Evaluating the Partitioning of GALS Systems Using Abstract System Level Modeling

SeyedMoein Hosseini Mehrdad Najibi Kamran Saleh Hossein Pedram

{mhosseini, najibi, k.saleh, pedram}@ce.aut.ac.ir

Computer Engineering Department, Amirkabir University of Technology (Tehran Polytechnic)

424 Hafez Ave, Tehran 15785, Iran

ABSTRACT

The complexity of digital design and time-to-market have arose many challenges in synchronous design methodology; the need for high frequency and low skew clock distribution with its profound effect on final circuits take a lot of time and implementation cost. Asynchronous design methodology by eliminating global clock and replacing synchronization with handshaking is not yet much promising for current microelectronic industries due to the lack of expert designers and design tools. Globally Asynchronous Locally Synchronous design methodology by partitioning a full synchronous design into smaller parts which communicate trough asynchronous media can overcome lot of these problems. While partitioning as the first step in GALS design is shown to have an intense effect on final circuit quality in term of performance and power consumption, it requires itself precise abstract models for performance and power. In this work we address such a high-level model for finding the best partitioning scheme among various options in term of performance. We modeled a Reed-Solomon Decoder with two partitioning schemes to show that our result fairly mimic the actual metrics. It is also shown that our model can also be used for power estimation in case of Reed-Solomon but we left generalized power estimation for future works.

Keywords

GALS, Partition Evaluation, Performance

1. INTRODUCTION

Recent progresses in technology and increasing in complexity of digital design, leads to needs for higher frequency and larger die size which complicates the problem of distributing the clock signal over chip area with a low skew. The need for low power implementations and the correlation between clock frequency and the amount of power loss worsen the situation. In average up to 40% of power dissipation in current digital systems corresponds to the clock distribution tree [1].

On the other hand, while design reuse is a necessity for decreasing the time to market of new products, in synchronous design methodologies the problem of synchronization limits the effectiveness of this approach. Asynchronous Design methodologies however, can overcome all synchronization problems by using handshaking instead of global clock signal which is not yet completely practical due to the lack of expert designers and supporting design tools.

Globally Asynchronous Locally Synchronous (GALS) scheme [2] shows to be a promising solution to fills the gap between these two approaches and it is shown that it can simplify the clock skew problem by decreasing the depth of the clock tree and also achieves lower power in some useful applications.

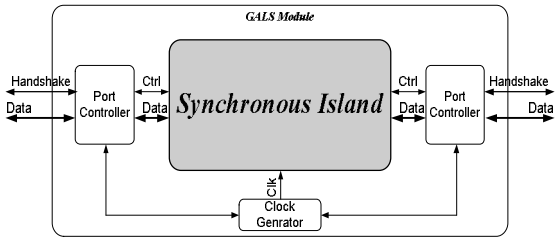

Figure 1. General GALS module

Each GALS system is composed of islands of Locally Synchronous (LS) modules communicating via global asynchronous wrapper circuits to exchange data.. Each LS module is surrounded by a wrapper which is responsible for eliminating metastability problems by controlling the local clock of each synchronous island. Figure 1 shows the general scheme of the LS modules in GALS systems.

The design of a GALS system generally starts with partitioning a fully designed and tested synchronous circuit. Availability of a large number of verified synchronous IP blocks is one of the motivations behind this approach. In GALS jargon each partition is named Locally Synchronous Island (LSI). This process includes partitioning the data path, and decomposing the central controller to generate autonomous synchronous islands.

In general, partitioning of synchronous circuits has deep effect on the performance and the power dissipation of the final GALS implementation. The functional behavior of the partitioned modules can highly impact the efficiency of GALS circuit. Furthermore, every partition needs to include some additional control signals for handshaking to the wrappers.

Because of these two reasons, partitioning for GALS Implementation must be done in RT level and it is not possible to be postponed to the circuit level. Up to now, Hemani et al in [1] proposed a partitioning model for GALS systems which only considered clock depth in partitioning. In [9] optimum partitioning of processor arrays, in term of power as an objective, is studied but the work was not considered general case design. In [8], the effect of different asynchronous interconnects on power and performance using Transaction Level Modeling (TLM) was studied. They modeled software defined radio design in a systemC detailed functional manner in five distinct LSI and compared effect of Pausible Clock Control and FIFO asynchronous interconnects on performance and power, but they overlooked the effect of partitioning on the overall quality of the design.

In this work similar to [8], we used Transaction Level Modeling but for evaluating the effects of different partitioning schemes on performance of a GALS system. In addition, our model is very abstract in contrast to the detailed functional model of [8] which can be extremely advantageous in reducing analysis time. We implement another type of GALS point to point interconnect which will be discussed in subsequent section in term of its clock gating property. Finally we applied our model to a Reed Solomon Decoder and evaluated two partitioning scheme of it and showed that one partition is better in performance and power consumption than the other one. After all we synthesized these two partitioning schemes to verify our model by post synthesis simulation.

2. GALS WRAPPER

Each GALS wrapper contains a local clock generator and required number of port controllers.

A globally asynchronous circuit requires asynchronous communication schemes. Asynchronous communications can use two-phase or four-phase handshaking protocol and ports can act in active or passive manner[4]. Another characteristic of an asynchronous communication controller is the encoding mechanism used for data transfer. One of the most famous techniques is the bundled data model, which is used usually for global communication due to its lower overhead. In this case, data lines are grouped with a request or strobe line which shows validity of data on the channel. The delay of the request line must be greater than all the data lines to ensure a safe communication. However, this is not so important because the delay of the control part that generates the request signal is usually larger than the delay of the data path.

There are two main types of port controllers in designing GALS circuits: demand controllers, and poll controllers[3,5,6].In the demand port controllers, it is assumed that the data being transferred is required immediately after communication. So, in this type of controllers the clock of the LS module should be immediately stopped and it is reactivated when the communication is made. On the other hand, poll type port controller does not stop the clock immediately and so the LS circuit has the opportunity to continue its operation to do some other useful task. This type of port controller targets to gain higher performance, but its efficacy highly depends on the nature of the task and the circuit. So this type of controller can't

be used extensively. Due to power saving by their clock-gating property we decided to implement our system with D-ports only.

Input and output D-port controllers are depicted in Figure 2. The asynchronous channels used in this system are push channels, this means that all inputs are passive and all outputs are active[6]. Globally asynchronous communication channels use four-phase bundled data handshake protocol, on the other hand Den signal uses two-phase handshake protocol to make the system capable of performing a data transfer in each clock cycle.

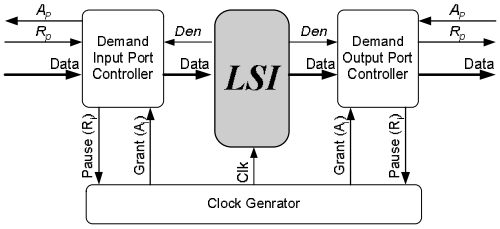

Figure 2. Input and output D-type port controller

An input or output data transfer operation can be summarized as follows: the LS module toggles Den signal to indicate a new data transfer request. Upon receiving an event on Den signal, port controller activates R_i+, and requests to stop the clock. Local clock generator stops the clock as soon as receiving the pause request and acknowledges the requesting port controller that the clock is safely stopped.

Hereafter, the input and output ports behave differently as follows:

The output port controller activates R_p+ to request a data transfer on its asynchronous data channel. On completion of the four phase asynchronous handshake, the controller inactivates R_i to cancel out its clock pause request. The port controller's job is finished by receiving A_i- and A_p- which indicates that the clock is resumed and communication is finished.

The input port controller waits to receive R_p+ and A_i+ to be assured that there is a data on the channel and local clock is stopped. It acknowledges by A_p+ and latches the received data. The clock is resumed after the completion of data transfer on the asynchronous channel and the circuit returns to its initial state.

The main idea of the clock generator circuit is to postpone the rising edge of the clock signal until the completion of input-output communication actions. On receiving a request to pause the clock on R_i, the clock generator suspends the next rising edge of the clock and acknowledges the requesting port controller by activating A_i. When the communication is finished resumes the clock.

3. IMPLEMENTING THE MODEL

Our proposed system level model for GALS systems is based on Transaction Level Modeling (TLM) introduced in [10] and is implemented in systemC. Based on TLM we considered two elements in modeling each system; the first element is computation unit and the second is communication unit. In our implementation, computation and communication units are both abstract models. Computation units are modeled time/cycle accurately while communication units are approximately modeled. We will show that for our needs in performance and power evaluation, this accuracy would be enough.

```
class asyncwrapper : public sc_channel, public
async_write_if, public async_read_if
{
   public:

asyncwrapper(sc_module_namename):sc_channel(name
){}
    void write(CGalsToken* pToken){…}
    CGalsToken* read(){…}
};
```

Figure 3. Asynchronous wrapper

Communication units are implemented using systemC channel mechanism. The code snippet in Figure 3shows how we have implemented two SystemC interfaces, async_write_if and async_read_if. In addition to these interfaces we have two functions for reading and writing from and to the channels.

When a module in one side of a channel invokes the read method, it will be blocked until its partner module on the other side of the channel invokes a write method and vice versa. After invoking write method(or read method in case of write method waiting), data transfer will be initiated which is assumed to take 2.1 ns as a reasonable time required for transferring data in real implementations, Then the channel and two corresponding modules connected to it will be reactivated. This mechanism mimics the behavior of a D-port.

As already mentioned, computation unit is implemented as a time/cycle accurate model. For modeling such a component in TLM, number of computation cycles must be known.

Figure 4 shows our implementation of a simple computation unit. Here we have a systemC module containing three ports; two for sending and receiving tokens and one port for the clock signal. All the information we need to know here, is the number of computation cycles of the module which in our model is translated to wait time needed by the module to produce a token on its output. This wait time can be a deterministic value or can be statistical value. This info can be extracted from RT level code of the synchronous module or it can be provided by designer. The frequency of clock which is another parameter of the model can be back annotated from synthesis results into top level module.The simple model, as shown in Figure 4, is capable of being extended to any number of easily defined input and output ports.

The body of each computation unit is a process which does all the processing. In this process we have an infinite loop includes three sequential parts: The first part is responsible for invoking the read method on input port to provide required data tokens for the computation unit. The second part is a waiting mechanism to mimic required number of computation cycles. The last part is corresponding to wait statements for writing tokens to the output port.

```
class module_name : public sc_module
{

   public:

     sc_port<async_read_if> in;
     //asynchronous input

     sc_port<async_write_if> out;
     //asynchronous output

     sc_in_clk Clk;
     // clock

     SC_HAS_PROCESS(module_name);
     module_name(sc_module_name name):
                            sc_module(name)
     {
           SC_THREAD(main);
          sensitive_pos << Clk;
     }

     void main()
     {
         while(true)
         {
             in->read();
             for (int i=1; i <= 15; i++) wait();
             out->write();
         }
     }
};
```

Figure 4. Simple computation unit

In a synchronous module there may be different paths from inputs to outputs depending on the value of control signals. For precise behavioral modeling we considered this fact by supporting different wait cycle based on different control bits in tokens. These bits are set in TOKEN GENERATOR module which will be discussed later.

For modeling a complete system we have a top module in which we instantiate communication and computation units and clock signals and connect computation units through communication units. Figure 5 is a pseudo code fragment showing the top module of a GALS system.

In addition to the computation units that have real counterpart in RTL, we integrate the test bench of system with our model. Here, for doing this we implemented two modules in SystemC for generating and sinking tokens named Token Generator and Token Sink.

```
class top : public sc_module
{
   public:

      {wrapper declaration…}
      {module declaration…}
      {clockgenerator declaration…}

     top(sc_module_name name) : sc_module(name)
     {
       {computation and communication
        components and their connections
        …}
     }
};
```

Figure 5. Top level module

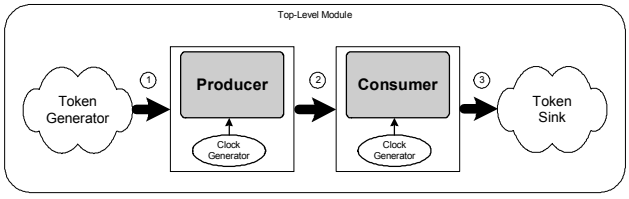

Figure 6. Producer Consumer model

Figure 6 demonstrates a top level module of producer-consumer model with token generator and token sink. Token Generator generates a token and writes it to its output port and waits for tokens to be read iteratively. Token Generator also selects a path that the generated token must pass through by setting some control bits in the token object. With this mechanism we can statistically send the tokens to different paths for overall evaluation of a design. Token Sink waits on its input port for a new token to be received then logs the info such as processing cycles, wait cycles regarding to the received token. The clock frequency and number of cycles that producer and consumer work on are reported in Table 1.

To illustrate the functionality, Figure 7 shows the timing behavior of the three channels labeled In Figure 6. At time 0 just channel 1 will have token number 1 and other channels are empty. At 2.1 ns channel 1 transfers its token to the producer and get token number 2 from token generator.

For transferring a token through a channel it takes 2.1ns. After the producer finished processing at 208.2ns, it writes token number 1 to its output port and at 230.28ns consumer writes token number 1 to sink module.

With a simulation result for five tokens, Figure 7 shows the timing diagram of channels in our producer-consumer model with their contents in time line.

Table 1. Frequency and number of cycles in producer-consumer model

	Clock Frequency	Number of Cycles
Producer	72 MHz	15
Consumer	82 MHz	2

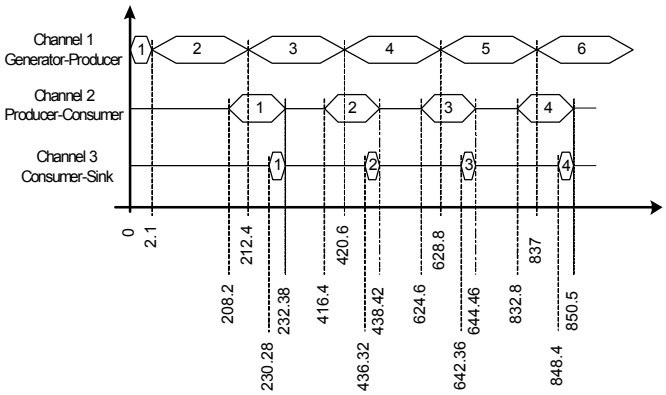

Figure 7. Timing diagram of channel in producer consumer

This process repeats for other tokens. In Figure 8 we have shown detail of read/write methods invocation in every channel of Figure 6.

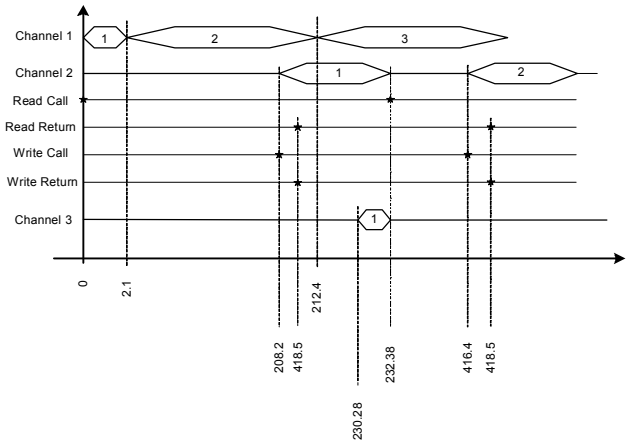

Figure 8. Read/Write methods invocation and their returns

4. EVALUATING GALS REED-SOLOMON

In this section we applied our proposed model to a Reed-Solomon decoder [7,11] as a benchmark for evaluating two

different partitioning schemes. Figure 9 demonstrates fully synchronous reed Solomon block diagram.

The path through Syndrome – RiBM – Chien Forney takes more time and cycle than FIFO path so we omitted the FIFO in our modeling for simplicity. We modeled two partitions in a totally abstract level and show that the results are consistent in order to confirm our model.

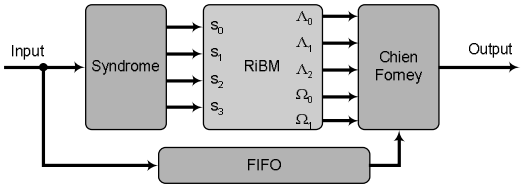

Figure 9. Reed-Solomon block diagram

The first partition includes three separated computation components as shown in Figure 10. The second partition includes two separated components shown in Figure 11.

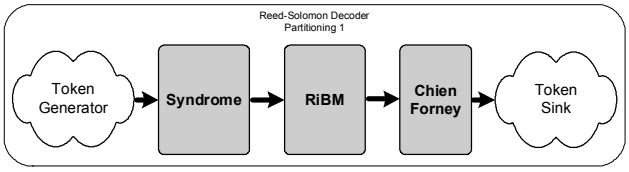

Figure 10. First partition of Reed-Solomon

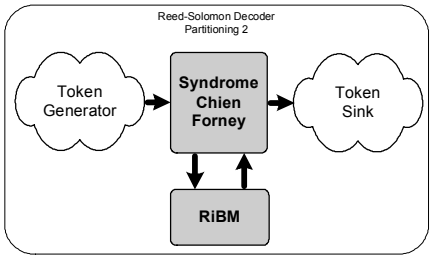

Figure 11. Second partition of Reed-Solomon

To obtain frequency of each of the modules we synthesized each module using synopsys with TSMC 0.13u library. The number of computation cycles is extracted from state machine of every module. Whenever we need to merge two modules we selected the lower frequency value. Table 2 shows the frequency and clock cycle of each module.

4.1 Performance Evaluation

After building the models for these two partitions, we performed a simulation for a period of 5 seconds. Considering that the suspension time of the input port of the syndrome is zero, overall performance can be measured as the rate of tokens on input port of the sink module.

Table 3 summarizes the performance metric of the two partitioning scheme for three different values of error rates as the number of tokens passed through the circuit from token generator to token sink. As it can be seen in Table 3 the performance of the second partitioning scheme, shown in Figure 11 is lower for all error rates.

Table 2. Frequency and number of cycles

Circuit		Clock freq.	Number of cycles
RS(15,11)		200Mhz	--
GALS RS(15,11)	Syndrome	250Mhz	15
	RiBM	250Mhz	Error-5 normal - 2
	Chien/Forney	218Mhz	11

Table 3.Throughput of two partitioning schemes with different error rates

	Number of tokens	
Error rate	**First partition**	**Second partition**
0%	24201	22687
50%	24135	22602
100%	24014	22513

4.2 Power Evaluation

For power consumption our current model still can not generally estimate power efficiency of every circuit but in the case of Reed Solomon decoder we see that our model can determine better partitioning regarding to the objective of power consumption in LS modules.

Although we could omit FIFO for measuring performance as it mentioned earlier, we need to add it for power estimation purpose because of its major effect on power.

Power consumption in GALS systems can be considered as sum of three factors:

$$P(GALS) = P(Sync) + P(OSC) + P(Async)$$

Here, *P(Sync)*, indicates the power consumed by the LS module while *P(OSC)* represents the power of the local clock generator. Both of these factors are proportional to the effective frequency of the local clock. It is remarkable that due to the pausible nature of the local clock, expected value of the effective frequency is less than its nominal value. In addition, an increase in read/write wait times, leads to lower effective frequency and then lower power consumption so the amount of the read/write wait times has a significant effect on the both *P(Sync)* and *P(OSC)*.

P(Async) stands for the power consumption in asynchronous wrappers which linearly increases by the activity of the

channels and number of wire (bits of data) between two modules that are transferring data.

In this work we compared *P(Sync)* of two partitioning schemes and left two other factors for future work.

It is know that that power is proportional to load capacitance and switching activity of circuits. For power comparison purpose in system level because there is no information about the implementation dependent factors, we used the size of active area as an indicator for consumed power. In addition, we know that in GALS systems every modules pauses its clock for waiting on input or output. Time utilization of every module is a good candidate for showing activity of circuit which can be simply estimated by our cycle accurate model. The size of active area is obtained using back annotating from synthesis tool. Table 4 shows utilization and area of every locally synchronous module.

Table 4. Utilization and Area of every LS block

	Utilization	Total Area
Syn-chien-fifo	97%	74900
Syndrome	97%	20597.5
RiBM	8%	118142
ChienForney	62%	40530
FIFO	68%	36662.5

Table 5. Calculated value Vs. Real power consumption

	Estimation	Real Power (mW)
GALS 1	146551.5	48.2084
GALS 2	167166.6	74.314

Table 5 shows our estimated value for every partition (Estimation column) and simulated power using synopsys power tool (Real Power column). Therefore, we can say power consumption in synchronous blocks of first GALS partition is less that second GALS partition. In contrast with simulated power consumption in two partitions, we have revealed our comparison was true.

As the result shows, our model predicts lower power for first partition when just power consumption in synchronous blocks is as an objective.

5. CONCLUSION AND FUTURE WORK

In this work, we have shown that abstract system level modeling can be used for performance evaluation of different partitioning schemes in GALS systems. As our investigations show, the effect of behavioral interaction of LSI modules in partitioning is considered for the first time in our work also all pervious efforts on modeling GALS systems was performed in lower levels.

Abstract representation is a good choice that designers can use without worrying about getting involved with complexity and functionality of the circuits so their works can be done in much less time.

On the other hand, our model has the ability to compare two partitions in term of their power consumption when just power in synchronous block is as an objective but we are trying to add *P(OSC)* and *P(Async)* to our comparison which remains for future works. In addition, automatic approaches for extracting required data for modeling, analytical modeling of modules, and making a library of abstract models for asynchronous communication schemes would be left for future works.

6. REFERENCES

[1] A. Hemani, T. Meincke, et al., Lowering power consumption in clock by using globally asynchronous, locally synchronous design style. In Proc. ACM/IEEE Design Automation Conference, pp. 873-878, June 1999.

[2] D.M.Chapiro.,Globally-Asynchronou Locally-Synchronous Systems, PhD thesis, Stanford University, 1984.

[3] J. Muttersbach, T. Villiger and W. Fichtner. Practical Design of Globally-Asynchronous Locally-Synchronous Systems. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, April 2000.

[4] J. Sparso, S. Furber, Principles of Asynchronous Circuit Design – A System Perspective, Kluwer Academic Publishers, 2002.

[5] D.S. Bormann and P.Y.K. Cheung.Asynchronous Wrappers for Heterogeneous Systems. In Proc. International Conference on Computer Design (ICCD), pp. 307–3014, October 1997

[6] S. Moore, G. Taylor, R. Mullins and P. Robinson. Point to point GALS interconnects. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 69-75, April 2002.

[7] S. Wicker, V. K. Bhargava, "Reed-Solomon Codes and Their Applications", IEEE Press, 1994.

[8] K. Niyogi, D. Marculescu, System level power and performance modeling of GALS point-to-point communication interfaces, In proceedings of international symposium on Low power electronics and design ISLPED '05 ,August 2005

[9] A. Upadhyay, S. R. Hasan, M. Nekili, Optimal partitioning of globally asynchronous locally synchronous processor arrays , Proceedings of the 14th ACM Great Lakes symposium on VLSI, April 2004

[10] Lukai Cai, Daniel Gajski, Transaction level modeling: an overview, Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software co-design and system synthesis , 2003

[11] K. Saleh, "Hardware Architectures for Reed-Solomon Decoders", *Technical Report*, Amirkabir University of Technology, January 2003