ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

MODEL-BASED DESIGN AND IMPLEMENTATION OF SCHEDULERS IN ARINC-664 END SYSTEM AS A SYSTEM ON CHIP

M.Sc. THESIS

Mustafa UZUNER

Department of Electronics Engineering

Electronics Engineering Programme

JANUARY 2022

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

MODEL-BASED DESIGN AND IMPLEMENTATION OF SCHEDULERS IN ARINC-664 END SYSTEM AS A SYSTEM ON CHIP

M.Sc. THESIS

Mustafa UZUNER
(504181280)

Department of Electronics Engineering

Electronics Engineering Programme

Thesis Advisor: Prof. Dr. Sıddıka Berna ÖRS YALÇIN
Thesis Co-Advisor: Dr. İbrahim HÖKELEK

JANUARY 2022

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

ARINC-664 UÇ SİSTEMİNDE ÇİZELGELEYİCİLERİN MODEL TABANLI TASARIMI VE KIRMIK ÜSTÜ SİSTEM UYGULAMASI

YÜKSEK LİSANS TEZİ

Mustafa UZUNER
(504181280)

Elektronik Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Sıddıka Berna ÖRS YALÇIN
Eş Danışman: Dr. İbrahim HÖKELEK

OCAK 2022

Mustafa UZUNER, a M.Sc. student of ITU Graduate School student ID 504181280, successfully defended the thesis entitled “MODEL-BASED DESIGN AND IMPLEMENTATION OF SCHEDULERS IN ARINC-664 END SYSTEM AS A SYSTEM ON CHIP”, which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor : **Prof. Dr. Sıddıka Berna ÖRS YALÇIN** ....
İstanbul Technical University

Co-advisor : **Dr. İbrahim HÖKELEK** ....
TÜBİTAK BİLGEM

Jury Members : **Asst. Prof. Dr. Tankut AKGÜL** ....
İstanbul Technical University

Assoc. Prof. Dr. Ali Emre PUSANE ....
Boğaziçi University

Asst. Prof. Dr. Faik BAŞKAYA ....
Boğaziçi University

Date of Submission : 2 Jan 2022
Date of Defense : 9 Feb 2022

To my family,

FOREWORD

First, I would like to show my gratitude to my supervisors, Prof. Dr. Sıddıka Berna ÖRS YALÇIN and Dr. İbrahim HÖKELEK, for their guidance, help, and advice throughout my MSc studies.

I want to thank my family for their support.

Lastly, I want to thank my colleagues at TUSAŞ and TÜBİTAK BİLGEM for their valuable ideas and contributions.

January 2022
Mustafa UZUNER
Digital Design Engineer

TABLE OF CONTENTS

ABBREVIATIONS

AXI : Advanced Extensible Interface
BAG : Bandwidth Allocation Gap
BRAM : Block Random Access Memory
DetNet : IETF Determinisic Networking
DMA : Direct Memory Access
DRR : Deficit Round Robin
DWRR : Deficit Weighted Round Robin
EDD : Earliest-Due-Date
ES : End System
FCFS : First-Come-First-Served
FIFO : First-In-First-Out
FPGA : Field Programmable Gate Array
FWFT : First-Word-Fall-Through
FF : Flip-Flop
Gbit/s : Gigabit per second
GPS : Generalized Processor Sharing
HDL : Hardware Description Language
HLS : High Level Synthesis
HoL : Head of Line
IMA : Integrated Modular Avionics
Mbit/s : Megabit per second
IP : Intellectual Property
IP Layer : Internet Protocol Layer
LFE : Largest Frame Earliest
LUT : Look Up Table
LQ : Longest Queue
MAC : Medium Access Control
MHz : MegaHertz
ms : millisecond
ns : nanosecond
PCIe : Peripheral Component Interface Express
PL : Programmable Logic
PS : Processing Subsystem
RR : Round Robin
RTL : Register Transfer Level
SB : Smallest BAG
SEU : Single Event Upset
SFE : Smallest Frame Earliest
SoC : System on Chip
SP : Strict Priority
SS : Smallest Size
SWaP : Size Weight and Power

Tcl : Tool command language
TSN : Time Sensitive Networking
TTEthernet : Time-Triggered Ethernet
UDP : User Datagram Protocol
us : microsecond
VHDL : Very High Speed Integrated Circuit Hardware Description Language
WF2Q : Worst-Case Weighted Fair Queueing
WF2Q+ : Worst-Case Weighted Fair Queueing+
WFQ : Weighted Fair Queueing
VL : Virtual Link
VL ID : Virtual Link Identification
WRR : Weighted Round Robin

SYMBOLS

D : Deterministic Process
E : Expected Value
G : General Process
$\mathbf{L}_{\mathbf{max}}$: Maximum Frame Length in ARINC-664
$\mathbf{L}_{\mathbf{min}}$: Minimum Frame Length in ARINC-664
L : Number of Items in a Queue
M : Markovian-Poisson Process
U : Utilization
W : Average Waiting Time an Item Spends in the Queue
λ : Arrival Rate
μ : Service Rate

LIST OF TABLES

LIST OF FIGURES

Page

MODEL-BASED DESIGN AND IMPLEMENTATION OF SCHEDULERS IN ARINC-664 END SYSTEM AS A SYSTEM ON CHIP

SUMMARY

In the last decades, the amount of exchanged data between avionics systems is tremendously increased. Traditional communication networks such as MILSTD-1553 and ARINC-429 cannot provide enough bandwidth for avionic systems. Instead, ARINC-Specification 664 Part 7 (ARINC-664) is proposed for next-generation aircraft.

ARINC-664 defines an Ethernet-based deterministic network protocol that provides bounded delay and jitter using redundant communication among the avionics applications. Achieving the end-to-end bounded delay objectives requires that incoming Ethernet frames must be regulated according to the ARINC-664 standard. In ARINC-664, each rate-constrained flow, i.e., Virtual Link (VL), is regulated by using End Systems (ESs) and Bandwidth Allocation Gap (BAG). Each regulated VL must be served at a time, so a scheduling mechanism must be used when more than one queue is ready to be served. ARINC-664 standard does not specify the details of the scheduling algorithm. However, some algorithms are proposed in the literature for ARINC-664 scheduling.

Field Programmable Gate Array (FPGA) is one of the most preferred implementation choices for ARINC-664 due to its low power consumption, low latency data transfer, and security advantages. Traditional FPGA development requires building design and verification with Hardware Description Languages (HDLs). Instead of this time-consuming FPGA development, using a model-based hardware design enables faster prototyping and testing environment.

In this thesis, first, a Single Queue model is designed and developed in Simulink to provide a basic queueing infrastructure for ARINC-664 ES. Then, the ARINC-664 ES model is developed on top of the Single Queue model. The scheduling algorithms in ARINC-664 ES are designed and developed using HDL convertible components. The Smallest BAG (SB), the Smallest Size (SS), the Longest Queue (LQ), and the First-In-First-Out (FIFO) ARINC-664 ES scheduling algorithms are implemented. This implementation allows collecting the mean, standard deviation, and maximum of jitter performances of the scheduling algorithms. In addition, an ARINC-664 ES Dynamic Scheduler model whose components can be converted to HDLs and C/C++ is built. This model contains all the scheduling algorithms, and the user can switch among the scheduling algorithms while the model is operating.

ARINC-664 UÇ SİSTEMİNDE ÇİZELGELEYİCİLERİN MODEL TABANLI TASARIMI VE KIRMIK ÜSTÜ SİSTEM UYGULAMASI

ÖZET

Birleşik aviyonik mimariler, yeni nesil uçaklarda aviyonik sistemlerin yüksek miktardaki bilgi alışverişi gereksinimlerini karşılamakta yetersiz hale gelmiştir. Alternatif olarak, ortak donanım ve yazılım modülleriyle paylaşılan bir bilgi işlem platformu üzerinde farklı kritiklik seviyelerine sahip birden fazla aviyonik uygulamasını barındıran Entegre Modüler Aviyonik (Integrated Modular Avionics - IMA) mimarileri, yapısal boyut, ağırlık ve güç avantajları nedeniyle tercih edilmektedir.

Ethernet tabanlı gerekirci ağ çözümleri çoğunlukla IMA mimarileri için yüksek hızlı yerel alan ağı olarak kullanılır. ARINC 429 ve MIL-STD 1553 gibi haberleşme standartları gerekirci yapıda olmalarına rağmen yeni nesil ağ sistemlerinin yüksek bant genişliği gereksinimlerini karşılamakta yetersiz kalmaktadır. Son teknoloji Ethernet tabanlı gerekirci ağ çözümlerine ARINC Spesifikasyonu 664 Bölüm 7 (ARINC-664), IEEE Zamana Duyarlı Ağ Oluşturma (Time Sensitive Networks - TSN), Zaman Tetiklemeli Ethernet (Time Triggered Ethernet - TTEthernet) ve Deterministik Ağ Oluşturma (DetNet) örnek gösterilebilir.

Zaman kritik uygulamaların en temel özellikleri sınırlı gecikme (bounded delay), düşük oranda bilgi kaybı (low data loss-rate) ve düşük seğirmedir (jitter). ARINC-664 her bir Ethernet paketinin transferi için sınırlı bant genişliği kullanarak sınırlı gecikme ve düşük seğirme sağlar. ARINC-664 karmaşık bir zaman senkronizasyonu mekanizması gerektirmez. Aynı zamanda, çakışmaya izin vermemesi, hata kaldırır yapısı ve yedekli haberleşme topolojisi sayesinde düşük oranda bilgi kaybı sağlar. Bütün bu özellikleri sayesinde ARINC-664 havacılıktaki zaman kritik sistemlerde sıklıkla kullanılmaktadır.

ARINC-664 standardı, Airbus tarafından yeni nesil uçak veri ağı olarak geliştirilmiştir. ARINC-664'te, her akışın, yani Sanal Bağlantı'nın (Virtual Link - VL) hızı düzenlenmiş ve sınırlandırılmıştır. Bu sınırlama ve düzenleme Bant Genişliği Tahsis Aralığı (Bandwidth Allocation Gap - BAG) konsepti ile Uç Sistemde (End System - ES) sızdıran kova (leaky bucket) algoritması kullanılarak sağlanır. ARINC-664, geleneksel ağ sistemlerinin aksine, bir Ethernet çerçevesinde Ortam Erişim Kontrolü (Medium Access Control - MAC) hedef adresinde taşınan Sanal Bağlantı Kimliği'ni (Virtual Link Identification - VL ID) kullanarak paketleri yönlendirir.

ARINC-664 Uç Sistem hem donanım hem de yazılım üzerine uygulanabilir bir protokoldür. Alan Programlanabilir Kapı Dizileri (Field Programmable Gate Array - FPGA) düşük güç tüketimi, düşük gecikmeli bilgi transferi ve güvenilirliği

sebebiyle ARINC-664 Uç Sistem gerçeklemesi için uygundur. FPGA ile uygulama geliştirirken en bilinen yaklaşım gerçeklenecek olan uygulamaların önce simülasyon ve modellemelerinin tamamlanıp sonuçların elde edilmesi, daha sonra ise FPGA implementasyonunun gerçeklenmesidir. Bu yaklaşımda, simülasyon ve modeller sadece doğrulama ve onaylama amacıyla kullanılır. Bu yaklaşımın geliştirme süresi ve iş gücü gereksinimi açısından verimli olmadığı açıktır. Alternatif yaklaşım, simülasyon modelinden Donanım Tanımlama Dili'ne (HDL) ve yazılım koduna doğrudan dönüştürmeye izin veren MATLAB Simulink programını kullanarak model tabanlı bir sistem oluşturmaktır.

ARINC-664 Uç Sistemi'nin gönderici tarafında, Sanal Ağ kuyrukları (VL queues), trafik şekillendirme (traffic shaping), çizelgeleyici (scheduling) ve trafik yedekleyicisi (redundancy manager) ile alıcı tarafında tutarlılık sezgi programı (integrity checker) ve fazlalık denetleyicisi (redundancy checker) fonksiyonları bulunmaktadır. Bu çalışmada Sanal Ağ kuyrukları, trafik şekillendirme ve çizelgeleyici fonksiyonları MATLAB Simulink üzerinde gerçeklenmiştir.

Kuyruk teorisi (queueing theory), kuyruktaki elemanların bekleme süresi sayısını tahmin etmek amacıyla yaygın olarak kullanılmaktadır. Bu çalışmada, öncelikle, kuyruk teorisinin temellerini esas alan tek bir kuyruk üreteci modeli MATLAB Simulink üzerinde gerçeklenmiştir. Bu üretecin amacı ARINC-664 Uç Sistemi'nde kullanılacak olan Sanal Ağ kuyruklarına Ethernet paketlerini yüklemektir. Model basit bir servis edici ile yüklenen Ethernet paketlerinin servis edilmesini sağlamaktadır. Bu modelin doğru çalışması ARINC-664 Uç Sistem modelinin doğru çalışması için elzemdir. Bu sebeple bu modelden elde edilen sonuçlar kuyruk teorisinin teorik sonuçları ile karşılaştırılmıştır ve kuyruk üreteci modelinin doğru bir şekilde çalıştığı gözlemlenmiştir.

Tek bir kuyruk için hazırlanan kuyruk üreteci modeli kopyalanarak birden fazla kuyruğa Ethernet paketlerini bağımsız olarak yükleyen bir sistem oluşturulmuştur. Birden fazla kuyruğun Ethernet paketlerinin servis edilmesi için bir çizelgeleyici uygulamasının oluşturulması gerekir. Bu amaçla içerisinde trafik düzenleyicisi ve çizelgeleyici bulunduran bir trafik servis edici modeli MATLAB Simulink üzerinde gerçeklenmiştir. Bu model donanım tanımlama dillerine dönüştürülebilir şekilde oluşturulmuştur. Modelde 4 adet çizelgeleyici algoritması gerçeklenmiştir. Bu algoritmalar En Küçük Bant Genişliği Tahsis Aralığı (Smallest BAG), İlk-Giren-İlk-Çıkar (FIFO), En Küçük Paket (Smallest Size) ve En Uzun Kuyruk (Longest Queue) olarak sıralanabilir.

Gecikme ve seğirme ağ sistemlerinin servis kalitesi (Quality of Service - QoS) üzerinde çok önemli bir etkiye sahiptir. Çizelgeleyici algoritmalarının her biri farklı karakteristiklere sahip olduğu için, bu algoritmaların gecikme sonuçları birbirinden farklı olacaktır. Bu çalışmada çizelgeleyici algoritmalarının ARINC-664 Uç Sistemi'ndeki her bir kuyruk için seğirme ortalaması, standart sapması ve maksimum seğirmenin hesaplanması amacıyla bir analiz modülü tasarlanmıştır. Bütün bu tasarımlar ARINC-664 Uç Sistem modelini oluşturmaktadır.

Yukarıda bahsedilen ARINC-664 Uç Sistem modeli sekiz kuyruk için tasarlanmış ve analiz sonuçları iki farklı konfigurasyon senaryosu için raporlanmıştır. Bu sonuçlar incelendiğinde, her bir çizelgeleyici algoritmasının avantajları ve dezavantajları olduğu görülür. Bu sebeple, kullanıcının sistem çalışırken çizelgeleyici algoritmaları arasında geçiş yapabilmesini sağlamak amacıyla bir dinamik çizelgeleyici modülü tasarlanmıştır. Bu modül bütün çizelgeleyici algoritmalarını içermektedir ve kullanılan algoritma analiz modülünden çıkan sonuçlara göre değiştirilebilir. Bu modül de donanım tanımlama dillerine dönüştürülebilir olarak tasarlanmıştır. Daha sonra, dinamik çizelgeleyici modülünün kendi kararlarını vererek algoritmalar arasında geçiş yapabilmesini sağlamak amacıyla ARINC-664 Uç Sistemi modeli için tasarlanan analiz modülü MATLAB Simulink ile C kodunda dönüştürülebilir hale getirilmiştir. Dinamik sistem modelindeki donanıma tanımlama diline dönüştürülen kodlar Programlanabilir Lojik (Programmable Logic - PL)'e, C koduna dönüştürülen kodlar ise İşlemci Alt Sistem (Processor Subsystem - PS)'e gömülerek Gelişmiş Genişletilebilir Arayüz (Advanced Extensible Interface - AXI) ve İki Kanallı Rastgele Erişilebilir Bellek (Dual Port Block Random Access Memory - BRAM) ile birbirlerine bağlanmıştır.

Tezin sonuçları çeşitli senaryolar için çizelgeleyici algoritmasının performanslarını ve FPGA implementasyonu sonucunda oluşan kaynak kullanımlarını içermektedir. Bu sonuçlar her çizelgeleyici uygulamasının avantajları ve dezavantajları olduğunu göstermektedir. Aynı zamanda ARINC-664 Uç Sistem'in MATLAB Simulink ile hızlı prototipleme ve test etmeye uygun olduğunu ve akıllı bir çizelgeleyiciye sahip ARINC-664 Uç Sistemi'nin tasarlanabileceğini basit bir senaryo kullanarak göstermiştir.

1. INTRODUCTION

Federated avionics architectures have become insufficient to meet the tremendous computing requirements of next-generation aircraft. Instead, Integrated Modular Avionics (IMA) architectures [1] hosting multiple avionics applications of different criticalities on a shared computing platform with common hardware and software modules are preferred due to their inherent size, weight, and power (SWaP) advantages [2].

Ethernet-based deterministic network solutions are primarily used as a high-speed local area network for IMA systems. State-of-the-art Ethernet-based deterministic network solutions include ARINC Specification 664 Part 7 (ARINC-664) [3], IEEE Time-Sensitive Networking (TSN) [4], Time-Triggered Ethernet (TTEthernet) [5], and IETF Deterministic Networking (DetNet) [6]. The essential necessities of time-critical applications are bounded latency, low data loss rates, and low packet delay variation (jitter). ARINC-664 provides bounded latency and jitter by using a limited band rate for Ethernet frames. It does not require a complex time synchronization mechanism. Also, it offers low data loss rates with its congestion-free, fault-tolerant, and redundant communication topology. These features make ARINC-664 an excellent candidate for time-critical avionic applications.

ARINC-664 standard is developed by Airbus as the next-generation aircraft data network. In the ARINC-664, each rate-constrained flow, i.e., Virtual Link (VL), is regulated by using the leaky bucket algorithm at End System (ES) with the concept of Bandwidth Allocation Gap (BAG). Contrary to traditional network systems, ARINC-664 routes packets using Virtual Link Identifier (VL ID), which is carried in the Medium Access Control (MAC) destination address in an Ethernet frame.

ARINC-664 ES can be implemented in either hardware or software, where Field Programmable Gate Array (FPGA) is a superior implementation choice due to its low power consumption, low latency data transfer, and security [7]. However, an FPGA-based design takes a significant effort, especially when several alternative algorithms need to be compared and contrasted. The most common approach is to perform the modeling and simulation studies before the FPGA implementation. In this approach, the simulation models are solely used for validation and verification purposes, and FPGA implementation needs to be done from scratch without reusing the simulation model code. An alternative approach is to build a model-based system using MATLAB Simulink, which allows direct conversion from the simulation model to the Hardware Description Language (HDL) and software code.

1.1 Motivation

This study's primary purposes are enabling fast hardware and embedded prototyping with model-based system design and measuring ARINC-664 ES delay statistics for various scheduling algorithms.

Delay is a critical Quality of Service (QoS) parameter in network systems, and choosing the scheduling algorithm in a network system has a significant impact on this parameter. Therefore, this study examines and compares the delay performance of four ARINC-664 ES scheduling algorithms. An ARINC-664 ES model and a simulation environment must be built to achieve this. Using a discrete event simulator is a good choice for building such a system.

MATLAB Simulink, for such purpose, provides fast prototyping and easy implementation with extensive Simulink libraries. Also, designers can easily manage the verification, validation, and requirement tracking of hardware and software models using Simulink libraries. Another essential property of MATLAB Simulink is that the built models can be converted to HDL and C codes by using MATLAB HDL Coder and MATLAB Embedded Coder. Due to these advantages, MATLAB Simulink is selected for implementing the ARINC-664 ES model.

In this thesis, first, MATLAB Simulink models for general purpose D/D/1 and M/D/1 queues are built and their latency performances are calculated and theoretically verified. These models are extended to build an HDL convertible ARINC-664 ES model with various scheduling algorithms such as the Smallest BAG (SB), Longest Queue (LQ), Smallest Size (SS), and First-In-First-Out (FIFO). Finally, an advanced

ARINC-664 ES model, which provides an infrastructure for dynamically switching among different scheduling algorithms in run-time, is presented.

1.2 Contribution

The first contribution of this thesis is to build a model-based simulation design of ARINC-664 ES traffic regulator and generate the necessary HDL and C codes for a System on Chip (SoC) implementation from this simulation design rather than using traditional methods, i.e., building the entire design from scratch. To the best of our knowledge, there is no other model-based SoC implementation of ARINC-664 ES traffic regulator in the literature.

The second contribution is to measure the mean, standard deviation and maximum of jitter of various scheduling algorithms for ARINC-664 ES and compare the results. In this regard, the SB, SS, LQ, and FIFO algorithms are implemented, and their performances are compared.

The third and last contribution is to build a dynamic scheduler for ARINC-664 ES. In the dynamic scheduler, the user can switch among the scheduling algorithms during run time based on the outcomes of the statistical performance. Dynamic scheduler consists of simulation modules that include network traffic generators, Programmable Logic (PL) implementation, including traffic shaper and scheduling algorithms, and Processing Subsystem (PS) implementation, which provides statistic calculators.

1.3 Organization of Thesis

This thesis is organized into seven chapters, including the Introduction chapter. Chapter 2 gives detailed information about ARINC-664, queueing theory, scheduling, and tools used in this thesis. Chapter 3 reviews the literature by examining similar works in this field. Chapter 4 presents the Simulink-based Single Queue model and ARINC-664 ES model with various scheduling algorithms. Chapter 5 presents the details of the ARINC-664 ES Dynamic Scheduler model. Chapter 6 presents the theoretical simulation and implementation results for each scheduling algorithm. Finally, Chapter 7 concludes the thesis.

2. BACKGROUND

In this chapter, first, the concepts of ARINC-664 are explained. Second, queueing theory and its basic concepts are presented. Third, the general and ARINC-664 ES-specific scheduling algorithms are overviewed. Finally, the related tools are explained.

2.1 ARINC-664

Recently, the necessity for reliable and predictable network services in many industries has emerged. Deterministic networks provide solutions to these necessities with bounded latency on a per-deterministic-flow basis and guaranteed low delay variation (jitter) on each flow. Deterministic networks are mainly used in real-time applications, video streaming, avionics, and automation technologies.

The amount of data that needs to be transferred in avionics systems increases day by day. Due to their low bandwidth, traditional deterministic network protocols such as MIL-STD-1553 and ARINC-429 cannot manage this increase. Some of the State-of-the-art deterministic network protocols are TTEthernet, TSN, and ARINC-664.

ARINC-664, also known as Avionics Full-Duplex Switched Ethernet (AFDX), is a safety-critical avionics network communication protocol that guarantees a bounded end-to-end delay and jitter, provides redundancy and QoS. There are three main elements in the ARINC-664 network: VL, ES, and Switch.

ARINC-664 is a fault-tolerant deterministic network protocol. It provides redundant communication on both the network and node levels. An example network topology with redundant communication of ARINC-664 is shown in Fig. 2.1. End System 1 and End System 2 communicate through Switch A and Switch B in this topology. An End System sends the same frame to Switch A and Switch B to ensure that if one of the Switches fails, the other can still complete the transfer.

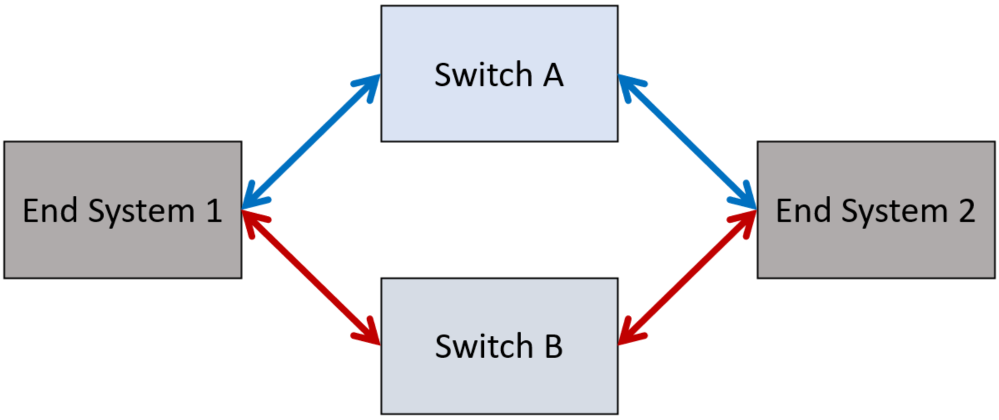
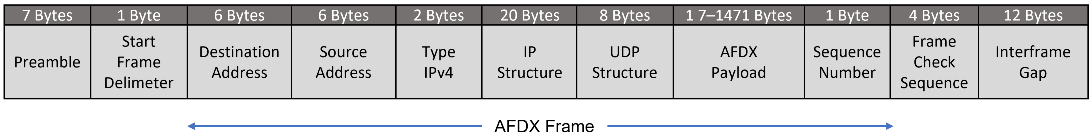


Figure 2.1 : ARINC-664 network topology with redundant communication.

2.1.1 Virtual link

In ARINC-664, frames are exchanged through unidirectional connection from one source to one or more destinations using logical communication channels called VLs. Each frame has a 16-bit unique VL ID that can be used to transfer incoming traffic flow to its logically separated VL queue. VL ID information is stored in the destination address of the MAC. The maximum number of VLs that each ES can use is 128. A general structure of ARINC-664 frames is shown in Fig. 2.2.



7 Bytes	1 Byte	6 Bytes	6 Bytes	2 Bytes	20 Bytes	8 Bytes	1 7–1471 Bytes	1 Byte	4 Bytes	12 Bytes
Preamble	Start Frame Delimeter	Destination Address	Source Address	Type IPv4	IP Structure	UDP Structure	AFDX Payload	Sequence Number	Frame Check Sequence	Interframe Gap

Figure 2.2 : ARINC-664 frame structure.

Each VL's minimum and maximum frame size is determined by the L_{min} and L_{max} parameters, respectively. L_{min} limits the smallest and L_{max} limits the largest Ethernet frame size that can be transmitted over the corresponding VL. In ARINC-664, each VL has a BAG parameter. BAG parameter specifies the frequency of transmission and smooths the burst frame traffic by using the leaky bucket algorithm for each queue. AG parameters are set as the multiple of two from 1 millisecond (ms) to 128 ms (1, 2, 4, ... 128 ms). Fig. 2.3 demonstrates how the traffic shaping regulates the unregulated incoming traffic using the BAG parameter for a single queue, i.e., VL.

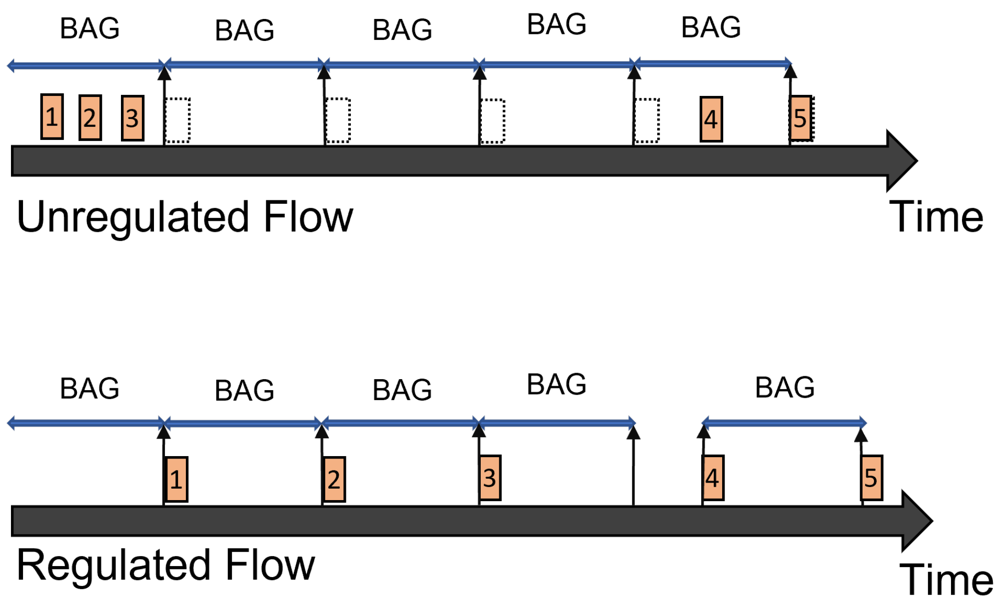


Figure 2.3 : Working principle of traffic shaping for a single queue.

2.1.2 End system

ARINC-664 ES is one of the main components of the ARINC-664 network protocol. ESs exchange data between each other through Switches. ESs aim to guarantee secure and reliable data exchanges to the partition software. Each ES incorporates traffic shaping and scheduling mechanisms to ensure that the bandwidth utilization of each traffic flow conforms with its contracted limit and that each frame is delivered to the transmission medium in a timely fashion, respectively. ARINC-664 ES consists of two parts, the receiver and transmitter. The main elements of the transmitter are the VL FIFOs, Traffic Regulator, and Redundancy Manager, while the main elements of the receiver are the Integrity Checker and Redundancy Checker. In the transmitter, incoming Ethernet frames from the upper layer are stored in distinct VL FIFOs (queues) based on their VL IDs. Traffic Regulator regulates incoming Ethernet frames for each queue. Redundancy Manager duplicates Ethernet frames to provide reliable communication. It also tags the order of each Ethernet frame with a sequence number. This number can be between 0 and 255. In the receiver, the Integrity Checker module checks the sequence number of each Ethernet frame to ensure that the incoming frames are properly ordered, and the Redundancy Checker module allows the first arriving frame among duplicated frames and drops the other one. Then, the frames are sent to the upper layer. The structure of the ARINC-664 ES transmitter is shown in Fig. 2.4 and the structure of the ARINC-664 ES receiver is shown in Fig. 2.5.

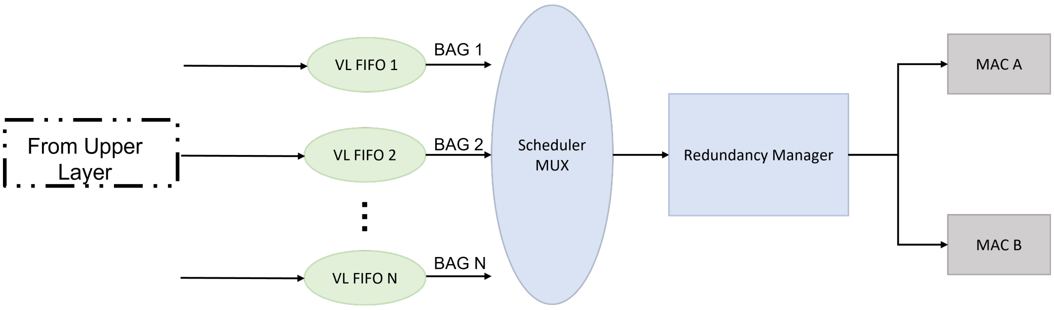


Figure 2.4 : Structure of the ARINC-664 ES transmitter.

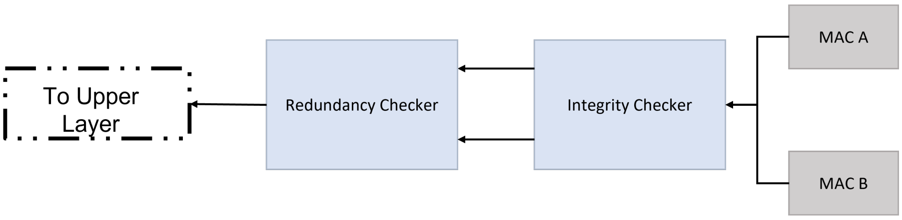


Figure 2.5 : Structure of the ARINC-664 ES receiver.

The ideal behavior of Fig. 2.3 cannot be achieved since ARINC-664 ES has multiple VLs served over only one output link (i.e., server). Considering that there are multiple VL queues and more than one queue can have frames ready to be served, a scheduler mechanism must be used to decide which queue to be served next. The scheduling algorithm results in the jitter, and the objective is to utilize a scheduling algorithm that can yield a minimum jitter. Fig. 2.6 demonstrates the concept of jitter in ARINC-664 ES.

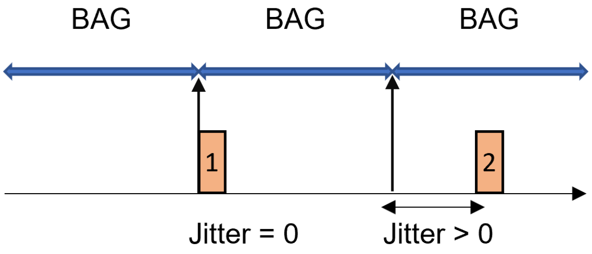


Figure 2.6 : The demonstration of jitter as a result of the scheduling conflict.

2.1.3 Switch

ARINC-664 Switch is one of the main components of the ARINC-664 network protocol. Each Switch is responsible for forwarding the incoming VL frames to their destination ports while enforcing timing, filtering, and policy requirements, limiting fault propagation, and ensuring time determinism. The filtering function is responsible for dropping the invalid frames that are corrupted or whose size is not in the L_{min} - L_{max} range of the corresponding VL. The policing function is responsible for applying the frame-based or byte-based token bucket algorithm to guarantee that any of the frames that violate the allocated bandwidth will be dropped. The forwarding function is responsible for directing the incoming frames to the corresponding destination ports according to the configuration table.

2.2 Queueing Theory

Network applications require complex queues with various algorithms. However, simplifying the application is beneficial when the aim is to analyze a complex queue [8]. The queueing theory aims to analyze queue systems. A commonly used notation for queue systems is called Kendall notation [9]. In this notation, arrival process, service distribution, number of servers, and the buffer size can be notated as follows:
[arrival process]/[service distribution]/[number of servers]/[buffer size]-[queue discipline]
The arrival process defines how an independent source loads the queue, and service distribution defines the policy for serving queues. The most common notations for arrival process and service distribution are M, D, and G. M defines the Markovian-Poisson or exponential process, D defines the deterministic process, and G defines the general process. The number of servers defines how many servers exist to serve the queue s. Buffer size defines the number of buffer spaces available in the queues. If no value is specified, the buffers are assumed unlimited. Scheduling policy defines the deciding mechanism of which queue to serve next. Queue discipline defines the serving order, e.g. FIFO, LIFO, and Processor Sharing. Fig. 2.7 represents a queueing system. In the figure, enqueuing means filling the queue and dequeuing emptying the queue. An important queueing system parameter is utilization, denoted

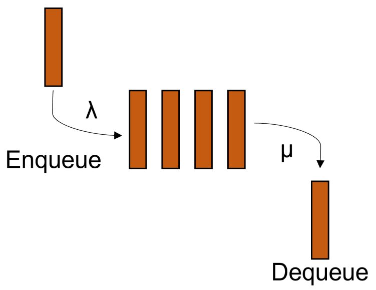


Figure 2.7 : Queueing system.

as U. Utilization can be defined as the number of busy servers divided by the total servers for a simulation time. In the case of a single server, it can be defined as the amount of time the server is busy divided by the entire simulation time. From the user's perspective, higher utilization means better performance. Let μ denote the service rate, and λ denote the arrival rate. If

$$\mu > \lambda \tag{2.1}$$

the queue is stable. However, if

$$\mu < \lambda \tag{2.2}$$

the queue will overflow eventually; hence, it is not stable. Utilization can be defined as

$$U = \mu / \lambda \tag{2.3}$$

Another important queueing system parameter is Little's Formula. It is used for predicting the average number of items in a stationary queueing system. The definition of this formula is

$$L = \lambda \cdot W \tag{2.4}$$

where L defines the number of items in a queue, and W defines the average waiting time an item spends in the queueing system.

Example simulation tables for two Kendall notations are set below. Table 2.1 shows an example queueing scenario for a D/D/1 queue. This notation means the arrival rate and the service rate of the queueing system are deterministic. Table 2.2 shows an example queueing scenario for an M/D/1 queue. This notation means that the arrival rate of the queueing system is exponential, and the service rate of the queueing system is deterministic. In each notation, the buffer is unlimited, and there is only a single server; hence there is no scheduling policy. These tables will be used to verify the queueing systems in the next chapters.

Table 2.1 : Simulation table for a D/D/1 queue [8].

Arrival Time	Service Duration	Queue Size on Arrival	Service Starts	Service Ends	Delay
1	4	0	1	5	4
5	4	0	5	9	4
9	4	0	9	13	4
13	4	0	13	17	4
17	4	0	17	21	4
21	4	0	21	25	4

Table 2.2 : Simulation table for an M/D/1 queue [8].

Arrival Time	Service Duration	Queue Size on Arrival	Service Starts	Service Ends	Delay
1	4	0	1	5	4
3	4	1	5	9	6
4	4	2	9	13	9
12	4	1	13	17	5
17	4	0	17	21	4
18	4	1	21	25	7

2.3 Scheduling

Scheduling is a crucial concept to improve the Quality of Service (QoS) in many applications. Generally, a scheduling policy aims to reduce and bound jitter. It also can aim at fairness (e.g., Jain's index [10]) or maximizing the throughput. Some traditional scheduling algorithms are the FIFO Scheduling, Earliest Deadline First (EDF) Scheduling, Shortest-Job-Next Scheduling, and Round Robin (RR) scheduling.

Generalized Processor Sharing (GPS) [11] behavior is an ideal approach to provide fair scheduling in network systems. Even though GPS has an ideal behavior, it is not implementable since multiple queues are ready to be served, but only one server serves them. Instead, GPS-like approaches have been proposed by researchers in recent years. Some implementable fair queueing based scheduling algorithms are Weighted Fair Queueing (WFQ) [12], Weighted Round Robin (WRR) [13], and Worst-Case Weighted Fair Queueing+ (WF2Q+) [14], [15]. WRR is an extension of the well-known RR scheduling algorithm, which rounds each queue from top to bottom or bottom to top.

It differs from RR with the ability to assign different frame weights to different queues. An alternative approach to WRR is Deficit Round Robin (DRR) algorithm. Contrary to WRR, DRR assigns byte-based weight, i.e., quantum value, to each queue.

A relatively new defined scheduling algorithm concept is virtual clock-based scheduling. In literature, there are many virtual clock-based scheduling [16]. Among them, WF2Q+, an extension of the Worst-Case Weighted Fair Queueing (WF2Q), is the most popular scheduling algorithm.

Fair queueing algorithms are considered to be useful in next-generation mixed-critical ARINC-664 Switch technologies. [17] suggests that complex scheduling algorithms can be considered in mixed-critical applications and [18] proposes a hierarchical scheduling structure including Strict Priority (SP) and WF2Q+. Fair queueing scheduling algorithms are beneficial for improving the QoS of network systems. However, these algorithms are not suitable to the ARINC-664 ES because the BAG parameter regulates the incoming Ethernet traffic; in other words, the BAG parameter assigns weights for each queue. ARINC-664 standard does not specify any scheduling algorithm. However, there are some studies for possible scheduling algorithms in ARINC-664 ES. These algorithms are the RR, SB, FIFO, SS, and LQ.

The RR is a common scheduling algorithm. It chooses a VL to serve among all the VLs in some rational order, usually from top to bottom and then starting again at the top. The Smallest BAG scheduling algorithm serves the VL with the smallest BAG value among all VLs. In the ARINC-664 standard, there are eight possible BAG parameters for each queue. So, some of the VLs can have the same priority in the case of multiple VLs. The FIFO algorithm serves the VL whose Head Of Line (HoL) arrival time is the smallest among all the queues. Under unregulated Ethernet traffic, this algorithm does not specify any priority or pattern among queues. The SS algorithm serves the VL whose HoL frame is among all VLs. The LQ algorithm serves the VL with the highest number of bytes among all VLs.

2.4 Related Tools

MATLAB [19] is a numeric computing and programming platform for engineers and scientists. It performs complex mathematical computations and provides extensive libraries with a model-based design environment. Also, MATLAB offers many products for converting hardware or software systems, such as Embedded Coder, C/C++ Converter, Xilinx Model Composer, and HDL Coder.

Simulink [20] is a MATLAB-based graphical programming environment for simulating, modeling, and analyzing systems. It is used in many fields, such as digital signal processing and control theory. It offers many libraries and tools for several development environments; some can operate with other third-party tools. Simulink provides state machine templates, large libraries, functions, and requirement tracker tools to evaluate performance, create design tests, and build prototypes.

Building a Register Transfer Level (RTL) design with traditional methods consumes a lot of time, and a verification environment must be created from the earliest stages to ensure that system works properly. MATLAB HDL Coder [21] is an alternative RTL development method to avoid the drawbacks of traditional RTL design developments with HDLs [22], [23]. It generates synthesizable Very High Speed Integrated Circuit Hardware Description Language (VHDL) or Verilog HDL files using Simulink blocks and MATLAB functions. MATLAB HDL Coder can generate HDL codes with or without clock enable, synchronous and asynchronous reset. A designer can select the target device and operation frequency. It also proposes more complex options such as pipelining.

MATLAB Embedded Coder [24] generates readable C/C++ codes for embedded processors of various device vendors such as ARM, Intel, and STMicroelectronics. The user can aim to be efficient, whether in memory or speed usage. Generated C codes contain necessary header files, Simulink function files, and a main file. The main file is called "`ert_main.c`". Ert is the abbreviation of "Embedded Real-Time". This main function has three subfunctions: Initialize, OneStep, and Terminate. The Initialize function assigns the initial value of all the parameters of the C code. The OneStep function executes, and the Terminate function terminates the Simulink model. Fig.

2.8 shows the working principle of MATLAB HDL Coder and MATLAB Embedded Coder.

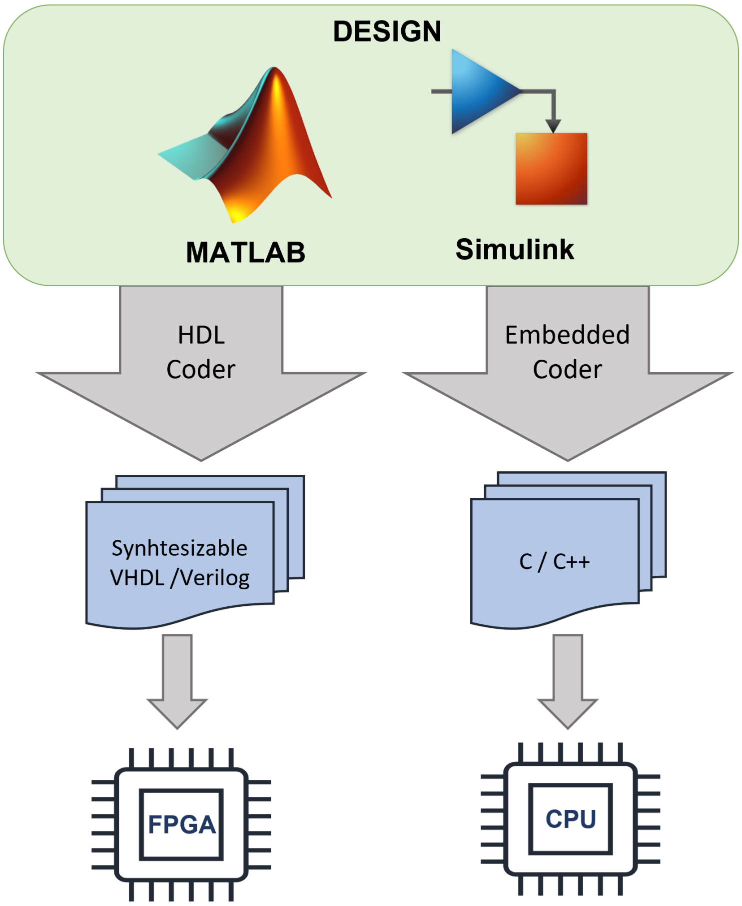


Figure 2.8 : Working principle of MATLAB HDL Coder and Embedded Coder.

Vivado Design Suite [25] is a software program produced by Xilinx for simulation, synthesis, and implementation of HDL based designs on FPGA. On top of that, Vivado offers a C-based High Level Synthesis (HLS) platform and an SoC development environment. The first version of Vivado, the sequel to ISE, was released on April 2012. Vivado can be used in both Graphical User Interface (GUI) and batch mode with Tool command language (Tcl). It supports 7 Series or newer FPGAs.

Xilinx Software Development Kit (SDK) [26] is an Eclipse-based Integrated Design Environment (IDE) for programming Xilinx's embedded processors such as Zynq devices and Microblaze softcore. Xilinx SDK works integrated with Xilinx Vivado. It provides Board Support Package (BSP) libraries that can control custom RTL designs and Xilinx Intellectual Property (IP) cores. Also, it brings together homogenous and heterogeneous multi-processor designs.

3. LITERATURE REVIEW

Many works are related to MATLAB HDL Coder and Embedded Coder, ARINC-664 simulation and hardware implementation, and performance evaluation of ARINC-664 scheduling algorithms in the literature. In this chapter, these works are reviewed.

3.1 MATLAB HDL Coder and Embedded Coder Applications

In literature, Simulink, Embedded Coder, and HDL Coder are widely used in many applications such as image and video processing [27], controlling of power systems [28] and machine learning [29].

A Wireless Communication SoC with both hand-written HDL code and MATLAB HDL Coder is built in [30]. Then, the utilization results are compared, and the code readability and development time is examined. This work shows that MATLAB HDL Coder spends fewer resources, its code is readable if the designer respects the concepts of the tool, and less time in development was spent than hand-written code.

Digital filter systems are built using three methods including hand-written RTL code, Vivado High Level Synthesis (HLS), and MATLAB HDL Coder, and their performances are compared in several aspects such as area optimization, latency, throughput optimization, and timing optimization in [31]. This work claims that HLS in FPGA development decreases the amount of time spent on product development cycles. It also claims that the area, throughput, latency, and timing objectives could be met easier with Vivado HLS compared to MATLAB HDL Coder. In addition, it is mentioned that the MATLAB Simulink environment provides many graphical libraries and block diagrams, especially in the field of control, signal processing, image processing, etc.

MATLAB codes are converted to C by using MATLAB Embedded Coder in [32]. Then, the pros and cons of Embedded Coder are stated. Generally, the pros are that the structure of the code is good, and variables and comments are placed correctly. The cons are that some names are strange, and the code readability is not very good.

3.2 Model-Based ARINC-664 Design

In [33], a model-based ARINC-664 simulation environment is built to measure the performance of the network system under various traffic situations by using OMNET++, a discrete event simulator. This work presents the worst-case end-to-end delay results of the ARINC-664 network. However, this work only builds the model for simulation purposes.

Modeling and simulation of the ARINC-664 network system is built by using OPNET in [34]. It builds the ES and Switch components of the ARINC-664 and simulates and compares the end-to-end delay, delay jitter, and packet loss rate results of the ARINC-664 network with traditional Ethernet. The results show that the ARINC-664 provides better performance. However, this work only builds the model for simulation purposes.

3.3 FPGA Implementation of Scheduler Algorithms and ARINC-664

The FPGA is claimed to be a good choice for implementing scheduling algorithms in [35] due to its high speed and reconfigurability. Then, it sorts some scheduling approaches according to some parameters such as simulation delay, resource utilization, and arbitration completion speed. In conclusion, this work represents a positive attitude about the FPGA development of scheduling algorithms in network switches. However, it states that more research is necessary in this field.

An FPGA-based dynamic scheduling system is presented in [36]. The presented system is capable of switching among Deficit Weighted Round Robin (DWRR) and WF2Q+ scheduling algorithms by using partial reconfiguration. This research claims that switching time in partial reconfiguration, which is a critical parameter for these systems to work efficiently, is negligible for both algorithms. Also, FPGA resources for both algorithms are presented.

Moving the software components of ARINC-664 to FPGA to mitigate the possible Single Event Upset (SEU) situations is aimed in [37]. This work concludes that the FPGA is a good candidate for ARINC-664 implementation. It also declares that sampling queues are easier to implement in FPGA, and FPGA can provide robustness against SEU problems.

ARINC-664 ES is implemented on FPGA in [38]. It provides the architecture details and resource utilization on both transmitter and receiver. Also, it provides the details of Direct Memory Access (DMA) technology and Peripheral Component Interface Express (PCIe), which are used for full-duplex data transferring between software and hardware. Then, it represents a reconfigurable system that can switch the BAG parameter of each VL via PCIe in run-time.

The possible ARINC-664 implementation solutions are discussed in [7]; processor-centric and hardware-centric. The entire design, except the physical interfaces, is located in Processing Subsystem (PS) in the processor-centric solution. The ARINC-664 protocol is implemented in the hardware below the Internet Protocol Layer (IP Layer), while the IP layer and above was implemented in the embedded processor in the hardware-centric solution. Also, it provides some suggestions for DO-254 and ARINC-664 approaches and briefly explains the concepts of ARINC-664 and similar network protocols.

3.4 ARINC-664 Scheduling Performance Analysis

ARINC-664 jitter and delay performance are evaluated in both ES and network domain under different scheduling and shaping algorithms and topologies [39], [40], [41].

In [42], the jitter-EDD(Earliest-Due-Date)'s scheduling algorithm is used in both source and destination hosts to provide a more reliable network system for aircraft. A simulation scenario with SimEvents, a MATLAB application for building discrete event simulators [43] is used, which includes 9 Switches and 13 ESs. This work shows that the end-to-end delay is much smaller with the jitter-EDD mechanism.

The performance of ARINC-664 ES scheduling algorithms (RR, SB, LQ, SS, and FIFO) on the ARINC-664 network are examined in [44]. Simulations are run on OMNET++. The maximum jitter, mean jitter, and the percentage of frames whose jitter is more than 500 microseconds (us) are presented in this research. The results show that the LQ and FIFO algorithm has the best performance in terms of low jitter.

Therefore, this research suggests the FIFO and LQ as the best scheduling algorithms in ARINC-664 ES.

A new shaping methodology for the FIFO scheduling algorithm, which provides uniform delay, is offered in [45]. This work claims that delay jitter significantly decreases, but the delay slightly increases on the low priority. This work has proved that this method will drastically reduce delay jitter with a slight increase of delay on the low priority queues.

In [46], BAG-based and rate-based scheduling algorithms are compared using the Network Calculus and Response Time Analysis. An example configuration scenario is set based on 4 VLs, and the results for the mean, standard deviation, and distributions of jitter are presented. This research claims that BAG-based scheduling is the optimal scheduling policy for ARINC-664 ES.

The SS algorithm, i.e., Smallest Frame Earliest (SFE) algorithm, is examined and compared with Largest Frame Earliest (LFE) and random priority assignment in [47]. It theoretically proves that the SFE provides the minimum average jitter using a 4 VL scenario.

4. END SYSTEM SCHEDULER MODELS

Since time-to-market is an important objective, developers need powerful tools for the rapid prototyping of highly complex systems. Model-based system design is presented as a solution for rapid prototyping of the ARINC-664 ES.

In this chapter, first, the Single Queue model built using MATLAB Simulink is described. Then, the structure of the ARINC-664 ES model is described.

4.1 Single Queue Model

As described in Chapter 2, simplifying a queue-based system allows queuing theory to be used to analyze that system. To fulfill this purpose, in this section, a Simulink model of a Single Queue system that is responsible for generating, storing, and serving Ethernet frames with varying lengths and data rates is presented. Fig. 4.1 demonstrates the architecture of this model consisting of the *Loader*, *Memory*, *Server*, and *Analysis* modules.

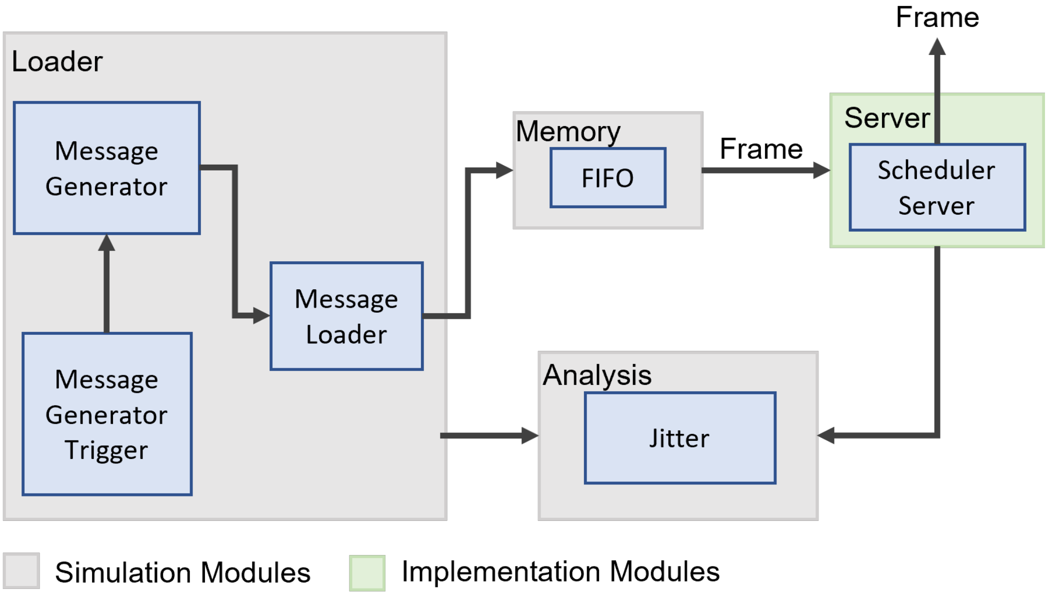


Figure 4.1 : Simulink model of the Singe Queue.

The *Loader*, *Memory*, and *Analysis* modules are used solely for simulation purposes, whereas the *Server* module is used for the FPGA implementation. The primary purpose of this model is to verify the *Loader* module in prior because the ARINC-664 ES model is more complex and hard to verify. Considering that the *Loader* module will be used for feeding the queues to evaluate the performance of the hardware convertible scheduling algorithms, it is clear that the proper functioning of this module is of great importance. In the *Loader* module, the $MessageGeneratorTrigger$ block determines the message generation time according to the traffic model. This block can generate frames in the deterministic or exponential arriving processes. The $MessageGenerator$ block creates ARINC-664 frames, which contain VL ID information in its MAC destination and the length information at the beginning of the frame. The Message *Loader* loads Ethernet frames to the memory emulating FIFO by sending 1 byte at each step. The *SchedulerServer* block in the *Server* module is responsible for serving Ethernet frames using the link capacity information. This block, first, reads the length information at the beginning of the frame and then reads 1 byte at each step until the entire frame is read. The service rate of the *SchedulerServer* block is 1 Gbit/s. The moments that events occurred are stored in the *Analysis* module to calculate performance measures at the end of the simulation.

Experiments are performed for D/D/1 and M/D/1 queuing models, and the simulation model results are compared with theoretical calculations of the queuing theory by using Table 2.1 and Table 2.2. In the *Analysis* module of the Single Queue model, the aim is to calculate the mean jitter values of each queue. The jitter value must be calculated for each frame of each queue to achieve this. Storing each arrival and departure time of frames for jitter measurement is not applicable because it would require excessive memory usage and downgrade the simulation speed. Instead, service time can be subtracted from the arrival time for each frame, and the mean value of these subtractions can be considered as the mean jitter. Even though this method works for D/D/1 queues, two or more frames can arrive before any of them is served in case of an M/D/1 queue. So, the subtraction operation in run-time is also not implementable for M/D/1 queues.

An alternative approach to performing both D/D/1 and M/D/1 measurements is to calculate the average jitter of arrival time and average jitter of service time independently at run-time. These two values are subtracted from each other at the end of the simulation to find the minimum jitter. The formula for mean jitter calculation is shown below:

$$E[U] = E[Y] - E[X] \tag{4.1}$$

where U denotes the jitter, Y denotes the service time, X denotes the arrival time, and E denotes the expected value, i.e., mean.

Single Queue model for D/D/1 and M/D/1 is simulated, and its statistics are compared with the theoretical statistics based on Table 2.1 and Table 2.2. Table 4.1 and Table 4.2 show the example scenario, and compare theoretical results with simulation results for D/D/1 and M/D/1 queues, respectively.

Table 4.1 : D/D/1 simulation table

Arrival Rate	Length (Byte)	Service Rate	Theoretical Mean	Simulation Mean	Number of Frames
990 Mbit/s	100	1 Gbit/s	0	32	1237500

In case of an arrival rate of 990 Megabit per second (Mbit/s) and service rate of 1 Gigabit per second (Gbit/s) in D/D/1, the queue must be empty 1 percent of the time and loaded with only one message 99 percent of the time. So, the mean jitter must be 0 in theory. In the simulation, the mean jitter value is 32 nanoseconds (ns) because it takes 4 cycles, each cycle is 8 ns, for the scheduler to trigger once the queue starts to fill. These 4 cycles can be described as overhead, independent of the frame length.

Table 4.2 : M/D/1 simulation table

Arrival Rate	Length (Byte)	Service Rate	Theoretical Mean	Simulation Mean	Number of Frames
990 Mbit/s	100	1 Gbit/s	33.4950	91.96	1237500

In case of an arrival rate of 990 Mbit/s and a service rate of 1 Gbit/s in M/D/1, the queue can be filled with more than one frame at some of the simulation time. In theory, the mean jitter value is 33.4950 ns. In practice, the mean jitter value is 91.96 ns. 32 ns is the overhead, independent of the frame length. The rest of the time (59.96 ns) is the mean jitter value of the simulated M/D/1 model.

4.2 ARINC-664 End System Model

ARINC-664 ES model is built in Simulink to implement and analyze the scheduling algorithms. It is an extension of the Single Queue model, which we described in the previous section. Fig. 4.2 demonstrates the logical structure of the ARINC-664 ES model consists of the *Loaders*, *Memory*, *Analysis*, and *Server* modules. In the figure, N represents the number of VLs. The *Loaders*, *Memory*, and *Analysis* modules are used for simulation, whereas the *Server* module is used for the FPGA implementation. In the ARINC-664 ES model, each queue must be filled independently. To achieve this, the *Loaders* module, which includes multiple *Loader* blocks, is added to the simulation model. Each *Loader* block can have a deterministic or exponential arriving process. Note that each *Loader* block of the ARINC-664 ES model is a copy of the *Loader* module of the Single Queue model. Therefore, the *Loader* block of the ARINC-664 ES model is already verified. The *Memory* module contains independent FIFOs for each VLs. The *Server* module includes the *Shaper*, *SchedulerDecider*, *EligibleQueues*, and *Scheduler* blocks. The *Shaper* block implements the leaky bucket algorithm. Its primary purpose is to regulate the incoming Ethernet frame traffic as described in Fig. 2.3. The *EligibleQueues* block keeps track of the eligibility of each VL based on the

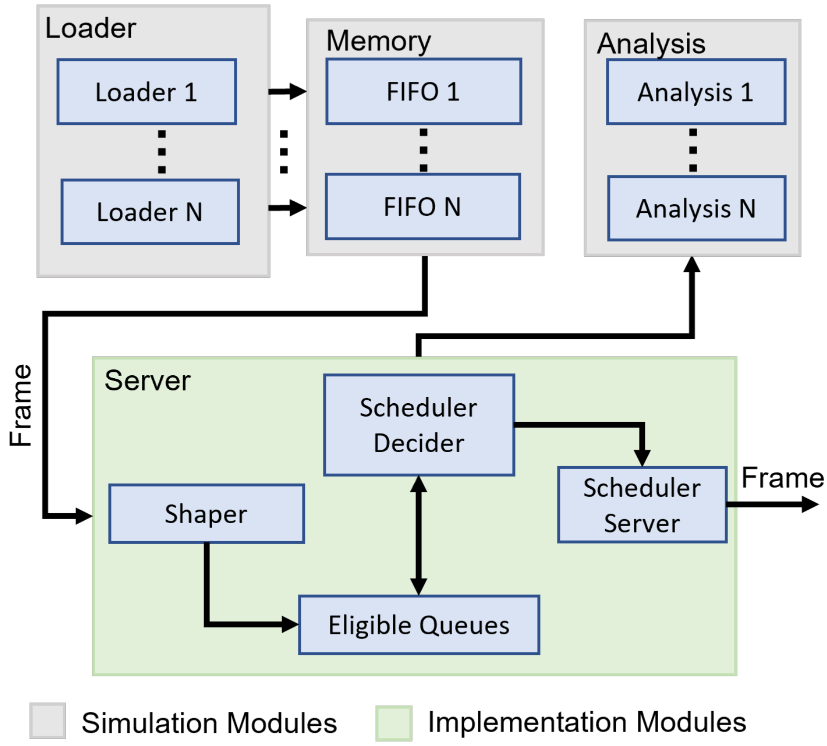


Figure 4.2 : Simulink model of the ARINC-664 ES

leaky bucket and the scheduling algorithm. A queue becomes eligible if it is ready for being served and remains eligible until it is decided that it will be served. The *SchedulerDecider* block determines which VL must be served among eligible queues according to the scheduling algorithm. The *SchedulerServer* block serves the Ethernet frame from one of the queues according to the outcome of the *SchedulerDecider* block. Its operation speed is 1 Gbit/s. In the *SchedulerServer* block, as in the Single Queue model, the length information is read from the first two bytes of each frame. Then, the *SchedulerServer* block reads the entire frame based on the length information by serving 1 byte at each step. The *SchedulerDecider* block determines which VL must be served among eligible queues according to the scheduling algorithm

4.2.1 Implementation of the server module

The *Server* module is hardware convertible; thus, its design should pay attention to the clock and hardware limits. The proposed ARINC-664 ES model operates at 1 Gbit/s and processes 8 bits at each transaction. Therefore, the operation frequency of the *Server* module must be at least 125 MHz. The *SchedulerDecider* block is the most challenging part of the *Server* module in converting the model to HDL. The SB, FIFO, SS, and LQ scheduling algorithms are implemented in this thesis. The *SchedulerDecider* block implementation for each scheduling algorithm is presented below. For the sake of simplicity, all scheduling algorithms are implemented for 8 queues; however, they can be extended to 128 queues which is the maximum number of VLs defined by the ARINC-664 ES standard.

The SB scheduling algorithm serves the queue with the smallest BAG value among all eligible queues. Fig. 4.3 shows the structure of the *SchedulerDecider* block, and Fig. 4.4 shows the Simulink implementation of the *SchedulerDecider* block of 8 queues for SB. In Fig. 4.3, the mechanism of the SB scheduling algorithm consists of three subsystems; the $Masker$, $MinimumFinder1$, and $MinimumFinderFinal$, and two functions; $Min4$ and $Min2$. The $Min4$ function finds the minimum value among 4 inputs, and $Min2$ finds the minimum value among 2 inputs. The *Masker* subsystem aims to eliminate the queues that are not ready to be served due to the *Shaper* block. In the case of the SB algorithm, the *Masker* subsystem sets its output to the maximum 32-bits integer value if the corresponding queue is not eligible. Later, masked BAG values are compared in the *MinimumFinder*1 and *MinimumFinderFinal* subsystems. The operation frequency of the *SchedulerDecider* block is 125 MHz, so finding the minimum BAG value in one step is not an applicable method. Instead, the smallest BAG value among every 4 queues is found parallelly in the *MinimumFinder*1 by using the *Min*4 function, and the two outputs of the *MinimumFinder*1 are compared with each other in the *MinimumFinderFinal* by using the *Min*2 function. Then, the output of the *MinimumFinderFinal* is sent to the output of the *SchedulerDecider* block. If this value is the maximum 32-bit integer value, it means that none of the queues is eligible at the moment, and none of the queues should be served. So, the *SchedulerDecider* block of the SB scheduling algorithm is triggered again and operates the same functions until the output value becomes valid. When the output value becomes valid, the selected queue is served.

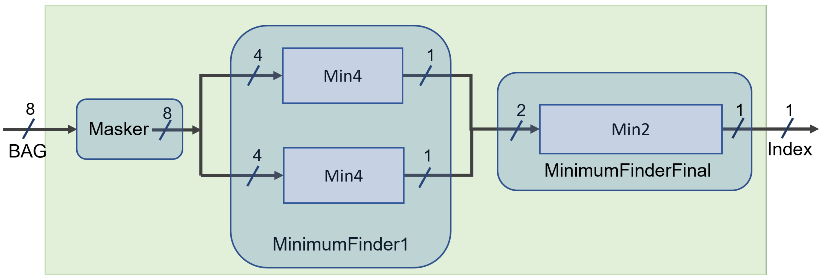


Figure 4.3 : Structure of the Scheduler Decider block of SB for 8 queues.

The FIFO scheduling algorithm serves the queue whose Head of Line (HoL) arrival time is the smallest among all eligible queues. Implementation of this algorithm follows a similar procedure as the SB algorithm. However, the HoL arrival time of each queue must be known in advance in the FIFO algorithm. A second First-Word-Fall-Through (FWFT) FIFO for each VL is placed in the *Memory* module, and the arrival time of each frame is stored inside of these FIFOs to achieve this. A 64-bit free-running time counter is used inside of the *Memory* module to obtain arrival time. Each time a frame transmission starts from the *Loaders* module to the *Memory* module, the value of the free-running counter is written to the FIFO. The FIFO scheduling algorithm mechanism is almost the same as the SB scheduling algorithm. The only difference is that the input parameters of the *Masker*, *MinimumFinder*1, and *MinimumFinderFinal* subsystems are 64-bits.

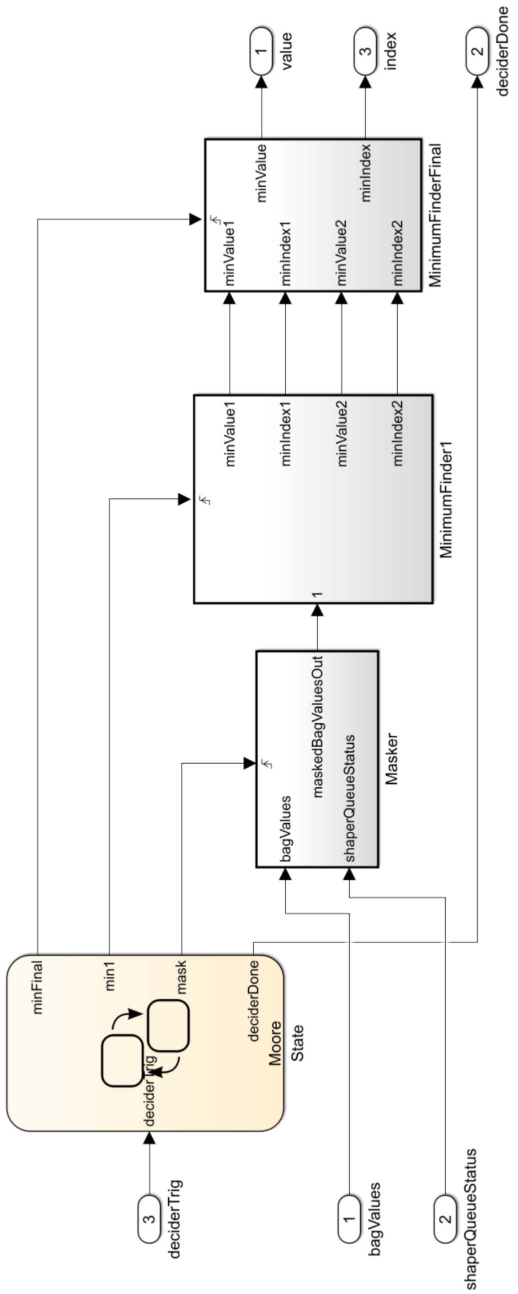


Figure 4.4 : Simulink implementation of the Scheduler Decider block of SB for 8 queues.

The SS algorithm serves the queue whose HoL frame is the smallest among all eligible queues. Implementation of this algorithm follows a similar procedure as the FIFO algorithm. However, the SS scheduling algorithm needs each queue's HoL frame length information in advance rather than the arrival time. A second FWFT FIFO for each VL is placed to the *Memory* module, and the frame length of each queue is stored inside of these FIFOs to achieve this. The mechanism of the SS scheduling algorithm is almost the same as the SB and FIFO scheduling algorithms. The only difference is that the parameters of the *Masker*, $MinimumFinder1$, and $MinimumFinderFinal$ subsystems are 16 bits.

The LQ algorithm serves the queue with the highest number of bytes among all eligible queues. Implementation of this algorithm follows a similar procedure as the SB algorithm. However, contrary to the SB, the LQ scheduling algorithm finds the maximum value among all queues. Fig. 4.5 shows the mechanism of the LQ scheduling algorithm which consists of three subsystems; the *Masker*, $MaximumFinder1$ and $MaximumFinderFinal$, and two functions; $Max4$ and $Max2$. The $Max4$ function finds the maximum value among 4 inputs, and $Max2$ finds the maximum value among 2 inputs. The *Masker* subsystem sets its output value to 0 if the corresponding queue is not eligible. The fullest queue among every 4 queues is found parallelly in the $MaximumFinder1$ subsystem by using the $Max4$ function and the 2 output values of the $MaximumFinder1$ are compared in the $MaximumFinderFinal$ by using the $Max2$ function. Then, the output of the $MaximumFinderFinal$ is sent to the output of the *SchedulerDecider* block. If this value is 0, it means that none of the queues are eligible at the moment, and none of the queues should be served. So, the *SchedulerDecider* block of the LQ scheduling algorithm is triggered again and operates the same functions until the output value becomes valid. When the output value becomes valid, the selected queue is served.

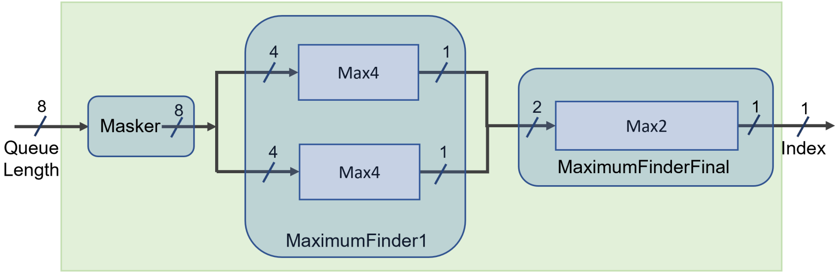


Figure 4.5 : Structure of the Scheduler Decider block of LQ for 8 queues.

As an addition, the *Server* module of the SB algorithm for 32 queues is designed. Fig. 4.6 shows the structure of the *SchedulerDecider* block, and Fig. 4.7 shows the Simulink implementation of the *SchedulerDecider* block of SB algorithm for 32 queues. In Fig. 4.6, the mechanism of the SB scheduling algorithm consists of four subsystems; $Masker$, $MinimumFinder1$, $MinimumFinder2$, and $MinimumFinderFinal$, and two functions; $Min4$ and $Min2$. The $Min4$ function finds the minimum value among 4 inputs, and $Min2$ finds the minimum value among 2 inputs. First, the $Masker$ subsystem masks the non-eligible queues in the system with 8 queues. Then, the $MinimumFinder1$ subsystem finds the smallest 8 of 32 queues in groups of 4 using $Min4$ fimction. Later, the $MinimumFinder2$ finds the smallest 2 of 8 queues in a group of 4 by using the $Min4$ function, and finally, the $MinimumFinderFinal$ finds the minimum value among 2 inputs by using the $Min2$ function. The rest of the decisions follow the same procedure as the 8 queues *Server* module.

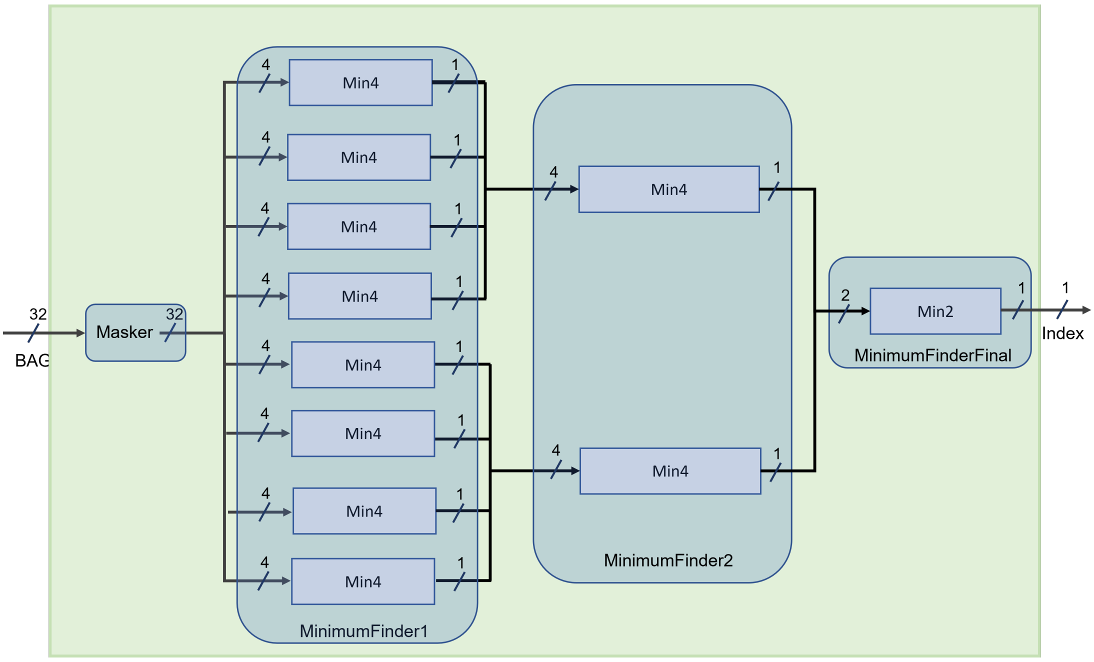


Figure 4.6 : Structure of the Scheduler Decider block of SB for 32 queues.

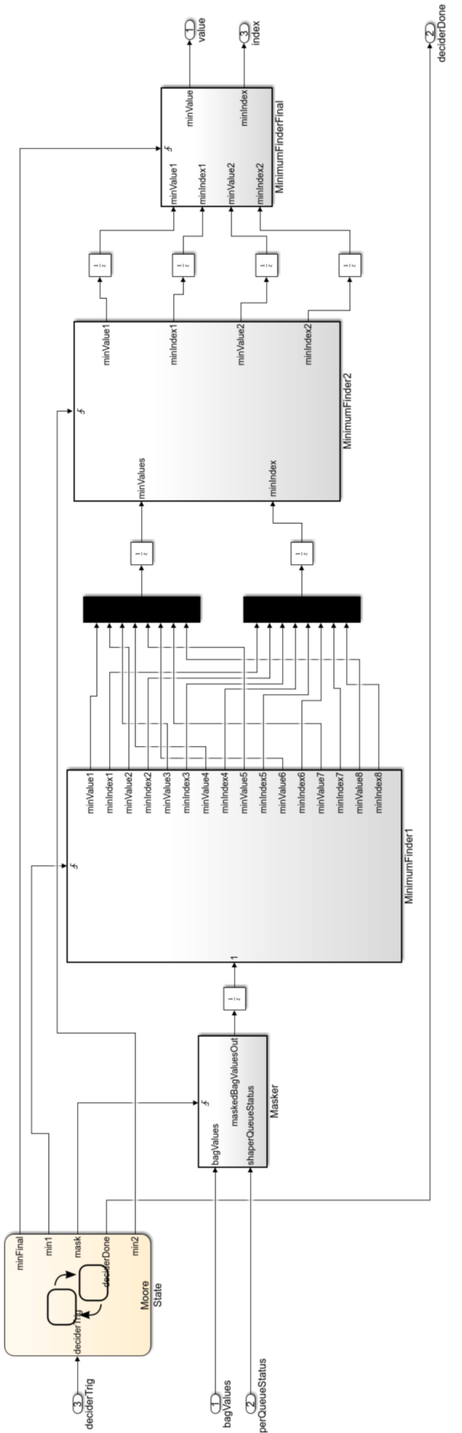


Figure 4.7 : Simulink implementation of the Scheduler Decider block of SB for 32 queues.

4.2.2 Implementation of the analysis module

The *Analysis* module is crucial for the ARINC-664 ES model because it operates statistical analysis. The analysis module measures four statistics for each queue: The mean, standard deviation, and maximum of jitter performances, and the number of served frames. The jitter is calculated for each frame on a per-flow basis to measure the first three statistics. Note that the jitter is defined as the delay between the beginning of the BAG and the date when the first bit of the frame is sent [48]. Calculating jitter value is easier in the ARINC-664 ES model than the Single Queue model since there is no need to keep track of two different values.

Storing the time difference for each Ethernet frame is not an applicable method because storing this information requires excessive memory and slows down the simulation significantly. Instead, similar to the Single Queue model, run-time mean and standard deviation calculators are used. Fig. 4.8 shows the running mean and running standard deviation calculator modules of the Digital Signal Processing System Toolbox [49]. Fig. 4.9 shows the Simulink implementation of the *Analysis* module for a single queue. In this implementation, the *JitterCounter* subsystem calculates the jitter by counting at each step, *RunMeanRunStd* subsystem recalculates the mean and standard deviation based on the new jitter value, *RunMax* subsystem calculates the maximum value, *RunFrameCounter* subsystem calculates the number of served frames, and *Logger* subsystem logs the statistics.

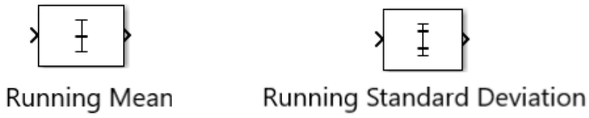


Figure 4.8 : Running mean and running standard deviation calculators.

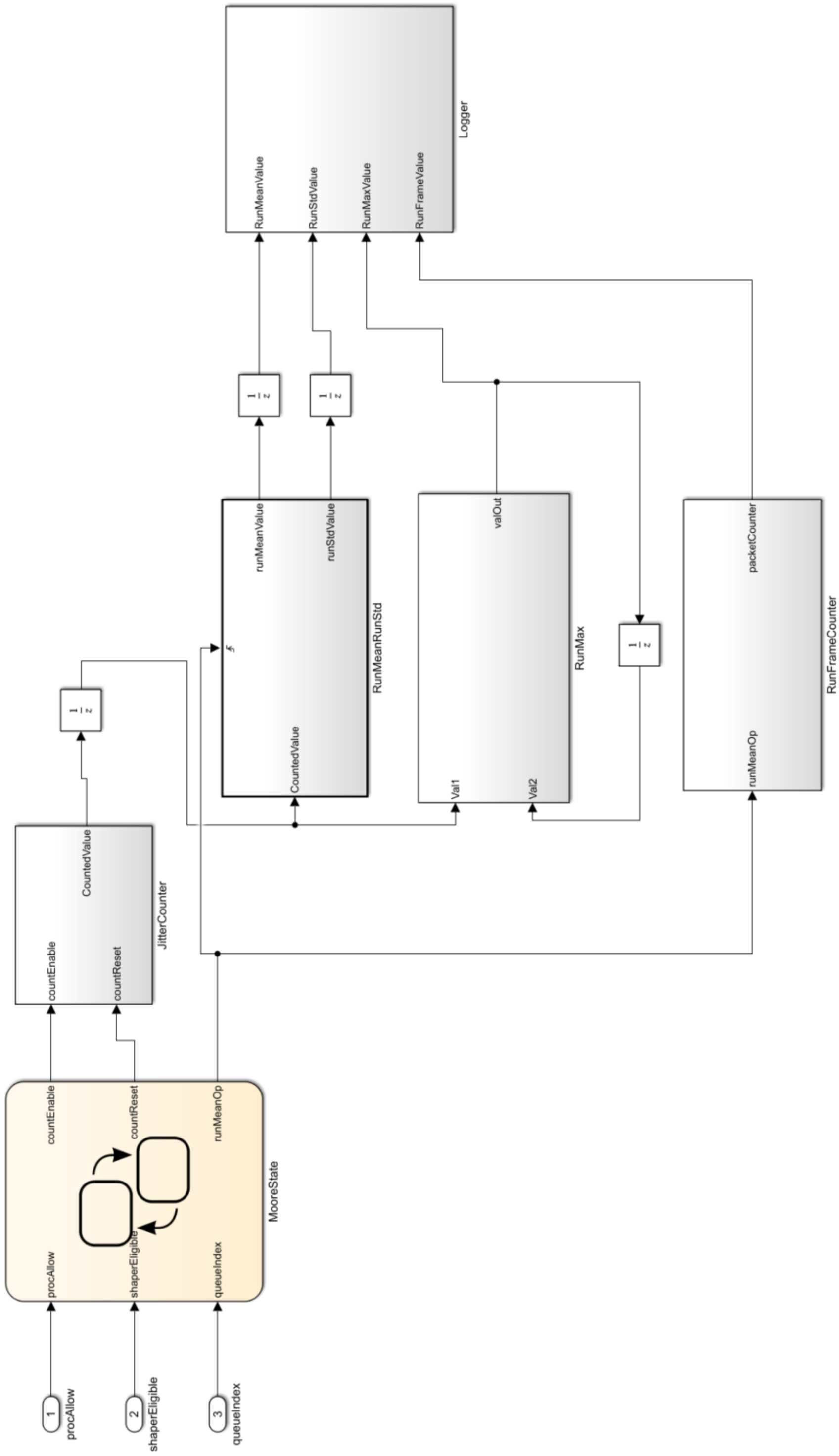


Figure 4.9 : Simulink implementation of the Analysis module in ARINC-664 ES.

5. END SYSTEM DYNAMIC SCHEDULER MODEL

In this chapter, the ARINC-664 ES Dynamic Scheduler model is presented.

ARINC-664 ES requires many configuration parameters such as VL ID, BAG, L_{min}, and L_{max}. Though today's applications mostly aim to configure the ES once according to the offline network planning when the system is on the ground, some reconfigurability studies which focus on reconfiguring the configuration parameters during run-time exist [38]. However, changing the scheduler strategy might yield significant performance improvement. ARINC-664 ES Dynamic Scheduler model, will be referred as the Dynamic Scheduler model in the rest of the thesis, is developed to take advantage of all the scheduling algorithms described in the previous chapter. Fig. 5.1 shows the structure of the Dynamic Scheduler model, which consists of the *Loaders*, *Memory*, *Analysis*, and *Server* modules.

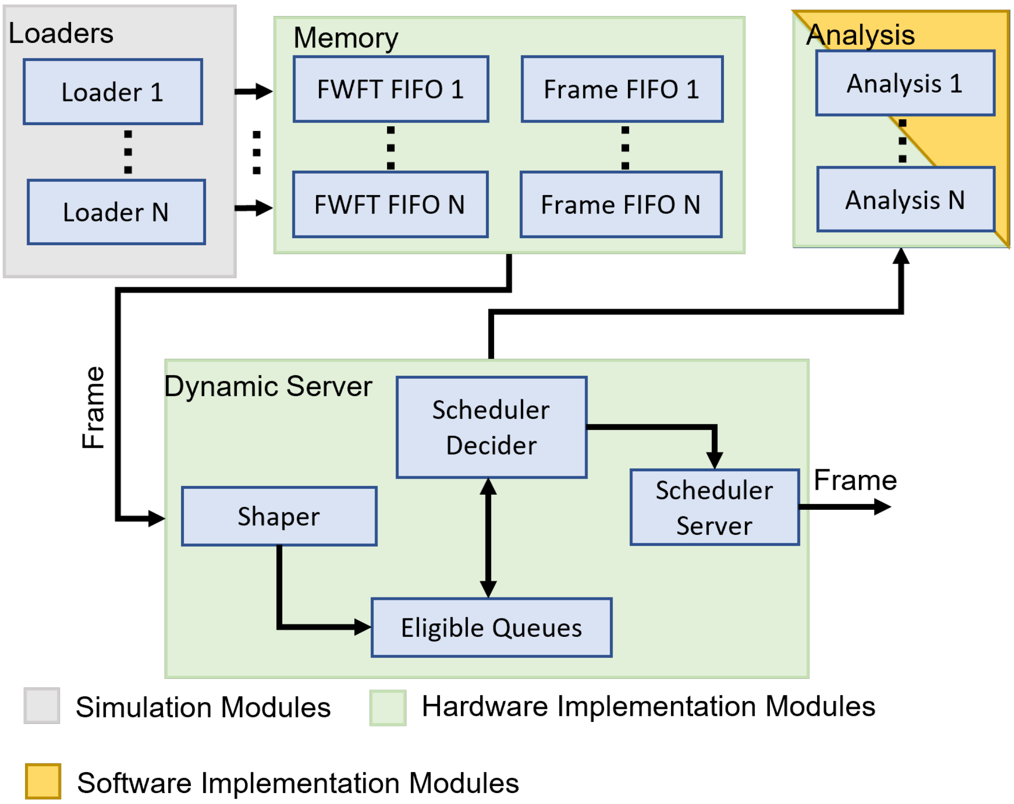


Figure 5.1 : Simulink model of the Dynamic Scheduler.

The *Loaders* module of the Dynamic Scheduler model has the same structure as in the *Loaders* module of the ARINC-664 ES model. However, the *Memory*, *Server*, and *Analysis* modules of the Dynamic Scheduler model differ from the ARINC-664 ES model. These differences are: FWFT FIFOs for each queue are implemented in the *Memory* module and the Memory module is converted to HDL to simplify Vivado simulation, four scheduling algorithms are implemented in the *SchedulerDecider* block of the *Server* module, and Simulink blocks are converted to both hardware and software in the *Analysis* module.

5.1 Implementation of the Dynamic Server Module

A hardware implementable *DynamicServer* module is proposed. This module can apply all the schedulers described in Chapter 4. Fig. 5.2 shows the structure of the *DynamicServer* module.

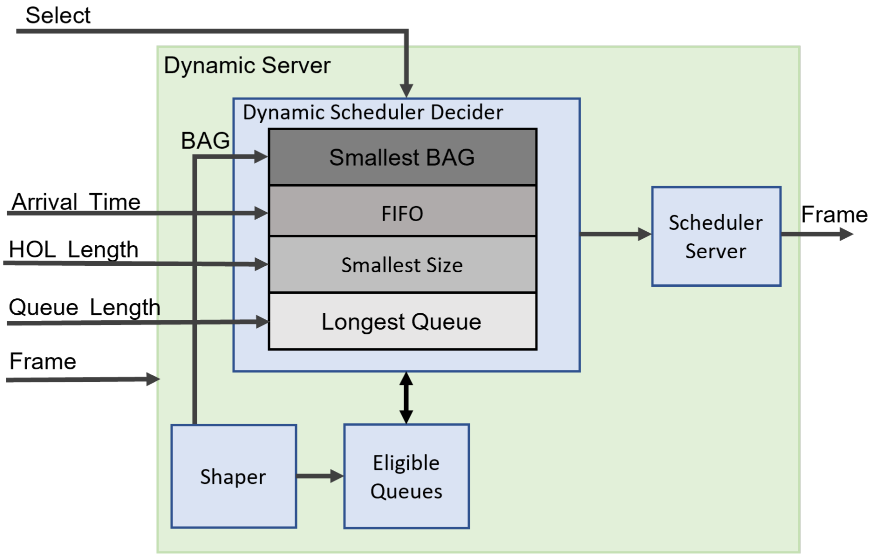


Figure 5.2 : Simulink model of the Dynamic Server module.

In run-time, the user can switch the scheduling algorithm with a *Select* pin. Necessary parameters for implementing the SB, LQ, FIFO, and SS scheduling algorithms are the BAG, number of bytes, HoL frame length, and HoL arrival time of each VL, respectively. BAG parameters are stored inside of the *Shaper* block. The number of bytes information is provided by the *FrameFIFO* blocks. HoL frame length and HoL arrival time of VLs must be known in advance to decide which queue to serve next. The HoL arrival time information is 64-bits, and the HoL frame length information is 16-bits. Instead of storing each value in a separate FWFT FIFO, i.e. spending two FWFT FIFOs for each queue, the concatenation of these two information, which is 80-bits, can be stored in one FWFT FIFO (*FWFTFIFO* block of the *Memory* module), and then split into two pieces in the *DynamicServer* module.

5.2 Implementation of the Analysis Module

The *Analysis* module is crucial for the Dynamic Scheduler model because it operates statistical analysis and decides which scheduler algorithm to run. Similar to the ARINC-664 ES model, the *Analysis* module of the Dynamic Scheduler model measures four statistics for each queue: The mean, standard deviation, and maximum of jitter performances, and the number of served frames. The jitter is calculated for each frame on a per-flow basis to measure the first three statistics.

Calculating the jitter is implemented by using the *JitterCalculator* subsystem whose Simulink implementation is shown in Fig. 5.3.

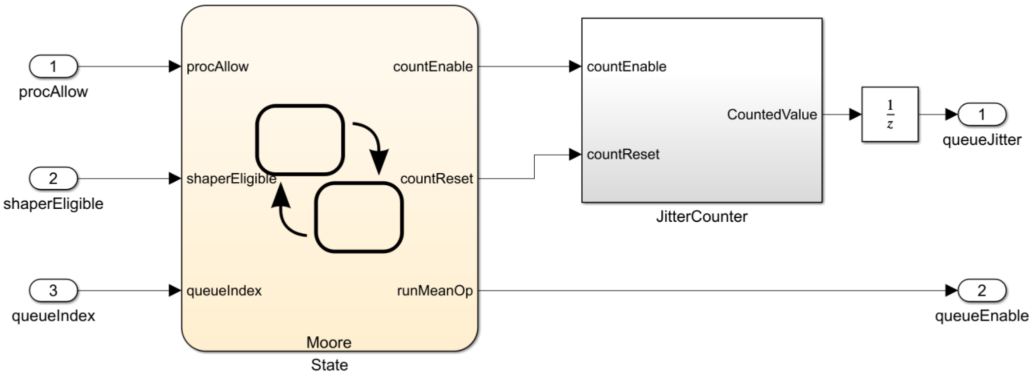


Figure 5.3 : Simulink implementation of the Jitter Calculator subsystem.

In this subsystem, the *JitterCounter* subsystem is responsible for calculating the jitter value by counting at each step from the moment the queue becomes eligible to the moment it is decided that the queue will be served. Operation of the *JitterCalculator* subsystem is suitable to PL; hence, this subsystem is converted to HDL with MATLAB HDL Coder. The mean, standard deviation and maximum of jitter performances, and the number of served frames are implemented by using the *StatisticCalculator* subsystem whose Simulink implementation is shown in Fig. 5.4.

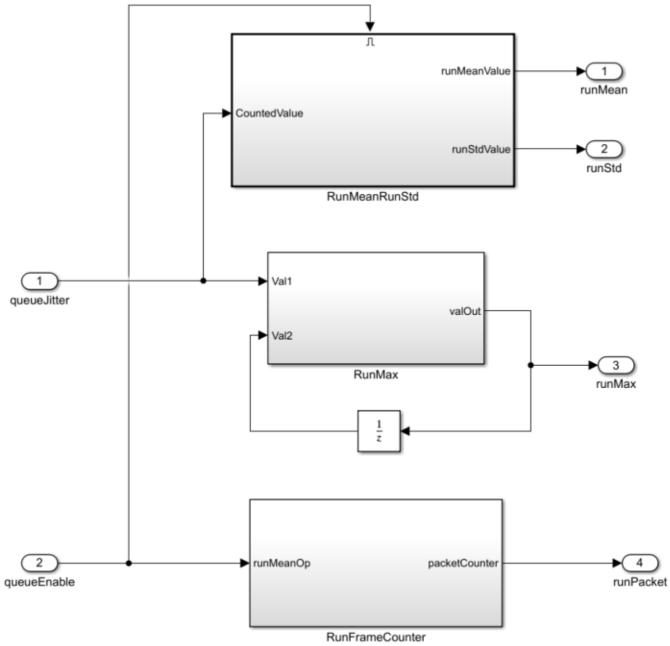


Figure 5.4 : Simulink implementation of the Statistic Calculator subsystem.

In this subsystem, the *RunMeanRunStd* subsystem recalculates the mean and standard deviation values based on the new jitter input, *RunMax* subsystem calculates the maximum jitter and the *RunFrameCounter* subsystem counter increases when a frame is served. Operations of the *StatisticCalculator* subsystem are suitable to PS; hence, this subsystem is converted to C with MATLAB Embedded Coder.

5.3 System on Chip Implementation

The *JitterCalculator* subsystem of the *Analysis* module is located on the PL and the *StatisticCalculator* subsystem of the *Analysis* module is located on the PS to build an SoC. In this study, PL is implemented by using the logic fabric of the FPGA, and the PS is implemented on the Microblaze, which is a soft-core processor built by using the logic fabric of the FPGA. Then, a block design is built by using Xilinx Vivado. Fig. 5.5 shows the block design. In this design, a True Dual Port Block Random Access Memory, will be referred as BRAM in the rest of the thesis, used as a communication interface between the PL and PS [50]. One channel of BRAM is called Channel A, and the other channel is called Channel B. Channel A communicates with Microblaze through Advanced Extensible Interface (AXI) interface, and Channel B communicates with PL through hand-written *BRAMWrapperr* HDL, which is responsible for the W/R operations of the PL logic. For each queue, PL has two outputs: *QueueJitter* and $QueueEnable$. The $QueueJitter$ is the calculated jitter value of each frame, and $QueueEnable$ is the indication that the jitter calculation for a frame is completed. Fig. 5.6 shows the BRAM interaction of PL outputs. Each $QueueJitter$ and $QueueEnable$ parameters are located in separate memory addresses. When the $QueueJitter$ is calculated, on the PL side, it is written to the BRAM $QueueJitter$ address of the related queue, and the BRAM $QueueEnable$ address of the related queue is set to 1, i.e., true. When this happens, the related queue is called an active queue. On the PS side, Microblaze starts to read the address of $QueueEnable$ for each queue through the AXI interface, beginning with the first queue. Then, if the $QueueEnable$ of the related queue is true, the $QueueJitter$ value of the queue is read, and $QueueEnable$ of the BRAM address is set to false by the Microblaze. This operation is followed for each queue, and then the $OneStep$ function is called to calculate new jitter statistics for active queues. Microblaze calculates the jitter statistics for each queue and decides which scheduling algorithm must be applied. So, each time the $OneStep$ function is completed, the current scheduling algorithm parameter (Scheduler Select - $schSelect$) is written to the related address (0x200). On the PL side, $BRAMWrapperr$ HDL periodically reads the scheduling address (0x200) and assigns the output to the scheduling algorithm input of the PL. In this approach, PS reads the information of PL with polling. As an alternative, an interrupt mechanism could be used. Fig. 5.7 shows an example flow of the Microblaze for a single queue.

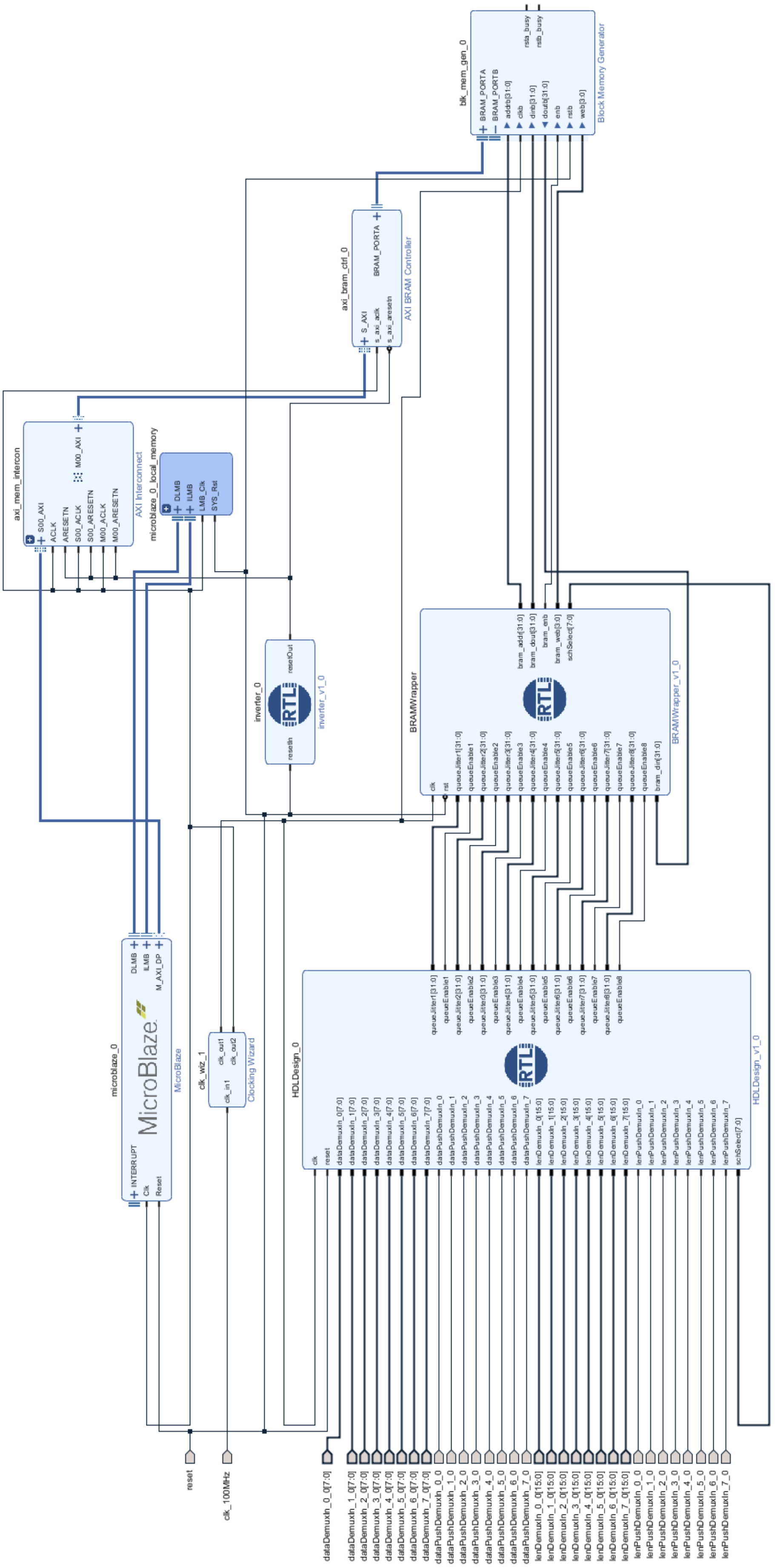
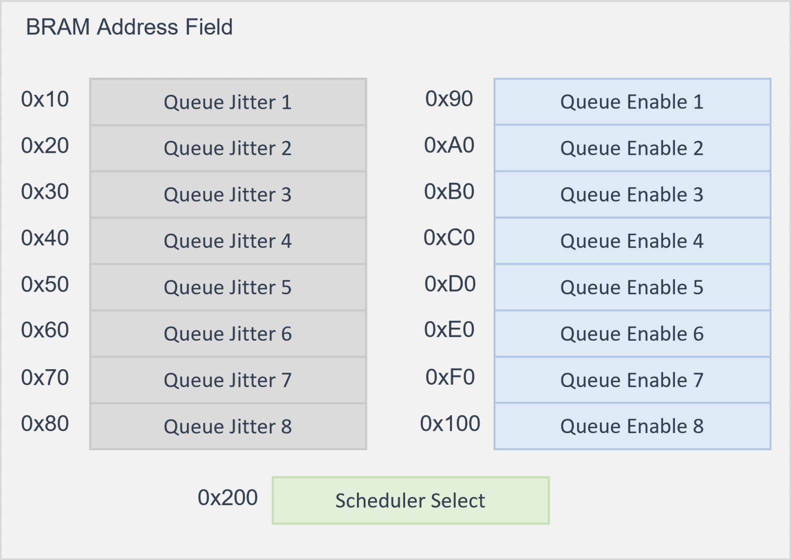


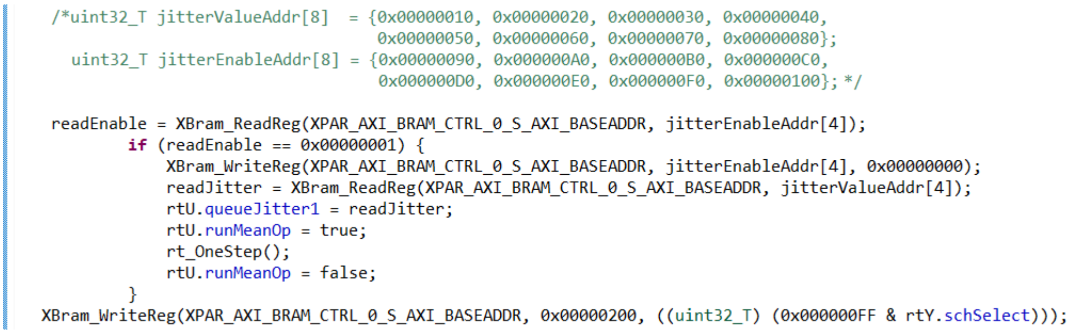
Figure 5.5 : Vivado implementation of Dynamic Scheduler SoC.



BRAM Address Field

0x10	Queue Jitter 1	0x90	Queue Enable 1
0x20	Queue Jitter 2	0xA0	Queue Enable 2
0x30	Queue Jitter 3	0xB0	Queue Enable 3
0x40	Queue Jitter 4	0xC0	Queue Enable 4
0x50	Queue Jitter 5	0xD0	Queue Enable 5
0x60	Queue Jitter 6	0xE0	Queue Enable 6
0x70	Queue Jitter 7	0xF0	Queue Enable 7
0x80	Queue Jitter 8	0x100	Queue Enable 8
0x200	Scheduler Select		

Figure 5.6 : BRAM addresses.



```
/*uint32_T jitterValueAddr[8]  = {0x00000010, 0x00000020, 0x00000030, 0x00000040,
                                  0x00000050, 0x00000060, 0x00000070, 0x00000080};
  uint32_T jitterEnableAddr[8] = {0x00000090, 0x000000A0, 0x000000B0, 0x000000C0,
                                  0x000000D0, 0x000000E0, 0x000000F0, 0x00000100}; */

readEnable = XBram_ReadReg(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR, jitterEnableAddr[4]);
        if (readEnable == 0x00000001) {
            XBram_WriteReg(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR, jitterEnableAddr[4], 0x00000000);
            readJitter = XBram_ReadReg(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR, jitterValueAddr[4]);
            rtU.queueJitter1 = readJitter;
            rtU.runMeanOp = true;
            rt_OneStep();
            rtU.runMeanOp = false;
        }
XBram_WriteReg(XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR, 0x00000200, ((uint32_T) (0x000000FF & rtY.schSelect)));
```

Figure 5.7 : Example flow of the Microblaze for a single queue.

6. RESULTS

In this section, first, the performance results of the ARINC-664 ES scheduling algorithms using the Simulink models are presented. The mean, standard deviation, and maximum of jitter performances are reported for 8 queues scenarios corresponding to 8 VLs.

The simulation time is set to 500 ms for all experiments. In the ARINC-664, the BAG values are expressed as powers of 2 from 1 ms to 128 ms (1, 2, 4, ... 128). However, the BAG values are reduced to the range of 50 - 400 µs at 50 µs intervals to decrease simulation time in this study. Then, the hardware implementation results for each scheduling algorithm are represented. In the rest of the thesis, queues from 1 to 8 will be referred as Q1 to Q8.

6.1 Simulation Results

Two configuration scenarios, including Packet Arrival Rate, BAG, and L_{max} parameters, are created and listed in Table 1 and Table 2, respectively. In both scenarios, packet traffic is generated in accordance with the carrying capacity of the queues, and the packet inter-arrival times are generated according to the exponential distribution (i.e., Markov-M) for all VLs. In the first scenario, L_{max} values are set in different lengths from 100 bytes to 1400 bytes. In the second scenario, L_{max} values are set as in 160-byte length intervals from 160 bytes to 1280 bytes.

6.1.1 Scenario 1

Table 6.1 shows the configuration parameters of this scenario. Note that each queue is uniquely dedicated to the incoming packets of the corresponding VL.

The *Analysis* module calculates the mean, standard deviation, and maximum of jitter performances and the number of served frames. Fig. 6.1, Fig. 6.2, and Fig. 6.3 shows the statistics for the mean, standard deviation, and maximum of jitter, respectively.

Table 6.1 : Configuration parameters for scenario 1.

Queue	Packet Arrival Rate (Mbit)	Queue Mode	Length (Byte)	BAG (us)	Theoretical Maximum Service Rate (Mbit)
Q1	220	M	1400	50	224
Q2	92	M	1200	100	96
Q3	50	M	1000	150	53.33
Q4	30	M	800	200	32
Q5	16	M	600	250	19.2
Q6	8	M	400	300	10.66
Q7	3	M	200	350	4.57
Q8	1	M	100	400	2

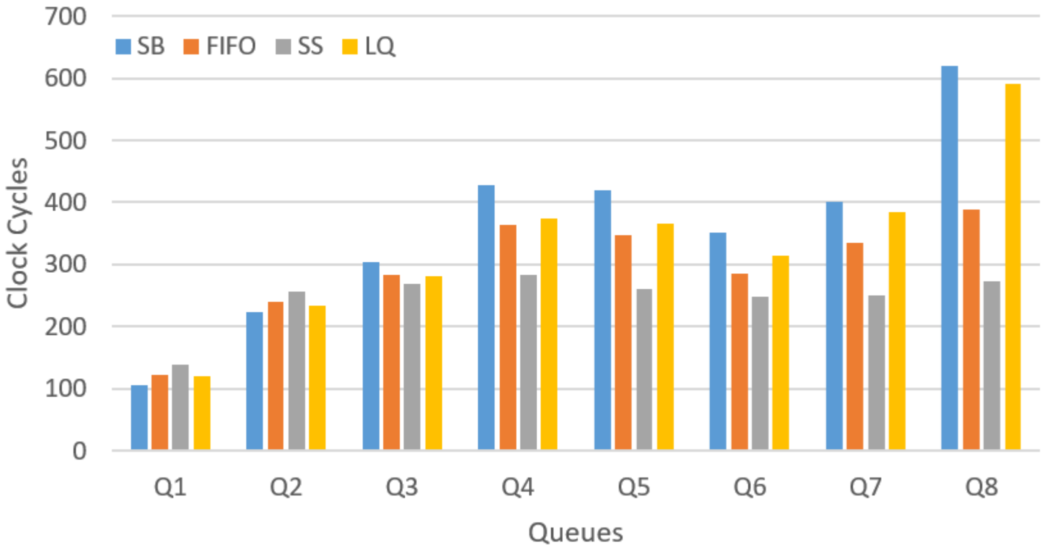


Figure 6.1 : Mean jitter results for scenario 1.

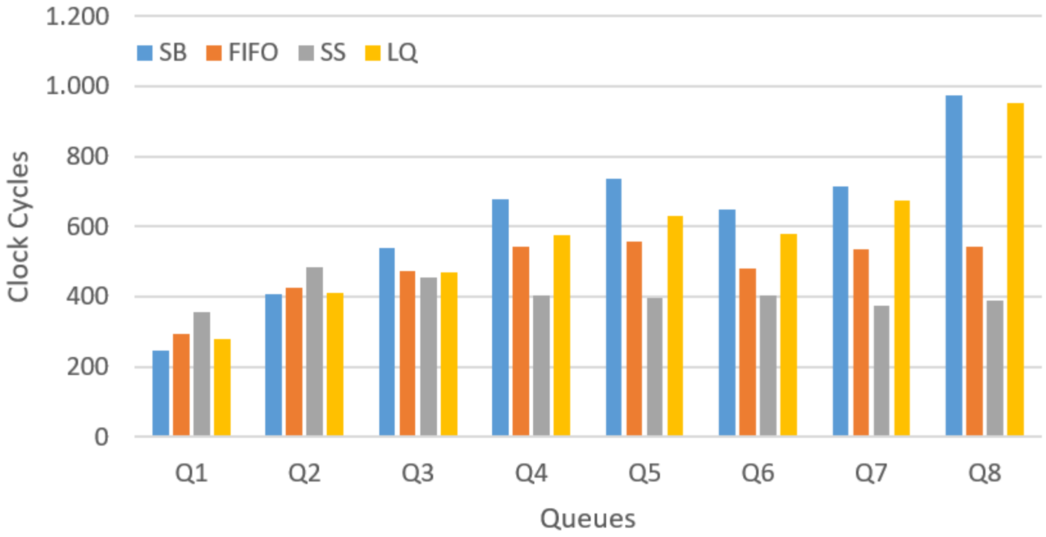


Figure 6.2 : Standard deviation of jitter results for scenario 1.

In scenario 1, the Q1 queue has the highest arrival rate; hence, frames will arrive at the *Memory* module of Q1 more often than any other queue. This means that Q1 will be the only queue that is not empty most of the simulation time; hence, the Q1 jitter is close to zero most of the simulation time, and the jitter of the Q1 is the lowest for all the scheduling algorithms. However, some differences can be seen among scheduling algorithms. For instance, the mean jitter for Q1 is the highest for the SS algorithm. It is an expected outcome because Q1 has the highest HoL frame length. From Q1 to Q8, the mean value increases in different ratios. For instance, the mean jitter of the Q8 with the SS algorithm is the lowest because its HoL frame length is the smallest.

Generally, the standard deviation of jitter increases from Q1 to Q8 as expected. However, the Q8 value of the SS algorithm shows a different pattern because the jitter is zero or close to zero due to HoL size.

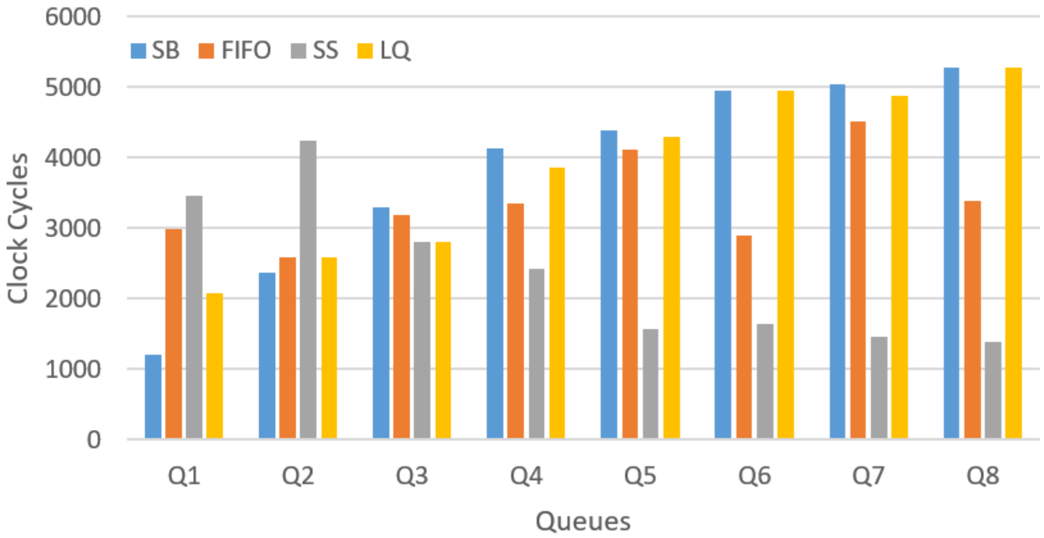


Figure 6.3 : Maximum jitter results for scenario 1.

The maximum jitter results demonstrate that the SB algorithm and SS algorithm values are opposite, the FIFO algorithm shows a random pattern, and the LQ algorithm generally increases from Q1 to Q8, as expected.

6.1.2 Scenario 2

Table 6.2 shows the configuration parameters of this scenario. Note that each queue is uniquely dedicated to the incoming packets of the corresponding VL.

Table 6.2 : Configuration parameters for scenario 2.

Queue	Packet Arrival Rate (Mbit)	Queue Mode	Length (Byte)	BAG (us)	Theoretical Maximum Service Rate (Mbit)
Q1	25	M	160	50	25.6
Q2	25	M	320	100	25.6
Q3	25	M	480	150	25.6
Q4	25	M	640	200	25.6
Q5	25	M	800	250	25.6
Q6	25	M	960	300	25.6
Q7	25	M	1120	350	25.6
Q8	25	M	1280	400	25.6

The *Analysis* module calculates the mean, standard deviation, and maximum of jitter and the number of served frames. Fig. 6.4, Fig. 6.5 and Fig. 6.6 shows the statistics for mean, standard deviation and maximum of jitter of scenario 2, respectively.

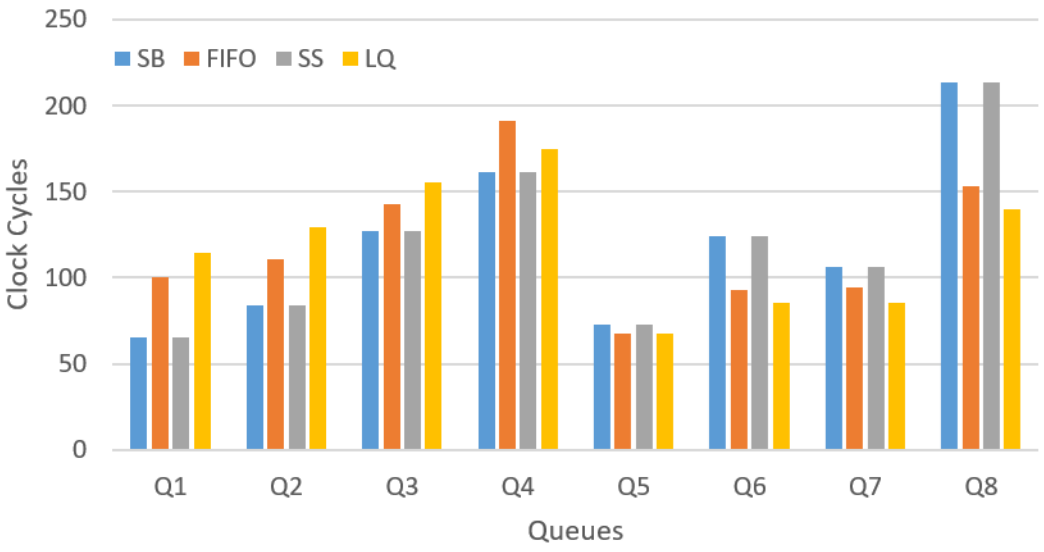


Figure 6.4 : Mean jitter results for scenario 2.

The traffic of scenario 2 is lighter than scenario 1. So, serving the frames with zero jitter occurs more often in scenario 2, and therefore the mean jitter values of scenario 2 are generally lower. In scenario 2, the BAG values and the HoL lengths have the same priority order; hence the SS and SB algorithms provide the same results.

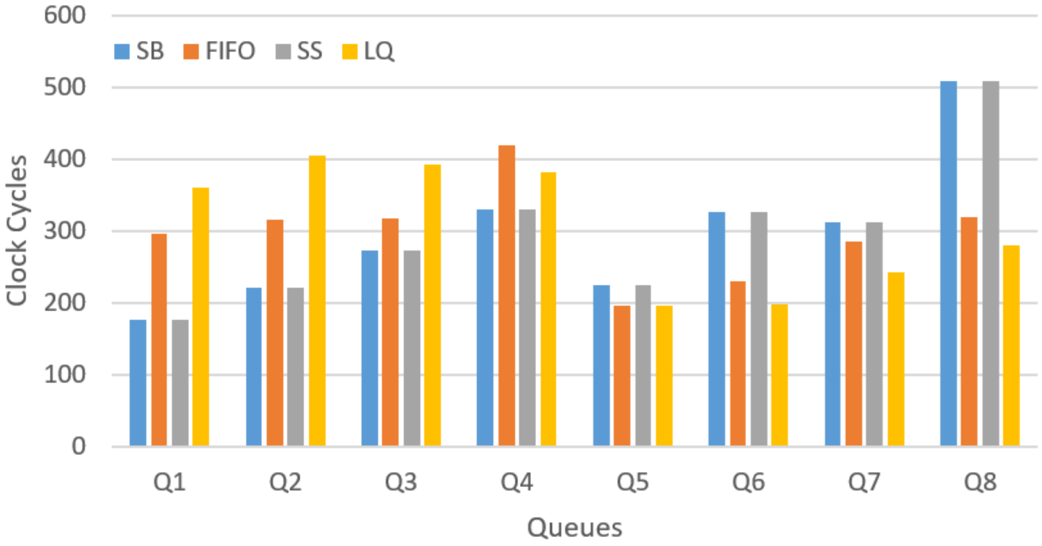


Figure 6.5 : Standard deviation of jitter results for scenario 2.

The standard deviation of jitter results demonstrates a similar pattern to the mean jitter results.

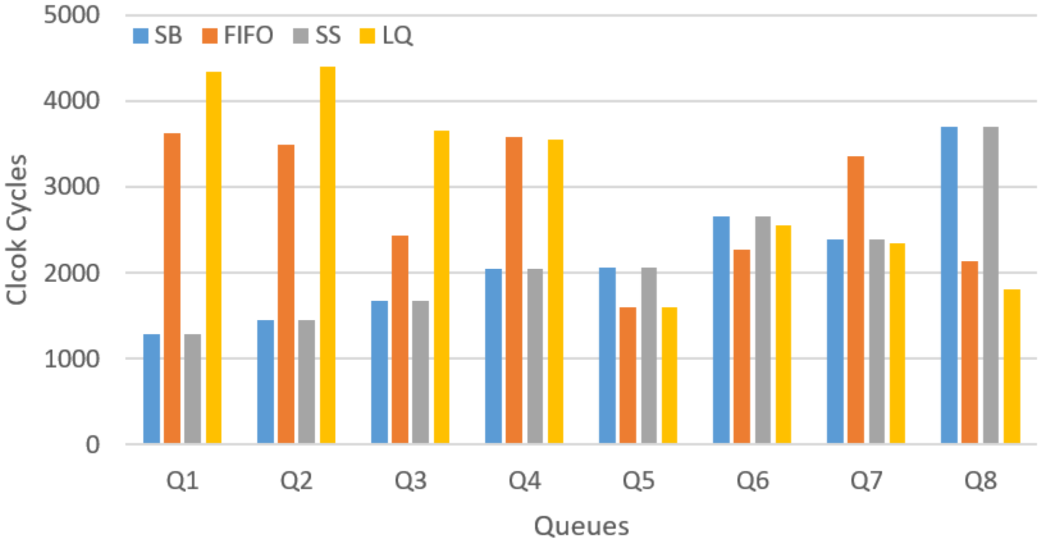


Figure 6.6 : Maximum jitter results for scenario 2.

In maximum jitter results, the FIFO shows a random pattern, the LQ algorithm generally increases from Q8 to Q1, and the SS and SB algorithms show the same pattern; both generally increase from Q1 to Q8, as expected.

The Pareto front figures of the scenario 1 and scenario 2 are shown in Fig. 6.7 and Fig. 6.8, respectively. In both figures, each dot represents a triplet, which includes the mean, standard deviation, and maximum of jitter. There are four algorithms and eight queues for each scenario; so 32 dots are used. The dots whose parameters have the best parameters in their queue are marked with different colours. The best dot of each queue can be tracked by using the legends of the figures. Also, two or more points are connected with a line to build a Pareto line if some of their parameters have the best values but they are not superior to each other. Please note that some dots overlap in 6.8 because the SB and SS algorithms show the exact same pattern in Scenario 2.

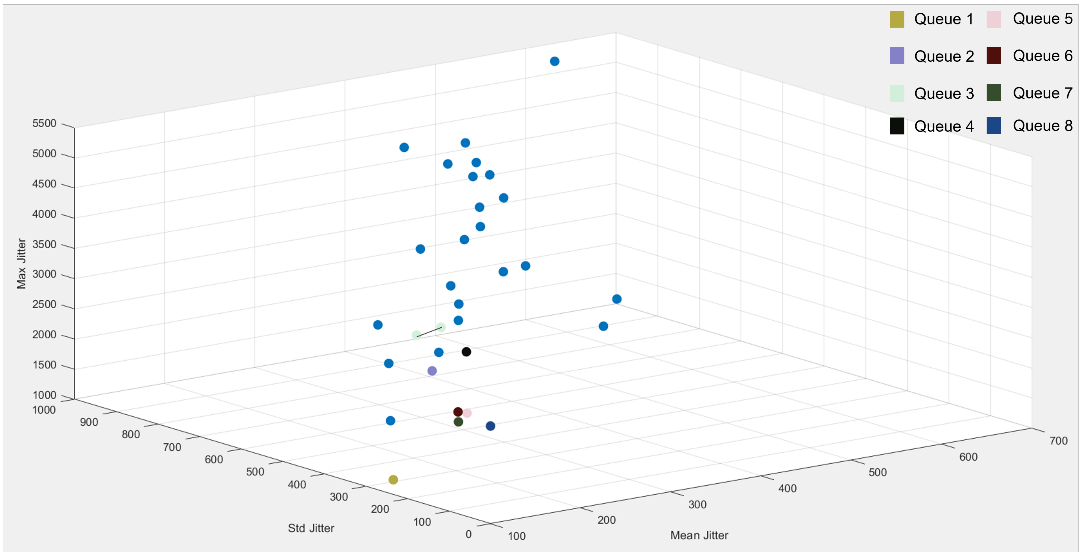


Figure 6.7 : The Pareto front figures of each queue in scenario 1.

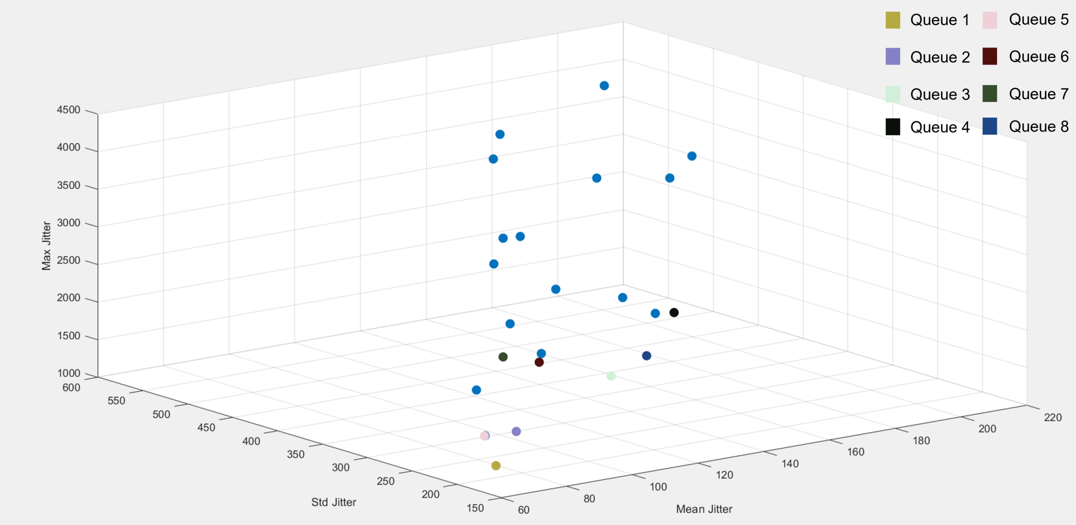


Figure 6.8 : The Pareto front figure of each queue in scenario 2.

6.2 Implementation Results

The Simulink model is converted to VHDL with MATLAB HDL Coder. During this conversion, minimize clock enable option is turned on due to simplicity. Then, generated VHDL files are implemented on Kintex-7 XC7K325T FPGA by using Xilinx Vivado. The clock frequency is set as 125 MHz.

Look Up Table (LUT), Flip-Flop (FF), and BRAM utilizations are reported for the ES implementation supporting 8 queues for all the scheduling algorithms and 32 queues for the SB algorithm. Table 6.3 shows the utilization for the *Server* module of four scheduling algorithms and the *DynamicServer* module for 8 queues. In this table, the differences occur due to different data widths of parameters for each module.

The FIFO algorithm spends the most resource among the four algorithms because arrival time information is 64 bits. The SS algorithm spends fewer resources than LQ because while queue size information is 32 bits, HoL length information is 16 bits. The SB algorithm spends the least resource because in contrary to other algorithms, SB does not need an external parameter for deciding which queue to serve next. Instead, it uses the BAG values. The *DynamicServer* module with a dynamic scheduling algorithm spends the most resources because it combines the other four algorithms.

Table 6.3 : Utilization report of the Server module under different scheduling scenarios for 8 queues.

Scheduling Algorithm	LUT	FF	BRAM
Smallest BAG	607	391	8
FIFO	1392	1051	16
Smallest Size	672	523	16
Longest Queue	940	699	8
Dynamic Scheduling	2336	1689	16

In addition to these results, the SB algorithm with 32 queues is implemented on the same FPGA. It spends 4217 LUTs, 3284 FFs, and 32 BRAMs, which is approximately 6 times more LUTs, 7 times more FFs, and 3 times more BRAMs than the SB scheduling algorithm with 8 queues.

Comparison of the LUT and FF utilization for each algorithm is shown in Fig. 6.9 and Fig. 6.10, respectively.

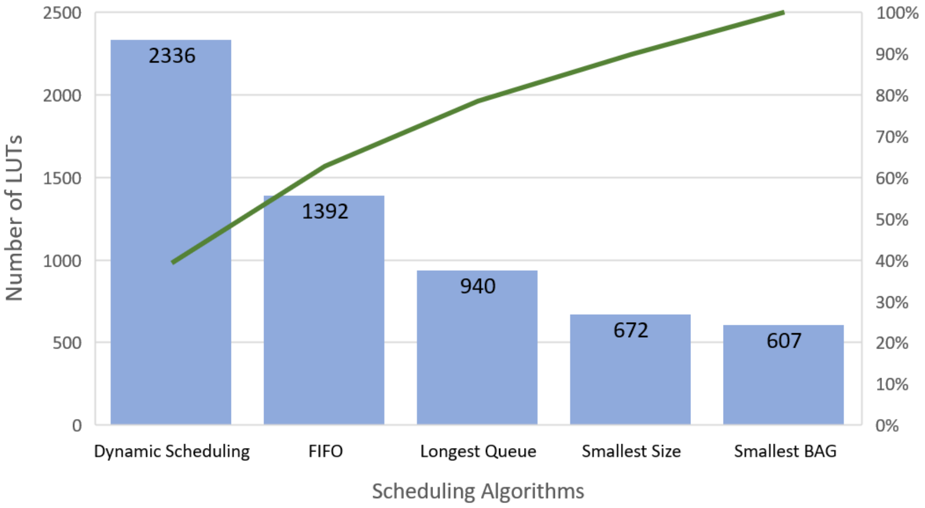


Figure 6.9 : Comparison of the LUT utilization.

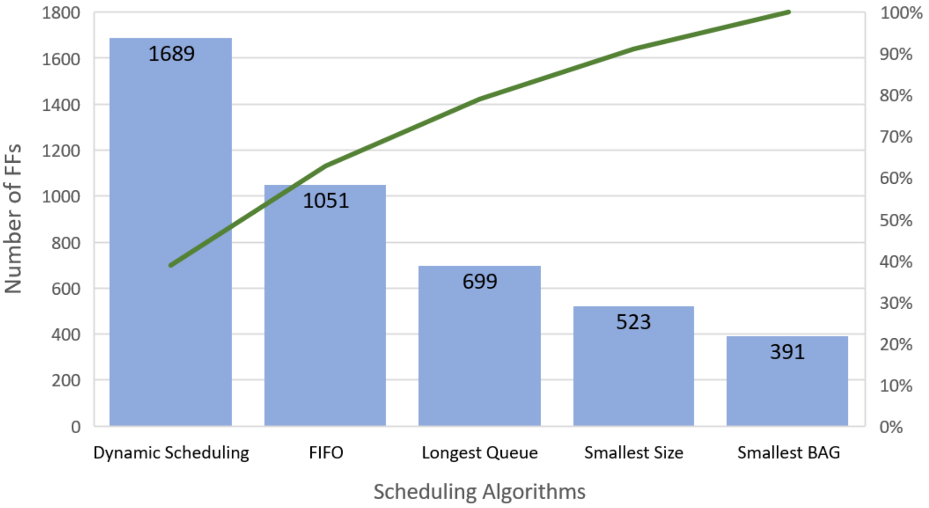
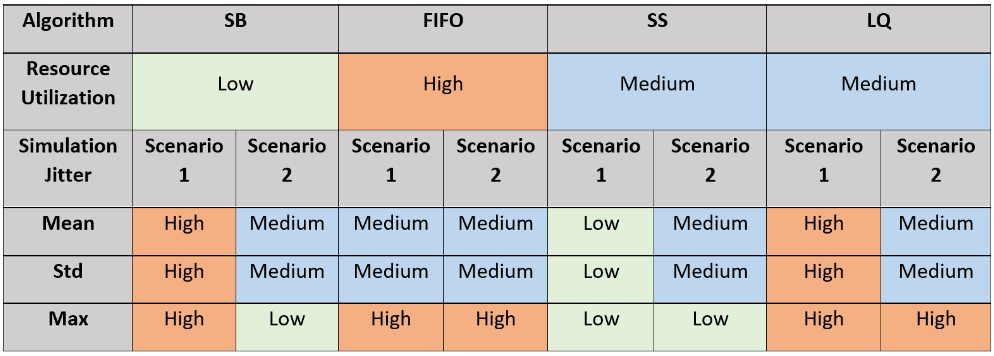


Figure 6.10 : Comparison of the FF utilization.

A comparison table is built to compare scheduling algorithms from both a hardware implementation and simulation perspective. For simulation, the mean value of 8 queues (Q1 to Q8) is calculated for the mean, standard deviation, and maximum of jitter of each scheduling algorithm. Then, 3 equal intervals are created between the minimum and maximum values of each parameter for each scenario, and these intervals are labeled as "low," "medium, and "high" in increasing order. The comparison table is shown in Fig. 6.11.



Algorithm	SB		FIFO		SS		LQ	
Resource Utilization	Low		High		Medium		Medium	
Simulation Jitter	Scenario 1	Scenario 2	Scenario 1	Scenario 2	Scenario 1	Scenario 2	Scenario 1	Scenario 2
Mean	High	Medium	Medium	Medium	Low	Medium	High	Medium
Std	High	Medium	Medium	Medium	Low	Medium	High	Medium
Max	High	Low	High	High	Low	Low	High	High

Figure 6.11 : Comparison of the scheduling algorithms from both implementation and simulation perspective.

This table shows that the implementation results are scenario independent, whereas the simulation results are scenario dependent. In scenario 1, the SB shows the worst, and the SS shows the best simulation performance. In scenario 2, the mean and standard deviation jitter values for each algorithm are so close to each other that the interval is not divided into three groups. Instead, all the values are classified as medium. At maximum jitter values, the SB and SS perform better than the FIFO and LQ scheduling algorithms.

Converted VHDL codes of the Dynamic Scheduling model are simulated in Vivado to ensure that the Simulink model is operating correctly. The ARINC-664 ES model includes the *Loaders*, *Memory*, *Analysis*, and *Server* modules. As mentioned in the previous chapter, the *Mmemory* module is converted to HDL to simplify simulation. There are only 4 inputs of the *Memory* module for each queue: two for *FWFTFIFO* blocks (*length* and *lengthpush*) and two for *FrameFIFO* blocks (*data* and *datapush*). In the Simulink model, these 4 inputs of each queue are logged in text files. Then, text files are read with a hand-written Verilog simulation code. Inputs are captured for 1250000 clock cycles (10 us). The scheduler algorithm is switched every 2.5 us. Fig. 6.12 shows an example scenario for the LQ algorithm. In this figure, cyan-colored signals show how many bytes are stored in each queue, red-colored signals show which of them are eligible, pink-colored signals show the masked output for each queue, and gray-colored signals show the scheduling mechanism. It can be seen that queues 2, 3, 5, 7, and 8 are eligible; however, others are not eligible. Therefore, the masked value of queue 4 is set to 0 because queue 4 is not eligible. The *SchedulerDecider* is triggered, and queue 2 is selected because it has the highest byte and is also eligible.

In the last part of the results, a simple scenario for the Dynamic Server model is created. For this scenario, first, data inputs of the first configuration are logged. Then, the scheduler algorithm is switched according to the following scenario: Initial scheduler algorithm is SB. If the maximum jitter value of Q5 exceeds 2000, the scheduler algorithm switches to the SS. In Fig. 6.3, it can be seen that the maximum jitter value of Q5 decreases significantly with the SS algorithm. Hence, this scenario aims to prevent excessive jitter of Q5. By using the flow of the Fig. 5.7, the scheduler algorithm switches when the conditions above are met. Fig. 6.13 shows the simulation of this scenario.

Below, each step of Fig. 6.13 is explained.

(a) When the jitter value of Q5 is calculated by PL, the associated *QueueEnable* address (0xD0) is set to 1, and the calculated value is written to the related *QueueJitter* address (0x50).

(b) Meanwhile, the Microblaze was periodically reading the related *QueueEnable* address (0xD0). When it reads 1, it starts to read the related *QueueJitter* address (0x50) to use the jitter value.

(c) The *Onestep* function is operated by the Microblaze. After some time, the function is completed, and the new scheduling algorithm is decided. The calculated jitter value was above 2000 (3194). So, 3, representing SS, is the new Scheduler Select value. The Microblaze assigns the new Scheduler Select (*schSelect*) value to the related address (0x200).

(d) The *BRAMWrapper* HDL reads the 0x200 and then sets the *schSelect* value to the *Select* pin of the *DynamicServer* module.

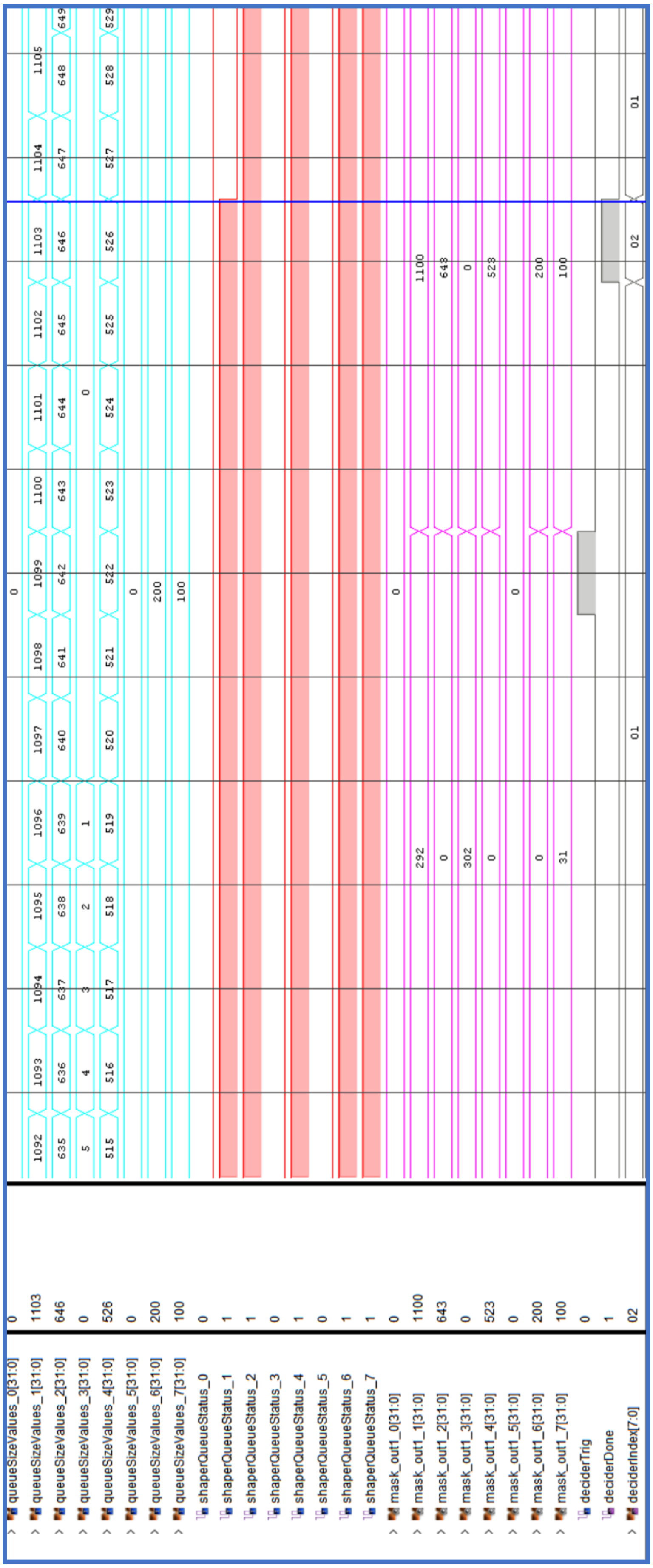


Figure 6.12 : Simulation of the LQ scheduling algorithm.

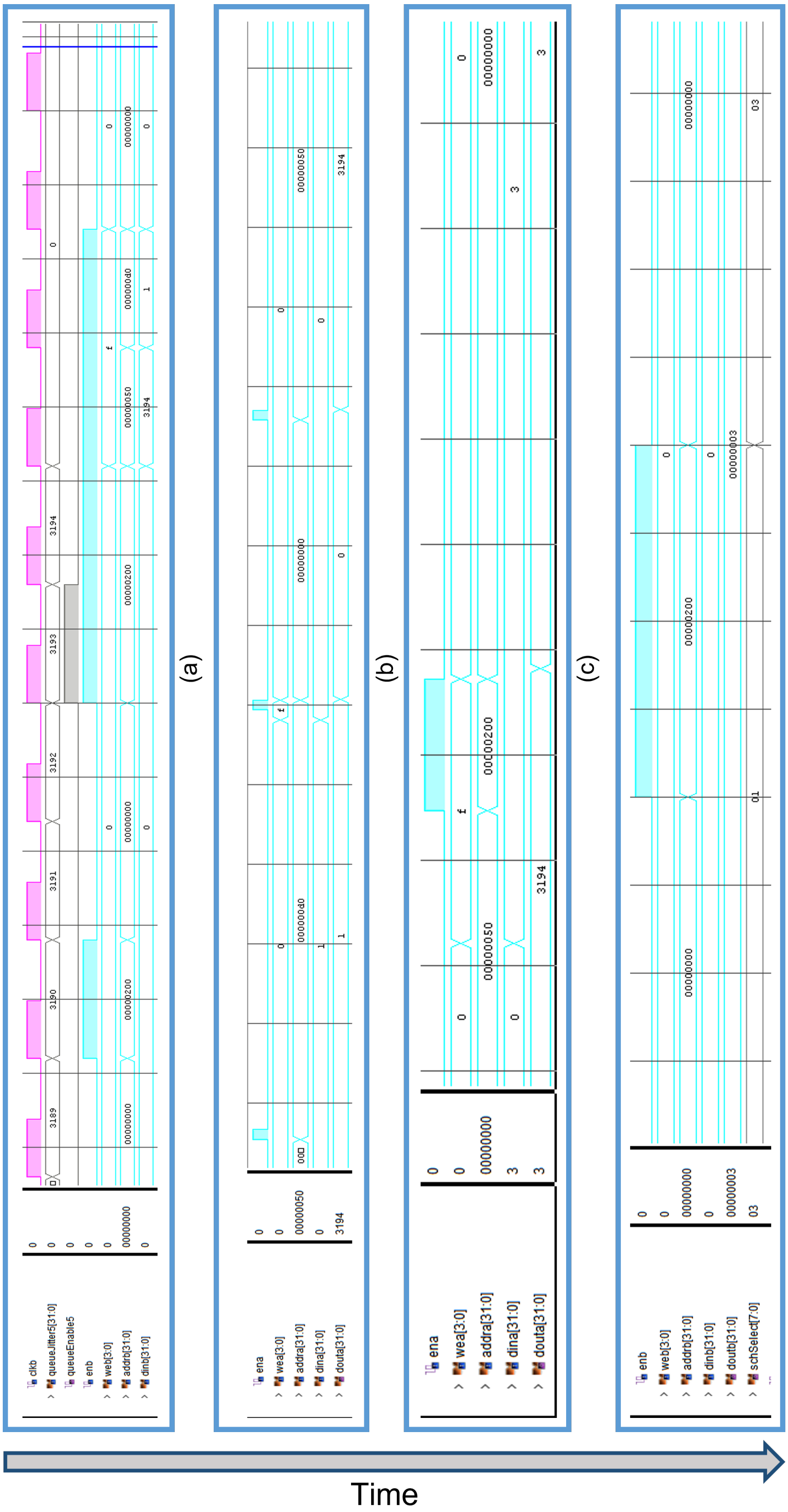


Figure 6.13 : Simulation of the Dynamic Server model.

7. CONCLUSION AND FUTURE WORK

In this study, the traffic regulator of the ARINC-664 ES is built and simulated using MATLAB Simulink. Then the simulation model is converted to HDL for hardware prototyping, and a dynamic scheduler system for ARINC-664 ES is built in the final step.

This work showed that MATLAB HDL Coder and MATLAB Embedded Coder are suitable tools for ARINC-664 ES. Generally, MATLAB HDL Coder showed excellent performance; it built readable VHDL codes and converted all the blocks to HDL correctly. MATLAB Embedded Coder also performed well; the blocks were converted to C correctly. However, variable names and the function context of the generated C codes could be more user-friendly.

From the simulation perspective, it is shown that scheduling algorithms of ARINC-664 ES have different characteristics; hence, each has its advantages and disadvantages. As seen in the results, performances of the scheduling algorithms depend on the configuration scenario. Thus, much more simulation scenarios must be run to reach more precise results.

From the implementation perspective, it is shown that the Smallest BAG algorithm spends the least and the FIFO algorithm spends the most FPGA resource. Also, the FIFO and the Smallest Size algorithm require more memory usage. These scheduling algorithms can cause a memory problem with the 128 VL ARINC-664 ES scenario in small FPGAs.

ARINC-664 ES Dynamic Server module is built on SoC. The entire SoC except the BRAM Wrapper code is designed in MATLAB Simulink, and yet, no problem occurred in building such a system. It shows that complex SoC systems can be built using MATLAB HDL Coder on PL and MATLAB Embedded Coder on PS.

An intelligent function for determining the most suitable algorithm to use will be developed in future work. The model will continuously switch using this algorithm as the input traffic type changes. Also, incoming Ethernet traffic (the *Loaders* module) will be transferred to the hardware design in future work. Finally, ZYNQ implementation will be performed to meet the speed necessities of an intelligent function.

REFERENCES

[1] **Cevher, S.**, **Mumcu, A.**, **Caglan, A.**, **Kurt, E.**, **Peker, M.K.**, **Hokelek, I. and Altun, S.** (2018). A fault tolerant software defined networking architecture for integrated modular avionics, *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, IEEE, pp.1–9.

[2] **Watkins, C.B. and Walter, R.** (2007). Transitioning from federated avionics architectures to Integrated Modular Avionics, *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*, pp.2.A.1–1–2.A.1–10.

[3] **ARINC** (2009). ARINC 664, P7: Avionics Full Duplex Switched Ethernet (AFDX) Network, *ARINC Specification*, *664*.

[4] **Kopetz, H.**, **Ademaj, A.**, **Grillinger, P. and Steinhammer, K.** (2005). The time-triggered ethernet (TTE) design, *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, IEEE, pp.22–33.

[5] **Gavriluţ, V.**, **Zhao, L.**, **Raagaard, M.L. and Pop, P.** (2018). AVB-aware routing and scheduling of time-triggered traffic for TSN, *Ieee Access*, *6*, 75229–75243.

[6] **Yang, X.**, **Scholz, D. and Helm, M.** (2019). Deterministic Networking (DetNet) vs Time Sensitive Networking (TSN), *Network*, *79*.

[7] **Land, I. and Elliott, J.** (2009). Architecting arinc 664, part 7 (afdx) solutions, *Xilinx, May*.

[8] **Zukerman, M.** (2013). Introduction to queueing theory and stochastic teletraffic models, *arXiv preprint arXiv:1307.2968*.

[9] **Kendall, D.G.** (1953). Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain, *The Annals of Mathematical Statistics*, 338–354.

[10] **Jain, R.** (1991). *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.*, Wiley professional computing, Wiley.

[11] **Parekh, A.K. and Gallager, R.G.** (1993). A generalized processor sharing approach to flow control in integrated services networks: the single-node case, *IEEE/ACM transactions on networking*, *1*(3), 344–357.

[12] **Katevenis, M.**, **Sidiropoulos, S. and Courcoubetis, C.** (1991). Weighted round-robin cell multiplexing in a general-purpose ATM switch chip, *IEEE Journal on selected Areas in Communications*, *9*(8), 1265–1279.

[13] **Demers, A.**, **Keshav, S. and Shenker, S.** (1989). Analysis and simulation of a fair queueing algorithm, *ACM SIGCOMM Computer Communication Review*, *19*(4), 1–12.

[14] **Bennett, J.C. and Zhang, H.** (1997). Hierarchical packet fair queueing algorithms, *IEEE/ACM Transactions on networking*, *5*(5), 675–689.

[15] **Balogh, T. and Medveckỳ, M.** (2011). Performance evaluation of WFQ, WF 2 Q+ and WRR queue scheduling algorithms, *2011 34th International conference on telecommunications and signal processing (TSP)*, IEEE, pp.136–140.

[16] **Miaji, Y. and Hassan, S.** (2009). A survey on the chronological evolution of timestamp schedulers in packet switching networks, *2009 2nd IEEE International Conference on Broadband Network & Multimedia Technology*, IEEE, pp.213–219.

[17] **Finzi, A.**, **Mifdaoui, A.**, **Frances, F. and Lochin, E.** (2018). Mixed-criticality on the AFDX network: Challenges and potential solutions, *The 9th European Congress EMBEDDED REAL TIME SOFTWARE AND SYSTEMS (ERTS 2018)*, pp.pp–1.

[18] **Geyer, F.** (2015). End-to-end flow-level quality-of-service guarantees for switched networks, *Ph.D. Dissertation*, Technische Universität München.

[19] **MathWorks**, (2021), MATLAB, `https://www.mathworks.com/products/matlab.html`, [Online; accessed 26-December-2021].

[20] **MathWorks**, (2021), Simulink, `https://https://www.mathworks.com/products/simulink.html`, [Online; accessed 26-December-2021].

[21] **MathWorks**, (2021), MATLAB HDL Coder, `https://www.mathworks.com/products/hdl-coder.html`, [Online; accessed 15-December-2021].

[22] **Aarenstrup, R.** (2015). Managing model-based design, *The MathWorks Inc*.

[23] **Erickson, J.**, (2021), HDL Coder Evaluation Reference Guide, `https://github.com/mathworks/HDL-Coder-Evaluation-Reference-Guide/releases/tag/v3.0.0`, [Online; accessed 15-December-2021].

[24] **MathWorks**, (2021), MATLAB Embedded Coder, `https://www.mathworks.com/products/embedded-coder.html`, [Online; accessed 15-December-2021].

[25] **Wikipedia**, (2021), Xilinx Vivado — Wikipedia, The Free Encyclopedia, `http://en.wikipedia.org/w/index.php?title=Xilinx%20Vivado&oldid=1061132879`, [Online; accessed 22-December-2021].

[26] **Xilinx**, (2021), Xilinx SDK, `https://www.xilinx.com/content/xilinx/en/products/design-tools/legacy-tools/sdk.html`, [Online; accessed 22-December-2021].

[27] **Hai, J.C.T.**, **Pun, O.C. and Haw, T.W.** (2015). Accelerating video and image processing design for FPGA using HDL coder and simulink, *2015 IEEE Conference on Sustainable Utilization and Development in Engineering and Technology (CSUDET)*, IEEE, pp.1–5.

[28] **Siwakoti, Y.P. and Town, G.E.** (2013). Design of FPGA-controlled power electronics and drives using MATLAB Simulink, *2013 IEEE ECCE Asia Downunder*, IEEE, pp.571–577.

[29] **Kechiche, L.** (2021). Hardware acceleration for deep learning of image classification, *2021 International Conference of Women in Data Science at Taif University (WiDSTaif)*, IEEE, pp.1–5.

[30] **Järviluoma, J.** (2015). Rapid prototyping from algorithm to FPGA prototype, *Ph.D. Dissertation*, Master's Thesis [Internet]. University of Oulu.

[31] **Baguma, G.** (2014). High Level Synthesis of FPGA-Based Digital Filters.

[32] **Vikström, A.**, (2009), A study of automatic translation of MATLAB code to C code using software from the MathWorks.

[33] **Akpolat, E.C.**, **Şeker, M.**, **Cevher, S.**, **Sapla, E. and Ata, S.Ö.** (2020). An Omnet++ Simulation for Performance Analysis of ARINC 664 P7 Avionics Data Network, *2020 28th Signal Processing and Communications Applications Conference (SIU)*, pp.1–4.

[34] **Liu, X.**, **Du, Z. and Lu, K.** (2018). Modeling and Simulation of Avionics Full Duplex Switched Ethernet (AFDX Network) Based on OPNET, *2018 3rd Joint International Information Technology, Mechanical and Electronic Engineering Conference (JIMEC 2018)*, Atlantis Press, pp.307–310.

[35] **Gebara, N.**, **Meng, J.**, **Luk, W. and Costa, P.** (2018). Scheduling Algorithms for High Performance Network Switching on FPGAs: A Survey, *2018 International Conference on Field-Programmable Technology (FPT)*, pp.166–173.

[36] **Kachris, C. and Vassiliadis, S.** (2006). A dynamically reconfigurable queue scheduler, *2006 International Conference on Field Programmable Logic and Applications*, IEEE, pp.1–4.

[37] **Molina, F.**, **Corral, P.**, **Aljaro, M.**, **de Scals, G. and Rodriguez, A.** (2020). Implementation of an AFDX Interface with Zynq SoC Board in FPGA, *Elektronika ir Elektrotechnika*, *26*(5), 11–15.

[38] **Durceylan, M.N.**, **Gemici, Ö.F.**, **Hökelek, I. and Aşmer, H.** (2020). Reconfigurable ARINC 664 End System Implementation on FPGA, *2020 28th Signal Processing and Communications Applications Conference (SIU)*, pp.1–4.

[39] **Kaya, S.**, **Gül, B.**, **Demir, M.S.**, **Hökelek, I.**, **Çöleri, S.**, **Garip, M. and Üvet, H.** (2021). Delay Optimization for Switchless ARINC 664 Mesh Networks with Cyclic Dependencies, *2021 IEEE Latin-American Conference on Communications (LATINCOM)*, pp.1–6.

[40] **Yeniaydın, M.**, **Gemici, Ö.F.**, **Demir, M.S.**, **Hökelek, I.**, **Çöleri, S. and Türeli, U.** (2021). Priority Re-assignment for Improving Schedulability and Mixed-Criticality of ARINC 664, *2021 IFIP Networking Conference (IFIP Networking)*, pp.1–6.

[41] **Adnan, M.**, **Scharbarg, J.L.**, **Ermont, J. and Fraboul, C.** (2010). Model for worst case delay analysis of an AFDX network using timed automata, *2010 IEEE 15th Conference on Emerging Technologies Factory Automation (ETFA 2010)*, pp.1–4.

[42] **Ren, Y.**, **Hu, F. and Li, J.** (2011). End to end jitter control on afdx network, *Proceedings 2011 International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE)*, IEEE, pp.515–518.

[43] **MathWorks**, (2021), SimEvents, `https://www.mathworks.com/products/simevents.html`, [Online; accessed 15-December-2021].

[44] **Suthaputchakun, C.**, **Sun, Z.** *et al.* (2015). Impact of end system scheduling policies on afdx performance in avionic on-board data network, *2015 2nd International Conference on Advanced Informatics: Concepts, Theory and Applications (ICAICTA)*, IEEE, pp.1–6.

[45] **Gao, X.**, **Fu, Y.**, **Liu, T. and Jiafang, W.** (2014). A new re-shaping method for reducing delay jitter on AFDX network, *Proceedings of 2nd International Conference on Information Technology and Electronic Commerce*, IEEE, pp.29–33.

[46] **Tawk, M.**, **Liu, X.**, **Jian, L.**, **Zhu, G.**, **Savaria, Y. and Hu, F.** (2011). Optimal scheduling and delay analysis for AFDX end-systems, **Technical Report**, SAE Technical Paper.

[47] **Gurjar, S. and Lakshmi, B.** (2014). Optimal scheduling policy for jitter control in AFDX End-System, *International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014)*, IEEE, pp.1–4.

[48] **Scarduelli, R.**, **Bourdil, P.A.**, **Zilio, S.D.**, **Botlan, D.L. and Bourdil, P.A.** (2018). Time-accurate Middleware for the Virtualization of Communication Protocols, *arXiv preprint arXiv:1805.09256*.

[49] **MathWorks**, (2021), Simulink, `https://www.mathworks.com/help/dsp/mathematics-and-statistics.html`, [Online; accessed 02-January-2022].

[50] **Xilinx**, (2021), Block Memory Generator v8.4, `https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v8_4/pg058-blk-mem-gen.pdf`, [Online; accessed 28-December-2021].

CURRICULUM VITAE

Name Surname	**:** Mustafa UZUNER
Place and Date of Birth	**:** İstanbul, 1996
E-Mail	**:** muzuner96@gmail.com

EDUCATION:

- **B.Sc.** : 2018, Yıldız Technical University, Faculty of Electrical and Electronics, Department of Electronics and Communication Engineering

PROFESSIONAL EXPERIENCE:

- 2018-2022 Digital Design Engineer at Turkish Aerospace Industries

PUBLICATIONS ON THE THESIS:

- **M. Uzuner, İ. Hökelek and B. Örs** 2021. A Model Based Hardware Implementation of Traffic Regulator in ARINC-664 ES, *2021 13th International Conference on Electrical and Electronics Engineering (ELECO)*