Analysis of Sub threshold voltage Designed in Verilog using Python

Xavier Dunkley and Matthew Guthaus

xavierxv18@gmail.com, mrg@soe.ucsc.edu

Computer Engineering Department, University of California Santa Cruz

Abstract

Sub threshold voltage circuit design is a growing field that involves lowering the power consumption of microprocessors used in electronic devices. This means that the microprocessor will use less power and will put less strain on the power supply. However, these circuits are very unstable and are therefore unreliable for practical use. The purpose of this research project is to develop a code which will read a model of an electronic system from a hardware descriptive language and perform tests to determine if the circuit design is stable for practical uses.

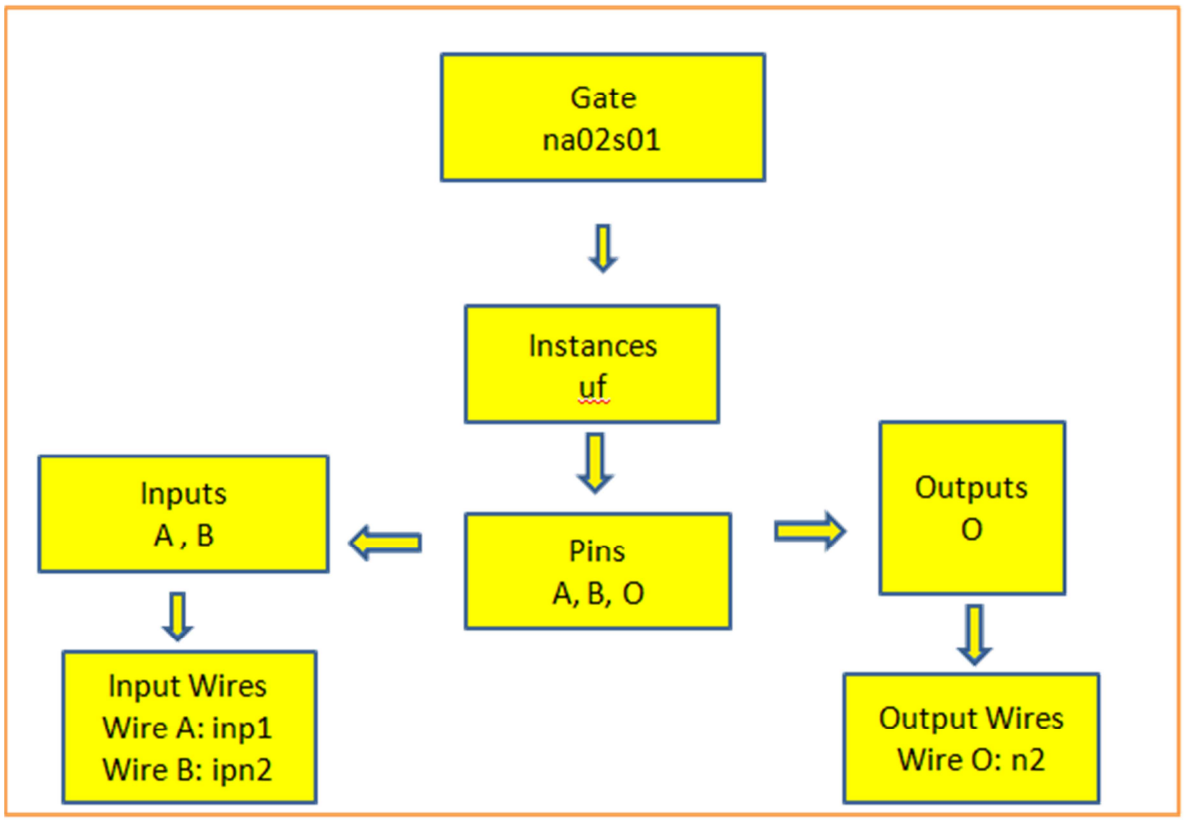
Introduction

Threshold voltage is the voltage that causes a gate to be in its "on" state. The problem with this is that is causes a strain on the energy supply of the device that is used. The team at the Very Large Scale Integration Lab (VLSI) at UCSC is working to solve this problem by using Sub threshold voltage applications. Sub threshold voltage operation is a method used for energy conservation in electronic devices by lowering the voltage required to turn on the device. This voltage is normally around 0.2mV which is below the standard that is normally used ordinary devices. Reducing the voltage to this level will also cause a drop in performance which is why sub threshold voltage circuits are usually used in devices that do not require high speed or performance. It involves using the leakage current to drive the microchip and therefore save on power consumption. However, this makes the chip very sensitive to fluctuations in energy and therefore makes the chip very unstable. The main goal of this research endeavor is to stabilize the circuits so that it can be used for practical applications.

Method

The circuits that were used in this project were made using Verilog which is a hardware descriptive language that is used for modeling, designing and testing electrical systems. The data that was produced in Verilog was then collected using Python, an interpreted high-level programming language, so that the data operations can be performed on it. The data that was collected needed to be organized in a data structure so that it can be used efficient by Python. Using Python, the Verilog data was parsed and put into a hash table. This hash table has three levels of keys which are the gates names, the instances of the gates and the pins for the gates inputs and outputs. The pin inputs keys contained an array which stored the wires that are connected to the gate input pins and the output pin key stored the wire that is leading out of the output pin. Figure 1 shows the arrangement of the hash table that was created in Python.

Figure 1: Diagram of Data Structure



By organizing the data from Verilog into this data structure, it is now possible to perform various operations on the data collected by Python from different designs in Verilog.

Breath-First Search

A breath-first search is a graphing technique use to search through a graph by visiting and inspecting a node of the graph and any neighboring node that is connected to the node that is being inspected. This process is continued until all the nodes are inspected. This technique was used to search through the gates of the Verilog code to find the wires that are connected to the gates input and what wire will come out of its output. It does this by first taking the input wires that are in the primary input array and putting them in a new array called the "queue" array. Next it compares the elements in the array with the elements in the arrays that are linked to each gate. If the queue finds that of its elements matches the elements in the array of the gate, it adds "1" to the counter that is linked to the gate. Once the value of the counter is the same as the number of elements in the array of the gate, the element of the gates output array is appended to the end of the queue and the process continued until there are no more elements in the queue.

Occurrence Counter

A code was also created to count the number of occurrence of different gate connections that were in the Verilog design. First, an ordered list of elements, known as tuples, was created using the names of the different gates that may be found in the Verilog file and kinds of one to one connections that occurred in the models was created. These were then used as keys for a data structure that will hold the number of occurrence of the connections in the Verilog model. The code, using a backward breath-first search technique, then takes the names of the two gate that are connected together to form a string that is the same name arrangement as the name of the tuples. If the name of the string matches the name of the tuple, the counter for the tuple is incremented by "1." This is continued throughout the data structure until every possible occurrence of every connection in the design was counted. The results were then put into an excel file.

Results

Table 1 and Table 2 show the most common and the least common circuit connections in the Verilog design in file B19_slow. By collecting this data, it is now possible to find out which gate arrangements will cause the most strain on the circuit and measures can be made to reduce the power consumptions of the gates.

Table 1: Most common connections in the Circuit B19_slow

Gate Connections	Occurrences in the Circuit
Inverter01---nand_gate02	40822
nand_gate02----nand_gate02	32756
Inverter01----and/or_gate12	28350
Inverter01----nor_gate02	26155
Inverter01----and/or22	24320
nor_gate02 ----and/or_gate12	21260
nand_gate02----nor_gate02	17050
nand_gate02----inverter01	16382
nor_gate02 ----inverter01	13262
and/or_gate12----nor_gate02	11134

Table 2: Least common connections in the Circuit B19_slow

Gate Connections	Occurrences in the Circuit
nor_gate04---or/and_gate22	8
nor_gate03---or/and_gate12	7
nor_gate02 ---nor_gate03	6
nor_gate03---nand_gate04	6
and/or_gate12---nor_gate04	4
nor_gate03---nand_gate03	4
nor_gate04---nand_gate04	4
or/and_gate22---or/and_gate12	3

or/and_gate12---or/and_gate22	2
or/and_gate22---nor_gate04	1

Other Work

For future work, the code must be refined to find the Logic Effort of the circuit. The logic effort is a strait forward way of calculating the delay of the circuit. This involves determining the capacitance of the gates in the circuit and then changing the size of the width of the gate. By doing this it will now be possible to not only change the gates to have the smallest delay possible, but we can also reduce the gate voltage so that it can consume less power. Then graphing code that will plot the Static Noise Margins of the circuit needs to be created.

Acknowledge

I want to thank Matthew Guthaus for guiding me and giving me the opportunity to work with him. I want to thank the National Science Foundation providing funding for the project and The University of California of Santa Cruz for hosting this program.

References

- Calhoun, B.H.; Brooks, D.; , "Can Subthreshold and Near-Threshold Circuits Go Mainstream?," *Micro, IEEE* , vol.30, no.4, pp.80-85, July-Aug. 2010
- Kwong, J.; Chandrakasan, A.P.; , "Variation-Driven Device Sizing for Minimum Energy Sub-threshold Circuits," *Low Power Electronics and Design, 2006. ISLPED'06. Proceedings of the 2006 International Symposium on* , vol., no., pp.8-13, 4-6 Oct. 2006
- R. F. Sproull and I. E. Sutherland, “Logical Effort: Designing for Speed on the Back of an Envelope”, *IEEE Advanced Research in VLSI, C. Sequin (editor), MIT Press, 1991*