ECE 128 – VerilogA Lab: Creating & simulating an 8-bit A-D Converter (ADC)

Adapted for GWU by: Thomas Farmer

Objectives:

- To create a sample 8-bit ADC in VerilogA for use with 8-bit CPU Project
- Familiarize student with Cadence VerilogA code generator known as the ModelWriter
- Simulate an 8-bit ADC using Spectre

Assumptions:

- Student has working knowledge of A/D converter
- Student understand verilog-A

Note:

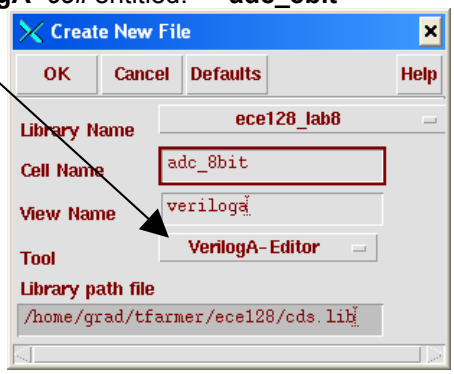
- This tutorial was created using Cadence 5.41, but it has been tested and works properly in Cadence 6.1. There are some basic menu differences, but if a student is familiar with Cadence 6.1, this should be no problem for the student.

SECTION I: Creating the ADC

1. Log into a Sun Workstation.

2. Start Cadence by typing these commands:
   $ cd cadence
   $ virtuoso &

3. Create a new library entitled: **ece128_lab_8 (Attach to .6 AMI)**
4. Create a new "**verilogA**" *cell* entitled: **adc_8bit**



5. When the editor opens, delete the existing lines of text and copy & paste this code for an 8-bit ADC converter below:

```
//     FUNCTION: Analog to Digital Converter
//      VERSION: $Revision: 2.12 $
//       AUTHOR: Cadence Design Systems, Inc.
//
// GENERATED BY: Cadence Modelwriter 2.31
//           ON: Fri Mar 07 09:37:38 EST 2008
//
// Description: Ideal Analog to Digital Converter
//   Generates an N bit ADC.
//     - selectable logic output levels
//     - model valid for negative values of vmin
//     - adjustable conversion time, and rise/fall time
//    This model is an example, provided "as is" without express or
//    implied warranty and with no claim as to its suitability for
//    any purpose.
//
// PARAMETERS:
//   slack = The smallest time interval considered negligible for
// cross event on clock [S]
//   tconv = Delay from threshold crossing to output change [S]
//   trise = Rise time for digital output signals [S]
//   trise = Rise time for digital output signals [S]
//    vmax = ADC Full scale output voltage [V]
//    vmin = ADC Zero scale output voltage [V]
//    vone = The voltage of a logical 1 on digital outputs [V]
//     vth = Threshold value of clock signal [V]
//   vzero = The voltage of a logical 0 on digital outputs [V]
//

`include  "discipline.h"
`include  "constants.h"
`define NUM_ADC_BITS   8

module adc_8bit (vin, clk, data);
 input   vin, clk;
 electrical vin, clk;

 output  [`NUM_ADC_BITS-1:0] data;
 electrical [`NUM_ADC_BITS-1:0] data;

  parameter real  vmax = 5;
  parameter real  vmin = 0;
  parameter real   one = 5.0;
  parameter real  zero = 0.0;
  parameter real   vth = 2.5;
  parameter real slack = 0.5p from (0:inf);
  parameter real trise = 1.0p from (0:inf);
  parameter real tfall = 1.0p from (0:inf);
  parameter real tconv = 0.5p from [0:inf);
  parameter integer traceflag = 1;

     real   sample, vref, lsb, voffset;
     real   vd[0:`NUM_ADC_BITS-1];
     integer ii, binvalue;

    analog begin
      @(initial_step or initial_step("dc", "ac", "tran", "xf"))  begin
        vref = (vmax - vmin) / 2.0;
        lsb  = (vmax - vmin) / (1 << `NUM_ADC_BITS) ;
        voffset = vmin;

        if (traceflag)
        $display("%M ADC  range ( %g v ) /  %d bits  = lsb %g volts.\n",
                    vmax - vmin, `NUM_ADC_BITS, lsb );

        generate i ( `NUM_ADC_BITS-1, 0) begin
            vd[i] = 0 ;
        end
    end
```

```
      @(cross ( V(clk)-vth,  1, slack, clk.potential.abstol)) begin
          binvalue = 0;
          sample = V(vin) - voffset;
          for ( ii = `NUM_ADC_BITS -1 ; ii>=0 ; ii = ii -1 ) begin
            vd[ii] = 0;
            if (sample > vref ) begin
              vd[ii] = one;
              sample = sample - vref;
              binvalue = binvalue + ( 1 << ii );
            end
            else begin
             vd[ii] = zero;
            end
            sample = sample * 2.0;
          end
          if (traceflag)
            $strobe("%M at %g sec. digital out: %d   vin: %g  (d2a: %g)\n",
                      $abstime, binvalue,  V(vin), (binvalue*lsb)+voffset);
      end

      generate i ( `NUM_ADC_BITS-1, 0) begin
         V(data[i])  <+   transition ( vd[i] , tconv, trise, tfall );
          end
    end
endmodule

`undef NUM_ADC_BITS
```

---

6. After the above code has been entered, save and close the editor window. Cadence will syntax check the verilogA code that has been entered. Fix any errors that may have occurred due to syntax. Note: Any time a change is made, cadence will automatically syntax check the code.

7. You will be prompted to create a symbol for this device, answer yes and save the symbol.