RISC-V as a basis for ASIP design
A Quantum-Resistant IoT Security Implementation
Agenda

• Introductions
• RISC-V and ASIPs
• Implementation of Security Methods
• Performance results
Codasip and SecureRF

- ASIP IP and tool company
  - Czech-based and venture backed
  - Research began 2006, launched 2014
- 40 R&D, worldwide sales and support
- Products in production use

World’s first “Linear-in-Time” Asymmetric Security
Addressing Authentication and Data Protection for the “Smallest” Internet of Things
“Every design is different, so why is every embedded processor the same?”
Why RISC-V and ASIPs?

Common questions...

- Where do I start?
- What happens if the company goes away?
- Are there standard extensions for ...?
- Do you support my favorite [RT]OS?
- Can you benchmark against ...?

RISC-V based ASIPs address all of this.
Implementing RISC-V for ASIP

- Codix Cores are high level processor models
  - Easily adapted to an application need
  - Codasip Studio generates SDK, RTL, etc.
  - Available for wide range of applications

- Codix-Bk is our RISC-V compatible processor line
  - Fully supported by LLVM based SDK, with profiler, simulator, debugger, etc.

• Berkelium Processor Features
  • Available in 3 and 6-stage pipelines
  • RV32IM and RV64IM support
  • Option “C” compact instruction set support
  • Configurable general purpose registers
  • Compiler or hardware-based hazard avoidance
  • Interrupt support
  • Configurable Branch Prediction Unit
  • Optional floating point
  • Optional instruction and data cache
  • JTAG support
  • Sleep mode support
SecureRF’s Security Solutions

• Quantum-resistant security for low resource devices using Group Theoretic Cryptography

• Toolkits: Key Agreement and Authentication protocols, Digital Signatures, Hash Functions

• Existing implementations on 8051, MSP430, ARM M0/M3, FPGAs and ASICs

• Today’s discussion: WalnutDSA™, a digital signature algorithm running on Codasip Z-Scale-inspired Codix-Bk core
The signature verification portion of the algorithm runs 46-178x faster than ECDSA, with half the code size and significantly lower energy consumption.
HW Platform and Testing

- Microsemi SmartFusion2+ board
  - Same board used in the workshop/tutorial
  - Used Codasip generated JTAG for in-circuit debug
WalnutDSA vs. ECDSA

<table>
<thead>
<tr>
<th>Metric</th>
<th>Micro-ECC</th>
<th>WalnutDSA (software only)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Improvement</td>
</tr>
<tr>
<td>Verification time</td>
<td>680 ms</td>
<td>10.7 ms</td>
</tr>
<tr>
<td>Code size (bytes)</td>
<td>9,076</td>
<td>4,504</td>
</tr>
</tbody>
</table>

Notes:
1. Micro-ECC compiled with -Os optimizations for Codix Berkelium core
2. Micro-ECC curve: secp256r1
3. WalnutDSA security level: $2^{128}$
4. Signatures stored as C constants
Using Codasip Studio

Integrated design, programming and debug

Advanced profiling and optimization
The Process

• Profiling used to identify resource intensive ops
  – Multiplication in GF(32) takes 24 clock cycles in software

• Replaced with single cycle custom instruction
  – Wrapped into the function and used as before:

```c
/* quad_gmul32 : A wrapper function around gmul32 that calculates
   four products in GF(32)
*/
uint32_t __attribute__((always_inline)) quad_gmul32(uint32_t argA,
                                                   uint32_t argB)
{
    uint32_t res;
    __asm__ ( "quad_gmul32 [%res], [%a], [%b]" :
        [res]="r"(res) : [a]"r"(argA),
        [b]"r"(argB));
    return res;
}
```
CodAL Processor Modeling

Single description generates RTL, ToolChain, VSP, verification environment

- Describe virtually any processor architecture
- C-like language Similar to “LISA” or “nML”
- Minimal model only needed for SDK generation

Modular and flexible

- Supports multiple implementations for single design
- Supports multiprocessor and platform prototyping

Supports external components

- Reuse existing implementations, or work in Chisel environment
The extensions

- SDK Support

```c
/* quad_gmul32 : A wrapper function around gmul32 that calculates
* four products in GF(32) (one for each byte in
* argA, argB)
*/

uint32 quad_gmul32(uint32 argA, uint32 argB)
{
    uint32 res;
    uint32 mask;
    uint8 i;

    res = 0;
    for (i=0; i<32; i+=8) {
        /* Isolate a single byte from a and b,
           write the result back
           * to the corresponding byte in res
        */
        mask = 0xFF << i;
        res |= (uint32)gmul32((argA & mask) >> i,
                              (argB & mask) >> i) << i;
    }
    return res;
}
```

- HW Support

```c
/* quad_gmul32 : A wrapper function around gmul32 that calculates
* four products in GF(32) (one for each byte in
* argA, argB)
*/

inline uint32 hw_quad_gmul32(uint32 argA, uint32 argB)
{
    uint32 res;
    uint8 res0, res1, res2, res3;

    /* Isolate a single byte from a and b, write the result back
       * to the corresponding byte in res
    */
    res0 = hw_gmul32(argA[ 7.. 0], argB[ 7.. 0]);
    res1 = hw_gmul32(argA[15.. 8], argB[15.. 8]);
    res2 = hw_gmul32(argA[23..16], argB[23..16]);
    res3 = hw_gmul32(argA[31..24], argB[31..24]);

    res = res3 :: res2 :: res1 :: res0;
    return res;
}
```

single instruction for GF(32) multiplication, only combinatorial logic:

quad_gmul32
## FPGA Resource Utilization

### On Microsemi M2S010 SmartFusion2

<table>
<thead>
<tr>
<th>Resource</th>
<th>Without Extension</th>
<th>With Extension</th>
<th>Overhead</th>
<th>Overhead %</th>
</tr>
</thead>
<tbody>
<tr>
<td>4LUT</td>
<td>5,797</td>
<td>5,913</td>
<td>116</td>
<td>1.9%</td>
</tr>
<tr>
<td>DFF</td>
<td>1,909</td>
<td>1,912</td>
<td>3</td>
<td>0.16%</td>
</tr>
<tr>
<td>Logic Element</td>
<td>5,851</td>
<td>5,988</td>
<td>137</td>
<td>2.3%</td>
</tr>
</tbody>
</table>

### Microsemi SmartFusion2 family

<table>
<thead>
<tr>
<th></th>
<th>M2S005</th>
<th>M2S010</th>
<th>M2S025</th>
<th>M2S050</th>
<th>M2S060</th>
<th>M2S090</th>
<th>M2S150</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Elements</td>
<td>6,060</td>
<td>12,084</td>
<td>27,696</td>
<td>56,340</td>
<td>56,520</td>
<td>86,316</td>
<td>146,124</td>
</tr>
<tr>
<td>% Utilization</td>
<td>98.8%</td>
<td>49.5%</td>
<td>21.6%</td>
<td>10.6%</td>
<td>10.5%</td>
<td>6.9%</td>
<td>4.0%</td>
</tr>
</tbody>
</table>
WalnutDSA utilizes tiny operands—just 5-bit—which makes it possible to implement hardware accelerators in very few gates.

The result on the Codasip Codix-Bk core is a nearly 3x speed-up over software alone by adding just 137 logic elements out of an available 12,084.
## WalnutDSA vs. ECDSA Redux

<table>
<thead>
<tr>
<th>Metric</th>
<th>Micro-ECC</th>
<th>WalnutDSA without Ext.</th>
<th>WalnutDSA with Ext.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Improvement</td>
<td>Improvement</td>
</tr>
<tr>
<td>Verification time</td>
<td>680 ms</td>
<td>10.7 ms</td>
<td>3.8 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>63x faster</td>
<td>178x faster</td>
</tr>
<tr>
<td>Code size (bytes)</td>
<td>9,076</td>
<td>4,504</td>
<td>3,052</td>
</tr>
<tr>
<td></td>
<td></td>
<td>50% smaller</td>
<td>66% smaller</td>
</tr>
</tbody>
</table>

**Notes:**
1. Micro-ECC compiled with -Os optimizations for Codix Berkelium core
2. Micro-ECC curve: secp256r1
3. WalnutDSA security level: $2^{128}$
4. Signatures stored as C constants
5. WalnutDSA code size = 3,912/2,980 bytes (with/without Ext.) with –Os optimizations
**WalnutDSA, GTC, and ASIP**

**GTC-based cryptography is faster than ECC and is quantum resistant**
- WalnutDSA signature verification runs up to 63x faster than ECDSA in software
- GTC-based cryptography uses tiny operands (< 8 bit) so hardware acceleration uses very few logic resources

**Hardware acceleration of GTC improves run time by almost 3x**
- At the expense of only 2% additional resources
- More possible with additional effort

**GTC methods plus Codasip ASIP tools streamlined development process**
- From concept to implementation took only one week
- Still RISC-V (no need for a custom ISA) with full LLVM-based SDK including ISS
- Significantly reduces risk since can leverage the complete RISC-V ecosystem
Trying it for yourself

Connect with SecureRF at the poster session
• Tonight

Codasip tools easily accessible
• Have an extensive industry and university partner programs

RISC-V models available
• Incl. the GF32 design files

RTH Security for a mobile world.
Trying it for yourself

Security development kits available from SecureRF, for:

• Key Agreement/Authentication
• Digital Signature Algorithms—Generation and Validation
• Hash Functions
• Secure Boot/Secure Load
• Certificate Functions
Contact Information

Dan Ganousis
ganousis@codasip.com

www.Codasip.com

Derek Atkins
DAtkins@SecureRF.com

www.SecureRF.com

Twitter: @SecureRF