RISC-V CUSTOMIZATION WITH STUDIO 8

Zdeněk Přikryl
CTO, Codasip GmbH
WHO IS CODASIP

Leading provider of RISC-V processor IP

- Introduced its first RISC-V processor in November 2015
- Offers its own portfolio of RISC-V processors: **Codasip Bk**
- Provides unique design automation tools for easy modification of RISC-V processors

Founding member of RISC-V Foundation → www.riscv.org

- Member of several working groups within the Foundation

Active contributor to LLVM and other open-source projects
CODASIP RISC-V PORTFOLIO

- Broad portfolio of low-power, high-performance processor cores for any design
- All fully compliant with the RISC-V specification
- All fully customizable

Enables our customers to build unique fine-tuned products for competitive advantage

“Codasip and RISC-V enable all the advantages of application tailoring, with the stability and predictability of off-the-shelf Arm designs, while delivering an order of magnitude performance improvement.”

Derek Atkins, CTO, SecureRF
CONFIGURATION VS CUSTOMIZATION

RISC-V offers a wide range of ISA extensions:

- I/E for integer instructions
- M for multiplication and division
- C for compact instruction
- WIP: B, P, V, ...
- and others

**Configuration:**
Selecting multiple ISA extensions
- Enabled by some vendors or open-source projects
- Still insufficient for some application domains

**Customization:**
Adjusting the core to suit your exact needs
STANDARD APPROACH TO CUSTOMIZATION

Manually adding new instructions to the RISC-V ISA:

1. Model and simulate the instruction
2. Modify the compiler
3. Add support in the debugger
4. Write Verilog to implement in hardware
5. Verify, verify, verify, ...

• Challenging and time-consuming
• **Automation** desirable for each of these steps
CODASIP APPROACH TO AUTOMATION

Processor Modeling → Software Analysis → SDK Synthesis → RTL Synthesis → Verification

- Application(s)/Program(s)
  - C/C++ Compiler
  - Assembler
  - Linker
  - IA Simulator, Profiler, Debugger
  - CA Simulator, Profiler, Debugger

- Functional Model
- Implementation Model

- Reference Model
- RTL Models
- UVM Verification
CODASIP APPROACH TO CUSTOMIZATION

Codasip Studio:
- Introduced in 2014
- Silicon-proven by major vendors
- Allows for fast & easy customization of base instruction set:
  - Single cycle MAC
  - Floating point
  - Custom crypto functions
  - and more...

Codasip Studio

<table>
<thead>
<tr>
<th>CodAL – processor description language</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>element i_mac {</code></td>
</tr>
<tr>
<td><code>use reg as dst, src1, src2;</code></td>
</tr>
<tr>
<td><code>assembler { “mac” dst “,” src1 “,” src2 };</code></td>
</tr>
<tr>
<td><code>binary { OP_MAC:8 dst src1 src2 0:9 };</code></td>
</tr>
<tr>
<td><code>semantics {</code></td>
</tr>
<tr>
<td><code>rf[dst] += rf[src1] * rf[src2];</code></td>
</tr>
<tr>
<td><code>};</code></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

---

**Integrated processor development**

**RTL Automation**
- Powerful high-level syntheses
  - Verilog
  - VHDL

**SDK automation**
- Standards-based tools & models
  - LLVM
  - Others

**Verification Automation**
- VSP and processor validation
  - UVM
  - Others

© Codasip Ltd.
CODASIP STUDIO TOOL SUITE

Codasip Studio Toolset

HDK
- RTL models
- Synthesis scripts
- Verification models and simulators
- Virtual prototypes

SDK
- Compiler
- Assembler
- Linker
- Debugger
- IDE

CodAL Models

ISA extensions are quickly implemented and analyzed during design space exploration.

Profiling of embedded application SW enables processor optimizations.

Codasip Studio automatically generates all processor IP design kits and verifies for RISC-V compliance.
CODAL MODELS

Processor IP at a high level of abstraction

Easy-to-understand C-like language

Features:
- Can model a rich set of processor capabilities
- Can implement multiple microarchitectures in a single model

Usage:
- Used to model and verify all Codasip processors
- Provided to Codasip IP customers as a starting point for their processor optimizations and modifications

/* Multiply and accumulate: semantics */
dst += src1 * src2

element i_mac {
    use reg as dst, src1, src2;
    assembler { "mac" dst "," src1 "," src2 };
    semantics {
        rf[dst] += rf[src1] * rf[src2];
    };
}
BENEFITS FOR CUSTOMERS

Performance improvement through high-level optimization:

FIR implementation in C with 200 16-bit input samples and 16 16-bit coefficients

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Clock Cycles</th>
<th>Code size</th>
<th>Speedup Against Base</th>
<th>Area (Gates)</th>
<th>Area Expansion Against Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>1,764,256</td>
<td>232</td>
<td></td>
<td>16.0k</td>
<td></td>
</tr>
<tr>
<td>Base + Serial Multiplier</td>
<td>427,561</td>
<td>148</td>
<td>4.12 x</td>
<td>19.7k</td>
<td>1.24 x</td>
</tr>
<tr>
<td>Base + Parallel Multiplier</td>
<td>133,061</td>
<td>148</td>
<td>13.26 x</td>
<td>26.2k</td>
<td>1.64 x</td>
</tr>
<tr>
<td>Base + DSP Extensions</td>
<td>31,371</td>
<td>64</td>
<td>56.24 x</td>
<td>38.7k</td>
<td>2.43 x</td>
</tr>
</tbody>
</table>

1 Fewer clock cycles → same software takes less time to run.
2 Smaller code size (optimized software) → less memory saves money.
3 More gates in advanced cores → higher cost. Here, only 2x area increase provides 50x performance gain.

© Codasip Ltd.
CASE STUDY: B ISA EXTENSION

Bit manipulation instructions, ca 30:

- Bit insert and extract
- Byte swapping
- Rotations
- Bit swapping/shuffling
- Zero/one counters
- and more...

Not yet ratified – must be implemented as custom extensions
CASE STUDY: FUNCTIONAL MODEL

Written in CodAL

- in 10 days by a single engineer

900 lines of code

Software development kit (SDK) automatically generated by Studio, including:

- C compiler
- Instruction set simulator (ISS)
- Profiler for checking the impact of the extensions

```c
element i_gzip
{
  use opc_gzip as opc;
  use reg_any as dst, src1;
  use shift_imm as imm;
  assembler ( opc dst "," src1 "," imm);
  semantics
  {
    rf_gpr_write (dst, gzip_uXlen(rf_gpr_read(src1), imm));
  }
};
set isa_b += i_gzip;
```

```
uint32 gzip_uXlen (const uXlen rsc1, const uXlen mode)
{
  uint32 zip_mode, x;
  x = rsc1;
  zip_mode = mode & 31;
  if(zip_mode & 1)
  {
    if(zip_mode & 2)
      x = gzip_stage (x, MASK_ZIP2_L, MASK_ZIP2_R, 1);
    if(zip_mode & 4)
      x = gzip_stage (x, MASK_ZIP4_L, MASK_ZIP4_R, 2);
    if(zip_mode & 8)
      x = gzip_stage (x, MASK_ZIP8_L, MASK_ZIP8_R, 4);
    if(zip_mode & 16)
      x = gzip_stage (x, MASK_ZIP16_L, MASK_ZIP16_R, 8);
  }
```
CASE STUDY: IMPLEMENTATION MODEL

Written in CodAL

• in 3 weeks by a single engineer

1500 lines of code

Hardware development kit (HDK) automatically generated by Studio, including:

• RTL
• Test bench
• UVM-based verification environment

```c
#ifdef OPTION_EXTENSION_B
  case SLO:
    ex_result = ones_shifter_32(SLO, ex_aluop1, ex_aluop2);
    break;
  case SRO:
    ex_result = ones_shifter_32(SRO, ex_aluop1, ex_aluop2);
    break;
  case ANDC:
    ex_result = (uxlen)ex_aluop1 & (~ ex_aluop2);
    break;
  case ROTR :
    ex_result = ex_aluop1 >>> ex_aluop2;
    break;
  case ROTL :
    ex_result = ex_aluop1 <<< ex_aluop2;
    break;
  case CTZ :  
    ex_result = codasip_ctlz_uint32(exaluop1);
    break;
  case CLZ:
    ex_result = codasip_cttz_uint32(exaluop1);
    break;
#endif
```
VERIFICATION

- Consistency checker
- Random assembler program generator
- UVM Verification Environment
  - For checking that RTL corresponds to specification (in this case, IA model definition)
  - Environment in SystemVerilog generated automatically from Codasip Studio
NEW: **STUDIO 8 TOOL SUITE**

- Support for LLVM debugger (LLDB) and OpenOCD
- LLVM 7.0
- Studio/CodeSpace IDEs based on Eclipse Oxygen along with more interactive consoles
- Improved test suites and verification to better support user-defined RISC-V extensions
- New I and D caches
- Improved tracing capabilities

And many more smaller or bigger improvements here and there...
CODASIP STUDIO 8.0.0 – LLVM

- Support for LLVM debugger (LLDB) and OpenOCD
- LLVM 7.0
CODESIP STUDIO 8.0.0 – ECLIPSE-BASED IDE

✓ Studio/CodeSpace IDEs based on Eclipse Oxygen along with more interactive consoles
NEW: **BK7** RISC-V PROCESSOR

Newest, fully customizable micro-architecture offering

- 7-stage pipeline, in-order execution
- Branch prediction
- 64-bit addressing
- Linux capable
- Instruction and data caches
- IEEE 1149.7-compliant debug
- Support for standard RISC-V extensions: I, M, F, C, A, ...
CONCLUSION

Off-the-shelf RISC-V processors
• Rich selection
• Ready to be used immediately

Configurability
• Useful, but often not sufficient

Customization
• When the best PPA for your application domain is required

Codasip offers an easy, automatized way to add your secret sauce:
• Custom ISA extensions
• Microarchitectural improvements

Example: B ISA extension, supported by SDK and RTL, done in a couple of weeks.
QUESTIONS?

Thank you.