A Structured Visual Approach to GALS Modeling for Communication Circuits

[1]Ravula S S V Tulasi Raja Priyanthi (M.Tech)[2] k.surya kumarai (M.Tech,(Ph.D), Assistant Professor)
[1,2]Pragati Engineering College, Surampalem, East godavari, Andhra Pradesh, INDIA

[1]rssvtrpriyanthi@gmail.com [2]kumarisep19@gmail.com

Abstract: In this paper, a novel globally asynchronous locally synchronous (GALS) modeling and verification is exploited in presenting a new unfolding algorithm that uses structured occurrence nets. A novel representation for deadlocks is introduced using deadlock relations enabling the causality of local and global dead-locks 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. Synthesis and Simulation is done using Xilinx tool.

Index Terms— Analysis, model checking, GALS modeling, verification.

I. INTRODUCTION

WHILST there has been a lot of interest in researching new architectures for globally asynchronous locally synchronous (GALS), there have been few attempts at providing modeling solutions for GALS communication. Thus, modeling of GALS from specifications has been limited to hardware description languages such as Verilog, VHDL, or synchronous programming languages such as C or ESTEREL. Specialist verification languages that have been introduced for GALS include GALS representation language and process calculi 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. xMAS model checking has been covered extensively at the Boolean level for purposes like deadlock checking little work has been done using net level models such as Petri nets. Basic techniques for GALS verification were also presented including unfolding to occurrence nets.

Introduction to GALS Circuits.

The increased complexity of digital circuits leads to severe challenges in the design process. Most modern digital systems are implemented as SoCs. Consequently, system integration has become a crucial problem. The modern design flow should incorporate all possible tools for coping with these issues. A promising option for dealing with such design challenges is the deployment of globally asynchronous, locally synchronous (GALS) systems. A GALS system consists of complex digital blocks operating synchronously. Those blocks are usually developed using standard synchronous CAD tools and design flow. However, the operation of the blocks is not mutually synchronized hence the term locally synchronous. These locally synchronous blocks communicate with one another asynchronously; on the block level (globally), the system is asynchronous. A common approach is to add an asynchronous wrapper, which provides an interface from the synchronous to the asynchronous environment (and vice versa), to every locally synchronous block. The intention of such an environment is to make it easier to gain visual insight into the causality of complex structural problems that may arise, such as deadlocks in GALS systems.

The main contributions of this paper are as follows.

1)A workflow for structured visual modeling and verification of GALS communication circuits.

2) A new approach to deadlock analysis based on deadlock relations.

II. LITERATURE REVIEW

Verification of GALS Systems by Combining Synchronous Languages and Process Calculi.

A Gals (Globally Asynchronous Locally Synchronous) system typically consists of a collection of sequential, deterministic components that execute concurrently and communicate using slow or unreliable channels.

This paper proposes a general approach for modeling and verifying Gals systems using a combination of synchronous languages (for the sequential components) and process calculi (for communication channels and asynchronous concurrency). This approach is illustrated with an industrial case-study provided by Airbus: a TftpUdp communication protocol between a plane and the ground, which is modeled using the Eclipse/Topcased workbench for model-driven engineering and then analyzed formally using the Cadp verification and performance evaluation toolbox.

Verifying deadlock freedom of communication fabrics.

Avoiding message dependent deadlocks in communication fabrics is critical for modern microarchitectures. If discovered late in the design cycle, deadlocks lead to missed project deadlines and suboptimal design decisions. One approach to avoid this problem is to get high level of confidence on an early microarchitectural model. However, formal proofs of liveness even on abstract models are hard due to large number of queues and distributed control. In this work we address liveness verification of communication fabrics described in the form of high-level microarchitectural models which use a small set of well-defined primitives. We prove that under certain realistic restrictions, deadlock freedom can be reduced to unsatisfiability of a system of Boolean equations. Using this approach, we have automatically verified liveness of several non-trivial models (derived from industrial microarchitectures), where state-of-the-art model checkers failed and pen and paper proofs were either tedious or unknown.

III. GALS MODELING

Several GALS methods address the problem of safe and reliable data transfer between independent clock domains. Taxonomy based on the hardware architecture used to transfer data safely. This leads to three main strategies for implementing GALS systems:

- Pausible-clock generators—applying local (pausible, stretchable, or data-driven) clocking, which avoids metastability by ensuring no clock pulses are generated when data is transferred.
- FIFO buffers—using asynchronous FIFO buffers between locally synchronous blocks to hide the synchronization problem.
- Boundary synchronization—performing boundary synchronization on the signals crossing the borders of the locally synchronous island without stopping the complete locally synchronous block during data transfer.

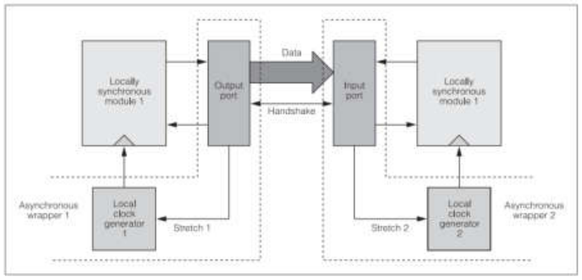


Figure 1. Globally asynchronous, locally synchronous (GALS) system with pausible clocking

GALS Asynchronous Primitive:

In addition to the standard xMAS symbols for all the basic primitives an asynchronous synchronization 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 xMAS format so that it can be interfaced to other xMAS primitives. The synchronization primitive is shown in Fig. 2. The new asynchronous primitive is generic and incorporates a number of synchronization 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, pausible clocking, etc.

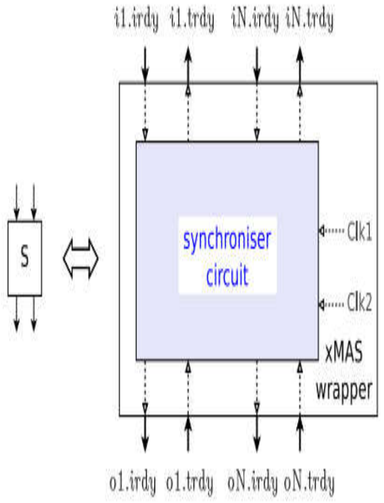


Fig.2 synchronization primitive

Synchronizer Modeling.

The basic synchronizer schemes provided by the tool are as follows.

1) Asynchronous—an implementation based on the use of synchronizers to transfer signals arriving from an outside timing domain to the local timing domain, e.g., two flipflops to synchronize a signal with the local clock.

2) Mesochronous—an implementation in which clocks are derived from the same source and the bounds on the frequencies of communicating blocks are exploited to meet the timing requirements.

3) Pausible—an implementation based on ring oscillators in which each locally synchronous block generates its own clock with a ring oscillator. An implementation style that is provided for the asynchronous scheme is shown in Fig.3. The implementation in Fig. 3 uses a first in first out (FIFO) and synchronizer circuits to transfer signals between the global timing domain and the local timing domain. In this implementation the FIFO buffer handshake signals may be asserted at any time relative to the transmitter or receiver clocks. The net model used for a synchronizer circuit which uses two flip-flops is depicted in Fig.4. In Fig. 4, for the pair of signals comprising the synchronizer circuit, the transitions have a level of prioritization which

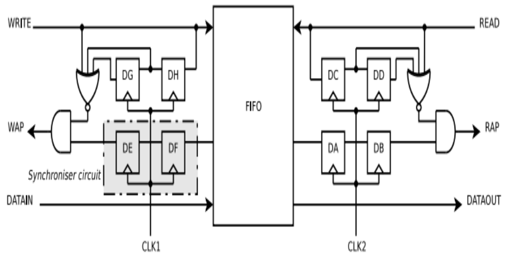


Fig. 3. Asynchronous synchronization.

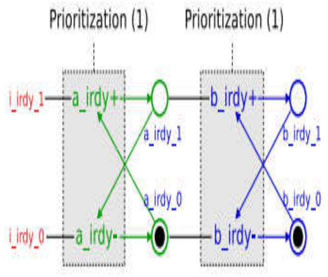


Fig. 4. Synchronizer circuit.

is similar to the level that was assigned to the queue loading in (1). This is required in order to give the flip-flops a lower priority than the other communication signals.

IV. PROPOSED SYSTEM

The xMAS models are verified by a process of unfolding to occurrence nets and deadlock analysis. For normal verifica-tion as used in the unfolding proceeds to occurrence nets using basic GALS unfolding. For structured net verification the net is translated from xMAS but the unfolding is made to SONs rather than Occurrence nets. A SON is a set of related occurrence nets linked together by specific types of relation. The type of relation determines the class of the SON.

The CSONs are limited to 1-safe communication only. One-safe implies a limit of one token per place in the CPN which is a necessary requirement for CPNs. A CSON is a tuple CSON = $(ON_1, \ldots, ON_k, P_0, l_0, F_0)$ consisting of k occurrence nets where $k \geq 1$ such that each $ON_i = (P_i, T_i, F_i, l_i)$ is an occurrence net, P_0 is a set of channel places (CPs) linking the occurrence nets together (the communication occurs across the CPs), l_0 is a labeling of P_0 and a flow relation

$F_0 \subseteq (T \times P_0) \cup (P_0 \times T)$, where $T =_{i\geq 1} T_i$.

A. Unfolding

For unfolding the GALS model is mapped to SONs and the local modules L_N are mapped to ordinary occurrence nets. The nets are mapped in a specific way to reflect the unique GALS structure. In the unfolding process labeled events are assigned accordingly; all labeled events that belong to each local module L_N are assigned to an occurrence net ON_N and all labeled events that belong to each $s \in S$ are assigned to ON_S. The irdy and trdy control signals corresponding to the ports of the local occurrence nets are linked directly to CPs. Similarly, the irdy and trdy control signals of the synchronizer wrapper of Fig. 2 are linked to the corresponding CPs. This is depicted in Fig.5 in which two local modules L_A and L_B are mapped to ON_A and ON_B, respectively, and connected to ON_S via CPs.

Corresponding to Fig. 5 the prefix is prepartitioned into two segments ON_L and ON_S. All transitions with the highest priority are processed first.

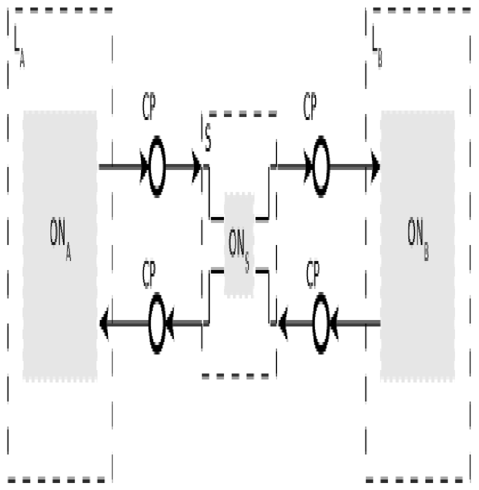


Fig. 5. CSON diagram.

As the unfolding proceeds the events are assigned to their partitions, respectively; the events e_L of all local transitions of t_A are assigned to ON_L and the events of all communication transitions e_S are assigned to ON_S. Any s ∊ postset corresponding to syncIO are converted to CPs. Here syncIO refers to synchronizer inputs and outputs. For different clock domains the unfolding in each ON_L is set at different rates according to the rate of queue firing. This is set in the unfolding algorithm by controlling the addition of the xMAS queue transitions which are clock sensitive at different rates according to the relative frequencies of the local partitions in which they reside. This also holds for the clock sensitive signals inside the synchronizers.

B. Deadlock Analysis

Deadlock checking is made at the communication level and modular level by analysis of the CSON model and localized occurrence nets. The concept of a local deadlock was introduced in terms of dead channels in which they define the concept of local deadlock based on sections of the model that become permanently inactive. These types of deadlock are split into two different types:

1) *blocking* where trdy signals become permanently inactive and

2) *idle* where irdy signals become permanently inactive.

EXTENSION.

DIGITAL CLOCK:

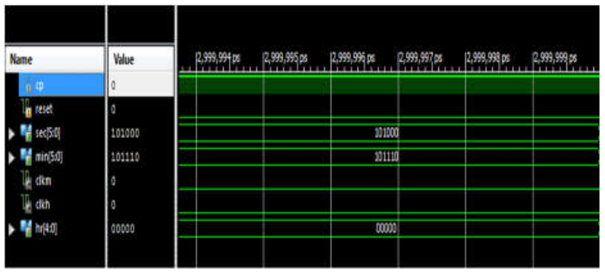
Digital clock is a type of clock that displays the time digitally, i.e. in ciphers, as opposed to an analog clock, where the time is displayed by hands. Digital clocks are often associated with electronics drives, but the digital description refers only to the display, not to the drive mechanism (Both analog and digital clocks can be driven either mechanically or electronically, but clockwork mechanisms with digital displays are rare).

Construction of digital clock Digital clocks typically use the 50 or 60 hertz oscillation of AC power or a 32.768kHz crystal oscillator. A more commonly used hour sequence option is 12 hour format (with some indication of AM or PM). Emulations of analog-style faces often use an LCD screen, and these are also sometimes described as digital.

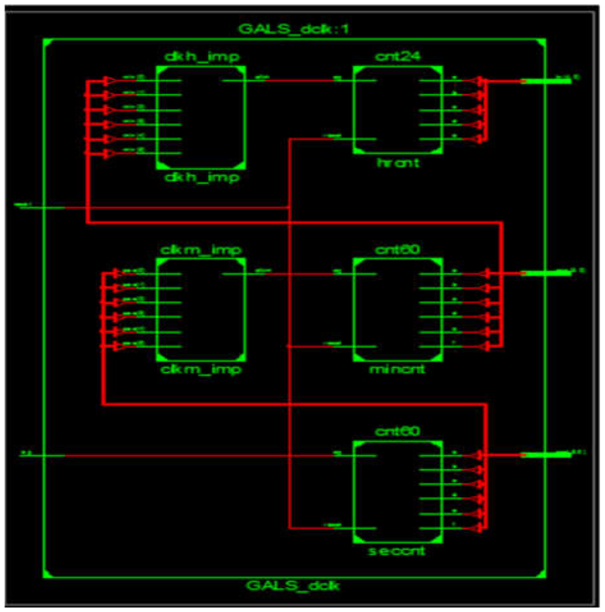
VI. RESULTS

The Verilog HDL Modules have successfully simulate, verified and synthesized using Xilinxise13.2.

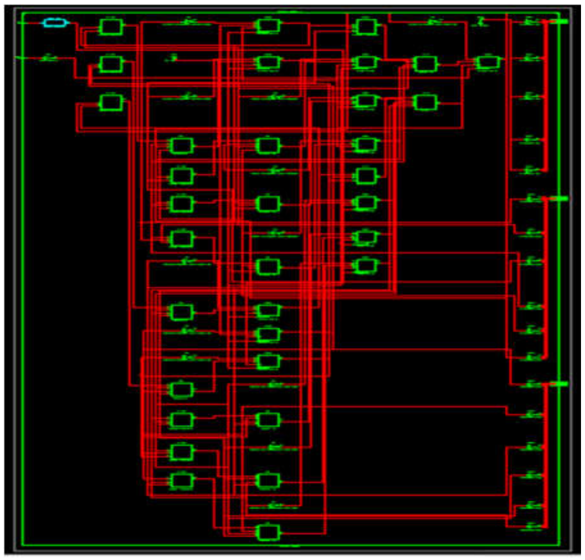
Simulation.



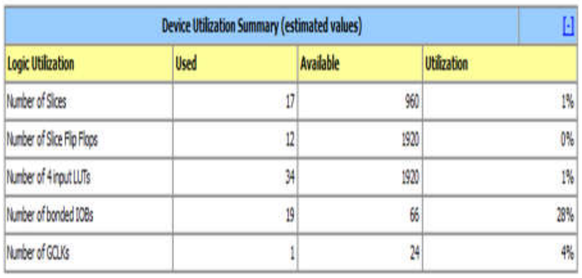
RTL Schematic.



Technology summary

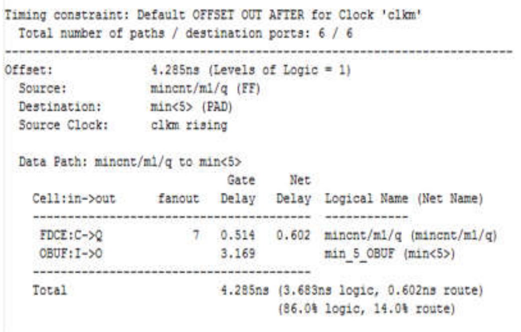


Design Summary.



Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	17	960	1%
Number of Slice Flip Flops	12	1920	0%
Number of 4 input LUTs	34	1920	1%
Number of bonded IOBs	19	66	28%
Number of GCLKs	1	24	4%

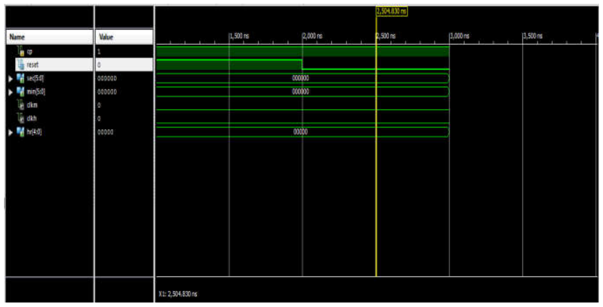
Timing Summary.



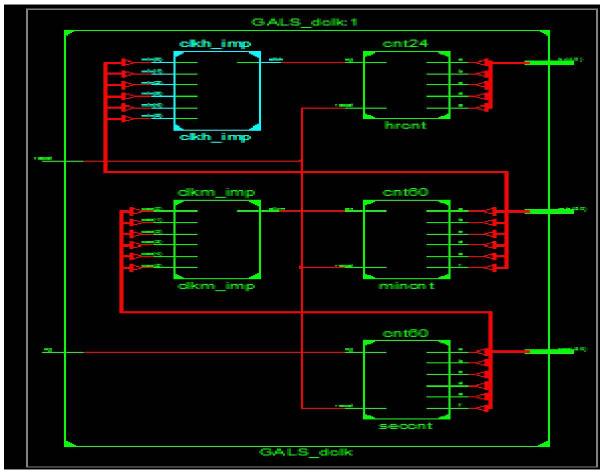
```
Timing constraint: Default OFFSET OUT AFTER for Clock 'clkm'
  Total number of paths / destination ports: 6 / 6
-------------------------------------------------------------------------
Offset:              4.285ns (Levels of Logic = 1)
  Source:            mincnt/m1/q (FF)
  Destination:       min<5> (PAD)
  Source Clock:      clkm rising

  Data Path: mincnt/m1/q to min<5>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDCE:C->Q             7   0.514   0.602  mincnt/m1/q (mincnt/m1/q)
     OBUF:I->O                 3.169          min_5_OBUF (min<5>)
    ----------------------------------------
    Total                      4.285ns (3.683ns logic, 0.602ns route)
                                       (86.0% logic, 14.0% route)
```

EXTENSION.

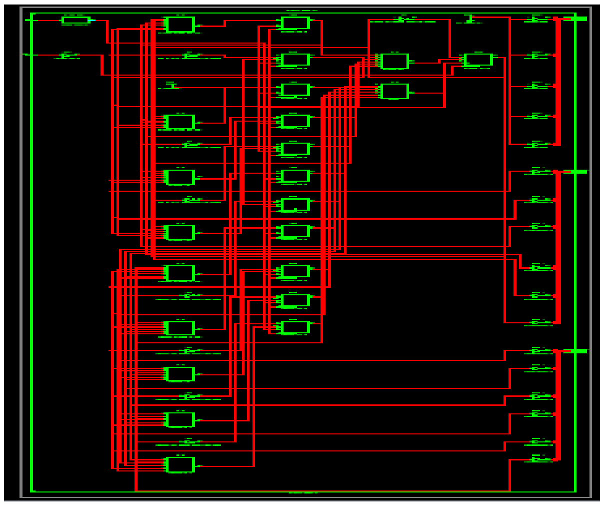
Simulation.



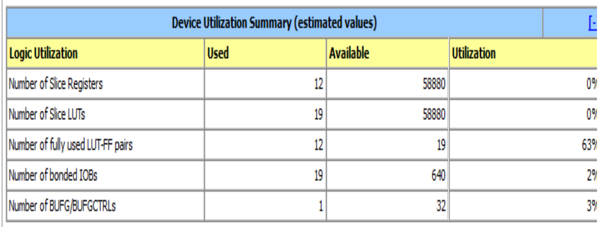
RTL Schematic.



Technology summary

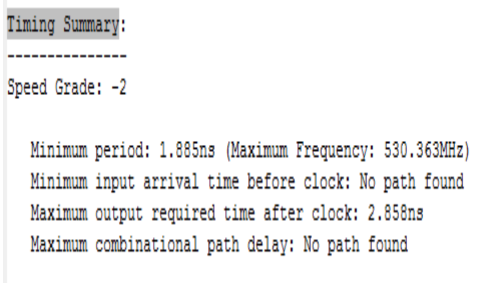


Design Summary.



Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice Registers	12	58880	0%
Number of Slice LUTs	19	58880	0%
Number of fully used LUT-FF pairs	12	19	63%
Number of bonded IOBs	19	640	2%
Number of BUFG/BUFGCTRLs	1	32	3%

Timing Summary.



```
Timing Summary:
---------------
Speed Grade: -2

   Minimum period: 1.885ns (Maximum Frequency: 530.363MHz)
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: 2.858ns
   Maximum combinational path delay: No path found
```

VII. CONCLUSION

In this paper, we have presented DLAU, which is a scalable and flexible deep learning accelerator based on FPGA. The DLAU includes three pipelined processing units, which can be reused for large scale neural networks. The proposed DLAU uses carry save adder in the computation process and as further process of the project to increase the computation speed parallel self timed adder is used in the design. Experimental results on Xilinx FPGA prototype show that DLAU accelerator achieved more speed when compared to other devices like ASIC.

VIII. REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] J. Hauswald *et al.*, "DjiNN and Tonic: DNN as a service and its impli-cations for future warehouse scale computers," in *Proc. ISCA*, Portland, OR, USA, 2015, pp. 27–40.

[3] C. Zhang *et al.*, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. FPGA*, Monterey, CA, USA, 2015, pp. 161–170.

[4] P. Thibodeau. *Data Centers are the New Polluters*. Accessed on Apr. 4, 2016. [Online]. Available: http://www.computerworld.com/ article/2598562/data-center/data-centers-are-the-new-polluters.html

[5] D. L. Ly and P. Chow, "A high-performance FPGA architecture for restricted Boltzmann machines," in *Proc. FPGA*, Monterey, CA, USA, 2009, pp. 73–82.

[6] T. Chen *et al.*, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. ASPLOS*, Salt Lake City, UT, USA, 2014, pp. 269–284.

[7] S. K. Kim, L. C. McAfee, P. L. McMahon, and K. Olukotun, "A highly scalable restricted Boltzmann machine FPGA implementation," in *Proc. FPL*, Prague, Czech Republic, 2009, pp. 367–372.

[8] Q. Yu, C. Wang, X. Ma, X. Li, and X. Zhou, "A deep learning prediction process accelerator based FPGA," in *Proc. CCGRID*, Shenzhen, China, 2015, pp. 1159–1162.

[9] J. Qiu *et al.*, "Going deeper with embedded FPGA platform for con-volutional neural network," in *Proc. FPGA*, Monterey, CA, USA, 2016, pp. 26–35