Network System Design

Board Support Package
Outline

• Embedded System Basics
• Embedded HW/SW Development Processes
• Bootloader and Board Support Package
What is an embedded system

- **Embedded system**
  - Embedded systems are computing systems with tightly coupled hardware and software integration
  - Are designed to perform a dedicated function
  - The word embedded reflects the fact that these systems are usually an integral part of a larger system, known as the embedding system
What is an embedded system (Cont.)

- Network Devices
Why embedded system?

- Application specific
  - MMU, FPU, VM, cache, …
- Size
  - Pentium vs. ARM
- Power consumption
  - Intel Pentium vs. ARM
- Price
  - x386 vs. StrongARM
- …
Embedded system inside

- Embedded system decomposition

**Embedded System**

- **H/W**
  - Processor, controller, …
  - Passive components (RLC circuits), …
  - ICs, FPGA, ASIC, …
  - SIP, SoC, …

- **F/W**
  - BIOS, …
  - Bootstrap loader, …
  - Driver, … (might be S/W)

- **S/W**
  - System software
  - Application software
  - Driver
  - EOS

Process to develop an embedded system

- General design flow

30-40% 20-30% 40%

Product Specification | SW/HW Partition | High level design & iteration | H/W Low level design & implementation | H/W testing | H/W & S/W integration | System testing | Product release & maintenance | S/W Low level design & implementation | S/W testing
Embedded system inside (Cont.)

- **Hardware**
  - Microprocessor-based vs. Microcontroller-based
Embedded system inside (Cont.)

- PC architecture
Embedded Processor

- General Purpose CPU vs. Embedded Processor
  - Pentium vs. ARM
    - Cost
    - Power consumption
    - MMU
    - FPU
    - Size
    - Extension
    - ...
Embedded Processor

• Digital signal processor
  – Provide fast, discrete-time, signal-processing instructions
  – Very large instruction word (VLIW)

• DSP vs. microcontroller/microprocessor
  – Processing vs. IO
Embedded Processor

- TI C6x vs. ARM7
Embedded system inside (Cont.)

- Hardware
  - ARM example
Embedded system inside (Cont.)

- Hardware
  - ARM example
Embedded system inside (Cont.)

Source: www.ti.com/omap2
Embedded Processor

• Application specific system processor (ASSP)
  – Special purpose, application oriented
  – MPEG 4 codec, i2Chip
Embedded Processor

- Field Programmable Gate Array (FPGA)
  - a general purpose chip which can be programmed to carry out a specific hardware function
Embedded Processor

• ASIC
  - Application-Specific Integrated Circuit
  - a chip designed for a particular application (as opposed to the integrated circuits that control functions such as RAM in a PC). ASICs are built by connecting existing circuit building blocks in new ways. Since the building blocks already exist in a library, it is much easier to produce a new ASIC than to design a new chip from scratch.
Embedded Processor

- SoC Example: GSM/GPRS terminal SoC
Embedded Processor

- SoC Example: Digital Video System-on-Chip
Embedded System Development
Tools and Platform

• Embedded System Evaluation Board (for example)

Source: www.arm.com
Embedded System Development Tools and Platform

• Embedded System Evaluation Board with FPGA add-on (for example)

Source: www.arm.com
Network processor

- Intel Xscale IXP 425
Network processor
Architecture support for system development

• AMBA
  - Advanced microcontroller bus architecture
  - A interface to allow the ARM to communicate with other on-chip macrocells
  - AHB, advanced high-performance bus
  - ASB, advanced system bus
  - APB, advanced peripheral bus
AMBA architecture
AHB bus scheme
Architecture support for system development

- Hardware system prototyping tools
  - VLSI Tech Inc. Rapid Silicon Prototyping
  - ARM Inc. ARM Integrator
Architecture support for system development

• JTAG boundary scan test architecture
  – IEEE 1149, “Standard Test Access Port and Boundary-Scan Architecture”
  – 5 pins serial protocol
  – JTAG (Joint Test Action Group)
JTAG

• Test signals and TAP controller
  - TRST
    • Test rest
  - TCK
    • Test clock, independent from system clocks
  - TMS
    • Test mode select
  - TDI
    • Test data input, supplies the data or instructions
  - TDO
    • Test data output

• TAP (test access port)
  - A state machine controlled by TMS
  - Control of data register or instruction register
JTAG organization

in enable

in

enable

TDI

TMS

TCK

TRST

device ID reg

bypass reg

instruction reg

TAP controller
TAP state diagram
JTAG extension for macrocell testing

- DRAM control
- ARM macrocell
- video macrocell
- custom logic

- TAP

- ARM test path
- PCB test scan path
Embedded system inside (Cont.)

- Put on target
Embedded System Initialization

Source: Qing Li “real-time concepts for embedded systems”
Embedded System Development Tools and Platform

- ICE/JTAG interface

Source: http://www.keil.com/ulink/
Embedded System Development
Tools and Platform

• ICE/JTAG interface

Source: www.arm.com
Embedded System Development Tools and Platform

- ICE/JTAG interface
Embedded Software Development Tools

Source: Volker Soffel, “Embedded Programming”
Embedded Software

- Bootstrap loader
- Embedded OS
  - Kernel
  - Driver
  - System services (modules)
- Embedded Applications
  - Middleware
  - Application
Embedded Software Development Tools

• Console mode
  – GNU toolchain
    • gcc : a cross-compiler
    • binutils : a set of tools for manipulating binaries
    • glibc : c-lib
    • gdb : debugger (ICE is required)

• IDE
  – ADS
  – GNUPro
  – …
Embedded Software Development Tools

ARM® RealView® Developer Suite v2.1

ARM Developer Suite™ v1.2.1 CD
- ARM Developer Suite (ADS) C/C++ Compiler
- ARM eXtended Debugger (AXD) 1.2
- ARMulator®

RealView Developer Suite v2.1 CD
- RealView RVCT compilation tools
- AXD debugger 1.3
- RealView RVD debugger
- RealView RVISS instruction set simulator

Source: www.arm.com
Embedded Software Development Tools

Source: www.arm.com
Embedded system inside (Cont.)

- Software
  - Boot loader, driver, OS, protocol stacks, applications are all programs

SW development and execution examples
Embedded system inside (Cont.)

- Embedded software
  - Drivers + control programs (8051 examples)
  - O.S + drivers + applications (x86 examples)
  - Bootloader + OS. + applications
Embedded system inside (Cont.)

- Creating an executable image
Embedded system inside (Cont.)

- Simplified schematic and memory map of a target system
Embedded system inside (Cont.)

- Load executable image into target system

Your target image

- Non-volatile parameters, ...
- Flash memory programmer
- program
- ROM
  - 0x0000h
  - 0x001Fh
  - 0x0040h
  - 0x0FFFh
  - 0x1FFFh
  - 0xFFFFh
- Flash
  - ...
  - 0x0040h
  - ...
  - 0xFFFh
  - ...
  - 0x1FFFh
  - ...
  - 0xFFFFh
- RAM
  - 0xFFFFh
Embedded system inside (Cont.)

- Put on target
Embedded System Initialization

Source: Qing Li “real-time concepts for embedded systems”
Embedded System Initialization (Cont.)

• Loading the executable image
  – Programming the entire image into the EEPROM or flash memory
  – Downloading the image over RS-232, network, USB, …
  – Downloading the image through JTAG or BDM interfaces

• Target system tools and image transfer
  – Embedded loader
  – Embedded monitor
  – Target debug agent
Embedded System Initialization (Cont.)

- Target boot scenarios
  - System reset
  - Bootstrap loader
  - Load application image
    - Including
      - EOS
      - Drivers
      - APs
    - From
      - Read-only memory devices on the target
      - Host development system
  - Target system software initialization sequence
    - H/W initialization
    - EOS initialization
    - Application initialization
Embedded System Initialization (Cont.)

- POWER/RESET
  - Processor
  - Fetch & Execute
    - reset vector is at 0x0000h

- Execute the Reset Vector
- Execute Loader
- Initialize Hardware
- Image in ROM/FLASH
  - No
    - Download Image from Host into RAM
  - Yes
    - Copy image from ROM/FLASH to RAM
      - Initialize code during copy
    - Execute the New Image

- Memory Map
  - 0x00000h
  - 0x0001Fh
  - 0x00040h
  - 0x0103Fh
  - 0x10000h
  - 0x103FFh

Source: Qing Li “real-time concepts for embedded system
Embedded System Initialization (Cont.)

• Three common image execution scenarios
  – execute from ROM while using RAM for data
  – execute from RAM after being copied from ROM
  – execute from RAM after being downloaded from a host system
Detail of the program image

- Overview of linkers and the linking process

Source: Qing Li “real-time concepts for embedded systems”
Embedded System Initialization (Cont.)

execute from ROM while using RAM for data

Source: Qing Li “real-time concepts for embedded system”
Embedded System Initialization (Cont.)

execute from RAM after being copied from ROM

Source: Qing Li “real-time concepts for embedded system"
execute from RAM after being downloaded from a host system.

Source: Qing Li “real-time concepts for embedded system”
Embedded System Initialization (Cont.)

Application

Other Embedded Components/Modules

Protocol Stacks (i.e. TCP/IP)

Real-Time Operating System (RTOS)

Board Support Package (BSP)

Source: Qing Li “real-time concepts for embedded systems”
Embedded System Initialization (Cont.)

Source: Qing Li “real-time concepts for embedded systems”
Bootloader

- Bootstrap flow
Bootstrap

• What is a bootloader?
  – The first program being executed by the processor

• Bootloader’s functions
  – Initialize the hardware setting
  – Basic monitor and debugger
  – Pass the control to OS

• Aim to simplify the development process
Bootloader

• Initialize the hardware setting
  – I/O
    – ram size, rom size, external peripherals (such as RS-232 ...) and etc
  – Set up an environment for EOS
    • Memory mapping
    • Stack pointer
    • Others …
Bootloader

• What’s the difference
  – BIOS
  – Bootloader/Bootstrap
  – Loader
  – Multi-boot program
Bootloader

1. Execute from fixed address
2. re-map the memory or move bootloader to RAM
3. Load executable image
4. Pass the control to the executable
• Basic Input and Output System
• Firmware
• For Intel boot architecture
  – Old but backward compatible
• Functions include
  – H/W initialization and parameter configurations
  – Device drivers
  – Debuggers/monitor
  – Load OS/another bootloader
1. Execute from fixed address
2. POST
3. TSR/ISR
4. Load OS/bootloader to RAM
5. Pass control to OS
Loader

- Normally a user level program
- Might use OS system services to access the storage and control the memory allocation
- Load and interpret executable files to memory and ask OS to run it
1. Locate the file in H/D or other storage
2. Parse the executable file
3. Load the program to the memory
3'. Load run time lib
4. Pass the control to the program or fork process

Loader
Linux Boot Example

1. Execute from fixed address
2. POST
3. Select boot device
4. Load MBR 512 bytes to 0xc700
5. Pass control to bootloader (LILO)
Linux Boot Example

1. Execute from fixed address
2. POST
3. Select boot device
4. Load bootsec.S 512 bytes to 0xc700
5. Pass control to bootsec.S
6. Move 0x90000
7. Read two more blocks
8. Load compressed kernel to 0x10000
9. Pass control to setup.s

CPU

BIOS

RAM

Bootsec.S

Compressed kernel

Setup.s

Video.s

floppy
iBoot

• Open source bootloader for Intel PXA 2XX
• Download from
  – [http://download.intrinsyc.com/supported/iso/](http://download.intrinsyc.com/supported/iso/)
• Code browsing tool
  – LXR
    • [http://sourceforge.net/projects/lxr](http://sourceforge.net/projects/lxr)
int
c_main(void)
{
    //~init all the devices we need.
    init_status();
    //init_serial(SERIAL_BAUD_115200);
    init_serial(SERIAL_BAUD_38400);
    
    ~
    //itc_printf("\r\n"
    "****************************************************************
    "** Intrinsyc Bootloader (IBoot)                          **\r\n"
    
    ~
    init_flash();
    init_crc_table();
    init_timer();
    inc_led();
    
    ~
    if (!init_i2c ())
    {
        itc_printf("ERROR: There is a problem initializing I2C
Controller\r\n");
    }
    ...
}
...  
  \#ifdef SMSC91C111_ETHERNET
  if (!init_eeprom ()){
    itc_printf("ERROR: There is a problem initializing the EEPROM\r\n");
  }
\#endif

~
init_ethernet(status.macaddr);

  \// Seed the random number generator with as much entropy as we have
  srand(get_time_timer() ^ ((status.macaddr[2] << 16) |
  status.macaddr[1]));

  \// Bring up the user interface. This will return on a timeout or through
  \// user
  \// direction. This will not return if the user tells us to boot.
  init_ui(UI_TIMEOUT, mode_default);
  inc_led();
  init_os();
  return 0;
}
Blob

- Open source bootloader for StrongARM
- Download from
  - http://www.lart.tudelft.nl/lartware/blob/
int main(void)
{
   ~
   led_on();
~
   SerialInit(baud9k6);
   TimerInit();
~
   SerialOutputString(PACKAGE " version " VERSION "\n" "Copyright (C) 1999 2000 2001 "
~
   get_memory_map();
~
   Reload("blob");
   Reload("kernel");
   Reload("ramdisk");
~
   SerialOutputString("Running from ");
   if(RunningFromInternal())
      SerialOutputString("internal");
   else
      SerialOutputString("external");
...
...  
/* wait 10 seconds before starting autoboot */
SerialOutputString("Autoboot in progress, press any key to stop ");
for(i = 0; i < 10; i++) {
    SerialOutputByte('.');
    retval = SerialInputBlock(commandline, 1, 1);
}
if(retval == 0) {
    boot_linux(commandline);
}
for(;;) {
    if(numRead > 0) {
        if(MyStrNCmp(commandline, "boot", 4) == 0) {
            boot_linux(commandline + 4);
        } else if(MyStrNCmp(commandline, "clock", 5) == 0) {
            SetClock(commandline + 5);
        } else if(MyStrNCmp(commandline, "download ", 9) == 0) {
            return 0;
        } /* main */
BSP

• A collection of device initialization and control routines specific to a particular type of board or collection of boards

• The BSP (Board Support Package) is the set of files that can be customized to run operating system on specific board architectures

• The BSP contains the boot and the generic and processor specific drivers required for your board
• BSP components
BSP example

- WinCE Platform Builder
BSP example

- WinCE Platform Builder (Cont.)
BSP example

- WinCE Platform Builder (Cont.)
BSP example
How Operating System Works?

• Drive and fully utilize the CPU
  – How schedule works

• Drive and fully utilize the device
  – How I/O works

• Provide APs to access the system services
  – How system call works
Kernel Image Structure

For i386
- Setup.S
- Video.S
- Head.S
- Misc.o
- Kernel.o
- Initrd (ramdisk.gz)

For ARM
- Real-mode operations
- Simple video driver
- Memory setup
- Decompress

Kernel Image Structure

- **Setup.S**
- **Video.S**
- **Head.S**
- **Misc.o**
- **Kernel.o**
- **Initrd (ramdisk.gz)**

**Head-armv.S**

- setup_arch()
- setup_processor()
- setup_architecture()
- init_bootmem_node(...)
- free_bootmem(...)
- paging_init(...),
- trap_init()
- init_IRQ()
- sched_init()
- softirq_init()
- time_init();
- console_init()
- init_modules()
- kmem_cache_init(),
- mem_init(),
- mount initrd (maybe)
- ...
- cpu_idle();
Kernel Boot Procedures

1. Jump from bootloader

2. Loader compressed kernel image to memory

3. Decompress kernel image

4. Initial ram disk
Kernel Boot Procedures (Cont.)

5. Mount ramdisk as root

6. Execute /initrc

7. Load system modules or drivers

8. Drive new device or file system and replace root FS

9. Clear up memory

RAM

kernel

Initrd (ramdisk.gz)

Kernel.o

Misc.o

Head.S

flash

Initrd (ramdisk.gz)

Kernel.o

Misc.o

Head.S
Why Schedule

My program...

... read()
... 

CPU

FS processing

I/O request processing

Issue I/O request to device

Wait for response

Receive I/O response

FS processing
Why Schedule (Cont.)

My program A

... read() ...

CPU

My program B

...
Why Schedule (Cont.)

My program A

read()  
...  

My program B

Context switch

CPU

scheduler
How Basic Schedule Works

1. read()
2. Context switch
3. scheduler
4. My program B

My program A

TCB A
TCB B
...
How Scheduler Works?

User program A
xyz();

User program B
abc();

Scheduler

PCB Table
Program A
Program B
...

Systemcall handler

System Call Service Routine XXX

CPU

PC

INT

Timer

PCB For A

PCB For B
How I/O Interrupt Works?

![Diagram showing the flow of interrupt signals from CPU to various devices through the system bus.]

- **CPU**
- **INT**
- **System Bus**
- **Command & status registers**
  - **Programmable I/O Controller**
  - **HD**
  - **Printer**
  - **Mouse**
  - **Memory**

How I/O Interrupt Works?

For example

\[ \text{PC} = (\text{INT #}) \times \text{size of vector} + \text{interrupt base address} \]
How System Call Works?

User program A

```
xyz();
```

System Stack

```
0x0051
```

Systemcall handler

```
X1: 0x01FF
X2: 0x02FF
X3: 0x0543
X4: 0x1234
```

System Call Service Routine #X2

```
xyz()
{
...
Systemcall number(X2) INT 80
...}
RETI
```
How Scheduler Works?
with I/O Device Driver

- User program A
  xyz();
- User program B
  abc();

H/D

CPU

INT

PC

Timer

Scheduler

PCB Table
  Program A
  Program B
  ...

Interrupt handler

ISR XXX

Request queue

Radio Access Software & System Lab.
How Scheduler Works? 
with System Module Driver
Why Embedded OS?

• Configurability
• Customization
• Size
• Power consumption
• Real-time characteristics
• I/O intensive
• Special purpose …
Questions?