Notice of Violation of IEEE Publication Principles

"Efficient RTL Design of SoCWire BUS Protocol"
by Ravi Kumar, R.K. Sarin, and Sarabjeet Singh,
in the Proceedings of the 2011 World Congress on Information and Communication Technologies (WICT), December 2011, pp. 1073-1078

After careful and considered review of the content and authorship of this paper by a duly constituted expert committee, this paper has been found to be in violation of IEEE's Publication Principles.

This paper contains significant portions of text from the papers cited below. The original text was copied without attribution (including appropriate references to the original author(s) and/or paper title) and without permission.

Due to the nature of this violation, reasonable effort should be made to remove all past references to this paper, and future references should be made to the following articles:

"Advanced System-on-Chip Design with In-Flight Reconfigurable Processing Cores for Space Applications"
by B. Osterloh, H. Michalik, B. Fiethe, and K. Kotarowski
Data Systems In Aerospace (DASIA), Palma de Majorca, May 2008

"SoCWire: A Network-on-Chip Approach for Reconfigurable System-on-Chip Designs in Space Applications"
by B. Osterloh, H. Michalik, B. Fiethe, and K. Kotarowski
NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2008), pp. 51-56, Noordwijk, June 2008

"System-on-Chip Wire (SoCWire) User Manual"
by B. Osterloh
www.socwire.org. SoCWire V1.0, 22.04.20094.

"SpaceWire Inspired Network-on-Chip Approach for Fault Tolerant System-on-Chip Designs"
by B. Osterloh, H. Michalik,and B. Fiethe,
in Dynamic Reconfigurable Network-on-Chip Design: Innovations for Computational Processing and Communication, IGI Global, April 2010

Efficient RTL Design of SoCWire BUS Protocol

Ravi kumar[1], R.K.Sarin[2], Sarabjeet Singh[3]
Department of Electronics & Communication [1,2],Department of Physics[3]
Dr. B. R. Ambedkar National Institute of Technology Jalandhar
kumarravi0086@gmail.com[1], sarinrk@nitj.ac.in[2], singhs@nitj.ac.in[3]

Abstract*—Configurable System-on-chip (SoC) approach for the Venus Express Monitoring Camera (VMC) has been successfully demonstrated in space. For future space missions e.g. Solar Orbiter, the demand for high performance onboard processing has drastically increased. To achieve these advanced design goals a Reconfigurable System-on-Chip (RSoC) architecture is proposed supported by an on chip flexible communication architecture. In order to communicate between static (which is running during the whole application runtime and stores all critical interfaces) and dynamic partial reconfiguration we design dedicated Network-on-Chip (NoC) approach called SoCWire .This SoCWire provides guaranteed system qualification with hot-plug ability, high speed point-to-point connection and support of the adaptive macro-pipeline as compared to the Bus Macros which suffers from more area and power consumptions. In this work, we present RTL mode SoCWire Codec for point to point communication. The model operates at maximum operating frequency 491 MHz.

Keywords-component; formatting; style; styling; insert (key words)

I. INTRODUCTION

Single chip integrated circuits are commonly referred to as *system-on-chip* (SoC), and typically consist of several complex heterogeneous components such as programmable processors, dedicated (custom) hardware to perform specific tasks, on-chip memories, input–output interfaces, and an on-chip communication architecture that serves as the interconnection fabric for communication between these components. The dual forces of advances in technology, coupled with an insatiable demand for convergent computing devices (e.g., smart phones that include cameras, GPS devices, MP3 players) have fueled the need for complex chips that incorporate multiple processors dedicated for specific computational needs. These emerging Network-*on-chip* (NoC) designs typically consist of multiple microprocessors, and tens to hundreds of additional components [1].

NoC-based SoC design uses two major concepts that are distinguishable from those of bus-based SoC architecture. There are packet transactions rather than circuit transactions, and there is a distributed network structure instead of a conventional globally shared bus or a centralized matrix.

BUS PROS & CONS			NETWORK PROS & CONS
Every unit attached adds parasitic capacitance, therefore electrical performance degrades with growth.	-	+	Only point-to-point one-way wires are used, for all network sizes.
Bus timing is difficult in a deep submicron process.	-	+	Network wires can be pipelined because the network protocol is globally asynchronous.
Bus testability is problematic and slow.	-	+	Dedicated BIST is fast and complete.
Bus arbiter delay grows with the number of masters. The arbiter is also instance-specific.	-	+	Routing decisions are distributed and the same router is Re-instantiated, for all network sizes.
Bandwidth is limited and shared by all units attached.	-	+	Aggregated bandwidth scales with the network size.
Bus latency is zero once arbiter has granted control.	+	-	Internal network contention causes a small latency.
The silicon cost of a bus is near zero.	+	-	The network has a significant silicon area.
Any bus is almost directly compatible with most available IPs, including software running on CPUs.	+	-	Bus-oriented IPs need smart wrappers. Software needs clean synchronization in multiprocessor systems.
The concepts are simple and well understood.	+	-	System designers need *reeducation for new concepts.*

Table 1: - Bus Vs NoC

In NoC-based SoC design, each of the functional modules should be designed to be latency-insensitive, to support packet transactions. Although this makes functional module design slightly more complex, many benefits are gained from having packet transactions. It improves reliability and the speed of interconnection links, because packet transactions are intrinsically pipelined so that the physical lengths of interconnection links can be kept short. Efficient link utilization is another advantage, because only part of the end-to-end path between functional modules is occupied by the traversing packets. It is also advantageous that the electrical parameters of one NoC are not influenced by the addition of other NoC modules, owing to the structured characteristic of the NoC. This enables the building up of large scale SoCs from smaller existing components by the a addition of any required functional modules. For all these reasons, advanced bus architectures, too, are gradually considering a packet transaction concept into their protocols; examples are multiple-outstanding addressing, split transactions, and multi-threaded transactions [2, 3]. In the near future, it is expected that commercial bus architectures for a high-end SoC will take the NoC design into their specifications [4, 5]. The paper is organized as follows: Section 2, present related work in bus protocols. This section presents exhaustive survey of traditional buses, segmentation, SAMBA BUS and Bus arbitration. Section 3 presents State machine, codec architecture and state machine of FIFO.

II. Related Work

In this section, traditional System-on-Chip interconnects, their design, operation, and constraints are discussed.

A. Traditional Buses

Shared buses like AMBA and CoreConnect have long been the traditional interconnect of choice for SoC designers because of their simple topology and extensibility. By providing a standardized interconnect, these shared buses allow designers to stitch together various hard and soft IP cores in a SoC. The general structure of such buses consists of some number of masters and slaves, an arbiter, the data bus, and the address bus. In the context of SoCs, masters are Processing Elements (PEs) that are allowed to initiate communication, and slaves are the PEs that can only respond to masters. The bus arbiter coordinates all communications between masters and slaves through a series of control signals.

The data bus itself is not a single monolithic broadcast bus, but instead two buses – read and write – that are each separated into two parts by multiplexers and OR gates in AMBA and CoreConnect, respectively. The address bus is similarly split into two portions by multiplexers and OR gates. Both AMBA and CoreConnect are *address-mapped* buses, meaning that slaves are bound to certain portions of the memory space. Such an addressing scheme is very suitable for processor/memory configurations. When a processor (master) wishes to perform a memory operation (with some slave), it simply asserts the address that it wants to read from or write to. A simple address decoding allows the appropriate slave to respond to the request. AMBA and CoreConnect are also *transaction-based* buses, that is, communication over the bus occurs in atomic transactions. Bus transactions occur in two phases – the address phase and the data phase.

When a slave is unable to respond to a request quickly (e.g., a slow memory performing a read), the slave can request a *transaction split*. The split breaks atomicity, allowing the transaction to be completed later while releasing the bus to other masters in the meantime. However, when the slave wishes to complete the request, it must undergo an arbitration delay to gain access to the bus.

B. Bus Segmentation

A problem present in traditional shared bus architectures is that a single bus transaction consumes all of the bus resources. An example of this is as follows: Several PEs (1 through 6) sit on a linear bus. If PE1 is communicating with PE3, then no other communication can take place on the bus. A common solution to this – known as segmentation – is to break up the bus into segments that can independently support transactions. If the bus in the previous example were segmented into two halves, then PE1 and PE3 could communicate on one segment while PE4 and PE5 communicate on the other segment. Figure 5 shows how segmentation would be applied to this example.

The SAMBA-BUS pushes bus segmentation to the limit by providing a segmentation point at each PE on the address and data buses. An additional improvement that SAMBA-BUS makes over traditional shared bus architectures is that even if a PE does not win the arbitration, it may still be allowed to access the bus, based on a set of rules which prevent overlapping communication from occurring. The downside to SAMBA-BUS's improved architecture is that the bus suffers from increased delay due to multiplexers in the critical path, and increased interface logic at each PE.

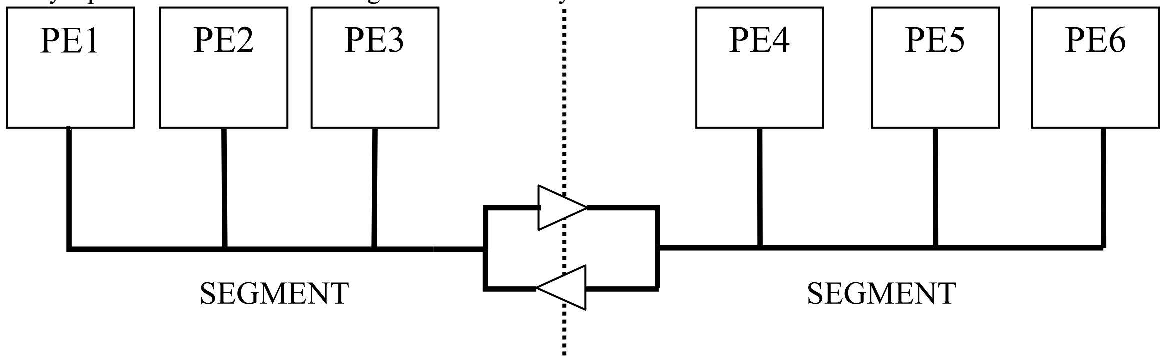

Figure 1: - Bus Segmentation [4].

C. *Bus Arbitration*

Arbitration policy plays a large role in determining the performance of buses, and as such, LOTTERYBUS is an improvement to arbitration policy. LOTTERYBUS performs arbitration by randomized selection of a bus master based on the master's ownership of a "ticket." Each master can be statically or dynamically assigned a number of tickets. Upon arbitration, the arbiter chooses a ticket in a uniform random fashion from a pool of the tickets of masters with pending requests – a process that reduces the starvation of low-priority PEs. LOTTERYBUS can also effectively guarantee bandwidth to certain PEs by allocating more tickets to them.

However, the drawback to the LOTTERYBUS is a relatively high hardware requirement. The arbitration that LOTTERYBUS performs is simple, yet the hardware implementation is quite complicated.

III. RTL Modeling of SoC Wire

SoCWire is a Network on chip approach (NoC) in a complete on-chip environment has been developed to provide a robust communication architecture for the harsh space environment and to support dynamic partial reconfiguration in future space applications. SoCWire It is based on the SpaceWire standard. The Space Wire interface is a well established standard, providing a layered protocol (physical, signal, character, exchange packet, network) and proven interface for space applications. It is an asynchronous communication, serial link, bidirectional (full-duplex) interface including flow control, error detection and recovery in hardware, hot-plug ability and automatic reconnection after a link disconnection. SpaceWire is a serial link interface and performance of the interface depends on skew, jitter and the implemented technology. Therefore SpaceWire interface has been modified to a parallel data interface. The advantage of this approach is that significantly higher data rates can be achieved as compared to the SpaceWire standard. Additionally, a scalable data word width to support medium to very high data rates has been implemented. On the other hand the advantageous features of the SpaceWire standard including flow control, hot-plug ability, error detection and link re-initialization are still fully support. The SoCWire character level, exchange level, packet level and network level is derived from the SpaceWire standard and refers to it. In Contrast to SpaceWire the SoCWire architecture does not require the physical layer.

A. *Design of SoC Wire Codec*

The SoCWire CODEC connects a node or host system to a SoCWire network. It consists of six modules.

- State machine
- Receiver
- Receiver FIFO
- Transmitter
- Transmitter FIFO
- Dual port Ram

The state machine controls the overall operation of the link interface. It provides link initialization, normal operation and error recovery services. The operation of the state machine is described in the form of a state diagram given below.

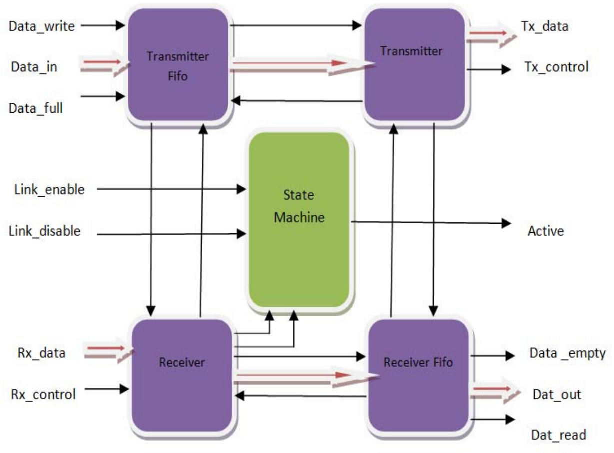

Fig.2 Block Diagram of SoC Wire Codec

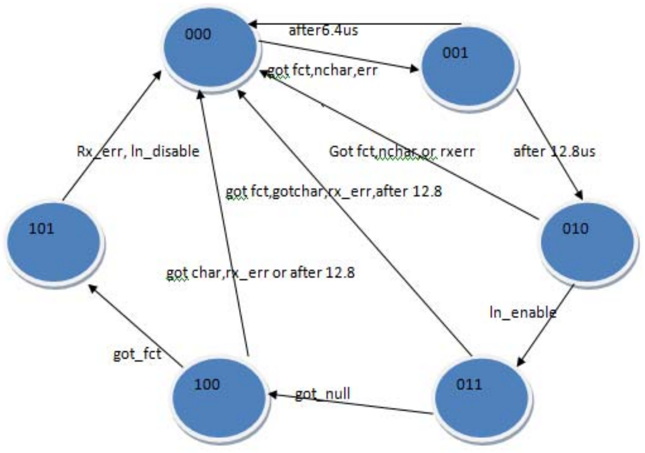

Fig.3 State Machine

Transmitter FIFO can be access with the low active data write signal. As soon as the FIFO is full the data full signal is '1'. FCTs received by the receiver are forwarded with the fct write signal to the transmitter. The fct_write increments the credit counters. If the credit counter receives 7 FCTs the fct_full signal is set. For a bi-directional full-duplex transfer the FCTs have to be included in the data transfer. The tx FIFO does not require a dedicated ram, because it just has to store 1 data word. There is one Special condition while making this block is Signal Swallow. Swallow '1' means transmitter will not receive (read) the data from tx-FIFO .Therefore data_empty must be 1when swallow is 1.it is one for non run state. For run state If data is successfully transferred & we got eop then it will be equal to zero Otherwise it remains one. By the concept of Swallow we get another condition of Data full. Suppose we successfully transmit data previously and got Eop =1 If state is non-running state Data full=0 & swallow =1, For running state swallow still maintain previous condition .i.e. one.

For swallow =1,Eop=1,data write is enable Data_in =0xxxxxxxx ,DATA FULL will be equal to ONE whatever will be the read.FCT can be full when fct counter value is from 49 to 55.

It receives N-Chars for transmission from the host system. If there is neither FCT nor an N-Char (data, EOP or EEP) to transmit, the transmitter sends NULL. The transmitter sends N-Chars only if the host system at the other end of the link (end B) has room in its host receive buffer.

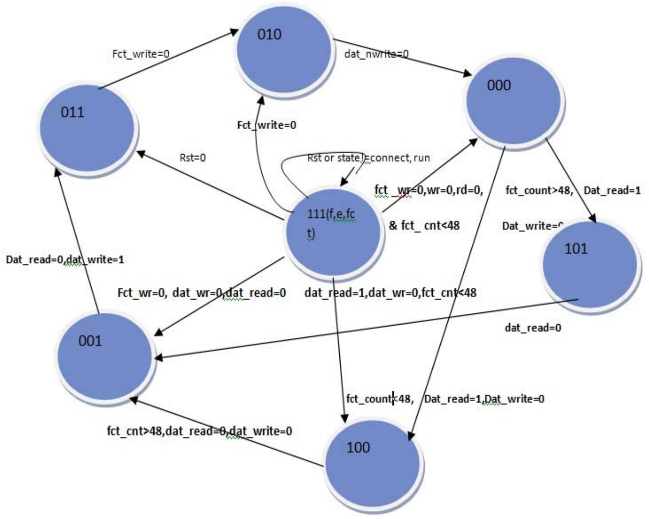

Fig.4 State machine of Tx FIFO

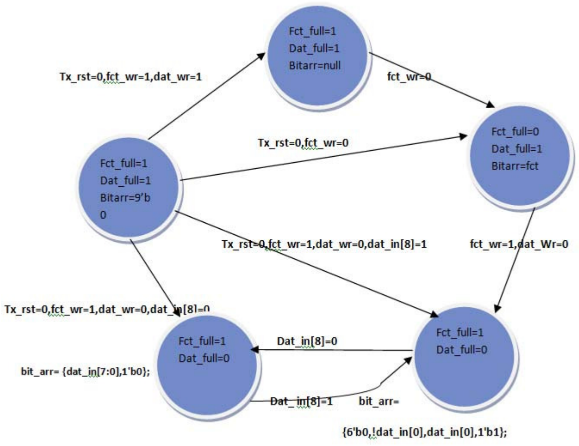

Fig.5 State Machine of Transmitter

This is indicated by the link interface at end B sending an FCT, showing that it is ready to accept another 8 N-Chars. The transmitter is responsible for keeping track of the FCTs received and the number of N-Chars sent to avoid input buffer overflow at the other end of the link. To do this the transmitter holds a credit count of the number of characters it has been given permission to send.

The transmitter can be in one of four possible states:

a. Reset: The transmitter does nothing.

b. Send NULLs: It can only send NULL on the link. It does not read N-Chars from the Transmit Host Interface. It does not accept an order to send FCT from the Host System.

c. Send FCTs or NULLs: It sends FCT or NULL but still does not read N-Chars from the Transmit Host Interface.

d. Send FCTs, N-Chars or NULLs: Normal behavior, sending NULLs, FCTs and N-Chars. The transmitter is also responsible for sending FCTs whenever the local Host System has space to receive eight more N--Chars . The host system requests the transmitter to send FCTs when it has room for at least eight more N--Chars that have not already been reserved for data by requesting a previous FCT. The transmitter can only send an FCT when it is in state“Send FCTs or NULLs” “Send FCTs. N-Chars or NULLs”.

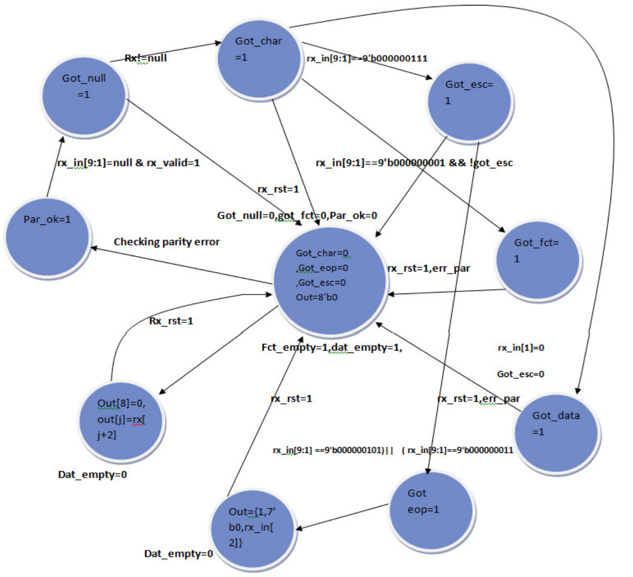

Fig.4 State Diagram of receiver

It receives NULLs FCTs. NULLs represent an active link. They are flagged to the exchange-level state machine but are ignored otherwise .When an FCT is received the receiver informs the transmitter so that it can update its credit count accordingly. All other control characters received are flagged to the state machine. The receiver ignores any N-Chars, L-Chars, parity errors or escape errors until the first NULL is received. The disconnection detection mechanism within the receiver is enabled as soon as the first bit arrives. The receiver is also responsible for detection of disconnect, parity, escape and credit errors and it flags these errors to the state machine.

The receiver can be in one of four states.

a. Reset: The receiver does nothing.

b. Enabled: The receiver is enabled and is waiting for the first bit to arrive.

c. Got Bit: The receiver has received the first bit (First Bit Received) and disconnect error detection is enabled. The receiver is enabled to listen for NULLs only.

d. Got NULL: The receiver has received a NULL and is enabled to receive NULLs, FCTs and N-Chars .Disconnect, parity and escape error detection is enabled.

The change of state from Reset to Enabled is controlled by the state machine .The receiver is responsible for receiving FCTs from the other end of the link and for passing these FCTs on to the credit counter in the transmitter.

The receiver FIFO works in two region connecting and run state. The output signal fct empty which is connected to the input of fct-write play the vital role in the working of the codec. Initially fct empty is one so that fct write is off since it is active low signal. At that time only NULL data is send from the transmitter and causes the state change from ready to connecting state. In connecting state the output of the transmitter fct full turn to ‘0’which is none other than ‘fct read’ an input of receiver FIFO. Since state is now connecting this module starts working now. For fct read equal to

zero ,fct empty turn to be zero and make the counter increase by '8'.In this way it shows that eight free space is available in the receiver FIFO and transmitter can send at most eight data now(but only after state machine reaches to run state).
After allotting eight data again fct empty turn off i.e 1(active low)So at the next clock cycle transmitter response by sending FCT to the Receiver side and change the state from Connecting to Run State. At another cycle the fct counter increases by '7 or 8' (depending upon condition)means it has same no of space to store data.it is interesting to see that the fct counter of transmitter FIFO and receiver FIFO increases at the same instant that means transmitter has liberty to send and receive has space to receive. In this way the full depth of the receiver FIFO is allotted. The counter decreases one by one when read signal is enable depending upon the condition. The receiver FIFO can be access from the host interface with the low active "dat_nread" signal. The fill state of the FIFO is indicated with the high active "dat_empty" signal. If "dat_empty" is '0', data can be read from the FIFO.

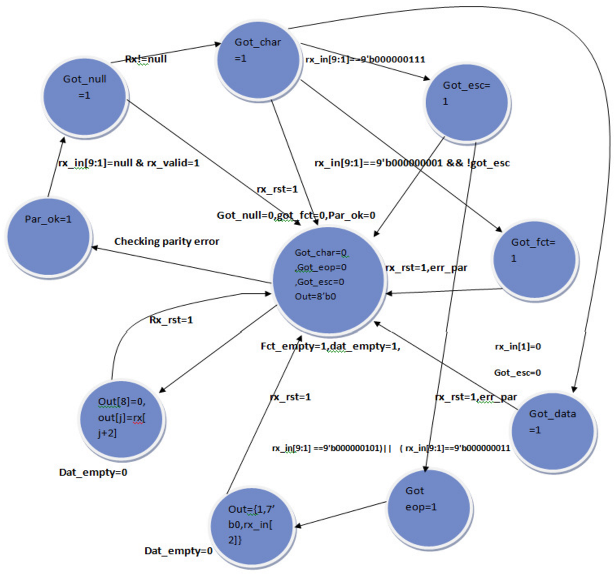

Fig.5 State Diagram of receiver

IV. Synthesis Report

# Adders/Subtractors	1
10-bit addsub	1
# Counters	4
10-bit up counter	3
14-bit up counter	1
# Accumulators	3
10-bit updown accumulator	1
6-bit updown accumulator	2
# Registers	75
Flip-Flops	75
# Comparators	3
10-bit comparator lessequal	1
6-bit comparator greater	1
6-bit comparator lessequal	1
# Multiplexers	47
1-bit 2-to-1 multiplexer	30
1-bit 7-to-1 multiplexer	1
10-bit 2-to-1 multiplexer	7
6-bit 2-to-1 multiplexer	4
9-bit 2-to-1 multiplexer	5
# FSMs	1
# Xors	3
1-bit xor10	1
1-bit xor2	1
1-bit xor9	1

# BELS	338
# GND	2
# INV	4
# LUT1	38
# LUT2	31
# LUT3	35
# LUT4	19
# LUT5	40
# LUT6	63
# MUXCY	49
# MUXF7	1
# VCC	2
# XORCY	54
# FlipFlops/Latches	136
# FD	18
# FDR	28
# FDRE	79
# FDS	9
# FDSE	2
# RAMS	1

The design Operates at Frequency 491MHz.

V. Conclusion

We have presented architecture of SoCWire. The SoCWire is a Network on chip approach in a complete on-chip environment that has been developed to provide a robust communication architecture for the harsh space environment and to support dynamic partial reconfiguration in future space applications. SoCWire It is based on the SpaceWire standard. The Space Wire interface is a well-established standard, providing a layered protocol (physical, signal, character, exchange packet, network) and proven interface for space applications. It is an asynchronous communication, serial link, bidirectional (full-duplex) interface including flow control, error detection and recovery in hardware, hot-plug ability and automatic reconnection after a link disconnection. We have synthesized the RTL model on FPGA device and found operating frequency of 491 MHz.

References

[1] Sudeep Pasricha, Nikil Dutt, "*On-Chip Communication Architectures*". Morgan Kaufmann Publications, U.S., 2008.

[2] P. Guerrier and A. Greiner, "*A Generic Architecture for On-Chip Packet-Switched Interconnections*," Proc. Design, Automation and Test in Europe (DATE '00), pp. 250-256, Mar. 2000.

[3] AMBA AXI specification.

[4] OCP 2.0 protocol specification.

[5] STBus functional specifications, STMicroelectronics public web support site, http://www.stmcu.com/inchtmlpages-STBus_intro.html, April 2003.

[6] Goossens, Kees et al. (2005) Æthereal network on chip: concepts, architectures, and implementations. IEEE Des. Test Comput., 22, 414–421.

[7] Chrysostomos Nicopoulos, Vijaykrishnan Narayanan, Chita R. Das, "*Network-On-Chip Architectures*". Springer Publications, Volume 53, U.S., 2009.

[8] L. Benini and G. de Micheli, "Networks on chips: a new SoC paradigm," Computer 35 (2002) (1): 70–78.

[9] A. JantschandH.Tenhunen, eds., Networks on Chip.NewYork: Kluwer Academic Publishers, 2003.

[10] E. Beyne, "3D system integration technologies," In International Symposium on VLSI Technology, Systems, and Applications, Hsinchu, Taiwan, April 2006, 1–9.

[11] Brett S. Feero, ParthaPratimPande, "Networks-On-Chip in a Three Dimensional Environment: A Performance Evaluation", IEEE Transactions on Computers (TC), vol.58, no. 1, pp. 32-45, January 2009.

[12] Natalie Enright Jerger, Li-ShiuanPeh, "On-ChipNetworksSynthesis Lectures On Computer Architecture", Morgan & Claypool Publishers, University of Wisconsin, Madison, 2009.

[13] Jeong-Ho Woo, Ju-Ho Sohn, Byeong-Gyu Nam, Hoi-Jun Yoo, "MOBILE 3D GRAPHICS SoC From Algorithm to Chip", John Wiley & sons, Singapore.

[14] William James Dally, Brian Towles, "Principles and Practices of Interconnection Networks", Morgan Kaufmann Publishers, 2004, U.S.

[15] Yuval Tamir and Gregory L. Frazier. Dynamically-allocated multi-queue buffers for VLSI communication switches. *IEEE Transactions on Computers*, 41(6):725–737, June 1992. DOI: 10.1109/12.144624

[16] Hubert Kaeslin, ETH Zurich, "Digital Integrated Circuit Design from VLSI Architectures to CMOS Fabrication," Cambridge University Press, 2008.

[17] Jagrit Kathuria, M Ayoub Khan, Arti Noor, "A Review of Clock Gating", MIT International Journal of Electronics & Communication Engineering (MITIJEC), August, 2011.

[18] S. Murali,, N. Vijaykrishnan, M.J. Irwin, L. Benini, and G. De Micheli, "Analysis of error recovery schemes for networks on chips," IEEE Design & Test of Computers, pp.434-442, Volume 22, Issue 5, Sep. 2005.

[19] N. Genko, D. Atienza, G. De Micheli, J. M. Mendias, R. Hermida, and F. Catthoor, "A Complete Network-On-Chip Emulation Framework," Proceedings of the conference on Design, Automation and Test in Europe (DATE' 05), pp.246-251, Vol.1, 2005.