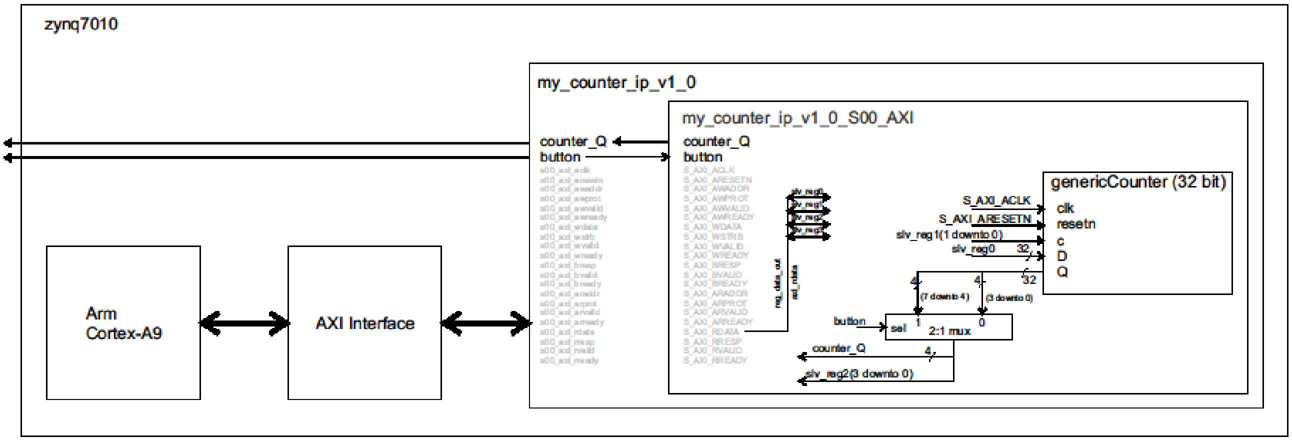
Creating an AXI peripheral.

Our goal will be to connect our custom IP to the Arm Cortex-A9 through the AXI interface. This will create the following architecture



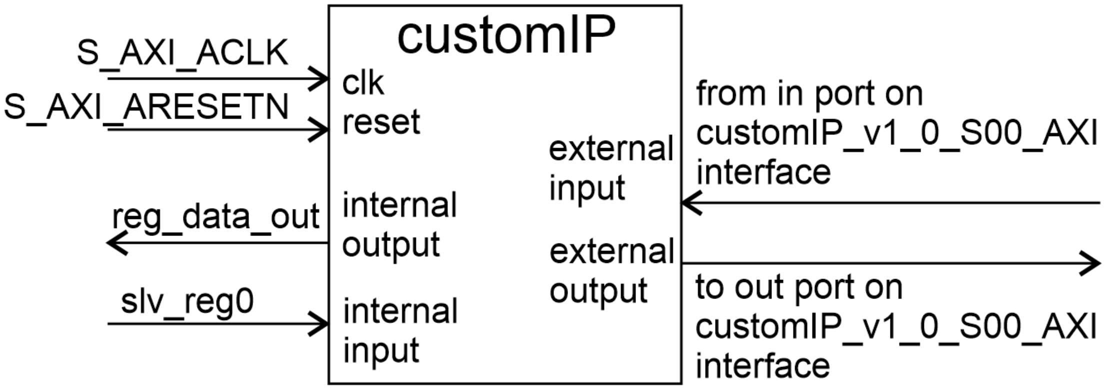
As a first step, you will want to add your custom IP inside the my_counter_ip_v1_0_S00_AXI architecture. Scroll to the bottom of this architecture and find the handy “insert your code here” comments. Paste in your code.

```
    -- Add user logic here
    myCounter: genericCounter
        generic map (32)
        port map (
            clk => S_AXI_ACLK,
            resetn => S_AXI_ARESETN,
            c => slv_reg1(1 downto 0),
            D => slv_reg0(31 downto 0),
            Q => counter_Q_Internal);

    counter_Q <= counter_Q_Internal(7 downto 4) when button='1' else counter_Q_Internal(3 downto 0);
    -- User logic ends

end arch_imp;
```

While the genericCounter and mux look familiar, some of the signals look strange. In general terms, your custom IP will get its input from either the Arm Cortex-A9 (internal) or from outside the Zynq chip (external). The same goes for the outputs. Let’s examine how to make each of these connections.



Any external inputs or outputs must be added to the my_counter_ip_v1_0_S00_AXI entities port. The red text shows where you should add these external signals.

```
entity my_counter_ip_v1_0_S00_AXI is
       generic (
             -- Width of S_AXI data bus
             C_S_AXI_DATA_WIDTH  : integer    := 32;
             -- Width of S_AXI address bus
             C_S_AXI_ADDR_WIDTH  : integer    := 4
       );
       port (
             counter_Q :  out std_logic_vector(3 downto 0);
             button:      in std_logic;

             -- Global Clock Signal
             S_AXI_ACLK    : in std_logic;
             -- Global Reset Signal. This Signal is Active LOW
```

You can provide internal inputs directly from the slv_regx directly. For example, the control and data inputs to the counter. Internal outputs must be assigned a local signal (counter_Q_Internal in the counter instantiation) and then use this local signal to replace one of the slv_reg reference in a process that sits around the middle of the architecture.

```
       process (slv_reg0, slv_reg1, counter_Q_Internal, slv_reg3, axi_araddr, S_AXI_ARESETN,…
       variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
       begin
           -- Address decoding for reading registers
           loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
           case loc_addr is
             when b"00" =>
               reg_data_out <= slv_reg0;
             when b"01" =>
               reg_data_out <= slv_reg1;
             when b"10" =>
               reg_data_out <= counter_Q_Internal;
             when b"11" =>
               reg_data_out <= slv_reg3;
             when others =>
               reg_data_out  <= (others => '0');
           end case;
       end process;
```

Next you need to provide the component declaration for any components you added to inside the my_counter_ip_v1_0_S00_AXI architecture as well as any signals needed to connect the parts of your IP together.

```
architecture arch_imp of my_counter_ip_v1_0_S00_AXI is

    component genericCounter is
        generic(N: integer:=4);
        port(   clk,resetn : in std_logic;
                c: in STD_LOGIC_VECTOR(1 downto 0);
                d : in  STD_LOGIC_VECTOR(N-1 downto 0);
                q : out STD_LOGIC_VECTOR(N-1 downto 0));
    end component;
    signal counter_Q_Internal: std_logic_vector(31 downto 0);

       -- AXI4LITE signals
       signal axi_awaddr   : std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
```

You are done with my_counter_ip_v1_0_S00_AXI. Now let's look at the my_counter_ip_v1_0 component which wraps everything up and will interface with the AXI bus.

Start by adding the external inputs or outputs from the my_counter_ip_v1_0_S00_AXI entities to the my_counter_ip_v1_0 component entity port. The red text shows where you should add these external signals.

```
entity my_counter_ip_v1_0 is
	generic (
		C_S00_AXI_DATA_WIDTH	: integer	:= 32;
		C_S00_AXI_ADDR_WIDTH	: integer	:= 4
	);
	port (
		-- Users to add ports here
		counter_Q : out std_logic_vector(3 downto 0);
		button: in std_logic;
		-- User ports ends
		-- Do not modify the ports beyond this line


		-- Ports of Axi Slave Bus Interface S00_AXI
```

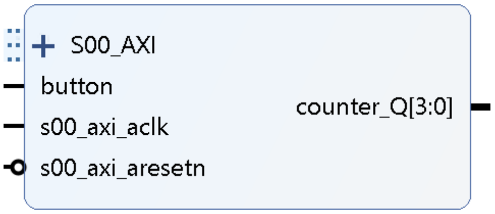
Next you need to edit the component declaration of the my_counter_ip_v1_0_S00_AXI entity to include the external input and output ports that you added in the previous step. Scroll down and find the my_counter_ip_v1_0_S00_AXI component declaration and add the red text.

```
	-- component declaration
	component my_counter_ip_v1_0_S00_AXI is
		generic (
		C_S_AXI_DATA_WIDTH	: integer	:= 32;
		C_S_AXI_ADDR_WIDTH	: integer	:= 4
		);
		port (
		counter_Q : out std_logic_vector(3 downto 0);
		button: in std_logic;
		S_AXI_ACLK	: in std_logic;
		S_AXI_ARESETN	: in std_logic;
```

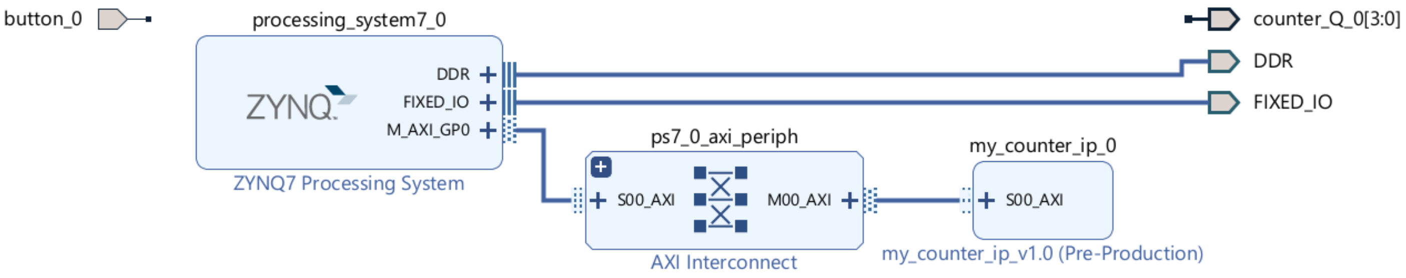
Finally, you need to connect the external ports on my_counter_ip_v1_0_S00_AXI entity to the external ports on the my_counter_ip_v1_0_S00_AXI entity.

```
-- Instantiation of Axi Bus Interface S00_AXI
my_counter_ip_v1_0_S00_AXI_inst : my_counter_ip_v1_0_S00_AXI
	generic map (
		C_S_AXI_DATA_WIDTH	=> C_S00_AXI_DATA_WIDTH,
		C_S_AXI_ADDR_WIDTH	=> C_S00_AXI_ADDR_WIDTH
	)
	port map (
		counter_Q => counter_Q,
		button => button,
		S_AXI_ACLK	=> s00_axi_aclk,
		S_AXI_ARESETN	=> s00_axi_aresetn,
		S_AXI_AWADDR	=> s00_axi_awaddr,
```

If you followed all these steps correctly, your custom IP should now have a block diagram that looks like this.



You will then take your custom IP and connect it the Arm Cortex-A9 through the AXI interconnect.



You will need to specify the pin assignment for the button_0 and counter_Q_0