A Flexible Hardware ECDLP Engine in Bluespec

Lyndon Judge & Patrick Schaumont

Center for Embedded Systems for Critical Applications (CESCA)
Bradley Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA 24061, USA

Presented by: Lyndon Judge

Prior Work: ECDLP Systems

Platform	Curve	Publication Year	Point Additions per Second
Spartan-3 FPGA	130 bit, binary field	2010	111M
Nvidia GTX 295 GPU	130 bit, binary field	2010	63M
Spartan-3E FPGA	130 bit, binary field	2009	33.67M
Cell Processor	130 bit, binary field	2009	27M
Nvidia GTZ295 GPU	130 bit, binary field	2009	12.56M
Spartan-3E FPGA	130 bit, binary field	2007	10M
Cell Processor	112 bit, prime field	2010	8.8M
Cell Processor	112 bit, prime field	2009	7M
Cell Processor	130 bit, binary field	2010	4.3M
Virtex 5 FPGA	112 bit, prime field	2011	660K
Spartan-3 FPGA	128 bit, prime field	2007	57.8K

Motivation

Hardware can outperform software, but most recent works are still based on software. Why?

How can Bluespec be used to implement flexible hardware designs?

Objective

- Investigate the Bluespec hardware design methodology
- Demonstrate design space exploration to determine optimal parameters for hardware ECDLP

Bluespec

What is Bluespec?

- Hardware synthesis toolset that claims faster behavioral hardware design at a higher abstraction level than HDLs
- Provides language and tools for hardware system design, synthesis, modeling, and verification
- Bluespec System Verilog: high abstraction level HDL

Modules & Interfaces

- Modules consist of
  - Datapath : Instantiate lower level components
  - Behavior : Control data manipulation via rules & methods
  - Interface : Encapsulate IO with methods

- Interface Methods
  - Describe IO behavior and enforce conditions for the behavior
  - Arguments = input ports
  - Return values= output ports

Control

- Expressed using atomic rules
- Rules consist of action(s) guarded by Boolean condition
- Atomicity means:
  - All actions either execute in parallel or do not execute
  - Rules are defined independently of each other

Scheduling

- Bluespec compiler derives execution schedule based on *implicit* and *explicit* conditions of rules and methods

- Conflicting rules scheduled based on priority specified by designer or assigned by compiler

Designing Modular Multiplier in Bluespec

Background:

Modular Arithmetic

- Target curve secp112r1 in $GF\left(\frac{2^{128}-3}{76439}\right)$
- Represent integers in

$$R = {}^{\mathbb{Z}}/_{q\mathbb{Z}},\ q = 76439 * p$$

- Perform arithmetic modulo $2^{128} - 3$ for fast reduction

Modular Multiplication Algorithm

- Algorithm & architecture previously presented in:

  S. Mane, L. Judge, P. Schaumont, "An Integrated Prime-field ECDLP Hardware Accelerator with High-performance Modular Arithmetic Units," 2011 International Conference on Reconfigurable Computing and FPGAs (RECONFIG), December 2011.

- Uses ideas from

  D. J. Bernstein, T. Lange, P. Schwabe. "On the correct use of the negation map in the Pollard Rho method," IACR ePrint (2011), http://eprint.iacr.org/2011/003.pdf.

  T. Guneysu, C. Paar. "Ultra high performance ECC over NIST primes on commercial FPGAs," CHES 2008, LNCS 5154, pp. 62-78, 2008.

Modular Multiplication Algorithm

$$
\begin{array}{cccccccc}
f_7 & f_6 & f_5 & f_4 & f_3 & f_2 & f_1 & f_0 \\
g_7 & g_6 & g_5 & g_4 & g_3 & g_2 & g_1 & g_0 \\
\hline
\end{array}
$$

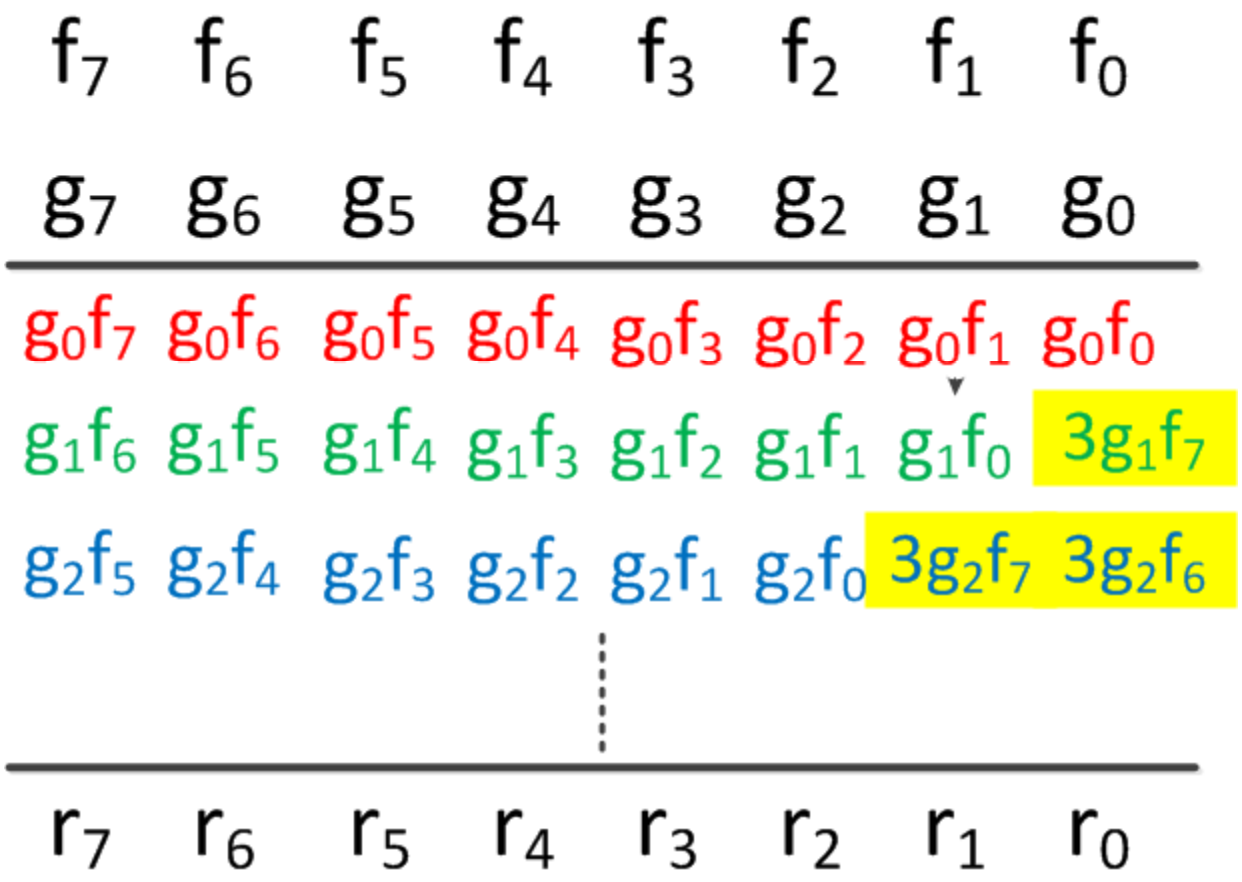
Modular Multiplication Algorithm

$$
\begin{array}{cccccccc}
f_7 & f_6 & f_5 & f_4 & f_3 & f_2 & f_1 & f_0 \\
g_7 & g_6 & g_5 & g_4 & g_3 & g_2 & g_1 & g_0 \\
\hline
g_0f_7 & g_0f_6 & g_0f_5 & g_0f_4 & g_0f_3 & g_0f_2 & g_0f_1 & g_0f_0
\end{array}
$$

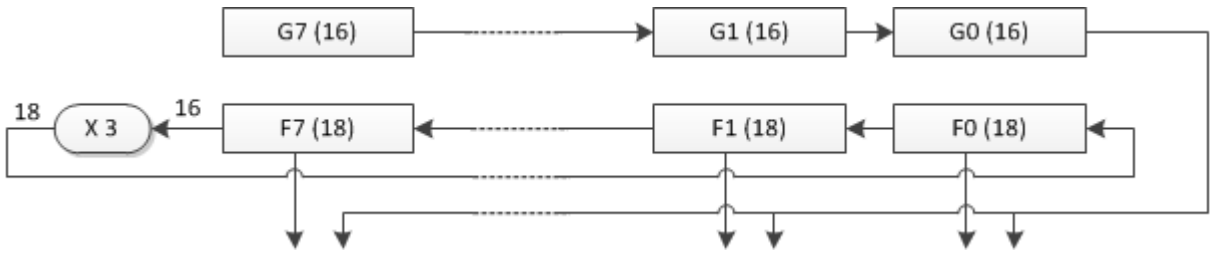
Modular Multiplication Algorithm

$$
\begin{array}{cccccccc}
 & f_7 & f_6 & f_5 & f_4 & f_3 & f_2 & f_1 & f_0 \\
 & g_7 & g_6 & g_5 & g_4 & g_3 & g_2 & g_1 & g_0 \\
\hline
 & g_0f_7 & g_0f_6 & g_0f_5 & g_0f_4 & g_0f_3 & g_0f_2 & g_0f_1 & g_0f_0 \\
 & g_1f_7 & g_1f_6 & g_1f_5 & g_1f_4 & g_1f_3 & g_1f_2 & g_1f_1 & g_1f_0 \\
g_2f_7 & g_2f_6 & g_2f_5 & g_2f_4 & g_2f_3 & g_2f_2 & g_2f_1 & g_2f_0 & \\
 & \vdots & & & & & & &
\end{array}
$$

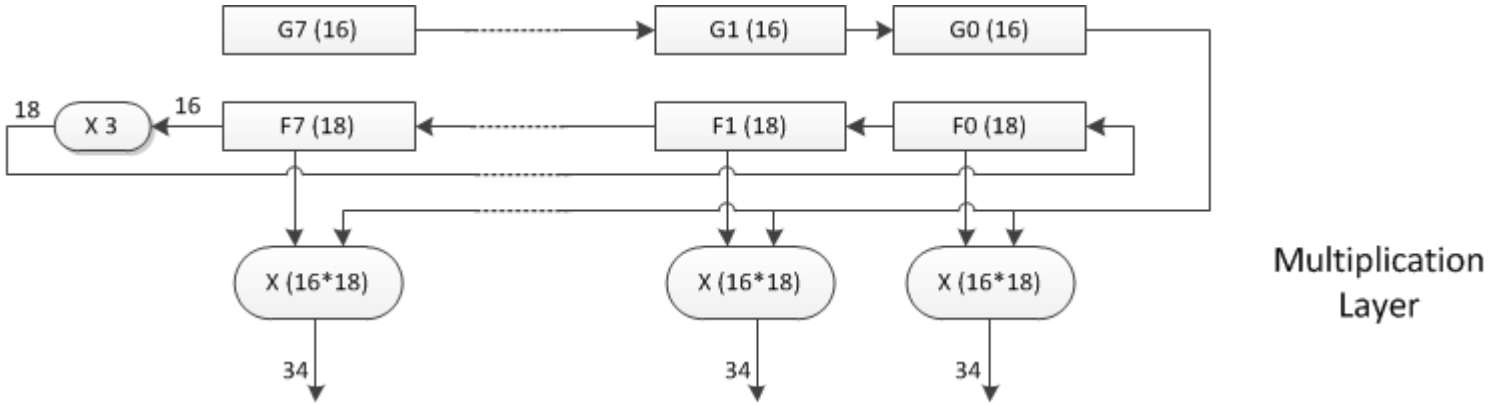
Modular Multiplication Algorithm



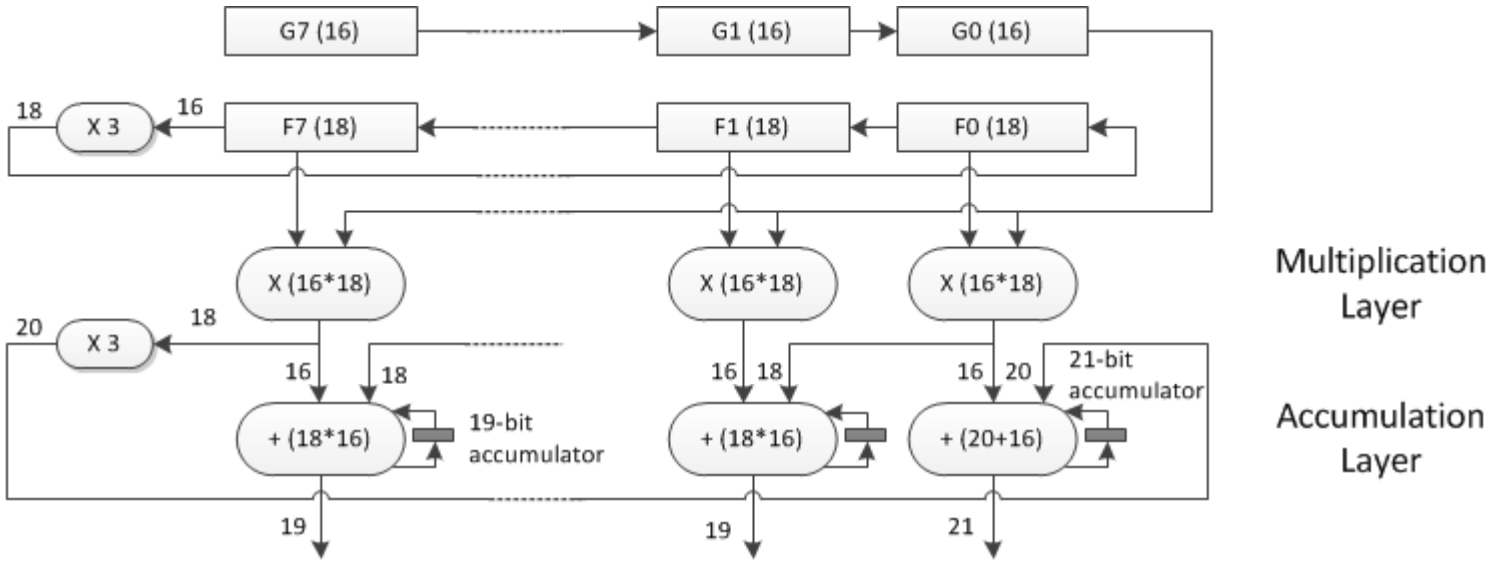
Modular Multiplier Architecture



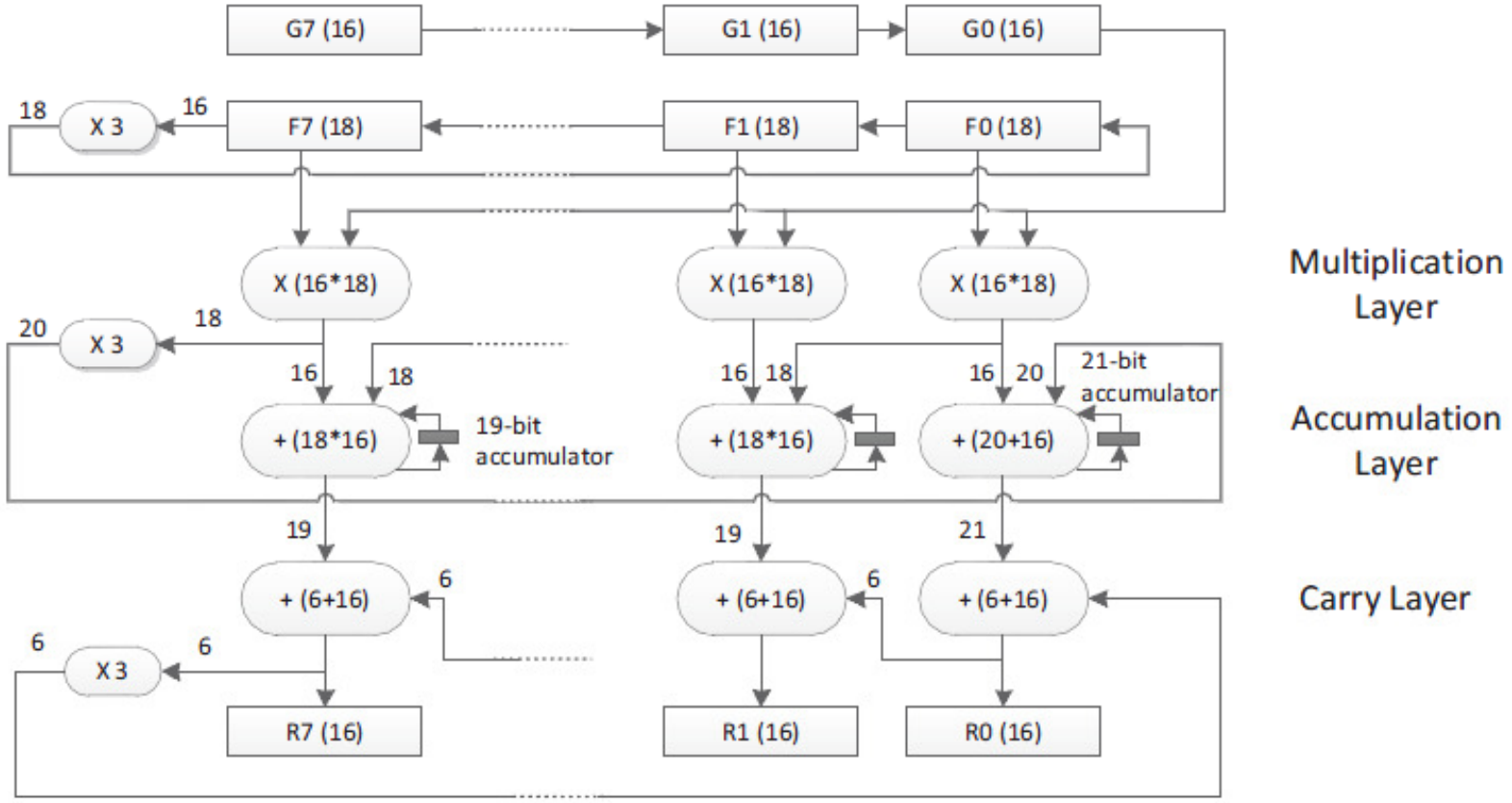
Modular Multiplier Architecture



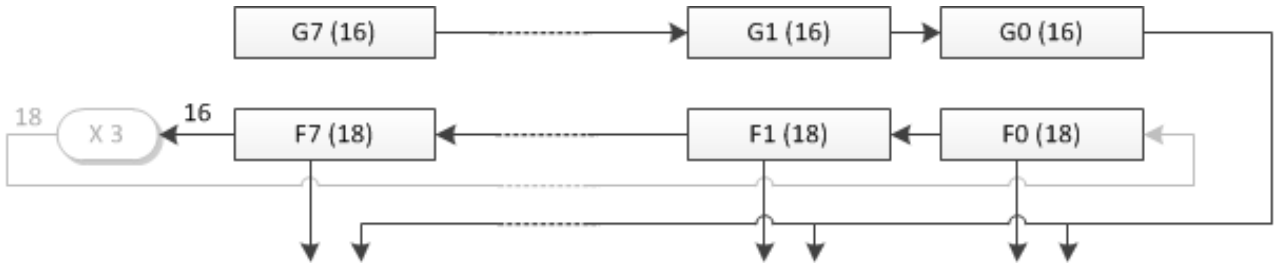
Modular Multiplier Architecture



Modular Multiplier Architecture

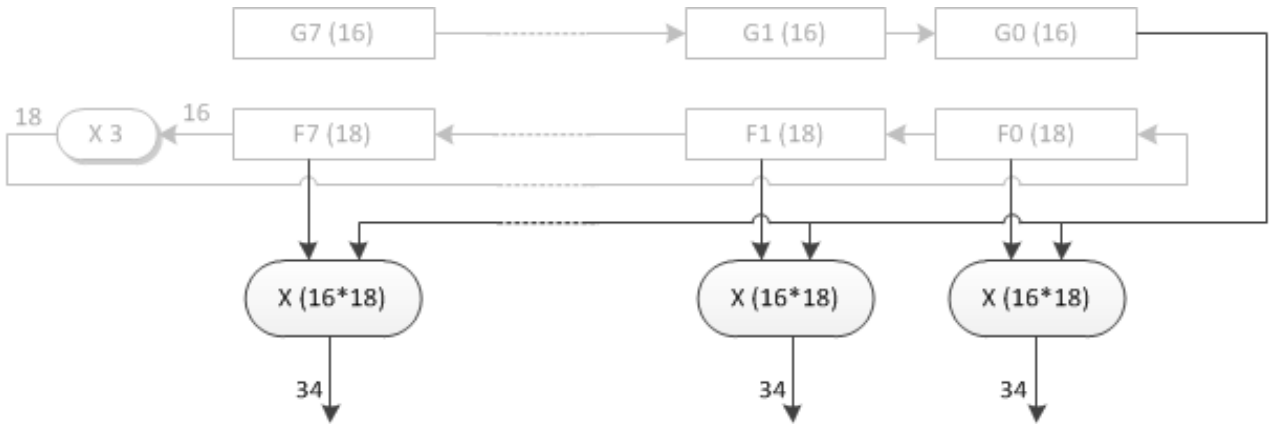


Bluespec Design: Datapath




```
//Operand storage registers
Reg#(Bit#(144)) f_in <- mkReg(0);
Reg#(Bit#(128)) g_in <- mkReg(0);
```

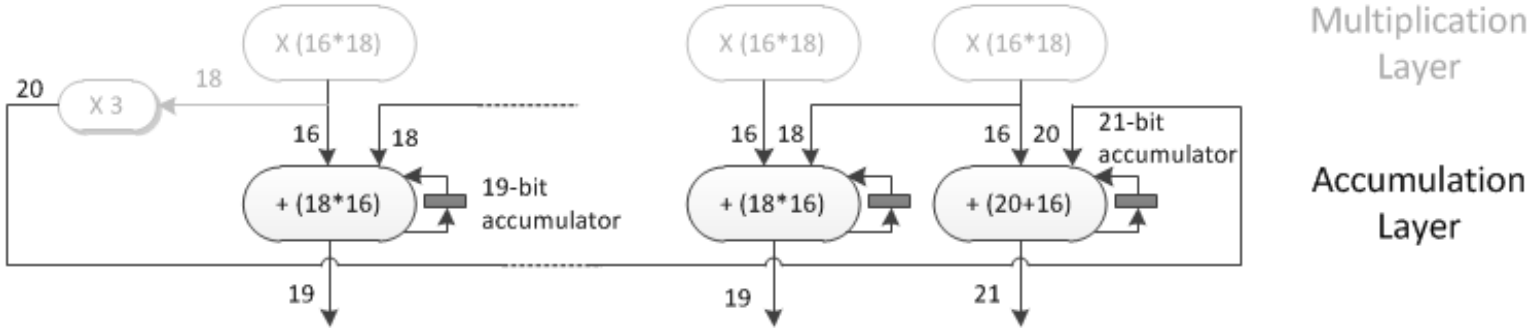
Bluespec Design: Datapath



Multiplication Layer

```
//18x16 single-cycle multipliers
Multiplier m0 <- mkMultiplier();
Multiplier m1 <- mkMultiplier();
...
Multiplier m7 <- mkMultiplier();
```

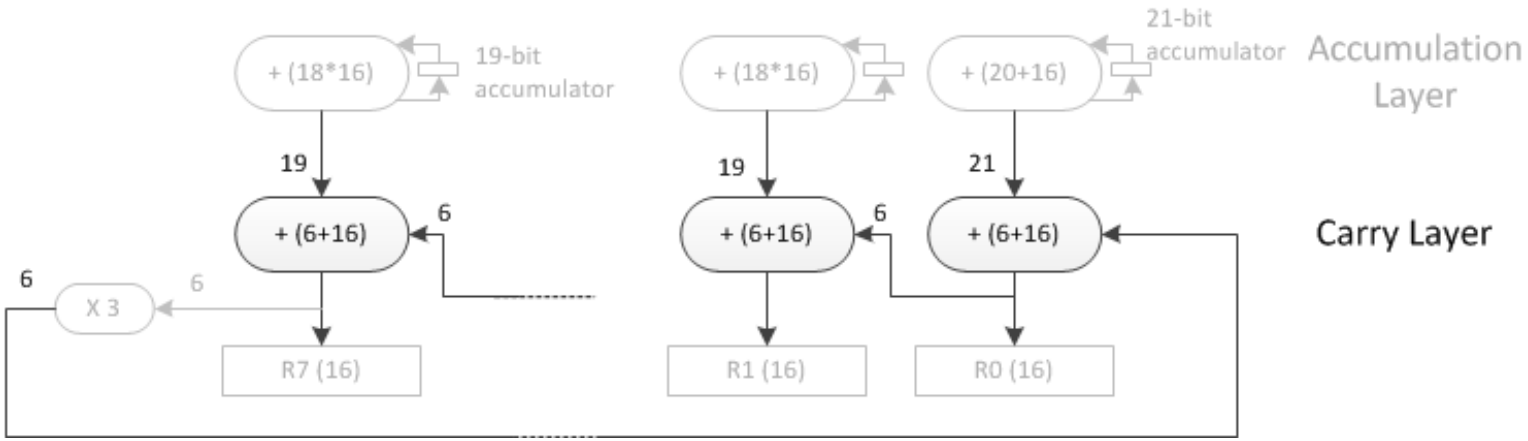
Bluespec Design: Datapath



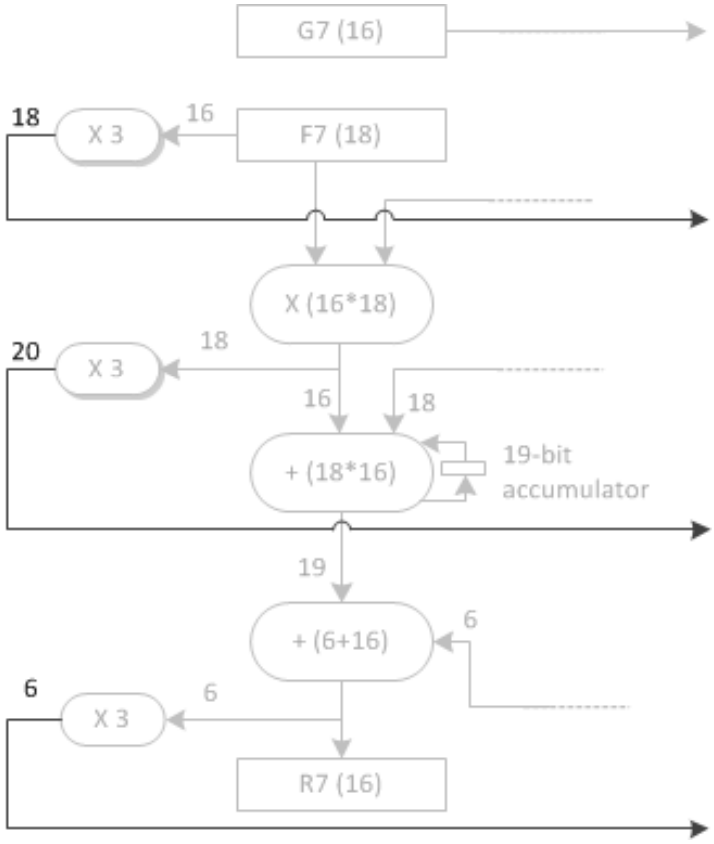

```
Accumulator a0 <- mkAccumulator();
Accumulator a1 <- mkAccumulator();
...
Accumulator a7 <- mkAccumulator();
```

Bluespec Design: Datapath

```
//Carry layer adders
Adder c0 <- mkAdder();
Adder c1 <- mkAdder();
...
Adder c7 <- mkAdder();
```



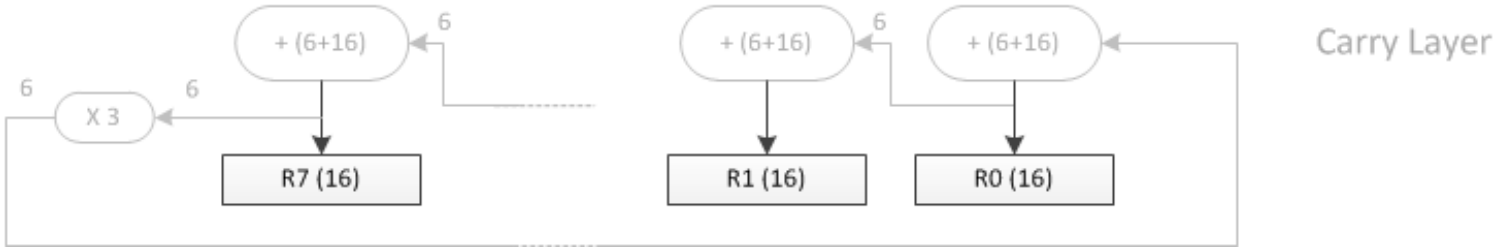
Bluespec Design: Datapath




```
//Reduction wrap-around values
Wire#(Bit#(34)) wrap_f <- mkWire();
Wire#(Bit#(34)) wrap_mult <- mkWire();
Wire#(Bit#(34)) wrap_carry <- mkWire();
```

Bluespec Design: Datapath

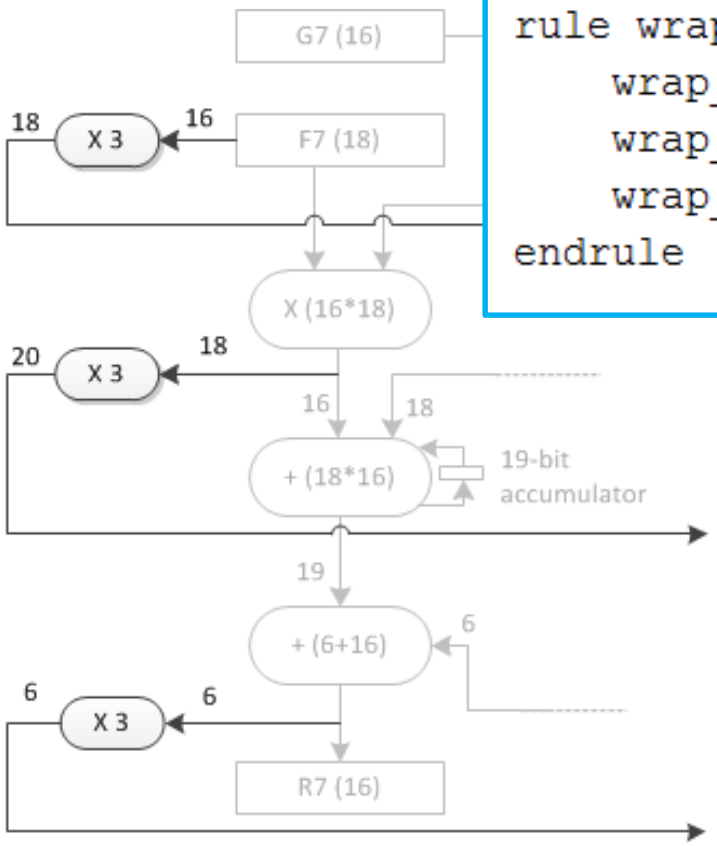
```
//4-bit counter value
Reg#(Bit#(4)) rcnt <- mkReg(15);
//Carry layer output
Wire#(Bit#(128)) tmp_res <- mkWire;
//Output
Reg#(Maybe#(Bit#(128))) cout <- mkWire;
```



Bluespec Design: Load

```
method Action load(Bit#(128) f, Bit#(128) g);
    f_in <= {2'd0, f[127:112], 2'd0, f[111:96],
             2'd0, f[95:80], 2'd0, f[79:64],
             2'd0, f[63:48], 2'd0, f[47:32],
             2'd0, f[31:16], 2'd0, f[15:0]};
    g_in <= g;
    a0.reset();
    a1.reset();
    ...
    a7.reset();
    rcnt <= 0;
endmethod
```

Bluespec Design: Reduce




```
rule wrap;
    wrap_f <= 3 * {0, f_in[143:126]};
    wrap_mult <= 3 * {0, m7.result()[33:16]};
    wrap_carry <= 3 * {0, c7.result()[21:16]};
endrule
```

Bluespec Design: Multiply

```
rule do_mult;
    f_in <= {f_in[125:0], wrap_f[17:0]};
    g_in <= g_in >> 16;
    m0.load(f_in[17:0], g_in[15:0]);
    m1.load(f_in[35:18], g_in[15:0]);
    ...
    m7.load(f_in[143:126], g_in[15:0]);
endrule
```

Bluespec Design: Accumulate

```
rule do_acc;
    a0.add(wrap_mult[19:0], m0.result()[15:0]);
    a1.add({0, m0.result()[33:16]}, m1.result()[15:0]);
    ...
    a7.add({0, m6.result()[33:16]}, m7.result()[15:0]);
endrule
```

Bluespec Design: Carry

```
rule do_carry;
    c0.add(a0.result(), wrap_carry[5:0]);
    c1.add(a1.result(), c0.result()[21:16]);
    ...
    c7.add(a7.result(), c6.result()[21:16]);
    tmp_res <= {c7.result()[15:0], c6.result()[15:0],
                c5.result()[15:0], c4.result()[15:0],
                c3.result()[15:0], c2.result()[15:0],
                c1.result()[15:0], c0.result()[15:0]};
endrule
```

Bluespec Design: Result

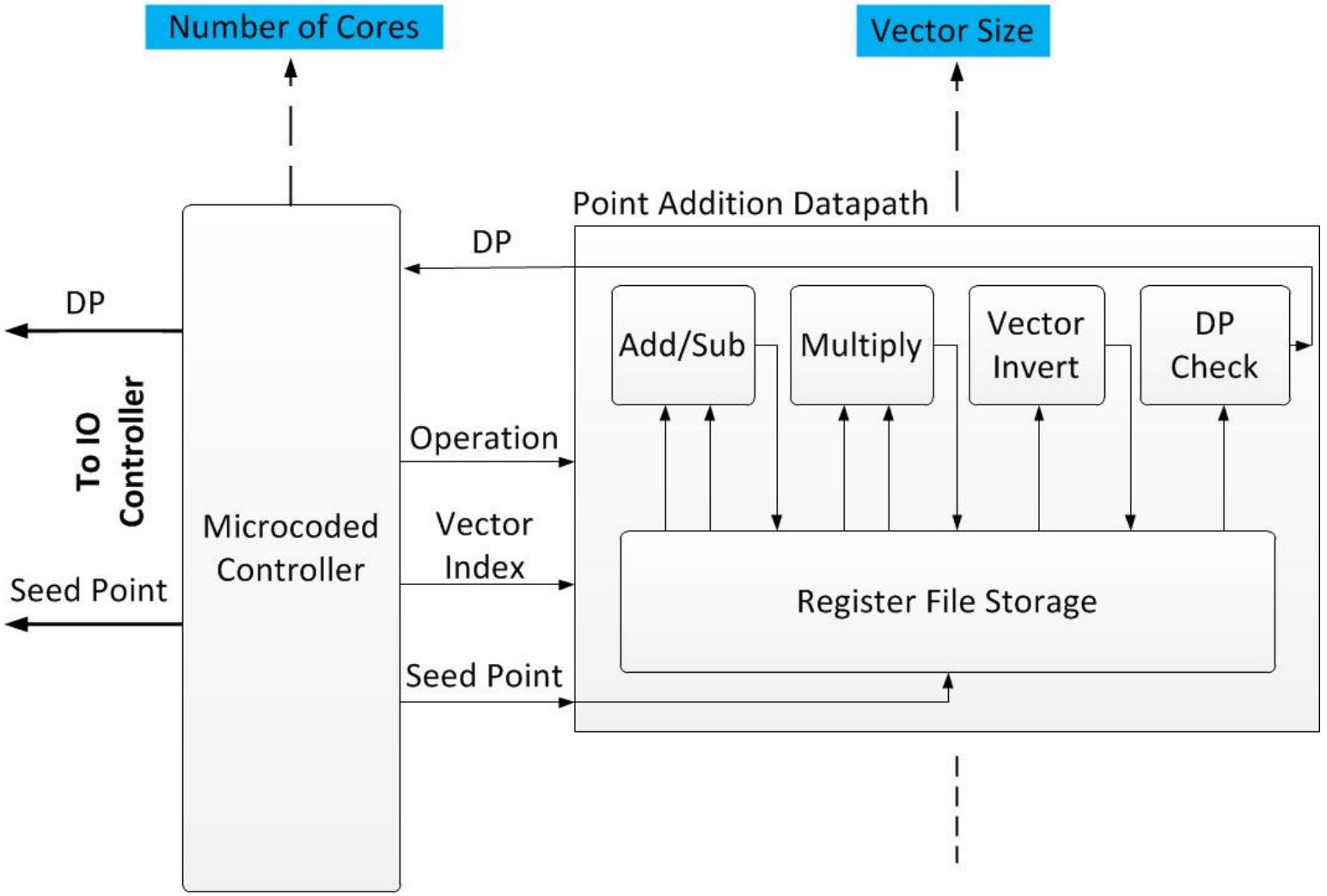
```
rule incr_counter (rcnt < 4'd15);
    rcnt <= rcnt + 1;
endrule

rule assign_res;
    if (rcnt == 4'd14)
        cout <= tagged Valid tmp_res;
    else
        cout <= tagged Invalid;
endrule

method Maybe#(Bit#(128)) result();
    return cout;
endmethod
```

ECDLP Design Space Exploration

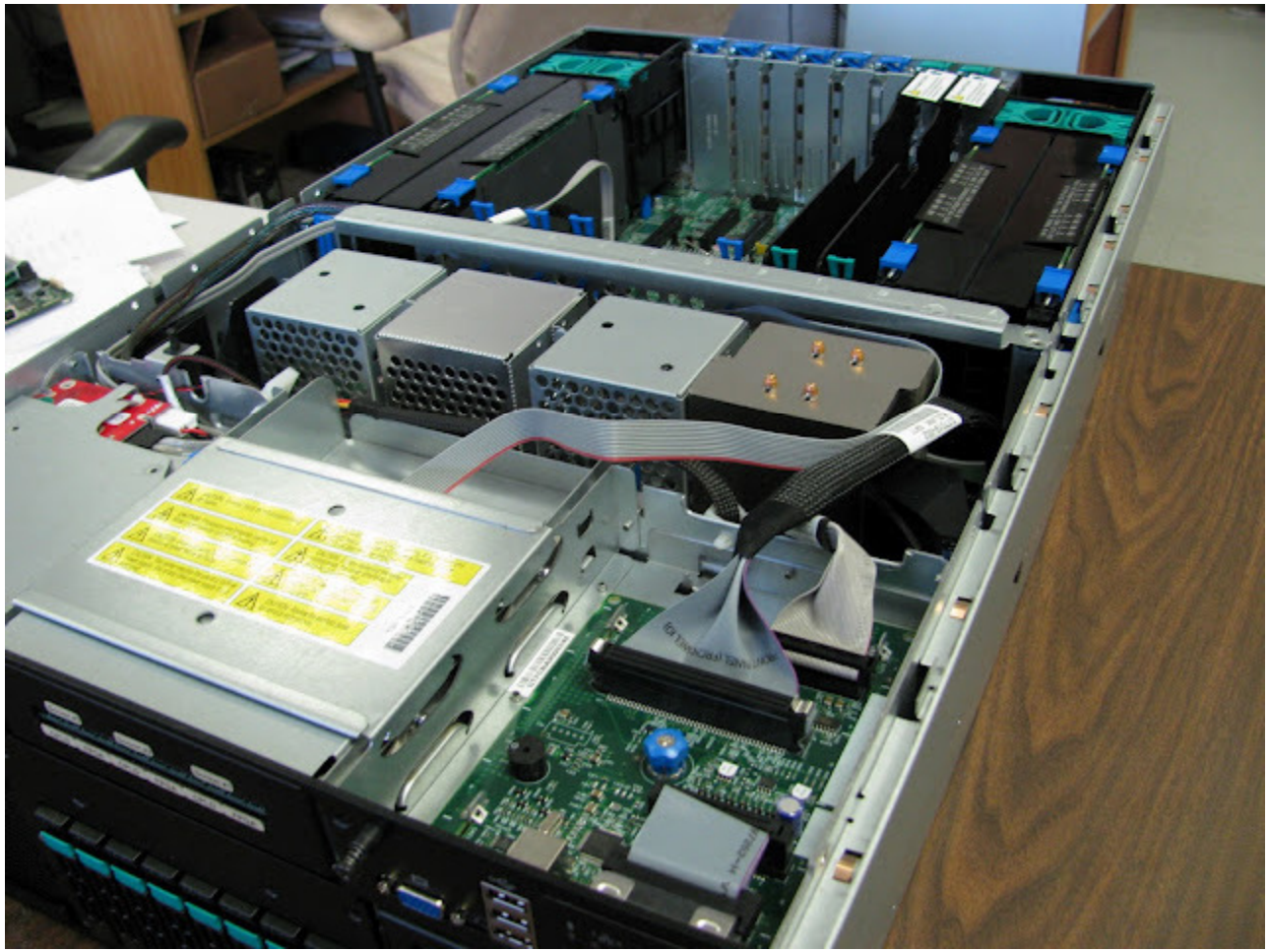
Hardware Architecture



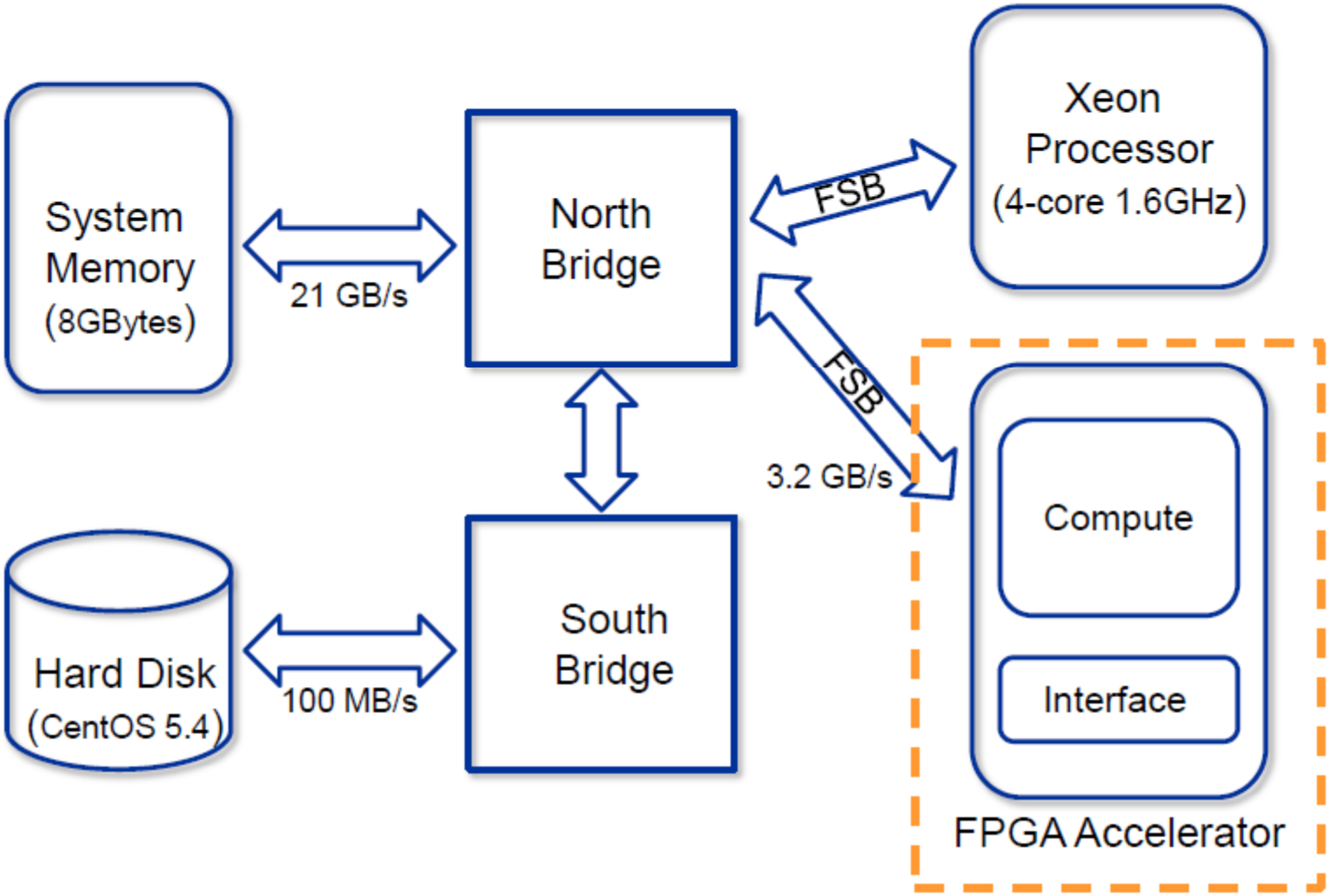
Experiment Details

- Measured results from prototype implemented on Nallatech computing platform
- Demonstrated collisions using seed points of low order (~ 2^{50})
- Open source design available at: https://sourceforge.net/p/ecdlpbluespec/

Nallatech Computing Platform



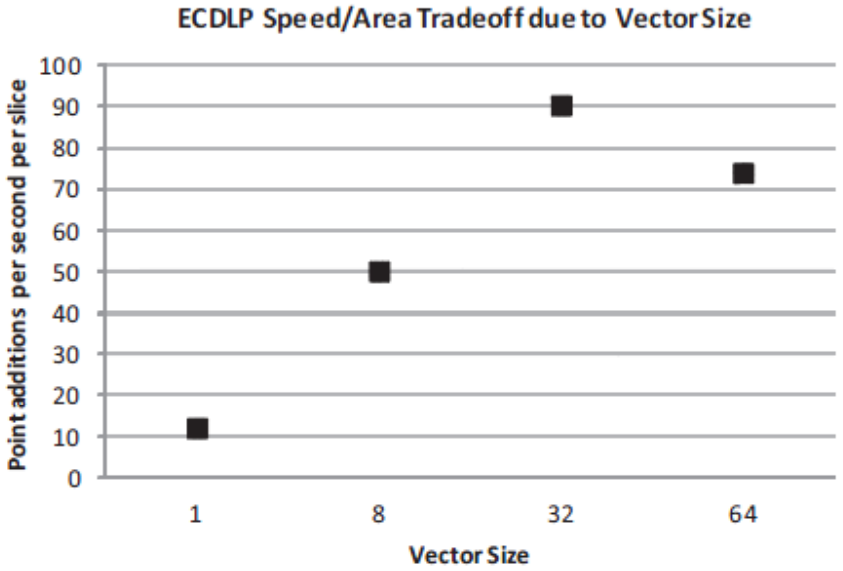
Nallatech Architecture



Implementation Results

Vector Size	One Core		Four Core	
	Speed (PA/s)	Area (slices)	Speed (PA/s)	Area (slices)
1	53K	4407	214K	12,391
8	300K	5977	1.20M	17,705
32	598K	6619	2.38M	29,478
64	717K	9692	2.87M	35,232

Speed/Area Tradeoff



- Optimal design has max. cores with vector size 32
- Extra storage for larger vectors negates performance benefits

Conclusions

- Bluespec supports a high abstraction level for complex designs
- Bluespec is well-suited for quick design space exploration to determine optimal parameters
- Our work is 1st open source ECDLP https://sourceforge.net/p/ecdlpbluespec/

Questions?

This research was supported in part by the National Science Foundation Grant no 0644070