SH-5: A First 64-bit SuperH Core with Multimedia Extension

Fumio Arakawa
Hitachi, Ltd.
SuperH Roadmap

- **SH-5**: First 64-bit Architecture
  - SHmedia ISA, 64-bit, SIMD
  - FPU, Superscalar
  - MMU
  - SH3-DSP
  - 480 MIPS
  - 1.9 GFLOPS
  - SH-4
  - 260 MIPS
  - 400 MOPS
  - SH-3
  - 78 MIPS
  - 120 MOPS
  - SH-2
  - SH-1
  - 20 MIPS
  - 24 GOPS
  - 0.7 GIPS
  - 7 GFLOPS
  - 24 GOPS

- **SH-6**:
  - >2 GIPS
  - >7 GFLOPS
  - >24 GOPS

- Continuously increasing Performance and integration
SH-5 Target Markets

**Consumer Market**
- Digital Home Appliances
  - Digital TV, Set-Top-Box
  - Network
- Car Information Systems
  - Navigation System
  - Telematics, ITS

**Balancing Needs**
- **Low Price**
  - Small die & code size
  - System-on-chip
- **Low Power**
  - Low cost package
  - No-fan system
- **High Performance**
  - 64-bit architecture, SIMD, Vector FPU
  - 7-stage superpipeline
SH-5 Specification

Supply Voltage: 1.5 V

Operating Frequency: 400 MHz

Cache: I/D 32/32 KB (4-way set-associative)

TLB: I/D 64-entry (full-associative)

SuperHyway (Internal Standard Bus)
  - 64-bit, 200 MHz, Split-transaction, 3.2 GB/s

Performance
  - Dhrystone: 714 MIPS (v1.1) and 604 MIPS (v2.1)
  - Peak SIMD: 9.6 GOPS (8 bit) and 1.6 MMACS (16 bit)
  - Peak Floating-point: 2.8 GFLOPS
1st Cut of SH-5 (Evaluation Chip)

- SH5 core
- Memory Interface:
  - EMI (DDR-SDRAM)
  - FEMI (SRAM, Flash)
- Debug
- PCI, Serial, etc.

IFU: Instruction Fetch Unit
IMU: Integer Multimedia Unit
LSU: Load Store Unit
BIU: Bus Interface Unit
EMI: External Memory Interface
FEMI: Flash EMI
Superpipeline

**SH-4**
- 5-stage pipeline

**SH-5**
- 7-stage pipeline
- x1.5 Higher MHz
- x1.5 Longer Latency is Hidden
  - Rich register states hide execution time like load latency
  - Split-branch architecture and target preload hide branch latency

<table>
<thead>
<tr>
<th>Branch Latency (I-cache + Decode)</th>
<th>Load Latency (ALU+ D-cache)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td>2 cycles</td>
<td>2 cycles</td>
</tr>
</tbody>
</table>

| SH-5 |
| F1 | F2 | D | E1 | E2 | E3 | W |
| 3 cycles | 3 cycles |

F,F1,F2: Instruction Fetch; D: Instruction Decode
E1,E2,E3: Execution; W: Write Back
### Rich Register States

<table>
<thead>
<tr>
<th>64 x 64-bit General-purpose Registers</th>
<th>64 x 32-bit Floating-point Registers</th>
<th>64 x 64-bit Control Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>R63 (zero)</td>
<td>R63</td>
<td>CR63</td>
</tr>
<tr>
<td>R19 (T)</td>
<td>R32 (FPUL)</td>
<td></td>
</tr>
<tr>
<td>R18 (PR)</td>
<td></td>
<td>CR0 (SR)</td>
</tr>
<tr>
<td>R17 (MACH,MACL)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R16 (GBR)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R0</td>
<td>R0</td>
<td></td>
</tr>
<tr>
<td>TR7</td>
<td>FPSCR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 8 x 64-bit Target Registers

- TR7
- TR0

#### Floating-point Status and Control Register

- FPSCR

#### Program Counter

- PC

---

SHcompact Registers mapped on SHmedia Registers

---

The 46 reserved CRs are not implemented.
Split Branch Architecture

Prepare Target Instructions

- PTA/l Label, TRa \quad (TRa = &Label )
- PTABS/l Rn, TRa \quad (TRa = Rn )
- PTREL/l Rn, TRa \quad (TRa = Rn+PC )

\( l=L/U: \) likely/unlikely preload is useful

Branch Instructions (Examples)

- BLINK TRb, Rd \quad (Rd=PC+4; \quad PC=TRb)
- BEQ/l Rm, Rn, TRc \quad (if(Rm==Rn) \quad PC=TRc)

\( l=L/U: \) likely/unlikely taken

Static prediction with likely bit
Compare and correct prediction miss

TRa, TRb, TRc: Target Registers; \quad PC: Program Counter
Rm, Rn, Rd: General-purpose Registers
No Branch Overhead

In case of three or more instructions between PTA and BLINK

- Calculate Target Address
- Transfer and Select it as Fetch Address
- Preload Target instruction and Select it as Next Instruction

No Branch Overhead
**SIMD Instructions**

- **MCMPGT.W Rm, Rn, Rd** (Compare)
  - Rm: 0 1 2 3
  - Rn: 8 4 2 1
  - Rd: 0 0 0 ffff
  - Comparison: >? >? >? >?
  - Result: False False False True

- **MCMV Rm, Rn, Rw** (Bitwise Conditional Move)
  - Rm: 0 1 2 3
  - Rn: 8 4 2 1
  - Rw: 0 0 0 ffff
  - for(i=0; i<64; i++)
  - if(Rn[i]==1)
  - Rw[i]=Rm[i]
SIMD Instructions (Cont’d)

MMULSUM.WQ  Rm, Rn, Rw  (Multiply-accumulate)

4 Multiplies

4 Adds

64-bit result:
Very High Accuracy
(No rounding or saturation is necessary)

issued every cycle

8 operations/cycle

3.2 GOPS
1.6 MMACS
@ 400 MHz
**SIMD Instructions (Cont’d)**

- **MPERM.W Rm, Rn, Rd** (Permuted)
  - Position # 11 10 01 00
  - Rm → Rn → Rd

- **MEXTR2 Rm, Rn, Rd** (Extract)
  - Rn → Rm → Rd
  - 2-byte offset
  - 7 instructions for 1-7 byte offsets

Control information per 16 bits: b00011011
SIMD v.s. Multiple Issues

❖ SH-5: **4-way SIMD for 16-bit Data**
   ❖ x4 Peak Performance *(Same Operations in Parallel)*
   ❖ Data Alignment Overhead Cycles
   ❖ **Lower Cost:** Simple Control and Small Area Overhead
     ❖ Simple Datapath Division: 64 bits into 4 x 16 bits

❖ **Reference Design: Multiple Issues**
   ❖ Three Issues *w/o Execution Module Duplication*
     ❖ Minimizing Area Difference from SIMD
     ❖ 1 Load/Store, 1 Multiplier, etc.
     ❖ Four or more issues are not effective without the duplication.
   ❖ x3 Peak Performance *(Different Operations in Parallel)*
   ❖ **Higher Cost:** Complicated Control for Multiple Issues
Example: Vector Maximum

Find the location and value of the maximum value in a vector
- Data Type: 16-bit Fixed Point

Kernel C Source

```c
for(i=1;i<N;i++)
    if(maxValue < in[i]){
        maxLocation = i;
        maxValue = in[i];
    }
```

SIMD Algorithm

1. search every four elements
2. search the four values

Maximum values of every four elements

Maximum values of every two elements

Maximum value
Vector Maximum (cont’d)

Non-SIMD Code

- 6 instructions/loop
- Repeat N-1 Times

<table>
<thead>
<tr>
<th>Instruction</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPGT R3,R4,R6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMVNE R6,R3,R4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMVNE R6,R2,R5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDX.W R0,R2,R3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDI R2, 2,R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE R1,R2,T0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SIMD Code

- 7 instructions/loop
- Repeat N/4-1 Times

<table>
<thead>
<tr>
<th>Instruction</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCMPGT.W R3,R4,R6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD R8,R7,R8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCMV R3,R6,R4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCMV R8,R6,R5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDX.Q R0,R2,R3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDI R2,8,R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE R1,R2,T0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

T0: Loop Top Address
R0: pointer to in
R1: N x2
R2: i x2
R3: in [i]
R4: maxValue
R5: maxLocation (x2)
R6: compare result
R7: 0x04040404
R8: i,i+1,i+2,i+3
Vector Maximum (3 Issues)

Non-SIMD Twice-unrolled Code for Three Issues
(Loop part)

11 instructions/loop, 4 cycles/loop, Repeat N/2-1 times

a CMVNE is issued every cycle

Three issues are enough to achieve the best performance
(assuming no module duplication)

<table>
<thead>
<tr>
<th>Issue Slot 1</th>
<th>Slot #2</th>
<th>Slot #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMVNE R6,R3,R4; LDX.W R0,R2,R3; ADDI R2, 2,R2;</td>
<td>CMVNE R6,R2,R5; CMPGT R9,R4,R6;</td>
<td>CMVNE R6,R9,R4; LDX.W R0,R2,R9; ADDI R2, 2,R2;</td>
</tr>
<tr>
<td>R6: compare result</td>
<td>R5: maxLocation (x2)</td>
<td>R6: compare result</td>
</tr>
<tr>
<td>R5: maxLocation (x2)</td>
<td>R0: pointer to in</td>
<td>R3: in[i]</td>
</tr>
<tr>
<td>R3: in[i]</td>
<td>R1: N x2</td>
<td>R4: maxValue</td>
</tr>
<tr>
<td>R9: in[i] for unrolling</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Vector Maximum (Results)

Kernel Execution Cycles (N=40)

- 4-way SIMD is better than the three issues
- Number of conditional moves limits the three-issue performance
- Loop unrolling reduces loop overhead

- **Non-SIMD 1 Issue**
- **4-way SIMD 1 Issue (SH-5)**
- **Non-SIMD 3 Issues (Reference Design)**
Example: Real Block FIR

for(i=0; i<N; i++){
    sum[i]=0;
    for(j=T-1; j>=0; j--)
        sum[i]+=in[i-j+T-1]*coefs[j];
    if(scaling) sum[i]*=FACTOR;
} /* in[0:T-1]: DL copy, in[T:N+T-1]: new input */

- **TN**: Multiply-Accumulate Operations
- **2T+N-1**: Source Operands

DL (Delay Line) values before kernel execution

DL values after kernel execution

Accumulating Direction

coefs[T-1]

coefs[j]

coefs[0]

in[i-j+T-1]*coefs[j]
Real Block FIR (Cont’d)

### Non-SIMD Code (Inner Loop part)
- **6 instructions/loop**
  - **Repeat TN/16 times**
  - 
    | Instruction   | Registers       |
    |---------------|-----------------|
    | LD.W          | R1, 0, R6       |
    | ADDI          | R0, -2, R0      |
    | ADDI          | R1, 2, R1       |
    | MMACFX.WL     | R4, R6, R10     |
    | BNE/L         | R0, R2, T0      |

**T0**: Loop Top Address   **R4**: coefs
**R0**: pointer to coefs   **R6-R8**: in
**R1**: pointer to in      **R10-R13**: sum
**R2**: pointer next to coefs

### SIMD Code Unrolled Four Times
- **13 instructions/loop**
  - **Repeat TN/16 times**
  - 
    | Instruction   | Registers       |
    |---------------|-----------------|
    | LD            | R1, 0, R6       |
    | LD.Q          | R0, -8, R0      |
    | ADDI          | R1, 8, R1       |
    | MMULSUM.WQ    | R4, R6, R10     |
    | MEXTR6        | R6, R7, R8      |
    | MMULSUM.WQ    | R4, R8, R11     |
    | MEXTR4        | R6, R7, R8      |
    | MMULSUM.WQ    | R4, R8, R12     |
    | MEXTR2        | R6, R7, R8      |
    | MMULSUM.WQ    | R4, R8, R13     |
    | ADDI          | R6, 0, R7       |
    | BNE/L         | R0, R2, T0      |
Real Block FIR (3 Issues)

- Non-SIMD Code Unrolled Four Times for Three Issues
  - 11 instructions/loop, 4 cycles/loop, Repeat TN/4 times
  - Software pipelining is applied to avoid pipeline stalls.

- an MMACFX is issued every cycle
  - Three issues are enough to achieve the best performance (assuming no module duplication)

<table>
<thead>
<tr>
<th>Issue Slot #1</th>
<th>#2</th>
<th>#3</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMACFX.WL</td>
<td>R4,R6,R10;</td>
<td>LD.W R0,0,R4;</td>
</tr>
<tr>
<td>MMACFX.WL</td>
<td>R5,R6,R11;</td>
<td>LD.W R1,2,R6</td>
</tr>
<tr>
<td>MMACFX.WL</td>
<td>R5,R7,R10;</td>
<td>LD.W R0,2,R5;</td>
</tr>
<tr>
<td>MMACFX.WL</td>
<td>R4,R7,R11;</td>
<td>LD.W R1,0,R7;</td>
</tr>
</tbody>
</table>

T0: Loop Top Address
R0: pointer to coefs
R1: pointer to in
R2: pointer next to coefs
R4,R5: coefs
R6,R7: in
R10,R11: sum
4-way SIMD is better than the three issues.

Loop unrolling reduces the number of source operand reloads, and enhances performance.

Rich register states enable extensive unrolling for higher performance.
Result Comparison

SH-5: Excellent Price-Performance Core

<table>
<thead>
<tr>
<th></th>
<th>Cycle(^v)</th>
<th>Time(^v)</th>
<th>Time x Price(^v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector Maximum</td>
<td>SH-5</td>
<td>76(^o)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MSC8101</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C5510</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>Real Block FIR</td>
<td>SH-5</td>
<td>420(^o)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MSC8101</td>
<td>183</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C5510</td>
<td>393</td>
<td></td>
</tr>
<tr>
<td>IIR Filter</td>
<td>SH-5</td>
<td>23(^o)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MSC8101</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C5510</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>

\(^v\)Ratio to SH-5

\(^o\) With caches preloaded. With empty caches 192, 716 and 101 (estimated) cycles for Vec Max, Block FIR and IIR respectively.

MSC8101 300MHz ($96) and TI C5510 160MHz ($29) data from “Buyer’s Guide to DSP Processors” 2001 Edition by BDTI

SH-5 results are projected based on execution on ISS (expected to be published by BDTI in the near future).

Hitachi is projecting that the SH-5 operating at 400MHz is priced at $40 in 10,000 units lots at the end of 2002.

© 2001 BDTI
Conclusion

**SH-5:**
- Good Balance of Performance, Power, and Price
- Targeting Cost-sensitive Consumer Market

**SIMD is Better than Multiple Issues**
- for Multimedia Applications
- Both Performance and Cost

**Future Plan: SH-6 and Beyond**
- Next-generation process: integrate more logic within a reasonable cost
- SIMD + Multiple Issues will be the “Next Approach”