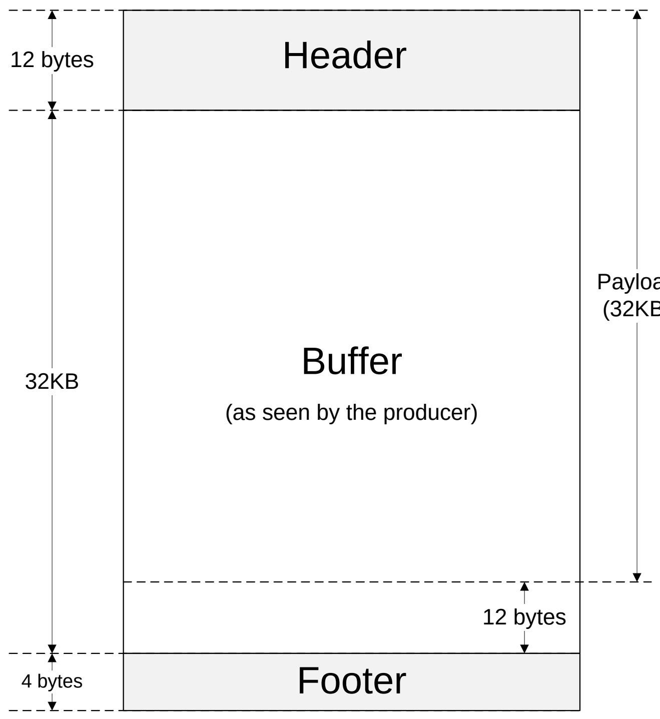
Move the UVC header addition and buffer commit code to the DMA callback.

In the default UVC project, adding a header to the produced buffer and committing it will be done in the frame blanking interval. Since this duration is reduced on a higher density stream, the header addition and commit code is moved to the DMA callback to be processed much faster.

Image payload size increased from 16K-4 bytes to 32K bytes

Having large buffers allows for larger uninterrupted transfers. So, the DMA buffer size was increased to 32K.
In addition, the payload was made exactly equal to 32K. The reasoning behind this is that shorter payloads could have caused the transfers to prematurely terminate on the host which may have led to reduced frame rates.

This is achieved by having a DMA channel with 32K+16 byte buffers, 12byte header and 4 byte footer. See the figure below for an illustration.



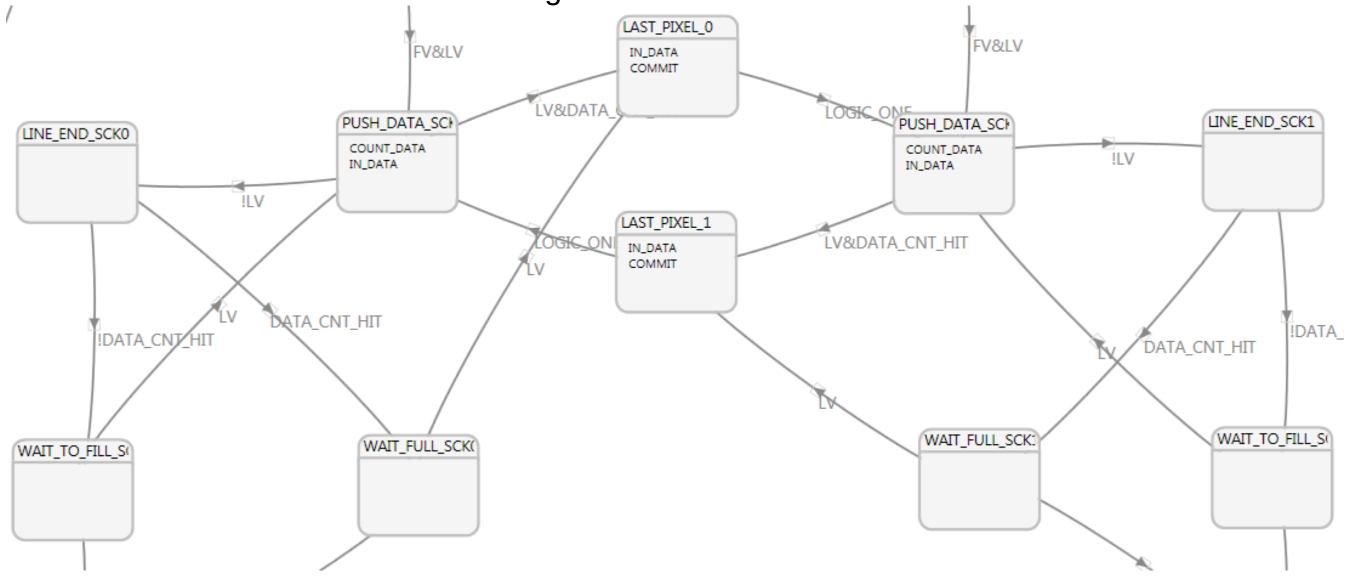
In this strategy, the GPIF state machine must be modified to fill 32K-12 bytes of data. Since this will not cause the buffer to be filled, the committing process will not happen automatically and so the COMMIT action must be used to force a wrapup.

So, the state machine reads in `(32K-12) – (gpif_bus_width/8)` bytes of data in one state which then branches to another state which reads one word (depending on GPIF data bus width) and commits the buffer.

The data counter's limit hence becomes

$$counter\ limit\ = \frac{dma\ buffer\ size\ (in\ bytes)\ as\ seen\ by\ producer - 12}{gpif\ bus\ width/_{8}} - 2$$

The state machine structure now changes to:



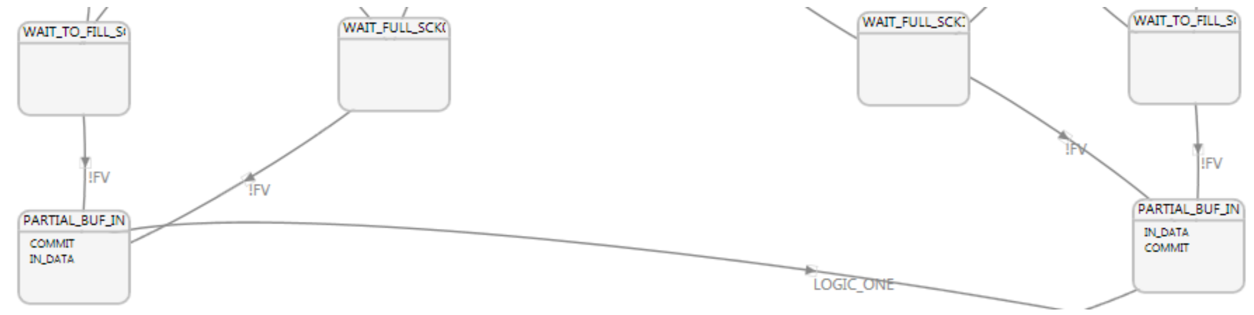
Here, the PUSH_DATA_SCK_x states read data until the counter hits after which the LAST_PIXEL_x state reads the last word and commits the buffer.

Remove the GPIF interrupt to the CPU

In the existing project, the ARM9 CPU is interrupted on frame completion to wrap the current partial buffer and toggle the Frame ID bit of the UVC header.
This interrupt is removed and the functions are replaced by

1. Wrapup is now done using the COMMIT action with a dummy IN_DATA action to prevent a zero length buffer being committed.
2. The Frame ID bit is toggled in the DMA callback after the last buffer of the frame is committed.

The relevant section of the state machine is:



Since a partial buffer is always produced, both the WAIT states will transition to a PARTIAL_BUF_IN_SCK_x state which commits the last buffer whilst reading in 1 word of data. This is to prevent a ZLB.

This extra word is compensated in the DMA callback as:

```
CyU3PDmaBuffer_t * produced_buffer = &(input->buffer_p);

if (produced_buffer->count == CY_FX_UVC_BUF_FULL_SIZE) {
    CyFxUVCAddHeader (produced_buffer->buffer - CY_FX_UVC_MAX_HEADER,
                                           CY_FX_UVC_HEADER_FRAME);
} else {
    /* If we have a partial buffer, this is guaranteed to be
       the end of the video frame for uncompressed images. */
    CyFxUVCAddHeader (produced_buffer->buffer - CY_FX_UVC_MAX_HEADER,
                                           CY_FX_UVC_HEADER_EOF);
    /* Toggle UVC header FRAME ID bit */
    glUVCHeader[1] ^= CY_FX_UVC_HEADER_FRAME_ID;

    /* Remove the dummy data */
    produced_buffer->count -= 4;
}

/* Commit the updated DMA buffer to the USB endpoint. */
prodCount++;
apiRetStatus = CyU3PDmaMultiChannelCommitBuffer (&glChHandleUVCStream,
                   produced_buffer->count + CY_FX_UVC_MAX_HEADER, 0);
if (apiRetStatus != CY_U3P_SUCCESS) {
    prodCount--;
    CyU3PDebugPrint (4, "Err/MC %x: %dB; counts: %d %d\r\n", apiRetStatus,
                   produced_buffer->count, prodCount, consCount);
}
```

The GPIF data bus is 32 bits wide and so 4 words are skipped during commit. For smaller buses, the compensation will vary.

Here CY_FX_UVC_BUF_FULL_SIZE == 32K – 12.

Remove the channel reset and restart

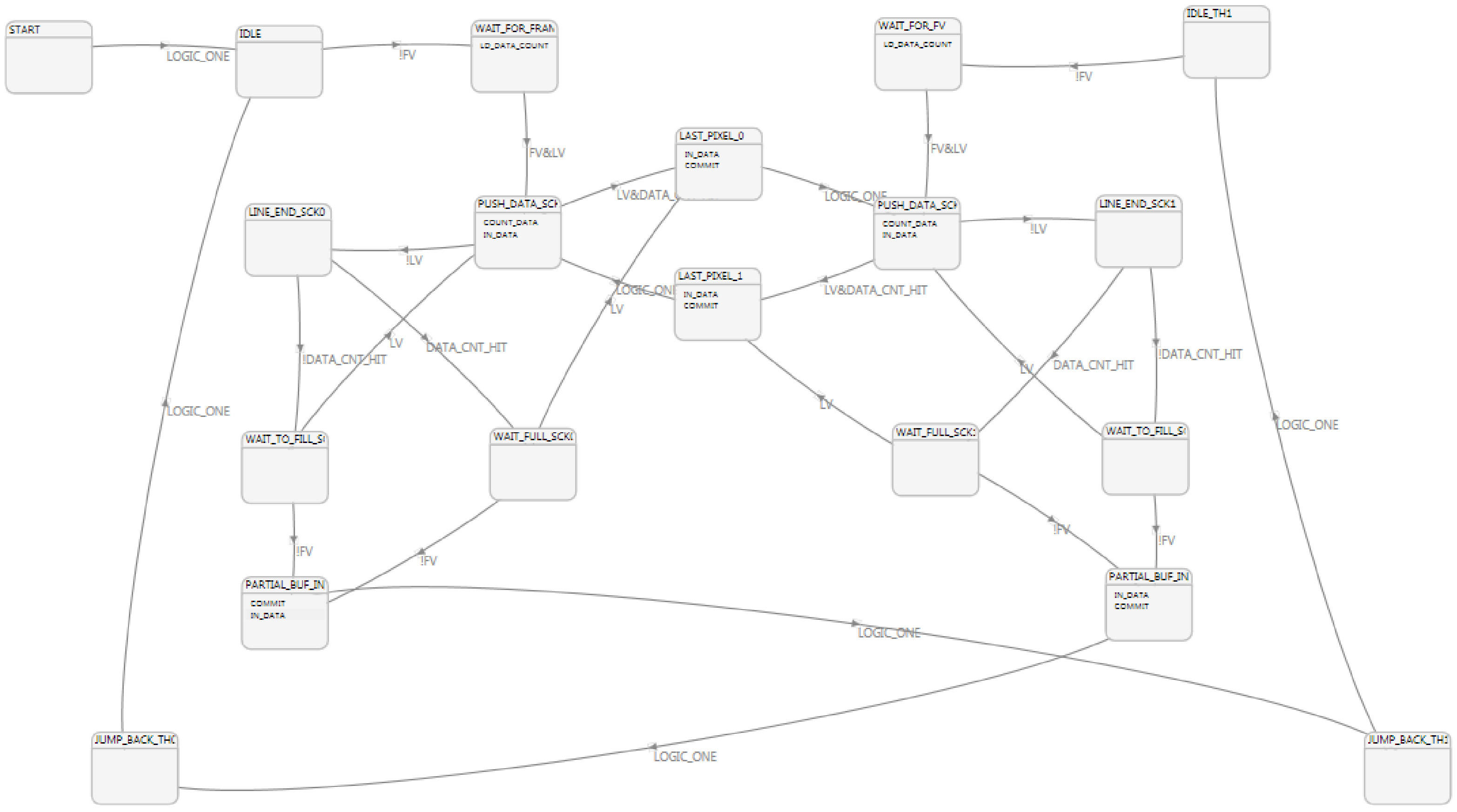
The channel is reset and restarted after a frame is transferred. This is to reset the producer descriptor index to that of the first socket as the state machine starts transferring data to that socket when started.

Since this code executes during frame blanking, it has been removed and instead the state machine is modified to account for the index change.

If the frame ends while in thread 0, the state machine then starts reading into thread 1 as it is the next in sequence. Similarly, if the last buffer was in thread 1, the image data is then read into thread 0.

In the state machine, the PARTIAL_BUF_IN_SCK_x states commit the last partial buffer and then jump to the JUMP_BACK_TO_TH_x states which jump to the IDLE states to wait for the next frame.

Since the PARTIAL_BUF_IN_SCK_0 state commits data in thread 0, it jumps to JUMP_BACK_TO_TH_1 which starts reading data into thread 1 and similarly the state that commits thread 1 jumps back into reading data to thread 0.



START
IDLE
LOGIC_ONE
!FV
WAIT_FOR_FRAN
LD_DATA_COUNT
WAIT_FOR_FV
LD_DATA_COUNT
IDLE_TH1
FV&LV
LAST_PIXEL_0
IN_DATA
COMMIT
LV&DATA_CNT_HIT
LOGIC_ONE
PUSH_DATA_SCK
COUNT_DATA
IN_DATA
LINE_END_SCK0
LINE_END_SCK1
!LV
LAST_PIXEL_1
LV
DATA_CNT_HIT
!DATA_CNT_HIT
WAIT_TO_FILL_S
WAIT_FULL_SCK
PARTIAL_BUF_IN
JUMP_BACK_TH0
JUMP_BACK_TH1