Windows Embedded Compact 2013

Getting Started using 86Duino EduCake

Date: April 19th, 2015
By: Samuel Phung
Twitter: @Samuelp101
Blog: http://Embedded101.com/Samuelp101

Table of Contents

Part-1: Development Environment & Tools

In part-1 of this Compact 2013 getting started series, let's go over the development environment, required software and connectivity between the development workstation and the target device and cover the following:

- Typical development tasks
- Required software
- Software Installation
- Dedevelopment environment
- Target device
- Compact 2013 terminology

Typical development tasks

Regardless of the development environment we ended up with, it's necessary for us to know about the various development tasks involve in process to create the device.

After hardware platform is decided, developing a Compact 2013 device typically involves the following:

1. Board support package (BSP) for the device, which include hardware adaptation code, device drivers and bootloader.
2. Operating system image for the device (OS design).
3. SDK to support application development.
4. Testing and debugging
5. Deploy OS image along with application onto target device for distribution

Note:

This series of application note is intended for beginner and intermediate developer with limited knowledge about the Compact 2013 development environment, and will not cover BSP, device driver, bootloader and other advanced subject.

Requirement Software

The following software components are needed to support Compact 2013 development:

- Visual Studio 2013 (VS 2013) or Visual Studio 2012 (VS 2012) with Update 4

  Note:

  The express version of VS 2012 and 2013 does not support Compact 2013 development.

  While the Visual Studio express version does not support Compact 2013 development, Microsoft released another version, Visual Studio 2013 Community edition, available for free that support Compact 2013 development. Visit the following URL for information and download link:

http://www.visualstudio.com/en-us/news/vs2013-community-vs.aspx

- Application Builder for Windows Embedded Compact 2013

  Application Builder is needed, along with an SDK for the target device to support application development for Compact 2013. Visit the following URL to download the software:

  http://www.microsoft.com/en-us/download/details.aspx?id=38819

- Windows Embedded Compact 2013 Platform Builder

  Platform Builder is the main development tool for Compact 2013, to develop OS design, device driver, BSP, native application, testing and debugging.

  From the following URL, look for Windows Embedded Compact 2013. You will need a valid Microsoft ID to register and get an activation key. While the site indicate it's a trial version, it is a full featured and fully functional version.

  http://www.microsoft.com/windowsembedded/en-us/downloads.aspx

- SDK for target device

  To develop application for a Compact 2013 device, you need a Compact 2013 SDK for the target device. You can generate the SDK from within the OS design project used to develop the OS runtime image for the target device. The target device's vendor may have an existing SDK for Compact 2013 to support their product.

Software Installation

Improper software installation is one of the major source of problem for developer new to Compact 2013. It's a complex development environment that involve huge amount of files. It's well worth the effort to take time and carefully install all of the required component.

VS 2013 Ultimate is used to perform the sample exercises in this getting started series.

> ***Note:***
> *While we work through the sample exercises using VS 2013, the same BSP, 3rd party components and sample codes work in the VS 2012 development environment.*

Here is the recommended installation sequence:

1. Visual Studio 2013.
2. Application Builder for Windows Embedded Compact 2013.

   There are two different version of Application Builder, one for VS 2013 and the other for VS 2012. Be sure to install the correct version.

3. Windows Embedded Compact 2013 Platform Builder.

In the event you have the Windows Embedded Compact 2013 installation software or DISC from the initial release (mid 2013), it was created to support VS 2012 and may have problem to install on a VS 2013 development machine.

If you have MSDN subscription, use “Windows Embedded Compact 2013 Update 5”, this update package is released as a full product installation that includes all of the QFEs up until Update 5.

If you are getting the installation package from the Windows Embedded product trial website, it’s an updated installation package that support VS 2013.

During the Compact 2013 Platform Builder installation process, you can chose to skip update. Otherwise, it will take a lot more time to check and download additional updates, as shown in the following installation screen (you can perform the update at a later time):

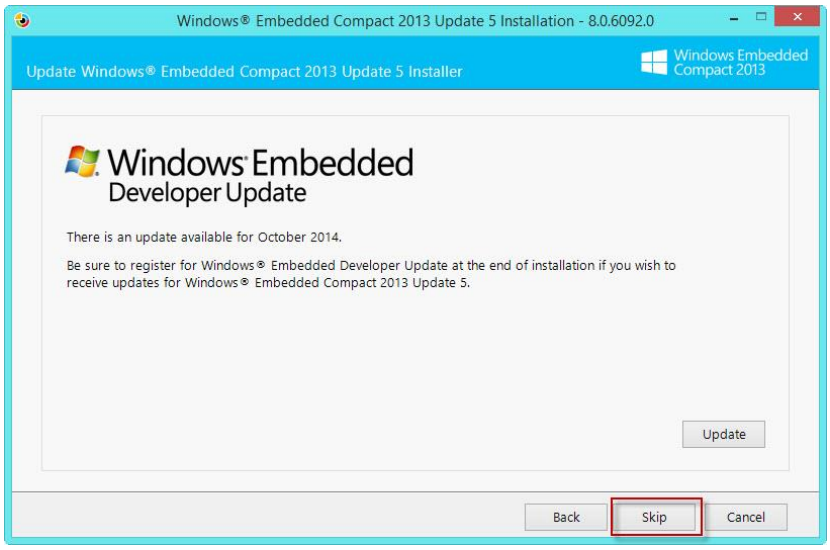

Fig-1.1: Compact 2013 installation

The “Full install” will occupy around 25 GB of storage. You can select the “Custom install” option and select the components needed to support the target device you plan to use, to minimize the required disk storage.

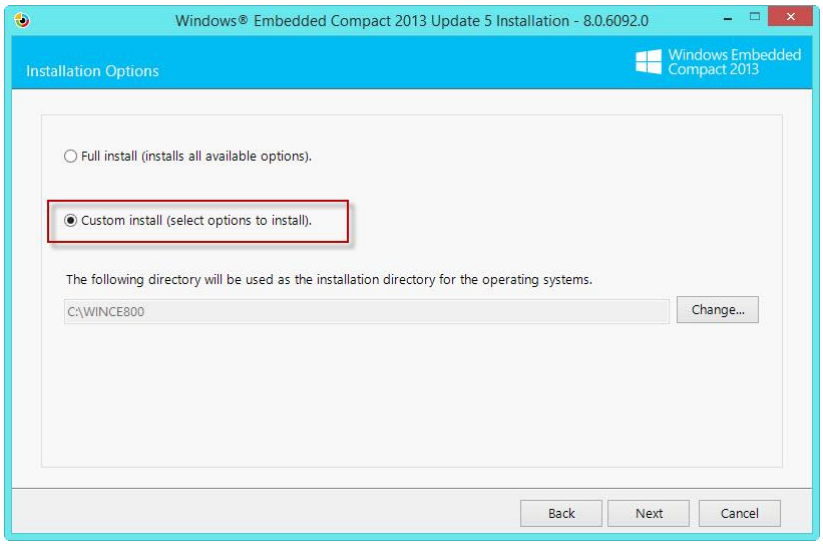

Fig-1.2: Compact 2013 installation

Since we are using an x86 device for the exercise in this series, we excluded ARM CPU support from the installation, as show in the following screen:

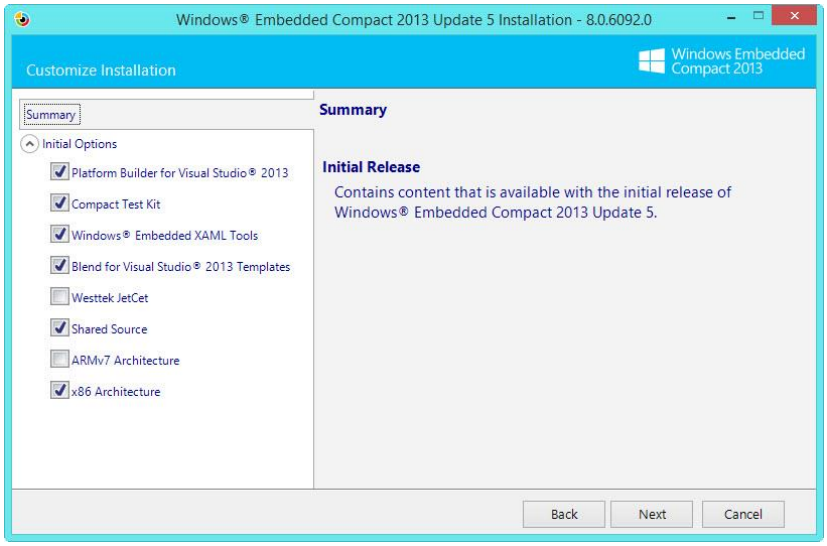

Fig-1.3: Compact 2013 installation

When the “offline layout” option is selected, a copy of the installation files are saved on the development machine’s local hard drive, which can be helpful later.

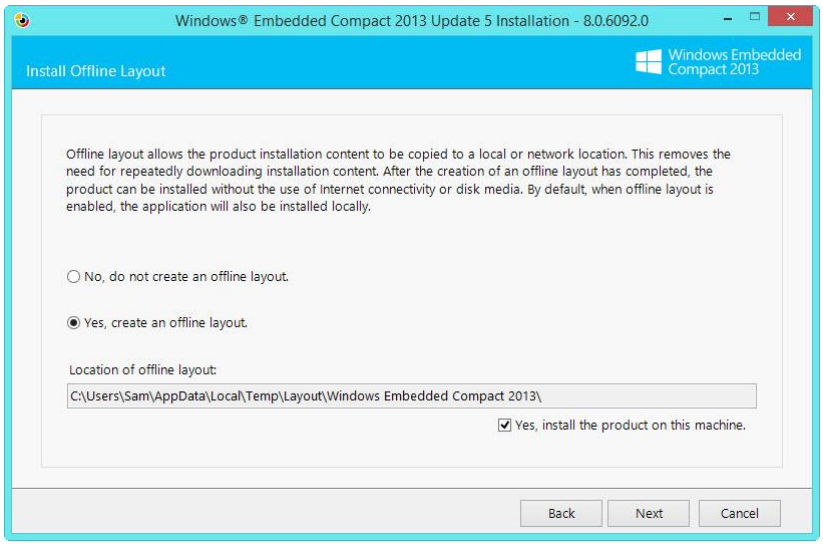

Fig-1.4: Compact 2013 installation

4. Next, install 3rd party BSP, device drivers and SDK needed to support the target device.

   Third party BSPs are installed to the following directory, assuming you install Compact 2013 to the default directory:

```
C:\WINCE800\Platform
```

   Third party components are installed to the following directory:

```
C:\WINCE800\3rdParty
```

Development Environment

In addition to properly install all of the required software, a proper development environment is another important aspect you need to be aware of.

Depending on the target device you are using, the setup may be different.

For the exercises in this getting started series, we are using an x86 target device. Both the development workstation and the target device are attached to the same Local Area Network (LAN) with DHCP service to provide IP address dynamically. In addition, a null RS-232 serial cable is used to connect a serial port on the target device to the development workstation, to capture serial debug messages.

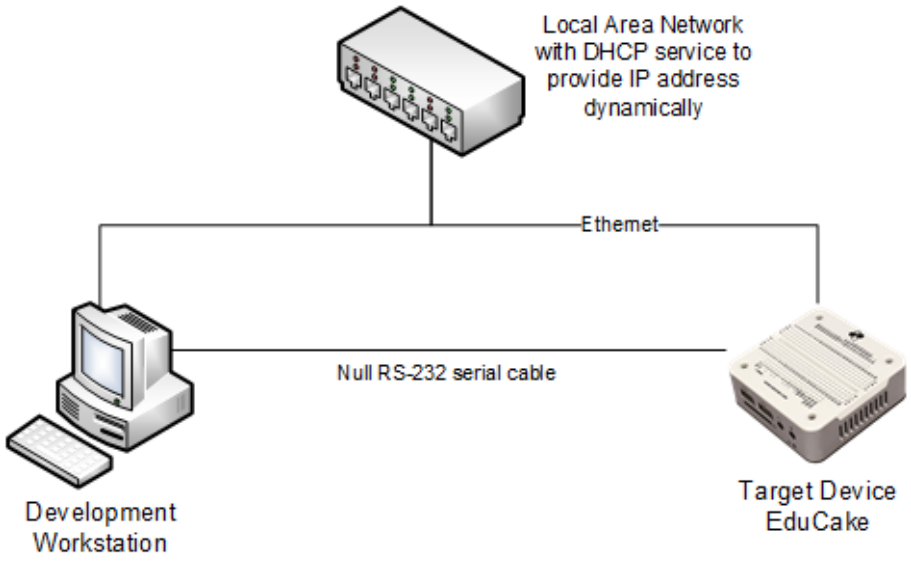

Fig-1.5: Development environment

The following figure shows the development flow for Windows Embedded Compact 2013:

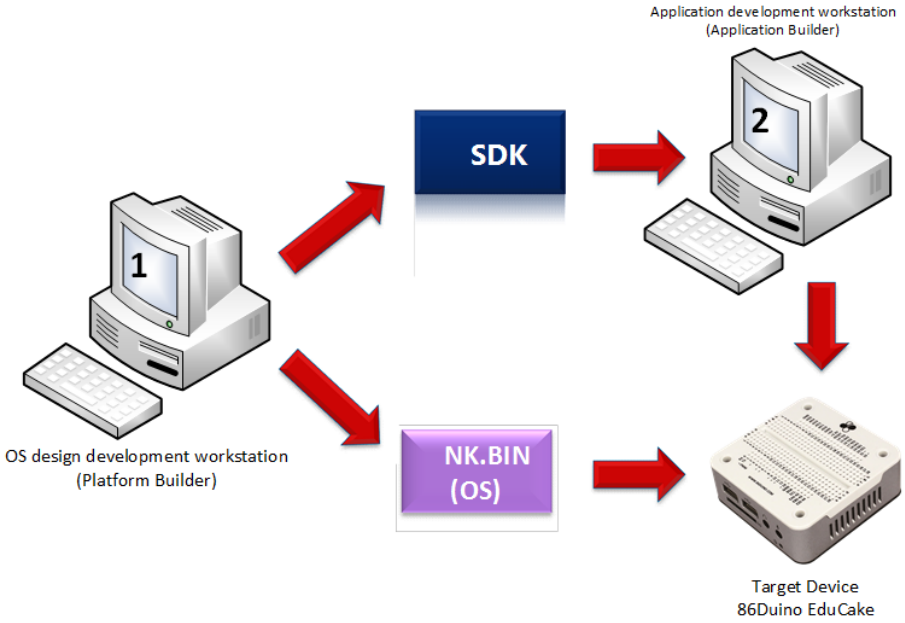

Fig-1.6: Development environment

In the above figure, workstation #1 is used to develop an OS design project, to generate and OS runtime image (NK.bin) for the target device. SDK is generated from the OS design project in workstation #1, which is needed by workstation #2 to develop application for the target device.

Note:
The same development workstation can be used to develop both the OS runtime image and application for a Compact 2013 device. In real life situation, it's common to separate and assign OS image development and application development to different developers.

Target Device

The exercise in this series is based on a target device with board support package (BSP) and bootloader available, 86Duino EduCake.

The 86Duino EduCake (EduCake) is designed with a 300 MHz Vortex86EX System-on-Chip (SoC), originally designed to emulate the Arduino platform. The I/O interfaces accessible on the EduCake's solderless breadboard is electronically compatible to the Arduino Leonardo and Arduino Uno, which is also referred to as Arduino 1.0 pinout, as shown in the following figure.

Fig-1.7: Target device - 86Duino EduCake

The EduCake, packaged in a metallic enclosure with an integrated solderless breadboard, is designed to provide an easy to use platform for academic and hobbyist developers to work with different experimental circuit, to explore and learn.

While it was originally designed to target the Arduino user community, the EduCake is capable to boot to DOS, Linux, Windows CE, Windows Embedded Compact, Windows XP, Windows XP Embedded, Windows Embedded Standard 2009 and other RTOS that support the typical PC built on 32-bit x86 processor.

For the exercise in this series, we will use the BSP, SDK and other resources from the 86Duino project on Codeplex, which is available via the following URL:

http://86duino.codeplex.com

For more information about the EduCake, refer to the following URL:

http://www.86duino.com/index.php?p=95

Compact 2013 Terminology

It's common practice for many industry to have its own set of common terminology. Knowing the terminology and what it represent is helpful, especially when you are new to the environment trying to learn the new environment. The following table is a list of some of the common terminology in the Compact 2013 environment.

Terminology	Description
BSP	Board support package is a set of software components that include device drivers and OEM adaptation layer codes for the supported target device.
Catalog	Catalog contains components for Compact 2013 OS such as OS features, modules, device drivers, BSP and application components.
OAL	OEM Adaptation Layer is a low-level code acting as the interface between the OS and the hardware.
OS design	A Visual Studio project to develop and generate a custom Compact 2013 OS runtime image for a target device.
OS runtime image	The binary image file generated from an OS design project.
Target device	The hardware platform used to develop Compact 2013 OS or application.
KITL	Kernel Independent Transport Layer is a communication protocol used for debugging in the Compact 2013 development environment.
CoreCon	Short for Core connectivity, used to establish connectivity between the development station and target device.
Release directory	Refers to the directory where the OS design project output files and software components during the build and compilation process.
Build release directory	same as Release directory

Part-2: OS Design Development

In part-2 of this Compact 2013 getting started series, we will work through the steps to develop an OS design, generate an OS runtime image, download and launch the image onto a target device (86Duino EduCake) for testing and debugging.

For information about the development environment, please refer to part-1 of this series (development environment and tools), via the following URL:

http://www.embedded101.com/Blogs/SamuelPhung/entryid/577/compact-2013-development-environment-tools

The board support package (BSP) for the 86Duino EduCake (86Duino_80B), used to create the OS design, is available on Codeplex:

http://86duino.codeplex.com/

New OS Design Project

Let's work through the following steps to create a new OS design project:

VS IDE resolution: 1000 x 720

5. From Visual Studio 2013 (VS2013) IDE, select **File→New→Project** to bring up the New Project wizard.
6. From the left pane, select Platform Builder. Click to hightligh OS Design on the center pane and enter project name, MyOSDesign, as shown in Fig-2.1.

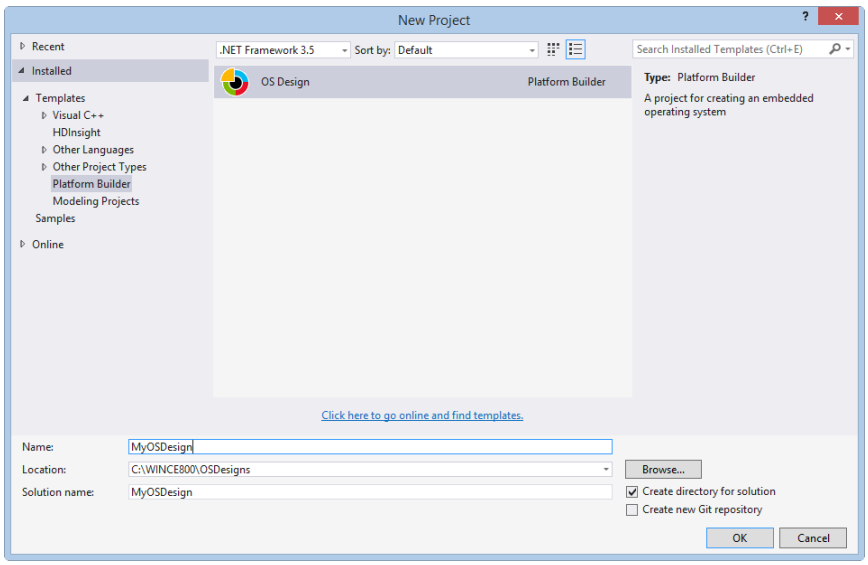

Fig-2.1: New OS Design project.

7. Click OK to launch OS Design Wizard.
8. Click Next to continue and launch the Board Support Packages selection screen. Select 86Duino_80B, as shown in Fig-2.2.

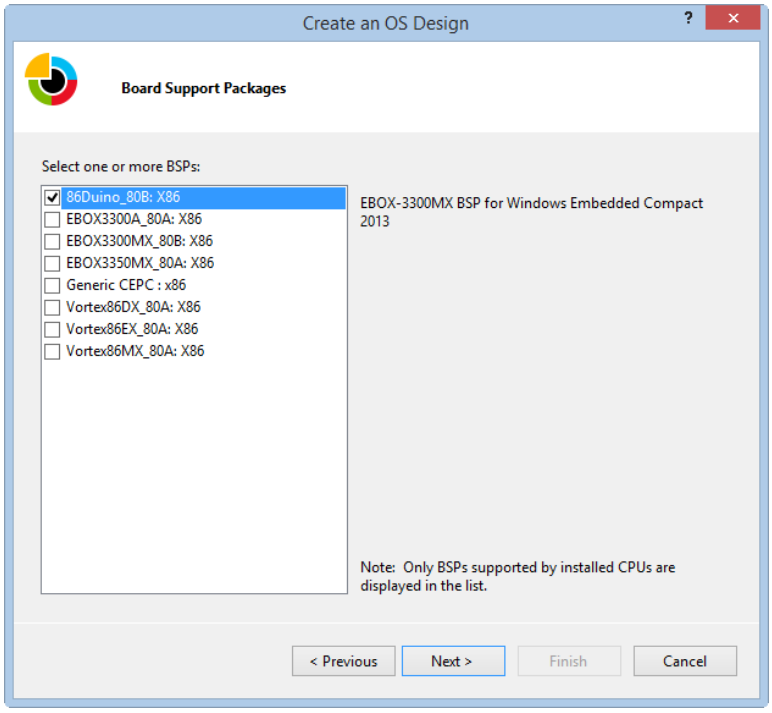

Fig-2.2: BSP selection

9. Click Next to continue and bring up the Design Templates selection screen. Select Headless Device, as shown in Fig-2.3.

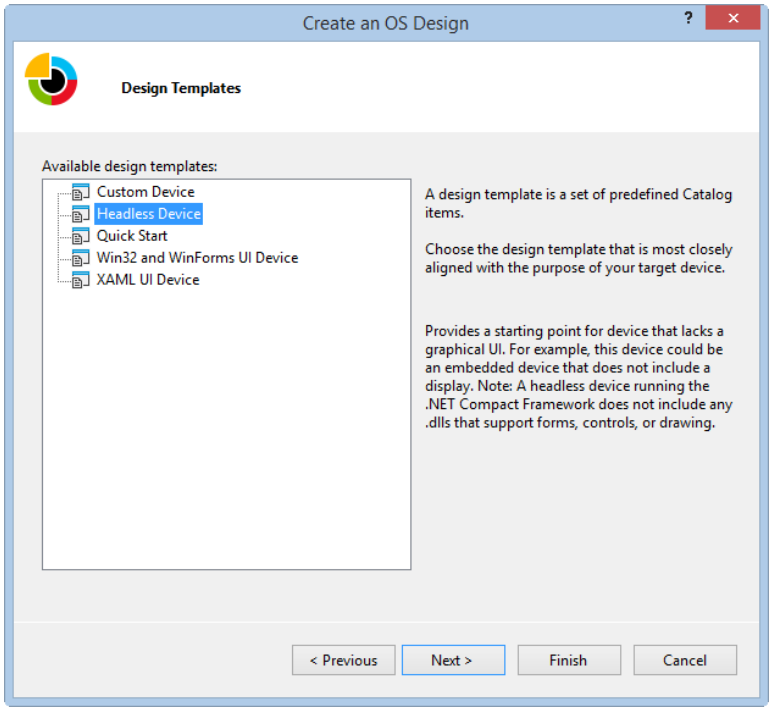

Fig-2.3: Design Templates selection

10. Click Next to continue and bring up the Headless Device OS components selection screen, as shown in Fig-2.4.

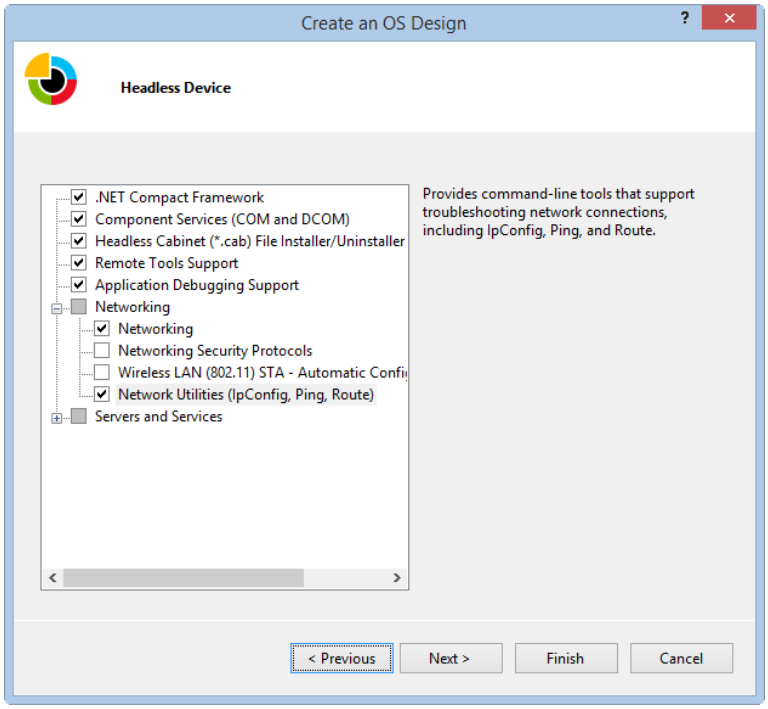

Fig-2.4: OS Components selection

11. Select components as shown in Fig-04 and click Finish to complete the OS Design wizard step.

At this point, VS2013 created the initial workspace for the new OS design project based on the BSP, design template and components selected. The following directories are created for the project:

- C:\WINCE800\OSDesigns\MyOSDesign

  This is the folder for the *MyOSDesign* solution. VS2013 supports different project types. A solution provides a centralized workspace to keep different project types supporting the same solution in one location.

  For example, the *MyOSDesign* solution may include the *MyOSDesign* OS design project, Visual C# managed code application project and “Visual C++ native code application project, all within the same VS2013 solution workspace.

The VS2013 IDE should look similar to the screen as shown in Fig-2.5.

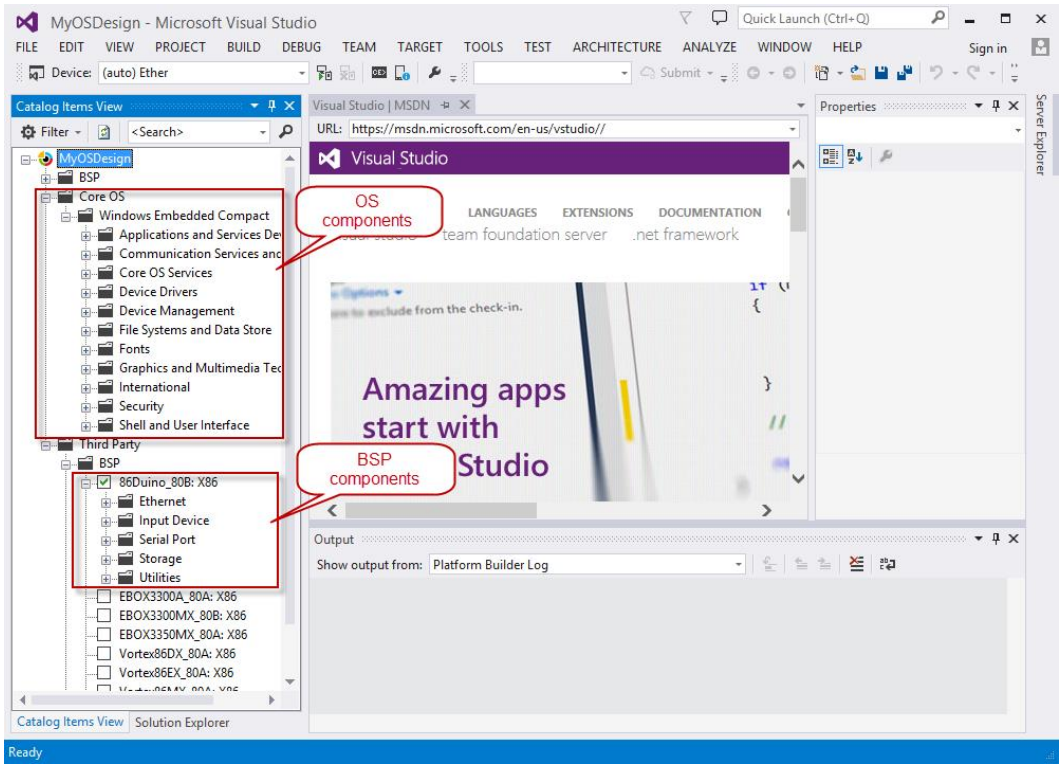

Fig-2.5: VS2013 IDE with MyOSDesign project active

Customize OSDesign: BSP Components

Next, to customize the OSDesign, we will work through the steps to select and add BSP components to MyOSDesign from the Catalog Items View window, as shown in Fig-2.5.

- Expand component folders and select components from the 86Duino_80B BSP, as shown in Fig-2.6.

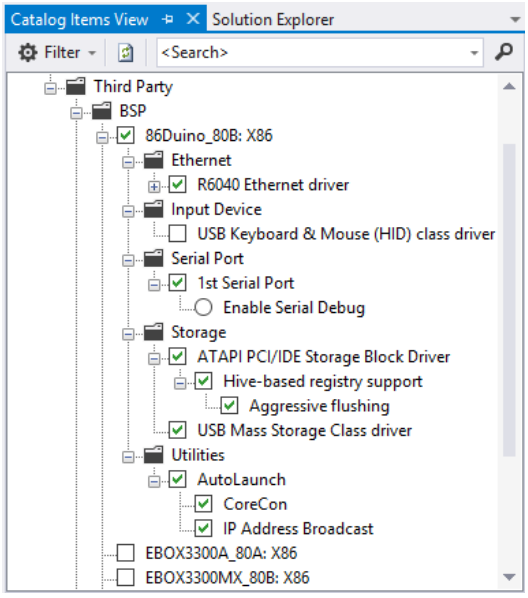

Fig-2.6: BSP components

- Select and include the following BSP components to the OS design:
  - R6040 Ethernet driver
  - 1st Serial Port
  - ATAPI PCI/IDE Storage Block Driver

    Driver to support IDE, SATA, Compact Flash, SD and Micro-SD storage.

  - Hive-based registry support

    The Hive-based registry component needed to save registry settings to non-volatile storage between power reset (cold boot).

  - Aggressive flushing

    Enable a background thread to actively flush and save changes to the registry.

  - USB Mass Storage Class driver

    This component set the SYSGEN_USB_STORAGE environment variable to include the USB storage class driver to support external USB storage devices, including USB flash and portable USB hard drive.

  - AutoLaunch

    Helper utility configurable to launch one or more application during startup.

  - CoreCon

    When this component is selected, CoreCon files and registry settings are added to the OS design, to support connectivity between VS2013 IDE and the target device to deploy VS2013 application to the device for testing and debugging.

  - IP Address Broadcast

    Since a headless device does not have any user interface to interrogate the device's IP address, needed by the VS2013 IDE to connect to the device, this component is configured to run during startup to broadcast the device's IP address, via UDP.

Note:

*The **IPAddressDiscovery.exe**, a Windows application to detect UDP broadcast messages from "IP Address Broadcast" and display the target device's IP address, is provided as part of the BSP, in the BSP's \Misc. directory. Launch this utility from the development machine before and have this application waiting for broadcast message before launching the OS design on the target device.*

Customize OSDesign: OS Components

In addition to the BSP components needed to support the target device's hardware, additional OS components are needed to support the device's application and function, as shown in Fig-2.7.

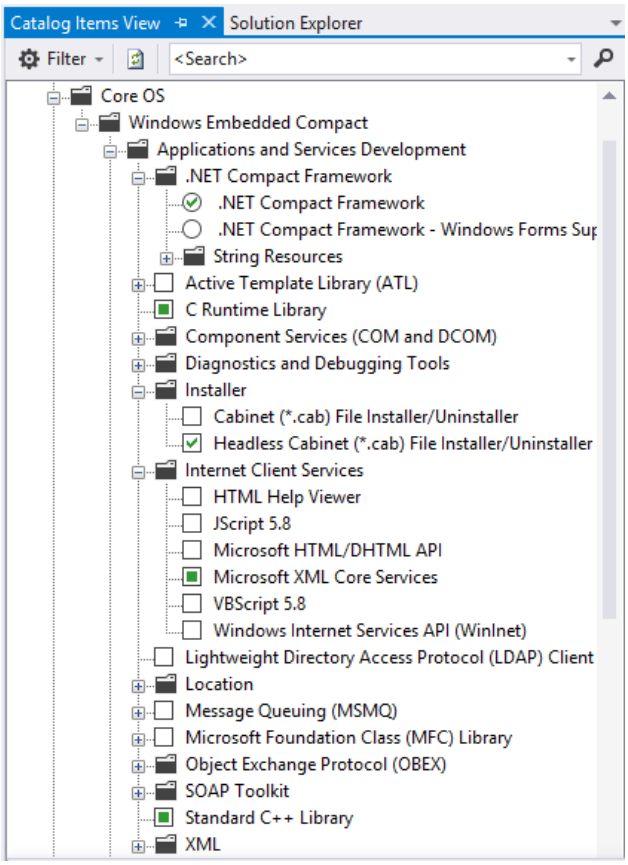

Fig-2.7: OS components

OS components are under the **\Core OS\Windows Embedded Compact** folder in the Catalog Items View window. During the steps to create the initial OS design workspace, the OS design wizard included a number of OS components, based on the selected design template, to the OS design.

Select and add the following OS components to the OS design:

- Display Driver Stub

  Compact 2013 OS, similar to the Windows OS for the desktop, is design to support rich user interface environment and is heavily dependent on GWES (Graphics, Windowing and Events Subsystem), even for headless configuration. The Display Driver Stub component is null display driver.

Build, Compile and Generate OS runtime image

With the required BSP and OS components added to the OS design project, it's time to build and generate the OS runtime image. From the VS2013 IDE, select **Build→Build Solution** to build the project.

Depending on the PC's performance, it may take anywhere from 10 to 20+ minutes to complete the build process. When the build process is completed, the VS2013 IDE's Output tab display the build result, as shown in Fig-2.8.

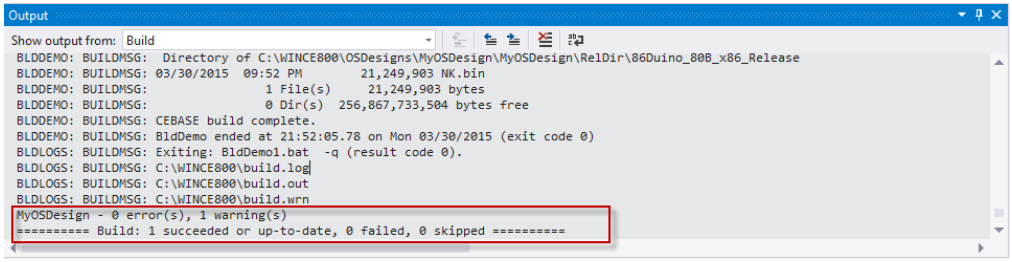

Fig-2.8: Build completed

When the build is successful, the build ends with zero error and generate an OS runtime image file, NK.bin, in the following build release directory:

```
C:\WINCE800\OSDesign\MyOSDesign\MyOSDesign\RelDir\86Duino_80B_x86_Release
```

When the build ends with 1 or more error, it does not generate an OS runtime image. Review the *build.log* and *build.err* files in the C:\WINCE800\ folder to analyze problem.

Summary

In part-2 of this series, we talk about developing an OS design, build, compile and generate an OS runtime image. In part-3, we will talk about establish connectivity and download OS runtime image to the target device.

Part-3: Download and Debug OS Runtime Image

In part-3 of this Compact 2013 getting started series, we will work through the steps to establish connectivity and download OS runtime image to the target device, an 86Duino EduCake (EduCake).

Target Device (EduCake) Preparation using DiskPrep

To establish connectivity between VS2013 IDE and EduCake and download OS runtime image, the EduCake needs to be configured with BIOSLoader to launch Eboot.bin (an Ethernet bootloader).

The EduCake can be configured to boot from USB and SD flash storage. Both USB and SD flash storage can be format and configured with BIOSLoader and Eboot.bin, using Diskprep.

Here are the steps to format and configure an SD flash storage (These same steps apply to USB flash storage):

- Insert the SD flash storage to the SD slot on the PC or use an SD-to-USB adapter.
- After the PC detected the SD flash storage, launch DiskPrep with elevated privilege.
- From DiskPrep program screen, select SD flash from Disk Selection, as shown in Fig-3.1.

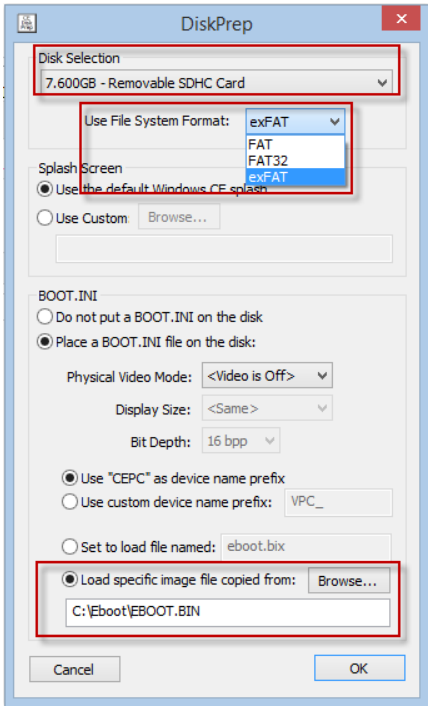

Fig-3.1: Diskprep

- Select file system.
- Click on Browse, next to the “Load specific image file copied from” option, and navigate to locate and select Eboot.bin.

(Eboot.bin is included in the 86Duino_80B BSP, in the \Misc folder)

- Click on OK to format and configure the SD flash storage.

Target Device for Compact 2013

To establish connectivity and download OS runtime image from VS2013 IDE to the target device, both the development PC and target device must be attached to the same Local Area Network, with IP addresses on the same subnet.

For the example here, the EduCake and development PC are both attached to the same Local Area Network with DHCP service to provide IP address dynamically, as shown in Fig-3.2.

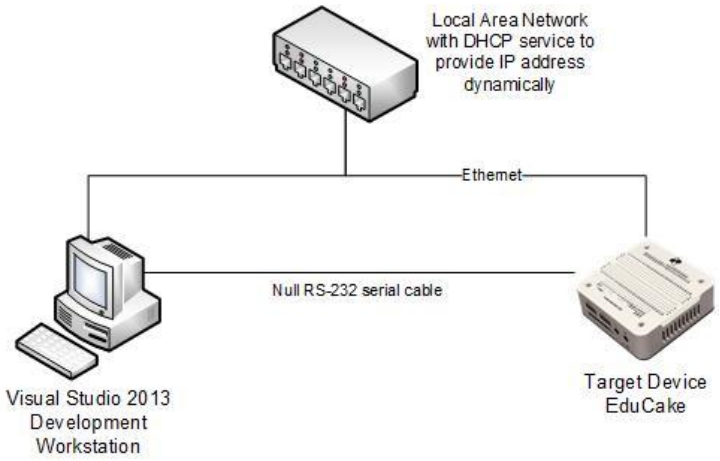

Fig-3.2: Development PC and EduCake attached to the same LAN

As part of the development process, we need to establish connectivity and download OS runtime image to the target device repeatedly for debugging and testing.

While we can select **TARGET | Attach Device** option from the VS2013 IDE to download OS image to the target device, without preconfigured device profile, this process goes through the steps to detect, identify and associate the device with the project for every download. By creating a target device profile, we can avoid the steps to detect and identify the device when downloading OS runtime image to the device.

Let's go through the following steps, using MyOSDesign project from part-2 of this series, to create a target device profile for the EduCake:

- With MyOSDesign project open, from the VS2013 IDE, select **TARGET | Connectivity Options** to bring up the Target Device Connectivity Options screen, as shown in Fig-3.3.

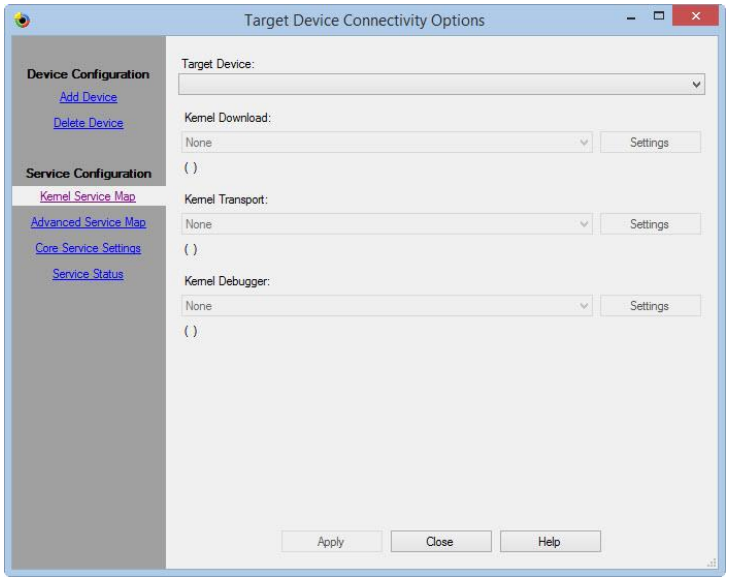

Fig-3.3: Target Device Connectivity

- Click on Add Device
- Enter EduCake as the name for the device and click **Add**.
- On the Target Device Connectivity Options screen, click on the top most Settings button, for Kernel Download, to bring up the Ethernet Download Settings screen, as shown in Fig-3.4.

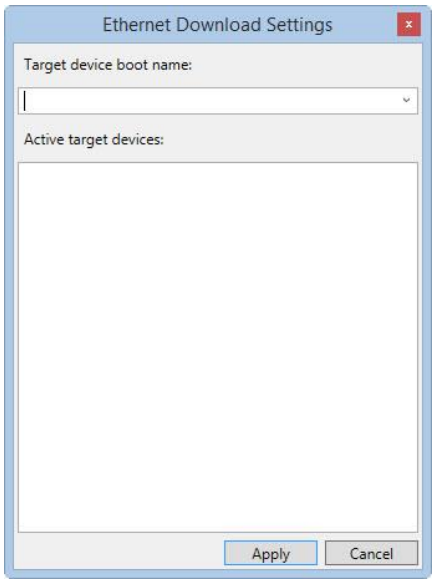

Fig-3.4: Ethernet Download Settings

- With the Ethernet Download Settings screen waiting for BOOTME messages from the target device, power on the EduCake.
- As BIOSLoader launches Eboot.bin on EduCake, it broadcasts a series of BOOTME messages, via UDP over the attached LAN.
- As the Ethernet Download Settings screen deteced the BOOTME message from EduCake, it display device ID from the EduCake, as shown in Fig-3.5.

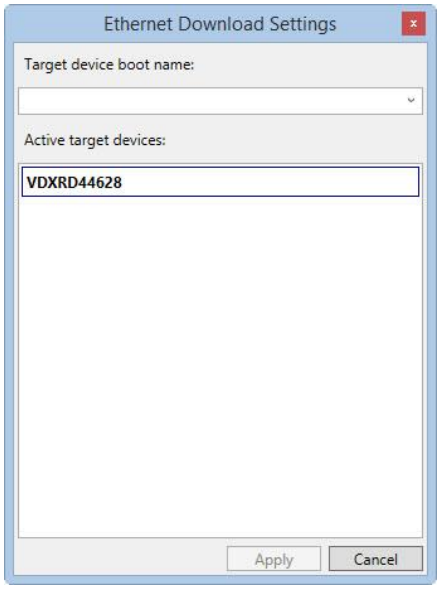

Fig-3.5: Target device detected

- From the Ethernet Download Settings screen, click on the device ID in the Active target device window to select the device and click **Apply**.
- From the Target Device Connectivity Options screen, click **Close**.
- From the VS2013 IDE, click on Device selection list box and select EduCake as the target device, as shown in Fig-3.6.

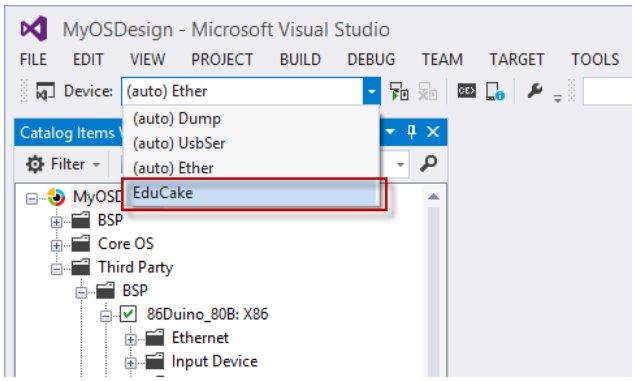

Fig-3.6: Select EduCake as target device.

Download OS Runtime Image to EduCake

With the EduCake and development machine both attached to the same LAN with DHCP service to provide IP address dynamically, insert the SD flash storage configured with BIOSLoader and Eboot.bin into EduCake, and go through the following steps to download OS runtime image generated from MyOSDesign project to EduCake:

- From VS2013 IDE, select **TARGET | Attach Device** to start the download process and bring up the Device Status screen, as shown in Fig-3.7.

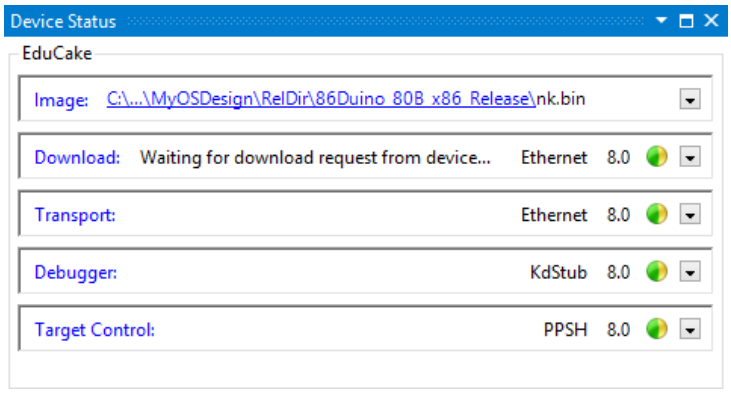

Fig-3.7: Target device status

- Power on the EduCake.
- As Eboot.bin is launched by BIOSLoader on the EduCake, it broadcasts a series of BOOTME messages.
- As BOOTME message is detected the download process starts and download activity is shown on the Device Status screen, as shown in Fig-3.8.

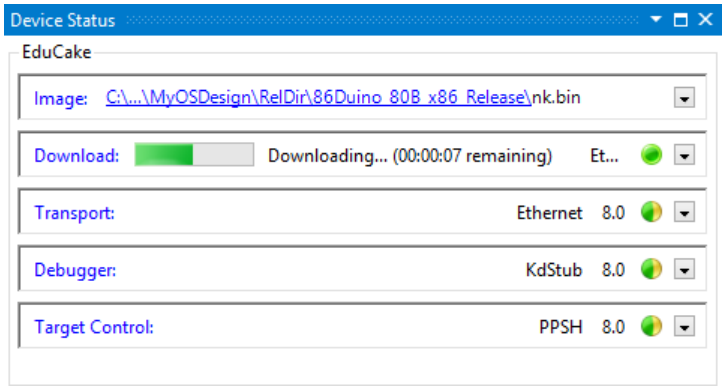

Fig-3.8: Target device status – Downloading OS runtime image

Kernel Independent Transport Layer (KITL)

The OS runtime image from MyOSDesign project is generated with KITL enabled. KITL is needed to establish connectivity and remotely debug the OS image running on the target device. Remote tools provided as part of the Compact 2013 Platform Builder also require KITL to function. With KITL enabled, as the OS runtime image launches on the EduCake, the Output window on the VS2013 IDE display activities from EduCake, as shown in Fig-3.9.

Fig-3.9: Target device startup activities.

Debug with Target Control

For headless device, without display or user interface, Compact 2013 provides the following useful debugging tools to simplify the tasks needed to debug OS runtime image on the target device remotely:

- Target Control
- Remote Tools

In this section, let's use Target Control to debug OS image on the EduCake. With MyOSDesign project open and OS runtime image downloaded to EduCake, go through the following steps to launch Target Control and debug OS image running on EduCake:

- From VS2013 IDE, select **TARGET | Target Control** to launch the Windows CE Command Prompt, as shown in Fig-3.10.

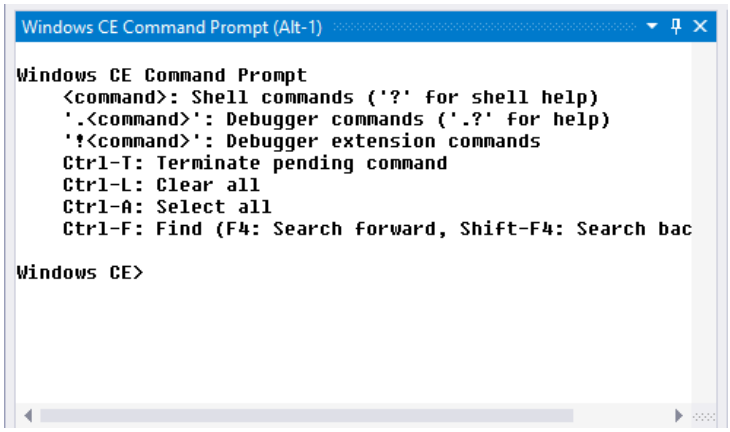

Fig-3.10: Windows CE Command Prompt

Think of the Windows CE Command Prompt (Target Control) window as the remote command prompt for the connected target device, where you can launch command to interrogate the device and application included as part of the OS runtime image.

Let's go through the following steps to run different Target Control command:

- From the Windows CE Command Prompt window, enter the following command to retrieve the target device's IP address:

```
s ipconfig /d
```

  In the above command, the "/d" parameter redirect the output to the VS2013 IDE's Output tab, as shown in Fig-3.11:

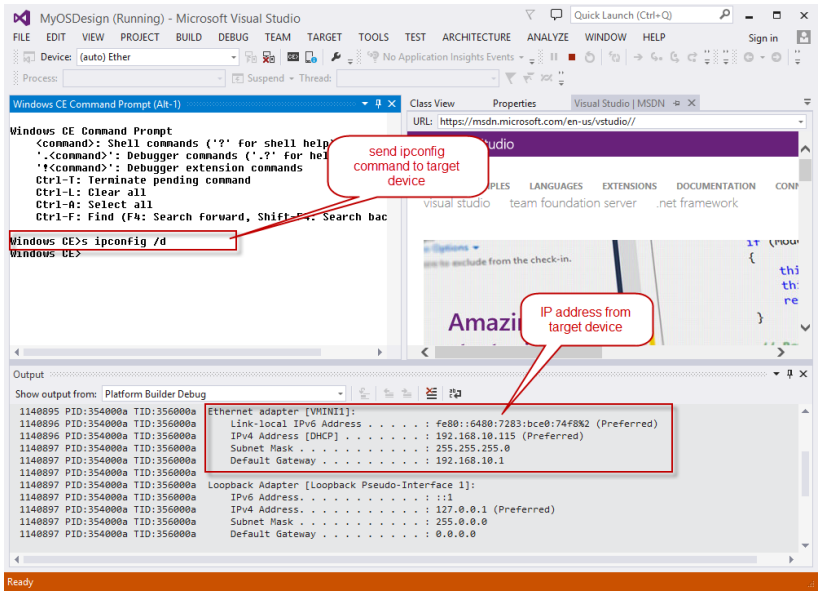

Fig-3.11: Target control – Retrieve IP address from target device

- Next, enter the following command to display active processes on the target device:

  gi proc

  As the above command execute, it lists the processes that are running on the target device, as shown in Fig-3.12.

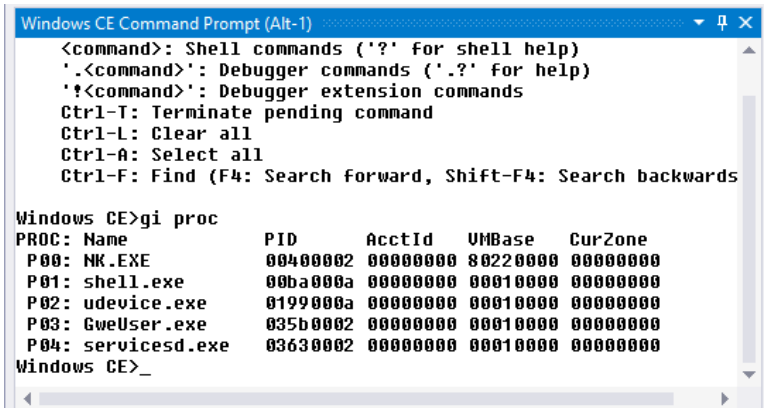

Fig-3.12: Target Control – List of running processes from target device

For more information about target control, visit the following URL:

https://msdn.microsoft.com/en-us/library/ee479807.aspx

Summary

In part-3 of this series, we talked about OS runtime image download to the target device and Target Control for debugging. In part-4, we will talk about debugging with remote tools.

Part-4: Application Development with VS2013 and C#

In part-4 of this getting started series, we will talk about the application development environment for Compact 2013 and work through the steps to develop a console C# application from Visual Studio 2013 IDE and deploy the application to the target device, EduCake, for testing and debug.

Compact 2013 Application Development

There are multiple options to develop Compact 2013 application. From the VS2013 IDE, you can develop application for Compact 2013 device using C, C++, C# or Visual Basic.

To develop application for a Compact 2013 device, you need the following:

- Visual Studio 2013
- Application Builder for Windows Embedded Compact 2013
- SDK for the Compact 2013 device

To test and debug the application, you need the following:

- A Compact 2013 target device preloaded with OS runtime image configure to support application deployment from Visual Studio 2013.

Compact 2013 OS Image for Application Development

Continue with the MyOSDesign project created in part-2 of this series, let's work through the steps to include additional components to the project, needed to support application development:

- Launch MyOSDesign project.
- From the Catalog Items View window, add additional components to the OS design project, as shown in Fig-4.1.

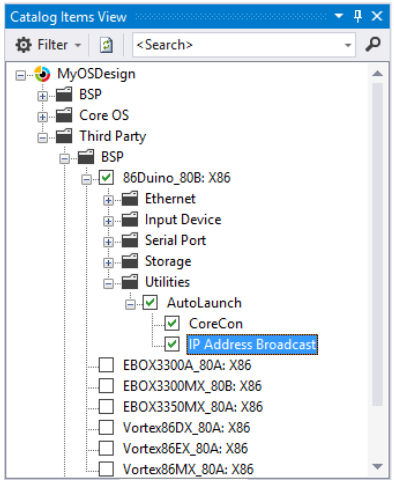

Fig-4.1: Catalog Items View window

- AutoLaunch

  When included in an OS image, the AutoLaunch application is configurable to launch one or more application during startup.

- CoreCon

  When included in an OS design project, the CoreCon component add the necessary files to establish CoreCon connectivity between VS2013 IDE and target device to download application to the device for debugging and testing.

- IP Address Broadcast

  As part of the application development process, to deploy application to the target device, you need to know the device's IP address. This component is designed to help you acquire IP address from the target device.

  When selected, this component add an application to the OS design, configured to launch during startup to broadcast the device's IP address via UDP.

  A Windows desktop application, IPAddressDiscovery, to listen for the UDP broadcast message from the target device is provided as part of the BSP, in the BSP's \Misc folder.

By default, OS design project is created with KITL enabled. KITL is known to cause connectivity problem for application development. Work through the following steps to disable KITL:

- From VS2013 IDE, select PROJECT | MyOSDesign Properties to bring up the MyOSDesign Property Pages screen, as shown in Fig-4.2.

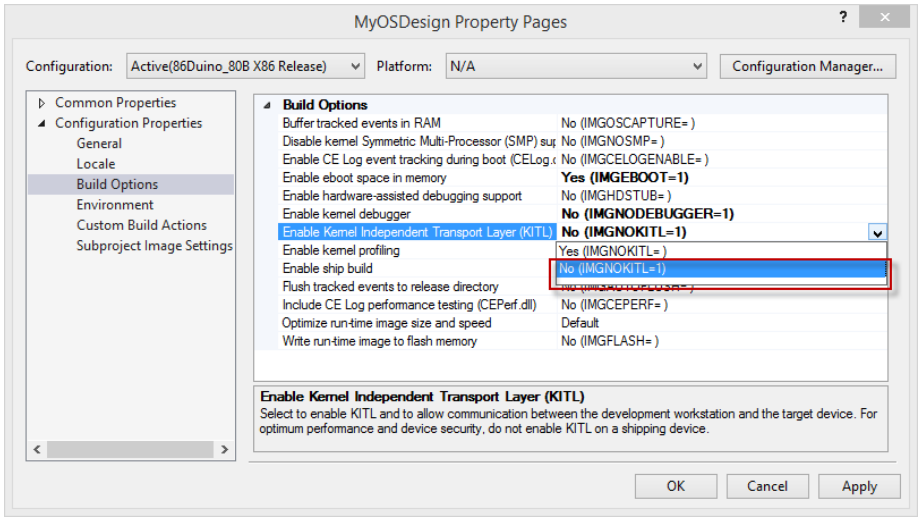

Fig-4.2: Disable KITL build option

- From MyOSDesign Property Pages screen, click on **Build Options** on the left pane.

- On the right, change the Enable Kernel Independent Transport Layer (KITL) build option to No.
- Click on Apply and then click on OK to close the screen.

Create an SDK from MyOSDesign

To support application development, we need to generate an SDK from the OS design project. Go through the following steps to create an SDK from the project:

- From the Solution Explorer window on VS2013 IDE, right mouse click on the SDKs folder and select **Add New SDK**, as shown in Fig-4.3.

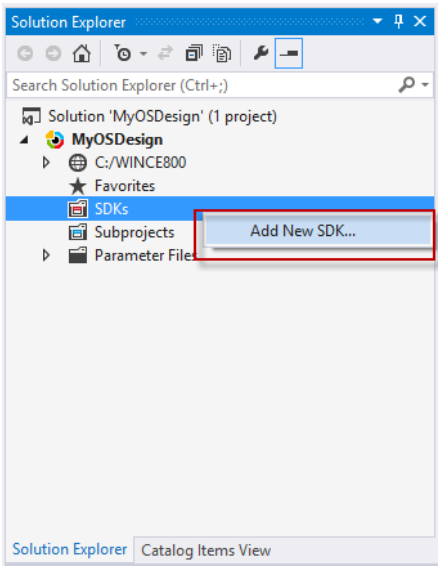

Fig-4.3: Add New SDK

- The SDK wizard brings up a screen to create the new SDK.
- Enter SDK name, product name, version info, company name and website, as shown in Fig-4.4.

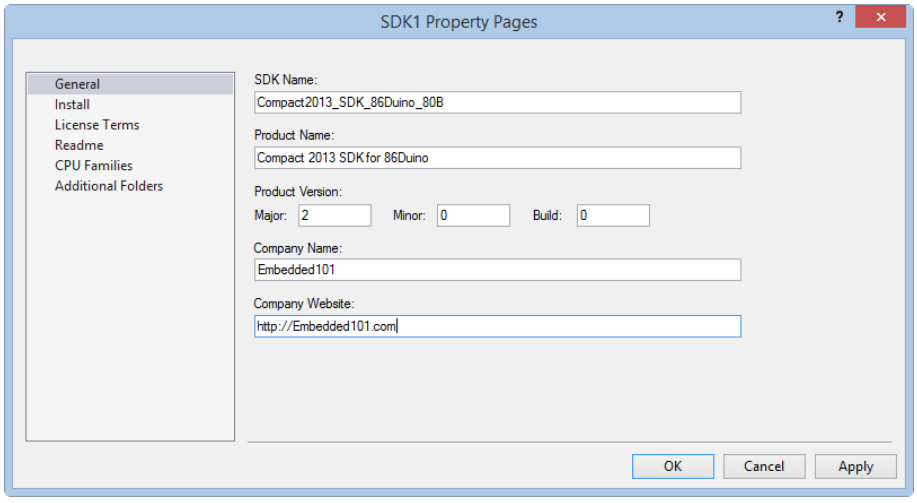

Fig-4.4: Create new SDK – SDK info

- On the left pane, click on Install and enter file name for the SDK, as shown in Fig-4.5.

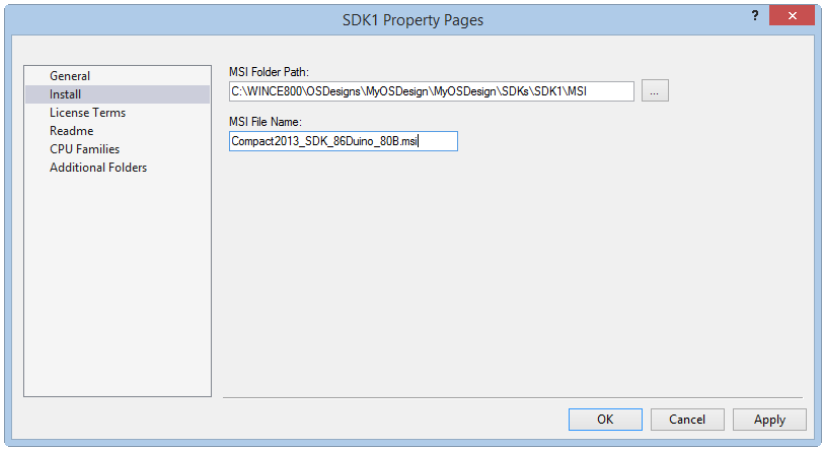

Fig-4.5: Create new SDK – SDK info

- Click Apply and then OK

Build and Generate OS Runtime Image

To build the OS design project, select **BUILD | Build Solution** from the VS2013 IDE.

If the build process ends with error, review the build.log and build.err files in the following directory to identify the causes:

```
C:\WINCE800\
```

After the OS design build process is completed, an OS runtime image, NK.bin, is generated in the following directory:

```
C:\WINCE800\OSDesigns\MyOSDesign\MyOSDesign\RelDir\86Duino_80B_Release
```

Build and Generate SDK

From the Solution Explorer window, right mouse click on Compact2013_SDK_86Duino_80B and select Build to build the SDK, as shown in Fig-4.6.

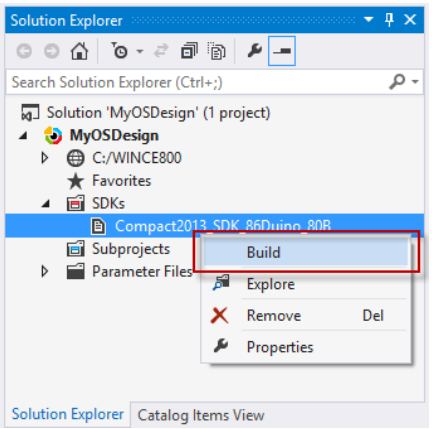

Fig-4.6: Build the SDK

After the build process is completed, the SDK file, Compact2013_SDK_86Duino_80B.msi, is generated in the following directory:

```
C:\WINCE800\OSDesigns\MyOSDesign\MyOSDesign\SDKs\SDK1\MSI
```

Install the Compact2013_SDK_86Duino_80B.msi SDK to support application development exercise in the next section.

Download OS Runtime Image to Target Device

Before getting into application development, we need to download OS runtime image to the target device, needed to test and debug the application.

With the development PC and target device attached to the same LAN, go through the following steps to download OS image to the target device:

- Launch the IPAddressDiscovery.exe application on the development PC, from the following folder:

```
C:\WINCE800\Platform\86Duino_80B\Misc
```

- With MyOSDesign project open, select **TARGET | Attach Device** from VS2013 IDE to initiate the download process.
- Power on the EduCake.
- As the OS image launches and the IPBroadcastCompact2013.exe excutes on the EduCake, the IPAddressDiscovery command prompt screen display IP address for the target device, as shown in Fig-4.7.

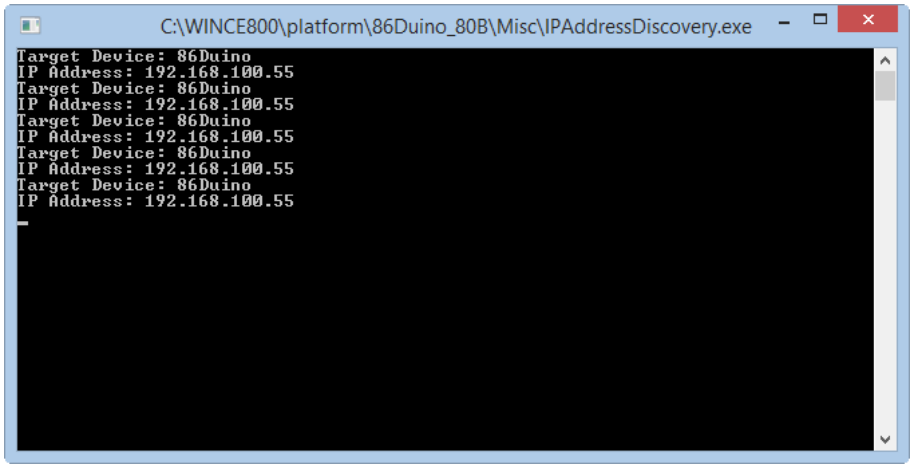

Fig-4.7: IPAddressDiscovery showing target device IP address

Note: The IPBroadcastCompact2013.exe application terminates itself after broadcasted 5 messages, with 2 second delay in between.

Develop Compact 2013 Application in C#

In this section, we will go through the steps to develop the IPBroadcastCompact2013 application in C# and deploy the application to run on the target device for testing and debugging.

Work through the following steps to create the application:

- Launch a new instance of VS2013 and click on **New Project**.
- From th left pane on the New Project screen, click on the following node:

```
\Other Languages\Visual C#\Windows Embedded Compact\Compact2013_SDK_86Duino_80B
```

- On the center pane, click to select Console Application and enter a name for the application, IPAddressBroadcast, and as shown in Fig-4.8.

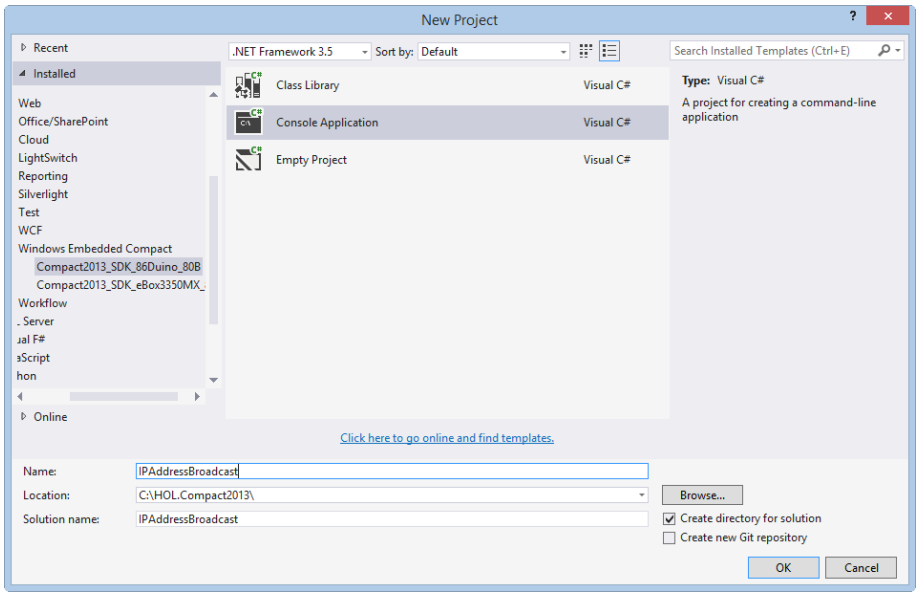

Fig-4.8: New C# application project for Compact 2013

- Click on OK to continue.
- Replace the codes in the Program.cs file with the following:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Threading;

namespace IPAddressBroadcast
{
    class Program
    {
        static void Main(string[] args)
        {
            for (int i = 0; i < 5; i++)
            {
```

```
                System.Net.Sockets.UdpClient sock =
                                                new System.Net.Sockets.UdpClient();

                IPEndPoint iep =
                        new IPEndPoint(IPAddress.Parse("255.255.255.255"), 15000);

                byte[] data = Encoding.ASCII.GetBytes("86Duino");
                sock.Send(data, data.Length, iep);
                sock.Close();

                Thread.Sleep(2000);

            }
        }
    }
}
```

- From VS2013 IDE, select **BUILD | Build Solution** to build the project.
- From the Solution Explorer window, right mouse click on the IPAddressBroadcast project and select properties to bring up the property page, as shown in Fig-4.9.

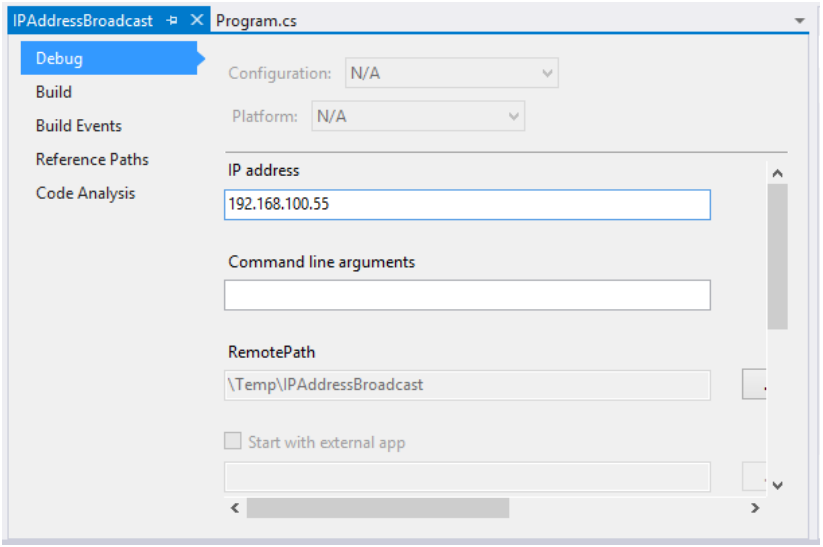

Fig-4.9: Application project property page.

- Enter the target device's IP address.

  If the IPAddressDiscovery application is not running on the development machine, launch it now before the next step.

- From VS2013 IDE, select DEBUG | Start Debugging to deploy the application to the target device.

  As the IPAddressBroadcast application is deployed and run on the target device, the IPAddressDiscovery command prompt window detect and display IP address from the target device.

  As you can see from the codes above, the program loops 5 times to send UDP broadcast message with 2 seconds delay in between, and terminate after the 5th iteration.

Application Debug

Continue from the previous step, let's place a break point on the following line of code:

```
sock.Send(data, data.Length, iep);
```

Now, select **DEBUG | Start Debugging** from VS2013 IDE to deploy the application to the target device.

As the application execute on the target device, it halt on the above line of code, as shown in Fig-4.10.

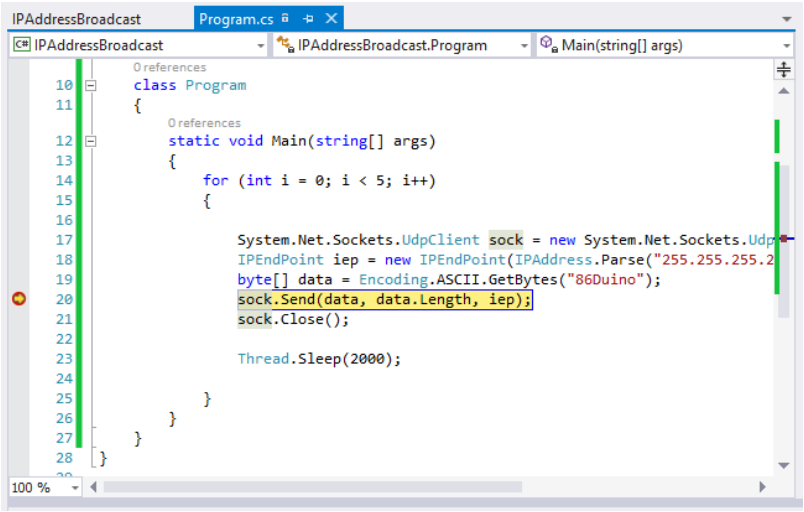

Fig-4.10: Application halt at breakpoint

At this point, you can press F11 to step through the code one line at a time and use debugging tools from VS2013 to view program variable value and operating status.

Next, press F5 for the application to continue. As the application continue to execute, it halt again after looping back to the same line of code, with the breakpoint.

Remove the breakpoint and press F5 again for the program to continue to run until completion and terminate.

Summary

Compact 2013 provides an efficient development environment that enables you to develop managed code application and the facility for you to remotely debug the application, from Visual Studio IDE, as the codes execute on the target device.

If you have existing managed code application development skills, you can leverage the existing skills you have to develop application for Compact 2013 devices.