PCI Timer Board Documentation

Written by: Tom Brimeyer
Date: 05/23/05

Table of Contents

Basic Operation:

The PCI Timer Board is used to output pulses for use with radar. It has the ability to output six pulses, BPULSE<0:5>. Each pulse can be programmed individually as to what **delay** and **pulse width** the user would like to define. These variables are noted as configuration variables. The user can also define four attributes that are common to all pulses. The first attribute is the **period**. The period defines amount of time before the next pulse occurs for a single BPULSE. Next is the **pulse enable** attribute. This allows the user to define which of the BPULSE<0:5> they would like to output. The last two variables are **polarization** and **phase**. These variables are also outputs from the board. These four variables are referred to as the sequence variables.

A sequence can also be referred to as a PRT(Pulse Repetition Time). The board can be programmed to output up to six PRTs when running. This meaning that for every BPULSE, there can be up to six different pulse frequencies operating at a time.

For Example: Suppose there are 2 PRTs programmed to the board, one at 4 kHz and the other at 5kHz. Then for BPULSE0, there will be two output pulses observed. One pulse will be at 4 kHz and the other at 5kHz.

Both the configuration and sequence(PRT) variables are preprogrammed from onboard memory of the PIC Processor(16F874) during bootup. These values must be reprogrammed by the user. This is done using the executable program, "Timerset.exe" and a data file, "config.tmr" that holds the user defined variable information. Upon execution, the new configuration and sequence values are written overtop the old values that were programmed at bootup. This means that everytime the board is supplied power after being without power; "Timerset.exe" must be executed in order to reprogram the variables.

(Hardware)User Defined Interface:

There are many parts of the board that all interface with each other. This section will discuss in detail, the interfacing parts that are user programmed. The first of these is the "S5920 PCI Target Interface". This chip takes care of the interfacing of the PCI bus which is used to talk to the other devices. This chip is programmed using an external EEPROM. The EEPROM contains the register information that is to be loaded into the PCI chip upon bootup. It contains information like the "Device ID #" as well as register information in order to setup the PCI bus to an 8 bit bus as opposed to a 32 bit bus. This is needed because this bus must be able to correctly communicate with other devices, like the external DPRAM. This external memory stores all of the information for the configuration and sequence variables. Upon bootup, the preset variables are stored into the DPRAM. These variables can be changed by running "Timerset.exe", which will take the new values from the "config.tmr" file and store them in the DPRAM. Then the board

is reset and it will take all of the values from the DPRAM and use them accordingly. Lastly, there is a PIC Processor(16F874) that must be programmed. This processor is programmed to carry out all of the operations needed in order to read information from the DPRAM, write information to the DRAM, supply addressing to other chips, write information to other chips, program the Timers, and run the interrupt service routine(ISR) that actually programs the PRT and outputs the pulse. The PIC Processor makes up for the majority of the user defined coding.

(Software)User Defined Interface:

All of the software involved with programming and interfacing the timer board was written in C++. There are two executable programs as well as one library that are involved in the test and setup of the timer boards. The first executable program "timertest.exe" is compiled from "timertest.cpp" which has the function of ensuring that the DPRAM is being written to and read from correctly. Its main function is to write one hundred bytes of data to the DPRAM beginning at memory location 0x40. It then reads those memory locations back and compares them with what was supposed to be written. If it reads back correctly, it will pass. If it reads back incorrectly, it will tell you what was read and what was supposed to be read in order to help the user debug the problem. This program is used merely as a test for interface between the PCI Target Interface and the DPRAM.

The second executable file, "timerset.exe" is compiled from "timerset.cpp". The purpose of this program is to set the timers with user defined configuration and sequence data. The source of the data is found in the file, "config.tmr". The program scans this file and puts the data into variables. It then sorts the data before writing it to the DPRAM and reinitializing the board. Once reinitialized, the DPRAM will have the new data stored and the board will be operating accordingly with the new data. Once this program is executed, it will only hold the set data values until one of two things occur. The board will change the current data values if "timerset.exe" is run again, or the board loses power and resets. Upon losing power and rebooting, the DPRAM is initialized with data values that a preprogrammed in PIC memory.

The last software file involved is the library file "timerlib.cpp". This file contains all of the definitions for the subroutines called for by both "timertest.exe" and "timerset.exe". This is the bulk of the software code as it contains subroutines for reading "config.tmr", reading/writing from/to the DPRAM, controlling the timers via the PIC. In order to control the timers, The PIC not only polls for hardware interrupts but also polls the DPRAM for interrupts that are initiated by the software. This way, the software can stop, reset, and start the timers as needed.

In-Depth Hardware Operation:

PCI Interface:

An attempt to step through the hardware processor will now be done. As stated before, there is the "S5920 PCI Target Interface" which has the purpose of interfacing the PCI port with the CPU and other hardware. This is especially needed when using software to communicate with other hardware. This chip provides both an address bus and a data bus that is set up to one port of the dual port memory. This allows direct accessibility of the DPRAM for the PCI bus. An EEPROM chip is also provided in order to setup the PCI Target Interface upon bootup. This provides the PCI Interface with the data needed in order to set the Device ID#, data bus length, etc.

Lattice PLD:

There are also two Lattice PLDs(ISPLSI1016). These chips are basic logic programmable devices that receive an input and run the input through preset logic equations. The outputs of these logic equations are for many functions. Some outputs provide clock synchronization, address latching, and hardware interrupts.

Clock:

In order to set up the clock to run the hardware, there are three options. Each of these three options can be chosen by how the J9 jumper is situated. If J9 pins 2-3 are connected, then the signal will come from an internal oscillator. There is also the option of having an external oscillator with adjustable input impedance. If J9 pins 1-2 are connected, this supports an external oscillator from port J8 with 50 Ohm impedance. If there are no connected pins on J9, then this supports an external oscillator from port J8 with high impedance. Once the oscillator has been provided, it runs through a digital phase lock loop(PLL) that acts as a low noise synthesizer that is to be inputted into a voltage controlled oscillator(VCO). The VCO creates a 48 MHz to 64 MHz signal that is then buffered and sent through a 30 MHz low pass filter(LPF). From there, it is processed by a differential TTL/CMOS to PECL. When a clock signal has been established, there is a PLL LED that is illuminated to let the user know that the clock is provided.

PIC Processor:

Once a clock signal has been established, this will provide a clock signal for many of the hardware devices. When the clock signal is provided to the PIC Processor, it then starts its boot sequence and begins operation. The PIC begins operation by initializing all data ports to ensure that they are operating as needed. This includes the initialization of Port A, Port B, Port C, Port D, and Port E. It then initializes the Serial Peripheral Interface(SPI). Then it writes frequency data to the PLL and resets it for the needed operation.

After all ports have been initialized, the DPRAM must be initialized. This involves first, erasing all of the DPRAM memory. Then data tables are read from PIC memory and written into the allocated locations in the DPRAM. There are two tables that contain needed information. The first being the configuration data which provide the information that are to program the hardware timers for the delay and pulse. The second being the sequence data which provide the information that are to program a different hardware timer for the period of the pulses. The sequence data also contains information that will enable which pulses to be outputted as well as polarization and phase data.

Once all of the needed data is written to the DPRAM, the PIC initializes the timers. The PIC reads the needed configuration and sequence data and then sends that data to the associated hardware timers. The timers are then stopped, and restarted so that they now operated with the loaded data.

Once everything is initialized, a loop takes place that will poll for interrupts that are to be provided via software and hardware. During this process, another LED is set to blink in order to assure the user that the PIC is working and running in the loop. The simpler of the two interrupts is the software interrupt. There are two bytes of memory in the DPRAM that are associated with the software interrupt. The first byte is at memory location 0xFF. This is the interrupt flag byte and when it reads 0x01, a request as been asserted via software. The PIC then addresses the second byte at memory location 0xFE. This byte tells the PIC, what operation must be done. It can tell the PIC to stop the timers, reset the timers, and start the timers. Upon the reset command, the timers are reinitialized from DPRAM. So once “timerset.exe” has been executed and the DPRAM altered, the software will tell the PIC to stop, reset, and start the timers. Upon the reset command, the new data from the “config.tmr” file will be used to initialize the timers.

The more complicated of the two interrupts is the external interrupt. This interrupt is provided to the PIC via one of the Lattice PLDs. Upon assertion, the PIC enters the Interrupt Service Routine(ISR). This is where the PRTs are executed and pulses are outputted. The PRT sequences have been stored in PIC memory for easier and faster access. First the timing mode is loaded. Then the period data is read from PIC memory and written to the Period Timer(Timer4). Once this is complete, there are three bytes of data that are outputted to sequence registers via the SPI. The three bytes in order of output are the pulse enable byte, the polarization byte, and the phase byte. Once all three of these bytes have been outputted successfully, the sequence registers are then latched. Once latched, the three bytes are buffered. Once buffered, the polarization and phase bytes are outputted via the DBF25 port. The pulse enable bits are then sent to the Delay Timers(Timer0 and Timer2). These timers then set the desired delay for the six pulses. The signals are then sent to the Pulsewidth Timers(Timer1 and Timer3). These timers set the desired pulsewidths for each of the six signals. The pulse signals are then buffered before being outputted by the DBF25 port.

PCI Timer Board Versions:

There are currently four different version of the timing board. While all hardware has remained the same, the differences are in the PIC Code, EEPROM Code, and "timerlib.cpp". A brief description of each will be provided.

Original Code by Mitch Randall		
Code	Filename	Description
PIC	Pcitimer.asm	The original PIC Code that used wordwise/32 bit memory addressing. This was slower and wasted much memory space of the DPRAM.
EEPROM	Eeprom.hex	The original EEPROM Code that programmed the PCI Bus Interface to operate on a 32 bit data bus.
Timerlib	Timerlib_w.cpp	The original timer definitions that read/wrote to every word/32 bits instead of every byte.
Bytewise Code by Aditi Kapoor		
Code	Filename	Description
PIC	Byteaddr.asm	An updated PIC Code that changed all wordwise addressing to bytewise addressing. Much faster and no more wasted DPRAM memory
EEPROM	Eeprom_byteaddr.hex	An updated EEPROM Code that programmed the PCI Bus Interface to operated on a 16 bit data bus.
Timerlib	Timerlib_byte.cpp	Updated timer definitions that eliminate the read/write to every word/32 bits and read/write to every byte/16 bits.
Opttimer Code by Aditi Kapoor		
Code	Filename	Description
PIC	Opttimer.asm	An update PIC Code that changed the arrangement of sequence data in the DPRAM for easier access. Improved timing but not significantly.
EEPROM	Eeprom_byteaddr.hex	Same file as Bytewise Code
Timerlib	Timerlib_opttim.cpp	Updated timer definitions that change the memory addressing of the sequence data to match the PIC Code.
FastISR Code by Tom Brimeyer		
Code	Filename	Description
PIC	FastISR.asm	An updated PIC Code that increased the clock speed of the SPI in order to decrease time needed to output data on the SPI. This Code also reads the sequence data upon initialization and stores the data in local PIC memory so the PIC can read the information faster than reading from the DPRAM.
EEPROM	Eeprom_byteaddr.hex	Same file as Bytewise Code
Timerlib	Timerlib_FastISR.cpp	Same file as Timerlib_opttim.cpp

PCI Timer Board Testing Procedure:

Currently the PCI Timer Board is using the FastISR Code to operate at the fastest speed possible. Before testing, make sure that the EEPROM and PIC are programmed with the FastISR Code files. Remember also, that the Timerlib file must be updated on the testing computer.

There are two computers and a power supply to be used in this testing procedure. To program the EEPROM and PIC, use the Gateway G6-200 Computer. To open the C++ development or run "timertest.exe" and "timerset.exe", use the NCAR RADAR DRX Computer. To supply power to the PCI Timer Board when programming the PIC Processor, use the QNX Computer which is only a power supply.

There is also a computer only needed if PIC code needs to be assembled. The assembler MPLAB/MPASMWIN.exe can be found on the RTF Processing Developement Computer.

Step 1) Program the EEPROM using the Gateway G6-200.

"Eeprom_byteaddr.hex" must be programmed to the EEPROM device. This is done using the hardware device, ALLPRO-LC (Logical Devices, Inc.) and the software, ALLPRO_LC for Windows. Making sure that the hardware is connected properly, run the software and load the file. Place the EEPROM into the programmer and program.

Step 2) Assemble the PIC Code using the RTF Processing Developement Computer.

(This step may not be required)

If "FastISR.hex" is not available, then "FastISR.asm" must first be compiled into a hex file before it can be programmed. This compilation is done by using the assembler MPASMWIN.exe of MPLAB (Microchip). Run the executable and load the "FastISR.asm" file. In the same directory, the PIC/DPRAM definition file "picdpram.h" must be present for compilation to work successfully.

The MPLAB software can be downloaded from the Microchip website, www.microchip.com.

Step 3) Program the PIC using the Gateway G6-200 or any computer with Epicwin.exe.

Connect the PIC Programmer hardware, Programmer (MicroEngineering Labs, Inc.) to the computer. Run the software "EPICWIN.exe". Load the needed hex file. The PCI Timer Board must be inserted into the PCI slot of the QNX Computer to supply power. Connect the ribbon cable of the hardware to jumper J5 of the timer board. Line up the arrow on the ribbon cable to pin1 of J5. Erase the PIC and then Program.

Step 4) Test the DPRAM using the NCAR RADAR DRX Computer.

The source for “Timertest.exe” can be found at:
C:\Develop\radar\timertest\timertest.cpp

Connect the PCI Timer Board to the NCAR RADAR DRX Computer and then turn the computer on. After bootup is complete, run “Timertest.exe”. There should be a shortcut on the desktop. This should display the device information first. Then it will check each byte of the DPRAM and evaluate as to whether it is a valid byte. It will then write 100 bytes starting at memory location 0x40. Upon completion, it will then read these memory locations back and verify that the written value was read back. If a value is not read back correctly, it will report an error. If all values are correct, it will pass the test. A memory dump of the DPRAM’s contents will also be display on the monitor.

Step 5) Test the Timer Board Pulse Outputs using the NCAR RADAR DRX Computer.

The source for “Timerlib.cpp” can be found at:
C:\Develop\radar\timer\timerlib.cpp

The correct “timerlib.cpp” file must be used to ensure that “Timerset.exe” will work properly. To ensure this, copy the needed Timerlib file into the directory above. Then rename the file “timerlib.cpp”. To ensure that this new file will be used, open the Visual C 6.0 workspace with the filename “radar” in the C:\Develop\radar directory. Open “timerlib.cpp” and rebuild it. Then rebuild “Timerset.exe” and it should use the newer library file.

The source for “config.tmr” can be found at:
C:\Develop\radar\timerset\debug\config.tmr

The “config.tmr” file contains all of the information needed to program the Timer Board using “Timerset.exe”. This is a text file that can be opened and edited so that the pulses will output the user defined period, pulsewidth, and sequence. Multiple sequences or PRTs can be added via this file. After editing is completed, save the file. This is also the file that is used to program the board for full operation.

The source for “Timerset.exe” can be found at:
C:\Develop\radar\timerset\timerset.cpp

The “Timerset.exe” file can be executed to set up the PCI Timer Board to output the user defined data from the “config.tmr” file. Once executed, the program stops the timers, resets the timers, and restarts the timers. The pulses, BPULSE<0:5> can now be observed via the DBF25 output. The pinout for BPULSE<0:5> is as follows.

Pulse Outputs 0-5 can be found from the corresponding DB25 pins.
Bpulse0 → Pin 9

Bpulse1 → Pin 22
Bpulse2 → Pin 10
Bpulse3 → Pin 23
Bpulse4 → Pin 11
Bpulse5 → Pin 24

Each test pulse must be measured in reference to a single test pulse or else the delay will not be evident. Use at least two scope probes at a time, using one to probe BPULSE0 as the reference, and the other to probe any other single BPULSE<1:5>. This way, the delay of BPULSE<1:5> can be observed in reference to BPULSE0. The "config.tmr" file can be edited and "Timerset.exe" reran to observe the desired changes to the BPULSE signals.

It is easier to use a male DB25 connector with soldered leads to observe the signals using the scope probes.

Example of "config.tmr" File Functions:

Follow this example to ensure that the pulse outputs are functioning correctly.

1) All pulses/pulse values are normalized to 8 MHz.
2) All test pulse values can be edited via the "config.tmr" file.

Configuration Variables:
delay → delay time before pulse begins

Time Delay = delay/8MHz i.e. delay=16 → Time Delay = 16/8MHz = 2us

width → width of pulse

Pulse Width = width/8MHz i.e. width=8 → Pulse Width = 8/8MHz = 1us

Sequence Variable:
period → length of period for all pulses

Period = period/8MHz i.e. period=4000 => Period = 4000/8MHz = 500us

Debugging the PCI Timer Board:

This section will discuss what to do the PCI Timer Board is not functioning correctly.

Problem: Upon bootup, the PLL LED is not illuminated. This is the left LED if facing the front of the CPU.
Solution: The VCO may not be calibrated correctly. Using adjust the variable inductor of the VCO until the light illuminates.

Problem: Upon bootup, the PIC LED is not blinking. This is the right LED if facing the front of the CPU.
Solution: The PIC is not operating its code which is probably due to a lack of a clock to the PIC. Test the PIC clock signal. If no signal, then adjust the variable inductor of the VCO.

Problem: When running "Timertest.exe", only every fourth memory location is successfully written to.
Solution: Update the EEPROM to "Eeprom_byteaddr.hex".

Problem: When running "Timerset.exe", the interrupt is not being processed and a PCI Timeout message is received.
Solution: Check to see if PIC is receiving a clock signal. If not, then adjust of the variable inductor of the VCO.