Analog Modelling to Suit Emulation for Hardware-Software Co-Verification

Saranya Das, Senior Design Verification Engineer, Analog Devices, Bengaluru, India
(saranya.das@analog.com)

***Abstract*—Firmware-Hardware co-verification is an important pre-silicon task to ensure faster silicon delivery to customers. Emulation platform is used to validate the firmware before silicon arrives for saving time to market. In simulations, analog signals are modeled using RNM (real number modelling), but RNM can't be used on an emulator as the datatype is 'real'(non-synthesizable). This paper describes how fixed-point arithmetic was used to model an analog block like ADC. Python code was used to visualize and decide the optimum bit-width for modelling. The paper also discusses techniques for creating synthesizable testbench and modelling of commonly used analog blocks to suit the emulation platform.**

Keywords—real number modelling, emulation, harware-software verification

I. INTRODUCTION

Boot code is the program that is burned into the ROM(Read Only Memory) of the SoC. It boots up and configures the SoC. In the process of SoC development, firmware is developed in parallel to digital/analog design by referring to specification. Any bug in the firmware code will be difficult to debug in silicon which will delay the delivery of silicon to customers. It may also result in silicon re-spin and further the delay. Verifying firmware routines pre-silicon is necessary but the regular RTL simulation environment is very time consuming as it can go on for days. This is where emulation comes into picture in which the DUT is brought up in an emulation platform and firmware verification tests are run on it. These tests can run from seconds to minutes depending on the size and code of DUT.

However, bringing up the DUT for emulation is not straight forward and requires modelling of some blocks especially analog to the make the DUT synthesizable and suit emulation.

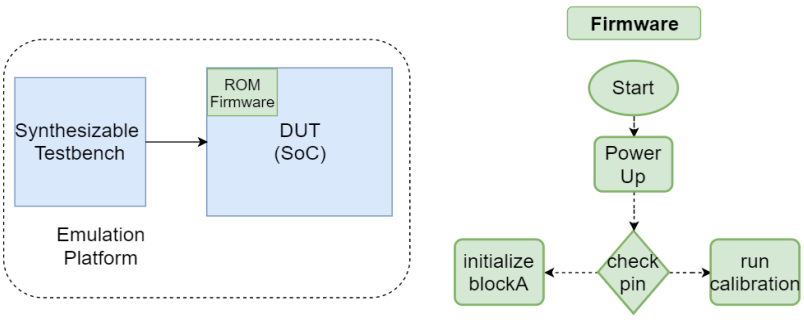

Figure 1. Verification Environment

II. MODELLING OF ANALOG BLOCKS

The biggest challenge was to make the analog components (modelled as RNMs in simulation) portable. Data types such a float and real are non-synthesizable constructs. This was solved using fixed-point arithmetic.

Typical modelling flow is as shown in Figure 2 where the analog circuits or real number models are converted to synthesizable models. As an example, some logic of the modeling done for an ADC block which takes an analog input and gives a digital output is shown in Figure 3.

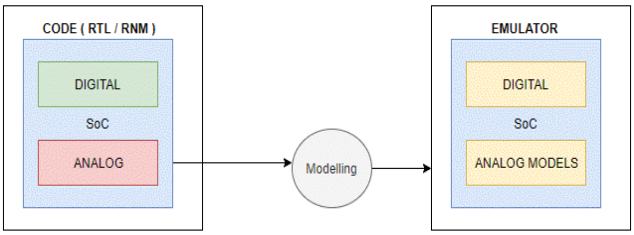

Figure 2. Modelling Flow for Emulation

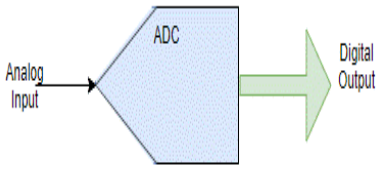

Figure 3. ADC Modelling

```
real vx=1.123
real va=0.234
real lsb=0.000392
real vc=1.002

vx=va-vc+lsb*256
```

Figure 4. Real Datatypes

Format	value A	Scaling factor	Input * scaling_factor	Integer value	Integer/scaling factor (B)	error(A-B)/A
1.10	0.0025	2^10	2.56	2	2/(2^10)=0.00195	22% e
1.20	0.0025	2*20	2621.44	2621	2621/(2^20)=0.00249	0.4 % e

Table 1. Converting Real Number to Fixed Point Arithmetic

Real data type or RNM cannot be used for emulator. The above code (Figure 4) will be converted to a fixed-point format. In fixed-point, we multiply the real number by a scaling factor such that we can represent it using integers, but this has the cost of losing resolution in the decimal points. The greater the scaling factor, lesser is the error in resolution as shown in Table 1. But larger scaling factor implies higher bus width and more ports need to be punched. This wider bus arithmetic would slow down the emulation platform. The example ADC taken here is a 12-bit ADC with a reference voltage of 1.6085 volts and 1 lsb is 1.6085/2^12 i.e 0.000392 volts.

A. Choosing the fixed-point bus width by calculating error in resolution

Error % = ((back_converted_lsb-lsb)/lsb)*100

a. scaled_lsb = (actual_lsb)*(2^bitwidth)
b. int_lsb = round(scaled_lsb)
c. back_converted_lsb = int_lsb/(2^bitwidth)

The above calculation was incorporated into a python code and a plot of the error (as calculated in examples in Table 1) vs the bus width was developed is shown in Figure 5. The visualization shows how the error decreases with increase in bit-width but saturates beyond a point. Hence, we take 14 as the bit-width.

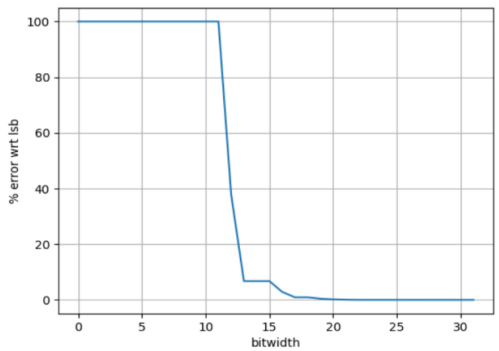

Figure 5. Error vs Bus-width

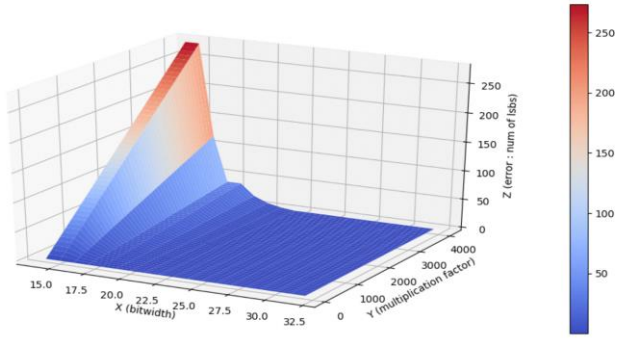

Figure6

B. Scaling to increase arithmetic accuracy:

Inside a module there may be real number multiplications. Consider the multiplication of two numbers 120 (denoted as ‘a’) and 0.000392 (denoted as ‘b’) as shown in Table 2 where scaling with 28 factor gives less error.

Format(1.f)	(a*b)	Scaled a, rounded (c)	Scaled b, rounded (d)	e=(c*d)/2^(2*f)	(x-e)/lsb
1.14	0.0474	1966080,1966080	6.422,6	0.0439	8.8 lsb
1.28	0.0474	32212254720,32212254720	105226.69, 105226	0.0470	1.02 lsb

Table 2. Scaling with factors 14 and 28

Steps for error in resolution due to internal multiplication (a*b): Scale and round off a and b. To get the actual value, divide by scaling factor twice (due to multiplication). Since ADC lsb is 0.000392, we compare the error with respect to lsb i.e how many lsbs is the error. Figure 7 shows the scaling and trimming internally done. Figure 6 shows a visualization developed using python code, where error is plotted for different bit-width. For each bit-width, the multiplier is swept from 1 to 4096 to show the effect of the multiplier on the error.

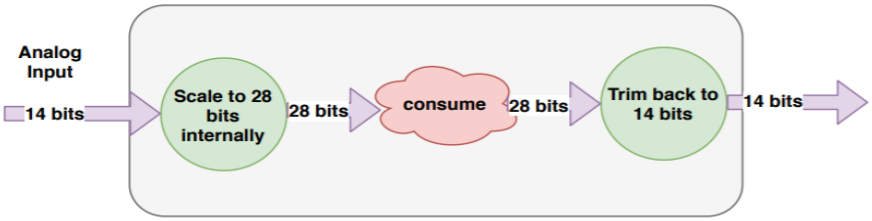

Figure 7. Scaling and Trimming

Following are steps needed to prepare the model for emulator platform.
1. Generate netlists of analog blocks from schematics with input and output ports, excluding internal logic.
2. Model required functionality using synthesizable constructs.
3. Convert analog signals to digital using fixed-point arithmetic.

1.1 Successive Approximation Register (SAR) ADC

The DUT has a Housekeeping-ADC to measure temperature from various sensors. To model analog signals, fixed-point arithmetic 2.14 format was used. This format was consistent for all analog signals. Inside a sub module for higher resolution arithmetic, this was extended to 2.32 format as shown in Table 3(Serial 2 & 3).

2.14 format was chosen as it was enough to represent the maximum voltage required by the system while preserving the resolution of the lsb(least significant bit) of the ADC. Table 3 shows the format conversion.

Serial	Analog voltage	Format		Back-conversion	Resolution lost
1	1.25v	2.14	1.25*(2^14)=0x5000	0x5000/(2^14)=1.25	0
2	1.25*1.23	2.14	1.5375*(2^14)=0x6266	0x6266/(2^14)=1.5374	0.0001
3	1.25*1.23	2.32	1.5375*(2^32)=0x189999999	0x189999999/(2^32)=1.5375	0

Table 3. Representation in 2.14 and 2.32 format

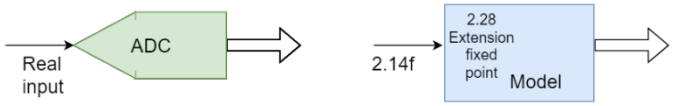

Figure 8. Real datatype converted to fixed-point for synthesizability

The subsections below discuss other modelling techniques for some commonly used analog blocks.

1.2 Clocking

Firmware configures the PLL(Phase Locked Loop) in the DUT for clock generation and expects a status signal from PLL. Only the status generation logic was modelled, and the analog logic needed for clock generation was black-boxed. Emulator generates a source clock which was used to derive required clocks for the DUT.

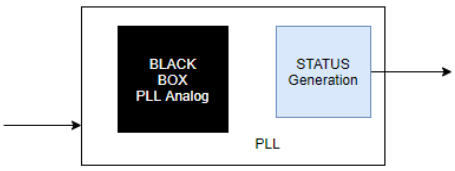

Figure 9. Black-boxing analog logic

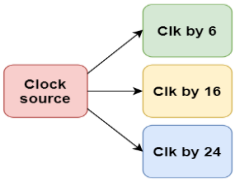

Figure 10. Clock divider

1.3 GPIO(General Purpose Input Output) Pad Modelling

GPIOs have bi-directional tri-state buffers which are not synthesizable. So, these were modelled using equivalent mux and tri-state buffer logic. However, if the GPIO instance is inside the DUT hierarchy, it doesn't function as expected. GPIO instances were created in the top module and the SPI signals were passed through these. This needed extra signals to be brought out of DUT to replicate the functionality.

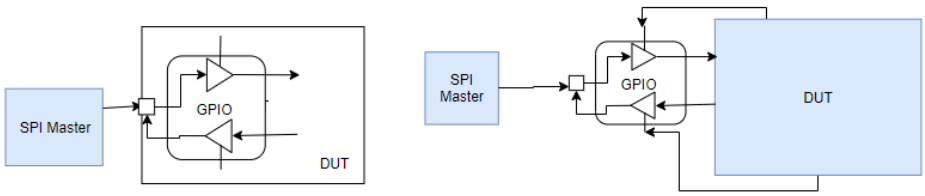

Figure 11. GPIO instance moved from DUT to testbench-top

III. RESULTS

Table 4. shows the comparison between RTL simulation and emulation time. All the tests were run before silicon came. And this saved time in the crucial window between silicon arrival and sampling to customers.

Scenarios/Tests	**Simulation time**	**RTL Simulation (wall clock time)**	**Emulation (wall clock time)**
Firmware without patch (silicon use-case) Patch code is given by Firmware team to skip long running functions, reducing simulation time.	1s	4 days	1.2 hour
Chip Initialization: routines to initialize various modules/memory in DUT	500ms	2.5 days	40 min
Long running tests: Watch Dog Timer	100ms	1.5 days	10 min

Table 4. Performance: Emulation vs RTL Simulation

Firmware Bugs	15
Firmware routines verified	6
Firmware iterations	8

Table 5. Verification Results

Silicon Results:

8 routines were verified pre-silicon and once silicon arrived; these functions ran without issues. 1 function which wasn't verified pre-silicon were stuck in silicon run. However, these could be run in emulation and the issue was zeroed in using waveforms. Thus, emulation was helpful even post-silicon.

IV. CONCLUSION

As shown in Table 5, 15 firmware bugs were found pre-silicon, these issues if found post silicon, would have delayed time to market. Thus, pre-silicon software verification is becoming an important aspect of the entire system verification. To accelerate this, emulation is the way to go. It helps to uncover any surprises when the silicon arrives. To achieve the benefits of emulation, synthesizability and analog modeling must be kept in mind while developing the testbench in any project. One huge challenge in porting RTL for emulation is modelling analog blocks and this paper shows how fixed-point modelling can be used to effectively model RNMs. The visualizations using python, acts as an effective tool for the reader to easily comprehend the trade-off between emulation speed and resolution error. The modelling approaches mentioned can be used to model any analog block to run faster and with less modelling error on emulation platforms.

ACKNOWLEDGMENT

I would like to extend our gratitude to Raman Kumar, Sriram Madavswamy, Ponnambalam Lakshamanan, Shreya Dasgupta, Pablo Cholbi, Devaphanindra Kumar, Karan Punj and Rahul Sharma. I would also like to thank all the team members of Automotive Safety- Radar group of ADI Bangalore for their constant encouragement and support.