Experience in Using Open Command Environment for Analysis in Education

Dejan Mirković, *Member, IEEE*, Predrag Petković and Vančo Litovski

***Abstract*—This paper sublimates experiences in using Open Command Environment for Analysis (OCEAN) for teaching IC design at University of Niš. IC design requires a lot of repetitive analysis needed to get better insight into possibilities of a new or unknown process. Since design automation implies usage of appropriate software environment, one such environment known as OCEAN will be considered. Basic guidelines and techniques for exploiting it will be given. Concrete example will demonstrate usefulness of the platform.**

***Index Terms*—Electronic Design Automation; Simulation; Integrated Circuits**

I. Introduction

ELECTRONIC design automation (EDA) proves to be inevitable when designing complex systems such are integrated circuits (IC). Even more, pre tape-out verification of the chip is impossible without utilizing some form of design automation environment. Therefore a rapid changes in IC industry provokes leading EDA and Computer Aided Design (CAD) software providers to respond with a verity of versatile verification/simulation environments. Now days it is not sufficient only to purchase and master adequate solver tools (simulator) when designing a complex mixed-signal ICs. Instead one needs to adopt some EDA platform. Accordingly sound knowledge of scripting language for target EDA software is quite useful and practically obligated in skill arsenal of an IC designer. This paper has a purpose to give insight into one EDA solution and provide basic guidelines for exploiting it. Lack of information covering this topic was the primer motive for the authors of this paper. IC industry standard Open Command Environment for Analysis (OCEAN) propagated by EDA/CAD software provider Cadence® is explored. Nevertheless techniques presented here can be applied in any other EDA platform. At the end this work will result in concise tutorial for circuit design engineering community.

Paper is organized as follows. Firstly some introductory discussion about OCEAN in general will be given. This will cover the generic script structure and explain types of scripting commands. In the third section concrete example will be presented. This section will demonstrate application of OCEAN scripting on a test-bench for extracting small-signal gain of MOS device in 350nm process node. Fourth section will present obtained simulation results. Conclusion will be drawn in the final section.

Dejan Mirković is with the Faculty of Electronic Engineering, University of Nis, 14 Aleksandra Medvedeva, 18000 Nis, Serbia (e-mail: dejan.mirkovic@ elfak.ni.ac.rs).

Predrag Petković is with the Faculty of Electronic Engineering, University of Nis, 14 Aleksandra Medvedeva, 18000 Nis, Serbia (e-mail: dejan.mirkovic@ elfak.ni.ac.rs).

II. OCEAN Basics

The goal of this section is to give brief insight into OCEAN scripting. OCEAN is primarily intended to automate analog IC circuit design. More about experience in using complete Cadence® Design System (CDS) can be found in [1]. From programming point of view OCEAN can be thought of as a subset of Lisp based SKILL language [2]. In other words SKILL is a proprietary dialect of Lisp provided by Cadence®. Lisp is well known as pioneer language used in early EDA/CAD tools since considered suitable enough to perform the task at that time [3]. According to [4] it is still attractive for solving problems in the field. This is important to emphasize because dealing with OCEAN requires at list familiarity with SKILL and Lisp syntax [5]. Fortunately OCEAN is tailored from SKILL exactly to match the designer's needs. Namely it provides tight integration between standard design data base entities of CAD tools such are libraries, cells and views (already familiar to the designer) and the powerful scripting/programming engine. This paradigm allows one to efficiently explore design behavior or confirm it's functionality under various conditions all in one step. For this convenience designer has to invest additional intellectual effort which will be paid of at the end with robust design and saved time. Since OCEAN contains only commands targeted towards simulation and results manipulation learning curve is quite upwards steep.

According to [6] OCEAN script has a rather simple structure depicted in Fig. 1.

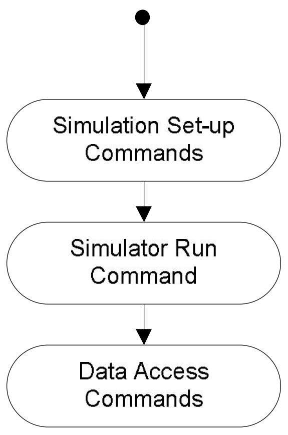

Fig. 1. OCEAN script structure.

Practically set of scripting commands can be roughly divided into three subsets.

Simulation Set-up Commands are used to declare target design data base and simulator setup options. This implies items like top-level netlist and model libraries location, analyses to be run, the nets and currents to be saved, simulator option values, additional circuit stimulus etc.

The second subset i.e. Simulation Run Commands serves to invoke simulator and monitor simulation process. Here start and stop simulation events are detected.

Third subset named Data Access is utilized to perform desired calculations, print information and plot waveforms.

The frequently used commands/functions form all three sub sets are surmised in Table 1.

TABLE I
KEY OCEAN COMMANDS

Subset	Command name
1	ocnWaveformTool, design, resultsDir, modelFile, analysis, paramAnalysis, save, stimulusFile, envOptions, desVar
2	run, paramRun
3	results, outputs, openResults, getResults, getData, v, pv, ocnPrint, plot, ocnYvsYplot, hardCopy

Detailed description of these commands is documented in [6] and [7] and it is far beyond scope of this paper. Here we only outlined them and explained their purpose while application will be demonstrated on concrete example in the following section.

At this point it is instructive to give guidelines for organizing working environment and writing clean scripts. Along the way some useful SKILL functions will be mentioned and explained as a side effect.

When dealing with scripts it is important to properly organize work environment i.e. to create appropriate directory tree. This is very important since a number of files are usually generated after script execution. Example of well organized directory tree is illustrated in Fig. 2.

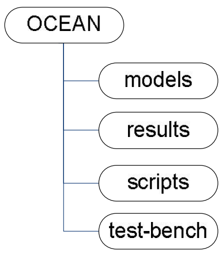

Fig. 2. OCEAN script structure.

In Fig. 2 *OCEAN* is the root working directory. Directories named *models*, *scripts* and *test-bench* contains model libraries, OCEAN scripts and design top-level netlists, respectively, while *results* will store simulation results. Once defined all created scripts should appreciate this tree structure. This way scripts are easy to maintain and port.

It is important to emphasize that OCEAN/SKILL is not strongly typed language. Precisely speaking everything is referred to as symbol rather then variable with associated type. This means that symbol inherits its type from the value assigned to it. Example in Fig. 3 illustrates this property.

```
x = '("str" d 10.0)             ; x is a list
x = (strcat nth( 0 x ) “ing" ) ; x becomes string
x = g                           ; x becomes symbol
x = 10                          ; x becomes integer
```

Fig. 3. Code example for type inheritance

In Fig. 3. symbol *x* is first defined as a list containing string “*str*”, symbol *d* and a real number of a value 10. Then it becomes a string value *“string”*. This is done by using SKILL function *strcat* for concatenation of strings. Therefore first element of the list, *x* previously was, is concatenated with string *“ing”* and the result is assigned to *x*. One should note that, since originate from Lisp, SKILL supports Polish notation for function calls. Function *nth* is called in the standard way i.e. passing the list of arguments. Type is changed again when equating with symbol *g*. Finally *x* becomes an integer value 10. This example shows that special care has to be taken when assigning values to a symbol since there is no strong type constraint like in conventional procedural languages (e.g. C).

Bearing in mind these concepts and guidelines one can develop a custom OCEAN script according to it’s circuit need. As promised earlier, automation process will be demonstrated on concrete example in section that follows.

III. AUTOMATION EXAMPLE

Example presented in this section will consider one rather simple design task. The goal is to extract small-signal gain of a basic, 350nm process node, MOS devices. For fast hand calculation one can use (1)

$$A_{vo} = \frac{2LV_E}{V_{ov}} \ , \tag{1}$$

where V_{ov} stands for transistor overdrive voltage ($V_{GS} - V_{TH}$), L represents device channel length and V_E represents process dependent parameter expressed in volts over meters [7]. This process parameter can be thought of as an equivalent for the Early voltage of bipolar transistor. It is well known that V_E parameter cannot be accurately estimated because it strongly depends on technology process. Therefore the standard way for calculating small signal gain is simulation. Even more it is advisable to explorer its behavior for different bias conditions and channel lengths. This clearly implies multiple parametric analyses. Both types i.e. P and N type of the devices has to be examined. Accordingly one needs to simulate two different test-benches. Finally, obtained results must be presented in proper way i.e. waveforms plots. To perform this relatively simple task only by using simulator and hand calculator is already tedious, error prone work. Problem becomes more serious for big and complex mixed signal designs where a large number of different scenarios must be covered.

Test-bench used for calculating small-signal voltage gain of N-MOS device is shown in Fig. 4.

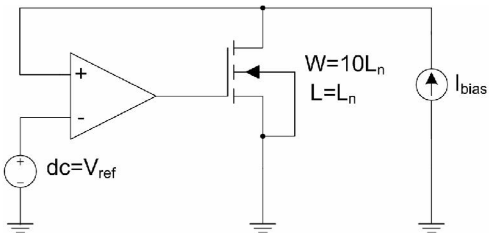

Fig. 4. Circuit for calculating small-signal gain.

Circuit in Fig. 4. provides possibility of sweeping two bias parameters. One is referent voltage, V_{ref}, and the other is bias current, I_{bias}. Practically this circuit tries to mimic real working conditions of MOS device when used in analog e.g. amplifying circuit. Operational amplifier is described as an ideal with not to high DC gain (about 100) in order to reduce simulation time. This setup should force drains-source voltage to follow V_{ref} changes while I_{bias} is held constant defining appropriate gate-source voltage. On the other hand positive feedback system maintains nearly constant drain-source voltage (equal to V_{ref}) while sweeping the I_{bias}. Therefore two scenarios are examined, constant bias current with swept voltage and vice verse. Same topology is used for P-MOS device but with reversed bias conditions. Further more all this should be repeated for different channel lengths. Therefore three channel lengths are picked {0.8, 4.8, 9.6}μm. First is quite typical staring channel length in analog applications. It is chosen to be two times the minimal (0.4μm) in target technology. Second is moderate and third is a bit extreme, twenty four times minimal. For sake of simplicity same lengths are adopted for both types of MOS devices. Width of the N-MOS is ten times while for P-MOS is twenty times the length in order to diminish carrier mobility difference.

To perform discussed task custom OCEAN script is written. It is completely presented in Fig. 5 thorough 7.

As one can see in Fig. 5 first block of script code declares a number of useful variables. Practically a set of lists is defined. These are used for holding design/model/results paths, sweep parameters/sources and its default values, pattern strings and flags. This way script is readable and easy to maintain.

Functional part of the script is presented in Fig. 6. Here one can see how the automation process is performed. Basically it is implemented with two nested *foreach* loops. The first loop controls which device is simulated i.e. N or P type and it is passed twice i.e. once for each device. Second loop also has two passes. It controls which type of analysis to run. First voltage (V_{ref}) and than current (I_{bias}) dc-sweep is preformed for three values of channel length parameter at 27°C temperature.

Since I_{bias} change covers large range, [1μA, 1mA], it advances logarithmically in 10 points per decade. It should be mentioned that all settings regarding design/model locations, simulator options, design variables has to be defined before calling *analysis* and *paramRun* (or just *run* for non-parametric analysis) commands.

```
desPth = "../test-bench/tsmc350n/";

netLst = '( "n/fb/netlist" "p/fb/netlist");

resLst = list(
     '( "../results/tsmc350n/n/etran-2014/vds-sweep"
          "../results/tsmc350n/p/etran-2014/vds-sweep" )
     '( "../results/tsmc350n/n/etran-2014/ib-sweep"
          "../results/tsmc350n/p/etran-2014/ib-sweep" )
);

parLst = '("Ln" "Lp");
parValLst = '( 0.8u 4.8u 9.6u );
devLst = list( '("Vrefn" "Vrefp") '("Ibn" "Ibp") );
devRangeLst = list(  '( 0.0 3.3 ?step 0.1) '( 1u 1m ?dec 10 ) );
trLst = '("Mn" "Mp");
outFlLst = '( "fig_0" "fig_1" "fig_2" "fig_3" );
flCnt = 0;
```

Fig. 5. Assigning useful variables.

```
foreach( (res dev devRng) resLst devLst devRangeLst

     resSubLst = res;
     devSubLst = dev;

     foreach( (devVar netVar resVar trVar parVar)
               devSubLst netLst resSubLst trLst parLst

          sweepDevice = devVar;
          sweepParams = devRng;
          designPath = strcat( desPth netVar) ;
          resultPath = resVar;
          transistorType = trVar;
          flCnt = flCnt + 1;

          ocnWaveformTool( 'viva );
          simulator( 'spectre );
          design( designPath );
          resultsDir( resultPath );
          modelFile('( "./models/tsmc035.spi"));
          stimulusFile( "./global-nets.scs" ?xlate nil );
          envOption( 'userCmdLineOption "+csfe" );
          temp( 27 );
          desVar( "vd_n" "1.65" );
          desVar( "vd_p" "1.65" );
          desVar( "ibias_p" 250u );
          desVar( "ibias_n" 250u );
          desVar( "Sn" 10 );
          desVar( "Sp" 20 );
          desVar( "Ln" 0.8u );
          desVar( "Lp" 0.8u );
          save( 'i strcat( trVar ":d" ) );

          analysis('dc ?saveOppoint t ?dev sweepDevice ?param "dc"
                    ?start nth( 0 sweepParams) ?stop nth( 1 sweepParams)
                     nth( 2 sweepParams) nth( 3 sweepParams) );
          paramAnalysis( parVar ?values parValLst    );
          paramRun();
```

Fig. 6. First part of the functional code

Finally the rest of the code covers small-signal gain calculation and various ways of saving/presenting results. It is shown in Fig. 7. Firstly, small-signal transconductance, g_m and drain-source admittance, g_{ds}, are extracted from results data base of parametric DC-sweep analysis using the *v* function. Then small-signal gain is calculated as a ratio of g_m and g_{ds}. Gain values obtained this way are far more precise then one with hand calculation therefore giving the better insight into behavior of the circuit under consideration.

Finally the results are stored in the textual file with scientific notation, hard copied in vector graphics (Encapsulated Post Script) and plotted on the screen with dedicated waveform tool (viva).

```
        ; Extracnt data form results DB
        gm = v( strcat( transistorType ":gm" )  ?result 'dc );
        gds = v( strcat( transistorType ":gds" ) ?result 'dc );

        ; Calculate values of interest
        avo = gm/gds;

        ; Print to file
        awvPrintWaveform( ?output strcat( "./etran-2014/" "avo-" trVar "-fb-" devVar ) avo
                            ?format 'scientific ?numSpaces 1 );

        ; Plot results

        ; Save as EPS options
        hardCopyOptions( ?hcPrinterName "EPS" ?hcPaperSize "a4" ?hcImageSizeUnits "cm"
                         ?hcImageWidth 15 ?hcImageHeight 10
                         ?hcDisplayLegend t ?hcDisplayTitle nil
                         ?hcOutputFile strcat("./etran-2014/" nthelem( flCnt outFlLst) ".eps" )
                       );

        trStr = case( trVar   ("Mn" "For NMOS Device")
                              ("Mp" "For PMOS Device")
                              (t println("No Transistor Selected for Window Title!")) );


        devStr = case( devSubLst ('("Vrefn" "Vrefp") ", and Vref sweep")
                                  ('("Ibn" "Ibp") ", and Ibias sweep" )
                                  (t println("No Device Selected for Window Title!")) );

        winTitle = strcat( trStr devStr);
        win = newWindow();
        plot( avo ?expr '("Small Signal Gain") );
        if( devStr == ", and Vref sweep" then
            awvSetXAxisLabel( win "Drain-Source voltage" );
        else
            awvSetXAxisLabel( win "Bias Current" );
        );
        awvSetYAxisLabel( win 1 "Samll Signal Gain" );
        addSubwindowTitle( winTitle );

        hardCopy(win);

    );end foreach

);end foreach
```

Fig. 7. Second part of the functional code

All values extracted from the data base are classified as waveform objects and can be plotted using *plot* functions. Besides simple waveform arithmetic (division, summation, multiplication) OCEAN also supports more sophisticated functions like integration, differentiation and Discrete/Fast Fourier Transform.

IV. SIMULATION RESULTS

The previously explained automation process results with a set of waveforms. Fig. 8 presents N-MOS small-signal gain dependence on drain-source voltage when bias current is constant for various device channel lengths. The same results for P-MOS device are shown in Fig. 9. As expected for typical and moderate channel lengths N-MOS device provides better amplification of small signal voltage then P-MOS device. For extreme channel lengths this advantage fades away.

Similar behavior is evident when observing small-signal gain versus bias current with reference voltage held constant at 1.65V i.e. half of power supply voltage 3.3V. Again, larger the channel length higher the gain is. One could note in Fig. 10 that for N-MOS device, with typical channel length, gain exhibits a local maximum at approximately 20μA.

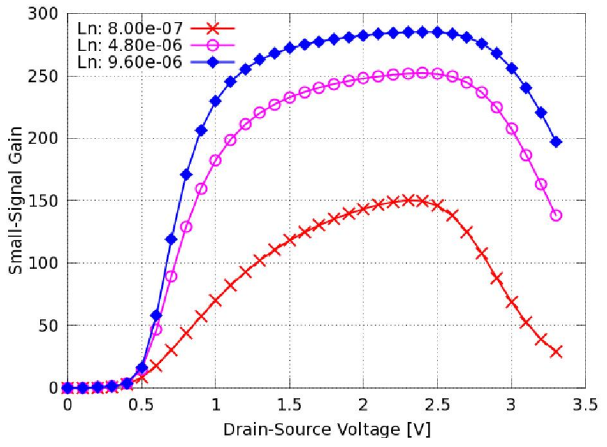

Fig. 8. Small-Signal Gain versus Drain-Source Voltage for N-MOS device

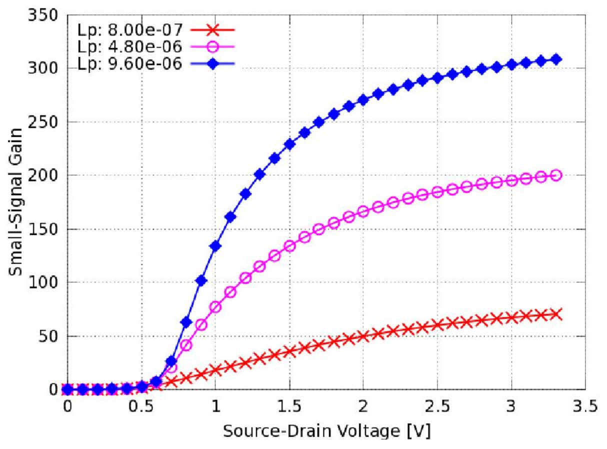

Fig. 9. Small-Signal Gain versus Drain-Source Voltage for P-MOS device

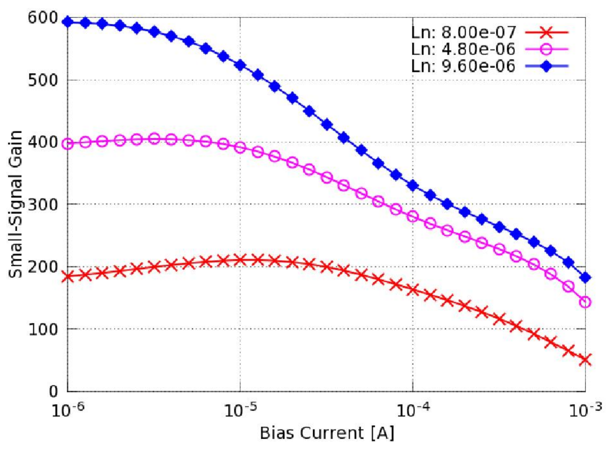

Fig. 10. Small-Signal Gain versus Bias Current for N-MOS.

This should be the optimum bias current point if device is used in amplifying application. Also for moderate and extreme channel lengths gain curves are lightly dent at higher bias currents (100μA).

Unlike the N-MOS, P-MOS gain curves preserves half-parabolic shape regardless of channel length. In this case N-MOS device wins P-MOS over whole range of bias currents for all values of channel lengths.

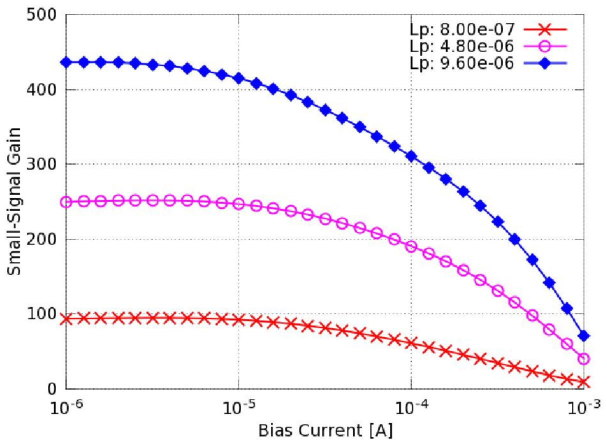

Fig. 11. Small-Signal Gain versus Bias Current for P-MOS.

Obtained results are valuable to a circuit designer since provide important information about key design parameter in target technology. Myriad of other design parameters can be derived using presented script as a template to start with.

V. Conclusion

It is known that circuit designers often repel to use programming/scripting languages and this paper should encourage them to incorporate one in its engineering practice. Each professional CAD tool supports some kind of scripting language therefore why not to exploit tool's full capabilities. Author's goal was to bring closer this topic to the circuit design engineering community and provide constructive tutorial.

The paper covered usage of industry standard EDA platform and emphasis importance and motive to use one such software solution. Characteristics and properties of OCEAN scripting language are discussed. Guidelines of proper script writing and organizing work environment are given, as well. Often used SKILL specific functions are explained. Potential errors such those concerning type casting are pointed out.

Application of OCEAN script on concrete, educational example is demonstrated. Reader is guided through custom written OCEAN script which implements desired automation procedure. Script functionality is discussed in detail. As a result a set of waveforms giving the insight into circuit behavior are presented and commented.

Acknowledgment

This work is funded by Serbian Ministry of Education, Science and Technological Development within the project No. TR, 32004 entitled: "Advanced technologies for measurement, control, and communication on the electric grid.

References

[1] M. Dimitrijević, B. Jovanović, B. Anđelković, M. Savić, M. Sokolović, "Experiences in Using CADENCE - The Industry Standard for Integrated Circuits," Proc. of XLVII ETRAN Conf., Herceg Novi, Montenegro, vol. 1, no. 47, pp. 31-34, 8.-13. June, 2003.

[2] T. J. Barnes, "SKILL: a CAD system extension language," Proc.of 27th ACM/IEEE Des. Auto. Conf., New York, USA, vol. 1, no. 27, pp. 266-271, 1990.

[3] J. McCarthy, "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I," *Comm .of the ACM*, vol. *3,* no. *4,* pp. *184-195, April*, 1960.

[4] D. Cerny, J. Dobes "Common LISP as Simulation Program (CLASP) of Electronic Circuits," *Radioeng.* , vol. *20,* no. *4,* pp. *880-889, Dec.*, 2011.

[5] "SKILL Language Reference", Cadence Design Systems, Inc., San Jose, 2002.

[6] "OCEAN Reference," Cadence Design Systems, Inc., San Jose, 2004.

[7] "Cadence Analog Design Environment SKILL Language Reference," Cadence Design Systems, Inc., San Jose, 2002.

[8] W. Sansen, *Analog Design Essentials*, 1st ed. Dordrecht, Netherlands: Springer, 2006.