

Software Development for a Mobile Projector

Master Thesis
KTH/ICT/ECS-2009-6

Jin Luo
jinluo@kth.se
Supervisor: Yijun Zhao, OPTEA AB
Supervisor at KTH: Dr. Zhonghai Lu
Examiner: Prof. Axel Jantsch

Abstract

The projector technology has continually developed to satisfy the consumer's needs in high quality. Base on the DLP (Digital Light Processing) technology proposed by TI, the OPTEA AB intends to introduce a graphic processor REON to its new projector VEO+. TI provides a complete DLP design supporting package DDP2000 which includes DDP2000 chip set, SDK and API for the third party device.

This project is aimed to establish communication between the DDP2000 system and the REON chip set, and then design a new GUI (graphic user interface) for the end user. After weeks' document study, I found that the REON chip set is connected with DDP2000 via I2C bus of DDP2000. According to the discussion result from the meeting between OPTEA and Silicon Optix (The supplier of REON), I set the DDP2000 system as the master device of the I2C bus. By using the I2C port functions from API of DDP2000, I realized the communication between DDP2000 and REON. The GUI was based on the original DDP2000 GUI system OSD (On Screen Display, a GUI used in for the projector), which has 33k lines of code. After digesting the large amount of code, I have modified the GUI relating to 8k lines of code and I also added a button function to enable one REON function into the original OSD system for the end user.

I2C/SPI/MDIO Protocol Analyzer had been used for the validation of the communication between DDP2000 and REON by capturing all the data on the I2C bus. The results show all the data were sent to the right I2C addresses. To validate the improved OSD, we set operational tests to ensure that the output images are correct.

This thesis was performed at the OPTEA AB, Stockholm, Sweden.

Acknowledgements

This is my final thesis for the International Master's Program of System on Chip Design at Royal Institute of Technology (KTH). The work was carried out at OPTEA in Teknikhöjden Stockholm for 6 months

I would like to express my highest gratitude to the following people at OPTEA AB for their kindness and support during the whole procedure of the project work.

Thanks Esteban Arboix (CEO of OPTEA), thank you for giving me good opportunity to performance my thesis work on such an interesting topic, and your patience on a rookie as me.

Thanks Yijun Zhao (Project Manager of OPTEA), for you great guide and support throughout the whole work.

And then, I would like to show my sincere gratitude to my college Examiner: Dr. Zhonghai Lu and Prof. Axel Jantsch. You are such erudite and kind, and thanks to discussing with me about this work from time to time, giving me lots of valuable suggestions, and shared your experience and skills.

I would also like to thank David Rowark, BLUE MICROELECTRONICS LTD**,** and Dirk Bartholdtsen, Calibre UK Ltd, for your knowledge about REON chip. And last but not least my parents and friends for their endless support

Abbreviations

DLP	Digital Light Processing
TI	Texas Instruments
GUI	Graphical User Interface
DMD	Digital Micromirror Devices
SOC	System On Chip
HQV	Hollywood Quality Video
OSD	On Screen Display
GPS	Global Positioning System
IC	Integrated Circuit
GPIO	general purpose input/output
I^2C	Inter-Integrated Circuit
SDA	serial data
SCL	serial clock
ADS	ARM Developer Suit
API	Application Programming Interface
OEM	Original Equipment Manufacture
IDE	Integrated Development Environment
EEPROM	Electrically Erasable Programmable Read-Only Memory
RTOS	Real-Time Operating System
ARM	Advanced RISC Machines
RISC	Reduced Instruction Set Computer

Term of the Report

- OSD Outline
  OSD Outline is the actual characters, icons, windows, slider bars and other features that make up a menu and are displayed for the user to see

- Co-Work
  The result (.bin file) from the ADS v1.2 compile had been used for the DLP Composer compile.

- #include
  The directive ``#include'' tells the C compiler to insert the contents of the specified file at that point in the code. All variables in C must be explicitly defined before use. The ``.h'' files are by convention ``header files'' which contain definitions of variables and functions necessary for the functioning of a program [9]

- Slider
  A character slider field that is made up of several different icons and each icon represents a part of the slider.

- Window
  A window is a simple rectangular area. It can optionally have a raised or depressed 2-pixel border around the rectangle.

- Static Text
  It’s a field showing value that is used at design-time, and can't be changed during runtime.

- Dynamic Text
  It’s a field whose value is changeable at runtime. The value can be either numeric or text (or even icons). It can have a default value that is set at design-time that will be displayed unless an OSD API is called to set a new value.

- OSD Functions
  They allow complicated programs to be parceled up into small blocks, each of which is easier to write, read, and maintain.

TABLE OF CONTENTS

Abstract 2
Acknowledgements 3
Abbreviations 4
Term of the Report 5
TABLE OF CONTENTS 6
LIST OF FIGURE 7
LISTING 8
1 INTRODUCTION 9
1.1 About the Project 9
1.1 About the OPTEA AB and Veo+ Projector 9
1.3 Structure of the Report 10
2 BACKGROUND 11
2.1 DLP (Digital Light Processing) Technology 11
2.2 DDP2000 12
2.3 On Screen Display (OSD) 14
2.3.1 OSD Working States 15
2.4 REON Chip. 17
2.4.1 The Motivation 17
2.4.2 Why REON VX. 17
2.5 I^2C (Inter-Integrated Circuit) Bus 17
2.6 Problem Definition 19
2.7 Project Goals 20
3 SPECIFICATION & DESIGN 21
3.1 The outline design for the DDP2000 OSD 21
3.1.1 Development Tools and Resources 21
3.1.2 OSD structure 22
3.1.3 Menu, Item and Parts 23
3.1.4 Display menu 26
3.1.5 Adding New Item 29
3.2 Coding Work 30
3.2.1 Software Architecture 30
3.2.2 Application Code Modification 32
3.2.3 I^2C Bus 42
3.3 Co-Work 43
4 TESTING & RESULTS 46
4.1 Test Strategy 46
4.1.1 Tools and Resources 46
4.1.2 Testing & Result 48
5 CONCLUSIONS & FUTURE WORK 51
REFERENCE 52

LIST OF FIGURE

Figure 1: The OPTEA's logo 9
Figure 2: The Veo+ projector 9
Figure 3: DMD chip 11
Figure 4: Micromirrors 11
Figure 5: Color Wheel 12
Figure 6: The DDP2000 Component set 13
Figure 7: Software Architecture 13
Figure 8: An example of an OSD for a projector 14
Figure 9: ASIC Functional Block Diagram 15
Figure 10: OSD states followchart 15
Figure 11: Example of multiple menus 16
Figure 12: REON VX 17
Figure 13: Sample I2C Implementation 18
Figure 14: Structure of DDP2000 with Reon VX 19
Figure 15: DLP Composer™ Tool Suite Interface 21
Figure 16: OSD Design Tree 22
Figure 17: Structure of OSD 23
Figure 18: OSD Parts Window 24
Figure 19: OSD Menu Layout Window 24
Figure 20: Property Window 25
Figure 21: Button "Display" 25
Figure 22: Design tree of DispayMenu 26
Figure 23: DisplayMenu 26
Figure 24: KeystoneItem 27
Figure 25: Property of StaticText 28
Figure 26: KeystoneItem 28
Figure 27: Language Window 29
Figure 28: New Display menu with TESTItem 30
Figure 29: Reference Application Architecture 31
Figure 30: Logical Code Partitioning 32
Figure 31: The command from User to Hardware 33
Figure 32: The files had been changed 34
Figure 33: Example of the .h file 34
Figure 34: Display menu start flow diagram 37
Figure 35: Keystone slider adjustment flow diagram 41
Figure 36: Compile procedure 44
Figure 37: check all files to build final image file 45
Figure 38: Terminal V1.9b Window 46
Figure 39: Evaluation Board with DMD 47

Figure 40: Beagle I2C/SPI/MDIO Protocol Analyze 47
Figure 41: Beagle Data Center Software 47

LISTING

Listing 1: GUI_MsgHandler 35
Listing 2: Function GUI_DisplayMenuOnStart 36
Listing 3: Function GUI_DisplayMenuAdjustItemRight 38
Listing 4: Function REON_KeystoneSettings 43
Listing 5: Testing & Result 50

1 INTRODUCTION

1.1 About the Project

This project was conducted by the student named Jin Luo as Master Thesis project. This project was undertaken at OPTEA AB in Stockholm, Sweden for the duration of 25 weeks.

The aim of this project is to design a new OSD (On-Screen Display) for the user to using the new function. The project can be divided into 2 major tasks which are:

- Design a new outline (with new function buttons) for the OSD by using DLP composer.
- Code modification for the new OSD function.

1.1 About the OPTEA AB

Figure 1: The OPTEA's logo

OPTEA AB, a company based in Stockholm, Sweden was founded by Mr. Esteban Arboix in the year 2005. The company holds several patents and specializes in projection technologies and unique screen solution for custom applications. These include projection on moving cars, caravans, boats and yachts for various purposes such as marine chart plotting, Global Positioning System (GPS) navigation, digital entertainment and gaming.



Figure 2: The Veo+ projector

Holding to the motto, 'Project Anything Anywhere', Optea AB is actively developing a new kind of 12V Digital Light Processing (DLP) projector known as the Veo+ which is targeted for the marine, car and caravan market. With the dimension of just 215 x 60 x 100 mm and weighing less than 1 kg, this projector will be among the smallest in the market in its category. Designed for ultra-portability, this projector is also equipped with Bluetooth technology for wireless connectivity. More information about the company and its range of products can be found at www.optea.com.

1.3 Structure of the Report

Chapter 1 provides an introduction of the project and describes the organization of this report.

Chapter 2 provides the background of this project work. It will begin with introduction for the readers with DDP2000 system. The chapter then continues with the introduction on the OSD technology and the REON VX. The chapter ends with the problem statements and project goals.

Chapter 3 presents the OSD outline design process which includes the development tool introduction, OSD structure description and the new item's implementation. Then, DDP2000's software architecture is discussed and followed by code modification. Next, the way to make the results from the two parts co-work will be presented.

Chapter 4 shows the test strategy and then evaluates the project based on the set of goals in Chapter 2. This chapter also outlines the method of loading .img file into to flash memory of Evaluation Board.

Chapter 5 concludes the project and also discusses several ideas on possible future works.

2 BACKGROUND

It’s necessary to provide an overview of DLP technology, DDP2000 system (includes hardware and software) before stepping further into the discussions, and then I will explain motivation of the implementation of third party device REON VX, and finally present the challenge of this project.

2.1 DLP (Digital Light Processing) Technology

The DLP ® (Digital Light Processing) technology which had been used by DDP2000 was invented in 1987 by Texas Instruments.

At the heart of every DLP ® projection system is an optical semiconductor known as the DMD® chip



Figure 3: DMD chip

The DMD® chip is probably the world's most sophisticated light switch. It contains a rectangular array of up to 2 million hinge-mounted microscopic mirrors; each of these Micromirrors measures less than one-fifth the width of a human hair.

A DMD® chip's Micromirrors are mounted on tiny hinges that enable them to tilt either toward the light source in a DLP® projection system (ON) or away from it (OFF)-creating a light or dark pixel on the projection surface.

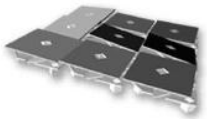


Figure 4: Micromirrors

The bit-streamed image code entering the semiconductor directs each mirror to switch on and off up to several thousand times per second. When a mirror is switched on more frequently than off, it reflects a light gray pixel; a mirror that's switched off more frequently reflects a darker gray pixel.

In this way, the mirrors in a DLP® projection system can reflect pixels in up to 1,024 shades of gray to convert the video or graphic signal entering the DMD® chip into a highly detailed grayscale image.

But only mirrors cannot do the projection. Light source is also necessary. The white light which is generated by the lamp in a DLP® projection system passes through a color wheel as it travels to the surface of the DMD® chip. The color wheel filters the light into red, green, and blue, from which a single-chip DLP® projection system can create at least 16.7 million colors. And the 3-chip system found in DLP Cinema® projection systems is capable of producing no fewer than 35 trillion colors.

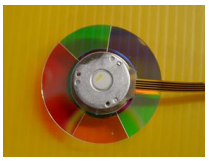


Figure 5: Color Wheel

The on and off states of each micromirror are coordinated with these three basic building blocks of color (RGB: Red, Blue, and Green). For example, a mirror responsible for projecting a purple pixel will only reflect red and blue light to the projection surface; our eyes then blend these rapidly alternating flashes to see the intended hue in a projected image.

2.2 DDP2000

The DDP2000 is the first DLP data processor component to integrate all of the system image processing and system control, as well as DMD data formatting, on to a single integrated circuit(IC).

The DDP2000 Component set consists of the DDP2000 projector controller ASIC, a Double Data Rate DMD (XGA or SVGA resolution), and the DAD1000 analog ASIC. The DDP2000 reference design includes these components along with peripheral devices which form the electronics portion of a projector design [6].

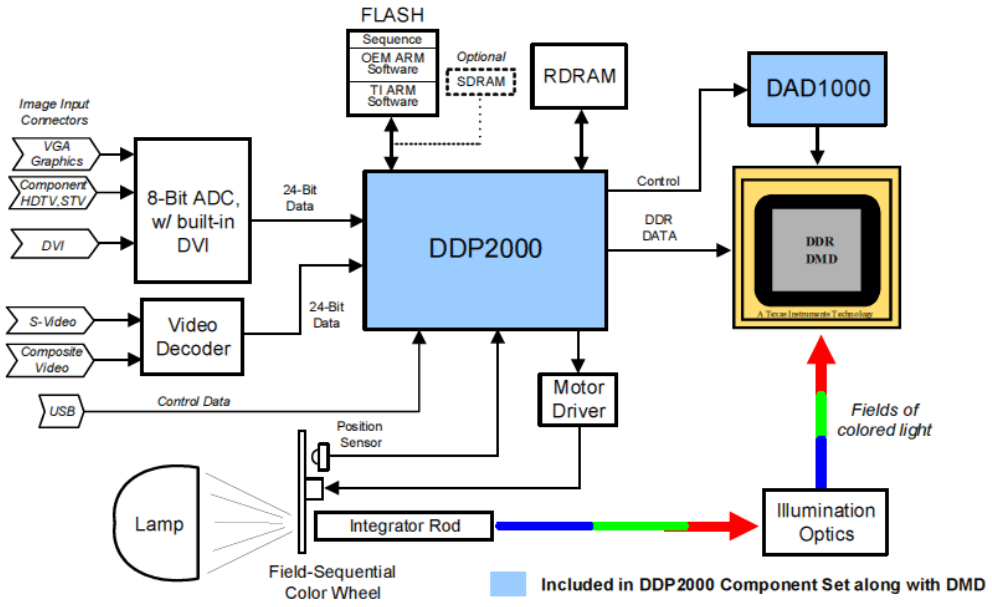


Figure 6: The DDP2000 Component set

From the figure 6 we know, DDP2000 receives data and signal from DVI and decoder, and reads data from FLASH and RDRAM. Once the necessary data is arriving, DDP2000 and DAD1000 will control the operation of the whole system.

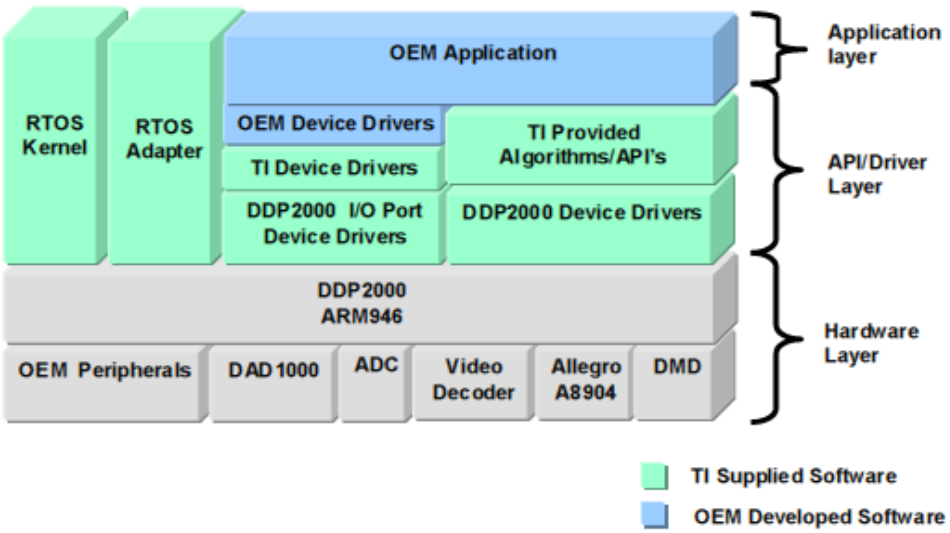


Figure 7: Software Architecture

Besides the hardware, the software also had been provided. Figure 7 shows the hardware and software block. The software parts include kinds of drivers and API. We also notice that there is an “Application Layer”, and it’s a part for the OEM developer.

2.3 On Screen Display (OSD)

An on-screen display (abbreviated OSD) is an image superimposed on a screen picture, commonly used by modern televisions, VCR, and projector to display information such as volume, channel, and time. The OSD for projector should enable the user to change the settings of the projector, like adjusting the brightness, contrast, color and keystone of the displayed image, and setting the language, setting noise level or switching on/off the projector's light source. The implementation of the OSD functionality is taken care of by the data processor inside the projector.

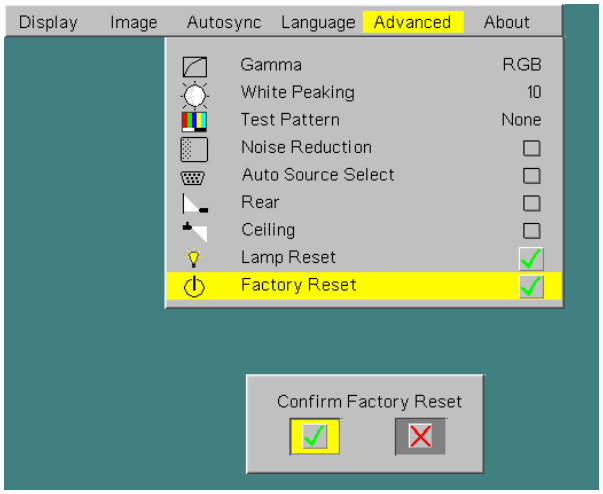


Figure 8: An example of an OSD for a projector

Then the end user can interact with the OSD menu to adjust the settings of a projector by means of only a few buttons. On a typical projector, these buttons can be found integrated on the projector's outer casing or supplied as an additional keypad.

The OSD menu implementation of the Veo+ projector is handled by the DDP2000. This data processor has general purpose input/output (GPIO) pins that can be connected to a keypad to control the OSD menu.

2.3.1 OSD Working States

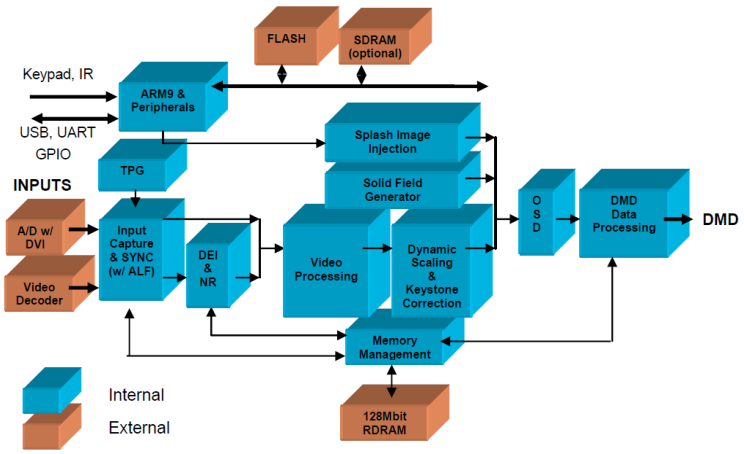


Figure 9: ASIC Functional Block Diagram

Figure 9 shows the ASIC Functional Block Diagram of DDP2000. Projector will do a series of operations when it starts. At first, the system will capture the input, video processing, and then dynamic scaling & keystone correction. At last all the system will send the data to DMD after the data processing.

After the projector starts, the OSD will not show until the end user calls the OSD. Figure 10 shows there are 4 working states for the OSD, waiting, initialization, stand by, and adjustments. When the projector finished start, OSD will stay in “Waiting” until the start order from end user. Once OSD receive “start” order, the OSD will go to “initialization”

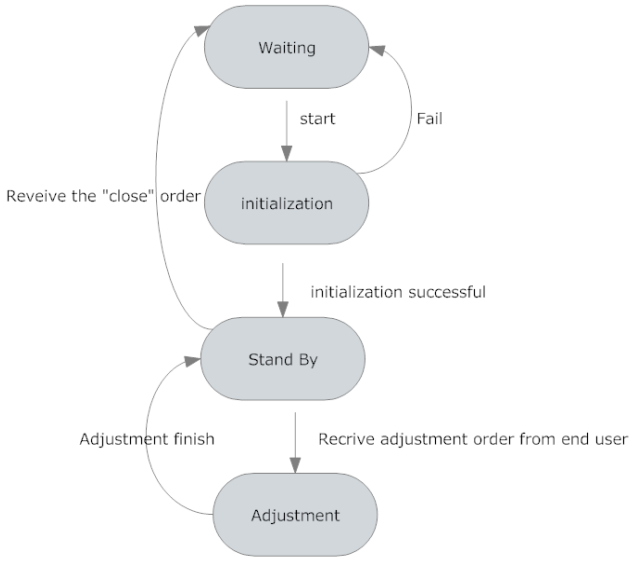


Figure 10: OSD states flowchart

With the OSD data loaded into flash memory, the OSD API controls the loading and display of his data on the screen. The application software can make calls to the OSD API to control loading, unloading, text strings and slider bar values. Depending on the size of the font, dynamic control over the font memory content is managed by the OSD API as built by the OSD tool. The OSD tool will intelligently partition characters into font subsets as dictated by the menus which can be on screen at any one time.

The function OSD_LoadMenu brings a menu into a temporary data buffer in dynamically allocated memory. At this time, the application will adjust the menu to the current state of the system. It can adjust slider bars and value fields corresponding to the current values set by the user.

If one OSD design is requires multiple menus are on screen at the same time, the application code must adhere to the following rule: unloading of the menus must be done in the reverse order of loading. For example, Menu 1 is loaded and displayed, followed by the loading and display of Menu 2, then Menu 3. When the application disables the display of the menus, Menu 3 must be unloaded first, followed by the unloading of Menu 2 then Menu 1.

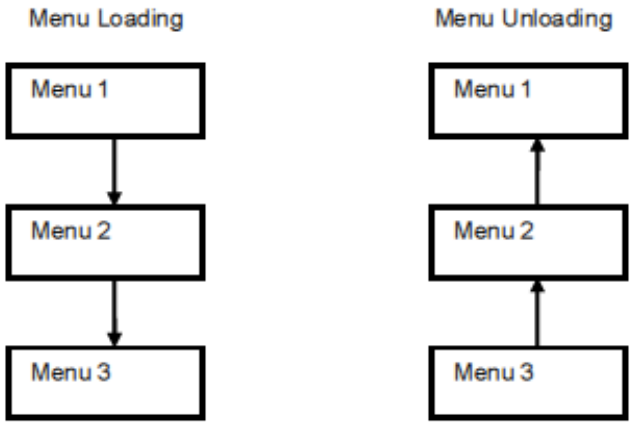


Figure 11: Example of multiple menus

But only the menu loading is not enough, and the necessary data for the item value should be read from EEPROM. When the “value reading” successful, the OSD will go to “Stand By”.

In the “Stand By”, the OSD will wait for the new command from the end user. If the command is closed, OSD will back to “wait” state. But if the command is “Adjustment”, OSD will go to “Adjustment”, and do a series operations for responds the adjustment, and store the new value of the parameter in the EEPROM. Whether the operations which happen during the “Adjustment” successful or not, OSD will back to “stand by” and waiting for a new command.

2.4 REON Chip.

2.4.1 The Motivation

The appropriate space (include appropriate distance, appropriate platform), appropriate temperature and enough power supply are necessary for the normal projector do the projection. But the truth is we cannot always have them. For the space issue, DDP2000 can provide Keystone functions to adjust the output image for fit the space condition, but of course, the DDP2000's graphic processor can only support a limited range for the adjustment of Keystone. For some extreme narrow condition, we need more adjustment, so using another powerful graphic processor is a good choice.

2.4.2 Why REON VX.

Reon VX is a highly integrated SOC which is targeted for high-volume display applications. Reon VX is the next generation of advanced video processor from Silicon Optix and delivers the Hollywood Quality Video (HQV) experience to consumers at a cost effective price point. Reon VX is the first video processor in its class to include codec noise reduction and color spectrum controls as standard on-chip features



Figure 12: REON VX

The Reon VX can increase the limitation of adjustment of the keystone function, and provides some extra functions like "Anyplace". Since Reon VX has master design and slave design, we chose slave design for DDP2000

2.5 I^2C (Inter-Integrated Circuit) Bus

When connecting multiple devices to a microcontroller, the address and data lines of each device were conventionally connected individually. This would

take up precious pins on the microcontroller, result in a lot of traces on the PCB, and require more components to connect everything together. This made these systems expensive to produce and susceptible to interference and noise. [7]

To solve this problem, Philips developed Inter-IC bus, or I2C, in the 1980s. I2C is a low-bandwidth, short distance protocol for on board communications. All devices are connected through two wires: serial data (SDA) and serial clock (SCL).

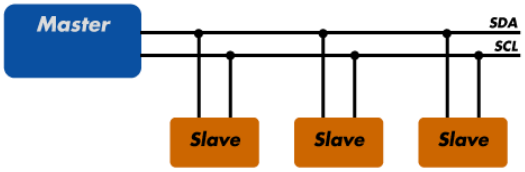


Figure 13: Sample I2C Implementation

Regardless of how many slave units are attached to the I2C bus, there are only two signals connected to all of them. Consequently, there is additional overhead because an addressing mechanism is required for the master device to communicate with a specific slave device.

Because all communication takes place on only two wires, all devices must have a unique address to identify it on the bus. Slave devices have a predefined address, but the lower bits of the address can be assigned to allow for multiples of the same devices on the bus.

The DDP2000 provides an I^2C controller (master) to communicate with peripheral devices. Function calls are available to read and write from external devices.

Configurable parameters include:

- SCL Clock rate (in Hz)
- SCL spike filtering sensitivity
- Address mode (7 bit or 10 bit)

The DDP2000 API creates a semaphore for protected sharing of the I^2C controller. This protection is required because multiple tasks may require the using of the I^2C controller simultaneously. If these tasks blindly request transfers simultaneously, it may writes to the controller registers in an inappropriate way. It has been proven that if this situation occurs, the I^2C

controller will enter a state from which it cannot exit. The only way to recover from this state is a system reset from a power cycle. When the I^2C controller is initialized with a call to I^2C_Init, the semaphore is created. Whenever the IC controller is used by the application, it should reserve the semaphore first. If the semaphore is in use, the application can choose to wait, in which case the RTOS will reschedule the running task and pass control to the next waiting task.

In this project, the DDP2000 will be chosen as the master device and the Reon VX will be chosen as the slave device. The communication between master and slave will be based on the I^2C bus.

2.6 Problem Definition

Figure 14 shows the structure of Veo+, DDP2000 + Reon VX.

We have a complete DDP2000 system. How to make the end user of projector use these functions is the main mission, so we have to modify the OSD.

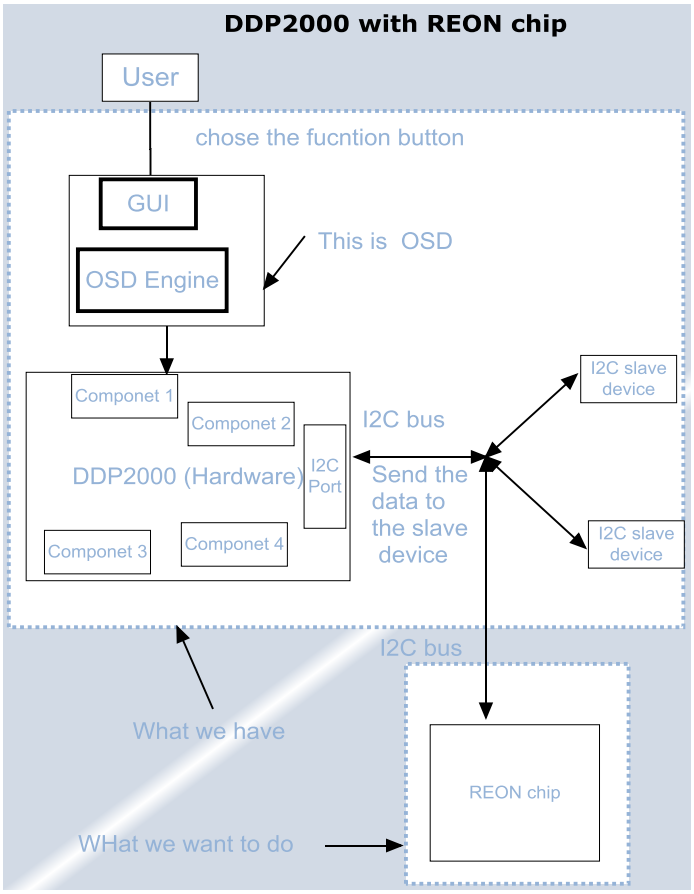


Figure 14: Structure of DDP2000 with Reon VX

2.7 Project Goals

From the introduction in Section 1.1, the project is to design a new OSD for the projector Veo+. The new OSD design can be divided into two major parts which are: 1) Design a new outline (with new function buttons) for the OSD by using DLP composer; 2) Change the source code in order to make the new item for the new function good working.

The new OSD outline (OSD Outline is the actual characters, icons, windows, slider bars and other features that make up a menu and are displayed for the user to see) design for the OSD should be able to achieve the following goals:

- It should be in the form of DLP Composer project (.project)file
- Add one item (a slider under menu Display) for the function Keystone from Reon VX with correct property. This button should be a slider and can be change the value by press the left and right buttons.

The modification of the code should enable to reach the following goals.

- It should be in the form of ADS (ARM Developer Suit) project (.mcp) and language C (.C) file. These files should be able to compile by ADS and generate an application binary (.bin) file for the DLP Composer.
- The range for the new slider should be from -100 to +100.
- The DLP Composer project (.project) file should be compliable by DLP Composer with the application binary (.bin) file from ADS, and generate a Flash Image File (.img), and these files should be good work on the Evaluation Board.

3 SPECIFICATION & DESIGN

3.1 The outline design for the DDP2000 OSD

The OSD consists of three parts. 1) The actual characters, icons, windows, slider bars and other features that make up a menu and are displayed for the user to see (This is the OSD Outline); 2) An OSD design consists of application software that loads the menus in response to inputs from the keypad or IR remote and changes projector settings based on these inputs; 3) The OSD API which responds to calls from the application software and retrieves menus and other OSD elements from the flash. The OSD API also operates on OSD elements to change slider bars, text values and lists. The application must implement a GUI system (the code which responds to user input) and this is not provided by the OSD API or DLP Composer.

3.1.1 Development Tools and Resources

We will use the DLP Composer™ Tool Suite for the new OSD outline design. The DLP Composer™ Tool Suite is a software package that aids manufacturers in designing, developing, and producing DLP™-based projectors faster and with better results. The OSD Design tool of DLP Composer™ Tool Suite creates the visual part of a projector's On-Screen-Display. An OSD Design consists of the following pieces: Global Options, Color Palettes, Fonts (collections of character and icon shapes, or "glyphs"), Language-specific Strings, and Menu layouts.

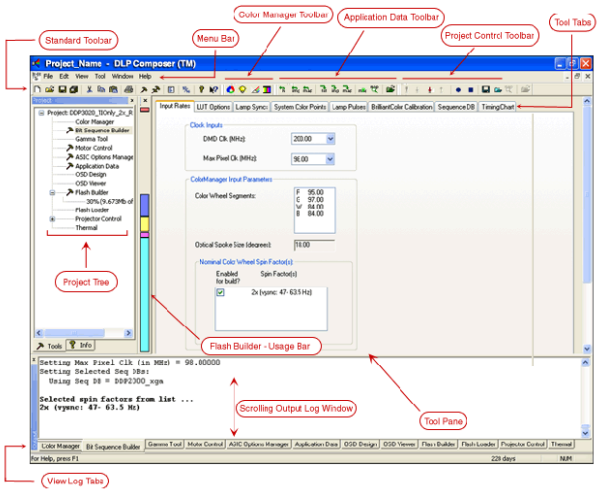


Figure 15: DLP Composer™ Tool Suite Interface

3.1.2 OSD structure

Before we start the OSD outline design work, we should understand the structure of the OSD. Figure 16 shows an example of The OSD Design Tree. It is shown on the Project window, in the OSD tab

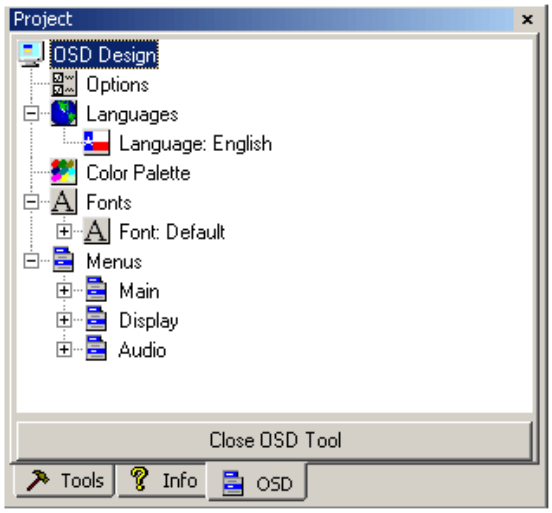


Figure 16: OSD Design Tree

There are five elements for the OSD design, options, colors, languages, fonts and menus.

- Options: The “Option” define the global values, like Default char spacing, Default line height, Default baseline offset and the names of the style for from the first font to the fourth font used by a language etc.

- Colors: The color palette window shows the colors in the current color palette. Each OSD Design must have no less than one (up to 255) color palette. When one color palette changed, all related icons are updated immediately.

- Languages: The language window is a powerful aspect of the tool. The designers can populate a table of expressions in which each column is for a different language and each row a different expression. Each row has an index number. Using this feature, a menu created in one language becomes available in all languages used in the OSD. Menus should be sized to accommodate the longest text string for an expression in any Language. The string for any language is known as a language specific string.

- Fonts: The Font/Icon graphically shows the glyphs (or character/icon

shapes) that are defined by an OSD font.

- Menus are building blocks of the OSD. They are created by the designer using the menu layout window. In this mode, the designer may size the menu, position it on the screen, and place text strings, icons and OSD parts (sliders, text and lists) onto the menu. Various text and parts may be combined on a line to form an item.[8]

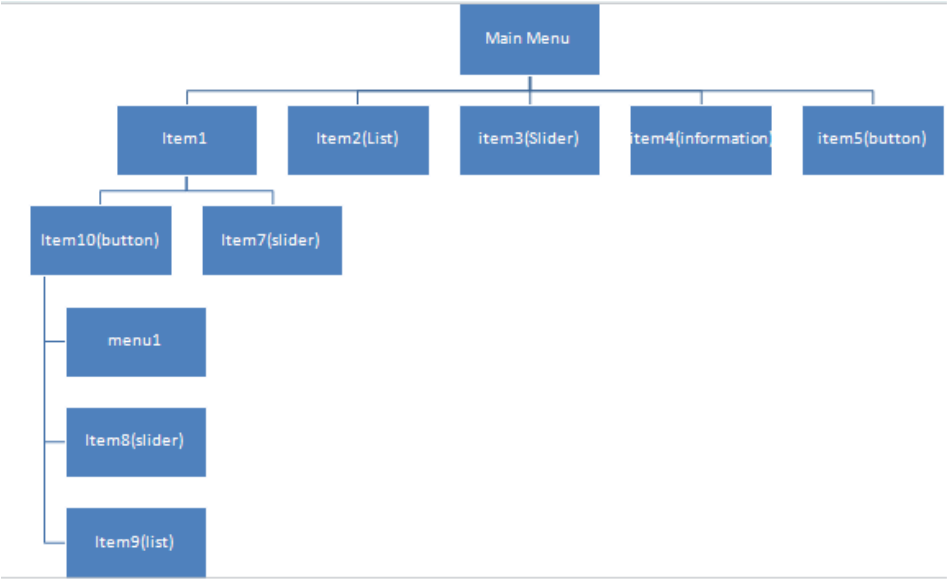


Figure 17: Structure of OSD

3.1.3 Menu, Item and Parts

The OSD has two important parts: Menu and Item. The menu contains vertical menu and horizontal menu. In figure 17, “main menu” is a horizontal menu and “menu 1” is a vertical menu. In an OSD design, there may have many menus, but only one main menu.

There are many kinds of items, sliders, lists, chose buttons and a buttons for expanding another menu. Item has many parts for their properties. Different kind of item has different parts, and DLP composer supplies several kinds of parts. We can check all of them from the OSD Parts Window.

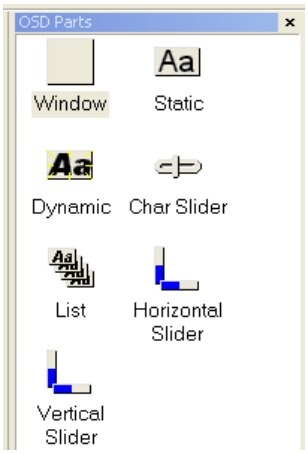


Figure 18: OSD Parts Window

The OSD Parts Window displays the parts which are used for design a menu. The actual contents of the parts window is controlled by parts files (*.parts) located in the OSD subdirectory of designer's DLP Composer library, or in designer's currently open project directory.

By default, the OSD Parts Window shows the icons (as above) that represent the seven basic types of menu parts, and we will discuss the property of the parts we will use later.

The OSD Menu Layout Window is used to add and arrange OSD parts to design a menu.

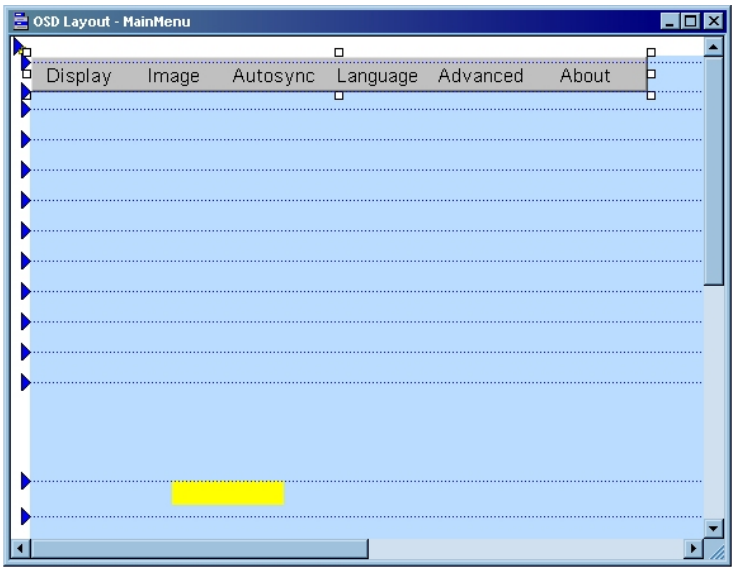


Figure 19: OSD Menu Layout Window

Since the menu we have chose to adding the new item in is the "Display Menu", thus the "Display Menu" is our object. In figure19, there is an example

of OSD Menu Layout Window to shows the main menu. We can find a button name “Display”, and this button has been used for expand the “Display Menu”.

The button “Display” is an item belonging to “Main Menu”. In order to check the property of the item (because each piece of the OSD Design has one or more properties), we need support from Property Window.

Property Window is used for displaying the properties of the currently selected item of the design tree or editor window.

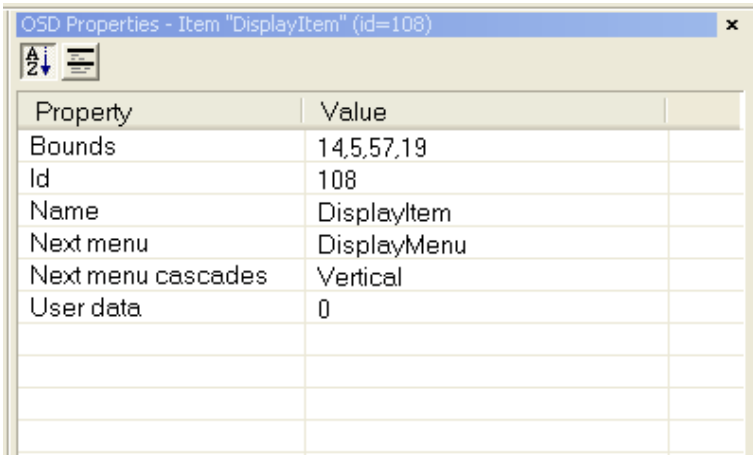


Figure 20: Property Window

Figure 20 is a sample of Property Window with the property of item. As seen, the property “Bounds” has 4 numbers, and the first two numbers 14 and 5 ensure the position of the button by define the top-left corner of the button.

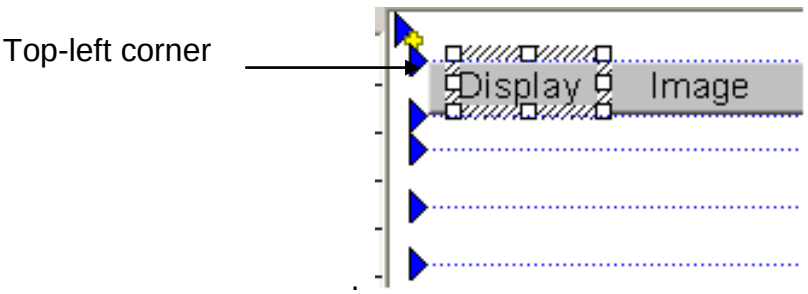


Figure 21: Button “Display”

The last 2 number define the size of the button, like here, the size of the button is 57 X 19.

Every item and menu has its own “id”. Id is the permanent, unique identifier for the item and menu which be assigned by the composer.

“Next menu” will give the information to the system that this button will

expand a menu if it has been selected. Because the button “Display” in main menu is used for expand the “DisplayMenu” (this is the name of display menu property, as the “name” in item property). Here should be “none” if this button is not use for expand a new menu. The property also define the cascades of the next menu, we can find it from the property of the next menu.

User data is a 32-bit integer value that are using for any purpose and it is accessible at runtime using OSD_GetItemMenuData. The default value is 0

3.1.4 Display menu

From the OSD design tree, we can check the content of the menu.

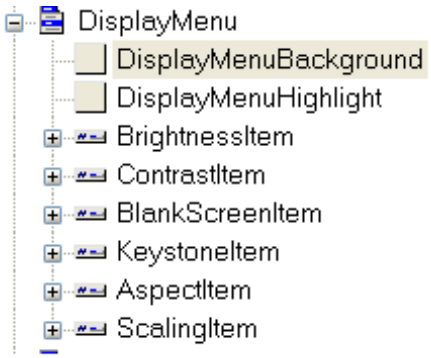


Figure 22: Design tree of DispayMenu

As the figure 22 shows, there are six items for displaying menu. The goal of this project is adding one item into this menu. The parts name “DisplayMenuBackground” is a “window”. A window is a simple rectangular area. It can optionally have a raised or depressed 2-pixel border around the rectangle. One thing I want to mention is that the parts “Window” **don’t** have any ID.

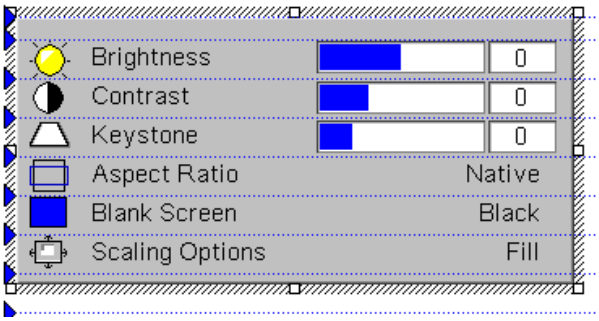


Figure 23: DisplayMenu

Figure 23 shows the “DisplayMenu”, and include the items: Brightness,

Contrast and Keystone (Slider), and the items Aspect, Blank Screen and Scaling Options (List). Because the item we want to add into this menu is a slider, we use Keystone as the example in our project.

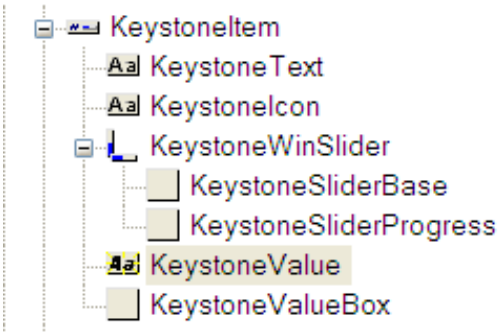


Figure 24: KeystoneItem

“Keystoneitem” shows the keystone’s value. In figure 24, there are two parts “KeystoneText” and “KeystoneIcon”, both of them are Static Text which is a kind of field for showing a certain value, and it is unchangeable during the runtime.

The properties of Static Text, “Langtext” and “Text” do the job of showing the text of the parts. We can choose a language-specific text string from the list for the property of “Langtext”. All strings from the current language are shown. We use the Layout Editor menu to change languages. If the "Text" property has a value, then this property is ignored. The property “Text” is useful for non-language specific text or icon data. Designers can enter any text. Once the field has focus, designers can click on one-or-more characters or icons in the Font window.

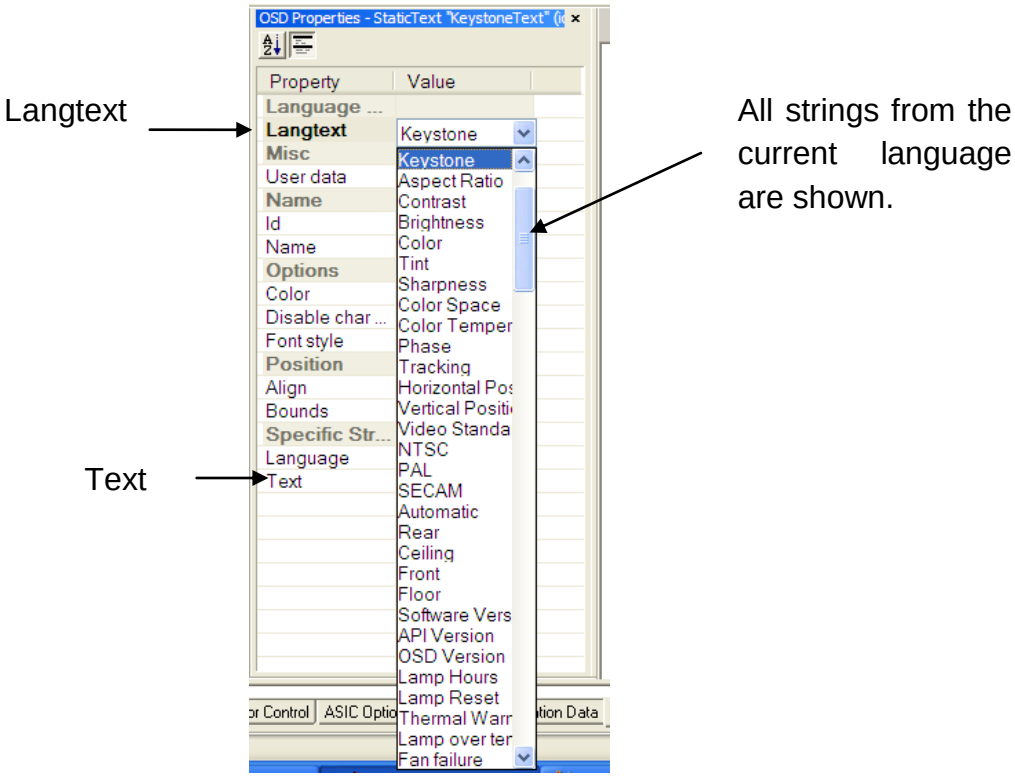


Figure 25: Property of StaticText

A Character slider is necessary for the “KeystoneItem”. A character slider field is made up of several different icons and each icon which is represents a part of the slider. Here, the slider has two parts, “Keystonesliderbase” and “Keystonesliderprogress”.

Also, there should be a number to show the value of the Keystone, The Dynamic Text, “Keystonevalue” will show the number of the keystone value. Dynamic Text is a field whose value is changeable at runtime. The value can be either numeric or text (or even icons). It can have a default value which will be displayed unless the OSD API is called to set a new value.

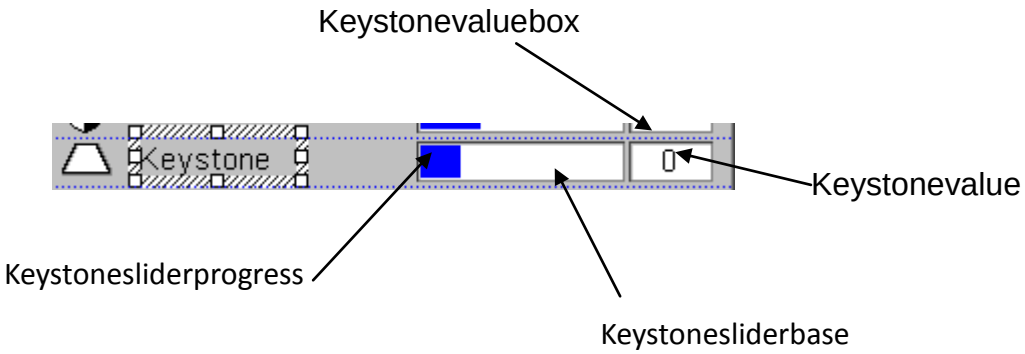


Figure 26: KeystoneItem

3.1.5 Adding New Item

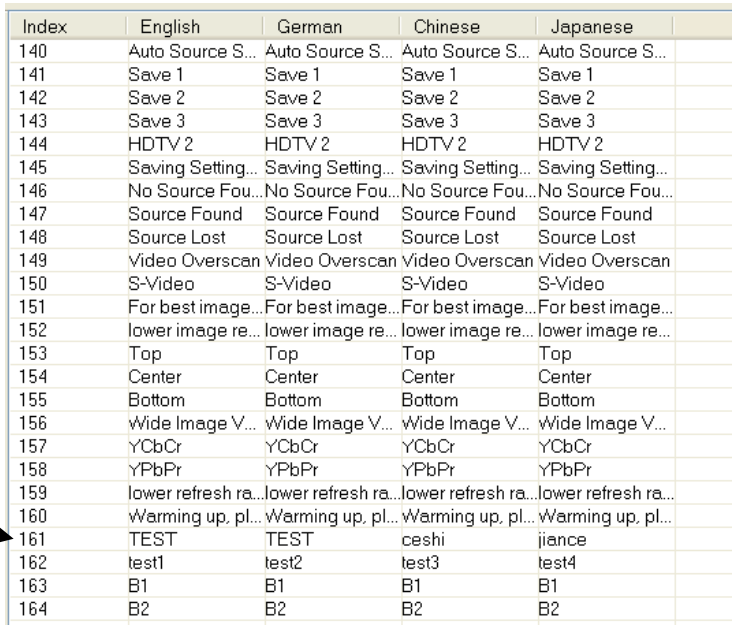
After studying the “Keystoneitem”, we are ready to add a new item into the display menu. The operation is following.

- Change the size of “DisplayMenuBackground” in order to give enough space for the new item.
- Add a Static Text “TESTIcon”, because we don’t have the special icon for the new item, so we use the icon which had been used for the “KeystoneItem”.
- When adding a new “TESTText”, we should choose an appropriate “Langtext”. We can choose a string from the current language or add a new one for the new items.

  To add a text string to a language, open the language window, and press the  toolbar button, or right-click on the language window, and choose Add String.

  Designers can also add text strings to a language using the Property Window, when editing the “LangText” property of a Static Text, Dynamic Text or List field.

The new text



Index	English	German	Chinese	Japanese
140	Auto Source S...	Auto Source S...	Auto Source S...	Auto Source S...
141	Save 1	Save 1	Save 1	Save 1
142	Save 2	Save 2	Save 2	Save 2
143	Save 3	Save 3	Save 3	Save 3
144	HDTV 2	HDTV 2	HDTV 2	HDTV 2
145	Saving Setting...	Saving Setting...	Saving Setting...	Saving Setting...
146	No Source Fou...	No Source Fou...	No Source Fou...	No Source Fou...
147	Source Found	Source Found	Source Found	Source Found
148	Source Lost	Source Lost	Source Lost	Source Lost
149	Video Overscan	Video Overscan	Video Overscan	Video Overscan
150	S-Video	S-Video	S-Video	S-Video
151	For best image...	For best image...	For best image...	For best image...
152	lower image re...	lower image re...	lower image re...	lower image re...
153	Top	Top	Top	Top
154	Center	Center	Center	Center
155	Bottom	Bottom	Bottom	Bottom
156	Wide Image V...	Wide Image V...	Wide Image V...	Wide Image V...
157	YCbCr	YCbCr	YCbCr	YCbCr
158	YPbPr	YPbPr	YPbPr	YPbPr
159	lower refresh ra...	lower refresh ra...	lower refresh ra...	lower refresh ra...
160	Warming up, pl...	Warming up, pl...	Warming up, pl...	Warming up, pl...
161	TEST	TEST	ceshi	jiance
162	test1	test2	test3	test4
163	B1	B1	B1	B1
164	B2	B2	B2	B2

Figure 27: Language Window

- Draw a slider icon from the OSD parts window to the menu. It’s a package

with “TESTSliderBase” window and “TESTSliderProgress” window. After adjusting the size and position, this step is finished.

- Draw a Dynamic Text to the menu for the “TESTValue” and a window for the “TESTValueBox”

- The last step is grouping all the new parts we created for the new item into an item “TESTItem”.

A new DDP2000.h file (with the new information of the change) for the source code had been generated after the compilation.

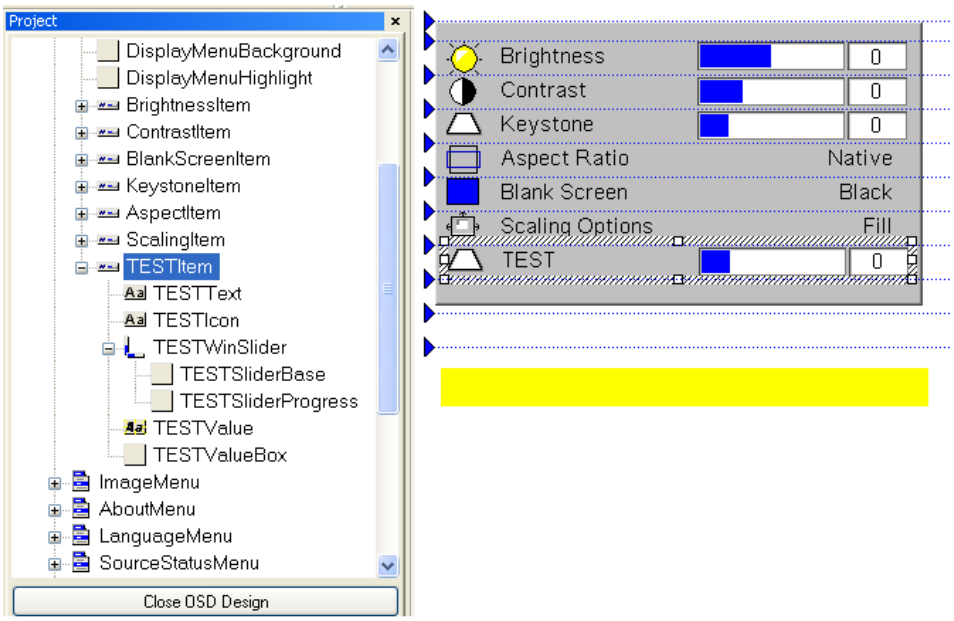


Figure 28: New Display menu with TESTItem

3.2 Coding Work

3.2.1 Software Architecture

The coding work for this project is about the OEM Application and DDP2000 API.

- The DDP2000 API has been implemented to support an event driven architecture. That is, when an interrupt occurs, the interrupt handler does minimal processing and sets an event through the RTOS. An RTOS task, which is waiting for the event, is then awakened to operate in response to the original event. The RTOS is required to activate tasks according to the priority of tasks as assigned by the application programmer.

Some tasks are provided by the DDP2000 API. This is necessary to protect intellectual property for the ways which TI has learned to control the system. The priorities and task stack sizes are assigned by TI at their recommended values. These tasks must coexist with OEM tasks in the end application.

The API protects access to registers when necessary. If an API function is required to modify a register value, it will guard the modification by disabling interrupts first. These critical code sections are employed when a read - modify - write operation occurs. If an interrupt were to occur after the modification and change the contents of the affected register, then the following write would overwrite new information.

API development was completed with Metrowerks CodeWarrior for ARM Developer Suite v1.2. The compiled libraries are to be included with an OEM developed application and linked to produce the final executable.

- TI provides application code serves as a reference for the projector software designer. The basic architecture of the reference application is shown in Figure 29.

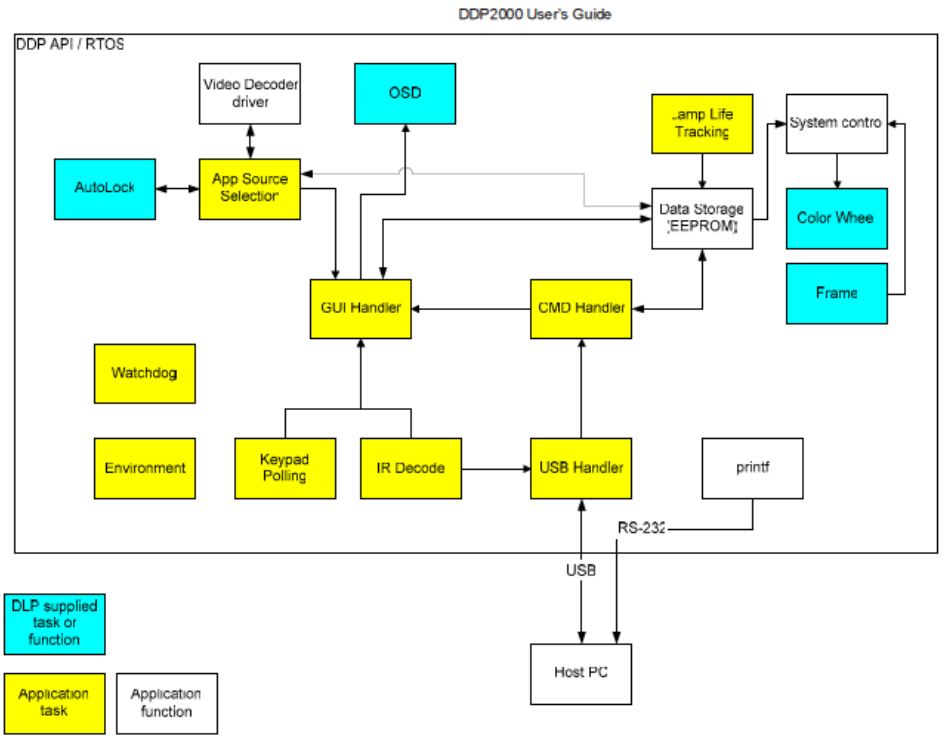


Figure 29: Reference Application Architecture

Reference application development was completed in Metrowerks CodeWarrior for ARM Developer Suite v1.2 (ADS). Code is partitioned

logically as shown in Figure 29, and all source code is contained in a single directory. The project manager of the IDE allows the code to be logically segmented for grouping by function or task. This capability has been used by TI in the reference design application and can be seen in the figure 29.

Application code was written in a modular style with most related functions occupying only one source code file. For example, code which services the IR receiver ports is contained only within the file ir_decode.c and ir_decode.h. Functions within modules are prefixed by the module name, for example, IR_InitPort or GUI_MsgHandler.

The application is built by the ARM assembler, compiler and linker from within the integrated development environment (IDE). The link order can be specified on the 'Link Order' tab of the project window..

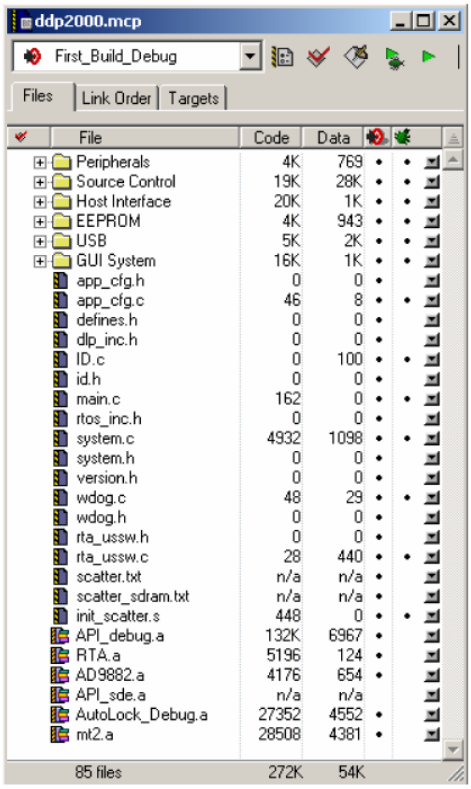


Figure 30: Logical Code Partitioning

3.2.2 Application Code Modification

Once the new OSD outline is finished, some code modification for the adding new item should be done. To understand the code structure is important for the code modification.

The OSD consists of three parts. First are the actual characters, icons, windows, slider bars and other features that make up a menu and are displayed for the user to see. Secondly, an OSD design consists of application software that loads the menus in response to inputs from the keypad or IR remote and changes projector settings based on these inputs. Third is the OSD API which responds to calls from the application software and retrieves menus and other OSD elements from the flash. The OSD API also operates on OSD elements to change slider bars, text values and lists. The application must implement a GUI system (the code which responds to user input), and this is not provided by the OSD API or DLP Composer.

The main modification work is on **Guimen_display.c** file, and the other file we modified is **DDP2000_OSD.h**

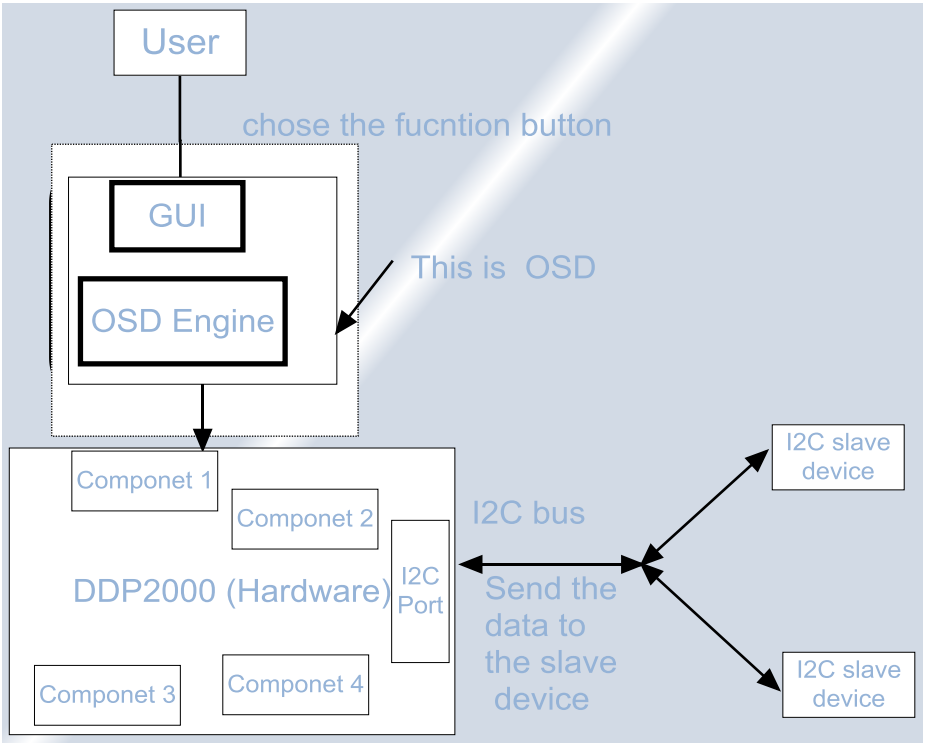


Figure 31: The command from User to Hardware

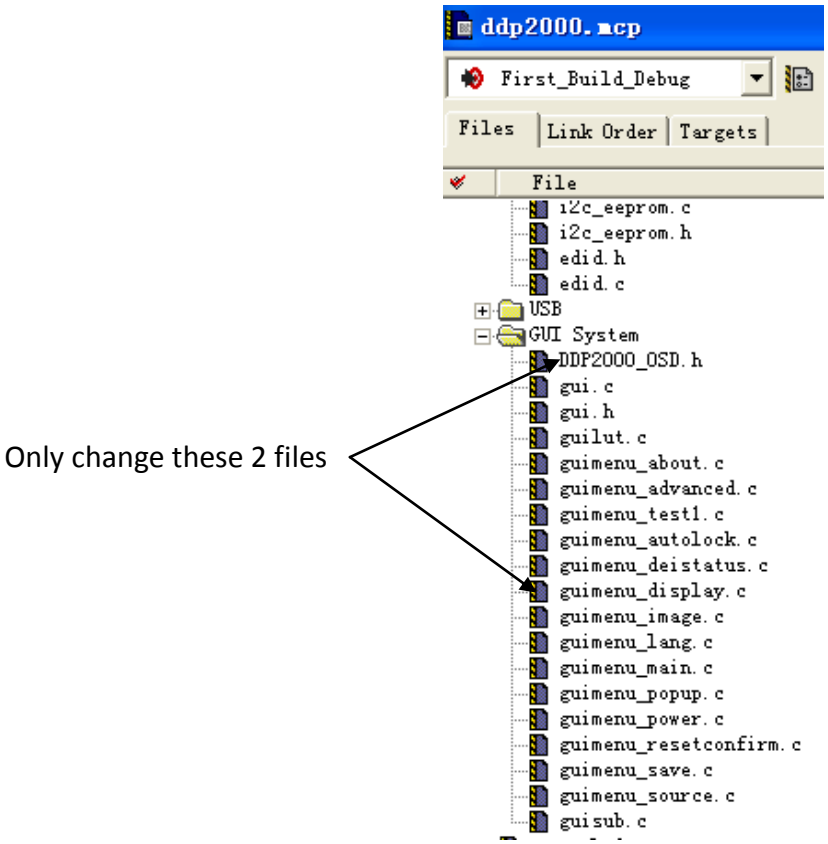


Figure 32: The files had been changed

- **.h file modification**

In section 3.1.5 we had discussed that there is a new **.h** file will be generated from the DLP Composer. The **.h** file define the name (is not the name we can saw in the OSD design, but the name for the software) of the menu, item and parts with the ID number of the OSD design. The new .h file will include the new information of the new item. We have to copy that information to the old **DDP2000_OSD.h** file which have be use for the reference design.

```
#define WIND_MAINHIGHLIGHT                  1
#define ITEM_DISPLAYITEM                    108
#define TEXT_DISPLAYTEXT                    2
#define ITEM_IMAGEITEM                      110
#define TEXT_IMAGETEXT                      3
#define ITEM_LANGUAGEITEM                   111
#define TEXT_LANGUAGETEXT                   5
```

Figure 33: Example of the .h file

The “handler” which had been defined in **gui.c is** used for read the ID number of the item for the software

```
void GUI_MsgHandler(void)
```

```
{
    int08 status;
    uint16 msg;
    uint08 index;
    DDP_DISPLAY_SOURCE source;
    static const uint08 task_running[] = "GUI task now running.\r\n";
    static const uint08 task_abort[] = "GUI mailbox not created, aborting task
startup.\r\n";
    static const uint08 task_abort2[] = "OSD Init failed, aborting task
startup.\r\n";
    static const uint08 keypress_up[] = "Keypress: up\r\n";
    static const uint08 keypress_down[] = "Keypress: down\r\n";
    static const uint08 keypress_left[] = "Keypress: left\r\n";
    static const uint08 keypress_right[] = "Keypress: right\r\n";
    static const uint08 keypress_leftright[] = "Keypress: left and right\r\n";
    static const uint08 keypress_menu[] = "Keypress: menu\r\n";
    static const uint08 keypress_power[] = "Keypress: power\r\n";
```

Listing 1: GUI_MsgHandler

- **“guimenu_display” file modfication**

Once the “display menu” had been required open, the file “guimenu_dispay” will define all the operations for this menu.

Before the menu expand, function GUI_DisplayMenuOnStart will do the pre-process for the menu.

```
void GUI_DisplayMenuOnStart(void)
{
#ifndef SDE
    int08 temp;
     int08 Test =0; // Set the default value of the slider
    SourceType CurrentSource = SRC_GetCurrentSource();.
.
.
.
    if (EE_GetData(&(pEEPROM->UserMachine.Keystone), TRUE, 1, (void*)&temp) ==
PASS)
        GUI_InitSliders(WSLD_KEYSTONEWINSLIDER, temp, -20, 20,
DTXT_KEYSTONEVALUE);
```

```
GUI_InitSliders(WSLD_TESTWINSLIDER, Test, -100, 100,    DTXT_TESTVALUE); .
// This function had been used for the TEST item slider initial. Here the limitation is -100 to 100.
.

    if (EE_GetData(&(pEEPROM->UserMachine.UserPortSettings[CurrentSource].AspectRatio), TRUE, 1, (void*)&temp) == PASS)
        GUI_InitLists(LIST_ASPECTLIST, (uint08)temp);

    if (EE_GetData(&(pEEPROM->UserSystem.BlankScreenColor), TRUE, 1, (void*)&temp) == PASS)
        GUI_InitLists(LIST_BLANKSCREENLIST, (uint08)temp);
.
.
.

    GUI_HighlightFirstItem();
}
```

Listing 2: Function GUI_DisplayMenuOnStart

Function GUI_DisplayMenuOnStart will read the value from the EEPROM (It is an IC slave device for store the variable of DDP2000, and the system will assign an address for the variable of the OSD in the EEPROM.) for each variable. EE_GetData(&(pEEPROM->UserMachine.Keystone), TRUE, 1, (void*)&temp) will get the Keystone value from the EEPROM and then store it in the "temp". This function will return a value, "pass" or "fail" to show this operation successful or not. If the return is "Pass", it is means the correct value will be used for the system, otherwise the value maybe an incorrect number, and, of course, cannot be used, and GUI will show 0 for the value.

Then the function GUI_InitSliders(WSLD_KEYSTONEWINSLIDER, temp, -20, 20, DTXT_KEYSTONEVALUE) had been called to initial the slider of the menu. WSLD_KEYSTONEWINSLIDE will show which slider should be changed, and the temp is the exactly value for the item. "-20, 20" define the up limitation and down limitation of the slider, and DTXT_KEYSTONEVALUE will show which number box to present this value.

The initial operating for the list has the same condition, and the designer define the members of list by DLP Composer. These members will be stored in the EEPROM, and every time the projector start, function GUI_InitLists will

read the current value for the list from EEPROM.

When all the initial operating finished, function GUI_HighlightFirstItem() will highlight the first item of this menu.

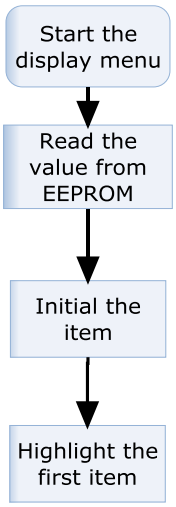


Figure 34: Display menu start flow diagram

Since the new item does not has the address in the EEPROM, designer have to define an integer Test = 0. This value is used for the initial operating for the TEST slider, and it should be start from 0 very time start the system. Here we also define the Range for the value for the item “TEST” from -100 to 100.

Function GUI_DisplayMenuAdjustItemRight defines the operating of increase the value of the item.

```
void GUI_DisplayMenuAdjustItemRight(void) //Here, “Right” it’s mean the increase the value
{
    uint16 normal_val, slide_val;
     uint08 list_size, list_val;
     int08 temp = 0;
switch(guiTopMenu->itemID)
    {    .
.
.
.    .
          case ITEM_KEYSTONEITEM:
               if (EE_GetData(&(pEEPROM->UserMachine.Keystone), TRUE, 1,
(void*)&temp) == PASS)
               {
                    /* don't do anything if already at top end of adjustment range */
                    if (temp == 20)
```

```
                    break;
                else
                    temp++;
                if (SRC_LoadKeystoneSettings(TRUE, temp) == PASS)
                {
                    GUI_InitSliders(WSLD_KEYSTONEWINSLIDER, (int16)temp, -20, 20,
DTXT_KEYSTONEVALUE);
                     /* write the new value to the EEPROM */
                    EE_SetData(&(pEEPROM->UserMachine.Keystone), TRUE, 1,
(void*)&temp);
                }
            #endif
            }
            break;

.          case ITEM_TESTITEM: // This case ITEM_KEYSTONEITEM for the operating
“Adjust item right” on TEST item

           temp = Test;
            #ifdef SDE
                if (temp < 100)
                {
                    temp++;
                    GUI_InitSliders(WSLD_TESTWINSLIDER, (int16)(temp), -100, 100,
DTXT_TESTVALUE);
                }
            #else
                /* don't do anything if already at top end of adjustment range */
                if (temp == 100)
                    break;
                else
                    temp++;
                if (REON_KeystoneSettings(ReonAddress, temp) == PASS)
                 {
                                GUI_InitSliders(WSLD_TESTWINSLIDER, (int16)temp,
-100, 100,   DTXT_TESTVALUE);
                 Test = temp;
                    /* write the new value to the EEPROM */
                                }
            #endif
            break;
```

Listing 3: Function GUI_DisplayMenuAdjustItemRight

Function GUI_DisplayMenuAdjustItemRight will get the itemID first, and then decide what kind of operating will do next. For example, if it is case ITEM_KEYSTONEITEM, system will read the value for the Keystone from the EEPROM by calling the function EE_GetData, and store it in “temp”. If it’s successful, then check this value. If it is equal to 20, break, otherwise add 1. This can control the number no bigger than 20. Now a new temp with the correct value of the Keystone will be used for the function. SRC_LoadKeystoneSettings which is a Library function used for the calling the hardware to do the right operating for the new value of Keystone. Since the hardware do the right adjustment for the new Keystone value (get the “pass” from the function SRC_LoadKeystoneSetting), the system will call the function GUI_InitSliders to change the slider and store the new value of the Keystone in EEPROM by calling function EE_SetData.

```
void GUI_DisplayMenuAdjustItemLeft(void) //Here, “Left” it’s mean decrease the value.
{
    uint16 normal_val, slide_val;
    uint08 list_size, list_val;
    int08 temp = 0;
    int16 contrast;
. case ITEM_KEYSTONEITEM: //If the “Decrease case happen in Keystone slider ”
            if (EE_GetData(&(pEEPROM->UserMachine.Keystone), TRUE, 1,
(void*)&temp) == PASS)
            {
            #ifdef SDE
                if (temp > -20) //ensure the value will not less than -20
                {
                    temp--;
                    GUI_InitSliders(WSLD_KEYSTONEWINSLIDER, (int16)temp, -20, 20,
DTXT_KEYSTONEVALUE);
                    Keystone = temp;
                }
            #else
                if (temp == -20)
                    break;
                else
                    temp--; //If value is no less than or equal to -20, the value decrease by
one.

                if (SRC_LoadKeystoneSettings(TRUE, temp) == PASS) //Call the hardware
working
                {
                    GUI_InitSliders(WSLD_KEYSTONEWINSLIDER, (int16)temp, -20, 20,
```

```
DTXT_KEYSTONEVALUE); //Change the OSD slider

                        /* write the new value to the EEPROM */
                        EE_SetData(&(pEEPROM->UserMachine.Keystone), TRUE, 1,
(void*)&temp); //Store the value in the EEPROM
                    }
                }
                #endif
                break;
            case ITEM_TESTITEM: // This case ITEM_KEYSTONEITEM for the operating
“Adjust item left” on TEST item

                 temp = Test;
                //if (EE_GetData(&(pEEPROM->UserMachine.TEST), TRUE, 1, (void*)&temp)
== PASS)
                //{
                #ifdef SDE
                    if (temp > -100)
                    {
                        temp--;
                        GUI_InitSliders(WSLD_TESTWINSLIDER, (int16)temp, -100, 100,
DTXT_TESTVALUE);
                        Test = temp;
                    }
                #else
                    if (temp == -100)
                        break;
                    else
                        temp--;

                    if (REON_KeystoneSettings(ReonAddress, temp) == PASS)
                    {
                         //temp = 1;
                        GUI_InitSliders(WSLD_TESTWINSLIDER, (int16)temp, -100, 100,
DTXT_TESTVALUE);

                        /* write the new value to the EEPROM */
                        //EE_SetData(&(pEEPROM->UserMachine.TEST), TRUE, 1,
(void*)&temp);
                        Test = temp;
                    }
                #endif
```

```
//}
break;.
```

Function GUI_DisplayMenuAdjustItemLeft will get the itemID first, and decide what kind of operating should to do next. For example, if it is case ITEM_KEYSTONEITEM, system will read the value for the Keystone from the EEPROM by calling the function EE_GetData, and store it in “temp”. If it’s successful, check this value, and if it is equal to -20, break, otherwise subtract 1. This operation can control the number no less than -20. Now a new temp with the correct value of the Keystone will be used for the function. SRC_LoadKeystoneSettings which is a Library function using for the calling the hardware to do the right operating for the new value of Keystone. Since the hardware do the right adjustment for the new Keystone value (get the “pass” from the function SRC_LoadKeystoneSetting), the system will call the function GUI_InitSliders to change the slider and store the new value of the Keystone in EEPROM by calling function EE_SetData.

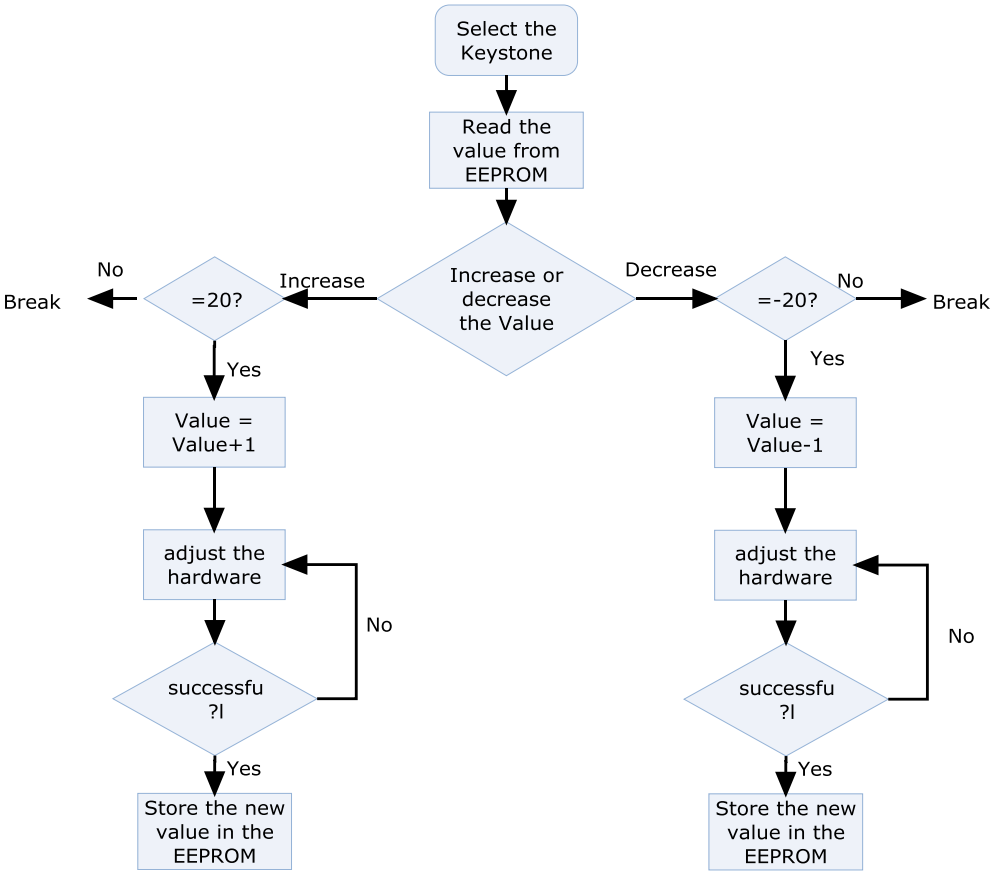


Figure 35: Keystone slider adjustment flow diagram

For the new item “TEST”, the procedure is quite similar. But, there is no address in EEPROM for the item “TEST”, and I always store the value in a

integer “Test”. Here we calling a function REON_KeystoneSettings which will be used for control the I^{2}C slave device REON VX, and we will discuss this function later.

3.2.3 I^{2}C Bus

The slave device REON VX will be connected to master device DDP2000 via I^{2}C bus. From the information of the meeting between OPTEA AB and SOI (Silicon Optix) 2008.05.07, I know DDP2000 can control the REON VX by sending the data (include the operating number and operating value) to the REON VX’s by I^{2}C address. DDP2000 API provides the function I2C_MasterWrite for the DDP2000 send the data to the I2C bus slave device.

```
int08 REON_KeystoneSettings(uint08 ReonAddress, uint08 OperationValu) //This is a new function for the I2C data sending which had been used in function GUI_DisplayMenuAdjustItemLeft and GUI_DisplayMenuAdjustItemRight

{
    int08   RetVal;
    uint08 WorkUint08[2];
    uint32 numWritten;
    static const uint08 sem_fail[] = " I2C reserve failed !!\n";

    /* configure the I2C controller */
    TVP_VerifyBusControlSetup();

    // Reserve I2C Semaphore
    if      (RTA_SemReserve(ID_I2C_Sem,      TMR_ConvertMSToTicks(1000))      !=
RTA_SUCCESS) {
        trace(sem_fail);
        return FAIL;
    }

    WorkUint08[0] = 0x02;
    WorkUint08[1] = OperationValu;

    RetVal = I2C_MasterWrite((uint16) ReonAddress,
                        sizeof(WorkUint08),     /* num write bytes */
                        WorkUint08,
                        0,                  /* byte delay */
```

```
                        1000,           /* timeout */
                        &numWritten);

    // Release I2C Semaphore
    RTA_SemRelease(ID_I2C_Sem);

    return RetVal;

}
```

Listing 4: Function REON_KeystoneSettings

Function REON_KeystoneSettings call the function TVP_VerifyBusControlSetup() to setup the bus controller, and then require the semaphore. Function RTA_SemReserve will return a value to shows this reserve is done or not. If it is successful, function RTA_SemReserve send the operating number (WorkUint08[0]) and operating value (WorkUint08[1]) to the REON's address.

In order to make this function good working, we should "**# include**" some **.h** files

```
//This the new .h file we adding the beginning of "guimenu_display.c"
#include "TVP5146.h"
#include "common.h"
#include "assert.h"
#include "app_cfg.h"
#include "stdio.h"
#include "string.h"
```

Listing 5: Adding the .h files

3.3 Co-Work

When the code modification was finished, we need to put the new code work together.

Start the ADS v1.2, and open the .mcp file from the folder with the reference design code.

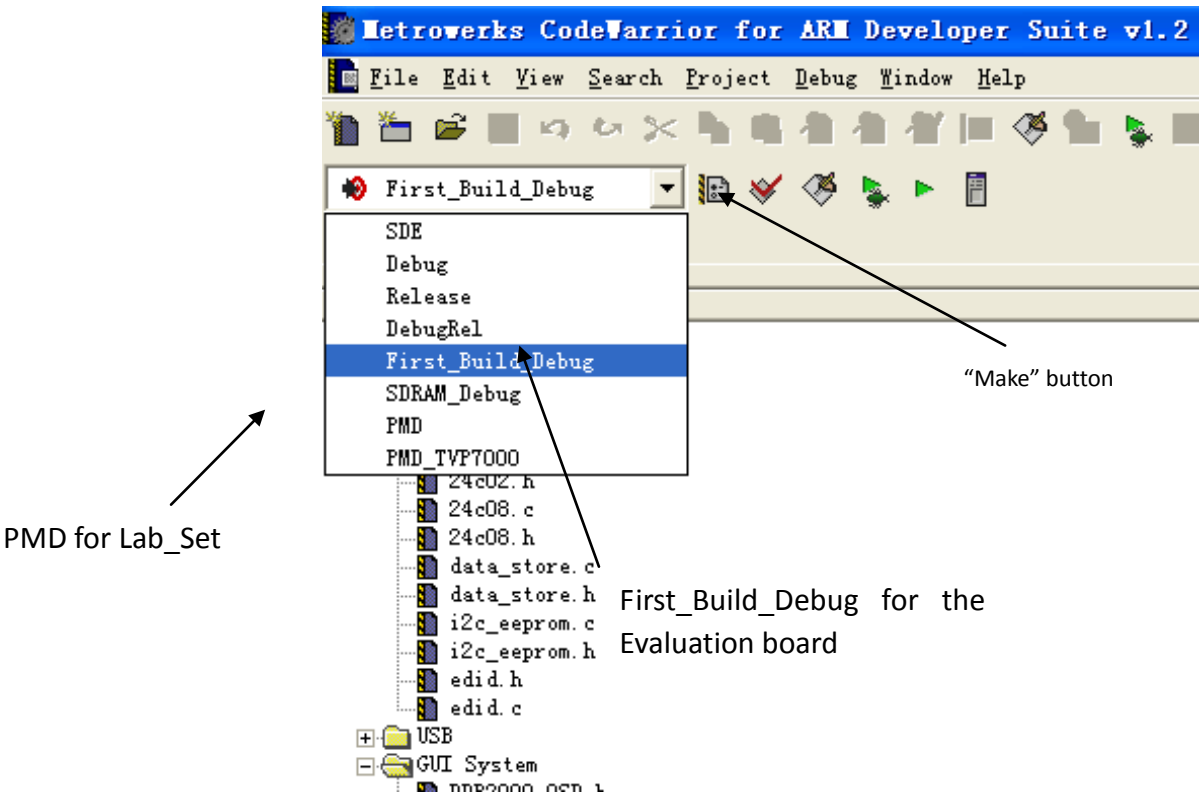


Figure 36: Compile procedure

Since we have 2 test boards, Evaluation Board and Lab_Set, and we be careful to choose PMD for the Lab_Set, and First_Build_Debug for the Evaluation Board. Then press the button "Make" to compile, if there is no problem with the code, a new file name DDP200_ First_Build_Debug.bin or a new file name DDP2000_PMD_AD9882.bin will be generated after the First_Build_Debug make or the PMD make.

Back to the DLP Composer, Click the "flash builder" at left window, and you will see all files to build final image file at the right window.

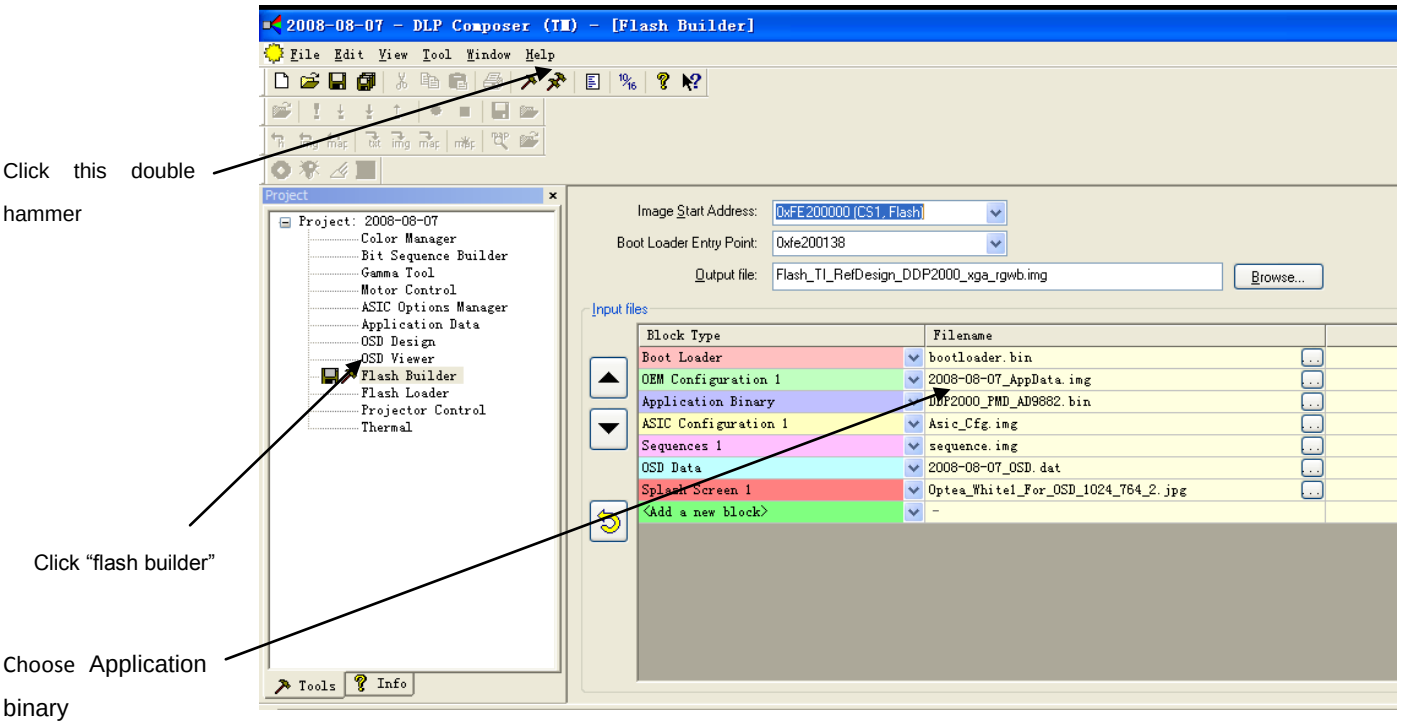


Figure 37: check all files to build final image file

Next step is choosing Application binary file generated by code warrior of ADS.1.2, PMD for the Lab_Set, and First_Build_Debug for the Evaluation Board. I keep the other file default setting. After click this double hammer to build all the files, an image file called flash_TI_Refdesign_ddp2000_XGA_RGWB.img will be generated.

4 TESTING & RESULTS

4.1 Test Strategy

A well-planned test strategy is important to detect error as soon as it occurs. This could save a lot of time, effort and hence cost.

4.1.1 Tools and Resources

Apart from the tools and resources described in Chapter 3, there are few others tools which are useful for testing purposes.

- **Terminal v1.9b**

  Terminal v1.9b will show the useful information when the system running from the DDP2000 by R232 port.

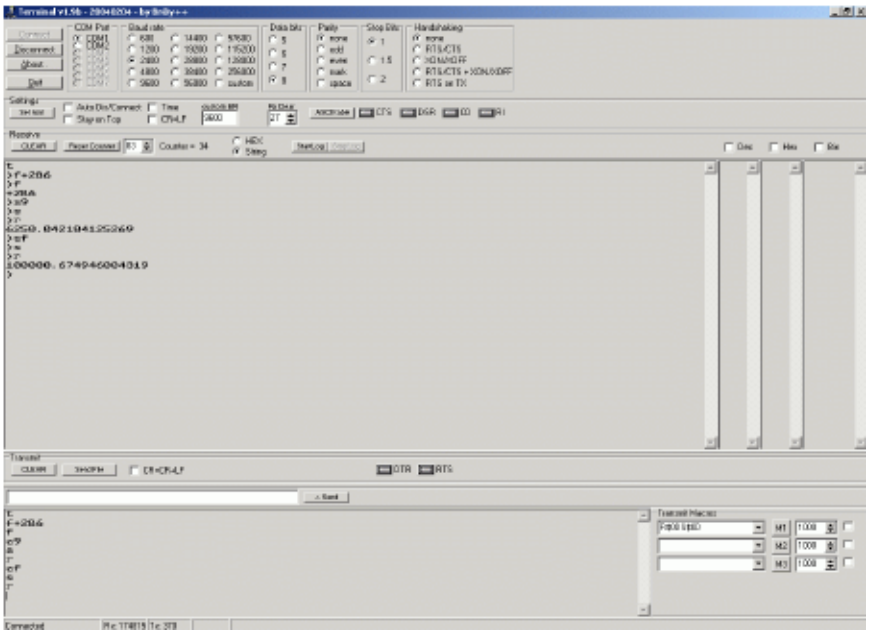


Figure 38: Terminal V1.9b Window

- Evaluation Board

  There are errors that might only apparent when both the software and hardware are integrated together, even the code and the OSD design have no problem to work together. Therefore it is useful to test the New

OSD system on the Evaluation Board. The Evaluation Board is also important to ensure the OSD menu of the projector of the function correctly.



Figure 39: Evaluation Board with DMD

- Beagle I2C/SPI/MDIO Protocol Analyzer

The versatile Beagle™ I2C/SPI/MDIO Protocol Analyzer will Non-intrusive capture I2C, SPI, and MDIO data, such data will show sending condition of the command data from the DDP2000 system to a certain I^2C address.



Figure 40: Beagle I2C/SPI/MDIO Protocol Analyze

Beagle Data Center Software will capture the data between DDP2000 and REON chip. In the testing, the Beagle™ I2C/SPI/MDIO Protocol Analyzer will help designer ensure the right data had been send to the right address.

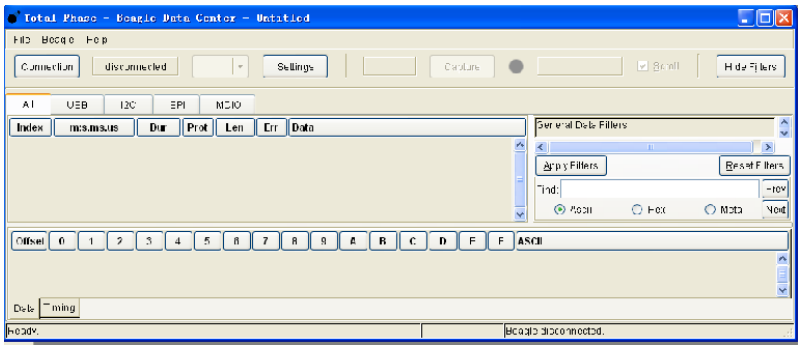


Figure 41: Beagle Data Center Software

4.1.2 Testing & Result

The following table shows the checklist for testing the New OSD. The intention is not to present a detailed testing but to test the most crucial parts of the application.

Test Number	Description	Operation	Result
1	**Testing Evaluation Board start**	**Start the Evaluation Board**	
1.1	**Check the Evaluation Board cans normal start with the new OSD system: The modification of the OSD system will not break the whole DDP2000 system, and the Evaluation Board should start.**		**Evaluation Board successful start.**
2	**OSD Testing**		
2.1	**Check there is high-light effect for the new item: the function provides the high-light effect will use the item ID from the system, the new item should be recognized by the system and the system will get its ID. So there should have high-light effect on the new item.**	**Move the high-light pointer to several items**	**The new item of the OSD has the high-light effect**
2.2	**Check the number box of the**	**Right move and**	**Number box**

	new item will correct change when the value is changed: The increase and decrease of the value will be shown by calling the correct function. The number box of the item will do the correct change.	**left move the number**	**changes correctly**
2.3	**Check the slider will correct change correspond to the value's change**	**Right move and left move the number**	**Slider correct change.**
3	**The code algorithm testing**		
3.1	**Check the value will correct change when there is increase operating or decrease operating: The value will correct change.**	**Right move and left move the number**	**Value correct change correspond to the increase and decrease operating**
3.2	**Check the value will not increase when it is 100**	**Increase the value when it reach 100**	**We cannot increase the value when it reaches 100.**
3.3	**Check the value will not decrease when it is -100**	**Decrease the value when it reach -100**	**We cannot decrease the value when it reaches -100.**
3.4	**Check the new will be stored after the OSD shut down**	**Adjust value in 5, shut down the OSD and start again**	**the value is 5 when start again**
4	**I^2C Testing**		

4.1	**Check the data send to the correct address**	**Increase and Decrease the value of “TEST”**	**Yes, the data had been send to the I^2C address 0X98**
4.2	**Check the data from the DDP2000 is the right data**	**Increase and Decrease the value of “TEST”**	**Yes, the data is the right data from the DDP2000**

Listing 6: Testing & Result

5 CONCLUSIONS & FUTURE WORK

The Embedded software development for the VEO+ shows one way to modify the OSD. By using the development tool DLP Composer, we are able to create the new item for the new function. For this project, I have done a careful study of the whole system and the development tool. The results are as follows:

- Add one item into the OSD with the slider property;
- Write new code to implement the functions of the new item;
- Validate the whole design, ensuring that the new OSD functions correctly on the evaluation board;

Based on the initial results, future work can be done in several aspects. Firstly, in this project, we only use one function from the REON chip, but if we want to use more functions, add a new “menu” is necessary. Secondly, the new function we created for the data sending to the REON chip is not only used for REON chip, but also for the other I^2C slave devices by only changing the I^2C address and the data format. The data format research will make the implementation of other I^2C slave devices (like ESS) possible.

REFERENCE

Books:

[1] Brian W. Kernighan and Dennis M. Ritchie. 1988. The C Programming Language, Second Edition. Prentice Hall. USA

[2] Chen Yu, Li Ming, Yang Ye. 2004. Analysis and Practice on Open Source Embedded Software -- Based on SkyEye and ARM Platform. TSINGHUA UNIVERSITY PRESS. China

[3] David E. Simon*. 1999.* An Embedded Software Primer. Pearson. UK

[4] Susan Weinschenk, Pamela Jamar, Sarah C. Yeo. 1997. GUI Design Essentials. John Wiley & Sons, Chichester, England

[5] Paret, Dominique; Fenger, Carl. 1997. The I2c Bus From Theory to Practice. John Wiley & Son. Chichester, England

Data Sheets, User Guides and Specification Documents

[6] DDP2000 User's Guide

[7] Reon Product Brief
http://www.siliconoptix.com/products/ReonVX/Documentation.cfm?CFID=16977793&CFTOKEN=15711395

[8]DLP Composer Tool Suite Help

[9]C Language Tutorial
http://www.physics.drexel.edu/students/courses/Comp_Phys/General/C_basics/

Programming Resources and Other Useful Links

[10] The official site of the embedded development community
http://www.embedded.com/tigforums/forum.jspa?forumID=23

[11] The Open GTOUP's Real-Time & Embedded Systems Forum
http://www.opengroup.org/rtforum/

[12] The programmer's discussion area

http://forums.devshed.com/c-programming-42/