

## Parallel and Reconfigurable VLSI Computing (10)

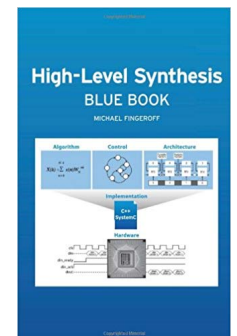
# HLS Optimizations

Hiroki Nakahara

Tokyo Institute of Technology

### References:

- [1] Micheal Fingeroff, "High-Level Synthesis Blue Book," Xlibris, 2010.
- [2] Ryan Kastner, Janarbek Matai, Stephen Neuendorffer, "Parallel Programming for FPGAs," arXiv:1805.03648, 2018.  
<https://arxiv.org/abs/1805.03648>



# Outline

- HLS Optimizations through FIR Design
- Code reconstruction
  - Useful coding guideline for debugging
  - Performance analysis on LLVM-IR
  - Area-time trade-off
  - Code hoisting
  - Loop fission
  - Loop unrolling
  - Array partition
  - Loop pipelining
  - Bitwidth optimization

# HLS Optimizations through FIR Design

# FIR Filter Background

- $x_n$ : N sampling signals and  $y_n$ : Output signal, then

$$y_n = \sum_{k=0}^{N-1} h_k x_{n-k}$$

, where a filter coefficient  $h_n$  is given by

$$h_n = \frac{\rho_n}{2\pi} \int_0^{2\pi} d\tilde{\omega} e^{i\tilde{\omega}(n-\tilde{\tau})} H_0(\tilde{\omega}).$$

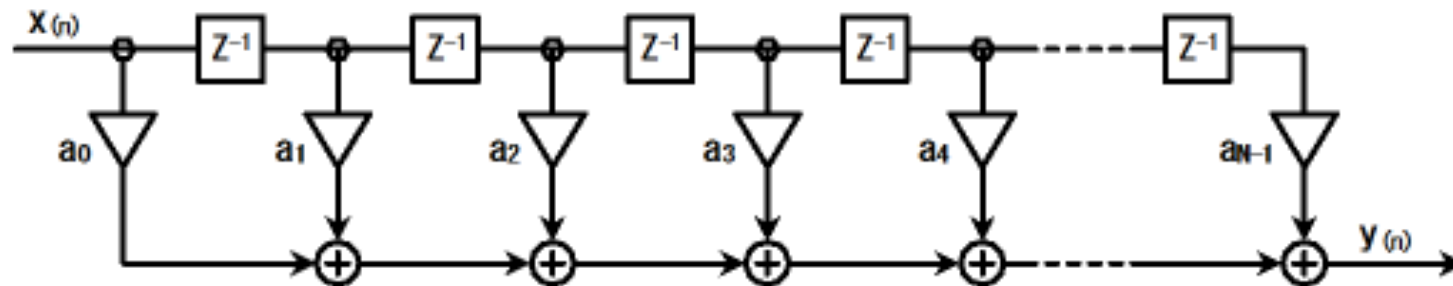
, where  $\tilde{\omega} = 2\pi\omega/\omega_s$  denotes normalized freq.,  $\omega_s$  denotes sampling freq.,  $H_0(\tilde{\omega}) \in \mathbb{R}$  denotes frequency characteristic,  $\rho_n$  denotes window function, and  $\tilde{\Gamma} = (N-1)/2$ .

# Cont'd

- Deference equation for a FIR filter:

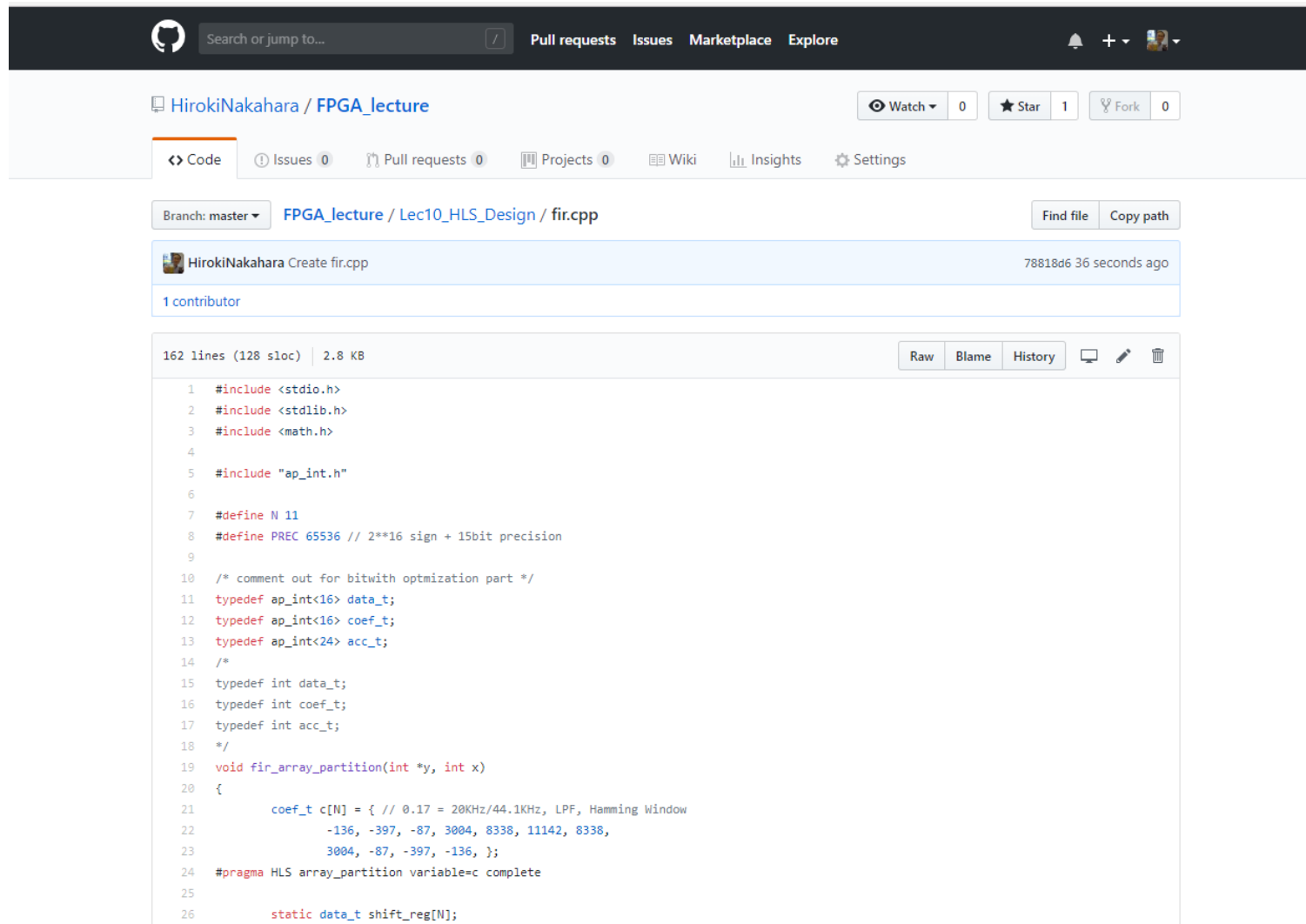
$$y[n] = a_0x[n] + a_1x[n-1] + a_2x[n-2] + a_3x[n-3] + a_4x[n-4] + \dots + a_{N-1}x[n-(N-1)]$$

- Diagram for a FIR filter:



# C++ Behavior for a FIR Filter

[https://github.com/HirokiNakahara/FPGA\\_lecture/tree/master/Lec10\\_HLS\\_Design/fir.cpp](https://github.com/HirokiNakahara/FPGA_lecture/tree/master/Lec10_HLS_Design/fir.cpp)



The screenshot shows the GitHub interface for the repository 'HirokiNakahara / FPGA\_lecture'. The file 'fir.cpp' is selected, showing its commit history and code. The code is a C++ implementation of a FIR filter, featuring includes for `<stdio.h>`, `<stdlib.h>`, and `<math.h>`, and a custom header `"ap_int.h"`. It defines constants `N` (11) and `PREC` (65536), and uses typedefs for `data_t`, `coef_t`, and `acc_t`. The function `fir_array_partition` is defined, which takes an integer `y` and `x` and returns a pointer to a static array `shift_reg` of type `data_t`.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  #include "ap_int.h"
6
7  #define N 11
8  #define PREC 65536 // 2**16 sign + 15bit precision
9
10 /* comment out for bitwith optimization part */
11 typedef ap_int<16> data_t;
12 typedef ap_int<16> coef_t;
13 typedef ap_int<24> acc_t;
14 /*
15 typedef int data_t;
16 typedef int coef_t;
17 typedef int acc_t;
18 */
19 void fir_array_partition(int *y, int x)
20 {
21     coef_t c[N] = { // 0.17 = 20KHz/44.1KHz, LPF, Hamming Window
22         -136, -397, -87, 3004, 8338, 11142, 8338,
23         3004, -87, -397, -136, };
24     #pragma HLS array_partition variable=c complete
25
26     static data_t shift_reg[N];
```

# Code Reconstruction

- Writing highly optimized synthesizable HLS code is often not a straightforward process.
- It involves a deep understanding of the application at hand, the ability to change the code such that the Vivado HLS tool creates optimized hardware structures and utilizes the directives in an effective manner
  - FSM-based RTL design experience will help to understand

# Convert to Fixed Point Precision

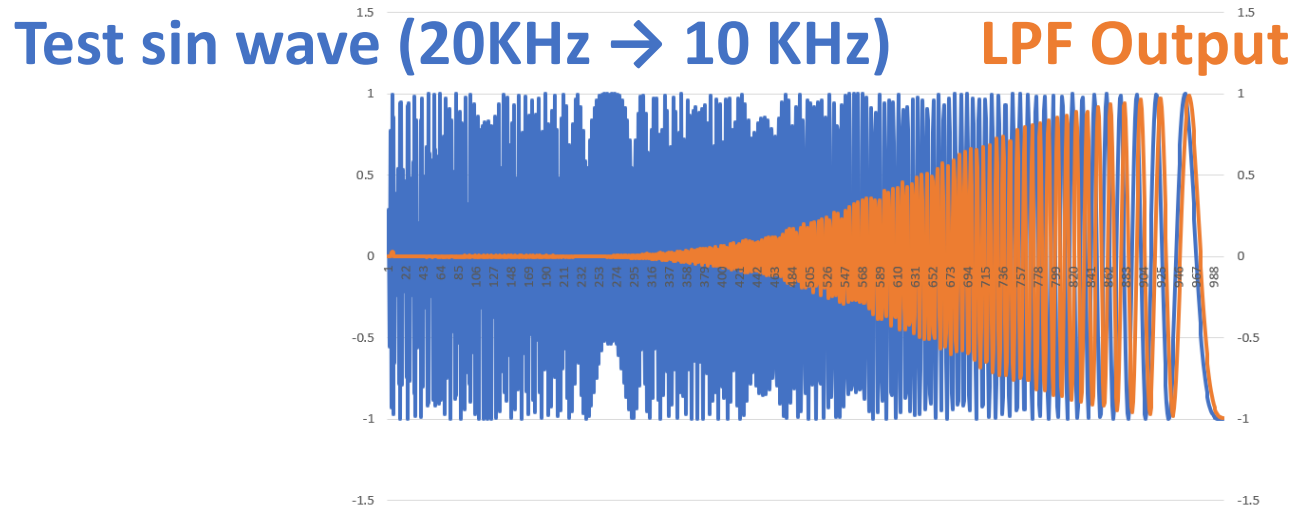
See, [https://github.com/HirokiNakahara/FPGA\\_lecture/tree/master/Lec7\\_Practical\\_RTL\\_design/fir\\_int.c](https://github.com/HirokiNakahara/FPGA_lecture/tree/master/Lec7_Practical_RTL_design/fir_int.c)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  #define N 11
6  #define PREC 65536 // 2**16 sign + 15bit precision
7
8  void fir(int *y, int x)
9  {
10     int c[N] = { // 0.17 = 20KHz/44.1KHz, LPF, Hamming Window
11                 -136, -397, -87, 3004, 8338, 11142, 8338,
12                 3004, -87, -397, -136, };
13
14     static int shift_reg[N];
15     int acc;
16     int i;
17
18     acc = 0;
19     for (i = N - 1; i >= 0; i--) {
20         if (i == 0) {
21             acc += x * c[0];
22             shift_reg[0] = x;
23         } else {
24             shift_reg[i] = shift_reg[i - 1];
25             acc += shift_reg[i] * c[i];
26         }
27     }
28     *y = acc;
29 }
30
31 void main()
32 {
33     float fs = 44100.0;
34     int len = 1000;
35
36     float f0 = 20000.0;
37     float sin_wave;
38     int fir_out;
39
40     int i;
41
42     for( i = 0; i < len; i++){
43         sin_wave = sin( 2.0 * M_PI * f0 * i / fs);
44
45         fir( &fir_out, (int)(sin_wave * PREC));
46
47         printf("%d %f %f\n", i, sin_wave, (float)fir_out / PREC);
48         f0 = f0 - 10.0;
49     }
50 }
```



# Debug for C Description

- Confirm the operation of FIR
- In/Out are reused as a testbench for HDL simulation
- Note, a parallel operation cannot be verified
- Area and speed of the circuit can not be estimated



# For Useful Coding

- Use typedef for different variables for changing the types of data (described later)
- Assign labels into loops for debugging

```
8 typedef int data_t;
9 typedef int coef_t;
10 typedef int acc_t;
11
12 void fir(int *y, int x)
13 {
14     coef_t c[N] = { // 0.17 = 20KHz/44.1KHz, LPF, Hamming Window
15                     -136, -397, -87, 3004, 8338, 11142, 8338,
16                     3004, -87, -397, -136, };
17
18     static data_t shift_reg[N];
19     acc_t acc;
20     int i;
21
22     acc = 0;
23     FIR_LOOP: for (i = N - 1; i >= 0; i--) {
24         if (i == 0) {
25             acc += x * c[0];
26             shift_reg[0] = x;
27         } else {
28             shift_reg[i] = shift_reg[i - 1];
29             acc += shift_reg[i] * c[i];
30         }
31     }
32     *y = acc;
33 }
```

# Performance Analysis on Vivado HLS

- Click "Analysis", right click on each block, then select "Goto Source"

The screenshot displays the Vivado HLS Performance Analysis tool. The top toolbar shows the 'Analysis' tab selected. The left pane shows the 'Current Module : fir' with a performance analysis table. The right pane shows a similar table with a right-click context menu open over the 'acc(phi mux)' operation at control step C1, with options 'Goto Source', 'Goto Verilog', and 'Goto VHDL'.

**Left Pane Performance Analysis Table:**

| Operation\Control Step  | C0 | C1 | C2 | C3 | C4 |
|-------------------------|----|----|----|----|----|
| 1 x read(read)          |    |    |    |    |    |
| 2 tmp(shl)              |    |    |    |    |    |
| 3 p neg(-)              |    |    |    |    |    |
| 4 tmp 7(shl)            |    |    |    |    |    |
| 5 tmp 2(-)              |    |    |    |    |    |
| 6 FIR LOOP              |    |    |    |    |    |
| 7 acc(phi mux)          |    |    |    |    |    |
| 8 i(phi mux)            |    |    |    |    |    |
| 9 tmp 1(icmp)           |    |    |    |    |    |
| 10 tmp 3(+)             |    |    |    |    |    |
| 11 shift reg load(read) |    |    |    |    |    |
| 12 c1 load(read)        |    |    |    |    |    |
| 13 node 39(write)       |    |    |    |    |    |
| 14 node 32(write)       |    |    |    |    |    |
| 15 tmp 6(*)             |    |    |    |    |    |
| 16 p pn(phi mux)        |    |    |    |    |    |
| 17 acc 1(+)             |    |    |    |    |    |
| 18 i 1(+)               |    |    |    |    |    |
| 19 node 47(write)       |    |    |    |    |    |

**Right Pane Performance Analysis Table:**

| Operation\Control Step  | C0 | C1 | C2 | C3 |
|-------------------------|----|----|----|----|
| 1 x read(read)          |    |    |    |    |
| 2 tmp(shl)              |    |    |    |    |
| 3 p neg(-)              |    |    |    |    |
| 4 tmp 7(shl)            |    |    |    |    |
| 5 tmp 2(-)              |    |    |    |    |
| 6 FIR LOOP              |    |    |    |    |
| 7 acc(phi mux)          |    |    |    |    |
| 8 i(phi mux)            |    |    |    |    |
| 9 tmp 1(icmp)           |    |    |    |    |
| 10 tmp 3(+)             |    |    |    |    |
| 11 shift reg load(read) |    |    |    |    |
| 12 c1 load(read)        |    |    |    |    |
| 13 node 39(write)       |    |    |    |    |
| 14 node 32(write)       |    |    |    |    |
| 15 tmp 6(*)             |    |    |    |    |

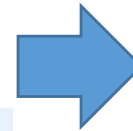
# Parallel Computation Manner

- As same as the RTL design, independent operations are executed in parallel

```

main.cpp x Synthesis(solution1) Performance(solution1)
1 void test( int a, int b, int c, int *x, int *y, int *z)
2 {
3     *x = a + b;
4     *y = b * c;
5     *z = c + a - b;
6 }
7

```

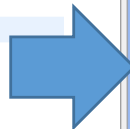


| Current Module : test |                  |    |
|-----------------------|------------------|----|
|                       | Operation\Con... | C0 |
| 1                     | c read(read)     |    |
| 2                     | b read(read)     |    |
| 3                     | a read(read)     |    |
| 4                     | x assign(+)      |    |
| 5                     | node 18(wr...    |    |
| 6                     | y assign(*)      |    |
| 7                     | node 20(wr...    |    |
| 8                     | tmp (-)          |    |
| 9                     | tmp 1(+)         |    |
| 10                    | node 23(wr...    |    |

```

8 void loop_test( int x[100], int y[100], int z[100], int a[30], int b[30], int c[30])
9 {
10     int i, j;
11
12     LOOP1: for( i = 0; i < 100; i++)
13         z[i] = x[i] * y[i];
14
15     LOOP2: for( j = 0; j < 30; j++)
16         c[i] = a[i] * b[i];
17 }
18
19

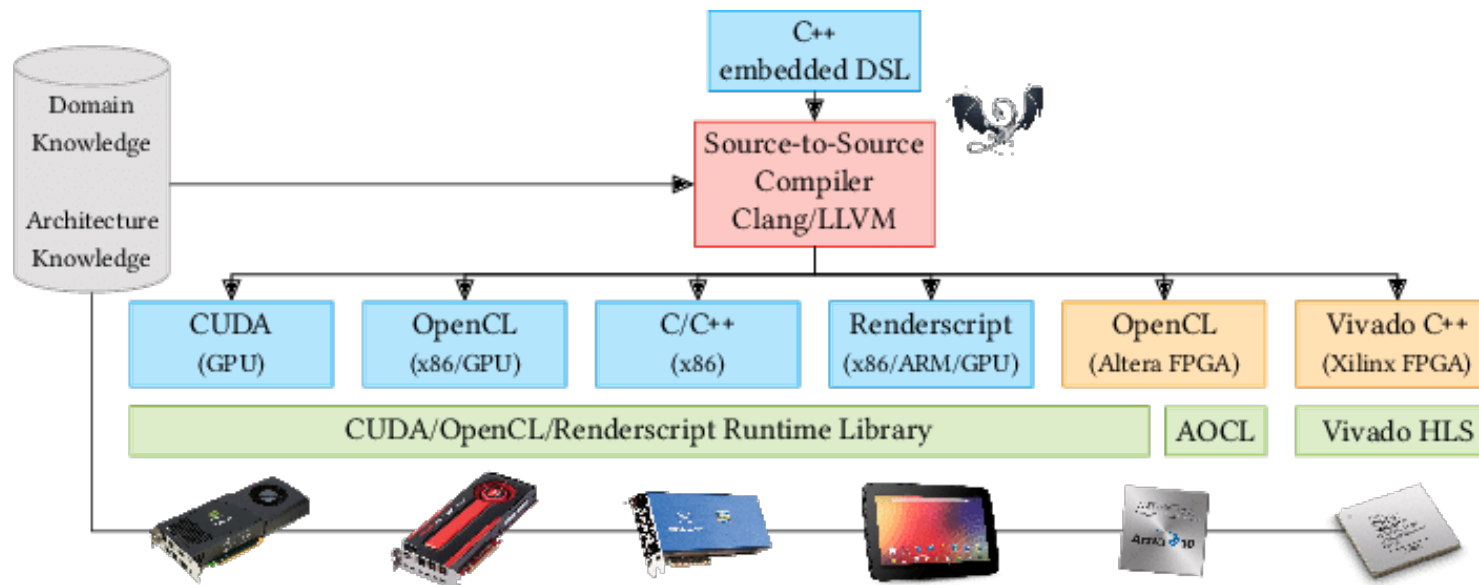
```



|    | Operation\Control... | C0 | C1 | C2 | C3 | C4 | C5 |
|----|----------------------|----|----|----|----|----|----|
| 1  | =LOOP1               |    |    |    |    |    |    |
| 2  | i(phi mux)           |    |    |    |    |    |    |
| 3  | exitcond1(icmp)      |    |    |    |    |    |    |
| 4  | tmp 2(+)             |    |    |    |    |    |    |
| 5  | x load(read)         |    |    |    |    |    |    |
| 6  | y load(read)         |    |    |    |    |    |    |
| 7  | tmp 1(*)             |    |    |    |    |    |    |
| 8  | node 30(write)       |    |    |    |    |    |    |
| 9  | =LOOP2               |    |    |    |    |    |    |
| 10 | j(phi mux)           |    |    |    |    |    |    |
| 11 | exitcond1(icmp)      |    |    |    |    |    |    |
| 12 | j 1(+)               |    |    |    |    |    |    |
| 13 | a load(read)         |    |    |    |    |    |    |
| 14 | b load(read)         |    |    |    |    |    |    |
| 15 | tmp 3(*)             |    |    |    |    |    |    |
| 16 | node 48(write)       |    |    |    |    |    |    |

# Low Level Virtual Machine (LLVM)

- Modularized, reusable compiler and toolchain technology
- Front end of C, C ++, Objective-C etc.
- Convert to LLVM-IR (Internal Representation)
- Then, optimized for Hardware (FPGA)
- Rust, Clang, LDC, Vivado HLS, Intel OpenCL

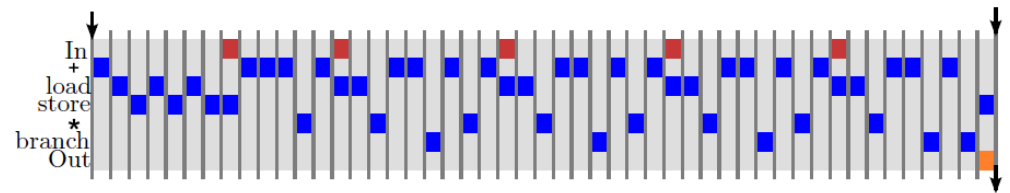


# LLVM-IR

## Example: FIR Filter

```
fir:
.frame r1,0,r15 # vars= 0, regs= 0, args= 0
.mask 0x00000000
addik r3,r0,delay_line.1450
lwi r4,r3,8 # Unrolled loop to shift the delay line
swi r4,r3,12
lwi r4,r3,4
swi r4,r3,8
lwi r4,r3,0
swi r4,r3,4
swi r5,r3,0 # Store the new input sample into the delay line
addik r5,r0,4 # Initialize the loop counter
addk r8,r0,r0 # Initialize accumulator to zero
addk r4,r8,r0 # Initialize index expression to zero
$L2:
mul r3,r4,4 # Compute a byte offset into the delay_line array
addik r9,r3,delay_line.1450
lw r3,r3,r7 # Load filter tap
lwi r9,r9,0 # Load value from delay line
mul r3,r3,r9 # Filter Multiply
addk r8,r8,r3 # Filter Accumulate
addik r5,r5,-1 # update the loop counter
bneid r5,$L2
addik r4,r4,1 # branch delay slot, update index expression

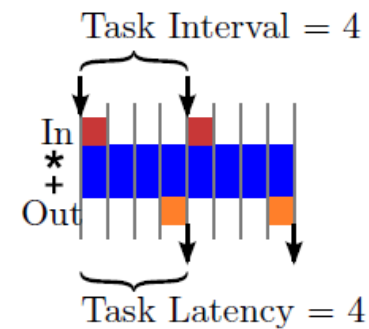
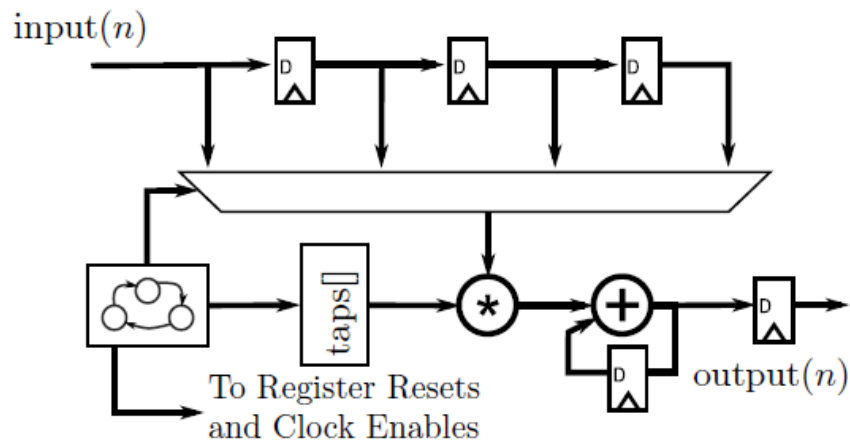
rtsd r15, 8
swi r8,r6,0 # branch delay slot, store the output
.end fir
```



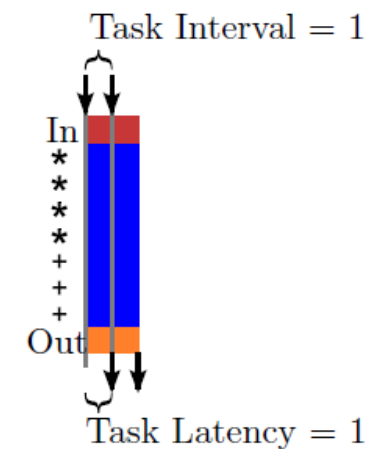
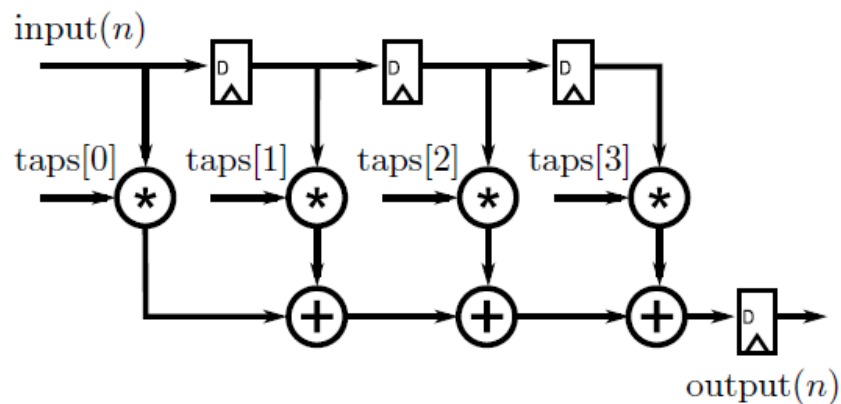
This code is generated using  
microblazeel-xilinx-linux-gnu-gcc -O1  
-mno-xl-soft-mul -S fir.c

# Different Architectures

- Sequential manner

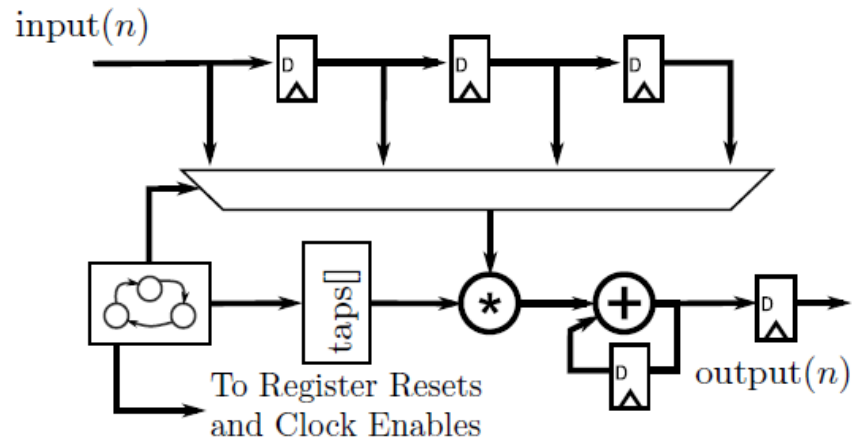


- Pipeline manner

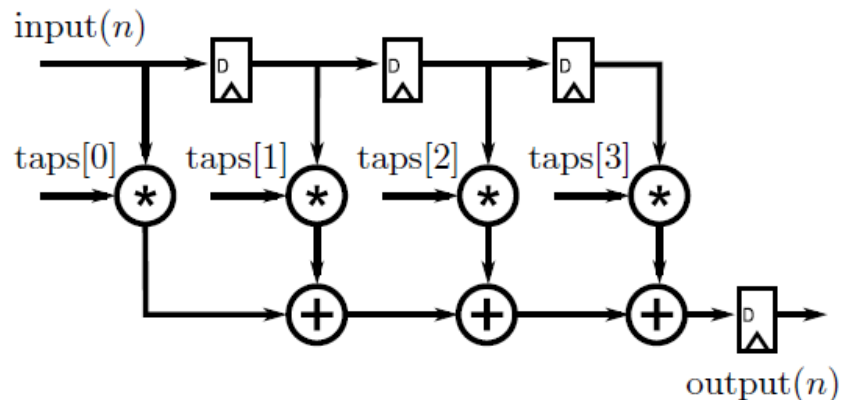


# Area-Time Trade-off

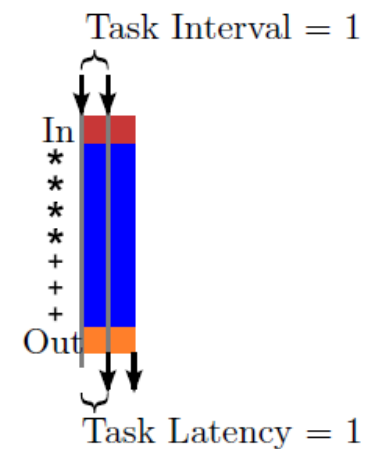
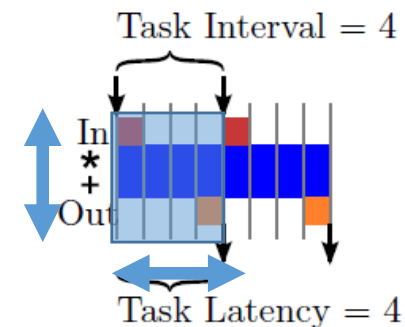
- Sequential manner



- Pipeline manner



Vertical (Area)  
Horizontal (Time)

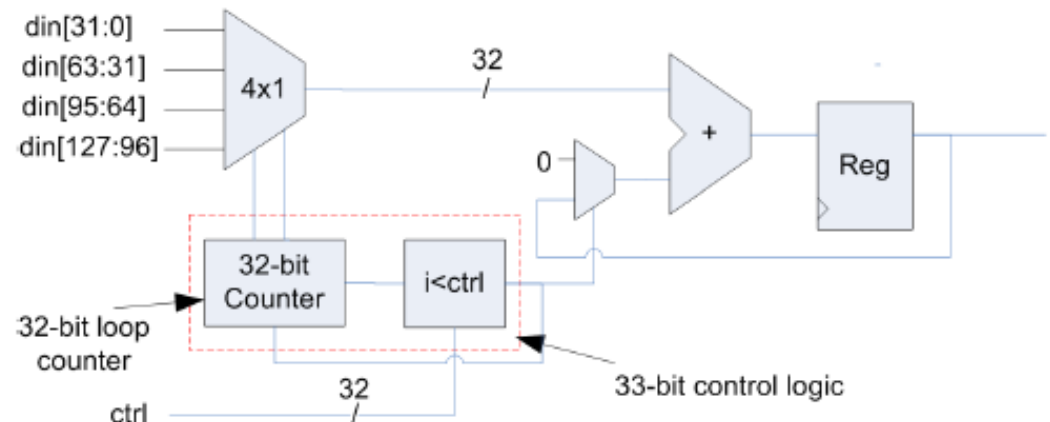




# Loop with Conditional Bounds

- Having a variable as the loop upper or lower bound often results in the loop counter hardware being larger than needed
  - Having an unconstrained bit width on the loop exit condition results in control logic larger than needed

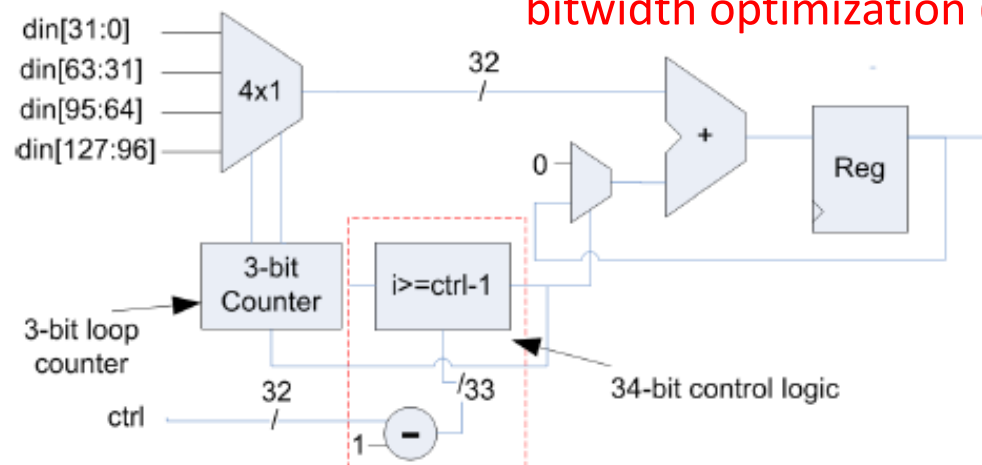
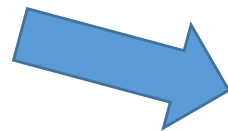
```
1  #include "accum.h"
2  #include <ac_int.h>
3  void accumulate(int din[4], int &dout, unsigned int ctrl){
4      int acc=0;
5      ACCUM:for(int i=0;i<ctrl;i++){
6          acc += din[i];
7      }
8      dout = acc;
9  }
10
```



# Optimizing the Loop Counter

- In order for HLS to reduce the bit width of the loop counter the loop upper bound should be set to a constant
- However, since the execution of each loop iteration is determined by the variable, "ctrl"
- It is done by using a conditional break in the loop body

```
1  #include "accum.h"
2  #include <ac_int.h>
3  void accumulate(int din[4], int &dout, int ctrl){
4      int acc=0;
5      ACCUM:for(int i=0;i<4;i++){
6          acc += din[i];
7          if(i>=ctrl-1)
8              break;
9      }
10     dout = acc;
11 }
```

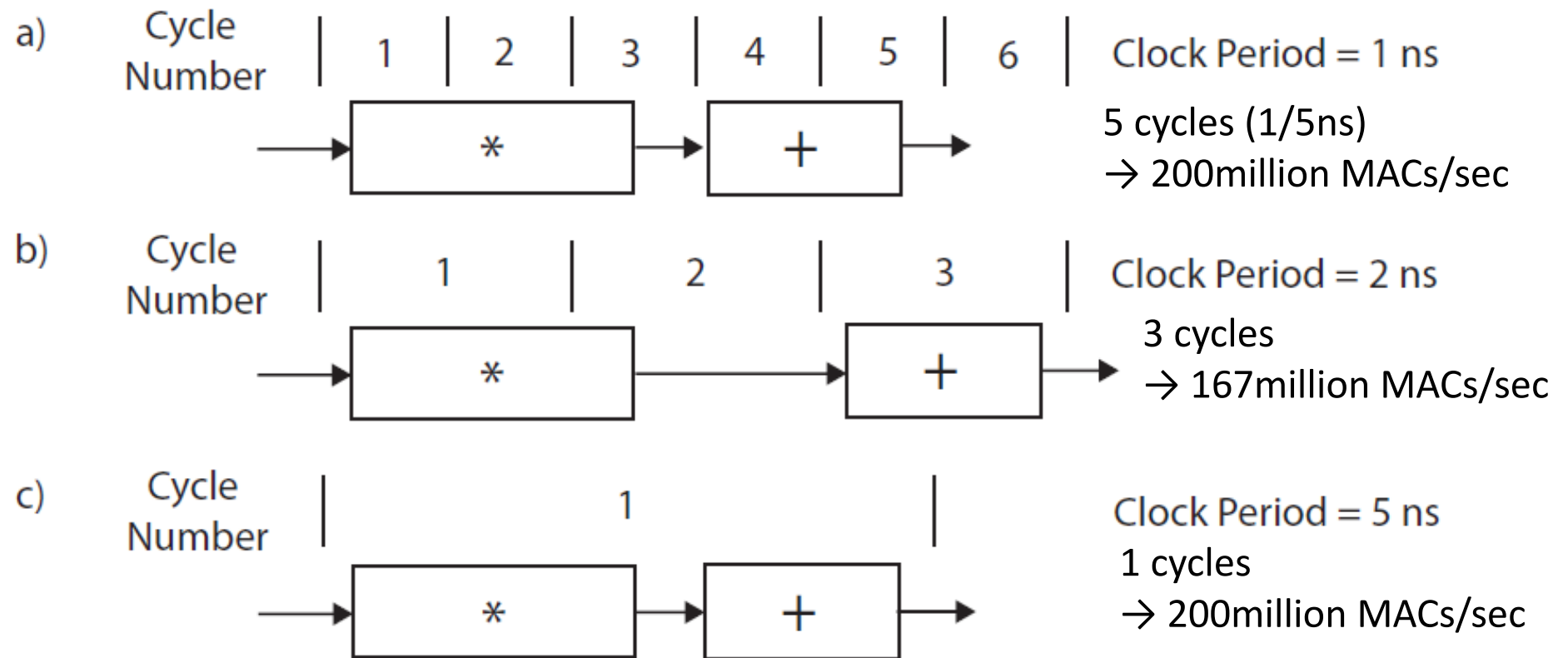


# Calculating Performance

- Necessary to define precise metrics
- What is "fast" design?
  - Efficiency?
    - operations/sec
    - MACs/sec
    - bits/sec
  - Latency? Throughput? Computation time?
- High-level synthesis tools talk about the designs in terms of number of cycles, and the frequency of the clock
- Select adequate measurement of a target application
- Compare them using the same metric

# Operation Chaining

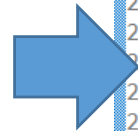
- Consider the multiply accumulate operation that is done in a FIR filter tap
- Assume that the add operation takes 2 ns to complete, and a multiply operation takes 3 ns



# Code Hoisting

- The if/else statement inside of the for loop is inefficient.
- For every control structure in the code, the Vivado HLS tool creates logical hardware that checks if the condition is met
- Therefore, the statements within the if branch can be "hoisted" out of the loop

```
35 void fir(int *y, int x)
36 {
37     coef_t c[N] = { // 0.17 = 20KHz/44.1KHz, LPF, Hamming Window
38         -136, -397, -87, 3004, 8338, 11142, 8338,
39         3004, -87, -397, -136, };
40
41     static data_t shift_reg[N];
42     acc_t acc;
43     int i;
44
45     acc = 0;
46     FIR_LOOP: for (i = N - 1; i >= 0; i--) {
47         if (i == 0) {
48             acc += x * c[0];
49             shift_reg[0] = x;
50         } else {
51             shift_reg[i] = shift_reg[i - 1];
52             acc += shift_reg[i] * c[i];
53         }
54     }
55     *y = acc;
56 }
```



```
12 void fir_hoisting(int *y, int x)
13 {
14     coef_t c[N] = { // 0.17 = 20KHz/44.1KHz, LPF, Hamming Window
15         -136, -397, -87, 3004, 8338, 11142, 8338,
16         3004, -87, -397, -136, };
17
18     static data_t shift_reg[N];
19     acc_t acc;
20     int i;
21
22     acc = 0;
23     FIR_LOOP[NOIF]: for (i = N - 1; i > 0; i--) {
24         shift_reg[i] = shift_reg[i - 1];
25         acc += shift_reg[i] * c[i];
26     }
27
28     acc += x * c[0];
29     shift_reg[0] = x;
30
31     *y = acc;
32 }
```

# Comparison

- Original

- ▢ Timing (ns)

- ▢ Summary

| Clock  | Target | Estimated | Uncertainty |
|--------|--------|-----------|-------------|
| ap_clk | 10.00  | 8.51      | 1.25        |

- ▢ Latency (clock cycles)

- ▢ Summary

| Latency |     | Interval |     | Type |
|---------|-----|----------|-----|------|
| min     | max | min      | max |      |
| 23      | 45  | 23       | 45  | none |

- ▢ Detail

- ⊕ Instance

- ⊕ Loop

## Utilization Estimates

- ▢ Summary

| Name            | BRAM_18K | DSP48E | FF    | LUT   |
|-----------------|----------|--------|-------|-------|
| DSP             | -        | -      | -     | -     |
| Expression      | -        | 3      | 0     | 149   |
| FIFO            | -        | -      | -     | -     |
| Instance        | -        | -      | -     | -     |
| Memory          | 0        | -      | 79    | 9     |
| Multiplexer     | -        | -      | -     | 120   |
| Register        | -        | -      | 215   | -     |
| Total           | 0        | 3      | 294   | 278   |
| Available       | 120      | 80     | 35200 | 17600 |
| Utilization (%) | 0        | 3      | ~0    | 1     |

- with Hoisting of "if"

## Performance Estimates

- ▢ Timing (ns)

- ▢ Summary

| Clock  | Target | Estimated | Uncertainty |
|--------|--------|-----------|-------------|
| ap_clk | 10.00  | 8.51      | 1.25        |

- ▢ Latency (clock cycles)

- ▢ Summary

| Latency |     | Interval |     | Type |
|---------|-----|----------|-----|------|
| min     | max | min      | max |      |
| 41      | 41  | 41       | 41  | none |

- ▢ Detail

- ⊕ Instance

- ⊕ Loop

## Utilization Estimates

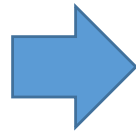
- ▢ Summary

| Name            | BRAM_18K | DSP48E | FF    | LUT   |
|-----------------|----------|--------|-------|-------|
| DSP             | -        | -      | -     | -     |
| Expression      | -        | 3      | 0     | 184   |
| FIFO            | -        | -      | -     | -     |
| Instance        | -        | -      | -     | -     |
| Memory          | 0        | -      | 79    | 9     |
| Multiplexer     | -        | -      | -     | 87    |
| Register        | -        | -      | 128   | -     |
| Total           | 0        | 3      | 207   | 280   |
| Available       | 120      | 80     | 35200 | 17600 |
| Utilization (%) | 0        | 3      | ~0    | 1     |

# Loop Fission

- The FIR has two fundamental operations: Shifts the data through the shift\_reg array, and the MAC operations
- Loop fission takes these two operations and implements each of them in their own loop
  - Each one is optimized independently, so it is a decomposition of an FSM

```
35 void fir(int *y, int x)
36 {
37     coef_t c[N] = { // 0.17 = 20KHz/44.1KHz, LPF, Hamming Window
38         -136, -397, -87, 3004, 8338, 11142, 8338,
39         3004, -87, -397, -136, };
40
41     static data_t shift_reg[N];
42     acc_t acc;
43     int i;
44
45     acc = 0;
46     FIR_LOOP: for (i = N - 1; i >= 0; i--) {
47         if (i == 0) {
48             acc += x * c[0];
49             shift_reg[0] = x;
50         } else {
51             shift_reg[i] = shift_reg[i - 1];
52             acc += shift_reg[i] * c[i];
53         }
54     }
55     *y = acc;
56 }
```



```
void fir_loop_fission(int *y, int x)
{
    coef_t c[N] = { // 0.17 = 20KHz/44.1KHz, LPF, Hamming Window
        -136, -397, -87, 3004, 8338, 11142, 8338,
        3004, -87, -397, -136, };

    static data_t shift_reg[N];
    acc_t acc;
    int i;

    acc = 0;
    SHIFT_REG: for (i = N - 1; i > 0; i--) {
        shift_reg[i] = shift_reg[i - 1];
    }
    shift_reg[0] = x;

    MACs: for (i = N - 1; i >= 0; i--) {
        acc += shift_reg[i] * c[i];
    }

    *y = acc;
}
```

# Loop Unrolling

- By default, the Vivado HLS tool synthesizes for loops in a sequential manner
- The data path executes sequentially for each iteration of the loop
- Manually unrolling the SHIFT\_REG loop

```
for (i = N - 1; i > 1; i = i - 2) {  
    shift_reg[i] = shift_reg[i - 1];  
    shift_reg[i - 1] = shift_reg[i - 2];  
}  
if (i == 1) {  
    shift_reg[1] = shift_reg[0];  
}  
shift_reg[0] = x;
```



# Unroll Pragma

#pragma HLS unroll factor=n

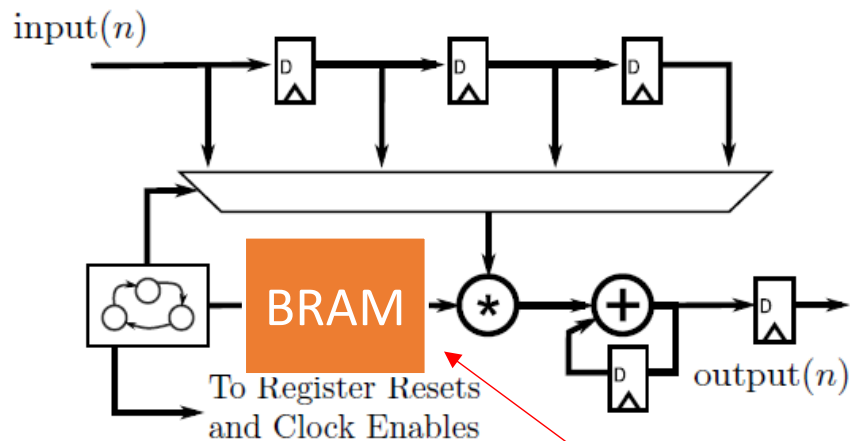
(if factor is none, the HLS tries to unroll all operations!!)

```
12 void fir_unroll(int *y, int x)
13 {
14     coef_t c[N] = { // 0.17 = 20KHz/44.1KHz, LPF, Hamming Window
15                     -136, -397, -87, 3004, 8338, 11142, 8338,
16                     3004, -87, -397, -136, };
17
18     static data_t shift_reg[N];
19     acc_t acc;
20     int i;
21
22     acc = 0;
23     FIR_LOOP: for (i = N - 1; i >= 0; i--) {
24 #pragma HLS unroll factor=4
25         if (i == 0) {
26             acc += x * c[0];
27             shift_reg[0] = x;
28         } else {
29             shift_reg[i] = shift_reg[i - 1];
30             acc += shift_reg[i] * c[i];
31         }
32     }
33     *y = acc;
34 }
```

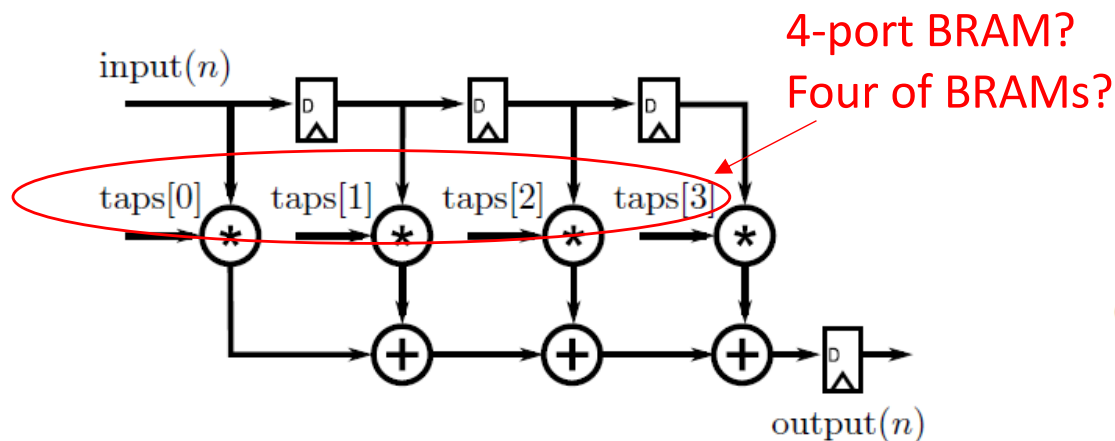
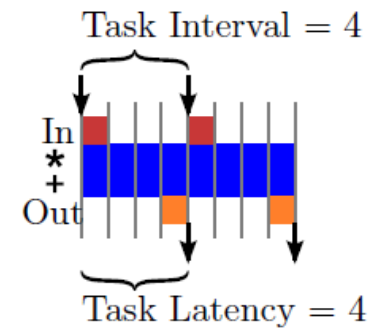
If your design does not synthesize in under 15 minutes, you should carefully consider the effect of your optimizations. It is certainly possible that large designs can take a significant amount of time for the Vivado HLS to synthesize them.

# Partition BRAM into Smaller One?

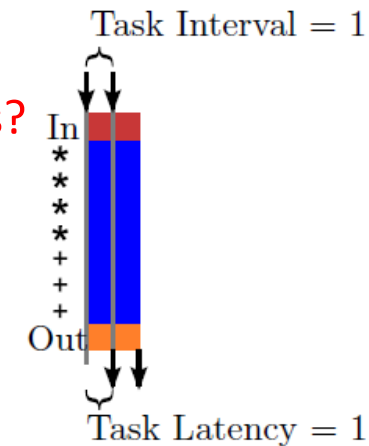
- Use "#pragma HLS array\_partition"



Single port BRAM

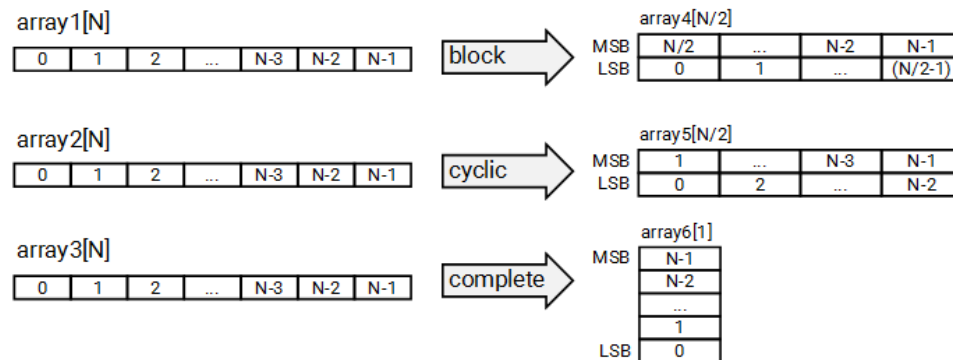


4-port BRAM?  
Four of BRAMs?



# Array\_Partition

#pragma HLS ARRAY\_PARTITION variable=(variable name) (access pattern) factor=(# of partitions)  
dim=(array dimension)



access pattern

`my_array[10][6][4]` → partition dimension 1 →

- `my_array_0[6][4]`
- `my_array_1[6][4]`
- `my_array_2[6][4]`
- `my_array_3[6][4]`
- `my_array_4[6][4]`
- `my_array_5[6][4]`
- `my_array_6[6][4]`
- `my_array_7[6][4]`
- `my_array_8[6][4]`
- `my_array_9[6][4]`

dimension

(dim=0 denotes all dimension are partitioned)

# Example

- Good performance! But...

```
void fir_array_partition(int *y, int x)
{
    coef_t c[N] = { // 0.17 = 20KHz/44.1KHz, LPF, Hamming Window
        -136, -397, -87, 3004, 8338, 11142, 8338,
        3004, -87, -397, -136, };
    #pragma HLS array_partition variable=c complete

    static data_t shift_reg[N];
    acc_t acc;
    int i;

    acc = 0;
    SHIFT_REG: for (i = N - 1; i > 0; i--) {
    #pragma HLS unroll
        shift_reg[i] = shift_reg[i - 1];
    }
    shift_reg[0] = x;

    MACs: for (i = N - 1; i >= 0; i--) {
    #pragma HLS unroll
        acc += shift_reg[i] * c[i];
    }

    *y = acc;
}
```

## Performance Estimates

### Timing (ns)

#### Summary

| Clock  | Target | Estimated | Uncertainty |
|--------|--------|-----------|-------------|
| ap_clk | 10.00  | 8.74      | 1.25        |

### Latency (clock cycles)

#### Summary

| Latency |     | Interval |     | Type |
|---------|-----|----------|-----|------|
| min     | max | min      | max |      |
| 2       | 2   | 2        | 2   | none |

#### Detail

##### + Instance

##### + Loop

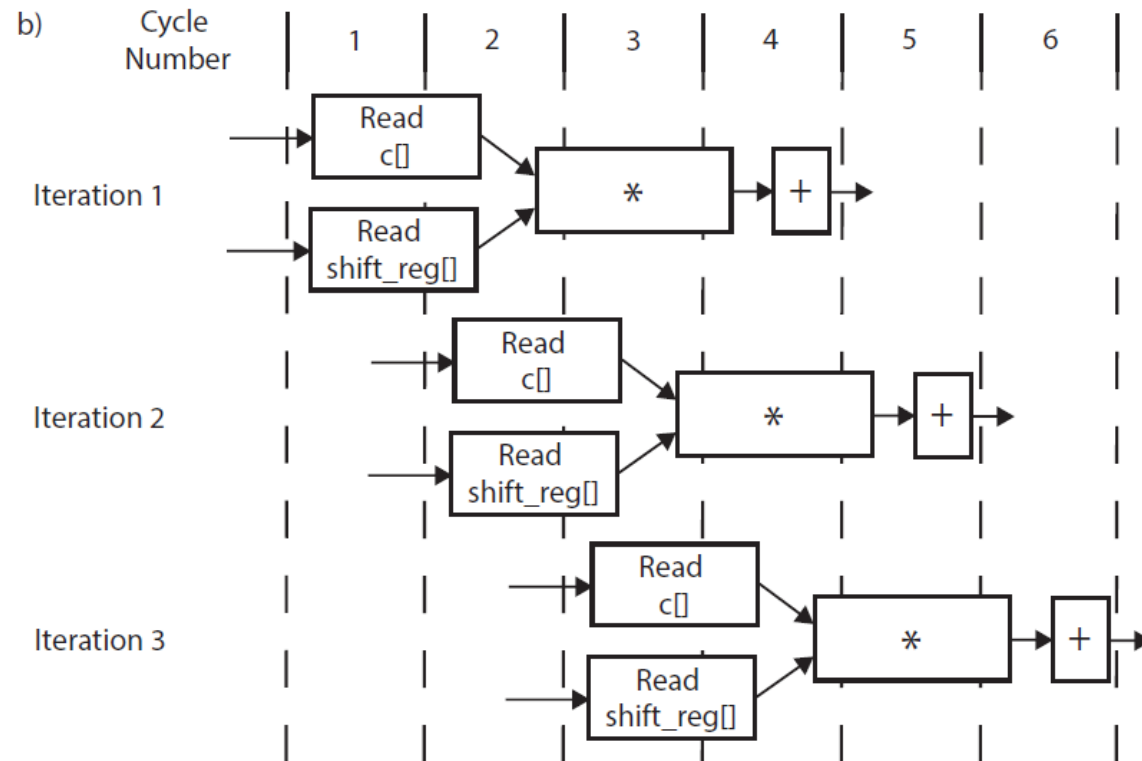
## Utilization Estimates

### Summary

| Name            | BRAM_18K | DSP48E | FF    | LUT   |
|-----------------|----------|--------|-------|-------|
| DSP             | -        | -      | -     | -     |
| Expression      | -        | 27     | 0     | 642   |
| FIFO            | -        | -      | -     | -     |
| Instance        | -        | -      | -     | -     |
| Memory          | -        | -      | -     | -     |
| Multiplexer     | -        | -      | -     | 21    |
| Register        | -        | -      | 764   | -     |
| Total           | 0        | 27     | 764   | 663   |
| Available       | 120      | 80     | 35200 | 17600 |
| Utilization (%) | 0        | 33     | 2     | 3     |

# Loop Pipelining

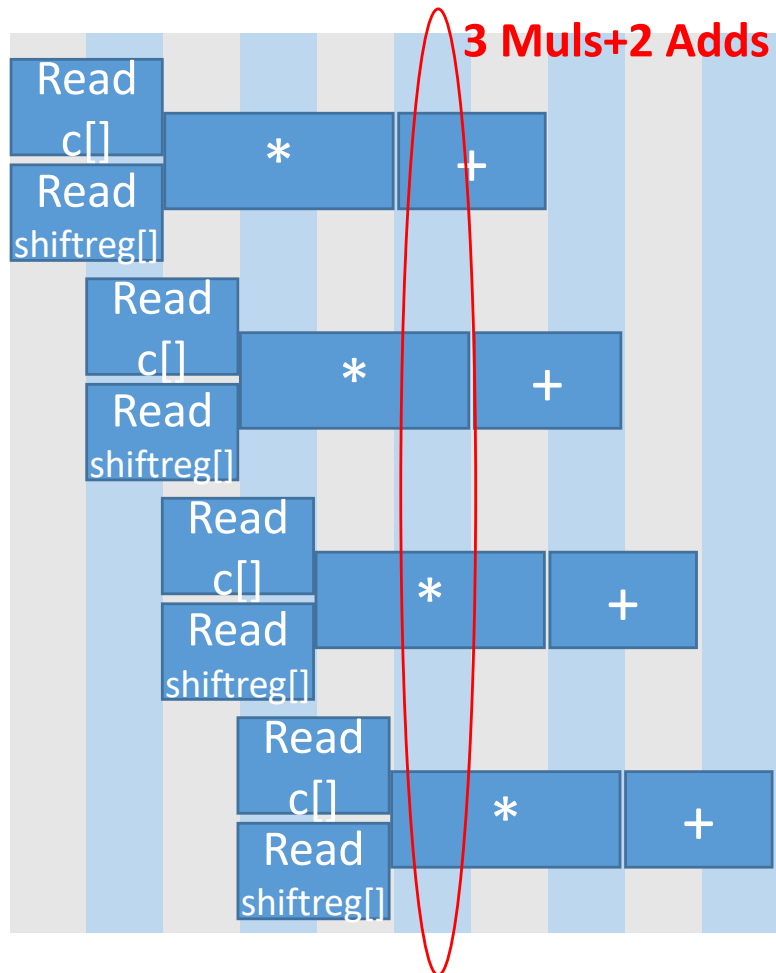
- All of the statements in the second iteration happen only when all of the statements from the first iteration are complete
- Schedule for three iterations of a pipelined version of the MAC for loop



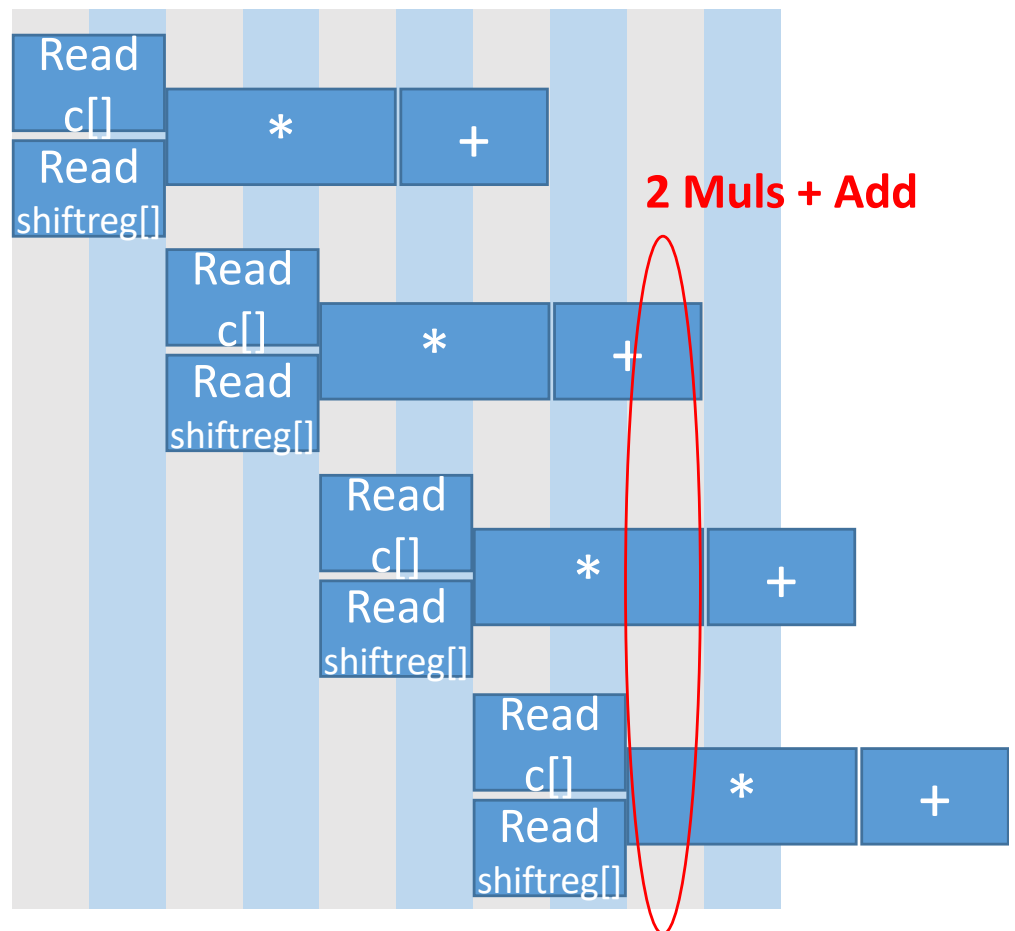
# Loop Initiation Interval (II)

- The number of clock cycles until the next iteration of the loop can start
- Note that, this may not always be possible due to resource/timing constraints and/or dependencies in the code

#pragma HLS pipeline II=1



#pragma HLS pipeline II=2



# Data Type in C-language

- C language provides many different data types to describe different kinds of behavior
- The primary benefits of using these different data types in software revolve around the amount of storage that the data type require
- All of these data type have a size which is a power of 2
  - (unsigned/singed) int
  - float
  - double
  - (unsigned/singed) char
  - short long
  - long
  - long long

# Bitwidth Optimization

- The same benefits are seen in an FPGA implementation, but they are even more pronounced
  - Since the Vivado HLS supports a custom (arbitrary precision) data types
- `#include "ap_int.h"`, then you can use
  - unsigned: `ap_uint<width>`, where width takes 1 to 1024
  - signed: `ap_int<width>`



# More Reduced!

```
/* comment out for bitwith optimization part */
typedef ap_int<16> data_t;
typedef ap_int<16> coef_t;
typedef ap_int<24> acc_t;
/*
typedef int data_t;
typedef int coef_t;
typedef int acc_t;
*/
void fir_array_partition(int *y, int x)
{
    coef_t c[N] = { // 0.17 = 20KHz/44.1KHz, LPF, Hamming Window
        -136, -397, -87, 3004, 8338, 11142, 8338,
        3004, -87, -397, -136, };
    #pragma HLS array_partition variable=c complete

    static data_t shift_reg[N];
    acc_t acc;
    int i;

    acc = 0;
    SHIFT_REG: for (i = N - 1; i > 0; i--) {
    #pragma HLS unroll
        shift_reg[i] = shift_reg[i - 1];
    }
    shift_reg[0] = x;

    MACs: for (i = N - 1; i >= 0; i--) {
    #pragma HLS unroll
        acc += shift_reg[i] * c[i];
    }

    *y = acc;
}
```

## Performance Estimates

### Timing (ns)

#### Summary

| Clock  | Target | Estimated | Uncertainty |
|--------|--------|-----------|-------------|
| ap_clk | 10.00  | 8.74      | 1.25        |

### Latency (clock cycles)

#### Summary

| Latency |     | Interval |     | Type |
|---------|-----|----------|-----|------|
| min     | max | min      | max |      |
| 2       | 2   | 2        | 2   | none |

#### Detail

##### + Instance

##### + Loop

## Utilization Estimates

### Summary

| Name            | BRAM_18K | DSP48E | FF    | LUT   |
|-----------------|----------|--------|-------|-------|
| DSP             | -        | -      | -     | -     |
| Expression      | -        | 27     | 0     | 642   |
| FIFO            | -        | -      | -     | -     |
| Instance        | -        | -      | -     | -     |
| Memory          | -        | -      | -     | -     |
| Multiplexer     | -        | -      | -     | 21    |
| Register        | -        | -      | 764   | -     |
| Total           | 0        | 27     | 764   | 663   |
| Available       | 120      | 80     | 35200 | 17600 |
| Utilization (%) | 0        | 33     | 2     | 3     |

# Exercise

1. (Mandatory) Compare an unrolling FIR design with a pipelined one with respect to HW resource and performance
2. (Optional) Execute an unrolling FIR design on your ZYBO board

If you meet any troubles, don't hesitate to contact me.

[nakahara@ict.e.titech.ac.jp](mailto:nakahara@ict.e.titech.ac.jp)

Deadline is 26<sup>th</sup>, July, 2019 JST PM13:20

(At the beginning of the next lecture)