

DSP to FIFO Interface

Can we program the FIFO $\overline{\text{PAE}}$ and $\overline{\text{PAF}}$ flag offsets easily? Why would we want this?

In the new PU architecture, the output data and its clock (from EMIFB) does not go through the Output FPGA any more. Since programming the flag offsets requires controlling the FIFO $\overline{\text{WCLK}}$ signal, this means that only the DSP could program the FIFO flag offsets.

For this, the DSP must also control the FIFO $\overline{\text{LD}}$ signal. This can be done using one of the EMIFB signals (for example $\overline{\text{BCEX}}$ or one of the $\text{BEA}[20:1]$ address lines) or by using a general-purpose DSP output (GPIO).

Using one of the EMIFB signals is the easiest way to program the flag offsets, by just one 32-bit write to a particular EMIFB address¹.

However, to select parallel flag programming, we must also ensure that $\overline{\text{LD}}$ is low during the FIFO master reset (with the proper timing). We also must be careful to assert $\overline{\text{LD}}$ only when no DMA is writing event data to the FIFO. In other words, **controlling $\overline{\text{LD}}$ must be coordinated with $\overline{\text{MRS}}$ and $\overline{\text{WEN}}$.**

The solution proposed is to have $\overline{\text{MRS}}$ controlled by $\overline{\text{BARE}}$. To perform a FIFO master reset and select the parallel programming, the DSP will have to perform a **read** from the appropriate EMIFB location. This memory space must be configured as asynchronous memory, so that the FIFO master reset timing is satisfied².

From here, I see two possibilities which I explain below.

Option 1

This is the preferred option, and is outlined in Figure 1.

- The DSP $\overline{\text{BSWE}}$ controls the FIFO $\overline{\text{WEN}}$
- The DSP $\overline{\text{BCE3}}$ controls the FIFO $\overline{\text{LD}}$
- The DSP $\overline{\text{BARE}}$ controls the FIFO $\overline{\text{MRS}}$.

To reset the FIFO and select the parallel flag programming, the DSP will perform a read from any $\overline{\text{BCE3}}$ memory location. This memory space must be configured as asynchronous memory. Note that the FIFO $\overline{\text{WEN}}$ will remain inactive (high) during the reset, as specified in the FIFO data sheet.

Programming of the FIFO flags is done using writes to $\overline{\text{BCE3}}$ which must now be reconfigured as synchronous memory.

1. The FIFO datasheet issued in May 2002 contains contradictory information about how many write cycles are needed to program the flags. I have contacted IDT technical support.
2. Refer to Figure 5 on page 24 of the IDT72V223 data sheet of May 2002 describing the master reset timing, and to Figure 17 on page 79 of the DSP data sheet SPRS146D describing the asynchronous memory timing.

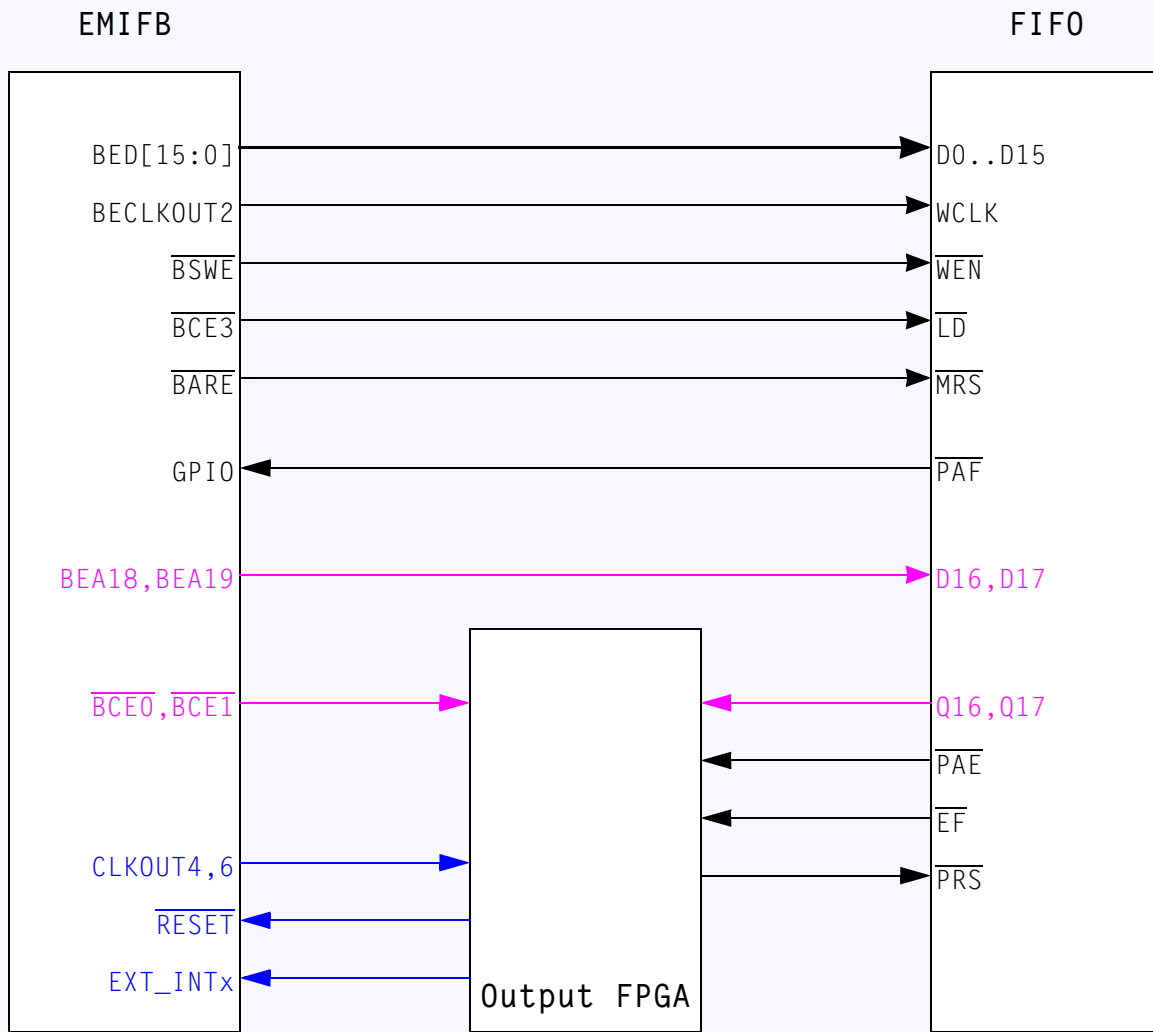


Figure 1: DSP FIFO interface. Connections shown in pink are discussed later in the note.

Normal FIFO synchronous writes can be performed in any non-BCE3 EMIFB memory space configured as programmable synchronous memory.

Option 2

This option does not have a corresponding figure.

- The DSP $\overline{\text{BCE0}}$ controls the FIFO $\overline{\text{WEN}}$
- The DSP $\overline{\text{BEA20}}$ controls the FIFO $\overline{\text{LD}}$
- The DSP $\overline{\text{BARE}}$ controls the FIFO $\overline{\text{MRS}}$.

To reset the FIFO and select the parallel flag programming, the DSP will perform a read from any non-BCE0 memory location provided that BEA20 stays low. For example, we may chose the DSP address 0x64000000 which is in BCE1. This memory space must be

configured as asynchronous memory. Note that the FIFO \overline{WEN} will remain inactive (high) during the reset, as specified in the FIFO data sheet.

Programming of the FIFO flags is done using writes to any BCE0 address provided that BEA20 stays low. For example, we may chose the DSP address 0x6000000. The BCE0 memory space must be configured as synchronous memory.

Normal FIFO synchronous writes can be performed in any BCE0 memory space provided that BEA20 stays high. We have chosen the highest address bit, BEA20, so that the entire FIFO could be written in one go without changing this bit. So output event data shall be written at the DSP addresses in the range 0x62000000 through 0x63ffffff.

Examples

The almost-full flag programming can be practical to actually **query** the FIFO for the availability of free space required to write a complete event. This can be important, since the 6414 EDMA, unlike the 6203 DMA, does not allow level triggering, only edge triggering. The flag offsets could be written before each event, if needed. So the FIFO could be queried only once at the beginning of each event transfer, and the (variable-size) events need not be split in blocks matching the predefined flag offsets.

The almost-empty flag programming could be practical to implement the EVTRDY protocol. Indeed, if the Output FPGA is monitoring PAE, then this flag could be retransmitted as EVTRDY. Before writing out each event, the DSP would have to program the almost-empty offset to the size of that event (minus a few words). *This requires further study of all possible cases!* The motherboard may have to ignore transitions on EVTRDY during the event transfer.

This may be important, since the 6414 EDMA, unlike the 6203 DMA, does not offer a DMAC (DMA complete) signal. The DMAC was a convenient way of implementing EVTRDY without software handling by the DSP. The other possibility is that the DSP must check the DMA completion (or be interrupted) and then issue a software DMA complete signal to the Output FPGA. This could result in a loss of output bandwidth.

Usage of D16 and D17

Wiring of these signals as shown in Figure 1 (pink lines) could allow the Output FPGA to monitor what's going on with the event transfers. Also connecting the DSP BCE0 and BCE1 (possibly also BCE2) to the Output FPGA could be used to notify the Output FPGA when an event transfer takes place.

A summary of the DSP to FIFO connections is shown in Table 1.

Other issues

In order to make the parallel flag programming convenient and easy to use, one should connect the DSP EMIFB data lines to the FIFO **respecting the signal names**. That is, these lines **may not** be swapped for easier PCB routing!

Table 1: Summary of DSP FIFO interface.

From	Signal Name	To	Signal name
DSP EMIFB	BECLKOUT2 (back-term)	Output FIFO	WCLK
	BED0 thru BED15	Output FIFO	D0 thru D15
	$\overline{\text{BSWE}}$	Output FIFO	$\overline{\text{WEN}}$
	BEA19, BEA18	Output FIFO	D16 and D17
	$\overline{\text{BCE0}}$, $\overline{\text{BCE1}}$	Output FPGA	I/O
Output FIFO	Q16 and Q17	Output FPGA	I/O
	$\overline{\text{ASYW}}$, $\overline{\text{ASYR}}$	Tied high	
	$\overline{\text{BE}}$ (big endian bar)	Tied low	
	IW, OW (bus widths)	Tied low	
	FWFT	Tied low	
	$\overline{\text{RT}}$, RM^a	Tied high	
	FSEL0, FSEL1	As needed for flag settings	
	$\overline{\text{LD}}$	DSP	$\overline{\text{BCE3}}$
	$\overline{\text{SEN}}$	Tied high	
	IP	Tied low	
	PFM (prog. flag mode)	Tied low?	
	$\overline{\text{MRS}}$	DSP	$\overline{\text{BARE}}$
	$\overline{\text{PRS}}$	Output FPGA	I/O
	$\overline{\text{PAF}}$	DSP	GPIO or EXT_INT
	$\overline{\text{PAE}}$, $\overline{\text{EF}}$, ($\overline{\text{FF}}$, $\overline{\text{HF}}$)	Output FPGA	I/O
	RCLK	System	
	$\overline{\text{REN}}$	System	
	Q0 thru Q15	System	
	$\overline{\text{OE}}$	Output FPGA?	I/O
	JTAG	System	

a. Timing of retransmit mode is not relevant since we do not use retransmit mode.