Reconfigurable Architecture (9) High-level synthesis (1 of 2)

Today

- * HLS: High-Level Synthesis technology

 - OpenCL (mostly used in GPUs) is also supported *

Generate RTL from C, C++, Java, System C or other languages

HLS tools

- Without particular targets
 - Impulse C (Impulse), Cyber WorkBench (NEC) *
- For specific device technology
 - Vivado HLS (Xilinx)
- For specific platform
 - Carte (SRC: C/Fortran), MaxCompiler (Maxeler: Java)



Better in HLS:

- Implementing complex algorithms: hard to implement with HDLs * Describe and verify as a software
- - Debug without HDL simulator but with compilers: very fast *
 - Easy to modify / update algorithm
- Easy to try design trade-offs: i.e, area vs speed



Popular HLS targets

- Financial applications: stock and FX predictions
- Signal processing: Digital filter implementations in HLS
 - Also effective for image processing, especially real-time ones
 - Matlab is also a popular design entry



HLS is not a magic bullet

- Not always fast, may result in a (very) large circuit
 - Tuning is crucial for HLS designs
- Not good at clock-accurate stuff as HDL
 - HDL is better for external interface logic
 - HLS is suitable for internal, complicated algorithms



How HLS tools work

- Extract control flow
 - Branches and loops
- Then, extract data flow
 - For each BBs (basic blocks)
 - Perform schedule and bind

Xilinx UG902 (v2014.1)



Data flow extraction

Xilinx UG902 (v2014.1)





Scheduling and binding

Scheduling

а

b

С

Initial Binding

Target Binding







Interface synthesis

- Scalar arguments:
 synthesized as I/O ports
- Array arguments: basically synthesized as BlockRAM I/Fs
 - Communication via RAMs
- Details later

Xilinx UG902 (v2014.1)

Loop optimizations

- Separate constant value calculations
 - Do only once before the loop begins

```
int x, y;
for(int i = 0; i < 3; i++) {
    x = in[i];
    y = a*x + b + c;
    out[i] = y;
}
```

Xilinx UG902 (v2014.1)

Iteration 1





Iteration 2 and later



Control State Control FSM is automatically generated State

b+c i++ 0 in_addr in[0] in_data_out x reg $a^{*}x + (b+c)$ out_data_we



Control flow and FSM

- Control flow = FSM states
 - I/O and branches changes state of the FSM
 - In other words, data path is switched by the FSM
- FSM state may not change every clock cycle: FSM may stay in a specific state (i.e., "Idle" state) for a long time

Interface synthesis

- * C functions \rightarrow How about the module ports in synthesized RTL?
 - In C, functions have 3 ways to perform I/O
 - Function arguments
 - Return valves

SI/OS: printf(), gettimeofday() and others (not synthesizable)

Arguments, Return values and pointers

- Arguments are basically input
 - Pointer arguments can be output
- Return values are always output



Basic module interface







BRAM I/F example

- BRAM I/F synthesized for array argument
 - BRAM is not included internally
 - Write before ap_start,
 Read after ap_done
- BRAMs are placed externally, this gives a great flexibility





More on Interface Synthesis

- Variety of interfaces are supported to meet various demands
 - Synthesized along coding style and directives
 - Directives given by #pragma or separate directive files
 - Refer tool's documentation: User Guide 902 for Vivado HLS



Optimization

- In RTL design, resulting hardware has less variations
 - Because RTL description gives a rigid, clock-accurate model
- * RTL generation in HDL has much more design options
 - Many possible designs from the same source code
 - This is good for fine-tuning of the design, but there's no "oneclick solution"

Performance metrics

- 2 major concerns: area and speed
 - * Area: How many LUTs, FFs and other blocks are used
 - * Latency: clock cycles required between input and output
 - Interval: clock cycles required between input and next input

Loop Pipelining

- * (2)(3)(4) in the previous example
 - * On ③, ② for next iteration is possible
- No change in latency
 - Throughput improved
- Effective with longer latency







Duplication

- BRAMs are dual-ported
 - 2x pipelines gives 2x performance
 - Of course, 2x area is required







Source of difficulties

- Dependencies between loop iterations
 - * This is really problematic: but appears in today's hands-on…
- Non-constant number of loop iterations
- Without these problems, loop are pipelineable easily



Optimization directive: pragma

pragma in source files

- Safely ignored by
 (software) compilers
- Good for making many small tests

```
int x, y;
for (int i=0; i<3; i++){
#pragma HLS PIPELINE
    x = in[i];
    y = a*x + b + c;
    out[i] = y;
  }
}</pre>
```



Optimization directive: directive file

Using separate directive file

- Only labels in source code
- Multiple directive files for
 multiple optimization
 strategies is possible

```
add_loop:
    for (int i=0; i<3; i++){
        x = in[i];
        y = a*x + b + c;
        out[i] = y;
    }
}
```

set_directive_resource -core RAM_1P "foo" in set_directive_pipeline "foo/add_loop"



C programming for HLS (1)

- Vivado HLS supports "normal" ANSI C and C++
 - * Under several restriction, most syntax can be synthesized
 - ★ Exceptions: pointers, recursions, memory allocations, OS I/Os…
 - * main() for testbench, some function will be the "top-level module"
 - Everything are allowed in testbench



C programming for HLS (2)

- Still understandable / executable as a software
 - But it's "hardware description"
 - Fine-tuned software will be very insufficient with HLS: different optimization is required for HLS design



Example: Least Common Multiple

 Very simple algorithm to calculate LCM

With C test bench

```
int lcm(int a, int b){
  int x = a*b;
 // make a > b
  if (a<b){
    int tmp=a;
    a = b;
    b = tmp;
  int r = a%b;
  while (r!=0){
    a=b;
    b=r;
    r=a%b;
  return x/b;
ר
```



C Testbench

Just call the lcm() function

Show the result by printf()

```
#include <stdio.h>
int lcm(int, int);
int main(){
    printf("2,3 %d\n", lcm(2,3));
    printf("5,12 %d\n", lcm(5,12));
```

}



Vivado HLS design flow

- Write source code + testbench
 - Just compile and run to test as a software
- * C Synthesis with Vivado HLS
 - * RTL Co-Simulation to verify with C TB + Synthesized RTL
 - Try optimize along C Synthesis reports
- * RTL Export \rightarrow import to RTL design as an IP core



Create Vivado HLS project

- Launch Vivado HLS
 - Not "Vivado" for HDL design
 - Vivado HLS and Vivado is different, of course separate project folders are required





Specify project name+location

"Location/project name" folder will be created

Project Con	figuration	New Vivado HLS F	Project	A C
Create Vivado	o HLS project of se	lected type		
<u>P</u> roject name	e: [Icm]			
Location: //h	ome/yasu/mac/wo	ork/reconf-class/	hls18	B <u>r</u> owse



Add design file

- "Add Files" then choose the design file (lcm.c)
- Then set top function by using "Browse" button

• • •	New Vivado HLS Project	t
Add/Remove File Add/remove C-ba	e s Ised source files (design specificat	tion)
Top Function:	n	Browse
Name	CFLAGS	Add Files
🗎 lcm.c		New File
		Edit CFLAGS
		Remove
	< <u>B</u> ack <u>N</u> ext >	Cancel <u>Einish</u>



Add testbench

Just "Add Files" then choose lcm-tb.c

• • •	X New Vivado HLS Project		
Add/Remove Files Add/remove C-based tes	stbench files (design test)		*
TestBench Files			
Name	CFLAGS		Add Files
📄 lcm-tb.c			New File
		-	Add Folder
		E	dit CFLAGS
			Remove
	1 12		
<u> </u>	ck <u>N</u> ext > (ancel	Einish



Set Solution & Select device

- Default clock period is 10ns
 - No problem for Nexys4
 - Details for solution later
- Device is the usual one:

* xc7a100tcsg324-1

😑 💿 💿 📉 New Vivado	HLS Project		n nennna opr						
Solution Configuration Create Vivado HLS solution for selected te	chnology								
Solution Name: solution1 Clock Period: 10 Uncerta	ainty:								
Part Selection									
Part: xc7a100tcsg324-1									
					wice Sele	ection D	lialog		
	Select: N Parts E	Boards				•	Package:	All	2
	Family: Al	1				-	Speed grade	e: All	
	Sub-Family: Al	I				•	Temp grade	: All	1
				F	Reset Al	l Filters	:		
< Back	Search: 🔻 xc7a100to	sg							
	Part	Family	Package	Speed	SLICE	LUT	FF	DSP	BRAM
	🔷 xc7a100tcsg324-3	artix7	csg324	-3	15850	6340	0 126800	240	270
	♦ xc7a100tcsg324-21	artix7	csg324	-21	15850	6340	0 126800	240	270
	xc7a100tcsg324-2	artix7	csg324	-2	15850	6340	0 126800	240	270
	🔷 xc7a100tcsg324-1	artix7	csg324	-1	15850	6340	0 126800	240	270



What's the "Solution?"

- * A optimization strategy set including:
 - * Target clock frequency
 - Target device
 - Optimization directives in directive file (not by #pragma)
- Multiple solutions in single project is possible!



C simulation

- Simulate as a software
 - * Project \rightarrow Run C simulation
 - Result will appear on the console





C synthesis

- Synthesis is done per solution basis
 - RTLs are generated

<u>Solution</u> <u>Window</u> <u>Help</u>	
Solution Settings	
Run C Synthesis	Active Solution
Run C/RTL Cosimulation	All Solutions
Export RTL	Select Solutions
& Open Analysis Perspective	
Dpen Wave Viewer	



Synthesis report

- Things to be certain:
 - Latency/Interval
 - Interface
- Circuit size (Utilization) is undetermined until mapped in Vivado

Latency (clock cycles)

Summary

Late	ency	Inte		
min	max	min	max	Туре
?	?	?	?	none

Detail

E Loop

	Late	ency	2	Initiation I	nterval	
Loop Name	min	max	Iteration Latency	achieved	target	-
- Loop 1	?	?	36	-	-	

Utilization Estimates

-	S	u	n	ır	n	а	ny	
---	---	---	---	----	---	---	----	--

Name	BRAM 18K	DSP48E	FF	IUT
DSP	-	-	-	-
Expression	-	3	0	121
FIFO	-	-	-	-
Istance	-	-	1182	714
Memory	-	-	-	-
Multiplexer	-	-		491
Register	-	-	235	-
Total	0	3	1417	1326
Available	270	240	126800	63400
Utilization (%)	0	1	1	2

100	+ -		-	~	~
	UC	11	d	C	e

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	С Туре
ap_clk	in	1	ap_ctrl_hs	lcm	return value
ap_rst	in	1	ap_ctrl_hs	lcm	return value
ap_start	in	1	ap_ctrl_hs	lcm	return value
ap_done	out	1	ap_ctrl_hs	lcm	return value
ap_idle	out	1	ap_ctrl_hs	lcm	return value
ap_ready	out	1	ap_ctrl_hs	lcm	return value
ap_return	out	32	ap_ctrl_hs	lcm	return value
а	in	32	ap_none	а	scalar
b	in	32	ap_none	b	scalar

Ele Edit Project Solution Window Help Image: Solution Help		📉 Vivado HLS 2018.2 - Icm (/home/yasu/mac/work/reconf-class/hIs18/k	cm)
Image: Second	t <u>P</u> roject <u>S</u> olution <u>W</u> indow <u>H</u> el		
Explorer X Performance Estimates Source Includes Includ	0 0 × 0 0 0 0 0 0	🕸 🐿 🧯 🚍 📮 🔸 🗴 🖶 👘 👘 🍪	ॐ Debug 🄁 Synthesis & Analysis
Sincludes Register Source Includes Source Interface Summary Test Bench ap_ctk in 1ap_ctrl hs Icm return value ap_rest in 1ap_ctrl hs Icm return value ap_idle out 1ap_ctrl hs Icm return value ap_i	rer 🕱 🥜 🗖 🗊	Synthesis(solution1) 🕱	😑 🗈 Outline 🛛 🖉 Directive 😐 🗖
 Source Interface Summary Test Bench ap_cfk in 1 ap_ctrl hs lcm return value ap_rst in 1 ap_ctrl hs lcm return value ap_ack in 1 ap_ctrl hs lcm return value ap_ack in 1 ap_ctrl hs lcm return value ap_return out 32 ap_cnone b in 32 ap_none b scalar Export the report(.html) using the Export Wizard Open Analysis Perspective Vivado HLS Console INPO: [SYSC 207-301] Generating SystemC RTL for lcm. INPO: [SYSC 207-301] Generating SystemC RTL for lcm. INPO: [SYSC 207-301] Generating Verilog RTL for lcm. INPO: [VLOG 209-307] Generating Verilog RTL for lcm. INPO: [HLS 200-112] Total elapsed time: 37.09 seconds; peak allocated memory: 62.807 MB. Finished C synthesis.	ncludes	Register	General Information
Image: Second State Image: State Protocol Source Object C Type Image: State S	Source In Lon.c	erface Summary	 ▼ E Performance Estimates ™ Timing (ns) ™ Latanay (clock cycles)
Console 22 O'Errors & Warnings 12 DRCs Vivado HLS Console INFO: [SYSC 207-301] Generating SystemC RTL for lcm. INFO: [VHDL 208-304] Generating VHDL RTL for lcm. INFO: [VHL 208-304] Generating Verilog RTL for lcm. INFO: [HLS 200-112] Total elapsed time: 37.09 seconds; peak allocated memory: 62.807 MB. Finished C synthesis.	est Bench e solution1 e Ex Ex Op	TL Ports Dir Bits Protocol Source Object C Type 0_clk in 1 ap_ctrl hs Icm return value 0_rst in 1 ap_ctrl hs Icm return value 0_start in 1 ap_ctrl hs Icm return value 0_done out 1 ap_ctrl hs Icm return value 0_done out 1 ap_ctrl hs Icm return value 0_done out 1 ap_ctrl hs Icm return value 0_return out 1 ap_none a scalar in 32 ap_none b scalar ort the Export Mizard en Analy	 ■ Latency (clock cycles) ▼ E) Utilization Estimates ■ Summary © Detail ▼ E Interface ■ Summary
Vivado HLS Console INFO: [SYSC 207-301] Generating SystemC RTL for lcm. INFO: [VHDL 208-304] Generating VHDL RTL for lcm. INFO: [VLOG 209-307] Generating Verilog RTL for lcm. INFO: [HLS 200-112] Total elapsed time: 37.09 seconds; peak allocated memory: 62.807 MB. Finished C synthesis.	•	console 🛿 🥑 Errors à Warnings 🖆 DRCs	
	Viv IN IN IN IN Fi	do HLS Console 0: [SYSC 207-301] Generating SystemC RTL for lcm. 0: [VHDL 208-304] Generating VHDL RTL for lcm. 0: [VLOG 209-307] Generating Verilog RTL for lcm. 0: [HLS 200-112] Total elapsed time: 37.09 seconds; peak ished C synthesis.	allocated memory: 62.807 MB.





Reading the report

- * Latency and Interval is "?"

 - loop
 - then the latency is estimated in the report *

* Because there's a "while" loop, the # iterations undetermined

Insert "#pragma HLS LOOP_TRIPCOUNT avg=10" in the while

```
int r = a%b;
 while (r!=0){
#pragma HLS LOOP TRIPCOUNT avg=10
    a=b;
```



Launch Co-simulation

- C TB + Synthesized RTL
 - Set "Dump trace" to "All"
 - C TB is translated to HDL, results are compared

Solution Window Help	Co-simulation Dialog
Solution Settings	C/RTL Co-simulation
Run C/RTL Cosimulation Export RTL Run C/RTL Cosi Open Report Open Wave Viewer	Verilog/VHDL Simulator Selection Auto RTL Selection Options Setup Only Dump Tracenone Dump Tracenone Optimizing Compile Reduce Diskspace Wave Debug Compiled Library Location Input Arguments Do not show this dialog box again. Cancel

Co-simulation result

- * printf() messages in Console
- Launch Waveform and see the signals
 - C Inputs/Outputs
 - Block Level IO Handshake: ap_{start,done,...}







									2,69
	Value	0 nș	500 ns	1,000 ns	, .	1,500 ns	.	2,000 ns	
ign Top Signals									
C Outputs									
📹 return(wire)									
> 10 ap_return[31:0]	000003c	20000000			X 0000006				
C Inputs									
ங b(wire)									
> 👹 b[31:0]	000000c	X 00000003			X 000000c				
👅 a(wire)									
> 👹 a[31:0]	0000005	κχ	0000002		x		0000000	15	
Block-level 10 Handshake									
🕌 ap_start	1								
🕌 ap_done	0				1				
🕌 ap_idle	0								
₩ ap_ready	0								





Adding Directives

- Directives on functions,
 variables and loops
- In source file (#pragma) or
 in directive file (per solution)







Example: interface directive

- Set the lcm() function's "INTERFACE" to "s axilite"
 - And also on argument a and b
 - Synthesized interface is gathered into single AXI slave interface



Analysis View

- Control state and dependency graph
 - Can refer source code



