

Parallel and Reconfigurable VLSI Computing (7)

# Practical RTL Design

Hiroki Nakahara

Tokyo Institute of Technology

# Outline

- Practical RTL design methodology
  - From behavior (C/C++ code) to HDL one
- Interface co-design
  - Control a hardware from an ARM processor
- RTL design optimization

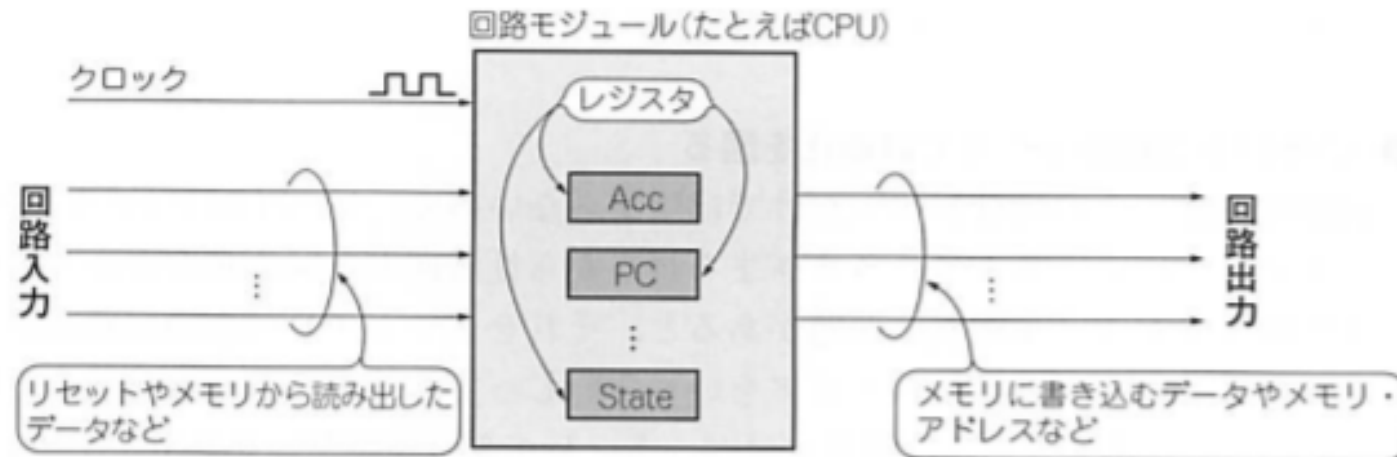
# Practical RTL Design Methodology

# C/C++ to RTL

- Determine the specifications of the circuit
  - Timing chart, state transition diagram, performance, block diagram
- Consider the configuration of the module
  - Design and combine for each IP core
- Function assigned to each core and its resource
  - Including a consideration of interface
  - Often written in C/C ++
  - It becomes a testbench for a verification
- Convert C/C++ description to RTL
- Optimize behaviors (pipelining and parallelization)
  - Automation by the remaining work with CAD

# C/C++ Description for a Concept of Module

- Approximate ~300 lines for single function
- Data input/output (Interface)
- Data processing



# Case Study: FIR Filter

- $x_n$ : 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

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

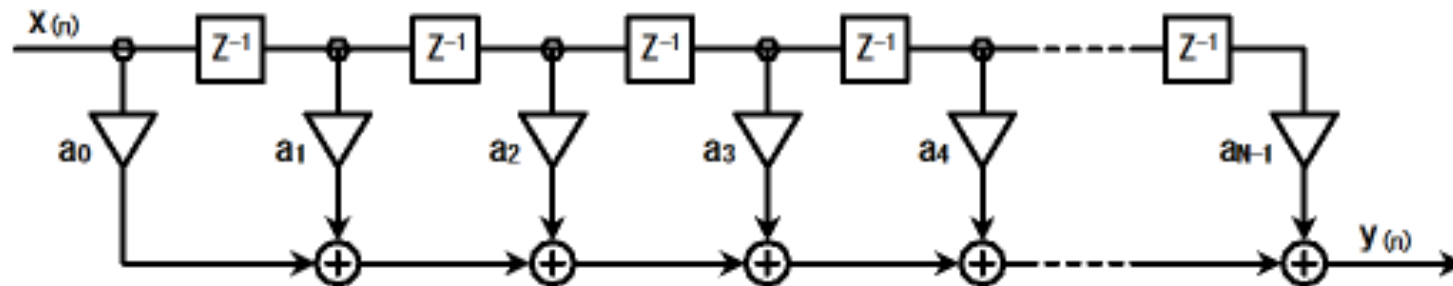
, where  $\tilde{\omega} = 2\pi\omega / \omega_s$  denotes normalized freq.,  $\omega_s$  denotes sampling freq.,  $H_0(\tilde{\omega}) \in \mathbb{R}$  denotes frequency characteristic,  $\rho_n$  denotes window function, and  $\tilde{\Gamma} = (N-1)/2$ .

# Cont'd

- Difference equation for a FIR filter:

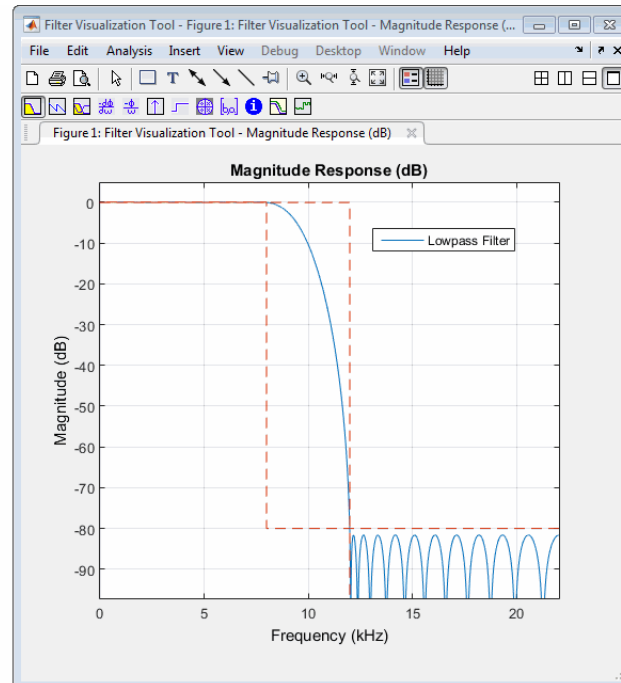
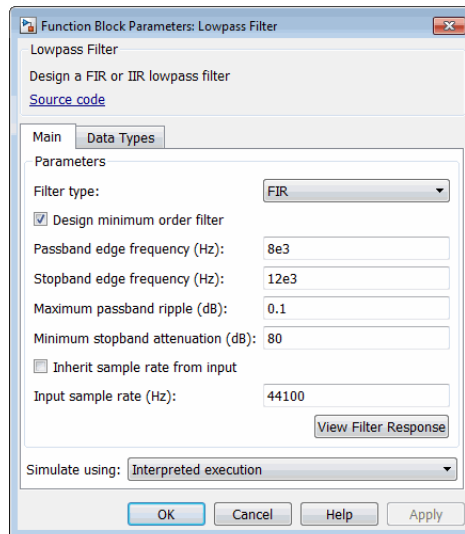
$$y[n] = a_0x[n] + a_1x[n-1] + a_2x[n-2] + a_3x[n-3] + a_4x[n-4] + \dots + a_{N-1}x[n-(N-1)]$$

- Diagram for a FIR filter:



# FIR Filter Coefficient Design

- MathWorks Matlab with DSP System Toolbox
  - Sampling Freq.: 44.1 kHz
  - LPF for 20 kHz → Normalized cut-off freq.
  - #Taps: 11
  - Window function: Hamming





# C Behavior for a FIR Filter

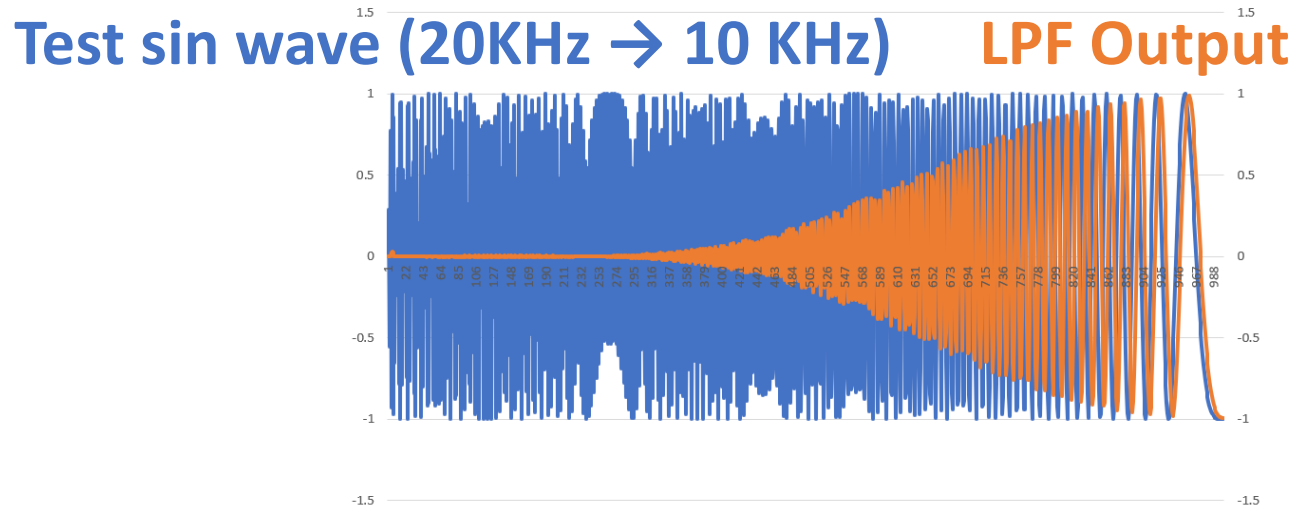
[https://github.com/HirokiNakahara/FPGA\\_lecture/tree/master/Lec7\\_Practical\\_RTL\\_design/fir.c](https://github.com/HirokiNakahara/FPGA_lecture/tree/master/Lec7_Practical_RTL_design/fir.c)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  #define N 11
6
7  void fir(float *y, float x)
8  {
9      float c[N] = { // 0.17 = 20KHz/44.1KHz, LPF, Hamming Window
10         -4.120289718403869e-03, -1.208600321298122e-02,
11         -2.650603053411641e-03,  9.166631627169690e-02,
12         2.544318483405623e-01,  3.400000000000001e-01,
13         2.544318483405623e-01,  9.166631627169690e-02,
14         -2.650603053411641e-03, -1.208600321298122e-02,
15         -4.120289718403869e-03, };
16
17     static float shift_reg[N];
18     float acc;
19     int i;
20
21     acc = 0.0;
22     for (i = N - 1; i >= 0; i--) {
23         if (i == 0) {
24             acc += x * c[0];
25             shift_reg[0] = x;
26         } else {
27             shift_reg[i] = shift_reg[i - 1];
28             acc += shift_reg[i] * c[i];
29         }
30     }
31     *y = acc;
32 }
```

```
34 void main()
35 {
36     float fs = 44100.0;
37     int len = 1000;
38
39     float f0 = 20000.0;
40     float sin_wave;
41     float fir_out;
42
43     int i;
44
45     for( i = 0; i < len; i++){
46         sin_wave = sin( 2.0 * M_PI * f0 * i / fs);
47
48         fir( &fir_out, sin_wave);
49
50         printf("%d %f %f\n", i, sin_wave, fir_out);
51         f0 = f0 - 10.0;
52     }
53 }
```

# 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



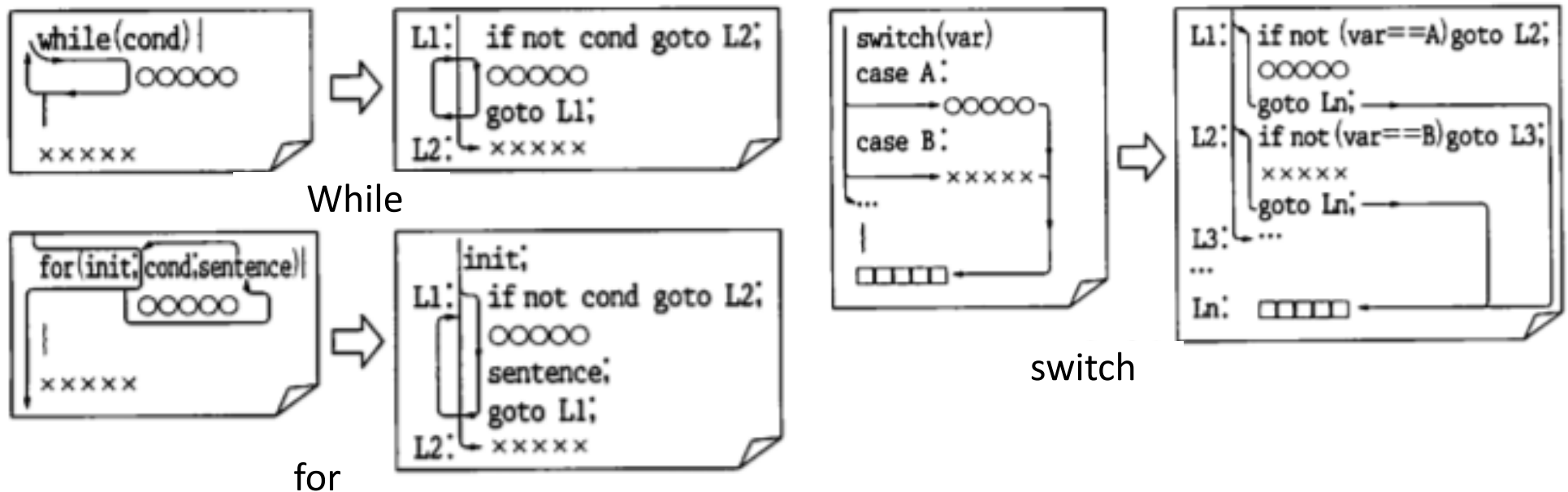
# Convert to Fixed Point Precision

[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 }
```

# C Behavior to RTL

- RTL  $\rightarrow$  Data path + FSM
- Re-write control *while*, *for*, *switch* statements to *if-then*, *goto* statements, then convert FSM
- Assign label to each statement  $\rightarrow$  FSM state number



# Example

```
L1: acc = 0;
L2: for (i = N - 1; i >= 0; i--) {
L3: if (i == 0) {
L3_1:  acc += x * c[0];
L3_2:  shift_reg[0] = x;
      } else {
L3_3:  shift_reg[i] = shift_reg[i - 1];
L3_4:  acc += shift_reg[i] * c[i];
      }
}
L4: *y = acc;
```



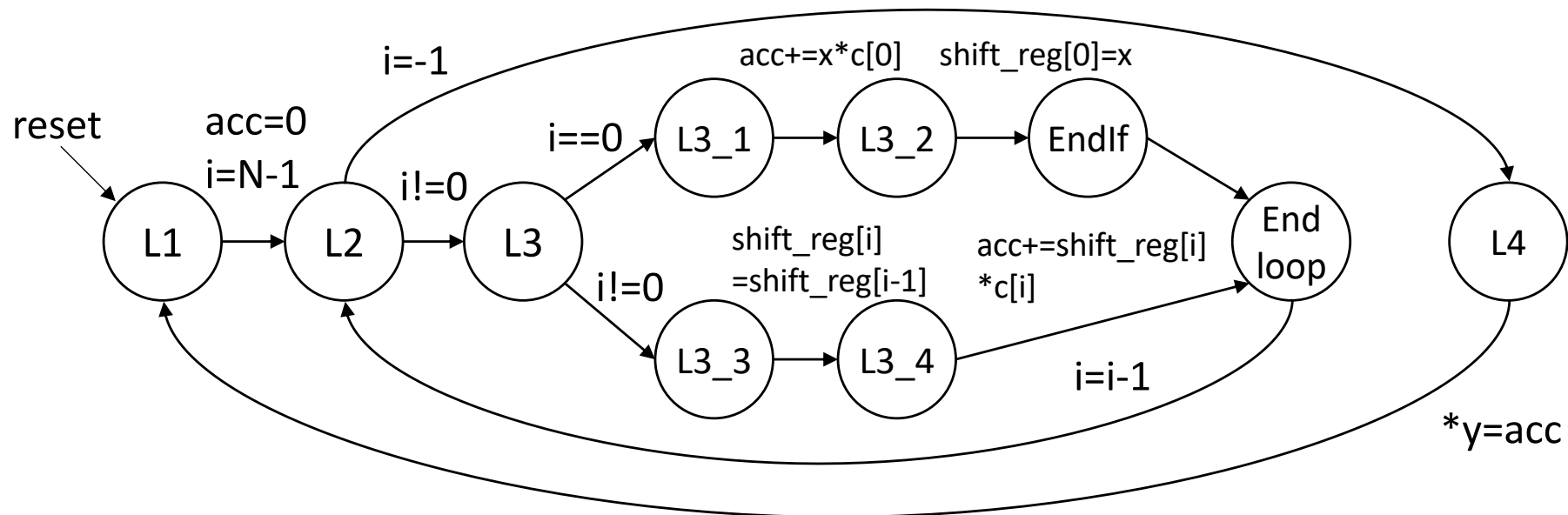
```
L1: acc = 0, i = N - 1;
L2: if i == -1 then goto L4:
L3:   if i != 0 then goto L3_3:
L3_1:   acc += x * c[0];
L3_2:   shift_reg[0] = x;
EndIf: goto Endloop:
L3_3:   shift_reg[i] = shift_reg[i - 1];
L3_4:   acc += shift_reg[i] * c[i];
Endloop: i = i - 1, goto L2:
L4: *y = acc;
```

# Write FSM

- Convert *if-then goto* statement to FSM
  - Writing an FSM until you get used to it!
- Add an initialization processing (register value after resetting)
- Make the whole process an infinite loop
  - Generally, return to the initial state after finished all processing

# Example

```
L1: acc = 0, i = N - 1;  
L2: if i == -1 then goto L4:  
L3:   if i != 0 then goto L3_3:  
L3_1:   acc += x * c[0];  
L3_2:   shift_reg[0] = x;  
EndIf:  goto Endloop:  
L3_3:   shift_reg[i] = shift_reg[i - 1];  
L3_4:   acc += shift_reg[i] * c[i];  
Endloop: i = i - 1, goto L2:  
L4: *y = acc;
```



# Parallel Processing

- Concurrent assignment

tmp=A;      A<=B;

A = B;      B<=A;

B = tmp;

- Continuous assignments

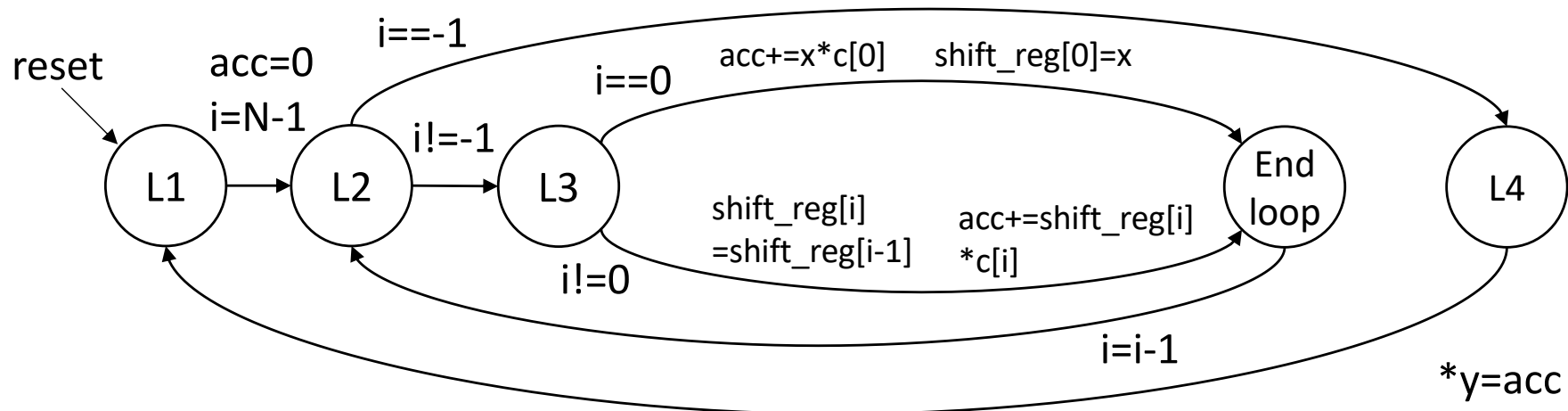
A=B; B=C;  $\rightarrow$  A=C;

- Reduce number of states by parallel processing
- Considering simultaneous assignment from the starting FSM description



# More Simplify

```
L1: acc = 0, i = N - 1;
L2: if i == -1 then goto L4:
L3:   if i == 0 then
      acc += x * c[0];
      shift_reg[0] = x;
      goto Endloop
    else
      shift_reg[i] = shift_reg[i - 1];
      acc += shift_reg[i] * c[i];
Endloop: i = i - 1, goto L2:
L4: *y = acc;
```



# RTL Simulation for an FIR Filter

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

- FIR Filter module: fir\_1.v
- Testbench for FIR Filter: testbench\_fir\_1.v

**1 Add internal signals to wave viewer**

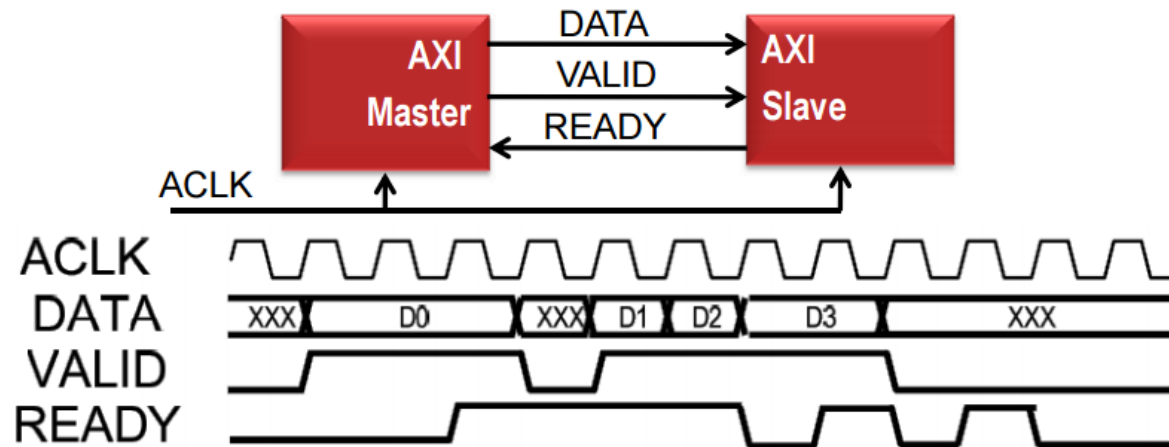
**2 Then, reset simulation "restart", and simulation again "run 1000ns"**

The screenshot shows the 'SIMULATION - Behavioral Simulation - Functional - sim\_1 - testbench\_fir\_1' window. The 'Scope' pane on the left lists the testbench and its components. The 'Objects' pane in the center shows the internal signals of the 'fir\_1' module, including 'clk', 'reset\_p', 'act', 'idx[2:0]', 'state', 'y[15:0]', and 'ready'. The 'ready' signal is circled in red. The 'Waveform' pane on the right shows the timing diagram for these signals. The 'ready' signal is also circled in red in the waveform. The 'Tcl Console' at the bottom shows the commands 'restart' and 'run 1000ns'.

# Interface Co-Design

# Interface

- Data Transfer/Receive between modules



Streaming



Burst



Irregular interval



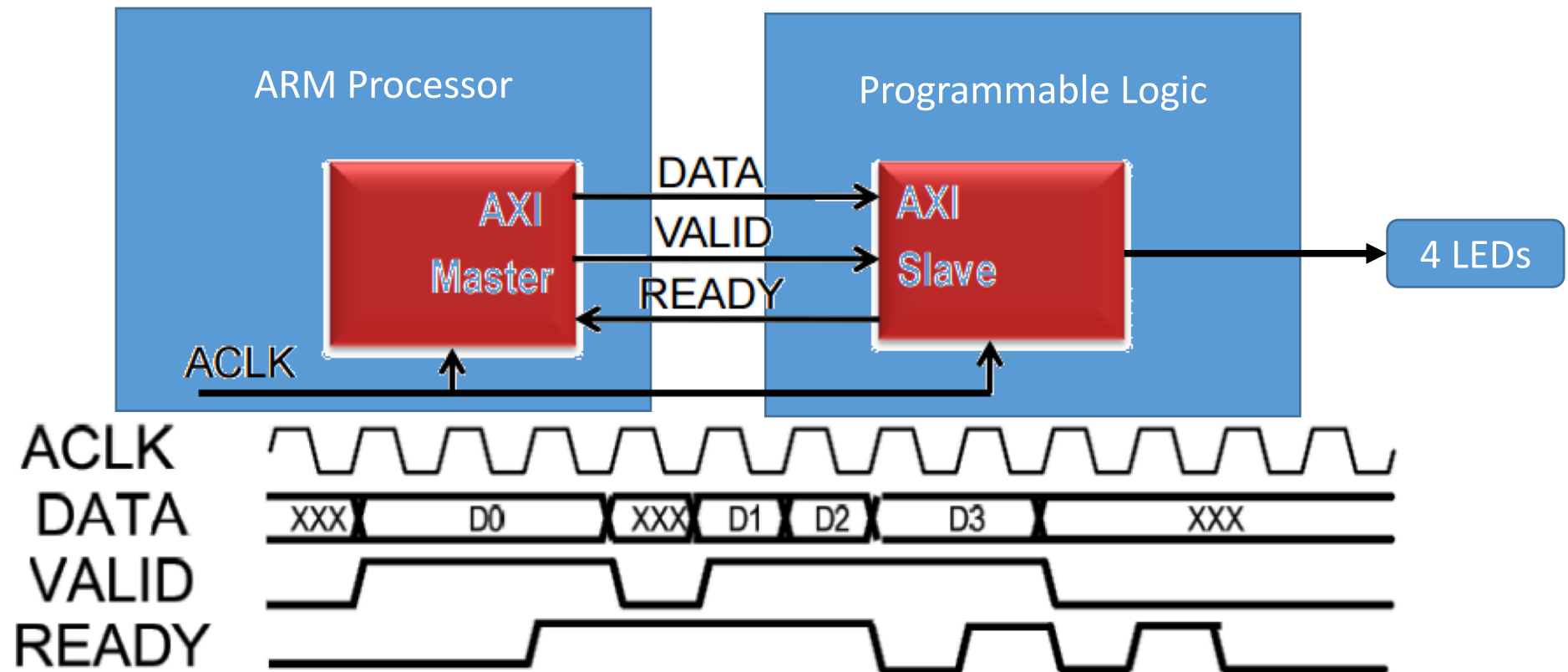
# AXI 4 bus: General Interface of ARM Embedded FPGA

- Complex protocols
  - High-level synthesis (HLS) can be easily generated with Directive
  - System design tool (SDSoC) automatically selects the best protocol

	AXI4	AXI4-Lite	AXI4-Stream
Dedicated for	high-performance and memory mapped systems	register-style interfaces (area efficient implementation)	non-address based IP (PCIe, Filters, etc.)
Burst (data beta)	up to 256	1	Unlimited
Data width	32 to 1024 bits	32 or 64 bits	any number of bytes
Applications (examples)	Embedded, memory	Small footprint control logic	DSP, video, communication

# Case Study: AXI4 Bus Connection

- Led blinking via AXI-lite bus



# Create a New Project

Project location: C:\FPGA\lect7\_2\led\_axi\_lite\_1

Target FPGA: Zybo-Z7-10 or (Z7-20)

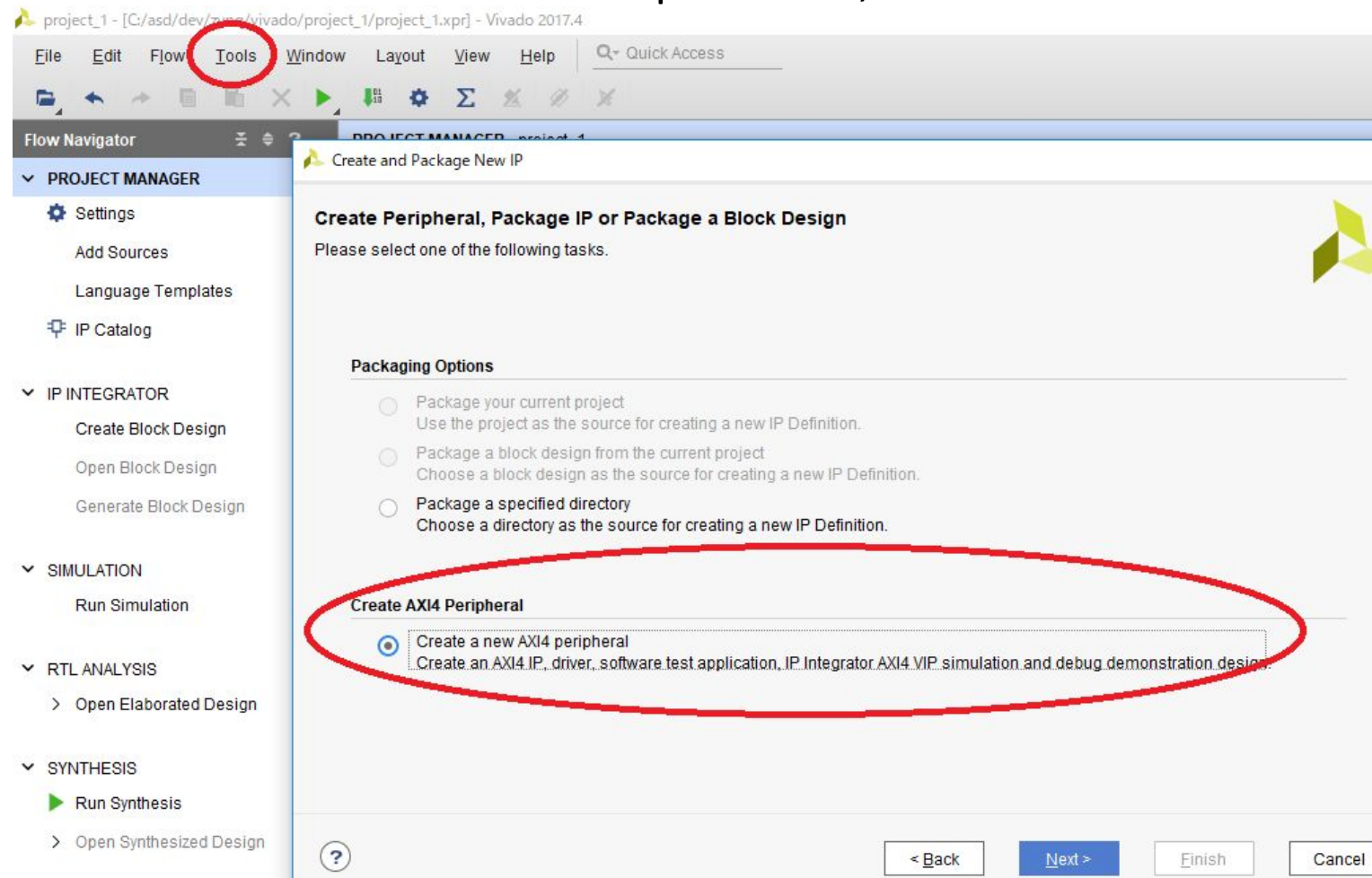
Design Sources: None

Constraints: Zybo-Z7-Master.xdc

Simulation Sources: None

# Create AXI4 Peripheral

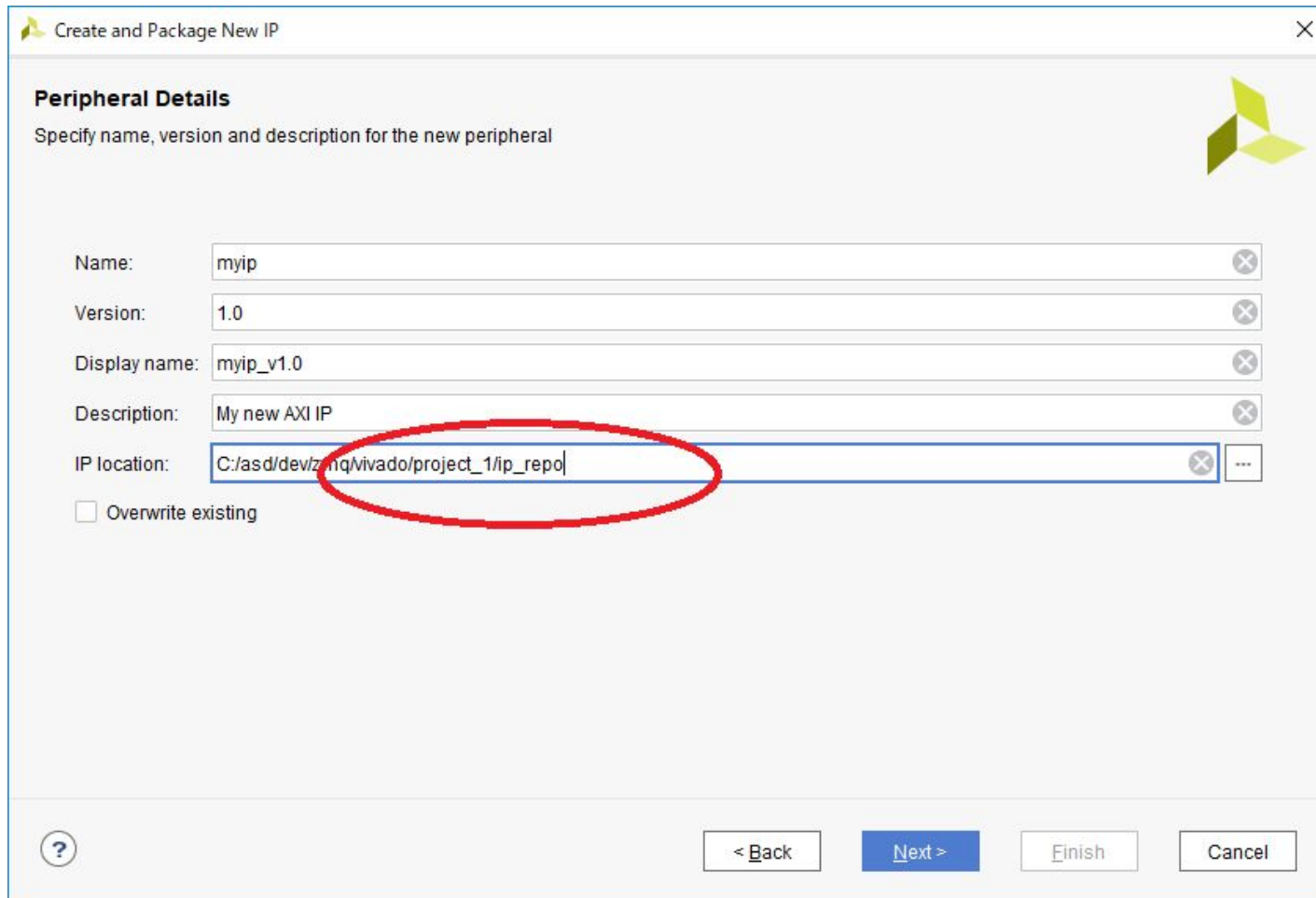
- Select "Tools->Create and Package New IP", then check "Create AXI4 Peripheral", and "Next"





# Specify IP Location

Type "ip\_repo" on your project directory, then "Next"



**Create and Package New IP**

**Peripheral Details**  
Specify name, version and description for the new peripheral

Name: myip

Version: 1.0

Display name: myip\_v1.0

Description: My new AXI IP

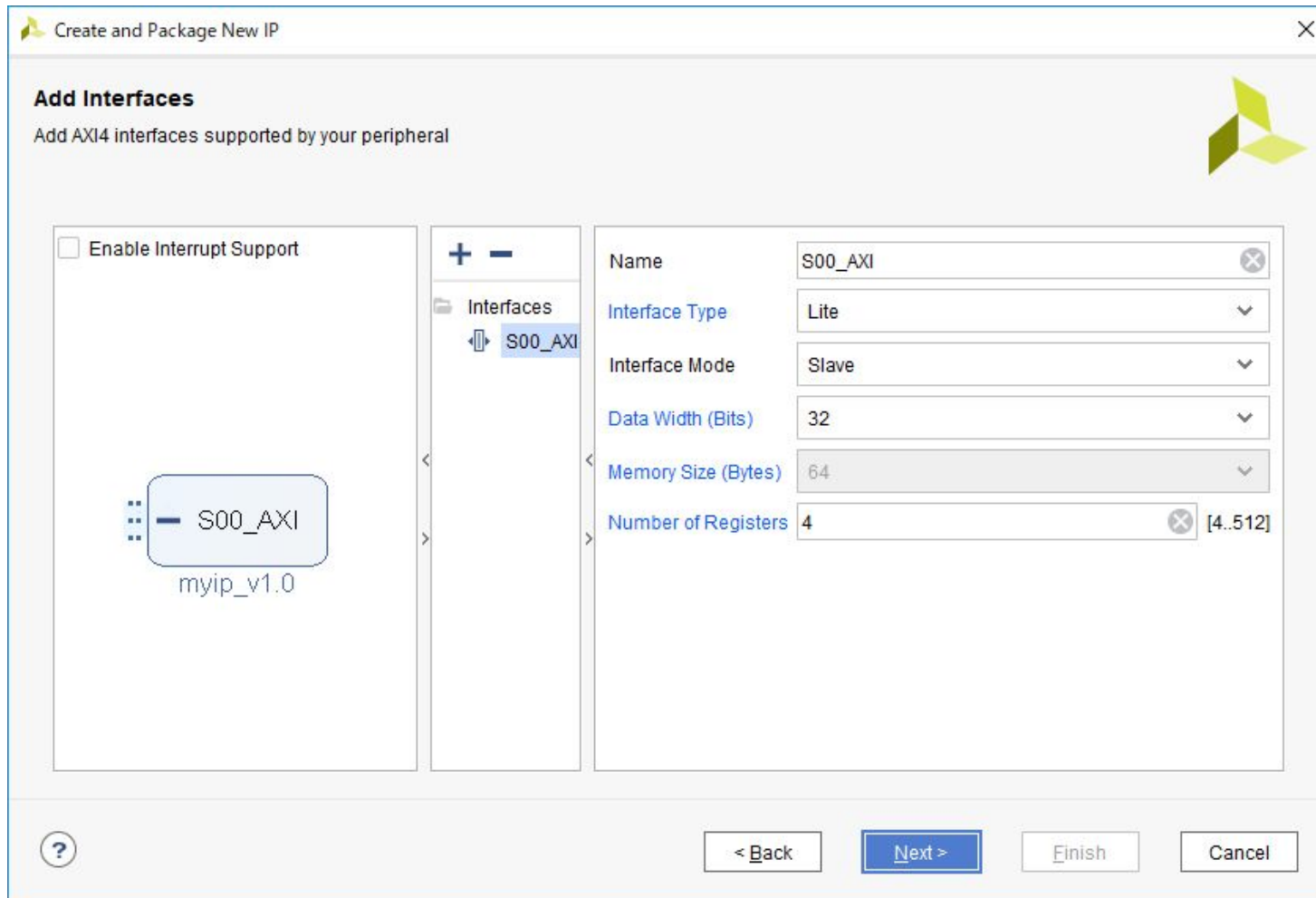
IP location: C:/asd/dev/mq/vivado/project\_1/ip\_repo

☐ Overwrite existing

< Back Next > Finish Cancel

# Edit Interface

- Set the default "AXI4-Lite Slave (four 32-bit registers)", and "OK", then "Finish"



The screenshot shows the 'Create and Package New IP' dialog box, specifically the 'Add Interfaces' tab. The dialog has a title bar with a yellow logo and a close button. The main area is divided into three sections. On the left, there is a checkbox labeled 'Enable Interrupt Support' which is unchecked. Below it is a diagram showing a block labeled 'S00\_AXI' with a small icon to its left, and the text 'myip\_v1.0' below the block. In the center, there is a list of interfaces under the heading 'Interfaces'. The list contains one item, 'S00\_AXI', which is highlighted. To the right of the list is a table of configuration options for the selected interface. The table has two columns: the first column lists the configuration options, and the second column shows the current values. The configuration options and their values are: Name (S00\_AXI), Interface Type (Lite), Interface Mode (Slave), Data Width (Bits) (32), Memory Size (Bytes) (64), and Number of Registers (4). The 'Number of Registers' field has a range indicator '[4..512]' to its right. At the bottom of the dialog, there is a row of buttons: a help button (question mark icon), a '< Back' button, a 'Next >' button (highlighted in blue), a 'Finish' button, and a 'Cancel' button.

Create and Package New IP

**Add Interfaces**  
Add AXI4 interfaces supported by your peripheral

☐ Enable Interrupt Support

Interfaces

- S00\_AXI

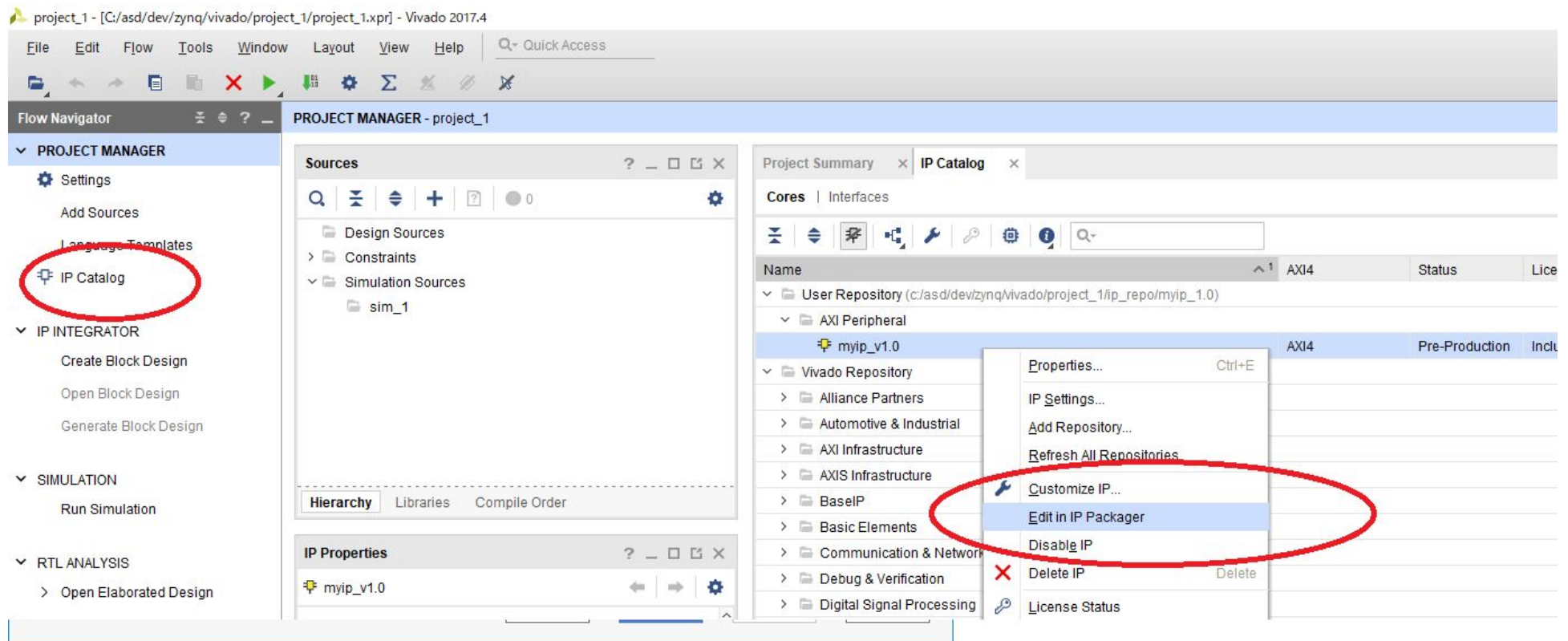
S00\_AXI  
myip\_v1.0

Name	S00_AXI
Interface Type	Lite
Interface Mode	Slave
Data Width (Bits)	32
Memory Size (Bytes)	64
Number of Registers	4 [4..512]

? < Back Next > Finish Cancel

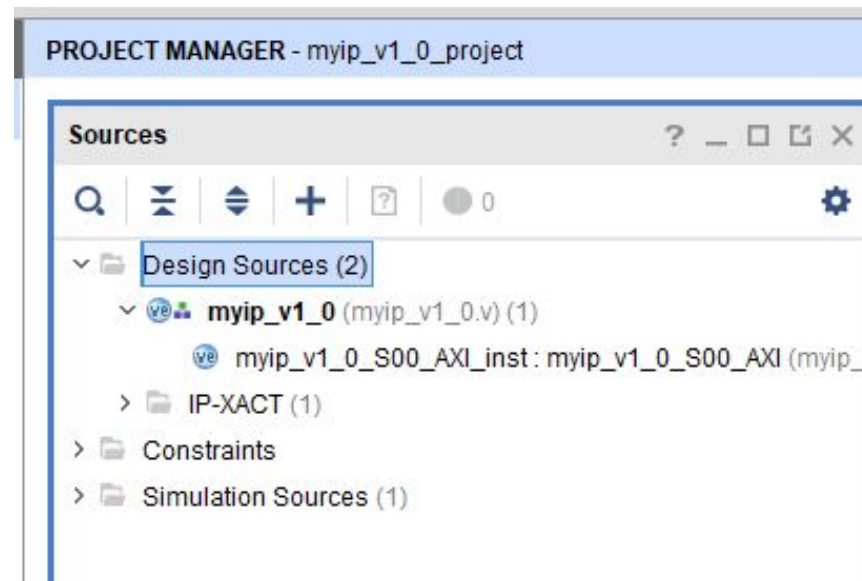
# Edit "myip\_v1.0"

- Click Flow Navigator->PROJECT MANAGER -> IP Catalog
- Make sure "myip\_v1.0" under "User Repository" on "IP Catalog"
- Right click on "myip\_v1.0", then select "Edit in IP Packager"
  - Click "OK" to save the project location

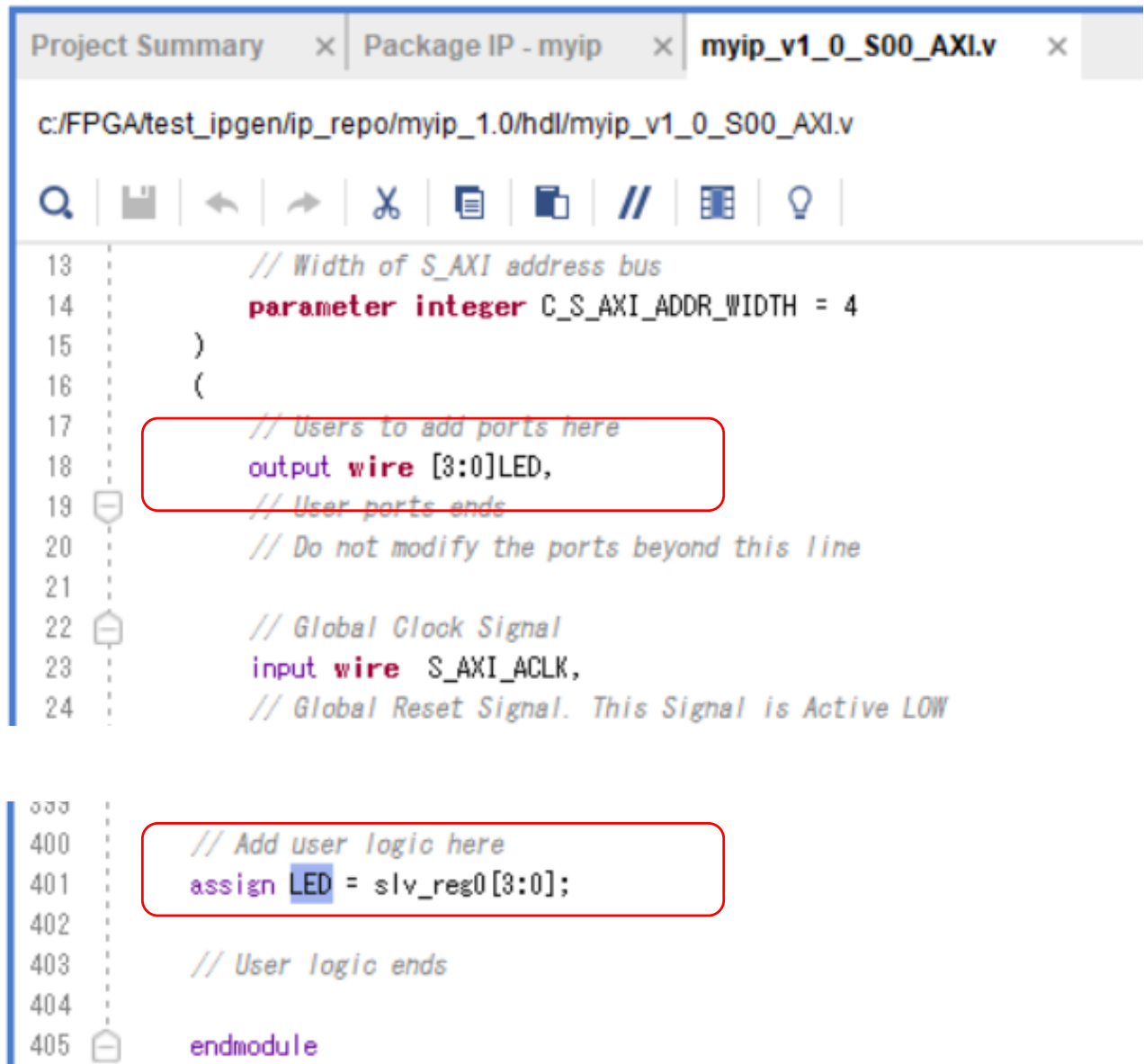


# Synthesis "myip" on a New Vivado

- Make sure "myip\_v1\_0.v" as a wrapper and "myip\_v1\_0\_S00\_AXI.v" as a top module



# Edit "myip\_v1\_0\_S00\_AXI.v"



```
Project Summary x Package IP - myip x myip_v1_0_S00_AXI.v x
c:/FPGA/test_ipgen/ip_repo/myip_1.0/hdl/myip_v1_0_S00_AXI.v

13 // Width of S_AXI address bus
14 parameter integer C_S_AXI_ADDR_WIDTH = 4
15 )
16 (
17 // Users to add ports here
18 output wire [3:0] LED,
19 // User ports ends
20 // Do not modify the ports beyond this line
21
22 // Global Clock Signal
23 input wire S_AXI_ACLK,
24 // Global Reset Signal. This Signal is Active LOW

399
400 // Add user logic here
401 assign LED = slv_reg0[3:0];
402
403 // User logic ends
404
405 endmodule
```

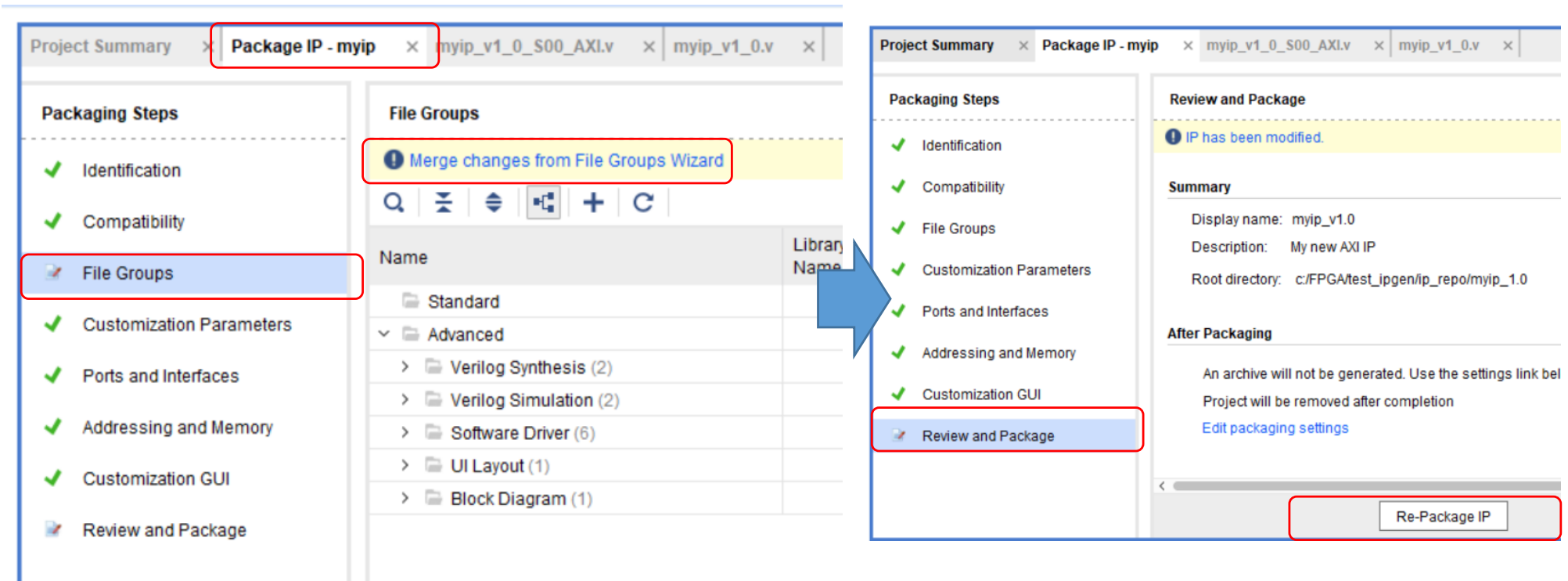
# Edit "myip\_v1\_0.v"

```
12 // Parameters of Axi Slave Bus Interface S00_AXI
13 parameter integer C_S00_AXI_DATA_WIDTH  = 32,
14 parameter integer C_S00_AXI_ADDR_WIDTH  = 4
15 )
16 (
17 // Users to add ports here
18 output wire [3:0]LED,
19 // User ports ends
20 // Do not modify the ports beyond this line
21
```

```
44 input wire s00_axi_rready
45 );
46 // Instantiation of Axi Bus Interface S00_AXI
47 myip_v1_0_S00_AXI # (
48     .C_S_AXI_DATA_WIDTH(C_S00_AXI_DATA_WIDTH),
49     .C_S_AXI_ADDR_WIDTH(C_S00_AXI_ADDR_WIDTH)
50 ) myip_v1_0_S00_AXI_inst (
51     .LED( LED),
52     .S_AXI_ACLK(s00_axi_aclk),
53     .S_AXI_ARESETN(s00_axi_aresetn),
54     .S_AXI_ADD00P(s00_axi_add00p)
```

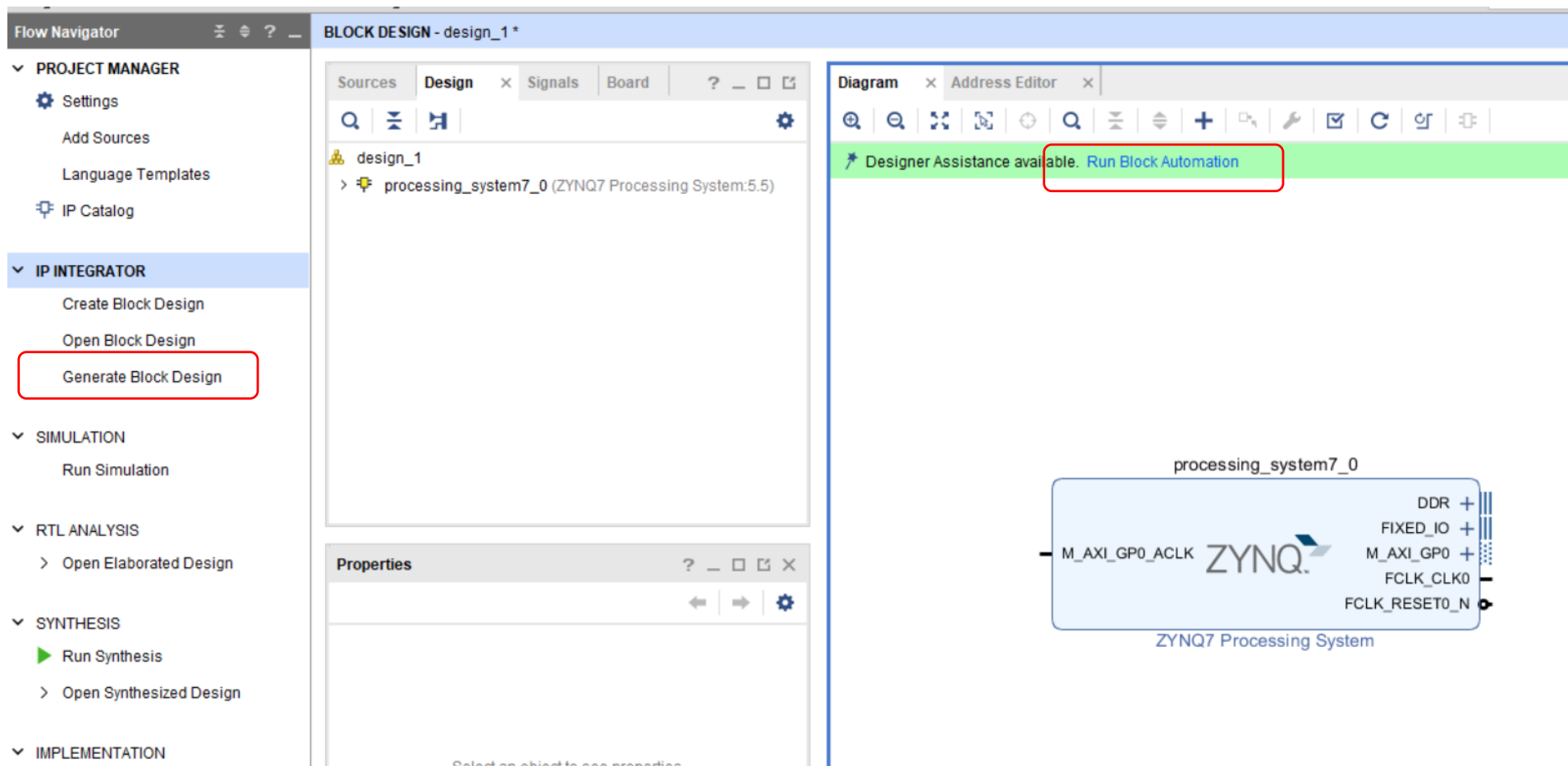
# Re-Package IP

- Switch to "Package IP" tab, then "Re-Package IP"



# Add a ZYNQ Processor

- In the initial Vivado, Flow Navigator -> IP INTEGRATOR -> Create Block Design, then add a ZYNQ Processor, and "Run Block Automation"

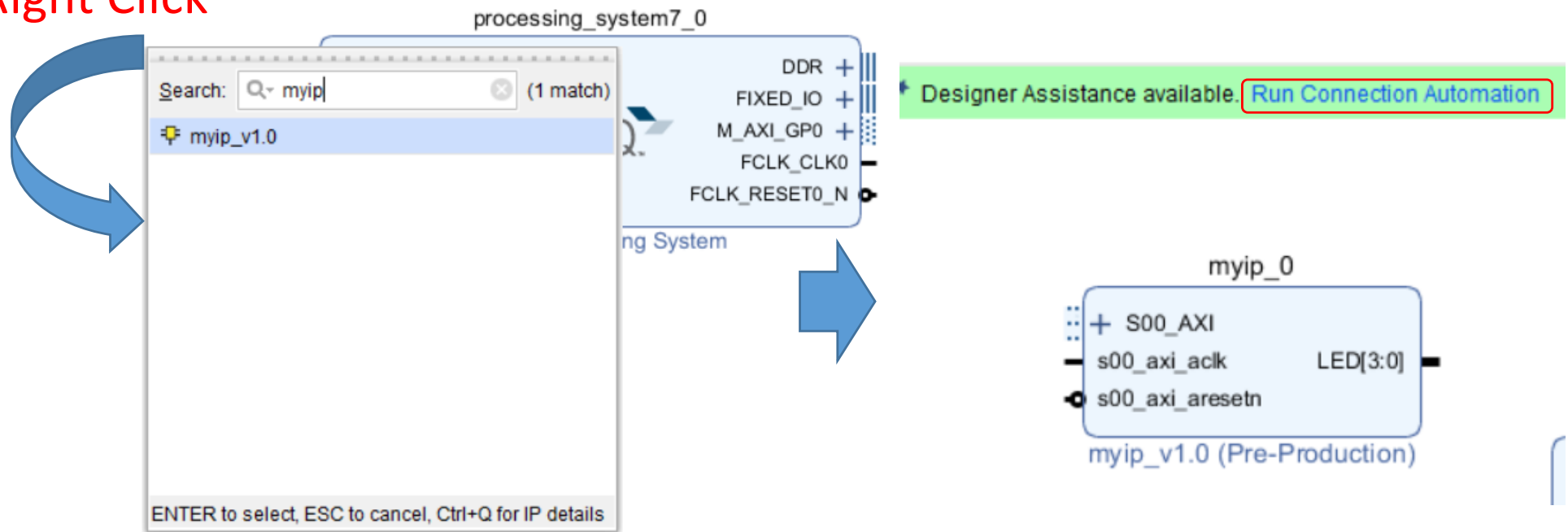




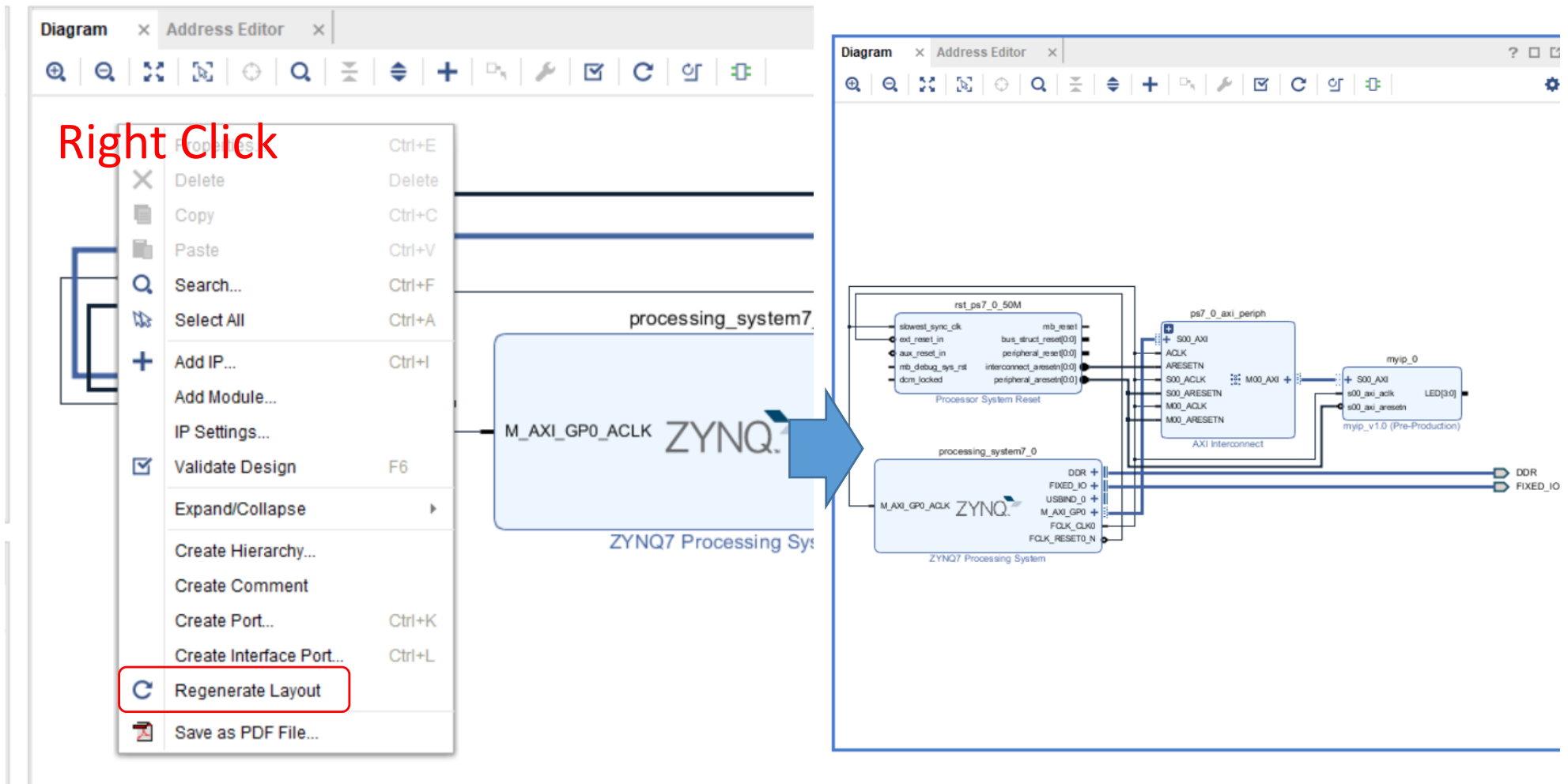
# Add a "myip" IP

- Place your "myip" on the Block Design View, then click "Run Connection Automation", and "OK"

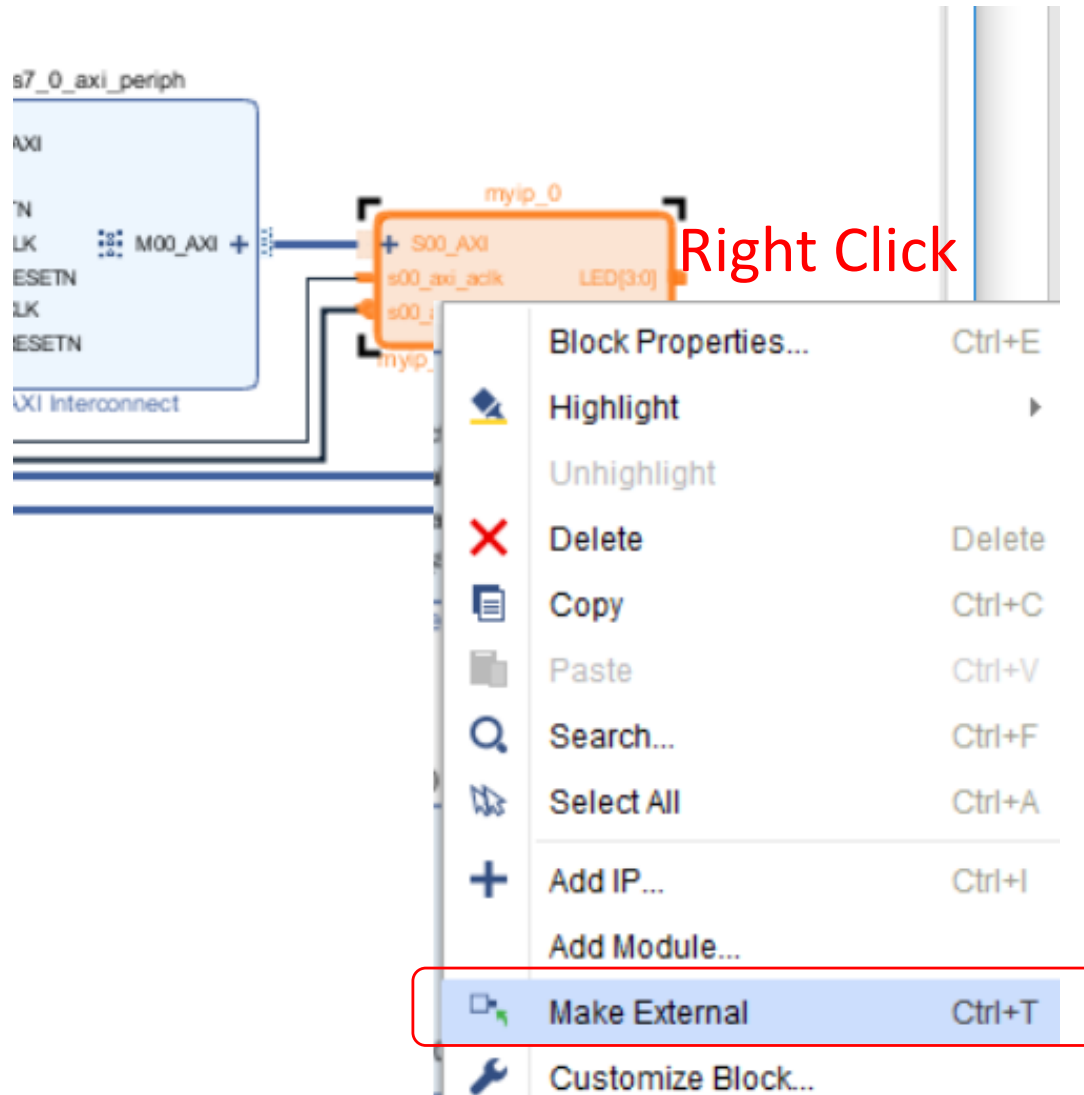
Right Click



# Regenerate Layout



# Make External



# Specify an External Port Name

The screenshot displays the Xilinx Block Design tool interface for a design named "design\_1". The left sidebar shows the project hierarchy, including "Design Sources (1)", "Constraints (1)", and "Simulation Sources (1)". The main workspace shows a block diagram of the "ZYNQ7 Processing System" with various components like "rst\_ps7\_0\_50M", "ps7\_0\_axi\_periph", and "processing\_system7\_0".

Two red annotations highlight the steps to specify an external port name:

- A red arrow points to the "Name:" field in the "External Port Properties" dialog, which is set to "led". A red text label "set 'led'" is placed next to this field.
- A red arrow points to the "External Port Properties..." menu item in the right-click context menu. A red text label "Right Click" is placed next to this menu item.

The "External Port Properties" dialog shows the following details:

- Name: led
- Direction: Output
- From: 3 To: 0
- Net: myip\_0\_LED

The right-click context menu includes options such as Highlight, Unhighlight, Delete, Copy, Paste, Search..., Select All, Add IP..., and Add Module...

# Write Software Code to Control "myip" from a ZYNQ Processor

- Click "Generate Bitstream", then "Export Hardware", and next, "Launch SDK"
- Create a new project as "myip\_test"

# Source Code

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
```

```
#include "xparameters.h"
```

```
#define LED *((volatile unsigned int *) XPAR_MYIP_0_S00_AXI_BASEADDR)
```

```
int main()
{
    init_platform();

    print("Hello World\n\r");

    int i, j;

    while(1){
        for( i = 0; i < 6; i++){
            xil_printf("i=%d\n", i);
            switch(i){
                case 0: LED = 0x1; break;
                case 1: LED = 0x2; break;
                case 2: LED = 0x3; break;
                case 3: LED = 0x4; break;
                case 4: LED = 0x5; break;
                case 5: LED = 0x6; break;
                default: LED = 0x0;
            }
            for( j = 0; j < 10000000; j++);
        }
    }

    cleanup_platform();
    return 0;
}
```

Memory map is automatically generated by Vivado, and it is written in "xparameters.h"

Build the project, then "Xilinx->Program FPGA".

Next, Connect the Zybo to the PC

Run Terminal software (e.g. Tera Term for Windows, gtkterm for Unix)

Connect "USB Serial Port" with 115200 bps

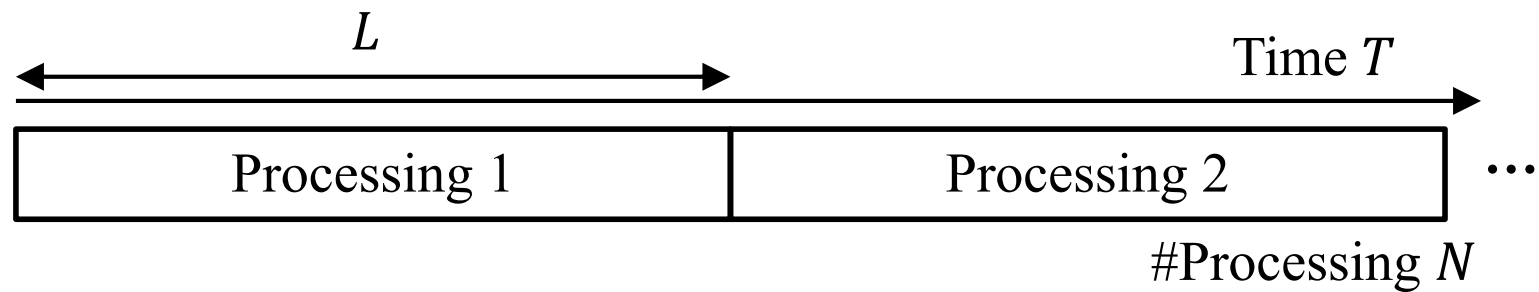
Select the project in the Project Explorer, then, in "Menu", "Run As" -> "Launch on Hardware (System Debugger)"

# RTL Design Optimization

# Pipelining

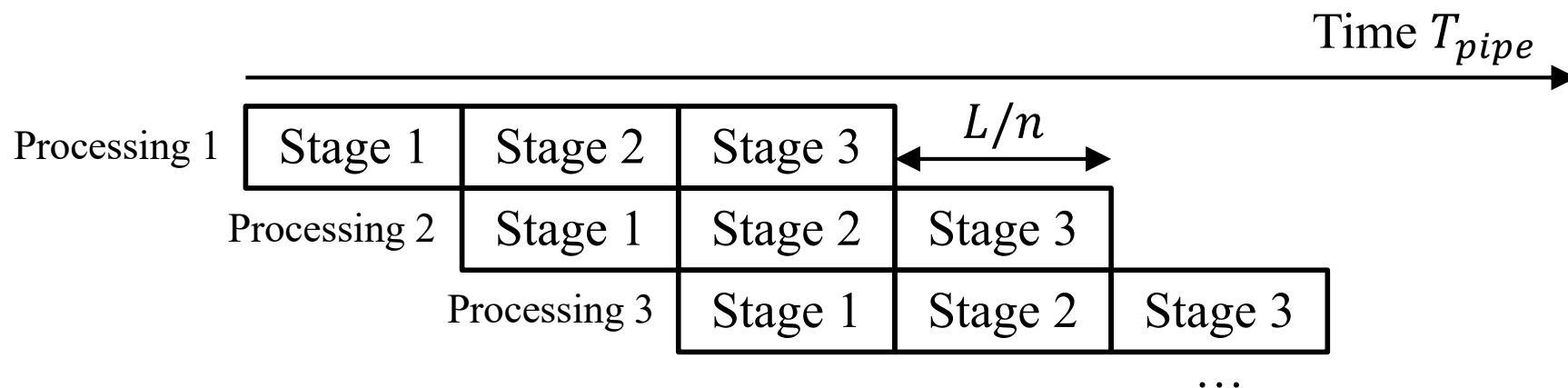
## (a) Non-pipelining

Processing iteration 2 is done sequentially **after the completion of iteration 1**



## (b) Pipelining ( $n = 3$ stage )

Processing iteration 2 is done **after the completion of stage 1 in iteration 1**





# Pipeline Efficiency

Percentage of the actually achieved speedup to the maximum

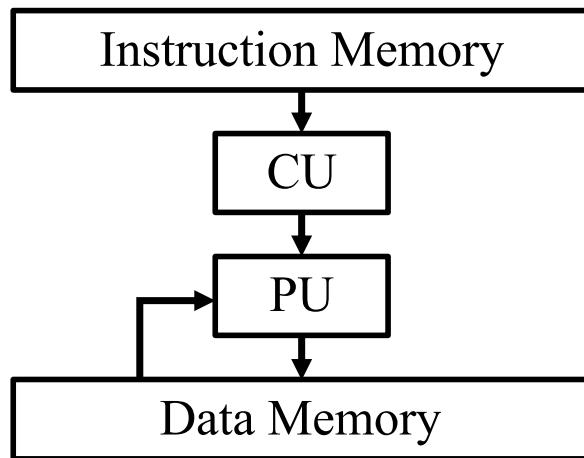
$$S_{pipe}(N) = \frac{T(N)}{T_{pipe}(N)} = \frac{nN}{n + N - 1} = \frac{n}{1 + \frac{n-1}{N}}$$

If  $n \ll N$ , then  $S_{pipe}(N) \cong n$  and the speed-up factor over non-pipelining is  $n$

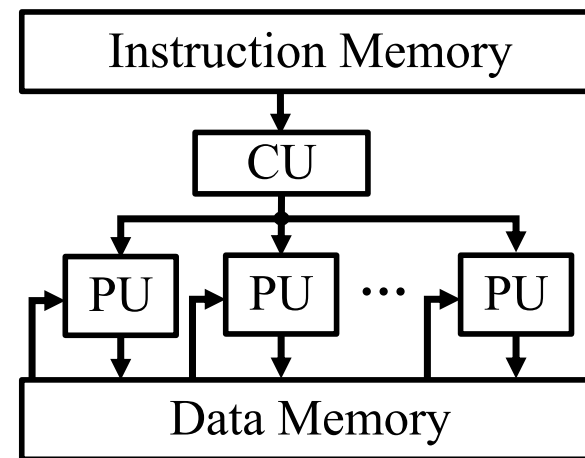
Percentage of the actually achieved speedup to the maximum

$$E_{pipe}(n, N) = \frac{S_{pipe}(N)}{n} = \frac{1}{1 + \frac{n-1}{N}} = \frac{N}{N + n - 1}$$

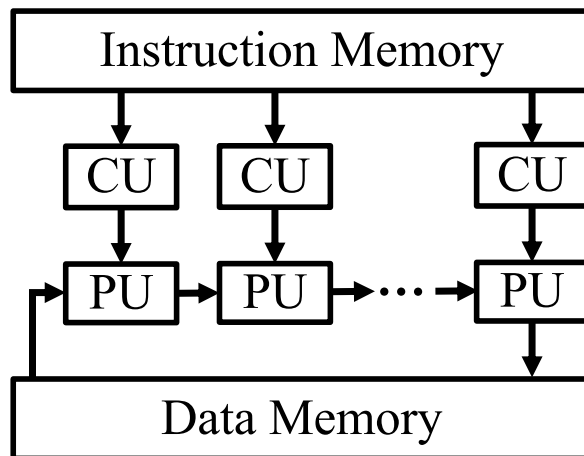
# Parallel Processing and Flynn's Taxonomy



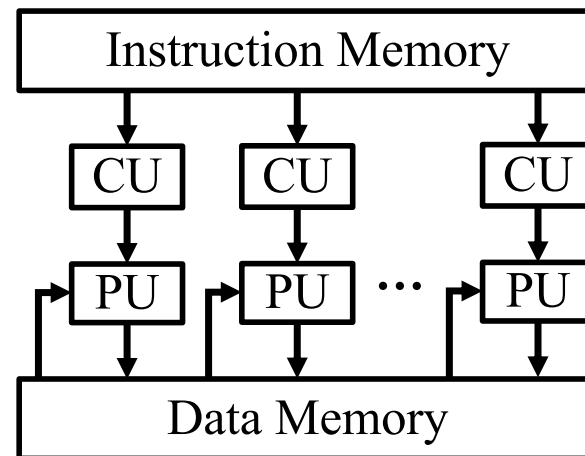
(a) SISD



(b) SIMD



(c) MISD



(d) MIMD

# Loop Unrolling

- Without unrolling

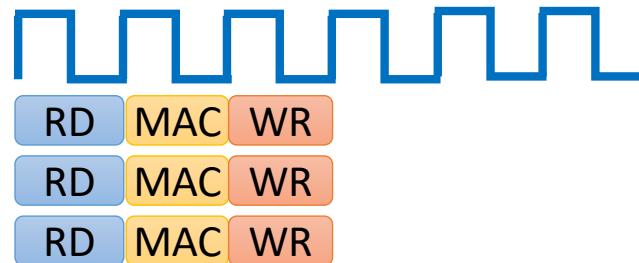
```
for ( int i = 0; i < N; i++){  
  op_Read[i];  
  op_MAC;  
  op_Write[i];  
}
```



Throughput: 3 cycles  
Latency: 3 cycles  
Operation: 1/3 data/cycle

- Loop Unrolling for 3 Operations

```
for ( int i = 0; i < N/3; i+=3){  
  op_Read[i*3];  
  op_MAC;  
  op_Write[i*3];  
  op_Read[i*3+1];  
  op_MAC;  
  op_Write[i*3+1];  
  op_Read[i*3+2];  
  op_MAC;  
  op_Write[i*3+2];  
}
```



Throughput: 3 cycle  
Latency: 3 cycle  
Operation: 1 data/cycle

# Unrolling for a FIR Filter

```
int c[N] = { // 0.17 = 20KHz/44.1KHz, LPF, Hami
    -136, -397, -87, 3004, 8338, 11142, 8338,
    3004, -87, -397, -136, };

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

acc = 0;
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;
```

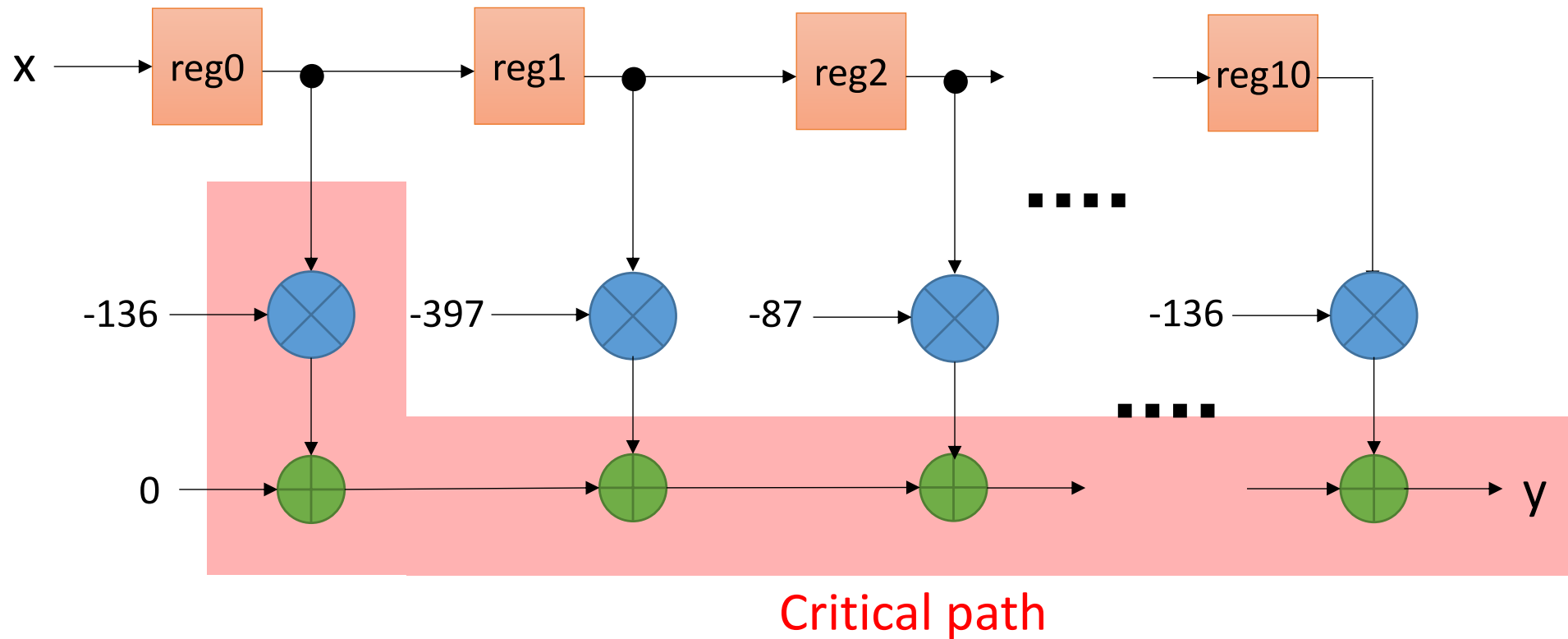


```
static int shift_reg[N];

shift_reg[10] = shift_reg[9];
shift_reg[ 9] = shift_reg[8];
shift_reg[ 8] = shift_reg[7];
shift_reg[ 7] = shift_reg[6];
shift_reg[ 6] = shift_reg[5];
shift_reg[ 5] = shift_reg[4];
shift_reg[ 4] = shift_reg[3];
shift_reg[ 3] = shift_reg[2];
shift_reg[ 2] = shift_reg[1];
shift_reg[ 1] = shift_reg[0];
shift_reg[ 0] = x;

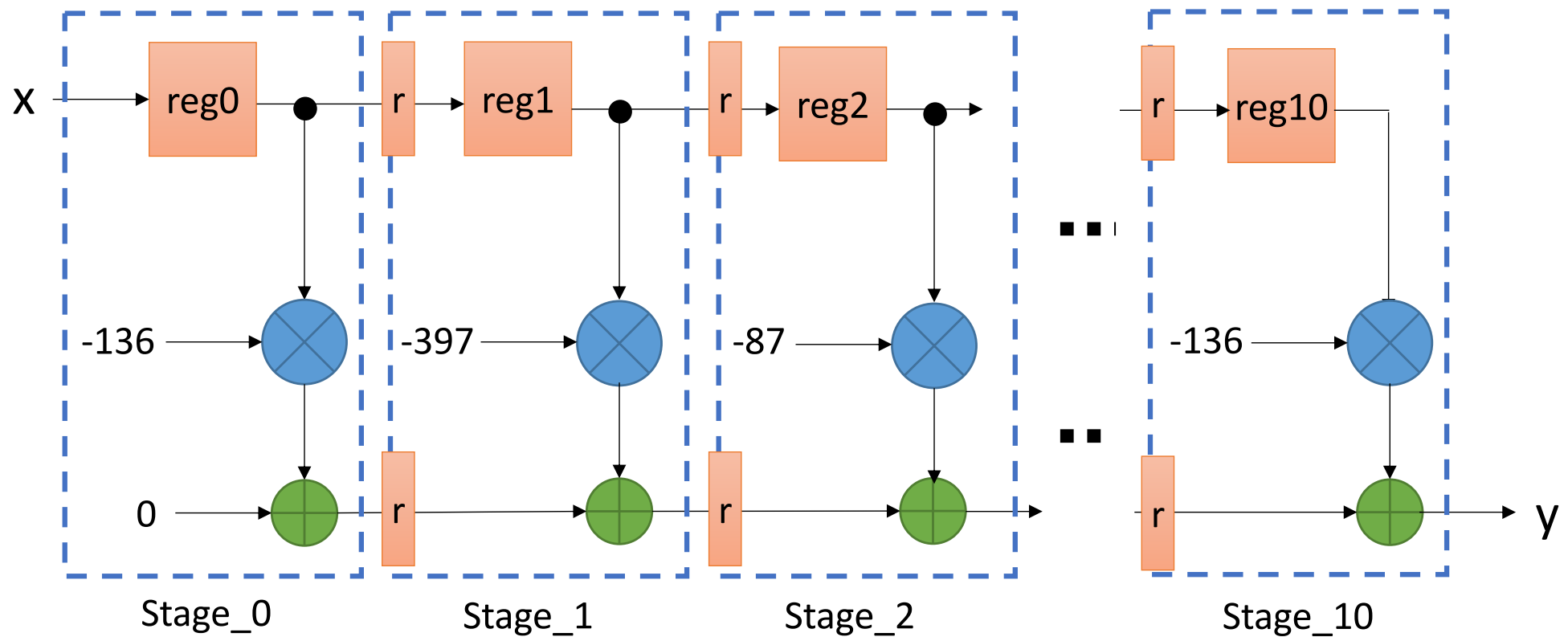
*y = shift_reg[10] * -136 + shift_reg[9] * -397
    + shift_reg[ 8] * -87 + shift_reg[7] * 3004
    + shift_reg[ 6] * 8338 + shift_reg[5] * 11142
    + shift_reg[ 4] * 8338 + shift_reg[3] * 3004
    + shift_reg[ 2] * -87 + shift_reg[1] * -397
    + shift_reg[ 0] * -136;
```

# Dataflow for Unrolling FIR Filter



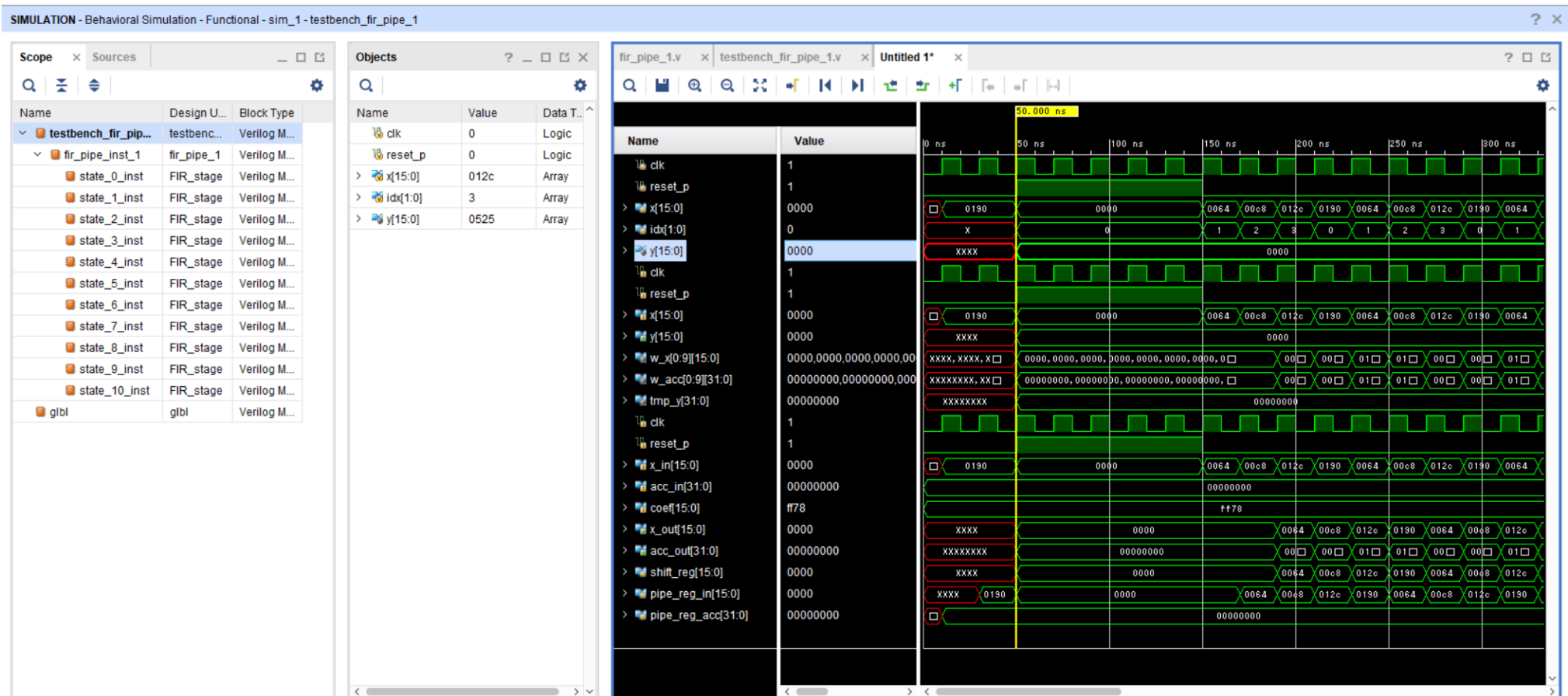
# Pipelined Dataflow

- Insert a pipeline register and realized by a DSP block



# RTL Simulation

- See, [https://github.com/HirokiNakahara/FPGA\\_lecture/tree/master/Lec7\\_Practical\\_RTL\\_design/](https://github.com/HirokiNakahara/FPGA_lecture/tree/master/Lec7_Practical_RTL_design/)
- Source Code: fir\_pipe\_1.v, Simulation Code: testbench\_fir\_pipe\_1.v



# Conclusion

- Conversion from Behavior to RTL by C Description
- Control HW via AXI 4 bus
- Optimization method
  - Concurrent assignment
  - Parallel Processing
  - Unrolling
  - Pipelining



# Exercise

- (Mandatory) Control the LED from the ARM processor via the AXI4 bus
- (Mandatory) For the FIR filter, discuss the Pros. and Cons. of pipeline version, unrolling version, sequential version by comparing latency, throughput, and multiplier
- (Optional 1) Reduce the number of multipliers by using a symmetry property for coefficients of an FIR filter
- (Optional 2) Design the RTL for above FIR filter and show the simulation result

Send a report to OCW-I by PDF format

Deadline is 9<sup>th</sup>, July, 2019