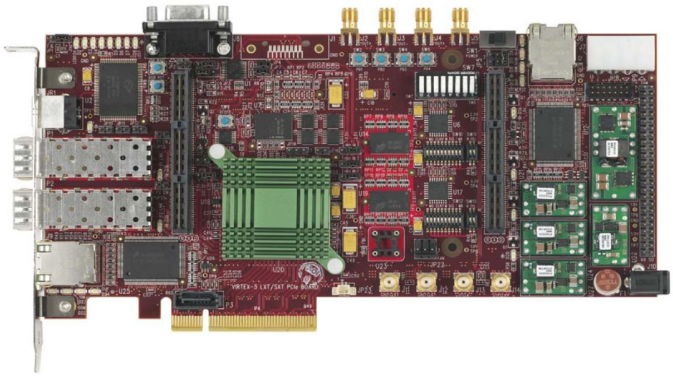
User Guide of the PCIe SG DMA Engine on AVNET Virtex5 Development Board

V1.3

Wenxue Gao

weng.ziti@gmail.com

14 September 2011



Revision	Date	Comment
1.0	20 Aug 2009	Created.
1.1	26 Nov 2009	Correction of some errors.
1.2	24 Aug 2011	Adapted for OpenCores.org.
1.3	14 Sep 2011	Testbench is added.

1. Overview

Figure 1 is the block diagram for the PCIe SG DMA engine in the Virtex5LX110T FPGA logic fabric. DMA engine and PIO engine are parallel established. The Memory Bridge is a module like crossbar switch. TLP manager is to manage the virtual channels on the transaction layer of PCIe. Registers space contains the system registers and other register related to DMA and status peeking. FIFO is internally looped back and the Block RAM (BRAM) module emulates the RAM memory space.

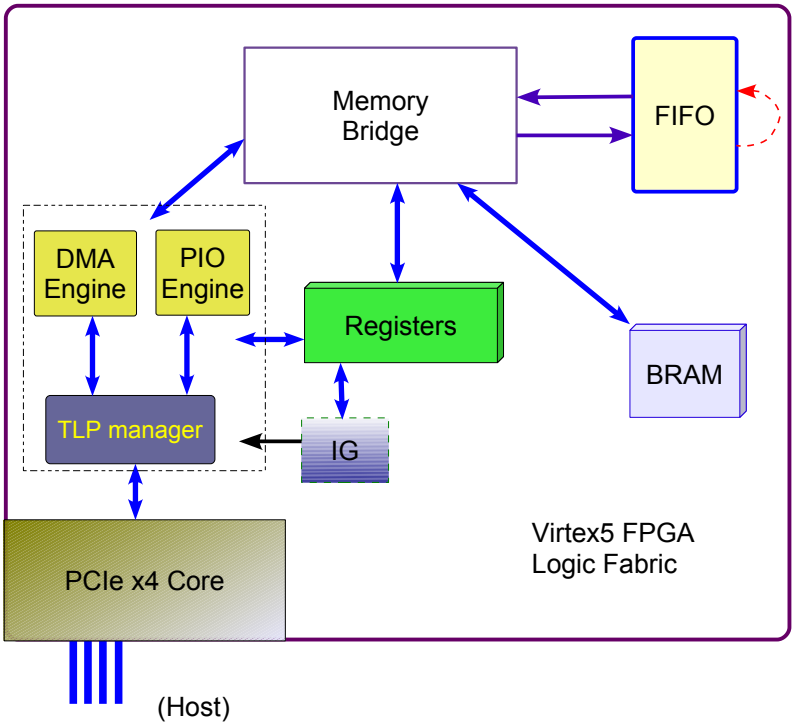


Figure 1 SG DMA in the FPGA logic block diagram

To implement the design, a valid license for the PCIe core from Xilinx Inc. should be available.

In the HDL codes, there are some confusion naming like, `DDR_*`, `Event_*`, etc., which are only legacy of a specific project. These will be improved in the future update.

2. Board Parameters

In terms of hardware layout, the AVNET Virtex5 PCIE development board has following major features,

- 8-lane PCIe
- 2 fibre channels
- 256 MB DDR2 SODIMM (64-bit)
- 64 MB DDR2 SDRAM (32-bit)
- 32 MB flash ROM attached to FPGA
- 2 GbE PHY
- 2 Cypress USB 2.0
- CX4
- SATA host
- RS232
- ...

** FPGA configuration in BPI mode as well as power-up preparation are founded in Appendix A.*

3. Memory Partition

This design holds 3 BAR's, BAR[0], BAR[1] and BAR[2], as its memory space. Registers are accessed via BAR[0], including the system registers, DMA channel registers and some other control and status registers. Block RAM are assigned to BAR[1], including the 32KB dual-port RAM and the write-only 32KB data generator table RAM. BAR[2] contains the FIFO data port, write and read. FIFO control and status registers reside in BAR[0]. BAR[3] to BAR[6] are reserved. All 3 applied BARs are reachable with PIO operation. DMA can only target on BAR[1] and BAR[2].

Registers are divided into groups, as shown in table 1. BAR[0] and BAR[1] spaces are 4-byte (DW) aligned, i.e. lowest 2 bits of addresses are taken as "00". BAR[2] is 8-byte (QW) aligned and its address offset is arbitrary.

Table 1 Address Assignment

Name	Offset	R/W	BAR
System section			
Design ID Register (DID)	0x0000	RO	0
Interrupt Status Register (ISR)	0x0008	RO	0
Interrupt Enable Register (IER)	0x0010	R + W	0
General Error Register (GER)	0x0018	RO	0
General Status Register (GSR)	0x0020	RO	0

General Control Register (GCR)	0x0028	R + W	0
Upstream DMA channel (Channel#1 in MPRACE)			
Peripheral Address high-DW	0x002C	R + W	0
Peripheral Address low-DW	0x0030	R + W	0
Host address high-DW	0x0034	R + W	0
Host address low-DW	0x0038	R + W	0
Next Buffer Descriptor Address high-DW	0x003C	R + W	0
Next Buffer Descriptor Address low-DW	0x0040	R + W	0
Length in bytes	0x0044	R + W	0
Control	0x0048	R + W	0
Status	0x004C	RO	0
Downstream DMA channel (Channel#0 in MPRACE)			
Peripheral Address high-DW	0x0050	R + W	0
Peripheral Address low-DW	0x0054	R + W	0
Host address high-DW	0x0058	R + W	0
Host address low-DW	0x005C	R + W	0
Next Buffer Descriptor Address high-DW	0x0060	R + W	0
Next Buffer Descriptor Address low-DW	0x0064	R + W	0
Length in bytes	0x0068	R + W	0
Control	0x006C	R + W	0
Status	0x0070	RO	0
PIO Path Controls			
MRd channel control	0x0074	WO	0
PCIe transaction layer Tx module control	0x0078	WO	0
ICAP (*reserved*)			
ICAP port write	0x007C	W	0
ICAP port read	0x007C	R	0
Interrupt Generation (*See Interrupt Generator chapter*)			
Interrupt Generation Control (IGC)	0x0080	R + W	0
Interrupt Generation Latency (IGL)	0x0084	R + W	0
Interrupt Generation On Statistic (IGN_ON)	0x0088	RO	0
Interrupt Generation Off Statistic (IGN_OFF)	0x008C	RO	0
FIFO Control and Status			
Control	0x0090	W	0
Status	0x0090	R	0
DMA Actual Transferred			

Upstream transferred byte count	0x0094	RO	0
Downstream transferred byte count	0x0098	RO	0
Large memory (1 MB)			
Block RAM (32 KB)	0x08000 ~ 0x0FFFC	R + W	1
Other regions reserved			*1*
Event FIFO Data Interface			
Read	0x0000	R	2
Write	0x0000	W	2

Notes for table 1,

R: Readable.
W: Writeable.
R+W: Readable and writeable.
RO: Read-only. Write no effect.
WO: Write-only. Read as zero.

3.1. Register definition — BAR[0]

Following are some registers definition. Greyed bits are reserved, which are read as zero and should avoid writes with non-zero values.

3.1.1 Design ID (+0x0000)

31 ~ 24	23 ~ 16	15 ~ 12	11 ~ 0
Design version	Design major revision	Author Code	Design minor revision

3.1.2 Interrupt Status Register (+0x0008)

31 ~ 6	5	4	3	2	1	0
	Downstream Time-out	Upstream Time-out		Interrupt Generator	Downstream DMA Done	Upstream DMA Done

** In MPRACE library, upstream is channel #1 and downstream #0.*

3.1.3 Interrupt Enable Register (+0x0010)

31 ~ 8	7	6	5	4	3	2	1	0
	CTL	DAQ	Downstream Time-out	Upstream Time-out		Interrupt Generator	Downstream DMA Done	Upstream DMA Done

** In MPRACE library, upstream is channel #1 and downstream #0.*

3.1.4 General Error Register (+0x0018)

31 ~ 20	20	19	18	17 ~ 0
	Event Buffer Overflow	Event Buffer Time-out	Tx Time-out	

3.1.5 General Status Register (+0x0020)

31 ~ 16	15 ~ 10	9 ~ 8	7~6	5	4	3 ~ 0
	PCIe Link Width		DCB Link Active[1:0]	DG Available	ICAP Busy	

** PCIe maximum link width ——*

'B000000 : *Reserved*
'B000001 : x1
'B000010 : x2
'B000100: x4
'B001000: x8
'B001100 : x12
'B010000 : x16
'B100000 : x32

3.1.6 General Control Register (+0x0028)

31 ~ 14	13 ~ 12	11	10 ~ 8	7 ~ 3	2 ~ 0
	Upstream MWr Attr.		Upstream MWr TC		Msg. Routing

** It is highly recommended that no write be made to General Control Register*.

3.1.7 MRd Channel Control Register (+0x0074)

31 ~ 8	7 ~ 0
	Reset (0x0A)

Write with 0x0A resets Mrd channel (also clears Event Buffer overflow).

3.1.8 PCIe transaction layer Tx module Control Register (+0x0078)

31 ~ 8	7 ~ 0
	Reset (0x0A)

Write with 0x0A resets PCIe transaction layer Tx module.

3.1.9 Event Buffer Control Register (+0x0090, write)

31 ~ 8	7 ~ 0
	Reset (0x0A)

Write with 0x0A resets the Event Buffer. During reset, the Event Buffer Status Register is read as zero.

3.1.10 Event Buffer Status Register (+0x0090, read)

31 ~ 26	25 ~ 3	2	1	0
	Data count in QW		Almost full	Empty

** A zero value read from this register is possible for the FIFO and means "unavailable", most probably it is being reset.*

3.2. Block RAM — BAR[1]

Block RAM size is 32 KB, 64-bit data bus times 4096 items. It is accessed in 32-bit DW units. It can be used to test DMA functions targeted on conventional RAM memory, with address incremented.

Data generator table RAM is also 64-bit data bus and 4096 items; written in 32-bit DW units. For this version, the data generator's table RAM content can not be read.

3.3. FIFO — BAR[2]

The current version FIFO is built on a built-in FIFO primitive inside the FPGA, 64-bit data bus width and 128 KB in size. Externally this buffer is treated exactly as an asynchronous dual-port FIFO. Its empty and almost full flags are connected to the Event Buffer Status Register (+0x0090). If an empty FIFO is read, the time-out might happen. Time-out can be cleared by a reset (+0x0A) written to the Event Buffer Control register (+0x0090). Overwrite is monitored in GER[20] and is cleared either by an Event Buffer reset or an MRd channel reset.

Event Buffer can be accessed with DMA or PIO. DMA size is exactly what the descriptor defines. However, the PIO TLP (Transaction Layer Packet) payload is always in 32-bit wide in our system, the higher 32 bits in the Event Buffer will get lost via PIO. In this sense, PIO and DMA transactions shall not be mixed for the same section of data transfer.

4. DMA Functions

4.1. Scatter-Gather DMA

A DMA transfer is initiated via writing the first descriptor to corresponding DMA channel

registers. The DMA engine supports multiple descriptors, which provides flexibility to applications in USER memory mode.

In multiple-descriptor DMA, the next descriptor (a group of parameters) is requested according to the Next Buffer Descriptor Address in current DMA descriptor after the current descriptor is executed. Then the DMA engine will execute that new descriptor. This process will continue until a descriptor with the END bit asserted in the Control parameter is done.

4.2. Descriptor description

Table 2 lists the descriptor definition for a DMA channel.

Table 2 DMA channel registers / Buffer descriptor

Name	R/W	Address Offset	+3								+2								+1								+0							
			31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Peri_Addr_H	R+W	+0x00	Higher 32-bit of DMA Peripheral address																															
Peri_Addr_L	R+W	+0x04	Lower 32-bit of DMA Peripheral address																															
Host_Addr_H	R+W	+0x08	Higher 32-bit of DMA Host address																															
Host_Addr_L	R+W	+0x0C	Lower 32-bit of DMA Host address																															
Next_BDA_H	R+W	+0x10	Higher 32-bit of next descriptor address																															
Next_BDA_L	R+W	+0x14	Lower 32-bit of next descriptor address																															
Length	R+W	+0x18	DMA transaction byte count																															
Control	R+W	+0x1C							V	Last				UPA		BN2	BN1	BN0	AInc							End	Rst (read as 0)							
Status	R	+0x20																												TO			Busy	Done

BN[2:0]: Encoded BAR Number. Default is 000.

V: Valid Control. A special bit for DMA channels. If set, the DMA engine uses this Control word; else, the DMA engine takes the previous Control word of that channel for current transaction. Even the reset command must also set this bit. Default value is '0'. Note: for the first descriptor this bit must be asserted.

Last: If set, the current descriptor is the last one for the current DMA chain. After the descriptor is processed, the DMA engine stops. Default value is '0'.

UPA: Use PA (Peripheral Address) from each descriptor, otherwise, use calculated PA. Default value is '0'. But for the initial descriptor, PA is always used, therefore this bit can be of any value for the first descriptor.

AInc: Address increments. If set, the address of the endpoint side increases by four for

every double word transfer. Default value is '1'.

End: If set, the DMA engine is paused after this descriptor is processed. This means, the engine can be resumed afterwards. Default value is '0'.

Rst: Channel can be reset by write "0x0200000A" to the corresponding Channel Control register. These bits return always 0x0 by reading.

TO: Time-out signal. If asserted, this bit indicates that a time-out event has occurred during the DMA operation. Default value is '0'.

Busy: DMA engine is running. Default value is '0'.

Done: DMA engine is already finished. Default value is '0'. Once asserted, this bit never changes until a channel reset command comes.

4.3. **DMA Commands**

A write to a DMA channel control register with the valid bit asserted is a DMA command. Bits like Last, UPA and AInc are the parameters of the command. Rst bits and End bit are the type of the command. Concerning the bits V, End and Rst, possible types of commands are listed in table 3.

Table 3 DMA commands

Command	V	End	Rst[7:0]	Description
Reset	1	x	0x0A	Reset the DMA channel state.
Start	1	0	0	Start/Resume DMA transaction.
Stop	1	1	0	Pause the current running DMA transaction, if any.
Redo	0	x	x	Repeat the last DMA command, which had the V bit set.

x: don't care.

4.4. **DMA Status**

Table 4 explains the DMA status.

Table 4 DMA statuses

State	Busy	Done	Description
Idle	0	0	DMA channel is idle, ready to start new DMA transaction.
Busy	1	0	DMA is running.
Done	0	1	A DMA is already finished.
Unreset	1	1	Abnormal state. A previous DMA is finished but state is not reset before a second DMA is started.

4.5. **DMA Chain**

For a DMA chain consisting of multiple descriptors, subsequent descriptors are in the same order and format as the initial one, yet resident in host memory. To do a chained DMA, the upper-level application should get the next descriptors prepared in the host memory before it initiates the first DMA by writing the first descriptor in the peripheral. The DMA engine goes along the chain until the one with Last=1 is executed, after which the DMA finishes.

Figure 3 illustrates a non-loop DMA consisting of multiple descriptors and figure 4 illustrates a looped DMA consisting of multiple descriptors. If the descriptor #0 in figure 3 is labeled with `Last=1`, it will turn out to be a single-descriptor DMA transaction.

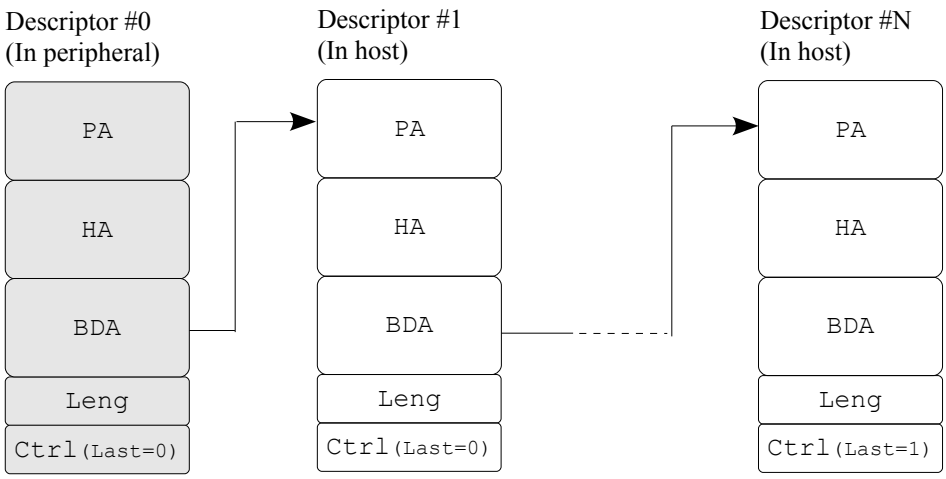


Figure 3 DMA Chain illustration – not looped

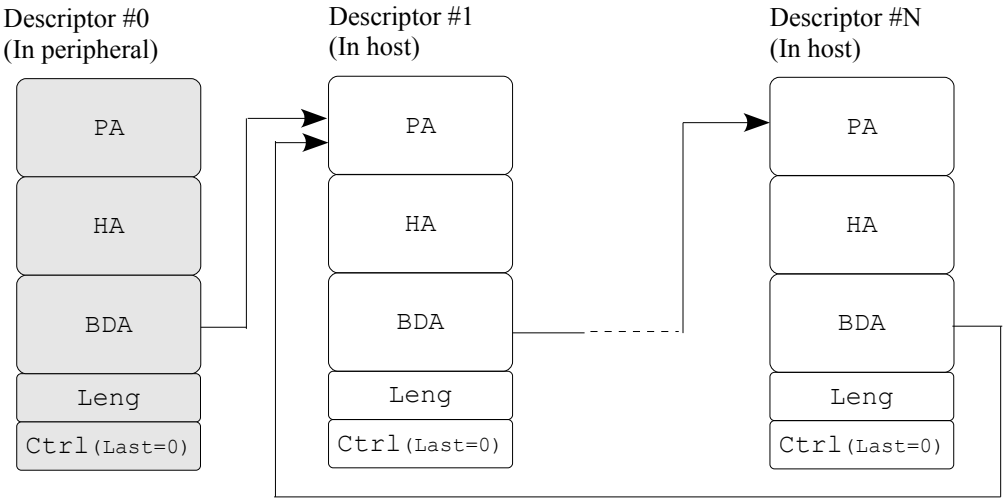


Figure 4 DMA Chain illustration – looped

5. Interrupt Generator

IGC (Interrupt Generator Control) and IGL (Interrupt Generator Latency) are used for the moment as the Interrupt Generator control and parameter registers. The corresponding bit in the Interrupt Status register (ISR) is bit 2.

A unit in IGL register is calculated as 8 nS in ABB2 4-lane version. The value in IGL is the delay between two interrupts generated. To start the Interrupt Generator, a non-zero value is necessary to the IGL register. Thus, a zero written to the IGL register serves as a pause command, which does not reset the statistics registers but makes the Interrupt Generator stop generating interrupts. Resuming the paused interrupt issuing is done by writing a non-zero value to the IGL register. To emulate the interrupt process service, another feature word ("0x00F0") is needed to be written into the IGC register, which clear one interrupt every time.

An example procedure is given below.

```
*(0xED000080) = 0x0A;          // Reset Interrupt Generator
*(0xED000010) = 0x0004;        // Enable interrupt generation
*(0xED000084) = 0x3000;        // Set Interrupt Generator Latency to 98304 ns,
                               //   and trigger it run
```

```
// Interrupt service program
void Int_Service ( )
{
 int Int_Index = *(0xED000008);
 if (Int_Index & 0x0004)          // Interrupt(s) from the Int Generator come
   {
    ... ...                       // Servicing

    *(0xED000080) = 0x00F0;       // Clear one interrupt
   }
}
```

Afterwards, the number of assert interrupts and that of the deassert interrupts can be obtained from the IGN_ON (+0x0088) and IGN_OFF (+0x008C), respectively. Their values give information of the status of the interrupt service performance. These two statistic registers and the Interrupt Generation register are reset by a reset word (0x0A) written to the IGC register (+0x0080).

6. Testbench

Simulation is provided in Verilog HDL, `tf64_pcie_trn.v`. In this simulation environment, 3 modules are instantiated and connected, (1) tlpControl, (2) bram_Control and (3) FIFO_wrapper.

Figure 5 shows the connection inside the testbench.

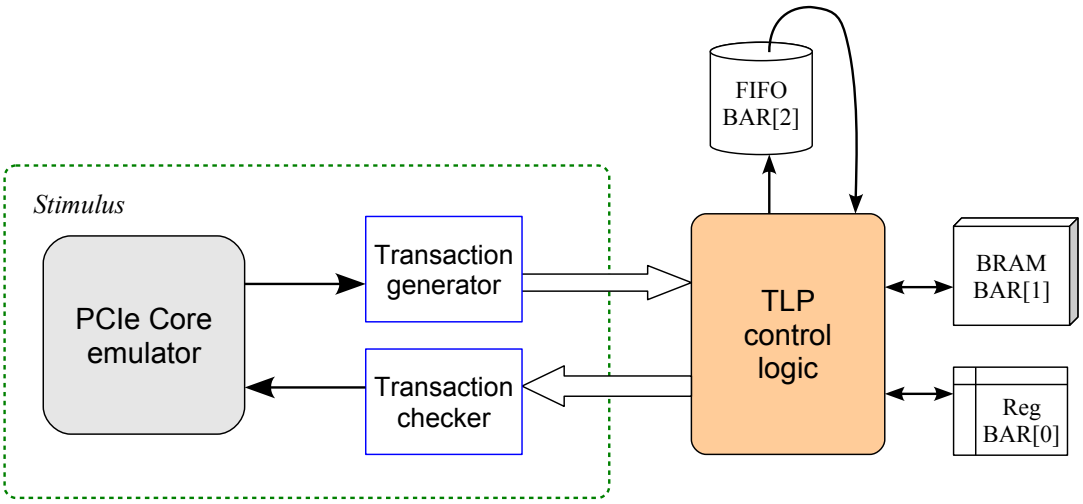


Figure 5 Testbench block diagram

5 types of transactions are simulated, as listed in table 5. Actually the testbench is trying to emulate the behaviour of the PCIe Core in sending and receiving TLPs. In terms of BAR[2] FIFO PIO transactions, note that only the lower 32 bits are accessible. To demonstrate the SG DMA in multiple-descriptor mode, the BAR[2] DMA simulation is done in two descriptors.

Table 5 DMA simulation coverage

	PIO	DMA
BAR[0]	Yes	NO
BAR[1]	Yes	Yes
BAR[2]	Yes *	Yes *

Interrupt is enabled and DMA status polling is simulated. Throttling at Rx and Tx ports can be activated by initializing `Rx_No_Flow_Control` and `Tx_No_Flow_Control` as 0. The simulation runs about 10 μs.

The TLP sending at Rx is made in task `TLP_Feed_Rx`, which is modular for further expansion of the testbench. Rx and Tx TLP format checking is also integrated, so that the upgrade upon the testbench brings no fatal errors.

7. Resource report from ISE 12.4 MAP

```
Release 12.4 Map M.81d (nt64)
Xilinx Mapping Report File for Design 'v5pcieDMA'

Design Information
------------------
Command Line   : map -intstyle ise -p xc5vlx110t-ff1136-1 -w -logic_opt on -ol
high -xe n -t 1 -register_duplication on -global_opt speed -retiming on
-equivalent_register_removal off -mt 2 -cm speed -ir off -ignore_keep_hierarchy
-pr off -lc off -power off -o v5pcieDMA_map.ncd v5pcieDMA.ngd v5pcieDMA.pcf
Target Device  : xc5vlx110t
Target Package : ff1136
Target Speed   : -1
Mapper Version : virtex5 -- $Revision: 1.52.76.2 $
Mapped Date    : Thu Sep 01 12:09:11 2011

Design Summary
--------------
Number of errors:      0
Number of warnings:  171
Slice Logic Utilization:
  Number of Slice Registers:                11,167 out of  69,120   16%
    Number used as Flip Flops:              11,163
    Number used as Latches:                      4
  Number of Slice LUTs:                     12,683 out of  69,120   18%
    Number used as logic:                   11,948 out of  69,120   17%
      Number using O6 output only:          11,125
      Number using O5 output only:             509
      Number using O5 and O6:                  314
    Number used as Memory:                     692 out of  17,920    3%
      Number used as Dual Port RAM:             72
        Number using O6 output only:             8
        Number using O5 and O6:                 64
      Number used as Shift Register:           620
        Number using O6 output only:           620
    Number used as exclusive route-thru:        43
  Number of route-thrus:                       578
    Number using O6 output only:               549
    Number using O5 output only:                26
    Number using O5 and O6:                      3

Slice Logic Distribution:
  Number of occupied Slices:                 5,296 out of  17,280   30%
  Number of LUT Flip Flop pairs used:       15,140
    Number with an unused Flip Flop:         3,973 out of  15,140   26%
    Number with an unused LUT:               2,457 out of  15,140   16%
    Number of fully used LUT-FF pairs:       8,710 out of  15,140   57%
    Number of unique control sets:             314
    Number of slice register sites lost
      to control set restrictions:             373 out of  69,120    1%

  A LUT Flip Flop pair for this architecture represents one LUT paired with
  one Flip Flop within a slice.  A control set is a unique combination of
  clock, reset, set, and enable signals for a registered element.
  The Slice Logic Distribution report is not meaningful if the design is
```

```
  over-mapped for a non-slice resource or if Placement fails.
  OVERMAPPING of BRAM resources should be ignored if the design is
  over-mapped for a non-BRAM resource or if placement fails.

IO Utilization:
  Number of bonded IOBs:                        10 out of     640    1%
    Number of LOCed IOBs:                       10 out of      10  100%
    Number of bonded IPADs:                     10
    Number of bonded OPADs:                      8

Specific Feature Utilization:
  Number of BlockRAM/FIFO:                      49 out of     148   33%
    Number using BlockRAM only:                 46
    Number using FIFO only:                      3
    Total primitives used:
      Number of 36k BlockRAM used:              46
      Number of 36k FIFO used:                   3
    Total Memory used (KB):                 1,764 out of   5,328   33%
  Number of BUFG/BUFGCTRLs:                      3 out of      32    9%
    Number used as BUFGs:                        3
  Number of BUFDSs:                              1 out of       8   12%
  Number of GTP_DUALs:                           2 out of       8   25%
    Number of LOCed GTP_DUALs:                   2 out of       2  100%
  Number of PCIEs:                               1 out of       1  100%
  Number of PLL_ADVs:                            1 out of       6   16%

Average Fanout of Non-Clock Nets:                4.12

Peak Memory Usage:  987 MB
Total REAL time to MAP completion:  16 mins 10 secs
Total CPU time to MAP completion (all processors):   17 mins 4 secs
```

Appendix A

AVNET V5LX110T Development Board Start-up

A.0 Xilinx Software for Programming

ISE10.1 (or later versions) needs to be installed to use iMPACT 10.1 configuring the flash on the Avnet V5LX110T development board.

A.1 Setting Up the Board

Figure A-1 shows the connector and jumper locations on the AVNET Virtex-5 LX110T development board.

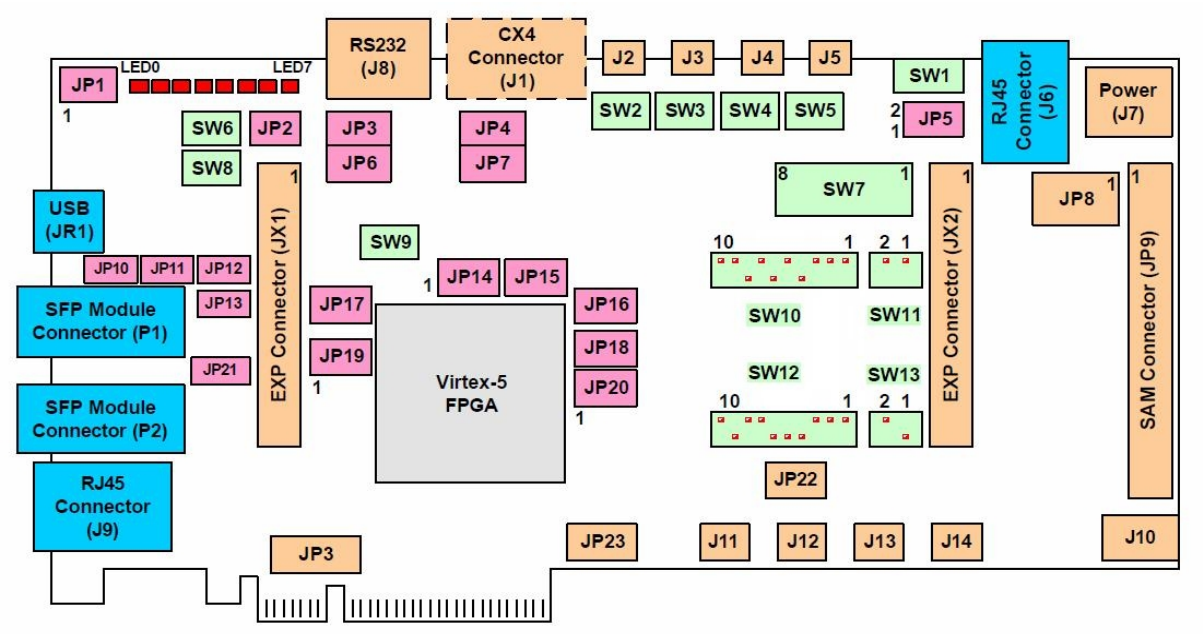


Figure A-1. Connectors and jumpers

Before plugging the board into PCIe slot, please follow the steps shown below to set the jumpers.

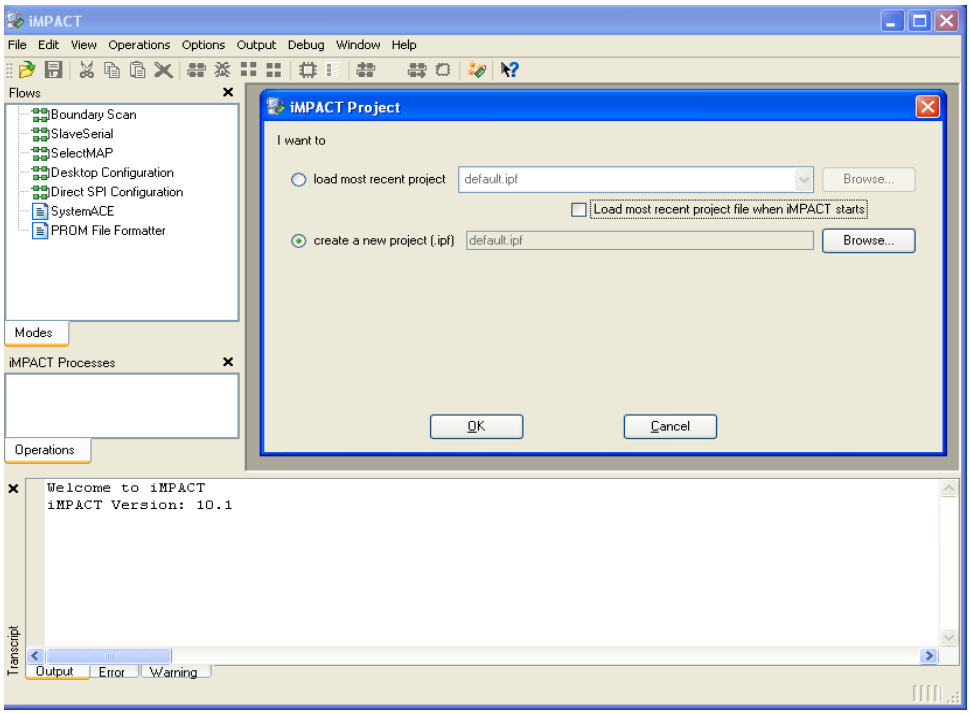
1). Set the **SW1** to ON position
2). Install a jumper on **JP5** pins 3-4 (the middle pair)
3). Install a jumper on **JP4**
4). Install a jumper on **JP7**
5). Install a jumper on **JP3** pins 2-3
6). Install a jumper on **JP6** pins 1-2
7). Install a jumper on **JP1** pins 1-2

8). Install a jumper on **JP16** pins 2-3
9). Install a jumper on **JP18** pins 2-3
10). Install a jumper on **JP20** pins 2-3
11). Install a jumper on **JP19** pins 2-3
12). Connect Xilinx JTAG cable to **JP8** and the parallel/USB port of the PC
13). Connect a proper power plug to **J7**

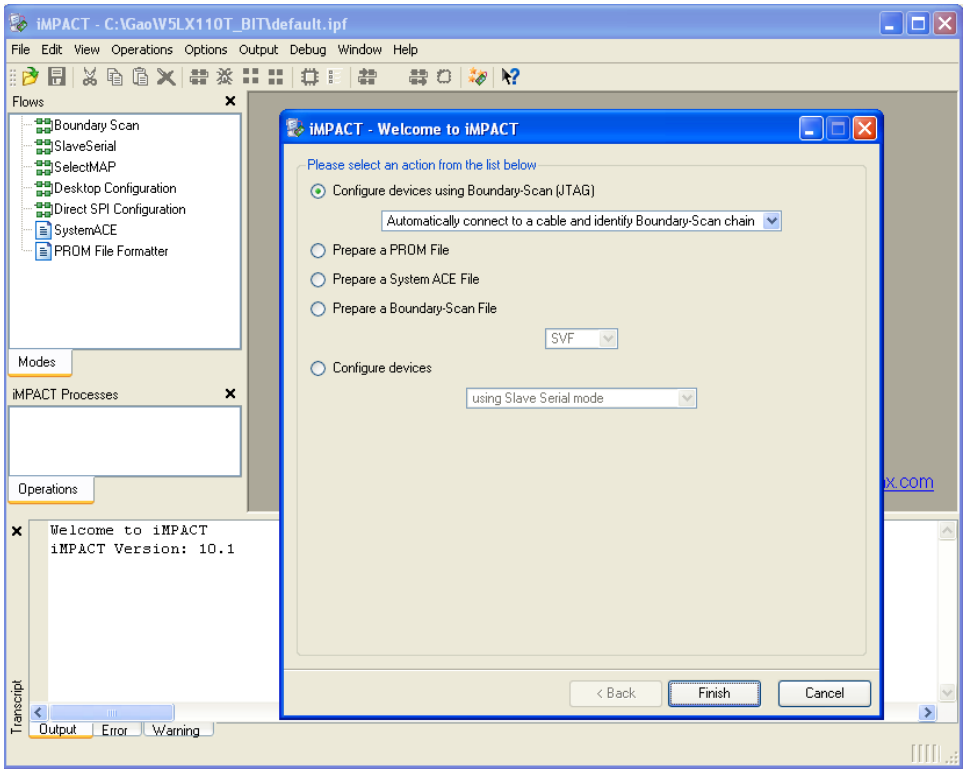
A.2 Programming the flash

After steps listed in A.1 are finished, plug the Avnet V5LX110T development board into the PCIe x4/x8 slot of the PC and turn the PC on. Meanwhile, unzip the zipped file on another computer, which has ISE10.1 (or later) and contains the .MCS file, open iMPACT to configure the Flash with the .MCS file, as followed.

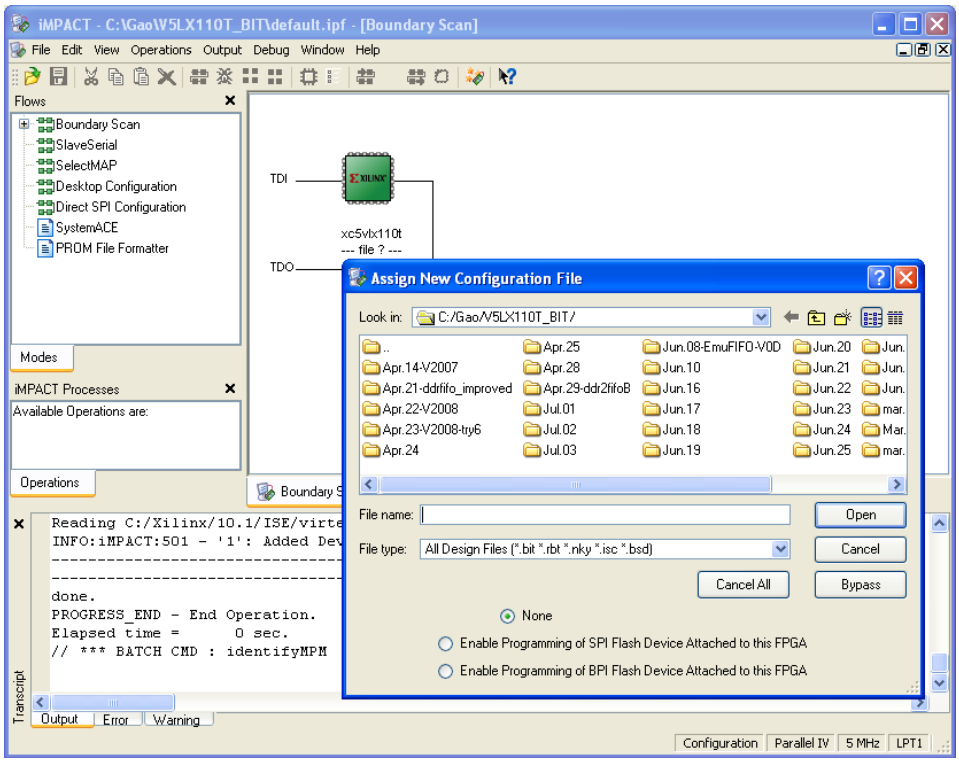
1) Click up the iMPACT 10.1 from the ISE →accessories→iMPACT. Create a new project (.ipf), click "OK".



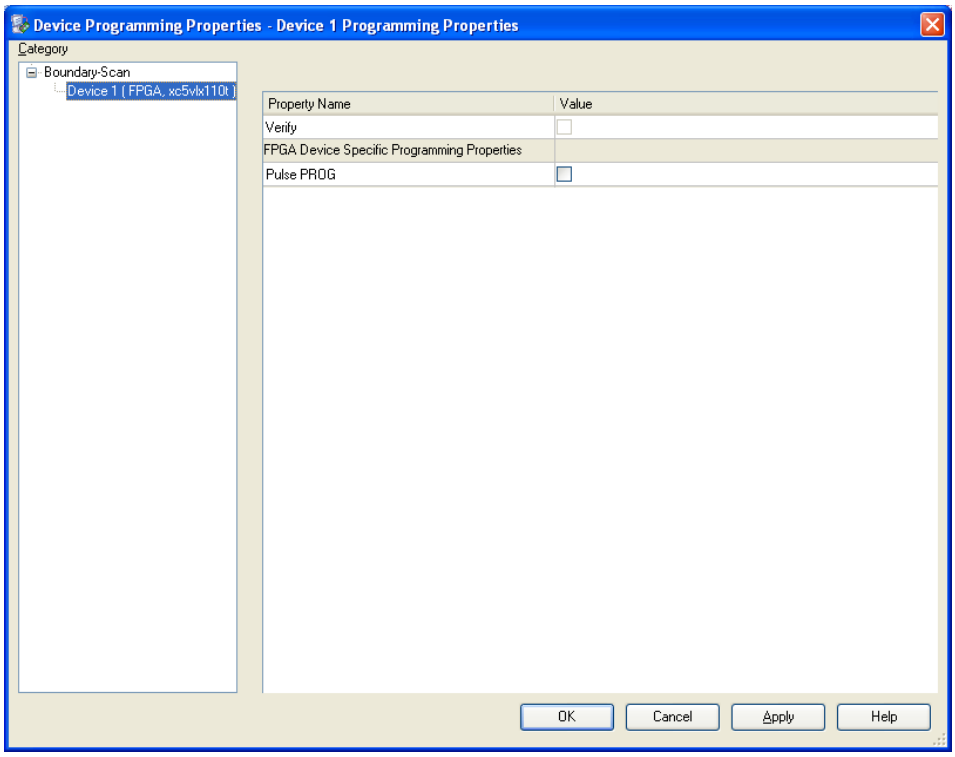
2) Configure devices using Boundary Scan (JTAG), "Finish".



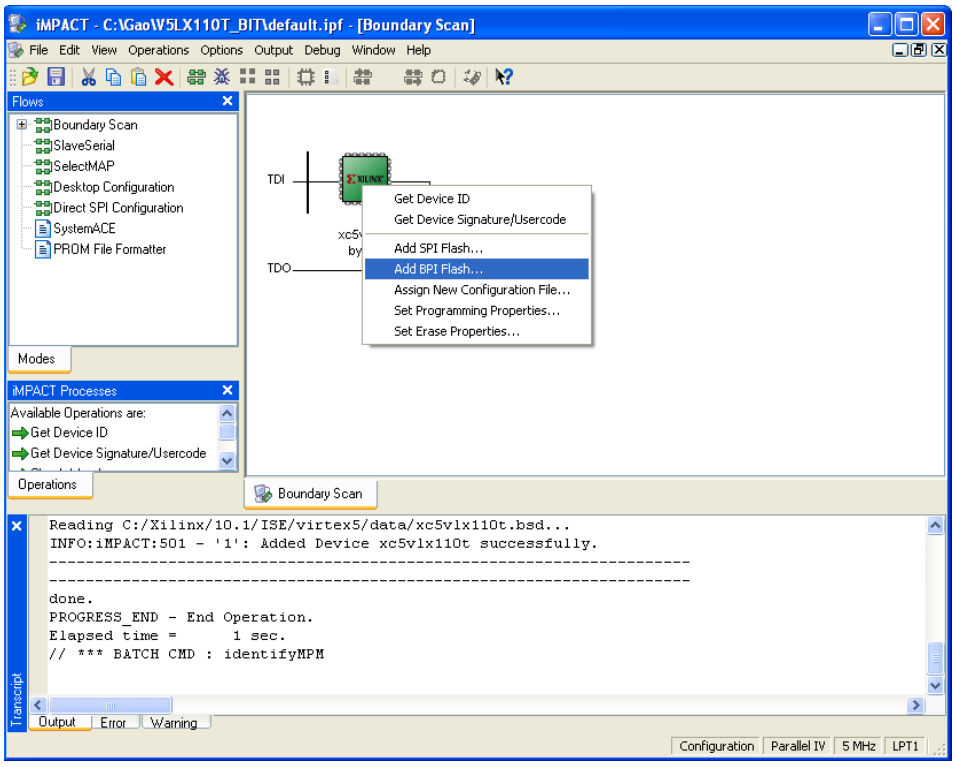
3) click “Bypass”



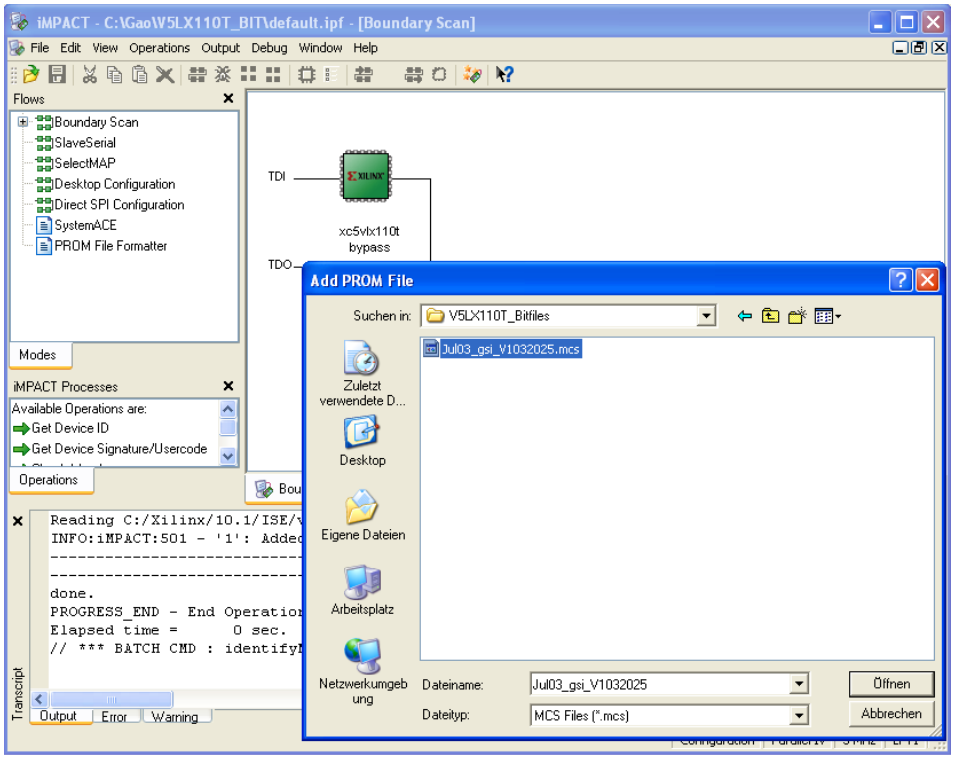
4) Click “OK”.



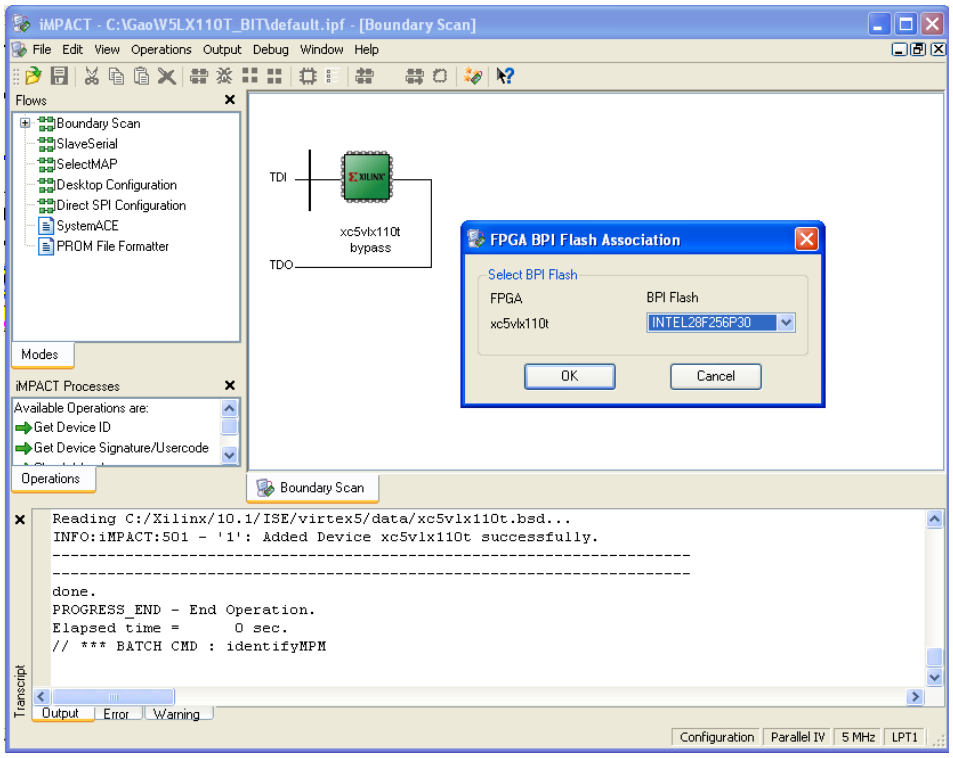
5) Right click the FPGA device icon, “Add BPI Flash”.



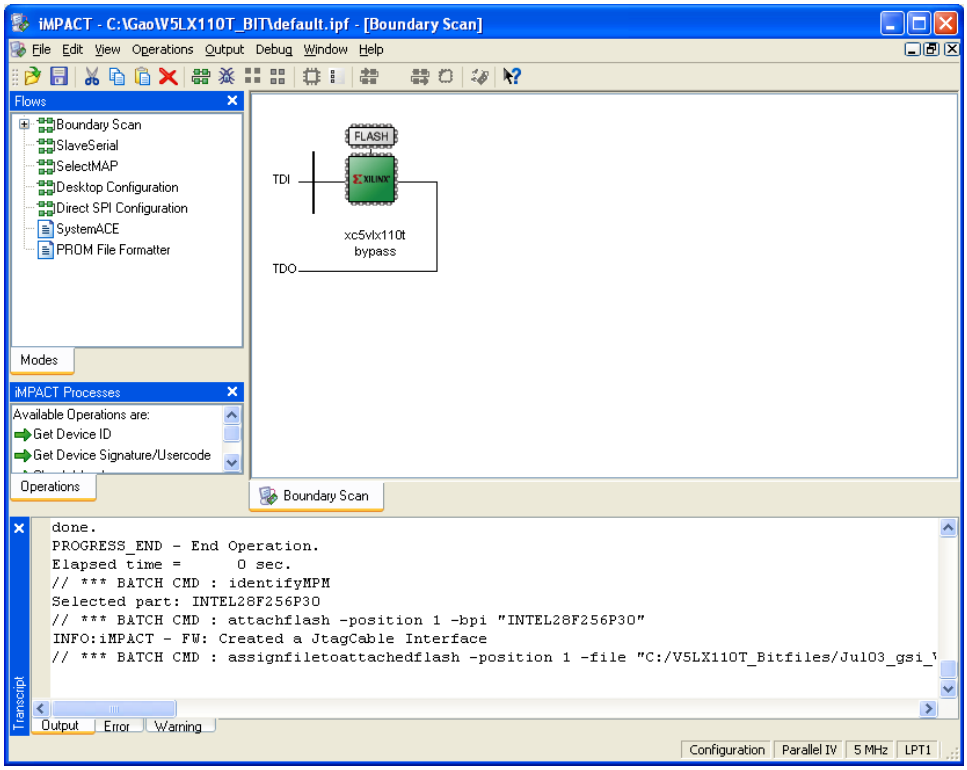
6) Select the unzipped .MCS file. “Open”.



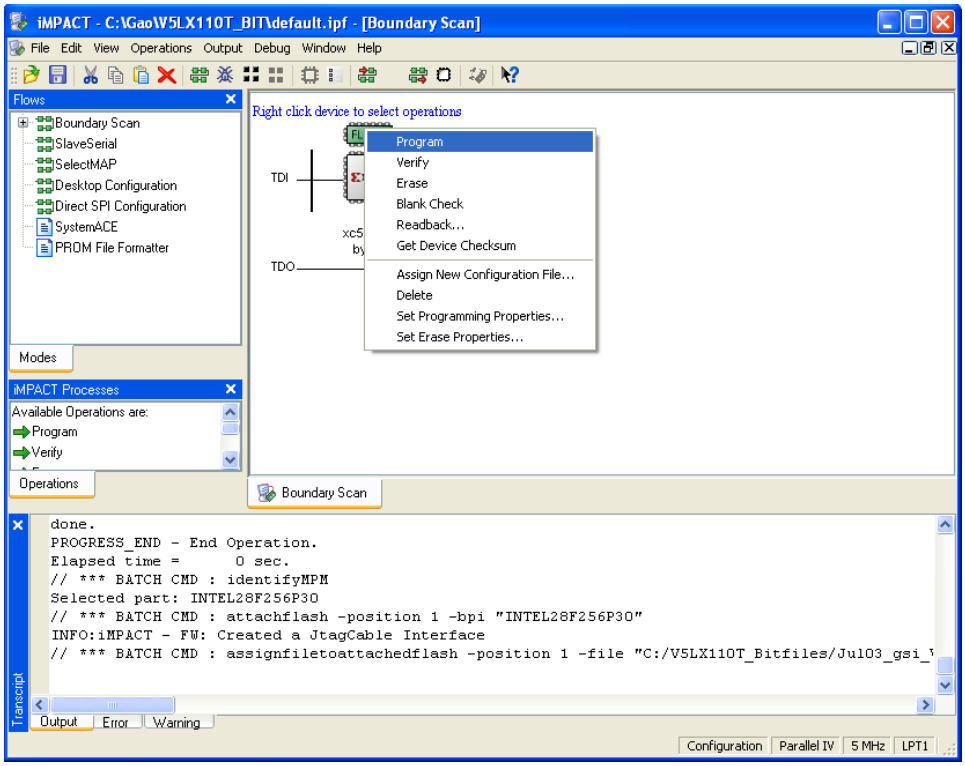
7) Select BPI Flash typed **`INTEL28F256P30`**. “OK”.



8) Flash is now attached to FPGA.



9) Right click the Flash icon, select “Program”.



10) After several minutes of patient wait, the flash is successfully programmed.