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:
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
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
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
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
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.
A Hybrid (Processor + FPGA) platform
IOT PLATFORM IN FPGA
Lattice’s Vision

A Hybrid (Processor + FPGA) platform

Cost

Data center / Cloud Computing

Performance
A Hybrid (Processor + FPGA) platform

Embedded, mobile, Internet-of-Things

Data center / Cloud Computing

Cost

Performance
IOT PLATFORM IN FPGA
IoT Platform Overview

Sensing  Processing  Communication

Things  Local Network  The Internet  Back-End Services

Gateway(s) Optional

Remote Server
User access and control
Business Data Analysis
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 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

Sensing
- Accelerometer
- Gyroscope
- Pressure
- Temperature

Processing
- Encryption
- Digital signal processing
- Security
- Data filtering

Communication
- Wi-Fi
- Bluetooth
- I2C
- SPI

Software description of the sensor component
Lattice IoT Platform provides APIs for sensors/actuators

FPGA
RISCV Processor
Hardware Accelerator

Software description of the communication device
Lattice IoT platform provides APIs for communication devices
IOT PLATFORM IN FPGA
Architecture Overview

- Encryption
- Digital Signal Processing
- Security
- Data Filtering

FPGA

RISCV Processor

Hardware Accelerator
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
Lattice RISC-V Processor:
• 4 stages pipeline CPU Core
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
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

<table>
<thead>
<tr>
<th></th>
<th>RV32I</th>
<th>RV32IM</th>
<th>RV32IC</th>
<th>LM32</th>
<th>LM32(M)</th>
<th>NIOS 2e*</th>
<th>NIOS 2f*</th>
<th>EM4*</th>
<th>8051*</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMIPS</td>
<td>1.35</td>
<td>1.64</td>
<td>1.35</td>
<td>0.8</td>
<td>0.9</td>
<td>0.15</td>
<td>1.16</td>
<td>1.5</td>
<td>0.1</td>
</tr>
<tr>
<td>Area</td>
<td>1.3K LUTs</td>
<td>1.5K LUTs</td>
<td>~1.6K LUTs</td>
<td>2K LUTs</td>
<td>2.5K LUTs</td>
<td>1.4K LEs</td>
<td>3K LEs</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Code Density**</td>
<td>1.45</td>
<td>1.4</td>
<td>1</td>
<td>1.8</td>
<td>1.8</td>
<td>1.17</td>
<td>1.17</td>
<td>1</td>
<td>1.25</td>
</tr>
</tbody>
</table>

*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)

www.LegUp.org
www.LegUpComputing.com

Hardware Description (Verilog)
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

<table>
<thead>
<tr>
<th>Rank</th>
<th>Language</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>99.9</td>
</tr>
<tr>
<td>2</td>
<td>C++</td>
<td>99.6</td>
</tr>
<tr>
<td>3</td>
<td>Assembly</td>
<td>67.9</td>
</tr>
<tr>
<td>4</td>
<td>Arduino</td>
<td>63.0</td>
</tr>
<tr>
<td>5</td>
<td>Python</td>
<td>61.4</td>
</tr>
<tr>
<td>6</td>
<td>Verilog</td>
<td>23.7</td>
</tr>
<tr>
<td>7</td>
<td>VHDL</td>
<td>35.4</td>
</tr>
</tbody>
</table>
int FIR(int ntaps, int sum) {
    int i;
    for (i=0; i < ntaps; i++)
        sum += h[i] * z[i];
    return (sum);
}

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

....

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

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

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

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

....
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;
}
We want to accelerate this C function in hardware

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

```c
int dotproduct(int *A, int *B, int N) {
    ...
    for (i=0; i<N; i++) {
        sum += A[i] * B[i];
    }
    return sum;
}
```
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;
}
We want to accelerate this C function in hardware.

Specify LegUp Tcl command:
```
set_accelerator_function “dotproduct”
```

Now run LegUp!
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;
}
int main () {
   
   sum = dotproduct(a, b, n);
   
}
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;
}
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
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;
}
int main () {
    ... 
    sum = legup_dotproduct(a,b,n);
    ... 
}
int main () {
    ... 
    sum = legup_dotproduct(a,b,n);
    ... 
}

Automatically replace function calls

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;
}
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 *) 0xf00000010
# define dotproduct_ARG3 (volatile int *) 0xf00000014

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;
}
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;
}
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) {
    *dotproduct_ARG1 = (volatile int) A;
    *dotproduct_ARG2 = (volatile int) B;
    *dotproduct_ARG3 = (volatile int) N;
    *dotproduct_STATUS = 1;
    return *dotproduct_RET;
}
RISC-V + LegUp-generated accelerators

RISC-V

LegUp-generated dotproduct()

Processor Data Memory

Share Data Memory

BUS INTERFACE
USER DESIGN FLOW

- **C Input Program**
- **Profiling + Hardware/Software Partitioning**
  - **RISC-V Toolchain/LegUp**
    - **RISC-V Binary**
    - **Verilog**
    - **RISC-V Processor**
    - **Hardware Accelerator**
    - **Complete System**
- **Lattice RTL Implementation**
- **User Tcl Configuration**

Iterative Design Process
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

ECP3 FPGA

Sensor In Controller → Edge Detector → Video Out Controller

Output: 1 pixel/cycle

Clock Frequency: 74 MHz
60 fps

Verilog
C
## STREAMING BENCHMARKS

Average across 9 benchmarks

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

<table>
<thead>
<tr>
<th>Metric</th>
<th>RTL</th>
<th>LegUp/RTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (us)</td>
<td>243</td>
<td>1.32</td>
</tr>
<tr>
<td>Cycles</td>
<td>38,544</td>
<td>1.00</td>
</tr>
<tr>
<td>FMax (MHz)</td>
<td>159</td>
<td>0.76</td>
</tr>
<tr>
<td>Slices</td>
<td>612</td>
<td>1.08</td>
</tr>
<tr>
<td>Registers</td>
<td>424</td>
<td>1.15</td>
</tr>
<tr>
<td>LUT4s</td>
<td>972</td>
<td>1.18</td>
</tr>
<tr>
<td>DSPs</td>
<td>16</td>
<td>1.00</td>
</tr>
</tbody>
</table>

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

- Time taken to complete the entire benchmark
## RISC-V+LEGUP BENCHMARKS

Accelerate most-compute intensive function with LegUp

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