Interfacing Analog to Digital Converters to FPGAs

A Lattice Semiconductor White Paper

October 2007

Lattice Semiconductor
5555 Northeast Moore Ct.
Hillsboro, Oregon 97124 USA
Telephone: (503) 268-8000
www.latticesemi.com

Introduction

As the need for data bandwidth increases for end systems, data transmission rates continue to increase for Analog to Digital Converters (ADC) and the associated FPGA solution to interface to the ADCs and other parts of the system. Manufacturers of ADCs and FPGAs have responded with faster, more capable devices at a lower cost.

This white paper will examine two of these fast A/D Converters from National Semiconductor and how to interface each ADC to low-cost FPGAs from Lattice Semiconductor.

ADC Devices

The first class of ADC, ADC14155, that we will examine uses a traditional 14-bit parallel data bus. The second class of ADC, ADC14DS065/080/095/105, that we will examine uses single or dual serial LVDS data lanes. The dual-lane mode allows the device to transfer data at the maximum data rate for a given ADC device.

The ADC14155 is capable of converting analog data into 14 bit words at sample rates up to 155MSPS. It has a separate 1.8 Volt power supply for the digital interface that allows low power operation with reduced noise. The digital outputs operate at CMOS voltage levels and include an over-range indication, data ready strobe, configuration pins and the 14 bit data bus.

The ADC14DS065/080/095/105 also converts analog data into 14 bit words, but it outputs the data on 1 or 2 serial data lines per channel. It can be ordered in various speed ranges from 65 to 105 MSPS and it operates from a single 3.3 Volt power supply. The digital outputs operate at LVCMOS voltage levels except for the serial signals and clock outputs, which are LVDS signals. Theses devices can operate up to 65 MSPS in the single lane mode while the higher data rates operate in the dual lane mode. When operating in the dual-lane mode, each lane operates at half the

data rate to keep the required clock frequencies from being excessive. Using this technique, the FPGA interface can support the highest data rate of 105 MSPS. The FPGA then will combine the two data streams appropriately to create the correct signals.

The serial data bus has some advantages in that it uses less board space for signals, is easier to route the PCB and achieves similar data rates to a parallel interface with less wires for the data bus.

FPGA Devices

The FPGAs chosen to interface to the ADC are 90nm process technology devices. The LatticeECP2™ and LatticeECP2M™ FPGA families are based on 90nm SRAM technology specifically designed for low-cost applications. The LatticeECP2M has the distinction of having high-speed serial (SERDES) channels for many industry standard interfaces such as PCI Express and GbE. The LatticeXP2™, on the other hand, is the industry's first true 90nm non-volatile FPGA and provides many security, programming and low-cost benefits. All of these device families have onboard PLLs for clock manipulation and can support I/O clock rates up to 420 MHz. In addition, they both have the pre-engineered generic DDR I/O interface, which allows capture of data on both clock edges as required for this application.

Interface Between ADC and FPGA

The parallel data bus from the ADC14155 can be connected to the FPGA using an I/O bank configured for 1.8 LVCMOS inputs. The data rate of this bus is 5-155 MHz, which is well within the I/O capabilities of these FPGAs. The ADC14155 requires a clock input signal and will generate a data ready signal to send back to the FPGA. This data ready signal is the clock signal that is used to read in the data and is aligned to the data stream by the ADC. The data is output at the falling edge of the data ready signal so that latching the data at the rising edge should provide

acceptable set-up and hold times provided good board layout practices are used. The data output uses standard single data rate (SDR) clocking.

The serial data bus from the ADC14DS065/080/095/105 can be connected to the FPGA using an I/O bank configured for LVDS differential inputs. The FPGA would use the low-skew edge clock and the pre-engineered generic DDR I/O interface to capture the data on both the rising and falling edges of the clock signal. The ADC requires a clock input signal. The ADC will send an OUTCLK signal synchronized to the data stream using its on-chip DLL to adjust the phase of the clock as required. The ADC also sends out a FRAME signal to let the user know when the start of each data frame occurs. This FRAME signal is aligned with the data by the ADC but may need to be shifted so that the setup and hold time requirements of the data are met correctly. This can be done using the PLL or the DLL of the LatticeECP2/M FPGA. The LatticeXP2 FPGA would use the PLL to shift the FRAME signal since it does not have a DLL on chip.

For accessing higher data rates, the ADC14DS065/080/095/105 uses a dual-lane mode. In the dual lane mode the first data sample is sent out on lane 1 and the second data sample is sent out on lane 0, with subsequent samples continuing to alternate lanes. This allows the higher sample rates to be supported while keeping the clock output rates lower to simplify interfacing to an FPGA or other device. The ADC14DS065/080/095/105 data output always uses double data rate (DDR) clocking.

FPGA Design Example for Parallel ADC Interface

A sample design was created for the ADC14155 to verify the performance of the FPGA. For this ADC the maximum data rate for each parallel data line is 155 mbps, which is well within the capabilities of the LatticeECP2/M FPGA. The DRDY signal from the ADC will be running at 155 MHz for the maximum sample rate of 155 MSPS. The DRDY signal is brought into a primary clock input pin (PCLK) and assigned to use primary clock resources. The DRDY signal is aligned to the data by the ADC

such that the data is presented at the falling edge of DRDY. Therefore the data can be captured at the rising edge of DRDY in the FPGA and the setup time can be met without any need to adjust the phase of this clock signal.

A block diagram of the connections between the LatticeECP2/M FPGA and the ADC14155 for the sample design is shown below:

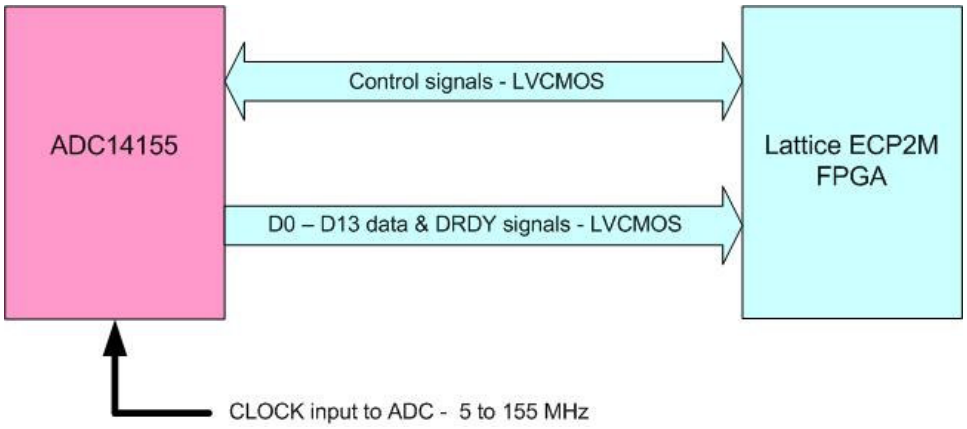

Figure 1 – Block Diagram of ADC Parallel Connections to the Lattice FPGA

If desired, the FPGA can also be used to perform pre-processing of the data to off-load this task from another DSP processor. The challenge is to see if the pre-processing can be accomplished at the full data rate. For this sample application, the input data was captured in a shift register with 8 elements. Then an average was calculated for the 8 samples and an engineering unit conversion was done on the average. Using the timing reporting capabilities of Lattice's ispLEVER® software design tools, it is possible to estimate the maximum operating frequency of the design while reading the data and performing this pre-processing. The results for several different examples are shown below.

Calculation Type	Results
Average 4 samples and convert to engineering units	166 MHz
Average 4 samples without conversion	223 MHz
Average 8 samples and convert to engineering units	128 MHz
Average 8 samples without conversion	176 MHz

Operating Frequency Reported for the Sample Design (using –7 speed grade FPGA device)

From the table results it can be observed that push button FPGA design flow can generally meet the 155 MHz operating speed of the ADC. These calculations were not optimized, and better results can likely be achieved by making use of the advanced capabilities of the DSP blocks within the LatticeECP2/M and LatticeXP2 FPGAs.

FPGA Design Example for Serial ADC Interface

A sample design was also created for the ADC14DS065/080/095/105 to verify the FPGA performance. This design uses the single lane data mode with a 60 MSPS sample rate. This provides the maximum input clock rate to the FPGA and demonstrates the techniques required to capture the data at this rate. The dual lane mode will operate at a lower input clock rate to the FPGA and should be easier to implement, with the exception of having to swap the data streams. The dual lane operation can be set up using the Word-Aligned mode or the Offset mode. In the Word-Aligned mode the data words from both lanes are captured on the rising edge of the FRAME signal. The Offset mode captures the data word of lane 1 on the rising edge of FRAME and the data word of lane 0 on the falling edge of the FRAME signal.

A block diagram of the connections between the LatticeECP2/M FPGA and the ADC for the sample design is shown below:

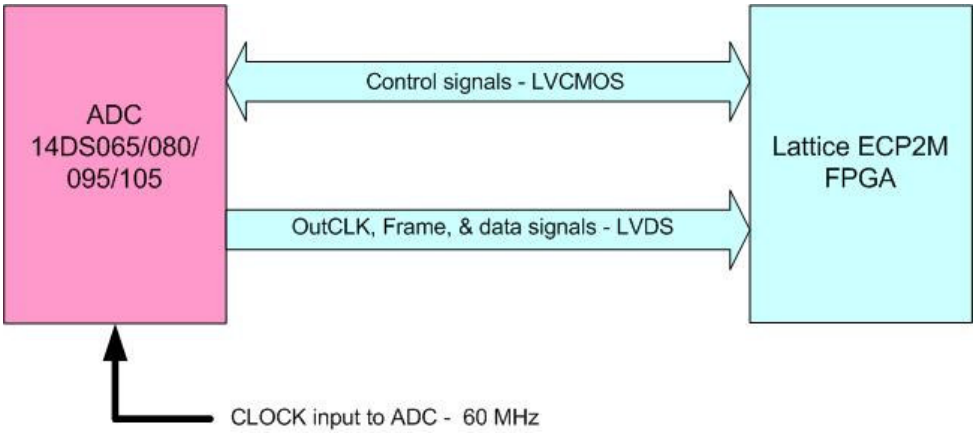

Figure 2 – Block Diagram of Serial ADC Connections to the Lattice FPGA

The sample design uses two shift registers to capture the data as it comes in from the ADC. The first one is used to capture the data on the positive edge of the OUTCLK signal and the second captures the data on the falling edge of OUTCLK. The pre-engineered generic DDR I/O interface could be used in place of this logic to capture the data on both edges of the clock. The user would then need a second clock running at half the rate of the OUTCLK signal to transfer the data into the shift registers from the DDR registers, 2 bits at a time.

The OUTCLK signal is specified as an Edge clock in order to handle the high-speed clock coming from the ADC. For the 60 MSPS data rate in the single lane mode, the clock rate would be 420 MHZ coming from the ADC. The edge clock in the LatticeECP2/M and LatticeXP2 FPGAs can handle clock rates up to 420 MHZ.

The data from the shift registers is then transferred into the data word register upon the FRAME signal rising edge. This data word register is stored in an EBR memory block for reading out later. The Memory is designed to capture the first 512 samples for reading out later to verify the operation of the design. A pseudo Dual Port RAM module is implemented using EBR memory to store the data. This module is generated using the IPexpress™ tool of Lattice's ispLEVER design software.

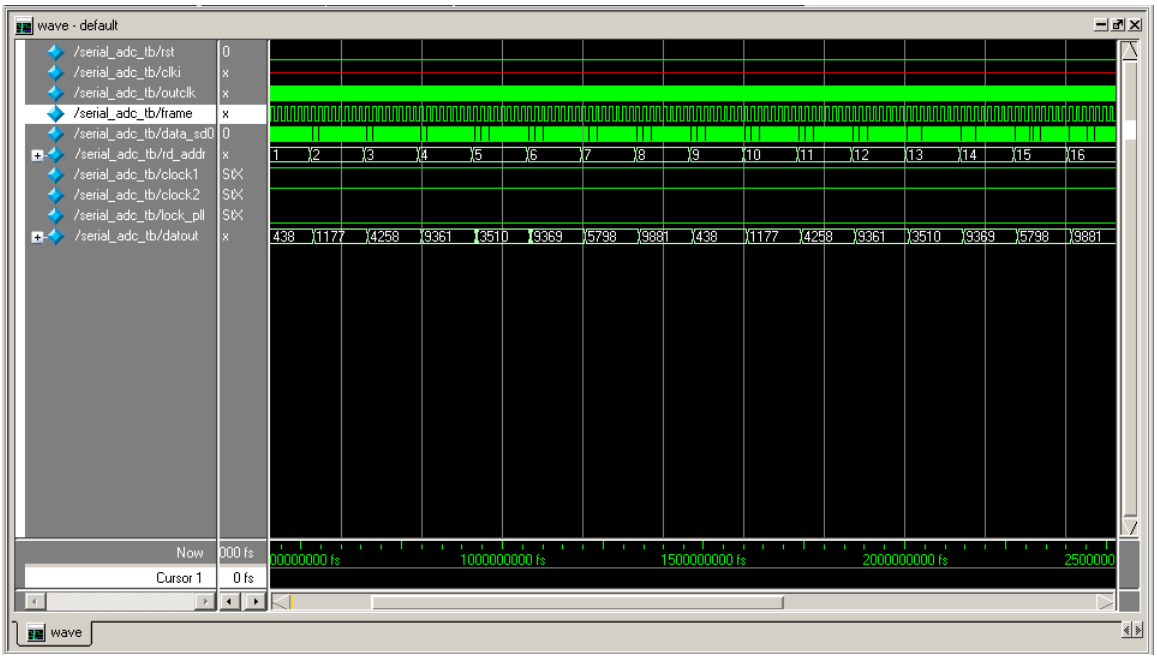

Figure 3 - Post-Route Timing Simulation Results Showing a Repeating Data Pattern

In order to verify this sample design, a simulation was run using the ModelSim tools provided with ispLEVER. These results are shown in Figures 3 and 4. In Figure 3, the repeating data pattern used as an input can be seen as the value shown for the dataout port. The dataout port is connected to the read port of the pseudo Dual Port RAM module. The data is read out of memory when the read address is incremented in the test-bench used for this simulation.

Figure 4 shows the timing relationship between the data input and the OUTCLK and FRAME signals. Note that the FRAME signal has been shifted in the test-bench to the position shown. This shows the relationship that is required at the maximum clock rate to read the data accurately. In actual practice, a PLL or DLL would be used to shift the FRAME signal to this alignment. Please see the ADC14DS065/080/095/105 datasheet for the FRAME signal alignment that is output from the ADC. The left cursor marks the beginning of a data frame. The right cursor marks the beginning of the next data frame. There are seven OUTCLK periods in the data frame for the 14 bits of data. The total data frame time shown is for an OUTCLK rate of 417 MHz, which corresponds to a 59.5 MSPS data rate in one lane mode.

The input signal from the ADC is “data_sd0” and it can be seen that the value in this frame is: 10 0100 1001 1001 This corresponds to a value of 9369, which is the sixth value in the sequence as shown in figure 3.

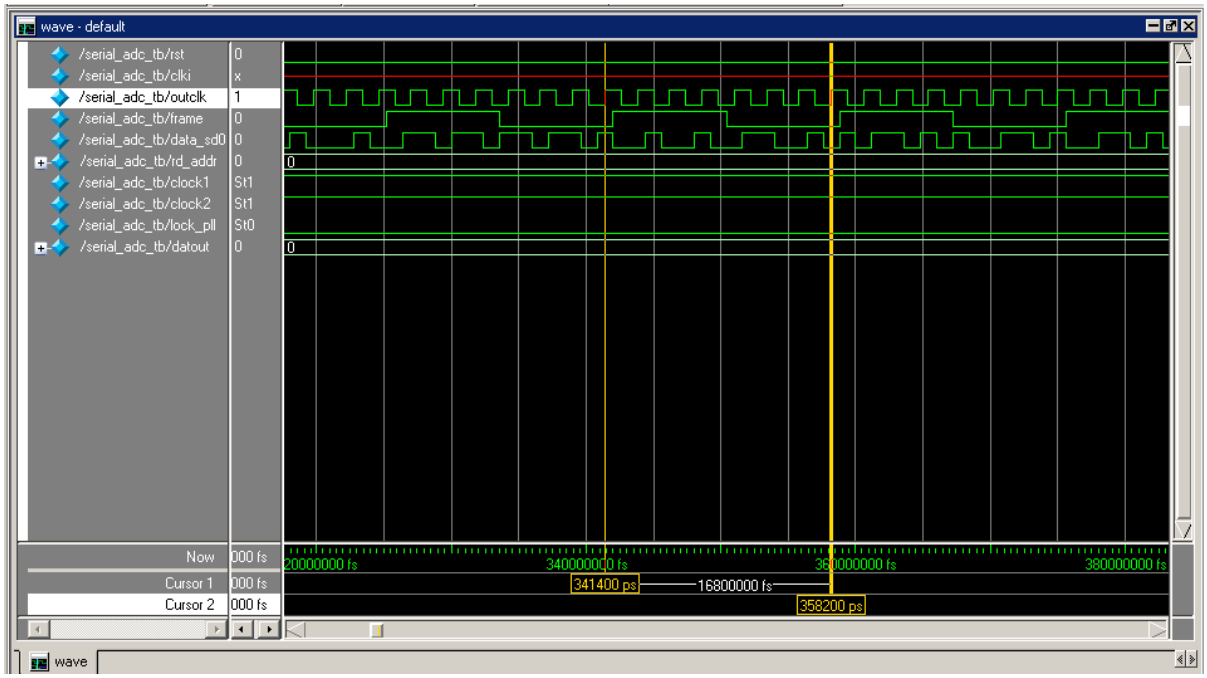

Figure 4 - Post-Route Timing Simulation Results Showing a Data Input Pattern

System Applications Using FPGAs and ADC

The high performance ADCs discussed in this white paper can be found in many different applications, including:

- High IF Sampling Receivers
- Wireless Base Station Receivers
- Radar Systems
- Power Amplifier Linearization
- Test and Measurement Equipment
- Multi-carrier Multi-mode Receivers
- Communications Instrumentation
- Portable Instrumentation

Summary

The LatticeECP2/M and LatticeXP2 FPGAs are ideally suited for use with the National Semiconductor ADC chips. The LatticeECP2/M and LatticeXP2 are low cost FPGAs with fast I/O and advanced capabilities that readily interface with the high-speed data signals that are output from these high performance, analog-to-digital converters.

Appendix

The Verilog code of the sample design for the serial ADC interface is shown below:

```
// 14 bit Data input from Analog to digital converter - serial single channel

module serial(rst,outclk,frame,data_sd0,datout,rd_addr);


input  rst, outclk, frame ;
input  data_sd0 ;

output [13:0]datout ;

input  [9:0]rd_addr ;

reg    [13:0] data_in ;
reg    [13:0] datareg0 ;
reg    [9:0] wr_addr ;
reg    wr_enable ;


/* Verilog module instantiation template generated by SCUBA ispLever_v70_Prod_Build
(55) */
/* Module Version: 4.1 */
/* Wed Aug 15 11:34:57 2007 */

/* parameterized module instance */
RAM_dp RAM_u1 (.WrAddress(wr_addr), .RdAddress(rd_addr), .Data(datareg0),
.RdClock(frame), .RdClockEn(1'b1), .Reset(rst), .WrClock(frame), .WrClockEn(1'b1),
.WE(wr_enable), .Q(datout));


//   Shift register to capture the serial data from data_sd0 port on positive edge
always @(posedge outclk or posedge rst)
       begin
          if (rst)
              begin
              data_in[13] = 1'b0 ;
              data_in[11] = 1'b0 ;
              data_in[9]  = 1'b0 ;
              data_in[7]  = 1'b0 ;
              data_in[5]  = 1'b0 ;
              data_in[3]  = 1'b0 ;
              data_in[1]  = 1'b0 ;
              end
```

```
          else
              begin
              data_in[13] = data_in[11] ;       //  shift register to accumulate
                                                //  data bits
              data_in[11] = data_in[9] ;
              data_in[9]  = data_in[7] ;
              data_in[7]  = data_in[5] ;
              data_in[5]  = data_in[3] ;
              data_in[3]  = data_in[1] ;
              data_in[1]  = data_sd0 ;          //  add new data bit into shift
                                                //  register
              end
       end

//   Shift register to capture the serial data from data_sd0 port on negative edge
always @(negedge outclk or posedge rst)
       begin
          if (rst)
              begin
              data_in[12] = 1'b0 ;
              data_in[10] = 1'b0 ;
              data_in[8]  = 1'b0 ;
              data_in[6]  = 1'b0 ;
              data_in[4]  = 1'b0 ;
              data_in[2]  = 1'b0 ;
              data_in[0]  = 1'b0 ;
              end
          else
              begin
              data_in[12] = data_in[10] ;       //  shift register to accumulate
                                                // data bits
              data_in[10] = data_in[8] ;
              data_in[8]  = data_in[6] ;
              data_in[6]  = data_in[4] ;
              data_in[4]  = data_in[2] ;
              data_in[2]  = data_in[0] ;
              data_in[0]  = data_sd0   ;        //  add new data bit into shift
                                                //  register
              end
       end

//   data capture at frame signal
always @(posedge frame or posedge rst)
       begin
          if (rst)
              begin
              datareg0 = 14'b00000000000000 ;
       wr_addr  = 9'b000000000 ;         //  initialize the memory write address
                                         //  counter
              wr_enable = 1'b1 ;         //  initialize the memory write enable
                                         //  signal
              end
          else
              begin
              datareg0 = data_in ;       //  add new word of data into data register

              if (wr_enable)
                  wr_addr = wr_addr + 1 ;       //  increment the address counter
              else
                  wr_addr = wr_addr ;    //  hold the address counter at this value
              if (wr_addr[9])
                  wr_enable = 1'b0 ;     //  disable write enable when address MSB =
1
```

```
                                        //  to prevent overwriting data in memory
            else
                wr_enable = 1'b1 ;

            end
        end

endmodule
```