Pluto

Pluto 5 Enhanced Development Kit - Quick Start

Document No. 80-16054 Issue 10 HEBER LTD

Current Issue:	Issue 10 – 24th December 2004
Previous Issues: -	Issue 1 - 23rdFebruary 2000
	Issue 2 - 8th May 2000
	Issue 3 - 9th May 2000
	Issue 4 - 17th July 2001
	Issue 5 – 26th June 2002
	Issue 6 – 11th July 2002
	Issue 7 – 26th July 2002
	Issue 8 – 16th September 2002
	Issue 9 – 12th January 2004

HEBER LTD
Belvedere Mill
Chalford
Stroud
Gloucestershire
GL6 8NT
England

Tel: +44 (0) 1453 886000
Fax: +44 (0) 1453 885013
Email: support@heber.co.uk
http://www.heber.co.uk

CONTENTS

1 OVERVIEW **1**

2 PROJECT FILES **1**

2.1 THE SOURCE FILE TYPES 1
2.1.1 'C' file *1*
2.1.2 Header file *1*
2.1.3 Prototypes file *1*
2.1.4 Static Prototypes file *1*
2.2 THE SOURCE FILES 1
2.2.1 Bell1 ..... *1*
2.2.2 Xnudrep2 ..... *1*
2.2.3 Config *1*
2.2.4 Demo - Vdemo *1*
2.2.5 Devices *1*
2.2.6 Fpga *2*
2.2.7 Game *2*
2.2.8 Others *2*
2.3 THE MAKE FILE 2
2.3.1 Make File Variables *2*
2.3.2 Implicit Rules *3*
2.3.3 The Default Goal *4*
2.3.4 Dependencies *4*
2.4 THE LINKER SCRIPT FILE 4

3 SOFTWARE-HARDWARE INTERACTION **5**

3.1 PLUTO 5 & PLUTO 5 CASINO DIFFERENCES 5
3.2 EPROM-RAM DIFFERENCES 5

4 VARIOUS METHODS TO RUN THE SOFTWARE **6**

4.1 RUNNING A PROGRAM FROM THE EPROM PROVIDED WITH DEVELOPMENT KIT 6
4.2 RUNNING A DEMONSTRATION PROGRAM FROM THE RAM/EPROM EXPANSION CARD 6
4.2.1 Modifying the Demonstration Program *6*
4.2.2 Building the Demonstration Program *7*
4.2.3 Downloading the Software to the Pluto 5 Board *7*
4.3 PUTTING THE SOFTWARE ONTO EPROM 7

5 UTILITIES **8**

5.1 MANAGING PROJECTS FOR PLUTO 5 BOARDS 8
5.1.1 DEP *8*
5.1.2 LOC *8*
5.1.3 MAKEMAP *8*
5.1.4 PRO *8*
5.1.5 SETLIB *8*
5.1.6 TOUCH *8*
5.2 CONTROLLING BINARY FILES: SCRAMBLE - SCRAMCAL 9
5.3 UNDERSTANDING VARIOUS TOOLS 9
5.3.1 Images: *9*
5.3.2 Fonts *13*
5.3.3 Sound *14*

6 INTRODUCTION TO VIDEO PROGRAMMING: **15**

6.1.1 Layer Architecture Video controler *15*
6.1.2 Manipulating and accessing layers *15*
6.1.3 Functional overview: *16*

7 RUNNING THE DEMONSTRATION PROJECT **18**

7.1 LAMPS 18
7.1.1 Strobe 18
7.1.2 Row 18
7.1.3 Caterpillar 18
7.1.4 Suit 19
7.1.5 On 19
7.1.6 Flash 19
7.1.7 Fastflash 19
7.1.8 Dim 19
7.1.9 Exit 19
7.2 DIGITS 19
7.2.1 Digit 19
7.2.2 Segment 20
7.2.3 Exit 20
7.3 VFD 20
7.3.1 Abc 20
7.3.2 Inputstring 20
7.3.3 Exit 20
7.4 REELS 20
7.4.1 Init 20
7.4.2 Step 20
7.4.3 Spin 20
7.4.4 Skill 20
7.4.5 Alternate 21
7.4.6 Status 21
7.4.7 Exit 21
7.5 INPUTS 21
7.6 SOUND 21
7.6.1 Play1 21
7.6.2 Incvol1 21
7.6.3 Decvol1 21
7.6.4 Play2 21
7.6.5 Incvol2 21
7.6.6 Decvol2 21
7.6.7 Exit 21
7.7 MXI 22
7.8 OUTPUTS 22

8 RUNNING THE CALYPSO16 PROJECT 23

8.1 EXPLANATION OF THE CODE: 23
8.2 DETECTING A CD-ROM 23

9 RUNNING THE COMPACT FLASH EXAMPLE PROJECT 24

9.1 INTRODUCTION 24
9.2 INSTALLING THE COMPACT FLASH BOARD. 24
9.2.1 PCB Configuration 24
9.2.2 Data 24
9.2.3 Power 24
9.2.4 Compact Flash card 24
9.2.5 VGA Monitor 24
9.2.6 Serial Connection 24
9.3 RUNNING THE PROJECT 25
9.3.1 Loading the Project File to RAM 25
9.3.2 Listing and changing directories. 26
9.3.3 Running the examples 26
9.3.4 Loading Text Files. 26

10 RUNNING THE CALYPSO ENHANCED PROJECT 26

10.1 SETUP 26
10.1.1 PCB Configuration 26
10.1.2 VGA Monitor 27
10.1.3 Building the project 27
10.2 RUNNING THE PROJECT 27

10.2.1 Loading the Project File to RAM ... *27*

11 RUNNING THE TOUCHSCREEN EXAMPLE PROJECT ... 27

11.1 SETUP ... 27

11.1.1 PCB Configuration ... *27*

11.1.2 VGA Monitor ... *27*

11.1.3 Serial Connection ... *28*

11.2 RUNNING THE PROJECT ... 28

11.2.1 Loading the Project file to RAM ... *28*

11.2.2 Calibrate and run ... *28*

12 CONNECTING DEVICES TO THE PLUTO5 BOARD ... 28

TABLE OF REFERENCE

REFERENCE 1: Instructions to load A picture 29

1 OVERVIEW

This document is a guide to getting started with the Pluto 5 Development Kit and it assumes that you have followed the instruction leaflet 80-16091, Setting up the Pluto 5 Development Kit. All the supplied software should be installed on your PC and the Pluto 5 Casino board should be connected to your PC via the P&E ICD cable and the serial cable, either via the Pluto 5/5C Evaluation board or stand alone.

2 PROJECT FILES

2.1 The Source File Types

There are three source modules and two assembly level sound modules. The assembly level modules consist of the assembly source file (.S) and the corresponding dependency files (.D). The three 'C' source modules consist of the actual 'C' file (.C), the header file (.H) the dependency file (.D), the prototypes file (.P), and the static prototypes file (.SP).

2.1.1 'C' file

Contains the 'C' code

2.1.2 Header file

Contains #defines and type definition used in the 'C' file. The header file in turn #includes the prototypes file. This means that including the .h file of a given module gives the user access to all functions and variables declared public.

2.1.3 Prototypes file

This file contains the declaration of all the external functions and external variables declared public in the 'C' files which can be used in other files. This file is automatically updated when running the makefile as long as it initially exists.

2.1.4 Static Prototypes file

This file contains prototypes for all functions declared to be local to the 'C' file. These functions cannot be accessed from any other file. This file should only be #included in the 'C' file of the same name. Its' inclusion means that functions can be used in the 'C' file prior to their declaration. This file is automatically updated when running the makefile as long as it initially exists.

2.2 The Source Files

2.2.1 Bell1

This is an assembly level sound module generated by the sound conversion utility.

2.2.2 Xnudrep2

This is an assembly level sound module generated by the sound conversion utility.

2.2.3 Config

This module contains device configuration data.

2.2.4 Demo - Vdemo

This module contains the demonstration software and associated constants.

2.2.5 Devices

This file contains the device list. After the low-level initialisation of the board, the device manager scans this list and installs the drivers it finds in it.

2.2.6 Fpga

This file controls the differences in addressing and Data acknowledge between the Pluto 5 and the Pluto 5 Casino. Depending on which platform your game is for, ensure that you have the correct version of this file installed in your project folder. The default copy in each project folder is for the Pluto 5 Casino.

2.2.7 Game

This module contains function 'Game()'. After system initialisation this function calls DemoFnc() and gets into an infinite loop until user ends the program.

2.2.8 Others

Heart, spade, club, diamond and smile (all .s files) are assembly image files. Part 3.8 explain how to obtain them and further in this document how to use them to display a small picture on the LED's.

Demo.s19 is the file that you download into the RAM. This file can be use with PEMicro ® software to debug at the assembly level. Demo.cof is another file use for other debuggers.

2.3 The Make File

The make file is named 'makefile' (no file extension) and can be executed from a windows command prompt. After running this program, depending on the type of ***output*** you have selected, a file entitled "file.s19" is created and can either be downloaded into the extension RAM of the development kit, or can be post processed for programming into EPROMs. This file generates binary files for each c file programmed by linking the libraries provided by Heber as well as the prototype and static prototype files include in the directory.

Lines beginning with '#' are comments.

Before downloading the file into the RAM "Makemap" can be ran from the command prompt to produce a map of the memory generated by all the functions and variables (see section 3.10).

For more information regarding the creation of a make file, refer to tutorials on GNU compiler. (www.gnu.org/manual and then select make)

2.3.1 Make File Variables

The first part of the make file defines various variables. A make file variable associates a single label with a string of characters. This label may then be used in several places in the make file and will be replaced by the string when it is encountered by make. The syntax used to define a make file variable is as follows.

path= c:/MyProject/SourceFiles/

To use the value of the label in a command or dependency the following syntax is used.

Gcc $(path)FileName.c

Notice that the GNU compiler Gcc uses UNIX style forward slashes in its path names.

2.3.1.1 Target

This variable is the name of the target platform the files are to be built for. It is also the name of the subdirectory, which contains the library archive for this platform. Script
This variable specifies the name of the linker script file to be used, lnkram.x or lnkrom.x.

2.3.1.2 Output

This variable specifies the name of the final output file that will be produced. (No file extension).

2.3.1.3 Hdw

This variable specifies the path where the header files, object files, and hardware library archive file may be found.

2.3.1.4 Per

This variable specifies the path where the header files, object files, and peripheral library archive file may be found.

2.3.1.5 Lstd

This variable specifies the path where the standard 'C' libraries may be found

2.3.1.6 FLAGS

This variable specifies compiler flags

2.3.1.7 CFLAGS

This variable specifies more compiler flags

2.3.1.8 CNFLAGS

This variable specifies compiler flags that may be used as an alternative to CFLAGS where no optimisation is required. (This can make debugging easier in certain circumstances)

2.3.1.9 IFLAGS

This variable specifies compiler flags associated with include file paths.

2.3.1.10 GNU

This variable specifies the standard 'C' libraries to search for unresolved externals.

2.3.1.11 Lib

This variable specifies the Heber libraries to search for unresolved externals.

2.3.1.12 pp

This variable specifies the command that invokes the utility to generate prototypes for a given 'C' file.

2.3.1.13 as

This variable specifies the command that invokes the GNU assembler.

2.3.1.14 cc

This variable specifies the command that invokes the GNU compiler.

2.3.1.15 ld

This variable specifies the command that invokes the GNU linker.

2.3.1.15.1 ar

This variable specifies the command that invokes the GNU librarian.

2.3.1.16 oc

This variable specifies the command that invokes the GNU object file copier. This converts the coff file format into a Motorola s-record file.

2.3.1.17 Objects

This variable specifies the object files that are to be linked.

2.3.2 Implicit Rules

Implicit rules are a means of specifying to make how to generate a particular class of file from a different class of file. (For example how to generate object files from 'C' files)

2.3.2.1 Rule 1

The first implicit rule specifies how to generate dependency files. A dependency file names, for each object file, every source, header, and prototype file it is dependant on. This information is used by make to calculate which files must be regenerated.

2.3.2.2 Rule2

The second rule specifies how to generate a prototype file (*.p) from a ‘C’ file. The prototype file contains an ‘extern’ for every public function and variable defined in the ‘C’ file.

2.3.2.3 Rule 3

The third rule specifies how to generate a static prototype file (*.sp) from a ‘C’ file. The static prototype file contains a prototype for every local function defined in the ‘C’ file.

2.3.2.4 Rule 4

The fourth rule specifies how to generate an object file from a ‘C’ file.

2.3.2.5 Rule 5

The fifth rule specifies how to generate an object file from an assembly file. For example a sound file generated by the sound conversion utility.

2.3.3 The Default Goal

The default goal is the target of the first non-implicit rule ‘make’ encounters. It is this file that ‘make’ endeavours to keep up to date. In this case the default goal is the executable file defined by the variable $(output).

2.3.4 Dependencies

The make file then includes all the dependency files. This means that make will scan all the dependency files and for each dependency it will make sure that the target is newer than the dependant files.

2.4 The Linker Script File

The linker script file specifies the format of the executable file generated by the linker and the addresses of the various RAM and ROM sections. Linker scripts are provided to link the demo program for execution from ROM, in programmed EPROMs, (lnkrom.x), or for execution from RAM, downloaded to the EPROM/RAM Expansion Board, (lnkram.x).

3 SOFTWARE-HARDWARE INTERACTION

3.1 Pluto 5 & Pluto 5 Casino differences

The Pluto 5 Casino Controller is a new version of the standard Pluto 5 Controller with the following additional functionality:

1. Provision for up to 6 extra serial communication ports, Channel C-H, by fitting DUARTs U53, U54 & U55.
2. Additional 32Kbyte RAM with independent battery backed supply.
3. Provision for a battery powered PIC Microcontroller, which allows power-down monitoring of up to 7 external switches.

To allow the above additions, the following changes have been made to the operation of the controller:

1. A standard Pluto 5 FPGA will not work in the Pluto 5 Casino Controller (the functions of pins 48 & 54 are changed).
2. The Multiplex Expansion (MPX2) facility offered on Pluto 5 is no longer available
3. The General Purpose TTL outputs and external I^2C Bus on connectors P12 & P13 have been remapped.

In each project folder a file called ‘fpga.h’ will have to reflect the board used. This file controls the differences in addressing and Data acknowledge between the Pluto 5 and the Pluto 5 Casino. Depending on which platform your game is for, ensure that you have the correct version of this file installed in your project folder. The default copy in each project folder is for the Pluto 5 Casino.

- For the Pluto 5, copy ***fpga5.h*** to ***fpga.h***
- For the Pluto 5 Casino, copy ***fpga5C.h*** to ***fpga.h***

3.2 EPROM-RAM differences

The Pluto family automatically detects what hardware is plugged onto the board. Pluto 5 Casino Boards access in priority from the top-level hardware, rather than from the bottom-level.

If EPROM’s are fitted in U1 or in U1 & U2 with no EPROM/RAM Expansion card fitted, the board will run software from the EPROMs. When U1 & U2 are fitted and the board detects the EPROM/RAM Expansion card, the board will try to access the EPROM/RAM card.

Important note. If a single EPROM if fitted in U1 of the Pluto 5 Casino Board, you will not be able to access the EPROM/RAM card, if fitted. The Pluto 5 Casino Board requires EPROMs in U1 & U2 in order to enable the Chip Select.

When the Ram is accessed, the processor uses a 16-bit data bus to retrieve data or code from the memory. This data access, controlled by the FPGA, cannot be changed. (WE STRONGLY ADVISED YOU NOT TO TRY TO CHANGE THIS SETTING OR THE BOARD WILL FAIL)

Alternatively, many EPROM configurations are possible for the same program (see eprom_locations.pdf).
For example:
Two half mega octets EPROM (040) fitted on U1 and U2
One 1 mega octets EPROM /801) fitted on U1.

However, in FPGA.h, a register (CSAM0) has been set up to optimise the performances of your board. This register allows the code to run “*fast*” as:

- The data bus is also set up to be a 16 bit data bus
- The delay to access data and code from EPROM is minimal.

```
#define          CSAM0          0x00FFFFF5     //EPROM settings
```

Having this register set to 0x00FFFFF5 **involves** using two EPROM’s fitted in U1 U2. In this configuration, your board is optimised to run in the fastest environment.

Programmers who will consider other constraints and decide that their code has to run with a single EPROM will still be able to do it but this register will have to be modified to:

```
#define        CSAM0        0x00FFFFFD    //EPROM settings
```

Otherwise the program will not pass the initialisation process and the red LED will stay ON.

4 VARIOUS METHODS TO RUN THE SOFTWARE

If you have completed the Pluto 5 Development Kit CD-ROM installation, there are five demonstration programs installed onto your hard drive. "v" being the current version of the development kit, these projects are located in

\heber\projects\demo-**v**,
\heber\projects\calypso16-**v**,
\heber\projects\compact_flash-**v**,
\heber\projects\touchscreen-**v**
\heber\projects\calypso_enhanced-**v**

The demo program communicates with your PC via the serial port and implements a simple menu system to allow the various functions of the board to be accessed.

The video software requires the Calypso 16 Card to be fitted to the Pluto 5 Casino and connected to a suitable VGA monitor. The software will run a limited demo. Some code referring on how to load images from a CD ROM has been commented out and can be uncommented if necessary.

The Compact Flash demo requires the Calypso 16 card to be fitted to the Pluto 5 Casino and connected to a suitable VGA monitor. The software will run a demo, which loads images from the Compact Flash card.

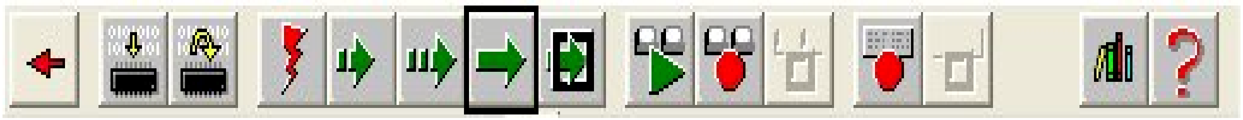
The Touchscreen demo requires either a Micro Touch ® or ELO ® touchscreen to be attached to channel C of the Pluto 5 Casino

The remainder of this section refers to the 'demo' project but the other projects can be compiled and run in the same way.

4.1 Running a program from the EPROM provided with development kit

Ensure the Eprom/Ram card is **<u>not</u>** fitted to the Pluto 5 Casino. The EPROM pairs containing either the Evaluation software or the Demo software should be installed in U1 & U2 of the Pluto 5 Casino board. Apply power to the Pluto 5 board and start the P&E software on your PC.

Start | Programs | P&E 683xx BDM Debugger | ICD32Z In Circuit Debugger. The debugger should initialise and display a disassembly from the reset vector address. Click the 'Go' button (the green arrow with no breaks in it)



The Evaluation software should start running.
Placing the mouse pointer over it can identify a button. A description of the button will appear after a short time.

4.2 Running a Demonstration Program from the RAM/EPROM Expansion Card

4.2.1 Modifying the Demonstration Program

Using your text editor, open demo.c in ***\heber\projects\demo-v*** folder and look at the function 'Flash the Software LED' starting around line 410. This function is installed in the 250ms-interrupt list, and is therefore called four times a second. Every four times this function will be called during a 250ms interrupt subroutine, the status of the green LED will be inverted, so the LED is on for 1 second and off for 1 second. The duration is controlled by the constant variable 'ledtime', which is set to 1.

Try changing this from 1 to 4 or 8 and re-build the program as follows. Notice the change in the led on/off time.

4.2.2 Building the Demonstration Program

To link the program for execution from EPROM / RAM expansion card, edit the makefile from the project directory, and verify that the line which assigns the 'script' variable to read is set to **script = lnkram.x,** run 'touch *.c' and 'make' the project again

Open a command prompt. (To open a command prompt, click « start », point to program, point to accessories and then click « command prompt ».[1])

From this command prompt, in the demo program directory type 'make'. The demo program will be made for execution from the RAM expansion board.

4.2.3 Downloading the Software to the Pluto 5 Board

From this command prompt , in the project directory type then "makemap <name>"
(Where <name> is the name of the s19 file with no extension)

This will generate a map file which will be recognised by the P&E software. This stage is optional. The software will load to RAM and run without this stage but the P&E® software will state that "***..the map file does not exist***".

- Power off the pluto5 casino board.
- Plug in the EPROM / RAM expansion card. You must ensure that the EPROM pairs containing the Demo software (or any other software) is installed in U1 & U2 of the Pluto 5 Casino board or the RAM **will not** be accessed
- Power up the pluto5 board
- Change to the P&E Debugger software.
- Click the reset button.
- Click the 'Go' button.

This time the program execution stops on instruction MOVE.L (1800000).L,D0.

- From the file menu choose Load S19 file and load demo.s19 from ***\heber\projects\demo-6***
- Click the 'Go' button on the toolbar to run the code.

The green LED should now flash on and off at the speed you set above.

4.3 Putting the software onto EPROM

- To link the program for execution from EPROM, edit the makefile and change the line which assigns the 'script' variable to read **script = lnkrom.x,** run 'touch *.c' and 'make' the project again.
- At the command prompt type **scramble3 1 P5_U1 040 demo.s19**. Please notice that scramble3 is case sensitive. These parameters will generate the binary file demoP5U1.bin**.** For detailed information on controlling binary files, see section "Controlling binary files: SCRAMBLE - SCRAMCAL "later in this guide.
- Make sure that power is removed from the Pluto 5 board and remove the EPROM / RAM expansion card.
- Program the file demoP5U1.bin on to a 27C040 or a 27C4001 EPROM and fit this to U1 on the Pluto5 board.
- Run the software from EPROM, again, using the P&E Debugger software, by clicking the 'Reset' & 'Go' buttons.

  You are reminded that if you attempt to use the EPROM/RAM Expansion card with only a single EPROM in U1 of the Pluto 5 Casino board, you will not be able to access RAM.

[1] **Warning**: If clicking start and run, make sure to type **cmd.exe** instead of **command**. "Command" is a shortcut to c:\windows\system32\command.com which is a dos prompt opening a setting which defaults the display to upper case and will cause problems in part 2.9

5 UTILITIES

5.1 Managing projects for Pluto 5 boards

The following utilities are provided to assist in managing project files. They assume that all project files are in the current directory.

5.1.1 DEP

Sometimes during the course of building your project certain errors will cause the generation of a dependency file to be aborted. Subsequently, make will not attempt to re-build this module since its dependency no longer exists. 'DEP' is used to re-generate a missing dependency file.

Alternatively, a dependency file may have an incorrect library issue number in its path, causing 'make' to fail and specifying the library it is attempting to find. Again, 'DEP' is used to re-generate the dependency file with the correct library paths.

At the command line type   DEP *modname*
Where *modname* is the module name with no file extension

5.1.2 LOC

LOC is used to manually generate a static prototypes file from a 'C' source file.

At the command line type   LOC *modname*
Where *modname* is the module name with no file extension.

5.1.3 MAKEMAP

This utility converts the GNU format map file into a format that can be read by the P&E debug software.

At the command line type   MAKEMAP modname

Where modname is the .s19 file name without extension. The map file will be loaded automatically at the same time as the S19 file.

5.1.4 PRO

PRO is used to manually generate a prototypes file from a 'C' source file.

At the command line type   PRO *modname*
Where *modname* is the module name with no file extension.

5.1.5 SETLIB

This utility must be used each time Heber issues new versions of libraries. Upon release of each new library, you will need to edit lines 3, 4 and 5 of this file to call the most current issue.

Then, if you run this utility from within a project directory, it will update the current library references within all the relevant project files to use the latest library issue releases.

Example:
- Hardware library 81-16082-X, where **X** is the current issue of the Hardware library.
- Peripherals library 81-16083-Y where **Y** is the current issue of the Peripherals library.
- Interface library 81-16178-Z where **Z** is the current issue of the Interface library.

5.1.6 TOUCH

This utility updates the timestamp on specified files. If the library references have been updated, or the project won't build for a non-obvious reason, it is useful to force a complete rebuild of the whole project.

At the command line type   TOUCH *.c
When MAKE is next run, all the C files will be rebuilt.

5.2 Controlling binary files: SCRAMBLE - SCRAMCAL

These Utilities are used to create EPROM binary files from a compiled Motorola S Record file.

These Utilities requires the compiled file to be in a ROM format.

Remake the project after you have set the following in the project ***makefile***: -

script=lnkrom.x.

Read manual 80-17122 : eprom_locations.pdf to find out all the different combination of eprom on the board and their respective command line to type

5.3 Understanding various tools

5.3.1 Images:

5.3.1.1 Cremson Driver

Files are generated in UPPER CASE. Due to the operations of the GCC compilers, following file generation, convert these files to lower case or failures will occur.

IMAG2GNU is used to convert multiple image files into the corresponding assembler source files, suitable for use with the GNU assembler. It can process many common file formats including TIF GIF and BMP. The image file should be in a subdirectory of the default directory called 'source', and the resulting files will be generated in the default directory.

This tool is intelligent enough to convert any 8 or 24 bit colour images into 8 bit colour assembly and palette files[2]. It is strongly recommended that you use this tool starting from a 24 bit picture, as the palette will be very efficient. If starting from an 8 bit picture (which already has its own palette) mismatch in colours will occur when loading the palette created from imag2gnu. Moreover, we strongly advise that whatever the situation, start a conversion from a 24 bit JPG image and not a BMP, because of the number of pixels per line.

- Using the optimised palette makes the images look much better quality on screen. To use the optimised palette you must ensure that all images are processed at once or the palette generated will be incorrect. The palette file is called ***ALL.PAL***.

- The assembly file can be added in the ***makefile*** to include any image in a project (see reference 1 for details)

- The image files have the extension .RAW and the structure of this file will be explained further in this part.

- If you have more than one image file with the same name, even if they are in different formats, the conversion utility will not post process any file after encountering the second file of the same name.

IMAG2GNUSTD is used as above; with the exception that the palette file generated is the standard VGA palette. Images are forced to use this palette and will be modified accordingly. The file ***standard.pal***, which is located in ***\heber\multimedia\images\source*** folder, must be copied to the default directory.

IMAG2GNU16 is used to convert multiple image files into the corresponding assembler source files, suitable for use with the GNU assembler. It can process many common file formats including TIF GIF and BMP. The image file should be in a subdirectory of the default directory called 'source', and the resulting files will be generated in the default directory. Once again, we strongly advise that whatever the situation, start a conversion from a 24 bit JPG image and not a BMP, because of the number of pixels per line.

[2] As a result, we strongly advise the user to split all their pictures into two different directories, images16 and images8. For each directory, it is recommended to create a subfolder source, copy the appropriate pictures and then use the correct tools to convert them.

- The assembly file can be added in the ***makefile*** to link any picture to the project. (look at reference 1 for more details)

- The image files have the extension .BIN and the structure of this file will be explained further in this part.

This tool will only convert 24 bits colour images into 8bit colour assembly and palette files. If another format other than a 24 bit colour is found in the subdirectory, it will not be converted.[2]

Description of the picture's format.

.RAW and .BIN picture files are completely uncompressed, unpacked and unpadded image data files built on the Image Alchemy HSI Raw File format. They tend to be larger than almost any compressed file format but they have the advantage of being able to be used directly, without any file decryption.

The file is composed of 800 bytes for the header first, followed by either HSIZE*VSIZE ***byte*** if the image is an 8 bit, or HSIZE*VSIZE ***word*** if the image is a 16 bit.

1) Header file: 800 bytes.

6 bytes used to identify the file as an HSI Raw file:	0x6D 0x68 0x77 0x61 0x6E 0x68
2 bytes used to identify the version HSI file:	0x00 0x04
2 bytes indicating the width	
2 bytes indicating the height	
2 bytes indicating the number of entries for the palette	
2 bytes indicating the horizontal resolution of pictures in dots per inch (NOT USE)	
2 bytes indicating the Vertical resolution of pictures in dots per inch (NOT USE)	
14 bytes unused	

Then, for a 256 colour picture, (image.raw), the palette has a maximum size of 256 (colours) * 3 (one byte for red, one for green, one for blue)[3]. For a 16 bit picture, no palette will be defined so from byte 32 to byte 800, the values will be equal to 0.

2) Picture data.

According to the extension of the files, each pixel of the image will be accessed either as:

- **8 bit Indirect Colour: image.raw**

In this mode each pixel is represented by an 8 bit code. The code is used as a reference into a 256 value colour palette. The colours in the palette are selected from possible 262,144 colours.

- **16 bit Direct Colour: image.bin**

In this mode each pixel is represented by a 16 bit code; five bits are used for Red, Green and Blue and one bit as a reserved intensity code (bit 15). This mode will allow up to 32,768 colours to be displayed.

The 16 bits of the colour value are as follows:

INT	**RED**					**GREEN**					**BLUE**				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

5.3.1.2 Calypso Driver

Palconv will generate a palette source file from an Image Alchemy(tm) format palette file (e.g. ***standard.pal***) The source file ***palette_name.c*** will be produced containing a converted palette table. This table is declared as an array of palette entry structures:

const PALETTE_RGB palette_name[256] = { 256 palette entry values… };

[3] Heber is not using the specific palette for each file. Using the optimised palette generated when converting all the pictures at once makes the images look much better quality on screen

where the palette entry type PALETTE_RGB is:

```
typedef struct palette_rgb {
        UBYTE alpha;        /* 1= enable alpha blending for this colour (console layer only)*/
        UBYTE red;          /* red value */
        UBYTE green;        /* green value */
        UBYTE blue;         /* rblue value */
} PALETTE_RGB;
```

In order to use the palette the 'palette_name.o' file must be added to the ***makefile***, and be declared as an external variable in the file ***palettes.h***:

```
extern const PALETTE_RGB    palette_name[];
```

It can then be loaded in the game code with the CalypsoLoadPalette() function in the enhanced video driver, see the Software user manual – Enhanced Calypso driver section for details.

IMAGCONV8 is used to convert multiple image files into the corresponding C source files, suitable for use with the GNU compiler. It also produces an optimised Image Alchemy palette file ***OPTIMUM.PAL***, and a corresponding C source file ***palette.c***. The image files should be in a subdirectory of the default directory called 'source', and the resulting files will be generated in the default directory.

This tool is intelligent enough to convert 8 or 24 bit colour images from many common file formats including TIF GIF and BMP into 8 bit colour C image source files[4]. It is strongly recommended that you use this tool starting from a 24 bit picture, as the palette will be very efficient. If starting from an 8 bit picture (which already has its own palette) mismatch in colours will occur when loading the palette created from this tool. Moreover, we strongly advise that whatever the situation, start a conversion from a 24 bit JPG image and not a BMP, because of the number of pixels per line.

- Using the optimised palette makes the images look much better quality on screen. To use the optimised palette you must ensure that all images are processed at once or the palette generated will be incorrect. The palette file is called ***OPTIMUM.PAL***.

- The C files can be added in the ***makefile*** to include any image in a project. The image names must be declared as external variables in the file ***images.h***:

  ```
  extern const CALYPSOIMAGE image0;
  extern const CALYPSOIMAGE image1;
  ```

If you have more than one image file with the same name, even if they are in different formats, the conversion utility will not post process any file after encountering the second file of the same name.

IMAGCONV8STD is used as above; with the exception that the palette file generated is the standard VGA palette. The file ***standard.pal***, which is located in ***\heber\multimedia\images\source*** folder, must be copied to the default directory. The palette C source file created is called ***stdpalette.c***.

IMAGCONV16 is used as above, with the exception that the C source files generated are 16 bit images, and hence no palette files are generated.

Description of the image source file

A source file for each image with the name ***image_name.c*** containing image data will be produced, containing the following structures:

```
static const PALETTE_RGB palette[] = {
{
        palette data…
};
```

[4] As a result, we strongly advise the user to split all their pictures into two different directories, images16 and images8. For each directory, it is recommended to create a subfolder source, copy the appropriate pictures and then use the correct tools to convert them.

```
const CALYPSOIMAGE image_name = {
            image parameters…
            palette data.. .
            bit map data ….
};
```

where the type *CALYPSOIMAGE* is defined as:

```
typedef struct calypsoimage {
            ULONG size_bytes;               /* image size bytes */
            WORD color8;                    /* 1=8bit 0=16bit */
            WORD palette_size;              /* number of palette entries */
            WORD width;                     /* width in pixels */
            WORD height;                    /* height in lines */
            const PALETTE_RGB *palette;     /* palette data */
            const WORD *data;               /* image data */
} CALYPSOIMAGE;
```

The palette table in each individual image source file is currently unused by default, but may be loaded in the game code with the CalypsoLoadPalette() function, passing 'image_name.palette' as the palette to load. See the Software user manual – Enhanced Calypso driver section for details.

5.3.2 Fonts

5.3.2.1 Cremson Driver

TTF2GNU is used to convert a font file into the corresponding assembler source file, suitable for use with the GNU assembler. It can process any font from windows as long as it is a standard font and not a Unicode font (for example, symbols or Chinese).

At the command line type: ttf2gnu {filename} {size}

Where {filename} is TrueType filename without extension and {size} is point size of font required

Example: ttf2gnu tempo 32 will convert tempo.ttf (True Type) to tempo32.s (Heber Source)

The font can be anywhere in the project as the only requirement is to create a folder "tmp" on the root of the drive (for example, C:\tmp). If this folder is not created, the command prompt will return during stage 1: ttf2bdf: unable to open temporary file '/tmp/ttf2bdf3944'.

5.3.2.2 Calypso Driver

FONTCONV is used to convert a font file into the corresponding C source file, suitable for use with the GNU compiler. It can process any font from windows including Unicode fonts.

At the command line type: fontconv {filename} {size} {Unicode_page}

Where {filename} is TrueType filename without extension, {size} is point size of font required and {Unicode_page} is the portion of the Unicode font to convert. The font file must be in the current directory when the command is run.
The Unicode page range must be specified because 16-bit Unicode character codes (UTF-16) cover a range of 65536 characters. The Enhanced Calypso Driver divides this range into 512 code pages of 128 characters. Therefore each Calypso font file generated contains 128 characters. Multiple Calypso font files can be assigned to cover different Unicode code ranges.

Unicode_page	UTF-16 character range
0	0x0000 - 0x007F (Latin)
80	0x0080 - 0x00FF (Latin extended)
100	0x0100 - 0x017F (Cyrillic)
180	0x0180 - 0x01FF … …
... ...	
... ...	
FF00	0xFF00 - 0xFF7F
FF80	0xFF80 - 0xFFFF

Examples:
True type font file: *example.ttf*
The following command lines:
fontconv example 32 0
fontconv example 32 80
fontconv example 32 400
will produce the following Calypso font files:
example32_0.c
example32_80.c
example32_400.c

In order to use the font in a project the object file (e.g. 'example32_0.o') must be added to the makefile, and the font must be declared in fonts.h:

extern const CALYPSOFONT example32_0;

For details of using the fonts in game code, see the Software user manual – Enhanced Calypso driver section.

5.3.3 Sound

SNDCNV32 converts a .WAV file into an assembly source file suitable for use with the GNU assembler. The file can then be compiled and linked with the software.

At the command line type SNDCNV32 WAVfile*name samplerate*
Where WAVfile*name is the name of the WAV file with no extension*
And samplerate is the sample rate required for the output file, for example, 16000.

<u>**NOTE**</u>

1. SNDCNV32 requires a file called ***template***, which must be copied to the current working directory. This file is located in ***\heber\templates***
2. On generation of the assembler source files, you will get either ***sourcefile.s***, which is mono, or ***sourcefilel.s*** and ***sourcefiler.s***, which are left and right. Sndcnv32 interrogates the WAV file to determine the type of assembler source file generated.
3. Command line characters must reflect the function definition in the game code.

   Example: - In ***demo.c*** you will see

   ```
   #define     SAMPLE1     XNUDREP2
   #define     SAMPLE2     BELL1
   ```

   When using sndcnv32, the command line must be in the same character “case”, ie: -

   ```
   sndcnv32 BELL1 16000
   ```

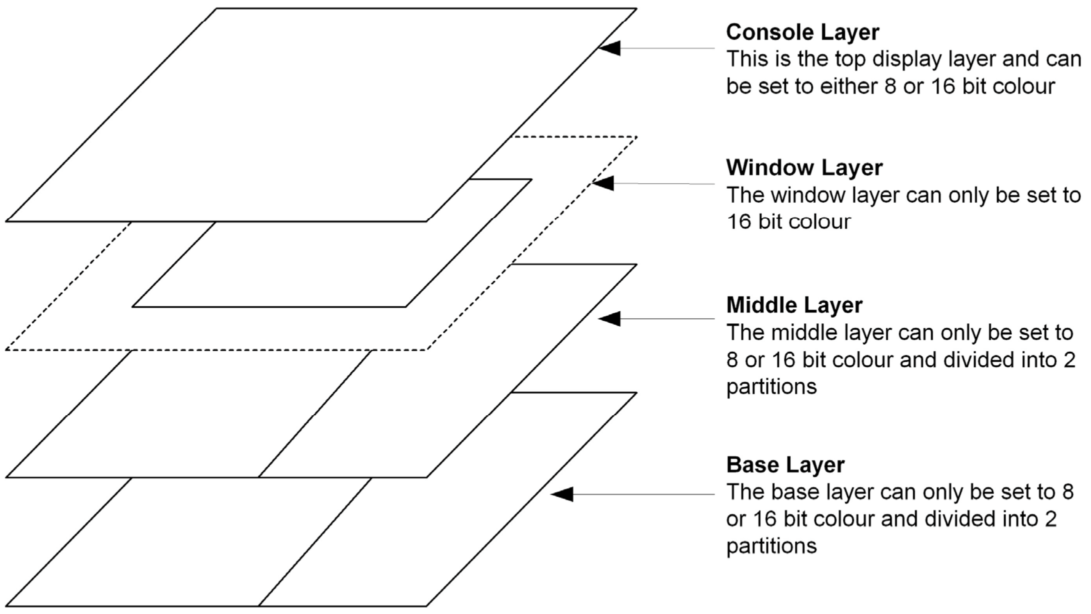
6 INTRODUCTION TO VIDEO PROGRAMMING:

The Pluto5 Calypso 16 video board uses the Fujitsu Cremson ® Graphic Controller.

This section is a general description of how the video works but for development purposes, the user will need to refer to the Pluto 5 Development Kit Software User Manual, 80-16040, supplied on the Development Kit CD.

6.1.1 Layer Architecture Video controler

To display an image on a screen, the processor can overlay four layers of display frames on top of each other. Each layer has different properties such as the number of bytes per pixel, ability to be split, display priority, possible transparency, etc…


Console Layer
This is the top display layer and can be set to either 8 or 16 bit colour
Window Layer
The window layer can only be set to 16 bit colour
Middle Layer
The middle layer can only be set to 8 or 16 bit colour and divided into 2 partitions
Base Layer
The base layer can only be set to 8 or 16 bit colour and divided into 2 partitions

- **Console Layer.** The Console display frame contains a single logical graphics field (**C-Layer**) which has the highest display priority and can display 8 or 16 bits per pixel images.

- ***Window Layer.*** The window layer contains a single logical graphics field (**W-Layer**) which has the second highest display priority and can only display 16 bits per pixel images. No transparency can be set for this layer.[5]

- ***Middle Layer.*** The middle display frame is split vertically into left (**ML-Layer**) and right (**MR-Layer**) logical graphics fields. It is possible to set one of to represent the dimensions of the physical display. In this case the layer behaves as though it has a single display field. The Middle left layer and Middle right layer logical graphics fields have the third highest display priority and can display 8 or 16 bits per pixel images. They also support double buffering.

- ***Base Layer.*** The base display frame is split vertically into left (**BL-Layer**) and right (**BR-Layer**) logical graphics fields. It is possible to set one of them to represent the dimensions of the physical display. In this case the layer behaves as though it has a single logical graphics field. The Base left layer and Base right layer logical graphics fields have the lowest display priority and can display 8 or 16 bits per pixel images. They also support double buffering

6.1.2 Manipulating and accessing layers

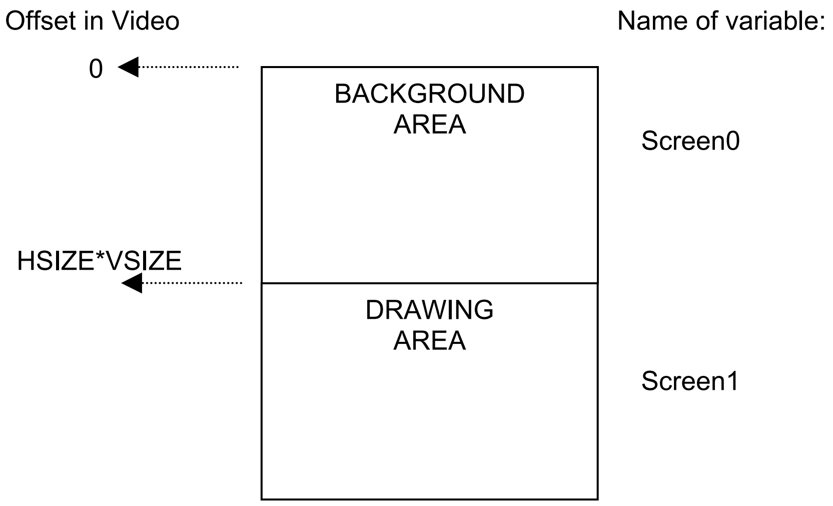
[5] Unlike the other layers, the *window layer* (not the logical graphics fields) size is not necessarily full screen. The position parameter as well as the size of the layer must be defined independently for it to be visible.

Each layer can be enabled or disabled according to the programmer requirements. When needed the programmer will initialise a layer in order to set: -

- The dimensions of the layer;
- The area of video memory assigned to the layer;[6]
- The transparency colour.

The video processor uses 15Mb of local display memory, which can be assigned freely according to the application. When the display is static (background, picture, drawing), one area of memory is assigned to one layer.

Imagine a monitor displaying 1024 pixels horizontally (HSIZE) by 768 pixels vertically (VSIZE) where a user wants to set a background on the base layer (256 colours) and draw on the middle layer (256 colours). In this case the memory can be organised as follows:



The initialisation process sets the base layer to point at screen0=0 and the middle layer to point at screen1=HSIZE*VSIZE. Then, on an interrupt from the monitor, these two layers will be refreshed with the content of the memory.
Video memory should be cleared on initialisation.

6.1.3 Functional overview:

6.1.3.1 Double buffering:

Double buffering is a technique used for displaying flicker free animations which is supported by the **Base** and **Middle** display layers on the Cremson Video processor.

When displaying animation, double buffering is implemented on the Calypso 16 by assigning **two** areas of video memory to one display layer. Whilst one of the areas is being displayed, the image data on the other area of memory can be updated. When the update is completed the screens are switched and the updated data is displayed.

To avoid display flicker, the programmer has a control on the refresh rate for simple animation. If filling the memory is more time consuming than a single interrupt cycle of the video processor, the programmer can also toggle screens manually as soon as the status of the update is completed

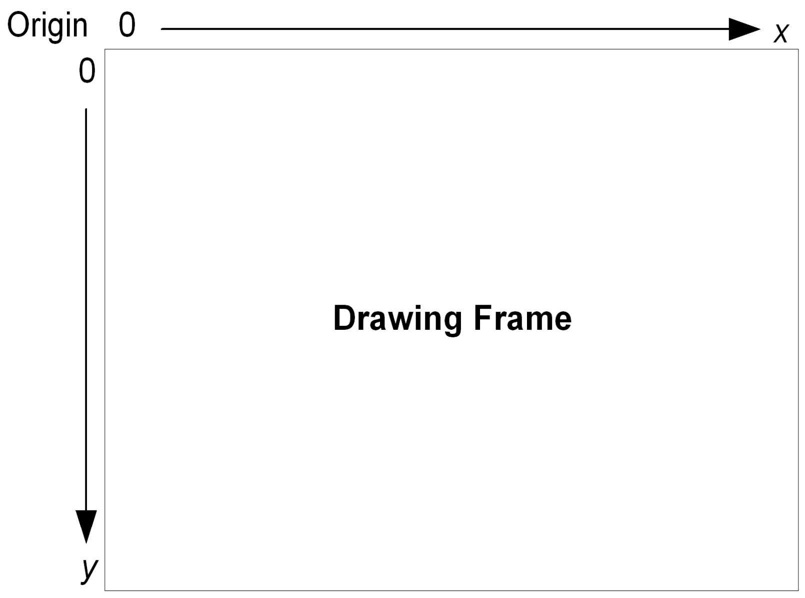
6.1.3.2 Writing text to a screen:

The video device driver provides functionality to allow the game developer to write text to the video display. The driver currently supports 8 bit font files, so can therefore only be used to write text to screens defined as 8 bit (INDIRECT_COLOUR). Explanation on how to create a font is available part 3.10.

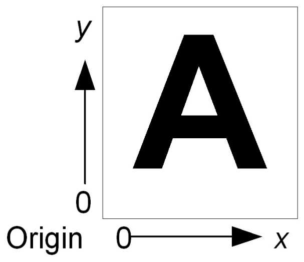
[6] For the window layer, only a call to SetWindowLayer will set the dimensions of the visible display, the area of memory should be defined regarding these parameters rather than the dimensions of the monitor.

The following functionalities are possible when working with text:
Set a frame to be a drawing frame. Place a cursor into this frame and start writing.
Load a prepared font file from system memory to video memory.
Set the background and foreground colours for the characters displayed.
Draw a single character or a full character string

Setting the text on a layer is quite easy as long as the size of the font is considered. The origin position within a drawing frame or an image (x,y co-ordinate 0,0) is at the top left of the drawing frame.



The origin position of a character bitmap in a font file (x,y co-ordinate 0,0) is at the bottom left of the character.



When setting the cursor position to write text a drawing frame, the y axis co-ordinate must be set to the desired position increased by the height of the character in pixels to ensure that the character is displayed.
If the edge of the drawing frame is reached the cursor position will word warp to the next line position of the screen.

6.1.3.3 Others:

Pluto 5 such as supports many other functions:
- Loading big images into the video memory and then display it onto the screen.
- Loading sprites and animating them.
- Drawing basic shapes (lines, rectangles, circles).
- Accessing directly to a particular pixel on screen.

Pluto 5 software libraries is provided with two different video drivers. CremsonDriver and CalypsoDriver.
A full description of these two drivers is available on the Software User Manual and a comparison between these two drivers is available in Enhanced Calypso Driver – General Description.

7 RUNNING THE DEMONSTRATION PROJECT

This software is located in the Demo EPROM pair. Ensure that they are fitted in U1 & U2 of the Pluto 5 Casino board.

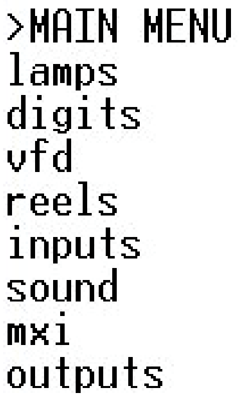
The demonstration software shows, one function at the time, how to drive lamps, seven segment digits, outputs and reels. It also shows how to read direct or multiplexed inputs and control an output.

Once the demonstration software is running it uses a serial port on the Pluto 5 Casino Board, which is fitted to the Pluto 5/C Evaluation Board, to communicate with the PC.

Before running the software from P&E or directly from the EPROM, ensure that the HyperTerminal settings are 9600 baud, 8 Data Bits, No Parity, 1 Stop Bit and No Flow Control.

- Open HyperTerminal and set it to communicate with RS232 Channel B of the Pluto 5/C Evaluation Board.
- Hit <ENTER> and the main menu command summary should be displayed. If the text is garbled, set hyperterminal to append line feeds to incoming carriage returns.
    - To do this Select File | Properties and click the setting tab.
    - Click the ASCII setup button.
    - In the ASCII setup dialog check the box which says ‘append line feeds to incoming line ends’.

Apply power to the Pluto 5/C Evaluation Board and, if fitted, run the program from the debugger. A list of commands should be displayed as follow.



```
>MAIN MENU
lamps
digits
vfd
reels
inputs
sound
mxi
outputs
```

The main menu lists all the possible functions of the demonstration program. Try typing one of the commands from the main menu. The commands are described in detail in the following sections.

7.1 Lamps

The lamps can be illuminated on a strobe-by-strobe or row-by-row basis or individually. Each lamp can be set constant, flashing and flashing fast. The brightness of a line or a column of lights can be modified.

7.1.1 Strobe

Lamps on column 0 are first illuminated. User has the choice to Press ‘x’ to exit or press any other key to advance to the next column.

7.1.2 Row

Lamps on row 0 are first illuminated. . User has the choice to Press ‘x’ to exit or press any other key to advance to the next row.

7.1.3 Caterpillar

This is another subprogram for use of multiplexed lamps. In previous versions of the development kit, the demo program provided the option to light lines or columns on multiplexed lamps. Most of the entertainment games will use the lamps to light a symbol. For this purpose, understanding how to drive a single lamp is sufficient.

This program shows how to control individually one lamp. As an example, each lamp is accessed separately and lit. The programs goes through the 256 available lights and then go back to the first one in order to switch all of them on at the time.

7.1.4 Suit

For graphics and video purposes, understanding how to load images into the video memory and access them when needed is very important. This part will be illustrated when running the calypso 16 Demo program but a programmer could use image functionalities right away, quite easily, with lights.

This is a small demonstration program providing a first glance of loading pictures (which will be an issue when working with graphics). A quick tutorial is available to explain how to load a 16*16 picture.

7.1.5 On

The status for the strobe and row sequences is set to ON. If status was changed to flashing or flashing fast, it will go back to original settings.

7.1.6 Flash

The status for the strobe and row sequences is set to flash (F1). Nothing actually happens until the next time a command is typed. At this time the multiplexed light will start flashing slowly

7.1.7 Fastflash

The status for the strobe and row sequences is set to fast flash (FF1). Nothing actually happens until the next time a command is typed. At this time the multiplexed light will start flashing faster.

7.1.8 Dim

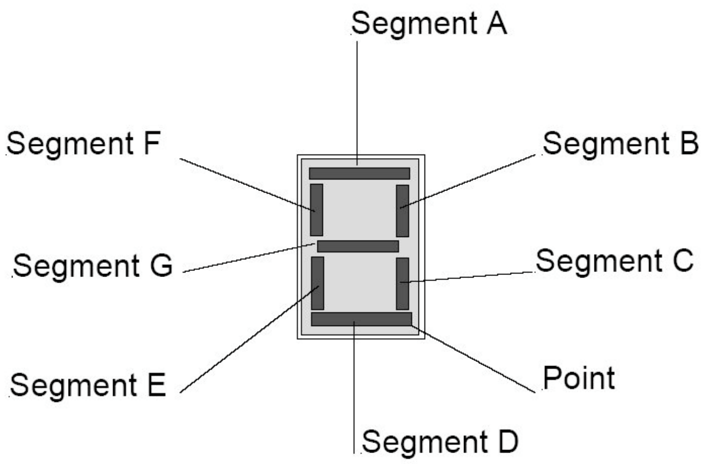
The brightness for the strobe and row sequences is set to dim (ON & PERIOD0). Nothing actually happens until the next time a strobe or row command is typed. At this time, the multiplexed light will be less bright.

7.1.9 Exit

Exit back to main command menu.

7.2 Digits

Two display panels are available. Each LED is accessed independently and lit according to the segment identity set to 1 or 0.



7.2.1 Digit

Hexadecimal characters will be displayed one after the other when typing any key. These characters are displayed on the top LED (LED16 up to 31).

The sequence then recommences until ‘x’ is pressed.

7.2.2 Segment

The segments of the entire bottom LED (LED0 up to 15) digits are illuminated in turn starting with SEGA up to SEGH. The sequence then recommences until ‘x’ is pressed.

7.2.3 Exit

Exit back to main command menu.

7.3 VFD

The Vacuum Fluorescent Display is able to display a string of 16 characters at the time. This screen can be accessed quite easily and display messages to the user.

7.3.1 Abc

The whole alphabet will scroll, from right to left, across the VFD display until any key is pressed. When pressed, the display will clear.

7.3.2 Inputstring

This sample program illustrates how easily a string can be read on the serial port and then displayed directly on the VFD.

7.3.3 Exit

Exit back to main command menu.

7.4 Reels

Entering the general command menu for the reels initialises up to four reels to their reference position, if they are connected to the Evaluation board, and the “reels” menu is displayed. Up to six wheels can be configured on the evaluation board but as a first approach, the demonstration is set up for four. Each reel can be accessed independently of the others, to spin one way or the other.
The demonstration program is set to operate a “*Starpoint 12RM*” reel. This reel is designed on a 48 step motor containing 16 symbols. To move one nudge, the motor needs to move three steps.

7.4.1 Init

Initialise the reels to their reference positions. If no reels are connected to the development kit, HyperTerminal will reply “bad bad bad bad” otherwise reels will be initialised to their reference position and HyperTerminal will reply “Ok Ok Ok Ok”

7.4.2 Step

Step one of the four installed reels forwards and/or backwards using function:

Step(reel,step_forward,step_backward)

Where:

- **reel** is a number between 0 and 3
- **step_forward** is a number between 0 and 16
- **step_backward** is a number between 0 and 16

7.4.3 Spin

Spin all reels and wait for them to stop in sequence, but not necessarily at the reference position.

7.4.4 Skill

Spin all reels. Each time a key is pressed the next reel stops.

7.4.5 Alternate

Spin all reels. Two of the reels spin forwards and two spin backwards. Each time a key is pressed the next reel stops.

7.4.6 Status

Report the reel status in the following format
TxCy where: -
x= Number of times a reel tab has been expected but not detected.
y= Number of times the position of the reel has been corrected as a result of the tab being detected.

7.4.7 Exit

Exit back to main command menu.

7.5 Inputs

Prints a string representing the status of the system inputs IP0 to IP31 followed by the DIL switches bank1 1-8 and bank 2 1-8.

‘1’ represents an active input. (Switch closed)
‘.’ represents an inactive input. (Switch open)

The input status is checked every second and the result is then displayed. Be careful with the values returned from IP0 to IP5 as they are shared with the sensor returning the position of the reels.
If the command “reels” was used before, the inputs from IP0 to IP5 will return the status of the reels. If one of the reels is in the position sensor, the input is considered on. Even though the switch may be in the “off” position, the light representing the input will still be switched on and the program will return 1 for the status of the input.

7.6 Sound

Demonstrates the sampled sound. Ensure that the speaker switches are turned “on”.

7.6.1 Play1

Plays a looping tune on channel 1.

7.6.2 Incvol1

Increment the volume of channel 1. Does not take effect until the tune is re-started.

7.6.3 Decvol1

Decrement the volume of channel 1. Does not take effect until the tune is re-started.

7.6.4 Play2

Plays a ‘bell’ sound on channel 2.

7.6.5 Incvol2

Increment the volume of channel 2. Does not take effect until the next tune begins.

7.6.6 Decvol2

Decrement the volume of channel 2. Does not take effect until the next tune begins.

7.6.7 Exit

Exit back to main command menu.

7.7 Mxi

MXI demonstrates the operation of the multiplexed inputs driver for hardware having two input strobes. This part of the demonstration program is working in a similar way to the “inputs” section described above.

The main difference is not in the string displayed on HyperTerminal but in the driver controlling the inputs or the multiplexed input. The “inputs” program is a direct reading of the input. The MXI is an indirect test of the inputs. The value display is the status of the input written in the memory.

In this program, the value of the DIL switches is not returned. The inputs are re-sampled every second and the new data is output. Press any key to exit.

7.8 Outputs

Prompts the user to enter a Heber output number. Enter a number 0 to 63 and press <enter>. The output is asserted. Press ‘X’ to negate the output and exit, press any key to negate the output and prompt for a new output to assert.

8 RUNNING THE CALYPSO16 PROJECT

8.1 Explanation of the code:

The example program provided with the kit is divided into three different part.

The first part displays 6 cherries into different area of the screen. In order to accomplish the display, the program starts by loading the sprites from the program memory into the video ram where it will access the pictures faster. Then it puts the sprite into an area of memory, which was initialised to be the middle layer. Finally on a vertical interrupt the cherries are displayed.

The second part fills an area of memory (which was initialised to be the base layer) with the CYAN colour. As it is a lower layer than the base layer, it will look like a background. Then it writes to the area of memory (which was initialised to be the middle layer) the same text with three different fonts. As the cherries where not deleted from the memory before, the cherries, the text and the background will appear to be the same united image.

The third part of the program deletes the memory and creates:

- Two areas of memory to double buffered the Middle Left Layer screen. (Animation of cherries)
- One area for the base left layer in order to contain an image bigger than the display area. (Scrolling of an image).
- One area of the base right layer to have a fix background which will be a small part of the big image.

8.2 Detecting a CD-ROM

The following will enable you to run a demonstration of CD-ROM images that are called from within the demo program in the **\heber\projects\calypso16-6** folder.

1) Connect a CD-ROM drive to the IDE connector P4 on the Calypso 16 card and ensure the Development Kit CD-ROM is inserted in the CD-ROM drive.
2) Go to the **\heber\projects\calypso16-6** folder.
3) Ensure that the following are **NOT** commented out in: -
   a) The config.c file.

```
const CDROMCFG CDRomCfg=
{
1500,    // Initialise time out
};
```

   b) The devices.c file

```
{&CDRomDevice          ,0        ,0,0},
```

4) Uncomment the following in the vdemo.c file

```
//          if (DetectCDRom())
//          CDRomFileStreamDemo(1000);
```

5) Ensure that if you have, or are using, the Compact Flash, the following **IS** commented out in the devices.c file. This line may not be present.

```
{&ATADevice            ,0        ,0,0},
```

Please note that you can only use the CD-ROM drive if the Compact Flash adaptor is not installed on the Calypso 16 card and visa versa.

9 RUNNING THE COMPACT FLASH EXAMPLE PROJECT

9.1 Introduction

The example program initialises the Calypso16 card with a 16-bit colour layer overlayed with an 8-bit colour layer.

Commands are provided to allow the following to be loaded from the Compact Flash card to the above layers: -
- full screen background images.
- images.
- animations.

The Compact Flash card provided with the Pluto 5 Enhanced Development Kit contains a number of example images, backgrounds and animations.

9.2 Installing the Compact Flash board.

9.2.1 PCB Configuration

The configuration of boards needs to be as follows: -

Pluto 5/5C Evaluation board
Pluto 5 Casino
Calypso 16 Video Graphics card
EPROM/RAM Expansion card

9.2.2 Data

Connect the 40-way ribbon cable from P2 on the Compact Flash board to P4 on the Calypso 16 Video Graphics card.

- On the Compact Flash board, pin 1 of connector P2 is nearest to the power connector P3.
- On the Calypso 16 card, pin 1 of connector P4 is furthest away from the serial connector P5.
- The red stripe on the 40-way ribbon cable denotes pin 1.

9.2.3 Power

Connect the Compact Flash board power lead from P3 on the Compact Flash board to the +5V OUT on the Pluto 5/5C Evaluation board.

- Note that if you connect this cable to either the + 12V OUT or the – 12V OUT on the Pluto 5/5C Evaluation board, you **will** damage the Compact Flash card.

9.2.4 Compact Flash card

Insert the Compact Flash card supplied with the Pluto 5 Enhanced Development Kit into P5 of the Compact Flash board. Due to the cards orientation slots, it will only fit one way without forcing it.

9.2.5 VGA Monitor

Connect a VGA monitor to P6 of the Calypso 16 Video Graphics card.

9.2.6 Serial Connection

Ensure that the serial cable is connected between the Pluto 5/5C Evaluation board Channel B and your PC. Start HyperTerminal (or similar). The settings should be 9600 baud, 8 Data Bits, No Parity, 1 Stop Bit and No Flow Control.

9.3 Running the Project

9.3.1 Loading the Project File to RAM

Power up the Pluto 5/5C Evaluation board. Load the Compact Flash example project, ***\heber\projects\compact_flash-6\cf.s19***, to RAM using the P&E® software and press ***GO*** to run.

On HyperTerminal, the following should appear: -

```
ATA Disk Drive Test:

Model: Flash Card
Rev: 14/05/0
Serial No: CF000000
Cylinders: 500, Heads:   8, Sectors:  32

Mounting Drive...

FAT32 Partition found on Primary Partition

Volume: NO NAME      Size: 65 Mbytes.


Type <help> for a list of commands.

CF:\>
```

Type ***help*** and enter. The following should appear: -

```
Type <help> for a list of commands.

CF:\>help

Commands:

dir                  - directory listing
cd <directory name>  - change directory
load8 <file name>    - load   8-bit image
load16 <file name>   - load  16-bt image
clear8               - clear  8-bit layer
clear16              -clear 16-bit layer
bgnd8                - load   8-bit background
bgnd16               - load  16-bit background
anim16               - show  16-bitanimation (MULTIPLE FILE)
anim16s              - show  16-bit animation (SINGLE FILE)
type <filename>      - echo file to terminal

CF:\>
```

This is a list of commands that will allow you to run the various demonstrations.

9.3.2 Listing and changing directories.

- ***dir*** lists all the file contents of the current directory.

- ***cd*** changes the current directory, for example,
  cd images8
  changes the current working directory to sub-directory ***images8***.

- ***cd ..*** takes you to the parent directory

- ***cd \*** takes you to the root directory.

9.3.3 Running the examples

The following examples can be typed from within any directory and will load to the screen connected to the Calypso 16.

Typing ***bgnd8*** loads a full screen, 8-bit background image.

Typing ***clear8*** will clear the 8-bit image.

Typing ***bgnd16*** loads a full screen, 16-bit background image.

Typing ***clear16*** will clear the 16-bit image.

Ensure that you have cleared both layers.

Typing ***anim8*** shows an 8-bit animation of spinning dice.

Typing ***clear8*** will clear the 8-bit animation.

Typing ***anim16*** shows a 16-bit animation of spinning dice.

Typing ***clear16*** will clear the 16-bit animation.

To run the following examples you will need to change to the required directory.

Change to the ***images8*** directory. Typing ***load8 image0.raw*** will load a single 8-bit image. As before, typing ***clear8*** will clear the layer.

Change to the ***image16*** directory. Typing ***load16 image0.bin*** will load a single 16-bit image. As before, typing ***clear16*** will clear the layer.

9.3.4 Loading Text Files.

Change to the root directory. Typing ***type hamlet.txt*** echoes the contents of the ***hamlet.txt*** file to the HyperTerminal screen.

10 RUNNING THE CALYPSO ENHANCED PROJECT

10.1 Setup

10.1.1 PCB Configuration

The configuration of boards needs to be as follows: -

Pluto 5/5C Evaluation board
Pluto 5 Casino
Calypso 16 Video Graphics card
EPROM/RAM Expansion card

10.1.2 VGA Monitor

Connect a VGA monitor to P6 of the Calypso 16 Video Graphics card.

10.1.3 Building the project

There are two build versions, for 8-bit and 16-bit colour. The differences are taken care of by a define switch in the compiler flags of two separate makefiles. Run the commands below in the project directory ***\heber\projects\calypso_enhanced\***.

To build the 8-bit colour version:
make -f make8

output:
video8.s19

To build the 16-bit colour version:
make -f make16

output:
video.s19

10.2 Running the Project

10.2.1 Loading the Project File to RAM

Power up the Pluto 5/5C Evaluation board. Load the project to RAM using the P&E® software and press ***GO*** to run.

The program will cycle through scrolling alpha blended playing cards, animated dice and displaying text in Unicode and ASCII fonts, first in 640x480 resolution, then 800x600.

11 RUNNING THE TOUCHSCREEN EXAMPLE PROJECT

11.1 Setup

Running the touchscreen example project requires that the code is edited depending on whether an ELO or Microtouch touchscreen monitor is to be used. This is detailed in the Pluto 5 Development Kit Software User Manual 80-16040, Touchscreen Driver section.

11.1.1 PCB Configuration

The configuration of boards needs to be as follows: -

Pluto 5/5C Evaluation board
Pluto 5 Casino
Calypso 16 Video Graphics card
EPROM/RAM Expansion card

11.1.2 VGA Monitor

Connect the VGA cable from the touchscreen monitor to P6 of the Calypso 16 Video Graphics card.

11.1.3 Serial Connection

Connect the serial cable from the touchscreen monitor to P17 (RS-232 channel C) on the Pluto 5/5C board (This is dependant on which port was specified in the touchscreen structure in ***config.c*** in the project)

11.2 Running the Project

11.2.1 Loading the Project file to RAM

Once the project is correctly configured and rebuilt, connect the power up the Pluto 5/5C Evaluation board. Load the configured touchscreen example project to RAM using the P&E® software and press ***GO*** to run.

11.2.2 Calibrate and run

A yellow box with arrows pointing inward will appear on a grey screen. Touch the middle of the box, another box appears, touch the middle of that. A king symbol then appears, this will move to wherever the screen is touched.

12 CONNECTING DEVICES TO THE PLUTO5 BOARD

Refer to the Pluto 5 Casino User Manual, 80-15744, for details on the Pluto 5 Casino connections.

REFERENCE 1: INSTRUCTIONS TO LOAD A PICTURE[7]

- Create a subfolder to the directory (c:\heber\demo-5r2) entitled images. Create a subfolder of this one (c:\heber\demo-5r2\images) entitled source and copy the five 16*16 bitmap images. (club.bmp, diamond.bmp, heart.bmp, spade.bmp, smile.bmp)

- Open a command prompt on "*c:\heber\demo-5r2\images*" and run IMAG2GNU from there. 12 new files have been created there. Rename all the files from uppercase to lowercase (HEART.S into heart.s) as it will cause problems running the makefile later. Copy club.s, diamond.s, heart.s, spade.s, smile.s into c:\heber\demo-5r2

- If you are looking at one of these files, for example heart.s, the second line is:

  ```
  .globl heart_width,heart_height,heart.
  ```

  These three assembly variables will be stored into the memory as global variables:
  - a word for heart_width
  - a word for heart_height
  - and 256 bytes for heart.

- In order to access these variables, one will need to create a new file images.h and include it into demo.c.

  ```
  #include        "images.h"
  ```

- From there the image to be access has to be declare for the program to know it is stored in the memory somewhere. This image is an external variable and needs to be declared as follow

  ```
  extern const BYTE heart[];
  ```

  As the size of the image is know there is no need to declare the size of the image (at other time for example video with calypso it will have to be declared as followed)

  ```
  extern const WORD     heart_width,heart_height;
  ```

- Finally, to store the image in the memory, the makefile will need to be modified as well. Where the dependencies of the project is declared:

  ```
  objects = game.o config.o devices.o demo.o custom.o  \
  $(lhdw)/mxidev.o sysram.o idram.o bell1.o xnudrep2.o \
  fpga.o
  ```

  The image to link to the project will also need to be declared by adding *heart.o* to the list of object files.

At this stage, the picture is stored in memory and can be accessed whenever required

```
for(cpt=0;cpt<256;cpt++)
{
        if(heart[cpt]==0)
                Do something
        else
                Do something else);
}
```

[7] This reference is detailed for images but sounds and fonts work in the same way.