*International Journal of Computer Applications (0975 – 8887)*
*Volume 95– No.26, June 2014*

# Analysis and Verification of Multi-Core Enabled ESL Model using SystemC and VLang

Dipti Girdhar
School of Engineering and Technology,
ITM University, Gurgaon, India

Neeraj Sharma
School of Engineering and Technology,
ITM University, Gurgaon, India

Neeraj Kr. Shukla, Ph.D
School of Engineering and Technology,
ITM University, Gurgaon, India

## ABSTRACT
Today most of the system on chip (Soc) integrate multiple processing cores, digital signal processors, as well as dedicated hardware accelerators, etc [1]. This results into large and complex systems which pose challenges to conventional design and verification flow. This raises the need for higher level of abstraction (i.e. Electronics system level abstraction). This paper discusses how we can make changes in an existing conventional method of simulation to achieve really fast high yielding method of verification. The model is implemented using SystemC and Vlang. The results demonstrate the impact of enabling multiple simulators for verification of hardware models.

## Keywords
SystemC, Vlang, Multicore, Multithread, ESL

## 1. INTRODUCTION
Today, designers develop systems that consist of application specific hardware and software and needs to be developed synchronously on a persevering schedule. These systems need to go through a profound verification processes in order to avert a calamitous collapse of the system in a real world [2].

As we all know that necessity is the mother of invention. In order to meet the demand of such complex systems in the market, that too in a short span of time, designers need some powerful tools on which they can count easily for high productivity.

Your guess is right. SystemC is one of the languages that have emerged in respond to inescapable need for a language that can support an increasing in complexity and productivity of such electronic systems [3].

Open SystemC initiative was formed in 1990, to make new design language open to the community and not be proprietary [9]. SystemC is a language which offers productivity gains by allowing engineers to synchronously develop a high level abstraction for both hardware and software as these would exist on the final systems. High level abstraction is a methodology that benefits engineers to access better understanding of intricate processes and communication among other modules at early stage. This enables better, earlier verification and overall productivity gains through reuse of early system models [4].

## 2. INTRODUCTION TO SYSTEM
SystemC is a C++ class library and Open SystemC initiative (OSCI) supports TLM 2.0 transaction level modeling.

Here we have considered a simple algorithm of converting a PNG format file into buffer data using libpng library. Libpng is a library file that makes apparent the various methods of modifying the PNG file into other formats. Libpng is a thread safe, on the condition that all the threads should be using different instances of the structures and their own image. Structure png_struct and png_info are the important structures. Structure png_struct is an internal structure, which is not used by a user. Whereas png_info structure specifies the information about the PNG file. The fields of png_info were set to be directly accessible to the user. But this is a cause of problem for the applications using dynamically loaded libraries. Therefore to avoid this problem set and get functions were developed. The png.h is a indispensable header file used inside the code when programming with libpng.

One of the features, that libpng library offer, is to handle any special transfigurations of the image data. Both SystemC and libpng are C++ libraries which are available for free. Here we are modeling a multithreaded model for converting the RGB pixels of any image into grayscale.

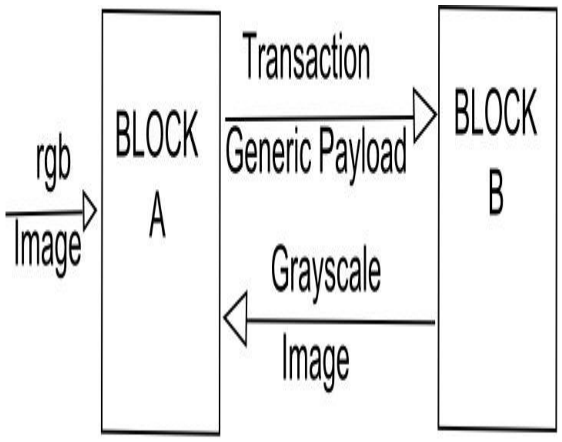


**Figure 1: Shows the block diagram of the system.**

Block A is modeled to take an image with rgb pixels as inputs. It extracts the information of each pixel and stores into a buffer. Here we add an additional payload known as generic payload to form a complete transaction as per the rules of TLM 2.0. Block A acts as initiator, as it initiates the transaction. Whereas block B acts as target as it responds to the transaction. Block B is responsible to read the data that is stored into the buffer by the block A and further making the changes in each pixel. Block B is modeled to run a following algorithm.

Image in png format, containing rgb pixels is fed into block A. Block A stores the information of the image in a buffer. It adds a generic payload with the existing stream of data stored in buffer. Thus initiate the transaction containing details of the image and acts as initiator. Block B acts as target as it responds to the transaction initiated by the initiator. Once the transaction reaches the block B through function calls, the algorithm is followed, shown in Figure 2.

Block B recollects the information through the transaction. Divide the complete image in eight parts. Since our normal system consists of four cores, we can examine the performance of model by actually enabling four simulators simultaneously. This concept is not easy to understand as none of the hardware language allows the feature of enabling multiple simulators for verification purpose. But this is very essential feature that must be provided to the verification engineers. This feature is important as it gives the real savor of multi-core. Recently while comparing different languages with SystemC, there was a language which proved to be very pleasing, when it comes to verification. Vlang is the language which offers a great variety of features for verification engineers.

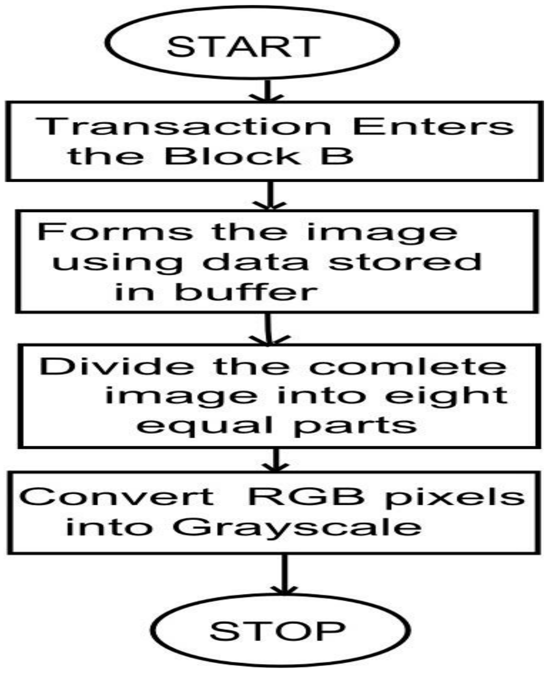

START
Transaction Enters the Block B
Forms the image using data stored in buffer
Divide the comlete image into eight equal parts
Convert RGB pixels into Grayscale
STOP

**Figure 2: Shows the algorithm followed by block B**

Vlang is the language which is recently developed by CoVerify Systems Technology. Unlike the old existing verification languages, Vlang is an open source verification language with some special features which will provide a new horizon for verification and testbench development. Vlang is a language which proves to be a great help for verification engineers as it consist of a very simple syntax and inherits some special features suchlike concept of object oriented programming, and multicore programming from D language. This makes it easy for a new developer to use this language. Even some new verification features and UVM has been implemented by the Vlang community. In this paper alsowe would be talking about the interface between the SystemC (TLM) and Vlang.

## 3. INTRODUCTION TO TRANSACTION LEVEL MODELING

Transaction level modeling in SystemC focus more on communication between processes rather than algorithm [5]. The communication between processes is done by function calls. It is assume that, some of the processes produce the data and some of the processes consume that data. While some processes are responsible for initiating the communication and other responds.

In TLM 2.0, the transaction is generally passed between initiator and target. A transaction is nothing but sets of C++ objects which are passed between initiator and target through function calls [6]. Both initiator and target use socket to send and receive the transaction respectively [7].

## 4. INTRODUCTION TO ALGORITHM

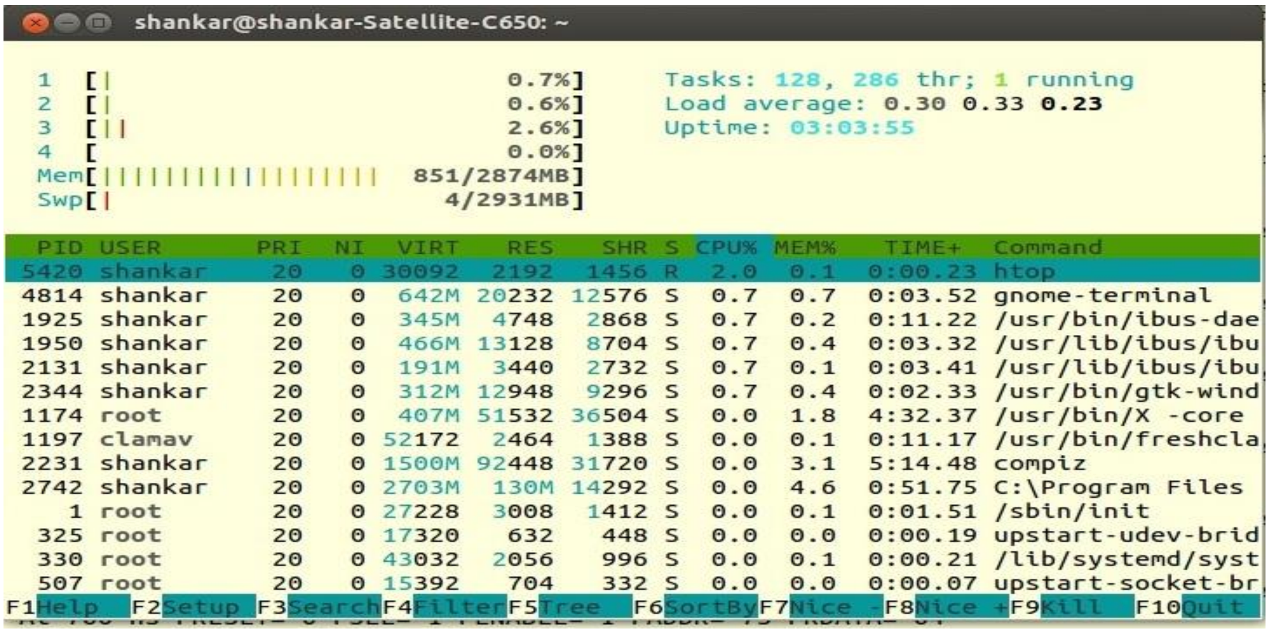
There are many algorithm for converting RGB image into grayscale are available. RGB image is the image containing the components of pure red, green, blue colour whereas grayscale image is the image containing only white and black component. Both RGB and gray values are incoded in a set of three equal numbers ranging from 0 to 255. For instance, white is represented as (255,255,255) and black is represented as (0,0,0). In order to determine grayscale value equivalent to its RGB value, one needs to calculate the average value of x by the given formula $x = 0.299r + 0.587g + 0.114b$.For instance, grayscale value equivalent to pure red (255,0,0) is gray (76,76,76).

In this paper work block A is designed using SystemC as shown in the Figure 1. As mentioned above block.

A is responsible for creating a transaction which will be fed into a block B. block B is designed using Vlang. This interface between SystemC and Vlang is at transaction level rather than signal level.

## 5. RESULTS

SystemC uses quick threads also known as Q threads as shown Figure 5. This means that if any multi-threaded application is written using SystemC, the compiler borrows one thread from the kernel of the operating system. Further, imposes all other threads of the application on a single thread borrowed from the kernel. This multiple threads present inside SystemC application are executed on the context based switching [8]. In context based switching, chunks of each thread are executed, one at a time. In order to perform a context switching, the scheduler has to save the state and instructor pointer for the current running task, has to work out the switching between the tasks and again it has to reload the CPU state. This also involves the loading and reloading of memory for data and instructions.

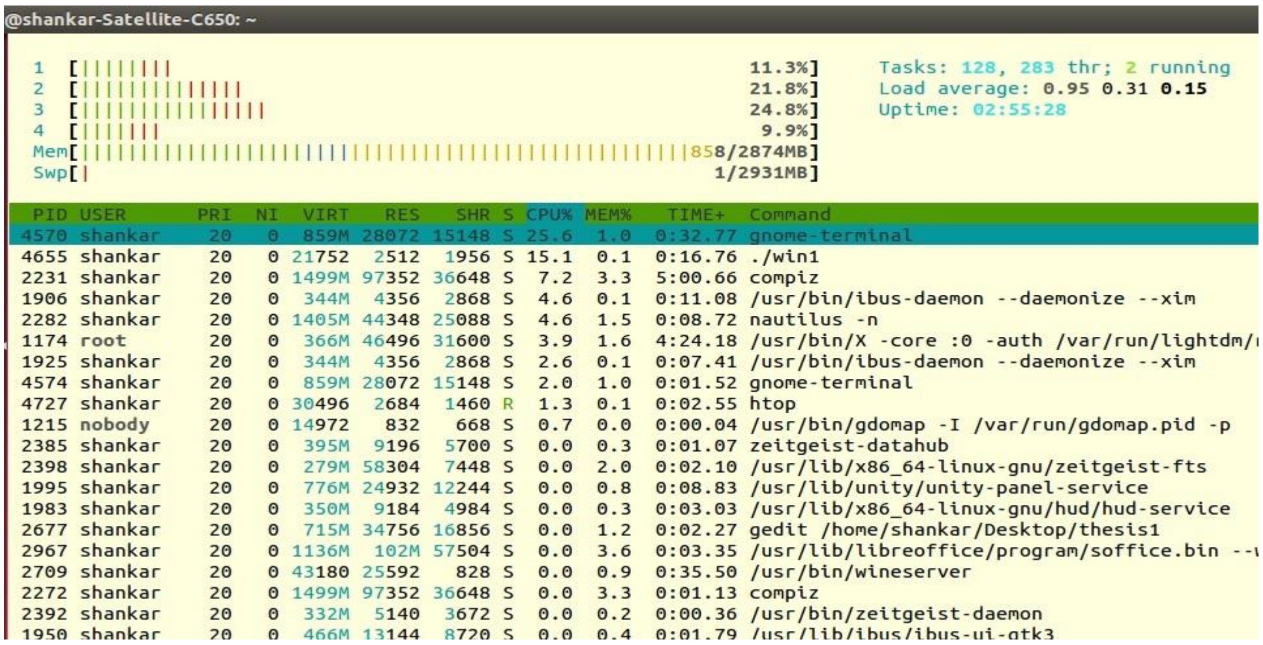


```
shankar@shankar-Satellite-C650: ~

  1  [|                         0.7%]     Tasks: 128, 286 thr; 1 running
  2  [|                         0.6%]     Load average: 0.30 0.33 0.23
  3  [||                        2.6%]     Uptime: 03:03:55
  4  [                          0.0%]
  Mem[||||||||||||||||||    851/2874MB]
  Swp[|                       4/2931MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 5420 shankar    20   0 30092  2192  1456 R  2.0  0.1  0:00.23 htop
 4814 shankar    20   0  642M 20232 12576 S  0.7  0.7  0:03.52 gnome-terminal
 1925 shankar    20   0  345M  4748  2868 S  0.7  0.2  0:11.22 /usr/bin/ibus-dae
 1950 shankar    20   0  466M 13128  8704 S  0.7  0.4  0:03.32 /usr/lib/ibus/ibu
 2131 shankar    20   0  191M  3440  2732 S  0.7  0.1  0:03.41 /usr/lib/ibus/ibu
 2344 shankar    20   0  312M 12948  9296 S  0.7  0.4  0:02.33 /usr/bin/gtk-wind
 1174 root       20   0  407M 51532 36504 S  0.0  1.8  4:32.37 /usr/bin/X -core
 1197 clamav     20   0 52172  2464  1388 S  0.0  0.1  0:11.17 /usr/bin/freshcla
 2231 shankar    20   0 1500M 92448 31720 S  0.0  3.1  5:14.48 compiz
 2742 shankar    20   0 2703M  130M 14292 S  0.0  4.6  0:51.75 C:\Program Files
    1 root       20   0 27228  3008  1412 S  0.0  0.1  0:01.51 /sbin/init
  325 root       20   0 17320   632   448 S  0.0  0.0  0:00.19 upstart-udev-brid
  330 root       20   0 43032  2056   996 S  0.0  0.1  0:00.21 /lib/systemd/syst
  507 root       20   0 15392   704   332 S  0.0  0.0  0:00.07 upstart-socket-br
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

**Figure 5: Shows the effect of running SystemC Application on kernel**



```
@shankar-Satellite-C650: ~

  1  [||||||||                                          11.3%]     Tasks: 128, 283 thr; 2 running
  2  [||||||||||||||                                    21.8%]     Load average: 0.95 0.31 0.15
  3  [||||||||||||||||                                  24.8%]     Uptime: 02:55:28
  4  [|||||||                                            9.9%]
  Mem[||||||||||||||||||||||||||||||||||||||||||||||||||858/2874MB]
  Swp[|                                                 1/2931MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 4570 shankar    20   0  859M 28072 15148 S 25.6  1.0  0:32.77 gnome-terminal
 4655 shankar    20   0 21752  2512  1956 S 15.1  0.1  0:16.76 ./win1
 2231 shankar    20   0 1499M 97352 36648 S  7.2  3.3  5:00.66 compiz
 1906 shankar    20   0  344M  4356  2868 S  4.6  0.1  0:11.08 /usr/bin/ibus-daemon --daemonize --xim
 2282 shankar    20   0 1405M 44348 25088 S  4.6  1.5  0:08.72 nautilus -n
 1174 root       20   0  366M 46496 31600 S  3.9  1.6  4:24.18 /usr/bin/X -core :0 -auth /var/run/lightdm/
 1925 shankar    20   0  344M  4356  2868 S  2.6  0.1  0:07.41 /usr/bin/ibus-daemon --daemonize --xim
 4574 shankar    20   0  859M 28072 15148 S  2.0  1.0  0:01.52 gnome-terminal
 4727 shankar    20   0 30496  2684  1460 R  1.3  0.1  0:02.55 htop
 1215 nobody     20   0 14972   832   668 S  0.7  0.0  0:00.04 /usr/bin/gdomap -I /var/run/gdomap.pid -p
 2385 shankar    20   0  395M  9196  5700 S  0.0  0.3  0:01.07 zeitgeist-datahub
 2398 shankar    20   0  279M 58304  7448 S  0.0  2.0  0:02.10 /usr/lib/x86_64-linux-gnu/zeitgeist-fts
 1995 shankar    20   0  776M 24932 12244 S  0.0  0.8  0:08.83 /usr/lib/unity/unity-panel-service
 1983 shankar    20   0  350M  9184  4984 S  0.0  0.3  0:03.03 /usr/lib/x86_64-linux-gnu/hud/hud-service
 2677 shankar    20   0  715M 34756 16856 S  0.0  1.2  0:02.27 gedit /home/shankar/Desktop/thesis1
 2967 shankar    20   0 1136M  102M 57504 S  0.0  3.6  0:03.35 /usr/lib/libreoffice/program/soffice.bin --
 2709 shankar    20   0 43180 25592   828 S  0.0  0.9  0:35.50 /usr/bin/wineserver
 2272 shankar    20   0 1499M 97352 36648 S  0.0  3.3  0:01.13 compiz
 2392 shankar    20   0  332M  5140  3672 S  0.0  0.2  0:00.36 /usr/bin/zeitgeist-daemon
 1950 shankar    20   0  466M 13144  8720 S  0.0  0.4  0:01.79 /usr/lib/ibus/ibus-ui-gtk3
```

**Figure 6: Shows the effect of running multi-threaded Vlang Application on kernel**

Vlang uses preemptive threads also known as P threads as shown in Figure 6. This means that when a multi-threaded application is written using Vlang, the compiler can borrow more than one thread from the kernel of the operating system. The maximum number of threads which can be borrowed is limited by the number of cores CPU. Once the threads are acquired from the kernel, the application can run its multiple threads using contact switching. Since block B is designed using Vlang, this scenario is common with block B. Whereas Figure 5 describes the scenario of block A.

## 6. CONCLUSION

In today's VLSI industry, the higher level of abstraction has become the necessity. Also, the Amdahl's law needs to be followed. Amdahl law states that in order to attain the maximum speed for the given application, one need to exploit the maximum number of cores of CPU.

This work revolves around the above mentioned facts. Block A is designed to form a transaction containing generic payload using SystemC. Block B is designed to perform the parallel simulation on the different parts of the image and converts its RGB components into gray value.

In case of small multi-threaded application, the SystemC application seems to compile and run faster than Vlang. But in case of more complex multi-threaded application Vlang seems to be faster as compared to SystemC. This is due to the fact that the time required to acquire the multiple threads from the kernel is more as compared to the time required to acquire the single thread from the kernel.

However this paper does not aim to compare the above mentioned languages with reference to specific figures. This is due to the following reasons.

First, to compare two languages, we need to work on some standard real world applications which are written unbiased to both the languages (such as Ethernet, PCI, etc).

Second, mentioning the figures will violate the License conditions. No doubt Vlang claims to have better features as compared to SystemC but as far as single-core is concerned, the choice may vary from user to user and application to application. If the user wants to make his design or Verification IP (VIP) with multi-core enabled feature, Vlang is more prominent.

## 7. REFERENCE

[1] C. Schumacher, J.H. Weinstock, R. Leupers, G. Ascheid, "Legacy SystemC Model Integration into Parallel SystemC Simultors." In the proceedings of *Parallel and Distributed Processign Symposium Workshops & PhD Forum international Conference,* Cambridge, pp. 2188 – 2193, 20 – 24 May 2013.

[2] T. Chan, " A robust multithreaded HDL/ESL simulator for deep submicron integrated circuit designs," In the proceedings of *Circuits and Systems Asia Pacific Conference,* Kaohsiung, pp. 416 – 419, 2 – 5 Dec. 2012.

[3] A. Pulka, L. Golly, "Multitask Real-time Systems modeling in SystemC," In the proceedings of *Signals and Electroni Systems International Conferenc**e**,* Wroclaw, pp. 1 – 6, 18 – 21 Sept. 2012

[4] A. Prakash, H.D. Patel, R. Sinha, " Parallel simulation of mixed – abstraction SystemC models on GPUs and multicore CPUs," In the proceedings of *Design Automation Conference*, Sydney, pp. 455 – 460, 30 Jan. – 2 Feb. 2012.

[5] R.S. Lobato, R. Spolon, M.A. Cavenaghi, R.C. Detomini, "Using GPU to exploit parallelism on cryptography, " In the proceedings of *Information Systems and Technologies Conference*, Chaves, pp. 1 – 6, 15 – 18 June 2011.

[6] T. Chan, "Race logic synthesis for a multithreaded HDL/ESL simulator for Soc designs," In the proceedings of *Circuits and Systems Asia Pacific Conference*, Kuala Lumpur, pp. 1179 – 1182, 6 – 9 Dec. 2010.

[7] Sen, A., "Mutation Operators for concurrent SystemC Designs," In the proceedings of *Microprocessor Test and Verification Conference*, Austin, pp. 27 – 31, 7 – 9 Dec. 2009.

[8] S. Sudharsanan, N. Manjikian, "Modeling and simulation of multicore multithreaded processor architectures in SystemC," In the proceedings of *Electrical and Computer Engineering conference*, Niagara Falls, Ontario, pp. 001155 – 001160, 4 – 7 May 2008.

[9] IEEE Standards for standards System CLanguage Reference Manual, pp.421.