Designing the first application – Hello World

Preface

The easiest way to get familiar with the VT-OS is through a simple example. This tutorial will describe how to build a "Hello World" application and show how to debug it on board.

Introduction

The tutorial will first introduce the compile tools, including environment settings, compiler & linker, resource compiler and application builder. Then we will start the design of the application, and finally discuss the coding of the application.

Requirements

1. GCC compiler for MIPS R3000A
2. Resource compiler
3. Application Builder
4. Windows 95/98 or DOS with DPMI memory support
5. Helio and cradle

The compilers can be found in the VT-OS SDK and you can get the application builder on the web easily. Please note that if the optimization option of the C compiler is enable (-O), more memory is required for compiling complex code. You may need to set a larger DMPI memory in this case. Please reference to the environment section for detail description.

Environment settings

In the directory \SDK\COMPILER\GNU you will find a batch file – SETENV.BAT. The content is :

SETENV.BAT

```
REM mips-idt-ecoff
SET PATH=d:\GNU\BIN;d:\BIN;%PATH%
SET GCC_EXEC_PREFIX=d:\GNU\LIB\
SET INFOPATH=d:\GNU\INFO
SET C_INCLUDE_PATH=d:\GNU\include
SET CPLUS_INCLUDE_PATH=d:\GNU\include\cxx;d:\GNU\include
SET GO32=DPMISTACK 2048000
REM Set TMPDIR to point to a ramdisk if you have one
SET TMPDIR=C:\TMP
```

Please remember to set the correct path for the GNU tools. In order to support optimization, you may need to set the DPMISTACK to a larger value, the size 2048000 is suitable for all VT-OS components. When operate under Windows DOS prompt, set the "MS-DOS protected-mode (DPMI) memory" to 65535 in the properties as well. If you have spare RAM, setting the TMPDIR to a ramdrive will speedup the compilation time.

You may also want to set the output redirection for long warning/error messages.

```
SET GO32=DPMISTACK 2048000 1R2
        Redirect stdout to stderr.
SET GO32=DPMISTACK 2048000 2R1
        Redirect stderr to stdout
```

The message output will put to the listing file if the list file option (-Wa,-a>$@~) of the GCC compiler is enable. Please refer to the MAKEFILE section for more information.

Compiler & Linker

The VT-OS and the application are compiled by the GCC compiler. Before you can start the compilation, you have to prepare two files. They are LINK.CMD and MAKEFILE. Here list the content of the two files for "Hello World" application.

LINK.CMD

```
OUTPUT_FORMAT("ecoff-littlemips")
ENTRY(appstart)

MEMORY
{
        FLASH           : org = 0x9FC30000, l = 1M
        RAM             : org = 0x10000000, l = 1M
}

SECTIONS
{
        .head :
        {
                _fhead = . ;
                        obj/appstart.o(.data)
                ehead = . ;
        } > RAM
        .text :
        {
        _ftext = . ;
                obj/appstart.o(.text)
                *(.text)
                obj/resdata.o(.data)
        etext = . ;
```

```
        } > RAM
        .data ALIGN(0x1000)     :
/*      .data :*/
        {
                _fdata = . ;
                        *(.data)
                        *(.rdata)
                edata = . ;
        } > RAM
        .bss      :
        {
                _fbss = . ;
                        *(.bss)
                        *(COMMON)
                        *(.common)
                        *(.scommon)
                ebss = . ;
        } > RAM
}
```

The LINK.CMD provides address information to the linker. The MEMORY{} section define the accessible address and the size for that address. The mapping of the address can be found in the file \SDK\OS MANUAL\OS OVERVIEW.PDF. Generally, all download applications, will use the RAM region for both code and data.

The SECTIONS section defines the address of each section of the program. For a download application, it contains a header section, which describes the location of the code and data region, and this information is stored in the file APPSTART.S.

Follow the header it is the code section. Please note that you need to place the file APPSTART.O before the others, the default starting address is at 0x10000020, which should be the code in APPSTART.S that jump to __main. Normally, you do not need to modify the APPSTART.S if you program starts at __main().

The data region is 4KB align and the uninitialised data region is placed after the data region. The OS will allocate RAM according to the .data and .bss size when the program is executed.

MAKEFILE

```
TOOLS=d:/gnu/bin

ENDIAN=EL
OBJ=obj

HWOBJ = $(OBJ)/appstart.o $(OBJ)/aplaunch.o $(OBJ)/app.o $(OBJ)/resdata.o $(OBJ)/syscall.o
$(OBJ)/main.o

PROCESSOR = POSEIDON
```

```
# Assembler option
AS= ${TOOLS}/gcc -Wa,-a>$@~ -O2 -G0 -c -g -$(ENDIAN) -DPR31700

# Compiler option
CC= ${TOOLS}/gcc -Wa,-a>$@~ -O2 -G0 -c -g -s -$(ENDIAN) -msoft-float -mno-abicalls -mips2 -
DPR31700

# Linker option
LD= ${TOOLS}/ld -n -warn-common -Map link.map -$(ENDIAN)

$(OBJ)/%.o: %.S
        echo $(AS) $<
        $(AS) $< -o $@

$(OBJ)/%.o: %.c
        echo $(CC) $<
        $(CC) $< -o $@


main.obj: $(HWOBJ)
        $(LD)  -nostdlib -G 0 -T link.CMD -o $@ $(HWOBJ)

#       echo size
        $(TOOLS)/size -x $@
#       echo ------------------ objcopy ------------------
#      $(TOOLS)/objcopy -O srec $@ a.

#       echo ------- Please enter "BUILD" to build the final image -------
```

The MAKEFILE contains the information for the compiler. We use simple syntax to enable the compilation. The details of the compiler and the assembler option are described in the directory \SDK\COMPILER\GNU\INFO\.

The Helio runs in little endian mode –EL. We add the –MIPS2 option to enable the branch likely instruction. After the compilation, a MAIN.OBJ is generated and the listing file is placed in directory .\OBJ as *.O~ if the option -Wa,-a>$@~ is enable.

The object code is then chop by the program CHOP.EXE to form the binary image as MAIN.OUT. You can use the batch file BUILD.BAT to complete the job automatically. The main.out is the file required for building the downloadable image by the application builder.

BUILD.BAT

```
del main.out
make
chop main.obj main.out .head .text .data
```

Please note that the CHOP.EXE arguments contain the name as defined in the LINK.CMD. Only the sections listed in the arguments will be copy to the image.

The .bss region does not include in the image since it will be created by the OS in run time.

Resource Compiler

Resource compiler is a tool helping developers to generate a resource file that can be read by the system.

After the completion of the plain-text resource file, then resource compiling can be proceeded. The steps are as below:

1. Execute the rcompile.exe as rcompile <name of resource file>
2. A resource.bin file is generated out
3. Execute the bin2hex.exe as bin2hex <resource.bin> <name of output C file>
4. The output C file is the compiled C-format resource file that the system can be read. The file should be included in the make file

Please note that the resource file should not constains any trailing space. The BMP file referenced in the resource file is in WINDOWS format with 16 grey level and no compression.

Application Builder

Application Builder is a tool helping developer to generate the required format of application or device driver to download to PDA. There are a number of options in the builder for the developers to set. They are Application Name, Application Run Speed, Application Type, Add-on System Calls and Add-on Hook Table. More details of the settings can be found in Quick Guide to AppBuilder.doc.

After all information is provided, the MAIN.OUT of the application or device driver can be converted to a .APP file for downloading to PDA.

Application Design

The application design of an application for the Helio PDA starts by designing the screen layout of the applications and by preparing the resource file.

A resource file is a plain-text file. It includes the descriptions of all the UI objects that are used in the application. In the tutorial example, a “Hello World” application consists of one screen layout with a label in the middle of the screen. The resource file consists of two UI objects: form and string. Here is the resource file of “Hello World”.

HWRES.TXT

```
//---------------------------------------------------------------- FORM_HELLO ----------------------------------------
--------------------------
#FORM
NAME=FORM_HELLO
ID=0
~F0
(B)OBJECT.TYPE=FORM
~F1
(SH)BOUNDS.X=0
(SH)BOUNDS.Y=0
(SH)BOUNDS.WIDTH=160
(SH)BOUNDS.HEIGHT=160
~F2
(U)FOCUSED_OBJECT=1
~F3
 (B)FORM_STYLE=NORMAL
~F4
 (S)FORM_TITLE=Hello World
~F5
 (SH)FORM_BITMAP.X=0
(SH)FORM_BITMAP.Y=0
 (SH)FORM_BITMAP.WIDTH=10
(SH)FORM_BITMAP.HEIGHT=12
 (P)FORM_BITMAP.FILE=Q_TWO_BIT,helloi.BMP
~F6
(U)NO_OF_OBJECTS=1
~F7
 (U)OBJECT_ID=1
 (B)OBJECT_TYPE=STRING
#END_FORM
//---------------------------------------------------------------- STRING_HELLO ----------------------------------------
---------------------------
#STRING
NAME=STRING_HELLO
ID=1
~F0
(B)OBJECT.TYPE=STRING
~F1
 (U)RELATED_TABLE_ID=65535
~F2
 (SH)BOUNDS.X=20
(SH)BOUNDS.Y=50
 (SH)BOUNDS.WIDTH=50
(SH)BOUNDS.HEIGHT=9
~F3
 (B)STRING_STYLE=STRING_O
~F4
 (S)STRING_TEXT=Hello World!
~F5
 (B)STRING_FONT_COLOR=COLOR_BLACK
~F6
 (B)STRING_BACKGROUND_COLOR=COLOR_WHITE
~F7
 (B)STRING_FONT=NORMAL_FONT
```

```
~F8
 (B)STRING_TEXTMENT=CENTRE
~F9
 (BO)STRING_ATTR.STRING_VISIBLE=TRUE
#END_STRING
```

There is a standard format of the resource files and are a number of rules on how to construct a resource file. They are shown as follows:

- #XXX indicates the following resource details describe a certain UI object while
- #END_XXX indicates the end of the object description where XXX represents the name of a UI object.
- within the resource details of a UI object, it divides into certain fields. Each field starts with an indicator, ~Fx, where x is the field number.
- the bracket letters represent different data type. For example,
  S = string
  B = BYTE
  BO = BOOLEAN
- each UI object inside the resource file has an object ID that acts as an identity within an application. So the object ID cannot be used repeatedly.
- spacing is not allowed within the resource file.
- different UI objects describe their resource details in a different format. The total numbers of fields are also different. See the other document in the SDK Resource File Format.pdf for more information.
- at the end of the resource file, remember to press enter if you use window's editor; otherwise the resource file cannot be compiled.

After the resource file is completed, resource compile (rcompile.exe) and binary-to-hexadecimal utility are used to convert the resource file to a file format that can be read the Helio VT-OS system.

Application Programming

VT - OS is a single-task and event driven operation system. Only one application can run at a time. Each application is composed of several forms, which contain certain UI objects to display. Before you start to write an application, you should realize the operation of Helio system and then you can structure your own application.

- Each application has a Main function which is inside main.c file. System sends a **launch code** to start an application. Note that the purpose of Main function is to receive launch codes and call AppLaunch function to handle them. This function exists in ApLaunch.c file.

- Since VT - OS is event driven, an event loop exists in main.c file to get events and then pass to event handler. Each application contains several forms. Different forms should have their own event handler function in App.c.
- VT - OS application is stopped when it receives EVT_APP_STOP event.

Therefore, as a developer, the AppLaunch function and different event handler functions of the forms in App.c must be required. For more information, there is a document called App Design.pdf in our SDK.

During the programming, debugging is another important issue for a good application or device driver. In the Helio developing environment, the only on board debugging tool is by calling printf() to display the required data or memory content to the terminal.

For optimized performance of the product, all the port from PDA to terminal is disabled. In order to print out the useful data to terminal, SysEnableDebug() must be called once before any calling to printf() function.

Please remember to free the COM port before launch the terminal to observe the print out data. It is not uncommon that the V-Sync manager is occupying the COM port after you download the application for debug.

Application Download

In order to download application or device driver to the Helio PDA, the following steps are required.

1. The application source code should be built by using AppBuilder. Then, a .APP file is generated out
2. Lauch Helio Desktop.
3. Select V-Sync from the menu and choose "Install Application to Helio"
4. Add the application to the list by pressing the [ADD] button. (You should browse to the directory that contains the application.)
5. Perform a V-Sync.

Then you will see the application icon appears in the main menu. You can delete the application in the System Setup->System Information->Delete Apps.

IMPORTANT: Remember to backup the data in Helio by V-Sync before pilot run your application. It is always possible for an application or device driver runs so far away.