BUILDING A SERIAL INTERFACE WITH ALTERA CYCLONE V FPGA

Jairo E. Villamizar Vásquez
Department of Electrical and Computer Engineering
Youngstown State University
OH, 44555, U.S.A
jevillamizar@student.ysu.edu

ABSTRACT

This paper presents the implementation of a serial interface using a Cyclone V development board to interface with analog-to-digital converters (ADCs) that output digitized data as a one-wire serial bitstream. Field Programmable Gate Arrays (FPGAs) are utilized for their flexibility in customizing digital systems, programmed using hardware description languages such as VHDL, Verilog, and SystemVerilog. The Serial Peripheral Interface (SPI) communication technique, known for its synchronous data transfer capabilities, is utilized to ensure precise timing and reliable data exchange. The DE10-Standard Development Kit, featuring the Cyclone V SoC FPGA and the LTC2308 ADC, is used to demonstrate the process, including configuring the ADC for single-ended and differential inputs and achieving a maximum sampling rate of 500KSPS. The system's architecture incorporates a Phase-Locked Loop (PLL) to match the required clock frequencies, and an algorithm to synchronize and process the bitstream into a 12-bit digital representation. Experimental results confirm the efficacy of the implemented SPI interface, making it a robust solution for real-world digital system applications.

***K*eywords** FPGA, Embedded systems, SPI communication

1 Introduction

In recent years, the versatility and configurability of Field Programmable Gate Arrays (FPGAs) have made them a key technology in digital systems design. Unlike Application-Specific Integrated Circuits (ASICs), which are fixed-function devices manufactured for specific tasks, FPGAs offer a unique advantage by allowing users to define and modify their hardware configurations post-production. This adaptability is achieved through programming languages such as VHDL, Verilog, and SystemVerilog, which enable precise control over the FPGA's digital logic [1][2]. The ability to customize FPGA functionality for specialized applications not only improves performance but also optimizes system efficiency [3].

A significant application of FPGAs is in the implementation of communication protocols, such as the Serial Peripheral Interface (SPI). SPI is a synchronous serial communication technique that facilitates high-speed data transfer between microcontrollers and peripheral devices. Unlike asynchronous communication, SPI uses a clock signal to synchronize data exchange, ensuring reliable and accurate data transmission [4][5]. This characteristic is essential in applications requiring high data integrity and speed.

The DE10-Standard Development Kit, equipped with a Cyclone V SoC FPGA, illustrates the practical use of FPGAs in interfacing with various peripherals. This kit includes the LTC2308 Analog-to-Digital Converter (ADC), which employs SPI communication for efficient data conversion. The LTC2308 supports multiple input modes and sampling rates, making it a flexible tool for digital systems that demand high-resolution and rapid data acquisition [6]. The configuration of this ADC and the subsequent data handling through the FPGA are important aspects of developing effective digital systems.

This paper explores the implementation of a serial interface using the Cyclone V FPGA and the LTC2308 ADC within the DE10-Standard Development Kit. We examine the SPI communication protocol, the ADC's configuration modes, and the system's architecture. Additionally, we provide detailed insights into the system's performance, including the design of a custom algorithm for data processing. Through experimental validation, we demonstrate the effectiveness of this SPI interface in achieving accurate and reliable data transfer, showcasing its potential for applications in real-world digital systems.

2 Field programmable gate arrays - FPGA

Field Programmable Gate Arrays (FPGAs) are versatile integrated circuits that allow users to customize their functionality. Unlike fixed-function chips, FPGAs are delivered in an unprogrammed state, enabling users to define specific configurations.

Compared to Application-Specific Integrated Circuits (ASICs), which are pre-configured during manufacturing, FPGAs provide significant flexibility. Users can modify their configuration as needed to optimize performance or address specific requirements without altering the hardware. [1]

FPGAs are typically programmed using hardware description languages such as VHDL, Verilog, and SystemVerilog. These languages specify the behavior and interconnections of digital circuits, enabling precise configuration of the FPGA's logic [2]. The primary advantage of FPGAs over conventional processors lies in their adaptability to specific tasks, offering enhanced efficiency and performance for specialized applications [3].

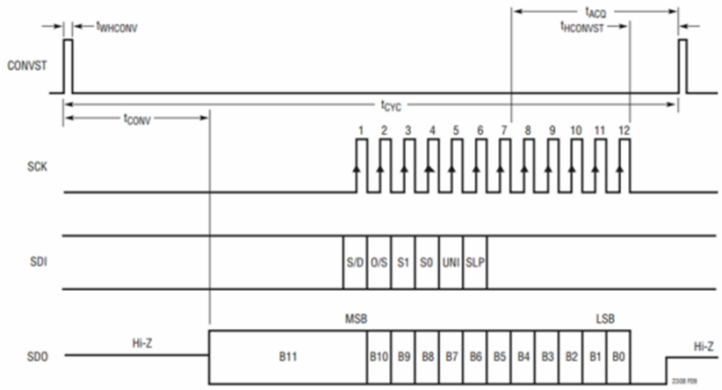
3 SPI Communication

Serial Peripheral Interface (SPI) is a widely used interface for data transfer between microcontrollers. It operates using independent clock and data wires, along with a chip selection line to choose the device to communicate with. In serial communication, there are asynchronous and synchronous modes. In asynchronous communication, there is no control over when data is transferred. In contrast, synchronous communication relies on a clock signal to establish precise timing, ensuring that both the emitter and receiver are synchronized [4]. This allows the emitter to send data and the receiver to know exactly when to expect and read the data. SPI operates as a synchronous serial communication technique. The clock signal ensures that both sides of the communication are perfectly synchronized. It dictates to the receiver exactly when to sample the incoming bits from the data channel, either on the rising or falling edge of the clock. Once the receiver detects the clock edge, it reads the next bit on the data line, ensuring accurate and reliable data transfer. [5]

4 DE10-Standard Development Kit

The DE10-Standard Development Kit includes a Cyclone V SoC FPGA and various peripherals, such as push buttons, 7-segment displays, audio interfaces, an I2C multiplexer, VGA outputs, a TV decoder, SDRAM, PS2 ports, an ADC, and more. This board includes an analog-to-digital converter, the LTC2308, which is a low-noise, 500ksps max, 8-channel, 12-bit resolution ADC with an SPI interface [6]. This microchip can sample 8 different analog signals and transmit the actual value of each in a 12-bit word. The maximum sampling rate is achieved when only one analog channel is used. If all eight channels are used, 62500 samples per second can be obtained from each analog input. The LTC2308 transmits data using a 4-wire SPI interface. When a rising edge of CONVST occurs, the conversion process is initiated. The manufacturer recommends waiting 40ns after the conversion starts to pull down CONVST. Alternatively, it can be pulled down just until the conversion ends and before the serial transmission. SCLK can operate up to 40MHz.

Figure 1: LTC2308 SPI signals



Source: Linear Technologies

5 Configuration Modes

The LTC2308 operates in several modes based on the values of a 6-bit word wired to the SDI port. Each bit on SDI is loaded on the rising edge of SCK, while SDO bits are loaded on the falling edge of SCK. The configuration bits determine the ADC's mode of operation, including single-ended and differential inputs. When using differential inputs, the input count is limited to four channels.

Figure 2: LTC2308 configuration bits

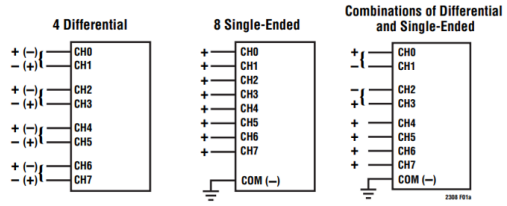
S/D	O/S	S1	S0	UNI	SLP

S/D = SINGLE-ENDED/$\overline{\text{DIFFERENTIAL}}$ BIT
O/S = ODD/$\overline{\text{SIGN}}$ BIT
S1 = ADDRESS SELECT BIT 1
S0 = ADDRESS SELECT BIT 0
UNI = UNIPOLAR/$\overline{\text{BIPOLAR}}$ BIT
SLP = SLEEP MODE BIT

Source: Linear Technologies

A combination of single-ended and differential inputs can be configured, affecting the number of input channels. The different combinations of the configuration bits are listed in the following table.

Figure 3: LTC2308 Input modes



Source: Linear Technologies

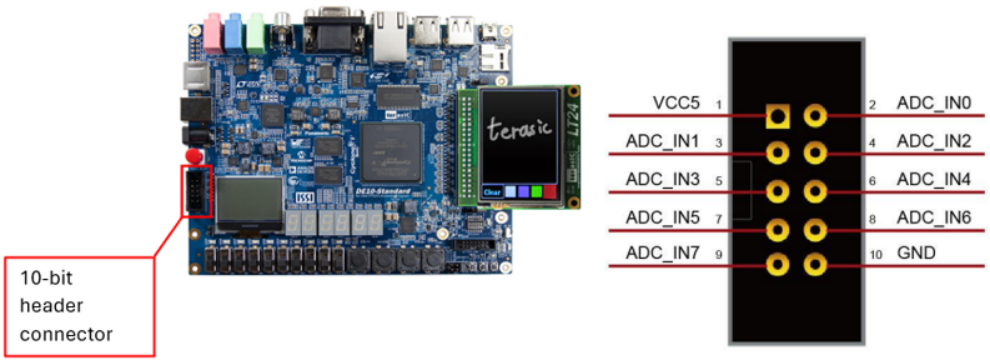
Analog signals are input to the DE10 board through a 10-wire header connector. It is important to note that PIN1 (VCC5) must be connected to a 5V rail. If the interface is configured in unipolar mode, the input voltage range is 0 - 4.095V. In contrast, in bipolar mode the input voltage range is from -2.048V to 2.047V.

Figure 4: LTC2308 Channel configuration

S/D	O/S	S1	S0	0	1	2	3	4	5	6	7	COM
0	0	0	0	+	−							
0	0	0	1			+	−					
0	0	1	0					+	−			
0	0	1	1							+	−	
0	1	0	0	−	+							
0	1	0	1			−	+					
0	1	1	0					−	+			
0	1	1	1							−	+	
1	0	0	0	+								−
1	0	0	1			+						−
1	0	1	0					+				−
1	0	1	1							+		−
1	1	0	0		+							−
1	1	0	1				+					−
1	1	1	0						+			−
1	1	1	1								+	−

Source: Linear Technologies

Figure 5: 10-wire header connector for analog signals in DE10 board



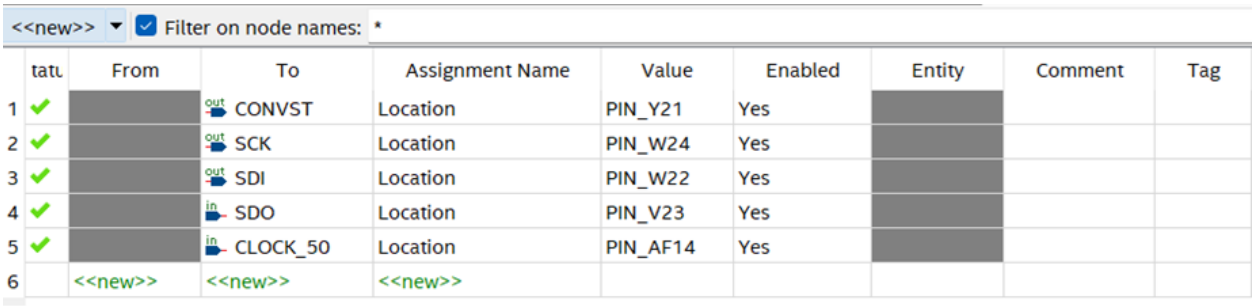
Source: DE10 Standard User Manual

6 One Single-Ended Input at 500KSPS

As mentioned in the ADC features, the maximum sampling rate is 500KSPS (500,000 samples per second). This means the entire process for one sample must take 2µs. The LTC2308 datasheet states that the conversion time is up to 240ns. With this in mind, and considering that SCK will run at 40MHz, the serial transmission will be completed in 300ns. After this time, the ADC will rest for 1460ns. For this implementation, the analog signal is input through ADC_IN1 (+) and COM (GND) in unipolar mode. Therefore, the channel configuration sent to the ADC device was "1000". It is relevant to notice that the 10-pin port must be powered by an external 5V DC supply (VCC5). The DE10 development board already has the pins of the LTC2308 connected to the Cyclone V FPGA and the 10-bit header connector. Therefore, the Pin Assignments must be performed as indicated in the DE10_Standard_User_Manual. For this implementation, Quartus Prime 18.1 was used. [7]

First, the embedded system utilizes the fast clock included on the DE10 board, running at 50MHz. However, the maximum clock for the analog-to-digital conversion using the LTC2308 is 40MHz. Therefore, a PLL is used to decrease the frequency from 50MHz to 40MHz. The PLL blocks are included in Altera FPGA microchips. This can be found in Quartus IP Catalog (Library/Basic Functions/Clocks; PLLs and Resets). Here, the input and output frequency of the PLL must be specified. For this design, only one PLL output is used. The logic inside the SPI interface block must replicate the CONVST, SCK, SDI, and SDO signals to obtain the 12-bit digital representation of the analog input.

Figure 6: Pin assignments in Quartus project

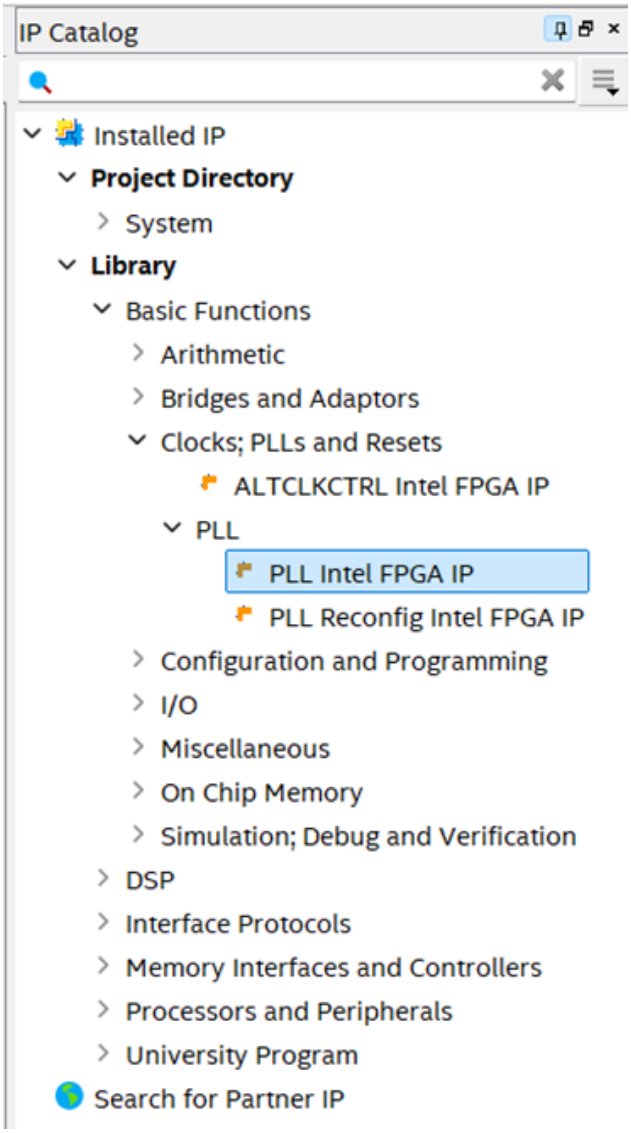


<<new>> Filter on node names: *

	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✔		CONVST	Location	PIN_Y21	Yes			
2	✔		SCK	Location	PIN_W24	Yes			
3	✔		SDI	Location	PIN_W22	Yes			
4	✔		SDO	Location	PIN_V23	Yes			
5	✔		CLOCK_50	Location	PIN_AF14	Yes			
6		<<new>>	<<new>>	<<new>>					

Source: Quartus Prime 18.1

Figure 7: IP Catalog in Quartus Prime



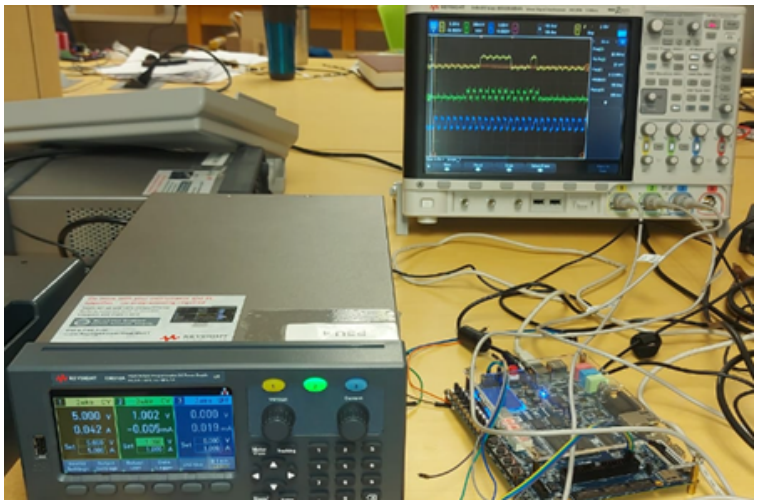
Source: Quartus Prime 18.1

Figure 8: Architecture of the embedded system

Source: Quartus Prime 18.1

The architecture of the embedded system includes the PLL mentioned before, named as clock divider, and a SPI interface block. The inputs of this architecture are the 50MHz clock included in the DE10 board, and the bitstream coming from LTC2308 through SDO pin. On the other hand, the architecture has three outputs: CONVST, SCK, and SDI, to replicate the SPI signals of the ADC device.

Figure 9: SPI interface experiment



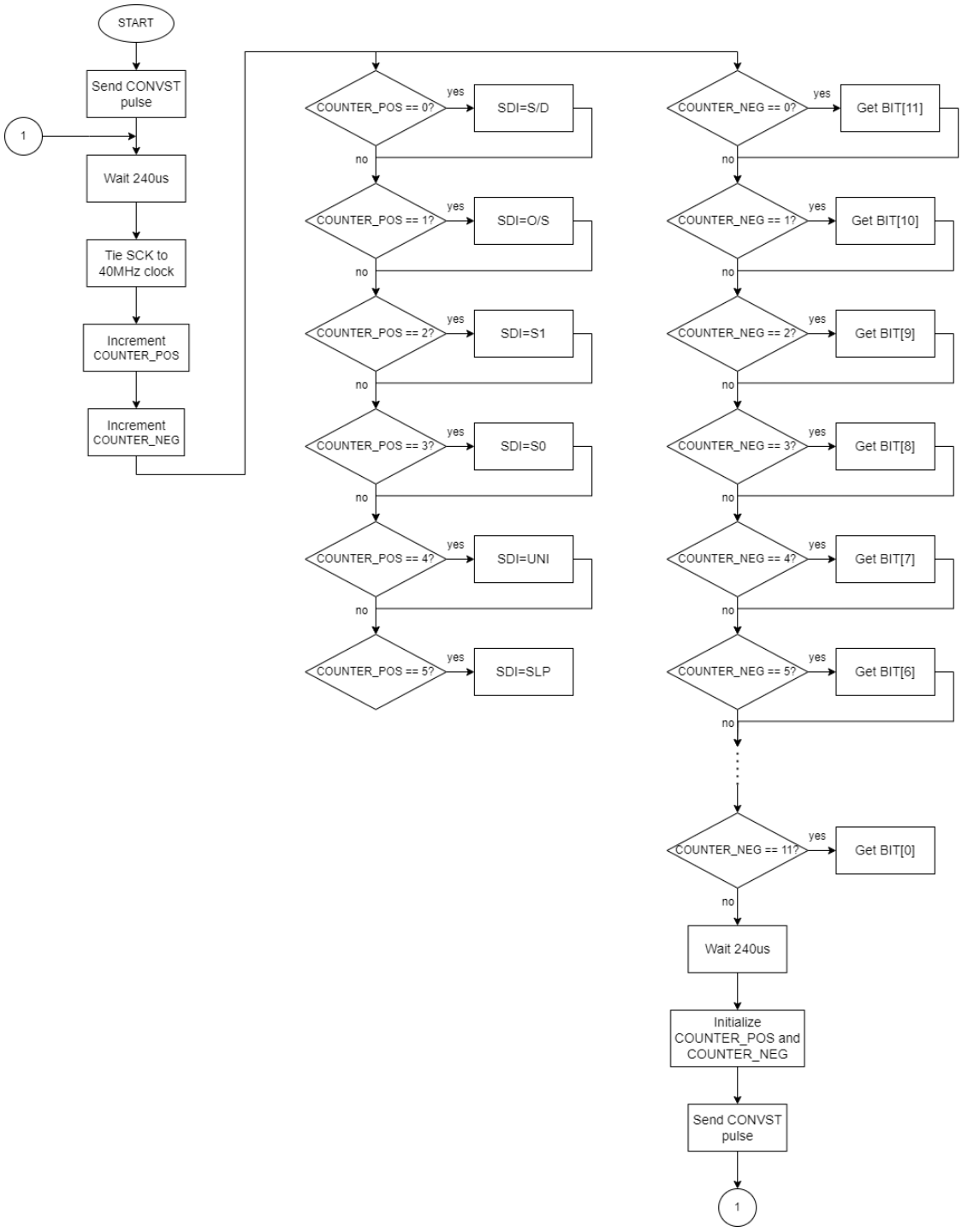
Source: Author

During the experiment, a DC power supply was used to feed the required 5V to power the device. Also, different DC values between 0 to 4.095V were applied to ADC_IN1 and the results were evaluated in oscilloscope.

7 Algorithm

The SPI interface block contains the algorithm that obtain a 12-bit word from a bitstream. First, it starts the synchronized communication by sending a 2.5ns pulse through CONVST pin. Then, it waits during $240\mu s$. After this time, the transmission occurs. To get the synchronization, the block ties SCK pin to the 40MHz clock obtained from the PLL. Two counters are used to control the bitstream transmission: COUNTER_POS for rising edges and COUNTER_NEG for falling edges in SCK. Depending on the counters' quantities, the configuration bits are sent to the ADC device, and the digitized conversion bitstream is received. The received bits are stored in a 12-bit word to be used in further stages.

Figure 10: SPI interface communication algorithm



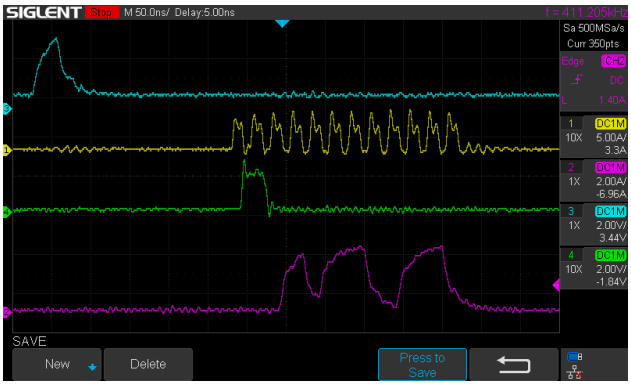
Source: Author

8 Results

As mentioned before, the serial interface was tested for different analog inputs in the unipolar mode. In the first test, 716mV was applied to ADC_IN1. Considering that 4.095V corresponds to the ”111111111111” 12-bit word, it is

expected that the digitized conversion of the voltage input is ”001011001100”. This was indeed the received word through the SDI pin.

Figure 11: SPI transmission with analog input equals 716mV

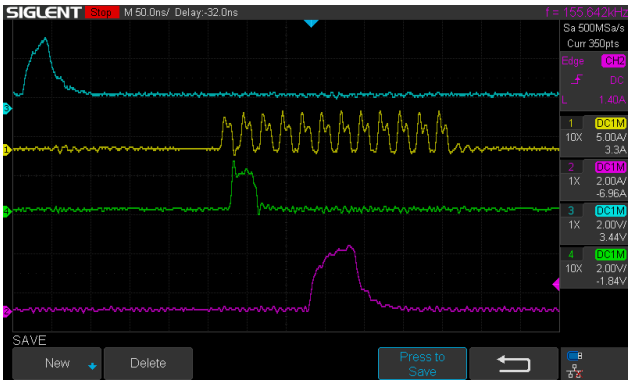


Source: Author

In another test, an input of 192mV was applied. As a result, ”000011000000” 12-bit word was received.

The results of these tests indicate that the SPI interface works properly. Therefore, the received 12-bit word can be used for further purposes in the digital system.

Figure 12: SPI transmission with analog input equals 192mV



Source: Author

9 Source codes

Figure 13: Global variable declaration

```
module interface_spi(
input SDO,
input CLOCK_40, //clock running at 40MHz
output SCK,
output CONVST,
output [11:0] adc_output,
output [11:0] SP
);
```

Source: Author

Figure 14: Local variable declaration

```
logic SDI;
logic [15:0] counter_pos;
logic [15:0] counter_neg;
logic S_D;
logic O_S;
logic S1;
logic S0;
logic UNI;
logic SLP;
```

Source: Author

Figure 15: Configuration of SPI transmission

```
always_ff @(posedge CLOCK_40_2) begin
        if (counter_pos< 160) begin
                counter_pos<=counter_pos+1;
        end
        if (counter_pos== 160) begin
                counter_pos<=0;
        end
        if (counter_pos> 80 && counter_pos<92) begin
                SCK<=CLOCK_40_2;
        end
        if (counter_pos<= 80 | counter_pos>=92) begin
                SCK<=CLOCK_40_2;
        end

        case (counter_pos)
                0: begin
                        CONVST<=1;
                end
                1: begin
                        CONVST<=0;
                end
                80: begin
                        SDI<=S_D;
                end
                81: begin
                        SDI<=O_S;
                end
                82: begin
                        SDI<=S1;
                end
                83: begin
                        SDI<=S0;
                end
                84: begin
                        SDI<=UNI;
                end
                85: begin
                        SDI<=SLP;
                end
        endcase
end
```

Source: Author

Figure 16: SPI Data extraction

```
always_ff @(negedge CLOCK_40) begin
        counter_neg<=counter_neg+1;
        if (counter_neg== 160) begin
                counter_neg<=0;
        end

        case (counter_neg)
                81: begin
                        adc_output[11]<=SDO;
                end
                82: begin
                        adc_output[10]<=SDO;
                end
                83: begin
                        adc_output[9]<=SDO;
                end
                84: begin
                        adc_output[8]<=SDO;
                end
                85: begin
                        adc_output[7]<=SDO;
                end
                86: begin
                        adc_output[6]<=SDO;
                end
                87: begin
                        adc_output[5]<=SDO;
                end
                88: begin
                        adc_output[4]<=SDO;
                end
                89: begin
                        adc_output[3]<=SDO;
                end
                90: begin
                        adc_output[2]<=SDO;
                end
                91: begin
                        adc_output[1]<=SDO;
                end
                92: begin
                        adc_output[0]<=SDO;
                end
        endcase
end
```

Source: Author

References

[1] Dumitrel Cătălin Costache, Lucian Andrei Perişoară, and Adriana Florescu. Fpga implementation of a sd card controller using spi communication. In *2020 12th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–4, 2020.

[2] Jairo E. Villamizar Vasquez. *Design and Development of FPGA-Based Control System for a 50KW IGBT-Driven Induction Heater*. PhD thesis, 2023.

[3] Ashish Alape Vivekananda and Eduard Enoiu. Automated test case generation for digital system designs: A mapping study on vhdl, verilog, and systemverilog description languages. *Designs*, 4(3), 2020.

[4] Jiayi Qiang, Yong Gu, and Guochu Chen. Fpga implementation of spi bus communication based on state machine method. *Journal of Physics: Conference Series*, 1449(1):012027, jan 2020.

[5] Abdelkrim Oudjida, Mohamed Lamine Berrandjia, R. Tiar, Liacha Ahmed, and Khalid Tahraoui. Fpga implementation of i2c & spi protocols: A comparative study. 12 2009.

[6] J. Velásquez-Aguilar, Outmane Oubram, and Luis Cisneros Villalobos. *Real-Time FPGA-Based Systems to Remote Monitoring*. 12 2019.

[7] Ross Snider. *Chapter 4: Introduction to the DE10-Nano Board*, pages 33–39. Springer International Publishing, Cham, 2023.