Texas Instruments

DSP/BIOS PAL SYS VLYNQ Device Driver

User's Manual

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address:
Texas Instruments
Post Office Box 655303, Dallas, Texas 75265

Preface

Read This First

About This Manual

The API reference guide serves as a software programmer's handbook for working with the VLYNQ device driver modules. This reference guide provides necessary information regarding how to use these modules in user systems and applications.

Abbreviations

Table of Abbreviations

Abbreviation	Description
API	Application Programming Interface
ISR	Interrupt Service Routine
OS	Operating System
SOC	System On Chip

Revision History

Date	Author	Comments	Version
June 17, 2006	Saloni Shah	Created the document	1.0
August 7, 2006	Saloni Shah	Changes as per Release 0.1.1	1.1
September 21, 2006	Saloni Shah	BIOS version change	1.2
October 8, 2006	Rinkal Shah	Review comments closed	1.3
December 1, 2006	Rinkal Shah	Modified for the release 0.3.0	1.4
March 24, 2007	Anuj Aggarwal	Modifying according to the VLYNQ latest code base	1.5
June 5, 2007	Anuj Aggarwal	Modified for the release 0.6.0	1.6
July 11, 2007	Anuj Aggarwal	BIOS version change	1.7
May 21, 2008	Chandan Nath	Updated for adding compiler switches in build options	1.8

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Topic

1.1. Introduction

This document is an API reference guide on VLYNQ Device Driver.

1.2. H/W S/W Support

This VLYNQ Device driver has been developed for the following DSP/BIOS operating system and using the TI supplied Chip Support Library. For more details on the version numbers refer to the release notes in the root of the installation.

1.3. Driver Components

The driver is constituted of following sub components:

VLYNQ PAL API's – OS Independent part of VLYNQ Driver Core

VLYNQ CSL – The low-level VLYNQ h/w abstraction module

System components:

PALOS - BIOS Abstraction

1.4. Driver Capabilities

Driver has to be configured by the application. There is no default driver configuration.

1.5. System Requirements

Refer system level release notes for tools and BIOS versions.

CHAPTER 2

INSTALLATION GUIDE

Topic

2.1. Component Folder

Upon installing the VLYNQ driver the following directory structure is found in the driver's directory.

Figure 1. VLYNQ Driver Directory Structure

This top level vlynq folder contains vlynq driver psp header file and XDC package files (package.bld, package.xdc and package.xs)

- **build:** This folder contains vlynq driver library project file. The generated driver library shall be included in the application where VLYNQ driver have to be used.
- **docs:** This folder contains architecture document, datasheet, release notes and user guide.

  Architecture document contains the driver details which can be helpful for the developers as well as consumers to understand the driver design.

  Datasheet gives the idea about the memory consumption by the driver and description of the top level APIs.

  Release Note gives the details about system requirements, steps to Install/Uninstall the package.This document list the known issues of the driver.

  User Guide provides information about how to use the driver. It contains description of sample applications which guide the end user to make their applications using this driver.
- **Lib:** This folder contains libraries generated in all the configuration modes(debug, idebug, irelease and release)
- **Package:** This folder contains files generated by XDC tool.
- **src:** This folder contains vlynq driver source files. It also contains header files that are used by the driver.

2.2. Build

This section describes for each supported target environment, the applicable build options, supported configurations and how selected, the featured capabilities and how enabled, the allowed user customizations for the software to be installed and how the same can be realized.

The component might be delivered to user in different formats:

- Source-less ie., binary executables and object libraries only
- Source-inclusive ie., The entire source code used to implement the driver is included in the delivered product
- Source-selective ie., Only a part of the overall source is included. This delivery mechanism might be required either because; certain parts of the driver require source-level extensions and/or customization at the user's end or because, specific parts of the driver is exposed to user at the source-level to insure user's software development.

When source is included as part of the product delivery, the CCS project file is provided as part of the package. When object format is distributed, the driver header files are part of the "inc" folder and the driver library is provided in /drivers/lib folder.

2.3. Build Options

This driver does not have any specific build option at the time of writing of this manual.

The build folder contains a CCS project file that builds the driver into a library for debug and release mode.

Following compiler switches are used to compile for different options.

- **_DEBUG**
  This is used as a flag to compiler whether to include the debug statements inserted in the code into the final image. This flag helps to build DEBUG image of the program. For RELEASE images this is not passed to the compiler.
- **CHIP_XXXX**
  The CSL layer is written in a common file for all the variants of a SOC. This flag differentiates the variant we are compiling for, for eg - CHIP_DM648, and the CSL definitions for that variant appropriately gets defined for regs base addresses, num of ports of a peripheral etc.
- **VLYNQ_INSTRUMENTATION_ENABLED**
  This flag is passed to the compiler to include the instrumentation code parts into the final image/lib of the program. This helps build the iRelease/iDebug versions of the image/lib with a common code base
- **VLYNQ_LOOPBACK_MODE**

This flag is passed to the compiler when vlynq loopback mode is used.

CHAPTER 3

VLYNQ

This chapter describes the functions, data structures, enumerations and macros for the List module.

Topic

3.1. Functions

This section lists the functions available in the PSP module.

3.1.1 PAL_sysVlynqInit ( ) - Initialize the VLYNQ control module.

PAL_Result PAL_sysVlynqInit (Void)

Parameters:

None

Returns:

Returns PAL_VLYNQ_OK.

3.1.2 PAL_sysVlynqInitSoc( ) - Initialize the VLYNQ control module.

PAL_VLYNQ_HND* PAL_sysVlynqInitSoc(PAL_VLYNQ_CONFIG_T* config)

Parameters:

config The collection of configuration parameters; to initialize the VLYNQ interface.

Note: **If the VLYNQ control module being initialized happens to be a non-root entity, then, it is necessary and MUST that this VLYNQ (identified by PAL_VLYNQ_HND) has to be appended to the chain before this handle can be used for any purpose.**

Returns:

On failure, it returns a NULL, otherwise, it returns a valid handle to the instance so initialized.

3.1.3 PAL_sysVlynqCleanUp( ) – Un-Initialize the VLYNQ control module.

PAL_Result PAL_sysVlynqCleanUp (PAL_VLYNQ_HND* vlynq)

Parameters:

vlynq Handle to the VLYNQ module instance; would have been returned by PAL_sysVlynqInitSoc ( ). Consequence of using this handle after a call to this function is undefined.

Returns:

0 – On success.

-1 – On failure.

3.1.4 PAL_sysVlynqDevCreate( ) – Creates a device reference.

PAL_VLYNQ_DEV_HND*	**PAL_sysVlynqDevCreate**	**(**	**PAL_VLYNQ_HND***	**vlynq,**
			char*	**name,**
			Uint32	**instance,**
			Int32	**reset_bit**
			Bool	**peer**
		)		

Parameters:

vlynq	The VLYNQ module instance to which a device has to be added; this value would have been returned by `PAL_sysVlynqInitSoc( ).`
name	Name of the device; shall not exceed 30 characters and shall be NULL terminated.
instance	The instance of the device; it is 0 based.
reset_bit	The reset bit for the VLYNQ device in the perspective of the system. Valid only for the on-board device. If not applicable, a value of –1 shall be set.
peer	Identifies whether the device is connected to the local or peer VLYNQ module.

Returns:
On Failure, returns NULL else a Valid Handle to the device instance.

3.1.5 PAL*_sysVlynqDevDestroy( ) – Destroys the device reference.*

PAL_Result	**PAL_sysVlynqDevDestroy**	**(**	**PAL_VLYNQ_DEV_HND***	**vlynq_dev**
		)		

Parameters:

vlynq_dev	The VLYNQ device instance which would have been returned by `PAL_sysVlynqDevCreate( ).` Use of this handle after calling this function is undefined.

Returns:
0 - On success
-1 – On failure.

3.1.6 PAL_sysVlynqMapRegion( ) – Map the memory regions of the device.

PAL_Result	**PAL_sysVlynqMapRegion**	**(**	**PAL_VLYNQ_HND***	**vlynq,**
			Bool	**remote,**
			Uint32	**region_id,**
			Uint32	**rx_offset,**
			Uint32	**rx_size,**
			PAL_VLYNQ_DEV_HND*	**vlynq_dev**
		)		

Parameters:

vlynq	The VLYNQ instance, which on which regions have to be mapped. This value would have been returned by `PAL_sysVlynqInitSoc( ).`
remote	Identifies whether the region to be mapped is on remote/peer or local VLYNQ. A device connected to peer VLYNQ **exports** its memory by programming the RX Region registers of the peer VLYNQ and **imports** the memory regions by programming the RX Registers of the local VLYNQ. So, for a "peer" device, it should set the remote flag as true (1) to export its regions. Similarly, a device connected to local VLYNQ **exports** its memory by programming the RX Region registers of the local VLYNQ and **imports** region by programming the peer VLYNQ. So, setting remote as true (1) shall mean that the device wants to imports memory regions. The parameter has to be viewed in the context of the device for which memory mapping is being envisaged. 0 (false) => local 1 (true) => remote
region_id	Identifier of the RX region; valid values are from 0 to 3.
rx_offset	The offset of the region in the physical memory map of the local or remote SOC.
rx_size	The size of the region to be mapped.
vlynq_dev	The device instance for which certain regions have to be mapped. This value would have been returned by `PAL_sysVlynqDevCreate( ).`

Returns:
0 – on success.

-1 – On failure

3.1.7 PAL_sysVlynqMappedRegion ( ) – Return the Mapped Region configuration for Local/Peer.

PAL_Result	**PAL_sysVlynqMapRegion**	**(**	**PAL_VLYNQ_HND***	**vlynq,**
			Bool	**remote,**
			Uint32	**region_id ,**
			Uint32	**rx_offset,**
			Uint32	**rx_size,**
			PAL_VLYNQ_DEV_HND*	**vlynq_dev)**

Parameters:

vlynq	The VLYNQ instance, which on which regions have to be mapped. This value would have been returned by `PAL_sysVlynqInitSoc( ).`
remote	Identifies whether the region to be mapped is on remote/peer or local VLYNQ. A device connected to peer VLYNQ **exports** its memory by programming the RX Region registers of the peer VLYNQ and **imports** the memory regions by programming the RX Registers of the local VLYNQ. So, for a "peer" device, it should set the remote flag as true (1) to export its regions. Similarly, a device connected to local VLYNQ **exports** its memory by programming the RX Region registers of the local VLYNQ and **imports** region by programming the peer VLYNQ. So, setting remote as true (1) shall mean that the device wants to imports memory regions. The parameter has to be viewed in the context of the device for which memory mapping is being envisaged. 0 (false) => local 1 (true) => remote
region_id	Identifier of the RX region; valid values are from 0 to 3.
rx_offset	The offset of the region in the physical memory map of the local or remote SOC.
rx_size	The size of the region to be mapped.
vlynq_dev	The device instance for which certain regions have to be mapped. This value would have been returned by `PAL_sysVlynqDevCreate( ).`

Returns:
0 – on success.

-1 – On failure

3.1.8 PAL_sysVlynqUnMapRegion( ) – UnMap the memory regions of the device.

PAL_Result	**PAL_sysVlynqUnMapRegion**	**(**	**PAL_VLYNQ_HND***	**vlynq,**
			Bool	**remote,**
			Uint32	**region_id,**
			PAL_VLYNQ_DEV_HND*	**vlynq_dev**
		)		

Parameters:

vlynq	The VLYNQ instance, on which previously mapped regions have to be un-mapped.
remote	Identifies whether the region to be mapped is on remote/peer or local VLYNQ. The parameter has to be viewed in the context of the device for which memory mapping is being envisaged. 0 (false) => local 1 (true) => remote
region_id	Identifier of the RX region; valid values are from 0 to 3.
vlynq_dev	The device instance for which certain regions have been previously mapped; and now are being un-mapped.

Returns:
0 – on success.

-1 – On failure

3.1.9 PAL_sysVlynqMapIrq( ) – Maps the IRQ hardware line onto the VLYNQ.

PAL_Result	**PAL_sysVlynqMapIrq**	**(**	**PAL_VLYNQ_HND***	**vlynq,**
			Uint32	**irq_hw_line,**
			Uint32	**irq,**
			PAL_VLYNQ_DEV_HND*	**vlynq_dev**
		)		

Parameters:

vlynq	The vlynq instance on which the IRQ mapping is to be carried out. . This value would have been returned by `PAL_sysVlynqInitSoc( )`.
irq_hw_line	Identifies the hardware vector line (in the perspective of the VLYNQ module), which runs from the device to the VLYNQ module. The valid values are 0 to 7 (both inclusive).
irq	Identifies the IRQ number to which the interrupt hardware line shall be mapped. The

	valid values are 0 to 31 (both inclusive). It is the responsibility of the user to ensure that there is a unique irq_hw_line to irq mapping within a VLYNQ chain.
vlynq_dev	The device instance for which certain hardware interrupts have to be mapped on the VLYNQ; this value would have been returned by `PAL_sysVlynqDevCreate()`.

Returns:
0 – on success.
-1 – On failure.

3.1.10 PAL_sysVlynqUnMapIrq( ) – UnMaps the IRQ hardware line.

```
PAL_Result    PAL_sysVlynqUnMapIrq    (    PAL_VLYNQ_HND*          vlynq,
                                           Uint32                  irq,
                                           PAL_VLYNQ_DEV_HND*      vlynq_dev
                                      )
```

Parameters:

vlynq	The VLYNQ instance on which the un-mapping has to be carried out.
irq	Identifies the IRQ number for which the interrupt hardware line shall be un-mapped. The valid values are 0 to 31 (both inclusive).
vlynq_dev	The device instance for which certain hardware interrupts are required to be un-mapped on VLYNQ; this value would have been returned by `PAL_sysVlynqDevCreate()`.

Returns:
0 – on success.
-1 – On failure.

3.1.11 PAL_sysVlynqChainAppend( ) –Append to the VLYNQ chain.

```
PAL_Result    PAL_sysVlynqChainAppend    (    PAL_VLYNQ_HND*    this,
                                              PAL_VLYNQ_HND*    to
                                         )
```

Parameters:

this	The VLYNQ module instance, which has to be added to the chain (away from the root). This value would have been returned by `PAL_sysVlynqInitSoc( ).`
to	The VLYNQ module instance, to which "this" has to be chained (away from the root). . This value would have been returned by `PAL_sysVlynqInitSoc( ).`

Returns:
0 – on success.
-1 – On failure.

3.1.12 PAL_sysVlynqAddDevice( ) –Add the device reference into VLYNQ.

```
PAL_Result PAL_sysVlynqAddDevice ( PAL_VLYNQ_HND* vlynq,
                                   PAL_VLYNQ_DEV_HND* vlynq_dev,
                                   Bool peer
                                 )
```

Parameters:

vlynq	The VLYNQ module instance, to which the device instance has to be added. This value would have been returned by `PAL_sysVlynqInitSoc( ).`
vlynq_dev	The device instance, which has to be added into the VLYNQ. This value would have been returned by `PAL_sysVlynqDevCreate().`
peer	The flag whether the device is associated with the local VLYNQ or the peer.

Returns:
0 – on success.
-1 – On failure.

3.1.13 PAL_sysVlynqRemoveDevice( ) – Removes the device reference from VLYNQ.

```
PAL_Result PAL_sysVlynqRemoveDevice ( PAL_VLYNQ_HND* vlynq,
```

	PAL_VLYNQ_DEV_HND* vlynq_dev
	)

Parameters:

vlynq	The VLYNQ module instance, from which the device instance has to be removed.
vlynq_dev	The device instance, which is to be removed from the VLYNQ.

Returns:

0 – On Success

-1- On Failure

3.1.14 PAL_sysVlynqChainUnAppend( ) – Remove (the tail) from the VLYNQ chain.

PAL_Result	**PAL_sysVlynqChainUnAppend**	**(**	**PAL_VLYNQ_HND***	**this,**
			PAL_VLYNQ_HND*	**from**
		)		

Parameters:

this	The VLYNQ module instance, which has to be removed from the chain (away from the root).
from	The VLYNQ module instance, from which “this” has to be removed (away from the root).

Returns:

0 – On Success

-1 – On Failure

3.1.15 PAL_sysVlynqRootIsr( ) – The Root ISR; register it with the system.

PAL_Result	**PAL_sysVlynqRootIsr**	**(**	**Int32**	***p_vlynq**
		)		

Parameters:

p_vlynq	Pass this parameter along while attaching the HWI to the PAL_sysVlynqRootIsr(). It is the address of the VLYNQ handle of the root VLYNQ module running.

Returns:

0 – On Success

-1 – On Failure

3.1.16 PAL_sysVlynqDevFind( ) – Get the handle for the device.

PAL_VLYNQ_DEV_HND* PAL_sysVlynqDevFind (char *name, Uint8 instance)

Parameters:

name	The name of the device whose information is being sought. This should same as that used in the `PAL_sysVlynqDevCreate( ).`
instance	The instance of the remote device. This should same as that used in the `PAL_sysVlynqDevCreate( ).`

Returns:
NULL, if no device could be found or a valid handle.

3.1.17 PAL_sysVlynqDevGetVlynq( ) – Get the VLYNQ for this device.

PAL_VLYNQ_HND* PAL_sysVlynqDevGetVlynq (PAL_VLYNQ_DEV_HND* dev)

Parameters:

dev	The device whose associated VLYNQ control module is being sought. This would have been returned by `PAL_sysVlynqDevFind( ).`

Returns:

NULL on error or a valid VLYNQ handle.

3.1.18 PAL_sysVlynqGetDevBase( ) – Get the physical base address of the device.

PAL_Result	**PAL_sysVlynqGetDevBase**	**(**	**PAL_VLYNQ_ HND***	**vlynq,**
			Uint32	**offset,**
			Uint32	***base_addr**
			PAL_VLYNQ_DEV_HND*	**dev**
		)		

Parameters:

vlynq	The instance on which the physical base address is being sought.
offset	The offset (in bytes) into the remote device memory map.
base_addr	The placeholder for the evaluated physical base address of the remote device in the context of the root SOC. The caller manages the memory for this.
dev	The handle to remote device instance; would have been returned by `PAL_sysVlynqDevFind( )`. The offset is being sought for this device.

Returns:

0 – On Success

-1 – On Failure

3.1.19 PAL_sysVlynqDevFindIrq( ) – Get the mapped interrupts of the device..

PAL_Result	**PAL_sysVlynqDevFindIrq**	**(**	**PAL_VLYNQ_DEV_HND***	**dev,**
			Uint8	**irqs[ ],**
			Uint32	**num_irqs**
		)		

Parameters:

dev	The handle to remote device instance; would have been returned by `PAL_sysVlynqDevFind( )`.
irqs	The placeholder for the identifiers of the IRQ(s) vectors in the perspective of the chain. The caller is responsible for managing the memory. Typically, the caller shall allocate memory for atleast

	"num_irqs".
num_irqs	The number of interrupts to be read; this should be same as the number of interrupts for which placeholder has been provided.

Returns:
0 – on success.
-1 – If the placeholder is inadequate.

-2 – Other errors.

3.1.20 PAL_sysVlynqDevGetResetBit( ) – Get the reset bit of the device.

PAL_Result	**PAL_sysVlynqDevGetResetBit**	**(**	**PAL_VLYNQ_DEV_HND***	**dev,**
			Uint32	***reset_bit**
		)		

Parameters:

dev	The handle to remote device instance; would have been returned by `PAL_sysVlynqDevFind( ).`
*reset_bit	The placeholder for the identifiers of reset bit. **Only for on board remote devices for which hardware reset provisions have been made.**

Returns:
0 – on success.

-1 – Error.

3.1.21 PAL_sysVlynqAddIsr( ) – Install the ISR for the device.

PAL_Result	**PAL_sysVlynqAddIsr**	**(**	**PAL_VLYNQ_HND***	**vlynq,**
			Uint32	**irq,**
			PAL_VLYNQ_DEV_ISR_FN*	**dev_isr,**
			PAL_VLYNQ_DEV_ISR_PARAM_GRP_T*	**isr_param**
		)		

Parameters:

vlynq	The VLYNQ module instance for whose associated remote device, an isr is being installed.
irq	One of the identifier of the IRQ as returned by `PAL_sysVlynqGetDevIrq( )or` VLYNQ module specific interrupt..
dev_isr	The OS specific function pointer (ISR handler); the type of the function PAL_VLYNQ_DEV_ISR_FN shall be defined for each of the OS by the user. Driver for the remote device shall load appropriate function instance. The number of parameters in the signature of this function should be exactly same as specified by PAL_VLYNQ_DEV_ISR_PARM_NUM.
isr_param	It is a collection of parameters for each instance of the remote device driver. The user shall define the type PAL_VLYNQ_DEV_ISR_PARAM_GRP_T for the collection of the parameters for each OS. The only caveat here is that the name of the members of the PAL_VLYNQ_DEV_ISR_PARAM_GRP_T should start (i.e. first member) with arg0 (inclusive) and should continue in sequence of arg1, arg2… up to arg9 (inclusive); it is assumed that no OS implementation shall require more than 10 callback ISR parameters. User shall define and specify the number of ISR parameter, PAL_VLYNQ_DEV_ISR_PARM_NUM as required for a specific OS implementation; this should not exceed 10. For a value, less than 10, the member names shall start as arg0, arg1… up to arg< PAL_VLYNQ_DEV_ISR_PARM_NUM– 1>. The keywords for member names here are arg0, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8 and arg9.

Returns:

0 – On Success

-1 – Error

3.1.22 PAL_sysVlynqRemoveIsr( ) – Uinstall the previously installed ISR.

PAL_Result	**PAL_sysVlynqRemoveIsr**	**(**	**PAL_VLYNQ_HND***	**vlynq,**
			Uint32	**irq,**
			PAL_VLYNQ_DEV_ISR_PARAM_GRP_T *	**isr_param**
		)		

Parameters:

vlynq	The VLYNQ module instance for whose associated remote device, an isr is being removed.
irq	The identifier of the IRQ as provided in the `PAL_sysVlynqAddIsr( ).`
isr_param	Same as that provided in the `PAL_sysVlynqAddIsr( ).`

Returns:
0 – on success.

-1 – Error.

3.1.23 PAL_sysVlynqDevCbRegister( ) – Register for the callbacks.

PAL_Result	**PAL_sysVlynqDevCbRegister**	**(**	**PAL_VLYNQ_DEV_HND***	**dev,**
			PAL_VLYNQ_DEV_DRV_CB_FN	**func,**
			void*	**this_driver**
		)		

Parameters:

dev	The handle to remote device instance; as returned by `PAL_sysVlynqDevFind( ).`
func	typedef int (*PAL_VLYNQ_DEV_CB_FN)(void* this_driver, Uint32 condition, Uint32 condition value); The callback function that is be called by implementation for local error, or remote error update conditions.
this_driver	The driver instance, which is registering for the callbacks from the VLYNQ implementation.

Returns:
0 – on success.

-1 – Error.

3.1.24 PAL_sysVlynqDevCbUnregister( ) – Unregister the callbacks.

PAL_Result	**PAL_sysVlynqDevCbUnregister**	**(**	**PAL_VLYNQ_DEV_HND***	**dev,**
			void*	**this_driver**
		)		

Parameters:

dev	The handle to remote device instance; as returned by `PAL_sysVlynqDevFind( ).`
this_driver	The driver instance, which is registered for the callbacks from the VLYNQ implementation.

Returns:
0 – on success.

-1 – Error.

3.1.25 PAL_sysVlynqIoctl( ) – Read/Write register of the VLYNQ module.

PAL_Result	**PAL_sysVlynqIoctl**	**(**	**PAL_VLYNQ_HND***	**vlynq,**
			Uint32	**cmd,**
			Uint32	**command_val**
		)		

Parameters:

vlynq	The VLYNQ module instance, whose register has to be controlled; this value would have been returned by `PAL_sysVlynqInitSoc().`
cmd	Various read and write commands to be carried out. Refer below for the 32 bit break up of the command.
Command_val	For write command(s) this provides the value to be written and for read operations it provides the placeholder for the value to be read.

Returns:
0 – on success.
-1 – On failure.

32 bit Command :

31:Bit Op	30: R/W*	29: Peer	28-24 Reserved	23-16 Major id	15-8 Reserved	7-0 Minor id

If Bit Op is not set, the major id identifies the commands for raw 32-bit accesses or any specific operation. For now, if major command is 32 bit accesses, then register id refer to minor id otherwise minor id are don't care.

If Bit Op is set, the major id identifies a specific register for a select bit operation. Minor id then shall identify the registers.

3.1.26 PAL_sysVlynqClockConfig( ) – Configures the Clock for the VLYNQ bridge.

```
PAL_Result  PAL_sysVlynqConfigClock  (  PAL_VLYNQ_HND*                  vlynq,
                                        PAL_VLYNQ_CLOCK_DIR_ENUM_T     local_clock_dir,
                                        PAL_VLYNQ_CLOCK_DIR_ENUM_T     peer_clock_dir,
                                        Uint8                          local_clock_div,
                                        Uint8                          peer_clock_div,
                                     )
```

Parameters:

vlynq	The VLYNQ module instance, whose register has to be controlled; this value would have been returned by `PAL_sysVlynqInitSoc()`.
local_clock_dir	The clock direction for the local VLYNQ. Refer to
peer_clock_dir	The clock direction for the peer VLYNQ. Refer to
local_clock_div	The divisor for the local clock. Valid values are 1 to 255.
peer_clock_div	The divisor for the peer clock. Valid values are 1 to 255.

Returns:
0 – on success.
-1 – On failure.

3.1.27 PAL_sysVlynqGetForIrq( ) – Get the VLYNQ for the IRQ.

PAL_VLYNQ_HND*	**PAL_sysVlynqGetForIrq**	**(**	**PAL_VLYNQ_HND***	**root,**
			Uint32	**irq**
		)		

Parameters:

root	The VLYNQ chain identifier (the root VLYNQ); the interrupt numbers are unique with in a chain.
irq	The interrupt number whose association with a VLYNQ module is being sought. The valid values are 0 to 31 (both inclusive).

Returns:
0 – on success.

-1 – Error.

3.1.28 PAL_sysVlynqSetIrqPol( ) – Set the polarity of the hardware IRQ line.

PAL_Result	**PAL_sysVlynqSetIrqPol**	**(**	**PAL_VLYNQ_HND***	**vlynq,**
			Uint32	**irq,**
			PAL_VLYNQ_IRQ_POL_ENUM_T	***polarity**
		)		

Parameters:

vlynq	The VLYNQ instance whose hardware interrupt line is to be set for polarity; .
irq	The interrupt number representing the mapped interrupt hardware line on the VLYNQ module. This should be unique for a given VLYNQ chain.
polarity	The polarity of the line.

Returns:
0 – on success.

-1 – Error.

3.1.29 PAL_sysVlynqSetIrqType( )- Set the type of the hardware IRQ line.

PAL_Result	**PAL_sysVlynqSetIrqType**	**(**	**PAL_VLYNQ_HND***	**vlynq,**
			Uint32	**irq,**
			PAL_VLYNQ_IRQ_TYPE_ENUM_T	**type**
		)		

Parameters:

vlynq	The VLYNQ instance whose hardware interrupt line is to be set for trigger type.
irq	The interrupt number representing the mapped interrupt hardware line on the VLYNQ module. This should be unique for a given VLYNQ chain.
type	The type of the interrupt line.

Returns:
0 – on success.

-1 – Error.

3.1.30 PAL_sysVlynqGetIrqPol ( ) – Get the polarity of the hardware IRQ line.

PAL_Result	**PAL_sysVlynqGetIrqPol**	**(**	**PAL_VLYNQ_HND***	**vlynq,**
			Uint32	**irq,**
			PAL_VLYNQ_IRQ_POL_ENUM_T*	**polarity**
		)		

Parameters:

vlynq	The VLYNQ instance whose hardware interrupt line is to being looked up for polarity setting.
Irq	The interrupt number representing the mapped interrupt hardware line on the VLYNQ module. This should be unique for a given VLYNQ chain.
polarity	The polarity of the line. The caller manages memory.

Returns:
0 – on success.

-1 – Error.

3.1.31 PAL_sysVlynqGetIrqType( ) – Get the type of the hardware IRQ type.

```
PAL_Result   PAL_sysVlynqGetIrqType   (   PAL_VLYNQ_HND*              vlynq,
                                          Uint32                     irq,
                                          PAL_VLYNQ_IRQ_POL_ENUM     type
                                          _T*
                                      )
```

Parameters:

vlynq	The VLYNQ instance whose hardware interrupt line is being looked up for trigger type.
Irq	The interrupt number representing the mapped interrupt hardware line on the VLYNQ module. This should be unique for a given VLYNQ chain.
type	The type of the interrupt line. The caller manages memory.

Returns:
0 – on success.

-1 – Error.

3.1.32 PAL_sysVlynqGetIrqCount( ) – Get the number of times this IRQ occurred.

```
Uint32   PAL_sysVlynqGetIrqCount   (   PAL_VLYNQ_HND*              vlynq,
                                       Uint32                     irq
                                       Uint32                     *count
                                   )
```

Parameters:

vlynq	The VLYNQ module instance for whose associated “irq” has to be queried for number of dispatches so far.
irq	The interrupt number representing the mapped interrupt hardware line on the VLYNQ module. This should be unique for a given VLYNQ chain.
count	Placeholder to store the number of times the interrupt “irq” has been raised. The value in the placeholder is valid only if the function returns success.

Returns:
0 – on success.

-1 – Error.

3.1.33 PAL_sysVlynqDisableIrq( ) – Disable the IRQ.

Uint32	**PAL_sysVlynqDisableIrq**	**(**	**PAL_VLYNQ_HND***	**vlynq,**
			Uint32	**irq**
		)		

Parameters:

vlynq	The VLYNQ module instance for whose associated “irq” hardware line is to be disabled for generating interrupts.
irq	The interrupt number representing the mapped interrupt hardware line on the VLYNQ module. This should be unique for a given VLYNQ chain.

Returns:
0 – on success.

-1 – Error.

3.1.34 PAL_sysVlynqEnableIrq( ) – Enable the IRQ.

Uint32	**PAL_sysVlynqEnableIrq**	**(**	**PAL_VLYNQ_HND***	**vlynq,**
			Uint32	**irq**
		)		

Parameters:

vlynq	The VLYNQ module instance for whose associated “irq” hardware line is to be enabled for generating interrupts.
irq	The interrupt number representing the mapped interrupt hardware line on the VLYNQ module. This should be unique for a given VLYNQ chain.

Returns:
0 – on success.

-1 – Error.

3.1.35 PAL_sysVlynqGetLinkStatus( ) – Get the status of the of the VLYNQ module.

Bool	PAL_sysVlynqGetLinkStatus	(	PAL_VLYNQ_HND*	vlynq
		)		

Parameters:

vlynq	The VLYNQ module instance, whose link status with the peer is being requested.

Returns:
1 – on link.
0 – on link failure.

3.1.36 PAL_sysVlynqGetNumRoot( ) – Get the number of the root VLYNQ(s).

Int32	PAL_sysVlynqGetNumRoot	(	void
		)	

Returns:
The number of root VLYNQ modules on the SOC.

A value of 0 means no VLYNQ.
1 means a root index of 0,
2 mean indices0 and 1.

3.1.37 PAL_sysVlynqGetRoot( ) – Get the handle to the specified root VLYNQ.

PAL_VLYNQ_HND*	PAL_sysVlynqGetRoot	(	Int32	index
		)		

Parameters:

index	0 based. Starts at 0 and extends to (inclusive of) one less number of roots returned by `PAL_sysVlynqGetNumRoot ( )`

Returns:
NULL, on failure otherwise a valid handle to access the root VLYNQ module.

3.1.38 PAL_sysVlynqGetRootVLYNQ( ) – Get root for the given VLYNQ.

PAL_VLYNQ_HND*	PAL_sysVlynqGetRoot	(	PAL_VLYNQ_HND*	vlynq
		)		

Parameters:

vlynq	Handle to the VLYNQ whose root is being sought.

Returns:
NULL, on failure otherwise a valid handle, to access the root VLYNQ module.

3.1.39 PAL_sysVlynqGetRootAtBase( ) – Get the root VLYNQ at the base address.

PAL_VLYNQ_HND*	PAL_sysVlynqGetRootAtBase	(	Uint32	base_addr
		)		

Parameters:

base_addr	The virtual base address of the VLYNQ module on the SOC.

Returns:
NULL, on failure otherwise a valid handle to access the root VLYNQ module.

3.1.40 PAL_sysVlynqGetBaseAddr( ) – Returns the base address of the VLYNQ.

PAL_Result	PAL_sysVlynqGetBaseAddr	(	PAL_VLYNQ_HND*	vlynq,
			Uint32	*base_addr
		)		

Parameters:

vlynq	The Vlynq module whose base address is being sought.
base_addr	The placeholder for the virtual base address of the VLYNQ module on the SOC.

Returns:
0 on success

-1 on failure.

3.1.41 PAL_sysVlynqGetNext( ) – Get the next VLYNQ module in the chain.

PAL_VLYNQ_HND*	PAL_sysVlynqGetNext	(	PAL_VLYNQ_HND*	this
		)		

Parameters:

this	The VLYNQ module instance whose next chained entity (away from the root) is being sought.

Returns:
NULL, on failure otherwise a valid handle to access the neighbor module.

3.1.42 PAL_sysVlynqIsLast( ) – Is this VLYNQ module the last one in the chain.

Int32	PAL_sysVlynqIsLast	(	PAL_VLYNQ_HND*	this
		)		

Parameters:

this	The VLYNQ module instance to be ascertained whether last in the chain (away from the root).

Returns:
0 – on false.

1 – on true.

3.1.43 PAL_sysVlynqGetChainLength( ) – Get the length of the VLYNQ chain.

Int32	PAL_sysVlynqGetChainLength	(	PAL_VLYNQ_HND*	this
		)		

Parameters:

this	The starting VLYNQ module instance inclusive of which, the number of VLYNQ modules existing in the chain (away from root) is being sought. If this happens to be the root VLYNQ module, then the length of the entire chain is ascertained.

Returns:
The number of VLYNQ modules in the chain (away from the root).

3.1.44 PAL_sysVlynqDump( ) – Dumps vital VLYNQ information into the buffer.

Int32	**PAL_sysVlynqDump**	**(**	**PAL_VLYNQ_HND***	**vlynq,**
			Uint32	**dump_type,**
			char*	**buf,**
			Int32	**limit,**
			Int32*	**eof**
		)		

Parameters:

vlynq	The vlynq instance for which the dump has to be provided.
dump_type	Identifies the information being sought. Some of the commands can be raw byte dump of the hardware, complete chain dump, raw register values or specific register value with enumeration such as status register or control register.
buf	The placeholder for the buffer. After the function, the buffer can be printed out for reading and information.
limit	The size of the buffer. It is strongly recommended to provide a buffer of at least 4096 bytes.
eof	Whether the buffer was insufficient.

Returns:
The number of bytes that have been formatted and placed in the buffer.

-1 on error.

3.2. Data Structures

This section lists the data structures available in the VLYNQ module.

```
typedef struct pal_vlynq_t
{
   Uint32                          base;
   Bool                            soc;
   Uint32                          vlynq_version;
   Uint32                          timeout_ms;
   struct pal_vlynq_t              *next;
   struct pal_vlynq_t              *prev;
   PAL_VLYNQ_DEV_HND               *local_dev[MAX_DEV_PER_VLYNQ];
   PAL_VLYNQ_DEV_HND               *peer_dev[MAX_DEV_PER_VLYNQ];
   PAL_VLYNQ_REGION_INFO_T         local_region_info[MAX_VLYNQ_REGION];
   PAL_VLYNQ_REGION_INFO_T         remote_region_info[MAX_VLYNQ_REGION];
   PAL_VLYNQ_IRQ_MAP_T             *root_irq_map;
   PAL_VLYNQ_IRQ_INFO_T            *root_irq_info;
   PAL_VLYNQ_ISR_INFO_T            *local_isr_info;
   PAL_VLYNQ_ISR_INFO_T            *peer_isr_info;
   Uint32                          backup_local_cntl_word;
   Uint32                          backup_local_intr_ptr;
   Uint32                          backup_local_tx_map;
   Uint32                          backup_local_endian;
   Uint32                          backup_peer_cntl_word;
   Uint32                          backup_peer_intr_ptr;
   Uint32                          backup_peer_tx_map;
   Uint32                          backup_peer_endian;
   Int8                            local_irq;
   Int8                            peer_irq;
   Bool                            local_swap;
   Bool                            peer_swap;

} PAL_VLYNQ_T;


typedef struct pal_vlynq_region_info_t
{
   Int8                            owner_dev_index;
   Bool                            owner_dev_locale;

} PAL_VLYNQ_REGION_INFO_T;

typedef struct pal_vlynq_dev_cb_t
{
   PAL_VLYNQ_DEV_CB_FN             cb_fn;
   void                            *cb_param;
   struct pal_vlynq_dev_cb_t       *next;

} PAL_VLYNQ_DEV_CB_T;
```

```
typedef struct pal_vlynq_dev_t
{
   Char                          name [32];
   PAL_VLYNQ_HND                 *vlynq;
   struct pal_vlynq_dev_t        *next;
   Uint8                         reset_bit;
   Uint8                         irq_count;
   Uint8                         instance;
   Bool                          peer;
   Int8                          irq [8];
   PAL_VLYNQ_DEV_CB_T             *dev_cb;

} PAL_VLYNQ_DEV_T;

typedef struct pal_vlynq_ioctl_info
{
   Int16                          id;
   Uint16                        offset; /* Register offset or bit offset. */
   const char                    *name;
   Uint32                        mask;
   Uint32                        start_revision;
   Uint32                        end_revision;
   Uint8                         access_flags; /* 0x01 - write, otherwise read,
                                 can be enhanced later. */

} PAL_VLYNQ_IOCTL_INFO_T;
```

Data Fields

PAL_VLYNQ_INIT_ERR_ENUM_T	error_status
Uint8	on_soc
char	error_msg[50]
Uint32	base_addr
Uint32	init_timeout_in_ms
Uint8	local_clock_div
PAL_VLYNQ_CLOCK_DIR_ENUM_T	local_clock_dir
Uint8	local_intr_local
Uint8	local_intr_vector
Uint8	local_intr_enable
Uint8	local_int2cfg
Uint32	local_intr_pointer
PAL_VLYNQ_ENDIAN_ENUM_T	local_endianness
Uint32	local_tx_addr
PAL_VLYNQ_RTM_CFG_ENUM_T	local_rtm_cfg_type
Uint8	local_rtm_sample_value

Bool	local_tx_fast_path
Uint8	peer_clock_div
PAL_VLYNQ_CLOCK_DIR_ENUM_T	peer_clock_dir
Uint8	peer_intr_local
Uint8	peer_intr_enable
Uint8	peer_intr_vector
Uint8	peer_int2cfg
Uint32	peer_intr_pointer
PAL_VLYNQ_ENDIAN_ENUM_T	peer_endianness
Uint32	peer_tx_addr
PAL_VLYNQ_RTM_CFG_ENUM_T	peer_rtm_cfg_type
Uint8	peer_rtm_sample_value
Bool	peer_tx_fast_path
Bool	init_swap_flag

Detailed Description:
VLYNQ driver configuration.

Field Documentation

Uint8 on_soc
the VLYNQ module is on the SOC.

Uint32 base_addr.
init_timeout_in_ms
Virtual Base Address of the module. Uint32
The number of millsecs that the software should allow for initialization to complete.

Uint8 local_clock_div
The clock divisor for the local VLYNQ module.

PAL_VLYNQ_CLOCK_DIR_ENUM_Tlocal_clock_dir
The clock direction; sink or source for the local VLYNQ module.

Uint8 local_intr_local
Interrupts are being handled locally or sent over BUS.

Uint8 local_intr_vector
The IRQ vector to be used on the local VLYNQ module. Valid values are 0 to 31.

Uint8 local_intr_enable

Uint8 local_int2cfg
Local VLYNQ interrupt pending register.

Uint32 local_intr_pointer
Address to which the irq should be written to,valid only if int2cfg is not set.

PAL_VLYNQ_ENDIAN_ENUM_T local_endianness
Endianess of the local VLYNQ module.

Uint32 local_tx_addr
The physical portal address of the local VLYNQ.

PAL_VLYNQ_RTM_CFG_ENUM_T local_rtm_cfg_type
The RTM configuration for the local VLYNQ.

Uint8 local_rtm_sample_value
The RTM sample value for the local VLYNQ.

Bool local_tx_fast_path
TX Fast path.

Uint8 peer_clock_div
The clock divisor for the peer VLYNQ module.

PAL_VLYNQ_CLOCK_DIR_ENUM_T peer_clock_dir
The clock direction - sink or source for the peer VLYNQ module.

Uint8 peer_intr_local
Interrupts are being handled by peer VLYNQ.

Uint8 peer_intr_enable
Enable the peer irq vector.

Uint8 peer_intr_vector
The IRQ vector to be used on the peer VLYNQ module.

Uint8 peer_int2cfg
The local VLYNQ interrupt pending register.

Uint32 peer_intr_pointer
Address to which the irq should be written.

PAL_VLYNQ_ENDIAN_ENUM_T peer_endianness
Endianess of the local VLYNQ module.

Uint32 peer_tx_addr
The physical portal address of the peer VLYNQ.

PAL_VLYNQ_RTM_CFG_ENUM_T peer_rtm_cfg_type
The RTM configuration for the peer VLYNQ.

Uint8 peer_rtm_sample_value
The RTM sample value for the local VLYNQ.

Bool peer_tx_fast_path
TX Fast path.

Bool init_swap_flag
Is the Vlynq module in Endian swapped state to begin with

3.3. Enumerations

This section lists the enumerations available in the PSP module.

enum PAL_VLYNQ_CLOCK_DIR_ENUM_T
Enumeration values:
pal_vlynq_clk_in Sink Direction
pal_vlynq_clk_out Source Direction

enum PAL_VLYNQ_ENDIAN_ENUM_T
Enumeration values:
pal_vlynq_ignore_en Ignore the endian.
pal_vlynq_little_en Little Endian
pal_vlynq_big_en Big Endian

enum PAL_VLYNQ_RTM_CFG_ENUM_T
Enumeration values:
no_rtm_cfg No Configuration
rtm_auto_select_sample_val Auto Configuration
rtm_force_sample_val Force Configuration

enum PAL_VLYNQ_INIT_ERR_ENUM_T
Enumeration values:
pal_vlynq_init_success Initialization Success.
pal_vlynq_init_no_link_on_reset Initialization link on Reset.
pal_vlynq_init_no_mem Initialization No meaning.
pal_vlynq_init_clk_cfg Initialization clock configuration
pal_vlynq_init_no_link_on_clk_cfg Initialization no link on config. clock
pal_vlynq_init_internal_problem Initialization internal Error
pal_vlynq_init_invalid_param Initialization invalid param
pal_vlynq_init_local_high_rev Initialization Local high rev.
pal_vlynq_init_peer_high_rev Initialization Peer high rev.
pal_vlynq_init_rtm_cfg Initialization of RTM configuration
pal_vlynq_init_no_link_on_rtm_cfg Initialization no link on RTM config.

enum PAL_VLYNQ_IRQ_POL_ENUM_T
Enumeration values:
pal_vlynq_high_irq_pol High IRQ Pol.
pal_vlynq_low_irq_pol Low IRQ Pol.

enum PAL_VLYNQ_IRQ_TYPE_ENUM_T
Enumeration values:
pal_vlynq_level_irq_type VLYNQ IRQ Level
pal_vlynq_edge_irq_type VLYNQ Edge Type.

3.4. Macros

This section lists the macros available in the PSP module.

 # #define MAX_VLYNQ_REGION 4
Maximum vlynq memory region

#define MAX_DEV_PER_VLYNQ MAX_VLYNQ_REGION
Maximum device per vlynq interface.

#define MAX_DEV_COUNT (2u)
Maximum device count per vlynq interface in DM648 SOC.

#define MAX_IRQ_PER_CHAIN 32
Device IRQ per chain.

#define MAX_IRQ_PER_VLYNQ 8
Number IRQ per VLYNQ .

#define TYPICAL_NUM_ISR_PER_IRQ 1
Number of ISR per IRQ.

```
#define PAL_VLYNQ_IOCTL_BIT_CMD                 (1 << 31)
#define PAL_VLYNQ_IOCTL_READ_CMD                (1 << 30)
#define PAL_VLYNQ_IOCTL_REMOTE_CMD              (1 << 29)


#define PAL_VLYNQ_IOCTL_MAJOR_VAL(val)          ((val & 0xff) << 16)
#define PAL_VLYNQ_IOCTL_MAJOR_DE_VAL(cmd)       ((cmd >> 16) & 0xff)

#define PAL_VLYNQ_IOCTL_MINOR_VAL(val)          (val & 0xff)
#define PAL_VLYNQ_IOCTL_MINOR_DE_VAL(cmd)       (cmd & 0xff)

/* Major commads; if bit option is not selected. */
#define PAL_VLYNQ_IOCTL_REG_CMD                 (0x00) /* Shall use vlynq regs
                                                          as minor cmd*/
#define PAL_VLYNQ_IOCTL_PREP_LINK_DOWN          (0x01) /* Prepare to teardown
                                                          the link. */
#define PAL_VLYNQ_IOCTL_PREP_LINK_UP            (0x02) /* Setup   now the link is up. */
#define PAL_VLYNQ_IOCTL_CLEAR_INTERN_ERR        (0x03) /*Clear internal interrupt errors.*/

/* Control Register parameters, valid for bit operations. */

#define PAL_VLYNQ_IOCTL_CNT_RESET_CMD           (0x00)
#define PAL_VLYNQ_IOCTL_CNT_ILOOP_CMD           (0x01)
#define PAL_VLYNQ_IOCTL_CNT_AOPT_CMD            (0x02)  /* Write */
#define PAL_VLYNQ_IOCTL_CNT_INT2CFG_CMD         (0x07)
#define PAL_VLYNQ_IOCTL_CNT_INTVEC_CMD          (0x08)
#define PAL_VLYNQ_IOCTL_CNT_INT_EN_CMD          (0x0d)
#define PAL_VLYNQ_IOCTL_CNT_INT_LOC_CMD         (0x0e)
#define PAL_VLYNQ_IOCTL_CNT_CLK_DIR_CMD         (0x0f)
```

```
#define PAL_VLYNQ_IOCTL_CNT_CLK_DIV_CMD       (0x10)  /* Write */
#define PAL_VLYNQ_IOCTL_CNT_CLK_MOD_CMD     (0x15)
#define PAL_VLYNQ_IOCTL_CNT_TX_FAST_CMD       (0x15u)
#define PAL_VLYNQ_IOCTL_CNT_RTM_SELECT_CMD (0x16u)
#define PAL_VLYNQ_IOCTL_CNT_RTM_VALIDWR_CMD (0x17u)
#define PAL_VLYNQ_IOCTL_CNT_RTM_SAMPLE_CMD (0x18u)
#define PAL_VLYNQ_IOCTL_CNT_CLK_SLKPU_CMD   (0x1e)  /* Write */
#define PAL_VLYNQ_IOCTL_CNT_PMEM_CMD          (0x1f)  /* Write */
```

/* Status Register parameters, valid for bit operations. */

```
#define PAL_VLYNQ_IOCTL_STS_LINK                (0x00)
#define PAL_VLYNQ_IOCTL_STS_MPEND               (0x01)
#define PAL_VLYNQ_IOCTL_STS_SPEND               (0x02)
#define PAL_VLYNQ_IOCTL_STS_NFEMP0              (0x03)
#define PAL_VLYNQ_IOCTL_STS_NFEMP1              (0x04)
#define PAL_VLYNQ_IOCTL_STS_NFEMP2              (0x05)
#define PAL_VLYNQ_IOCTL_STS_NFEMP3              (0x06)
#define PAL_VLYNQ_IOCTL_STS_LERR                (0x07)
#define PAL_VLYNQ_IOCTL_STS_RERR                (0x08)
#define PAL_VLYNQ_IOCTL_STS_OFLOW               (0x09)
#define PAL_VLYNQ_IOCTL_STS_IFLOW               (0x0A)
#define PAL_VLYNQ_IOCTL_STS_RTM                 (0x0Bu)
#define PAL_VLYNQ_IOCTL_STS_RTM_VAL             (0x0Cu)
#define PAL_VLYNQ_IOCTL_STS_SWIDOUT             (0x14)
#define PAL_VLYNQ_IOCTL_STS_MODESUP             (0x15)
#define PAL_VLYNQ_IOCTL_STS_SWIDIN              (0x18)
#define PAL_VLYNQ_IOCTL_STS_SWIDTH               (0x18)
#define PAL_VLYNQ_IOCTL_STS_DEBUG               (0x1d)
```

/* VLYNQ Registers */

```
#define PAL_VLYNQ_IOCTL_REVSION_REG             (0x00)
#define PAL_VLYNQ_IOCTL_CONTROL_REG             (0x04)
#define PAL_VLYNQ_IOCTL_STATUS_REG              (0x08)
#define PAL_VLYNQ_IOCTL_INT_PRIR_REG            (0x0c)
#define PAL_VLYNQ_IOCTL_INT_STS_REG             (0x10)
#define PAL_VLYNQ_IOCTL_INT_PEND_REG            (0x14)
#define PAL_VLYNQ_IOCTL_INT_PTR_REG             (0x18)
#define PAL_VLYNQ_IOCTL_TX_MAP_REG              (0x1c)
#define PAL_VLYNQ_IOCTL_RX1_SZ_REG              (0x20)
#define PAL_VLYNQ_IOCTL_RX1_OFF_REG             (0x24)
#define PAL_VLYNQ_IOCTL_RX2_SZ_REG              (0x28)
#define PAL_VLYNQ_IOCTL_RX2_OFF_REG             (0x2c)
#define PAL_VLYNQ_IOCTL_RX3_SZ_REG              (0x30)
#define PAL_VLYNQ_IOCTL_RX3_OFF_REG             (0x34)
#define PAL_VLYNQ_IOCTL_RX4_SZ_REG              (0x38)
#define PAL_VLYNQ_IOCTL_RX4_OFF_REG             (0x3c)
#define PAL_VLYNQ_IOCTL_CVR_REG                 (0x40)
#define PAL_VLYNQ_IOCTL_AUTO_NEG_REG            (0x44)
#define PAL_VLYNQ_IOCTL_MAN_NEG_REG             (0x48)
#define PAL_VLYNQ_IOCTL_NEG_STS_REG             (0x4c)
#define PAL_VLYNQ_IOCTL_ENDIAN_REG              (0x4c)
#define PAL_VLYNQ_IOCTL_IVR30_REG               (0x60)
```

```
#define PAL_VLYNQ_IOCTL_IVR74_REG          (0x64)

/* Dumping options, not part of ioctl options. */

#define PAL_VLYNQ_DUMP_ALL_ROOT            (0x10000)
#define PAL_VLYNQ_DUMP_RAW_DATA            (0x20000)
#define PAL_VLYNQ_DUMP_ALL_REGS            (0x30000)
#define PAL_VLYNQ_DUMP_STS_REG             (0x00008)
#define PAL_VLYNQ_DUMP_CNTL_REG            (0x00004)
```

3.5 Dependency of Sample application:

Following Components needs to be linked for successful build and functionality of the application.

- VLYNQ
- PAL_OS
- SoC specific PAL_SYS
- EDMA3