VLYNQ v1.4

DS324 April 24, 2009 Product Specification

Introduction

The Xilinx® VLYNQ™ interface core is a fully verified, serial, low-pin-count communications interface for cost-sensitive applications. Developed by Texas Instruments, the VLYNQ interface enables the extension of an internal bus segment to one or more external devices in a scalable fashion without extensive software modification, and was originally designed as a mechanism to preserve software investment.

Features

- Master and/or slave with a peer-to-peer connection
- Memory-mapped accesses
- Low external pin count (3 to 10 pins)
- Full-duplex operation: 1 to 4 serial data bits in both transmit and receive direction
- LVCMOS serial I/O signaling at up to 100 MHz
- OPB bus signaling on user-side
- Automatic serial data I/O bit-width detection
- In-band flow control
- Power management

LogiCORE™ IP Facts	
Core Specifics	
Supported Device Families	Spartan®-3/XA, Spartan-3A/AN/DSP, Spartan-3, Spartan-3E/XA, Virtex®-5, Virtex-4
Performance	• 90 MHz: Spartan-3E/XA • 100 MHz: Spartan-3/XA, Spartan-3A/AN/DSP • 125 MHz: Virtex-5, Virtex-4
Core Resources	
Slices	2923[1]
LUTs	5122
FFs	2612
Block RAMs	8
BUFG	1 to 4
IOBs	3 to 10
Provided with Core	
Documentation	Product Specification Release Notes
Design File Formats	VHDL, Verilog
Constraints File	User Constraints File (UCF)
Design Tool Requirements	
Supported HDL	VHDL, Verilog
Synthesis	XST, Synplify
Xilinx Tools	ISE® software v11.1
Simulation tools	Mentor ModelSim 6.4b and above Cadence IUS v8.1 -s009 and above
Core Highlights	
Compliant to rev. 1.2	Demo Test Environment
Hardware Verified	Hardware Evaluation
Compatible with Texas Instruments VLYNQ v2.6	

1. The precise number of slices depends on the user configuration.

Overview

The Xilinx VLYNQ interface can be used to communicate with additional VLYNQ-enabled devices in a system. In a VLYNQ system, there is a single *root device* (typically a communication processor) that executes the enumerator software, which configures each VLYNQ device in the system to create a common address map.

The root VLYNQ interface is connected to a VLYNQ module, called a *gateway*, on another device, which from the perspective of the root is defined as the first VLYNQ module encountered. A device may have additional VLYNQ modules, called *portals*, implemented. A VLYNQ-enabled device may be attached to a portal, and subsequent devices may be attached beyond that device in a daisy chain. Any device in a VLYNQ daisy chain may be a Xilinx FPGA, including the following: the root device, an intermediate daisy chain device, or an endpoint to the daisy chain.

Applications

A typical application for the VLYNQ core is to enable a processor device with a VLYNQ interface to communicate with a customized peripheral function implemented on a Xilinx FPGA. In this context, the Xilinx FPGA implements a VLYNQ core along with the peripheral function. Any task the processor wants the peripheral function to do is executed by sending a VLYNQ transaction to the Xilinx device. If appropriate for the application, the peripheral function can return data to the processor. Figure 1 illustrates a sample system.

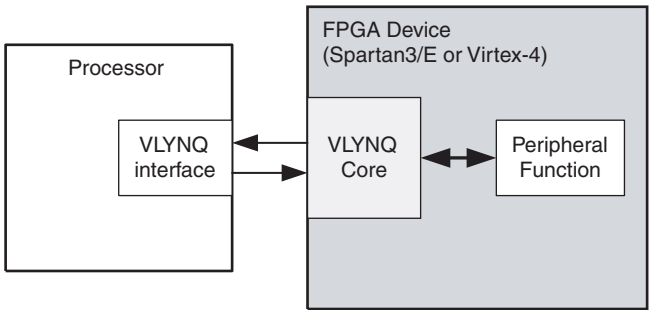

Figure 1: **Sample System-Level Block Diagram**

Feature Summary

- **Master and/or Slave with a Peer-to-peer Connection.** The VLYNQ core allows the user logic to master read and write transactions, and also operates as a slave when a remote VLYNQ device masters a read or write transaction. All transactions happen over a serial interface that connects to only one other VLYNQ device.
- **Memory-mapped Accesses**. A VLYNQ system has a root device that enumerates all VLYNQ devices and maps them into a common address space.
- **Low External Pin Count**. An external VLYNQ serial interface may have as few as 3 pins (1 Tx, 1 Rx, and 1 clock) and as many as 10 pins (4 Tx, 4 Rx, 1 clock, and 1 power management) for maximum bandwidth.
- **Full-duplex Operation**. Separate Tx and Rx serial buses allow data to be simultaneously transmitted and received. There can be 1 to 4 Tx and Rx pins each, and the number of Tx and Rx pins do not have to be equal.

- **LVCMOS Serial I/O Signaling at Up to 100 MHz**. Signaling does not require the use of MGTs; normal LVCMOS I/O is all that is required. A serial clock speed of 100 MHz is possible for some devices, which can yield a maximum bandwidth of 285 Mbps per direction using 4 pins (570 Mbps total throughput).
- **OPB Bus Signaling on User-side**. The standard OPB bus interface is used on the user interface, which permits reuse of developed IP for controlling the core.
- **Automatic Serial Data I/O Bit-width Detection**. The VLYNQ core is capable of determining how many Tx and Rx pins are actually connected, eliminating the need to configure the part with this information beforehand. It is possible to change the number of connected bit lanes in hardware without making any modifications to the FPGA or software.
- **In-band Flow Control**. If buffer space is limited and the user interface is not capable of accepting data for some time, the VLYNQ core automatically signals the remote core to stop transmitting by sending a command in-band. When buffer space is available, the VLYNQ core re-enables transmission.
- **Power Management**. A VLYNQ device may optionally implement a SCRUN_N pin, which can be used to indicate that the interface can be stopped for power savings.

Functional Description

The VLYNQ core is comprised of logic dedicated to transmitting transactions onto the serial Tx lines and logic dedicated to receiving transactions from the serial Rx lines. The transmitted transactions can be master read commands or write commands with data, or they may be slave responses to remote read requests. The received transactions can be remote read commands or write commands with data, or they may be remote responses to master read commands.

The VLYNQ core has two interfaces:

- An OPB bus interface for the user side
- The VLYNQ serial interface

When you master transactions on the OPB bus to the VLYNQ OPB slave interface, they are recorded into an Outbound Command FIFO, and when they are ready to be transmitted, they are 8b/10b encoded and serialized in preparation for transmission over the serial TX interface. If the command is a read, read data is expected to be returned on the serial Rx interface, which is then deserialized, 8b/10b decoded, stored into the OPB Slave Read Return FIFO, and returned to you on the VLYNQ OPB Slave Interface. When the VLYNQ core receives a command from the remote VLYNQ device, that command is deserialized, 8b/10b decoded, stored into the Inbound Command FIFO, and retrieved for presentation on the VLYNQ OPB Master Interface. If that command is a read, you are expected to provide the read data to the OPB Master, which is stored in the OPB Master Read Return FIFO for transmission. Figure 2 illustrates the sub-functions of the VLYNQ core.

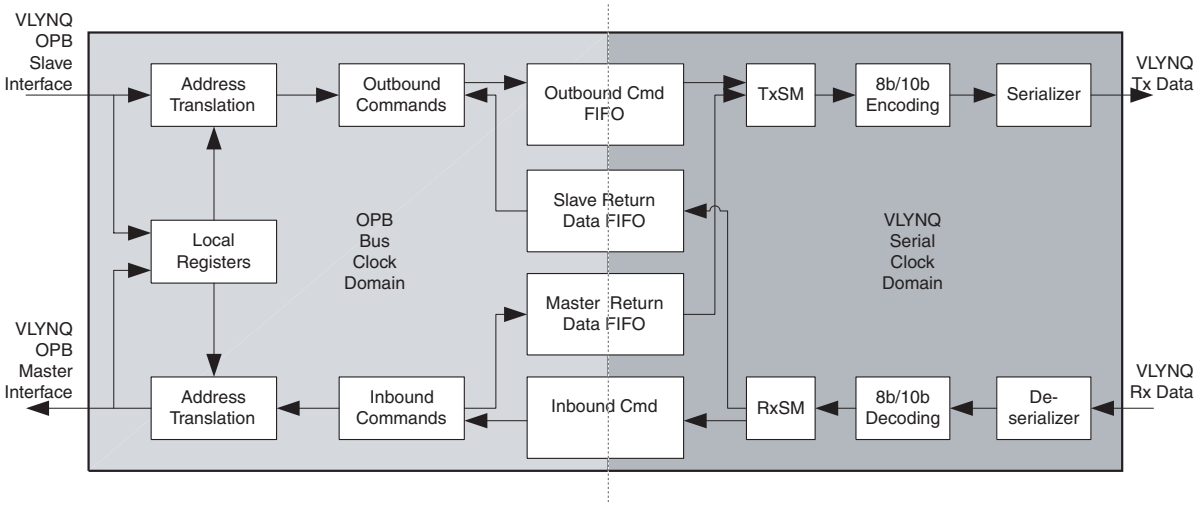

Figure 2: **VLYNQ Core Block Diagram**

Address Translation

The VLYNQ core allows each receive packet address to be translated into one of four mapped regions. A packet address sent on the Local Device within a given Local Tx Map Region is translated in the Remote Device into a packet address at the same relative location in the corresponding Remote Rx Map Region.

No restriction is placed on the size or offset of each mapped region, except that each must be aligned to 32-bit words. Address mapping is accomplished using four register pairs in both the Tx and Rx (Tx/Rx Address Map Size i and Tx/Rx Address Map Offset i) that define the size and offset of each mapped region. Figure 3 illustrates an example of one possible map.

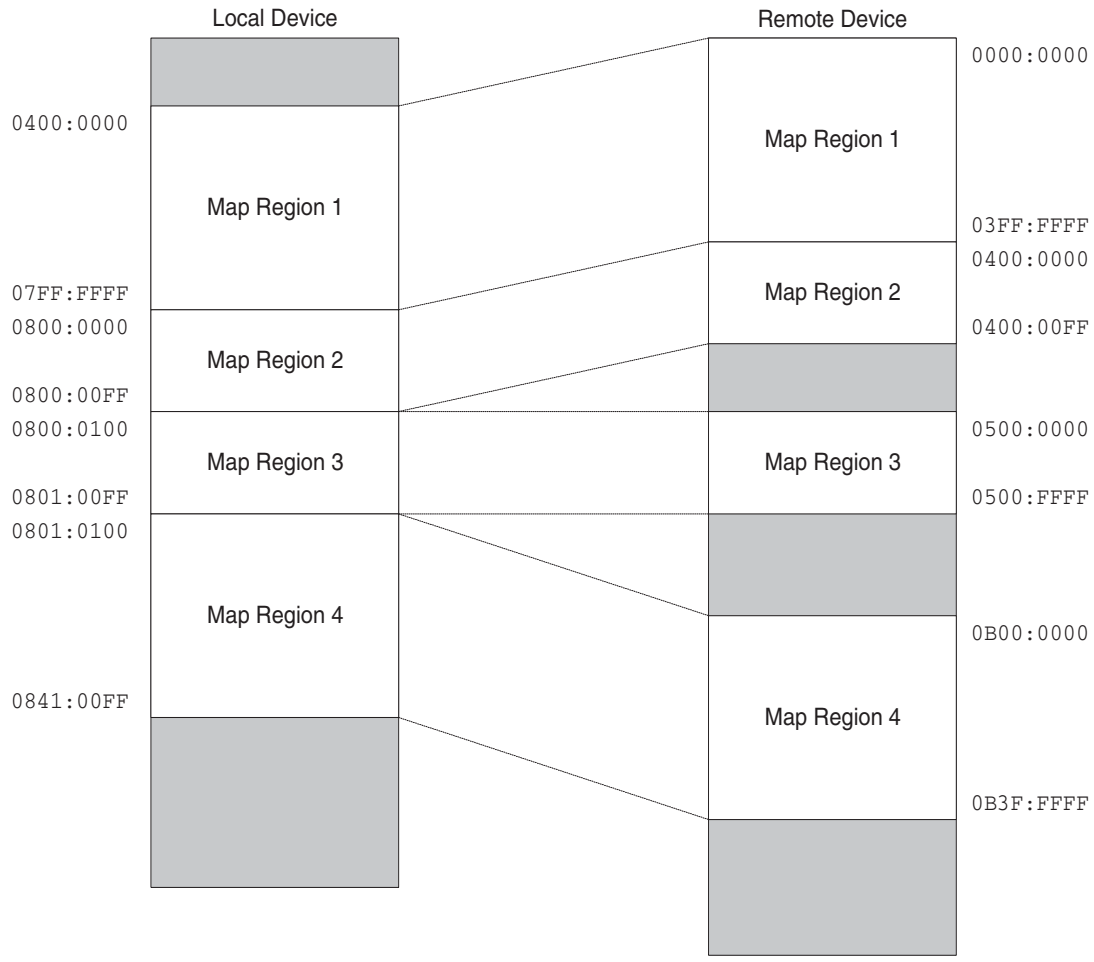

Figure 3: **Example Address Memory Map**

Address Memory Map Example

Assume that the local Tx Address Map Register is 0x04000000 and the remote Rx Address Map Size 1 and Offset 1 are 0x00000100 and 0x00000000, respectively. If an address of 0x04000054 is transmitted on the local OPB interface, the address is translated in the following way:

Packet address transmitted = 0x04000054 - 0x04000000 = 0x00000054

The remote VLYNQ module sees this address, 0x00000054, and the data is presented to its internal logic as:

Comparison (0x00000054 < 0x00000100) is true, so
Remote translated address = 0x00000054 + 0x00000000 = 0x00000054

The precise algorithm Rx uses for address translation is as follows:

```
If (Rx Packet Address < Rx Address Map Size 1 Register) {
     Translated Address = Rx Packet Address +
Rx Address Map Offset 1 Register
} else if (Rx Packet Address < (Rx Address Map Size 1 Register +
                                Rx Address Map Size 2 Register)) {
     Translated Address = Rx Packet Address +
                          Rx Address Map Offset 2 Register -
                          Rx Address Map Size 1 Register
} else if (Rx Packet Address < (Rx Address Map Size 1 Register +
                                Rx Address Map Size 2 Register +
                                Rx Address Map Size 3 Register)) {
     Translated Address = Rx Packet Address +
                          Rx Address Map Offset 3 Register -
                          Rx Address Map Size 1 Register -
                          Rx Address Map Size 2 Register
} else if (Rx Packet Address < (Rx Address Map Size 1 Register +
                                Rx Address Map Size 2 Register +
                                Rx Address Map Size 2 Register +
                                Rx Address Map Size 4 Register)) {
     Translated Address = Rx Packet Address +
                          Rx Address Map Offset 4 Register -
                          Rx Address Map Size 1 Register -
                          Rx Address Map Size 2 Register -
                          Rx Address Map Size 3 Register
} else {
     Translated Address = 0x0
}
```

Multiple VLYNQ modules may be included on any device such that VLYNQ serial interfaces are effectively daisy chained. The only requirement is that the address translation registers are configured to include the outbound VLYNQ module of the remote device.

In the OPB Slave interface of VLYNQ, there are two requests to distinguish between access to the internal register space of the VLYNQ core and access to the address map space of the remote VLYNQ. Two additional OPB signals are used to indicate the destination of the transaction: SL_REG and SL_MAP. When SL_REG is set to '1,' the current OPB transaction accesses the internal register space of the local (0x00-0x7F offset) and the remote (0x80-0xFF offset) VLYNQ interface, which is a total of 256 contiguous bytes. To request address map space, the signal SL_MAP should be driven to '1.' If this occurs and a transaction is started on the OPB Master, the core creates a transaction across the serial interface. The chip architect needs to add the address decode for this request signal to meet his or her chip address map requirements. Care should be taken to ensure that the SL_REG and SL_MAP signals are never simultaneously driven to '1.'

Clocking

The VLYNQ core provides several clock inputs and outputs to ensure maximum flexibility in implementation. The core logic is potentially clocked by up to 4 different clocks: OPB, VLYNQ reference clock, VLYNQ Tx, and VLYNQ Rx. In practice, all clocks are not necessarily distinct, and several of the clock inputs can be tied together. Your system requirements determine the number of clocks required. The core does not instantiate any BUFGs internally. The number of BUFGs in the system is determined by the clocking scheme that the customer uses.

OPB Clock

The OPB clock should be clocked at the rate your interface operates. This clock is normally different than the other VLYNQ clocks, and can potentially be much slower than the VLYNQ clock domains without affecting overall bandwidth. For example, if there are only 1 Tx and 1 Rx pin in operation, the 32-bit OPB bus can potentially be clocked as slow as 1/32 of the VLYNQ clocks. It can be clocked even slower, but the consequence is that idle cycles may need to be inserted into the VLYNQ data stream in the Tx direction, and the core may have to use flow control for the Rx direction. In practice, the OPB bus is likely to be clocked at the rate that other IP connected to the OPB bus is running.

VLYNQ Reference Clock and Clock Outputs

A very tiny portion of the VLYNQ core is directly clocked by the core reference clock, `VLYNQ_REF_CLK`. The VLYNQ core is capable of producing a clock output that is either the VLYNQ reference clock precisely (`VLYNQ_CLK_FULL`), or a divided-down version (`VLYNQ_CLK_DIV`). This resulting clock output can drive the external `VLYNQ_CLK` output if the core has been programmed to do so; if it does, both the remote device and the VLYNQ core use that clock for transmitting and receiving data, and if it does not, both the remote device and the VLYNQ core use the remote clock of the device for transmitting and receiving data.

If the system designer knows that the Xilinx device will never produce the VLYNQ serial clock, this signal can be tied off. It is possible to use the OPB clock to drive this input.

VLYNQ Tx Clock

The transmit path is clocked by the VLYNQ Tx clock of the core, `VLYNQ_TCLK`. If the system is a fully compliant VLYNQ, this clock can be driven from either a local source or the remote VLYNQ device, and the driver can be dynamically changed during operation. If this is the case, or if the Xilinx device is always driven by the remote clock, this clock will likely need be driven from the `VLYNQ_CLK` pin (via a BUFG) of the chip.

Note: It may be impractical to use a Xilinx DLL to compensate for the clock delay because the duty cycle of the generated VLYNQ module clocks may not be 50/50 and may not be within the tolerances allowed by Xilinx DLLs.

VLYNQ Rx Clock

The receive path is clocked by the VLYNQ Rx clock input of the clock, `VLYNQ_RCLK.` If the system is a fully compliant VLYNQ interface, this clock could be driven from either a local source or the remote VLYNQ device, and the driver can be dynamically changed during operation. If that is the case, or if the Xilinx device is always driven by the remote clock, this clock will likely need be driven from the `VLYNQ_CLK` pin (via a BUFG) of the chip.

Note: It may be impractical to use a Xilinx DLL to compensate for the clock delay, because the duty cycle of the generated VLYNQ module clocks may not be 50/50 and may not be within the tolerances allowed by Xilinx DLLs.

Figure 4 illustrates the implementation of a fully compliant VLYNQ clocking scheme. If the local VLYNQ core is to drive the clock (determined by programming the register space), the signal VLYNQ_CLK_OE_N is driven to '0,' and causes the IOBs output buffer to drive out on the VLYNQ_CLK pad.

If the register space has not been programmed with a divisor value, the full rate VLYNQ_CLK_FULL is the driven clock and VLYNQ_CLK_SELECT is '1;' if a divisor has been programmed, VLYNQ_CLK_DIV is the driven clock and VLYNQ_CLK_SELECT is '0.' In this design, it is assumed that you do not want to use a LUT MUX to select between the two, due to indeterminate timing. Instead, a DDR output register clocked by VLYNQ_CLK_FULL is used to clock out because it has more controllable timing. The way this scheme operates is that if the VLYNQ_CLK_SELECT is '1,' VLYNQ_CLK_FULL clocks out an alternating '1' and '0' on the DDR register, which effectively reproduces VLYNQ_CLK_FULL. Otherwise, if VLYNQ_CLK_SELECT is '0,' VLYNQ_CLK_FULL samples and clocks out VLYNQ_CLK_DIV.

If the register space has been programmed to expect a clock from the remote agent, the signal VLYNQ_CLK_OE_N is driven to '1,' which causes the IOBs output buffer to tri-state on the VLYNQ_CLK pad (a pullup is used to prevent the clock from floating).

Regardless of whether the clock pad VLYNQ_CLK is driven locally or remotely, the signal appearing on this pad is sent into the chip via a BUFG to clock the internal VLYNQ logic. To more closely match the clock DDR register, a Tx DDR register is also added to this design. The Tx data appears on the pad VLYNQ_TXD with some skew to VLYNQ_CLK; the value of this skew is approximately the sum of the input pad delay (Tiopi), clock buffer and net delay, and clock-to-out of the DDR registers.

Note: The blue highlighting is the VLYNQ clocking.

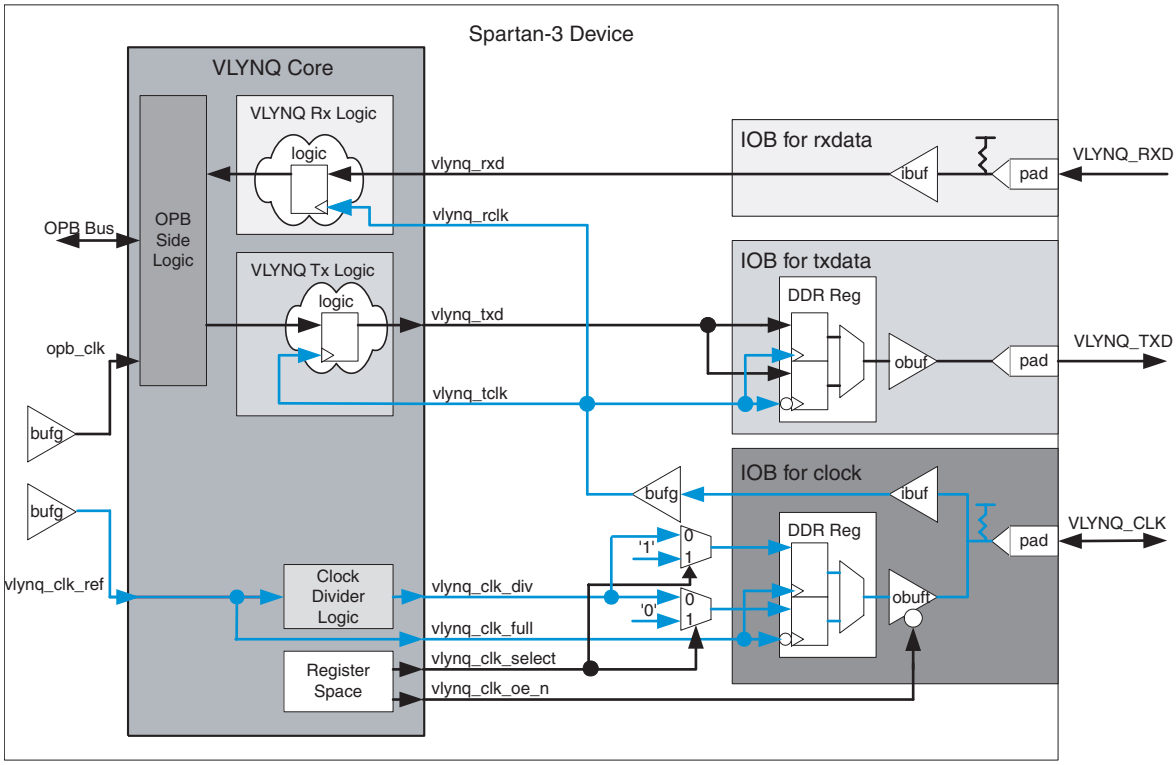

Figure 4: **VLYNQ Clocking Structure**

Note: The BUFG usage scheme detailed in the Clocking section and also in Figure 4 is the configuration used in the VLYNQ example design. This configuration is only indicatory and can be modified by users based on their applications. VLYNQ core does not instantiate any BUFGs internally.

Interrupts

The VLYNQ core is capable of generating, detecting, and forwarding interrupts. An interrupt is generated whenever the Interrupt Pending/Set Register is written, which results in either a local assertion of the interrupt signal or the sending of an interrupt packet.

To generate the interrupt, either write to a bit in the Interrupt Pending/Set Register via the OPB interface or assert one of the `VLYNQ_INT` inputs. An assertion of one of the 8 `VLYNQ_INT[i]` inputs causes the Interrupt Pending/Set Register[*Interrupt_Vector i*] to be set, assuming that the corresponding Interrupt Enable bit *i* is set in the Interrupt Vector Register, and the *Interrupt_Polarity* and *Type* bits *i* correctly describe whether the input is active high or low and the input is pulsed or level. For example, assume the Interrupt Vector 3-0 Register bits are set in the following way:

- *Interrupt_Enable 2* (bit 23) to '1'
- *Interrupt_Type 2* (bit 22) to '1'
- *Interrupt_Polarity 2* (bit 21) to '0'
- *Interrupt_Vector 2* (bits 20:16) to 0x05

In this instance, an active-high pulse on VLYNQ_INT[2] causes Interrupt Pending/Set Register[5] to be set. There is a 9th interrupt input generated from the core to flag local or remote errors, which is enabled and mapped in the Control Register.

If the VLYNQ_INT[*i*]input is configured as a pulsed input, each pulse of the signal (one OPB clock) causes the Interrupt Pending/Set Register[*Interrupt_Vector i*] bit to be set. If it is configured as a level input, the register bit is only set on an edge (rising edge if it is active-high, falling edge for active-low), unless an End-of-Interrupt (EOI) event occurs, which consists of any write to the Interrupt Pending/Set Register. If this happens, any level interrupt active on VLYNQ_INT[*i*] causes the register bit to be set, even if there is no edge.

When a bit in the Interrupt Pending/Set Register is set to '1,' either of the following can occur:

- If the *Interrupt_Local* bit in the Control Register is '0,' the content of the Interrupt Pending/Set Register is sent as a data field in an interrupt packet to the remote device, and when transmission is done the Interrupt Pending/Set Register is cleared.
- If the *Interrupt_Local* bit in the Control Register is '1,' the content of the Interrupt Pending/Set Register is ORd with and transferred to the Interrupt Status/Clear Register, and the Interrupt Pending/Set Register is cleared. No interrupt packets are sent, but VLYNQ_INTLVL asserts.

The signal VLYNQ_INTLVL is the OR of all the bits of the Interrupt Status/Clear Register and remains asserted until Interrupt Status/Clear is cleared by the software. If software clears an interrupt in Interrupt Status/Clear while an interrupt is pending, VLYNQ_INTPLS pulses.

When an interrupt packet is received, the interrupt status is extracted from the packet and written to the register indicated by the Interrupt Pointer Register. This register may point to any location in memory-mapped address space, including the VLYNQ core's own Interrupt Pending/Set Register (address location 0x14). If this option is chosen, you should set the Control Register field *Interrupt_to_Configuration_Register*. If this register is written to by the reception of an interrupt packet, interrupt generation occurs in the same way as described previously.

For additional flexibility in interrupt handling, the highest priority interrupt vector set in the Interrupt Pending/Set Register is reported in the Interrupt Priority Vector Status/Clear Register if the Interrupt_Local bit is set in the Control Register. The assumption is that bit 0 of the Interrupt Pending/Set Register is the highest priority interrupt vector, and bit 31 is the lowest. Software can clear this, or any other interrupt, by writing the vector value to the register.

Power Management

The VLYNQ core supports various power-saving features to permit it to be used in low-power applications. The features include support for a SCRUN_N pin on the device, which indicates if clocks are needed, and the ability to turn clocks off via register space writes. There are also signals from the core that permit disabling of pullups on Rx and the serial clock, but using this feature would require pullups be implemented external to the Xilinx device, because Xilinx devices cannot have their pullups dynamically enabled or disabled.

For the serial side of the logic, the VLYNQ core is capable of stopping the clock VLYNQ_CLK_DIV in a high value if it is to be idled. If it is necessary to stop VLYNQ_CLK_REF, SERIAL_BUSY or SERIAL_BUSY_REQ can be monitored by any clock control logic you implement to see if the clock can be halted. The clock output VLYNQ_CLK_FULL is simply a passthrough of VLYNQ_CLK_REF.

The Xilinx core does not support stopping of the OPB_CLK for power management.

The VLYNQ interface can implement serial clock management through the use of a SCRUN_N device pin. The pin is bi-directional with a weak pullup (which may be implemented inside the Xilinx part), and connects to the remote device SCRUN_N pin. SCRUN_N is driven low if either the local VLYNQ core or the remote VLYNQ device is in the process of transmitting or receiving a serial packet (or some are pending in the FIFOs), which is indicated by the VLYNQ core as SERIAL_BUSY being high. If both sides have no activity, both tri-state the SCRUN_N pins and the weak pullups cause it to go to '1.' The SCRUN_N I/O pin must be implemented external to the core, using the CRUN_IN_N, CRUN_OUT_N, and CRUN_OE_N ports of the core, as illustrated in Figure 5. The CRUN_OUT_N is simply tied to '0,' and CRUN_OE_N is a registered version of the inverse of SERIAL_BUSY.

If stopping the clock I/O VLYNQ_CLK is a possibility, either SCRUN_N should be used or a separate external pin should be implemented that communicates with the remote device on this matter. If SCRUN_N is used and if it is possible that the local device is going to drive the clock, SERIAL_BUSY_REQ can be monitored to see if it goes low, in which case the VLYNQ_CLK_REF can be stopped with a high value. If the remote device needs the clock, SCRUN_N is driven low, which causes the local core SERIAL_BUSY_REQ output to go high. If the local clock controller notices that SERIAL_BUSY_REQ has gone high, it should restart VLYNQ_CLK_REF.

For this implementation, the *Power_Management_Enabled* bit in the Control Register should be set. If the system is not going to stop the serial clock VLYNQ_CLK, it is not necessary to implement the SCRUN_N I/O pin.

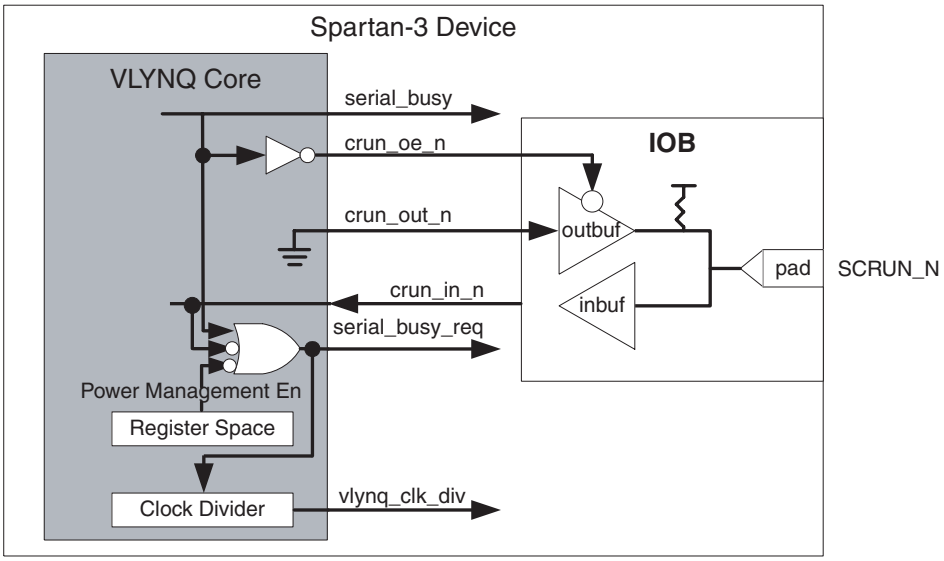

Figure 5: **VLYNQ SCRUN Implementation**

If the system is not implementing SCRUN_N but instead using a different input to indicate the clock should be stopped, and if it is possible that the local device is going to drive the clock, the clock control should monitor SERIAL_BUSY. If it is low, VLYNQ_CLK_REF may be stopped, and if SERIAL_BUSY is high or if the extra input indicates the remote device needs the clock, the clock control should restart the clock.

Interface Behavior

Figure 6 illustrates the VLYNQ core interfaces. All signals are defined in their respective interface in the sections following the illustration.

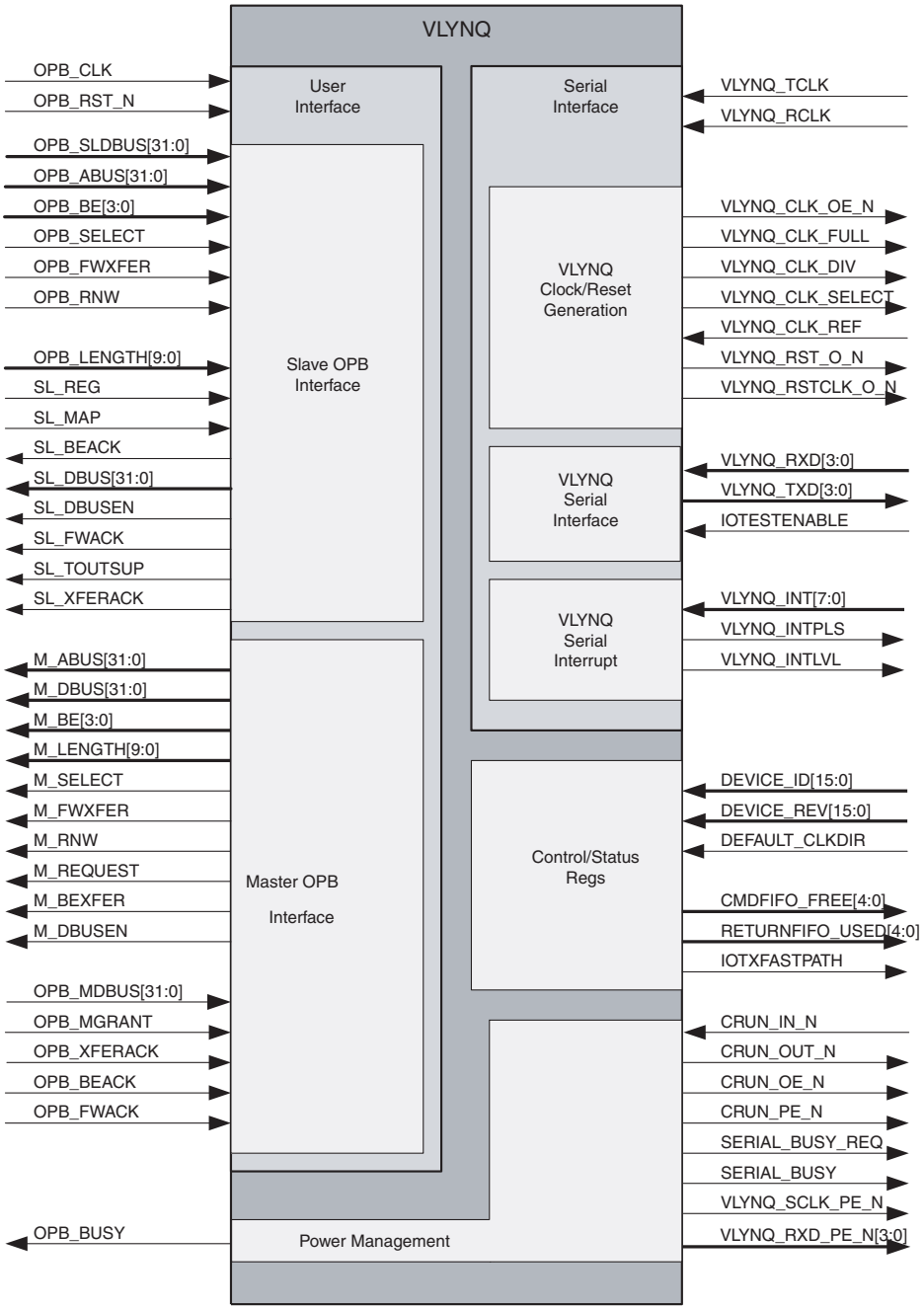

Figure 6: **VLYNQ Core Interfaces**

VLYNQ

Each of the VLYNQ serial pins, VLYNQ_TXD and VLYNQ_RXD, can be from 1 to 4 bits wide, and the width of the Tx and the Rx can differ. If the number of signals on the core interface exceeds the number of lanes to be implemented, tie off any unused upper bits of VLYNQ_RXD to ground, and leave unused upper bits of VLYNQ_TXD unconnected. When a VLYNQ system is started, the Xilinx VLYNQ core automatically interacts with the remote VLYNQ device to negotiate the width of the Tx and Rx interface. After this is determined, and the initial link handshake is complete, data packets can be sent.

There are two versions of the VLYNQ serial protocol: the legacy VLYNQ V1.X, and the newer VLYNQ V2.X. The Xilinx VLYNQ core supports both, and attempts to use V2.X if possible. However, if it is connected to a legacy product that supports only V1.X, the auto-negotiation procedure may take longer.

The VLYNQ interface is not directly coupled to the OPB interface; there are asynchronous FIFOs between the two interface domains, and the interfaces operate independently. However, if the OPB fails to generate sufficient commands and data to consume all the VLYNQ interface's bandwidth, the VLYNQ interface generates idle packets. If the OPB fails to immediately accept all remotely generated commands and data, the FIFOs fill and the VLYNQ interface turns flow control on.

OPB

The VLYNQ module implements the On-chip Peripheral Bus (OPB) standard. The OPB interface can support multiple master and slave devices, although a typical use with the VLYNQ core is a simple point-to-point connection. If you need multiple masters or slaves, OPB arbitration can be controlled by a separate OPB bus arbiter module available from Xilinx. The VLYNQ module implements both a master and a slave OPB interface.

Basic OPB Bus Transfers

This section describes the steps required of the OPB master and OPB slave during a bus transfer to or from the VLYNQ OPB master/slave interface. The arbitration is performed by logic outside of the VLYNQ core.

- For a VLYNQ core functioning as the *master*, a transfer begins by the VLYNQ OPB master asserting its bus request signal M_REQUEST. The VLYNQ OPB master must wait until it has sampled the grant signal OPB_MGRANT before continuing (OPB_MGRANT may be tied to M_REQUEST if the VLYNQ core is the only bus master). When a grant has been received, the VLYNQ OPB master may begin a data transfer between the VLYNQ OPB master and VLYNQ OPB slave.

  After access is granted, the VLYNQ OPB master asserts the select signal M_SELECT to the OPB slave, indicating a data transaction. Data is driven and held on M_DBUS until the OPB slave acknowledges the transaction by asserting the correct acknowledge signal for the transaction that was performed. In the case of the VLYNQ OPB master, OPB_XFERACK and either OPB_BEACK or OPB_FWACK are expected (indicating a bytes-enabled or full-word transfer occurred, respectively).
- For a VLYNQ core functioning as a *slave*, a transfer begins by an OPB master asserting its bus request signal. The OPB master must wait until it has sampled a grant signal to proceed (grant is usually driven by an arbiter, but it can simply be tied to the request signal if there is only one master on the bus). When a grant has been received, the OPB master may begin a data transfer between the OPB master and VLYNQ OPB slave.

After access is granted, the OPB master asserts the select signal OPB_SELECT to the VLYNQ core, indicating a data transaction. Data is either driven and held on OPB_SLDBUS (for writes) or awaited on SL_DBUS (for reads) until the VLYNQ OPB slave acknowledges the transaction by asserting the correct acknowledge signal for the data transaction that was performed. In the case of the VLYNQ OPB slave, SL_FWACK and SL_XFERACK are asserted (indicating a full-word transfer occurred).

For additional information about the OPB bus, see information about the OPB bus, see www.xilinx.com/bvdocs/ipcenter/data_sheet/opb_ipif.pdf.

Variations from the OPB Specification

In addition to the signals specified in the OPB specification, additional sideband control signals are added to specify whether the address map or register space is being accessed, and to reduce the latency of the VLYNQ core communicating to the remote VLYNQ device. The sideband signals are SL_REG, SL_MAP, OPB_LENGTH, and M_LENGTH; their operation is described in the following section.

To allow users to create their own address map for the VLYNQ OPB slave interface, the SL_REG, and SL_MAP signals are used:

- When SL_REG is asserted, the VLYNQ OPB slave directs OPB transactions to the VLYNQ register space. If the OPB_ABUS is 0x00-0x7C, the transaction accesses the *local* registers. If the OPB_ABUS is 0x80-0xFC, the transaction accesses the *remote* registers on the remote VLYNQ device.
- When SL_MAP is asserted, the VLYNQ OPB slave directs the OPB transactions across the serial interface. SL_REG and SL_MAP are enabled by the OPB_SELECT signal, and should be considered as part of an OPB transaction. The SL_REG and SL_MAP signals should not be asserted at the same time.

To indicate the length of the current transfer in bytes, the OPB_LENGTH and M_LENGTH signals are used:

- The VLYNQ OPB *slave* interface allows for requesting multiple-cycle reads and writes by receiving the OPB_LENGTH signal.
- The VLYNQ OPB *master* interface drives the M_LENGTH signal, which can be used for status/informational purposes or for driving the OPB_LENGTH of other VLYNQ slaves.

The use of OPB_LENGTH allows for a read request through the VLYNQ serial interface to request several contiguous bytes that may be greater than the width of the data bus. The length field is sent at the beginning of a VLYNQ serial packet, and therefore the value needs to be known ahead of time. OPB_LENGTH specifies the number of bytes of the current and remaining transactions of this address (each transaction starts with an assertion of OPB_SELECT and ends with the core asserting SL_XFERACK). The address should be incremented and the length should be decremented by the same amount, which is 4 for a full-word (4-byte) transfer (possibly less if byte-enabled). The OPB_LENGTH signal must be driven to 0 when no transaction is occurring.

Aborting or terminating a transaction is not supported by the VLYNQ core. After a transaction starts, the VLYNQ core requires that it be completed. Aborting a transaction is not possible due to the transaction becoming serialized and sent to the remote VLYNQ device. Therefore, the OPB_TIMEOUT, OPB_RETRY, SL_RETRY, OPB_ERRACK, SL_ERRACK, OPB_PENDREQN signals have *not* been implemented.

The VLYNQ implementation of the OPB specification does not support burst data transfers. Double-word and half-word transactions are not supported by the VLYNQ core. A bus lock is not requested by the VLYNQ core from the OPB arbiter and is not supported. During a packetized transfer (length > 4, requiring multiple transfers) the addresses must advance in a sequential fashion; however the VLYNQ core does not verify OPB_SEQADDR, nor assert M_SEQADDR.

The following signals are not implemented, as specified in the OPB specification:

- OPB_BUSLOCK, OPB_DWACK, OPB_DWXFER, OPB_ERRACK, OPB_HWACK, OPB_HWXFER, OPB_PENDREQN, OPB_RETRY, OPB_SEQADDR, OPB_TIMEOUT
- M_BUSLOCK, M_DWACK, M_DWXFER, M_HWACK, M_HWXFER, M_SEQADDR
- SL_ERRACK, SL_RETRY

Waveforms

This following sections illustrate the VLYNQ OPB bus operation waveforms.

VLYNQ Core Master Write

Figure 7 illustrates a sample operation of the VLYNQ core performing a write to the OPB bus via the VLYNQ OPB master interface. The VLYNQ OPB master first requests control of the OPB by asserting the M_REQUEST signal after the first rising clock edge. The VLYNQ OPB master waits for the OPB arbiter to grant access to the OPB. Depending on the priority of the VLYNQ core as assigned by the OPB arbiter, the VLYNQ OPB master interface has the potential of waiting for many cycles for the OPB arbiter to grant access to the OPB. In this example, after the second rising clock edge the OPB arbiter has granted the VLYNQ OPB master control of the OPB. The VLYNQ OPB master interface samples the OPB_MGRANT signal on the third rising clock edge and determines that it has been granted access to the OPB.

After access has been granted, the VLYNQ OPB master deasserts the request signal and begins presenting the current transaction on the OPB. Each OPB transaction is indicated by assertion of the M_SELECT signal and is ended by the OPB slave device asserting the OPB_XFERACK signal. The transaction beginning after the third rising edge is a write of 4 bytes originating from the VLYNQ OPB master interface, beginning at address 0x00112233, with data 0x44556677. This is indicated by asserting the M_BEXFER, M_FWXFER, and M_DBUSEN signals. The M_RNW signal remains low to indicate a write transaction. The M_BE bus is driven to 0xF to indicate that all four byte lanes are valid. The length bus M_LENGTH is driven to 0x4 to indicate that the transaction is writing a packet of 4 bytes. At any time after the transfer has begun, the OPB slave device may respond with an acknowledge signal, however the OPB arbiter may require an acknowledge after the 16th clock cycle, unless the OPB slave has asserted the timeout suppress signal.

After the sixth rising clock edge, the OPB slave device that is mapped to address 0x00112233 has acknowledged the transaction. On the seventh rising clock edge the VLYNQ OPB master samples the OPB_XFERACK signal and determines the transfer has been successful and drives all the OPB bus signals to '0.'

The Xilinx OPB specification allows for multiple master and slave devices to use a common bus by driving unused signals to '0.' The Xilinx VLYNQ device complies with multiple master/slave combinations on a common OPB.

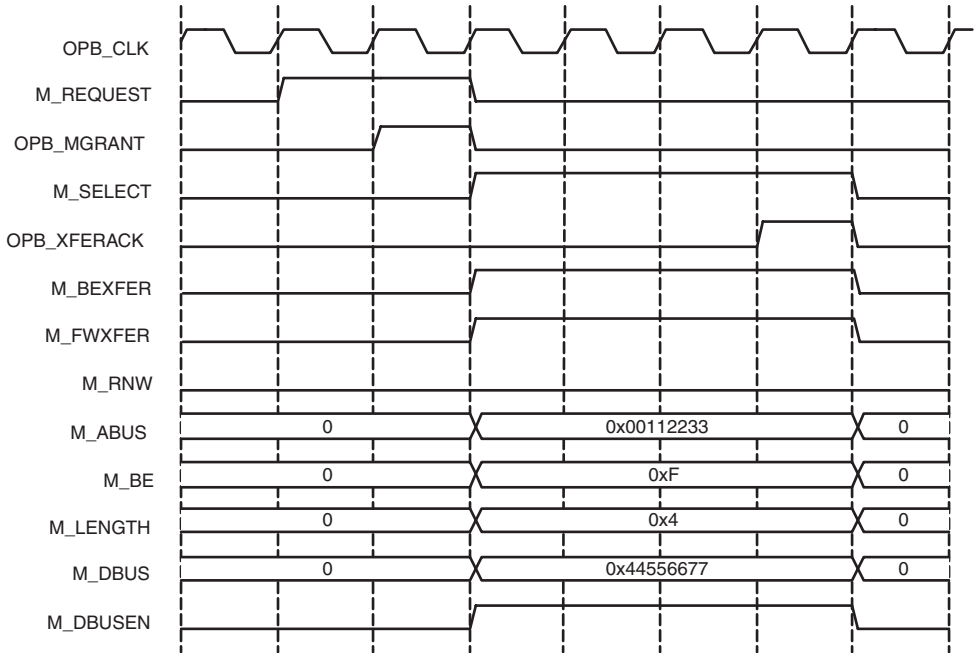

Figure 7: **VLYNQ Core Mastering OPB Bus: Single Write**

VLYNQ Core Master Read

Figure 8 illustrates a sample operation of the VLYNQ core performing a read on the OPB bus. The VLYNQ core performs a similar request grant sequence as described in the previous figure to receive control of the OPB bus. A read transaction is similar to a write transaction with a few exceptions. The `M_RNW` signal is driven high during the transaction to indicate a read transaction. Additionally the `M_DBUSEN` and `M_DBUS` signals remain at '0.' The returning data from the read request is transferred back to the VLYNQ OPB master via the `OPB_MDBUS`. When the OPB slave device has acknowledged the read transaction, the OPB slave drives the `OPB_MDBUS` with the valid return data as requested by the VLYNQ OPB master during the same period that the acknowledge signal `OPB_XFERACK` is asserted. In this example the returned data is 0x22446688.

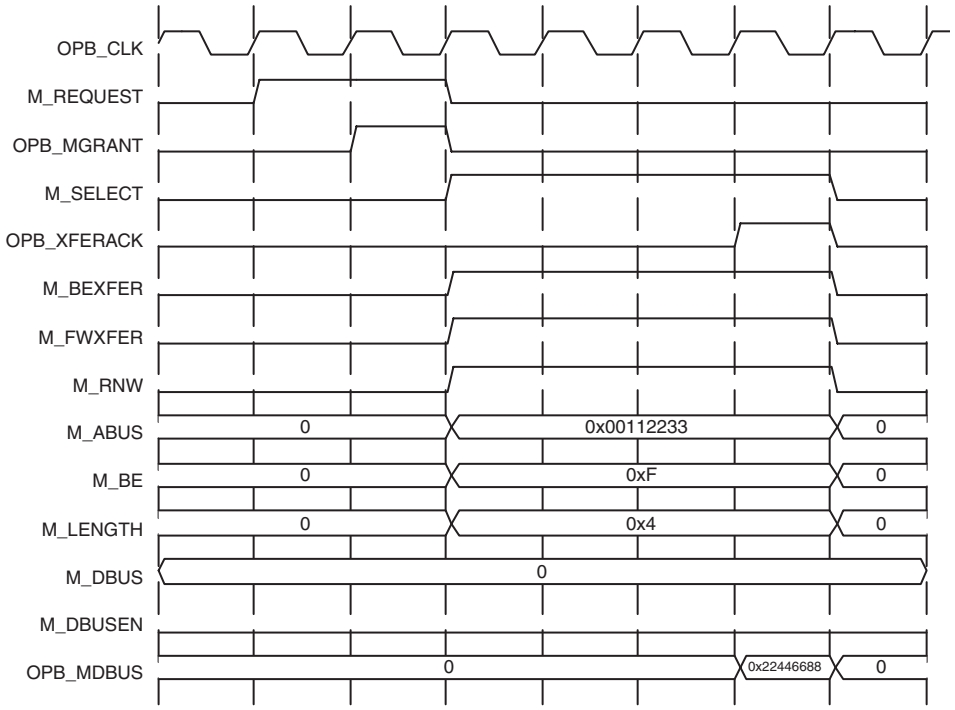

Figure 8: **VLYNQ Core Mastering OPB Bus: Single Read**

VLYNQ Core Slave Write

Figure 9 illustrates a sample operation of an OPB performing a multiple write request to consecutive addresses to the VLYNQ OPB slave interface. Because the VLYNQ OPB slave interface is unaware of the master arbitration, those signals are not shown. After the first rising clock edge, the OPB drives the OPB signals indicating a packetized (length value greater than 4) write request. It should be noted here that the process for writing or reading a packetized transfer into the VLYNQ OPB slave interface is the same process that is used by the VLYNQ OPB master interface. The OPB asserts the `OPB_SELECT`, `OPB_FWXFER`, `OPB_BEXFER`, and the `SL_REG` signals. The OPB also asserts a byte enable value of 0x7 on `OPB_BE` indicating the bits `OPB_SLDBUS[23:0]` are valid. The `OPB_SLDBUS` is driven to 0xXX001122, the XX indicating bytes that will not be written to the destination. The `OPB_ABUS` is driven to 0x0000002. The `OPB_LENGTH` is driven to 0xA indicating a packetized transaction of 10 bytes is occurring. The `SL_REG` signal is asserted high to indicate that the current transaction is to occur in the register space.

After the VLYNQ OPB slave interface has determined that the address is within its own address map, the VLYNQ OPB slave responds by asserting the `SL_TOUTSUP` signal. By asserting the timeout suppress signal, the VLYNQ OPB slave is able to process the transaction without the arbiter timing out after 16 cycles. The OPB must not change any signals on the OPB bus until the current transaction has been acknowledged.

At some later point in time the VLYNQ OPB slave responds with an acknowledge by asserting the `SL_BEACK`, `SL_FWACK` and `SL_XFERACK` signals for one cycle. The OPB samples the acknowledge and drives the signals to '0' until the OPB master receives an additional grant from the OPB arbiter.

This process continues until the packetized transfer has been completed. During the second transaction, the OPB_BE bus is driven to 0xF indicating that all bytes are enabled. The length during the second transaction is driven to 0x7 which corresponds to 0xA - 0x3 (the 0x3 value is the number of bytes that were transmitted in the previous transaction, and the 0xA value is the length of the transaction as driven in the first transaction). The OPB_SLDBUS is driven to 0x33445566, and the OPB_ABUS is driven to 0x00000005 (this value is 0x00000002+0x3).

The third transaction has an OPB_BE of 0xE corresponding to bits OPB_SLDBUS[31:8] being valid. The OPB_SLDBUS is driven to 0x778899XX, the XX indicating bytes that will not be written to the destination. The OPB_ABUS is driven to 0x00000009, which is 0x00000005+0x4 (the 0x4 value is the number of bytes that were enabled in the previous transaction).

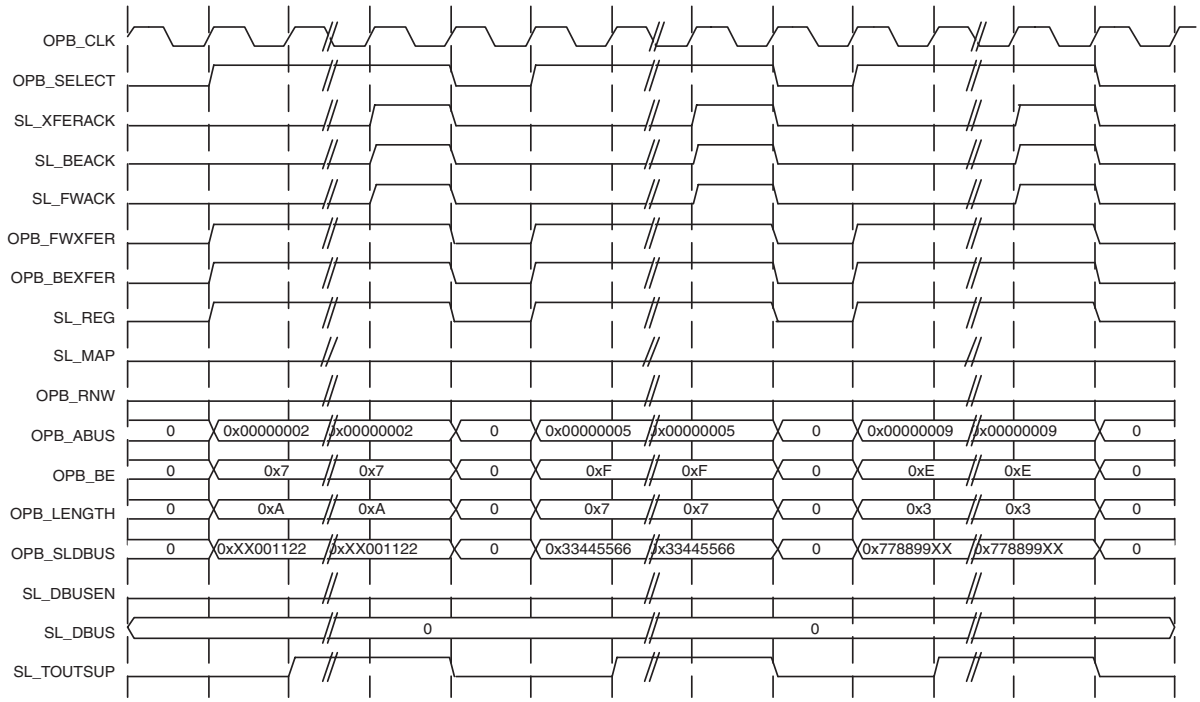

Figure 9: **VLYNQ Core Slave to OPB Bus: Multiple Writes**

VLYNQ Core Slave Read

Figure 10 illustrates a sample operation of the OPB performing a read of data. The VLYNQ core performs a similar request grant sequence as described in the previous figure to receive control of the OPB. The packetized read operation is similar to the packetized write transaction. The first transaction is a read request of 5 bytes with an initial byte enable of 0x1 of address 0x00001000 of the VLYNQ Map space. The VLYNQ OPB slave interface has a high latency during the first read of a packetized read transaction. This is due to the request having to be sent over the serial interface, processed remotely, and returned once again over the serial interface. However, because the length of the packetized transfer was transmitted with the initial read request, the remaining read request data to be returned starts transmission from the remote device and is locally buffered until it is formally requested through the OPB interface, which reduces the latency of future read requests within the same packet. Once requested data has been returned from the remote VLYNQ device, the VLYNQ OPB slave interface asserts the SL_BEACK, SL_FWACK, and SL_XFERACK signals in addition to presenting the requested data on the SL_DBUS (0xXXXXXXCC in the figure), and asserting SL_DBUSEN.

The second and last transaction shown indicates a byte enable of 0xF and a remaining length of 0x4. An address of 0x00001001 is given which is the result of the addition 0x00001000+0x1, with the value 0x1 being the number of bytes transferred in the first transaction. The VLYNQ OPB slave responds to the last transaction in this packetized transfer with data 0xDDEEFFAA.

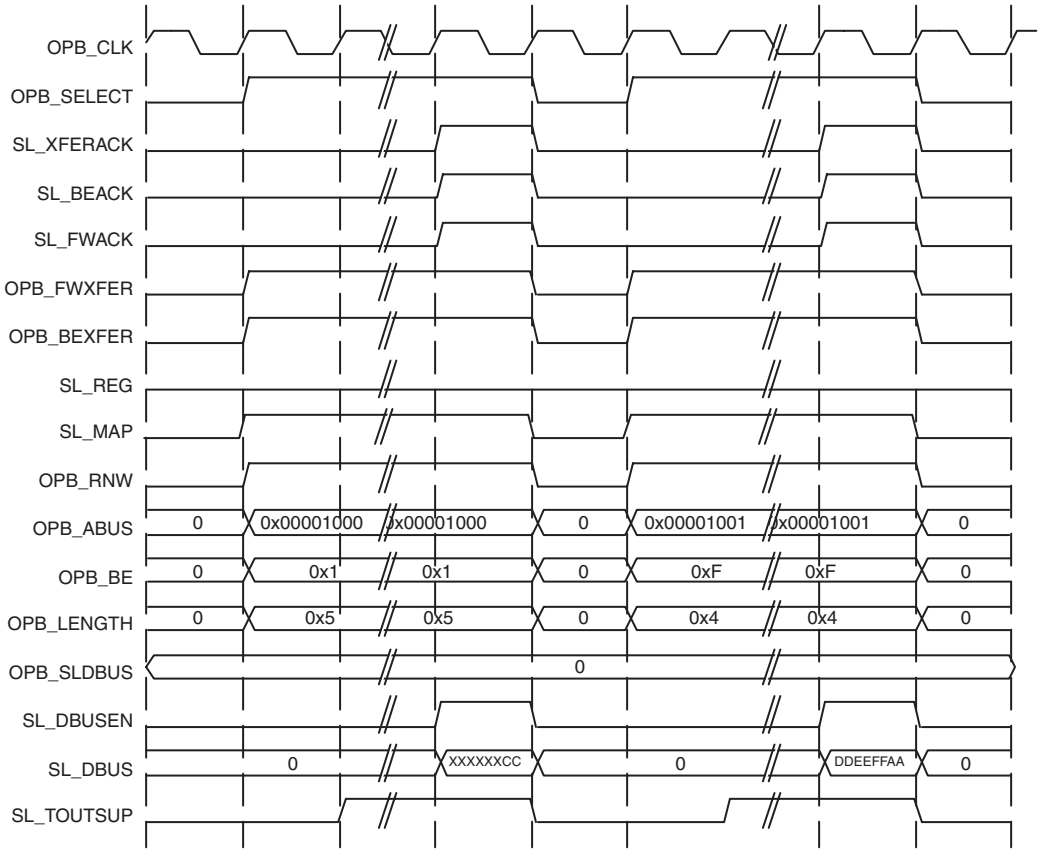

Figure 10: **VLYNQ Core Slave to OPB Bus: Multiple Reads**

Signal List

Table 1: **Clocks and Reset Interface**

Name	Direction	Description
OPB_CLK	Input	**OPB Clock**: Clocks the OPB logic.
OPB_RST_N	Input	**OPB Reset**: Reset signal used to reset the VLYNQ core. This reset is synchronized internal to the core to reset the VLYNQ serializer logic.
VLYNQ_TCLK	Input	**VLYNQ Transmit Clock**: This clocks the VLYNQ transmit logic.
VLYNQ_RCLK	Input	**VLYNQ Receive Clock**: This clocks the VLYNQ receive logic.
VLYNQ_CLK_REF	Input	**VLYNQ Reference Clock**: The VLYNQ core can generate a clock output that is either precisely this clock (VLYNQ_CLK_FULL) or a divided down version of this clock (VLYNQ_CLK_DIV).
VLYNQ_CLK_FULL	Output	**VLYNQ Full-Clock Output**: A clock output that is simply a pass-through of VLYNQ_CLK_REF.
VLYNQ_CLK_DIV	Output	**VLYNQ Divided-Clock Output**: A clock output that has a period of VLYNQ_CLK_REF divided by the Control Register field Serial Clock Output Divider+1, for values of Serial Clock Output Divider other than 0. The duty cycle of this clock is not guaranteed to be 50/50.
VLYNQ_CLK_SELECT	Output	**VLYNQ Clock Select**: Used to permit logic external to the core to select between VLYNQ_CLK_FULL and VLYNQ_CLK_DIV. Driven to '1' if the value of the Control Register field Serial Clock Output Divider is '000' and '0' if otherwise.
VLYNQ_CLK_OE_N	Output	**VLYNQ Clock-Output Enable**: Used to permit logic external to the core to determine if one of the core clock outputs (or some other local source) should drive the device VLYNQ_CLK pad, or if it should be tri-stated to allow a remote device to drive. This is set by Control Register field Clkdir; when '0,' causes VLYNQ_CLK_OE_N to be '1,' and when '1,' causes VLYNQ_CLK_OE_N to be '0.'
VLYNQ_RSTCLK_O_N	Output	**VLYNQ Clock-Logic Reset Output**: A reset that can be used for the clock output logic. The core uses this internally-generated reset for the clock-division circuitry, and any synchronous logic that is in between VLYNQ_CLK_DIV and the VLYNQ_CLK output pad (such as a DDR flop) should use this reset. This reset is OPB_RESET synchronized to VLYNQ_CLK_REF.
VLYNQ_RST_O_N	Output	**VLYNQ Reset Output**: A reset that can be used for Tx or Rx I/O logic external to the core. This reset is OPB_RESET synchronized to VLYNQ_RCLK.

Table 2: **Serial Data Interface**

Name	Direction	Description
VLYNQ_RXD[3:0]	Input	**VLYNQ Receive Data**: The signal(s) from which data is driven by the remote device to this VLYNQ core. Implemented width may be from 1 to 4 bits. If the number of signal on the core interface exceeds the number of lanes to be implemented, tie off any unused upper bits to ground.
VLYNQ_TXD[3:0]	Output	**VLYNQ Transmit Data:** The signal(s) from which data is driven by the VLYNQ core to the remote device. Implemented width may be from 1 to 4 bits. If the number of signal on the core interface exceeds the number of lanes to be implemented, leave unused upper bits unconnected.

Table 3: **Interrupt Interface**

Name	Direction	Description
VLYNQ_INT[7:0]	Input	**VLYNQ Interrupt Bus**: Used to indicate the core should set the appropriate bit of the Interrupt Pending/Set Register, which causes either generation of a local interrupt output or transmission of an interrupt packet. For a given VLYNQ_INT[i], the core requires that the Register Space value Interrupt Enable i be set, and that Interrupt Polarity i and Interrupt Type i properly describe the nature of how VLYNQ_INT[i] is signaled. The bit position of the Interrupt Pending/Set Register is determined by Interrupt Vector i. If interrupts are not to be locally generated, tie this bus low.
VLYNQ_INTPLS	Output	**VLYNQ Local Interrupt Pulsed Output**: If Interrupt Local is set in the Control Register, this is used to indicate that a bit in the Interrupt Status/Clear Register has been cleared while VLYNQ_INTLVL is asserted. This signal pulses for one cycle.
VLYNQ_INTLVL	Output	**VLYNQ Local Interrupt Level Output**: If Interrupt Local is set in the Control Register, this is used to indicate that a bit in the Interrupt Status/Clear Register is set. This signal remains asserted until the register is cleared.

Table 4: **Control/Status Interface**

Name	Direction	Description
DEVICE_ID[15:0]	Input	**VLYNQ Device ID**: The Device ID value read out of the Chip Version Register. The device must be registered with the VLYNQ IP Group to be assigned an ID.
DEVICE_REV[15:0]	Input	**VLYNQ Device Revision**: The revision of the design of this device.
DEFAULT_CLKDIR	Input	**Default Serial Clock Source**: The power-up/reset value of Control Register[15], which is the Serial_Clock_Direction.
CMDFIFO_FREE[4:0]	Output	**Outbound Command FIFO Free Space**: The number of free entries left in the OPB Slave Outbound Command FIFO.

Table 4: **Control/Status Interface *(Continued)***

Name	Direction	Description
RETURNFIFO_USED [4:0]	Output	**OPB Slave Return Data FIFO Count**: The number of words in the OPB Slave Return Data FIFO.
IOTXFASTPATH	Output	**I/O Tx Fast Path**: This represents the value of Control Register[21]. It is used to support user logic to select between Tx output delay paths.
IOTESTENABLE	Output	**I/O Test Enable**: This causes data coming in on VLYNQ_RXD to be output onto VLYNQ_TXD.

Note: In order to be aligned with TI VLYNQ nodes, the command FIFO has a depth of 24. Since FPGA internal memory primitives can only be configured to sizes that are powers of 2. Hence when the command FIFO is filled with 24 values, the FIFO is shown as full (wused=24) and no further writes are possible to it, -1 (that is, (2^8) - 1), with the closest depth configuration being a depth of 31. Because of this, the 'cmdfifo_free' output will show a value of 0x07 (which is 31-24), instead of the expected 0x00.

Table 5: **Power Management**

Name	Direction	Description
CRUN_IN_N	Input	**SCRUN# Input**: Used to pass the SCRUN_N signal (for serial clock power management) into the VLYNQ core (if this feature is implemented). It is expected that SCRUN_N has a pullup on it (internal or external to FPGA), as it is either driven to '0' or tri-stated. If serial clock power management is not implemented, tie this to '0.'
CRUN_OUT_N	Output	**SCRUN# Output**: Meant to be the value driven onto SCRUN_N when the serial clock is to be running; output should be tri-stated otherwise. Tied to '0' internally.
CRUN_OE_N	Output	**SCRUN# Output Enable**: Determines if SCRUN_N is to be driven to '0' (which indicates the serial clock should be toggling), or is tri-stated (which indicates this core does not need the clock). Signal is active-low, so '0' indicates output should be driven to '0.'
CRUN_PE_N	Output	**SCRUN# Pullup Enable**: Permits the pullup attached to SCRUN_N to be disabled for power savings. The pullups in Xilinx IOBs cannot be dynamically turned on or off, so this signal can only be used if the pullup is implemented external to the Xilinx device. Signal is active-low, so '0' indicates pullup should be disabled.
SERIAL_BUSY	Output	**Serial Interface Busy**: Indicates there are transactions pending or in progress on the VLYNQ serial bus. This is used to determine if it is possible to activate power-saving options.
SERIAL_BUSY_REQ	Output	**Serial Interface Busy/Request**: Indicates that either there are transactions pending or in progress on the VLYNQ serial bus or the remote device wishes to restart the serial clock. This is used to determine if it is possible to activate power-saving options.

Table 5: **Power Management *(Continued)***

Name	Direction	Description
VLYNQ_SCLK_PE_N	Output	**VLYNQ Serial Clock Pullup Enable**: Used to cause any pullups attached to the VLYNQ_CLK output to be enabled if the remote VLYNQ device is responsible for driving the clock; it is disabled otherwise for power savings. The signal also causes pullups to be disabled if the Control Register field Serial Clock Pullup Disable is set. If the pullup is to be enabled, the signal drives a '0,' otherwise it drives a '1.' Although Xilinx devices have pullups available in their IOBs, they cannot be selectively turned on or off. This signal is only useful if pullups are implemented outside the Xilinx device.
VLYNQ_RXD_PE_N [3:0]	Output	**VLYNQ Receive Data Pullup Enable**: Used to cause any pullups attached to the VLYNQ_RXD output to be enabled if the link to the remote device is not yet established, and disabled otherwise for power savings. If the pullup is to be enabled, the signal drives a '0,' otherwise it drives a '1.' Although Xilinx devices have pullups available in their IOBs, they cannot be selectively turned on or off. This signal is only useful if pullups are implemented outside the Xilinx device.
OPB_BUSY	Output	**OPB Interface Busy**: Indicates there are transactions pending or in progress on the OPB bus.

Table 6: **Slave OPB Interface**

Name	Direction	Description
OPB_ABUS[31:0]	Input	**OPB Address Bus**: The address of the VLYNQ OPB slave read or write transaction.
OPB_SLDBUS[31:0]	Input	**OPB Slave Data Bus**: Used to transfer data between OPB masters and slaves. It is used to write data to the VLYNQ OPB slave interface.
OPB_BE[3:0]	Input	**OPB Byte Enable**: Indicates which byte lanes are valid on the OPB_SLDBUS; OPB_BE[i] indicates byte OPB_SLDBUS[7+8i : 8i] is valid. This allows masters to perform unaligned multi-byte operations in a single OPB transfer. Non-contiguous byte enables are not allowed.
OPB_SELECT	Input	**OPB Select**: Indicates that a valid transfer is in progress. This signal qualifies all master control signals and is the enable for OPB_ABUS, OPB_SLDBUS, OPB_BE, OPB_BEXFER, OPB_DWXFER, OPB_FWXFER, OPB_HWXFER, OPB_RNW, and OPB_LENGTH. The OPB master may NOT terminate the current transfer by deasserting the OPB_SELECT signal.
OPB_BEXFER	Input	**OPB Byte Enable Transfer**: Asserted during byte enabled transfers. The assertion of OPB_BEXFER is a request for the VLYNQ OPB slave interface to transfer the byte lanes indicated via the asserted OPB_BE[3:0] signals. This signal must be asserted during all valid transactions received by the VLYNQ module.
OPB_FWXFER	Input	**OPB Full Word Transfer**: Used to indicate that the current transaction is a full word (32-bits) transfer. Half word and double word are not supported.

Table 6: **Slave OPB Interface *(Continued)***

Name	Direction	Description
OPB_RNW	Input	**OPB Read Not Write**: Used to indicate the direction of the data transfer. When the signal is high the request is for the VLYNQ OPB slave to supply data to be read into the master. If the signal is low the request is for the VLYNQ OPB slave to accept write data from the master.
OPB_LENGTH[9:0]	Input	**OPB Length**: An additional signal added to the OPB interface to indicate packetized transfers. This signal indicates the remaining length of the packet including the current byte count of the current transaction. To reduce latency of reads and writes over the VLYNQ serial interface, transfers are given a length in an attempt to pipeline transfers. If it is known that sequential reads or writes are to be performed, this signal should be used to indicate how many bytes are to be transferred. The OPB master is required to request as many transactions as specified by the given length and may not terminate the packetized transfer once it has begun.
SL_REG	Input	**OPB Slave Register Transaction**: Used to indicate to the destination of the current transfer. When this signal is high the request is for the VLYNQ OPB slave to direct the transaction to the register space. SL_REG and SL_MAP should not be asserted simultaneously.
SL_MAP	Input	**OPB Slave Map Transaction**: Used to indicate to the destination of the current transfer. When this signal is high the request is for the VLYNQ OPB slave to direct the transaction over the VLYNQ serial interface to the remote VLYNQ device. SL_REG and SL_MAP should not be asserted simultaneously.
SL_BEACK	Output	**OPB Slave Byte Enable Acknowledge**: Used to acknowledge a byte enabled transaction by the VLYNQ slave interface.
SL_DBUS[31:0]	Output	**OPB Slave Data Bus**: Used to transfer data between the VLYNQ OPB slave and the OPB bus.
SL_DBUSEN	Output	**OPB Slave Data Bus Enable**: Used to indicate to the requesting master device that the data presented on the SL_DBUS is valid.
SL_FWACK	Output	**OPB Slave Full Word Acknowledge**: Used to acknowledge a full word transmission. Double word and half word transfers are not permitted.
SL_TOUTSUP	Output	**OPB Slave Timeout Suppress**: Asserted by the VLYNQ OPB slave to indicate to the OPB Arbiter that the bus operation will be delayed for an extended period of time.
SL_XFERACK	Output	**OPB Slave Transfer Acknowledge**: Asserted by the VLYNQ OPB slave to indicate the completion of a data transfer.

Table 7: **Master OPB Interface**

Name	Direction	Description
M_ABUS[31:0]	Output	**OPB Master Address Bus**: The address of the OPB read or write transaction.
M_DBUS[31:0]	Output	**OPB Master Data Bus**: Used to transfer data between VLYNQ OPB master and the OPB bus.
M_BE[3:0]	Output	**OPB Master Byte Enable**: Indicates which byte lanes are valid on the M_DBUS; M_BE[i] indicates byte M_DBUS[7+8i : 8i] is valid. This allows the VLYNQ OPB master to perform unaligned multi-byte operations in a single OPB transfer.
M_LENGTH[9:0]	Output	**OPB Master Length**: A sideband signal on the VLYNQ OPB master interface to communicate the overall length of a packetized transfer, including the remaining current byte count of the current transaction. This signal indicates the remaining length of the packet. This signal is included to allow for multiple VLYNQ cores to inter-communicate correctly over the OPB, and may also be used for status or informational purposes.
M_SELECT	Output	**OPB Master Select**: Asserted by the VLYNQ OPB master to assume control of the OPB bus and to indicate that a valid data transfer cycle is in progress.
M_FWXFER	Output	**OPB Master Full Word Transfer**: Used by the VLYNQ OPB master to indicate that the current transaction is a full word transfer. Double word and half word transfers are not generated by the VLYNQ OPB master.
M_RNW	Output	**OPB Master Read Not Write**: Used to indicate the direction of the data transfer. When the signal is high the request is for the OPB to supply data to be read into the VLYNQ OPB master. If the signal is low, the request is for the OPB to accept write data from the VLYNQ OPB master.
M_REQUEST	Output	**OPB Master Request**: Asserted by the VLYNQ OPB master to request control of the bus, which is granted by the OPB arbiter via the OPB_MGRANT signal.
M_BEXFER	Output	**OPB Master Byte Enable Transfer**: Asserted by the VLYNQ OPB master to indicate that the current transfer uses the byte enabled signal M_BE. The VLYNQ OPB master performs all transactions byte enabled.
M_DBUSEN	Output	**OPB Master Data Bus Enable**: Asserted by the VLYNQ OPB master it indicate that the data present on the M_DBUS signal is valid.
OPB_MGRANT	Input	**OPB Master Grant**: Asserted by the OPB arbiter to grant control of the OPB bus to the VLYNQ OPB master. If an external OPB arbiter is not desired and the VLYNQ core is the only OPB master, it is possible to wire the M_REQUEST signal to the OPB_MGRANT signal.
OPB_XFERACK	Input	**OPB Transfer Acknowledge**: Asserted by the OPB to indicate the completion of a data transfer between the VLYNQ OPB master and the OPB slave.

Table 7: **Master OPB Interface *(Continued)***

Name	Direction	Description
OPB_BEACK	Input	**OPB Byte Enable Acknowledge**: Asserted by the OPB to indicate that the OPB has correctly received the byte enabled transfer.
OPB_FWACK	Input	**OPB Full Word Acknowledge**: Asserted by the OPB to indicate that the OPB has correctly received the full word transfer. Double word and half word transfers are not permitted.
OPB_MDBUS[31:0]	Input	**OPB Master Data Bus**: Used to transfer data between OPB masters and slaves. It is used to provide read data to the master interface.

Register Space

The register space consists of a total of 256 contiguous bytes. The first 128 bytes access the local VLYNQ registers, and the remaining 128 bytes access the remote VLYNQ registers using the serial interface. Table 8 summarizes the Register Space assignments. A description of each register follows.

Table 8: **Register Map**

Address Offset	Register
0x00	Revision/ID Register
0x04	Control Register
0x08	Status Register
0x0C	Interrupt Priority Vector Status/Clear Register
0x10	Interrupt Status/Clear Register
0x14	Interrupt Pending/Set Register
0x18	Interrupt Pointer Register
0x1C	Tx Address Map Register
0x20	Rx Address Map Size 1 Register
0x24	Rx Address Map Offset 1 Register
0x28	Rx Address Map Size 2 Register
0x2C	Rx Address Map Offset 2 Register
0x30	Rx Address Map Size 3 Register
0x34	Rx Address Map Offset 3 Register
0x38	Rx Address Map Size 4 Register
0x3C	Rx Address Map Offset 4 Register
0x40	Chip Version Register
0x44	Auto Negotiation Register
0x48	reserved
0x4C	Negotiation Status Register
0x50-0x5C	reserved

Table 8: **Register Map *(Continued)***

Address Offset	Register
0x60	Interrupt Vector 3-0 Register
0x64	Interrupt Vector 7-4 Register
0x68-0x7C	reserved for Interrupt Vectors 8-31
0x80	Remote Revision/ID Register
0x84	Remote Control Register
0x88	Remote Status Register
0x8C	Remote Interrupt Priority Vector Status/Clear Register
0x90	Remote Interrupt Status/Clear Register
0x94	Remote Interrupt Pending/Set Register
0x98	Remote Interrupt Pointer Register
0x9C	Remote Tx Address Map Register
0xA0	Remote Rx Address Map Size 1 Register
0xA4	Remote Rx Address Map Offset 1 Register
0xA8	Remote Rx Address Map Size 2 Register
0xAC	Remote Rx Address Map Offset 2 Register
0xB0	Remote Rx Address Map Size 3 Register
0xB4	Remote Rx Address Map Offset 3 Register
0xB8	Remote Rx Address Map Size 4 Register
0xBC	Remote Rx Address Map Offset 4 Register
0xC0	Remote Chip Version Register
0xC4	Remote Auto Negotiation Register
0xC8	reserved
0xCC	Remote Negotiation Status Register
0xD0-0xDC	reserved
0xE0	Remote Interrupt Vector 3-0 Register
0xE4	Remote Interrupt Vector 7-4 Register
0xE8-0xFC	Reserved for Remote Interrupt Vectors 8-31

Revision Register

The Revision Register contains the major and minor revisions of the VLYNQ core.

Table 9: **VLYNQ Revision Register (Base Address + 0x00)**

Bits	Type	Description
31:16	r/o	**Unique Module ID:** This is the unique module ID for the Xilinx VLYNQ core. This release number is 0x0001.
15:8	r/o	**Major Revision:** The major release number for this version of the VLYNQ core is 0x2.
7:0	r/o	**Minor Revision:** The minor release number for this version of the VLYNQ core is 0x6.

Control Register

The Control Register determines the operation of the VLYNQ core.

Table 10: **VLYNQ Control Register (Base Address + 0x04)**

Bits	Type	Description
31	r/w	**Power Management Enabled:** When set to '1,' VLYNQ_CLK_DIV goes into idle mode with a value of '1' when there is no traffic over the serial bus. This bit must not be set if the SCRUN_N pad is implemented but not connected to the remote device.
30	r/w	**Serial Clock Pullup Disable**: When set to '1,' VLYNQ_SCLK_PE_N is forced to '1,' disabling any pullups you have implemented, to achieve power savings. Although Xilinx devices have pullups available in their IOBs, they cannot be selectively turned on or off. This field is only useful if pullups are implemented outside the Xilinx device.
29:22	r/o	Reserved: Reads return 0, writes have no effect.
21	r/w	**Transmit Fast Path**: Indicates that the transmit data should take a different (presumably faster) path to the output pin of the FPGA. Note there is no logic implemented in the VLYNQ core to implement this; any support must be implemented by you. This register bit is reflected on IOTXFASTPATH.
20:19	r/o	Reserved: Reads return 0, writes have no effect.
18:16	r/w	**Serial Clock Output Divider**: The period of the divided clock output VLYNQ_CLK_DIV is VLYNQ_CLK_REF/(1+*Serial_Clock_Output_Divider*). If this value is "000" it is interpreted as the output period is the same as VLYNQ_CLK_REF, but VLYNQ_CLK_DIV is not capable of toggling at the same frequency as VLYNQ_CLK_REF. In this case, the signal VLYNQ_CLK_SELECT asserts to '1,' to permit external MUXing to select VLYNQ_CLK_FULL instead of VLYNQ_CLK_DIV.
15	r/w	**Serial Clock Direction**: Determines if clock on the VLYNQ_CLK pad is an input from a remote device ('0'), or an output from the local clock source ('1'). The reset value is determined by the signal DEFAULT_CLKDIR.
14	r/w	**Interrupt Local**: Determines whether interrupts are posted in the local Interrupt Status/Clear Register ('1'), or whether they are forwarded out of the serial interface as a packet to the remote device ('0'). If they are local, the interrupt signals VLYNQ_INTLVL and VLYNQ_INTPLS can assert.
13	r/w	**Interrupt Enable**: Permits VLYNQ module status interrupts (errors reported in Status Register[8:7]) to be posted in the Interrupt Pending/Set Register at bit position *Interrupt_Vector*.
12:8	r/w	**Interrupt Vector**: Bit position of the Interrupt Pending/Set Register that is set if a VLYNQ module status interrupt occurs and Interrupt_Enable is set.

Table 10: **VLYNQ Control Register (Base Address + 0x04) *(Continued)***

Bits	Type	Description
7	r/w	**Interrupt to Configuration Register**: Indicates that Interrupt packets received from the remote device should be flagged in the Interrupt Pending/Set Register. If clear, the interrupt packet is passed onto the OPB interface, using the value in Interrupt Pointer Register[31:2] as the address.
6:3	r/o	Reserved: Reads return 0, writes have no effect.
2	r/w	**Address Optimization Disable**: Disables address optimization for version 2.X mode packets. No effect for version 1.X mode. Address optimization causes a packet's address bytes that have not changed from the previous packet's address bytes to be dropped from the packet transmission.
1	r/w	**Internal Loopback**: When set to '1,' causes transmitted data to be wrapped back to the receive data path.
0	r/w	**Software Reset**: When set to '1,' the state machines are reset, the serial interface disabled, and the link is lost.

Status Register

The Status Register is used to detect conditions that may be of interest to the system designer.

Table 11: **VLYNQ Status Register (Base Address + 0x08)**

Bits	Type	Description
31:28	r/o	Reserved: Reads return 0, writes have no effect.
27:24	r/o	**Rx Data Width:** Number of bit lanes detected on VLYNQ_RXD.
23:20	r/o	**Tx Data Width:** Number of bit lanes detected on VLYNQ_TXD.
19:12	r/o	Reserved: Reads return 0, writes have no effect.
11	r/o	**RTM Present:** Always read as 0. This core version does not include RTM.
10	r/o	**Inbound Flow Control:** A flow control enable symbol has been received, and packet transmission stalls until a flow control disable symbol is received.
9	r/o	**Outbound Flow Control:** This core's OPB Master Read Return Data FIFO or the OPB Slave Inbound Command FIFO is in danger of overflowing, so a flow control enable symbol has been sent.
8	w/c	**Remote Error:** A downstream VLYNQ module has detected a packet error. This bit is set when the /E/ symbol (K28.1, an 8b/10b control character) is received on the serial interface. If set, this bit causes an interrupt if enabled in Control Register[13]. Write a '1' to it to clear.
7	w/c	**Local Error:** This core has detected an error on an inbound packet. If set, this bit causes an interrupt if enabled in Control Register[13]. Write a '1' to it to clear.
6	r/o	**OPB Slave Outbound Command FIFO Not Empty:** There is at least one command or write data pending in the OPB Slave Outbound Command FIFO.
5	r/o	**OPB Slave Read Return FIFO Not Empty:** There is at least one word of read data to be returned to you in the OPB Slave Read Return FIFO.
4	r/o	**OPB Master Inbound Command FIFO Not Empty:** There is at least one command or write data pending in the OPB Master Inbound Command FIFO.
3	r/o	**OPB Master Read Return FIFO Not Empty:** There is at least one word of read data to be returned to the remote device in the OPB Master Read Return FIFO.

Table 11: **VLYNQ Status Register (Base Address + 0x08) *(Continued)***

Bits	Type	Description
2	r/o	**Pending OPB Master Requests:** A request has been detected on the OPB Master interface.
1	r/o	**Pending OPB Slave Requests:** A request has been asserted on the OPB Slave interface.
0	r/o	**Link Established:** The Serial Interface initialization sequence has completed.

Interrupt Priority Vector Status/Clear Register

When read, the Interrupt Priority Vector Status/Clear Register displays the highest priority vector with a pending interrupt. When writing, only bits [4:0] are valid and the value represents the vector of the interrupt to be cleared.

Table 12: **VLYNQ Interrupt Priority Vector Status/Clear Register (Base Address + 0x0C)**

Bits	Type	Description
31	r/o	**Interrupt Pending:** If there are pending interrupts from the Interrupt Status/Clear Register, this reads as a '0,' otherwise it reads as '1.'
30:5	r/o	Reserved: Reads return 0, writes have no effect.
4:0	r/w	**Highest Priority Interrupt:** The bit position of the highest priority interrupt pending in the Interrupt Status/Clear Register, assuming bit 0 is the highest priority and bit 31 is the lowest priority. Writing a value into this field can clear the interrupt in the corresponding bit position in the Interrupt Status/Clear Register.

Interrupt Status/Clear Register

The Interrupt Status/Clear Register indicates the unmasked interrupt status. Writing a '1' to any bit in this register clears the corresponding interrupt.

Table 13: **VLYNQ Interrupt Status/Clear Register (Base Address + 0x10)**

Bits	Type	Description
31:0	r/w	**Interrupt Status:** Indicates the unmasked status of each interrupt. Writing a '1' to any bit clears the corresponding interrupt. When the Interrupt_Local bit is set in the Control Register, the `VLYNQ_INTLVL` pin is driven high when any bit in this register is set.

Interrupt Pending/Set Register

The Interrupt Pending/Set Register indicates the pending interrupt status when the Interrupt_Local bit in the Control Register is not set. When the interrupt packet is forwarded on the serial interface, these bits are cleared.

Table 14: **VLYNQ Interrupt Pending/Set Register (Base Address + 0x14)**

Bits	Type	Description
31:0	r/w	**Interrupt Status:** Indicates the unmasked status of each interrupt. Writing a '1' to any bit sends an interrupt packet on the serial interface if the Interrupt_Local bit is not set in the Control Register. If Interrupt_Local is set, the contents of the Interrupt Status/Clear Register becomes the OR of that register and this register, and this register clears.

Interrupt Pointer Register

The Interrupt Pointer Register should be written with the address of the interrupt set register for the device. This register should contain the address of either the interrupt set register of a interrupt controller module implemented outside the VLYNQ core, or the Interrupt Pending/Set Register within the VLYNQ core. If the Interrupt Pending/Set Register within the VLYNQ core is the intended set method, this register should be programmed to 0x14, and the Interrupt_to_Configuration_Register bit of the Control Register should be set to '1.'

Table 15: **VLYNQ Interrupt Pointer Register (Base Address + 0x18)**

Bits	Type	Description
31:2	r/w	**Interrupt Pointer:** The address of the Interrupt Set Register, which may be any memory-mapped address, including the VLYNQ core module itself.
1:0	r/o	Reserved: Reads return 0, writes have no effect.

Tx Address Map Register

The Tx Address Map Register is used to translate transmit packet addresses to remote device OPB addresses.

Table 16: **VLYNQ Tx Address Map Register (Base Address + 0x1C)**

Bits	Type	Description
31:2	r/w	**Tx Address Map Offset:** The address offset of OPB Master addresses on OPB_ABUS to the VLYNQ core. This value is subtracted from the value on OPB_ABUS to obtain the zero relative transmit packet address, and that value is used as the address field in the serial packet.
1:0	r/o	Reserved: Reads return 0, writes have no effect.

Rx Address Map Size 1 Register

The Rx Address Map Size 1 Register is used to identify the intended destination of inbound serial packets. If an inbound packet address satisfies the conditions:

$$Inbound\ Address < Rx\ Address\ Size1$$

the address is translated to:

$$OPB\ Address = Inbound\ Address + Rx\ Address\ Offset1$$

Table 17: **VLYNQ Rx Address Map Size 1 Register (Base Address + 0x20)**

Bits	Type	Description
31:2	r/w	**Rx Address Map Size 1:** This field is used to determine if received packets are destined for the 1st of 4 mapped address regions. This field is compared with the address contained in the received packet, and if the received address is less than this value, the packet address is added to the Rx Address Map Offset 1 Register to obtain the translated address.
1:0	r/o	Reserved: Reads return 0, writes have no effect.

Rx Address Map Offset 1 Register

The Rx Address Map Offset 1 Register is used with the Rx Address Map Size 1 Register to translate receive packet addresses to local device OPB addresses.

Table 18: **VLYNQ Rx Address Map Offset 1 Register (Base Address + 0x24)**

Bits	Type	Description
31:2	r/w	**Rx Address Map Offset 1:** This field is used with the Rx Address Map Size 1 Register to determine the translated address for serial data. If the received packet address is less than the value in the Rx Address Map Size 1 Register, the packet address is added to the contents of this register to obtain the translated address.
1:0	r/o	**Reserved:** Reads return 0, writes have no effect.

Rx Address Map Size 2 Register

The Rx Address Map Size 2 Register is used to identify the intended destination of inbound serial packets. If an inbound packet address satisfies the conditions:

$$Inbound\ Address >= Rx\ Address\ Size1$$

$$Inbound\ Address < Rx\ Address\ Size1 + Rx\ Address\ Size2$$

the address is translated to:

$$OPB\ Address=Inbound\ Address + Rx\ Address\ Offset2 - Rx\ Address\ Size1$$

Table 19: **VLYNQ Rx Address Map Size 2 Register (Base Address + 0x28)**

Bits	Type	Description
31:2	r/w	**Rx Address Map Size 2:** This field is used to determine if received packets are destined for the 2nd of 4 mapped address regions. The sum of this field and Rx Address Map Size 1 is compared with the address contained in the received packet. If the received packet address is less than this sum but greater than or equal to the value in Rx Address Map Size 1, the packet address, minus Rx Address Map Size 1, is added to the Rx Address Map Offset 2 Register to obtain the translated address.
1:0	r/o	Reserved: Reads return 0, writes have no effect.

Rx Address Map Offset 2 Register

The Rx Address Map Offset 2 Register is used with the Rx Address Map Size 2 Register to translate receive packet addresses to local device OPB addresses.

Table 20: **VLYNQ Rx Address Map Offset 2 Register (Base Address + 0x2C)**

Bits	Type	Description
31:2	r/w	**Rx Address Map Offset 2:** This field is used with the Rx Address Map Size 1 and 2 Registers to determine the translated address for serial data. If the received packet address is less than the sum of the Rx Address Map Size 1 and 2 Registers but greater than or equal to the value in Rx Address Map Size 1, the difference between the packet address and Rx Address Map Size 1 is added to the contents of this register to obtain the translated address.
1:0	r/o	Reserved: Reads return 0, writes have no effect.

Rx Address Map Size 3 Register

The Rx Address Map Size 3 Register is used to identify the intended destination of inbound serial packets. If an inbound packet address satisfies the conditions:

Inbound Address >= Rx Address Size1 + Rx Address Size2

Inbound Address < Rx Address Size1 + Rx Address Size2 + Rx Address Size3

the address is translated to:

OPB Address=Inbound Address + Rx Address Offset3 - Rx Address Size1 - Rx Address Size2

Table 21: **VLYNQ Rx Address Map Size 3 Register (Base Address + 0x30)**

Bits	Type	Description
31:2	r/w	**Rx Address Map Size 3:** This field is used to determine if received packets are destined for the 3rd of 4 mapped address regions. The sum of this field and Rx Address Map Size 1 and 2 is compared with the address contained in the received packet. If the received packet address is less than this sum but greater than or equal to the sum of Rx Address Map Size 1 and 2, the packet address, minus Rx Address Map Size 1 and 2, is added to the Rx Address Map Offset 3 Register to obtain the translated address.
1:0	r/o	Reserved: Reads return 0, writes have no effect.

Rx Address Map Offset 3 Register

The Rx Address Map Offset 3 Register is used with the Rx Address Map Size 3 Register to translate receive packet addresses to local device OPB addresses.

Table 22: **VLYNQ Rx Address Map Offset 3 Register (Base Address + 0x34)**

Bits	Type	Description
31:2	r/w	**Rx Address Map Offset 3:** This field is used with the Rx Address Map Size 1, 2 and 3 Registers to determine the translated address for serial data. If the received packet address is less than the sum of the Rx Address Map Size 1, 2 and 3 Registers but greater than or equal to the sum of Rx Address Map Size 1 and 2, the difference between the packet address and Rx Address Map Size 1 and 2 is added to the contents of this register to obtain the translated address.
1:0	r/o	Reserved: Reads return 0, writes have no effect.

Rx Address Map Size 4 Register

The Rx Address Map Size 4 Register is used to identify the intended destination of inbound serial packets. If an inbound packet address satisfies the conditions:

Inbound Address >= Rx Address Size1 + Rx Address Size2 + Rx Address Size3

Inbound Address < Rx Address Size1 + Rx Address Size2 + Rx Address Size3 + Rx Address Size4

the address is translated to:

OPB Address=Inbound Address + Rx Address Offset4 - Rx Address Size1 - Rx Address Size2 - Rx Address Size3

Table 23: **VLYNQ Rx Address Map Size 4 Register (Base Address + 0x38)**

Bits	Type	Description
31:2	r/w	**Rx Address Map Size 4:** This field is used to determine if received packets are destined for the 4th of 4 mapped address regions. The sum of this field and Rx Address Map Size 1, 2 and 3 is compared with the address contained in the received packet. If the received packet address is less than this sum but greater than or equal to the sum of Rx Address Map Size 1, 2 and 3, the packet address, minus Rx Address Map Size 1, 2 and 3, is added to the Rx Address Map Offset 4 Register to obtain the translated address.
1:0	r/o	Reserved: Reads return 0, writes have no effect.

Rx Address Map Offset 4 Register

The Rx Address Map Offset 4 Register is used with the Rx Address Map Size 4 Register to translate receive packet addresses to local device OPB addresses.

Table 24: **VLYNQ Rx Address Map Offset 4 Register (Base Address + 0x3C)**

Bits	Type	Description
31:2	r/w	**Rx Address Map Offset 4:** This field is used with the Rx Address Map Size 1, 2, 3 and 4 Registers to determine the translated address for serial data. If the received packet address is less than the sum of the Rx Address Map Size 1, 2, 3 and 4 Registers but greater than or equal to the sum of Rx Address Map Size 1, 2 and 3, the difference between the packet address and Rx Address Map Size 1, 2 and 3 is added to the contents of this register to obtain the translated address.
1:0	r/o	Reserved: Reads return 0, writes have no effect.

Chip Version Register

The Chip Version Register reflects the value on the `DEVICE_ID` and `DEVICE_REV` core inputs. This register provides a mechanism for software to determine the type and version of VLYNQ devices. The device must be registered with the VLYNQ IP group to be assigned a unique ID to distinguish from other VLYNQ devices.

Table 25: **VLYNQ Chip Version Register (Base Address + 0x40)**

Bits	Type	Description
31:16	r/o	**Device Revision:** This field reflects the value on the `DEVICE_REV` input.
15:0	r/o	**Device ID:** This field reflects the value on the `DEVICE_ID` input. The device must be registered with the VLYNQ IP Group to be assigned an ID.

Auto Negotiation Register

The Auto Negotiation Register reflects the ability of the VLYNQ core to communicate with the remote VLYNQ device after reset.

Table 26: **VLYNQ Auto Negotiation Register (Base Address + 0x44)**

Bits	Type	Description
31:17	r/o	Reserved: Reads return 0, writes have no effect.
16	r/o	**Version 2.X Mode**: A value of '1' indicates that the VLYNQ core communicates with the remote VLYNQ device using the Version 2.X protocol. A value of '0' indicates a link was established with a Version 1.X VLYNQ.
15:0	r/o	Reserved: Reads return 0, writes have no effect.

Negotiation Status Register

The Negotiation Status Register reflects the current abilities communicated between the local and remote VLYNQ.

Table 27: **VLYNQ Negotiation Status Register (Base Address + 0x4C)**

Bits	Type	Description
31:1	r/o	Reserved: Reads return 0, writes have no effect.
0	r/o	**Mode Mask**: A value of '1' in a particular bit position indicates the core supports the mode corresponding to a particular bit number. This core always returns 0x0001 when read, indicating it is a mode 0 device.

Interrupt Vector 3-0 Register

The Interrupt Vector 3-0 Register enables and maps interrupts sourced from the core `VLYNQ_INT[3:0]` inputs.

Table 28: **VLYNQ Interrupt Vector 3-0 Register (Base Address + 0x60)**

Bits	Type	Description
31	r/w	**Interrupt Enable 3**: When set, indicates that interrupts detected on the `VLYNQ_INT[3]` input should be written to the Interrupt Pending/Set Register, which subsequently generates an interrupt depending on the status of the Interrupt_Local bit in the Control Register.
30	r/w	**Interrupt Type 3**: When set, indicates that the `VLYNQ_INT[3]` interrupt is pulsed. When clear, it indicates it is level sensitive.
29	r/w	**Interrupt Polarity 3**: When set, indicates that the `VLYNQ_INT[3]` interrupt is active low. When clear, it indicates it is active high.
28:24	r/w	**Interrupt Vector 3**: This field maps the `VLYNQ_INT[3]` input to the corresponding bit position of the Interrupt Pending/Set Register.
23	r/w	**Interrupt Enable 2**: When set, indicates that interrupts detected on the `VLYNQ_INT[2]` input should be written to the Interrupt Pending/Set Register, which subsequently generates an interrupt depending on the status of the Interrupt_Local bit in the Control Register.
22	r/w	**Interrupt Type 2**: When set, indicates that the `VLYNQ_INT[2]` interrupt is pulsed. When clear, it indicates it is level sensitive.
21	r/w	**Interrupt Polarity 2**: When set, indicates that the `VLYNQ_INT[2]` interrupt is active low. When clear, it indicates it is active high.

Table 28: **VLYNQ Interrupt Vector 3-0 Register (Base Address + 0x60) *(Continued)***

Bits	Type	Description
20:16	r/w	**Interrupt Vector 2**: This field maps the VLYNQ_INT[2] input to the corresponding bit position of the Interrupt Pending/Set Register.
15	r/w	**Interrupt Enable 1**: When set, indicates that interrupts detected on the VLYNQ_INT[1] input should be written to the Interrupt Pending/Set Register, which subsequently generates an interrupt depending on the status of the Interrupt_Local bit in the Control Register.
14	r/w	**Interrupt Type 1**: When set, indicates that the VLYNQ_INT[1] interrupt is pulsed. When clear, it indicates it is level sensitive.
13	r/w	**Interrupt Polarity 1**: When set, indicates that the VLYNQ_INT[1] interrupt is active low. When clear, it indicates it is active high.
12:8	r/w	**Interrupt Vector 1**: This field maps the VLYNQ_INT[1] input to the corresponding bit position of the Interrupt Pending/Set Register.
7	r/w	**Interrupt Enable 0**: When set, indicates that interrupts detected on the VLYNQ_INT[0] input should be written to the Interrupt Pending/Set Register, which subsequently generates an interrupt depending on the status of the Interrupt_Local bit in the Control Register.
6	r/w	**Interrupt Type 0**: When set, indicates that the VLYNQ_INT[0] interrupt is pulsed. When clear, it indicates it is level sensitive.
5	r/w	**Interrupt Polarity 0**: When set, indicates that the VLYNQ_INT[0] interrupt is active low. When clear, it indicates it is active high.
4:0	r/w	**Interrupt Vector 0**: This field maps the VLYNQ_INT[0] input to the corresponding bit position of the Interrupt Pending/Set Register.

Interrupt Vector 7-4 Register

The Interrupt Vector 7-4 Register enables and maps interrupts sourced from the core VLYNQ_INT[7:4]inputs.

Table 29: **VLYNQ Interrupt Vector 7-4 Register (Base Address + 0x64)**

Bits	Type	Description
31	r/w	**Interrupt Enable 7**: When set, indicates that interrupts detected on the VLYNQ_INT[7] input should be written to the Interrupt Pending/Set Register, which subsequently generates an interrupt depending on the status of the Interrupt_Local bit in the Control Register.
30	r/w	**Interrupt Type 7**: When set, indicates that the VLYNQ_INT[7] interrupt is pulsed. When clear, it indicates it is level sensitive.
29	r/w	**Interrupt Polarity 7**: When set, indicates that the VLYNQ_INT[7] interrupt is active low. When clear, it indicates it is active high.
28:24	r/w	**Interrupt Vector 7**: This field maps the VLYNQ_INT[7] input to the corresponding bit position of the Interrupt Pending/Set Register.
23	r/w	**Interrupt Enable 6**: When set, indicates that interrupts detected on the VLYNQ_INT[6] input should be written to the Interrupt Pending/Set Register, which subsequently generates an interrupt depending on the status of the Interrupt_Local bit in the Control Register.
22	r/w	**Interrupt Type 6**: When set, indicates that the VLYNQ_INT[6] interrupt is pulsed. When clear, it indicates it is level sensitive.
21	r/w	**Interrupt Polarity 6**: When set, indicates that the VLYNQ_INT[6] interrupt is active low. When clear, it indicates it is active high.

Table 29: **VLYNQ Interrupt Vector 7-4 Register (Base Address + 0x64) *(Continued)***

Bits	Type	Description
20:16	r/w	**Interrupt Vector 6**: This field maps the VLYNQ_INT[6] input to the corresponding bit position of the Interrupt Pending/Set Register.
15	r/w	**Interrupt Enable 5**: When set, indicates that interrupts detected on the VLYNQ_INT[5] input should be written to the Interrupt Pending/Set Register, which subsequently generates an interrupt depending on the status of the Interrupt_Local bit in the Control Register.
14	r/w	**Interrupt Type 5**: When set, indicates that the VLYNQ_INT[5] interrupt is pulsed. When clear, it indicates it is level sensitive.
13	r/w	**Interrupt Polarity 5**: When set, indicates that the VLYNQ_INT[5] interrupt is active low. When clear, it indicates it is active high.
12:8	r/w	**Interrupt Vector 5**: This field maps the VLYNQ_INT[5] input to the corresponding bit position of the Interrupt Pending/Set Register.
7	r/w	**Interrupt Enable 4**: When set, indicates that interrupts detected on the VLYNQ_INT[4] input should be written to the Interrupt Pending/Set Register, which subsequently generates an interrupt depending on the status of the Interrupt_Local bit in the Control Register.
6	r/w	**Interrupt Type 4**: When set, indicates that the VLYNQ_INT[4] interrupt is pulsed. When clear, it indicates it is level-sensitive.
5	r/w	**Interrupt Polarity 4**: When set, indicates that the VLYNQ_INT[4] interrupt is active low. When clear, it indicates it is active high.
4:0	r/w	**Interrupt Vector 4**: This field maps the VLYNQ_INT[4] input to the corresponding bit position of the Interrupt Pending/Set Register.

Remote Configuration Registers

The Remote Configuration Registers (Base Address 0x80-0xfc) are the same registers described previously but for the remote VLYNQ device. Configuration accesses are used to access these registers and do not require configuration of the address translation registers.

Related Information

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.

Ordering Information

This LogiCORE IP product is provided under the SignOnce IP Site License. A free evaluation version of the VLYNQ module is available; please visit the VLYNQ product page or visit Xilinx IP Center for more information about purchasing this and other Xilinx IP products.

Revision History

Date	Version	Revision
2/28/06	1.0	Early access release.
7/13/06	1.1	Initial Xilinx release.
9/21/06	1.2	Updated core to version 1.2; added Spartan-3A FPGA to device support.
3/24/08	1.3	Updated core to version 1.3; Xilinx tools 10.1.
04/24/09	1.4	Updated core to version 1.4; Xilinx tools 11.1.
The product was discontinued as of October 31, 2009.		

Notice of Disclaimer