2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
DECEMBER 6 - 7, 2022

A Reconfigurable Interface Architecture to Protect System IP

Dr.Arshad Riazuddin, Center for Advanced Research in Engineering (CARE), Islamabad, Pakistan (*riazuddin.arshad@gmail.com*)

Dr. Shoab A. Khan, Center for Advanced Research in Engineering (CARE), Islamabad, Paksitan (*shoab@carepvtltd.com*)

***Abstract*— In this paper, we present an on the fly reconfigurable interface which can be used for on-chip or off-chip interconnect. The reconfiguration of the interface at a variable rate without any dependence on a reconfigurable fabric is one of its plus points. The chameleon like changing of the interfaces providing connectivity in a system can thwart any intelligent or brute force attacks to decipher the functionality of the system and its interfaces. This architecture has applications for interconnecting chiplets, implementation in Internet of Things (IoT) and Cyber Physical Systems (CPS).**

Keywords*— *Reconfigurable IP, chiplets, IoT, CPS

I. INTRODUCTION

In this paper we extend the concept of intellectual property (IP) reusability and protection through reconfiguration of communication interfaces between different components in a system. An on-the-fly reconfigurable interface is presented which can be used to secure intercommunication between different integrated circuits (IC) in a system or inside an Application Specific Integrated Circuit (ASIC) built on the chiplet concept. In our proposed idea the interface can be reconfigured at a variable rate to prevent information leakage through monitoring and deciphering of the interface. The proposed reconfigurable interface is not dependent on any Field-Programmable Gate Array (FPGA) features, and can be easily implemented in ASIC methodologies.

The problem of information leakage in FPGA/ASIC, using, temperature [1], [2] static leakage [3], execution time [4], dynamic power consumption [5], [6] have been studied and solutions proposed for them. In [5], [6] the researchers show that long routing wires are a new source of information leakage on FPGAs, and the effect is measurable for both dynamic and static signals across multiple families of Xilinx devices (Virtex5-6, Artix7, and Spartan7). The authors in [6] propose countermeasures of using: Xilinx Isolation Design Flow tools which allow physically isolating IP cores inside a FPGA, adding guard wires around transmitting wires, generating random data bits on wires. Researchers have proposed and developed different techniques to protect the IP inside these IC as well as in a system. The authors in [7], [8], [9] propose a technique based on access control policies and behavior learning techniques for detection of unwanted behavior. Protecting the system from information leakage attacks in order to steal IP, to modify their functionality or to render them unusable are active research areas.

The novel architecture proposed in this paper is not dependent on any specific tool flows as in [6]. Our proposed architecture is focused on protecting information leakage through interconnect. While the approach outlined in [7], [8], [9] is focused on protecting hardware from Trojan attacks, which can be carried out by integration of third party IP in a SoC using Root of Trust methodology.

II. PROPOSED ARCHITECTURE

To protect critical IP elements in a system the reconfigurable interface (RI) can be added to various interfaces of a system. The reconfigurable interconnect has the capability to support multiple industry standard or proprietary interfaces for interconnection between modules/chiplets of various functionality. The ‘on the fly’ configurable interface prevents adversaries from using snooping methods to monitor the communication interfaces between two entities.

The RI is a reconfigurable block which can support both custom and proprietary serial interfaces. The RI block consists of a programmable sequencer along with data path elements. The read/write protocol of multiple buses can be encoded using the sequencer instructions into the sequencer program memory. Figure 1 shows the architecture of the RI.

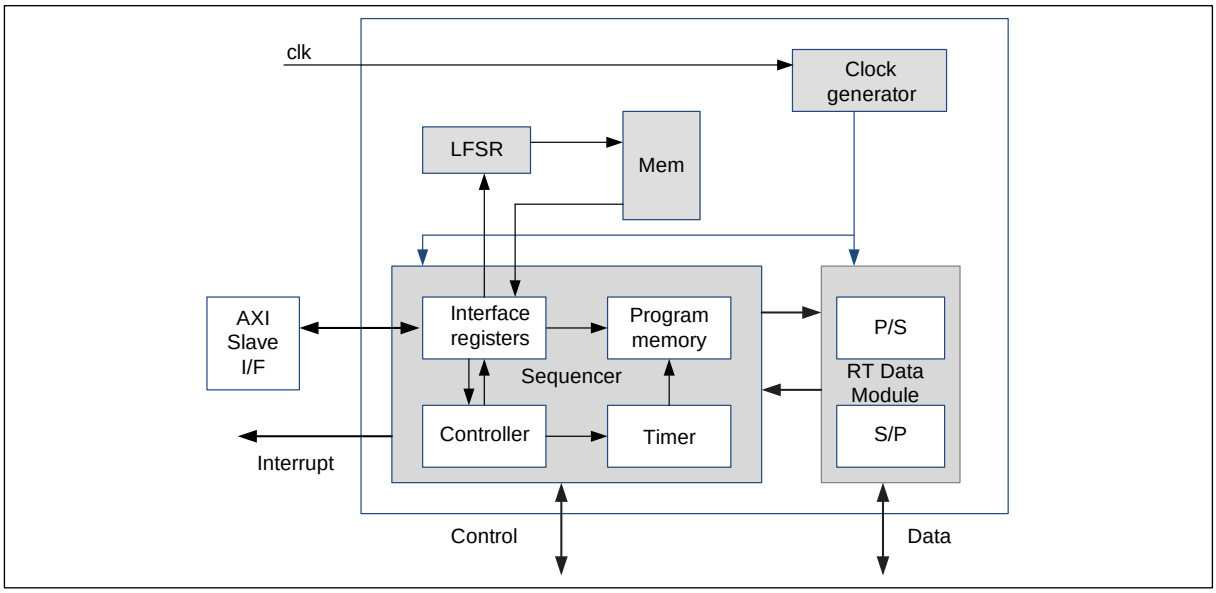

Figure 1 Block Diagram of the Reconfigurable Interface (RI) Architecture

A. *Sequencer*

The sequencer is the heart of the proposed architecture as it allows the reconfigurable interface to change the bus protocols on the run. The sequencer consists of the following submodules:

- Controller
- Program Memory
- Timer
- Interface Registers

The controller block is responsible for: decoding the instructions read from the program memory, the address generation unit for generating the next address, and accepting reconfiguration information from the interface register block. An analysis was performed of the common industry standard protocols which are used to interconnect various peripherals in a typical system. Based on this analysis a list of OPCODES was generated which comprehensively implements majority of the interfaces. Table I lists the OPCODES supported by the RI

Table I. Opcodes of the Sequencer

Opcode	***Description***
No Operation (NOP)	No Operation
Sample Flag (SAMPF)	Wait for an input flag to be equal to a particular value, before going to next instruction
Compare Flag (CMPF)	Compare the flag, before going to next instruction
Jump Unconditionally (JUMP)	Jump unconditionally to the address specified in the control word
Jump Conditiioanlly (JUMPC)	Jump conditionally to the address specified in the control word

Table II describes the control word of the sequencer, which is organized as a 32-bit word.

Table II. Control Word of the Sequencer

Bits	***Description***
31:29	Opcode
28:21	Jump address used in jump instruction
20:0	Wait for conditional flags which are used to move to the next instruction in the sequence

The program memory contains the microcode for the various buses which the module will support. The controller will generate the control signals to the program memory for accessing the data at particular memory locations. There are many instances where the sequencer has to wait for some particular timed events in order to start a new cycle. For example in certain serial buses like Integrated Inter-IC Sound Bus (I2S), Audio Serial Port (ASP) etc. a frame signal is generated periodically to which the entire data transmission/reception process is aligned. The timer module which is provided in the sequencer can be used to generate such a frame alignment signal and the sequencer will also align the data transmission/reception with this frame signal.

The interface registers module implements control/status registers and the data buffers implemented in the RI. An Advanced eXtensible Interface (AXI) slave interface is provided for accessing the interface registers module. These registers can be used to configure programmable parameters of the RI bus interfaces. The programmable clock generator is used to support interfaces of different clock speeds. For example the size of data transmission/reception, parity bits, clock polarity on which to align data/control signals and clock speed of the interface. Hence, this would allow us to support Serial Peripheral Interface (SPI) mode 0, 1, 2, 3; different addressing modes of Integrated Inter-IC Control Bus (I2C) interface; various sizes of data transmission reception (8/16/24/32 bits) for any interface; different configurations of the Universal Asynchronous Receiver Transmitter (UART), start/stop bits, data bits, even/odd parity etc. This reconfiguration allows to further add deception within the ever changing external interface. The generated clock is used as clock enable signal in the sequencer block. The AXI slave interface is used to program the desired clock rate.

B. R/T Data Module

The RT data module performs the parallel to serial (P/S) and serial to parallel (S/P) data conversion. For transmission case the data is written into the P/S converter through the AXI slave interface. The AXI slave interface also programs the P/S module as to how much data is to be transmitted. The data transmission on the serial interface happens under the control of the sequencer block. Once, the data transmission is completed the sequencer is informed about this through an empty signal.

For reception case the data is written into the S/P converter through the serial interface. The AXI slave interface programs the S/P module as to how much data is to be received. The sequencer through a control signal informs the S/P module as to when the data reception will start. Once, the data reception is completed the sequencer is informed about this through an empty signal.

C. Variable Rate Interface Change

The purpose for changing the interface at variable rate is to protect the interfaces in a system from hardware snooping, resulting in IP disclosure of a system. Therefore, a mechanism has been developed for changing the functionality of the interface at variable rate in the RI. For this purpose a linear feedback shift register (LSFR) has been implemented in the RI module. A pseudo-random number is generated by the linear feedback shift register (LFSR). The random number from the LFSR is used as an address pointer to a memory inside the RI module, called the LFSR memory. There are two types of data present in the LFSR memory: Transaction Data, LFSR Data. The LFSR memory is organized as 1K depth and 64-bits wide. The transaction data word consists of the bus interface code, programmable interface parameters (parity, clock polarity etc.), the clock frequency at which the interface

will work and the delay parameter between the start of back to back transactions. Figure 2 shows the transaction data word structure in memory. The transfer of the direction (read/write) comes through the AXI slave from a master in the System on Chip (SoC)/FPGA, but the bus interface code comes from the LFSR memory. The combination of the bus interface code, and the transfer direction will point to the starting address in the program memory. Hence, change in the starting address of the microcode corresponds to RI functionality changing on the fly.

When the RI is implemented in multiple devices in a system, the OS will load a common seed into the LFSR at boot time. This will make the LFSR behave in a similar manner in all the devices in the system. The LFSR memory address is incremented linearly and the transaction word read out from the memory is used as the basis to transfer data on the RI bus interface. While, sequentially addressing the LFSR memory the address can hit a location where LFSR data is present. The LFSR data is identified by a special code present in the upper 8-bits of the word. The LFSR memory jumps to the address pointer in the LFSR data for starting a new transaction. If the memory address hits these locations then the new value of LFSR is transmitted to the other devices in the system over a fixed interface.

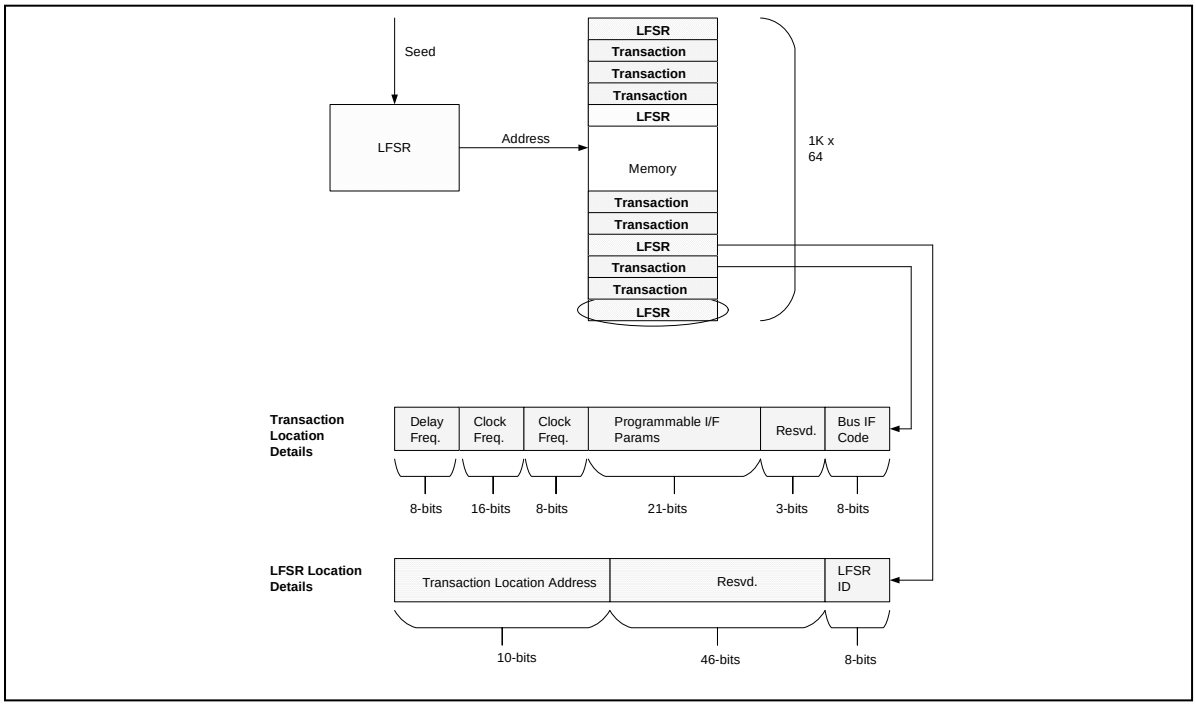

Figure 2 Word Structure and Details of the LFSR Memory

Table III. Description of the LFSR Transaction Word

Bits	***Description***
63:56	RI Bus interface code
55:53	Reserved
52:32	Programmable I/F Parameters
31:24	Frame generation frequency. The frequency of the frame generation, e.g. for ASP. The frame generation depends on the word width of the interface.
23:8	RI interface frequency. The frequency at which the RI interface is running. It is some multiple of the clock frequency at which the entire RI module is running.
7:0	Delay between back to back transaction

III. IMPLEMENTATION RESULTS

The proposed architecture of the RI has been executed and various industry standard interfaces have been implemented to verify the architecture. The interfaces implemented so far are: SPI, UART and ASP. These interfaces have been verified in simulations and validated. The implementation results on a ZYNQ UltraSCALE+ device (XCZU28DR-FFVG1517-2-e) for the RI interface are given in Table III.

Table IV. Implementation Results

RI Resource Utilization	
Components	***Count***
LUT	780
Flip-Flop (FF)	570
LUTRAM	176
BRAM	2

A. Simulation

The microcode developed for SPI, UART and ASP was verified in simulations before being validated.

Figure 3 shows a SPI write cycle with an address and data phase generated by the SPI master implemented in the sequencer. This is followed by an ASP transmit cycle generated by the ASP master implemented in the sequencer. The SPI clock stops generating once the transaction is finished and the ASP bit clock is generated. The sequencer address and OPCODE can be seen in the waveform.

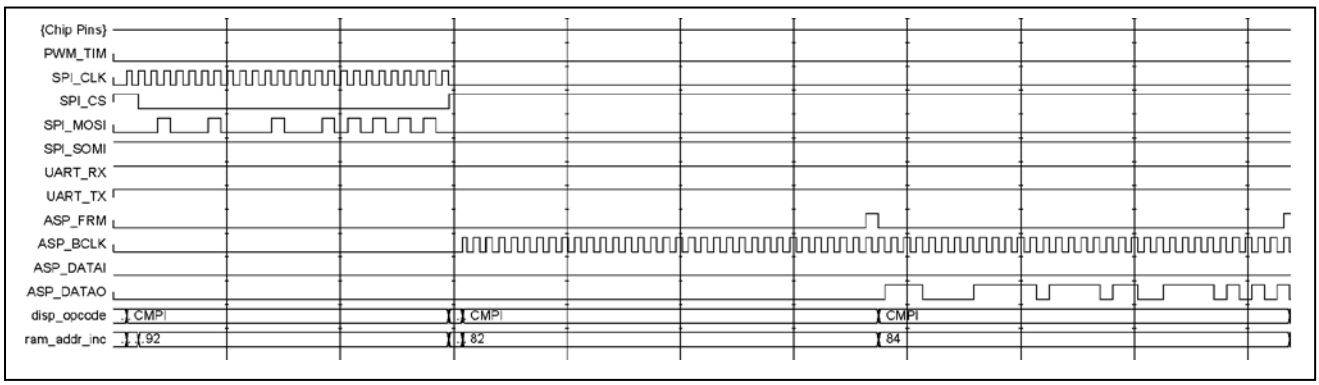

Figure 3 Combination of SPI Write Cycle followed by ASP Transmit Cycle

IV. CONCLUSION

This paper presents a novel on the fly RI which can be used to protect the IP of SoC/ASIC/FPGA from being compromised through intelligent or brute force attacks. In a communication system frequency hopping concept is used to change the communication frequency at regular intervals, to guard against eavesdropping and frequency jamming by the enemy. In a similar manner, the functional behavior of interconnect between various IC's in a system can be changed at regular intervals for guarding against information leakage using our proposed concept. The second contribution of this paper is that the RI is independent of any underlying FPGA technologies such as reconfiguration/partial reconfiguration technologies of FPGA, and is fully portable to any ASIC technology.

REFERENCES

[1] M. Gag, T. Wegner, A. Waschki and D. Timmerman, "Temperature and on-chip crosstalk measurement using ring oscillators in FPGA," in IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)., 2012.

[2] M. Hutter and J.-M. Schmidt, "The Temperature Side Channel and Heating Fault Attacks," in CARDIS 2013: Smart Card Research and Advanced Applications, 2014.

[3] A. Moradi, "Side-Channel Leakage through Static Power.," Cryptographic Hardware and Embedded Systems – CHES 2014. Lecture Notes in Computer Science, vol. 8731, pp. 562-579, 2014.

[4] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," Advances in Cryptology — CRYPTO '96. CRYPTO 1996. Lecture Notes in Computer Science, pp. 104-113, 1996.

[5] I. Giechaskiel, K. B. Rasmussen and K. Eguro, "Long-Wire Leakage: The Threat of Crosstalk" IEEE Design & Test, vol.39, no.4, pp.41-48, 2022.

[6] I. Giechaskiel, K. Eguro and K. B. Rasmussen, "Leakier Wires: Exploiting FPGA Long Wires for Covert- and Side-Channel Attacks," ACM Transactions on Reconfigurable Technology and Systems, vol. 1, no. 1, 2019.

[7] D. M. Shila, V. Venugopalan and C. D. Patterson, "FIDES: Enhancing Trust in Reconfigurable Based Hardware Systems," in Proceedings of the 2015 IEEE High Performance Extreme Computing Conference (HPEC), 2015.

[8] V. Venugopalan, C. D. Patterson and D. M. Shila, "Detecting and Thwarting Hardware Trojan Attacks in Cyber-Physical Systems," in Proceedings of 2016 IEEE Conference on Communications and Network Security (CNS): International Workshop on Cyber-Physical Systems Security (CPS-Sec), 2016.

[9] V. Venugopalan and C. D. Patterson, "Architectural Refinements for Enhancing Trust and Securing Cyber-Physical Systems," in 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), San Francisco, CA, USA, 2017.