Practical usage of Configurable DSPs in High Performance Systems

Marcus Binning, Application Engineering Director
High Performance Digital Systems Event, Sept 13th 2017
University of Warwick
The Case for Configurable, Custom Cores

• People pay for features / capabilities
  – Rarely for implementations
• Easiest, fastest, most flexible way to deploy features – Software
  – E.g. advanced imaging in cell phone, video stabilisation, voice control – valuable feature$
• We live in an energy-constrained world – the software must run on extremely power efficient platforms (cores, memory systems)
  – From always-on triggers to highest performance datacentre...
• We live in a Time-Constrained world – rate of design has increased dramatically in the last few years
• One size does NOT fit all – what is optimal for large applications is not optimal for audio, communications, imaging, classification, voice triggering …
  – It’s a HETEROGENEOUS, “OFFLOAD-ENABLED” world …
  – Oh, and we need it NOW ..
What affects the efficiency of Software?

• Cycle counts – the higher the longer it takes to complete
  – Targeting the right instructions, number representations crucially important

• Sufficient local register storage
  – Access to registers very low energy cost. Insufficient registers can be a major efficiency loss

• Efficient Memory System
  – This varies greatly from system to system
  – With large software sets, a sophisticated cache based or DMA based system may be appropriate
    – Support for Prefetch of Instructions and/or Data may be beneficial
  – With self-contained very small, low power systems the complete opposite may apply.
    – Local tightly coupled memory may be appropriate

• Connecting accelerators – place on the bus or go point-to-point?

• Efficient Architecture must span all these
  – (and more, of course – debug, trace ..)
Comparative System View
Transferring Data from a HW engine to DSP

- Classic DMA based scheme, using AXI Slave port on the DSP
  - Note - could also use integrated DMA on P5 - same performance
- Background task - should not interfere with current processing
- Can be limited by AXI width and clock rate (128b)

Utilise Input Queue and FIFO to interface to external engine
- Queue input can be wider - e.g. same width as register file (512b)
  - Minimum 4x raw bandwidth compared to “full rate” AXI fabric
- Queue can be accessed every cycle by instructions - greater bandwidth
# Application Targeted Processor Portfolio from Cadence

## Broad Range of Application Specific DSPs

<table>
<thead>
<tr>
<th>HiFi DSPs</th>
<th>Fusion DSPs</th>
<th>ConnX DSPs</th>
<th>Vision DSPs</th>
<th>Custom ISAs</th>
</tr>
</thead>
<tbody>
<tr>
<td>![HiFi DSPs Image]</td>
<td>![Fusion DSPs Image]</td>
<td>![ConnX DSPs Image]</td>
<td>![Vision DSPs Image]</td>
<td>![Custom ISAs Image]</td>
</tr>
<tr>
<td>• Audio Pre and Post-Processing</td>
<td>• Auto Radar</td>
<td>• Narrow to wide band Wireless</td>
<td>• Image Pre-/Post-Processing</td>
<td>• High Performance DSPs, NPUs, CPUs</td>
</tr>
<tr>
<td>• Voice trigger</td>
<td>• Always-alert Sensor processing</td>
<td>• LTE/LTE-A/5G</td>
<td>• Convolutional Neural Networks (CNN)</td>
<td>• Application specific data types</td>
</tr>
<tr>
<td>• Noise Reduction</td>
<td>• Low-end Imaging</td>
<td>• WiFi, Smart Grid</td>
<td>• AR/VR</td>
<td>• Custom ISA</td>
</tr>
<tr>
<td>• Audio Encode &amp; Decode</td>
<td>• Audio, Video and Speech</td>
<td>• Infrastructure &amp; Terminals</td>
<td>• ADAS</td>
<td>• Special Functions</td>
</tr>
</tbody>
</table>

## Automated User-Defined Customization

- Interfaces, Instructions, State & Registers, Unique and Secure features

## Xtensa® Processor Generator

- Configurable and Extensible Common Foundation Technology of all Tensilica Processors
Example – Computer Vision
Emerging use cases: Cameras everywhere

- Mobile: HDR, Video Stabilizer, Face Detection
- Automotive: Traffic Sign Recognition, Gesture Control
- Drone: 3D Vision
- Security: People Detection
- Wearables and IoT: 3D Vision
The basics of real-time neural networks

**Training**: Runs once per database, server-based, very compute intensive

- **Server Farm**: Labeled dataset
- **Selection of layered network**
- **Iterative derivation of coefficients by stochastic descent error minimization**
- **Set of coefficients (1M-1B weights)**
- **10^{16}-10^{22} MACs/dataset**

**Deployment (“Inference”)**: Runs on every image, device based, compute intensive

- **Embedded**: Single-pass evaluation of input image
- **Most probable label**
- **10^8-10^{12} MACs/image**
Tensilica® Vision P6 DSP for Computer vision/imaging/CNN
Tensilica® Vision C5 DSP for Neural Networks

Vector Processing Units
- 1024 MACs
- 128-Way SIMD VLIW Architecture
- On-the-Fly Decompression
- Flexible ISA for Quantization

Timers, Interrupts, Performance Counters
Power Management
Pipeline Management
Instruction Decoder
Cache Controller
Scalar Processing Units
Scalar Register File

Data Memory 0
Bank 0
Bank 1
512

Data Memory 1
Bank 0
Bank 1
512

Memory Mux
Load
Load/Store
Custom Instructions
Vector Register File (32)
Accumulator Register File (16)

iDMA

AXI4 Interface
128

AXI4 Interface
128

Instruction Memory 1
128

Instruction Memory 2
128

Instruction Cache
128
Bringing Efficient *Programmability* to any Design

**Tensilica Processor**

**CPU Strengths**
- General Purpose Control
- Easy C Programming
- Real-time Operating System Support

**Custom Strengths**
- Task-specific, Virtually unlimited I/O, Arbitrary computations, Differentiation,

**DSP Strengths**
- SIMD, VLIW, Vectorization
- N-way programming, Fixed and Floating Point

Optimized Instruction Set, Data Path and I/O Interfaces
One development flow with C programmability and full debug
Example – high performance programming
Source Code - popcount
Counts the number of '1's in a series of 64bit vectors

```c
static void popcount_v2(char *__restrict__ resPtr, int *__restrict__ srcPtr, int words)
{
    int i;
    x_bvec2nx8U *vbP = (x_bvec2nx8U *) srcPtr;
    x_bvec2nx8U *pP = (x_bvec2nx8U *) resPtr;
    x_bvec2nx8U ptL, ptH;
    x_bvec2nx80 vecSel1 = *(x_bvec2nx80*)selMatrixA1;
    x_bvec2nx80 vecSel2 = IVP_SEQ2NX8U();

    ptL = *(x_bvec2nx80 *) &poptable[0];
    ptH = *(x_bvec2nx80 *) &poptable[64];

    for (i = 0; i < words / 64; i++) {
        x_bvec2nx8U vec0, vec1, vec2, vec3, vec4, vec5, vec6, vec7;
        x_bvec2nx16U vec8, vec9, vec10, vec11, vec12, vec13, vec14, vec15;

        // compute byte pop count for 8 vectors
        // vec0
        vec0 = *vbP++;
        vec0 = IVP_SEQ2NX8U(ptH, ptL, vec0) + (vec0 >> 7);
        // vec1
        vec1 = *vbP++;
        vec1 = IVP_SEQ2NX8U(ptH, ptL, vec1) + (vec1 >> 7);
        // vec2
        vec2 = *vbP++;
        vec2 = IVP_SEQ2NX8U(ptH, ptL, vec2) + (vec2 >> 7);
        // vec3
        vec3 = *vbP++;
        vec3 = IVP_SEQ2NX8U(ptH, ptL, vec3) + (vec3 >> 7);
        // vec4
        vec4 = *vbP++;
        vec4 = IVP_SEQ2NX8U(ptH, ptL, vec4) + (vec4 >> 7);
        // vec5
        vec5 = *vbP++;
        vec5 = IVP_SEQ2NX8U(ptH, ptL, vec5) + (vec5 >> 7);
    }

    // partial sum
    IVP_SEQ2NX8U(vec9, vec8, CVT_8U_16(vec1), CVT_8U_16(vec0), vecSel1);
    vec10 = vec8 + vec9;
    IVP_SEQ2NX8U(vec9, vec8, CVT_8U_16(vec3), CVT_8U_16(vec2), vecSel1);
    vec11 = vec8 + vec9;
    IVP_SEQ2NX8U(vec9, vec8, CVT_8U_16(vec5), CVT_8U_16(vec4), vecSel1);
    vec12 = vec8 + vec9;
    IVP_SEQ2NX8U(vec9, vec8, CVT_8U_16(vec7), CVT_8U_16(vec6), vecSel1);
    vec13 = vec8 + vec9;

    // partial sum 2
    IVP_SEQ2NX8U(vec9, vec8, vec10, vecSel2);
    vec14 = vec8 + vec9;
    IVP_SEQ2NX8U(vec9, vec8, vec12, vecSel2);
    vec15 = vec8 + vec9;

    // partial sum 3
    IVP_SEQ2NX8UB(vec1, vec8, CVT_16_8U(vec15), CVT_16_8U(vec14),
                   IVP_EL1_88_DEINTERLEAVE_1);
    vec0 += vec1;

    // store 64 results
    *pP++ = vec0;
}
```
• Declaration uses standard 'C' types
  ➢ Makes it easy to test with pure 'C' implementations - any cast to machine specific types done inside function
  ➢ Use of __restrict same as normal 'C' (== no pointer aliasing)

• New vector types specific to Vision P5
  ➢ Type xb_vec2Nx8U is a vector of 8bit unsigned quantities
  ➢ In this machine 'N' == 32 so vbP is a pointer to 64Byte items

• Compiler is 'taught' through TIE how to do pointer casting, arithmetic etc

• Declare some local variables
  ➢ Scope rules exactly the same as 'C'
  ➢ Type xb_vecNx16U is a 32-way vector of 16bit "unsigned" values
  ➢ In this machine there are 32 vector registers, so having lots of "live" variables will not cause spills
  ➢ Compiler "knows" how to align these vector variables correctly

• Vector instructions inferred by the compiler
  ➢ In this case vec0 is an array of 8bit unsigned values
  ➢ Therefore in this case compiler will infer a vector LOGICAL right shift (not ARITHMETIC).

• The "+" operator in this case means "Vector add"
  ➢ Because both "types" are unsigned 8bit vectors, ad 64-way unsigned add will be inferred by the compiler
Source Code - Explained (part 2)

```c
// vec6
vec6 = *vbP++;
vec6 = IVP_SEL2NX8U(ptH, pt1, vec6) + (vec6 >> 7);
// vec7
vec7 = *vbP++;
vec7 = IVP_SEL2NX8U(ptH, pt2, vec7) + (vec7 >> 7);

// partial sum 1
IVP_DSELNX16(vec0, vec8, CVT_BU_16(vec1), CVT_BU_16(vec8), vecSel1);
vec10 = vec8 + vec9;
IVP_DSELNX16(vec0, vec8, CVT_BU_16(vec3), CVT_BU_16(vec2), vecSel1);
vec11 = vec8 + vec9;
IVP_DSELNX16(vec0, vec8, CVT_BU_16(vec5), CVT_BU_16(vec4), vecSel1);
vec12 = vec8 + vec9;
IVP_DSELNX16(vec0, vec8, CVT_BU_16(vec7), CVT_BU_16(vec6), vecSel1);
vec13 = vec8 + vec9;

// partial sum 2
IVP_DSELNX16(vec0, vec8, vec11, vec10, vecSel2);
vec14 = vec8 + vec9;
IVP_DSELNX16(vec0, vec8, vec12, vec10, vecSel2);
vec15 = vec8 + vec9;

// partial sum 3
IVP_DSELNX8U(vec1, vec6, CVT_16_BU(vec15), CVT_16_BU(vec14),
               IVP_DSEL168_REINTERLEAVE1);
vec0 ++ vec1;
// store 64 results
"rP++ = vec0 ;
```
Benchmark and Analyze
Use Xplorer to Interactively Check Performance As You Build

Check boxes to select pre-designed options

Profile your software to see the critical loops and what operations are used most

Instantly view PPA as you select options

Analyze performance bottlenecks with pipeline view
Debugging

Displays Processes & Call Stacks

Source File

Breakpoint Set/Clear

Views for:
- Variables
- Expressions
- Registers
- Breakpoints
- TIE wires

Views for:
- Console
- Problems
- Memory

Debug Action Buttons

Resume
HW Sync
Pause
Step Into
Step Over
Step Return
Instruction Stepping Mode
HW Disconnect
Terminate
Step Return
Automation – the key to making it happen
Automated Tool, ISS, Model, RTL, and EDA Script Generation...

Base Processor
Dozens of Templates for many common applications

Pre-Verified Options
Off-the-shelf DSPs, interfaces, peripherals, debug, etc.

Optional Customization
Create your own instructions, data types, registers, interfaces

Complete Hardware Design
Pre-verified Synthesizable RTL EDA Scripts Test suite…

Tensilica IP
Iterate in minutes!

Processor Generator

Customer IP

Advanced Software Tools
IDE
C/C++ Compiler
Debugger
ISS Simulator
SystemC Models
DSP code libraries
Simple Click Box Configurability

• From fine tuning of performance, power & area…
  – Size, type, width & latency of memories
  – Optional pre-fetch unit, extra registers…
  – Load/Store unit characteristics
  – Number of general purpose registers
  – Number and priority levels of interrupts

• High-level application specific blocks
  – Scalar or Vector Floating point, multiplier, divider…
  – FLIX: Flexible length Instruction Extensions

• Select DSP ISAs..
  – HiFi DSPs for Audio, Voice, and Speech
  – ConnX BBE16/32/64EP DSPs for Baseband
  – Vision DSPs for Imaging and Computer Vision and CNN
  – Fusion DSPs for multi-purpose DSP, IoT, and Wearables
Extensibility Customize To Your Task

• Simple Verilog-like language you can define…
  – Input/output queues and ports
  – Custom register files
  – Fast lookup tables and local memories
  – Simple single-cycle instructions
  – Multi-cycle instructions
  – SIMD for vectorization
  – FLIX for grouping parallel operations into one instruction

Example: Define a single-cycle instruction

`pop_count` counts the “1”s in a 32-bit register by adding the bits together. This simple Verilog-like code is all it takes to create both the pre-verified adder RTL (175 gates) and the instruction

```verilog
operation pop_count { out AR co, in AR ci[]}{}
  wire [5:0] sum = a0 + a1 + a2 + a3;
  assign co = {26’b0, sum};
}
```

Example; Define I/O Queues

Create three 256 bit queues and an “add” operation:

```verilog
queue inA 256 in
queue inB 256 in
queue outC 256 out
operation ADD XFER {} { inA, inB, out outC} {
  assign outC = inA + inB;
}
```

High throughput without using system bus: 64 bytes in and 32 bytes out per operation

10x speedup: this simple instruction takes it down to just one cycle;
Best hand-coded ASM using standard instructions takes >10 cycles
Xtensa Fully Automated Hardware and Software Tools Generation

Custom Instructions (optional)  Set configuration options (optional)  Choose processor template

Xtensa Processor Generator

Xtensa Processor Generator Outputs

Hardware
- EDA scripts
- RTL

System Modeling / Design
- Instruction Set Simulator (ISS)
- Fast Function Simulator (TurboXim)
- XTSC SystemC System Modeling
- XTMP C-based System Modeling

Software Tools
- Xplorer IDE
- GNU Software Toolkit (Assembler, Linker, Debugger, Profiler)
- Xtensa C/C++ (XCC) Compiler
- C Software Libraries
- Operating Systems

Application Source C/C++
- Compile
- Executable
- Profile using ISS
- Optimize configuration - or - Develop custom instructions

System Development

Software Development

System Development

To Fab / FPGA

Synthesis
- Block Place & Route

Verification
- Chip Integration / Co-verification

EDA scripts

Hardware

RTL

RTL
Comprehensive System Software

Customer Application
- Init and Control
- Interrupt Handlers
- Register Access & Control
- Loadable Library
- Program Overlays
- Remote Debug

Customer libraries

Hardware Abstraction Layer (HAL)

XTOS
Single-Thread Executive

XOS
Multi-Thread Real Time Kernel

Library Loader
AXOM Automatic Overlay Manager
XMON Debug Monitor

Processor

Xtensa Software

© 2017 Cadence Design Systems, Inc.
Summary

• Configurable Processor Technology allows you to get from concept..
• … to implementation …

• Rapidly
• With a rich set or Software tools
• And Models
• In a short amount of time
• With a low engineering effort.