

## Read This First

---

---

---

### ***About This Manual***

This manual provides installation, information, usage and operating instructions for a Windows Embedded CE 6.0 BSP built for the Texas Instruments OMAP35xx Evaluation Module (EVM). It also includes information and usage instructions for the Texas Instruments EVMFlash Application.

### ***How to Use This Manual***

Users are advised to become familiar with the complete contents of this manual prior to working with the BSP and the EVM. Users should also be advised that this manual contains insufficient information to work with the Windows Embedded CE 6.0 R2 Operating System and associated tools such as Platform Builder.

### ***Information About Cautions and Warnings***

This book may contain cautions and warnings.

**This is an example of a caution statement.**

**A caution statement describes a situation that could potentially damage your software or equipment.**

**CAUTION**

**This is an example of a warning statement.**

**A warning statement describes a situation that could potentially cause harm to you.**

**WARNING**

The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

### ***If You Need Assistance***

For technical support, registered users should contact their local Texas Instruments FAE, or go to our website at <http://www.bsquare.com/omap3>.

For helpful developer hints, try our collaborative Wiki site at <http://wiki.omap.com/index.php?title=WinCE>.

### ***Trademarks***

Microsoft® and Windows® are Registered Trademarks of Microsoft Corporation. OMAP™ is a Trademark of Texas Instruments Inc. Samsung™ and OneNAND™ are Trademarks of Samsung Group.

# Contents

---

---

---

<b>Read This First</b> .....	<b>iii</b>
About This Manual .....	iii
How to Use This Manual .....	iii
Information About Cautions and Warnings .....	iii
If You Need Assistance .....	iv
Trademarks .....	iv
<b>Contents</b> .....	<b>v</b>
<b>Figures</b> .....	<b>vii</b>
<b>Tables</b> .....	<b>vii</b>
<b>Introduction</b> .....	<b>1-1</b>
1.1 Requirements .....	1-1
1.1.1 Required Experience.....	1-1
1.1.2 Required Software .....	1-1
1.1.3 Required JTAG Tools.....	1-2
1.1.4 Required Network Capabilities.....	1-2
1.2 Notation Conventions .....	1-2
1.2.1 CPU Name .....	1-2
1.2.2 Other Terms .....	1-2
1.3 Related Documentation .....	1-2
1.3.1 Visual Studio 2005 / Platform Builder Documentation .....	1-2
1.3.2 EVM Schematic.....	1-2
<b>BSP Features</b> .....	<b>2-1</b>
2.1 Bootloader (EBOOT) .....	2-1
2.1.1 Bootloader and Initialization Sequence.....	2-1
2.1.2 Low Level Initialization .....	2-2
2.1.3 Device Configuration Parameters .....	2-2
2.1.4 Serial User Interface .....	2-2
2.2 NAND Flash Organization .....	2-3
2.2.1 Reserved Area (Bootloader/Configuration Parameters) .....	2-4
2.2.2 OS Partition.....	2-4
2.2.3 File System Partition .....	2-4
2.3 OAL Features .....	2-5
2.3.1 Serial Debug Message.....	2-5
2.3.2 KITL.....	2-5
2.3.3 VMINI .....	2-5
2.4 Drivers.....	2-6
2.4.1 GPIO Driver.....	2-6
2.4.2 Display Driver .....	2-6
2.4.3 Backlight Driver .....	2-6
2.4.4 I2C Bus Driver.....	2-6
2.4.5 SPI Driver.....	2-6
2.4.6 FLASH Media Driver .....	2-7
2.4.7 SDIO Host Controller Interface Driver.....	2-7
2.4.8 Serial Driver.....	2-7
2.4.9 TSC2046 Touch Driver .....	2-7

2.4.10	USBOTG Driver.....	2-7
2.4.11	TWL4030 Wave Driver .....	2-7
2.4.12	Keyboard .....	2-7
2.5	Unsupported Hardware Features .....	2-8
<b>3</b>	<b>BSP Installation .....</b>	<b>3-1</b>
3.1	Install Required Software .....	3-1
3.2	Installing the EVM BSP .....	3-1
3.2.1	Version 6.11 or earlier .....	3-1
3.2.2	Version 6.12 or later .....	3-1
<b>4</b>	<b>Building Images.....</b>	<b>4-1</b>
4.1	Create an OS Design Using Platform Builder .....	4-1
4.1.1	Using the Sample OS Design.....	4-1
4.1.2	Creating a New OS Design .....	4-2
4.2	Building the BSP .....	4-2
4.2.1	Procedure .....	4-2
4.2.2	Checking For Excessive Image Size.....	4-3
4.3	Output File .....	4-3
4.3.1	NK.BIN.....	4-3
4.3.2	XLDRONENAND.BIN/XLDRNAND.BIN .....	4-4
4.3.3	EBOOTONENAND.BIN/EBOOTNAND.BIN .....	4-4
4.3.4	TIEVM3530-ONENAND.RAW/TIEVM3530-NAND.RAW .....	4-4
4.3.5	Locating the Image Files .....	4-4
<b>5</b>	<b>Loading Images .....</b>	<b>5-1</b>
5.1	Programming Bootloader Using Boot ROM .....	5-1
5.2	Downloading Images using Platform Builder .....	5-1
5.2.1	EVM Setup .....	5-1
5.2.2	Setup Platform Builder .....	5-2
5.2.3	Configure Platform Builder Connectivity Options .....	5-2
5.2.4	Download Image .....	5-3
<b>6</b>	<b>Booting the EVM.....</b>	<b>6-1</b>
6.1	Booting from OneNAND Flash .....	6-1
6.2	Booting from NAND Flash .....	6-2
6.3	Booting from SD Memory Card .....	6-3
<b>7</b>	<b>Debugging.....</b>	<b>7-1</b>
7.1	Debugging the Bootloader.....	7-1
7.2	Debugging the Operating System .....	7-1
7.2.1	Image Configuration .....	7-2
7.2.2	Ethernet KITL .....	7-2
7.2.3	USB RNDIS KITL .....	7-2
7.2.4	Serial Debug Messages .....	7-2
<b>8</b>	<b>EVMFlash Application.....</b>	<b>8-1</b>
8.1	Installing EVMFlash .....	8-1
8.1.1	If the target EVM board is TMDXEVM3503 (EVM1) populated with OneNAND Flash:.....	8-1
8.1.2	If the target EVM board is TMDXEVM3503 (EVM1) populated with NAND Flash:.....	8-1
8.1.3	If the target EVM board is TMDSEVM3530 (EVM2) populated with NAND Flash:.....	8-1
8.2	EVMFlash Features Supported .....	8-2
8.2.1	Flashing the BSP bootloader over UART .....	8-2
8.2.2	Flashing the BSP bootloader over USB .....	8-2
8.3	Getting familiar with a EVM board .....	8-3
8.4	Bootloader Flashing using EVMFlash.....	8-5

# Figures

---

---

---

Figure 1.	EVM Bottom View .....	8-3
Figure 2.	EVM Top View .....	8-4
Figure 3.	EVMFlash Application, main window. ....	8-7

# Tables

---

---

---

Table 1.	VK_... code mapping.....	2-8
Table 2.	SW4 Switch Positions, OneNAND Flash Boot .....	6-1
Table 3.	SW4 Positions, Serial Cable EVM Flash Tool. OneNAND Boot.....	6-1
Table 4.	SW4 Positions, USB Cable EVM Flash Tool. OneNAND Boot.....	6-2
Table 5.	SW4 Switch Positions, NAND Flash Boot .....	6-2
Table 6.	SW4 Positions, Serial Cable EVM Flash Tool. NAND Boot .....	6-2
Table 7.	SW4 Positions, USB Cable EVM Flash Tool. NAND Boot.....	6-2
Table 8.	SW4 Switch Positions, SD Card Boot.....	6-3
Table 9.	SW4 Positions, Serial Cable EVM Flash Tool. SD Card Boot.....	6-4
Table 10.	SW4 Positions, USB Cable EVM Flash Tool. SD Card Boot.....	6-4



# Introduction

---



---



---

This document provides information for the user of the Microsoft® Windows® Embedded CE 6.0 BSP (BSP) for the Texas Instruments OMAP™35xx Evaluation Module (EVM). Instructions are provided for installing the BSP, building images and loading images onto the EVM.

Topic	Section
Requirements .....	1.1
Notation Conventions.....	1.2
Related Documentation.....	1.3

## 1.1 Requirements

There are many requirements for software, hardware, infrastructure and user experience level that should be met prior to working with EVM BSP and Windows CE.

### 1.1.1 Required Experience

This document is not intended for novice Windows CE users. Readers unfamiliar with Windows CE are encouraged to seek training and to study the documentation included with Windows CE before using the EVM BSP and related documentation.

### 1.1.2 Required Software

The following software is required prior to installing the EVM3530 BSP. Visit Microsoft website for more information about the listed software and known issues with the installation process:

- Windows Embedded CE 6.0

Installation instructions can be found in the Windows CE 6 Installation guide. For more information about Windows CE and how to obtain the software visit Microsoft website:

<http://msdn.microsoft.com/en-us/windowseembedded/ce/dd430902.aspx>

### **1.1.3 Required JTAG Tools**

There are no JTAG tools required for use with the EVM3530. Texas Instruments provides a tool that can be used to communicate with the internal boot ROM on the OMAP35xx processor called EVM Flash Tool. This tool can be used to program the bootloader into the EVM if there is no functional bootloader available.

### **1.1.4 Required Network Capabilities**

The EVM supports communications with the development PC running Platform Builder using either Ethernet or USB RNDIS. Platform Builder requires the device to be on the same subnet as the development PC. The bootloader and kernel support both manual and dynamic IP address assignment. DHCP is enabled by default; this can be changed using the bootloader menu.

## **1.2 Notation Conventions**

The following section describes terms used in this document.

### **1.2.1 CPU Name**

This document is written to support the OMAP35xx EVM family. The term OMAP3530 will be used to indicate one of the CPU names supported by the EVM family. When working with a particular CPU type, the term OMAP3530 should be understood as being the name of the CPU.

### **1.2.2 Other Terms**

Knowledge of Visual Studio, Platform Builder and Windows Embedded CE terms is assumed. The user should consult Platform Builder documentation for a definition of any unfamiliar terms.

## **1.3 Related Documentation**

This section describes other documentation that should be referenced when working with the EVM BSP.

### **1.3.1 Visual Studio 2005 / Platform Builder Documentation**

This documentation is the primary reference material for understanding the Windows Embedded CE 6.0 development system. Familiarity with this material is essential to successful Windows CE development using the EVM.

### **1.3.2 EVM Schematic**

Schematics for the EVM hardware are included as part of the EVM platform documentation.

# BSP Features

---



---

This chapter discusses features of the EVM BSP.

Topic	Section
Bootloader .....	2.1
NAND Flash Organization .....	2.2
OAL Features .....	2.3
Drivers.....	2.4
Unsupported Hardware Features .....	2.5

## 2.1 Bootloader (EBOOT)

### 2.1.1 Bootloader and Initialization Sequence

The EVM boots from Flash memory using the internal boot ROM of the OMAP3530 processor. The bootloader architecture consists of an initial bootstrap loader called the XLDR and a secondary loader called EBOOT. The internal boot ROM performs a minimum hardware setup, and then copies the XLDR from the first good block of Flash memory to a fixed location in internal SRAM. The boot ROM then jumps to the entry point of the XLDR.

The XLDR is a BSP specific bootstrap loader whose function is to do basic hardware initialization and copy the second stage, full featured EBOOT from Flash memory into RAM for execution. The XLDR is limited in size by the boot ROM specification and does not implement any features other than what are needed for bootstrap.

Note: the BSP only builds the XLDR in Release configurations. Debug configurations use un-optimized code libraries that cause the XLDR to exceed the allotted size.

The primary development bootloader is called EBOOT. The primary purpose of the EBOOT loader is to load the operating system image from a specified source and launch it. EBOOT also provides a serial based user interface that allows a developer to control various aspects of the loader behavior.

### 2.1.2 Low Level Initialization

Low level hardware initialization is performed in the XLDR as part of its bootstrap requirements. This includes multifunction pin configuration, clock setup, chip select timings, and DDR configuration. The file:

```
SRC\BOOT\XLDR\platform.c
```

contains the routines responsible for initialization. This file must be modified for every hardware platform. The \INC directory contains a number of header files with defines that control many of the timing and memory map related initialization parameters.

### 2.1.3 Device Configuration Parameters

The bootloader provides user accessible configuration parameters that control various aspects of bootloader behavior. These parameters are primarily used during development to control where the OS image comes from, network configuration etc. Changes to the configuration parameters can optionally be saved to nonvolatile storage (Flash memory) and preserved across power cycles. The configuration parameters can be accessed by the user via the serial user interface.

### 2.1.4 Serial User Interface

EBOOT supports a user interface over the serial port exposed on the main board at the RS-232 connector labeled UART3. The actual physical CPU UART corresponding to this connector is UART3 and is configured to run with the following configuration:

- 115200 baud
- 8 data bits
- 1 stop bit
- No parity

The UART3 connector is wired as a DTE device and requires a null modem cable to communicate with a terminal program on a Host PC.

#### 2.1.4.1 [1] Show Current Settings

The first menu option provides a summary of the current bootloader configuration parameters. These include the current bootloader and OS kernel debug transport, device name, bootloader/kernel network configuration, and Ethernet MAC address.

Note: the settings shown are current values and have not necessarily been saved to nonvolatile storage. If modified configuration parameters are not manually saved to flash the changes are lost on a reboot

#### 2.1.4.2 [2] Select Boot Device

This menu option configures the bootloader to load the OS image from the specified source. The options include downloading an OS image from Platform Builder over Ethernet (default), USB RNDIS, SD Card or loading the image from the OS partition on NAND Flash.

#### **2.1.4.3 [3] Select Debug Device**

This menu option configures the KITL transport that will be used in a KITL enabled OS image. The options include Ethernet (default) or USB RNDIS

#### **2.1.4.4 [4] Network Settings**

This menu option configures various network settings used by the bootloader and kernel transports. It also includes a mechanism to program the MAC address into the EEPROM associated with the SMSC LAN9115 network controller.

The default network configuration uses DHCP, and therefore requires a DHCP server to be present on the subnet. Note that this network configuration applies only to bootloader and OS kernel KITL network communications. The regular SMSC LAN9115 Ethernet NDIS miniport driver has a completely independent configuration.

#### **2.1.4.5 [5] Flash Management**

This menu option provides a mechanism to manage the Flash memory on the EVM. Flash blocks can be reserved, erased etc. These options are primarily available for diagnostic purposes and should not need to be used during development.

This menu also contains a mechanism to do a low level Flash format. The format operation partitions the non-reserved portions of Flash (area beyond the bootloader) into an OS partition and a file system partition. The size of the OS partition is fixed in a header file in the:

```
\SRC\INC directory
```

The bootloader will automatically reformat the Flash during initialization if it detects that the partition does not exist or does not match the configured size (a new bootloader flashed with a changed OS partition size parameter).

#### **2.1.4.6 [6] Set Device ID**

This menu option provides a mechanism to set the device ID. The device ID is used for various identification purposes. It serves as an input in creating the Universally Unique Identifier (UUID), creating a hardware entropy value etc.

#### **2.1.4.7 [7] Save Settings**

This menu option provides a mechanism to save the current settings to nonvolatile storage. The settings will be retained across power cycles.

Note: This option is not applicable when the system is booted from SD card.

## **2.2 NAND Flash Organization**

The NAND Flash used on the EVM is 2048 byte page, 64 pages per block (128KB block size), 1024 blocks (131072 KB). Note that some versions of EVM hardware contain OneNAND flash. These devices have slightly different Flash geometry but still have 128KB blocks with a total size of 131072KB.

The NAND Flash is divided into 3 regions: Reserved area (bootloader), OS partition, and file system partition. All regions start on NAND chip block boundaries.

### 2.2.1 Reserved Area (Bootloader/Configuration Parameters)

The reserved area is located at the beginning of the NAND Flash and contains the XLDR, EBOOT, and configuration parameters. The size of the reserved area is dictated by the size of EBOOT and the requirements imposed on the XLDR by the internal boot ROM.

The internal boot ROM will attempt to load the XLDR from block 0. If an error occurs, the boot ROM will attempt to load from the next block up to a maximum of 4 blocks. This mechanism requires the XLDR to be located in each of the first 4 NAND blocks. For ease of implementation the BSP places the XLDR in the first four **good** blocks which could result in a copy of the XLDR being located beyond the first 4 physical blocks if a bad block is encountered. Note that the XLDR itself requires much less than a full block but the first four complete blocks are still used due to boot ROM requirements.

EBOOT is located in the blocks immediately following the last XLDR copy. Currently the EBOOT implementation requires 2 good blocks. Note that entire blocks must be reserved even though EBOOT may only require a small portion of the last block.

The configuration parameters are stored in the last page of the last EBOOT block; they do not take up an entire block themselves. The rest of the block containing EBOOT is saved and restored when the configuration parameters are updated.

The total number of blocks in the reserved area is the sum of the 4 XLDR blocks and the 2 EBOOT blocks, plus any bad blocks that are encountered in this region. Bad blocks are skipped and the next good block selected in the Flash algorithms.

### 2.2.2 OS Partition

The OS partition follows the reserved area and contains the OS image. This partition is created using the BOOTPART APIs. The partition is marked as a PART\_BOOTSECTION indicating that it contains the operating system image. This partition type also prevents the Flash file system code from attempting to manage this area as part of its wear leveling algorithms.

The OS partition is created by the bootloader with a low level format during system initialization if it does not already exist. A low level format is also automatically performed if the bootloader detects an existing OS partition with a size different from what the bootloader expects. The OS partition size is set at compile time in the file:

```
\SRC\INC\image_cfg.h.
```

Note: A low level format destroys all data on the flash outside of the reserved area
--

The Master Boot Record (MBR) is also created during the low level format and defined to be located in the first sector past the reserved area. Therefore it is located at the beginning of the OS partition. The Master Boot Record contains information describing the layout of the NAND flash to the Flash file system driver.

### 2.2.3 File System Partition

The File System partition takes up the remainder of the NAND flash. It is also created during the low level format operation in the bootloader. The File System partition is marked as a FAT partition and can be automatically mounted by the operating system.

## 2.3 OAL Features

This section provides a brief overview of some of the significant OAL features included in the EVM BSP.

### 2.3.1 *Serial Debug Message*

The OAL uses the same serial port (CPU UART3) and configuration as the bootloader for serial debug messages. All debug messages are sent to the serial port if KITL is not enabled. When KITL is enabled some low level debug messages still appear on the serial port instead of over the KITL link.

The regular serial driver must not be allowed to conflict with the debug serial port. The serial driver must be disabled for CPU UART3 by setting the BSP\_NOCOM3 environment variable in the platform batch file. The serial debug messages can be disabled by calling the function OEMDeinitDebugSerial() in OEMInit (\SRC\OAL\OALLIB\init.c) and ensuring debug messages aren't enabled in BSPPowerOn() (\SRC\OAL\OALLIB\power.c). This will allow the regular serial driver to use this port.

### 2.3.2 *KITL*

The EVM BSP supports debug connections to Platform Builder over Ethernet or USB RNDIS using the Kernel Independent Transport Layer (KITL). The KITL link allows Platform Builder to communicate with the device providing debug message services, kernel debugger support, target control, Release directory file system, and Remote Tools support. The desired KITL transport is selected via the serial bootloader menu.

The KITL link will be enabled when the KITL OS component is included in the image. This can be done in either debug or retail configuration by ensuring the IMGNOKITL environment variable is not set. It is important to note that if KITL is included in the image and active it requires a connection to Platform Builder running on the Host PC.

The network section of the bootloader serial menu provides an option to disable the KITL link even if the KITL component is included in the OS image. This allows the same image to support both KITL-enabled and regular builds. The user can decide at run time whether to activate the KITL connection or not. The KITL option is enabled by default so any image that includes KITL will attempt to connect to Platform Builder across the KITL link.

The KITL transports (Ethernet and USB RNDIS) are built directly into the OAL and are dedicated for use by the kernel. These transports do not use the regular NDIS miniport driver (in the case of Ethernet) or the USB Function controller driver (in the case of USB RNDIS). The regular driver must not be allowed to run simultaneously with the KITL transport or they will both contend for the same hardware resources. These drivers are automatically disabled when the KITL transport is used.

### 2.3.3 *VMINI*

The SMSC LAN9115 NDIS miniport driver is not permitted to load if the Ethernet KITL transport is in use so the Ethernet port would not normally be available for application use. The Virtual Miniport (VMINI) driver provides a mechanism for applications to leverage the Ethernet KITL link by exposing an NDIS miniport driver interface to the KITL transport. The EVM BSP supports the VMINI implementation for both the Ethernet KITL transport and the USB RNDIS KITL transport.

VMINI requires an active KITL link meaning there must be a connection to Platform Builder running on the host PC. However the VMINI architecture is otherwise transparent to the network stack and applications. It exists simply as a network miniport driver.

The network portion of the bootloader serial user interface provides a mechanism for the user to control VMINI. VMINI is enabled by default and will be available when KITL is active, but can be disabled if desired.

## 2.4 Drivers

This section provides a brief overview of some of the drivers included in the EVM BSP.

### 2.4.1 GPIO Driver

The GPIO driver exposes the GPIO pins that are available on the CPU and TWL4030 companion chip. The GPIO driver provides a consistent interface to access these IO pins from other drivers. It also provides protection against concurrent operation.

The GPIO driver identifies all available GPIO pins with a unique number. The GPIO numbering system is as follows:

- ❑ 0-191 CPU GPIO pins
- ❑ 192-209 TWL4030 pins

### 2.4.2 Display Driver

The display driver supports the VGA display on the EVM. The display driver supports DirectDraw, and can support hardware accelerated rotation using the Virtual Rotated Frame Buffer (VRFB) hardware. However, the VRFB implementation is not compatible with the software Codecs provided by Microsoft for decoding video media. The VRFB hardware acceleration is disabled by default in the registry.

### 2.4.3 Backlight Driver

The backlight driver supports the backlight module on the EVM. This module leverages a PWM output on the TWL4030 companion chip. The backlight driver is integrated with the system Power Manager and supports multiple power levels based on system activity.

### 2.4.4 I2C Bus Driver

The I2C Bus driver supports the I2C controllers on the OMAP35xx processor. The I2C bus is used by the TWL4030 as a control channel. The I2C bus is also made available on the expansion bus. The I2C interface API is defined in the file:

```
\SRC\CSP\INC\i2c.h
```

### 2.4.5 SPI Driver

The SPI bus driver supports the McSPI controllers in the OMAP3530 processor. The SPI bus is currently used by the touch controller. It is also made available on the expansion header. The SPI interface API is defined in the file:

```
\SRC\CSP\INC\spi.h
```

### **2.4.6 FLASH Media Driver**

The Flash Media driver manages the portion of the NAND Flash device that is not reserved. It exposes the File System partition to the operating system as a block device.

### **2.4.7 SDIO Host Controller Interface Driver**

The SDIO Host Controller Interface Driver interfaces the SDIO stack to the SDIO host controllers in the OMAP3530. There are two controllers in the CPU. The first controller is dedicated for use on Slot 1; this slot supports only 3.3V SDIO cards. The second controller is exposed on the expansion header.

### **2.4.8 Serial Driver**

The serial driver supports the UARTs on the OMAP3530 CPU. UART1 and UART2 are multiplexed together at the UART1/2 connector on the main board. These UARTs are also exposed on the expansion connector. UART3 is typically used as a debug port and is exposed at the UART3 connector on the EVM.

### **2.4.9 TSC2046 Touch Driver**

The touch driver supports the Texas Instruments TSC2046 touch controller used on the EVM. The touch controller is located on the SPI bus at channel 0.

### **2.4.10 USBOTG Driver**

The USBOTG driver supports the High Speed USB2.0 Controller on the EVM. This driver supports OTG functionality, allowing the port to be either an A device or a B device depending on how it is connected to the opposing device. This driver integrates into the USB stack provided by Windows Embedded CE 6.0 . The actual functionality of this port will depend on the client drivers and registry entries included in the operating system image.

### **2.4.11 TWL4030 Wave Driver**

The TWL4030 wave driver is an audio driver for the codec contained in the TWL4030 companion chip. The driver supports both playback and record operations.

### **2.4.12 Keyboard**

The keyboard driver supports the EVM keypad switches using the matrix keypad functionality in the TWL4030. The VK\_... code mapping is shown in Table 1.

Table 1. *VK\_... code mapping*

---

<b>Key</b>	<b>Mapping</b>	<b>Key</b>	<b>Mapping</b>	<b>Key</b>	<b>Mapping</b>	<b>Key</b>	<b>Mapping</b>
S4	VK_ESCAPE	S8	N/A	S12	VK_SPACE	S16	VK_TBACK
S5	VK_TAB	S9	N/A	S13	VK_TUP	S17	VK_TRIGHT
S6	VK_THOME	S10	VK_SOFT2	S14	N/A	S18	VK_TACTION
S7	VK_TSOFT1	S11	VK_TLEFT	S15	VK_TDOWN		

---

## 2.5 Unsupported Hardware Features

The BSP does not support some of the hardware features included in the design of the EVM. The following hardware features are not supported:

- Battery gas gauge / charging
- Expansion header
- QVGA display mode

# BSP Installation

---



---



---

This section provides basic instructions for installation of the EVM BSP.

Topic	Section
Install Required Software .....	3.1
Installing the EVM BSP .....	3.2

## 3.1 Install Required Software

Other software must be installed prior to installing the EVM BSP. See section “**Required Software**” in this document for more information.

## 3.2 Installing the EVM BSP

### 3.2.1 *Version 6.11 or earlier*

The EVM BSP is delivered in the form of a zip file. The .zip file should be extracted into the WINCE600 platform tree at

WINCE600\PLATFORM\TI\_EVM\_3530

### 3.2.2 *Version 6.12 or later*

In the distribution there are two zip archives under the folder BSP that contains a Microsoft Windows CE 6.00 BSP for the Texas Instruments EVM3530 development board.

The TI\_EVM\_3530\_x.xx.xx.zip archive should be unzipped under  
 WINCE600\PLATFORM  
 to create a folder containing the platform dependent BSP files called TI\_EVM\_3530.

The OMAP35XX\_TPS659XX\_TI\_V1\_x.xx.xx.zip archive should be unzipped under  
 WINCE600\PLATFORM\COMMON\SRC\SOC  
 to create a folder containing the platform independent BSP files called  
 OMAP35XX\_TPS659XX\_TI\_V1.



# Building Images

---



---



---

This section provides a brief overview of the steps needed to use Platform Builder to create an OS Design for the EVM BSP. Working knowledge of the Windows CE development environment and procedures is assumed.

Topic	Section
Create an OS Design Using Platform Builder .....	4.1
Building the BSP .....	4.2

## 4.1 Create an OS Design Using Platform Builder

### 4.1.1 Using the Sample OS Design

The EVM BSP includes a set of sample OS Design solutions. This may be used as a starting point for creating your OS Design. Three are included in the BSP release 6.13 or later:

1. EVM\_3503 for TMDXEVM3503 (EVM1)
2. EVM\_3503\_MDC for TMDXEVM3503 and Multimedia Daughter Card (EVM1+MDC)
3. EVM\_3530 for TMDSEVM3530 (EVM2)

Follow these steps to use the sample OS Design.

- 1) Locate the Sample OS Design folders in the SampleOSDesign directory of the installer.
- 2) Copy the selected sample OS Design folder to %\_WINCEROOT%\OSDesigns directory.
- 3) Three new sample build scripts with default build settings for the supported platforms are included in this release. They are located in %\_WINCEROOT%\platform\ti\_evm\_3530 :
  - evm1\_ti\_evm\_3530.bat for TMDXEVM3503
  - evm1\_mdc\_ti\_evm\_3530.bat for TMDXEVM3503 + TI Multimedia Daughter Card
  - evm2\_ti\_evm\_3530.bat for TMDSEVM3530

Rename the appropriate script to ti\_evm\_3530.bat for building the configuration chosen.

- 4) Open the solution file in Visual Studio 2005 by navigating to the sample OS Design folder and selecting the solution file.

### 4.1.2 Creating a New OS Design

The Platform Builder New Platform Wizard can be used to create an OS Design for use with the EVM BSP.

- 1) Start Visual Studio 2005.
- 2) Select **New ->Project** from the **File** menu.
- 3) Select the **Platform Builder for CE 6.0** project type.
- 4) Select a name for the new project. This name does not have to have any relationship to the BSP name.
- 5) Select OK to launch the **Windows Embedded CE 6.0 OS Design Wizard**
- 6) Select the **TI\_EVM\_3530** BSP from the list of available BSPs.

Note that this BSP will only appear if it has been installed and the support for ARM was installed as a part of the Windows Embedded CE 6.0 installation
--

- 7) Continue the Platform Builder New Platform Wizard until complete.

For more information about using Platform Builder to create or modify an OS Design, consult the Platform Builder documentation.

## 4.2 Building the BSP

### 4.2.1 Procedure

Start Microsoft Visual Studio using the desired OS Design (the one that includes the EVM BSP).

Use the **Build** menu, **Global Build Settings** submenu and enable the following:

- Copy Files to Release Directory After Build
- Make Run-Time Image After build

Use the **Project** menu, **<OS Configuration Name> Properties...** command to examine the Build Options page:

- If the Enable Kernel Debugger or Enable KITL boxes are checked then the newly created OS image will only boot if the EVM is connected to Platform Builder using a KITL supported hardware connection.
- If the Write Run-time Image to Flash Memory box is not checked then the newly created OS image will be loaded into RAM for execution and the NAND flash will not be updated.

Use the **Build** menu, **Build Solution** command to build the BSP. This operation will do a complete build including sysgen of the OS components. This operation can also be performed using the context menu in the Solution Explorer.

---

After the first successful complete build you can speed up your development cycle by limiting the build to only components that have changed. One way to do this is to navigate to your BSP in the Solution Explorer and do Targeted Builds using the context menu.

Note that any time the OS design has changed you will need to do a full rebuild that includes the sysgen step.

#### 4.2.2 Checking For Excessive Image Size

The allowable size of the OS image depends on the configuration parameters in the file \FILES\config.bib. The final image must fit within the space allocated to the RAMIMAGE region. The space allocated to the RAM region can also be used if the AUTOSIZE option is enabled. If the size of the OS image exceeds the allowable size the image will not boot reliably.

Examine the output of the build log to for warnings that indicate that records were skipped when creating the .bin image. This indicates that the image did not fit in the space allocated. Unfortunately this does not generate a hard error and is easy to overlook.

A second restriction on the OS image is the size of the OS partition on the NAND flash chip. If you build an image that is targeted to flash it must be able to fit in the space allocated.

Note that this process applies to all image output files, including the XLDR and EBOOT. These bootloader components can't make use of the AUTOSIZE option and therefore must be able to fit entirely within the space defined for the RAMIMAGE region in their respective .BIB files.

### 4.3 Output File

The BSP build process produces several output files including the OS image and several bootloader files.

#### 4.3.1 NK.BIN

The operating system image is located in a single file called NK.BIN. NK.BIN is structured in a proprietary Microsoft file format containing data records. This file cannot be executed directly by the CPU nor can it generally be programmed directly into flash. However the bootloader is designed to understand this format and it parses the data and copies it to the correct destination in memory. NK.BIN is the file that Platform Builder will download to the bootloader running on the EVM.

The bootloader will copy the OS image to its destination in RAM and then jump to the image entry point. At that point control has passed from the bootloader to the initialization routines in the operating system image.

The build process will optionally create a second operating system image file called NK.NB0. This file will only be created if the correct options are defined in the config.bib configuration file. NK.NB0 is an exact copy of the operating system image as it should exist in memory. The NK.NB0 file is normally used with JTAG equipment or other tools that directly access Flash or RAM.

For more detail descriptions on the Windows CE system images refers to official Microsoft Windows CE documentations.

#### 4.3.2 **XLDRONENAND.BIN/XLDRNAND.BIN**

The initial bootstrap loader is located in a single file called XLDRNAND.BIN. This file also uses the proprietary Microsoft .BIN file format. The bootloader understands this file format, and also understands the unique XLDR storage requirements in NAND Flash.

Platform Builder can optionally be configured to download XLDRNAND.BIN to the EVM. When the bootloader detects this file it properly programs it to the first four good blocks of NAND Flash.

The build process also creates the XLDRNAND.NB0 file. This file is the exact representation of the XLDR as it must appear in memory. The build process uses this file as part of the process of creating a RAW image file that can be used by the EVM Flash Tool to program the entire bootloader region outside of bootloader control.

EVM platforms that contain OneNAND memory should use the OneNAND version of the XLDR. EVM platforms that contain Micron NAND Flash should use the NAND version of XLDR.

#### 4.3.3 **EBOOTONENAND.BIN/EBOOTNAND.BIN**

The primary bootloader is located in a single file called EBOOTNAND.BIN. This file also uses the proprietary Microsoft .BIN file format.

Platform Builder can optionally be configured to download EBOOTNAND.BIN to the EVM. When the bootloader detects this file it properly programs it to after the XLDR region in the reserved portion of NAND Flash.

The build process also creates the EBOOTNAND.NB0 file. This file is the exact representation of EBOOT as it must appear in memory. The build process uses this file as part of the process of creating a RAW image file that can be used by the EVM Flash Tool to program the entire bootloader region outside of bootloader control.

EVM platforms that contain OneNAND memory should use the OneNAND version of the EBOOT. EVM platforms that contain Micron NAND Flash should use the NAND version of EBOOT.

#### 4.3.4 **TIEVM3530-ONENAND.RAW/TIEVM3530-NAND.RAW**

The XLDR and EBOOT are combined into a single file in a format that is suitable for programming directly into NAND or OneNAND flash memory. This .RAW file can be used with EVM Flash Tool to program a functional bootloader into flash memory.

#### 4.3.5 **Locating the Image Files**

After a successful build, the image output files will be located in the flat release directory. This directory is typically a subdirectory of the OS design directory in the top level OSDESIGNS folder. The build process copies all the files that will eventually make up the final image output files to this directory. It then uses this directory as a staging area to create the final bootloader and OS images.

The Platform Builder **Build** menu, **Open Build Window** command can be used to open a command window with the current directory set to the flat release directory.

# Loading Images

---



---

The bootloader included in the EVM3530 BSP is capable of downloading and using new bootloader and OS images. This section describes the most commonly used image download methods. A working knowledge of the Windows CE development environment and procedures is assumed

Topic	Section
Error! Not a valid result for table. ....	5.1
<b>Downloading Images using Platform Builder</b> Error! Reference source not found. ....	<b>5.2</b>

## 5.1 Programming Bootloader Using Boot ROM

EVM Flash Tool can be used to program the bootloader into flash over the serial or USB ports. This tool leverages functionality provided by the internal boot ROM and is typically used when there is no functional bootloader in flash.

## 5.2 Downloading Images using Platform Builder

The bootloader is able to communicate with Platform Builder to download updated images. This includes the operating system image as well as updated versions of the XLDR and EBOOT.

The bootloader programs XLDR and EBOOT images into their correct location in the reserved portion of NAND flash. It automatically programs the OS image into the OS partition or downloads it directly to RAM depending on how the image was built.

### 5.2.1 EVM Setup

The bootloader must be configured to load an image from the Ethernet or USB RNDIS. Ethernet is the default setting if the configuration parameters have not been changed.

#### 5.2.1.1 Download Images using Ethernet

- Select option [2] on the serial boot menu to configure the boot device.
- Select option [1] LAN9115 MAC.
- Optionally save the settings.
- Connect the EVM3530 Ethernet to the same subnet as the Host PC.

### 5.2.2 Setup Platform Builder

Platform Builder must be configured to download the OS image or bootloader file.

- 1) Start Visual Studio and select the Platform Builder project containing the EVM BSP.
- 2) Select the desired configuration (Release/Debug) from the Solution Configuration pull down box.
- 3) Select **Properties** from the **Project** menu to bring up the project properties dialog box.
- 4) Expand the Configuration tree and select the General node.
- 5) Select the desired .BIN file from the drop down dialog box. The available options will be the .BIN files that exist in the flat release directory corresponding to the OS design.

The default setting will be NK.BIN, allowing the OS image to be downloaded and optionally debugged. However, this mechanism allows the XLDR or EBOOT images to also be downloaded if necessary.

### 5.2.3 Configure Platform Builder Connectivity Options

Platform Builder must be configured to communicate with the EVM over the Ethernet connection (if using either Ethernet or USB RNDIS Fn). Platform Builder can communicate with any device on the same subnet as long as that device uses the EBOOT protocol.

- 1) Use the **Target** menu, **Connectivity Options** entry, to enter the **Target Device Connectivity Options**.
- 2) Select **Kernel Service Map** to bring up the kernel services settings.
- 3) Set **Target Device** to CE Device.
- 4) Set **Download** to Ethernet.
- 5) Set **Transport** to Ethernet. This setting will be used whenever a KITL enabled image is run on the EVM.
- 6) Set **Debugger** to KdStub. This setting will be enable the kernel debugger component when the debugger is included in the image running on the EVM.
- 7) Click the **Settings** button next to the Download box to bring up the **Ethernet Download Settings** dialog box.
- 8) Start the download operation on the EVM by selecting Option [0] (Exit and Continue). The EVM will obtain an IP address via DHCP if necessary, then start broadcasting BOOTME packets to the network.
- 9) Select the ID string corresponding to the EVM that will appear in the Ethernet Download Settings dialog box. It will be of the form EVM3530-xxxxx, where the last 5 digits are derived from the MAC address. Close the dialog box.

Note that the name under both the Download and Transport drop down boxes contain the same name that was just selected. Platform Builder will attempt to communicate with the IP address associated with this name on all subsequent connection attempts. Close the Connectivity Options dialog.

### 5.2.4 Download Image

Once Platform Builder has been configured to communicate with the EVM the image can be downloaded. The download operation is triggered from the EVM by broadcasting the BOOTME packet. Platform Builder listens for these packets and responds when it detects the request.

- 1) Ensure the EVM is still broadcasting BOOTME packets. Once an IP address has been obtained, it will broadcast the packets for approximately two minutes. The device will need to be reset and the download operation reinitiated if this time period has been exceeded.
- 2) Select **Attach Device** from the **Target** menu. The Download dialog box will appear, and the download will proceed as soon as a BOOTME packet is detected.



# Booting the EVM

There are a number of sources that the EVM can be set to automatically boot from once they are programmed using the EVM Flash Tool. This chapter describes those sources.

Topic	Section
Booting from OneNAND Flash.....	6.1
Booting from NAND Flash.....	6.2

## 6.1 Booting from OneNAND Flash

To select OneNAND Flash as the boot device, change SW4 to one of the settings in Table 2.

*Table 2. SW4 Switch Positions, OneNAND Flash Boot*

SW 4 Switch Position							
1	2	3	4	5	6	7	8
ON	ON.	ON	ON	ON	ON	OFF	OFF
ON	OFF	ON	ON	ON	ON	OFF	OFF
ON	ON	OFF	ON	ON	ON	OFF	OFF
ON	ON	ON	ON	OFF	ON	OFF	OFF
ON	OFF	OFF	ON	OFF	ON	OFF	OFF

For serial cable EVM Flash Tool programming (refer Chapter 8 for details), select UART as the first boot device and OneNAND Flash as the second boot device and change SW4 to Table 3.

*Table 3. SW4 Positions, Serial Cable EVM Flash Tool. OneNAND Boot*

SW 4 Switch Position							
1	2	3	4	5	6	7	8
ON	OFF.	OFF	ON	OFF	OFF	OFF	OFF

For USB cable EVM Flash Tool programming (refer Chapter 8 for details), select USB as the first boot device and OneNAND Flash as the second boot device and change SW4 to Table 4.

*Table 4. SW4 Positions, USB Cable EVM Flash Tool. OneNAND Boot*

SW 4 Switch Position							
1	2	3	4	5	6	7	8
ON	ON.	OFF	ON	ON	OFF	OFF	OFF

## 6.2 Booting from NAND Flash

This section is applicable to both TMDXEVM3503 (EVM1) with NAND flash as well as TMDSEVM3530 (EVM2) platforms. To select NAND Flash as the boot device, change SW4 to one of the settings in Table 5.

*Table 5. SW4 Switch Positions, NAND Flash Boot*

SW 4 Switch Position							
1	2	3	4	5	6	7	8
OFF	ON.	ON	ON	ON	ON	OFF	OFF
ON	ON	ON	OFF	ON	ON	OFF	OFF
OFF	OFF	OFF	OFF	ON	ON	OFF	OFF
OFF	ON	OFF	ON	OFF	ON	OFF	OFF
OFF	OFF	ON	OFF	OFF	ON	OFF	OFF

For serial cable EVM Flash Tool programming (refer Chapter 8 for details), select UART as the first boot device and NAND Flash as the second boot device and change SW4 to Table 6.

*Table 6. SW4 Positions, Serial Cable EVM Flash Tool. NAND Boot*

SW 4 Switch Position							
1	2	3	4	5	6	7	8
OFF	ON.	OFF	ON	OFF	OFF	OFF	OFF

For USB cable EVM Flash Tool programming (refer Chapter 8 for details), select USB as the first boot device and NAND Flash as the second boot device and change SW4 to Table 7.

*Table 7. SW4 Positions, USB Cable EVM Flash Tool. NAND Boot*

SW 4 Switch Position							
1	2	3	4	5	6	7	8
OFF	OFF.	ON	OFF	OFF	OFF	OFF	OFF

## 6.3 Booting from SD Memory Card

The EVM is capable of booting the BSP from a correctly formatted, programmed and inserted SD memory card. The following steps must be followed exactly for a successful boot:

### a) Preparing the SD Card to be bootable on the EVM

#### Option 1 (Recommended): Use SD card boot utility

Install the TI\_SDCard\_boot\_utility\_v1\_0.exe that can be found in Tools\SDCard\_boot\_utility folder in the BSP distribution.

- Step 1. Run the installed utility
- Step 2. Select the SD card drive
- Step 3. Browse and Select the MLO file
- Step 4. Browse and Select the EBOOTSD.NB0 and NK.BIN files
- Step 5. Click Proceed.
  - Click "Start" to format the SD card
  - Click "OK" for Format Warning
  - Click "OK" once "Format Complete" window pops up
  - Click "Close" to close the format window
  - Click "Quit" once the files are copied

#### Option 2: Manual steps

- Step 1. Format the SD Card from WinXP Explorer
- Step 2. Make the SD Card bootable using Roadkil's sector editor application (or other sector editor utility)
  - 1) Open the SD Card physical sector 0
  - 2) Write 0x80 at offset 0x1BE (1st partition entry)
  - 3) Save sector
  - 4) Exit sector editor application
- Step 3. Copy ONLY the MLO file to the SD card
- Step 4. Copy both the EBOOTSD.NB0 and the NK.BIN files to the SD card

### b) Configure the EVM SW4 switch to select MMC1 as the first boot device, use one of the SW4 switch settings in Table 8.

Table 8. SW4 Switch Positions, SD Card Boot

SW 4 Switch Position							
1	2	3	4	5	6	7	8
ON	OFF.	OFF	ON	ON	ON	OFF	OFF
ON	OFF	ON	ON	OFF	ON	OFF	OFF
ON	ON	ON	OFF	OFF	ON	OFF	OFF

Alternately, when using EVMFlash tool with serial cable interface, configure the EVM SW4 switch to select UART as the first boot device and MMC1 as the second boot device as in Table 9.

*Table 9. SW4 Positions, Serial Cable EVM Flash Tool. SD Card Boot*

<b>SW 4 Switch Position</b>							
<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>
ON	ON.	ON	OFF	OFF	OFF	OFF	OFF

Alternately, when using EVMFlash tool with USB cable interface, configure the EVM SW4 switch to select USB as the first boot device and MMC1 as the second boot device as in Table 10.

*Table 10. SW4 Positions, USB Cable EVM Flash Tool. SD Card Boot*

<b>SW 4 Switch Position</b>							
<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>
ON	ON.	ON	OFF	OFF	OFF	OFF	OFF

- c) Plug SD Card into the EVM.
- d) Power on the EVM.

The CPU internal ROM will load the SD XLDR (MLO), the SD XLDR will load SD EBOOT (ebootsd.nb0). After the SD EBOOT bootloader is running, use the menu to select the SD card as the boot device then select continue to boot the NK.BIN file. Note that the MMC/SD boot takes approximately two seconds per MB of NK.BIN file size.

# Debugging

---



---

This section discusses the options available to debug EVM bootloader and OS images. A working knowledge of the Windows CE development environment and procedures is assumed.

Topic	Section
Debugging the Bootloader .....	7.1
Debugging the Operating System .....	7.2

## 7.1 Debugging the Bootloader

The bootloader does not contain a software debugger component that allows it to communicate with the kernel debugger in Platform Builder. As a result it is not possible to use breakpoints, step through code etc. The only supported debug method is the serial debug port.

Platform Builder provides a debugger plug-in interface called ExDI2 that allows JTAG vendors to leverage the kernel debugger features. A third party tool that supports the OMAP3530 processor and the ExDI2 interface would provide the option to do bootloader source level debugging using the JTAG interface.

## 7.2 Debugging the Operating System

Platform Builder supports source level debugging with a debugger component in the kernel. The debugger must be included in the operating system image and the image must be configured to support a KITL connection to make use of this feature.

An OS design typically has two configurations that are created by default when the project is created. The Debug configuration disables compiler optimizations for ease of debugging, and includes the kernel debugger and KITL components. A Release configuration enables compiler optimizations and generally does not contain the kernel debugger. These options can be changed in the Configuration Manager.

It is possible to include the kernel debugger and KITL in both Release and Debug configurations. These two items are kernel DLLs that are included when the image is built and have no dependencies on the Release and Debug configurations. It is sometimes useful to include the debugger and KITL in a Release configuration along with debug versions of selected modules to enable a debugging environment that is closer to a normal production environment.

### **7.2.1 Image Configuration**

The Kernel Independent Transport Layer driver is added to the OS image by clearing the IMGNOKITL environment variable. This is usually done in the Build Options node in the project Properties dialog. Select the desired configuration (Release or Debug) and ensure that the Enable KITL check box is set appropriately.

The kernel debugger is included in the image in a similar fashion. This component is controlled with the IMGNODEBUGGER environment variable. This variable is exposed in the Properties dialog with a check box as well.

It is important to note that an image with KITL included and enabled requires a connection to Platform Builder in order for it to boot. The operating system will hang waiting for the connection to complete if the KITL link is active. Do not include KITL in an image unless a connection to Platform Builder is available or KITL is disabled via the serial bootloader menu.

### **7.2.2 Ethernet KITL**

The Ethernet port is normally used as the KITL transport as it provides the fastest speeds. The Ethernet port must be connected to a hub that is on the same subnet as the Host PC running Platform Builder. The Ethernet port must be selected as the Debug device using the serial bootloader menu.

The Ethernet KITL transport uses a dedicated set of routines to communicate with the kernel. It does not use the Ethernet NDIS miniport driver. It is critical that the Ethernet miniport driver be disabled when Ethernet KITL is in use. Failure to do so will cause the device to hang because the Ethernet driver will interfere with the KITL communications. The Ethernet NDIS miniport driver can be disabled by setting the BSP\_NOETHERNET environment variable. The VMINI component can be used to expose the Ethernet KITL link to applications.

### **7.2.3 USB RNDIS KITL**

The USB OTG port can also be used as the KITL transport. This port is normally used when an Ethernet port is not available. It is more difficult to set up because it requires a USB driver to be installed on the Host PC and it requires either a network bridge or Internet Connection Sharing to be enabled. USB RNDIS KITL is typically not as stable as Ethernet KITL because of the increased complexity of the interface.

The USB RNDIS KITL transport uses a dedicated set of routines to communicate with the kernel. It does not use the standard USB function or OTG stacks. It is critical that all USB drivers be disabled when USB RNDIS KITL is in use. The USB RNDIS KITL implementation automatically disables the standard USB drivers when KITL is active.

### **7.2.4 Serial Debug Messages**

The serial debug port at CPU UART3 is exposed on the RS-232 connector labeled UART3 on the main board. This port will print out all debug messages until the KITL transport is loaded. After KITL is initialized only low level messages that bypass the KITL interface will be written to this port.

The debug port can be disabled and made available for normal application use if desired. It is important to note that the UART3 serial driver should not be loaded if this port is to be used as a debug port. The UART3 serial driver can be disabled by setting the environment variable BSP\_NOCOM3.

It is sometimes beneficial to use serial debug messages instead of the KITL link. This option can print debug messages to the serial port much faster than they will appear over an Ethernet/USB RNIDS KITL link because of the overhead of the KITL protocol and Platform Builder synchronization. To use this feature simply remove KITL and the kernel debugger from the OS image. All messages will then be monitored with a terminal program.

Note that these messages will not appear in the Platform Builder Debug window because Platform Builder only shows the messages that come over a KITL link.



# EVMFlash Application

---



---

This chapter is a guide for working with the Texas Instruments EVM3530 platform and EVMFlash flashing utility. Hardware setup and EVMFlash how-to information are provided.

Topic	Section
Installing EVMFlash .....	8.1
EVMFlash Features Supported .....	8.2
Getting familiar with the EVM board .....	8.3
Bootloader Flashing using EVMFlash.....	8.4

## 8.1 Installing EVMFlash

### 8.1.1 *If the target EVM board is TMDXEVM3503 (EVM1) populated with OneNAND Flash:*

- Install .\EVMFlashTool\_vSamsungOneNAND\evmflash3530\_v1\_1\_bin.exe
- Run C:\Program Files\Texas Instruments\EVMFlash3530\_v1.1\EVMFlash.exe

### 8.1.2 *If the target EVM board is TMDXEVM3503 (EVM1) populated with NAND Flash:*

- Install .\EVMFlashTool\_vMicronNAND\evmflash3530\_v1\_2\_1\_bin.exe
- Run C:\Program Files\Texas Instruments\EVMFlash3530\_v1.2\EVMFlash.exe

### 8.1.3 *If the target EVM board is TMDSEVM3530 (EVM2) populated with NAND Flash:*

- Install .\EVMFlashTool\_vEVM2MicronNAND\evmflash3530\_v2\_0\_bin.exe
- Run C:\Program Files\Texas Instruments\EVMFlash3530\_v2.0\EVMFlash.exe

## 8.2 EVMFlash Features Supported

### 8.2.1 *Flashing the BSP bootloader over UART*

OneNAND and NAND download, read and write operations are supported.

Note: EVMFlash Diagnostics and Signing are NOT supported.

### 8.2.2 *Flashing the BSP bootloader over USB*

OneNAND and NAND download, read and write operations are supported.

Note: EVMFlash Diagnostics and Signing are NOT supported.

### 8.3 Getting familiar with a EVM board

Below is a reference to the TMDXEVM3503 EVM board. The EVM bottom view is shown in Figure 1 and the EVM top view is shown in Figure 2.

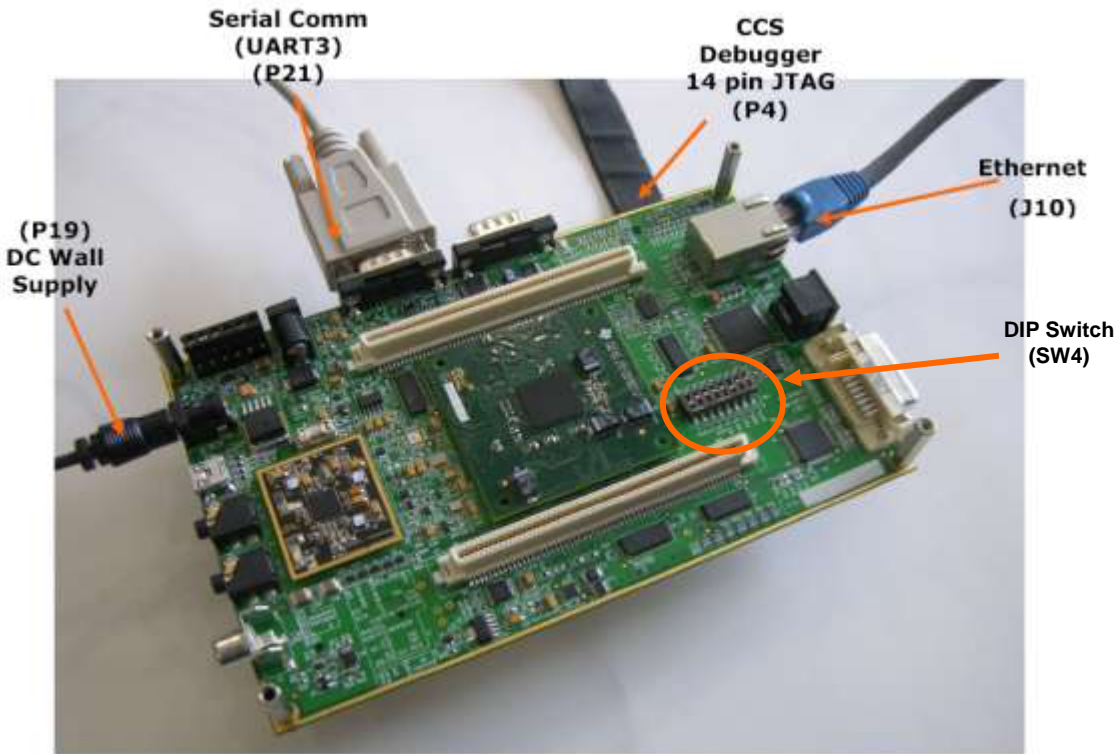


Figure 1. EVM Bottom View

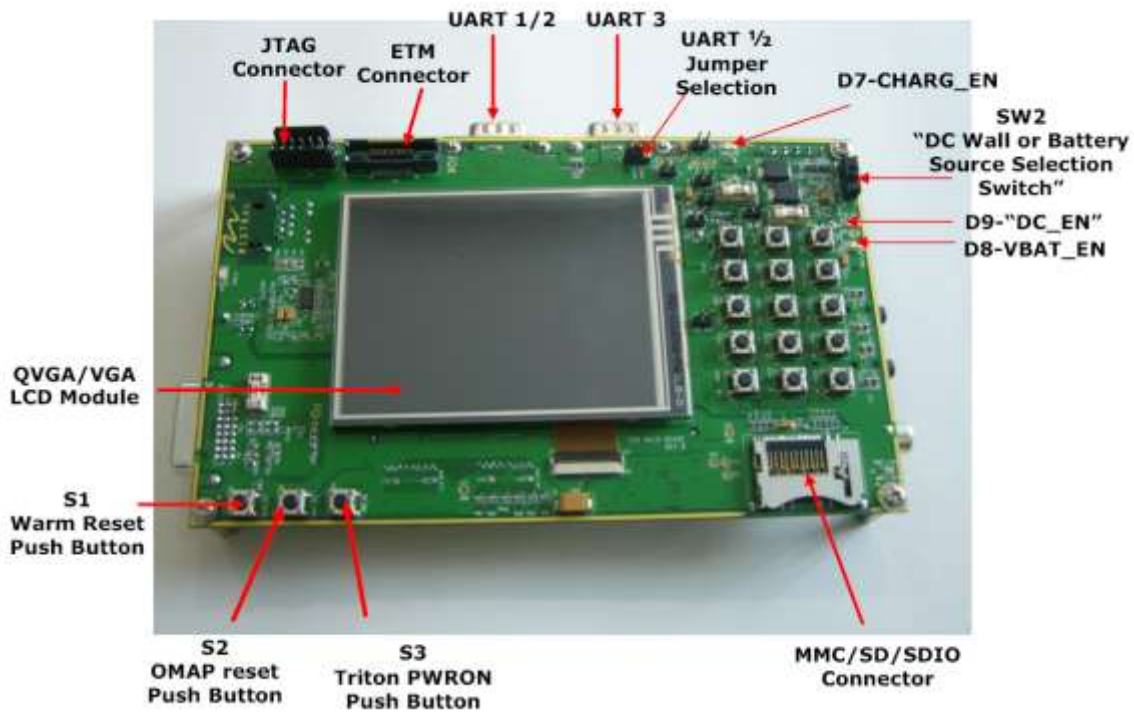


Figure 2. EVM Top View

## 8.4 Bootloader Flashing using EVMFlash

The following procedure is applicable to all versions of the EVM and any difference is brought out where necessary. Please note the physical location of the switches mentioned in the steps below may be different depending on the EVM version in use. Please refer the EVM setup guide for exact location of the switches. The procedure also gives a choice between UART (recommended) and USB download.

To load the bootloader into EVM FLASH:

**Step 1. If a previous EBOOT is installed, remove it. This can be done from the EBOOT menu:**

- Select [5] Flash Management
- Select [4] Show Flash geometry
- Select [4] Erase block range and enter first reserved block id and enter last reserved block id shown in the previous menu command output.
- Power off the EVM

**Step 2. Set the DIP switch SW4 on the EVM to one of the following:**

a) UART download (use the correct one for your EVM):

---

**SW 4 Switch Position (EVM with OneNAND)**

1	2	3	4	5	6	7	8
ON	OFF.	OFF	ON	OFF	OFF	OFF	OFF

---

**SW 4 Switch Position (EVM with NAND)**

1	2	3	4	5	6	7	8
OFF	ON.	OFF	ON	OFF	OFF	OFF	OFF

---

b) USB download (use the correct one for your EVM):

---

**SW 4 Switch Position (EVM with OneNAND)**

1	2	3	4	5	6	7	8
ON	ON.	OFF	ON	ON	OFF	OFF	OFF

---

**SW 4 Switch Position (EVM with NAND)**

1	2	3	4	5	6	7	8
OFF	OFF.	ON	OFF	OFF	OFF	OFF	OFF

---

**Step 3. Install EVMFlash on the Host PC and run.**

**Note:** Different version of EVMFlash Utility exist for TMDXEVM3503 (EVM1) with OneNAND, TMDXEVM3503 (EVM1) with NAND and TMDSEVM3530 (EVM2) with NAND. Ensure that you install and execute the correct version for your EVM.

**Note:** If using UART download, ensure that COM1 on the Host PC is not in use by any other application.

**Step 4. If not already in the target list, select the EVM3530 target and press Add Target**

**Step 5. Select the Link Type:**

- a) UART: UART, Port: 1, Baud Rate: 115200 and Mode: BOOT ROM
- b) USB: USB, Mode: BOOT ROM

**Step 6. Select the Download tree item. On the Download tab, select Device Type NAND and press the Add File button.**

**Step 7. Select the bootloader file from the release directory. Set the default address location:**

- OneNAND: TIEVM3530-onenand.raw, address – 0x20000000
- NAND: TIEVM3530-nand.raw, address – 0x20000000

**Step 8. Select the Erase Entire Flash checkbox option to clear the Flash**

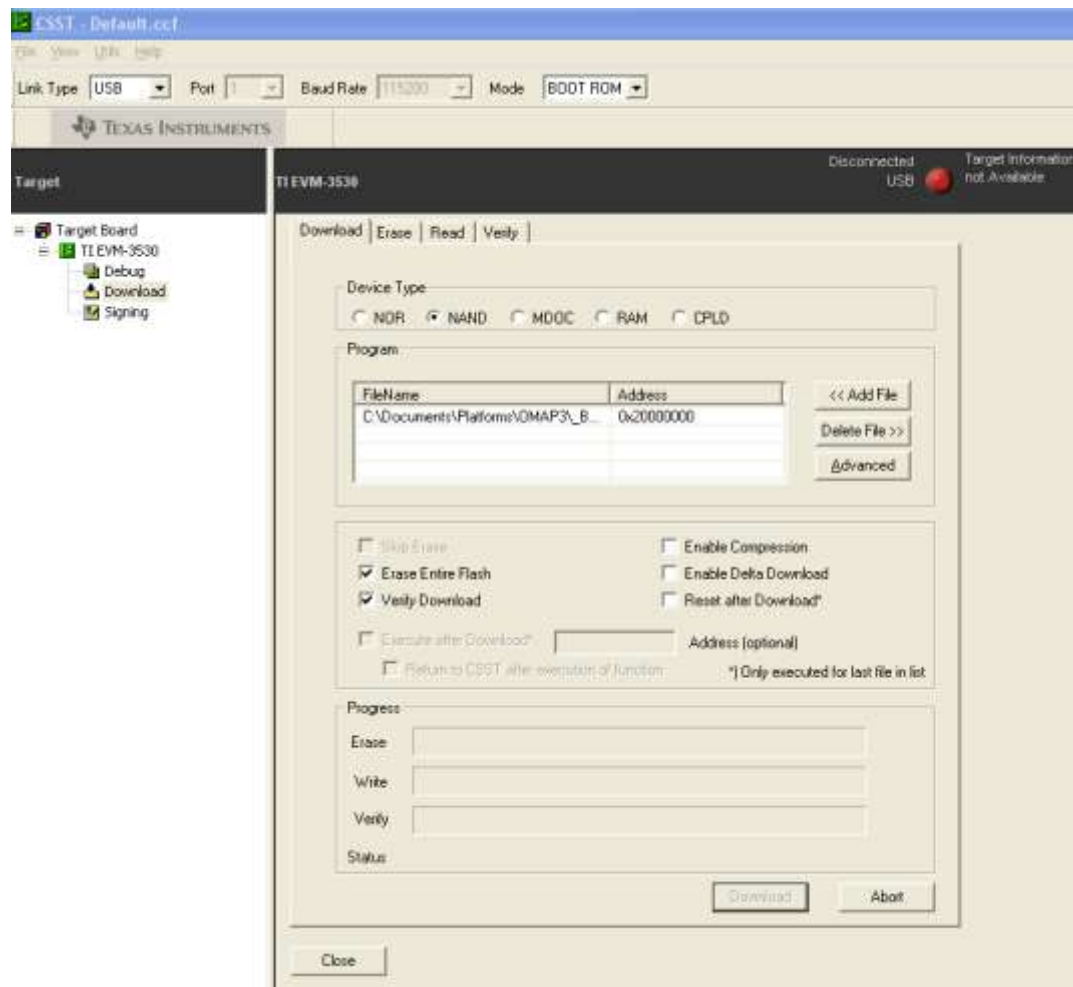


Figure 3. EVMFlash Application, main window.

**Step 9. (USB only) If Host PC USB RNIDS Drivers have not been previously installed:**

- a) Connect the USB cable between the Host PC and the target EVM.
- b) Turn on the power to the EVM.
- c) When prompted to install drivers for the new USB device, select the EVMFlash USB driver located in the folder usb\_drv\_windows where EVMFlash is installed.
- d) After installation of the drivers is finished, turn off the EVM power

**Step 10. (USB only) Plug USB cable between the Host PC and the target EVM.**

**Step 11. Press the Download button. When the “Reset Target to proceed” window appears, turn on the power to the EVM (Switch SW2 shall be on DC position).**

**Step 12. EVMFlash will automatically download and FLASH the bootloader onto the device.**

**Step 13.** After the flashing is completed, turn off power of the EVM and exit EVMFlash. Disconnect the USB cable from the EVM (if using USB download).

**Step 14.** Change the DIP switch SW4 settings as follows depending on your EVM to enable NAND booting (see Chapter 6 for alternative boot configurations):

---

**SW 4 Switch Position (EVM with OneNAND)**

1	2	3	4	5	6	7	8
ON	OFF	OFF	ON	OFF	ON	OFF	OFF

---

**SW 4 Switch Position (EVM with NAND)**

1	2	3	4	5	6	7	8
OFF	OFF	ON	OFF	OFF	ON	OFF	OFF

**Step 15.** Connect the serial cable to UART3 connector.

**Step 16.** Start a serial terminal program (115200 baud, 8 bits, 1 stop)

**Step 17.** Power on the EVM. The bootloader banner will display on the terminal program screen.

**Step 18.** Press the spacebar to break into the bootloader after the formatting of the flash is completed

**Step 19.** EVMFlash does not mark the bootloader blocks in NAND as read-only/reserved. The bootloader will automatically detect this and mark the bootloader blocks as reserved and read-only. This ensures the bootloader blocks will be protected from possible erasure by the OS.