Reliable Asynchronous Links for SoC

Ethiopia Nigussie, Juha Plosila, and Jouni Isoaho
Communication Systems Laboratory
Department of Information Technology, University of Turku
Turku, Finland
{ethnig, juplos, jisoaho}@utu.fi

***Abstract*— this paper presents two asynchronous links between any two independently clocked synchronous modules. The first link is based on using synchronizers and synchronous and asynchronous FIFOs which compensates the increase of latency due to synchronization. Due to this the latency of this link is reduced to 2.08nsec. The mean time between failures of this link is 35 years, which is more than enough for any design. The second link generates clock for each module locally and stops it whenever there is communication between module and link. In this link there is no synchronization failure at all. The latency and power consumption of both links are very small which makes them efficient links for SoC. Since the two link architectures let the use of different clocks in each synchronous module, it makes the system modular and enables easy re-usage of different synchronous modules in the system. The circuits are simulated using the analog environment of Spectre with 0.13um technology.**

I. INTRODUCTION

Due to technology scaling, gigascale integration of devices on a single chip becomes a reality. At this integration level clock distribution and alignment has become an increasingly challenging problem, consuming an increasing portion of resources such as wiring area, power, and design time. Furthermore, with the advent of Systems-on-a Chip (SoC), synchronizing all blocks of a large SoC to a single global clock without degrading performance becomes a very difficult task. Because of this modern SoC designs are compositions of several independently clocked subsystems which communicate each other through different ways.

One of the solutions to this problem is to divide the system into several optimized clock domains which interact via self-timed handshake signaling [1]. Such a system enables flexible use of stoppable clocks providing automatic power down of idle system modules. Also the clock skew-related timing problems are restricted to relatively small locally synchronous modules. In addition to these, it makes easy to have a modular system.

In this paper, we presented two different types of asynchronous links between any two independently clocked modules. The first link is based on the principle of using synchronizers and additional circuitry which helps to make use of the clock cycle taken by the synchronizers. The second one uses stoppable clock when there is a risk of metastability. This is possible by generating the clock from a delay line to each module.

Therefore, in this design the problem of clock skew are not anymore system level design issues. Instead the clock skew and routing problems are limited only to a locally synchronous module which is relatively small compared to the whole system. This simplifies the design process of a complex system enabling easy re-usage of different synchronous nodes. Furthermore, the asynchronously controlled stoppable clocks make the automatic power-down operation of currently idle blocks possible. [2]

The two link architectures are presented in Section II, and their performance analysis in Section III.

II. LINK ARCHITECTURES

As we know in SoC, there are many processing elements (blocks) which communicate with each other. Every block consists of sender, receiver and data processing modules. Four phase handshake signaling is used for both links. The architectures of the two different types of links are presented as follows

A. Link I

As it is shown in Figure 1, there are active sender and passive receiver. The active sender initiates the communication by sending a request to the passive receiver and the passive receiver accepts the data and sends back an acknowledgment.

Both the synchronous and asynchronous FIFOs help in handling burst mode data transfer besides compensating the throughput degradation which occurred due to the two synchronization delays. These FIFOs store the data while the synchronizer is synchronizing the acknowledgment signal with clock 1 and the request signal with clock 2. The size of these two FIFOs is determined by many factors. The main factors are the two clock frequencies, in other words, the data production and consumption rate, the average data traffic in

0-7803-9294-9/05/$20.00 ©2005 IEEE

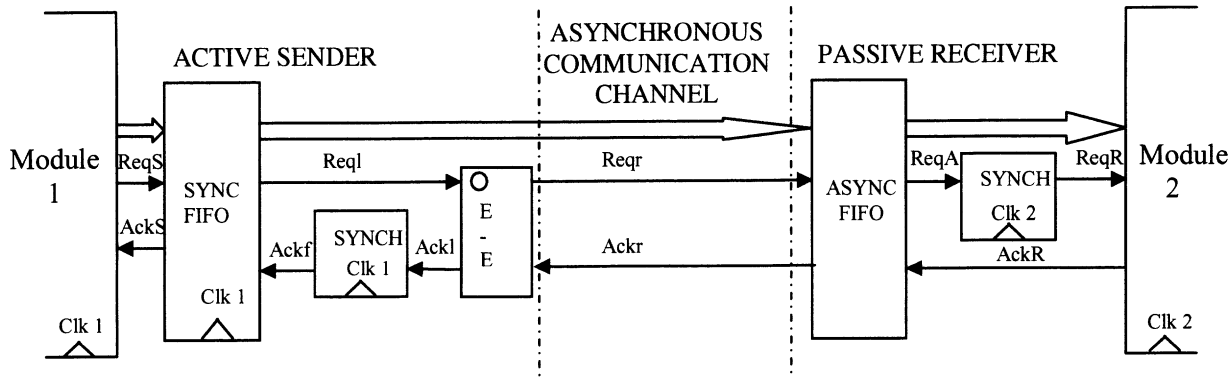

Figure 1. General Block Diagram of Link I

the link and the delay due to synchronizations.

The synchronous FIFO uses a ring counter for generating the write and read pointer. When the FIFO receives request signal from the sender, first it checks whether the FIFO is full or not. If it has space for the incoming data, the enable signal for the write counter becomes high. This in turn makes the write pointer to start enabling the data latch to store the incoming data. At the next rising edge of the clock the acknowledgment signal will be sent back to the sender. The acknowledgment signal goes down at the next clock edge after the request signal goes down by the module. As soon as the synchronous FIFO stores one word data or whenever it is not empty, it sends request signal to the E-element. The E-element in turn sends request to the asynchronous FIFO control unit.

The E-element shown in Figure 1, starts performing its task after detecting a request event, Reql, from the synchronous FIFO. Then Reqr will be in up going phase of the handshake cycle and starts then the return to zero phase immediately after sending an acknowledgement through the passive port. Here the E-element serves in saving time by making the initialization of the request and acknowledgment signal in parallel. The gate-level design of E-element and its lower asymmetric C-element is shown in Fig 2. The lower asymmetric C-element acts as C-element for falling output transitions. One of the two inputs, b, is like an active-high set signal. The VHDL abstraction of this gate is as follows,

```
if (b='1') then c <= '1';
elsif (a='0') and (b='0') then c <= '0';
else c <= c;
end if;
```

To read the data from the synchronous FIFO, the synchronized acknowledgment signal which comes from the E-element is used as enable input for the read counter, which generates the read pointer for the FIFO. If the asynchronous FIFO has place to store the word, it accepts the data and sends acknowledgment to the E-element. In parallel with this it sends request to the next control unit and so on. Finally the request reaches to the receiver after synchronized with clock 2. Then the receiver sends acknowledgment back after accepting the data.

There are two synchronizers in this link as it is shown in Figure 1. We used brute force synchronization for both synchronizers [3].

The asynchronous FIFO is a simple micropipelined datapath with control unit. It uses an enabled latch for storing the data and the control unit which helps in generating the handshake and the enable signals for the data latch. The data latch is active-low enabled by a lock signal, which is also used as a request signal for the next latch. The control logic used for this FIFO is shown in Figure 3. The control unit has two three input upper asymmetric C-elements and one NOR gate with delay logic.

The upper asymmetric C-element acts as a C-element for rising output transitions. One of the three inputs (a) is like an active-low reset signal. The output is cleared by setting this input low. The VHDL abstraction of this gate is as follows,

```
if (a='1') and (b='1') and (d='1') then c <= '1';
elsif (a='0') then c <= '0';
else c <= c;
end if;
```

One of the three inputs Reqi and the inverted AckO of the left and right upper asymmetric C-element respectively acts like an active low reset signal. The ReqO signal is used as an input request for the next control unit and an active-low enable signal (Lock) for its data latch. This signal locks the data latch as far as the data is not taken by the consecutive data latch or in case of the last data latch by the module. When the FIFO is full all the data latches get locked.

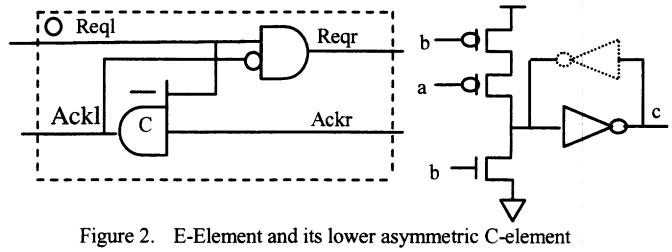

Figure 2. E-Element and its lower asymmetric C-element

*[[Reql] ; Reqr↑ ; [Ackr] ; Ackl↑ ; Reqr↓ ; [¬Reql ∧¬Ackr] ; Ackl↓]
communication action of E-element, [ ] means wait; *[ ] means repeat

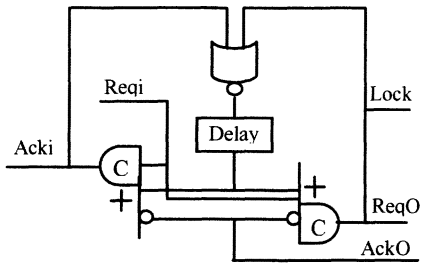

Figure 3. Control Unit of the Asynchronous FIFO

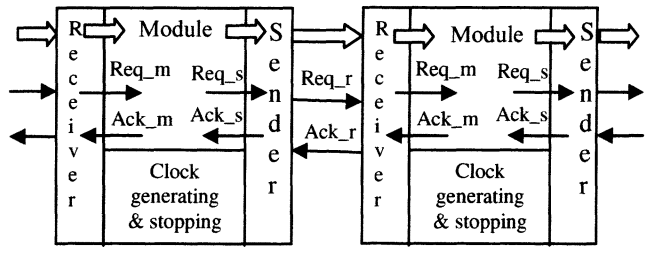

Figure 4. Block diagram of Link II

B. Link II

The block diagram of this link is shown in Figure 4, it has sender, receiver, synchronous module (processing element) with its clock generating and stopping circuitry. In each module a specific clock is generated locally and whenever there is a need of communication between the sender and module or receiver and module the generated clock will be stopped.

If there is no request either from sender or receiver to communicate with the module, the output clock is inverted and then fed back to a delay line and to one input of the arbiter 2. The arbiter then grants in the favor of the clock circuit and is merged with the output from the delay line. This oscillatory process continues and a stable clock signal is produced. The C-element is used to hold the state of the clock whenever arbiter 2 grants in favor of request.

There are two clock stopping requests, one from the sender to send acknowledgment to the module and the other from the receiver to send request to the module. The block diagram of clock generating and stopping is shown in Figure 7.

Arbiter 1 grants in the favor of one of the request to stop the clock and then Arbiter 2 grants either the clock or the granted request from arbiter 1. One of the requests to stop the clock gets acknowledged whenever arbiter 2 grants the request not the clock. The upper asymmetric C-element is used to allow the clock to run after every grant of the request.

The sender requests to stop the clock if there is request from the module to transfer data to another module, if the sender is not sending request to transfer data to receiver, if there is no acknowledgment from the sender to the module and if the clock is not stopped due to request of stopping from the sender. The block diagram of sender module is shown in Fig. 5. It consists of the active-high enabled data latch, AND gate, E-element which has the same advantages as in link I, two active-low reset D-flip flops and upper asymmetric C-element.

The receiver module is the same as passive receiver of link I with few exceptions, Link II receiver doesn't need synchronizer and instead of N place asynchronous FIFO it has one place FIFO. In addition to these it has E-element, the rise in request from the control unit, causes an immediate rise in request to stop the clock and as soon as the clock stopping gets acknowledged the request to the synchronous module will be high. The logic diagram of Link II receiver is shown in Figure 6.

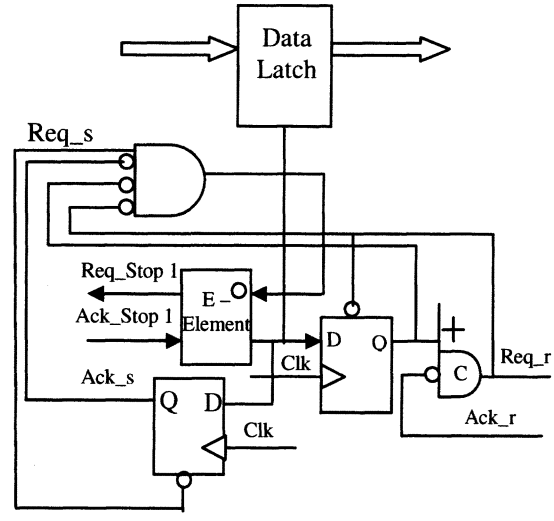

Figure 5. Link II Sender

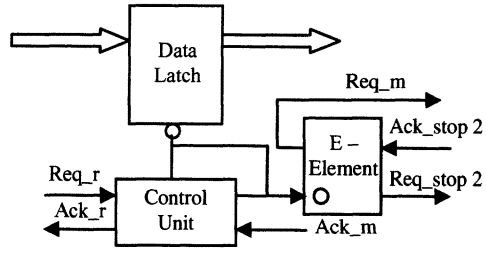

Figure 6. Link II Receiver

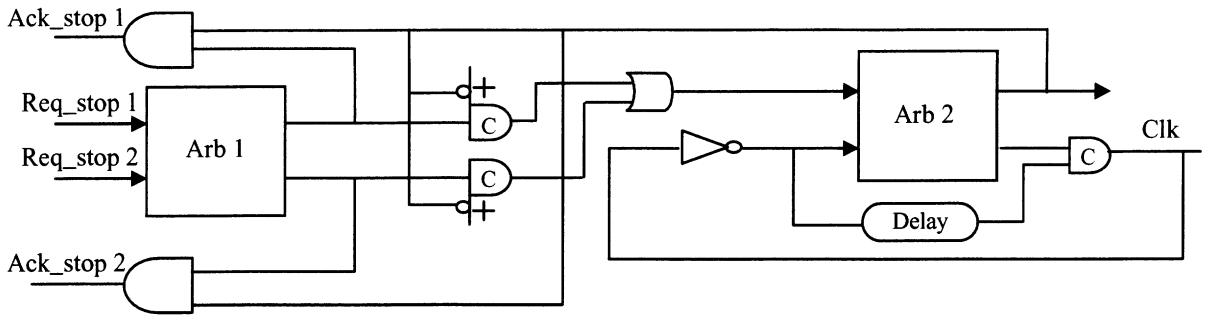

Figure 7. Clock generating and stopping of Link II

III. PERFORMANCE ANALYSIS

As shown in Fig. 1, Link I has risk of synchronization failure and increase in latency due to the time needed to synchronize the request and acknowledgment signal to the two independently clocked modules. In this design, we used two different types of FIFOs, synchronous and asynchronous to compromise the throughput degradation due to synchronization delay. The size of each of the two FIFOs in this design is three and with these sizes it works correctly as it is verified by simulation. There is no data dropping or duplicating.

To have reliable link throughout the life time of the system, the link should have mean time between failures greater than the life time of the system. Since the link is designed with 0.13um technology, regeneration time constant, τ = 30psec. Taking average clock frequency of the two modules, f_{clk} = 800MHz and data frequency, f_{data} = 200MHz, the mean time between failures using the standard formula [4] is 35 years. This is long enough for any design since almost all design life time is less than 35 years.

In Link II, instead of using synchronizer to prevent metastability failure, clock stretching is used when there is a risk of metastability. So this means synchronization failure is not any more concern of Link II. Furthermore the arbiter has metastability filter, metastability is a delay issue not a failure. For example in Arbiter 2, if the clock falls and request goes high around the same time then the arbiter may go metastable. This is safe since the arbiter guarantees that the clock cannot continue and the request can not be propagated until the metastablility has resolved.

As it is seen from Table I below, link I latency is almost three times of Link II. This is due to the time taken by the synchronization. The power consumptions on both links are very small and almost comparable.

TABLE I. PERFORMANCE ANALYSIS

PERFORMANCE PARAMETER	LINK I	LINK II
Latency (nsec)	2.08	0.7
Power consumption (μW)	54.63	65.54

IV. CONCLUSIONS

Two different asynchronous links for any two independently clocked modules are presented. The first link uses synchronizers to communicate between the modules and the asynchronous link. As it is well known using synchronizers has risk of failure and in our design the mean time between failures is 35 years, which is more than enough for any design. The speed of this link mainly depends on the data production and consumption rate, the average data traffic on the channel and the size of the FIFOs. As expected the latency of this link is three times greater than link II latency, this is due to synchronization. The two links have almost equal power consumption. In Link II there is no synchronization failure, instead there may be little increase in latency to resolve the metastability. The two links are designed using four phase signaling, we are continuing the link performance analysis for two phase signaling and will make conclusion about which signaling is efficient.

The presented link architectures illustrate two feasible ways to construct a globally asynchronous locally synchronous system with very small synchronization failure rate or without any synchronization failure. Both links have very small latency and power consumption, which makes them more suitable for SoC communication channel. Furthermore, the two links architecture allows truly modular design and easy plug and play of different synchronous modules, since no global timing assumptions have to be made. In addition, the physical timing problems are made local, and thereby the simulation and verification tasks are simplified.

REFERENCES

[1] J. Sparso, S. Furber, Principles of Asynchronous Circuit Design—A System Perspective, Kluwer Academic Publishers, Dordrecht, 2001.

[2] L. Plana and S. Nowick Architectural Optimization for Low-Power Nonpipelined Asynchronous Systems. In IEEE Trans. On VLSI Systems, 6(1) March 1998.

[3] W.J. Dally, J.W. Poulton, Digital System Engineering, Cambridge University Press, Cambridge, 1998.

[4] C. Dike and E. Burton, "Miller and Noise Effects in a Synchronizing Flip-flop," IEEE Journal of Solid-State Circuits, 34(6), pp. 849-855, 1999.