**Jae Lee**

**Derek McQuay**

**ECEn 427-001**

**Dr. Hutchings**

**Lab 6 PIT**

**Register Descriptions**

All PIT registers are accessed through the AXI interface. The base address for these registers is provided by the configuration parameter, XPAR\_PIT\_0\_BASEADDR. Each register is 32 bits although some bits may be unused and is accessed on a 4-byte boundary offset from the base address.

Because AXI addresses are byte addresses, PIT register offsets are located at integral multiples of four from the base address. The table below illustrates the registers and their offsets from the base address.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Base Address + Offset(hex) | Register Name | Access Type | Default Value(hex) | Description |
| XPAR\_PIT\_0\_BASEADDR + 0x0 | SLV\_REG0 | Read / Write | 0x0 | PIT control register |
| XPAR\_PIT\_0\_BASEADDR + 0x4 | SLV\_REG1 | Read / Write | 0xFFFFFF | Delay-value register |

**PIT control register**

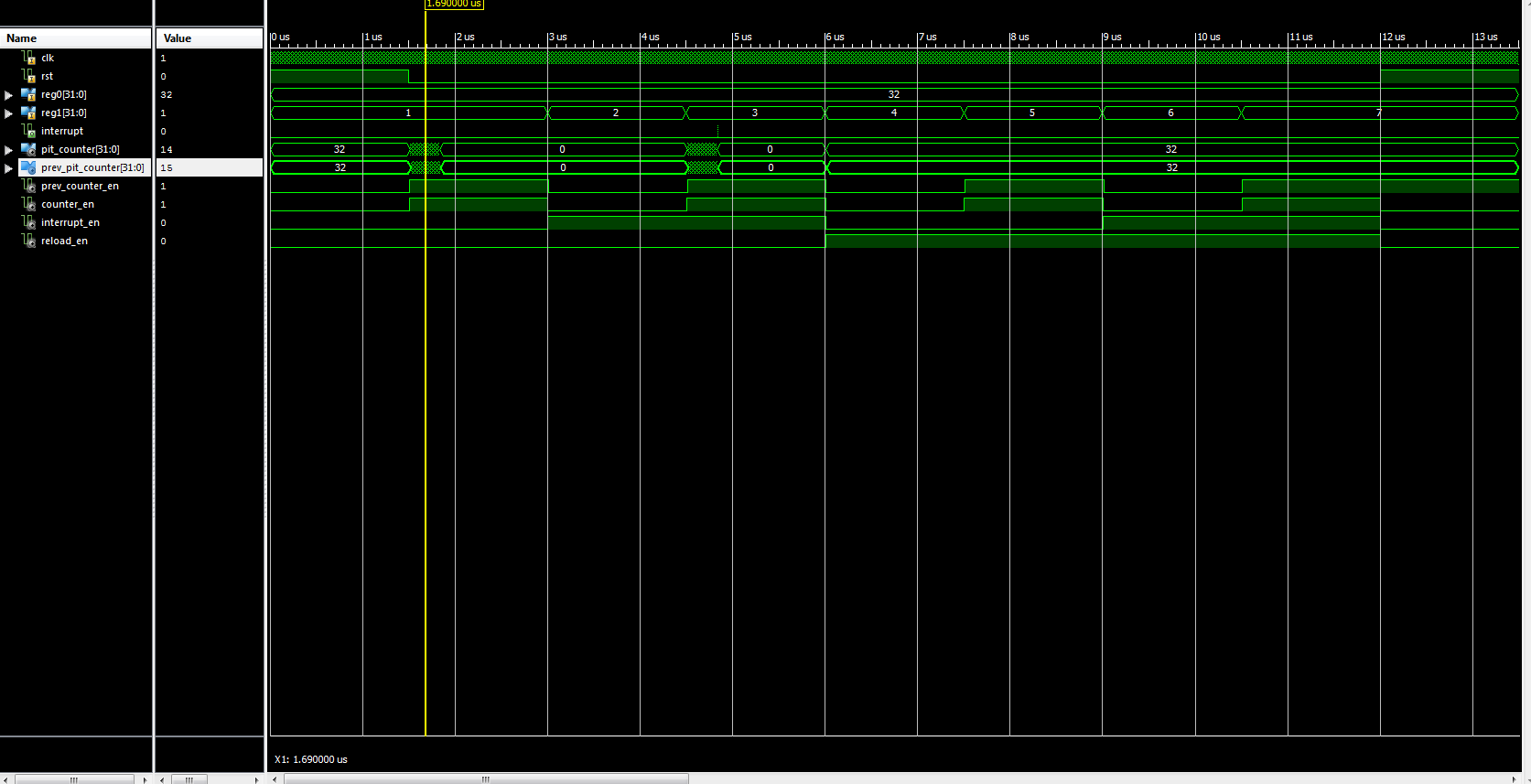
This is a 32-bit register, but only bits 0-2 are used. The function of each bit is described below.

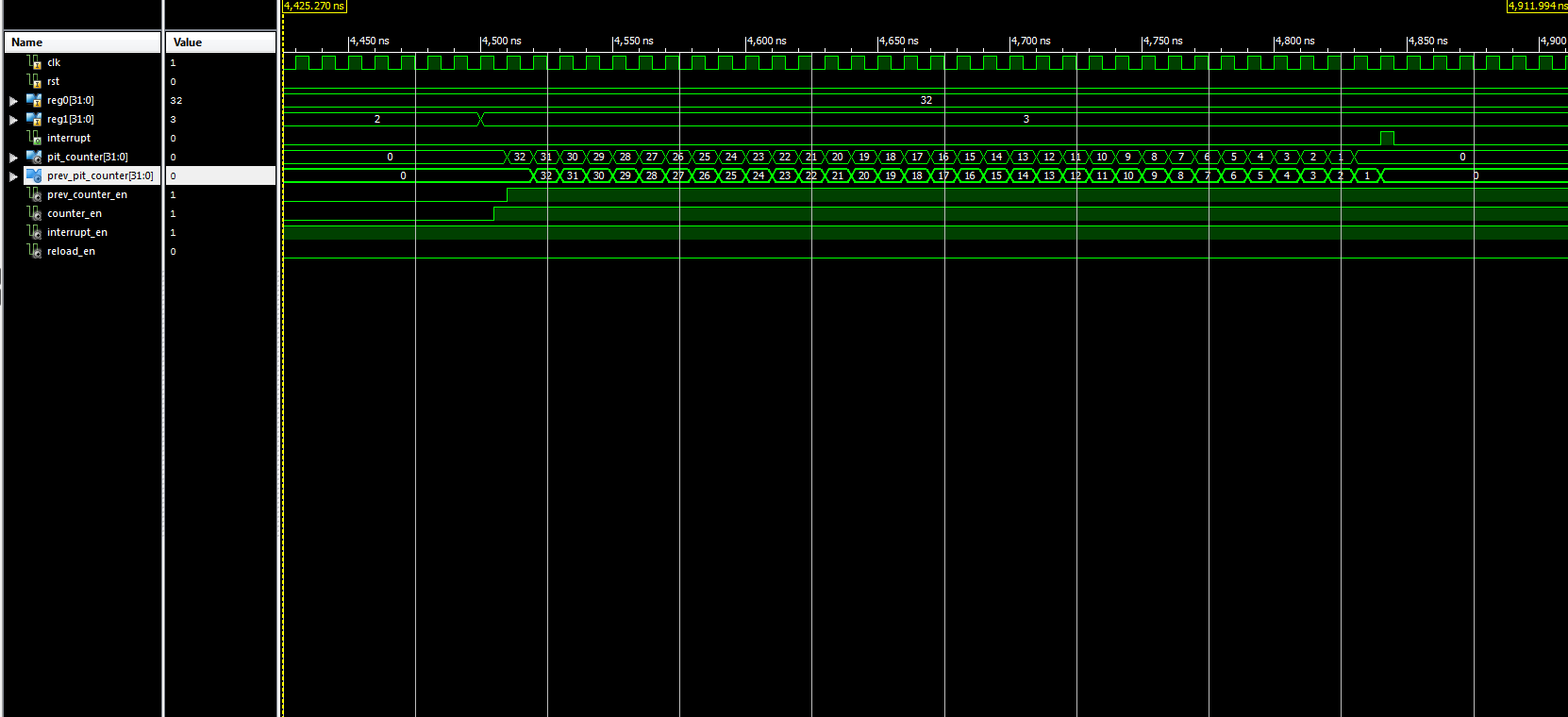
* bit 0: allows the counter to decrement if set to '1', holds the counter at its current value if set to a '0'.
* bit 1: enables interrupts if set to a '1', disables interrupts if set to a '0'.
* bit 2: if set to '1', the timer-counter reloads with the contents of the delay-value register when it reaches 0. No reload is performed if the bit is '0'.

**Delay-value register**

The value in this register will be reloaded onto the counter when ‘reload bit’ is set.

**Timing Diagram**





**Driver API**

/\*\*

\*

\* Write/Read 32 bit value to/from BLINK user logic slave registers.

\*

\* @param BaseAddress is the base address of the BLINK device.

\* @param RegOffset is the offset from the slave register to write to or read from.

\* @param Value is the data written to the register.

\*

\* @return Data is the data from the user logic slave register.

\*

\* @note

\* C-style signature:

\* void BLINK\_mWriteSlaveRegn(Xuint32 BaseAddress, unsigned RegOffset, Xuint32 Value)

\* Xuint32 BLINK\_mReadSlaveRegn(Xuint32 BaseAddress, unsigned RegOffset)

\*

\*/

#define BLINK\_mWriteSlaveReg0(BaseAddress, RegOffset, Value) \

Xil\_Out32((BaseAddress) + (BLINK\_SLV\_REG0\_OFFSET) + (RegOffset), (Xuint32)(Value))

#define BLINK\_mWriteSlaveReg1(BaseAddress, RegOffset, Value) \

Xil\_Out32((BaseAddress) + (BLINK\_SLV\_REG1\_OFFSET) + (RegOffset), (Xuint32)(Value))

#define BLINK\_mReadSlaveReg0(BaseAddress, RegOffset) \

Xil\_In32((BaseAddress) + (BLINK\_SLV\_REG0\_OFFSET) + (RegOffset))

#define BLINK\_mReadSlaveReg1(BaseAddress, RegOffset) \

Xil\_In32((BaseAddress) + (BLINK\_SLV\_REG1\_OFFSET) + (RegOffset))

**Bug Report**

We encountered a lot of bugs while working on this lab. Here is a report of the bugs that we found:

* We had a strange up that when we initialized our PIT to a certain value, it would draw another block of aliens. We would then have two alien blocks. We just had to make sure that it didn’t overwrite the value that contained the starting position of the aliens and it fixed the problem
* When we tried to read in values for the UART we got this weird error that would make our buttons stop working. We just had to be smart about where we decided to read in those values and it fixed the issue
* PIT interrupt didn’t work because we didn’t connect the output of the PIT to the interrupt controller. This took a really long time because we had to wait for the hardware to build again.
* The red ufo would stop flying in the middle of its flight. This was due to reading values from the UART. We just had to make sure that it didn’t overwrite certain values and we made sure that it safely completed its flight all the way across the screen.
* Space invaders, the game itself, is very bug prone
* We accidently flipped the priority of the PIT controller register. This caused reloading the value of the delay value and enabled the counter. It really messed up our PIT. Once we figured out what the bug was, we were able to figured out what caused the problem and made it work out perfectly
* Our reset value was set for too high to trigger the first interrupt. This caused it to wait a really long time for that first interrupt to happen. This made the game just sit there waiting for that to happen. After a while the game would start and work normally.
* Our GPIO block stopped responding. We don’t know exactly what solved this problem because we changed multiple things at once and it solved the problem but it was a pretty annoying bug.

**Code**

Turn in the VHDL code for your PIT and the driver code. I don't need your Space Invaders or Real Time Clock (Lab 2) code.

Driver Code

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Filename: C:\Users\superman\Desktop\korea\SpaceInvaderTestHW2/drivers/blink\_v1\_00\_a/src/blink.h  \* Version: 1.00.a  \* Description: blink Driver Header File  \* Date: Tue Nov 04 17:15:42 2014 (by Create and Import Peripheral Wizard)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #ifndef BLINK\_H  #define BLINK\_H  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Include Files \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "xbasic\_types.h"  #include "xstatus.h"  #include "xil\_io.h"  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Constant Definitions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*  \* User Logic Slave Space Offsets  \* -- SLV\_REG0 : user logic slave module register 0  \* -- SLV\_REG1 : user logic slave module register 1  \*/  #define BLINK\_USER\_SLV\_SPACE\_OFFSET (0x00000000)  #define BLINK\_SLV\_REG0\_OFFSET (BLINK\_USER\_SLV\_SPACE\_OFFSET + 0x00000000)  #define BLINK\_SLV\_REG1\_OFFSET (BLINK\_USER\_SLV\_SPACE\_OFFSET + 0x00000004)  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Type Definitions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Macros (Inline Functions) Definitions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*  \*  \* Write a value to a BLINK register. A 32 bit write is performed.  \* If the component is implemented in a smaller width, only the least  \* significant data is written.  \*  \* @param BaseAddress is the base address of the BLINK device.  \* @param RegOffset is the register offset from the base to write to.  \* @param Data is the data written to the register.  \*  \* @return None.  \*  \* @note  \* C-style signature:  \* void BLINK\_mWriteReg(Xuint32 BaseAddress, unsigned RegOffset, Xuint32 Data)  \*  \*/  #define BLINK\_mWriteReg(BaseAddress, RegOffset, Data) \  Xil\_Out32((BaseAddress) + (RegOffset), (Xuint32)(Data))  /\*\*  \*  \* Read a value from a BLINK register. A 32 bit read is performed.  \* If the component is implemented in a smaller width, only the least  \* significant data is read from the register. The most significant data  \* will be read as 0.  \*  \* @param BaseAddress is the base address of the BLINK device.  \* @param RegOffset is the register offset from the base to write to.  \*  \* @return Data is the data from the register.  \*  \* @note  \* C-style signature:  \* Xuint32 BLINK\_mReadReg(Xuint32 BaseAddress, unsigned RegOffset)  \*  \*/  #define BLINK\_mReadReg(BaseAddress, RegOffset) \  Xil\_In32((BaseAddress) + (RegOffset))  /\*\*  \*  \* Write/Read 32 bit value to/from BLINK user logic slave registers.  \*  \* @param BaseAddress is the base address of the BLINK device.  \* @param RegOffset is the offset from the slave register to write to or read from.  \* @param Value is the data written to the register.  \*  \* @return Data is the data from the user logic slave register.  \*  \* @note  \* C-style signature:  \* void BLINK\_mWriteSlaveRegn(Xuint32 BaseAddress, unsigned RegOffset, Xuint32 Value)  \* Xuint32 BLINK\_mReadSlaveRegn(Xuint32 BaseAddress, unsigned RegOffset)  \*  \*/  #define BLINK\_mWriteSlaveReg0(BaseAddress, RegOffset, Value) \  Xil\_Out32((BaseAddress) + (BLINK\_SLV\_REG0\_OFFSET) + (RegOffset), (Xuint32)(Value))  #define BLINK\_mWriteSlaveReg1(BaseAddress, RegOffset, Value) \  Xil\_Out32((BaseAddress) + (BLINK\_SLV\_REG1\_OFFSET) + (RegOffset), (Xuint32)(Value))  #define BLINK\_mReadSlaveReg0(BaseAddress, RegOffset) \  Xil\_In32((BaseAddress) + (BLINK\_SLV\_REG0\_OFFSET) + (RegOffset))  #define BLINK\_mReadSlaveReg1(BaseAddress, RegOffset) \  Xil\_In32((BaseAddress) + (BLINK\_SLV\_REG1\_OFFSET) + (RegOffset))  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function Prototypes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*  \*  \* Run a self-test on the driver/device. Note this may be a destructive test if  \* resets of the device are performed.  \*  \* If the hardware system is not built correctly, this function may never  \* return to the caller.  \*  \* @param baseaddr\_p is the base address of the BLINK instance to be worked on.  \*  \* @return  \*  \* - XST\_SUCCESS if all self-test code passed  \* - XST\_FAILURE if any self-test code failed  \*  \* @note Caching must be turned off for this function to work.  \* @note Self test may fail if data memory and device are not on the same bus.  \*  \*/  XStatus BLINK\_SelfTest(void \* baseaddr\_p);  /\*\*  \* Defines the number of registers available for read and write\*/  #define TEST\_AXI\_LITE\_USER\_NUM\_REG 2  #endif /\*\* BLINK\_H \*/ |

VHDL code

|  |
| --- |
| ------------------------------------------------------------------------------  -- user\_logic.vhd - entity/architecture pair  ------------------------------------------------------------------------------  --  -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  -- \*\* Copyright (c) 1995-2011 Xilinx, Inc. All rights reserved. \*\*  -- \*\* \*\*  -- \*\* Xilinx, Inc. \*\*  -- \*\* XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" \*\*  -- \*\* AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND \*\*  -- \*\* SOLUTIONS FOR XILINX DEVICES. BY PROVIDING THIS DESIGN, CODE, \*\*  -- \*\* OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, \*\*  -- \*\* APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION \*\*  -- \*\* THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT, \*\*  -- \*\* AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE \*\*  -- \*\* FOR YOUR IMPLEMENTATION. XILINX EXPRESSLY DISCLAIMS ANY \*\*  -- \*\* WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE \*\*  -- \*\* IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR \*\*  -- \*\* REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF \*\*  -- \*\* INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS \*\*  -- \*\* FOR A PARTICULAR PURPOSE. \*\*  -- \*\* \*\*  -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  --  ------------------------------------------------------------------------------  -- Filename: user\_logic.vhd  -- Version: 1.00.a  -- Description: User logic.  -- Date: Tue Nov 04 17:15:41 2014 (by Create and Import Peripheral Wizard)  -- VHDL Standard: VHDL'93  ------------------------------------------------------------------------------  -- Naming Conventions:  -- active low signals: "\*\_n"  -- clock signals: "clk", "clk\_div#", "clk\_#x"  -- reset signals: "rst", "rst\_n"  -- generics: "C\_\*"  -- user defined types: "\*\_TYPE"  -- state machine next state: "\*\_ns"  -- state machine current state: "\*\_cs"  -- combinatorial signals: "\*\_com"  -- pipelined or register delay signals: "\*\_d#"  -- counter signals: "\*cnt\*"  -- clock enable signals: "\*\_ce"  -- internal version of output port: "\*\_i"  -- device pins: "\*\_pin"  -- ports: "- Names begin with Uppercase"  -- processes: "\*\_PROCESS"  -- component instantiations: "<ENTITY\_>I\_<#|FUNC>"  ------------------------------------------------------------------------------  -- DO NOT EDIT BELOW THIS LINE --------------------  library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_arith.all;  use ieee.std\_logic\_unsigned.all;  library proc\_common\_v3\_00\_a;  use proc\_common\_v3\_00\_a.proc\_common\_pkg.all;  -- DO NOT EDIT ABOVE THIS LINE --------------------  --USER libraries added here  ------------------------------------------------------------------------------  -- Entity section  ------------------------------------------------------------------------------  -- Definition of Generics:  -- C\_NUM\_REG -- Number of software accessible registers  -- C\_SLV\_DWIDTH -- Slave interface data bus width  --  -- Definition of Ports:  -- Bus2IP\_Clk -- Bus to IP clock  -- Bus2IP\_Resetn -- Bus to IP reset  -- Bus2IP\_Data -- Bus to IP data bus  -- Bus2IP\_BE -- Bus to IP byte enables  -- Bus2IP\_RdCE -- Bus to IP read chip enable  -- Bus2IP\_WrCE -- Bus to IP write chip enable  -- IP2Bus\_Data -- IP to Bus data bus  -- IP2Bus\_RdAck -- IP to Bus read transfer acknowledgement  -- IP2Bus\_WrAck -- IP to Bus write transfer acknowledgement  -- IP2Bus\_Error -- IP to Bus error response  ------------------------------------------------------------------------------  entity user\_logic is  generic  (  -- ADD USER GENERICS BELOW THIS LINE ---------------  --USER generics added here  -- ADD USER GENERICS ABOVE THIS LINE ---------------  -- DO NOT EDIT BELOW THIS LINE ---------------------  -- Bus protocol parameters, do not add to or delete  C\_NUM\_REG : integer := 2;  C\_SLV\_DWIDTH : integer := 32  -- DO NOT EDIT ABOVE THIS LINE ---------------------  );  port  (  -- ADD USER PORTS BELOW THIS LINE ------------------  --USER ports added here  -- ADD USER PORTS ABOVE THIS LINE ------------------  interrupt : out std\_logic;  -- DO NOT EDIT BELOW THIS LINE ---------------------  -- Bus protocol ports, do not add to or delete  Bus2IP\_Clk : in std\_logic;  Bus2IP\_Resetn : in std\_logic;  Bus2IP\_Data : in std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);  Bus2IP\_BE : in std\_logic\_vector(C\_SLV\_DWIDTH/8-1 downto 0);  Bus2IP\_RdCE : in std\_logic\_vector(C\_NUM\_REG-1 downto 0);  Bus2IP\_WrCE : in std\_logic\_vector(C\_NUM\_REG-1 downto 0);  IP2Bus\_Data : out std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);  IP2Bus\_RdAck : out std\_logic;  IP2Bus\_WrAck : out std\_logic;  IP2Bus\_Error : out std\_logic  -- DO NOT EDIT ABOVE THIS LINE ---------------------  );  attribute MAX\_FANOUT : string;  attribute SIGIS : string;  attribute SIGIS of Bus2IP\_Clk : signal is "CLK";  attribute SIGIS of Bus2IP\_Resetn : signal is "RST";  end entity user\_logic;  ------------------------------------------------------------------------------  -- Architecture section  ------------------------------------------------------------------------------  architecture IMP of user\_logic is  --USER signal declarations added here, as needed for user logic  signal count : unsigned(31 downto 0) := "00000000000000001111111111111111";  signal interrupt\_flag : std\_logic := '0';  ------------------------------------------  -- Signals for user logic slave model s/w accessible register example  ------------------------------------------  signal slv\_reg0 : std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);  signal slv\_reg1 : std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);  signal slv\_reg\_write\_sel : std\_logic\_vector(1 downto 0);  signal slv\_reg\_read\_sel : std\_logic\_vector(1 downto 0);  signal slv\_ip2bus\_data : std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);  signal slv\_read\_ack : std\_logic;  signal slv\_write\_ack : std\_logic;  begin  --USER logic implementation added here  ------------------------------------------  -- Example code to read/write user logic slave model s/w accessible registers  --  -- Note:  -- The example code presented here is to show you one way of reading/writing  -- software accessible registers implemented in the user logic slave model.  -- Each bit of the Bus2IP\_WrCE/Bus2IP\_RdCE signals is configured to correspond  -- to one software accessible register by the top level template. For example,  -- if you have four 32 bit software accessible registers in the user logic,  -- you are basically operating on the following memory mapped registers:  --  -- Bus2IP\_WrCE/Bus2IP\_RdCE Memory Mapped Register  -- "1000" C\_BASEADDR + 0x0  -- "0100" C\_BASEADDR + 0x4  -- "0010" C\_BASEADDR + 0x8  -- "0001" C\_BASEADDR + 0xC  --  ------------------------------------------  slv\_reg\_write\_sel <= Bus2IP\_WrCE(1 downto 0);  slv\_reg\_read\_sel <= Bus2IP\_RdCE(1 downto 0);  slv\_write\_ack <= Bus2IP\_WrCE(0) or Bus2IP\_WrCE(1);  slv\_read\_ack <= Bus2IP\_RdCE(0) or Bus2IP\_RdCE(1);    -- implement slave model software accessible register(s)  SLAVE\_REG\_WRITE\_PROC : process( Bus2IP\_Clk ) is  begin  if Bus2IP\_Clk'event and Bus2IP\_Clk = '1' then  if Bus2IP\_Resetn = '0' then  slv\_reg0 <= (others => '0');  slv\_reg1 <= (others => '0');  else  case slv\_reg\_write\_sel is  when "10" =>  for byte\_index in 0 to (C\_SLV\_DWIDTH/8)-1 loop  if ( Bus2IP\_BE(byte\_index) = '1' ) then  slv\_reg0(byte\_index\*8+7 downto byte\_index\*8) <= Bus2IP\_Data(byte\_index\*8+7 downto byte\_index\*8);  end if;  end loop;  when "01" =>  for byte\_index in 0 to (C\_SLV\_DWIDTH/8)-1 loop  if ( Bus2IP\_BE(byte\_index) = '1' ) then  slv\_reg1(byte\_index\*8+7 downto byte\_index\*8) <= Bus2IP\_Data(byte\_index\*8+7 downto byte\_index\*8);  end if;  end loop;  when others => null;  end case;  end if;  end if;  end process SLAVE\_REG\_WRITE\_PROC;  -- implement slave model software accessible register(s) read mux  SLAVE\_REG\_READ\_PROC : process( slv\_reg\_read\_sel, slv\_reg0, slv\_reg1 ) is  begin  case slv\_reg\_read\_sel is  when "10" => slv\_ip2bus\_data <= slv\_reg0;  when "01" => slv\_ip2bus\_data <= slv\_reg1;  when others => slv\_ip2bus\_data <= (others => '0');  end case;  end process SLAVE\_REG\_READ\_PROC;  ------------------------------------------  -- Example code to drive IP to Bus signals  ------------------------------------------  IP2Bus\_Data <= slv\_ip2bus\_data when slv\_read\_ack = '1' else  (others => '0');  IP2Bus\_WrAck <= slv\_write\_ack;  IP2Bus\_RdAck <= slv\_read\_ack;  IP2Bus\_Error <= '0';    -- Create counter  -- slv\_reg0(0) value 1 to decrement, 0 to hold  -- slv\_reg0(1) value 1 to enable interrupt, 0 to disable interrupt  -- slv\_reg0(2) value 1 to reload the value of the slv\_reg1, 0 for no reload    -- signal count : unsigned(31 downto 0) := "00000000000000001111111111111111";  counter: process(Bus2IP\_Clk)  begin    if(Bus2IP\_Resetn = '0') then  count <= x"00FFFFFF";  elsif(Bus2IP\_Clk 'event and Bus2IP\_Clk = '1') then  interrupt <= '0';  if(slv\_reg0(2) = '1' and count = 0) then -- reload enable  count <= unsigned(slv\_reg1);  interrupt\_flag <= '0';    end if;    if(slv\_reg0(1) = '1') then -- interrupt enable  if(count = 0 and interrupt\_flag = '0') then  interrupt\_flag <= '1';  interrupt <= '1';  end if;  end if;    if(slv\_reg0(0) = '1' and count /= 0) then -- counter enable  count <= count - 1;  end if;    end if;  end process counter;    end IMP; |
| ------------------------------------------------------------------------------  -- blink.vhd - entity/architecture pair  ------------------------------------------------------------------------------  -- IMPORTANT:  -- DO NOT MODIFY THIS FILE EXCEPT IN THE DESIGNATED SECTIONS.  --  -- SEARCH FOR --USER TO DETERMINE WHERE CHANGES ARE ALLOWED.  --  -- TYPICALLY, THE ONLY ACCEPTABLE CHANGES INVOLVE ADDING NEW  -- PORTS AND GENERICS THAT GET PASSED THROUGH TO THE INSTANTIATION  -- OF THE USER\_LOGIC ENTITY.  ------------------------------------------------------------------------------  --  -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  -- \*\* Copyright (c) 1995-2011 Xilinx, Inc. All rights reserved. \*\*  -- \*\* \*\*  -- \*\* Xilinx, Inc. \*\*  -- \*\* XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" \*\*  -- \*\* AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND \*\*  -- \*\* SOLUTIONS FOR XILINX DEVICES. BY PROVIDING THIS DESIGN, CODE, \*\*  -- \*\* OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, \*\*  -- \*\* APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION \*\*  -- \*\* THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT, \*\*  -- \*\* AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE \*\*  -- \*\* FOR YOUR IMPLEMENTATION. XILINX EXPRESSLY DISCLAIMS ANY \*\*  -- \*\* WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE \*\*  -- \*\* IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR \*\*  -- \*\* REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF \*\*  -- \*\* INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS \*\*  -- \*\* FOR A PARTICULAR PURPOSE. \*\*  -- \*\* \*\*  -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  --  ------------------------------------------------------------------------------  -- Filename: blink.vhd  -- Version: 1.00.a  -- Description: Top level design, instantiates library components and user logic.  -- Date: Tue Nov 04 17:15:41 2014 (by Create and Import Peripheral Wizard)  -- VHDL Standard: VHDL'93  ------------------------------------------------------------------------------  -- Naming Conventions:  -- active low signals: "\*\_n"  -- clock signals: "clk", "clk\_div#", "clk\_#x"  -- reset signals: "rst", "rst\_n"  -- generics: "C\_\*"  -- user defined types: "\*\_TYPE"  -- state machine next state: "\*\_ns"  -- state machine current state: "\*\_cs"  -- combinatorial signals: "\*\_com"  -- pipelined or register delay signals: "\*\_d#"  -- counter signals: "\*cnt\*"  -- clock enable signals: "\*\_ce"  -- internal version of output port: "\*\_i"  -- device pins: "\*\_pin"  -- ports: "- Names begin with Uppercase"  -- processes: "\*\_PROCESS"  -- component instantiations: "<ENTITY\_>I\_<#|FUNC>"  ------------------------------------------------------------------------------  library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_arith.all;  use ieee.std\_logic\_unsigned.all;  library proc\_common\_v3\_00\_a;  use proc\_common\_v3\_00\_a.proc\_common\_pkg.all;  use proc\_common\_v3\_00\_a.ipif\_pkg.all;  library axi\_lite\_ipif\_v1\_01\_a;  use axi\_lite\_ipif\_v1\_01\_a.axi\_lite\_ipif;  library blink\_v1\_00\_a;  use blink\_v1\_00\_a.user\_logic;  ------------------------------------------------------------------------------  -- Entity section  ------------------------------------------------------------------------------  -- Definition of Generics:  -- C\_S\_AXI\_DATA\_WIDTH -- AXI4LITE slave: Data width  -- C\_S\_AXI\_ADDR\_WIDTH -- AXI4LITE slave: Address Width  -- C\_S\_AXI\_MIN\_SIZE -- AXI4LITE slave: Min Size  -- C\_USE\_WSTRB -- AXI4LITE slave: Write Strobe  -- C\_DPHASE\_TIMEOUT -- AXI4LITE slave: Data Phase Timeout  -- C\_BASEADDR -- AXI4LITE slave: base address  -- C\_HIGHADDR -- AXI4LITE slave: high address  -- C\_FAMILY -- FPGA Family  -- C\_NUM\_REG -- Number of software accessible registers  -- C\_NUM\_MEM -- Number of address-ranges  -- C\_SLV\_AWIDTH -- Slave interface address bus width  -- C\_SLV\_DWIDTH -- Slave interface data bus width  --  -- Definition of Ports:  -- S\_AXI\_ACLK -- AXI4LITE slave: Clock  -- S\_AXI\_ARESETN -- AXI4LITE slave: Reset  -- S\_AXI\_AWADDR -- AXI4LITE slave: Write address  -- S\_AXI\_AWVALID -- AXI4LITE slave: Write address valid  -- S\_AXI\_WDATA -- AXI4LITE slave: Write data  -- S\_AXI\_WSTRB -- AXI4LITE slave: Write strobe  -- S\_AXI\_WVALID -- AXI4LITE slave: Write data valid  -- S\_AXI\_BREADY -- AXI4LITE slave: Response ready  -- S\_AXI\_ARADDR -- AXI4LITE slave: Read address  -- S\_AXI\_ARVALID -- AXI4LITE slave: Read address valid  -- S\_AXI\_RREADY -- AXI4LITE slave: Read data ready  -- S\_AXI\_ARREADY -- AXI4LITE slave: read addres ready  -- S\_AXI\_RDATA -- AXI4LITE slave: Read data  -- S\_AXI\_RRESP -- AXI4LITE slave: Read data response  -- S\_AXI\_RVALID -- AXI4LITE slave: Read data valid  -- S\_AXI\_WREADY -- AXI4LITE slave: Write data ready  -- S\_AXI\_BRESP -- AXI4LITE slave: Response  -- S\_AXI\_BVALID -- AXI4LITE slave: Resonse valid  -- S\_AXI\_AWREADY -- AXI4LITE slave: Wrte address ready  ------------------------------------------------------------------------------  entity blink is  generic  (  -- ADD USER GENERICS BELOW THIS LINE ---------------  --USER generics added here  -- ADD USER GENERICS ABOVE THIS LINE ---------------  -- DO NOT EDIT BELOW THIS LINE ---------------------  -- Bus protocol parameters, do not add to or delete  C\_S\_AXI\_DATA\_WIDTH : integer := 32;  C\_S\_AXI\_ADDR\_WIDTH : integer := 32;  C\_S\_AXI\_MIN\_SIZE : std\_logic\_vector := X"000001FF";  C\_USE\_WSTRB : integer := 0;  C\_DPHASE\_TIMEOUT : integer := 8;  C\_BASEADDR : std\_logic\_vector := X"FFFFFFFF";  C\_HIGHADDR : std\_logic\_vector := X"00000000";  C\_FAMILY : string := "virtex6";  C\_NUM\_REG : integer := 1;  C\_NUM\_MEM : integer := 1;  C\_SLV\_AWIDTH : integer := 32;  C\_SLV\_DWIDTH : integer := 32  -- DO NOT EDIT ABOVE THIS LINE ---------------------  );  port  (  -- ADD USER PORTS BELOW THIS LINE ------------------  --USER ports added here  interrupt : out std\_logic;  -- ADD USER PORTS ABOVE THIS LINE ------------------  -- DO NOT EDIT BELOW THIS LINE ---------------------  -- Bus protocol ports, do not add to or delete  S\_AXI\_ACLK : in std\_logic;  S\_AXI\_ARESETN : in std\_logic;  S\_AXI\_AWADDR : in std\_logic\_vector(C\_S\_AXI\_ADDR\_WIDTH-1 downto 0);  S\_AXI\_AWVALID : in std\_logic;  S\_AXI\_WDATA : in std\_logic\_vector(C\_S\_AXI\_DATA\_WIDTH-1 downto 0);  S\_AXI\_WSTRB : in std\_logic\_vector((C\_S\_AXI\_DATA\_WIDTH/8)-1 downto 0);  S\_AXI\_WVALID : in std\_logic;  S\_AXI\_BREADY : in std\_logic;  S\_AXI\_ARADDR : in std\_logic\_vector(C\_S\_AXI\_ADDR\_WIDTH-1 downto 0);  S\_AXI\_ARVALID : in std\_logic;  S\_AXI\_RREADY : in std\_logic;  S\_AXI\_ARREADY : out std\_logic;  S\_AXI\_RDATA : out std\_logic\_vector(C\_S\_AXI\_DATA\_WIDTH-1 downto 0);  S\_AXI\_RRESP : out std\_logic\_vector(1 downto 0);  S\_AXI\_RVALID : out std\_logic;  S\_AXI\_WREADY : out std\_logic;  S\_AXI\_BRESP : out std\_logic\_vector(1 downto 0);  S\_AXI\_BVALID : out std\_logic;  S\_AXI\_AWREADY : out std\_logic  -- DO NOT EDIT ABOVE THIS LINE ---------------------  );  attribute MAX\_FANOUT : string;  attribute SIGIS : string;  attribute MAX\_FANOUT of S\_AXI\_ACLK : signal is "10000";  attribute MAX\_FANOUT of S\_AXI\_ARESETN : signal is "10000";  attribute SIGIS of S\_AXI\_ACLK : signal is "Clk";  attribute SIGIS of S\_AXI\_ARESETN : signal is "Rst";  end entity blink;  ------------------------------------------------------------------------------  -- Architecture section  ------------------------------------------------------------------------------  architecture IMP of blink is  constant USER\_SLV\_DWIDTH : integer := C\_S\_AXI\_DATA\_WIDTH;  constant IPIF\_SLV\_DWIDTH : integer := C\_S\_AXI\_DATA\_WIDTH;  constant ZERO\_ADDR\_PAD : std\_logic\_vector(0 to 31) := (others => '0');  constant USER\_SLV\_BASEADDR : std\_logic\_vector := C\_BASEADDR;  constant USER\_SLV\_HIGHADDR : std\_logic\_vector := C\_HIGHADDR;  constant IPIF\_ARD\_ADDR\_RANGE\_ARRAY : SLV64\_ARRAY\_TYPE :=  (  ZERO\_ADDR\_PAD & USER\_SLV\_BASEADDR, -- user logic slave space base address  ZERO\_ADDR\_PAD & USER\_SLV\_HIGHADDR -- user logic slave space high address  );  constant USER\_SLV\_NUM\_REG : integer := 2;  constant USER\_NUM\_REG : integer := USER\_SLV\_NUM\_REG;  constant TOTAL\_IPIF\_CE : integer := USER\_NUM\_REG;  constant IPIF\_ARD\_NUM\_CE\_ARRAY : INTEGER\_ARRAY\_TYPE :=  (  0 => (USER\_SLV\_NUM\_REG) -- number of ce for user logic slave space  );  ------------------------------------------  -- Index for CS/CE  ------------------------------------------  constant USER\_SLV\_CS\_INDEX : integer := 0;  constant USER\_SLV\_CE\_INDEX : integer := calc\_start\_ce\_index(IPIF\_ARD\_NUM\_CE\_ARRAY, USER\_SLV\_CS\_INDEX);  constant USER\_CE\_INDEX : integer := USER\_SLV\_CE\_INDEX;  ------------------------------------------  -- IP Interconnect (IPIC) signal declarations  ------------------------------------------  signal ipif\_Bus2IP\_Clk : std\_logic;  signal ipif\_Bus2IP\_Resetn : std\_logic;  signal ipif\_Bus2IP\_Addr : std\_logic\_vector(C\_S\_AXI\_ADDR\_WIDTH-1 downto 0);  signal ipif\_Bus2IP\_RNW : std\_logic;  signal ipif\_Bus2IP\_BE : std\_logic\_vector(IPIF\_SLV\_DWIDTH/8-1 downto 0);  signal ipif\_Bus2IP\_CS : std\_logic\_vector((IPIF\_ARD\_ADDR\_RANGE\_ARRAY'LENGTH)/2-1 downto 0);  signal ipif\_Bus2IP\_RdCE : std\_logic\_vector(calc\_num\_ce(IPIF\_ARD\_NUM\_CE\_ARRAY)-1 downto 0);  signal ipif\_Bus2IP\_WrCE : std\_logic\_vector(calc\_num\_ce(IPIF\_ARD\_NUM\_CE\_ARRAY)-1 downto 0);  signal ipif\_Bus2IP\_Data : std\_logic\_vector(IPIF\_SLV\_DWIDTH-1 downto 0);  signal ipif\_IP2Bus\_WrAck : std\_logic;  signal ipif\_IP2Bus\_RdAck : std\_logic;  signal ipif\_IP2Bus\_Error : std\_logic;  signal ipif\_IP2Bus\_Data : std\_logic\_vector(IPIF\_SLV\_DWIDTH-1 downto 0);  signal user\_Bus2IP\_RdCE : std\_logic\_vector(USER\_NUM\_REG-1 downto 0);  signal user\_Bus2IP\_WrCE : std\_logic\_vector(USER\_NUM\_REG-1 downto 0);  signal user\_IP2Bus\_Data : std\_logic\_vector(USER\_SLV\_DWIDTH-1 downto 0);  signal user\_IP2Bus\_RdAck : std\_logic;  signal user\_IP2Bus\_WrAck : std\_logic;  signal user\_IP2Bus\_Error : std\_logic;  begin  ------------------------------------------  -- instantiate axi\_lite\_ipif  ------------------------------------------  AXI\_LITE\_IPIF\_I : entity axi\_lite\_ipif\_v1\_01\_a.axi\_lite\_ipif  generic map  (  C\_S\_AXI\_DATA\_WIDTH => IPIF\_SLV\_DWIDTH,  C\_S\_AXI\_ADDR\_WIDTH => C\_S\_AXI\_ADDR\_WIDTH,  C\_S\_AXI\_MIN\_SIZE => C\_S\_AXI\_MIN\_SIZE,  C\_USE\_WSTRB => C\_USE\_WSTRB,  C\_DPHASE\_TIMEOUT => C\_DPHASE\_TIMEOUT,  C\_ARD\_ADDR\_RANGE\_ARRAY => IPIF\_ARD\_ADDR\_RANGE\_ARRAY,  C\_ARD\_NUM\_CE\_ARRAY => IPIF\_ARD\_NUM\_CE\_ARRAY,  C\_FAMILY => C\_FAMILY  )  port map  (  S\_AXI\_ACLK => S\_AXI\_ACLK,  S\_AXI\_ARESETN => S\_AXI\_ARESETN,  S\_AXI\_AWADDR => S\_AXI\_AWADDR,  S\_AXI\_AWVALID => S\_AXI\_AWVALID,  S\_AXI\_WDATA => S\_AXI\_WDATA,  S\_AXI\_WSTRB => S\_AXI\_WSTRB,  S\_AXI\_WVALID => S\_AXI\_WVALID,  S\_AXI\_BREADY => S\_AXI\_BREADY,  S\_AXI\_ARADDR => S\_AXI\_ARADDR,  S\_AXI\_ARVALID => S\_AXI\_ARVALID,  S\_AXI\_RREADY => S\_AXI\_RREADY,  S\_AXI\_ARREADY => S\_AXI\_ARREADY,  S\_AXI\_RDATA => S\_AXI\_RDATA,  S\_AXI\_RRESP => S\_AXI\_RRESP,  S\_AXI\_RVALID => S\_AXI\_RVALID,  S\_AXI\_WREADY => S\_AXI\_WREADY,  S\_AXI\_BRESP => S\_AXI\_BRESP,  S\_AXI\_BVALID => S\_AXI\_BVALID,  S\_AXI\_AWREADY => S\_AXI\_AWREADY,  Bus2IP\_Clk => ipif\_Bus2IP\_Clk,  Bus2IP\_Resetn => ipif\_Bus2IP\_Resetn,  Bus2IP\_Addr => ipif\_Bus2IP\_Addr,  Bus2IP\_RNW => ipif\_Bus2IP\_RNW,  Bus2IP\_BE => ipif\_Bus2IP\_BE,  Bus2IP\_CS => ipif\_Bus2IP\_CS,  Bus2IP\_RdCE => ipif\_Bus2IP\_RdCE,  Bus2IP\_WrCE => ipif\_Bus2IP\_WrCE,  Bus2IP\_Data => ipif\_Bus2IP\_Data,  IP2Bus\_WrAck => ipif\_IP2Bus\_WrAck,  IP2Bus\_RdAck => ipif\_IP2Bus\_RdAck,  IP2Bus\_Error => ipif\_IP2Bus\_Error,  IP2Bus\_Data => ipif\_IP2Bus\_Data  );  ------------------------------------------  -- instantiate User Logic  ------------------------------------------  USER\_LOGIC\_I : entity blink\_v1\_00\_a.user\_logic  generic map  (  -- MAP USER GENERICS BELOW THIS LINE ---------------  --USER generics mapped here  -- MAP USER GENERICS ABOVE THIS LINE ---------------  C\_NUM\_REG => USER\_NUM\_REG,  C\_SLV\_DWIDTH => USER\_SLV\_DWIDTH  )  port map  (  -- MAP USER PORTS BELOW THIS LINE ------------------  --USER ports mapped here  interrupt => interrupt,  -- MAP USER PORTS ABOVE THIS LINE ------------------  Bus2IP\_Clk => ipif\_Bus2IP\_Clk,  Bus2IP\_Resetn => ipif\_Bus2IP\_Resetn,  Bus2IP\_Data => ipif\_Bus2IP\_Data,  Bus2IP\_BE => ipif\_Bus2IP\_BE,  Bus2IP\_RdCE => user\_Bus2IP\_RdCE,  Bus2IP\_WrCE => user\_Bus2IP\_WrCE,  IP2Bus\_Data => user\_IP2Bus\_Data,  IP2Bus\_RdAck => user\_IP2Bus\_RdAck,  IP2Bus\_WrAck => user\_IP2Bus\_WrAck,  IP2Bus\_Error => user\_IP2Bus\_Error  );  ------------------------------------------  -- connect internal signals  ------------------------------------------  ipif\_IP2Bus\_Data <= user\_IP2Bus\_Data;  ipif\_IP2Bus\_WrAck <= user\_IP2Bus\_WrAck;  ipif\_IP2Bus\_RdAck <= user\_IP2Bus\_RdAck;  ipif\_IP2Bus\_Error <= user\_IP2Bus\_Error;  user\_Bus2IP\_RdCE <= ipif\_Bus2IP\_RdCE(USER\_NUM\_REG-1 downto 0);  user\_Bus2IP\_WrCE <= ipif\_Bus2IP\_WrCE(USER\_NUM\_REG-1 downto 0);  end IMP; |