Flag Basics:

Flags are used for providing flow control on the rapid data transfer taking place on the GPIF block of FX3. The flags indicate the vacancy / data availability in FX3's DMA Buffers to the external processor, by the virtue of which the external processor can read / write data such a way that under-run and back-flow issues are avoided.

There are two types of Flags in FX3: DMA_READY flags and DMA_WATERMARK Flags. A ready flag indicates the full or empty status of buffers inside FX3. Monitoring the Ready flag alone is not enough for reading / writing data. There is a finite delay present between the point at which the buffers inside FX3 become full / empty and actual assertion of flags. The exact delay is documented in Table 4 of AN65974 (Rev *I).

So before the assertion could take place, the external processor could misunderstand the status of FX3's internal DMA buffers and perform read / write option which will result in loss of data. In order to avoid this scenario, the partial flags are used. This is essential for the cases in which the external processor sends random amount of data every time, and does not keep a track of data it sends. (If the External processor sends fixed amount of data every time, the counters in the GPIF Interface can be used to keep a track of number of words read / written).

The watermark flags indicate the status of the buffers well in advance, so that the external processor is given an alert that it should continue doing the data transfers only for specific amount of clock cycles. Thus watermarks flags are used to end a transfer successfully.

Uncertainty on the Behavior of Watermark Flags:

But when beginning a transfer, the customers are requested to monitor only the DMA_READY Flags and not the watermark flags. This is recommended in the AN65974 which makes mandatory to have the DMA_READY flags in the design**. Our existing documentations do not explain the reasons clearly on why the watermark flags should not be monitored for beginning a transfer. Many times, the customers had come up with the same question, wondering if they can eliminate the use of DMA_READY flags, there by getting enough GPIOs for other purpose.** So the behavior of Watermark flags is tested at the time of beginning of transfer, the defects have been documented and Work around is suggested in this memo for using the watermark flags alone in the design.

USB to GPIF Transfer:

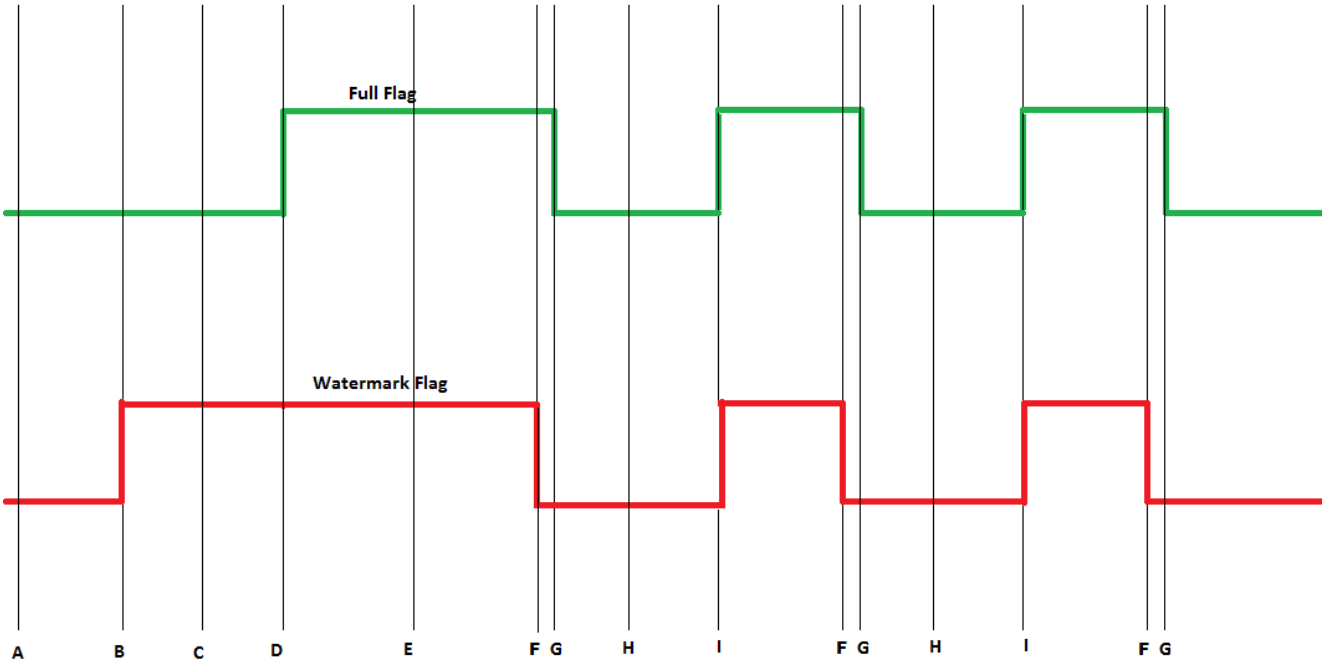
In this transfer, the flags act as Empty Flags. If the flag is asserted, it means all the buffers inside FX3 are empty. If it is de-asserted, then it means at least one buffer is

filled with data. The external processor monitors if the flag is de-asserted and reads out the data from the buffers, and does not read when the flag is asserted.

Issue with the partial flag:

In USB to GPIF Transfer, the partial flag is observed to reflect the status of the buffers only after one full DMA Buffer is read out. So as soon as the state machine is started the partial flag cannot be monitored to know the empty status of the buffers.

Assume that the DMA Channel is created, and as soon as the state machine is started, there is no data inside the buffers at this point of time. So ideally both the DMA_READ Y and DMA_WATERMARK flags should be asserted. But it is observed that the DMA_WATERMARK flag is de-asserted and it is found to indicate the buffer status only after a full buffer has been read out by the GPIF.



Polarity : Low
Initial Value : Low

Events in the above figure

A: The DMA Channel from USB to GPIF is created in the firmware

B: The GPIF State Machine is started. The DMA_READY flag is de-asserted indicating correctly that there is no data inside the buffers.

But the DMA_WATERMARK Flag is de-asserted. This causes the confusion while using the partial flag alone for beginning a data transfer.

C: The PC starts sending data through the USB Interface.

D: Assume that a PC has sent one DMA Buffer of data. The DMA_READY Flag is de-asserted indicating that there are no more empty buffers.

E: Assume that the External processor which had sampled the state of the flag, starts reading the data from FX3, at this point

F: As the external processor reads data from FX3, when the watermark level is reached, the DMA_Watermark flag is asserted first, so that the external processor is alerted to terminate the Data Transfer well in advance. After this the DMA_WATERMARK flag acts in the expected way.

G: At this point, the Data buffer has been completely read by the External Processor and DMA_READY flags is de-asserted indicating that there are no more filled buffers. The duration between E and G includes the time delay (2 clock cycles) between the point at which the buffer actually becomes empty and the point at which the flag is asserted.

H: Similar to C, except the DMA_Watermark flag also shows the correct status

Because of the issue seen at ‘B’, we are unable to start a Data transfer (reading from FX3) by monitoring the watermark flag alone. The event D will be missed if WM flag is monitored.

So the designer will be on safer side, if the DMA_READY Flag is monitored for beginning the data transfer and DMA_WATERMARK flag is monitored for ending the data transfer.

The Workaround:

If the customers run short of extra GPIOs and are not able to use two separate pins for flags, then the following ugly workaround can be done for using the Watermark flag alone.

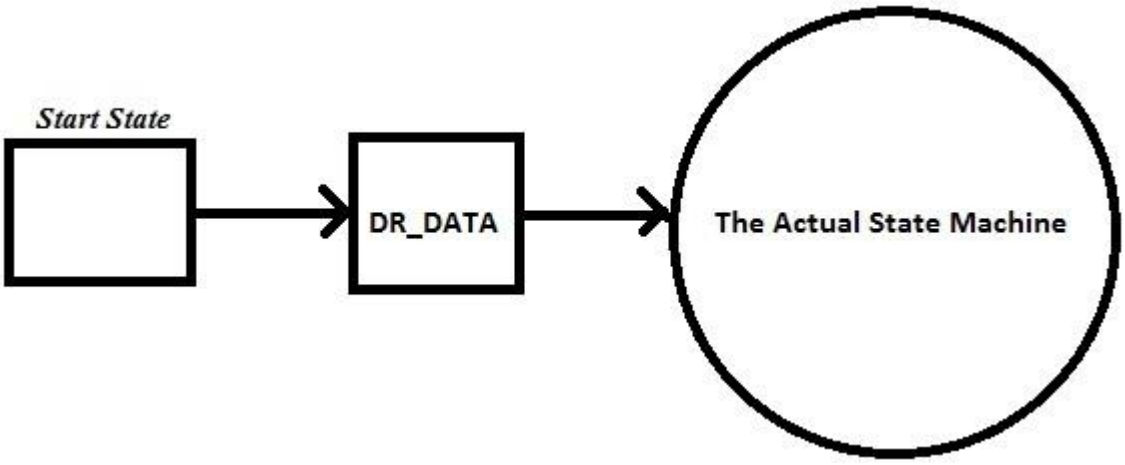
The events from B to F are made to occur before the actual data transfer from UBS to GPIF starts. As soon as the DMA Channel is created, we send a custom made buffer to the Consumer (GPIF) side and make the state machine to drive this buffer out.

This can be implemented using the following code which is inserted after the DMA Channel is created:

```
uint8_t TempBuf[32] __attribute__ ((aligned (32)));
CyU3PDmaBuffer_t    Buf;

     Buf.buffer = (uint8_t *)TempBuf;
     Buf.count  = 32;
     Buf.size   = 32;
     Buf.status = 0;

/* Send the Sample Buffer to the PIB Consumer */

    apiRetStatus = CyU3PDmaChannelSetupSendBuffer (&glDmaChHandle, &Buf);
    if (apiRetStatus != CY_U3P_SUCCESS)
      {
          CyU3PDebugPrint (4, "Setupsendbuffer failed = %d\n", apiRetStatus);
          CyFxAppErrorHandler (apiRetStatus);
      }

    /* Load and start the GPIF state machine. */
    apiRetStatus = CyU3PGpifLoad (&CyFxGpifConfig);
    if (apiRetStatus != CY_U3P_SUCCESS)
    {
        CyU3PDebugPrint (4, "CyU3PGpifLoad failed, error code = %d\n", apiRetStatus);
        CyFxAppErrorHandler (apiRetStatus);
    }

    CyU3PGpifSocketConfigure (0,CY_U3P_PIB_SOCKET_0,6,CyFalse,1);


/* Start the State Machine */


    apiRetStatus = CyU3PGpifSMStart (START, ALPHA_START);
      if (apiRetStatus != CY_U3P_SUCCESS)
      {
          CyU3PDebugPrint (4, "CyU3PGpifSMStart failed, error code = %d\n",
apiRetStatus);
          CyFxAppErrorHandler (apiRetStatus);
      }

/* Start the DMA Channel Data Transfer */

    apiRetStatus = CyU3PDmaChannelSetXfer (&glDmaChHandle, CY_FX_GPIFTOUSB_DMA_TX_SIZE);
    if (apiRetStatus != CY_U3P_SUCCESS)
    {
        CyU3PDebugPrint (4, "CyU3PDmaChannelSetXfer failed, Error code = %d\n",
apiRetStatus);
        CyFxAppErrorHandler(apiRetStatus);
    }
```

In the State Machine also, initially the DR_DATA action should be called as soon as the state machine is started inside an additional state, which is appended to the original

state machine. This is done so that the sample TempBuf committed inside the firmware can be read out by the State Machine. This makes the DMA_Watermark flag to reach the state F, and it can henceforth be used to indicate the beginning of a Data Transfer also. The DR_DATA action should be repeated enough such that all the data in the dummy TempBuf buffer is read out.



Note: This workaround may be even simple for a Manual Channel, but for applying for an AUTO Channel also, the above workaround may be used.

GPIF to USB Transfer:

The flags associated with GPIF Sockets that transfer data to the USB, behave as full flag. If they are asserted, it means that the Current Buffer inside FX3 is full, if not it means that the buffer has space inside it for accommodating some more incoming data. Before any data is transferred, the full flag remains de-asserted. When a buffer gets filled, the full flag is asserted after 3 clock cycles and remains asserted until the next buffer becomes active This time is usually from 1.2 uS and can go up to 5 uS.

It is a usual practice to keep the flags active low, so that the external processor can get to know about the availability of Buffers to write, by means of a de-assertion, as soon as the channel is created. (Low to high transition) If they were set logic high, this transition may not be noted because the flags would remain low even before the channel is created and continue to remain low even after that.

Unlike those in USB to GPIF transfer, the Watermark flags in GPIF to USB Transfer are actually able to indicate when to begin a transfer. The confusion here arises in the scenarios where the DMA Channel is created after the state machine is started. In this case, as soon as the State Machine is started the DMA_Watermark flag goes high (de-asserted), irrespective of whatever be the “Initial Value” specified in the state machine. If the

DMA_Watermark flags are monitored for starting a transfer, the external processor would start sending the data as soon as the state machine is started, without knowing that the DMA Channel has not yet been created. But here scenario, the DMA_Ready flag works fine. It goes into High State only after the DMA Channel is created.

So for GPIF to USB Transfer, the Watermark flags can be used for starting a transfer, but it should be ensured the State machine is started after the channel is created. In the Example in An65974 the DMA Channels are created after SM is started, so many customers follow the same, and face issue with using the DMA_Watermark flag for beginning a transfer.

_____ END OF MEMO _____