PHENIX Technical Note
The JSEB PCI Interface

Jack Fried
Instrumentation Division
Brookhaven National Laboratory

John S. Haggerty
Physics Department
Brookhaven National Laboratory

September 22, 2003
Version 1.0

Abstract

The PCI interface from the PHENIX Data Collection Modules to the Sub-Event Buffer computers is described. The behavior of the card is described, and the software support in Windows 2000 and Linux is documented.

The JSEB is a PCI board, shown in Figure 1, designed to interface the VME Partition Module (PM) which transmit data from a group of PHENIX Data Collection Modules (DCM's) to x86 computers called Sub-Event Buffer's (SEB's) which are the entry point for data into the Event Builder. The PCI interface chip is the PLX 9080, which implements the 32 bit 33 MHz PCI 2.1 standard PCI bus master with 3.3V or 5V PCI signaling in CMOS [1]. The board has two banks of 1 Mbyte static RAM, which can be written alternately from the PM, providing double-buffering in hardware. The PLX 9080 and the

Figure 1: The JSEB card.

memory is controlled with an Altera FLEX10k FPGA. Since most of the logic on the board resides in this FPGA (which is on-board reprogrammable by means of a JTAG port), the behavior of the board is dependent on the design loaded into the FPGA.

This document is meant to describe the current firmware and software configuration of the JSEB.

1 Communication Protocol with Partition Module

The communication protocol with the Partition Module was designed to be simple and fast. A 50 pair cable with LVDS signals carries 32 bits of data and a small number of control signals at 40 MHz from the Partition Module to the JSEB. Most of the control signals are generated by the Partition Module; the only exception is a HOLD signal generated by the JSEB to request the Partition Module to stop sending data. Since the cable between the JSEB and the PM can be quite long, the PM cannot stop sending data immediately, and so the JSEB must be able to buffer a few words of data even after it has asserted the HOLD signal. A 25m cable has a delay of about 125 ns, or 6 (40 MHz) clock ticks, and the PM may need some number of clock ticks to respond, so the JSEB design includes a FIFO in the FPGA of 12 words.

The basic protocol of the data transfer is shown in Figure 2. A logic analyzer was programmed to observe the protocol at the test points on the JSEB. The top signal labeled DCMCLOCK is the 40 MHz clock from the Partition Module and is too fast to see on the timescale of this transmission. The next trace labeled DCMVALID is the envelope of valid data transmitted by the PM. DCMLASTWD is the signal that the last word of event has been transmitted. DCMVALID and DCMLASTWD may appear to be somewhat redundant, but it was envisioned that the PM might have to turn off the VALID during some transmissions, and then resume the transmission of the same event some number of clock ticks later. In practice, this seems to rarely, if ever, happen. The JSEBHOLD signal is generated by the JSEB to tell the PM to stop the transmission of data. When the number of events per bank is set to zero (so that there is one event per bank), as in this transmission, the HOLD signal is asserted at the end of every event; with more events per bank, the HOLD is asserted when the bank is full, either because the maximum number of events or words has been reached. The design allows for the HOLD to be asserted during the transmission of an event, and the PM is required to respond to it by de-asserting the VALID until the HOLD is released. In practice, this also rarely, if ever, happens, since the JSEB memory can keep pace with data written by the PM.

Note that there is no error detection or correction in the protocol itself. In practice, some error detection has been added by computing a checksum in the DSP's in the DCM on the packets.

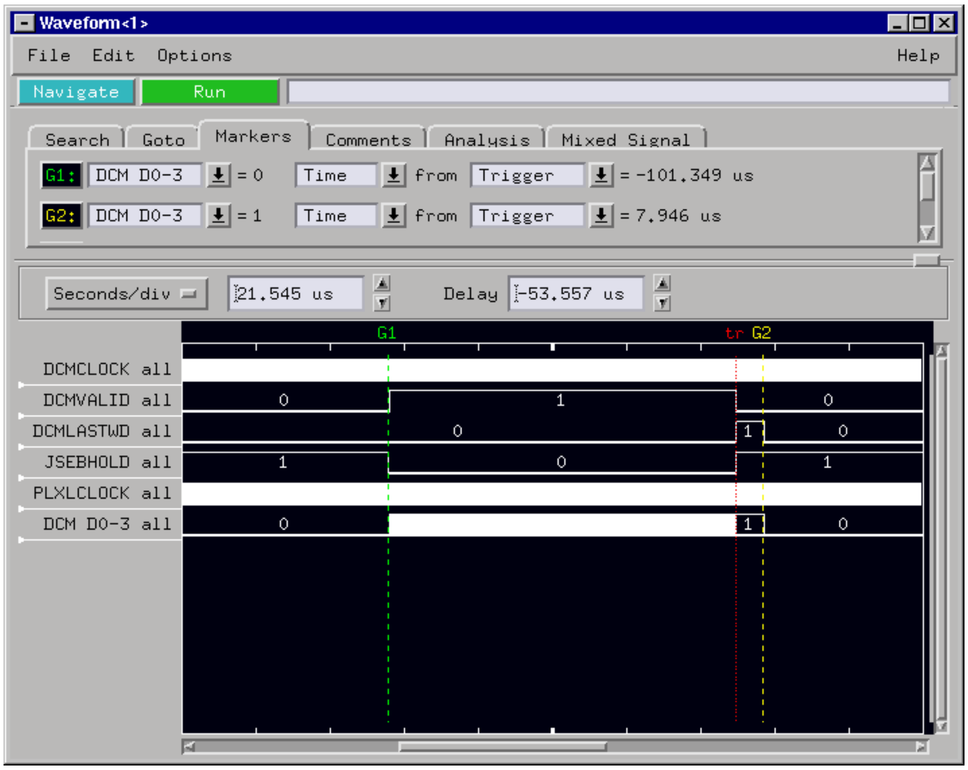

Figure 2: Transmission protocol between the Partition Module and JSEB as observed for a single event. The signals are described in the text.

2 Firmware

The design of the FPGA logic was done with the Altera Max+Plus II [2] software. The resulting pof file is loaded into the Altera FPGA (Altera EPF10K30AFC484-1) by means of an Altera Byte Blaster cable and the serial EEPROM (Altera EPC2LC20) through the P1 header on the board. The designs that have been implemented so far utilize about 80% of the blocks available in the FPGA.

2.1 Version 0x11

Version 0x11 was used for all data taken in Run 3. This version corrected problems with the pointer bank in previous versions, although there was evidence that the next-to-last pointer in bank 1 was sometimes corrupted; the workaround for this problem was to compare the pointer with the value in CSR[6], which was found to be correct. DMA writes from the PCI bus were known not to work correctly, so that memory tests had to be done with single word writes (but DMA reads).

Some of the DCMGROUP's were found to have data errors (detected by the DCM checksum) in the last hundred or so words of a bank, usually in a small fraction of the events, but sometimes in a substantial fraction of the events. A workaround for this was developed, by adding a "padding" packet of all zeroes at the end of an event. When this was done, the rest of the data were found to have errors in a negligible fraction of the events.

2.2 Version 0x13

The main change to version 0x13 was to fix the DMA writes from the PCI bus. However, the compilation from schematics was done with a newer version of the Max+Plus II software.

2.3 Future Enhancements

There are a number of changes that are being discussed for future versions. It is hoped that burst mode DMA reads can be made reliable. It may make sense to modify the mechanism for clearing the registers so that they can all be read after swapping banks; this would allow the interrupt handler to do nothing except swap banks. This would not only be slightly more efficient, it would simplify the interrupt handling.

3 Memory Map

The register map of the basic control and status registers is shown in Table 1. The pointers to the events in the bank when multiple events are buffered is shown in Table 2. Note that the pointer to the first event should always be zero, and that the addresses read are to long words. Also,the pointer bank is read after swapping banks, so the pointers for bank 1 must be read after bank 0 is found ready.

Basic operation of the JSEB consist of the following:

- Reset by setting CSR[0]<2> = 1
- Set the number of events and words per bank by writing to CSR[3] and CSR[4]
- Wait until CSR[0]<1> == 1, indicating bank ready
- Determine which bank is ready by reading CSR[0]<0>
- Read CSR[1] or CSR[2] to determine the number of words and events in the bank
- Swap banks by writing CSR[0]<0> = 1
- Read pointers from CSR[0x400] or CSR[0x600] if multiple events per bank

Register	<Bits>	Read/Write	Offset	Use
CSR[0]		R/W	0x0	Main Control and Status Register
	<0>	R		bank 0/1 visible
	<0>	W		swap banks
	<1>	R		bank ready
	<2>	W		reset
	<3>	R/W		enable interrupt
	<16:23>	R		FPGA version
CSR[1]		R	0x4	Bank 0 status
	<0:19>	R		Words in bank 0
	<20:26>	R		Events in bank 0
	<31>	R		Bank (should be 0)
CSR[2]		R	0x8	Bank 1 status
	<0:19>	R		Words in bank 1
	<20:26>	R		Events in bank 1
	<31>	R		Bank (should be 1)
CSR[3]		R/W	0xc	Event limit
	<0:6>	R/W		Maximum number of events per bank + 1 (default 0x0)
CSR[4]		R/W	0x10	Word count limit
	<0:17>	R/W		Maximum number of words per bank (default 0x20000)
CSR[5]		R/W	0x14	Pointer to last event in bank 1
	<0:17>	R		Added for diagnostic purposes
CSR[6]		R/W	0x18	Pointer to next-to-last event in bank 1
	<0:17>	R/W		Added for diagnostic purposes

Table 1: Control and Status Registers.

- Read the data from address 0x0
- Wait for the next bank to be ready

3.1 Interrupts

The JSEB can generate a PCI interrupt on bank ready. The interrupt is the PCI Local Interrupt, and to enable it, one must set CSR[0]<3> as described above, and set bits 8 and 11 in the INTCSR in the PLX. The source of a PCI interrupt must be cleared in the interrupt handler, or the computer will hang.

Location	Bank	Pointer to event
CSR[400]	0	0
CSR[404]	0	1
.		
.		
.		
CSR[5fc]	0	127
CSR[600]	1	0
CSR[604]	1	1
.		
.		
.		
CSR[7fc]	1	127

Table 2: Pointer bank.

4 Jungo Driver Support

Software support for boards based on the PLX 9080 have been provided with WinDriver by Jungo [3]. This is commercial software which provides a proprietary general purpose driver for PCI devices and access routines for a specific board which can be tailored to the design by creation from a template or by use of a software wizard. The access routines (which are often referred to as the driver) are identical in Windows and Linux. All the access routines are implemented in two files, jseb_lib.c and jseb_lib.h.

4.1 Linux

The Linux library was created from a template provided for the PLX 9080 called p9080. The JSEB driver was created from the p9080 driver template by simply replacing all instances of p9080 with JSEB. Additional functions were added to customize the driver by analogy with the driver created by the WinDriver software wizard (wdwizard). The driver was created under Red Hat 9.0.

The generic Jungo PCI driver (windrvr6) is provided as an installable kernel module which gives kernel support for any PCI board. (It is possible for Version 5.x Jungo drivers to coexist with Version 6.x drivers.) The version described in this document is WinDriver 6.02 for Linux.

In Linux, the driver is a loadable module, which can be loaded like this:

```
/sbin/insmod windrvr6
```

4.2 Windows

The same access routines can be used in Win32. A GNU makefile is constructed for building applications in Windows 2000 so that development can take place in the Cygwin [4] environment.

4.3 PLX Registers

The basic PLX registers [1] can be accessed by standard routines provided in the driver, for example:

```
intcsr_read = JSEB_ReadReg (hPlx, JSEB_INTCSR);
JSEB_WriteReg (hPlx, JSEB_INTCSR, intcsr_write);
```

The names of the registers are defined in jseb_lib.h; INTCSR is the Interrupt Control/Status Register.

4.4 Basic Low Level Functions

4.4.1 JSEB_ReadCSRn

Purpose Reads the values of CSR[n] as described in Table 1.

Prototype DWORD JSEB_ReadCSRn(JSEB_HANDLE hJSEB)

Parameters

Name	Type	Input/Output
hJSEB	HANDLE	Input

Return Value Returns value of CSR[n].

4.4.2 JSEB_WriteCSRn

Purpose Write to CSR[n].

Prototype void JSEB_ReadCSRn(JSEB_HANDLE hJSEB, DWORD data)

Parameters

Name	Type	Input/Output
hJSEB	HANDLE	Input
data	DWORD	Input

Return Value None.

4.4.3 JSEB_Reset

Purpose Full power up reset. Sets maximum number of events and words to defaults.

Prototype void JSEB_Reset(JSEB_HANDLE hJSEB)

Parameters

Name	Type	Input/Output
hJSEB	HANDLE	Input

Return Value None.

4.4.4 JSEB_SwapBanks

Purpose Swaps the active bank.

Prototype void JSEB_SwapBanks(JSEB_HANDLE hJSEB)

Parameters

Name	Type	Input/Output
hJSEB	HANDLE	Input

Return Value None.

4.4.5 JSEB_BufferReady

Purpose Checks the bank ready bit in CSR[0].

Prototype void JSEB_BufferReady(JSEB_HANDLE hJSEB)

Parameters

Name	Type	Input/Output
hJSEB	HANDLE	Input

Return Value TRUE indicates that a bank is ready to read.

4.4.6 JSEB_Version

Purpose Returns the version of the firmware.

Prototype DWORD JSEB_Version(JSEB_HANDLE hJSEB)

Parameters

Name	Type	Input/Output
hJSEB	HANDLE	Input

Return Value Version number of the firmware in the JSEB.

4.5 Functions Used in Normal Operation

4.5.1 JSEB_OpenOne

Purpose Opens a JSEB.

Prototype BOOL JSEB_OpenOne(JSEB_HANDLE *phJSEB)

Parameters

Name	Type	Input/Output
phJSEB	*HANDLE	Input

Return Value Returns TRUE on successful opening of a JSEB. The pointer to the JSEB handle is used in subsequently called functions.

4.5.2 JSEB_Initialize

Purpose Resets the JSEB and sets the maximum number of events and words per bank.

Prototype void JSEB_Initialize(JSEB_HANDLE hJSEB, DWORD event_limit, DWORD word_limit)

Parameters

Name	Type	Input/Output
hJSEB	HANDLE	Input
event_limit	DWORD	Input
word_limit	DWORD	Input

Return Value None.

4.5.3 JSEB_GetStatus

Purpose Reads essential registers and then swaps banks, making the event counters and pointers available in a returned structure.

Prototype BOOL JSEB_GetStatus(JSEB_HANDLE hJSEB, JSEB_Status *pStatus)

Parameters

Name	Type	Input/Output
hJSEB	HANDLE	Input
pStatus	*JSEB_Status	Output
.bank		
.events		
.words		
.pointer[128]		

Return Value Returns TRUE on successful read of the status.

4.5.4 JSEB_ReadData

Purpose Reads data from the active bank after swapping.

Prototype BOOL JSEB_ReadData(JSEB_HANDLE hJSEB, DWORD word_count, PVOID buffer)

Parameters

Name	Type	Input/Output
hJSEB	HANDLE	Input
word_count	DWORD	Input
buffer	PVOID	Output

Return Value Returns TRUE on successful read of the status.

4.6 PLX Registers

In addition to registers defined in the Altera firmware, the PLX registers [1] can be accessed.

The PLX 9080 is capable of burst mode DMA transfers.

5 Test Programs and Examples

A number of test programs have been written to exercise the JSEB hardware and the software.

5.1 memtest

memtest is a basic read/write memory test with a variety of test patterns.

```
memtest [-sv]
```

Where the options are:

-s Single word writes (used to test whether reading or writing data has errors, since single word writes have been found to be very reliable).

-v Verbose printout of errors (can be very lengthy).

5.2 speed

speed is used to measure the read speed of data over the PCI bus.

```
speed bytecount loopcount
```

The two arguments are:

bytecount Number of bytes to read in a single transfer

loopcount Number of times to repeat the transfer (to get average timing)

This program can be called from a Perl script (speedmap.pl) which measures the read speed as a function of the length of the transfer (data shown in Figure 3 was taken this way).

5.3 readdata

readdata is a program for reading data from the PM and printing it to stdout, generally without regard to the data format or organization. The program is built to buffer 100 events per bank by default.

```
readdata [i]
```

The optional argument i is an integer such that:

0 Prints event and word counts and all pointers and data to stdout (the default)

1 Prints event and word counts and all pointers to stdout

2 Prints event and word counts to stdout

>**2** Prints nothing (but reads all the data)

5.4 jsebdaq

jsebdaq is a program for reading data from the PM and recording it to the PHENIX Raw Data Format (PRDF), by Martin Purschke. The program is built to buffer 100 events per bank by default.

```
jsebdaq [options] filename
```

The options are:

-b <size in MB> Buffersize for output prdf (not more that 12 MB)

-w <milliseconds> Wait so many ms between events

-n <events> So many events (default 1000)

-v Verbose

-h Help text

5.5 Data Integrity Checking

A number of programs are useful for checking the integrity of the data; indeed any program sensitive to errors in the data can be useful. Two programs [5] written by Mickey Chiu, have proven especially useful:

badjseb filename Checks the "padding" bank

dcmchecksum filename Compares the checksum calculated by the DCM's with one calculated from the raw data

5.6 Using the DCM's to Produce Fake Data

It is very useful to use a DCMGROUP (a group of adjacent DCM's and their Partition Module) to produce fake data.

Here are the steps necessary to do that.

- Create the DCM configuration files from the pcf files in $RC_HW_CONF like this:

  ```
  process_pcf_file.sh -DPAR_JSEB -DFAKE dc.w.pcf
  ```

  or

  ```
  process_pcf_file.sh -DPAR_JSEB -DFAKE5 dc.w.pcf
  ```

  The first produces data in exactly the format of the granule, but the data payload are all 0x5555 and 0xaaaa, the second produces a data payload which consist of a counter.

- Use the DCM program to initialize the DCM's and take data:

  ```
  dcm dc.w
  301
  303
  305
  0
  none
  ```

  This will start the DCM running and generating data. You can leave it running, and just start and stop a JSEB test program, such as the one shown in the next section, to get data into the JSEB.

5.7 Simple Example of Reading Data

This is a very simple example of a program that reads data from the JSEB. This program illustrates the use of basic functions in the library.

```
#include "lib/jseb_lib.h"
#include <stdio.h>

// 1 Mbyte - 8 kbyte in words
#define BUFFER_LENGTH 260096

int main(int argc, char *argv[])
{
    JSEB_HANDLE hPlx;
```

```
    JSEB_Status JStatus;

    DWORD max_events = 100;
    DWORD max_words = 200000;

    DWORD word_count = BUFFER_LENGTH;
    DWORD buffer[BUFFER_LENGTH];
    int wait;

    if ( !JSEB_OpenOne( &hPlx ) )
      {
        printf("error opening jseb\n");
        exit(0);
      }

    JSEB_Initialize( hPlx, max_events, max_words );

    for ( ;; )
      {

        for ( wait=0;; wait++ )
          if ( JSEB_BufferReady( hPlx ) ) break;

        JSEB_GetStatus( hPlx, &JStatus );
        JSEB_ReadData( hPlx, JStatus.words, buffer );

      }

    if (hPlx) JSEB_Close(hPlx);

    return 0;
}
```

6 Speed

The speed of reading banks of various lengths is shown in Figure 3. The speed is measured by repeatedly carrying out a DMA read of a block of data and measuring the elapsed time with gettimeofday (which returns microsecond precision, although since the total time of the repeated transfers is several seconds, the precision does not contribute significantly to the uncertainty). The speed was measured with burst mode enabled and without, although there is evidence for data corruption in some cases in transfers with burst mode enabled.

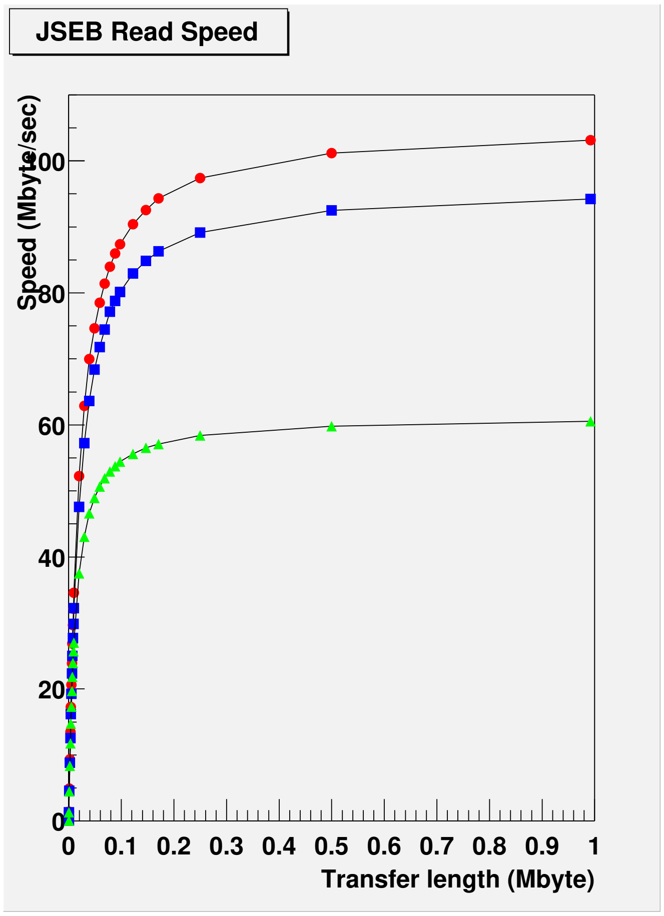

Figure 3: The speed for reading banks as a function of the bank length. The circles are with continuous burst mode DMA, the squares are for four word burst mode DMA, and the triangles are for normal DMA.

7 More Information

There is more information about the JSEB on a web page [6]. Code and additional documentation can be found in two places:

/home/phoncs/haggerty/jseb Early development, mainly for Windows NT, and test records

/home/phnxrc/haggerty/jseb Development described in this document, including Linux driver

8 Acknowledgments

Many people have contributed to the development of the JSEB since the inception of the project. Steve Lin worked on an early version of a board that demonstrated the communication with the Partition Modules as a graduate student at Stony Brook and later as a staff engineer. The communication protocol with the Partition Module was developed and tested in collaboration with Cheng-Yi Chi and Bill Sippach. Peter Steinberg and Brian Cole developed driver software based on the PLX Software Development Kit which was used in PHENIX runs in FY01 and FY02, and a class library for handling JSEB operations. Mickey Chiu observed data corruption at the end of some banks, created the DCM padding banks as a workaround to the problem, and wrote some software to check the integrity of the data. Martin Purschke adapted his event libraries in order to be able to log data in PHENIX Raw Data Format from the SEB. The new Smart Partition Module was tested in collaboration with Jamie Nagle. Several other groups have adapted the JSEB to their uses, notably Sam Hoblitt in the LEGS group, who has written an Alpha VMS driver for it, and has shared his experience with us. Michael Bjorndal helped make some of the measurements of rates.

References

[1] http://www.plxtech.com/products/default.htm "PCI 9080 Data Book Version 1.06."

[2] http://www.altera.com/products/software/pld/products/maxplus2/mp2-index.html

[3] http://www.jungo.com

[4] http://www.cygwin.com

[5] Mickey Chiu developed these programs and I made minor modifications and rebuilt them in /home/phnxrc/haggerty/chiu/utilities.

[6] http://www.phenix.bnl.gov/phenix/project_info/electronics/haggerty/jseb