Porting Linux to the M32R Processor

Hirokazu Takata
Renesas Technology Corp., System Core Technology Div.
takata.hirokazu@renesas.com
Renesas = Hitachi + Mitsubishi

- **Renesas Technology Corporation**
  ~ Renaissance Semiconductor  
  for Advanced Solutions  ~

  - New joint company established by *Hitachi* and *Mitsubishi* (April 2003)
  - World’s Largest Microcontroller Company

- 32-bit RISC Microcomputer
  - SuperH Family … for processor application
  - M32R Family … for controller application

http://www.renesas.com/
Outline

• Introduction
  – Why Linux on M32R?
  – Object: to provide “M32R Linux platform”

• Target hardware environment
  – M32R softmacro core, FPGA evaluation board

• Development of Linux/M32R
  – Porting of the Linux kernel
  – Development of GNU tools and libraries
  – Preparation of software packages (**deb** packages)

• Summary

• Demonstration
Introduction

• Background
  – Progress of system LSI technology
    ⇒ System-on-a-Chip (SoC)
  – Embedded systems will be connected each other.
    • Embedded systems will be more functional.
    • It is required to develop software efficiently on a de facto standard environment (Linux etc.).

• Objects
  – Establish GNU/Linux environment for the M32R
  – Prototyping of a new M32R processor
    • support SMP
    • with MMU
M32R Linux Platform

• M32R microprocessor
  – 32-bit RISC microprocessor for embedded systems (Renesas original architecture)

• Linux/M32R Project (2000〜)
  – GNU/Linux Environment for M32R
  – Development of Linux/M32R (A new architecture port to the M32R)
    • Development of target hardware environment:
      – New M32R cores (with MMU) and evaluation boards
    • Porting Linux kernel
    • Development of GNU toolchains (GCC, Binutils)
    • Porting GNU C libraries
    • Preparation of self tools and root filesystem
Linux/M32R Current Status

• Linux kernel
  – linux-2.4.19
  – with SMP support

• Device drivers
  – Serial driver
  – Ethernet LAN driver
  – Frame buffer device
  – PC/CF card
    • Wireless LAN, Compact flash, etc.

• User land
  – Root filesystem: Based on the Debian GNU/Linux
  – Standalone/NFSRoot environment
  – Self tools
    (GCC, Binutils, etc.)
  – glibc-2.2.5
  – LinuxThreds library (Pthreads)
Linux/M32R Current Status

- GUI Environment
  - Window Systems
    - X
    - Qt-Embedded
    - MicroWindows

Snapshot of the X desktop image
Target Hardware Environment

- Develop the target hardware environment of the M32R Linux Platform
  - M32R softmacro core
  - FPGA Evaluation board “Mappi”
M32R Softmacro Core

• Softmacro Core
  (Full Synthesizable Core)
  – Not dependent on process technologies
  – Can be mapped to an FPGA
    → Easy revise and update

• M32R-II Core
  – Upper compatible ISA to the M32R
  – 5-stage pipeline, dual-issue
  – out-of-order completion
  – MMU support
  – Compact size
# M32R Registers (M32R-II)

## General Purpose Registers

<table>
<thead>
<tr>
<th>0</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td></td>
</tr>
<tr>
<td>R10</td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td></td>
</tr>
<tr>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>R13</td>
<td></td>
</tr>
<tr>
<td>R14 (Link Register; LR)</td>
<td></td>
</tr>
<tr>
<td>R15 (Stack Pointer; SP)</td>
<td></td>
</tr>
</tbody>
</table>

## Control Registers

<table>
<thead>
<tr>
<th>0</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR0 (PSW)</td>
<td></td>
</tr>
<tr>
<td>CR1 (CBR)</td>
<td></td>
</tr>
<tr>
<td>CR2 (SPI)</td>
<td></td>
</tr>
<tr>
<td>CR3 (SPU)</td>
<td></td>
</tr>
<tr>
<td>CR5 (EVB)</td>
<td></td>
</tr>
<tr>
<td>CR6 (BPC)</td>
<td></td>
</tr>
<tr>
<td>Processor Status Word</td>
<td></td>
</tr>
<tr>
<td>Condition Bit Register</td>
<td></td>
</tr>
<tr>
<td>Interrupt Stack Pointer</td>
<td></td>
</tr>
<tr>
<td>User Stack Pointer</td>
<td></td>
</tr>
<tr>
<td>EIT Vector Base Register</td>
<td></td>
</tr>
<tr>
<td>Backup PC</td>
<td></td>
</tr>
</tbody>
</table>

## Accumulators

<table>
<thead>
<tr>
<th>0</th>
<th>8</th>
<th>63</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Program Counter

<table>
<thead>
<tr>
<th>0</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation Board “Mappi”

- Simple board … minimum hardware for Linux
- 2 FPGAs on the CPU board;
  An M32R softmacro core can be mapped into one FPGA.
  → Dual processor system can be achieved.

**Block Diagram**

- CPU Board
  - CPU
  - Mem
  - BIU
  - FPGA#0
  - I/O
  - FPGA#1
    - CPU #1 or User-Logic

- Extension Board
  - SDRAM 64MB
  - FLASH 4MB
  - DispC
  - LAN
  - PC-card
Development of Linux/M32R

• **Port the Linux kernel**
  – Porting Linux kernel to the M32R processor
  – Development of SMP kernel
• Enhance GNU tools (GCC, Binutils)
• Port GNU libraries (glibc, etc.)
• Prepare debug environment
• Build software packages
Porting of the Linux Kernel

• Architecture dependent portions
  – include/asm-m32r/, arch/m32r/

• M32R specific implementations
  – Asm function routines
  – System call interface
  – Memory management routines
    • Based on the M32R’s MMU/Cache specification
    • Split MMU exception handlers to lighten the TLB miss operation.
Porting the Linux Kernel (Cont.)

- Linux kernel for the M32R
  - Started to port linux-2.2 kernel (v2.2.16~)
  - After that, upgraded to linux-2.4 kernel (the latest ver. is v2.4.19)

- Development process
  - How we developed Linux kernel for the M32R …
    - Started porting from the scheduler (by utilizing GNU simulator)
    - It is difficult to complete compilation, if header files are not complete.
      ⇒ Having made stub routines, we developed the kernel gradually.
  - What were problems?
    - In Linux kernel, GCC enhancement features are heavily used
      (inline functions, asm functions)
    - Maturity of cross tools (Develop and debug tools in parallel)
System Call Interface

- System call I/F
  - System call: TRAP #2
    - R7: System call number
    - R0 ~ R6: arg 0 ~ arg 6 (max. 7 arguments)
  - Pass pt_regs as an implicit stack parameter
  - Stack is explicitly changed by CLRPSW instruction

<table>
<thead>
<tr>
<th>User process</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>User stack (SPU)</td>
<td>Kernel stack (SPI)</td>
</tr>
</tbody>
</table>

System call invocation

TRAP#2

System call execution

RTE
Memory Management

• Memory management of Linux (Paging)
  – Demand loading operation and Copy-On-Write operation can be executed by utilizing MMU exceptions.
    ⇒ MMU is necessary

• M32R-II Core
  – MMU
    • TLB entries are updated by software (cf. MIPS)
    • Number of TLB entries: Instruction/Data 16 entries (FPGA ver.)
    • 4KB/page (Linux/M32R), Large page (4MB)
  – Cache
    • Instruction/Data separated cache
    • Physically indexed physically tagged cache
      ⇒ Need no cache flushing operation
MMU Exception Handler

- Separate MMU exception operation into two exception handlers (because TLB misses happen more frequently than page faults)
  - TLB miss handler
  - Access exception handler

- In order to lighten the TLB miss handler ...
  - Simplify the TLB miss handler; it just sets a TLB entry
  - Write down in assembly code not to save full context

```
TLB miss handler

Set the page attribute field of the TLB entry
(set the page attribute to “not accessible”)

Access exception handler

dpage_fault()

handle_mm_fault()
```

MMU exception!
1. TLB miss exception
2. Access exception
   ↓
   reexecute after the execution of exception handlers

Renesas
Everywhere you imagine.
Porting the Linux kernel

- Porting Linux kernel to the M32R
- Development of SMP kernel
M32R Evaluation Chip

• M32R On-Chip Multiprocessor  (Ref.: Proc. of ISSCC 2003, 14.5)
Development of MP Linux System

• Development of the SMP kernel
  – Synchronization mechanism for SMP
    • Semaphore
    • Atomic access methods for variables
    • Spin lock … LOCK/UNLOCK instructions
  – Inter-Processor Communication
    • Inter-Processor Interrupt (IPI)
  – Boot operation

• Enhance GNU C Library (for multithread programming)
  – LinuxThreads library (Pthreads; POSIX 1003.1c)
  – User-level mutual exclusion
Inter-Processor Interrupt (IPI)

- Inter-Processor Interrupt (IPI)
  - To avoid a dead lock due to IPI collisions, only one CPU can send IPI request in the M32R implementation.

- M32R’s IPI spec.
  - IPI is non-maskable for the ICU
    → To mask IPI request, set IE bit (interrupt enable bit)
  - IPI requests are not queued
    → Sender CPU must confirm that the request have received by the receiver CPU.

```
Sender (CPU#0)
Get ipi_lock
Send IPI
  IPICR0 bit#30 = 1
Confirmation
  wait until
  IPICR0 bit#0 = 0
Done

Receiver (CPU#1)
EIT happens
Receive IPI
  read ICUISTS1
IPI operation
  exec IPI#0 handler
Done
```
Inter-Processor Interrupt (IPI) (Cont.)

- IPI requests
  - For the Linux/M32R, the following 5 IPI factors (IPI0～IPI4) are used.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPI0</td>
<td>Rescheduling request</td>
</tr>
<tr>
<td>IPI1</td>
<td>TLB flushing request</td>
</tr>
<tr>
<td>IPI2</td>
<td>Function execution request</td>
</tr>
<tr>
<td></td>
<td>• Flush whole TLB entries</td>
</tr>
<tr>
<td></td>
<td>• CPU stop request</td>
</tr>
<tr>
<td></td>
<td>• Request to free the slab cache</td>
</tr>
<tr>
<td>IPI3</td>
<td>Request for local timer operation</td>
</tr>
<tr>
<td>IPI4</td>
<td>CPU activation request</td>
</tr>
</tbody>
</table>

- IPI5～IPI7 are not used
- Local Timer

Currently, the local timer request is handled by IPI, because a broadcasted global timer request is not able to be accepted by all CPUs due to the ICU spec.
Boot Operation of SMP Kernel

- **BSP (Boot Strap Processor)**
  - CPU boots the system (only one)
  - BSP is selected by H/W
- **AP (Application Processor)**
  - CPUs except BSP
- **Boot sequence**
  - APs wait in sleeping at boot time (only IPIs can be accepted)
  - BSP initializes H/W and Linux, and finally boots all APs.
  - BSP sequentially activates APs, then set synchronization flags (smp_commenced) and makes APs into idle state (idle thread).
Development of Linux/M32R

• Port the Linux kernel
• **Enhance GNU tools (GCC, Binutils)**
  – m32r-linux toolchain
  – Dynamic linking support for shared libraries
• Port GNU libraries (glibc, etc.)
• Build software packages
• Prepare debug environment
Development of GNU Toolchain

• Enhancement of GNU tools (GCC, Binutils)
  – GCC (gcc-2.95 → gcc-3.2.3), Binutils (v2.11.92)
    • Based on the Cygnus GNUPro (m32r-elf toolchain)
    – Support ELF’s dynamic linking function
      • PIC generation, shared library support
      • Enhancement of BFD library
    – No changes of the C-language’s ABI (Application Binary Interface)
    – Endian support (little-endian is newly supported)

• Cross tools
  – Linux/x86 version cross tools (m32r-linux toolchain)

• Development of self tools
  – gcc, binutils, bash, sed, awk, perl, tcl
Dynamic Linking Support

• Dynamic Linking
  – Dynamic linking/loading must be supported to utilize shared libraries.
  – Program’s location (where to be loaded) is determined in runtime.
  ↓
  A program must be a relocatable and position independent binary.
  ※ PIC (Position Independent Code)

• PIC (Position Independent Code)
  – Address of global symbols are dynamically stored into a GOT.
  – GOT is used to resolve global symbol references.
    ※ GOT (Global Offset Table)
  – Target of subroutine calls are also resolved by using a PLT.
    ※ PLT (Procedure Linkage Table)
Implement ELF Dynamic Linking

- **GOT (Global Offset Table)**
  - GOT is accessed via the R12 register
  - PC value is fetched by `BL` (branch&link) instruction

- **PLT (Procedure Linkage Table)**
  - Symbol reference of a subroutine call is executed by indirect referencing of GOT.
    (like IA-32 implementation)

⇒ Invalidation of instruction cache lines are not required, because the code fragment of the PLT entries are not changed.
Development of Linux/M32R

- Port the Linux kernel
- Enhance GNU tools (GCC, Binutils)
  - Port GNU libraries (glibc, etc.)
    - Dynamic linker for the dynamic linking
    - LinuxThreads library (Pthreads)
- Prepare debug environment
- Build software packages
Porting Libraries

• Porting GNU C Library
  – glibc-2.2.3 ⇒ glibc-2.2.5
  – Dynamic Linker (ld-linux.so)
    • To use shared libraries (dynamically-linked libraries)
  – Implement LinuxThreads library (Pthreads)

• Development process
  1. Statically-linked “hello” binary
    • newlib version
    • glibc version
  2. Dynamically-linked binaries
    • hello.c, busybox, ...
LinuxThreads Library

• Enhance GNU C Library (to support multi-thread)
  – Multi-thread library: provides multi-thread programming env.
  – LinuxThreads library (Pthreads; POSIX 1003.1c)
    → User-level mutual exclusion has to be implemented to port

• Implement user-level mutual exclusion functions
  – On the user-level, interrupts cannot be disabled directly, and M32R’s LOCK/UNLOCK instructions cannot be applied.
  – Compare with some kind of mutual exclusion support methods
    • system call implementation
    • mutual exclusion algorithm
    → Employ the Lamport’s algorithm version
Development of Linux/M32R

- Port the Linux kernel
- Enhance GNU tools (GCC, Binutils)
- Port GNU libraries (glibc, etc.)
- **Prepare debug environment**
  - Debug tools
  - Development environment
- Build software packages
On-Chip Debugging Function

• SDI (Scalable Debug Interface)
  – SDI: debug interface specification commonly used by the M32R family
  – On-chip debugging function can be used through the JTAG port
    • Download target programs
    • Execute a monitor program

• Features
  – Monitor program and/or monitor ROM are not required on the target board.
  – High speed download
Debug Tools

• GDB with SDI support
  – Remote target: \texttt{m32rsdi}
  – Download, execution and debug by using SDI function
    • Virtual address is transformed by MMU
      ⇒ PC-break function is necessary.

• For the kernel debug
  – GDB (with SDI support)
  – Others: KGDB, GNU simulator (not support MMU)

• For the application debug
  – strace (trace system call invocations)
  – gdbserver (remote debugging via ethernet connection)
Development Environment

- ICE/emulator
- Serial (Console)
- Parallel or USB
- Ethernet (10BaseT)
- Hub
- Evaluation Board (Mappi)
- M32R-SDI (JTAG)
- Host PC (NFS server)
- FPGA adaptor for download
- USB
- USB FPGA adaptor for download
- Evaluation Board (Mappi)
Development of Linux/M32R

- Port the Linux kernel
- Enhance GNU tools (GCC, Binutils)
- Port GNU libraries (glibc, etc.)
- Prepare debug environment
- **Build software packages**
  - **Self packages for the target**
  - **Cross packages for the host machine**
Building Software Packages

• Employ the Debian GNU/Linux as a base distribution
  – Sophisticated Package Management (efficient for developing)
  – With cross development support
    • dpkg-cross
    • dpkg-buildpackage –a m32r –t m32r-linux
  – .deb packages for M32R:
    bash, libc6, perl, etc. … more than 300 packages

• Problems under cross-development
  – Header/library path is different from native environment.
    • Cannot configure/make correctly
      (Perl, X server/clients, etc.)
  – Management of header files and shared libraries of target
⇒ Utilize both self and cross development environment
Building Packages for Cross Dev.

Source Packages

Cross compile (dpkg-buildpackage -a"m32r")

"*_m32r.deb"

deb Packages for Self Env.
Converted by dpkg-cross

Install self packages

Self Env.

Root Filesystem
  • RAM disk (initrd)
  • NFS Root

deb Packages for Cross Env.

"*-m32r-cross_*_all.deb"

Install header files / libraries for the target.

Cross Tools
Evaluation

• Validation of Linux/M32R
  – LSB test suites v1.2.0
LSB Test Suites

- LSB (Linux Standard Base) Test Suite
  - Validation Test Suites for Linux
    - http://www.linuxbase.org/test/
- LSB Test Suite v1.2 … LSB Specification 1.2
  - Functional validation test suites: VSX-PCTS, LSB-OS
    - Runtime Environment test suite
    - Validation of the system call and standard library APIs

LSB 1.2

<table>
<thead>
<tr>
<th>VSX-PCTS (runtime)</th>
<th>LSB-OS</th>
<th>LSB-FHS</th>
<th>LSB-Usengroups</th>
</tr>
</thead>
<tbody>
<tr>
<td>VSXgen – Generic test suite layer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Environment Toolkit</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### LSB Test Results

- **Validation Result:** Good
- **The result of Linux/M32R is comparable to RedHat7.3.**

<table>
<thead>
<tr>
<th>Section</th>
<th>ANS hdr</th>
<th>ANS Los</th>
<th>POSIX hdr</th>
<th>POSIX os</th>
<th>LSB os</th>
<th>Total</th>
<th>Red Hat 7.3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F</td>
<td>M</td>
<td>F</td>
<td>M</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>Expect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>386</td>
<td>1244</td>
<td>1244</td>
<td>394</td>
<td>1600</td>
<td>1600</td>
<td>908</td>
<td>908</td>
</tr>
<tr>
<td></td>
<td>Actual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>386</td>
<td>1244</td>
<td>1244</td>
<td>394</td>
<td>1600</td>
<td>1600</td>
<td>908</td>
<td>908</td>
</tr>
<tr>
<td><strong>Succeed</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>176</td>
<td>1112</td>
<td>86</td>
<td>207</td>
<td>1333</td>
<td>0</td>
<td>695</td>
<td>0</td>
</tr>
<tr>
<td><strong>Failed</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>49</td>
<td>0</td>
</tr>
<tr>
<td><strong>Warnings</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td><strong>FIP</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Unresolved</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td><strong>Uninitiated</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Unsupported</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>203</td>
<td>0</td>
<td>0</td>
<td>179</td>
<td>72</td>
<td>0</td>
<td>59</td>
<td>0</td>
</tr>
<tr>
<td><strong>Untested</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>59</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td><strong>NotInUse</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>116</td>
<td>1158</td>
<td>1</td>
<td>179</td>
<td>1600</td>
<td>58</td>
<td>908</td>
</tr>
</tbody>
</table>

**Key:** F: Function, M: Macro; FIP: Further Information Provided
Future Work

• Linux/M32R Platform
  – Performance evaluation, Tuning, and Stabilization
  – Continue to develop and enhance
    • Prepare development environment for middlewares, and application programs
    • Upgrade kernel version (2.5 kernel)
      – MP performance, O(1) scheduler, Preemptive kernel, …
  – Feedback to the processor core design

• M32R GNU/Linux development environment
  – Publish the M32R GNU/Linux development environment
    • We’d like to merge source code to main stream if possible.
Summary

• Linux/M32R
  – The GNU/Linux environment for the M32R architecture
  – Linux system (UP / MP version) operates on both the M32R softmacro cores mapped on FPGAs, and an M32R single-chip multiprocessor evaluation chip.
    • Hardware/software co-design approach is employed
    • FPGA and M32R softmacro are useful for co-development or co-design of software programs and hardware IPs.

• Linux for embedded systems
  – The Open Source will provide a large impact on developing and designing of embedded systems.
  – Linux will play a great role in the field of embedded systems.
Linux/M32R Demonstration

Hirokazu Takata
Renesas Technology Corp., System Core Technology Div.
takata.hirokazu@renesas.com
Demonstration

- **Linux/M32R Demo. on an FPGA board “Mappi-II”**
  - The M32R softmacro runs on an FPGA
  - NFSroot mount with LAN connection
  - GDB (GNU debugger) with JTAG connection support

- **Linux/M32R SMP Operation Demonstration**
  - A tiny evaluation board “MicroServer” which has an M32R evaluation chip.
  - SMP kernel on the on-chip M32R multiprocessor

※ **MicroServer** : Developed by Mitsubishi Electric Corp.
Demonstration Environment

FPGA Board (Mappi-II)
- Configuration ROM
- FPGA

Host PC
- Serial (Console)
- Ethernet
- JTAG-ICE Interface
- Micro Server
- Wireless Network

Linux/M32R
- kernel-2.4.19
- glibc-2.2.5
- Debian GNU/Linux based root filesystem
Evaluation Board “Mappi-II”

- **FPGA**: XCV2000E (2M system gates)
- **On Board Memory**
  - SDRAM: 64MB
  - Flash ROM: 4MB
- **PC104 Bus**
- **Serial×1**
- **LAN×1** (100BaseT x)
- **USB×2**
- **Compact Flash×1**
- **JTAG Emulator**
- **M3A-ZA36**
- **Compact Flash**

**SDRAM**: 64MB
**Flash ROM**: 4MB
Extension Boards for “Mappi-II”

Main Board

M3A-ZA36
XCV2000E
(2M system gates)

Extension FPGA Board

M3A-ZA36G01
XC2V4000
(4M system gates)

M3A-ZA36G02
XC2V1000
(1M system gates)
Example of Embedded Micro Server

System Image

Name card box size

insert both sides

55mm

65mm

CPU

Specification

- CPU: CF size CPU module
- Network: Wired/Wireless LAN card of CF size
- Others: RS232C x1
Micro Server Module

“CF Card Size” CPU Module

• Features
  - CPU: M32R (Dual CPU)
  - OS: Linux
  - MW: WebServer (Boa)
  - SDRAM: 32MB
  - Flash: 8MB
  - I/F Con.: System, Debug, Power Supply

• System Components
  - I/O: Compact Flash Card (*)
  - System Board, Power Supply

(*) LAN, PHS, MicroDrive, etc.
  Lightweight wireless network

M32R (Evaluation Chip)