
# Software-Programmable FPGA IoT Platform

**Kam Chuen Mak (Lattice Semiconductor)**
**Andrew Canis (LegUp Computing)**
**July 13, 2016**

# Agenda

- Introduction
  - Who we are
- IoT Platform in FPGA
  - Lattice's IoT Vision
  - IoT Platform Overview
  - Architecture Overview
- RISC-V Architecture
  - RISC-V Processor Overview
- LegUp High-Level Synthesis
  - Design Flow
- Benchmark Results

# Agenda

- Introduction
  - Who we are
- IoT Platform in FPGA
  - Lattice's IoT Vision
  - IoT Platform Overview
  - Architecture Overview
- RISC-V Architecture
  - RISC-V Processor Overview
- LegUp High-Level Synthesis
  - Design Flow
- Benchmark Results

# INTRODUCTION
## Who we are?

**Lattice Semiconductor (NASDAQ: LSCC)**

We provide smart connectivity solutions powered by our low power FPGA, video ASSP, 60 GHz millimeter wave, and IP products to the consumer, communications, industrial, computing, and automotive markets worldwide. Our unwavering commitment to our customers enables them to accelerate their innovation, creating an ever better and more connected world.

# INTRODUCTION
## Who we are?

**Lattice Semiconductor (NASDAQ: LSCC)**
We provide smart connectivity solutions powered by our low power FPGA, video ASSP, 60 GHz millimeter wave, and IP products to the consumer, communications, industrial, computing, and automotive markets worldwide. Our unwavering commitment to our customers enables them to accelerate their innovation, creating an ever better and more connected world.

**LegUp Computing (LegUpComputing.com)**
We are a startup spun out of 7 years of research at the University of Toronto. We provide high-level synthesis tools that compile software into hardware running on an FPGA. Our product enables software engineers to easily target FPGA hardware to achieve huge gains in computational throughput and energy efficiency.

# Agenda

- Introduction
  - Who we are
- IoT Platform in FPGA
  - Lattice's IoT Vision
  - IoT Platform Overview
  - Architecture Overview
- RISC-V Architecture
  - RISC-V Processor Overview
- LegUp High-Level Synthesis
  - Design Flow
- Benchmark Results

Lattice's vision for an FPGA IoT platform:

# IOT PLATFORM IN FPGA
## Lattice's Vision

Lattice's vision for an FPGA IoT platform:

1) **Ease of use**
   - Use C/C++ as our design entry for customers.
   - Users can even create hardware accelerators using C
     - No More Verilog or VHDL

# IOT PLATFORM IN FPGA
## Lattice's Vision

Lattice's vision for an FPGA IoT platform:

1) **Ease of use**
   - Use C/C++ as our design entry for customers.
   - Users can even create hardware accelerators using C
     - No More Verilog or VHDL

We use a **RISC-V with LegUp accelerators** to fulfil this requirement

# IOT PLATFORM IN FPGA
## Lattice's Vision

Lattice's vision for an FPGA IoT platform:

1) **Ease of use**
   - Use C/C++ as our design entry for customers.
   - Users can even create hardware accelerators using C
     - No More Verilog or VHDL

   We use a **RISC-V with LegUp accelerators** to fulfil this requirement

2) **Flexibility**
   - Support a wide range of sensors, actuators, and communication devices APIs
   - Capability to generate custom instructions or acceleration libraries if they are required for the user application

# IOT PLATFORM IN FPGA
## Lattice's Vision

Lattice's vision for an FPGA IoT platform:

1) **Ease of use**
   - Use C/C++ as our design entry for customers.
   - Users can even create hardware accelerators using C
     - No More Verilog or VHDL

   We use a **RISC-V with LegUp accelerators** to fulfil this requirement

2) **Flexibility**
   - Support a wide range of sensors, actuators, and communication devices APIs
   - Capability to generate custom instructions or acceleration libraries if they are required for the user application

   We can use a **FPGA** to fulfil this requirement

# IOT PLATFORM IN FPGA
## Lattice's Vision

Lattice's vision for an FPGA IoT platform:

1) **Ease of use**
   - Use C/C++ as our design entry for customers.
   - Users can even create hardware accelerators using C
     - No More Verilog or VHDL

   We use a **RISC-V with LegUp accelerators** to fulfil this requirement

2) **Flexibility**
   - Support a wide range of sensors, actuators, and communication devices APIs
   - Capability to generate custom instructions or acceleration libraries if they are required for the user application

   We can use a **FPGA** to fulfil this requirement

**A hybrid computing solution:**
Combine the RISC-V Processor with FPGA hardware and utilize our low power and small footprint FPGA advantages for user customization.

# IOT PLATFORM IN FPGA
## Lattice's Vision

**A Hybrid (Processor + FPGA) platform**

# IOT PLATFORM IN FPGA
## Lattice's Vision

**A Hybrid (Processor + FPGA) platform**

Cost

Data center / Cloud Computing

Performance

# IOT PLATFORM IN FPGA
## Lattice's Vision

**A Hybrid (Processor + FPGA) platform**

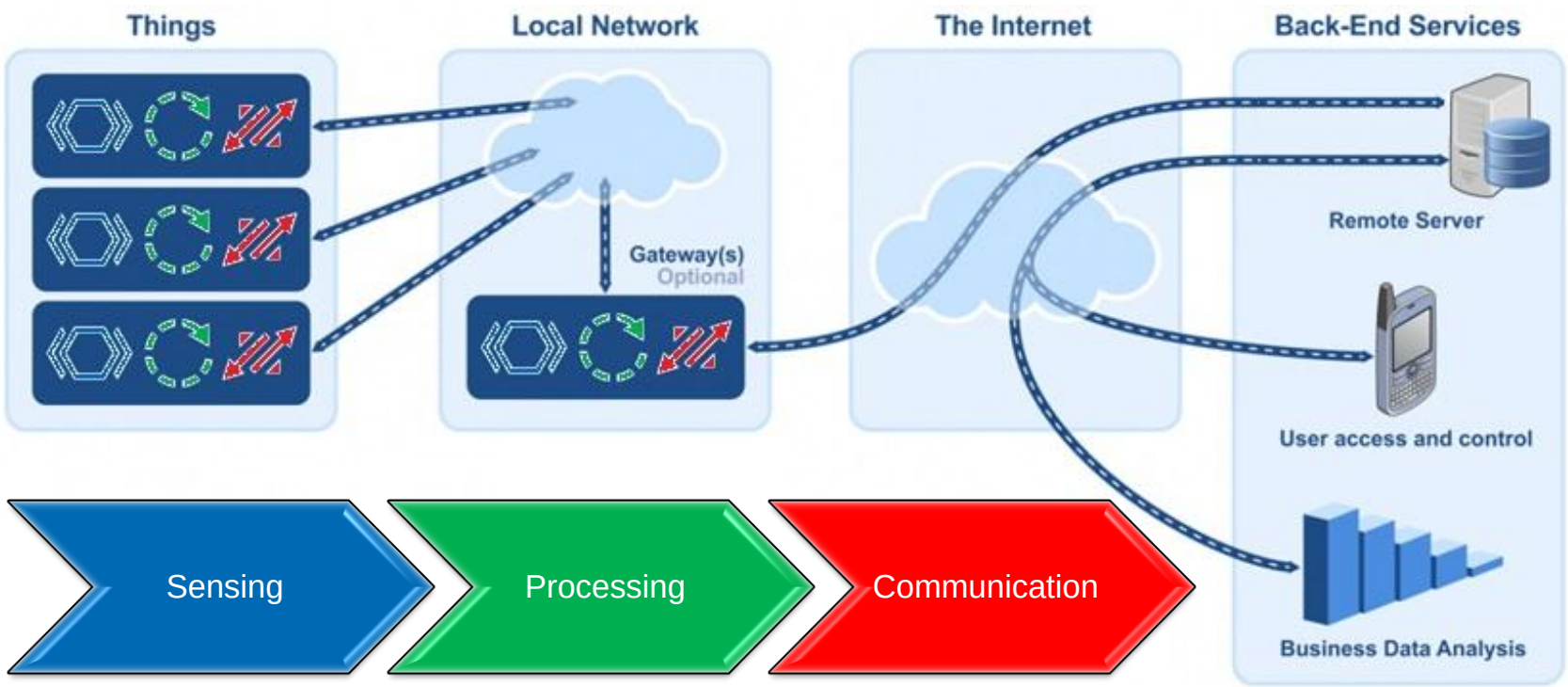
Cost

Embedded, mobile,
Internet-of-Things

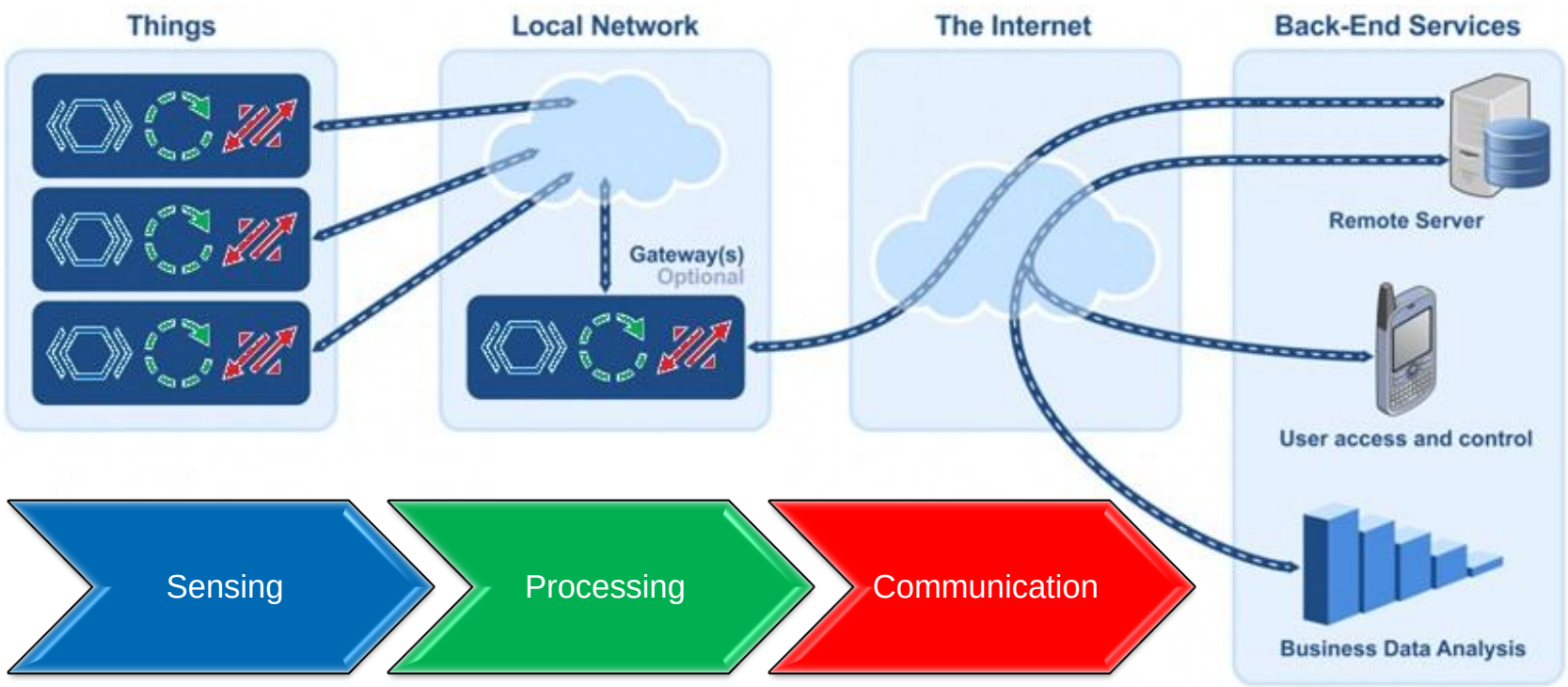

Data center / Cloud Computing

Performance

# IOT PLATFORM IN FPGA
## IoT Platform Overview



# IOT PLATFORM IN FPGA
## IoT Platform Overview




The **Internet of Things (IoT)** refers to the ever-growing network of physical objects that feature an IP address for internet connectivity, and the communication that occurs between these objects and other Internet-enabled devices and systems. (From Webopedia)

# IOT PLATFORM IN FPGA
## IoT Platform Overview

**Sensing**

- Accelerometer
- Gyroscope
- Pressure
- Temperature

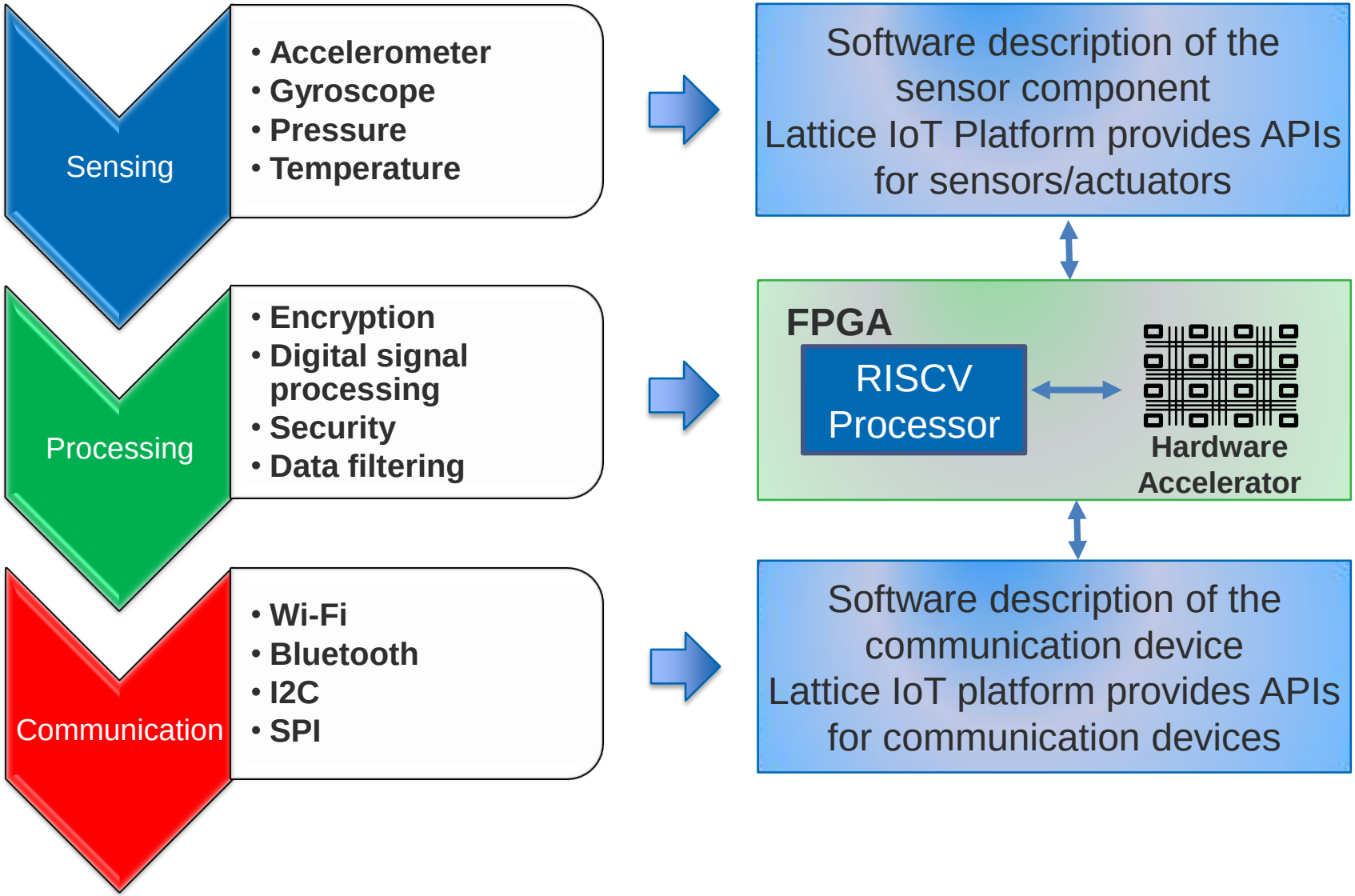
**Processing**

- Encryption
- Digital signal processing
- Security
- Data filtering

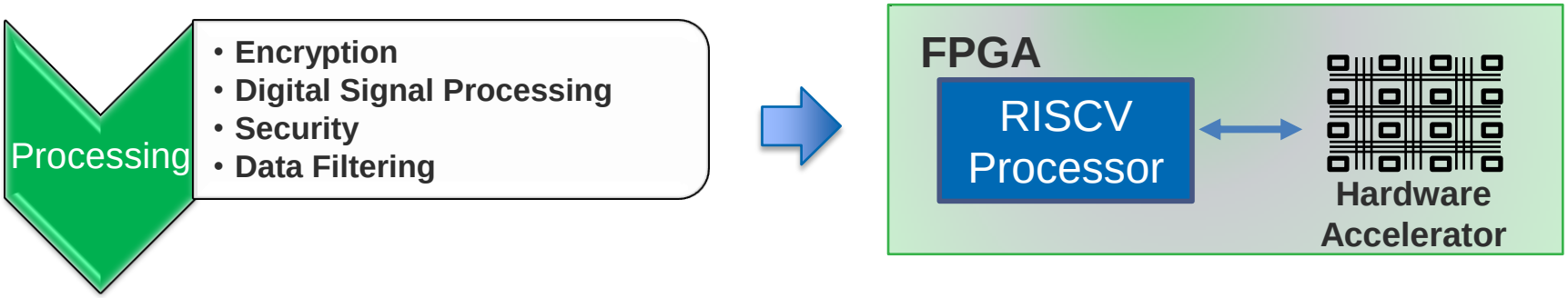
**Communication**

- Wi-Fi
- Bluetooth
- I2C
- SPI

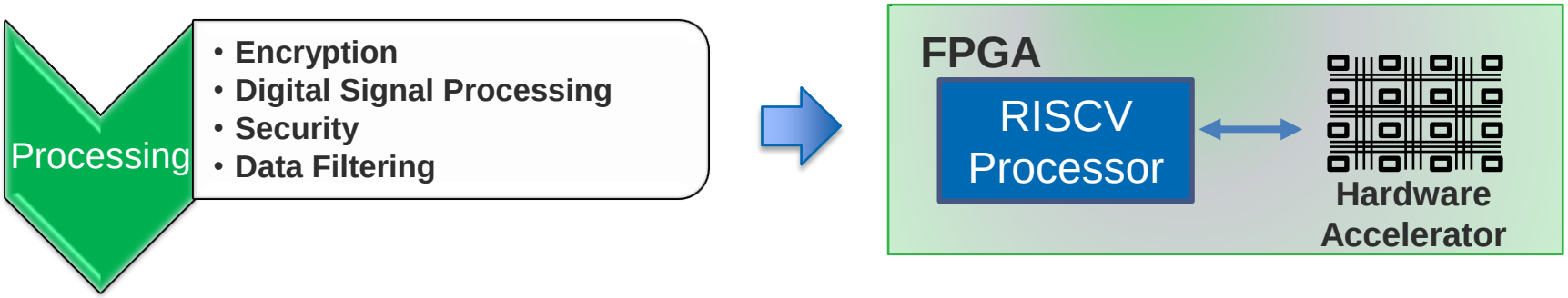
# IOT PLATFORM IN FPGA
## IoT Platform Overview



# IOT PLATFORM IN FPGA
## Architecture Overview



# IOT PLATFORM IN FPGA
## Architecture Overview



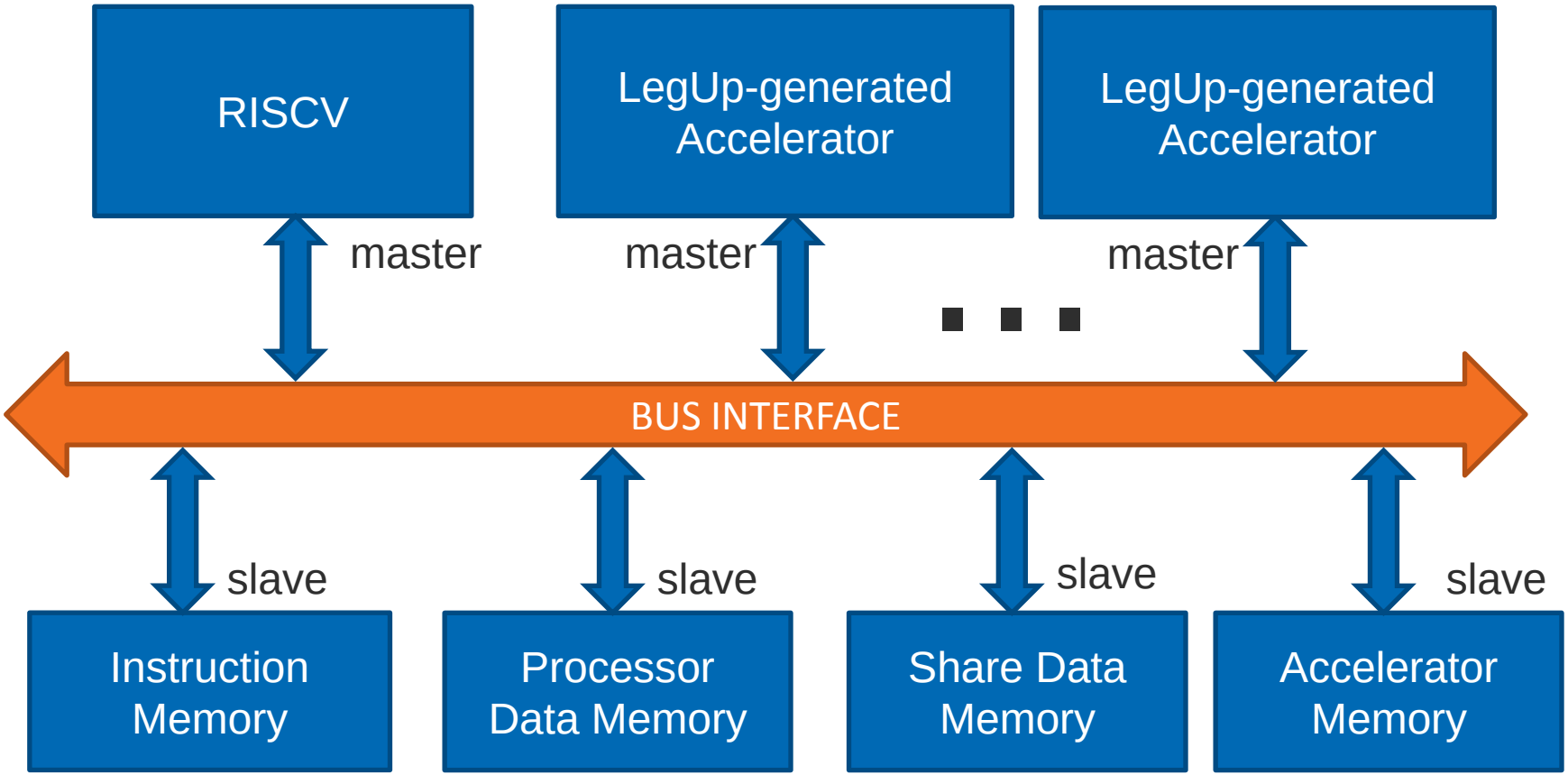

RISC-V processor plus LegUp-generated hardware accelerators to handle the processing part of the IoT Platform:

- **Low-power** and **small** footprint solution
- Identify the critical hotspots in C program, use LegUp to synthesize them into the FPGA and speed up the overall performance
- RISC-V processor executes the rest of the C program
- Maintain a low power and gain high throughput

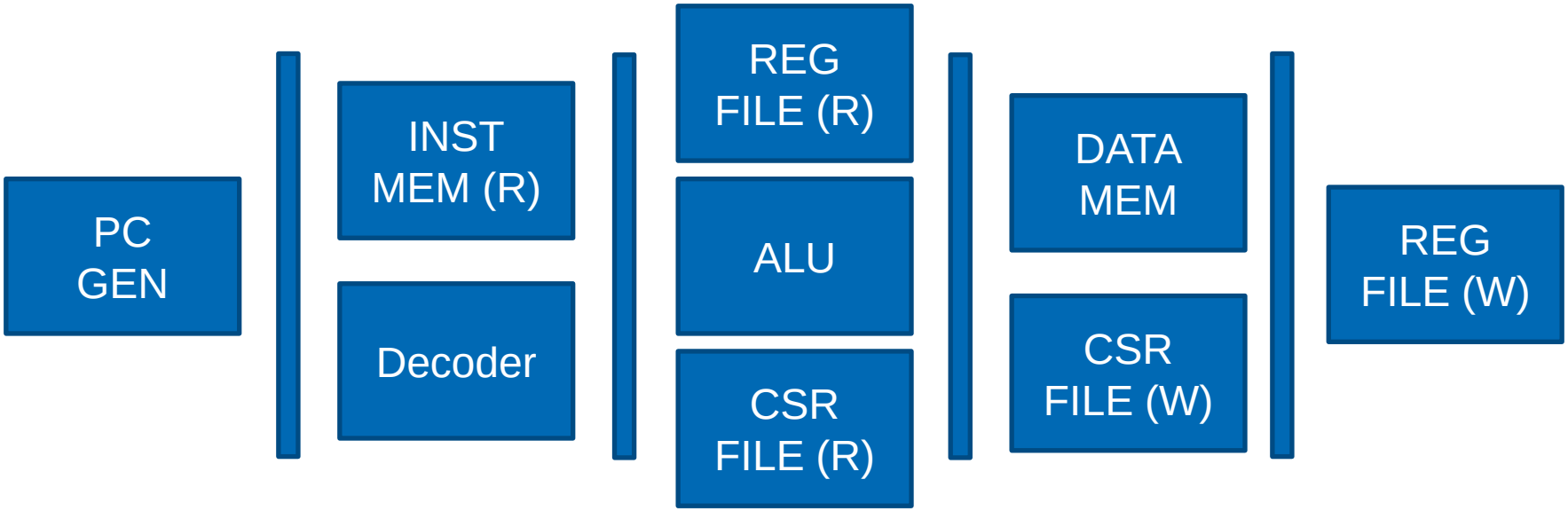
# Agenda

- Introduction
  - Who we are
- IoT Platform in FPGA
  - Lattice's IoT Vision
  - IoT Platform Overview
  - Architecture Overview
- RISC-V Architecture
  - RISC-V Processor Overview
- LegUp High-Level Synthesis
  - Design Flow
- Benchmark Results

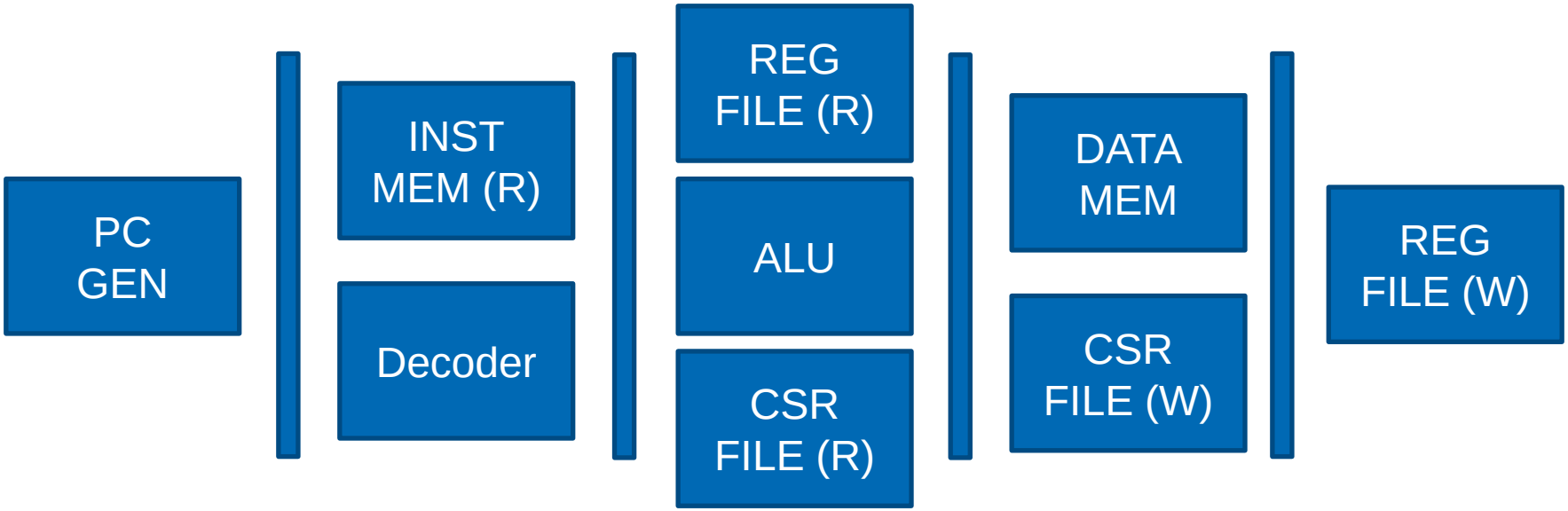
## RISC-V + LegUp-generated accelerators



# RISC-V ARCHITECTURE
## RISC-V Processor Overview



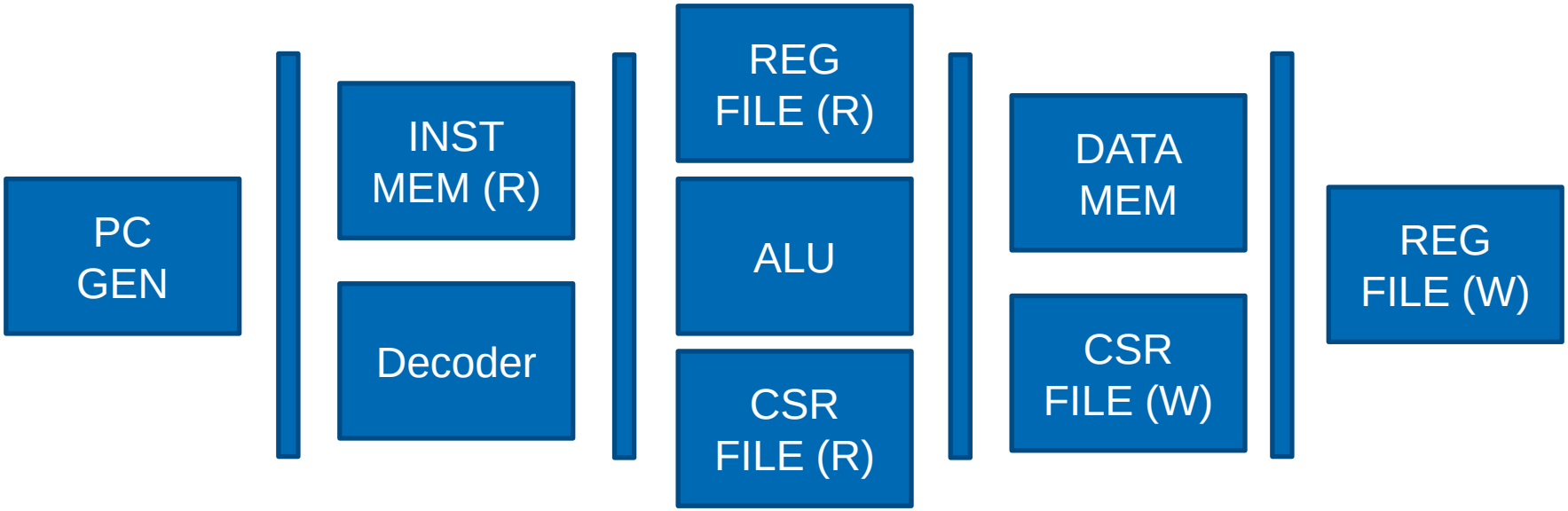
# RISC-V ARCHITECTURE
## RISC-V Processor Overview




Lattice RISC-V Processor:

- 4 stages pipeline CPU Core

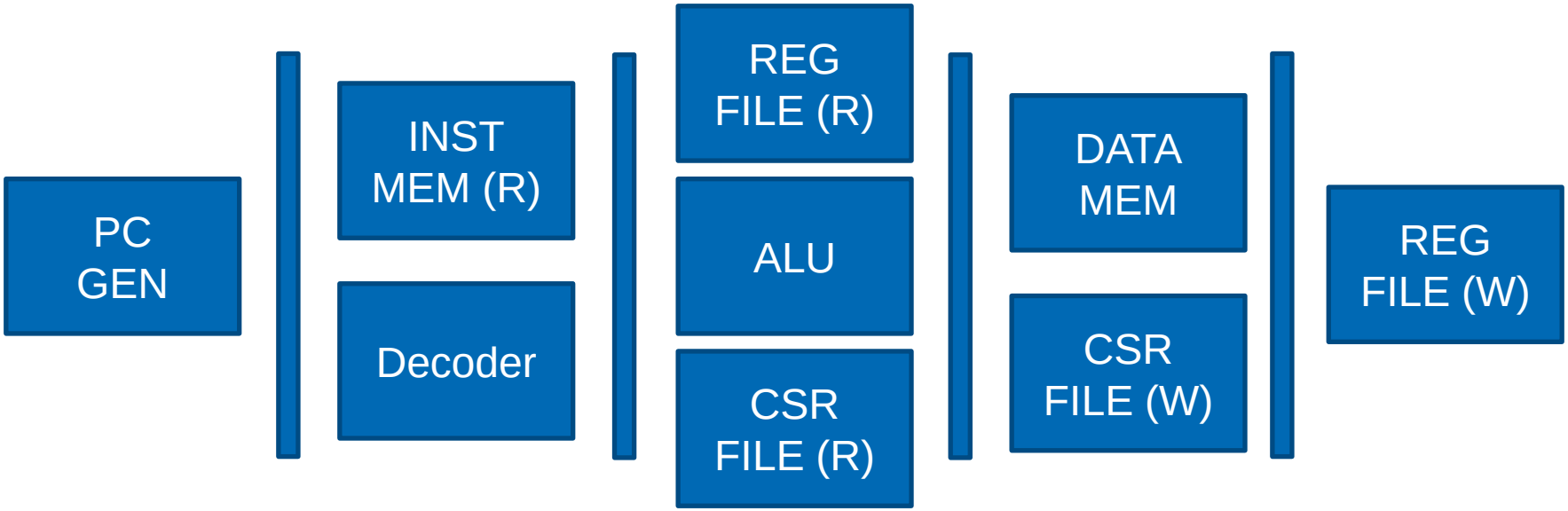
# RISC-V ARCHITECTURE
## RISC-V Processor Overview




Lattice RISC-V Processor:

- 4 stages pipeline CPU Core
- RISC-V **RV32IMC** Instruction Set
  - RV32I v2.0
  - RV32M v2.0 with Multiplier Only
  - RV32C v1.9

# RISC-V ARCHITECTURE
## RISC-V Processor Overview




Lattice RISC-V Processor:

- 4 stages pipeline CPU Core
- RISC-V **RV32IMC** Instruction Set
  - RV32I v2.0
  - RV32M v2.0 with Multiplier Only
  - RV32C v1.9
- Other optional features which can be configured through the Verilog parameters, i.e. Enable external interrupt, allow external stalls from accelerators, and use custom instructions

# RISC-V ARCHITECTURE
## RISC-V Processor Overview

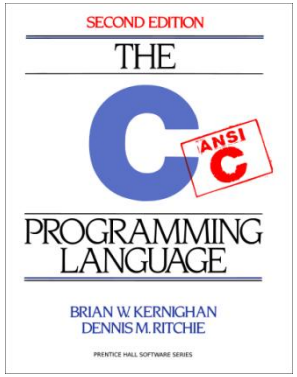
| | RV32I | RV32IM | RV32IC | LM32 | LM32(M) | NIOS 2e* | NIOS 2f* | EM4* | 8051* |
|---|---|---|---|---|---|---|---|---|---|
| DMIPS | 1.35 | 1.64 | 1.35 | 0.8 | 0.9 | 0.15 | 1.16 | 1.5 | 0.1 |
| Area | 1.3K LUTs | 1.5K LUTs +DSP | ~1.6K LUTs | 2K LUTs | 2.5K LUTs | 1.4K LEs | 3K LEs | N/A | N/A |
| Code Density** | 1.45 | 1.4 | 1 | 1.8 | 1.8 | 1.17 | 1.17 | 1 | 1.25 |

*Data from the third party vendor datasheet
**Benchmark using Lattice internal designs

# Agenda

- Introduction
  - Who we are
- IoT Platform in FPGA
  - Lattice's IoT Vision
  - IoT Platform Overview
  - Architecture Overview
- RISC-V Architecture
  - RISC-V Processor Overview
- LegUp High-Level Synthesis
  - Design Flow
- Benchmark Results

# LEGUP HIGH-LEVEL SYNTHESIS




**Software (C code)**

LegUp


**Hardware Description (Verilog)**

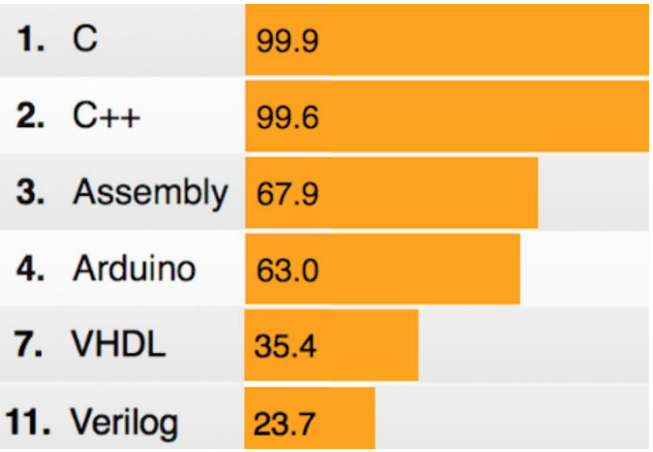
**www.LegUp.org**

**www.LegUpComputing.com**

# BENEFITS OF LEGUP

- Harness parallelism of hardware
- Design in C (#1 embedded language)
  - Many engineers don't know Verilog
- Finish in weeks instead of months
  - Faster time to market
- C testbench 100X faster than simulation
- Allows design space exploration
- Easier debugging in C

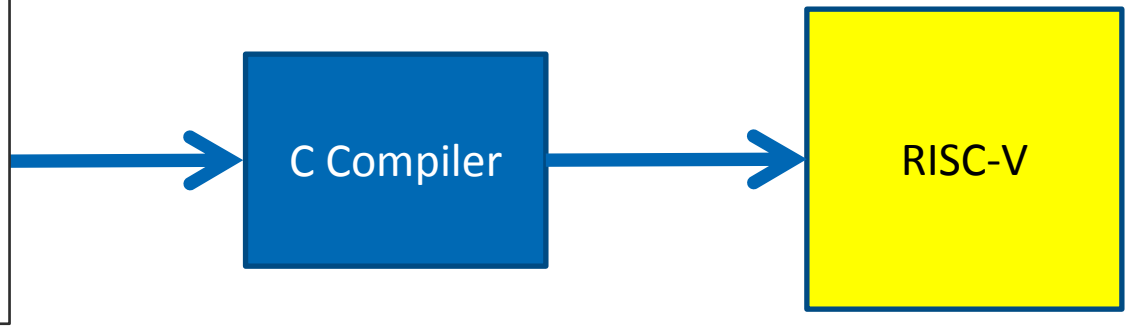
IEEE Spectrum 2015



# LEGUP DESIGN FLOW

```
int FIR(int ntaps, int sum) {
  int i;
  for (i=0; i < ntaps; i++)
    sum += h[i] * z[i];
  return (sum);
}
....
```

Program code



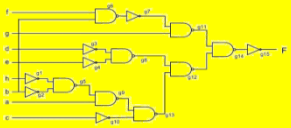
# LEGUP DESIGN FLOW

```
int FIR(int ntaps, int sum) {
  int i;
  for (i=0; i < ntaps; i++)
    sum += h[i] * z[i];
  return (sum);
}
....
```

Program code

C Compiler

**Profiling**



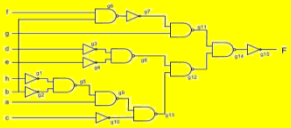
# LEGUP DESIGN FLOW

```
int FIR(int ntaps, int sum) {
  int i;
  for (i=0; i < ntaps; i++)
    sum += h[i] * z[i];
  return (sum);
}
....
```

Program code

C Compiler

**Profiling**



Profiling Data:

Execution Cycles
Power

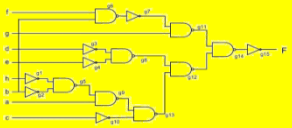
# LEGUP DESIGN FLOW

```
int FIR(int ntaps, int sum) {
  int i;
  for (i=0; i < ntaps; i++)
    sum += h[i] * z[i];
  return (sum);
}
....
```

Program code

C Compiler

**Profiling**



Profiling Data:

Execution Cycles
Power

Suggested program segments to target to HW

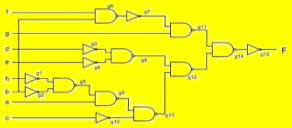
# LEGUP DESIGN FLOW

```
int FIR(int ntaps, int sum) {
  int i;
  for (i=0; i < ntaps; i++)
    sum += h[i] * z[i];
  return (sum);
}
....
```

Program code

C Compiler

**Profiling**



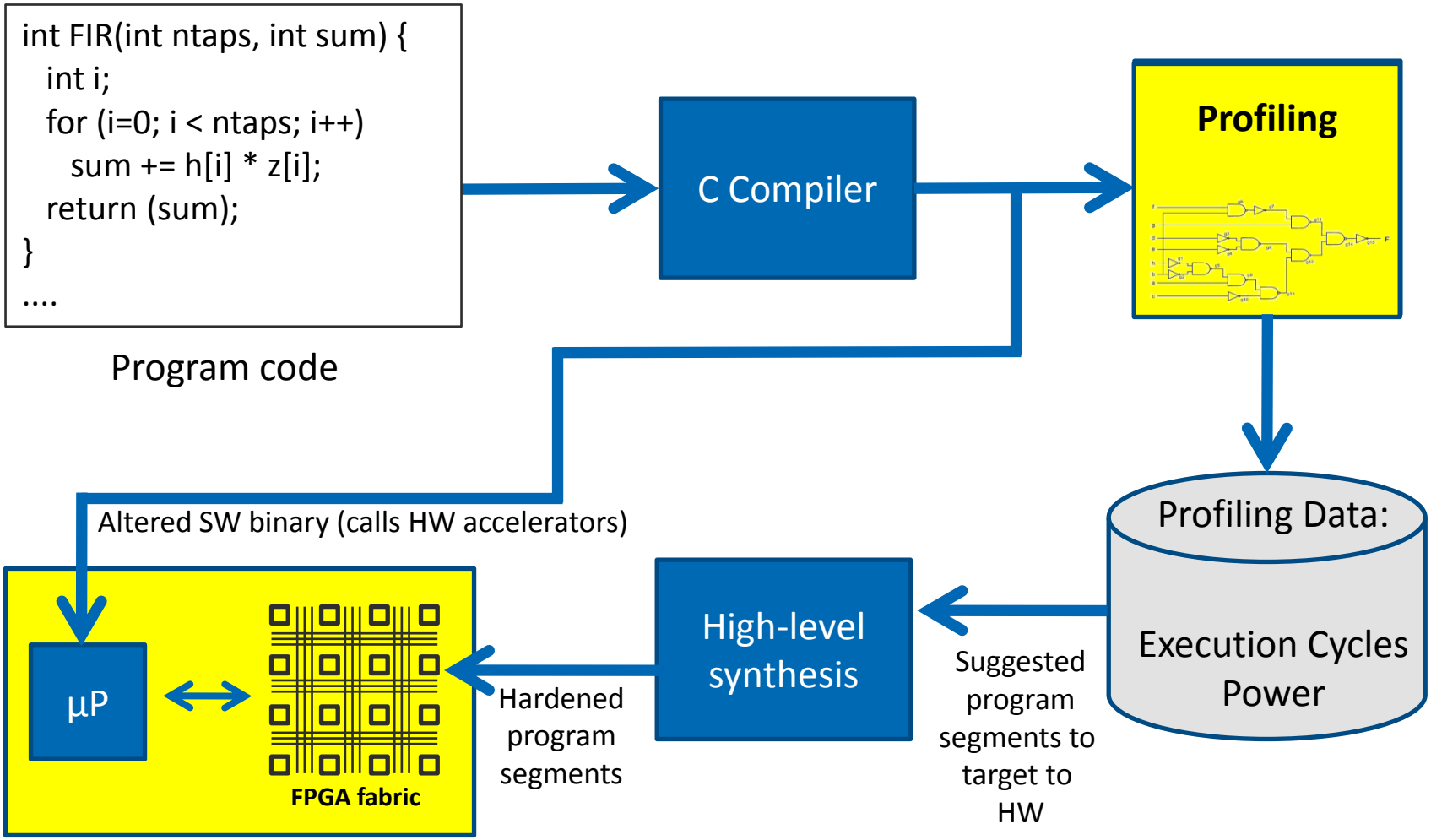
Profiling Data:

Execution Cycles
Power

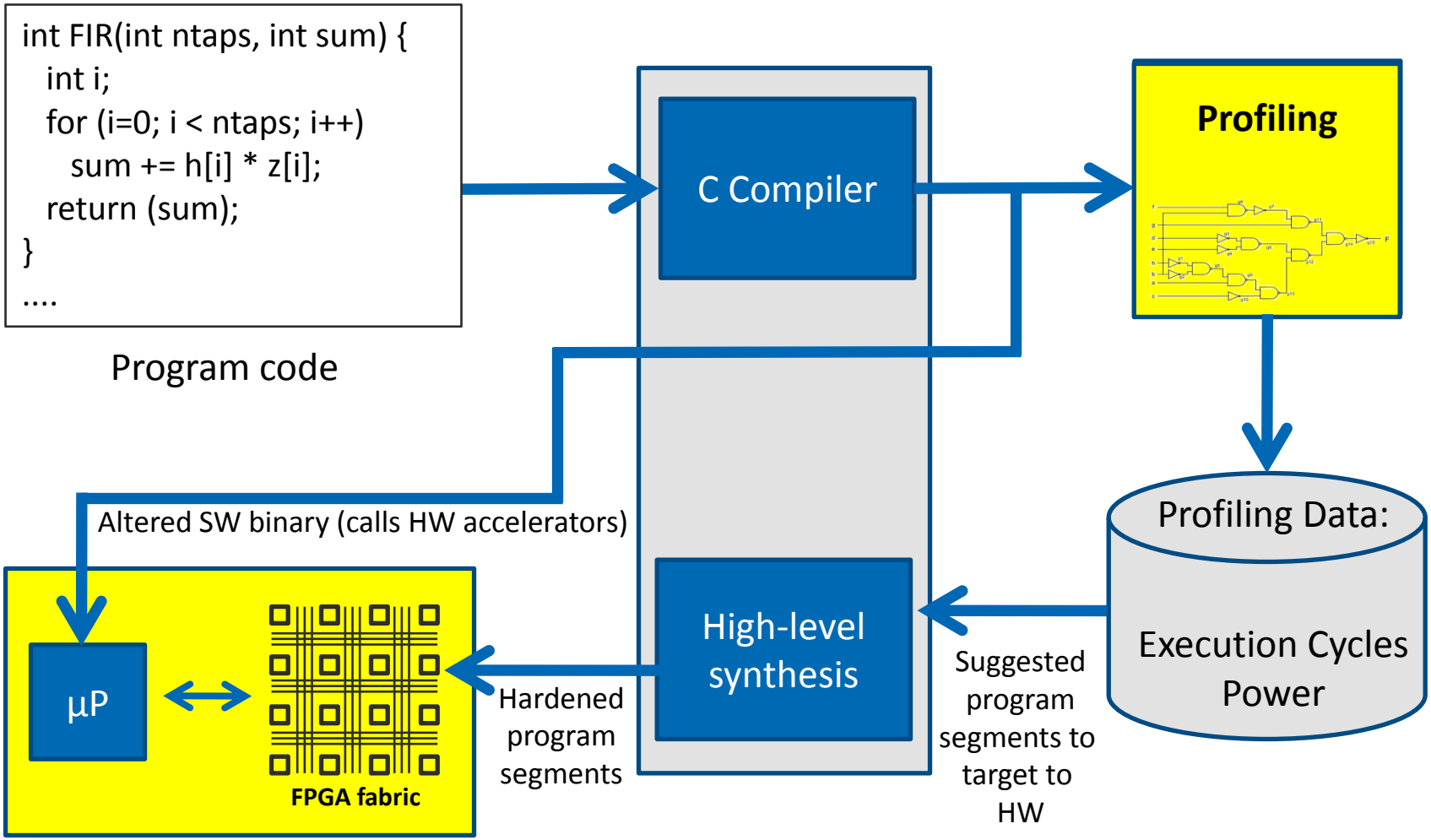
Suggested program segments to target to HW

High-level synthesis

# LEGUP DESIGN FLOW



# LEGUP DESIGN FLOW



# LEGUP DESIGN FLOW

```
int main () {
    …
    sum = dotproduct(a, b, n);
    …
}
```

```
int dotproduct(int *A, int *B,
  int N) {
  …
  for (i=0; i<N; i++) {
        sum += A[i] * B[i];
  }
  return sum;
}
```

# LEGUP DESIGN FLOW

```
int main () {
    ...
    sum = dotproduct(a, b, n);
    ...
}
```

We want to accelerate this C function in hardware

```
int dotproduct(int *A, int *B,
  int N) {
  ...
  for (i=0; i<N; i++) {
        sum += A[i] * B[i];
  }
  return sum;
}
```

# LEGUP DESIGN FLOW

```
int main () {
    ...
    sum = dotproduct(a, b, n);
    ...
}
```

**Specify LegUp Tcl command:**
**set_accelerator_function**
**"dotproduct"**

```
int dotproduct(int *A, int *B,
  int N) {
  ...
  for (i=0; i<N; i++) {
        sum += A[i] * B[i];
  }
  return sum;
}
```

# LEGUP DESIGN FLOW

```
int main () {
    ...
    sum = dotproduct(a, b, n);
    ...
}
```

```
int dotproduct(int *A, int *B,
  int N) {
  ...
  for (i=0; i<N; i++) {
        sum += A[i] * B[i];
  }
  return sum;
}
```

**Specify LegUp Tcl command:**
**set_accelerator_function**
**"dotproduct"**

**Now run LegUp!**

# LEGUP DESIGN FLOW

```
int main () {
    …
    sum = dotproduct(a, b, n);
    …
}
```

```
int dotproduct(int *A, int *B,
  int N) {
  …
  for (i=0; i<N; i++) {
        sum += A[i] * B[i];
  }
  return sum;
}
```

# LEGUP DESIGN FLOW

```
int main () {
    …
    sum = dotproduct(a, b, n);
    …
}
```

```
int dotproduct(int *A, int *B,
  int N) {
  …
  for (i=0; i<N; i++) {
        sum [illegible] * B[i];
  }
  return sum;
}
```

LegUp

# LEGUP DESIGN FLOW

```
int main () {
    …
    sum = dotproduct(a, b, n);
    …
}
```

```
int dotproduct(int *A, int *B,
  int N) {
  …
  for (i=0; i<N; i++) {
        sum += A[i] * B[i];
  }
  return sum;
}
```

LegUp

Verilog
HW Accelerator

# LEGUP DESIGN FLOW

```
int main () {
    ...
    sum = dotproduct(a, b, n);
    ...
}
```

```
#define dotproduct_RET  (volatile int *) 0xf0000000
#define dotproduct_STATUS (volatile int *) 0xf0000008
#define dotproduct_ARG1 (volatile int *) 0xf000000C
#define dotproduct_ARG2 (volatile int *) 0xf0000010
#define dotproduct_ARG3 (volatile int *) 0xf0000014

int legup_dotproduct(int *A, int *B, int N) {
  *dotproduct_ARG1 = (volatile int) A;
  *dotproduct_ARG2 = (volatile int) B;
  *dotproduct_ARG3 = (volatile int) N;
  *dotproduct_STATUS = 1;
  return *dotproduct_RET;
}
```

**Automatically created C "wrapper" to interface with hardware**

# LEGUP DESIGN FLOW

```
int main () {
    ...
    sum = dotproduct(a, b, n);
    ...
}
```

```
#define dotproduct_RET  (volatile int *) 0xf0000000
#define dotproduct_STATUS (volatile int *) 0xf0000008
#define dotproduct_ARG1 (volatile int *) 0xf000000C
#define dotproduct_ARG2 (volatile int *) 0xf0000010
#define dotproduct_ARG3 (volatile int *) 0xf0000014

int legup_dotproduct(int *A, int *B, int N) {
  *dotproduct_ARG1 = (volatile int) A;
  *dotproduct_ARG2 = (volatile int) B;
  *dotproduct_ARG3 = (volatile int) N;
  *dotproduct_STATUS = 1;
  return *dotproduct_RET;
}
```

# LEGUP DESIGN FLOW

```
int main () {
    ...
    sum = legup_dotproduct(a,b,n);
    ...
}
```

**Automatically replace function calls**

```
#define dotproduct_RET  (volatile int *) 0xf0000000
#define dotproduct_STATUS (volatile int *) 0xf0000008
#define dotproduct_ARG1 (volatile int *) 0xf000000C
#define dotproduct_ARG2 (volatile int *) 0xf0000010
#define dotproduct_ARG3 (volatile int *) 0xf0000014

int legup_dotproduct(int *A, int *B, int N) {
  *dotproduct_ARG1 = (volatile int) A;
  *dotproduct_ARG2 = (volatile int) B;
  *dotproduct_ARG3 = (volatile int) N;
  *dotproduct_STATUS = 1;
  return *dotproduct_RET;
}
```

# LEGUP DESIGN FLOW

```
int main () {
    ...
sum = legup_dotproduct(a,b,n);
    ...
}
```

**Automatically replace function calls**

```
#define dotproduct_RET  (volatile int *) 0xf0000000
#define dotproduct_STATUS (volatile int *) 0xf0000008
#define dotproduct_ARG1 (volatile int *) 0xf000000C
#define dotproduct_ARG2 (volatile int *) 0xf0000010
#define dotproduct_ARG3 (volatile int *) 0xf0000014

int legup_dotproduct(int *A, int *B, int N) {
    *dotproduct_ARG1 = (volatile int) A;
    *dotproduct_ARG2 = (volatile int) B;
    *dotproduct_ARG3 = (volatile int) N;
    *dotproduct_STATUS = 1;
    return *dotproduct_RET;
}
```

# LEGUP DESIGN FLOW

```
int main () {
    ...
  sum = legup_dotproduct(a,b,n);
    ...
}
```

```
#define dotproduct_RET  (volatile int *) 0xf0000000
#define dotproduct_STATUS (volatile int *) 0xf0000008
#define dotproduct_ARG1 (volatile int *) 0xf000000C
#define dotproduct_ARG2 (volatile int *) 0xf0000010
#define dotproduct_ARG3 (volatile int *) 0xf0000014

int legup_dotproduct(int *A, int *B, int N) {
  *dotproduct_ARG1 = (volatile int) A;
  *dotproduct_ARG2 = (volatile int) B;
  *dotproduct_ARG3 = (volatile int) N;
  *dotproduct_STATUS = 1;
  return *dotproduct_RET;
}
```

# LEGUP DESIGN FLOW

```
int main () {
    ...
  sum = legup_dotproduct(a,b,n);
    ...
}
```

```
#define dotproduct_RET  (volatile int *) 0xf0000000
#define dotproduct_STATUS (volatile int *) 0xf0000008
#define dotproduct_ARG1 (volatile int *) 0xf000000C
#define dotproduct_ARG2 (volatile int *) 0xf0000010
#define dotproduct_ARG3 (volatile int *) 0xf0000014

dotproduct(int *A, int *B, int N) {
  duct_ARG1 = (volatile int) A;
  duct_ARG2 = (volatile int) B;
  *dotproduct_ARG3 = (volatile int) N;
  *dotproduct_STATUS = 1;
  return *dotproduct_RET;
}
```

**Software Compiler**

# LEGUP DESIGN FLOW

```
int main () {
    ...
  sum = legup_dotproduct(a,b,n);
    ...
}
```

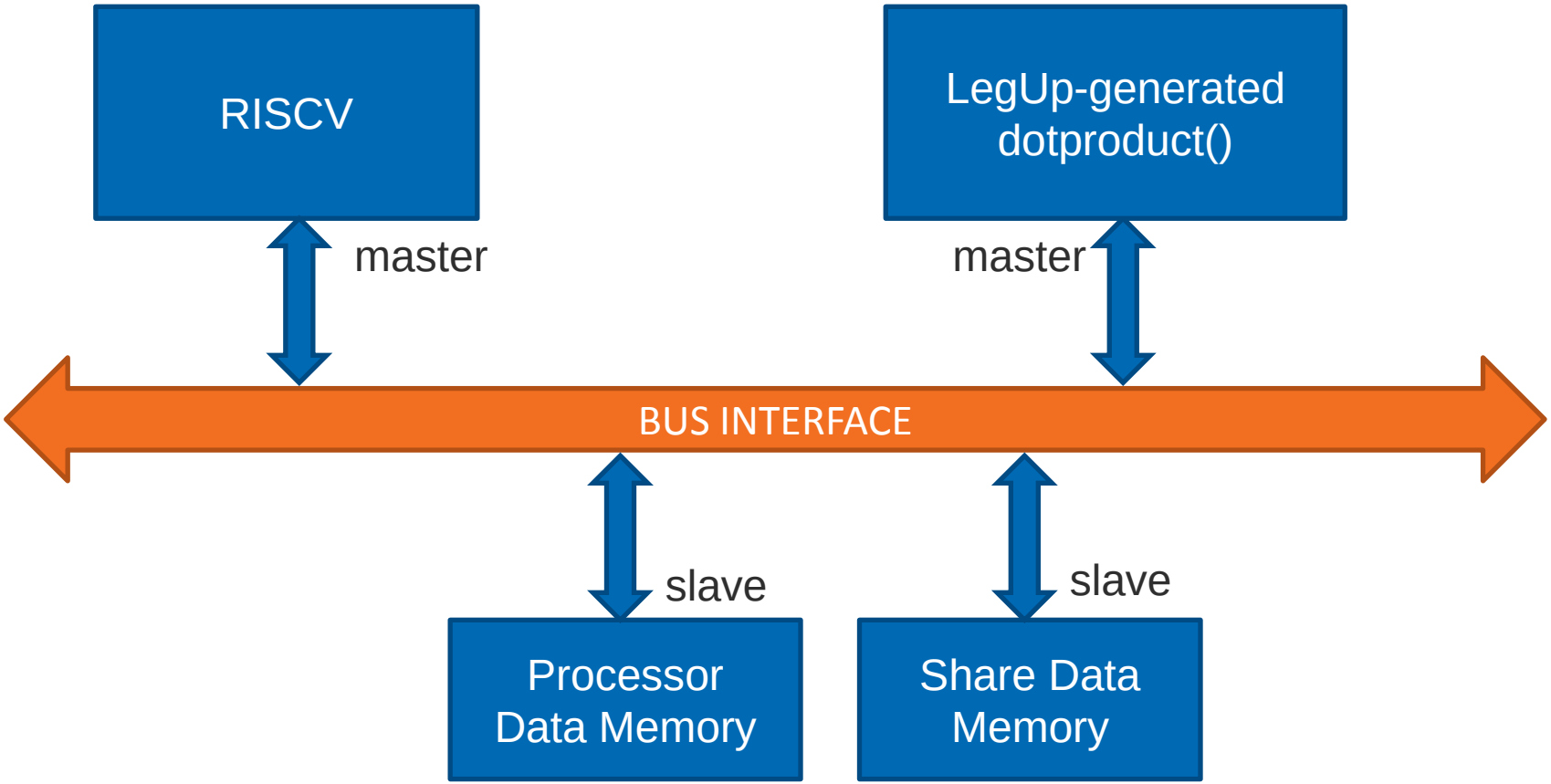
```
#define dotproduct_RET  (volatile int *) 0xf0000000
#define dotproduct_STATUS (volatile int *) 0xf0000008
#define dotproduct_ARG1 (volatile int *) 0xf000000C
#define dotproduct_ARG2 (volatile int *) 0xf0000010
#define dotproduct_ARG3 (volatile int *) 0xf0000014

dotproduct(int *A, int *B, int N) {
duct_ARG1 = (volatile int) A;
duct_ARG2 = (volatile int) B;
*dotproduct_ARG3 = (volatile int) N;
*dotproduct_STATUS = 1;
n *dotproduct_RET;
```

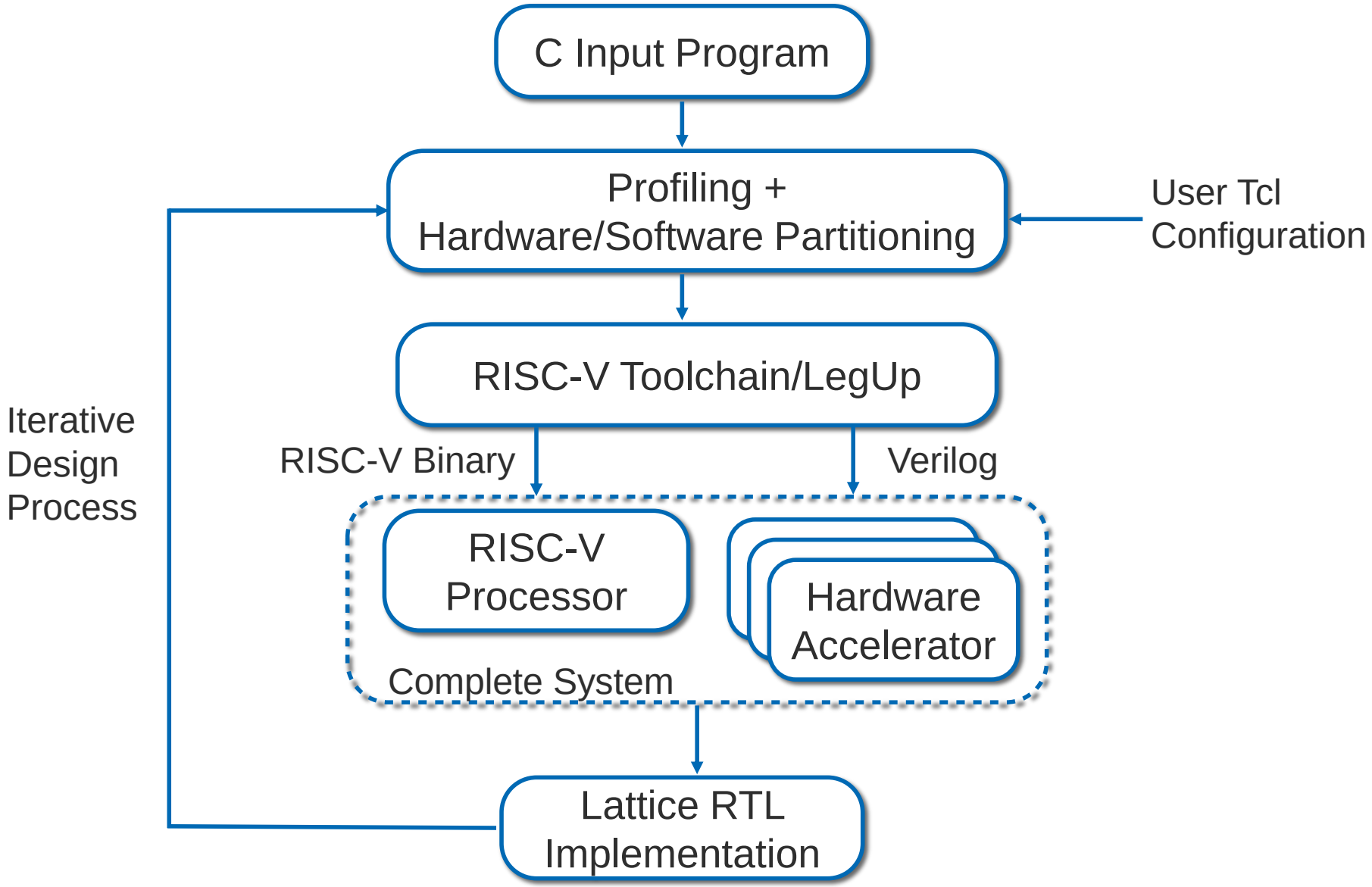
**Software Compiler**

**RISC-V Processor**

## RISC-V + LegUp-generated accelerators



# USER DESIGN FLOW



# CASE STUDY

- Energy is calculated as a sum of squares of speech samples:

```
for (i = 0; i < 256; i++)
        energy += samples[i]*samples[i];
```

- Running on RISC-V32IM (DMIPS – 1.64):
  - Each loop iteration takes about 6 cycles: load, increment address, multiply, add, branch
  - Clock cycles to complete: **1550**

- LegUp synthesized accelerator:
  - Generates a pipelined hardware circuit exploiting parallelism
  - One iteration completes every clock cycle
  - Clock cycles to complete: **288**

  - Speedup: **5.4X**

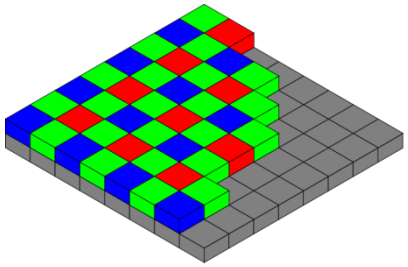
# Agenda

- Introduction
  - Who we are
- IoT Platform in FPGA
  - Lattice's IoT Vision
  - IoT Platform Overview
  - Architecture Overview
- RISC-V Architecture
  - RISC-V Processor Overview
- LegUp High-Level Synthesis
  - Design Flow
- Benchmark Results

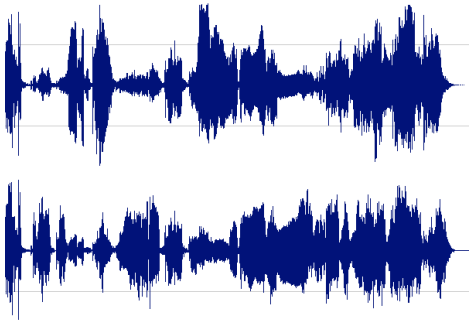
# STREAMING BENCHMARKS

## Edge Detection: 4 image filters

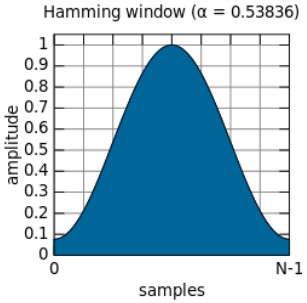
## Bayer/deBayer Filter



## Beamforming

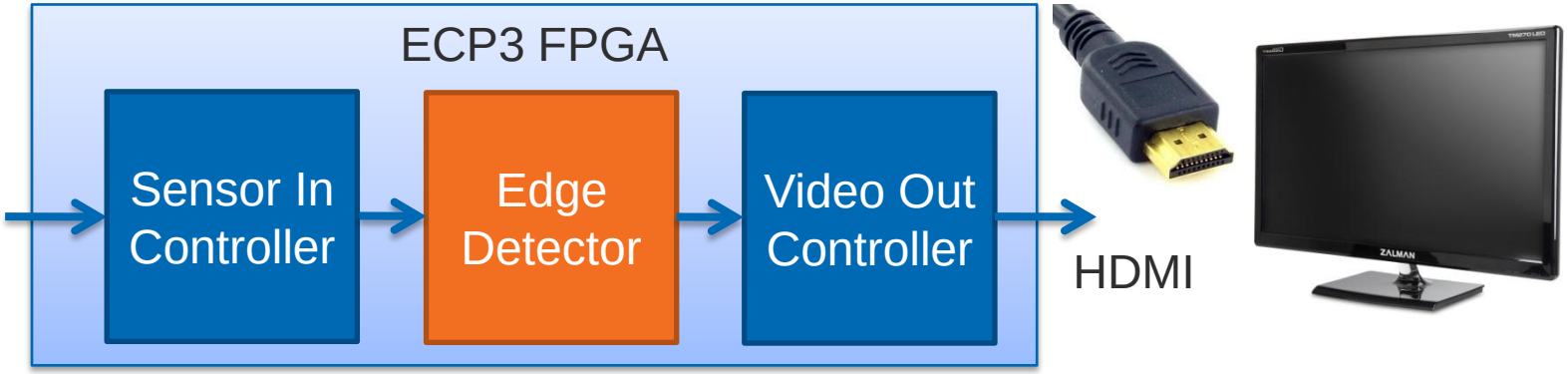


## FIR Filter



# STREAMING BENCHMARKS

Aptina Image Sensor


**Input: 1 pixel/cycle**

**Output: 1 pixel/cycle**

Verilog

C

Clock Frequency: 74 MHz

60 fps

# STREAMING BENCHMARKS

**Average across 9 benchmarks**

**LegUp vs RTL: FMax within 30% and slices < 10% larger**

| | RTL | LegUp/RTL |
|---|---|---|
| Time (us) | 243 | 1.32 |
| Cycles | 38,544 | 1.00 |
| FMax (MHz) | 159 | 0.76 |
| Slices | 612 | 1.08 |
| Registers | 424 | 1.15 |
| LUT4s | 972 | 1.18 |
| DSPs | 16 | 1.00 |

**Metric: Time = cycles * clock period**

- **Time taken to complete the entire benchmark**

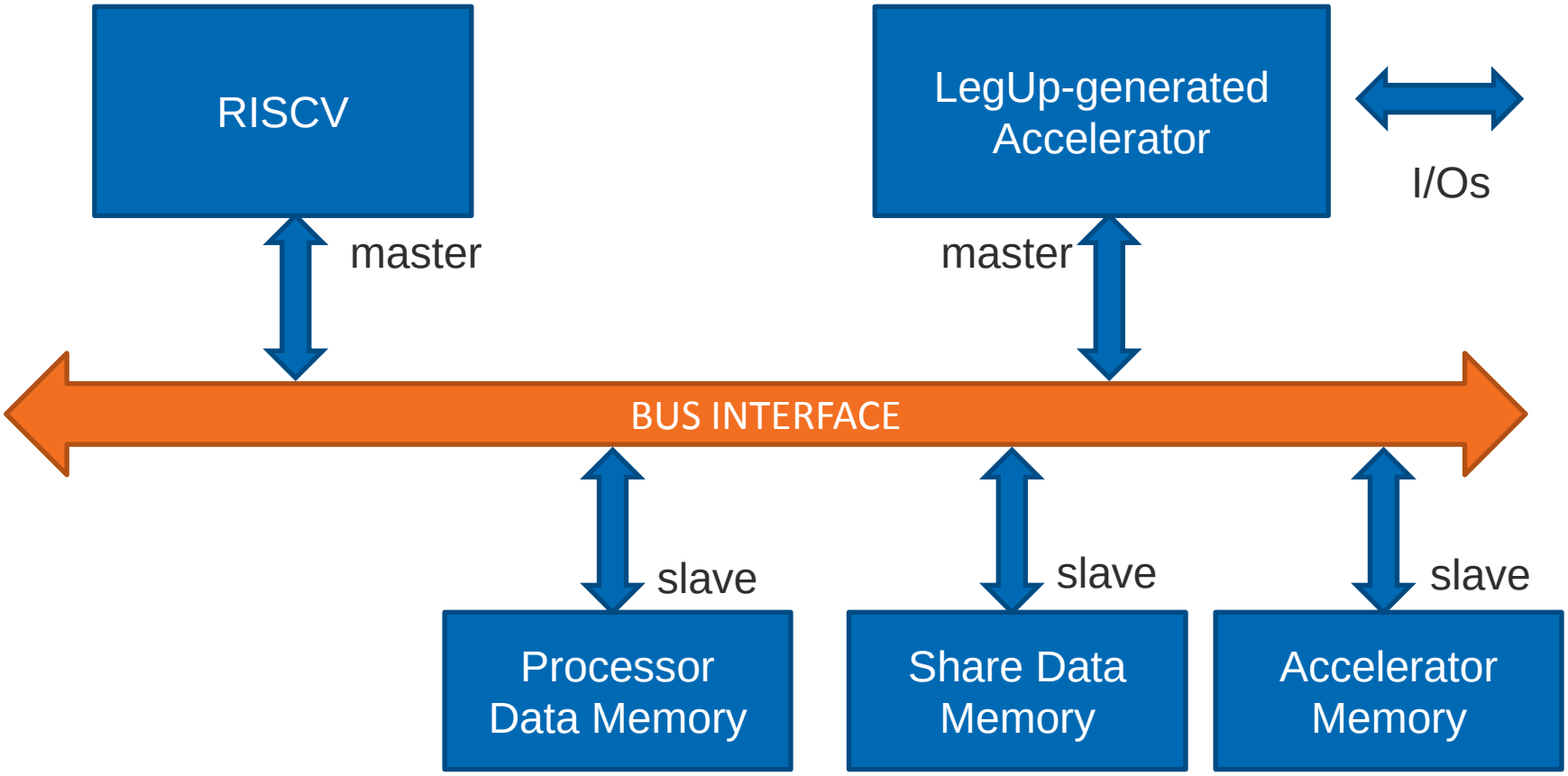
# RISC-V+LEGUP BENCHMARKS

## Accelerate most-compute intensive function with LegUp

| Benchmark | RISC-V clock cycles | RISC-V + LegUp clock cycles | Speedup |
|---|---|---|---|
| Energy sum | 1,550 | 288 | 5.4 |
| OPUS function | 48,947 | 16,523 | 3.0 |
| FIR filter | 44,610 | 13,049 | 3.4 |
| Matrix multiply | 67,823 | 28,105 | 2.4 |
| ADPCM | 88,288 | 20,400 | 4.3 |
| AES | 29,071 | 12,278 | 2.4 |
| Blowfish | 752,774 | 460,274 | 1.6 |
| GSM | 17,092 | 6,243 | 2.7 |
| Motion | 13,193 | 4,912 | 2.7 |
| SHA | 726,321 | 190,451 | 3.8 |
| **Geomean** | **48,542** | **16,107** | **3.0** |

# IOT PLATFORM IN FPGA
## Architecture Overview

**RISC-V + LegUp-generated coprocessor connected to I/Os**



# CUSTOM INSTRUCTIONS

- The current architecture assumes we want to accelerate a large chunk of code
- Instead the user may want to add a custom instruction
  - i.e. multiply-accumulate for DSP applications
- LegUp can synthesize the hardware needed for the custom instruction
- Hardware accelerator is tightly coupled with RISC-V processor: can read/write registers
- Benefit: area efficient reuse of hardware many times
- Work in progress

# THANK YOU
# ANY QUESTION?