1-1-2004

An Asynchronous GALS Interface with Applications

Jennifer A. Smith
Boise State University

DOI: 10.1109/WMED.2004.1297347

An Asynchronous GALS Interface with Applications

Scott F. Smith
Department of Electrical and Computer Engineering
Boise State University
Boise, Idaho, U.S.A.
SFSmith@BoiseState.edu

***Abstract*—A low-latency asynchronous interface for use in globally-asynchronous locally-synchronous (GALS) integrated circuits is presented. The interface is compact and does not alter the local clocks of the interfaced local clock domains in any way (unlike many existing GALS interfaces). Two applications of the interface to GALS systems are shown. The first is a single-chip shared-memory multiprocessor for generic supercomputing use. The second is an application-specific coprocessor for hardware acceleration of the Smith-Waterman algorithm. This is a bioinformatics algorithm used for sequence alignment (similarity searching) between DNA or amino acid (protein) sequences and sequence databases such as the recently completed human genome database.**

Keywords-interface; multiprocessor; globally-asynchronous locally-synchronous; bioinformatics; coprocessor

I. INTRODUCTION

The idea of globally-asynchronous locally-synchronous (GALS) systems was first proposed by Chapiro [1]. One very attractive feature of this design paradigm for systems on a chip (SoC) is that it greatly reduces the amount of design effort that goes into designing a clock distribution network for large high-speed integrated circuits. The idea is that clock skew within a local clock domain will be highly controlled, but minimal effort will be employed to reduce clock skew between state-holding elements in different local clock domains. Synchronous intellectual property (IP) blocks may be "shrunk" to a smaller CMOS feature size along with their associated clock distribution network and then this shrunk IP reused in many SoC designs. By using the GALS design methodology, the design effort involved in connecting the local clock distribution networks together in such a way as to obtain timing closure is greatly reduced.

A major drawback to the GALS methodology is that it introduces latency into signal propagation across clock domain boundaries. The purpose of this paper is to introduce a clock-domain interface for GALS systems with low latency and which does not require the interface to alter the local clock signals in any way. The interface must address the issue of metastability and one way to do this is to have the interface take control of the two local clock signals at the clock-domain boundary. Having the interface stop or stretch local clock signals to avoid metastability was deemed unacceptable since this can potentially cause some logic gate styles to fail (namely some types of dynamic logic gates). The GALS interface is then applied to the design of a single-chip shared-memory multiprocessor. It is shown that the extra latency of the GALS multiprocessor causes very little performance penalty for a bioinformatic and a cryptography application running on the processor as compared to the equivalent globally-clocked multiprocessor. The GALS interface is then also applied to the design of a special-purpose coprocessor for acceleration of the Smith-Waterman sequence alignment algorithm used in bioinformatics.

The GALS interface design is discussed in Section II. The first application of the interface is a single-chip shared-memory multiprocessor which is presented in Section III. Section IV introduces a coprocessor to accelerate the Smith-Waterman dynamic-programming-based sequence alignment algorithm. The results are summarized in Section V.

II. GALS INTERFACE

The GALS interface is based on asynchronous FIFOs from the GasP family introduced by Sutherland and Fairbanks [2]. Two minor modifications were made to the FIFOs themselves. The first was to replace NMOS pass transistors with full transmission gates in the datapath to allow low voltage operation (whereas the original GasP paper focused on a 3.3V technology). The second was to replace a self-reset circuit driving the datapath with a string of asymmetric inverters such that a very wide datapath could be driven without distortion of control pulse shape. The major innovation is the circuitry surrounding the GasP asynchronous FIFO which allows it to be used as a clock-domain interface with selectable synchronizer mean time between failures (MTBFs). It is theoretically impossible to make the probability of synchronizer failure zero in this type of circuit [3], but MTBFs of millions of years are easily achievable with interface latencies on the order of nanoseconds.

This is not the first GALS interface to have been designed. Examples of others are found in [4] and [5]. The most important difference between the interface presented here and others in the literature is that this one does not require the interface to control the local clock signals, whereas the others require a "stoppable clock". While having a stoppable clock does make it possible to have zero probability of synchronizer failure, it is not compatible with all logic design styles. The probability of synchronizer failure for the interface described here can be made as small as desired (but never zero) by increasing the latency of the interface. Normally, the probability of synchronizer failure can be made insignificant by introducing additional latency which is a fraction of a clock period in duration.

0-7803-8369-9/04/$20.00 ©2004 IEEE.

A high-level depiction of the interface is shown in Fig. 1. Details of the inner workings of the two blocks in this figure can be found in [6]. The two local clocks signals are given as *sclk* (send clock) and *rclk* (receive clock). The sending clock domain first checks that there is at least one space available in the synchronous buffer by observing the *bfull* signal. If there is space in the buffer then the sending circuit can assert the *send* signal and is guaranteed that either the synchronous buffer or the asynchronous interface will absorb the sent data. If the relative phases of the local clock domains happen to align well and there is no pending data in the synchronous buffer, the buffer will be completely bypassed and the data immediately entered into the asynchronous interface. The datapath is labeled with *sdata*, D_{in}, and D_{out}. This datapath may be an arbitrary number of bits wide. The receiving clock domain circuit signals that it is ready for new data by asserting *rreq** and is in turn signaled if new data is available with *rack**. The synchronous buffer signals the asynchronous interface that it has new data available (either in the buffer or on the buffer input) by asserting *sreq*. When new data is actually absorbed by the asynchronous interface this is signaled with *sack*.

The interface was simulated in SPICE using parameters for a 180 nm CMOS process from the Taiwan Semiconductor Manufacturing Company (TSMC) which is available through the MOS Implementation Service (MOSIS). In this process a MTBF in the millions of years can be achieved with a base latency of less than 2 ns plus a clock signal phase differential delay. With circuits that can recover from synchronizer error, a base latency about 1 ns can be achieved with synchronizer error recovery required every few seconds.

Layout area for the asynchronous interface and synchronous buffer are dominated by datapath area for wide datapaths. Table I shows the layout areas in the same 180 nm CMOS process discussed above. The areas are for the applications in Sections III and IV below. The multiprocessor interface datapath is about 100 bits wide. The bioinformatics coprocessor uses two interfaces, the one for constants is 8 bits wide and the one for calculations is 85 bits wide. The areas given are for a unidirectional interface, so the area in Table I must be doubled for the multiprocessor case that requires bidirectional communication across clock domain boundaries. For the 8-bit coprocessor constants interface, the layout area is about evenly divided between datapath and control. For the 100-bit multiprocessor and 85-bit coprocessor interfaces, the layout area is dominated by the datapath. For these area calculations, the depth of the synchronous buffer was taken to be one. Allowing for deeper synchronous buffers will increase interface layout area significantly.

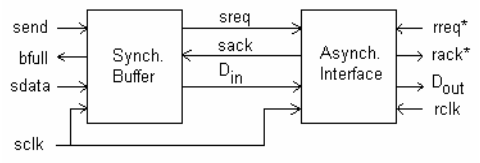

Figure 1. The asynchronous interface with buffer

TABLE I. LAYOUT AREAS FOR INTERFACE AND BUFFER

Application	Layout area (mm^2)	
	Synchronous buffer	*Asynchronous Interface*
Multiprocessor	0.07	0.14
Coproc. - Const.	0.01	0.02
Coproc. - Calc	0.06	0.12

a. Based on TSMC 180 nm CMOS process.

III. SINGLE-CHIP MULTIPROCESSOR

The single-chip shared-memory multiprocessor is based on a binary tree bus structure with a single ARM 922T CPU core (32-bit RISC processor with separate instruction and data caches) at each local-clock node [7]. The local bus segments within each node are standard AMBA buses. Layout of the interfaces and layout area data from ARM Ltd. shows that such a multiprocessor with over 100 processors can be built using 180 nm technology. Fig. 2 shows a single clock domain of the multiprocessor. Each of the "Asynchronous Interface Pair" blocks in Fig. 2 contains two copies of the asynchronous interface with buffer shown in Fig. 1.

The Smith-Waterman sequence alignment algorithm and the Advanced Encryption Standard (AES) were hand coded in 32-bit ARM assembly language and cache block replacement frequencies determined from this code. A VHDL model of the multiprocessor system and another of its globally-clocked analog were constructed that allow for different numbers of processors. Simulation using the VHDL models shows that the performance penalty for both the bioinformatics and cryptography applications is small when compared to the globally-clocked system [8][9].

The efficiency of the multiprocessor has been quantified in terms of the fraction of time that processors spend computing rather than waiting for a data cache block replacement. Algorithms which require a high number of computations per data element have higher efficiencies. For both the Smith-Waterman algorithm and AES these efficiencies are usually 0.95 or above.

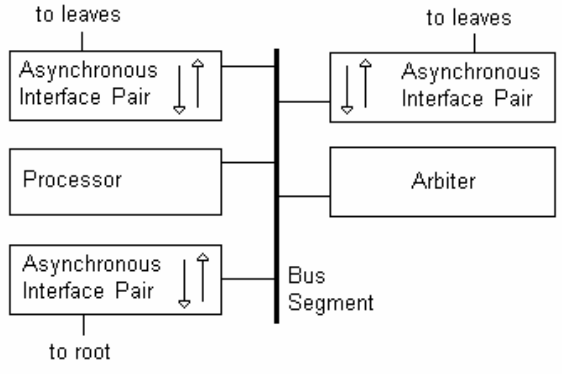

Figure 2. A single clock domain of the multiprocessor.

Data cache block replacement latency depends on where the processor is located within the bus tree, the relative phases of all the local clock signals of clock domains between the processor and the shared memory, and contention with bus traffic for other processors. The memory read request for a cache block replacement contends with other memory read requests and with memory writes. The memory read data contends with memory read data for other processors. The local bus segment in Fig. 2 is really two independent bus segments, one for traffic toward shared memory in the tree root and one for traffic away from shared memory. The mean data-cache block-replacement latency over all processors is less than 100 clock periods when the number of processors is less than 128.

IV. BIOINFORMATICS COPROCESSOR

Sequence alignment is the basis for many tasks in the field of bioinformatics. Good sequence alignments are very computationally demanding and the amount of data available for processing is increasing at an exponential rate. Often poorer quality alignments are used because the best quality alignments simply take too long to compute. The best quality alignments come from linear programming methods such as those of Smith and Waterman [10]. Sequence alignment is the process of introducing gaps into the two sequences such that the character pairs at each sequence position give a maximum score.

In order to allow the Smith-Waterman algorithm to be used more extensively, application-specific integrated circuits (ASICs) may be designed with hardware coprocessors specific to the Smith-Waterman algorithm. It is desirable that these ASICs be easily scalable since growing molecular biology databases and decreasing CMOS feature sizes will continuously spur demand to place increasingly powerful coprocessors on a single integrated circuit.

The Smith-Waterman algorithm is given in equations 1-4. The algorithm proceeds from the upper-left to the lower-right of a two-dimensional matrix. The row and column indices are given by i and j respectively and three scores are calculated at every position in the matrix. These values are I (the score if we insert), D (the score if we delete), and M (the score if we match or mutate). The initialization of the algorithm starts by setting all of the boundary variables on the left and top to zero as given by

$$I_{i,j} = D_{i,j} = M_{i,j} = 0, \forall\ i \text{ and } j \text{ s.t. } i = 0 \text{ or } j = 0. \quad (1)$$

The algorithm is symmetric, so insertions in one string are equivalent to deletions in the other string. There is a penalty for starting a new insertion or deletion and a (usually smaller) penalty for continuing an insertion or deletion. The start penalty is given as g and the continuation penalty as c. The insertion score at a position is the previous insertion score minus c if the insertion is to be continued. The insertion score is the previous match/mutation score minus g if a new insertion is to be made. The optimum choice at this position for an insertion score is the maximum of the two as given by

$$I_{i,j} = \max\{I_{i-1,j} - c,\ M_{i-1,j} - g\}. \quad (2)$$

A similar method is used to find the optimum deletion score at a given matrix position. The maximum score over the possibilities of continuing a new deletion or starting a new deletion is given by

$$D_{i,j} = \max\{D_{i,j-1} - c,\ M_{i,j-1} - g\}. \quad (3)$$

The score for the possibility that the current matrix location represents a match or mutation is complicated by the fact that different degrees of matching are allowed. A substitution matrix $d(a_i,b_j)$ is specified which gives a different weight for every possible pair of characters a_i and b_j that can appear at the matrix position. The three possible conditions are that this match/mutation ends an insertion, ends a deletion, or continues a string of match/mutation. A maximum over these three conditions finds the optimum choice. In order to do a local alignment where distant misalignments do not penalize finding a matching substring locally, the score is not allowed to fall below zero. The resulting equation is

$$M_{i,j} = \max\{I_{i-1,j-1} + d(a_i,b_j),\ D_{i-1,j-1} + d(a_i,b_j),\ M_{i-1,j-1} + d(a_i,b_j),\ 0\}. \quad (4)$$

A Smith-Waterman ASIC coprocessor has previously been designed [11]. However, this coprocessor was not designed using a GALS methodology and therefore designing a new clock tree every time the design is ported to a CMOS process with smaller feature size may be time consuming. Fig. 3 shows a design that is amenable to being divided into independent computing elements that can reside in individual local clock domains, thus allowing for the GALS design approach.

The two major parts of the compute element are the constants section (left) and the calculations section (right). The flow of information into and out of the constants section is independent of information flow for the calculation section as evidenced by distinct pairs of request/acknowledge signals for each section. This requires two copies of the asynchronous interface with buffer of Fig. 1 to be placed between each compute element and the compute element that follows it in the chain. The datapath for the constants interface is 8 bits wide and that of the calculations interface is 85 bits wide.

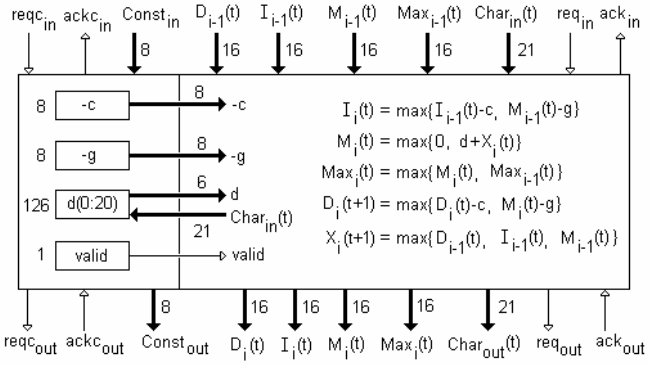

Figure 3. A compute element of the Smith-Waterman coprocessor.

Each compute element holds the data associated with one character of one of the sequences of the alignment. If there are not enough compute elements to hold the entire sequence, multiple passes through the chain of compute elements will be required.

The characters of the sequence which is not assigned to the compute elements flow through the chain of elements. The characters are given as a 21-bit one-hot code in vector *Char*. A one-hot code is used to allow easy access to the portion of the substitution matrix stored in the constants section. There are 21 possible values allowing for each of the 20 possible amino acids plus a character for "unknown."

Only a single column of the substitution matrix is stored in the constants section since one character of the pair is always the same for all calculations. Six bits are required for each value to allow for the range of values in the commonly used substitution matrices. For example, the very commonly used PAM250 matrix [12] has minimum value of -7 and maximum value of 17. These matrices all contain integer values. The gap start and continuation penalties can easily be stored as 8-bit values since these are integers with magnitude normally less than 100. The *valid* bit allows a compute element to be bypassed if there are more compute elements than sequence characters for a given pass.

Some of the comparisons need to do the match/mutate maximum calculation can be done before the current *Char* value arrives. The auxiliary variable *X* is introduced to allow this pre-calculation. Most of the operations done for Smith-Waterman are comparisons. These would be done in the adder unit of a microprocessor, but can be implemented more efficiently in hardware specifically built to do value comparisons. The output of the value comparator controls a multiplexer which passes the larger value.

V. CONCLUSION

Use of the GALS design methodology can greatly reduce the design effort required to create clock trees for large high-frequency integrated circuits. The use of multiple clock domains requires attention to the possibility of metastability and possible resulting synchronizer failure. It also generally increases signaling latency across clock domain boundaries.

The GALS interface presented here has the advantage of not requiring the interface to be allowed to pause the local clock signals of the attached local clock domains. The cost of removing the pausing requirement is that the probability of synchronizer failure can never be made zero, although it can be made arbitrarily close to zero.

Potential applications of the interface and associated GALS design methodology are single-chip multiprocessors and special-purpose coprocessors which are easily scalable to more processors or more coprocessor units as CMOS feature sizes decrease.

REFERENCES

[1] D. Chapiro, *Globally-Asynchronous Locally-Synchronous Systems*, Ph. D. Thesis, Stanford University, 1984.

[2] I. Sutherland and S. Fairbanks, "GasP: A Minimal FIFO Control," *IEEE Inter. Symp. on Asynchronous Circuits and Systems*, pp. 46-53, 2001.

[3] H. Veendrick, "The Behavior of Flip-Flops Used as Synchronizers and Perdiction of Their Failure Rate," *IEEE J. of Solid State Circuits*, vol. 15, pp. 169-176, 1980.

[4] J. Muttersbach, T. Villiger, H. Kaeslin, N. Felber, and W. Fichtner, "Globally-Asynchronous Locally-Synchronous Architectures to Simplify the Design of On-Chip Systems," *IEEE Inter. ASIC/SOC Conf.*, pp. 317-321, 1999.

[5] K. Yun and A. Dooply, "Pausible Clocking-Based Heterogeneous Systems," *IEEE Trans. on Computers*, vol. 45, pp. 482-488, 1999.

[6] S. Smith, *A Multiple-Clock-Domain Bus Architecture Using Asynchronous FIFOs as Elastic Elements*, Ph. D. Thesis, University of Idaho, 2003.

[7] S. Furber, *ARM System-on-Chip Architecture*, 2nd Ed., New York: Addison-Wesley, 2000.

[8] S. Smith, "The Advanced Encryption Standard on an Asynchronous Shared-Memory Multiprocessor," *Inter. Conf. on VLSI*, pp. 379-381, 2003.

[9] S. Smith and J. Frenzel, "Bioinformatics Application of a Scalable Supercomputer-on-Chip Architecture," *Inter. Conf. on Parallel and Distributed Processing Techniques and Applications*, vol. 1, pp. 385-391, 2003.

[10] T. Smith and M. Waterman, "Identification of Common Molecular Sequences," *J. of Molecular Biology*, vol. 147, pp. 195-197, 1981.

[11] L. Grate, M. Diekhans, D. Dahle, and R. Hughey, "Sequence Analysis with the Kestrel SIMD Parallel Processor," *Pacific Symp. on Biocomputing*, pp. 263-274, 2001.

[12] D. Mount, *Bioinformatics: Sequence and Genome Analysis*, New York: Cold Spring Hardor Laboratory Press, 2001.