Celoxica

Platform Developer's Kit

PAL API Reference Manual

For PDK v2.0

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

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

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

Authors: JAA, MA

Document number: RM-1150-1.2

Table of contents

TABLE OF CONTENTS 1

CONVENTIONS 3

ASSUMPTIONS 4
OMISSIONS 4

1. INTRODUCTION 5
1.1 HEADER AND LIBRARY FILES 5

2. GENERIC API 6
2.1 RESOURCE HANDLES 6
2.2 VERSIONING 6
2.2.1 API version numbers 6
2.2.2 Library version information 7
2.3 ERROR HANDLING 8
2.4 GENERIC METHODS 8
2.4.1 Querying the number of resources 8
2.4.2 Getting resource handles 9
2.4.3 Querying resource properties 9
2.4.4 Running resources 10
2.4.5 Resetting resources 10
2.4.6 Enabling and disabling resources 11
2.4.7 Reading and writing resources 11

3. RESOURCE-SPECIFIC API 12
3.1 LEDS (LED API) 12
3.2 SEVEN-SEGMENT DISPLAYS (SEVENSEG API) 12
3.3 BINARY SWITCHES AND BUTTONS (SWITCH API) 13
3.4 GENERIC DATA I/O (DATAPORT API) 14
3.5 PARALLEL PORTS (PARALLELPORT API) 15
3.6 RS-232 SERIAL PORTS (RS232PORT API) 15
3.7 PS/2 SERIAL PORTS (PS2PORT API) 16
3.8 SINGLE-CYCLE RAMS (FASTRAM API) 16
3.9 PIPELINED SINGLE-CYCLE RAMS (PL1RAM API) 18
3.10 PIPELINED PL2 RAMS (PL2RAM API) 20
3.11 RAMS WITHOUT GUARANTEED TIMING (SLOWRAM API) 22
3.12 BLOCK STORAGE DEVICES (BLOCKSTORE API) 24
3.13 VIDEO CAPTURE DEVICES (VIDEOIN API) 26
3.14 VIDEO OUTPUT DEVICES (VIDEOOUT API) 30
3.15 AUDIO INPUT DEVICES (AUDIOIN API) 35

Table of contents

3.16 AUDIO OUTPUT DEVICES (AUDIOOUT API) 36
3.17 ETHERNET DEVICES (ETHERNET API) 37

INDEX.. I

Conventions

A number of conventions are used in this document. These conventions are detailed below.

☒ 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

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 MS Windows

Omissions

This manual does not include:

- instruction in the use of place and route tools
- tutorial example programs. These are provided in the *PAL tutorial guide.*

1. Introduction

This reference manual describes the Celoxica Platform Abstraction Layer (PAL) Application Programming Interface (API).

The 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
- Block storage devices
- Video input devices
- Video output devices
- Audio input devices
- Audio output devices
- Ethernet devices

PAL defines a set of methods common to all of the devices. These are described in section 0. Device specific methods are described in the relevant section about that device.

1.1 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_rc100.hcl` is needed to target the Celoxica RC100 board. This in turn requires the PSL library `rc100.hcl` and the standard library `stdlib.lib`. More information about choosing the appropriate libraries and header files is supplied in the *PAL user manual*.

2. Generic API

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 `PalX()` and `PalXCT()` family of macros. All macros that need to refer to a particular resource take a `PalHandle` as their first argument.

2.2 Versioning

2.2.1 API version numbers

`macro expr PalVersionMajor ();`

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.

`macro expr PalVersionMinor ();`

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.

`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.

2.2.2 Library version information

`macro expr PalVendorCode ();`

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.

`macro expr PalVendorRelease ();`

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.

`macro expr PalVendorString ();`

Return value String constant.

Description Returns a vendor specific string, primarily for use in debugging.

2.3 Error handling

`PalErrorCode`

Description The type of PAL run-time error codes. Possible values are:

Value	**Meaning**
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.

`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).

2.4 Generic methods

2.4.1 Querying the number of resources

`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).

`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.

`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.

2.4.2 Getting resource handles

`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.

`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.

2.4.3 Querying resource properties

`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.

`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.

`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`).

2.4.4 Running resources

`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.

2.4.5 Resetting resources

`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.

2.4.6 Enabling and disabling resources

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.

macro proc PalXDisable (*Handle*);

Arguments **Handle**: PalHandle to resource of type X.

Timing 0 or more clock cycles.

Description Disables the resource. Reverses PalXEnable ().

2.4.7 Reading and writing resources

macro proc PalXRead (*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 (PalXGetMaxDataWidthCT ()).

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**.

macro proc PalXWrite (*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 (PalXGetMaxDataWidthCT ()).

Timing 1 or more clock cycles.

Description Statement that writes a single item of data to the resource.

3. Resource-specific API

3.1 LEDs (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 ();

macro expr PalLEDCT (*Index*);

see macro expr PalXCT (***Index***);

macro expr PalLEDCount ();

see macro expr PalXCount ();

macro expr PalLEDUniqueCount ();

see macro expr PalXUniqueCount ();

macro proc PalLEDRequire (*Count*);

see macro proc PalXRequire (***Count***);

macro proc PalLEDWrite (*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.

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 PalX ();

macro expr PalSevenSegCT (*Index*);

see macro expr PalXCT (***Index***);

macro expr PalSevenSegCount ();

see macro expr PalXCount ();

macro expr PalSevenSegUniqueCount ();

see macro expr PalXUniqueCount ();

macro proc PalSevenSegRequire (*Count*);
see `macro proc PalXRequire (`***Count***`);`

macro proc PalSevenSegEnable (*Handle*);
see `macro expr PalXEnable (`***Handle***`);`

macro proc PalSevenSegDisable (*Handle*);
see `macro expr PalXDisable (`***Handle***`);`

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).

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

Arguments: ***Handle***: PalHandle to resource of type X.

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).

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 PalX (`***Index***`);`

macro expr PalSwitchCT (*Index*);
see `macro expr PalXCT (`***Index***`);`

macro expr PalSwitchCount ();
see `macro expr PalXCount ();`

macro expr PalSwitchUniqueCount ();
see `macro expr PalXUniqueCount ();`

`macro proc PalSwitchRequire (`*`Count`*`);`

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

`macro proc PalSwitchRead (`*`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).

3.4 Generic data I/O (DataPort API)

The data port API supports generic bi- or uni-directional data I/O.

`macro expr PalDataPort (`*`Index`*`);`

see `PalX (`***`Index`***`);`

`macro expr PalDataPortCT (`*`Index`*`);`

see `PalXCT (`***`Index`***`);`

`macro expr PalDataPortCount ();`

see `macro expr PalXCount ();`

`macro expr PalDataPortUniqueCount ();`

see `macro expr PalXUniqueCount ();`

`macro proc PalDataPortRequire (`*`Count`*`);`

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

`macro expr PalDataPortGetMaxDataWidthCT ();`

see `macro expr PalXGetMaxDataWidthCT ();`

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

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

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

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

`macro proc PalDataPortRun (`*`HandleCT`*`, `*`ClockRate`*`);`

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

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

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

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

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

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

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

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***.

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.

3.5 Parallel ports (ParallelPort API)

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

macro expr PalParallelPort (*Index*);

see `PalX (`***Index***`);`

macro expr PalParallelPortCT (*Index*);

see `PalXCT (`***Index***`);`

macro expr PalParallelPortCount ();

see `macro expr PalXCount ();`

macro expr PalParallelPortUniqueCount ();

see `macro expr PalXUniqueCount ();`

macro proc PalParallelPortRequire (*Count*);

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

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 `PalX (`***Index***`);`

macro expr PalRS232PortCT (*Index*);
see PalXCT (***Index***);

macro expr PalRS232PortCount ();
see macro expr PalXCount ();

macro expr PalRS232PortUniqueCount ();
see macro expr PalXUniqueCount ();

macro proc PalRS232PortRequire (*Count*);
see macro proc PalXRequire (***Count***);

3.7 PS/2 serial ports (PS2Port API)

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

macro expr PalPS2Port (*Index*);
see PalX (***Index***);

macro expr PalPS2PortCT (*Index*);
see PalXCT (***Index***);

macro expr PalPS2PortCount ();
see macro expr PalXCount ();

macro expr PalPS2PortUniqueCount ();
see macro expr PalXUniqueCount ();

macro proc PalPS2PortRequire (*Count*);
see macro proc PalXRequire (***Count***);

3.8 Single-cycle RAMs (FastRAM API)

FastRAMs are single-cycle RAMs that can be read from or written to in exactly one clock cycle, hence they are fast in terms of clock cycles taken. However, FastRAMs often have 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 PalX (***Index***);

macro expr PalFastRAMCT (*Index*);
see PalXCT (***Index***);

macro expr PalFastRAMCount ();
see macro expr PalXCount ();

macro expr PalFastRAMUniqueCount ();
see macro expr PalXUniqueCount ();

`macro proc PalFastRAMRequire (`*`Count`*`);`

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

`macro expr PalFastRAMGetMaxDataWidthCT ();`

see `macro expr PalXGetMaxDataWidthCT ();`

`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.

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

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

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

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.

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

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

`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`).

`macro proc PalFastRAMRun (`*`HandleCT`*`, `*`ClockRate`*`);`

see `macro expr PalXRun (`***`HandleCT`***`, `***`ClockRate`***`);`

macro proc PalFastRAMReset (*Handle*);

see macro expr PalXReset (***Handle***);

macro proc PalFastRAMEnable (*Handle*);

see macro expr PalXEnable (***Handle***);

macro proc PalFastRAMDisable (*Handle*);

see macro expr PalXDisable (***Handle***);

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***.

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.

3.9 Pipelined single-cycle RAMs (PL1RAM API)

PL1RAMs 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 a 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 PalX (***Index***);

macro expr PalPL1RAMCT (*Index*);

see macro expr PalXCT (***Index***);

macro expr PalPL1RAMCount ();

see macro expr PalXCount ();

3. Resource-specific API

macro expr PalPL1RAMUniqueCount ();

see macro expr PalXUniqueCount ();

macro proc PalPL1RAMPortRequire (*Count*);

see macro proc PalXRequire (***Count***);

macro expr PalPL1RAMGetMaxDataWidthCT ();

see macro expr PalXGetMaxDataWidthCT ();

macro expr PalPL1RAMGetMaxAddressWidthCT ();

see macro expr PALFastRAMGetMaxAddressWidthCT ();

macro expr PalPL1RAMGetDataWidth (*Handle*);

see macro expr PalXGetDataWidth (***Handle***);

macro expr PalPL1RAMGetAddressWidth (*Handle*);

see macro expr PalFastRAMGetAddressWidth (***Handle***);

macro expr PalPL1RAMGetDataWidthCT (*HandleCT*);

see macro expr PalXGetDataWidthCT (***HandleCT***);

macro expr PalPL1RAMGetAddressWidthCT (*HandleCT*);

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

macro proc PalPL1RAMRun (*HandleCT*, *ClockRate*);

see macro expr PalXRun (***HandleCT***, ***ClockRate***);

macro proc PalPL1RAMReset (*Handle*);

see macro expr PalXReset (***Handle***);

macro proc PalPL1RAMEnable (*Handle*);

see macro expr PalXEnable (***Handle***);

macro proc PalPL1RAMDisable (*Handle*);

see macro expr PalXDisable (***Handle***);

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.

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

Arguments ***Handle***: PalHandle to a PL1RAM resource.

Address: Address of data to be written, of type unsigned (`PalPL1RAMGetMaxAddressWidthCT ()`).

Timing 1 clock cycle.

Description Sets the address for a write that will occur on the next clock cycle.

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 (`PalPL1RAMGetMaxDataWidthCT ()`).

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***.

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

Arguments ***Handle***: PalHandle to a PL1RAM resource.

Data: expression of type unsigned (`PalPL1RAMGetMaxDataWidthCT ()`).

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.

3.10 Pipelined PL2 RAMs (PL2RAM API)

PL2RAMs are similar to PL1 RAMs but there are two clock cycles between setting the address and reading/writing data. The RAMs can be read from or written to in exactly one clock cycle, but with the address supplied two 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 PalX (`***Index***`);`

macro expr PalPL2RAMCT (*Index*);

see `macro expr PalXCT (`***Index***`);`

macro expr PalPL2RAMCount ();

see `macro expr PalXCount ();`

3. Resource-specific API

macro expr PalPL2RAMUniqueCount ();

see macro expr PalXUniqueCount ();

macro proc PalPL2RAMPortRequire (*Count*);

see macro proc PalXRequire (***Count***);

macro expr PalPL2RAMGetMaxDataWidthCT ();

see macro expr PalXGetMaxDataWidthCT ();

macro expr PalPL2RAMGetMaxAddressWidthCT ();

see macro expr PALFastRAMGetMaxAddressWidthCT ();

macro expr PalPL2RAMGetDataWidth (*Handle*);

see macro expr PalXGetDataWidth (***Handle***);

macro expr PalPL2RAMGetAddressWidth (*Handle*);

see macro expr PalFastRAMGetAddressWidth (***Handle***);

macro expr PalPL2RAMGetDataWidthCT (*HandleCT*);

see macro expr PalXGetDataWidthCT (***HandleCT***);

macro expr PalPL2RAMGetAddressWidthCT (*HandleCT*);

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

macro proc PalPL2RAMRun (*HandleCT*, *ClockRate*);

see macro expr PalXRun (***HandleCT***, ***ClockRate***);

macro proc PalPL2RAMReset (*Handle*);

see macro expr PalXReset (***Handle***);

macro proc PalPL2RAMEnable (*Handle*);

see macro expr PalXEnable (***Handle***);

macro proc PalPL2RAMDisable (*Handle*);

see macro expr PalXDisable (***Handle***);

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);
}
```

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

Arguments ***Handle***: PalHandle to a PL2RAM resource.

Address: Address of data to be written, of type unsigned (`PalPL2RAMGetMaxAddressWidthCT ()`).

Timing 1 clock cycle.

Description Sets the address for a write that will occur two clock cycles later.

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

Arguments ***Handle***: PalHandle to a PL2RAM resource.

DataPtr: pointer to an 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***.

macro proc PalPL2RAMWrite (*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.

3.11 RAMs without guaranteed timing (SlowRAM API)

SlowRAMs are RAMs that can be read from or written to in multiple cycles. This is the most generic and platform independent interface to RAMs and other random access storage devices. This API would be 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 PalX (`***Index***`);`

macro expr PalSlowRAMCT (*Index*);

see `macro expr PalXCT (`***Index***`);`

macro expr PalSlowRAMCount ();

see `macro expr PalXCount ();`

3. Resource-specific API

macro expr PalSlowRAMUniqueCount ();
see macro expr PalXUniqueCount ();

macro proc PalSlowRAMRequire (*Count*);
see macro proc PalXRequire (***Count***);

macro expr PalSlowRAMGetMaxDataWidthCT ();
see macro expr PalXGetMaxDataWidthCT ();

macro expr PalSlowRAMGetMaxAddressWidthCT ();
see macro expr PalFastRAMGetMaxAddressWidthCT ();

macro expr PalSlowRAMGetDataWidth (*Handle*);
see macro expr PalXGetDataWidth (***Handle***);

macro expr PalSlowRAMGetAddressWidth (*Handle*);
see macro expr PALFastRAMGetAddressWidth (***Handle***);

macro expr PalSlowRAMGetDataWidthCT (*HandleCT*);
see macro expr PalXGetDataWidthCT (***HandleCT***);

macro expr PalSlowRAMGetAddressWidthCT (*HandleCT*);
see macro expr PalFastRAMGetAddressWidthCT (***HandleCT***);

macro proc PalSlowRAMRun (*HandleCT*, *ClockRate*);
see macro proc PalXRun (***HandleCT***, ***ClockRate***);

macro proc PalSlowRAMReset (*Handle*);
see macro proc PalXReset (***Handle***);

macro proc PalSlowRAMEnable (*Handle*);
see macro proc PalXEnable (***Handle***);

macro proc PalSlowRAMDisable (*Handle*);
see macro proc PalXDisable (***Handle***);

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***.

`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.

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 PalX (`**_`Index`_**`);`

`macro expr PalBlockStoreCT (`*`Index`*`);`

see `macro expr PalXCT (`**_`Index`_**`);`

`macro expr PalBlockStoreCount ();`

see `macro expr PalXCount ();`

`macro expr PalBlockStoreUniqueCount ();`

see `macro expr PalXUniqueCount ();`

`macro proc PalBlockStoreRequire (`*`Count`*`);`

see `macro proc PalXRequire (`**_`Count`_**`);`

`macro expr PalBlockStoreGetMaxDataWidthCT ();`

see `macro expr PalXGetMaxDataWidthCT ();`

`macro expr PalBlockStoreGetMaxAddressWidthCT ();`

see `macro expr PALFastRAMGetMaxAddressWidthCT ();`

`macro expr PalBlockStoreGetMaxBlockLengthCT ();`

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.

3. Resource-specific API

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

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

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

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

`macro expr PalBlockStoreGetBlockLength (`*`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.

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

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

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

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

`macro expr PalBlockStoreGetBlockLengthCT (`*`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`).

`macro proc PalBlockStoreRun (`*`HandleCT, ClockRate`*`);`

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

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

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

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

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

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

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

`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***.

`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.

`macro proc PalBlockStoreEraseBlock (`*`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.

3.13 Video capture devices (VideoIn API)

The VideoIn API supports generic video capture devices.

`macro expr PalVideoIn (`*`Index`*`);`

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

`macro expr PalVideoInCT (`*`Index`*`);`

see `macro expr PalXCT (`***`Index`***`);`

`macro expr PalVideoInCount ();`

see `macro expr PalXCount ();`

`macro expr PalVideoInUniqueCount ();`

see `macro expr PalXUniqueCount ();`

`macro proc PalVideoInRequire (`*`Count`*`);`

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

`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.

`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.

`macro expr PalVideoInGetMaxColorWidthCT ();`

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.

`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.

`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.

`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.

`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`).

3. Resource-specific API

`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`)

`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`)

`macro proc PalVideoInRun (HandleCT, ClockRate);`

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

`macro proc PalVideoInReset (Handle);`

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

`macro proc PalVideoInEnable (Handle);`

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

`macro proc PalVideoInDisable (Handle);`

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

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.

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 PalX (***Index***);

macro expr PalVideoOutCT (*Index*);

see macro expr PalXCT (***Index***);

macro expr PalVideoOutCount ();

see macro expr PalXCount ();

macro expr PalVideoOutUniqueCount ();

see macro expr PalXUniqueCount ();

macro proc PalVideoOutRequire (*Count*);

see macro proc PalXRequire (***Count***);

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 handle the optimal video output 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.

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 PalVideoOutGetColorWidthCT (`*`HandleCT`*`);`

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

`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.

`macro expr PalVideoOutGetVisibleY (`*`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.

`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.

`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.

`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`).

`macro expr PalVideoOutGetVisibleYCT (`*`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`)

`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`)

`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`)

`macro proc PalVideoOutRun (`*`HandleCT`*`, `*`ClockRate`*`);`

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

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

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

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

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

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

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

`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()`.

`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()`.

`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.

`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.

`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()`.

3.15 Audio input devices (AudioIn API)

The AudioIn API supports generic audio input devices.

`macro expr PalAudioIn (`*`Index`*`);`

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

`macro expr PalAudioInCT (`*`Index`*`);`

see `macro expr PalXCT (`***`Index`***`);`

`macro expr PalAudioInCount ();`

see `macro expr PalXCount ();`

`macro expr PalAudioInUniqueCount ();`

see `macro expr PalXUniqueCount ();`

`macro proc PalAudioInRequire (`*`Count`*`);`

see `macro expr PalXRequire (`***`Count`***`);`

`macro expr PalAudioInGetMaxDataWidthCT ();`

see `macro expr PalXGetMaxDataWidthCT ();`

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

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

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

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

`macro proc PalAudioInRun (`*`HandleCT`*`, `*`ClockRate`*`);`

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

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

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

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

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

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

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

`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.

`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.

3.16 Audio output devices (AudioOut API)

The AudioOut API supports generic audio output devices.

`macro expr PalAudioOut (`*`Index`*`);`

see macro expr PalX (***Index***);

`macro expr PalAudioOutCT (`*`Index`*`);`

see macro expr PalXCT (***Index***);

`macro expr PalAudioOutCount ();`

see macro expr PalXCount ();

`macro expr PalAudioOutUniqueCount ();`

see macro expr PalXUniqueCount ();

`macro proc PalAudioOutRequire (`*`Count`*`);`

see macro expr PalXRequire (***Count***);

`macro expr PalAudioOutGetMaxDataWidthCT ();`

see macro expr PalXGetMaxDataWidthCT ();

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

see macro expr PalXGetDataWidth (***Handle***);

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

see macro expr PalXGetDataWidthCT (***HandleCT***);

macro proc PalAudioOutRun (*HandleCT*, *ClockRate*);
see macro proc PalXRun (***HandleCT***, ***ClockRate***);

macro proc PalAudioOutReset (*Handle*);
see macro proc PalXReset (***Handle***);

macro proc PalAudioOutEnable (*Handle*);
see macro proc PalXEnable (***Handle***);

macro proc PalAudioOutDisable (*Handle*);
see macro proc PalXDisable (***Handle***);

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.

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

Arguments ***Handle***: PalHandle to an AudioOut resource.

Left, ***Right***: Expressions of type signed (PalAudioOutGetMaxDataWidthCT ()).

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.

3.17 Ethernet devices (Ethernet API)

The Ethernet API supports generic Ethernet interface devices.

macro expr PalEthernet (*Index*);
see macro expr PalX (***Index***);

macro expr PalEthernetCT (*Index*);
see macro expr PalXCT (***Index***);

macro expr PalEthernetCount ();
see macro expr PalXCount ();

macro expr PalEthernetUniqueCount ();
see macro expr PalXUniqueCount ();

macro proc PalEthernetRequire (*Count*);
see macro expr PalXRequire (***Count***);

macro proc PalEthernetRun (*HandleCT*, *MACAddress*, *ClockRate*);

Arguments ***HandleCT***: constant PalHandle to a PalEthernet resource.

MACAddress: Ethernet MAC address to be used by the network chip, of type unsigned 48.

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

Timing Does not terminate in normal use.

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

macro proc PalEthernetReset (*Handle*);
see macro proc PalXReset (***Handle***);

macro proc PalEthernetEnable (*Handle*);
see macro proc PalXEnable (***Handle***);

macro proc PalEthernetDisable (*Handle*);
see macro proc PalXDisable (***Handle***);

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 received packet, excluding the header.

ErrorPtr: Pointer to data of type 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 the 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.

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.

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.

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.

`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.

`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.

Index

API
- audio input 35
- audio output 36
- block storage 24
- buttons 13
- data I/O 14
- Ethernet 37
- LED macros 12
- parallel port 15
- PS/2 ports 16
- RAM 16, 18, 20, 22
- resource handles 6
- RS-232 15
- serial ports 15, 16
- seven-segment 12
- switches 13
- version numbers 6
- video input 26
- video output 30

audio input API 35
audio output API 36
buttons API 13
data I/O API 14
errors
- detecting 8

Ethernet API 37
header files 5
ID codes 7
LED API 12
library files 5
memory
- block 24
- fast RAM 16
- Flash 24
- generic 22
- pipelined 18, 20
- PL1 18
- PL2 20
- single cycle 16, 18, 20
- slow RAM 22

pal.hch 5
PalAudioIn 35
PalAudioInCount 35
PalAudioInCT 35
PalAudioInDisable 35
PalAudioInEnable 35
PalAudioInGetDataWidth 35
PalAudioInGetMaxDataWidthCT 35
PalAudioInRead 36
PalAudioInRequire 35
PalAudioInReset 35
PalAudioInRun 35
PalAudioInSetSampleRate 36
PalAudioInUniqueCount 35
PalAudioOut 36
PalAudioOutCount 36
PalAudioOutCT 36
PalAudioOutDisable 37
PalAudioOutEnable 37
PalAudioOutGetMaxDataWidthCT 36
PalAudioOutRequire 36
PalAudioOutReset 37
PalAudioOutRun 37
PalAudioOutSetSampleRate 37
PalAudioOutUniqueCount 36
PalAudioOutWrite 37
PalBlockStore 24
PalBlockStoreCount 24

Index

PalBlockStoreCT 24
PalBlockStoreDisable 25
PalBlockStoreEnable 25
PalBlockStoreEraseBlock 26
PalBlockStoreGetAddressWidth 25
PalBlockStoreGetAddressWidthCT 25
PalBlockStoreGetBlockLength 25
PalBlockStoreGetBlockLengthCT 25
PalBlockStoreGetDataWidth 25
PalBlockStoreGetDataWidthCT 25
PalBlockStoreGetMaxAddressWidthCT 24
PalBlockStoreGetMaxBlockLengthCT 24
PalBlockStoreGetMaxDataWidthCT 24
PalBlockStoreRead 25
PalBlockStoreRequire 24
PalBlockStoreReset 25
PalBlockStoreRun 25
PalBlockStoreUniqueCount 24
PalBlockStoreWrite 26
PalDataPort 14
PalDataPortCount 14
PalDataPortCT 14
PalDataPortDisable 14
PalDataPortEnable 14
PalDataPortGetDataWidth 14
PalDataPortGetDataWidthCT 14
PalDataPortGetMaxDataWidthCT 14
PalDataPortRead 15
PalDataPortRequire 14
PalDataPortReset 14
PalDataPortRun 14
PalDataPortUniqueCount 14
PalDataPortWrite 15
PalErrorCode 8
PalErrorHandlerRun 8
PalEthernet 37
PalEthernetCount 37
PalEthernetCT 37
PalEthernetDisable 38
PalEthernetEnable 38
PalEthernetRead 39
PalEthernetReadBegin 38
PalEthernetReadEnd 39
PalEthernetRequire 38
PalEthernetReset 38
PalEthernetRun 38
PalEthernetUniqueCount 37
PalEthernetWrite 40
PalEthernetWriteBegin 39
PalEthernetWriteEnd 40
PalFastRAM 16
PalFastRAMCount 16
PalFastRAMCT 16
PalFastRAMDisable 18
PalFastRAMEnable 18
PalFastRAMGetAddressWidth 17
PalFastRAMGetAddressWidthCT 17
PalFastRAMGetDataWidth 17
PalFastRAMGetDataWidthCT 17
PalFastRAMGetMaxAddressWidthCT 17
PalFastRAMGetMaxDataWidthCT 17
PalFastRAMRead 18
PalFastRAMRequire 17
PalFastRAMReset 18
PalFastRAMRun 17
PalFastRAMUniqueCount 16
PalFastRAMWrite 18
PalHandle 6
PalLED 12
PalLEDCount 12
PalLEDCT 12
PalLEDRequire 12
PalLEDUniqueCount 12
PalLEDWrite 12
PalParallelPort 15
PalParallelPortCount 15

Index

PalParallelPortCT 15
PalParallelPortRequire 15
PalParallelPortUniqueCount 15
PalPL1RAM 18
PalPL1RAMCount 18
PalPL1RAMCT 18
PalPL1RAMDisable 19
PalPL1RAMEnable 19
PalPL1RAMGetAddressWidth 19
PalPL1RAMGetAddressWidthCT 19
PalPL1RAMGetDataWidth 19
PalPL1RAMGetDataWidthCT 19
PalPL1RAMGetMaxAddressWidthCT 19
PalPL1RAMGetMaxDataWidthCT 19
PalPL1RAMPortRequire 19
PalPL1RAMRead 20
PalPL1RAMReset 19
PalPL1RAMRun 19
PalPL1RAMSetReadAddress 19
PalPL1RAMSetWriteAddress 20
PalPL1RAMUniqueCount 19
PalPL1RAMWrite 20
PalPL2RAM 20
PalPL2RAMCount 20
PalPL2RAMCT 20
PalPL2RAMDisable 21
PalPL2RAMEnable 21
PalPL2RAMGetAddressWidth 21
PalPL2RAMGetAddressWidthCT 21
PalPL2RAMGetDataWidth 21
PalPL2RAMGetDataWidthCT 21
PalPL2RAMGetMaxAddressWidthCT 21
PalPL2RAMGetMaxDataWidthCT 21
PalPL2RAMPortRequire 21
PalPL2RAMRead 22
PalPL2RAMReset 21
PalPL2RAMRun 21
PalPL2RAMSetReadAddress 21
PalPL2RAMSetWriteAddress 22
PalPL2RAMUniqueCount 21
PalPL2RAMWrite 22
PalPS2Port 16
PalPS2PortCount 16
PalPS2PortCT 16
PalPS2PortRequire 16
PalPS2PortUniqueCount 16
PalRS232Port 15
PalRS232PortCount 16
PalRS232PortCT 16
PalRS232PortRequire 16
PalRS232PortUniqueCount 16
PalSevenSeg 12
PalSevenSegCount 12
PalSevenSegCT 12
PalSevenSegDisable 13
PalSevenSegEnable 13
PalSevenSegRequire 13
PalSevenSegUniqueCount 12
PalSevenSegWriteDigit 13
PalSevenSegWriteShape 13
PalSlowRAM 22
PalSlowRAMCount 22
PalSlowRAMCT 22
PalSlowRAMDisable 23
PalSlowRAMEnable 23
PalSlowRAMGetAddressWidth 23
PalSlowRAMGetAddressWidthCT 23
PalSlowRAMGetDataWidth 23
PalSlowRAMGetDataWidthCT 23
PalSlowRAMGetMaxAddressWidthCT 23
PalSlowRAMGetMaxDataWidthCT 23
PalSlowRAMRead 23
PalSlowRAMRequire 23
PalSlowRAMReset 23
PalSlowRAMRun 23
PalSlowRAMUniqueCount 23

Index

PalSlowRAMWrite 24
PalSwitch 13
PalSwitchCount 13
PalSwitchCT 13
PalSwitchRead 14
PalSwitchRequire 14
PalSwitchUniqueCount 13
PalVendorCode 7
PalVendorRelease 7
PalVendorString 7
PalVersionMajor 6
PalVersionMinor 6
PalVersionRequire 7
PalVideoIn 26
PalVideoInCount 26
PalVideoInCT 26
PalVideoInDisable 29
PalVideoInEnable 29
PalVideoInGetColorWidth 28
PalVideoInGetColorWidthCT 29
PalVideoInGetMaxColorWidthCT 27
PalVideoInGetMaxXWidthCT 27
PalVideoInGetMaxYWidthCT 27
PalVideoInGetXWidth 27
PalVideoInGetXWidthCT 28
PalVideoInGetYWidth 28
PalVideoInGetYWidthCT 29
PalVideoInRead 30
PalVideoInRequire 27
PalVideoInReset 29
PalVideoInRun 29
PalVideoInUniqueCount 27
PalVideoOut 30
PalVideoOutCount 30
PalVideoOutCT 30
PalVideoOutDisable 34
PalVideoOutEnable 34
PalVideoOutGetColorWidth 31
PalVideoOutGetColorWidthCT 31
PalVideoOutGetHBlank 34
PalVideoOutGetMaxColorWidthCT 31
PalVideoOutGetMaxXWidthCT 30
PalVideoOutGetMaxYWidthCT 31
PalVideoOutGetTotalX 32
PalVideoOutGetTotalXCT 33
PalVideoOutGetTotalY 32
PalVideoOutGetTotalYCT 33
PalVideoOutGetVBlank 34
PalVideoOutGetVisibleX 31
PalVideoOutGetVisibleXCT 32
PalVideoOutGetVisibleY 31
PalVideoOutGetVisibleYCT 33
PalVideoOutGetX 34
PalVideoOutGetXWidth 31
PalVideoOutGetXWidthCT 31
PalVideoOutGetY 34
PalVideoOutGetYWidth 31
PalVideoOutGetYWidthCT 31
PalVideoOutOptimalCT 30
PalVideoOutRequire 30
PalVideoOutReset 34
PalVideoOutRun 33
PalVideoOutUniqueCount 30
PalVideoOutWrite 35
PalX 9
PalXCount 8
PalXCT 9
PalXDisable 11
PalXEnable 11
PalXGetDataWidth 10
PalXGetDataWidthCT 10
PalXGetMaxDataWidthCT 9
PalXRead 11
PalXRequire 9
PalXReset 10
PalXRun 10

Index

PalXUniqueCount 9
PalXWrite 11
parallel port API 15
PS/2 API 16
RAM
  pipelined 18, 20
  PL1 18
  PL2 20
  single cycle 16, 18, 20
RAM API 16, 18, 20, 22
resetting resources 10
resources
  enabling and disabling 11
  handles 6, 9
  number of 8
  properties 9
  reading from 11
  resetting 10
  running 10
  writing to 11
RS-232 API 15
serial ports API 15, 16
seven-segment API 12
switches API 13
vendor ID codes 7
version numbers
  API 6
  libraries 7
video input API 26
video output API 30

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

Celoxica in Europe	Celoxica in Japan	Celoxica in Asia Pacific	Celoxica in the Americas
T: +44 (0) 1235 863 656	T: +81 (0) 45 331 0218	T: +65 6896 4838	T: +1 800 570 7004
E: sales.emea@celoxica.com	E: sales.japan@celoxica.com	E: sales.apac@celoxica.com	E: sales.america@celoxica.com