

Warp Nine Engineering

F/Port ISA Card

Register Description and Programming Examples

Revised 31 May 1995

Summary

This document describes the register set for the FarPoint Communications' F/Port card (P/N 510-0002). The logic for this card is implemented using the Xilinx 2064 Field Programmable Gate Array. The FPGA allows us to have multiple design 'personalities' with one card. The current implemented designs are for two types of parallel port protocols, EPP (Enhanced Parallel Port) and Fast Centronics (FPORT, HPLJ4, METRUM_A, TI600_RA, AP9300A). For applications or protocols other than these, FarPoint can generate a personality that can meet your custom requirements. For example, a PRINTER personality is available that makes the parallel port behave as a Centronics style input port for a laser printer.

The EPP personality implements the IEEE 1284 EPP interface protocol. This is a bi-directional, fully interlocked signaling method capable of 1.25MBytes per second.

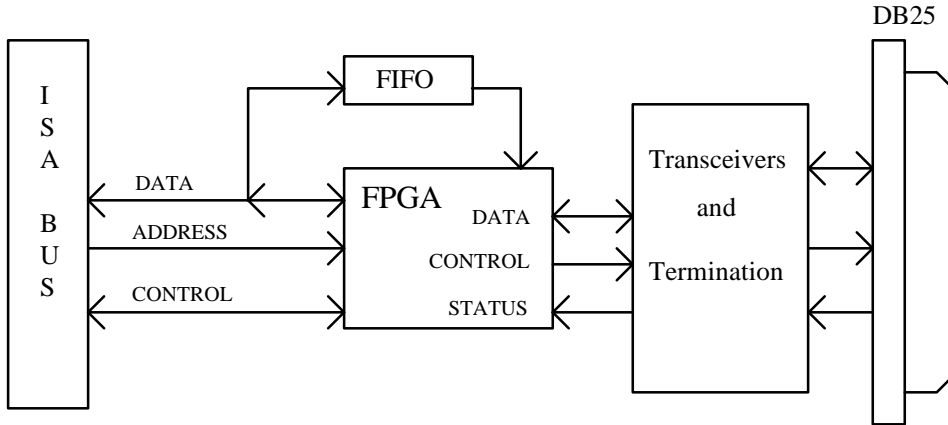
The FPORT, and other Fast Centronics personalities, use the onboard FIFO and state machine to implement a block oriented high speed Centronics signaling interface. This protocol is capable of 800K Characters per second. The different personalities are optimized to perform the fastest Centronics handshaking with the targeted printer. Using the correct mode will guarantee that the F/Port card will drive the printer interface as fast as the *printer* can take the data.

For this document, FPORT mode will refer to all the Fast Centronics modes. These modes differ only in the printer handshake timing, not in any register definitions.

In either EPP or FPORT mode, the standard parallel port (SPP) registers are still available.

Block Diagram

The F/Port card is essentially a FIFO and Xilinx FPGA with an ISA interface and a DB25 with transceivers and termination. The following diagram illustrates the architecture:



I/O Map and IRQ

The F/Port card uses 8 contiguous I/O registers beginning at one of four starting locations:

- 278 LPT2
- 280 General I/O
- 290 General I/O
- 378 LPT1

The base address is jumper selectable.

The interrupt, IRQ, level is jumper selectable. The options are:

- 5 7 10 11 12 15

The card may be inserted in either an 8 bit or 16 bit I/O slot. If used in an 8 bit slot then only IRQ levels 5 or 7 may be selected.

PORT Definitions

The following table defines the ports for the EPP and FPORT modes:

Base Offset	R/W	SPP Mode	FPORT Modes	EPP Mode
0	R/W	WPA data port	WPA data port	WPA data port
1	R	RPB status port	RPB status port	RPB status port
2	R/W	WPC control port	WPC control port	WPC control port
3	R/W	not defined	not used	Address port
4	R/W	not defined	FIFO (block) data port ⁽¹⁾	Data port 0 ⁽³⁾
5	N/A	not defined	not used	Data port 1
6	W	not defined	Config register	Config register
7 ⁽²⁾	W	not defined	FPGA download	FPGA download

(1) Reading from this port returns the data written into the FIFO, not the external data. This can be used for diagnostic testing.
 (2) This port is used to reset and configure the FPGA. This port should never be written to.

(3) EPP mode can use 16 bit I/O.

Register Definitions

The following tables define the bits used by the various registers. All signals are asserted true unless defined as "nXXX" which implies that signal 'XXX' is asserted low when its corresponding bit is written as a logic 1. On registers that are read only, 'nXXX' implies that the signal condition is true when read low.

Port 0-WPA Data Port

Bit	SPP	FPORT Modes	EPP
<0:7>	I/O Data	I/O Data	I/O Data

Port 1-RPB Status Port

Bit	SPP	FPORT Modes	EPP
0		nFIFO_EMPTY	TIMEOUT ⁽¹⁾
1		nFIFO_FULL	EPP_TEST ⁽²⁾
2		LOGIC_0	RESET_TIMEOUT ⁽³⁾
3	nERROR	nERROR	nERROR
4	SELECT	SELECT	SELECT
5	PE	PE	PE
6	nACK	nACK	nACK
7	nBUSY	nBUSY	nBUSY

- (1) During an EPP cycle, if the peripheral does not respond within 5 μ S then the timeout bit is set and the ISA I/O cycle is terminated. This will allow a block move to proceed uninterrupted without hanging the PC. This status should be checked at the end of a block move. This condition is reset by the RESET_TIMEOUT bit in the configuration register.
- (2) EPP_TEST is the read back of EPP_TEST in the WPC register. This is used to test the presence of the EPP personality.
- (3) RESET_TIMEOUT is the readback of the RESET_TIMEOUT bit in the configuration register.

Port 2-WPC Control Port

Bit	SPP	FPORT Modes	EPP
0	nSTROBE	nSTROBE	nSTROBE
1	nAUTOFEED	nAUTOFEED	nAUTOFEED
2	INIT	INIT	INIT
3	nSELECT_IN	nSELECT_IN	nSELECT_IN
4	IRQ_ENABLE	IRQ_ENABLE	IRQ_ENABLE
5		DIRECTION ⁽¹⁾	EPP_PROTECT ⁽²⁾
6			
7		HSP_MODE ⁽²⁾	EPP_TEST ⁽³⁾

- (1) When set to 0 the data port is write only. Readback will return the last data written to the port. When set to 1, the data port is read only. Readback will return the data on the parallel port interface.
- (2) This bit is READ only from this register. To set it, see Port 6 "Configuration Register".
- (3) This bit is WRITE only to this register. The readback is on bit 1 of the RPB Status register, Port 1.

Port 3- Address Port

This port is used only in EPP mode. Data written to this port will execute an EPP Address Write cycle. A read from this port executes an EPP Address Read cycle.

Port 4- Data Port

EPP Mode- Data written to this port will execute an EPP Data Write cycle. A read from this port will execute an EPP Data Read Cycle.

FPORT Mode-When the FIFO_EMPTY status is read from the RPB port, a block of 512 data bytes may be written to this port. Whenever the FIFO is not empty, a centronics cycle with auto-strobing is initiated. These cycles will continue until the FIFO is emptied. If the HSP_MODE bit is set, then an interrupt will be generated when the FIFO goes empty.

Port 5- Data Port

EPP Mode- Used for 2nd byte of word I/O transfer. Transparent to the software

Port 6- Configuration Register

Bit	FPORT Modes	EPP
0	HSP_MODE	EPP_PROTECT
1	FIFO_RESET	RESET_TIMEOUT
<3:7>	not used	not used

HSP_MODE- When set, disables the WPC nSTROBE function, enables interrupt on FIFO not empty, and enables the auto-strobing function. Can be read back through the WPC (bit 7) to detect the presence of the F/Port configuration.

FIFO_RESET- When pulsed to a 1 and then 0 will reset the FIFO to the cleared state.

EPP_PROTECT- When set to 1, the WPC functions nSTROBE, nAUTOFEED and nSELECT_IN are disabled. This will prevent an ill behaving driver from setting the control lines in a non-EPP condition.

RESET_TIMEOUT- Should be pulsed to a 1 then 0 to reset the timeout error status on the RPB port.

Port 7- Xilinx Configuration Port

Used to reset and configure the Xilinx FPGA. This port should never be accessed.

WRITING SOFTWARE DRIVERS FOR F/Port CARD

There are two considerations when writing drivers for the F/Port card: Personality download and the Driver programming interface. As discussed earlier, this card is based upon a field programmable gate array. This means that when the PC is turned on, the card is unconfigured, and the system BIOS will not 'see' the parallel port. A driver is needed to download a personality to the card and then to setup whatever is appropriate so that the system will see this as an installed card. For instance, in a machine booting to DOS the driver FASTPORT.SYS or the FPCONGIG program will download a personality to the card and then set the address of the card into the BIOS LPT data area (40:08-0D) and update the BIOS equipment field (bits 14-15 of 40:0A). This will make the F/Port appear as a standard parallel port to any other port driver. The FLPT.SYS driver then is used to enable high speed printing as a standard LPT device. For operating systems other than those that are DOS based, Windows NT, OS/2, for example, you must provide these types of services. FarPoint does not support these operating systems at this time, but will be happy to assist you in the development of these drivers.

In 'Fast Port' mode the F/Port will drive the printer as fast as it can receive data; up to 800,000 bytes per second. This is accomplished through the use of an on-board FIFO buffer and a hardware state machine. The F/Port software driver outputs data to the FIFO buffer, not the printer. The hardware state machine fetches data from the FIFO and performs the printer handshaking - pulsing data strobe - transferring the FIFO data to the printer. The state machine uses the BUSY line to flow control data to the printer; it will output FIFO data to the printer whenever the printer is *NOT BUSY*. Also, when in Fast mode (when the card's interrupts are enabled) the F/Port will generate an interrupt when the FIFO becomes empty, not when the printer pulses the ACK line.

To write a software driver for the F/Port card is relatively simple. You have only to enable the Fast mode and output your printer data via the FIFO DATA port (I/O Base + 4). To maximize performance you must send the data to the FIFO via the BLOCK I/O instruction REP OUTSB (NOTE: This instruction is good for a 80186 or better CPU). The FIFO buffer is only 512 bytes. In order not to overflow the buffer you should not output to the FIFO unless it is empty. You can determine that the FIFO is empty by receiving the FIFO empty interrupt, or by polling the FIFO empty status bit - bit 0 of the STATUS port (I/O Base + 1).

The following code fragment below illustrates the setting of the Fast mode and outputting to the FIFO buffer. They are written in assembly language but are designed to be called from "C" programs.

MODEL Dependencies

If operating in SMALL model then set:

```
parm1      equ    4
parm2      equ    6
```

If operating in MEDIUM or LARGE model then set:

```
parm1      equ    6
parm2      equ    8
```

It is assumed that the F/Port card's I/O base address is stored in a "C" global variable:

```
extrn  _IO_BASE:word
```

```
*****
;
;      Set F/Port mode                                     *
;
;
;      void setMode(int mode)                             *
;          mode = 1      - to set FAST mode               *
;          mode = 0      - to set standard parallel port mode *
;
;*****
```

```
public  _setMode
_setMode:
    push  bp
    mov   bp, sp
    mov   ax, parm1[bp]          ; get mode parameter
    mov   dx, _IO_BASE
    add   dx, 6                  ; point to F/Port CONFIG port
    out   dx, al
    pop   bp
    ret
```

```
*****
;
;      Check if FIFO empty                               *
;
;
;      "C" call:    int isFifoEmpty()                    *
;                  returns True (1) if empty; else False (0) *
;*****
```

```

public _isFifoEmpty
_isFifoEmpty:
    mov     dx, _IO_BASE
    inc     dx                ; point to STATUS port
    in      al, dx           ; read F/Port status
    test    al, 01h         ; test bit 0
    jnz     isFifo_NotEmpty ; jump if not empty (bit 0 = 1)
;     FIFO is empty
    mov     ax, 1
    ret
;     FIFO is not empty
isFifo_NotEmpty:
    xor     ax, ax
    ret

```

```

;
;     Output block (up to 512 bytes) to FIFO buffer *
;     This must be done in assembly language because you cant do block I/O in "C" *
;     It is assumed that the FAST mode is currently enabled and the FIFO is empty *
; *
;     "C" call     void     outputToFifo(char far *printData, int cnt) *
*****

```

```

public _outputToFifo
_outputToFifo:
    push    bp
    mov     bp, sp          ; setup "C" call frame
    push    si
    push    ds
    mov     dx, _IO_Base    ; get I/O base address of F/Port card
    add     dx, 4           ; point to FIFO data port
    mov     cx, parm2[bp]   ; get block byte count
    lds     si, DWORD PTR parm1[bp] ; get block pointer (DS:SI)
    cld
    rep     outsb           ; output 512 bytes to FIFO buffer
    pop     ds
    pop     si
    pop     bp
    ret

```

The following "C" code fragments illustrate how a driver would call the assembly language routines.

```
//      Function prototypes for assembly language routines
int      isFifoEmpty(void);
void     setMode(int);
int      outputToFifo(char far *, int);

int IO_Base;                // global variable that stores F/Port's I/O base address

//      arbitrary driver initialization routine
int driverInit()
{
    :
    :                        // driver initialization code
    :
    IO_Base = FPORT_BASE_ADDRESS;
    setMode(1);              // set Fast mode
    :
}

//      arbitrary driver routine used to output LARGE block to printer
void     outputBlock(char *printBlock, int size)
{
    while(size)
    {
        // wait until FIFO is empty
        if(isFifoEmpty())
        {
            if(size > 512)
            {
                outputToFifo(printBlock, 512); // output next 512 block
                size = size - 512;
                printBlock = printBlock - 512; // point to next 512 block
            }
            else                // output last block
            {
                outputToFifo(printBlock, size)
                size = 0;
            }
        }
    }
}
```