A Software Configurable Processor Architecture

Ricardo E. Gonzalez
Embedded System Design Dilemma

- COMPUTE PERFORMANCE
- SYSTEM TIME-TO-MARKET

- ASSP
- GPP
- FPGA
- DSP
- MEDIA
- SECURITY
- ETC.
- SW
- HW
- ASIC
Solving Compute Intensive Problems

- Processor with embedded programmable logic
  - Lots of compute resources

- Utilize fabric by extending the ISA
  - Design your own FU
  - Add processor state

- Simple programming model
  - Issue instructions to perform computation

- Tailored for high-throughput applications
  - Compute intensive
Compute Intensive Markets & Applications

- Sonar/Radar
- Biometrics
- CAT Scan
- Ultrasound
- Printers
- Scanners
- Broadcast Equipment
- Audio Studio
- Encryption/Decryption
- Network Security
## EEMBC Telemark

<table>
<thead>
<tr>
<th>Processor</th>
<th>EEMBC Telemark Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stretch S5000 - 300 MHz - C OPT</td>
<td>629</td>
</tr>
<tr>
<td>Intrinsity FastMATH - 2 GHz - ASM OPT</td>
<td>688</td>
</tr>
<tr>
<td>TI C6416 - 720 MHz - ASM OPT</td>
<td>379</td>
</tr>
<tr>
<td>TI C6416 - 720 MHz - C OPT</td>
<td>275</td>
</tr>
<tr>
<td>BOPS Manta v2.0 - 1GHz - C OPT</td>
<td>181</td>
</tr>
<tr>
<td>Motorola MPC7447 - 1.3GHz - AltiVec OPT</td>
<td>164</td>
</tr>
<tr>
<td>Motorola MPC7455 - 1GHz - AltiVec OPT</td>
<td>122</td>
</tr>
<tr>
<td>TI C6203 - 300 MHz - ASM OPT</td>
<td>45</td>
</tr>
<tr>
<td>TI C6203 - 300 MHz - C OPT</td>
<td>40</td>
</tr>
<tr>
<td>LSI 402ZX - 200 MHz - ASM OPT</td>
<td>37.7</td>
</tr>
<tr>
<td>Motorola MPC8245-400</td>
<td>28.3</td>
</tr>
<tr>
<td>Motorola MPC755-400</td>
<td>19.5</td>
</tr>
<tr>
<td>IBM 440GPI-500</td>
<td>15.3</td>
</tr>
<tr>
<td>IBM PowerPC 750CX - 500</td>
<td>13.6</td>
</tr>
<tr>
<td>MIPS 20Kc 600 MHz</td>
<td>13.3</td>
</tr>
<tr>
<td>Motorola PowerPC 7400-500</td>
<td>12.7</td>
</tr>
<tr>
<td>IBM 440GP-500</td>
<td>11.4</td>
</tr>
<tr>
<td>MPC Sierra RM7000C - 625MHz</td>
<td>11.2</td>
</tr>
<tr>
<td>IBM PowerPC 7400</td>
<td>10.2</td>
</tr>
<tr>
<td>Motorola PowerPC6230-300</td>
<td>10.2</td>
</tr>
<tr>
<td>IBM 440GP-500</td>
<td>9.5</td>
</tr>
<tr>
<td>AMD K6-III+ 550/ACR</td>
<td>8.7</td>
</tr>
<tr>
<td>AMD K6-2E+ 500/ACR</td>
<td>8.3</td>
</tr>
<tr>
<td>IBM 405GPr - 400 MHz</td>
<td>7.9</td>
</tr>
<tr>
<td>NEC VR5500 - 400</td>
<td>7.1</td>
</tr>
<tr>
<td>TI TMS3226203-300</td>
<td>6.8</td>
</tr>
<tr>
<td>AMD K6-2E 400/ACR</td>
<td>6.8</td>
</tr>
<tr>
<td>AMD 640GPr - 600 MHz</td>
<td>6.3</td>
</tr>
<tr>
<td>AMD 640GPr - 360 MHz</td>
<td>6.3</td>
</tr>
<tr>
<td>NEC VR5500 - 300</td>
<td>5.8</td>
</tr>
<tr>
<td>AMD 640GPr - 266 MHz</td>
<td>5.8</td>
</tr>
<tr>
<td>Infineon Carmel 10xx-170</td>
<td>5.6</td>
</tr>
<tr>
<td>IBM 405GPr - 266 MHz</td>
<td>5.5</td>
</tr>
<tr>
<td>Stretch S5000 - 300 MHz</td>
<td>5.4</td>
</tr>
<tr>
<td>NEC VR5500 - 250</td>
<td>5.3</td>
</tr>
<tr>
<td>LSI 4022ZX - 200</td>
<td>3.3</td>
</tr>
<tr>
<td>IBM 405GPr - 266 MHz</td>
<td>3.3</td>
</tr>
<tr>
<td>NEC VR5500 - 250</td>
<td>3.3</td>
</tr>
<tr>
<td>IDT779RC6457S-250</td>
<td>2.8</td>
</tr>
<tr>
<td>Toshiba TMPR4927ATB-200</td>
<td>2.8</td>
</tr>
</tbody>
</table>

### Embedded Microprocessor Benchmark Consortium
- EEMBC creates, certifies and publishes performance benchmark results

### S5000 Performance With ISEF Acceleration
Application Acceleration Process
RGB → YC<sub>b</sub>C<sub>r</sub>

**Color Conversion Function:**

```c
SE_FUNC /* Tells Stretch C-Compiler to reduce this function to an instruction */
Void RGB2YCBCR (WR A, WR *B) {
    char Y[4], Cb[4], Cr[4];
    char R[4], G[4], B[4];

    R[0] = A(23,16); G[0] = A(15,8); B[0] = A(7,0);
    /* and so on. . . */

    for (i = 0; i < 4; i++) {
        Y[i]  = (77*R[i] + 150*G[i] + 29*B[i]) >> 8;
        Cb[i] = (32768 - 43*R[i] - 85*G[i] + (B[i] << 7)) >> 9;
        Cr[i] = (32768 + (R[i] << 7) - 107*G[i] - 21*B[i]) >> 9;
    }
    *B = (Y[3],Cr[3]+Cr[2],Y[2],Cb[3]+Cb[2],Y[1],Cr[1]+Cr[0],Y[0],Cb[1]+Cb[0]);
}
```

**Program Loop:**

```c
for (...) {
    WRGET0(&A, 12);        /* Load 12 bytes (4 RGB pixels) */
    RGB2YCBCR(A, &B);     /* Convert 4 pixels */
    WRPUT0(B, 8);         /* Store 8 bytes (4 YCbCr pixels */
}
```
Programming Flow

APPLICATION C/C++ → ANNOTATED C/C++

Designer → Stretch C-Compiler → EXECUTABLE

© 2005 Stretch Inc. All rights reserved.
Development & Debug

- Programmers develop & debug using familiar tools
- Faster debug cycle
- Port & accelerate apps within Stretch IDE
- No hardware design experience required!
Stretch S5 Development Flow

Start

C/C++

Application code

Compile

Execute

Correct output?

Fast enough?

Y

N

Y

N

Done

Gnu-based Optimizing compiler

Target simulator or chip

GUI Debugger

Modify application

Create or Modify Extension
Instructions

Select hot-spot

Profile

Easy to use GUI

One EI does the work of many operations and iterations

C/C++

Gnu-based Optimizing compiler

Target simulator or chip

GUI Debugger

Modify application

Create or Modify Extension Instructions

Select hot-spot

Profile

Easy to use GUI

One EI does the work of many operations and iterations
Under The Hood
S5 Datapath and Key Parameters

Wide Register File
- 32 128-bit entries

Load/store unit
- 128-bit load/store
- Auto increment/decrement
- Immediate, indirect, circular
- Variable-byte load/store
- Variable-bit load/store

ISEF
- 3 inputs and 2 outputs
- Pipelined, interlocked
- 32 16-bit MACs and 256 ALUs
- Bit-sliced for arbitrary bit-width
- Allow control logic
- Allow internal processor state

RISC Processor
- Tensilica – Xtensa V
- 32 KB I & D Cache
- On-Chip Memory, MMU
- 24 Channels of DMA, FPU
Instruction Set Architecture

- Based on Xtensa ISA
- Single opcode space
  - Different for each application
- Hardware “caches” EI subset
  - # instructions not limited by hardware

Xtensa V

WR loads/stores

El’s

Always present

Cached set

App 1

App 2
Matching Operating Frequency

- Two isochronous clock domains
  - Processor runs @ higher MHZ
  - Programmable clock ratio (1:1→1:4)
  - WR acts as gateway between them
  - Constraints EI issue rate

- Clock ratio invisible to programmer

- Table-driven EI decode
S5 Pipeline

- Standard 5-stage RISC pipeline
- Extension instruction may be much longer
  - Fully pipelined
- Compiler schedules all instructions
- Hardware interlocks ensure correctness
  - Instruction groups subdivided by use/def requirements
  - Interlocks driven by config table in Instruction Unit
Pipeline Schedule

- Extension Instructions may have different latencies
- Initiation interval of EI a function of:
  - ISEF clock ratio
  - WR and state use/def
- Optimal schedule typically has initiation interval > 1
  - No performance penalty for lower clock rate!
- Statically scheduled
  - Pipeline behavior is completely known at compile time
ISEF Architecture

- Array of 64-bit ALUs
  - May be broken at 4-bit or 1-bit boundaries
  - Conditional ALU operation: \( Y = C ? (A \ op_1 B) : (A \ op_2 B) \)
  - Registers to implement state, pipelining
- Array of 4x8 multipliers, cascadable to 32x32
  - Programmable pipelining
- Programmable routing fabric
- Execution clock divided from processor clock: 1:1, 1:2, 1:3, 1:4
- Up to 16 instructions per ISEF configuration
- Many configurations per executable
- Reconfiguration
  - User directed or on demand
  - \(~100 \mu s\)ec for complete reconfiguration
Dynamic ISEF Configuration

Two ISEFs
- Each holds up to 16 instructions
- Configured via system bus
- Managed by DMA
- Independently configurable

On-Demand Configuration
- Auto managed by HW and OS
- Handled like cache miss

Configuration Preloading
- Managed by application
- Handled like instruction pre-fetch

© 2005 Stretch Inc. All rights reserved.
ISEF Bitstream Generation

C/C++ → cc → xt-xcc → linker → ELF

cc → opt → map → place → route → retime → bitstream
C Compiler Tradeoffs
Examples of Extension Instructions
RGB2YCbCr: SCP Instruction Efficiency

RGB 96 bits

YCbCr 64 bits

© 2005 Stretch Inc. All rights reserved.
Flexibility in FIR Implementations

- Up to 32-MAC/Cycle
- Flexibility to trade-off ISEF usage vs. compute speed
- Wide Load & Stores feed the compute array
Sample Applications
H.264 Encoder Challenges

Source Picture

Frame Store

ME

MC

Forward Transform

Quantization

CABAC or VLC

Deblocking Filter

Select Intra Pred

Intra Predict

Inverse Transform

Inverse Quantization

Compute Intensive Tasks
H.264 Encode

- H.264 Encoding Acceleration
- Optimizes multiple-DSP designs
- Four man-months to port algorithms from C/C++

DIAGRAM:
- SD, 30 FPS, MAIN-PROFILE
- 1750%
- 250%
- 350%
- 1150%
- 100%

- ORIGINAL C
- ACCELERATED C
802.16d (WiMAX) Challenges
802.16d Transmit (PHY)

- Complex WiMax MAC & PHY processing
- Single-chip acceleration replaces DSPs & FPGAs
- Four man-months to port algorithms from C/C++

3.5 MHz, 64 QAM, 3/4 RATE
TDD 50% TRANSMIT, 50% RECEIVE

© 2005 Stretch Inc. All rights reserved.
802.16d Receiver (PHY)

- Complex WiMax MAC & PHY processing
- Single-chip acceleration replaces DSPs & FPGAs
- Four man-months to port algorithms from C/C++
Summary

- C/C++ Programmers can tailor the processor for their application
  - Continue to develop & debug in a familiar environment (IDE/gdb)

- Application-specific instruction provide 10-100X speedup
  - Modest porting effort

- C/C++ programmers can easily design high-performance systems
  - Enabling a new system design methodology