electronics

Article

A Knowledge Base Technique for Detecting Multiple High-Speed Serial Interface Synchronization Errors in Multiprocessor-Based Real-Time Embedded Systems

Sabeen Masood [1,*], Shoab Ahmed Khan [1], Ali Hassan [1] and Fatima Khalique [2]

[1] Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology (NUST), Islamabad 44000, Pakistan
[2] Department of Computer Science, Bahria University, Islamabad 44000, Pakistan
* Correspondence: sabeen.masood@ceme.nust.edu.pk

Abstract: The heterogeneity of the multiple processing elements (PEs) is a feature of real-time embedded systems. General-purpose processors and several embedded processors, as well as dedicated high-speed interfaces, are among these elements. Communication between the processors is among the most significant characteristics of developing such complex systems. Furthermore, synchronization is a common issue during interprocessor communication in embedded systems. Debugging and testing such systems is time-consuming, difficult, and laborious, with the majority of the complexities centered on debugging real-time interprocessor communication, such as synchronization in terms of timing and accuracy. While the hardware design features of heterogeneous multiprocessor real-time embedded systems have received a lot of attention, the design and development of software-based solutions still have the potential to be addressed. In particular, software-based testing becomes challenging due to interprocessor communication and the synchronization of real-time applications. A knowledge-based technique that aids in testing high-speed serial interfaces in multiprocessor-based real-time embedded systems is proposed that needs debugging in real time while an application is running. It is becoming much more important to test and validate these interfaces in real time as the demand for high data transmission rates increases. The presented work uses a technique to simulate, create and enhance the knowledge base used as correlation-based error detection that reduces the development time. The proposed technique helps in detecting synchronization-related errors that occur during communication among multiple high-speed serial interfaces. The presented work also lists a series of experiments to validate the effectiveness of the proposed technique. The results show that the presented techniques are effective for error identification in real-time embedded systems.

Keywords: real-time and embedded systems; embedded processors; interfaces; synchronization; interprocessor communication; debugging; testing

Citation: Masood, S.; Khan, S.A.; Hassan, A.; Khalique, F. A Knowledge Base Technique for Detecting Multiple High-Speed Serial Interface Synchronization Errors in Multiprocessor-Based Real-Time Embedded Systems. *Electronics* **2022**, *11*, 2945. https://doi.org/10.3390/electronics11182945

Academic Editors: Alexander Barkalov, Larysa Titarenko, Dariusz Kania and Remigiusz Wiśniewski

Received: 1 August 2022
Accepted: 13 September 2022
Published: 17 September 2022

1. Introduction

Modern real-time embedded systems are demanding in terms of computing power and programmable performance; thus, the number of processors used in these applications is increasing [1–3]. System-on-Chip (SoC) designs have multiple processing elements (PEs) on the same die for the sake of meeting the exponential growth in the recent processing requirements of embedded system applications. These processing components usually share the system memory and peripherals while execution, thus requiring synchronization to ensure system integrity [4]. Tasks can be distributed to each processor in multiprocessor systems to minimize the overall execution time, since the processors are tightly coupled [5]. Multiple processors make it easier to give quick responses and satisfy the requirements of real-time and functional complexity for complex real-time systems [6]. The field of real-time embedded systems encompasses a wide variety of systems, from basic control loops on microcontrollers to complex integrated distributed systems. These systems are

used in a variety of applications such as energy management [7], autonomous vehicles [8,9], medical systems [10], avionics [11] and consumer electronics [12]. Many of the latest multiprocessor systems are heterogeneous, such as a multiple digital signal processors (DSPs), advanced RISC machines (ARMs), general-purpose processors (GPPs) or microcontrollers, field-programmable gate arrays (FPGAs), application-specific integrated circuits (ASICs), etc. In general-purpose computing, heterogeneous multiprocessors are less common. The combinations of heterogeneous multiprocessors pose significant challenges when compared with the symmetric multiprocessor world [13]. Accordingly, acquiring software from many types of processors and making them operate together becomes extremely challenging. The hardware design features of heterogeneous embedded multiprocessor systems have received a lot of attention in the last few years. Meanwhile, the challenge of creating efficient software for this kind of platform still has not been fully addressed yet [14]. With several PEs, a real-time embedded system connected via different physical interfaces (synchronous and asynchronous) using interprocessor communication is shown in Figure 1. Real-time interprocessor communications and time synchronization are important in ensuring good performance for critical applications in multiprocessor-based embedded systems [15]. Improving the communication efficacy among the processors is one of the key challenges of an embedded system implementing real-time processing applications. Normally, the main issues of an embedded system implementing a real- time processing application are the interprocessor communications and synchronization. Since a single processor is insufficient for real-time processing, therefore, a way is required for the processors to synchronize their communication. Processors must synchronize themselves when they collaborate with each other [16]. Synchronization on a multiprocessor is expensive due to the hardware levels at which synchronization and communication must happen [17]. The situation becomes even worse when the CPU used to interact with other processors or other hardware components [18] is interrupted, which is common in today's multiprocessor systems. Moreover, the resulting gap in the bandwidth of the interface among the memory and processors has become a critical hurdle for total system performance as the data rate demands have increased [19]. The demand for increased bandwidth interfaces in computing devices has led to the adoption of high-speed interfaces identical to those used in communication devices [20]. In today's computing industry, featuring communication systems as well as data centers [21], high-speed serial interfaces (HSSI) are prevalent [22]. Therefore, the demand for availability, reliability, and high performance in real-time embedded systems has resulted in more complex computing systems. Developing systems of this complexity necessitates the use of sophisticated validation and verification approaches [23]. The development time is reduced, and difficulties are blocked in the realm of real-time embedded systems if the errors are detected and fixed earlier in the product cycle. It is significantly important to understand and examine the behavior in all the possible situations, thus better achieving the overall system quality for complex systems [24].

As the complication of real-time and embedded systems grows, the demand for developing the methodologies and tools that give productive ways for the efficient development of relevant embedded software is growing as well. Developers often devote a considerable portion of their time to testing and correcting errors [25], which is aided by advanced software testing and debugging tools [26]. However, testing such programs is typically more difficult than testing sequential applications, and it is made much more challenging by the heterogeneous parallelism in a multiprocessor system on chip (MPSoC). Simultaneous environments introduce new types of errors in the error taxonomy that do not exist in sequential programs, which is one reason for the increased complexity [27]. Furthermore, due to their complexity, debugging and testing these multiprocessor systems during the design phase is particularly difficult and time-consuming, as the majority of the complexities are found in debugging real-time interprocessor communication, which is synchronous in terms of accuracy and timing. One of the biggest challenges in the creation of embedded system projects, according to [28], is testing and debugging [29]. Several tools have been proposed

for the development of embedded software, with the validation activity accounting for more than 60% of the total development effort [30].

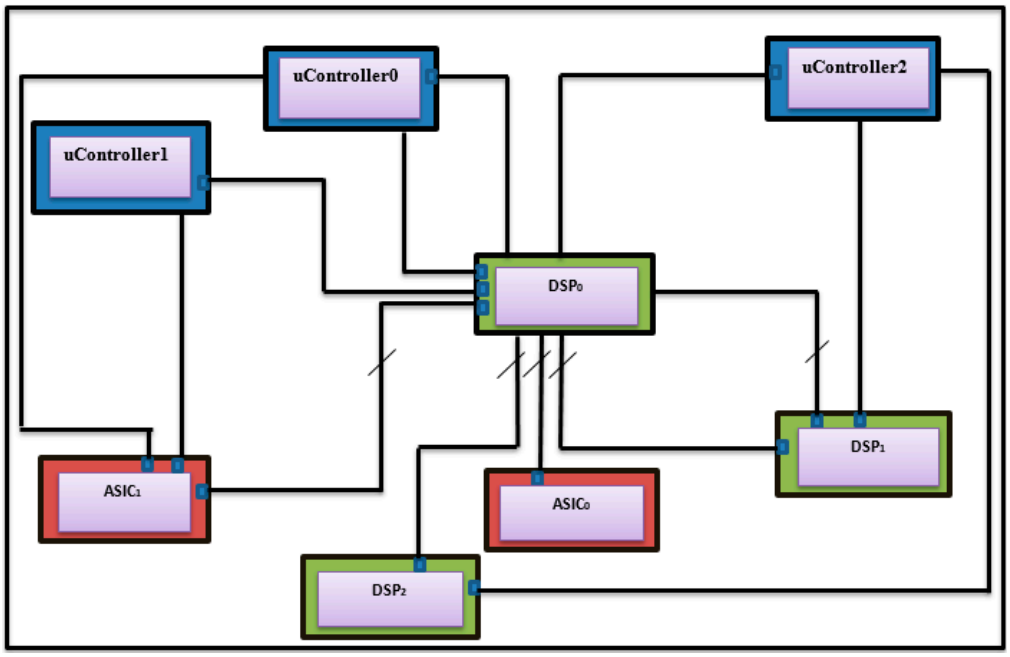


Figure 1. Multiple PEs with physical interfaces.

Despite the availability of a wide range of testing and debugging techniques, there are still weaknesses in the testing and debugging process that may be addressed, most notably in comprise between time, cost, and test efficiency. As embedded systems have become more complex, specialized testing methodologies have become more recognized than general testing frameworks for embedded systems. In addition to errors in the program logic, timing and accuracy problems in interprocessor communication are possible. Interchanging among several interfaces might incur significant time delays through transitioning or processing a different interface when the processors employ a real-time operating system due to time constraints. Furthermore, emulating high-speed interfaces is difficult and uncontrolled in order to mimic real-time events. The developers have tested all the functionality of the PEs separately, but still, the multiprocessor-based real-time embedded system fails due to the high-speed serial (asynchronous as well as synchronous) interfaces. High-speed interfaces transport a large amount of data across multi PEs during interprocessor communication. Finding synchronization and timing errors is extremely difficult without observing the trace of System under Test's (SuT's) execution due to the high-speed interfaces. So, there is a lot of possibility for research in this field to look at ways to verify interprocessor communications in real-time settings.

The goal of this work tackles the complicated problem of testing and debugging high-speed serial interfaces while an application is running, which requires real-time testing and debugging. A variety of factors, i.e., the computational complexity of executing different algorithms, the kind of operating system that runs on a real-time system, etc. affect interprocessor communications. The research was driven by the experience of building a multi-PEs based real-time embedded system in which the most challenging part of the implementation was the testing and debugging of high-speed serial interfaces with several apparent and concealed factors affecting interprocessor communication. When high-speed processing is essential in real-time embedded systems, certain kinds of issues arise in numerous fields, i.e., ultrasound systems [31], automobile electronic control unit applications [32], software-defined radios [33], radar applications [34], etc. As the demand for the high data transfer rises, testing and debugging high-speed interfaces become much more complex [35]. Particularly, testing the application with high-speed serial

interfaces is the focus in this work. The proposed knowledge base technique is created based on synchronization errors encountered on single as well as multiple interfaces when communication between multi PEs occurs in a real-time embedded system. When an interface error is received on the real-time embedded system, we compare it with the knowledge base. To localize the error, we also capture the data through the multiple emulators at run time. This allows us to identify where the synchronization error is and which interface is causing this error. The proposed technique rapidly addresses errors while also examining and diagnosing the behavior of multiple connected serial interfaces with complex internal architectures. The results show that our presented technique is effective in discovering and fixing issues that would otherwise be difficult to find and fix.

The research contribution is to design and develop a knowledge base that aids in the testing of multiple high-speed serial interfaces in multi-processor-based real-time embedded systems, as well as to simulate, store and detect all possible synchronization-related errors that occur during PE communication.

The rest of the paper is structured as follows. The related work is discussed in Section 2. The conceptual idea and the implementation of the approach are discussed in particular in Sections 3 and 4, respectively. An experimental evaluation of our work is provided in Section 5. Sections 6 and 7 summarize and conclude this paper.

2. Related Work

Several techniques for testing and debugging high-speed serial interfaces have been explored in this section and are shown in Table 1. It is becoming more difficult to develop fault-free electronic equipment as data rates and integration levels grow. As a result, post-fabrication validation currently consumes about 25% of the total design resources at Intel [36]. In the target environment, real-time embedded system debugging and testing has traditionally become difficult and time-consuming due to the inherent absence of internal system visibility [37]. Furthermore, as the need for increased bandwidth grows, high-speed serial interfaces (HSSI) are being pushed toward larger data rates [38]. Post-silicon validation, testing, and debugging of HSSI has made it more crucial to maintain the design and device quality with the coinciding increase in the complexity of the design and decrease in the timing budget [39]. It has also become a serious issue and has become much more difficult to address as a consequence of longer test times, jitter, noise, signal integrity issues, and the usage of expensive instruments [40]. As the data rate grows for transferring data faster, it eventually becomes significant to test and verify these interfaces throughout the development [41]. It is critical and typically takes the majority of the design time.

Testing and Debugging of High-Speed Interfaces

Various methods for testing as well as debugging embedded systems' high-speed serial interfaces have been proposed by several studies. A solution to the issue of decreased General Purpose Input/Outputs (GPIO) pin availability during manufacturing tests and running complete structural content while embedded in a functioning system is to use the functional protocol of an existing HSIO port for testing proposed in [42]. By exploiting the high-speed functional interfaces that are already present in a SoC, such as PCIe or USB, the approach effectively executes both scan tests and in-system tests. By delivering packetized test data to the DUT at speeds that are noticeably faster than those made possible using GPIOs, this technique shortens test times by utilizing the native protocol of these scaled high-speed interfaces. HSIO links validation uses the modified golden section direct search optimization approach published in [43] for quick and accurate jitter tolerance (JTOL) testing. The algorithm performed admirably when compared to the USB3 Gen1, SATA, and PCIe standards. Using the suggested techniques, other standards based on the receiver JTOL test, including XAUI, can be effectively adapted using the proposed methods. The work in [44] investigates and analyzes the causes of timing (jitter) noise in serial transmission lines. A complete method for estimating the frequency components of timing noise as well as simulating the behavior of timing noise sources is also presented.

The focus of the research is on jitter reduction methods. The half-rate series-parallel pseudo random binary sequence (PRBS) generator in the proposed BIST system uses a specific pattern to self-synchronize the received data stream with the reference data at the bit error checker for HSSI. To test and validate the high-speed PCI-E interface for Opengear's devices, the synchronous reconstruction approach is proposed in [35]. To define the quality of communication interfaces, a Bit Error Rate Tester (BERT) technique is proposed in [45]. The BER tester core in FPGAs and Special Additive White Gaussian Noise (AWGN) generating core used for BERT are introduced. It is more efficient to combine BERT and an AWGN on FPGAs in respect to volume, cost, and energy than current related speed stand-alone systems; it offers an advantage of significant speed over the software simulations. The spread spectrum, error-correcting codes, data recovery interfaces/native clock, and user-defined modulation are among the communication devices that can be tested and evaluated using the complete BERT technique. Two case studies are used to validate the proposed solution: an AWGN baseband transmission system is assessed by one of them, and a high-speed serial interface is evaluated by the other. BERT electronic test equipment is employed to ensure that the high-speed serial interface transceivers function properly. To enable bit error rate monitoring without the need for off-chip subsystems such as memory and the PRBS generator, a half-rate BIST system is presented in [46].

The work in [47] proposed a test and debug technique for the serial interface, i.e., JESD204B Rx PHY, allowing at-speed testing on 28 nm Silicon-On-Insulator (SOI) technology. Custom analog circuits are used for the primary BIST, while to debug BIST, conventional digital logic is used. The BIST circuitry is used to deliver a very low-cost testing mechanism with a 0.5% area overhead and a 2.5% current consumption overhead. To test the JESD Rx PHY manufacturing defects, a combination of ATPG patterns and BIST is used. The BIST focuses on at-speed transition defects as well as analog block defects in the 2.4 GHz digital domain. ATPG is implemented to overcome systematic jamming issues and defects in @60 MHz at-speed transitions. The second batch of serial pattern generators and data checkers is created on the digital boundaries to aid debugging. According to [20], the data eye margin test has become a common design-for-test (DfT) based high–speed links test technique when used in conjunction with loopback configuration. The test methodologies and DfT circuitry for backing high-speed serial interfaces are summarized in this paper (e.g., SATA). This work also provides fundamental implementation descriptions and silicon experiences as well as manufacturing test techniques for devices that employ external loopback. In addition, they provided an overview of test pattern creation, acquisition, and control implementations. The work in [48] explained how to test and classify HSIO interfaces using external measuring devices and ATE. The problems that the tester faced when working through high-speed devices were also mentioned. This work also included ATEs hardware testing-based solutions. The work in [49] has been used to build a test harness intended for a DMA controller and high-speed synchronous serial interface. The suggested test harness is employed to stimulate the device under test's various capacities as well as to assess its behavior. The test harness enables testers to analyze the behavior of peripherals integrated with the SoC. The work in [50] demonstrated a subtle method for debugging, testing, and validating high-speed serial interfaces as well as conducting functional testing. With the external loopback approach, many of the present restrictions of ATE can be alleviated. Using a BERT for FPGA and novel jitter injection method, high-speed interfaces were assessed and tested without needing the DfT capabilities and ATE instruments; this approach also overcomes traditional ATE instrument limitations. Depending on the applications, the technique intends to use ATE instruments or conducts the testing of an external loopback except for an ATE. Electronic test equipment BERT is utilized to accelerate post-silicon validation.

According to the literature review, several studies have been conducted on the high-speed serial interfaces in multiprocessor-based embedded systems. However, the majority of these studies focus on functional testing and debugging; specialized electronic equipment for testing are employed by many of the studies or to test and debug high-speed serial

interfaces through an external provider. Furthermore, there is currently a gap in the research on testing and debugging high-speed serial interfaces regarding synchronization-related issues during interprocessor communication. As a result, developing a knowledge base technique that identifies errors on single as well as multiple interfaces that can assist the developers in testing and debugging multiprocessor-based real-time embedded systems is the main objective of this study. No previous research has addressed the issues highlighted in this work to our highest knowledge.

Table 1. A benchmark table was created to contrast our results with the literature. The table summarizes our research and highlights the contributions and limits of each study.

Sr No	Research Paper and Year	Contributions	Limitation
1	Pandey et al. (2022) [42]	Tested existing high-speed functional interfaces of a System on Chip (SoC), such as PCI-E or USB, using both scan and in-system testing.	Implementation uses a limited bandwidth for testing.
2	Wacher et al. (2021) [43]	Performed jitter tolerance testing for high-speed Input/Output (HSIO) links validation using the modified golden section direct search optimization approach.	External measuring device used.
3	Tsimpos (2020) [44]	Calculation of the jitter noise frequency components and the behavioral modeling of jitter noise sources in HSSI.	Focus on jitter tolerance testing of HSSI.
4	Gabauer (2019) [35]	Use synchronous reconstruction method implemented on an FPGA to test and validate the high-speed interface, i.e., PCI-E interface.	Test only PCI-E Interface.
5	Bodha et al. (2019) [46]	Half-rate built-in self-test (BIST) system for high-speed serial interface is used to enable bit error rate measurement.	Focus on manufacturing testing.
6	Piplani et al. (2017) [47]	To test and debug high-speed serial interface i.e., JESD204B Receiver Physical Layer (JESD204B Rx PHY) using BIST and Automatic Test Pattern Generator (ATPG) patterns.	Focus on manufacturing testing.
7	Moreira and Werkmann (2016) [48]	Use of automated test equipment (ATE) and external measuring devices to characterize and test high-speed I/O digital interfaces.	External measuring devices are used.
8	Arora and Jaliminche (2015) [49]	Proposed software test harness for direct memory access (DMA) controller) and high-speed synchronous serial interface.	Use test harness for identifying manufacturing faults.
9	Masood et al. (2022)	A knowledge-based technique is proposed that has been developed to aid in detecting and fixing synchronization-related errors on multiple high-speed serial interfaces in multi-PEs-based real-time embedded systems. The proposed technique helps to identify and fix the synchronization related errors that occur during interprocessor communication among multiple high-speed serial interfaces.	

3. Synchronization Errors

One of the most critical factors influencing the complexity and performance of the real-time embedded systems is synchronization [2]. Several processing elements (PEs) are associated via various kinds of high-speed interfaces in an embedded system.

Testing and debugging these kinds of systems differ from testing and debugging non-real-time systems, making it a very challenging task. Furthermore, real-time application synchronization makes testing and debugging more complex. The implications of faults on synchronization must be addressed in critical real-time applications. Faults worsen the synchronization problem by generating worst-case scenarios such as message propagation delays and read errors.

Many circuit-level synchronization approaches have been developed, such as VLSI arrays or concurrent system models that use distributed clock lines or semaphores to provide the required synchrony. It can also occur at different levels, including logical clocks, communications, and computing processes. Preserving consistent distributed

information, guaranteeing consistent scheduling, diagnostics, coordinating functional units, reconfiguration, and application-specific choices are all aspects of synchronization. The present literature contains a wide range of system models and synchronization approaches. Both software and hardware methods have been formulated to offer the various degrees of granularity required to coordinate system services [51]. To develop new ways for detecting and locating synchronization errors in multi-PE embedded systems as well as improving interprocessor communication, embedded system testers and researchers are working consistently.

We noticed synchronization errors in embedded system real-time communication while making the embedded system, as illustrated in Table 2 and further reported in [52]. These are synchronization and timing errors rather than functional errors, and they are caused by high-speed synchronous serial interfaces. Figure 2 depicts the synchronization errors that arise during interprocessor communication in detail.

Table 2. Synchronization Errors.

Sr No	Symbol	Name
1	Rx_Delay	Receiving Delay
2	Tx_Delay	Transmitting Delay
3	D_Override	Data Override
4	Out_of_Sync	Out of Synchronization
5	No_Rx	No Receiving
6	No_Tx	No Transmission
7	Bit_Miss	Bit Miss
8	D_Read	Double Read

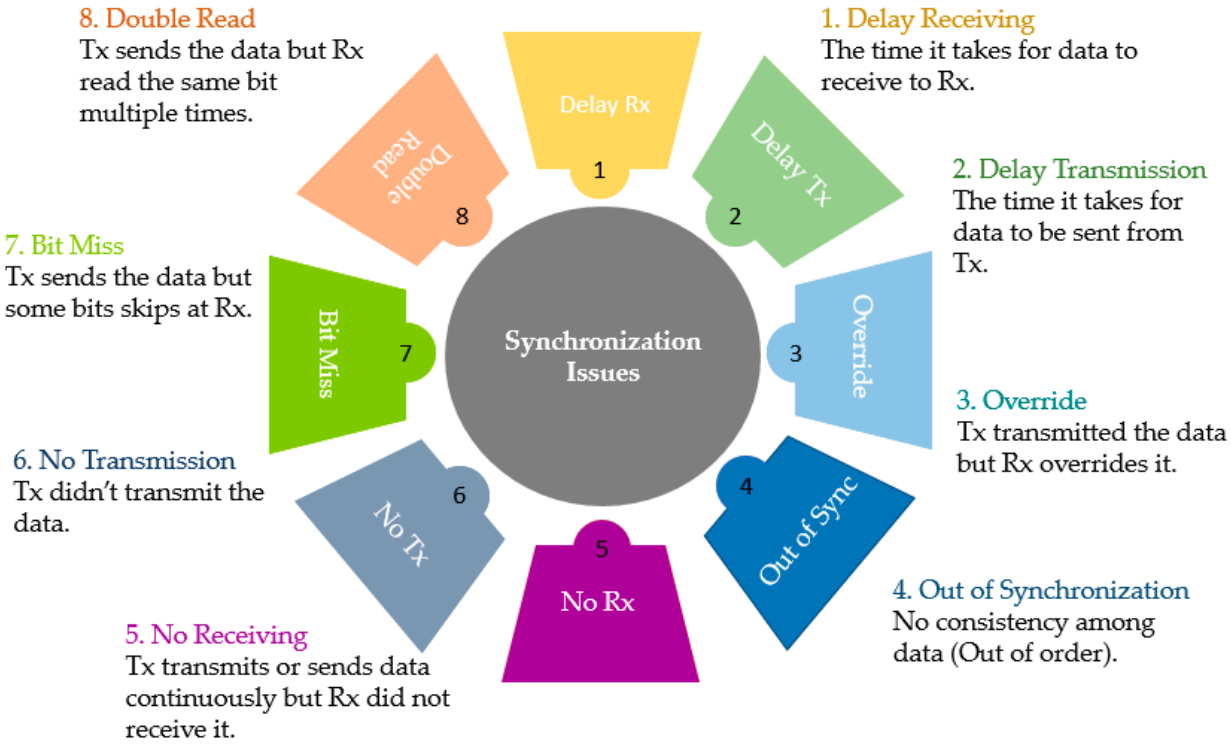


Figure 2. Detail of the synchronization errors during interprocessor communication.

4. Proposed Technique

We proposed a knowledge base technique that is utilized for testing high-speed serial interfaces in multiprocessor-based real-time embedded systems as well as finding and restraining synchronization-related issues in inter-processor communication. The proposed technique is catering to an embedded system having many PEs joined with multiple

high-speed serial interfaces. The PEs have all been functionally tested and verified for correctness. Several synchronization issues occur when these PEs communicate with one another through these interfaces in multiprocessor-based real time embedded systems. These issues might occur due to inter-PEs communication rather than the functionality of a given PE. Designers cannot be accessible at all times to monitor and observe what is happening on these interfaces.

The proposed technique used scenario-based analysis for test case generation. Scenario analysis is a popular way of building test cases from use cases. Use cases are the descriptions of the scenarios for which communication's impacts are examined. Each use case has a particular scenario. Additionally, use cases are employed as a direction for the simulation. First, use case scenarios are developed in which communications at various links occurred in a synchronization sink. At a few links, there were also synchronization issues. As a result, we were able to achieve all noticeable effects. Numerous synchronization errors were discovered in interprocessor communication. Test cases are generated using a test case generator to cover all scenarios across multiple interfaces to assess that PEs communication occurs both with and without synchronization errors using use cases in the proposed technique. The test cases are stored in a knowledge base in circumstances where there is a specific error. Numerous synchronization errors are induced in the interfaces, since these interfaces are prone to a variety of errors. The synchronization error generator will generate these errors. The simulators receive the test cases after that. In the proposed technique, across multiple PEs, the communication is simulated on multi simulators. Every simulator instance is associated with a different PE. These instances are used to develop to simulate interprocessor communication for testing and debugging the embedded system. The simulator simulates the outputs of each particular error depending on the particular test case and created synchronization errors. The simulated results are saved in the knowledge base. These errors in interprocessor communication are detected and simulated in Xilinx.

The knowledge base technique contains test cases, synchronization errors and the simulated waveform across these synchronization errors. Figure 3 depicts the knowledge base creation steps. First, we created a knowledge base that includes every conceivable and specified combination of synchronization-related errors that might arise in high-speed serial interfaces, as shown in Figure 2. Then, the proposed technique was used for localizing the synchronization errors by capturing the real-time data across multi PEs interfaces in a multiprocessor-based real-time embedded system using multiple emulators. Then, we compared the actual result of the real-time embedded system with the result stored in the knowledge base to find the correlations between them. The steps for the detection and localization of the errors are shown in Figure 4. Each emulator is connected with a different PE. So, if there are six PEs, it means that they connected with six different PCs using emulators.

This way, we will have six screens that show what is happening inside the PEs using emulators. The environmental setup of the emulator is shown in Figure 5. Emulators communicate with the PC over the universal serial bus (USB) and with processors over a joint test action group (JTAG). The emulator is used to find the outputs at each interface, as further reported in [53]. The input to the system is the ramp signal. The ideal signal for testing the interprocessor communication with a high-speed serial interfaces is a ramp (rectifier in analogy to half-wave rectification).

The input to the system is the ramp signal. The ideal signal for testing the interprocessor communication with high-speed serial interfaces is a ramp (rectifier in analogy to half-wave rectification). Ramp signals contain values that provide visible signs of several errors that may happen during interprocessor communication as well as assist in understanding the dynamic system behavior via the velocity factor. The ramp function r (n) is a function that only occurs on the positive side and is zero on the negative side [54]. It is denoted by r (n) and may be represented in equation form as shown below.

$$\mathrm{r(n)} = \begin{cases} \mathrm{n}, & \text{for } \mathrm{n} \geq 0 \\ 0, & \text{for } \mathrm{n} < 0 \end{cases} \quad (1)$$

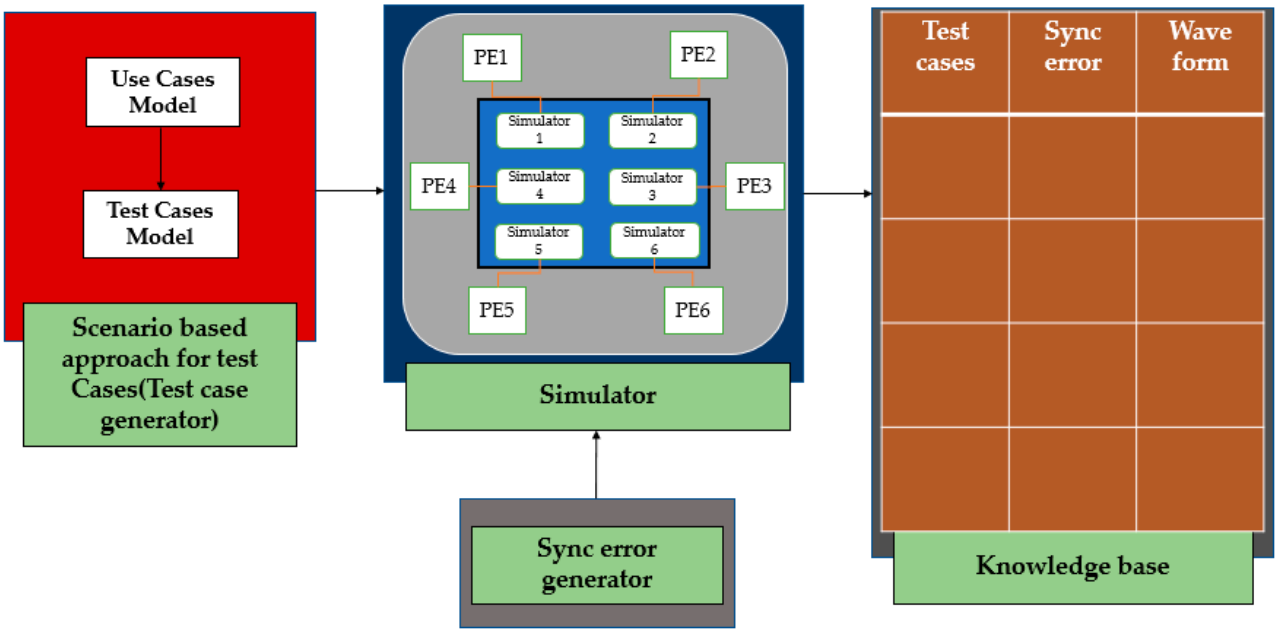


Figure 3. Creation of knowledge base technique.

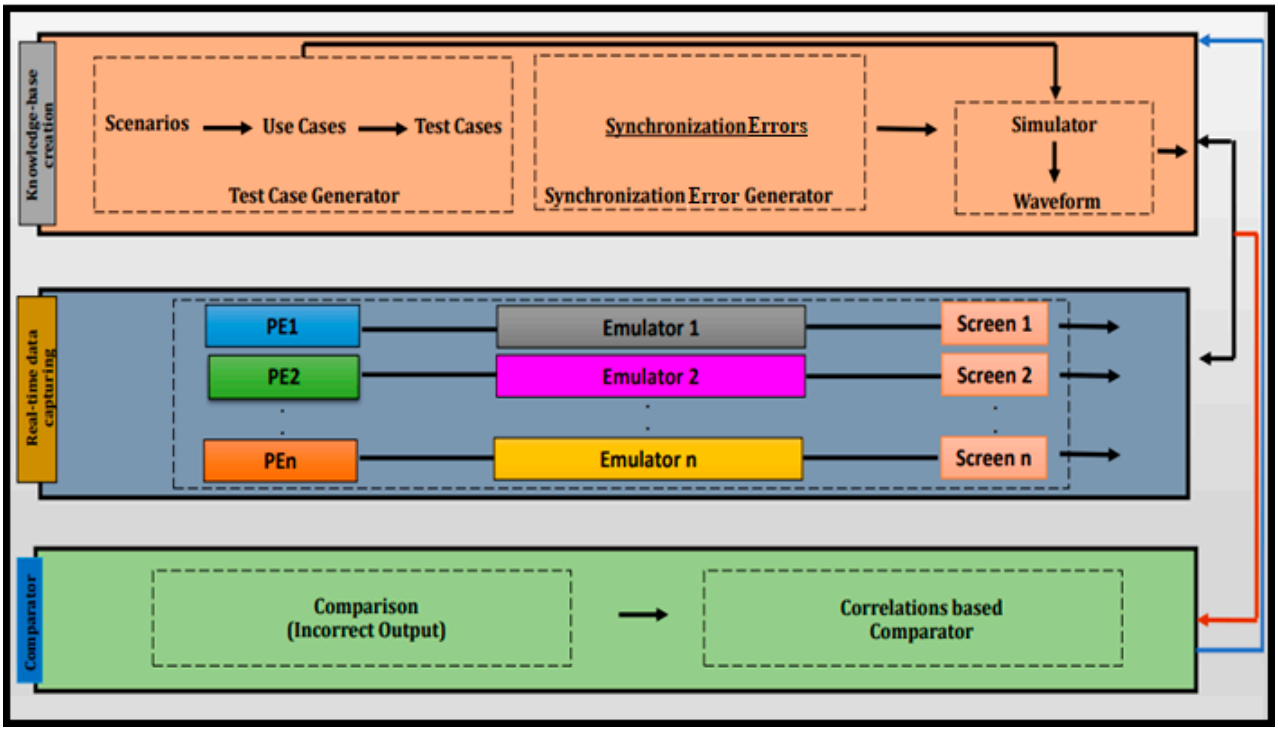


Figure 4. Steps for detection and localization of the errors.

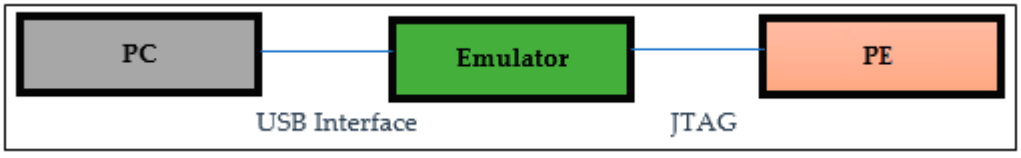


Figure 5. Environmental setup of emulator.

To test the multiple interfaces across multi PEs, we generated a ramp signal continuously and increased the ramp signal from 0 to 255 where the value of n in Equation (1) is 0 to 255. To produce all conceivable forms of the ramp, we reported in the article to observe how the shape of a ramp signal varies by inducing synchronization errors.

The ramp signal is generated continuously in these PEs and increased from 0 to 255. Once a ramp signal is given input to the interface, the ramp signal is generated as the output. Sometimes, intricacies are faced in the interfaces, due to which different waveforms are received as the output of the last processor. Some values have been missed or overlapped if the ramp signal is erroneous.

It is possible that after passing through six different interfaces, six different errors will occur, and it is also possible that the same error will occur on all six different interfaces. In consideration of the problem, we capture all the real-time data of every PE when the signal goes from one processor to another by using emulators, and their results are shown on different screens. We used a debug board in the multiprocessor embedded system environment where multi PEs are connected. Figure 6 shows the emulator environment where two emulators are connected with two PEs.

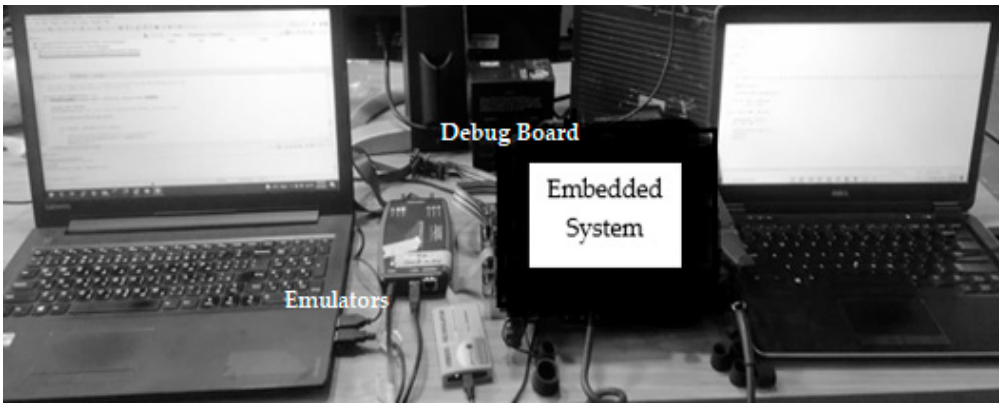


Figure 6. Emulator environment where two emulators connected with two PEs.

The waveform of the signal of simulated synchronization errors that are already recorded in the knowledge base was compared to the incoming output signal of high-speed interfaces. Then, the system's specific use case was compared with the closest correlation. In this way, the proposed technique has been validated. The outputs are also used to automatically narrow down and determine the individual interfaces that are creating errors.

5. Real-Time Embedded System under Test

The system under test consists of six PEs, i.e., advanced RISC machines (ARMs), field programmable gate arrays (FPGAs), and digital signal processors (DSPs), as shown in Figure 7. The PEs interact with each other with different types of high-speed serial interfaces. The ASP line is an audio serial port. Figure 8 shows the ASP initialization code. It is used for audio streaming as well as data transmission because the SoC is used for video processing. The ARM interface operates at a speed of 4 Mbps, while the shared memory interface operates at 10 Mbps, the UART operates at 115,200 bps, and the EMIF operates at 20 Mbps.ARMs control all the signals. UART (universal asynchronous receiver-transmitter) is used only for the control path.

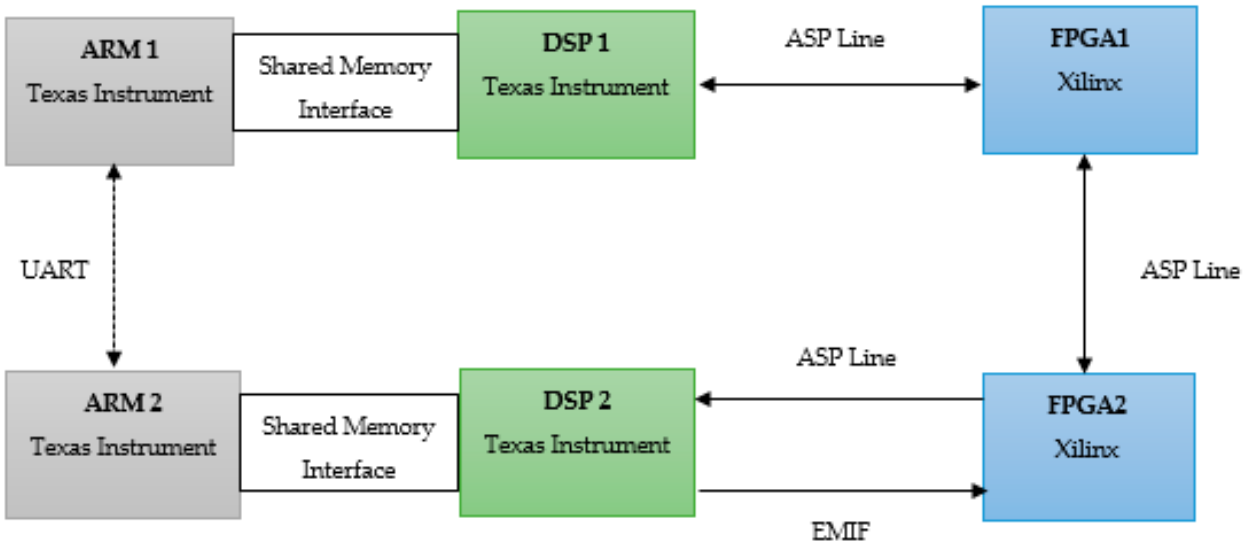


Figure 7. Real-time embedded system under test.

```
void InitializeASP()
{// Next State = Enable for the ASP Module
                MDCTL17 = 0x03;
                MDCTL2 = 0x03;
                MDCTL3 = 0x03;
                MDCTL4 = 0x03;
                // Enable State Transition
                PTCMD = 0x1;                                        // Start power state transition for ALWAYSON
                while((PTSTAT & 0x1) != 0);    // Wait for power state transition to finish
                // Enable the ASP pins on the GPIO
                PINMUX1 |= 0x00000000 |
                                        (0x1 << 10); // ASP Multiplex
                //ASP Serial Port Control Register
                SRGR =  0x00000000|
                                (0x1 << 29); // CLKSM
                SPCR =  0x00000000|
                                (0x1 << 25)| // 1 FREE
                                (0x0 << 24)| // 0 Soft mode disabled
                                (0x0 << 23)| // 0 FRST =0, as external frame sync
                                (0x0 << 22)| // 0 GRST = 0, as above
                                (0x0 << 20)| // 0 Transmit Interrupt driven by XRDY(end-of-word)
                                                                        // 2 for FS int
                                (0x0 << 19)| // 0 NA
                                (0x0 << 18)| // XEMPTY, XSR is empty, status
                                (0x0 << 16)| //
                                (0x0 << 17)| // XRDY, status
                                (0x0 << 15)| // 0, DLB disabled
                                (0x0 << 13)| // RJUST = 0,
                                (0x0 <<  4)| // 0, RINT driven by RRDY(end-of-word)
                                                                        // 2 for FS int
                                (0x0 <<  3)| // NA
                                (0x0 <<  2)| // RFULL, status
                                (0x0 <<  1)| // RRDY, status
                                (0x0 <<  0);
                //ASP Receive Control Register
                RCR = 0x00000000 |
                                (0x0   << 31)| // 0 single phase frame
                                (0x2   << 24)| // 0 words in phase 2
                                //(0x0 << 21)| // 0 word length in phase 2
                                (0x2  << 21)| // RWDLEN2
                                (0x0   << 19)| // No companding
                                (0x0   << 18)| // RFIG  = 0
                                (0x1   << 16)| // receive data delay = 1
                                (0x0   << 17)| // receive data delay = 1
 ((ASP_SLOTS-1) <<  8)| // 16 slots
                                (0x2   <<  5)| // RWDLEN1 = 16 bits
                                (0x0   <<  4); // 32 bit reversal off*/
                //ASP Transmit Control Register
                XCR = 0x00000000 |
((ASP_SLOTS-1) << 8) | // 16 slots
                                (0x2  << 5) | // RWDLEN1
                                (0x2  << 5) | // RWDLEN1
                                (0x0   << 16)| // receive data delay = 1
                                (0x0   << 17)| // receive data delay = 1
                                (0x2  << 21); // RWDLEN2
                PCR = 0x00000000 |
                                (0x1 << 7) | // Select external sclk as source SCLKME
                                (0x1 << 1); // CLKXP        }
```

Figure 8. Initialize ASP function Code.

The functionality of each processing element is tested individually, but still, it fails when interprocessor communication occurs due to high-speed serial interfaces. High-speed serial interfaces act as conveyor belts in the system. Several synchronization-related errors have been identified due to high-speed serial interfaces while developing and testing the embedded system. In debugging mode, processing elements are working normally, but they send ramps instead of data. Once the system is fully loaded, numerous synchronization issues arise due to high-speed serial interfaces. Therefore, we run the system in debugging mode and send the ramps.

Initially, a knowledge base has created with simulated waveforms where we induced synchronization errors on multiple different high-speed serial interfaces. We have created thousands of combinations of these errors in the knowledge base and then perform actual testing by capturing the real-time data using emulators. We have used Spectrum Digital XDS560-V2 (Spectrum Digital emulator, Stafford, TX, USA), Blackhawk USB200 (EWA Technologies Blackhawk emulator, Cerritos, CA, USA), and E-Elements v2-1(E-Elements technology co ltd emulator, Taipei, Taiwan) for capturing real-time data across PEs. Spectrum Digital XDS560-V2 and Blackhawk USB200 emulators are used for capturing the data in ARM 1 and ARM2. They are also used for DSP1 and DSP2. E-Elements v2-1 emulators used for FPGA1 and FPGA2.

After acquiring the actual system ramps using emulators, we compare the actual system's ramps to the stored simulated ramps of the knowledge base and compute the correlations between them. In this way, we can identify and localize the errors using the knowledge base technique. For example, the embedded system is running, and if we do not achieve the desired results, we attribute it to some type of synchronization error. A mode in the DSPs is enabled (debugging mode) where the DSPs perform their respective processing so they run in maximum load, but on the links, a ramp is sent. Every DSP receives a ramp from its connected PE and passes it forward at the time DSP is interrupted for transmission under actual load. The knowledge base created is then used to localize the error by matching the outputs of PE with the one stored in the knowledge base. Then, the comparator computes the correlations between them. Using the knowledge base technique, we would detect and locate the errors in this manner.

6. Experiments

Several experiments are conducted in detail to determine the applicability of our technique. The design architecture of the embedded system for the knowledge-based technique is developed using Verilog HDL, which was implemented with the help of the Xilinx ISE tool with four processing elements to validate the knowledge base technique. Behavioral simulations have been performed, as shown in Figure 9. The target device is xc7z010-3clg400 where the optimization goal is speed. Since it is not possible to force errors in real-time communication, we simulate the environment in Verilog where synchronization errors are introduced in the incoming signals using the Xilinx tool, and then, we plot these signals in Matlab. Generating synchronization-related errors on DSPs is not easy, because these are usually undeterministic. Therefore, the FPGA platform is only used to create all types of synchronization errors in Verilog. The simulated waveforms of the errors are then stored in the knowledge base. We induced the synchronization-related errors on multiple interfaces and then simulated those errors using Verilog and plot them in Matlab. We have used a ramp signal for simulation where we induced errors in single interfaces and then induced multiple errors on multiple interfaces.

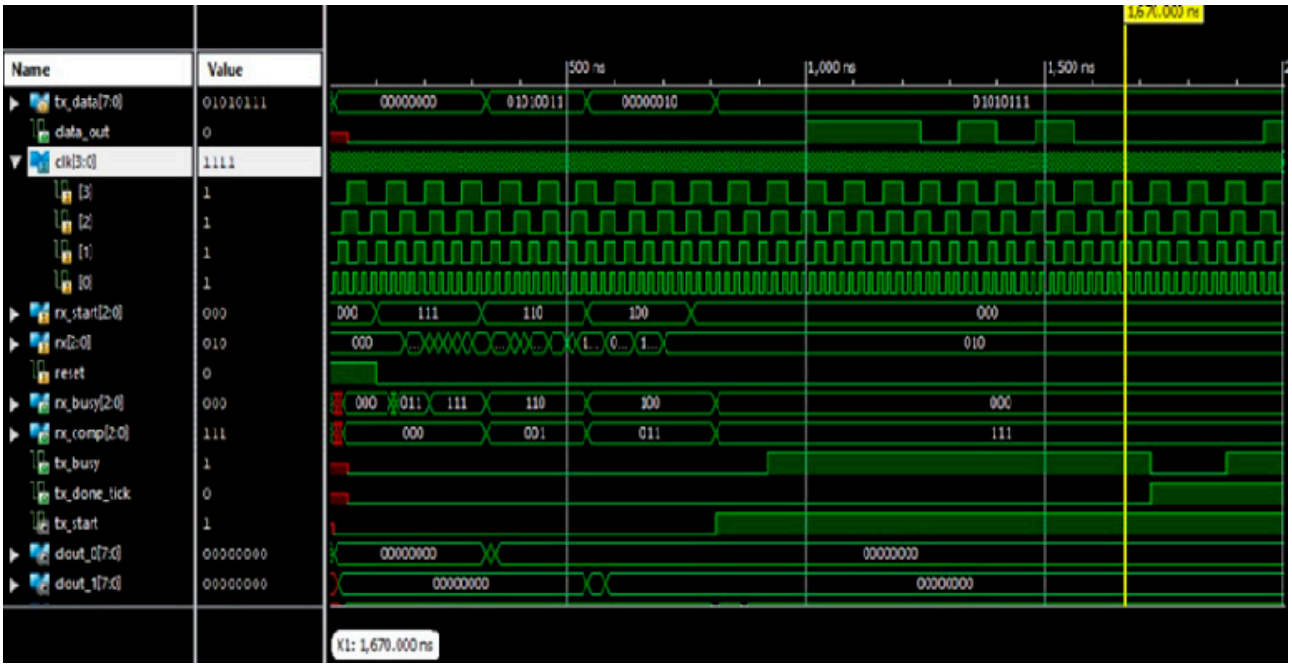


Figure 9. Verilog behavioral simulation.

Experimental Setup

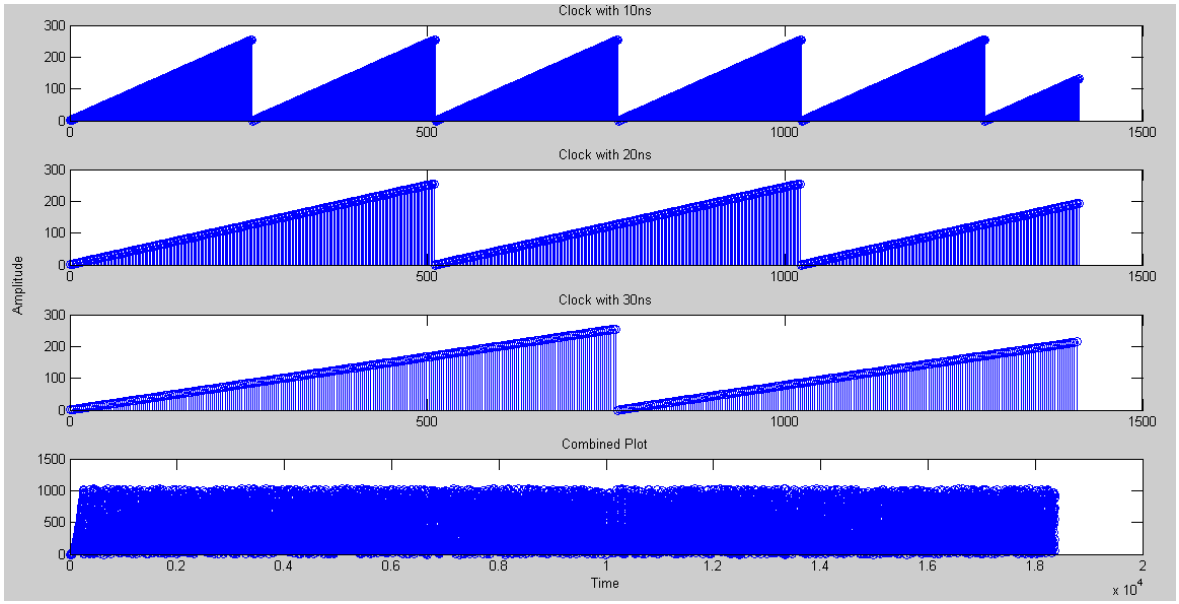
The experimental setup involves four PEs. All four PEs connected with each other and run on different clock rates in which PE1, PE2 and PE3 send the data to PE4. PE4 generates the cumulative output of all the three PEs.

Several synchronization errors have been induced on a single interface as well as the combination of the interfaces using Verilog in the experiments. Although these errors can be induced on the software, it is not possible to induce these errors on real-time embedded systems. Then, we self-simulate all potential synchronization errors on a single interface. Afterwards, the different combinations of errors are again simulated on different interfaces, and input and output are collected and saved into a knowledge base. In Figure 10, the PE1, PE2, and PE3 run on 10 ns, 20 ns, and 30 ns. PE4 generates the cumulative output of all the PEs where all the PEs are running at different clock rates. Interprocessor communication occurs without any synchronization errors on all the PEs in Figure 10. A different experiment has been conducted in this study.

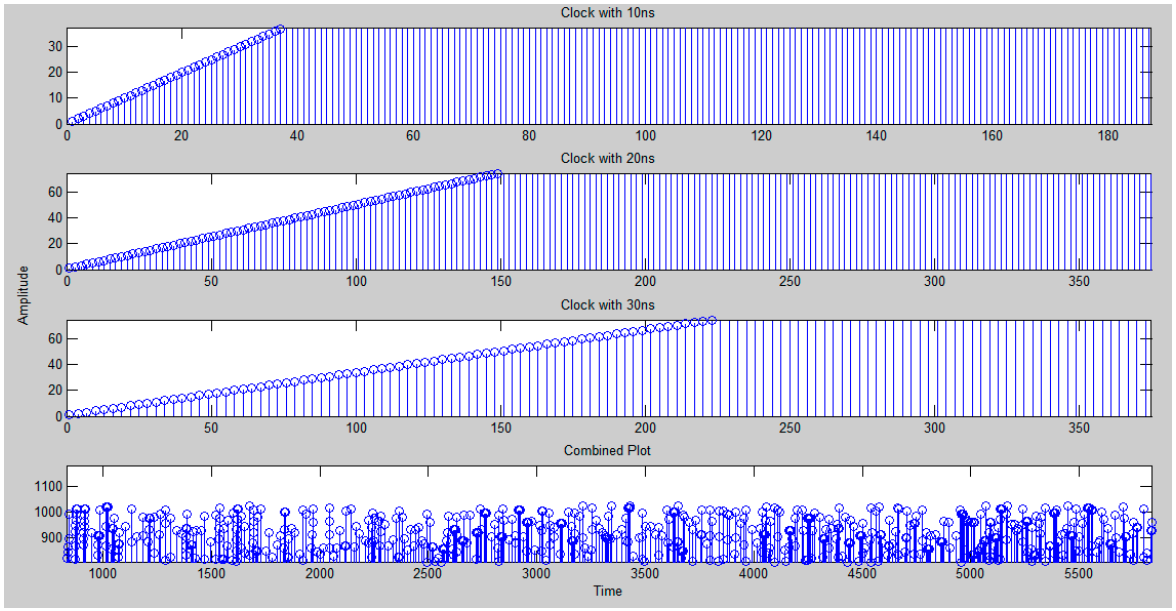
Initially, a single synchronization error is induced on a single interface. Later, two synchronization errors are induced on two different interfaces. Then, three synchronization errors are induced on three interfaces. Thus, three different combinations are generated to be analyzed, which are shown in Table 3.

Table 3. Synchronization errors on three different interfaces.

Experiment No	PE1 Clock (10 ns)	PE2 Clock (20 ns)	PE3 Clock (30 ns)	PE4
1	Delay	Delay	Delay	Combined Output
2	Delay	Bit miss	Bit miss	Combined Output
3	Delay	Bit miss	Double Read	Combined Output



(a)



(b)

Figure 10. (**a**) Simple View. (**b**) Expanded View-Normal Communication between PEs.

All simulations of these synchronization errors are stored on a knowledge base. Synchronization errors generated by the high-speed synchronous serial interfaces are simulated this way all across the system. We presume that the code is functioning and that the problems are limited to interprocessor communication.

7. Results Analysis

The experimental results of the technique that are showing the benefits in terms of performance and scalability are shown in this section. The first experiment involves four PEs in which the ramp signal is given to the input of the PEs. All PEs are communicated with each using different clock rates where delays are induced in all three PEs, i.e., PE1, PE2, and PE3, as shown in Figure 11.

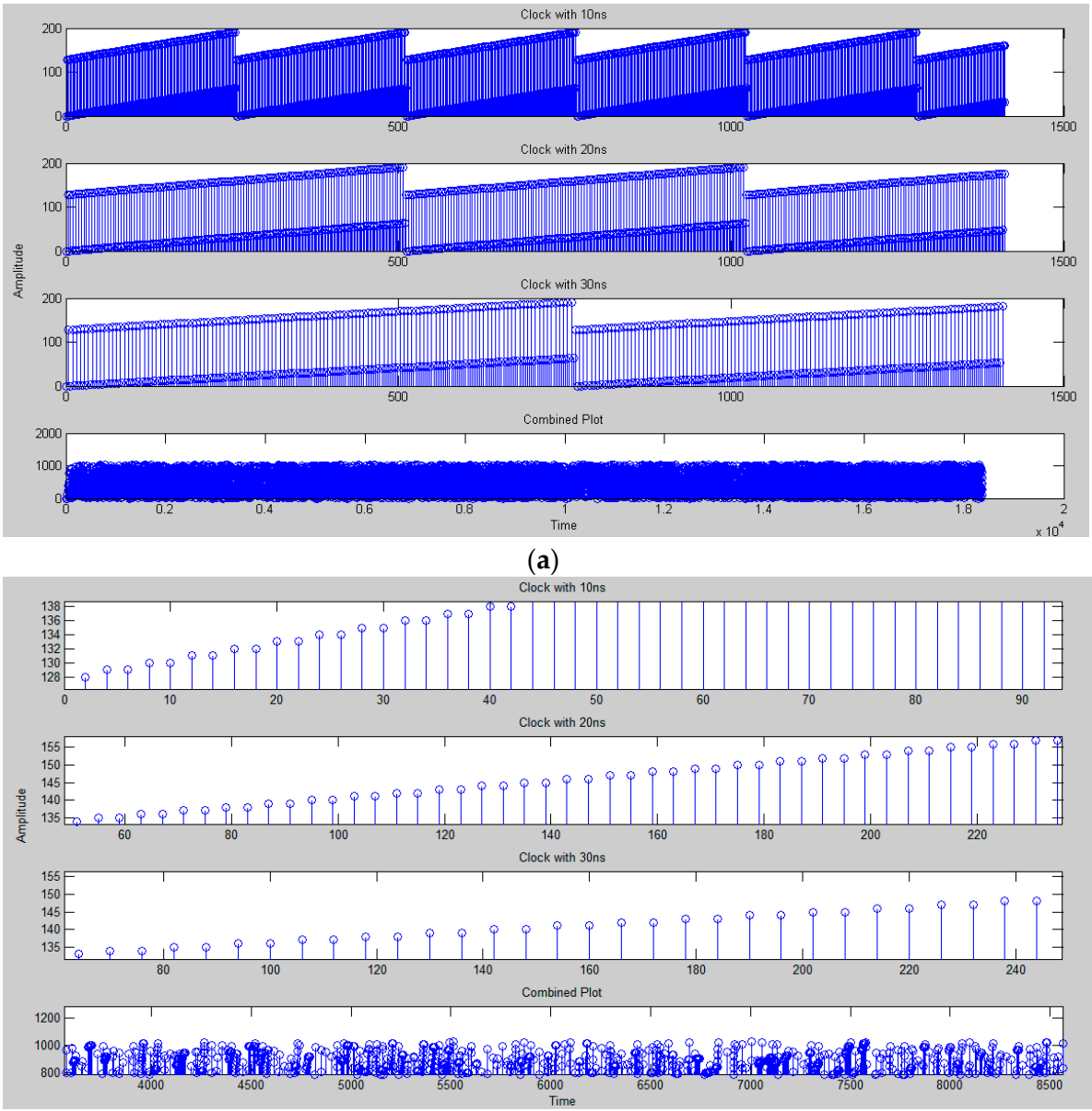
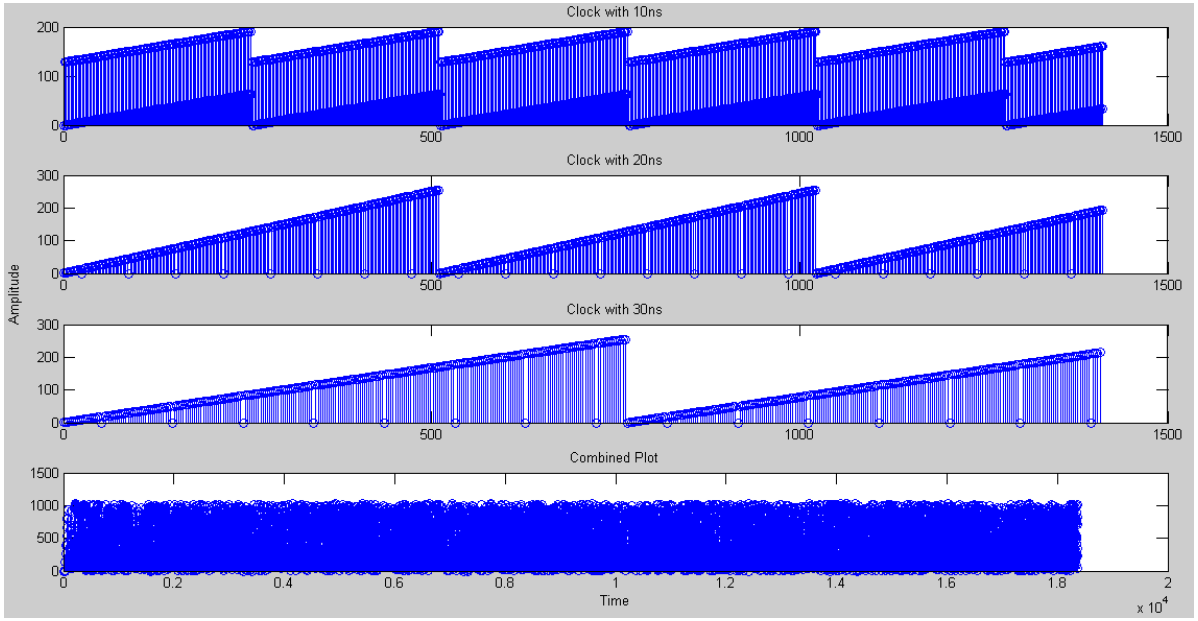


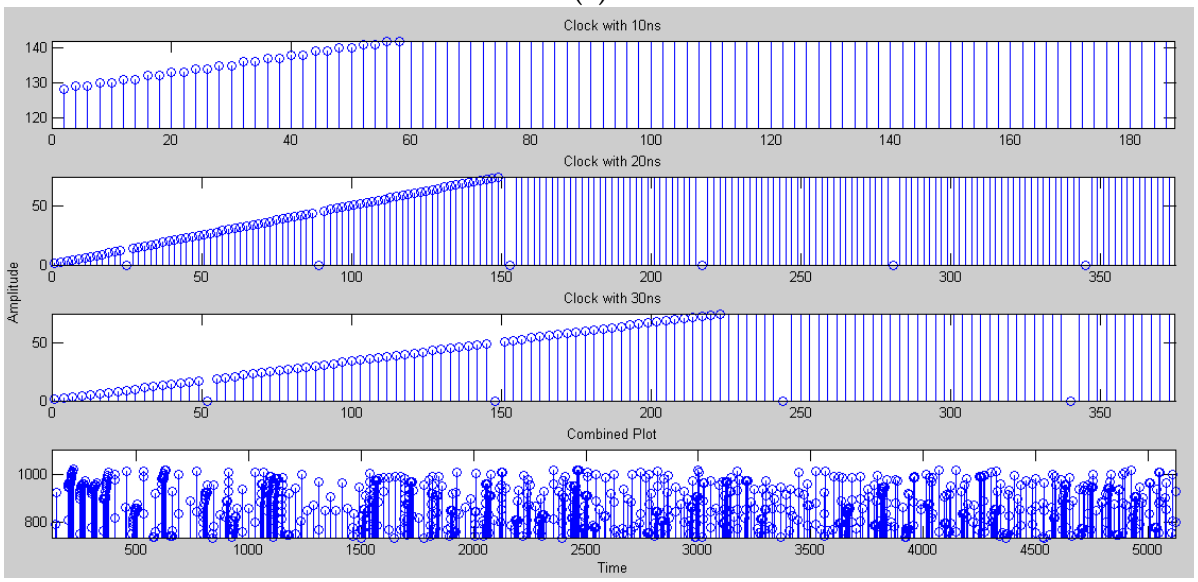
Figure 11. (**a**) Communication between PEs with delays−Simple view. (**b**) Communication between PEs with delays−Expanded view.

PE 4 generates the cumulative output of all the PEs shown in Figure 11a, which provides the simple view of the embedded system, and Figure 11b provides the expended view of where delays are induced in all three PEs.

Two synchronization errors such as delay and bit miss errors induced on the PEs in the second experiment are shown in Table 3. All PEs communicate with each other with different data rates. Figure 12a provides a simple view of the embedded system where PE1, PE2, and PE3 contained two errors. Figure 12b provides the expended view of the PEs of the embedded system where PE1 contained a delay error, PE2 and PE3 contained bit miss errors, and PE4 generated the cumulative output of all the PEs in the system.



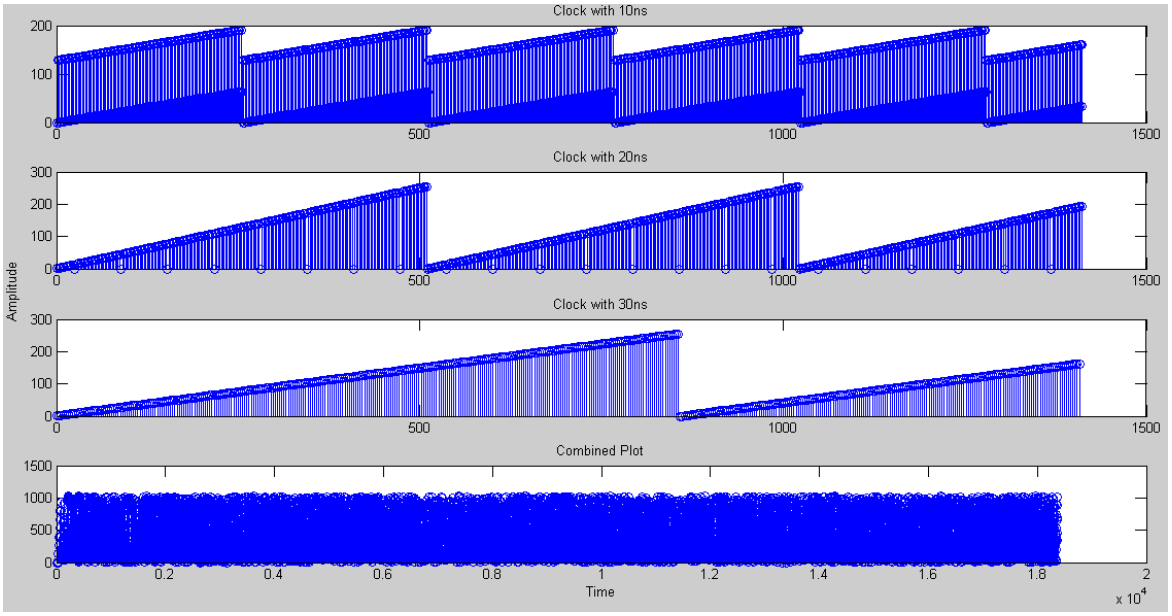
(**a**)



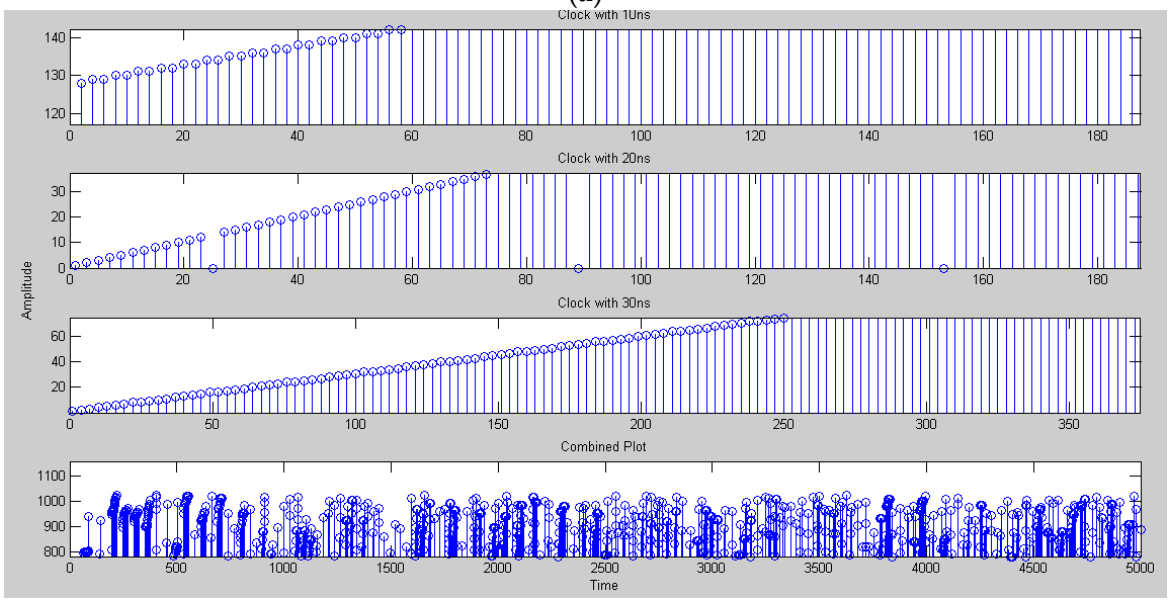
(**b**)

Figure 12. (**a**) Communication between PEs with delays and miss bit errors−Simple view. (**b**) Communication between PEs with delays and miss bit errors−Expanded view.

Thus, in the third experiment on three different PEs, three different synchronization errors are induced in the embedded system. Firstly, the delay was induced on PE1; then, a bit miss error was induced on PE2. Lastly, a double read error was induced on PE3. Figure 13a provides a simple view of the embedded system where PE1, PE2, and PE3 contained three errors on three different interfaces. Figure 13b provides the expended view of the PEs of the embedded system where PE1 contained a delay error, PE2 contained a bit miss error, PE3 contained a double read error, and PE 4 generated the cumulative output of all the PEs in the system.



(**a**)



(**b**)

Figure 13. (**a**) Communication between PEs with delays, bit miss and double read−Compressed view. (**b**) Communication between PEs with delays, bit miss and double read−Expanded view.

Every possible error was simulated by ourselves and stored in the knowledge base. If a new error comes into the real-time system, our proposed framework will simulate that error and will store it on the knowledge base. Once a knowledge base is created, the next step is to capture the real-time data from different interfaces using emulators.

Every emulator is attached to a PE with a different PC in the proposed approach. Different emulators were used that connected with the interfaces of the PE in which the data in real time were captured.

Input from one processor to the other is sent by a ramp signal. Using emulators, every PE's real-time data are captured. The comparisons are being made with the already simulated synchronization waveforms that have been stored in the knowledge base. The outputs of the emulators are compared with the stored results in the knowledge base for possible matches using correlations shown in Figures 14 and 15. This helps identify the

specific PE that is malfunctioning. When the real-time data from multiple interfaces do not reflect the actual outcomes, then correlations between the actual results and the knowledge base's stored results are examined. This allows us to identify where the synchronous error is and which interface is causing the error; then, we fix that error immediately.

Figure 14. Comparison result of P3 running on 30 ns.

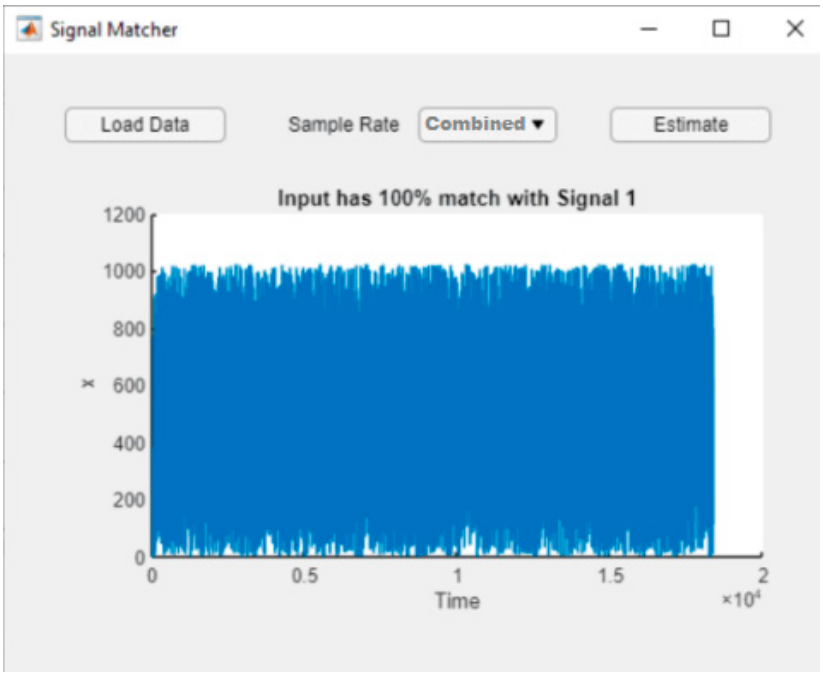


Figure 15. Comparison result of P4 (Combined output).

8. Discussion

Numerous PEs in a real-time embedded system interacted using a variety of physical interfaces. The foundation of communication today is high-speed interfaces, which are essential elements of any electrical design. Embedded to high-performance systems, on-chip

to longer distance communications employ a high-speed serial interface. The necessity for fast data processing has led to a steady increase in the data transmission speed of these interfaces. As the requirement for fast transfer rates has grown, it has become more complicated than ever to test, debug, and validate these interfaces throughout development. This can be accomplished either via the acquisition of specialist test equipment and software or by a third-party vendor. Testing is a systematic procedure for locating and reducing errors in computer software or a piece of electronic hardware so that it performs as intended. Several investigations have been made on high-speed serial interfaces in multiprocessor-based embedded systems in the literature; most of these studies were based on functional debugging and testing, and many of the studies used electronic equipment to test and debug high-speed serial interfaces. Moreover, testing and debugging approaches do functional testing and debugging, but the problem reported in this study is that the synchronization errors are caused due to interprocessor communication. These are not functionality-related errors of the PEs. All individual PEs are functionally debugged and tested. Furthermore, research into synchronization-related errors on high-speed serial interfaces during interprocessor communication is still lacking in the literature. Therefore, we present a knowledge-based technique for real-time testing of embedded systems' high-speed serial interfaces. To detect and localize errors in real-time interprocessor communication, we proposed a technique that simulates communication across several PEs. We conducted research based on interprocessor communication for this aim and were able to uncover synchronization errors as a result. The proposed system aids in the detection of possible interprocessor communication synchronization errors.

Numerous experiments are carried out to prove the technique's effectiveness by creating synchronization-related errors on interfaces. Synchronization and interprocessor communication are the most critical aspects that influence the performance of the embedded system. In this study, different kinds of synchronization errors are identified and induced on individual PEs. Then, they are induced cumulatively on several PEs in different experiments. The induced errors are simulated and stored on the knowledge base. Emulation is now most commonly used to investigate and troubleshoot the behavior of sophisticated devices with internal structures that are way too complex to be effectively described by computer simulation tools. From each PE, emulators are used to gathering real-time data. If the expected results do not reflect the actual results of PEs, a comparison with the knowledge base's results is performed. Correlations between actual and simulated results are then calculated; i.e., the incoming waveforms are compared to the waveforms of simulated synchronization errors recorded in the knowledge base. In this way, different types of synchronization errors across single as well as multiple interfaces are detected and localized. Early detection and correction of these errors lower development time and block difficulties in the line. In the proposed technique, a knowledge base including all of the simulated waveforms of possible synchronization errors has already been built; thus, testing can be performed quickly and development time can be reduced once the test case fails. According to these findings, the technique is useful to test and debug high-speed serial interfaces in multiprocessor-based embedded systems. The work's limitation is the need for more test cases, which may be developed by using various communication environments. Each new simulation environment, on the other hand, will have to be able to supplement current test cases that are continually updated to undertake more system testing and debugging.

9. Conclusions

The complexity of embedded systems has steadily arisen in recent years. Multiprocessors are being widely used in embedded devices these days. In multiprocessor systems, the linking architecture connects the processors and allows data to flow between them. One of the major issues with embedded systems that handle real-time processing applications is interprocessor communication and synchronization. Multiple PEs in an embedded system must be synchronized in order to interact with one another via a variety of high-speed

serial interfaces. Developers typically spend a substantial amount of time testing and correcting errors in real-time embedded system programs. In this paper, we presented a knowledge base technique for zeroing errors on multiple high-speed serial interfaces in a multiprocessor-based real time embedded system. The proposed technique aids in the testing of high-speed serial interfaces in multiprocessor-based real-time embedded systems that require debugging in real time, whereas an application is running because several factors influence interprocessor communication. The work's drawback, as in other circumstances like it, is the requirement for more test cases, which might be produced by using real-time constraints in different communication contexts. Experimental results have been carried out running on a multiprocessor-based embedded system. The results show that the proposed framework is capable of identifying and localizing errors that are usually hard to recognize as well as helping to reduce design time.

Author Contributions: Conceptualization, S.M. and S.A.K.; Data curation, S.M., A.H. and F.K.; Investigation, S.M., A.H. and F.K.; Methodology, S.M., S.A.K., A.H. and F.K.; Project administration, S.A.K.; Resources, F.K.; Software, S.M. and A.H.; Supervision, S.A.K. and A.H.; Validation, S.M. and S.A.K.; Visualization, S.M. and F.K.; Writing—original draft, S.M.; Writing—review & editing, S.A.K. and A.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cassano, L.; Cozzi, D.; Jungewelter, D.; Korf, S.; Hagemeyer, J.; Porrmann, M.; Bernardeschi, C. An inter-processor communication interface for data-flow centric heterogeneous embedded multiprocessor systems. In Proceedings of the 2014 9th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS), Santorini, Greece, 6–8 May 2014. [CrossRef]
2. Chung, M.K.; Shim, H.; Kyung, C.M. Performance improvement of multiprocessor simulation by optimizing synchronization and communication. In Proceedings of the 16th IEEE International Workshop on Rapid System Prototyping (RSP'05), Montreal, QC, Canada, 8–10 June 2005. [CrossRef]
3. Maruf, M.A.; Azim, A. Requirements-preserving design automation for multiprocessor embedded system applications. *J. Ambient. Intell. Humaniz. Comput.* **2021**, *12*, 821–833. [CrossRef]
4. Moudgill, M.; Kalashnikov, V.; Senthilvelan, M.; Srikantiah, U.; Li, T.; Balzola, P.; Glossner, J. Synchronization on heterogeneous multiprocessor systems. In Proceedings of the International Symposium on Systems, Architectures, Modeling, and Simulation, Samos, Greece, 20–23 July 2009. [CrossRef]
5. Rahman, M.M. Process synchronization in multiprocessor and multi-core processor. In Proceedings of the International Conference on Informatics, Electronics & Vision (ICIEV), Dhaka, Bangladesh, 18–19 May 2012. [CrossRef]
6. Chen, Y.; Yang, Y.; Wang, F.; Kai, G. Inter Multi processor communication scheme and shared memory control in the HDTV decoder SOC design. In Proceedings of the 2005 IEEE International Workshop on VLSI Design and Video Technology, Suzhou, China, 28–30 May 2005. [CrossRef]
7. Pinheiro, E.M.; Correia, S.D. Software Model for a Low-Cost, IoT oriented Energy Monitoring Platform. *Int. J. Comput. Sci. Eng.* **2018**, *5*, 1–5. [CrossRef]
8. Dezan, C.; Zermani, S.; Hireche, C. Embedded Bayesian Network Contribution for a Safe Mission Planning of Autonomous Vehicles. *Algorithms* **2020**, *13*, 155. [CrossRef]
9. Baras, N.; Nantzios, G.; Ziouzios, D.; Dasygenis, M. Autonomous Obstacle Avoidance Vehicle Using LIDAR and an Embedded System. In Proceedings of the 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, Greece, 13–15 May 2019; pp. 1–4.
10. Qiang, Z.; Wang, H.; Ning, W.; Song, X.; Sha, M.; Kang, Z.; Sun, X. Application of medical embedded system and clinical nursing effect of neonatal intestinal bacteria. *Microprocess. Microsyst.* **2021**, *83*, 103981. [CrossRef]
11. Garcia, J.; Shannon, R.; Jacobson, A.; Mosca, W.; Maldonado, R.; Burger, M. Powerful authentication regime applicable to naval OFP integrated development (PARANOID): A vision for non-circumventable code signing and traceability for embedded avionics software. *J. Def. Anal. Logist.* **2021**, *5*, 46–76. [CrossRef]
12. Akesson, B.; Nasri, M.; Nelissen, G.; Altmeyer, S.; Davis, R.A. An Empirical Survey-based Study into Industry Practice in Real-time Systems. In Proceedings of the Real-Time Systems Symposium, Houston, TX, USA, 1–4 December 2020. [CrossRef]
13. Shee, S.L.; Parameswaran, S. Design methodology for pipelined heterogeneous multiprocessor system. In Proceedings of the 44th ACM/IEEE Design Automation Conference, San Diego, CA, USA, 4–8 June 2007. [CrossRef]
14. Senouci, B.; Kouadri, M.A.M.; Rousseau, F.; Petrot, F. Multi-CPU/FPGA platform based heterogeneous multiprocessor prototyping: New challenges for embedded software designers. In Proceedings of the 19th IEEE/IFIP International Symposium on Rapid System Prototyping, Monterey, CA, USA, 2–5 June 2008. [CrossRef]

15. Xiao, H.; Isshiki, T.; Li, D.; Kunieda, H.; Nakase, Y.; Kimura, S. Optimized communication and synchronization for embedded multiprocessors using ASIP methodology. *IPSJ Trans. Syst. LSI Des. Methodol.* **2012**, *5*, 118–132. [CrossRef]
16. Chen, C.; Du, G.; Zhang, D.; Song, Y.; Hou, N. Communication synchronous scheme for MPSoC. In Proceedings of the International Conference on Anti-Counterfeiting, Security and Identification, Chengdu, China, 18–20 July 2010. [CrossRef]
17. Tullsen, D.M.; Lo, J.L.; Eggers, S.J.; Levi, H.M. Supporting fine-grained synchronization on a simultaneous multithreading processor. In Proceedings of the Fifth International Symposium on High-Performance Computer Architecture, Orlando, FL, USA, 9–13 January 1999.
18. Brunel, J.Y.; Kruijtzer, W.M.; Kenter, H.J.H.N.; Petrot, F.; Pasquier, L.; De Knok, E.A.; Smits, W.J.M. COSY communication IP's. In Proceedings of the 37th Design Automation Conference, Los Angeles, CA, USA, 5–9 June 2000. [CrossRef]
19. Kim, H.; Abraham, J.A. On-chip source synchronous interface timing test scheme with calibration. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 12–16 March 2012. [CrossRef]
20. Meixner, A.; Kakizawa, A.; Provost, B.; Bedwani, S. External loopback testing experiences with high speed serial interfaces. In Proceedings of the IEEE International Test Conference, Santa Clara, CA, USA, 28–30 October 2008. [CrossRef]
21. Arora, S.; Aflaki, A.; Biswas, S.; Shimanouchi, M. SERDES external loopback test using production parametric-Test hardware. In Proceedings of the International Test Conference, Fort Worth, TX, USA, 15–17 November 2016. [CrossRef]
22. Fan, Y.; Zilic, Z. *Accelerating Test, Validation and Debug of High Speed Serial Interfaces*; Springer: Dordrecht, The Netherlands, 2011. [CrossRef]
23. Junior, J.C.V.S.; Brito, A.V.; Nascimento, T.P. Testing real-time embedded systems with hardware-in-the-loop simulation using high level architecture. In Proceedings of the Brazilian Symposium on Computing Systems Engineering (SBESC), Foz do Iguacu, Brazil, 3–6 November 2015. [CrossRef]
24. Hopkins, A.B.T.; McDonald-Maier, K.D. Debug support for embedded processor reuse. In Proceedings of the 2006 IEEE International Symposium on Circuits and Systems, Kos, Greece, 21–24 May 2006. [CrossRef]
25. Parnin, C.; Orso, A. Are automated debugging techniques actually helping programmers? In Proceedings of the International Symposium on Software Testing and Analysis, Toronto, ON, Canada, 17–21 July 2011. [CrossRef]
26. Bagherzadeh, M.; Hili, N.; Dingel, J. Model-level, platform-independent debugging in the context of the model-driven development of real-time systems. In Proceedings of the 11th Joint Meeting on Foundations of Software Engineering, Paderborn, Germany, 4–8 September 2017. [CrossRef]
27. Pouget, K. Programming-Model Centric Debugging for Multicore Embedded Systems. Embedded Systems. Université de Grenoble, 2014. NNT: 2014GRENM008. Available online: https://tel.archives-ouvertes.fr/tel-01548327/document (accessed on 12 September 2022).
28. Bergeron, J. *Writing Testbenches: Functional Verification of HDL Models*, 2nd ed.; Kluwer Academic Publishers: Norwell, MA, USA, 2003.
29. Bergeron, J. *Writing Testbenches Using System Verilog*; Springer: Berlin/Heidelberg, Germany, 2006.
30. Jeannet, B.; Gaucher, F. Debugging embedded systems requirements with stimulus: An automotive case-study. In Proceedings of the 8th European Congress on Embedded Real Time Software and Systems, Toulouse, France, 27–29 January 2016; Available online: https://hal.archives-ouvertes.fr/hal-01292286 (accessed on 12 September 2022).
31. Song, J.; Zhang, Q.; Zhou, L.; Quan, Z.; Wang, S.; Liu, Z.; Fang, X.; Wu, Y.; Yang, Q.; Yin, H.; et al. Design and Implementation of a Modular and Scalable Research Platform for Ultrasound Computed Tomography. *IEEE Trans. Ultrason. Ferroelectr. Freq. Control* **2022**, *69*, 62–72. [CrossRef] [PubMed]
32. Bandiziol, A.; Grollitsch, W.; Brandonisio, F.; Nonis, R.; Palestri, P. Design of a transmitter for high-speed serial interfaces in automotive micro-controller. In Proceedings of the 39th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2016-Proceedings, Opatija, Croatia, 30 May–3 June 2016.
33. Mohammadi, R.; Ndiritu, S. *Software Defined Radio: High Performance, Flexible Technology for Spectrum Monitoring*; Per Vices: Toronto, ON, Canada, 2021.
34. Ying, W.; Jie, W. Radar Broadband Signal High-precision On-line Testing Method. In Proceedings of the 2021 Global Reliability and Prognostics and Health Management (PHM-Nanjing), Nanjing, China, 15–17 October 2021; pp. 1–6. [CrossRef]
35. Gabauer, J. *Test and Validation of the Integrity and Performance of High Speed Interfaces*; University of Queensland: St Lucia, Australia, 2019.
36. Patra, P. On the cusp of a validation wall. *IEEE Des. Test Comput.* **2007**, *24*, 193–196. [CrossRef]
37. Vermeulen, B. Functional Debug Techniques for Embedded Systems. *IEEE Des. Test Comput.* **2008**, *25*, 208–215. [CrossRef]
38. Fan, Y.; Cai, Y.; Zilic, Z. A high accuracy high throughput jitter test solution on ATE for 3GBPS and 6GBPS serial-ata. In Proceedings of the IEEE International Test Conference, Santa Clara, CA, USA, 21–26 October 2007. [CrossRef]
39. Hong, D.; Cheng, K.T. *Efficient Test Methodologies for High-Speed Serial Links*; Lecture Notes in Electrical Engineering; Springer: Dordrecht, The Netherlands, 2010; Volume 51. [CrossRef]
40. Association, S.I. International Technology Roadmap for Semiconductors, 2013 edition. Available online: http://www.itrs2.net/2013-itrs.html (accessed on 29 December 2021).
41. Kandalaft, N.; Attaran, A.; Rashizadeh, R. High speed test interface module using MEMS technology. *Microelectronics. Reliab.* **2015**, *55*, 374–382. [CrossRef]

42. Pandey, A.; Tully, B.; Samudra, A.; Nagarandal, A.; Natarajan, K.; Singhal, R. Novel Technique for Manufacturing & In-system Testing of Large Scale SoC using Functional Protocol Based High-Speed I/O. In Proceedings of the 2022 IEEE 40th VLSI Test Symposium (VTS), San Diego, CA, USA, 25–27 April 2022.
43. Wacher, A.V.; Baylon, R.B.; Rangel-Patino, F.E.; Silva-Cortes, J.L.; Vega-Ochoa, E.A.; Rayas-Sánchez, J.E. Fast Jitter Tolerance Testing for High-Speed Serial Links in Post-Silicon Validation. *IEEE Trans. Electromagn. Compat.* **2021**, *64*, 516–523. [CrossRef]
44. Tsimpos, A. Multi-Data Rate Receiver for High Speed Serial Interfaces. Ph.D. Thesis, Electronics and Computers Section, University of Patras, Patras, Greece, 2020.
45. Fan, Y.; Zilic, Z. BER testing of communication interfaces. *IEEE Trans. Instrum. Meas.* **2008**, *57*, 897–906. [CrossRef]
46. Bodha, R.R.R.; Sarafi, S.; Kale, A.; Koberle, M.; Sturm, J. A Half-Rate Built-In Self-Test for High-Speed Serial Interface using a PRBS Generator and Checker. In Proceedings of the Austrochip Workshop on Microelectronics (Austrochip), Vienna, Austria, 24 October 2019.
47. Piplani, S.; Fonseca, H.; Sharma, V.M.; Cervini, D.; Hardisty, D. Test and debug strategy for high speed JESD204B Rx PHY. In Proceedings of the IEEE 26th Asian Test Symposium (ATS), Taipei, Taiwan, 27–30 November 2017. [CrossRef]
48. Moreira, J.; Werkmann, H. *An Engineer's Guide to Automated Testing of High-Speed Interfaces*, 2nd ed.; Artech: Boston, MA, USA, 2016; Volume 2.
49. Arora., H.; Jaliminche, L.N. Design and implementation of test harness for device drivers in SOC on mobile platforms. In Proceedings of the International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI-SATA), Bengaluru, India, 8–10 January 2015. [CrossRef]
50. Fan, Y.; Zilic, Z. A versatile scheme for the validation, testing and debugging of high speed serial interfaces. In Proceedings of the IEEE International High Level Design Validation and Test Workshop, San Francisco, CA, USA, 4–6 November 2009. [CrossRef]
51. Suri, N.; Hugue, M.M.; Walter, C.J. Synchronization Issues in Real Time Systems. *Proc. IEEE* **1994**, *82*, 41–54. [CrossRef]
52. Masood, S.; Khan, S.A.; Hassan, A. Simulating synchronization issues on a multiprocessor embedded system for testing. In Proceedings of the IEEE International Conference on Information Communication and Software Engineering, Chengdu, China, 19–21 March 2021. [CrossRef]
53. Masood, S.; Khan, S.A.; Hassan, A.; Fatima, U. A novel framework for testing high-speed serial interfaces in multiprocessor based real-time embedded system. *Appl. Sci.* **2021**, *11*, 7465. [CrossRef]
54. Singh, S. Proposed Concept of Signals for Ramp Functions. In Proceedings of the World Congress on Engineering and Computer Science 2010 Vol I WCECS 2010, San Francisco, CA, USA, 20–22 October 2010.