SPI COMMUNICATION WITH SPARROW BOARD

VISHWA TEJA REDDY

B.Tech(ECE)

Intern

Specifications

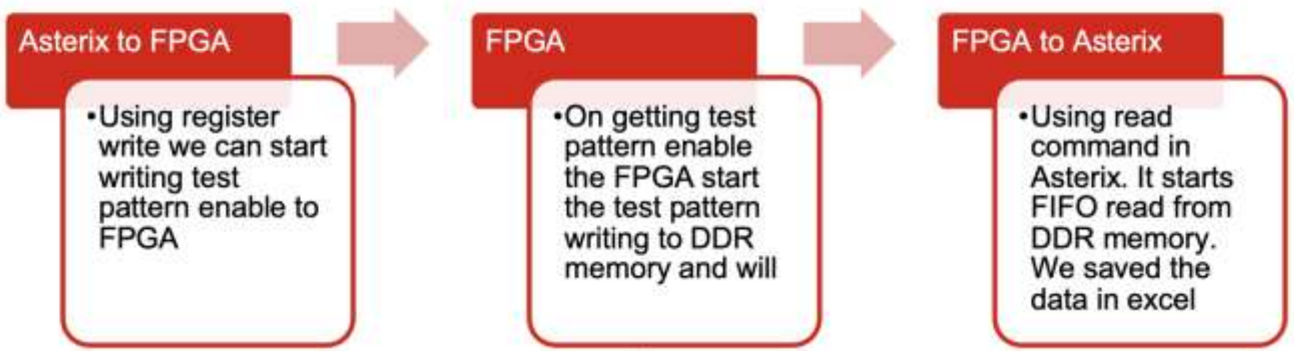
- Using maximum Data From DDR memory to write and Read
- Write and Reading Registers of AFE at 20MHz frequency using Interface
- Writing and Reading data From registers continuously at 20MHz Frequency using SPI interface
- Reading the Register data at 20MHz frequency Using SPI interface and updating this data to DDR and reading through Asterix using FX3 interface
- Reading the FIFO data at 20MHz frequency Using SPI interface and updating this data to DDR and reading through Asterix using FX3 interface

Agenda

- Arevo2p0 code is used to establish communication between Asterix and the Sparrow board.
- Using the test pattern included in the arevo2p0 code, test and verify the asterix setup.
- Find out how much DDR memory the previous code was using and increase it as much as possible.
- Understanding SPI lines and how data loads in SPI communication requires reading AFE.
- For the AFE, provide a standard Register read SPI module.
- Including delay adjustment options in standard SPI mode.
- Make a FIFO SPI mode that reads data constantly for the specified SPI cycles.
- Similar delay settings should be added for this mode.
- Remove any unusable blocks before adding these modules to the Arevo code. Utilise an oscilloscope to validate and test devices.
- Using the data from SPI, write a block that can generate 128-bit padpacket data for DDR.
- Through Asterix, confirm and test on hardware.
- Block that reads continuous REG SPI data from AFE while using register addresses immediately following the completion of FIFO SPI with enable. SPI all reg and SPI fifo along with packet maker
- All of these inputs are added in ADC RAW MODE in the Arevo2p0 code mux
- Test and verify on hardware.

Asterix Set Up

- We programmed the FPGA using arevo2p0 and the given bitstream files, then went to Asterix and sent commands to read data from the sparrow DDR memory.
- An incorporated test pattern that provides a ramp up with synchronised clock is available for setup verification.
- With pre-encoded fifo read depth and write depth, we were able to read and write around 6Mb of data from the DDR memory.


Asterix to FPGA
•Using register write we can start writing test pattern enable to FPGA
FPGA
•On getting test pattern enable the FPGA start the test pattern writing to DDR memory and will
FPGA to Asterix
•Using read command in Asterix. It starts FIFO read from DDR memory. We saved the data in excel

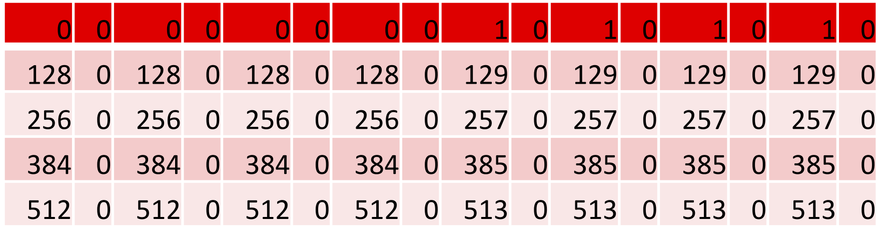
Maximum Capacity DDR

- We are using MT41K64M16TW-107 AAT:j, a micron technology chip with a 1Gb density, as our built-in DDR3 memory.
- We want to use more of the DDR3 memory, which has a maximum capacity of 1024Mb, so we're going to raise the read and write depths.
- We changed the read depth and write depth, and the highest amount of data we could use was 1020Mb. We utilised a ramp pattern for verification to demonstrate that we are utilising the entire RAM.

$$\begin{tikzcd} \text{6Mb} \arrow[r] & \text{1020Mb} \end{tikzcd}$$

Asterix output and observations

- In the extracted excel file we observed that there is a loss of a 16 bit packet data in the asterix because of the arevo2p0 code. We need to deal with it and

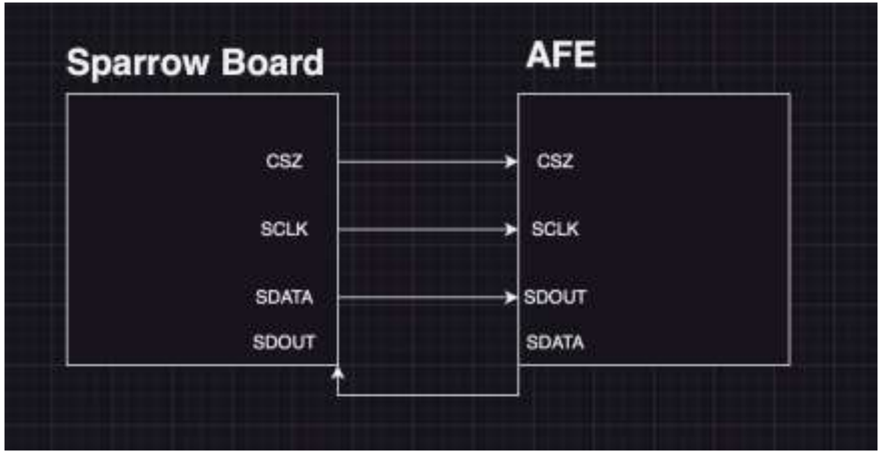


0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0
128	0	128	0	128	0	128	0	129	0	129	0	129	0	129	0
256	0	256	0	256	0	256	0	257	0	257	0	257	0	257	0
384	0	384	0	384	0	384	0	385	0	385	0	385	0	385	0
512	0	512	0	512	0	512	0	513	0	513	0	513	0	513	0

- In the above picture is when the test pattern is written into DDR memory, we can observe that first packet of 0 is missing.

SPI COMMUNICATIN and LINES

- As in the above block we are planning a SPI comm lines from Sparrow board to AFE with FIFO interrupts and REG interrupts.

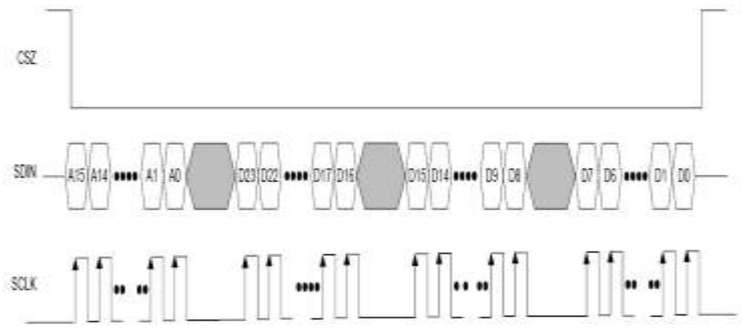


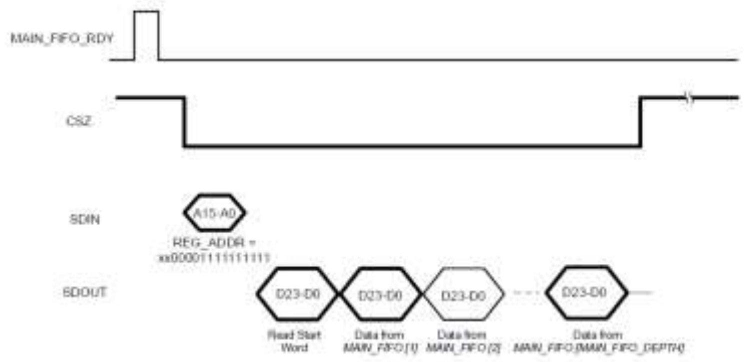
AFE

- There are two methods of SPI communication, as shown by the AFE data sheet. Register spi and fifo spi both use the same SPI lines in AFE, however fifo spi has two SPI communication depending on which AFE we are using.
- Therefore, register SPI is just regular SPI that employs a register interrupt pin before beginning the transmission and uses 24 bit data and 16 bit address bits.
- A 16-bit register with a fifo interrupt is sent out before communication, and the fifo has 24 bit data as well. There are pace fifo and main fifo, and the addresses for each are distinct.
- For reading the Main FIFO, the address should be set to xx00001111111111, and for reading the PACE FIFO, it should be set to xx00001111111110, where the first two bits stand for the Chip address.
- In below slide there are timing diagrams for both the FIFO read and REG read.
- In the first 24 bit data of fifo SPI we get the pointer difference that we got from AFE and we will be using that to continuously read data from the FIFO.

Timing Requirements to AFE

- The timing diagrams for the REG SPI and FIFO SPI

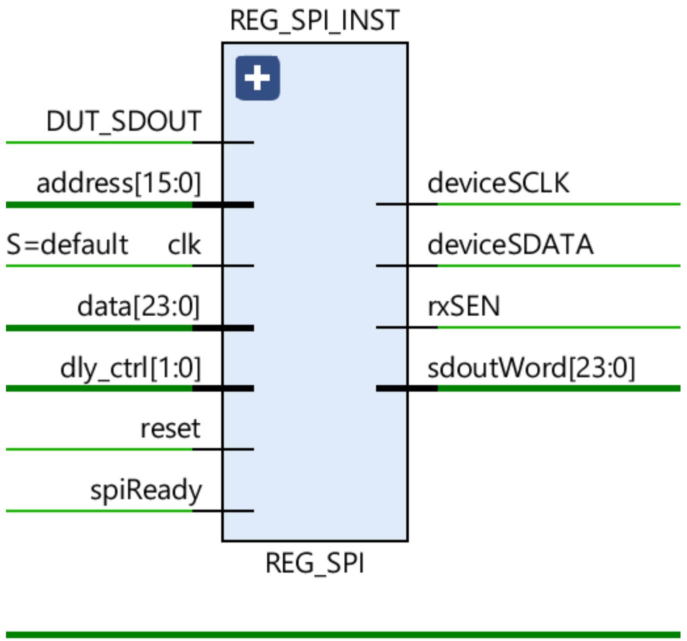




Reg spi SDIN is sampled at posedge of SCLK and SDOUT is stable at posedge

FIFO spi gives data of 24 bit data continuously upto pointer difference

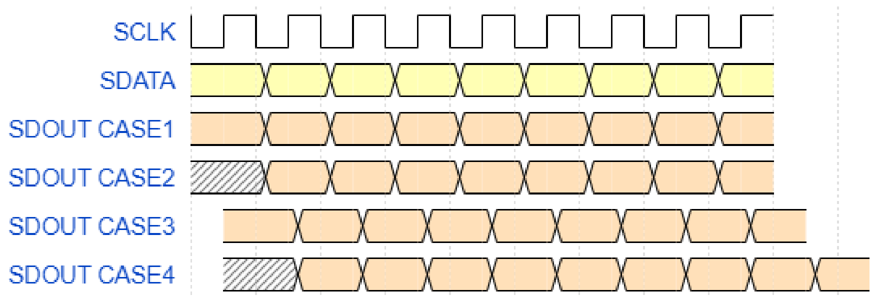
REG SPI BLOCK



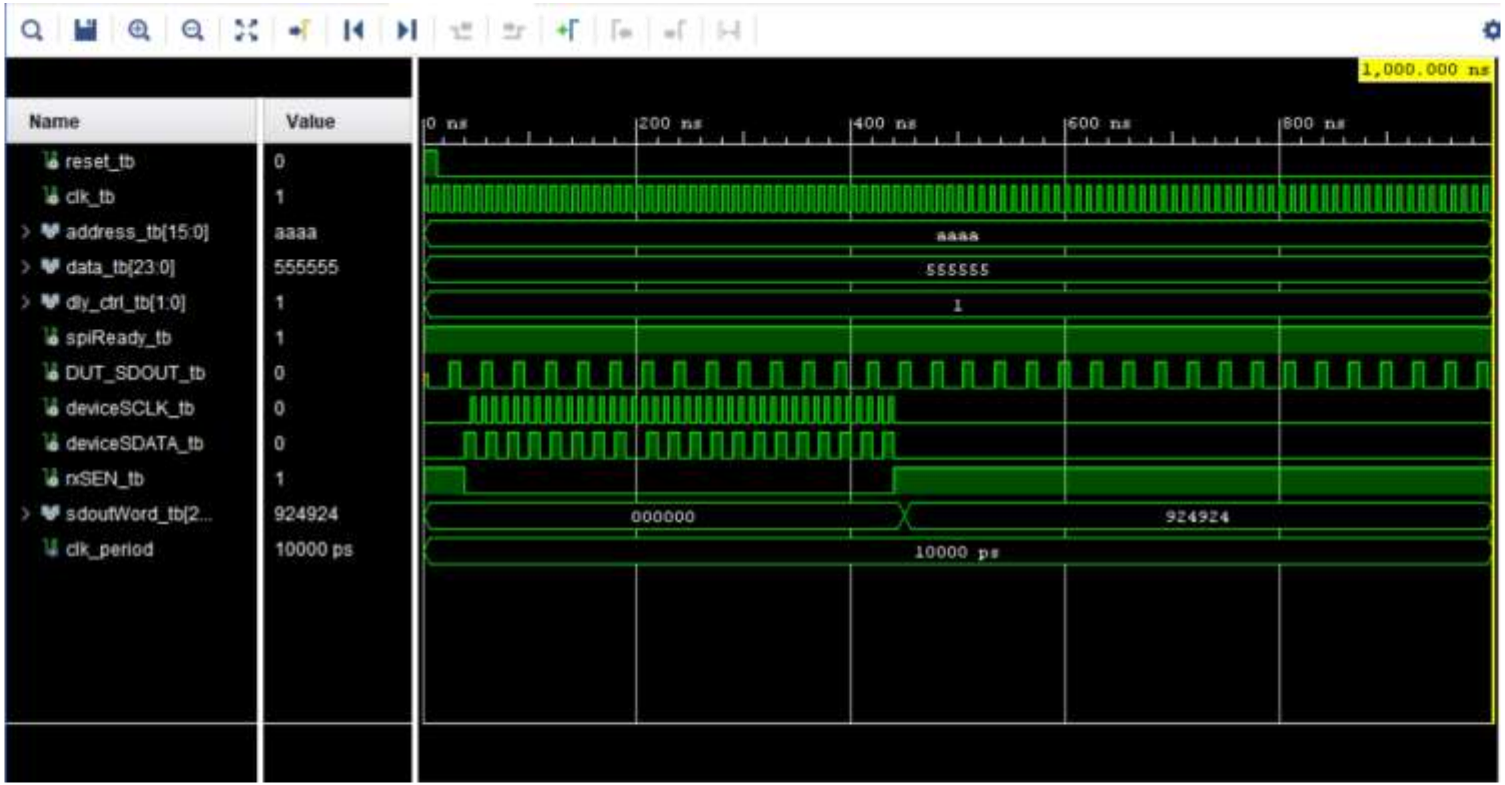
- Normal Register write and Read
- Address 16 bit given by Reg 34[15:0]from Regmap
- Data 24 bit given by Reg 35[23:0] from Regmap
- Delay_ctrl is given by Reg26[1:0]
- SpiReady given by Reg26[2]
- Reset from Reset system
- Clk is clk_SPI(20Mhz clk)
- Device SCLK is not of clk_SPI(20MHz clk)
- rxSEN is chip select
- Write enable is not attached here

Problems with SPI lines with Delay

- We know there may be issues regarding delays from AFE the data may be reached late to the SPI line so we take care of these 4 challenges in the SPI communication. The 4 adjustments are given below

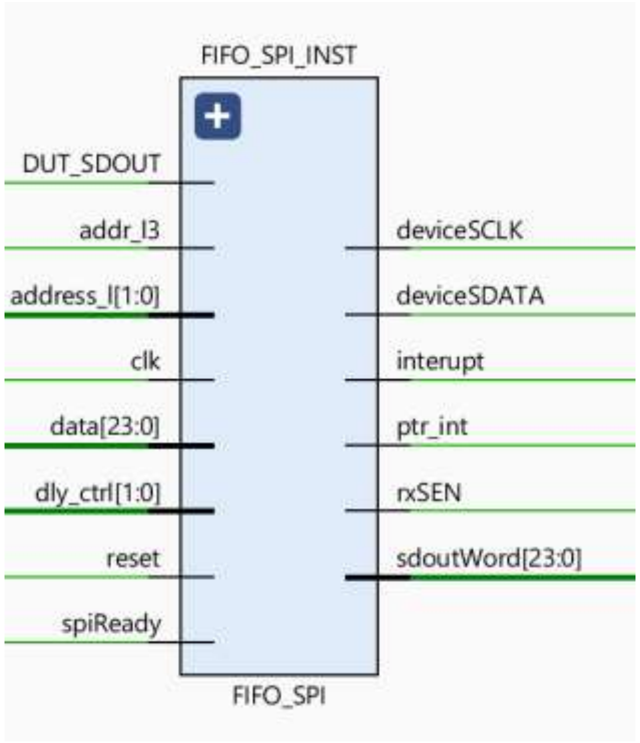


Results for 01 delay control



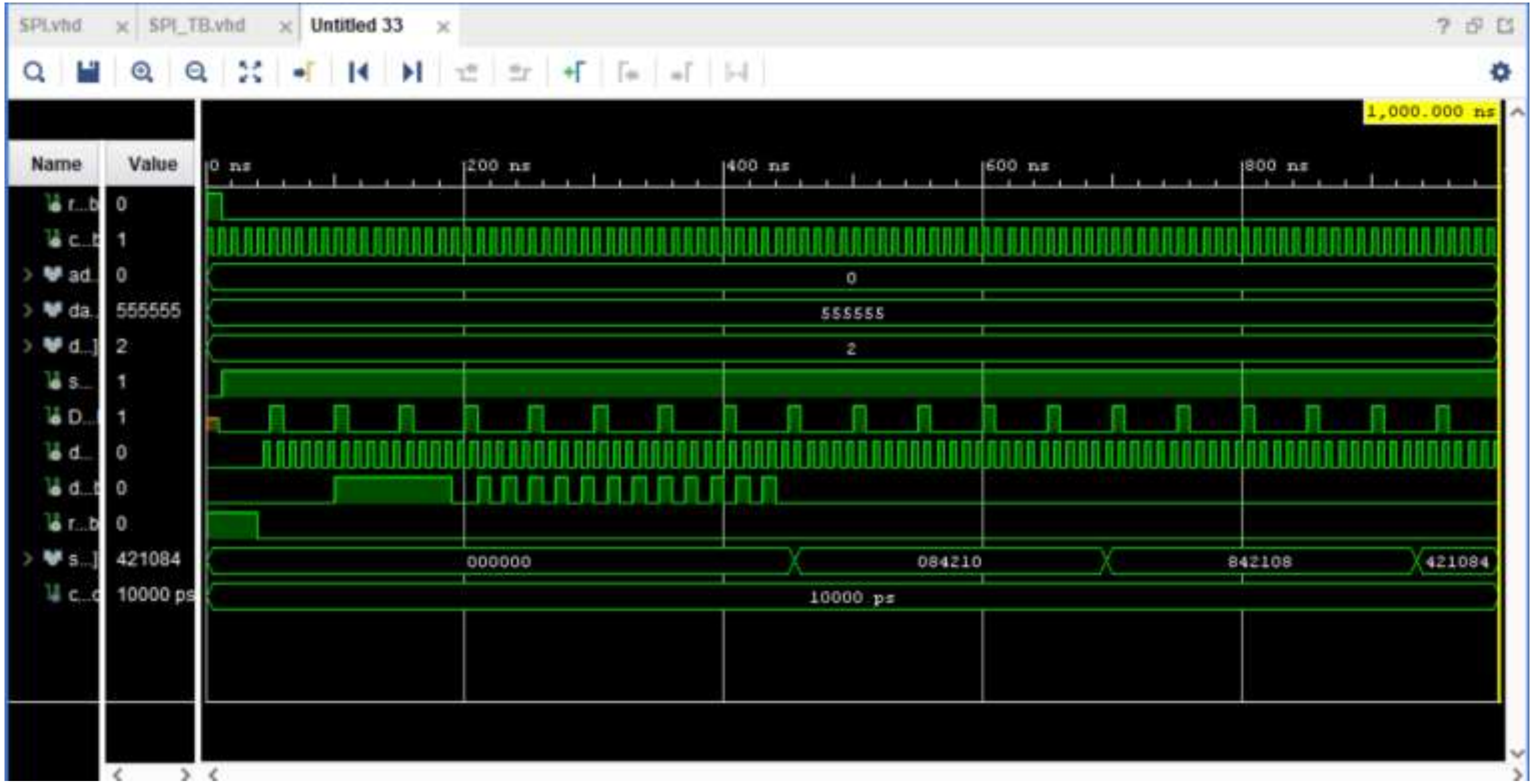
Name	Value
reset_tb	0
clk_tb	1
address_tb[15:0]	aaaa
data_tb[23:0]	555555
dly_ctrl_tb[1:0]	1
spiReady_tb	1
DUT_SDOUT_tb	0
deviceSCLK_tb	0
deviceSDATA_tb	0
rxSEN_tb	1
sdoutWord_tb[2...	924924
clk_period	10000 ps

FIFO SPI BLOCK



- FIFO SPI for Read FIFO Data
- Address 16 bit given by address_l(information about AFE) and addr_l3 (information about Main or Phase FIFO)
- Address_l from Reg26[4:5] and Addr_l3 from Reg26[3]
- Data 24 bit given by Reg 26[31:8] from Regmap (no use)
- Delay_ctrl is given by Reg26[1:0]
- spiReady is given by output pin FIFO_int
- Reset from Reset system
- Clk is clk_SPI(20Mhz clk)
- Device SCLK is not of clk_SPI(20MHz clk)
- rxSEN is chip select
- Ptr_int gives interrupt when sdoutword has pointer difference
- Interupt is up when sdoutWord is Ready to read

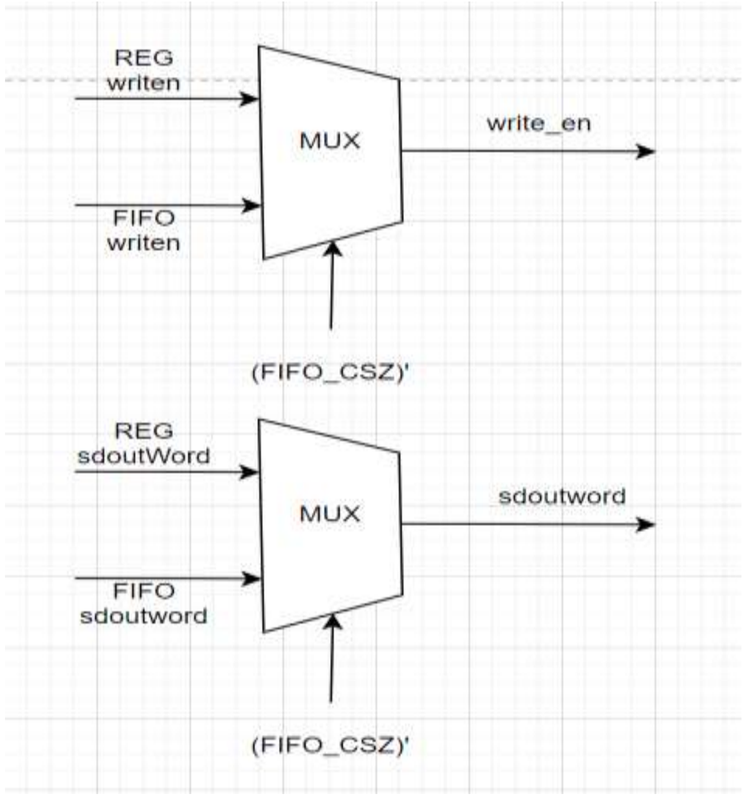
Results for 01 delay control of burst

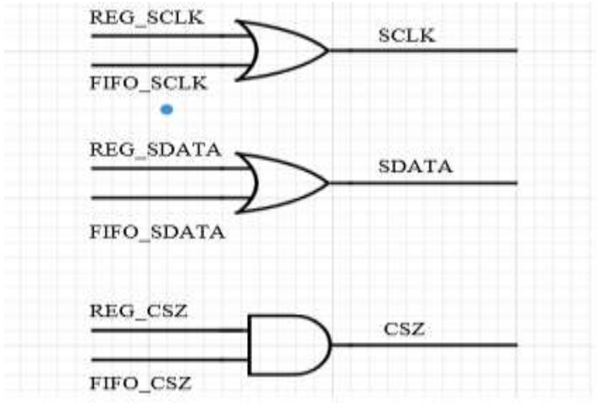


Name	Value
r...b	0
c...b	1
ad...	0
da...	555555
d...]	2
s...	1
D...	1
d...	0
d...b	0
r...b	0
s...]	421084
c...d	10000 ps

```
0
555555
2
000000  084210  842108  421084
10000 ps
```

Adding FIFO and REG block separately





- We Muxed the output of SdoutWord
- We muxed the output of write en
- We or'ed the SCLK
- We and'ed the CSZ
- We or'ed the SDATA

Now implementing SPI on Sparrow and Testing using test pins inside Sparrow

TP_BUS14_2p5_1-P19-SCLK-2

TP_BUS14_2p5_2-R19-SDATA-3

TP_BUS14_2p5_3-T20-SDOUT-4

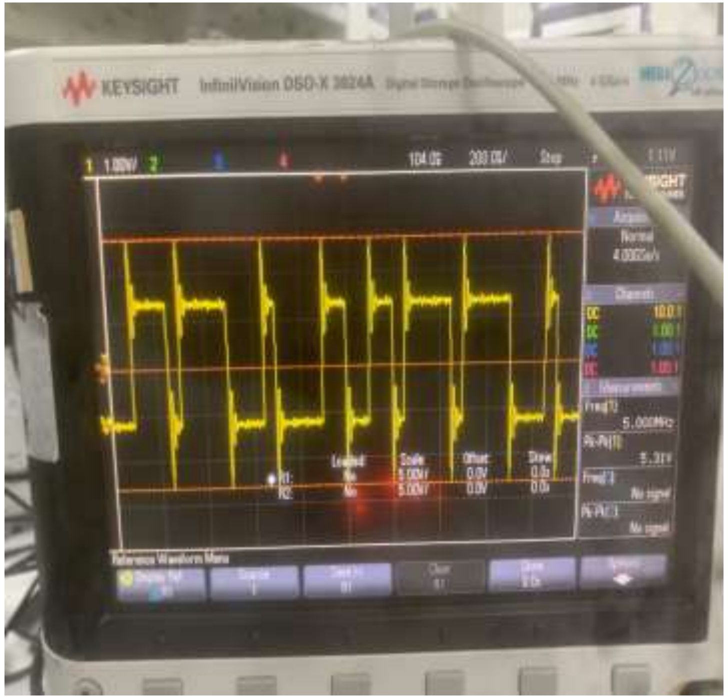
TP_BUS16_2p5_1-C14-CSZ-5

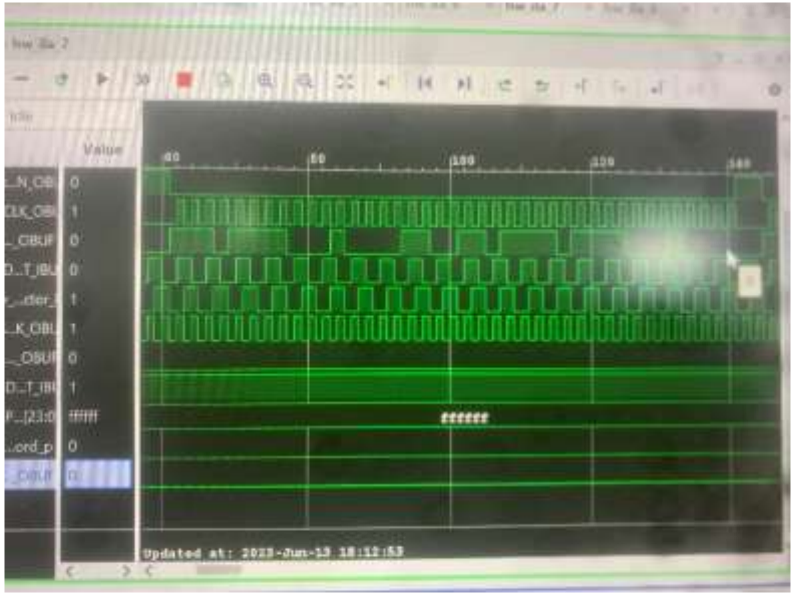
TP_BUS16_2p5_2-C15- FIFO_int-6

TP_BUS16_2p5_3-E13- REG_int-7

- Above shown are test inputs on giving the pulses to the SDOUT and interupt in from signal generator
- Observing the SDATA and SCLK in DSO given below results

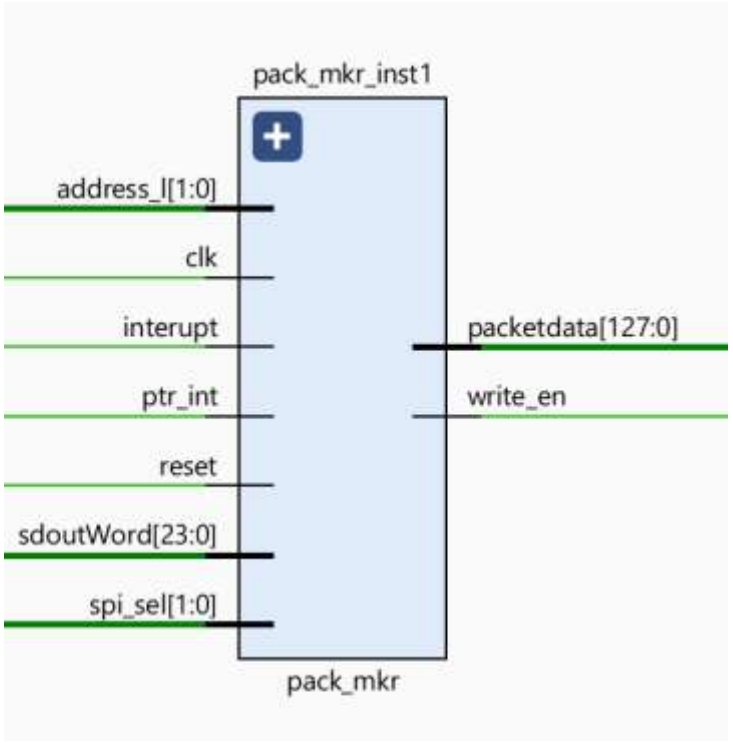
Ressults of SDATA on REG SPI





Pack maker inst

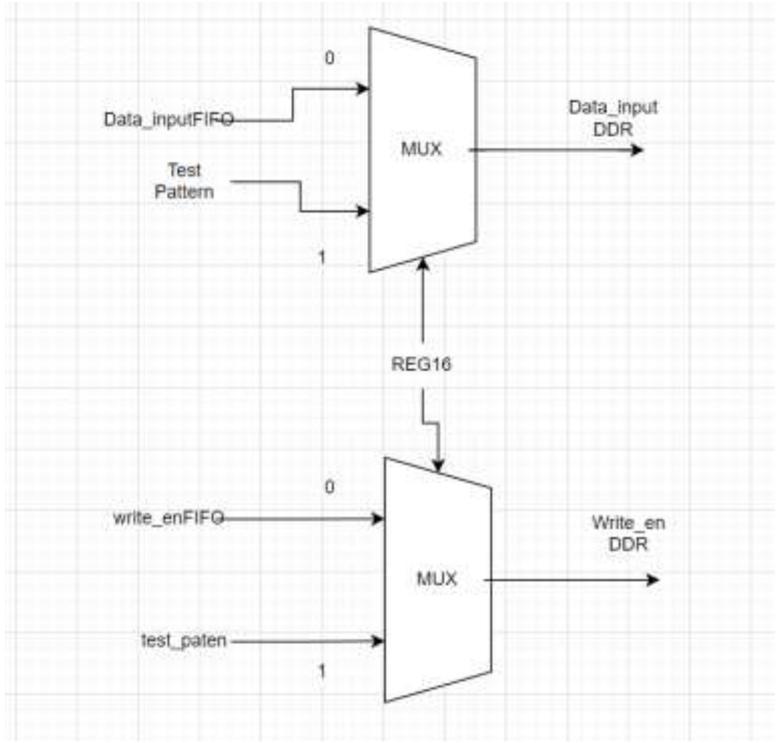
- This block is used for efficiently pad SPI sdoutword to fill 128 bit data which is pre coded inside arevo to write into DDR memory
- Address_l comes from Reg26[5:4]
- Sdoutword,Interupt and ptr_int will be dependent on FIFO SPI outputs for now
- Reset normal internal rst_sys
- SPI_sel is pre coded data depending on which data we are reading will discuss briefly in DATA_SPI block
- Packetdata(8->header,SPIdata)



Pad packet data for FIFO

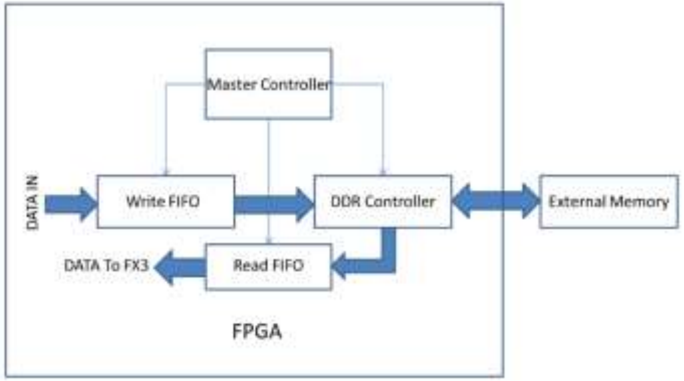
- Instead of padding the data with all zeros and sending it to the DDR memory in the Arevo, we came up with the idea of padding the data from 5 spi with header to make it 128 bit data.
- Our header is constructed so that the first two bits provide information about AFE, the following two bits contain information about the SPI we are using, and the following bit has information about count and pointer difference.
- Next 120 bits will give SPI Data, for this PAD to work we made changes in fifo such that we get pointer interrupts and interrupts.
- Which give info as pad the sdoutword that present at that instant

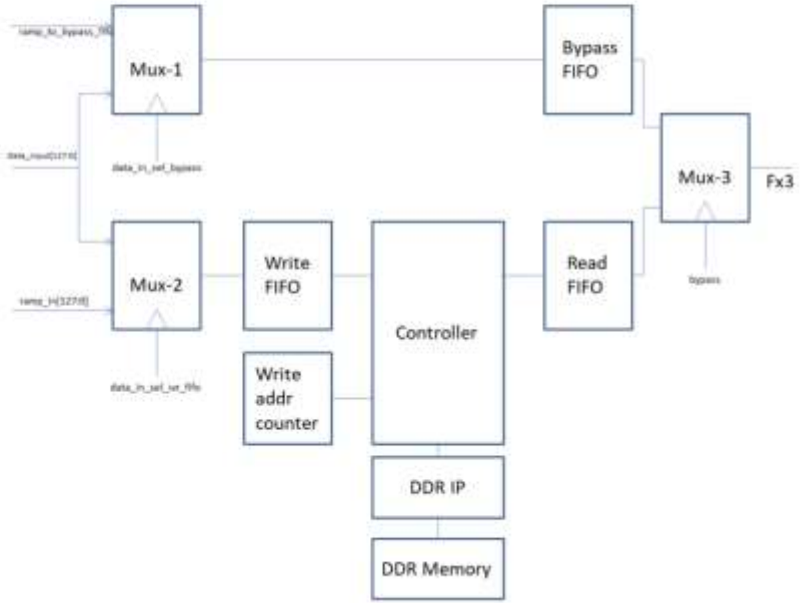
Integrating with sparrow code



- To write into DDR we muxed the Test pattern with our data input FIFO
- Data input fifo is the data coming out from packet maker because of only FIFO
- If Reg16 is 0 then the FIFO data is inside the DDR block input
- If Reg 16 is 1 Then the inbuilt test pattern which is ramp up is initiated and written in DDr top

DDR TOP



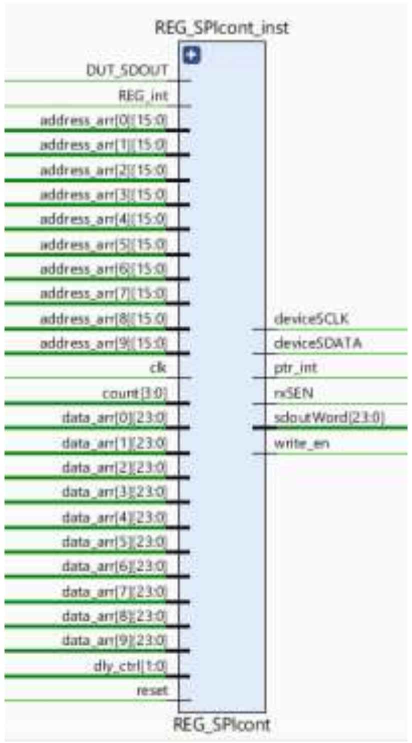


- The Arevo uses above DDR top module to Write into DDR and Read into DDR all the signals are given from Asterix to control the writing and Reading the DDR depth captures

Asterix TEST results with resolving previous error

As the data_out is updated at clk_sys every time we missded the first 16 bit data initially, so I added a delay of clk_sys so now we are able to capture the initial data also.

DA	TA	RE	AD	TO	AS	TE	Rix		He	xD	AT	A				
51200.00	5.00	21845.00	21845.00	21845.00	21845.00	21845.00	21845.00		C800	0005	5555	5555	5555	5555	5555	5555
49152.00	0.00	0.00	0.00	0.00	0.00	85.00	21845.00		C000	0000	0000	0000	0000	0000	0055	5555
51200.00	5.00	21845.00	21845.00	21845.00	21845.00	21845.00	21845.00		C800	0005	5555	5555	5555	5555	5555	5555
49152.00	0.00	0.00	0.00	0.00	0.00	85.00	21845.00		C000	0000	0000	0000	0000	0000	0055	5555



- Works on posedge of REG_int once started it won't search for REG_in
- 10 address array and 10 data arrar are benn used
- Delay_ctrl is given by Reg26[1:0]
- spiReady is given by output pin FIFO_int
- Reset from Reset system
- Clk is clk_SPI(20Mhz clk)
- Device SCLK is not of clk_SPI(20MHz clk)
- rxSEN is chip select
- Ptr_int gives interrupt when sdoutword has count of REG SPI difference
- Write_en is up when sdoutWord is Ready to read

Continuous REG read block

- As for the request we created a continuous register read block that will take address array from registers and data from registers
- It will start reading the data from that addresses with provided count times and update sdoutword with an interrupt to let read the data.
- Continuous register block uses our previous reg block and update the addresses and sdout came from them depending on the logic we wrote

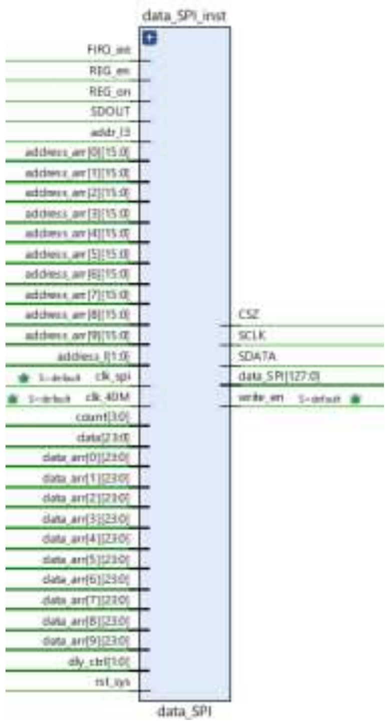
```
•assign   address_arr[0] = Reg29[15:0];
•assign   address_arr[1] = Reg29[31:16];
•assign   address_arr[2] = Reg30[15:0];
•assign   address_arr[3] = Reg30[31:16];
•assign   address_arr[4] = Reg31[15:0];
•assign   address_arr[5] = Reg31[31:16];
•assign   address_arr[6] = Reg32[15:0];
•assign   address_arr[7] = Reg32[31:16];
•assign   address_arr[8] = Reg33[15:0];
•assign   address_arr[9] = Reg33[31:16];
•assign   data_arr[0] = Reg12[23:0];
•assign   data_arr[1] = Reg14[23:0];
•assign   data_arr[2] = Reg15[23:0];
•assign   data_arr[3] = Reg17[23:0];
•assign   data_arr[4] = Reg18[23:0];
•assign   data_arr[5] = Reg19[23:0];
•assign   data_arr[6] = Reg20[23:0];
•assign   data_arr[7] = Reg21[23:0];
•assign   data_arr[8] = Reg22[23:0];
•assign   data_arr[9] = Reg23[23:0];
```

Merging all SPI and pad packet data

- We wanted to merge all the SPI and pad packet data and make a data_SPI block which provides us with the FIFO data REG data conditionally.
- If we want to read Reg data then we need to keep REG_en as high depending on the count it will give out the interrupts and the pttr interrupts and sdoutword.
- So as in case by muxing all the data and assigning it to packet data provides us the merging packets of all data with relevant headers so that the computer will understand which data is which.
- Below is the state diagram for the data SPI block

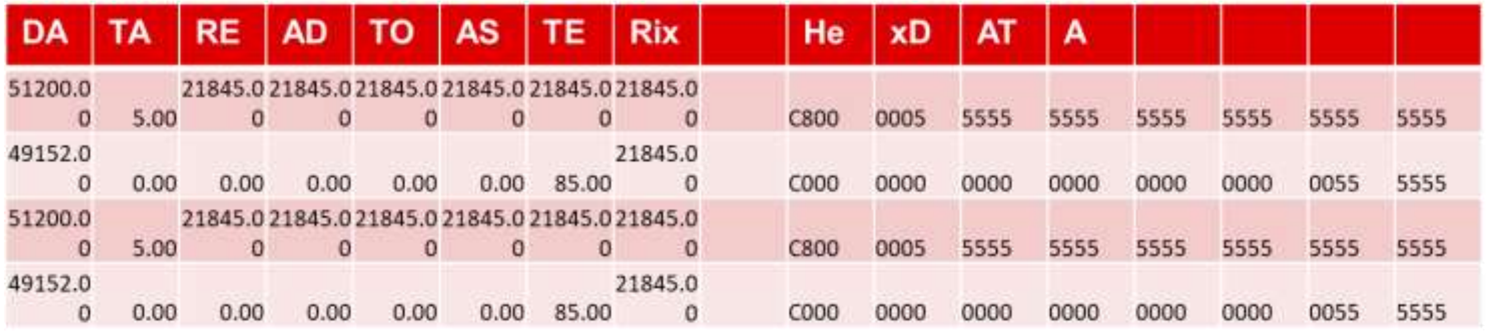
DAT SPI BLOCK

- Same registers for fifo and others
- Extra we added are REG_en and REG_on
- Reg_en is 1 for if you want to read or write data to registers continuously after FIFO is done
- Reg_on is only if you want to write or read the continuous Register data not including FIFO



Asterix Test Results

FIFO DATA

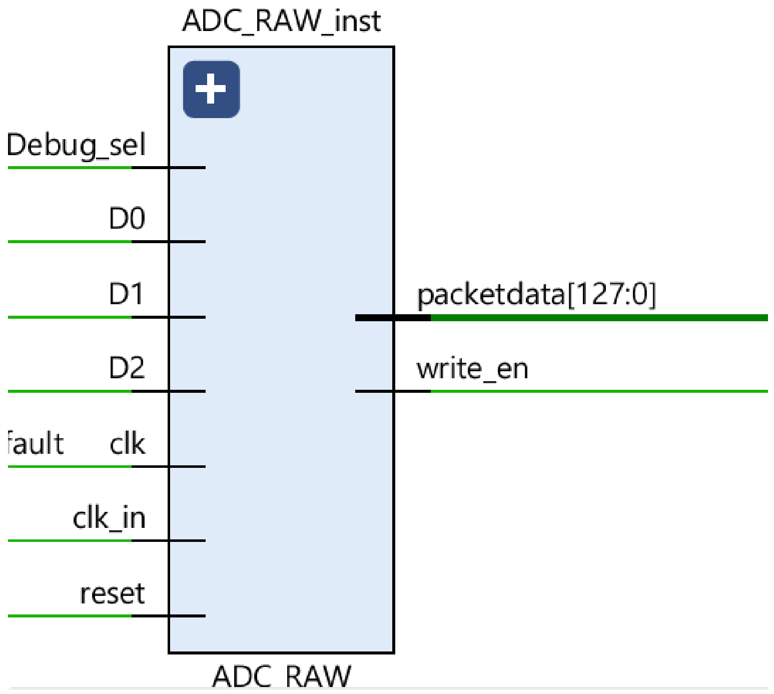


DA	TA	RE	AD	TO	AS	TE	Rix		He	xD	AT	A				
51200.00	5.00	21845.00	21845.00	21845.00	21845.00	21845.00	21845.00		C800	0005	5555	5555	5555	5555	5555	5555
49152.00	0.00	0.00	0.00	0.00	0.00	85.00	21845.00		C000	0000	0000	0000	0000	0000	0055	5555
51200.00	5.00	21845.00	21845.00	21845.00	21845.00	21845.00	21845.00		C800	0005	5555	5555	5555	5555	5555	5555
49152.00	0.00	0.00	0.00	0.00	0.00	85.00	21845.00		C000	0000	0000	0000	0000	0000	0055	5555

REG DATA

2048	5	65535	65535	65535	65535	65535	65535	FIFO-AFE00	800	5	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
0	0	0	0	0	0	255	65535	FIFO-AFE00	0	0	0	0	0	0	FF	FFFF
14336	5	65535	65535	65535	65535	65535	65535	REG	3800	5	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF
12288	0	0	0	0	0	255	65535	REG	3000	0	0	0	0	0	FF	FFFF

ADC RAW MODE BLOCK



- ADC RAW mode work on Debug SEL
- When debug sel is high ADC will start padding the data when it fills all the 128 bits it is updated and given write_en
- D0,D1,D2, clk_in are coming from GPIO pins of AFE
- Clk which we are using here is 40MHz clock

ADC raw Mode

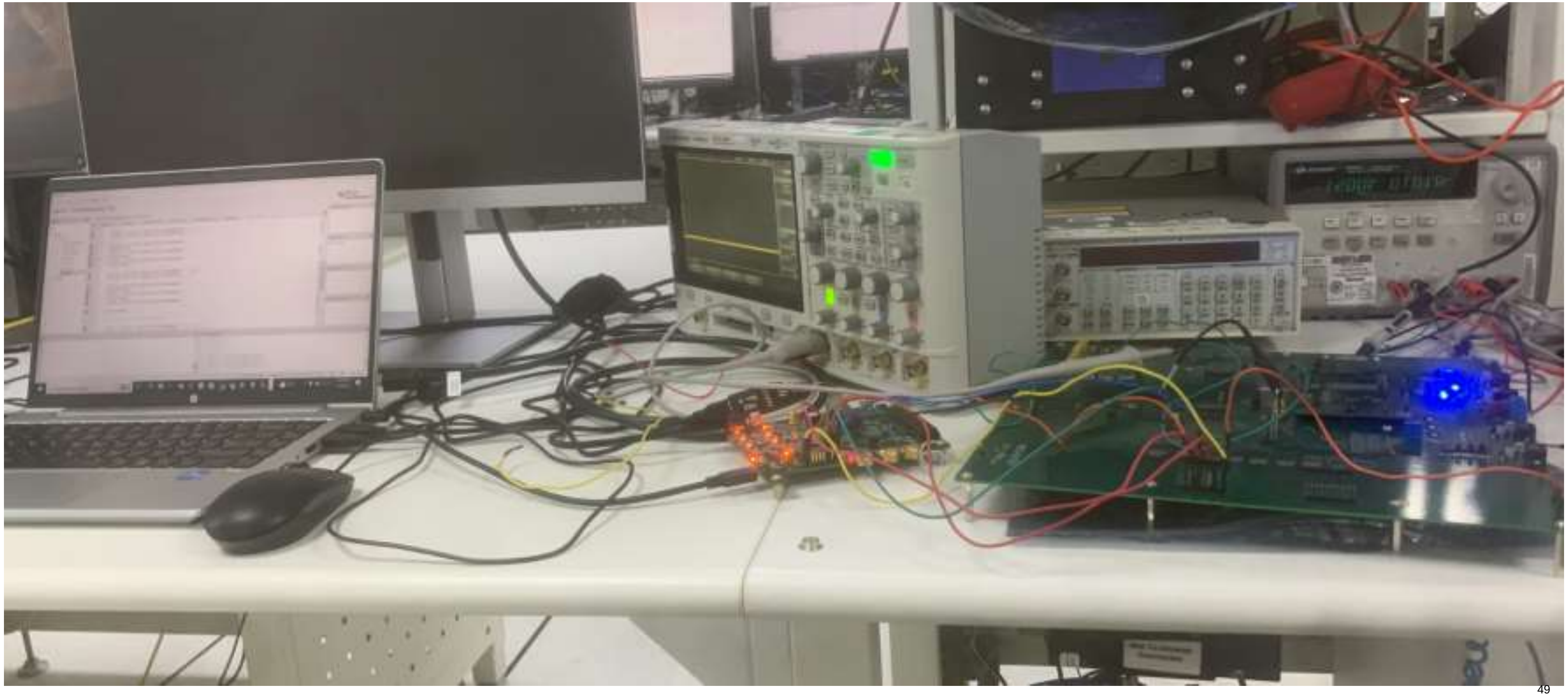
- AFE gives data D0 D1 and D2 with 2MHz clock from the AFE.
- We need to write this data to the 128 bit packet by same process of padding.
- We padded them as D2,D1,D0 in order and pad all this data and there is a header to this raw mode data.
- We made a separate block for RAW mode data packet block after filling the 128 bit data then there will be interrupt for reading this packet data

Asterix Test Results

Normal test data generated from AWG

4.37E+04	5.62E+04	4.68E+04	2.81E+04	5.62E+04	4.68E+04	2.81E+04	5.62E+04		AAB6	DB6D	B6DB	6DB6	DB6D	B6DB	6DB6	DB6D
4.37E+04	5.62E+04	4.68E+04	2.81E+04	5.62E+04	4.68E+04	2.81E+04	5.62E+04		AAB6	DB6D	B6DB	6DB6	DB6D	B6DB	6DB6	DB6D
4.37E+04	5.62E+04	4.68E+04	2.81E+04	5.62E+04	4.68E+04	2.81E+04	5.62E+04		AAB6	DB6D	B6DB	6DB6	DB6D	B6DB	6DB6	DB6D
4.37E+04	5.62E+04	4.68E+04	2.81E+04	5.62E+04	4.68E+04	2.81E+04	5.62E+04		AAB6	DB6D	B6DB	6DB6	DB6D	B6DB	6DB6	DB6D

Integrating and testing with AFE



Testing Normal Register write and Read

- We wrote to some of the registers and read the data at the same time and verified that it is working

Testing continuous Reg Write and Read to Asterix

14336	8	204	52224	43690	218	48128	52651		3800	8	00CC	CC00	AAAA	00DA	BC00	CDAB
12288	0	219	51712	52666	188	55808	43981		3000	0	00DB	CA00	CDBA	00BC	DA00	ABCD
14336	8	204	52224	43690	218	48128	52651		3800	8	00CC	CC00	AAAA	00DA	BC00	CDAB
12288	0	219	51712	52666	188	55808	43981		3000	0	00DB	CA00	CDBA	00BC	DA00	ABCD
14336	8	204	52224	43690	218	48128	52651		3800	8	00CC	CC00	AAAA	00DA	BC00	CDAB
12288	0	219	51712	52666	188	55808	43981		3000	0	00DB	CA00	CDBA	00BC	DA00	ABCD
14336	8	204	52224	43690	218	48128	52651		3800	8	00CC	CC00	AAAA	00DA	BC00	CDAB
12288	0	219	51712	52666	188	55808	43981		3000	0	00DB	CA00	CDBA	00BC	DA00	ABCD

Testing ADC data to Asterix

43747	36359	58254	2019	36359	58254	14367	36408		AAE3	8E07	E38E	07E3	8E07	E38E	381F	8E38
43744	32312	58254	14367	36408	58241	63715	36408		AAE0	7E38	E38E	381F	8E38	E381	F8E3	8E38
43747	36408	1009	50972	29127	7281	50972	4039		AAE3	8E38	03F1	C71C	71C7	1C71	C71C	0FC7
43744	28735	8065	49404	32263	1009	63516	28728		AAE0	703F	1F81	C0FC	7E07	03F1	F81C	7038
43747	36408	58254	16131	36408	58254	14563	61496		AAE3	8E38	E38E	3F03	8E38	E38E	38E3	F038
43747	36408	64526	14563	36800	58353	63516	29176		AAE3	8E38	FC0E	38E3	8FC0	E3F1	F81C	71F8
43548	29127	57457	51168	29127	57457	51168	29176		AA1C	71C7	E071	C7E0	71C7	E071	C7E0	71F8
43548	29176	7294	2016	29127	58241	50972	32312		AA1C	71F8	1C7E	07E0	71C7	E381	C71C	7E38

Future work

- N

Thank you