrl_type>& ctrl_in,
6              hls::stream<ctrl_type>& ctrl_out) {
7  #pragma HLS PIPELINE II = 1
8      ctrl_type temp_int = ctrl_in.read();
9      ctrl_out.write(temp_int); // one write
10     WRITE:for (ctrl_type i = 0; i != temp_int; i++) {
11         dout.write(din.read() * 13);
12     }
13 }
14
15 void BLOCK1(hls::stream<uint4>& din, hls::stream<uint4>& dout, hls::stream<ctrl_type>&
       ctrl) {
16 #pragma HLS PIPELINE II = 1
17     ctrl_type temp_int = ctrl.read(); // one read
18     READ:for (ctrl_type i = 0; i != temp_int; i++) {
19         dout.write(din.read());
20     }
21 }
22
23 void top(hls::stream<uint4>& din, hls::stream<uint4>& dout, hls::stream<ctrl_type>& ctrl) {
24 #pragma HLS dataflow
25     hls::stream<uint4> data_int;
26     hls::stream<ctrl_type> ctrl_int;
27
28     BLOCK0(din, data_int, ctrl, ctrl_int);
29     BLOCK1(data_int, dout, ctrl_int);
30 }
```

temporary variable to avoid generating multiple read strobes

Scheduling

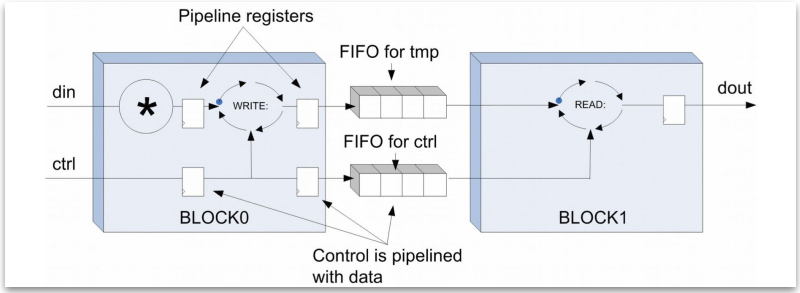






The scheduling seems bizarre.
We tried two designs:

1. With dataflow pragma in top function
2. Without dataflow pragma: No weird stall, but the FIFO has to be as large as the input size, since in the setting second block has to wait until first block finishes.



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.268 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
?	?	?	?	?	?	dataflow

Detail

- Instance
- Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	–	–	–	–	–
Expression	–	–	0	2	–
FIFO	0	–	10	32	–
Instance	–	–	32	210	–
Memory	–	–	–	–	–
Multiplexer	–	–	–	–	–
Register	–	–	–	–	–
Total	0	0	42	244	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	~0	~0	0

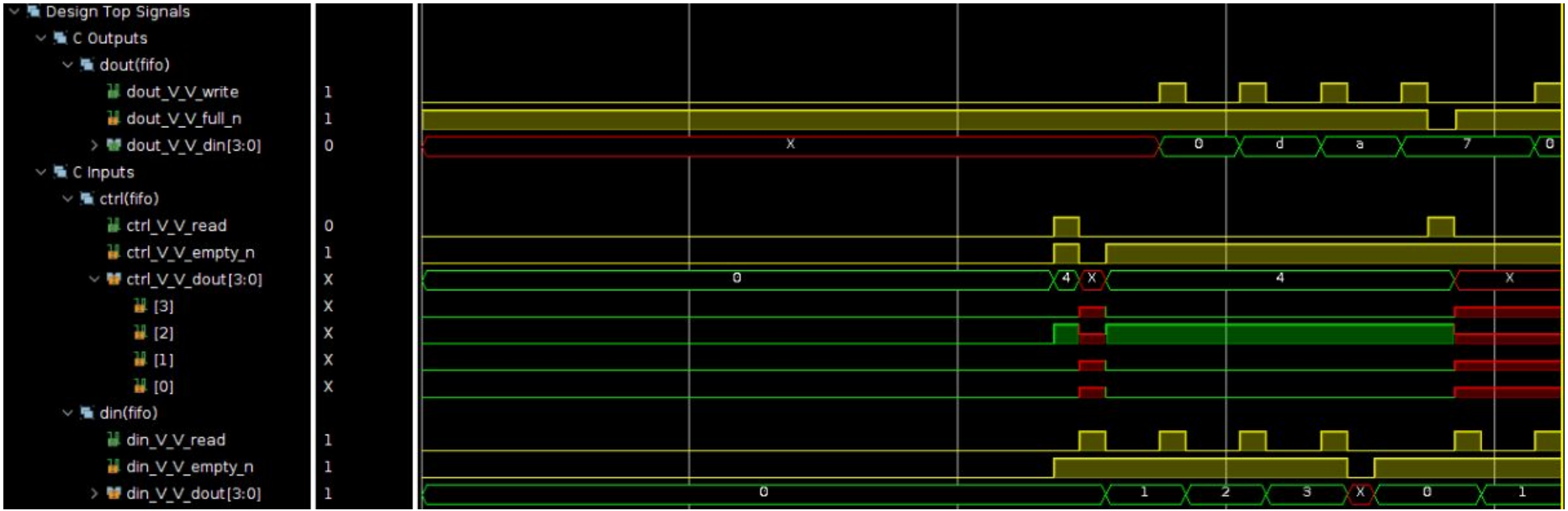
Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
din_V_V_dout	in	4	ap_fifo	din_V_V	pointer
din_V_V_empty_n	in	1	ap_fifo	din_V_V	pointer
din_V_V_read	out	1	ap_fifo	din_V_V	pointer
dout_V_V_din	out	4	ap_fifo	dout_V_V	pointer
dout_V_V_full_n	in	1	ap_fifo	dout_V_V	pointer
dout_V_V_write	out	1	ap_fifo	dout_V_V	pointer
ctrl_V_V_dout	in	4	ap_fifo	ctrl_V_V	pointer
ctrl_V_V_empty_n	in	1	ap_fifo	ctrl_V_V	pointer
ctrl_V_V_read	out	1	ap_fifo	ctrl_V_V	pointer
ap_clk	in	1	ap_ctrl_hs	top	return value
ap_rst	in	1	ap_ctrl_hs	top	return value
ap_start	in	1	ap_ctrl_hs	top	return value
ap_done	out	1	ap_ctrl_hs	top	return value
ap_ready	out	1	ap_ctrl_hs	top	return value
ap_idle	out	1	ap_ctrl_hs	top	return value

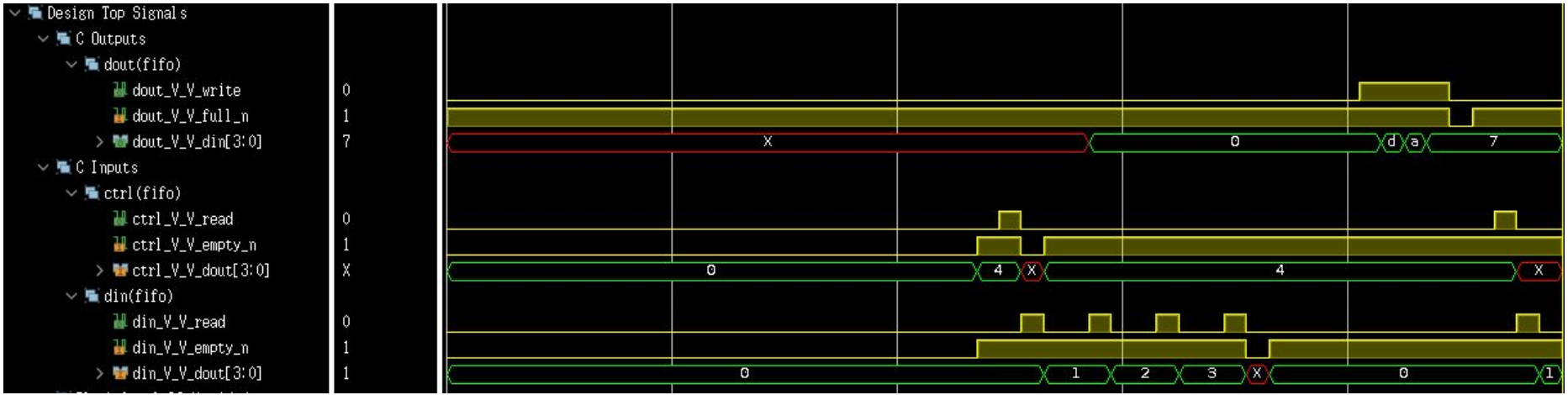
Waveform

dataflow

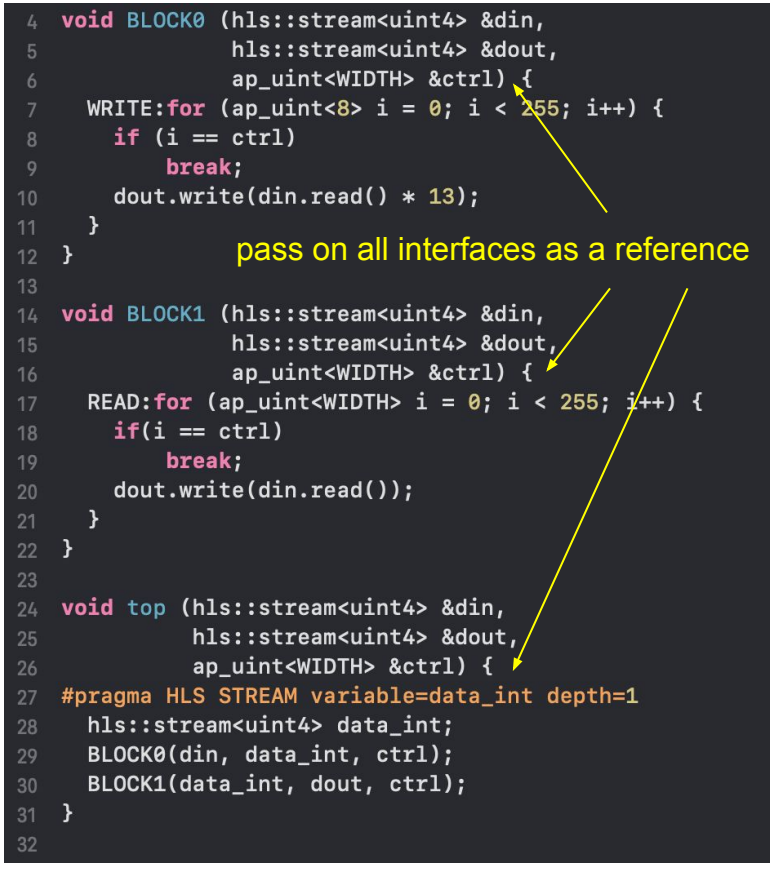


Waveform

no dataflow

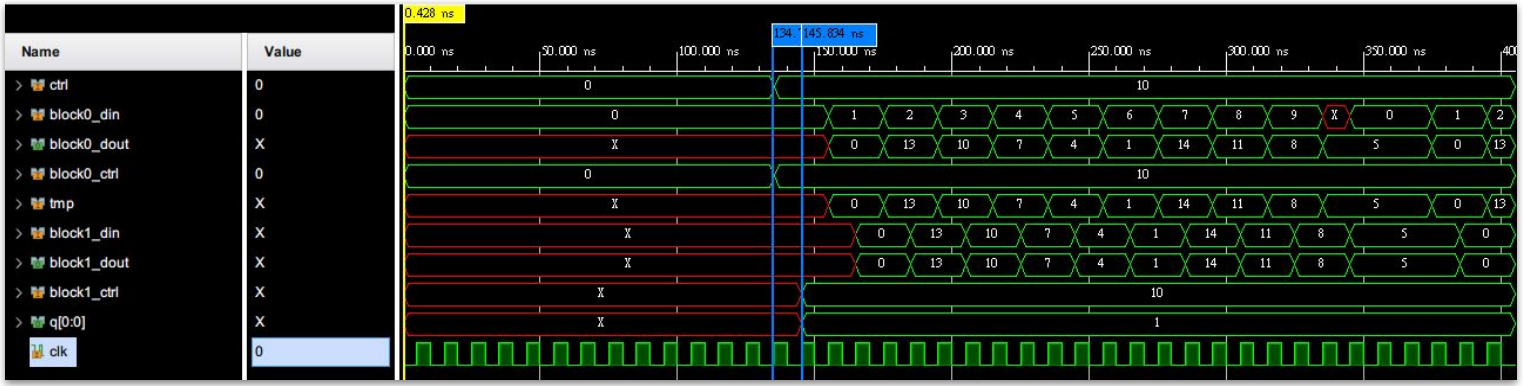


Connecting Interface Control Variables to Multiple Blocks

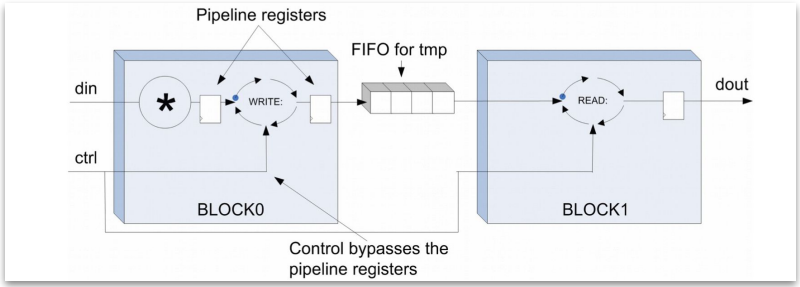


(to avoid creating hierarchy)

Waveform



There seems to be an additional module synthesized to control the timing of block 1.



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.268 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
5	767	50.000 ns	7.670 us	5	767	none

Detail

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	96	-
FIFO	0	-	5	16	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	63	-
Register	-	-	32	-	-
Total	0	0	37	175	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	~0	~0	0

Detail

Interface

Summary

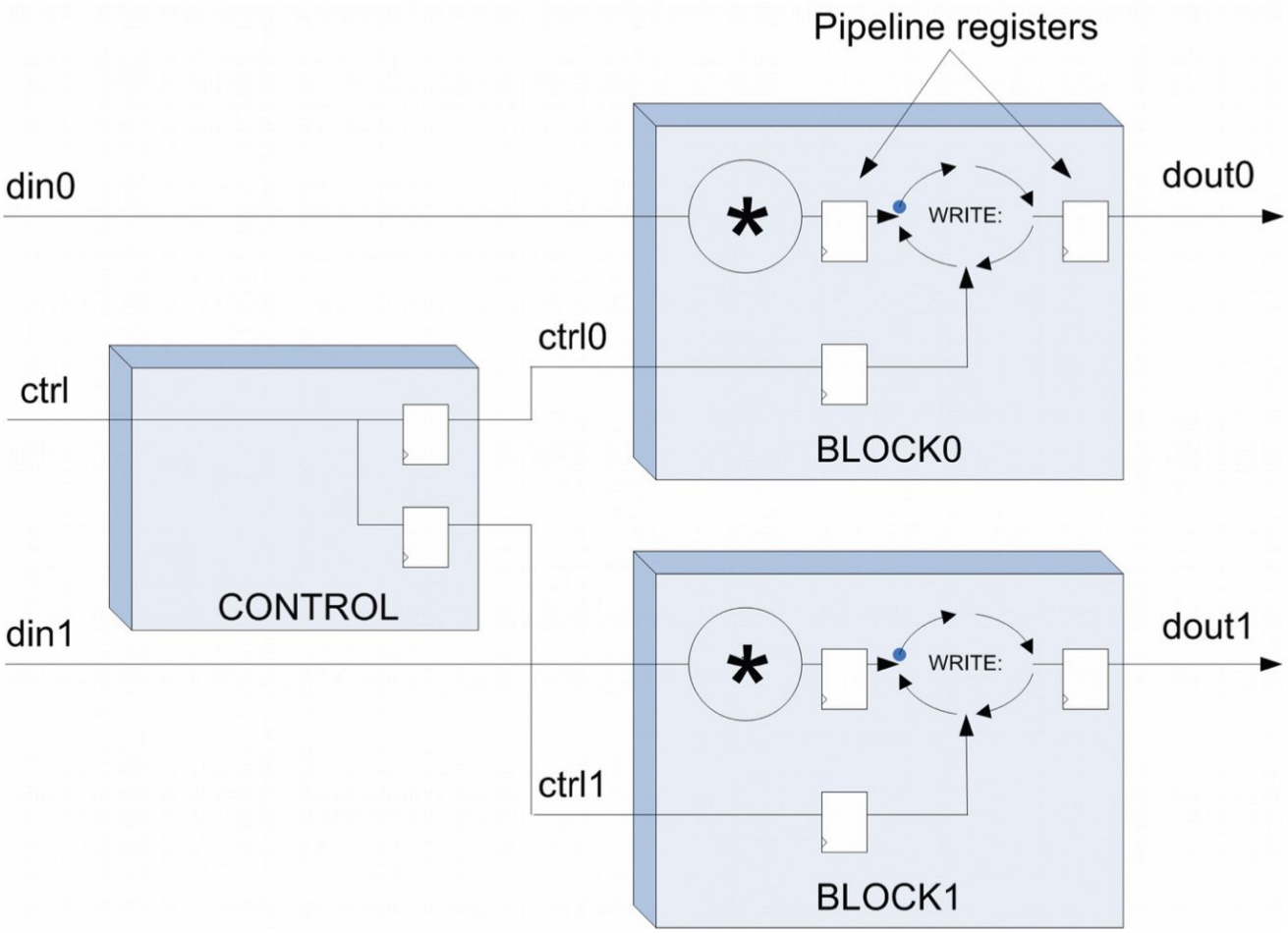
RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	top	return value
ap_rst	in	1	ap_ctrl_hs	top	return value
ap_start	in	1	ap_ctrl_hs	top	return value
ap_done	out	1	ap_ctrl_hs	top	return value
ap_idle	out	1	ap_ctrl_hs	top	return value
ap_ready	out	1	ap_ctrl_hs	top	return value
din_V_V_dout	in	4	ap_fifo	din_V_V	pointer
din_V_V_empty_n	in	1	ap_fifo	din_V_V	pointer
din_V_V_read	out	1	ap_fifo	din_V_V	pointer
dout_V_V_din	out	4	ap_fifo	dout_V_V	pointer
dout_V_V_full_n	in	1	ap_fifo	dout_V_V	pointer
dout_V_V_write	out	1	ap_fifo	dout_V_V	pointer
ctrl_V	in	8	ap_none	ctrl_V	pointer

Duplicating Control IO

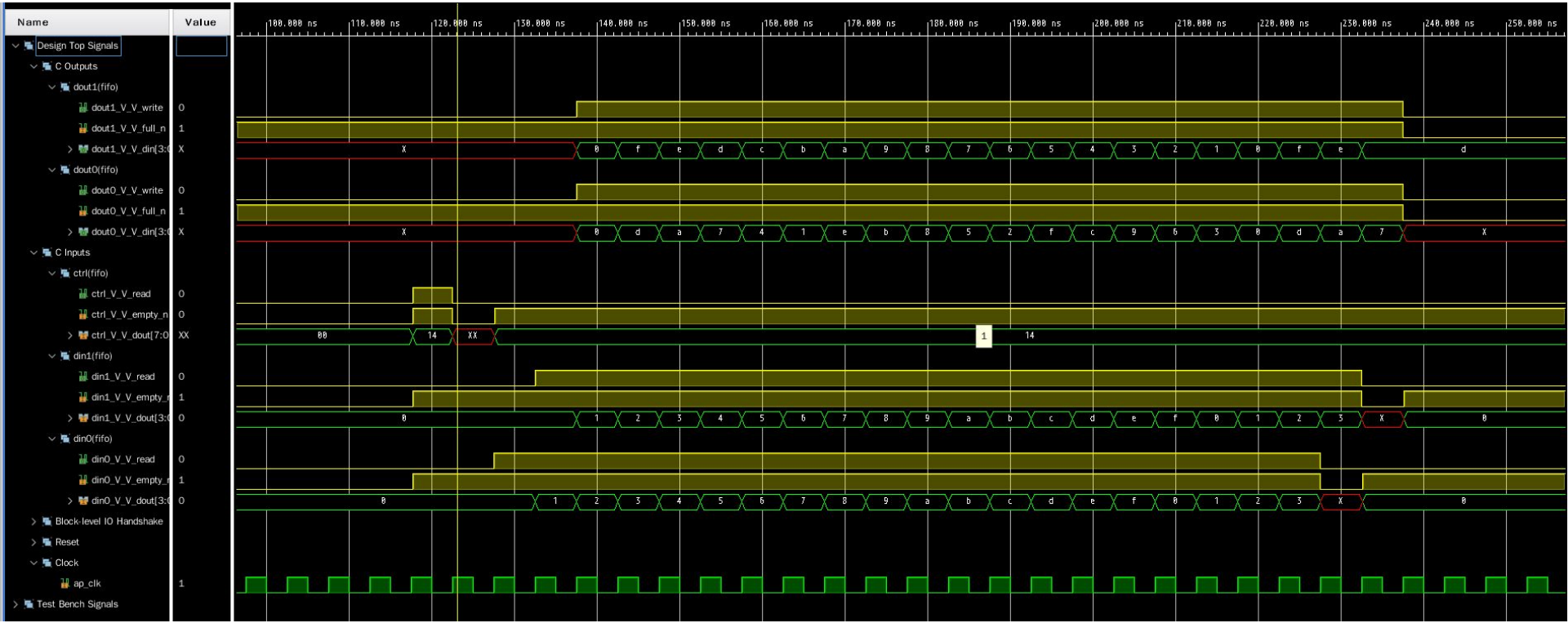
```
class control_mult {
    hls::stream<ctr_type> ctrl0;
    hls::stream<ctr_type> ctrl1;

#pragma hls_design
    void CONTROL(hls::stream<ctr_type>& ctrl, hls::stream<ctr_type>& ctrl0,
        hls::stream<ctr_type>& ctrl1) {
#pragma HLS PIPELINE II = 1
        ctr_type ctrl_int = ctrl.read(); // one read
        ctrl0.write(ctrl_int);
        ctrl1.write(ctrl_int);
    }

#pragma hls_design
    void BLOCK0(hls::stream<dat_type>& din, hls::stream<dat_type>& dout,
        hls::stream<ctr_type>& ctrl) {
#pragma HLS PIPELINE II = 1
        ctr_type ctrl_int = ctrl.read();
        WRITE:for (ctr_type i = 0; i < 255; i++) {
            if (i == ctrl_int) break;
            dout.write(din.read() * 13);
        }
    }

#pragma hls_design
    void BLOCK1(hls::stream<dat_type>& din, hls::stream<dat_type>& dout,
        hls::stream<ctr_type>& ctrl) {
#pragma HLS PIPELINE II = 1
        ctr_type ctrl_int = ctrl.read();
        WRITE:for (ctr_type i = 0; i < 255; i++) {
            if (i == ctrl_int) break;
            dout.write(din.read() * 111);
        }
    }
```

control copy block

```
  public:
    control_mult() {}

#pragma hls_design interface
    void run(hls::stream<dat_type>& din0,
             hls::stream<dat_type>& din1,
             hls::stream<dat_type>& dout0,
             hls::stream<dat_type>& dout1,
             hls::stream<ctr_type>& ctrl) {
#pragma HLS STREAM variable=ctrl0 depth=1
#pragma HLS STREAM variable=ctrl1 depth=1
        CONTROL(ctrl, ctrl0, ctrl1);
        BLOCK0(din0, dout0, ctrl0);
        BLOCK1(din1, dout1, ctrl1);
    }
};
```


Pipeline registers
din0
dout0
WRITE:
ctrl0
ctrl
BLOCK0
CONTROL
din1
dout1
WRITE:
ctrl1
BLOCK1

Waveform


Name
Value
Design Top Signals
C Outputs
dout1(fifo)
dout1_V_V_write 0
dout1_V_V_full_n 1
dout1_V_V_din[3:0 X
dout0(fifo)
dout0_V_V_write 0
dout0_V_V_full_n 1
dout0_V_V_din[3:0 X
C Inputs
ctrl(fifo)
ctrl_V_V_read 0
ctrl_V_V_empty_n 0
ctrl_V_V_dout[7:0 XX
din1(fifo)
din1_V_V_read 0
din1_V_V_empty_n 1
din1_V_V_dout[3:0 0
din0(fifo)
din0_V_V_read 0
din0_V_V_empty_n 1
din0_V_V_dout[3:0 0
Block-level IO Handshake
Reset
Clock
ap_clk 1
Test Bench Signals
100.000 ns
110.000 ns
120.000 ns
130.000 ns
140.000 ns
150.000 ns
160.000 ns
170.000 ns
180.000 ns
190.000 ns
200.000 ns
210.000 ns
220.000 ns
230.000 ns
240.000 ns
250.000 ns

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	4.375 ns	0.62 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
258	258	1.290 us	1.290 us	258	258	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	6	-
FIFO	0	-	10	40	-
Instance	-	-	2072	15659	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	1311	-
Register	-	-	261	-	-
Total	0	0	2343	17016	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	2	31	0

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	top	return value
ap_rst	in	1	ap_ctrl_hs	top	return value
ap_start	in	1	ap_ctrl_hs	top	return value
ap_done	out	1	ap_ctrl_hs	top	return value
ap_idle	out	1	ap_ctrl_hs	top	return value
ap_ready	out	1	ap_ctrl_hs	top	return value
din0_V_V_dout	in	4	ap_fifo	din0_V_V	pointer
din0_V_V_empty_n	in	1	ap_fifo	din0_V_V	pointer
din0_V_V_read	out	1	ap_fifo	din0_V_V	pointer
din1_V_V_dout	in	4	ap_fifo	din1_V_V	pointer
din1_V_V_empty_n	in	1	ap_fifo	din1_V_V	pointer
din1_V_V_read	out	1	ap_fifo	din1_V_V	pointer
dout0_V_V_din	out	4	ap_fifo	dout0_V_V	pointer
dout0_V_V_full_n	in	1	ap_fifo	dout0_V_V	pointer
dout0_V_V_write	out	1	ap_fifo	dout0_V_V	pointer
dout1_V_V_din	out	4	ap_fifo	dout1_V_V	pointer
dout1_V_V_full_n	in	1	ap_fifo	dout1_V_V	pointer
dout1_V_V_write	out	1	ap_fifo	dout1_V_V	pointer
ctrl_V_V_dout	in	8	ap_fifo	ctrl_V_V	pointer
ctrl_V_V_empty_n	in	1	ap_fifo	ctrl_V_V	pointer
ctrl_V_V_read	out	1	ap_fifo	ctrl_V_V	pointer

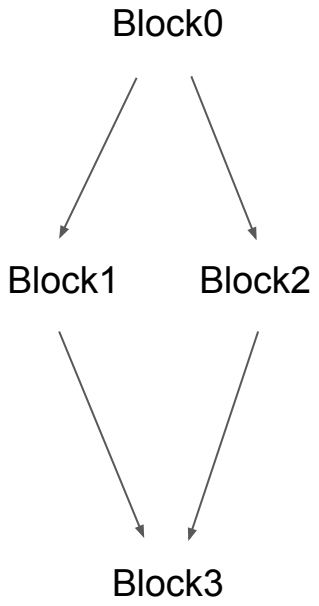
Outline

- Arrays Shared Between Blocks
- Blocks with Common Interface Control Variables
- Reconvergence: Balancing the Latency Between Blocks

Balancing the Latency Between Blocks

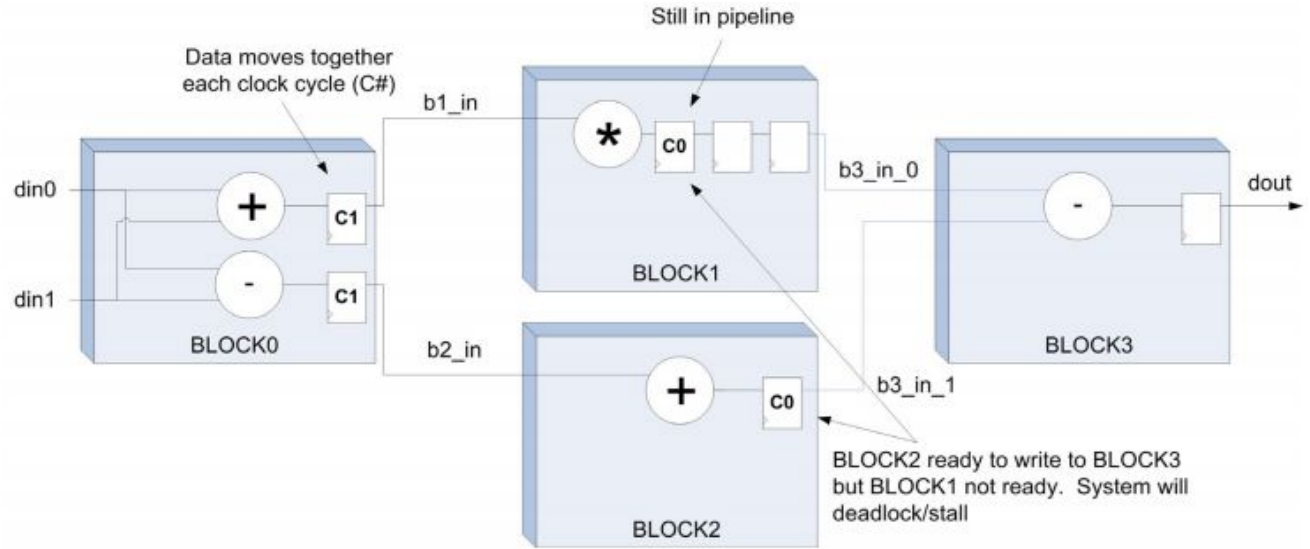
Example 136 Multi-Block Design with Reconvergence

```
 1 #include "ac_int.h"
 2 #include "ac_channel.h"
 3
 4 typedef ac_int<4,false> uint4;
 5
 6 class recon_multadd_diff {
 7
 8
 9   ac_channel<uint4> b1_in;
10   ac_channel<uint4> b2_in;
11   ac_channel<uint4> b3_in_0;
12   ac_channel<uint4> b3_in_1;
13
14   #pragma hls_design
15   void BLOCK0 (ac_channel<uint4> &din0,
16                ac_channel<uint4> &din1,
17                ac_channel<uint4> &dout0,
18                ac_channel<uint4> &dout1) {
19     uint4 tmp0, tmp1;
20     tmp0 = din0.read();
21     tmp1 = din1.read();
22     dout0.write(tmp0 + tmp1);
23     dout1.write(tmp0 - tmp1);
24   }
25
26   #pragma hls_design
27   void BLOCK1 (ac_channel<uint4> &din,
28                ac_channel<uint4> &dout) {
29     uint4 tmp;
30     tmp = din.read() * 13;
31     dout.write(tmp);
32   }
33
34   #pragma hls_design
35   void BLOCK2 (ac_channel<uint4> &din1,
36                ac_channel<uint4> &dout) {
37     uint4 tmp;
38     tmp = din1.read() + 111;
39     dout.write(tmp);
40   }
41
42   #pragma hls_design
43   void BLOCK3 (ac_channel<uint4> &din0,
44                ac_channel<uint4> &din1,
45                ac_channel<uint4> &dout) {
46     uint4 tmp;
47     tmp = din0.read() - din1.read();
48     dout.write(tmp);
49   }
50
51   public:
52
53   recon_multadd_diff () {}
54
55   #pragma hls_design interface
56   void run (ac_channel<uint4> &din0,
57                 ac_channel<uint4> &din1,
58                 ac_channel<uint4> &dout) {
59     BLOCK0(din0,din1,b1_in,b2_in);
60     BLOCK1(b1_in,b3_in_0);
61     BLOCK2(b2_in,b3_in_1);
62     BLOCK3(b3_in_0,b3_in_1,dout);
63   }
64 };
```



Balancing the Latency Between Blocks

Two paths from block 0 to block 3
The latency might be different.



Solution1: Automatic Pipeline Flushing

```
12 #pragma hls_design
13     void BLOCK0(hls::stream<uint4>& din0,
14                 hls::stream<uint4>& din1,
15                 hls::stream<uint4>& dout0,
16                 hls::stream<uint4>& dout1) {
17 #pragma HLS PIPELINE II = 1 enable_flush
18         uint4 tmp0, tmp1;
19         tmp0 = din0.read();
20         tmp1 = din1.read();
21         dout0.write(tmp0 + tmp1);
22         dout1.write(tmp0 - tmp1);
23     }
24
25 #pragma hls_design
26     void BLOCK1(hls::stream<uint4>& din, hls::stream<uint4>& dout) {
27 #pragma HLS PIPELINE II = 1
28         uint4 tmp;
29         tmp = din.read() * 13;
30         dout.write(tmp);
31     }
```

```
33 #pragma hls_design
34     void BLOCK2(hls::stream<uint4>& din1, hls::stream<uint4>& dout) {
35 #pragma HLS PIPELINE II = 1 enable_flush
36         uint4 tmp;
37         tmp = din1.read() + 111;
38         dout.write(tmp);
39     }
40
41 #pragma hls_design
42     void BLOCK3(hls::stream<uint4>& din0, hls::stream<uint4>& din1,
           hls::stream<uint4>& dout) {
43 #pragma HLS PIPELINE II = 1 enable_flush
44         uint4 tmp;
45         tmp = din0.read() - din1.read();
46         dout.write(tmp);
47     }
```

Solution1: Automatic Pipeline Flushing

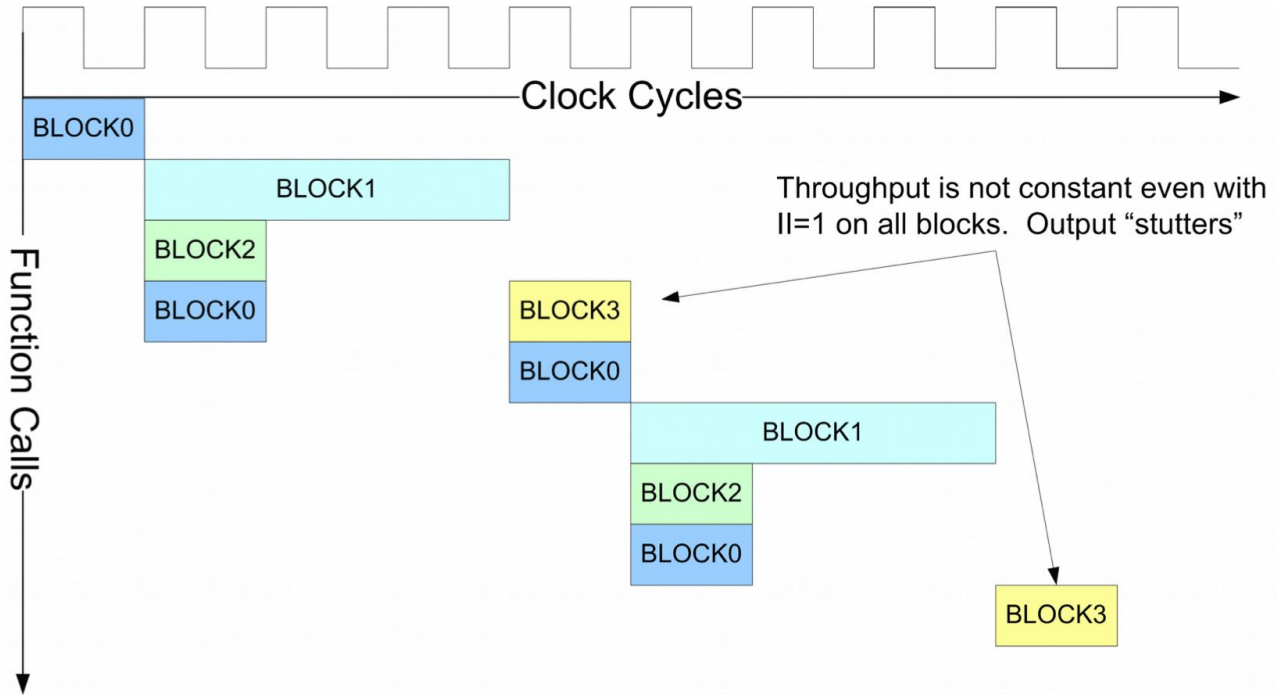


Illustration 130: Multi-block design with Automatic Pipeline Flushing

Latency

Synthesis Report for 'BLOCK1'

General Information

Date:	Mon Apr 12 23:28:31 2021
Version:	2019.2 (Build 2704478 on Wed Nov 06 22:10:23 MST
Project:	feedback_reconverge_flush
Solution:	solution1
Product	zynq
Target device:	xc7z020-clg484-1

Performance Estimates

+ Timing
- Latency

  - Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
2	2	10.000 ns	10.000 ns	1	1	function

Synthesis Report for 'BLOCK2'

General Information

Date:	Mon Apr 12 23:28:31 2021
Version:	2019.2 (Build 2704478 on Wed Nov 06 22:10:23 MST
Project:	feedback_reconverge_flush
Solution:	solution1
Product	zynq
Target device:	xc7z020-clg484-1

Performance Estimates

+ Timing
- Latency

  - Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
1	1	5.000 ns	5.000 ns	1	1	function

Solution2: Manually Setting FIFO Depths

```
52 #pragma hls_design interface
53     void run(hls::stream<uint4>& din0, hls::stream<uint4>& din1, hls::stream<uint4>& dout) {
54 #pragma HLS STREAM variable=b1_in depth=1
55 #pragma HLS STREAM variable=b2_in depth=1
56 #pragma HLS STREAM variable=b3_in_0 depth=1
57 #pragma HLS STREAM variable=b3_in_1 depth=2
58         BLOCK0(din0, din1, b1_in, b2_in);
59         BLOCK1(b1_in, b3_in_0);
60         BLOCK2(b2_in, b3_in_1);
61         BLOCK3(b3_in_0, b3_in_1, dout);
62     }
63 };
64
65 void top(hls::stream<uint4>& din0, hls::stream<uint4>& din1, hls::stream<uint4>& dout);
```

Solution2: Manually Setting FIFO Depths

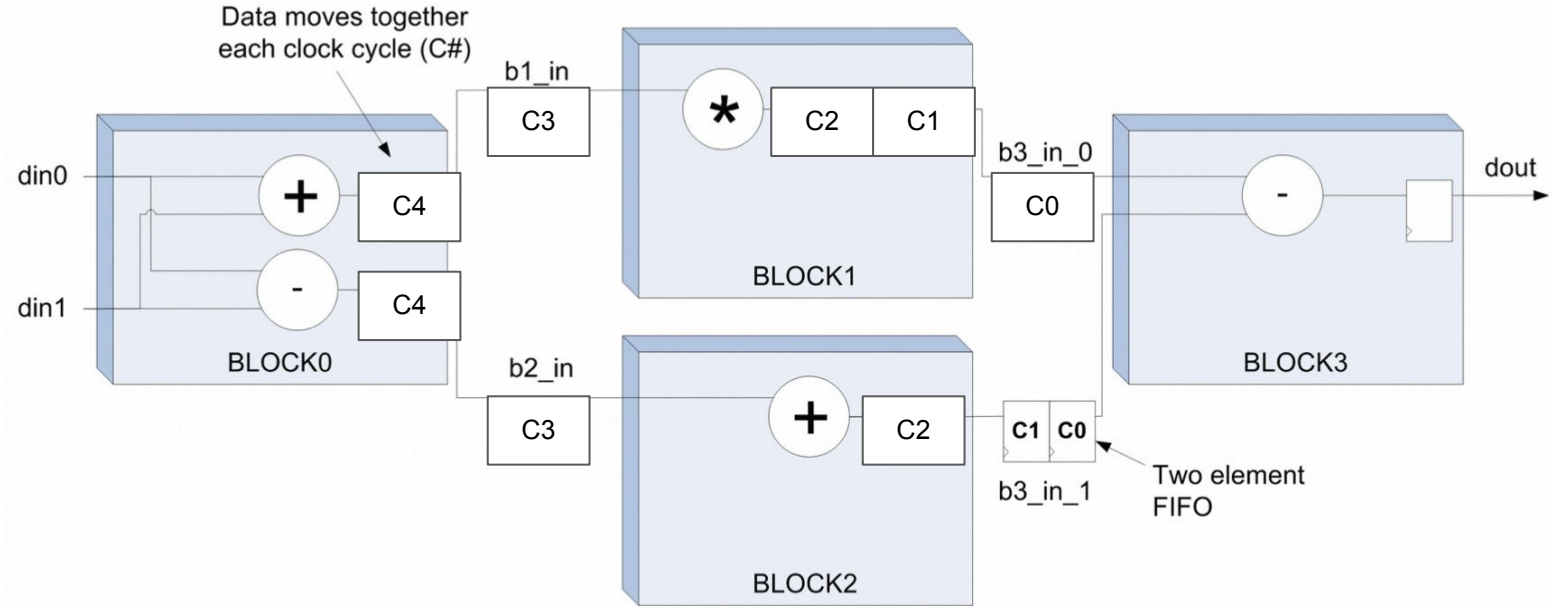
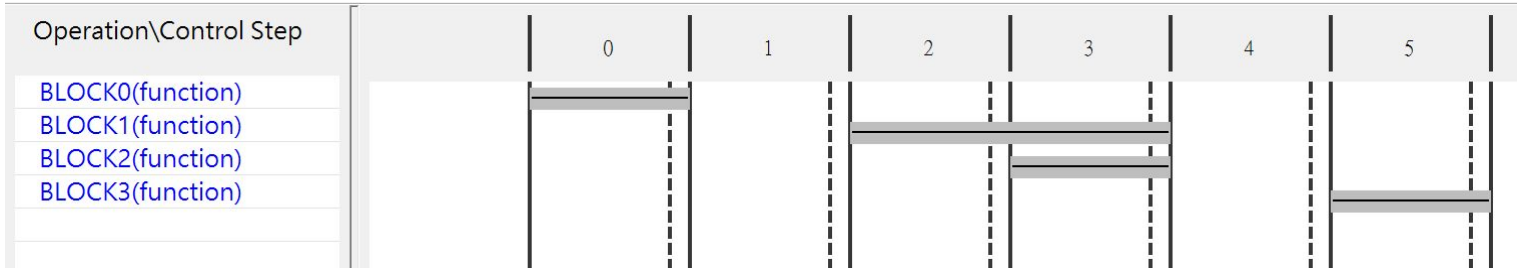


Illustration 131: Manually Setting FIFO Depths

Solution2: Manually Setting FIFO Depths



Solution2: Manually Setting FIFO Depths

```
52 #pragma hls_design interface
53     void run(hls::stream<uint4>& din0, hls::stream<uint4>& din1, hls::stream<uint4>& dout) {
54 #pragma HLS STREAM variable=b1_in depth=1
55 #pragma HLS STREAM variable=b2_in depth=2
56 #pragma HLS STREAM variable=b3_in_0 depth=1
57 #pragma HLS STREAM variable=b3_in_1 depth=1
58         BLOCK0(din0, din1, b1_in, b2_in);
59         BLOCK1(b1_in, b3_in_0);
60         BLOCK2(b2_in, b3_in_1);
61         BLOCK3(b3_in_0, b3_in_1, dout);
62     }
63 };
64
65 void top(hls::stream<uint4>& din0, hls::stream<uint4>& din1, hls::stream<uint4>& dout);
```

Solution2: Manually Setting FIFO Depths

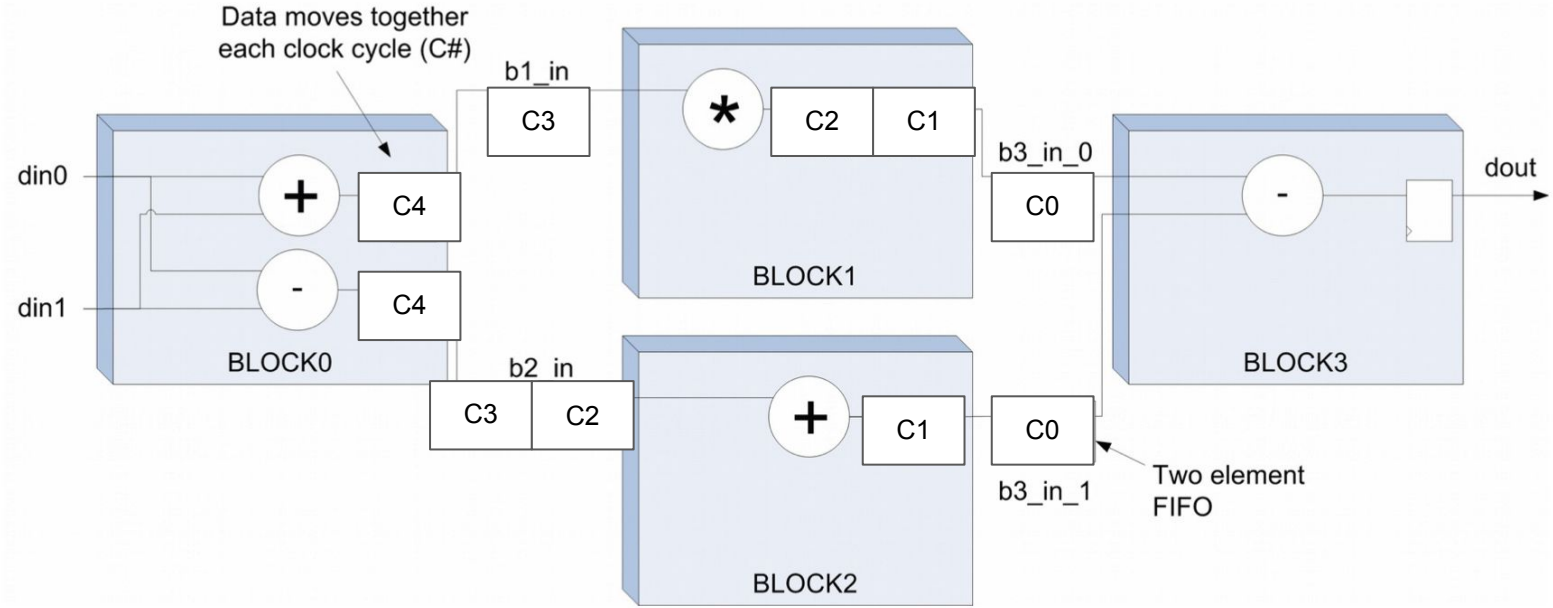
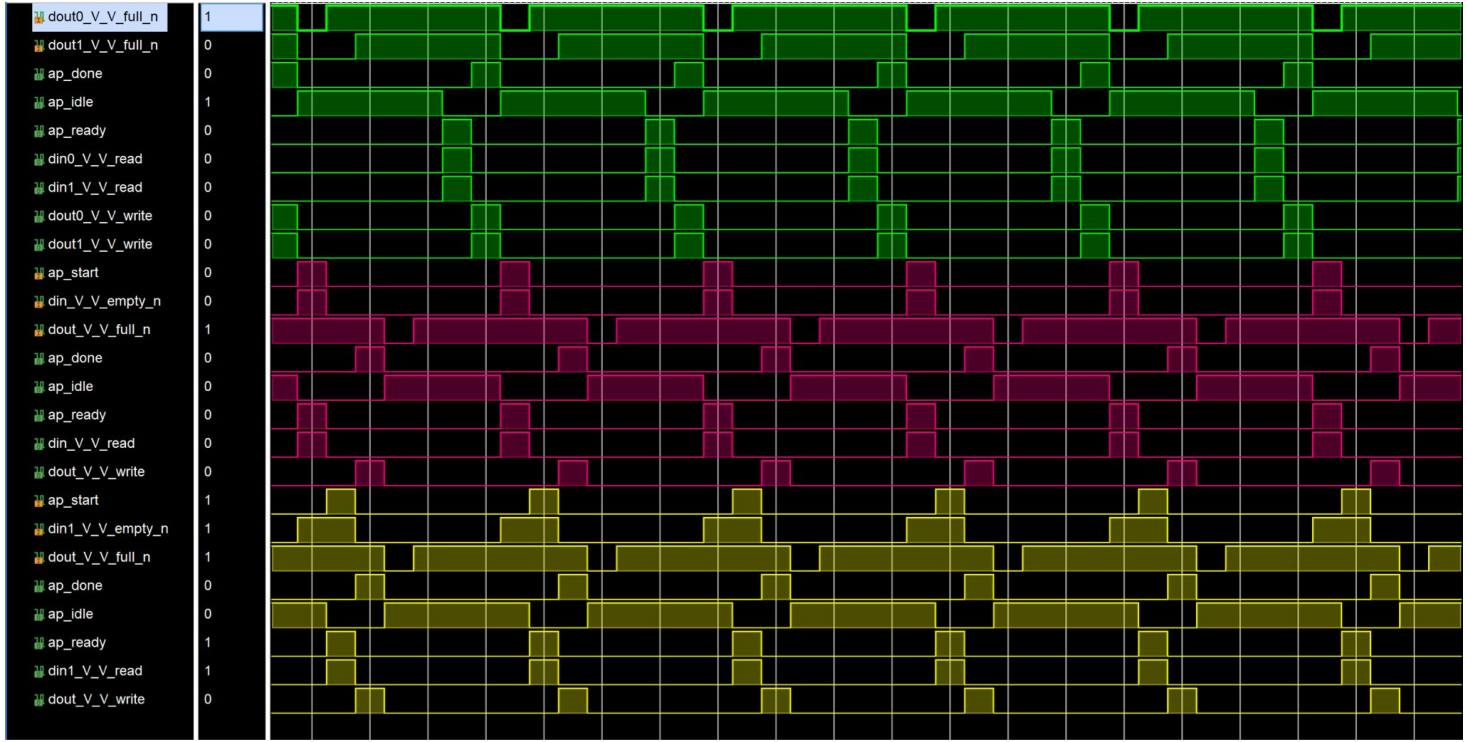
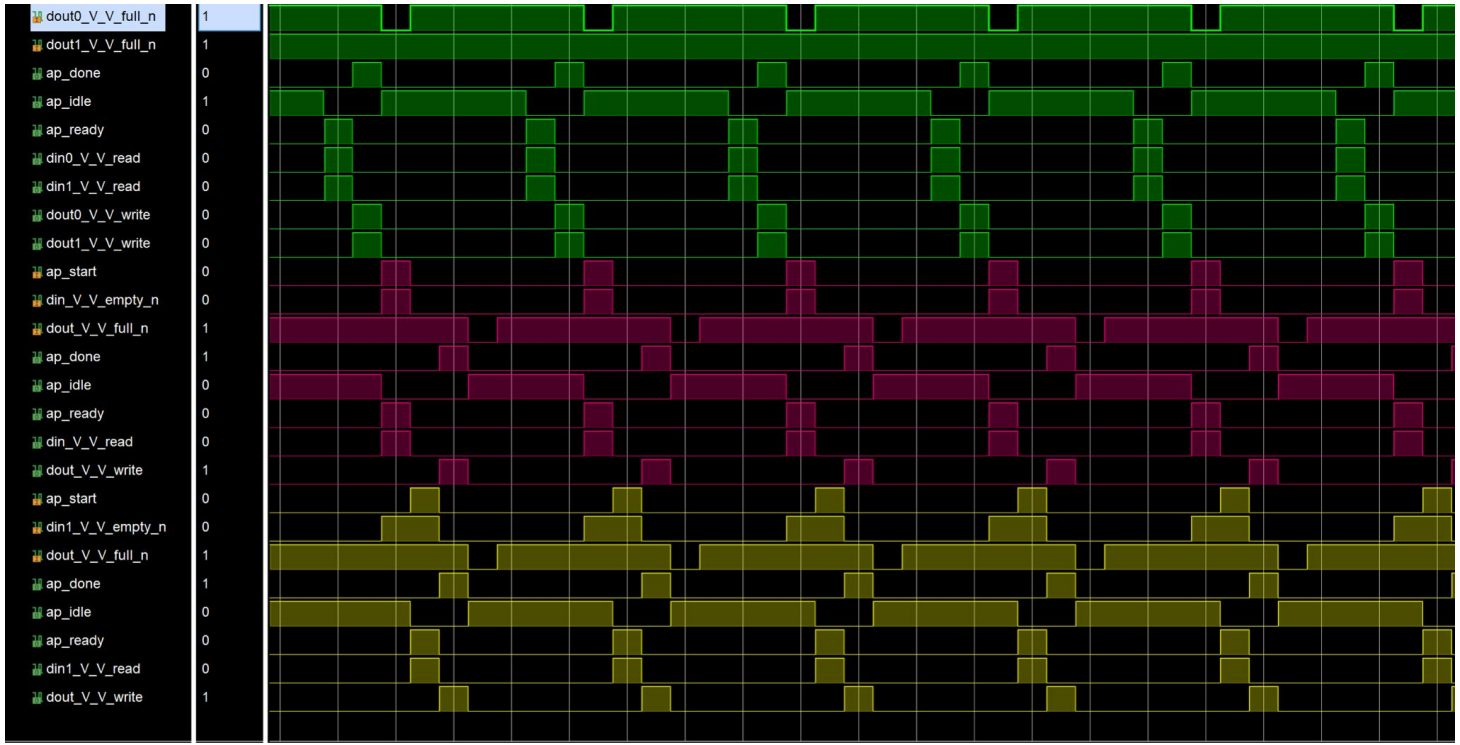


Illustration 131: Manually Setting FIFO Depths

Solution2: Manually Setting FIFO Depths



Solution2: Manually Setting FIFO Depths



Github

- https://github.com/s950449/HLS_LabA_Team11

Thank you for listening!