Platform Developer’s Kit

PAL Cores manual
Celoxica, the Celoxica logo and Handel-C are trademarks of Celoxica Limited.

All other products or services mentioned herein may be trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous development and improvement. All particulars of the product and its use contained in this document are given by Celoxica Limited in good faith. However, all warranties implied or express, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. Celoxica Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any incorrect use of the product.

The information contained herein is subject to change without notice and is for general guidance only.

Copyright © 2005 Celoxica Limited. All rights reserved.

Authors: RG

Document number: 1

Customer Support at http://www.celoxica.com/support/

Celoxica in Europe  Celoxica in Japan  Celoxica in the Americas
T: +44 (0) 1235 863 656  T: +81 (0) 45 331 0218  T: +1 800 570 7004
E: sales.emea@celoxica.com  E: sales.japan@celoxica.com  E: sales.america@celoxica.com

www.celoxica.com
## Contents

1 **PAL CORES: INTRODUCTION** .............................................................. 5

2 **PAL KEYBOARD** ............................................................................... 6

   2.1 **PAL KEYBOARD API** ................................................................ 6

   2.1.1 PAL Keyboard Core: PalKeyboardRun ........................................... 6

   2.1.2 PAL Keyboard Core: PalKeyboardDisable ....................................... 7

   2.1.3 PAL Keyboard Core: PalKeyboardEnable ......................................... 7

   2.1.4 PAL Keyboard Core: PalKeyboardReset .......................................... 7

   2.1.5 PAL Keyboard Core: PalKeyboardReadASCII .................................... 8

   2.1.6 PAL Keyboard Core: PalKeyboardReadScanCode ............................... 8

   2.2 **PAL KEYBOARD example** ............................................................ 9

3 **PAL CONSOLE** .................................................................................. 11

   3.1 **PAL CONSOLE API** ................................................................... 11

   3.1.1 PAL Console Core: PalConsoleRun .................................................. 11

   3.1.2 PAL Console Core: PalConsoleDisable ........................................... 12

   3.1.3 PAL Console Core: PalConsoleEnable ............................................. 12

   3.1.4 PAL Console Core: PalConsoleReset .............................................. 13

   3.1.5 PAL Console Core: PalConsoleClear ............................................... 13

   3.1.6 PAL Console Core: PalConsolePutChar ........................................... 13

   3.1.7 PAL Console Core: PalConsolePutHex ............................................ 14

   3.1.8 PAL Console Core: PalConsolePutUInt ......................................... 14

   3.1.9 PAL Console Core: PalConsolePutString ...................................... 14

   3.2 **PAL CONSOLE example** ............................................................ 15

4 **PAL MOUSE** .................................................................................... 18

   4.1 **PAL MOUSE API** ...................................................................... 18

   4.1.1 PAL Mouse Core: PalMouseGetMaxXWidthCT .................................. 19

   4.1.2 PAL Mouse Core: PalMouseGetMaxYWidthCT .................................. 19

   4.1.3 PAL Mouse Core: PalMouseGetMaxZWidthCT .................................. 19

   4.1.4 PAL Mouse Core: PalMouseGetMaxButtonCT ................................... 20

   4.1.5 PAL Mouse Core: PalMouseRun ..................................................... 20

   4.1.6 PAL Mouse Core: PalMouseReset ................................................... 20

   4.1.7 PAL Mouse Core: PalMouseEnable ............................................... 21

   4.1.8 PAL Mouse Core: PalMouseDisable ............................................... 21

   4.1.9 PAL Mouse Core: PalMouseSetMaxX ............................................. 21

   4.1.10 PAL Mouse Core: PalMouseSetMaxY .......................................... 22

   4.1.11 PAL Mouse Core: PalMouseSetMaxZ .......................................... 22

   4.1.12 PAL Mouse Core: PalMouseSetXY .............................................. 23

   4.1.13 PAL Mouse Core: PalMouseSetWrap ......................................... 23

   4.1.14 PAL Mouse Core: PalMouseGetX ............................................... 24

   4.1.15 PAL Mouse Core: PalMouseGetY ............................................... 24

   4.1.16 PAL Mouse Core: PalMouseGetZ ............................................... 24

   4.1.17 PAL Mouse Core: PalMouseGetButton ...................................... 25
4.1.18 PAL Mouse Core: PalMouseReadChange

4.2 PAL Mouse example

5 PAL Framebuffer

5.1 PAL Framebuffer8 API
5.1.1 PAL Framebuffer8 Core: PalFrameBuffer8Run
5.1.2 PAL Framebuffer8 Core: PalFrameBuffer8Reset
5.1.3 PAL Framebuffer8 Core: PalFrameBuffer8Enable
5.1.4 PAL Framebuffer8 Core: PalFrameBuffer8Disable
5.1.5 PAL Framebuffer8 Core: PalFrameBuffer8ReadQuad
5.1.6 PAL Framebuffer8 Core: PalFrameBuffer8WriteQuad
5.1.7 PAL Framebuffer8 Core: PalFrameBuffer8Read
5.1.8 PAL Framebuffer8 Core: PalFrameBuffer8Write
5.1.9 PAL Framebuffer8 example

5.2 PAL Framebuffer16 API
5.2.1 PAL FrameBuffer16 Core: PalFrameBuffer16Run
5.2.2 PAL FrameBuffer16 Core: PalFrameBuffer16Reset
5.2.3 PAL FrameBuffer16 Core: PalFrameBuffer16Enable
5.2.4 PAL FrameBuffer16 Core: PalFrameBuffer16Disable
5.2.5 PAL FrameBuffer16 Core: PalFrameBuffer16ReadPair
5.2.6 PAL FrameBuffer16 Core: PalFrameBuffer16WritePair
5.2.7 PAL FrameBuffer16 Core: PalFrameBuffer16Read
5.2.8 PAL FrameBuffer16 Core: PalFrameBuffer16Write
5.2.9 PAL Framebuffer16 example

5.3 PAL FramebufferDB API
5.3.1 PAL FrameBufferDB Core: PalFrameBufferDBRun
5.3.2 PAL FrameBufferDB Core: PalFrameBufferDBReset
5.3.3 PAL FrameBufferDB Core: PalFrameBufferDBEnable
5.3.4 PAL FrameBufferDB Core: PalFrameBufferDBDisable
5.3.5 PAL FrameBufferDB Core: PalFrameBufferDBRead
5.3.6 PAL FrameBufferDB Core: PalFrameBufferDBWrite
5.3.7 PAL FrameBufferDB Core: PalFrameBufferDBSwapBuffers
5.3.8 PAL FrameBufferDB Core: PalFrameBufferDBSetBackground
5.3.9 PAL FramebufferDB example

5.4 PAL Block RAM Framebuffer
5.4.1 Running the Block RAM Framebuffer
5.4.2 Writing to the Block RAM Framebuffer

6 Index
Conventions

The following conventions are used in this document.

**Warning Message.** These messages warn you that actions may damage your hardware.

**Handy Note.** These messages draw your attention to crucial pieces of information.

Hexadecimal numbers will appear throughout this document. The convention used is that of prefixing the number with '0x' in common with standard C syntax.

Sections of code or commands that you must type are given in typewriter font like this:

```c
void main();
```

Information about a type of object you must specify is given in italics like this:

```c
copy SourceFileName DestinationFileName
```

Optional elements are enclosed in square brackets like this:

```c
struct [type_Name]
```

Curly brackets around an element show that it is optional but it may be repeated any number of times.

```c
string ::= "(character)"
```
Assumptions & Omissions

This manual assumes that you:

- have used Handel-C or have the Handel-C Language Reference Manual
- are familiar with common programming terms (e.g. functions)
- are familiar with your operating system (Linux or MS Windows)

This manual does not include:

- instruction in VHDL or Verilog
- instruction in the use of place and route tools
- tutorial example programs. These are provided in the Handel-C User Manual
1 PAL Cores: Introduction

The Platform Abstraction Layer (PAL) is a component of the Platform Developer’s Kit.

PAL is an Application Programming Interface (API) for peripherals. The API offers a standard interface to hardware, enabling you to write portable Handel-C applications that can run on different FPGA/PLD boards without modification.

You can read more about PAL in the PAL User Guide.

PAL is augmented with a set of PAL Cores. These are generic IP Cores that add functionality to the PAL devices. The PAL Cores currently supplied with PAL are:

- PAL Console: a generic video console library.
- PAL Framebuffer: generic libraries for 16 and 8-bit frame buffers, and a 24-bit double-buffered frame buffer.
- PAL Mouse: a generic mouse driver library.
- PAL Keyboard: a generic keyboard driver library.

The PAL Cores API may be subject to change in the future.
2 PAL Keyboard

This PAL Core provides a generic keyboard driver, using the PAL API. The driver uses the PAL DataPort API to communicate with the keyboard. The following functionality is provided to the user:

- Keyboard enable, disable and reset.
- Read ASCII character.
- Read keyboard scan code.

The topics below give further information on the PAL Keyboard core:

- PAL Keyboard API
- PAL Keyboard example (see page 9)

2.1 PAL Keyboard API

The PAL Keyboard API consists of the following macro procedures:

- PalKeyboardRun
- PalKeyboardReset
- PalKeyboardEnable
- PalKeyboardDisable
- PalKeyboardReadASCII
- PalKeyboardReadScanCode

2.1.1 PAL Keyboard Core: PalKeyboardRun

macro proc PalKeyboardRun (KeyboardPtrPtr, DataPortHandleCT, ClockRate);

Arguments

KeyboardPtrPtr  Pointer to a pointer to a PalKeyboard structure.
DataPortHandleCT Constant PAL Handle to a PalDataPort resource.
ClockRate       Clock rate of the clock domain of call to this macro, in Hz.

Timing

Does not terminate in normal use.
Description
Runs the device management tasks for the keyboard. Must always run in parallel with accesses to the device.

2.1.2 PAL Keyboard Core: PalKeyboardDisable

```c
macro proc PalKeyboardDisable (KeyboardPtr);
```

**Arguments**
- `KeyboardPtr` Pointer to a `PalKeyboard` structure.

**Timing**
One or more clock cycles.

**Description**
Disables the keyboard driver.

2.1.3 PAL Keyboard Core: PalKeyboardEnable

```c
macro proc PalKeyboardEnable (KeyboardPtr);
```

**Arguments**
- `KeyboardPtr` Pointer to a `PalKeyboard` structure.

**Timing**
One or more clock cycles.

**Description**
Enables the keyboard driver.

2.1.4 PAL Keyboard Core: PalKeyboardReset

```c
macro proc PalKeyboardReset (KeyboardPtr);
```

**Arguments**
- `KeyboardPtr` Pointer to a `PalKeyboard` structure.

**Timing**
One or more clock cycles.
Description
Resets the keyboard data port to a known state.

2.1.5 PAL Keyboard Core: PalKeyboardReadASCII

macro proc PalKeyboardReadASCII (KeyboardPtr, CharPtr);

Arguments
KeyboardPtr Pointer to a PalKeyboard structure.
CharPtr Pointer to a char to store the ASCII code in.

Timing
One or more cycles.

Description
Returns an ASCII character to the user in the variable pointed to by CharPtr. This macro must be allowed to complete before calling it again, and it must not be called in parallel with PalKeyboardReadScanCode. This macro will block until a valid ASCII character is entered on the keyboard. To be sure of not missing any keyboard input, this macro should be called as frequently as keys are likely to be pressed, which should not be a problem given the slow rate at which people type.

2.1.6 PAL Keyboard Core: PalKeyboardReadScanCode

macro proc PalKeyboardReadScanCode (KeyboardPtr, KeyReleasedPtr, ScanCodePtr);

Arguments
KeyboardPtr Pointer to a PalKeyboard structure.
KeyReleasedPtr Pointer to an unsigned 8 variable to store the scan code in.
ScanCodePtr Pointer to an unsigned 1 variable to store the ASCII code in.

Timing
One or more cycles.

Description
Returns a scan code to the user in the variable pointed to by ScanCodePtr, and the status of the current key press in the variable pointed to by KeyReleasedPtr. This macro must be allowed to complete before calling it again, and it must not be called in parallel with PalKeyboardReadASCII. This macro will block until a scan code is received from the keyboard. To be sure of not missing any keyboard input, this macro should be called as
frequently as keys are likely to be pressed, which should not be a problem given the slow rate at which people type.

2.2 PAL Keyboard example

The first requirement to use the PAL Keyboard core is to include the header file:
```c
#include "pal_keyboard.hch"
```

The library file (pal_keyboard.hcl) must also be included, by adding it on the Linker tab of the Project Settings dialog box.

The code sample below shows how to specify a requirement for PS/2 ports (2 in this case, as the mouse is on port 0 on most Celoxica boards), then run and enable the keyboard driver:
macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;

void main (void)
{
    /*
    * Create a pointer to a PalKeyboard structure
    */
    PalKeyboard *KeyboardPtr;

    /*
    * Specify the number of PS/2 ports we require (2 because on most
    * Celoxica boards port 0 is mouse, and port 1 is keyboard).
    */
    PalPS2PortRequire (2);

    par
    {
        /*
        * Run the keyboard driver, using PalPS2PortCT to select
        * the physical PS/2 port to use.
        */
        PalKeyboardRun (&KeyboardPtr, PalPS2PortCT (1), ClockRate);
    }
    seq
    {
        /*
        * Enable the keyboard, then run the user code.
        */
        PalKeyboardEnable (KeyboardPtr);
        UserCode (KeyboardPtr);
    }
}

macro proc UserCode (KeyboardPtr)
{
    unsigned Char;
    PalKeyboardReadASCII (KeyboardPtr, &Char);

    etc ...
}
3 PAL Console

This PAL Core provides a generic console driver, using the PAL API. The driver uses the PAL VideoOut API to communicate with the video display. The following functionality is provided to the user:

- Console enable, disable and reset.
- Clear the console.
- Write a character to the console.
- Write an unsigned hexadecimal number to the console.
- Write an unsigned decimal number to the console.
- Write a string to the console.

The topics below give further information on the PAL Console core:

- PAL Console API
- PAL Console example (see page 15)

3.1 PAL Console API

The PAL Console API consists of the following macro procedures and functions:

- PalConsoleRun
- PalConsoleReset
- PalConsoleEnable
- PalConsoleDisable
- PalConsoleClear
- PalConsolePutString
- PalConsolePutHex

Note that some of these macros and functions have high latencies, especially when clearing the console and writing strings. This must be taken account of in your design.

3.1.1 PAL Console Core: PalConsoleRun

macro proc PalConsoleRun (ConsolePtrPtr, Font, VideoOutHandleCT, ClockRate);

Arguments

ConsolePtrPtr Pointer to a pointer to a PalConsole structure.
Select from the list of built-in fonts.

Constant PAL Handle to a `PalVideoOut` resource.

Clock rate of the clock domain of call to this macro, in Hz.

**Timing**

Does not terminate in normal use.

**Description**

Runs the device management tasks for the console. Must always run in parallel with accesses to the device.

### 3.1.2 PAL Console Core: `PalConsoleDisable`

```cpp
macro proc PalConsoleDisable (ConsolePtr);
```

**Arguments**

- `ConsolePtr`: Pointer to a `PalConsole` structure.

**Timing**

One or more clock cycles

**Description**

Disables the PAL Console video output.

### 3.1.3 PAL Console Core: `PalConsoleEnable`

```cpp
macro proc PalConsoleEnable (ConsolePtr);
```

**Arguments**

- `ConsolePtr`: Pointer to a `PalConsole` structure.

**Timing**

One or more clock cycles

**Description**

Enables the PAL Console video output.
3.1.4 PAL Console Core: PalConsoleReset

macro proc PalConsoleReset (ConsolePtr);

Arguments
ConsolePtr Pointer to a PalConsole structure.

Timing
One or more clock cycles

Description
Resets the PAL Console video output to a known state.

3.1.5 PAL Console Core: PalConsoleClear

macro proc PalConsoleClear (ConsolePtr);

Arguments
ConsolePtr Pointer to a PalConsole structure.

Timing
More than one clock cycle.

Description
Clears the PAL Console output screen. Should not be called in parallel with other macros or functions which write to the console.

3.1.6 PAL Console Core: PalConsolePutChar

macro proc PalConsolePutChar (ConsolePtr, Char);

Arguments
ConsolePtr Pointer to a PalConsole structure.
Char Character to print to the console.

Timing
One or more clock cycles.
Description
Prints a single character to the console. Should not be called in parallel with other macros or functions which write to the console.

3.1.7 PAL Console Core: PalConsolePutHex

void PalConsolePutHex (PalConsole *ConsolePtr, unsigned 32 Value);

Arguments
ConsolePtr Pointer to a PalConsole structure.
Value unsigned 32-bit value to print to the console in hexadecimal format.

Timing
More than one clock cycle.

Description
Prints an unsigned 32-bit value in hexadecimal format to the console. Should not be called in parallel with other macros or functions which write to the console.

3.1.8 PAL Console Core: PalConsolePutUInt

void PalConsolePutUInt (PalConsole *ConsolePtr, unsigned 32 Value);

Arguments
ConsolePtr Pointer to a PalConsole structure.
Value unsigned 32-bit integer to print to the console in decimal format.

Timing
More than one clock cycle.

Description
Prints an unsigned 32-bit value in decimal format to the console. Should not be called in parallel with other macros or functions which write to the console.

3.1.9 PAL Console Core: PalConsolePutString

macro proc PalConsolePutString (ConsolePtr, String);
Arguments

ConsolePtr        Pointer to a PalConsole structure.
String            A string (array of chars) to print to the console.

Timing

One or more clock cycles.

Description

Prints a string of characters to the console. The string would normally be stored in a RAM of char elements. The macro should not be called in parallel with other macros or functions which write to the console.

3.2 PAL Console example

The first requirement to use the PAL Console core is to include the header file:

```
#include "pal_console.hch"
```

The library file (pal_console.hcl) must also be included, by adding it on the Linker tab of the Project Settings dialog box.

The code sample below shows how to specify a requirement for a video output, run and enable the Console driver, then write something to it.
macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;

void main (void)
{
    /*
    * Create a pointer to a PalConsole structure
    */
    PalConsole *ConsolePtr;

    /*
    * Specify the number of video outputs we require
    */
    PalVideoOutRequire (1);

    par
    {
        /*
        * Run the Console driver, using PalVideoOutOptimalCT to
        * select the optimal video output mode for the clock rate.
        */
        PalConsoleRun (&ConsolePtr, PAL_CONSOLE_FONT_NORMAL,
                        PalVideoOutOptimalCT (ClockRate), ClockRate);

    seq
    {
        /*
        * Enable the Console, then print some stuff to it.
        */
        PalConsoleEnable (ConsolePtr);
        UserCode(ConsolePtr);
    }
}

macro proc UserCode(ConsolePtr)
{
    static ram unsigned char String[32] = "Hello World\n";
    static unsigned Number = 0;

    /*
    * Clear the screen
    */
    PalConsoleClear (ConsolePtr);

    /*
* Print a string
*/
PalConsolePutString (ConsolePtr, String);
do{
    PalConsolePutHex  (ConsolePtr, Number);
    PalConsolePutChar (ConsolePtr, ' ');
    PalConsolePutUInt (ConsolePtr, Number);
    PalConsolePutChar (ConsolePtr, '\n');
    Number++;
}while (Number != 20);

etc ...
4 PAL Mouse

This PAL Core provides a generic mouse driver, using the PAL API. The driver uses the PAL DataPort API to communicate with the mouse. The following functionality is provided to the user:

- Mouse enable, disable and reset.
- Set maximum range for mouse co-ordinates.
- Set new mouse co-ordinates.
- Turn wrapping on/off.
- Read current mouse position.
- Read current mouse button state.
- Read mouse position change.

The topics below give further information on the PAL Mouse core:

- PAL Mouse API
- *PAL Mouse example* (see page 25)

4.1 PAL Mouse API

The PAL Mouse API consists of the following macro procedures:

- PalMouseGetMaxXWidthCT
- PalMouseGetMaxYWidthCT
- PalMouseGetMaxZWidthCT
- PalMouseGetMaxButtonCT
- PalMouseRun
- PalMouseReset
- PalMouseEnable
- PalMouseDisable
- PalMouseSetMaxX
- PalMouseSetMaxY
- PalMouseSetMaxZ
- PalMouseSetXY
- PalMouseSetWrap
- PalMouseGetX
- PalMouseGetY
- PalMouseGetZ
• PalMouseGetButton
• PalMouseReadChange

4.1.1 PAL Mouse Core: PalMouseGetMaxXWidthCT

macro expr PalMouseGetMaxXWidthCT ();

Arguments
none

Timing
None - this is a compile-time expression.

Description
Returns the maximum bit width of the X coordinate.

4.1.2 PAL Mouse Core: PalMouseGetMaxYWidthCT

macro expr PalMouseGetMaxYWidthCT ();

Arguments
none

Timing
None - this is a compile-time expression.

Description
Returns the maximum bit width of the Y coordinate.

4.1.3 PAL Mouse Core: PalMouseGetMaxZWidthCT

macro expr PalMouseGetMaxZWidthCT ();

Arguments
none

Timing
None - this is a compile-time expression.
Description
Returns the maximum bit width of the Z coordinate.

4.1.4 PAL Mouse Core: PalMouseGetMaxButtonCT

macro expr PalMouseGetMaxButtonCT ();

Arguments
none

Timing
None - this is a compile-time expression.

Description
Returns the maximum number of mouse buttons.

4.1.5 PAL Mouse Core: PalMouseRun

macro proc PalMouseRun (MousePtrPtr, DataPortHandleCT, ClockRate);

Arguments
MousePtrPtr Pointer to a pointer to a PalMouse structure.
DataPortHandleCT Constant PAL Handle to a PalDataPort resource.
ClockRate Clock rate of the clock domain of call to this macro, in Hz.

Timing
Does not terminate in normal use.

Description
Runs the device management tasks for the mouse. Must always run in parallel with accesses to the device.

4.1.6 PAL Mouse Core: PalMouseReset

macro proc PalMouseReset (MousePtr);

Arguments
MousePtr Pointer to a PalMouse structure.
Timing
One or more clock cycles.

Description
Resets the mouse data port to a known state.

4.1.7 PAL Mouse Core: PalMouseEnable

macro proc PalMouseEnable (MousePtr);

Arguments
MousePtr Pointer to a PalMouse structure.

Timing
Zero or more clock cycles.

Description
Enables the mouse driver.

4.1.8 PAL Mouse Core: PalMouseDisable

macro proc PalMouseDisable (MousePtr);

Arguments
MousePtr Pointer to a PalMouse structure.

Timing
Zero or more clock cycles.

Description
Disables the mouse driver.

4.1.9 PAL Mouse Core: PalMouseSetMaxX

macro proc PalMouseSetMaxX (MousePtr, MaxX);

Arguments
MousePtr Pointer to a PalMouse structure.
MaxX Unsigned integer, width equal to PalMouseGetMaxXWidthCT().
Timing
One clock cycle.

Description
Sets the maximum X coordinate that is to be returned by PalMouseGetX(). If wrapping is off, the pointer position will either be clipped at this maximum (and similarly at zero). If wrapping is on, the pointer position will be wrapped around (so passing through the maximum point brings it back to zero, and vice versa).

By default MaxX is set to zero. This means the pointer will not be clipped and will range over 0 to \((2^{\text{PalMouseGetMaxXWidthCT()}}) - 1\) with wrapping.

4.1.10 PAL Mouse Core: PalMouseSetMaxY

PalMouseSetMaxY (MousePtr, MaxY);

Arguments

MousePtr  Pointer to a PalMouse structure.
MaxY    Unsigned integer, width equal to PalMouseGetMaxYWidthCT().

Timing
One clock cycle.

Description
Sets the maximum Y coordinate that is to be returned by PalMouseGetY(). If wrapping is off, the pointer position will either be clipped at this maximum (and similarly at zero). If wrapping is on, the pointer position will be wrapped around (so passing through the maximum point brings it back to zero, and vice versa).

By default MaxY is set to zero. This means the pointer will not be clipped and will range over 0 to \((2^{\text{PalMouseGetMaxYWidthCT()}}) - 1\) with wrapping.

4.1.11 PAL Mouse Core: PalMouseSetMaxZ

macro proc PalMouseSetMaxZ (MousePtr, MaxZ);

Arguments

MousePtr  Pointer to a PalMouse structure.
MaxZ    Unsigned integer, width equal to PalMouseGetMaxZWidthCT().

Timing
One clock cycle.
Description
Sets the maximum Z coordinate that is to be returned by PalMouseGetZ(). If wrapping is off, the pointer position will either be clipped at this maximum (and similarly at zero). If wrapping is on, the pointer position will be wrapped around (so passing through the maximum point brings it back to zero, and vice versa).

By default MaxZ is set to zero. This means the pointer will not be clipped and will range over $0$ to $((2 ^ (\text{PalMouseGetMaxZWidthCT}()))) - 1$ with wrapping.

4.1.12 PAL Mouse Core: PalMouseSetXY

macro proc PalMouseSetXY (MousePtr, NewX, NewY);

Arguments
MousePtr Pointer to a PalMouse structure.
NewX Unsigned integer, width equal to PalMouseGetMaxXWidthCT().
NewY Unsigned integer, width equal to PalMouseGetMaxYWidthCT().

Timing
One or more clock cycles.

Description
Sets a new X/Y co-ordinate for the mouse.

4.1.13 PAL Mouse Core: PalMouseSetWrap

macro proc PalMouseSetWrap (MousePtr, Wrap);

Arguments
MousePtr Pointer to a PalMouse structure.
Wrap unsigned 1 (boolean)

Timing
One clock cycle.

Description
Turns wrapping on or off according to the value of Wrap (1=on). Pointer wrapping is off by default, which will cause the pointer to be limited to the region set by PalMouseSetMaxX(), PalMouseSetMaxY() and PalMouseSetMaxZ().
4.1.14 PAL Mouse Core: PalMouseGetX

macro expr PalMouseGetX (MousePtr);

Arguments
MousePtr Pointer to a PalMouse structure.

Timing
None - this is a macro expression.

Description
Returns the current X coordinate of the mouse pointer, of type unsigned and width PalMouseGetMaxXWidthCT().

4.1.15 PAL Mouse Core: PalMouseGetY

macro expr PalMouseGetY (MousePtr);

Arguments
MousePtr Pointer to a PalMouse structure.

Timing
None - this is a macro expression.

Description
Returns the current Y coordinate of the mouse pointer, of type unsigned and width PalMouseGetMaxYWidthCT().

4.1.16 PAL Mouse Core: PalMouseGetZ

macro expr PalMouseGetZ (MousePtr);

Arguments
MousePtr Pointer to a PalMouse structure.

Timing
None - this is a macro expression.

Description
Returns the current Z coordinate of the mouse pointer, of type unsigned and width PalMouseGetMaxZWidthCT().
4.1.17 PAL Mouse Core: PalMouseGetButton

macro expr PalMouseGetButton (MousePtr, Index);

Arguments
MousePtr Pointer to a PalMouse structure.
Index Button index, type unsigned log2ceil (PalMouseGetMaxButtonCT())

Timing
None - this is a macro expression.

Description
Returns the state of button Index as an unsigned 1 where a value of 1 is down and 0 is up. The buttons are typically numbered sequentially, left=0, middle=1, right=2.

4.1.18 PAL Mouse Core: PalMouseReadChange

macro proc PalMouseReadChange (MousePtr, DeltaXPtr, DeltaYPtr, DeltaZPtr, ButtonMaskPtr);

Arguments
MousePtr Pointer to a PalMouse structure.
DeltaXPtr Pointer to an signed 9-bit integer
DeltaYPtr Pointer to an signed 9-bit integer
DeltaZPtr Pointer to an signed 4-bit integer
ButtonMaskPtr Pointer to an unsigned integer of width PalMouseGetMaxButtonCT().

Timing
One or more clock cycles.

Description
This macro blocks for an indefinite length of time until the mouse is moved, and then assigns delta change values to the DeltaX, DeltaY and DeltaZ parameters, and the current state of the mouse buttons to the ButtonMask parameter.

4.2 PAL Mouse example

The first requirement to use the PAL Mouse core is to include the header file:
#include "pal_mouse.hch"

The library file (pal_mouse.hcl) must also be included, by adding it on the Linker tab of the Project Settings dialog box.

The code sample below shows how to specify a requirement for a PS/2 port, then run and enable the mouse driver:
macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;

void main (void)
{
    /*
    * Create a pointer to a PalMouse structure
    */
    PalMouse *MousePtr;

    /*
    * Specify the number of PS/2 ports we require
    */
    PalPS2PortRequire (1);

    par
    {
        /*
        * Run the mouse driver, using PalPS2PortCT to select
        * the physical PS/2 port to use.
        */
        PalMouseRun (&MousePtr, PalPS2PortCT (0), ClockRate);

        seq
        {
            /*
            * Enable the mouse, then run the user code.
            */
            PalMouseEnable (MousePtr);
           UserCode (MousePtr);
        }
    }
}

macro procUserCode (MousePtr)
{
    unsigned MouseX, MouseY;
    /*
    * Setup mouse
    */
    par
    {
        PalMouseSetMaxX (MousePtr, 640);
        PalMouseSetMaxY (MousePtr, 480);
        PalMouseSetMaxZ (MousePtr, 31);
        PalMouseSetWrap (MousePtr, 0);
    }
}
}  

MouseX = PalMouseGetX (MousePtr);
MouseY = PalMouseGetY (MousePtr);

etc ...
5 PAL Framebuffer

Four types of PAL Framebuffer are available:

Framebuffer8
Framebuffer8 uses a single PAL PL1 RAM, storing four pixels per memory word and using 8 bits per pixel. Each pixel is represented in RGB format using 3 bits for red and green, and 2 bits for blue. Because there are multiple pixels per memory word, the display process does not need to access the RAM every clock cycle, which leaves spare cycles for you to modify the Framebuffer, and allows the use of a single PL1 RAM.

The Framebuffer can be updated at the same rate as the display providing the PalFrameBuffer8WriteQuad macro is used. Using PalFrameBuffer8Write will result in a much lower performance.

See these topics for more information:
PAL Framebuffer8 API (see page 30)
PAL Framebuffer8 example (see page 34)

Framebuffer16
Framebuffer16 uses a single PAL PL1 RAM, storing two pixels per memory word and using 16 bits per pixel. Each pixel is represented in RGB format using 5 bits for red and blue, and 6 bits for green. Because there are multiple pixels per memory word, the display process does not need to access the RAM every clock cycle, which leaves spare cycles for you to modify the Framebuffer, and allows the use of a single PL1 RAM.

The Framebuffer can be updated at the same rate as the display providing the PalFrameBuffer16WritePair macro is used. Using PalFrameBuffer16Write will result in a much lower performance.

See these topics for more information:
PAL Framebuffer16 API (see page 36)
PAL Framebuffer16 example (see page 40)

FramebufferDB
FramebufferDB is a double-buffered Framebuffer, and requires two PAL PL1 RAMs. At any instant one of the buffers is being displayed, and the other is available for you to access. Once you have prepared the next frame to be displayed, you instruct FramebufferDB to swap the buffers over. FramebufferDB stores the pixels in 24-bit RGB format, using one memory word per pixel.
By using two PL1 RAMs, the FramebufferDB can use 24-bit colour while allowing you fast enough access to the buffers to update the screen at the refresh rate.

See these topics for more information:

**PAL FrameBufferDB API** (see page 42)

**PAL FramebufferDB example** (see page 46)

**Block RAM Framebuffer**

The PAL Block RAM Framebuffer operates in a similar way to the other Framebuffers, but uses on-chip Block RAM rather than off-chip memory.

**PAL Block RAM Framebuffer** (see page 48)

### 5.1 PAL Framebuffer8 API

The API for PAL FrameBuffer8 includes the following macros:

- PalFrameBuffer8Run
- PalFrameBuffer8Reset
- PalFrameBuffer8Enable
- PalFrameBuffer8Disable
- PalFrameBuffer8ReadQuad
- PalFrameBuffer8WriteQuad
- PalFrameBuffer8Read
- PalFrameBuffer8Write

#### 5.1.1 PAL Framebuffer8 Core: PalFrameBuffer8Run

```
macro proc PalFrameBuffer8Run (FrameBuffer8PtrPtr, PL1RAMHandleCT, VideoOutHandleCT, ClockRate);
```

**Arguments**

- **FrameBuffer8PtrPtr**: Pointer to a pointer to a `PalFrameBuffer8` structure.
- **PL1RAMHandleCT**: Constant PAL Handle for a PAL PL1RAM.
- **VideoOutHandleCT**: Constant PAL Handle for a `PalVideoOut` resource.
- **ClockRate**: Clock rate of the clock domain of call to this macro, in Hz.
Timing
Does not terminate in normal use.

Description
Runs the device management tasks for the Framebuffer. Must always run in parallel with accesses to the Framebuffer. All accesses to the Framebuffer are in 24-bit RGB format, but the Framebuffer stores pixels internally in 8-bit format, using three bits for red and green, and two bits for blue.

5.1.2 PAL Framebuffer8 Core: PalFrameBuffer8Reset

macro proc PalFrameBuffer8Reset (FrameBuffer8Ptr);

Arguments
FrameBuffer8Ptr Pointer to a PalFrameBuffer8 structure.

Timing
One or more clock cycles.

Description
Resets the Framebuffer, which includes resetting the VideoOut and the PL1RAM.

5.1.3 PAL Framebuffer8 Core: PalFrameBuffer8Enable

macro proc PalFrameBuffer8Enable (FrameBuffer8Ptr);

Arguments
FrameBuffer8Ptr Pointer to a PalFrameBuffer8 structure.

Timing
One or more clock cycles.

Description
Enables the Framebuffer, which includes enabling the VideoOut and the PL1RAM.

5.1.4 PAL Framebuffer8 Core: PalFrameBuffer8Disable

macro proc PalFrameBuffer8Disable (FrameBuffer8Ptr);
Arguments
FrameBuffer8Ptr Pointer to a PalFrameBuffer8 structure.

Timing
One or more clock cycles.

Description
Disables the Framebuffer, which includes disabling the VideoOut and the PL1RAM.

5.1.5 PAL Framebuffer8 Core: PalFrameBuffer8ReadQuad

macro proc PalFrameBuffer8ReadQuad (FrameBuffer8Ptr, X, Y, PixelQuadPtr);

Arguments
FrameBuffer8Ptr Pointer to a PalFrameBuffer8 structure.
X X coordinate. Unsigned integer of width log2ceil(PalVideoOutGetVisibleXCT())
Y Y coordinate. Unsigned integer of width log2ceil(PalVideoOutGetVisibleYCT())
PixelQuadPtr Pointer to an unsigned 96 variable to receive the four pixels.

Timing
One or more clock cycles.

Description
Reads four adjacent pixels from a row in the Framebuffer. The location to be read is specified by the X and Y parameters, but the two least significant bits of X will be ignored, as the pixels read from the Framebuffer must be aligned to a four-pixel boundary.

The four pixels returned via PixelQuadPtr are in 24-bit RGB format, but the Framebuffer stores them internally in 8-bit format, using three bits for red and green, and two bits for blue.

5.1.6 PAL Framebuffer8 Core: PalFrameBuffer8WriteQuad

macro proc PalFrameBuffer8WriteQuad (FrameBuffer8Ptr, X, Y, Pixel0, Pixel1, Pixel2, Pixel3);

Arguments
FrameBuffer8Ptr Pointer to a PalFrameBuffer8 structure.
X coordinate. Unsigned integer of width \(\log_2\lceil\text{PalVideoOutGetVisibleXCT()}\rceil\)

Y coordinate. Unsigned integer of width \(\log_2\lceil\text{PalVideoOutGetVisibleYCT()}\rceil\)

Pixel0 First pixel to write. Must be unsigned 24 in RGB format.

Pixel1 Second pixel to write. Must be unsigned 24 in RGB format.

Pixel2 Third pixel to write. Must be unsigned 24 in RGB format.

Pixel3 Fourth pixel to write. Must be unsigned 24 in RGB format.

Timing
One or more clock cycles.

Description
Writes four adjacent pixels to a row in the Framebuffer. The location to be written is specified by the X and Y parameters, but the two least significant bits of X will be ignored, as the pixels written to the Framebuffer must be aligned to a four-pixel boundary.

The four pixels passed to \texttt{PalFrameBuffer8WriteQuad} must be in 24-bit RGB format, but the Framebuffer stores them internally in 8-bit format, using three bits for red and green, and two bits for blue.

5.1.7 PAL Framebuffer8 Core: \texttt{PalFrameBuffer8Read}

macro proc PalFrameBuffer8Read (FrameBuffer8Ptr, X, Y, PixelPtr);

Arguments

FrameBuffer8Ptr Pointer to a \texttt{PalFrameBuffer8} structure.

X X coordinate. Unsigned integer of width \(\log_2\lceil\text{PalVideoOutGetVisibleXCT()}\rceil\)

Y Y coordinate. Unsigned integer of width \(\log_2\lceil\text{PalVideoOutGetVisibleYCT()}\rceil\)

PixelPtr Pointer to an unsigned 24 variable to receive the pixel.

Timing
One or more clock cycles.
Description
Reads a single pixel from the Framebuffer, from the location specified by \( x \) and \( y \). The pixel returned via \( \text{PixelPtr} \) is in 24-bit RGB format, but the Framebuffer stores pixels internally in 8-bit format, using three bits for red and green, and two bits for blue.

5.1.8 PAL Framebuffer8 Core: PalFrameBuffer8Write

macro proc PalFrameBuffer8Write (FrameBuffer8Ptr, X, Y, Pixel);

Arguments
FrameBuffer8Ptr: Pointer to a PalFrameBuffer8 structure.
X: \( x \) coordinate. Unsigned integer of width \( \log_2\text{ceil}(\text{PalVideoOutGetVisibleXCT}()) \)
Y: \( y \) coordinate. Unsigned integer of width \( \log_2\text{ceil}(\text{PalVideoOutGetVisibleYCT}()) \)
PixelPtr: Pixel to write to Framebuffer. Must be unsigned 24 in RGB format.

Timing
One or more clock cycles.

Description
Writes a single pixel to the Framebuffer, at the location specified by \( x \) and \( y \). The Framebuffer stores four pixels per memory word, so this macro performs a read-modify-write, so that only the specified pixel is overwritten.

The pixel passed to \( \text{PalFrameBuffer8Write} \) must be in 24-bit RGB format, but the Framebuffer stores pixels internally in 8-bit format, using three bits for red and green, and two bits for blue.

5.1.9 PAL Framebuffer8 example

The first requirement to use the PAL Framebuffer8 core is to include the header file:
#include "pal_framebuffer8.hch"

The library file (\texttt{pal_framebuffer8.hcl}) must also be included, by adding it on the Linker tab of the Project Settings dialog box.

The code sample below shows how to specify a requirement for a video output and a PL1 RAM, then run and enable the Framebuffer8 driver:
macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;

void main (void)
{
    /*
    * Create a pointer to a PalFrameBuffer8 structure
    */
    PalFrameBuffer8 *FBPtr;

    /*
    * Specify the number of video outputs we require
    */
    PalVideoOutRequire (1);
    PalPL1RAMRequire   (1);

    par
    {
        /*
        * Run the Framebuffer8 driver, using PalVideoOutOptimalCT to
        * select the optimal video output mode for the clock rate,
        * and PalPL1RAMCT to select the PL1RAM to use.
        */
        PalFrameBuffer8Run (&FBPtr, PalPL1RAMCT (0),
            PalVideoOutOptimalCT (ClockRate), ClockRate);

    seq
    {
        /*
        * Enable the Framebuffer8, then run the user code
        */
        PalFrameBuffer8Enable(FBPtr);
        UserCode (FBPtr);
    }
    }
}

macro proc UserCode (FBPtr)
{
    static unsigned (PalVideoOutGetMaxXWidthCT ()) X = 0;
    static unsigned (PalVideoOutGetMaxYWidthCT ()) Y = 0;

    do
    {
        do
        {
            par
5.2 PAL Framebuffer16 API

The API for PAL FrameBuffer16 includes the following macros:

- PalFrameBuffer16Run
- PalFrameBuffer16Reset
- PalFrameBuffer16Enable
- PalFrameBuffer16Disable
- PalFrameBuffer16ReadPair
- PalFrameBuffer16WritePair
- PalFrameBuffer16Read
- PalFrameBuffer16Write

5.2.1 PAL FrameBuffer16 Core: PalFrameBuffer16Run

Macro proc PalFrameBuffer16Run (FrameBuffer16PtrPtr, PL1RAMHandleCT, VideoOutHandleCT, ClockRate);

Arguments

FrameBuffer16PtrPtr Pointer to a pointer to a PalFrameBuffer16 structure.
PL1RAMHandleCT Constant PAL Handle for a PAL PL1RAM.
VideoOutHandleCT Constant PAL Handle for a PalVideoOut resource.
ClockRate Clock rate of the clock domain of call to this macro, in Hz.

Timing

Does not terminate in normal use.
Description
Runs the device management tasks for the Framebuffer. Must always run in parallel with accesses to the Framebuffer. All accesses to the Framebuffer are in 24-bit RGB format, but the Framebuffer stores pixels internally in 16-bit format, using five bits for red and blue, and six bits for green.

5.2.2 PAL FrameBuffer16 Core: PalFrameBuffer16Reset

macro proc PalFrameBuffer16Reset (FrameBuffer16Ptr);

Arguments
FrameBuffer16Ptr       Pointer to a PalFrameBuffer16 structure.

Timing
One or more clock cycles.

Description
Resets the Framebuffer, which includes resetting the VideoOut and the PL1RAM.

5.2.3 PAL FrameBuffer16 Core: PalFrameBuffer16Enable

macro proc PalFrameBuffer16Enable (FrameBuffer16Ptr);

Arguments
FrameBuffer16Ptr       Pointer to a PalFrameBuffer16 structure.

Timing
One or more clock cycles.

Description
Enables the Framebuffer, which includes enabling the VideoOut and the PL1RAM.

5.2.4 PAL FrameBuffer16 Core: PalFrameBuffer16Disable

macro proc PalFrameBuffer16Disable (FrameBuffer16Ptr);

Arguments
FrameBuffer16Ptr       Pointer to a PalFrameBuffer16 structure.
Timing
One or more clock cycles.

Description
Disables the Framebuffer, which includes disabling the VideoOut and the PL1RAM.

5.2.5 PAL FrameBuffer16 Core: PalFrameBuffer16ReadPair

```c
macro proc PalFrameBuffer8ReadPair (FrameBuffer8Ptr, X, Y, PixelPairPtr);
```

**Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrameBuffer16Ptr</td>
<td>Pointer to a PalFrameBuffer16 structure.</td>
</tr>
<tr>
<td>X</td>
<td>X coordinate. Unsigned integer of width log2ceil(PalVideoOutGetVisibleXCT())</td>
</tr>
<tr>
<td>Y</td>
<td>Y coordinate. Unsigned integer of width log2ceil(PalVideoOutGetVisibleYCT())</td>
</tr>
<tr>
<td>PixelPairPtr</td>
<td>Pointer to an unsigned 48 variable to receive the pair of pixels.</td>
</tr>
</tbody>
</table>

**Timing**
One or more clock cycles.

**Description**
Reads two adjacent pixels from a row in the Framebuffer. The location to be read is specified by the X and Y parameters, but the least significant bit of X will be ignored, as the pixels read from the Framebuffer must be aligned to a two-pixel boundary.

The pair of pixels returned via PixelPairPtr are in 24-bit RGB format, but the Framebuffer stores them internally in 16-bit format, using five bits for red and blue, and six bits for green.

5.2.6 PAL FrameBuffer16 Core: PalFrameBuffer16WritePair

```c
macro proc PalFrameBuffer16WritePair (FrameBuffer16Ptr, X, Y, LeftPixel, RightPixel);
```

**Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrameBuffer16Ptr</td>
<td>Pointer to a PalFrameBuffer16 structure.</td>
</tr>
</tbody>
</table>
X  
X coordinate. Unsigned integer of width
    $\log_2\lceil\text{PalVideoOutGetVisibleXCT()}\rceil$

Y  
Y coordinate. Unsigned integer of width
    $\log_2\lceil\text{PalVideoOutGetVisibleYCT()}\rceil$

LeftPixel  
First pixel to write. Must be unsigned 24 in RGB format.

RightPixel  
Second pixel to write. Must be unsigned 24 in RGB format.

**Timing**

One or more clock cycles.

**Description**

Writes two adjacent pixels to a row in the Framebuffer. The location to be written is specified by the X and Y parameters, but the least significant bit of X will be ignored, as the pixels written to the Framebuffer must be aligned to a two-pixel boundary.

The pair of pixels passed to `PalFrameBuffer16WritePair` must be in 24-bit RGB format, but the Framebuffer stores them internally in 16-bit format, using five bits for red and blue, and six bits for green.

### 5.2.7 PAL FrameBuffer16 Core: PalFrameBuffer16Read

macro proc PalFrameBuffer16Read (FrameBuffer16Ptr, X, Y, PixelPtr);

**Arguments**

FrameBuffer16Ptr  
Pointer to a `PalFrameBuffer16` structure.

X  
X coordinate. Unsigned integer of width
    $\log_2\lceil\text{PalVideoOutGetVisibleXCT()}\rceil$

Y  
Y coordinate. Unsigned integer of width
    $\log_2\lceil\text{PalVideoOutGetVisibleYCT()}\rceil$

PixelPtr  
Pointer to an unsigned 24 variable to receive the pixel.

**Timing**

One or more clock cycles.

**Description**

Reads a single pixel from the Framebuffer, from the location specified by X and Y. The pixel returned via `PixelPtr` is in 24-bit RGB format, but the Framebuffer stores pixels internally in 16-bit format, using five bits for red and blue, and six bits for green.
5.2.8 PAL FrameBuffer16 Core: PalFrameBuffer16Write

macro proc PalFrameBuffer16Write (FrameBuffer16Ptr, X, Y, Pixel);

**Arguments**

- **FrameBuffer16Ptr**  
  Pointer to a PalFrameBuffer16 structure.

- **X**  
  X coordinate. Unsigned integer of width log2ceil(PalVideoOutGetVisibleXCT())

- **Y**  
  Y coordinate. Unsigned integer of width log2ceil(PalVideoOutGetVisibleYCT())

- **PixelPtr**  
  Pixel to write to Framebuffer. Must be unsigned 24 in RGB format.

**Timing**

One or more clock cycles.

**Description**

Writes a single pixel to the Framebuffer, at the location specified by X and Y. The Framebuffer stores two pixels per memory word, so this macro performs a read-modify-write, so that only the specified pixel is overwritten.

The pixel passed to PalFrameBuffer16Write must be in 24-bit RGB format, but the Framebuffer stores pixels internally in 16-bit format, using five bits for red and blue, and six bits for green.

5.2.9 PAL Framebuffer16 example

The first requirement to use the PAL Framebuffer16 core is to include the header file:

```c
#include "pal_framebuffer16.hch"
```

The library file (pal_framebuffer16.hcl) must also be included, by adding it on the Linker tab of the Project Settings dialog box.

The code sample below shows how to specify a requirement for a video output and a PL1 RAM, then run and enable the Framebuffer16 driver:
macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;

void main (void)
{
    /*
    * Create a pointer to a PalFrameBuffer16 structure
    */
    PalFrameBuffer16 *FBPtr;
    /*
    * Specify the number of video outputs we require
    */
    PalVideoOutRequire (1);
    PalPL1RAMRequire (1);

    par
    {
        /*
        * Run the Framebuffer16 driver, using PalVideoOutOptimalCT to
        * select the optimal video output mode for the clock rate,
        * and PalPL1RAMCT to select the PL1RAM to use.
        */
        PalFrameBuffer16Run (&FBPtr, PalPL1RAMCT (0),
                             PalVideoOutOptimalCT (ClockRate), ClockRate);
    }
    seq
    {
        /*
        * Enable the Framebuffer16, then run the user code
        */
        PalFrameBuffer16Enable(FBPtr);
        UserCode (FBPtr);
    }
}

macro proc UserCode (FBPtr)
{
    static unsigned (PalVideoOutGetXWidthCT ()) X = 0;
    static unsigned (PalVideoOutGetYWidthCT ()) Y = 0;

    do
    {
        do
        {
            par
            {
            
            }
        }
    }
}
{  
  /*  
  * Colour the whole frame in green  
  */  
  PalFrameBuffer16Write (FBPtr, X, Y, 0x00FF00);  
  X++;  
  }  
  while (X != 0);  
  Y++;  
  }  
  while (Y != 0);  
}  

5.3 PAL FramebufferDB API

The API for PAL FrameBufferDB includes the following macros:

- PalFrameBufferDBRun
- PalFrameBufferDBReset
- PalFrameBufferDBEnable
- PalFrameBufferDBDisable
- PalFrameBufferDBRead
- PalFrameBufferDBWrite
- PalFrameBufferDBSwapBuffers
- PalFrameBufferDBSetBackground

5.3.1 PAL FrameBufferDB Core: PalFrameBufferDBRun

macro proc PalFrameBufferDBRun (FrameBufferDBPtrPtr, PL1RAMHandleCT0,  
PL1RAMHandleCT1, VideoOutHandleCT, ClockRate);

Arguments

FrameBufferDBPtrPtr  Pointer to a pointer to a PalFrameBufferDB structure.
PL1RAMHandleCT0  Constant PAL Handle for a PAL PL1RAM.
PL1RAMHandleCT1  Constant PAL Handle for a PAL PL1RAM.
VideoOutHandleCT  Constant PAL Handle for a PalVideoOut resource.
ClockRate  Clock rate of the clock domain of call to this macro, in Hz.
Timing
Does not terminate in normal use.

Description
Runs the device management tasks for the Framebuffer. Must always run in parallel with accesses to the Framebuffer. All accesses to the Framebuffer are in 24-bit RGB format, and the Framebuffer stores pixels internally in this format, using one memory word per pixel.

5.3.2 PAL FrameBufferDB Core: PalFrameBufferDBReset

macro proc PalFrameBufferDBReset (FrameBufferDBPtr);

Arguments
FrameBufferDBPtr Pointer to a PalFrameBufferDB structure.

Timing
One or more clock cycles.

Description
Resets the Framebuffer, which includes resetting the VideoOut and the PL1RAMs.

5.3.3 PAL FrameBufferDB Core: PalFrameBufferDBEnable

macro proc PalFrameBufferDBEnable (FrameBufferDBPtr);

Arguments
FrameBufferDBPtr Pointer to a PalFrameBufferDB structure.

Timing
More than one clock cycle.

Description
Enables the Framebuffer, which includes enabling the VideoOut and the PL1RAMs. This macro may take a significant number of cycles to execute, as it will wait for PalFrameBufferDBRun to complete its start-up procedures.
5.3.4 PAL FrameBufferDB Core: PalFrameBufferDBDisable

macro proc PalFrameBufferDBDisable (FrameBufferDBPtr);

Arguments
FrameBufferDBPtr  Pointer to a PalFrameBufferDB structure.

Timing
One or more clock cycles.

Description
Disables the Framebuffer, which includes disabling the VideoOut and the PL1RAMs.

5.3.5 PAL FrameBufferDB Core: PalFrameBufferDBRead

macro proc PalFrameBufferDBRead (FrameBufferDBPtr, X, Y, PixelPtr);

Arguments
FrameBufferDBPtr  Pointer to a PalFrameBufferDB structure.
X  X coordinate. Unsigned integer of width log2ceil(PalVideoOutGetVisibleXCT())
Y  Y coordinate. Unsigned integer of width log2ceil(PalVideoOutGetVisibleYCT())
PixelPtr  Pointer to an unsigned 24 variable to receive the pixel.

Timing
More than one clock cycle.

Description
Reads a single pixel from the Framebuffer, from the location specified by \( X \) and \( Y \). The pixel returned via PixelPtr is in 24-bit RGB format.

Note that if you read from a location which you have not written to since the last call to PalFrameBufferDBSwapBuffers, then the read will return pixels set to the background colour, which is black by default.
5.3.6 PAL FrameBufferDB Core: PalFrameBufferDBWrite

macro proc PalFrameBufferDBWrite (FrameBufferDBPtr, X, Y, Pixel);

Arguments  
FrameBufferDBPtr  Pointer to a PalFrameBufferDB structure.
X  X coordinate. Unsigned integer of width \( \log_2\text{ceil}(\text{PalVideoOutGetVisibleXCT}) \)
Y  Y coordinate. Unsigned integer of width \( \log_2\text{ceil}(\text{PalVideoOutGetVisibleYCT}) \)
PixelPtr  Pixel to write to Framebuffer. Must be unsigned 24 in RGB format.

Timing  
One or more clock cycles.

Description  
Writes a single pixel to the Framebuffer, at the location specified by \( X \) and \( Y \). The pixel passed to PalFrameBufferDBWrite must be in 24-bit RGB format. Pixels which are not written to will be displayed in the background colour, which is black by default. The background colour can be changed using PalFrameBufferDBSetBackground.

Note that you should prepare an entire frame for display (using this macro) before calling PalFrameBufferDBSwapBuffers.

5.3.7 PAL FrameBufferDB Core: PalFrameBufferDBSwapBuffers

macro proc PalFrameBufferDBSwapBuffers (FrameBufferDBPtr);

Arguments  
FrameBufferDBPtr  Pointer to a PalFrameBufferDB structure.

Timing  
More than one clock cycle.

Description  
Causes the two frame buffers to be swapped, so that the one which was previously being modified is now displayed. Note that swapping the buffers causes the buffer which is no longer being displayed to be cleared (using a method which does not require writing to every address in the buffer). This means that you must prepare a complete new frame before calling PalFrameBufferDBSwapBuffers again.
5.3.8 PAL FrameBufferDB Core: PalFrameBufferDBSetBackground

macro proc PalFrameBufferDBSetBackground (FrameBufferDBPtr, Colour);

**Arguments**

FrameBufferDBPtr  Pointer to a PalFrameBufferDB structure.
Colour             unsigned 24 to specify new background colour.

**Timing**

One clock cycle.

**Description**

Changes the background colour which the Framebuffer will display when a pixel has not been written to. The Colour parameter must be in 24-bit RGB format.

5.3.9 PAL FramebufferDB example

The first requirement to use the PAL FramebufferDB core is to include the header file:

```
#include "pal_framebufferDB.hch"
```

The library file (pal_framebufferDB.hcl) must also be included, by adding it on the Linker tab of the Project Settings dialog box.

The code sample below shows how to specify a requirement for a video output and two PL1 RAMs, then run and enable the FramebufferDB driver:
macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;

void main (void)
{
  /*
   * Create a pointer to a PalFrameBufferDB structure
   */
  PalFrameBufferDB *FBPtr;

  /*
   * Specify the number of video outputs we require
   */
  PalVideoOutRequire (1);
  PalPL1RAMRequire  (2);

  par
  {
    /*
     * Run the FramebufferDB driver, using PalVideoOutOptimalCT to
     * select the optimal video output mode for the clock rate,
     * and PalPL1RAMCT to select the PL1RAM to use.
     */
    PalFrameBufferDBRun (&FBPtr, PalPL1RAMCT (0),
                         PalVideoOutOptimalCT (ClockRate), ClockRate);

    seq
    {
      /*
       * Enable the FramebufferDB, then run the user code
       */
      PalFrameBufferDBEnable(FBPtr);
      UserCode (FBPtr);
    }
  }
}

macro proc UserCode (FBPtr)
{
  static unsigned (PalVideoOutGetMaxXWidthCT ()) X = 0;
  static unsigned (PalVideoOutGetMaxYWidthCT ()) Y = 0;

  do
  {
    do
    {
      par
{ 
    /*
    * Colour the whole frame in blue
    */
    PalFrameBufferDBWrite (FBPtr, X, Y, 0x0000FF);
    X++;
}
while (X != 0);
Y++;
}
while (Y != 0);

/*
 * Swap buffers, to display what we've written.
 */
PalFrameBufferDBSwapBuffers (FBPtr);

5.4 PAL Block RAM Framebuffer

The PAL Block RAM Framebuffer operates in a similar way to the other Framebuffers, but uses on-chip Block RAM rather than off-chip memory. This limits the resolution and colour, but allows Framebuffer use on platforms where off-chip memory is either not present or is being used for other purposes. There are several features of Block RAM Framebuffer which are not present in the other Framebuffers:

- There is never any delay in writing to the Framebuffer, as there are separate memory ports for read and write.
- The Framebuffer can scale its output up by a power of 2, allowing higher resolution displays to be driven by it.
- Dithered writes are implemented for two types of Block RAM Framebuffer, allowing 24-bit colour pixels to be written to the the 16-bit Framebuffer, and 8-bit monochrome pixels to be written to the 1-bit Framebuffer.

Four types of Block RAM Framebuffer have been implemented:

**RGB888**

This is a 24-bit colour Framebuffer, using 8 bits each for the Red, Green and Blue components. This requires a large amount of memory, so the maximum resolution will be low on all but the largest devices.
RGB565
This is a 16-bit colour Framebuffer, using 5 bits each for the Red and Blue components, and 6 bits for the Green component. The advantage of using 16-bit colour is that the pixel size maps well to the memory width available in typical Block RAMs.

Mono8
This is an 8 bit monochrome Framebuffer, providing 256 grey levels from black at 0x00 to white at 0xFF.

Mono1
This is a 1-bit monochrome Framebuffer, so pixels are either black or white. While the image quality of this Framebuffer is relatively low, the memory usage is very small, allowing fairly high-resolution displays to be driven without scaling.

5.4.1 Running the Block RAM Framebuffer

macro proc PalFrameBufferRGB888Run (FBPtr, Width, Height, Scale, VideoOut, ClockRate);
macro proc PalFrameBufferRGB565Run (FBPtr, Width, Height, Scale, VideoOut, ClockRate);
macro proc PalFrameBufferMono8Run (FBPtr, Width, Height, Scale, VideoOut, ClockRate);
macro proc PalFrameBufferMono1Run (FBPtr, Width, Height, Scale, VideoOut, ClockRate);

Arguments
- FBPtr: Pointer to a pointer to a Framebuffer structure, either PalFrameBufferRGB888, PalFrameBufferRGB565, PalFrameBufferMono8 or PalFrameBufferMono1 as appropriate.
- Width: Width of the framebuffer in pixels.
- Height: Height of the framebuffer in pixels.
- Scale: Factor to scale the video output by.
- VideoOut: Constant PAL Handle for a PalVideoOut resource.
- ClockRate: Clock rate of the clock domain of call to this macro, in Hz.

Timing
Does not terminate in normal use.
Description

Runs the device management tasks for the Framebuffer. Must always run in parallel with accesses to the Framebuffer. Note that the RGB888 Framebuffer uses a very large amount of Block RAM, so the maximum resolution will be low on all but the largest devices.

The Scale parameter is used to allow a higher-resolution video output to be driven by a lower resolution Framebuffer. The amount of scaling will be $2^\text{Scale}$, so if Scale is set to 0, the video output must be the same resolution as the Framebuffer, if it is set to 1, the video output must be double the Framebuffer resolution, and so on. For example, if you run the Framebuffer with a size of 160 x 120, and set Scale to 2, then the actual video output must be 640 x 480.

5.4.2 Writing to the Block RAM Framebuffer

macro proc PalFrameBufferRGB888Write (FBPtr, X, Y, RGB888);
macro proc PalFrameBufferRGB565Write (FBPtr, X, Y, RGB565);
macro proc PalFrameBufferRGB565WriteDither (FBPtr, X, Y, RGB888);
macro proc PalFrameBufferMono8Write (FBPtr, X, Y, );
macro proc PalFrameBufferMono1Write (FBPtr, X, Y, );
macro proc PalFrameBufferMono1WriteDither (FBPtr, X, Y, Mono8);

Arguments

FBPtr Pointer to a pointer to a Framebuffer structure, either PalFrameBufferRGB888, PalFrameBufferRGB565, PalFrameBufferMono8 or PalFrameBufferMono1 as appropriate.

X X co-ordinate to write the pixel to. Unsigned integer

Y Y co-ordinate to write the pixel to.

RGB888 Pixel in 24-bit colour RGB888 format

RGB565 Pixel in 16-bit colour RGB565 format

Mono8 Pixel in 8-bit monochrome format

Mono1 Pixel in 1-bit monochrome format

Timing

1 cycle

Description

Write a pixel to the specified location in the Framebuffer. For the standard write macros, the pixel must be in the same format as used by the Framebuffer. For the dithered write macros, 24-bit RGB pixels can be passed the the RGB565 Framebuffer, and 8-bit
monochrome pixels can be passed to the Mono1 Framebuffer. The pixel will then be dithered appropriately for the Framebuffer being used.
6 Index

C
Console ......................................... 11

D
Double Buffered ............................. 42, 46

F
FrameBuffer .................................. 29, 30, 34, 36, 40, 42, 46

K
Keyboard ......................................... 6

M
Mouse ........................................... 18

P
PAL Console ..................................... 11
PAL Cores ..................................... 6, 11
PAL FrameBuffer 29, 30, 34, 36, 40, 42, 46
PAL FrameBuffer16 ......................... 36, 40
PAL FrameBuffer8 ............................. 30, 34
PAL FrameBufferDB ......................... 42, 46
PAL Keyboard ................................. 6
PAL Mouse ..................................... 18