



## Virtex Analog to Digital Converter

XAPP155 September 23, 1999 (Version 1.1)

Application Note: John Logue

### Summary

When digital systems are used in real-world applications, it is often necessary to convert an analog voltage level to a binary number. The value of this number is directly or inversely proportional to the voltage. The analog to digital converter (ADC) described here uses a Virtex FPGA, an analog comparator, and a few resistors and capacitors. An 8-bit ADC can be implemented in about 16 Virtex CLBs, and a 10-bit ADC requires about 19 CLBs.

**Xilinx Family:** Virtex Family

### Introduction

This ADC uses the Delta-Sigma Digital to Analog converter (DAC) described in Xilinx Application Note XAPP 154. The DAC is used to generate a reference voltage for the negative input to the external comparator. The analog signal feeds the positive input of the comparator (see Figure 1). The voltage range of the DAC output is 0V to  $V_{CC0}$ , so that is also the range of analog voltages that may be converted. (If the analog input voltage is not always between 0V and  $V_{CC0}$ , either the DAC output or the analog signal itself may be biased, attenuated, or amplified with external components to achieve the desired voltage range compatibility.)

The analog level is determined by performing a serial binary voltage search, starting at the middle of the voltage range. For each complete sample, only the upper bit of the DAC input is initially set, which drives the reference voltage to midrange. Depending on the output of the comparator, the upper bit is then cleared or it remains set, and the next most significant bit of the DAC input is set. This process continues for each bit of the DAC input.

The DAC is one bit wider than the ADC output. This is required in order for the lowest numbered bit of the ADC output to be significant. When all of the bits have been sampled, the upper bits of the register feeding the DAC is transferred to the ADC output register.

Because of the serial nature of both the DAC and the analog sampling process, this ADC is useful only on signals that are changing at a fairly low rate. A typical high-precision application for this ADC would be monitoring a physical metric, such as ambient air temperature or water pressure. It can also be used for applications with lower precision at a higher sampling rate, such as utility voice recording (for example, a telephone answering machine).

If the analog input voltage changes during the sampling process, it effectively causes the sample point to randomly move. This adds a noise component that becomes larger as the input frequency increases. For many applications, the strength of the additional noise will be so low that it will be acceptable. This noise component can be removed with an external sample and hold circuit for the analog input signal.

Figure 1 shows how the ADC is connected to the other components. As described in the DAC application note, a 24 mA LVTTTL output buffer is normally used to drive the RC filter. Most comparators have uncommitted collector/drain outputs, so  $R_P$  is usually needed.

The interface to module `adc` in Figure 1 includes four output and four input signals, which are defined in Table 1. All signals are active high.

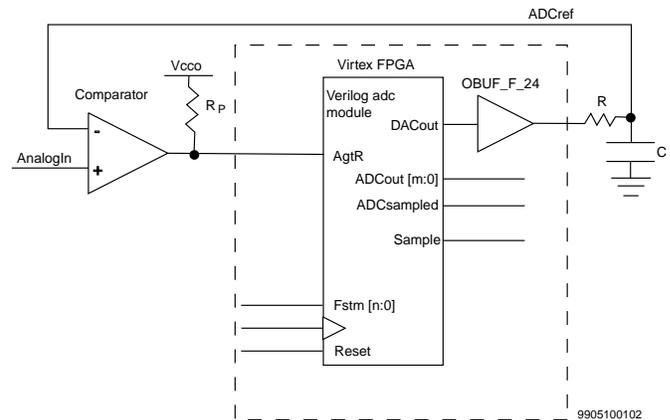


Figure 1: ADC Symbol, RC Filter, and Comparator

Table 1: ADC Interface Signals

Signal	Direction	Description
DACout	Output	Pulse string that drives the external low pass filter. See DAC application note for more information. (XAPP154)
ADCout[m:0]	Output	ADC output bus.
ADCsampled	Output	Clock enable for the DACout register. It indicates that DACout will change on the next positive Clk edge, if AnalogIn has changed.
Sample	Output	Sample/Hold. Signal is True when ADC is sampling upper bit of each word. Can drive an external Sample and Hold circuit.
AgtR	Input	Analog greater than Reference. This is the output of external comparator.
Fstm[n:0]	Input	Filter Settle Time Multiplier. Static value that allows the user to control bit sample time based on the low pass filter settle time requirements. The bit sample time is multiplied by (Fstm+1).
Clk	Input	Positive edge clock for the internal registers.
Reset	Input	Initializes internal registers.



A comparator has a functional range (the "Common Mode Limits") that is determined in part by the power supply voltages used. In the above circuit, the LM319 is supplied with ground (0V), and +5V. This gives it a functional range of about 1.0V to 3.8V. Note from Figure 5 that  $V_T$  is within these boundaries when the temperature range is from 0 to +40°C.

These limits reduce the usable digital range (to about 178 to 391 in this example). By avoiding high and low DAC limits, one also avoids the worst case inputs to the RC filter, which reduces the noise on signal ADCref. (Note that the thermistor used in this example has a functional range of -40 to +100°C. The temperature range is limited primarily by the comparator common mode limits.)

Another key comparator parameter is the sensitivity, or Input Offset Voltage. This is the voltage difference between the positive and negative inputs required to switch the output. The LM319 has a worst case Input Offset Voltage of 8.0 mV. The Input Offset Voltage must be less than the DAC step voltage, minus peak-to-peak noise.

It is important that the thermistor voltage,  $V_S$ , be from the same source as the OBUF\_F\_24  $V_{CCO}$ . When this is the case, variations on  $V_{CCO}$  are "common mode", and are thus self-canceling.

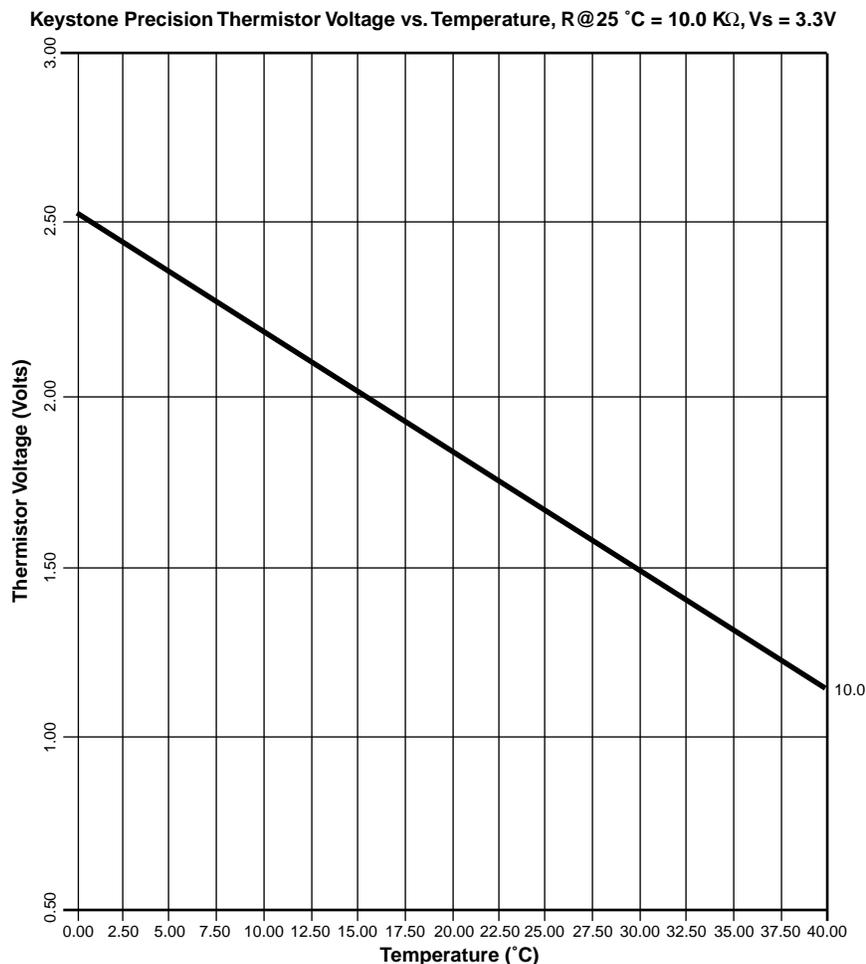
The thermistor used in this example has an accuracy of 0.2°C without calibration. A precision level to match this accuracy can be maintained with a 9-bit ADC. Thermistor noise reduction capacitor  $C_F$  is optional. It may not be necessary if wire lengths are short.

See "Appendix B - ADC Top Level Verilog Code - File ADCtop.v" on page 8 which shows an example of top-level Verilog code for the ADC

that includes a DLL, 24 mA RC filter driver, etc. Note that constants MSBI and MSBR are defined in this module, so these definitions should be removed from adc.v.

## Conclusion

This analog to digital converter is another example of how high speed FPGAs may be used in mixed-signal systems to minimize the number of components. When circuitry is moved inside an FPGA that already exists in the system, a reduction in component cost is realized as well. Also, FPGAs provide a level of in-field reconfigurability that cannot be achieved in any other way. The speed and density of the Virtex family of FPGAs make them ideal for a wide range of analog signal generating and processing applications.

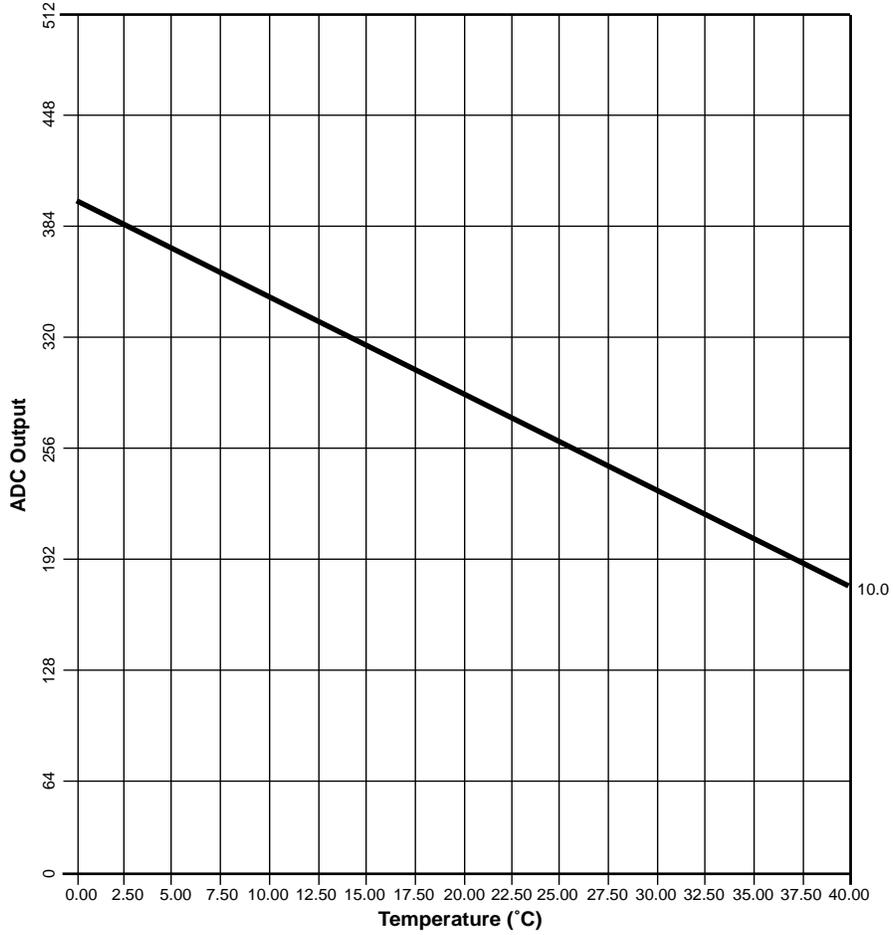


The number at the end of each curve is the Source resistance in  $\text{K}\Omega$

9904300101

**Figure 4: Thermistor Temperature vs. Voltage V**

Thermistor ADC Out vs. Temperature, R@25°C = 10.0 KΩ, Vs = Vcco = 3.3 V, DAC width = 9



The number at the end of each curve is the Source resistance in KΩ.

9904300202

Figure 5: Thermistor Temperature vs. ADC Output

## Appendix A - ADC Verilog Implementation

```

`include "dac.v"

`timescale 100 ps / 10 ps
// This is an Analog to Digital Converter that uses an external
// analog comparator which compares the input voltage to a
// voltage generated by the built-in DAC. The DAC voltage starts at midrange,
// and continues to narrow down the result by successive division by 2
// until the DAC LSB is reached. This requires 'MSBI+1 (the width of
// the DAC input) samples. The width of the result is 1 bit less than the
// width of the DAC. The precision of the result is within 1/2 LSB.

module adc (ADCout, DACout, ADCsampled, Sample, Fstm, AgtR, Clk, Reset) ;

output [ `MSBI-1:0 ] ADCout; // ADC result register
output DACout; // This is the DAC average output that feeds low pass filter
output ADCsampled; // Status signal, CE for ADCout
output Sample; // Sample Analog signal when True; Hold when False

input [ `MSBR:0 ] Fstm;// Input constant. Sample time is multiplied by Fstm+1
input AgtR;// Comparator output; analog greater than Reference
input Clk;
input Reset;

reg [ `MSBR:0 ] DACsampleCntr;// DAC sample counter
reg DACsampled; // Pipelining DAC Sampled status
reg DecFstmCntr; // Pipeline DecFstmCntr
reg [ `MSBR:0] FstmCntr; // Low Pass Filter settle time counter
reg Shift; // Pipeline Shift signal
reg [ `MSBI:0] M; // Mask Shifter
reg [ `MSBI:0] RefShftr; // Reference Shifter (DAC input)
reg ADCsampled; // Pipeline ADCsampled status
reg [ `MSBI-1:0] ADCout; // ADC result buffer
reg Sample; // Sample Analog signal when True; Hold when False

// DAC sample counter. Same width as DACin. Counts up continuously.
// One cycle of this counter is the minimum time it takes for the
// DAC output to resolve to DACin.
always @ (posedge Clk or posedge Reset)
begin
    if (Reset) DACsampleCntr <= #1 0;
    else DACsampleCntr <= #1 DACsampleCntr + 1;
end

// pipeline DACsampled status for speed
always @ (posedge Clk or posedge Reset)
begin
    if (Reset) DACsampled <= #1 1 'b0;
    else DACsampled <= #1 (DACsampleCntr == 0);
end

```

```

end

// pipeline      DecFstmCntr signal so it is in phase with Shift Signal
always @ (posedge Clk or posedge Reset)
begin
    if (Reset)      DecFstmCntr <=  #1 1 ' b0;
    else            DecFstmCntr <=  #1 DACsampled;
end

// Filter settle time multiplier. This counter counts down to 0 from Fstm.
// When this counter reaches 0, the output of the comparator is sampled,
// so the DAC output reference voltage must be stable at that time.
// Fstm may be set to the lowest value that ensures a stable
// reference voltage, based on the low pass filter parameters.
always @ (posedge Clk or posedge Reset)
begin
    If (Reset)      FstmCntr <=  #1 1;
    else
        begin
            if (Shift)      FstmCntr <=  #1 Fstm;
            else if (DecFstmCntr) FstmCntr <=  #1 FstmCntr - 1;
        end
end

end

wire      FstmCntrEq0 = (FstmCntr == 0);

// pipeline Shift signal for speed
always @(posedge Clk or posedge Reset)
begin
    if (Reset) Shift <= #1 1 'b0;
    else      Shift <= #1 (DACsampled & FstmCntrEq0);
end

end

// pipeline ADCsampled status for speed
always @(posedge Clk or posedge Reset)
begin
    if (Reset) ADCsampled <= #1 1 ' b0;
    else      ADCsampled <= #1 (DACsampled & FstmCntrEq0 & M [ 0 ] ) ;
end

end

// Mask shifter. Same width as DACin. A single bit rotates right endlessly.
// This bit specifies the next bit that will be sampled.
always @(posedge Clk or posedge Reset)
begin
    if (Reset)  M <=  #1 0;
    else
        if (Shift)
            begin
                if (M [ 1 ] | ~( |M ) ) M <=  #1 1; // Guarantees 1 and only 1 bit set
                else M <=  #1 {M [ 0 ], M[ `MSBI :1 ]}; // shift right
            end
    end
end

```

```

end
end

// Reference shifter. This shifter feeds DACin. Starts with only upper bit set.
// Shifts right, forcing next bit set, but conditionally clearing previous
// bit based on AgtR.
always @(posedge Clk or posedge Reset)
begin
    if (Reset)                RefShftr <= #1 0;
    else if (Shift)           RefShftr <= #1
        {M[0], M [ `MSBI:1] } | (RefShftr & ~{( `MSBI+1) {M[0]}} & ~(M & ~{( `MSBI+1) {AgtR}}));
end

// ADCout register. Snaps upper bits of Refshiftr when sample is justified.
always @(posedge Clk or posedge Reset)
begin
    if (Reset)                ADCout <= #1 0;
    else if (ADCsampled)      ADCout <= #1 RefShftr [ `MSBI:1] ;
end

// Sample Latch
always @(posedge Clk or posedge Reset)
begin
    if (Reset)                Sample <= #1 0;
    else
        if (Shift)
            begin
                if (M [0])      Sample <= #1 1 'b1;
                else            Sample <= #1 1 'b0;
            end
end
end

// Instantiate DAC
dac DC (.DACout (DACout),
        .DACin (RefShiftr),
        .Clk (Clk),
        .Reset (Reset));

endmodule

```

## Appendix B - ADC Top Level Verilog Code - File ADCtop.v

```

`define MSBI          9          // Most Significant Bit of Dac input
`define MSBR          3          // Most Significant Bit of Filter Settle Time Multiplier

`include "adc.v"

`timescale 100 ps / 10 ps // This is a top level implementation module for adc

module ADCtop (ADCout,          DACoutDrvr, ADCsampled, SampleDrvr, DLLlocked, ClkTp,
              Fstm, AgtR, ClkIn, Reset);

output [ `MSBI-1:0] ADCout;          // ADC result output buffer register
output DACoutDrvr;                  // This is the DAC output that feeds low pass filter
output ADCsampled;                  // Status signal, CE for ADCout register
output SampleDrvr;                  // Sample Analog signal when True; Hold when False
output DLLlocked;                   // DLL locked
output ClkTP;                       // 2X clock test point

input [ `MSBR:0] Fstm;              // Input constant. Sample time is multiplied by Fstm+1
input AgtR;                          // external comparator output; analog greater than
Reference
input ClkIn;
input Reset;

IBUFG clk_pad (.I (ClkIn), .O (clk_w));

CLKDLL DLL2X (.CLKIN(clk_w), .CLKFB(Clk), .RST(Reset), .CLK0 ()
              .CLK90(), .CLK180(), .CLK270(), .CLK2X(Clk2X), .CLKDV(),
              .LOCKED (DLLlocked));

BUFG CLKG (.I (Clk2X), .O (Clk));
OBUF_F_24 CLKO (.I (Clk), .O (ClkTP)); // CLK2X test point driver

OBUF_F_24 AB (.O (DACoutDrvr) , .I (DACout)) ; // DACout 24ma Driver

OBUF_S_24 SD (.O (SampleDrvr), .I (Sample)); // Sample 24 mA Driver

adc AD (.ADCout (ADCout), .DACout (DACout), .ADCsampled (ADCsampled), .Sample (Sample),
        .Fstm (Fstm), .AgtR (AgtR), .Clk (Clk), .Reset (Reset));

endmodule
    
```

## Revision History

Date	Revision #	Activity
7/26/99	1.0	Initial Release
9/23/99	1.1	Updated for Virtex-E devices