XAPP806 (v1.2) June 5, 2007

Determining the Optimal DCM Phase Shift for the DDR Feedback Clock

Abstract

This application note describes how to build a system that can be used for determining the optimal phase shift for a Double Data Rate (DDR) memory feedback clock. In this system, the DDR memory is controlled by a controller that attaches to either the OPB or PLB and is used in an embedded microprocessor application. This reference system also uses a DCM that is configured so that the phase of its output clock can be changed while the system is running and a GPIO core that controls that phase shift. The GPIO output is controlled by a software application that can be run on a PowerPC® 405 or Microblaze™ microprocessor. This application is run out of an internal FPGA block RAM (BRAM) and it traverses the entire range of possible phase shifts (-255 to 255) that correspond to the complete 360 degree phase shift cycle. At each phase shift value, the application performs memory tests and records if the tests passed or failed. The passing ranges are reported by printing them to a hyperterminal through a UART. The optimal phase shift values are simply calculated by choosing the middle values for the passing ranges.

Included Systems

Included with this application note are two reference systems:

- PPC_ph_shift - Built for ML310, uses PowerPC 405 procesor and DDR on PLB
  www.xilinx.com/bvdocs/appnotes/xapp806_PPC_ph_shift.zip
- MB_ph_shift - Built for ML310, uses MicroBlaze processor and DDR on OPB
  www.xilinx.com/bvdocs/appnotes/xapp806_MB_ph_shift.zip

Introduction

When building a new circuit board that uses an FPGA and a DDR (Double Data Rate) memory, the user must know the optimal phase shift for the DDR feedback clock because the double data rate memory controllers write and read data at both the falling and rising edges of the controller clock. Therefore, the optimal time to sample the data is in the middle of each of the half clock cycles, or 90 and 270 degrees (See Figure 1). To ensure that the sampling occurs at the desired times, it is necessary to account for the routing delay of the DDR feedback clock. In a Xilinx FPGA system this is accomplished with the help of a Digital Clock Manager (DCM). The DCMs that are available in most Xilinx FPGAs have a built in phase shifter component. If the DDR feedback clock is fed into a DCM, the phase of the output clocks can be adjusted so that the 90 and 270 degree shifted clocks rise in the middle of the controller clock half periods. The only difficulty with this approach is knowing exactly how much to shift the phase. This amount is dependent not only on how much routing delay there is from the DDR chip to the FPGA, but also how much delay there is from the FPGA input pin to the DCM. This application note explains how to create a very simple system to experimentally find this optimal phase shift without having to know anything about these routing delays.

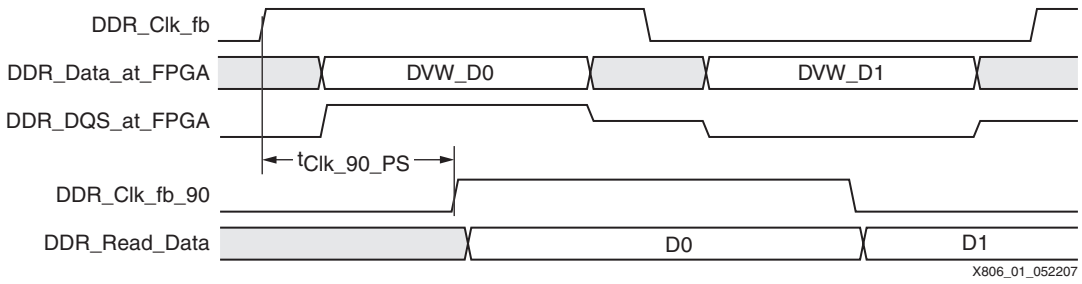


Figure 1: **DDR Data Capture**

Hardware Requirements:

Xilinx Virtex™-II Pro ML310 development board.

Serial cable: 9-pin female to 9-pin female “null modem” (cross-over).

Platform USB Cable or Xilinx Parallel Cable 4 (with flat ribbon cable).

Software Requirements:

Xilinx Embedded Development Kit (EDK). Run this project on the same version of EDK in which it was installed, or later.

Xilinx ISE™ (see the *Getting Started With EDK* for the required version of ISE).

HyperTerminal (or similar) for host communication with the embedded system.

Building the System

This application note accompanies two reference systems that were built on the ML310 development board.

The first system uses the embedded PowerPC (PPC) as the microprocessor and the DDR memory controller attaches to the Processor Local Bus (PLB).

The second system uses MicroBlaze (MB) for the microprocessor and the DDR memory controller attaches to the On-Chip Peripheral Bus (OPB).

See Figure 2 for block diagram of the PowerPC 405 based phase shift reference system and Figure 2 for the MicroBlaze based reference systems. These two systems encompass the two most common microprocessor/memory setups used in Xilinx embedded systems.

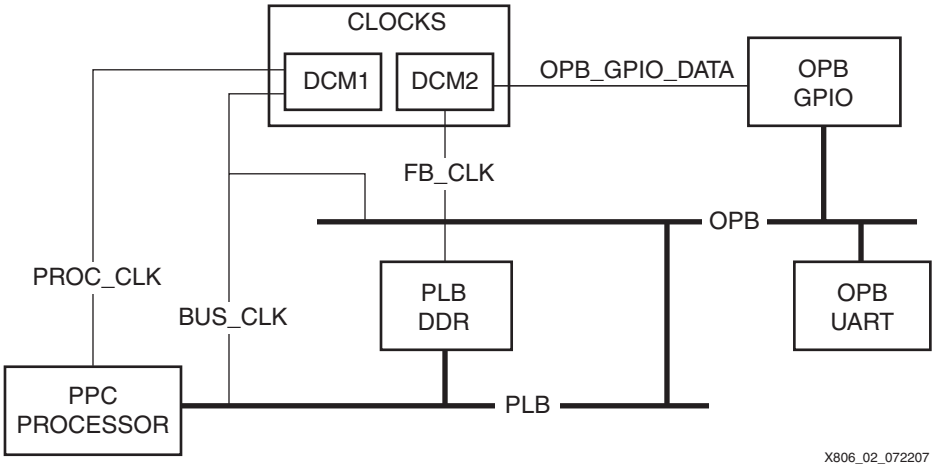


Figure 2: **PPC Reference System Block Diagram**

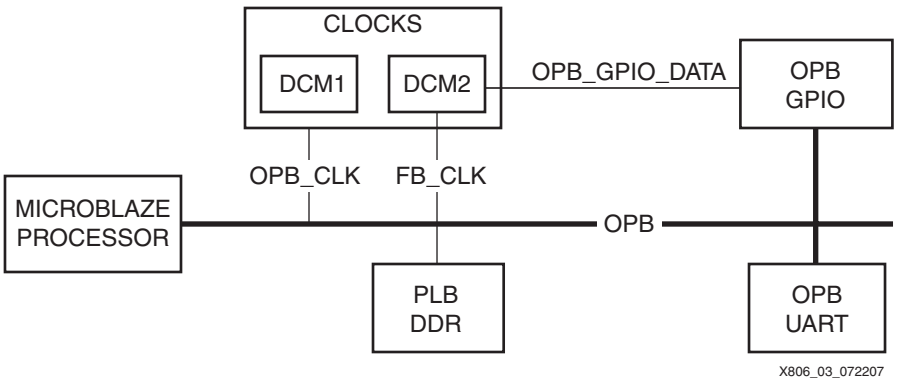


Figure 3: **MB Reference System Block Diagram**

The bitstreams and the compiled software application for each of these systems are available in `ready_for_download/` under the project root directory.

Although these reference systems are specifically targeted to work on the ML310 board, they can be modified to work on any other board that contains an FPGA and a DDR memory chip. This application note describes in detail how to go through this process. Additionally, it describes the process of building a system through the Base System Builder (BSB) which is available through the Embedded Development Kit (EDK) and modifying that system to determine the optimal phase shift for the DDR feedback clock.

Building the Initial System Through BSB

1. Start the BSB wizard to create a new EDK project.
2. Select one of the boards supported by BSB or create a project for a custom board. Choose the FPGA that is available on the board and define whether to use a PowerPC or a MicroBlaze soft processor is to be used in the system. The two reference systems that accompany this application note provide examples for either choice. No debug module is needed for this system, however it can be added if desired.
3. Choose the frequencies for the reference, processor and bus clocks (the PPC reference system has the OPB and PLB clocks set to **100** MHz and the processor clock set to **300** MHz; the MB reference system has all the clock frequencies set to **100** MHz).

Note: Although these frequencies should not affect the optimal phase shift for the DDR feedback clock, it is best to create this system so that it uses the same frequencies as your final design.

4. Add a BRAM block on OPB or PLB or LMB. Although the reference systems use **64** KB of memory, only **16** KB is required. Run the application out of local BRAM to preclude the failure of the data reads from the DDR memory at certain phase shifts. If the application itself resided in DDR memory, the microprocessor would not be able to fetch instructions and the application would hang.
5. Make certain that the other peripherals that are needed for this application to run are present — a DDR memory controller, a UART, and a GPIO core.
6. Set the DDR controller parameters up so that they correspond to the parameters of tge DDR chip.
7. The reference systems use a UART 16550 attached to the OPB. The UART communicates the optimal phase shift values to the user through a HyperTerminal.
8. Set up the GPIO core so that it is bidirectional and has a width of 2 bits.
9. Additional peripherals may be added to the system, but are not necessary for the phase shifting application to run.
10. Ensure that the instructions, data, and the stack for the application program are all contained within the processor local memory block or BRAM.
11. Direct **STDIN** and **STDOUT** to the UART receive and transmit ports. Create a sample application and a linker script using the Base System Builder .
12. The initial base system has now been created. It may be modified to run the application that determines the optimal phase shift for the DDR memory feedback clock.

Note: If the system was created for a custom board, edit the system constraints file data/system.ucf to lock down the UART and DDR pins. Additionally, it may be necessary to set the correct position number of the FPGA in the JTAG chain in etc/download.cmd (default is 1).

Although not necessary, it is suggested to build this initial system and run the sample application built by BSB to make sure that the system is set up correctly and that the location constraints are what they are supposed to be.

Overview of Modifications Required to the Initial System

Ensure that the initial build is as described in the previous section before starting through the instructions in this section. The initial build does not have to be identical to the one described in the previous section, but it must have a processor, local memory, a DDR controller, a UART, and a GPIO.

Two modificatons must be made to the initial build so that it can be used in the phase shifting application:

1. Change the clocking so that the phase shifting of the DCM that is fed by the DDR feedback clock is variable and connect that DCM phase shifting control inputs to the outputs of the GPIO core.

2. Change the sample application so that it performs the phase shifting by controlling the outputs of the GPIO.

These modifications are described in the following paragraphs.

Modifying the Clocking Structure and Other Related Hardware

1. Copy the clocks module from the pcores area of the appropriate reference system to the pcores area of the project.
   a. If the initial system uses PPC, copy the entire `clocks_v1_00_c` module from `PPC_ph_shift/pcores/` to the pcores area under the project.

b. If the project uses MB, copy the `MB_ph_shift/pcores/clocks_v1_00_c` directory under the pcores directory of the project.

2. Look in the IP Catalog tab of the Project Information Area window under the Project Repository section to confirm that the clocks module that were just added are listed. If they do not appear in the list of available modules, XPS must be resynchronized or closed and reopened to see the clocks module listed.

3. Right click on **clocks**, then **Add IP** as shown in Figure 4 to add the clocks module to the list of used peripherals.

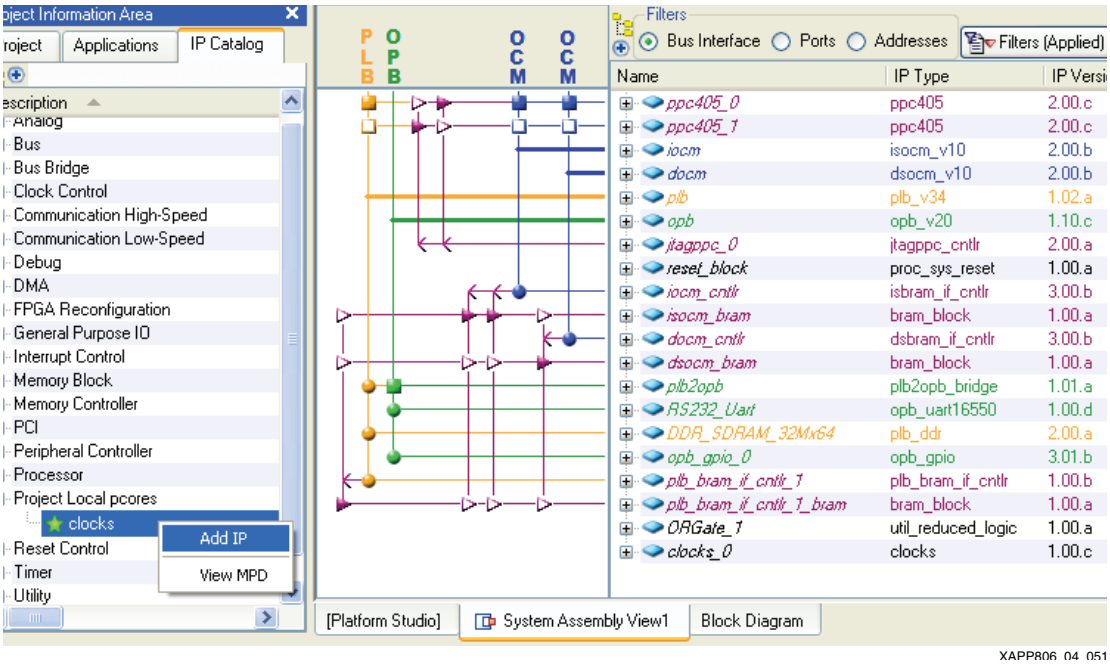


Figure 4: **Add Clocks module**

4. Choose the ports filter in the System Assembly View window so that the new clocks module ports can be connected. The clocks inputs and the outputs of the module and the required port connections are described in Table 1

Table 1: **Clocks Module Ports and Connections**

Port Name	Connection(1)	Description
dcm_clk	dcm_clk_s	**Input:** raw system clock. It must be tied to the signal that is connected to the clk pin of the FPGA
dcm_rst_n	sys_rst_s	**Input:** system reset pin. It must be tied to the signal that attaches to the reset pin of the FPGA
sys_clk	sys_clk_s	**Output**: bus clock. It is assumed that if both the PLB and OPB are in the system, that they are running synchronously and at the same frequency, thereby allowing the same clock to be used for both the OPB and PLB clocks
sys_clk_n	sys_clk_n_s	**Output:** 180 degrees out of phase bus clock
sys_clk90	clk_90_s	**Output**: 90 degrees out of phase bus clock
sys_clk270	clk_90_n_s	**Output**: 270 degrees out of phase bus clock
ddr_fb_clk	ddr_feedback_s	**Input**: DDR memory feedback clock
ddr_fb_clk90	ddr_clk_90_s	**Output**: DDR feedback clock 90 degrees out of phase. This clock must be phase shifted so that it rises in the middle of the first half of the clock cycle

Table 1: **Clocks Module Ports and Connections *(Continued)***

Port Name	Connection(1)	Description
ddr_fb_clk270	ddr_clk_90_n_s	**Output** DDR feedback clock 270 degrees out of phase. This clock must be phase shifted so that it rises in the middle of the second half of the clock cycle
dcm_locked	dcm_2_lock	**Output**: ANDed signal of the two DCM locks used in the clock module. In a PPC system, tie this signal to the dcm_locked signal on the proc_sys_reset module. BSB ties this signal to the LOCKED output of the last DCM used. For example, if the system contains three DCMs, it would probably be tied to dcm_2_lock. In an MB system, this output does not need to be tied to anything
proc_clk	proc_clk_s	**Output**: processor clock used in PPC systems. It exists only in the clock module copied from PPC_ph_shift project
ps_in	opb_gpio_data_out_0	**Input**: phase shifting input. It is driven by one of the GPIO outputs to signal the DCM when it must shift to the next phase. Tie this input signal to the signal that is connected to GPIO_d_out of the 2-bit wide GPIO core (bit 0).
ps_incdec	opb_gpio_data_out_1	**Input**: signal that controls whether the phase of the DCM outputs is incremented or decremented. Tie this signal to the other bit of the GPIO_d_out port of the 2-bit wide GPIO core (bit 1).

Notes:

1. These are the default names of the signals as created by BSB. The actual names of the connected signals could vary from system to system

All the ports for the clocks module have been added and connected as described in the above paragraphs. Note that the clocks module signals, aside from **ps_in** and **ps_incdec**, should be tied to signals that exist in the design. They are connected to the inputs or outputs of the DCMs that are currently in the design. If the signals cannot be seen in the drop-down-menu, it is likely that the user is attemptying to connect to a wrong signal.

Figure 5 illustrates the ports filter of the System Assembly view window with the ports for the clocks module filtered.

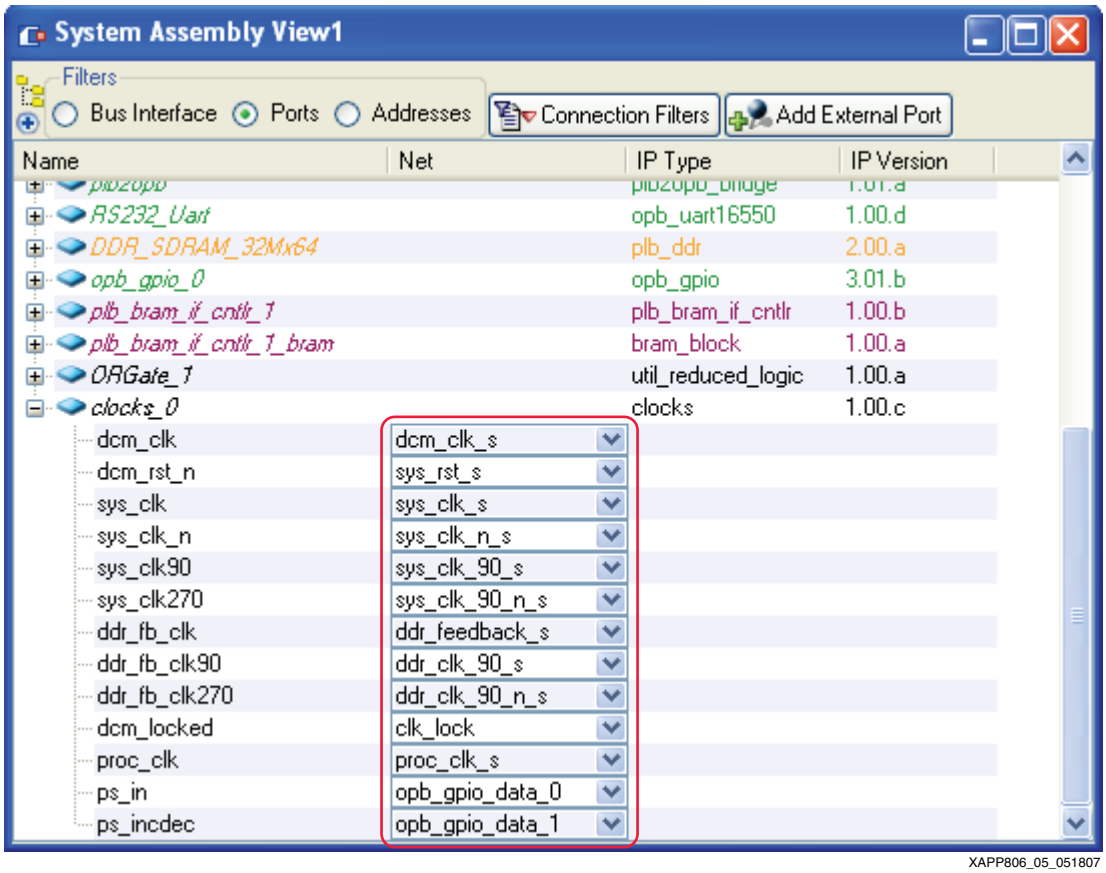
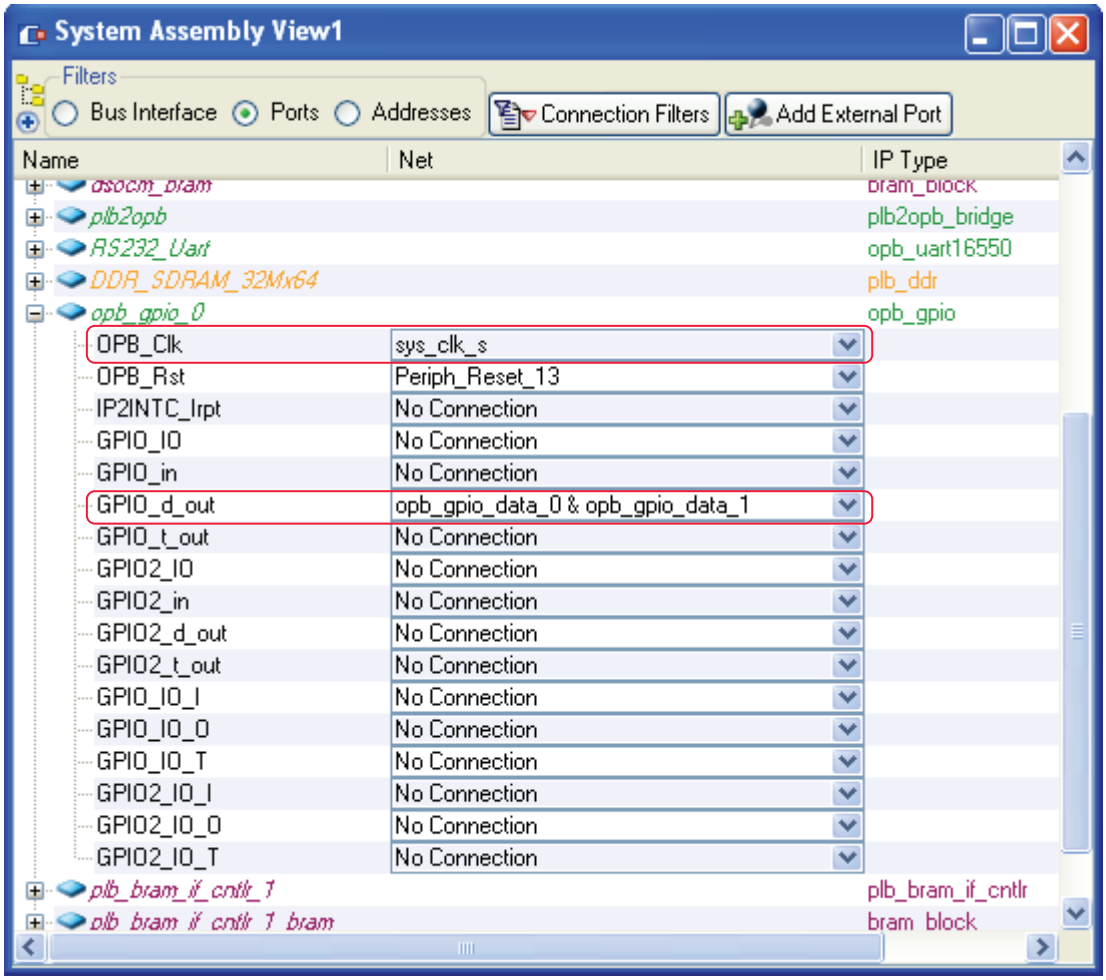


Figure 5: **Clocks module port connections**

2. As shown in Figure 5, the **ps_in** and **ps_incdec** ports from the clocks module are connected to **opb_gpio_data_out_0** and **opb_gpio_data_out_1**, respectively. This 2-bit wide **gpio_data_out** signal then must to be connected to the **GPIO_d_out** signal of the **GPIO** core. To do so, filter the ports for the GPIO core instance. Delete all the connected **opb_gpio** ports, except **OPB_Clk,** then add the **GPIO_d_out** port back into the list of internal port connections. Connect that port to **gpio_data_out**. The resulting connections for the GPIO core should look as shown in Figure 6. Ensure that the changes have been applied.

Note: An external port connection for gpio_data_out is not necessary, however the user may want to connect it to a set of LEDs if they available on the board. The reference designs that accompany this application note do not have an external port connection for the gpio_data_out signals.



XAPP806_06_051807

Figure 6: **OPB GPIO module port connections**

3. In the peripherals tab of the Add/Edit Cores dialog, select and delete the DCM that created the clocks used by the DDR memory controller, the DCM that created the bus clock (also the **90, 180** and **270** degrees out of phase bus clocks), and the DCM that created the processor clock, if applicable. These DCMs can now be deleted from the design because the DCMs that generate these clocks are already included in the clocks module that was integrated into the design. The remaining DCMs in the design can be left unchanged or ported to be included in the clocks module. The modification of the clocks module is not covered by this application note.

The PPC reference system has the OPB and PLB clocks set to **100** MHz and the processor clock set to **300** MHz, while the the MB reference system has the clock frequencies set to **100** MHz. If the system needs to use clocks of different frequencies, edit the verilog source of the clocks module. For example, if the system clock is to have a period of **12.5** ns (frequency of **80** MHz), change:

```
// synthesis attribute CLKIN_PERIOD          of dcm1 is "10"
// synthesis attribute CLKIN_PERIOD          of dcm2 is "10"
```

to

```
// synthesis attribute CLKIN_PERIOD          of dcm1 is "12.5"
// synthesis attribute CLKIN_PERIOD          of dcm2 is "12.5"
```

As another example, to change the frequency of the processor clock, make the following changes

```
// synthesis attribute CLKFX_DIVIDE          of dcm1 is "1"
// synthesis attribute CLKFX_MULTIPLY        of dcm1 is "3"
```

so that the MULTIPLY/(DIVIDE*CLKIN_PERIOD) yields the desired frequency. In the PPC_ph_shift reference system, these parameters are set to yield 3/(1*10) = **300** MHz for the processor clock frequency.

It is suggested to change the parameters in the **synthesis xc_props** and **defparam** areas of the appropriate DCMs.

The Software Application

Preparing the Software Application

1. It is assumed that the initial system that was built has an accompanying sample application, and that the compiler options are set so that BRAMs are initialized at build time.
2. If during the creation of the initial system BSB, the user was asked to create a sample application (as instructed in the building the initial system section), the user need not be concerned about this because a linker script is created automatically. The linker script sets up all the compile and link time options. If there is no linker script, set the compiler options or create a linker script to ensure that the application is compiled correctly and initialized into the correct area of the BRAM.
3. The sample application that was created by BSB must be replaced by the application from one of the reference systems that accompany this application note. Because the applications for the MB and PPC systems are identical, either copy can be used. Copy `PPC(MB)_ph_shift/TestApp/src/TestApp.c` under the `/TestApp/src/` area of the project. This action will overwrite the TestApp.c file that was created by BSB. The only items in TestApp.c that might have to be changed are the XPAR parameter names for the GPIO, UART and DDR cores, and the UART parameters.

For example, if the reference systems have the GPIO instance named OPB_GPIO_0, the same name for the instance in the application source would use the same name.

```
#define GPIO_0_BASEADDR XPAR_OPB_GPIO_0_BASEADDR
```

If in the system, the GPIO core is named **MY_GPIO_7**, change the above statement to:

```
#define GPIO_0_BASEADDR XPAR_MY_GPIO_7_BASEADDR
```

In the provided application, the UART uses a baud rate of **9600**, which can be changed, if desired.

DCM Phase Shift Application

The application source is simple and it is fully commented to let the user know what is done at each of the steps. In summary, the application starts with the phase of the DDR feedback DCM output phase set to **0**. The phase is then decremented to **-255** (the lowest possible phase shift), by pulsing the OPB GPIO output **255** times with the increment/decrement signal set to decrement. This information is passed to the DCM that actually shifts the phase. Once the phase is set to **-255**, the phase is incremented one by one all the way to **255** (the highest possible phase shift). At each phase value, the application performs **32-bit**, **16-bit**, and **8-bit** memory tests, then records if the tests passed or failed. Because the DDR memory controller writes and reads data during both the negative and positive portions of the clock cycle, two passing ranges can be expected — one during the positive half and the other during the negative half.

Running the Application to Determine the Optimal Phase Shift

Ensure that the bitstream for the project has been updated and the application has been initialized into BRAM.

Start a hyperterminal and set it up so that it is connected to the COM port that is connected to the UART on the board. Set the HyperTerminal to the Bits per second of **9600**, Data Bits to **8**, Parity to **None,** and Flow Control to **None** as shown in Figure 7. The UART terminal is used to capture the results of the phase shift test.

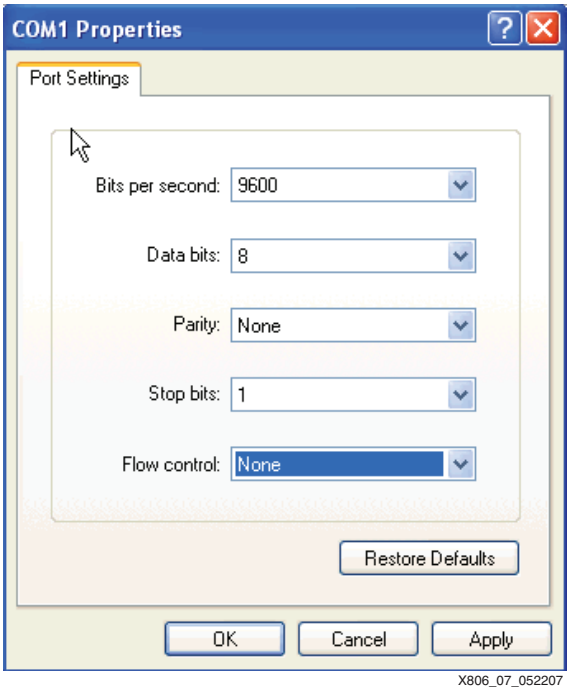


Figure 7: **HyperTerminal Settings**

Executing the Reference System using the Pre-Built Bitstream and the Compiled Software Applications

To execute the system using files inside the `ready_for_download/` in the project root directory, follow these steps:

1. Change directories to the `ready_for_download` directory.
2. Use iMPACT to download the bitstream by using the following:

   ```
   impact -batch xapp806.cmd
   ```

3. Invoke XMD and connect to the respective processor by using the command:

   ```
   xmd -opt xapp806.opt
   ```

4. Download the executables by using the command:

   ```
   dow executable.elf
   ```

Executing the Reference System from EDK

To execute the system using EDK, follow these steps:

1. Open `system.xmp` inside EDK.
2. Use **Hardware→Generate Bitstream** to generate a bitstream for the system.
3. Download the bitstream to the board with **Device Configuration→Download Bitstream**.

Because the BRAMs were initialized at build time, the application should start running as soon as the downloading is completed. The output should look similar to this:

DCM PHASE SHIFT TEST

_____

Moving the initial phase to -255

.....

.....

.....

..... Done

Incrementing through all the phases and reporting the passing ranges

Passing range is -255 to -169, optimal phase shift for this range is -212

Passing range is -1 to 86, optimal phase shift for this range is 42

Set your DDR feedback clock phase shift to 42

Finished testing

The application should report two passing ranges for the reasons discussed at the end of the section that describes how to modify the application. The positive optimal phase shift is the number that needs to be used for the actual phase shift for the DDR feedback clock in your final system.

Modifying the DCM Phase Shift for the DDR Feedback Clock in Your System

Once the optimal DCM phase shift for the DDR feedback clock on the board is determined, change it in the final design, and fix the phase shift so it is no longer variable. In the case of the output from the previous section, for example, if optimal phase shift is 42, the DCM parameters should read as follows:

```
// synthesis attribute CLKOUT_PHASE_SHIFT    of dcm1 is "FIXED"
// synthesis attribute PHASE_SHIFT           of dcm1 is "42"
```

The design should also work with the phase shift of the DCM set to the optimal value from the negative range. Using that value sets the clocks 180 degrees out of phase. For example, clk_90 becomes clk_270.

It is important to note that the optimal phase shift value provided by the application is specific to the board and also the DCM used. If a new design is created that uses a different DCM, the optimal phase shift value could also change. That is why the DCM must be contrained in the design. This can be accomplished by editing the data/system.ucf constraints file. For example, the following constraint will lock DCM1 to a specific DCM in the FPGA.

```
inst "clocks_0/clocks_0/dcm1" LOC = DCM_X1Y1;
```

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
4/28/05	1.0	Initial Xilinx release.
1/08/07	1.1	Modified for EDK 8.2.2 release.
6/5/07	1.2	Updated for EDK 9.1.01i.