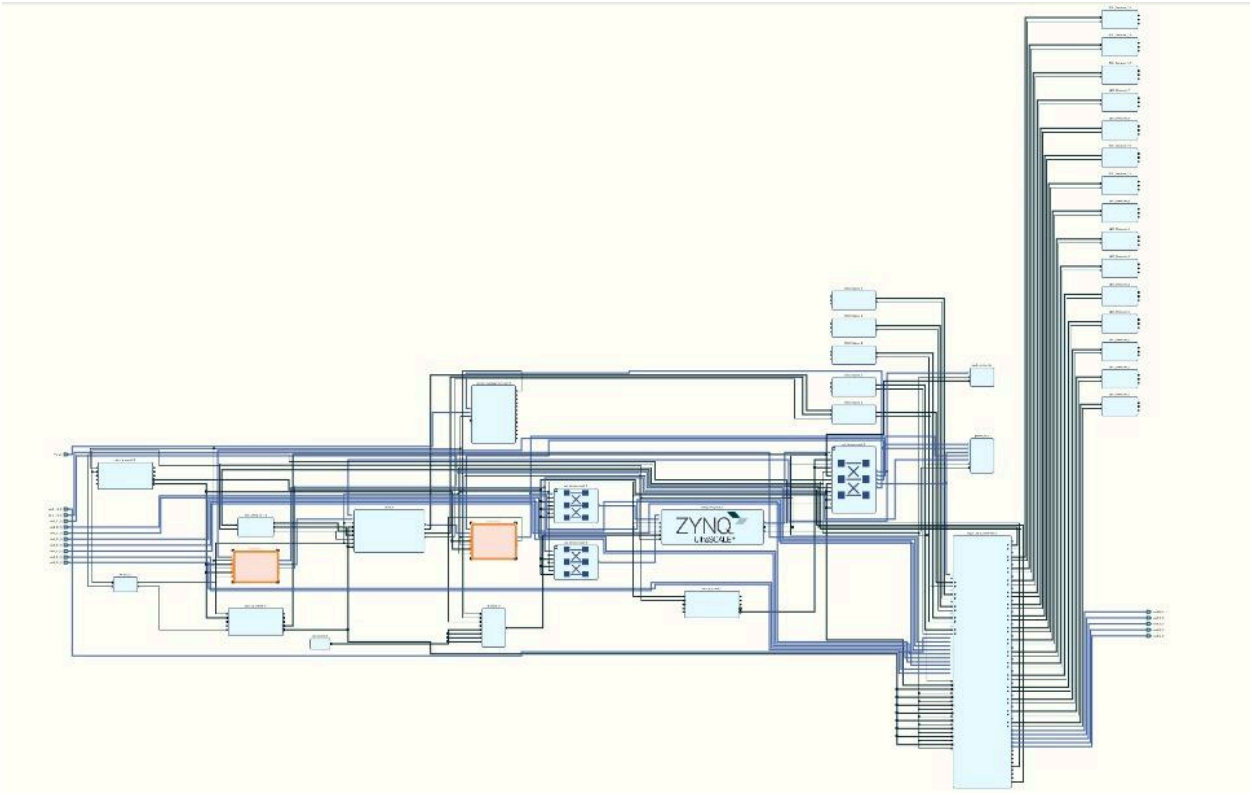
QPSK implementation on HTG-ZRF8

- Previously, I was successful in generating the devicetree for the DMA Master IP (provided by Xilinx) and today I worked on the DMA IP generated by Matlab.
- That DMA IP is actually provided by the analog devices and it is **dmac (DMA controller**).
- Two of these IPs are used in the QPSK design, one as **S2MM** and other as **MM2S**. Following is the QPSK design [axi_dma_s2mm and axi_dma_mm2s IPs are highlighted].



- Following the below link, I updated the device tree node and am able to get the ADI DMA controller detected in the kernel through the device tree:

https://android.googlesource.com/kernel/common/+/android-trusty-4.4/Documentation/devicetree/bindings/dma/adi%2Caxi-dmac.txt

```
axi_dmac_s2mm: axi_dmac_0@A0020000 {
            compatible    = "adi,axi-dmac-1.00.a";
            status        = "okay";
            reg           = <0x0 0xA0020000 0x0 0x10000>;

            #dma-cells = <1>;                /* 0=MM2S, 1=S2MM */

            interrupt-parent = <0x5>;
            interrupts = <0 123 4>;
            clocks        = <&zynqmp_clk 71>;

        adi,channels {
                #size-cells = <0>;
                #address-cells = <1>;
                dma-channel@A0020000 {
                    reg = <0>;
                    adi,source-bus-width = <32>;
                    adi,source-bus-type = <1>;
                    adi,destination-bus-width = <32>;
                    adi,destination-bus-type = <0>;

                };
            };
        };
```

- The values for source and destination bus are as follow:

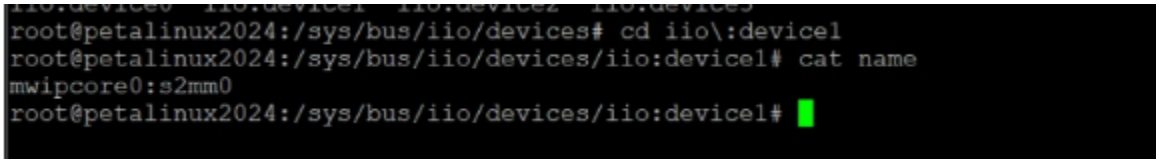
```
- adi,source-bus-type,
  adi,destination-bus-type: Type of the source or destination bus. Must be one
  of the following:
       0 (AXI_DMAC_TYPE_AXI_MM): Memory mapped AXI interface
       1 (AXI_DMAC_TYPE_AXI_STREAM): Streaming AXI interface
       2 (AXI_DMAC_TYPE_AXI_FIFO): FIFO interface
```

And following the below link, I updated the DMA channel node in the IP node:

https://android.googlesource.com/kernel/common/+/android-trusty-4.4/Documentation/devicetree/bindings/dma/dma.txt

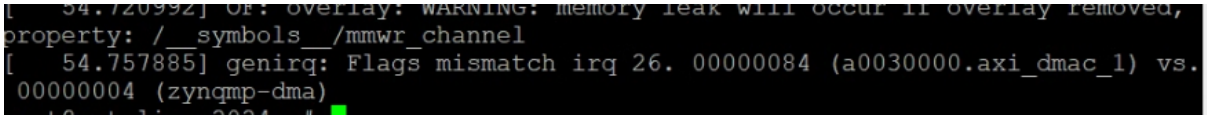
```
stream-channel@0 {
    reg = <0x0>
    compatible = "mathworks,axi4stream-s2mm-channel-v1.00";
    dma-names = "s2mm";
    dmas = <&axi_dmac_s2mm 0x0>;
    mathworks,dev-name = "s2mm0";
    mathworks,sample-cnt-reg = <0x8>;
    data-channel@0 {
        reg = <0x0>;
        compatible = "mathworks,iio-data-channel-v1.00";
        mathworks,data-format = "u32/32>>0";
    };
};
```

DMA shows as IIO device:



For the 2nd DMA, i faced some issues regarding the interrupt:

I connected it to interrupt number 3 of pl_ps_irq pin of zynqmp and assigned interrupt ID 124 in the device tree. But it resulted in this error:



On checking the base device tree, i found out that these interrupts are already taken:

```
dma-controller@fd500000 {
    status = "okay";
    compatible = "xlnx,zynqmp-dma-1.0";
    reg = <0x00 0xfd500000 0x00 0x1000>;
    interrupt-parent = <0x05>;
    interrupts = <0x00 0x7c 0x04>;
    clock-names = "clk_main\0clk_apb";
    #dma-cells = <0x01>;
    xlnx,bus-width = <0x80>;
    iommus = <0x13 0x14e8>;
    power-domains = <0x12 0x2a>;
    clocks = <0x04 0x13 0x04 0x1f>;
    phandle = <0x41>;
};

dma-controller@fd510000 {
    status = "okay";
    compatible = "xlnx,zynqmp-dma-1.0";
    reg = <0x00 0xfd510000 0x00 0x1000>;
    interrupt-parent = <0x05>;
    interrupts = <0x00 0x7d 0x04>;
    clock-names = "clk_main\0clk_apb";
    #dma-cells = <0x01>;
    xlnx,bus-width = <0x80>;
    iommus = <0x13 0x14e9>;
    power-domains = <0x12 0x2a>;
    clocks = <0x04 0x13 0x04 0x1f>;
    phandle = <0x42>;
};

dma-controller@fd520000 {
    status = "okay";
    compatible = "xlnx,zynqmp-dma-1.0";
    reg = <0x00 0xfd520000 0x00 0x1000>;
    interrupt-parent = <0x05>;
    interrupts = <0x00 0x7e 0x04>;
```

So i connected it to an available pin:

Pin Mapping to Interrupt IDs

Interrupt ID	Pin
121	pl_ps_irq0 [0]
122	pl_ps_irq0 [1]
123	pl_ps_irq0 [2]
124	pl_ps_irq0 [3]
125	pl_ps_irq0 [4]
126	pl_ps_irq0 [5]
127	pl_ps_irq0 [6]
128	pl_ps_irq0 [7]
136	pl_ps_irq1 [0]
137	pl_ps_irq1 [1]
138	pl_ps_irq1 [2]
139	pl_ps_irq1 [3]
140	pl_ps_irq1 [4]
141	pl_ps_irq1 [5]
142	pl_ps_irq1 [6]
143	pl_ps_irq1 [7]

Table:

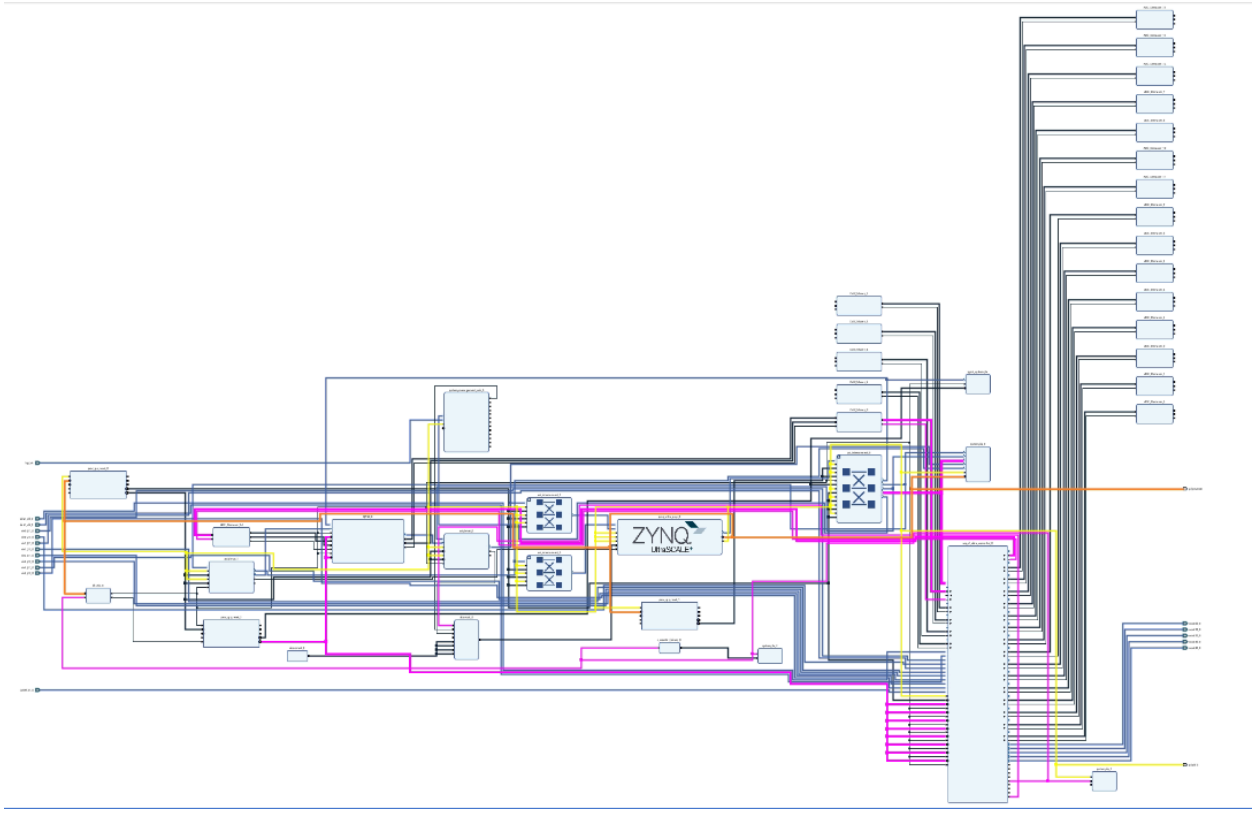
i.e. 136.

PLL configuration:

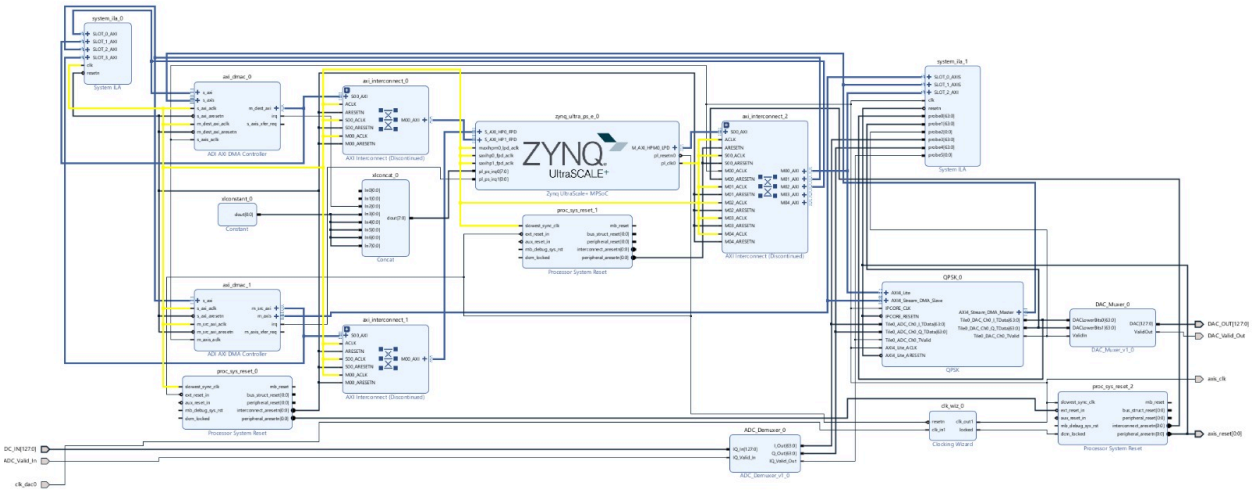
- After that, the DAC clock of the RFDC IP in Vivado was not providing any output. I tried debugging the current design but wasn’t successful.

- While doing some research online, I came across a repository that provides HTG-ZRF8 based examples: alexforencich/taxi.
- After exploring the repository, I navigated to the following path which contains the HTG-ZRF8 project:
  `taxi/src/eth/example/HTG_ZRF8/fpga/fpga_R2_ZU48DR/`.
- From this directory, I opened the **fpga.xpr** file in **Vivado** to inspect the project setup.
- Upon reviewing the project, I noticed that it does not include a block design. Instead, the RF Data Converter (RFDC) IP is instantiated directly in the SystemVerilog code.
- As a first step, I decided to test the provided example to verify whether the RFDC clocks were functional. The test confirmed that the RFDC clocks were indeed working perfectly in this setup.
- After confirming that, I proceeded to create a new block design. In this design, I intentionally did not include the RFDC IP itself.
- For all the pins that were connected to the RFDC IP in the original SystemVerilog instantiation, I made them external in my block design to interface them with the RFDC IP.

- Following is the complete block design [**Wires coming from and going to RFDC IP are highlighted in Purple color**]



- Following is the design without RFDC ip and all the connections made external.



- After this, I instantiated the block design in the top-level module and made relevant connections between the block design and the RFDC IP.

- Following is the instantiated block design.

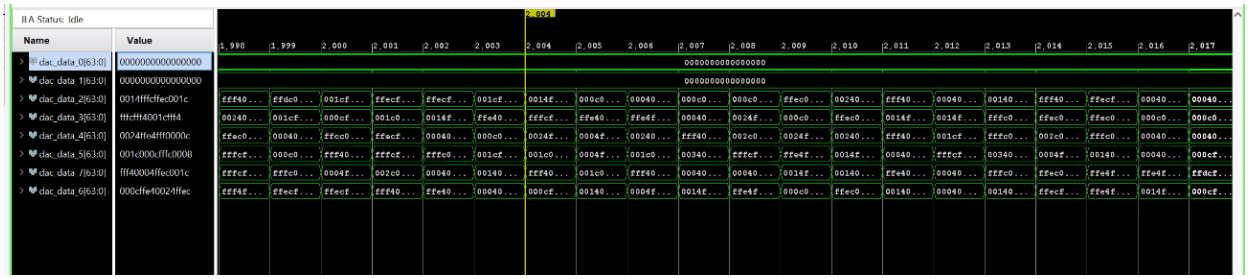
```
wire [127:0]ADC_IN;
wire ADC_Valid_In;
wire [127:0]DAC_OUT;
wire DAC_Valid_Out;
wire axis_clk;
wire [0:0]axis_reset;
wire clk_dac0;

qpsk_for_htg qpsk_for_htg_i
     (.ADC_IN(ADC_IN),
      .ADC_Valid_In(ADC_Valid_In),
      .DAC_OUT(DAC_OUT),
      .DAC_Valid_Out(DAC_Valid_Out),
      .axis_clk(axis_clk),
      .axis_reset(axis_reset),
      .clk_dac0(dac_clk_out[0]));
```

- Following is the RFDC IP instance

```
usp_rfdc_0 rfdc_inst (
    // Common
    .sysref_in_p(rfdc_sysref_p),
    .sysref_in_n(rfdc_sysref_n),
    .s_axi_aclk(axil_rfdc_clk),
    .s_axi_aresetn(!axil_rfdc_rst),
    .s_axi_awaddr(axil_rfdc.awaddr),
    .s_axi_awvalid(axil_rfdc.awvalid),
    .s_axi_awready(axil_rfdc.awready),
    .s_axi_wdata(axil_rfdc.wdata),
    .s_axi_wstrb(axil_rfdc.wstrb),
    .s_axi_wvalid(axil_rfdc.wvalid),
    .s_axi_wready(axil_rfdc.wready),
    .s_axi_bresp(axil_rfdc.bresp),
    .s_axi_bvalid(axil_rfdc.bvalid),
```

Note: Only a few inputs and outputs are shown in the instance of RFDC IP.

- Then I generated the bitstream and programmed the hardware. (At this point I was not using the SD card and the FPGA was in jtag mode).
- Finally, I got some output, which means the RFDC IP is functioning and generating the clock required for QPSK.



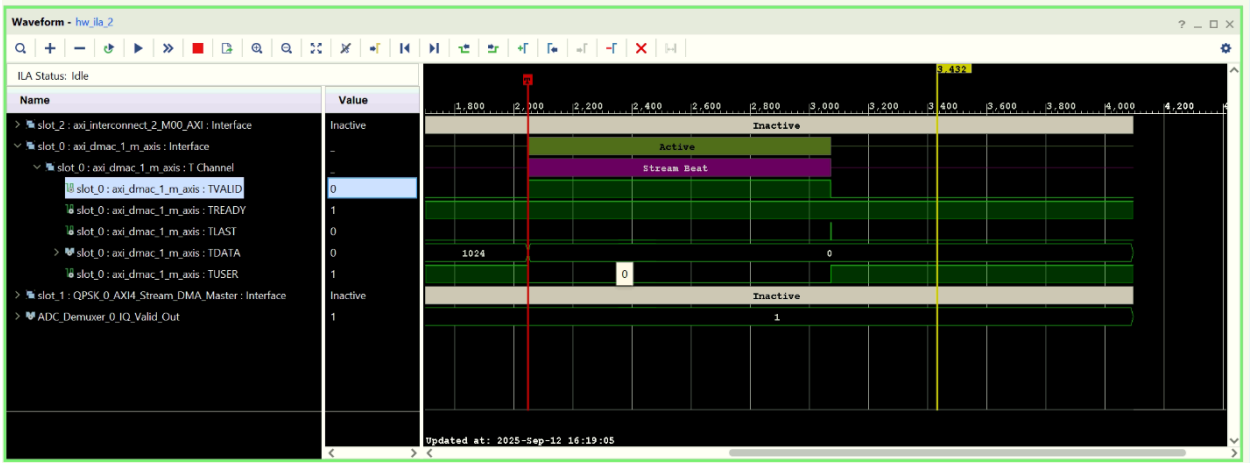
- After this i inserted the SD card back to the board and loaded the bitstream and devicetree overlay using the following instructions.
  fpgautil -b fpga.bit
  fpgautil -o qpsk_dt_overlay.dtbo
  systemctl restart iiod 2>/dev/null || { killall iiod 2>/dev/null; iiod -F & }

```
root@petalinux2024:~#
root@petalinux2024:~#
root@petalinux2024:~#
root@petalinux2024:~# fpgautil -b fpga.bit
Time taken to load BIN is 479.000000 Milli Seconds
BIN FILE loaded through FPGA manager successfully
root@petalinux2024:~# fpgautil -o qpsk_dt_overlay.dtbo
mkdir: cannot create directory '/configfs': File exists
[  177.147451] OF: overlay: WARNING: memory leak will occur if overlay removed,
property: /__symbols__/axi_dmac_s2mm
[  177.157809] OF: overlay: WARNING: memory leak will occur if overlay removed,
property: /__symbols__/axi_dmac_mm2s
[  177.168156] OF: overlay: WARNING: memory leak will occur if overlay removed,
property: /__symbols__/mwipcore0
[  177.178149] OF: overlay: WARNING: memory leak will occur if overlay removed,
property: /__symbols__/mmrd_channel
[  177.188408] OF: overlay: WARNING: memory leak will occur if overlay removed,
property: /__symbols__/mmwr_channel
```

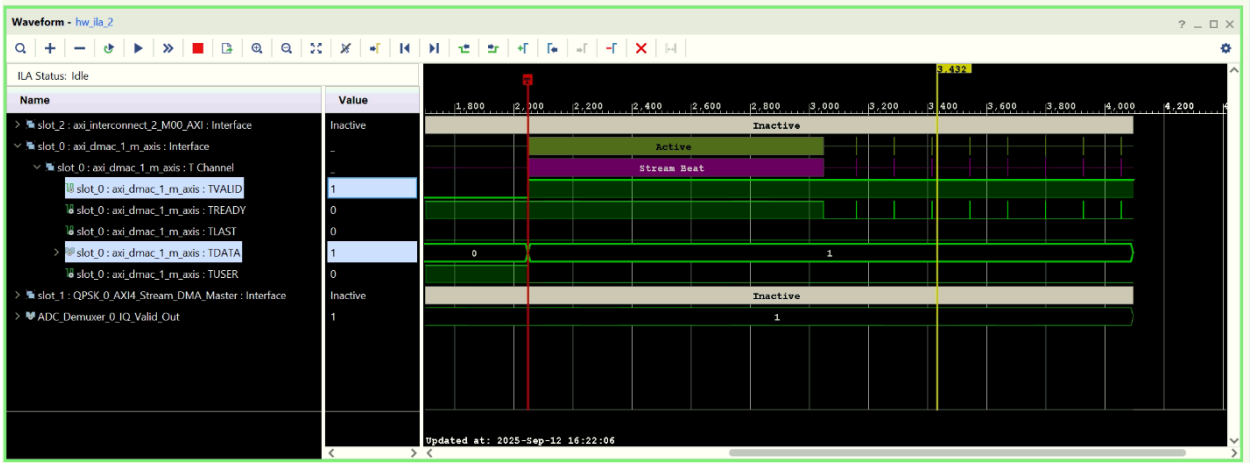
- Once uploaded I ran the matlab script called `gs_hdlcoder_QPSKTxRx_RFSoC_interface.m` and this time it executed without any errors.
- I modified the `gs_hdlcoder_QPSKTxRx_RFSoC_interface.m` script as follows

```
% There are two ways to write a DUT bus ports
% (1). Prepare a struct value and write it to the whole bus port.
% writePort(hFPGA, "regIn", struct());
% (2). Prepare a value for each member of the bus and write it individually.
writePort(hFPGA, "regIn.tx_enable", 1);
writePort(hFPGA, "regIn.rx_reset_cs", 0);
writePort(hFPGA, "regIn.rx_src_sel", 0);
writePort(hFPGA, "regIn.capture_start", 1);
writePort(hFPGA, "regIn.capture_length", 100);
writePort(hFPGA, "regIn.capture_src_sel", 1);
writePort(hFPGA, "regIn.tx_output_gain", zeros([1 1]));
writePort(hFPGA, "regIn.rx_input_gain", zeros([1 1]));
writePort(hFPGA, "regIn.capture_mode", 1);


% There are two ways to read a DUT bus ports
% (1). Read the whole bus port, and it will returns a struct value.
 data_regOut = readPort(hFPGA, "regOut");
% (2). Read each member of the bus individually.
 data_regOut_rx_nSynced = readPort(hFPGA, "regOut.rx_nSynced");
 data_regOut_rx_normCoarseFreqEst = readPort(hFPGA, "regOut.rx_normCoarseFreqEst");


%% AXI4-Stream DMA

 writePort(hFPGA, "mm2sData", zeros([1024 1]));
 data_s2mmData = readPort(hFPGA, "s2mmData");

%% Release hardware resources
release(hFPGA);
```

- In the second-last section of the script, under **AXI4-Stream DMA**, there is a `writeport` instruction that initiates the transfer of an array of size 1024, pre-initialized with all zeros.
- Following figure shows when TVALID goes high the transfer of 1024 zero happens



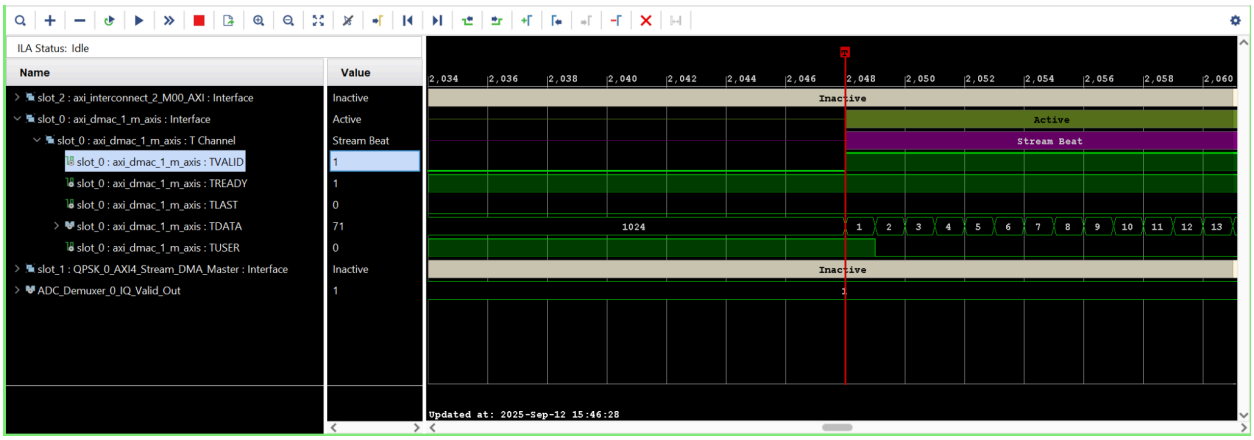
- I tried changing this instruction to `writePort(hFPGA, "mm2sData", ones([1024 1]));`
- Now a transfer of buffer of size 1024 containing all ones can be seen.



- Following modifications were done to the script (`gs_hdlcoder_QPSKTxRx_RFSoC_interface`) to transfer an array of size 1024 containing values from 1 to 1024.

```
% (1). Read the whole bus port, and it will returns a struct value.
 data_regOut = readPort(hFPGA, "regOut");
% (2). Read each member of the bus individually.
 data_regOut_rx_nSynced = readPort(hFPGA, "regOut.rx_nSynced");
 data_regOut_rx_normCoarseFreqEst = readPort(hFPGA, "regOut.rx_normCoarseFreqEst");


%% AXI4-Stream DMA
data = 1:1024;
 writePort(hFPGA, "mm2sData", data);
 data_s2mmData = readPort(hFPGA, "s2mmData");

%% Release hardware resources
release(hFPGA);
```

- . The following image shows the start of the transfer.



- The following image shows the end of the transfer.