

## Simulation-based fault-tolerant multiprocessors system

Ahmad F. Al-Allaf, Ziyad Khalaf Farej

Department of Computer Technology Engineering, Engineering Technical College, Northern Technical University, Mosul, Iraq

---

### Article Info

#### Article history:

Received Oct 22, 2021

Revised Jun 09, 2022

Accepted Nov 12, 2022

---

#### Keywords:

Fault-tolerant  
Multiprocessor  
Pipelined  
SimEvents  
Stateflow

---

### ABSTRACT

System reliability is an important issue in designing modern multiprocessor systems. This paper proposes a fault-tolerant, scalable, multiprocessor system architecture that adopts a pipeline scheme. To verify the performance of the proposed system, the SimEvent/Stateflow tool of the MATLAB program was used to simulate the system. The proposed system uses twelve processors (P), connected in a linear array, to build a ten-stage system with two backup processors (BP). However, the system can be expanded by adding more processors to increase pipeline stages and performance, and more backup processors to increase system reliability. The system can automatically reorganize itself in the event of a failure of one or two processors and execution continues without interruption. Each processor communicates with its neighboring processors through input/output (I/O) ports which are used as bypass links between the processors. In the event of a processor failure, the function of the faulty processor is assigned to the next processor that is free from faults. The fast Fourier transform (FFT) algorithm is implemented on the simulated circuit to evaluate the performance of the proposed system. The results showed that the system can continue to execute even if one or two processors fail without a noticeable decrease in performance.

*This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.*



---

### Corresponding Author:

Ahmad F. Al-Allaf

Department of Computer Technology Engineering, Engineering Technical College

Northern Technical University, Mosul, Iraq

Email: Ahmed.faleh@ntu.edu.iq

---

## 1. INTRODUCTION

Multiprocessor systems are used to provide a high degree of parallelism in a variety of applications. The efficiency of these systems is affected by unforeseen events such as aging that can cause permanent malfunctions in some parts. Therefore, very stringent reliability requirements made the provision of fault tolerance an important aspect of the design of these systems [1]-[3].

Two types of defects occur in multiprocessor systems: permanent defects and transient defects. Permanent breakdowns can result from material damage or limited operational life. An effective mechanism must be used to isolate the faulty part and reconfigure it. The treatment system to maintain the level of performance. Permanent and transient fault detection can be done by making a connection attempt and declaring the fault as permanent if there is no response after a specified number of attempts. For transient faults that may result due to overheating, overloading, or otherwise, these can be addressed programmatically but may cause some delay [4].

There are many approaches to hardware fault-tolerant designs in multiprocessor systems [5] and they differ in the basic concepts of fault-tolerant design and various problems that the designer must take care of. Another side, one of the goals of multiprocessor system designs is scalability [6]. It is preferable to adapt the size of the parallel processor system to the size of the tasks to be addressed.

Fault tolerance means avoiding system failure in existence malfunctions in some of its parts. It can also be considered that a system tolerates faults if the presence of faults does not affect the output of that system. A fault-tolerant system should have a mechanism to detect faults and recover from them. Figure 1 shows the steps to be taken to obtain a fault-tolerant system [7].

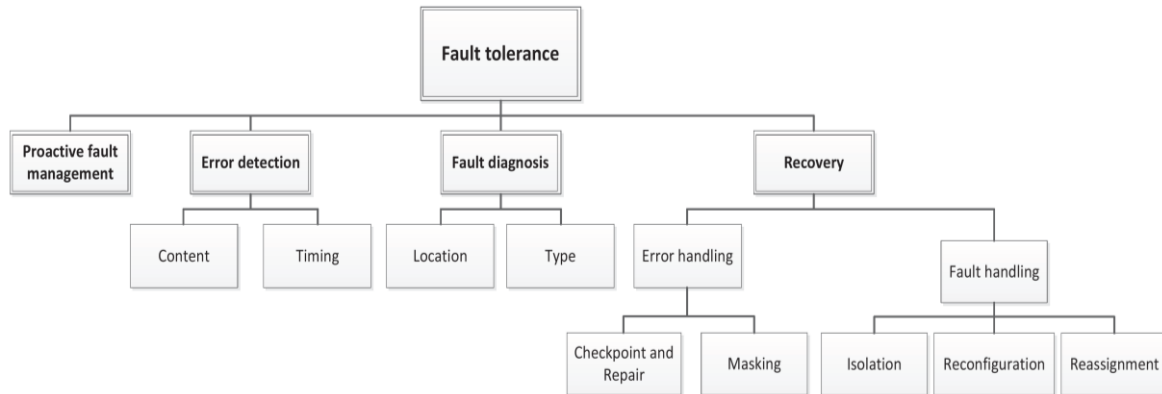


Figure 1. The steps used to Fault-tolerance

As shown in Figure 1, in a fault-tolerant, system recovery can be either by error handling or fault handling, or both. Error handling means eliminating errors from the system without removing the source of the error. While fault handling is the process of removing the source of the fault. Error handling is used with transient faults, as it is not necessary to determine the source of error.

Processor failure can occur on a uniprocessor or multiprocessor system uniprocessor system, no measures can be taken to recover the system from the fault. But in the case of multiple processes, the execution of time-critical tasks can be guaranteed by achieving hardware redundancy, that is, using backup units that can perform the tasks of a faulty processor. The process of performing reliability tests on computer architecture and true multiprocessor systems is complicated by timing and energy restrictions. Therefore, many studies use simulation as a tool to perform reliability studies on these systems [8]-[10].

Most modern multiprocessor systems are often so complex that the validation of the design concept systems is only possible through modeling techniques such as simulations. Many analytical solution methods require complex models to conform to certain standards, their solutions may require a highly capable computer system. While the simulation does not contain such complications. This is the main reason for using simulation as a popular solution method despite the development of new analytical methods.

One of the most popular discrete event simulation programs used in research, is the SimEvents/Stateflow program provided by the MATLAB platform [11]. This emulator is capable of configuring multiple processor systems with a variety of implementation policies. In this paper, a discrete event simulation based on MATLAB's SimEvents/Stateflow was used to provide a robust and scalable, fault-tolerant, multiprocessor model for evaluation purposes. The system can reconfigure itself dynamically when a failure occurs in one or more of its processors. The proposed system is an expandable multiprocessor system that relies on a pipeline architecture based on input/output (I/O) ports for interprocessor communications. This architecture (processors connected in a pipeline) is well suited to the streaming signal processing and data flow nature of multimedia applications [12], [13]. The main design goal is to simple and effective inter-processor communication model with the possibility of unlimited expansion along with fault tolerance. The proposed system does not require a separate algorithm to detect faulty processors. It can detect the fault implicitly during message transfer protocol, that reduces the extra time caused by the fault detection algorithms.

## 2. LITERATURE REVIEW

Explaining simulation-based approaches for designing and verifying multiprocessor systems have been proposed in [8], [14]-[20]. In the field of a fault-tolerant multiprocessor systems, Yuan *et al.* [21] proposes an evolutionary algorithm (EA)-based design space exploration (DSE) method for the design of fault-tolerant multiprocessor systems. The main goals of this paper include task mapping and design space of task hardening. However, the limitations of this work are considering only replicating a task or re-executing a task.

Aliee and Zarandi [22] present a dynamically scheduled pipeline architecture for chip multiprocessors (CMPs). Their technique exploits existing simultaneous multithreading, superscalar chip

multiprocessors' redundancy to provide low-overhead, and broad coverage of faults. However, these benefits come at the cost of performance degradation for processors. The goal of the work is to enhance the reliability of the system without adding extra redundancy for fault tolerance, to detect both transient and permanent faults, and to recover existing faults. Baek and Lee [23] propose a fault-tolerant scheme that can be incorporated with real-time scheduling algorithms on multiprocessor systems to improve the reliability of the target system without a tradeoff against schedulability. They applied this fault-tolerant scheme to existing fixed-priority scheduling and earliest deadline zero-laxity scheduling, and they claimed that it enhances reliability without schedulability loss.

A fault recovery mechanism named fault-tolerant fair scheduler (FT-FS), has been proposed by Nair *et al.* [24] for real-time multiprocessor systems scheduled using the proportional fair scheduling technique. The proposed scheduler tries to minimize rejections of critical tasks, during transient overloads within recovery intervals using two features namely, weight donation and post rejection backtracking. The objective is to maximize the possibility of keeping the system fail-operational even in the occurrence of faults.

Al-Allaf and Nayif [25] proposed architecture of a multiprocessing system that uses TMS320C40 digital signal processors. The architecture uses hypercube topology and redundancy to address faults in the system. Field programmable gate array (FPGA) circuits were used to diagnose faults, reconfigure, and extend the system. Xilinx foundation F2.1i simulator was used to simulate communications between processors and the fault detection mechanism.

A real-time multiprocessor system model has been proposed in [26] that tolerates faults using redundancy and can handle the extra workload at no additional cost. The paper presents the design of a general simulator called FaRRReTSim, to perform intensive real-time model simulation studies under fault and fault-free conditions. On another side, advances in technology have allowed multiple processing cores to be executed on the same chip. Many researchers have taken to fault tolerance (FT) techniques to increase reliability and reduce the likelihood of these systems failing [27]-[30].

### 3. THE PROPOSED ARCHITECTURE AND INTERPROCESSOR COMMUNICATION SCHEME

The proposed system topology consists of tightly coupled processing nodes (P) arranged in a pipeline architecture. A node can communicate with its next neighbors through 8-bit I/O ports. These connections are organized so that each processor can communicate directly with three processors of the next or preceding it, as shown in Figure 2. This arrangement can keep the pipeline operational even in the occurrence of failure of a maximum of two successive processors or multiple non-contiguous processors depending on the number of backup processors available. The architecture can be expanded by adding more core processors and thus extending the reliability of the system in handling faults by adding more backup processors. All processing nodes have the same internal structure except for the last two processors, as they differ from the rest in the number of I/O ports.

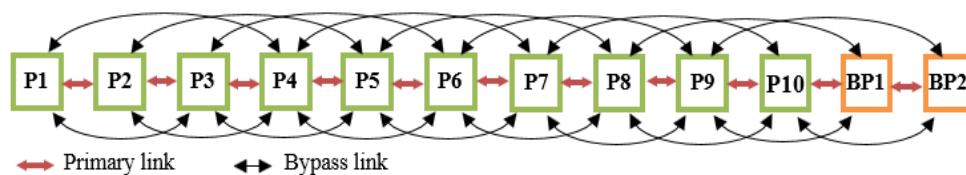


Figure 2. The architecture of the proposed multiprocessor system

Each link shown in Figure 2 consists of three signals:

- 8-bit channel ( $chn$ ).
- 1-bit channel request ( $RQ_{chn}$ ).
- 1-bit channel acknowledge ( $AK_{chn}$ ).

Where  $n$  is the channel number.

Every processor contains 3 input channels for communication with the 3 preceding processors and 3 output channels for communication with the 3 subsequent processors. Figure 3 shows the scheduling scheme for implementing the 1024-point decimation in time-fast Fourier transform (DIT-FFT) algorithm as a benchmark problem to evaluate system performance. In this figure, there is a series of ten stages that are executed on ten processors (fault-free system) each processor executing one stage.

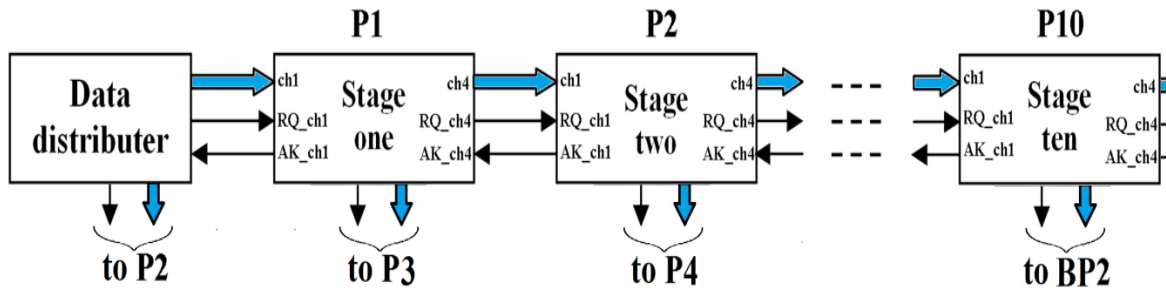


Figure 3. Scheduling 1024-point FFT algorithm in the proposed system

The communications between adjacent processors are done using the request-response protocol. When a processor wants to communicate with the neighboring processor, it sends a connection request and must wait for approval by receiving a response. To obtain high system performance, tasks must be scheduled between processors so that the task execution time is close or equal to reduce request-response time (i.e., reduce inter-processor communication overhead).

The primary concern for designing this system is to use a method to bypass permanent processor failures. The number of faults that can be resolved without affecting the implementation of the required task depends on the number of available backup processors. In this system, two backup processors were used, so the system can continue executing the task even when two processors malfunction. Once the malfunction is discovered, an appropriate mechanism is initiated to bypass the faulty processor and start the communication with the next processor to perform the task of the faulty processor.

#### 4. MATLAB SIMULATION MODEL

Figure 4 shows the simulation model for the proposed system, implemented using Stateflow/SimEvents in the MATLAB program. The unit delay block shown in Figure 4 is used with every  $AK\_CHn$  signal to hold and delay its input by the sample period we specified (one sample period delay). This block is corresponding to the  $z^{-1}$  discrete-time operator. It provides a mechanism for resampling the signal at a different rate. Our model contains multi-rate transitions, then we must add these blocks between the slow-to-fast transitions.

##### 4.1. Model assumptions

The proposed model contains 10 processors and two backup processors, each processor contains three input ports and three output ports. The malfunction is in the entire processor, i.e., as a processing unit and I/O ports, in order not to cause noticeable degradation in system performance, the number of failed processors should not exceed one or two. However, the number of failed processors can increase as the number of backup processors increases. Failure may occur on two adjacent or not adjacent processors. When any processor fails, tasks are shifted forward, meaning that the task of the idle processor is transferred to the next operated processor, and so on for the rest of the processors. Every processor ( $P_n$ ) has three I/O ports, one is the primary I/O port for communication with the directly adjacent processors ( $P_{n+1}$ ). While the other two are used as bypass links to communicate with the more distant processors ( $P_{n+2}$ , and  $P_{n+3}$ ), each I/O port is 8-bit wide. The I/O ports in every processor and the control signal associated with them are:

- Ports named ( $ch1$ ,  $ch2$ , and  $ch3$ ) are used as input ports to receive data from the preceding processor,  $P_{n-1}$ ,  $P_{n-2}$ , and  $P_{n-3}$ , respectively.
- The channel ReQuest signals  $RQ\_ch1$ ,  $RQ\_ch2$ , and  $RQ\_ch3$  are associated with the input ports and are used to receive the request to send data from the preceding processors. While the acknowledgment signals  $AK\_ch1$ ,  $AK\_ch2$ , and  $AK\_ch3$  are associated also with the input ports and are used to send the acknowledgment signals to the preceding processor.
- Ports named ( $ch4$ ,  $ch5$ , and  $ch6$ ) are used as output ports to send data to the next processors,  $P_{n+1}$ ,  $P_{n+2}$ , and  $P_{n+3}$ , respectively.
- The ReQuest signals  $RQ\_ch4$ ,  $RQ\_ch5$ , and  $RQ\_ch6$  are associated with the output ports and are used to send the request signal to send data to the next processors. While the acknowledgment signals  $AK\_ch4$ ,  $AK\_ch5$ , and  $AK\_ch6$  are associated also with the output ports and are used to receive the acknowledgment signals from the next processors.

### 4.2. Modeling the processor elements

The Processing elements represent the processors of a multiprocessor system. As mentioned earlier, twelve processing elements were used, including backup processing elements. The simulation models for the system elements can be divided into five part:

- a) Processor elements (P3-P9).
- b) Processor elements (P1-P2).
- c) Processor elements (P10-BP1).
- d) Processor element (BP2).
- e) Data distributor (DS).

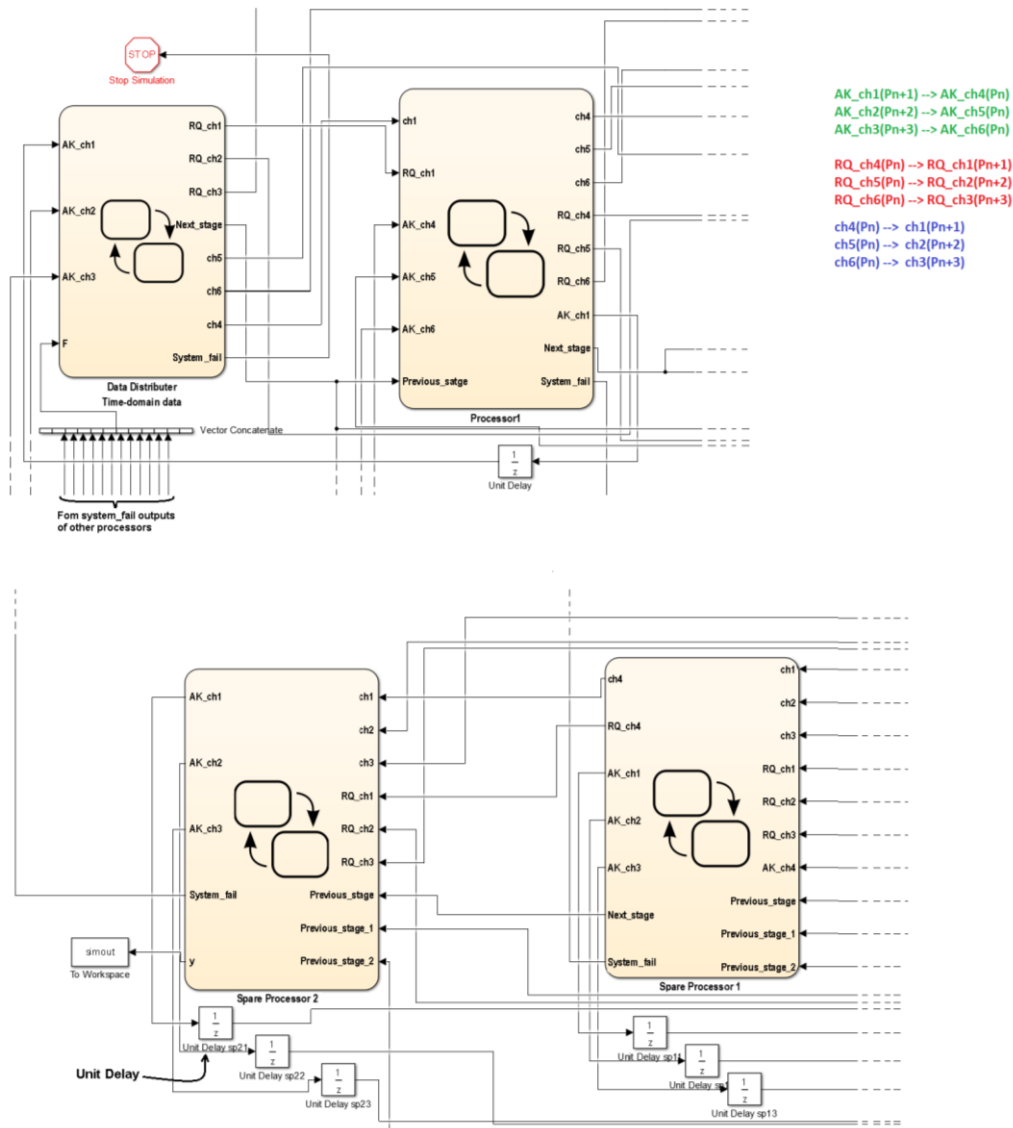


Figure 4. Simulation model for the proposed system

#### 4.2.1. Processor elements (P3-P9)

Processors P3 to P9 have the same simulation model as in Figure 5 using Stateflow/SimEvents program. The model consists of 21 states divided into the following groups: states in group-a are used to receive data transmission request signals ( $RQ\_chn$  signals) from the preceding processors. In the beginning, the processor (let's say processor number 3) checks the data transmission request signal ( $RQ\_ch1$ ) sent by the previous processor (i.e., processor number -2) connected to it via channel -1. If there is no response (i.e.,  $RQ - ch1 = 0$ ) then wait for a time (1 ms is chosen, it is linked to the variable  $w1$  so that the simulation result is ready at that step) and then the request signal ( $RQ\_ch1$ ) is rechecked.

This signal is re-checked many times (depending on the value of the variable  $w1$ ) before it is considered that the processor connected to channel-1 is faulty. At this point, it goes to the farthest processor (i.e., processor -1) to check the data transmission request signal  $RQ\_ch2$ . The same previous procedure is repeated in the process of examining the data transmission request signal, but with a different number of waiting times ( $w1$  is equal to 20 for the processor P3 shown in Figure 6). If there is no response, the processor P3 transfers to the remote processor, in this case, will be the DS data distributor with the different value of the variable  $w1$  which will be set in this case to 24. The value of  $w1$  in the delay loop is an important issue in this simulation. It was determined according to the requirements and mechanism of the simulation program, as well as based on the program settings, so that the simulation result in that step is ready, and it changes from one processor to another. Also, these values can change if the emulator settings are changed. As its value changes (incremental time accumulation) as the simulation stages progress. It also depends on the step size of the simulation software that represents the base sample time of the model, and it ensures that the simulator solver will take a step every sample time the model specifies.

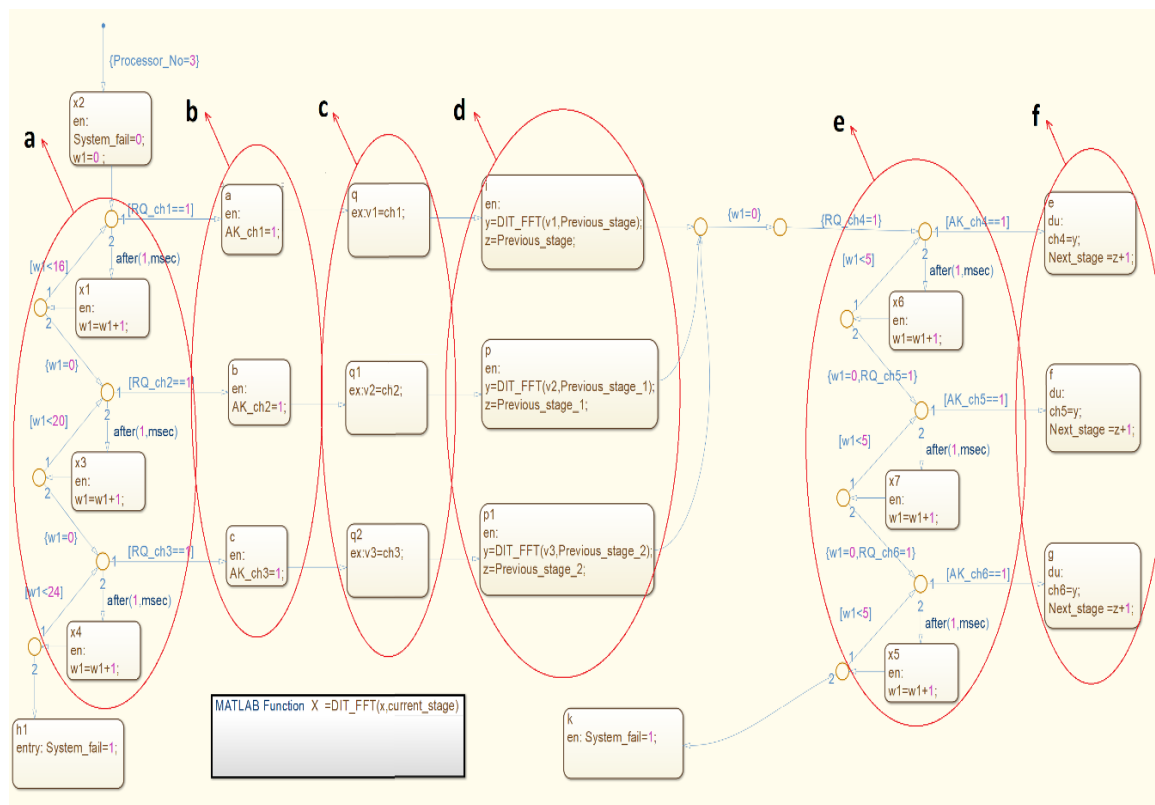


Figure 5. Stateflow diagram of the processors P3-P9

The states in group-b and c are used to initiate the values of acknowledging  $AK\_chn$  signal and data channel  $chn$  depending on the active processor (i.e., active  $RQ\_chn$ ) from which the data will be received. States in group-d are used to call the MATLAB function  $DIT\_FFT$  to execute one stage of the algorithm (depending on the location of the processor) using the data receiver from the preceding processor. In group-e, the states are used to send the request signal  $RQ\_chn$  (and receive the acknowledge signal  $AK\_chn$ ) to (from) the next processor. As in group-a states, this group uses the loops and delay times to determine the next processor which is P4 or P5 if P4 failed, or P6 if P4 and P5 failed. The last group of states (group-f) is used to send the processed data to the channel of the next active processor. Also, this group sends a “previous\_stage” signal which specifies the number of the last executed stage. This number is used by the next active processor to determine the next stage of the algorithm to be performed. In addition to the aforementioned state groups, Figure 5 contains some other states, which are state  $x1$  which is used as an initial state, and the  $x5$  and  $x15$  states where the system switches to when a failure occurs with more than two consecutive processors, previous or later of the current processor. Finally, there is a MATLAB function that is called by states of group f.



**4.2.2. Processor elements (P1-P2)**

The Stateflow diagram of the processors P1 and P2 is the same as the Stateflow diagram of the processors P3-P9 except that in P1 it has one Receiving reQuest (*RQ\_ch1*) and one data receiving channel (*ch1*) coming from the data distributor. While P2 has two data receiving requests (*RQ\_ch1* and *RQ\_ch2*) and two data transmission channels (*ch1* and *ch2*) that are connected to the P1 processor and the data distributor. Therefore, the states groups a, b, c, and d deal with one channel (*ch1*) concerning P1 and with two channels (*ch1* and *ch2*) concerning P2. As in P3-P9 processors, the group-e in P1 and P2, deals with three output channels (*ch4*, *ch5*, and *ch6*).

**4.2.3. Processor elements (P10-BP1)**

These processors have the same Stateflow diagram as processors 3-9, except for the last group (e). Whereas, processor 10 contains two output channels which are (*ch4* and *ch5*) connecting it with the processor BP1 and BP2. While processor BP1 contains one output channel (which is *ch4*) connecting it with the processor BP2.

**4.2.4. Processor element (BP2)**

BP2 has the same Stateflow diagram as P3-P9 except that does not have output communication ports. Since it is the last processor element. It contains only one output data channel to output the final result.

**4.2.5. Data distributor**

The data distributor is responsible for distributing the data in blocks, each block consisting of 1024 points. It does not have an input communication channel, only has three communication data channels for communication with the next processors (P1, P2, and P3) as shown in Figure 6. The MATLAB function *data\_gen* in this model is responsible to generate a block of 1024 points consciously (using the sine function) to perform 1024-point DIT-FFT algorithm. In radix-2 the DIT-FFT algorithm, the input is bit-reversed order (hence decimation-in-time) and the output is in order. The bit-reversal process is performed by the data distributor unit.

The *system\_fail* signals coming from all processors are entered to OR operation in data distributor and the output of OR operation is output from DS unit as signal *F* (system failure). If any of the *system\_fail* signals is set to one (meaning more than two adjacent processors are failed) then the output *F* becomes 1. This output is connected to the block named “stop” which is used to stop the simulation when the input is non-zero.

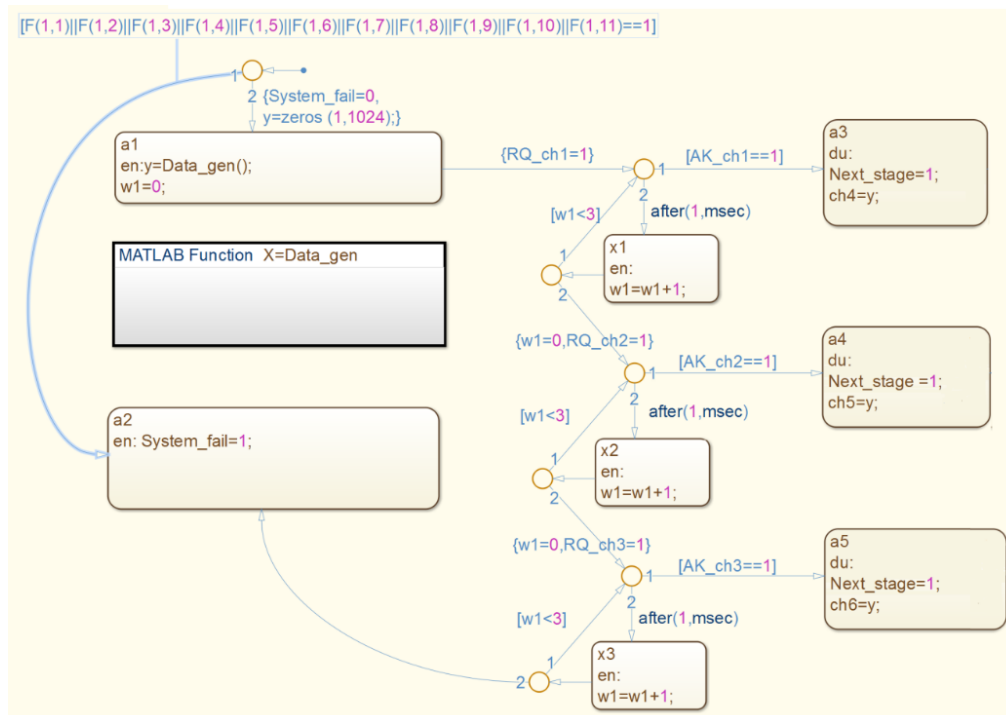


Figure 6. Stateflow diagram of the data distributor

## 5. PERFORMANCE EVALUATION AND RESULTS

The integrity of the proposed model has been examined by executing a radix-2 decimation in time (DIT) 1024-point FFT algorithm. To schedule the algorithm on the proposed multiprocessor system, we assumed the existence of a circuit called the data distributor (see Figure 3). The task of this circuit is to distribute the data points in the form of blocks, each block consisting of 1024 to be executed through ten stages, each processor executes one stage. Several scenarios will be studied to verify the ability of the system to handle malfunctions at the processor level. The following scenarios are implemented:

- No processor malfunctioning (system performance in the extreme case).
- In the event of a malfunction in one or two processors (contiguous or non-contiguous), the system can under these circumstances address the faults, reconfigure the system and then continue to work with almost the same efficiency.
- When a malfunction occurs in more than two processors (provided that the number of failed processors does not exceed two contiguous), the system can continue to work, but this will affect the efficiency of the system's performance. Here, the remaining stages of the algorithm will be carried over to the last processor (BP2) for implementation. BP2 can know the number of remaining stages by checking the "previous\_stage" signal value.
- When a malfunction occurs in more than two neighboring processors, the current system fails and stops working. Generally, it depends on the number of backup processors used. Figure 7 shows the configuration of the system under different scenarios. Where Figure 7(a) represents a fault-free case, Figure 7(b) one fault case, Figure 7(c) two adjacent faults case, and Figure 7(d) two non-adjacent faults case.

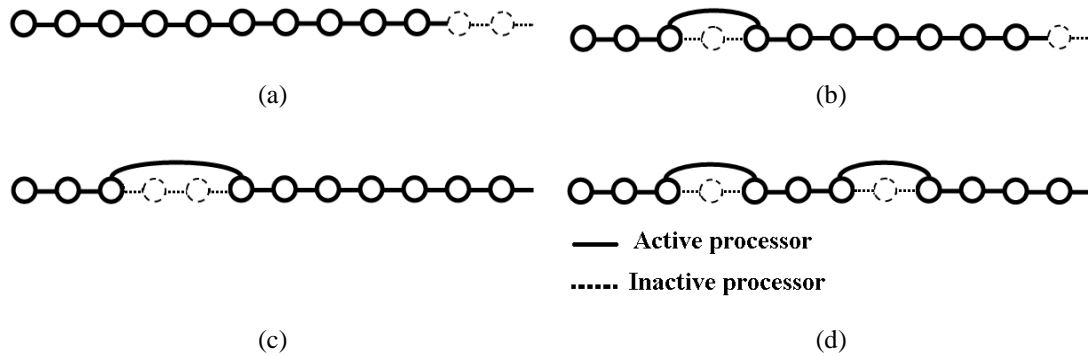


Figure 7. Shows system configuration under different scenarios: (a) fault free, (b) one fault, (c) two adjacent faults, and (d) two non-adjacent faults

When P1 completes the first stage, the results are sent to the next processor to complete the second stage of the algorithm and at the same time, a new block of points is received from the data distributor to start implementing the first stage of the new block, and so on. We implement the proposed framework using MATLAB on a 2.30 GHz 32-bit Windows 10 machine with 4 GB RAM. Matlab is not a real-time platform, so estimating the execution time of the proposed model under different scenarios using commands like Tic / Toc can give us a different value every time for various reasons, such as processor activity. However, the difference in the execution time of the model between the absence of a fault and the failure of one or more processors will be only in the reconfiguration overhead time. This time will have a slight impact (negligible) on the performance of the system for the following reasons:

- The overhead time will be small compared to the execution time of the algorithm.
- The fault detection procedure is only executed while the fault is occurring.
- Due to the distributing processing nature, the proposed architecture can handle multiple faults which occur at the same time which hides the overload caused by the fault detection protocol and the system reconfiguration.

Also, the overhead time caused by the reconfiguration is independent of the system size. Figure 8 shows the magnitude and phase response due to the implementation 1024-point FFT algorithm of the sine function in the simulated system obtained from the system in the absence of a failure, or a failure of one or two processors. Another situation to consider is that the efficiency of a multiprocessor system using a pipeline architecture is greatly influenced by the task scheduling process and the waiting time between the processors. Whenever the task processing times in system processors are equal to or close to that, the better the system efficiency. Conversely, the greater the difference between task processing times in different processors, the lower the system's performance.



The scheme presented in this paper achieved fault tolerance by effectively exploiting the redundant resources in the system, namely backup processors. The proposed system is characterized by the absence of centralized control to discover malfunctions and deal with the rescheduling of tasks, as this process is done in a decentralized manner without the need for additional resources.

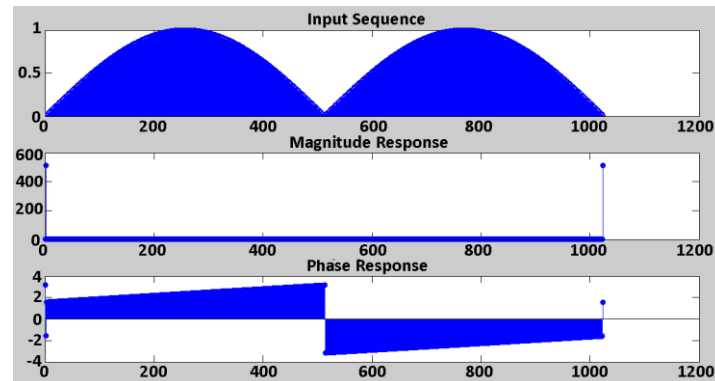


Figure 8. Magnitude and phase response of implementing 1024-point DIT-FFT algorithm of the sine function

## 6. CONCLUSION

Simulation of discrete events is an important way to determine the performance and reliability of many systems, including multiprocessor systems. This paper presents a fault-tolerant multiprocessor architecture suitable for applications requiring a high degree of reliability. The proposed system architecture meets the following requirements.

The ability to deal with processor failures by allocating additional processors as backup processors to be integrated into the system when the failure occurs. The proposed system utilizes the pipeline structure with the task forward-moving mechanism when the fault occurs to cross the defective processor. The proposed architecture is scalable, as the number of basic and backup processors can be increased. The proposed architecture relies on the use of I/O ports as an easy and cost-effective means of communication between processors. The proposed model was validated by implementing the FFT algorithm as a benchmark tool for the validity and effectiveness of the design and the mechanism for overcoming failure in one or more processors with minimal overhead.




## REFERENCE

- [1] S. Tzilis, "Runtime Management of Multiprocessor Systems for Fault Tolerance, Energy Efficiency and Load Balancing." Ph.D. dissertation, Division of Computer Engineering, Department of Computer Science and Engineering, Chalmers University of Technology, Göteborg, Sweden, 2019. [Online]. Available: [https://research.chalmers.se/publication/508764/file/508764\\_Fulltext.pdf](https://research.chalmers.se/publication/508764/file/508764_Fulltext.pdf)
- [2] S. S. Kolisetty and B. S. Rao, "Scalable epidemic message passing interface fault tolerance," *Bulletin of Electrical Engineering and Informatics*, vol. 11, no. 2, pp. 998-1006, 2022, doi: 10.11591/eei.v11i2.3374.
- [3] V. Von, "Fault tolerance in Parallel Data Processing Systems," Ph.D. dissertation, Faculty of Electrical Engineering and Computer Science, Technical University of Berlin, Germany, 2019. [Online]. Available: [https://depositonce.tu-berlin.de/bitstream/11303/10101/4/hoeger\\_mareike.pdf](https://depositonce.tu-berlin.de/bitstream/11303/10101/4/hoeger_mareike.pdf)
- [4] J. Wu, Y. Wu, G. Jiang, and S. K. Lam, "Algorithms for Reconfiguring NoC-based Fault-Tolerant Multiprocessor Arrays," *Journal of Circuits, Systems, and Computers*, vol. 28, no. 7, 2018, doi: 10.1142/S0218126619501111.
- [5] A. F. Mahmood, "A Pipelined Fault Tolerant Architecture for Real time DSP Applications," *AL-Rafidain Engineering Journal*, vol. 16, no. 4, pp. 93-102, 2008, doi: 10.33899/rengj.2008.44737.
- [6] K. Zhigalov, S. Y. Kuznetsova, and M. V. Sygotina, "Development of functional fault-tolerant system," *Journal of Physics: Conference Series*, 2020, vol. 1661, doi: 10.1088/1742-6596/1661/1/012166.
- [7] H. Mushtaq, Z. Al-Ars, and K. Bertels, "Survey of fault tolerance techniques for shared memory multicore/multiprocessor systems," In *2011 IEEE 6th International Design and Test Workshop (IDT)*, 2011, pp. 12-17, doi: 10.1109/idt.2011.6123094.
- [8] T. A. Fathi, H. N. Yahya, and Q. A. Hasan, "Implementation and Simulation of a Multiprocessor System Using 8051 Microcontroller for Industrial Applications," in *Proc. of the 1st International Multi-Disciplinary Conference Theme: Sustainable Development and Smart Planning, IMDC-SDSP*, 2020, doi: 10.4108/eai.28-6-2020.2298126.
- [9] A. Akram and L. Sawalha, "A Survey of Computer Architecture Simulation Techniques and Tools," *IEEE Access*, vol. 7, pp. 78120-78145, 2019, doi: 10.1109/ACCESS.2019.2917698.
- [10] L. Thiele, I. Bacivarov, W. Haid, and K. Huang, "Mapping Applications to Tiled Multiprocessor Embedded Systems," in *Seventh International Conference on Application of Concurrency to System Design (ACSD 2007)*, 2007, pp. 29-40, doi: 10.1109/acsd.2007.53.
- [11] The MathWorks, Inc. *MATLAB&SIMULINK Stateflow User's Guide*. Natick, MA: The MathWorks, 1997-2022. [Online]. Available: [https://www.mathworks.com/help/pdf\\_doc/stateflow/stateflow Ug.pdf](https://www.mathworks.com/help/pdf_doc/stateflow/stateflow Ug.pdf)
- [12] H. Javaid and S. Parameswaran, *Pipelined multiprocessor system-on-chip for multimedia*, Switzerland: Springer Cham, 2014, doi: 10.1007/978-3-319-01113-4.




- [13] H. Javaid, A. Ignjatovic, and S. Parameswaran, "Performance estimation of pipelined multiprocessor system-on-chips (MPSoCs)," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2159-2168, 2014, doi: 10.1109/TPDS.2013.268.
- [14] L. Pomante, V. Muttillio, M. Santic, and P. Serri, "SystemC-based electronic system-level design space exploration environment for dedicated heterogeneous multi-processor systems," *Microprocessors and Microsystems*, vol. 72, 2020, doi: 10.1016/j.micpro.2019.102898.
- [15] A. Alali, I. Assayad, and M. Sadik, "Multilevel MPSoC Performance Evaluation, ISS Model with Timing and Priority Management," in *UNet 2015: Advances in Ubiquitous Networking*, 2015, pp 425-437, doi: 10.1007/978-981-287-990-5\_34.
- [16] H. Ismail, D. N. A. Jawawi, I. Ahmedy and M. A. Isa, "Evaluation of the Proposed Hybrid Multiprocessor Real-Time Scheduling Approach with Partitioned and Global Approaches," in: *Engineering Application of Artificial Intelligence Conference 2018 (EAAIC 2018)*, 2018. [Online]. Available: [https://www.matec-conferences.org/articles/mateconf/pdf/2019/04/mateconf\\_eaaic2018\\_05004.pdf](https://www.matec-conferences.org/articles/mateconf/pdf/2019/04/mateconf_eaaic2018_05004.pdf)
- [17] A. Alali, I. Assayad, and M. Sadik, "Modeling and simulation of multiprocessor systems MPSoC by SystemC/TLM2," *arXiv preprint*, 2014. [Online]. Available: <https://arxiv.org/ftp/arxiv/papers/1408/1408.0982.pdf>
- [18] K. Huang *et al.*, "Energy-efficient fault-tolerant mapping and scheduling on heterogeneous multiprocessor real-time systems," *IEEE Access*, vol. 6, pp. 57614-57630, 2018, doi: 10.1109/access.2018.2873641.
- [19] X. -B. Yu, J. -S. Zhao, C. -W. Zheng, and X. -H. Hu, "A Fault-Tolerant Scheduling Algorithm using Hybrid Overloading Technology for Dynamic Grouping based Multiprocessor Systems," *International Journal of Computers Communications & Control*, vol. 7, no. 5, 2012, doi: 10.15837/ijccc.2012.5.1358.
- [20] A. Choudhury and B. K. Sikdar, "Modeling Remapping Based Fault Tolerance Techniques for Chip Multiprocessor Cache with Design Space Exploration," *Journal of Electronic Testing*, vol. 36, pp. 59-73, 2020, doi: 10.1007/s10836-019-05852-6.
- [21] B. Yuan, H. Chen, and X. Yao, "Toward Efficient Design Space Exploration for Fault-Tolerant Multiprocessor Systems," *IEEE transactions on evolutionary computation*, vol. 24, no. 1, pp. 157-169, 2020, doi: 10.1109/tevc.2019.2912726.
- [22] H. Aliee and H. R. Zarandi, "A Fault-Tolerant, Dynamically Scheduled Pipeline Structure for Chip Multiprocessors," in *SAFECOMP 2011: Computer Safety, Reliability, and Security*, 2011, pp. 324-337, doi: 10.1007/978-3-642-24270-0\_24.
- [23] H. Baek, and J. Lee, "Task-Level Re-Execution Framework for Improving Fault Tolerance on Symmetry Multiprocessors," *Symmetry*, vol. 11, no. 5, 2019, doi: 10.3390/sym11050651.
- [24] P. P. Nair, A. Sarkar, and S. Biswas, "Fault-Tolerant Real-Time Fair Scheduling on Multiprocessor Systems with Cold-Standby," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 4, pp. 1718-1732, 2021, doi: 10.1109/TDSC.2019.2934098.
- [25] A. F. Al-Allaf and S. W. Nayif, "An FPGA-based Fault Tolerance Hypercube Multiprocessor DSP System," *AL-Rafidain Engineering Journal (AREJ)*, vol. 18, no. 1, pp. 69-82, doi: 10.33899/rengj.2010.27993.
- [26] A. Anand, Y. Rajendra, S. Narayanan, and P. V. Radhamani, "Modelling, implementation and testing of an effective fault tolerant multiprocessor real-time system," *2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing*, 2012, pp. 107-113, doi: 10.1109/PDGC.2012.6449800.
- [27] V. Vargas, P. Ramos, J. -F. Méhaut, and R. Velazco, "NMR-MPar: A fault-tolerance approach for multi-core and many-core processors," *Applied Sciences*, vol. 8, no. 3, 2018, doi: 10.3390/app8030465.
- [28] G. D'Angelo, S. Ferretti, and M. Marzolla, "Fault tolerant adaptive parallel and distributed simulation through functional replication," *Simulation Modelling Practice and Theory*, vol. 93, pp. 192-207, 2019, doi: 10.1016/j.simpat.2018.09.012.
- [29] A. R. P. Domingues, and J. C. Hamerski, and A. D. M. Amory, "A fault recovery protocol for brokers in centralized publish-subscribe systems targeting multiprocessor systems-on-chips," *Analog Integrated Circuits and Signal Processing*, vol. 106, pp. 139-154, 2021, doi: 10.1007/s10470-020-01637-6.
- [30] F. F. Barreto, A. M. Amory, and F. G. Moraes, "Fault recovery protocol for distributed memory MPSoCs," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 421-424, doi: 10.1109/iscas.2015.7168660.

## BIOGRAPHIES OF AUTHOR



**Ahmad F. Al-Allaf**    received the B.Sc. degree in electrical engineering, M.Sc. degree in electronics and Communication and Ph.D. in computer networks from the department of electrical engineering, University of Mosul, Iraq. He has published many research papers in the areas of computer networks, fault tolerance computing, reconfigurable computing, and parallel processing. From July 2006 until now, he has been a faculty member in the Computer Technology Engineering Department at the Technical College of Engineering / Mosul - Northern Technical University, Iraq. His research interests include reconfigurable computing, fault-tolerant computing, computer network hardware design, and parallel processing. He can be contacted at email: [Ahmed.faleh@ntu.edu.iq](mailto:Ahmed.faleh@ntu.edu.iq).



**Ziyad Khalaf Farej**    is an Assist. Prof. and has BSc. in Electronic System Eng. from Cranfield University/UK with honor degree in 1989.MSc. in Spread Spectrum System/FH from Mosul University in 2003. Ph.D. in Computer and Communication Networks (Mesh Network based WIMAX IEEE 802.16 Standard technology) from Mosul University in 2012 with excellent grade. He has been to Salford Greater Manchester University/UK in Research Scholarship for 6 months during his Ph.D. studying period. He can be contacted at email: [drziyad.farej@ntu.edu.iq](mailto:drziyad.farej@ntu.edu.iq).