

## **PCI Express Test Build.**

There follows a description of the build to run the PCI express interface on the Raggedstone2 board.

### **Content.**

#### **Design Structure.**

**PCI Express Input and Outputs with IO Controller input and outputs.**

**Raggedstone2 PCI Express Test Procedure.**

**Obtaining the code, driver and GUI.**

**Project Files and Location.**

**Procedure for Building the Design.**

**IPCORE Generated files**

**Running the example design.**

**Synthesizing the design example.**

**Modifying the design to include the IO controller.**

**Code Edits.**

**Reset.**

**Synthesizing the Modified Design.**

**Implementing the design.**

**Generate Programming File.**

**Configure Target Device running Impact.**

**Inserting ChipScope.**

**User Constraints File.**

**Testing the Build.**

**Pre installed Raggedstone2.**

**Installing the Raggedstone2.**

**Check Installation has worked.**

**Running with the Device Manager.**

**Running PCI32.**

**Installing the Driver.**

**Testing the PCI Express.**

**GUI working buttons.**

**Further Tests.**

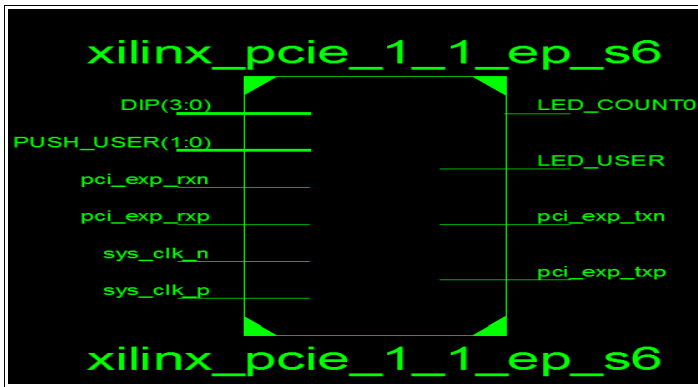
**Raggedstone2 PCI Express test using ChipScope.**

#### **Design Structure.**

The project starts by generating a PCI Express core. With this core, a Xilinx design example is automatically created, which when run, adds a PIO core to the PCI Express core as an Endpoint. This project adds in addition to the PIO an IO controller, connected between the PIO and the PC interface.

The input to the PC interface is a GUI and this allows two of the Raggedstone2 LEDs to be toggled. As the downloaded files were in Verilog, this has been the coding language used for this project.

#### **PCI Express Input and Outputs with IO Controller input and outputs.**



Top Level xilinx\_pcie\_1\_1\_ep\_s6.

IO controller inputs/outputs DIP, PUSH\_USER, LED\_COUNT0 and LED\_USER have to be added.

## **Raggedstone2 PCI Express Test Procedure.**

This test uses the driver, GUI and part of the test code downloaded from the AVNET site.

### **Obtaining the code, driver and GUI.**

The link to the Avnet site is:

<http://www.em.avnet.com/evk/home/0,4534,CID%3D57572%26CCD%3DUSA%26SID%3D32214%26DID%3DDF2%26LID%3D32232%26BID%3DDF2%26CTP%3DEVK,00.html?SUL=spartan6lx150t-dev>

### **Go to Support Files and Downloads.**

Before clicking on this, a new account has to be created.

Then download **S6LX150T Development Board PCIe Example Design.**

The downloaded code is unzipped to **s6lx150t\_pcie\_design\_edk11\_4**

This contains the code, driver and GUI.

### **Project Files and Location.**

```

raggedstone2\rs2_pcie\FPGA_BUILDS\MACROS\chipscope_icon.v
raggedstone2\rs2_pcie\FPGA_BUILDS\MACROS\chipscope_ila.v
raggedstone2\rs2_pcie\FPGA_BUILDS\UCF\xilinx_pcie_1_lane_ep_xc6slx45t-fgg484-2.ucf
raggedstone2\rs2_pcie\FPGA_BUILDS\bar0_io_devices.v
raggedstone2\rs2_pcie\FPGA_BUILDS\bar0_registers.v
raggedstone2\rs2_pcie\FPGA_BUILDS\W\xilinx_pcie_1_1_ep_s6.v
raggedstone2\rs2_pcie\ipcore_dir\s6_pcie\example_design\pcie_app_s6.v
raggedstone2\rs2_pcie\ipcore_dir\s6_pcie\example_design\PIO.v
raggedstone2\rs2_pcie\ipcore_dir\s6_pcie\example_design\PIO_32_RX_ENGINE.v
raggedstone2\rs2_pcie\ipcore_dir\s6_pcie\example_design\PIO_32_TX_ENGINE.v
raggedstone2\rs2_pcie\ipcore_dir\s6_pcie\example_design\PIO_EP.v
raggedstone2\rs2_pcie\ipcore_dir\s6_pcie\example_design\PIO_EP_MEM.v
raggedstone2\rs2_pcie\ipcore_dir\s6_pcie\example_design\PIO_EP_MEM_ACCESS.v
raggedstone2\rs2_pcie\ipcore_dir\s6_pcie\example_design\PIO_TO_CTRL.v
raggedstone2\rs2_pcie\ipcore_dir\s6_pcie\source\gtpa1_dual_wrapper.v
raggedstone2\rs2_pcie\ipcore_dir\s6_pcie\source\gtpa1_dual_wrapper_tile.v
raggedstone2\rs2_pcie\ipcore_dir\s6_pcie\source\pcie_brams_s6.v
raggedstone2\rs2_pcie\ipcore_dir\s6_pcie\source\pcie_bram_s6.v
raggedstone2\rs2_pcie\ipcore_dir\s6_pcie\source\pcie_bram_top_s6.v
raggedstone2\rs2_pcie\ipcore_dir\s6_pcie\source\s6_pcie.v

```

The files used in this project include:

### **Procedure for Building the Design.**

We start by creating a new project **rs2\_pcie**.

**File>New Project**

**Create New Project**  
Specify project location and type.

Enter a name, locations, and comment for the project

Name:

Location:

Working Directory:

Description:

New Project **rs2\_pcie** and folder location.

We then create our PCI Express core.

**Project>New Source**

This generates the core page windows. We name the component **s6\_pcie**.

**LogiCORE** **Spartan-6 Integrated Block for PCI Express**

Component Name

PCIe Device / Port Type

The Integrated Block for PCI Express allows selection of the Device / Port Type

Device / Port Type

Page 1: Component name and Device type.

Click on and the next for the next window.

**Base Address Registers**

Base Address Registers (BARs) serve two purposes. Initially, they se space in the system memory map. After the BIOS or OS determines Registers are programmed with addresses and the device uses this i

**BAR 0 Options**

Bar0 Type   64 bit  Prefetchable

Size

Value  (Hex)

Page 2: BAR0 should be not selected. BAR1 to BAR5 should be deselected.

ID Initial Values		
Vendor ID	<input type="text" value="1597"/>	Range: 0000..FFFF
Device ID	<input type="text" value="0301"/>	Range: 0000..FFFF
Revision ID	<input type="text" value="00"/>	Range: 00..FF
Subsystem Vendor ID	<input type="text" value="1597"/>	Range: 0000..FFFF
Subsystem ID	<input type="text" value="0001"/>	Range: 0000..FFFF
Class Code		
Base Class	<input type="text" value="0B"/>	Range: 00..FF
Sub-Class	<input type="text" value="40"/>	Range: 00..FF
Interface	<input type="text" value="00"/>	Range: 00..FF
Class Code	<input type="text" value="0B4000"/>	(Hex)
Cardbus CIS Pointer		
Cardbus CIS Pointer	<input type="text" value="00000000"/>	Range: 00000000..FFFFFFFF

Page 3: The ID and Class Codes should be as above.

Page 4-8 should be left as default values.

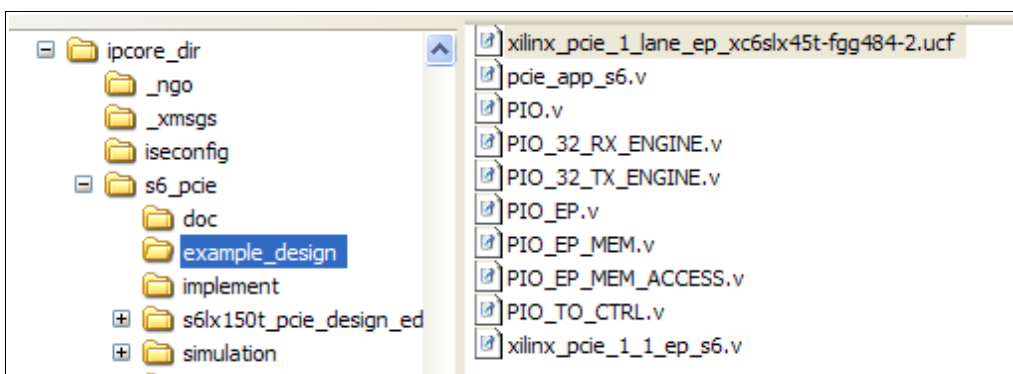
Reference Clock Frequency	
The Integrated Block for PCI Express allows selection of the reference clock frequency	
Frequency (MHz)	<input type="text" value="100 MHz"/>

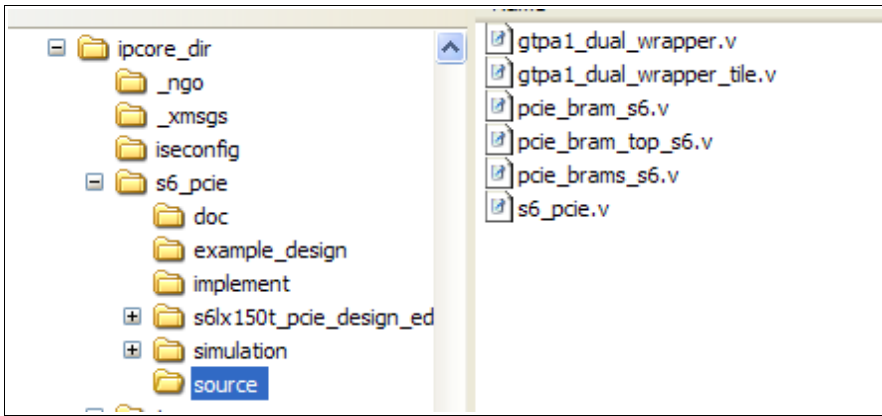
Page 9 should have the reference clock set at 100MHz.  
Following this we click on **Generate**.

From the Design Hierarchy **s6\_pcie.xco** has been generated.

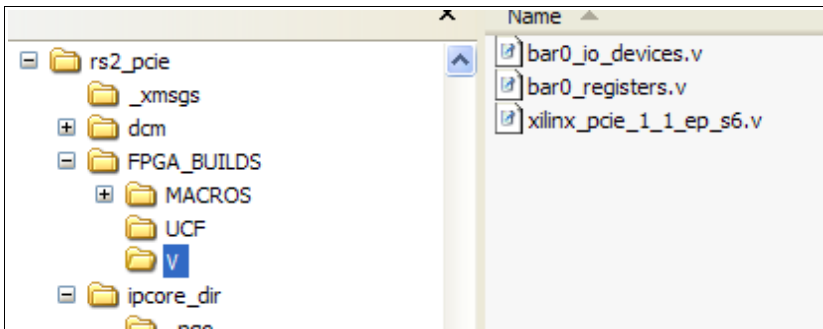
### **IPCORE Generated files**

Generating the core creates new folders for the **ipcore** and also the example design.  
Example design folder.





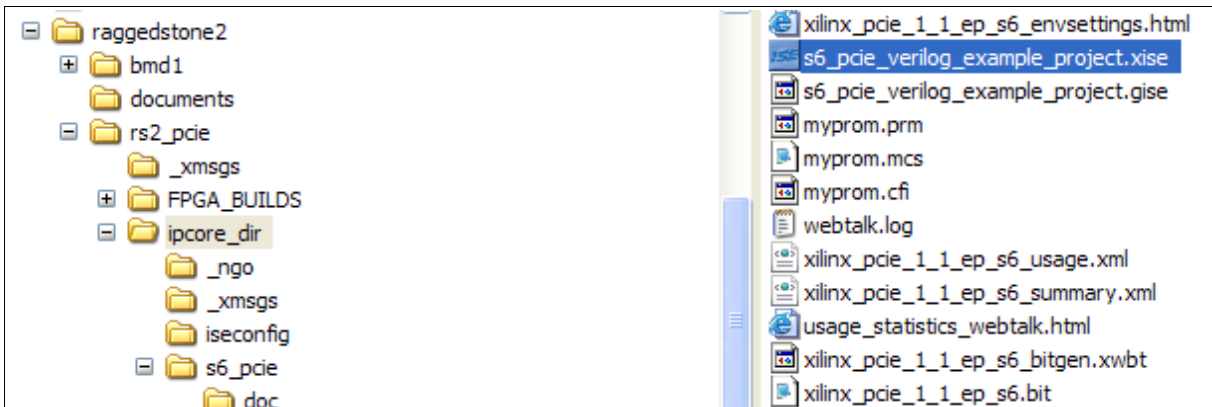
Source folder auto generated files.



We copy `xilinx_pcie_1_1_ep_s6.v` to the V folder and also `bar0_io_devices.v` and `bar0_registers.v` files.

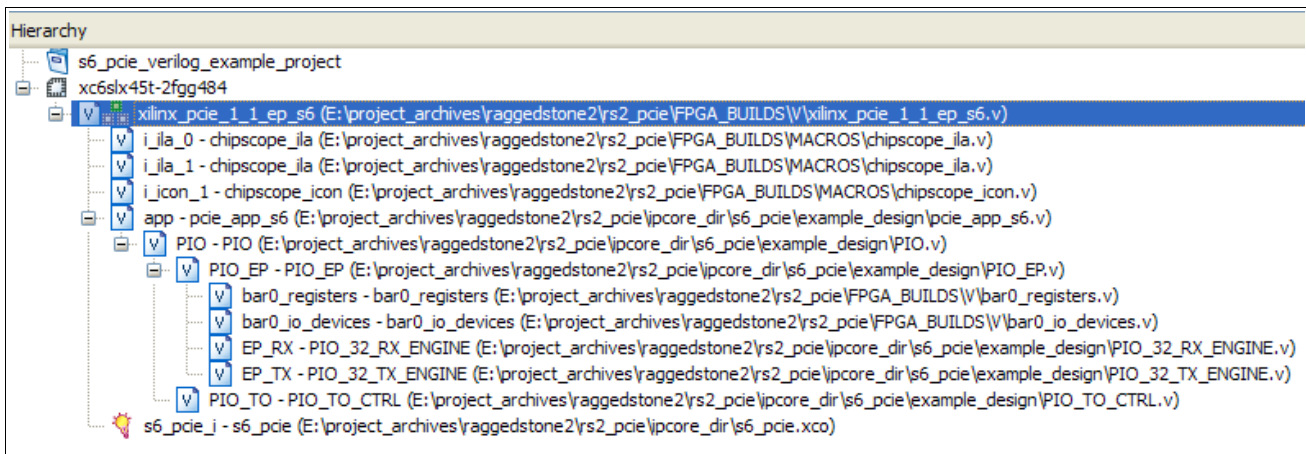
### **Running the example design.**

We can now run the example design by loading the example project in the ipcore folder.



File location of `s6_pcie_verilog_example_project.xise`

We execute this file and load the design example.



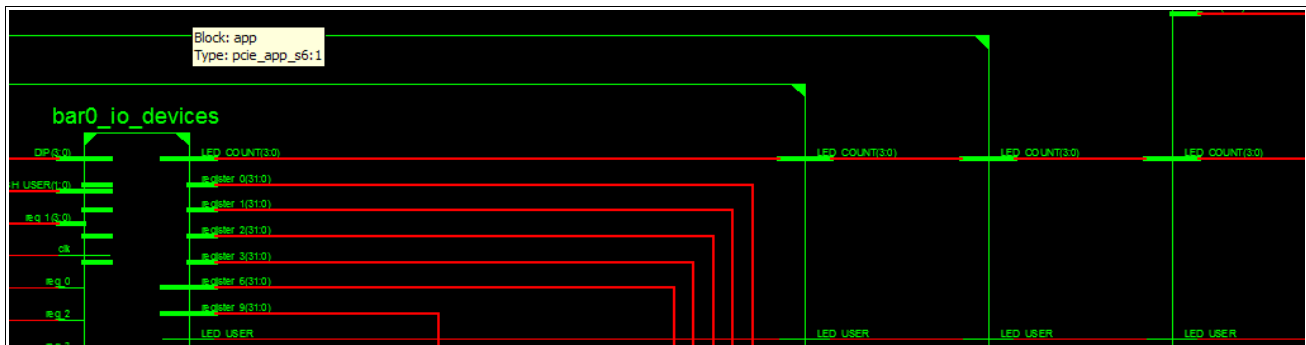
File structure having run `s6_pcie_verilog_example_project.xise`.

### Synthesizing the design example.

Once the design example has been run, it should be synthesized to make sure all is correct before moving on to editing the code.

### Modifying the design to include the IO controller.

The IO Controller has inputs at the top level down through to the PIO. At the PIO level we instantiate bar registers, which read these inputs from the toplevel (DIP and PUSH\_USER) and then output to the top level (LED\_USER and LED\_COUNT).



LED\_COUNT[3:0] and LED\_USER instantiated at each level from bar0\_io\_devices to toplevel.

The inputs and outputs that connect to the GUI are as follows:

### Code Edits.

The following files need to be edited to allow the PCIE to be tested using the GUI interface with the io controller placed at the PIO level.

```

module xilinx_pcie_1_1_ep_s6
(
    output          LED_USER,
    output [3:0]    LED_COUNT,
    input  DIP[3:0],
    input  PUSH_USER[1:0]
);

pcie_app_s6 app          //
instantiation.
(
    .LED_USER(LED_USER),
    .LED_COUNT(LED_COUNT),
    .DIP(DIP),
    .PUSH_USER(PUSH_USER)
);

```

**File: xilinx\_pcie\_1\_1\_ep\_s6.v.**

Add input outputs to top module xilinx\_pcie\_1\_1\_ep\_s6 and instantiate those inputs/outputs in pcie\_app\_s6.

```

module pcie_app_s6
(
    output          LED_USER,
    output [3:0]    LED_COUNT,
    input  DIP[3:0],
    input  PUSH_USER[1:0]
);

PIO PIO          // instantiation.
(
    .LED_USER(LED_USER),
    .LED_COUNT(LED_COUNT),
    .DIP(DIP),
    .PUSH_USER(PUSH_USER)
);

```

**File: pcie\_app\_s6.v edits.**

Add input outputs to pcie\_app\_s6 and instantiate those inputs/outputs in PIO

```

module PIO
(
    output          LED_USER,
    output [3:0]    LED_COUNT,
    input  DIP[3:0],
    input  PUSH_USER[1:0]

);

PIO_EP PIO_EP      // instantiation.
(
    .LED_USER(LED_USER),
    .LED_COUNT(LED_COUNT),
    .DIP(DIP),
    .PUSH_USER(PUSH_USER)
);

```

**File: PIO.v.**

Add input/outputs to PIO and instantiate those inputs/outputs in PIO\_EP.

```

input          cfg_bus_mstr_enable,
output        LED_USER,
output [3:0]   LED_COUNT,
input  [3:0]   DIP,
input  [1:0]   PUSH_USER
);

```

File: PIO\_EP.v. Instantiate in module PIO\_EP the input and outputs.

```

bar0_registers bar0_registers (

    .clk(clk),                // I
    .rst_n(rst_n),            // I

    .reg_0(reg_0),             // O
    .reg_1(reg_1),             // O [3:0]

    .register_0(register_0),    // I [31:0]
    .register_1(register_1),    // I [31:0]
    .register_6(register_6),    // I [31:0]
    .register_9(register_9),

    // Read Port

    .rd_addr_i(rd_addr),       // I [10:0]
    .rd_be_i(rd_be),           // I [3:0]
    .rd_data_o(rd_data),       // O [31:0]

    // Write Port

    .wr_addr_i(wr_addr),       // I [10:0]
    .wr_be_i(wr_be),           // I [7:0]
    .wr_data_i(wr_data),       // I [31:0]
    .wr_en_i(wr_en),           // I
    .wr_busy_o(wr_busy)        // O

);

```

File: PIO\_EP.v. Instantiate the bar0\_registers.

```

bar0_io_devices bar0_io_devices (
    .clk(clk),                // I
    .rst_n(rst_n),           // I

    .LED_USER(LED_USER),     // O
    .LED_COUNT(LED_COUNT),   // O [3:0]
    .DIP(DIP),               // I [3:0]
    .PUSH_USER(PUSH_USER),   // I [1:0]

    .reg_0(reg_0),           // I
    .reg_1(reg_1),           // I [3:0]

    .register_0(register_0),  // O [31:0]
    .register_1(register_1),  // O [31:0]
    .register_6(register_6),  // O [31:0]
    .register_9(register_9)
);

```

File: PIO\_EP.v. Instantiate the bar0\_io\_devices.

```

190     .register_6(register_6), // O [31:0]
191     .register_2(register_2), // O [31:0]
192     .register_9(register_9)
193 );
194
195
196
197 //
198 // ENDPOINT MEMORY : 8KB memory aperture implemented in FPGA BlockRAM(*)
199 //
200
201 // PIO_EP_MEM_ACCESS EP_MEM (
202 //     .clk(clk),                // I
203 //     .rst_n(rst_n),           // I
204 //
205 //     // Read Port
206 //     .rd_addr_i(rd_addr),     // I [10:0]
207 //     .rd_be_i(rd_be),        // I [3:0]
208 //     .rd_data_o(rd_data),    // O [31:0]
209 //
210 //     // Write Port
211 //     .wr_addr_i(wr_addr),     // I [10:0]
212 //     .wr_be_i(wr_be),        // I [7:0]
213 //     .wr_data_i(wr_data),    // I [31:0]
214 //     .wr_en_i(wr_en),       // I
215 //     .wr_busy_o(wr_busy)     // O
216 //
217 // );
218

```

File: PIO\_EP.v, comment out PIO\_EP\_MEM\_ACCESS

### **Reset.**

The PCI Express reset is not connected to the interface so this is hardwired to 1.

```

// System (SYS) Interface
.sys_clk             ( sys_clk_c
.sys_reset_n        ( 1'b1
.received_hot_reset (
);

```

File: xilinx\_pcie\_1\_1\_ep\_s6.v Top level instantiation of s6\_pcie,

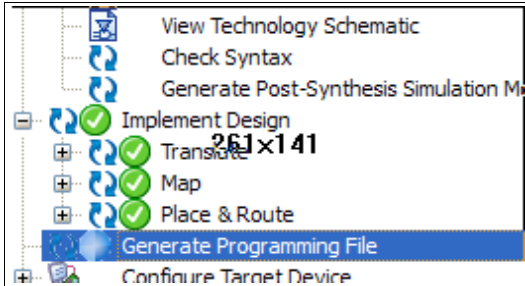
```
reset = 1'b1.
```

### Synthesizing the Modified Design.

Double click on Synthesize - XST and remove all errors.

### Implementing the design.

Double click on Implement.  
All boxes should be ticked.

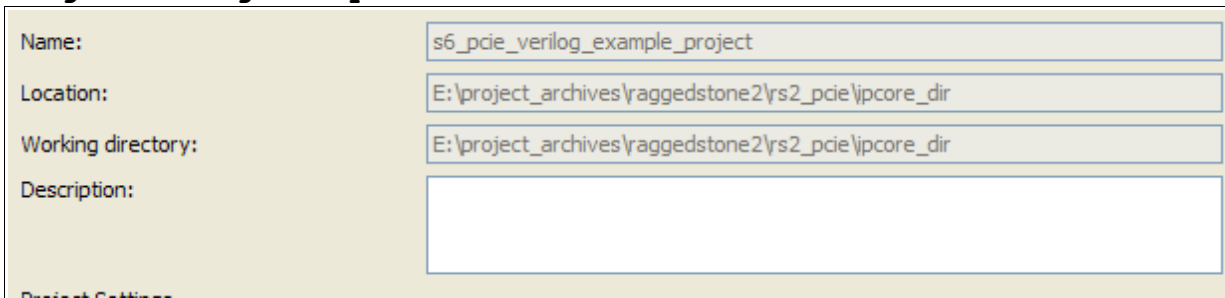


Implementation: A good result.

### Generate Programming File.

We double click on Generate Programming File and this creates xilinx\_pcie\_1\_1\_ep\_s6.bit (toplevel name.bit).  
This file is located in the work directory.

### Project>Design Properties



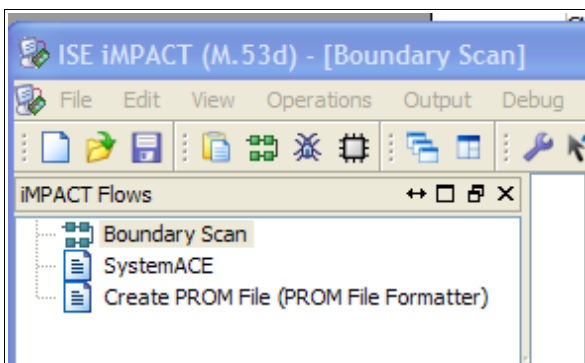
The working directory is the **ipcore** folder and the .bit file will be found here.

### Configure Target Device running Impact.

Double click on the Configure Target Device.

This runs **Impact**.

From the IMPACT window and with the Raggedstone2 board connected for programming we double click on Boundary Scan.

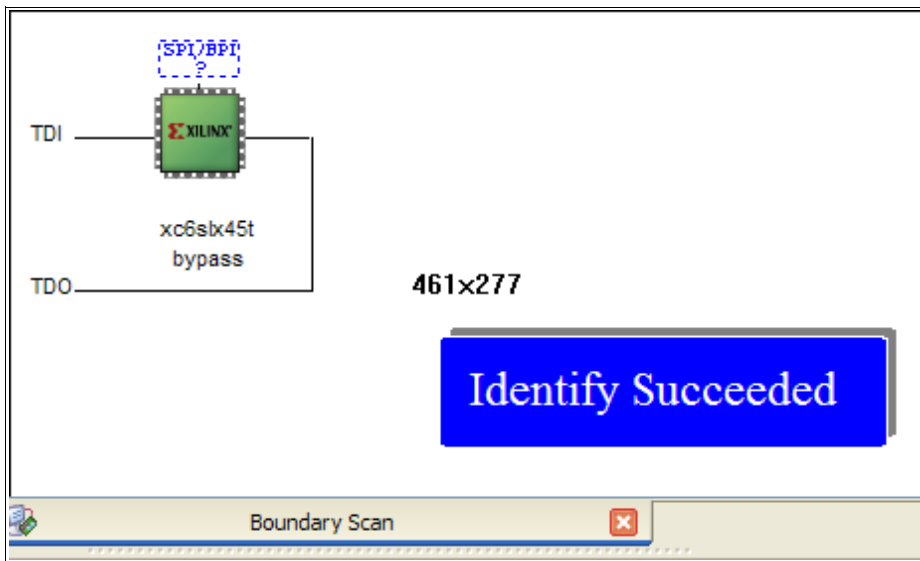


Boundary scan clicked on.

Click on the initialize chain symbol.



Initialize chain symbol

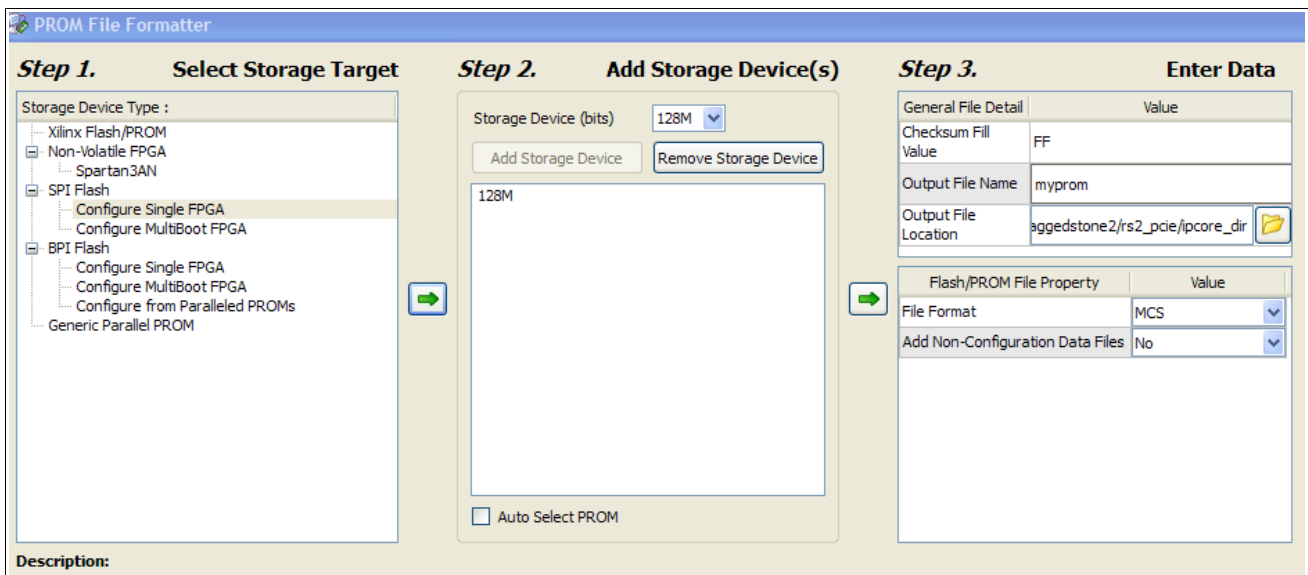


Succeeded connection to the Raggedstone2 board.

We then click on the Create Prom File Formatter.

This generates the window below. We follow the steps for an SPI Flash, a Configure Single FPGA, a device of 128M. Add Storage once selected.

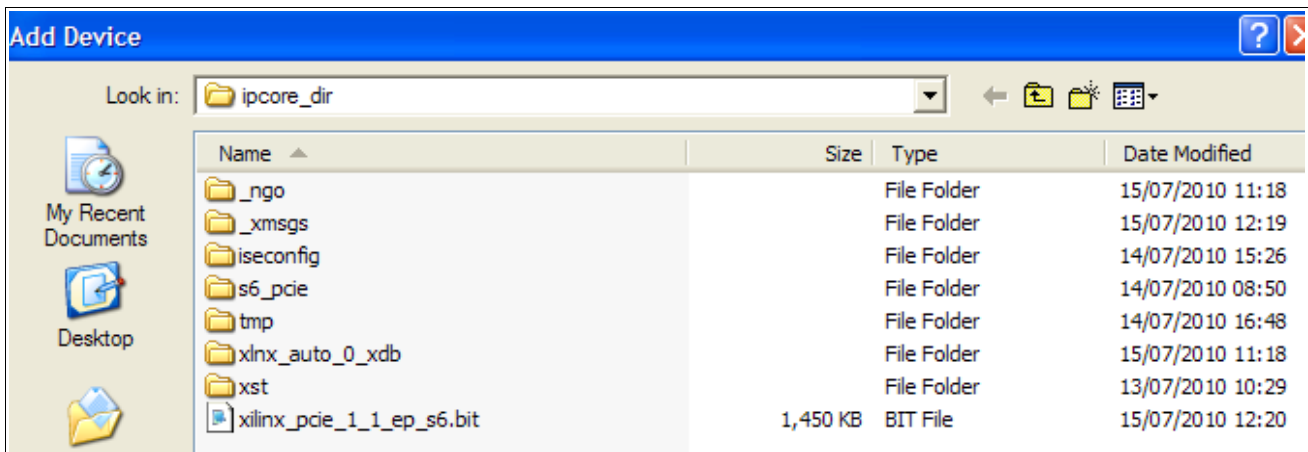
We locate the folder for the .mcs file, generated on the next stage.



PROM file formatter. Single FPGA, Storage Device 128M, location work directory, output file name myprom.

When OK is pressed, we are told to select the file.

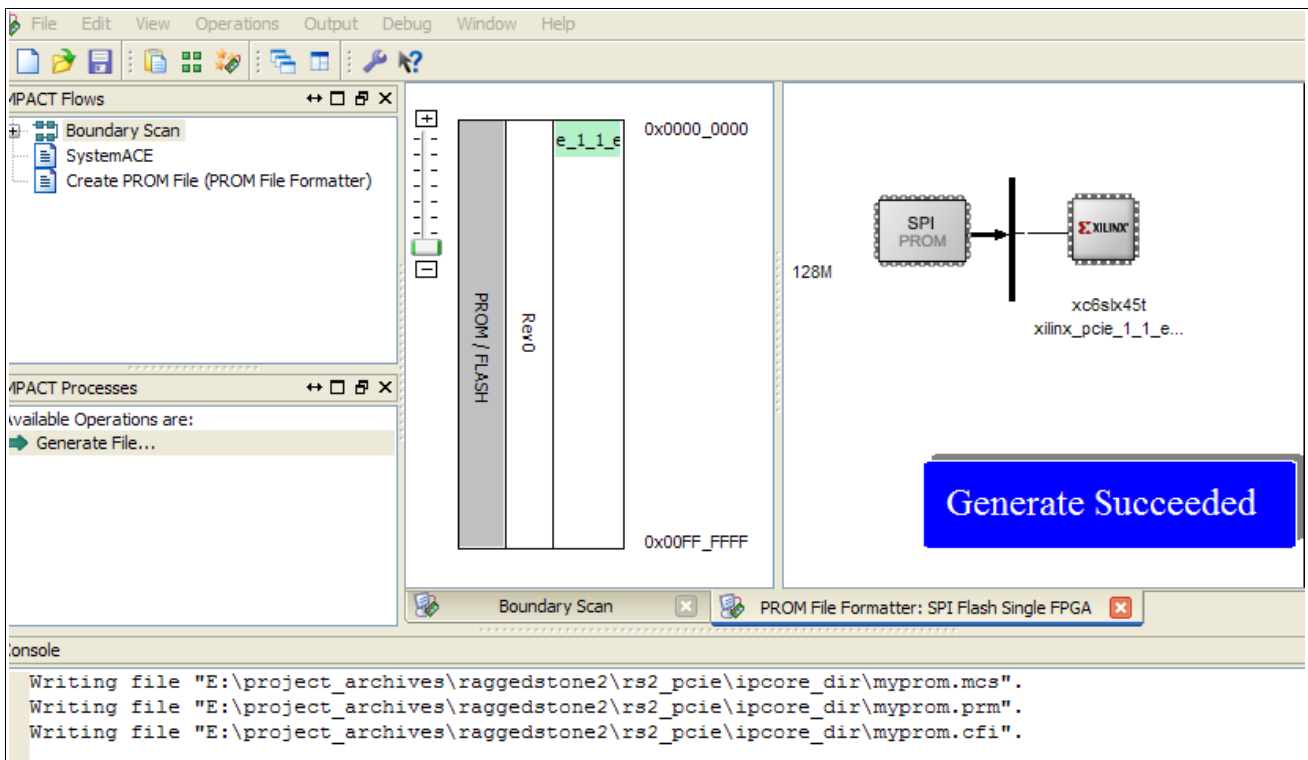
It should be in the work folder and check that the modified time is correct.



Add Device: top level .bit file from Generate File.

Select this file.

Next we click on **Generate File** to create the programming myprom.mcs file.

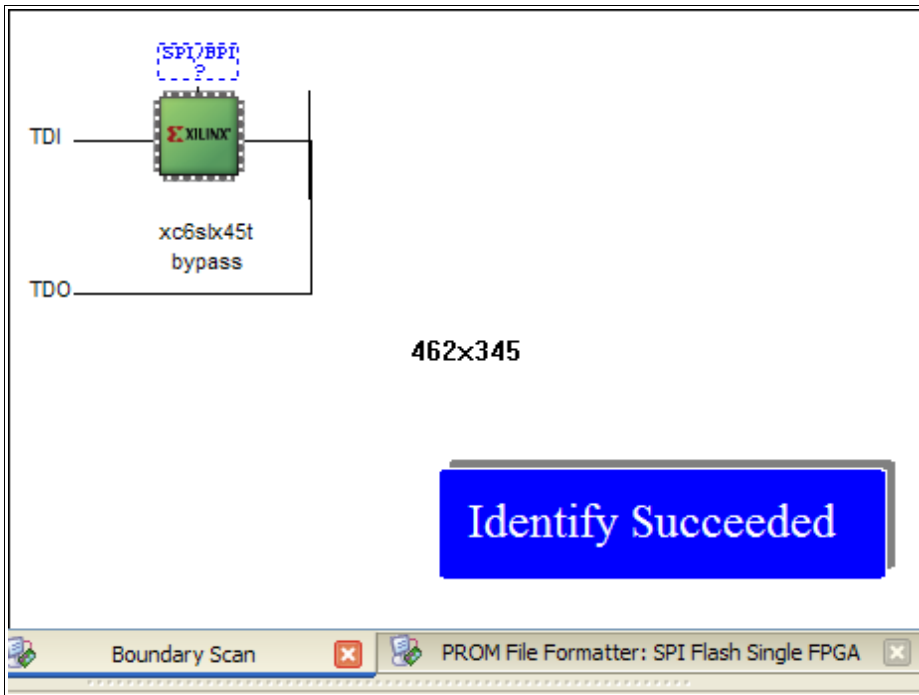


Generate File: succeeded and shows the folder for the myprom.mcs file.

Now to load myprom.mcs file.

Click on the Boundary Scan Tab.

Note: If this is not evident, Impact might have to be run again.

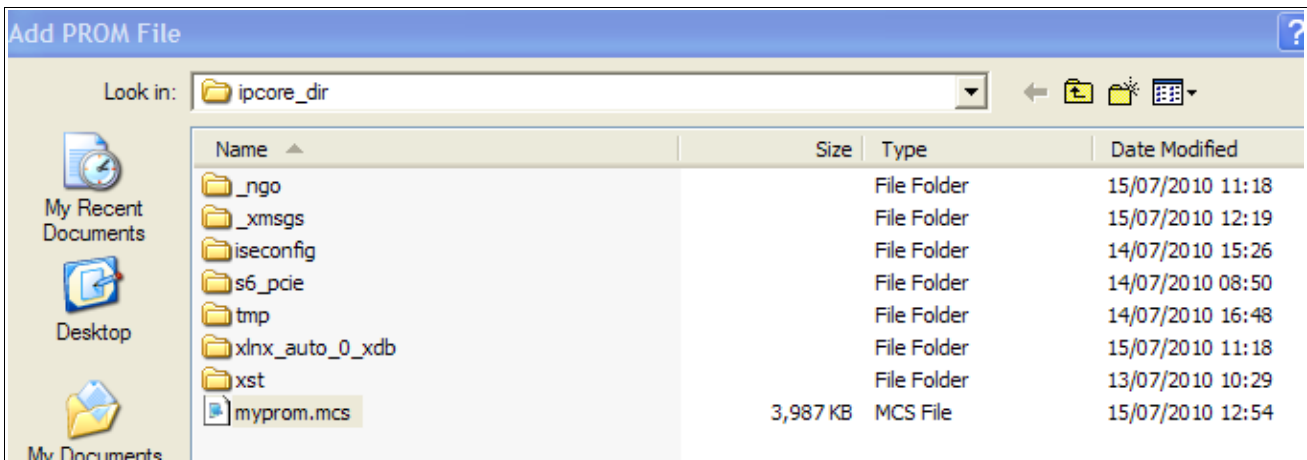


Identify succeeded.

We are now ready to program the prom.  
Right click on the light blue box above the FPGA.

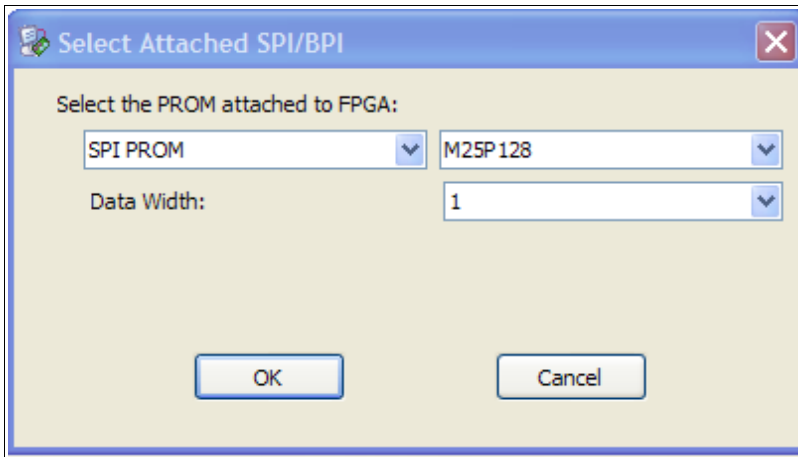
**>Add SPI Flash.**

This takes us to myprom.mcs file in the work folder.  
Check that the modified time is correct.

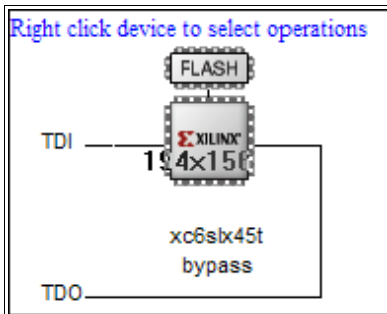


Myprom.mcs programming file for the Raggedstone2 board.

When we click OK to this, another window appears.  
We select the SPI PROM. Select M25P128.

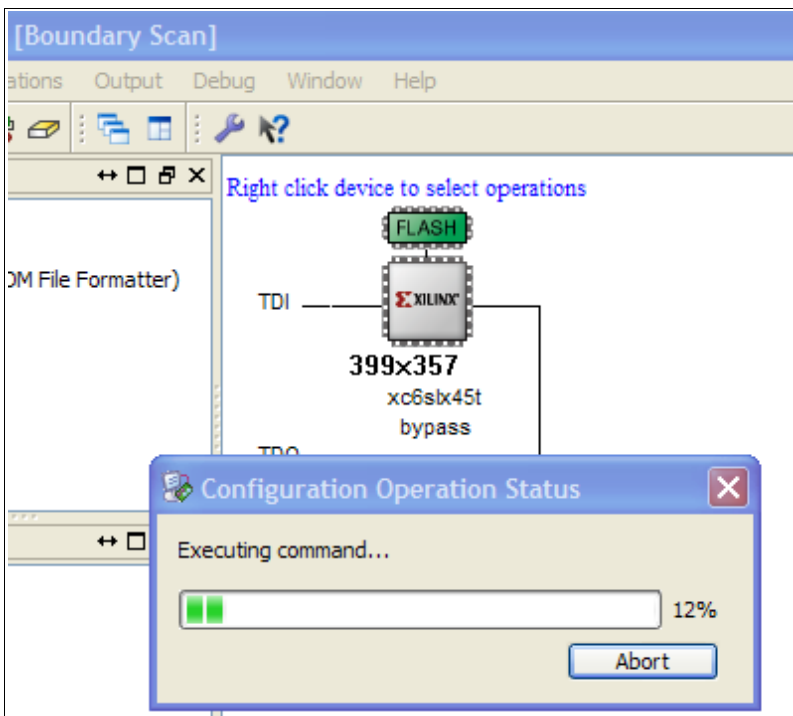


Selection of the PROM type.



Flash inserted.

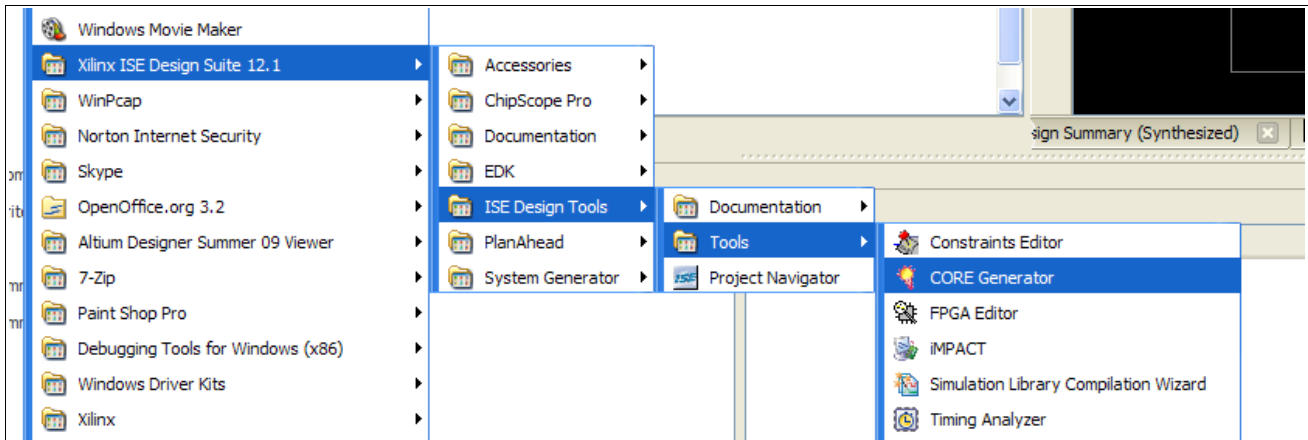
We now program the flash. Right click the FLASH and select **Program**. The flash will program to 15% and should succeed. This takes about 4minutes.



## Inserting ChipScope.

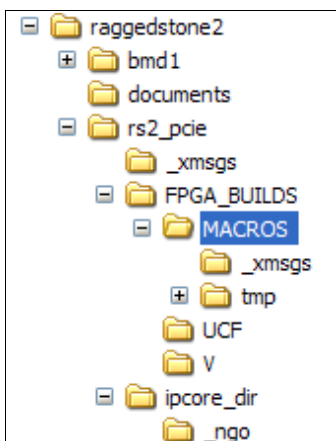
To test that the GUI is doing what it says and that the Raggedstone2 PCI Express is working correctly , we insert ChipScope to look at the signals.

First we have to generate the ChipScope Cores **ChipScope\_icon** and **ChipScope\_ila**.



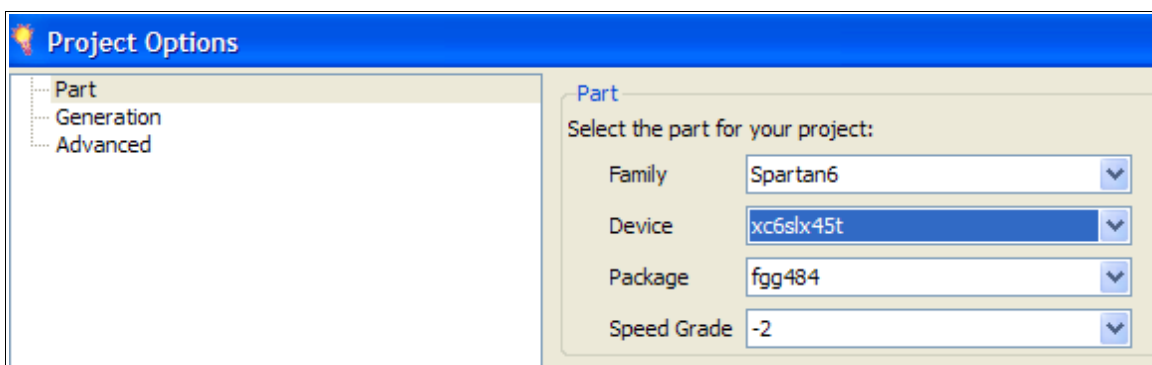
Run the Core Generator.

Once running, create a new project. This is saved in a new folder MACROS.



MACROS folder for ChipScope macros.

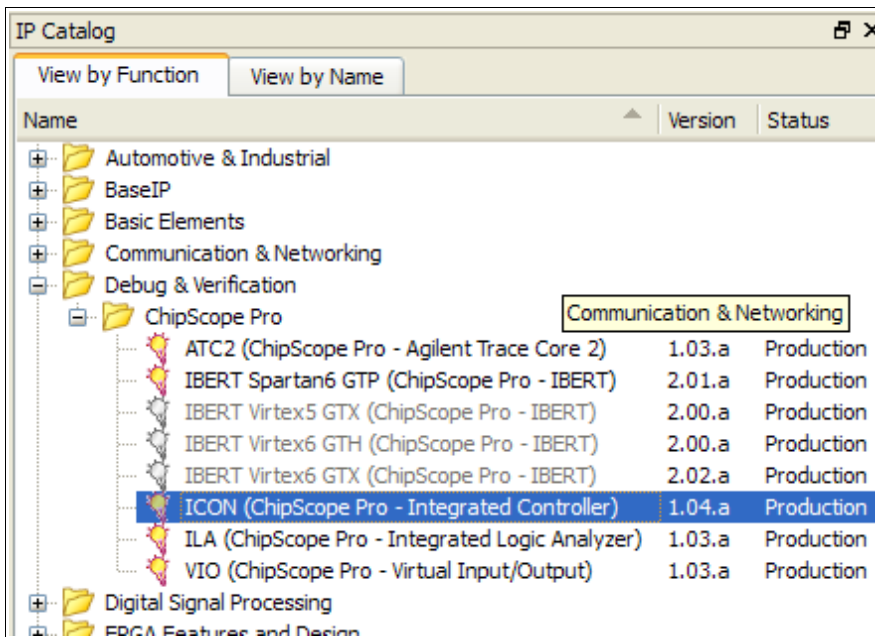
Another window appears for project options.



PART project options.

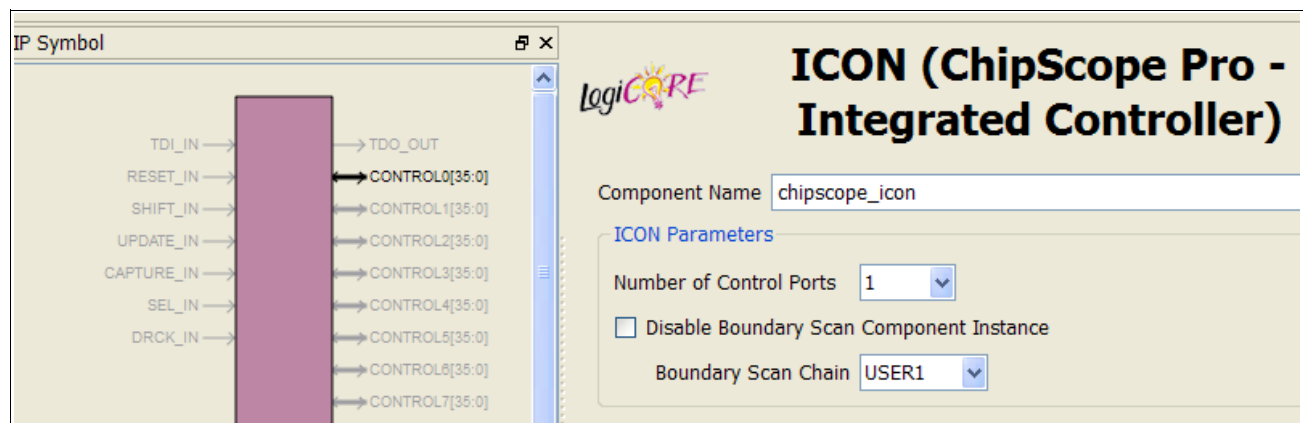
Click on Generation and select **Verilog** for Design Entry. Click OK to the window.

From the Xilinx Core Generator list, select ICON and double click on this.



Selecting ICON from the core generator.

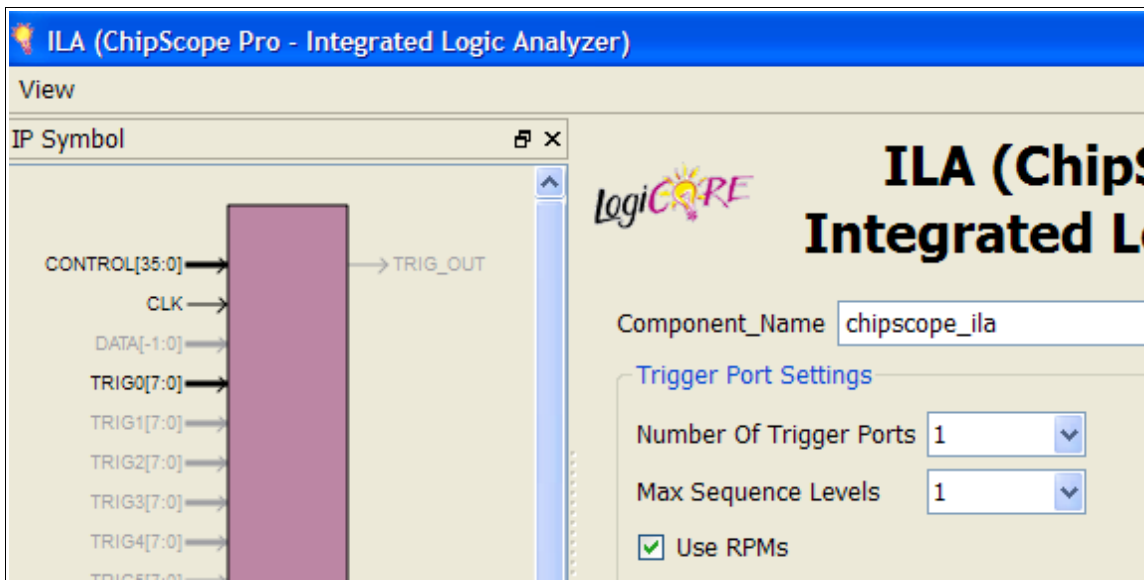
To test the PCI Express we will be using the PCI clock `trn_clk`.



One control signals selected.

### >Generate

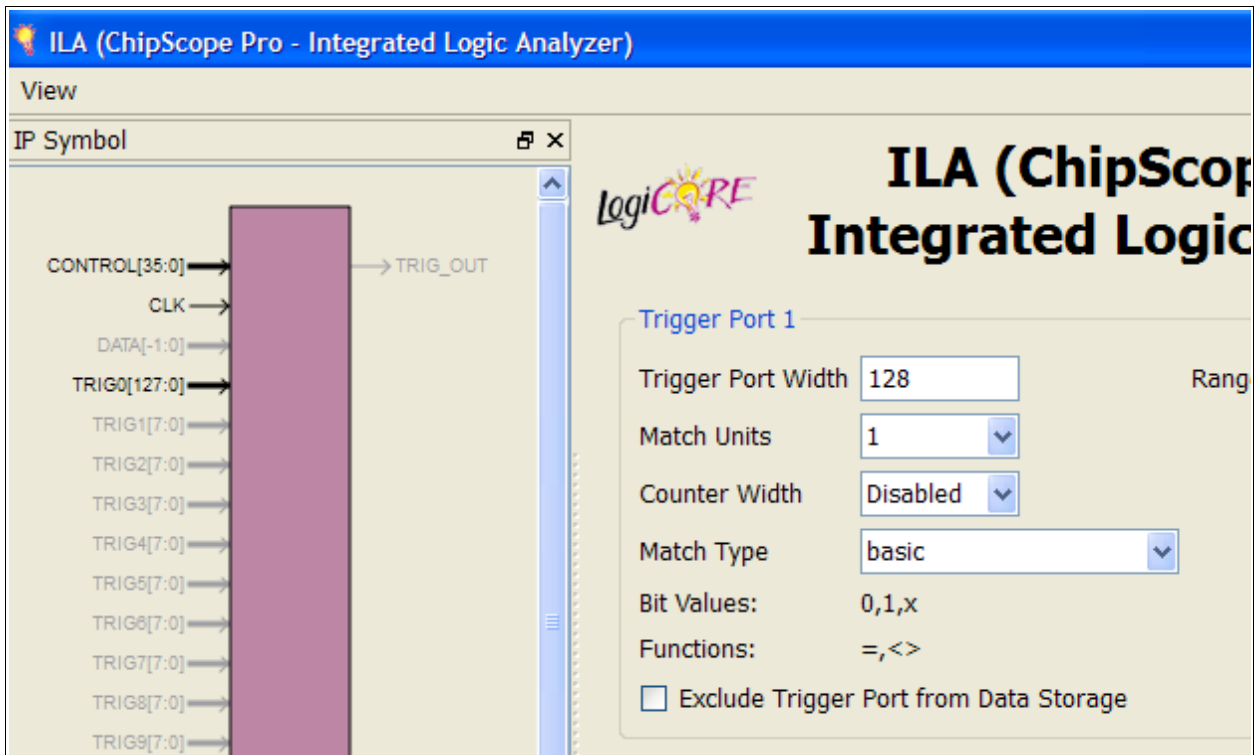
Next we double click on ILA from the core generator. We want one trigger port.



Page 1: One Trigger port

**>Next**

We want a trigger of 128 bits.



Page 2: Trigger Width changed to 128

**>Generate.**

We now have the cores for ChipScope. They just need instantiating in the code and signals assigned that we want to see, that is for loopback test and PCI Express test.

Back to the code.

We instantiate the ChipScope cores in the top level.

```

}7
}8 //----- chipscope -----
}9
}10 reg[127:0] chipscope_data;
}11 reg cnt;
}12 wire [35:0] control0;
}13
}14     chipscope_ila i_ila (
}15     .CLK(trn_clk),
}16     .TRIG0(chipscope_data[127:0]),
}17     .CONTROL(control0)
}18     );
}19
}20     chipscope_icon i_icon_1 (
}21     .CONTROL0(control0)
}22     );
}23

```

Instantiation of the ChipScope core.

We then assign the signals we want to see.

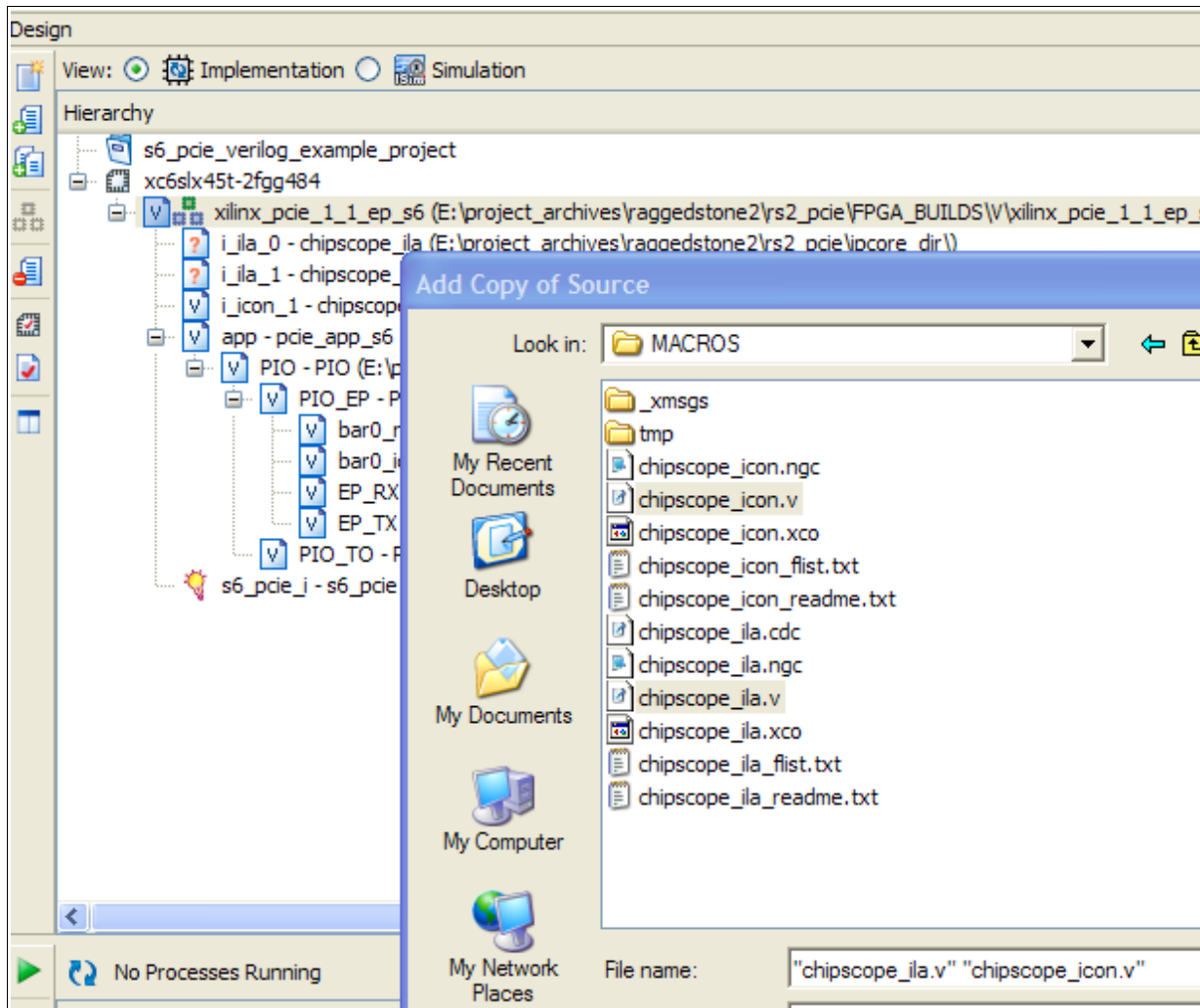
```

}
}1 always @(posedge trn_clk)|
}2
}3     begin
}4     chipscope_data[31:0]   <= trn_td;
}5     chipscope_data[63:32] <= trn_rd;
}6     chipscope_data[64]    <= trn_rsof_n;
}7     chipscope_data[65]    <= trn_reof_n;
}8     chipscope_data[66]    <= trn_rsrc_rdy_n;
}9     chipscope_data[67]    <= trn_rsrc_dsc_n;
}10    chipscope_data[68]    <= trn_rerrfwd_n;
}11    chipscope_data[74:69] <= trn_tbuf_av[5:0];
}12    chipscope_data[75]    <= trn_tcfg_req_n;
}13    chipscope_data[76]    <= trn_terr_drop_n;
}14    chipscope_data[77]    <= trn_tdst_rdy_n;
}15    chipscope_data[78]    <= trn_tsof_n;
}16    chipscope_data[79]    <= trn_teof_n;
}17    chipscope_data[80]    <= trn_tsrc_rdy_n;
}18    chipscope_data[81]    <= trn_tsrc_dsc_n;
}19    chipscope_data[82]    <= trn_terrfwd_n;
}20    chipscope_data[83]    <= trn_tcfg_gnt_n;
}21    chipscope_data[84]    <= trn_tstr_n;
}22    chipscope_data[85]    <=trn_reset_n;
}23    chipscope_data[86]    <=trn_lnk_up_n;
}24    chipscope_data[87]    <= LED_USER;
}25    chipscope_data[103:88] <= cfg_command[15:0];
}26    chipscope_data[110:104] <= trn_rbar_hit_n[6:0];
}27    chipscope_data[111]    <= trn_rdst_rdy_n;
}28    chipscope_data[112]    <= trn_rnp_ok_n;
}29    // chipscope_data[113] <= LED_COUNT[0];
}30    chipscope_data[114]    <= LED_USER;
}31    chipscope_data[115]    <= LED_USER2;
}32    chipscope_data[116]    <= LED_USER3;
}33    chipscope_data[117]    <= cfg_rd_wr_done_n;
}34    chipscope_data[118]    <= cfg_rd_wr_done_n;
}35    chipscope_data[121:119] <= trn_fc_sel[2:0];
}36    chipscope_data[122]    <= DIP[0];
}37    chipscope_data[123]    <= DIP[1];
}38    chipscope_data[124]    <= DIP[2];
}39    chipscope_data[125]    <= DIP[3];
}40    chipscope_data[126]    <= PUSH_USER[0];
}41    chipscope_data[127]    <= PUSH_USER[1];
}42     end
}43

```

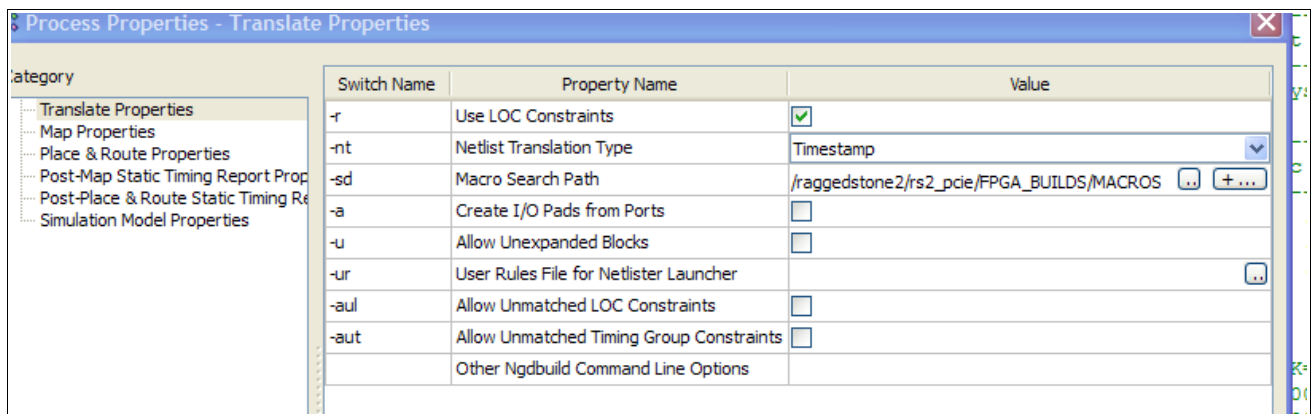
For the PCIE we need to look at the main signals for TLPs, that is received and transmitted data, start, end and frame ready signals. PCIE ChipScope signals.

Having edited the toplevel file we are ready to run Synthesis again.  
 We now have to load **ChipScope\_icon.v** and **ChipScope\_ila.v** into the design.



Loading **ChipScope.v** files.

We also have to tell implement where to find the macros.  
 Right click on Implement and locate the MACROS folder from the Macro Search Path.



Locating the ChipScope MACROS folder.

Once done, the design should successfully build with ChipScope.

### User Constraints File.

The UCF file has been generated from the design example. We retain those constraints and add others which affect the IO Controller inputs and outputs.

```
#
# PCIe Lane 0

INST s6_pcie_i/GT_i/tile0_gtpa1_dual_wrapper_i/gtpa1_dual_i LOC = GTPA1_DUAL_X0Y0;

# diff inputs/outputs
NET pci_exp_txp LOC = B6;
NET pci_exp_txn LOC = A6;
NET pci_exp_rxp LOC = D7;
NET pci_exp_rxn LOC = C7;

# clocks
NET sys_clk_n LOC = B10;
NET sys_clk_p LOC = A10;
NET sys_clk_c PERIOD = 10ns;

NET s6_pcie_i/gt_refclk_out(0) TNM_NET = GT_REFCLK_OUT;
TIMESPEC TS_GT_REFCLK_OUT = PERIOD GT_REFCLK_OUT 10ns HIGH 50 % ;

# output LEDs
NET "LED_COUNT<0>" LOC = "Y21" | IOSTANDARD = LVTTTL | SLEW = SLOW;
NET "LED_USER" LOC = "W20" | IOSTANDARD = LVTTTL | SLEW = SLOW ;

# input switches
NET "DIP<0>" LOC = "Y17" | IOSTANDARD = LVTTTL | PULLDOWN;
NET "DIP<1>" LOC = "AB17" | IOSTANDARD = LVTTTL | PULLDOWN;
NET "DIP<2>" LOC = "Y15" | IOSTANDARD = LVTTTL | PULLDOWN;
NET "DIP<3>" LOC = "AB15" | IOSTANDARD = LVTTTL | PULLDOWN;

NET "PUSH_USER<0>" LOC = "AA14" | IOSTANDARD = LVTTTL | PULLDOWN;
NET "PUSH_USER<1>" LOC = "AB14" | IOSTANDARD = LVTTTL | PULLDOWN;

NET "PB1" LOC = "R7" | IOSTANDARD = LVTTTL | PULLUP;
NET "PB2" LOC = "W4" | IOSTANDARD = LVTTTL | PULLUP;

# other connections.
NET "LED_COUNT<1>" LOC = "AA18" | IOSTANDARD = LVTTTL ;
NET "LED_COUNT<2>" LOC = "AB18" | IOSTANDARD = LVTTTL ;
NET "LED_COUNT<3>" LOC = "V17" | IOSTANDARD = LVTTTL ;
```

Clock and IO constraints.

### Testing the Build.

Having built the project successfully we are ready to plug in the Raggedstone2 board and check that it has been recognised by the PC device manager. We can then install the driver and run the GUI and use ChipScope to see the movement of packets.

### Pre installed Raggedstone2.

If we are downloading to the Raggedstone2 board already installed in the PC, then once the code is downloaded we **turn off** the PC. Resetting the PC doesn't always work as the reset input to the Raggedstone2 isn't connected via the PC.

## Installing the Raggedstone2.

Turn off PC.

Remove the Raggenstone2 on-board oscillator and all loop-back test connectors.

Place jumper J4 to the PC connector position.

Insert board with USB cable attached.

Turn on PC.

## Check Installation has worked.

To check that the Raggedstone2 board has been installed correctly, we can:

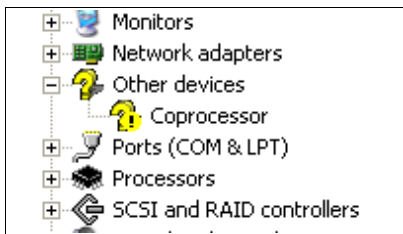
- 1) Check that the Device Manager and check the ID.
- 2) Run PCI32 and search for our processor.

## Running with the Device Manager.

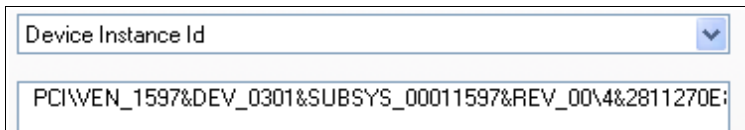
On an XP machine:

**Start>Settings>Control Panel>System.  
Hardware Tab>Device Manager.**

There will be displayed a list of devices.



If we right-click on **Coprocessor> Properties>Details** and select **Device Instance** from the tab, we can see the vendor and device name, as set up in the PCIE core.



## Running PCI32.

This test program explores all PCI devices and downloads the information as a text file.

The link is to Craig's PCI programs.

<http://members.net-tech.com.au/dft0802/downloads.htm>

Having run PCI32 we have get the following information form the **pcie.txt** file.

```
Bus 4 (PCI Express), Device Number 0, Device Function 0
Vendor 1597h Memec Design Services
Device 0301h Unknown
Command 0007h (I/O Access, Memory Access, BusMaster)
Status 0010h (Has Capabilities List, Fast Timing)
Revision 00h, Header Type 00h, Bus Latency Timer 00h
Self test 00h (Self test not supported)
Cache line size 32 Bytes (8 DWords)
PCI Class Processor, type Coprocessor
```

Note that this lists all the information typed in when creating the PCIe core.

This file also contains a lot more information.

### **Installing the Driver.**

Right click Coprocessor.

>**Update Driver**

Windows now appear.

Window 1:

Qn: Can windows connect Windows Update to search for software?

Ans: **No, not this time**

Window 2:

Qn: What do you want wizard to do?

Ans: **Install from a list or specific location**

Window 3:

Qn: Search installation location?

Ans: **Don't search, I will chose the driver to install.**

Window 4:

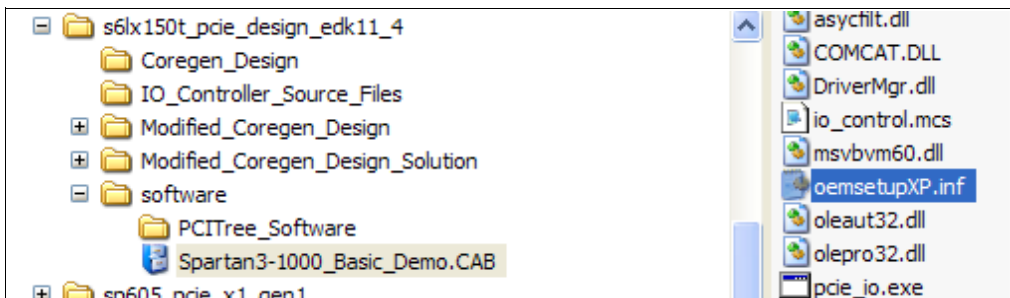
Qn: select Drive driver?

Ans: **Have Disk**

Window 5:

Qn: Locate file?

Ans: **oemsetupXP.inf**

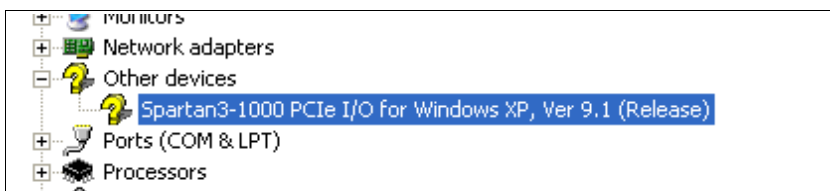


The .CAB file might need unzipping.

We click next and windows queries that the driver hasn't been tested or signed.

This is okay and we **continue anyway**.

The driver is installed.

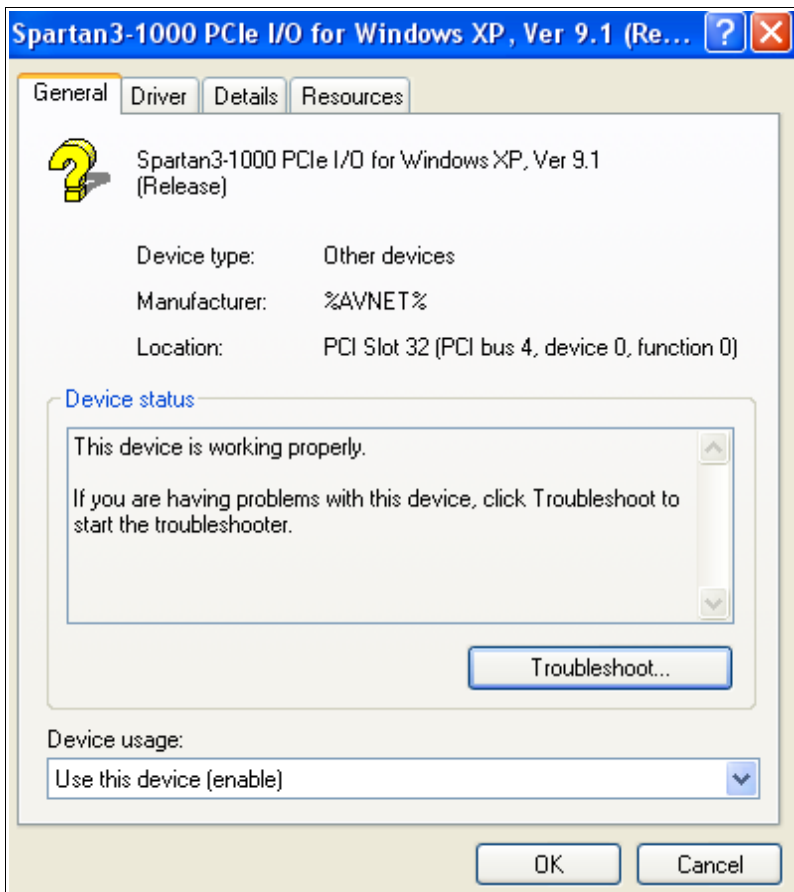


To check it's working properly:

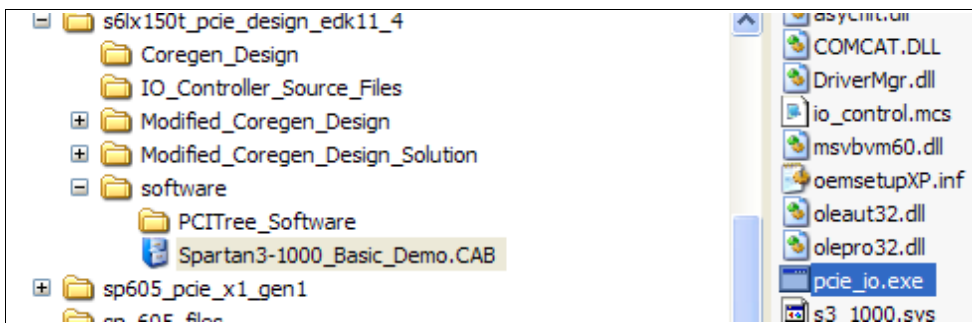
Right-click on the driver in the device manager

Click on **Properties**.

Windows tells us that the device is working properly.

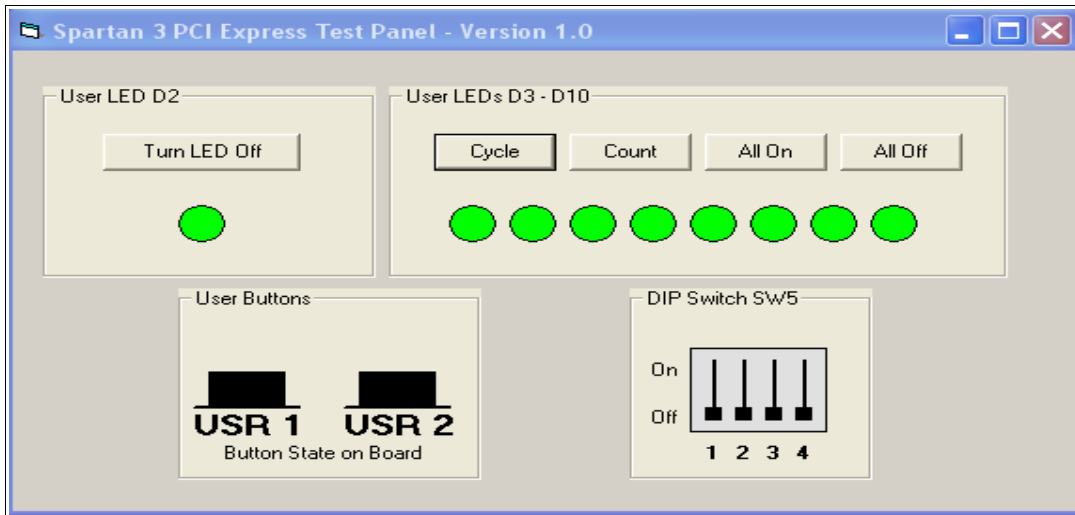


**Testing the PCI Express.**



We run **Spartan3-1000\_Basic\_Demo.CAB/pcie\_io.exe**

This runs the GUI.



## GUI working buttons.

The GUI only affects LED5 and LED4 on Raggedstone2.

CYCLE is not available.

GUI User LED will toggle Raggedstone2 LED4.

GUI ALL On and All Off buttons affect Raggedstone2 LED5.

Count will turn Raggedstone2 LED5 on and off at a ½ second frequency.

## Further Tests.

A connector can also be placed across the inputs to 3v3 to see the User and DIP switch buttons turn on and off. The far right connector Y17 is the easier to access and will show USR1 switch.

USR1 = Y21

USR2 = W20

DIP[3]=AB15

DIP[2]=Y15

DIP[1]= AB17

DIP[0]=Y17

## Raggedstone2 PCI Express test using ChipScope.

With the above GUI, we can use the LEDS as a trigger to catch our PCI express packet or Transaction Layer Packet.

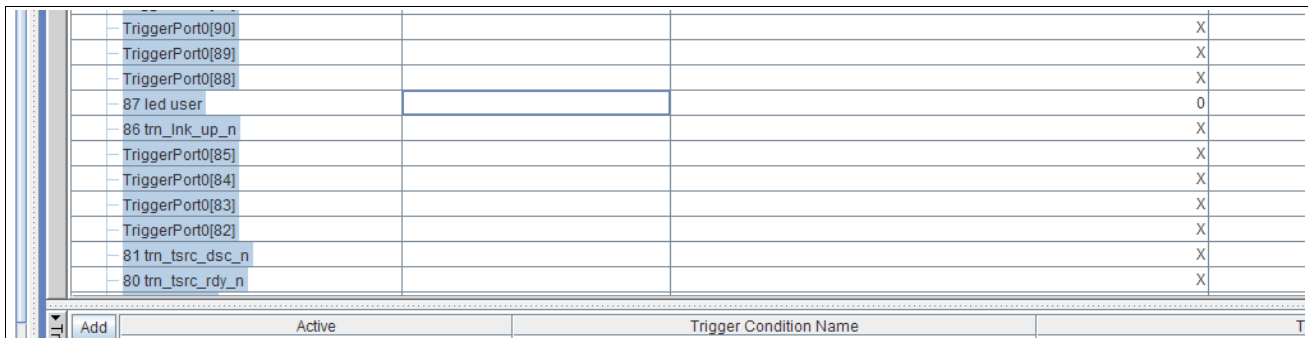
### Run the **ChipScope Analyser.**

We have to rename signals from DataPort to the actual signal name as defined in the top level module. Once done, save the project (name.cpj) and the signal names can be reloaded every time ChipScope is run.

Next, to set the trigger window, open out the trigger port, go down to **port 87 led user.**

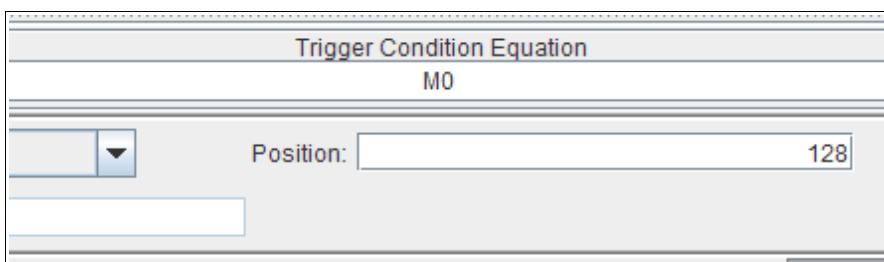
This is the led user input from the GUI. This should be set to 0.

We also set the Position to 128 out of 1024



Signal Name	Active	Trigger Condition Name	T
TriggerPort0[90]			X
TriggerPort0[89]			X
TriggerPort0[88]			X
87 led user			0
86 trn_ink_up_n			X
TriggerPort0[85]			X
TriggerPort0[84]			X
TriggerPort0[83]			X
TriggerPort0[82]			X
81 trn_tsrc_dsc_n			X
80 trn_tsrc_rdy_n			X

87 led user set to 0.



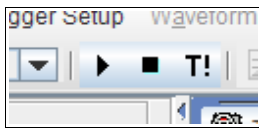
Trigger Condition Equation

M0

Position:

position of trigger set to 128.

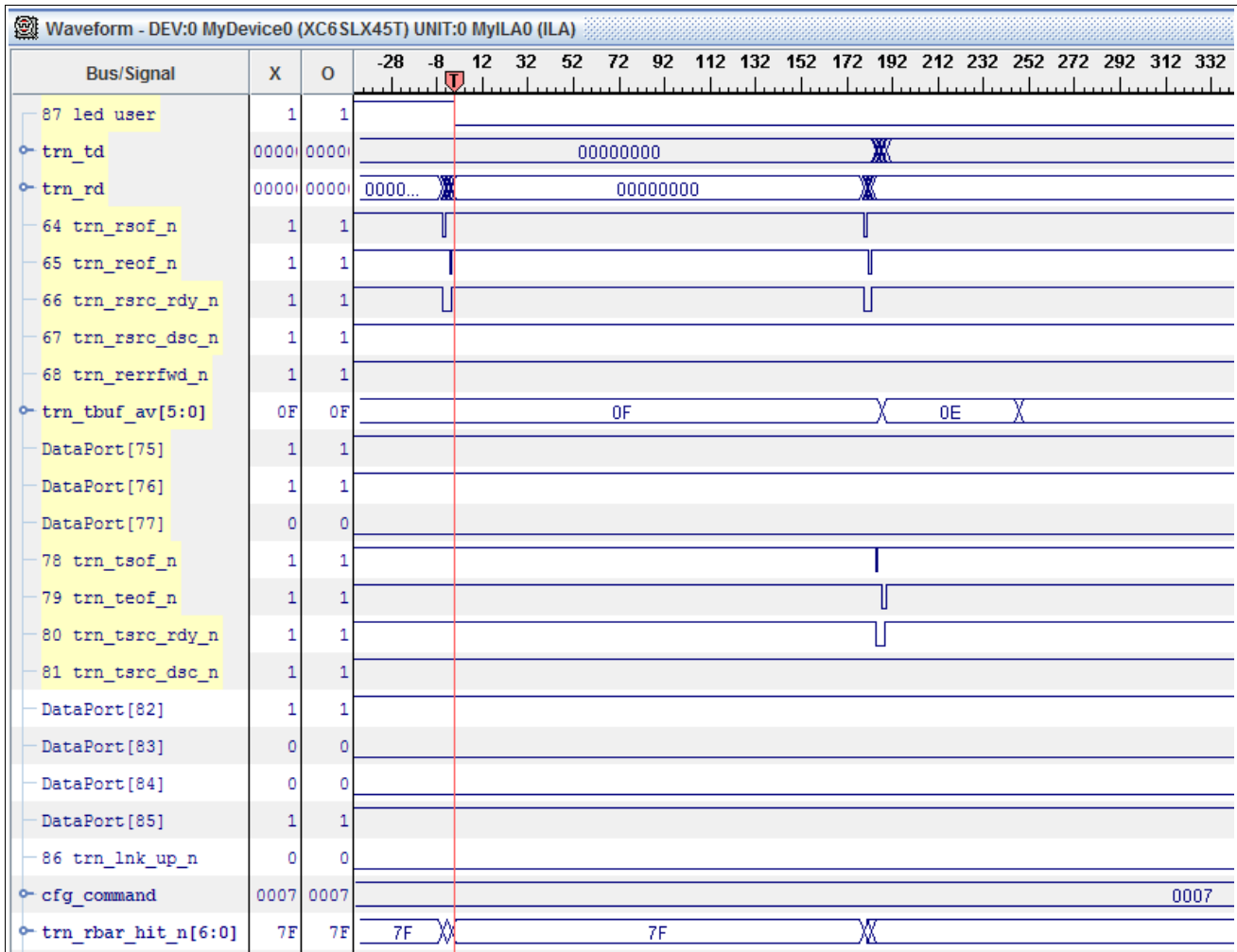
Press the trigger triangle and ChipScope will wait for Led User to be switched.



Trigger triangle.

Go to the GUI and press Led User.

The captured screen should look like the diagram below.



Led User going low.

The Led User can be seen going low and the received data with a start and end of frame.

If we open out the second frame at position 172, this shows the trn\_rd (received data) with a start of frame

trn_rd	Received data
trn_td	Transmitted data
trn_rsof_n	Receive start of frame
trn_reof_n	Receive end of frame.
trn_rsrc_rdy_n	Receive source ready.

We can zoom in and open out trn\_td and note that whenever:

GUI User Led is pressed off (Raggedstone2 LED4 off), trn\_td[24] = 1.

GUI User Led is pressed on (Raggedstone2 LED4 on), trn\_td[24] = 0.