Porting WinCE 5.0 to your NXP BlueStreak MCU Board

Durgesh Pattamatta
Staff Systems Engineer, Product Line Microcontrollers, NXP Semiconductors.
http://www.standardics.nxp.com/support/microcontrollers/
October 2nd, 2007
Agenda

- Overview of Windows CE 5.0 Architecture
- Overview of Windows CE development
  - Development Tools
- Developing a Board Support Package
  - Boot-loader
  - OEM Abstraction Layer (OAL)
  - Configuration files
  - Device drivers
- Porting LH7A404-SDK BSP to custom boards
Overview of Windows CE 5.0
Architecture
Windows CE 5.0
Architecture – Layer view
Windows CE 5.0
Architecture – Process view

GWES.exe
- GUI process
- Display driver
- keyboard/mouse
- Touch-screen

Device.exe
- Most device drivers
- USB
- SD/MMC
- Audio
- HKEY_LOCAL_MACHINE\Drivers\BuiltIn
- ...

FileSync.exe
- Storage Manager
- FATFS
- MS Partition
- Registry store
- HKEY_LOCAL_MACHINE\System\StorageManager

Nk.exe (OAL + Kernel library)

Explorer.exe

Services.exe

Shell.exe

Custom Application.exe
## Windows CE 5.0

### Architecture – Memory Architecture

<table>
<thead>
<tr>
<th>Kernel Space</th>
<th>Address Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0xE0000000 - 0xFFFFFFFF</td>
<td>Trap area, others</td>
</tr>
<tr>
<td></td>
<td>0xC4000000 - 0xE0000000</td>
<td>Statically Mapped additional OEM virtual address</td>
</tr>
<tr>
<td></td>
<td>0xC2000000 - 0xC4000000</td>
<td>Slot 97: NK.exe (secure slot)</td>
</tr>
<tr>
<td></td>
<td>0xC0000000 - 0xC2000000</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>0xA0000000 - 0xC0000000</td>
<td>Statically Mapped virtual address - UNCAHEDED</td>
</tr>
<tr>
<td></td>
<td>0x80000000 - 0xA0000000</td>
<td>Statically Mapped virtual address - CACHED</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User Space</th>
<th>Address Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x7E000000 - 0x80000000</td>
<td>Slot 63: Resource mappings</td>
</tr>
<tr>
<td></td>
<td>0x42000000 - 0x7E000000</td>
<td>Slot 33 – 62: Object store &amp; memory mapped files</td>
</tr>
<tr>
<td></td>
<td>0x04000000 - 0x42000000</td>
<td>Slot 2 – 32: Processes</td>
</tr>
<tr>
<td></td>
<td>0x02000000 - 0x04000000</td>
<td>Slot 1: XIP DLLs</td>
</tr>
<tr>
<td></td>
<td>0x00000000 - 0x02000000</td>
<td>Slot 0: Current Process</td>
</tr>
</tbody>
</table>
Overview of Windows CE 5.0
Development Process
Overview of Windows CE Development

To develop an OS design based on the Microsoft® Windows® CE OS, you must complete the following major tasks:

- Create a board support package (BSP) for your specific target device.
- Create an OS design, based on a standard or custom BSP, that you can use to create a run-time image that you can download to a standard development board (SDB), which is also called a hardware platform.
- Create and customize device drivers for your target BSP.
- Customize the OS design with additional projects and Catalog items.
- Build the run-time image, download it to your SDB, and debug the run-time image by using the debugging tools found in the Microsoft Platform Builder integrated development environment (IDE).
- When the run-time image is complete, export a software development kit (SDK) for your application developers.
Overview of Windows CE Development

<table>
<thead>
<tr>
<th>Device Bring-up</th>
<th>Platform Development</th>
<th>Application Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Design</td>
<td>Develop Custom Drivers</td>
<td>Custom WinCE OS features</td>
</tr>
<tr>
<td>Debug Hardware</td>
<td>Boot Minimum Kernel</td>
<td>Test &amp; Integration</td>
</tr>
<tr>
<td>Create Boot loader</td>
<td>Develop &amp; Integrate OS features</td>
<td></td>
</tr>
<tr>
<td>OAL Development</td>
<td></td>
<td>Application &amp; Middleware</td>
</tr>
<tr>
<td>BSP Development</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
WinCE 5.0 Development Tools

- Platform Builder 5.0
  - Develop Board Support package
  - Develop Device drivers
  - Develop Custom OS design
  - Develop software development kit (SDK) for Visual Studio
  - Develop Applications

- eMbedded Visual C++ 4.0
  - Develop Applications
  - Limited support

- Visual Studio 2005 Professional
  - Develop Applications
  - Supports WinCE 6.0 Platform builder plugin
WinCE 5.0 Development Tools

- WinCE based OS-design
- OEM Adaptation Layer (OAL)
- Runtime libraries

Custom SDK
- Header & libraries
- SDK documentation
- Runtime files
- OS design extensions
- Platform manager

Microsoft Embedded development software
- Microsoft eMbedded Visual C++ 4.0
- Microsoft Visual Studio 2005
Developing a Board Support Package
Developing Board Support Package

- Developing a Boot-loader
- Developing an OEM Adaptation Layer (OAL)
- Creating Device drivers
- Creating OS design configuration files
Developing Board Support Package

1. Board bring-up using ICE, ROM Monitors
2. Clone Reference BSP
3. Develop Boot loader
4. Develop OAL (Tiny Kernel)
5. Package BSP or Design custom OS
6. Power management
7. Add Device drivers

Diagram shows the flow from board bring-up to adding device drivers while incorporating power management and BSP or custom OS design.
Developing a Board Support Package

Boot-loader
Developing a Boot-loader

- A boot loader manages the boot process of the target device by initializing the target device, downloading the run-time image, and booting the run-time image on the target device.

- Using the boot loader during the board support package (BSP) development process saves time.

- Final products may not have the boot loader
  - hardware platforms that need to perform pre-boot tasks, such as run-time image updates, might include the boot loader in the final product
Developing a Boot-loader

- Implement StartUp function
  - Set SVC mode, clear caches & TLBs, disable interrupts,
  - Initialize memory controller & PLLs

- Implement OEM functions called by the BLCOMMON library
  - Control flow, debug, ethernet, flash, initialization, timer

- Create a sources file and makefile file

- Create the .bib file, which will be used by Romimage.exe to convert the
  boot loader .exe file into .bin and .nb0 files

- Check Platform builder help for complete list of steps to implement
Developing a Board Support Package

OEM Adaptation Layer (OAL)
Developing an OEM Adaptation Layer (OAL)

- An OEM adaptation layer (OAL) is a layer of code between the Windows CE kernel and the hardware of your target device.
- Contains code to handle interrupts, timers, power management, bus abstraction, generic I/O control codes (IOCTLS)
- Creating the OAL is one of the more complex tasks in the process of getting a Windows CE–based OS to run on a new hardware platform. In general, the easiest way to create an OAL is to copy the OAL implementation from a working OS design, and then modify it to suit the specific requirements of your hardware platform.
Developing an OEM Adaptation Layer (OAL)

- Implement the **Startup** function for the OAL
- Create a sources file and a makefile
- Implement required kernel independent transport layer (KITL) functions (http://msdn2.microsoft.com/en-us/library/ms903776.aspx)
- Implement ILTiming and profiling function
- Create config.bib, platform.bib, platform.reg, platform.db, platform.dat files.
- Check Platform builder help for complete list of steps to implement (http://msdn2.microsoft.com/en-us/library/ms903969.aspx)
Developing a Board Support Package

Device Drivers
Device Drivers

- A device driver is software that abstracts the functionality of a physical (e.g., UART, network, audio) or virtual device (e.g., File system).

- Many Windows CE device drivers implement the stream interface
  - XXX_Open (Device Manager), XXX_Close (Device Manager), XXX_Read (Device Manager), and XXX_Write (Device Manager).

- Determine the driver type
  - **FileSys.exe** loads file system drivers
  - **Device.exe** loads audio drivers, battery drivers, keyboard drivers, mouse drivers, NDIS drivers, notification LED drivers, serial drivers, PC Card drivers, USB drivers, and any other driver that exposes the stream interface.
  - **GWES.exe** loads a device driver if GWES is the only client of a driver. Device drivers loaded by GWES present a standard set of functionality for all similar devices. Drivers that GWES loads might expose the stream interface or they might expose other interfaces. Having alternatives make accessing the drivers much faster. GWES loads display drivers, printer drivers, and touch screen drivers.
Device Drivers
Concepts

- Monolithic vs Layered
  - Most Windows CE device drivers consist of a platform dependent driver (PDD) and a model device driver (MDD). A monolithic driver combines all PDDs and MDDs into one driver.
  - MDDs
    - Code common to all drivers
    - Call PDD functions
    - Expose device driver interface (DDI) functions to the operating system (OS)
    - Contain any interrupt service threads (ISTs)
  - PDDs
    - Hardware platform specific code
    - Work with specific MDD implementations
    - Device driver service-provider interface (DDSI) functions that the MDD calls

- Check (http://msdn2.microsoft.com/en-us/library/ms923714.aspx)
Developing a Board Support Package

OS design Configuration Files
OS Design Configuration Files
Run-time Image Configuration Files

- Run-time image configuration files are used by a number of tools that are called by the Make Binary Image Tool to create the run-time image.

- Binary Image Builder File (.bib)
  - Defines the modules and files to be included in the run-time image.

- Registry File (.reg)
  - Defines the registry keys and values for a run-time image created during a cold boot.

- File System File (.dat)
  - Defines the RAM file system directories, files, and links for a run-time image created during a cold boot.

- Database File (.db)
  - Defines the databases to be included in the object store of a run-time image created during a cold boot.

- String File (.str)
  - Defines locale-specific string replacements for text that is visible to a user in .reg, .dat, and .db files. Each line in the .str file must end with a <CR> to enable correct processing.
OS Design Configuration Files
Source Code Configuration Files

- Source code configuration files are used by the Build tool (Build.exe) to build modules and features and to build the source code for your run-time image.

- Dirs File (dirs)
  - A text file that specifies the subdirectories that contain source code to be built.

- Makefile File (makefile)
  - Each subdirectory in a source code tree should have this file. In platform builder always refers to common makefile file, Makefile.def.

- Module-definition File (.def)
  - Contains the statements defining an executable or dynamic-link library. Used by the linker tool to create .exe files and DLL files.

- Sources File (sources)
  - Contains the macro variables and source files list needed to build source code.
Platform Builder 5.0 - Build System
Catalog Item Files

- After you create BSP or project, you can use a Catalog item (.cec) file to associate it with a Catalog item that you can display in the Platform Builder Catalog.

- For more information (http://msdn2.microsoft.com/en-us/library/aa448620.aspx)
Porting LH7A404-SDK BSP to custom boards
Porting LH7A404-SDK BSP to custom boards

- Boot-loader changes
- OAL changes
- Device driver changes
  - Display driver
  - USB function
  - USB host
  - SD/MMC
  - Audio
  - Touch-screen
- Modifying run-time image configuration files
LH7A404 Customer BSP

You could use Microsoft’s “Cloning a BSP” method or modify existing LH7A404 BSP with your board specific changes.

Cloning

- Clean-up and modify the reference to BSP_KEV7A404_BOARD, BSP_LPD7A404_BOARD and BSP_FIREFLY_BOARD with your board specific changes.
- Merging new BSP release from NXP will be little confusing.

Modify Existing BSP

- Add new reference BSP_CUSTOMER_BOARD.
- Add new board type in lh7a404.cec file
- Search for BSP_KEV7A404_BOARD, BSP_LPD7A404_BOARD & BSP_FIREFLY_BOARD references and code for BSP_CUSTOMER_BOARD
- Merging new BSP release from NXP will be easier.
Boot-loader changes

- **Cloning**
  - Rename `wince500/platform/lh7a404/src/bootloader/kevboot` or `fireflyboot` to your board name.
  - Modify the functions which are different for your board.

- **Modify existing BSP**
  - Create new directory for your board under `wince500/platform/lh7a404/src/bootloader/`
  - Change `wince500/platform/lh7a404/src/bootloader/dirs` file to add your directory
  - Add bootloader code use `kevboot` as an example.
OAL Changes

- Create new `intr_cust.c` file similar to `intr_lpd7a404.c` file in `WINCE500\PLATFORM\CUSTLH7A404\SRC\KERNEL\OAL` directory.
  - Add the new file to sources file
  - Contains board specific interrupt handler

- If you have custom IOCTLS, create a file similar to `ioctl_firefly.c` file.

- Modify all header files in `WINCE500\PLATFORM\CUSTLH7A404\SRC\INC` except header files with names `lh7a404_*.h`

- Search for `BSP_KEV7A404_BOARD`, `BSP_LPD7A404_BOARD` & `BSP_FIREFLY_BOARD` references and add code for your board if those function effect your board.

- KITL changes
  - Modify `WINCE500\PLATFORM\CUSTLH7A404\SRC\KERNEL\KERNKITL\eth_drv.c` file with ethernet driver specific to your board.
  - If don’t plan to use KITL in your developement process ignore this step.
Device Drivers

Display

- Create new cust_panel.c file similar to kev_panel.c file in WINCE500\PLATFORM\CUSTLH7A404\SRC\DRIVERS\DISPLAY directory.
  - Add the new file to sources file
  - Contains board specific LCD panel parameters
Device Drivers

USB function

- Create new cust_usbfn.cpp file similar to kev_usbfn.cpp file in WINCE500\PLATFORM\CUSTLH7A404\SRC\DRIVERS\USBFN directory.
  - Add the new file to sources file
  - Contains board specific implementation
    - Dplus pull-up handling
    - Cable detect handling
Device Drivers

USB host

- Modify
  WINCE500\PLATFORM\LH7A404\SRC\DRIVERS\USBHCD\PDD\system.c file
  - With board specific 5V regulator handling
  - Search for BSP_FIREFLY_BOARD in that file to Firefly board handlers as an example
Device Drivers
SD/MMC

- Create new cust_impl.c file similar to kev_impl.c file in WINCE500\PLATFORM\CUSTLH7A404\SRC\DRIVERS\SDHC directory.
  - Add the new file to sources file
  - Contains board specific implementation
    - Card detect handling (insert/remove)
    - Card write protect handling
Device Drivers

Audio

- Create new cust_XXXX.c file similar to kev_cs4201.c file in WINCE500\PLATFORM\CUSTLH7A404\SRC\DRIVERS\WADEVDEV directory.
  - Add the new file to sources file
  - Contains board specific AC97 CODEC driver implementation
    - Codec vendor specific register handlers
    - Codec specific initialization routines
    - Codec specific volume, sample format and power handlers
Device Drivers

Touch-screen

- Create new cust_board.cpp file similar to kev_board.cpp file in WINCE500\PLATFORM\CUSTLH7A404\SRC\DRIVERS\TOUCH directory.
  - Add the new file to sources file
  - Contains board specific implementation
    - Change initialization routine based on 4-wire or 5-wire touch screen used on the device
    - Add other routines to handle any other Analog-to-digital-converter functions implemented on board. Check firefly_board.cpp file as an example to see how ambient light sensor and battery monitor ADC functions are implemented.
Modifying run-time image configuration files

- Update WINCE500\PLATFORM\LH7A404\FILES\config.bib with on board SDRAM sizes & planned OS image sizes.
- Update WINCE500\PLATFORM\LH7A404\FILES\platform.bib with any new on board driver files.
- Update WINCE500\PLATFORM\LH7A404\FILES\platform.reg with any board specific changes to various driver registry values.
References

- Platform Builder 5.0 Help
- Windows Embedded CE How-tos and Tutorials
- Big Book of BSP