LoCE Windows Embedded CE BSP User's Guide

BSP Documentation

Logic Product Development
Published: September 2003
Last Revised: November 2007

Abstract

This document is a user's guide for Logic's LoCE Windows Embedded CE Board Support Package (BSP).

REVISION HISTORY

REV	EDITOR	DESCRIPTION	APPROVAL	DATE
1	Michael Erickson	Original Version	ME	09/12/03
A	James Wicks	Editing and Formatting: Beta Release	ME	09/12/03
B	James Wicks	Added Number Format to Headings	JW	10/08/03
C	Aaron Stewart	Added new bootscript commands and updated for relocation images	HAR	05/20/04
D	Chris Rempel Mike Aanenson	-Rewrote for LoCE 2.x for WinCE -Referred Readers to 'Readme.txt' Notes -Added Boot Parameters Appendix -Added Resources Appendix -Added Definitions Appendix -Added information regarding Image types -Updated for use with LogicLoader 2.x -Added LogicLoader 'download' options -Updated LoCE Directory Structure -Updated 'Installation' changes for LoCE 2.x	MAA	09/06/05
E	Jed Anderson Mike Aanenson Erik Seres	-General formatting and grammatical editing -Added note about supported vs. unsupported BSP components in Section 3 -Updated Section 5.4.3 "Using Boot Parameters in Drivers" -Updated Section 6: Removed placeholder categories; Added IOCTLs info to Section 6.2 -Added Figure Titles where needed -Added IOCTLs Appendix	ES	07/10/06
F	Aaron Stewart Jed Anderson	-Generalized terminology for all SOMs and Development Kits -Added "Embedded" to Windows Embedded CE name	JCA	12/29/06
G	Jed Anderson	-Added "coproc" to "Boot Parameters Appendix"	ES	07/25/07
H	Jed Anderson	-Section 5.4.2 & Appendix C: Corrected typo in "skiplcdcinit" parameter, originally was mistyped as "skiplcdinit"; -Formatting changes throughout	RGL	11/16/07

Table of Contents

1 Introduction 4
2 Releases and Naming Conventions 5
3 Installation 7
3.1 LoCE Folder Structure 9
4 Using LoCE with Platform Builder 10
4.1 Image Types 10
4.1.1 RAM Images 10
4.1.2 Flash Images 10
4.1.3 Relocation Images 10
4.2 Configuring OS Memory Map 11
4.2.1 Memory Structure 12
4.3 Drivers and Components 13
4.3.1 LoCE Binary Driver Files 13
4.3.2 Add a Component to a Platform Builder Workspace 13
4.4 Creating an OS Image 14
4.4.1 Building an Image 14
4.4.2 Differences between the LoCE BSP and other BSPs 21
4.4.3 Building a Flash or Relocation OS Image 22
4.4.4 Building a relocation image 23
5 Launching Windows Embedded CE 25
5.1 Using LogicLoader to Download the OS Image (NK.bin) 25
5.1.1 Using LogicLoader to Download the OS Image to RAM over Ethernet 25
5.1.2 Using LogicLoader to Download the OS Image to RAM over Serial 29
5.1.3 Using LogicLoader to Download the OS Image to RAM from a CompactFlash Card 31
5.2 Using LogicLoader to Boot the LoCE BSP Created Image 33
5.3 Booting Different Image Types (Flash/Relocation Images) 35
5.4 Boot Parameters 38
5.4.1 Use of Boot Parameters in LogicLoader 38
5.4.2 Boot Parameter Table 38
5.4.3 Using Boot Parameters in Drivers 39
5.5 Debugging over KITL using Platform Builder and LogicLoader 39
5.5.1 Connecting to Images in RAM 39
6 Development with LoCE 45
6.1 LoCE Directory Structure 45
6.2 LoCE Kernel IOCTLs 45
6.2.1 IOCTL_LPD_GET_HAL_PARAM_VALUE 45
6.2.2 IOCTL_HAL_REBOOT 47
7 Using LoCE with Embedded Visual C++ 49
7.1 Creating an Software Development Kit 49
7.2 Using Embedded Visual C++ 51
8 Going Forward 52
Appendix A: Definitions 53
Appendix B: Additional Resources 54
Microsoft Support 54
Books 54
Web Resources 54
Logic Product Development Support 54
Appendix C: Boot Parameter Definitions 55
Platform Independent Boot Parameters 55
Platform Dependent Boot Parameters 56
Driver Dependent Boot Parameters 59
Appendix D: Supported Kernel IOCTLs 60

1 Introduction

Logic Product Development's LoCE Board Support Package (BSP) is a production ready solution for building Windows Embedded CE Operating System images for use on Logic Product Development's production-ready System on Module (SOM) hardware platforms. The LoCE BSP is a cross-platform BSP that has a common source code base for all supported hardware platforms, and is released in a multi-component binary format for each CPU type.

The LoCE BSP contains the OAL (OEM Adaptation Layer), drivers, and build environment for use with Microsoft Windows Embedded CE Platform Builder to create customized Windows Embedded CE images for a target hardware platform. The components of the LoCE BSP are modular so that each driver or OAL component can generally be modified and released independently of the other components.

The remainder of this document describes various facets of the LoCE BSP and, in general, expects an intermediate knowledge of Windows Embedded CE development by the reader. Topics that are related to the LoCE BSP, but are not focused on BSP usage, development, or debugging within Platform Builder, can be found in other associated documentation (QuickStart guides, evaluation guides, application notes, white papers, etc.).

For more details regarding Windows Embedded CE architecture, please refer to the following link on MSDN: Windows Embedded CE Basic Architecture Diagram.

2 Releases and Naming Conventions

There are essentially three different component types in Logic's LoCE BSP. Each is released with a version number and is packaged as a self-installing application with a readme file that details the version number changes, known issues, dependencies, and other useful instructions.

Base BSP

Description:
This is the infrastructure of the LoCE BSP. It creates the folder structure, build files, and header files. All kernel and driver files are installed into this folder structure and rely on it to be built correctly into an image.

Important Note: This must be installed before kernel or driver components.

Naming Convention:
lpd_loce_bsp_<version_number>

Example:
lpd_loce_bsp_2_1_0

Installation Location:
x:\WINCE500\PLATFORM\LoCE

Kernel

Description:
The kernel component is the OAL (OEM Adaptation Layer) for the specific device.

Naming Convention:
lpd_kernel_<cpu_name>_<ceversion>_<version_number>

Example:
lpd_kernel_a404_50_2_1_5_0
lpd_kernel_pxa270_50_2_1_5_0
lpd_kernel_7760_50_2_1_5_0

Binary Installation Location:
x:\WINCE500\PLATFORM\LoCE\bin\lpd_kernels

Driver

Description:
Each driver component is specific to a peripheral on a hardware platform.

Naming Convention:
lpd_<peripheral_name/type>_<cpu_name>_<ceversion>_<version_number>

Examples:
lpd_lh7a400_uart_a400_50_1_3_0
lpd_ti16c552_a400_1_1_0
lpd_lh7a40x_lcdc_a40x_50_1_3_1
lpd_pxa270_usbf_pxa270_50_0_1_0

Binary Installation Location:
x:\WINCE500\PLATFORM\LoCE\bin\lpd_drivers

Subsequent releases of components may be installed side by side with previous version releases. Each of these releases should share the same name with the exception of the version number and possibly the release type. By doing this, customers may chose whether or not to use upgraded versions of the components provided by Logic Product Development.

3 Installation

The LoCE BSP for Windows Embedded CE requires that Microsoft Windows Embedded CE Platform Builder be installed on the development workstation before installing BSP components. Ensure that all current Quick Fix Engineering (QFE) patches from Microsoft for Platform Builder and Windows Embedded CE have been applied before working with the LoCE BSP.

The LoCE BSP is a versatile multi-component package designed for simple update procedures and support. Minimally, the *LoCE Base BSP* and *LoCE Kernel* components are needed to successfully build a Windows Embedded CE Operating System (OS) image with Platform Builder. For more advanced OS images, applicable driver components for the selected hardware platform should also be used.

Please obtain the latest LoCE BSP components from Logic Product Development's website at http://www.logicpd.com/auth/. To download the LoCE BSP, you will need to create an account, register a development kit, and meet the requirements stipulated in the *WinCE BSP License Agreement* (available for download after registration).

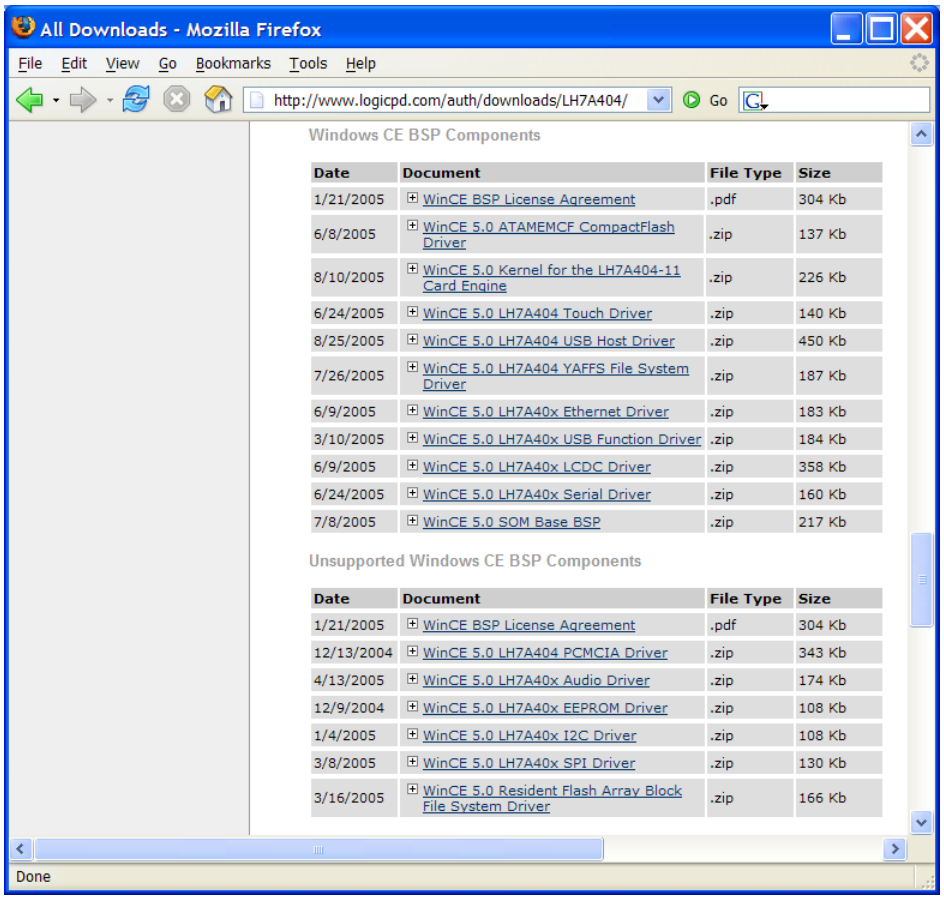


Figure 3.1: LoCE BSP Downloads

LoCE BSP components are released in binary format as a self-installing executable or in the .msi file format. Components are typically packaged in a file archive along with release notes for the component. There are both "supported" and "unsupported" BSP components available for download. Please see Logic's document, *Unsupported Windows Embedded CE BSP Components Readme*, for an explanation of the differences between "supported" and "unsupported" components. To install LoCE BSP components, take the following steps:

1. Download and install the *LoCE Base BSP* component.
2. Download and install the *LoCE WinCE Kernel* component for the target SOM.
3. Download and install applicable peripheral driver components for the target SOM.

Important Note: The LoCE Base BSP must be the first component installed. If the LoCE Base BSP is ever reinstalled, all other LoCE BSP components must also be reinstalled. If the other LoCE BSP components are not reinstalled following a LoCE Base BSP update, the OS image build process will fail.

3.1 LoCE Folder Structure

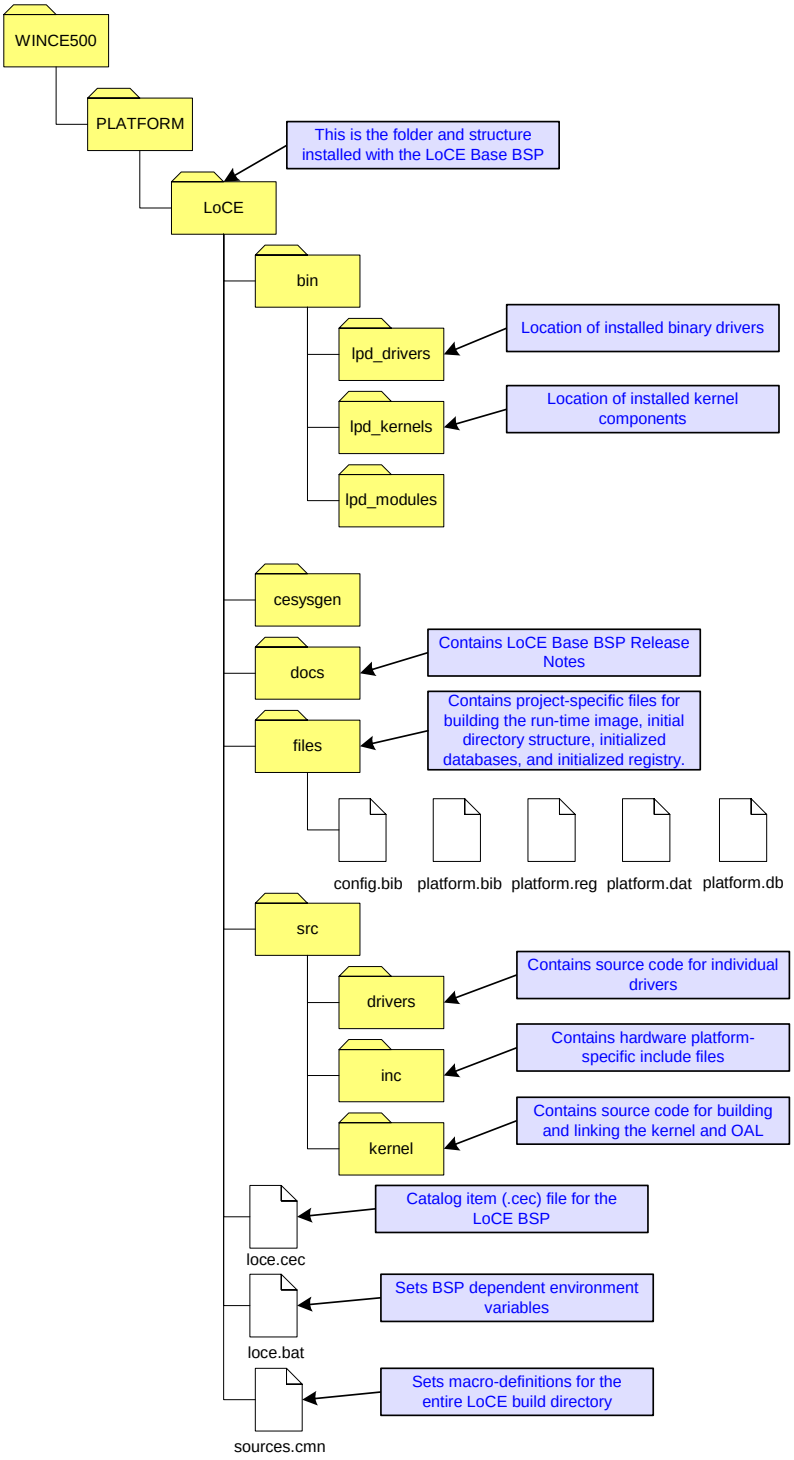


Figure 3.2: LoCE Folder Structure

4 Using LoCE with Platform Builder

When LoCE BSP components have been installed on a system with Windows Embedded CE Platform Builder, the BSP components are available for use in building customized Windows Embedded CE OS images for the target hardware platform.

Installed LoCE BSP components are available and viewable in the Platform Builder catalog view under the “Third Party” category. Once installed, the components can be removed or re-installed by using the Platform Builder “File -> Manage Catalog Items…” menu selection.

Microsoft Windows Embedded CE Platform Builder User Guide

4.1 Image Types

Windows Embedded CE OS images built with the LoCE BSP can be configured for three different storage and executable formats. These formats, or image types, can be one of the following: flash, RAM, or relocation (on supported platforms). These image types are described more thoroughly below and a diagram of the memory structures can be found here.

4.1.1 RAM Images

RAM images refer to Windows Embedded CE OS images that are configured to be stored in and run from system RAM memory (e.g., system SDRAM).

This is a popular OS image type for both development and production environments. The main drawback to this image type is that the OS image must be “unpacked” or downloaded from some non-volatile memory or communications medium into system RAM before running. However, once placed into system RAM, the boot performance and executable performance of the system is the fastest that can be expected.

To configure an image to be a RAM image type, use the default state of a new Platform Builder workspace. Essentially, neither the IMGFLASH nor LOCE_RELOCATE_FROM_FLASH environment variable can be set during the “makeimg” build phase.

4.1.2 Flash Images

Flash images refer to Windows Embedded CE OS images that are configured to be stored in and run from directly addressable flash memory (e.g., NOR flash memory). This image type is often referred to as an XIP image.

Although the LoCE BSP makes this image type available, it is rarely used in production systems unless a specific set of criteria apply. The main reason for its infrequent use is because execution speeds out of directly-accessible flash memory are typically much slower than execution speeds out of synchronous RAM technologies. Therefore, if a flash image is used, its performance will be much slower than running the same image out of RAM (SDRAM, etc.). However, reasons to use flash images include instances when the need for fastest start-up speeds can be traded for slower full-boot time or when system RAM usage does not allow running an OS image from RAM.

To configure an image to be a flash image type, the environment variable “IMGFLASH” must be set to ‘1’ for the build environment during the “makeimg” build phase. The easiest way to set this variable is to check the box by “Write Run-time Image to Flash Memory (IMGFLASH=1)” in the Platform Builder “Platform -> Settings…” menu selection under the “Build Options” tab.

4.1.3 Relocation Images

Relocation images refer to Windows Embedded CE OS images that are stored in directly-accessible flash memory (e.g., NOR flash), but self-relocate and finally execute from system RAM memory.

This is a popular OS image type for production environments due to its non-volatility, fast relocation times, and fast execution times. The main drawback to this image type is that the OS image must be stored in a non-checksum fashion, directly in NOR-type flash memory.

To configure an image to be a relocation image type, set the environment variable "LOCE_RELOCATE_FROM_FLASH" to '1' during the "makeimg" build phase. This can be done by setting the environment variable in the Platform Builder "Platform -> Settings…" menu option under the "Environment" tab.

4.2 Configuring OS Memory Map

(The default values in these files are known good values and in most cases do not need to be changed unless the image size becomes very large.)

The config.bib file is used to specify memory locations and other characteristics of the NK.bin file you are generating using Platform Builder. The config.bib file can be found in the 'LoCE/files/' folder. In this file, the size and location of the image and the size and location of the available RAM are set. Instead of hard-coding values into this file, the LoCE BSP relies on a specific header file for each hardware platform to provide the values. These header files can be found in the 'LoCE/src/inc/card_engine/' folder and will have the following naming syntax: 'processor'_xx_config_bib.h.

Default RAM, flash, and relocation image values are already set in these files to correctly build each type of image. The table below shows a graphical relation between these values and the memory of the system.

These header files can be customized, although it can be dangerous to change any values without proper system memory knowledge. For example, if a flash image was created that overwrote the bootloader of a device, that device may no longer boot.

Device_config_bib.h Customizable Values:

RAM Images:

RAM_BUILD_RAM_START:	Address of open RAM for CE to use.
RAM_BUILD_RAM_SIZE:	Size of open RAM for CE to use.
RAM_BUILD_IMAGE_START:	Address for CE Image to reside and run in RAM.
RAM_BUILD_IMAGE_SIZE:	Size of RAM area reserved for CE Image to occupy.
RAM_BUILD_ROMOFFSET:	Should always be 0x00000000.

Flash Images:

FLA_BUILD_RAM_START:	Address of open RAM for CE to use.
FLA _BUILD_RAM_SIZE:	Size of open RAM for CE to use.
FLA _BUILD_IMAGE_START:	Address for XIP CE Image to reside and run in flash.
FLA _BUILD_IMAGE_SIZE:	Size of flash area reserved for CE Image to occupy.
FLA_BUILD_ROMOFFSET:	Should always be 0x00000000.
FLA_BUILD_BOOTJUMP:	Should always be same as FLA_BUILD_IMAGE_START.

Relocation Images:

RAM_BUILD_RAM_START:	Address of open RAM for CE to use.
RAM_BUILD_RAM_SIZE:	Size of open RAM for CE to use.
RAM_BUILD_IMAGE_START:	Address for CE Image to reside and run in RAM.
RAM_BUILD_IMAGE_SIZE:	Size of RAM area reserved for CE Image to occupy.
RELOCATE_BUILD_BOOTJUMP:	Address for CE Image to reside in and relocate to RAM from.
RELOCATE_BUILD_ROMOFFSET:	Must equal RELOCATE_BUILD_BOOTJUMP - RAM_BUILD_IMAGE_START.

4.2.1 Memory Structure

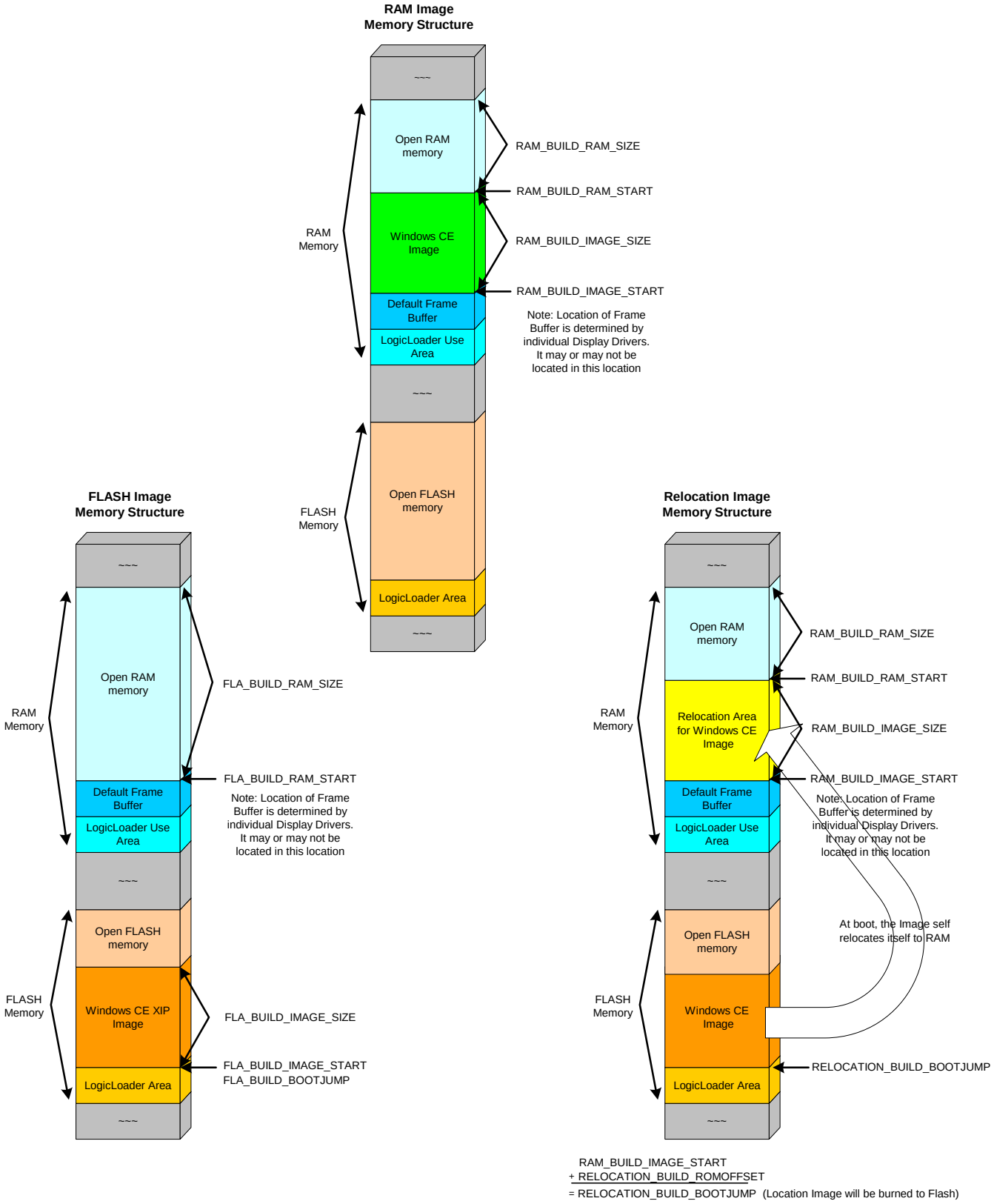


Figure 4.1: LoCE Memory Structure

4.3 Drivers and Components

Device drivers have been written to be as device dependent and platform independent as possible. Doing so allows Logic Product Development to leverage device drivers across many hardware platforms. All Logic Product Development device drivers use installable interrupt service routines and leverage the Windows Embedded CE registry to make them as configurable as possible. Each device driver includes a *.bib and *.reg file in its top-level directory. Please see these files for important information about using and configuring each device driver.

When a LoCE binary driver is installed, the component is automatically added to Platform Builder's catalog under "Third Party/Device Drives/" and the driver folder is placed in '\LoCE\PLATFORM\bin\lpd_drivers\'.

The same can be said for a binary kernel component, which gets put into "Third Party/LoCE Platform Support/kernels" and '\LoCE\PLATFORM\bin\lpd_kernels\' respectively.

4.3.1 LoCE Binary Driver Files

The picture below shows a basic structure of a LoCE binary driver folder. The 'bin' folder contains the actual .dll files that are added to an image at build time. (**Note:** This is all done automatically.)

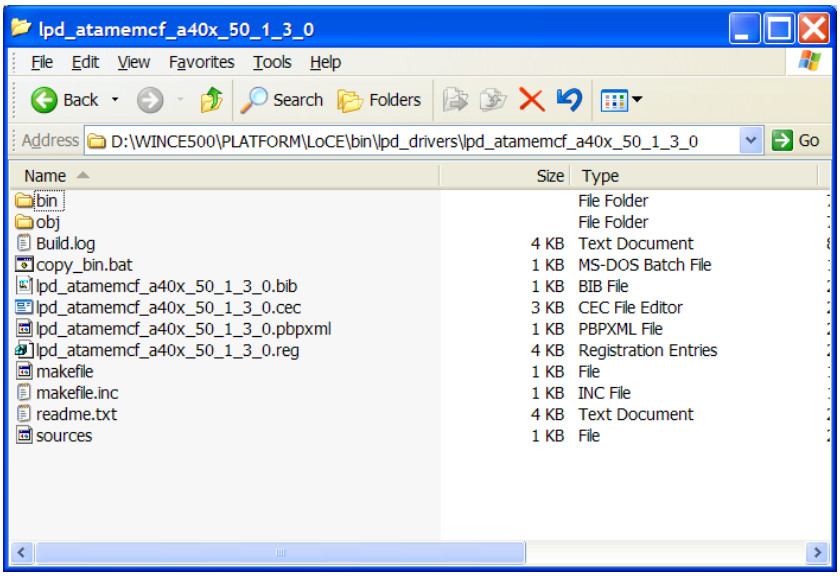


Figure 4.2: Example Binary Driver Folder Contents

There are two important files that users should look at when using the driver. The first is the "readme.txt" file which contains release notes, a description of what's new for the specific version, known issues, history of the driver, dependencies, and notes about use.

The second file that is important for users to understand is the .reg file. This contains all of the configurable values for a driver that are used to load that driver (e.g., memory locations, COM port numbers, memory size, frame buffer addresses). Each registry entry has a description in the comments section of the file.

4.3.2 Add a Component to a Platform Builder Workspace

To use a driver or kernel component in a Platform Builder project, right-click on the component and click "Add to OS Design". If a component was not compiled for the project's processor type, it will be grayed out.

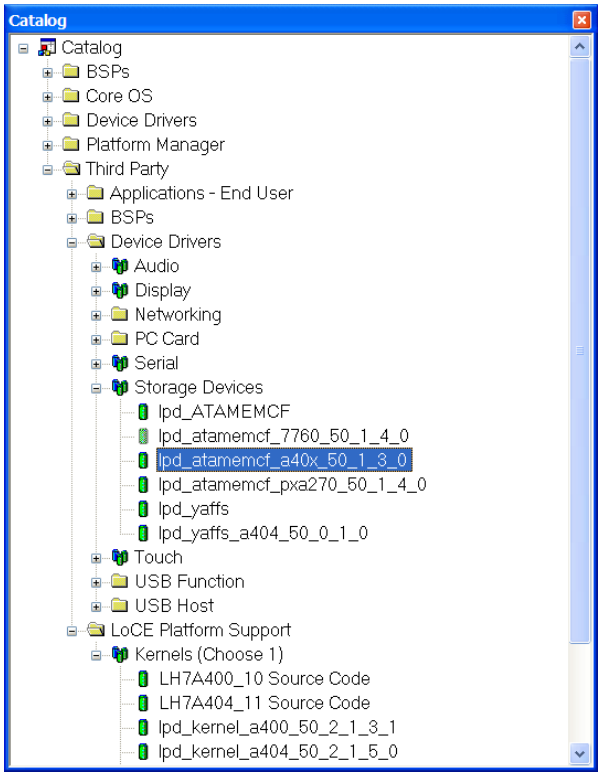


Figure 4.3: Add a Component to Platform Builder Workspace

4.4 Creating an OS Image

The default behavior of the LoCE BSP is to build Windows Embedded CE images that are linked to be downloaded and run directly from the SOM's RAM. Users may also choose to build images that are linked to be downloaded, stored, and run from the SOM's flash memory. Another type of image that can be built is a "relocation" image. These images are packaged in the .bin file to be stored in flash, but the images will self-relocate themselves into RAM during the initialization sequence of the WinCE image.

4.4.1 Building an Image

Once the LoCE Base and a kernel, and possibly drivers, have been installed, a new platform can be created in Platform Builder.

The following steps will build a very simple RAM image that will include several basic drivers and support for those drivers. Note that the BSP, kernel, and drivers are chosen as examples and may be older versions of those components or for a different device altogether. Some drivers may not be available for specific hardware devices. For example, a device may not have a USB Host port, in which case adding USB Host components and driver is not possible.

Steps:

1. Use the New Platform Wizard in Platform Builder. Start by going to "File->New Platform". The new platform will need to be given a name.

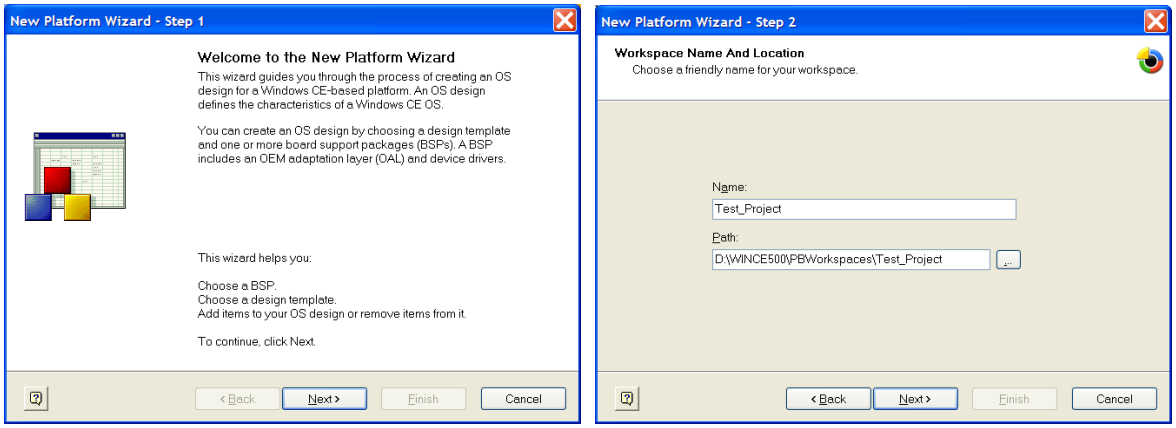


Figure 4.4: Platform Builder 'Platform Wizard'

2. Choose a BSP to use. LoCE BSPs begin with 'LOCE' as a prefix followed by the processor type (e.g., LOCE ARMV4I BSP or LOCE SH4 BSP).

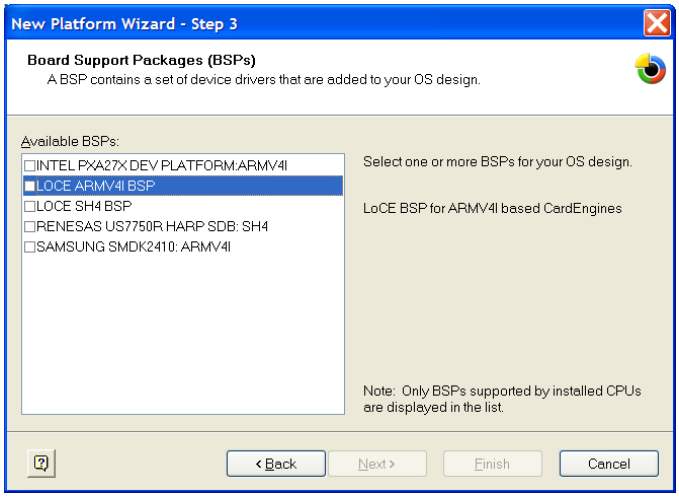
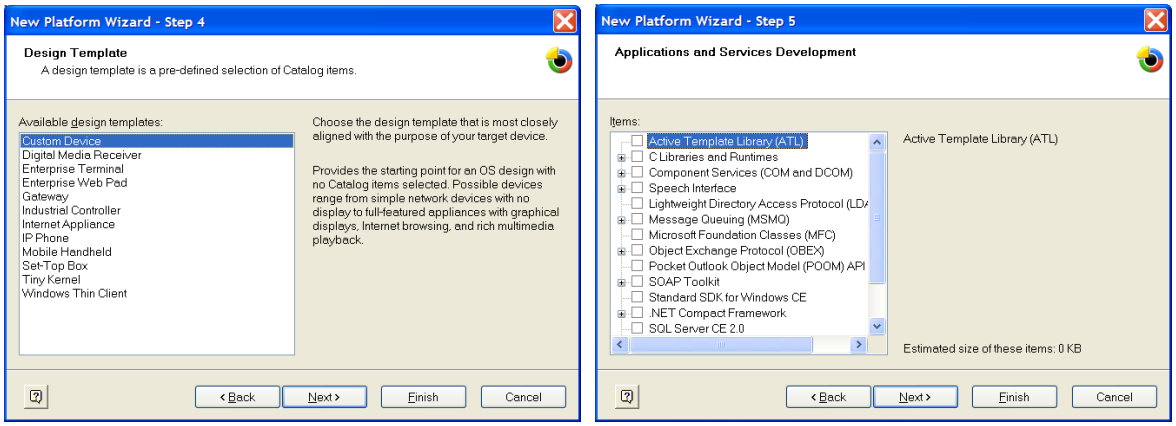
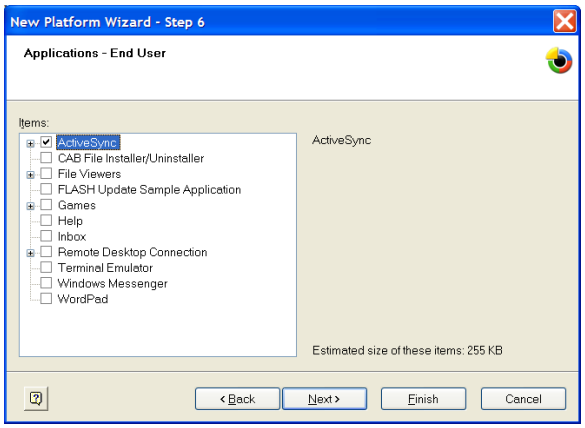
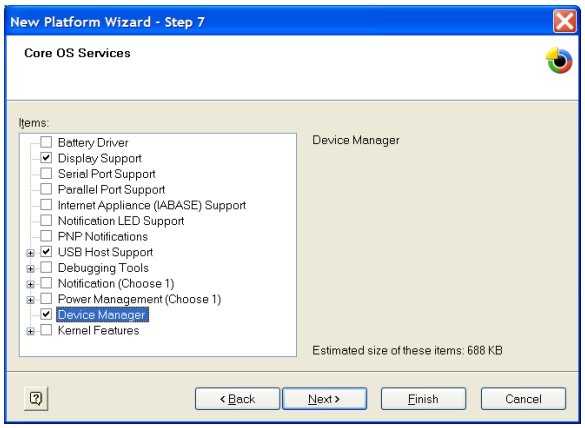


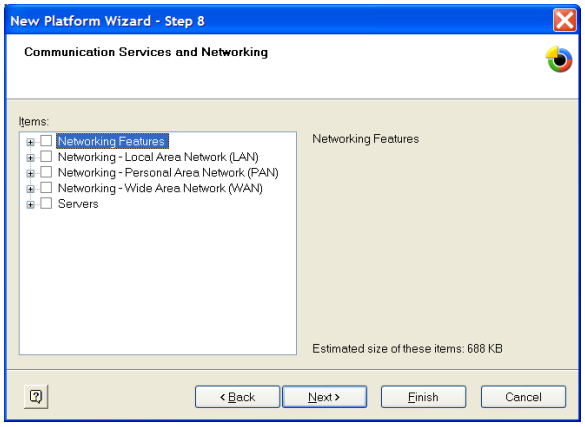
Figure 4.5: Platform Wizard Select BSP Option

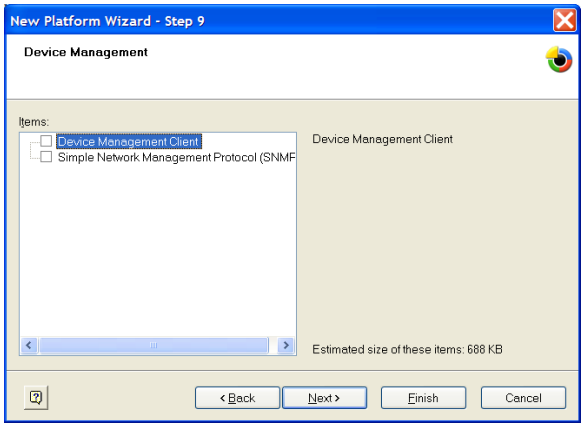
3. Platform Builder gives several 'design templates' that help to choose default catalog items for the image. 'Custom Device' does not choose any components by default. The steps that follow allow the user to add specific components to the default platform. Note that components can also be added after the wizard has completed.

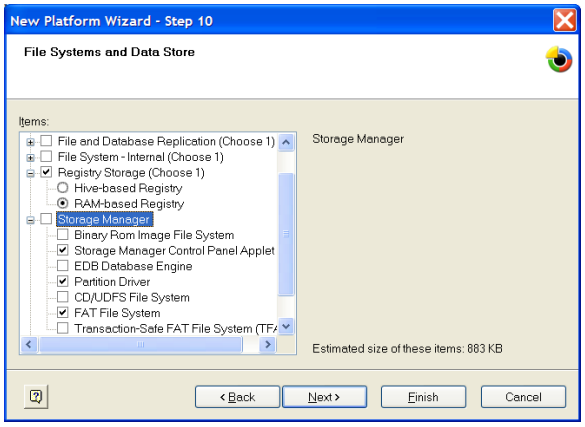


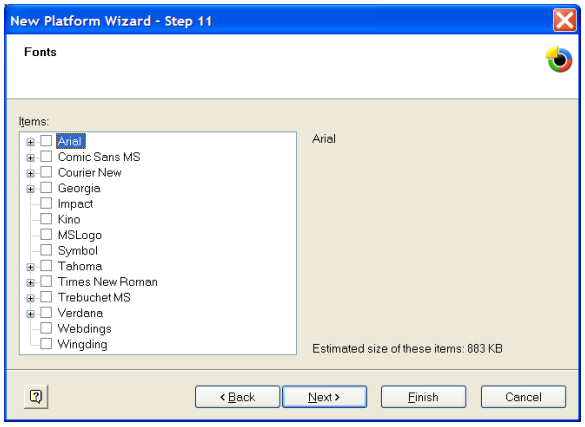

New Platform Wizard - Step 6
Applications - End User
Items:
ActiveSync
CAB File Installer/Uninstaller
File Viewers
FLASH Update Sample Application
Games
Help
Inbox
Remote Desktop Connection
Terminal Emulator
Windows Messenger
WordPad
ActiveSync
Estimated size of these items: 255 KB
< Back
Next >
Finish
Cancel

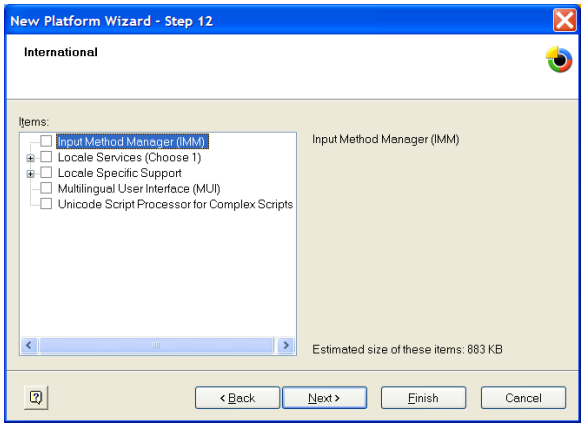

New Platform Wizard - Step 7
Core OS Services
Items:
Battery Driver
Display Support
Serial Port Support
Parallel Port Support
Internet Appliance (IABASE) Support
Notification LED Support
PNP Notifications
USB Host Support
Debugging Tools
Notification (Choose 1)
Power Management (Choose 1)
Device Manager
Kernel Features
Device Manager
Estimated size of these items: 688 KB
< Back
Next >
Finish
Cancel

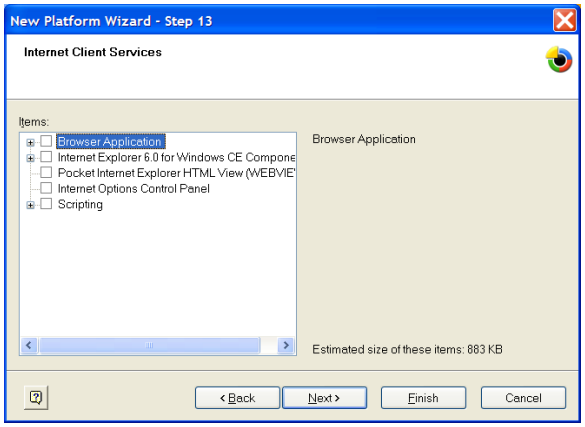

New Platform Wizard - Step 8
Communication Services and Networking
Items:
Networking Features
Networking - Local Area Network (LAN)
Networking - Personal Area Network (PAN)
Networking - Wide Area Network (WAN)
Servers
Networking Features
Estimated size of these items: 688 KB
< Back
Next >
Finish
Cancel

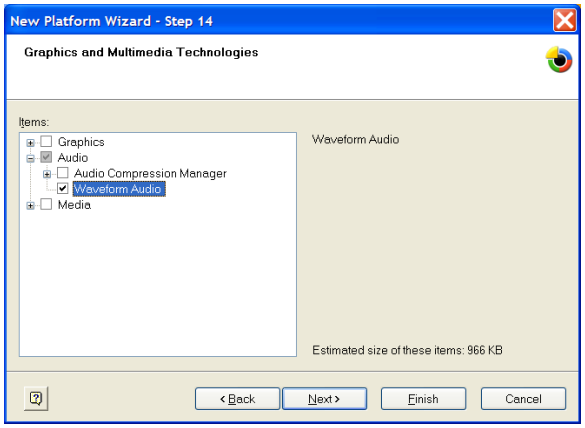

New Platform Wizard - Step 9
Device Management
Items:
Device Management Client
Simple Network Management Protocol (SNMP
Device Management Client
Estimated size of these items: 688 KB
< Back
Next >
Finish
Cancel

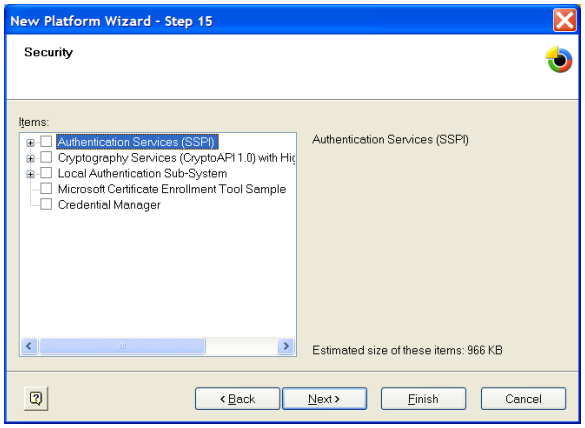

New Platform Wizard - Step 10
File Systems and Data Store
Items:
File and Database Replication (Choose 1)
File System - Internal (Choose 1)
Registry Storage (Choose 1)
Hive-based Registry
RAM-based Registry
Storage Manager
Binary Rom Image File System
Storage Manager Control Panel Applet
EDB Database Engine
Partition Driver
CD/UDFS File System
FAT File System
Transaction-Safe FAT File System
Storage Manager
Estimated size of these items: 883 KB
< Back
Next >
Finish
Cancel

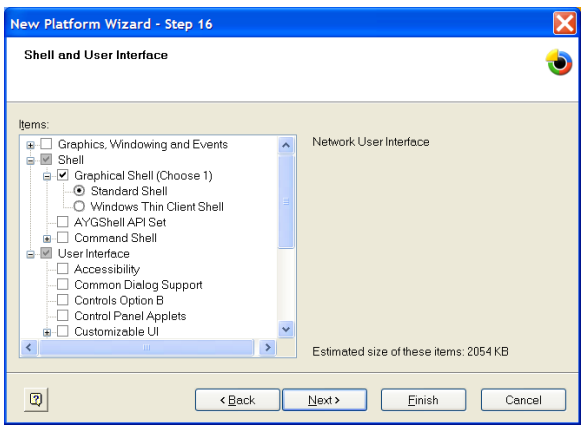

New Platform Wizard - Step 11
Fonts
Items:
Arial
Comic Sans MS
Courier New
Georgia
Impact
Kino
MSLogo
Symbol
Tahoma
Times New Roman
Trebuchet MS
Verdana
Webdings
Wingding
Arial
Estimated size of these items: 883 KB
< Back
Next >
Finish
Cancel

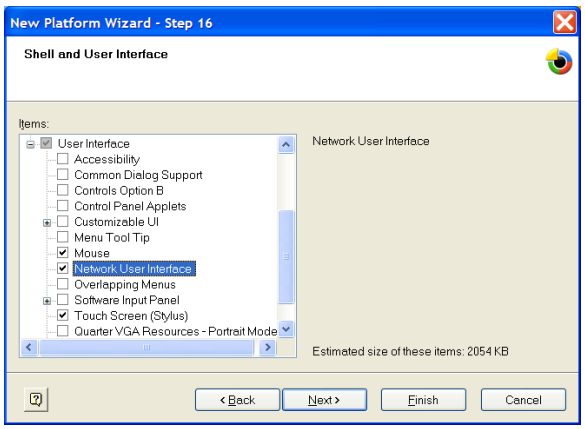

New Platform Wizard - Step 12
International
Items:
Input Method Manager (IMM)
Locale Services (Choose 1)
Locale Specific Support
Multilingual User Interface (MUI)
Unicode Script Processor for Complex Scripts
Input Method Manager (IMM)
Estimated size of these items: 883 KB
< Back
Next >
Finish
Cancel


New Platform Wizard - Step 13
Internet Client Services
Items:
Browser Application
Internet Explorer 6.0 for Windows CE Compone
Pocket Internet Explorer HTML View (WEBVIE
Internet Options Control Panel
Scripting
Browser Application
Estimated size of these items: 883 KB
< Back
Next >
Finish
Cancel


New Platform Wizard - Step 14
Graphics and Multimedia Technologies
Items:
Graphics
Audio
Audio Compression Manager
Waveform Audio
Media
Waveform Audio
Estimated size of these items: 966 KB
< Back
Next >
Finish
Cancel


New Platform Wizard - Step 15
Security
Items:
Authentication Services (SSPI)
Cryptography Services (CryptoAPI 1.0) with Hig
Local Authentication Sub-System
Microsoft Certificate Enrollment Tool Sample
Credential Manager
Authentication Services (SSPI)
Estimated size of these items: 966 KB
< Back
Next >
Finish
Cancel


New Platform Wizard - Step 16
Shell and User Interface
Items:
Graphics, Windowing and Events
Shell
Graphical Shell (Choose 1)
Standard Shell
Windows Thin Client Shell
AYGShell API Set
Command Shell
User Interface
Accessibility
Common Dialog Support
Controls Option B
Control Panel Applets
Customizable UI
Network User Interface
Estimated size of these items: 2054 KB
< Back
Next >
Finish
Cancel


New Platform Wizard - Step 16
Shell and User Interface
Items:
User Interface
Accessibility
Common Dialog Support
Controls Option B
Control Panel Applets
Customizable UI
Menu Tool Tip
Mouse
Network User Interface
Overlapping Menus
Software Input Panel
Touch Screen (Stylus)
Quarter VGA Resources - Portrait Mode
Network User Interface
Estimated size of these items: 2054 KB
< Back
Next >
Finish
Cancel

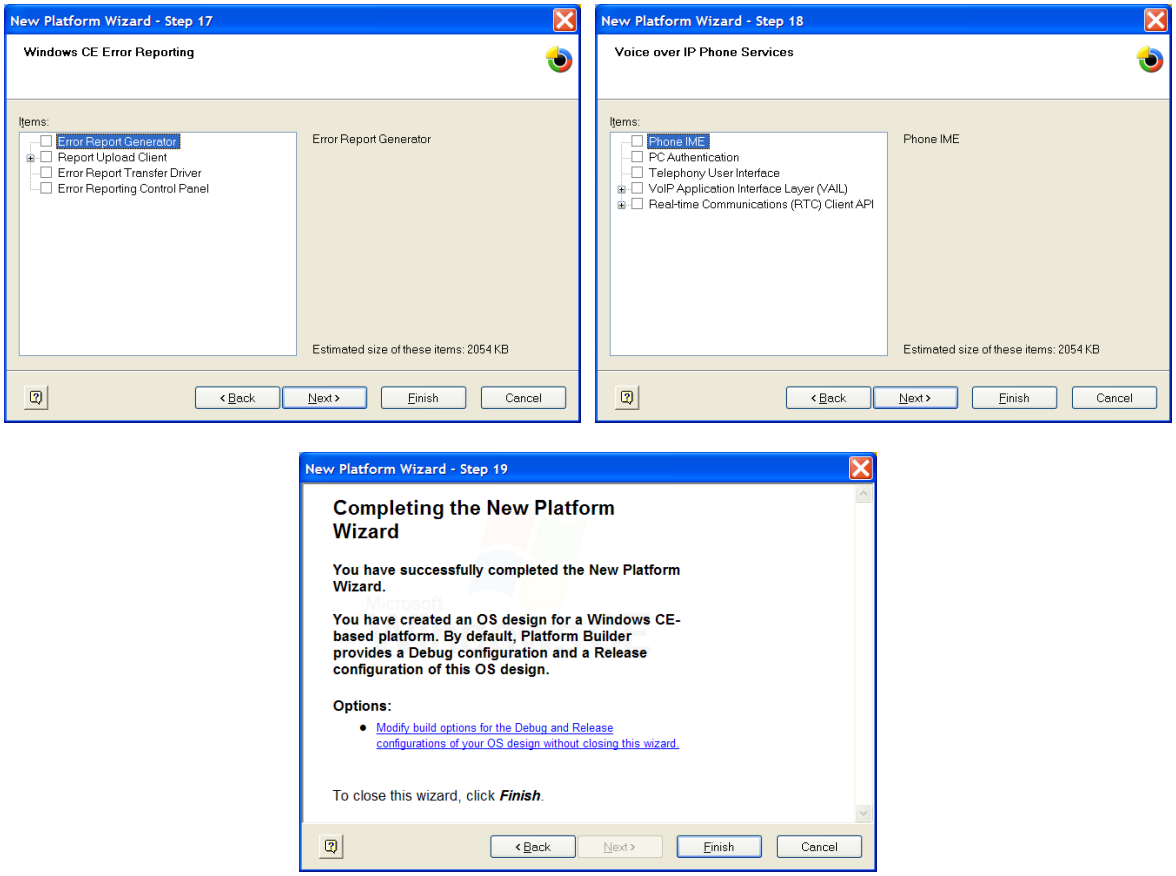


Figure 4.6: Platform Wizard Configuration Steps

4. At this point, the catalog should look something like the following diagram. No drivers or kernel have been added yet.

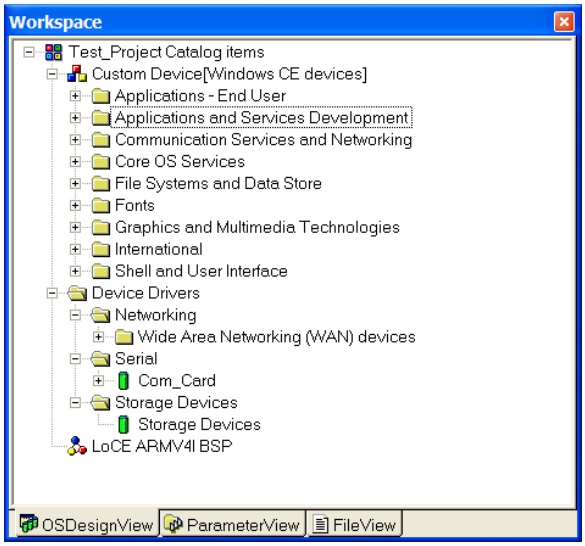


Figure 4.7: Example Catalog

5. For this example, the display, touch, audio, USB Host, USB Function, and Memory Mapped CompactFlash drivers will be added in addition to the kernel.

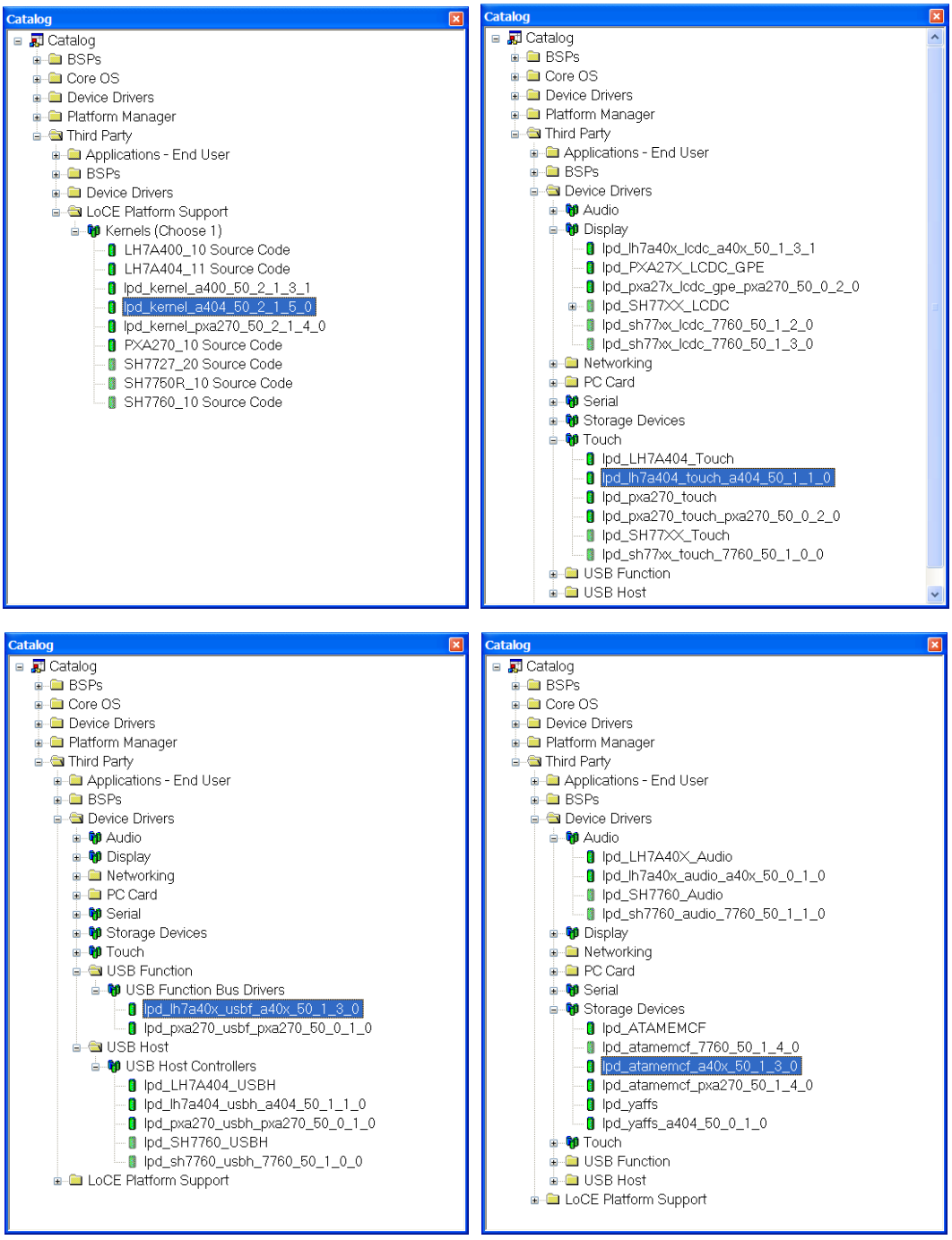


Figure 4.8: Adding Drivers

6. The final workspace should look like the following diagram.

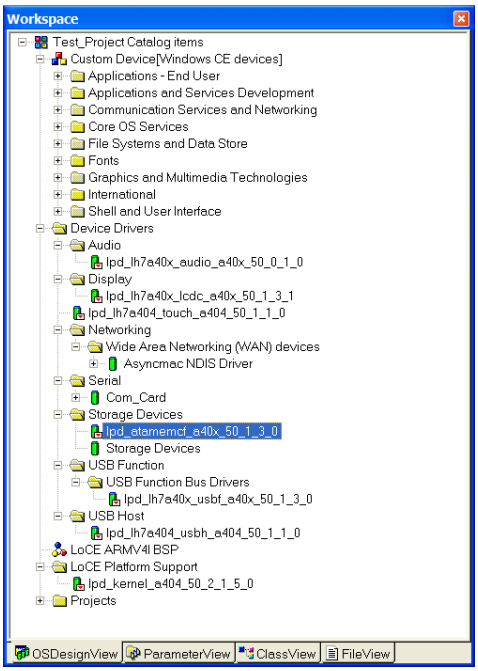


Figure 4.9: Platform Builder Workspace View

7. At this point, the project can be built by going to “Build OS->Sysgen”. A new project build can take several minutes or longer.

   For an in depth description of the “Build OS Menu Options”, please refer to the following link on MSDN: Build OS Menu Options.

8. When Platform Builder has completed building, the output windows should show that there were “0 errors”. The NK.bin is now ready to load onto a device.

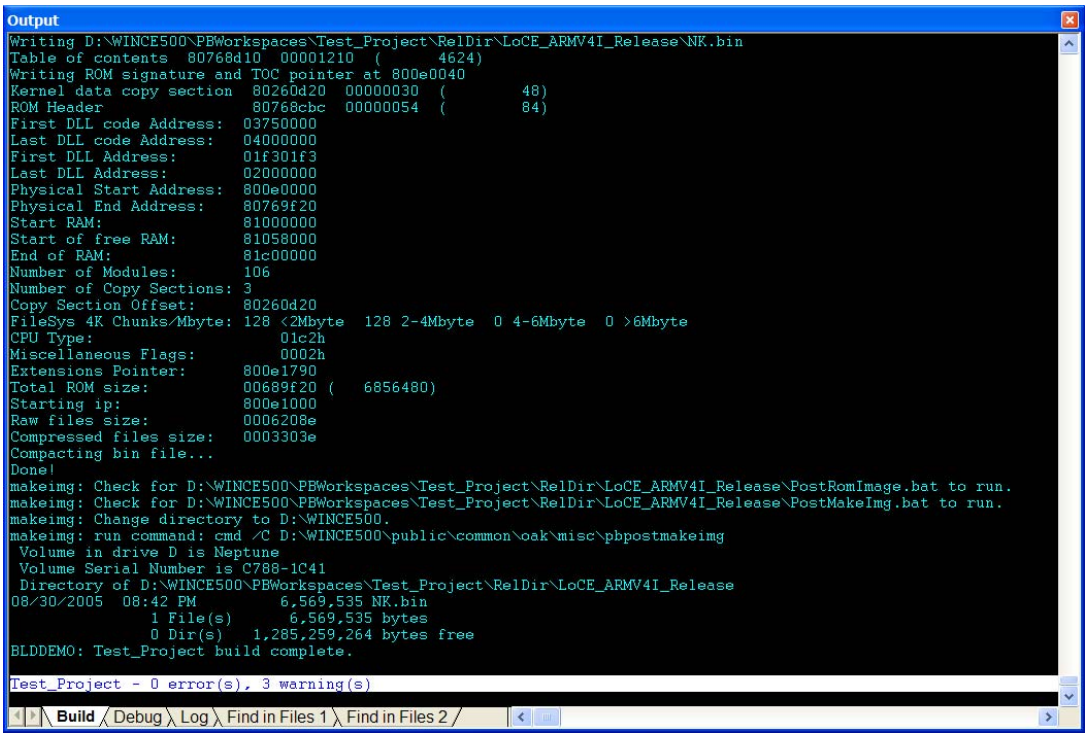


Figure 4.10: Platform Builder Output Windows (Build Complete)

4.4.2 Differences between the LoCE BSP and other BSPs

Certain aspects of Logic's LoCE Binary BSP differ from other BSPs, and these differences must be considered before building the Windows Embedded CE image.

1. An OAL (Kernel) component must be added to the image for the specific SOM targeted for the image. The kernel that has been installed is found under "Catalog -> Third Party -> LoCE Platform Support -> Kernels". Right-click on the desired kernel support and click "Add to OS Design".

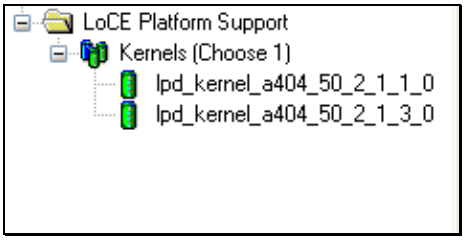


Figure 4.11: Right-Click on Desired Kernel to Add

2. Driver components also have to be added manually to the Windows Embedded CE image. The driver components that have been installed are found under "Catalog -> Third Party -> Device Drivers". Right-click on the specific driver components that are required for the image and then click "Add to OS Design".

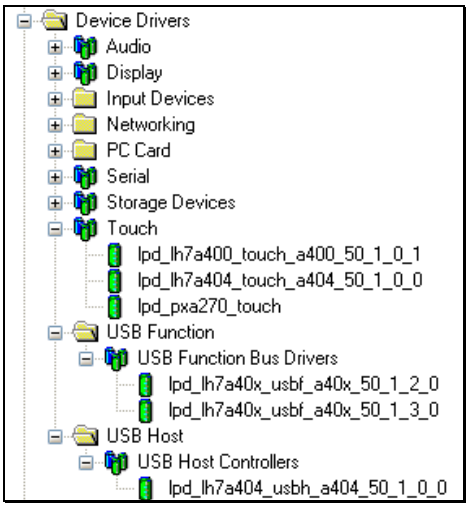


Figure 4.12: Right-Click on Desired Drivers to Add

3. Before building the image please verify registry file settings for each of the driver components in the image to make sure that they are correctly built for the specific SOM.

4.4.3 Building a Flash or Relocation OS Image

The default behavior of the LoCE BSP is to build Windows Embedded CE images which are linked to be downloaded and run directly from the SOM's RAM. Users may also choose to build images, which are linked to be downloaded, stored, and run from the SOM's flash memory. Another type of image that can be built is a "relocation" image. These images are packaged in the .bin file to be stored in flash, but the images will self-relocate themselves into RAM in the initialization sequence of the WinCE image.

To build an image for flash, perform the following steps.

1. Select "Platform -> Settings".
2. Select the "Build Options" tab.
3. Check the "Write Run-time Image to Flash Memory" box, as shown below.

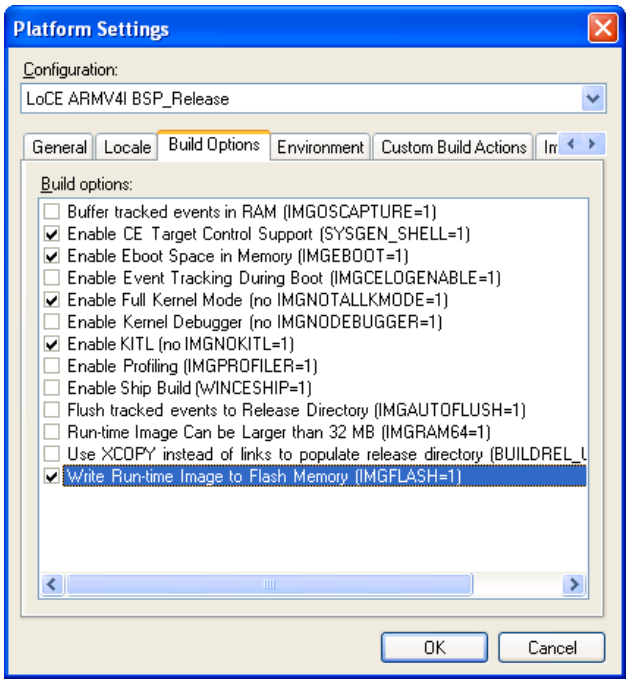


Figure 4.13: Building an Image for Flash

You will need to rebuild the Windows Embedded CE image after you make this change.

4.4.4 Building a relocation image

To build a relocation image, perform the following steps.

1. Select “Platform -> Settings”.
2. Select the “Environment” tab.
3. Click the “New…” button and enter the new environment variable “LOCE_RELOCATE_FROM_FLASH”, as shown in Figure 4.14.

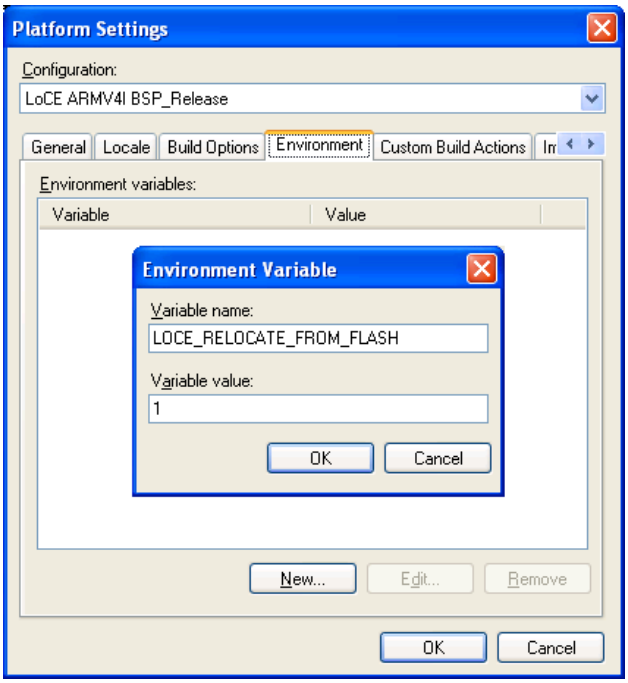


Figure 4.14: Building a Relocation Image

You will need to rebuild the Windows Embedded CE image after you make this change. For an in depth description of the “Build OS Menu Options”, please refer to the following link on MSDN: Build OS Menu Options.

5 Launching Windows Embedded CE

When a LoCE Windows Embedded CE image is built, it must be placed in a location accessible by the bootloader. For instance, the image can be placed in RAM (volatile memory) or in flash (non-volatile memory) on the platform used. Once the image is downloaded and placed in the memory location, it must be jumped-to from a bootloader. Logic has not only designed LogicLoader™ to boot Windows Embedded CE images, but also to easily download from several mediums, allow for Ethernet debugging, automatic booting, scripting, and to pass boot time parameters to Windows Embedded CE.

This section only covers LogicLoader as it pertains to booting the OS image created by the LoCE Windows Embedded CE BSP. For a complete reference on LogicLoader, please see the LogicLoader User's Manual, which provides information on loading binary files from different mediums, storing scripts, and auto-booting systems.

5.1 Using LogicLoader to Download the OS Image (NK.bin)

LogicLoader Download Mediums:

- Serial
- Ethernet
- YAFFS (Flash File System for onboard flash)
- CompactFlash

The LogicLoader Manual and Command Description can be found on Logic's website:

- *LogicLoader User's Manual*
- *LogicLoader Command Description Manual*

Note: It is assumed that LogicLoader 2.0.1 or newer is used for the following steps.

5.1.1 Using LogicLoader to Download the OS Image to RAM over Ethernet

1. From the LogicLoader prompt, initialize the SOM's network interface with the "ifconfig" command. See the *LogicLoader User's Manual* or type 'help ifconfig' at the command prompt for more information.

   losh> ifconfig sm0 dhcp

   Use LogicLoader's "bootme" command to initiate a connection to the Platform Builder IDE on the workstation.

   losh> bootme

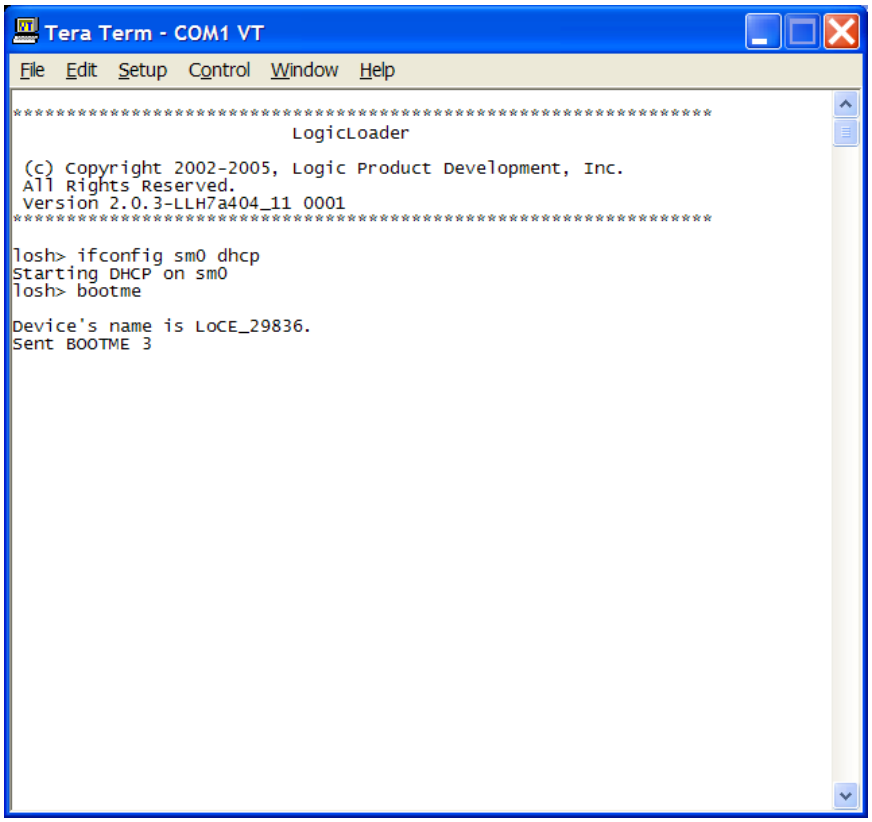


Figure 5.1: Set-up LogicLoader Network Connection

2. Set-up the Platform Builder “Connectivity Options”, which can be found under the “Target” menu, to use Ethernet and find the device on the network. Once the “Download” and “Transport” have been set to “Ethernet”, click the “settings” button to bring up the Active Device. The device should show up if LogicLoader continues to say “sent BOOTME #”. The device will show up as “LOCE_#####”, with the number corresponding to the MAC Address of the device, in order to distinguish between multiple devices. Click ‘Apply’.

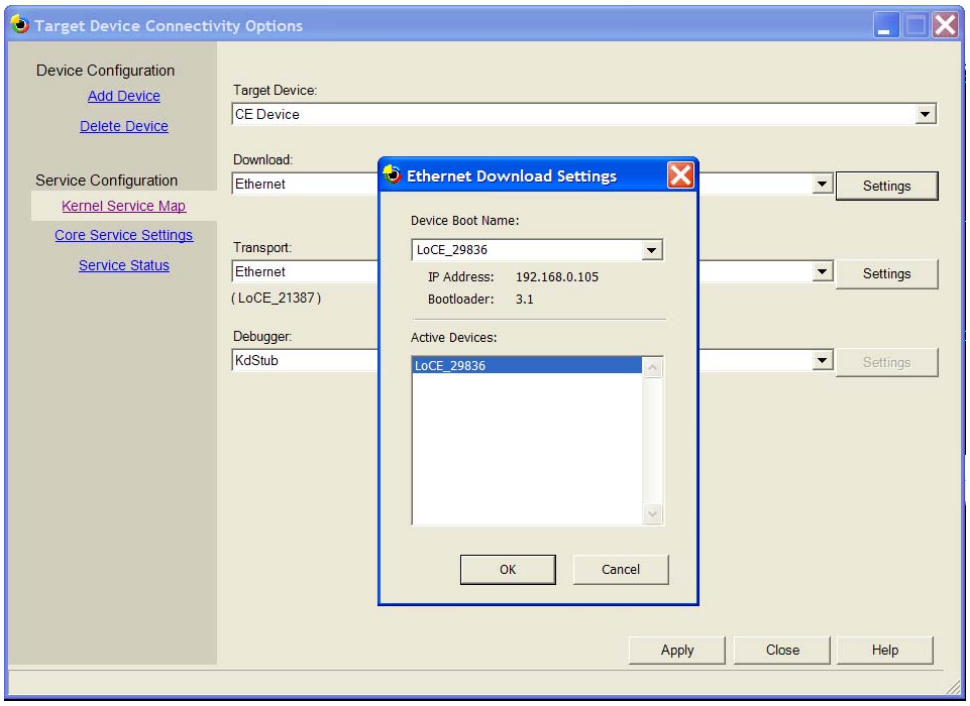


Figure 5.2: Platform Builder Connectivity Options

3. To start the download, go to the “Target” menu in Platform Builder and select “Attach Device”. This should begin downloading the image. A status window will pop up and you will also see the status progression in LogicLoader.

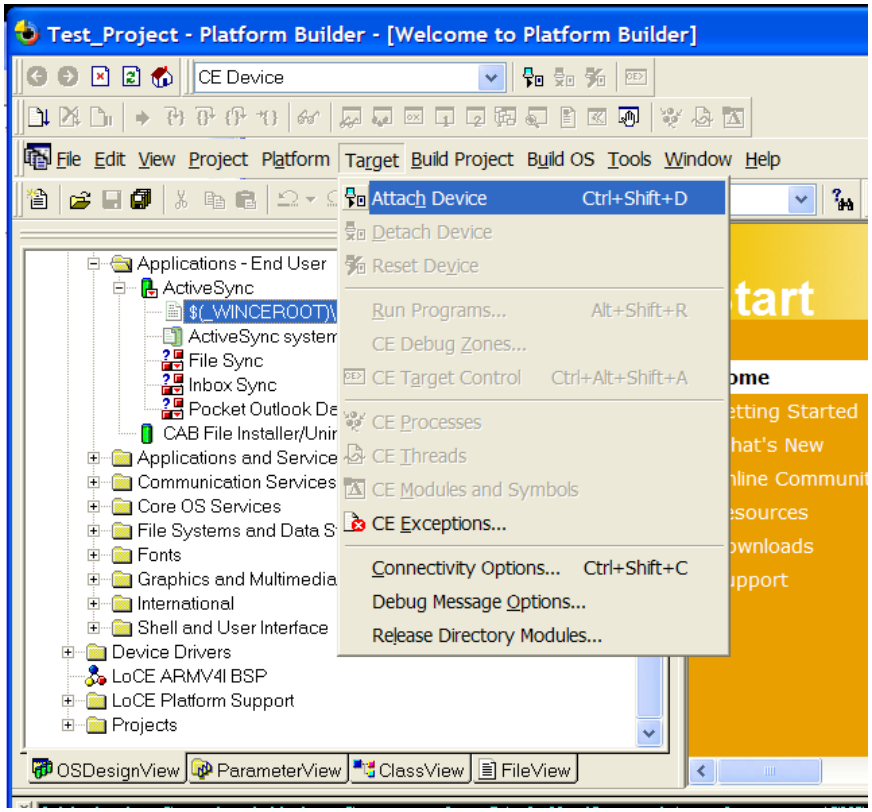


Figure 5.3: Platform Builder “Target -> Attach Device”

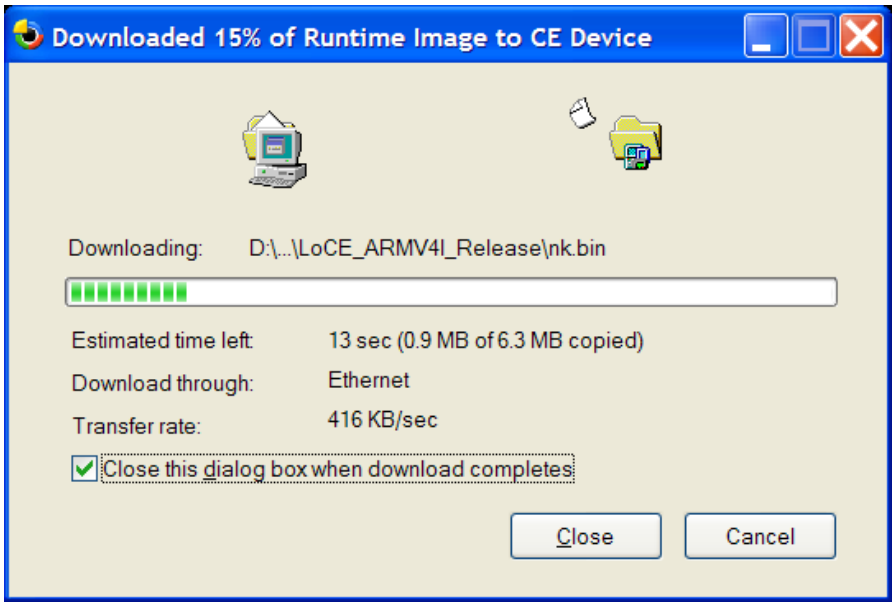


Figure 5.4: Platform Builder Download Status

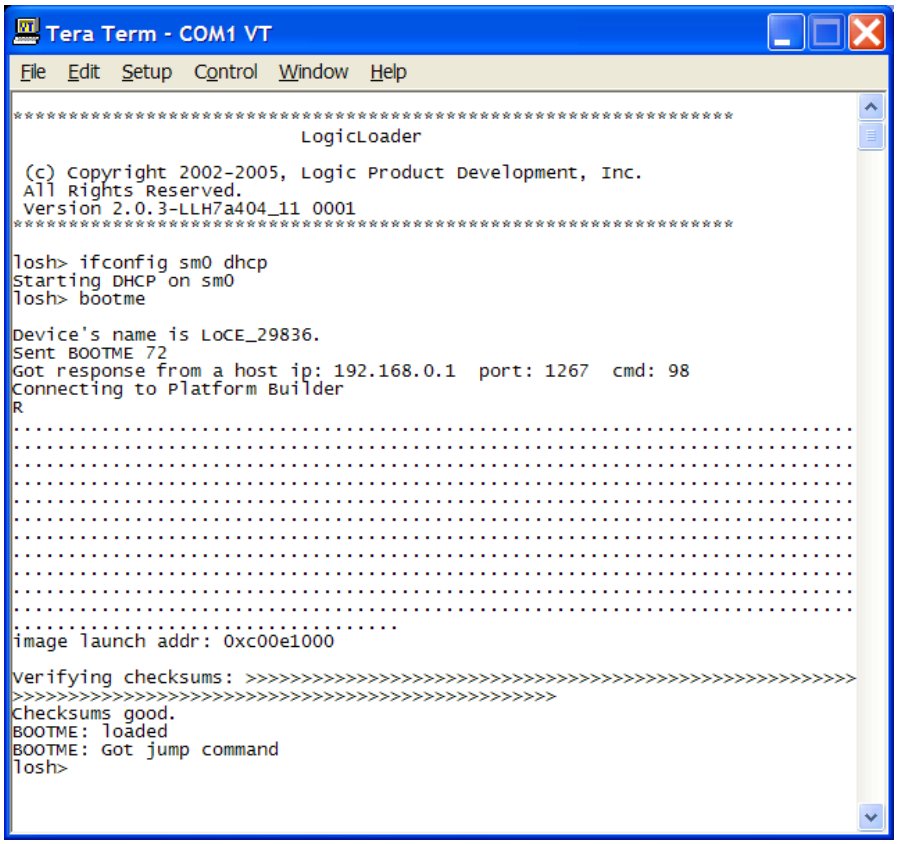


Figure 5.5: LogicLoader Ethernet Download Status

4. If the image downloads correctly, LogicLoader will output the following:

   Checksums good.
   BOOTME: Loaded

BOOTME: Got jump command
losh>

The image is now ready to be run out of RAM.

5.1.2 Using LogicLoader to Download the OS Image to RAM over Serial

1. Use the LogicLoader “load bin” command to begin the serial download.

   losh> load bin

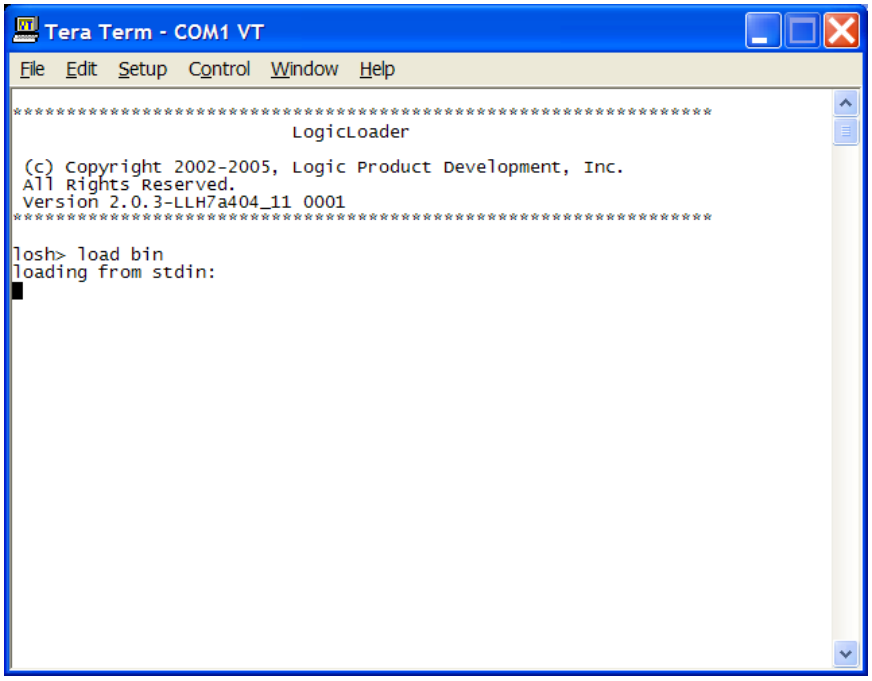


Figure 5.6: LogicLoader Serial Download

2. Use the TeraTerm “File -> Send File…” command. This will open a browsing window. Find the NK.bin in the workspace release folder. Make sure the “Binary” box is checked. Click “Open”.

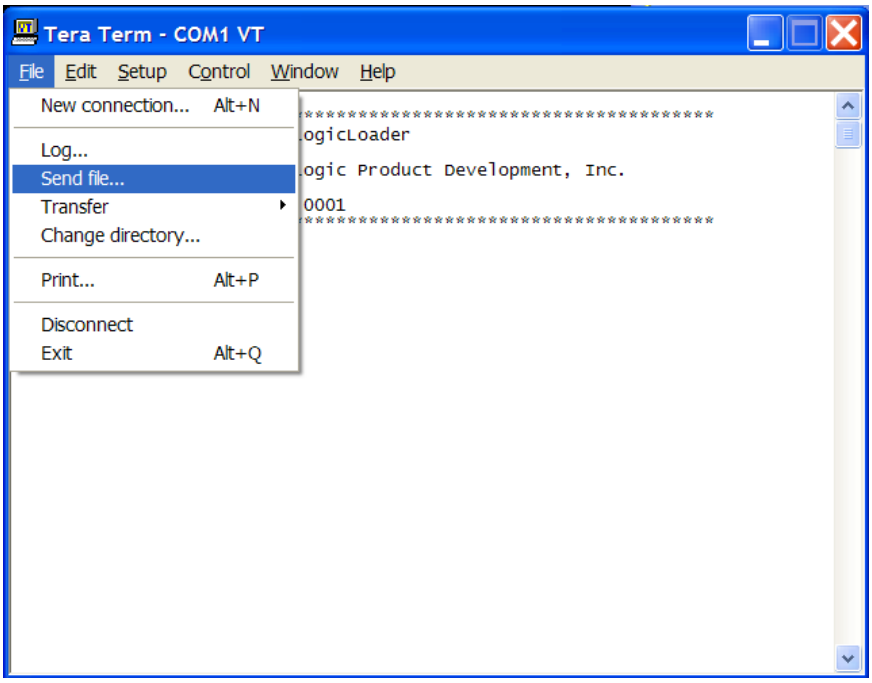


Figure 5.7: LogicLoader "Send File"

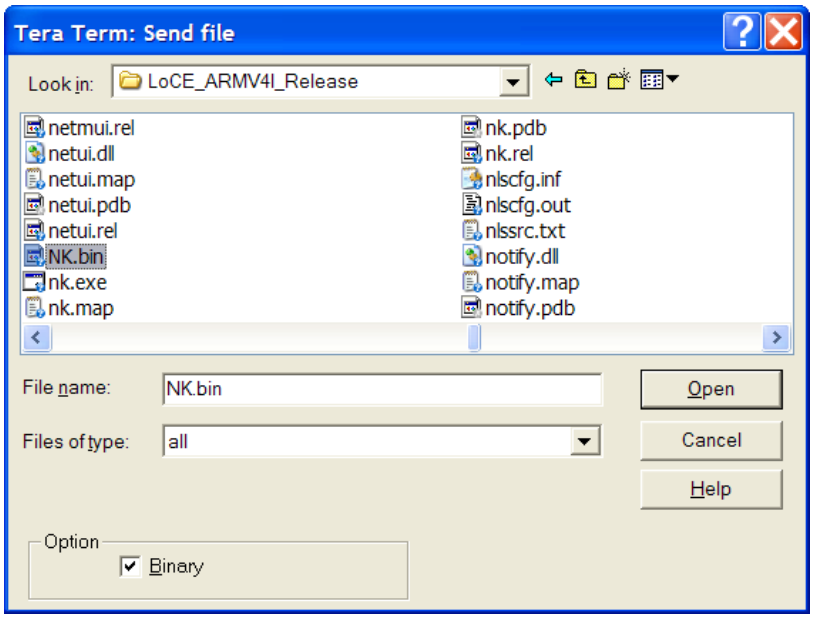


Figure 5.8: LogicLoader "Send File" Browser

3. This will begin the transfer, which may take several minutes. (Serial transfer is the slowest download medium in LogicLoader.)

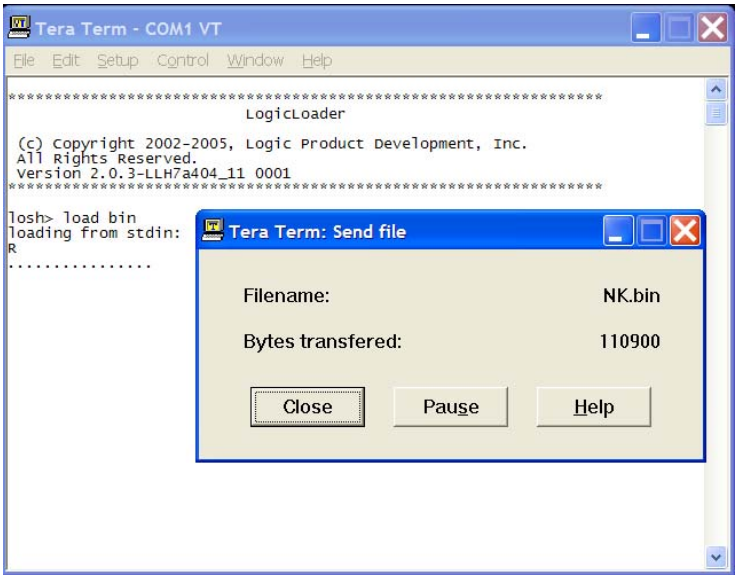


Figure 5.9: LogicLoader Serial Download Status

4. If the image downloads correctly, LogicLoader will output the following:

Checksums good.
File loaded
losh>

The image is now ready to be run out of RAM.

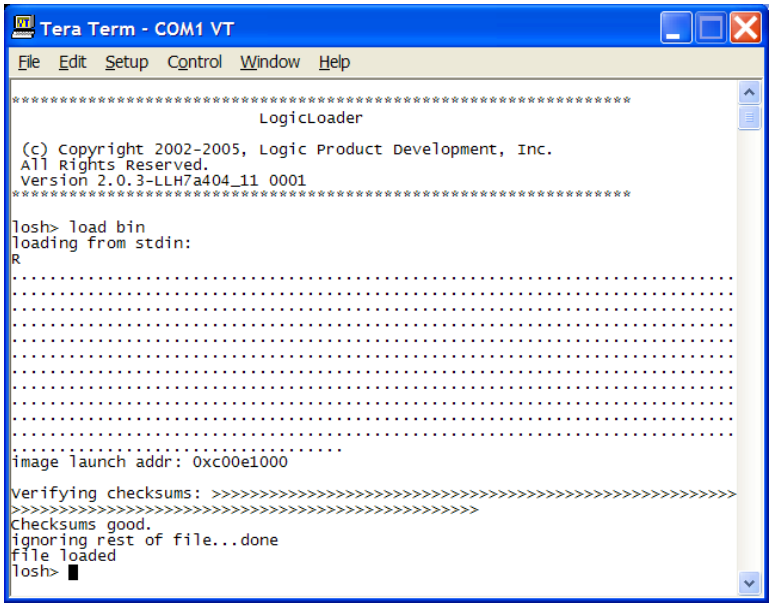


Figure 5.10: LogicLoader Serial Download Success

5.1.3 Using LogicLoader to Download the OS Image to RAM from a CompactFlash Card

1. Copy the NK.bin, which is found in '\WINCE500\PBWorkspaces\Test_Project\RelDir\LOCE_ARMV4I_Release', to a CompactFlash card.

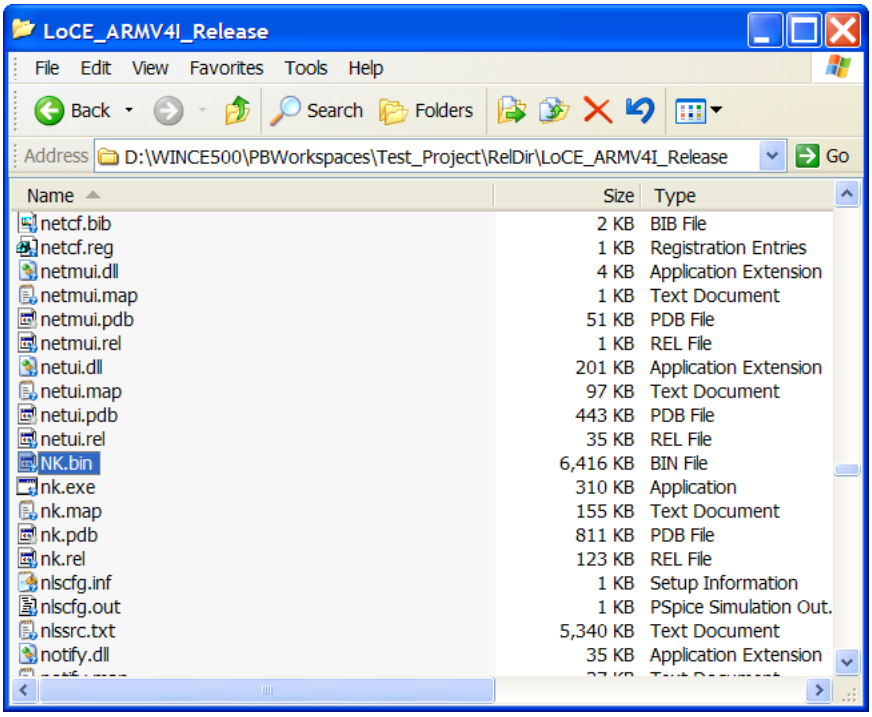


Figure 5.11: Built NK.BIN Location

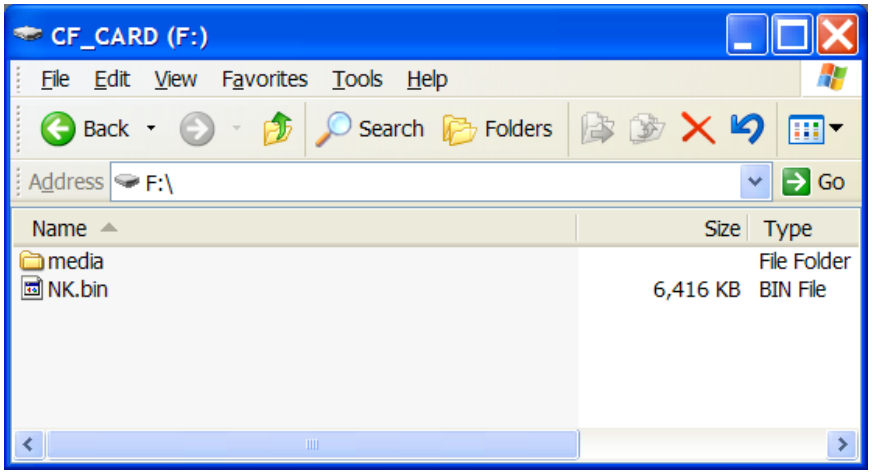


Figure 5.12: CompactFlash Card NK.bin

2. Insert the CompactFlash card into the memory-mapped CompactFlash slot on the device. Use the LogicLoader “mount fatfs” command to mount the CompactFlash card and then use the “load bin” command to copy the image into RAM.

```
losh> mount fatfs /cf
losh> load bin /cf/NK.BIN
```

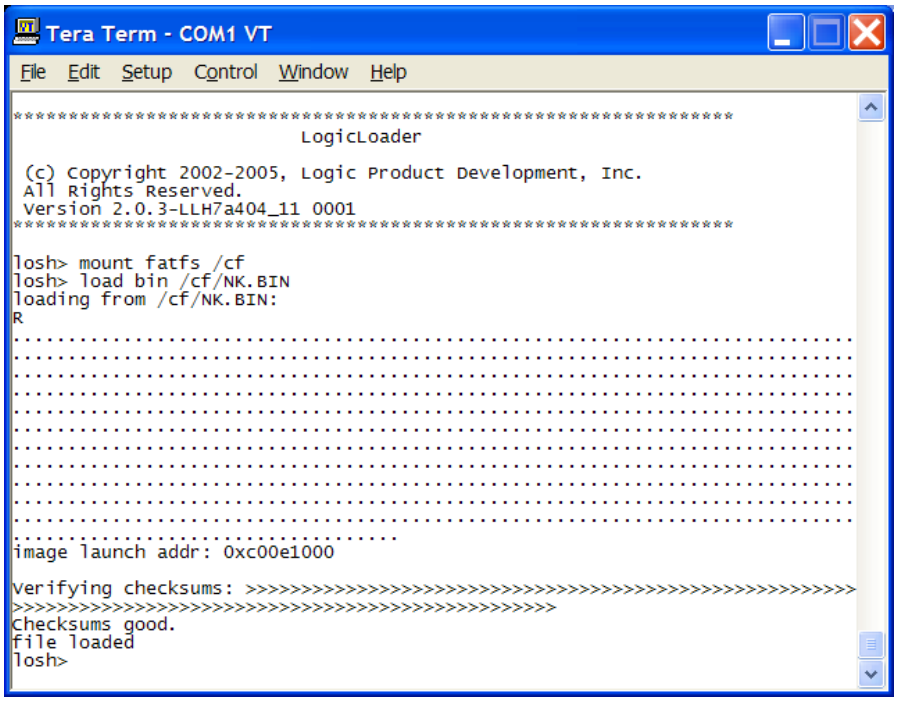


Figure 5.13: LogicLoader CompactFlash Download

3. If the image downloads correctly, LogicLoader will output the following:

Checksums good.
File loaded
losh>

The image is now ready to be run out of RAM.

5.2 Using LogicLoader to Boot the LoCE BSP Created Image

In order to boot the LoCE Windows Embedded CE image, LogicLoader's "exec" command is used followed by an appropriate boot parameter string. For instance, the LogicLoader command shown in the figure below can be used to boot a LoCE image built for the LH7A404-11 SOM. The example below assumes that the Windows Embedded CE image (NK.bin) was downloaded into RAM, in this case over the serial port.

exec [boot parameter string]

Example: **exec dbg_uart:A404_UART:rtc:rtc_a404_int:**

For information regarding Boot Parameters, please see the Boot Parameter section.

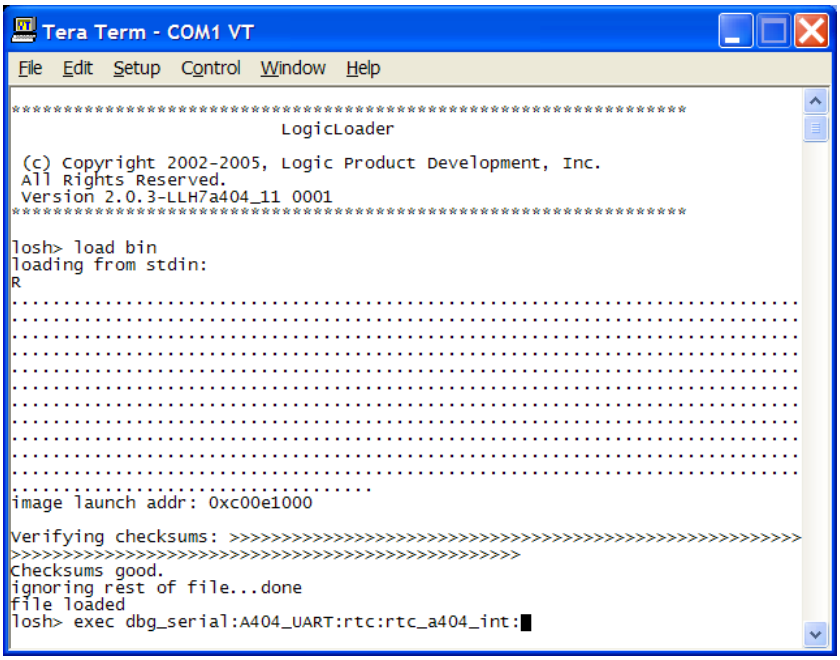


Figure 5.14: LogicLoader "exec" Command

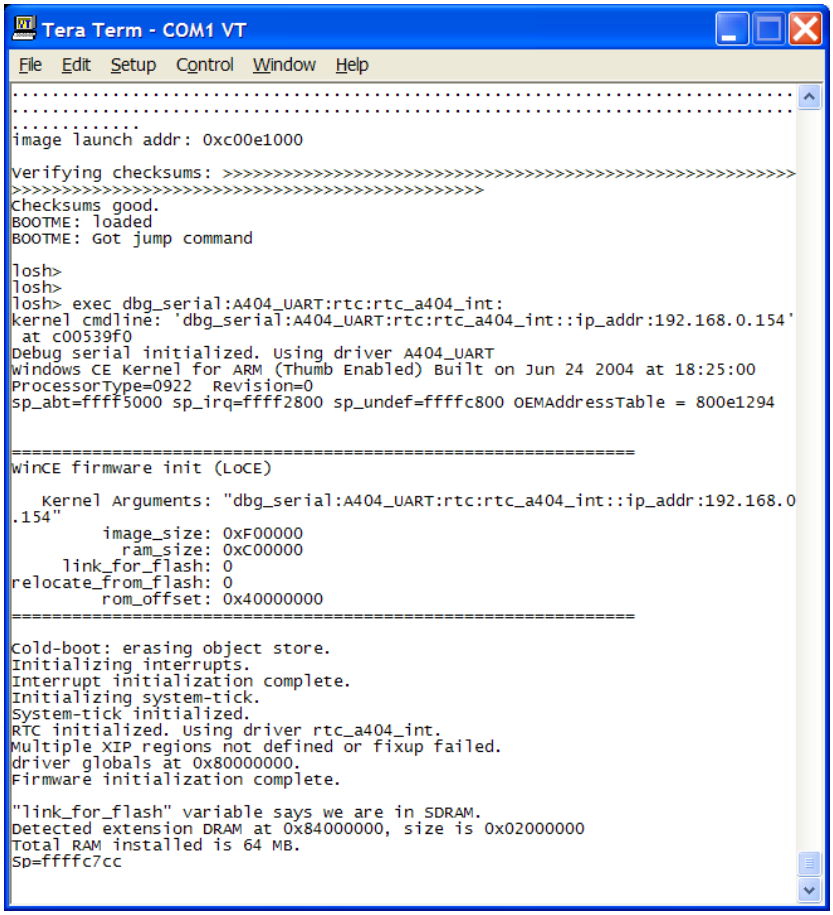


Figure 5.15: Windows Embedded CE Initialization Serial Debug Output

Figure 5.16: Windows Embedded CE Desktop on Device's Display

5.3 Booting Different Image Types (Flash/Relocation Images)

There are two important differences between launching a flash or relocation image and launching a RAM image.

First, after a flash or relocation image is downloaded, LogicLoader's "burn" command must be invoked in order to commit the downloaded image into the SOM's flash memory. This is shown in Figure 5.18 below. It is important to note that burning the CE image into flash can take anywhere from a few seconds to a minute depending on its size.

Secondly, to boot an image that is stored in flash, the address of the image's starting instruction needs to be specified. The syntax of the LogicLoader "exec" command issued should be:

exec [image start address] – [boot parameter string]

Example: **exec 0x00101000 – dbg_uart:A404_UART:**

In the above examples, the image start address can be found after LogicLoader has downloaded an image and has reported the "image launch addr". Figure 5.18 shows "image launch addr: 0x00101000"

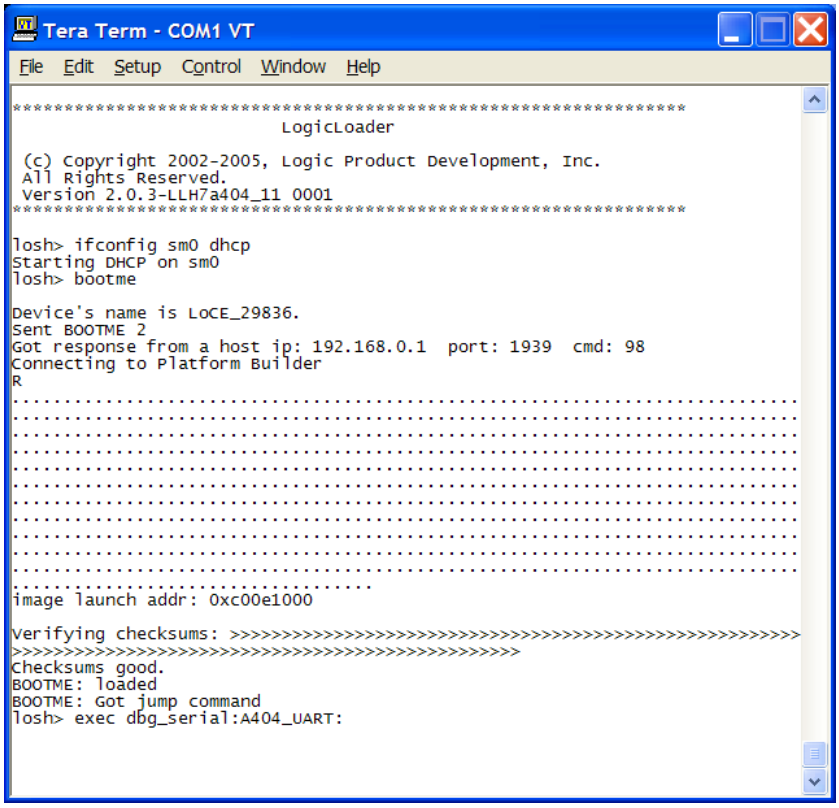


Figure 5.17: RAM Image Download, and "exec" in LogicLoader

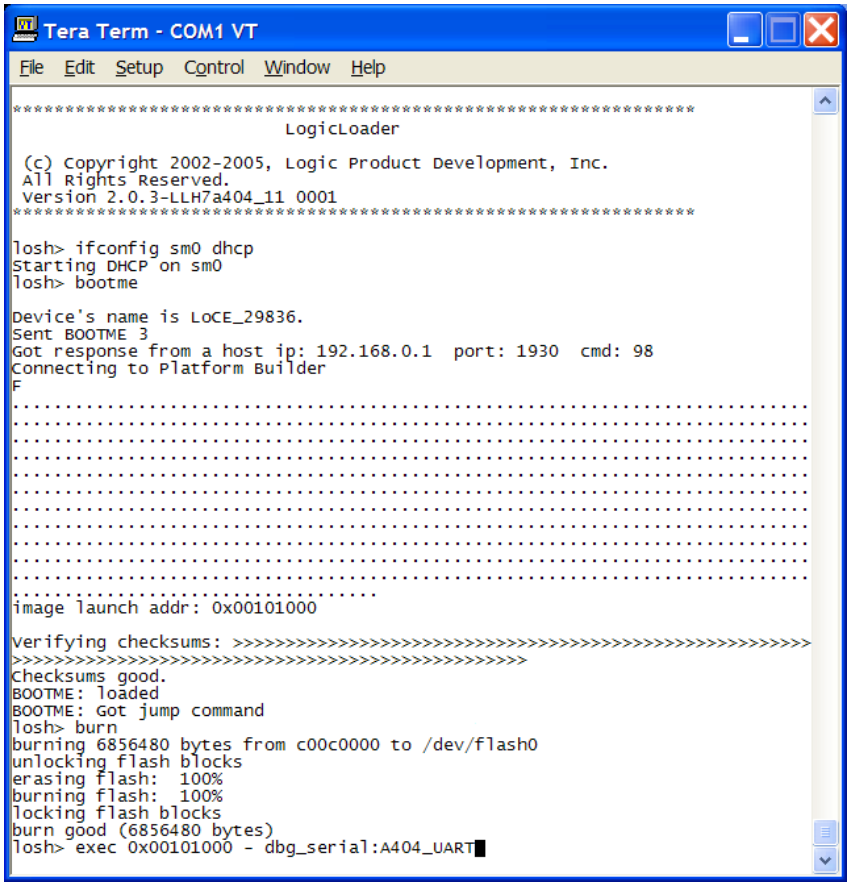


Figure 5.18: Flash Image Download, "burn", and "exec" in LogicLoader

The launch address can also be obtained by using the program "viewbin.exe" to view information about the built Windows Embedded CE image.

1. In Platform Builder, select "Build -> Open Build Release Directory".
2. Type 'viewbin nk.bin'. The "viewbin.exe" program prints out information concerning the built Windows Embedded CE image (nk.bin) as shown in Figure 5.19.

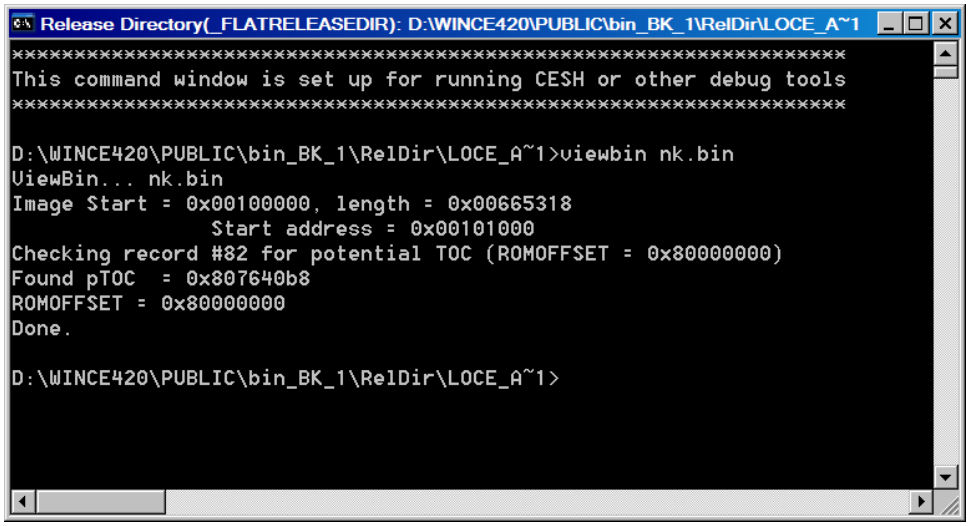


Figure 5.19: Output of the Viewbin Program

In the example shown in Figure 5.19, the starting address is displayed in the line: “Start address = 0x00101000”.

5.4 Boot Parameters

The LoCE Board Support Package takes a parameter string as an argument at boot-time. This string sets several kernel parameters and can also be used through drivers. The parameter-string has the following syntax:

1. Tokens are separated by the colon character ‘:’.
2. Tokens should come in pairs.
3. The first token of the pair denotes the parameter to be set.
4. The second token of the pair denotes the value the parameter should be set to.

An example of a valid LoCE boot parameter string is:

rtc:rtc_a404_int:dbg_serial:A400_UART:dbg_enet:91C111:dbg_enet_base:0x70000000:dbg_enet_irq: 0x00000007:share_eth:1:kitl:true:

5.4.1 Use of Boot Parameters in LogicLoader

When booting Windows Embedded CE, the LogicLoader “exec” command is used. In order to use boot parameters, they should be appended to the “exec” command.

exec [boot parameter string] (RAM Images)

exec [image start address] – [boot parameter string] (Flash Stored Images)

Examples:

losh> exec dbg_serial:A404_UART:dbg_enet:null:rtc:rtc_a404_int:disp_num:5

losh> exec 0x00101000 - dbg_serial:A404_UART:rtc:rtc_a404_int:disp_num:5

losh> exec dbg_serial:A404_UART:dbg_enet:91C111:dbg_enet_base:0x70000000:dbg_enet_irq:0x00000016:rtc:rtc_a404_int:share_eth:1:kitl:true:disp_num:1

5.4.2 Boot Parameter Table

Parameter	Settings	Example	Type	Summary
share_eth	1 / 0 / [not set]	share_eth:1:	I	Enable bridged VMINI w/KITL
kitl	true / false / [not set]	kitl:true:	I	Enable KITL Ethernet Debugger
ip_addr	xxx.yyy.xxx.yyy / [not set]	ip_addr:192.168.0.154:	I	Manual IP Address for Ethernet Debugger
clean_syshive	1 / [not set]	clean_syshive:1:	I	Force clean System Hive on boot
clean_usrhive	1 / [not set]	clean_usrhive:1:	I	Force clean User Hive on boot
dbg_serial	{name} / null / [not set]	dbg_serial:A400_UART:	P	Debug Serial Port

Parameter	Settings	Example	Type	Summary
dbg_enet	{name} / null / [not set]	dbg_enet:91C111:	P	Debug Ethernet Controller
dbg_enet_base	0xXXXXYYYY / [not set]	dbg_enet_base:0x70000000:	P	Debug Ethernet Controller Physical Address
dbg_enet_irq	0xXXXXYYYY / [not set]	dbg_enet_irq:0x0000001A:	P	Debug Ethernet Controller Interrupt
rtc	{name} / rtc_null / [not set]	rtc:rtc_a400_int:	P	RTC Interface
disp_num	x / [not set]	disp_num:5:	D	LCD Driver display mode override
skiplcdcinit	x / [not set]	skiplcdcinit:1	D	LCD Driver initialization override

Type:

I	Platform Independent	This parameter will work across any platform.
P	Platform Dependent	This parameter is specifically defined per platform. Valid settings will be documented in Kernel Usage Notes.
D	Driver Dependent	This parameter is specific to a platform driver. For information regarding availability and usage, the driver document and the driver's .reg file should be used.

* Important Note: All Driver Dependent boot parameters may be not listed here. Each driver document and the driver's .reg file should be examined for more information.

Table 5.1: Boot Parameter Table

More detailed information regarding specific boot parameters can be found in Appendix C: Boot Parameter Definitions.

5.4.3 Using Boot Parameters in Drivers

The LoCE BSP provides functionality to drivers and applications for obtaining values from the boot parameters using the Windows Embedded CE function "KernelIoControl()". To obtain the boot parameter values using this function, one must specify 'IOCTL_LPD_GET_HAL_PARAM_VALUE' as the "dwIoControlCode" parameter. (Please see Section 6.2 for specific usage information.)

5.5 Debugging over KITL using Platform Builder and LogicLoader

LoCE was designed to properly use KITL (Kernel Independent Transport Layer) for Ethernet debugging with Platform Builder. Once a new project has been built, the following steps should be followed.

5.5.1 Connecting to Images in RAM

1. Follow the instructions to download the image over Ethernet as detailed in section "Using LogicLoader to Download the OS Image to RAM over Ethernet".

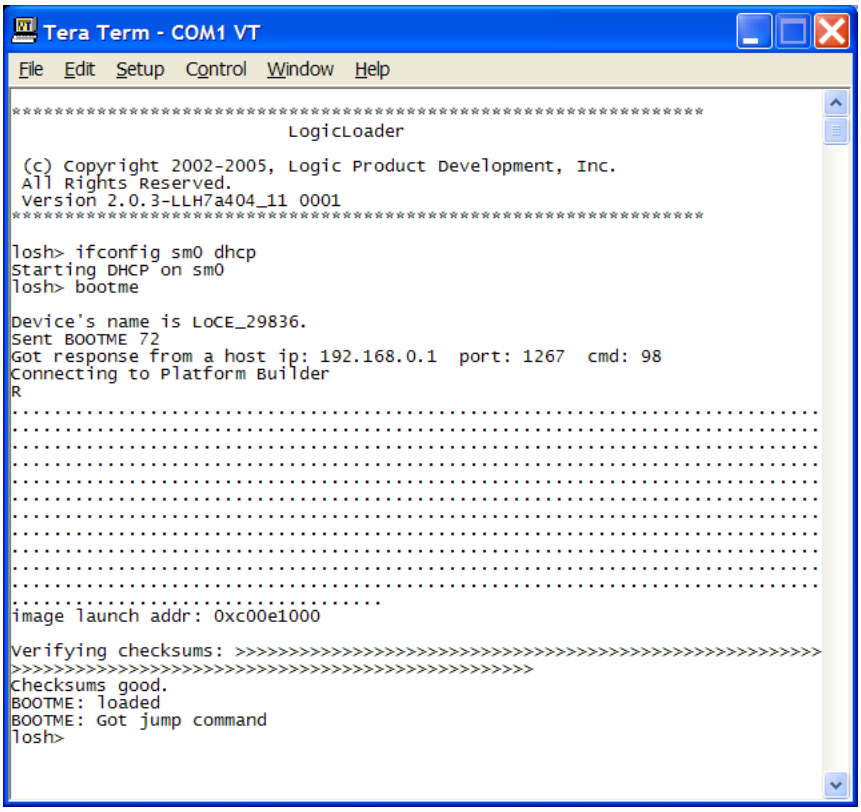


Figure 5.20: LogicLoader Download over Ethernet Complete

2. Use LogicLoader's "exec" command along with the proper boot parameters to start Windows Embedded CE, use the Ethernet Debugger, and connect to KITL.

Example:

losh> exec dbg_serial:A404_UART:dbg_enet:91C111:dbg_enet_base:0x70000000:dbg_enet_irq:0x00000016:rtc:rtc_a404_int:share_eth:1:kitl:true:

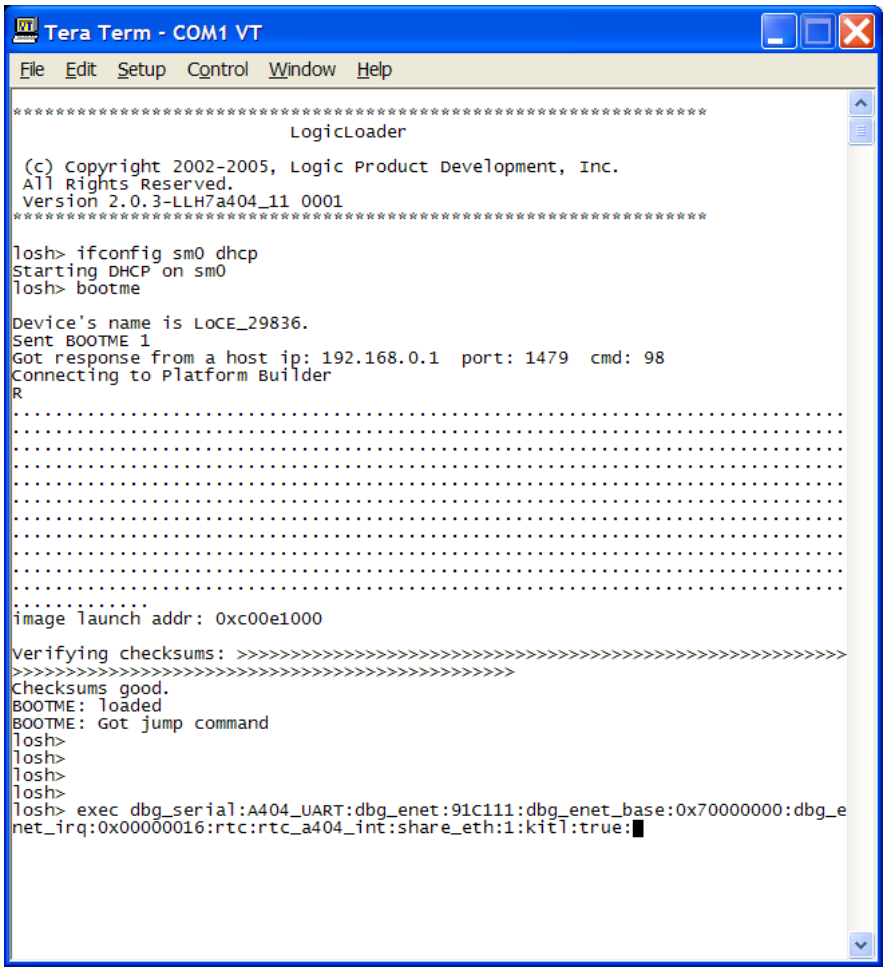


Figure 5.21: LogicLoader "exec" Command with Ethernet Debug and KITL Boot Parameters

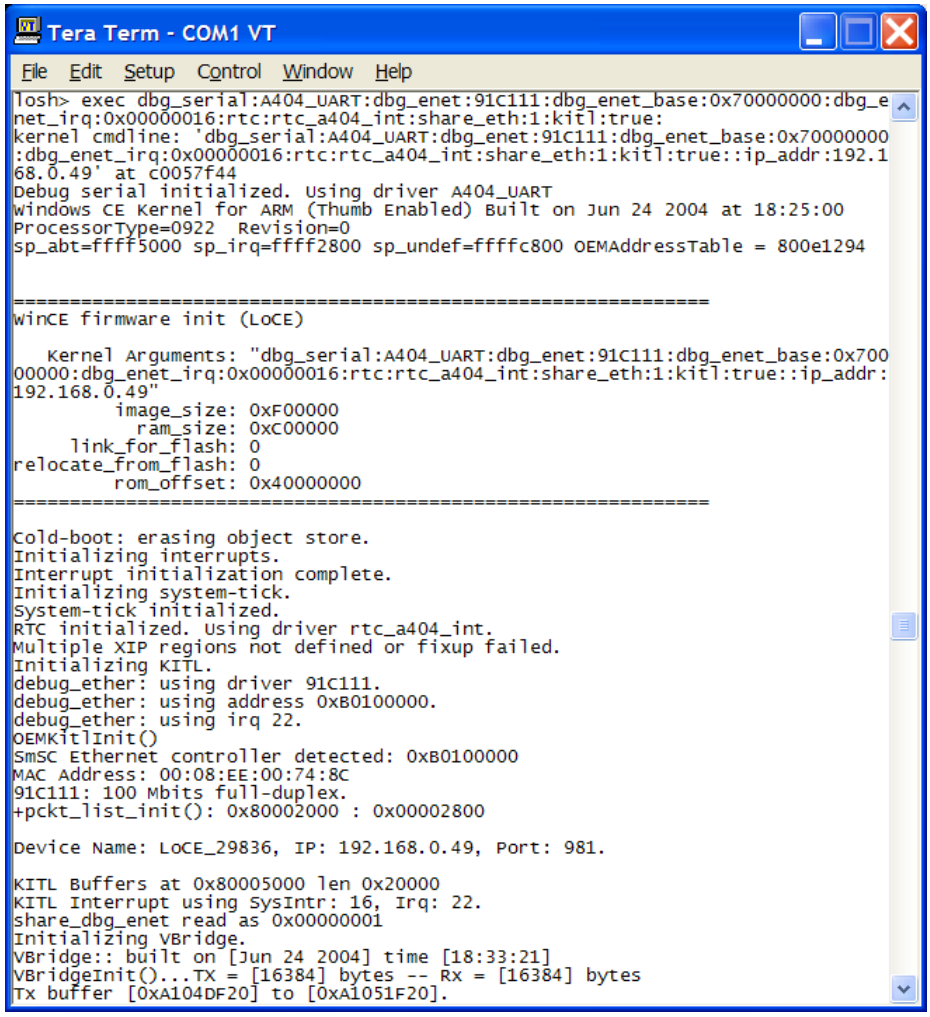


Figure 5.22: Windows Embedded CE Serial Debug Output with KITL Connection

3. Platform Builder should connect. The debug windows will show up in Platform Builder's main window. All of the debugging tools found under the "Tools" menu should be usable.

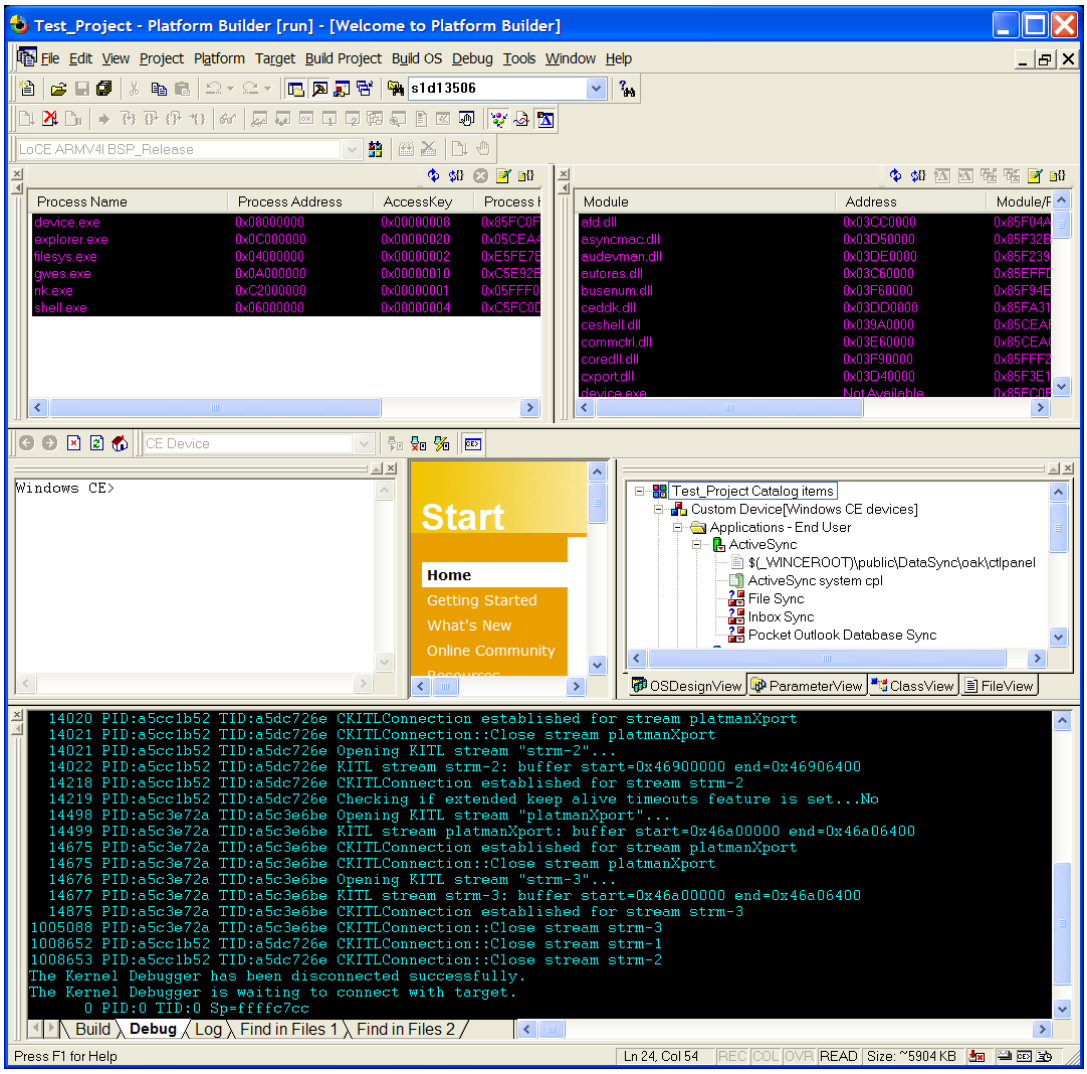


Figure 5.23: Platform Builder Debugging View

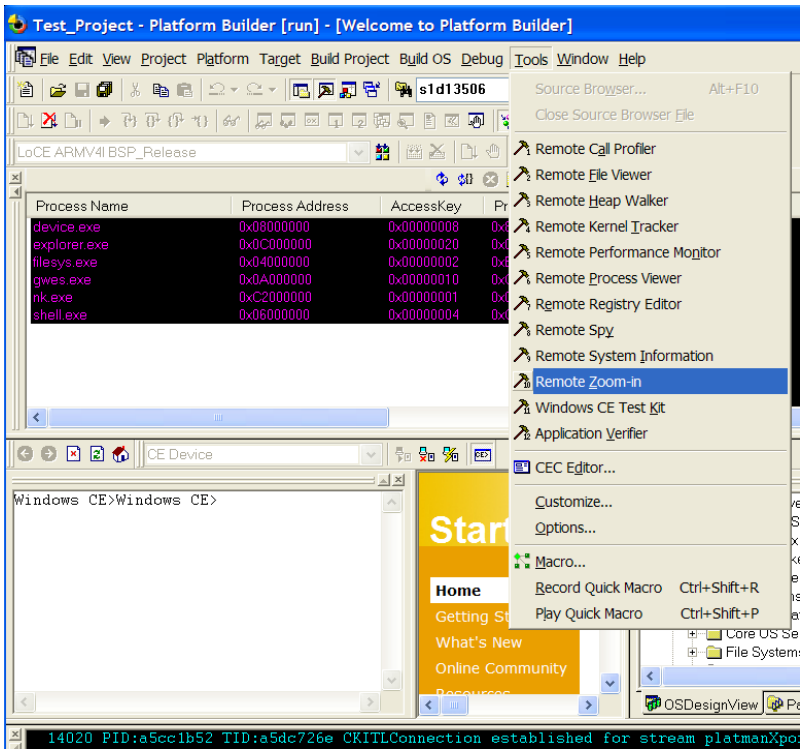


Figure 5.24: Platform Builder "Tools -> Remote Tools"

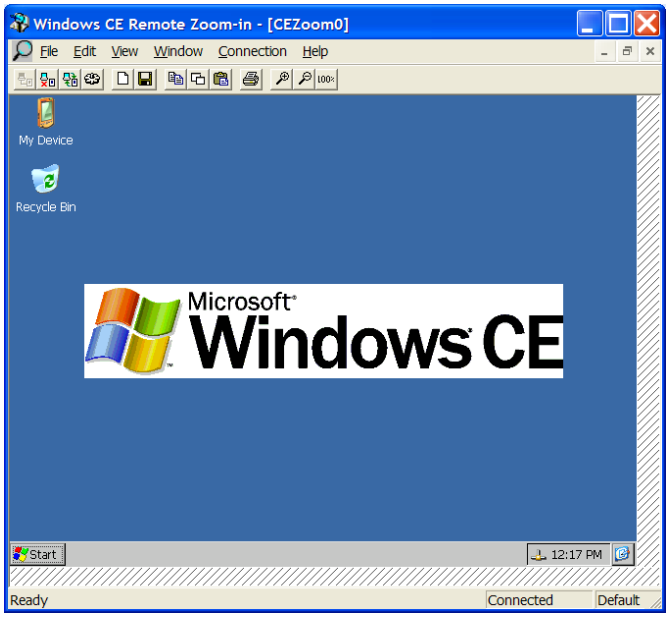


Figure 5.25: Platform Builder "Remote Zoom-In" Screen-Shot

6 Development with LoCE

6.1 LoCE Directory Structure

The LoCE Board Support Package consists of a Base BSP which is the build infrastructure, the Windows Embedded CE Hardware Adaptation Layer (HAL) (also referred to as the OAL – OEM Adaptation Layer, and listed as the “kernel” component), as well as device drivers which support the various Logic Product Development SOM hardware platforms. Logic Product Development has made a concerted effort to develop software packages that are loosely coupled and have a high degree of cohesion. This is evidenced in the structure of the LoCE BSP.

All platform specific software components can be found in the ‘x:\WINCE500\Platform\LoCE’ directory.

6.2 LoCE Kernel IOCTLs

The LoCE kernel implements most of the standard kernel IOCTLs defined by Microsoft. A list of supported IOCTLs can be found at the end of this document in Appendix D. For more information on these IOCTL codes, please refer to the Microsoft MSDN documentation at http://msdn.microsoft.com/library/en-us/wcehardware5/html/wce50grfOALIOCTLs.asp

In certain cases, particular IOCTLs extend their standard implementations. The LoCE kernel implements additional IOCTLs to access LoCE specific functionality.

The IOCTLs can be invoked from both the driver context and the application level.

The following subsections will describe standard IOCTLs with extended functionality and those specific to LoCE.

6.2.1 IOCTL_LPD_GET_HAL_PARAM_VALUE

This input/output control code allows an application or device driver to query a value within the boot configuration string. An example usage scenario is having a display driver query the value of the default display it should open, regardless of the corresponding registry setting.

Parameters:

dwIoControlCode
[in] Set to ‘IOCTL_LPD_GET_HAL_PARAM_VALUE’.

lpInBuf
[in] Set to the address of the input buffer, which contains an ASCII character string containing the boot-string parameter name to be queried.

nInBufSize
[in] Set to the character length of the string “lpInBuf” points to.

lpOutBuf
[out] Set to the address of the output buffer supplied by the caller. When the function returns success, this buffer will contain the ASCII character string value of the queried boot-string parameter.
If the size of the requested information exceeds the buffer size specified in ”nOutBufSize”, the return value will be FALSE and “GetLastError” will return the ERROR_INSUFFICIENT_BUFFER flag.

nOutBufSize
[in] Set to the size of the buffer “lpOutBuf” points to. The buffer must be large enough to contain the queried string value plus a NULL termination character.

lpBytesReturned

[out] Set to the address of a DWORD that receives the size, in bytes, of the returned NULL terminated string.

Return values:

TRUE indicates success. FALSE indicates failure.

Remarks:

This IOCTL allows the caller to obtain a string-value of arbitrary length associated with the requested boot-string parameter name. It is recommended that you query the system for the string size so that you can allocate the right buffer size, and then get the actual string data.

To query the string size:

1. Set “lpOutBuf” to NULL and “lpBytesReturned” parameter to the address of a DWORD.
2. Call “KernelIoControl” using ‘IOCTL_LPD_GET_HAL_PARAM_VALUE’.

The “KernelIoControl” call will return FALSE and “GetLastError” will return ERROR_INSUFFICIENT_BUFFER. The “lpBytesReturned” will contain the number of bytes needed for the queried string-value.

To get the string-value:

Allocate a buffer large enough and then call “KernelIoControl” using: ‘IOCTL_LPD_GET_HAL_PARAM_VALUE’.

The data will be copied to “lpOutBuf”. If “lpBytesReturned” is not NULL, then this address will contain the size of the data copied to the buffer. If this process is successful, the function returns TRUE. If an error occurs, this function returns FALSE and sets “GetLastError”. If “GetLastError” returns ERROR_INVALID_DATA, then the queried parameter was not found in the boot-string.

Usage example:

```
#define IOCTL_LPD_GET_HAL_PARAM_VALUE \
CTL_CODE(FILE_DEVICE_HAL,('l'+'p'+'d'+0),METHOD_BUFFERED,FILE_ANY_ACCESS)

{
    BOOL bSuccess;
    DWORD dwRetBytes;
    char* pName = "disp_num";

    dwRetBytes = 0;
    bSuccess = KernelIoControl(
             IOCTL_LPD_GET_HAL_PARAM_VALUE
            ,pName ,strlen(pName)
            ,NULL ,0
            ,&dwRetBytes
    );

    if(ERROR_INSUFFICIENT_BUFFER == GetLastError() && dwRetBytes > 0)
    {
            DWORD dwBufSize = dwRetBytes;
            char* pValue = (char*)malloc(dwBufSize);
            dwRetBytes = 0;

            bSuccess = KernelIoControl(
                    IOCTL_LPD_GET_HAL_PARAM_VALUE
```

```
                        ,pName ,strlen(pName)
                        ,pValue ,dwBufSize
                        ,&dwRetBytes
                );

                if(bSuccess)
                {
                        unsigned int disp_num;
                        //convert from string to hex number
                        sscanf(pValue, "%x", &disp_num);
                        RETAILMSG(1, (TEXT("Boot string calls for display: %x\r\n"),
                        disp_num));
                }
        }
    }
```

6.2.2 IOCTL_HAL_REBOOT

This IOCTL implements the Microsoft defined functionality. As an extended feature, it gives the caller the option to specify a cold boot instead of a warm boot. A request to perform a warm or cold boot is made by calling the “KernelIoControl” function with ‘IOCTL_HAL_REBOOT’.

Parameters:

dwIoControlCode
[in] Set to ‘IOCTL_HAL_REBOOT’.

lpInBuf
[in] Optional. “lplnBuf” Points to a DWORD variable specifying whether a cold or warm boot is to be performed. Set the value of the DWORD variable to ‘1’ for cold boot and ‘0’ for warm boot. If the pointer is set to NULL, the function will perform a warm boot.

nInBufSize
[in] Size of buffer “lpInBuf” points to.

lpOutBuf
[out] Set to NULL.

nOutBufSize
[in] Set to zero.

lpBytesReturned
[out] Set to NULL.

Return values:

If the system is successfully reset, this function will not return. If the system fails to reset, it will return FALSE.

Usage example:

```
/*-------------------------------------------------------------------------*/
/* Perform a hard reset
 */
{
        BOOL ret_value;
        DWORD dwResetType = 1;

        #define IOCTL_HAL_REBOOT \
        CTL_CODE(FILE_DEVICE_HAL,('l'+'p'+'d'+0),METHOD_BUFFERED,FILE_ANY_ACCESS)

        ret_value = KernelIoControl(IOCTL_HAL_REBOOT
```

```
                              ,&dwResetType
                              ,sizeof( dwResetType )
                              ,NULL
                              ,0
                              ,NULL);

        if ( !ret_value )
        {
                RETAILMSG(1, (TEXT("Hard reset failed! \r\n")));
        }
}
/*-----------------------------------------------------------------------*/
```

7 Using LoCE with Embedded Visual C++

7.1 Creating an Software Development Kit

In order to do application development in Microsoft's Embedded Visual C++ environment you will need to build an SDK (Software Development Kit) in Platform Builder from your OS image.

Building an SDK

1. In Platform Builder, go to "Platform -> SDK -> New SDK". When the dialogue box appears, click "Next".

Figure 7.1: SDK Wizard

2. Provide information to uniquely identify your SDK and click "Next".

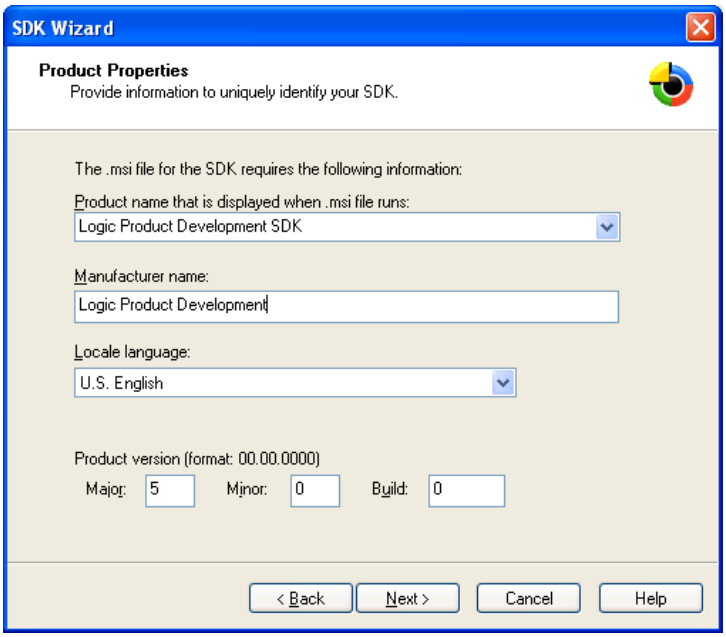


Figure 7.2: SDK Identification

3. Select the development languages that you want your SDK to support and click “Next”.

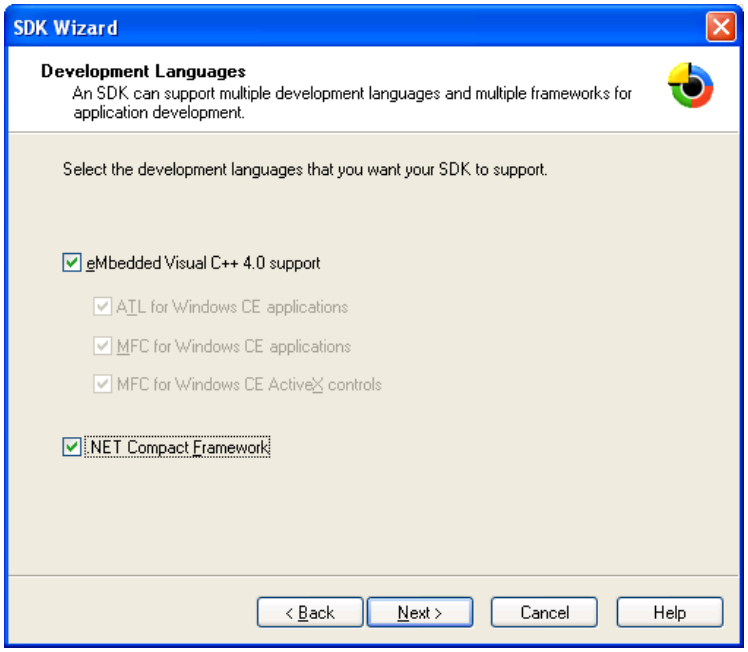


Figure 7.3: SDK Supported Languages

4. You will be presented with a dialogue that gives you two options on how to proceed. You can either: “close the SDK Wizard and build your SDK” or “close the Wizard and continue to configure your SDK”. For this example, we will close the SDK Wizard and build the SDK. Once the SDK has finished building, the .msi installation file for the SDK will be located in the workspace folder of the Windows Embedded CE workspace you are currently using. For example, the path to this file could be: ‘\WINCE500\PBWorkspaces\A404_Example\SDK’
5. Double-click on the .msi installation file to install the SDK on your system.

7.2 Using Embedded Visual C++

The Zoom Development Kit Windows Embedded CE Evaluation Guide details the steps required to build an example application and download it via ActiveSync.

8 Going Forward

Congratulations, if you have read everything up to this point, you are well on your way to becoming extremely productive with both Windows Embedded CE and the Logic Product Development LoCE Board Support Package. If something went wrong, please review this document and check all of your hardware settings. Also please note that the Platform Builder tool build environment takes time to learn and the use of Microsoft's help channels (MSDN, Platform Builder Help) are essential to a user's learning experience. If you need more hands-on help, please visit the Logic Product Development's support website for more information.

Please check the Logic Product Development website often for information on new releases. Logic also periodically releases new documentation and white papers designed to help you, the developer, get the most out of our products. Look for tutorials on how to build specialized configurations of the Windows Embedded CE operating system that take full advantage of the LoCE BSP running on the SOM hardware.

We at Logic Product Development sincerely hope that you will take the time to both learn how to use the LoCE Board Support Package, as well as provide us with feedback on how to improve it.

Best regards,

Logic Product Development

Appendix A: Definitions

BSP: A Board Support Package is formed with customized code components that are made up of drivers, an OAL, files, and a folder structure, and is created for a specific hardware device.

Driver: Software that is specific to a physical peripheral or virtual device that allows the Operating System and applications to communicate through standard functions.

Image: The binary program file, NK.bin, built by Platform Builder that includes the entire Operating System, drivers, files, and applications.

Kernel: The core of the CE Operating System. For more information regarding the kernel, follow this MSDN link. The LoCE Kernel component is the OAL for the specific hardware device that allows the Kernel to communicate with it.

KITL: Kernel Independent Transport Layer. For more information regarding the KITL, follow this MSDN link.

LoCE: Logic Product Development Windows Embedded CE BSP

OAL: The OEM Adaptation Layer is the layer of code that allows the Windows Embedded CE kernel to communicate with the hardware. This includes handling interrupts, timers, IOCTLs, the RTC, etc.

SOM: System on Module

Appendix B: Additional Resources

Microsoft Support

- Platform Builder for Microsoft Windows Embedded CE Help (Installed with P.B. installation)

Books

- Windows Embedded CE Texts: A Reference Guide (Logic Product Development White Paper #215)

Web Resources

- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/windowsce.asp
- http://msdn.microsoft.com/embedded/windowsce/default.aspx

Logic Product Development Support

- http://www.logicpd.com/support/
- Windows Embedded CE Sample Images – Found on a specific product's download page (http://www.logicpd.com/auth/) under the "Windows Embedded CE Sample Images" Section. These can be used to verify a "known good" image on the kit.
- Zoom Development Kit WindowsCE Evaluation Guide – This document walks through using Logic's Windows Embedded CE Sample Images.
- LoCE Windows Embedded CE BSP QuickStart Guide – Document walks through a simple Platform Builder project build and then running it on a Zoom Development Kit.
- Windows Embedded CE Driver Description Documents – Found on a specific product's download page (http://www.logicpd.com/auth/) under the "Windows Embedded CE Documents" Section. These documents are written for each specific driver and detail the use of that driver. (Not all drivers have a Driver Description Document.)
- Windows Embedded CE-specific Application Notes and White Papers – Found on a specific product's download page (http://www.logicpd.com/auth/) under the "Application Notes" and "White Paper" Sections. Example topics include: application debugging, custom display integration, hive registry, boot-time quantification.
- Windows Embedded CE Sample Code – Found on a specific product's download site (http://www.logicpd.com/auth/) under the "Windows Embedded CE Sample Code and Add Ons" Section.

Appendix C: Boot Parameter Definitions

Platform Independent Boot Parameters

Platform Independent Boot Parameters are independent of the platform being used. All valid settings can be found in this document.

share_eth

This parameter determines whether or not the BSP should multiplex application network traffic with debug network traffic. In order for this to work correctly, the environment variables "BSP_NOSHAREETH" and "IMGNOSHAREETH" must not be set to any value. This enables the VMINI Ethernet bridging functionality in the Windows Embedded CE OAL. Enabling this functionality allows the user to use the Ethernet port for application development while also using it for Windows Embedded CE debugging capabilities. In this way, the bandwidth is shared between an application and the debugger.

Valid settings:

1	Share debug and application network traffic.
0	Don't share debug and application network traffic.
[not set]	Same as '0'

Example boot-string:

share_eth:1

kitl

This parameter is used to enable the KITL debugging interface. This must be set to 'true' in order to use the Ethernet debugger.

Valid settings:

true	KITL enabled
false	KITL disabled
[not set]	Same as 'false'

Example boot-string:

kitl:true

ip_addr

This parameter is used to provide an IP address for the kernel Ethernet debugger to use.

Important Note: LogicLoader will automatically append this to the boot-string if the "bootme" command has been used. Please see the *LogicLoader User's Guide* for more information.

Valid settings:

xxx.xxx.xxx.xxx	IP Address to be used
[not set]	Debug Ethernet will attempt to get an IP address.

Example boot-string:

ip_addr:192.168.0.154

clean_syshive

This parameter is used by the Kernel to determine if the kernel should force a System Hive registry clean at boot time. If it is not set, Windows Embedded CE will decide if it is appropriate or not to clean this registry.

Valid settings:

1	Windows Embedded CE will clean the System Hive registry.
[not set]	Windows Embedded CE will decide if it is appropriate to clean the System Hive registry

Example boot-string:

clean_syshive:1

clean_usrhive

This parameter is used by the Kernel to determine if the kernel should force a User Hive registry clean at boot time. If it is not set, Windows Embedded CE will decide if it is appropriate or not to clean this registry.

Valid settings:

1	Windows Embedded CE will clean the User Hive registry.
[not set]	Windows Embedded CE will decide if it is appropriate to clean the User Hive registry

Example boot-string:

clean_usrhive:1

Platform Dependent Boot Parameters

Platform Dependent Boot Parameters depend of the platform being used. All valid settings to use the specific parameter are found in the “User Notes” for each specific kernel release.

dbg_serial

This parameter denotes the serial (UART) port Windows Embedded CE will use to output debug messages.

Valid settings:

xxxxxxx	*Platform Specific Value [Name]
null	Don't use any debug serial port
[not set]	Same as 'null'

Example boot-string:

dbg_serial:A400_UART:

dbg_enet

This parameter denotes which Ethernet controller the kernel should use as a debug connection with Platform Builder.

Valid settings:

xxxxxxx	*Platform Specific Value [Name]
null	Don't use any debug Ethernet controller
[not set]	Same as 'null'

Example boot-string:

dbg_enet:91C111

dbg_enet_base

This parameter denotes the physical address of the debug Ethernet controller.

Valid settings:

0xXXXXYYYY	*Platform Specific Value [Hex Number]
[not set]	Debug Ethernet routines will not access the chip at the correct address. This must be set to use the debug Ethernet interface.

Example boot-string:

dbg_enet_base:0x70000000

dbg_enet_irq

This boot-string parameter denotes the Processor's IRQ number that the Ethernet controller uses. The value is listed in the IRQ source table found in the specific product's header file.

(Header file location: \WINCE500\PLATFORM\LoCE\src\inc\card_engine\xxxxxx_xx.h)

Valid settings:

0xXXXXYYYY	*Platform Specific Value [Hex Number]

[not set]	Debug Ethernet routines will not initialize the proper interrupt. This must be set to use the debug Ethernet interface.

Example boot-string:

dbg_enet_irq:0x0000001A

rtc

This parameter denotes the Real Time Clock (RTC) peripheral interface that Windows Embedded CE will use.

Valid settings:

xxxxxxx	*Platform Specific Value [Name]
rtc_null	No RTC used.
[not set]	Same as 'null'

Example boot-string:

rtc:rtc_a400_int

coproc

This parameter controls the activation of coprocessors available in the system. The coproc parameter is only available on ARM platforms; however, not all ARM platforms implement the same coprocessors. Please reference the specific ARM processor's user manual for more details.

The coproc parameter takes a 32-bit number and directly writes it to the ARM specific Coprocessor Access Control Register with the following instruction:

mcr p15, 0, r0, c1, c0, 2

where r0 holds the coproc parameter value.

The Coprocessor Access Control Register holds fourteen two-bit fields, each specifying access rights for one particular coprocessor. Each bit field can assume one of the following settings:

Valid settings:

xxxxxxxx	*To enable a coprocessor, specify a two-bit bit field for the coprocessor to be enabled.
00	Disabled
01	Privileged mode (supervisor) access only
10	Reserved
11	Full access
[not set]	Same as '00'

Example boot-string:

The following example boot-string grants full access to coprocessor 7 and privileged mode to coprocessor 11:

coproc:0040C000

Driver Dependent Boot Parameters

Driver Dependent Boot Parameters are specific to drivers used. Not all drivers utilize them and this document lists several examples and should not be consider exhaustive. For valid settings and details to use these parameters, the driver description document and the driver .reg file should be referenced.

disp_num

This parameter over-writes the video mode set in the registry for the LCD panel. In this way, the video mode can be changed with the boot-string instead of recompiling the Windows Embedded CE image to use a different display.

Valid settings:

x	*Platform/Driver Specific Value [Decimal Number] Please see the specific LCD driver registry file for appropriate settings.
[not set]	LCD driver will use display mode set in the registry.

Example boot-string:

disp_num:5

skiplcdcinit

This parameter over-writes the video initialization parameter set in the registry for the LCD panel. With this parameter, the LCD driver can be directed to initialize the hardware in different ways. For example, the driver can be told to not initialize any hardware registers; therefore, if a bootloader initialized the LCD controller already, it will stay “as is” through the boot process.

Valid settings:

x	*Platform/Driver Specific Value [Decimal Number] Please see the specific LCD driver registry file for appropriate settings.
[not set]	LCD driver will use value set in the registry.

Example boot-string:

skiplcdcinit:1

Appendix D: Supported Kernel IOCTLs

CEDDK IOCTLs

- IOCTL_HAL_DDK_CALL
- IOCTL_HAL_RELEASE_SYSINTR
- IOCTL_HAL_REQUEST_IRQ
- IOCTL_HAL_TRANSLATE_IRQ
- IOCTL_HAL_REQUEST_SYSINTR

Device Information IOCTLs

- IOCTL_HAL_GET_DEVICE_INFO
- IOCTL_HAL_GET_DEVICEID
- IOCTL_HAL_GET_UUID
- IOCTL_HAL_SET_DEVICE_INFO
- IOCTL_PROCESSOR_INFORMATION

Filesys.exe IOCTLs

- IOCTL_HAL_GET_HIVE_CLEAN_FLAG

KITL IOCTLs

There are currently no supported KITL IOCTLs.

Power Manager IOCTLs

- IOCTL_HAL_DISABLE_WAKE
- IOCTL_HAL_ENABLE_WAKE
- IOCTL_HAL_REBOOT

  Note: The implementation of this IOCTL is extended. Please refer to Section 6.2.2 for further details.

Security IOCTLs

- IOCTL_HAL_AUTHENTICATE_DEVICE

Tools IOCTLs

- IOCTL_HAL_ILTIMING

VMINI IOCTLs

- IOCTL_VBRIDGE_802_3_MULTICAST_LIST
- IOCTL_VBRIDGE_CURRENT_PACKET_FILTER
- IOCTL_VBRIDGE_GET_ETHERNET_MAC
- IOCTL_VBRIDGE_GET_RX_PACKET

- IOCTL_VBRIDGE_GET_RX_PACKET_COMPLETE
- IOCTL_VBRIDGE_GET_TX_PACKET
- IOCTL_VBRIDGE_GET_TX_PACKET_COMPLETE
- IOCTL_VBRIDGE_SHARED_ETHERNET
- IOCTL_VBRIDGE_WILD_CARD

Other OAL IOCTLs

- IOCTL_HAL_GET_CACHE_INFO
- IOCTL_HAL_INIT_RTC
- IOCTL_HAL_POSTINIT

LogicPD-specific IOCTLs

- IOCTL_LPD_GET_HAL_PARAM_VALUE
  Note: Please refer to Section 6.2.1 for further details.