FTF-Orlando, June 25-28 2007

AN317: Porting Single-Core Applications to Multi-Core Platforms

Case Study on MSC8144

Michael Kardonik
Applications Engineer
After This Presentation You Will

► Know basic approaches on MSC8144 multi-core processing

► Know some simple guidelines for choosing the right programming model when porting to multi-core

► Come see our multi-core demo at workstation 505 in the Networking Section of the TechLab
Agenda

► Two basic models for multi-core programming
  • Each core acts independently - “multiple single cores”
  • Cores cooperate each other – “true multi-core”

► Examples of typical applications and flow

► How to identify what model to use

► Detailed example porting a single core application to multi-core
MSC8144 – Background

- 4 Cores
- L2 and L1 Instruction cache
- L1 Data cache
- Inter-core interrupts
- M2 and M3 shared memory
DSP Multi-core Processing – “Multiple Single Cores”

Possible data flow example – media gateway
“Multiple Single Cores”

► Each core acts independently

• Pros
  - Simplifying a porting from Single core systems
  - The minimum of interaction between cores – less overhead and more predictable system
  - No cache coherency issues between the cores
  - Tools support may remain the same as it was for single core
  - Good scalability – however depends on hardware support

• Cons
  - Load balancing issues – some cores maybe idle and some overloaded.
  - Hardware should support this mode of operations by providing I/O Queues for network interfaces.
“Multiple Single Cores”

► “Good” candidate – application’s features

• I/O can be statically assigned to each core

• Complicated control path and very strict hard-real time constraints

• Small code size of the application – cache can be used more efficiently
“True Multi-Core”

Possible data flow example – video application

[Diagram showing possible data flow example with components such as PCI Bus, CORE 1, CORE 0, M3 memory, M2 Memory, QUICC Engine, and RTP DATA flow.]
“True Multi-Core”

Cores cooperate each other

• Pros
  - Better possibilities for balance loading meaning more effective usage of system resources
  - L1 instruction cache can be used more efficiently (cache affinity)

• Cons
  - Porting from single core is typically more complicated
  - Possible cache coherency issues between the cores
  - System becomes more complex especially when dependencies exist between tasks. As a result, hard-real time scheduling is harder to achieve
“True Multi-Core”

“Good” candidate – application’s features

- Impossible to accomplish by only one core and possible to divide to several concurrent tasks
- I/O can not be statically assigned to each core
- Soft-real time applications
Another Example of “True Multi-Core” – Master-Slave

Master core is responsible for all I/O operations and uses all the cores as the slaves. It decides what task each core performs.

Slave cores do not communicate each other but only through a master core.
Another Example of “True Multi-Core” – SMP

► One of the cores is responsible for all I/O operations

► Scheduling is made on global basis. Scheduler information is shared between all the cores and every core executes it
Porting a Single Core Application to Multi-Core – Guidelines

► Identify the threads (tasks) that can be executed concurrently by different cores

► How to choose these tasks?
  - Minimize inter-task dependencies
  - Each task should have schedulable real-time characteristics for single core
    - Avoid too short tasks because of overhead
    - Keep place for tuning at implementation stage

► Identify inter-task dependencies
  - Inter-task dependencies may cause performance degradation as one core will have to wait for other cores and as a result to missing deadlines.
  - Inter-task dependencies may affect your scheduler decisions
Task Dependencies Existing in Multi-Core

There are task dependencies that are hidden in single-core applications but exposing in multicore

- Serialization – single core applications are serial. If there is no intertask dependencies, first released task of the same priority will be finished first even if its execution time is longer then the next task. Therefore, it may be situation that a single core application relies on this fact. On multi-core this “not in-order” situation may happen as the next task can be executed by other core.

- Concurrent execution – in many cases, tasks that can not execute concurrently in single core application, will execute at the same time in multi-core environment (for example ISRs).
The 8x8 Discrete Cosine Transform (DCT) on each of (Y, Cb, Cr)

The zig-zag reordering of the 64 DCT coefficients from previous step

Quantization
- Each value is divided by a a number specified in a vector with 64 values and rounded to next integer

\[
\text{for } (i = 0 ; i<63; i++) \text{ vector}[i] = (\text{int}) (\text{vector}[i] / \text{quantization_table}[i] + 0.5)
\]

The Zero Run Length Coding (RLC)

Huffman coding
Motion JPEG – Applications Characteristics

- Real-Time stream of raw video data arrives to one IP address
- Need to encode this stream and send it back in encoded format
- There are no hard-real time constraints but minimizing latency is important
- Code size is small
- Data arrives in chunks of 2 MCU (Minimum Coded Unit - 512 bytes)
- There is no relation between any two chunks of data
- Data need to be sent back in the same exact order as it was received
MJPEG – Tools We Are Using

► SmartDSP Operating System (Free OS targeting StarCore coming with CodeWarrior)

► CodeWarrior and Kernel Awareness plug-in

► CodeWarrior Profiler
MJPEG – Multi-Core Porting Process

► First we ran a single core application on MSC8144 on one core only.

► Then we analyzed what steps to take in order to run two or more instantiations of JPEG encoder on the same core as a debugging process is simpler on one core – we have used a functional simulator for this purpose.

► Examples of the problems
  • The code of JPEG encoder has used internal counter for number of MCU. In our case every instance of encoder will work with random MCU number so we had to change it to be an argument of a function. (serialization problem)
  • On master core we had to implement a serializer that will make sure that the output order of MCUs is the same as input – problem that would not exist in single core application (serialization problem)
MJPEG – Application Design Decisions

► Because the data is sent to the same IP we decided to choose one core as a master core that will receive all the data and send it to other cores (from previous analysis – static I/O port cannot be assigned to each core as we have only one stream so it is a good candidate to “True Multi-Core”)

► Latency is important in the application so we decided to process data immediately upon receiving of Ethernet frame. It also may minimize a memory usage in the system (latency is important and there is only one stream of data – “True Multi-Core”)

► Because all the MCUs are independent we may process any of them on any core

► Code size is very small so we will not take into account Cache affinity

► Master core will en-queue arrived Ethernet frames to a queue and all other cores will compete for a job including a master core

► Slave cores will use “passive polling” technique: all the cores are in wait state and interrupts wake them from this state but no ISR is executed

► Master core will be responsible to send encoded data in right order (serialize the data)
MJPEG – Data Flow

MSC8144

CORE 1

Cache

CORE 2

Cache

CORE 0

Cache

CORE 3

Cache

M3 memory

Queue

Rx

QUICC Engine

Raw DATA

Cores

Peripherals

Data Flow

RAM
MJPEG – Data Flow

MSC8144

- CORE 1 Cache
- CORE 2 Cache
- CORE 3 Cache
- CORE 0 Cache
- M3 memory
- QUICC Engine
- Rx
- Raw DATA

Colors:
- Cores: Green
- Peripherals: Blue
- Data Flow: Red
- RAM: Blue

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © Freescale Semiconductor, Inc. 2007-2008.
MJPEG – Data Flow

MSC8144

CORE 1 Cache

CORE 2 Cache

M3 memory

CORE 0 Cache

CORE 3 Cache

Queue

Rx

QUICC Engine

Raw DATA

Cores

Peripherals

Data Flow

RAM

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © Freescale Semiconductor, Inc. 2007-2008.
MJPEG – Data Flow
MJPEG – Control Flow

**MSC8144**

- CORE 1 Cache
- CORE 2 Cache
- CORE 0 Cache
- CORE 3 Cache
- M2 Memory
- QUICC Engine

**Flow**
- Rx Interrupt every several frames
- Raw DATA

**Legend**
- Cores
- Peripherals
- Data Flow
- RAM
- Interrupts
MJPEG – Control Flow

MSC8144

- CORE 1 Cache
- CORE 2 Cache
- CORE 3 Cache
- CORE 0 Cache
- M2 Memory
- Queue
- QUICC Engine
- Raw DATA
- Rx Interrupt every several frames

Legend:
- **Cores**
- **Peripherals**
- **Data Flow**
- **RAM**
- **Interrupts**
MJPEG – Control Flow

MSC8144

- CORE 0 Cache
- CORE 1 Cache
- CORE 2 Cache
- CORE 3 Cache
- M2 Memory
- QUICC Engine

Flow:
- Raw DATA
- Rx Interrupt every several frames

Legend:
- **Cores**
- **Peripherals**
- **Data Flow**
- **RAM**
- **Interrupts**
**MJPEG – Control Flow**

![Diagram of MSC8144 processor layout with cores, cache, RAM, and peripheral connections highlighting data flow and interrupts.]

- **CORE 1** Cache
- **CORE 2** Cache
- **CORE 0** Cache
- **CORE 3** Cache
- **M2 Memory**
- **QUICC Engine**

**Key Points:**
- **Rx Interrupt every several frames**
- **Raw DATA**

**Legend:**
- **Cores**
- **Peripherals**
- **Data Flow**
- **RAM**
- **Interrupts**
Motion JPEG – GUI

The image shows a user interface for a motion JPEG application with several graphs and data displayed. The interface includes:

- An image labeled "NJ Turnpike East".
- A graph showing application utilization with values for Core 1, Core 2, Core 3, and System:
  - Core 1: 46%
  - Core 2: 0%
  - Core 3: 45%
  - System: 34%
- A graph showing history running average with values for 36%, 17%, 31%, 36%, and 30%.
- A histogram labeled "History is 60 Items".
- Properties show frames per second: 46, and frames sent: 2093.
- Buttons for test and pause.
- Options for open file, send file, and close.
Multi-Core Starter

► Freescale has more than 5 years experience with DSP multi-core and has built good knowledge base to share with customers

► Basic multi-core application for MSC8144 in this presentation will be soon available to customers

► Multi-core paradigm that was used is platform independent and can be used on other platforms

► This application utilizes the following features
  • Inter-core communication queues
  • Multi-core memory management
  • Cache coherency support

► It can serve as a good starting point for many new multi-core designs
Conclusions

► Designing of multi-core application should take into account many factors such as task properties, cache properties, inter-core communication abilities, etc.

► Generally, it is impractical to search for optimal task assignment in real-time as there are too many factors that affect it, so typically static scheduling will be used and extensive testing and tuning are needed.

► Use the Freescale’s tools to measure performance for the entire system such as cache hit ratio, bus utilization etc. Those measurements are especially important in tuning process in the complex multi-core system.

► Customers can use the infrastructure provided by Freescale that includes full solution including hardware, software and tools.