SW: Embedded Software Tools

USER

application source code

compiler

Application software

a.out

RTOS

debugger

simulator

CPU

ASIC

ASIC

ASIC

ASIC

ROM

RAM
Another View of Microprocessor Architecture

Let’s look at current architectural evolution from the standpoint of the software developers …, in particular Jerry Fiddler
**Fiddler’s Predictions for the Next Ten Years (2010)**

End of the “Age of the PC”

Lots of Exciting Applications

Development Will Continue To Be Hard

- Even as we and our competitors continue to make incredible efforts

Chips - No predictions

MEMS / Nano-technology & Sensors Will Impact Us
Fundamental Principles

Computers are, and will be, everywhere

The world itself is becoming more intelligent

Our infrastructure will have major software content

Most of our access to information will be through embedded systems

Economics will inexorably drive deployment of embedded systems

The Internet is one important factor in this trend

Reliability is a critical issue

EVERY tech and mfg. business will need to become good at embedded software

J. Fiddler - WRS
What Will Be Embedded in Ten Years?

Everything That is Now Electro-Mechanical
Machines (Nano-Machines)
Analog Signals
Anything that communicates
Lots of stuff in our cars
Our Bodies
  ● Today - Pacemakers
  ● Soon - De-Fibrillators, Insulin Dispensers
  ● We can all be the $6M Person, for a lot cheaper
All sorts of interfaces
  ● Speech, DNI, etc.
Embedded CPU cores are getting smaller; ~ 2mm² for up to 400 mHz
  - Less than 5% of CPU size

Higher Performance by:
  - Faster clock, deeper pipelines, branch prediction, ...

Trend is towards higher integration of processors with:
  - Devices that were on the board now on chip: “system on a chip”
  - Adding more compute power by add-on DSPs, ...
  - Much larger L1 / L2 caches on silicon

J. Fiddler - WRS
Microprocessor Chaos

J. Fiddler - WRS

Kurt Keutzer
A Challenging Environment

J. Fiddler - WRS

Expanding Functional Demands
Of Embedded Applications

And keep it small, stupid!

Numerous Microprocessor Architectures
Derivative Processors
Application-Specific CPUs
Systems On A Chip

Kurt Keutzer
New Hardware Challenges Software Development

J. Fiddler - WRS

More & More Architectures
  ● User-Customizable µprocessors

More Power Demands More Software Functionality
  ● Software is not following Moore’s law (yet)

System-on-a-chip

DSP
Embedded Software Crisis

- Cheaper, more powerful Microprocessors
- Increasing Time-to-market pressure
- More Applications
- Bigger, More Complex Applications
**SW: Embedded Software Tools**

- **User**
- **Application source code**
- **Compiler**
- **Debugger**
- **Simulator**
- **Application software**
- **a.out**
- **RTOS**

Diagram showing the flow from user input to application source code, through compiler, debugger, simulator, and finally to application software (a.out) and RTOS.
Outline on RTOS

Introduction

VxWorks
- General description
  - System
  - Supported processors
- Details
  - Kernel
  - Custom hardware support
  - Closely coupled multiprocessor support
  - Loosely coupled multiprocessor support

pSOS

eCos

Conclusion
Embedded Development: Generation 0

Development: Sneaker-net

Attributes:

- No OS
- Painful!
- Simple software only
Embedded Development: Generation 1

Hardware: SBC, minicomputer

Development: Native

Attributes:

- Full-function OS
  - Non-Scalable
  - Non-Portable
- Turnkey
- Very primitive
Embedded Development: Generation 2

Hardware: Embedded
Development: Cross, serial line

Attributes

- Kernel
- Originally no file sys, I/O, etc.
- No development environment
- No network
- Non-portable, in assembly
Embedded Development: Generation 3

Hardware: SBC, embedded
Development: Cross, Ethernet
  - Integrated, text-based, Unix

Attributes
  - Scalable, portable OS
    - Includes network, file & I/O sys, etc.
  - Tools on target
    - Network required
    - Heavy target required for development
  - Closed development environment
Embedded Development: Generation 4

Hardware: Embedded, SBC

Development: Cross
  • Any tool - Any connection - Any target
  • Integrated GUI, Unix & PC

Attributes
  • Tools on host
    • No target resources required
    • Far More Powerful Tools (WindView, CodeTest, …)
  • Open dev. environment, published API
  • Internet is part of dev. environment
    • Support, updates, manuals, etc.
**Embedded Development: Generation 5???

Super-scalable

Communications-centric

Virtual application platform

- Java?

Multi-media

Way-cool development environment

- Much easier to create, debug & re-use code
- Easy for non-programmers to contribute
The RTOS Evolution

*Percent of total software supplied by RTOS vendor in a typical embedded device
Introduction to RTOS

Wind River Systems Inc.        VxWorks
http://www.wrs.com

Integrated Systems Inc.        pSOS
http://www.isi.com

Cygnus Inc. => RedHat        eCos
http://www.cygnus.com => www.redhat.com
# VxWorks

## Real-Time Embedded Applications

<table>
<thead>
<tr>
<th>Graphics</th>
<th>Multiprocessing support</th>
<th>Internet support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java support</td>
<td>POSIX Library</td>
<td>File system</td>
</tr>
</tbody>
</table>

## WindNet Networking

## Core OS

- Wind Microkernel

---

**VxWorks 5.4 Scalable Run-Time System**
Supported Processors

PowerPC
68K, CPU 32
ColdFire
MCORE
80x86 and Pentium
i960
ARM and Strong ARM
MIPS
SH

SPARC
NEC V8xx
M32 R/D
RAD6000
ST 20
TriCore
Wind microkernel

Task management

- multitasking, unlimited number of tasks
- preemptive scheduling and round-robin scheduling (static scheduling)
- fast, deterministic context switch
- 256 priority levels
Wind microkernel

Fast, flexible inter-task communication

- binary, counting and mutual exclusion semaphores with priority inheritance
- message queue
- POSIX pipes, counting semaphores, message queues, signals and scheduling
- control sockets
- shared memory
Wind microkernel

- High scalability
- Incremental linking and loading of components
- Fast, efficient interrupt and exception handling
- Optimized floating-point support
- Dynamic memory management
- System clock and timing facilities
``Board Support Package’’

BSP = Initializing code for hardware device + device driver for peripherals

BSP Developer’s Kit

<table>
<thead>
<tr>
<th>Hardware independent code</th>
<th>Processor dependent code</th>
<th>Device dependent code</th>
</tr>
</thead>
</table>

BSP
A closely coupled multiprocessor support accessory for VxWorks.

Capabilities:

- Support up to 20 CPUs
- Binary and counting semaphores
- FIFO message queues
- Shared memory pools and partitions
- VxMP data structure is located in a shared memory area accessible to all CPUs
- Name service (translate symbol name to object ID)
- User-configurable shared memory pool size
- Support heterogeneous mix of CPU
**VxMP**

**Hardware requirements:**

- Shared memory
- Individual hardware read-write-modify mechanism across the shared memory bus
- CPU interrupt capability for best performance
- Supported architectures:
  - 680x0 and 683xx
  - SPARC
  - SPARClite
  - PPC6xx
  - MIPS
  - i960
VxFusion

VxWorks accessory for loosely coupled configurations and standard IP networking;
An extension of VxWorks message queue, distributed message queue.

Features:
- Media independent design;
- Group multicast/unicast messaging;
- Fault tolerant, locale-transparent operations;
- Heterogeneous environment.

Supported targets:
- Motorola: 68K, CPU32, PowerPC
- Intel x86, Pentium, Pentium Pro
pSOS

Loader | Debug | C/C++ | File System
---|---|---|---
I/O system | BSPs | Memory Management | POSIX Library

pSOS+ Kernel

pSOS 2.5
Supported processors

- PowerPC
- 68K
- ColdFire
- MIPS
- ARM and Strong ARM
- X86 and Pentium
- i960
- SH
- M32/R
- m.core
- NEC v8xx
- ST20
- SPARClite
pSOS+ kernel

Small Real Time multi-tasking kernel;
Preemptive scheduling;
Support memory region for different tasks;
Mutex semaphores and condition variables (priority ceiling)
No interrupt handling is included
Board Support Package

BSP = skeleton device driver code + code for low-level system functions each particular devices requires
pSOS+m kernel

Tightly coupled or distributed processors;
pSOS API + communication and coordination functions;
Fully heterogeneous;
Connection can be any one of shared memory, serial or parallel links, Ethernet implementations;
Dynamic create/modify/delete OS object;
Completely device independent
eCos

- ISO C Library
- Native Kernel C API
- μITRON 3.0 API

- Internal Kernel API
  - Kernel
  - pluggable schedulers, mem alloc, synchronization, timers, interrupts, threads

- Device Drivers
- HAL
Supported processors

Advanced RISC Machines ARM7
Fujitsu SPARClite
Matsushita MN10300
Motorola PowerPC
Toshiba TX39
Hitachi SH3
NEC VR4300
MB8683x series
Intel strong ARM
Kernel

No definition of task, support multi-thread

Interrupt and exception handling

Preemptive scheduling: time-slice scheduler, multi-level queue scheduler, bitmap scheduler and priority inheritance scheduling

Counters and clocks

Mutex, semaphores, condition variable, message box
Hardware Abstraction Layer

Architecture HAL abstracts basic CPU, including:
- interrupt delivery
- context switching
- CPU startup and etc.

Platform HAL abstracts current platform, including
- platform startup
- timer devices
- I/O register access
- interrupt control

Implementation HAL abstracts properties that lie between the above,
- architecture variants
- on-chip devices

The boundaries among them blurs.
## Summary on RTOS

<table>
<thead>
<tr>
<th>Task</th>
<th>VxWorks</th>
<th>pSOS</th>
<th>eCos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduler</td>
<td>Preemptive, static</td>
<td>Preemptive</td>
<td>Preemptive</td>
</tr>
<tr>
<td>Synchronization mechanism</td>
<td>No condition variable</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>POSIX support</td>
<td>Y</td>
<td>Y</td>
<td>Linux</td>
</tr>
<tr>
<td>Scalable</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Custom hw support</td>
<td>BSP</td>
<td>BSP</td>
<td>HAL, I/O package</td>
</tr>
<tr>
<td>Kernel size</td>
<td>-</td>
<td>16KB</td>
<td>-</td>
</tr>
<tr>
<td>Multiprocessor support</td>
<td>VxMP/ VxFusion (accessories)</td>
<td>PSOS+m kernel</td>
<td>None</td>
</tr>
</tbody>
</table>

- **Y** indicates support
- **-** indicates no support
Recall the ``Board Support Package”

BSP = Initializing code for hardware device + device driver for peripherals

BSP Developer’s Kit

<table>
<thead>
<tr>
<th>Hardware independent code</th>
<th>Processor dependent code</th>
<th>Device dependent code</th>
</tr>
</thead>
</table>

BSP
What are device drivers?

- Make the attached device work.
- Insulate the complexities involved in I/O handling.
Proliferation of Interfaces

New Connections
- USB
- 1394
- IrDA
- Wireless

New Models
- JetSend
- Jini
- HTTP / HTML / XML / ???
- Distributed Objects (DCOM, CORBA)
Leads to Proliferation of Device Drivers
Device Driver Characterization

Device Drivers’ Functionalities

- initialization
- data access
- data assignment
- interrupt handling
Device Characterization

Block devices

- fixed data block sizes devices

Character devices

- byte-stream devices

Network device

- manage local area network and wide area network interconnections
I/O Processing Characteristics

Initialization

- make itself known to the kernel
- initialize the interrupt handling
- optional: allocate the temporary memory for device driver
- initialize the hardware device

Front-End Processing

- initiation of an I/O request

Back-End Processing

- handles the completion of I/O operations
Commercial Resources

Aisys DriveWay 3DE
- Motorola MPC860, MC68360, MC68302, AMD E86, Philips XA, 8C651, PIC 16/17

Stenkil MakeApp
- Hitachi H8, SH1, SH3, SH7x, HCAN

Intel’s ApBuilder

Motorola MCUnit

GO DSP Code Composer
- TI DSPs

CoWare
Aysis 3DE DriveWay Features

Extensive documentation: KB help along the way as detailed as a chip manual: traffic.ext, traffic.dwp

CNFG for configuring the chip such as memory and clock.

Gives warning if necessary

Can generate test function

Can insert user code

One file for each peripheral
DriveWay Design Methodology

GUI → .DLL → K.B. → .DWP → Code "generator"

User data → Little generation

Chip specific → more manipulation

Output files → Manipulation of K.B.database
A specific K.B. per chip family

Family of chips

- chip

  - peripherals
    - functional objects (timer, PWM counter)
      - functions
      - physicals (register setting, values, clock rate)
      - actual code
DriveWay Builder

Add chip

Add peripheral

Create skeleton, link to other thins such as GUI

Code reuse in adding a new chip in an existing family, e.g.,
  use code in MPC 860 for MPC 821

Easy to create infrastructure but specifics has to be written
About the code generator (1)

Cut and paste K.B. database

Areas where we can use automation for device driver generation:

- model user specification
- extract useful information for drivers from HDL description of the chip
  - MAP registers
  - interrupt
About the code generator (2)

Why is Aysis not using automation?

- Commercial efficiency
  - e.g., easy to capture user specification from the GUI rather than using a model such as UML or state machine
- HDL code too low level, hard to extract information
**CoWare Interface Synthesis™**

System suggests hardware/software interface protocols

- Handshaking, memory mapped I/O, interrupt scheme, DMA...

Designer selects communication protocols & memory

System synthesizes efficient device drivers and glue logic
Interface Synthesis Example: Memory Mapped I/O

- SW \(ightarrow\) HW
- Port = value;

- HW \rightarrow\) SW
- Device Driver
- Glue Logic

- SW \rightarrow\) HW
- Device Driver
- Glue Logic

- Processor

- Memory Address FFA3

- Compiled on processor

Kurt Keutzer
**SW: Embedded Software Tools**

Diagram showing the process of developing application software for embedded systems. The process includes:

- **USER** providing application source code.
- **Compiler** generating a.out.
- **RTOS** and **a.out** interfacing with the application software.
- **Debugging tools** and **simulator** supporting developer testing and optimization.

Components include:

- CPU
- ROM
- RAM
- ASICs

Kurt Keutzer
ASIC Value Proposition

- 20% area decrease in ASIC portion
- 25% higher performance
- Move to higher level - HDL description at RTL
The Importance of Code Size

Based on base 0.18µm implementation plus code RAM or cache
Xtensa code ~10% smaller than ARM9 Thumb, ~50% smaller than MIPS-Jade, ARM9 and ARC
ARM9-Thumb has reduced performance
RAM/cache density = 8KB/mm²
SW Compiler Value Proposition

- 20% area decrease in RAM portion
- 25% higher performance
- move to higher level - C rather than assembler

20% area decrease over ASIC portion
Memory? StrongARM Processor

Compaq/Digital StrongARM
Compiler Support

BUT, few companies focused on compiler support for embedded systems:

- Cygnus => RedHat
- Tartan => TI
- Green Hills

Why?

Bad ``buying behaviors” – few seats, low ASP’s
Current Status on Compiler Support

Adequate compiler and debugger support in breadth and quality for embedded microprocessors/microcontrollers

- ARM
- MIPS
- Power PC
- Mot family

From

- Cygnus/RedHat
- Manufacturer
- Green Hills

DSP’s still poorly supported

- Tartan acquired by Texas Instruments
- WHY????

NO support for growing generation of special purpose processors:

- TMS320C80
- IXP1200
Recall: Architectural Features of DSPs

Data path configured for DSP
- Fixed-point arithmetic
- MAC- Multiply-accumulate

Multiple memory banks and buses -
- Harvard Architecture
- Multiple data memories

Specialized addressing modes
- Bit-reversed addressing
- Circular buffers

Specialized instruction set and execution control
- Zero-overhead loops
- Support for MAC

Specialized peripherals for DSP
Example: IXP1200

- Host CPU (optional)
- PCI Bus Unit
- SDRAM Memory Unit
- SRAM Memory Unit
- IX Bus Interface Unit
- Ethernet MAC
- ATM, T1/E1
- Another IXP1200

- SDRAM (up to 256 MB)
- SRAM (up to 8 MB)
- Boot ROM (up to 8 MB)
- Peripherals

- PCI Bus 66 Mhz
- FIFO Bus 66 Mhz
- StrongARM core
- Microengine 1
- Microengine 2
- Microengine 3
- Microengine 4
- Microengine 5
- Microengine 6

- Host CPU
- PCI MAC Devices

- 64 bit
- 32 bit
**IXP1200 Network Processor**

- **6 micro-engines**
  - RISC engines
  - 4 contexts/eng
  - 24 threads total
- **IX Bus Interface**
  - packet I/O
  - connect IXPs
    - scalable
- **StrongARM**
  - less critical tasks
- **Hash engine**
  - level 2 lookups
- **PCI interface**
Summary

Embedded software support for microcontrollers and microprocessors is broadly available and of adequate quality:
- RTOS
- Device drivers
- Compilers
- Debuggers

Embedded software support for DSP processors is inadequate:
- Patchy support – many parts lack support
- Quality poor – lags hand coding by 20-100%

Embedded software support for special purpose processors often non-existent

Still in a “build a hardware then write the software” world

Alternatives?
ASIP/Extensible micro DESIGN FLOW

APPLICATION_1

APPLICATION_CODE

RETARGETABLE_COMPILER

OBJECT_CODE

µARCHITECTURE

INSTRUCTION_SET

SIMULATION_MODEL

PERFORMANCE_ANALYSIS

DESIGNER

APPLICATION_2

APPLICATION_CODE

APPLICATION_7
**Tensilica TIE Overview**

- **Configure Base uP**
- **Describe new inst in TIE**
- **Application**

---

- **Processor Generator**
- **Software Generator**
- **Processor Verilog RTL**
- **Software Tools**
- **ASIC flow**
- **uP**
- **Mem**
- **Software compile**
Tensilica TIE Design Cycle

1. Develop application in C/C++
2. Profile and analyze
3. Identify potential new instructions
4. Describe new instructions
5. Generate new software tools
6. Compile and run application
7. Correct?
   - Y: Build the entire processor
   - N: Acceptable?
     - Y: Measure hardware impact
       - Y: Build the entire processor
       - N: Acceptable?

Killian - Tensilica

Run cycle-accurate ISS

Acceptable?

Acceptable?
Conclusions

Full embedded software support for future embedded system “platforms” will be requirement for future embedded system “platforms”

Companies evolving hardware and software together will have a significant competitive advantage

Few examples beginning to emerge- Tensilica, ST Microelectronics