Topaz BSP User Guide

For Windows CE and the Topaz i.MX25 Development Kit

Table of Contents
1 Introduction .......................................................................................................................... 3
2 Topaz Binary BSP .................................................................................................................. 3
3 System requirements ............................................................................................................. 3
  3.1 Important Installation Notes .............................................................................................. 3
4 Topaz BSP Supported features ............................................................................................. 5
5 Topaz BSP Catalog ................................................................................................................... 5
  5.1 Customizing the Topaz OS Design ....................................................................................... 5
  5.2 Adding drivers to the Topaz BSP ....................................................................................... 6
  5.3 Topaz drivers ..................................................................................................................... 7
    5.3.1 Topaz LCD Driver .......................................................................................................... 7
    5.3.2 Topaz GPIO Driver ........................................................................................................ 7
    5.3.3 Topaz GPIO SDK .......................................................................................................... 8
    5.3.4 GPIO SDK example ....................................................................................................... 11
    5.3.5 I²C SDK example ........................................................................................................... 13
    5.3.6 Other SDK Drivers ....................................................................................................... 15
6 Bootloader ........................................................................................................................... 16
  6.1 Serial connection ............................................................................................................... 16
  6.2 Bootloader Menu ............................................................................................................... 17
    6.2.1 [0..4] Network settings ................................................................................................. 17
    6.2.2 [5] Boot Delay ............................................................................................................. 17
    6.2.3 [6] Select Boot Device ................................................................................................. 17
    6.2.4 [7] Reset to Factory Default Configuration .................................................................. 17
    6.2.5 [8] Format OS NAND Region ....................................................................................... 17
    6.2.6 [9] Format All NAND Regions ..................................................................................... 17
    6.2.7 [C] Clean registry & databases .................................................................................... 18
    6.2.8 [B] Bootloader Shell ................................................................................................... 18
    6.2.9 [W] KITL Work Mode .................................................................................................. 18
    6.2.10 [P] KITL Passive Mode ............................................................................................. 18
6.2.11 [E] Select Ether Device : FEC ................................................................. 18
6.2.12 [D] Download Image Now ...................................................................... 19
6.2.13 [L] Launch Existing Flash Resident Image Now ..................................... 19
6.2.14 [M] MMC and SD Utilities ................................................................. 19
6.2.15 [R] Reset ......................................................................................... 19
6.2.16 [S] Save Settings .............................................................................. 19

6.3 Downloading and debugging Windows CE images .................................... 19
   6.3.1 Configure the bootloader for downloading images .............................. 19

7 About us ........................................................................................................... 21
   7.1 GuruCE .............................................................................................. 21
   7.2 Blog .................................................................................................... 21
   7.3 Support options ...................................................................................... 21
1 Introduction

This document provides a detailed description of the Topaz i.MX25 Board Support Package (BSP) for Windows Embedded CE 6.0 R3. The BSP contains all board specific code required to run Windows CE on the Topaz i.MX25 CPU Module & Development Kit. You need the BSP if you want to create your own custom Windows CE image using Platform Builder for Windows CE 6.0 R3.

The Topaz i.MX25 Development Kit comes preinstalled with a Windows CE 6.0 R3 kernel. If the functionality contained in that kernel is sufficient for your needs, you do not need the BSP. The BSP is only needed if you want to create a Windows CE kernel containing different features.

GuruCE provides two versions of the BSP:

- **Binary BSP**
  This BSP is available for free download from [http://guruce.com/topaz/release/latest](http://guruce.com/topaz/release/latest). As the name implies, the Binary BSP consists of binaries, no source code.

- **Source BSP**
  This BSP is available for 2400 USD or 1800 EUR (single design license) and comes with 16 hours of support. As the name implies, the Source BSP contains source code of all drivers and board specific code.

The Topaz BSP is derived from the Freescale’s i.MX25 PDK BSP. GuruCE has made several additions, modifications and bug fixes to the Freescale BSP to make it run smoothly on the Topaz i.MX25 CPU Module and Development Kit. If you need more detailed information than what is provided in this document, please refer to the Freescale i.MX25 PDK documentation which can be found at: [http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=IMX25PDK](http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=IMX25PDK)

2 Topaz Binary BSP


3 System requirements

Platform Builder is the tool required for building Windows CE 6.0 R3 kernels. Platform Builder comes as a plugin for Visual Studio 2005. We recommend you to run the entire Windows CE build environment in a virtual machine running Windows XP SP3. The tools do not support 64bit machines and Windows 7/Vista is causing issues as well.

3.1 Important Installation Notes

The Windows CE OS Design Tools are **not supported on 64 bit operating systems**. Please make sure you install the tools only on a 32bit OS or install on a Virtual PC running a 32 bit OS, see XP Mode for Windows 7 ([http://www.microsoft.com/windows/virtual-pc](http://www.microsoft.com/windows/virtual-pc)) or use VMWare Workstation ([http://www.vmware.com/products/workstation](http://www.vmware.com/products/workstation))

**Tip:** If you have a big number of Windows CE Updates (QFE’s) to install you can download the QFE Installer from GuruCE. This tool automates all the mouse clicks and makes installing QFE’s a breeze: [http://guruce.com/blogpost/downloadqfes](http://guruce.com/blogpost/downloadqfes).

**Tip:** The Windows CE R2 and R3 downloads are huge (>1 GB). If you need to install these updates on multiple PC’s downloading these over and over is a waste of time and bandwidth. You can download some simple tools from the GuruCE website to download these updates to a local folder so you can
install them multiple times without having to download them every time:
http://www.guruce.com/blogpost/offlineinstallationofthece60r2update

Before you can build Windows CE 6.0 R3 kernels you need to install the following programs in exactly this order:

2. Visual Studio 2005 Service Pack 1
3. **If using Windows Vista or Windows 7**: Visual Studio 2005 SP1 Update for Windows Vista
4. Windows Embedded CE 6.0
5. For Topaz you need to select ARMv4I. All other processor architectures can be deselected.
   For source code reference you may want to select the other processor architectures and accept the shared source license.
6. Windows Embedded CE 6.0 R2
7. Windows Embedded CE 6.0 R3
   Download and run the msi files corresponding to the CPU architectures you installed in step 2.
9. Any Windows CE 6.0 Update released after that (DO NOT install any updates released before 2010)
10. Topaz BSP
    a. Unpack the ZIP to your WINCE600 tree (typically C:\WINCE600). After extracting the ZIP file contents to C:\WINCE600 the BSP can be found in C:\WINCE600\PLATFORM\TopazBinary and the Topaz OS Design can be found in C:\WINCE600\OSDesigns\Topaz.

**Warning:** Never use the “Advanced Build” command “Build and Sysgen”. For a explanation why, and how to get rid of this command please read http://www.guruce.com/blogpost/what-to-build-when.

To check if the installation of the above items was performed successfully build the “Topaz” solution (in the WINCE600\OSDesigns\Topaz folder). This should build without any errors (but with some warnings that can be discarded).

**Tip:** Make sure to read http://www.guruce.com/blogpost/what-to-build-when for a better understanding of the Windows CE build system and tips on how to shorten build times.
4 Topaz BSP Supported features

The BSP and OS Design support the Windows Embedded CE 6.0 R3 standard features like KITL Debugging and downloading, power management and support for all available Platform Builder Remote Tools. Additionally the Topaz BSP supports the following features:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 MB Mobile DDR RAM</td>
<td></td>
</tr>
<tr>
<td>10/100 Ethernet</td>
<td></td>
</tr>
<tr>
<td>LCD Auto Detect</td>
<td>Currently 2 LCD’s supported: 4.3” 480x272 and 7” 800x480</td>
</tr>
<tr>
<td>SD Card Interface</td>
<td></td>
</tr>
<tr>
<td>Serial Interfaces</td>
<td>5 total (1x RS232, 4x logic level)</td>
</tr>
<tr>
<td>Freescale MMA7660FC Accelerometer</td>
<td>Including event detection</td>
</tr>
<tr>
<td>Freescale SGTL5000 audio codec</td>
<td>Stereo audio, line and microphone input</td>
</tr>
<tr>
<td>CAN</td>
<td>SDK support</td>
</tr>
<tr>
<td>SPI interface</td>
<td>SDK support</td>
</tr>
<tr>
<td>I2C interface</td>
<td>SDK support</td>
</tr>
<tr>
<td>SIM/smart card interface</td>
<td></td>
</tr>
<tr>
<td>USB OTG</td>
<td></td>
</tr>
<tr>
<td>GPT (General Purpose Timers)</td>
<td>SDK support</td>
</tr>
<tr>
<td>DVFC</td>
<td>Decoupled Voltage and Frequency Controller</td>
</tr>
<tr>
<td>GPIO</td>
<td>Free GPIO depending on board usage</td>
</tr>
<tr>
<td>ADC (Analog to Digital Converter)</td>
<td>SDK support</td>
</tr>
</tbody>
</table>

Table 1 – Topaz BSP feature

5 Topaz BSP Catalog

The Topaz BSP integrates into the Platform Builder Component Catalog. Using the catalog, OS Design developers can add or remove BSP functionality (drivers, utilities, etc) simply by checking or unchecking the component.

5.1 Customizing the Topaz OS Design

Open the “Topaz” OS Design solution in Visual Studio and open the “Catalog Items View”. You can open this view by selecting the “Catalog items View” in the View | Other Windows | “Catalog Items View” (see Figure 1)
Navigate to the “Third Party” category:

When you expand the TopazBinary BSP node you’ll see a list of drivers and components the Topaz BSP exposes. You can manually add or remove drivers or components from the OS Design by simply checking or unchecking the check box in front of the specific component. Some components are included automatically because other components depend on them. For instance, if you include the display driver you’ll see that the I²C driver is automatically included as well because the display driver uses the I²C driver to communicate with the backlight chip. You can distinguish components included by dependency by the green coloured square instead of a symbol.

5.2 Adding drivers to the Topaz BSP

If you are using the Topaz i.MX25 CPU Module in your own design, you may need to add a driver for some device connected to the module on your board to the BSP. Once you’ve developed and added the driver to the BSP you also might want to add the driver to the catalog. To do that you’ll need to add entries for the driver to TopazBinary.pbxml located in the BSP in the folder CATALOG.

You can open the catalog file with Visual Studio 2005 via the Visual Studio file menu or by double clicking the TopazBinary.pbxml file. Once opened you are able to customize the catalog to your specific requirements and add your drivers or applications:
5.3 Topaz drivers

This section describes the drivers and packages available in the Topaz BSP. The Topaz BSP is derived from the i.MX25 Development Kit from Freescale. This document only describes the additions & modifications to the Freescale BSP. More detailed information can be found in the i.MX25 PDK Windows CE Reference Manual available in the documentation section of the Freescale website: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=iMX25PDK

5.3.1 Topaz LCD Driver

The Topaz LCD driver is capable of automatically detecting the attached LCD type. Currently it supports the 4.3" 480x272 and 7" 800x480 LCD available from Device Solutions.

If you want to rotate the screen you can do so by changing the registry settings in mx25_lcdc.reg:

```plaintext
[HKEY_LOCAL_MACHINE\SYSTEM\GDI\ROTATION]
    "Angle"=dword:0 ;; no rotation
    ; "Angle"=dword:5A ;; 90 degrees rotated clockwise
    ; "Angle"=dword:B4 ;; 180 degrees rotated clockwise
    ; "Angle"=dword:10E ;; 270 degrees rotated clockwise
```

5.3.2 Topaz GPIO Driver

The Topaz GPIO driver makes configuring, reading and writing GPIO very easy. The i.MX25 uses a very complex multiplexing scheme to set various functions per pin and to set pin features (like drive strength, voltage, pull-ups, etc). Some pins require you to set 3 different PAD configurations plus the GPIO direction and data registers. The GPIO driver takes away the complexity of it all and simply offers you a simple interface through which you can set all features per pin in one single call.

The GPIO driver needs to be configured to set the “safe” set of GPIO’s available. Free GPIO depends on the board design. For instance; if you are using SPI module 1 you won’t be able to use GPIO 1.14, 1.15, 1.16, 1.17, 1.18, 2.22 and 3.18. If you are not using SPI module 1 than these GPIOs can be freely used by the GPIO driver.

Configure the free GPIOs by setting the port masks in WINCE600\PLATFORM\[Topaz BSP]\SRC\DRIVERS\GPIO\mx25_gpio.reg. This example shows the free GPIO on the Topaz Development Kit:

```plaintext
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\GPIO]
    ; Port 1 Safe GPIO Mask:
    ; 31 0
    ; 0000 0000 0000 0000 0000 0000 0001 0000
    "Port1"=dword:10

    ; Port 2 Safe GPIO Mask:
    ; 31 0
    ; 0000 0000 0001 0000 0000 1111 1111 1111
    "Port2"=dword:100FFF

    ; Port 3 Safe GPIO Mask:
    ; 31 0
    ; 1111 1100 0111 1011 1011 1111 1110 0000
```
5.3.3  Topaz GPIO SDK
The Topaz GPIO SDK provides a simple interface to the driver. Its functions are described below:

5.3.3.1  BOOL GpioInit()
Must be called before any calling any other function.
Returns: FALSE on error. GetLastError() will return an error code.

5.3.3.2  BOOL GpioDeinit()
Must be called when your application doesn’t need to access GPIO anymore. This cleans up handles created in GpioInit.
Returns: FALSE on error. GetLastError() will return an error code.

5.3.3.3  GPIO_CONFIG structure
Port: GPIO Port
- GPIO_PORT_1
- GPIO_PORT_2
- GPIO_PORT_3
- GPIO_PORT_4
Pin: GPIO Pin
- GPIO_PIN_0
- GPIO_PIN_1
- ...
- GPIO_PIN_30
- GPIO_PIN_31
Function: GPIO Function (READ ONLY)
- GPIO_FUNCTION_ALT0
- GPIO_FUNCTION_ALT1
- ...
- GPIO_FUNCTION_ALT6
- GPIO_FUNCTION_ALT7
Direction: GPIO Direction
- GPIO_DIR_IN
- GPIO_DIR_OUT
Interrupt: GPIO Interrupt Configuration
- GPIO_INTR_LOW_LEV
- GPIO_INTR_HIGH_LEV
- GPIO_INTR_RISE_EDGE
- GPIO_INTR_FALL_EDGE
- GPIO_INTR_BOTH_EDGE
- GPIO_INTR_NONE
Loopback: GPIO Loopback mode
- GPIO_LOOPBACK_DISABLE
- GPIO_LOOPBACK_ENABLE
- GPIO_LOOPBACK_INVALID
Slew: GPIO Slew rate
- GPIO_SLEW_SLOW
- GPIO_SLEW_FAST
- GPIO_SLEW_INVALID

Drive: GPIO Drive Strength
- GPIO_DRIVE_NORMAL
- GPIO_DRIVE_HIGH
- GPIO_DRIVE_MAX
- GPIO_DRIVE_INVALID

OpenDrain: GPIO Open Drain mode
- GPIO_OPENDRAIN_DISABLE
- GPIO_OPENDRAIN_ENABLE
- GPIO_OPENDRAIN_INVALID

Pull: GPIO Pull up/pull down/keep Configuration
- GPIO_PULL_NONE
- GPIO_PULL_KEEPER
- GPIO_PULL_DOWN_100K
- GPIO_PULL_UP_47K
- GPIO_PULL_UP_100K
- GPIO_PULL_UP_22K
- GPIO_PULL_INVALID

Hysteresis: GPIO Hysteresis mode
- GPIO_HYSTERESIS_DISABLE
- GPIO_HYSTERESIS_ENABLE
- GPIO_HYSTERESIS_INVALID

Voltage: GPIO Voltage
- GPIO_VOLTAGE_3V3
- GPIO_VOLTAGE_1V8
- GPIO_VOLTAGE_INVALID

5.3.3.4  BOOL GpioGetConfig(LPGPIO_CONFIG pGpioConfig)

Gets the current GPIO configuration

In: Port, Pin

Out: All other GPIO_CONFIG members

Note: GPIO_CONFIG members set to GPIO_XXX_INVALID indicate that field is not configurable for the specified GPIO. See appendix A in the Freescale i.MX25 Reference Manual

Returns: FALSE on error (GetLastError() will return error code)

5.3.3.5  BOOL GpioSetConfig(LPGPIO_CONFIG pGpioConfig)

Sets the current GPIO configuration

In: GPIO_CONFIG members set to desired configuration.

Note: Member 'Function' is ignored (will always be set to the 'GPIO' function)

Out: Configuration as read back after setting it (this can be different than what you expect because not all PADs support all configurations; see 4.7.2 in the iMX25 Reference Manual)

Returns: FALSE on error (GetLastError() will return error code)

Note: This function will return FALSE with GetLastError() ERROR_ACCESS_DENIED if the GPIO is not 'safe' to set as specified by the Port1...Port4 values in [HKEY_LOCAL_MACHINE\Drivers\BuiltIn\GPIO]
5.3.3.6  GPIO structure

5.3.3.6.1  GPIO for Pin Functions
Port: GPIO Port
- GPIO_PORT_1
- GPIO_PORT_2
- GPIO_PORT_3
- GPIO_PORT_4
Pin: GPIO Pin
- GPIO_PIN_0
- GPIO_PIN_1
- ...
- GPIO_PIN_30
- GPIO_PIN_31
Value: in/out depending on function

5.3.3.6.2  GPIO for Port Functions
Port: GPIO Port
- GPIO_PORT_1
- GPIO_PORT_2
- GPIO_PORT_3
- GPIO_PORT_4
Mask: 32 bit GPIO Pin Mask
Value: in/out depending on function

5.3.3.7  BOOL GpioReadPin(LPGPIO pGpio)
Reads the GPIO pin value
In: Port, Pin
Out: Value (1 or 0)
Returns: FALSE on error (GetLastError() will return error code)

5.3.3.8  BOOL GpioWritePin(LPGPIO pGpio)
Sets the GPIO pin value
In: Port, Pin
Out: Value (1 or 0)
Returns: FALSE on error (GetLastError() will return error code)
Note: This function will return FALSE with GetLastError() ERROR_ACCESS_DENIED if the GPIO is not 'safe' to set as specified by the Port1...Port4 values in [HKEY_LOCAL_MACHINE\Drivers\BuiltIn\GPIO]

5.3.3.9  BOOL GpioReadPort(LPGPIO pGpio)
Reads multiple GPIO pins on the same port
In: Port, Mask
Out: Value
Returns: FALSE on error (GetLastError() will return error code)

5.3.3.10 BOOL GpioWritePort(LPGPIO pGpio)
Sets multiple GPIO pins on the same port
In: Port, Mask
Out: Value
Returns: FALSE on error (GetLastError() will return error code)
Note: This function will adjust (and return) the Mask according to the 'safe' mask as specified by the Port1...Port4 values in [HKEY_LOCAL_MACHINE\Drivers\BuiltIn\GPIO] and will return FALSE with GetLastError() ERROR_ACCESS_DENIED if the Mask is 0 (no GPIO's could be safely written)

5.3.3.11 BOOL GpioReadIntrPin(LPGPIO pGpio)
Reads the interrupt status of the GPIO pin
In: Port, Mask
Out: Value
Returns: FALSE on error (GetLastError() will return error code)

5.3.3.12 BOOL GpioReadIntrPort(LPGPIO pGpio)
Reads the interrupt status of multiple GPIO pins on the same port
In: Port, Mask
Out: Value
Returns: FALSE on error (GetLastError() will return error code)

5.3.3.13 BOOL GpioClearIntrPin(LPGPIO pGpio)
Cleareds the interrupt of the GPIO pin
In: Port, Mask
Out: Value
Returns: FALSE on error (GetLastError() will return error code)
Note: This function will return FALSE with GetLastError() ERROR_ACCESS_DENIED if the GPIO is not 'safe' to set as specified by the Port1...Port4 values in [HKEY_LOCAL_MACHINE\Drivers\BuiltIn\GPIO]

5.3.3.14 BOOL GpioClearIntrPort(LPGPIO pGpio)
Cleareds the interrupt of multiple GPIO pins on the same port
In: Port, Mask
Out: Value
Returns: FALSE on error (GetLastError() will return error code)
Note: This function will adjust (and return) the Mask according to the 'safe' mask as specified by the Port1...Port4 values in [HKEY_LOCAL_MACHINE\Drivers\BuiltIn\GPIO] and will return FALSE with GetLastError() ERROR_ACCESS_DENIED if the Mask is 0 (no GPIO's could be safely written)

5.3.4 GPIO SDK example
The code below shows how to use the GPIO SDK header (gpiosdk.h) and library (gpiosdk.lib). This example sets the user led (on the bottom of the Topaz Development Kit) according to the user switch position (switch 4). The program terminates when the user presses the user button. Note that for this code to work properly you have to exclude the “Power Button” driver (since it configures the user button as an interrupt wake source).

#include <gpiosdk.h>

int WINAPI WinMain(HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPTSTR     lpCmdLine,
                    int        nCmdShow)
{
    if (GpioInit())
    {
        GPIO_CONFIG gpioConfig;
        gpioConfig.Port = GPIO_PORT_4; // Select user button
        gpioConfig.Pin = GPIO_PIN_4;
        gpioConfig.Direction = GPIO_DIR_IN;
        gpioConfig.Drive = GPIO_DRIVE_NORMAL;
        gpioConfig.Hysteresis = GPIO_HYSTERESIS_ENABLE;
        gpioConfig.Interrupt = GPIO_INTR_NONE;
        ...
    }
    // Your code here...
}
gpioConfig.Loopback = GPIO_LOOPBACK_DISABLE;
gpioConfig.OpenDrain = GPIO_OPENDRAIN_ENABLE;
gpioConfig.Pull = GPIO_PULL_UP_22K;
gpioConfig.Slew = GPIO_SLEW_FAST;
gpioConfig.Voltage = GPIO_VOLTAGE_3V3;
GpioSetConfig(&gpioConfig);

gpioConfig.Port = GPIO_PORT_1; // Select user switch
gpioConfig.Pin = GPIO_PIN_4;
gpioConfig.Direction = GPIO_DIR_IN;
gpioConfig.Drive = GPIO_DRIVE_NORMAL;
gpioConfig.Hysteresis = GPIO_HYSTERESIS_ENABLE;
gpioConfig.Interrupt = GPIO_INTR_NONE;
gpioConfig.Loopback = GPIO_LOOPBACK_DISABLE;
gpioConfig.OpenDrain = GPIO_OPENDRAIN_ENABLE;
gpioConfig.Pull = GPIO_PULL_UP_22K;
gpioConfig.Slew = GPIO_SLEW_FAST;
gpioConfig.Voltage = GPIO_VOLTAGE_3V3;
GpioSetConfig(&gpioConfig);

gpioConfig.Port = GPIO_PORT_3; // Select user LED
gpioConfig.Pin = GPIO_PIN_17;
gpioConfig.Direction = GPIO_DIR_OUT;
gpioConfig.Drive = GPIO_DRIVE_NORMAL;
gpioConfig.Hysteresis = GPIO_HYSTERESIS_DISABLE;
gpioConfig.Interrupt = GPIO_INTR_NONE;
gpioConfig.Loopback = GPIO_LOOPBACK_DISABLE;
gpioConfig.OpenDrain = GPIO_OPENDRAIN_ENABLE;
gpioConfig.Pull = GPIO_PULL_NONE;
gpioConfig.Slew = GPIO_SLEW_FAST;
gpioConfig.Voltage = GPIO_VOLTAGE_3V3;
GpioSetConfig(&gpioConfig);

GPIO gpioButton = {GPIO_PORT_4, GPIO_PIN_4, 1};
GPIO gpioSwitch = {GPIO_PORT_1, GPIO_PIN_4, 1};
GPIO gpioLED = {GPIO_PORT_3, GPIO_PIN_17, 1};
do {
    GpioReadPin(&gpioButton);
    GpioReadPin(&gpioSwitch);
    gpioLED.Value = !gpioSwitch.Value;
    GpioWritePin(&gpioLED);
    Sleep(500);
} while (1 == gpioButton.Value); // Wait for user button press

GpioDeinit();
return 0;
5.3.5 I2C SDK example

The Inter-Integrated Circuit (I2C) module is responsible for handling the communication from the CPU to a device connected to the I2C bus.

5.3.5.1 Communicating with a device over I2C

Communicating directly with a driver involves getting a handle to the driver and talking to it through Device IO Control calls. All these low level difficult calls are handled by a DLL that makes communicating with a device over I2C as easy.

By including i2cbus.h and linking to i2csdk.lib (both available in the SDK) you’ll get access to the following list of functions:

```c
HANDLE I2COpenHandle(LPCWSTR lpDevName);
BOOL I2CCloseHandle(HANDLE hI2C);
BOOL I2CSetSlaveMode(HANDLE hI2C);
BOOL I2CSetMasterMode(HANDLE hI2C);
BOOL I2CIsMaster(HANDLE hI2C, PBOOL pbIsMaster);
BOOL I2CIsSlave(HANDLE hI2C, PBOOL pbIsSlave);
BOOL I2CGetClockRate(HANDLE hI2C, PWORD pwClkRate);
BOOL I2CSetClockRate(HANDLE hI2C, WORD wClkRate);
BOOL I2CGetFrequency(HANDLE hI2C, DWORD dwFreq);
BOOL I2CGetSelfAddr(HANDLE hI2C, BYTE bySelfAddr);
BOOL I2CGetSlaveSize(HANDLE hI2C, DWORD dwSize);
BOOL I2CTransfer(HANDLE hI2C, PI2C_TRANSFER_BLOCK pI2CTransferBlock);
BOOL I2CReset(HANDLE hI2C);
BOOL I2CEnableSlave(HANDLE hI2C);
BOOL I2CDisableSlave(HANDLE hI2C);
```

An example on how to use these functions is listed below. The Topaz i.MX25 Development Kit contains a 3-axis orientation/motion detection sensor (accelerometer) that is connected to the I2C bus. The code below shows you how to communicate with this chip:

```c
#include <i2cbus.h>
#define MMA7660FC_I2C_ADDRESS 0x4C
typedef enum : BYTE
{
    XOUT,
    YOUT,
    ZOUT,
    TILT,
    SRST,
    SPCNT,
    INTSU,
    MODE,
    SR,
    PDT,
    PD
} MMA7660FC_REGISTERS;

BOOL WriteI2CRegister(HANDLE hI2C, BYTE devAddress, BYTE regAddress, BYTE regValue)
{
    BOOL     retValue;
    INT32    Result;
    I2C_TRANSFER_BLOCK I2CXferBlock;
    I2C_PACKET   I2CPacket;
    BYTE     writeData[2];

    // Set the write data
    writeData[0] = regAddress;
    writeData[1] = regValue;
```
// Write register
I2CPacket.byAddr = devAddress;
I2CPacket.byRW = I2C_RW_WRITE;
I2CPacket.pbyBuf = writeData;
I2CPacket.wLen = sizeof(writeData);
I2CPacket.lpiResult = &Result;
I2CXferBlock.pI2CPackets = &I2CPacket;
I2CXferBlock.iNumPackets = 1;
retValue = I2CTransfer(hI2C, &I2CXferBlock);
RETAILMSG(!retValue, (L"ERROR: Failed to write 0x%02X to I2C register 0x%02X in device 0x%02X! (Result:%d, Err:%d)
" , regValue, regAddress,
    devAddress, Result, GetLastError()));

return retValue;
}

BYTE ReadI2CRegister(HANDLE hI2C, BYTE devAddress, BYTE regAddress)
{
    BYTE     retValue = 0;
    INT32    Result;
    I2C_TRANSFER_BLOCK I2CXferBlock;
    I2C_PACKET   I2CPacket[2];

    // Select register
    I2CPacket[0].byAddr = devAddress;
    I2CPacket[0].byRW = I2C_RW_WRITE;
    I2CPacket[0].pbyBuf = &regAddress;
    I2CPacket[0].wLen = sizeof(regAddress);
    I2CPacket[0].lpiResult = &Result;

    // Read register
    I2CPacket[1].byAddr = devAddress;
    I2CPacket[1].byRW = I2C_RW_READ;
    I2CPacket[1].pbyBuf = &retValue;
    I2CPacket[1].wLen = sizeof(retValue);
    I2CPacket[1].lpiResult = &Result;

    I2CXferBlock.pI2CPackets = I2CPacket;
    I2CXferBlock.iNumPackets = sizeof(I2CPacket)/sizeof(I2CPacket[0]);

    if(!I2CTransfer(hI2C, &I2CXferBlock))
        RETAILMSG(1, (L"ERROR: Failed to read I2C register 0x%02X from device 0x%02X! 
    (Result:%d, Err:%d)
" , regAddress, devAddress, Result, GetLastError()));

    return retValue;
}

int ReadAxis(HANDLE hI2C, BYTE devAddress, MMA7660FC_REGISTERS Axis)
{
    int retValue = 0;
    if ((XOUT == Axis) || (YOUT == Axis) || (ZOUT == Axis))
    {
        do
            // Read axis until Alert bit is not set (see datasheet)
            retValue = ReadI2CRegister(hI2C, devAddress, Axis);
        while (retValue & (1<<6));

        // Take sign bit into account
        if (retValue & (1<<5))
            retValue = -(retValue & 0x1F);
        else
            retValue = (retValue & 0x1F);
    }

    return retValue;
}

int _main(int argc, _TCHAR* argv[])
{
    // Open the I2C driver
    HANDLE hI2C = I2COpenHandle(L"I2C1:"
    if (INVALID_HANDLE_VALUE != hI2C)
    
    return retValue;
    
}
```c
{ // Use a loop for easy break out cleanup
    for (;;) {
        // Set Topaz as I2C master
        if (!I2CSetMasterMode(hI2C)) {
            RETAILMSG(1, (L"ERROR: Failed to set Topaz as I2C master! (Err:%d)\r\n", GetLastError()));
            break;
        }

        // Set I2C Clock to 400 kHz
        if (!I2CSetFrequency(hI2C, 400 * 1000)) {
            RETAILMSG(1, (L"ERROR: Failed to set I2C frequency to 400kHz! (Err:%d)\r\n", GetLastError()));
            break;
        }

        // Read accelerometer Mode Register
        BYTE mode = ReadI2CRegister(hI2C, MMA7660FC_I2C_ADDRESS, MODE);
        // Set accelerometer Active Bit
        mode |= 0x01;
        WriteI2CRegister(hI2C, MMA7660FC_I2C_ADDRESS, MODE, mode);

        // Read the accelerometer axis values and report
        for (int i=0; i<10000; i++) {
            int xVal = ReadAxis(hI2C, MMA7660FC_I2C_ADDRESS, XOUT);
            int yVal = ReadAxis(hI2C, MMA7660FC_I2C_ADDRESS, YOUT);
            int zVal = ReadAxis(hI2C, MMA7660FC_I2C_ADDRESS, ZOUT);

            RETAILMSG(1, (L"X=%3d, Y=%3d, Z=%3d\r\n", xVal, yVal, zVal));
        }

        // Always break out of loop
        break;
    } // Clean up
    I2CCloseHandle(hI2C);
} else
    RETAILMSG(1, (L"ERROR: Can't open I2C driver! (Err:%d)\r\n", GetLastError()));

    return 0;
}
```

Please refer to the Freescale reference manual for a full description of all the functions in the I²C SDK library.

### 5.3.6 Other SDK Drivers

The BSP automatically copies the LIB and header files of the following drivers to the correct location so that they are included in the SDK (if these drivers are included in the OS Design):

- **ADC**
  Analog to Digital Converter: adc_sdk.h, adcsdk.lib
- **CAN**
  CAN Bus: canbus.h, cansdk.lib
- **SPI**
  SPI Bus: cspibus.h, spisdk.lib
- **GPT**
  General Purpose Timers: gpt.h, gptsdk.lib
- **I²C**
  I²C Bus: i2cbus.h, i2csdk.lib

The exported functions of these libraries are easily PInvoked from managed code as well.
6 Bootloader

The bootloader’s main responsibility is to start the Windows CE image. The bootloader can be configured to start Windows CE in several different ways. This section describes how to enter the bootloader menu, and what the various options are.

To enter the bootloader menu you have to connect a serial cable between the Topaz development kit and a laptop or desktop system.

6.1 Serial connection

Connect an RS232 null-modem cable between the Topaz development kit and a PC. Open your favorite terminal client and setup a serial connection with the following settings:

<table>
<thead>
<tr>
<th>Baud rate</th>
<th>115200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data bits</td>
<td>8 bits</td>
</tr>
<tr>
<td>Parity</td>
<td>None</td>
</tr>
<tr>
<td>Stop</td>
<td>1 bit</td>
</tr>
<tr>
<td>Flow control</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 1 - Serial connection settings

Next power the board and the following text should appear in your terminal:

Microsoft Windows CE Ethernet Bootloader 1.0 for MX25 3DS (Apr 13 2010 14:05:08)
INFO: Bootloader launched from NAND
INFO: OEMPlatformInit: Initialized NAND flash device.
INFO: Loading boot configuration from NAND
System ready!
Preparing for download...

Press [ENTER] to launch image stored in NAND flash or [SPACE] to cancel.

Initiating image launch in 2 seconds.

Hit space within 3 seconds to enter the boot loader menu.
6.2 Bootloader Menu

The Topaz i.MX24 bootloader menu:

<table>
<thead>
<tr>
<th>Selection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IP Address : 0.0.0.0</td>
</tr>
<tr>
<td>1</td>
<td>Set IP Mask : 0.0.0.0</td>
</tr>
<tr>
<td>2</td>
<td>Set Gateway : 0.0.0.0</td>
</tr>
<tr>
<td>3</td>
<td>Set MAC Address : 0-50-C2-3F-90-6A</td>
</tr>
<tr>
<td>4</td>
<td>DHCP : Enabled</td>
</tr>
<tr>
<td>5</td>
<td>Boot Delay : 3</td>
</tr>
<tr>
<td>6</td>
<td>Select Boot Device : NK from NAND</td>
</tr>
<tr>
<td>7</td>
<td>Reset to Factory Default Configuration</td>
</tr>
<tr>
<td>8</td>
<td>Format OS NAND Region</td>
</tr>
<tr>
<td>9</td>
<td>Format All NAND Regions</td>
</tr>
<tr>
<td>C</td>
<td>Clean registry &amp; databases : Disable</td>
</tr>
<tr>
<td>B</td>
<td>Bootloader Shell</td>
</tr>
<tr>
<td>W</td>
<td>KITL Work Mode : Interrupt</td>
</tr>
<tr>
<td>K</td>
<td>KITL Enable Mode : Disable</td>
</tr>
<tr>
<td>P</td>
<td>KITL Passive Mode : Disable</td>
</tr>
<tr>
<td>E</td>
<td>Select Ether Device : FEC</td>
</tr>
<tr>
<td>D</td>
<td>Download Image Now</td>
</tr>
<tr>
<td>L</td>
<td>Launch Existing Flash Resident Image Now</td>
</tr>
<tr>
<td>M</td>
<td>MMC and SD Utilities</td>
</tr>
<tr>
<td>R</td>
<td>Reset</td>
</tr>
<tr>
<td>S</td>
<td>Save Settings</td>
</tr>
</tbody>
</table>

6.2.1 [0..4] Network settings
Options 0-4 allow you to configure the network. You can set IP address, IP Mask, Gateway, MAC address and whether or not to use DHCP. Note that these settings are not only used to establish the debug connection (see ) but are also reflected in Windows CE.

6.2.2 [5] Boot Delay
Option 5 in the menu lets you configure the number of seconds the bootloader will wait before it jumps to its configured action. The default boot delay is set to 3 second. The user can cancel the default action and jump into the bootloader menu by pressing the space within the delay specified here.

6.2.3 [6] Select Boot Device
This option can be used to select the default boot device. The bootloader will try to load an image from the device selected here. You can choose from the following boot devices:

- NK from NAND (default)
- NK from SD/MMC
- Disabled (used for debugging)

6.2.4 [7] Reset to Factory Default Configuration
This will restore the original boot loader settings as they were when the device was shipped.

6.2.5 [8] Format OS NAND Region
This option will erase the Windows CE image from flash.

6.2.6 [9] Format All NAND Regions
This option will erase the entire flash, including the bootloader settings. After this command you'll have to reflash XLDR, EBOOT and NK using the Topaz Flasher to get the device working again, and
you’ll have to re-enter the MAC address and setup the boot settings. Note that the device will not boot without a valid MAC address set!

6.2.7 [C] Clean registry & databases
The default Topaz OS Design uses a hive based persistent registry. This means registry settings made in Windows CE are persisted (saved) between power cycles or resets. If, for some reason, the registry on the device gets corrupted it may prevent Windows CE from booting up. Using this option in the bootmenu will force Windows CE to start with a clean registry and clean databases. The normal way of using this option is to enable this option and immediately choose option ‘L’ to load the existing Windows CE kernel in flash (so without saving the setting to flash). If you don’t want Windows CE to persist any settings in between power cycles or resets, you can of course enable this option and saving this setting to flash (by choosing the ‘S’ option). That way Windows CE will always start up with clean registry and databases.

6.2.8 [B] Bootloader Shell
The bootloader shell provides a way to peek and poke memory and memory mapped I/O.
**WARNING:** This option is for advanced users only. Incorrect use of the options here can damage your board!

```
-----------  Topaz i.MX25 Boot Shell  ----------
type ? for help
command -- ?

-------------------------------------- Help --------------------------------------
?                                               Help
  e                                              Exit Shell
d RegAddress                                      Show Reg
s RegAddress RegValue                             Set Reg
b RegAddress BitOffset(0-31) Value(0 or 1)        Set Bit

---------------------------------------- End ----------------------------------------

All the input parameters are separated by space key!
command -- ?
```

6.2.9 [W] KITL Work Mode
The Kernel Independent Transport Layer (KITL) is a communications protocol between Platform Builder on your development PC and the Topaz i.MX25 CPU Module. KITL can work in either interrupt or polling mode. Interrupt mode is recommended because it’s more efficient and hence performs better. Polling is slower and does not allow near-instantaneous user initiated breaks but it can be useful when in early system debugging before peripheral interrupts are enabled. By default this option is disabled because the image automatically starts from NAND without KITL being required. If you want to enable debugging on the Topaz with platform builder, you need to set the KITL mode.

6.2.10 [P] KITL Passive Mode
KITL can be started and used in 2 different modes: Passive and Active mode. Active KITL is best suited for development (the debugger can maintain a constant connection) and passive KITL is better suited in a real-world scenario where the debugger is not constantly needed. Passive KITL allows you to connect a debugger after the device has crashed, allowing you to investigate the cause of the crash. For a more detailed description about these 2 modes please refer to MSDN:

6.2.11 [E] Select Ether Device : FEC
This option allows you to select the Ethernet transport. FEC is the Ethernet MAC. Other options are USB Serial and USB RNDIS.
6.2.12 [D] Download Image Now
This option will initialize the selected ether device and start the image download.

6.2.13 [L] Launch Existing Flash Resident Image Now
Selecting ‘L’ will launch the image which is stored into NAND.

6.2.14 [M] MMC and SD Utilities
This will option will take you to a sub-menu of where you can use several options to, for example, format the MMC and/or SD storage devices, if present on the board.

6.2.15 [R] Reset
This option will (warm) reset the Topaz

6.2.16 [S] Save Settings
The “Save Settings” option writes the menu settings to NAND flash.

6.3 Downloading and debugging Windows CE images
The Topaz OS Design provided in each release contains 3 configurations:

- Debug (Debug info, DEBUGMSG & RETAILMSG enabled, kernel debugger, KITL and remote tooling capabilities)
- Release (Compiler optimizations, RETAILMSG enabled, KITL and remote tooling capabilities)
- Shipbuild (Compiler optimizations, flashed to NAND when downloading to the Topaz)

Each of the images generated from the above configurations can be downloaded to the Topaz. To be able to download an image you need to configure the bootloader using the options described above.

6.3.1 Configure the bootloader for downloading images
When developing drivers you most likely want to use the release or debug configuration. These configurations allow you to break the entire kernel and step through your driver code. To do this you must enable KITL and configure the Topaz so that it automatically starts downloading the image at boot:

```
[0] IP Address : 0.0.0.0
[1] Set IP Mask : 0.0.0.0
[2] Set Gateway : 0.0.0.0
[3] Set MAC Address : 00-14-CA-38-16-EB
...
[6] Select boot Device: Disabled
...
[K] KITL Enable Mode : Enable
```

Don’t forget to save your settings by pressing ‘S’ (Save settings) in the bootloader menu.
Once you’ve finished your driver development you’ll probably want to flash the Topaz so you can ship it to a customer. In that case you’ll build the Shipbuild configuration and download it to the Topaz. The Topaz will recognize it’s a final shipbuild image and will ask you if you want to flash the image to NAND:

<table>
<thead>
<tr>
<th>BL_IMAGE_TYPE_BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO: OEMMultiBINNotify (dwNumRegions = 1, dwRegionStart = 0x91580000).</td>
</tr>
<tr>
<td>INFO: OEMVerifyMemory (CA = 0x91580000, PA = 0xBB280000, length = 0x1C56200)</td>
</tr>
<tr>
<td>INFO: Downloading NK NAND image.</td>
</tr>
<tr>
<td>rom_offset=0x11480000.</td>
</tr>
<tr>
<td>ImageStart = 0x91580000, ImageLength = 0x1C56200, LaunchAddr = 0x91581000</td>
</tr>
<tr>
<td>Completed file(s):</td>
</tr>
<tr>
<td>[0]: Address=0x91580000 Length=0x1C56200 Name=&quot;&quot; Target=FLASH</td>
</tr>
</tbody>
</table>

WARNING: Flash update requested. 
Do you want to continue (y/n)?

Select ‘Y’ if you want to flash or ‘N’ otherwise. When it’s done flashing reset the board by pressing ‘Y’ followed by enter.

| INFO: Writing NK image to NAND (please wait)...
INFO: Programming NAND flash blocks [0x14 ~ 0x113].
INFO: Programming and Verifying image are 100% completed.
INFO: Verifying image succeed.
INFO: Downloading NK NAND image. |
| rom_offset=0x11480000. |
| ImageStart = 0x91580000, ImageLength = 0x1C56200, LaunchAddr = 0x91581000 |
| Completed file(s): |
| [0]: Address=0x91580000 Length=0x1C56200 Name="" Target=FLASH |

INFO: OEMMultiBINNotify (dwNumRegions = 1, dwRegionStart = 0x91580000).
INFO: OEMVerifyMemory (CA = 0x91580000, PA = 0xBB280000, length = 0x1C56200)
INFO: Downloading NK NAND image.
rom_offset=0x11480000.
ImageStart = 0x91580000, ImageLength = 0x1C56200, LaunchAddr = 0x91581000
Completed file(s):
[0]: Address=0x91580000 Length=0x1C56200 Name="" Target=FLASH

WARNING: Flash update requested.
Do you want to continue (y/n)?

Select ‘Y’ if you want to flash or ‘N’ otherwise. When it’s done flashing reset the board by pressing ‘Y’ followed by enter.

INFO: Writing NK image to NAND (please wait)...
INFO: Programming NAND flash blocks [0x14 ~ 0x113].
INFO: Programming and Verifying image are 100% completed.
INFO: Verifying image succeed.
INFO: Downloading NK NAND image.
rom_offset=0x11480000.
ImageStart = 0x91580000, ImageLength = 0x1C56200, LaunchAddr = 0x91581000
Completed file(s):
[0]: Address=0x91580000 Length=0x1C56200 Name="" Target=FLASH

Of course when you want to start the image just flashed, you’ll need to re-configure the bootloader’s “Boot device” to “NK from NAND” and save the settings to the bootloader menu.
7 About us

7.1 GuruCE
GuruCE offers deep technical knowledge of the Windows Embedded CE (Windows Embedded Compact Edition) operating system. The consultants of GuruCE are among the best in Windows CE BSP & driver development, training and consulting.

GuruCE can help you and your company get to market faster by taking care of all the Windows CE low-level issues so that your experts can focus on what they do best, or we can teach you how to do it yourself through training by one of our consultants. We can help you with general system design (both hardware & software), application design & development, real-time embedded design issues and driver development.

7.2 Blog
For general tips & tricks on Windows CE and other related issues please have a look at our blog: http://guruce.com/blog.

7.3 Support options
GuruCE offers various support options:

- **Custom OS Design**
  If you don't have Platform Builder or don't have the time or expertise to generate an image using the Topaz Binary BSP we can do it for you. We'll create a Windows CE kernel for you according to your specific needs. We can even sort out the correct Microsoft CE licenses for you.
  Cost: 600 USD / 450 EUR

- **Source BSP**
  If you need to change or modify existing drivers or OAL code you can buy the complete source code of the BSP including all drivers. This package comes with 8 hours of dedicated support.
  Cost: 2400 USD / 1800 EUR

- **Extra support**
  16 hours of extra support, development services or consultancy for your project.
  Cost: 2400 USD / 1800 EUR

Please contact us directly for more support options and more detailed information on how we can help you:

**GuruCE APAC/NZ**
Contact: Michel Verhagen
Email : michel@guruce.com
Phone : +64 (0)9 525 0070
Mobile : +64 (0)21 104 6208
292 Ellerslie-Panmure Highway
Ellerslie, Auckland 1060
New Zealand

**GuruCE EMEA/NL**
Contact: Erwin Zwart
Email : erwin@guruce.com
Phone : +31 (0)72 850 3119
Mobile : +31 (0)629 512 116
Tuin van Halo 19
Heerhugowaard, 1705 TD
The Netherlands