CO A Toolkit for Designing Hardware DSLs

Griffin Berlstein, Rachit Nigam, Chris Gyurgyik, Adrian Sampson

Computer Architecture and Programming Abstractions Group





Designing hardware accelerators is *challenging*, particularly for *domain experts* from other fields



Lack of a *robust compiler infrastructure* hinders DSL development

Limited debugging tools hinder DSL adoption





High-level control flow





Image by Eucalyp





 $div = std_div(32);$

Cal Х if 🔴 { while 🔴 { } else { par { ; control

r1 = std_reg(32); mul = std_mul(32); r2 = std_reg(32); add = std_add(32); div = std_div(32);

cells





```
r1 = std_reg(32);
mul = std_mul(32);
r2 = std_reg(32);
```

```
add = std_add(32);
```

```
div = std_div(32);
```

cells

```
group pink {
   mul.left = reg.out;
   mul.right = mem.out;
   mem.in = mul.out;
}
```

```
group orange {
   mem.in = add.out;
   add.left = mem.out;
   add.right = mem.out;
   ...
```

```
group green {
```

.....

wires

if pink { while orange { green; } } else { par { orange; green;

control

r1 = std_reg(32); mul = std_mul(32); r2 = std_reg(32); add = std_add(32); div = std_div(32);

cells

```
group pink {
   mul.left = reg.out;
   mul.right = mem.out;
   mem.in = mul.out;
}
```

```
group orange {
   mem.in = add.out;
   add.left = mem.out;
   add.right = mem.out;
   ...
```

```
group green {
```

.....

wires

Optimizations in Calyx



Debugging with COLX

Debugging in Software



Breakpoints

Step execution

Program.cs 😐 🗙				▼ Solu				
💷 DebuggingDemo	👻 🔩 Debuggi	ngDemo.Program	🝷 🎯 Main(string[] arg	s) Tion				
10 { 11 12	public static string GetN	+ A						
13 🖻 14 15 16	<pre>public static string GetWord() { return "hello"; }</pre>							
17 18 ⊡ 19	<pre>public static void Main(string[] args) args = {string[0]} {</pre>							
20 21 22 23	<pre>var name = GetName(); name = {null} var word = GetWord(); var result = \$"{name} says {word}!";</pre>							
24 25 100 % • •	Console.WriteLine(result);							
Locals		••••••••••••••••••••••••••••••••••••••	Call Stack	– ¶ ×				
Name	Value	Туре	Name	Lang				
argsname	{string[0]}	string[] string	DebuggingDemo.exe!	DebuggingDem(C#				
✓ word✓ result	null	string						
Autos Locals Watch	1	stillig	> 50 \Xi 🍙	° □ □ →				
🗁 Ready			↑ Add to So	urce Control 🔺 🏼 🍺 🔵 📰				

Complex state inspection

Debugging in Hardware

	100ns	200ns	300ns	400ns	500ns	600ns	700ns	800ns
ل	mmm	MMM	mmm	www	www	www	MMM	MM
X						2684354	156	
X						2684354	156	
X						00		
		_						
				<u></u>				0
=				\sim				
				1				
X		0	2684 X					0
X		0	<u> </u>					0
		0						0

Debugging in Hardware

RTL doesn't know about the *control* of the accelerator

- Any assignment could change at each clock cycle
- So, debugging tools must capture and display every change

Even simple accelerators can have many thousands of cycles!

Even worse: *generated* RTL does not resemble the surface application

We cannot expect domain experts to code in a DSL and debug in RTL

RTL is too low-level for DSL *accelerator* debugging

Instead debug at the level of the Calyx IR



High-level control flow



Low-level structure

Calyx knows about control flow explicitly. This bypasses many of the challenges of RTL simulation

Image by Eucalyp





Calyx represents many of the realities of hardware which are not visible when debugging at the DSL level (or C for HLS)

Image by Eucalyp

Calyx Interactive Debugger (CIDR)

Built on top of the Calyx Interpreter

interp — interp • fud e --to debugger tests/complex/unsigned-dot-product.futil -s verilog.data tests/comp... [[~/research/capra/calyx/interp]\$ fud e --to debugger tests/complex/unsigned-dot-product.futil -s verilog.data] tests/complex/unsigned-dot-product.futil.data == Calyx Interactive Debugger == > break unknown There is no group named: unknown > break initialize_mem_0 > break initialize_mem_1 > info break Current breakpoints: 1. enabled initialize_mem_0 2. enabled initialize_mem 1 >

```
interp — interp < fud e --to debugger tests/complex/unsig...
                                                                  🖲 😑 🛑 interp — interp 🛛 fud e --to debugger tests/complex/unsig...
== Calyx Interactive Debugger ==
> info break
   Current breakpoints:
                                                                  > info break
   1. enabled initialize_mem_0
                                                                    Current breakpoints:
   2. enabled initialize_mem_1
                                                      Interp
                                                                    1. disabled initialize_mem_0
> step
                                                                     2. disabled initialize_mem_1
> continue
                                                                  > break add
[Hit breakpoint: initialize_mem_0
                                                                  > break mul
> print mem0
                                                                  > continue
                                                      x/unsigned-dot-
   mem0
                                                                 Hit breakpoint: add
     addr0 = [000]
                                                      ts/complex/unsi[ > step
    addr1 = [000]
                                                                  > continue
    write_data = [00000000000000000000000000000000]
                                                                 Hit breakpoint: add
    write_en = [0]
                                                                  > step
    clk = [0]
                                                                  > step
    > continue
    done = [0]
                                                                 Hit breakpoint: mul
> disable 1
                                                                  > step
> continue
                                                                  > continue
Hit breakpoint: initialize_mem_1
                                                                 Hit breakpoint: mul
                                                                  >
> print mem0
   mem0
    addr0 = [000]
    addr1 = [000]
    write_data = [00000000000000000000000000000000]
    write_en = [0]
    clk = [0]
    done = [0]
> print mem0.read_data
   > print mem1.read_data
   > s
> print mem1.read_data
   >
```

Future extensions to **CIDR**

- Data structure rematerialization for frontends
- Source Location support and gui interaction
- IR level debugging constructs/annotations
 - Watch values & asserts
- FPGA integration (capture error state from FPGA to debug in software)
- And much more in the broader Calyx ecosystem



High-level control flow





Image by Eucalyp



Thanks to my co-authors and the rest of the Calyx team!

