Platform Developer’s Kit

PAL manual

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

Contents

1 PLATFORM ABSTRACTION LAYER (PAL) 7

2 INTRODUCTION TO PAL 8
2.1 PAL LIBRARY AND HEADER FILES 8
2.2 SUPPORTED PLATFORMS 8
2.3 SUPPORTED DEVICES 8
2.4 PAL EXAMPLES 9
2.5 PAL CORES 9
2.6 PAL VIRTUAL PLATFORM (PALSIM) 10
2.6.1 PALSim examples 11

3 PAL USER GUIDE 12
3.1 CREATING YOUR OWN PAL DESIGN 12
3.1.1 Creating a generic PAL library 12
3.2 USING PAL CORES 12
3.3 TARGETING A SPECIFIC PLATFORM 13
3.3.1 Saving project settings for different platforms 13
3.4 TARGETING THE PALSIM PLATFORM 14
3.4.1 Running a PALSim simulation 14
3.4.2 Stopping a PALSim simulation 14
3.5 PROGRAMMING WITH PAL 14
3.5.1 Resource handles 14
3.5.2 Compile-time and run-time selection 15
3.5.3 Data width abstraction 15
3.5.4 Setting up a PAL device 16
3.5.5 PAL Kit workspace 16

4 PAL API REFERENCE 18
4.1 HEADER AND LIBRARY FILES 18
4.2 GENERIC API 19
4.2.1 Resource handles 19
4.2.2 API version numbers 19
4.2.3 Library version information 20
4.2.4 Error handling in PAL 21
4.2.5 Generic methods 21
4.3 RESOURCE-SPECIFIC API 24
4.3.1 PAL LED API 25
4.3.2 Seven-segment displays (SevenSeg API) 25
4.3.3 Binary switches and buttons (Switch API) 26
4.3.4 Generic data I/O (DataPort API) 27
4.3.5 Parallel port API 28
4.3.6 RS-232 serial ports (RS232Port API) 28
4.3.7 PS/2 serial ports (PS2Port API) 29
4.3.8 Single cycle RAMs (FastRAM API) 29
4.3.9 Pipelined single-cycle RAMs (PL1RAM API) 31
4.3.10 Pipelined PL2 RAMs 33
4.3.11 RAMs without guaranteed timing (SlowRAM API) 35
4.3.12 Block storage devices (BlockStore API) 37
4.3.13 Video capture devices (VideoIn API) 40
4.3.14 Video output devices (VideoOut API) 43
4.3.15 Audio input devices (AudioIn API) 48

4.3.16 Audio output devices (AudioOut API) 49
4.3.17 PAL SDRAM API 50
4.3.18 Ethernet devices (Ethernet API) 56
4.3.19 PAL Reconfigure API 58
4.3.20 PAL TouchScreen API 59

5 INDEX 63

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:

```
void main();
```

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

`copy` ***`SourceFileName DestinationFileName`***

Optional elements are enclosed in square brackets like this:

`struct` [type_Name]

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

`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 Platform Abstraction Layer (PAL)

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.

PAL is implemented as a thin layer on top of a Platform Support Library (PSL).

The Celoxica PAL distribution requires DK3 or later. You can check which version of DK you have installed by launching DK and selecting Help>About DK Design Suite.

2 Introduction to PAL

2.1 PAL library and header files

The PAL implementation exists as a set of Handel-C header files and library files. The header file `pal.hch` declares the general PAL API. This is supplemented by platform specific header files (`pal_rc100.hch`, `pal_ndb.hch` etc) which provide information about clocking and the specific resources on a platform. The library files contain PAL implementations for specific platforms. The PAL header and library files are in the `Include` and `Lib` directories under your `PDK\Hardware` installation directory.

To use PAL, you need to set the paths to the library and include files for PAL in DK:

1. Select Tools>Options and open the Directories tab.
2. Click Add and browse to ***`InstallDir`***`\PDK\Hardware\Include`. Then click OK.
3. Select Library modules in the Show directories for drop down list.
4. Click Add and browse to ***`InstallDir`***`\PDK\Hardware\Lib`. Then click OK.
5. Click OK to close the Tool Options dialog.

If you want to target a platform that is not currently supported by PAL, you will need to create your own Platform Support Library. This is described in the PSL Tutorial Guide.

2.2 Supported platforms

PAL version 1.3 has support for the following platforms:

- Celoxic RC10
- Celoxica RC100
- Celoxica RC1000
- Celoxica RC200 and RC200E
- Celoxica RC203 and RC203E
- Celoxica RC250 and RC250E
- Celoxica RC300 and RC300E
- Celoxica RC2000 (ADM-XRC-II)
- Altera NiosII development board stratix edition (NDBS)
- Memec MV2P
- Celoxica PALSim Virtual Platform

2.3 Supported devices

PAL version 1.3 has support for the following devices:

- LEDs

- Seven segment displays
- Switches and buttons
- Data ports
- Parallel ports
- RS-232 serial ports
- PS2 ports
- Fast RAMs
- Pipelined RAMs (PL1 and PL2)
- Slow RAMs
- SDRAMs
- Block storage devices (such as flash memory)
- Video output devices
- Video input devices
- Ethernet
- Audio input devices
- Audio output devices

See the ***PAL resource-specific API reference*** (see page 24) for more information.

2.4 PAL examples

PAL comes with several example projects which you can compile and run on the different PAL platforms. There is also a PAL tutorial application.

To use the examples, open the PAL Examples Workspace in DK by selecting Start>Programs>Celoxica>Platform Developer's Kit>PAL>PAL Examples Workspace.

The workspace includes a set of configurations that will let you build the examples on platforms that feature the required peripherals. You can choose the combination of project and platform from the drop down menus in the Build toolbar. Alternatively you can select a configuration by clicking on the Build>Set Active Configuration menu.

Once you have selected the configuration you wish to build, click on the build icon, or press F7 to start the compilation.

If you target a Celoxica supported platforms, DK will automatically run the Xilinx or Altera place and route tools for you to produce a configuration file for these platforms. You must already have the place and route software installed for this to work.

If you select the Celoxica Virtual Simulation platform (Sim configuration), you can simulate the example after building it by pressing F5. This will launch the PALSim application.

2.5 PAL Cores

PAL is augmented with a set of PAL Cores. These are generic IP Cores that add functionality to the PAL devices. The PAL Cores supplied with PAL version 1.2 are:

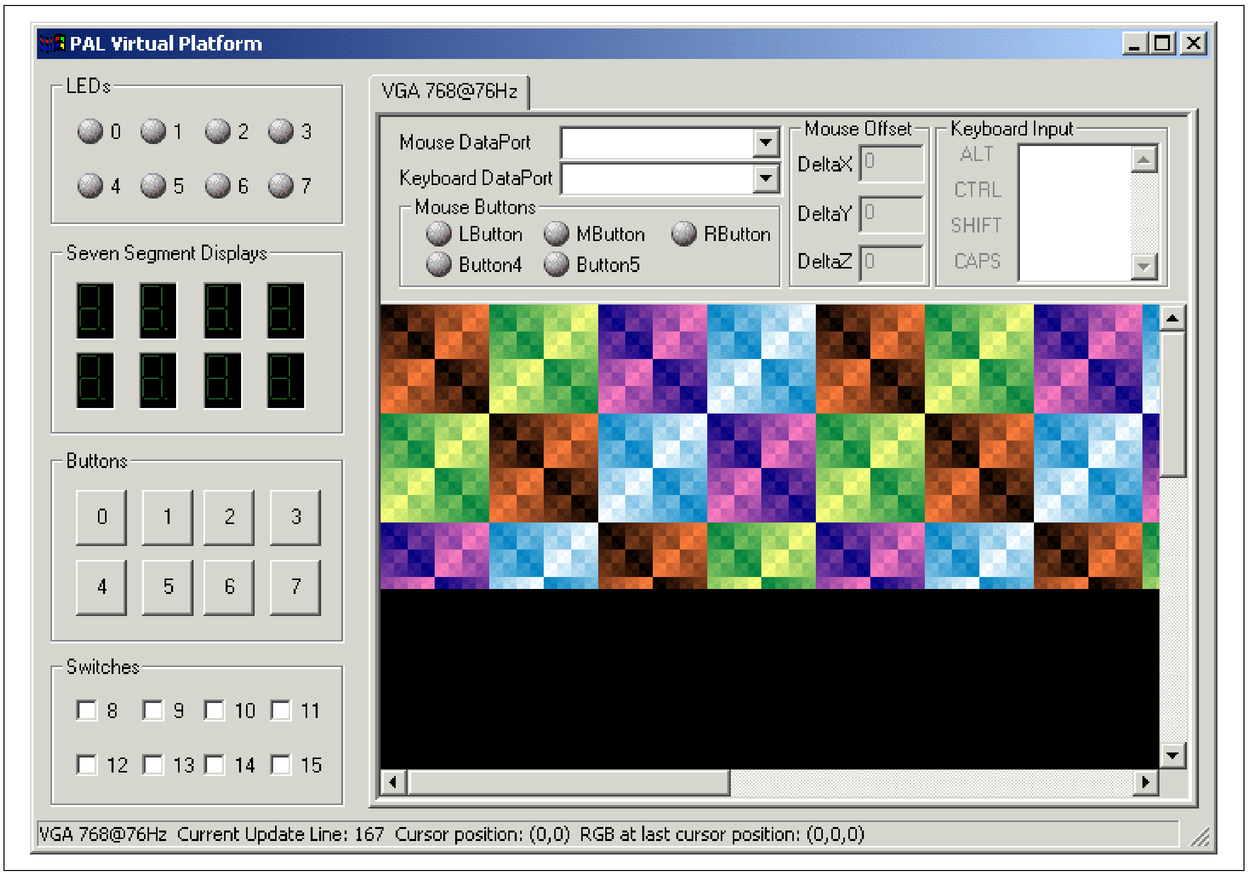
- Console: a generic video console library.
- Framebuffer: generic libraries for 16 and 8-bit frame buffers.
- Mouse driver: a generic mouse driver library.

- Keyboard driver: a generic keyboard driver library.

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

2.6 PAL Virtual Platform (PALSim)

PALSim allows you to simulate PAL designs. When you run a simulation, the PALSim application appears and provides a visual representation of the behaviour of devices such as a VGA screen, RAM and LEDs.



VGA 768@76Hz Current Update Line: 167 Cursor position: (0,0) RGB at last cursor position: (0,0,0)

PAL VIRTUAL PLATFORM (PALSIM)

The following devices are included in the PALSim application:

- LEDs
- Seven segment displays
- Switches
- Buttons
- Generic data ports. These can be used to simulate a parallel port or an RS232 serial port. You can browse and select a file for input or output.
- Mouse. Mouse input from your PC during a simulation is shown on as a cursor on the VGA simulation. If you click the mouse buttons, this information is sent to the data port and displayed in the Mouse Buttons area above the VGA screen.
- Keyboard. You can type characters into a window on the PALSim application. The information is sent to the data port and can be displayed on the VGA screen simulation.
- Fast RAM

- PL1 Pipelined RAM
- Block storage devices. These can be used to simulate Flash RAM.
- Video output devices. You can simulate the following VGA outputs: 480 at 60Hz refresh, 480 at 75Hz refresh, 600 at 60Hz refresh, 600 at 72Hz refresh, 768 at 60Hz refresh or 768 at 76Hz refresh.
- Video input devices. You can simulate the following inputs: 160 x 120, 176 x 144, 320 x 240, 352 x 288, 720 x 480 or 720 x 576.
- Ethernet. If you enable an Ethernet simulation, it will detect any packets sent across the network to your computer. For example, if you open an external web page, the packets for the page will be displayed in PALSim. The following information is displayed: the MAC address of computer the packets have come from, the destination address (your computer), the size of packet, the time and the type of packet. The contents of the packet are also displayed, and you can save this information in a file. You can also view and capture packets as they are sent from an application onto the network.
- Audio input devices
- Audio output devices

2.6.1 PALSim examples

Most of the examples in the PAL examples workspace are configured to work with PALSim.

To open the examples workspace, select Start>Programs>Celoxica>Platform Developer’s Kit>PAL>PAL Examples Workspace.

Choose a project in the left-hand drop down box at the top of the DK screen, and then select Sim as the build configuration in the box next to this.

Some of the examples take a long time to compile. Quicker examples include: sevenseg, led, videoout and fastram.

3 PAL User Guide

3.1 Creating your own PAL design

You need to set up your project to use the correct header files and libraries. You also need to include the relevant header files in your source code.

The quickest way to get started with PAL is to take one of the existing projects in the PAL Examples Workspace (Start>Programs>Celoxica>Platform Developer's Kit>PAL>PAL Examples Workspace) and modify the code to meet your own design. Alternatively, you can use the Pal Kit Workspace (Start>Programs>Celoxica>Platform Developer's Kit>PAL>PAL Kit Workspace), which contains empty implementations for each of the supported PAL devices.

If you need to create a new workspace and project you must manually ***set the project configurations in DK*** (see page 13).

The examples provided with the PAL distribution use a unified header file `pal_master.hch`. This automatically includes one of the platform-specific header files depending on C preprocessor `#define` macros that are set in the project configuration. Different project configurations link against different libraries and set different `#define` macros to enable you to switch between target platforms by changing the active project configuration. You are encouraged to follow this approach, and extend the `pal_master.hch` header file to include any new PAL implementations that you create.

3.1.1 Creating a generic PAL library

If you wish to create a library that uses PAL and is not specific to any platform, you only need to include the `pal.hch` header in your source files. When that library is used as part of an application to generate hardware or a simulation model, the application will be linked against a specific PAL implementation and the application source will include the platform-specific header file.

3.2 Using PAL Cores

If you wish to use PAL Cores in your application you should also include the appropriate headers in your source files and link with the appropriate library files. The libraries and header files are in the `Lib` and `Include` directories within ***`InstallDir`***`\PDK\Hardware\`.

Some of the ***PAL examples*** (see page 9) use PAL Cores.

Core	***Header file***	***Library file***	***Examples***
Console	`pal_console.hch`	`pal_console.hcl`	console, dumb terminal and keyboard
16-bit Framebuffer	`pal_framebuffer16.hch`	`pal_framebuffer16.hcl`	blockstore, framebuffer16 and videoin
8-bit Framebuffer	`pal_framebuffer8.hch`	`pal_framebuffer8.hcl`	framebuffer8
Mouse	`pal_mouse.hch`	`pal_mouse.hcl`	mouse
Keyboard	`pal_keyboard.hch`	`pal_keyboard.hcl`	dumb terminal, keyboard

3.3 Targeting a specific platform

If you wish to create a PAL application that targets a specific PAL platform you must include the platform-specific header file in your source files and set DK to link your project with the following libraries:

- PAL library for your chosen target (e.g. `pal_rc200.hcl`)
- PSL for your chosen target (e.g. `rc200.hcl`)
- The standard Handel-C library `stdlib.hcl`

You need to add these libraries to the Object/Library modules box on the Linker tab in Project Settings. The libraries are in ***`InstallDir`***`\PDK\Hardware\Include`.

PALSim (see page 14) requires different settings.

3.3.1 Saving project settings for different platforms

You can save project settings for PAL projects as a new configuration. This allows you to change the settings to build for different targets easily.

1. Select Build>Configurations... and then click the Add button.
2. Enter a name for your new configuration, and select the configuration type that you wish to use as a base in the Copy settings from box. Click OK.
3. Click the Close box.
4. Open the Project settings dialog, select the new configuration and edit the settings as required.

The configuration will only be available in the project you defined it in.

If you want to set a configuration for the RC300, RC200, RC100, RC1000, Nios or PALSim, you can copy the project settings for the relevant configuration used in the PAL Examples workspace.

3.4 Targeting the PALSim platform

If you wish to target the PALSim virtual platform, you must set your project to link with the PAL and PSL Handel-C libraries. You also need to add the following settings:

1. Open the Project>Settings dialog and check that you are in Debug configuration.
2. On the Linker tab, add `sim.hcl` and `pal_sim.hcl` to the Object/library modules box and `PalSim.lib` to the Additional C/C++ Modules box. These files are in ***`InstallDir`***`\PDK\Hardware\Lib` and ***`InstallDir`***`\PDK\Software\Lib`.
3. On the Preprocessor tab, add `USE_SIM` to the Preprocessor definitions box.

You can save these settings as a project configuration.

> `sim.hcl` is the PSL for the PALSim platform and `pal_sim.hcl` is the PAL library for PALSim. `PALSim.lib` is the linker library which connects to the PALSim application.

The PALSim virtual platform also requires the dynamic link library `PalSim.dll` to be in your system path. This is copied into the windows system directory of your computer by the PAL installer.

If you want to simulate PAL Ethernet applications, you need to have a packet driver installed on your PC. The WinPCap packet driver is installed as part of PDK.

3.4.1 Running a PALSim simulation

To run a PALSim simulation, make sure that your project has the correct project settings and then compile your code using the Debug configuration or your own PALSim configuration. The PAL examples have a Sim configuration which you can use. Press F5 to run the program in the debugger, which will start PALSim as well.

3.4.2 Stopping a PALSim simulation

You can end a PALSim simulation by selecting Debug>Stop debugging in DK, or by pressing Shift + F5. If you use the Shift + F5 shortcut, you must have the DK window in focus, rather than the PALSim window. It is also possible to close the PALSim window directly, which will leave the DK simulation still running. Note that if you do this you will need to restart the simulation in DK to get the PALSim window back again.

3.5 Programming with PAL

3.5.1 Resource handles

Resource handles provide abstraction over the peripherals on a platform. An implementation of PAL provides each peripheral on a platform with at least one resource handle.

Every PAL implementation will provide macros that tell you how many resources of a particular type are present. For example, the macro `PalLEDCount()` will return the number of LEDs there are on your target platform. Even if there are no LEDs, the macro is still implemented and will simply return 0. An implementation also provides macros for accessing resource handles. For example, the macro

PalLED(Index) takes an argument Index and returns the resource handle corresponding to a specific LED. This enables you to write PAL applications that are parameterized to match the resources on the platform for which they are built.

You use the resource handle to specify which peripheral you wish to access using other members of the PAL API. For example, the macro PalLEDWrite(***Handle***, ***Data***) sets a value on the LED referenced by ***Handle***.

Resource handles are also used to select between different modes of operation that a peripheral has. For example, a video output resource that can operate in different resolutions will have a resource handle for each resolution mode. The number of resources returned by Pal***X***Count() (see page 22) may be greater than the number of physical X-type resources on the platform. You can query exactly how many different resources a platform has by using the macro Pal***X***UniqueCount() (see page 22). See the ***PAL API reference*** (see page 18) for further information.

PAL resource handles example

The following code example turns on all the LEDs on a platform in a platform-independent fashion. It uses PalLEDCount() to return the number of LEDs on the platform, PalLED() to get a handle to each resource, and PalLEDWrite() to set a value on each LED resource.

```
static unsigned (log2ceil(PalLEDCount()) + 1) i = 0;
while (i < PalLEDCount())
{
   PalLEDWrite( PalLED (i), 1);
   i++;
}
```

3.5.2 Compile-time and run-time selection

PAL provides macros for compile-time and run-time evaluation.

The compile-time macros can be used to perform conditional compilation (with ifselect) and to set the widths of variables. These macros take arguments which the DK compiler must be able to evaluate at compile time, so for instance they cannot be passed through a function parameter. Macros which require arguments that can be evaluated at compile time are denoted by the suffix CT. For example the macro Pal***X*** (***Index***) (see page 21) takes a non-constant argument Index and returns a non-constant resource handle, whereas Pal***X***CT(***IndexCT***) (see page 21) requires the argument ***IndexCT*** to be a constant and returns a constant resource handle.

PAL run-time macros can be used to dynamically index through a set of peripherals, as shown in the PAL resource handles example.

3.5.3 Data width abstraction

PAL provides abstraction over the different data widths of peripherals. You can use the PAL API to query the data widths of a resource and parameterize your code accordingly.

There are three general macros that you can use to abstract over data width:

- Pal***X***GetMaxDataWidthCT()
- Pal***X***GetDataWidth(***Handle***)
- Pal***X***GetDataWidthCT(***HandleCT***)

The first macro, Pal***X***GetMaxDataWidthCT() returns the maximum width of the data for any devices of type ***X*** at compile time. The PAL macros that access these devices expect arguments of this width.

You can use this macro to define the widths of variables in your code and to perform conditional compilation.

The second macro, `PalXGetDataWidth(`***`Handle`***`)` returns the actual data width of the resource referenced by ***`Handle`*** at run time. The return value is not a compile-time constant and so it cannot be used to define the width of variables or with the Handel-C take and drop operators. However, the argument ***`Handle`*** need not be a compile-time constant, and so it can be passed through a function parameter or stored in a run-time variable.

The third macro, `PalXGetDataWidthCT(`***`HandleCT`***`)` returns the actual data width of the resource referenced by ***`HandleCT`*** at compile time. The return value can be used to define the width of variables and to perform conditional compilation. The argument ***`HandleCT`*** must be a compile-time constant.

A similar set of three macros is used to abstract over RAM address widths, and other device characteristics such as the dimensions of blocks in a block store type device. See the ***PAL API reference*** (see page 18) for further information.

3.5.4 Setting up a PAL device

LEDs, seven segment displays, buttons and switches have a simplified interface in PAL. To use them, set up the PAL project and use the PAL access macros, `PalXWrite(`***`Handle, Data`***`)` and `PalXRead(`***`Handle, DataPtr`***`)`.

More complex devices have a slightly more complex, but standardized interface. The generic pattern of use is:

```
par
{
    PalXRun (Handle, ClockRate);
    seq
    {
        PalXEnable (Handle);
        PalXRead (Handle, &Data);     // etc.
        PalXDisable (Handle);
    }
}
```

The `PalXRun(`***`Handle, ClockRate`***`)` (see page 23) macro initializes the resource and then performs device management tasks. This macro does not terminate and must be run in parallel with other PAL macros that access the resource.

`PalXEnable(`***`Handle`***`)` ensures the device referenced by `Handle` is ready for use. This macro will block until the initialization performed by `PalXRun(`***`Handle, ClockRate`***`)` is complete. On some platforms where separate devices share a common bus it may be necessary for you to disable one resource before you can successfully enable another. The release notes for a specific PAL implementation will include this information. PAL applications should always disable resources when they are not required.

3.5.5 PAL Kit workspace

The PAL distribution includes a PAL Kit workspace that you can use as a starting point for your implementation (Start>Programs>Celoxica>Platform Developer's Kit>PAL>PAL Kit Workspace). The PAL kit contains empty implementations for each of the supported PAL devices.

To create your own implementation, copy the PAL kit header files and source files and replace the empty implementations with the devices that are present on your target platform. The PAL kit project uses separate header files and source files to implement missing PAL devices. You should use these to provide the default implementation for any missing resources on your own platform.

4 PAL API reference

The Celoxica Platform Abstraction Layer Application Programming Interface (PAL API) contains methods for accessing the following devices:

- LEDs
- Seven segment displays
- Switches
- Data ports
- Parallel ports
- RS-232 serial ports
- PS/2 ports
- Fast RAMs
- Pipelined RAMs
- Slow RAMs
- SDRAM
- Block storage devices
- Video input devices
- Video output devices
- Audio input devices
- Audio output devices
- Ethernet devices
- Audio
- Touchscreen
- Reconfiguration resources

PAL defines a set of methods common to all of the devices, the PAL generic API. Device-specific methods are described in the ***PAL resource-specific API*** (see page 24).

4.1 Header and library files

The PAL distribution contains the header file `pal.hch` which contains the declarations of all the types and macros described in this document. All applications and libraries that use PAL should `#include` this header.

Applications must link to the appropriate PAL library for the platform they are targeting. For instance, `pal_rc200.hcl` is needed to target the Celoxica RC200 board. This in turn requires the PSL library `rc200.hcl` and the standard library `stdlib.lib`. More information about choosing the appropriate libraries and header files is supplied in the ***PAL User Guide*** (see page 12).

4.2 Generic API

4.2.1 Resource handles

PALHandle

Description The type of references to PAL resources.

A `PalHandle` is used to distinguish a particular resource on a platform from the other resources of the same type. Handles to particular resources are obtained by the `Pal`***`X`***`()` and `Pal`***`X`***`CT()` family of macros. All macros that need to refer to a particular resource take a `PalHandle` as their first argument.

4.2.2 API version numbers

PalVersionMajor () macro

```
macro expr PalVersionMajor ();
```

Arguments None.

Return value Integer constant.

Description Expression that returns the major component of the API version number. For example, it would return 2 for API version 2.3.

Changes in the major version may add and remove functionality, or may change the behaviour of existing APIs. Code written with one major number may or may not work with another.

PalVersionMinor () macro

```
macro expr PalVersionMinor ();
```

Arguments None.

Return value Integer constant.

Description Expression that returns the minor component of the API version number. For example, it would return 3 for API version 2.3.

Changes in the minor version may add functionality to the API, but should not break existing code. New APIs are therefore backwards compatible.

PalVersionRequire () macro

```
macro proc PalVersionRequire (Major, Minor);
```

Arguments	***`Major`***: major API version number required.
	`Minor`: minor API version number required.
Timing	0 clock cycles.
Description	Procedure that fails with a compile-time assertion if the provided version number is incompatible with the API version of the library it is linked against.
	Applications can ensure that they are compiling with a compatible API by calling `PalVersionRequire(`***`Major`***`, `***`Minor`***`)` with the major and minor API version for which they were originally written.

4.2.3 Library version information

PalVendorCode () macro

```
macro expr PalVendorCode ();
```

Arguments	None.
Return value	Integer constant.
Description	Returns a specific integer code uniquely identifying a vendor.
	Vendors should use their PCI vendor ID code if possible. For instance, Celoxica is 0x4144, Intel is 0x8086. Vendors without a PCI vendor ID code should use the space above 0x10000, contacting Celoxica before releasing a library in order to avoid conflicts.

PalVendorRelease () macro

```
macro expr PalVendorRelease ();
```

Arguments	None.
Return value	Integer constant.
Description	Returns a vendor-specific integer code uniquely identifying a given library and release number.
	Applications typically only need to check vendor code and release number to provide source-level workarounds for known bugs in specific library releases.

PalVendorString () macro

```
macro expr PalVendorString ();
```

Arguments	None.
Return value	String constant.
Description	Returns a vendor specific string, primarily for use in debugging.

4.2.4 Error handling in PAL

PalErrorCode

Description	The type of PAL run-time error codes. Possible values are: `PAL_ERROR_OK`: No error. `PAL_ERROR_INVALID_HANDLE`: A method was passed a handle referring to a resource that is not of an allowed type for that method. `PAL_ERROR_DEVICE_FAILURE`: A method has encountered physical device failure.

PalErrorHandlerRun () macro

```
macro proc PalErrorHandlerRun (ErrorHandlerProc);
```

Arguments	***`ErrorHandlerProc`***: a macro proc taking a single argument of type `PalErrorCode`.
Timing	Terminates after executing `ErrorHandlerProc()`, if an error occurs.
Description	This procedure runs indefinitely until a runtime error condition is detected, at which point it calls the error handler procedure with the error code as an argument. The error handler procedure may, for example, reset the whole system or drive some form of external error reporting (such as an LED).

4.2.5 Generic methods

PalX () macro: non constant PalHandle

```
macro expr PalX (Index);
```

Arguments	***`Index`***: an `unsigned int` index.
Return value	Non-constant `PalHandle`.
Description	Returns the ***`Index`*** numbered `PalHandle` from all resources of type ***`X`*** available on the platform. ***`Index`*** need not be a constant. ***`Index`*** must be in the range 0 to (`PalXCount ()` – 1) inclusive.

PalXCT () macro: constant PalHandle

```
macro expr PalXCT (IndexCT);
```

Arguments	***`IndexCT`***: an integer constant index.
Return value	Constant `PalHandle`.
Description	Returns the ***`IndexCT`*** numbered `PalHandle` from all resources of type ***`X`*** available on the platform. ***`Index`*** must be a constant in the range 0 to (`PalXCount ()` – 1) inclusive.

PalXCount () macro

```
macro expr PalXCount ();
```

Arguments	None.
Return value	Integer constant.
Description	Expression that returns the total number of resources of type ***X*** on the platform. The resources are not necessarily unique (the same physical resource may be referred to by several indices).

PalXUniqueCount () macro

```
macro expr PalXUniqueCount ();
```

Arguments	None.
Return value	Integer constant.
Description	Expression that returns the total number of unique resources of type ***X*** on the platform. All unique resource indices map to distinct physical resources.

PalXRequire () macro

```
macro proc PalXRequire (Count);
```

Arguments	***`Count`***: integer constant.
Timing	0 clock cycles.
Description	Checks that at least ***`Count`*** resources of type ***X*** are available at compile time, and produces a suitable error message if not.

PalXGetMaxDataWidthCT () macro

```
macro expr PalXGetMaxDataWidthCT ();
```

Arguments	None.
Return value	Integer constant.
Description	Returns the maximum width of data that is to be passed to or from resources of type ***X***, at compile time. This is typically the width that should be used for read and write methods.

PalXGetDataWidth () macro

```
macro expr PalXGetDataWidth (Handle);
```

Arguments	***`Handle`***: `PalHandle` to resource of type ***X***.
Return value	Integer value, evaluated at runtime.
Description	Returns the actual width of data that is used by the handle referred to, at runtime. Since this method works at runtime it can be used even when ***`Handle`*** is not a constant (such as when it is passed through a function parameter). However, it cannot be used to set the width of variables. It may, for example, be used to choose suitable data formats at runtime.

PalXGetDataWidthCT () macro

```
macro expr PalXGetDataWidthCT (HandleCT);
```

Arguments	***HandleCT***: constant `PalHandle` to resource of type ***X***.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the actual width of data that is used by the handle referred to, at compile time. Since this method works at compile time it can only be used where the PAL Handle is a constant, and can be deduced to be a constant by the DK compiler. So for instance, this cannot be used if ***HandleCT*** is passed through a function parameter. However, since it returns a compile time constant it can be used to set variable widths and do conditional compilation (using `select` and `ifselect`).

PalXRun () macro

```
macro proc PalXRun (HandleCT, ClockRate);
```

Arguments	***HandleCT***: constant `PalHandle` to resource of type ***X***.
	ClockRate: clock rate, in Hertz, of the domain in which the macro is being called.
Timing	Does not terminate in normal use.
Description	Runs the device management tasks for resource ***X***.
	Must always run in parallel with Enable/Disable/Read/Write etc.

PalXReset () macro

```
macro proc PalXReset (Handle);
```

Arguments	***Handle***: `PalHandle` to resource of type ***X***.
Timing	0 or more clock cycles.
Description	Resets the resource, i.e. brings the resource to a known state. As far as possible this known state is the state of the resource immediately after initialization (this is not always possible for some peripherals). The statement finishes executing once the reset is complete.

PalXEnable () macro

```
macro proc PalXEnable (Handle);
```

Arguments	***Handle***: `PalHandle` to resource of type ***X***.
Timing	0 or more clock cycles.
Description	Enables the resource. This is typically needed to arbitrate between shared resources, or resources on shared buses.

PalXDisable () macro

```
macro proc PalXDisable (Handle);
```

Arguments	***Handle***: `PalHandle` to resource of type ***X***.
Timing	0 or more clock cycles.
Description	Disables the resource. Reverses `PalXEnable ()`.

PalXRead (Handle, DataPtr);

`macro proc Pal`***`X`***`Read (`***`Handle`***`, `***`DataPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to resource of type ***`X`***.
	`DataPtr`: pointer to an lvalue (a writable resource, e.g. variable or signal) of the appropriate type. For many resources this is an `unsigned int` of width `(Pal`***`X`***`GetMaxDataWidthCT ())`.
Timing	1 or more clock cycles.
Description	Reads a single item of data from the resource, and stores it in the lvalue pointed at by ***`DataPtr`***.

PalXWrite () macro

`macro proc Pal`***`X`***`Write (`***`Handle`***`, `***`Data`***`)`

Arguments	***`Handle`***: `PalHandle` to resource of type ***`X`***.
	`Data`: expression of the appropriate type. For many uses this is an `unsigned int` of width `(Pal`***`X`***`GetMaxDataWidthCT ())`.
Timing	1 or more clock cycles.
Description	Statement that writes a single item of data to the resource.

4.3 Resource-specific API

The following resource-specific APIs are included in the current version of PAL:

- LED API
- Seven-segment API
- Switch API (binary switches and buttons)
- Generic data I/O API
- Parallel port API
- RS-232 serial port API
- PS/2 serial port API
- Single-cycle RAM access API
- Pipelined single-cycle RAM access API
- Pipelined PL2 RAM API
- SlowRAM API (RAMs without guaranteed timing)
- SDRAM (Single-Rate Synchronous dynamic random memory)
- Block storage device API
- Video capture device API
- Video output device API
- Audio input device API
- Audio output device API
- Ethernet API
- Reconfigure API

4.3.1 PAL LED API

The LED API supports simple binary LEDs. More complex LEDs (such as multicolour) can be supported by using several LED resources in conjunction, or via the data port API.

- `macro expr PalLED (`***`Index`***`);`
  see `macro expr PalX (`***`Index`***`);` (see page 21)
- `macro expr PalLEDCT (`***`IndexCT`***`);`
  see `macro expr PalXCT (`***`IndexCT`***`);` (see page 21)
- `macro expr PalLEDCount ();`
  see `macro expr PalXCount ();` (see page 22)
- `macro expr PalLEDUniqueCount ();`
  see `macro expr PalXUniqueCount ();` (see page 22)
- `macro proc PalLEDRequire (`***`Count`***`);`
  see `macro proc PalXRequire (`***`Count`***`);` (see page 22)
- `macro proc PaLEDWrite (`***`Handle`***`, `***`Data`***`);`

PalLEDWrite() macro

`macro proc PaLEDWrite (`***`Handle`***`, `***`Data`***`);`

Arguments	***`Handle`***: `PalHandle` to an LED resource.
	`Data`: expression of type `unsigned 1`.
Timing	1 clock cycle.
Description	Statement, Turns an LED on or off. A value of 1 means on, 0 means off.

4.3.2 Seven-segment displays (SevenSeg API)

The seven-segment display API supports standard seven-segment (plus decimal point) displays. More complex displays can be supported via the data port API.

- `macro expr PalSevenSeg (`***`Index`***`);`
  see `macro expr Pal`***`X`***` (`***`Index`***`);` (see page 21)
- `macro expr PalSevenSegCT (`***`IndexCT`***`);`
  see `macro expr Pal`***`X`***`CT (`***`IndexCT`***`);` (see page 21)
- `macro expr PalSevenSegCount ();`
  see `macro expr Pal`***`X`***`Count ();` (see page 22)
- `macro expr PalSevenSegUniqueCount ();`
  see `macro expr Pal`***`X`***`UniqueCount ();` (see page 22)
- `macro proc PalSevenSegRequire (`***`Count`***`);`
  see `macro proc Pal`***`X`***`Require (`***`Count`***`);` (see page 22)
- `macro proc PalSevenSegEnable (`***`Handle`***`);`
  see `macro expr Pal`***`X`***`Enable (`***`Handle`***`);`
- `macro proc PalSevenSegDisable (`***`Handle`***`);`

see `macro expr Pal`***`X`***`Disable (`***`Handle`***`);`

- `macro proc PalSevenSegWriteShape (`***`Handle`***, ***`ShapeMask`***`);`
- `macro proc PalSevenSegWriteDigit(`***`Handle`***, ***`Value, DecimalPoint`***`);`

PalSevenSegWriteShape () macro

`macro proc PalSevenSegWriteShape (`***`Handle`***`, `***`ShapeMask`***`);`

Arguments	***`Handle`***: `PalHandle` to a SevenSeg resource.
	`ShapeMask`: expression of type `unsigned 8`.
Timing	1 clock cycle.
Description	Set a particular shape in the seven-segment display. ***`ShapeMask`*** is a binary mask where 1 means ON and 0 means OFF. Each of the eight bits corresponds to a segment of the display (7-segments for the digit and one for the decimal point).

PalSevenSegWriteDigit () macro

`macro proc PalSevenSegWriteDigit (`***`Handle`***`, `***`Value`***`, `***`DecimalPoint`***`);`

Arguments	***`Handle`***: `PalHandle` to a SevenSeg resource.
	`Value`: expression of type `unsigned 4`.
	`DecimalPoint`: expression of type `unsigned 1`.
Timing	1 clock cycle.
Description	Set a particular hexadecimal digit in the seven-segment display. ***`Value`*** is the hex value, and ***`DecimalPoint`*** specifies whether the decimal point should be turned on (1) or off (0).

4.3.3 Binary switches and buttons (Switch API)

The switch API supports simple binary switches or buttons, typically used for debugging. More complex switches can be supported via the data port API.

- `macro expr PalSwitch (`***`Index`***`);`
  see `macro expr Pal`***`X`***` (`***`Index`***`); (see page 21)`
- `macro expr PalSwitchCT (`***`IndexCT`***`);`
  see `macro expr Pal`***`X`***`CT (`***`IndexCT`***`); (see page 21)`
- `macro expr PalSwitchCount ();`
  see `macro expr Pal`***`X`***`Count (); (see page 22)`
- `macro expr PalSwitchUniqueCount ();`
  see `macro expr Pal`***`X`***`UniqueCount (); (see page 22)`
- `macro proc PalSwitchRequire (`***`Count`***`);`
  see `macro proc Pal`***`X`***`Require (`***`Count`***`); (see page 22)`
- `macro proc PalSwitchRead (`***`Handle`***`, `***`DataPtr`***`);`

PalSwitchRead () macro

`macro proc PalSwitchReadPalSwitchRead (`***`Handle`***`, `***`DataPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to a Switch resource.
	`DataPtr`: pointer to an lvalue (e.g. variable or signal) of type `unsigned 1`.
Timing	1 clock cycle.
Description	Read the current state of a switch. A value of 1 means ON (or closed), a value of 0 means OFF (or open).

4.3.4 Generic data I/O (DataPort API)

- `macro expr PalDataPort (`***`Index`***`);`
  see `macro expr Pal`***`X`***` (`***`Index`***`); (see page 21)`
- `macro expr PalDataPortCT (`***`IndexCT`***`);`
  see `macro expr Pal`***`X`***`CT (`***`IndexCT`***`); (see page 21)`
- `macro expr PalDataPortCount ();`
  see `macro expr Pal`***`X`***`Count (); (see page 22)`
- `macro expr PalDataPortUniqueCount ();`
  see `macro expr Pal`***`X`***`UniqueCount (); (see page 22)`
- `macro proc PalDataPortRequire (`***`Count`***`);`
  see `macro proc Pal`***`X`***`Require (`***`Count`***`); (see page 22)`
- `macro expr PalDataPortGetMaxDataWidthCT ();`
  see `macro expr Pal`***`X`***`GetMaxDataWidthCT ();`
- `macro expr PalDataPortGetDataWidth (`***`Handle`***`);`
  see `macro expr Pal`***`X`***`GetDataWidth (`***`Handle`***`);`
- `macro expr PalDataPortGetDataWidthCT (`***`HandleCT`***`);`
  see `macro expr Pal`***`X`***`GetDataWidthCT (`***`HandleCT`***`);`
- `macro proc PalDataPortRun (`***`HandleCT`***`, `***`ClockRate`***`);`
  see `macro proc Pal`***`X`***`Run (`***`HandleCT`***`, `***`ClockRate`***`); (see page 23)`
- `macro proc PalDataPortReset (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Reset (`***`Handle`***`);`
- `macro proc PalDataPortEnable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Enable (`***`Handle`***`);`
- `macro proc PalDataPortDisable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Disable (`***`Handle`***`);`
- `macro proc PalDataPortRead (`***`Handle`***`, `***`DataPtr`***`);`
- `macro proc PalDataPortWrite (`***`Handle`***`, `***`Data`***`);`

PalDataPortRead () macro

`macro proc PalDataPortRead (`***`Handle`***`, `***`DataPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to a DataPort resource.
	`DataPtr`: pointer to an lvalue (e.g. variable or signal) of type `unsigned (PalDataPortGetMaxDataWidthCT ())`.
Timing	1 or more clock cycles.
Description	Reads a single item of data from the data port, and stores it in the lvalue pointed at by ***`DataPtr`***.

PalDataPortWrite () macro

`macro proc PalDataPortWrite (`***`Handle`***`, `***`Data`***`);`

Arguments	***`Handle`***: `PalHandle` to a DataPort resource.
	`Data`: expression of type `unsigned (PalDataPortGetMaxDataWidthCT ())`.
Timing	1 or more clock cycles.
Description	Writes a single item of data to the data port.

4.3.5 Parallel port API

These methods are used only for getting handles to parallel ports, which otherwise implement the DataPort API

- `macro expr PalParallelPort (`***`Index`***`);`

  see `macro expr Pal`***`X`***` (`***`Index`***`); (see page 21)`
- `macro expr PalParallelPortCT (`***`IndexCT`***`);`

  see `macro expr Pal`***`X`***`CT (`***`IndexCT`***`); (see page 21)`
- `macro expr PalParallelPortCount ();`

  see `macro expr Pal`***`X`***`Count (); (see page 22)`
- `macro expr PalParallelPortUniqueCount ();`

  see `macro expr Pal`***`X`***`UniqueCount (); (see page 22)`
- `macro proc PalParallelPortRequire (`***`Count`***`);`

  see `macro proc Pal`***`X`***`Require (`***`Count`***`); (see page 22)`

4.3.6 RS-232 serial ports (RS232Port API)

These methods are used only for getting handles to RS-232 serial ports, which otherwise implement the DataPort API.

- `macro expr PalRS232Port (`***`Index`***`);`

  see `macro expr Pal`***`X`***` (`***`Index`***`); (see page 21)`
- `macro expr PalRS232PortCT (`***`IndexCT`***`);`

see `macro expr Pal`***`X`***`CT (`***`IndexCT`***`);` `(see page 21)`

- `macro expr PalRS232PortCount ();`
  see `macro expr Pal`***`X`***`Count ();` `(see page 22)`
- `macro expr PalRS232PortUniqueCount ();`
  see `macro expr Pal`***`X`***`UniqueCount ();` `(see page 22)`
- `macro proc PalRS232PortRequire (`***`Count`***`);`
  see `macro proc Pal`***`X`***`Require (`***`Count`***`);` `(see page 22)`

4.3.7 PS/2 serial ports (PS2Port API)

These methods are only used for getting handles to PS/2 serial ports, which otherwise implement the DataPort API.

- `macro expr PalPS2Port (`***`Index`***`);`
  see `macro expr Pal`***`X`***` (`***`Index`***`);` `(see page 21)`
- `macro expr PalPS2PortCT (`***`IndexCT`***`);`
  see `macro expr Pal`***`X`***`CT (`***`IndexCT`***`);` `(see page 21)`
- `macro expr PalPS2PortCount ();`
  see `macro expr Pal`***`X`***`Count ();` `(see page 22)`
- `macro expr PalPS2PortUniqueCount ();`
  see `macro expr Pal`***`X`***`UniqueCount ();` `(see page 22)`
- `macro proc PalPS2PortRequire (`***`Count`***`);`
  see `macro proc Pal`***`X`***`Require (`***`Count`***`);` `(see page 22)`

4.3.8 Single cycle RAMs (FastRAM API)

Fast RAMs are single cycle RAMs that can be read from or written to in exactly one clock cycle, hence they are fast in terms of the clock cycles taken. However, FastRAMs often have a lower performance than pipelined PL1RAMs since they rely on a very short combinational path through the entire RAM access circuitry. The FastRAM API is suitable for standard asynchronous static RAMs.

- `macro expr PalFastRAM (`***`Index`***`);`
  see `macro expr Pal`***`X`***` (`***`Index`***`);` `(see page 21)`
- `macro expr PalFastRAMCT (`***`IndexCT`***`);`
  see `macro expr Pal`***`X`***`CT (`***`IndexCT`***`);` `(see page 21)`
- `macro expr PalFastRAMCount ();`
  see `macro expr Pal`***`X`***`Count ();` `(see page 22)`
- `macro expr PalFastRAMUniqueCount ();`
  see `macro expr Pal`***`X`***`UniqueCount ();` `(see page 22)`
- `macro proc PalFastRAMRequire (`***`Count`***`);`
  see `macro proc Pal`***`X`***`Require (`***`Count`***`);` `(see page 22)`

- `macro expr PalFastRAMGetMaxDataWidthCT ();`
  see `macro expr Pal`***`X`***`GetMaxDataWidthCT ();`
- `macro expr PalFastRAMGetMaxAddressWidthCT ();`
- `macro expr PalFastRAMGetDataWidth (`***`Handle`***`);`
  see `macro expr Pal`***`X`***`GetDataWidth (`***`Handle`***`);`
- `macro expr PalFastRAMGetAddressWidth (`***`Handle`***`);`
- `macro expr PalFastRAMGetDataWidthCT (`***`HandleCT`***`);`
  see `macro expr PalXGetDataWidthCT (`***`HandleCT`***`);`
- `macro expr PalFastRAMGetAddressWidthCT (`***`HandleCT`***`);`
- `macro proc PalFastRAMRun (`***`HandleCT`***`, `***`ClockRate`***`);`
  `see macro proc Pal`***`X`***`Run (`***`HandleCT`***`, `***`ClockRate`***`); (see page 23)`
- `macro proc PalFastRAMReset (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Reset (`***`Handle`***`);`
- `macro proc PalFastRAMEnable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Enable (`***`Handle`***`);`
- `macro proc PalFastRAMDisable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Disable (`***`Handle`***`);`
- `macro proc PalFastRAMRead (`***`Handle`***`, `***`Address`***`, `***`DataPtr`***`);`
- `macro proc PalFastRAMWrite (`***`Handle`***`, `***`Address`***`, `***`Data`***`);`

PalFastRAMGetMaxAddressWidthCT () macro

`macro expr PalFastRAMGetMaxAddressWidthCT ();`

Arguments	None.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the maximum width of the address bus of the RAM, at compile time. This is the width that should be used for the address parameter to the RAM read and write methods.

PalFastRAMGetAddressWidth () macro

`macro expr PalFastRAMGetAddressWidth ();`

Arguments	***`Handle`***: `PalHandle` to FastRAM resource.
Return value	Integer value, evaluated at runtime.
Description	Returns the actual width of the address bus of the RAM that is used by the handle referred to, at runtime. Since this method works at runtime it can be used even when ***`Handle`*** is not a constant (such as when it is passed through a function parameter). However, it cannot be used to set the width of variables.

PalFastRAMGetAddressWidthCT () macro

`macro expr PalFastRAMGetAddressWidthCT (`***`HandleCT`***`);`

Arguments	***`HandleCT`***: constant `PalHandle` to FastRAM resource.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the actual width of the address bus of the RAM that is used by the handle referred to, at compile time. Since this method works at compile time it can only be used where ***`Handle`*** is a constant, and can be deduced to be a constant by the DK compiler. So for instance, this cannot be used if ***`Handle`*** is passed through a function parameter. However, since it returns a compile-time constant it can be used to set variable widths and do conditional compilation (using `select` and `ifselect`).

PalFastRAMRead () macro

`macro proc PalFastRAMRead (`***`Handle`***`, `***`Address`***`, `***`DataPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to FastRAM resource.
	`Address`: Address of data to read, of type `unsigned (PalFastRAMGetMaxAddressWidthCT ())`.
	`DataPtr`: pointer to an lvalue (e.g. variable or signal) of type `unsigned (PalFastRAMGetMaxDataWidthCT ())`.
Timing	1 clock cycle.
Description	Reads a single item of data from the RAM at the specified address, and stores it in the lvalue pointed at by ***`DataPtr`***.

PalFastRAMWrite () macro

`macro proc PalFastRAMWrite (`***`Handle`***`, `***`Address`***`, `***`Data`***`);`

Arguments	***`Handle`***: `PalHandle` to FastRAM resource.
	`Address`: Address of data to be written, of type `unsigned (PalFastRAMGetMaxAddressWidthCT ())`.
	`Data`: expression of type `unsigned (PalFastRAMGetMaxDataWidthCT ())`.
Timing	1 clock cycle.
Description	Writes a single item of data to the RAM at the specified address.

4.3.9 Pipelined single-cycle RAMs (PL1RAM API)

PL1 RAMs are RAMs that can be read from or written to in exactly one clock cycle, but with the address supplied one cycle earlier (and hence pipelined). This is typically as fast way of accessing RAMs (in terms of combinational path length). The PL1RAM API is suitable for standard zero bus turnaround (ZBT) synchronous static RAM.

- `macro expr PalPL1RAM (`***`Index`***`);`
  see `macro expr Pal`***`X`***` (`***`Index`***`); (see page 21)`
- `macro expr PalPL1RAMCT (`***`IndexCT`***`);`
  see `macro expr Pal`***`X`***`CT (`***`IndexCT`***`); (see page 21)`
- `macro expr PalPL1RAMCount ();`
  see `macro expr Pal`***`X`***`Count (); (see page 22)`

- `macro expr PalPL1RAMUniqueCount ();`
  see `macro expr Pal`***`X`***`UniqueCount (); (see page 22)`
- `macro proc PalPL1RAMRequire (`***`Count`***`);`
  see `macro proc Pal`***`X`***`Require (`***`Count`***`); (see page 22)`
- `macro expr PalPL1RAMGetMaxDataWidthCT ();`
  see `macro expr Pal`***`X`***`GetMaxDataWidthCT ();`
- `macro expr PalPL1RAMGetMaxAddressWidthCT ();`
  see `macro expr PalFastRAMGetMaxAddressWidthCT ();`
- `macro expr PalPL1RAMGetDataWidth (`***`Handle`***`);`
  see `macro expr Pal`***`X`***`GetDataWidth (`***`Handle`***`);`
- `macro expr PalPL1GetAddressWidth (Handle);`
  see `macro expr PalFastRAMGetAddressWidth (`***`Handle`***`);`
- `macro expr PalPL1GetDataWidthCT (`***`HandleCT`***`);`
  see `macro expr Pal`***`X`***`GetDataWidthCT (`***`HandleCT`***`);`
- `macro expr PalPL1GetAddressWidthCT (HandleCT);`
  see `macro expr PalFastRAMGetAddressWidthCT (`***`HandleCT`***`);`
- `macro proc PalPL1Run (`***`HandleCT`***`, `***`ClockRate`***`);`
  `see macro proc Pal`***`X`***`Run (`***`HandleCT`***`, `***`ClockRate`***`); (see page 23)`
- `macro proc PalPL1Reset (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Reset (`***`Handle`***`);`
- `macro proc PalPL1Enable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Enable (`***`Handle`***`);`
- `macro proc PalPL1Disable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Disable (`***`Handle`***`);`
- `macro proc PalPL1RAMSetReadAddress (`***`Handle`***`, `***`Address`***`);`
- `macro proc PalPL1RAMSetWriteAddress (`***`Handle`***`, `***`Address`***`);`
- `macro proc PalPL1RAMRead (`***`Handle`***`, `***`DataPtr`***`);`
- `macro proc PalPL1RAMWrite (`***`Handle`***`, `***`Data`***`);`

PalPL1RAMSetReadAddress () macro

`macro proc PalPL1RAMSetReadAddress (`***`Handle`***`, `***`Address`***`);`

Arguments	***`Handle`***: `PalHandle` to a PL1RAM resource. ***`Address`***: Address of data to read, of type `unsigned (PalPL1RAMGetMaxAddressWidthCT ())`.
Timing	1 clock cycle.
Description	Sets the address for a read that will occur on the next clock cycle.

PalPL1RAMSetWriteAddress () macro

`macro proc PalPL1RAMSetWriteAddress (`***`Handle`***`, `***`Address`***`);`

Arguments	***`Handle`***: `PalHandle` to a PL1RAM resource.
	`Address`: Address of data to read, of type `unsigned (PalPL1RAMGetMaxAddressWidthCT ())`.
Timing	1 clock cycle.
Description	Sets the address for a write that will occur on the next clock cycle.

PalPL1RAMRead() macro

`macro proc PalPL1RAMRead (`***`Handle`***`, `***`DataPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to a PL1RAM resource.
	`DataPtr`: pointer to an lvalue (e.g. variable or signal) of type `unsigned (PalPL1RAMGetMaxWidthCT())`.
Timing	1 clock cycle.
Description	Reads a single item of data from the address in the RAM set by `PalPL1RAMSetReadAddress()` on the previous clock cycle and stores it in the lvalue pointed at by ***`DataPtr`***.

PalPL1RAMWrite() macro

`macro proc PalPL1RAMWrite (`***`Handle`***`, `***`Data`***`);`

Arguments	***`Handle`***: `PalHandle` to a PL1RAM resource.
	`Data`: expression of type `unsigned (PalPL1RAMGetMaxWidthCT())`.
Timing	1 clock cycle.
Description	Writes a single item of data to the RAM at the address set by `PalPL1RAMSetWriteAddress()` on the previous clock cycle.

4.3.10 Pipelined PL2 RAMs

PL2 RAMs are similar to PL1 RAMs but the address must be set exactly two clock cycles before reading or writing data. The RAMs can be read from or written to in exactly one clock cycle, but with the address supplied two clock cycles earlier. This is typically a fast way of accessing RAMs.

Some platforms, such as the ADM-XRC-II (RC2000) do not support PL1 RAMs. However, platforms which do support PL1 RAMs also support PL2 RAMs, so you can make your code more portable by using PL2 RAMs.

- `macro expr PalPL2RAM (`***`Index`***`);`
  see `macro expr Pal`***`X`***` (`***`Index`***`); (see page 21)`
- `macro expr PalPL2RAMCT (`***`IndexCT`***`);`
  see `macro expr Pal`***`X`***`CT (`***`IndexCT`***`); (see page 21)`
- `macro expr PalPL2RAMCount ();`
  see `macro expr Pal`***`X`***`Count (); (see page 22)`
- `macro expr PalPL2RAMUniqueCount ();`

see `macro expr Pal`***X***`UniqueCount ();` (see page 22)

- `macro proc PalPL2RAMRequire (`***Count***`);`
  see `macro proc Pal`***X***`Require (`***Count***`);` (see page 22)
- `macro expr PalPL2RAMGetMaxDataWidthCT ();`
  see `macro expr Pal`***X***`GetMaxDataWidthCT ();`
- `macro expr PalPL2RAMGetMaxAddressWidthCT ();`
  see `macro expr PalFastRAMGetMaxAddressWidthCT ();`
- `macro expr PalPL2RAMGetDataWidth (`***Handle***`);`
  see `macro expr Pal`***X***`GetDataWidth (`***Handle***`);`
- `macro expr PalPL2GetAddressWidth (Handle);`
  see `macro expr PalFastRAMGetAddressWidth (`***Handle***`);`
- `macro expr PalPL2GetDataWidthCT (`***HandleCT***`);`
  see `macro expr Pal`***X***`GetDataWidthCT (`***HandleCT***`);`
- `macro expr PalPL2GetAddressWidthCT (HandleCT);`
  see `macro expr PalFastRAMGetAddressWidthCT (`***HandleCT***`);`
- `macro proc PalPL2Run (`***HandleCT***`, `***ClockRate***`);`
  see `macro proc Pal`***X***`Run (`***HandleCT***`, `***ClockRate***`);` (see page 23)
- `macro proc PalPL2Reset (`***Handle***`);`
  see `macro proc Pal`***X***`Reset (`***Handle***`);`
- `macro proc PalPL2Enable (`***Handle***`);`
  see `macro proc Pal`***X***`Enable (`***Handle***`);`
- `macro proc PalPL2Disable (`***Handle***`);`
  see `macro proc Pal`***X***`Disable (`***Handle***`);`
- `macro proc PalPL2RAMSetReadAddress (Handle, Address);`
- `macro proc PalPL2RAMSetWriteAddress (`***Handle***`, `***Address***`);`
- `macro proc PalPL1RAMRead (`***Handle***`, `***DataPtr***`);`
- `macro proc PalPL1RAMWrite (`***Handle***`, `***Data***`);`

PalPL2RAMSetReadAddress () macro

`macro proc PalPL2RAMSetReadAddress (`***Handle***`, `***Address***`);`

Arguments	***`Handle`***: `PalHandle` to a PL2RAM resource.
	`Address`: Address of data to read, of type `unsigned (PalPL2RAMGetMaxAddressWidthCT ())`.
Timing	1 clock cycle.
Description	Sets the address for a read that will occur two clock cycles later. For example:

```
seq
{
    PalPL2RAMSetReadAddress (Handle, Addr);
    delay;
    PalPL2RAMRead (Handle, &Data);
}
```

PalPL2RAMSetWriteAddress () macro

`macro proc PalPL2RAMSetWriteAddress (`***`Handle`***`, `***`Address`***`);`

Arguments	***`Handle`***: `PalHandle` to a PL2RAM resource.
	`Address`: Address of data to read, of type `unsigned (PalPL2RAMGetMaxAddressWidthCT ())`.
Timing	1 clock cycle.
Description	Sets the address for a write that will occur two clock cycles later.

PalPL2RAMRead () macro

`macro proc PalPL2RAMRead (`***`Handle`***`, `***`DataPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to a PL2RAM resource.
	`DataPtr`: Pointer to a lvalue (e.g. variable or signal) of type `unsigned (PalPL2RAMGetMaxDataWidthCT())`.
Timing	1 clock cycle.
Description	Reads a single item of data from the address in the RAM set by `PalPL2RAMSetReadAddress()` two clock cycles earlier, and stores it in the lvalue pointed at by ***`DataPtr`***.

PalPL2RAMWrite () macro

`macro proc PalPL2RAWrite (`***`Handle`***`, `***`Data`***`);`

Arguments	***`Handle`***: `PalHandle` to a PL2RAM resource.
	`Data`: expression of type `unsigned (PalPL2RAMGetMaxDataWidthCT ())`.
Timing	1 clock cycle.
Description	Writes a single item of data to the RAM at the address set by `PalPL2RAMSetWriteAddress` two clock cycles earlier.

4.3.11 RAMs without guaranteed timing (SlowRAM API)

Slow RAMs are RAMs that can be read from or written to in multiple clock cycles. This is the most generic and platform-independent interface to RAMs and other random access storage devices. This

API is suitable for accessing shared RAMs arbitrated by some external circuitry, or for accessing RAMs which cannot guarantee timing due to refresh constraints (such as DRAM).

- `macro expr PalSlowRAM (`***`Index`***`);`
  see `macro expr Pal`***`X`***` (`***`Index`***`); (see page 21)`
- `macro expr PalSlowRAMCT (`***`IndexCT`***`);`
  see `macro expr Pal`***`X`***`CT (`***`IndexCT`***`); (see page 21)`
- `macro expr PalSlowRAMCount ();`
  see `macro expr Pal`***`X`***`Count (); (see page 22)`
- `macro expr PalSlowRAMUniqueCount ();`
  see `macro expr Pal`***`X`***`UniqueCount (); (see page 22)`
- `macro proc PalSlowRAMRequire (`***`Count`***`);`
  see `macro proc Pal`***`X`***`Require (`***`Count`***`); (see page 22)`
- `macro expr PalSlowRAMGetMaxDataWidthCT ();`
  see `macro expr Pal`***`X`***`GetMaxDataWidthCT ();`
- `macro expr PalSlowRAMGetMaxAddressWidthCT ();`
  see `macro expr PalFastRAMGetMaxAddressWidthCT ();`
- `macro expr PalSlowRAMGetDataWidth (`***`Handle`***`);`
  see `macro expr Pal`***`X`***`GetDataWidth (`***`Handle`***`);`
- `macro expr PalSlowGetAddressWidth (Handle);`
  see `macro expr PalFastRAMGetAddressWidth (`***`Handle`***`);`
- `macro expr PalSlowGetDataWidthCT (`***`HandleCT`***`);`
  see `macro expr Pal`***`X`***`GetDataWidthCT (`***`HandleCT`***`);`
- `macro expr PalSlowGetAddressWidthCT (HandleCT);`
  see `macro expr PalFastRAMGetAddressWidthCT (`***`HandleCT`***`);`
- `macro proc PalSlowRun (`***`HandleCT`***`, `***`ClockRate`***`);`
  `see macro proc Pal`***`X`***`Run (`***`HandleCT`***`, `***`ClockRate`***`); (see page 23)`
- `macro proc PalSlowReset (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Reset (`***`Handle`***`);`
- `macro proc PalSlowEnable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Enable (`***`Handle`***`);`
- `macro proc PalSlowDisable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Disable (`***`Handle`***`);`
- `macro proc PalSlowRAMRead (`***`Handle`***`, `***`Address`***`, `***`DataPtr`***`);`
- `macro proc PalSlowRAMWrite (`***`Handle`***`, Address, Data);`

PalSlowRAMRead () macro

`macro proc PalSlowRAMRead (`***`Handle`***`, `***`Address`***`, `***`DataPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to a SlowRAM resource.
	`Address`: Address of data to read, of type `unsigned (PalSlowRAMGetMaxAddressWidthCT ())`.
	`DataPtr`: pointer to an lvalue (e.g. variable or signal) of type `unsigned (PalSlowRAMGetMaxDataWidthCT ())`.
Timing	1 or more clock cycles.
Description	Reads a single item of data from the RAM at the specified address, and stores it in the lvalue pointed at by ***`DataPtr`***.

PalSlowRAMWrite () macro

`macro proc PalSlowRAMWrite (`***`Handle`***`, `***`Address`***`, `***`Data`***`);`

Arguments	***`Handle`***: `PalHandle` to SlowRAM resource.
	`Address`: Address of data to be written, of type `unsigned (PalSlowRAMGetMaxAddressWidthCT ())`.
	`Data`: expression of type `unsigned (PalSlowRAMGetMaxDataWidthCT ())`.
Timing	1 or more clock cycles.
Description	Writes a single item of data to the RAM at the specified address.

4.3.12 Block storage devices (BlockStore API)

PalBlockStore is typically used by devices that have block oriented access, and in particular require the user to erase a block of one or more locations before writing new data, such as Flash memory. The address space is treated as being the contiguous concatenation of all of the blocks, i.e. address location 0 is first entry in block 0, address location `(PalBlockStoreGetBlockLength (`***`Handle`***`))` is the first entry in block 1.

- `macro expr PalBlockStore (`***`Index`***`);`

  see `macro expr Pal`***`X`***` (`***`Index`***`); (see page 21)`
- `macro expr PalBlockStoreCT (`***`Index`***`);`

  see `macro expr Pal`***`X`***`CT (`***`Index`***`); (see page 21)`
- `macro expr PalBlockStoreCount ();`

  see `macro expr Pal`***`X`***`Count (); (see page 22)`
- `macro expr PalBlockStoreUniqueCount ();`

  see `macro expr Pal`***`X`***`UniqueCount (); (see page 22)`
- `macro proc PalBlockStoreRequire (`***`Count`***`);`

  see `macro proc Pal`***`X`***`Require (`***`Count`***`); (see page 22)`
- `macro expr PalBlockStoreGetMaxDataWidthCT ();`

  see `macro expr Pal`***`X`***`GetMaxDataWidthCT ();`
- `macro expr PalBlockStoreGetMaxAddressWidthCT ();`

  see `macro expr PalFastRAMGetMaxAddressWidthCT ();`
- `macro expr PalBlockStoreGetMaxBlockLengthCT ();`

- `macro expr PalBlockStoreGetDataWidth (`***`Handle`***`);`
  see `macro expr Pal`***`X`***`GetDataWidth (`***`Handle`***`);`
- `macro expr PalBlockStoreGetAddressWidth (`***`Handle`***`);`
  see `macro expr PalFastRAMGetAddressWidth (`***`Handle`***`);`
- `macro expr PalBlockStoreGetBlockLength (`***`Handle`***`);`
- `macro expr PalBlockStoreGetDataWidthCT (`***`HandleCT`***`);`
  see `macro expr Pal`***`X`***`GetDataWidthCT (`***`HandleCT`***`);`
- `macro expr PalBlockStoreGetAddressWidthCT (`***`HandleCT`***`);`
  see `macro expr PalFastRAMGetAddressWidthCT (`***`HandleCT`***`);`
- `macro expr PalBlockStoreGetBlockLengthCT (`***`HandleCT`***`);`
- `macro proc PalBlockStoreRun (`***`HandleCT`***`, `***`ClockRate`***`);`
  see `macro proc Pal`***`X`***`Run (`***`HandleCT`***`, `***`ClockRate`***`); (see page 23)`
- `macro proc PalBlockStoreReset (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Reset (`***`Handle`***`);`
- `macro proc PalBlockStoreEnable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Enable (`***`Handle`***`);`
- `macro proc PalBlockStoreDisable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Disable (`***`Handle`***`);`
- `macro proc PalBlockStoreRead (`***`Handle`***`, `***`Address`***`, `***`DataPtr`***`);`
- `macro proc PalBlockStoreWrite (`***`Handle`***`, `***`Address`***`, `***`Data`***`);`
- `macro proc PalBlockStoreEraseBlock (`***`Handle`***`, `***`BlockNumber`***`);`

PalBlockStoreGetMaxBlockLengthCT () macro

`macro expr PalBlockStoreGetMaxLengthCT ();`

Arguments	None.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the maximum length of a block for all of the block store devices on the platform, at compile time. This is typically, but not always, a power of two.

PalBlockStoreGetBlockLength () macro

`macro expr PalBlockStoreGetLength (`***`Handle`***`);`

Arguments	***`Handle`***: `PalHandle` to a BlockStore resource.
Return value	Integer value evaluated at runtime.
Description	Returns the actual length of a block for the block store device used by the handle referred to, at runtime. Since this method works at runtime it can be used even when ***`Handle`*** is not a constant (such as when it is passed through a function parameter). However, it cannot be used to set the width of variables.

PalBlockStoreGetBlockLengthCT () macro

`macro expr PalBlockStoreGetLengthCT (`***`HandleCT`***`);`

Arguments	***`HandleCT`***: constant `PalHandle` to a BlockStore resource.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the actual length of a block for the block storage device that is used by the handle referred to, at compile time. Since this method works at compile time it can only be used where ***`Handle`*** is a constant, and can be deduced to be a constant by the DK compiler. So for instance, this cannot be used if ***`Handle`*** is passed through a function parameter. However, since it returns a compile time constant it can be used to set variable widths and do conditional compilation (using `select` and `ifselect`).

PalBlockStoreRead () macro

`macro proc PalBlockStoreRead (`***`Handle`***`, `***`Address`***`, `***`DataPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to a BlockStore resource. ***`Address`***: Address of data to read, of type `unsigned (PalBlockStoreGetMaxAddressWidthCT ())`. ***`DataPtr`***: pointer to an lvalue (e.g. variable or signal) of type `unsigned (PalBlockStoreGetMaxDataWidthCT ())`.
Timing	1 or more clock cycles.
Description	Reads a single item of data from the block store device at the specified address, and stores it in the lvalue pointed at by ***`DataPtr`***.

PalBlockStoreWrite () macro

`macro proc PalBlockStoreWrite (`***`Handle`***`, `***`Address`***`, `***`Data`***`);`

Arguments	***`Handle`***: `PalHandle` to a BlockStore resource. ***`Address`***: Address of data to be written, of type `unsigned (PalBlockStoreGetMaxAddressWidthCT ())`. ***`Data`***: expression of type `unsigned (PalBlockStoreGetMaxDataWidthCT ())`.
Timing	1 or more clock cycle.
Description	Writes a single item of data to the block store device at the specified address.

PalBlockStoreEraseBlock () macro

`macro proc PalBlockStoreErase (`***`Handle`***`, `***`BlockNumber`***`);`

Arguments	***`Handle`***: `PalHandle` to a BlockStore resource. ***`BlockNumber`***: Index of the block to erase, of type `unsigned (PalBlockStoreGetMaxAddressWidthCT ()- log2floor (PalBlockStoreGetMaxBlockLengthCT))`.
Timing	1 or more clock cycles.
Description	Statement that erases a complete block indexed by ***`BlockNumber`*** from the block store device used by ***`Handle`*** to prepare it for re-writing.

4.3.13 Video capture devices (VideoIn API)

The VideoIn API supports generic video capture devices.

- `macro expr PalVideoIn (`***`Index`***`);`
  see `macro expr Pal`***`X`***` (`***`Index`***`); (see page 21)`
- `macro expr PalVideoInCT (`***`Index`***`);`
  see `macro expr Pal`***`X`***`CT (`***`Index`***`); (see page 21)`
- `macro expr PalVideoInCount ();`
  see `macro expr Pal`***`X`***`Count (); (see page 22)`
- `macro expr PalVideoInUniqueCount ();`
  see `macro expr Pal`***`X`***`UniqueCount (); (see page 22)`
- `macro proc PalVideoInRequire (`***`Count`***`);`
  see `macro proc Pal`***`X`***`Require (`***`Count`***`); (see page 22)`
- `macro expr PalVideoInGetMaxXWidthCT ();`
- `macro expr PalVideoInGetMaxYWidthCT ();`
- `macro expr PalVideoInGetMaxColorWidthCT ();`
- `macro expr PalVideoInGetXWidth (`***`Handle`***`);`
- `macro expr PalVideoInGetYWidth (`***`Handle`***`);`
- `macro expr PalVideoInGetColorWidth (`***`Handle`***`);`
- `macro expr PalVideoInGetXWidthCT (`***`HandleCT`***`);`
- `macro expr PalVideoInGetYWidthCT (`***`HandleCT`***`);`
- `macro expr PalVideoInGetXColorWidthCT (`***`HandleCT`***`);`
- `macro proc PalVideoInRun (`***`HandleCT`***`, `***`ClockRate`***`);`
  see `macro proc Pal`***`X`***`Run (`***`HandleCT`***`, `***`ClockRate`***`); (see page 23)`
- `macro proc PalVideoInReset (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Reset (`***`Handle`***`);`
- `macro proc PalVideoInEnable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Enable (`***`Handle`***`);`
- `macro proc PalVideoInDisable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Disable (`***`Handle`***`);`
- `macro proc PalVideoInRead (`***`Handle`***`, `***`XPtr`***`, `***`YPtr`***`, `***`PixelPtr`***`);`

PalVideoInGetMaxXWidthCT () macro

```
macro expr PalVideoInGetMaxXWidthCT ();
```

Arguments	None.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the maximum width of the X co-ordinate of all of the video input devices on the platform, at compile time. This is the width that should be used for the lvalue that is pointed to by ***`XPtr`*** in the `PalVideoInRead` method.

PalVideoInGetMaxYWidthCT () macro

`macro expr PalVideoInGetMaxYWidthCT ();`

Arguments	None.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the maximum width of the Y co-ordinate of all of the video input devices on the platform, at compile time. This is the width that should be used for the lvalue that is pointed to by ***`YPtr`*** in the `PalVideoInRead` method.

PalVideoInGetMaxColorWidthCT () macro

`macro expr PalVideoInGetMaxYColorWidthCT ();`

Arguments	None.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the maximum width of the pixel colour of all of the video input devices on the platform, at compile-time. This is the width that should be used for the lvalue that is pointed to by ***`PixelPtr`*** in the `PalVideoInRead` method.

PalVideoInGetXWidth () macro

`macro expr PalVideoInGetXWidth (`***`Handle`***`);`

Arguments	***`Handle`***: `PalHandle` to a VideoIn resource.
Return value	Integer value evaluated at runtime.
Description	Returns the actual width of the X co-ordinate of the video input device used by the handle referred to, at runtime. Since this method works at runtime it can be used even when ***`Handle`*** is not a constant (such as when it is passed through a function parameter). However, it cannot be used to set the width of variables.

PalVideoInGetYWidth () macro

`macro expr PalVideoInGetYWidth (`***`Handle`***`);`

Arguments	***`Handle`***: `PalHandle` to a VideoIn resource.
Return value	Integer value evaluated at runtime.
Description	Returns the actual width of the Y co-ordinate of the video input device used by the handle referred to, at runtime. Since this method works at runtime it can be used even when ***`Handle`*** is not a constant (such as when it is passed through a function parameter). However, it cannot be used to set the width of variables.

PalVideoInGetColorWidth () macro

`macro expr PalVideoInGetColorWidth (`***`Handle`***`);`

Arguments	***`Handle`***: `PalHandle` to a VideoIn resource.
Return value	Integer value evaluated at runtime.
Description	Returns the actual width of the pixel colour of the video input device used by the handle referred to, at runtime. Since this method works at runtime it can be used even when ***`Handle`*** is not a constant (such as when it is passed through a function parameter). However, it cannot be used to set the width of variables.

PalVideoInGetXWidthCT () macro

```
macro expr PalVideoInGetXWidthCT (HandleCT);
```

Arguments	***`HandleCT`***: constant `PalHandle` to a VideoIn resource.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the actual width of the X co-ordinate of the video input device that is used by the handle referred to, at compile-time. Since this method works at compile time it can only be used where ***`HandleCT`*** is a constant, and can be deduced to be a constant by the DK compiler. So for instance, this cannot be used if ***`HandleCT`*** is passed through a function parameter. However, since it returns a compile time constant it can be used to set variable widths and do conditional compilation (using `select` and `ifselect`).

PalVideoInGetYWidthCT () macro

```
macro expr PalVideoInGetYWidthCT (HandleCT);
```

Arguments	***`HandleCT`***: constant `PalHandle` to a VideoIn resource.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the actual width of the Y co-ordinate of the video input device that is used by the handle referred to, at compile-time. Since this method works at compile time it can *only* be used where ***`HandleCT`*** is a constant, and can be deduced to be a constant by the DK compiler. So for instance, this cannot be used if ***`HandleCT`*** is passed through a function parameter. However, since it returns a compile time constant it can be used to set variable widths and do conditional compilation (using `select` and `ifselect`)

PalVideoInGetColorWidthCT () macro

```
macro expr PalVideoInGetColorWidthCT (HandleCT);
```

Arguments	***`HandleCT`***: constant `PalHandle` to a VideoIn resource.
Return value	Compile-time constant `unsigned int` for specifying a width.
Description	Returns at the actual width of the pixel color of the video input device that is used by the handle referred to, compile-time. Since this method works at compile time it can *only* be used where ***`HandleCT`*** is a constant, and can be deduced to be a constant by the DK compiler. So for instance, this cannot be used if ***`HandleCT`*** is passed through a function parameter. However, since it returns a compile time constant it can be used to set variable widths and do conditional compilation (using `select` and `ifselect`)

PalVideoInRead () macro

`macro proc PalVideoInRead (`***`Handle`***`, `***`XPtr`***`, `***`YPtr`***`, `***`PixelPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to a VideoIn resource. ***`XPtr`***: Pointer to an lvalue, of type `unsigned (PalVideoInGetMaxXWidthCT ())`. ***`YPtr`***: Pointer to an lvalue, of type `unsigned (PalVideoInGetMaxYWidthCT ())`. ***`PixelPtr`***: pointer to an lvalue (e.g. variable or signal) of type `unsigned (PalVideoInGetMaxColorWidthCT ())`.
Timing	1 or more clock cycles.
Description	Reads a single pixel from the video input device. The call sets both the color value in ***`PixelPtr`*** and the X and Y coordinates of the captured pixel in ***`XPtr`*** and ***`YPtr`***. Frames may be interlaced. This function needs to be called repeatedly without delay in order to be sure of not missing pixels. The video format is typically 888 RGB, giving a width of 24.

4.3.14 Video output devices (VideoOut API)

The VideoOut API supports generic progressive scan video output, such as VGA.

- `macro expr PalVideoOut (`***`Index`***`);`

  see `macro expr Pal`***`X`***` (`***`Index`***`); (see page 21)`
- `macro expr PalVideoOutCT (`***`Index`***`);`

  see `macro expr Pal`***`X`***`CT (`***`Index`***`); (see page 21)`
- `macro expr PalVideoOutCount ();`

  see `macro expr Pal`***`X`***`Count (); (see page 22)`
- `macro expr PalVideoOutUniqueCount ();`

  see `macro expr Pal`***`X`***`UniqueCount (); (see page 22)`
- `macro proc PalVideoOutRequire (`***`Count`***`);`

  see `macro proc Pal`***`X`***`Require (`***`Count`***`); (see page 22)`
- `macro expr PalVideoOutOptimalCT (`***`ClockRate`***`);`
- `macro expr PalVideoOutGetMaxXWidthCT ();`

  see `macro expr PalVideoInGetMaxXWidthCT ();`
- `macro expr PalVideoOutGetMaxYWidthCT ();`

  see `macro expr PalVideoInGetMaxYWidthCT ();`
- `macro expr PalVideoOutGetMaxColorWidthCT ();`

  see `macro expr PalVideoInGetMaxColorWidthCT ();`
- `macro expr PalVideoOutGetXWidth (`***`Handle`***`);`

  see `macro expr PalVideoInGetXWidth (`***`Handle`***`);`
- `macro expr PalVideoOutGetYWidth (`***`Handle`***`);`

  see `macro expr PalVideoInGetYWidth (`***`Handle`***`);`

- `macro expr PalVideoOutGetColorWidth (`***`Handle`***`);`
  see `macro expr PalVideoInGetColorWidth (`***`Handle`***`);`
- `macro expr PalVideoOutGetXWidthCT (`***`HandleCT`***`);`
  see `macro expr PalVideoInGetXWidthCT (`***`HandleCT`***`);`
- `macro expr PalVideoOutGetYWidthCT (`***`HandleCT`***`);`
  see `macro expr PalVideoInGetYWidthCT (`***`HandleCT`***`);`
- `macro expr PalVideoOutGetXColorWidthCT (`***`HandleCT`***`);`
  see `macro expr PalVideoInGetXColorWidthCT (`***`HandleCT`***`);`
- `macro expr PalVideoOutGetVisibleX (`***`Handle`***`, `***`ClockRate`***`);`
- `macro expr PalVideoOutGetVisibleY (`***`Handle`***`);`
- `macro expr PalVideoOutGetTotalX (`***`Handle`***`, `***`ClockRate`***`);`
- `macro expr PalVideoOutGetTotalY (`***`Handle`***`);`
- `macro expr PalVideoOutGetVisibleXCT (`***`Handle`***`, `***`ClockRate`***`);`
- `macro expr PalVideoOutGetVisibleYCT (`***`Handle`***`);`
- `macro expr PalVideoOutGetTotalXCT (`***`Handle`***`, `***`ClockRate`***`);`
- `macro expr PalVideoOutGetTotalYCT (`***`Handle`***`);`
- `macro proc PalVideoOutRun (`***`HandleCT`***`, `***`ClockRate`***`);`
  see `macro proc Pal`***`X`***`Run (`***`HandleCT`***`, `***`ClockRate`***`); (see page 23)`
- `macro proc PalVideoOutReset (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Reset (`***`Handle`***`);`
- `macro proc PalVideoOutEnable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Enable (`***`Handle`***`);`
- `macro proc PalVideoOutDisable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Disable(`***`Handle`***`);`
- `macro expr PalVideoOutGetX (`***`Handle`***`);`
- `macro expr PalVideoOutGetY (`***`Handle`***`);`
- `macro expr PalVideoOutGetHBlank (`***`Handle`***`);`
- `macro expr PalVideoOutGetVBlank (`***`Handle`***`);`
- `macro proc PalVideoOutWrite (`***`Handle`***`, `***`Pixel`***`);`

PalVideoOutOptimalCT () macro

`macro expr PalVideoOutOptimalCT (`***`ClockRate`***`);`

Arguments	***`ClockRate`***: The frequency of the clock in the clock domain of the call to `PalVideoOutRun()`, in units of *Hertz*.
Return value	Constant `PalHandle` evaluated at compile time.
Description	Returns the optimal handle to a display for a given clock frequency, at compile time. Typically this will be the video mode that results in output closest to the standard 4:3 ratio, and therefore the most square pixels.

PalVideoOutGetVisibleX () macro

```
macro expr PalVideoOutGetVisibleX (Handle, ClockRate);
```

Arguments	***`Handle`***: `PalHandle` to a VideoOut resource.
	`ClockRate`: The frequency of the clock in the clock domain of the call to `PalVideoOutRun()`, in units of Hertz.
Return value	Integer value evaluated at runtime.
Description	Returns the actual horizontal resolution of the video output device used by the handle referred to, at runtime. Since this method works at runtime it can be used even when ***`Handle`*** is not a constant (such as when it is passed through a function parameter). However, it cannot be used to set the width of variables.

PalVideoOutGetVisibleY () macro

```
macro expr PalVideoOutGetVisibleXY (Handle);
```

Arguments	***`Handle`***: `PalHandle` to a VideoOut resource.
Return value	Integer value evaluated at runtime.
Description	Returns the actual vertical resolution of the video output device used by the handle referred to, at runtime. Since this method works at runtime it can be used even when ***`Handle`*** is not a constant (such as when it is passed through a function parameter). However, it cannot be used to set the width of variables.

PalVideoOutGetTotalX () macro

```
macro expr PalVideoOutGetTotalX (Handle, ClockRate);
```

Arguments	***`Handle`***: `PalHandle` to a VideoOut resource.
	`ClockRate`: The frequency of the clock in the clock domain of the call to `PalVideoOutRun()`. In units of Hertz.
Return value	Integer value evaluated at runtime.
Description	Returns the actual total number of columns (including the blanking period) of the video output device used by the handle referred to, at runtime. Since this method works at runtime it can be used even when ***`Handle`*** is not a constant (such as when it is passed through a function parameter). However, it cannot be used to set the width of variables.

PalVideoOutGetTotalY () macro

```
macro expr PalVideoOutGetTotalY (Handle);
```

Arguments	***`Handle`***: `PalHandle` to a VideoOut resource.
Return value	Integer value evaluated at runtime.
Description	Returns the actual total number of rows (including the blanking period) of the video output device used by the handle referred to, at runtime. Since this method works at runtime it can be used even when ***`Handle`*** is not a constant (such as when it is passed through a function parameter). However, it cannot be used to set the width of variables.

PalVideoOutGetVisibleXCT () macro

`macro expr PalVideoOutGetVisibleXCT (`***`HandleCT`***`, `***`ClockRate`***`);`

Arguments	***`HandleCT`***: constant `PalHandle` to a VideoOut resource.
	`ClockRate`: The frequency of the clock in the clock domain of the call to `PalVideoOutRun()`. In units of Hertz.
Return value	Integer compile-time constant.
Description	Returns the actual horizontal resolution of the video output device that is used by the handle referred to, at compile time. Since this method works at compile time it can *only* be used where ***`HandleCT`*** is a constant, and can be deduced to be a constant by the DK compiler. So for instance, this cannot be used if ***`HandleCT`*** is passed through a function parameter. However, since it returns a compile time constant it can be used to set variable widths and do conditional compilation (using `select` and `ifselect`).

PalVideoOutGetVisibleYCT () macro

`macro expr PalVideoOutGetVisibleCT (`***`HandleCT`***`);`

Arguments	***`HandleCT`***: constant `PalHandle` to a VideoOut resource
Return value	Integer compile-time constant.
Description	Returns the actual vertical resolution of the video output device that is used by the handle referred to, at compile time. Since this method works at compile time it can only be used where ***`HandleCT`*** is a constant, and can be deduced to be a constant by the DK compiler. So for instance, this cannot be used if ***`HandleCT`*** is passed through a function parameter. However, since it returns a compile time constant it can be used to set variable widths and do conditional compilation (using `select` and `ifselect`)

PalVideoOutGetTotalXCT () macro

`macro expr PalVideoOutGetTotalXCT (`***`HandleCT`***`, `***`ClockRate`***`);`

Arguments	***`HandleCT`***: constant `PalHandle` to a VideoOut resource.
	`ClockRate`: The frequency of the clock in the clock domain of the call to `PalVideoOutRun`. In units of Hertz.
Return value	Integer compile-time constant.
Description	Returns the actual total number of columns (including blanking period) of the video output device that is used by the handle referred to, at compile time. Since this method works at compile time it can *only* be used where ***`HandleCT`*** is a constant, and can be deduced to be a constant by the DK compiler. So for instance, this cannot be used if ***`HandleCT`*** is passed through a function parameter. However, since it returns a compile time constant it can be used to set variable widths and do conditional compilation (using `select` and `ifselect`)

PalVideoOutGetTotalYCT () macro

`macro expr PalVideoOutGetTotalYCT (`***`HandleCT`***`);`

Arguments	***HandleCT***: constant `PalHandle` to a VideoOut resource.
Return value	Integer compile-time constant.
Description	Returns the actual total number of rows (including blanking period) of the video output device that is used by the handle referred to, at compile time. Since this method works at compile time it can only be used where ***HandleCT*** is a constant, and can be deduced to be a constant by the DK compiler. So for instance, this cannot be used if ***HandleCT*** is passed through a function parameter. However, since it returns a compile time constant it can be used to set variable widths and do conditional compilation (using `select` and `ifselect`)

PalVideoOutGetX () macro

```
macro expr PalVideoOutGetX (Handle);
```

Arguments	***Handle***: `PalHandle` to a VideoOut resource.
Return value	`unsigned int (PalVideoOutGetMaxXWidthCT ())`
Description	Returns the current horizontal scan position of the screen output. A call to `PalVideoOutWrite()` will write a pixel to position on the video output device returned by this method and `PalVideoOutGetY()`.

PalVideoOutGetY () macro

```
macro expr PalVideoOutGetY (Handle);
```

Arguments	***Handle***: `PalHandle` to a VideoOut resource.
Return value	`unsigned int (PalVideoOutGetMaxYWidthCT ())`.
Description	Returns the current horizontal scan position of the screen output. A call to `PalVideoOutWrite()` will write a pixel to position on the video output device returned by this method and `PalVideoOutGetX()`.

PalVideoOutGetHBlank () macro

```
macro expr PalVideoOutGetHBlank (Handle);
```

Arguments	***Handle***: `PalHandle` to a VideoOut resource.
Return value	`unsigned 1`.
Description	Returns the horizontal blanking status of the current scan position. A return value of 1 means the scan position is inside the blanking portion of the display. A return value of 0 means the scan position is inside the visible portion of the display.

PalVideoOutGetVBlank () macro

```
macro expr PalVideoOutGetVBlank (Handle);
```

Arguments	***`Handle`***: `PalHandle` to a VideoOut resource.
Return value	`unsigned 1`.
Description	Returns the vertical blanking status of the current scan position. A return value of 1 means the scan position is inside the blanking portion of the display. A return value of 0 means the scan position is inside the visible portion of the display.

PalVideoOutWrite () macro

`macro proc PalVideoOutWrite (`***`Handle`***`, `***`Pixel`***`);`

Arguments	***`Handle`***: `PalHandle` to a VideoOut resource. ***`Pixel`***: An expression of type `unsigned (PalVideoOutGetMaxColorWidthCT ())`.
Timing	1 clock cycle.
Description	Writes a single pixel value to the video output device at the current scan position, which can be obtained using the methods `PalVideoOutGetX()` and `PalVideoOutGetY()`.

4.3.15 Audio input devices (AudioIn API)

The AudioIn API supports generic audio input devices.

- `macro expr PalAudioIn (`***`Index`***`);`
  see `macro expr Pal`***`X`***` (`***`Index`***`); (see page 21)`
- `macro expr PalAudioInCT (`***`Index`***`);`
  see `macro expr Pal`***`X`***`CT (`***`Index`***`); (see page 21)`
- `macro expr PalAudioInCount ();`
  see `macro expr Pal`***`X`***`Count (); (see page 22)`
- `macro expr PalAudioInUniqueCount ();`
  see `macro expr Pal`***`X`***`UniqueCount (); (see page 22)`
- `macro proc PalAudioInRequire (`***`Count`***`);`
  see `macro proc Pal`***`X`***`Require (`***`Count`***`); (see page 22)`
- `macro expr PalAudioInGetMaxDataWidthCT ();`
  see `macro expr Pal`***`X`***`GetMaxDataWidthCT ();`
- `macro expr PalAudioInGetDataWidth (`***`Handle`***`);`
  see `macro expr Pal`***`X`***`GetDataWidth (`***`Handle`***`);`
- `macro expr PalAudioInGetDataWidthCT (`***`HandleCT`***`);`
  see `macro expr Pal`***`X`***`GetDataWidthCT (`***`HandleCT`***`);`
- `macro proc PalAudioInRun (`***`HandleCT`***`, `***`ClockRate`***`);`
  see `macro proc Pal`***`X`***`Run (`***`HandleCT`***`, `***`ClockRate`***`); (see page 23)`
- `macro proc PalAudioInReset (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Reset (`***`Handle`***`);`

- `macro proc PalAudioInEnable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Enable (`***`Handle`***`);`
- `macro proc PalAudioInDisable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Disable (`***`Handle`***`);`
- `macro proc PalAudioInSetSampleRate (`***`Handle`***`, `***`SampleRate`***`);`
- `macro proc PalAudioInRead (`***`Handle`***`, `***`LeftPtr`***`, `***`RightPtr`***`);`

PalAudioInSetSampleRate () macro

`macro proc PalAudioInSetSampleRate (`***`Handle`***`, `***`SampleRate`***`);`

Arguments	***`Handle`***: `PalHandle` to an AudioIn resource. ***`SampleRate`***: integer constant, equal to the number of samples per second that the input device should generate.
Timing	1 or more clock cycles.
Description	Sets the sample rate of the audio input device. If the sample rate requested is not supported by the device then the rate is left unchanged.

PalAudioInRead () macro

`macro proc PalAudioInRead (`***`Handle`***`, `***`LeftPtr`***`, `***`RightPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to an AudioIn resource. ***`LeftPtr`***, ***`RightPtr`***: Expressions of type `signed (PalAudioInGetMaxDataWidthCT ())`.
Timing	1 or more clock cycles.
Description	Reads a stereo pair of samples from the audio input device. Will block until the device has a new sample. If `PalAudioInRead()` is not called quickly enough, some samples may be missed.

4.3.16 Audio output devices (AudioOut API)

The AudioOut API supports generic audio output devices.

- `macro expr PalAudioOut (`***`Index`***`);`
  see `macro expr Pal`***`X`***` (`***`Index`***`); (see page 21)`
- `macro expr PalAudioOutCT (`***`Index`***`);`
  see `macro expr Pal`***`X`***`CT (`***`Index`***`); (see page 21)`
- `macro expr PalAudioOutCount ();`
  see `macro expr Pal`***`X`***`Count (); (see page 22)`
- `macro expr PalAudioOutUniqueCount ();`
  see `macro expr Pal`***`X`***`UniqueCount (); (see page 22)`
- `macro proc PalAudioOutRequire (`***`Count`***`);`
  see `macro proc Pal`***`X`***`Require (`***`Count`***`); (see page 22)`

- `macro expr PalAudioOutGetMaxDataWidthCT ();`
  see `macro expr Pal`***`X`***`GetMaxDataWidthCT ();`
- `macro expr PalAudioOutGetDataWidth (`***`Handle`***`);`
  see `macro expr Pal`***`X`***`GetDataWidth (`***`Handle`***`);`
- `macro expr PalAudioOutGetDataWidthCT (`***`HandleCT`***`);`
  see `macro expr Pal`***`X`***`GetDataWidthCT (`***`HandleCT`***`);`
- `macro proc PalAudioOutRun (`***`HandleCT`***`, `***`ClockRate`***`);`
  see `macro proc Pal`***`X`***`Run (`***`HandleCT`***`, `***`ClockRate`***`); (see page 23)`
- `macro proc PalAudioOutReset (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Reset (`***`Handle`***`);`
- `macro proc PalAudioOutEnable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Enable (`***`Handle`***`);`
- `macro proc PalAudioOutDisable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Disable (`***`Handle`***`);`
- `macro proc PalAudioOutSetSampleRate (`***`Handle`***`, `***`SampleRate`***`);`
- `macro proc PalAudioOutWrite (`***`Handle`***`, `***`Left`***`, `***`Right`***`);`

PalAudioOutSetSampleRate () macro

`macro proc PalAudioOutSetSampleRate (`***`Handle`***`, `***`SampleRate`***`);`

Arguments	***`Handle`***: `PalHandle` to an AudioOut resource. ***`SampleRate`***: integer constant, equal to the number of samples per second that the output device should expect.
Timing	1 or more clock cycles.
Description	Sets the sample rate of the audio output device. If the sample rate requested is not supported by the device then the rate is left unchanged.

PalAudioOutWrite() macro

`macro proc PalAudioOutWrite (`***`Handle`***`, `***`Left`***`, `***`Right`***`);`

Arguments	***`Handle`***: `PalHandle` to an AudioOut resource. ***`Left`***, ***`Right`***: Expressions of type `unsigned (PalAudioOutGetMaxWidthCT())`.
Timing	1 or more clock cycles.
Description	Writes a stereo pair of samples to the audio output device. Will block until the device is ready to accept a new sample. If `PalAudioOutWrite()` is not called quickly enough, gaps may occur in the audio output.

4.3.17 PAL SDRAM API

Synchronous dynamic random access memory (SDRAM) is a memory technology that uses a clock to synchronize signal input and output on a memory chip. The SDRAM clock is coordinated with the

SDRAM controller clock so the timing of the memory chips and the timing of the controller are synchronized. SDRAM delivers bursts of data at high speeds using the synchronous interface. Its is actually SDR SDRAM (single data rate SDRAM) but is usually referred to as just "SDRAM".

The summary in the controller is available in SDRAM controller library document.

- `macro expr PalSDRAM (Index);`

  see `macro expr Pal`***`X`*** `(`***`Index`***`);` `(see page 21)`
- `macro expr PalSDRAMCT (IndexCT);`

  see `macro expr Pal`***`X`***`CT (`***`IndexCT`***`);` `(see page 21)`
- `macro proc PalSDRAMRequire (Count);`

  see `macro proc Pal`***`X`***`Require (`***`Count`***`);` `(see page 22)`
- `macro expr PalSDRAMCount ();`

  see `macro expr Pal`***`X`***`Count ();` `(see page 22)`
- `macro expr PalSDRAMUniqueCount ();`

  see `macro expr Pal`***`X`***`UniqueCount ();` `(see page 22)`
- `macro expr PalSDRAMGetMaxDataWidthCT ();`

  see `macro expr Pal`***`X`***`GetMaxDataWidthCT ();`
- `macro expr PalSDRAMGetDataWidth (Handle);`

  see `macro expr Pal`***`X`***`GetDataWidth (`***`Handle`***`);`
- `macro expr PalSDRAMGetAddressWidth (Handle);` `(see page 52)`
- `macro expr PalSDRAMGetDataWidthCT (HandleCT);`

  see `macro expr PalXGetDataWidthCT (`***`HandleCT`***`);`
- `macro expr PalSDRAMGetAddressWidthCT ();` `(see page 52)`
- `macro expr PalSDRAMGetMaxBurstLengthCT ()` `(see page 52);`
- `macro expr PalSDRAMGetBurstLengthCycles (Handle);` (see page 52)
- `macro proc PalSDRAMRun (HandleCT, BurstLength, ClockRate);` `(see page 53)`
- `macro proc PalSDRAMRunEx (HandelCT, MemoryName, ClockRate);` (see page 54)
- `macro proc PalSDRAMSetAddress (Handle, Address);` (see page 54)
- `macro proc PalSDRAMSetAddressMask (Handle, Address, Mask);` (see page 54)
- `macro proc PalSDRAMWrite (Handle, Data);` (see page 55)
- `macro proc PalSDRAMRead (Handle, DataPtr);` (see page 52)
- `macro proc PalSDRAMWriteBuffer (Handle, BufferPtr);` (see page 55)
- `macro proc PalSDRAMReadBuffer (Handle, BufferPtr);` (see page 53)
- `macro proc PalSDRAMReset (Handle);` `(see page 53)`
- `macro proc PalSDRAMEnable (Handle);`

  see `macro proc Pal`***`X`***`Enable (`***`Handle`***`);`
- `macro proc PalSDRAMDisable (Handle);`

  see `macro proc Pal`***`X`***`Disable (`***`Handle`***`);`

PalSDRAMGetAddressWidth () macro

```
macro expr PalSDRAMGetAddressWidth ();
```

Arguments	***`Handle`***: `PalHandle` to SDRAM resource.
Return value	Integer value, evaluated at runtime.
Description	Returns the actual width of the address bus of the SDRAM that is used by the handle referred to, at runtime. Since this method works at runtime it can be used even when ***`Handle`*** is not a constant (such as when it is passed through a function parameter). However, it cannot be used to set the width of variables.

PalSDRAMGetBurstLengthCycles () macro

```
macro expr PalSDRAMGetBurstLengthCycles (Handle)
```

Arguments	***`Handle`***: `PalHandle` to SDRAM resource.
Return value	Integer value for specifying a burst length in number of clock cycles.
Description	Returns the burst length of the SDRAM, at compile time.

PalSDRAMGetMaxAddressWidthCT () macro

```
macro expr PalSDRAMGetMaxAddressWidthCT ();
```

Arguments	None.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the maximum width of the address bus of the SDRAM, at compile time. This is the width that should be used for the address parameter to the RAM read and write methods.

PalSDRAMGetMaxBurstLengthCT () macro

```
macro expr PalSDRAMGetMaxBurstLengthCT ();
```

Arguments	None.
Return value	Integer compile-time constant for specifying a burst length.
Description	Returns the maximum burst length of the SDRAM, at compile time.

PalSDRAMRead () macro

```
macro proc PalSDRAMRead (Handle, DataPtr);
```

Arguments	***`Handle`***: `PalHandle` to an SDRAM resource. ***`DataPtr`***: Pointer to variable to read into.
Timing	Variable number of clock cycles. Because the refresh request generated internally has the highest priority the macro will wait until the refresh is finished. Minimal latency is CAS latency + 2 clock cycles.
Description	Reads data from the SDRAM resource. Must be called as many times as the burst length is set. For example:

```
seq (i = 0; i < PalSDRAMGetBustLength(Handle); i++)
{
     PalSDRAMRead (Handle, &Data[i]);
}
```

PalSDRAMReadBuffer () macro

`macro proc PalSDRAMReadBuffer (`***`Handle`***`, `***`BufferPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to an SDRAM resource. ***`BufferPtr`***: Pointer to a buffer (e.g. variable or signal) of the appropriate type. For many uses this an unsigned of width (PalSDRAMGetMaxDataWidthCT ()). The size of the buffer is derived during compile time from burst length value set in the SDRAM run macro.
Timing	Variable number of clock cycles. Because the refresh request generated internally has the highest priority the macro will wait until the refresh is finished. Minimal latency is CAS latency + 2 clock cycles.
Description	Reads data from SDRAM memory to buffer. Waits until all data from the burst transfer are read into the buffer. For example:

```
unsigned MyBuffer[];
     PalSDRAMReadBuffer (Handle, MyBuffer);
```

PalSDRAMReset () macro

`macro proc PalSDRAMReset (`***`Handle`***`);`

Arguments	***`Handle`***: `PalHandle` to resource of type SDRAM.
Timing	POWER_UP_DELAY_US microseconds.
Description	Resets the SDRAM controller. When reset is issued the first read/write command will be accepted after POWER_UP_DELAY_US set by the user in DEFINE_SDRAM_MEMORY_PARAMETERS macro declared in sdram_controller.hch.

PalSDRAMRun

`macro proc PalSDRAMRun (`***`HandleCT`***`, `***`BurstLength`***`, `***`ClockRate`***`);`

Arguments	***`HandleCT`***: constant `PalHandle` to an SDRAM resource
	`BurstLength`: burst length to use for SDRAM data transfers
	`ClockRate`: Clock rate of the clock domain of the call to this macro, in Hz.
Timing	Does not terminate in normal use.
Description	Runs the device management tasks for the SDRAM interface. Must always run in parallel with accesses to the device. CAS latency is fixed at 2. If you need to use a different CAS setting, see ***PalSDRAMRunEx () macro*** (see page 54).

PalSDRAMRunEx () macro

```
macro proc PalSDRAMRunEx (HandleCT, MemoryName, ClockRate);
```

Arguments	***`HandleCT`***: constant `PalHandle` to an SDRAM resource
	`MemoryName`: SDRAM memory parameters defined using DEFINE_SDRAM_MEMORY_PARAMETERS macro declared in sdram_controller.hch
	`ClockRate`: Clock rate of the clock domain of the call to this macro, in Hz.
Timing	Does not terminate in normal use.
Description	Runs the device management tasks for the SDRAM interface. Must always run in parallel with accesses to the device.
	If you do not know what type or configuration of SDRAM memory will be connected to the FPGA chip (for example: SDRAM memory modules with different memory size, Mobile SDRAM, or replaceable modules) this macro allows you to pass memory parameters defined in DEFINE_SDRAM_MEMORY_PARAMETERS directly to the run macro from PAL layer.

PalSDRAMSetAddress () macro

```
macro proc PalSDRAMSetAddress (Handle, Address);
```

Arguments	***`Handle`***: `PalHandle` to an SDRAM resource.
	`Address`: Address of data to read, of type `unsigned (PalSDRAMGetMaxAddressWidth (Handle))`.
Timing	1 clock cycle.
Description	Sets the address for a read or write. For example:

```
seq
{
    PalSDRAMSetAddress (Handle, Addr);
    PalSDRAMRead (Handle, &Data);
}
```

PalSDRAMSetAddressMask () macro

```
macro proc PalSDRAMSetAddressMask (Handle, Address, Mask);
```

Arguments ***`Handle`***: `PalHandle` to an SDRAM resource.

`Address`: Address of data to read, of type `unsigned (PalSDRAMGetMaxAddressWidth (Handle))`.

`Mask`: Byte write enable signals

Timing 1 clock cycle.

Description Sets the address for a read or write. For example:

```
seq
{
    PalSDRAMSetAddressMask (Handle, Addr, 0b0011);
    PalSDRAMWrite (Handle, Data);
}
```

PalSDRAMWrite () macro

`macro proc PalSDRAMWrite (`***`Handle`***`, `***`Data`***`);`

Arguments ***`Handle`***: `PalHandle` to an SDRAM resource.

`Data`: Data to write

Timing Variable number of clock cycles. Because the refresh request generated internally has the highest priority the macro will wait until the refresh is finished. Minimal latency is 1 clock cycles.

Description Writes data to SDRAM memory. Must be called as many times as the burst length is set. For example:

```
seq (i = 0; i < PalSDRAMGetBustLength(Handle); i++)
{
     PalSDRAMWrite (Handle, Data[i]);
}
```

PalSDRAMWriteBuffer () macro

`macro proc PalSDRAMWriteBuffer (`***`Handle`***`, `***`BufferPtr`***`);`

Arguments ***`Handle`***: `PalHandle` to an SDRAM resource.

`BufferPtr`: Pointer to a buffer of lvalues (e.g. variable or signal) of the appropriate type. For many uses this an unsigned of width (PalSDRAMGetMaxDataWidthCT ()). The size of the buffer is derived during compile time from burst length value set in the SDRAM run macro.

Timing Variable number of clock cycles. Because the refresh request generated internally has the highest priority the macro will wait until the refresh is finished. Minimal latency is 1 clock cycles.

Description Writes data buffer to SDRAM memory. Waits until all data from the buffer are written to the SDRAM memory. For example:

```
unsigned MyBuffer[];
MyBuffer[0] = 0xDEAD;
     PalSDRAMWriteBuffer (Handle, MyBuffer);
```

4.3.18 Ethernet devices (Ethernet API)

The Ethernet API supports generic Ethernet interface devices.

- `macro expr PalEthernet (`***`Index`***`);`
  see `macro expr Pal`***`X`***` (`***`Index`***`); (see page 21)`
- `macro expr PalEthernetCT (`***`Index`***`);`
  see `macro expr Pal`***`X`***`CT (`***`Index`***`); (see page 21)`
- `macro expr PalEthernetCount ();`
  see `macro expr Pal`***`X`***`Count (); (see page 22)`
- `macro expr PalEthernetUniqueCount ();`
  see `macro expr Pal`***`X`***`UniqueCount (); (see page 22)`
- `macro proc PalEthernetRequire (`***`Count`***`);`
  see `macro proc Pal`***`X`***`Require (`***`Count`***`); (see page 22)`
- `macro proc PalEthernetRun (`***`HandleCT`***`, `***`MacAddress`***`, `***`ClockRate`***`);`
- `macro proc PalEthernetReset (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Reset (`***`Handle`***`);`
- `macro proc PalEthernetEnable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Enable (`***`Handle`***`);`
- `macro proc PalEthernetDisable (`***`Handle`***`);`
  see `macro proc Pal`***`X`***`Disable (`***`Handle`***`);`
- `macro proc PalEthernetReadBegin (`***`Handle`***`, `***`DestinationPtr`***`, `***`SourcePtr`***`, `***`TypePtr`***`, `***`DataByteCountPtr`***`, `***`ErrorPtr`***`);`
- `macro proc PalEthernetRead (`***`Handle`***`, `***`DataPtr`***`, `***`ErrorPtr`***`);`
- `macro proc PalEthernetReadEnd (`***`Handle`***`,  `***`ErrorPtr`***`);`
- `macro proc PalEthernetWriteBegin (`***`Handle`***`, `***`Destination`***`, `***`Type`***`, `***`DataByteCount`***`, `***`ErrorPtr`***`);`
- `macro proc PalEthernetWrite (`***`Handle`***`, `***`Data`***`, `***`ErrorPtr`***`);`
- `macro proc PalEthernetWriteEnd (`***`Handle`***`,  `***`ErrorPtr`***`);`

PalEthernetRun () macro

`macro proc PalEthernetRun (`***`HandleCT`***`, `***`MacAddress`***`, `***`ClockRate`***`);`

Arguments	***`HandleCT`***: constant `PalHandle` to a PalEthernet resource
	`MacAddress`: Ethenet MAC address to be used by the network chip, or type `unsigned 48`.
	`ClockRate`: Clock rate of the clock domain of the call to this macro, in Hz.
Timing	Does not terminate in normal use.
Description	Runs the device management tasks for the Ethernet interface. Must always run in parallel with accesses to the device.

PalEthernetReadBegin () macro

`macro proc PalEthernetReadBegin (`***`Handle`***`, `***`DestinationPtr`***`, `***`SourcePtr`***`, `***`TypePtr`***`, `***`DataByteCountPtr`***`, `***`ErrorPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to a PalEthernet resource.
	`DestinationPtr`: Pointer to data of type `unsigned 48`. Will return the destination MAC address from the received packet.
	`SourcePtr`: Pointer to data of type `unsigned 48`. Will return the source MAC address from the received packet.
	`TypePtr`: Pointer to data of type `unsigned 16`. Will return the type of the received packet.
	`DataByteCountPtr`: Pointer to data of type `unsigned 11`. Will return the number of data bytes in the packet, excluding the header.
	`ErrorPtr`: Pointer to data of `unsigned 1`. Returns success = 0, failure = 1.
Timing	1 or more clock cycles.
Description	Checks to see if a packet is waiting to be read, and if it is initiates the read process and returns source, destination etc. This must be followed by calls to `PalEthernetRead()` to get data from the packet and `PalEthernetReadEnd()` to complete the process. If no packet is waiting to be read, the macro will return ***`ErrorPtr`*** as 1, indicating an error.

PalEthernetRead () macro

`macro proc PalEthernetRead (`***`Handle`***`, `***`DataPtr`***`, `***`ErrorPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to a PalEthernet resource.
	`DataPtr`: Pointer to data of type `unsigned 8`. Will return a byte of data from the received packet.
	`ErrorPtr`: Pointer to data of type `unsigned 1`. Returns success = 0, failure = 1.
Timing	One or more clock cycles.
Description	Returns a single data byte from the packet currently being read. Will return `ErrorPtr = 1`, indicating an error, if it is called when there is no data remaining in the packet.

PalEthernetReadEnd () macro

`macro proc PalEthernetReadEnd (`***`Handle`***`, `***`ErrorPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to a PalEthernet resource.
	`ErrorPtr`: Pointer to data of type `unsigned 1`. Returns success = 0, failure = 1.
Timing	One or more clock cycles.
Description	Completes the process of reading a packet from the Ethernet device. Must be called after all data has been read from a packet.

PalEthernetWriteBegin () macro

`macro proc PalEthernetWriteBegin (`***`Handle`***`, `***`Destination`***`, `***`Type`***`, `***`DataByteCount`***`, `***`ErrorPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to a PalEthernet resource.
	`Destination`: Data of type `unsigned 48`. Specifies the destination MAC address for the packet.
	`Type`: Data of type `unsigned 16`. Specifies the type of the outgoing packet.
	`DataByteCount`: Data of type `unsigned 11`. Specifies the number of data bytes to be sent, excluding the header.
	`ErrorPtr`: Pointer to data of type `unsigned 1`. Returns success = 0, failure = 1.
Timing	One or more clock cycles.
Description	Initiates a packet write operation, to send data to the network via the Ethernet device. ***`Destination`*** is the MAC address to send the packet to, and ***`DataByteCount`*** is the number of data bytes to be sent, which must be in the range 46-1500 for Ethernet. ***`ErrorPtr`*** will be set to 0 if the call was successful, and 1 otherwise.

PalEthernetWrite () macro

`macro proc PalEthernetWrite (`***`Handle`***`, `***`Data`***`, `***`ErrorPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to a PalEthernet resource.
	`Data`: Data of type `unsigned 8`, containing a byte of data to write to the packet.
	`ErrorPtr`: Pointer to data of type `unsigned 1`. Returns success = 0, failure = 1.
Timing	One or more clock cycles.
Description	Writes a single byte to the Ethernet device, to be added to the packet currently being written. Will return `ErrorPtr = 1`, indicating an error, if called when the expected number of bytes has already been written to the packet.

PalEthernetWriteEnd() macro

`macro proc PalEthernetWriteEnd (`***`Handle`***`, `***`ErrorPtr`***`);`

Arguments	***`Handle`***: `PalHandle` to a PalEthernet resource.
	`ErrorPtr`: Pointer to data of type `unsigned 1`. Returns success = 0, failure = 1.
Timing	One or more clock cycles.
Description	Completes the process of sending a packet. Must be called after all data has been written to a packet.

4.3.19 PAL Reconfigure API

The Reconfigure API supports simple resource reconfiguration.

- `macro expr PalReconfigure (`***`Index`***`);`

  see `macro expr PalX (Index); (see page 21)`
- `macro expr PalReconfigureCT (`***`IndexCT`***`);`

  see `macro expr PalXCT (`***`IndexCT`***`); (see page 21)`
- `macro expr PalReconfigureCount ();`

  see `macro expr PalXCount (); (see page 22)`

- `macro expr PalReconfigureUniqueCount ();`
  see `macro expr PalXUniqueCount (); (see page 22)`
- `macro proc PalReconfigureRequire (`***`Count`***`);`
  see `macro proc PalXRequire (`***`Count`***`); (see page 22)`
- `macro proc PalReconfigureNow (`***`Handle, ImageAddress`***`);`
  see `macro proc PalReconfigureNow (`***`Handle, ImageAddress`***`);`

4.3.20 PAL TouchScreen API

The touch screen API supports standard touch screen displays.

- `macro expr PalTouchScreenPort (`***`Index`***`);`
  `see macro expr PalX (Index); (see page 21)`
- `macro expr PalTouchScreenCT (`***`IndexCT`***`);`
  `see macro expr PalXCT (IndexCT); (see page 21)`
- `macro expr PalTouchScreenCount ();`
  `see macro expr PalXCount (); (see page 22)`
- `macro expr PalTouchScreenUniqueCount ();`
  `see macro expr PalXUniqueCount (); (see page 22)`
- `macro proc PalTouchScreenRequire (`***`Count`***`);`
  `see macro proc PalXRequire (Count); (see page 22)`
- `macro proc PalTouchScreenReadScaled (`***`Handle, XPtr, YPtr, TouchPtr`***`);`
  `see macro proc PalTouchScreenReadScaled (Handle, XPtr, YPtr, TouchPtr); (see page 61)`
- `macro proc PalTouchScreenReadRaw (`***`Handle, XPtr, YPtr, TouchPtr`***`);`
  `see macro proc PalTouchScreenReadRaw (Handle, XPtr, YPtr, TouchPtr); (see page 60)`
- `macro proc PalTouchScreenRun (`***`Handle, ClockRate`***`);`
  `see macro proc PalXRun (HandleCT, ClockRate); (see page 23)`
- `macro expr PalTouchScreenGetScaledXWidthCT ();`
  `see macro expr PalTouchScreenGetScaledXWidthCT (); (see page 60)`
- `macro expr PalTouchScreenGetScaledYWidthCT ();`
  `see macro expr PalTouchScreenGetScaledYWidthCT (); (see page 61)`
- `macro expr PalTouchScreenGetRawXWidthCT ();`
  `see macro expr PalTouchScreenGetRawXWidthCT (); (see page 61)`
- `macro expr PalTouchScreenGetRawYWidthCT ();`
  `see macro expr PalTouchScreenGetRawYWidthCT (); (see page 61)`
- `macro expr PalTouchScreenGetTouchWidthCT ();`
  `see macro expr PalTouchScreenGetTouchWidthCT ();` (see page 60)

PalTouchScreenGetScaledXWidthCT () macro

```
macro expr PalTouchScreenGetScaledXWidthCT ();
```

Arguments	None.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the maximum width of the X co-ordinate of the last touch on the touch screen, at compile time. This is the width that should be used for the lvalue that is pointed to by ***`XPtr`*** in the `PalTouchScreenReadScaled` method. Note that for the "Scaled" touch screen methods, the X and Y coordinates and corresponding widths are scaled for the pixel dimensions of the display available on a given platform.

PalTouchScreenReadRaw (Handle, XPtr, YPtr, TouchPtr) macro

```
macro proc PalTouchScreenReadRaw (Handle, XPtr, XPtr, TouchPtr);
```

Arguments	***`Handle`***: `PalHandle` to a PalTouchScreen resource. ***`XPtr`***: Pointer to an lvalue, of type `unsigned (PalTouchScreenGetRawXWidthCT ())`. ***`YPtr`***: Pointer to an lvalue, of type `unsigned (PalTouchScreenGetRawYWidthCT ())`. ***`TouchPtr`***: pointer to an lvalue (e.g. variable or signal) of type `unsigned (PalTouchScreenGetTouchWidthCT ())`.
Timing	1 or more cycles.
Description	Reads the current state of the TouchScreen pointed to by ***`Handle`***. The call sets both the touch state value in ***`TouchPtr`*** and the X and Y coordinates of the captured pixel in ***`XPtr`*** and ***`YPtr`***. Note that for the "Raw" touch screen methods, the X and Y coordinates and corresponding widths are NOT scaled for the pixel dimensions of the display available on a given platform.

PalTouchScreenGetTouchWidthCT () macro

```
macro expr PalTouchScreenGetTouchWidthCT ();
```

Arguments	None.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the maximum touch screen touch status width, at compile time. This is the width that should be used for the lvalue that is pointed to by ***`YPtr`*** in both the `PalTouchScreenReadRaw` and `PalTouchScreenReadScaled` methods. Note that the Touch width is the same regardless of X and Y scaling.

PalTouchScreenGetRawYWidthCT () macro

```
macro expr PalTouchScreenGetRawYWidthCT ();
```

Arguments	None.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the maximum width of the Y co-ordinate of the last touch on the touch screen, at compile time. This is the width that should be used for the lvalue that is pointed to by ***`YPtr`*** in the `PalTouchScreenReadRaw` method. Note that for the "Raw" touch screen methods, the X and Y coordinates and corresponding widths are NOT scaled for the pixel dimensions of the display available on a given platform.

PalTouchScreenGetScaledYWidthCT () macro

```
macro expr PalTouchScreenGetScaledYWidthCT ();
```

Arguments	None.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the maximum width of the Y co-ordinate of the last touch on the touch screen, at compile time. This is the width that should be used for the lvalue that is pointed to by ***`YPtr`*** in the `PalTouchScreenReadScaled` method. Note that for the "Scaled" touch screen methods, the X and Y coordinates and corresponding widths are scaled for the pixel dimensions of the display available on a given platform.

PalTouchScreenGetRawXWidthCT () macro

```
macro expr PalTouchScreenGetRawXWidthCT ();
```

Arguments	None.
Return value	Integer compile-time constant for specifying a width.
Description	Returns the maximum width of the X co-ordinate of the last touch on the touch screen, at compile time. This is the width that should be used for the lvalue that is pointed to by ***`XPtr`*** in the `PalTouchScreenReadRaw` method. Note that for the "Raw" touch screen methods, the X and Y coordinates and corresponding widths are NOT scaled for the pixel dimensions of the display available on a given platform.

PalTouchScreenReadScaled (Handle, XPtr, YPtr, TouchPtr) macro

```
macro proc PalTouchScreenReadScaled (Handle, XPtr, XPtr, TouchPtr);
```

Arguments	***`Handle`***: `PalHandle` to a PalTouchScreen resource.
	`XPtr`: Pointer to an lvalue, of type `unsigned (PalTouchScreenGetScaledXWidthCT ())`.
	`YPtr`: Pointer to an lvalue, of type `unsigned (PalTouchScreenGetScaledYWidthCT ())`.
	`TouchPtr`: pointer to an lvalue (e.g. variable or signal) of type `unsigned (PalTouchScreenGetTouchWidthCT ())`.
Timing	1 or more cycles.
Description	Reads the current state of the TouchScreen pointed to by ***`Handle`***. The call sets both the touch state value in ***`TouchPtr`*** and the X and Y coordinates of the captured pixel in ***`XPtr`*** and ***`YPtr`***.
	Note that for the "Scaled" touch screen methods, the X and Y coordinates and corresponding widths are scaled for the pixel dimensions of the display available on a given platform.

5 Index

E

errors 21
examples
  PAL 9, 11

H

handles 19
header files 8

P

PAL 7
  compile-time selection 15
  creating your own design 12
  data width abstraction 15
  devices supported 8
  generic libraries 12
  header files 8
  introduction 7
  library files 8
  platforms supported 8
  resource handles 14
  run-time selection 15
  setting up a device 16
  simulating 10, 14
  targeting hardware 13, 14
  user guide 12
PAL API 18
  audio input 50
  audio output 51
  block storage 38
  buttons 27
  data I/O 27
  Ethernet 58
  LEDs 25
  parallel ports 28
  PS/2 ports 29
  RAM 30, 32, 34, 36, 38, 52
  Reconfigure 61
  resource handles 19
  RS-232 29
  serial ports 29
  seven-segment displays 25
  switches 27
  version numbers 19
  video input 40
  video output 44
PAL Cores........ 9, 12
PAL Kit 16
PAL resources 19
PAL TouchScreen API 61
pal.hch 18
PalAudioIn........ 50
PalAudioInCount 50
PalAudioInCT 50
PalAudioInDisable........ 50
PalAudioInEnable 50
PalAudioInGetDataWidth........ 50
PalAudioInGetMaxDataWidthCT 50
PalAudioInRead 51
PalAudioInRequire 50
PalAudioInReset 50
PalAudioInRun 50
PalAudioInSetSampleRate 51
PalAudioInUniqueCount 50
PalAudioOut........ 51
PalAudioOutCount 51
PalAudioOutCT 51
PalAudioOutDisable........ 51
PalAudioOutEnable 51
PalAudioOutGetMaxDataWidthCT 51
PalAudioOutRequire 51
PalAudioOutReset 51
PalAudioOutRun 51
PalAudioOutSetSampleRate 52
PalAudioOutUniqueCount........ 51
PalBlockStore 38
PalBlockStoreCount........ 38
PalBlockStoreCT........ 38
PalBlockStoreDisable 38
PalBlockStoreEnable 38
PalBlockStoreEraseBlock 40
PalBlockStoreGetAddressWidth 38
PalBlockStoreGetAddressWidthCT 38
PalBlockStoreGetBlockLength........ 39
PalBlockStoreGetBlockLengthCT........ 39
PalBlockStoreGetDataWidth........ 38

PalBlockStoreGetDataWidthCT 38
PalBlockStoreGetMaxAddressWidthCT 38
PalBlockStoreGetMaxBlockLengthCT.......... 38
PalBlockStoreGetMaxDataWidthCT............. 38
PalBlockStoreRead 39
PalBlockStoreRequire 38
PalBlockStoreReset...................................... 38
PalBlockStoreRun 38
PalBlockStoreUniqueCount 38
PalBlockStoreWrite....................................... 40
PalDataPort 27
PalDataPortCount... 27
PalDataPortCT 27
PalDataPortDisable 27
PalDataPortEnable 27
PalDataPortGetDataWidth............................ 27
PalDataPortGetDataWidthCT....................... 27
PalDataPortGetMaxDataWidthCT 27
PalDataPortRead.. 28
PalDataPortRequire...................................... 27
PalDataPortReset... 27
PalDataPortRun.. 27
PalDataPortUniqueCount 27
PalDataPortWrite 28
PalErrorCode 21
PalErrorHandlerRun 21
PalEthernet 58
PalEthernetCount 58
PalEthernetCT 58
PalEthernetDisable 58
PalEthernetEnable.. 58
PalEthernetRead 59
PalEthernetReadBegin 59
PalEthernetReadEnd 60
PalEthernetRequire 58
PalEthernetUniqueCount.............................. 58
PalEthernetWrite 60
PalEthernetWriteBegin 60
PalEthernetWriteEnd 61
PalFastRAM 30
PalFastRAMCount.. 30
PalFastRAMCT... 30
PalFastRAMDisable.....................................30
PalFastRAMEnable......................................30
PalFastRAMGetAddressWidth 31
PalFastRAMGetAddressWidthCT.................31
PalFastRAMGetDataWidth 30
PalFastRAMGetDataWidthCT 30
PalFastRAMGetMaxAddressWidthCT..........30
PalFastRAMGetMaxDataWidthCT 30
PalFastRAMRead 31
PalFastRAMRequire 30
PalFastRAMReset 30
PalFastRAMRun 30
PalFastRAMUniqueCount............................30
PalFastRAMWrite 31
PalHandle 19
PalLED 25
PalLEDCount 25
PalLEDCT 25
PalLEDRequire 25
PalLEDUniqueCount....................................25
PalLEDWrite 25
PalParallelPort 28
PalParallelPortCount 28
PalParallelPortCT 28
PalParallelPortRequire 28
PalParallelPortUniqueCount 28
PalPL1RAM 32
PalPL1RAMCount..32
PalPL1RAMCT...32
PalPL1RAMDisable 32
PalPL1RAMEnable 32
PalPL1RAMGetAddressWidth 32
PalPL1RAMGetAddressWidthCT 32
PalPL1RAMGetDataWidth............................32
PalPL1RAMGetDataWidthCT.......................32
PalPL1RAMGetMaxAddressWidthCT 32
PalPL1RAMGetMaxDataWidthCT 32
PalPL1RAMPortRequire 32
PalPL1RAMRead...35
PalPL1RAMReset 32
PalPL1RAMRun...32
PalPL1RAMSetReadAddress 33

PalPL1RAMSetWriteAddress....................... 33
PalPL1RAMUniqueCount............................ 32
PalPL1RAMWrite... 33
PalPL2RAM 34
PalPL2RAMCount 34
PalPL2RAMCT 34
PalPL2RAMDisable 34
PalPL2RAMEnable....................................... 34
PalPL2RAMGetAddressWidth...................... 34
PalPL2RAMGetAddressWidthCT................. 34
PalPL2RAMGetDataWidth 34
PalPL2RAMGetDataWidthCT 34
PalPL2RAMGetMaxAddressWidthCT.......... 34
PalPL2RAMGetMaxDataWidthCT................ 34
PalPL2RAMPortRequire............................... 34
PalPL2RAMRead 35
PalPL2RAMReset.. 34
PalPL2RAMRun 34
PalPL2RAMSetReadAddress....................... 35
PalPL2RAMUniqueCount............................ 34
PalPL2RAMWrite... 36
PalPL2SetWriteAddress 35
PalPS2Port 29
PalPS2PortCount 29
PalPS2PortCT 29
PalPS2PortRequire 29
PalPS2PortUniqueCount.............................. 29
PalReconfigure 61
PalRS232Port.. 29
PalRS232PortCount 29
PalRS232PortCT 29
PalRS232PortRequire 29
PalRS232PortUniqueCount.......................... 29
PalSevenSeg... 25
PalSevenSegCount 25
PalSevenSegCT 25
PalSevenSegDisable................................... 25
PalSevenSegEnable.................................... 25
PalSevenSegRequire 25
PalSevenSegUniqueCount........................... 25
PalSevenSegWriteDigit 26
PalSevenSegWriteShape 26
PALSim 10, 11, 14
PalSlowRAM 36
PalSlowRAMCount 36
PalSlowRAMCT 36
PalSlowRAMDisable 36
PalSlowRAMEnable.....................................36
PalSlowRAMGetAddressWidth.....................36
PalSlowRAMGetAddressWidthCT................36
PalSlowRAMGetDataWidth 36
PalSlowRAMGetDataWidthCT 36
PalSlowRAMGetMaxAddressWidthCT.........36
PalSlowRAMGetMaxDataWidthCT...............36
PalSlowRAMRead 37
PalSlowRAMRequire 36
PalSlowRAMReset.......................................36
PalSlowRAMRun 36
PalSlowRAMUniqueCount............................36
PalSlowRAMWrite..37
PalSwitch 27
PalSwitchCount..27
PalSwitchCT 27
PalSwitchRead...27
PalSwitchRequire...27
PalSwitchUniqueCount 27
PalTouchScreenGetRawXWidthCT..............64
PalTouchScreenGetRawYWidthCT..............63
PalTouchScreenGetScaledXWidthCT..........62
PalTouchScreenGetScaledYWidthCT..........64
PalTouchScreenGetTouchWidthCT 63
PalTouchScreenReadRaw 62
PalTouchScreenReadScaled........................64
PalVendorCode..20
PalVendorRelease 20
PalVendorString...20
PalVersionMajor...19
PalVersionMinor...19
PalVersionRequire 19
PalVideoIn...40
PalVideoInCount 40
PalVideoInCT...40
PalVideoInDisable..40
PalVideoInEnable 40

PalVideoInGetColorWidth............................. 42
PalVideoInGetColorWidthCT........................ 43
PalVideoInGetMaxColorWidthCT................. 42
PalVideoInGetMaxXWidthCT....................... 41
PalVideoInGetMaxYWidthCT....................... 41
PalVideoInGetXWidth................................... 42
PalVideoInGetXWidthCT.............................. 43
PalVideoInGetYWidth................................... 42
PalVideoInGetYWidthCT.............................. 43
PalVideoInRead... 44
PalVideoInRequire....................................... 40
PalVideoInReset.. 40
PalVideoInRun... 40
PalVideoInUniqueCount............................... 40
PalVideoOutCount....................................... 44
PalVideoOutCT.. 44
PalVideoOutDisable 44
PalVideoOutEnable 44
PalVideoOutGetColorWidth.......................... 44
PalVideoOutGetColorWidthCT..................... 44
PalVideoOutGetHBlank................................ 49
PalVideoOutGetMaxColorWidthCT.............. 44
PalVideoOutGetMaxXWidthCT 44
PalVideoOutGetMaxYWidthCT 44
PalVideoOutGetTotalX 46
PalVideoOutGetTotalXCT 48
PalVideoOutGetTotalY 47
PalVideoOutGetTotalYCT 48
PalVideoOutGetVBlank................................ 49
PalVideoOutGetVisibleX.............................. 46
PalVideoOutGetVisibleXCT.......................... 47
PalVideoOutGetVisibleY.............................. 46
PalVideoOutGetVisibleYCT.......................... 47
PalVideoOutGetX 48
PalVideoOutGetXWidth................................ 44
PalVideoOutGetXWidthCT 44
PalVideoOutGetY 49
PalVideoOutGetYWidth................................ 44
PalVideoOutGetYWidthCT 44
PalVideoOutOptimalCT................................ 46
PalVideoOutRequire..................................... 44
PalVideoOutReset.. 44
PalVideoOutRun 44
PalVideoOutUniqueCount............................44
PalVideoOutWrite 49
PalX 21
PalXCount..22
PalXCT...21
PalXDisable 24
PalXEnable 23
PalXGetDataWidth.......................................22
PalXGetDataWidthCT..................................23
PalXGetMaxDataWidthCT 22
PalXRead...24
PalXRequire...22
PalXReset..23
PalXRun...23
PalXUniqueCount 22
PalXWrite...24
PL1RAM...32
PL2RAM...34
Platform Abstraction Layer (PAL) 7
Platform Developer's Kit (PDK)
    PAL 7

T

touch screen 61
TouchScreen..61

V

vendor ID codes...20
version numbers 19