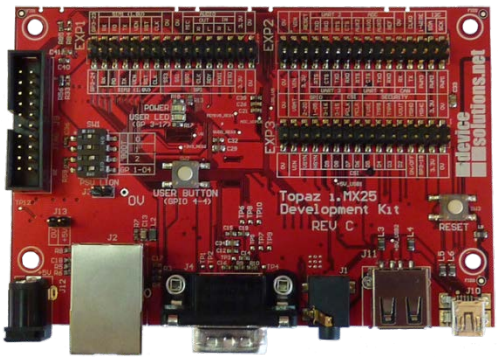
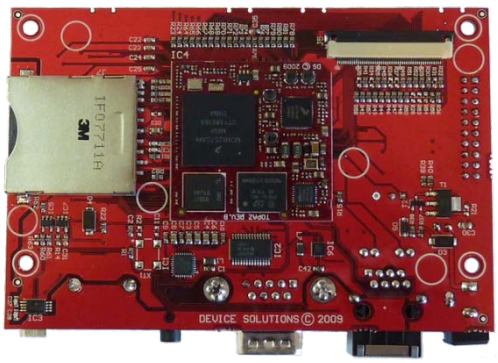
Topaz BSP User Guide

For Windows CE and the Topaz i.MX25 Development Kit

Table of Contents





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 the full source BSP free of charge with any support contract. Please visit http://guruce.com/support for more information.

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

2 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.

2.1 Important Installation Notes

The Windows CE OS Design Tools are **not officially supported on 64 bit operating systems**[1]**.** 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) or use VMWare 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.*

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
http://www.guruce.com/blogpost/offline-installation-of-ce-60-r3

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

[1] Unofficial guide to setup the Windows CE 6.0 tools on a 64-bit OS can be found online: http://geekswithblogs.net/WindowsEmbeddedCookbook/archive/2010/08/31/installing-windows-ce-6.0-tools-on-a-windows7-64bit-pc.aspx

1. Visual Studio 2005 Professional Edition
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
   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.
5. Windows Embedded CE 6.0 R2
6. Windows Embedded CE 6.0 R3
7. The latest (and only the latest) yearly "Cumulative Product Update Rollup" for Windows Embedded CE 6.0. Download and install the msi files corresponding to the CPU architectures you installed in step 4.
8. Any Windows CE 6.0 Update released after that (DO NOT install any updates released before the year of the yearly rollup installed in step 7 above). Download and install the msi files corresponding to the CPU architectures you installed in step 4.
9. 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:\WINC600\PLATFORM\Topaz and the Topaz OS Design can be found in C:\WINCE600\OSDesigns\Topaz.

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).

Warning: *Never use the "Advanced Build" command "Build and Sysgen". For an explanation why, and how to get rid of this command please read http://www.guruce.com/blogpost/what-to-build-when and http://guruce.com/blogpost/how-to-remove-the-demonic-build-and-sysgen-commands-from-platform-builder-for-windows-embed.*

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.*

3 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:

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

Table 1 – Topaz BSP feature

4 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.

4.1 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)

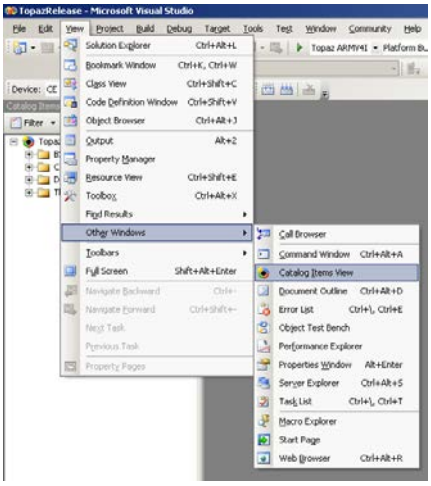


Figure 1 - Catalog Items View

Navigate to the “Third Party” category:

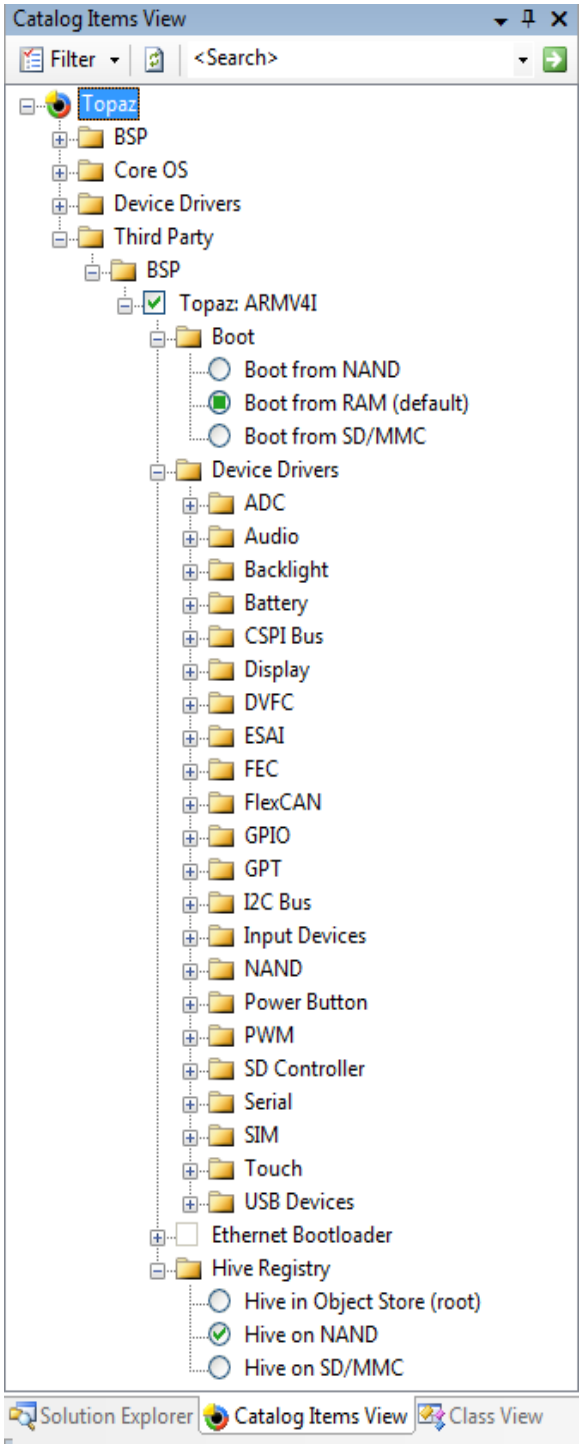


Figure3 - Topaz BSP Catalog Items

When you expand the Topaz 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 I^2C driver is automatically included as well because the display driver uses the I^2C driver to communicate with the backlight chip. You can distinguish components included by dependency by the green coloured square instead of a √ symbol.

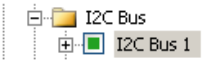


Figure 2 - I2C driver added as a dependency

4.2 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 Topaz.pbcxml 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 Topaz.pbcxml file. Once opened you are able to customize the catalog to your specific requirements and add your drivers or applications:

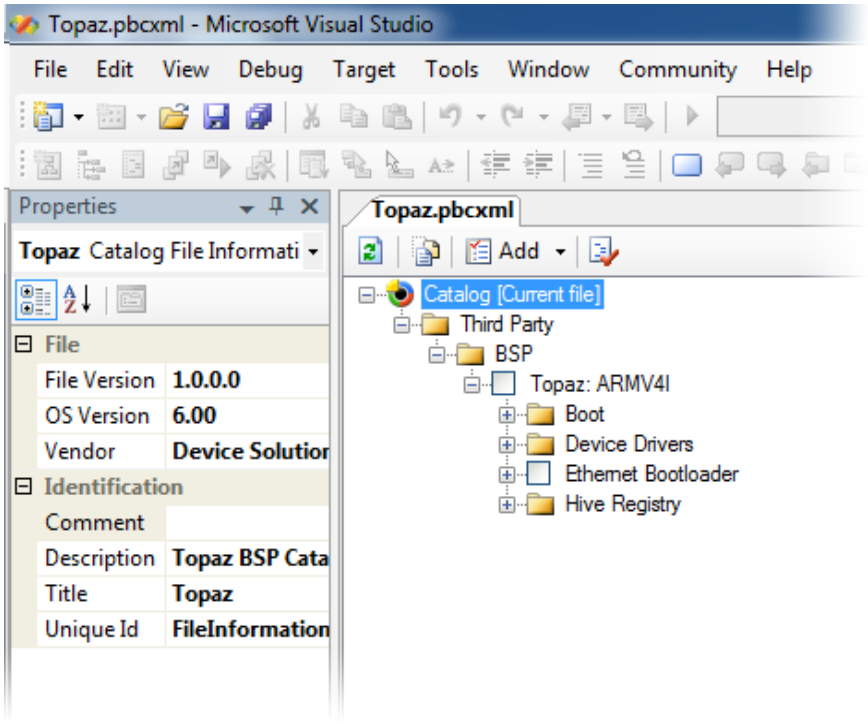


Figure 3 - Topaz Platform builder catalog file (pbcxml file)

For more information about catalog files and how to add catalog components please refer to the Microsoft MSDN library available at:
http://msdn.microsoft.com/en-us/library/ee482151(v=WinEmbedded.60).aspx

4.3 Topaz drivers and SDK headers

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

Note: We have managed wrappers available for most of the SDK headers. Please contact us directly for pricing information.

4.3.1 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:

```
[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
```

4.3.2 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 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\SRC\DRIVERS\GPIO\mx25_gpio.reg. This example shows the free GPIO on the Topaz Development Kit:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\GPIO]
   ;Port 1 Safe GPIO Mask:
   ;31  28   24   20   16   12   8    4    0
   ; 1000 0000 0000 0000 0000 1000 0111 0000
   "Port1"=dword:80000870

   ;Port 2 Safe GPIO Mask:
   ;31  28   24   20   16   12   8    4    0
   ; 0000 0000 0011 0000 0000 1111 1111 1111
   "Port2"=dword:300FFF

   ;Port 3 Safe GPIO Mask:
   ;31  28   24   20   16   12   8    4    0
   ; 0000 0001 1010 1011 1100 0000 0000 0000
   "Port3"=dword:1ABC000

   ;Port 4 Safe GPIO Mask:
   ;31  28   24   20   16   12   8    4    0
   ; 0000 0000 0000 0000 0000 0001 1000 0000
   "Port4"=dword:180
```

4.3.3 Topaz GPIO SDK

The Topaz GPIO SDK provides a simple interface to the driver. Its functions are described below:

4.3.3.1 BOOL GpioInit()

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

4.3.3.2 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.

4.3.3.3 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

4.3.3.4 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)

4.3.3.5 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]

4.3.3.6 GPIO structures

4.3.3.6.1 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

4.3.3.6.2 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

4.3.3.6.3 GPIO for Event (interrupt) 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

pszEvent: Named event to signal when interrupt occurs
dwPriority256: Priority of internal Interrupt Service Thread

4.3.3.7 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)

4.3.3.8 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]

4.3.3.9 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)

4.3.3.10 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)

4.3.3.11 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)

4.3.3.12 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)

4.3.3.13 BOOL GpioClearIntrPin(LPGPIO pGpio)

Clears 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]

4.3.3.14 BOOL GpioClearIntrPort(LPGPIO pGpio)

Clears 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)

4.3.3.15 BOOL GpioRegisterEvent(LPGPIO_EVENT pGpioEvent)

Registers an event to be signalled when the GPIO pin generates an interrupt
In: Port, Pin, Event Name, Priority
Out: N/A
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]

4.3.3.16 BOOL GpioDeregisterEvent(LPGPIO_EVENT pGpioEvent)

Deregisters a previously registered event on a GPIO pin
In: Port, Pin
Out: N/A
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]

4.3.3.17 BOOL GpioGetEvent(LPGPIO_EVENT pGpioEvent)

Returns the name of a registered event on a GPIO pin (if any)
In: Port, Pin and an allocated buffer of MAX_PATH WCHARs in pszEvent
Out: Event name (in pszEvent)
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]

4.3.4 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;
      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;
}
```

4.3.5 Topaz PWM SDK

GuruCE has added an SDK header and library for the PWM module as well. SDK access is through pwmsdk.h and link with pwmsdk.lib:

```
typedef struct
{
    UINT32 PWM_CR;
    UINT32 PWM_SR;
    UINT32 PWM_IR;
    UINT32 PWM_SAR;
    UINT32 PWM_PR;
    UINT32 PWM_CNR;
} PWM_REGS, *LPPWM_REGS;

// PwmInit
// Must be called before anything else
// pwmIndex: 1..4
// ret: FALSE on error (GetLastError() will return error code)
BOOL PwmInit(DWORD pwmIndex);

// PwmDeinit
// Must be called when PWM is no longer needed (cleans up handles etc)
// pwmIndex: 1..4
// ret: FALSE on error (GetLastError() will return error code)
BOOL PwmDeinit(DWORD pwmIndex);

// PwmStart/PwmStartDuration
// Starts the PWM, optionally for a duration in mS.
// pwmIndex: 1..4
// Frequency: Required frequency
// DutyCycle: Required dutycycle (in percent)
// Duration: Duration of output signal in mS.
// ret: FALSE on error (GetLastError() will return error code)
BOOL PwmStart(DWORD pwmIndex, DWORD Frequency, DWORD DutyCycle);
BOOL PwmStartDuration(DWORD pwmIndex, DWORD Frequency, DWORD DutyCycle, DWORD
Duration);

// PwmStop
// Stops the PWM
// pwmIndex: 1..4
// ret: FALSE on error (GetLastError() will return error code)
BOOL PwmStop(DWORD pwmIndex);

// PwmReset
// Resets the PWM
// pwmIndex: 1..4
// ret: FALSE on error (GetLastError() will return error code)
BOOL PwmReset(DWORD pwmIndex);

// PwmReadRegisters
// Gets the PWM registers
// pwmIndex: 1..4
// pPwmRegisters: pointer to PWM_REGS structure to receive the register values
// ret: FALSE on error (GetLastError() will return error code)
BOOL PwmReadRegisters(DWORD pwmIndex, LPPWM_REGS pPwmRegisters);
```

4.3.5.1 PWM Example

```
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPTSTR     lpCmdLine,
                   int       nCmdShow)
{
    DWORD pwmIndex = 1;
    if (wcslen(lpCmdLine)>0)
        pwmIndex = _wtol(lpCmdLine);
    wprintf(L"PwmTest(%d)\r\n", pwmIndex);
    if ((pwmIndex > 0) && (pwmIndex <=4))
    {
        if (PwmInit(pwmIndex))
        {
            wprintf(L"10 kHz 50%% duty cycle on PWM%d for 5 seconds\r\n",
                                                                pwmIndex);
            PwmStart(pwmIndex, 10000, 50);
            Sleep(5000);

            PWM_REGS pwmRegs = {0};
            PwmReadRegisters(pwmIndex, &pwmRegs);
            wprintf(L"CR  = 0x%08X\r\n", pwmRegs.PWM_CR);
            wprintf(L"SR  = 0x%08X\r\n", pwmRegs.PWM_SR);
            wprintf(L"PR  = 0x%08X\r\n", pwmRegs.PWM_PR);
            wprintf(L"CNR = 0x%08X\r\n", pwmRegs.PWM_CNR);
            wprintf(L"SAR = 0x%08X\r\n", pwmRegs.PWM_SAR);
            wprintf(L"IR  = 0x%08X\r\n", pwmRegs.PWM_IR);

            wprintf(L"10 kHz 0..100%% duty cycle sweep x2 on PWM%d for 20 seconds "
                                    "(10 seconds per sweep)\r\n", pwmIndex);
            for (int j=0; j<2; j++)
            {
                for (int i=0; i<=100; i++)
                {
                    PwmStart(pwmIndex, 10000, i);
                    Sleep(100);
                }
            }

            PwmStop(pwmIndex);
            PwmDeinit(pwmIndex);
        }
        else
            wprintf(L"ERROR: PwmInit(%d) FAILED!\r\n", pwmIndex);
    }
    else
        wprintf(L"ERROR: pwmIndex (%d) must be 1..4\r\n", pwmIndex);

    return 0;
}
```

4.3.6 I²C SDK example

The Inter-Integrated Circuit (I^2C) module is responsible for handling the communication from the CPU to a device connected to the I^2C bus.

4.3.6.1 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 I^2C 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:

```
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 I2CSetFrequency(HANDLE hI2C, DWORD dwFreq);
BOOL I2CSetSelfAddr(HANDLE hI2C, BYTE bySelfAddr);
BOOL I2CGetSelfAddr(HANDLE hI2C, PBYTE pbySelfAddr);
BOOL I2CTransfer(HANDLE hI2C, PI2C_TRANSFER_BLOCK pI2CTransferBlock);
BOOL I2CReset(HANDLE hI2C);
BOOL I2CEnableSlave(HANDLE hI2C);
BOOL I2CDisableSlave(HANDLE hI2C);
BOOL I2CGetSlaveSize(HANDLE hI2C, PDWORD pdwSize);
BOOL I2CSetSlaveSize(HANDLE hI2C, DWORD dwSize);
BOOL I2CGetSlaveText(HANDLE hI2C, PBYTE pbyTextBuf, DWORD dwBufSize, PDWORD pdwTextLen);
BOOL I2CSetSlaveText(HANDLE hI2C, PBYTE pbyTextBuf, DWORD dwTextLen);
```

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 I^2C bus. The code below shows you how to communicate with this chip:

```
#include <i2cbus.h>

#define MMA7660FC_I2C_ADDRESS 0x4C

typedef enum : BYTE
{
   XOUT,
   YOUT,
   ZOUT,
   TILT,
   SRST,
   SPCNT,
   INTSU,
   MODE,
   SR,
   PDET,
   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)\r\n", 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)\r\n", 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 _tmain(int argc, _TCHAR* argv[])
{
   // Open the I2C driver
   HANDLE hI2C = I2COpenHandle(L"I2C1:");
   if (INVALID_HANDLE_VALUE != hI2C)
   {
        // 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 I^2C SDK library.

4.3.7 FlexCAN Driver

The standard BSP and image binaries include the standard FlexCAN driver and SDK header for the 2 FlexCAN ports of the iMX25. GuruCE also offers a high performance FlexCAN driver supporting many more configuration options and making use of the FlexCAN receive FIFO for uninterrupted high speed reception of messages on the CAN bus. It also supports complex filtering schemes so you can block reception of unwanted messages. The High Performance FlexCAN driver supports all features offered by the iMX25 FlexCAN module. Please contact GuruCE for pricing information.

FlexCAN Driver Comparison Chart

	Standard FlexCAN Driver	High Performance FlexCAN Driver
Configure bitrate in registry	√	√
Set/Get bitrate (runtime)	√	√
Manual fine-tuning of CAN timing parameters and clock frequency	×	√
Configure IST priority in registry	√	√
Send a single CAN message	√	√
Receive a single CAN message	√	√
Send/receive multiple CAN messages in one call	√	√
High speed reception & buffering of an unlimited amount of messages	×	√
Configure “Receive Own” in registry	×	√
Enable/Disable “Receive Own” (runtime)	×	√
Configure Rx FIFO (Enable/Receive All/FIFO Size) in registry	×	√
Enable/Disable Rx FIFO (runtime)	×	√
Set/Get Rx FIFO ID filtering (runtime)	×	√
Configure “Listen Only” mode in registry	×	√
Enable/Disable “Listen Only” mode (runtime)	×	√
Configure oversampling in registry	×	√
Enable/Disable oversampling (runtime)	×	√
Configure loopback mode in registry	×	√
Enable/Disable loopback mode (runtime)	×	√
Configure Timer Synchronization in registry	×	√
Enable/Disable Timer Synchronization (runtime)	×	√

4.3.8 FlexCAN SDK

The FlexCAN SDK headers allow easy use of the CAN bus. The SDK handles all underlying communications with the FlexCAN driver. Usage documentation is included in the flexcansdk.h header file:

```
//-------------
// FlexCANInit
//-------------
// Must be called before anything else.
//
// Input:
//      dwIndex         - CAN instance index (1-based)
// Return:
//      FALSE on error (GetLastError() will return error code)
BOOL FlexCANInit(DWORD dwIndex);

//---------------
// FlexCANDeinit
//---------------
// Must be called when the FlexCAN is no longer needed (cleans up handles etc).
//
// Input:
//      dwIndex         - CAN instance index (1-based)
// Return:
//      FALSE on error (GetLastError() will return error code)
BOOL FlexCANDeinit(DWORD dwIndex);

//-----------------
// FlexCANTransfer
//-----------------
// Transfer multiple packets (either read or write).
//
// Input:
//      dwIndex         - CAN instance index (1-based)
```

```
//      pTransferBlock  - Pointer to CAN_TRANSFER_BLOCK structure containing
//                        an array of CAN_PACKETs:
//          CAN_PACKET.btIndex      - CAN Bus Message Buffer Index to use for this
//                                    transmission. Set to 0xFF to let driver find a free
//                                    message buffer
//          CAN_PACKET.bWrite       - This transmission is read (CAN_READ) or write
//                                    (CAN_WRITE)
//          CAN_PACKET.bRemote      - Request remote frame (CAN_REMOTE) or send data
//                                    (CAN_DATA)
//          CAN_PACKET.canFrame     - The actual frame data. See FlexCANWriteFrame for a
//                                    description of the CAN_FRAME items
//          CAN_PACKET.bTxOnRemoteRx- CAN_TXNOW to send immediately, CAN_TXONREMOTERX to send
//                                    when a frame with the same ID is received. Only valid
//                                    when (CAN_WRITE && CAN_DATA)
//          CAN_PACKET.btPriority   - The transmission priority (0 = highest priority, 7 =
//                                    lowest priority)
//          CAN_PACKET.ui32IDMask   - Bitmask for ID (when CAN_READ or CAN_REMOTE or
//                                    CAN_TXONREMOTERX): 0; don't care, 1; the corresponding
//                                    bit in the ID is checked against the one received
//          CAN_PACKET.dwTimeout    - Maximum time this transmission can take in ms
//          CAN_PACKET.dwError      - Receives the result of the action on this packet
//                                    (ERROR_SUCCESS == All ok)
// Output:
//          CAN_PACKET.canFrame     - See FlexCANWriteFrame for a description of the output
//                                    when CAN_WRITE and see FlexCANReadFrame for a
//                                    description of the output when CAN_READ
// Return:
//      FALSE on error (GetLastError() will return error code)
//      The dwError item of each CAN_PACKET will contain the transfer result per packet (see
//      FlexCANWriteFrame for a list of possible errors)
BOOL FlexCANTransfer(DWORD dwIndex, PCAN_TRANSFER_BLOCK pTransferBlock);

//-------------------
// FlexCANWriteFrame
//-------------------
// Write a single frame over the CAN bus.
//
// Input:
//      dwIndex         - CAN instance index (1-based)
//      bRemote         - CAN_DATA to send a date frame, CAN_REMOTE to request a remote frame
//      bTxOnRemoteRx   - CAN_TXNOW to send immediately, CAN_TXONREMOTERX to send when a frame
//                        with the same ID is received
//      btPriority      - Priority to use for transmitting this frame (0 = highest priority, 7
//                        = lowest priority)
//      pCanFrame       - Pointer to a CAN_FRAME:
//          CAN_FRAME.bExtended     - Frame format is standard (CAN_STANDARD) or extended
//                                    (CAN_EXTENDED)
//          CAN_FRAME.ui32ID        - Message ID. If using standard frame format the ID is 11
//                                    bits, if using the extended format the ID is 29 bits
//          CAN_FRAME.btDataLen     - Number of valid bytes in btData
//          CAN_FRAME.btData[0..8]  - Message data, max 8 bytes
//      ui32IDMask      - Bitmask for ID (when CAN_REMOTE or CAN_TXONREMOTERX): 0; don't care,
//                        1; the corresponding bit in the ID is checked against the one
//                        received
//      dwTimeout       - Maximum time this transmission can take in ms
// Output:
//          CAN_FRAME.ui16Timestamp - Receives the transmission timestamp (captured when the
//                                    frame first appeared on the bus)
//      if CAN_REMOTE:
//          CAN_FRAME.bExtended     - Frame format is standard (CAN_STANDARD) or extended
//                                   (CAN_EXTENDED)
//          CAN_FRAME.ui32ID        - Message ID. If standard frame format the ID is 11 bits,
//                                    if using the extended format the ID is 29 bits
//          CAN_FRAME.ui16Timestamp - Receives the transmission timestamp (captured when the
//                                    frame first appeared on the bus)
//          CAN_FRAME.btDataLen     - Number of valid bytes in btData
//          CAN_FRAME.btData[0..8]  - Message data, max 8 bytes
// Return:
//      ERROR_SUCCESS           - Successful
//      ERROR_INVALID_PARAMETER - Parameter error
//      ERROR_INVALID_DATA      - Received packet (when bRemote == CAN_REMOTE) contained
//                                invalid data
//      ERROR_TIMEOUT           - Timeout while sending (or receiving when bRemote ==
//                                CAN_REMOTE)
```

```
DWORD FlexCANWriteFrame(DWORD dwIndex, BOOL bRemote, BOOL bTxOnRemoteRx, BYTE btPriority,
PCAN_FRAME pCanFrame, DWORD dwTimeout, UINT32 ui32IDMask=0);

//------------------
// FlexCANReadFrame
//------------------
// Read a single frame from the CAN bus.
//
// Input:
//      dwIndex         - CAN instance index (1-based)
//      pCanFrame       - Pointer to a CAN_FRAME:
//          CAN_FRAME.bExtended     - Frame format is standard (CAN_STANDARD) or extended
//                                    (CAN_EXTENDED)
//          CAN_FRAME.ui32ID        - Message ID. If using standard frame format the ID is 11
//                                    bits, if using the extended format the ID is 29 bits
//      dwTimeout       - Maximum time this transmission can take in ms
//      ui32IDMask      - Bitmask for ID: 0; don't care, 1; the corresponding bit in the ID is
//                        checked against the one received
// Output:
//          CAN_FRAME.bExtended     - Frame format is standard (CAN_STANDARD) or extended
//                                    (CAN_EXTENDED)
//          CAN_FRAME.ui32ID        - Message ID. If standard frame format the ID is 11 bits,
//                                    if using the extended format the ID is 29 bits
//          CAN_FRAME.ui16Timestamp - Receives the transmission timestamp (captured when the
//                                    frame first appeared on the bus)
//          CAN_FRAME.btDataLen     - Number of valid bytes in btData
//          CAN_FRAME.btData[0..8]  - Message data, max 8 bytes
// Return:
//      ERROR_SUCCESS           - Successful
//      ERROR_INVALID_PARAMETER - Parameter error
//      ERROR_INVALID_DATA      - Received packet (when bRemote == CAN_REMOTE) contained
//                                invalid data
//      ERROR_TIMEOUT           - Timeout while sending (or receiving when bRemote ==
//                                CAN_REMOTE)
DWORD FlexCANReadFrame(DWORD dwIndex, PCAN_FRAME pCanFrame, DWORD dwTimeout, UINT32
ui32IDMask);

//-------------------
// FlexCANSetBitrate
//-------------------
// Set the FlexCAN bus bitrate.
//
// Input:
//      dwIndex         - CAN instance index (1-based)
//      pdwBitrate      - Pointer to a DWORD containing the requested bitrate (in bps)
// Output:
//      pdwBitrate      - Contains the actual bitrate set (this can differ due to prescaler
//                        roundings)
// Return:
//      FALSE on error (GetLastError() will return error code)
BOOL FlexCANSetBitrate(DWORD dwIndex, LPDWORD pdwBitrate);

//-------------------
// FlexCANGetBitrate
//-------------------
// Get the FlexCAN bus bitrate.
//
// Input:
//      dwIndex         - CAN instance index (1-based)
//      pdwBitrate      - Pointer to a DWORD
// Output:
//      pdwBitrate      - Contains the current bitrate
// Return:
//      FALSE on error (GetLastError() will return error code)
BOOL FlexCANGetBitrate(DWORD dwIndex, LPDWORD pdwBitrate);
```

4.3.9 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
- SPI
  SPI Bus: cspibus.h, spisdk.lib
- GPT
  General Purpose Timers: gpt.h, gptsdk.lib

The exported functions of these libraries are easily P/Invoked from managed code as well. GuruCE has managed (C#) wrappers available for GPIO, SPI, I2C, PWM and CAN. Please contact GuruCE for pricing information.

5 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.

5.1 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:

Baud rate	115200
Data bits	8 bits
Parity	None
Stop	1 bit
Flow control	None

Table 1 - Serial connection settings

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

```
Microsoft Windows CE Bootloader Common Library Version 1.4 Built Apr 13 2010 13:48:31
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.

5.2 Bootloader Menu

The Topaz i.MX24 bootloader menu:

```
------------------------------------------------------------------------------
Topaz i.MX25 Boot Menu
------------------------------------------------------------------------------
 [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 : 0-50-C2-3F-90-6A
 [4] DHCP : Enabled
 [5] Boot Delay : 3
 [6] Select Boot Device : NK from NAND
 [7] Reset to Factory Default Configuration
 [8] Format OS NAND Region
 [9] Format All NAND Regions
 [C] Clean registry & databases : Disabled
 [B] Bootloader Shell
 [W] KITL Work Mode : Interrupt
 [K] KITL Enable Mode : Disabled
 [P] KITL Passive Mode : Disabled
 [E] Select Ether Device : FEC
 [U] Select Windows CE Debug UART : Disabled
 [D] Download Image Now
 [L] Launch Existing Flash Resident Image Now
 [M] MMC and SD Utilities
 [R] Reset
 [S] Save Settings

 Selection:
```

5.2.1 [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.

5.2.2 [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.

5.2.3 [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)

5.2.4 [7] Reset to Factory Default Configuration

This will restore the original boot loader settings as they were when the device was shipped.

5.2.5 [8] Format OS NAND Region

This option will erase the Windows CE image from flash.

5.2.6 [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!

5.2.7 [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.

5.2.8 [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 --
```

5.2.9 [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.

5.2.10 [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:
http://msdn.microsoft.com/en-us/library/ms898144.aspx

5.2.11 [E] Select Ether Device

This option allows you to select the Ethernet transport. FEC is the Ethernet MAC. Other options are USB Serial and USB RNDIS.

5.2.12 [U] Select Windows CE Debug UART

This option allows you to select the serial port used for debug logging in Windows CE. Default is disabled. Other options are UART1, UART2 and UART3.

5.2.13 [D] Download Image Now

This option will initialize the selected ether device and start the image download.

5.2.14 [L] Launch Existing Flash Resident Image Now

Selecting 'L' will launch the image which is stored into NAND.

5.2.15 [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.

5.2.16 [R] Reset

This option will (warm) reset the Topaz

5.2.17 [S] Save Settings

The "Save Settings" option writes the menu settings to NAND flash.

5.3 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.

5.3.1 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
[4] DHCP : Enabled
…
[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:

```
BL_IMAGE_TYPE_BIN

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: Updating of image completed successfully.
Reboot the device manually...
SpinForever...
Do you want to reset [Y\N]
```

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.

6 About us

6.1 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.

6.2 Blog

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

6.3 Support options

GuruCE offers various support options. Please visit http://guruce.com/support for more information.

GuruCE APAC/NZ

Contact : Michel Verhagen
Email : michel@guruce.com
Phone : +64 (0)7 929 5807
Mobile : +64 (0)21 104 6208

240 Ohiwa Harbour Road
RD2, Opotiki, 3198
New Zealand

GuruCE EMEA/NL

Contact : Erwin Zwart
Email : erwin@guruce.com
Phone : +31 (0)728 503 119
Mobile : +31 (0)629 512 116

Tuin van Halo 19
Heerhugowaard, 1705 TD
The Netherlands