IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 4, Issue 4, Ver. II (Jul-Aug. 2014), PP 12-17
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Software Programmable ADC and DAC interfaces in VHDL

Sangeetha Muthukrishnan, Reena. P
Department of Electronics & Communication Engineering, Government College of Engineering Kannur, Kannur, Kerala 670 563

Abstract**:** *This paper proposes a design to interface ADCs and DACs of different data width to a processor on an FPGA using VHDL. The ADCs and DACs are connected to a processor and the characteristics of the interface can be changed by programming the processor. Any generic device can be connected using the same interface. Verification is done by Open verification methodology using System Verilog.*
Index Terms**:** *ADC, DAC, IP core, FPGA, OVM testbench.*

I. Introduction

Today the world is moving towards a digital platform. At the same time, it is seen that all physical world signals available to us are still very much analog in nature, but their processing obviously happens in the digital domain. So for such a wide a variety of applications, it is of utmost importance that we have properly designed Analog-To-Digital Convertors (ADCs) and Digital-to-Analog Converters (DACs).

A FPGA (Field Programmable Gate Array) board can be used to perform a lot of operations on the real world signals, be it simple arithmetic operations or complex transforms. The primary motivation behind taking up this project is the large utility of the digital signals on which a lot of operations and transforms, viz Short time Fourier transform, Stockwell Transform, Wavelet transform etc. could be done, which are very tedious and time consuming processes if the signals are in the analog domain. Simultaneously we must also remember that all digital signals cannot be used directly, as all the real world signals are more or less analog in nature. Hence, it is of utmost importance that we are able to use the ADC and DAC effectively and frequently.

Field Programmable Gate Array which provides flexibility in configuring the device according to the user requirement [1]. The major defining characteristic of the FPGA is that it can be programmed. Programming an FPGA is very different from a microprocessor or a DSP processor. A computer system contains both a CPU and a separate memory that stores the instruction and data. The FPGAs program is interwoven into the structure of FPGA. The programming directly implements the logic functions and interconnections. When an FPGA is used in final design, the jump from prototype to product is much smaller and easier. They are having a large number of input and output lines compared to microprocessors, micro-controllers and DSPs. FPGAs are having a higher process-

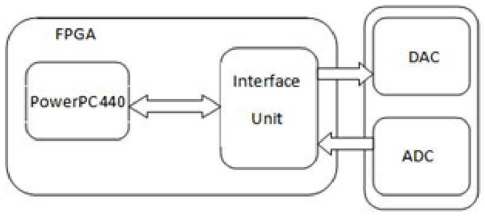


Fig. 1. Basic Block Diagram

ing speed compared to microprocessors and microcontrollers which is a need in most of the control application. A system usually has an embedded user interface as a form of software and encompasses many components inside, not only the hard-ware but also the software that constitutes the system [2]. Such a complicated entity can be handled only with computer-aided design

The existing systems are based on traditional processor based systems with multiple ICs for various functionalities. This introduces a larger space and power which leads to low reliability. In order to reduce the space and power, an FPGA based systems is proposed, where all the digital functionalities are implemented on an FPGA [3]. All the analog circuitry is kept external with proper interface to the processor within the FPGA [4]. The system will be realized using Xilinx Virtex 5-Fx series FPGA, where the PowerPC 440 processor core is established. The processor can be interface with all the peripheral through a high speed On-chip Peripheral Bus.

Traditionally one interface can connect to only a single ADC or DAC for which it is designed [5]. This is a disadvantage when the final system is uncertain or several peripherals with different characteristics need to be interfaced. All the ADCs or DACs can be connected via a multiplexer easily but the internal code also needs to be changed for each device. Since this is not a feasible solution, we need a generic design to which any arbitrary device can be connected.

II. Design Procedure

The basic block diagram as shown in Fig. 1 is shown. The PowerPC 440 hardcore processor can be used or the same

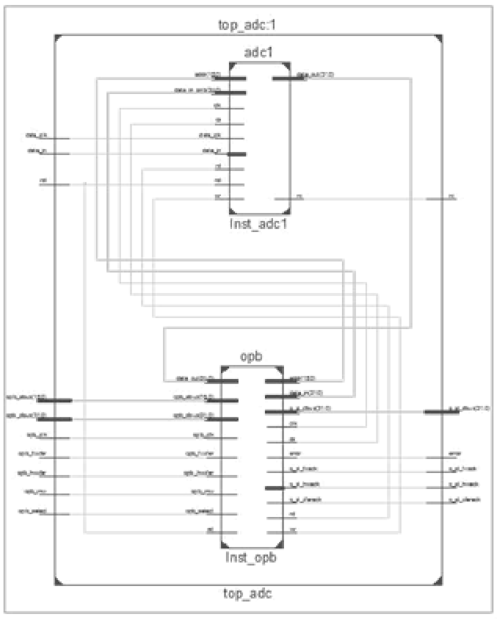


Fig. 2. RTL of ADC top module

design can be used to interface to a softcore processor or external processor. The interface to each is designed separately and also an OPB interface is designed to connect to the processor.

A. Implementation of ADC interface

The state IDLE initializes the interface of ADC. START initiates the ADC conversion process. CONV state produces a RC signal low which tells the ADC to start conversion. In the READ state a READ FLAG is set. The data from ADC is then stored in a temporary register and a FINISH FLAG set after the data transfer is completed. The serial data is made parallel and output to the DATA IN OPB pin which is the input data pin to the OPB interface [6].

The on-chip peripheral bus (OPB) is designed for easy connection of on-chip peripheral devices [7]. It provides a common design point for various on-chip peripherals. The OPB is a fully synchronous bus which functions independently at a separate level of bus hierarchy. It is not intended to connect directly to the processor core. The processor core can access the slave peripherals on this bus through the PLB to OPB bridge unit which is a separate core.

B. Implementation of DAC interface

The state IDLE initializes the interface of DAC. LOAD START initiates the DAC – conversion process by making the signal LOAD DATA to the DAC low. SCLK GEN OFF and SCLK GEN ON generate SCK high and low signals. 1 bit of data is output on the SDI pin for each pulse of SCLK. The number of bits is determined by the

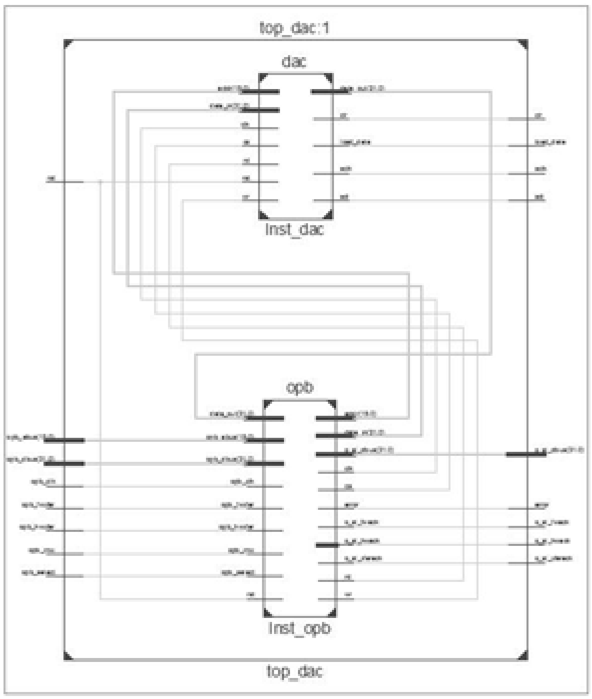


Fig. 3. RTL of DAC top module

value stored in the DATA WIDTH register. The input data is received through the OPB interface.

C. Software programmable registers

A set of registers present in both ADC and DAC modules make this design entirely software programmable. These registers can be written by the processor to which the design is connected.

In ADC, DATA WIDTH register stores the number of bits the ADC output, the ACQUISITION TIME register stores the number of clock cycles the ADC takes to acquire data after the RC (read/convert) signal is made low. The width of the RC pulse is stored in the RD TIME register.

In DAC, DATA WIDTH register stores the number of bits the DAC input, the SCLK ON TIME and SCLK OFF TIME register stores the number of clock cycles of the SCLK on and off time. The width of the LOAD pulse needed is stored in the LOAD TIME register.

Apart from this there is an OP DIRECTION register in both to change the polarity of the data if required.

The registers can be written or read by asserting WRITE REG and READ REG signals. All the registers are 32 bit registers and can be used to store the data related to any desired serial ADC and DAC even if this information is not previously known. It is highly beneficial in applications where the peripherals may be changed or is undecided.

III. Verification Procedure

A. Open Verification Methodology

An OVM testbench is composed of reusable verification environments called OVM verification components (OVCs) [8]. An OVC is an encapsulated, ready-to-use, configurable verification environment (as in Fig. 4) for an interface protocol,

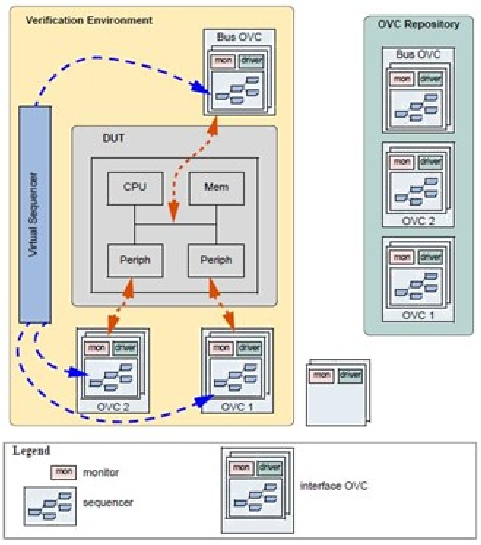


Fig. 4. OVM Environment

a design sub-module or a full system. Every OVC follows a consistent architecture and consists of a complete set of elements for stimulating, checking, and collecting coverage in-formation for a specific protocol or design. The OVC is applied to the device under test (DUT) to verify your implementation of the protocol or design architecture. OVCs expedite creation of efficient testbenches for your DUT and are structured to work with any hardware description language (HDL) and high-level verification language (HVL) including Verilog, VHDL, e, SystemVerilog, and SystemC.

This method is to build a coverage-driven functional verification testbench, with constraint random stimulus and direct stimulus. The hierarchical testbench, which is developed based on Open Verification Methodology (OVM), is reusable and efficient. When compared to traditional verification methods, it has been proven that this new one has greatly saved verification time and manpower, enhanced verification unit reuse [9].

B. OVC Overview

1) Data Item : Data items represent the input to the DUT. Examples include networking packets, bus transactions, and instructions. The fields and attributes of a data item are derived from the data items specification.

2) Driver: A driver is an active entity that emulates logic that drives the DUT. A typical driver repeatedly receives a data item and drives it to the DUT by sampling and driving the DUT signals. For example, a driver controls the read/write signal, address bus, and data bus for a number of clocks cycles to perform a write transfer.

3) Sequencer: A sequencer is an advanced stimulus generator that controls the items that are provided to the driver for execution. By default, a sequencer behaves similarly to a simple stimulus generator and returns a random data item upon request from the driver. This default behavior allows you to add constraints to the data item class in order to control the distribution of randomized values. Unlike generators that randomize arrays of transactions or one transaction at a time, a sequencer captures important randomization requirements out-of-the-box.

4) Monitor: A monitor is a passive entity that samples DUT signals but does not drive them. Monitors collect coverage information and perform checking. Even though reusable drivers and sequencers drive bus traffic, they are not used for coverage and checking.

5) Agent: Sequencers, drivers, and monitors can be reused independently, but this requires the environment integrator to learn the names, roles, configuration, and hookup of each of these entities. To reduce the amount of work and knowledge required by the test writer, OVM recommends that environment developers create a more abstract container called an agent. Agents can emulate and verify DUT devices. They en-capsulate a driver, sequencer, and monitor. OVCs can contain more than one agent. Some agents (for example, master or transmit agents) initiate transactions to the DUT, while other agents (slave or receive agents) react to transaction requests. Agents should be configurable so that they can be either active or passive. Active agents emulate devices and drive transactions according to test directives. Passive agents only monitor DUT activity.

IV. Simulation Results

The simulation software is Xilinx ISim. And the selected device is Virtex- 5 FXT FPGA.

Verification is done using OVM. System verilog is the language used. Clock frequency of 50MHz is used. Arbitrary inputs are given and output observed. For ADC the input is externally given and the same is transmitted to the OPB connection regardless of the data width and other factors as in Fig. 5. Similarly the DAC input given is transmitted at any desired clock rate as in Fig. 6. The system is found to be stable for any clock and SCLK with varying duty cycles. Using the OVM monitor 100 % code coverage is ensured.i.e all combinations of input is tried out in both ADC and DAC to get this coverage for the finite state machine designed

V. Results And Summary

100% functional code coverage has been obtained using OVM. The ADC uses 349 that is 3% of the registers and 469 that is 9% of the LUTs 7 whereas the the DAC uses 512 that is 5% of the registers and 906 that is 18% of the LUTs. Hence a total of 8% of registers and 27 % of registers have been utilised by this design. 227 and 340 flip flops are used respectively.

VI. Applications

The standalone implementation of ADC could be integrated in different projects. A DAC could be achieved and used in conjunction with ADC for numerous applications like Fast

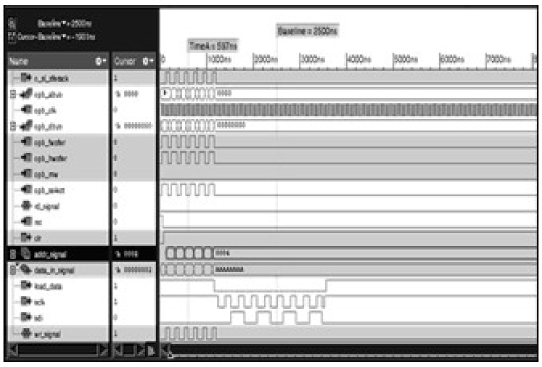


Fig. 5. ADC Verification result

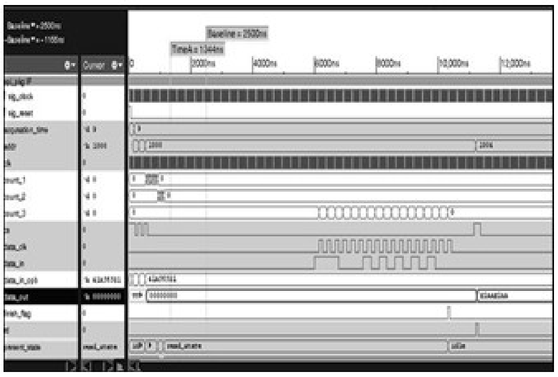
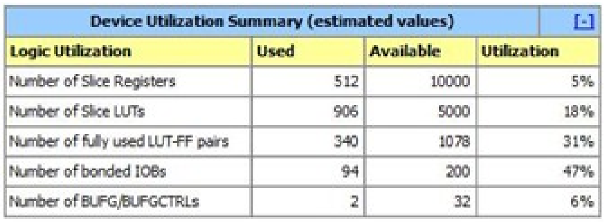


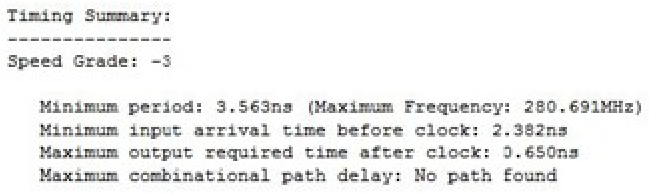
Fig. 6. DAC Verification result

Fourier Transforms, implementation of Fuzzy Logic control using FPGA etc. This IP core design can be easily integrated into any design for data acquisition. It is better than other IP cores due to its programmability.



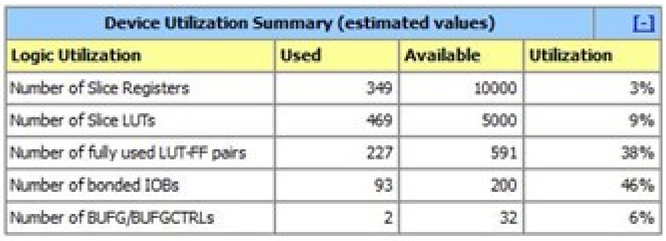
Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	512	10000	5%
Number of Slice LUTs	906	5000	18%
Number of fully used LUT-FF pairs	340	1078	31%
Number of bonded IOBs	94	200	47%
Number of BUFG/BUFGCTRLs	2	32	6%

Fig. 7. Utilization Summary for ADC



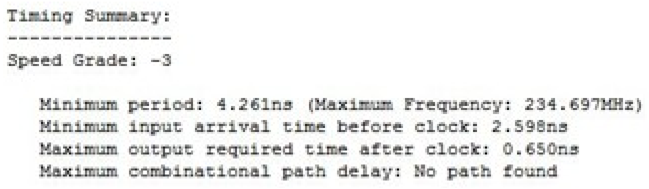
```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 3.563ns (Maximum Frequency: 280.691MHz)
   Minimum input arrival time before clock: 2.382ns
   Maximum output required time after clock: 0.650ns
   Maximum combinational path delay: No path found
```

Fig. 8. Timing Summary for ADC



Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice Registers	349	10000	3%
Number of Slice LUTs	469	5000	9%
Number of fully used LUT-FF pairs	227	591	38%
Number of bonded IOBs	93	200	46%
Number of BUFG/BUFGCTRLs	2	32	6%

Fig. 9. Utilization Summary for DAC



```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 4.261ns (Maximum Frequency: 234.697MHz)
   Minimum input arrival time before clock: 2.598ns
   Maximum output required time after clock: 0.650ns
   Maximum combinational path delay: No path found
```

Fig. 10. Timing Summary for DAC

VII. Conclusion

This design uses VHDL as design language to achieve the interface to ADC and DAC. Using Xilinx ISE software and OVM testbench to complete simulation and test. The results are stable and reliable. The design has great flexibility and high integration. Especially in the field of electronic design, where SOC technology has recently become increasingly mature, this design has great significance.

References

[1] H. Lei, "Design of embedded data acquisition system based on fpga," in Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on, Aug 2009, pp. 530–534.

[2] R. A. Bergamaschi and W. R. Lee, "Designing systems-on-chip using cores," in Proceedings of the 37th Annual Design Automation Conference, ser. DAC '00. New York, NY, USA: ACM, 2000, pp. 420–425. [Online]. Available: http://doi.acm.org/10.1145/337292.337526

[3] J. Weber and M. Chin, "Using fpgas with embedded processors for complete hardware and software systems," in Beam Instrumentation Workshop 2006(AIP Conference Proceedings Volume 868), vol. 868, 2006, pp. 187–192.

[4] S. Toscher, T. Reinemann, R. Kasper, and M. Hartmann, "A reconfigurable delta-sigma adc," in Industrial Electronics, 2006 IEEE International Symposium on, vol. 1, July 2006, pp. 495–499.

[5] C. Sisterna, M. Segura, M. Guzzo, G. Ensinck, and C. Gil, "Fpga implementation of an ultra-high speed adc interface," in Programmable Logic (SPL), 2011 VII Southern Conference on, April 2011, pp. 161–166.

[6] A. K. Oudjida, M. L. Berrandjia, A. Liacha, R. Tiar, K. Tahraoui, and Y. N. Alhoumays, "Design and test of general-purpose spi master/slave ips on opb bus," in Systems Signals and Devices (SSD), 2010 7th International Multi-Conference on, June 2010, pp. 1–6.

[7] On-Chip Peripheral Bus, Architecture Specifications. IBM corp, Copy-right 2011.

[8] O. Cadenas and E. Todorovich, "Experiences applying ovm 2.0 to an 8b/10b rtl design," in Programmable Logic, 2009. SPL. 5th Southern Conference on, April 2009, pp. 1–8.

[9] Q. Li and F. Dong-qin, "Functional verification of epa chip," in Elec-tric Information and Control Engineering (ICEICE), 2011 International Conference on, April 2011, pp. 4216–4220.