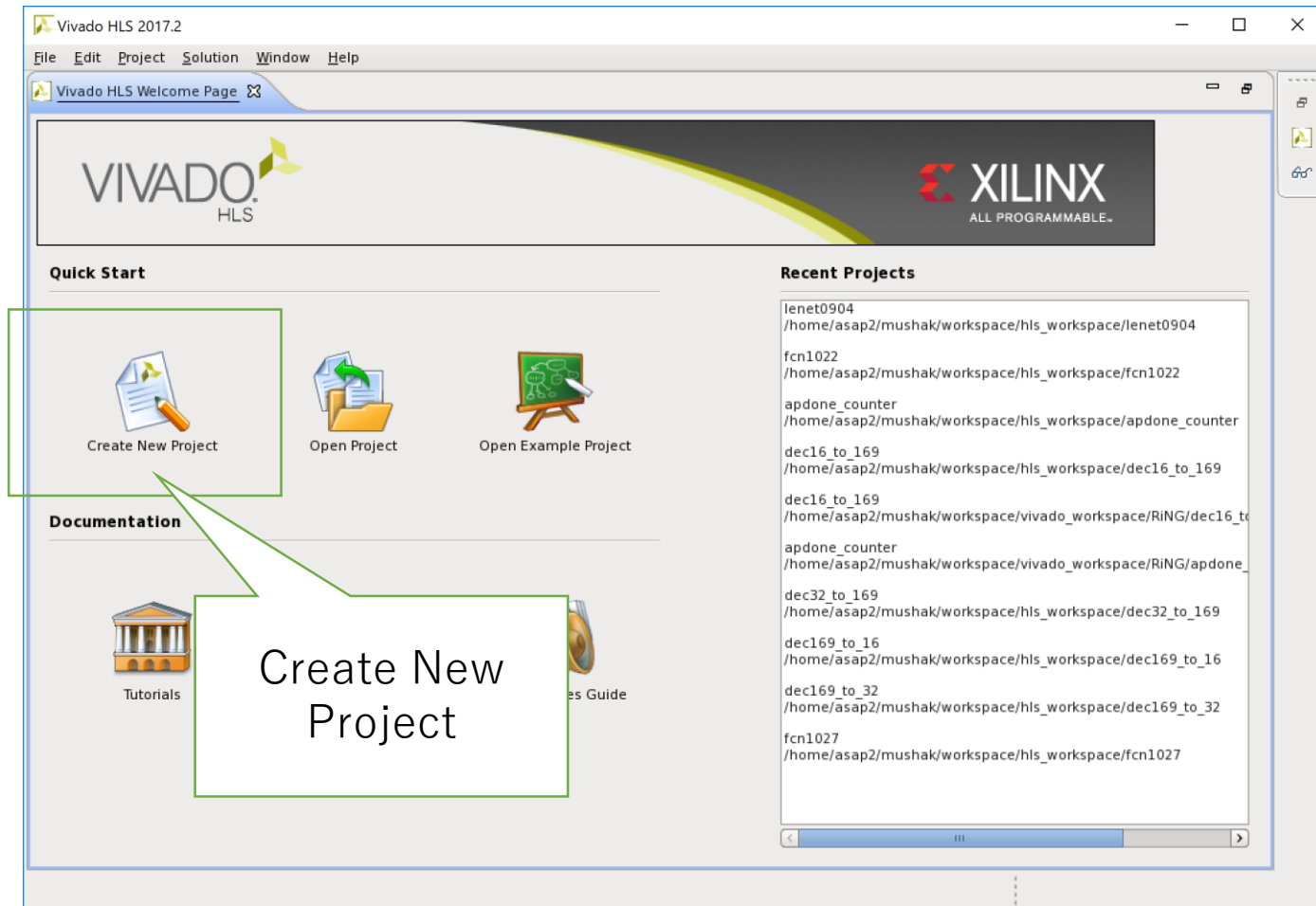


VivadoHLS design

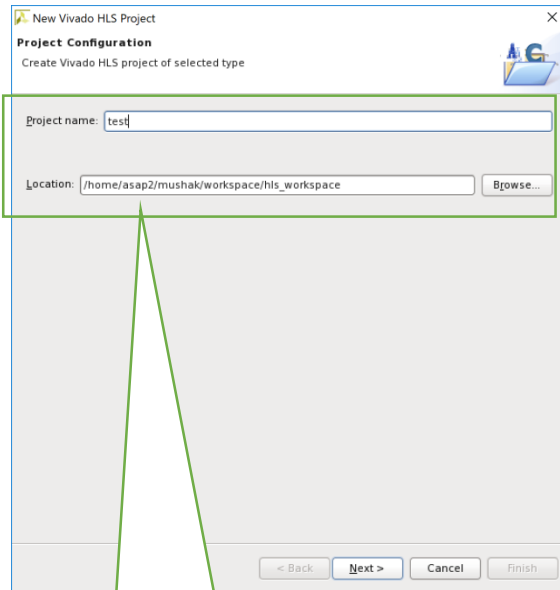
Kazusa Musha

2019.3.18

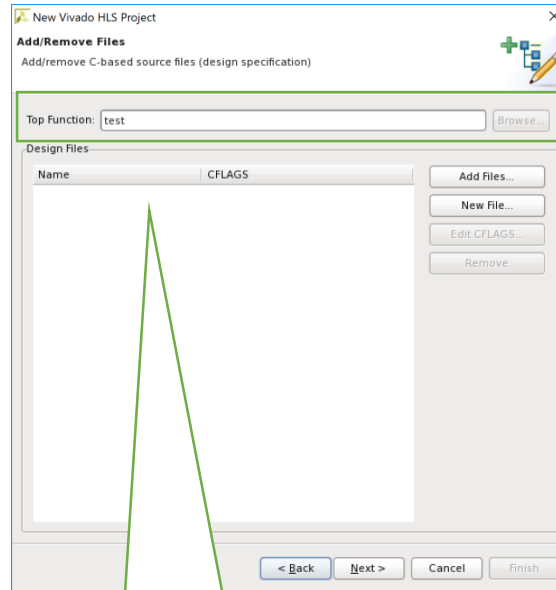
How to create a new project (1)



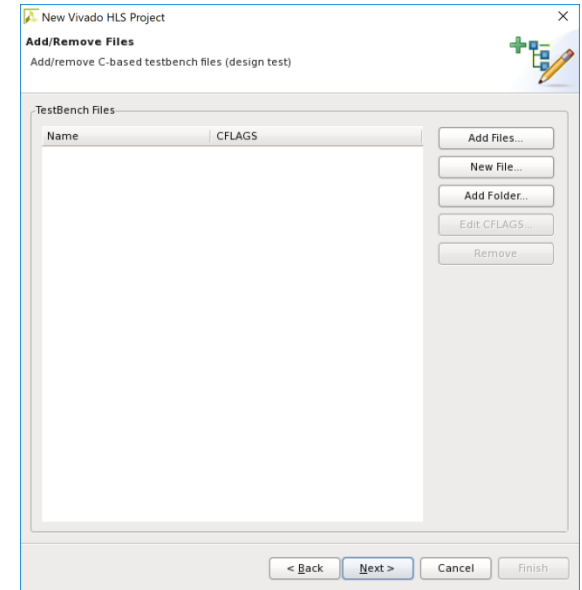
How to create a new project (2)



Project name and directory are specified. Here, "test" is used.

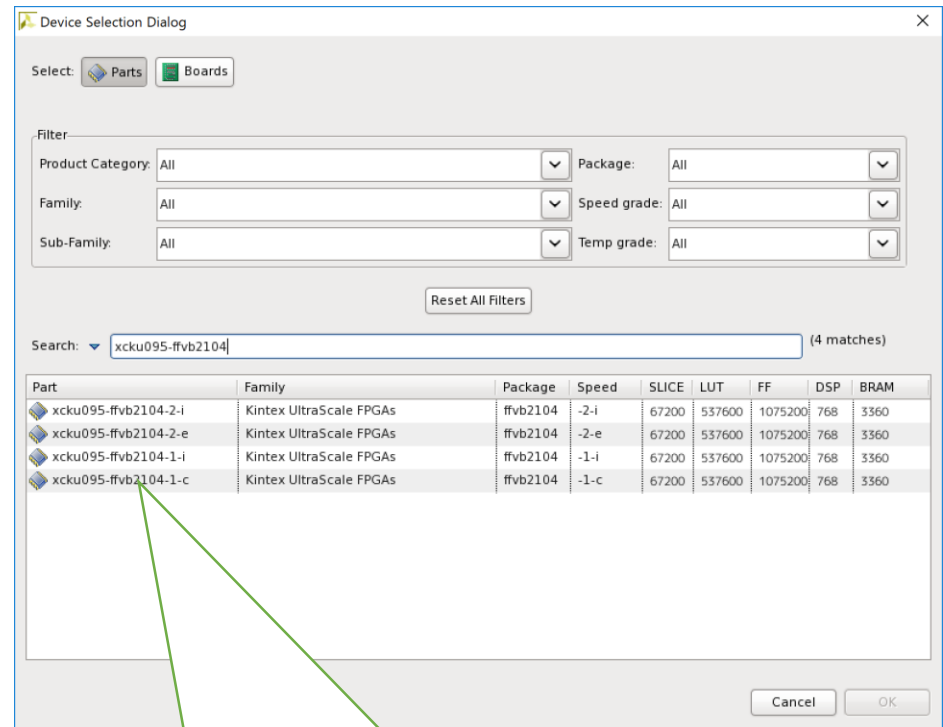
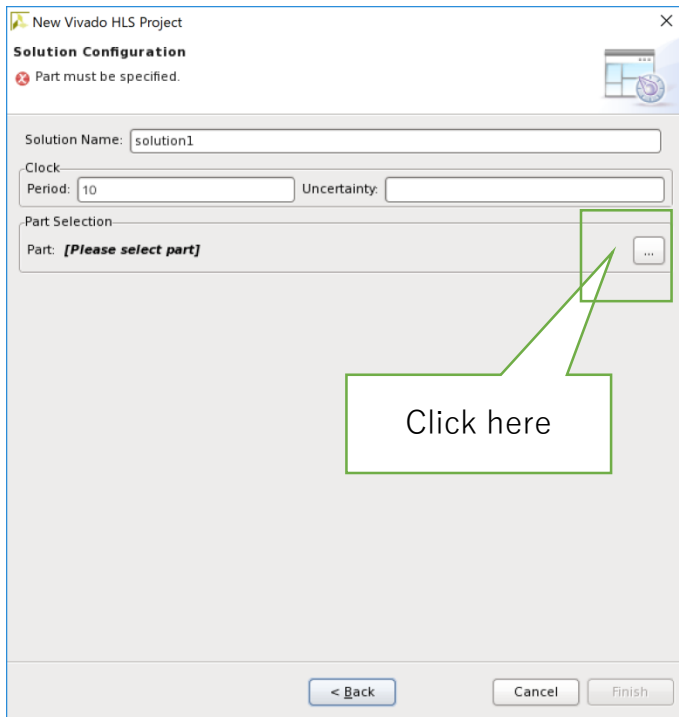


Top function name. Here, test is used.

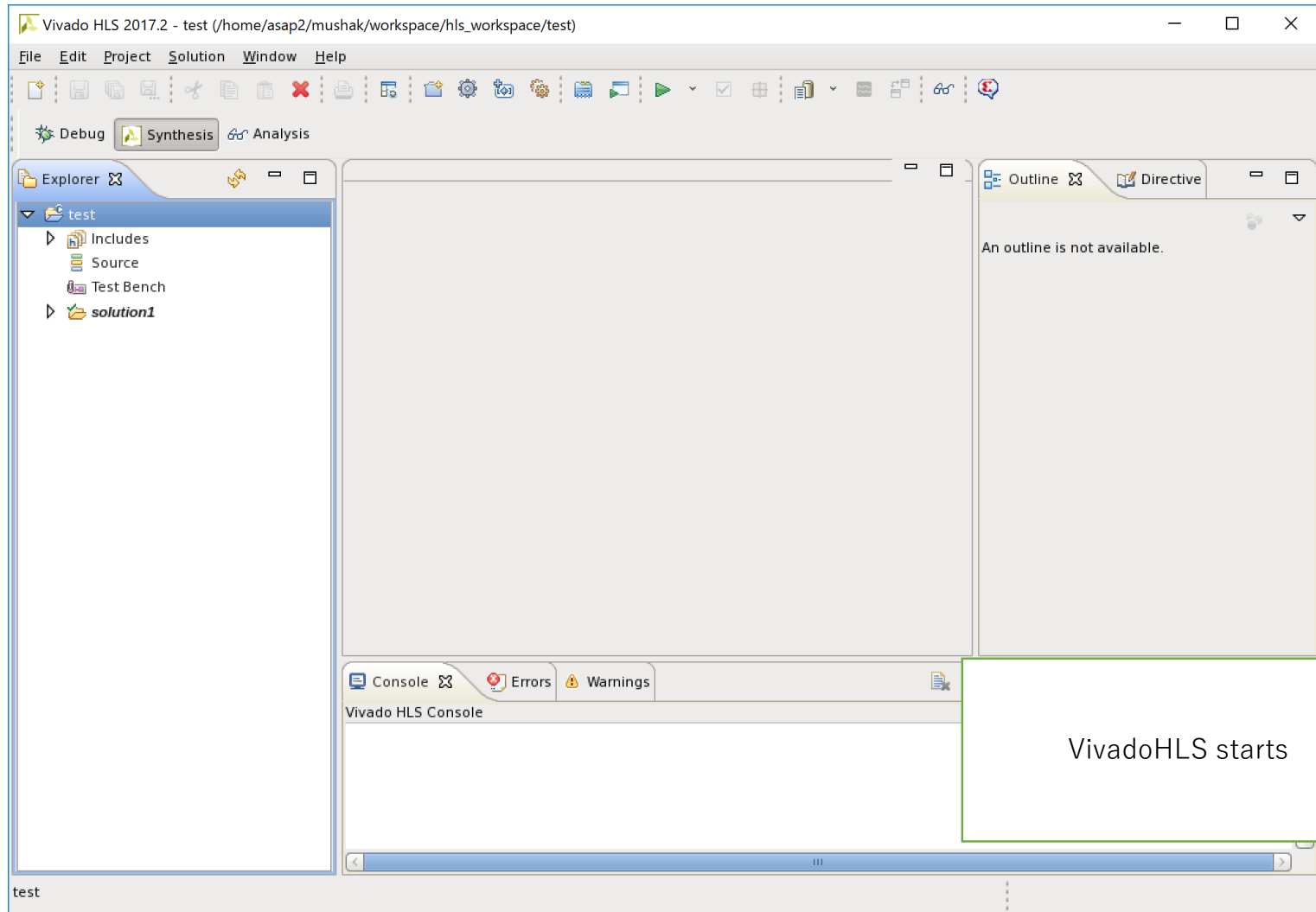


Test bench file is specified. It can be added later, so here, it is empty.

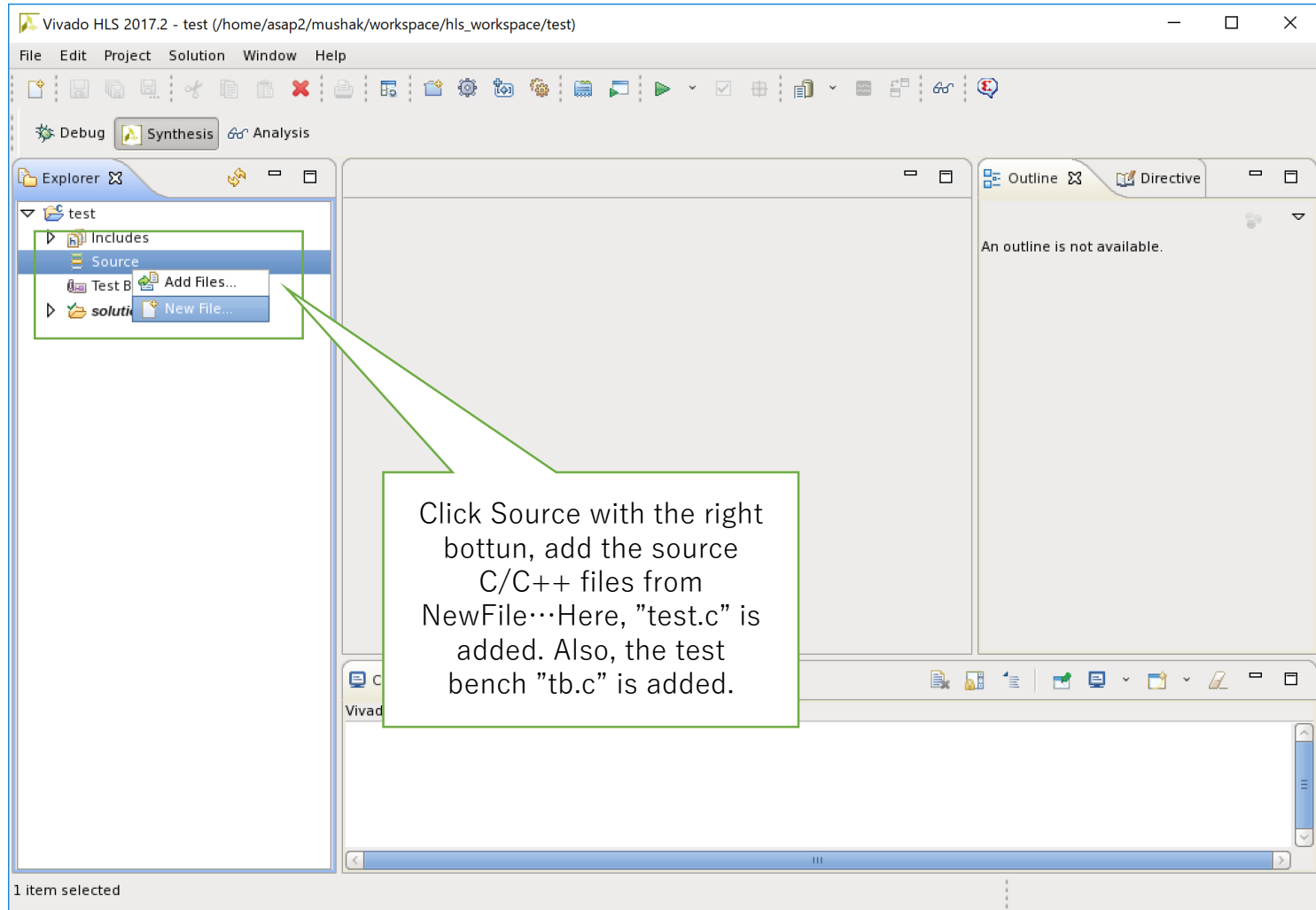
How to create a new project (3)



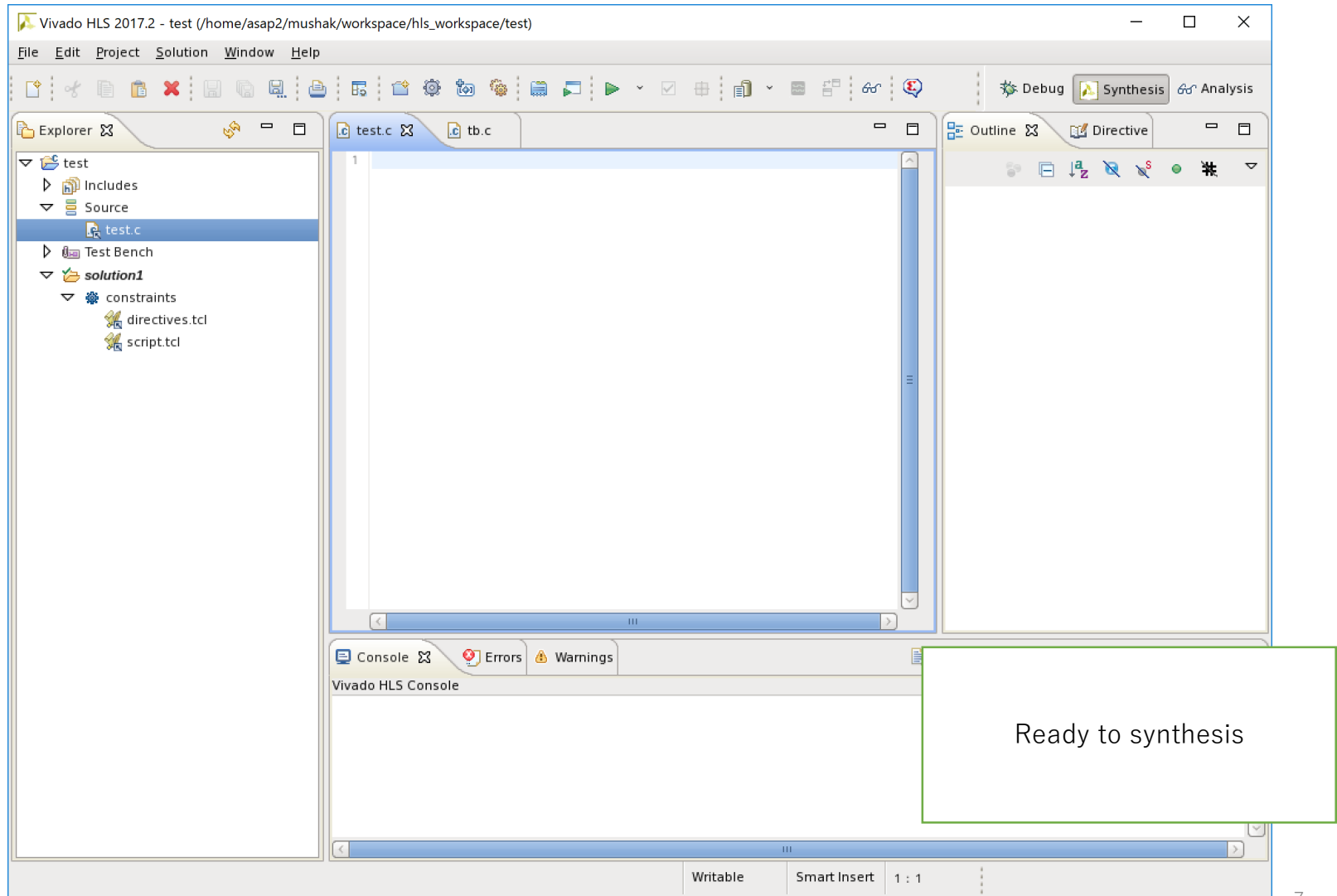
How to create a new project (4)



How to create a new project (5)



How to create a new project (6)



How to use GUI

The image shows the Vivado HLS 2017.2 GUI with several callouts explaining the workflow:

- <<Synthesis>> Basic tab for high level synthesis**: Points to the Synthesis button in the toolbar.
- <<Analysis>> Analysis of the synthesized results**: Points to the Analysis button in the toolbar.
- <<Directive>> Insertion**: Points to the Directive button in the toolbar.
- Simulation with the test bench**: Points to the Run Simulation button in the toolbar.
- Synthesis the source and check results**: Points to the Synthesis button in the toolbar.
- IP is generated from the synthesized result.**: Points to the IP button in the toolbar.

A large green arrow labeled **Design Flow** points from the Explorer window to the Synthesis button. Another large green arrow points from the IP button to the Vivado HLS Console window.

VivadoHLS design flow

$N \times M$ Product-sum operation

- $N \times M$ product-sum operation
($N=12$, $M=12$)

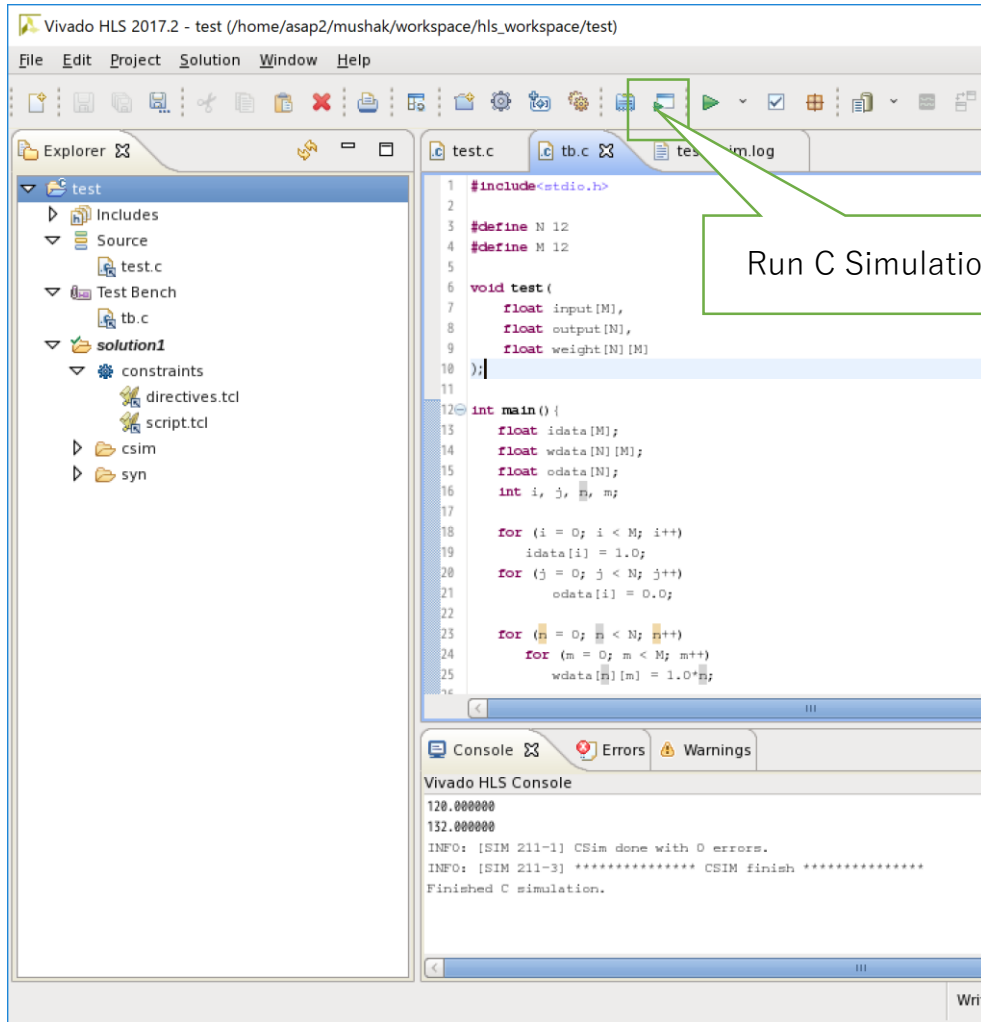
Similar to computation in
LeNet, the contest this year.

```
#define N 12
#define M 12

void test(
    float input[M],
    float output[N],
    float weight[N][M]
){
    int n, m;

    for(n = 0; n < N; n++)
        for(m = 0; m < M; m++)
            output[n] += input[m]
                * weight[n][m];
}
```

C-level simulation(1/2)



- Simulation must be done before synthesis
 - “Run C Simulation”
 - Call “test.c” from the test bench “tb.c” like the normal C design.
 - When ap_cint.h header (free- size integer) is used, simulation will cause errors.

C-level simulation(2/2)

test.c

```
#define N 12
#define M 12

void test(
    float input[M],
    float output[N],
    float weight[N][M]
){
    int n, m;

    for(n = 0; n < N; n++)
        for(m = 0; m < M; m++)
            output[n] += input[m]
                * weight[n][m];
}
```

tb.c

```
#include<stdio.h>

#define N 12
#define M 12

void test(
    float input[M],
    float output[N],
    float weight[N][M]
);

int main(){
    float idata[M];
    float wdata[N][M];
    float odata[N];
    int i, j, n, m;
    for (i = 0; i < M; i++) idata[i] = 1.0;
    for (j = 0; j < N; j++) odata[j] = 0.0;

    for (n = 0; n < N; n++)
        for (m = 0; m < M; m++)
            wdata[n][m] = 1.0*n;

    test(idata, odata, wdata);

    for (j = 0; j < N; j++)
        printf("%f\n", odata[j]);

    return 0;
}
```

test_csim.log

```
1 INFO: [SIM 2] ***
2 INFO: [SIM 4] CSI
3 make: `csim.exe'
4 0.000000
5 12.000000
6 24.000000
7 36.000000
8 48.000000
9 60.000000
10 72.000000
11 84.000000
12 96.000000
13 108.000000
14 120.000000
15 132.000000
16 INFO
17 INFO
```

If the results are correct, go to synthesis.

Try Synthesis

The screenshot displays the Vivado HLS 2017.2 interface. The main editor shows a C file named `test.c` with the following code:

```
1 #define N 12
2 #define M 12
3
4 void test (
5     float input[M],
6     float output[N],
7     float weight[N][M]
8 ){
9     int n, m;
10
11     for(n = 0; n < N; n++)
12         for(m = 0; m < M; m++)
13             output[n] += input[m]
14                 * weight[n][m];
15 }
16
```

The Vivado HLS Console at the bottom shows the following output:

```
Vivado HLS Console
120.000000
132.000000
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] ***** CSIM finish *****
Finished C simulation.
```

Annotations in the image include a green box around the Synthesis button in the toolbar with a callout box containing the text "Click here", and another green box around the console output with a callout box containing the text "The results are generated for a while."

Check the synthesis results(1/2)

The screenshot displays the Vivado HLS 2017.2 interface. The Explorer pane on the left shows the project structure for 'test', including source files (test.c, tb.c), test benches, and a solution named 'solution1' with constraints and scripts. The main window shows the 'Synthesis Report for test' with the following sections:

General Information

- Date: Mon Dec 3 15:21:50 2018
- Version: 2017.2 (Build 1909853 on Wed Aug 09 16:50:22 MDT 2017)
- Project: test
- Solution: solution1
- Product: kintexu
- Target device: xcku095-ffvb2104-1-c

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.57	1.25

Latency (clock cycles)

Summary

Latency	Interval			
min	max	min	max	Type
1321	1321	1322	1322	none

Console

```
Vivado HLS Console
INFO: [SYSC 207-301] Generating SystemC RTL for test.
INFO: [VHDL 208-304] Generating VHDL RTL for test.
INFO: [VLOG 209-307] Generating Verilog RTL for test.
INFO: [HLS 200-112] Total elapsed time: 11.81 seconds; peak allocated
Finished C synthesis.
```

• Performance

• Timing

- Unit is “ns”. If it is less than 10ns=100MHz, it’s OK.

• Latency

- Response time from data input to data output.

• Interval

- Interval of data inputs. The time from a data-set input to the next data-set input.

• Utilization Estimates

- FPGA Resource utilization
- BRAM,DSP,FF, and LUT must be in 100%.

• Interface

- Inputs/Outputs of the module

Check the synthesis results(2/2)

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.57	1.25

Latency		Interval		
min	max	min	max	Type
1321	1321	1322	1322	none

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	98	54
FIFO	-	-	-	-
Instance	-	5	355	352
Memory	-	-	-	-
Multiplexer	-	-	-	75
Register	-	-	134	-
Total	0	5	587	481
Available	3360	768	1075200	537600
Utilization (%)	0	~0	~0	~0

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	test	return value
ap_rst	in	1	ap_ctrl_hs	test	return value
ap_start	in	1	ap_ctrl_hs	test	return value
ap_done	out	1	ap_ctrl_hs	test	return value
ap_idle	out	1	ap_ctrl_hs	test	return value
ap_ready	out	1	ap_ctrl_hs	test	return value
input_r_address0	out	4	ap_memory	input_r	array
input_r_ce0	out	1	ap_memory	input_r	array
input_r_q0	in	32	ap_memory	input_r	array
output_r_address0	out	4	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	32	ap_memory	output_r	array
output_r_q0	in	32	ap_memory	output_r	array
weight_address0	out	8	ap_memory	weight	array
weight_ce0	out	1	ap_memory	weight	array
weight_q0	in	32	ap_memory	weight	array

- Timing is satisfied. No problem.
- Latency is 1177cycles, and Interval is 1178 cycles. The smaller the better.
- Utilization is almost 0.
- Interface:
 - ap** are for control. Address and data are provided for input,output, and weight.
 - The memory is outside the module, and the data are fetched in advance.

Check the results (Analysis)

Analysisタブ

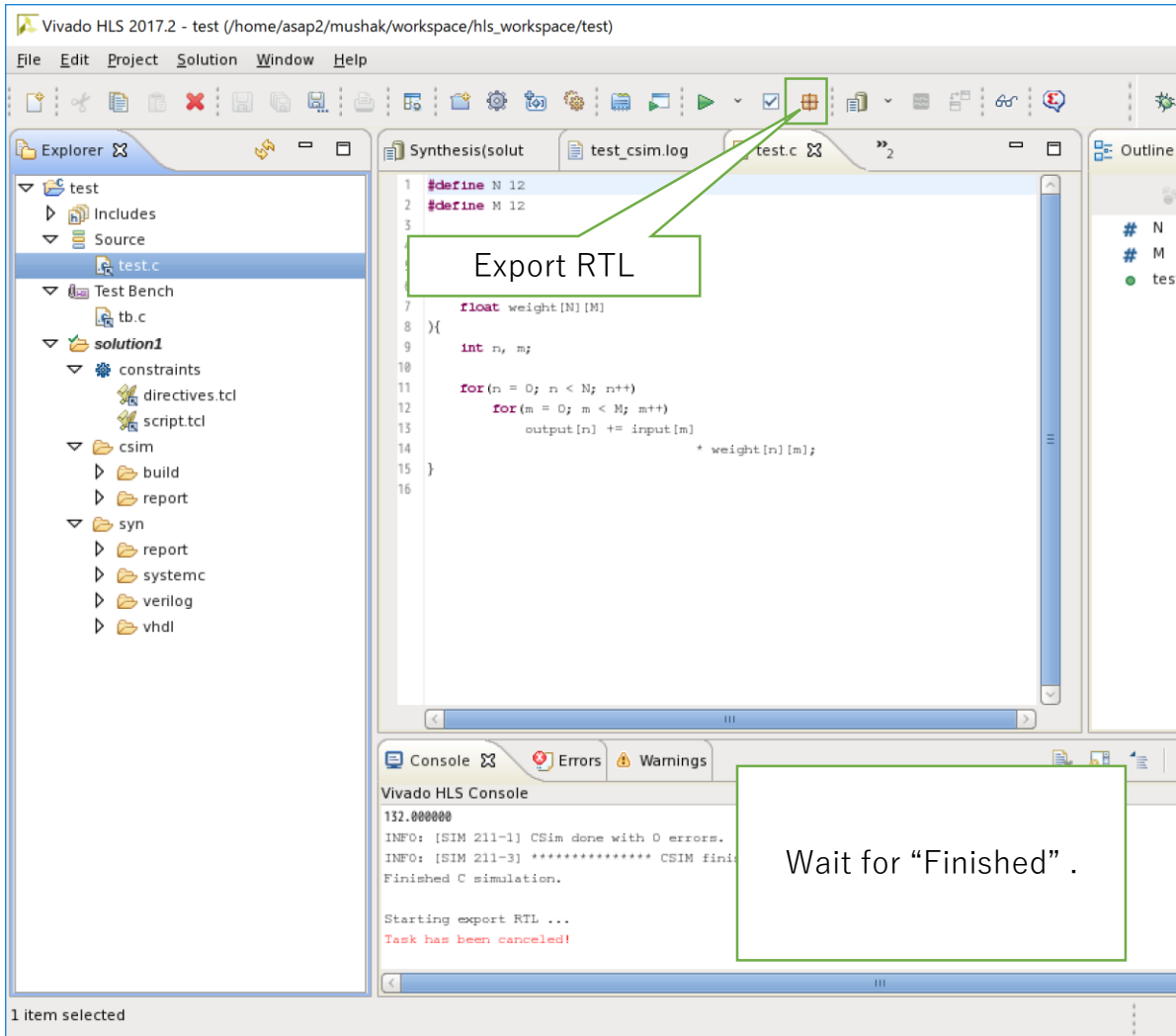
Operation/Control Step	C0	C1	C2	C3	C4	C5	C6	C7
1-16 Loop 1								

Module	BRAM	DSP	FF	LUT	Latency	Ir
test	0	5	587	481	1321	1

Module	Pipelined	Latency	Initiation
test	-	1321	1322
Loop 1	no	1320	-
Loop 1	no	108	-

- Analysis tab is used to analyze the module in detail.
 - The timing of data load is shown. It is useful for debugging.
 - Latency or interval of each module(for loop or function) is checked.
 - Useful for optimization shown later.

Generating IP (Export RTL)



- “Export RTL” generates IP useful by the IP-catalog in Vivado.
- This step is done after debugging.

Directives

What are directives ? (Pragma)

- Commands for optimization of the HLS description
 - UNROLL, PIPELINE, DATAFLOW, etc...
- Various types solutions can be implemented.
 - tips: multiple solutions can be generated.
- Key factors to optimize HLS.

High speed implementation using directives

- The target source code is optimized.
- The I/O bandwidth(input,weight, and output) is assumed up to 128bit/cycle

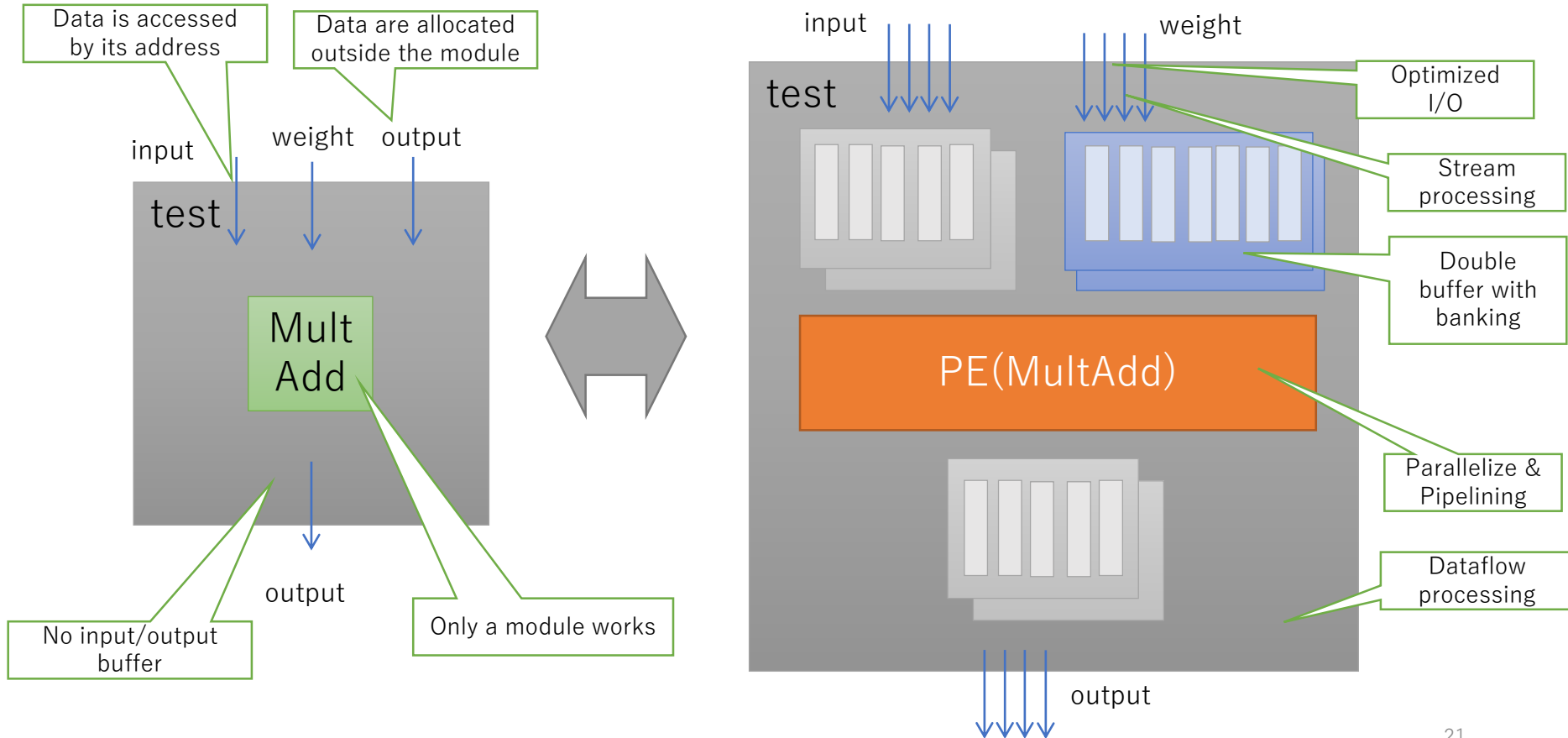
```
#define N 12
#define M 12

void test(
    float input[M],
    float output[N],
    float weight[N][M]
){
    int n, m;

    for(n = 0; n < N; n++)
        for(m = 0; m < M; m++)
            output[n] += input[m]
                * weight[n][m];
}
```

The target image

- Comparing the current design and the target image.



#pragma HLS UNROLL (1/3)

- HLS UNROLL unrolls the loop statically.
 - Without unrolling, only a statement is executed in a loop.
 - Moreover, branch processing takes another cycle in each loop.
- factor controls the degree of unrolling
 - 4 means four loops executed in parallel.
 - Undividable factor causes an error
 - If factor is not specified, the loop is completely unrolled.
 - Note that too many unrolling may cause an memory error.

```
void test(  
    float input[M],  
    float output[N],  
    float weight[N][M]  
)  
{  
    int n, m;  
  
    for(n = 0; n < N; n++)  
        for(m = 0; m < M; m++)  
            #pragma HLS UNROLL factor=4  
                output[n] += input[m]  
                    * weight[n][m];  
}
```

#pragma HLS UNROLL (2/3)

- Let's synthesize and compare

```
void test(
  float input[M],
  float output[N],
  float weight[N][M]
){
  int n, m;

  for(n = 0; n < N; n++)
    for(m = 0; m < M; m++)
      output[n] += input[m]
                 * weight[n][m];
}
```



```
void test(
  float input[M],
  float output[N],
  float weight[N][M]
){
  int n, m;

  for(n = 0; n < N; n++)
    for(m = 0; m < M; m++)
      #pragma HLS UNROLL factor=4
      output[n] += input[m]
                 * weight[n][m];
}
```

Performance improvement has been achieved.

← Interval is reduced from 1 3 2 2 cycles to 7 8 2 cycles

← Instead, the resource is increased.

Latency		Interval		
min	max	min	max	Type
1321	1321	1322	1322	none

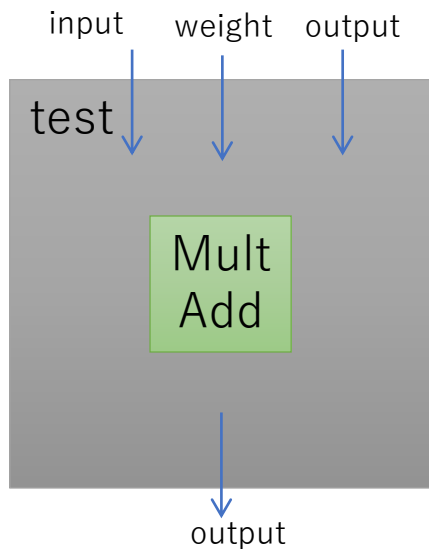
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	98	54
FIFO	-	-	-	-
Instance	-	5	355	352
Memory	-	-	-	-
Multiplexer	-	-	-	75
Register	-	-	134	-
Total	0	5	587	481
Available	3360	768	1075200	537600
Utilization (%)	0	~0	~0	~0

Latency		Interval		
min	max	min	max	Type
781	781	782	782	none

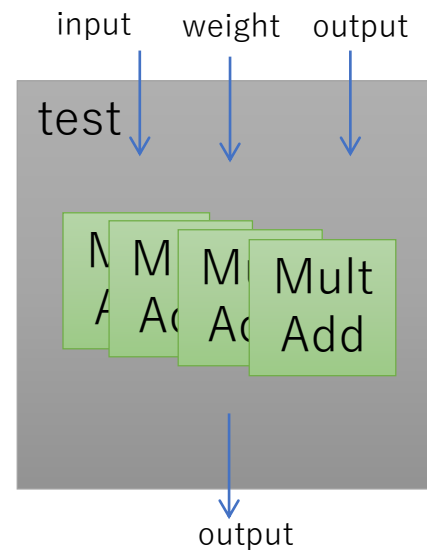
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	194	111
FIFO	-	-	-	-
Instance	-	5	355	352
Memory	-	-	-	-
Multiplexer	-	-	-	200
Register	-	-	178	-
Total	0	5	727	663
Available	3360	768	1075200	537600
Utilization (%)	0	~0	~0	~0

#pragma HLS UNROLL (3/3)

- Comparison of the structures
 - The parallel processing with 4 Mult-Add increases the performance.



Without directives



With directives

#pragma HLS PIPELINE (1/2)

- HLS PIPELINE increases the throughput by pipelining.
 - Frequently used like as well as UNROLL.
 - Simple performance improvement is obtained.
 - It is efficient when used at the upper hierarchy of UNROLL-ed loop.
 - Note that the UNROLL inside the PIPELINE-ed loop executes the complete unrolling.
 - factor is ignored.

```
void test(  
    float input[M],  
    float output[N],  
    float weight[N][M]  
)  
{  
    int n, m;  
  
    for(n = 0; n < N; n++)  
#pragma HLS PIPELINE  
        for(m = 0; m < M; m++)  
#pragma HLS UNROLL  
            output[n] += input[m]  
                        * weight[n][m];  
}
```

Complete
Unroll

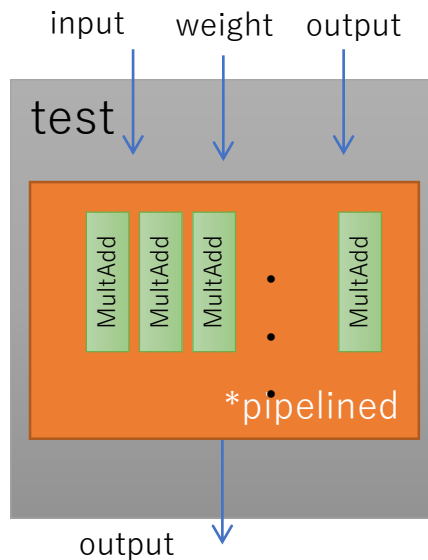


Latency		Interval		
min	max	min	max	Type
121	121	122	122	none

#pragma HLS PIPELINE (2/2)

- HLS PIPELINE can be applied to the function as well as loops.
 - All UNROLL in the function becomes the complete unrolling.

The current image



```

void test(
    float input[M],
    float output[N],
    float weight[N][M]
){
    #pragma HLS PIPELINE
    int n, m;

    for(n = 0; n < N; n++)
        #pragma HLS UNROLL
        for(m = 0; m < M; m++)
            #pragma HLS UNROLL
            output[n] += input[m]
                * weight[n][m];
}
    
```



Latency		Interval		
min	max	min	max	Type
79	79	80	80	function

Exercise

- Optimize the example design 'test' by using directives shown here.
- Try to evaluate the clock cycles after the synthesis.

Small module based design

- The functional module is changed into the form of “pe” here.
- Small module based design is suitable for providing buffer and dataflow design.

```
void test(  
    float input[M],  
    float output[N],  
    float weight[N][M]  
)  
{  
    pe(input, output, weight);  
}
```

```
void pe(float input[M], float output[N], float  
weight[N][M]){  
    #pragma HLS PIPELINE  
    int n, m;  
    for(n = 0; n < N; n++)  
        #pragma HLS UNROLL  
        for(m = 0; m < M; m++)  
            #pragma HLS UNROLL  
            output[n] += input[m] * weight[n][m];  
}
```

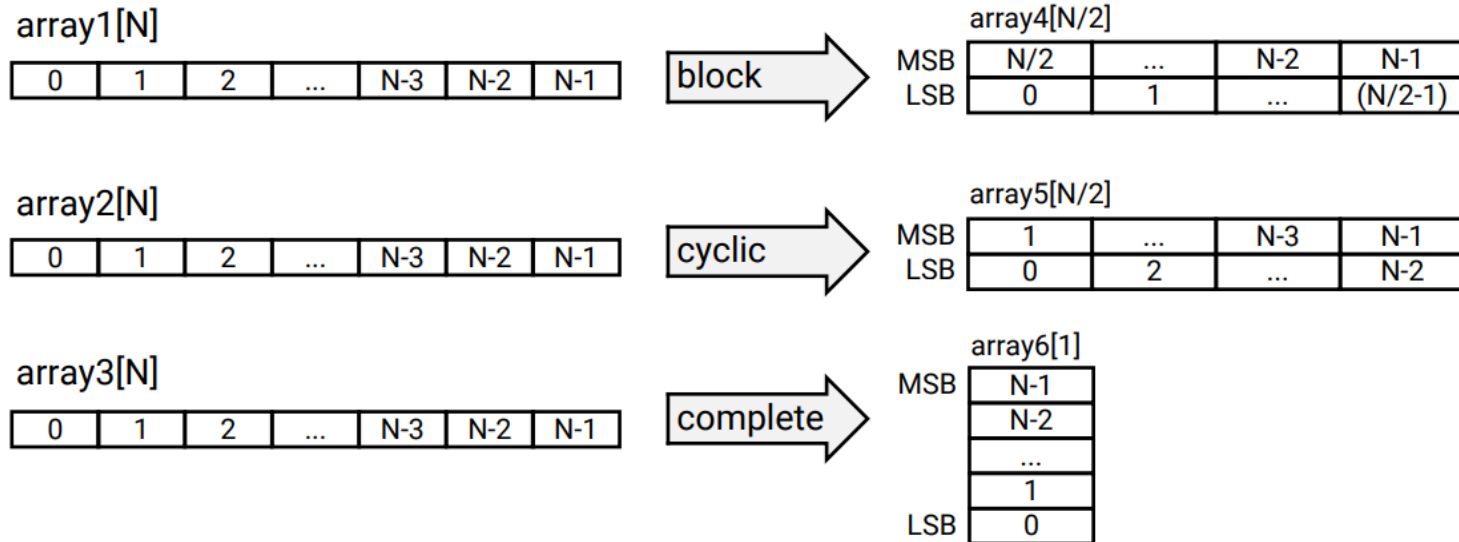
#pragma HLS ARRAY_PARTITION (1/2)

- HLS ARRAY_PARTITION divides the array (input,output,etc.) for increasing the parallelism.
 - Usually, array division is automatically done by HLS.
 - Here, it is used to divide the input/output completely.
 - dim specifies the dimension which the partition is applied. 0 means all dimension.

```
void pe(float input[M], float output[N], float weight[N][M]){  
    #pragma HLS ARRAY_PARTITION variable=input complete dim=0  
    #pragma HLS ARRAY_PARTITION variable=output complete dim=0  
    #pragma HLS ARRAY_PARTITION variable=weight complete dim=0  
    #pragma HLS PIPELINE  
    int n, m;  
    for(n = 0; n < N; n++)  
        #pragma HLS UNROLL  
        for(m = 0; m < M; m++)  
            #pragma HLS UNROLL  
            output[n] += input[m] * weight[n][m];  
}
```

#pragma HLS ARRAY_PARTITION (2/2)

- There are three ways of ARRAY_PARTITION
 - Please refer Xilinx's document UG1270



X14307-110217

Vivado HLS 最適化手法ガイド UG1270 (v2017.4) 67ページより引用

https://www.xilinx.com/support/documentation/sw_manuals_j/xilinx2017_4/ug1270-vivado-hls-opt-methodology-guide.pdf

Trial and error on PE design (1/2)

- For high performance computation, the way of computation itself is changed.
- The current problem: the array “output” has both input/output -> Too much amount of communication.
 - Initialize “output” in the module (“read” out the output can be eliminated.)
 - Registers are provided inside the PE to store the data to store the data temporally.

Trial and error on PE design (2/2)

```
void pe(float input[M], float output[N], float weight[N][M]){  
#pragma HLS ARRAY_PARTITION variable=input complete dim=0  
#pragma HLS ARRAY_PARTITION variable=output complete dim=0  
#pragma HLS ARRAY_PARTITION variable=weight complete dim=0  
#pragma HLS PIPELINE  
  int n, m, o, i;  
  float output_reg[N];  
  
  for (i = 0; i < N; i++)  
#pragma HLS UNROLL  
    output_reg[i] = 0.0;  
  
  for(n = 0; n < N; n++)  
#pragma HLS UNROLL  
    for(m = 0; m < M; m++)  
#pragma HLS UNROLL  
      output_reg[n] += input[m] * weight[n][m];  
  
  for(o = 0; o < N; o++)  
    output[o] = output_reg[o];  
}
```

Now, PE is
OK.



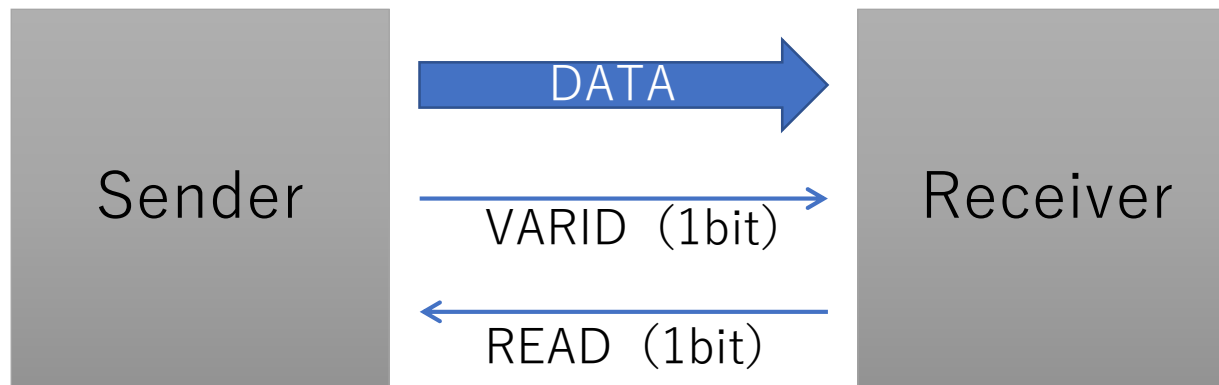
Latency		Interval		
min	max	min	max	Type
50	50	51	51	none

Optimization of data transfer

- The performance of PE has been enhanced. But the total effect is not so much.
- This is because the I/O bandwidth and internal communication bandwidth are not enough
- Let's enhance them.

AXI Stream

- AXI is a standard interface for Xilinx's FPGA
 - Master-Slave
- AXI Stream is a stream style interface included in the AXI



Refer the manual for detail

#pragma HLS INTERFACE (1/2)

- HLS INTERFACE specifies the interface explicitly.
 - Without it, address-data interface is automatically assigned.
 - The following description causes an error

- AXI Stream only accepts input/output along with the order of array index.

```
void test(  
    float input[M],  
    float output[N],  
    float weight[N][M]  
)  
{  
    #pragma HLS INTERFACE axis port=input  
    #pragma HLS INTERFACE axis port=output  
    #pragma HLS INTERFACE axis port=weight  
    pe(input, output, weight);  
}
```

They cause an error

#pragma HLS INTERFACE (2/2)

- tmp buffer is introduced for streaming access.

```
void test(  
    float input[M],  
    float output[N],  
    float weight[N][M]  
)  
{  
    #pragma HLS INTERFACE axis port=input  
    #pragma HLS INTERFACE axis port=output  
    #pragma HLS INTERFACE axis port=weight  
    float input_tmp[M], output_tmp[N], weight_tmp[N][M];  
  
    load_weight(weight, weight_tmp);  
    load_input(input, input_tmp);  
  
    pe(input_tmp, output_tmp, weight_tmp);  
  
    store_output(output_tmp, output);  
}
```

```
void load_weight(float weight[N][M], float  
weight_tmp[N][M]){  
    #pragma HLS PIPELINE  
    int n, m;  
    for(n = 0; n < N; n++)  
    #pragma HLS UNROLL  
        for(m = 0; m < M; m++)  
    #pragma HLS UNROLL  
            weight_tmp[n][m] = weight[n][m];  
}
```

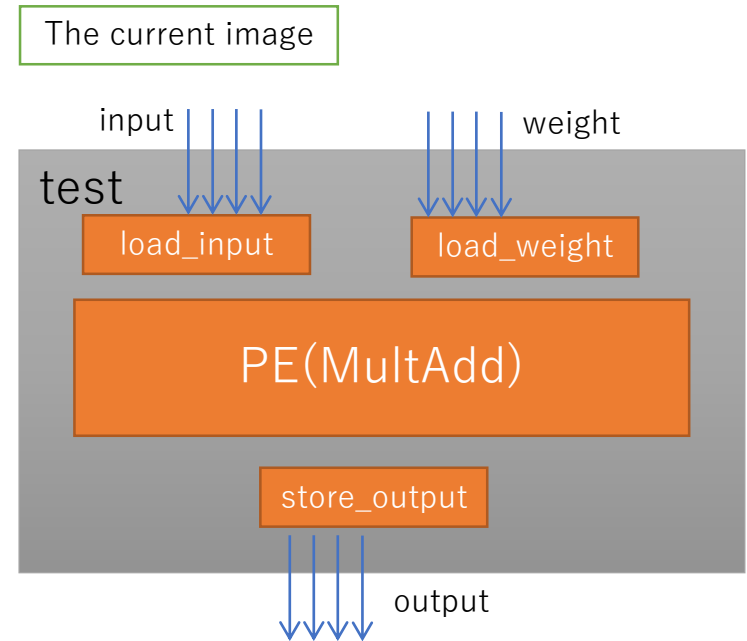
```
void load_input(float input[M], float input_tmp[M]){  
    #pragma HLS PIPELINE  
    int m;  
    for(m = 0; m < M; m++)  
    #pragma HLS UNROLL  
        input_tmp[m] = input[m];  
}
```

```
void store_output(float output_tmp[N], float  
output[N]){  
    #pragma HLS PIPELINE  
    int n;  
    for(n = 0; n < N; n++)  
    #pragma HLS UNROLL  
        output[n] = output_tmp[n];  
}
```

The bandwidth extension

- For extending the bandwidth, HLS ARRAY_PARTITION cyclic is convenient.
 - factor is set to be four in order to extend the bandwidth 128bits.

```
void test(float input[M], float output[N], float weight[N][M]){  
  #pragma HLS INTERFACE axis port=input  
  #pragma HLS INTERFACE axis port=output  
  #pragma HLS INTERFACE axis port=weight  
  #pragma HLS ARRAY_PARTITION variable=input cyclic  
  factor=4 dim=1  
  #pragma HLS ARRAY_PARTITION variable=output cyclic  
  factor=4 dim=1  
  #pragma HLS ARRAY_PARTITION variable=weight cyclic  
  factor=4 dim=2  
  
  float input_tmp[M], output_tmp[N], weight_tmp[N][M];  
  
  load_weight(weight, weight_tmp);  
  load_input(input, input_tmp);  
  pe(input_tmp, output_tmp, weight_tmp);  
  store_output(output_tmp, output);  
}
```



#pragma HLS DATAFLOW (1/2)

- HLS DATAFLOW is a directive to generate dataflow with loops and functions.
 - Each loop and function is independent module and data are transferred between them.
 - Each function is connected with FIFO or PIPO (double buffer) .
 - Only a function can read/write from/to an argument (*_tmp, here)
 - It is quite natural considering the hardware structure.

```
void test(float input[M], float output[N], float
weight[N][M]){
#pragma HLS INTERFACE axis port=input
#pragma HLS INTERFACE axis port=output
#pragma HLS INTERFACE axis port=weight
#pragma HLS ARRAY_PARTITION variable=input cyclic
factor=4 dim=1
#pragma HLS ARRAY_PARTITION variable=output cyclic
factor=4 dim=1
#pragma HLS ARRAY_PARTITION variable=weight cyclic
factor=4 dim=2

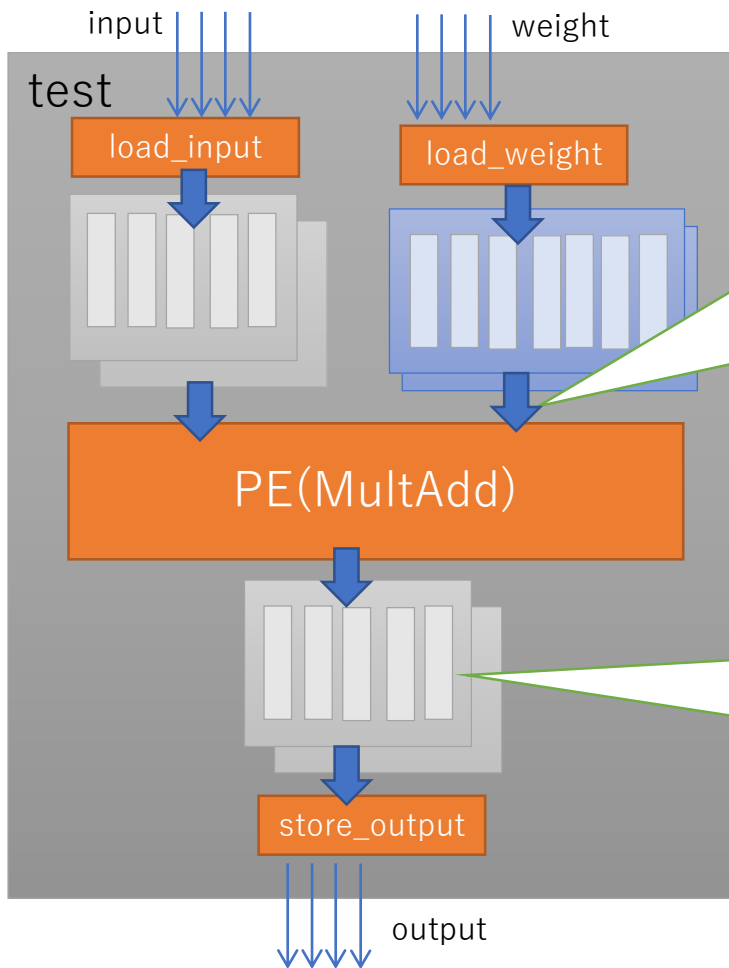
#pragma HLS DATAFLOW

float input_tmp[M], output_tmp[N], weight_tmp[N][M];

load_weight(weight, weight_tmp);
load_input(input, input_tmp);
pe(input_tmp, output_tmp, weight_tmp);
store_output(output_tmp, output);
}
```

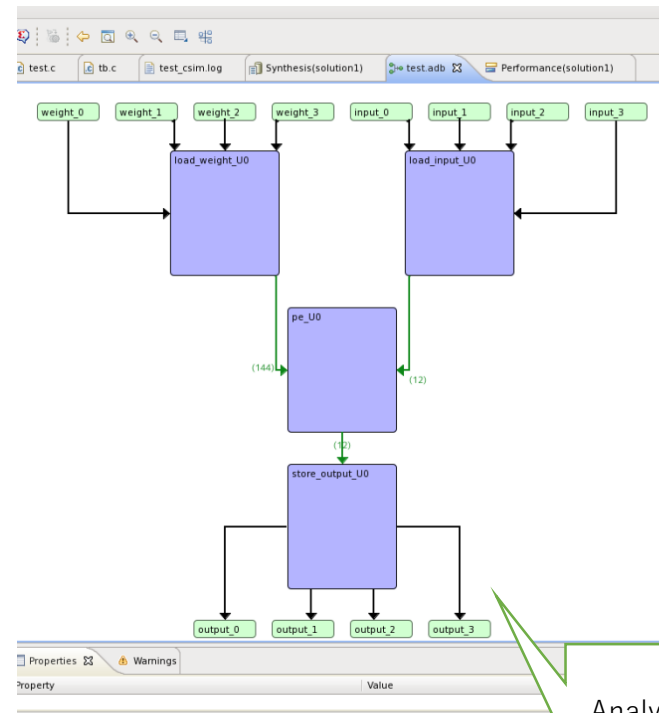
#pragma HLS DATAFLOW (2/2)

- The generated image



Now branch or reverse flow. DATAFLOW is suitable for the streaming application.

Array data between functions are transferred through the PIPO.



Analysis can visualize the dataflow structure.

Analyzing the performance

- Interval is only 3 6 cycles.
 - 37x performance compared to the original version.

[-] Timing (ns)

[-] Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.29	1.25

[-] Latency (clock cycles)

[-] Summary

Latency		Interval		
min	max	min	max	Type
91	91	36	36	dataflow

[-] Detail

[-] Instance

Instance	Module	Latency		Interval		Type
		min	max	min	max	
pe_U0	pe	50	50	1	1	function
load_weight_U0	load_weight	36	36	36	36	function
store_output_U0	store_output	3	3	3	3	function
load_input_U0	load_input	3	3	3	3	function

• FPGA resource

- DSP is almost sold out.

[-] Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	44	726
FIFO	0	-	0	168
Instance	-	720	79431	58745
Memory	-	-	-	-
Multiplexer	-	-	-	1548
Register	-	-	174	-
Total	0	720	79649	61187
Available	3360	768	1075200	537600
Utilization (%)	0	93	7	11

Review

- HLS UNROLL
 - Loop unrolling
- HLS PIPELINE
 - Pipelining
- HLS ARRAY_PARTITION
 - Array partition explicitly for interleaving and banking.
- HLS INTERFACE
 - Select the interface of a function (AXI Stream, AXI4 Lite, etc...)
- HLS DATAFLOW
 - Data flow implementation of for loops and functions.