University of Windsor

Electrical and Computer Engineering

ELEC-4430: Embedded System Design

Data Buffering System for FPGA, Progress Report 3

Last Name: **ILIEVSKI**
First Name: **BRAJAN**

March 8, 2021

Testbench Timing Issue and Resolution:

In the second progress report, we encountered timing issue with the write function of the FIFO asynchronous buffer. Particularly, the fifo_full flag is active-high after 7 words, indicating the FIFO is fully loaded with words in only 7 rising edges. This can be further explained by the fact that fifo_full flag and write increment pointer are two different process that run in parallel (attached below).

```
--set full flag
full <= '1' when writePtr + '1' = syncReadPtrBin else '0';
fifo_full <= full;

-- write pointer handling
if enW_fill = '1' and not full = '1' then
    writePtr <= writePtr + '1';
end if;
```

When the write enable (enW_fill) is selected (active high signal) and the full condition has not met yet (active low signal), the write pointer of the FIFO system is incremented by one for every rising edge of iclk. For instance, referring to figure 1, the write pointer is incremented for 000 to 001 as soon as enW_fill is asserted, and the first word is written in the FIFO buffer as a result. However, there was an issue that the last word (8th word) has not been written yet the full flag is raised as indicated in Progress Report 2. Regarding the code, both write increment process and set full flag are running in parallel; therefore, at the last write process, write pointer is incremented from 111 to 000 (rollover case), which equal to read pointer and full is set to “1” immediately. Simultaneously, the newly asserted full condition prevents the write operation to increment, and 8th word is not written to FIFO buffer.

An adjustment in the code that helps to fix the timing issue is to add a small delay in set full flag operation.

```
--new set full flag
process (iclk)
begin
if rising_edge(iclk) then
  if (writePtr + '1' = syncReadPtrBin) then
  fifo_full_1 <= '1';
  full <= fifo_full_1;
  else
  fifo_full_1 <= '0';
  full <= fifo_full_1;
  end if;
  end if;
  end process;
  fifo_full <= full;
```

The new implemented code introduced a new signal (fifo_full_1), which store the full condition with 1 and 0 according to the if condition. With the new signal, full condition is delayed by exactly one ick rising edge and prevent any time overlapping issue with parallel running processes. As expected, the last word is now written into ram. However, there is another issue with the fifo_full condition. The flag when back to zero instead of keep staying high-active (indicating that the fifo is not full after the 8th word is written)

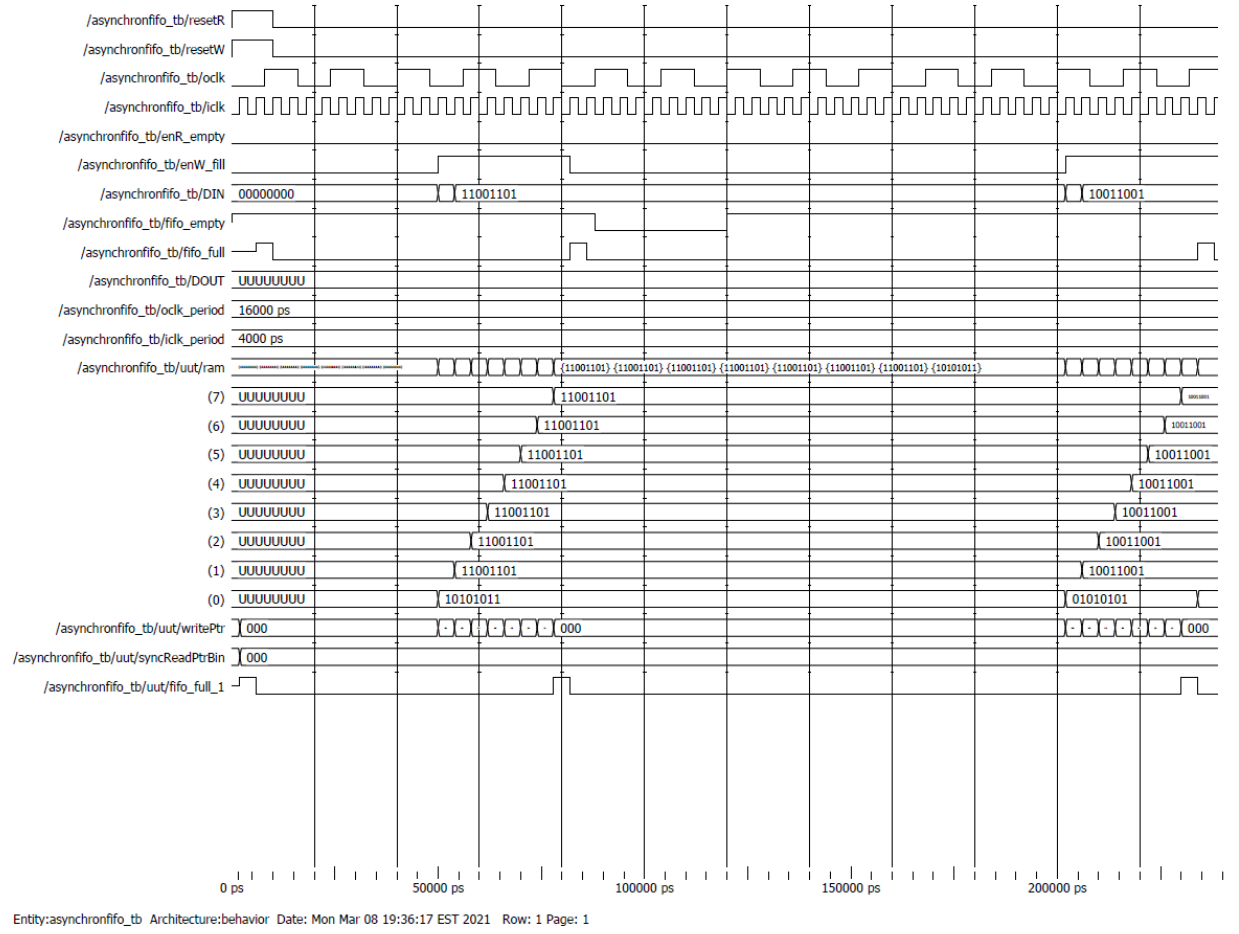


Figure 1: write process output of the asynchronous FIFO buffer system by testbench

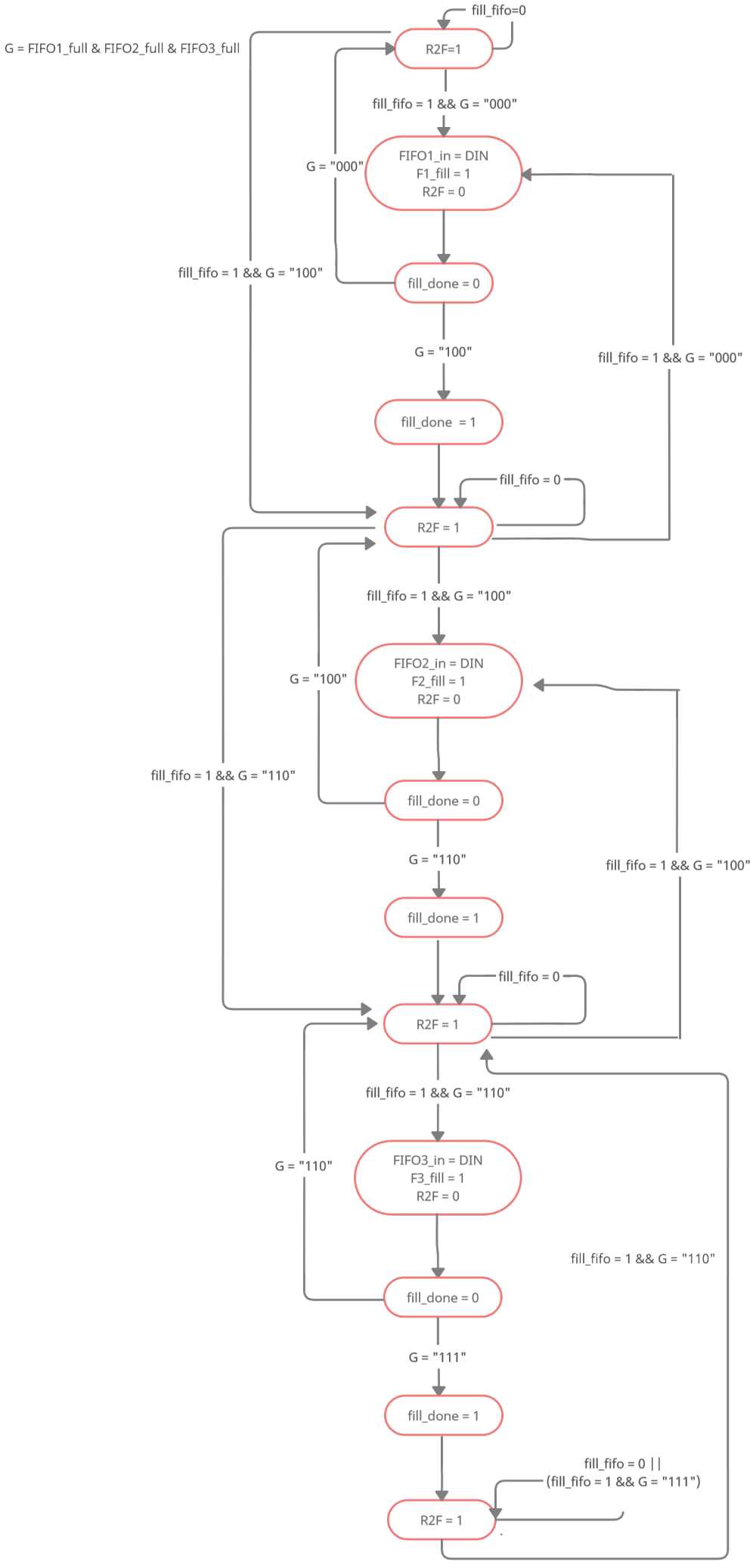
FIFO Handshaking Process:

Further efforts for this progress report focused on developing the ASM charts and VHDL code for the processes responsible for handling the sequential reading and writing of the three FIFOs. Given that the write and read must be executed in the order FIFO1, FIFO2, and FIFO3, the ASM chart focuses on ensuring that this order is followed whenever a FIFO in the priority is freed. For writing to the FIFOs, the ASM chart begins by defining a 3-bit signal G, which is a concatenation of the *full* outputs of each FIFO and where the MSB starts with the *full* of FIFO1 (G = F1_full & F2_full & F3_full).

At the initial state where R2F (ready_to_fill) is equal to 1, we evaluate whether the *fill_fifo* input has been asserted and whether all FIFO's are not full (G = "000"). When the latter condition is met, filling of the first FIFO is initiated whereby the FIFO input receives DIN from the sender signal. Upon exiting this state, *fill_done* is kept low and two conditions are checked. If FIFO1 is still not filled (G = "0XX"), we revert back to the initial state and expended additional clock cycles to fill up the first FIFO with additional 8-bit words. When filling of the first FIFO is completed (G = "1XX"), the ASM proceeds to the next state and *fill_done* is asserted. In the next ready to fill state, we evaluate several important conditions:
1) *fill_fifo* is no longer asserted and the current state is maintained.
2) *fill_fifo* is asserted but FIFO1 has been read (G = "000"). In this case FIFO1 is no longer full and the flow reverts back to the filling of FIFO1.
3) *fill_fifo* is asserted, FIFO1 has not been read, and FIFO2 is still not full (G = "100"). In this case the flow proceeds to fill in FIFO2 with 8-bit words following a similar flow to that of FIFO1.
4) *fill_fifo* is asserted, FIFO1 has not been read, and FIFO2 is fill (G = "110"). In this case the flow skips the filling of FIFO2 is flows to the next ready to fill state of FIFO3.

In the third ready to fill state, a similar evaluation of conditions is encountered; however, in this state the flow can only revert back to the ready to fill state of FIFO2 when FIFO2 has been read and no longer full (G = "100"). When FIFO3 is not full, the flow proceeds to fill FIFO3 until its respective full flag is asserted (G = "111"). In this case, the flow proceeds to the fourth and last ready to fill state where we evaluate several conditions: 1) maintain the final state when *fill_fifo* is no longer asserted *or* if *fill_fifo* is asserted but all FIFOs are full (this prevents overwriting), 2) revert back to the third ready to fill state when FIFO3 is read and no longer full.

Further efforts will continue to develop the necessary VHDL code for this ASM chart and the necessary code for dealing with the reading logic required to handle the receiver. This will finalize our design and further testing of the holistic system will be carried out.


fill_fifo=0
R2F=1
G = FIFO1_full & FIFO2_full & FIFO3_full
fill_fifo = 1 && G = "000"
FIFO1_in = DIN
F1_fill = 1
R2F = 0
G = "000"
fill_done = 0
fill_fifo = 1 && G = "100"
G = "100"
fill_done = 1
fill_fifo = 1 && G = "000"
fill_fifo = 0
R2F = 1
fill_fifo = 1 && G = "100"
FIFO2_in = DIN
F2_fill = 1
R2F = 0
G = "100"
fill_done = 0
fill_fifo = 1 && G = "110"
G = "110"
fill_done = 1
fill_fifo = 1 && G = "100"
fill_fifo = 0
R2F = 1
fill_fifo = 1 && G = "110"
FIFO3_in = DIN
F3_fill = 1
R2F = 0
G = "110"
fill_done = 0
fill_fifo = 1 && G = "110"
G = "111"
fill_done = 1
fill_fifo = 0 ||
(fill_fifo = 1 && G = "111")
R2F = 1