

Team 12 - Sequential and Combinational Hardware

tags: hls

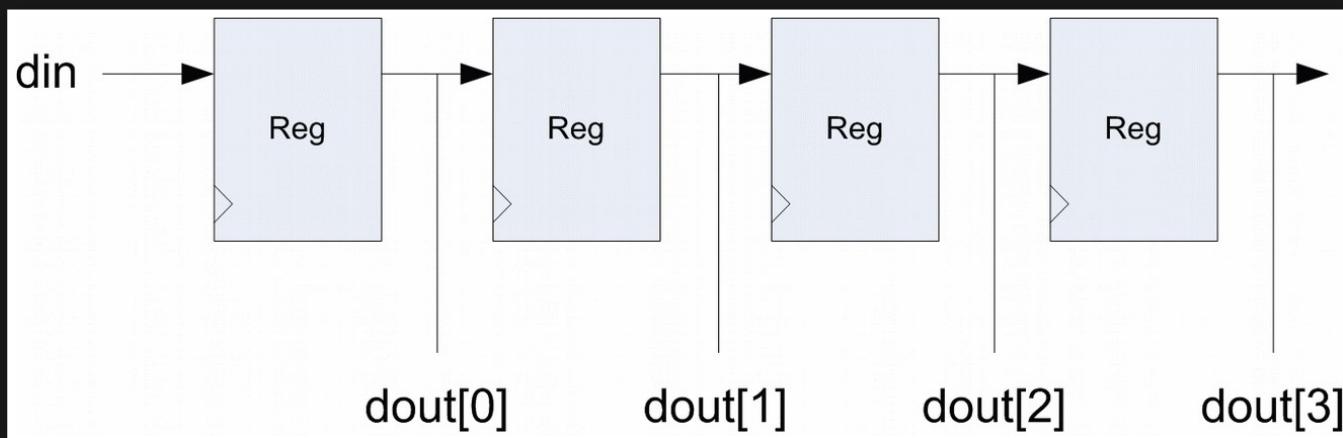
B07902143 陳正康
D08922024 蔡德育
R09943159 王則勛

GitHub Repo: <https://github.com/soyccan/HLS-Bluebook>

Shift Registers

Basic Shift Register

```
// shift_reg_basic.cpp
void shift_reg_basic(int din, int dout[4])
{
    static int regs[4];
SHIFT:
    for (int i = 4 - 1; i >= 0; i--) {
        #pragma HLS UNROLL
        if (i == 0)
            regs[i] = din;
        else
            regs[i] = regs[i - 1];
    }
WRITE:
    for (int i = 0; i < 4; i++) {
        #pragma HLS UNROLL
        dout[i] = regs[i];
    }
}
```



Why static array?

```
1 void top() {
2     int dout[4];
3
4     // shift in 1
5     shift_reg_basic(1, dout);
6     // Use dout...
7
8     // shift in 2 (1 should live still)
9     shift_reg_basic(2, dout);
10    // Use dout...
11 }
```

Design Constraints

- regs array mapped to registers
 - What directive?
- SHIFT and WRITE loops fully unrolled
 - Necessary to synthesis a shift register

Register Usage

- With unrolling

Name	FF	LUT	Bits	Const Bits
ap_CS_fsm	4	0	4	0
regs_0	8	0	8	0
regs_0_load_reg_315	8	0	8	0
regs_1	8	0	8	0
regs_1_0	32	0	32	0
regs_1_0_load_reg_300	32	0	32	0
regs_1_1	32	0	32	0
regs_1_2	32	0	32	0
regs_1_2_load_reg_305	32	0	32	0
regs_1_3	32	0	32	0
regs_1_4	32	0	32	0
regs_1_4_load_reg_310	32	0	32	0
regs_1_5	32	0	32	0
regs_1_6	32	0	32	0
regs_2	8	0	8	0
Total	356	0	356	0

- Without unrolling

Name	FF	LUT	Bits	Const Bits
ap_CS_fsm	8	0	8	0
grp_shift_reg_fu_235_ap_start_reg	1	0	1	0
i_0_i_reg_223	4	0	4	0
icmp_ln13_reg_295	1	0	1	0
tmp_reg_291	1	0	1	0
Total	15	0	15	0

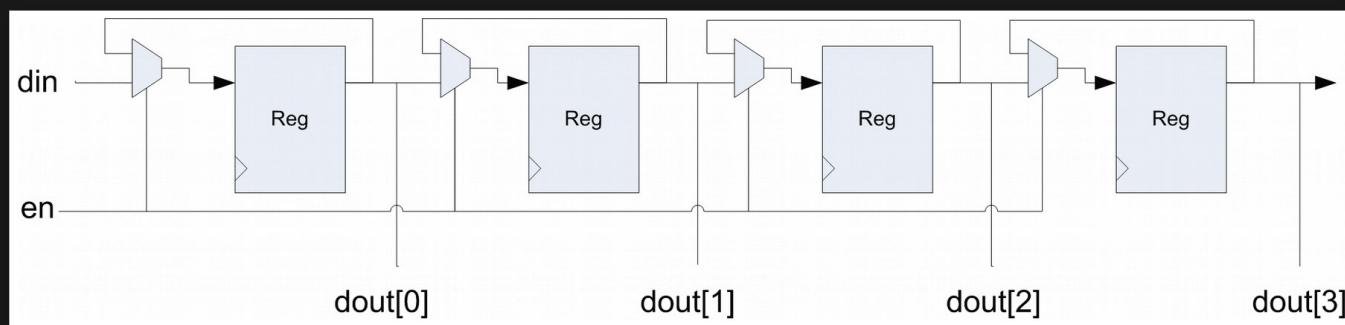
```
* Memory:
+-----+-----+-----+
|     Memory |     Module | BRAM_18K|
+-----+-----+-----+
|shift_reg0_regs_U |shift_reg_shift_rbkb | 1|
+-----+-----+-----+
|Total           |                   | 1|
+-----+-----+-----+
```

Basic Block Signals

- `clock`, `clock enable`, `reset` are added automatically during synthesis process
- We can add a few more manually

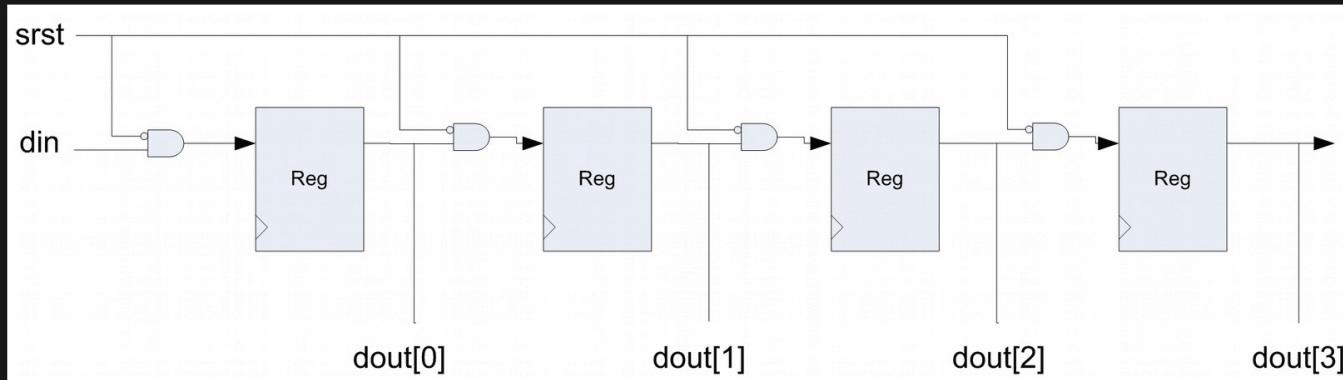
With Enable Signal

```
// shift_reg_en.cpp
void shift_reg_en(int din, int dout[4], bool en)
{
    static int regs[4];
    for (int i = 4 - 1; i >= 0; i--) {
        #pragma HLS UNROLL
        if (en) {
            if (i == 0)
                regs[i] = din;
            else
                regs[i] = regs[i - 1];
        }
    }
    for (int i = 0; i < 4; i++) {
        #pragma HLS UNROLL
        dout[i] = regs[i];
    }
}
```



With Synchronous Reset Signal

```
// shift_reg_sreset.cpp
void shift_reg_sreset(int din,
                      int dout[4],
                      bool srst)
{
    static int regs[4]
    for (int i = 4 - 1; i >= 0; i--) {
        #pragma HLS UNROLL
        if (srst)
            regs[i] = 0;
        else {
            if (i == 0)
                regs[i] = din;
            else
                regs[i] = regs[i - 1];
        }
    }
    for (int i = 0; i < 4; i++) {
        #pragma HLS UNROLL
        dout[i] = regs[i];
    }
}
```

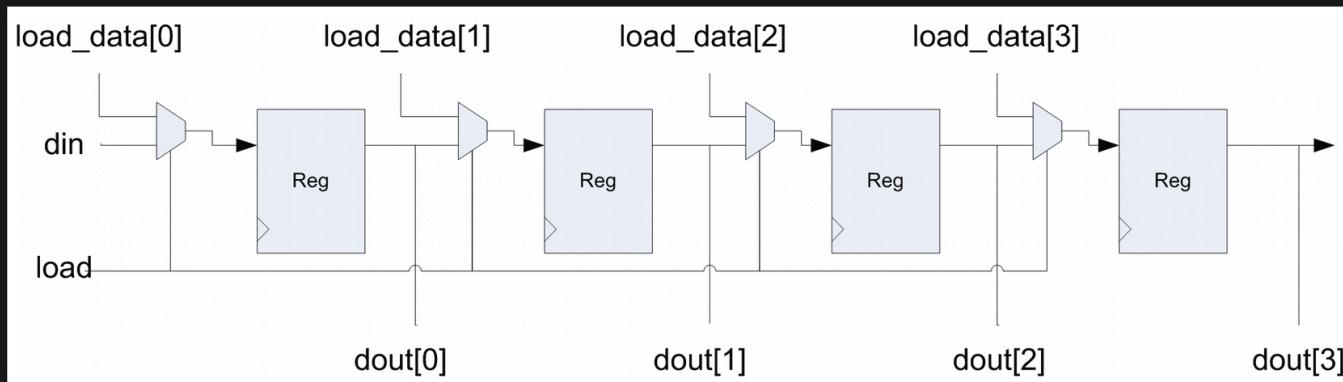


Synchronous vs. Asynchronous

- Any logic described in C++ is **synchronous**

With Load Signal

```
// shift_reg_load.cpp
void shift_reg_load(int din, int load_data[4],
                     int dout[4], bool load) {
    static int regs[4];
    for (int i = 4 - 1; i >= 0; i--) {
        #pragma HLS UNROLL
        if (load)
            regs[i] = load_data[i];
        else {
            if (i == 0)
                regs[i] = din;
            else
                regs[i] = regs[i - 1];
        }
    }
    for (int i = 0; i < 4; i++) {
        #pragma HLS UNROLL
        dout[i] = regs[i];
    }
}
```



Reuse Problem

- static array are shared across multiple function calls
- To create multiple registers, one function should be written for each of them
 - Reduce reusability
 - C++ template function/class can help

Template Function

- Improve reusability

```
// shift_reg_tmpl.h
template <int ID, typename dType, int NUM_REGS>
void shift_reg(dType din, dType dout[NUM_REGS]) {
    static dType regs[NUM_REGS];
    for (int i = NUM_REGS - 1; i >= 0; i--) {
        #pragma HLS UNROLL
        if (i == 0)
            regs[i] = din;
        else
            regs[i] = regs[i - 1];
    }
    for (int i = 0; i < NUM_REGS; i++) {
        #pragma HLS UNROLL
        dout[i] = regs[i];
    }
}
```

Template Function

```
// shift_reg_template.cpp
#include "shift_reg_tmpl.h"
void shift_reg_template(int din0,
                       char din1,
                       int dout0[4],
                       char dout1[8])
{
    // first call
    shift_reg<1, int, 4>(din0, dout0);

    // second call
    shift_reg<2, char, 8>(din1, dout1);
}
```

Template Class

- Alternative approach of template function
 - Problem: Error when synthesizing

```
// shift_class.h (troublesome version)
template <typename dataType, int NUM_REGS>
class shift_class
{
private:
    dataType regs[NUM_REGS];
    bool en;
    bool sync_rst;
    bool ld;
    dataType* load_data;

public:
    shift_class(bool en = false,
                bool sync_rst = false,
                bool ld = false,
                dataType load_data[NUM_REGS]) {
        this->en = en;
        this->sync_rst = sync_rst;
        this->ld = ld;
        this->load_data = load_data;
    }
}
```


Template Class

- Pass by reference problem: The class member `load_data` loses information of array length

```
1  class shift_class {  
2      // member  
3      dataType* load_data;  
4  
5      // constructor  
6      shift_class(...) {  
7          this->load_data = load_data;  
8      }  
9  }
```

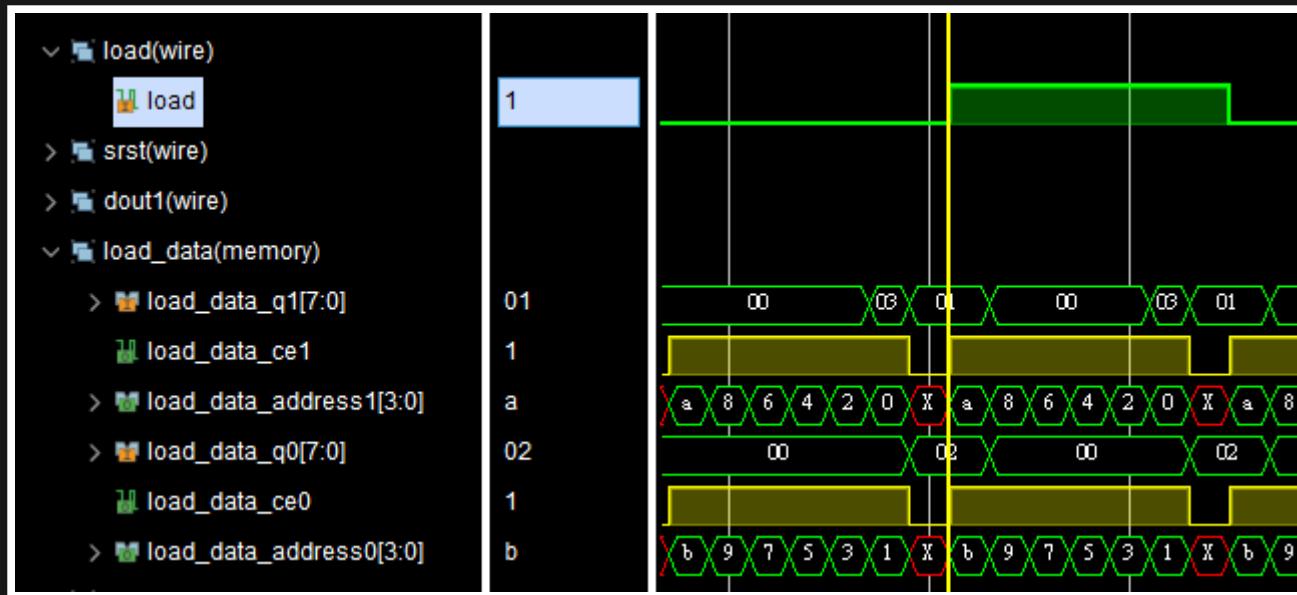
```
1 // shift_class.h (refined version)
2 template <typename dataType, int NUM_REGS>
3 class shift_class
4 {
5 private:
6     dataType regs[NUM_REGS];
7     bool en;
8     bool sync_rst;
9     bool ld;
10    //dataType* load_data;
11
12 public:
13 public:
14     shift_class(bool en = false,
15                 bool sync_rst = false,
16                 bool ld = false,
17                 /*dataType load_data[NUM_REGS]*) {
18     this->en = en;
19     this->sync_rst = sync_rst;
20     this->ld = ld;
21     //this->load_data = load_data;
```

```
1 void shift(dataType din, dataType load_data[NUM_REGS])
2 {
3     for (int i = NUM_REGS - 1; i >= 1; i--) {
4         #pragma HLS UNROLL
5         if (en) {
6             if (sync_rst)
7                 regs[i] = 0;
8             else if (ld)
9                 regs[i] = load_data[i];
10            else if (i == 0)
11                regs[i] = din;
12            else
13                regs[i] = regs[i - 1];
14        }
15    }
16 }
17 dataType operator[](int i) { return regs[i]; }
18 };
19 #endif
```

```
1 // shift_reg_class.cpp (refined version)
2 void shift_reg_class(int din,
3                      int load_data[4],
4                      int dout0[4],
5                      bool srst,
6                      bool ld,
7                      bool en) {
8     static shift_class<int, 4>
9         shift_reg0(en, srst, ld);
10
11    shift_reg0.shift(din, load_data);
12
13    for (int i = 0; i < 4; i++)
14        #pragma HLS UNROLL
15        dout0[i] = shift_reg0[i];
16 }
```

Template Class

- Correct result: Pointer accessed as array



Template Class

- Another problem: Class member `regs[]` array is synthesized as memory, instead of register
 - Solution 1:
Add `#pragma HLS INLINE` at all member function
(include operator[])
 - Solution 2:
Add `#pragma HLS ARRAY_PARTITION
variable=regs complete dim=1` at constructor

	Negative Slack	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
shift_reg	-	1	0	1605	1950	1~13	2 ~ 14	none
shift_reg_class	-	1	0	253	442	6~11	6 ~ 11	none
shift_reg_en	-	0	0	282	222	1	1	none
shift_reg_load	-	0	0	185	204	0	0	none
shift_reg_sreset	-	0	0	185	204	0	0	none
shift_reg_ip	-	0	0	165	165	12	12	none
shift_reg_basic	-	0	0	185	108	0	0	none
shift_reg_template	-	0	0	345	108	0	0	none

Inline Function

```
1 int child(int x) {
2     #pragma HLS INLINE
3     return x + 1;
4 }
5
6 void parent() {
7     int x = 0;
8     int y = child(x);
9 }
10
11 // Becomes:
12 void parent() {
13     int x = 0;
14     int y = x + 1;
15 }
```

Template Class - Conclusion

- Class is problematic with array
- Class vs. Function
 - Class: Pass to object member
 - Members don't know operations in the far future
 - Undesired hardware when object member is array or pointer to array
 - Require directives to do inline, unroll...
 - Function: Pass to function argument
 - All operations described within the function
 - More predictable

Shift Register IP Provided by Xilinx

- Array written in C may not be synthesized to be a shift register if:
 - It's accessed in the middle
 - Somehow FF is synthesized instead
- The IP guarantees a shift register

- Simplified from Xilinx code
 - No INLINE, UNROLL

```
1 #include <ap_shift_reg.h>
2 void shift_reg_ip(int din, int dout[4], bool en)
3 {
4     // Define a variable of type ap_shift_reg<type, depth>
5     // - Sreg must use the static qualifier
6     // - Sreg will hold integer data types
7     // - Sreg will hold 4 data values
8     static ap_shift_reg<int, 4> Sreg;
9     int var1;
10
11    // Read location 3 of Sreg into var1
12    // THEN if En=1
13    // Shift all values up one and load In1 into location 0
14    var1 = Sreg.shift(din, 3, en);
15
16 WRITE:
17     for (int i = 0; i < N_REGS; i++)
18         #pragma HLS UNROLL
19         dout[i] = Sreg.read(i);
20 }
```

- Unrolling loop “WRITE” failed since ap_shift_reg::read() does not inline

Operation\Control Step		0	1	2	3	4	5	6	7	8	9	10	11	12
en_read(read)														
din_read(read)														
DataOut_0(shift_read)														
DataOut(shift_read)														
DataOut_1(shift_read)														
DataOut_2(shift_read)														
DataOut_3(shift_read)														
DataOut_4(shift_read)														
DataOut_5(shift_read)														
DataOut_6(shift_read)														
DataOut_7(shift_read)														
DataOut_8(shift_read)														
DataOut_9(shift_read)														
DataOut_10(shift_read)														
DataOut_11(shift_read)														

Compare Xilinx IP and Custom Implementation

Basic + Enable Version
L: Xilinx IP; R: Our version

* Summary:						
Name	BRAM_18K	DSP48E	FF	LUT	URAM	
DSP	-	-	-	-	-	-
Expression	-	-	-	-	-	-
FIFO	-	-	-	-	-	-
Instance	-	-	-	-	-	-
Memory	0	-	64	32	-	-
Multiplexer	-	-	-	133	-	-
Register	-	-	101	-	-	-
Total	0	0	165	165	0	
Available	280	220	106400	53200	0	
Utilization (%)	0	0	~0	~0	0	

==== Utilization Estimates =====						
* Summary:						
Name	BRAM_18K	DSP48E	FF	LUT	URAM	
DSP	-	-	-	-	-	-
Expression	-	-	-	-	-	-
FIFO	-	-	-	-	-	-
Instance	-	-	-	-	-	-
Memory	-	-	-	-	-	-
Multiplexer	-	-	-	222	-	-
Register	-	-	282	-	-	-
Total	0	0	282	222	0	
Available	280	220	106400	53200	0	
Utilization (%)	0	0	~0	~0	0	

Compare Xilinx IP and Custom Implementation

Xilinx IP

* Memory:									
Memory	Module	BRAM_18K	FF	LUT	URAM	Words	Bits	Banks	W*Bits*Banks
Sreg_Array_U	shift_reg_ip_Sregbkb	0	64	32	0	12	32	1	384
Total		0	64	32	0	12	32	1	384

Our

* Memory:
N/A

Compare Xilinx IP and Custom Implementation

L: Xilinx IP; R: Our version

* Multiplexer:				
Name	LUT	Input Size	Bits	Total Bits
Sreg_Array_address0	56	13	4	52
Sreg_Array_ce0	9	2	1	2
Sreg_Array_we0	9	2	1	2
ap_NS_fsm	59	14	1	14
Total	133	31	7	70

* Multiplexer:				
Name	LUT	Input Size	Bits	Total Bits
ap_NS_fsm	15	3	1	3
ap_return_0	9	2	8	16
ap_return_1	9	2	8	16
ap_return_10	9	2	8	16
ap_return_11	9	2	8	16
ap_return_2	9	2	8	16
ap_return_3	9	2	8	16
ap_return_4	9	2	8	16
ap_return_5	9	2	8	16
ap_return_6	9	2	8	16
ap_return_7	9	2	8	16
ap_return_8	9	2	8	16
ap_return_9	9	2	8	16
dout_0_write_assign_reg_136	9	2	8	16
dout_10_write_assign_reg_46	9	2	8	16
dout_1_write_assign_reg_127	9	2	8	16
dout_2_write_assign_reg_118	9	2	8	16
dout_3_write_assign_reg_109	9	2	8	16
dout_4_write_assign_reg_100	9	2	8	16
dout_5_write_assign_reg_91	9	2	8	16
dout_6_write_assign_reg_82	9	2	8	16
dout_7_write_assign_reg_73	9	2	8	16
dout_8_write_assign_reg_64	9	2	8	16
dout_9_write_assign_reg_55	9	2	8	16
Total	222	49	185	371

Compare Xilinx IP and Custom Implementation

L: Xilinx IP; R: Our version

* Register:				
Name	FF	LUT	Bits	Const Bits
ap_CS_fsm	13	0	13	0
trunc_ln25_10_reg_347	8	0	8	0
trunc_ln25_1_reg_302	8	0	8	0
trunc_ln25_2_reg_307	8	0	8	0
trunc_ln25_3_reg_312	8	0	8	0
trunc_ln25_4_reg_317	8	0	8	0
trunc_ln25_5_reg_322	8	0	8	0
trunc_ln25_6_reg_327	8	0	8	0
trunc_ln25_7_reg_332	8	0	8	0
trunc_ln25_8_reg_337	8	0	8	0
trunc_ln25_9_reg_342	8	0	8	0
trunc_ln25_reg_297	8	0	8	0
Total	101	0	101	0

* Register:				
Name	FF	LUT	Bits	Const Bits
ap_CS_fsm	1	0	1	0
ap_return_0_preg	8	0	8	0
ap_return_10_preg	8	0	8	0
ap_return_11_preg	8	0	8	0
ap_return_1_preg	8	0	8	0
ap_return_2_preg	8	0	8	0
ap_return_3_preg	8	0	8	0
ap_return_4_preg	8	0	8	0
ap_return_5_preg	8	0	8	0
ap_return_6_preg	8	0	8	0
ap_return_7_preg	8	0	8	0
ap_return_8_preg	8	0	8	0
ap_return_9_preg	8	0	8	0
regs_5_0	8	0	8	0
regs_5_1	8	0	8	0
regs_5_10	8	0	8	0
regs_5_2	8	0	8	0
regs_5_3	8	0	8	0
regs_5_4	8	0	8	0
regs_5_5	8	0	8	0
regs_5_6	8	0	8	0
regs_5_7	8	0	8	0
regs_5_8	8	0	8	0
regs_5_9	8	0	8	0
Total	185	0	185	0

Top Module Wrapper

- Completely partition input and output array
- Include all examples above in a top module
- Set `#pragma HLS INLINE off` in each function so that modules are separated in analyzer

```
1 void shift_reg(dType din0,
2                 int din1,
3                 dType load_data[N_REGS],
4                 dType dout0[N_REGS],
5                 int dout1[N_REGS1],
6                 bool srst,
7                 bool load,
8                 bool en,
9                 sType select)
10 {
11 #pragma HLS ARRAY_PARTITION variable=dout1 complete dim=1
12 #pragma HLS ARRAY_PARTITION variable=dout0 complete dim=1
13 #pragma HLS ARRAY_PARTITION variable=load_data complete dim=1
```

```
1      switch (select) {
2      case 0:
3          shift_reg_basic(din0, dout0);
4          break;
5      case 1:
6          shift_reg_en(din0, dout0, en);
7          break;
8      case 2:
9          shift_reg_load(din0, load_data, dout0, load);
10         break;
11     case 3:
12         shift_reg_sreset(din0, dout0, srst);
13         break;
14     case 4:
15         shift_reg_class(din0, load_data, dout0, dout0 + N_REGS0, srst, load, en);
16         break;
17     case 5:
18         shift_reg_ip(din0, dout0, en);
19         break;
20     default:
21         shift_reg_template(din0, din1, dout0, dout1);
```

Multiplexors

- Binary MUX
 - Used in array indexing
- Onehot MUX
 - Used in complicated control logic
 - Cost less than Binary MUX

Binary MUX

Operator ?

```
1 | int mux_2to1(int din[2], bool sel)
2 | {
3 |     return sel ? din[0] : din[1];
4 | }
```

Array Indexing

```
1 int mux_binary(int din[8], ap_uint<3> sel)
2 {
3     return din[sel];
4 }
```

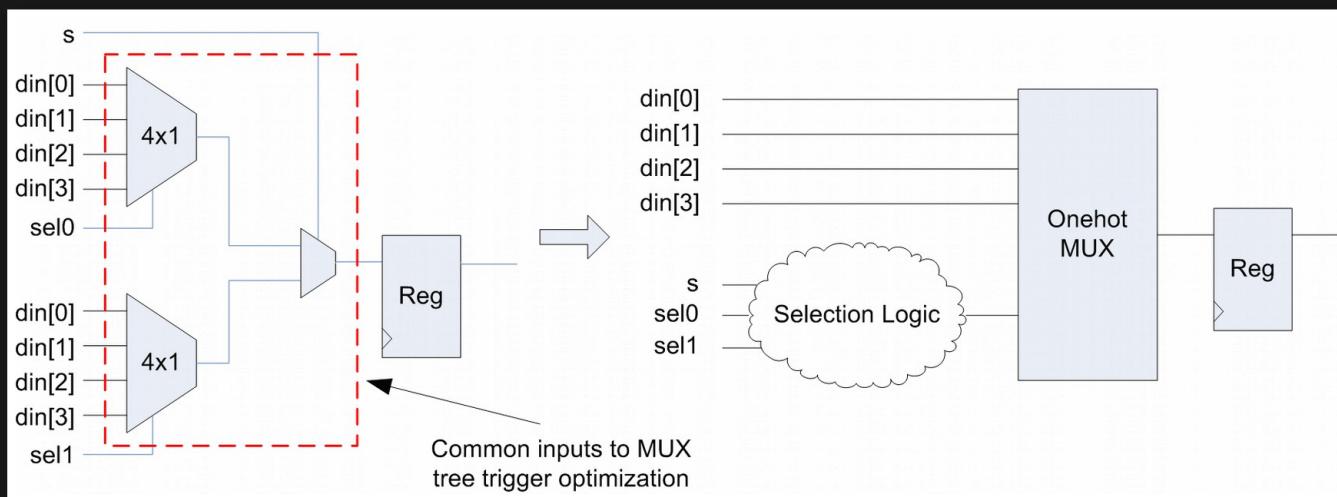
* Summary:						
Name	BRAM_18K	DSP48E	FF	LUT	URAM	
DSP	-	-	-	-	-	-
Expression	-	-	-	-	-	-
FIFO	-	-	-	-	-	-
Instance	-	-	0	45	-	-
Memory	-	-	-	-	-	-
Multiplexer	-	-	-	-	-	-
Register	-	-	-	-	-	-
Total	0	0	0	45	0	
Available	280	220	106400	53200	0	
Utilization (%)	0	0	0	~0	0	

Data Type for Select

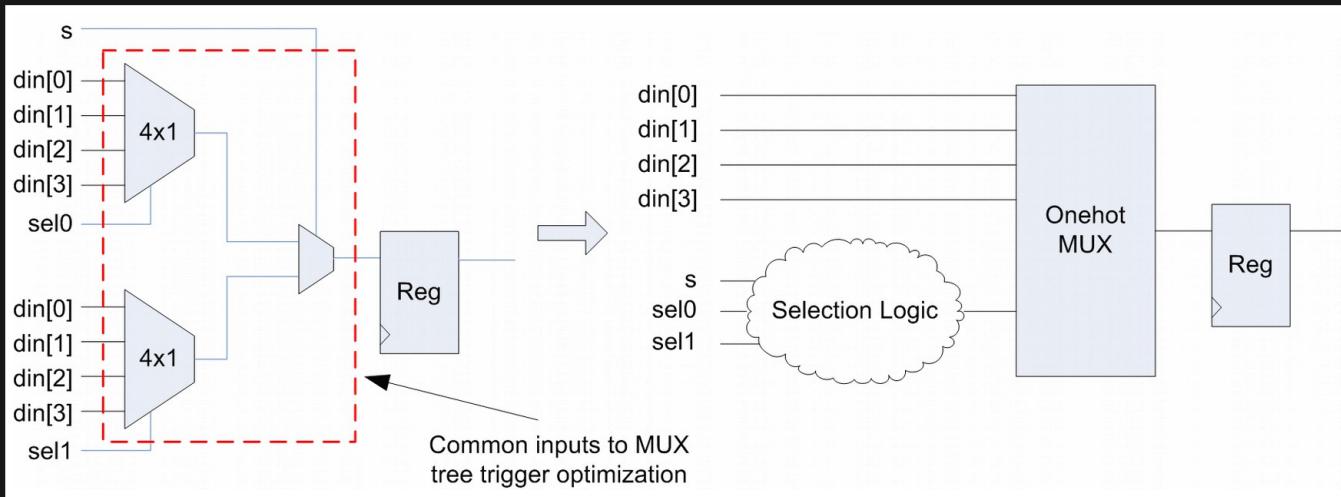
- An array of N elements requires $\text{ceil}(\log_2(N))$ bits to index
- LOG2_CEIL template
 - Calculate $\text{ceil}(\log_2(N))$ on compile time

Binary to Onehot MUX Automatically

- The book mentioned:
 - Cascade structure is automatically optimized with onehot MUX
 - To get better performance
 - But larger area
- Not in Xilinx?

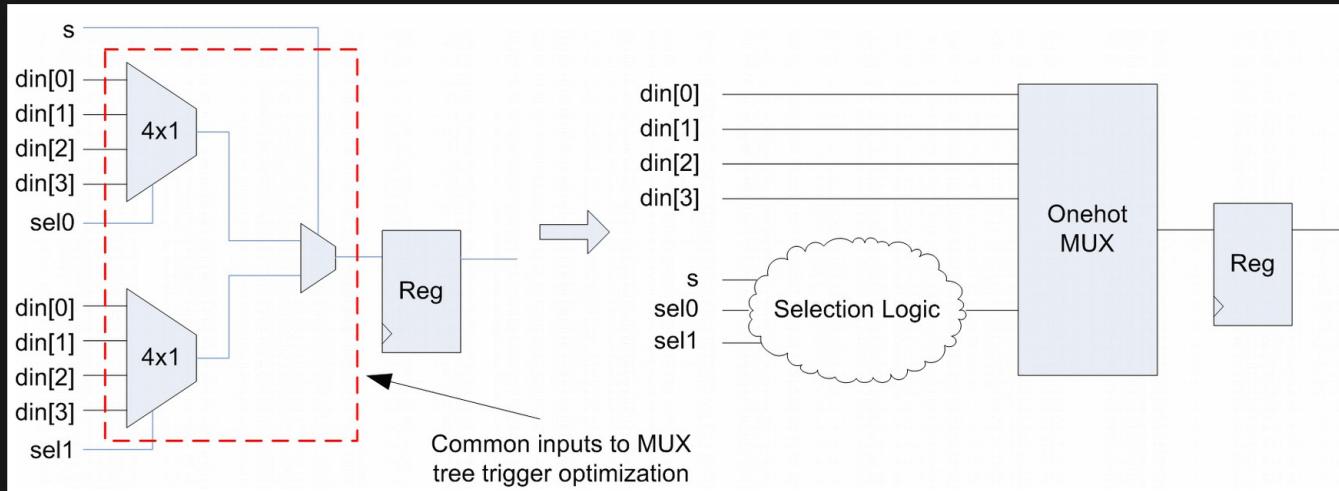


Binary to Onehot MUX Automatically



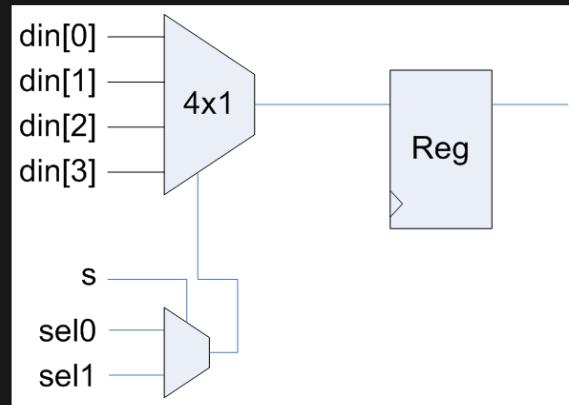
```
1 int mux_binary2onehot_opt(int din[4],  
2                           ap_uint<2> sel0,  
3                           ap_uint<2> sel1,  
4                           bool s) {  
5     int tmp;  
6     if (s)  
7         tmp = din[sel0];  
8     else  
9         tmp = din[sel1];  
10    return tmp;  
11 }
```

- Two 4x1 MUX is used



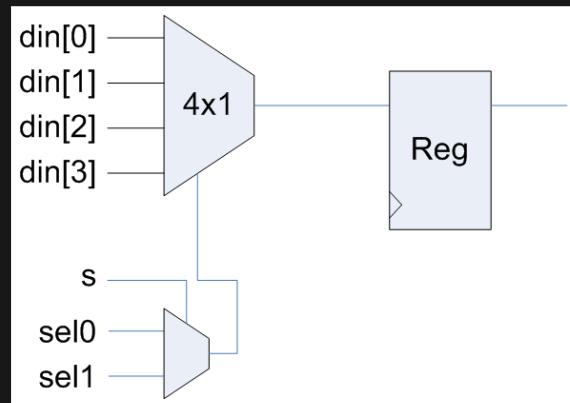
* Instance:						
Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
top_mux_83_8_1_1_U22	top_mux_83_8_1_1	0	0	0	45	0
top_mux_83_8_1_1_U23	top_mux_83_8_1_1	0	0	0	45	0
Total		0	0	0	90	0

Optimizing Binary MUX Manually



```
1 int mux_binary_opt(int din[4],  
2                     ap_uint<2> sel0,  
3                     ap_uint<2> sel1,  
4                     bool s)  
5 {  
6     int tmp;  
7     ap_uint<2> sel_int;  
8     if (s)  
9         sel_int = sel0;  
10    else  
11        sel_int = sel1;  
12    return din[sel_int];  
13 }
```

Optimizing Binary MUX Manually



* Instance:						
Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
top_mux_83_8_1_1_U9	top_mux_83_8_1_1	0	0	0	45	0
Total		0	0	0	45	0

* Expression:						
Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
tmp_fu_104_p9	select	0	0	3	1	3
Total		0	0	3	1	3

One Hot MUX

- Using “switch” statement
- Often controlled by data path FSM

```
int mux_onehot(int din[4], ap_uint<4> sel)
{
    int tmp;
    switch (sel) {
        case 1: tmp = din[0]; break; // 0001
        case 2: tmp = din[1]; break; // 0010
        case 4: tmp = din[2]; break; // 0100
        case 8: tmp = din[3]; break; // 1000
        default: tmp = 0; break;
    }
    return tmp;
}
```


* Expression:

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
icmp_ln7_1_fu_98_p2	icmp	0	0	11	8	7
icmp_ln7_2_fu_104_p2	icmp	0	0	11	8	6
icmp_ln7_3_fu_110_p2	icmp	0	0	11	8	5
icmp_ln7_4_fu_116_p2	icmp	0	0	11	8	4
icmp_ln7_5_fu_122_p2	icmp	0	0	11	8	3
icmp_ln7_6_fu_128_p2	icmp	0	0	11	8	2
icmp_ln7_7_fu_134_p2	icmp	0	0	11	8	1
icmp_ln7_fu_92_p2	icmp	0	0	13	8	9
or_ln7_1_fu_162_p2	or	0	0	2	1	1
or_ln7_2_fu_176_p2	or	0	0	2	1	1
or_ln7_3_fu_190_p2	or	0	0	2	1	1
or_ln7_4_fu_204_p2	or	0	0	2	1	1
or_ln7_5_fu_218_p2	or	0	0	2	1	1
or_ln7_6_fu_232_p2	or	0	0	2	1	1
or_ln7_fu_148_p2	or	0	0	2	1	1
ap_return	select	0	0	8	1	8
select_ln7_1_fu_154_p3	select	0	0	8	1	8
select_ln7_2_fu_168_p3	select	0	0	8	1	8
select_ln7_3_fu_182_p3	select	0	0	8	1	8
select_ln7_4_fu_196_p3	select	0	0	8	1	8
select_ln7_5_fu_210_p3	select	0	0	8	1	8
select_ln7_6_fu_224_p3	select	0	0	8	1	8
select_ln7_fu_140_p3	select	0	0	8	1	8
Total		0	0	168	79	108

One Hot MUX

- Using “if-else” statements

```
1  #include "mux.h"
2
3  int mux_onehot_if(int din[4], ap_uint<4> sel)
4  {
5      int tmp;
6      if (sel == 1) tmp = din[0]; // 0001
7      else if (sel == 2) tmp = din[1]; // 0010
8      else if (sel == 4) tmp = din[2]; // 0100
9      else if (sel == 8) tmp = din[3]; // 1000
10     else tmp = 0;
11     return tmp;
12 }
```

Priority Search Hardware

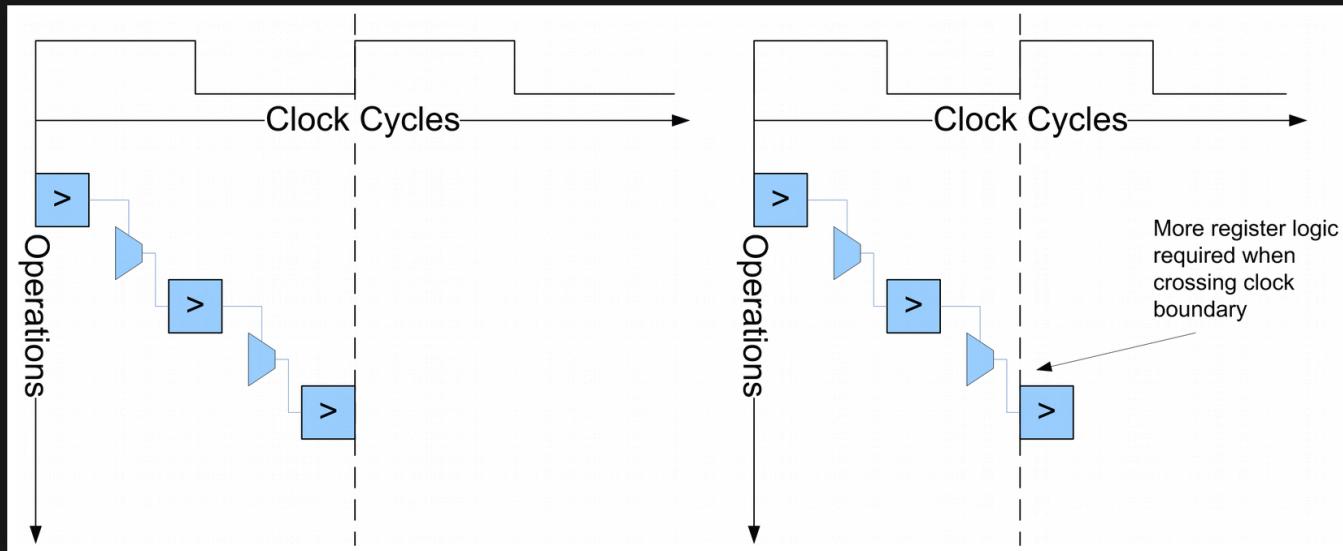
- For algorithms performed on an array of size N
 - if: software will do in $O(N)$
 - then: hardware may do in $O(\log N)$, saving time and area

Finding Array Maximum - Linear Search

```
void max_algorithmic(int din[8], int &dout)
{
    int max;
    int tmp;
    for (int i = 0; i < 8; i++) {
        #pragma HLS UNROLL
        if (i == 0)
            max = din[i];
        else {
            tmp = din[i];
            if (tmp > max)
                max = tmp;
        }
    }
    dout = max;
}
```

Finding Array Maximum - Linear Search

- Long delay
- May pipeline, but require more area



Divide & Conquer => Recursion?

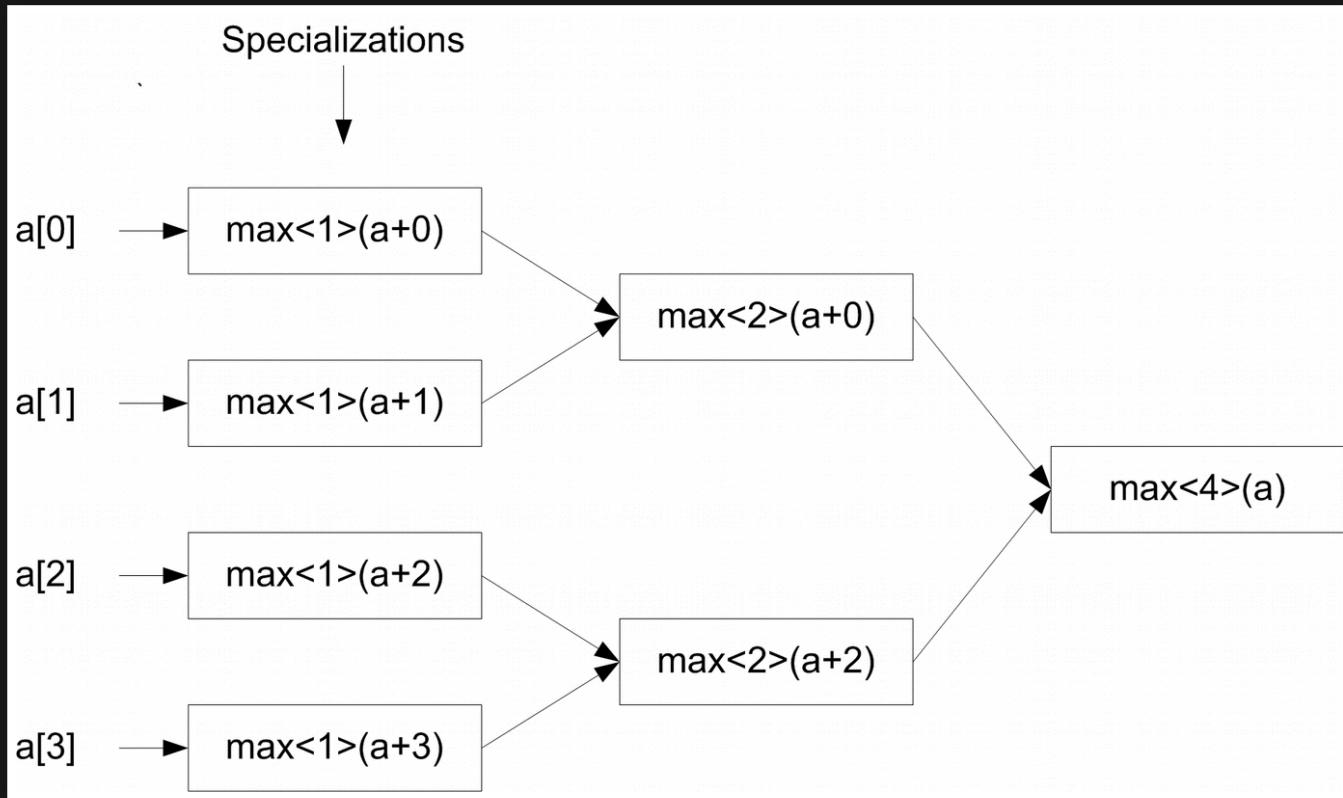
- Partially possible in HLS!
- Recursion pros:
 - Balanced hardware
- Recursion cons:
 - No trading-off about loop unrolling, since fully parallel
- Algorithms that is fully parallel often use recursion

Template Recursion

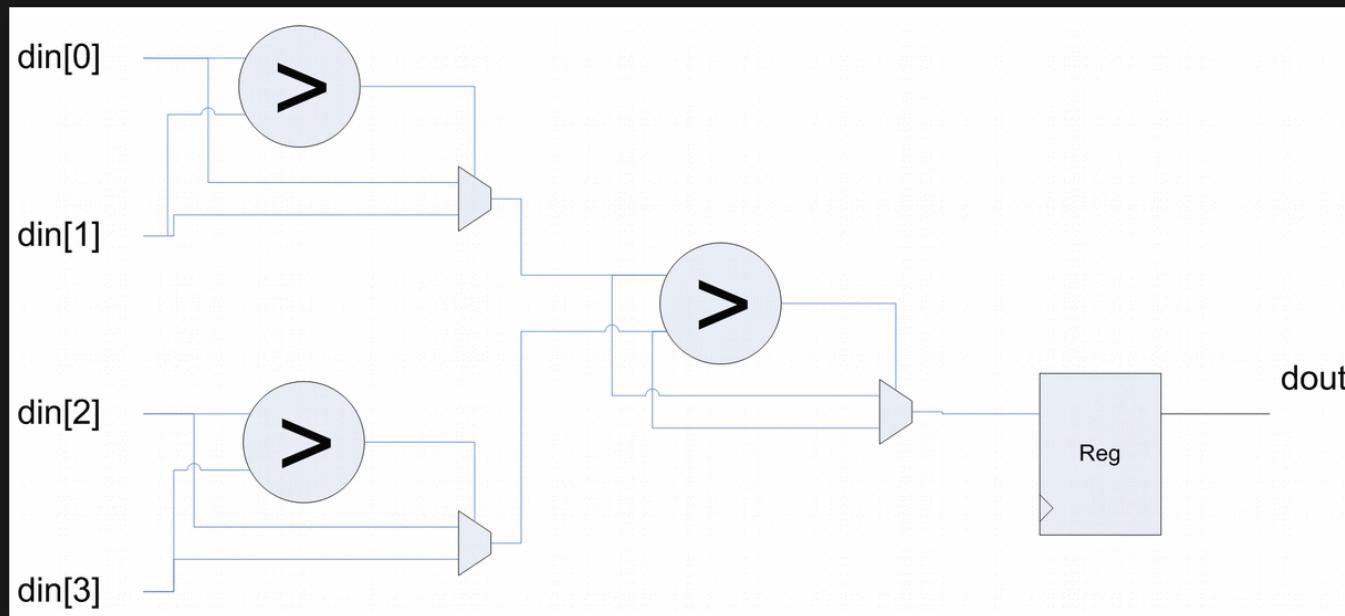
How is recursion achieved in HLS?

- Template recursion is resolved by compiler using **different function**
- Key is that a template argument must be known at compile time

```
1  template <int N>
2  int max(int a[]) {
3      int m0 = max<N / 2>(a);
4      int m1 = max<N - N / 2>(a + N / 2);
5      return m0 > m1 ? m0 : m1;
6  }
7  template <>
8  int max<1>(int a[]) { return a[0]; }
9
10 // When function calls:
11 max<4>(a);
12
13 // After compiler resolves template:
14 template <>
15 int max<4>(int a[]) { /* call max<2>(a) twice */ }
16
17 template <>
18 int max<2>(int x) { /* call max<1>(a) twice */ }
```



Finding Array Maximum - Divide and Conquer



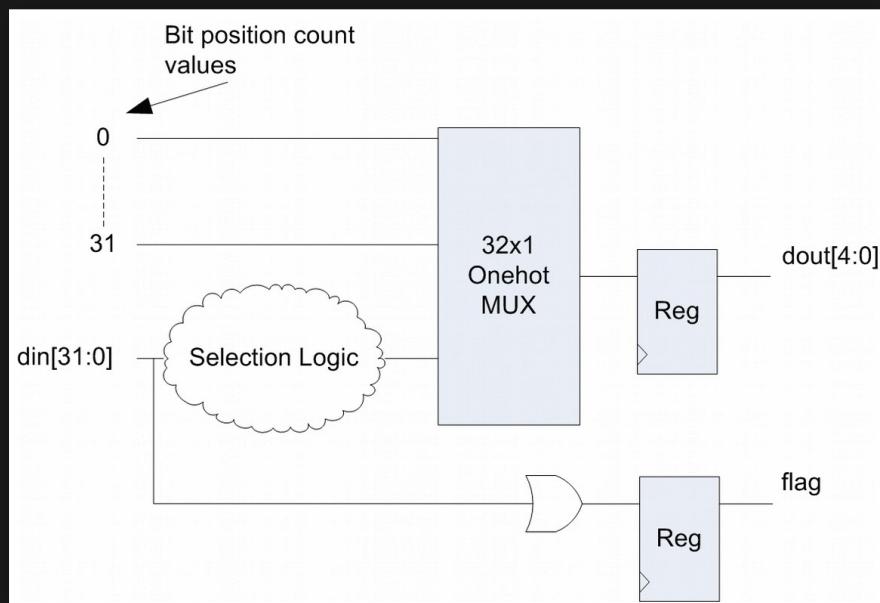
Finding Leading 1 - Linear Search

- Get position of the leading 1
- $0100 \Rightarrow 2$
- Note the UNROLL

```
1  bool leading_ones(ap_uint<32> din,
2                      ap_uint<LOG2_CEIL<32>::val> &dout)
3  {
4      int tmp = 0;
5      bool flag = false;
6      for (int i = 32 - 1; i >= 0; i--) {
7          #pragma HLS UNROLL
8          if (din[i]) {
9              flag = true;
10             tmp = i;
11             break;
12         }
13     }
14     dout = tmp;
15     return flag;
16 }
17 }
```

Finding Leading 1 - Linear Search

- A big MUX is synthesized after unrolling



* Multiplexer:

Name	LUT	Input Size	Bits	Total Bits
ap_NS_fsm	15	3	1	3
ap_return_0	9	2	1	2
ap_return_1	9	2	5	10
flag_0_reg_265	9	2	1	2
phi_ln301_reg_138	141	31	5	155
Total	183	40	13	172

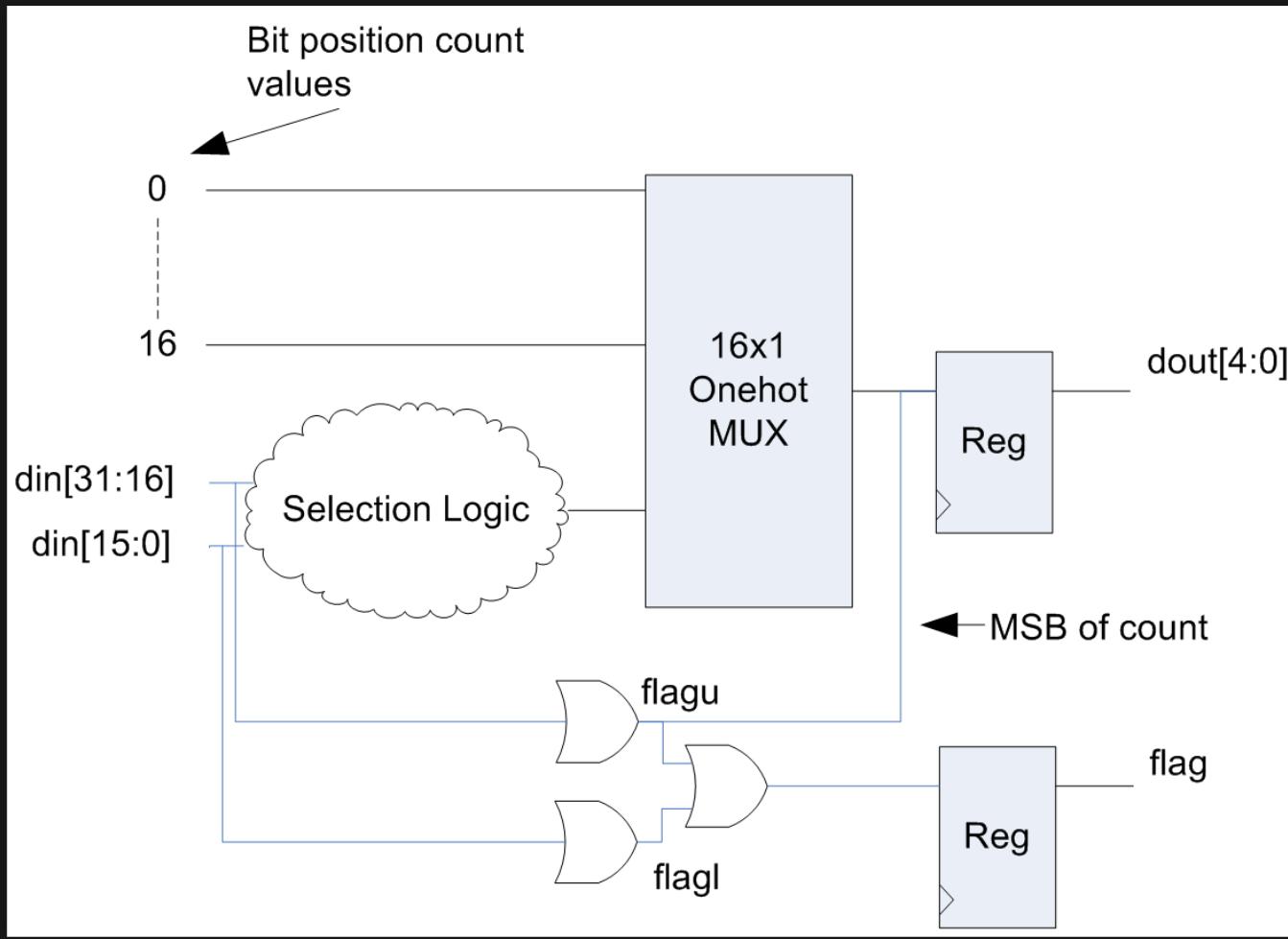
Finding Leading 1 - Brute Force

- Divide into 2 parts, save time

```
1 bool leading_ones_bruteforce(ap_int<32> din,
2                               ap_int<LOG2_CEIL<32>::val> &dout) {
3     int upper = 0, lower = 0;
4     bool flagu = false, flagl = false;
5     for (int i = 32 - 1; i >= 32 / 2; i--)
6         #pragma HLS UNROLL
7         if (din[i]) {
8             upper = i;
9             flagu = true;
10            break;
11        }
12    for (int i = 32 / 2 - 1; i >= 0; i--)
13        #pragma HLS UNROLL
14        if (din[i]) {
15            lower = i;
16            flagl = true;
17            break;
18        }
19    dout = flagu ? upper : lower;
20    return flagu | flagl;
21 }
```

Finding Leading 1 - Brute Force

- 32x1 MUX replaced with a 16x1 MUX (in Xilinx: two 16x1 MUX)
- MSB of output bit position is determined by flag u
- Require padding if #bits not power of 2



* Multiplexer:					
	Name	LUT	Input Size	Bits	Total Bits
	ap_NS_fsm	15	3	1	3
	ap_return_0	9	2	1	2
	ap_return_1	9	2	5	10
	flag_0_reg_265	9	2	1	2
	phi_ln301_reg_138	141	31	5	155
	Total	183	40	13	172

* Multiplexer:					
	Name	LUT	Input Size	Bits	Total Bits
	ap_NS_fsm	21	4	1	4
	ap_return_0	9	2	1	2
	ap_return_1	9	2	5	10
	flagl_0_reg_325	9	2	1	2
	flagu_0_reg_208	9	2	1	2
	phi_ln26_1_reg_262	62	15	4	60
	phi_ln26_reg_140	65	16	5	80
	Total	184	43	18	160

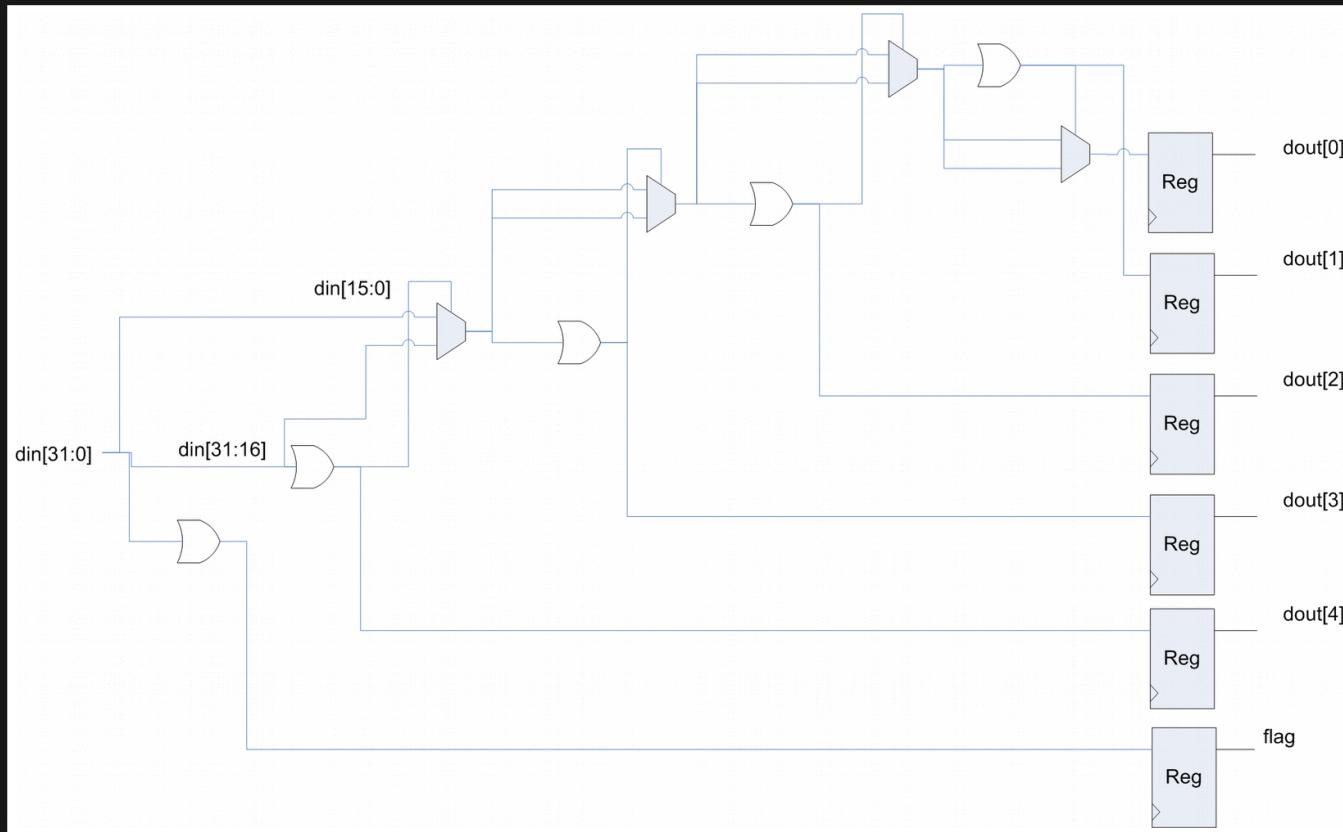
Log2(N) Based Search Leading 1

- Divide & Conquer Using loop

```
1 // leading_ones_log2.cpp
2 bool leading_ones_log2(ap_uint<32> din,
3                         ap_uint<LOG2_CEIL<32>::val> &dout)
4 {
5     enum { P2 = NEXT_POW2<32>::val };
6     enum { L2 = LOG2_CEIL<32>::val };
7     ap_uint<P2> upper, lower;
8     ap_uint<P2> mask = 0;
9     ap_uint<P2> din_tmp = 0; // pad din to power of 2
10    bool flag = false;
11    int idx = 0;
12    din_tmp = din;
13    mask = ~mask;
14    flag = (din_tmp != 0) ? 1 : 0;
```

```
1     for (int i = 0; i < L2; i++) {
2         #pragma HLS UNROLL
3         mask = mask >> ((P2 / 2) >> i);
4         upper = lower = 0;
5         upper = din_tmp >> ((P2 / 2) >> (i));
6         lower = din_tmp & mask;
7         din_tmp = 0;
8         if (upper != 0) {
9             idx = idx + ((P2 / 2) >> i);
10            din_tmp = upper;
11        } else
12            din_tmp = lower;
13    }
14    dout = idx;
15
16    return flag;
17 }
```

Log2(N) Based Search Leading 1



Recursive Search Leading 1

- Recursive template

```
// leading_ones_tmpl.h
template <int N_BITS>
bool leading_ones(ap_uint<N_BITS> &din,
                  ap_uint<LOG2_CEIL<N_BITS>::val> &dout)
{
    enum { P2 = NEXT_POW2<(N_BITS + 1) / 2>::val };
    ap_uint<N_BITS - P2> upper;
    ap_uint<P2> lower;
    ap_uint<LOG2_CEIL<N_BITS>::val> idx = 0;
    ap_uint<LOG2_CEIL<N_BITS - P2>::val> idxu = 0;
    ap_uint<LOG2_CEIL<P2>::val> idxl = 0;
    static bool flag = false;
```

```

upper = din >> P2;
lower = din;
flag = (din != 0) ? 1 : 0;
if (upper) {
    leading_ones<N_BITS - P2>(upper, idxu);
    idx = idxu | P2;
} else {
    leading_ones<P2>(lower, idxl);
    idx = idxl;
}
dout = idx;
return flag;
}

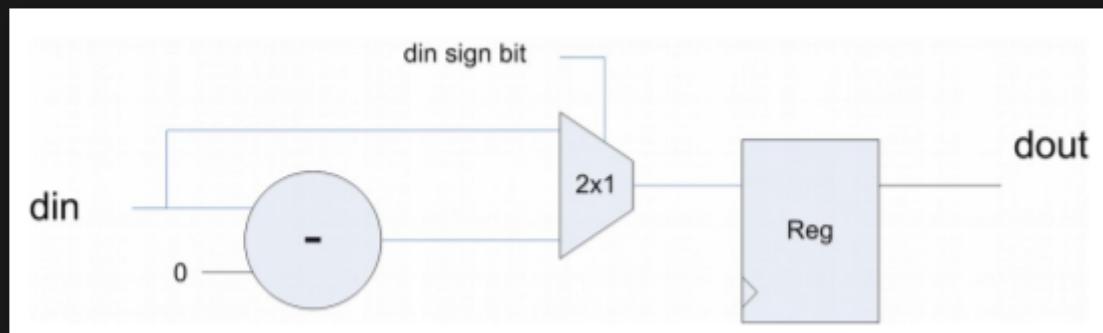
template <>
bool leading_ones<1>(ap_uint<1> &din, ap_uint<1> &dout)
{
    dout = 0;
    return din[0];
}
#endif

```


Abs

Naive abs

```
1 ap_int<8> abs_algorithmic(ap_int<8> din) {  
2     ap_int<8> tmp = din;  
3     if (tmp<0)  
4         tmp = -tmp;  
5     return tmp;  
6 }
```



Naive abs (cont.)

- one Mux, one Sub

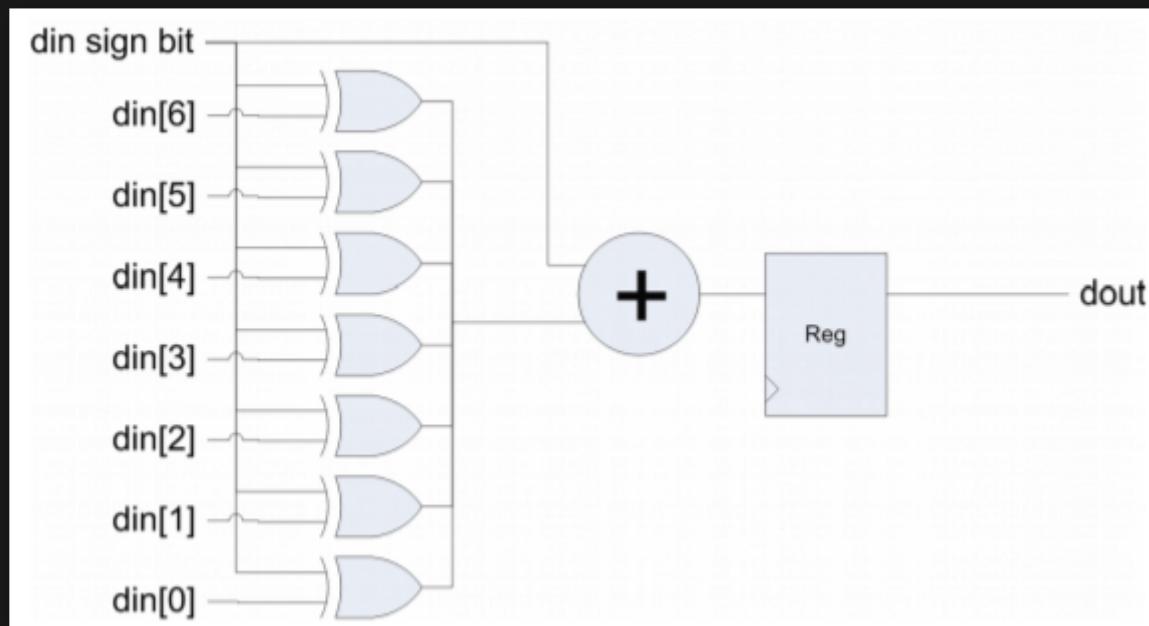
Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	3.163 ns	0.63 ns

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
tmp_V_fu_30_p2	-	0	0	15	1	8
ap_return	select	0	0	8	1	8
Total		0	0	23	2	16

Better abs

- Remove “select” operator. “Can help reduce area by as much as 10 to 20 percent in some instances.”

```
1 ap_int<8> abs_algorithmic(ap_int<8> din) {  
2     ap_int<8> tmp0=0, tmp1=0;  
3     tmp0 = din;  
4     for (int i=0; i<8; i++)  
5         #pragma HLS UNROLL  
6         tmp1[i] = tmp0[i]^tmp0[7];  
7     return tmp1+tmp0[7];  
8 }
```



Better abs (cont.)

- Faster
- Select -> Xor
- Subtractor -> Adder
- Note: Use much more LUT!?

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	1.915 ns	0.63 ns

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
ap_return	+	0	0	15	8	8
xor_ln816_1_fu_74_p2	xor	0	0	2	1	1
xor_ln816_2_fu_88_p2	xor	0	0	2	1	1
xor_ln816_3_fu_102_p2	xor	0	0	2	1	1
xor_ln816_4_fu_116_p2	xor	0	0	2	1	1
xor_ln816_5_fu_130_p2	xor	0	0	2	1	1
xor_ln816_6_fu_144_p2	xor	0	0	2	1	1
xor_ln816_7_fu_158_p2	xor	0	0	2	1	1
xor_ln816_fu_60_p2	xor	0	0	2	1	1
Total		0	0	31	16	16

Xilinx version

- Almost the same as naive version

```
// hls_math.h
template<int I>
ap_int<I> abs(ap_int<I> x) {
    return fabs_fixed(x);
}
```

```
// hls_round_copysign_apfixed.h
template <int I>
ap_int<I> fabs_fixed(ap_int<I> x)
{
    ap_int<I> xs      = -x;
    xs[I_-1] = 0;
    return ( ( x[I_-1] ) ? xs : x );
}
```

Xilinx version (cont.)

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	3.163 ns	0.63 ns

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
tmp_V_fu_30_p2	-	0	0	15	1	8
ap_return	select	0	0	8	1	8
Total		0	0	23	2	16

A little difference between Naive and Xilinx version

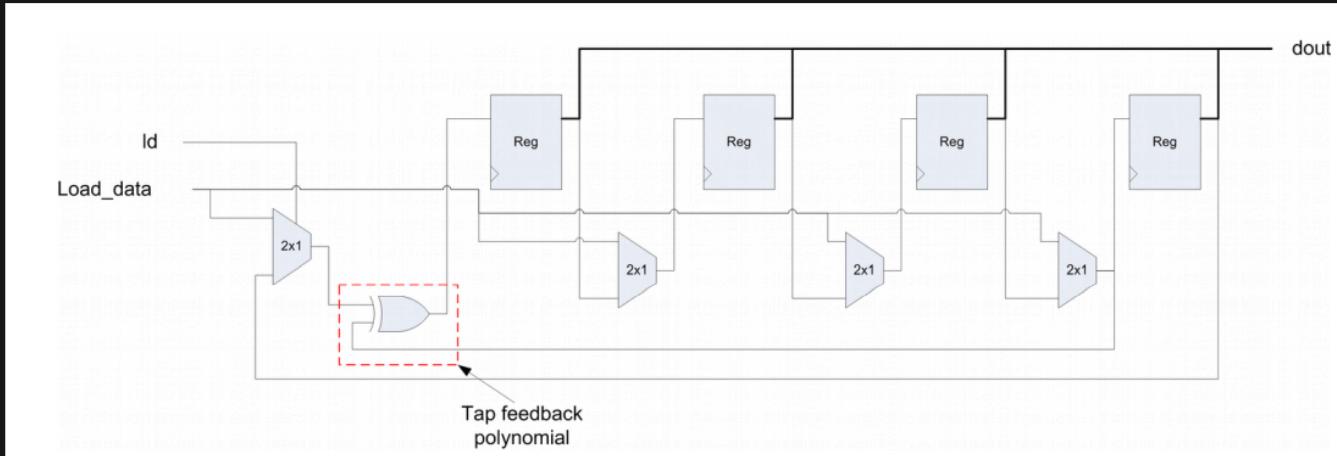
- Think about the "-128" on int8 case

```
1 // Naive
2 ABSTYPE abs_naive(ABSTYPE din) {
3     ABSTYPE tmp = din;
4     if (tmp<0)
5         tmp = -tmp;
6     return tmp;
7 }
```

```
1 // Xilinx
2 ABSTYPE abs_xilinx(ABSTYPE din) {
3     ABSTYPE xs      = -din;
4     xs[8-1] = 0;
5     return ( (din[8-1]) ? xs : din );
6 }
```

Linear Feedback Shift Register (LFSR)

LFSR



```
1 void lfsr (ap_uint<8> load_data,
2             bool ld,
3             ap_uint<8> &dout) {
4     static ap_uint<8> reg;
5     ap_uint<1> bit;
6
7     if (ld) {
8         reg = load_data;
9     } else {
10        bit = reg[3] ^ reg[2];
11        reg<<=1;
12        reg[0] = bit;
13    }
14    dout = reg;
15 }
```

LFSR Utilization

* Expression:

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
dout_V	select	0	0	8	1	8
bit_V_fu_67_p2	xor	0	0	2	1	1
Total		0	0	10	2	9

* Register:

Name	FF	LUT	Bits	Const Bits
ap_CS_fsm	1	0	1	0
reg_V	8	0	8	0
Total	9	0	9	0

GitHub Repo

- <https://github.com/soyccan/HLS-Bluebook>

Reference

- Xilinx HLS IP Libraries:
https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/hls_ip_libraries.html
- Xilinx HLS Math Libraries:
https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/vitis_hls_math.html