Design of Low Power Globally Asynchronous Locally Synchronous Modeling Using Xilinx

N.SRIKANTH[1], Dr.SACHIN S SHARMA[2]
[1]P.G Student, Department of Electronics and Communication Engineering
[2]Assistant professor, Avn institute of engineering and technology

Abstract- In this paper a novel Globally Asynchronous Locally Synchronous (GALS) modeling and verification tool is introduced for XILINX circuits. The tool provides a structured environment for GALS in which organization of the modeling and verification enables it to handle a variety of implementation tasks facilitating a process which would otherwise be difficult for the end user. The tool provides verification techniques at different levels. A new unfolding algorithm is presented that uses Structured Occurrence nets. A novel representation for deadlocks is introduced using deadlock relations enabling the causality of local and global deadlocks to be visualized. This helps in the investigation of total or partial system shutdown. In particular, the approach enables the visualization of point-to-point causality of problems occurring between different parts of the system which are more difficult to analyze. In addition different types of deadlock related to the synchronizer can be detected. The work presented here provides structured visualization capability facilitating the analysis of complex communication systems.

I. INTRODUCTION

there has been a lot of interest in researching new architectures for GALS [1] [2], there have been few attempts at providing modeling solutions for GAL's communication. Thus, modeling of GALS from specifications has been limited to hardware description languages such as Verilog, VHDL [3] or synchronous programming languages such as C or ESTEREL [4]. Specialist verification languages that have been introduced for GALS include GRL [5] and process calculi [6] but these languages tend to be used at a higher level of abstraction than hardware. A graphical tool has been developed in [7] but the models here are also used at a higher level i.e. they are not used for circuit deadlock analysis. Although the techniques are higher level they offer better modeling of things like protocols. The work in [8] is more similar in the sense that different formats are interchangeable allowing different tools to be linked which is a useful approach to take but is centered on co-simulation rather than verification or deadlock analysis. Hardware models for communication logic in the past have relied on standard languages, e.g. Verilog, which require a significant amount of"glue logic" to connect communicating primitives together. This kind of modeling tends to be unwieldy and non-intuitive. XILINX [9] [10] [11] represents a significant improvement in the representation and modeling of communication systems. It provides a set of graphical communication primitives which are more natural, i.e. they are closer to the hardware, and their higher level of abstraction enables them to be easily understood. Although XILINX model checking has been covered extensively at the Boolean level for purposes like deadlock checking [12] [13] [14] [15] [16] little work has been done using net level models such as Petri nets. In [17] basic techniques for GALS synthesis to Circuit Petri nets [18] for XILINX were presented offering some distinct advantages: they are well suited to the visualization of distributed models of local machines in terms of concurrency and for verification they capture a complete knowledge in the unfolding hence providing a representation of the full causality. In [17] an additional XILINX synchronizer primitive was introduced to provide a synchronization wrapper for synthesizing a range of"glue" solutions e.g. asynchronous, mesochronous, etc. Basic techniques for GALS verification were also presented including unfolding to occurrence nets.

II. LITERATURE SURVEY

Convolutional neural network (CNN), a well-known deep learning architecture extended from artificial neural network, has been extensively adopted in various applications, which include video surveillance, mobile robot vision, image search engine in data centers, etc [6] [7] [8] [10] [14]. Inspired by the behavior of optic nerves in living creatures, a CNN design processes data with multiple layers of neuron connections to achieve high accuracy in image recognition. Recently, rapid growth of modern applications based on deep learning algorithms has further improved research on deep convolutional neural network. Due to the specific computation pattern of CNN, general purpose processors are not efficient for CNN implementation and can hardly meet the performance requirement. Thus, various accelerators based on FPGA, GPU, and even ASIC design have been proposed recently to improve performance of CNN designs [3] [4] [9].
Among these approaches, FPGA based accelerators have attracted more and more attention of researchers because they have advantages of good performance, high energy efficiency, fast development round, and capability of reconfiguration [1] [2] [3] [6] [12] [14]. For any CNN algorithm implementation, there are a lot of potential solutions that result in a huge

design space for exploration. In our experiments, we find that there could be as much as 90% performance difference between two different solutions with the same logic resource utilization of FPGA. It is not trivial to find out the optimal solution, especially when limitations on computation resource and memory bandwidth of an FPGA platform are considered. In fact, if an accelerator structure is not carefully designed, its computing throughput cannot match the memory bandwidth provided an FPGA platform. It means that the performance is degraded due to under-utilization of either logic resource or memory bandwidth. 161 Unfortunately, both advances of FPGA technology and deep learning algorithm aggravate this problem at the same time. On one hand, the increasing logic resources and memory bandwidth provided by state-of-art FPGA platforms enlarge the design space. In addition, when various FPGA optimization techniques, such as loop tiling and transformation, are applied, the design space is further expanded. On the other hand, the scale and complexity of deep learning algorithms keep increasing to meet the requirement of modern applications.

III. PROPOSED SYSTEM

A. GALS Asynchronous Primitive

In addition to the standard XILINX symbols for all the basic primitives an asynchronous synchronisation primitive has been added. The primitive is used for inserting asynchronous”glue” components in communication channels that cross clock domains. The interface signals are defined using the XILINX format so that it can be interfaced to other XILINX primitives. The synchronization primitive is shown in Fig. 1.

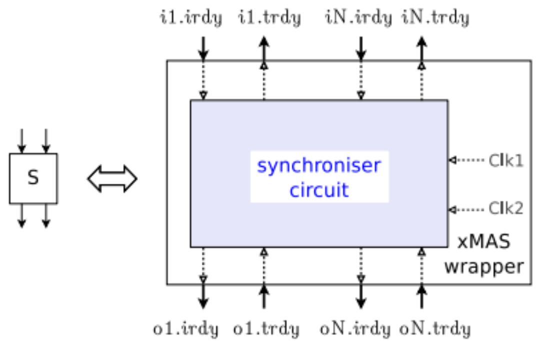


Fig.1: xMAS synchronization primitive.

A synchronization primitive is used for communication between two islands. The synchronization primitive accepts a variable number of send signals, i1.irdy .. iN.irdy, from the incoming primitives from one island and returns the required number of receive signals, i1.trdy .. iN.trdy. Similarly it communicates with the target island by issuing the required number of send signals, o1.irdy .. oN.irdy and by accepting the required number of receive signals, o1.trdy .. oN.trdy. The new asynchronous primitive is generic and incorporates a number of synchronisation schemes. A black box is used to house the specific implementation style used for synchronization, which is designed to accommodate different GALS implementation styles: asynchronous, mesochronous, plausible clocking, etc. [22].

The Circuit Petri net translator is implemented inside our WORKCRAFT tool. The translator accepts a JSON (data interchange format) representation of the XILINX model and translates it into a Petri net representation. For translation each primitive is generated as shown in the examples in Fig. 6 to Fig. 9. The net primitives are connected by a process which links together all external connections. A data line is added later automatically to include data signals. Data is treated as signals e.g. 0.. 1. The signals pass through the data CPNs of the respective gates similar to the examples shown in Fig. 2.

T qload < (T − T qload − T oracle) (1)

Where T is the set of all transitions, T qload is the set of transitions associated with loading/unloading of the queue slots and T oracle is the set of source and sink oracle transitions. Proposition 1: There is a logical equivalence between the CPN primitives and XILINX primitives [equivalence is determined by truth table] Proposition 2: If ζ = enabled (T qload) then each t ∈ ζ must fire in the same step (clock step) as they have a unique common priority level. Lemma1: If t ∈ ζ fires in a single clock step ≡ (read/write) [32] and [χ = (T −T qload−T oracle)] > T qload then the execution semantics of (1) are equivalent to the clocked XILINX execution semantics. Proof: Given χ = communication transitions which execute between clock steps (read/writes) - similar to the transitions islands of [32]. If χ > T qload then T qload cannot fire until all χ have fired even if they are enabled. Therefore, χ will fire in a different step to T qload. If proposition 1 and proposition 2 hold it follows due to the alternating firing of χ and T qload that the execution semantics of (1) are equivalent to the XILINX execution semantics. In addition the source and sink oracles must occur as a multiple of queue transfers. For this the following relation is required:

T oracle < (T − T qload − T oracle) (2)

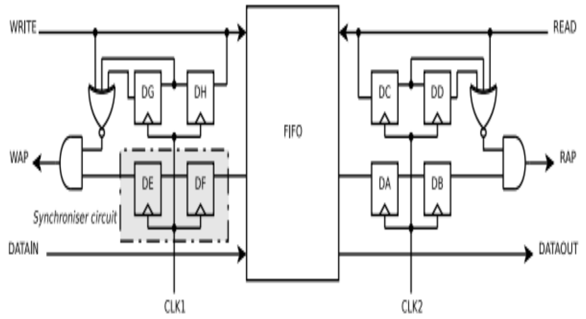
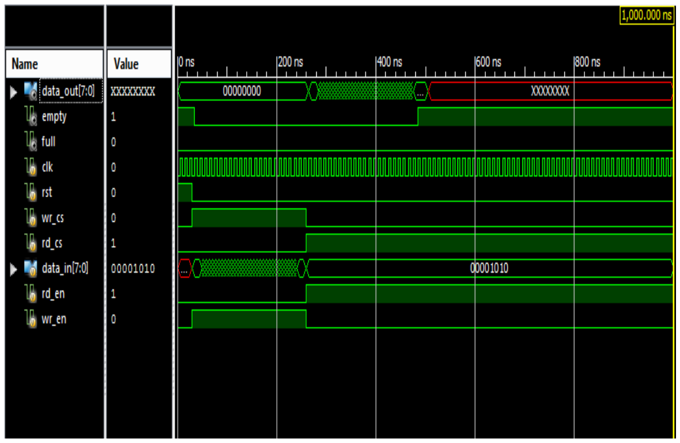


Fig.2: Asynchronous synchronization.

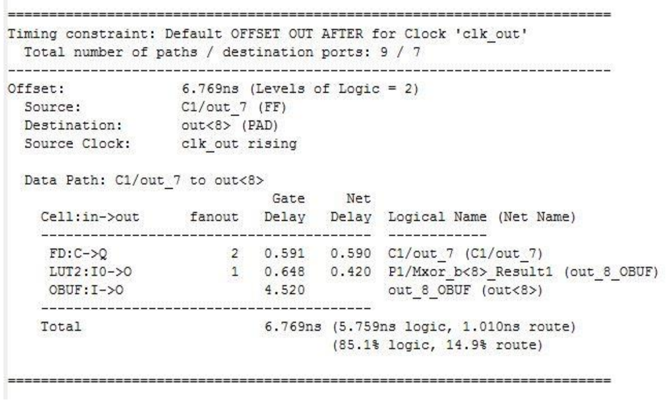
which assigns the oracles a lower priority than all signals apart from the queue: if the source and sink are eager they are activated once at the start via the system control signals (see Fig. 6); if they are non-deterministic then according to (2) they can only be activated when the communication signals other than the queue have already been activated. Lemma 2: If each t ∈ enabled (T oracle) fires in a single clock step and [T − T qload − T oracle] > T oracle then the execution semantics of (2) are equivalent to the XILINX execution semantics. Proof: the proof follows from lemma1 with T qload switched with T oracle i.e. the same must hold for the sources and sinks as holds for the queues. Finally a specific mapping Π = ΠQO is required which relates queue to source and sink oracles: ΠQO = (T oracle = T qload if eager T oracle ≤ T qload if non-deterministic (3) Here ΠQO represents a priority mapping relation between the queue and the source and sink oracles. The first part of the expression operates in a similar manner to (2) with regards initialization i.e. the oracles are only activated once at the start if they are eager. The second part of the expression in (3) dynamically sets the prioritization of the oracle to be less than or equal to the queue depending on the non-deterministic setting that is generated by the system for each oracle. Theorem 1: If Case 1: T oracle = T qload (the setting of the oracle is eager) or Case 2: T oracle ≤ T qload (the setting of the oracle is non-deterministic), then for either case, Case 1 or Case 2, the execution semantics must be equivalent to the XILINX execution semantics.

IV. SIMULATION RESULTS

A. WAVEFORMS

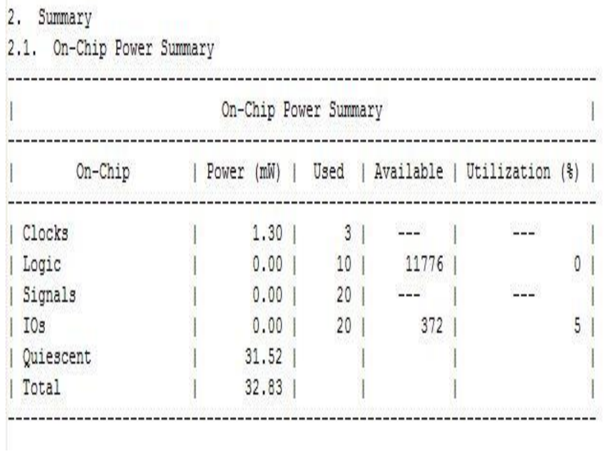


B. TIMING REPORT



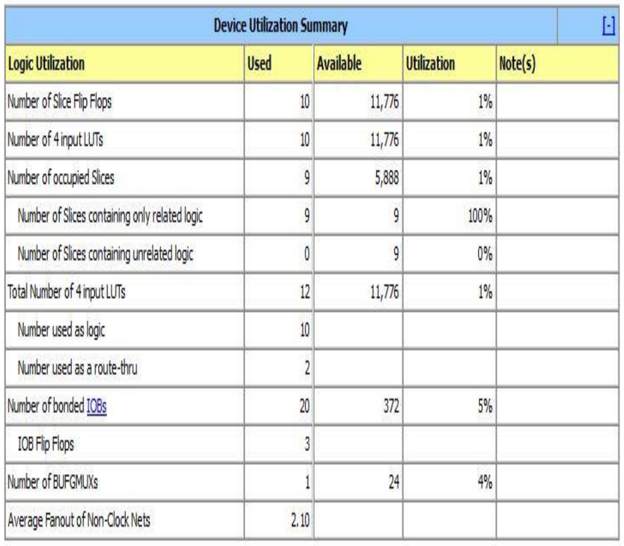
```
================================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk_out'
  Total number of paths / destination ports: 9 / 7
--------------------------------------------------------------------------------
Offset:              6.769ns (Levels of Logic = 2)
  Source:            C1/out_7 (FF)
  Destination:       out<8> (PAD)
  Source Clock:      clk_out rising

  Data Path: C1/out_7 to out<8>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FD:C->Q               2   0.591   0.590  C1/out_7 (C1/out_7)
     LUT2:I0->O            1   0.648   0.420  P1/Mxor_b<8>_Result1 (out_8_OBUF)
     OBUF:I->O                 4.520          out_8_OBUF (out<8>)
    ----------------------------------------
    Total                      6.769ns (5.759ns logic, 1.010ns route)
                                       (85.1% logic, 14.9% route)

================================================================================
```

C. POWER REPORT



```
2.  Summary
2.1.  On-Chip Power Summary
-----------------------------------------------------------------------------
|                           On-Chip Power Summary                           |
-----------------------------------------------------------------------------
|        On-Chip        | Power (mW) |  Used  | Available | Utilization (%) |
-----------------------------------------------------------------------------
| Clocks                |       1.30 |      3 |    ---    |       ---       |
| Logic                 |       0.00 |     10 |     11776 |               0 |
| Signals               |       0.00 |     20 |    ---    |       ---       |
| IOs                   |       0.00 |     20 |       372 |               5 |
| Quiescent             |      31.52 |        |           |                 |
| Total                 |      32.83 |        |           |                 |
-----------------------------------------------------------------------------
```

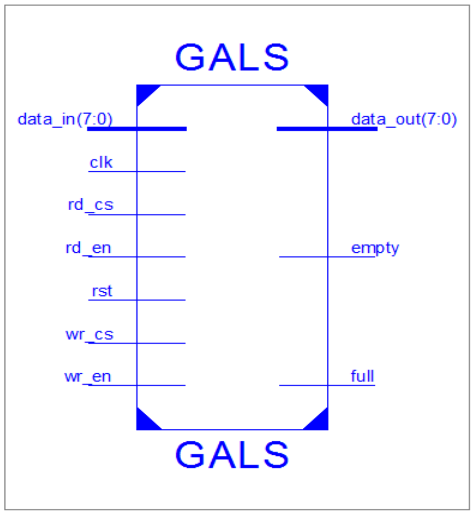
D. DESIGN SUMMARY



Device Utilization Summary [-]

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	10	11,776	1%	
Number of 4 input LUTs	10	11,776	1%	
Number of occupied Slices	9	5,888	1%	
Number of Slices containing only related logic	9	9	100%	
Number of Slices containing unrelated logic	0	9	0%	
Total Number of 4 input LUTs	12	11,776	1%	
Number used as logic	10			
Number used as a route-thru	2			
Number of bonded IOBs	20	372	5%	
IOB Flip Flops	3			
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	2.10			

E. RTL SCHEMATIC



V. CONCLUSION

We have introduced a structured visual GALS modeling and verification environment for communication circuits. An integrated GALS platform has been provided using the WORKCRAFT tool which allows a comprehensive approach by integrating multiple tasks into a unified environment. The visualization capabilities provide enhanced feedback to the user during verification making it much easier for the user to investigate the causality of problems. The verification approach is based on unfolding and deadlock analysis using Structured Occurrence nets which is well suited for GALS analysis. A novel representation has been presented using deadlock relations which enables the point-to-point causality between deadlocks to be viewed. This includes analysis of local and global deadlocks and enables visualization of total or partial system shutdown. Results show that deadlocks can be visualized easily and resolved efficiently. For future work we intend to adapt the system to check for lovelock and test a wider range of synchronizers including MUTEX based.

VI. REFERENCES

[1]. S. Suhaib, D. Mathaikutty and S. Shukla, “Dataflow Architectures for GALS,” ACM Journal. Electronic Notes in Theoretical Computer Science (ENTCS), Vol 200, No. 1, pp. 33–50, 2008.

[2]. T. Jungeblut, J. Ax, M. Porrmann and U. Ruckert, “A TCM-based architecture for GALS NoCs,” Proceedings of ISCAS’2012, pp. 2721– 2724, 2012.

[3]. A. Yakovlev, P. Vivet and M. Renaudin, “Advances in asynchronous logic: from principles to GALS and NOC, recent industry applications, and commercial CAD tools,” Proceedings of DATE’2013, 2013.

[4]. C. Koch-Hofer, Y. Renaudin, Y. Thonnart and P. Vivet, “ASC, a System C extension for modelling asynchronous systems, and its application to an asynchronous NOC,” Proc. on Networks-on-Chip NOCS’2007, pp. 295– 306, 2007.

[5]. Fatma Jebali, Frdric Lang and Radu Mateescu, “A Specification Language for Globally Asynchronous Locally Synchronous Systems,” Proc. ICFEM’2014, pp. 219–234, 2014.