CSE 291: FPGA for Computer Vision

Janarbek Matai

Department of Computer Science and Engineering

University of California, San Diego

04/13/2017

UCSDCSE

Computer Science and Engineering

CSE 3219

- Key: 6951495
- Do not bring food
- Room needs to be kept locked (do not prop open the door)
- If for any reason if you disconnect equipment (keyboard), please reconnect it
- Please keep the room clean and respect the room.

Assignment 1 ?

- Due today

Final Project Proposal

I. Bitbukcet page
   1) Motivation & Background → Why you choose to implement this ?
   2) Description of the algorithm (Software)
      - ✓ Discuss any algorithm specific bottlenecks
   3) Description of hardware architecture
      - ✓ A block diagram and detailed architectural diagram + short explanation
   4) Scope and Schedule
      - ✓ **Explicitly state the function (or functions) to be ported to an FPGA platform**
      - ✓ Schedule for the rest of quarter

II. SW folder (May 02, 2017)
   I. Must contain software reference model and instructions to run the software

III. HW folder (June 06, 2017)
   I. HLS
      - ✓ Source code + Instructions + data
   II. SDSoC
      - ✓ Source code + instructions + data

Project Proposal

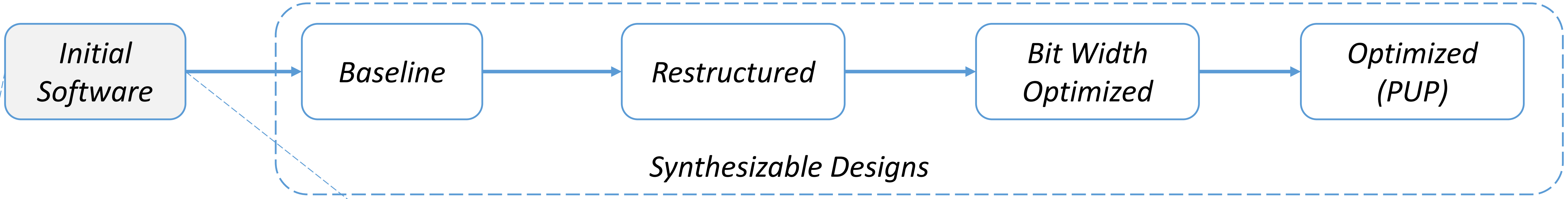
- Implement Binarized Convolutional Neural Network on an FPGA (Note* do not implement MLP)
- Implement Sequeezenet on an FPGA
- Implementing HOG transform on an FPGA
- Implementing Lane Detection on an FPGA (It has been done in previous years)
- Implementing Optical Flow on an FPGA
- Implementing Deep Compression on an FPGA
- Verilog versus HLS ??? Binary Matrix Multiplication ??
- ,….

Resources for Final Project

- Vivado HLS
- SDSoC
- Zedboard (Target an algorithm that can run on an ARM processor)
- http://xillybus.com/
- RIFFA: http://riffa.ucsd.edu/
- Pynq board ?
- Virtex 709 board ? –Talk to me if you need
- https://github.com/fengbintu/Neural-Networks-on-Silicon
- http://www.embedded-vision.com/

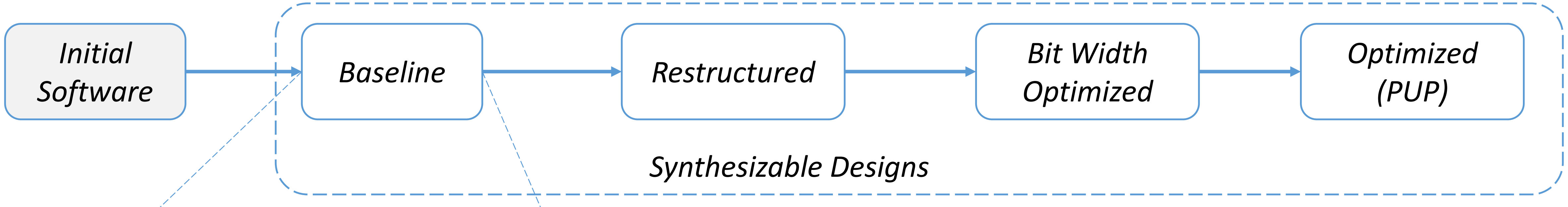
Last week: Vivado HLS Introduction

HLS Design Flow



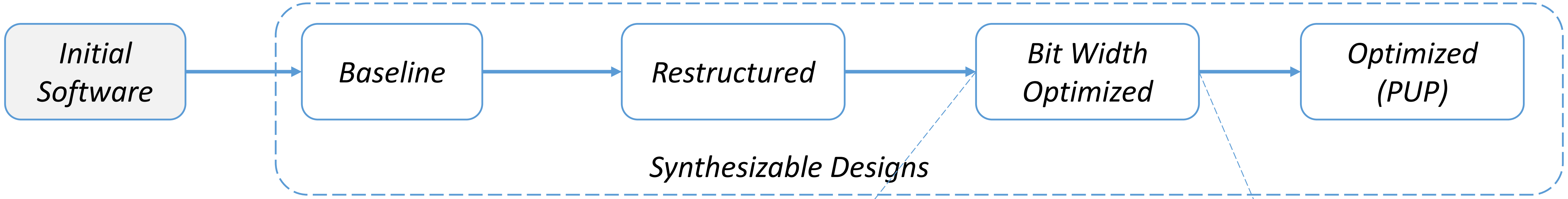
```
int *array;
array = (int *)malloc(sizeof(int)*N);
for(int i=0;i<N;i++) {

array = … ;
}
```

HLS Design Flow



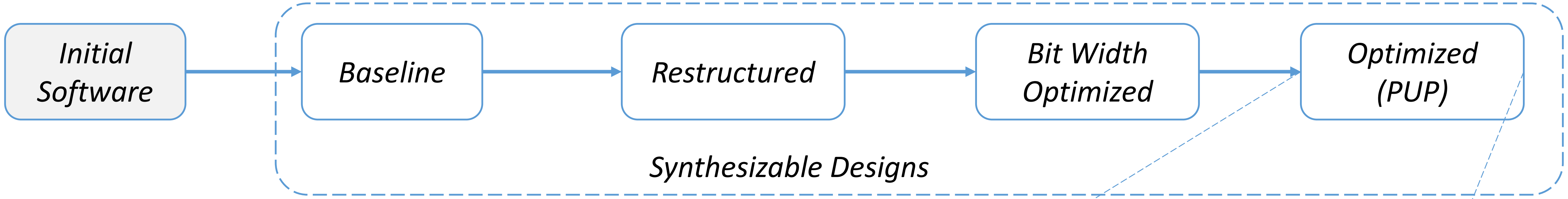
```
int *array;
array[N] ={0};
for(int i=0;i<N;i++) {

array = … ;
}
```

HLS Design Flow



```
ap_int<10> array[N] ={0};
for(int i=0;i<N;i++) {

array = … ;
}
```

HLS Design Flow



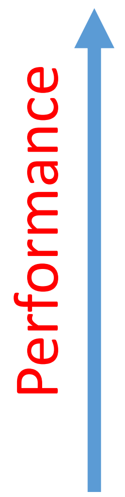
```
ap_int<10> array[N] ={0};
for(int i=0;i<N;i++) {
#pragma HLS PIPELINE
array = … ;
}
```

Vivado HLS

- Pipeline, Partition, Unroll
- C-Sim, Co-Sim, Implementation,
- ,…

Today: High-Level Synthesis: Performance & Area Optimizations

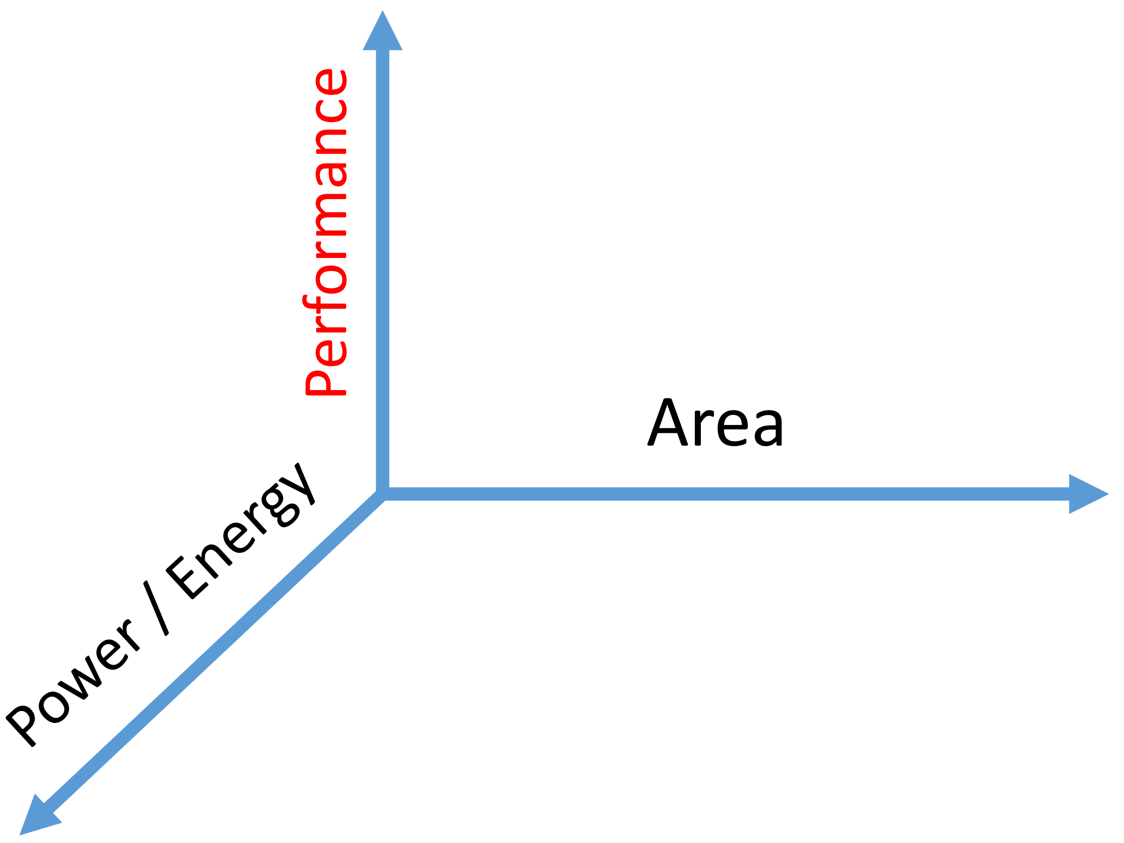
Hardware Design



Hardware Design

Performance
Area

Hardware Design



Today

- HLS Area Optimization
  - HLS pragmas

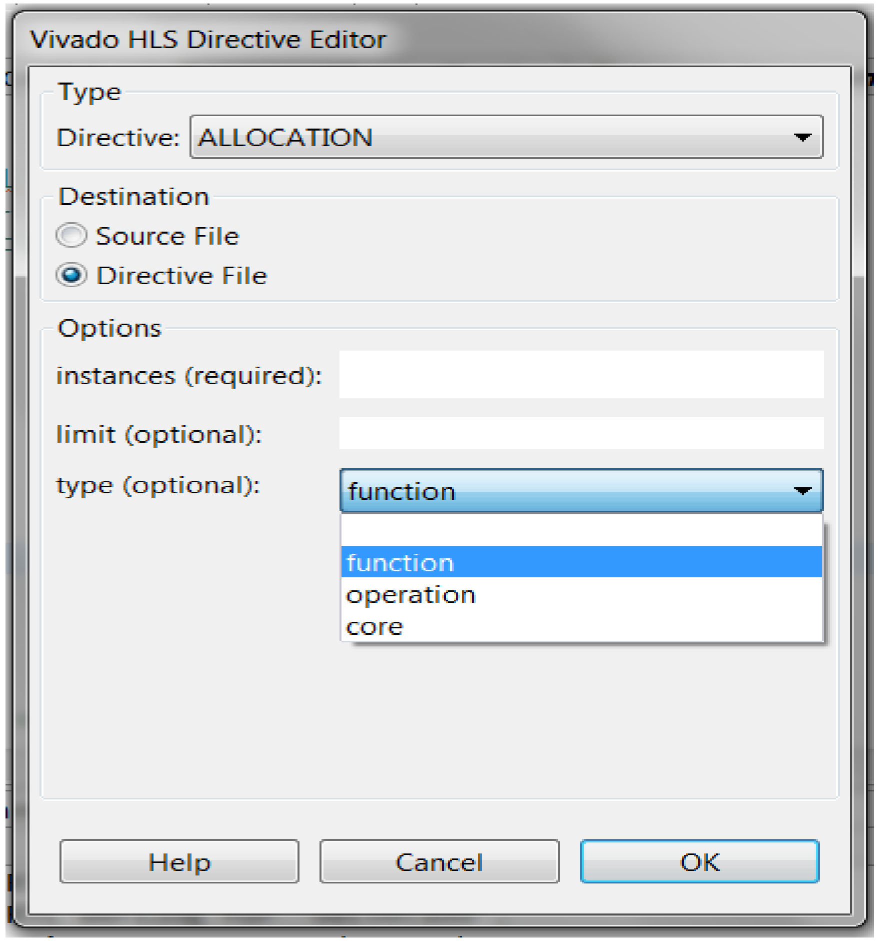
HLS Area Optimization Techniques

HLS Area Optimization Techniques

1. Control the number of elements and types
2. Combining memories (arrays)
3. Control design hierarchy
4. Bit-width optimizations

Allocation

- Limits different hardware resources
  - Operation
    - Add, mul
  - Function
    - foo
  - Core
    - PipeMul2


Vivado HLS Directive Editor

Type

Directive: ALLOCATION

Destination

Source File [✓ Directive File]

Options

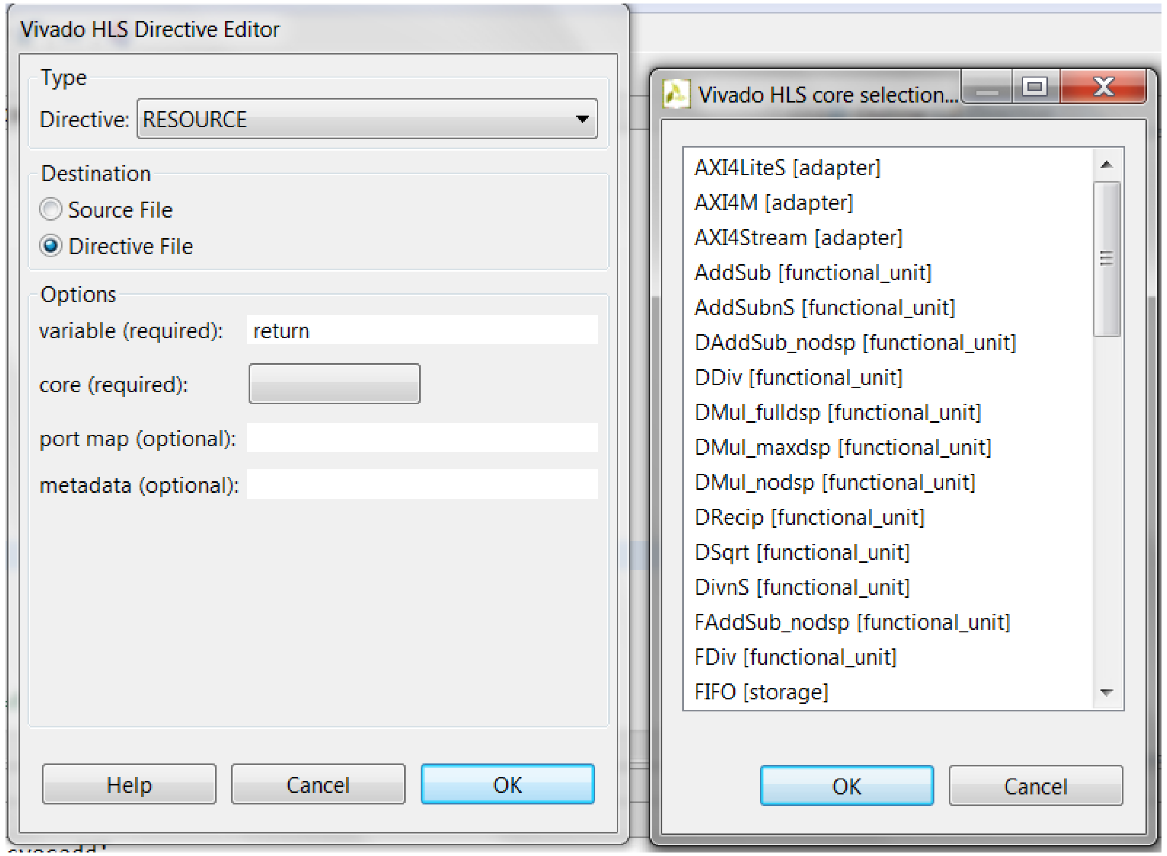
instances (required):

limit (optional):

type (optional): function

Resource

- Selects types of resources



Vivado HLS Directive Editor

Type

Directive: RESOURCE

Destination

( ) Source File
[✓ Directive File]

Options

variable (required): return

core (required):

port map (optional):

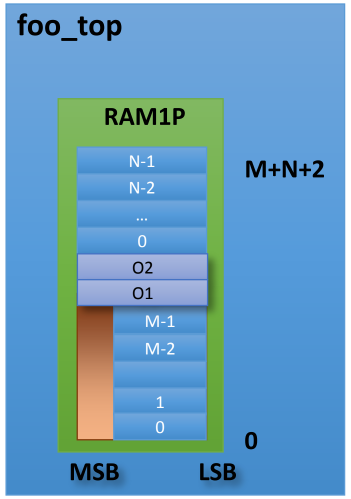
metadata (optional):

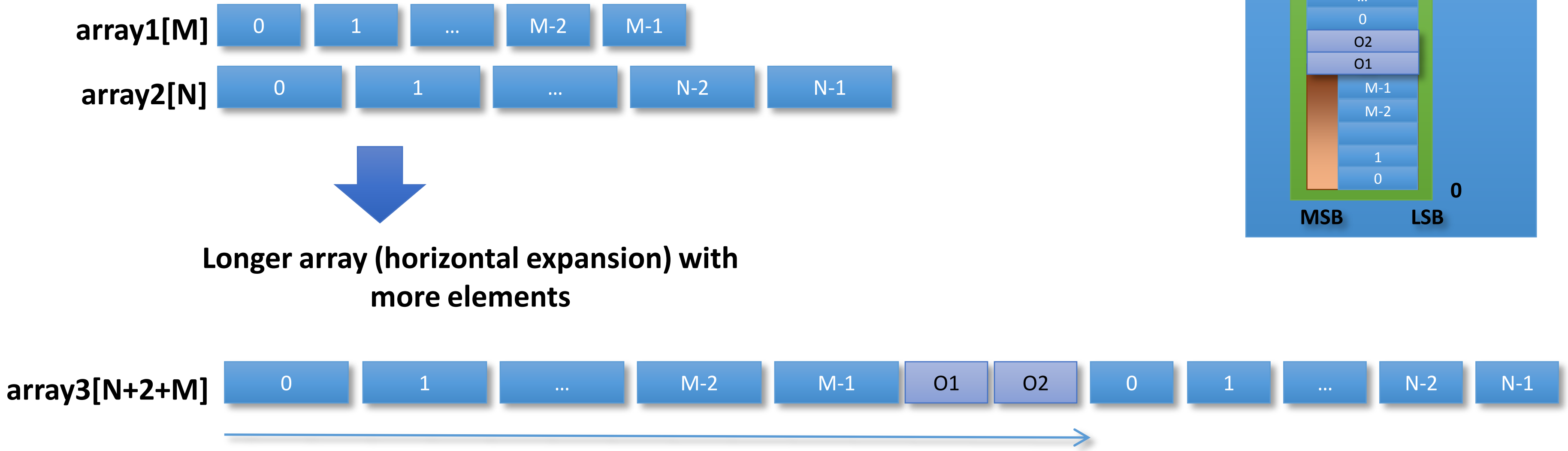
Vivado HLS core selection...

AXI4LiteS [adapter]
AXI4M [adapter]
AXI4Stream [adapter]
AddSub [functional_unit]
AddSubnS [functional_unit]
DAddSub_nodsp [functional_unit]
DDiv [functional_unit]
DMul_fulldsp [functional_unit]
DMul_maxdsp [functional_unit]
DMul_nodsp [functional_unit]
DRecip [functional_unit]
DSqrt [functional_unit]
DivnS [functional_unit]
FAddSub_nodsp [functional_unit]
FDiv [functional_unit]
FIFO [storage]

Array map: Horizontal

- Merging smaller arrays into one (size increases)
  - Saves BRAM





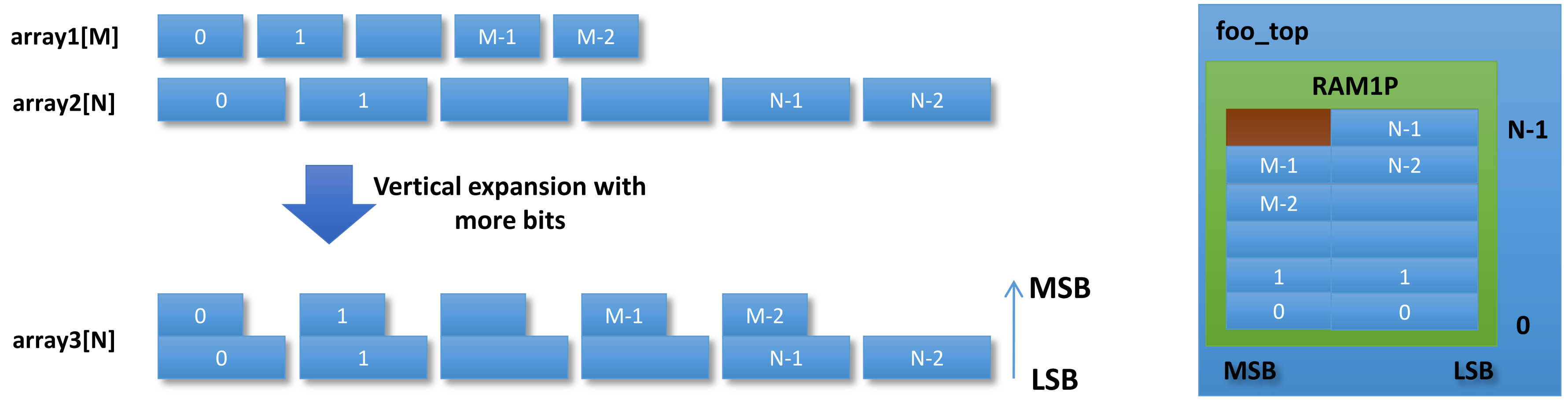
Longer array (horizontal expansion) with more elements

Optionally apply an offset

Offset of M+ 1 from the start

Array map: Vertical

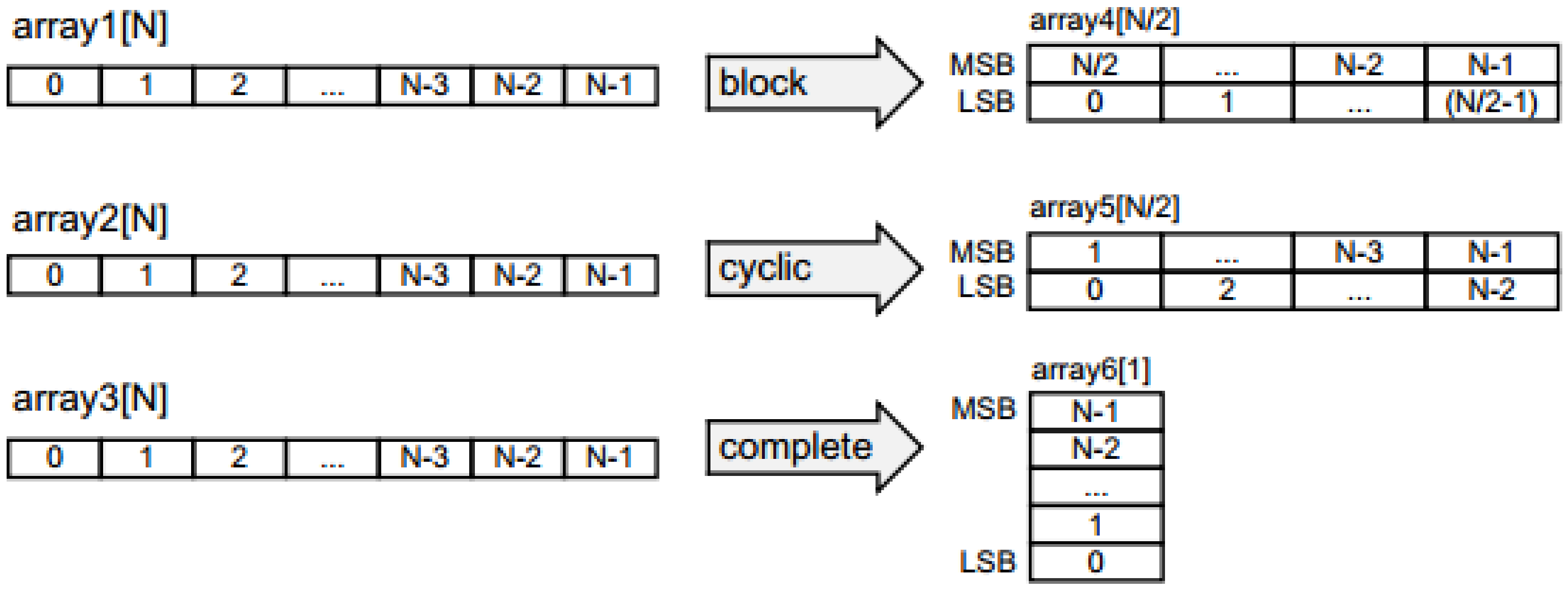
- Merging smaller arrays into one (bit width increases)



- Reduces number of BRAM & Allows parallel access

Array Reshape

- Array partition + Array Map (Vertical)

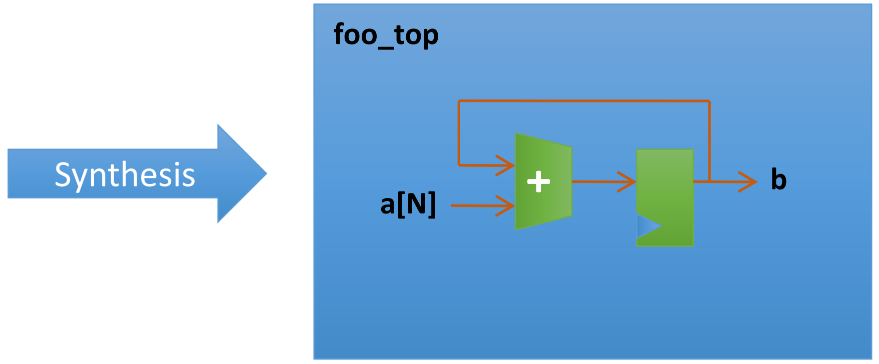


Source: Xilinx resources

Loops

- Loops (not unrolled) share resources

```
void foo_top (…) {
  ...
  Add: for (i=3;i>=0;i--) {
            b = a[i] + b;
  ...
  }
```



Source: Xilinx resources

Loop Merge

```
My_Region: {
for (i = 0; i < N; ++i)
    A[i] = B[i] + 1;
for (i = 0; i < N; ++i)
    C[i] = A[i] / 2;
}
```

Merge

```
for (i = 0; i < N; ++i) {
    A[i] = B[i] + 1;
    C[i] = A[i] / 2;
}
```

```
for (i = 0; i < N; ++i)
    C[i] = (B[i] + 1) / 2;
```

Removes A[i] related logic (address, jump)

Arbitrary Precision Types

Ap_int

Ap_fixed

Summary: Area Optimization

- Allocation
- Resource
- Inline
- Loop (merge) / unnecessary logic removal
- Array map
- Array reshape
- Arbitrary precision types

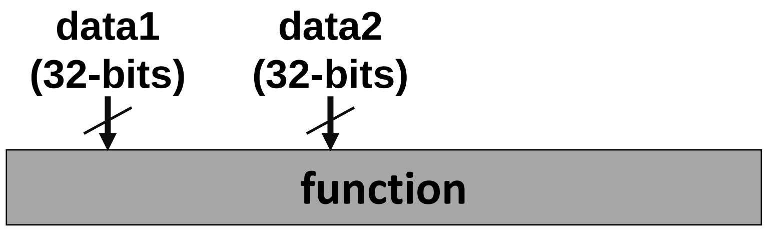
Vivado HLS directives

1. `set_directive_data_pack`
2. `set_directive_dependence`
3. `set_directive_expression_balance`
4. `set_directive_function_instantiate`
5. `set_directive_occurrence`
6. `set_directive_protocol`
7. `set_directive_reset`

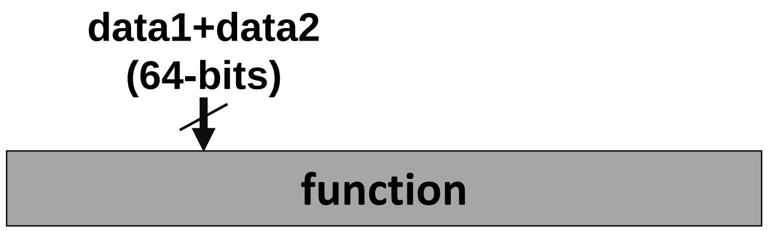
set_directive_data_pack

- Packs the data fields of a struct into a single scalar with a wider word width.

```
typedef struct{
            ITYPE data1; // 32-bits
            ITYPE data2; // 32-bits
}STYPE;

void function(STYPE in, STYPE out);

#pragma HLS data_pack variable=in instance = input
#pragma HLS data_pack variable=out instance = output
```



[Without directives]

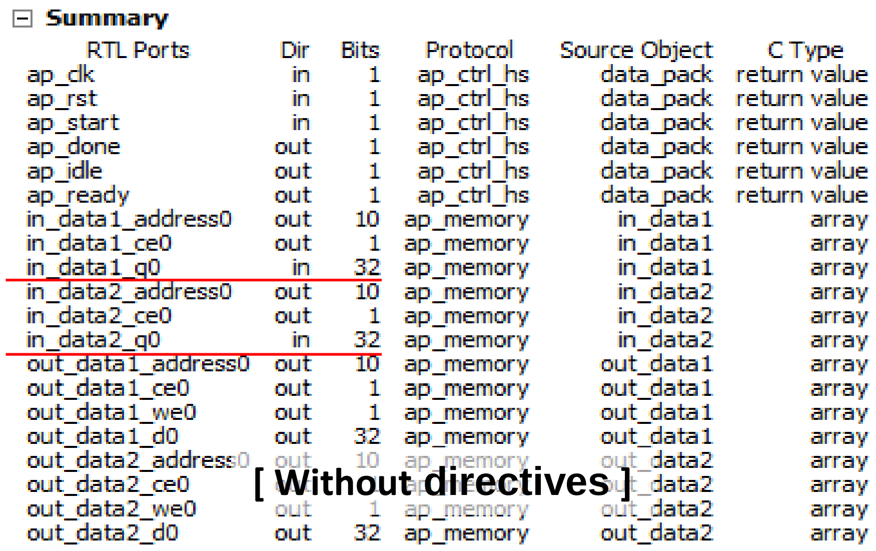


[With directives]

set_directive_data_pack

- Packs the data fields of a struct into a single scalar with a wider word width.

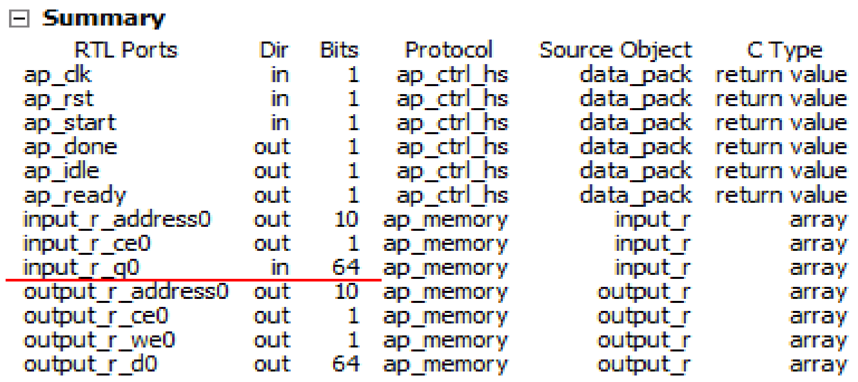
```
typedef struct{
		ITYPE data1; // 32-bits
		ITYPE data2; // 32-bits
}STYPE;

void function(STYPE in, STYPE out);

#pragma HLS data_pack variable=in instance = input
#pragma HLS data_pack variable=out instance = output
```



Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	data_pack	return value
ap_rst	in	1	ap_ctrl_hs	data_pack	return value
ap_start	in	1	ap_ctrl_hs	data_pack	return value
ap_done	out	1	ap_ctrl_hs	data_pack	return value
ap_idle	out	1	ap_ctrl_hs	data_pack	return value
ap_ready	out	1	ap_ctrl_hs	data_pack	return value
in_data1_address0	out	10	ap_memory	in_data1	array
in_data1_ce0	out	1	ap_memory	in_data1	array
in_data1_q0	in	32	ap_memory	in_data1	array
in_data2_address0	out	10	ap_memory	in_data2	array
in_data2_ce0	out	1	ap_memory	in_data2	array
in_data2_q0	in	32	ap_memory	in_data2	array
out_data1_address0	out	10	ap_memory	out_data1	array
out_data1_ce0	out	1	ap_memory	out_data1	array
out_data1_we0	out	1	ap_memory	out_data1	array
out_data1_d0	out	32	ap_memory	out_data1	array
out_data2_address0	out	10	ap_memory	out_data2	array
out_data2_ce0	out	1	ap_memory	out_data2	array
out_data2_we0	out	1	ap_memory	out_data2	array
out_data2_d0	out	32	ap_memory	out_data2	array

[Without directives]



Summary

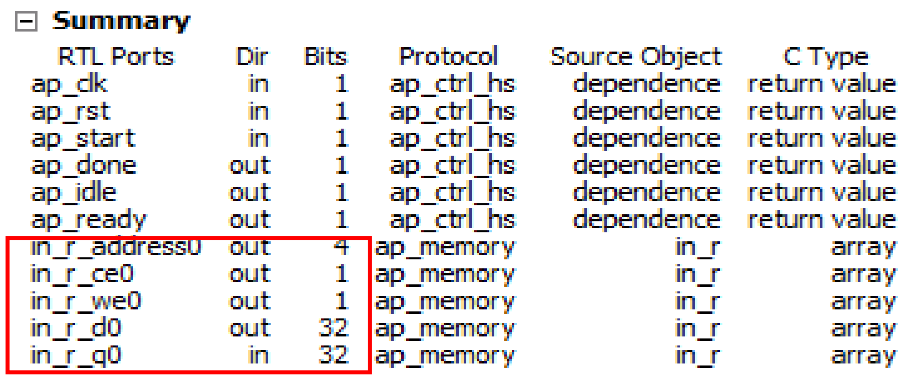
RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	data_pack	return value
ap_rst	in	1	ap_ctrl_hs	data_pack	return value
ap_start	in	1	ap_ctrl_hs	data_pack	return value
ap_done	out	1	ap_ctrl_hs	data_pack	return value
ap_idle	out	1	ap_ctrl_hs	data_pack	return value
ap_ready	out	1	ap_ctrl_hs	data_pack	return value
input_r_address0	out	10	ap_memory	input_r	array
input_r_ce0	out	1	ap_memory	input_r	array
input_r_q0	in	64	ap_memory	input_r	array
output_r_address0	out	10	ap_memory	output_r	array
output_r_ce0	out	1	ap_memory	output_r	array
output_r_we0	out	1	ap_memory	output_r	array
output_r_d0	out	64	ap_memory	output_r	array

[With directives]

set_directive_dependence

- Used to provide additional information that can overcome loop-carry dependencies and allow loops to be pipelined (or pipelined with lower intervals).

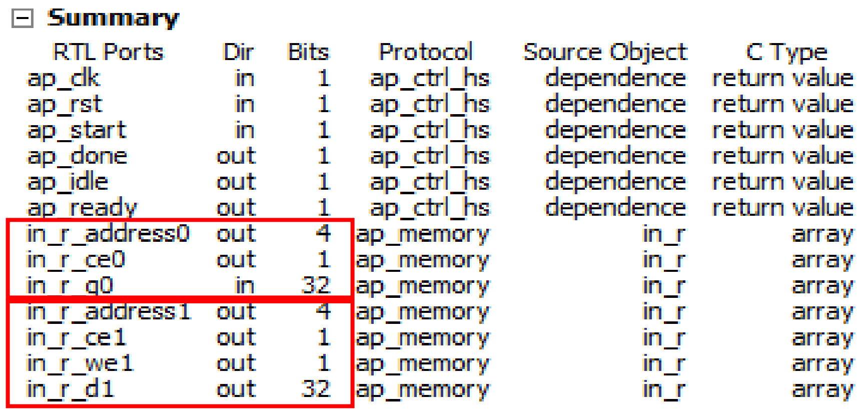
```
for( i=1 ; i<SIZE ; i++ ){
#pragma HLS PIPELINE
#pragma HLS DEPENDENCE variable=in inter false
                    in[i] = in[i-1]*2;
          }
```



⊟ **Summary**

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	dependence	return value
ap_rst	in	1	ap_ctrl_hs	dependence	return value
ap_start	in	1	ap_ctrl_hs	dependence	return value
ap_done	out	1	ap_ctrl_hs	dependence	return value
ap_idle	out	1	ap_ctrl_hs	dependence	return value
ap_ready	out	1	ap_ctrl_hs	dependence	return value
in_r_address0	out	4	ap_memory	in_r	array
in_r_ce0	out	1	ap_memory	in_r	array
in_r_we0	out	1	ap_memory	in_r	array
in_r_d0	out	32	ap_memory	in_r	array
in_r_q0	in	32	ap_memory	in_r	array

[Without directives]



⊟ **Summary**

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	dependence	return value
ap_rst	in	1	ap_ctrl_hs	dependence	return value
ap_start	in	1	ap_ctrl_hs	dependence	return value
ap_done	out	1	ap_ctrl_hs	dependence	return value
ap_idle	out	1	ap_ctrl_hs	dependence	return value
ap_ready	out	1	ap_ctrl_hs	dependence	return value
in_r_address0	out	4	ap_memory	in_r	array
in_r_ce0	out	1	ap_memory	in_r	array
in_r_q0	in	32	ap_memory	in_r	array
in_r_address1	out	4	ap_memory	in_r	array
in_r_ce1	out	1	ap_memory	in_r	array
in_r_we1	out	1	ap_memory	in_r	array
in_r_d1	out	32	ap_memory	in_r	array

[With directives]

set_directive_function_instantiate

- Allows different instances of the same function to be locally optimized.

```
ITYPE function_instantiate_sub(DTYPE inval, DTYPE incr){
#pragma HLS function_instantiate variable=incr
    if(incr > 150)
                    return inval * incr;
    else
                    return inval / incr;
}
void function_instantiate(DTYPE in1, DTYPE in2, DTYPE in3,
                              DTYPE *out1, DTYPE *out2, DTYPE *out3 ){

          *out1 = function_instantiate_sub(in1,10);
          *out2 = function_instantiate_sub(in2,100);
          *out3 = function_instantiate_sub(in3,200);
}
```

set_directive_function_instantiate

- Allows different instances of the same function to be locally optimized.

```
ITYPE function_instantiate_sub(DTYPE inval, DTYPE incr){
#pragma HLS function_instantiate variable=incr
   if(incr > 150)
                        return inval * incr;
   else
                        return inval / incr;
}
void function_instantiate(DTYPE in1, DTYPE in2, DTYPE in3,
                                    DTYPE *out1, DTYPE *out2, DTYPE *out3 ){

            *out1 = function_instantiate_sub(in1,10);  ← inval / incr;
            *out2 = function_instantiate_sub(in2,100); ← inval / incr;
            *out3 = function_instantiate_sub(in3,200); ← inval * incr;
}
```

Performance (min/max latency)	15/26
Resources(FFs/LUTs)	4632/9432

[Without directives]

Performance (min/max latency)	22/22
Resources(FFs/LUTs)	3000/5381

[With directives]

set_directive_protocol

- This commands specifies a region of the code to be a protocol region. A protocol region can be used to manually specify an interface protocol.

```
P1: {
#pragma HLS PROTOCOL floating
   read1 = response[0];
   opcode = 5;
   ap_wait();// Added ap_wait statement
   *request = opcode;
   read2= response[1];
             }
C1: {
             *z1 = a + b;
             *z2 = read1 + read2;
             }
```

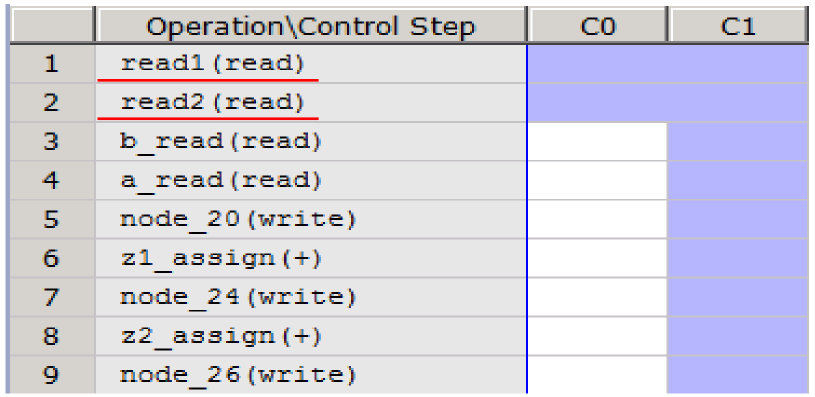
set_directive_protocol

- This commands specifies a region of the code to be a protocol region. A protocol region can be used to manually specify an interface protocol.

```
P1: {
   read1 = response[0];
   opcode = 5;
   *request = opcode;
   read2= response[1];
            }
```

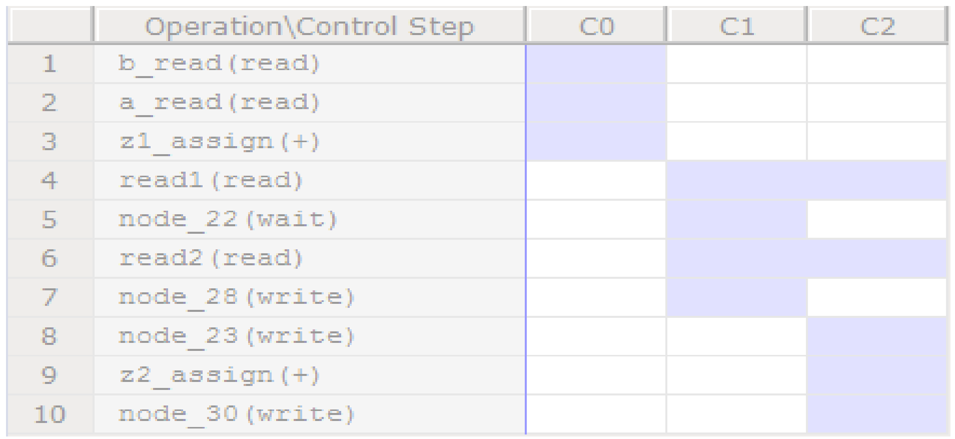
No data dependency

- **HLS re-orders and implements them in parallel**



	Operation\Control Step	C0	C1
1	read1(read)	▨	▨
2	read2(read)	▨	▨
3	b_read(read)		▨
4	a_read(read)		▨
5	node_20(write)		▨
6	z1_assign(+)		▨
7	node_24(write)		▨
8	z2_assign(+)		▨
9	node_26(write)		▨

[Without directives]



	Operation\Control Step	C0	C1	C2
1	b_read(read)	▨		
2	a_read(read)	▨		
3	z1_assign(+)	▨		
4	read1(read)		▨	▨
5	node_22(wait)		▨	
6	read2(read)		▨	▨
7	node_28(write)		▨	
8	node_23(write)			▨
9	z2_assign(+)			▨
10	node_30(write)			▨

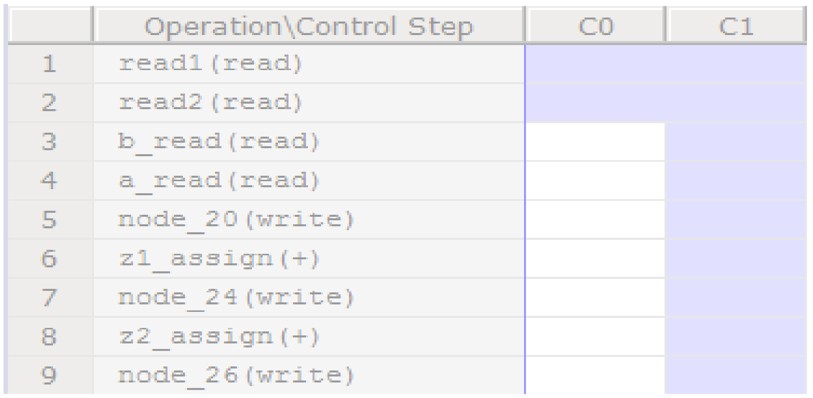
[With directives]

set_directive_protocol

- This commands specifies a region of the code to be a protocol region. A protocol region can be used to manually specify an interface protocol.

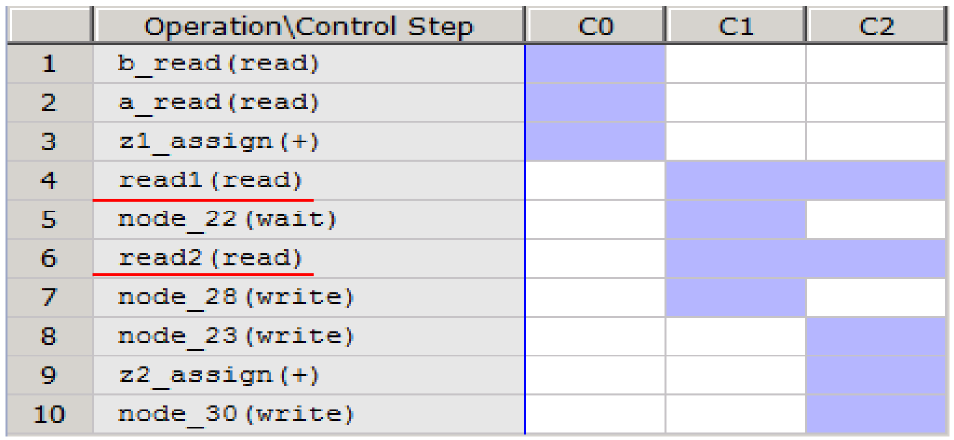
```
P1: {
#pragma HLS PROTOCOL floating
    read1 = response[0];
    opcode = 5;
    ap_wait();// Added ap_wait statement
    *request = opcode;
    read2= response[1];
            }
```

This results in the following exact I/O behavior specified in the code.



	Operation\Control Step	C0	C1
1	read1(read)	▨	▨
2	read2(read)	▨	▨
3	b_read(read)		▨
4	a_read(read)		▨
5	node_20(write)		▨
6	z1_assign(+)		▨
7	node_24(write)		▨
8	z2_assign(+)		▨
9	node_26(write)		▨

[Without directives]



	Operation\Control Step	C0	C1	C2
1	b_read(read)	▨		
2	a_read(read)	▨		
3	z1_assign(+)	▨		
4	read1(read)		▨	▨
5	node_22(wait)		▨	
6	read2(read)		▨	▨
7	node_28(write)		▨	
8	node_23(write)			▨
9	z2_assign(+)			▨
10	node_30(write)			▨

[With directives]

set_directive_reset

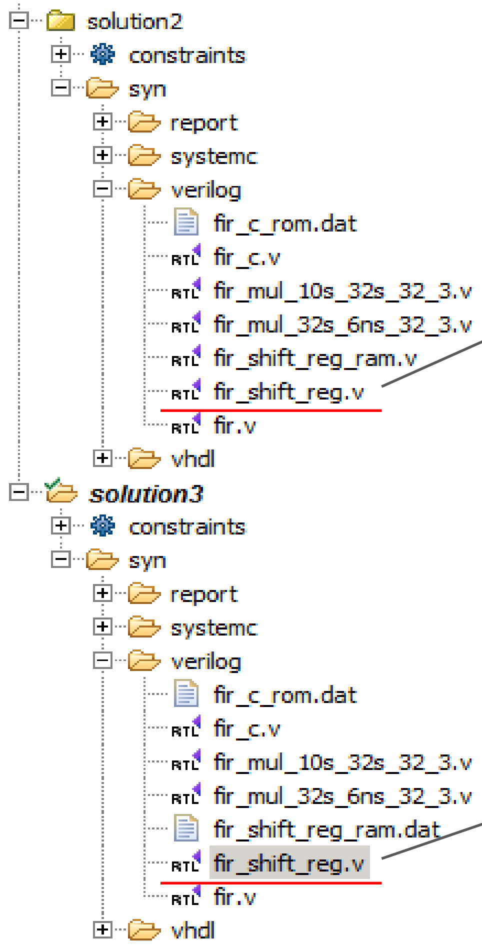
- This directive is used to add or remove reset on a specific state variable (global or static).

```
static data_t shift_reg[N];
#pragma HLS RESET variable = shift_reg off
acc=0;
Shift_Accum_Loop: for (i=N-1;i>=0;i--) {
        if (i==0) {
                acc+=x*c[0];
                shift_reg[0]=x;
        } else {
                shift_reg[i]=shift_reg[i-1];
                acc+=shift_reg[i]*c[i];
        }
}
*y=acc;
```

Off reset signal for shift_reg

set_directive_reset

- This directive is used to add or remove reset on a specific state variable (global or static).



```
always @(posedge clk) begin
    if (reset)
        written <= 1'b0;
    else begin
        if (ce0 & we0) begin
            written[address0] <= 1'b1;
        end
    end
end
```

[Without directives]

```
always @(posedge clk) begin
    if (ce0)
    begin
        if (we0)
        begin
          …
        end
        else
          …
    end
end
```

No reset used in RTL code!

[With directives]

set_directive_occurrence

- Used when pipelining functions or loops, to specify that the code in a location is executed at a lesser rate than the code in the enclosing function or loop.

```
void top( hls::stream<int> &stream_input, hls::stream<int> &stream_output) {
    int i,buff[4];
    ap_uint<2> buff_index=0;
loop_a: for ( i=0 ; i<16; i++ ) {
#pragma HLS pipeline II=1
        buff[buff_index]=stream_input.read();
my_occurrence_region: {
                                   if (buff_index==3) { // this is executed every 4 cycles.
                buff_index=0;
                int tmp;
                my_func(buff,tmp);
                stream_output.write(tmp);
          }else {
                buff_index++;
          }
        }
    }
}
```

Because of this lines, it cannot achieve II=1, but it is executed not every cycle.

set_directive_occurrence

- Used when pipelining functions or loops, to specify that the code in a location is executed at a lesser rate than the code in the enclosing function or loop.

```
void top( hls::stream<int> &stream_input, hls::stream<int> &stream_output) {
    int i,buff[4];
    ap_uint<2> buff_index=0;
loop_a: for ( i=0 ; i<16; i++ ) {
#pragma HLS pipeline II=1
        buff[buff_index]=stream_input.read();
my_occurrence_region: {
                                    if (buff_index==3) {
#pragma HLS OCCURRENCE cycle=4
                buff_index=0;
                int tmp;
                my_func(buff,tmp);
                stream_output.write(tmp);
          }else {
                buff_index++;
          }
        }
    }
}
```

but done every 4 cycles.

set_directive_occurrence

- Used when pipelining functions or loops, to specify that the code in a location is executed at a lesser rate than the code in the enclosing function or loop.

```
void top( hls::stream<int> &stream_input, hls::stream<int> &stream_output) {
    …
loop_a: for ( i=0 ; i<16; i++ ) {
#pragma HLS pipeline II=1
        buff[buff_index]=stream_input.read();
my_occurrence_region: {
                                    if (buff_index==3) {
#pragma HLS OCCURRENCE cycle=4
                                                    …
          }else {
               buff_index++;
          }
        }
    }
}
```

but done every 4 cycles.

Performance(latency)	**59**
Resources(FFs/LUTs)	385/423

[Without directives]

Performance(latency)	**27**
Resources(FFs/LUTs)	286/387

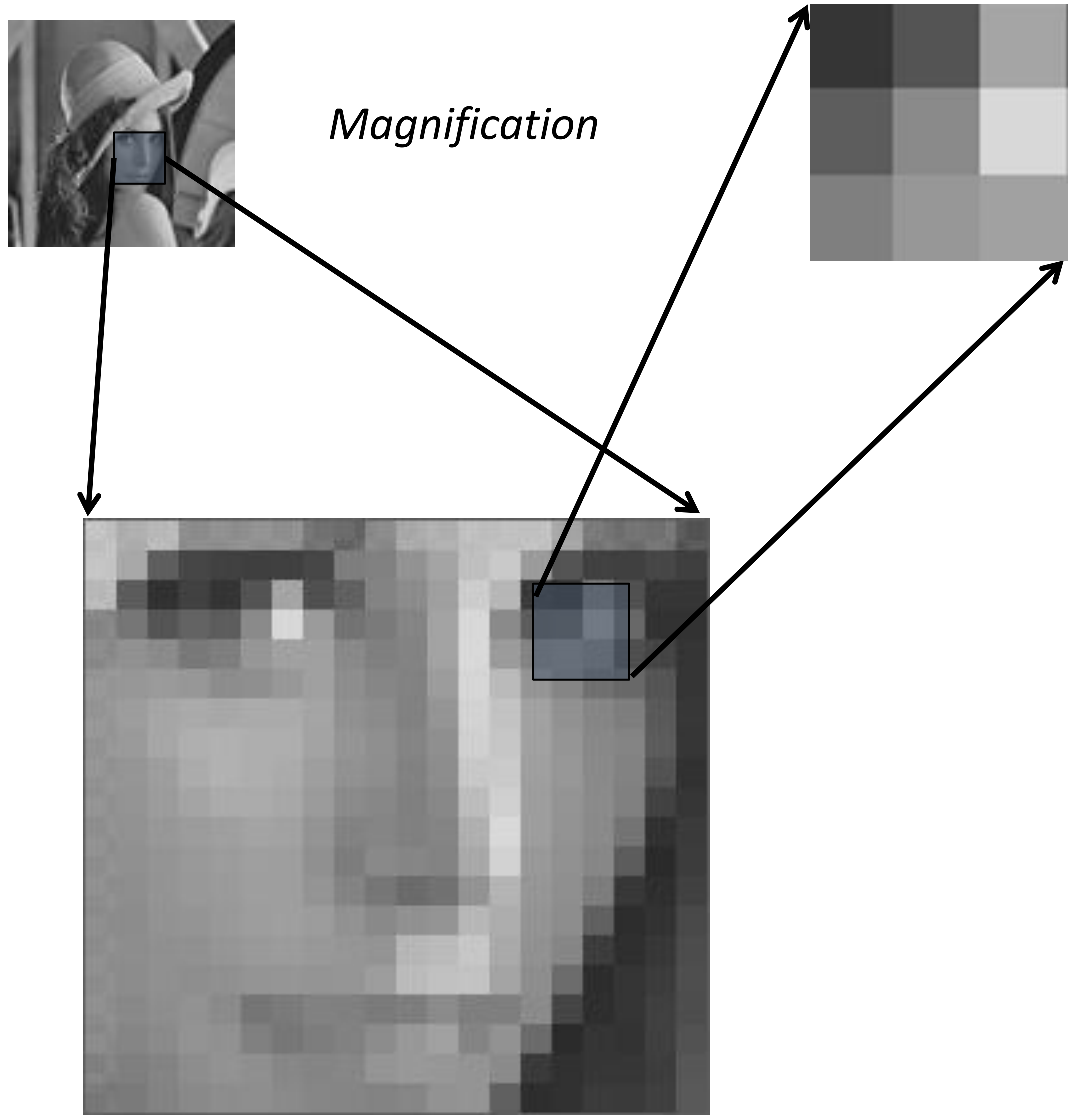
[With directives]

Summary

```
set_directive_allocation             - Directive ALLOCATION
set_directive_array_map              - Directive ARRAY_MAP
set_directive_array_partition        - Directive ARRAY_PARTITION
set_directive_array_reshape          - Directive ARRAY_RESHAPE
set_directive_data_pack              - Directive DATA_PACK
set_directive_dataflow               - Directive DATAFLOW
set_directive_dependence             - Directive DEPENDENCE
set_directive_expression_balance     - Directive EXPRESSION_BALANCE
set_directive_function_instantiate - Directive FUNCTION_INSTANTIATE
set_directive_inline                 - Directive INLINE
set_directive_interface              - Directive INTERFACE
set_directive_latency                - Directive LATENCY
set_directive_loop_flatten           - Directive LOOP_FLATTEN
set_directive_loop_merge             - Directive LOOP_MERGE
set_directive_loop_tripcount         - Directive LOOP_TRIPCOUNT
set_directive_occurrence             - Directive OCCURRENCE
set_directive_pipeline               - Directive PIPELINE
set_directive_protocol               - Directive PROTOCOL
set_directive_reset                  - Directive RESET
set_directive_resource               - Directive RESOURCE
set_directive_stream                 - Directive STREAM
set_directive_top                    - Directive TOP
set_directive_unroll                 - Directive UNROLL
```

Convolution Kernel Design on an FPGA

Convolution: Software Sobel Code



-1	0	+1
-2	0	+2
-1	0	+1

+1	+2	+1
0	0	0
-1	-2	-1

G_X = 1(-1) + 2*0 + 3*1 +…+ = 8*

G_Y = 1(-1) + 2*2 +3*1 + … + = -32*

$\mathbf{G} = sqrt(\mathbf{G\_X}^2 + \mathbf{G\_Y}^2)$

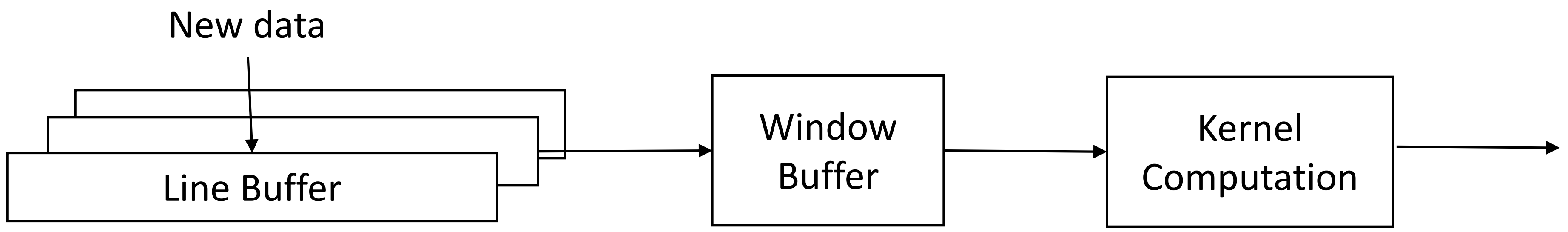
```
for(int i = 0; i < rows; i++){
   for(int j=0; j < cols; j++){
      Gx = 0;
      Gy = 0;
      for(int rowOffset = -1; rowOffset <= 1; rowOffset++){
         for(int colOffset = -1; colOffset <=1; colOffset++){
            Gx = Gx + …;
            Gy = Gy + …;
            G = …;
         }
      }
   }
}
```

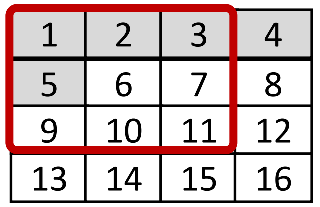
Convolution: Software

```
for(int i = 0; i < rows; i++){
   for(int j=0; j < cols; j++){
      Gx = 0;
      Gy = 0;
      for(rowOffset = -1; rowOffset <= 1; rowOffset++){
         for(colOffset = -1; colOffset <=1; colOffset++){
            Gx = Gx + …;
            Gy = Gy + …;
            G = …;
         }
      }
   }
}
```

Based on Xilinx tutorial: "Zynq all programmable soc sobel filter implementation using the vivado hls tool."

Convolution: Hardware Architecture





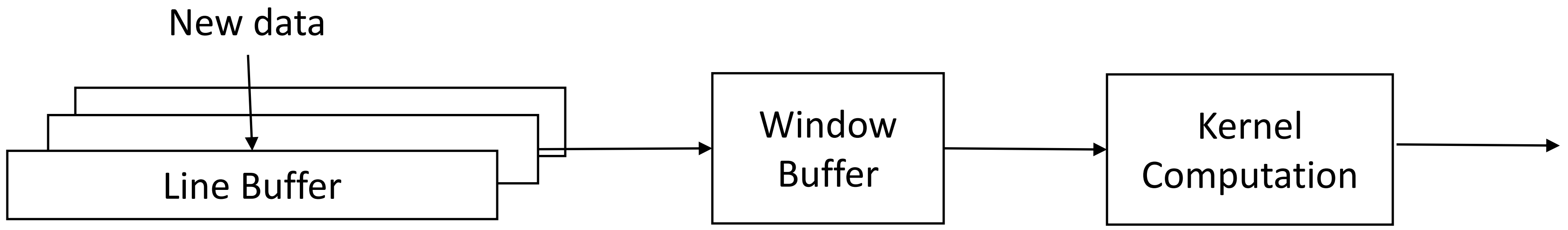
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

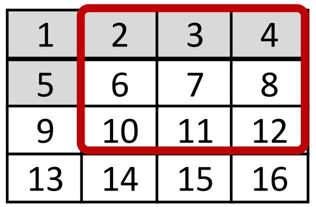
Input Image

Window Buffer 1

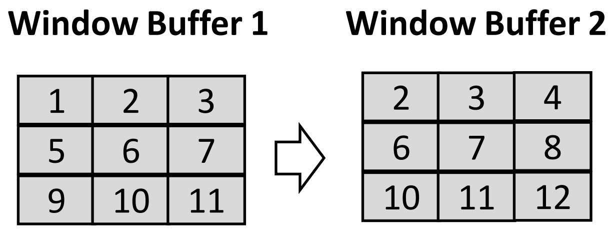
1	2	3
5	6	7
9	10	11

Convolution: Hardware Architecture

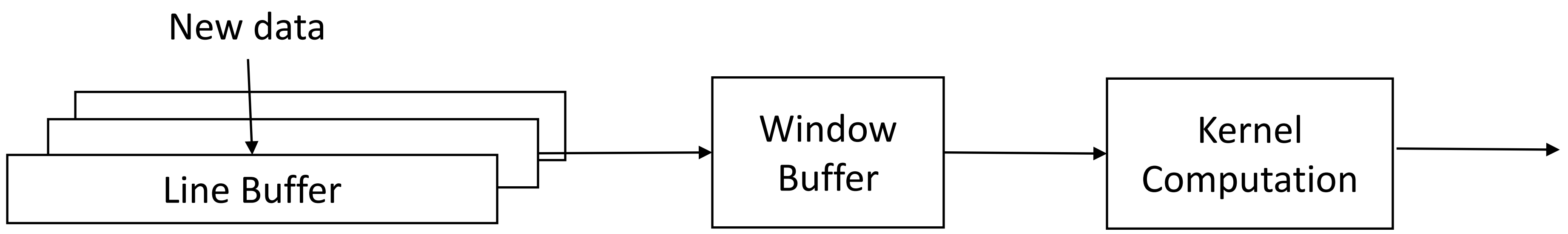




Input Image



Convolution: Hardware Architecture



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Input Image

Line Buffer at time *t*

Line 1	1	2	3	4
Line 2	5	6	7	8
Line 3	9	10	11	12

Window Buffer 1

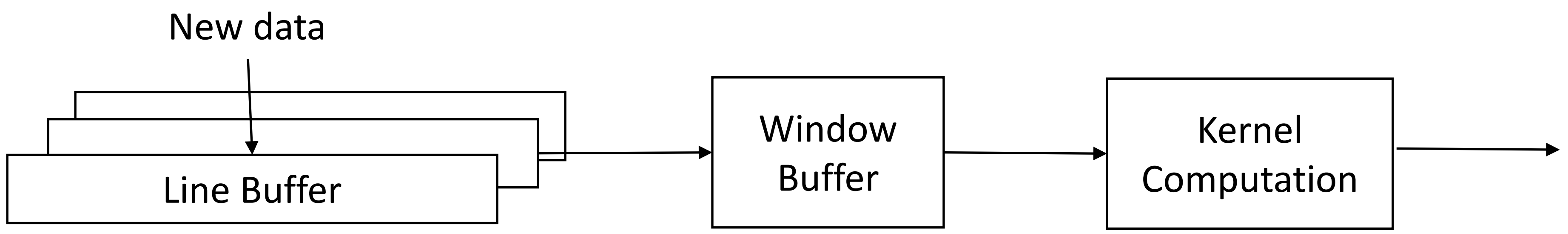
1	2	3
5	6	7
9	10	11

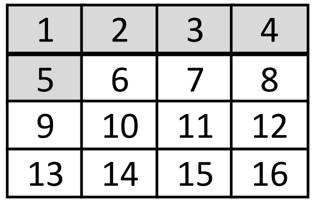
$\Rightarrow$

Window Buffer 2

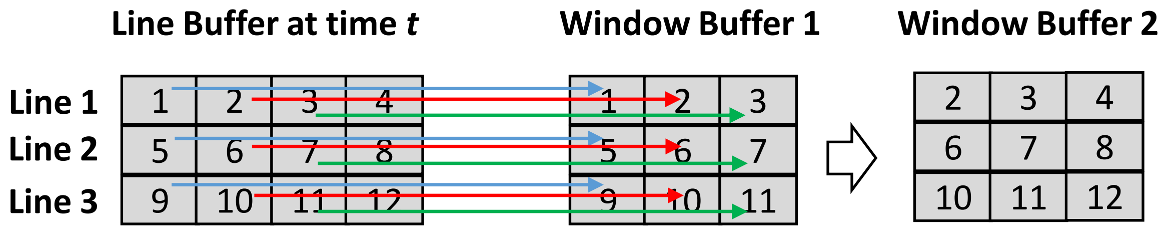
2	3	4
6	7	8
10	11	12

Convolution: Hardware Architecture

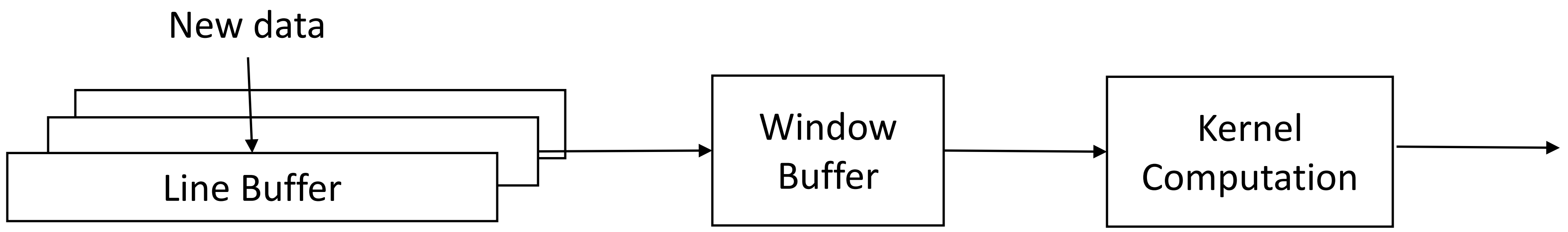


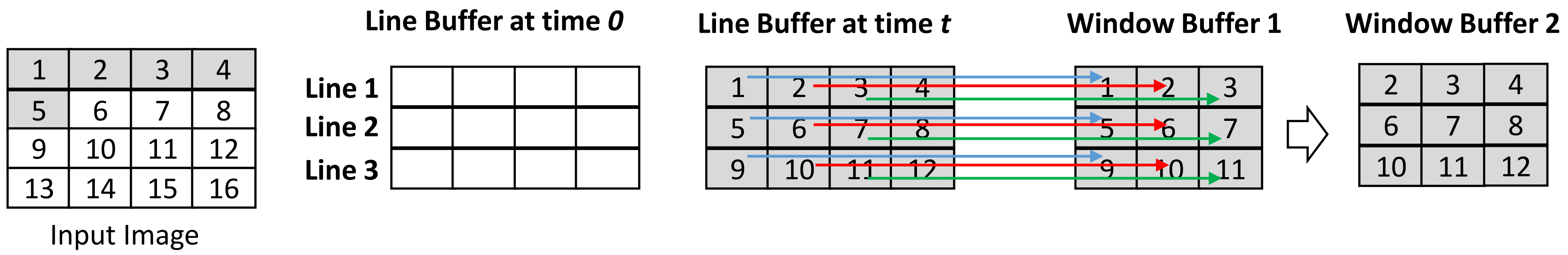


Input Image

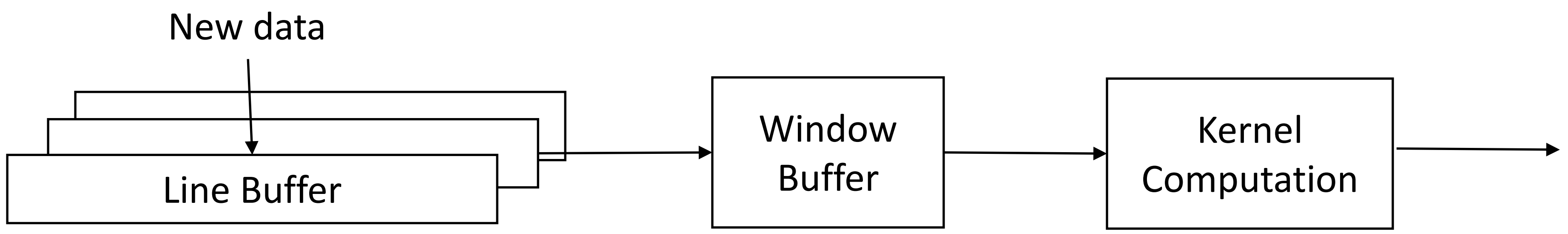


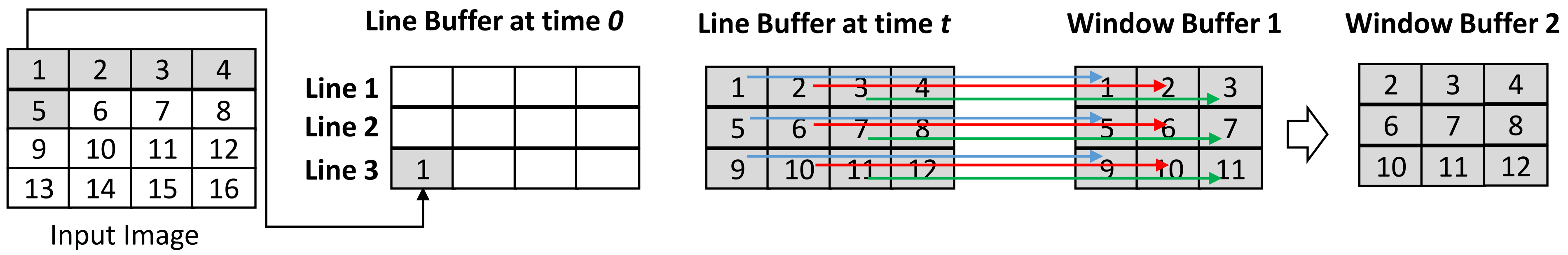
Convolution: Hardware Architecture



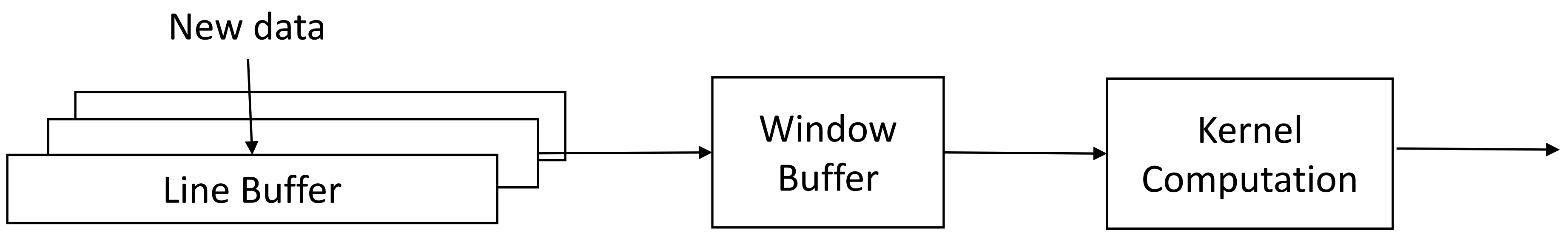


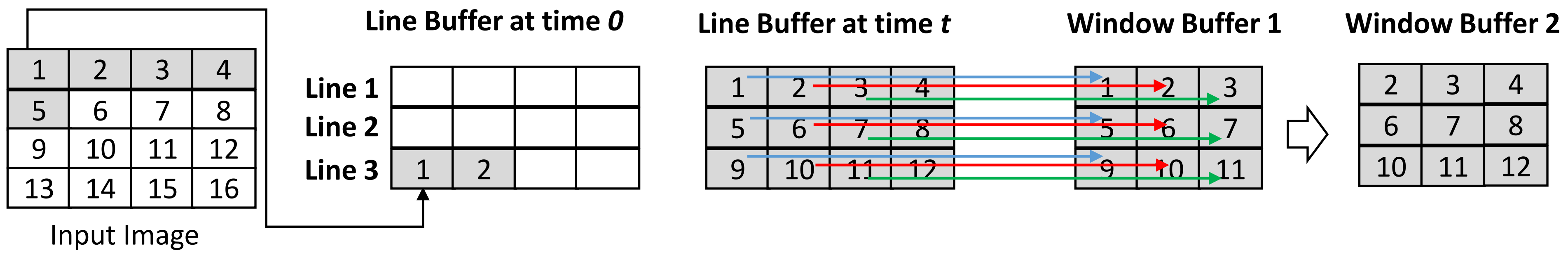
Convolution: Hardware Architecture



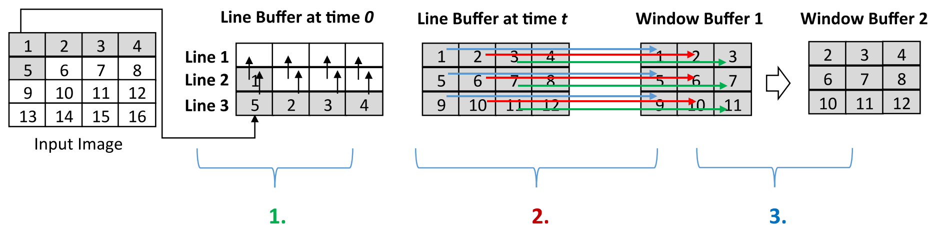


Convolution: Hardware Architecture





Convolution: Hardware



```
sobel_filter (WindowBuffer) {

for(int i=0; i<kernel_size; i++)
   for(int j=0; j<cols; j++)
      #pragma pipeline
     // Do  kernel computation using Window Buffer
      result = ,...
    }
}
return result;
```

```
int LineBuffer[3][IMG_W];
int WindowBuffer[3][3];

for(int i=0; i<rows; i++)
   for(int j=0; j<cols; j++)
      #pragma pipeline

      LineBuffer[0][j]=LineBuffer[1][j];
      LineBuffer[1][j]=LineBuffer[2][j];
      LineBuffer[2][j]=input_image[i][j];

      WindowBuffer[0][0] = LineBuffer[0][j];
      WindowBuffer[1][0] = LineBuffer[1][j];
      WindowBuffer[2][0] = LineBuffer[2][j];

      for(int k = 0; k < 3; k++) {
         WindowBuffer[k][2] = WindowBuffer[k][1];
         WindowBuffer[k][1] = WindowBuffer[k][0];
      }
      sobel_filter(WindowBuffer);
```

Convolution as a Matrix Multiplication

- Chellapilla, Kumar, Sidd Puri, and Patrice Simard. "High performance convolutional neural networks for document processing." *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.