A Multithreaded RISC/DSP Processor with High Speed Interconnect

Erik Norden
Senior Architect
Agenda

- TriCore 2 Overview
- Microarchitecture
- High Speed Interconnect
- Multithreading
- Hardmacro Development
- Summary
The Embedded Processing Challenge

Control and DSP Functionality

- Small Core Area
- Small Code Size
- Configurability
- Scalable Solutions
- DSP Instruction Set
- Real Time Performance
- Overcome Memory Bottleneck

Low Cost
High Performance
Low Power

Small, Fast System-on-Chip and Embedded Core Solution
TriCore 2 Microprocessor System

- ISA superset of TriCore 1 family, the first unified RISC/DSP architecture
- 6-stage superscalar pipeline
- Multithreading extension (optional)
- Improved co-processor interface / support
- Improved optional floating point unit (FPU)
- High bandwidth system interconnect hierarchy
Configurability

Configurable / scalable:
- Instruction cache and data cache sizes
- Instruction scratch and data scratch memory sizes
- Crossbar: number of master and slave ports

Optional:
- Multithreading extension
- FPU
- MMU
- Individual co-processors
TriCore 2 Pipeline

- Fetch Pipeline
  - Thread 0 PC
  - Thread 1 PC
- Pre-Decide
- Instruction Buffer
- Fetch Pipeline
- Register File Thread 0
- Register File Thread 1
- Queue
- Decode
- Execute 1
- Execute 2
- Write Back
- Integer Pipeline
- Load/Store Pipeline
- Loop Pipeline
- Execution Pipelines
Problem: Pipeline Effects

- Deeper six-stage pipeline achieves MHz goal, creates IPC problems:
  - Longer branch resolution / latency
  - Load to use delay slots, use to store delay slots
  
  ▶ Possible solution: unrolling DSP loops, with resultant register pressure

- Different characteristic of longer pipeline causes performance problems for existing DSP code base
Efficient DSP Operations

FIR filter (main loop):

L1: dMAC
Load.dw
dMAC
Load.dw
Loop

L1: dMAC Load.dw
dMAC Load.dw Loop

→ 2 cycles per iteration

Vector multiplication (main loop):

L2: dMUL
Load.dw
dMUL
Load.dw
St.dw
Loop

L2: dMUL Load.dw
dMUL Load.dw St.dw Loop

→ 3 cycles per iteration
Load to Use Delay Slots

Instruction in integer decode stage (Dec) requires load data from instruction in load/store writeback stage (WB)

Example:
load d0, [a0]
add d3, d0, #1
Load to Use Delay Slots

- “Trombone” concept couples load/store pipeline and integer pipeline to eliminate delay
- Integer pipe slides relative to load / store pipe

Example:
- load d0, [a0]
- add d3, d0, #1

Instruction in integer decode stage (Dec) requires load data from instruction in load/store writeback stage (WB)
Load to Use Delay Slots

- “Trombone” concept couples load/store pipeline and integer pipeline to eliminate delay
- Integer pipe slides relative to load / store pipe
- Continue issuing instructions to both pipelines
  - No issue slots lost

Instruction in integer decode stage (Dec) requires load data from instruction in load/store writeback stage (WB)

Example:
- load d0, [a0]
- add d3, d0, #1
Use to Store Delay Slots

Result is ready *after* the memory pipeline stage

- Write target buffers to eliminate stalls due to store operations

Example:

```
mul d3,d0,d1,
store [a0],d3
```
Use to Store Delay Slots

Result is ready *after* the memory pipeline stage

- Write target buffers to eliminate stalls due to store operations
- So result goes into buffer and uses the next Mem slot

Example:

```
mul d3,d0,d1
store [a0],d3
```
Use to Store Delay Slots

Result is ready *after* the memory pipeline stage

- Write target buffers to eliminate stalls due to store operations
- So result goes into buffer and uses the next Mem slot

And is designed to compensate for “Trombone” concept
Solution for Pipeline Effects

- Branch resolution / latency
  - Early resolution & fetch decoupling

- Load to use delay slots
  - "Trombone" concept couples load/store pipeline and integer pipeline to eliminate delay

- Use to store delay slots
  - Write target buffers to eliminate stalls due to store operations

Result is:
- For DSP code, the TriCore 2 pipeline looks same as TriCore 1 pipeline
  - Existing DSP code base can be re-used, no TriCore 2 specific performance optimization required

- IPC is almost identical to TriCore 1: ~1.5 instructions/clock
High Speed Interconnect

- Interface to bus / system
  - Full 64-bit crossbar
  - Full core frequency operation
  - Scalable / modular sockets
    - Local memories
    - External interfaces
    - Busses / cores
- Isolation of local traffic
- Maximum concurrent bandwidth
SRI Protocol: Main Features

- Crossbar based protocol
- Synchronous bus, 32 bit address, 64 bit data
- Burst length: 2 / 4
- Single data transactions for 8/16/32/64 bit
- RMW transaction support
- Supports pipelined transactions
- No wait states during block data transmission
- System scalability and isolation
- Full core frequency operation
- Debug- and power saving features
A Multiprocessing Core
Multithreading

Typical Core

- Tasks
- Interrupt Service Routines
- Stalls due to Cache Miss
- Memory Resident Task
- Deadline

Multithreaded Core

- Tasks
- Interrupt Service Routines
- Saved Execution Time
- Deadline

Memory Resident Task
Target Application Profile

- Typically a mix of:
  - General purpose tasks
    - E.g. control/protocol related
    - With large code footprint
    - Reside in cached off-chip flash
  - Algorithmic (DSP) tasks
    - With small code footprint
    - Fit in fast on-chip memory (scratch memory) - memory resident tasks

- Common profile of many deeply embedded systems
TriCore 2 Multithreading Extension

- Block multithreading – 2 threads supported
  - High performance gain and power minimization
  - Diminishing benefits for more than 2 threads
- Thread switching is primarily on instruction cache misses.
- Thread timer guarantees minimum execution time for both threads.
- New “YIELD” instruction allows one thread to relinquish execution to the other
- Maintains compatibility with existing processor architecture
  - 2 virtual processors, two copies of processor state are present in the CPU
Improved System Performance

- Multithreading: One thread in scratch memory
- Multithreading: Both threads in cache with same miss rate
- No multithreading

In all 3 cases: CPU at 333 MHz; 66 MHz 32-bit flash on crossbar

Typical multithreading performance area

Typical Cache Performance Area
~200% to 400% Improvement

Mobile Handset
Automotive Engine Control

Effective CPU MHz vs. Miss Rate %

- Typical multithreading performance area
- Mobile Handset
- Automotive Engine Control
- No multithreading
- Multithreading: Both threads in cache with same miss rate

In all 3 cases: CPU at 333 MHz; 66 MHz 32-bit flash on crossbar
TriCore 2 Hard Implementation

- 0.13 micron technology
- Core area: 3 mm²
- Hardmacro area: 8 mm²
  includes CPU, MMU, Multithreading extension, FPU, 160 Kbyte total memory
- Frequency 400-500 MHz (typical)
- 1.5 MIPS / MHz (typical compiled code)
- 2 MMACS / MHz
- 0.5 mW/MHz @ 1.5V
TriCore 2 Multithreaded - Hardmacro
TriCore 2 Summary

- High configurability
  - Scratch memory and cache sizes
  - Optional FPU, MMU, coprocessors, multithreading

- High performance
  - Efficient pipeline

- Multithreading
  - Saves costs and power consumption

- Open, scalable crossbar architecture
  - SRI protocol
  - Efficient concurrent communication to code and data memory and other IP

- Multiprocessing capability
Hot Chips 15

Never Stop Thinking

Erik Norden, Senior Architect
Erik.Norden@infineon.com
http://www.infineon.com