A RISC-V ISA EXTENSION FOR ULTRA-LOW POWER IOT WIRELESS SIGNAL PROCESSING

Carolynn Bernier, Hela Belhadj Amor, Zdeněk Přikryl

Oct 1, 2019
ULP WIRELESS DESIGN @ LETI

2003

- RFID

2005

- ULP RF SoC
- Atmel-Starchip

2010

- VHBR 65nm
- LDR-TCR

Today

- UWB / RFID
- Hybrid UWB/RFID
- Wake-up Radio
- UMETAG
Motivation: A software-defined “Smart” wireless transceiver for IoT

- **PHY-agnostic solution for LPWA-IoT**
  - Address « multi-mode » markets and lower hardware bug fix costs

- **Offer future-proofed designs to our clients**
  - Our clients’ advanced prototypes have evolving needs: satellite-IoT, Ultra-wide band localization, LPWA-IoT.

- **A new experimental platform**
  - Design new “RF software sensors”
  - Use light-weight ML algorithms to extract information from the RF signal
• Bottleneck: Existing software-defined radio (SDR) solutions are **NOT** ULP!

<table>
<thead>
<tr>
<th>Programmability</th>
<th>Flexibility</th>
<th>Portability</th>
<th>Modularity</th>
<th>Computing Power</th>
<th>Energy Efficiency</th>
<th>Soft Core</th>
<th>FPGA</th>
<th>Cost (USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imagine-based [151]</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>Medium</td>
<td>Low</td>
<td>Imagine Stream Processor</td>
<td>N/A</td>
</tr>
<tr>
<td>USRP X300 [17]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>High</td>
<td>Low</td>
<td>PC</td>
<td>Xilinx Virtex-2</td>
</tr>
<tr>
<td>USRP E310 [17]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>High</td>
<td>High</td>
<td>Dual-core ARM Cortex-A9</td>
<td>Xilinx Artix-4</td>
</tr>
<tr>
<td>KUAR [34]</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>Medium</td>
<td>Low</td>
<td>PC + 2 x PowerPC cores</td>
<td>Xilinx Virtex II Pro</td>
</tr>
<tr>
<td>LimeSDR [146]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>High</td>
<td>Low</td>
<td>PC</td>
<td>Intel Cyclone IV</td>
</tr>
<tr>
<td>Ziria [147]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>High</td>
<td>Low</td>
<td>PC</td>
<td>Depends on App</td>
</tr>
<tr>
<td>Sora [18]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>High</td>
<td>Low</td>
<td>PC</td>
<td>Xilinx Virtex-5</td>
</tr>
<tr>
<td>SODA [67]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>High</td>
<td>High</td>
<td>ARM Cortex-M3 + Processing Elements</td>
<td>N/A</td>
</tr>
<tr>
<td>Iris [148]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>High</td>
<td>High</td>
<td>Dual-core ARM Cortex-A9</td>
<td>Xilinx Kintex-4</td>
</tr>
<tr>
<td>Atomix [19]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>High</td>
<td>Medium</td>
<td>TI 6670 DSP</td>
<td>N/A</td>
</tr>
<tr>
<td>BeagleBoard-X15 [159]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>High</td>
<td>Medium</td>
<td>2 x TI C6670DSPs + 2 x ARM Cortex-A15 &amp; 2 x M4</td>
<td>N/A</td>
</tr>
<tr>
<td>Airblue [20]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>High</td>
<td>High</td>
<td>N/A</td>
<td>Intel Cyclone IV</td>
</tr>
<tr>
<td>WARP v3 [21]</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>High</td>
<td>High</td>
<td>2 x Xilinx MicroBlaze cores</td>
<td>Xilinx Virtex-6</td>
</tr>
<tr>
<td>PSoc SLP [164]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Low</td>
<td>High</td>
<td>ARM Cortex-M3</td>
<td>N/A</td>
</tr>
<tr>
<td>Zynq-based [166]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>High</td>
<td>High</td>
<td>Dual-core ARM Cortex-A9</td>
<td>Xilinx Kintex-4</td>
</tr>
</tbody>
</table>

[High cost (200 - 5K USD)]

[General purpose → High power]

[Akeela, 2018]
**Solution: Design of ULP-SDR**

- **Wide-band RF**
  - **Configurable DFE**
  - **ULP SDR**

**Similar requirements in most IoT transceivers (BW < 5MHz)**

- **2.4 GHz (ISM)**
  - or
  - **1.6 GHz (satellite)**
  - or
  - **UWB**
  - or
  - **subGHz (ISM)**

**Differing requirements in most IoT transceivers**

**SDR-based IoT node**

- **MCU**
  - Application
  - Protocol stack
- **MEM**
- **PMU**
- **Sensor I/F**
- **...**

**Heterogeneous multi-core platform**

**Challenge:**

- Target mW-level power consumption
SYSTEM REQUIREMENTS

- **Target Architecture**
  - A very small and fast core (signoff ~300 MHz) associated to a TCPM and TCDM

- **Software DSP limited to decimated sample streams**
  - DFE includes easily configurable and common HW operators: FIR filters, down-converters, AGC...

- **Real-time processing of complex samples**
  - Samples are temporarily stored in sample buffer and processed in blocks
  - Integer processing only

- **Limit size of memory → big impact on power → configurable in size**
  - TCPM (high speed non volatile)
  - TCDM (stack usage !)
  - Sample buffer
  - Limit read/write to TCM

- **Single-cycle sleep**
  - Wait for next block of samples
  - Radio = OFF/ON
• Wireless DSP requires linearity and low distortion
  • Operators **MUST NOT** saturate
  • Operators **MUST NOT** overflow \( \rightarrow \) but checking for overflows is too costly
• Wireless DSP must conserve dynamic range (DR)
  • The useful signal is often contained in the least significant bits
  • Beware of quantification noise \( \rightarrow \) take care when rescaling the signal !
• **Most wireless signals are complex** : \( i(t) + j^*q(t) \)
  • Frequent use of MUL, ADD, SUB, MAG, SHIFT, … instructions on 8/16/32 bit **complex** data
• Demodulation/compensation algorithms are mostly based on correlations
  \( \rightarrow \) i.e. multiplication
• **Input signal stream is typically <= 8 bits**
  • i.e. data streams are typically 8 / 16 / 32 bits
  • \( \rightarrow \) fits well on a 32-bit machine
WHICH PROCESSOR FOR OUR SDR?

- **Academic**: Dedicated processors
  
  **Custom SIMD** [Chen, HPCA16]
  
  - Promising power consumption
  
  - Dedicated architectures → difficult to program
  
  - No software tool-chains
  
  - Low frequency clock → Large surface overheads
  
  - Inefficient use of advanced CMOS nodes

- **Commercial**: GP processors, DSP

  Previous work:
  
  M3/M0+ vs. RISCY
  
  - **Lessons learned**: GP processor can rival dedicated SoA processor architectures (with additional benefits)
  
  - **Lessons learned**: size of register file has huge impact on cycle count
  
  - **Lessons learned**: RISC-V advantage!
  
  - **Lessons learned**: post-increment, HW loop, SIMD → not important in our test benches (mix of DSP computing and control)

  [Belhadj, DATE19]
• RISC-V-based acceleration?

• Extend RISC-V ISA using dedicated instructions
  • Codasip Studio: → An easy task?
  • Instruction Accurate (IA) model of new instructions
  • Dedicated to RF DSP computing “zero cost” hardware implementation
EXPLORING THE INSTRUCTION JUNGLE

• **Wanted**
  - Minimal set of USEFUL instructions.
  - Only 32-bit opcodes for low decoding complexity.

• **Opportunities**
  - Wide opcodes means up to 5 operands!
  - First operation on 8-bit data is ALWAYS a complex multiplication
  - Advanced CMOS allows single-cycle operators
  - Tiny relative cost of ALU operators

---

**REJECTED**

• **More general solution preferred**:
  - Halving variants (e.g. RADD)

• **Not clearly indispensable**:
  - CSMUL (complex-scalar multiply)

• **Useless**:
  - saturating instructions, MIN/MAX, 8 bit SIMD, CONJ

---

45 nm, 0.9 V [M. Horowitz, ISSCC 2014]
PROPOSED EXTENSION

• 15 instructions using 3 major opcodes

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDC16 rd, rs1, rs2, imm</td>
<td>16-bit Addition &amp; Shift Right Arithmetic Immediate</td>
<td>rd.L = (rs1.L + rs2.L)&gt;&gt;imm</td>
</tr>
<tr>
<td>SUBC16 rd, rs1, rs2, imm</td>
<td>16-bit Subtraction</td>
<td>rd.L = (rs1.L - rs2.L)&gt;&gt;imm</td>
</tr>
<tr>
<td>MULC8-16 rd, rs1, rs2, HI, H2, imm</td>
<td>Two &quot;8x8&quot; and Signed Subtraction</td>
<td>if Hx = 1, {tx,qx} = {rsx.B2,rsx.B3}</td>
</tr>
<tr>
<td></td>
<td>Two Crossed &quot;8x8&quot; and Signed Addition</td>
<td>if Hx = 0, {tx,qx} = {rsx.B0,rsx.B1}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rd.L = ((i1 * i2) + (q1 * q2))&gt;&gt;imm</td>
</tr>
<tr>
<td>MULC16 rd, rs1, rs2</td>
<td>Two &quot;16x16&quot; and Signed Subtraction</td>
<td>rd.L = (rs1.L * rs2.L)&gt;&gt;16 - (rs1.H * rs2.H)&gt;&gt;16</td>
</tr>
<tr>
<td></td>
<td>Two Crossed &quot;16x16&quot; and Signed Addition</td>
<td>rd.H = (rs1.H * rs2.H)&gt;&gt;16</td>
</tr>
</tbody>
</table>

- « Zero-cost »
  > Reconfigurable HW
  > Systematic output DR adjust

- « Low-cost »
  > 4 output / 2 input port register file
  > Duplicated ALU

- « Higher-cost »
  > 3 more 32-bit multipliers

Shorthand definitions: r.B3→r[31:24], r.B2→r[23:16], r.B1→r[15:8], r.B0→r[7:0], r.H→r[31:16], r.L→r[15:0]

C. Bernier | October 1, 2019 | 11
Testbench 1: FSK demodulation

Testbench 2: LoRa preamble synchronization
- Spreading Factor (SF) = 7, 11

Testbench 3: 16 and 32-bit FFT
- Radix-4 decimation-infrequency, complex FFT with bit-reversed outputs, N = 128, 2048
- Based on source code from a port of the ARM CMSIS DSP library to RISC-V

Testbench 4: CORDIC algorithm
- 10 iteration CORDIC algorithm applied to 32-bit complex input data.
**RESULTS**

- Expect at least ~50% power reductions with reduced clock and VDD.

<table>
<thead>
<tr>
<th>Testbench</th>
<th>Cycle count improvement (IA model)</th>
<th>Energy improvement (est.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSK Demod</td>
<td>22 %</td>
<td></td>
</tr>
<tr>
<td>LoRa, SF=7</td>
<td>49 %</td>
<td>46 %</td>
</tr>
<tr>
<td>LoRa, SF=11</td>
<td>52 %</td>
<td>50 %</td>
</tr>
<tr>
<td>16-bit FFT, N=128</td>
<td>55 %</td>
<td>53 %</td>
</tr>
<tr>
<td>16-bit FFT, N=2048</td>
<td>57 %</td>
<td>55 %</td>
</tr>
<tr>
<td>32-bit FFT, N=128</td>
<td>34 %</td>
<td>32 %</td>
</tr>
<tr>
<td>32-bit FFT, N=2048</td>
<td>34 %</td>
<td>30 %</td>
</tr>
<tr>
<td>32-bit CORDIC, 10 iteration</td>
<td>28 %</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Power Model</th>
<th>Baseline</th>
<th>+Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>All instr. except NOP and MUL</td>
<td>1</td>
<td>1.05</td>
</tr>
<tr>
<td>MUL</td>
<td>1.14</td>
<td>1.14</td>
</tr>
<tr>
<td>MULC16-32 / MULC16</td>
<td>-</td>
<td>1.3</td>
</tr>
<tr>
<td>MULC32</td>
<td>-</td>
<td>1.59</td>
</tr>
</tbody>
</table>

• Expect at least ~50% power reductions with reduced clock and VDD.
FUTURE WORK

- Finish CA model & run Power/Area analysis in 22 nm
- Reconfigurable hardware blocks designed in CodAL. Ex: 32-bit multiplication
Special thanks to:

Hela Belhadj Amor
Zdeněk Přikryl
Jerry Ardizzone

And

Ivan Miro Panades
Yves Durand
Henri-Pierre Charles
Simone Bacles-Min
Romain Lemaire

… and all of LISAN!
**Step 1: ISA exploration using IA model**

```plaintext
element opc_name
{
  use instance_data_type as name of instances;

  assembler {textual form of the instruction};
  binary {The instruction's binary coding};

  semantics
  {
    The instruction's behavior is described using a subset of the ANSI C language.
  }
}
```

Used by IA and CA models

```
element i_load
{
  use opc_load as opc;
  use gpr_all as gpr_dst, gpr_src1;
  use sim12;

  assembler {opc gpr_dst "," sim12 "(* gpr_src1 *)

  binary {sim12 gpr_src1 opc[OPC_FRAGI] gpr_dst opc[OPC_FRAGI]};

  semantics
  {
    ADDR_TYPE address;
    WORD_TYPE result;

    codasip_compiler_schedule_class(sc_load);

    address = rf_gpr_read(gpr_src1) + sim12;
    result = load(opc, address);
    rf_gpr_write(result, gpr_dst);
  }
}
```

Used by IA model

Call to memory interface if_ldst
RECONFIGURABLE MULTIPLIER (8 BIT EXAMPLE HERE)

State 1: the block performs 8-bit integer multiplication

\[
a[7:0] \times b[7:0] = P[15:0]
\]

\[
p_{00}[7:0] = a[3:0] \times b[3:0]
\]

\[
p_{10}[7:0] = a[7:4] \times b[3:0]
\]

\[
p_{01}[7:0] = a[3:0] \times b[7:4]
\]

\[
p_{11}[7:0] = a[7:4] \times b[7:4]
\]

\[
P[15:0] = p_{00}[7:0] + p_{10}[7:0] \ll 4 + p_{01}[7:0] \ll 4 + p_{00}[7:0] \ll 8
\]
RECONFIGURABLE MULTIPLIER (8 BIT EXAMPLE HERE)

State 2: the block performs a 4-bit complex integer multiplication:

\[(I_1 + jQ_1) \times (I_2 + jQ_2) = \mathbf{P}_{\text{real}} + j\mathbf{P}_{\text{imag}}\]

Input is redefined:

\[
\begin{align*}
I_1[3:0] &= a[3:0] \\
Q_1[3:0] &= a[7:4] \\
I_2[3:0] &= b[3:0] \\
Q_2[3:0] &= b[7:4]
\end{align*}
\]
(I_1 + j^*Q_1) * (I_2 + j^*Q_2) = P_{real} + j^*P_{imag}

\[ P_{real} = I_1^*I_2 - Q_1^*Q_2 \]
\[ P_{real} = p_{00}[7:0] - p_{11}[7:0] \]

\[ P_{imag} = I_1^*Q_2 + Q_1^*I_2 \]
\[ P_{imag} = p_{01}[7:0] + p_{10}[7:0] \]