LUND
UNIVERSITY

Electrical and Information Technology

Integrated A/D and D/A Converters (ETIN55)

Lab3 Manual

Martin Anderson
Pietro Andreani

February/October 2021

Lab 3 – Mixed-Signal Modeling and Simulation with AMS

This lab will introduce you to the procedure of modelling and simulating mixed mode circuits (a circuit with both analog and digital sub-circuits). A flash ADC and a pipelined ADC will be modeled and simulated. It will serve two purposes. First, you will get familiar with mixed mode simulations and modeling. Second, you will familiarize with two different A/D converter architectures.

Analog components will be modeled using the hardware describing language VerilogA. Digital circuits will be modeled using Verilog. Analog circuit simulators are accurate but slow, whereas digital circuit simulators are quite fast, but less accurate (since it only needs to represent the signal as a 0 or a 1). In simulation of large mixed signal systems, a problem arises if you try to simulate the digital circuits in an analog circuit simulator, because it will take endless amounts of time. Yet, for many mixed signal systems, the noise and distortion performance is due to imperfections of analog circuit elements. Thus, the accuracy of the analog circuit simulators is needed in order to get correct simulation results for the mixed signal system.

Cadence provides the powerful AMS (= analog mixed signal) simulator, where the design is partitioned into two domains, one analog and one digital, while the interface signals between the two domains are of a mixed nature. The analog circuits are simulated using an analog circuit simulator (spectre), while the digital circuits are simulated using a digital circuit simulator (VerilogXL). The circuit partitioning into analog and digital domains, as well as the most prominent features of AMS, are introduced in this lab.

Laboratory report

Include all plotted simulation results, the schematics and answers to all questions in your laboratory report.

Homework exercises

1. Read the text on high speed ADCs [Maloberti, p.147-157 and p.184-199], and search the internet as well as the corresponding lecture notes for information about flash ADCs and pipelined ADCs.
2. Review your knowledge of Cadence (schematic drawing, analog and digital simulations) by reading the lab manuals and related material for the **Analog IC Design** and the **Digital IC Design** courses.
3. Read this lab instruction and the corresponding part of the lab seminar slides. Make sure that you understand how the circuit models work, especially the **veriloga** and **verilog** codes.
4. Calculate the number of comparators needed for a 5-bit flash ADC.
5. Explain why flash A/D converters are rarely used for more than 6/7-bit resolution.
6. What is the maximum offset voltage V_o for the comparators in order to guarantee that no codes are missing in a 5-bit flash ADC if the input signal range is ± 500mV?
7. If the input signal swing of a pipelined ADC is ± 500mV, what are the reference voltage levels to be used in the 1.5b ADSC1p5Bit cell? What signal levels are needed at the output of the 1.5 bit DAC?
8. Assume a resistor string like the one in Figure 13 is used to generate the reference voltages for ADSC1p5Bit. What should the resistor size ratio **R0:R1:R2** be?
   (V_R=± 500mV)
9. With a $1V_{pp}$ input signal, calculate the accuracy requirement of the first stage sampling, addition and multiplication to achieve (N - 0.5) bit accuracy in a pipelined ADC.

Laboratory exercises

Part 1: Modeling and simulating a 5-bit flash ADC

A flash ADC consists of 3 major building blocks: a reference voltage generator, a number of comparators, and a thermometer-to-binary (or 2's complement) encoder. Every clock cycle, the input signal Vin is compared to the reference voltages. By calculating the number of ones at the comparator outputs, the digital word corresponding to the analog input is encoded. A simplified block view is shown in Figure 1.

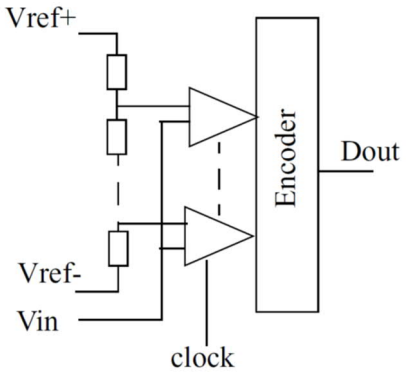

Figure 1. Flash ADC.

1. Preliminary steps:
    a. Log into a UNIX workstation.
    b. Create a Cadence directory in your home directory, for example CDSLAB, and descend into it:

    ```
    > mkdir CDSLAB
    > cd CDSLAB
    ```

    c. Download the zip file `AMSLab.zip` from the course homepage, and unzip it in your Cadence directory:

    ```
    > unzip AMSLab.zip
    ```

    `AMSLab.zip` contains *ConnRules_1V2.vams* (a file defining the interface between analog and digital blocks in the mixed-signal design), and *AMSLab*, the Cadence library to use in the lab.

    d. If you are working on a UNIX workstation in the undergraduate lab or in the research lab: while in your Cadence directory, start **Cadence**:

    ```
    > inittde adda21
    > virtuoso &
    ```

    e. Select *Tools > Library Manager...* in the **Virtuoso** window to open the *Library Manager* window if it did not start automatically. Add the **AMSLab** library to your library path in the *Library Manager* with *Edit > Library Path...* Save your changes to the library path when done.
2. We start by creating a **veriloga** model of the comparator to be used in the flash ADC:
    a. Open the **veriloga** view of the comparator **Comp**.

b. In the *emacs* editor window that pops up, edit the **veriloga** description of the comparator, i.e. replace the **???** with your own code. Do not forget to model the comparator offset, **voffset**. Make sure that the comparator output is high when **(ip-in) > 0**, and low when **(ip-in) < 0**.

c. To compile the code, save it and close the window. If there are errors, you will be asked if you want to correct them. Answer *yes*... After editing and correcting all errors, save the file and quit.

3. Simulation of the comparator: since it is a **veriloga** description, *spectre* can be used:

a. To simulate the **veriloga** description of the comparator, a test bench shall be used. Open the schematic view of the cell **CompTb** containing the test bench. The process is the same as for any analog circuit, and the result is shown in Figure 2. Use a **10MHz** clock generating a square wave between **0V** and **1.2V**, let the signal at the positive comparator input be a **1MHz** sine wave with **500mV** peak amplitude and **0V** DC value, and set the other input to **0V**. Let the logic levels be **0V** and **1.2V**. Create now a **config** view for **CompTb** with *File* > *New* > *Cellview…* and by choosing **config** in the *Type* field. Click on *OK*, and the *New Configuration* window pops up. Click on *Use Template* at the window bottom and select *spectre* in the scroll-down menu of the pop-up window. Click on *OK* and then again *OK*. Click on the *Save* icon in the *Hierarchy Editor* window. Close both **schematic** and **config** windows.

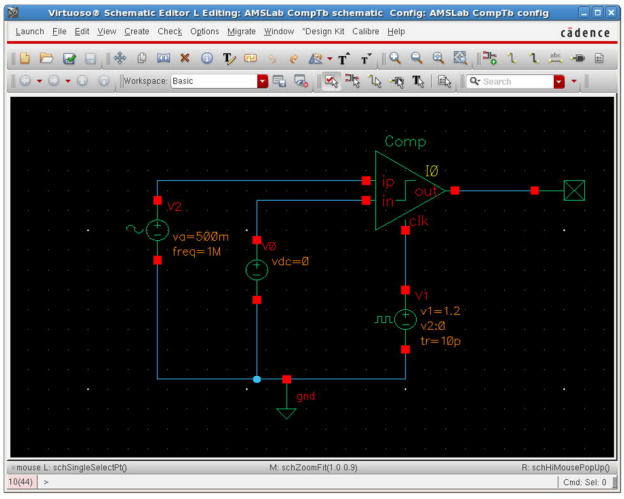

Figure 2. The test bench of the comparator, CompTb.

b. Double-click on the **config** view in the *Library Manager*, and select **schematic** and deselect **config** in the pop-up window. The **schematic** view that opens up is configured by the **config** view, which decides what views for the various instances contained in the test bench are going to be used. The title of the **schematic** window should be something like that in Figure 2: *Virtuoso ® Schematic Editor L Editing: AMSLab CompTb schematic Config: AMSLab CompTb config.*

c. Click on *Launch* > *ADE L* and run a **2µs** transient analysis with **moderate** accuracy and visualize the relevant waveforms. Check that the comparator works as intended.

4. Build the mixed-mode flash ADC model, completing the schematic view of a 5-bit flash ADC with both analog and digital components:

a. Complete the 5-bit flash ADC design found in the schematic view of the cell **FlashADC**, using the analog blocks **Comp** and **RefLadder**, as well as the digital encoder **Term2Bin5Bit** (see Figure 3). **Term2Bin5Bit** assumes that the thermometric signal is made of zeros at the bottom and ones at the top. To drive the encoder, use buffers **HS65_GS_BFX2**, found in the library **CORE65GPSVT**, supplied by **STMicroelectronics**. **Term2Bin5Bit** is a **verilog HDL** (**functional**) description of a thermometer-to-binary encoder. Open the code and make sure that you understand what it does. **RefLadder** is a resistor ladder used to generate all reference voltages for the comparators.

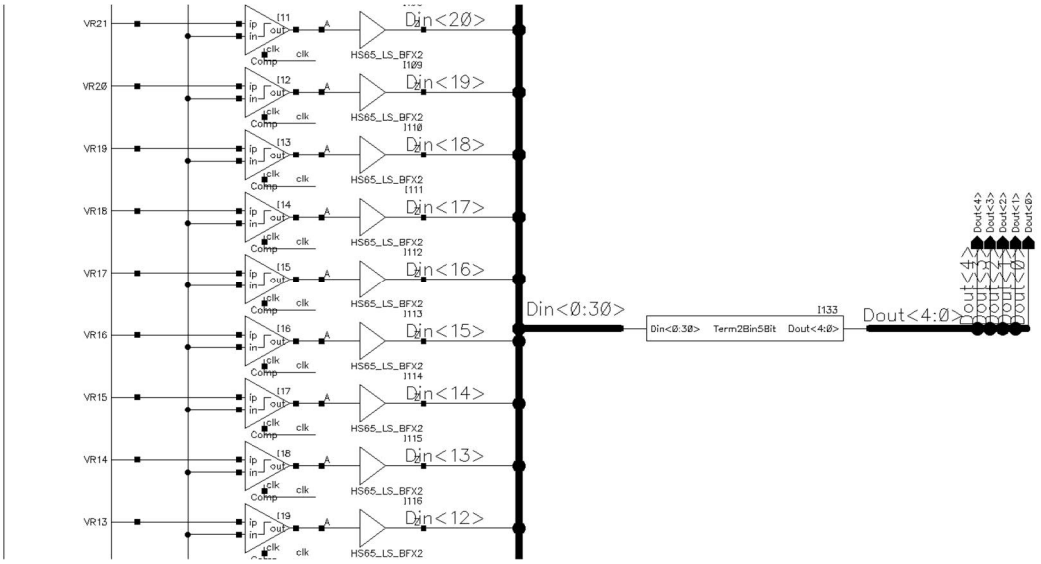

Figure 3. Partial schematic view of the flash ADC, FlashADC.

b. To reduce the complexity of the routing of digital signals, bus notation such as **Dout<0:30>** can be used. Select *Create > Wire Name….* In the pop-up **Add Wire Name** window shown in Figure 4, bus names are added when **Bus Expansion** is **off**, and individual wire names are added when **Bus Expansion** is **on**. You can also place all individual wire names on the respective wires by selecting **Placement** as **multiple**. Choose **Bus Expansion on** and **Placement multiple** to quickly assign **Dout<30>, Dout<29> ... Dout<0>** to the respective nets.

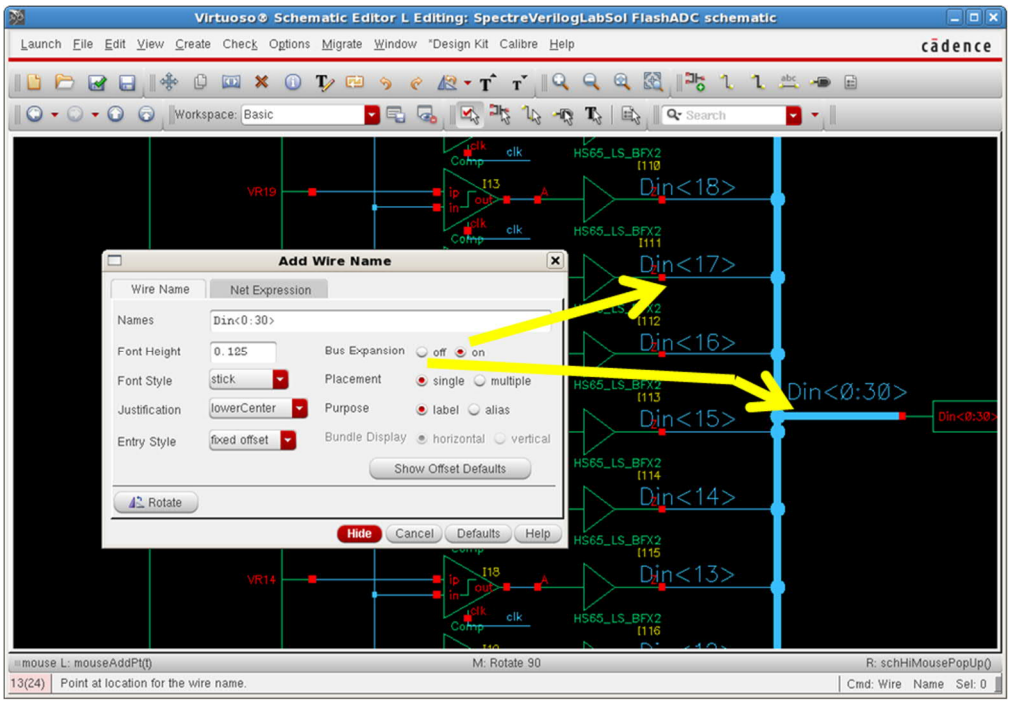

Figure 4. Illustration of bus notation.

5. Simulation of the flash ADC: build a test bench for the flash ADC and perform a mixed-mode simulation of its **config** view:

    a. Open the test bench **FlashADCTb** schematic. Add the **FlashADC** instance to it, as well as the **veriloga** description of the 5-bit DAC **DAC5Bit** from your **AMSLab** library. **DAC5Bit** converts the digital output signal back into an analog voltage.

    b. You are going to use a ramp signal as ADC input, to perform a (quite primitive) test on missing codes for the flash ADC. Every quantization level should be exercised exactly once, if the ADC has no missing codes. The test setup is arranged so that the input signal is sampled when the input signal is exactly between two quantization levels (there is a 1/2 LSB offset on the input). In this test, the difference between the output signal and input signal should never be more than 1 LSB (if the offset is removed). It should be noted that this test only guarantees no missing codes. For the actual linearity and noise performance, dynamic testing techniques should be used (which is done later in this assignment). However, thanks to the low complexity of this testing technique, it is a good sanity check even for schematic-level models of ADCs. Use the ramp input specified in the test bench schematic. Let the logic levels be **0V** and **1.2V**, and the input signal between **±500mV**. The top and bottom reference voltages to the resistor ladder should have the same peak signal range as the input signal (see Figure 1). The test bench should look something like Figure 5. Save and close the test bench schematic.

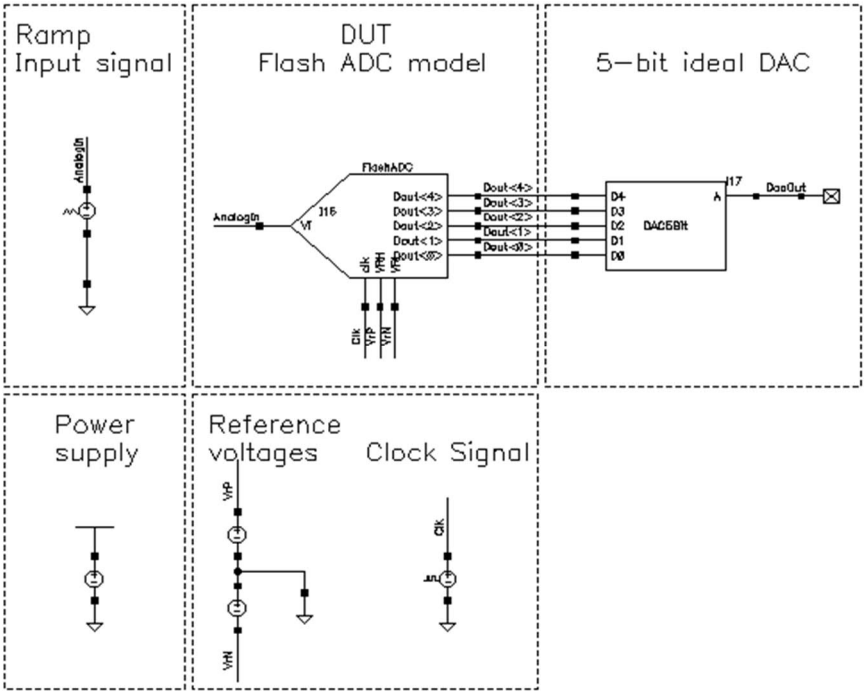

Figure 5. The flash ADC test-bench schematic view, FlashADCTb.

c. Create a **config** view for the **FlashADCTb** instance. Click on **Use Template** and select **AMS;** select the **Top Cell View** to be **schematic** if it is not already so.

d. If the **AMS View Found** of **Term2Bin5Bit** is **NONE**, right-click on it and select *Set Cell View > functional*. Different views for the various cells can be selected in the same way. For **HS65_GS_BFX2** select the **verilog** view, and save **config** (notice that if you select the **cmos_sch** view for **HS65_GS_BFX2** and save **config**, the **SimMosfetStandard** view of the MOS devices present in **HS65_GS_BFX2** will appear).

e. In the **config** view, add **CORE65GPSVT** in the **Library List** and **verilog** in the **View List** (in the **Global Bindings** sub-window) and save. It is possible to check or edit the value of the constants **$simlevel**, **$stop** etc. in the **View List** and **Stop List** by selecting *Edit > Constants…* The **config** view should be similar to the one in Figure 6.

f. Click on the **Tree View** to have a look at the tree view of the design. Here it is also possible to indicate what view should be used for different instances of the same cell: in Figure 7, the view to use for instance **I102** of **HS65_GS_BFX2** has been changed to **cmos_sch**, and all other kept to **verilog**.

g. Click on the **Open** button to open the **schematic** view managed by the **config** view.

h. Launch the **ADE L** environment from the **schematic** window. In the **ADE L** window, go to *Setup > Simulator/Directory/Host…* and set the simulator to **ams** (you may have got a warning about this). Click on *Setup > Connect Rules/IE Setup….* In the pop-up window, enable the option **Connect Rules/ Connect Module Based Setup**

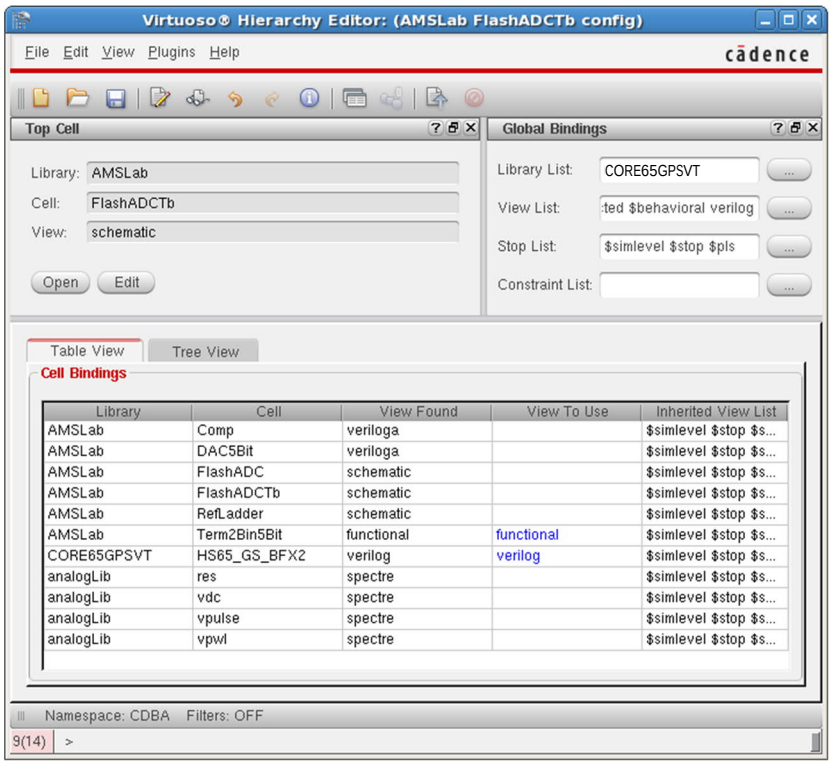

Figure 6. The flash ADC test bench config view (**Table View**).

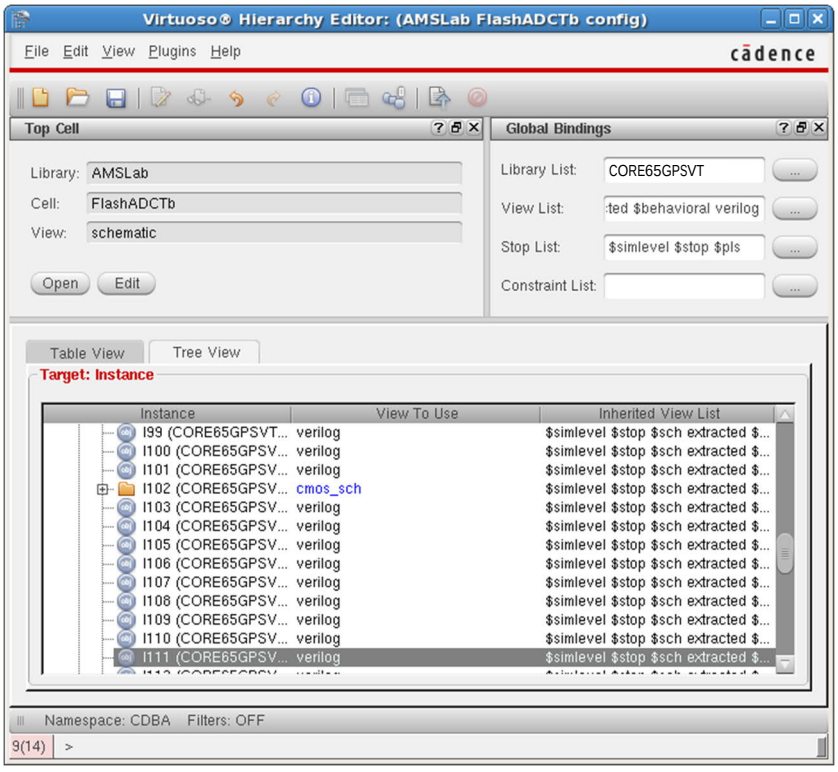

Figure 7. The flash ADC test bench config view (**Tree View**).

i. Click on **Browse** near the field **Connect Rules/Modules File**, and select the file **ConnRules_1V2.vams** that was provided to you. Write **ConnRules_1V2_fast** in the **Rules Names**, and click on **Add** and then **OK**. This file contains the rules (threshold voltages and other parameters) to convert an analog signal into a valid digital signal and *vice versa*. This is of fundamental importance in a mixed-signal simulation.

j. Still in **ADE L**, click on *ArtistKit* > *Setup Corners…* In the pop-up window, select **<all typical>** in the tab under GLOBAL VARIATIONS, **no** in the tab under LOCAL VARIATIONS, and **DK** in the tab under LIBRARY. Click on **Save Model File** at the top of the window and close the window.

k. Still in **ADE L**, select *Outputs* > *Save all* … and select **all** for *Save nets* and **all** for *Levels of hierarchy to save*, both under *NETS* and *CONNECT MODULES*. Click on *OK*.

l. Select *Simulation* > *Options* > *Analog(Spectre)* … and set **reltol** to **1e-5**.

m. Run a **3.7µs transient analysis** with **moderate** accuracy and plot the difference between the ADC input signal and the D/A output for your lab report. If the simulator complains about netlisting errors, try saving the **config** view one more time. If the curves look discontinuous, it is because there are visualized in a sample-and-hold fashion. To force a continuous plot, select the curve, right-click on it and select *Type* > *Continuous line*.

n. If you want to get rid of possible "glitches" in the DAC output, set to zero the parameters **trise**, **tfall** and **tdel** in all comparators and in **DAC5Bit**. To do this, select e.g. all comparators, press the **q** key (= **Edit Object Properties**) and choose **veriloga** in the **CDF Parameter of view** field; check that the **Apply to field** is set to **all selected**.

o. In the **schematic** window, run *AMS* > *Display Partition* > *Initialize* (in fact, we would like to take this step before running the simulation, but the simulation database is actually needed to look at the signal partition). Check the different signal domains (analog, digital, mixed) with *AMS* > *Display Partition* > *Hilight All*. Analog blocks/wires are colored green, digital are turquoise, analog-to-digital are yellow, and digital-to-analog are orange. Identify the nature of the various signals, and understand why they are what they are.

p. Select now the 6 top comparators and change their **voffest** parameter to **100mV**. Redo the simulation and see the difference compared to your initial offset-free simulation. Save the results for your lab report.

   Question: why is comparator offset important in the design of flash A/D converters?

q. In the **config** view of **FlashADCTb**, change the view for **HS65_GS_BFX2** from **verilog** to **cmos_sch** and rerun the simulation. Do you notice any difference? How does the signal partitioning change?

Part 2: Modeling and simulating an 8-bit pipeline ADC

In this part, you will design an 8-bit pipeline ADC built with 1.5-bit pipeline stages, using the knowledge of mixed-mode simulations and modeling that you gained in part 1.

A pipeline ADC consists of a series of pipeline stages. In the tracking phase, each stage tracks its input signal. A digital approximation of the input signal is made using a 1.5-bit A/D sub-converter (**ADSC1p5Bit**). Thereafter the digital approximation is converted back to analog form by a DAC (**DAC1p5Bit**), and subtracted from the held input signal to form the quantization error (residue) signal for the next stage. To regain the full signal amplitude, the residue voltage is amplified by two. A block schematic of a pipeline stage is shown in Figure 11.

In this lab you will model a pipeline stage, and some of its circuit elements. Thereafter, you will evaluate its performance and study some non-ideal behavior that may occur in a circuit implementation of this A/D converter model.

6. Design the pipeline ADC as follows:
    a. Open **PipeADC schematic**. Instantiate six pipeline stages and connect them together.
    b. Instantiate the shift register (**ShiftReg**) and the digital correction unit (**FullAdder**).
    c. To make sure that there is a valid input signal for every pipeline stage, the clock must be skewed between adjacent stages. The tracking phase of every pipeline stage has to be overlapped by the holding phase of the previous pipeline stage. Connect the clocks **CP1** and **CP2** according to Figure 8.
    d. To synchronize the digitally corrected output, a register array should be used after the **FullAdder**. Use an array of the standard cell **HS65_GS_DFPQX4** from **CORE65GPSVT**, as shown in Figure 9. Make sure that you are using the correct clock in the register array.
    e. When you have connected all blocks and pins together, the **schematic** view of **PipeADC** should look something like Figure 10. Save and close it.

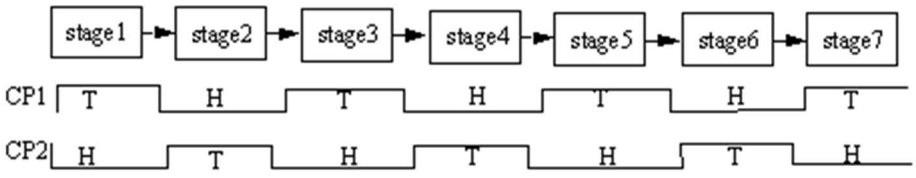

Figure 8. Clock skew between adjacent stages.

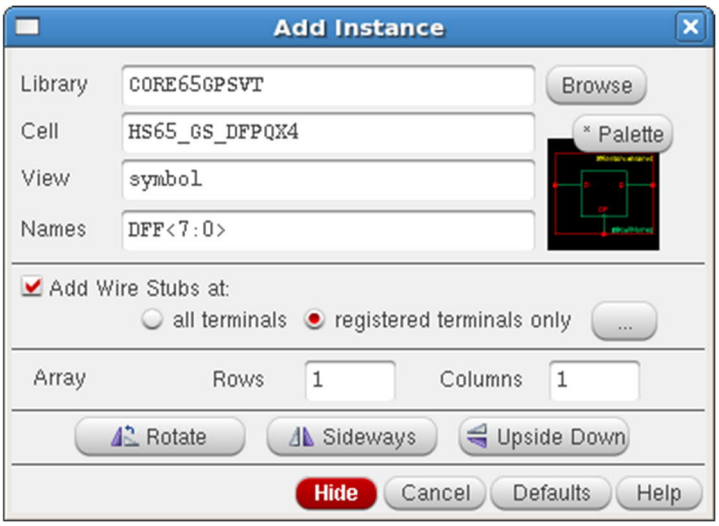

Figure 9. Adding an array of components.

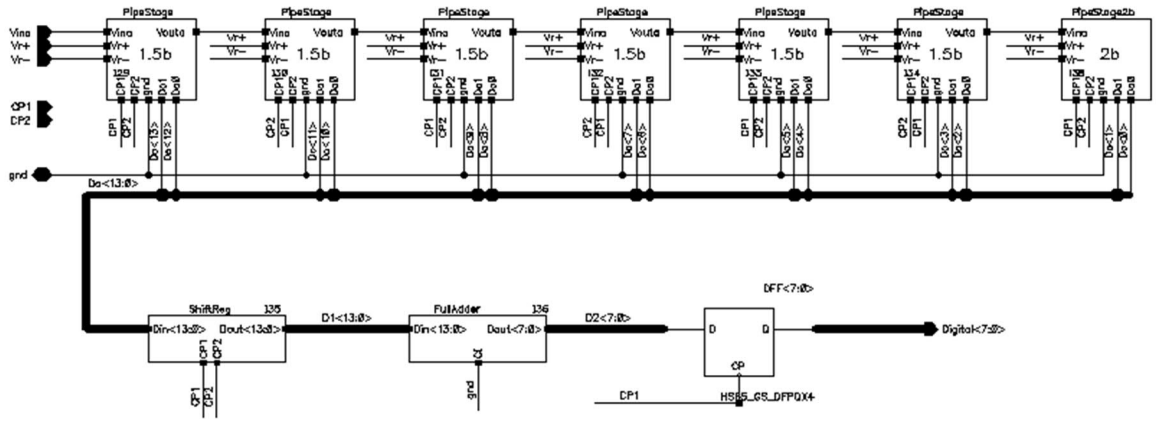

Figure 10. The pipeline ADC model.

7. Design the pipeline stage as follows:

   a. Open the **schematic** view of the pipeline stage **PipeStage**. Instantiate all the required blocks and arrange them as in Figure 11. Connect the pins. Save and close the schematic.

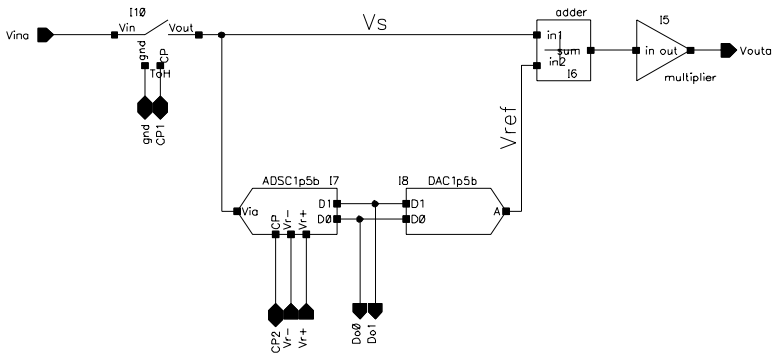

Figure 11. The 1.5-bit pipeline stage, PipeStage.

8. Design the blocks belonging to the pipeline stage. To reduce the workload, some schematics and symbols have been completed for you. It will be your task to model the contents of every missing circuit element using **veriloga** or some circuital components from **analogLib**. Open each block and make sure that you know its task.

    a. The track-and-hold **ToH** function. Arrange the schematic like in Figure 12. The components (**switch** and **cap**) can be found in **analogLib**.

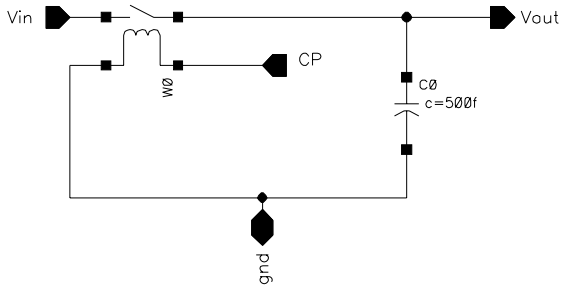

Figure 12. Track-and-hold circuit.

    b. The 1.5-bit ADC **ADSC1p5Bit**. Complete it according to Figure 13. Use the comparators from the flash ADC with reasonable rise and fall times. Use the resistor values from the homework exercise. Draw all wires so that **ADSC1p5Bit** performs correctly.

    c. The 1.5-b DAC **DAC1p5Bit**. Open the **veriloga** view of **DAC1p5Bit**. Complete the code so that the correct analog output is created depending on the digital input signals **D0** and **D1**. Replace **???** with either **+Vref/2**, **0**, or **-Vref/2**. Save and close.

    d. The **adder** and the **multiplier**. Open the **veriloga** view of the adder. The code should realize an addition of the input voltages. The **veriloga** description of the multiplier shall multiply the signal voltage from the adder by two.

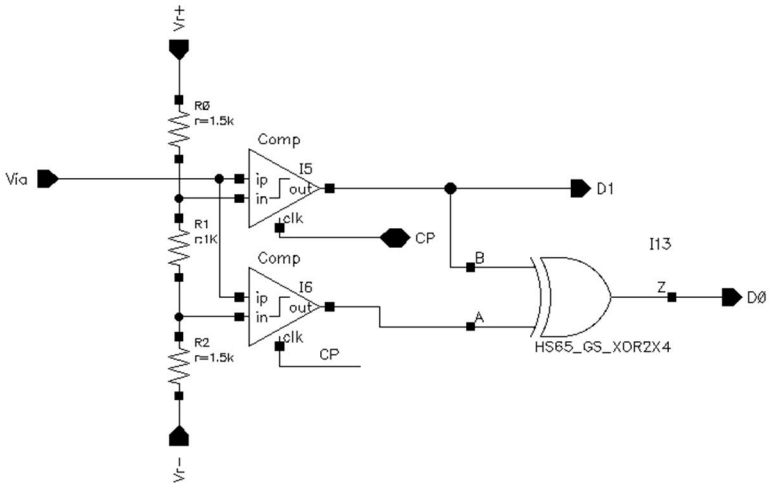

Figure 13. The 1.5-bit ADC, ADSC1p5Bit.

9. The **digital skew** and **code correction** units. Because of the pipeline structure, the ADC delivers a serial output code. The digital correction however, requires parallel inputs (i.e., aligned in time). A shift register has to be used, see Figure 14. The shift register **ShiftReg** and the digital correction unit **FullAdder** have already been designed for you.

    a. Open the full adder **FullAdder** schematic, which implements the digital correction unit. **FullAdder** has been designed using the standard cell **HS65_GS_FA1X4** from **CORE65GPSVT**, as shown in Figure 15.

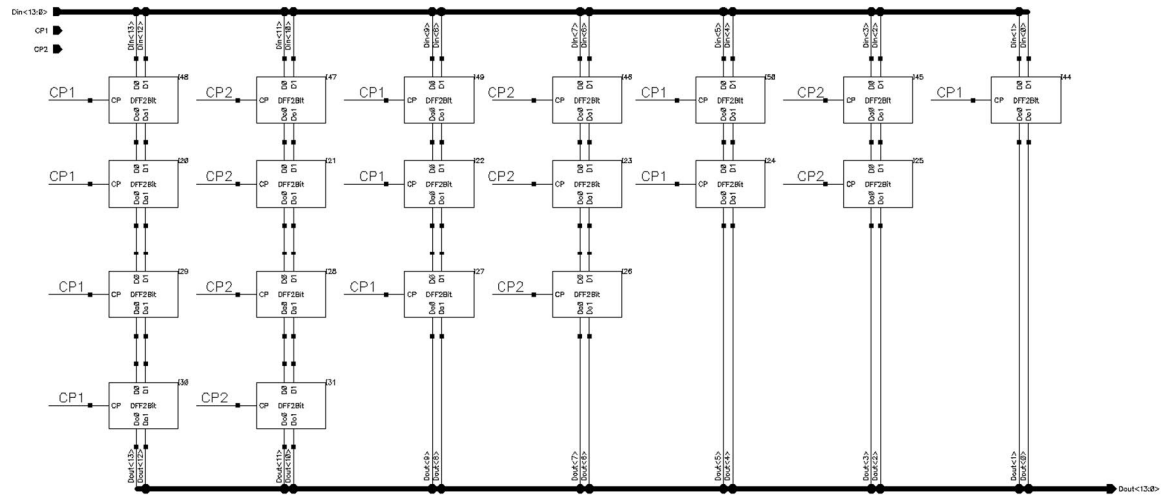

Figure 14. The shift register.

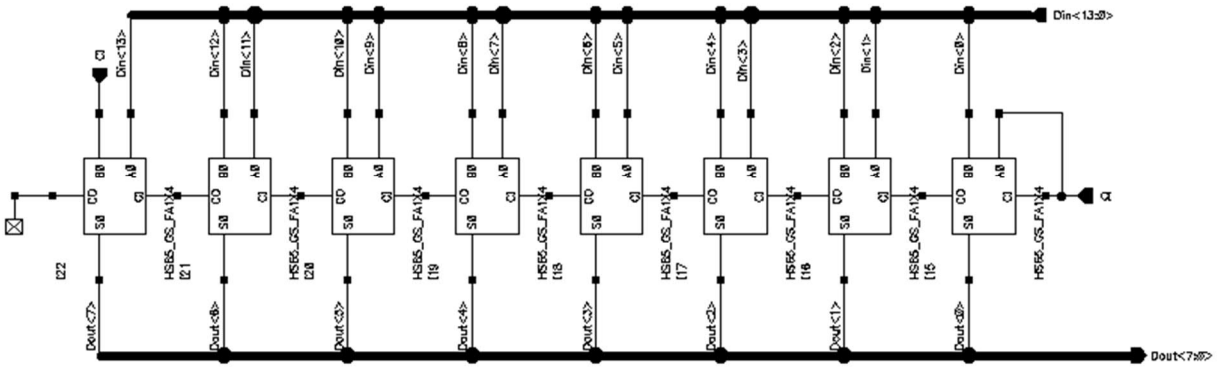

Figure 15. The full adder (digital correction unit).

10. You have now finished modeling the pipeline ADC, and it is time to see if it works correctly.

    a. Create a **config** view for the **PipeADCTb**. Click on **Use Template** and select **AMS;** select the **Top Cell View** to be **schematic** if it is not already so.

    b. Change the view of all digital cells from **cmos_sch** to **verilog**.

    c. Repeat steps **#5.e** to **#5.l** of the previous simulation. The **config** view should be similar to the one in Figure 16.

    d. To avoid filling your disk quota with simulation data, write */var/tmp/* in *Setup* > *Simulator/Directory/Host…* > *Project Directory* in the **ADE L** window.

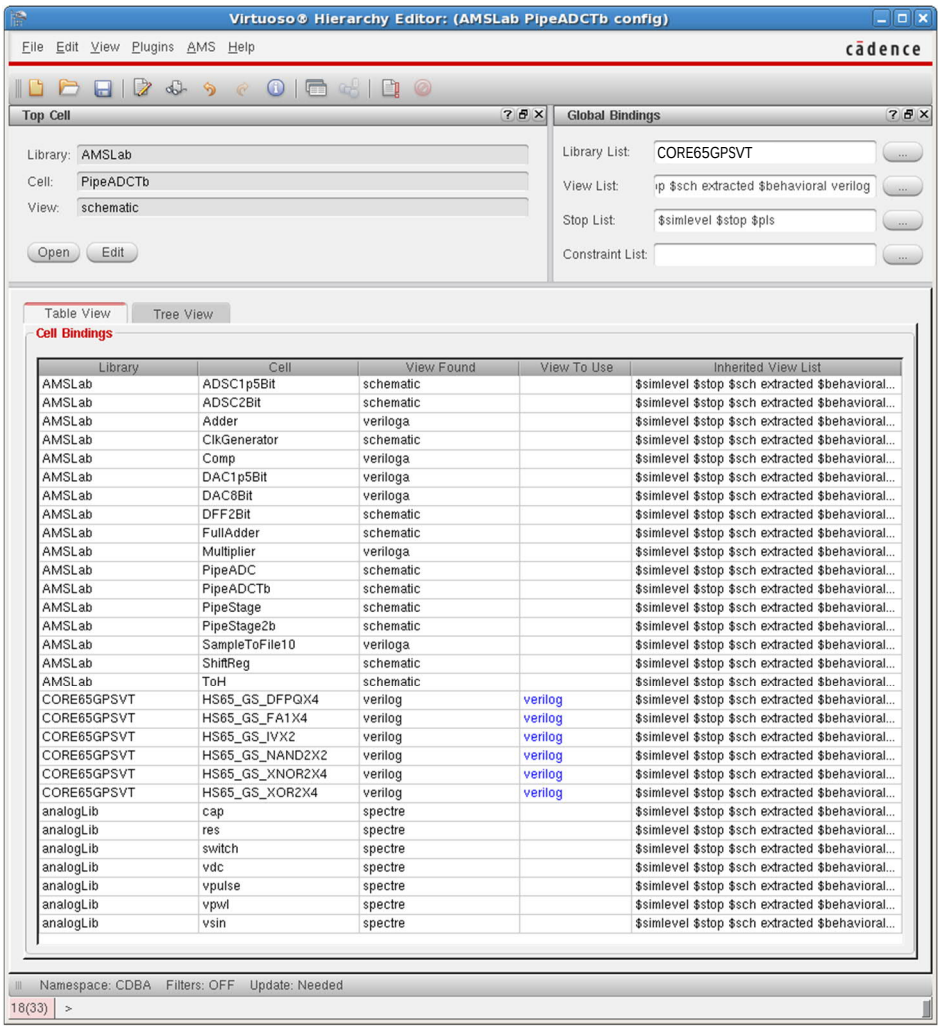

Figure 16. PipeADCTb config setup.

e. Use the **ramp** input signal (generated by the **vpwl** source from **analogLib**) to perform a monotonicity test. The DC voltages **VrP** and **VrN** (generated by two **vdc** sources) to **DAC8Bit** set the full-scale output range of the DAC. The full-scale range of the DAC is the same as for the ADC. Run a **135µs transient** simulation with **moderate** accuracy. Make sure that the output tracks the input of the ADC by plotting the difference between the analog input and the DAC output. Is the result what you expected? Check that the latency between output and input is the expected one.

f. Also in this case you can check the partitioning of the various signal with *AMS > Display Partition > Hilight All*.

g. Use now the **cmos_sch** view for some of the digital cells (avoid the flip-flops though, as they can give convergence problems at start-up), and check how this impacts on the simulation speed (if an error message is generated by the **config** view, click on **OK** and proceed). When you are done, change back the view to **verilog**.

11. Single-tone test. You shall now simulate the dynamic performance of the ADC. The performance described by SNDR, SFDR, HD2 and HD3 is going to be simulated in both the ideal case, and with non-ideal effects.

a. Use a **sine wave** input with $\mathbf{f_{sig} = x/N \times f_s}$, where **x** is an odd number, **N=1024**, and $\mathbf{f_s}$**=2MHz**. The amplitude shall be **490mV**. Add a small DC offset **(< 10mV)** to the sine wave, in order to avoid repetitive patterns in the quantization error.
Question: Why? What happens with this offset?

b. Specify a path and a filename for your saved data in the properties of **SampleToFile10** by selecting **CDF Parameter of view** as **veriloga** and writing the full path and filename for the data file in the **fname** field. Put your path and filename within citation marks (e.g., *"/full-path/filename"*).

c. Run a **520µs transient** simulation with **moderate** accuracy. The simulation may take some time, in case we have not been able to purchase new computers yet. Please have patience. If you have problems with your disk quota, you can also saving all nets by clicking on *Outputs > Save all...* and checking **selected** instead of **all**.

d. Open the **calculator** window with *Tools > Calculator....* Set up a **DFT** analysis of the input signal and of the AD/DA-converted output signal, as e.g. shown in Figure 17: select a signal clicking on **vt** (= transient voltage) in the **calculator** and then on the net *Vdac* in the **schematic** view; select **Special Functions** in the **Function Panel** (towards the bottom of the calculator window) and click on **dft**; use the selected **vt** signal in the **Signal** field; write **3µ** in the **From** field (start time of the DFT) and **515µ** in the **To** field (this choice results in an integer number of signal periods), and **1024** in the **Number of Samples** (leave the default values in the remaining fields). Click on **Apply**; click on **Special Functions** and **Modifier** and select **dB20**. Finally, plot the DFT with *Tools > Plot* (or by clicking on the appropriate icon). Figure 17 shows how the calculator window could look like (the exact content depends on the actual steps taken in setting up the DFT analysis).
Question: How can you estimate the SQNR from the DFT plot? (do not forget about the processing gain of the DFT/FFT). Is it in accordance with the theoretical value for an ideal 8-bit ADC? Why is 1MHz the maximum frequency in the DFT spectrum?
Question: How is the noise floor of the DFT affected by the number of samples used? Try using half as many points in the DFT. What happens?

e. Start **MATLAB** and evaluate the **SNR**, **SNDR** and **SFDR** for the output signal stored in the file filename, taking the FFT on an integer number of signal periods and employing rectangular windowing (use the knowledge acquired in the first lab). Filename contains three columns: the first is the absolute simulation time, the second is the ADC output (converted into analog data by the DAC), and the third is the ADC input. Load filename in the current working directory of MATLAB. The MATLAB command

```
> load filename
```

loads the data contained in filename into a MATLAB array called **filename**.

f. Change the **offset** of the comparators to **100 mV**. Repeat the single-tone simulation.
Question: what is the level of the SNDR, HD2 and HD3 now?

g. Reset the comparator offset to **0V**. Change the amplification of the analog signal in all the pipeline stages from **2** to **1.96** instead (for a 2% gain error). Repeat the transient simulation.
Question: what is the level of the SNDR, HD2 and HD3 now?

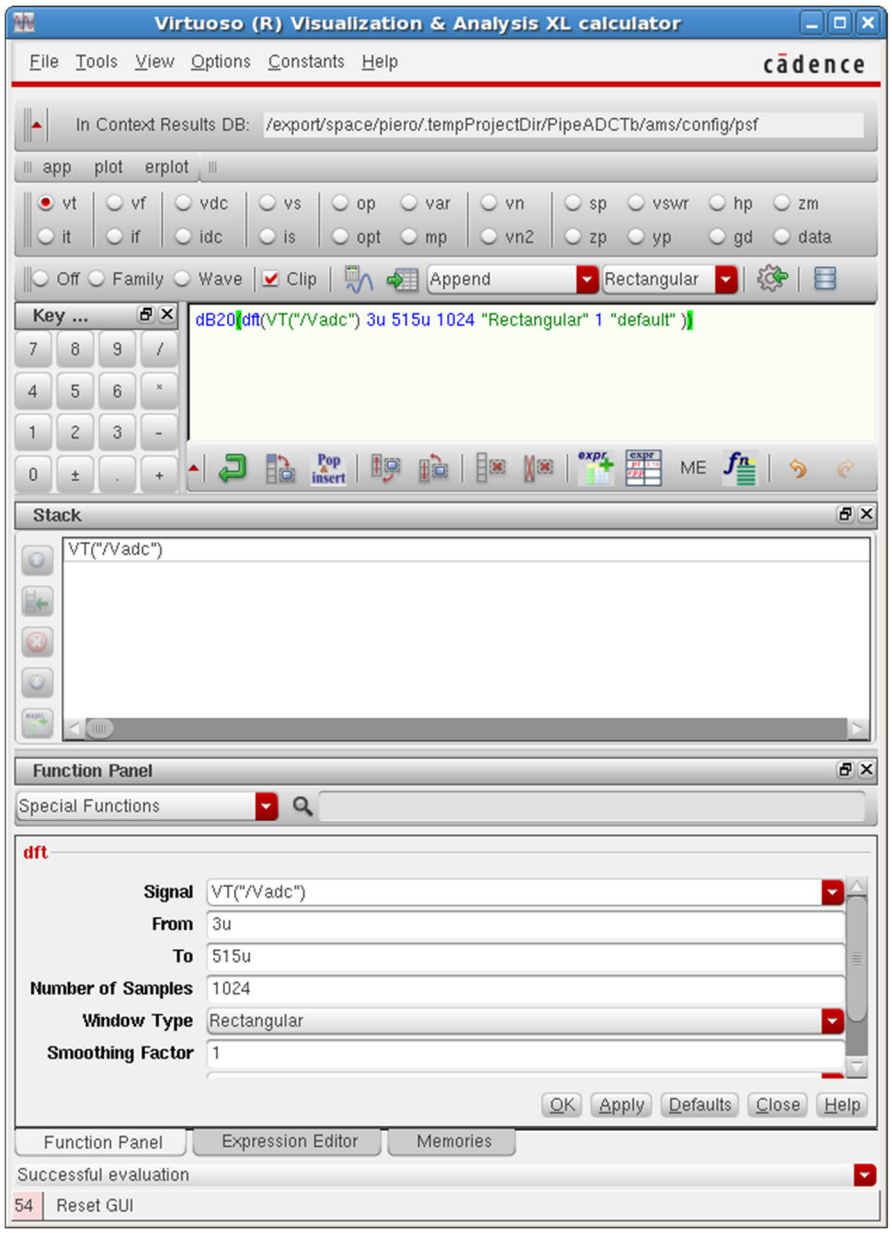

Figure 17. Calculator window for evaluating a DFT.

12. Now you will evaluate the static nonlinearity of the ADC by simulating and calculating the **INL** and the **DNL** using MATLAB.

    a. Change the input signal to a ramp with a length of **1280µs**.

    b. Use the block **SampleToFile10** to save the output voltage V_{ADC} to a file.

    c. Update your **config** view of the test bench. Make sure that the gain of the pipeline stage is exactly **2**, and the comparator offset is **0V**. Run a **1286µs transient** analysis with **moderate** accuracy.

    d. Use MATLAB to plot the INL and the DNL of the ADC. Since you used an ideal ADC model, the ADC will be perfectly linear as well. To evaluate INL/DNL, the file `INLDNL.m` from **MatlabLab.zip** (already used in the first lab) as well as your data file `filename` has to be in your MATLAB working directory. Since the input ramp is 1280µs long, for the ideal ramp there will be 10 identical output values for each code,

a fact that is exploited in the call to INLDNL (third parameter in the call). You should **remove** some initial lines (use *emacs* or your preferred text editor) from filename, so that the latency of the ADC is taken into account. In our case, the latency from input to output is 6 clock cycles, which means that the valid output data appears **3µs** after the corresponding data appeared at the ADC input. If the input ramp starts with some delay, this delay must be added to the latency to find the first relevant output sample. Furthermore, the length of the output ramp must be exactly **1280µs**, which means that you may have to remove some of the last samples as well. If you have done everything correctly, you should get a DNL (and INL) of exactly zero for all codes. Include the plot in your lab report.

```
> a = INLDNL('filename',8,10);
```

e. Change the **amplification** of the analog signal in all the pipeline stages to **1.96**. Run a new simulation and evaluate the INL / DNL. The result should qualitatively look like that in Figure 18.

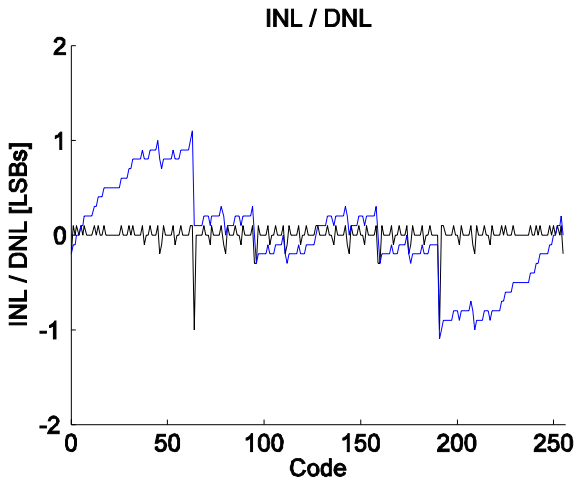

Figure 18. INL and DNL of the pipeline ADC with and without 2% gain error in all stages.

13. Two-tone test. When two signals with different frequencies are applied to a nonlinear system, the output in general exhibits some components that are not harmonics of the input frequencies. In this exercise you will simulate the **IM3** using a two-tone test.

    a. Change the input signal to the **two-tone signal** of the test bench.

    b. Keep the amplification of the analog signal in all the pipeline stages at **1.96**.

    c. Run a **520µs** transient simulation and make a DFT in Cadence using the calculator with the same settings as before (ensuring that the DFT time window is an exact multiple of the period of both input signals).
    Question: what is the level of the IM3?
    Question: What is the most critical operation in a pipeline ADC with 1.5-bit pipeline stages, amplification or comparison?