MULTICHANNEL ADC IP CORE ON XILINX SOC FPGA

A.Suresh, S.Shyama, Sangeeta Srivastava and Nihar Ranjan

Embedded Systems, Product Development and Innovation Center (PDIC), BEL, India

ABSTRACT

Sensing of analogue signals such as voltage, temperature, pressure, current etc. is required to acquire the real time analog signals in the form digital streams. Most of the static analog signals are converted into voltage using sensors, transducers etc. and then measured using ADCs. The digitized samples from ADC are collected either through serial or parallel interface and processed by the programmable chips such as processors, controllers, FPGAs, SOCs etc. In some cases, Multichannel supported ADCs are used to save the layout area when the functionalities are to be realized in a small form factor. In such scenarios, parallel interface for each channel is not a preferred interface considering the more number of interfaces / traces between the components. Hence, Custom, Sink synchronized, Configurable multichannel ADC soft IP core has been developed using VHDL coding to interwork with multichannel supported, time division multiplexed ADCs with serial interface. The developed IP core can be used either as it is with the SPI interface as specified in this paper or with necessary modifications / configurations. The configurations can be the number of channels, sample size, sampling frequency, data transfer clock, type of synchronization – source / sink, control signals and the sequence of the operations performed to configure ADC. The efficiency of implementation is validated using the measurements of throughput, and accuracy for the required range of input with acceptable tolerances. ZYNQ FPGA and LTC2358 ADC are used to evaluate the developed IP core. Integrated Logic Analyser (ILA) which is an integrated verification tool of Vivado is used for Verification.

KEYWORDS

Configurable ADC soft IP Core, Sensor, Multichannel ADC, Sink Synchronization, FPGA, VHDL [3].

1. INTRODUCTION

The latest digital technology revolution and the rapid growth of semiconductor technology in the fields of FPGA, SOC [2][5][6], RFSOC [5], etc. and programming flexibility lead to choosing either SOC or RFSOC based applications. ASIC based solution is expensive for very high throughput [3]. In general ASIC solution is not cost effective unless there is no mass manufacturing. Programmable chip based platform demands the engineer to develop VHDL, Verilog, System C based IP cores when there are no readily available IP cores from the vendor for the particular interface or for the application. Since the FPGA has further grown into SOC and RFSOC with the integration of inbuilt multicore processor, controller such as ARM cores, Application Processing Unit (APU), real time Processing Unit (RPU) etc., these platform demand further to have the software development skill based on C, C++, Embedded C either with Operating system such as LINUX, VxWORKS, or customized OS such as PetaLINUX or without any OS – in bare metal mode. Programmable chip has to incorporate with indigenously

David C. Wyld et al. (Eds): NLP, MLTEC, CLBD, SEAPP, NeTIoT, VLSIE, ITCS, ARIA, DMS - 2021
pp. 341-353, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.112326

developed IP core or vendor specific, technology dependent, readily available free IP core or purchasable third party IP cores.

The developed architecture and the platform uses such SOC FPGA with inbuilt dual ARM core, logic resources such as LEs, memory bits, etc. The developed reconfigurable soft IP core and the approach can be used to develop such custom IP cores for any FPGA and for any type of ADC except the ADC which is hard IP core of RFSOC. The developed IP core also is configurable to change the clock mode, sampling frequency, no of channels etc. The soft ADC IP core and inbuilt ADC hard IP core which is available in RFSOC FPGAs have the limitation in supporting input voltage range for higher voltages, generally beyond 1V. In such scenarios, ADC IC has to be used.

As per the requirement, the input voltage range from -10.24 volts to +10.24 volts is to be measured. In addition to this requirement, considering the number of channels, sample size, accuracy, number of pins available at FPGA, sampling frequency, acceptable propagation and processing delay, complexity of the configuration, obsolescence of the ADC IC etc; ADC LTC2358 [1] was chosen. These aforementioned parameters are the motivation to choose a specific IC and serial interface with minimal interconnections for high speed at Mbps, advantage of sink sync scheme over source sync scheme are the motivation to the associated algorithm. But the reusability of the configurable multichannel ADC soft IP core across the different technology based FPGAs has been considered and behavioural VHDL has been written for the core ADC IP core. More details are provided in Section II.

In practical scenarios, the input voltage level range may vary and expected accuracy also may vary. Hence, it would be a desired feature / requirement to configure the sensing input voltage range and other parameters either through GUI (Graphical User Interface) / CUI (Command User Interface) or to set from the possible options / settings as a part of the factory settings of the firmware. Provisions are provided to select the parameters through CUI after the POST is successful. Even in this implementation, the IP core is set with default configuration and further can be changed through CUI as and when the requirement changes. LTC2358 has an option to set the voltage ranges from +/-10.24V, 0V to 10.24V, +/-5.12V, or 0V to 5.12V. Correspondingly, the achievable accuracy in terms of error is 0.12 mV theoretically and practically 11.37 mV.

Sink synchronization is preferred in few scenarios where the whole processes have to be operated with reference to a single master clock for better synchronization, to reduce the number of interconnections, and to avoid the multiple clock domain, to avoid glitches, to avoid or to reduce the phase error and to mitigate the effect of the accumulated jitter. Otherwise, additional techniques such as buffering or one clock delay in the case of source synchronous mode are required. Desired Serial interface and sink synchronization are exploited as additional advantages with the configurable and reconfigurable custom multichannel ADC soft IP core. At the same time, JEDEC JESD204 is recommended for high speed ADC interfaces at Gbps [9]. The developed IP core's main functionality is verified by sensing the static analog signals, and the processing of dynamic ADC samples is not the scope of the work. That's why the performance measurement such as SINAD measurement, SFDR, etc are not required and not done. Integrated Logic Analyser (ILA) [7] which is an integrated verification tool of Vivado is used for Verification. No third party tool is required, whereas [2] uses Synopsis Discovery AMS platform. The main contributions are the realization of Sink synchronized, Configurable multichannel ADC soft IP core in PL section, the driver, API, and the embedded software developed for the inbuilt ARM core and the developed CUI for the configuration of ADC IP core and for the verification of results. CUI is operated through debug port based on UART interface. The implemented ADC IP core has a configurable FIFO size which can be configured only in the

firmware. This FIFO size can be varied based on the application requirement. Depth of FIFO or storage space is configurable from 8 samples to 1K samples. The same space is required in ARM processor also. The achieved processing speed of the algorithm supports 200 Ksps / Channel with the acquisition time of 570ns.

In the present paper, concept and flow are explained through Section II to Section VI. Section II covers the relevant portion of the hardware platform architecture, data flow between the components, and brief summary about the relevant components used in the hardware. The software development approach, associated tools, and software mapping into the hardware are explained in Section III. Section IV depicts the data transfer protocol, flow chart, pseudo random code / portion of the code, and brief explanation. Section V demonstrates the simulation and test results of the implementation to verify the functional requirements and the measurement of the main parameters such as throughput, and accuracy. Section VI concludes with the novel claims, developments and chances for further proliferation.

2. HARDWARE PLATFORM ARCHITECTURE, AND DATAFLOW

IP core for Max104 ADC is developed on Virtex2Pro in [3]. But, here, the hardware consists of ZYNQ [4] XC7Z045FFG900-0I as the core. The hardware platform architecture (HPA) with the functional off chip peripherals is shown in Figure 1. The analogue signals to the hardware is coming through a 3U VPX board edge connector which is filtered and given to the ADC LTC2358. The digitized output from ADC is then received into the PL section of FPGA and received packetized / interleaved / multiplexed 6 channels data is unwrapped into 6 channels at PL, and then serial data is sent to the PS section using AXI interface where it is further processed and converted into digital voltage and displayed on the terminal. If the data to be received into the ARM is not required to be sent to any other chip, Board or system, channel identifier is not to be sent from PL to PS and they can be discarded at PL. If it is to be sent to other chip, board, or system, then channel identifiers also are to be received into ARM processor. GPIO address identifier is used for all the peripherals realised in PL section and connected to ARM core through AXI – On chip peripheral bus to transfer the data through AXI bus. But, those address identifiers are not the better choice to identify the channel, since the size of the address identifier is greater than or equal to 32 bits, because, generally an address range is reserved for each AXI peripheral. The complete dataflow is shown in figure 2.

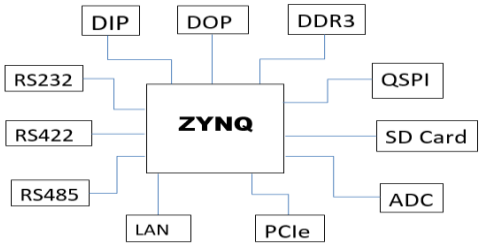


Figure 1. Hardware platform architecture

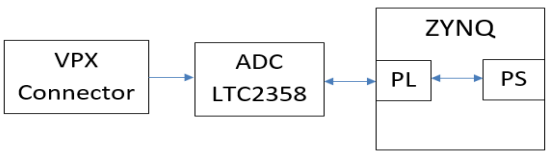


Figure 2. Dataflow

LTC2358 from Analogue Devices is the ADC chosen. It is a 16 bit SAR ADC for which throughput of 200ksps can be achieved per channel. It has eight buffered differential input channels, of which six have been used but can be upgraded for all 8 channels. The board space is considerably decreased by accommodating 8 channels into one ADC. The input ranges supported are +/-10.24V, 0V to 10.24V, +/-5.12V, 0V to 5.12V, +/-6.25V, and 0V to 6.25V. It operates from a 5V low voltage supply. LTC2358 supports both LVDS and CMOS modes of operation. CMOS mode is chosen since it gives the output of 6 channels in 6 different pins whereas in LVDS mode all the 6 channel data is multiplexed and given in the same set of differential pins and hence the speed is considerably reduced. Also the power dissipation of CMOS mode is 259mW which is less compared to 287mW in LVDS mode. Hence CMOS mode of operation allows the user to optimize bus width, throughput and power.

ZYNQ 7000 SOC [2],[4],[5],[6] is chosen to achieve the complex algorithms in the design. It integrates a feature-rich dual-core ARM® Cortex™-A9 based processing system (PS) and 28 nm Xilinx programmable logic (PL) in a single device.

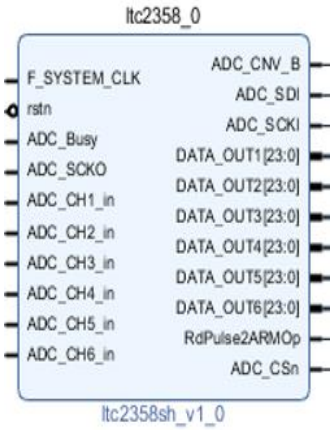


Figure 3. Custom ADC IP core

3. SOFTWARE DEVELOPMENT APPROACH, ASSOCIATED TOOLS, AND SOFTWARE MAPPING INTO THE HARDWARE

The software design to receive the digitized output from ADC and to packetize it to transfer to PS section was done in VHDL logic using VIVADO 2019.2 tool from XILINX. After receiving the data into the PS section it is processed and verified in SDK [6] 2019.2 provided by XILINX. Petalinux OS version 2019.2 is used for the development of embedded software and the CUI. In the VIVADO [6] tool, VHDL is written in the VHDL editor code to receive input from the ADCs. Later this VHDL logic was converted to a custom IP core as shown in figure 3 and invoked to the top level block diagram. The digitized packetized data output from the ADC custom IP core is given to the AXI GPIO IP core [2]. For each channel of ADC, one AXI GPIO IP core is used through which an address identifier is mapped to each AXI peripheral.

Diagram | Address Editor

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_gpio_0	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
axi_gpio_1	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF
axi_gpio_2	S_AXI	Reg	0x4122_0000	64K	0x4122_FFFF
axi_gpio_3	S_AXI	Reg	0x4123_0000	64K	0x4123_FFFF
axi_gpio_4	S_AXI	Reg	0x4124_0000	64K	0x4124_FFFF
axi_gpio_5	S_AXI	Reg	0x4125_0000	64K	0x4125_FFFF
axi_gpio_6	S_AXI	Reg	0x4126_0000	64K	0x4126_FFFF

Figure 4. Address mapping from PL to PS

Tcl Console | Messages | Log | Reports | Design Runs | Find Results | Debug

Name	Direction	Interface	Neg Diff ...	Package...	Fi...	Ba...	I/O Std
ADC_Busy	IN			Y23	✓	11	LVCMOS33*
ADC_CH1_in	IN			AA24	✓	11	LVCMOS33*
ADC_CH2_in	IN			AB24	✓	11	LVCMOS33*
ADC_CH3_in	IN			AA22	✓	11	LVCMOS33*
ADC_CH4_in	IN			AA23	✓	11	LVCMOS33*
ADC_CH5_in	IN			AF15	✓	10	LVCMOS33*
ADC_CH6_in	IN			AG15	✓	10	LVCMOS33*
ADC_CNV_B	OUT			Y22	✓	11	LVCMOS33*
ADC_CSn	OUT			Y21	✓	11	LVCMOS33*
ADC_SCKI	OUT			AE16	✓	10	LVCMOS33*
ADC_SCKO	IN			AF17	✓	10	LVCMOS33*
ADC_SDI	OUT			AE15	✓	10	LVCMOS33*
DDR_addr[0]	INOUT	DDR_45990		L25	✓	502	SSTL135*
DDR_addr[1]	INOUT	DDR_45990		K26	✓	502	SSTL135*

Figure 5. I/0 ports pin assignment

Each AXI GPIO IP core is mapped to the PS using a unique base address as shown in figure 4. Then the ADC data is passed on to the PS through AXI interconnect block from AXI GPIO IP core. After completing the block diagram in VIVADO, HDL wrapper is generated for the design. Then the design is synthesized and pin assignments are done in I/O port assignments as shown in figure 5. Then the design is implemented and the bit stream is generated. The bit stream is then exported to the SDK tool where the BSP to test the interface is developed. Then the bit stream is flashed into the PL through JTAG emulator and the BSP is run on PS using the debug port and viewed in SDK terminal.

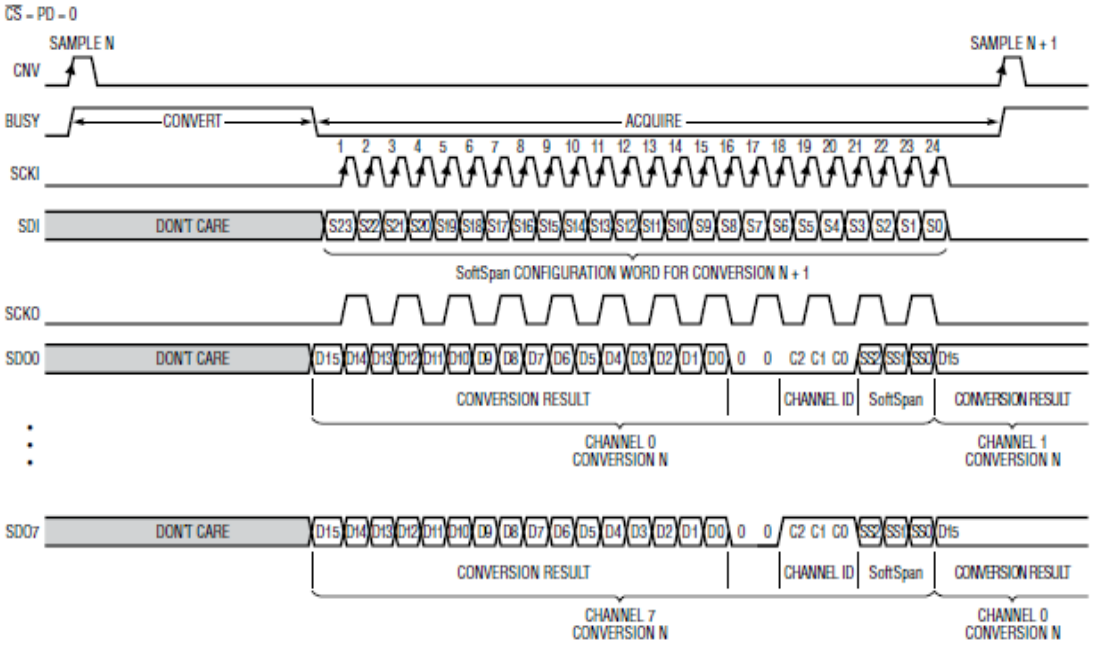


Figure 6: Protocol / Timing diagram [1]

4. DATA TRANSFER PROTOCOL, FLOW CHART, PSEUDO RANDOM CODE AND BRIEF EXPLANATION

The ADC custom IP core is designed based on standard SPI CMOS interface in CMOS I/O mode. SPI interface generally consists of slave select (SS), Master Output / Slave Input (MO/SI), Master Input / Slave Output (MI/SO), and Serial Clock (SCK). Respectively, the following signals / Pins are utilised as follows : (i) ADC chip select signal -"ADC_CSn" as $\overline{CS}$ / (SS), (ii) SoC's output /ADC's input serial data to configure ADC -"ADC_SDI" as SDI (MO/SI), (iii) SoC's input /ADC's output serial data for 8 channels – "DATA_OUT1" / "ADC_CH1_in" to "DATA_OUT6" / "ADC_CH6_in" as SDO0 to SDO7 (MI/SO), and (iv) Serial clock from Master (SoC) – "ADC_SCKI" as SCKI (SCK).

First the CHIP SELECT pin - "ADC_CSn" is made low (Active Low Reset) from the FPGA to enable ADC. This active low reset signal is generated only once and then maintained or kept low throughout the ADC operation. Protocol does not demand any resynchronization between ADC chip and FPGA. The developed IP core also is working consistently.

Then ADC convertor pulse "ADC_CNV_B" is made high for 20ns. The same pulse "ADC_CNV_B" is repeated once for every 24 bits, since all the 8 channels send the digitized data and the channel related data sequentially through its own data pins (SDO0 to SDO7) to FPGA. During the high of "ADC_CNV_B" pulse, ADC samples the analog input for the enabled channels. The channels are configured with the range of input voltage as mentioned in Table 1.

Next during high value of "BUSY" signal, digitized 16 bit data is packetized with 3 bits of channel id, 3 bits of soft span code as given in table 1, and with 2 bits of zeros as per the protocol shown in figure 6. Binary soft span code [2:0] generated by the SDO lines are configured by the FPGA for the ADC configuration and it can be updated on the fly if instructed through CUI for various voltage levels. This information comes along with the sampled data on the ADC_CHx_IN (SDO) lines.

For example, if the digitized 16 bit sample data for a particular analog input is “0000 0101 0111 1100”, then the packetized 24 bit word as per protocol is “0000010101111100 00 000 111” for the enabled channel “000” and for the voltage to be measured “+/- 10.24V”.

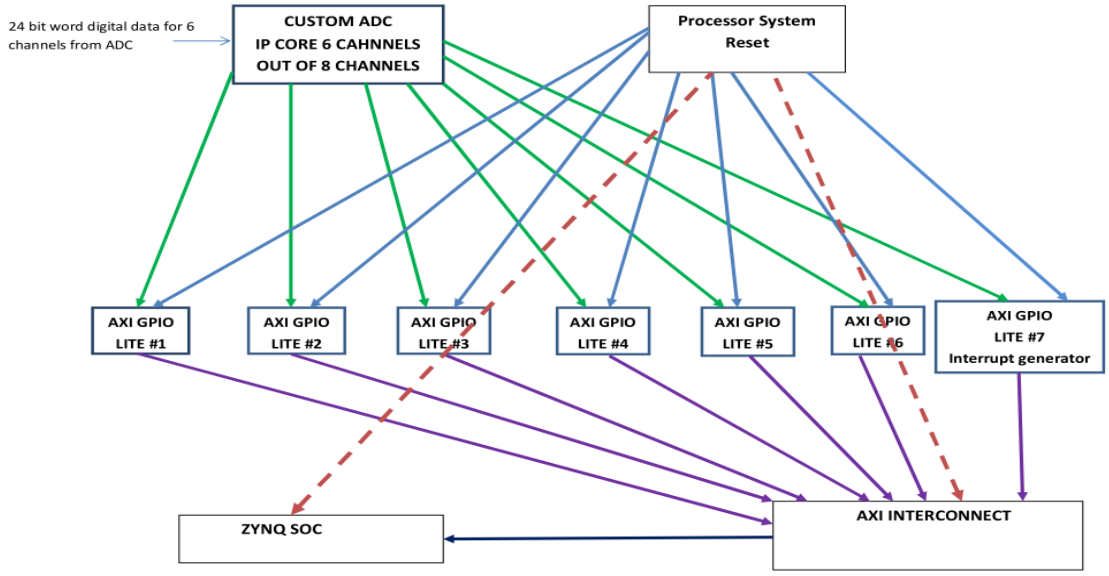


Figure 7. PL section Design with ADC IP core

New data transaction starts at the end of each conversion on the falling edge of BUSY. As per the protocol as shown in Figure 6, 24 bit data frame is sent from ADC to ADC IP core, named as “ltc2358sh_V1_0” whose symbol or IO block diagram is shown in Figure 3.

The mapping between the serial Soft Span configuration word, the internal Soft Span configuration register, and each channel’s 3-bit Soft Span code are illustrated in Table 1 [1]. The input voltage range for each channel and enabling of channels can be enabled from the ARM core in the PS section of Zynq FPGA, based on the requirement by setting the corresponding soft span code. The flow chart is shown in figure 7. PL section Design with ADC IP core is shown in Figure 8. A portion of the code or Pseudo random code is provided. Next the inner block diagram of ADC IP core and the realization is explained below.

Table 1. Soft Span Configuration

Sl no	**Binary Soft Span CODE SS[2:0]**	**Analog Input Range**
1	111	+/- 10.24V
2	110	+/- 10V
3	101	0 to 10.24V
4	100	0 to 10V
5	011	+/- 5.12V
6	010	+/- 5V
7	001	0 to 5.12V
8	000	CHANNEL DISABLE

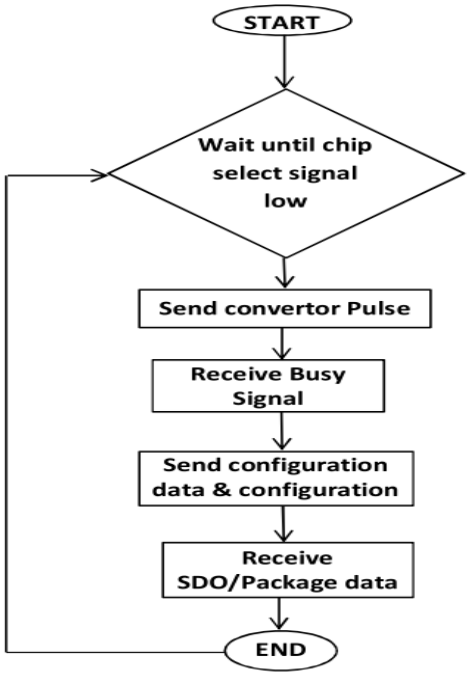


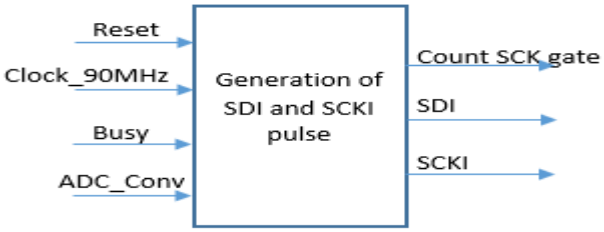
Figure 8. Flow chart

4.1. Generation of Conversion Pulse

According to the figure shown if the reset signal is zero, the output as well as internal counter is set / initialised to zero. And if the reset is high, the process for the generation of ADC conversion pulse begins. An ADC convertor pulse of 44.44ns is generated by running an internal counter. The counter is made high for 2 consecutive clock pulses of 45 MHz.

4.2. Generation of SDI and SCKI pulses

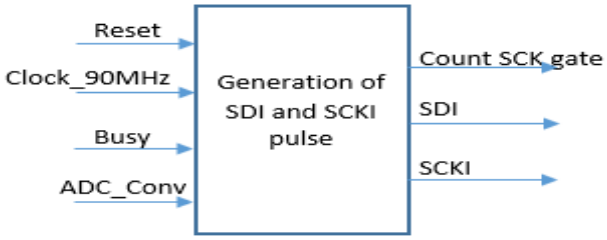
According to the figure shown if the reset signal is zero, the output internal signals, internal counter and shift register are set to zero. And if the reset is high, the process for the generation of Serial data input (SDI) and serial clock input (SCKI) begin with respect to 90 MHz clock.



If busy pulse equals to “0” and ADC convertor pulse equals to “0”, it is checked if sense busy is “1”, then the counter is incremented for 48 pulses. SCKI of 45MHz is generated by toggling the 90 MHz SCKI pulse and SDI is transferred with the configuration data “03FFFF” from a 24 bit shift register which was already initialized to “03FFFF”. In all the other cases of busy pulse and ADC convertor pulse, internal signals and counters are reset to “0”. After this all the internal counter and shift register is reset to zero.

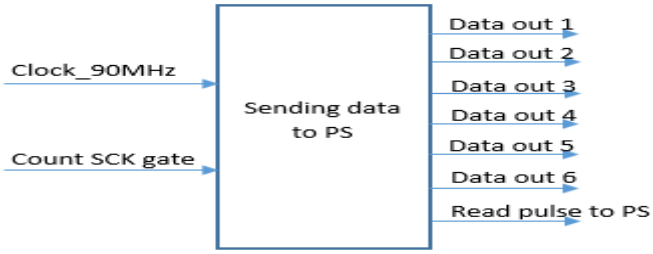
4.3. ADC Channel Data Reception

For the data reception process, during each falling edge of SCKI, 24 bit shift registers (ADC_CH1 to ADC_CH6) are filled from the data coming from 6 ADC channels respectively.



4.4. Sending ADC data to PS (ARM core)

As shown in the figure below for the rising edge of 90MHz clock, when the *Count_SCK_Gate = “49”*, a frame of complete 24 bit data is transferred to ZYNQ. An interrupt pin called read pulse is made high which indicates the PS that now the data is ready to be read from PL. After this process, the data in 6 channels are kept constant until the next data frame is ready and the read pulse is made “0”.



A portion of the ADC IP core to generate convertor pulse in PL section is,

```
if( Count_conv < 447 )    then Count_conv <=Count_conv + 1;
    elsif (Count_conv >= 447) then Count_conv <= 0;
```

The pseudo random code to generate SDI and SCKI in PL section is,

```
if (sense_busy = '1') then
                if(Count_SCK_Gate < 447) then
                    if(Count_SCK_Gate < 48) then
        SCK_Gate <= '1';            SCKI <= not SCKI;
        if(SCKI = '1') then SDI <= SDIreg(23);  SDIreg <= SDIreg(22 downto 0) & '0';
```

```
elsif(SCKI = '0') then null;
end if;
elsif(Count_SCK_Gate >= 48) then
        SCKI <= '1';     SDI <= '0';     SCK_Gate <= '0';
end if;
     Count_SCK_Gate <= Count_SCK_Gate+1;   sense_busy <= '1';
elsif (Count_SCK_Gate >= 447) then
          Count_SCK_Gate <= 0;   sense_busy<='0';
             SCKI <= '1';     SDI <= '0';     SCK_Gate <= '0';
end if;
```

The pseudo random code to capture ADC data is,

```
if(SCK_Gate = '1') then
DATA_OUT1<=ADC [1]_CH1(23 DOWNTO 0); RdPulse2ARM<='1'; datacount<= 0;
```

Pesudo Random code / Section of the code written in PS section is provided below:

```
/* Initialize the ctrl driver */
Status = XGpio_Initialize(&ctrl, GPIO_ctrl_DEVICE_ID);
if (Status != XST_SUCCESS) {
        xil_printf("Read control from PL Failed\r\n"); return XST_FAILURE;
                }
                /* Set ctrl as inputs*/
XGpio_SetDataDirection(&ctrl, control, RdPulse2ARMOp);
                xil_printf("Capturing ADC data of all 6 channels\r\n");
                while(Rdcnt < 10)
                readctrl_adc = XGpio_DiscreteRead(&ctrl, control);
                if (readctrl_adc == 0x00)      OnePadcrd  = 0x01;

        else if ((readctrl_adc == 0x01) && (OnePadcrd  == 0x01))
                OnePadcrd  = 0x00;              Rdcnt += 1;
                xil_printf("-----%dth Read ---\r\n",Rdcnt);
//ADC_CH1    // Initialize the ADC_CH1 driver
Status = XGpio_Initialize (&ADC_CH1,  GPIO_ADC_CH1_DEVICE_ID);
        if (Status != XST_SUCCESS) {
        xil_printf("ADC1 Initialization Failed\r\n");
                return XST_FAILURE;
        // Set ADC_CH1 as inputs
 XGpio_SetDataDirection(&ADC_CH1, ADC1_CHANNEL, adcin);
                // Reading ADC input
CH1_IN = XGpio_DiscreteRead(&ADC_CH1,ADC1_CHANNEL);
                xil_printf("CH1_IN = %x\n",CH1_IN);
```

5. SIMULATION RESULTS AND TEST RESULTS

The simulation results were viewed on ILA in VIVADO tool, as shown in figure 9. As a part of booting, ADC IP core and its associated functional modules are implemented in PL section to

transfer the data from ADC IP core to Zynq PS section through AXI interconnect and to transfer the configuration words for all the 6 channels through AXI interconnect from PS section. After successful booting of the board in PetaLinux, the ADC program was executed in PS section as per the application imposed sequence. The output will be shown in the running console to verify the ADC value and Voltage of all 6 channels as shown in figure 9. The figure 9 shows the test results when an input voltage of zero is fed to all the 6 channels.

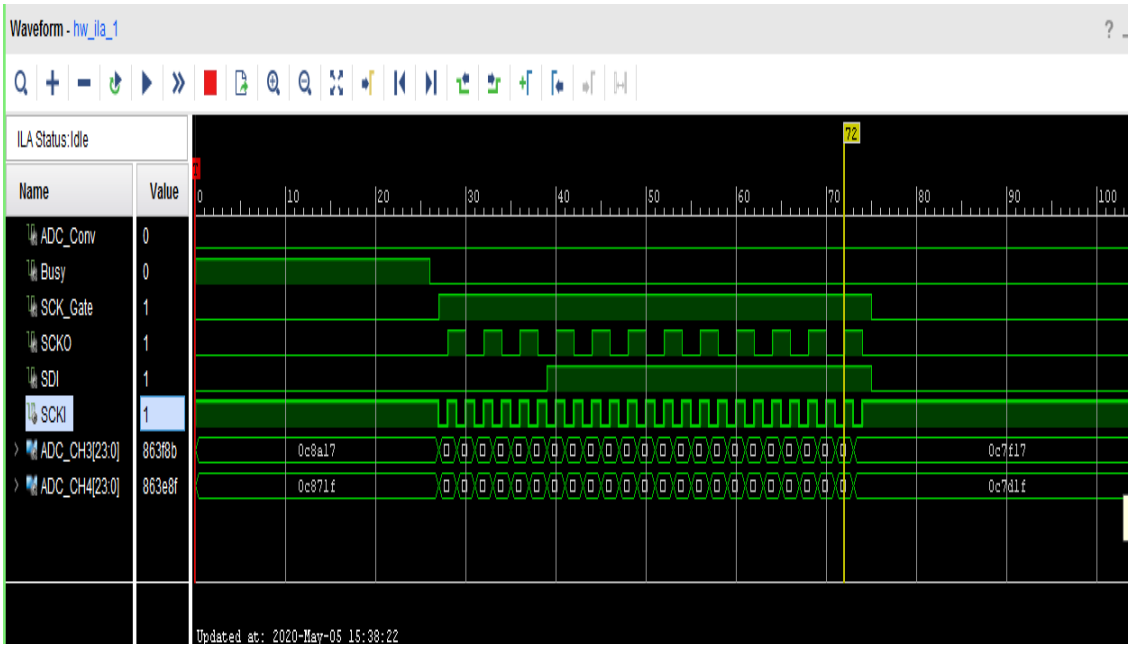


Figure 9: Simulation result

```
Channel 1 Data: 0x000007 >>> Voltage: 0.0000V
Channel 2 Data: 0xfffe0f >>> Voltage: -0.0006V
Channel 3 Data: 0x000017 >>> Voltage: 0.0000V
Channel 4 Data: 0x00001f >>> Voltage: 0.0000V
Channel 5 Data: 0xffff27 >>> Voltage: -0.0003V
Channel 6 Data: 0xfffe2f >>> Voltage: -0.0006V
```

Figure 10. Test result

6. CONCLUSIONS

Provision is made to configure the input analogue voltage range of each channel of ADC based on the application through soft span configuration, So that the developed IP core can be reused for many applications. Most of the Custom IP cores for medium logic can be developed by following the development flow explained in this paper, unless the requirement needs to any high speed data transfer. Even for such application requirement, DMA based data transfer is implemented to store the data at the end in the DDR attached with PS. Since PS section has the on chip peripheral support for DDR, DDR was connected to PS. Integrated Logic Analyser (ILA) [7] which is an integrated verification tool of Vivado is used for Verification. No Third party tool is required. Accuracy, and the consistency have been verified successfully in standalone mode testing and even in the integrated mode when this code is integrated with others in the equipment

and then at the end with the system at the actual field. The ADC IP core is developed in a modular approach [8] to scale up to accommodate the balance 2 channels for the variant / different user requirement. Because of Modular design, the same design can be used to enhance to N numbers such ADC channels.

The realised ADC IP core and the approach followed will be used as reference to develop other AXI peripherals such as NVSRAM, RTC interface, UART, MRAM, etc., since the front core interface only differs based on the off chip peripheral interface for future work. Once the data is received into PL section as per the custom interface, then those data are to be reframed, and packed into FIFO or BRAM to send them to ARM processor through AXI. Memory interface with custom front core interface is recommended to allot and distribute the available processor's execution time to multiple AXI peripherals and other off chip peripherals directly connected to on chip ARM processors.

Since the SPI interface based data transfer protocol can support up to Mbps only, JESD204A, B, C are used for Gbps range data transfer. JESD204C is the latest version of JESD204 which is the only interface recommended for Gigabit ADCs and DACs. The same methodology will be exploited to develop the custom JESD204C IP core with MPSoC [10].

ACKNOWLEDGEMENTS

The authors would like to thank all the people who have come across in their life from whom the authors have learnt either directly or indirectly.

REFERENCES

[1] LTC2358 datasheet Buffered Octal, 16-Bit, 200 ksps / Channel Differential ±10.24V ADC with 30VP-P Common Mode Range. D16870-0-5/18(A) . www.analog.com ANALOG DEVICES, INC. 2016-2018.

[2] Yueli Hu, Wenyi Jing, Ying Liu, "Integrating ADC IP in SoC and Its Verification", IEEE International Symposium on High Density Packaging and Microsystem Integration, Shanghai, China, 26-28 June 2007.

[3] Cristian Sisterna, Marcelo Segura, Martin Guzzo, Gustavo Ensinck, Carlos Gil, "FPGA Implementation of Ultra-High Speed ADC interface." IEEE VII Southern Conference on Programmable Logic [SPL], Cordoba, Argentina, 13-15 April 2011.

[4] ZYNQ -7000 SoC Technical Reference Manual UG585 (v1.12.2) July 1, 2018 Xilinx

[5] ZYNQ UltraScale+ RFSOC Data Sheet: Overview DS889 (v1.10) September 14, 2020 Xilinx

[6] Xilinx VIVADO/SDK Tutorial (Laboratory Session 1, EDAN15) Flavius.Gruian@cs.lth.se March 21, 2017

[7] Integrated Logic Analyser V6.2 – LogicCORE IP Product Guide, Vivado Design Suite, PG172 October 5, 2016, XILINX

[8] Yunpeng Bai, Dominic Gaisbauer, Stefan Huber, Igor Konorov, Dmytro Levit, Dominik Steffen, Stephan Paul, "Intelligent FPGA Data Acquisition Framework", IEEE-NPSS Real Time Conference, Padua, Italy, 6-10 June 2016.

[9] Jiiadong Yuan, Min Xie, Siyuan Liu, Dengyue Zhai, "Design of JESD204B Multichannel Data Acquisition and playback system based on SoPC", 11th international congress on Imange and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), 2018

[10] https://www.zynq-mpsoc-book.com

AUTHORS

A. Suresh was born in Virudhunagar District in TN. He received his B.E. degree in ECE from Government College of Engineering, Tirunelveli in 1998. He worked as a Lecturer in AKCE, Krishnankovil, TN from 1998 to 2000. He received his M.E in Communication Systems from REC, Trichy in 2002. Since 2002, he is working in Bharat Electronics Limited, Bengaluru. Now, he is DGM in Embedded Systems department, PDIC, Bengaluru, India.