PPC Reference Design

Version 1.1
August 2002

Table of Contents

Figures

Tables

1 Overview

The PPC Reference Design is intended to demonstrate some of the features and capabilities of the PowerPC processor within the Virtex-II Pro™ FPGA. An IBM Core Connect™ infrastructure is used to connect the PPC core to numerous peripherals using the Processor Local Bus (PLB), On-Chip Peripheral Bus (OPB), and Device Control Register (DCR) buses to build a complete system. This document describes the PPC Reference Design and provides information about how the system is organized and implemented. Several of the peripheral functions included in the design (i.e. SRAM and Flash) are not actually functioning without the addition of the optional Memec Design P160 Communications Module. The document also discusses verification methodologies including software-driven and bus-model-driven simulations. A complete design cycle incorporating simulation, synthesis, and FPGA implementation is described. The information presented introduces many aspects of the PPC Reference System, but the user should refer to additional documentations contained in the Xilinx Virtex-II Pro Developers Kit for more detailed information about the software, tools, peripherals, interface protocols, and capabilities of the FPGA.

2 Block Diagram

The following figure provides a high-level view of the PPC Reference Design. This design demonstrates a system that uses devices that are connected to the PLB, OPB, and DCR buses. The PLB is primarily used to interface to the devices with higher bandwidth requirements (such as high-speed memory devices), while the OPB is dedicated to low-speed I/O devices such as UART, LCD, and GPIO. The OPB offers a less complex protocol relative to the PLB, making it easier to design peripherals that do not require high performance. The OPB also has the advantage of supporting a greater number of devices, for systems requiring many low-speed I/O devices. The DCR bus is primarily used to interface to simple I/O devices that contain a few data, control, and status registers. Refer to the PLB, OPB, and DCR CoreConnect Architecture Specifications for more information.

The Virtex-II Pro development board is used to test this reference design. This board is based on the Xilinx XC2VP4 FPGA and provides a set of features that make this development board a suitable platform for evaluating Virtex-II Pro based designs. Some of the features of the Virtex-II Pro development board are listed below:

- Xilinx XC2VP4/P7-FG456 FPGA
- Four Rocket I/O™ ports supporting 2.5Gbits/port
- Two on-board oscillators @ 100, and 125MHz
- On-board Oscillator Socket (4/8-Pin Oscillators)
- User Clock inputs via Differential SMA Connectors
- 32M Mobil SDRAM Memory
- LCD Panel
- Two 18V04 ISP PROMs
- JTAG Programming/Configuration Port
- CPU JTAG/Debug Port
- CPU TRACE Port
- System ACE™ Connector
- RS232 Port
- User LEDs
- User DIP Switch
- User Push-Button Switches

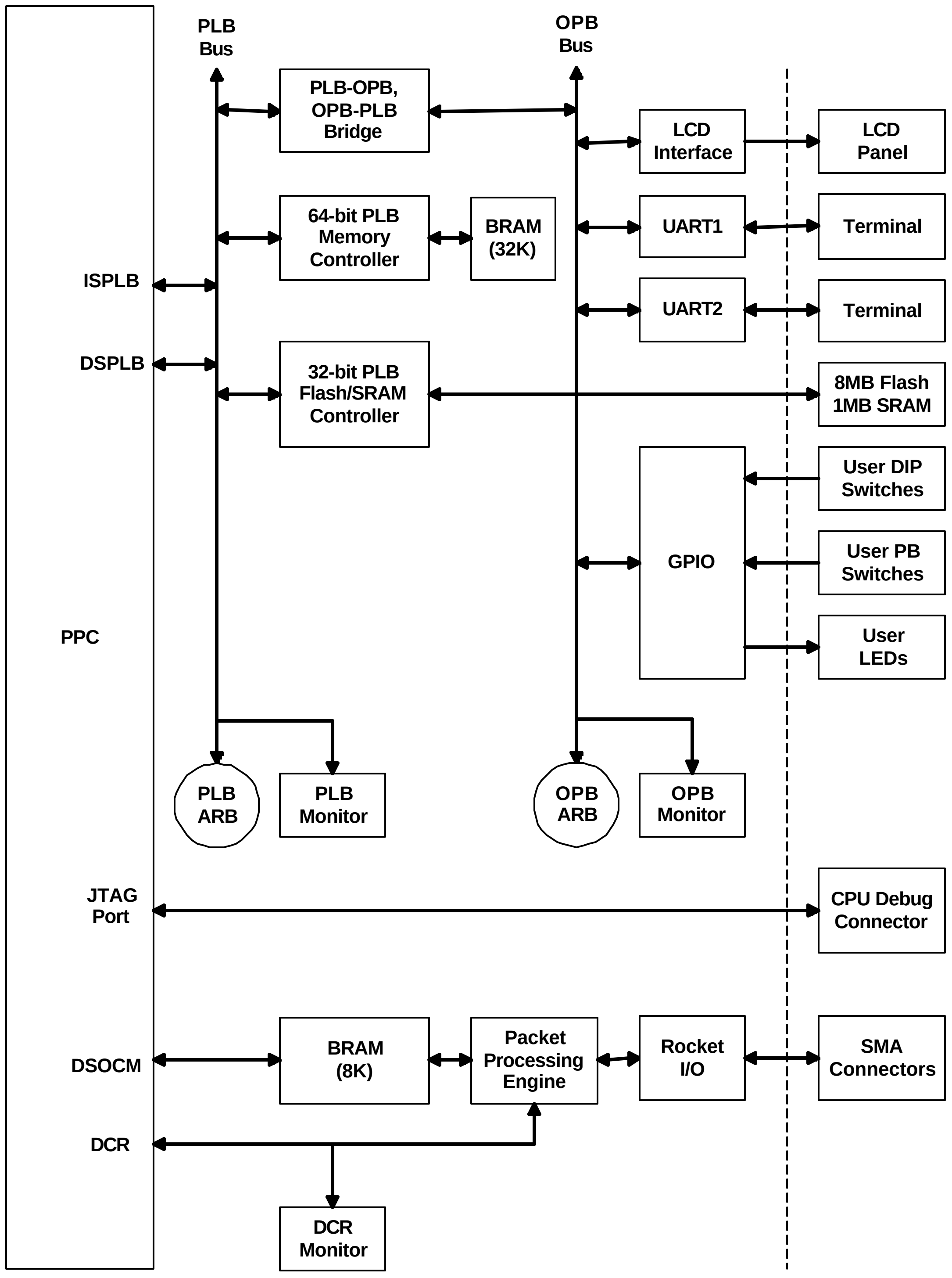

Figure 1 - PPC Reference Design Block Diagram

2.1 Processor Local Bus (PLB)

The PLB is used to connect the CPU to high-performance devices, such as high-speed memory controllers and offers a rich set of bus transactions that allow the CPU to access external memory for code and operand transfers at full-speed. The high-performance features of the PLB protocol, such as synchronous architecture and independent read/write data paths, enable the CPU to achieve maximum speed in performance intensive applications. This reference design uses a 64-bit PLB infrastructure with 64-bit master and 64/32-bit slave devices attached to the bus.
The PLB devices in this reference system consist of:

PLB Masters
- CPU Instruction-Side PLB Interface (ISPLB)
- CPU Data-Side PLB Interface (DSPLB)
- OPB-to-PLB Bridge

PLB Slaves
- BRAM Controller (CPU Instruction and Data Storage)
- SRAM/Flash Controller
- PLB-to-OPB Bridge

PLB Arbiter
- 8 Master, 64-bit Xilinx PLB Arbiter

PLB Bus Logic
- The specification for the PLB protocol requires that some additional logic be included in the design to OR together the outputs of the slaves to create the PLB slave bus. This sample system supports up to nine PLB slave devices, but can easily be expanded to support additional slaves.

In general, all PLB devices are optimized around the Virtex-II Pro architecture and make use of pipelining to improve maximum clock frequencies and reduce logic utilization. Refer to the accompanying documentation for each device for more information about its design.

2.2 On-Chip Processor Bus (OPB)

The OPB is used to connect lower-performance peripheral devices to the CPU. The OPB has a less complex architecture, which simplifies peripheral development. A PLB-to-OPB Bridge translates PLB transactions into OPB transactions, allowing the CPU to access the devices connected to the OPB. Devices that reside on the OPB can also access PLB devices by way of an OPB-to-PLB Bridge. The OPB devices in this reference system consist of:

OPB Masters
- PLB-to-OPB Bridge-Out

OPB Slaves
- OPB Arbiter Configuration Registers
- General-Purpose Input/Output (GPIO)
- 16450 UART (UART1)
- 16550 UART (UART2)
- LCD Controller
- OPB-to-PLB Bridge-In

OPB Arbiter
- Master, 32-bit Xilinx OPB Arbiter

OPB Bus Logic
- The OPB protocol specification requires that some additional logic be included in the design to AND/OR together the outputs of the slaves to create the OPB slave bus. This

sample system supports up to 13 OPB slave devices, but can easily be expanded to support additional slaves.

In general, all OPB devices are optimized around the Virtex-II Pro architecture and make use of pipelining to improve maximum clock frequencies and reduce logic utilization. Refer to the accompanying documentation for each device for more information about its design.

The OPB devices in the reference design make use of Intellectual Property InterFace (IPIF) modules to further simplify IP development. The IPIF converts the OPB protocol into common interfaces, such as an SRAM protocol or a control register interface. IPIF modules also provide support for DMA and interrupt functionality. IPIF modules simplify software development since the IPIF framework has many common features and supports a discovery process using a Configuration ROM (CROM). The CROM contains information about the devices in the system (such as address mapping, device capabilities, and revisions) that software can read at boot-up to configure its device drivers.

Note that the IPIF is designed mainly to support a wide variety of common interfaces, but may not be the optimal solution in all cases. Where additional performance or functionality is required, the user can develop a custom OPB interface. The IPIF protocols can also be extended to support other bus standards, such as PLB. This allows the backend interface to the IP to remain the same while the bus interface logic in the IPIF is changed. This provides an efficient means for supporting different bus standards with the same IP device.

The OPB specification supports masters and slaves of up to 64 bits with a *dynamic bus sizing* capability that allows OPB masters and slaves of different sizes to communicate with each other. The PPC Reference Design uses a subset of the OPB specification, which only supports 32-bit byte enable masters and slaves. Legacy devices utilizing 8- or 16-bit interfaces or those that require dynamic bus sizing functionality are not directly supported. For systems requiring legacy support, an OPB interface module is available to convert OPB byte enable cycles into dynamic bus sizing transactions. It is recommended that all new OPB peripherals support byte enable operations for better performance and reduced logic utilization.

2.3 On-Chip Memory Bus (OCM)

The Data Side of the OCM Bus (DSOCM) is used in this reference design to interface to the Rocket I/O data buffers. These buffers consist of a Transmit buffer and a Receive buffer. The DSBRAM memory (8KB block organized as 2K x 32 bits) connected to the DSOCM bus is partitioned into segments to provide storage for program data, as well as transmit and receive buffers for the Rocket I/O interface.

The Rocket I/O interface consists of DSBRAM, Packet Processing Engine (PPE) IP core from Xilinx, and the Rocket I/O SERDES. The PPE control and status registers are accessed via the processor DCR bus. All of the transmit and receive packet flow control is performed via the PPE control and status registers.

2.4 Device Control Registers (DCR)

The DCR offers a very simple interface protocol and is used for accessing control and status registers in various devices. It allows for register access to various devices without loading down the OPB and PLB interfaces. The only DCR devices used in this reference design are the control and status registers of the PPE and the DSOCM internal registers.

The CPU contains a DCR master interface that is accessed through special Move To DCR (**mtdcr**) and Move From DCR (**mfdcr**) instructions.

The DCR slave devices connected to the DCR bus consist of:

- Packet Processing Engine (Rocket I/O interface)
- Data-Side OCM (DSOCM) registers

2.5 PPC Reference Design Interrupts

The CPU also contains two interrupt pins, one for critical interrupt requests and the other for non-critical interrupts. The critical interrupt request is not used for this application while the non-critical interrupt request of the processor is connected to the UART1 and UART2 interrupt outputs (these interrupts are connected to the processor via an OR gate).

2.6 PPC Reference Design Clocks

Two Digital Clock Managers (DCM) are used to generate the clocks for the PPC Reference Design. A 50 MHz input reference clock input is used to generate the main 50 MHz clock for the PLB and OPB busses while a 100 MHz clock is generated for the DCR and OCM interfaces. The PLB clock is multiplied by 4 to generate a 200 MHz clock for the CPU. A fully synchronous system-level design is achieved by generating various clocks from the same 50 MHz clock input.

The CPU clock can run at any integer multiple of the PLB clock up to the maximum CPU clock frequency. During reset, internal clock synchronizers in the CPU detect the phase alignment of the PLB and CPU clocks and adjust for it automatically. The OCM clock must be divided down from the CPU clock by an integer multiple (up to eight), and the two clocks must be synchronous to each other. A second DCM acts independently to drive the clock input to the Rocket I/O section. The Rocket I/O transceiver clock is set to 125 MHz to yield a 2.5-GHz serial output rate.

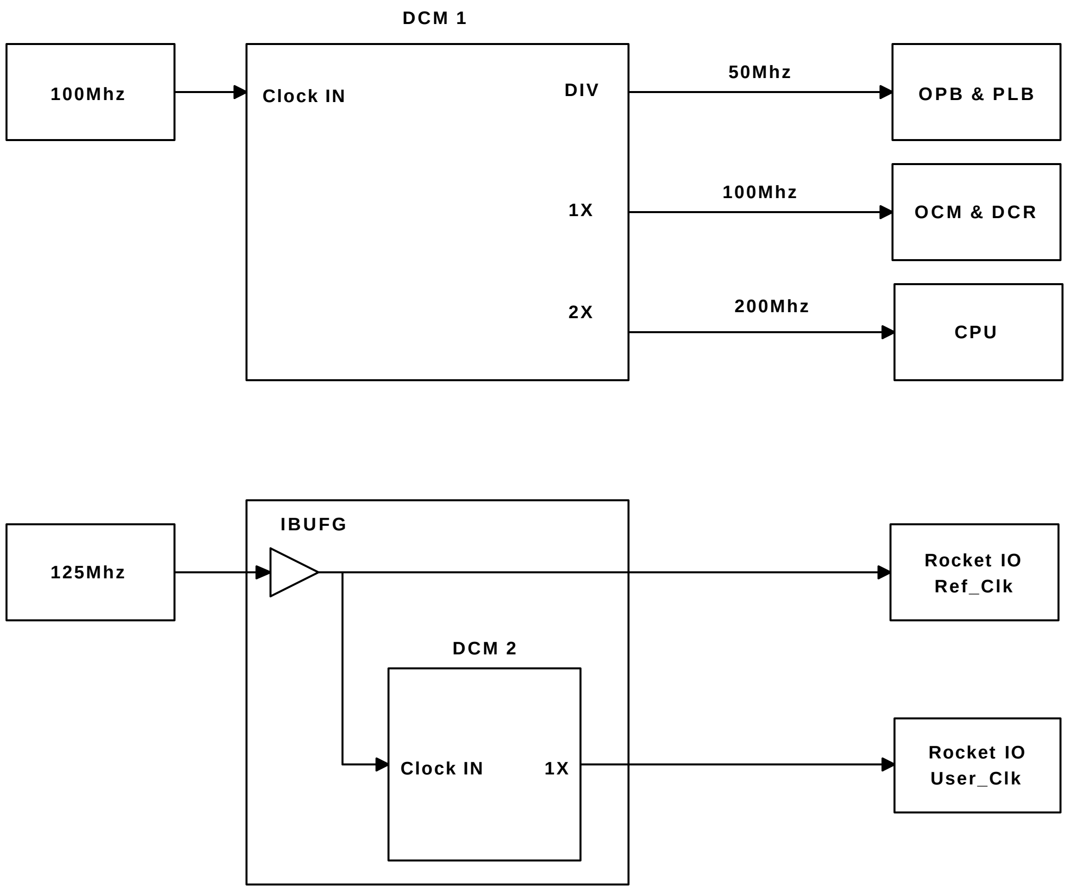

Figure 2 - Clock Generation

2.7 PPC Reference Design Reset

After a system reset or at FPGA startup, the FPGA is held in reset until the DCM has locked onto its reference clock. Once the DCM is locked and the clocks remain stable for several cycles (may take several microseconds in simulation), the reset condition is released to allow the system logic to begin operating. (For example, the CPU will begin fetching instructions a few cycles after reset is released.) Since the reset net is a high-fanout signal, it may not be able to reach all the logic in the design within one clock cycle. User IP blocks should be designed to take into account the possible skew in the global reset and still start up properly. Alternatively, the global reset can be registered locally in each IP block to generate a synchronous reset signal.

The design implements the three levels of reset supported by the PPC405:

- Core reset
- Chip reset
- System reset

The core reset only affects the processor while the chip reset clears all the logic on the FPGA. The system reset is designed to reset the entire system including the FPGA and external devices connected to the FPGA. The CPU provides an internal special-purpose register that allows software to request that one of the three resets be performed.

The following figure shows how the reset logic in the PPC Reference Design is implemented. The chip and core reset request lines from the CPU are driven to the corresponding reset nets through shift registers to ensure that minimum pulse width requirements are met. The system reset request line from the CPU drives an external pin as an open-drain signal (with resistive pullup) so that other external devices can sense or drive the system reset (sys_rst_n). Note that the system reset has the effect of resetting the DCMs in the FPGA, while the chip reset does not affect the DCMs. The reset logic in the PPC Reference Design is an example implementation of the PPC405 reset architecture. Designers should set the scope, boundaries, and effects of resets as appropriate to their designs. It should be noted that the sys_rst_n signal is connected to the CPU RESET Push Button Switch on the Virtex-II Pro development board.

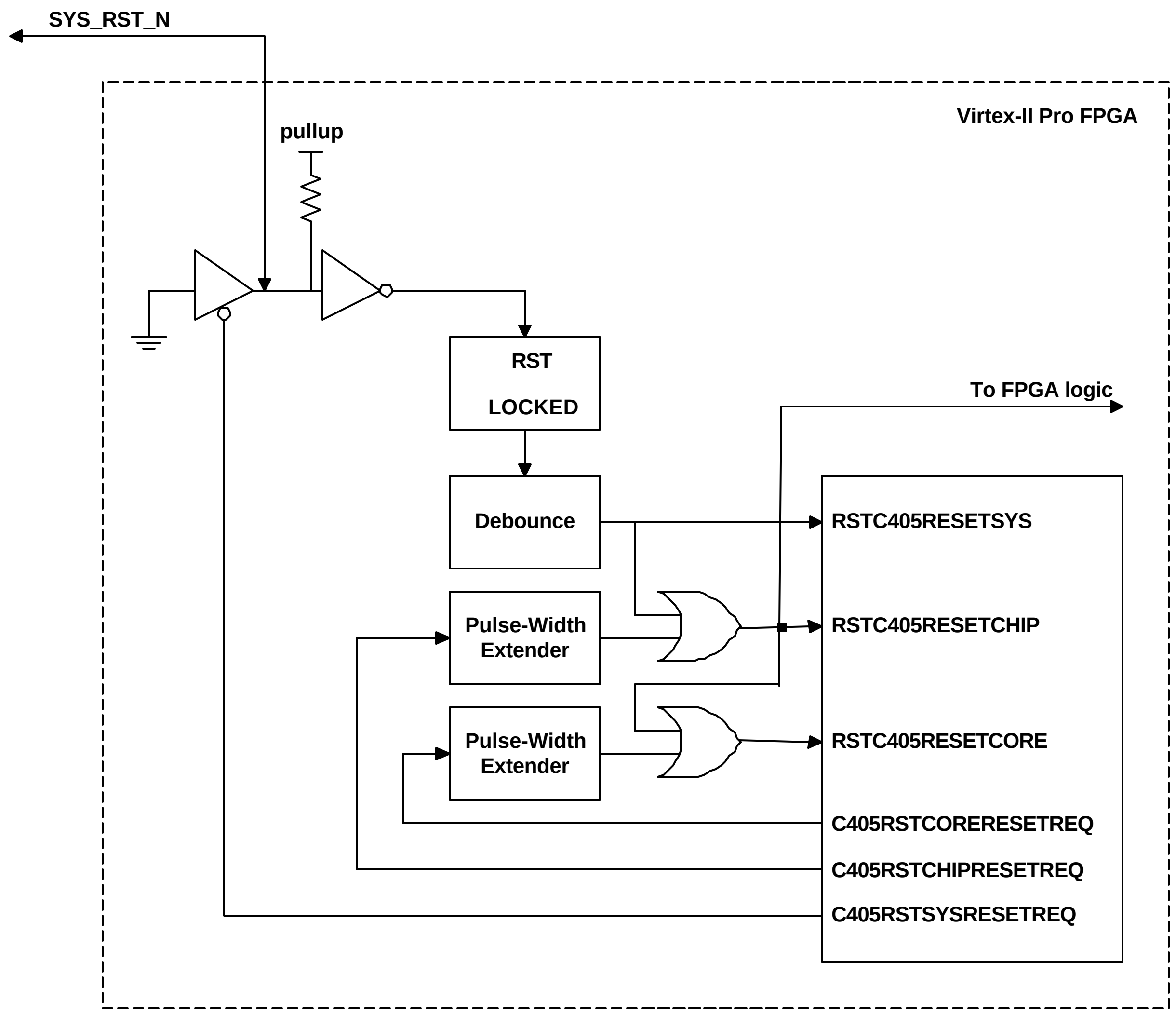

Figure 3 - Reset Generation

3 PPC Reference Design Memory Map

This section diagrams the system memory map for the PPC Reference Design. All memory and I/O devices that reside on the PLB. OPB, and OCM busses are mapped into the 4G address space of the processor. The memory map reflects the default location of the system devices as defined in the **global_params.v** file.

3.1 PLB Memory Map

The following tables show the devices located on the PLB. Although 32M of address space is allocated to the FLASH and SRAM that are located on the optional P160 module, only 8M of physical FLASH and 1M of physical SRAM is available on this module (32M of address space is allocated to these memory devices in order to reduce the number of address bits that are required to generate their associated chip selects).

Table 1 - PLB Memory Map

Address Range	Size	Usage
0x00000000 – 0x01FFFFFF	32M	SRAM (1M of SRAM)
0x02000000 – 0x03FFFFFF	32M	FLASH (8M of Flash)
0x04000000 – 0x7FFFFFFF	2085M	Unused
0x80000000 – 0xEFFFFFFF	1877M	PLB-OPB Bridge (OPB memory space)
0xF0000000 – 0xF3FFFFFF	67M	Unused
0xF4000000 – 0xF4001FFF	8K	DSBRAM (Dual-Port RAM for the Packet Processor)
0xF4002000 – 0xFFFF7FFF	201M	Unused
0xFFFF8000 – 0xFFFFFFFF	32K	PLB BRAM

Table 2 - OPB Memory Map

Address Range	Size	Usage
0x80000000 – 0x8FFFFFFF	268M	Unused
0x90000000 – 0x90000007	8B	GPIO (32-bit GPIO Port)
0x90000008 – 0xA0000FFF	268M	Unused
0xA0001000 – 0xA000101F	32B	UART1 (located on the main board)
0xA0001020 – 0xA0010FFF	2K	Unused
0xA0011000 – 0xA001101F	32B	UART2 (located on the P160 module)
0xA0011020 – 0xBFFFFFFF	536M	Unused
0xC0000000 – 0xC0000007	8B	LCD (located on the main board)
0xC0000008 – 0xEFFFFFFF	805M	Unused

3.2 DCR Memory Map

The device-control register (DCR) interface provides a mechanism for the processor block to initialize and control peripheral devices that reside on the same FPGA chip. For example, the memory-transfer characteristics and address assignments for a bus-interface unit (BIU) can be configured by software using DCR. The DCR is accessed using the PowerPC **mfdcr** and **mtdcr** instructions and it consists of the following:

- A 10-bit address bus.
- Separate 32-bit input and output data busses.
- Separate read and write control signals.
- A read/write acknowledgement signal.

Because the processor block is the only bus master on the bus, the address bus is driven by the processor block and received by each peripheral containing DCR. The read and write control signals are also distributed to each DCR peripheral.

Table 3 - DCR Bus Memory Map

Address Range	Size	Usage
0x01A – 0x01B	2B	DSOCM
0x200 – 0x20F	16B	Packet Processing Engine (PPE)

4 PPC Reference Design GPIO Port

A 32-bit GPIO port is implemented in order to provide user interface to the on-board DIP Switches, Push Button Switches, and LEDs. The following table shows how the GPIO bits are utilized for this reference design.

Table 4 - GPIO Port Description

GPIO Bit	Direction	Usage
0	Input	User DIP Switch 1
1	Input	User DIP Switch 2
2	Input	User DIP Switch 3
3	Input	User DIP Switch 4
4	Input	User DIP Switch 5
5	Input	User DIP Switch 6
6	Input	User DIP Switch 7
7	Input	User DIP Switch 8
8	Input	Unused
9	Input	Unused
10	Input	Unused
11	Input	Unused
12	Input	User Push Button Switch 3
13	Input	User Push Button Switch 2
14	Input	User Push Button Switch 1
15	Input	P160 FLASH BY/RY Signal
16	Output	Unused
17	Output	Unused
18	Output	Unused
19	Output	Unused
20	Output	Unused
21	Output	Unused
22	Output	Unused
23	Output	P160 FLASH Reset Signal
24	Output	Unused
25	Output	Unused
26	Output	Unused
27	Output	Unused
28	Output	User LED4
29	Output	User LED3
30	Output	User LED2
31	Output	User LED1

5 PPC Reference Design HDL Organization

The following figure shows the organization of the higher-level HDL files that comprise the system and test bench environment.

All of the peripherals, the memory controllers, and the CoreConnect infrastructure are instantiated in the **ip_wrapper.v** file. The external I/O signals, the clock/reset connections, and the processor block connections are propagated up to the top-level file, **top.v**. The processor block, clock/reset logic, Rocket I/O transceivers, and the **ip_wrapper.v** file are instantiated in the **top.v** file. The module ports of **top.v** represent the external I/O signals of the FPGA. Therefore, all files in the hierarchy from the **top.v** level and below make up the Virtex-II Pro FPGA design that is synthesized and routed into an FPGA.

For simulation, the file **testbench.v** instantiates the FPGA as the device under test and includes behavioral models for the FPGA to interact with. In addition to behavioral models for clock oscillators, and external peripherals, the test bench also instantiates the CoreConnect bus monitors to observe the PLB, OPB, and DCR buses for protocol violations. The **sim_params.v** file is designed to be modified by the user to customize various simulation options. These options include message display options, maximum simulation time, and clock frequency. The user should edit this file to reflect personal simulation preferences.

An additional HDL file, called **global_params.v**, defines the global constants and parameters used throughout the reference design. This information may include, memory map definitions, device configuration values, or any other information that is better organized in a global context.

Some of the test bench code is used to access signals internal to the design using hierarchy path names to reach into the design without changing any of the port interfaces. It is important that the design source files used for simulation match the source files for synthesis. Therefore, port interfaces should not be different or else inconsistencies can result.

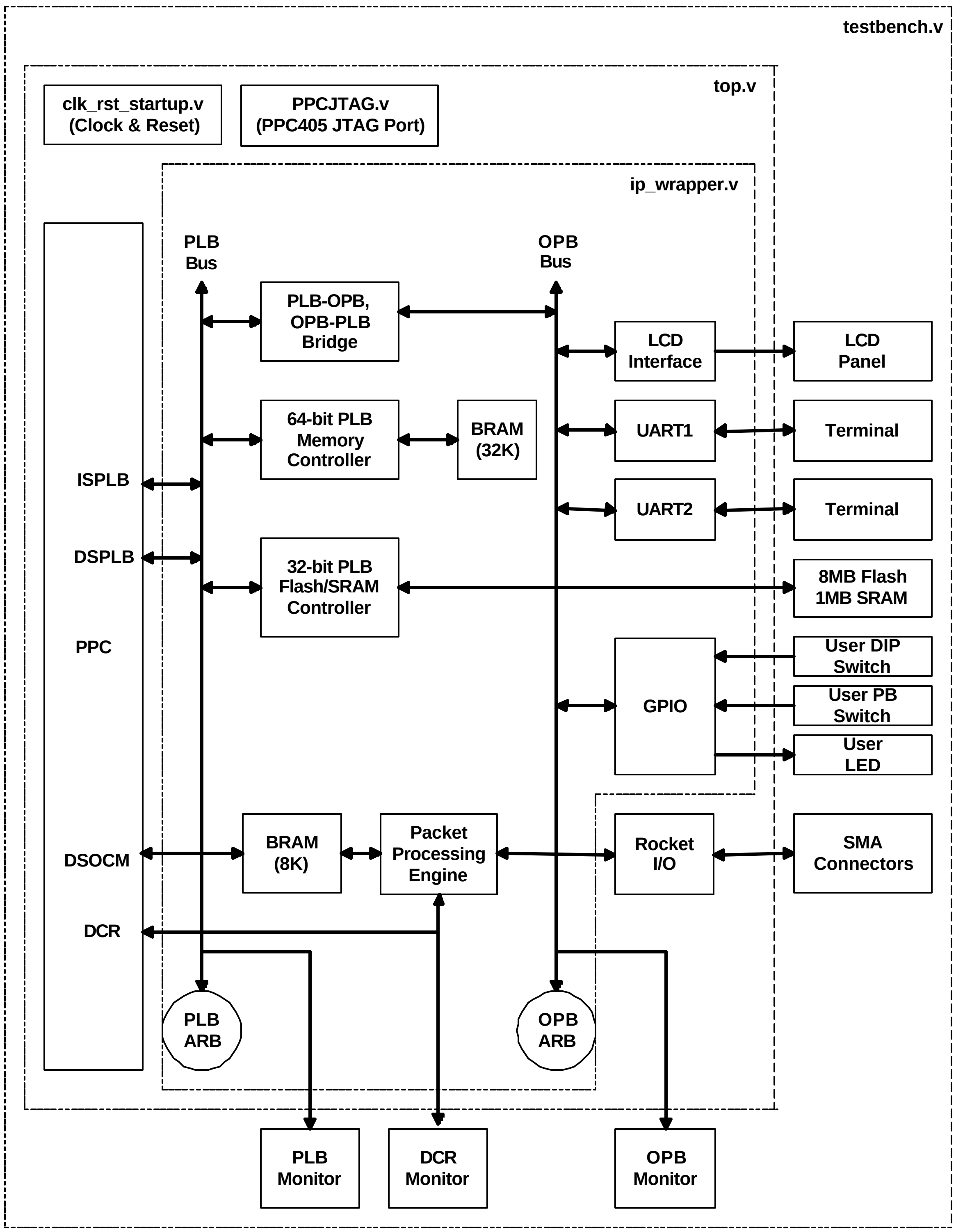

Figure 4 - HDL Organization

The following table shows the Verilog modules that are included in the wrapper file and it also provides a brief description for each file.

Table 5 - PPC Reference Design IP Wrapper File

File Name	Description
arbiter.v	PLB Arbiter is used by various maters on the PLB to gain control of the PLB. For the PPC Reference Design, there are 3 masters, the ISPLB, DSPLB, and OPB-PLB Bridge.
opb_arbiter.v	OPB Arbiter is used to gain access to the OPB bus. For the PPC Reference Design, CPU uses the arbiter (via the PLB) to access devices that reside on the OPB bus.
plb_bus_logic.v	OR logic for the PLB is used to combine the signals from slaves that reside on the PLB bus.
opb_bus_logic.v	OR logic for the OPB is used to combine the signals from slaves that reside on the OPB bus.
plb_bram_cntlr.v	64-bit PLB Memory Controller is used to interface to the PLB BRAM (32K).
plb2opb_bgo.v	PLB-OPB Bridge is used to access devices on the OPB bus
bram_block.v	Wrapper file for the PLB BRAM and it is used to instantiate RAMB16_S4 BlockRAMs to build the 32K bytes of the PLB BRAM.
opb_gpio_top.v	GPIO source file
plb_sram_flash_interface_top.v	Top-level module for the 32-bit PLB Flash/SRAM Controller. This module is used to access the 8M Flash and 1M SRAM that reside on the PLB.
opb_lcd_cntlr_top.v	LCD Controller source file
opb2plb_bgi.v	OPB-PLB Bridge is used by OPB masters to gain control of the devices that reside on the PLB bus.
opb_uart16450_eval.v	UART 16450 Wrapper file
opb_uart16550_eval.v	UART 16550 Wrapper file
DS_bram_wrap.v	Wrapper file for the PLB BRAM and it is used to instantiate RAMB16_S9_S9 BlockRAMs to build the 8K bytes of the DSBRAM.
PPE.v	Packet Processing Engine source file
pkt_proc_dcr_module.v	Packet Processing Engine DCR source file

The following table shows the Verilog modules that are instantiated in the top-level file and it also provides a brief description for each file.

Table 6 - PPC Reference Design Top Level File

File Name	Description
ip_wrapper.v	Wrapper file containing the devices connected to the PLB, OPB, DCR, and OCM busses.
PPC405.v	PowerPC Core Wrapper file
clk_rst_startup.v	Clock and Reset
PPCJTAG.v	CPU JTAG that can be connected to the FPGA JTAG pins (the other option would be to connect the CPU JTAG pins to general purpose FPGA I/O pins.
GT_CUSTOM.v	Verilog Module declaration for the Rocket I/O SERDES (CUSTOM configuration mode).

The following table shows the Verilog modules that are instantiated in the testbench file and it also provides a brief description for each file.

Table 7 - PPC Reference Design Test Bench File

File Name	Description
top.v	PPC Reference Design Top Level File
plb_monitor3x.v	PLB Bus Monitor
dcr_monitor.v	DCR Bus Monitor
opb_monitor.v	OPB Bus Monitor
LCD behavioral model	Behavioral Model for the LCD Panel
UART behavioral model	Behavioral Model for the UART Panel

6 PPC Reference Design Software Source Files

The source files needed to do a complete firmware build are located in **C:/Memec_Desgin_V2Pro_Board** directory. The following table gives a brief overview of what each source file is for. Include files are located in **C:/Memec_Desgin_V2Pro_Board /includes** directory.

Table 8 - Software Source Files

File Name	Description
boot.s	Contains a single instruction jumps to **_start** in **crt0.S** for reset vector (0xFFFFFFFC). The boot section is mapped to the reset vector during linking**.**
eabi.s	Contains code to initialize the EABI environment (i.e., setting the registers R2 and R13 to the correct values). The compiler inserts a call to **_eabi** at the beginning of the **main()** function.
crt0.s	Contains code to initialize the **.bss** and **.sbss** data sections and to set up the stack. This file also contains code to initialize processor features listed in the Initialization section.
data.c	Contains functions for packet processing, UART, LCD, and GPIO. Also contains device driver functions to initialize and test the Packet Processing Engine hardware.
initialize.c	Contains function calls to initialize all of the high-speed serial hardware, UART, LCD, and GPIO.
Memec_Desgin_V2Pro_Board.c	Contains the **main()** function and declares global variables used by all modules.
ppc-asm.h	Contains register defines specific to the PPC405.
includes.h	Includes all other **.h** files and it is present in all **.c** files. Contains defines and macros for all modules. Also, contains extern global variable declarations.
data.h	Contains extern function declarations for functions used by other modules.
initialize.h	Contains extern function declarations for functions used by other modules.

makefile	Builds the PPC Reference Design application.
mapfile	The mapfile is the linker script file. It defines the memory map and how sections are mapped within memory.

7 Simulation and Verification

7.1 SWIFT and BFM CPU Models

The PPC Reference Design demonstrates two different simulation methods to help verify designs using the PPC405 CPU. One method uses a full simulation model of the CPU based on the actual silicon. The second method employs bus functional models (BFMs) to generate processor bus cycles from a command scripting language. These two methods offer different trade-offs between behavior in real hardware, ease of generating bus cycles, and the amount of real time to simulate a given clock cycle.

A SWIFT model can be used to simulate the CPU executing software instructions. In this scenario, the executable binary images of the software are preloaded into memory from which the CPU can boot up and run the code. Though this is a relatively slow way to exercise the design, it more accurately reflects the actual behavior of the system.

The SWIFT model is most useful for helping to bring up software and for correlating behavior in real hardware with simulation results. The PPC Reference Design demonstrates the SWIFT model simulation flow, by allowing the user to write a C program that is compiled into an executable binary file. This executable (in ELF format) is then converted into BRAM initialization commands using a tool called Data2BRAM (Note that Data2BRAM can also generate memory files for the Verilog command **readmemh** to initialize other memories such as SRAM or DDR memory).

When a simulation begins and reset is released, the CPU SWIFT model fetches the instructions from BRAM (which is mapped to the boot vector) and begins running the program. The user can then observe the bus cycles generated by the CPU or any other signal in the design. For debugging purposes, the values of the CPU's internal program counter, general-purpose registers, and special-purpose registers are available for display during simulation.

Generating a desired sequence of bus operations from the CPU may require a lot of software setup or simulation time. For early hardware bring-up or IP development, a bus functional model can be used to speed up simulation cycles and avoid having to write software. A model of the CPU is available in which two PLB master BFMs and one DCR BFM are instantiated to drive the CPU's PLB/DCR ports. These BFMs are provided in the CoreConnect toolkits and allow the user to generate bus operations by writing a script written in the Bus Functional Language (BFL). The PPC Reference Design provides a sample BFL script that exercises many of the peripherals in the system. Refer to the CoreConnect Toolkit documentation for more information.

Since the CPU SWIFT model and BFM model both have the same set of port interfaces, users can switch between the two simulation methods by compiling the appropriate set of files without having to modify the system's design source files. Users may, however, need to modify their test benches to take into account which model is being used.

7.2 Behavioral Models

The PPC Reference Design includes some behavioral models to help exercise the devices and peripherals in the FPGA. Many of these models are freely available from various manufacturers and include interface protocol-checking features. The behavioral models and features included in the reference design are:

- Pull-ups connected to the GPIO for reading and driving outputs without getting unknown values
- Terminal interface connected to the UARTs for sending and receiving serial data
- The terminal allows a user to interact with the simulation in real time
  - Characters sent out by the UARTs are displayed on a terminal while characters typed into the terminal program are serialized and sent to the UARTs
  - A simple file I/O mechanism passes data between the hardware simulator and the terminal program
- LCD model that implements some registers that the LCD controller can write to and read back from
  - It does not decode the control register commands to display the characters sent to the LCD
  - A terminal program similar to the UART terminal is provided to display characters written to the LCD
- A Rocket I/O packet processor module

8 Synthesis and Implementation

The PPC Reference Design can be synthesized and placed/routed into a Virtex-II Pro FPGA. A basic set of timing constraints for the design is provided to allow the design to go through place and route. Note that some peripherals in the design are pre-synthesized and provided only in net list format (**.edf**, **.edn**, **.ngc**, **.ngo**). For these devices, a corresponding black box has been instantiated in the source code to force the synthesis tool to leave a placeholder for the netlist. After a successful place and route, it is possible to run a simulation of the design using a back-annotated timing model of the FPGA with a SWIFT or BFM version of the processor block.

9 Design Flow Environment

A flow engine provides an environment to help manage the design flow for the PPC Reference System. This engine uses the utility program called **make** and a set of PERL scripts to allow the user to perform tasks, such as running simulations, synthesizing a design, or implementing it on an FPGA with a simple set of commands. The design flow tool is implemented with a generic architecture to allow it to be adapted for use on a variety of different designs.

10 PPC Reference Design File Listings

The files and directories specific to the PPC Reference Design are located below the **$V2PRO/platforms/ Memec_Design_V2Pro_Board** directory, shown in the following figure, and listed in the tables that follow. Source files for the IP modules and software applications are kept in the **$V2PRO/source** directory. The **$V2PRO/source** directory generally contains files that can be shared by different designs. The **File Structure for the PPC Reference System** section diagrams the full file directory structure for this design. Refer to the appropriate instructions for installation and setup information. Note that the following tables only list files in the **$V2PRO/platforms/ Memec_Desgin_V2Pro_Board** directory that are present after installation. After running simulation, synthesis, or place and route, additional files may be created. Directory path names are shown separated by the "/" character as is the UNIX convention. For Windows, the "\" should be used to separate directory paths.

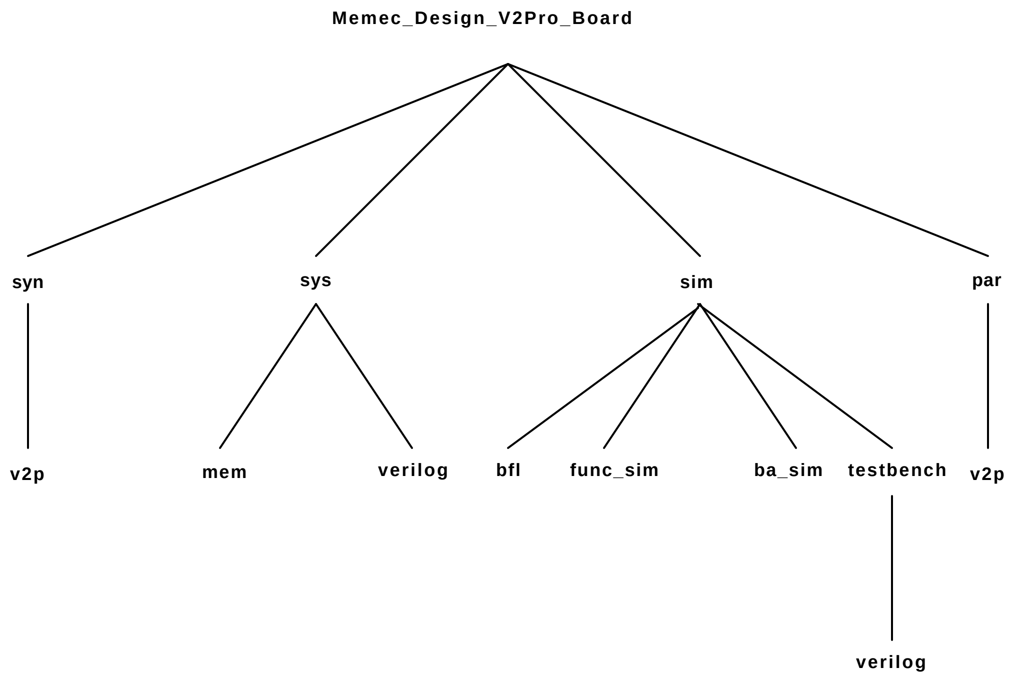

Figure 5 - PPC Reference Design Directory Structure

Table 9 - Files in $V2PRO/platforms/ Memec_Design_V2Pro_Board

File	Description
Makefile	Redirection file to allow the centralized design flow scripts to be invoked from different locations. The centralized design flow script (**flow.mk**) is located in the **$V2PRO/tools/cygwin/xilinx/data** directory
flow.cfg	File used to configure the design flow scripts

10.1 Using the makefile and flow.cfg Files

The makefile located in the Memec_Design_V2Pro_Board directory is used as a redirection file to point to the centralized design flow script (flow.mk) that is located in the $V2PRO/tools/cygwin/xilinx/data directory. The flow.cfg file is used to configure the flow.mk script for the PPC Reference Design. The following figure shows a graphical representation of the relationship between the flow.cfg and makefile files and the flow.mk script.

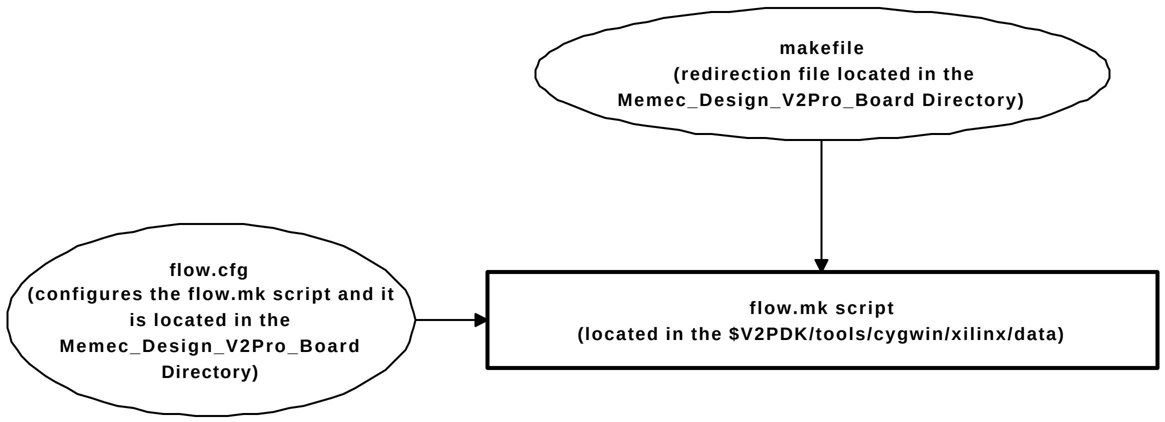
makefile
(redirection file located in the Memec_Design_V2Pro_Board Directory)
flow.cfg
(configures the flow.mk script and it is located in the Memec_Design_V2Pro_Board Directory)
flow.mk script
(located in the $V2PDK/tools/cygwin/xilinx/data)

Figure 6 - makefile and flow.cfg Usage

Table 10 - Files in $V2PRO/platforms/ Memec_Design_V2Pro_Board /sys

File	Description
bram_init.bmm	Data2BRAM configuration file describing the system memory devices

10.2 The bram_init.bmm File

The bram_init.bmm describes the system memory and it is used by the DATA2BRAM program to initialize the PLB BRAM and also the DSBRAM. The PLB BRAM is a 64-bit memory block that resides in the 0xFFFF8000 – 0xFFFFFFFF memory space, while the DSBRAM is a 32-bit memory block that is located in the 0xF4000000 – 0xF4001FFF memory space. The content of the bram_init.bmm file for the PPC Reference Design is shown below.

```
ADDRESS_BLOCK plb_bram_controller RAMB16 [0xFFFF8000:0xFFFFFFFF]
  BUS_BLOCK
    ip_wrapper/bram_block/block_ram0  [63:60];
    ip_wrapper/bram_block/block_ram1  [59:56];
    ip_wrapper/bram_block/block_ram2  [55:52];
    ip_wrapper/bram_block/block_ram3  [51:48];

    ip_wrapper/bram_block/block_ram4  [47:44];
    ip_wrapper/bram_block/block_ram5  [43:40];
    ip_wrapper/bram_block/block_ram6  [39:36];
    ip_wrapper/bram_block/block_ram7  [35:32];

    ip_wrapper/bram_block/block_ram8  [31:28];
    ip_wrapper/bram_block/block_ram9  [27:24];
    ip_wrapper/bram_block/block_ram10 [23:20];
    ip_wrapper/bram_block/block_ram11 [19:16];

    ip_wrapper/bram_block/block_ram12 [15:12];
    ip_wrapper/bram_block/block_ram13 [11:8];
    ip_wrapper/bram_block/block_ram14 [7:4];
    ip_wrapper/bram_block/block_ram15 [3:0];
  END_BUS_BLOCK;
END_ADDRESS_BLOCK;


ADDRESS_BLOCK dsocma RAMB16 [0xF4000000:0xF4001FFF]
        BUS_BLOCK
                ip_wrapper/DSBRAM/u3 [31:24];
                ip_wrapper/DSBRAM/u2 [23:16];
                ip_wrapper/DSBRAM/u1 [15:8];
                ip_wrapper/DSBRAM/u0 [7:0];
        END_BUS_BLOCK;
END_ADDRESS_BLOCK;
```

Table 11 - Files in $V2PRO/platforms/ Memec_Desgin_V2Pro_Board /sys/verilog

File	Description
src.lst	List of source files used in the PPC Reference Design
bram_block.v	Wrapper file for the BRAM connected to the PLB
global_params.v	Centralized location for the global parameters used within the design
clk_rst_startup.v	Clock, reset, and startup logic for the PPC Reference Design
plb_sram_flash_interface_top.v	Top-level module for the 32-bit PLB Flash/SRAM Controller. This module is used to access the 8M Flash and 1M SRAM that reside on the PLB.
sram_flash_interface_core.v	Core module for the 32-bit PLB Flash/SRAM Controller.
ip_wrapper.v	Wrapper file containing the devices connected to the PLB, OPB, DCR, and OCM busses
opb_bus_logic.v	OR logic for the OPB
plb_bus_logic.v	OR logic for the PLB
top.v	Top-level file for the PPC Reference Design

10.3 The src.lst File

The src.lst is a list of all source files used in the PPC Reference Design. The following shows a section of this file for the PPC Reference Design.

- Modules Connected to PLB
  $V2PRO/source/hw/verilog/plb_arbiter/src.lst;
  $V2PRO/source/hw/verilog/plb_bram_cntlr/src.lst;
  $V2PRO/source/hw/verilog/ipif_slv_sram_simple/src/plb_ipif_slv_sram.v;
  $V2PRO/source/hw/verilog/plb2opb_bgo/src.lst;
  $V2PRO/source/hw/verilog/opb2plb_bgi/src_bgi_simple.lst

- Modules Connected to OPB
  $V2PRO/source/hw/verilog/opb_arb/src.lst;
  $V2PRO/source/hw/verilog/ipif_slv_sram_simple/src/opb_ipif_slv_sram.v;
  $V2PRO/source/hw/verilog/gpio/src.lst;

```
$V2PRO/source/hw/verilog/lcd_cntlr/src.lst;
$V2PRO/source/hw/verilog/uart/src_16450_eval.lst;
$V2PRO/source/hw/verilog/uart/src_16550_eval.lst;
```

10.4 The global_param.v File

The global_param.v file is a centralized location for the global parameters used within the PPC Reference Design. A section of this file is given below that shows the base address for the PLB BRAM, PLB FLASH/SRAM, LCD Controller, and the GPIO.

```
// PLB BRAM, 17 address bits are used in the address decoding
`define PLB_S0_BASE_ADDR 32'hffff_8000
`define PLB_S0_LSB_ADDR 16

// PLB FLASH and SRAM, 6 address bits are used in the address decoding
`define PLB_S2_BASE_ADDR 32'h70000000
`define PLB_S2_LSB_ADDR 5

// LCD base address
`define OPB_S1_AddrBase 32'hC000_0000

// GPIO base address
`define OPB_S2_AddrBase 32'h9000_0000
```

Table 12 - Files in $V2PRO/platforms/ Memec_Desgin_V2Pro_Board /sim/func_sim

File	Description
Uart1.input	Character to be sent to UART 1 from the terminal (set first line to "@0 0" for no input)
Uart2.input	Character to be sent to UART 2 from the terminal (set first line to "@0 0" for no input)
wave.do	MTI command to display a sample set of interesting signals for functional simulation

Table 13 - Files in $V2PRO/platforms/ Memec_Desgin_V2Pro_Board /sim/bfl

File	Description
test_Memec_Desgin_V2Pro_Board_ref_des.bfl	Sample BFL script to exercise the devices in the system during functional simulation

10.5 Bus Functional Language

Bus Functional Language is used for functional simulation of the PPC Reference Design. Simple memory commands are performed to access devices located on the PLB, OPB, and DCR busses. The following shows a section of the test_Memec_Desgin_V2Pro_Board_ref_des.bfl file that demonstrates how the BRAM on the PLB bus is tested. For more information on BFL, please refer to the CoreConnect Tool Kit documentation.

```
set_device (path=/testbench/top/PPC405/M_DCU,device_type=plb_master)
   configure  (msize=01) // 64 Bit DCU Master
   mem_update(addr=FFFFF040, data=0123_4567_89ab_cdef)
   mem_update(addr=FFFFF048, data=fedc_ba98_7654_3210)
   mem_update(addr=FFFFF050, data=0011_2233_4455_6677)
   mem_update(addr=FFFFF058, data=8899_aabb_ccdd_eeff)
   write     (addr=FFFFF040, size=0010)

// Read back and verify the data
   read      (addr=FFFFF040, size=0010)
   read      (addr=FFFFF060, size=0010)
   read      (addr=FFFFFF80, size=0010)
```

Table 14 - Files in $V2PRO/platforms/ Memec_Desgin_V2Pro_Board /sim/ba_sim

File	Description
Uart1.input	Character to be sent to UART 1 from the terminal (set first line to "@0 0" for no input)
Uart2.input	Character to be sent to UART 2 from the terminal (set first line to "@0 0" for no input)
wave.do	MTI command to display a sample set of interesting signals for back-annotated timing simulation

Table 15 - Files in $V2PRO/platforms/ Memec_Desgin_V2Pro_Board /sim/testbench/verilog

File	Description
opb_dcl.inc	OPB bus monitor configuration file set for 13 OPB devices
sim_params.v	Centralized location for parameters and settings that affect the simulation
testbench.v	Testbench source file
view_reg.v	Behavioral code to display internal CPU register contents whenever they are updated (SWIFT Simulations only)

10.6 The Opb_dcl.inc File

The OPB Bus Monitor Configuration file is used to tailor the bus for a given application. The following shows the content of the OPB Bus Monitor Configuration file for the PPC Reference Design.

```
`define true  1'b1
`define false 1'b0
`define random  2'b00
`define round  2'b01
`define priority  2'b10
`define test  2'b11
`define opb_data_bus_width  64
`define opb_be_bus_width (`opb_data_bus_width / 8 )
`define max_opb_devices 13
`define opb_unit_delay 2
```

```
`define opb_monitor_cmd_array_size  1024
`define opb_monitor_record_array_size  16
`define opb_monitor_synch_array_size  32
`define slave_cmd_array_size  2048
`define slave_check_array_size  1024
`define slave_mem_array_size  1024
`define q_max  7
`define opb_master_g_reg_array_size  32
`define master_cmd_array_size  4096
`define byte_count_max  8
`define byte_count_min  1
```

Table 16 - Files in $V2PRO/platforms/ Memec_Desgin_V2Pro_Board /par

File	Description
top.ucf	User constraints file used by FPGA implementation tools

Table 17 - Files in $V2PRO/platforms/ Memec_Desgin_V2Pro_Board /syn

File	Description
synplify.opt	Optional synthesis project file information

11 Instructions for Running Functional Simulations

The PPC Reference Design comes with SWIFT and BFM based simulation examples. This section describes the necessary steps for running the functional simulations. It assumes the Virtex-II Pro Design Kit is properly installed and that the **$V2PRO/platforms/ Memec_Desgin_V2Pro_Board/flow.cfg** file is properly edited to correctly call the user's simulation tools. The user may also modify the **sim_params.v** file to customize various simulation parameters. The following figure shows a high-level view of the functional simulation flow.

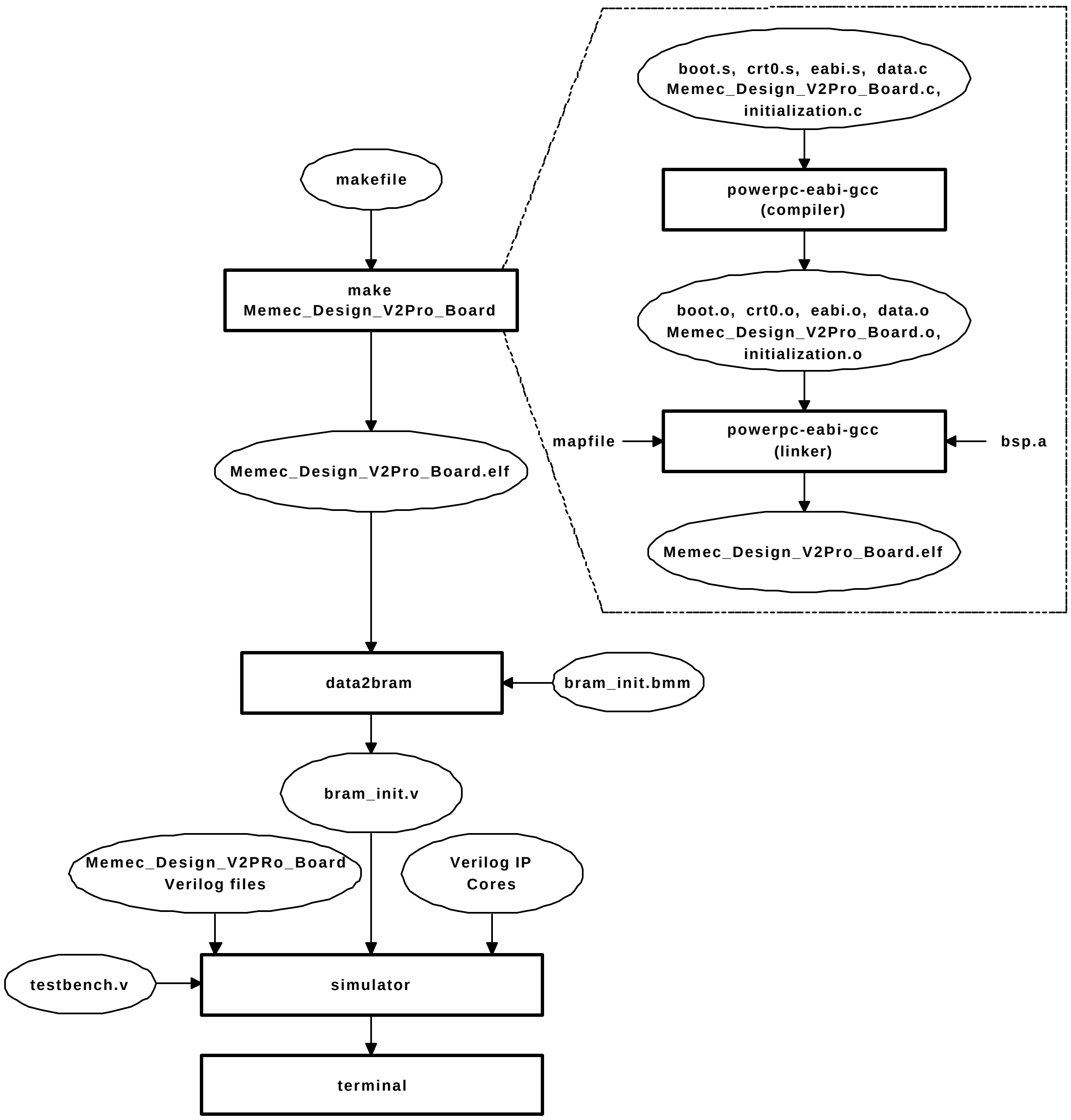

Figure 7 – Functional Simulation

11.1 SWIFT Simulations

The steps to run a SWIFT simulation are described below:

Table 18 - Instructions for Running Functional Simulations (CPU SWIFT Model)

Step	Action
1	Edit the **flow.cfg** file: $ cd $V2PRO/platforms/Memec_Desgin_V2Pro_Board $ <*text editor*> flow.cfg (On a Windows PC, you can use the Explorer to browse for this file and edit it.) The <*text editor*> can be a program like Emacs, vi, or WordPad. Make sure that the PPC_MODEL entry is set to "swift". Save and close the **flow.cfg** file.
2	Invoke the design flow engine to start the simulation: $ make func_sim (On a Windows PC, you can also select **Start > Programs > Virtex-II Pro Developer's Kit > Reference Platforms > Memec_Desgin_V2Pro_Board > Make Functional Simulation**) The flow engine begins by compiling and linking the application software (as necessary) to generate a binary executable program file (in **.elf** format). The **.elf** file is copied locally and the **data2bram** is invoked to generate the necessary BRAM initialization command files. The simulator tool is then launched along with terminals for the UART and LCD. **Note**: For MTI, the scripts will copy a corresponding NT or Solaris version of the **modelsim.ini** configuration file into the local directory. This file can be found in the **$V2PRO/tools/$V2PRO_HOST_OSTYPE/xilinx/data/modelsim.ini** directory.
3	Run the simulation: In the MTI window, type **run -all**. The simulator will test each UART in the system and if the test is successful, U1 is displayed in “UART1 Terminal” and U2 is displayed in the “UART2 Terminal”. Upon successful completion of the LCD test, “LCD” in printed on the “LCD Terminal”. The simulator will also perform a complete loop back test of one of the Rocket I/O ports and writes to the test result to the GPIO port. It is normal to see some warnings from the PLB monitor or behavioral memory models during reset, but the PLB/OPB/DCR monitors should not report any protocol errors during simulation (warnings and notes may occur depending on the circumstances). You can modify the **sim_params.v** file to stop the simulation at a desired time or press **Ctrl c** (break).

11.2 BFM Simulations

The steps to run a BFM simulation are described below:

Table 19 - Instructions for Running Functional Simulations (BFM)

Step	Action
1	Edit the **flow.cfg** file: $ cd $V2PRO/platforms/Memec_Desgin_V2Pro_Board $ <*text editor*> flow.cfg (On a Windows PC, you can use the Explorer to browse for this file and edit it.) The <*text editor*> can be a program like Emacs, vi, or WordPad. Make sure that the PPC_MODEL entry is set to "swift". Save and close the **flow.cfg** file.
2	Invoke the design flow engine to start the simulation: $ make func_sim (On a Windows PC, you can also select **Start > Programs > Virtex-II Pro Developer's Kit > Reference Platforms > Memec_Desgin_V2Pro_Board > Make Functional Simulation**) The flow engine begins by compiling and linking the application software (as necessary) to generate a binary executable program file (in **.elf** format). The **.elf** file is copied locally and the **data2bram** is invoked to generate the necessary BRAM initialization command files. The simulator tool is then launched along with terminals for the UART and LCD.
3	Run the simulation: In the MTI window, type **run -all**. The simulator will test each UART in the system and if the test is successful, "Testing Uart 1" is displayed in "UART1 Terminal" and "Testing Uart 12" is displayed in the "UART2 Terminal". Upon successful completion of the LCD test, "Testing LCD" in printed on the "LCD Terminal". The simulator will also perform a loop back test of one of the Rocket I/O ports. The simulation stops when the BFL script is finished executing or when an error occurs. If the simulation completes successfully, the simulator displays the following message: **Synch 31 received… Simulation Completed** If an error is detected due to a protocol violation reported by a bus monitor or a read comparison error, the simulation stops and an error message is displayed. It is a useful exercise to view the simulator's waveform display and correlate the commands in the **.bfl** script with the bus transaction waveforms over PLB, OPB, and DCR.

12 Instructions for Synthesizing the Design

The PPC Reference Design can be synthesized into FPGA primitive components. This section describes the necessary steps for synthesizing the design using the Synplify synthesis tool. It assumes the Virtex-II Pro™ Design Kit is properly installed and that the **$V2PRO/platforms/ Memec_Desgin_V2Pro_Board /flow.cfg** file has been properly edited to correctly call the user's synthesis tool. The following figure shows a high-level view of the synthesis flow.

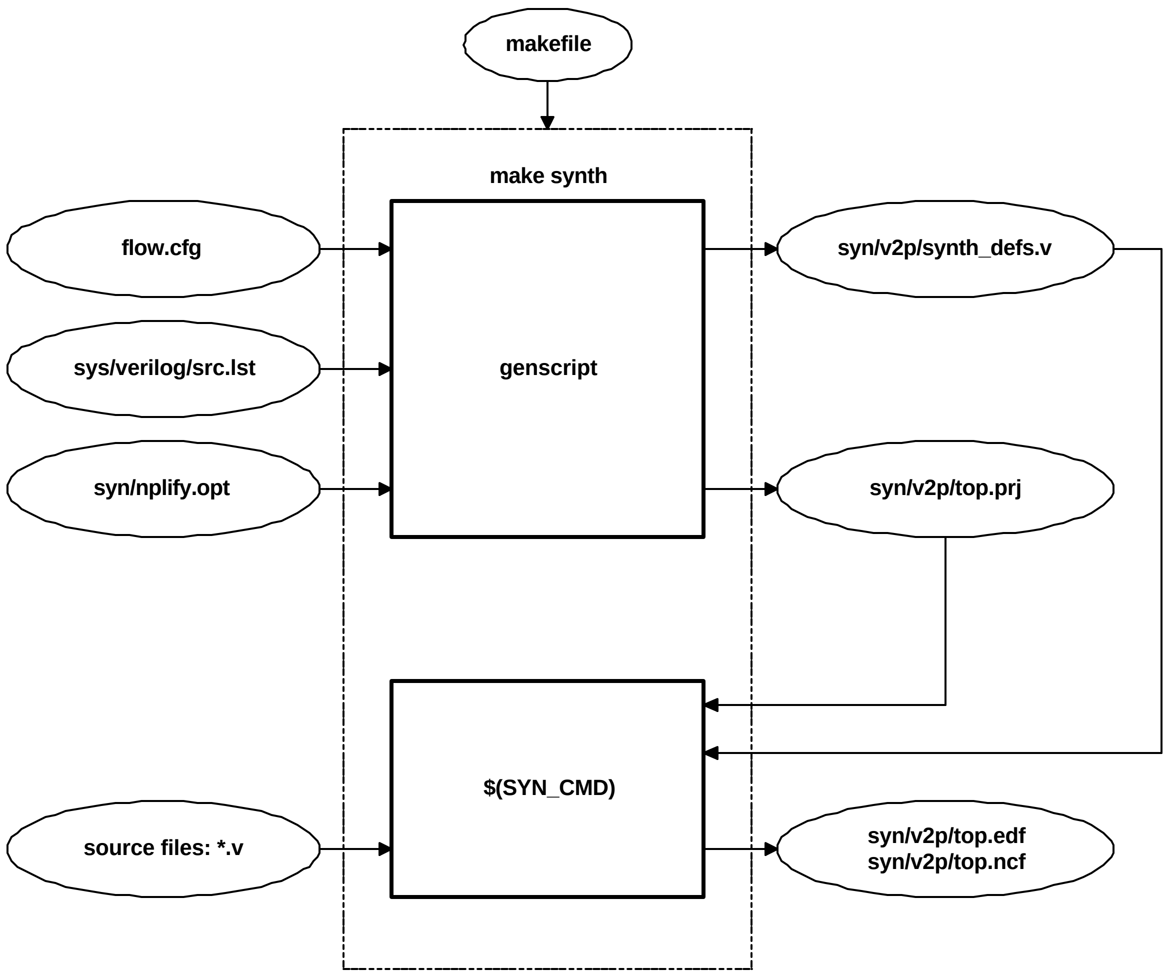

Figure 8 – Synthesis

Table 20 - Instructions for Synthesizing the Design

Step	Action
1	Invoke the design flow engine to start the synthesis: $ cd $V2PRO/platforms/Memec_Desgin_V2Pro_Board $ make synth (On a Windows PC, you can also select **Start > Programs > Virtex-II Pro Developer's Kit > Reference Platforms > Memec_Desgin_V2Pro_Board > Make Synthesis**) This script creates a project file for the synthesis tool and invokes it. For Synplify, the GUI is launched.
2	If using Synplify, click the Run button to start the synthesis process. **Note**: Any changes saved to the Synplify project file (**top.prj**) will be overwritten the

	next time "make synth" is called. You should transfer any changes in the project file to the **synplify.opt** file.
3	After completion, close the Synplify GUI. You can edit the **$V2PRO/platforms/ Memec_Desgin_V2Pro_Board /syn/synplify.opt** file to add or modify any of the synthesis options. You can also edit the **$V2PRO/platforms/ Memec_Desgin_V2Pro_Board /flow.cfg** file to change the target part. (The default part is xc2vp4fg456-7).

13 Instructions for FPGA Implementation

After synthesis, The PPC Reference Design can be targeted into an FPGA using the Xilinx implementation tools. This section describes the necessary steps for implementing the design. It assumes the Xilinx ISE software is properly installed. The following figure shows a high-level view of the implementation flow.

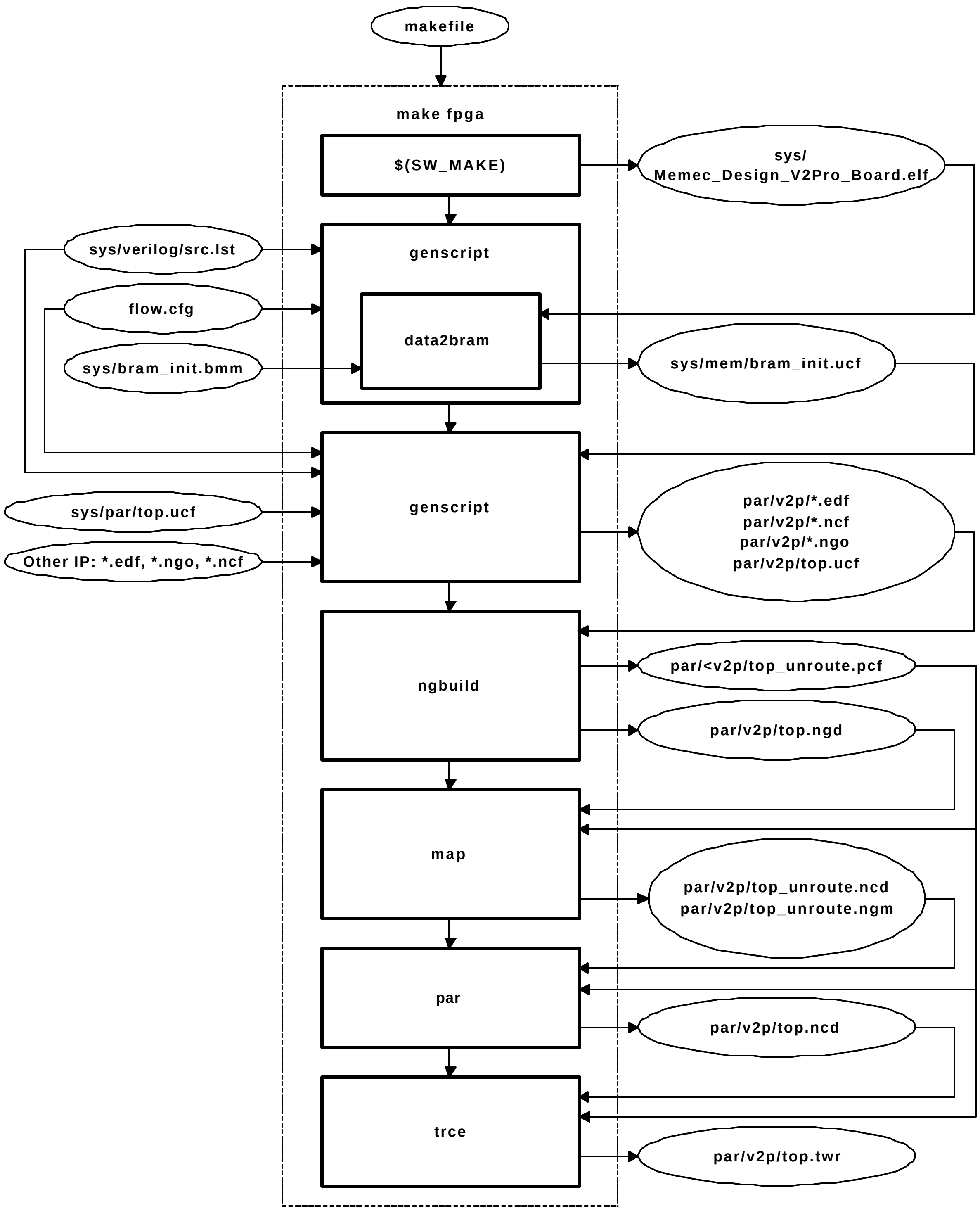

Figure 9 – Implementation

Table 21 - Instructions for Implementing the Design

Step	Action
1	Invoke the design flow engine to start the implementation process: $ cd $V2PRO/platforms/ Memec_Desgin_V2Pro_Board $ make fpga (On a Windows PC, you can also select **Start > Programs > Virtex-II Pro Developer's Kit > Reference Platforms > Memec_Desgin_V2Pro_Board > Make FPGA**). The script then copies the synthesis output files to the **$V2PRO/platforms/ Memec_Desgin_V2Pro_Board /par/v2p** directory and runs through the implementation tools (**ngdbuild**, **map**, **par**, **trce**). Any necessary net lists (**.edf** , **.edn** , **.ngc** , **.ngo**) for devices in the system are also copied in for **ngdbuild** to process. The timing constraints used by the implementation tools are copied from **$V2PRO/platforms/ Memec_Desgin_V2Pro_Board /par/top.ucf** along with any BRAM initialization commands passed in from **data2bram**.
2	With a successfully routed design, you can generate a bitstream: $ cd $V2PRO/platforms/ Memec_Desgin_V2Pro_Board $ make bit This generates a bitstream for the targeted Virtex-II Pro™ device. Note that the UART Core is only an evaluation version with a fixed functionality. It will timeout and become non-functional 240 clock cycles after the FPGA is configured.

14 Instructions for Running Back-Annotated Timing Simulations

A placed and routed FPGA design can be simulated with full timing back-annotation. This section describes the necessary steps for simulating the back-annotated design using SWIFT model of the processor block. In general, the steps are identical to those of a functional simulation except that the "make ba_sim" command is used. The following figure shows a high-level view of the back-annotated timing simulations flow.

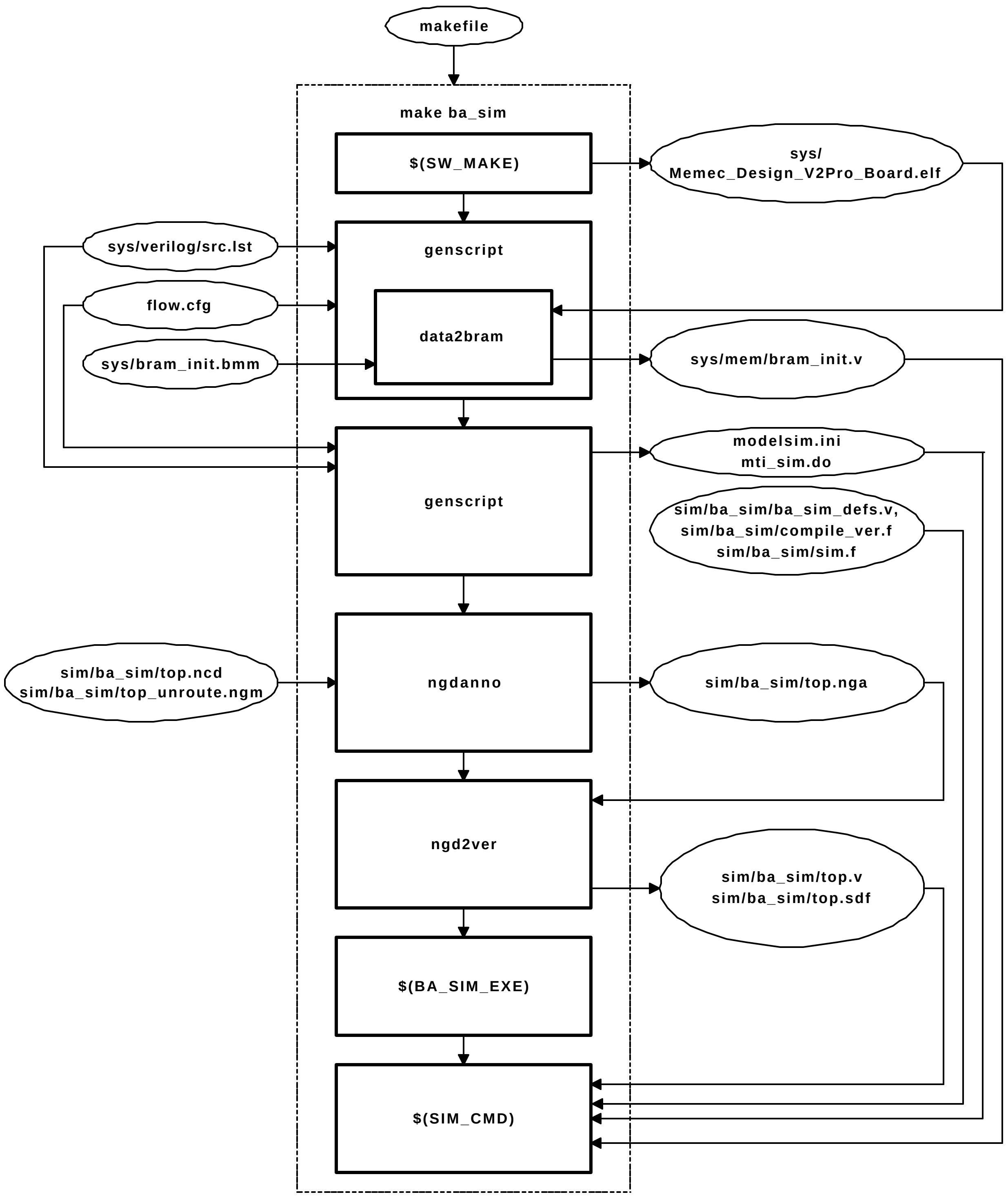

Figure 10 - Back-Annotated Timing Simulations

14.1 SWIFT Simulations

The steps to run a SWIFT simulation are described below:

Table 22 - Instructions for Running Back-Annotated Timing Simulations

Step	Action
1	Go to the main directory of the design and edit the **flow.cfg** to set the processor block model: $ cd $V2PRO/platforms/ Memec_Desgin_V2Pro_Board $ <*text editor*> flow.cfg (On a Windows PC, you can use the Explorer to browse for this file and edit it.) Make sure that PPC_MODEL = swift. After editing the **flow.cfg** file, save it.
2	Invoke the design flow engine to start the simulation: $ make ba_sim (On a Windows PC, you can also select **Start > Programs > Virtex-II Pro Developer's Kit > Reference Platforms > Pro_Board > Make Timing Simulation**) The flow engine compiles/links the software, runs **data2bram**, and invokes the Xilinx tools for back-annotation (**ngdanno**, **ngd2ver**). It then launches the simulator tool along with terminals for the UART and LCD.
3	Run the simulation: In the MTI window, type **run -all**. The simulator will test each UART in the system and if the test is successful, U1 is displayed in "UART1 Terminal" and U2 is displayed in the "UART2 Terminal". Upon successful completion of the LCD test, "LCD" in printed on the "LCD Terminal". The simulator will also perform a complete loop back test of one of the Rocket I/O ports and writes to the test result to the GPIO port.

15 Flash and SRAM Interface IP Core (IPIF Application)

The Intellectual Property InterFace (IPIF) is designed to ease the creation of new IP, as well as the integration of existing IP into the Virtex-II Pro based designs. This section describes how IPIF can be used to interface the 32-bit Flash and SRAM that reside on the Memec Design P160 module to the PowerPC processor via the PLB bus.

Intellectual Property InterFace (IPIF) modules simplify the development of CoreConnect™ devices. The IPIF converts complex system buses, such as the PLB or OPB, into common interfaces, such as an SRAM protocol or a control register interface. This makes IPIF modules ideal for quickly developing new bus peripherals, or converting existing IP to work in a CoreConnect bus-based system. The IPIF modules provide point-to-point interfaces using simple timing relationships and very light protocols.

The IPIF is designed to be bus-agnostic. This allows the back-end interface for the IP to remain the same while only the bus interface logic in the IPIF is changed. It therefore provides an efficient means for supporting different bus standards without change to the IP device.

The PPC Reference Design utilizes the SRAM Protocol IPIF to interface the Flash/SRAM IP core to the PLB bus. The following figure shows how this core is connected to the SRAM Protocol IPIF.

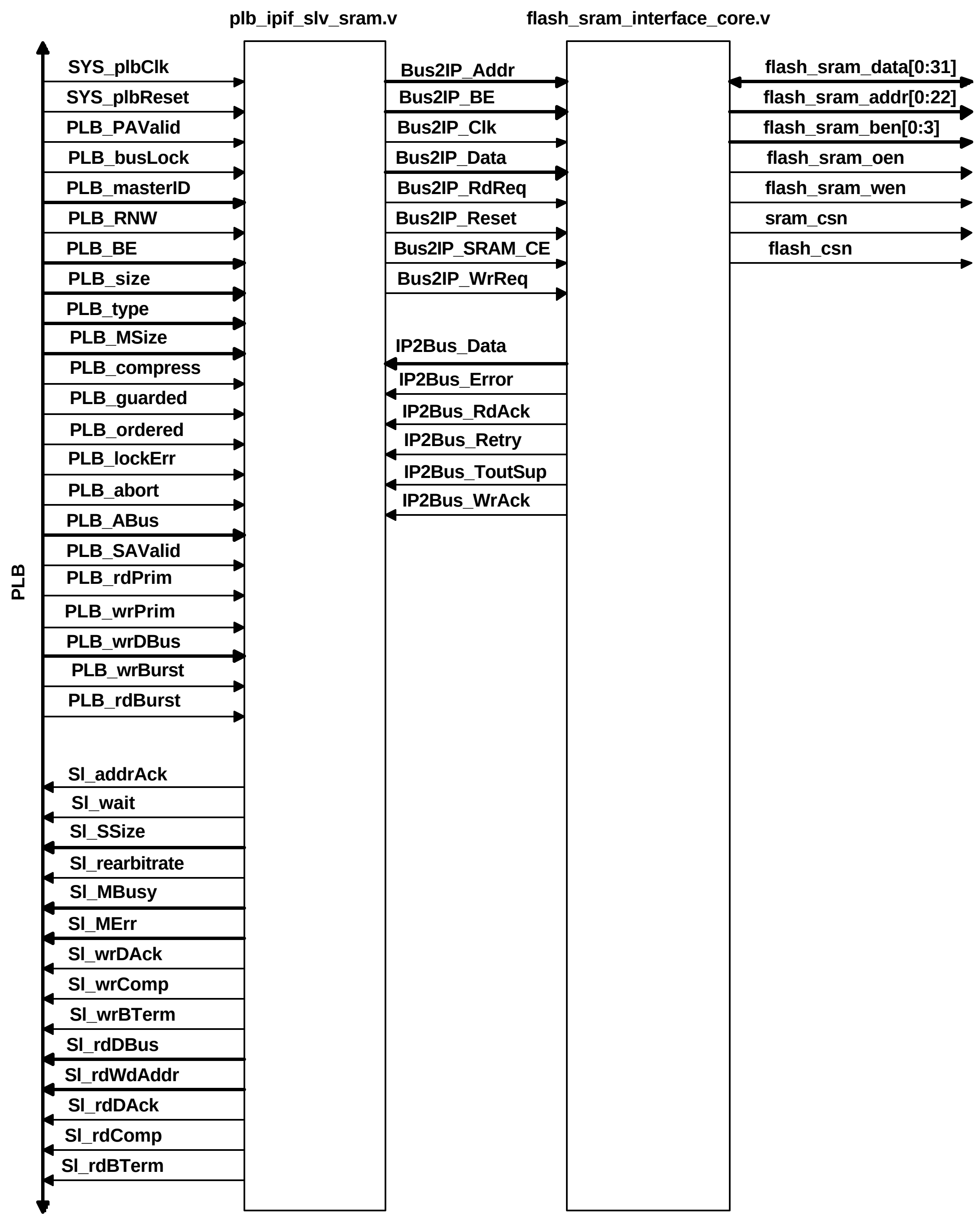

Table 23 - Flash and SRAM Interface IP Core

15.1 Flash and SRAM Read Cycle

The following figure shows the Flash and SRAM read cycle using the SRAM protocol IPIF. The read cycle begins when the IPIF asserts the BUS2IP_RdReq signal along with the Bus2IP_SRAM_CE signal on the rising edge of the BUS2IP_Clk. Upon activation of these signals, the Flash/SRAM Interface Core returns the IP2BUS_RdAck after 11 BUS2IP_Clk clocks. The Flash/SRAM Interface Core adds sufficient number of wait states into the read cycle to meet the timing requirements of the 90ns Flash/SRAM located on the Virtex-II Pro development board.

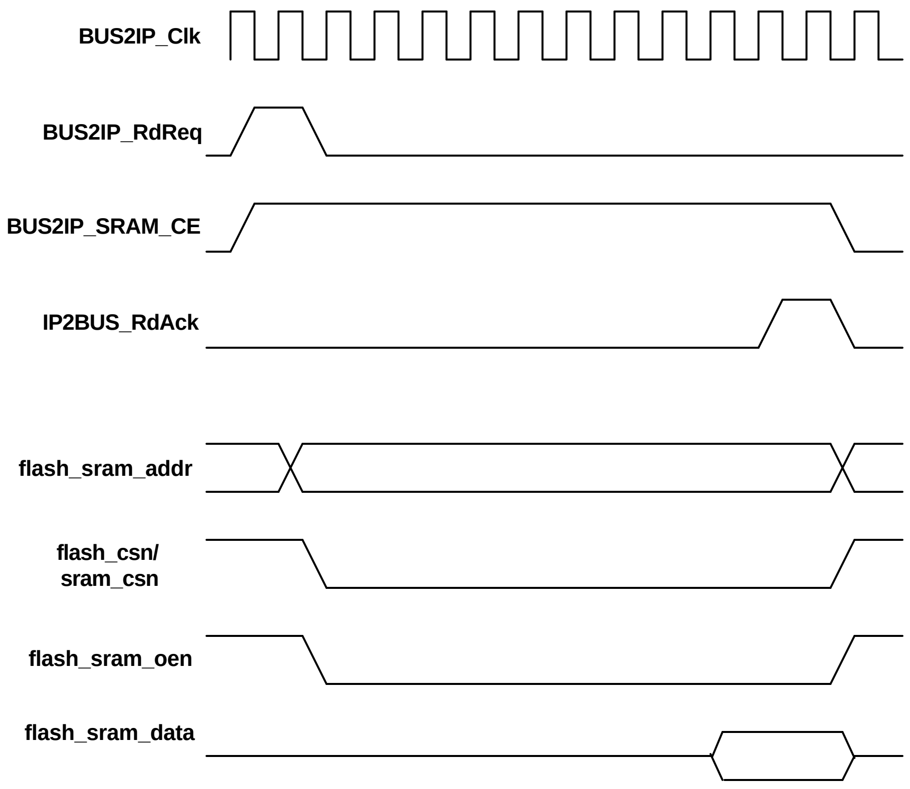

Figure 11 - Flash and SRAM Read Cycle

15.2 Flash and SRAM Write Cycle

The following figure shows the Flash and SRAM write cycle using the SRAM protocol IPIF. The write cycle begins when the IPIF asserts the BUS2IP_WrReq signal along with the Bus2IP_SRAM_CE signal on the rising edge of the BUS2IP_Clk. Upon activation of these signals, the Flash/SRAM Interface Core returns the IP2BUS_WrAck after 11 BUS2IP_Clk clocks. The Flash/SRAM Interface Core adds sufficient number of wait states into the write cycle to meet the timing requirements of the 90ns Flash/SRAM located on the Virtex-II Pro development board.

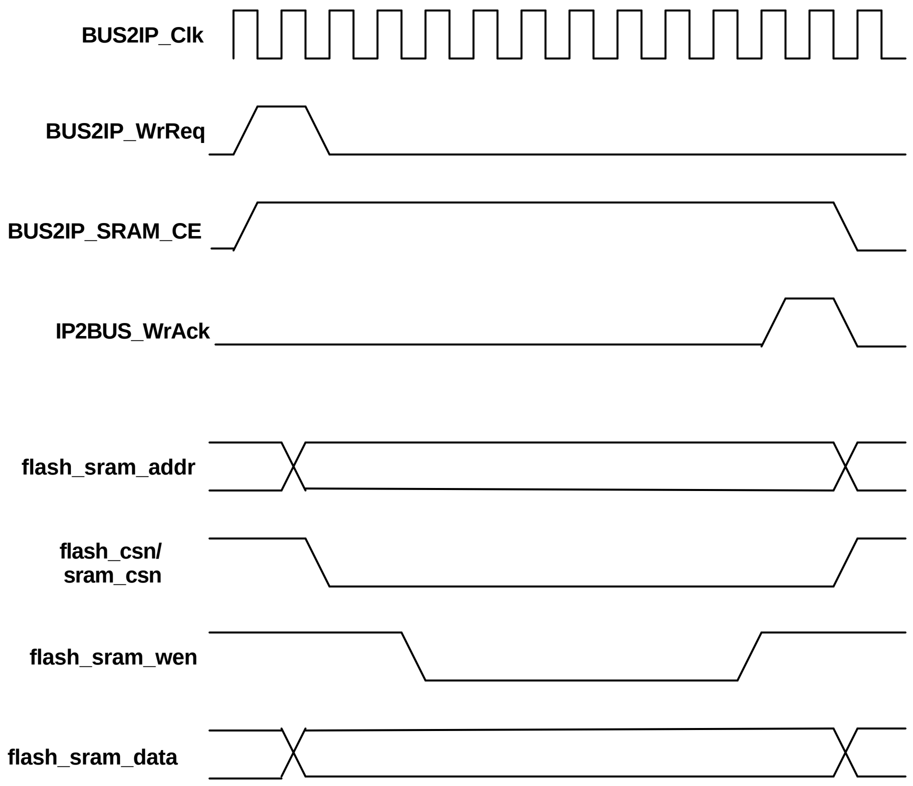

Figure 12 - Flash and SRAM Write Cycle