High-Level Synthesis (HLS) Design: Introduction

Hiroki Nakahara

Tokyo Institute of Technology

Outline

- HLS Introduction
- Walk Through HLS Design

HLS Introduction

FPGA Potential

- Implementing computations in hardware can have speed/energy advantages over software:
  - Lithography simulation: **15x** speed-up [Cong & Zhou, TRETS'09]
  - Linear system solver: **2.2x** speed-up, **5x** more energy efficient [Zhang, Betz, Rose, TRETS'12]
  - Monte Carlo simulation for photodynamic therapy: **80x** faster, **45x** more energy efficient [Lo et al., J. Biomed Optics'09]
  - Option pricing: **4.6x** faster, **25x** more energy efficient [Tse, Thomas, Luk, TVLSI'12]

Two Ways Computations

Write Software
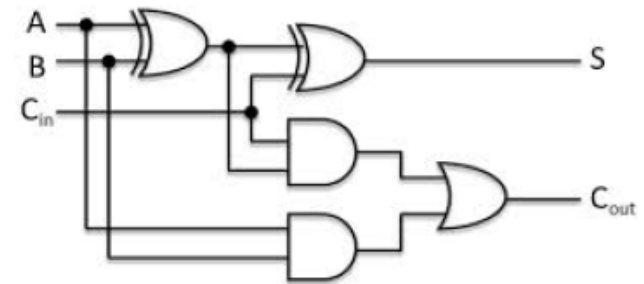
Design Custom Circuits

Hardware Design is Difficult

- FPGA "success stories" are pervasive, yet the technology remains inaccessible to software engineers
  - Requires use of hardware description languages: Verilog and VHDL
- Hardware design skills are rare:
  - 10 software engineers for every hardware engineer*

$$\downarrow$$

- Make hardware design easier for hardware engineers
- Allow software engineers to design hardware and reap its energy/performance benefits

*Source: US Bureau of Labor Statistics, 2012

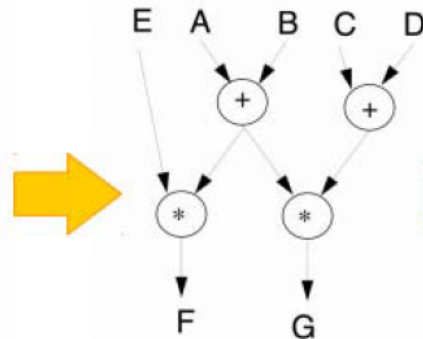# High-level synthesis (HLS) with FPGAs

- Plays a central role, enabling the automatic synthesis of high-level, untimed or partially timed specifications (e.g. C or System C) to low-level cycle accurate RTL specifications

- Target devices includes application-specific integrated circuits (ASIC) or field-programmable gate arrays (FPGAs)

- It can be optimized taking into account the performance, power, and cost requirements
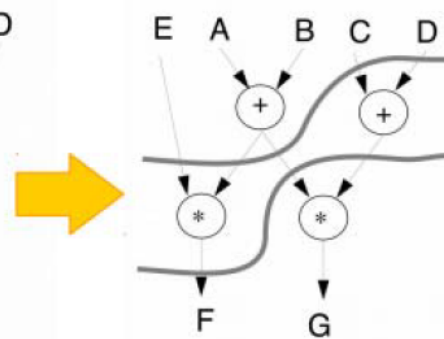
High-level Synthesis Flow

Input Behavioral Spec.

Dataflow

Scheduling

Data-path generation

Mapping to resources (Binding)

Controller (FSM) Generation

Scheduling

- How to assign the computations of a program into the hardware time steps (clock cycles)?

- Must consider under given target clock period:
  - Which operations can be scheduled in the same time step?
  - Which operations are dependent on others?

Schedule strategies

- As soon as possible (ASAP)
- As late as possible (ALAP)

(a) Sorted DFG

(b) ASAP schedule
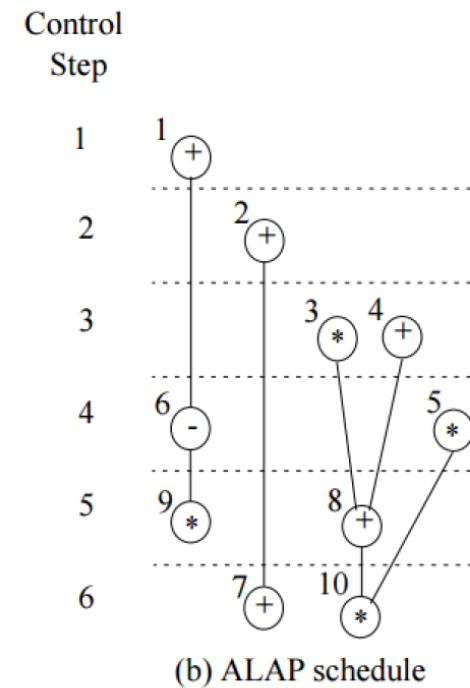
(b) ALAP schedule

High-level Synthesis Flow


```
PROCEDURE Test;
  VAR

A,B,C,D,E,F,G:integer;
  BEGIN
  Read(A,B,C,D,E);
  F := E*(A+B);
  G := (A+B)*(C+D);
  ...
  END;
```
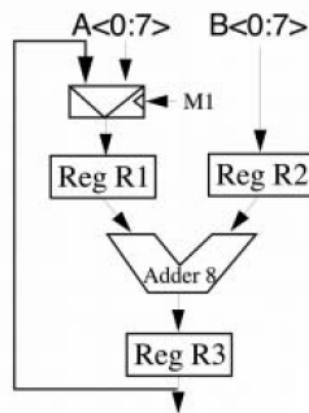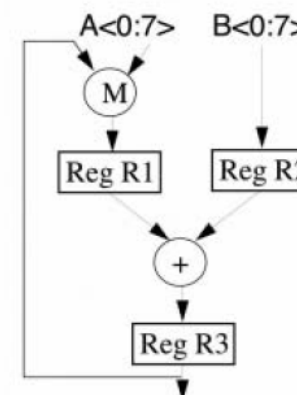
Input Behavioral Spec.

Dataflow

Scheduling

Data-path generation
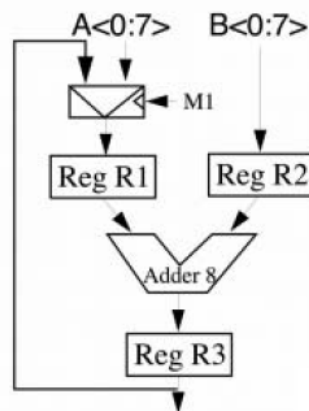
Controller ROM:
```
0000 : 11000000 0001
0001 : 00100000 0010
0010 : 00011000 0011
0011 : 01000000 0100
```
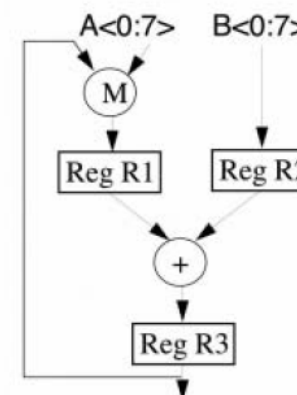
Controller description:
```
S1: M1=1, Load R1 next S2;
S2: Load R2 next S3;
S3: Add, Load R3 next S4;
S4: M1=0, Load R1 next...
```
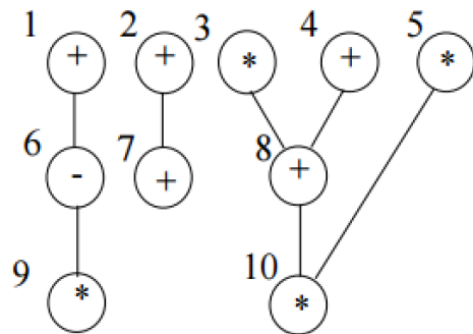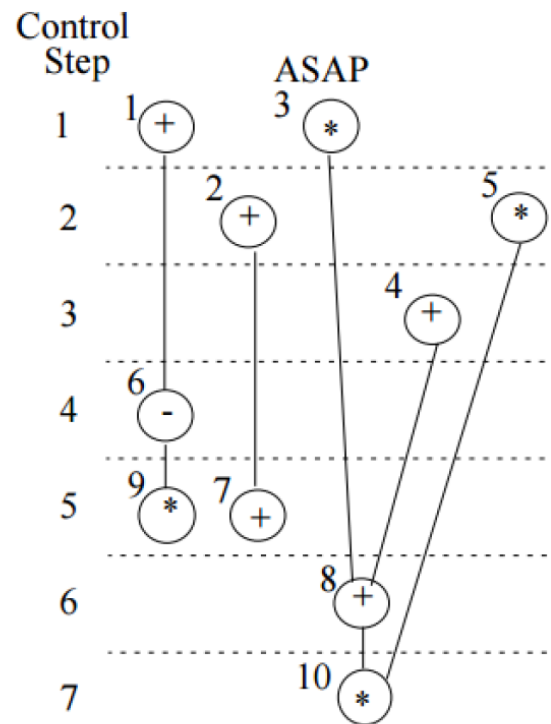
Mapping to resources
(Binding)

Controller (FSM) Generation

Binding

- Assign (bind) each operation to a hardware functional unit

- Example:
  - LegUp uses bipartite matching-based binding [Huang DAC'90] using the Hungarian Method (Polynomial order)

operations

edge costs

hardware functional units
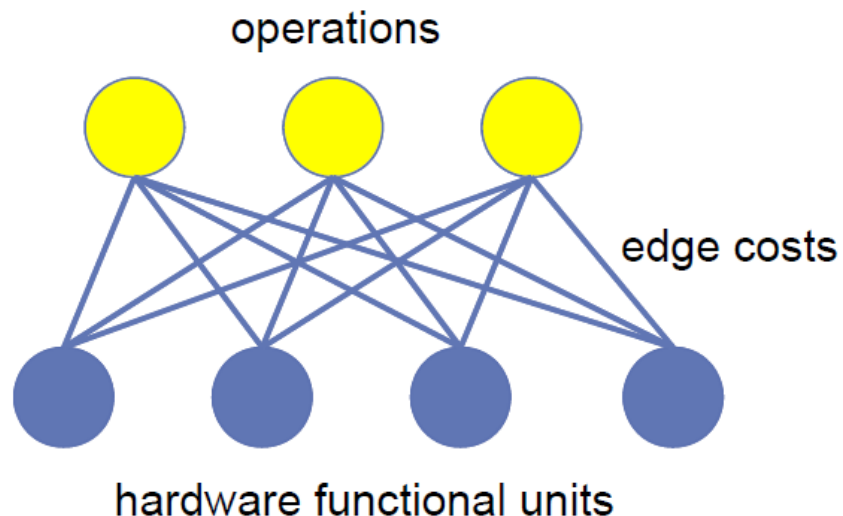
AutoESL and Xilinx C-to-FPGA design flow
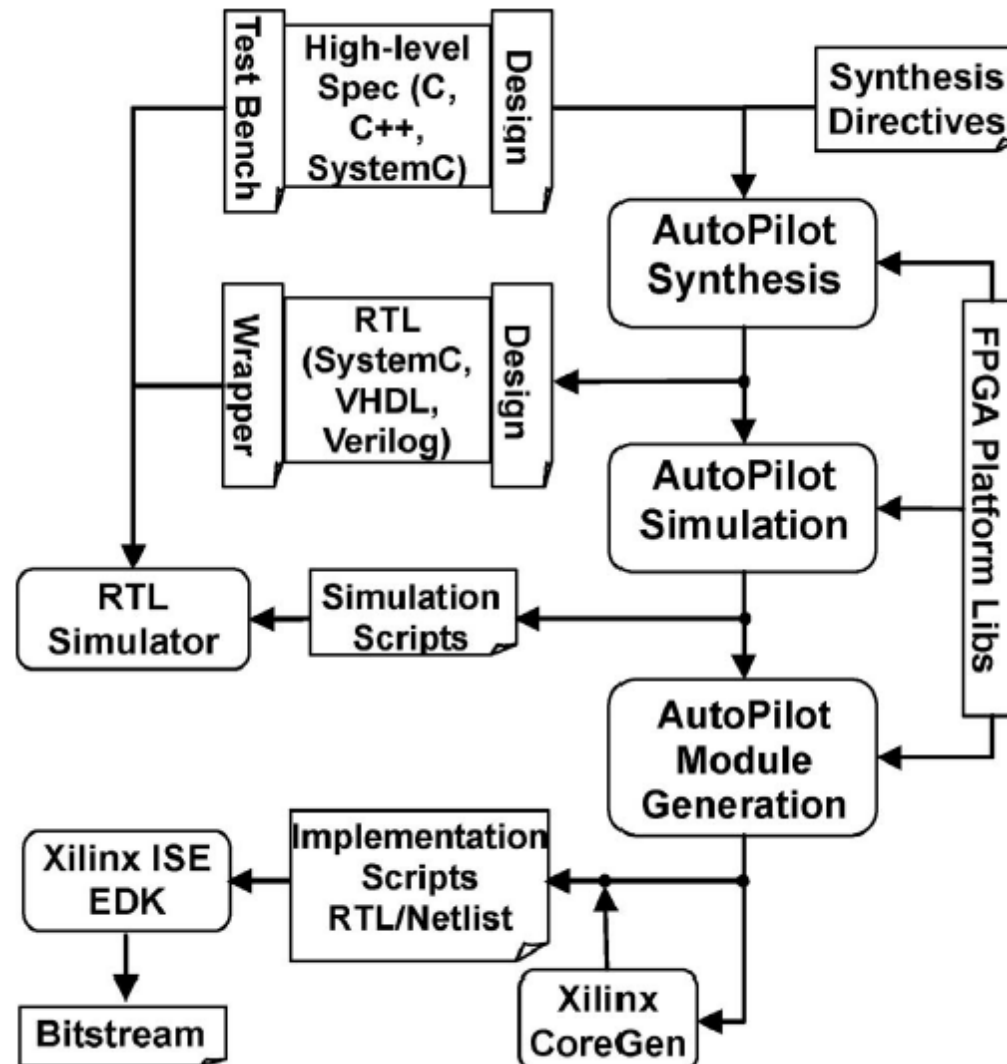
Performance Feature: Loop Pipelining

for (int i = 0; i < N; i++)
    sum[i] = a + b + c + d
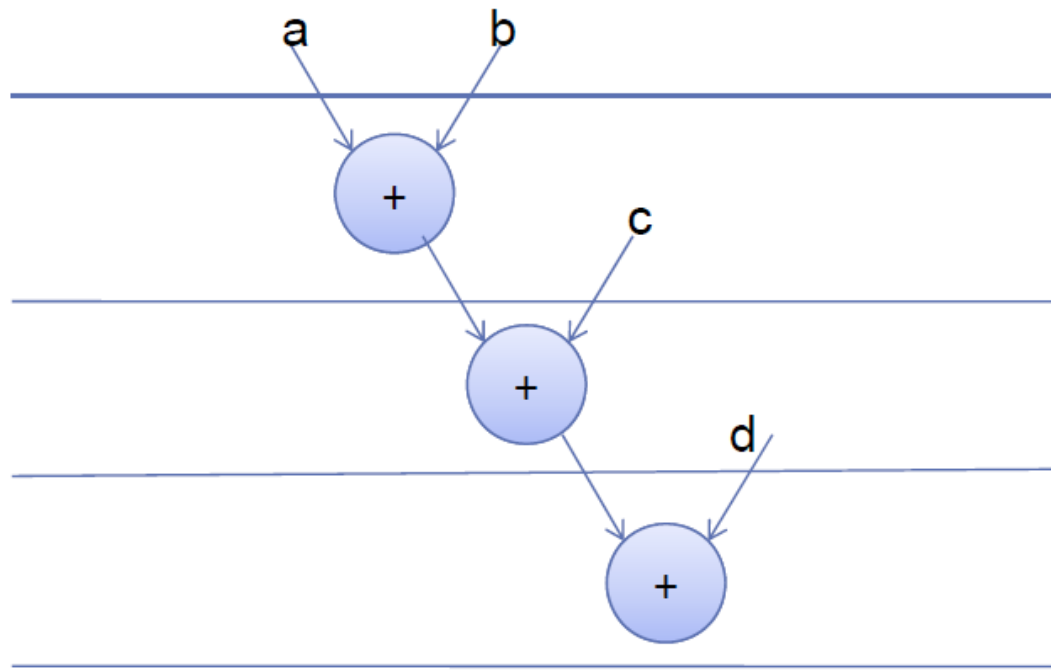
Sequential Execution

Not efficient!

- 3 Cycles/Iteration
- Total Cycles: 3N
- Adders: 3
- Adder Utilization: 33%

```
for (int i = 0; i < N; i++)
    sum[i] = a + b + c + d
```

1. Sequential Execution

Cycles: 1 2 3 4 5 6 7 8 9 …

i = 0   [+][+][+]

i = 1   [+][+][+]

i = 2   [+][+][+]

⋮
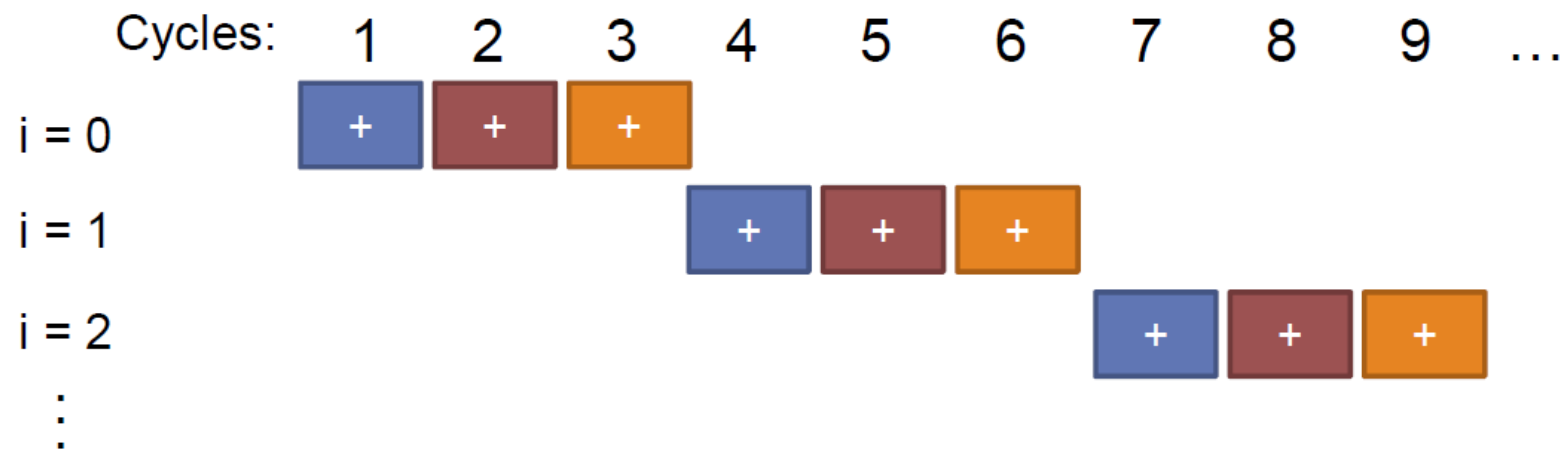
Poop Pipelining

#pragma pipeline

```
for (int i = 0; i < N; i++)
    sum[i] = a + b + c + d
```

3. Parallel Execution : Loop pipelining
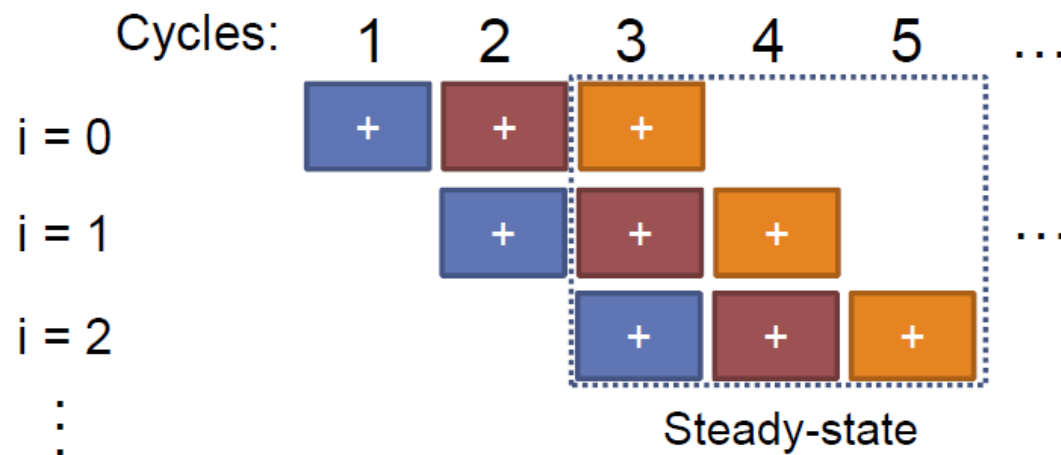
Steady-state

Efficient!

- 1 Cycles/Iteration (steady-state)
- Total Cycles: N+2
- Adders: 3
- Adder Utilization: 100%

16

High-quality and rapid design using HLS: Why?

- Embedded processors are in almost every SoC
- Huge silicon capacity requires a higher level of abstraction
  - With the HLS, the code density can be easily reduced by 7x-10x in C/C++
- Behavioral IP reuse improves design productivity
- Verification drives the acceptance of HLS
  - It avoids slow and error-prone manually RTL cording
- Trend toward extensive use of accelerators and heterogeneous SoCs

Faster Adoption of HLS Tools with FPGAs: Why?

- Less pressure for formal verification
  - Iterations of the FPGA design can avoid huge manufacturing costs

- Ideal for platform-based synthesis
  - Achieve higher quality of results (QoR)

- More pressure for time-to-market
  - Designers may accept increased performance, power, or cost in order to reduce design time

- Accelerated or reconfigurable computing calls for C/C++ based compilation/synthesis to FPGAs

Overview of HLS Tools

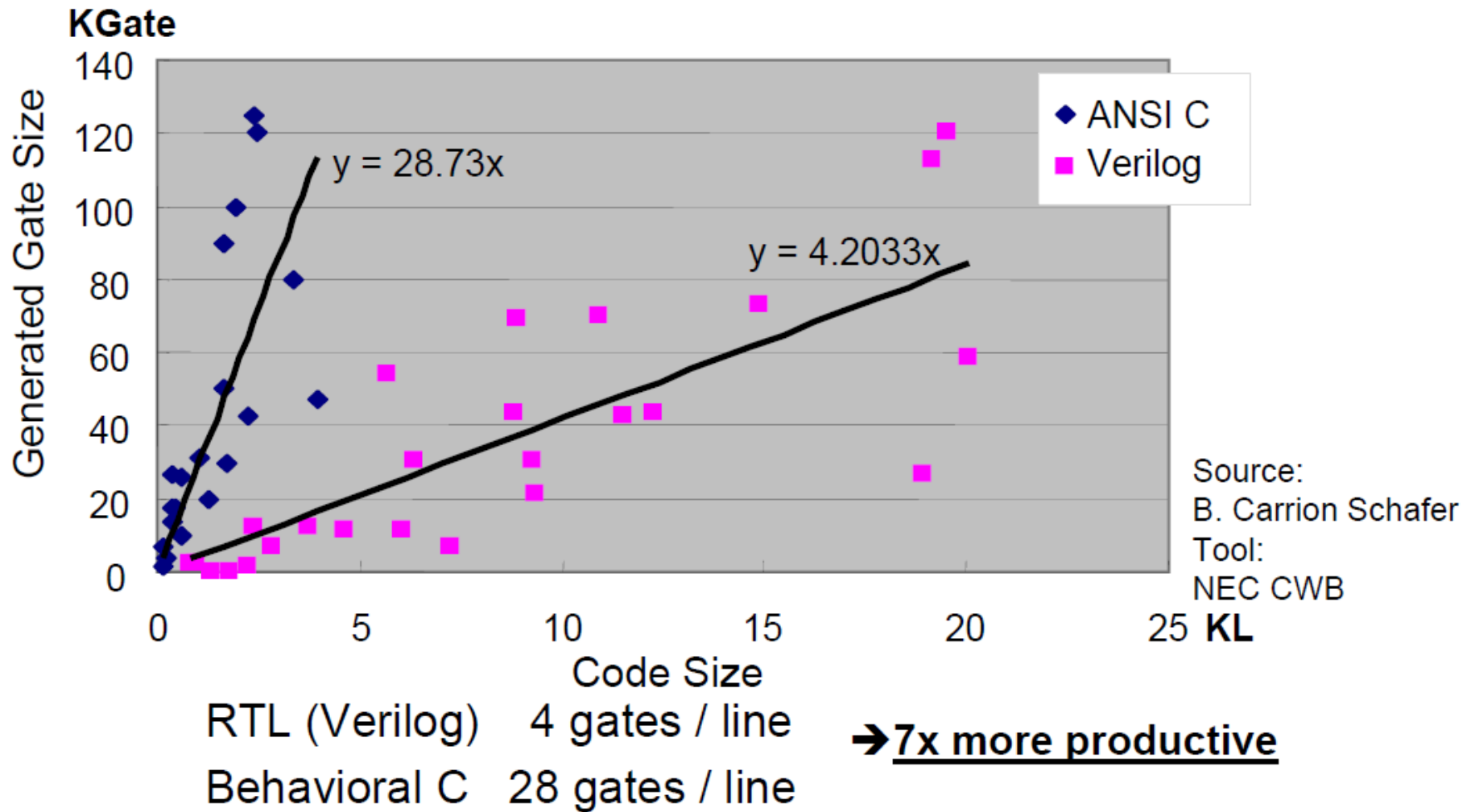| Status	Compiler	Owner	License	Input	Output	Year	Domain	TestBench	FP	FixP
In Use	eXCite	Y Explorations	Commercial	C	VHDL/Verilog	2001	All	Yes	No	Yes
	CoDeveloper	Impulse Accelerated	Commercial	Impulse-C	VHDL Verilog	2003	Image Streaming	Yes	Yes	No
	Catapult-C	Calypto Design Systems	Commercial	C/C++ SystemC	VHDL/Verilog SystemC	2004	All	Yes	No	Yes
	Cynthesizer	FORTE	Commercial	SystemC	Verilog	2004	All	Yes	Yes	Yes
	Bluespec	BlueSpec Inc.	Commercial	BSV	SystemVerilog	2007	All	No	No	No
	CHC	Altium	Commercial	C subset	VHDL/Verilog	2008	All	No	Yes	Yes
	CtoS	Cadence	Commercial	SystemC TLM/C++	Verilog SystemC	2008	All	Only cycle accurate	No	Yes
	DK Design Suite	Mentor Graphics	Commercial	Handel-C	VHDL Verilog	2009	Streaming	No	No	Yes
	GAUT	U. Bretagne	Academic	C/C++	VHDL	2010	DSP	Yes	No	Yes
	MaxCompiler	Maxeler	Commercial	MaxJ	RTL	2010	DataFlow	No	Yes	No
	ROCCC	Jacquard Comp.	Commercial	C subset	VHDL	2010	Streaming	No	Yes	No
	Synphony C	Synopsys	Commercial	C/C++	VHDL/Verilog SystemC	2010	All	Yes	No	Yes
	Cyber-WorkBench	NEC	Commercial	BDL	VHDL Verilog	2011	All	Cycle/Formal	Yes	Yes
	LegUp	U. Toronto	Academic	C	Verilog	2011	All	Yes	Yes	No
	Bambu	PoliMi	Academic	C	Verilog	2012	All	Yes	Yes	No
	DWARV	TU. Delft	Academic	C subset	VHDL	2012	All	Yes	Yes	Yes
	VivadoHLS	Xilinx	Commercial	C/C++ SystemC	VHDL/Verilog SystemC	2013	All	Yes	Yes	Yes
N/A	Trident	Los Alamos NL	Academic	C subset	VHDL	2007	Scientific	No	Yes	No
	CHiMPS	U. Washington	Academic	C	VHDL	2008	All	No	No	No
	Kiwi	U. Cambridge	Academic	C#	Verilog	2008	.NET	No	No	No
	gcc2verilog [45]	U. Korea	Academic	C	Verilog	2011	All	No	No	No
	HercuLeS	Ajax Compiler	Commercial	C/NAC	VHDL	2012	All	Yes	Yes	Yes
Abandoned	Napa-C	Sarnoff Corp.	Academic	C subset	VHDL/Verilog	1998	Loop	No	No	No
	DEFACTO	U. South Cailf.	Academic	C	RTL	1999	DSE	No	No	No
	Garp	U. Berkeley	Academic	C subset	bitstream	2000	Loop	No	No	No
	MATCH	U. Northwest	Academic	MATLAB	VHDL	2000	Image	No	No	No
	PipeRench	U.Carnegie M.	Academic	DIL	bitstream	2000	Stream	No	No	No
	SeaCucumber	U. Brigham Y.	Academic	Java	EDIF	2002	All	No	Yes	Yes
	SA-C	U. Colorado	Academic	SA-C	VHDL	2003	Image	No	No	No
	SPARK	U. Cal. Irvine	Academic	C	VHDL	2003	Control	No	No	No
	AccelDSP	Xilinx	Commercial	MATLAB	VHDL/Verilog	2006	DSP	Yes	Yes	Yes
	C2H	Altera	Commercial	C	VHDL/Verilog	2006	All	No	No	No
	CtoVerilog	U. Haifa	Academic	C	Verilog	2008	All	No	No	No

J. Cong et. al, "A survey and evaluation of FPGA high-level synthesis tools," IEEE Trans. on CAD, 2015.

Productivity

KGate

Generated Gate Size

$y = 28.73x$

$y = 4.2033x$

- ◆ ANSI C
- ■ Verilog

Code Size

Source:
B. Carrion Schafer
Tool:
NEC CWB

RTL (Verilog) 4 gates / line
Behavioral C 28 gates / line

→**7x more productive**

RTL Expert vs. HLS Expert

SPHERE DECODER IMPLEMENTATION RESULTS

Metric	RTL Expert	AutoPilot Expert	Difference (%)
Dev. time (man-weeks)	16.5	15	−9
LUTs	32 708	29 060	−11
Registers	44 885	31 000	−31
DSP48s	225	201	−11
18K BRAMs	128	99	−26

8×8 RVD-QRD IMPLEMENTATION RESULTS

Metric	RTL Expert	AutoPilot Expert	AutoPilot Expert
Dev. time (man-weeks)	4.5	3	5
LUTs	5082	6344	3862
Registers	5699	5692	4931
DSP48s	30	46	30
18K BRAMs	19	19	19

J. Cong et. al, "High-level synthesis for FPGAs: From prototyping to deployment," IEEE Trans. on CAD, 2011.

Walk Through HLS Design

Vivado HLS Design Flow

- Design C/C++ code for your target specification in advance

    ↓

- Create a new project
- Refactoring for HLS suitable C/C++
- High-level synthesis
- RTL-C Co-verification (Optional)
- IP generation

    ↓

- Integrate HW on Vivado block diagram design
- Design SW on XSDK
- Run on your FPGA board

Design a Specification by C/C++

- See, Github for lecture 9: float_mult.c

```c
#include <stdio.h>
#include <stdlib.h>

#define ERROR_THRESH 0.1

void float_mult(float a, float b, float *c)
{
  *c = a * b;
}

int main(void)
{
  float fres, fia, fib, fref, err;
  int ec = 0;
  for (unsigned int i = 0; i < 100; i++) {
    fia = (rand() % 1000) / 100.0;
    fib = (rand() % 1000) / 100.0;
    float_mult(fia, fib, &fres);
    fref = fia * fib;

    err = fref - fres;
    if( err < 0) err *= -1.0;

    printf("fia=%f fib=%f fres=%f fref=%f", fia, fib, fres, fref);
    if ( err > ERROR_THRESH) {
        ec++;
        printf(" ERROR¥n");
    } else {
        printf("¥n");
    }
  }

  if( ec == 0) printf("[Test Passed]¥n"); else printf("[Test Failed]¥n");

  return 0;
}
```

Create a New Project

- Windows: Start > All Programs > Xilinx Design Tools > Vivado 2017.4 > Vivado HLS > Vivado HLS

- Linux (Ubuntu):

#source /opt/Xilinx/Vivado/2017.4/settings64.sh

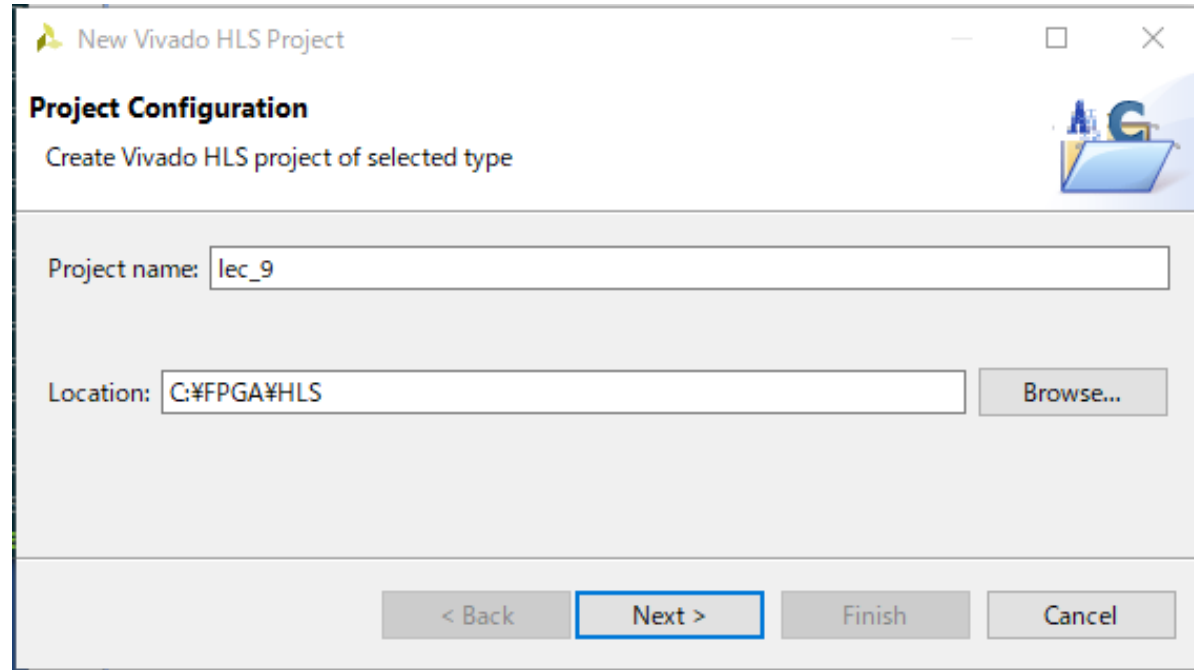#vivado_hls &

On the Vivado HLS Welcome Page

- In the Welcome Page, select "Create New Project" to open the Project wizard.

On Project Configuration Page

- a. Enter the project name as "lec_9"
- b. Enter the project Location as "C:¥FPGA¥HLS" (Windows), "/home/(usrname)/FPGA/HLS" (Ubuntu)
- c. Click Next

On an Add/Remove Files Dialog

- Enter the following information to specify the C design files:
  - a. Click Add Files.
  - b. Select "**float_mult.c**" and click Open.
 - c. Use Browse button to specify "**float_mult**" as the top-level function.
  - d. Click Next.
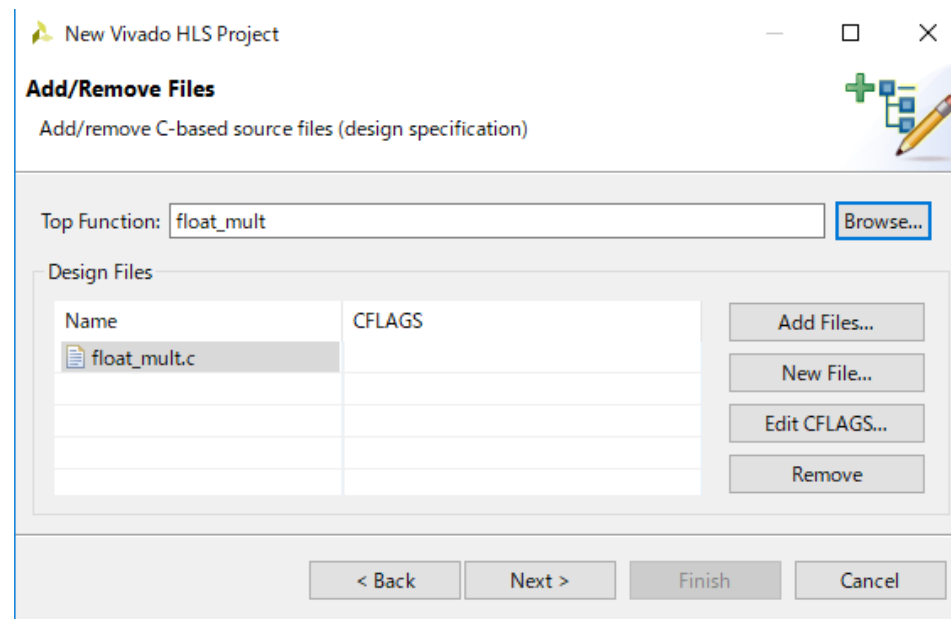
Add/Remove C-based Testbench Files

- Click the Add Files button to include both test bench files "float_mult.c"

- Then, Click Next.

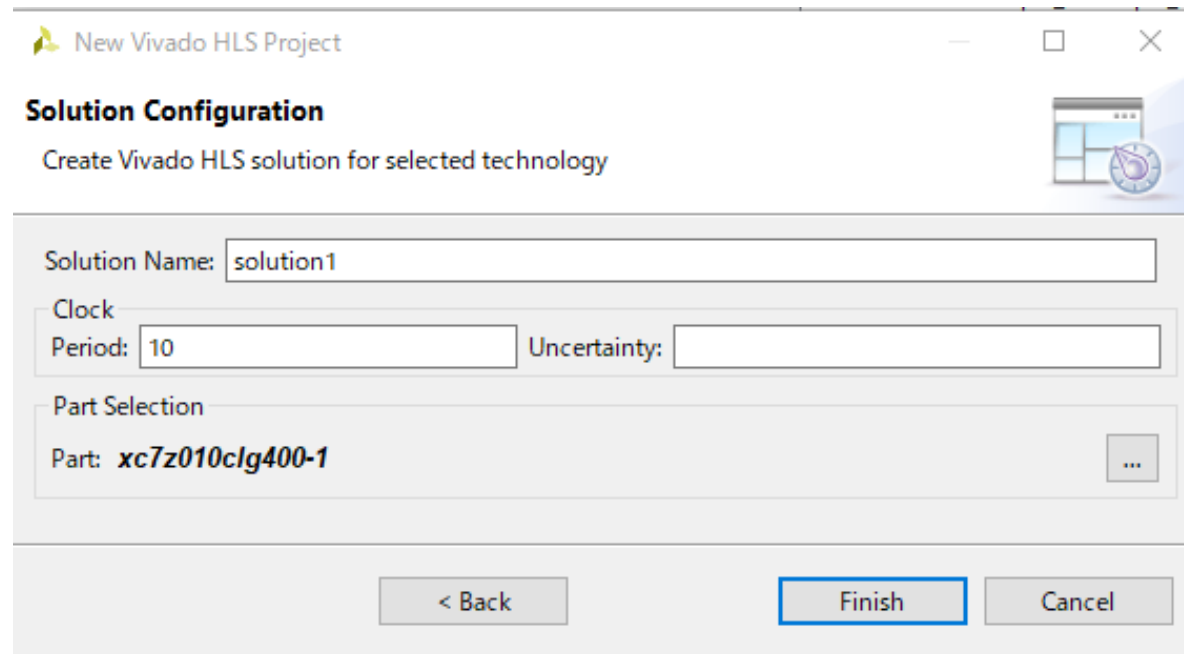- Both C simulation and RTL co-simulation execute in subdirectories of the solution.
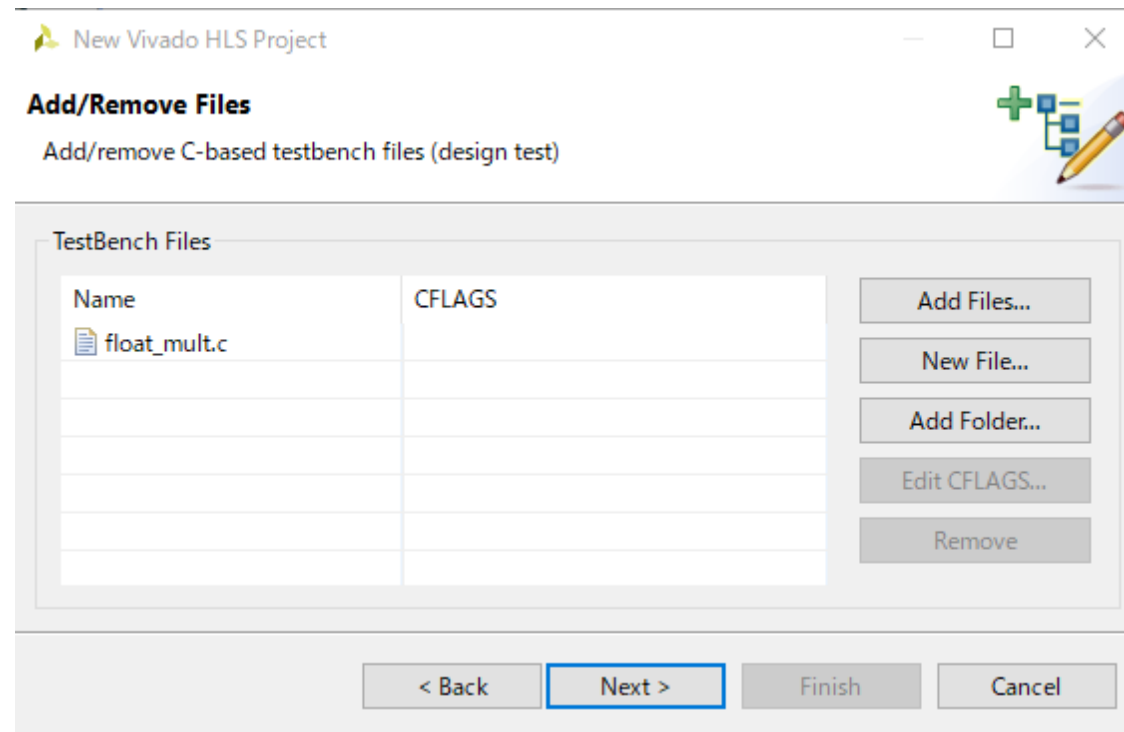
Solution Configuration

- Accept the default solution name (solution1), clock period (10 ns), and clock uncertainty (defaults to 12.5% of the clock period, when left blank/undefined)
- Click the part selection button to open the part selection window
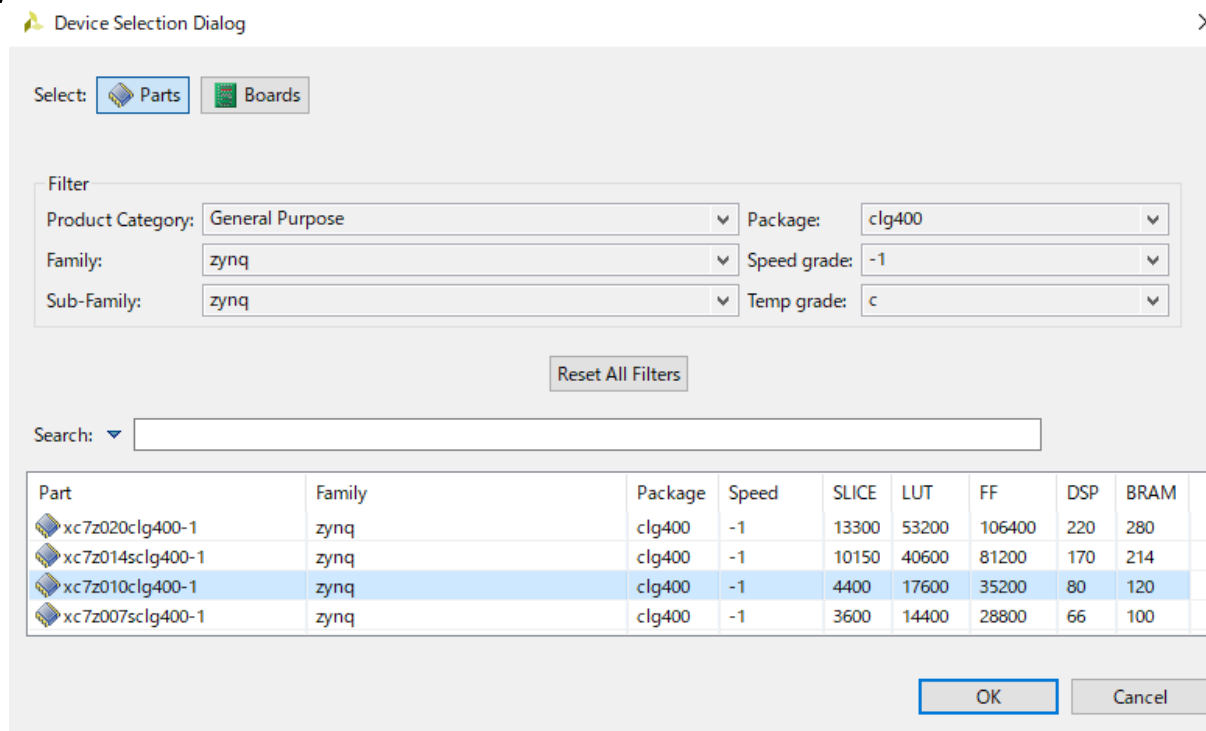
Cont'd

- Carefully Select Your FPGA on the Zybo!!
  - From the list of available devices
- Click "OK", and click "Finish"

Vivado HLS GUI
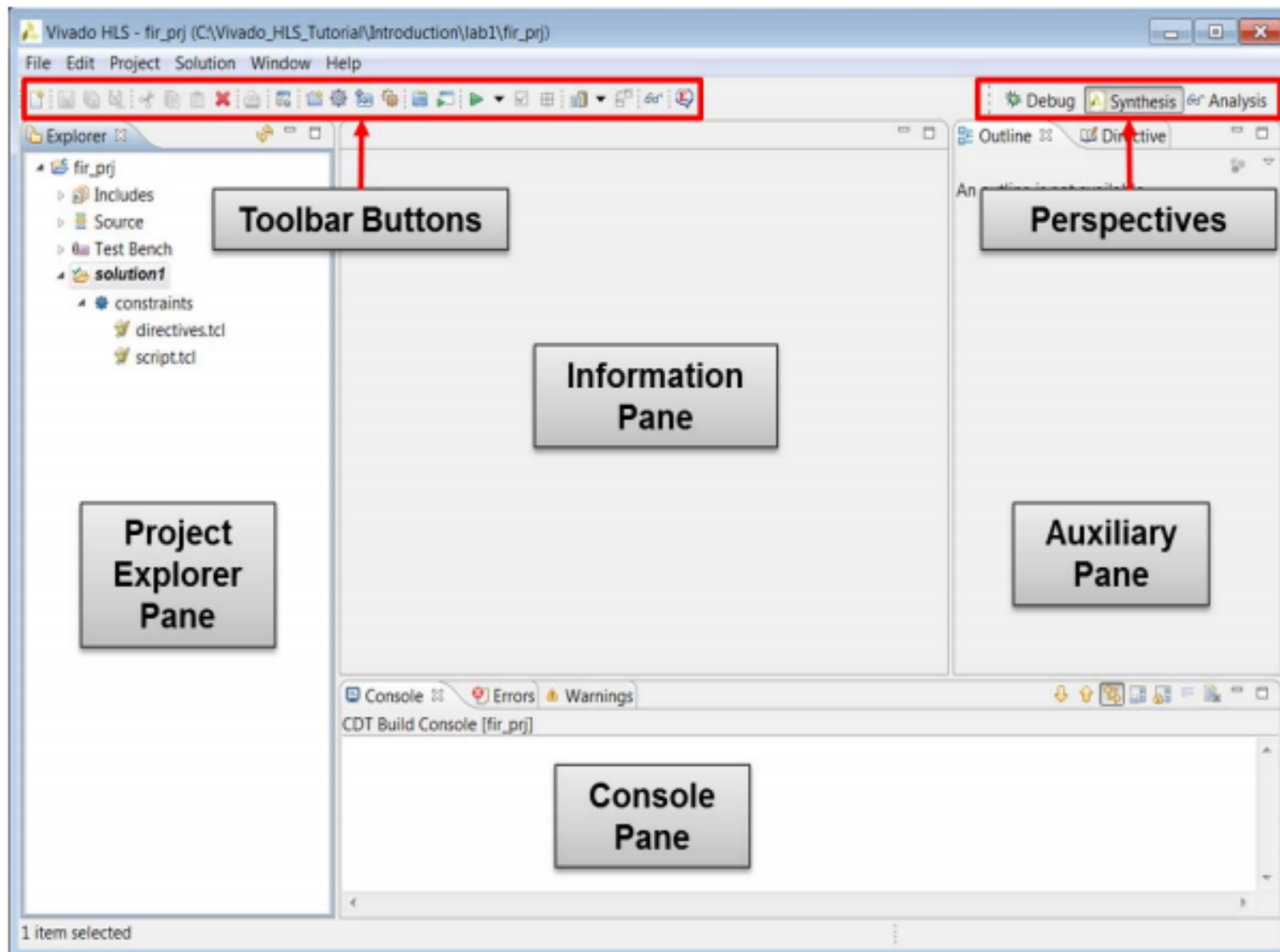
Refactoring your C/C++ Code

- See, Github: hls_float_mult.c
- Define an union variable
- Insert pragmas

```c
 6 typedef union _ftoi {
 7    float f;
 8    unsigned int u;
 9 } ftoi;
10
11 int float_mult(float a, float b, float *c)
12 {
13 #pragma HLS INTERFACE s_axilite register port=c bundle=slv0
14 #pragma HLS INTERFACE s_axilite port=b bundle=slv0
15 #pragma HLS INTERFACE s_axilite port=a bundle=slv0
16 #pragma HLS INTERFACE s_axilite register port=return bundle=slv0
17
18    static int cycle = 0;
19
20    *c =  a * b;
21    cycle++;
22
23    return cycle;
24 }
```

Validate the C/C++ Code

- 4. Click the Run C Simulation button, or use menu Project > Run C Simulation, to compile and execute the C design.
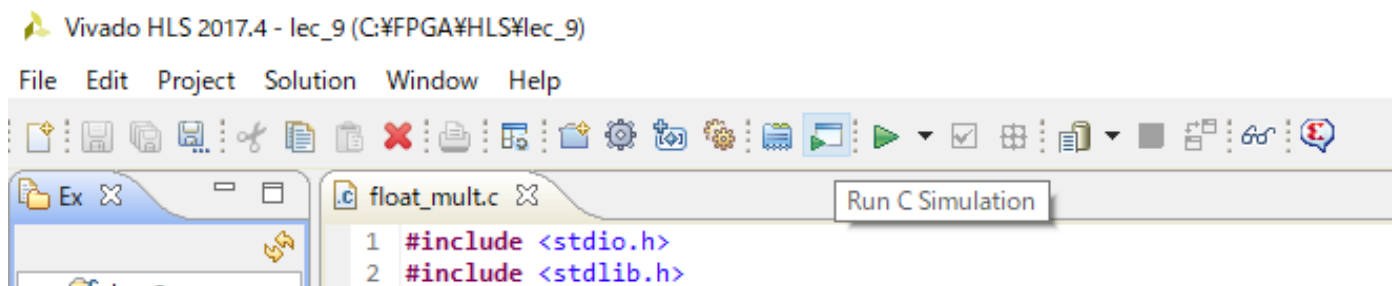

- 5. In the C Simulation dialog box, click OK.
  - The Console pane confirms the simulation executed successfully.
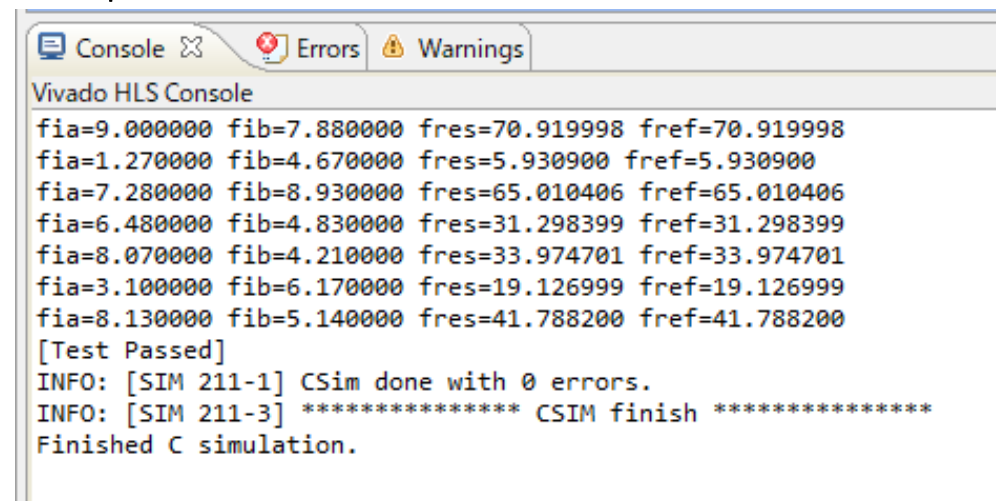
High-Level Synthesis

- Click the Run C Synthesis toolbar button or use the menu Solution > Run C Synthesis > Active Solution

- When synthesis completes, the report file opens automatically

    - Because the synthesis report is open in the Information pane, the Outline tab in the Auxiliary pane automatically updates to reflect the report information.
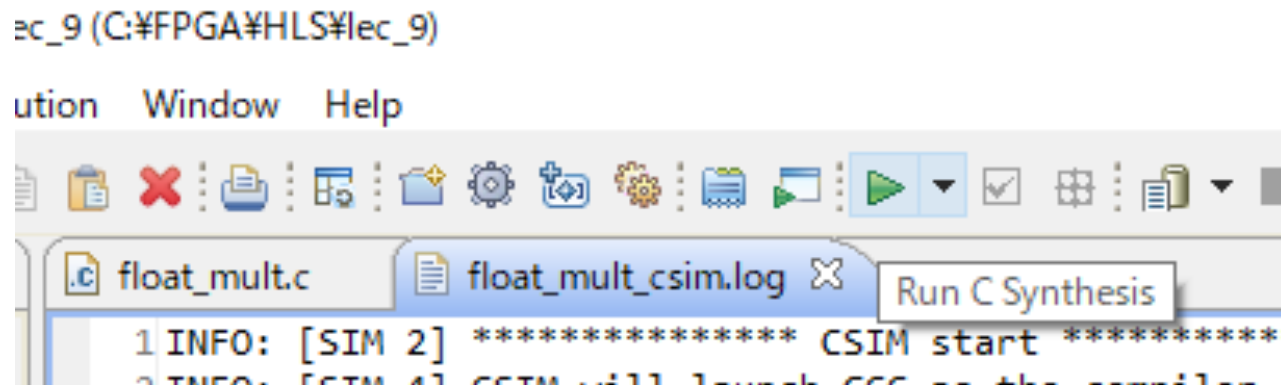
Performance Estimation

- Check "Performance Estimates" in the Outline tab

- In the Performance Estimates pane, you can see that the clock period is set to 10 ns. Vivado HLS targets a clock period of Clock Target minus Clock Uncertainty (10.00-1.25 = 8.75 ns in this example).
  - The clock uncertainty ensures there is some timing margin available for the (at this stage) unknown net delays due to place-and-routing

- The estimated clock period (worst-case delay) is 6.70 ns, which meets the 8.75 ns timing requirement

Performance Estimates

⊟ **Timing (ns)**

⊟ **Summary**

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	6.70	1.25

⊟ **Latency (clock cycles)**

⊟ **Summary**

Latency		Interval		
min	max	min	max	Type
4	4	4	4	none

Utilization Estimation

- The resource utilization numbers are estimates because RTL synthesis might be able to perform additional optimizations, and these figures might change after RTL synthesis.

Utilization Estimates

⊟ Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	39
FIFO	-	-	-	-
Instance	0	3	325	617
Memory	-	-	-	-
Multiplexer	-	-	-	33
Register	-	-	102	-
Total	0	3	427	689
Available	120	80	35200	17600
Utilization (%)	0	3	1	3

Interface Report

- The Interface section shows the ports and I/O protocols created by interface synthesis

Interface

☐ Summary

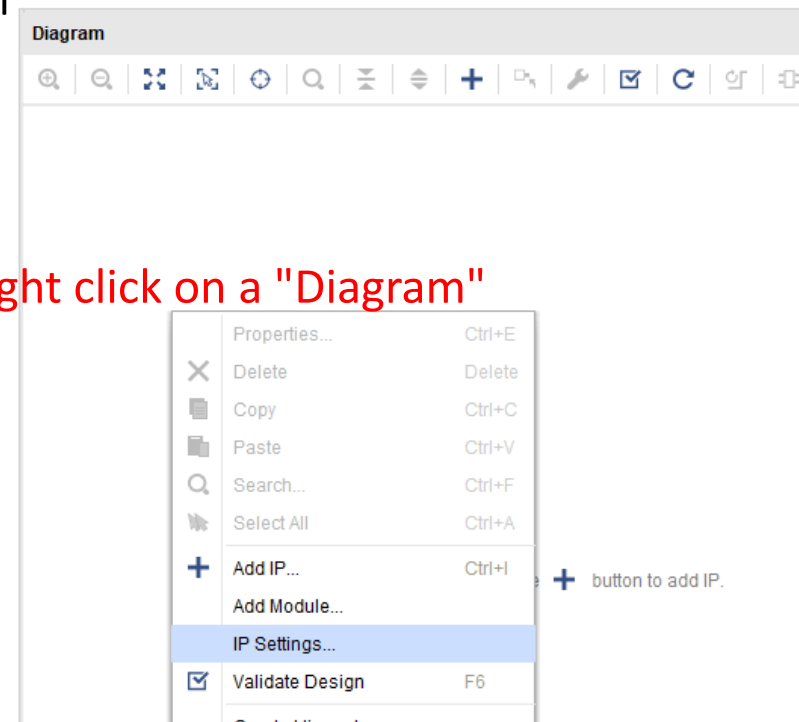| RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_slv0_AWVALID	in	1	s_axi	slv0	pointer
s_axi_slv0_AWREADY	out	1	s_axi	slv0	pointer
s_axi_slv0_AWADDR	in	6	s_axi	slv0	pointer
s_axi_slv0_WVALID	in	1	s_axi	slv0	pointer
s_axi_slv0_WREADY	out	1	s_axi	slv0	pointer
s_axi_slv0_WDATA	in	32	s_axi	slv0	pointer
s_axi_slv0_WSTRB	in	4	s_axi	slv0	pointer
s_axi_slv0_ARVALID	in	1	s_axi	slv0	pointer
s_axi_slv0_ARREADY	out	1	s_axi	slv0	pointer
s_axi_slv0_ARADDR	in	6	s_axi	slv0	pointer
s_axi_slv0_RVALID	out	1	s_axi	slv0	pointer
s_axi_slv0_RREADY	in	1	s_axi	slv0	pointer
s_axi_slv0_RDATA	out	32	s_axi	slv0	pointer
s_axi_slv0_RRESP	out	2	s_axi	slv0	pointer
s_axi_slv0_BVALID	out	1	s_axi	slv0	pointer
s_axi_slv0_BREADY	in	1	s_axi	slv0	pointer
s_axi_slv0_BRESP	out	2	s_axi	slv0	pointer
ap_clk	in	1	ap_ctrl_hs	float_mult	return value
ap_rst_n	in	1	ap_ctrl_hs	float_mult	return value
interrupt	out	1	ap_ctrl_hs	float_mult	return value

IP Creation

- The final step in the High-Level Synthesis flow is to package the design as an IP block for use with other tools in the Vivado Design Suite

- 1. Click the Export RTL toolbar button or use the menu Solution > Export RTL

- 2. Ensure the Format Selection drop-down menu shows IP Catalog

- 3. Click OK. The IP packager creates a package for the Vivado IP Catalog

- 4. Expand Solution1 in the Explorer

- 5. Expand the impl folder created by the Export RTL command

- 6. Expand the ip folder and find the IP packaged as a zip file, ready for adding to the Vivado IP Catalog (See, right)
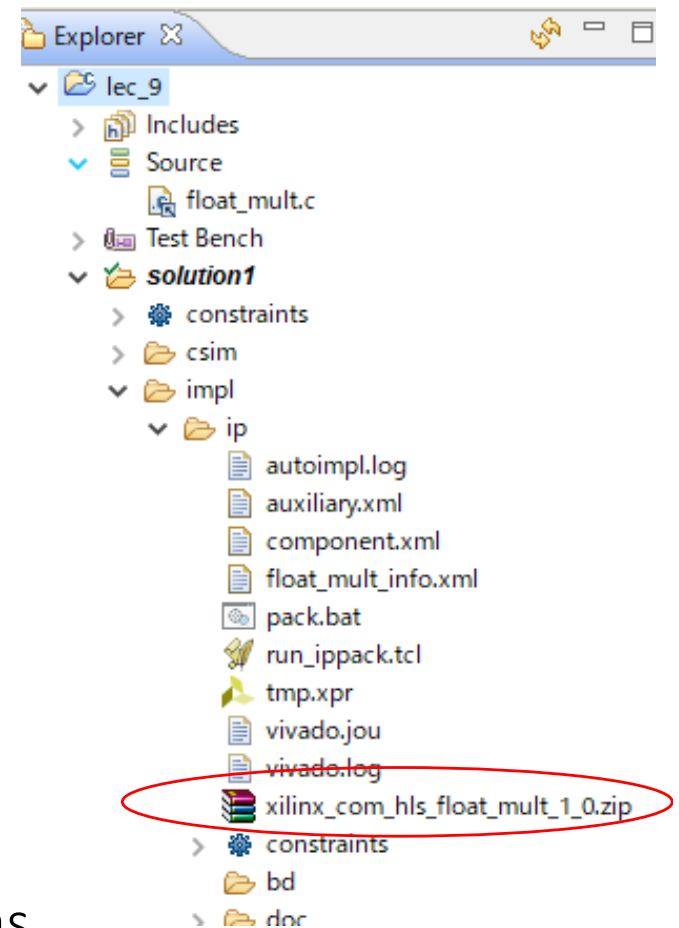
Import to IP Catalog

- Create a new project on Vivado, then create a block design
  - Project name as "hls_walk_through_1"
  - RTL project, no add sources, constraints, and specify your Zybo board
  - ~~Make sure the constraint file "Zybo-Z7-Master.xdc" is loaded~~

- Click an "IP Settings..." on a diagram, then open a project settings window

Diagram

Right click on a "Diagram"

	Properties...	Ctrl+E
✕	Delete	Delete
	Copy	Ctrl+C
	Paste	Ctrl+V
🔍	Search...	Ctrl+F
	Select All	Ctrl+A
＋	Add IP...	Ctrl+I
	Add Module...	
	IP Settings...	
☑	Validate Design	F6
	Create Hierarchy...	

＋ button to add IP.

Add a Path to IP Repositories

- Add a path to Repository manager for the IP generated on Vivado HLS
  - In the tutorial, the IP path is "C:/FPGA/HLS/lec_9/solution1/impl/ip"
  - Make sure your "Float_mult" is loaded
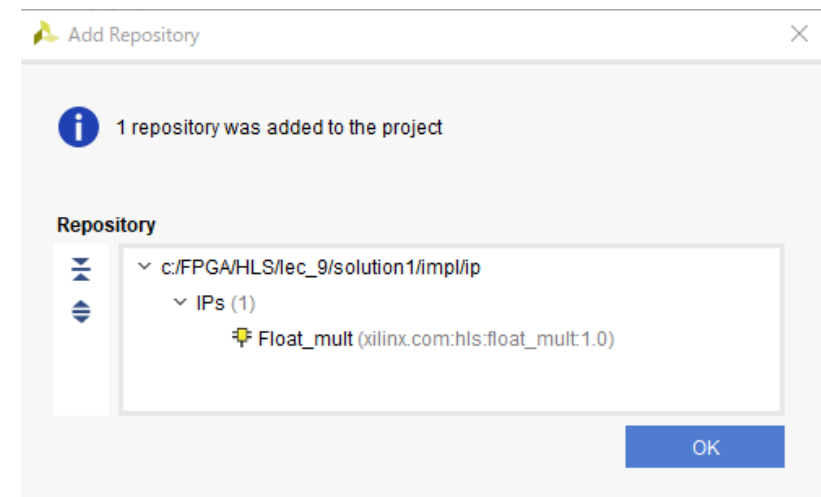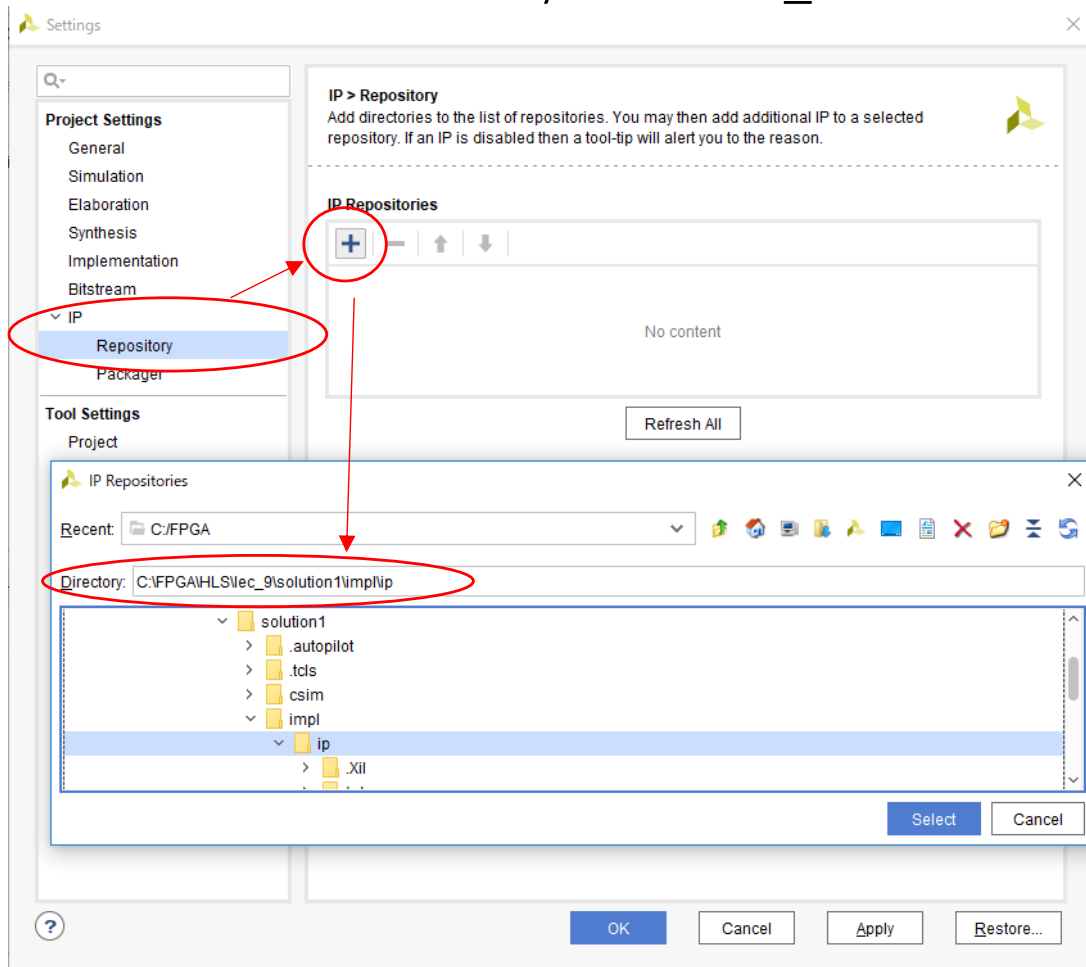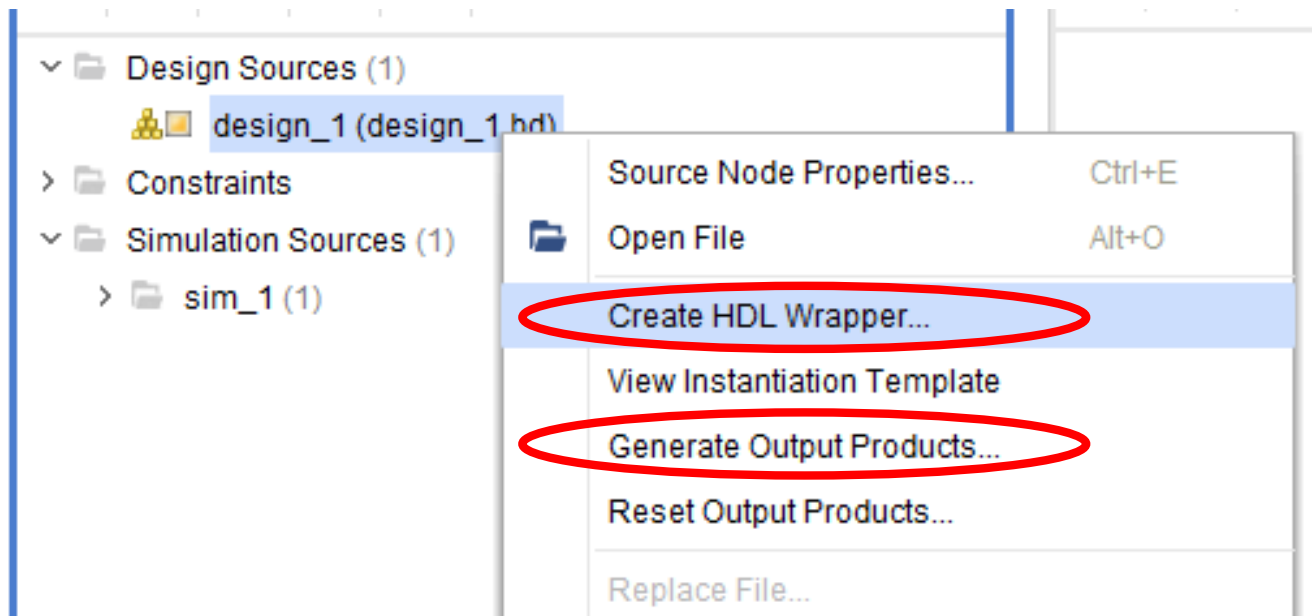
System Generation

- On Block Design, instance a "ZYNQ7 Processing System"

- Then, also instance your IP core "float_mult"

- Next, concatenate with them (Processor System Reset and AXI Interconnect are inserted automatically)
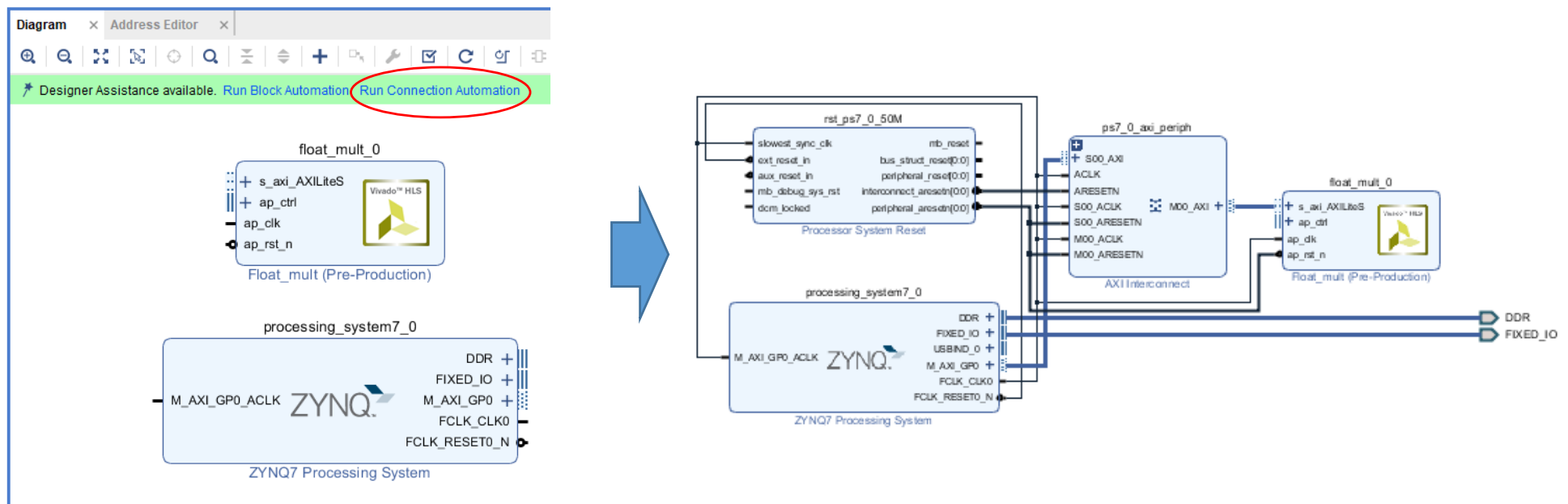
Generate Bitstream

To re-use HDL design flow, do following steps:

  1. Right-click on "design_1", and select "Generate Output Products...", then "Generate"

  2. Again, Right-click on "design_1", and select "Create HDL Wrapper...", then "OK"

Synthesis Hardware and Export

- Click "Generate Bitstream", then "Yes" and "OK"

- After finish bitstream generation, then "Cancel"

- In "Menu", select "File" and "Export", then "Export Hardware"

- Check "Include bitstream", then "OK"
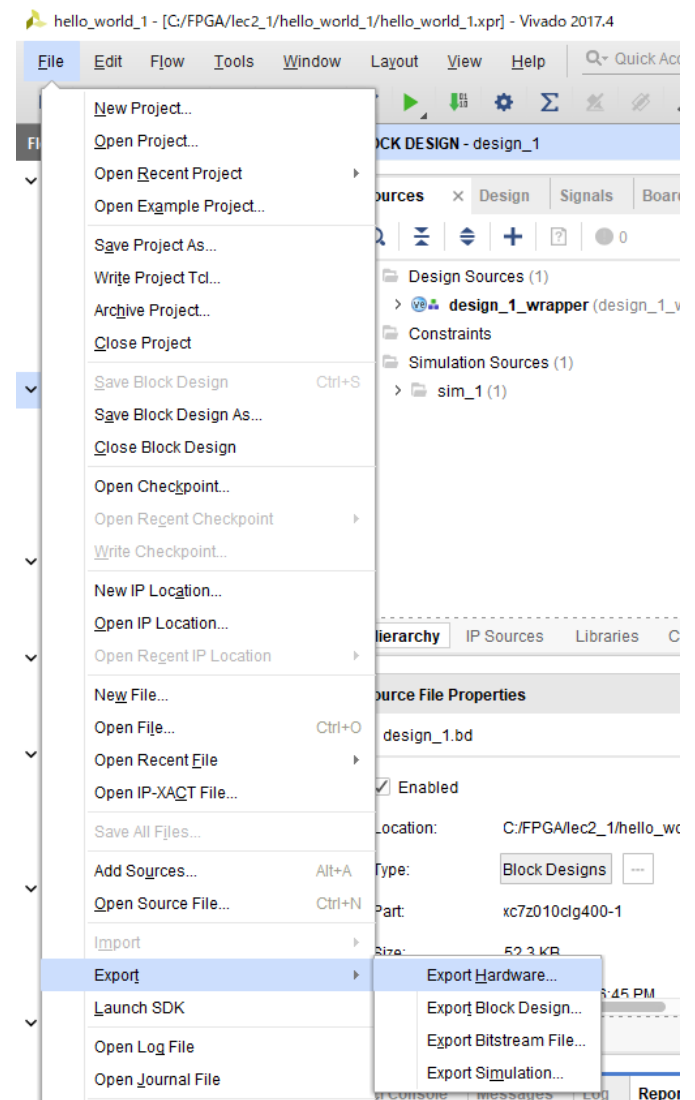
Developing SW

- Launch SDK on Vivado

- From the SDK File menu, select New > Application Project.

    a. In the New Project dialog enter the project name "zynq_float_mult"

    b. Click Next.

    c. Select the "Hello World" template.

    d. Click Finish
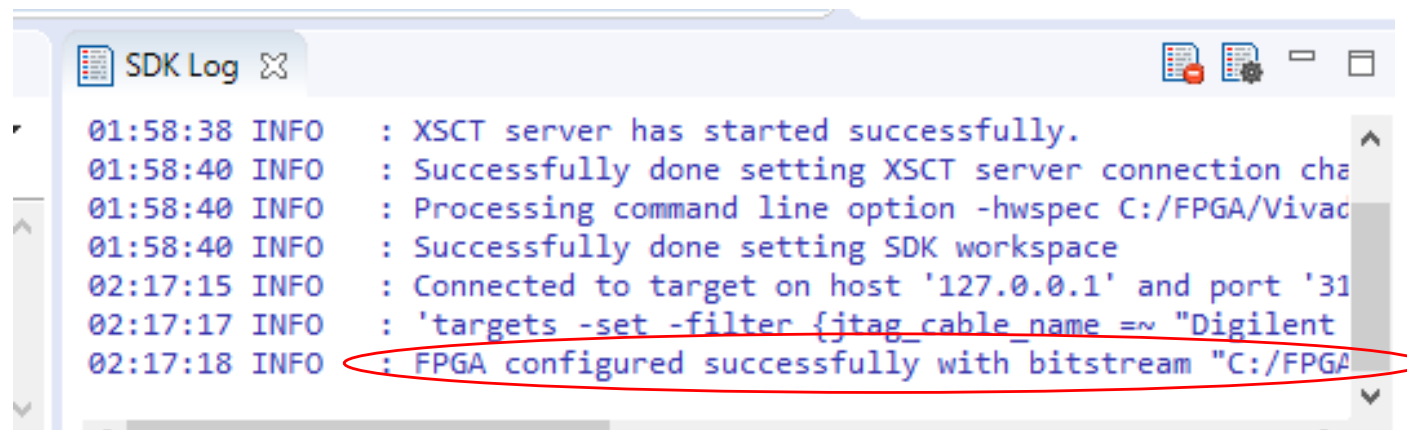
Execute "Hello World" SW

- Power up the Zybo board and test the Hello World application

- Ensure the board has all the connections to allow you to download the bitstream on the FPGA device

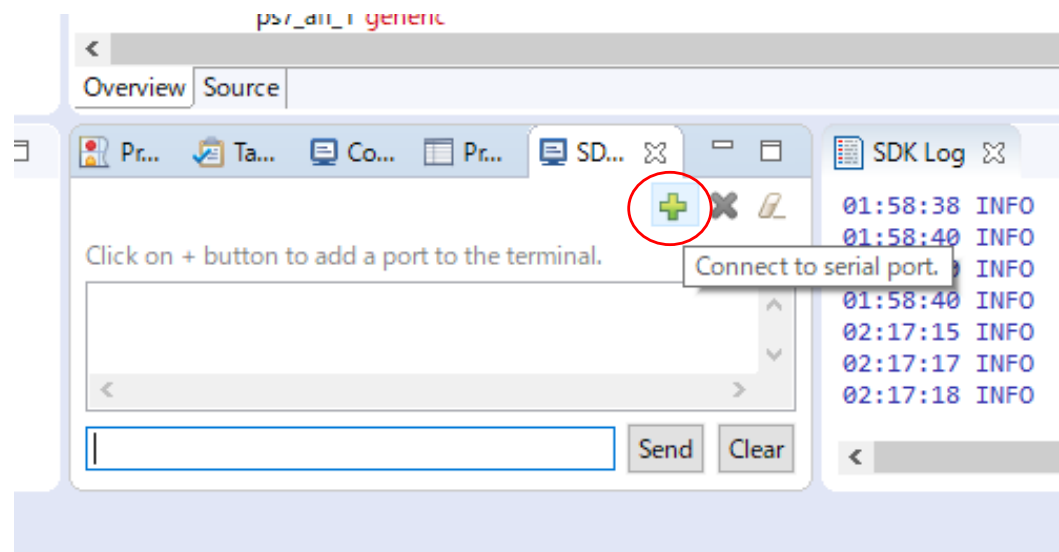- Click Xilinx > Program FPGA (or toolbar icon)

Cont'd

- Click on SDK Terminal and click on add button to add a port to the terminal

   a. Click the Connect icon (See, bottom)

   b. Select Connection Type > Serial.

   c. Select the COM port to which the USB UART cable is connected

   d. Change the Baud Rate to 115200

   e. Click OK to exit the Terminal Settings dialog box.

Terminal Settings Dialog

- Right-click the application project "zynq_float_mult" in the Explorer panel
    - a. Click Run As > Launch on Hardware.

- Switch to the Terminal tab and confirm that "Hello World" was received
    - Or, you can use TeraTermianl (Windows) GtkTerm (Ubuntu)
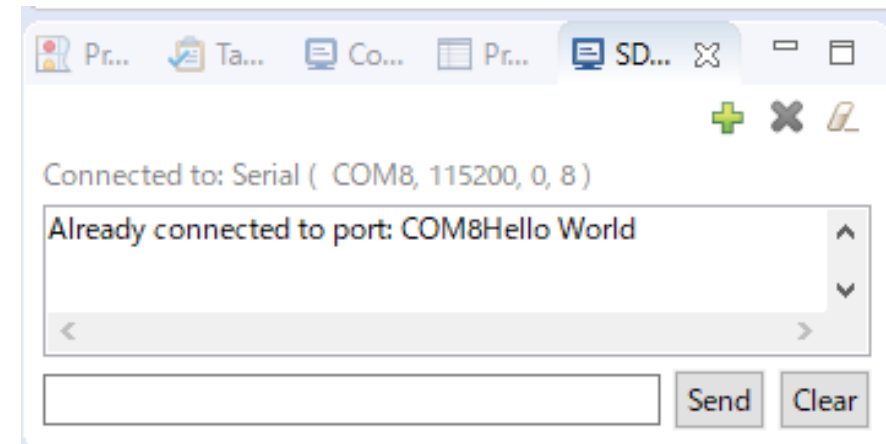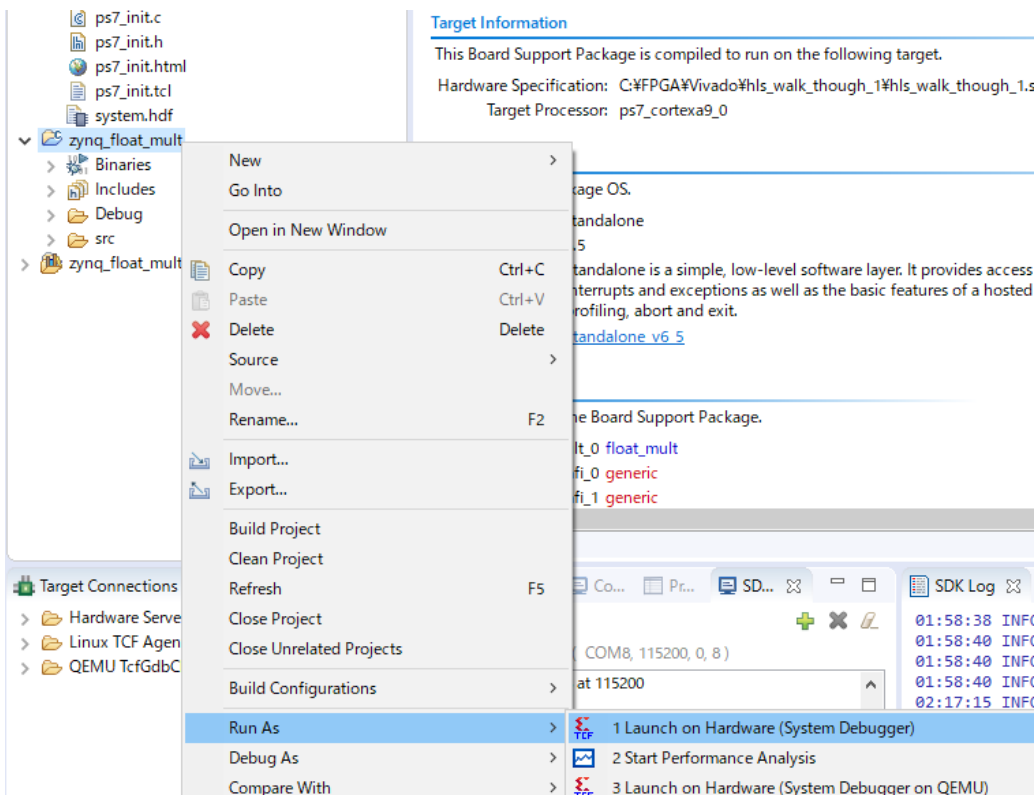
Modify SW to Communicate with Your IP Core

- The "float_mult" function has "a" and "b" as inputs and "c" as an output

- Following processing exists in the API generated by Vivado HLS
  - Set value to a
  - Set value to b
  - Read the value of c

- In order to control the hardware, there is an API that performs the following processing
  - Start operation
  - Terminate of calculation

APIs for a Generated IP Core

- The IP core API is stored in the following directory (in this example)

  design_1_wrapper_hw_platform_0/drivers/float_top_v1_0/ src

- In the directory, the following files are stored
  - xfloat_top_hw.h
  - xfloat_top_linux.c
  - xfloat_top_sinit.c
  - xfloat_top.c
  - xfloat_top.h

- The detail of the defined API is described in "ug902-vivado-high-level-synthesis.pdf"

SW Code for an ARM Processor on Zybo

- See, Github: sw_code.c

```c
#include <stdio.h>
#include "platform.h"
#include "xfloat_mult.h"
#include "xparameters.h"

typedef union _ftoi {
    float f;
    unsigned int u;
} ftoi;

// from xparameters.h
#define XF_DEVICE_ID    XPAR_FLOAT_MULT_0_DEVICE_ID

XFloat_mult XFT;// Instance for Vivado HLS Generated IP

// Testbench
float ref_float_mult( float a, float b)
{
    return a * b;
}

int main()
{
    int i,Status;
    XFloat_mult_Config *ConfigPtr;
    ftoi ref_a, ref_b, out_c;
    ftoi in_a, in_b;
    int cycle;

    // Initialization
    init_platform();
    XFloat_mult_Initialize( &XFT,XPAR_FLOAT_MULT_0_DEVICE_ID);

    ConfigPtr = XFloat_mult_LookupConfig(XF_DEVICE_ID);
    Status = XFloat_mult_CfgInitialize(&XFT, ConfigPtr);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }
    Status = XFloat_mult_IsIdle(&XFT);
```

```c
Status = XFloat_mult_IsIdle(&XFT);

for (i=0;i<10;i++) {
    printf("TEST LOOP %d:\n",i);
    ref_a.f = (float)i;
    ref_b.f = (float)(i*100 + i);
    while(!XFloat_mult_IsReady(&XFT)) ;   // Check Ready Signal
    XFloat_mult_Set_a(&XFT, ref_a.u);   // Set register a
    XFloat_mult_Set_b(&XFT, ref_b.u);   // Set register b
    in_a.u = XFloat_mult_Get_a(&XFT);
    in_b.u = XFloat_mult_Get_b(&XFT);
    printf("  Get value a = %f(%x), b = %f(%x)\n",
            in_a.f,in_a.u,in_b.f,in_b.u);
    XFloat_mult_Start(&XFT);          // Start HW
    while (!XFloat_mult_IsDone(&XFT)) ;  // Wait Done signal

    out_c.u = (unsigned int)XFloat_mult_Get_c(&XFT); // Load register c
    cycle = XFloat_mult_Get_return(&XFT);

    printf("%d:  Get value c = %f(%x)\n", cycle, out_c.f,out_c.u);

    if ( ref_float_mult(ref_a.f, ref_b.f) != out_c.f )  // Comparison
        printf("  ERROR! ref = %f, val = %f\n",
                ref_float_mult(ref_a.f, ref_b.f),out_c.f);
}

printf("float_mult test done.\n");
cleanup_platform();
return 0;
}
```

Execution Results

Summary

- HLS brings value to the FPGA design:
  - Make hardware design easier for hardware engineers
  - Allow software engineers to design hardware and reap energy/performance benefits
- Benefits: Productivity, lower non-recurring engineering costs, maintainability, faster time to market
- Execute a tutorial through a floating point precision multiplication

Exercise

- (Mandatory) Describe the advantage and the disadvantage for the high-level synthesis based design comparing with the RTL based design

- (Mandatory) Execute a tutorial for a floating point precision multiplication on your Zybo board

- (Optional) Compare with non "#pragma HLS pipeline" version, and report differences of pipelined one

Send a report to OCW-i

Deadline is 28th, July, 2020 JST PM 13:20

(At the beginning of the next lecture)