Available online at www.sciencedirect.com
ScienceDirect

Electronic Notes in Theoretical Computer Science 245 (2009) 103–113

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

GALS for Bursty Data Transfer based on Clock Coupling[1]

Miloš Krstić[2], Xin Fan[3], Eckhard Grass[4]

IHP Microelectronics
Im Technologiepark 25
15236 Frankfurt Oder, Germany

Frank K. Gürkaynak[5]

Microelectronics Design Center
ETH Zurich
CH-8092 Zurich, Switzerland

Abstract

In this paper we introduce a novel burst-mode GALS technique. The goal of this technique is improving the performance of the GALS approach for systems with predominantly bursty data transfer. This new technique has been used to implement a GALS-based version of a hardware accelerator of a 60 GHz OFDM baseband processor. The simulation results show a significant performance improvement in comparison with a classical implementation of GALS using pausible clocking.

Keywords: GALS, bursty data transfer, pausible clocking.

1 Introduction

Developing complex digital systems in state-of-the-art nano-scale technologies becomes increasingly difficult. In addition to standard performance parameters such as power dissipation, clock frequency and silicon area, new design flows have to cope with process variation and reliability without significantly increasing the time to market. The leading problem for designers of large System-on-Chips is undoubtedly the on-chip data exchange between blocks. This has become increasingly complex

[1] This work has been supported by the European Project GALAXY under grant reference number FP7-ICT-214364 (www.galaxy-project.org)

[2] Email: krstic@ihp-microelectronics.com

[3] Email: fan@ihp-microelectronics.com

[4] Email: grass@ihp-microelectronics.com

[5] Email: kgf@ee.ethz.ch

doi:10.1016/j.entcs.2009.07.031

due to increasing system frequency and complexity. As a consequence, the complete data interconnect structure is being redirected into communication-centric Networks on Chips (NoCs), that are frequently implemented in the asynchronous fashion [1] [2].

For wireless communication systems, the design challenges are often even more difficult. The design complexity increases rapidly. For example, the complexity of a currently utilized 5 GHz WLAN OFDM baseband processor supporting data rates up to 54 Mbps is around 500k gates [3], the state-of-the-art 60 GHz OFDM baseband processor supporting data rates up to 1 Gbps has a complexity of more than 23M gates [4], and research efforts are now targeting data rates of 10 Gbps which will increase the complexity by one order of magnitude. Furthermore, the communication systems impose additional issues such as low-power and EMI reduction, in order to increase mobility and achieve System on Chip (SoC) integration.

To develop such complex systems using a classical synchronous paradigm, where a synchronous clock supplies an entire system, is an onerous task. One reasonable alternative to the completely synchronous approach is a Globally Asynchronous Locally Synchronous (GALS) methodology. This technique has been developed already for years. In principle, GALS technology can be utilized in three ways: applying synchronizers between the synchronous blocks, using asynchronous FIFOs as interfaces, or utilizing pausible clocking schemes for data transfer [5]. GALS based on pausible clocking is very interesting as a system integration technique. Some mature solutions have been developed [6] and practically evaluated [8]. Recently, this solution was also practically applied as a physical layer of one complex NoC system [9]. Although this method is very attractive for many applications (for example cryptography), intensive data transfer could lead to performance degradation. The performance degradation due to intensive clock stretching reported in [6] was 23%. With more optimal designs this performance loss can be reduced but not completely avoided. In particular, this performance loss will be clearly visible for bursty data transfer, which is usually needed for datapath baseband processing in wireless communication systems. We have already tried to cope with this issue, and suggested a so called *request-driven GALS technique* [10]. This solution supports bursty data transfer with low performance loss. However, the complete GALS interface design was relatively complex and frequency limited. Therefore, in this paper, we propose a more refined GALS interface solution optimized for bursty data transfer.

In the following section we describe the concept of the novel burst-mode GALS wrapper. In section 3, we give some details of the wrapper implementation and compare it to the other GALS solutions. Section 4 is dedicated to the practical application of this concept to a hardware accelerator for a 60 GHZ OFDM baseband processor. Finally, we will draw some conclusions.

2 Burst Mode GALS Wrapper

In the previous section we have noted some features of the usually applied GALS interfaces based on pausible clocking. One important issue for many of those interfaces is the inability to support one data transfer per one clock cycle. Most GALS implementations support one data transfer per two clock cycles (or less). On the other hand, there is a limited set of interfaces able to cope with data bursts, i.e. enabling data transfer on each clock cycle of the local clock. However, there is a general problem of such pausible clocking interfaces with bursty data transfer. Traditionally, pausible clock based GALS approaches, synchronize between communicating modules during each data transfer, adding unnecessary overhead. In addition to that, it looks unnecessary to synchronize the clocks for each clock cycle when operating in burst mode.

An efficient solution would be to utilize a single clock during a burst (normally the one with lower clock frequency) for both clock domains or to lock both clock sources to the same clock frequency. In this case, during the burst transfer both locally synchronous (LS) domains would use the clock with same frequency. When the burst is finally transferred, those LS domains can be triggered again with their own independent clock sources. With such solution, we can achieve an independent operation of the locally synchronous blocks when data is not being transfered and a locked operation in data transfer mode. The clock synchronization is then needed only once per data burst. Another positive aspect of this technique is the ability to use a separate operation speed during data transfer and in operation. A similar idea is used in [10] with so called "request-driven approach". However, the solution presented there is very complex.

One possible solution based on the described technique is illustrated on Fig. 1. In this figure a typical configuration of two independent LS blocks is given. Those independent blocks are triggered with different clock generation units. Data transfer is controlled over master and slave port controllers. Burst data transfer is initiated when a LS block activates the synchronous burst enable (*Ben*) signal to the master control port. When a burst transfer starts, the slave clock control is handed over from its own clock generator to the master clock. This is indicated with activation of (*lock*) signal. When the burst is being transferred, this is indicated to the slave with the activation of the burst valid signal (*Bv*). The main point is that when the burst transfer is initiated, the clocks are locked, i.e. both master and slave are trigerred from the same clock source. Therefore, we are indicating with the lock signal the necessity that one of the clock sources has to be locked to the other. In the configuration provided in Fig. 1 the slave clock has to be locked to the master clock. The *lock* signal is generated from the arbitrated *req* signal and must be sampled with one additional latch to avoid the clock locking when the master clock is on high. Burst request (*req*) is arbitrated with the locally generated clock. When the local clock is not active, request will propagate and generate a burst acknowledge (*ack*). Furthermore, the *lock* signal is used to control the clock multiplexer. When the *lock* signal is low, the slave clock *sclk* is driven from its local clock generator (*lclk*). Otherwise, the slave clock *sclk* is driven from the master clock (*mclk*). In order to have a safe clock transfer, it is allowed to change the value of the *lock* signal only when both input clocks (*lclk* and *mclk*) are low. To assure a safe behavior of clock control transition, we have to apply MUTEX component as shown in Fig. 1.

In particular, we have to disable any new slave clock (rising edge) originated from its local clock when the *req* signal is activated.

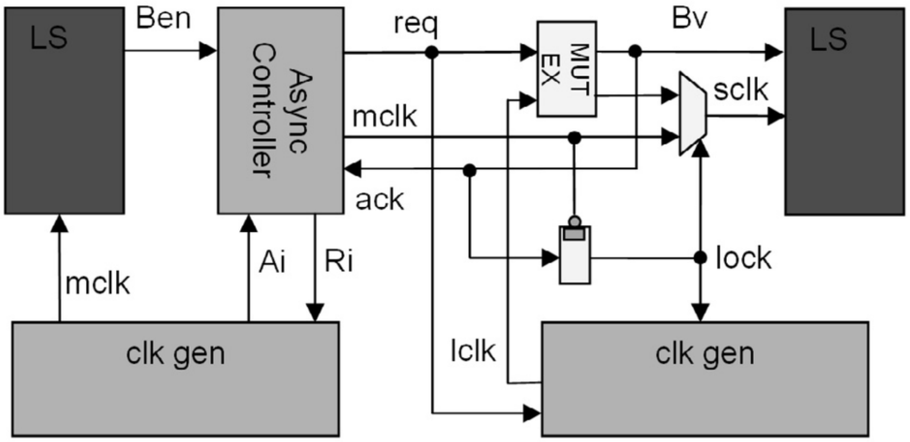

Fig. 1. Coupled GALS controllers for bursty data transfer

The STG specification of the master controller is given on Fig. 2. In principle, this is a very simple asynchronous controller which pauses the clock when burst activation arrives ($Ben\uparrow$) and activates a request for burst ($req\uparrow$). In order to assure that the master doesn't produce further clock pulses, its local clock must be paused until the burst is granted ($Ri\uparrow$). When the burst is acknowledged ($ack\uparrow$), the master clock stretch signal is released ($Ri\downarrow$), the *lock* signal goes high, and the burst transfer starts. In this case, the master clock (*mclk*) propagates to the slave module. When all data are transferred, *Ben* is deactivated ($Ben\downarrow$) and the port controller then deactivates the handshake signal between the GALS blocks ($req\downarrow$). When the slave port deactivates acknowledge ($ack\downarrow$), the master port is ready to start with another burst.

Here is the result of the logic synthesis, using the Petrify tool [12], of the controller shown in Fig. 2:

$Ri = Ben \cdot ack'$

$req = Ben \cdot ack + Ai$

From those equations it is clear that the implementation of such controller would be very simple and end-up with just a couple of gates.

In order to apply those interfaces, some changes of the clock generator must be performed in comparison to the one used in the standard GALS solutions [11]. The block diagram of the modified clock generator is given in Fig. 3. In the normal (locally driven) mode this circuit behaves as a usual ring oscillator clock generator. In the locked mode (burst arrives, *req* or *lock* on high) this circuit behaves as a stoppable clock. In this way we are disabling the positive clock edge on the *lclk* output whenever *req* arrives and until *lock* lowers. If the same locally synchronous block is used as a master block for some other locally synchronous module, the arbitration unit must be added to the proposed slave clock generator.

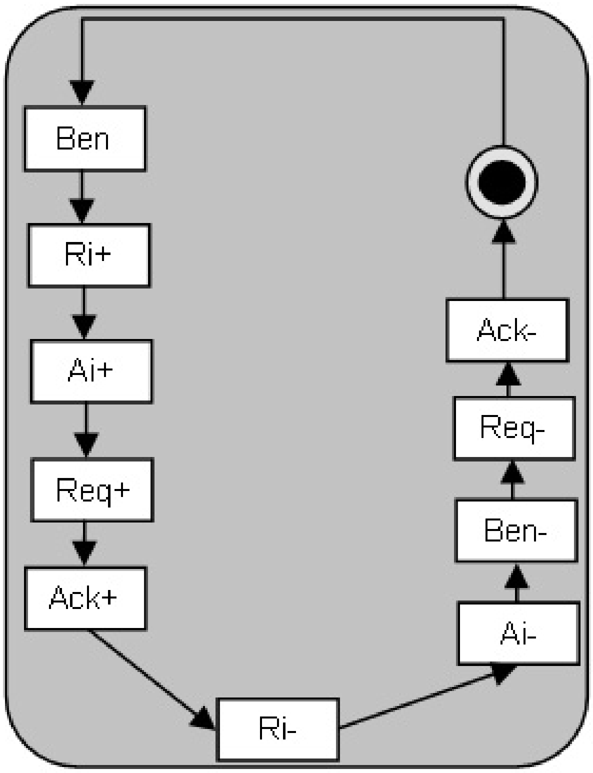

Fig. 2. Specifications of the master GALS controller

The arbitration unit, consisting of the mutual exclusion element(s), is needed to provide a pausible feature to the clock source. This needed for the master bahaviour.

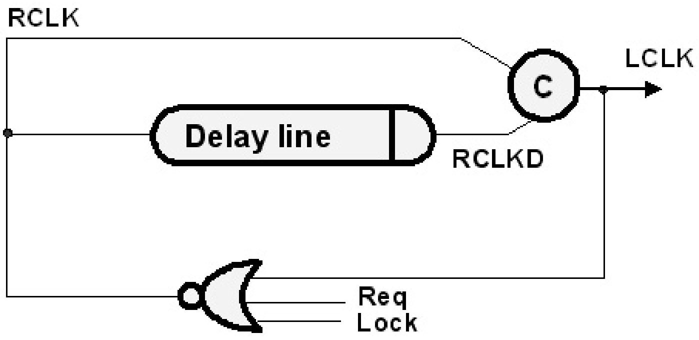

Fig. 3. Modifications of the clock generator

In principle, it is not necessary that the slave clock is always locked to the master clock. The other configuration is also possible (when a master clock is locked to the slave clock). The rule is usually the following: the clocks have to be locked to the one having the lower clock frequency. The reason for that is simple. We are always optimizing some synchronous block for some frequency but this circuit can always operate at a lower frequency. The reciprocal configuration where the master clock is locked to the slave clock is also possible.

The configuration where the master clock has to be locked to the slave is similar to the one shown at Fig. 1. The only point is that the lock signal is generated from the master port and that the master clock generator has to be modified instead of the slave generator.

The proposed GALS technique can also be applied in combination with the classical pausible GALS technique. In principle, it is sufficient that the timing critical interconnects for long bursty data transfers are covered by this type of interface.

3 Wrapper Implementation

In order to evaluate the proposed burst mode GALS scheme we have performed the modeling of the simple GALS point-to-point system that is used as a verification vehicle of the concept. In Fig. 4, a simulation run of two burst transfers over this simple GALS link is given. In the first burst, 10 data symbols are transferred, and in the second, additional 15 data. The synchronization between the two modules is performed just once at the beginning of the burst. Some time is used to perform the locking of the system. This is actually the only time interval where we are incurring additional time loss and where performance is reduced. The cost of this initial synchronization is normally less than one clock cycle. The complete data transfer is performed afterwards without any loss and additional synchronization. During data transfer, the locally synchronous modules are locked and are using the same clock source.

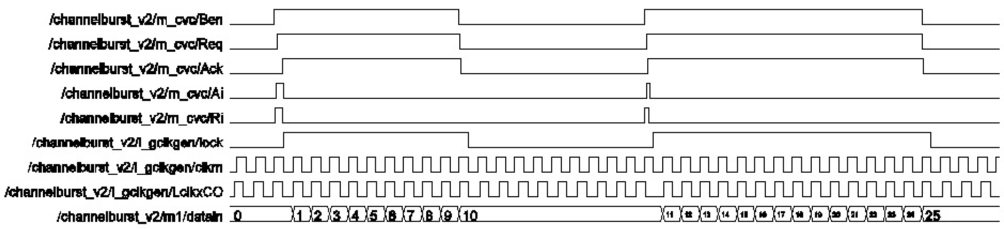

Fig. 4. Transfer of one burst over GALS link

We have synthesized the complete burst GALS interfaces using a 0.13 um CMOS process provided from IHP. For asynchronous controllers we were using the Petrify tool [12]. As already explained, the complexity of the controllers is very low. The master controller is of equivalent complexity of 4 inverter gates. More complexity is added by the clock generators. A typical clock generator has a complexity of about 600 gate equivalents.

The throughput simulation shows that the maximum frequency of the controller is limited to 609 MHz for typical PVT conditions (this is approx. 15 FO4 delays). However, the maximum throughput can be even higher since during the burst transfer, there is no synchronization and the clock is directly transferred from one block to the other. Therefore, during the transfer the clock frequency can be higher than this limit. On the other hand, it is very unlikely for this process that we will need the clock frequency higher than this. Of course, in the first synchronization cycle the clock will be stopped and the timing interval needed to perform data transfer will be increased for this time. The maximum frequencies of the clock generators are 763 MHz (master) and 581 MHz (slave).

Regardless of the design, GALS implementations will always introduce some overhead due to the interface circuitry. However, typical implementations of GALS favor coarse functional blocks which frequently exceed 100k gate complexity, where the area overhead of a few hundred gates will not be a major factor. Also, the frequency limit of the proposed circuitry is in accordance with the utilized process and it should not normally impose any limitation for locally synchronous blocks. It is plausible to expect that the performance of the proposed interfaces will scale very well with smaller device sizes. Finally, the performance loss introduced over the synchronization period that precedes each data burst is usually limited to one clock cycle. This loss can be tolerated for longer data bursts.

The GALS influence on power consumption was carefully analyzed in [7]. With our approach we can expect the similar number since conceptually from the power consumption point of view there are not much differences in the analyzed work and our implemented method. The main potential advantage which can be successfully utilized is different clock frequency in data acquisition mode and in data processing mode. This can be the good point for introduction of dynamic frequency and/or voltage scaling technique.

4 Burst Mode Wrappers in Practice

A more complex but also more realistic case is to have more than one master and slave in the system. Our burst mode GALS scales well also for more involved systems. However, the clock generator for the slave block has to be also pausible if this block is also master block for some other GALS module, as defined in section 2.

In order to test burst mode GALS wrappers we have generated a realistic hardware system consisting of 5 GALS modules. This system is part of a 60 GHz OFDM baseband processor and can be used as a hardware accelerator [4]. This processor has an innovative streaming architecture that can achieve a throughput of 1 Gbps operating with only 100 MHz clock rate. The block diagram of the implemented hardware accelerator is shown on Fig. 5. The first block in this scheme contains the soft demapper/power weighting block. In this case the system supports two different streams. Therefore, we have placed a set of two deinterleavers/viterbi decoders. The demapper/weighting block in this system distributes data for different streams into one of two available datapaths. With such a parallel approach, it is possible to double the throughput.

We have implemented the system using two different GALS approaches: classical pausible clocking (Fig. 6) as described in [6], and burst mode GALS (Fig. 7). In Figure 7, we have shown a relatively complex system of 5 GALS blocks and 8

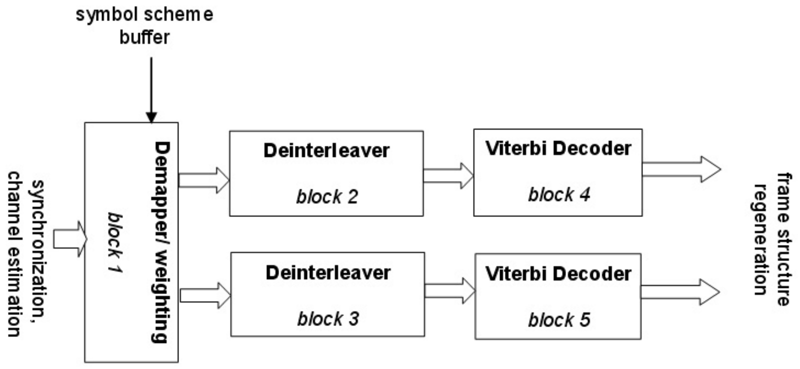

Fig. 5. Hardware accelerator for 60 GHz OFDM baseband processor

different GALS interface ports. Block 1 acts as a master for Block 2 and Block 3. However, since the activation signal (Ben) is independent for those two blocks we need in fact two master ports (MP12 and MP23). Blocks 2 and 3 act as the master for blocks 4 and 5, respectively. Therefore, their local oscillator has to be pausible to be able to stretch its clock during burst initiation between B2-4 and B3-5. In addition to that, whenever local clock 2 or 3 are stretched ($r2$ or $r3$) we also have to stretch master clock 1 because the slave clocks could be driven from the master clock in this moment. Stretch acknowledge signals for master-slave block ($a2$, $a3$) are then constituted by a join (using C-element) of the acknowledge signals coming from LO2(or LO3) and master LO1. This is not shown in this figure in order to reduce the complexity of the diagram. This system is successfully implemented and simulated. This implementation confirms the ability of our burst GALS scheme to deal also with more complex designs.

The designed hardware accelerator is very well suited for application of burst mode GALS wrappers since the complete data transfer activity is performed in bursts of data as shown in Fig. 8. In this figure the length of each burst for bursts 1-2 and 1-3 is 49 data transfers. Bursts 2-4 and 3-5 have a variable number of data transfers starting from 97 up to 769 consecutive data exchanges. For comparison, the same system is implemented using a classical GALS concept [6] (Fig. 6). In this case, we have used the demand type input and output ports in order to fulfill the requested timing of the system. For both GALS implementations we have used behavioral description of synchronous blocks and synthesized netlists for the asynchronous wrappers. The performance analysis has shown that for a plesiochronous system (frequencies of the local clocks for the synchronous blocks f_{lcki} are similar and almost identical) the performance gain is very low. For example, if the difference between the highest and lowest clock frequency is 3.6%, the performance gain of burst mode circuits is only 1.6%. However, if this difference is higher the performance gain also rises. If the frequency difference is 9.6%, a burst mode GALS system achieves 6.34% higher performance.

The proposed burst GALS interface represents one step ahead in comparison to existing solutions. For bursty data transfer we just perform the synchronization

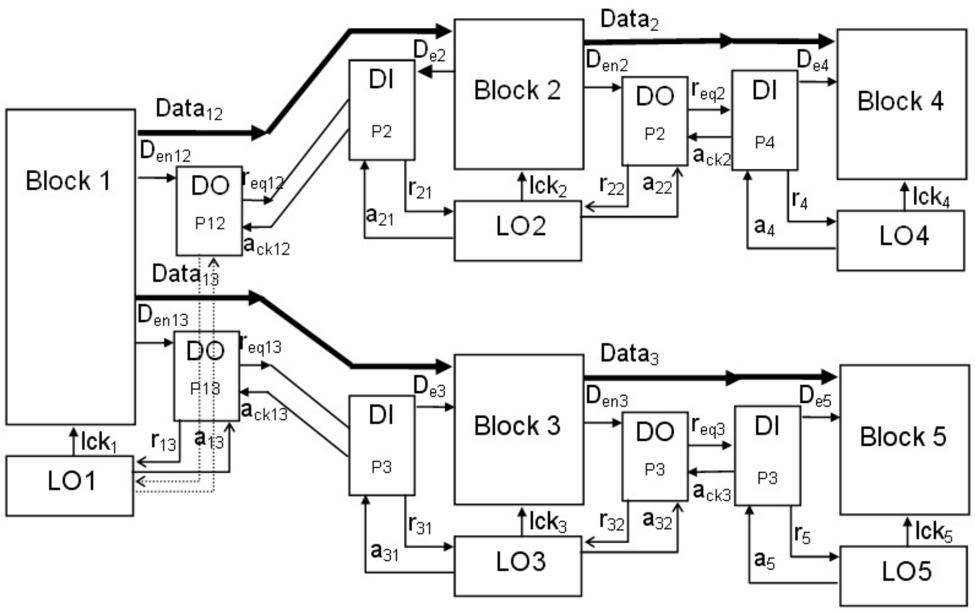

Fig. 6. GALS partitioning for classical pausible clocking (DI - demand input port, DO - demand output port, LO - local oscillator)

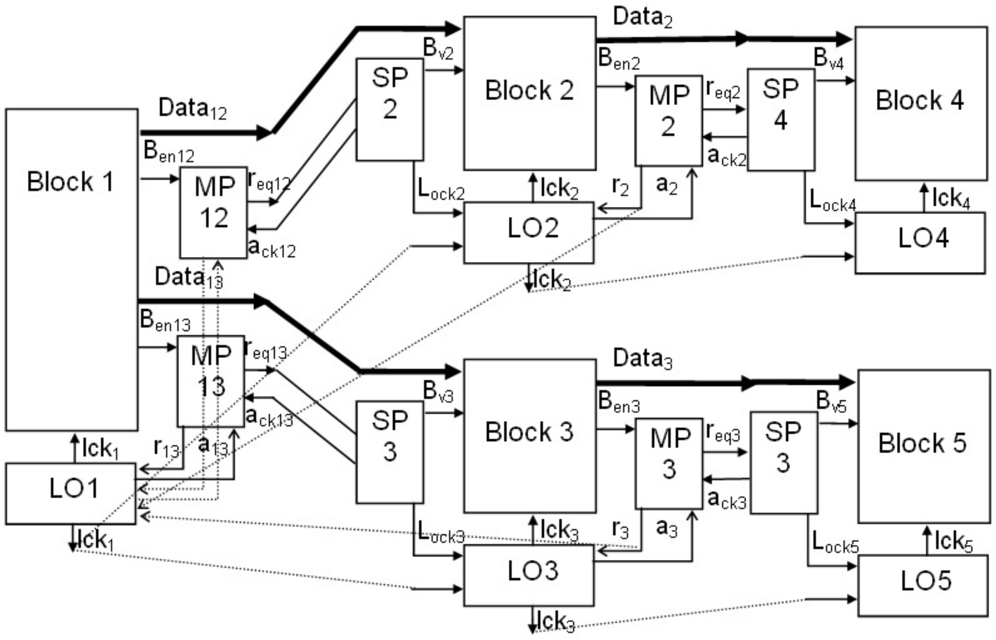

Fig. 7. GALS partitioning for burst mode GALS (MP - master port, SP - slave port, LO - local oscillator)

once at the beginning of the burst and from this moment no synchronization is needed. This is a significant advantage in comparison with classical pausible clocking interfaces. The applied solution is by nature similar to the one proposed in [10]. However, the implementation is much simpler and the throughput is much higher.

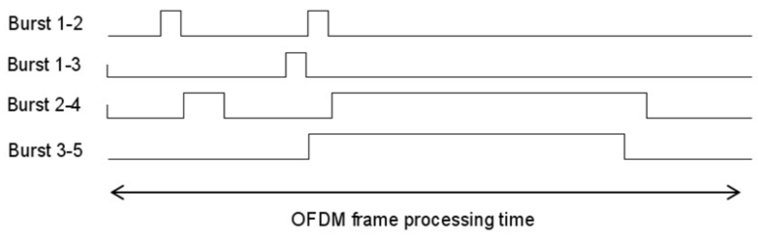

Fig. 8. Burst activity of designed system

The critical path of the interfaces given in [10] were around 45 FO4 delays and the critical path of the burst mode GALS interface is around 15 FO4 delays.

For systems that do not predominantly use bursty data transfer, it is still possible to use the proposed scheme. However, due to its initial synchronization delay, the results may be suboptimal for frequent single data transfers. For such system it is possible to use different ports to decouple single data transfers and bursty data transfers. For single data transfer the classical pausible clocking interface may be used and for bursts, we can apply the proposed scheme. Alternatively, it is possible to modify the burst interfaces to support the special mode of the single data transfers. For solution we would not perform the frequency locking and use the usual pausible clocking scenario. However, the introduction of such mode needs certain modification of the port specification (differentiating a single and the burst transfer request, disabling the lock for single transfer). Such modification will lead to higher complexity of the burst port controllers and possibly also to the lower maximum throughput of them. Therefore, it is best to use the burst controllers for systems with predominantly bursty data transfer and if necessary add a special classical GALS controller for single data transfers.

5 Conclusions

In this work, we have introduced a novel burst mode GALS technique. This technique is optimized for GALS systems with predominantly bursty transfer between the asynchronous blocks. We have proposed one simple solution that can be utilized for high performance systems. We have evaluated the proposed solution by applying it to the GALS hardware accelerator of a 60 GHz OFDM baseband processor. This evaluation has shown that a burst mode GALS system exhibits better performance for bursty data transfer than the GALS method proposed in [6]. The proposed solution is particularly useful when locally synchronous blocks have different local clock frequencies. In this way, we are able to use for the same local block one clock frequency for data acquisition and the other for data processing, optimizing the power dissipation and performance.

This paper describes one particular configuration of GALS interfaces with a master output port and a slave input port. However, other configurations are also possible. This will be a direction for our future research. The ability of this technique to solve system integration issues has to be verified in a real implementation. However, we are very optimistic that this burst mode GALS method can be successfully applied for digital systems with bursty data transfer between GALS blocks.

References

[1] Lines, A., *Asynchronous interconnect for synchronous SoC design*, IEEE Micro, 2004, **24**, (**1**), pp. 32-41.

[2] L. A. Plana W. J. Bainbridge, S. B. Furber. *The design and test of a smartcard chip using a CHAIN self-timed network-on-chip.* In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Vol. **3**, page 274, Feb 2004.

[3] M. Krstić, K. Maharatna, A. Troya, E. Grass, U. Jagdhold, *Baseband Processor for IEEE 802.11a standard with embedded BIST*, Facta Universitatis, Series: Electronics and Energetics, Nis, Serbia, Vol. **17**, Aug 2004, pp. 231-239.

[4] M. Krstić, M. Piz, M. Ehrig, E. Grass, *OFDM Datapath Baseband Processor for 1 Gbps Datarate*, Proceedings of IFIP/IEEE VLSI-SoC 2008 - International Conference on Very Large Scale Integration, Rhodes, Greece, Oct 13-15 2008, pp. 156-159.

[5] M. Krstić, E. Grass, F. Gürkaynak, P. Vivet, *Globally Asynchronous, Locally Synchronous Circuits: Overview and Outlook*, IEEE Design & Test of Computers, Vol. **24**, No. **5**, September-October 2007, pp. 430-441.

[6] J. Muttersbach, "Globally-Asynchronous Locally-Synchronous Architectures for VLSI Systems", Doctor of Technical Sciences Dissertation, ETH, Zurich, Switzerland, 2001.

[7] E. Talpes, D. Marculescu, *Toward a Multiple Clock/Voltage Island Design Style for Power-Aware Processors*, IEEE Transactions on Very Large Scale Integrations (VLSI) Systems, Vol. **13**, No. **5**, pp. 591-603, May 2005.

[8] F. K. Gürkaynak, S. Oetiker, N. Felber, H. Kaeslin and W. Fichtner, *Improving DPA Security by Using Globally-Asynchronous Locally-Synchronous Systems*, ESSCIRC 2005, 31st European Solid-State Circuits Conference, pp. 407-411, 2005, Grenoble, France

[9] E. Beigne, F. Clermidy, S. Miermont, P. Vivet, *Dynamic Voltage and Frequency Scaling Architecture for Units Integration within a GALS NoC*, Proceedings of 2nd ACM/IEEE International Symposium on Networks-on-Chip (NoCS 2008), pp. 129-138, April 2008.

[10] M. Krstić, E. Grass, C. Stahl, M. Piz, *System Integration by Request-driven GALS Design*, IEE Proc. Computers & Digital Techniques, Vol. **153**, Issue **5**, September 2006, pp 362-372.

[11] S. Moore, G. Taylor, R. Mullins, P. Cunningham, Peter Robinson, *Self Calibrating Clocks for Globally Asynchronous Locally Synchronous System*, Proceedings of International Conference on Computer Design (ICCD), September 2000.

[12] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, *Petrify: a Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers*, IEICE Transactions on Information and Systems E80-D (1997), no. **3**, 315325.