IN-VEHICLE INFOTAINMENT (IVI) DUAL DISPLAY SHELL SYSTEM

NORHASLIZA BINTI MOHAMAD YUSOFF

ELECTRICAL & ELECTRONICS ENGINEERING
UNIVERSITI TEKNOLOGI PETRONAS
JUNE 2009

IN-VEHICLE INFOTAINMENT (IVI) DUAL DISPLAY SHELL SYSTEM

By

NORHASLIZA BT MOHAMAD YUSOFF

FINAL PROJECT REPORT

Submitted to the Electrical & Electronics Engineering Programme
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Universiti Teknologi Petronas
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

In-Vehicle Infotainment Dual Display Shell System

by

Norhasliza bt Mohamad Yusoff

A project interim submitted to the

Electrical & Electronics Engineering Programme

Universiti Teknologi PETRONAS

in partial fulfilment of the requirement for the

BACHELOR OF ENGINEERING (Hons)

(ELECTRICAL & ELECTRONICS ENGINEERING)

Approved by,

(DR YAP VOOI VOON)

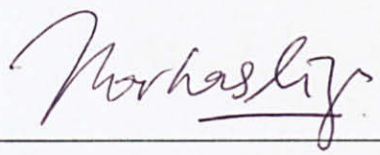
UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

June 2009

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



NORHASLIZA BT MOHAMAD YUSOFF

ABSTRACT

In-vehicle Infotainment system development is moving one step from the current technology. This project is focused on developing a dual display framework for In-Vehicle Infotainment System. The dual display is meant for the usage of the front and rear passenger in car. The framework will help car manufacturer to apply the recent technology to improve the in-vehicle infotainment system. Instead of having several hardware such as GPS and DVD player in a car, combining all in single platform will save space and provide a wide range of entertainment such as games and internet browsing. The development of this project includes building the Windows CE as the operating system, integrating and enabling dual display monitor using Intel Embedded Graphic Driver, and specifying the coordinate that differentiate the two monitor for application purposes. This report will discuss the method on developing the framework with the solution for problem that arises during the development of the project. This project was developed using a specific development machine and the problem are solved only for the machine used. Recommendations have been added for future development of the project.

ACKNOWLEDGEMENT

I would like to express my biggest appreciation to the following persons who had given great contribution towards making my project successful. First of all, I would like to thank to my project supervisor; Dr. Yap Vooi Voon who willing to spend time in his busy work schedule for giving an idea and advice for this project. His commitment and passion in guiding me has inspired me to work hard.

Secondly, I would like give a bunch of thanks to my internship supervisor, Douglas Cheah and Alan Previn Teres Alexis who gives the initial idea for my final year project. Their contribution on knowledge about Window CE and driver architecture gives in depth information toward the completion of this project so as gives help to any problem arises during the project development.

In particular, I would also like to express my gratitude towards great idea contribution and tools support given by my colleagues. Thanks to them for all the knowledge and skills that they share with me. Working with my project with a little help from them is a great moment to remember. Sometimes we joke around with each other to reduce our work stress.

Last but not least, thanks a lot to the examiner, because of their willingness to evaluate my project and give some idea for project improvement.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background Study

Many reputed companies are planning to establish new province in the field of in-vehicle infotainment systems. Today, a car is not just a medium of transport, but a medium to practice digital life style. It becomes a place to carry out business, access real time road and destination information, a medium for entertainment to listen to music and watch movies, get in touch with e-mails and short messages.

Although, it can be observed that many advance changes in the in vehicle infotainment system, there are some obstacles on the way to technology expansion. Some of them are as follows:

- High cost of technology
- Disruptive communication system for GPS system and internet access.
- Require additional knowledge to manage such systems.
- Increase number of hardware components like ports, cable circuits, antennas etc.

However, looking at the current advancement of technology, a "digital life" in car can be implemented. All of above factors suggest an evolution of in vehicle infotainment.

In the past, it is not possible to bring full computer ability into in vehicle infotainment system such as word processing, emails, entertainment (movie), games and even with GPS system all in one car. Despite having one user in front of the car, it will be an interesting application to have the children to be able to indulge themselves with entertainment of technology. Long distance journey will not be such a boring trip anymore.

1.2 Problem Statement

The recent technology has only one application per system for example GPS (Global Positioning System) system or DVD player. These two systems are working on different hardware independently which provide issues of space where the spacing in car are limited and the power supply also have to be utilized for both hardware. Instead, having both systems working in a single hardware might give solution to space and power problem. The idea now is to have a single platform that do all the jobs; games, GPS, movies, emails, and even internet. This is more like bringing the PC capabilities into car.

Intel's Tech Bulletin, May 21, 2008; cited Intel Corporation is committed to providing technology and collaborating with leaders in the embedded and automotive industries to advance in vehicle infotainment (IVI) solutions [2]. The product is called Intel® Atom™ processor that is being introduced April 2008 which is ideal for IVI system based on the small footprint and low-power design [2]. This product is an open platform which means any company that uses Intel's solution are able to customize different application.

The above discussion indicates that the IVI system can have one platform but with two different users; front seat and rear seat. Intel Embedded Graphic Driver support dual display environment for Microsoft Windows CE 6.0 via vertical extended display [3]. The most relieving part is Intel Embedded Graphic Driver support the chipset use by IVI Reference Design; Intel® System Controller Hub US15W [4]. By the combination of Intel® Atom processor and Intel® System Controller Hub US15W introduced in Low-Power Intel® In-Vehicle Infotainment Reference Design, enabled with Intel Embedded Graphic Driver, IVI System in a car will be everyone's dream.

1.2.1 Problem Identification

In general, automotive manufacturers have little or no experience in software development, and especially User Interface development [1]. The framework that will be created in this project is an implementation of new technology. Knowledge of both Windows CE and Intel Embedded Graphics Driver has to be combined to achieve the dual display framework shell system as proposed.

1.2.2 Significant of the Project

Manufacturer of the world's vehicles face a revolutionary moment as vehicles in future may have onboard computer systems, Internet access, and advanced display/interaction hardware. The project of creating the framework for automotive manufacture may breaks the IVI market that gives them solution to the product introduce by Intel; Low-Power Intel® IVI Design. Having a framework, customizing

the IVI System will be a simple programming helped by the framework's manuals and guidelines.

Looking deep into the significant of this project, the same framework can be extended to be implemented for other purposes that need two displays with two different users and two different usages in various fields.

1.3 Scope of Study

The aim of this project is developing a framework called Shell System for In-Vehicle Infotainment System. As mentioned above, the operating system used; Windows CE 6.0 is completely configurable especially the user interface. Windows CE 6.0 is a product of Microsoft and in order to help their user, video tutorial, blogs and 24 hour helpdesk is provided. The Windows CE team will work closely with several hardware manufacturers as they designed their device or product to ensure best possible integration of hardware and software [5].

The framework will provide the protocol that enables the application to specify the position of the display whether it is suppose to display at front seat display or at the rear seat display based on the user query. In order to do that, some Windows CE application has to be included in the Windows CE image to be tested out the protocols. However, the coverage of the project is not including the application that can be added in the framework which as refer to the figure below is the contribution by the third party and the scope of this project.

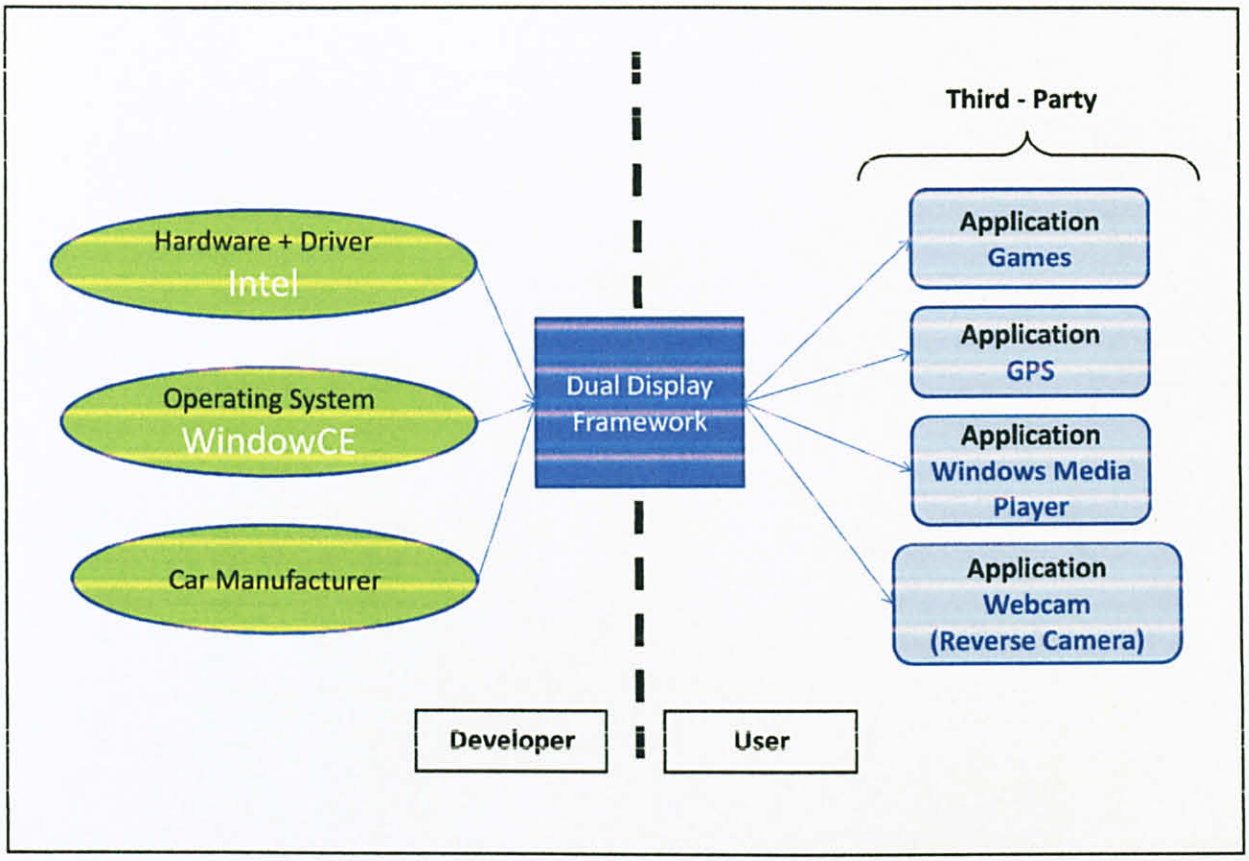


Figure 1: Scope of the project

CHAPTER 2

LITERATURE REVIEW

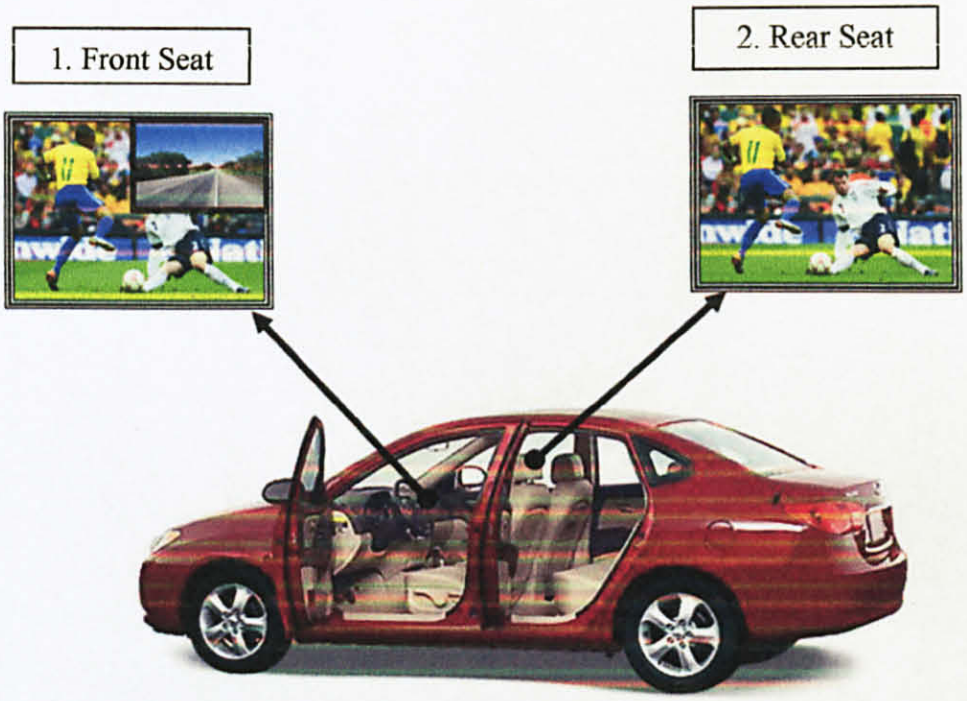


Figure 2: Concept of In-Vehicle Infotainment System

The aim of this project is to develop framework using Windows CE 6.0 as the operating system to have the environment for IVI System as displayed in Figure 2. The framework will have two displays that have applications relevant to the usage.

Theoretically, the front passenger has access to applications such as movie player, D3DM application, mp3 player and vertical to clone switch. While the rear passenger, will have access of application such as movie player, mp3 player and games. The applications for both front and rear monitor can be customize depending on the needs of the user.

Vertical Extended mode is the utility that separate a desktop into two different displays. By doing this, the user are able to customize the desktop so that physically it looks like it is two different platforms but theoretically it is one platform with one operating system. This mode can be enabled by using Intel Embedded Graphic Driver [3].

2.1 The Operating System

There are several operating system manufacturers such as Microsoft, Linux and Mac. However, for embedded market such as in-vehicle infotainment system the most important element is the capability for the operating system to be reconfigured according to the needs of the application [6]. According to the survey by the chief editor of embedded system magazine, Jim Turley, there are approximately 27 types of operating system that is being used in embedded market [7]. The article is about a survey of the type of operating system that is favorable to be used in the embedded project. The survey shows top three of the list are VXWorks, XP Embedded and Windows CE. The survey also shows that developers are prone to use operating system that they are familiar with which in the project coverage is Windows.

The available embedded operating systems for embedded market proposed by Microsoft are Windows CE, Window XP Embedded, Windows Embedded POSReady,

Windows Embedded Enterprise and Windows Embedded NavReady [8]. The operating system chosen must be suitable with this project requirement and Window CE is chosen for several factors. Window CE develops small footprint devices which is an important feature for embedded system which leads to low power consumption [8]. Window CE is a componentized, real time operating system which is an advantage for in-vehicle infotainment GPS system implementation while other operating system offered by Microsoft requires third party plug-in [8]. Furthermore, this operating system offer customized Win32 Application compared to full Win32 application compatibility in Windows XP Embedded [8]. This feature provides customizable operating system and it required small space that gives advantage to the application in In-Vehicle Infotainment System.

Furthermore, Window CE or any other embedded operating system trial version is available for evaluation. This is sufficient for this project as the license of the operating system is available for trial.

2.2 The Display Driver

The aim of this project is to get the framework to display into two monitor and behave independently. The key here is to make sure the framework display into both monitor where in fact it is working with one operating system. Extended mode is a capability that normally seen in Window XP which is provided by the graphic driver used. Examples of other modes are Clone and Twin mode.

In Window CE this features is also available with the usage of display driver. In this project, Intel Embedded Graphic Driver is chosen to be used as the driver. This is

due to the features of enabling vertical extended mode for Window CE [3]. The desktop of Window CE will be stretched vertically until the upper side of the desktop is displayed on the first monitor while the lower side of the desktop is displayed on the second monitor as shown in Figure 3 below. This feature is the key of success of the Dual Display framework.

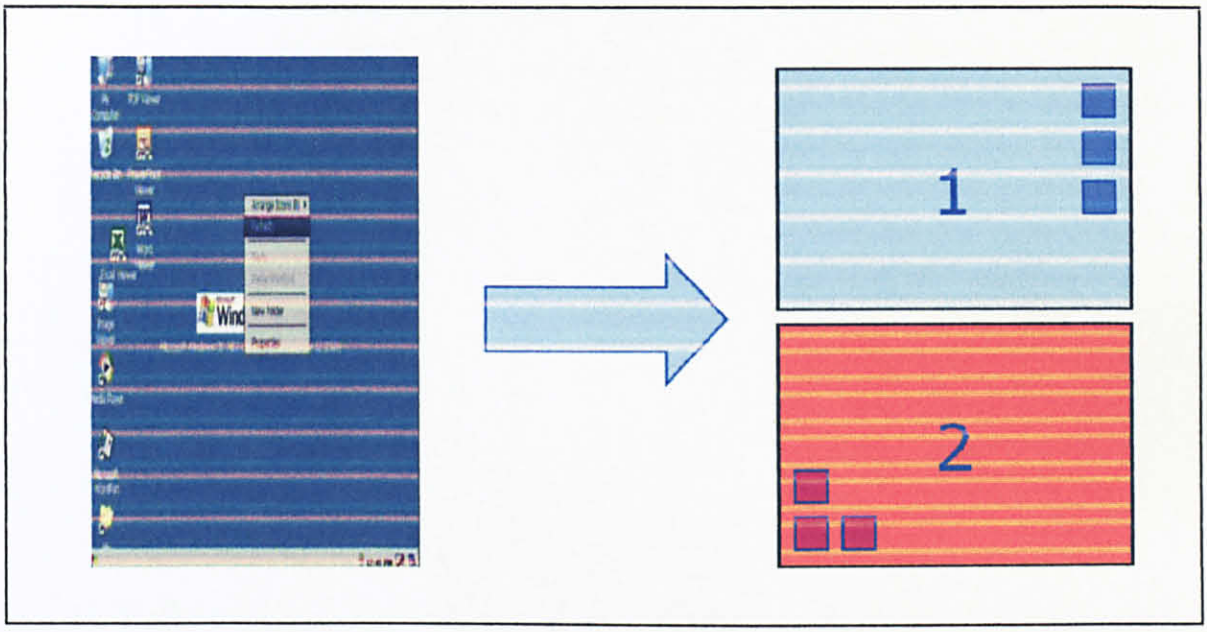


Figure 3: Vertical Extended to Two Different Display Concept

2.3 The Hardware

The scope of this project only covers the development of the framework. In order to get develop this framework with Window CE as the operating system, some requirement for powering the Window CE has to be highlighted. In-vehicle Infotainment market is a new technology that is under development process. Intel produce a reference board which is design specialize to be implemented in In-Vehicle Infotainment market [4]. This project coverage will only touches the display part of the whole system, purchasing the reference board for the project is not appropriate. Since

this project is related to the display of the framework, the important component that needs to be similar to the reference board is the graphic.

The reference board produced by Intel uses Intel latest processor, Intel Atom with Intel® System Controller Hub US15W chipset [2]. According to the product brief of Intel Embedded Graphic Driver, the driver produce support the chipset and the important element is that the driver can enable vertical extended mode.

The hardware for the target machine is decided to be chosen according to the chipset used by the board. As long as it can support USB boot where Window CE will be deploy and the chipset is supported by Intel Embedded Graphic Driver which capable of enabling vertical extended mode is sufficient for this project. It is decided to use a board that uses G33 express chipset which is supported by Intel Embedded Graphic Driver [3]. Monitor used for display are analog and DVI monitor. This is due to the availability of the monitor.

CHAPTER 3

METHODOLOGY

3.1 Flow Chart

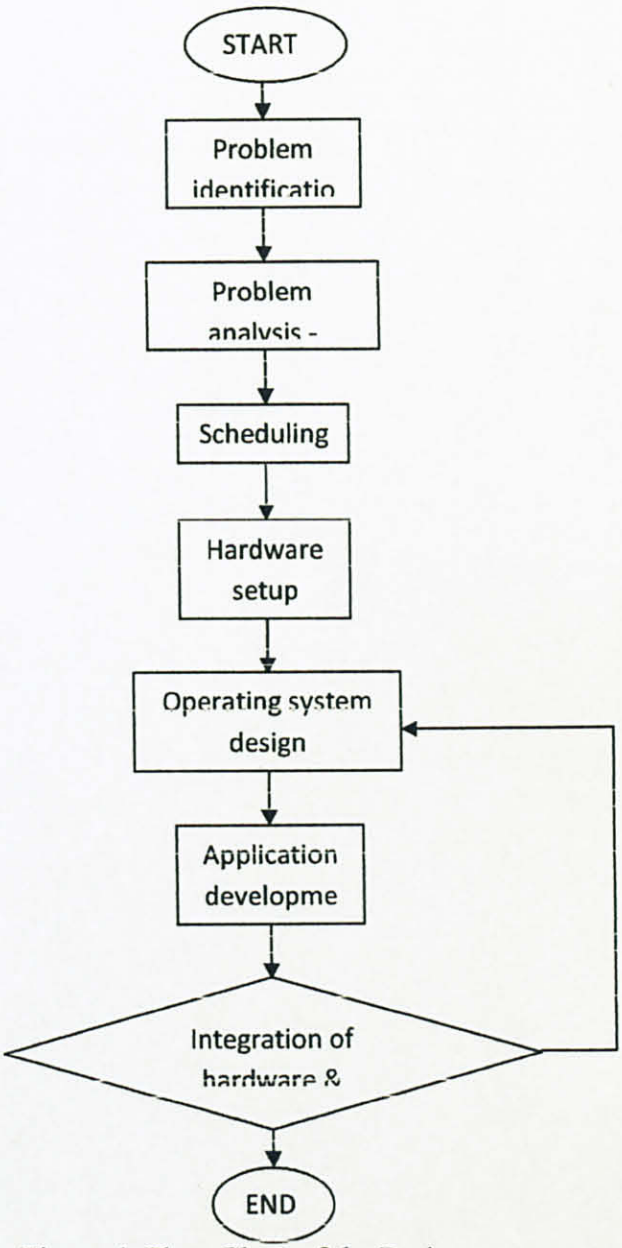


Figure 4: Flow Chart of the Project

3.2 Procedure

The progress of this project can be divided into three stages. By referring to the flowchart above, the first stage consist of problem identification, problem analysis and scheduling which was done at the beginning of final year project period. Then, the second stage is called development stage which consists of hardware setup, operating system design and application development. Last stage is the testing stage where the integration of the hardware and software is being done in this stage. Most of the effort is concentrated in the last stage since as the framework design has to be compatible with the hardware used.

3.2.1 Hardware Setup

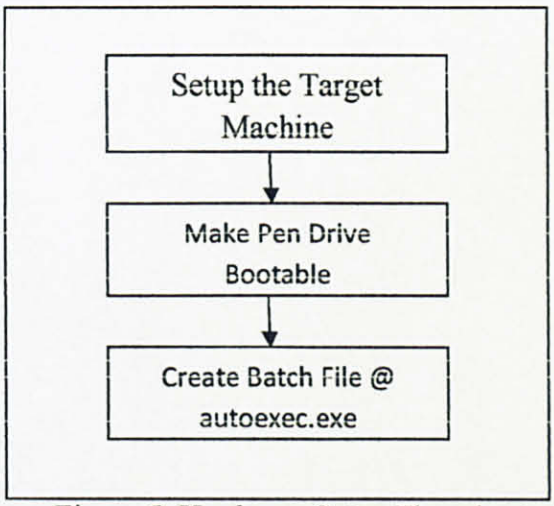


Figure 5: Hardware Setup Flowchart

Developing the framework leads to the need of having the exact environment of the IVI System proposed. The first step is to setup the target machine which is where the Window CE image will be booted up. The hardware display must be two displays

that portray the front seat display and rear seat display. In this case, one DVI monitor and one analog monitor are chosen for display purposes. Since it is still the stage of developing the framework, only normal keyboard and mouse are utilized. While for the type of hard disk used is a 2 GB pen drive.

As for the target machine where Windows CE image will be booted up, some interrupts must be disable in order for Windows CE to work properly. The Windows CE images that have been compiled will be stored in a pen drive with the bootloader. Development of the operating system and application is done in another machine. In order to make the image of Window CE to boot automatically, a batch file called autoexec.exe is created with following command:

loadcepc /v nk.bin

loadcepc : the boot loader, included in the release folder while nk.bin is created

/v : verbose option, that display the current activity when booting up the image

nk.bin : image of Windows CE

It is important to make sure that the boot loader and the image is in the same folder where the autoexec.bat is located if the command line used is the same as above. If the boot loader or the image is located in a specified folder, the command line must include the path of the object.

3.2.2 Operating System Development

As mentioned earlier, Windows CE is a configurable real time operating system where it has to be built with the desired component. In my case, the binaries of IEGD driver will be included in during the compilation of the operating system. The steps of building the operating system are available in the internet. Appendix 1 consists of the list of the steps that have been simplified to create a simple Windows CE operating system.

Windows CE 6.0 is being chosen as the operating system for this project is based on the ability to customize the look and feel of the desktop. It also support vertical extended mode that is being enable by Intel Embedded Graphic Driver [3]. Appendix 2 shows the steps to include the IEGD driver into the image. While Appendix 3 shows the steps to enable the vertical extended mode. Appendix 4 shows the step to include the Shell System created into the OS image.

However, the same dual display concept can also be applied using another operating system such as Window XP and Linux. As long as the driver used can support extended mode, the concept applied will still be the same. This will give some flexibility to the car manufacturer to choose as the same framework can be used in other type of application that requires dual display.

3.2.3 Application Development

Windows CE 6.0 uses C++ language as the SDK is delivered as the extensions to the Microsoft Visual C++ development environment [5]. However, the Shell

programming is being done using C# language as there is a tools called CEFileWiz that integrate the application to the image as long as the application is Windows CE based. For other application like movie players and D3DM will be develop in C++ language directly integrated with the image.

The lists of application need to be completed are:

1. Shell System – C# language
2. D3D application – mimic of GPS system – C# language
3. Vertical Extended to Clone Display Switch(violin Switch) – C++ language
4. CEplayit – C++ language

The shell system application was developed using C# language where the main reason is to create the GUI where C# provides GUI programming to create GUI application. Appendix 5 shows a sample coding of shell system. Each button reacts when it is clicked on and process the instruction assigned to it. For example:

```
private void button2_Click(object sender, EventArgs e)
  {
      Process proc = new Process();//Calling an application
      proc.EnableRaisingEvents = false;
      proc.StartInfo.FileName = "SpinningDonut";
      proc.Start();
      proc.WaitForExit();
  }
```

The application will retrieved the codes when the button2 is clicked. This is automatically assigned when I choose a button to be placed in my application. The coding inside the curly bracket are the instruction for the application to start a new application. In this case once the button is clicked, the ShellApp will invoke to start a

new application named SpinnindDonut.exe. SpinningDonut.exe is a D3Dm application that I have included in the image.

Since the backgound image will be covering two desktop, the proposed scheme will be using two different background image. As being set in iegd.reg, once the OS is booted up, the OS will be supporting resolution of 800x1200 where in a normal windows this resolution is being compared with 800x600. The background image can be included using the toolbox available in Visual C#. The image originated form two different image are attached to it's bottom and top.

The movie player appication coding was taken form the coding provided by WinCE 6.0 platform builder. Some modification in it's configuration need to be made in order to make sure that he coding is compiled as executable application. The features of this player is that it can play two instances at a time. Therefore, one movie can be played in the forst monitor and a second instances can be running on the second monitor. In order to do that, when the application is called in to be run the correct coordinate must be assigned in the command line so that the movie is being played at the correct monitor. Refering to Appendix 5, the coordinate for the first monitor is (0,0) where the width and height for the movie window is specified. Below is a figure showing the size and coordinate to be assigned for the first and second monitor.

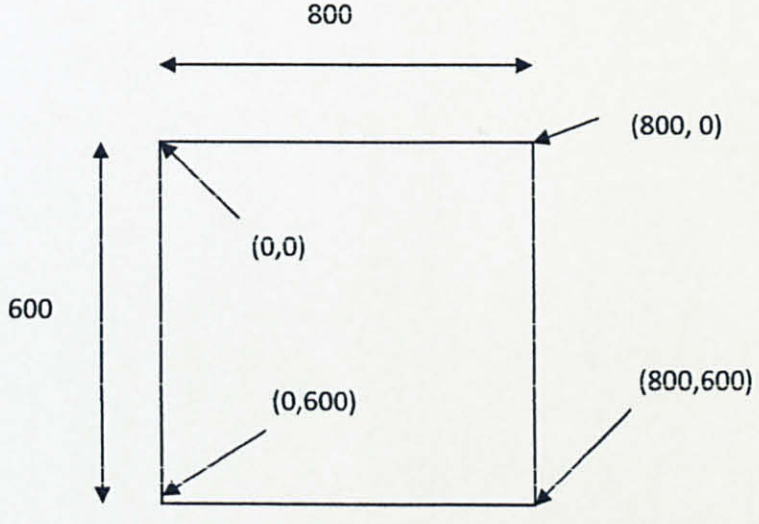


Figure 6: First monitor size and coordinates

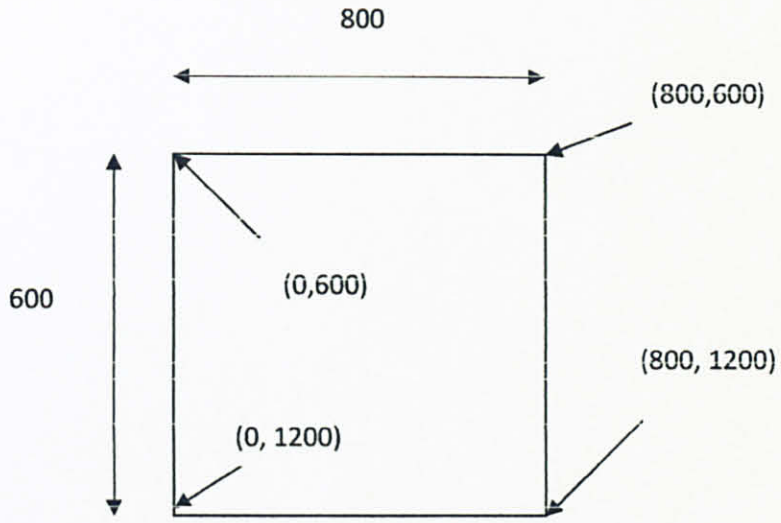


Figure 7: Second monitor size and coordinates

Once the correct coordinates have been correctly assigned, the ceplayit windows appeared with its default window size initially before resize to the size assigned. This application was written in class programming style. It involves several cpp files that handles different job at a time. Looking through the codes, some modification made so that the windows will directly displayed into assigned size. Appendix 5 enclosed consist of snippet codes from ceplayit application. There are two areas that were written by Microsoft that cause the window to appear to its original position before resizing to the new size. Both codes already being commented out and marked as "*change by li^jaja*"

The D3DM application can be run in the WinCE. The sample codes of the application are included in Appendix 7. The idea of this application is to have a mimic of GPS system that could visualize the D3D usage on this machine. The plan was to create a window and have a texture to be render inside the windows. The texture is then being rotate in cylindrical motion results in a moving effect of a map; having a map as

the texture. The code enclosed manages to create and render unmoved texture. This is the function that handles the rotation of the maps:

```
private void rotateSprite()
        {
            //Change the count to move to the next sprite tile
            countY++;
            if (countY >= 600)
            {
                countY = 0;
                countX++;
                //if (countX >= 4)
                //{
                //    countX = 0;
                //}
            }

            //Set the coordinates for the current tile
            tilePosition.X = tileSet.XOrigin + (countX *
            tileSet.ExtentX);
            tilePosition.Y = tileSet.YOrigin + (countY *
            tileSet.ExtentY);
        }
```

The rotateSprite function will be called by the main program. Suppose that the equation will rotationg the cylinder that wrap with texture which in this case the texture is a map image. However, since the rendering process it too fast, the texture is not visible. This application is the prove of the ability to develop application that uses D3DM library.

3.3 Tools

3.3.1 Hardware

To achieve the objective the hardware used has to be similar to the environment plan. The hardware used in this project is:

1. Monitors; 1 DVI and 1 analog
2. DVI card; Silicon Image DVI card
3. Target Machine; micro-ATX Intel Desktop Board DG31PR, Core2Duo processor, integrated 10/100/1000 Network connection and on board vga.
4. Development Machine – 945GM Mobile Board, Centrino Duo processor.
5. Kingston 2GB pen drive.
6. PS/2 keyboard & mouse.

3.3.2 Software

There are two software needed for this project and are available as a trial version in Microsoft webpage. The activation key can be request thru the webpage. The software are:

1. Microsoft Visual Studio 2005
2. Windows CE 6.0 Platform Builder

CHAPTER 4

RESULTS & DISCUSSION

4.1 Results

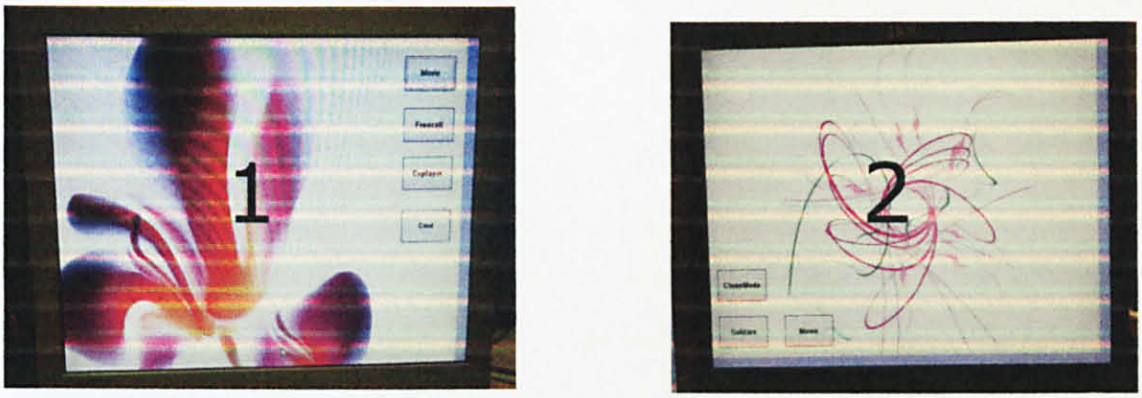


Figure 8: Initial IVI Dual Display Desktop Looks & Feel

To date, the Windows CE operating system with IEGD driver included is build and the shell as startup application is implemented. The shell is configured so that the look of the two displays as shown in Figure 3. This is the requirement for basic framework of the IVI system. By booting the Windows CE 6.0 with custom shell is the turning point of the success of the project. It proves that the main idea of having two different displays under one operating system can be implemented by using Intel Embedded Graphic Driver.

Move on to the next step is including applications for the shell to work with. The first application included in the image is a movie player. The basic coding of the movie player is provided in the Windows CE 6.0 Platform Builder package but is not configured to be an executable application. Some of the configuration need to be modify so that the player can be used in the OS. The movie player or known as ceplayit can be called using command line with display size and position set by user input. The movie player can now be use and display in different monitor.

The framework is ready to be used. Now is the time to integrate the framework to a real life model for better experience using the framework. The touch screens that support Window CE is the one that has board connected to it. Touch screen that is available with the size needed and driver support is meant for Window XP, Linux and Mac OS. The DVI output can be converted to VGA using DVI to VGA converter.

Now is the stage of applying the framework using touch screen. However, there is a slight problem with the DVI to VGA converter. The converter purchased cannot convert the signal from the DVI port to the monitor. The monitor is working fine, so as the DVI port when I use DVI monitor. This may due to the compatibility problem between the dvi card with the converter. Usually, DVI to analog converter is ship with the display card itself.

Figure 9: The latest look of the framework

4.2 Discussion

The main idea of this project is providing a framework that can be used by Intel® solution; Low-Power Intel® In-Vehicle Infotainment Design with Intel Embedded Graphic Driver. Since the reference design is available for Intel's employee for internal development used only, the target machine is substitute with a micro-ATX Intel Desktop Board DG31PR. The concept is still the same with the usage of IEGD driver and lucky for me that IEGD driver also support other chipset. However, the architecture of the machine used is the same to the reference design which is x86 processor.

Booting Windows CE took quite an effort due to the interrupt problem. This is because the higher priority will get the machine attention compared to lower priority. Since a pen drive is used as the hard disk for this project, the USB priority might be lower than other interrupt in the system, causes hangs during booting up Windows CE. After disabling some unused feature in the system bios, the operating system boots normally.

The shell system is being developed using C# language. There is a tool called CEFileWiz that convert the executable file (.exe) into the Windows CE 8.0 catalog. This way, we can select the component created in the catalog that will then include the application directly as the subproject. This feature gives the flexibility of developing graphical user interface like application.

CHAPTER 5

CONCLUSION & RECOMMENDATION

5.1 Conclusion

The aim of this project is to be able to use IEGD driver to develop a framework that can be use in In-Vehicle Infotainment System. There is a bright possibility that the framework can be developed and customized as the turning point of developing dual display has already been achieved. All of the applications are able to run on the platform and operating system. However, there are still some polishing programming needs to be added for the application to be run flawlessly. The application included will depend on the user based on the usage. The integration steps are included in the appendix. This way, any application developer can create application for IVI system and then integrate with the framework.

This project require several steps to enabled the framework that requires to the building the operating system, the driver, disable the default and attached with the framework that handle the dual display. All this steps need to be done accordingly to integrate the framework. The objective of this project is achieved as a framework of the dual display was created and can be further.

5.2 Recommendation

The framework for Dual Display Shell System was created in this project. Basically, more improvement can be done from this point in term of customizing with more interesting graphical user interface (GUI) for the framework, depending on the requirement of the user.

Implementing the framework to different hardware architecture may require different steps in term of compatibility. It is highly recommended to make sure the architecture of the processor, as in this project the framework is develop on x86 processor and the driver compatibility which in this project the driver used is Intel Embedded Graphic Driver.

Windows CE is a customizable real time operating system. Some of the features can be added during the operating system development. This will give more flexibility to the user so that it can be customize according to their need. Application such as reverse camera can be included for further improvement. This is proven as one of the application in the project uses D3DM library.

REFERENCES

[1] Deborah A. Boehm-Davis, Aaron Marcus, Paul Allen Green, Hideki Hada, David Wheatly, The next revolution: vehicle user-interfaces and the global rider/driver experience, CHI '03 extended abstract on Human factor in computing systems, April 05 – 10, 2003, Ft. Lauderdale, Florida, USA.

[2] Intel® Corporation, INTEL ADVANCES IN-VEHICLE INFOTAINMENT SOLUTIONS WITH OPEN PLATFORMS, http://www.intel.com/pressroom/archive/reference/IVIOpenPlatforms.pdf, NOVI Mich., May 21, 2008.

[3] Intel® Corporation, Intel® Embedded Graphics Driver for Embedded Intel® Architecture-Based Chipsets, http://www.intel.com/design/intarch/swsup/graphics_drivers.htm.

[4] Intel® Corporation, Low-Power Intel® IN-Vehicle Infotainment Reference Design, http://download.intel.com/design/embedded/infotainment/docs/Intel_IVI_Reference_Design.pdf, 0408/LS/OCG/XX/PDF, 319785-0001US, 2008.

[5] Robert O'Hara, Microsoft Windows CE: A New Handheld Computing Platform, Microsoft Corporation, One Microsoft Way, Redmond WA 98052-6399, ACM 0-89791-850-9 97 0002, 1997.

[6] S. Baskiyar, N. Meghanathan, A Survey of Contemporary Real Time Operating Systems, June 26, 2004.

[7] Jim Turley, Embedded System on the Rise, http://www.embedded.com/columns/showArticle.jhtml?articleID=187203732, June 21st, 2006.

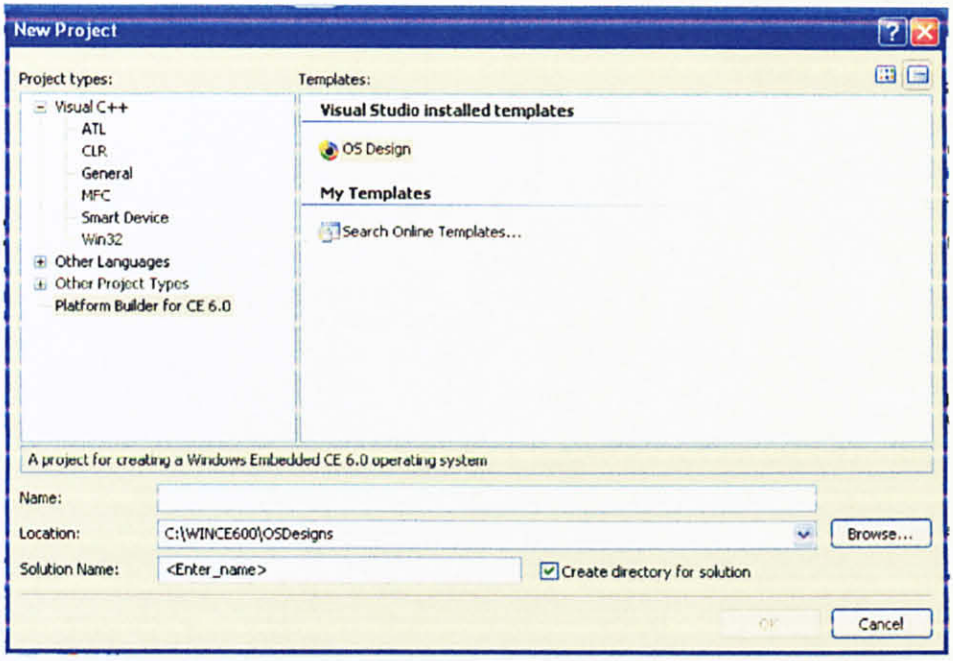
[8] Windows Embedded Product Description, http://www.microsoft.com/windowsembedded/en-us/products/whichproduct/default.mspx.

APPENDICES

Appendix 1: Creating a Window CE 6.0 Operating System

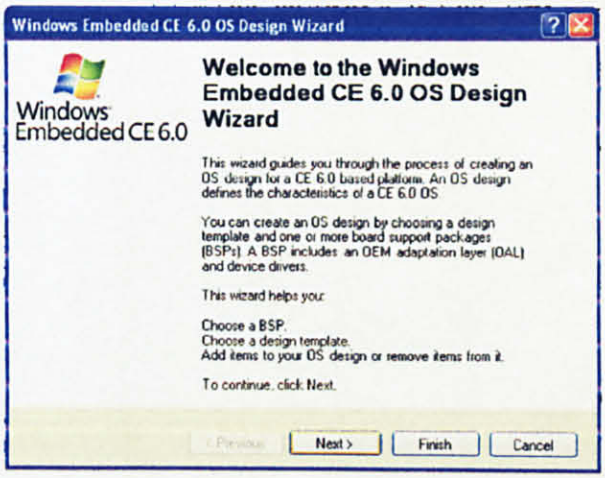
Basics:

- Start Visual Studio 2005, File -> New -> Project.



Select Platform Builder for CE 6.0, OS Design

Create custom device platform with the following selections for the step specified. If a step is not specified, then just click NEXT.

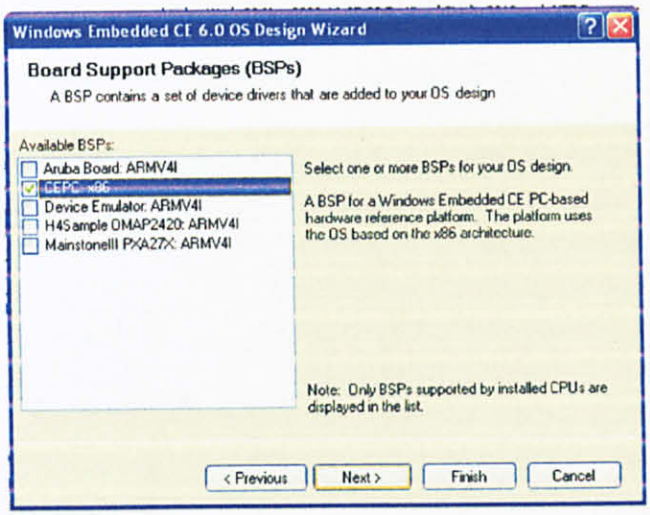


- **Step 1: New Project**

Just give it a name, e.g. ShellSystem

- **Step 2: Board Support Packages (BSPs)**

✓ CEPC: X86



- **Step 3: Design Templates**

✓ Custom Device

- **Step 4: Applications - End User**

✓ Games

✓ WordPad

- **Step 5: Applications and Services Development**

✓ .NET Compact Framework 2.0

✓ C Library and Runtimes

- **Step 6: Communication Services & Networking**

None

- Step 7: Core OS Services

- ✓ Device Manager
- ✓ Display Support
- ✓ Kernel Functionality → Target Control Support (Shell.exe)

- Step 8: Devices Management

None

- Step 9: File Systems and Data Store

- ✓ File System →Internal → RAM and ROM File System
- ✓ Registry Storage → RAM→based Registry

- Step 10: Fonts

- ✓ Arial
- ✓ Courier New
- ✓ Times New Roman

- Step 11: Graphics and Multimedia Technologies

- ✓ Graphics
- ✓ Media

- Step 12: International

None

- Step 13: Internet Client Services

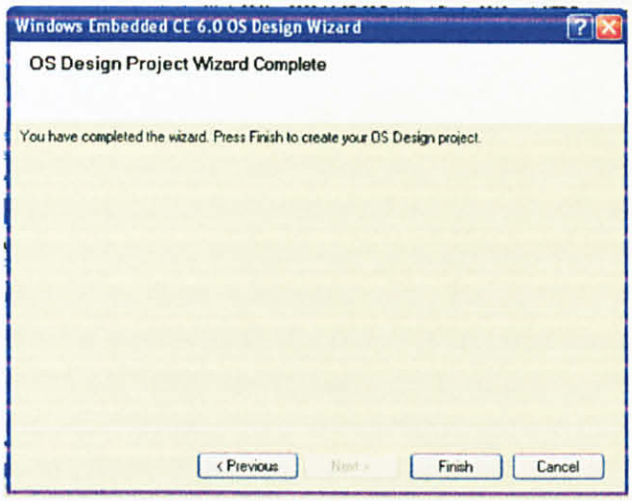
- ✓ Internet Explorer 6.0 for Windows Embedded

- Step 14: Security

None

- Step 15: Graphics, Windowsing and Events

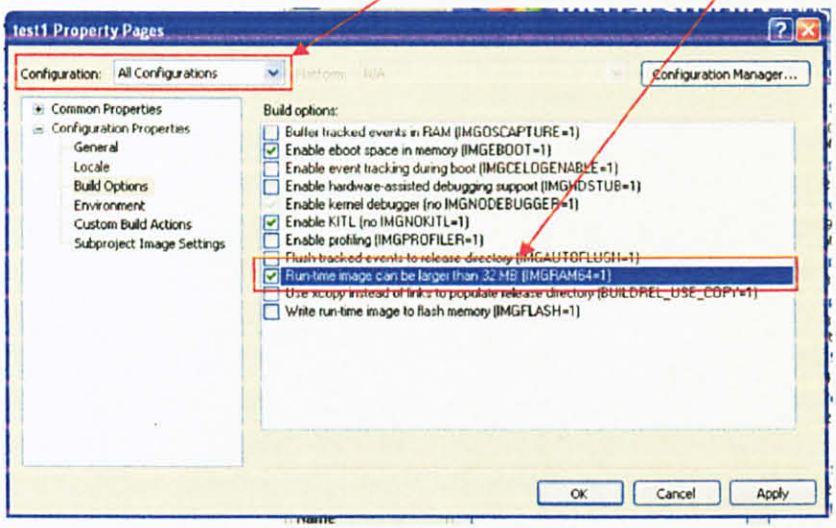
- ✓ Shell → Graphical Shell → Standard Shell
- ✓ Shell → Command Shell → Console Window
- ✓ UI: Everything except for Software Input Panel and Touch Screen



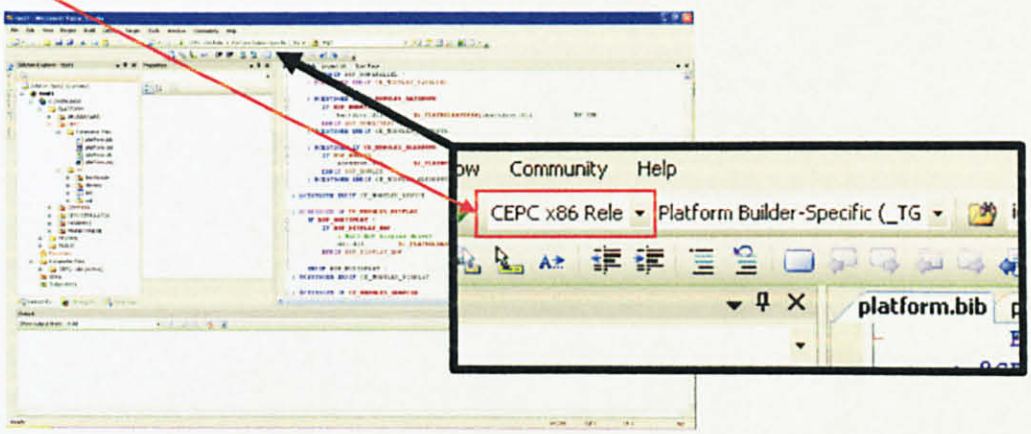
The below notification is the acknowledgement of using third party DVD Renderer. Click on Acknowledge.

In Solution Explorer, right click and choose on properties @ go to Project → Properties.

- A new window will pop up, <project> Property Pages
- Make sure the Configuration is set to "All Configuration"
- In Build Options - tick on Runtime image can be larger than 32MB (IMGRAM64=1)



For this project, the image used is a released version, so set the Solution Configuration to CEPC X86 Release



The final step is to click on Build → Build Solution (F7)

- ✓ This step usually takes around one hour to complete

Appendix 2: Include IEGD Driver into Image

1. In Solution Explorer, right click and click on properties or go to project-->properties.

-A new window will pop up, <project> Property Pages

-In Build Options - tick on Runtime image can be larger than 32MB (IMGRAM64=1)

-In Environment create this variables and its value

SSIGD=<location oF the driver>ssigd

SSOAL=wince

SSLPD=wince

MAKE_MODE=unix

2. In Solution Explorer

-Go to Platform->CEPC->Parameter Files->Platform.bib

-Append this at the end of platform.bib where you can see there is a place to add Files

ddi_iegd.dll	$(SSIGD)\ddi_iegd.dll	NK	SHK
iegd3dg3.dii	$(SSIGD)\iegd3dg3.dii	NK	SHK
iegd3dg4.dll	$(SSIGD)\iegd3dg4.dll	NK	SHK
analog.dll	$(SSIGD)\analog.dll	NK	SHK
lvds.dll	$(SSIGD)\lvds.dll	NK	SHK
sdvo.dll	$(SSIGD)\sdvo.dll	NK	SHK
hdmi.dll	$(SSIGD)\hdmi.dll	NK	SHK

3. Go to Platform->Cepc->platform.reg

-Click on the source.

-Add this just before ENDIF "BSP_NODISPLAY!"

```
;-------------------------------------------------------------------------

; IEGD

;-------------------------------------------------------------------------

[HKEY_LOCAL_MACHINE\System\GDI\DisplayCandidates]

    "Candidate3"="Drivers\\Display\\Intel"

#include "<location of the driver binaries>\iegd.reg"
```

4. Once u have finish adding the changes...Rebuild the solution in Build->Rebuild Solution @ CTRL+ALT+F7

Appendix 3: Enabling the vertical extended mode

Included with the binaries, there is a file named iegd.reg. This file is used to set the initial state of the driver once the image booted up. Here is where the setting of vertical extended mode is being done.

Step 1- gets 2 displays – one DVI and one analog panel

Step 2 – ensure both displays can support the same resolution

- Example, connect DVI that supports 10x7 and get an analog panel that is also supporting 10x7

Step 3 - In existing iegd registry configuration – u need to set the height = 2Xwidth

- Example below:

"Width"=dword:400 ; = 1024

"Height"=dword:600 ; = 1536 = 768 x 2

Step 4 – configure for vertical extended

"DisplayConfig"=dword:1

Step 5 – ensure the correct ports are selected

"PortOrder"="24000"

; 2 = SDVOB, 4 = int-LVDS

** There are instructions on top of every configuration in iegd.reg.

Appendix 4: Integrating the Shell System into OS image

Step 1:

In Wince Platform Builder, in Catalog view, go to

Core OS → CEBASE → Shell and User Interface → Shell → Graphical Shell.

Uncheck the option. This is to disable the Standard Shell.

Step 2:

Create a Subproject under a name PlatformStartup.

Choose Add and Item →add PlatformStartup.cpp file

Sample of PlatformStartup.cpp file is included in Appendix 8. This is to prepare the OS to boot using the Shell System created.

Step 3:

Run CEFileWiz.exe (this is to make the application as a catalog added in the public folder).

i. A window named Windows CE File Wizard will pop up. Click on AddFile(s) button
ii. Add the exe file of the application (in this case ShellApp.exe - located in the <appProjectDirectory>/bin/debug@release).
iii. Put the Component name column as ShellAppCatalog.
iv. Click Build, add vendor's name and Done!

Step 4:

i. Go to catalog items view in Platform Builder → Click refresh → Expend the third Party
ii. The ShellAppCatalog option is visible now and then mark the ShellAppCatalog option.
iii. Check the Solution explorer, the ShellApp is already included in the subproject.
iv. Any changes in the Shell app, just rebuild the solution in the Shell app and rebuilt solution for ShellAppCatalog.
v. Add these lines in the ShellAppCatalog → Parameter → ShellAppCatalog.dat

root:-Directory("\"):-Directory("Startup")

Directory("\Startup"):-File("ShellApp.exe","\windows\ShellApp.exe")

Appendix 5: Sample coding for Shell System

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;

namespace ShellApp
{
    public partial class Form1 : Form
        {
         public Form1()
      {
        InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
        Process.Start("ceplayit", "a.wmv -y100 -w800 -h400");
                //Calling media player
         }


      private void button2_Click(object sender, EventArgs e)
          {
             Process proc = new Process();//Calling an application
        proc.EnableRaisingEvents = false;
        proc.StartInfo.FileName = "SpinningDonut";
        proc.Start();
        proc.WaitForExit();
          }

      private void button3_Click(object sender, EventArgs e)
      {
        Process proc = new Process();
        proc.EnableRaisingEvents = false;
        proc.StartInfo.FileName = "navigator3D";
        proc.Start();
        proc.WaitForExit();
      }
```

```
    private void button4_Click(object sender, EventArgs e)
    {
       Process proc = new Process();
       proc.EnableRaisingEvents = false;
       proc.StartInfo.FileName = "cmd";
       proc.Start();
       proc.WaitForExit();
    }

    private void button6_Click(object sender, EventArgs e)
    {
      Process.Start("CEDisplaySetting", "4 5");
    }

    private void button5_Click(object sender, EventArgs e)
    {
       Process.Start("ceplayit", "a.wmv -x640 -y100 -w160 -h120");
    }

    private void button7_Click(object sender, EventArgs e)
    {
       Process.Start("ceplayit", "a.wmv  -x100 -y700 -w600 -h400");
    }

    private void pictureBox2_GotFocus(object sender, EventArgs e)
    {
       /* this is to ensure client always remained at the backgroun*
          This function puts it at th emost bottom from z poitn*/
      this.TopMost = false;

    }

  }
}
```

Appendix 6: Sample coding of Ceplayit

```
// Main program code
int
APIENTRY
WinMain (
        HINSTANCE hInstance,
        HINSTANCE hPrevInstance,
        LPTSTR lpCmdLine,
        int nCmdShow
        )
{
   TCHAR szFile[MAX_PATH]={0}, szCmd[MAX_PATH]={0},
szOpt[MAX_PATH]={0};

   // Copy command line for manipulation
   StringCchCopy(szFile, MAX_PATH, lpCmdLine);
   ParseOptions(szFile, MAX_PATH, szCmd, MAX_PATH, szOpt, MAX_PATH);


   if (!ParseCommandLine(szOpt))
      return 1;
   if (! ConvertFilename(szFile, MAX_PATH, szCmd))
      return 1;

   RETAILMSG(1, (TEXT("CEPlayit: Playing file %s.\r\n"), szFile));

   // Create the main window
   if (!DoInit(hInstance, nCmdShow, szFile))
      return 1;


   // Get COM interfaces
   CoInitializeEx(NULL, COINIT_MULTITHREADED);


   if (FAILED(GetInterfaces()))
   {
      CoUninitialize();
      return 1;
   }

      //// Set initial window position – change by liz^jaja;disable initial //window position
   //SetInitialPosition();

         OutputDebugString(TEXT("before entering playmedia.\r\n\r\n"));
```

```
    // Now play the movie
    PlayMedia(szFile, hInstance);

        //OutputDebugString(TEXT("Lepas Play Media.\r\n\r\n"));

    // Release COM interfaces
    CleanupInterfaces();
    CoUninitialize();

    // If a temporary file was created (for CESH source), delete it
    if (!g_bKeepCESHFile)
        DeleteTemporaryFile();

   OutputDebugString(TEXT("CEPlayit exiting.\r\n\r\n"));
    return 0;
}
```

```
HRESULT PlayMedia(LPCTSTR lpszMovie, HINSTANCE hInstance)
{
    HRESULT hr = S_OK;
    TCHAR buf[256];


    if (g_bWindowless)
    {
        JIF(AddWindowlessRendererToGraph(pGB));
    }

    // Allow DirectShow to create the FilterGraph for this media file
    hr = pGB->RenderFile(lpszMovie, NULL);
    if (FAILED(hr)) {
                StringCchPrintf(buf, 256, TEXT("Failed(%08lx) in RenderFile(%s)!\r\n"),
          hr, lpszMovie);
        OutputDebugString(buf);
        return hr;
    }

    // Set the message drain of the video window to point to our hidden
    // application window.  This allows keyboard input to be transferred
    // to our main window for processing.
    //
            // If this is an audio-only or MIDI file, then put_MessageDrain will fail.
    //
```

```
hr = pVW->put_MessageDrain((OAHWND) g_hwndMain);
if (FAILED(hr))
{
    // Show the main hidden window for keyboard control
    ShowCurrentAudioFile(lpszMovie);

    ShowWindow(g_hwndMain, SW_SHOWNORMAL);
    g_bShowingMainWindow = TRUE;
    g_bAudioOnly = TRUE;
    SetForegroundWindow(g_hwndMain);
    SetFocus(g_hwndMain);
}
else
{
    ShowCurrentVideoFile(lpszMovie);

}

// Initialize state variables
g_psCurrent = psSTOPPED;
g_vsCurrent = vsDEFAULT;

        //MessageBox(NULL,L"Try Test",NULL,MB_OK);
// Display first frame of the movie
/* hr = pMC->Pause();
if (FAILED(hr)) {
    StringCchPrintf(buf, 256, TEXT("Failed(%08lx) in Pause()!\r\n"), hr);
    OutputDebugString(buf);
    return hr;
}*/
        //liz^jaja - change to make the original window not to display before set initial
        position


// Override defaults with command line options
ProcessStartupOptions();


// Get focus for keyboard input
SetFocus(g_hwndMain);
SetForegroundWindow(g_hwndMain);


// Start playback
hr = pMC->Run();
if (FAILED(hr)) {
```

```
        StringCchPrintf(buf, 256, TEXT("Failed(%08lx) in Run()!\r\n"), hr);
        OutputDebugString(buf);
        return hr;
    }

    // Set to fullscreen mode if requested
    if (g_bFullScreenStart)
        ToggleFullScreen();

    // Update state variables
    g_psCurrent = psRUNNING;
    g_bContinue = TRUE;

        g_bShowEvents = TRUE;
    // Process messages and DirectShow events
    ProcessEventLoop();

    if (g_bShowingMainWindow)
    {
        ShowWindow(g_hwndMain, SW_HIDE);
    }

CLEANUP:
    return hr;
}
```

Appendix 7: Sample coding D3Dm application

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Microsoft.WindowsMobile.DirectX;
using Microsoft.WindowsMobile.DirectX.Direct3D;
using System.Reflection;

namespace navigator3D
{
    partial class Form1 : Form
    {
      Device DX_Device = null; // Drawing device
      Texture texture = null; //Sprite bmp
      Rectangle tilePosition; //Location of tile
      TileSet tileSet = null; //Local TileSet
```

```
Sprite sprite;        //Sprite Varible
Int16 countX = 0;     //Count to Track Columes of Tiles
Int16 countY = 0;     //Count to Track Rows of Tiles
    Vector3 spritePosition = new Vector3(0, 0, 0); //Start Location
                                                   //Of Sprite
Vector3 spriteCenter = new Vector3(0, 0, 0);//Sprite Center
Vector2 spriteVelocity = new Vector2(1, 1);// x,y velocity

 public Form1()
 {
    this.ClientSize = new Size(400,400);//Specify the client
                                        //size
    this.Text = "GPS System"; // Specify the title
    this.MinimizeBox = false;
    this.MaximizeBox = false;
    this.BringToFront();
 }

 public bool InitializeDirect3D()
 {
    try
    {
       PresentParameters pps = new PresentParameters();
       pps.Windowed = true; // Specify that it will be in a
                            //window
       // When a new frame is swapped to the front buffer,
       // the old frame will be discarded
       pps.BackBufferCount = 1;
       pps.BackBufferFormat = Format.R5G6B5;
       pps.BackBufferHeight = this.Height;
       pps.BackBufferWidth = this.Width;
       pps.SwapEffect = SwapEffect.Discard;
             // After the current screen is drawn, it will be
             //automatically deleted from memory
       DX_Device = new Device(0, DeviceType.Default, this,
CreateFlags.None, pps); // Put everything into the
                       //device

       DX_Device.DeviceReset += new EventHandler(
          this.OnResetDevice);

       this.OnResetDevice(DX_Device, null);

    }
    catch (DirectXException e)
    {
```

```
        MessageBox.Show(e.ToString(), "Error"); // Handle all
                                                    //the exceptions
        return false;
    }
    return true;
}

void OnResetDevice(object sender, EventArgs e)
{

    Device dev = (Device)sender;

    //Create sprite object on device
    sprite = new Sprite(dev);

    texture = TextureLoader.FromStream(dev,
            Assembly.GetExecutingAssembly()
            .GetManifestResourceStream("navigator3D.aaa.jpg"),
            D3DX.Default, 256, 256, 0,
            Usage.None, Format.Unknown, Pool.VideoMemory,
            (Microsoft.WindowsMobile.DirectX.Direct3D.Filter)D3DX.Default,
            (Microsoft.WindowsMobile.DirectX.Direct3D.Filter)D3DX.Default, 0);


    //Configure TileSet to the size of tile in bmp
    tileSet = new TileSet(texture, 0, 0, 600, 0, 800, 600);

            //Set tilePosition including the offset to the center of the
    //sprite
    tilePosition = new Rectangle(tileSet.XOrigin,
        tileSet.YOrigin, tileSet.ExtentX, tileSet.ExtentY);

}
private void Render()
{
            if (DX_Device == null) // If the device is empty don't bother rendering
    {
        return;
    }

    DX_Device.Clear(ClearFlags.Target, Color.White, 1.0f, 0);
            // Clear the window to white
    DX_Device.BeginScene();

    sprite.Begin(SpriteFlags.None);
```

```
        //Draw a sprite at the current position
        sprite.Draw(tileSet.Texture, tilePosition,
            spriteCenter, spritePosition, Color.White.ToArgb());

        sprite.End();

        DX_Device.EndScene();
        DX_Device.Present();

        rotateSprite();
    }

    private void rotateSprite()
    {
        //Change the count to move to the next sprite tile
        countY++;
        if (countY >= 600)
        {
            countY = 0;
            countX++;
            //if (countX >= 4)
            //{
            //    countX = 0;
            //}
        }
        //Set the coordinates for the current tile
                tilePosition.X = tileSet.XOrigin + (countX * tileSet.ExtentX);
                tilePosition.Y = tileSet.YOrigin + (countY * tileSet.ExtentY);
    }

    protected override void OnPaint(PaintEventArgs e)
    {
        // Render on painting
        this.Render();

        // Render again
        this.Invalidate();
    }


  }
}
```

Appendix 7: Sample of PlatformStartup.cpp file

```
// PlatformStartup.cpp : Defines the entry point for the application.
//

#include "stdafx.h"

void ProcessStartupFolder();


int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPTSTR    lpCmdLine,
                   int       nCmdShow)
{
 // Since this is application is launched through the registry HKLM\Init we need to call
SignalStarted passing in the command line parameter
 SignalStarted(_wtol(lpCmdLine));
 ProcessStartupFolder( );

 return 0;
}

// The purpose of this function is to start all applications found in our custom Startup
folder
// the Windows Explorer shell contains a "Special Folder" \Start Menu\Programs\Startup
- CSIDL_STARTUP
// For this demo I'm going to override this with a fixed location folder called "\Startup"
// This code is taken from the Microsoft Windows CE Shared Source in the following
location.
// c:\wince500\public\shell\oak\hpc\explorer\main\explorer.cpp

void ProcessStartupFolder()
{
   WCHAR szPath[MAX_PATH];
   HANDLE hFind = NULL;
   WIN32_FIND_DATA fd = {0};
   WCHAR pszFileName [MAX_PATH],

  // Note that we will need to create the Startup folder
  // this can be achieved in one of two ways
  //
  // 1. Create the Startup Folder using this projects .DAT file
  // or..
  // 2. Map our applications into the Startup Folder using CEFileWiz (my prefered option)
```

```
lstrcpy(szPath,L"\\Startup\\*.*");

   hFind = FindFirstFile(szPath, &fd);
   if (INVALID_HANDLE_VALUE != hFind)
   {
      SHELLEXECUTEINFO sei = {0};
      sei.cbSize = sizeof(sei);
      sei.nShow = SW_SHOWNORMAL;
      sei.lpFile = pszFileName;

      do
      {
  lstrcpy(pszFileName,L"\\Startup\\");
         if ((!(FILE_ATTRIBUTE_DIRECTORY & fd.dwFileAttributes)) &&
            (0 != _tcsicmp(TEXT("desktop.ini"), fd.cFileName)))
         {
   lstrcat(pszFileName, fd.cFileName);
            ShellExecuteEx(&sei);
         }
      } while (FindNextFile(hFind, &fd));
      FindClose(hFind);
   }
}
```