ISTANBUL TECHNICAL UNIVERSITY

ELECTRICAL AND ELECTRONICS ENGINEERING FACULTY

MODEL BASED DESIGN AND IMPLEMENTATION OF AN IOT NETWORK RESISTANCE AGAINST RPL ROUTING ATTACKS

SENIOR DESIGN PROJECT

Fregis SALA
Tolga KARAKAYA

Electronics And Communication Engineering

Department

JUNE 2021

ISTANBUL TECHNICAL UNIVERSITY

ELECTRICAL AND ELECTRONICS ENGINEERING FACULTY

MODEL BASED DESIGN AND IMPLEMENTATION OF AN IOT NETWORK RESISTANCE AGAINST RPL ROUTING ATTACKS

SENIOR DESIGN PROJECT

Fregis SALA
(040160915)
Tolga KARAKAYA
(040150049)

Electronics And Communication Engineering

Department

Project Advisor: Prof. Dr. Sıddıka Berna ÖRS YALÇIN

JUNE 2021

We are submitting the Senior Design Project entitled as “Model Based Design and Implementation of an IoT Network Resistance Against RPL Routing Attacks” The Senior Design Project Interim Report has been prepared as to fulfill the relevant regulations of the Electronics and Communication Engineering Department of Istanbul Technical University. We hereby confirm that we have realized all stages of the Senior Design Project Interim Report by ourselves, and we have abided by the ethical rules with respect to academic and professional integrity .

Fregis SALA .....
(040160915)

Tolga KARAKAYA .....
(040150049)

To our loved ones,

FOREWORD

We firstly would like to thank Prof. Dr. Sıddıka Berna Örs Yalçın, for being in contact with us at every stage of the project, share her knowledge with us when we needed it and take her own time to meet us at days. We would also like to thank Res. Asst. Mehmet Onur Demirtürk for guiding us through the project

Lastly, we would like to say thank you to all our friends for being there at our times of need, our professors who were understanding when we needed and our families, who always supported us in any condition throughout our education life.

June 2021

Fregis SALA

Tolga KARAKAYA

TABLE OF CONTENTS

Page

ABBREVIATIONS

AI : Artificial Intelligence
ALU : Arithmetic Logic Unit
API : Application Programming Interface
ARM : Advanced RISC Machines
ASIC : Application Specific Integrated Circuit
BRAM : Block Random-Access Memory
BUFG : Global Clock Buffer
CMOS : Complementary Metal Oxide Semiconductor
DSP : Digital Signal Processor
DVI : Digital Visual Interface
FF : Flip-Flop
FPGA : Field Programmable Gate Array
FPU : Floating Point Unit
GNU : GNU's Not Unix
GPU : Graphic Processing Unit
HDL : Hardware Description Language
HDMI : High-Definition Multimedia Interface
IO : Input - Output
IoT : Internet of Things
IP : Intellectual Property
ISA : Instruction Set Architecture
I2C : Inter-Integrated Circuit
LUT : Look-Up Table
LUTRAM : Look-Up Table Distribution Random-Access Memory
ML : Machine Learning
PLL : Phase-Locked Loop
RAM : Random-Access Memory
RGB : Red Green Blue
RISC : Reduced Instruction Set Computer
RTL : Register Transfer Level
SCCB : Serial Camera Control Bus
SoC : System on Chip
SVGA : Super Video Graphics Array
TF : Tensor Flow
UART : Universal Asynchronous Receiver Transmitter
VGA : Video Graphics Array

LIST OF TABLES

Page

LIST OF FIGURES

Page

MODEL BASED DESIGN AND IMPLEMENTATION OF AN IOT NETWORK RESISTANCE AGAINST RPL ROUTING ATTACKS

SUMMARY

The latest technological developments have made various IoT devices accessible and pivotal in daily life. From mobile phones to airplanes, many devices and machines use IoT systems. The main reason is the immense data exchange the IoT systems provide. The IoT systems by the exchange of data among others could provide a feedback system to allow the user improvements about their device. Also, the device datas could show the manufacturers which areas they should focus to improve their design to improve their efficiency.

Furthermore due to IoT systems requiring low cost and low power, the applications and devices made on IoT systems increase more each day. Smart devices are becoming very common for daily tasks from watches to kitchen applications which are simplifying the daily life. The need for IoT devices is increasing day by day due to the development on IoT devices cause the integration of them in daily life more which requires an average human to use more IoT devices. This creates a need to increase the production speed. Also, the cyberattacks which could have simple to catastrophic results for the users are possible scenarios if the designed IoT system has a weak cryptography structure.

For the designing part of the project in the software area Simulink, which is a program that allows the user to use or create block designs to create models, is utilized to create the Models of IoT systems with increased security via the usage of cryptography algorithms. Furthermore, the hardware implementation of the designed model in Simulink is done in Vivado Design Suite and Vitis IDE. Vivado Design Suite is a program that allows the user to simulate hardware designs like FPGA Evaluation Boards where Vitis IDE allows the user to create software programs to write instructions for FPGAs and other hardware designs.

In this project an aim was set to overcome these two problems with a solution propose of Model driven designing with secure implementation. The Model driven designing could make the IoT Systems to be designed rapidly. Also implementing high security algorithms to provide extra security on the IoT devices could help against cyberattacks. The proposed solution if done correctly could open the way for the fast production of secure IoT devices which would create a snowball effect on the development of IoT technology. All in all the project is stands on an edge where being able to go through could open the gates of immense technological advancements.

MODEL TABANLI RPL ROUTİNG SALDIRILARINA KARŞI IOT ŞEBEKE SAVUNMASI TASARIM VE UYGULAMASI

ÖZET

Teknolojideki gelişmelerle beraber birçok çeşit IoT cihazı günlük hayatın içinde yer almaya başladı. Cep telefonlarından uçaklara, birçok cihaz ve makine IoT sistemlerini kullanıyor. Bunun ana sebebi IoT sistemlerinin sağladığı uçsuz bucaksız bilgi değişimi. IoT sistemleri diğerleri arasında bilgi değişiminde bulunarak bir geri besleme sistemi aracılığıyla kullanıcılara cihazlarıyla ilgili gelişmeler sunabilir. Ayrıca, cihaz bilgileri, üreticilere cihazlarını geliştirerek verimlerini arttırmak için hangi alanlara odaklamaları gerektiğini gösterebilir.

Dahası IoT sistemlerinin düşük bütçe ve düşük güç istemelerinden ötürü, IoT sistemleri üzerine yapılan uygulama ve cihazlar gün geçtikçe artıyor. Saatlerden mutfak uygulamalarına kadar günlük hayatı kolaylaştıran akıllı cihazlar günlük hayatta çok yaygın hale geliyor. IoT cihazlara olan ihtiyaç, IoT cihazlarının gelişmesi sonucu günlük hayattaki kullanılmalarının artmasına yol açıp ortalama bir insanın daha fazla IoT cihazı kullanmasını sağladığı için, gün geçtikçe artıyor. Bu da üretim hızını arttırma ihtiyacını doğuruyor. Ayrıca IoT sistemlerinin zayıf şifreleme yapıları olması durumunda kullanıcılar için en basitinden en felaketine sonuçlar doğurabilecek siber saldırılar olası senaryolardır.

Projenin tasarım kısmı için yazılım alanında, kullanıcıya blok tasarımları kullanarak ya da tasarlayarak model yaratmalarını sağlayan bir program olan Simulink'ten şifreleme algoritmaları kullanımıyla güçlendirilmiş savunmaya sahip IoT Sistem modelleri oluşturmak için yararlanıldı. Dahası Simulink'te tasarlanan modelin donanım uygulaması Vivado Design Suite ve Vitis IDE ile yapıldı. Vivado Design Suite kullanıcıya FPGA Evrimsel Kartlar gibi donanımsal tasarımları benzetmesine olanak sağlıyorken, Vitis IDE kullanıcıya FPGA'ler gibi donanımsal tasarımlara yazılım programlarıyla komutlar yazmasına olanak sağlıyor.

Bu projede bu iki problemin üstesinden gelme hedefi, Model güdümlü tasarım ile güvenli uygulama çözüm önerisiyle kondu. Model güdümlü tasarım IoT sistemlerinin çok hızlı bir şekilde tasarlanmasını sağlayabilir. Ayrıca yüksek güvenlikli algoritmaları uygulamak IoT cihazlara extra güvenlik sağlamakla siber saldırılara karşı yardımcı olabilir. Önerilen çözüm önerisi, doğru yapılırsa, güvenli IoT cihazlarının hızlı üretiminin önünü açmakla IoT teknolojisinin gelişiminde kar topu etkisi yaratır. Sonuç olarak proje, ileriye gidilebilmesi durumunda uçsuz bucaksız teknolojik gelişmelerin kapısının açılabileceği bir eşikte bulunuyor.

1. INTRODUCTION

The technological advancements brought many devices to be in connection with each other for faster information flow to ease human life. Due to this Internet of Things (IoT) has created a massive global network around the whole globe which still continues to expand. Multiple devices being inter-connected to ease the information flow while is favorable it also creates many windows for information stealing through these connections. To prevent this, while keeping a low power for minimum energy usage with minimum area coverage IoT systems are being designed to have security against these potential attacks by encrypting the sent data.

Aside from these restrictions, one of the most significant needs of secure IoT Systems is the rapidity of the product development. Due to the need for IoT systems increasing exponentially, fast creation of IoT systems with low power and low cost with minimum area coverage is seen as a problem which is tried to be solved in this following paper. As a solution Model Driven Development (MDD) [1] method for embedded system design is used for the software and hardware design and implementation to achieve a quickly produced secure IoT systems. The project is focused on the configuration of the hardware algorithm to see if production of IoT systems with low cost and low power with minimum area coverage by combining hardware and software is possible to be done rapidly. In the making of the project following programs are utilized: MatLab, Simulink, Vivado and Vitis. To simulate and configurate the algorithm created Field Programmable Gate Arrays (FPGAs) [2] are benefitted from.

The thesis consists of 7 chapters which are shown in following order:

• Introduction

• IoT

• Utilized Programs

• Utilized Algorithms and Modules

• Model Design

• Hardware Implementation

• Constraints, Future Work, Conclusions and Recommendations

2. INTERNET OF THINGS

2.1 What is Internet of Things

The Internet of Things (IoT) is a term used to define the network of devices or physical objects, which are the 'things' in the name, that have technologies such as sensors, software and so on to connect and interact with other connected devices or systems by exchanging data through the internet. The connected devices vary from basic house objects such as a smart microwave to industrial tools such as self-driving cars. The connected devices has gone over 21.5 Billion by 2020 and predicted to reach 46 Billion in 2021. [3]

The importance of IoT comes from the exchange of data. The data exchange allows many various things both to the user and to the manufacturer. The data exchanged could allow the user to get feedbacks over the connected systems that would help the user in improving themselves while the manufacturer could use the the exchanged data from each device they produced to find the best configurations that could be made to the device and get the feedbacks directly. Due to most of the devices connected to the IoT network being wireless, the structure of IoT is difficult to manage. Therefore to make designs for IoT devices one should understand the architecture behind the IoT systems. IoT architecture will be briefly explained in the next section.

2.2 IoT Architecture

The IoT development differs regarding to the technologies used, implementation areas and business aspect. Many different IoT architectures are present to design the IoT devices of need. Nevertheless the 5 Layer Architecture of IoT is considered to be the best among all other proposed architectures for IoT design. [4]

2.2.1 5 Layer Architecture

The 5 Layer Architecture of IoT is considered to be based on the basic architecture of IoT with 2 additional layers. Every layer of IoT will be explained in the subsections below. The 5 Layer Architecture can be seen in the figure 2.1.

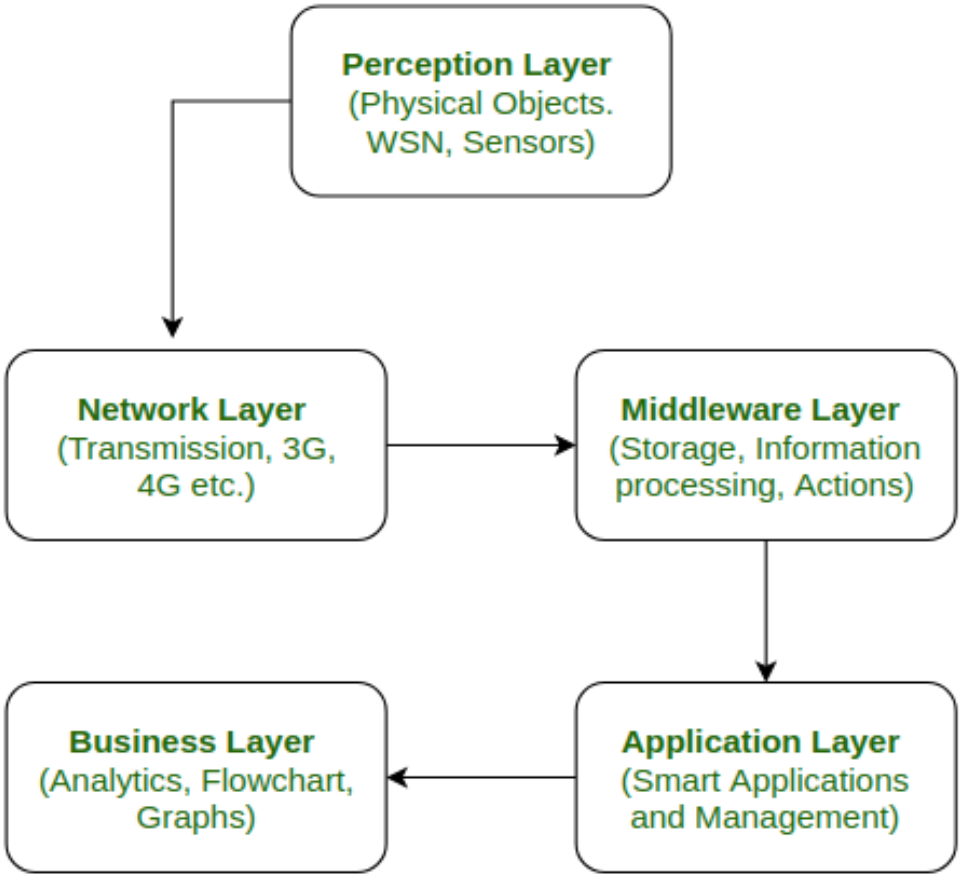

Figure 2.1: 5-Layer IoT Architecture

2.2.2 Perception Layer

The first layer of the 5-Layer architecture is called Perception Layer, that can also be called as the physical layer, which is responsible for gathering data such as temperature, moisture, sounds etc with the sensors or actuators. Basically the Perception Layer is responsible for gathering information from the surrounding area and send the gathered data to another layer where the data will be processed.

2.2.3 Network Layer

Second layer is called the Network Layer which can also be called as the communication layer due to being responsible for the communication that happens between perception and middleware layer Network Layer takes the data from the perception layer and, while taking the security of the data into account to keep the confidentiality of it, transfers it to the middle layer by utilizing technologies as 3G, 4G, UTMS, WiFI, ZigBee, Infrared etc.

2.2.4 Middleware Layer

Middleware Layer includes storatge cumpation, processing and action taking capabilities which can be called as advanced features. Middleware Layer is responsible for the storage of the data-set and give appropriate data to the device that the data is taken from. It also has the capability to make decisions by the evaluation of the data-set that is taken from sensors. The storage of taken data-set is done by taking device adress and name into consideration.

2.2.5 Application Layer

Application Layer is responsible for management of application processes from the data gathered from the Middleware Layer. Examples of application processes can be counted as activating microwave, powering or shutting down devices, smart agriculture etc.

2.2.6 Business Layer

Business Layer is responsible for how the produced technology is obtained by the user which is an important part of a design process since without a good service an amazing technology could not be seen successful thus makes the business layer is an important part of the IoT Architecture. Business Layer's tasks include of creating flowcharts, graphs, analysis of results and improvements for the device etc. [4]

As can be seen from the architecture above, IoT systems have applications in many areas around the world which are given in examples in the next section below.

2.3 Applications of IoT

2.3.1 Consumer Products

Many smart devices used in daily life falls under this category, as the name suggests. Examples of these products can be given as smart devices such as smartwatches, smart microwaves and other kitchen appliances, security cameras and so on.

2.3.2 Industrial Machinery

The automation in industry began with Industrial Revolution [5] and continous to this age. Replacing human power with machines aside from the quickness and continous working capability of machines, while saving time and energy, machine usage lowers the risk of human error and cost during the production as well. Nowadays IoT devices are in many areas of the economy due to easy manage of industrial machinery and improve productions and profit. [6]

2.3.3 Infrastructure

Integration and utilization of IoT systems in structures allows various simplification and efficiency to the users. The efficiency they bring with them allows IoT systems to be a possible must in the structures that people use, from cities to urban areas. Smart homes are commonly known application of IoT infrastructures.

2.3.4 Management and Logistics

This category involves many business areas that are IoT based or utilizes IoT. A simple example can be given for Logistics companies which are generaly utilizes vehicles, ships, aircraft and all types of machinery. The management of a large company fleet could be simplified by IoT system integration. With the integration of IoT systems on the vehicles also the cost of operation could be lowered by acting on the taken data from the IoT systems thus it could potentially enlarge the profit of the company. Due to all this, the prediction for IoT Management Market value by 2025 is USD 16.86 billion. [7]

Thanks to all these applications IoT systems ease the daily life of an average human and increases the profit of companies. However, IoT systems also opens the window for security issues, that is detailed in the next section.

2.4 IoT Security

The IoT devices' advantages comes from having connection to the internet. That allows them to exchange data with other systems that allows them to improve for user's benefit. The internet advantage of IoT could become the weakness of it due to cyber attacks. The internet connection creates an opportunity for hackers to breach into the IoT systems that could allow the hackers to manipulate or divert data according to their own will. The calamity of this could range from a basic device camera hacking that would affect a person's life to shutting down an entire city's power grids to cause thousands of people to lose their electricity. The prospects of it is unlimited. A possible way to prevent cyberattacks or make their systems harder to breach users can update their firmwares in their smart devices. Yet according to Bitdefender, 60% of Americans have never updated the firmware of their internet routers. [8]

As the number of IoT devices increase the security problem also becomes a bigger issue since more potential devices to hack are proposed in the system and without the right security systems they are vulnerable for cyber attacks. Use of high security cryptography algorithms in IoT designs could lessen the vulnerability of cyber attacks.

3. UTILIZED PROGRAMS

In this paper various simulation programs are utilized to achieve more accurate and reliable simulation results of the designed system. The programs are detailed in the following part of this section.

3.1 MatLab by Mathworks

Matlab is a program that is used by more than 6500 Colleges and universities with more than 4 millions of users in the whole world. [9] Many colleges and industries chooses to benefit MatLab as a tool for educational or research purposes. Yet the program was not always as sophisticated and full of features as present day. MatLab program when first created was a basic matrix calculator which did not include any programs nor toolboxes, or even the widely used FFTs. [10] As can be seen in the figure 3.1, a 1981 version of Matlab only included 71 words and functions.

```
        < M A T L A B >
     Version of 05/12/1981
<>

 The functions and commands are...
 ABS    ATAN   BASE   CHAR   CHOL   CHOP   COND   CONJ
 COS    DET    DIAG   DIAR   DISP   EIG    EPS    EXEC
 EXP    EYE    FLOP   HESS   HILB   IMAG   INV    KRON
 LINE   LOAD   LOG    LU     MAGI   NORM   ONES   ORTH
 PINV   PLOT   POLY   PRIN   PROD   QR     RAND   RANK
 RAT    RCON   REAL   ROOT   ROUN   RREF   SAVE   SCHU
 SIN    SIZE   SQRT   SUM    SVD    TRIL   TRIU   USER
 CLEA   ELSE   END    EXIT   FOR    HELP   IF     LONG
 RETU   SEMI   SHOR   WHAT   WHIL   WHO    WHY
```

Figure 3.1: Functions in 1981 version of Matlab

Improvements on MatLab have been made over the years where the present day MatLab which involves many toolboxes, functions and a user-friendly environment is created. The Modern Day MatLab is a high-performance language that is being used in many areas for technical computing. MatLab has a wide use in mathematical computations, algorithms, designing and simulating models, analysing and visualising data, plotting graphics and so on. MatLab's toolbox environment, which is highly rich as said above, provides the users to be able to use many different designs and technologies easily. Many areas from signal processing to neural networks have available toolboxes to be used in the MatLab environment. An example for plotting is given in figure 3.2.

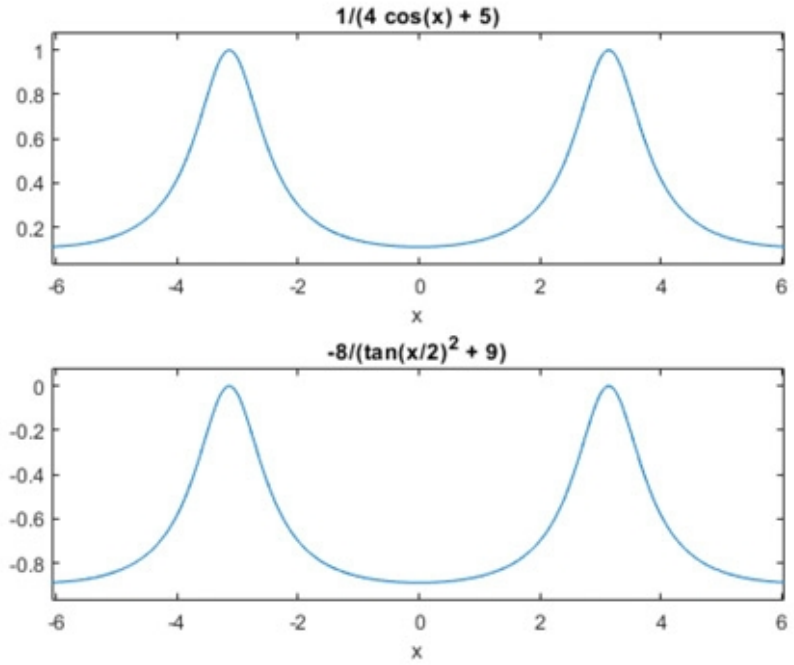

Figure 3.2: A plot of functions by MatLab

In the project MatLab's computational side is widely used to design and edit functions before they were implemented in the simulink environment, which is also a MatLab feature that will be explained in detail in the next section.

3.2 Simulink

Simulink is a simulation and model design software that is a feature in the MatLab program. Simulink software allows the user to design a system or a program by models and do the model simulations. As MatLab is, Simulink is also a software that is created by MathWorks. Its massive library allows the user to model many designs. Furthermore the Simulink supports custom designed block libraries which allows the users to share their design and implement the needed blocks in their own model. Simulink also supports Coder softwares such as Simulink Coder and Embedded Coder, which is a tool of Simulink that is widely used while designing the models in this paper, to generate C or embedded codes respectively, of the designed models automatically. Some of the applications for Simulink is given with the library browser in the figures 3.3 and 3.4 respectively.

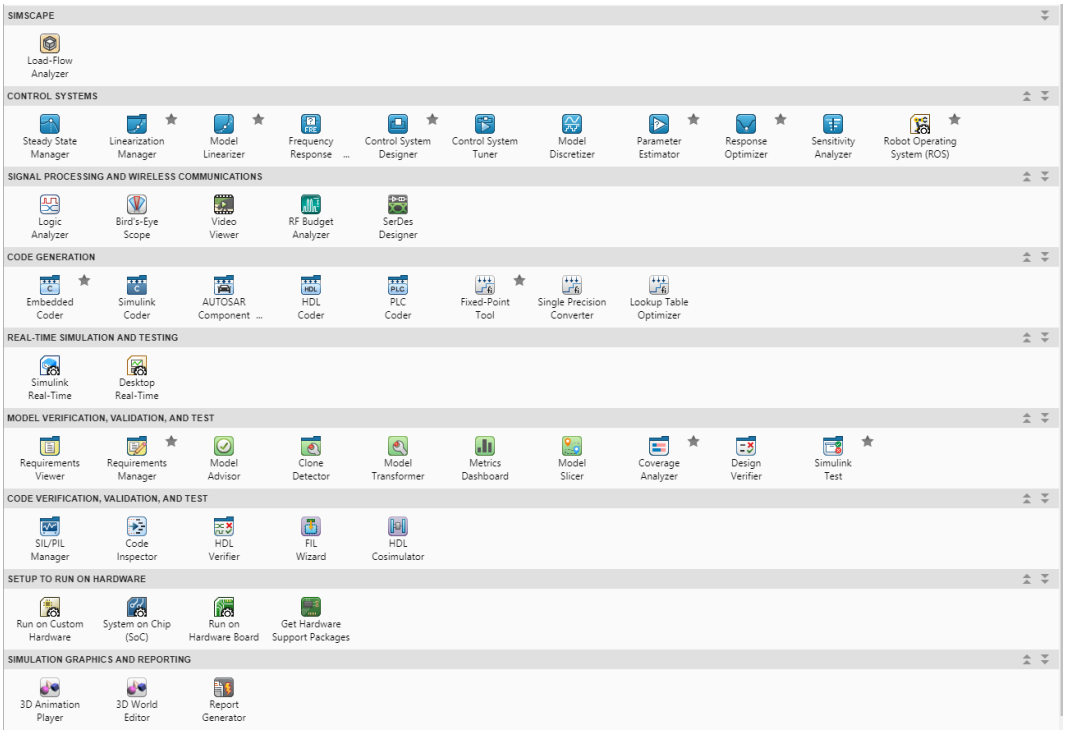

Figure 3.3: Simulink Applications List

The Embedded Coder feature of Simulink is the most widely used simulink feature in this paper. The Embedded Coder is designed to generate C and C++ code from the model for the embedded processors with the aim of producing a readable and fast code. [11] The Embedded Coder uses high performance optimization methods

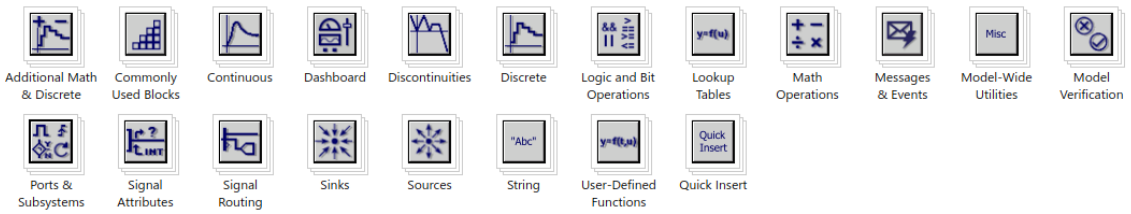

Figure 3.4: Simulink Library Browser

to maintain generated functions and data with excellent control. Due to these the produced code has higher efficiency. The Embedded Coder also provides options to select processors from or to create a C program for RAM or Execution efficiency and so on.

3.3 Xilinx Vivado

Xilinx Vivado Design Suite is a program that is developed to be used as an Integrated Development Environment (IDE) that has system-to-IC level tools which have been designed with a shared scalable data model and a common debug environment. Vivado has the ability to synthesize and analyse HDL designs. It has replaced the Xilinx ISE with having an in-built logic simulator ISIM [12] and by adding extra features to system on a chip[eejournal] developing with high-level synthesis [13]. Vivado also has Electronics System Level (ESL), standards based packaging for the reuse of the algorithmic and RTL IP, standards based IP stitching with the integration of systems for system building blocks for all kinds with ability to verify blocks and systems. [14] With High-Level Synthesis that came with Vivado, also a toolchain to create programmable logic from C code was made available. Aside from all these features of Vivado Microblaze [15] feature of Vivado is one of the most utilized features of Vivado in the duration of this project which is benefitted to simulate the design in a real life environment. Microblaze is an RISC Harvard architecture, that is produced by Xilinx for 32/64-bit, soft processor core that includes a rich instruction set that is made applicable for embedded applications. Microblaze soft-processor gives the user a wide range of options and feature selection from peripheral, memory and interface which grants the user to create the needed system with the lowest cost that can be reached on an FPGA. The design of Microblaze is detailed in the Microblaze Creation section.

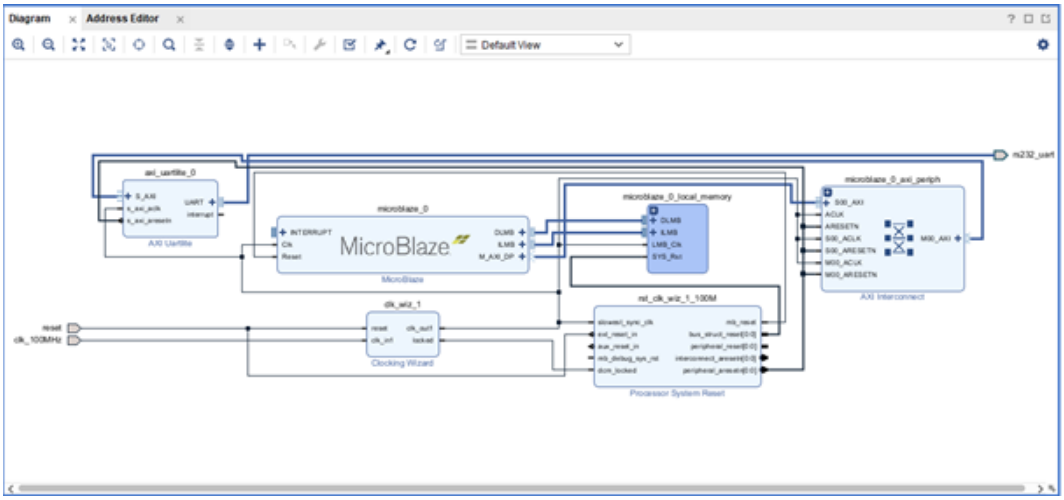

Figure 3.5: Microblaze Schematic in Vivado

3.4 Vitis

Vitis is a feature presented in the new versions of Vivado Design Suite which is also produced by Xilinx. It offers the user the ability to be able to write the software side of the FPGA programming. Vitis can be called as a combination of various Xilinx tools. Some of those tools are:

• Xilinx SDK, which is a tool to write C/C++ codes for the processor that is in the model which is designed in Vivado environment to run on. The produced code from Vitis is usually found to be in use for the configuration and control of the hardware model elements in the very least. The use of The produced code from Vitis is generally used for the configuration of the hardware model's elements and to control them efficiently since debugging and rebuilding the code is much simpler than the hardware.

• Vivado HLS tool provides a feature to allow the user to create blocks which have built-in C codes written by the user that are addable in Vivado projects. The created blocks are generally reusable thus provides the user to use the same block for various projects.

• SDSoC tool allows the user to create and process a software design to be built inside the hardware, which in other terms takes in a Vivado design that is created in advance to associate it with the written C/C++ code which has functions that are

used to process the data. This allows the user to build a hardware system by writing in software languages.

Vitis' hardware programming side is heavily used in the designed project for the results part to prove the created design is usable with processors.

Figure 3.6: Vitis Logo

4. UTILIZED ALGORITHMS AND MODULES

The project consisted of trans-receivers and encryption modules. For this purpose Zigbee module is selected for transmitting and receiving the data. Also for security measurements, AES Encryption model is used. In the following of this section both are explained.

4.1 Zigbee Module

Zigbee is an open global standard that is created with wireless technology to fullfil the needs of low cost and low power wireless IoT networks. It uses IEEE 802.15.4 physical radio specification with operations in the unlicensed bands which also involve the 2.4 GHz, 900 MHz and 868 MHz. [16] The protocol Zigbee operates in is intended for the battery powered devices with low cost. The Zigbee protocol was created by the comprise of many leading electronics companies from semiconductor manufacturers to service companies, which have created an alliance called Zigbee Allience. [16] The aim was to create a protocol, which already has the aim for low cost and low power, with secure wireless network architectures to ensure a wireless data solution with simple usage. Due to providing long battery life with low cost Zigbee technology is used in many areas from Lighting controls to Medical devices. The figure 4.1 shows various devices Zigbee protocol can be used in at a smart home.

4.2 Advanced Encryption Standard

Advanced Encryption Standard (AES) is the new encryption standard that came in DES' place in October 2000 after an encryption algorithm competition made by National Institute of Standards and Technology (NIST) in 1997. [17] The DES has been losing its reliability due to having a 64 bit key space thus a search for a better algorithm to take its place has been launched by NIST where the winner algorithm

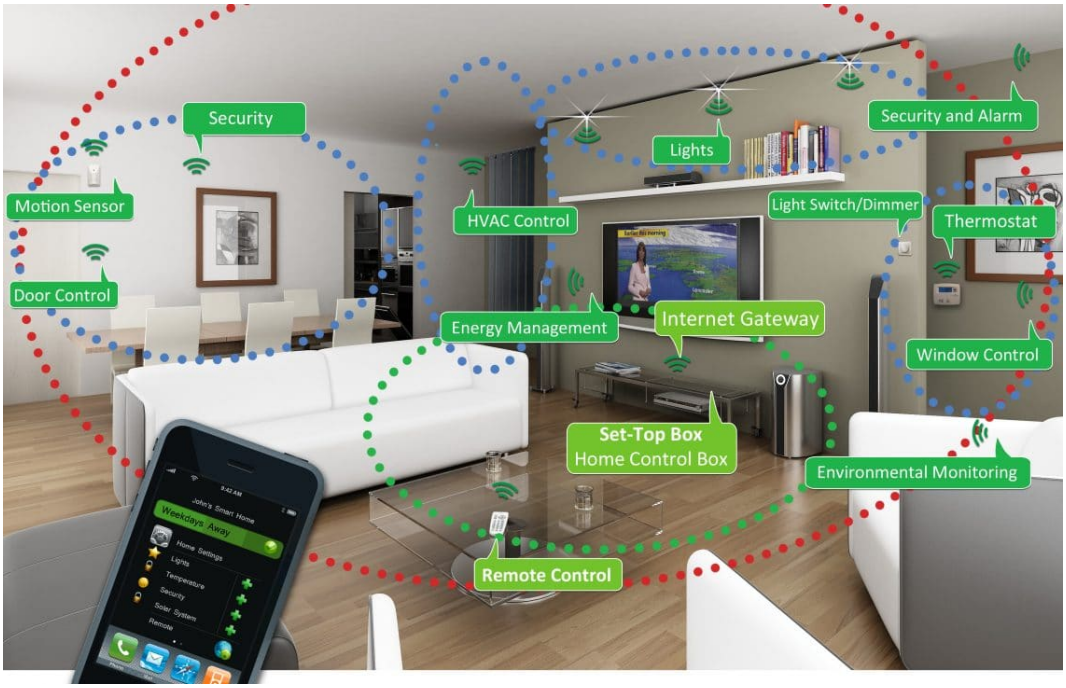

Figure 4.1: Smart Devices that Utilize Zigbee Protocol

would be named as AES. The competition lasted for more than 3 years and in the end the algorithm Rijndael was selected as the new AES by NIST.

4.2.1 Algorithm

The AES algorithm is a standard that is still used to this day with high reliability due to having 128 bits for inputs, outputs and matrixes. Matrixes are called “state” in the algorithm where for each 4x4 division of the matrix it holds 1 byte of data. The main feature of AES algorithm lies in the key options it offers. The key of AES can be 128, 192 or 256 bits which highers the reliability and security of the encrypted data as it increases. The difference between key lengths is they change the loops of security algorithm as it increases, where it also increases the number of data. The Table 4.1 below shows how the AES key length affects the loops.

	Number of Rounds
AES-128	10
AES-192	12
AES-256	14

Table 4.1: Number of Rounds for AES Types

The algorithm consists of 4 sub functions that are used iteratively. The functions can be listed as AddRoundKey, Byte Substitution (SubBytes), ShiftRows and Mixcolumns. After getting the 128 bits of plain text the sub functions are used according to the number of rounds, which is linked to the key size given in Table. A basic illustration of how AES structure processes is given in figure 4.2.

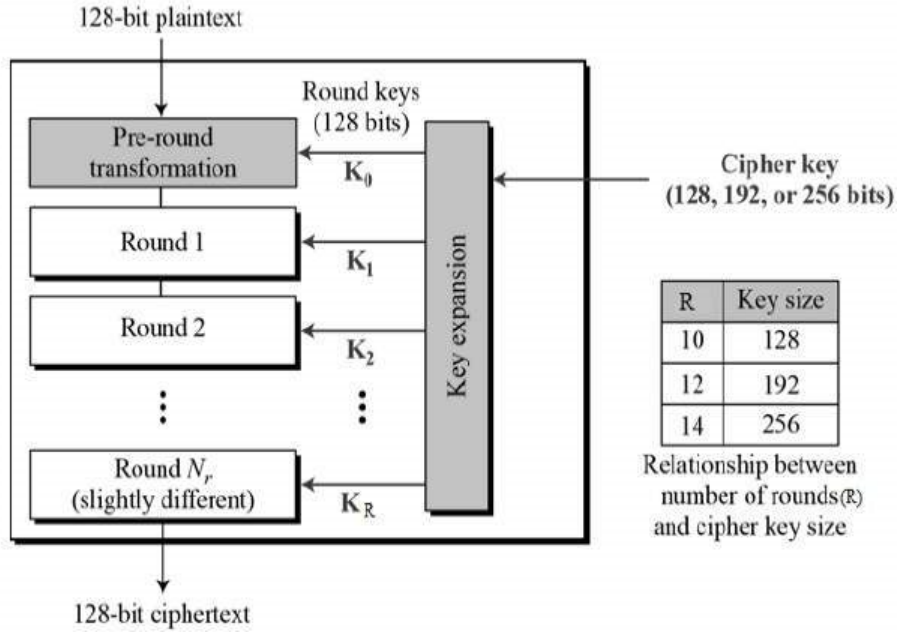

Figure 4.2: AES Algorithm Processing

4.2.2 Encryption Process

To understand how the encryption process of AES Algorithm is done, first the sub functions are needed to be understood. To understand what is done in each round the figure 4.3 can be used while sub functions are analysed.

4.2.2.1 S-Box

S-Box, which comes from the Substitution, is used generally in symmetrical encryption methods where it decides on what value the substitution is going to be made with. The S-Box is a fixed 2D table where it is divided into even and odd bits. An example of S box is given in figure 4.4. For an input given as 1010 the even bits are 00 (1st and 3rd bits) and odd bits are 11 (2nd and 4th bits). From the table the found value for 00 row and 11 column will replace the even bits in the original number which is found as 11 therefore the substituted value will be 1111.

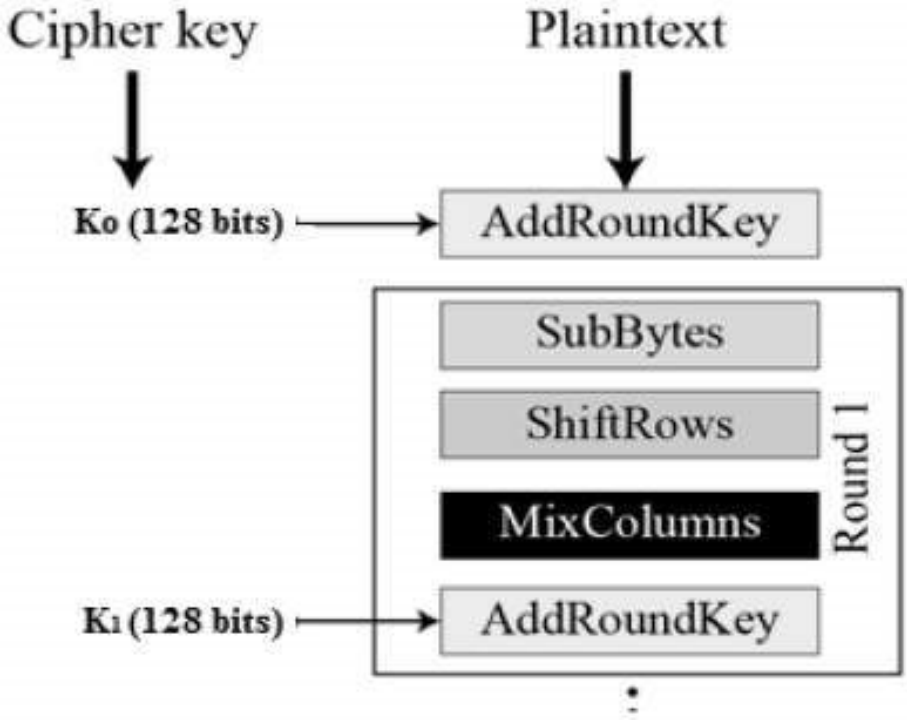

Figure 4.3: AES Algorithm Rounds

4.2.2.2 State Matrix

The state matrix is created by dividing the taken input into 128 bits of pieces and assign it to the state matrix. After that the AddRoundKey is processed to achieve the State Matrix which then used in the functions to create a safe encryption.

4.2.2.3 SubBytes

The SubBytes function is as the name suggests, substitutes elements in state matrice with the elements in the fixed S-Box, which is a 16x16 for 128-Bit applications, created prior to function call.

4.2.2.4 ShiftRows

The shift row function shifts the rows on the matrixes to the left and adds the exceeding values to the LSB. The shifting happens in a circular shape where the first row does not shift, second row shifts by one bit, third one shifts by two and fourth row shifts by three. All the rows shift with respect to each other.

```
function state = SubBytes(state)
Sbox =[

    99   202   183     4     9    83   208    81   205    96   224   231   186   112   225   140
   124   130   253   199   131   209   239   163    12   129    50   200   120    62   248   161
   119   201   147    35    44     0   170    64    19    79    58    55    37   181   152   137
   123   125    38   195    26   237   251   143   236   220    10   109    46   102    17    13
   242   250    54    24    27    32    67   146    95    34    73   141    28    72   105   191
   107    89    63   150   110   252    77   157   151    42     6   213   166     3   217   230
   111    71   247     5    90   177    51    56    68   144    36    78   180   246   142    66
   197   240   204   154   160    91   133   245    23   136    92   169   198    14   148   104
    48   173    52     7    82   106    69   188   196    70   194   108   232    97   155    65
     1   212   165    18    59   203   249   182   167   238   211    86   221    53    30   153
   103   162   229   128   214   190     2   218   126   184   172   244   116    87   135    45
    43   175   241   226   179    57   127    33    61    20    98   234    31   185   233    15
   254   156   113   235    41    74    80    16   100   222   145   101    75   134   206   176
   215   164   216    39   227    76    60   255    93    94   149   122   189   193    85    84
   171   114    49   178    47    88   159   243    25    11   228   174   139    29    40   187
   118   192    21   117   132   207   168   210   115   219   121     8   138   158   223    22];
state=Sbox(state+1);
end
```

Figure 4.4: SubBytes Example with S-Box

```
function state = ShiftRows(state)
state(2,:)=circshift(state(2,:),[0 -1]);
state(3,:)=circshift(state(3,:),[0 -2]);
state(4,:)=circshift(state(4,:),[0 -3]);
end
```

Figure 4.5: ShiftRows Function

4.2.2.5 MixColumns

This process uses the old elements of the columns to create new columns. The calculation involves constant a(x)=3x3+x2+x+2 where if the old column's 1st element is S1(x) and the new one is S1'(x) then the function for the new element would be S1'(x)=a(x)*S1(x) which would be repeated until the whole matrix is changed with the new columns. It's not repeated in the last round.

```
function State = MixColumns(state)
State=state;
for a=1:4:13
    State(a)=bitxor(bitxor(bitxor(xtime(state(a),2),xtime(state(a+1),3)),state(a+2)),state(a+3));
    State(a+1)=bitxor(bitxor(bitxor(xtime(state(a+1),2),xtime(state(a+2),3)),state(a)),state(a+3));
    State(a+2)=bitxor(bitxor(bitxor(xtime(state(a+2),2),xtime(state(a+3),3)),state(a)),state(a+1));
    State(a+3)=bitxor(bitxor(bitxor(xtime(state(a+3),2),xtime(state(a),3)),state(a+1)),state(a+2));
end
end
```

Figure 4.6: MixColumns Function

4.2.2.6 AddRoundKey

At the end of each round the final state is created by the XOR of each element in the State matrix that is created after the functions and the Roundkey. Both being 128 bits creates another 128 bits matrix.

```
function state = AddRoundKey(statel,w)

state=zeros(4,4);
state=statel;
for k=1:4
    state(:,k)=bitxor((state(:,k)),(w(:,k)));
end
end
```

Figure 4.7: AddRoundKey Function

4.2.2.7 KeyExpansion

The each round key is produced by the system right after the key and input is taken by the system to be used in each round. For a 128 bit key 10 different key is produced. The process of key producing happens cumulatively where each round uses the key that is produced before it. The producing of a new round key is basically adding columns. The process happens as for the new column of the new key, adding the previous and 4 previous column together and add it to the new column where for a 4x4 matrix, it's just adding the previous column and current column together.

The only exception happens at whenever the multiples of 4 is going to be addeed. Before that happens, the multiple of 4th column goes through a process called T Process which can be listed as; a circshift function to perform a one-byte circular left shift, SubBytes to perform a substitution by S-Box and XOR with a round constant Rconst(x) value where x indicates the round number which is defined over GF(28) field as Rconst(x)=(Rc(x),0,0,0) where is defined as Rc(x)=2*Rc(x-1) and Rc(1)=1. An example value table 4.2 of Rc(x) in table for 10 rounds in hexadecimal.

x	Rc(x)
1	01
2	02
3	04
4	08
5	10
6	20
7	40
8	80
9	1B
10	36

Table 4.2: Rc(x) Values for x

4.2.3 Encryption Steps

The Encryption steps of AES begins by creating a State Matrix by dividing the input into 128 bits pieces then calling AddRoundKey then go into the next steps until the rounds are completed:

• Call SubBytes on the State Matrix to Substitute Elements

• Call ShiftRows to change the places of the elements

• Call MixColumns (Skipped if last round) to create different columns

• Call AddRoundKey

• Repeat if round number is not reached

At the end of the rounds the CipherText is given as an output which is then available to be used in decipher with the same key to achieve the original text, which is actually the opposite process of Encryption. In the Cipher Function in listing A.2 a function named "ara_func" is defined to gather all the functions to represent one round.

```
function w = KeyExpansion(key,Nk)

w=zeros(4,4*(Nk+7));
wl=double(reshape(key,4,[]));
[p,q]=size(wl);
for j=1:p
    for k=1:q
        w(j,k)=wl(j,k);
    end
end

for i=Nk:4*(Nk+7)-1
    temp=w(:,i);
    if mod(i,Nk)==0
        temp=SubBytes(circshift(temp,-1));
        n=1;
        m=0;
        while m<i/Nk-1%needed to modulate higher powers of 2 per standard
            n=xtime(2,n);
            m=m+1;
        end
        temp=bitxor(temp,[n,0,0,0]');
    elseif Nk>6 && mod(i,8)==4
        temp=SubBytes(temp);
    end
    w(:,i+1)=bitxor(w(:,i-Nk+1),temp);
end
end
```

Figure 4.8: KeyExpansion Function

4.2.4 Decryption Process

Decryption process is basically doing the opposite of what is done in the encryption process. The functions in the encryption step are inversed to achieve the reverse order thus they act opposite of what they did in the encryption step, as they will be explained below.

4.2.4.1 Inverse ShiftRows

The same process applies to this function as in ShiftRows, where only difference lies in the direction it shifts changes to right instead of left.

4.2.4.2 Inverse SubBytes

A change in S-Box is done before processing with SubBytes at the beginning on decipher where the S-Box is changed according to produce the first input back when used. Then Inverse SubBytes is used as in SubBytes to proceed.

```
function state = InvShiftRows(state)
state(2,:)=circshift(state(2,:),[0 1]);
state(3,:)=circshift(state(3,:),[0 2]);
state(4,:)=circshift(state(4,:),[0 3]);
end
```

Figure 4.9: Inverse ShiftRows Function

```
function state = InvSubBytes(state)
Sbox =[

    82   124    84     8   114   108   144   208    58   150    71   252    31    96   160    23
     9   227   123    46   248   112   216    44   145   172   241    86   221    81   224    43
   106    57   148   161   246    72   171    30    17   116    26    62   168   127    59     4
   213   130    50   102   100    80     0   143    65    34   113    75    51   169    77   126
    48   155   166    40   134   253   140   202    79   231    29   198   136    25   174   186
    54    47   194   217   104   237   188    63   103   173    41   210     7   181    42   119
   165   255    35    36   152   185   211    15   220    53   197   121   199    74   245   214
    56   135    61   178    22   218    10     2   234   133   137    32    49    13   176    38
   191    52   238   118   212    94   247   193   151   226   111   154   177    45   200   225
    64   142    76    91   164    21   228   175   242   249   183   219    18   229   235   105
   163    67   149   162    92    70    88   189   207    55    98   192    16   122   187    20
   158    68    11    73   204    87     5     3   206   232    14   254    89   159    60    99
   129   196    66   109    93   167   184     1   240    28   170   120    39   147   131    85
   243   222   250   139   101   141   179    19   180   117    24   205   128   201    83    33
   215   233   195   209   182   157    69   138   230   223   190    90   236   156   153    12
   251   203    78    37   146   132     6   107   115   110    27   244    95   239    97   125];

state=Sbox(state+1);
end
```

Figure 4.10: Inverse SubBytes Function and Inverse S-Box

4.2.4.3 Inverse MixColumns

For this function, only multiplied polynom a(x) is different than MixColumns function where for inverse MixColumns it is defined as a(x)=11x3+13x2+9x+14. The rest of the process is similar to MixColumns.

```
function State = InvMixColumns(state)
State=state;
for a=1:4:13
    State(a)=bitxor(bitxor(bitxor(xtime(state(a),14),xtime(state(a+1),11)),xtime(state(a+2),13)),xtime(state(a+3),9));
    State(a+1)=bitxor(bitxor(bitxor(xtime(state(a),9),xtime(state(a+1),14)),xtime(state(a+2),11)),xtime(state(a+3),13));
    State(a+2)=bitxor(bitxor(bitxor(xtime(state(a),13),xtime(state(a+1),9)),xtime(state(a+2),14)),xtime(state(a+3),11));
    State(a+3)=bitxor(bitxor(bitxor(xtime(state(a),11),xtime(state(a+1),13)),xtime(state(a+2),9)),xtime(state(a+3),14));
end
end
```

Figure 4.11: Inverse MixColumns Function

4.2.4.4 AddRoundKey

Since AddRoundKey is using XOR in the function, using XOR again with the same key would produce the original input on the given round thus the same function for Cipher Block is used with different inputs to initiate the function in reverse.

4.2.4.5 Inverse KeyExpansion

The last key in the Cipher is used to produce keys for decipher by doing the inverse process of KeyExpansion. To define how the process Works the columns from left to right are named w0 to w3. The process begins with adding w2 and w3 to produce new w3, then it goes as w1 and w2 and w0 and w1. As going in this cycle the new columns are created with only difference happens at w0 where the inverse T process needs to be added to the process where circular shift, S-Box and InvRconst(x), which includes InverseRc(x) function, is used on the w0 to produce the new w0 column. The inverseRconst(x) values for 10 round is given in the table 4.3 below in hexadecimal where x defines round number. In the project the same function from Cipher Block is used with different inputs to initiate the function in reverse.

x	Rc(x)
1	36
2	1B
3	80
4	40
5	20
6	10
7	08
8	04
9	02
10	01

Table 4.3: Inverse Rc(x) Values for x

4.2.5 Decryption Steps

The Decryption steps of AES begins also by creating a State Matrix by dividing the input into 128 bits pieces. Then calls AddRoundKey function as the Cipher block, but unlike Cipher block, since Decryption process works in reverse to Cipher, begins a round without an inverse MixColumns, just like in the Cipher's last step the round steps are repeated without MixColumns function. After the round is completed without Inverse MixColumns, the regular rounds begin as follows until the rounds are completed. The steps can be summarized as following:

• Call InverseMixColumns (Skipped if first round) to create different columns

• Call ShiftRows to change the places of the elements

• Call SubBytes on the State Matrix to Substitute Elements

• Call AddRoundKey

• Repeat if round number is not reached

Same as in Cipher Function, Decipher Function in listing A.3 has a function named "ara_func2" to define all the functions as one to represent one round as well.

At the end of the rounds the DecipheredText is given as an output which is the original PlainText.

5. MODEL DESIGN

The designed IoT system is aimed for use in intensive care rooms to check the vitals of the patients. The designed system in Simulink has 5 main area: Patient Rooms Area, AES Algorithm Area, Transmitter Area, Receiver Area and Output Area. In the following part these areas are explained respectively.

5.1 Patient Rooms Area

The subsystem that is responsible for patient data is designed to constantly gather the data of temperature and heart rate of the patients from each room of the intensive care unit. For simulation only 4 room is used. By using a selection system via 2 clocks that is integrated subsystem only generates the output for one room's data, while making the others zero. This way each cycle of clocks generates a different room data.

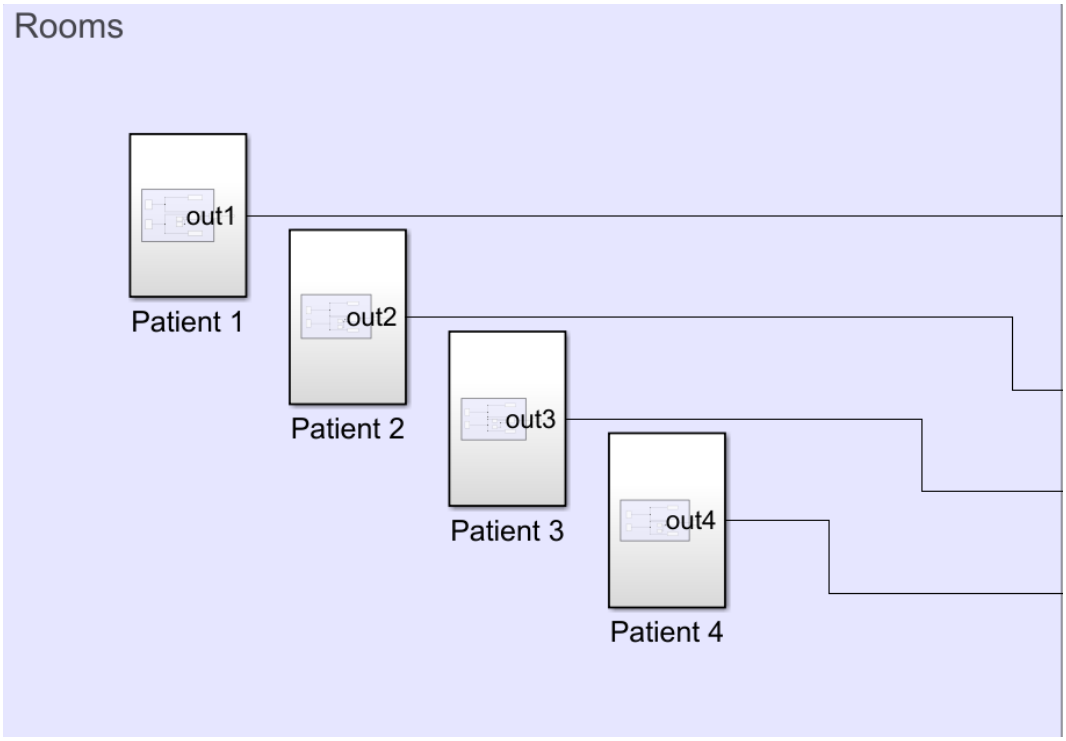

Figure 5.1: Patient Rooms Area

The subsystem of each patient involves 2 more subsystems and a MatLab function that controls the values generated by the subsystems. The subsystems are made to simulate real-life situations where sensors provide the data. The Temperature and Heart Rate subsystems generates a random temperature and heart rate data for simulation. The generated datas are controlled by a matlab function to generate appropriate output data for the patient by using clocks as enables.

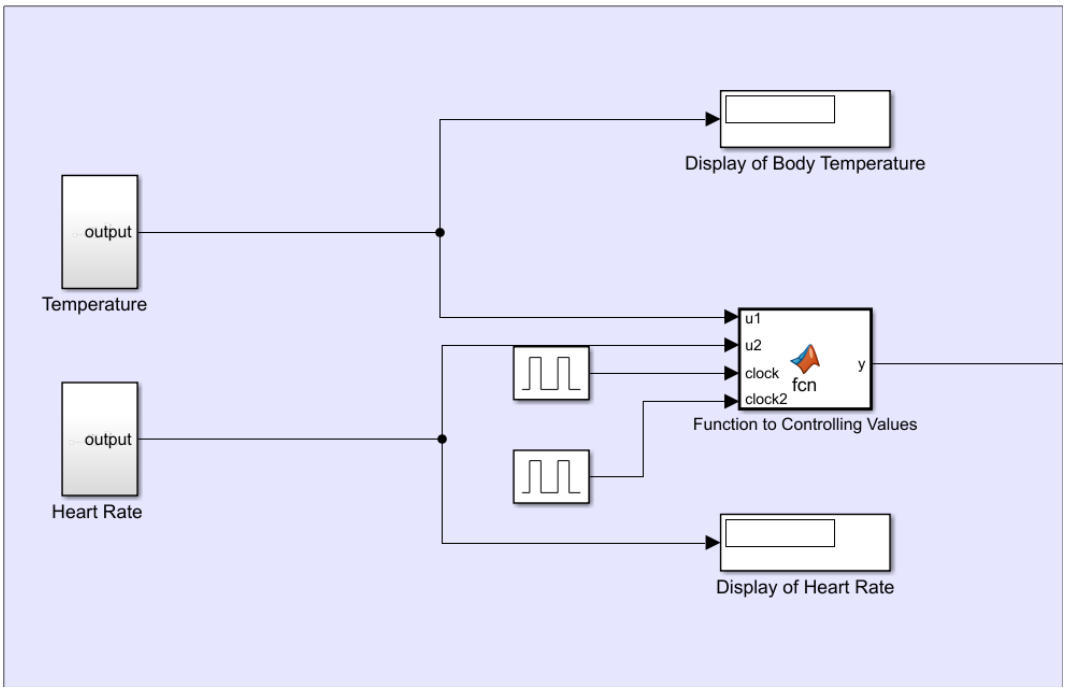

Figure 5.2: Patient Room Subsystem

5.1.1 Sensors

For the Temperature and Heart Rate subsystems same design is used. The design begins with a PS Random Number generator that generates a random data in the given range. For the data to properly controlled, a Controlled Temperature Source block is used. This block takes the input generated by PS Random Number from S which is the input that is going to be delivered to the output B, and A port is the thermal reference port which makes TA =0. The Temperature Sensor takes the output of Controlled Temperature Source block from input A. The port B is the opposite of port A which is positive heart. The output T is derived by using T = TA - TB that is also identical to the equation T = A.T – B.T. [18]. Due to this, to take the output as the input temperature A, B port is taken from the Thermal Reference thus makes TB=0. Due to Simscape library blocks a solver Configuration block is required to

be connected to the system as can be seen in figure 5.3. The output T is created as Kelvin by default. To create a system that works on Celcius a PS – Simulink Converter block is added to the system to convert Kelvin to Celcius. As said earlier the heart rate subsystem is also using similar design with only PS Random Number generator's range is changed to coincide with actual heart rate values.

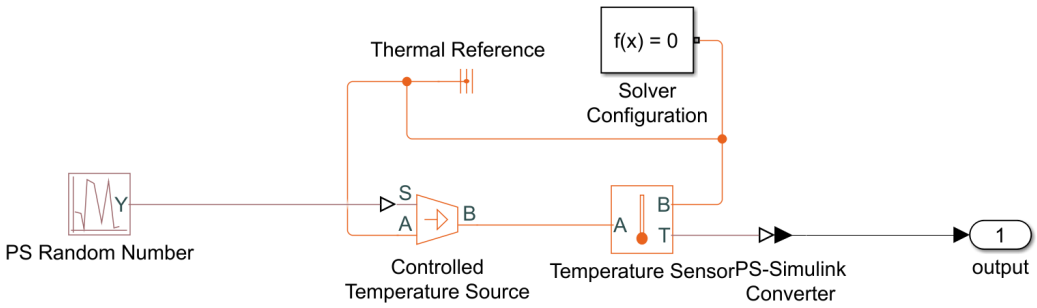

Figure 5.3: Temperature Subsystem

The parameters of the PS Random Number are given as can be seen in figure 5.4 for Temperature subsystem. After researching to find a suitable number range for Temperature and Heart Rate Subsystems for simulations an average of 37°C and 60 to 100 bpm is accepted as normal due to varying temperature between each humans. The Heart Rate range is the generally accepted range by the medical community. [19] The Temperature subsystem is designed to generate values between 30 and 44 to be able to simulate every possible situation, eventhough when 44 body temperature is reached the patient is long gone, the possibility remains thus the selection for parameters are made according to this in mind. Due to transferring each bit n 0.025 seconds, for 32 bits, the sample time is selected as 0.8 seconds. Same is applied for the Heart Rate subsystem with suitable parameter values.

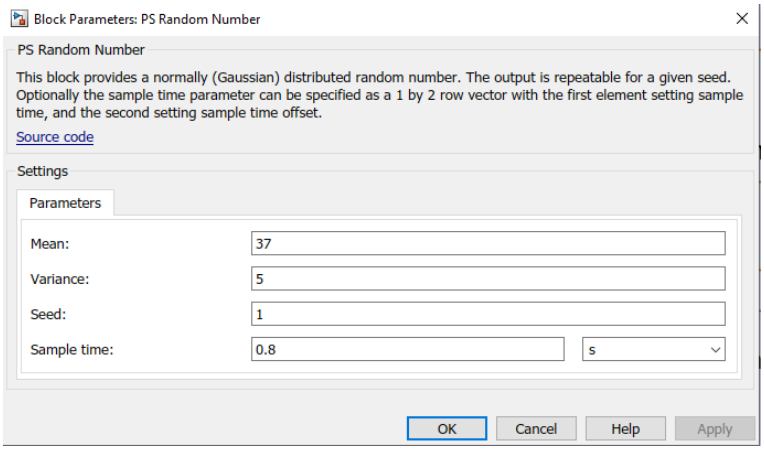

Figure 5.4: Parameters for PS Random Number Generator

5.1.2 Controller Function

To evaluate the taken inputs from the sensors an if-else function structure is created to decide the alarm level of taken inputs. As an example, when the patient with low temperature such as 34.5 and a heart rate of 60 bpm is taken from the sensors the output of the written structure would be y=248, which is the coded value for Emergency level which can be seen in the figure 5.5 below.

```
function y = fcn(u1,u2,enable,enable2)

if (36.5 < u1 ) && (u1 < 37.5) && ( 60 < u2 ) && ( u2 < 100) && (enable ==1)&& (enable2 ==1)
    %disp('Normal Level')
    y = 8;
elseif(35 < u1 ) && (u1 < 36.5) && ( 60 < u2 )&& ( u2 < 100) && (enable ==1)&& (enable2 ==1)
    %disp('Cold Level')
    y = 80;
elseif(36.5 < u1 ) && (u1 < 37.5) && ((u2 < 60) || (u2 > 100))&& (enable ==1)&& (enable2 ==1)
    %disp('Heart Rate Problem Level')
    y = 88;
elseif(37.5 < u1 ) && (u1 < 39)  && ( 60 < u2 )&& ( u2 < 100)&& (enable ==1)&& (enable2 ==1)
    %disp('Fever Level')
    y = 96;
elseif (35 < u1 ) && (u1 < 36.5)  && ((u2 < 60) || (u2 > 100)) && (enable ==1)&& (enable2 ==1)
    %disp('Cold and Heart Rate Problem Level')
    y = 168;
elseif (37.5 < u1 ) && (u1 < 39) && ((u2 < 60) || (u2 > 100)) && (enable ==1)&& (enable2 ==1)
    %disp('Fever and Heart Rate Problem Level')
    y = 176;
elseif(enable ==1)&& (enable2 ==1)
    %disp('Medical Emergency')
    y = 248;
else
    y = 0;
end
```

Figure 5.5: Function to Controlling Values

The selection of y values are done according to the UART [20] component, which can Show up to 255 due to having 8 bits, therefore the maximum value is selected according to this. Moreover to add every information into one data output the, the

binary convertion of the decimal output is coded. The 3 least significant bits of the binary counterpart of each decimal y value is to indicate which room the information is coming from. That way up to 8 patient's data can be transferred at once. The 2 most significant bits of the 8 bit is coded for the alarms. The 3rd to 5rd most significant bits are left for the levels of each situation. For example, the y=168 equals to 10101000 which corresponds to Room 1, Level 5, Alarm 2. Further explanation for the function will be made in the Output Area of the model.

5.2 AES Algorithm Area

For the purpose of designing an AES Cryptography for the model, search for an example file for the algorithm in MathWorks was done. An AES example with the function files [21] is found that has features of 128, 192 and 256 bits key length. For the purpose of the project, 128 bit key length is found to be enough to achieve a working and secure system. The AES Algorithm Area consists of 3 subsystems that are Data-Room Selection function, Cipher block and Decipher block which are seen in figure 5.6 respectively from left to right which will be explained in this order.

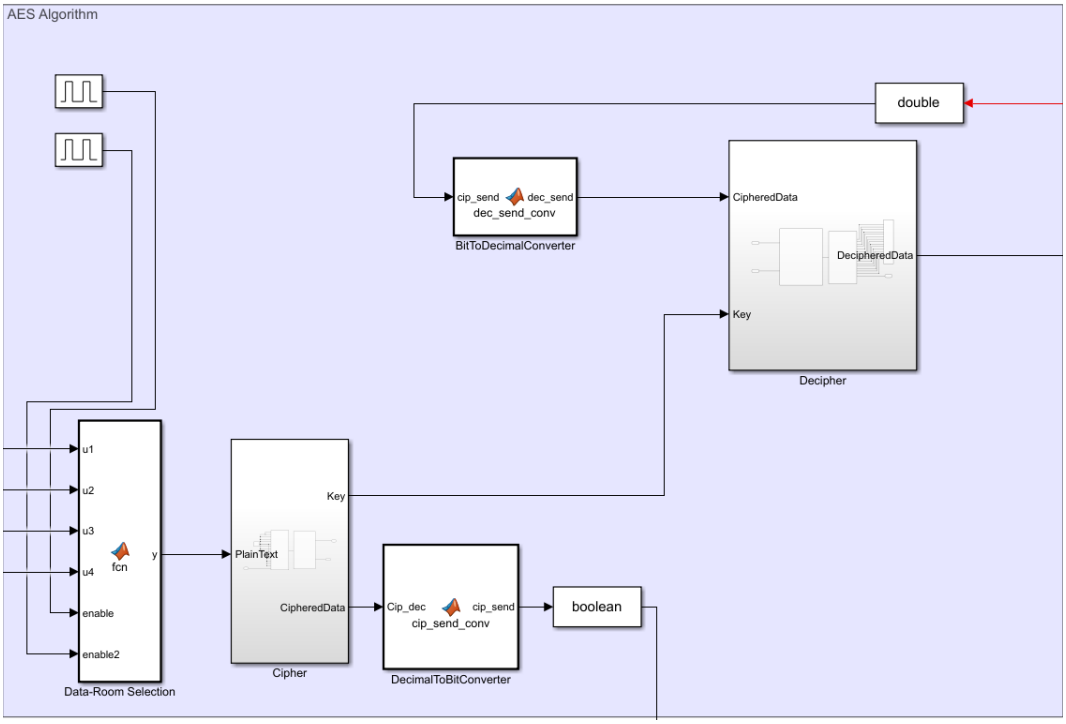

Figure 5.6: AES Algorithm Area

5.2.1 Data Room Selection

The Data Room Selection Function is similar to a multiplexer that processes and transfers the input datas in order by using clocks as enable signals. Since there are only 4 rooms in the model, 2 clocks were used in the model to achieve the ordering of data transfer. The clocks periods are set to 1.6 seconds and 3.2 seconds due to the 32 bit transferred to the receiver in 0.8 seconds. With the 4 data inputs that goes from u1 to u4 which are coming from the patient rooms and 2 enable inputs which are the clock inputs, the function block has 6 inputs in total. The only output of the function block is the y output which is the coded value of input datas that came from the patient rooms.

```
function y  = fcn(u1,u2,u3,u4,enable,enable2)

if (enable == 1)&&(enable2 == 1)
    y = u1;

elseif(enable == 0)&&(enable2 == 1)
    y = u2;

elseif(enable == 1)&&(enable2 == 0)
    y = u3;

else
    y = u4;

end
```

Figure 5.7: Data Room Selection Function

5.2.2 Cryptography

The AES Algorithm consists of 2 main blocks for the implementation as explained in the AES Algorithm section, which are Cipher and Decipher blocks. From the files that are taken from MathWorks the algorithm is verified to be usable in matlab environment. While implementing the algorithm files to Simulink there have been variable size and character output errors. Due to these modifications on the functions of the taken files needed to be made before transfering the functions into block form in Simulink. The changes in functions are explained in 3 parts: Changes for both block functions, Cipher Block Functions and Decipher Block Functions.

5.2.2.1 Changes for Both Block Functions

For both blocks, changes are made for key expansion function and input key. For simulation reasons the input key is implemented as a constant for both functions where it is converted from hexadecimal to decimal to prevent the errors that occur in Simulink implementation of the model. Later the key functions in both blocks are changed to be consistent with this change. Also to prevent variable-size errors, most of the variables are created first before being used in functions to prevent errors with assigning zeros matrixes for with the intended sizes for the 128-bit implementation of the model. Also due to the errors in simulink created by For loop, the for loop is written by hand with corresponding values for each loop for both Cipher and Decipher blocks.

5.2.2.2 Changes for Cipher Block Functions

In The Cipher Block Functions only S-Box in the SubBytes function is modified due to the convertion error of hexadecimal to decimal in simulink. The Convertion is made in simulink environment and the S-Box is re-written by hand for it's corresponding decimal matrix.

5.2.2.3 Changes for Decipher Block Functions

Since the model files that are found in MathWorks did not include an inverse key expansion function, the KeyExpansion function is used in reverse in Decipher block function to achieve an inverse key expansion. Also the inverse S-Box matrix in decimal is also written inside the invSubBytes function beforehand to prevent errors.

5.2.3 Cipher Block

The Key function in the Cipher block produces the key for both cipher and decipher blocks. Due to our key being 128 Bits the Cipher and Decipher number of Rounds is 10. The Encryption process begins with the Cipher function taking the input data. Since the algorithm used required 128 bits of input data where our input is 8 bits, a mux is used to create an 128 bits of input by adding 0s on the left side of the input number, thus making it an 128 bits integer. Due to simulink giving error, this method was found convenient.

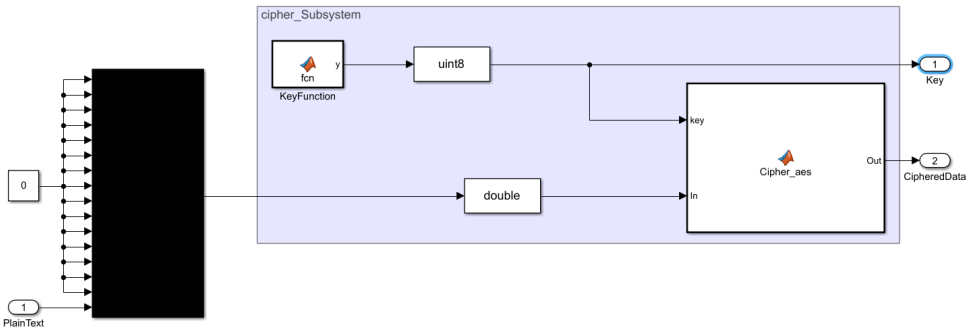

Figure 5.8: Cipher Block

The input is then given into the Cipher block where it is converted into a double to be used in the matlab function. The Cipher Function then takes the input and the key to calculate Nk to be used in KeyExpansion to produce w, round keys. Afterwards the State matrix is created and by AddRoundKey the key is added to the State matrix. The 10 rounds of encryption then begins by the order of steps mentioned in AES Algorithm section. First the State matrix is used in SubBytes function to create a substitution of elements. Later via ShiftRows the State function is shifted. Thirdly the MixColumns function is called to change the elements of the matrix. Lastly round key is added to the state matrix via AddRoundKey function. This Process is generated for 10 rounds. After the last round once again the SubBytes, ShiftRows and AddRoundKey functions are called. MixColumns function is not called in the last process. The output state is then converted into a 1x16 double vector to be given as an output from the Cipher Block. The Rounds of the Cipher is given in detail in AES Algorithm section.

5.2.4 DecimalToBitConverter

The Output of Cipher Block is a 16x1 double data. To be able to transfer via Zigbee Transmitter the output of Cipher Block needed to be converted into a binary number where bit by bit transfer is going to be executed by Zigbee Transmitter. To achieve this without any Simulink error the output of the Cipher Block is transferred into the DecimalToBitConverter Function to turn the encrypted data into a 1x128 binary vector form. The output data is in double format thus data type conversion block is used to convert the data into boolean and transferred into the bit serializer and transmitter area.

```
function cip_send = cip_send_conv(Cip_dec)
alan1=Cip_dec;
mat2=alan1';
binariil=zeros(16,8);
binariil=de2bi(mat2,8);
res=zeros(1,128);
res=reshape(binariil,1,128);
cip_send=zeros(1,128);
cip_send=res;
end
```

Figure 5.9: Decimal to Bit Converter Function

5.2.5 Decipher Block

The Decipher Block takes the input from the BitToDecimalConvertor function that takes the output of ShiftRegister and converts it into a 16x1 double to be used in the Decipher block. In the Decipher Block the key that is taken from the Key Function and encrypted data are taken as inputs to evaluate and generate the encrypted data. The process of Decipher is the basically a reverse cipher function where the Key Expansion is done in reverse to generate round keys. A state matrix is also created by the taken input as it was done in Cipher Function. But the process after this point is the reverse of what is done in Cipher Function where after adding roundkey to the state matrix the function begins the rounds according to the amount of the key length. The function's and their properties are detailed in the AES Algorithm section. The process begins with calling the Inverse ShiftRows function for state matrix. The following functions after the Inverse ShiftRows are InverseSubBytes and AddRoundKey. The first round of the Decipher Function does not processes Inverse MixColumns function. In the coming rounds of the decipher function the calling order of the functions is as follows: Inverse MixColumns, Inverse ShiftRows, Inverse SubBytes and AddRoundkey. Since the KeyExpansion is done in reverse which produced the round keys in reverse order, AddRoundkey can be used as it was used in Cipher Function. After the rounds of Decipher function finishes the output of the

function is given as a 1x16 double number. The number then is used in a data type conversion block to be translated into uint8 format. Nevertheless, since our input was a 128 Bit number where only the 8 least significant bits were holding the plaintext, the output of Decipher block is then transferred into a 1 to 16 Demux to take the original plaintext out of the 1x16 vector. The 16th output of Demux is then given as an output while the others are terminated to prevent warnings given by Simulink. The output of Decipher Block is then used in transferred to the Output Area to be used in the Screen Function.

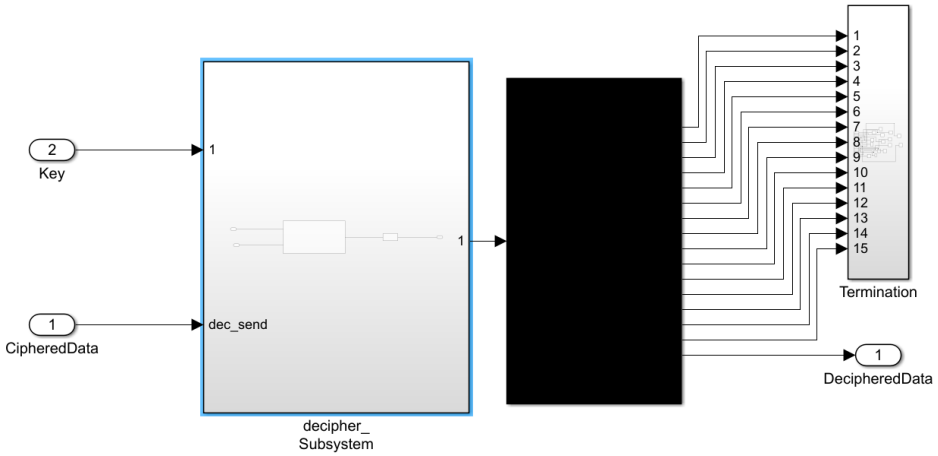

Figure 5.10: Decipher Block

5.3 Transmitter Area

After the data's encryption is completed, the encrypted data is transmitted to the receiver by a transmitter. Zigbee module [22], found from mathworks file exchange, is used for the transmitter and receiver parts to create the transmitter and receiver system blocks. The transmitter block of the Zigbee can be divided into three section: Bit serializer, which takes the input and transmits them bit by bit, the transmitter which is responsible for using Zigbee module to create a data to transmit and transmitter selection, which is responsible for which data to be sent, similar to a demux. Every room has the first 2 sections and all are connected to the Transmitter Selection function which decides which data to be transferred by it's enable selections. To transmit every room's data in a cycle enables are connected to clocks in Simulink.

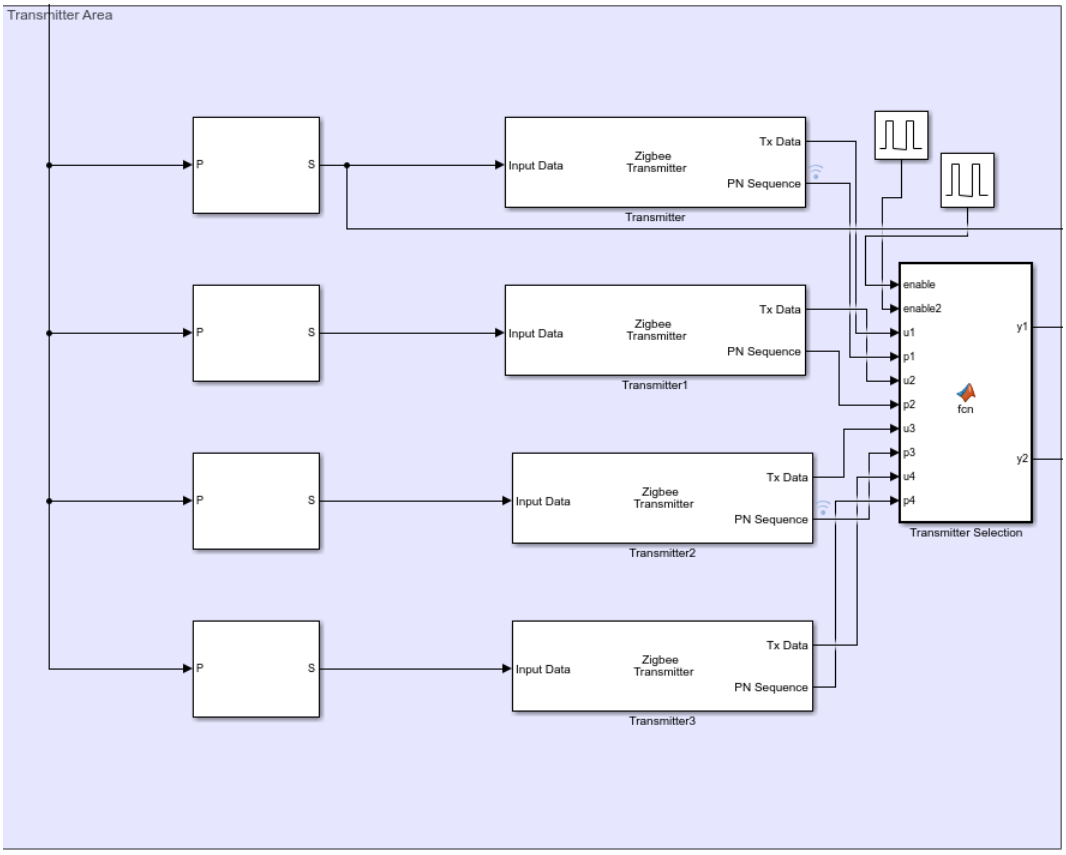

Figure 5.11: Transmitter Area

5.3.1 Serializer

The AES cipher produces a 16x1 decimal matrix which then proceeds to go into the decimal to bit converter to produce a 128bit binary number before going into the serializer as can be seen in figure 5.11. This is done to be able to transfer the decrypted text one by one via Serializer 1D. The output of the Serializer1D then connected to the input of the transmitter to make data eligible for transfer.

5.3.2 Zigbee Transmitter

In the figure 5.12 the inside of the transmitter subsystem can be seen. The transmitter has 2 outputs where the modulation is done via OQPSK block to produce the transmittable data and give it as an output from Tx Data. The sample time of the transmitter is fixed at 0.025 seconds.

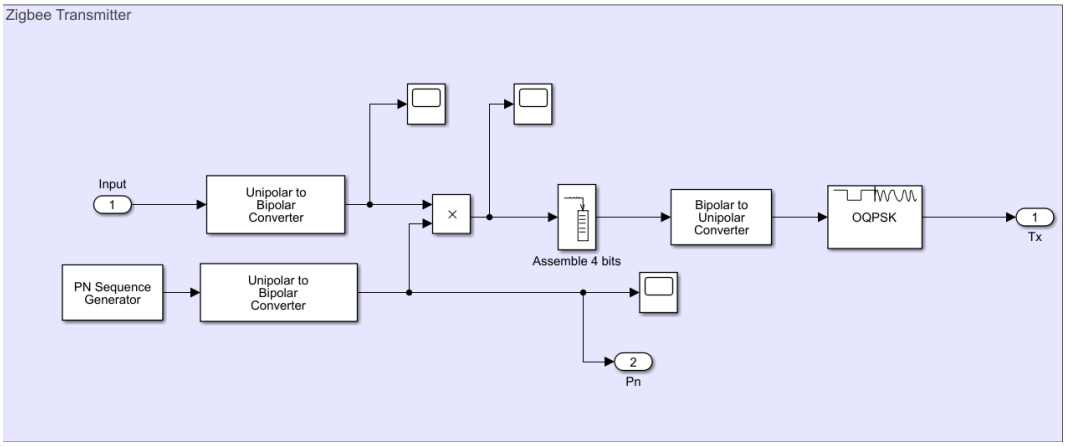

Figure 5.12: Zigbee Transmitter Subsystem

5.3.3 Transmitter Selection Function

Due to the model having only one receiver the outputs of the transmitters have to be in an order to prevent data mixing in the receiver side. Thus for the 4 room's 4 transistors, a controlling function module is written. As can be seen below the function acts as a multiplexer thus requires a selection signal, which are given as 2 clocks connected to the function as enables. A simple multiplexer could have been used yet due to errors it gives, this method is found more convenient.

```
function [y1,y2]= fcn(enable,enable2,u1,p1,u2,p2,u3,p3,u4,p4)

if (enable == 1)&&(enable2 == 1)

    y1 = u1;
    y2 = p1;

elseif(enable == 0)&&(enable2 == 1)

    y1 = u2;
    y2 = p2;

elseif(enable == 1)&&(enable2 == 0)

    y1 = u3;
    y2 = p3;

else

    y1 = u4;
    y2 = p4;

end

end
```

Figure 5.13: Transmitter Selection Function

5.4 Receiver Area

The figure 5.14 below shows the receiver area and the receiver's inside. The receiver area involves an AWGN block which is used to portray the noise between the transmitter and receiver which is an important factor in data transfer that has to be taken into account. The output data coming from the transmitter first goes through the AWGN before getting demodulated in the Zigbee Receiver.

Another block in the receiver area is Error Rate Calculation block which is used to calculate the error rate between the sent and demodulated data as the name suggests.

Last block is the Zigbee Receiver, which is seen in figure 5.15, is responsible for the demodulation of the received data. Since the data is received in 1 bit, the demodulated data will be 1 bit as well thus a concatenation is required before decrypting the taken data.

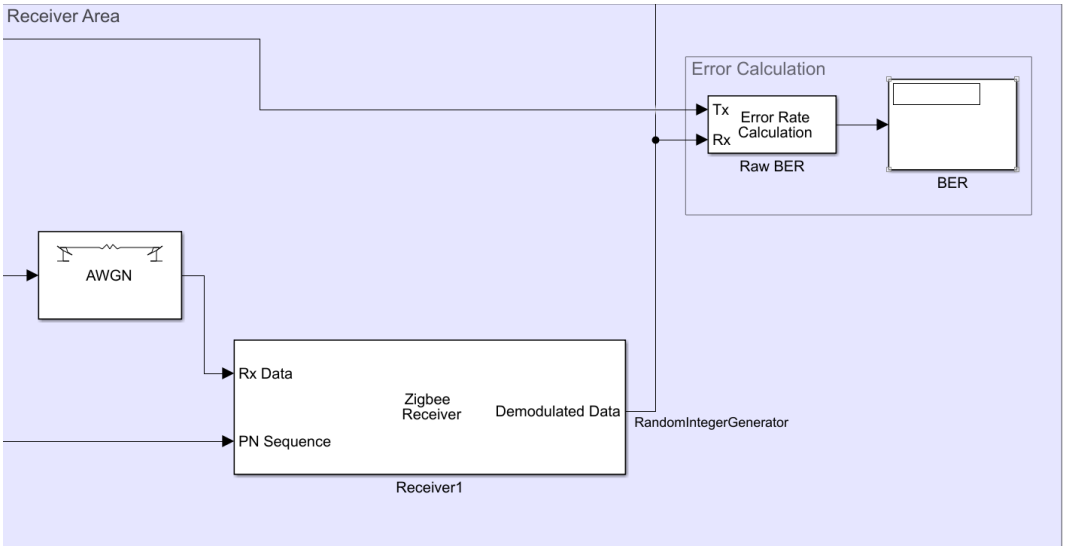

Figure 5.14: Receiver Area

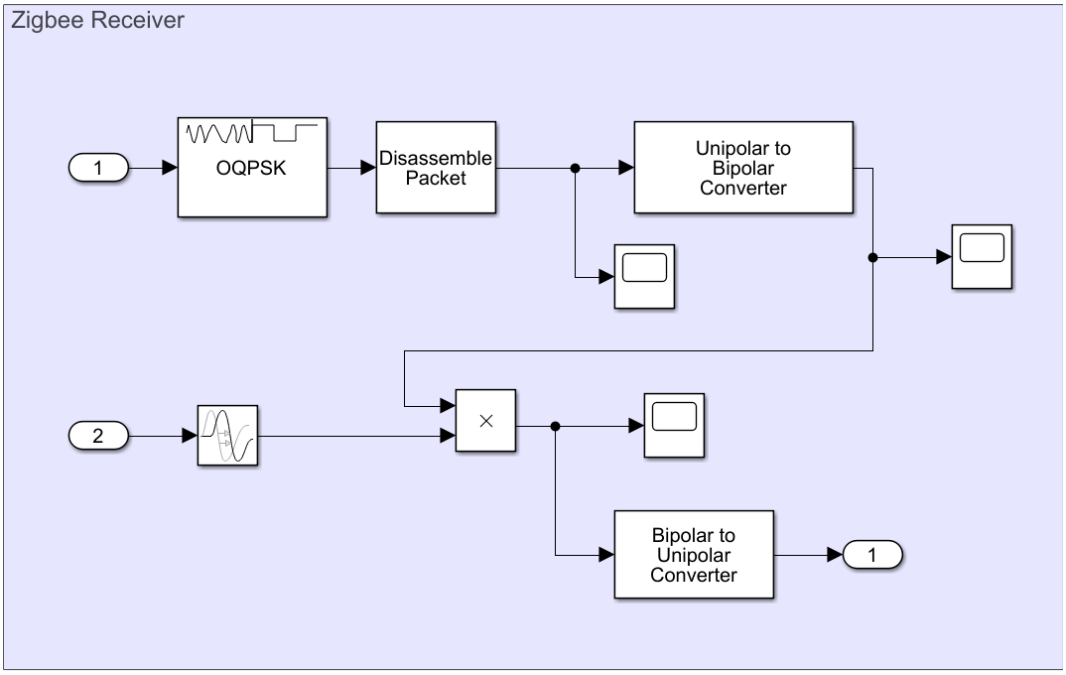

Figure 5.15: Zigbee Receiver Subsystem

5.5 Output Area

As can be seen in the figure 5.16, the Output Area consists of a Shift Register Function, Screen Function to decode the taken alarm level data and displays for the room number, level and alarm and Change Detector to clear display before the new room data is gathered.

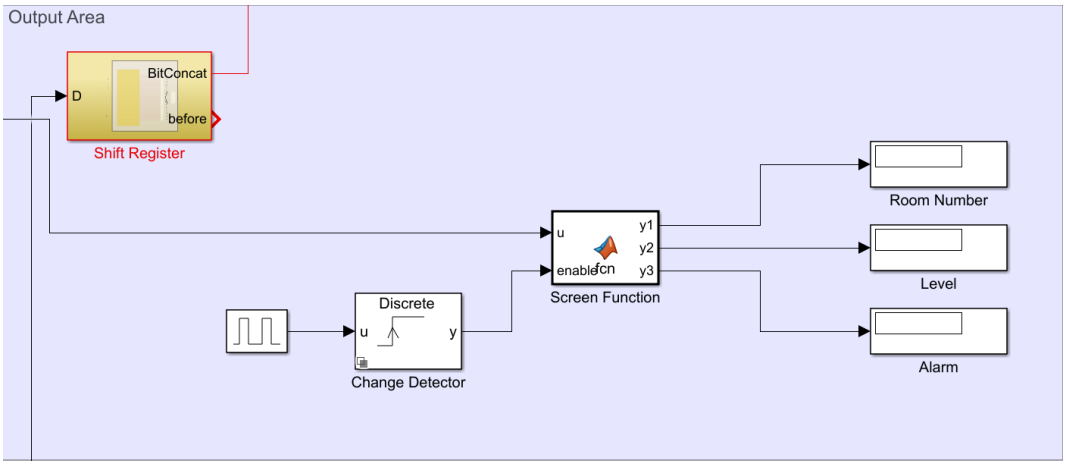

Figure 5.16: Output Area

5.5.1 Shift Register

As mentioned in the Receiver, the output of the Receiver is 1 bit, thus is not the real encrypted number as before. Shift registers could be used to overcome this problem. A shift register is by using flip flops with connected end to end where they all share one clock, shifts the taken input to the next flip flop thus could add the taken inputs end to end. Simulink has a shift register block which had failed for the model thus a hand made shift register is created to overcome this problem. The subsystem of the shift register is given in figure 5.17. 128 D Flip Flops are connected to add the received data together as can be seen in figure 5.18. Q output of each flip flop is then given as an input to the 16 8 to 1-Mux Blocks which are also connected to one 16 to 1-Mux block in simulink, which concatenates the taken inputs to produce one single number. After the number is taken a small function to reshape the output of mux is used to create a 1x128 vector output which is the same output the DecimalToBitConverter function gave after taking the Cipher Output Data. The Output of this ShiftRegister is then sent to the AES Algorithm Area to be Deciphered. The timings of the clocks are designed carefully so that no new data is produced from the rooms before the transfer and concatenation of the data is finished. In the figures 5.17 and 5.18, the subsystem of the shift register block can be seen.

5.5.2 Screen Function

The output of the shift register, that is a binary number, is then transferred to the BitToDecimalConverter function to enlarge it into the decimal matrix that is taken from the cipher block originally. The output matrix of the function is then connected to the input of the decryption block to decipher the taken message and transferred back to the Output Area as an input to Screen Function where the created message in patient room is decoded by using bitand function. The evaluation of the function goes as to achieve the room number, that is the 3 least signicant bits, the received input is used in bitand with decimal 7, which in binary equals to 00000111 where the output would only give the least significant 3 bits of the input. For Level decoding it is done with 56 (00111000) and for Alarm, 192 (11000000) in decimal is used.

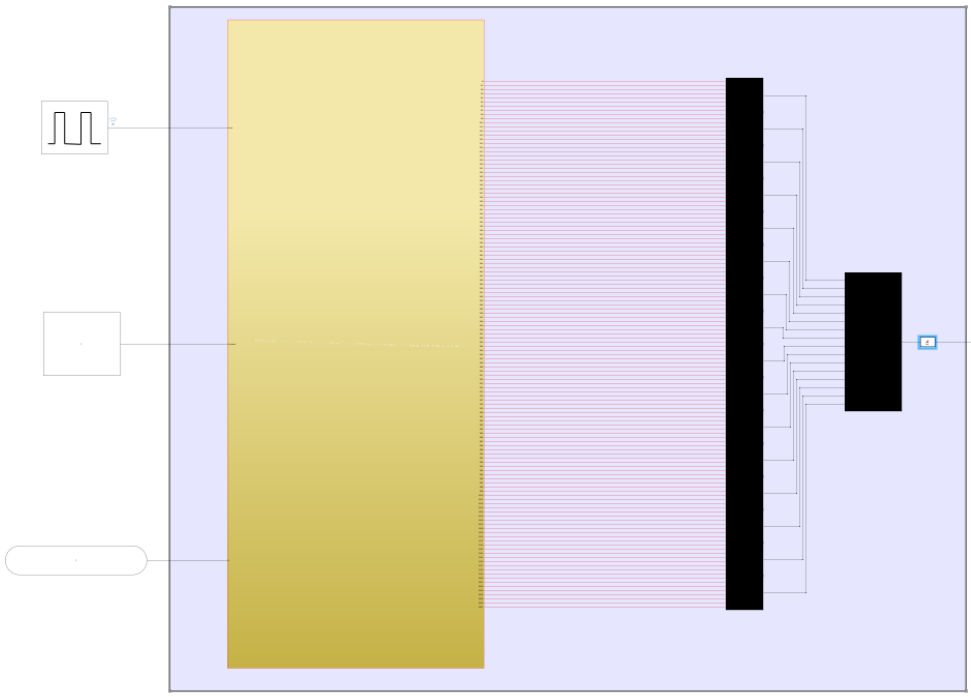

Figure 5.17: Shift Register Block

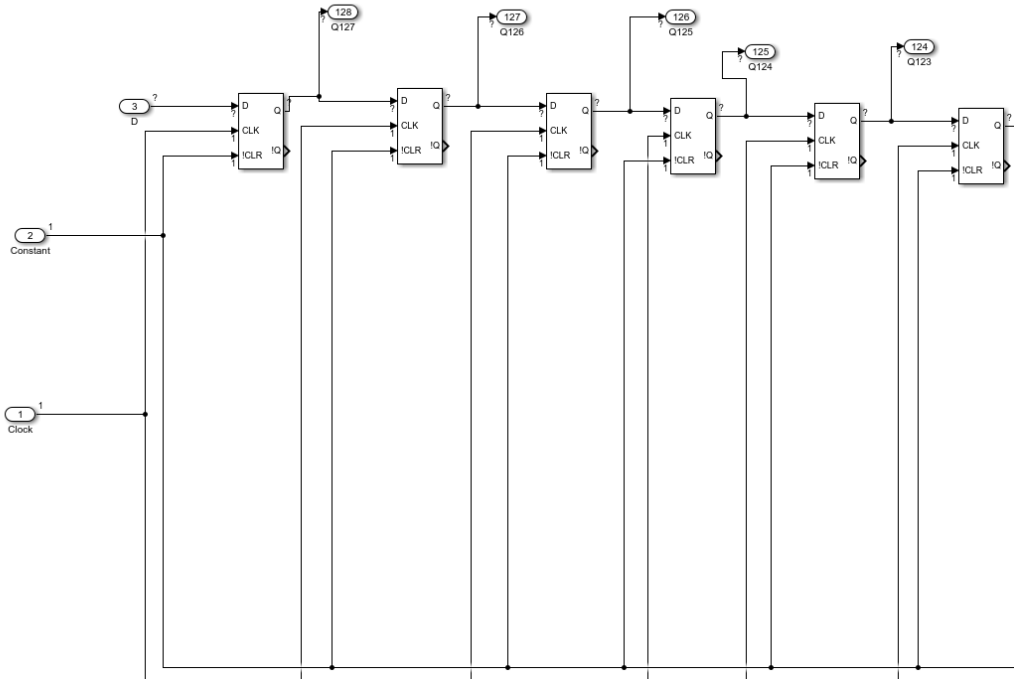

Figure 5.18: Concatenation of D Flip-Flops

```
function [y1,y2,y3]= fcn(u,enable)

room = uint8(7);
level = uint8(56);
alarm = uint8(192);

if (enable == 1)

     y1 = bitand(u,room) + 1;
     y2 = (bitsrl(bitand(u,level),3));
     y3 = (bitsrl(bitand(u,alarm),6));

else

     y1 = uint8(0);
     y2 = uint8(0);
     y3 = uint8(0);

end
end
```

Figure 5.19: Screen Function

6. HARDWARE IMPLEMENTATION

For the purpose of implementation of the model created in simulink on hardware, Vivado Design Suite program is utilized. The steps for Vivado implementation is given in the following sections.

6.1 Simplified Model

Due to the original model's operation time and many other errors faced during the code generations, the model is simplified to a model where only the input and output is given to the system, which is shown below as doing the ciphering as AES should do. Another output EncryptedText is added to show the model works properly. During the code generation, EncryptedText output is removed to prevent any complexity in code.

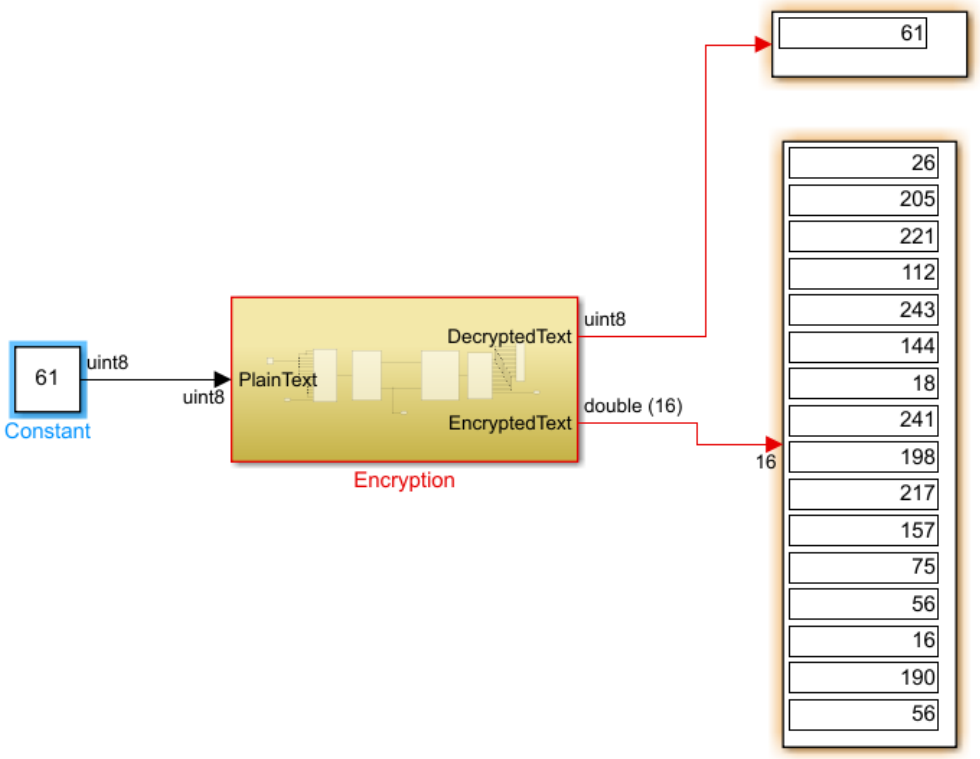

Figure 6.1: Simplified Model Simulation on Simulink

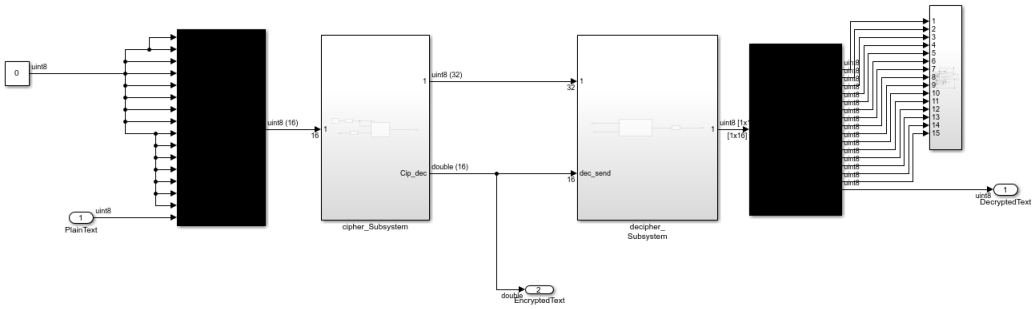

Figure 6.2: Model on Simulink

After the model was proven working in Simulink environment, the model was ready for code generation for use in Vitis platform which is detailed in the further sections.

6.2 Designing of Microblaze

As explained in Vivado Design Suite section[reff] microblaze is a soft core processor to simulate various types of FPGAs to achieve real-life results of the design. With the use of Vitis the Microblaze can be programmed to give simulation results as according to the written C/C++ code. Due to this the Microblaze is designed in Vivado environment before proceeding with code generation of the model via Simulink Embedded Coder.

To create Microblaze, Vivado Design Suite is used. As can be seen in the figure 6.3 below Vivado offers various types of projects. Among those projects RTL Project type is selected.

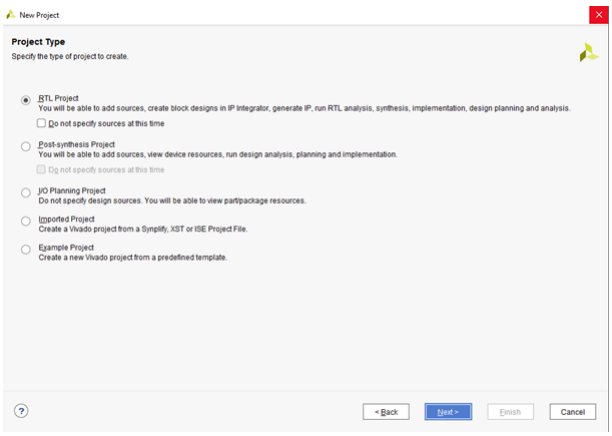

Figure 6.3: Project Type Screen

The project is created with no source is added to prevent any errors it might cause. Sources are going to be added after the Microblaze is created.

For the part selection, Artix-7 AC701 Evaluation Platform Board [23] is selected for the simulation, since the selected board was used for other projects thus it is known to have enough pins for the given inputs and outputs, as can be seen in the figure 6.4.

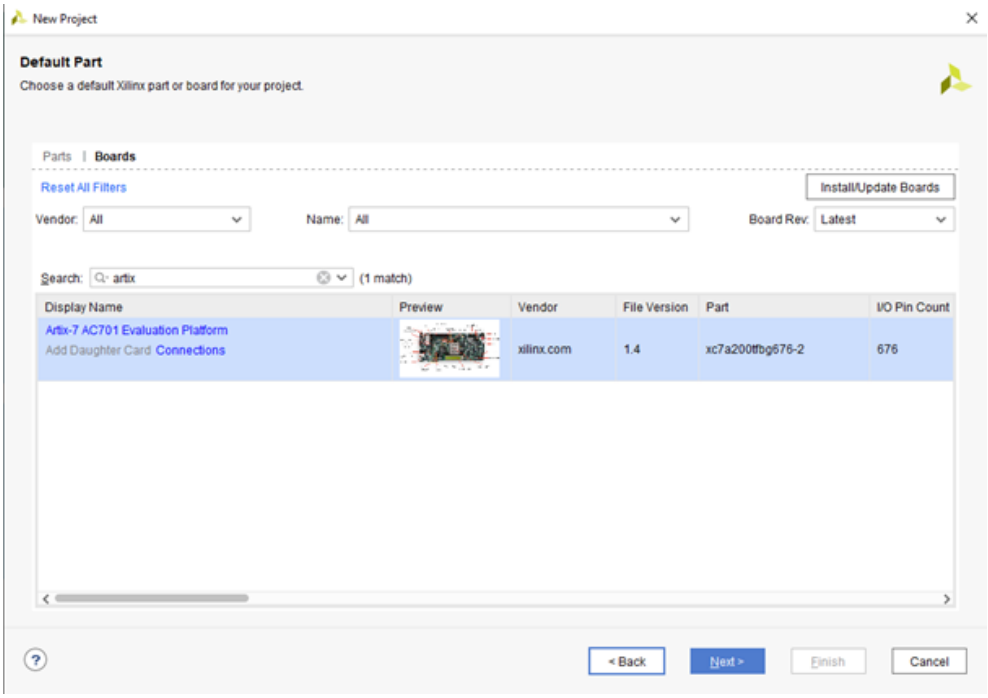

Figure 6.4: Board Type Screen

For the addition of Microblaze a block design is needed to be created first. To do this, simply Create Block Design option is selected under the IP INTEGRATOR section of Vivado. After the creation of Block Design Microblaze is added from Add Ip section on the Diagram. The added Microblaze in the Block Design Diagram can be seen in figure 6.5.

Before proceeding further, the Microblaze was needed to be customized. For this purpose Customize Block option of Microblaze is selected to Re-Customize IP. Microblaze debug module is disabled from General Settings to prevent errors it might cause. Which can be seen in the figure 6.6.

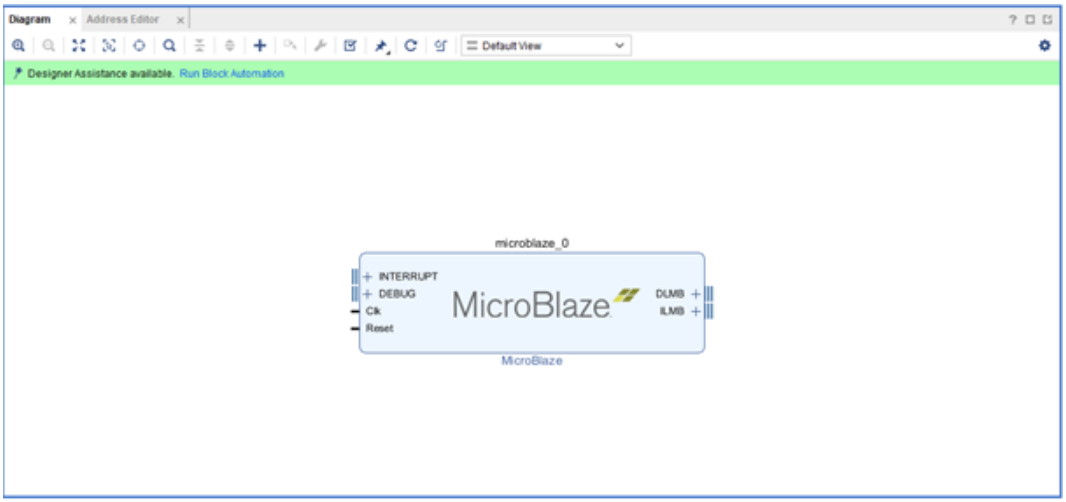

Figure 6.5: Microblaze

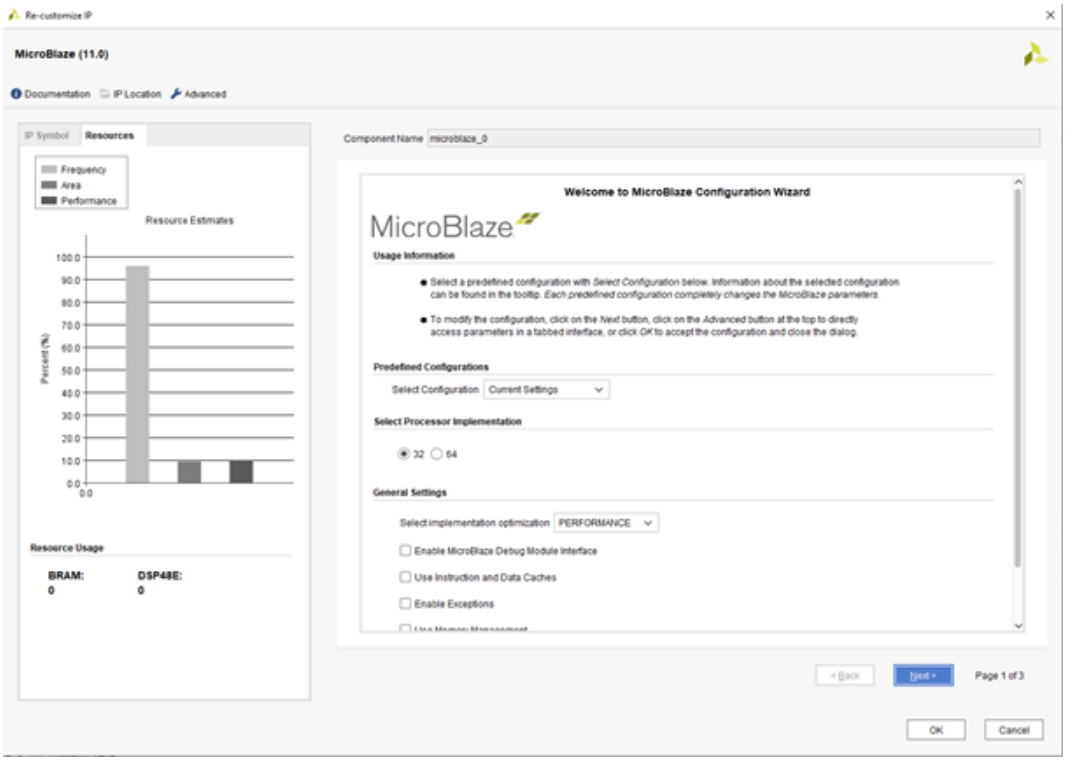

Figure 6.6: Microblaze Configuration Screen

After disabling Debug Module from Microblaze, Run Block Automation feature of Vivado is used to put a Microblaze system together. The Run Block Automation requires some configuration preferances. The Local Memory is set to 128 KB instead of the default 8KB to prevent any memory errors. Also Debug Module is disabled again. In the figure 6.7 the preferances made can be seen.

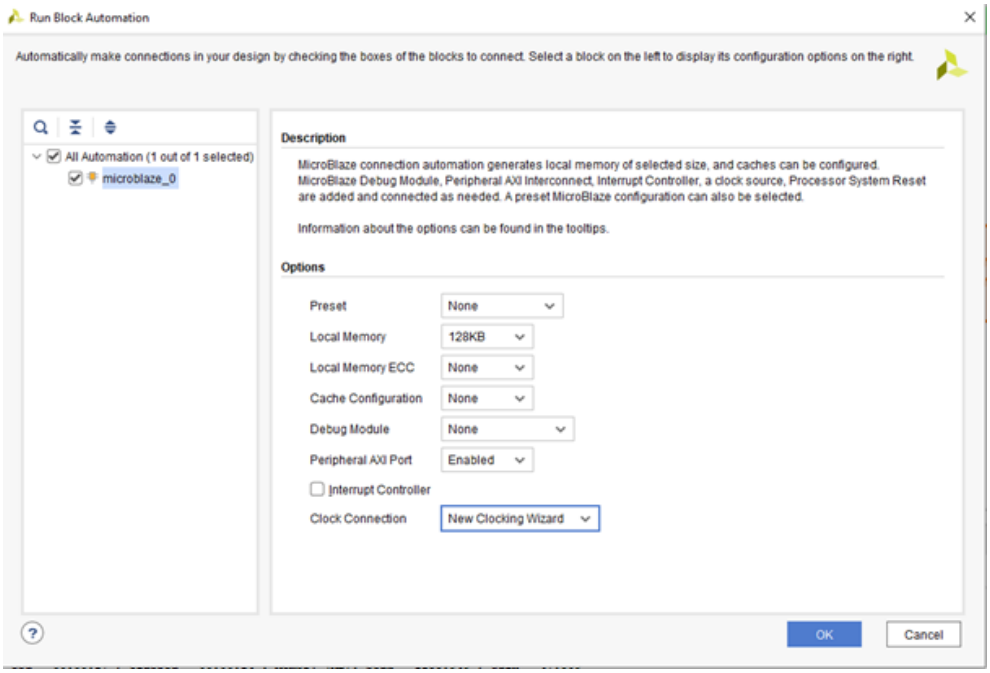

Figure 6.7: Run Block Automation Screen

Block Automation automatically adds the needed blocks for Microblaze system which are Microblaze Local Memory, Processor System Reset and Clocking Wizard. In the Clocking wizard section the clock is changed into single ended since only one clock is going to be used for the simulation. The schematic after the Block Automation can be seen in figure 6.8.

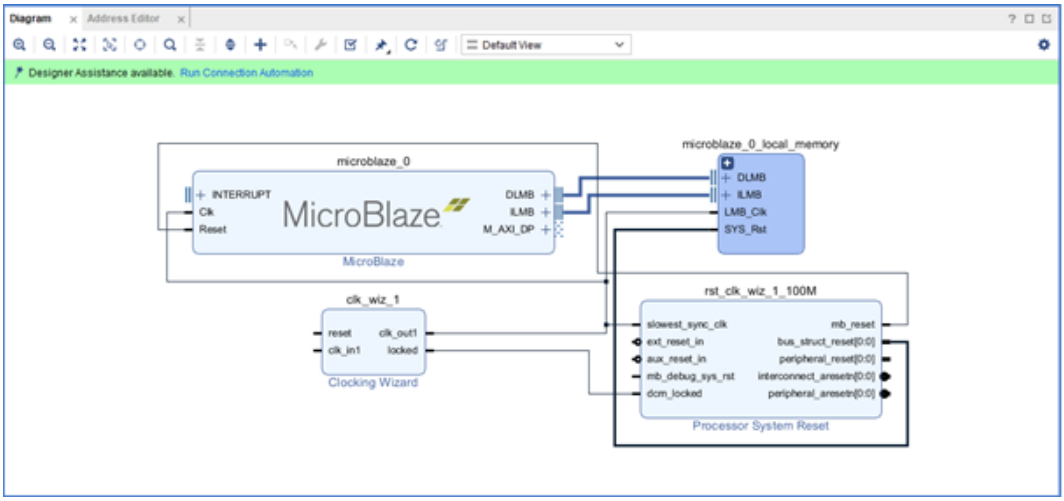

Figure 6.8: Schematic After Block Automation

After the changes done in the Clocking Wizard, AXI UART is added to the design to be able to read the signals that are processed in the system to be able to see the output of the design. After running Connection Automation with all the blocks in the diagram Vivado creates all the connections for inputs and outputs for the model. Later from Address Editor section Axi_Uartlite_0 is found to increase the range of UART to 128K to make sure UART is able to process the model signals. The created circuit schematic is given in figure 6.9.

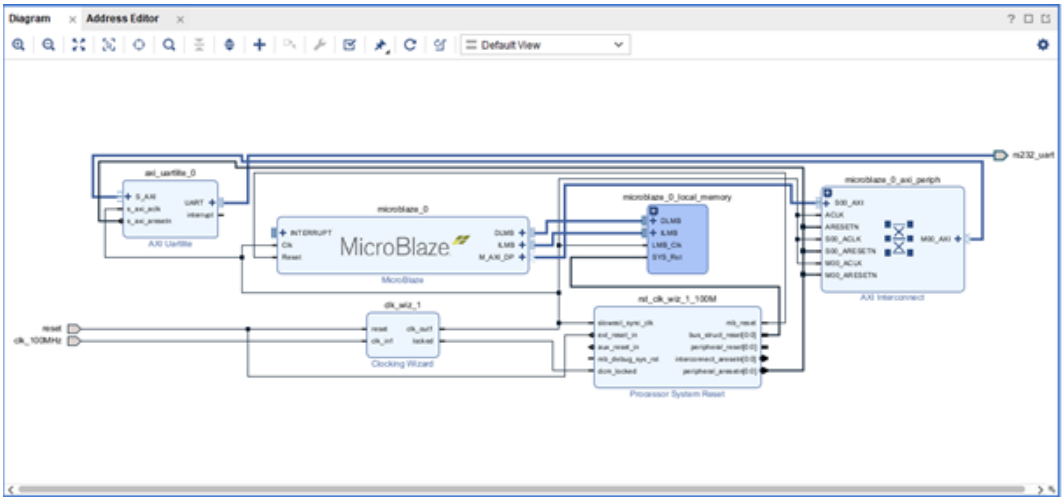

Figure 6.9: Final Microblaze Schematic

Since the block designs are unable to be synthesized, HDL wrapper of the block design is created from the design sources. Then the TestBench for the simulation part is added from Sources section as Simulation Source to the project and the model is synthesized.

After the synthesis ends for use in vivado the hardware is exported as an .xsa file to integrate the hardware with the software in Vitis environment. For the export options, Fixed Platform and Pre-Synthesis is selected.

6.3 Vitis Program

After the hardware exporting of Vivado is completed, Vitis is launched from Vivado from the tools tab of Vivado. A workspace for the Vitis is created or selected. An important part of Vitis is while creating a directory, there should be no spaces or any letter out of English alphabet since Vitis cannot process when those are included in directory.

In the Vitis software firstly an application project is needed to be created. The platform of the project is selected as the hardware that is exported from the Vivado design as can be seen figure 6.10.

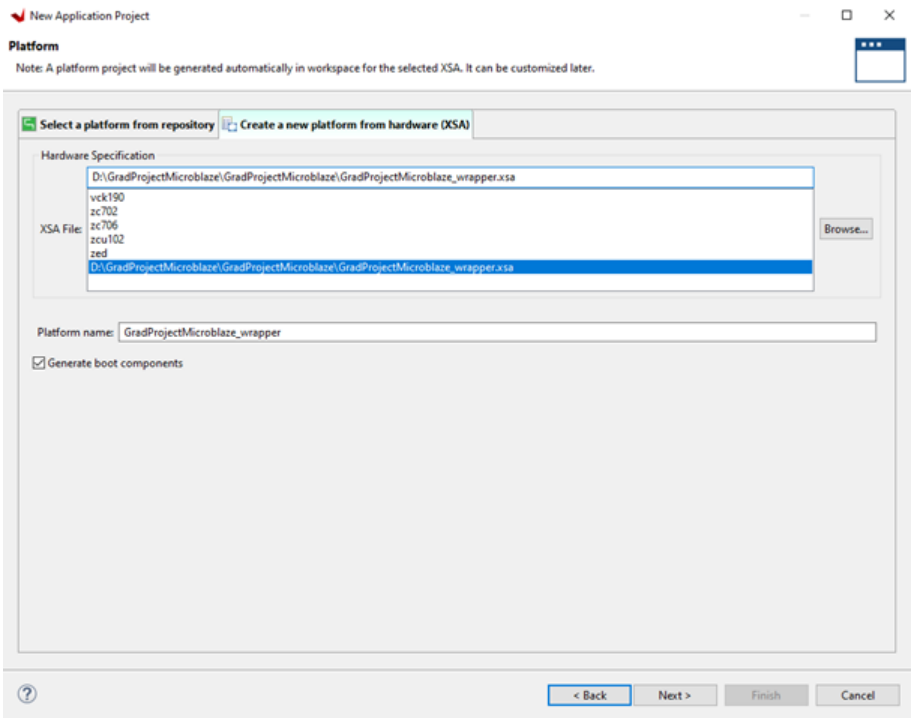

Figure 6.10: Platform Selection

Later an application name is given to the project and proceeded. The Domain section does not require configuration since hardware platform is given. For the Templates section Empty Application is selected. By finishing the configurations our Vitis project is created from the exported hardware.

On the Explorer section, a new file is created under the src file to write the configuration functions for the hardware. Also the generated codes from the Simulink are included in the project under the domain, which is seen selected and named in the figure above as “GradProjectVitis [domain_microblaze_0]”, by right clicking and opening its properties. From there C/C++ General section is extended to reach Paths and Symbols. In the Includes tab of Paths and Symbols the addition of the code file is made as a file system. The figure 6.11 below shows where the code file is to be added explicitly.

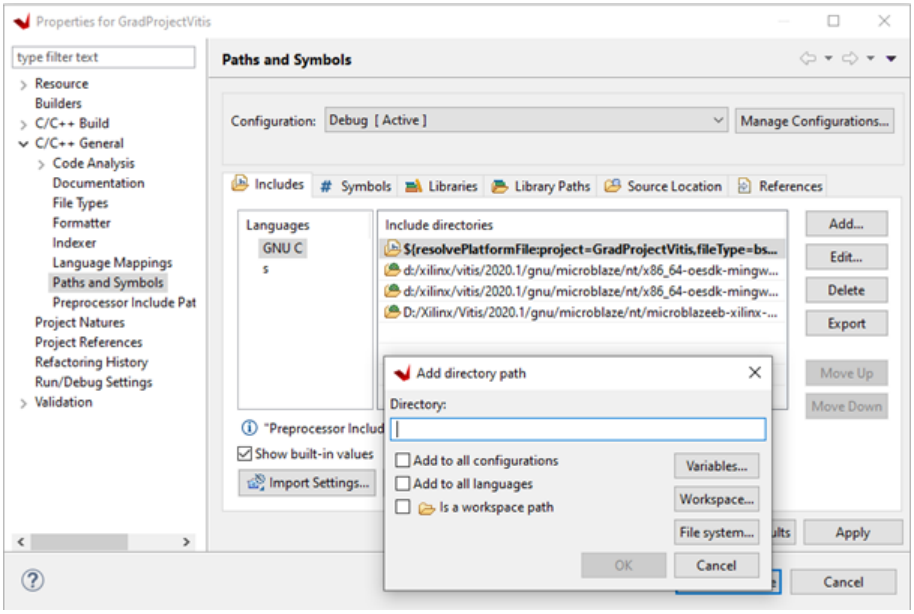

Figure 6.11: Directory for Code Folder Addition

The produced code, that can be seen in listing A.1, of the simplified model is added to the Vitis directory path to be able to utilize the code for hardware implementation. A simple code is written for the instructions and the produced functions from Simulink are used to check the validity of the code produced by the Embedded Coder. For the verification, UART module's signals are read as can be seen from the given code. The written program can be seen in figure 6.12.

```
#include <stddef.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "rtwtypes.h"
#include <stdlib.h>
#include <xparameters.h>
#include <xuartlite_l.h>
#include "Encryption.c"
#include "Encryption.h"

ExtU rtU;

ExtY rtY;

int main(){

    rtU.PlainText=5;

    XUartLite_WriteReg(XPAR_AXI_UARTLITE_0_BASEADDR, 4, rtU.PlainText);
    XUartLite_WriteReg(XPAR_AXI_UARTLITE_0_BASEADDR, 4, 6);

    Encryption_initialize();
    Encryption_step();

    XUartLite_WriteReg(XPAR_AXI_UARTLITE_0_BASEADDR, 4, rtY.DecryptedText);

    return rtY.DecryptedText;
}
```

Figure 6.12: Instruction Code

After building the project, an elf file is produced by Vitis to be used in Vivado. The produced elf file is then added to the sources of the Vivado project for both simulation and design sources. After the addition is completed, from the tools tab Associate Elf Files tab is selected to associate the elf file with the design. Afterwards the simulation is ran. Lastly, to see the UART signals, the data_to_transfer[0:7] is selected under the Scope tab and added to the waveform.

The Vivado simulation gave Undefined output as can be seen in figure 6.13. Due to this, a verification of a simple code was needed to understand the problem.

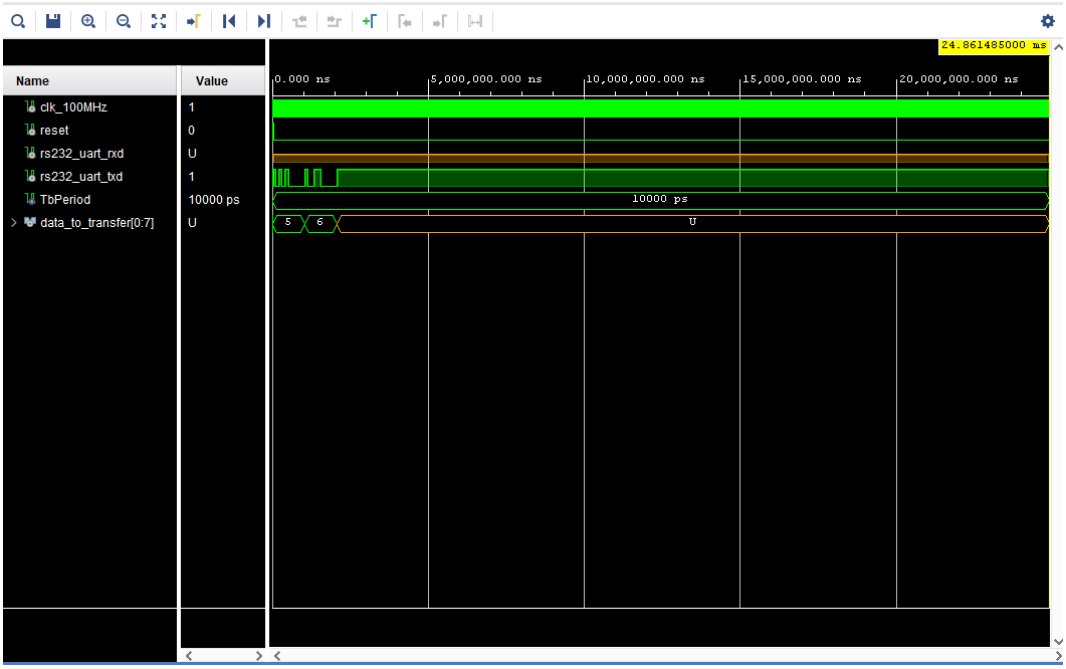

Figure 6.13: Simulation Results for the Model

6.4 Summation Model

For verification, a simple simulink code is made to be tested in simulink environment. A basic summation operation is made in the Simulink environment. A Matlab Function is made that takes 2 inputs and gives their summation as the output as can be seen in figure 6.14.

After following the same steps described in generation of code for the model the code is generated which is much shorter than the one that is generated for the model since it is a very simple design.

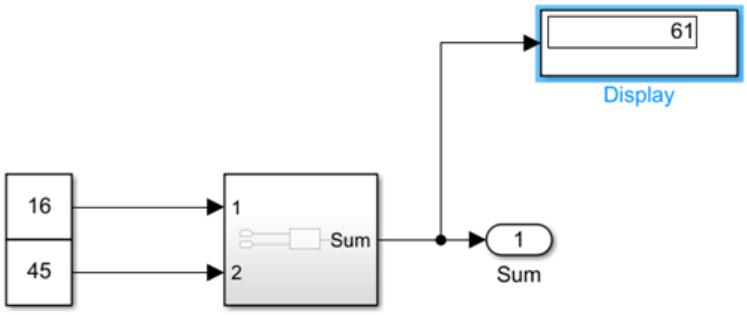

Figure 6.14: Simple Summation Model in Simulink

After adding the Summation model's code into Vitis the instruction code is written for the hardware. As used before, UART is benefitted to read the signals. The written code is seen in figure 6.15.

```
#include <stddef.h>
#include <stdio.h>
#include "rtwtypes.h"
#include <stdlib.h>
#include <xparameters.h>
#include <xuartlite_l.h>
#include "Summ0.c"
#include "Summ0.h"

ExtU rtU;

ExtY rtY;

int main(){

    rtU.Input=16;
    rtU.Input1=45;

    XUartLite_WriteReg(XPAR_AXI_UARTLITE_0_BASEADDR, 4, rtU.Input);
    XUartLite_WriteReg(XPAR_AXI_UARTLITE_0_BASEADDR, 4, 6);

    Summ0_step();

    XUartLite_WriteReg(XPAR_AXI_UARTLITE_0_BASEADDR, 4, rtY.Sum);

    return rtY.Sum;
}
```

Figure 6.15: Instruction Code for Sum

After following the same steps as before the elf file is added to Vivado system and the created hardware is simulated. The simulation result can be seen in figure 6.16.

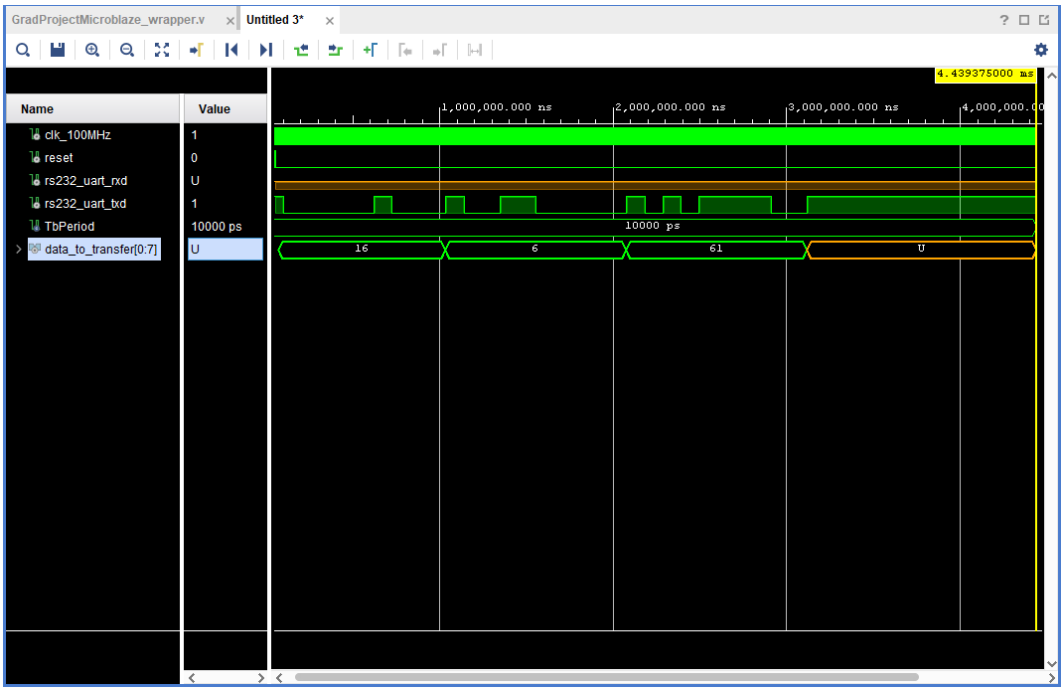

Figure 6.16: Simple Summation Model Simulation Results

As can be seen from figure 6.16, the output of the hardware system is the same as the model thus the code generation for a simple model in Simulink works properly.

7. CONSTRAINTS, FUTURE WORK, CONCLUSIONS AND RECOMMENDATIONS

7.1 Possible Applications of the Project

The project is planned to lean on creating healthcare applications with improved security by IoT systems. The system is planned to track the vitals of patients and transfer them to a command center where the data is evaluated. The evaluation of data is aimed to allow the medical departments to track the patients and during emergencies get an alarm to reduce the response time which could potentially save many lives. On the other hand, the creation of such a project would open the window for such applications to be produced on the other sectors in the market.

7.2 Realistic Constraints

7.2.1 Code Generation

Aside from all the positive sides of the project, the problems may arise during the implementation of complex systems due to the code generation used. The more complex systems would be generated with more complex and long codes. Therefore the operation time would increase with the possibility to debug the code would become harder. Due to all these, the implementation of complex systems could lead to unsuccessful design attemps.

7.2.2 Social, Environmental and Economic Impact

As explained in the first section, a secure IoT system could ease the daily life as it increases the profits of the companies. While the low cost and low power applications of IoT could create a rise in the economy, it could also help to create a better future of the nature by increasing the efficiency of the devices. The higher efficiency would mean less waste. Therefore the use of IoT could potentially slow down or even create

a regression for the one of the biggest natural disasters of the 21st century, the Global Warming. [24]

7.2.3 Cost Analysis

All the programs used in the project, such as Xilinx Vivado Design Environment, have free versions for academic purposes with the licensing done the faculty. The need for an FPGA Evaluation Board was compensated by the faculty as well. The computers used during the completion of the projects were all personal computers of the project members. Thefore the cost factor of the project is, other than the school's license cost, none.

7.2.4 Standards

Many types of standards should be applicable for this type of project. The selected standards cover IEEE and NIST standards.

7.2.5 Health and Safety Concerns

The project focuses on a project level of design therefore there is no real health or safety concerns. Yet if the project is taken to a further level there could be some possible hazards if project is not done properly. As said above, one of the biggest safety concerns of an IoT design is cyberattacks. Therefore the safety of the design is taken to the highest priority, to minimize the safety concerns. The designed system, if has execution errors due to engineering mistakes, could cost a person's life therefore to prevent this every possible option is designed in the system to prevent any casualty.

7.3 Conclusion, Future Works and Recommendations

As can be seen from figure 6.16 the simple code produced from Simulink was able to run on Vivado thus the method was proven correct. Yet, it cannot be decided whether the produced code is broken or the system the model is simulated on is not powerful enough. As said before, AES due to being a very complex program which can be seen from the produced code from Simulink that has over 1700 lines of code only, requires multiple functions to be repeated multiple times thus requires a powerful system to process on for this application thus the system might be weak to handle this

program. Then again, the program gets stuck at the UNDEFINED result eventhough the program was ran for 10 hours without interruption. The cause of error cannot be decided from these results thus it stays inconclusive.

The AES Algorithm, eventhough is a highly secure algorithm against cyberattacks, has a very complex structure. Due to this the generated code via Simulink is unapplicable, either due to the errors of the long and complex code, which can be seen in the A.1, or the powerful system requirement of the code. Therefore, since the aim of the design is creating a rapidly produced secure IoT systems, in the future works simpler security algorithms could be tried to prevent the problems of AES algorithm.

REFERENCES

[1] B. Selic, "Model-driven development: Its essence and opportunities," 01 2006, pp. 313–319.

[2] I. Petrescu, B. Pavaloiu, and D. George, "Digital logic introduction using fpgas," *Procedia - Social and Behavioral Sciences*, vol. 180, pp. 1507–1513, 05 2015.

[3] L. S. Vailshery, "Global iot and non-iot connections 2010-2025," Mar 2021. [Online]. Available: https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/

[4] "5 layer architecture of internet of things," Sep 2020. [Online]. Available: https://www.geeksforgeeks.org/5-layer-architecture-of-internet-of-things/

[5] H. Mohajan, "The first industrial revolution: Creation of a new global human era," vol. 5, pp. 377–387, 10 2019.

[6] "How many iot devices are there in 2021? more than ever!" Mar 2021. [Online]. Available: https://techjury.net/blog/how-many-iot-devices-are-there/#gref

[7] "Iot fleet management market size worth usd 16.86 billion by 2025." [Online]. Available: https://www.grandviewresearch.com/press-release/global-internet-of-things-iot-fleet-management-market

[8] L. Pascu, "The iot threat landscape and top smart home vulnerabilities in 2018," 2018.

[9] "Mathworks 2020 company factsheet." [Online]. Available: https://uk.mathworks.com/content/dam/mathworks/handout/2020-company-factsheet-8-5x11-8282v20.pdf

[10] C. Moler, "A brief history of matlab." [Online]. Available: https://www.mathworks.com/company/newsletters/articles/a-brief-history-of-matlab.html

[11] "Embedded coder." [Online]. Available: https://www.mathworks.com/products/embedded-coder.html

[12] "Ise simulator (isim)." [Online]. Available: https://www.xilinx.com/products/design-tools/isim.html

[13] "An introduction to high-level synthesis." [Online]. Available: https://ieeexplore.ieee.org/document/5209958

[14] Edn, "The vivado design suite accelerates programmable systems integration and implementation by up to 4x," Jun 2012. [Online]. Available: https://www.edn.com/the-vivado-design-suite-accelerates-programmable-\systems-integration-and-implementation-by\-up-to-4x/

[15] “Microblaze.” [Online]. Available: https://www.xilinx.com/products/intellectual-property/microblazecore.html

[16] “What is zigbee wireless mesh networking?” [Online]. Available: https://www.digi.com/solutions/by-technology/zigbee-wireless-standard

[17] J. Daemen and V. Rijmen, *The Design of Rijndael The Advanced Encryption Standard (AES)*. Springer-Verlag Berlin Heidelberg, 2002.

[18] “Temperature sensor.” [Online]. Available: https://www.mathworks.com/help/physmod/simscape/ref/temperaturesensor.html

[19] J. Hart, “Normal resting pulse rate ranges,” *Journal of Nursing Education and Practice*, vol. 5, 05 2015.

[20] “Axi uart lite.” [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi_uartlite.html

[21] D. Hill, “Advanced encryption standard (aes)-128,192, 256.” [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/73412-advanced-encryption-standard-aes-128-192-256

[22] “Modulation demodulation in zigbee.” [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/36258-modulation-demodulation-in-zigbee?s_tid=srchtitle

[23] “Xilinx artix-7 fpga ac701 evaluation kit.” [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-a7-ac701-g.html

[24] U. Shahzad, “Global warming: Causes, effects and solutions,” 08 2015.

APPENDIX A.1

```
1
2  /*
3   * Academic License - for use in teaching, academic research, and meeting
4   * course requirements at degree granting institutions only.  Not for
5   * government, commercial, or other organizational use.
6   *
7   * File: Encryption.c
8   *
9   * Code generated for Simulink model 'Encryption'.
10  *
11  * Model version                  : 4.15
12  * Simulink Coder version         : 9.4 (R2020b) 29-Jul-2020
13  * C/C++ source code generated on : Wed Jun 16 17:39:04 2021
14  *
15  * Target selection: ert.tlc
16  * Embedded hardware selection: Custom Processor->Custom Processor
17  * Emulation hardware selection:
18  *    Differs from embedded hardware (Custom Processor->MATLAB Host Computer)
19  * Code generation objectives:
20  *    1. Execution efficiency
21  *    2. RAM efficiency
22  * Validation result: Not run
23  */
24
25 #include "Encryption.h"
26 #define NumBitsPerChar                 8U
27
28 /* External inputs (root inport signals with default storage) */
29 ExtU rtU;
30
31 /* External outputs (root outports fed by signals with default storage) */
32 ExtY rtY;
33
34 /* Real-time model */
35 static RT_MODEL rtM_;
36 RT_MODEL *const rtM = &rtM_;
37
38 /* Forward declaration for local functions */
39 static real_T mod(real_T x);
40 static void circshift(real_T a[4]);
41 static real_T function_handle_parenReference(real_T varargin_1, real_T
42   varargin_2);
43 static void KeyExpansion(const char_T key[32], real_T w[240]);
44 static void circshift_h(real_T a[4]);
45 static void circshift_h5(real_T a[4]);
46 static void circshift_h5f(real_T a[4]);
47 static real_T xtime(real_T x, real_T c);
48 static void MixColumns(const real_T state[16], real_T State[16]);
49 static void circshift_m1s(real_T a[4]);
50 static void ara_func(real_T state[16]);
51 static real_T rtGetNaN(void);
52 static real32_T rtGetNaNF(void);
```

```
53 extern real_T rtInf;
54 extern real_T rtMinusInf;
55 extern real_T rtNaN;
56 extern real32_T rtInfF;
57 extern real32_T rtMinusInfF;
58 extern real32_T rtNaNF;
59 static void rt_InitInfAndNaN(size_t realSize);
60 static boolean_T rtIsInf(real_T value);
61 static boolean_T rtIsInfF(real32_T value);
62 static boolean_T rtIsNaN(real_T value);
63 static boolean_T rtIsNaNF(real32_T value);
64 typedef struct {
65   struct {
66     uint32_T wordH;
67     uint32_T wordL;
68   } words;
69 } BigEndianIEEEDouble;
70
71 typedef struct {
72   struct {
73     uint32_T wordL;
74     uint32_T wordH;
75   } words;
76 } LittleEndianIEEEDouble;
77
78 typedef struct {
79   union {
80     real32_T wordLreal;
81     uint32_T wordLuint;
82   } wordL;
83 } IEEESingle;
84
85 real_T rtInf;
86 real_T rtMinusInf;
87 real_T rtNaN;
88 real32_T rtInfF;
89 real32_T rtMinusInfF;
90 real32_T rtNaNF;
91 static real_T rtGetInf(void);
92 static real32_T rtGetInfF(void);
93 static real_T rtGetMinusInf(void);
94 static real32_T rtGetMinusInfF(void);
95
96 /*
97  * Initialize rtNaN needed by the generated code.
98  * NaN is initialized as non-signaling. Assumes IEEE.
99  */
100 static real_T rtGetNaN(void)
101 {
102   size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
103   real_T nan = 0.0;
104   if (bitsPerReal == 32U) {
105     nan = rtGetNaNF();
106   } else {
107     union {
108       LittleEndianIEEEDouble bitVal;
109       real_T fltVal;
110     } tmpVal;
111
```

```
    tmpVal.bitVal.words.wordH = 0xFFF80000U;
    tmpVal.bitVal.words.wordL = 0x00000000U;
    nan = tmpVal.fltVal;
  }

  return nan;
}

/*
 * Initialize rtNaNF needed by the generated code.
 * NaN is initialized as non-signaling. Assumes IEEE.
 */
static real32_T rtGetNaNF(void)
{
  IEEESingle nanF = { { 0 } };

  nanF.wordL.wordLuint = 0xFFC00000U;
  return nanF.wordL.wordLreal;
}

/*
 * Initialize the rtInf, rtMinusInf, and rtNaN needed by the
 * generated code. NaN is initialized as non-signaling. Assumes IEEE.
 */
static void rt_InitInfAndNaN(size_t realSize)
{
  (void) (realSize);
  rtNaN = rtGetNaN();
  rtNaNF = rtGetNaNF();
  rtInf = rtGetInf();
  rtInfF = rtGetInfF();
  rtMinusInf = rtGetMinusInf();
  rtMinusInfF = rtGetMinusInfF();
}

/* Test if value is infinite */
static boolean_T rtIsInf(real_T value)
{
  return (boolean_T)((value==rtInf || value==rtMinusInf) ? 1U : 0U);
}

/* Test if single-precision value is infinite */
static boolean_T rtIsInfF(real32_T value)
{
  return (boolean_T)(((value)==rtInfF || (value)==rtMinusInfF) ? 1U : 0U);
}

/* Test if value is not a number */
static boolean_T rtIsNaN(real_T value)
{
  boolean_T result = (boolean_T) 0;
  size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
  if (bitsPerReal == 32U) {
    result = rtIsNaNF((real32_T)value);
  } else {
    union {
      LittleEndianIEEEDouble bitVal;
      real_T fltVal;
    } tmpVal;
```

```
171
172     tmpVal.fltVal = value;
173     result = (boolean_T)((tmpVal.bitVal.words.wordH & 0x7FF00000) == 0x7FF00000 &&
174                          ( (tmpVal.bitVal.words.wordH & 0x000FFFFF) != 0 ||
175                            (tmpVal.bitVal.words.wordL != 0) ));
176   }
177
178   return result;
179 }
180
181 /* Test if single-precision value is not a number */
182 static boolean_T rtIsNaNF(real32_T value)
183 {
184   IEEESingle tmp;
185   tmp.wordL.wordLreal = value;
186   return (boolean_T)( (tmp.wordL.wordLuint & 0x7F800000) == 0x7F800000 &&
187                       (tmp.wordL.wordLuint & 0x007FFFFF) != 0 );
188 }
189
190 /*
191  * Initialize rtInf needed by the generated code.
192  * Inf is initialized as non-signaling. Assumes IEEE.
193  */
194 static real_T rtGetInf(void)
195 {
196   size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
197   real_T inf = 0.0;
198   if (bitsPerReal == 32U) {
199     inf = rtGetInfF();
200   } else {
201     union {
202       LittleEndianIEEEDouble bitVal;
203       real_T fltVal;
204     } tmpVal;
205
206     tmpVal.bitVal.words.wordH = 0x7FF00000U;
207     tmpVal.bitVal.words.wordL = 0x00000000U;
208     inf = tmpVal.fltVal;
209   }
210
211   return inf;
212 }
213
214 /*
215  * Initialize rtInfF needed by the generated code.
216  * Inf is initialized as non-signaling. Assumes IEEE.
217  */
218 static real32_T rtGetInfF(void)
219 {
220   IEEESingle infF;
221   infF.wordL.wordLuint = 0x7F800000U;
222   return infF.wordL.wordLreal;
223 }
224
225 /*
226  * Initialize rtMinusInf needed by the generated code.
227  * Inf is initialized as non-signaling. Assumes IEEE.
228  */
229 static real_T rtGetMinusInf(void)
```

```
{
  size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
  real_T minf = 0.0;
  if (bitsPerReal == 32U) {
    minf = rtGetMinusInfF();
  } else {
    union {
      LittleEndianIEEEDouble bitVal;
      real_T fltVal;
    } tmpVal;

    tmpVal.bitVal.words.wordH = 0xFFF00000U;
    tmpVal.bitVal.words.wordL = 0x00000000U;
    minf = tmpVal.fltVal;
  }

  return minf;
}

/*
 * Initialize rtMinusInfF needed by the generated code.
 * Inf is initialized as non-signaling. Assumes IEEE.
 */
static real32_T rtGetMinusInfF(void)
{
  IEEESingle minfF;
  minfF.wordL.wordLuint = 0xFF800000U;
  return minfF.wordL.wordLreal;
}

/* Function for MATLAB Function: '<S3>/MATLAB Function' */
static real_T mod(real_T x)
{
  real_T r;
  if (rtIsNaN(x)) {
    r = (rtNaN);
  } else if (rtIsInf(x)) {
    r = (rtNaN);
  } else if (x == 0.0) {
    r = 0.0;
  } else {
    r = fmod(x, 8.0);
    if (r == 0.0) {
      r = 0.0;
    } else {
      if (x < 0.0) {
        r += 8.0;
      }
    }
  }

  return r;
}

/* Function for MATLAB Function: '<S3>/MATLAB Function' */
static void circshift(real_T a[4])
{
  real_T b_a_idx_1;
  real_T b_a_idx_2;
```

```
  real_T b_a_idx_3;
  b_a_idx_1 = a[2];
  b_a_idx_2 = a[3];
  b_a_idx_3 = a[0];
  a[0] = a[1];
  a[1] = b_a_idx_1;
  a[2] = b_a_idx_2;
  a[3] = b_a_idx_3;
}

/* Function for MATLAB Function: '<S3>/MATLAB Function' */
static real_T function_handle_parenReference(real_T varargin_1, real_T
  varargin_2)
{
  return (real_T)((uint64_T)varargin_1 ^ (uint64_T)varargin_2);
}

/* Function for MATLAB Function: '<S3>/MATLAB Function' */
static void KeyExpansion(const char_T key[32], real_T w[240])
{
  static const uint8_T b[256] = { 99U, 124U, 119U, 123U, 242U, 107U, 111U, 197U,
    48U, 1U, 103U, 43U, 254U, 215U, 171U, 118U, 202U, 130U, 201U, 125U, 250U,
    89U, 71U, 240U, 173U, 212U, 162U, 175U, 156U, 164U, 114U, 192U, 183U, 253U,
    147U, 38U, 54U, 63U, 247U, 204U, 52U, 165U, 229U, 241U, 113U, 216U, 49U, 21U,
    4U, 199U, 35U, 195U, 24U, 150U, 5U, 154U, 7U, 18U, 128U, 226U, 235U, 39U,
    178U, 117U, 9U, 131U, 44U, 26U, 27U, 110U, 90U, 160U, 82U, 59U, 214U, 179U,
    41U, 227U, 47U, 132U, 83U, 209U, 0U, 237U, 32U, 252U, 177U, 91U, 106U, 203U,
    190U, 57U, 74U, 76U, 88U, 207U, 208U, 239U, 170U, 251U, 67U, 77U, 51U, 133U,
    69U, 249U, 2U, 127U, 80U, 60U, 159U, 168U, 81U, 163U, 64U, 143U, 146U, 157U,
    56U, 245U, 188U, 182U, 218U, 33U, 16U, MAX_uint8_T, 243U, 210U, 205U, 12U,
    19U, 236U, 95U, 151U, 68U, 23U, 196U, 167U, 126U, 61U, 100U, 93U, 25U, 115U,
    96U, 129U, 79U, 220U, 34U, 42U, 144U, 136U, 70U, 238U, 184U, 20U, 222U, 94U,
    11U, 219U, 224U, 50U, 58U, 10U, 73U, 6U, 36U, 92U, 194U, 211U, 172U, 98U,
    145U, 149U, 228U, 121U, 231U, 200U, 55U, 109U, 141U, 213U, 78U, 169U, 108U,
    86U, 244U, 234U, 101U, 122U, 174U, 8U, 186U, 120U, 37U, 46U, 28U, 166U, 180U,
    198U, 232U, 221U, 116U, 31U, 75U, 189U, 139U, 138U, 112U, 62U, 181U, 102U,
    72U, 3U, 246U, 14U, 97U, 53U, 87U, 185U, 134U, 193U, 29U, 158U, 225U, 248U,
    152U, 17U, 105U, 217U, 142U, 148U, 155U, 30U, 135U, 233U, 206U, 85U, 40U,
    223U, 140U, 161U, 137U, 13U, 191U, 230U, 66U, 104U, 65U, 153U, 45U, 15U,
    176U, 84U, 187U, 22U };

  real_T temp[4];
  real_T n;
  real_T temp_tmp;
  real_T temp_tmp_0;
  real_T temp_tmp_1;
  real_T x;
  int32_T j;
  int32_T m;
  int32_T w_tmp;
  uint8_T w1[32];
  memset(&w[0], 0, 240U * sizeof(real_T));
  for (j = 0; j < 32; j++) {
    w1[j] = (uint8_T)key[j];
  }

  for (m = 0; m < 8; m++) {
    w_tmp = m << 2;
    w[w_tmp] = w1[w_tmp];
```

```
    w[w_tmp + 1] = w1[w_tmp + 1];
    w[w_tmp + 2] = w1[w_tmp + 2];
    w[w_tmp + 3] = w1[w_tmp + 3];
  }

  for (j = 0; j < 52; j++) {
    w_tmp = (j + 7) << 2;
    temp_tmp = w[w_tmp];
    temp[0] = temp_tmp;
    x = w[w_tmp + 1];
    temp[1] = x;
    temp_tmp_0 = w[w_tmp + 2];
    temp[2] = temp_tmp_0;
    temp_tmp_1 = w[w_tmp + 3];
    temp[3] = temp_tmp_1;
    n = mod((real_T)j + 8.0);
    if (n == 0.0) {
      n = 1.0;
      for (m = 0; m < ((real_T)j + 8.0) / 8.0 - 1.0; m++) {
        temp_tmp = n;
        n = 0.0;
        x = 1.0;
        while (x > 0.0) {
          temp_tmp = (real_T)((uint64_T)temp_tmp << 1);
          if (((uint64_T)temp_tmp & 256ULL) != 0ULL) {
            temp_tmp = (real_T)((uint64_T)temp_tmp & 18446744073709551359ULL);
            temp_tmp = (real_T)((uint64_T)temp_tmp ^ 27ULL);
          }

          if (((uint64_T)x & 1ULL) != 0ULL) {
            n = (real_T)((uint64_T)n ^ (uint64_T)temp_tmp);
          }

          x = (real_T)((uint64_T)x >> 1);
        }
      }

      circshift(temp);
     temp[0] = function_handle_parenReference((real_T)b[(int32_T)
     (temp[0] + 1.0)- 1], n);
      temp[1] = function_handle_parenReference((real_T)b[(int32_T)
      (temp[1] + 1.0)- 1], 0.0);
      temp[2] = function_handle_parenReference((real_T)b[(int32_T)
      (temp[2] + 1.0)- 1], 0.0);
      temp[3] = function_handle_parenReference((real_T)b[(int32_T)
      (temp[3] + 1.0)- 1], 0.0);
    } else {
      if (n == 4.0) {
        temp[0] = b[(int32_T)(temp_tmp + 1.0) - 1];
        temp[1] = b[(int32_T)(x + 1.0) - 1];
        temp[2] = b[(int32_T)(temp_tmp_0 + 1.0) - 1];
        temp[3] = b[(int32_T)(temp_tmp_1 + 1.0) - 1];
      }
    }

    w_tmp = j << 2;
    m = (j + 8) << 2;
    w[m] = function_handle_parenReference(w[w_tmp], temp[0]);
    w[m + 1] = function_handle_parenReference(w[w_tmp + 1], temp[1]);
```

```
    w[m + 2] = function_handle_parenReference(w[w_tmp + 2], temp[2]);
    w[m + 3] = function_handle_parenReference(w[w_tmp + 3], temp[3]);
  }
}

/* Function for MATLAB Function: '<S3>/MATLAB Function' */
static void circshift_h(real_T a[4])
{
  real_T buffer;
  int32_T d_k;
  buffer = a[0];
  for (d_k = 0; d_k < 3; d_k++) {
    a[d_k] = a[d_k + 1];
  }

  a[3] = buffer;
}

/* Function for MATLAB Function: '<S3>/MATLAB Function' */
static void circshift_h5(real_T a[4])
{
  real_T buffer[2];
  int32_T d_k;
  for (d_k = 0; d_k < 2; d_k++) {
    buffer[d_k] = a[d_k];
  }

  for (d_k = 0; d_k < 2; d_k++) {
    a[d_k] = a[d_k + 2];
  }

  for (d_k = 0; d_k < 2; d_k++) {
    a[d_k + 2] = buffer[d_k];
  }
}

/* Function for MATLAB Function: '<S3>/MATLAB Function' */
static void circshift_h5f(real_T a[4])
{
  real_T buffer;
  int32_T k;
  buffer = a[3];
  for (k = 4; k >= 2; k--) {
    a[k - 1] = a[k - 2];
  }

  a[0] = buffer;
}

/* Function for MATLAB Function: '<S3>/MATLAB Function' */
static real_T xtime(real_T x, real_T c)
{
  real_T a;
  a = 0.0;
  if (((uint64_T)x & 1ULL) != 0ULL) {
    a = c;
  }

  x = (real_T)((uint64_T)x >> 1);
```

```
  while (x > 0.0) {
    c = (real_T)((uint64_T)c << 1);
    if (((uint64_T)c & 256ULL) != 0ULL) {
      c = (real_T)((uint64_T)c & 18446744073709551359ULL);
      c = (real_T)((uint64_T)c ^ 27ULL);
    }

    if (((uint64_T)x & 1ULL) != 0ULL) {
      a = (real_T)((uint64_T)a ^ (uint64_T)c);
    }

    x = (real_T)((uint64_T)x >> 1);
  }

  return a;
}

/* Function for MATLAB Function: '<S3>/MATLAB Function' */
static void MixColumns(const real_T state[16], real_T State[16])
{
  State[0] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[0], 2.0) ^ (uint64_T)xtime(state[1], 3.0)) ^ (uint64_T)state[2]) ^
                      (uint64_T)state[3]);
  State[1] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[1], 2.0) ^ (uint64_T)xtime(state[2], 3.0)) ^ (uint64_T)state[0]) ^
                      (uint64_T)state[3]);
  State[2] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[2], 2.0) ^ (uint64_T)xtime(state[3], 3.0)) ^ (uint64_T)state[0]) ^
                      (uint64_T)state[1]);
  State[3] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[3], 2.0) ^ (uint64_T)xtime(state[0], 3.0)) ^ (uint64_T)state[1]) ^
                      (uint64_T)state[2]);
  State[4] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[4], 2.0) ^ (uint64_T)xtime(state[5], 3.0)) ^ (uint64_T)state[6]) ^
                      (uint64_T)state[7]);
  State[5] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[5], 2.0) ^ (uint64_T)xtime(state[6], 3.0)) ^ (uint64_T)state[4]) ^
                      (uint64_T)state[7]);
  State[6] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[6], 2.0) ^ (uint64_T)xtime(state[7], 3.0)) ^ (uint64_T)state[4]) ^
                      (uint64_T)state[5]);
  State[7] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[7], 2.0) ^ (uint64_T)xtime(state[4], 3.0)) ^ (uint64_T)state[5]) ^
                      (uint64_T)state[6]);
  State[8] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[8], 2.0) ^ (uint64_T)xtime(state[9], 3.0)) ^ (uint64_T)state[10]) ^
                      (uint64_T)state[11]);
  State[9] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[9], 2.0) ^ (uint64_T)xtime(state[10], 3.0)) ^ (uint64_T)state[8]) ^
                      (uint64_T)state[11]);
  State[10] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[10], 2.0) ^ (uint64_T)xtime(state[11], 3.0)) ^ (uint64_T)state[8]) ^
                       (uint64_T)state[9]);
  State[11] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[11], 2.0) ^ (uint64_T)xtime(state[8], 3.0)) ^ (uint64_T)state[9]) ^
                       (uint64_T)state[10]);
  State[12] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[12], 2.0) ^ (uint64_T)xtime(state[13], 3.0)) ^ (uint64_T)state[14]) ^
                       (uint64_T)state[15]);
```

```
  State[13] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[13], 2.0) ^ (uint64_T)xtime(state[14], 3.0)) ^ (uint64_T)state[12]) ^
                       (uint64_T)state[15]);
  State[14] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[14], 2.0) ^ (uint64_T)xtime(state[15], 3.0)) ^ (uint64_T)state[12]) ^
                       (uint64_T)state[13]);
  State[15] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (state[15], 2.0) ^ (uint64_T)xtime(state[12], 3.0)) ^ (uint64_T)state[13]) ^
                       (uint64_T)state[14]);
}

/* Function for MATLAB Function: '<S4>/MATLAB Function1' */
static void circshift_m1s(real_T a[4])
{
  real_T buffer[2];
  int32_T k;
  for (k = 0; k < 2; k++) {
    buffer[k] = a[k + 2];
  }

  for (k = 4; k >= 3; k--) {
    a[k - 1] = a[k - 3];
  }

  for (k = 0; k < 2; k++) {
    a[k] = buffer[k];
  }
}

/* Function for MATLAB Function: '<S4>/MATLAB Function1' */
static void ara_func(real_T state[16])
{
  static const uint8_T b[256] = { 82U, 9U, 106U, 213U, 48U, 54U, 165U, 56U, 191U,
    64U, 163U, 158U, 129U, 243U, 215U, 251U, 124U, 227U, 57U, 130U, 155U, 47U,
    MAX_uint8_T, 135U, 52U, 142U, 67U, 68U, 196U, 222U, 233U, 203U, 84U, 123U,
    148U, 50U, 166U, 194U, 35U, 61U, 238U, 76U, 149U, 11U, 66U, 250U, 195U, 78U,
    8U, 46U, 161U, 102U, 40U, 217U, 36U, 178U, 118U, 91U, 162U, 73U, 109U, 139U,
    209U, 37U, 114U, 248U, 246U, 100U, 134U, 104U, 152U, 22U, 212U, 164U, 92U,
    204U, 93U, 101U, 182U, 146U, 108U, 112U, 72U, 80U, 253U, 237U, 185U, 218U,
    94U, 21U, 70U, 87U, 167U, 141U, 157U, 132U, 144U, 216U, 171U, 0U, 140U, 188U,
    211U, 10U, 247U, 228U, 88U, 5U, 184U, 179U, 69U, 6U, 208U, 44U, 30U, 143U,
    202U, 63U, 15U, 2U, 193U, 175U, 189U, 3U, 1U, 19U, 138U, 107U, 58U, 145U,
    17U, 65U, 79U, 103U, 220U, 234U, 151U, 242U, 207U, 206U, 240U, 180U, 230U,
    115U, 150U, 172U, 116U, 34U, 231U, 173U, 53U, 133U, 226U, 249U, 55U, 232U,
    28U, 117U, 223U, 110U, 71U, 241U, 26U, 113U, 29U, 41U, 197U, 137U, 111U,
    183U, 98U, 14U, 170U, 24U, 190U, 27U, 252U, 86U, 62U, 75U, 198U, 210U, 121U,
    32U, 154U, 219U, 192U, 254U, 120U, 205U, 90U, 244U, 31U, 221U, 168U, 51U,
    136U, 7U, 199U, 49U, 177U, 18U, 16U, 89U, 39U, 128U, 236U, 95U, 96U, 81U,
    127U, 169U, 25U, 181U, 74U, 13U, 45U, 229U, 122U, 159U, 147U, 201U, 156U,
    239U, 160U, 224U, 59U, 77U, 174U, 42U, 245U, 176U, 200U, 235U, 187U, 60U,
    131U, 83U, 153U, 97U, 23U, 43U, 4U, 126U, 186U, 119U, 214U, 38U, 225U, 105U,
    20U, 99U, 85U, 33U, 12U, 125U };

  real_T b_state[16];
  real_T c[4];
  int32_T i;
  memcpy(&b_state[0], &state[0], sizeof(real_T) << 4U);
  state[0] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[0], 14.0) ^ (uint64_T)xtime(b_state[1], 11.0)) ^ (uint64_T)xtime
```

```
    (b_state[2], 13.0)) ^ (uint64_T)xtime(b_state[3], 9.0));
  state[1] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[0], 9.0) ^ (uint64_T)xtime(b_state[1], 14.0)) ^ (uint64_T)xtime
    (b_state[2], 11.0)) ^ (uint64_T)xtime(b_state[3], 13.0));
  state[2] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[0], 13.0) ^ (uint64_T)xtime(b_state[1], 9.0)) ^ (uint64_T)xtime
    (b_state[2], 14.0)) ^ (uint64_T)xtime(b_state[3], 11.0));
  state[3] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[0], 11.0) ^ (uint64_T)xtime(b_state[1], 13.0)) ^ (uint64_T)xtime
    (b_state[2], 9.0)) ^ (uint64_T)xtime(b_state[3], 14.0));
  state[4] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[4], 14.0) ^ (uint64_T)xtime(b_state[5], 11.0)) ^ (uint64_T)xtime
    (b_state[6], 13.0)) ^ (uint64_T)xtime(b_state[7], 9.0));
  state[5] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[4], 9.0) ^ (uint64_T)xtime(b_state[5], 14.0)) ^ (uint64_T)xtime
    (b_state[6], 11.0)) ^ (uint64_T)xtime(b_state[7], 13.0));
  state[6] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[4], 13.0) ^ (uint64_T)xtime(b_state[5], 9.0)) ^ (uint64_T)xtime
    (b_state[6], 14.0)) ^ (uint64_T)xtime(b_state[7], 11.0));
  state[7] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[4], 11.0) ^ (uint64_T)xtime(b_state[5], 13.0)) ^ (uint64_T)xtime
    (b_state[6], 9.0)) ^ (uint64_T)xtime(b_state[7], 14.0));
  state[8] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[8], 14.0) ^ (uint64_T)xtime(b_state[9], 11.0)) ^ (uint64_T)xtime
    (b_state[10], 13.0)) ^ (uint64_T)xtime(b_state[11], 9.0));
  state[9] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[8], 9.0) ^ (uint64_T)xtime(b_state[9], 14.0)) ^ (uint64_T)xtime
    (b_state[10], 11.0)) ^ (uint64_T)xtime(b_state[11], 13.0));
  state[10] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[8], 13.0) ^ (uint64_T)xtime(b_state[9], 9.0)) ^ (uint64_T)xtime
    (b_state[10], 14.0)) ^ (uint64_T)xtime(b_state[11], 11.0));
  state[11] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[8], 11.0) ^ (uint64_T)xtime(b_state[9], 13.0)) ^ (uint64_T)xtime
    (b_state[10], 9.0)) ^ (uint64_T)xtime(b_state[11], 14.0));
  state[12] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[12], 14.0) ^ (uint64_T)xtime(b_state[13], 11.0)) ^ (uint64_T)xtime
    (b_state[14], 13.0)) ^ (uint64_T)xtime(b_state[15], 9.0));
  state[13] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[12], 9.0) ^ (uint64_T)xtime(b_state[13], 14.0)) ^ (uint64_T)xtime
    (b_state[14], 11.0)) ^ (uint64_T)xtime(b_state[15], 13.0));
  state[14] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[12], 13.0) ^ (uint64_T)xtime(b_state[13], 9.0)) ^ (uint64_T)xtime
    (b_state[14], 14.0)) ^ (uint64_T)xtime(b_state[15], 11.0));
  state[15] = (real_T)((uint64_T)(real_T)((uint64_T)(real_T)((uint64_T)xtime
    (b_state[12], 11.0) ^ (uint64_T)xtime(b_state[13], 13.0)) ^ (uint64_T)xtime
    (b_state[14], 9.0)) ^ (uint64_T)xtime(b_state[15], 14.0));
  c[0] = state[1];
  c[1] = state[5];
  c[2] = state[9];
  c[3] = state[13];
  circshift_h5f(c);
  state[1] = c[0];
  c[0] = state[2];
  state[5] = c[1];
  c[1] = state[6];
  state[9] = c[2];
  c[2] = state[10];
  state[13] = c[3];
  c[3] = state[14];
```

```
643   circshift_m1s(c);
644   state[2] = c[0];
645   c[0] = state[3];
646   state[6] = c[1];
647   c[1] = state[7];
648   state[10] = c[2];
649   c[2] = state[11];
650   state[14] = c[3];
651   c[3] = state[15];
652   circshift_h(c);
653   state[3] = c[0];
654   state[7] = c[1];
655   state[11] = c[2];
656   state[15] = c[3];
657   for (i = 0; i < 16; i++) {
658     state[i] = b[(int32_T)(state[i] + 1.0) - 1];
659   }
660 }
661
662 /* Model step function */
663 void Encryption_step(void)
664 {
665   static const uint8_T b[256] = { 99U, 124U, 119U, 123U, 242U, 107U, 111U, 197U,
666     48U, 1U, 103U, 43U, 254U, 215U, 171U, 118U, 202U, 130U, 201U, 125U, 250U,
667     89U, 71U, 240U, 173U, 212U, 162U, 175U, 156U, 164U, 114U, 192U, 183U, 253U,
668     147U, 38U, 54U, 63U, 247U, 204U, 52U, 165U, 229U, 241U, 113U, 216U, 49U, 21U,
669     4U, 199U, 35U, 195U, 24U, 150U, 5U, 154U, 7U, 18U, 128U, 226U, 235U, 39U,
670     178U, 117U, 9U, 131U, 44U, 26U, 27U, 110U, 90U, 160U, 82U, 59U, 214U, 179U,
671     41U, 227U, 47U, 132U, 83U, 209U, 0U, 237U, 32U, 252U, 177U, 91U, 106U, 203U,
672     190U, 57U, 74U, 76U, 88U, 207U, 208U, 239U, 170U, 251U, 67U, 77U, 51U, 133U,
673     69U, 249U, 2U, 127U, 80U, 60U, 159U, 168U, 81U, 163U, 64U, 143U, 146U, 157U,
674     56U, 245U, 188U, 182U, 218U, 33U, 16U, MAX_uint8_T, 243U, 210U, 205U, 12U,
675     19U, 236U, 95U, 151U, 68U, 23U, 196U, 167U, 126U, 61U, 100U, 93U, 25U, 115U,
676     96U, 129U, 79U, 220U, 34U, 42U, 144U, 136U, 70U, 238U, 184U, 20U, 222U, 94U,
677     11U, 219U, 224U, 50U, 58U, 10U, 73U, 6U, 36U, 92U, 194U, 211U, 172U, 98U,
678     145U, 149U, 228U, 121U, 231U, 200U, 55U, 109U, 141U, 213U, 78U, 169U, 108U,
679     86U, 244U, 234U, 101U, 122U, 174U, 8U, 186U, 120U, 37U, 46U, 28U, 166U, 180U,
680     198U, 232U, 221U, 116U, 31U, 75U, 189U, 139U, 138U, 112U, 62U, 181U, 102U,
681     72U, 3U, 246U, 14U, 97U, 53U, 87U, 185U, 134U, 193U, 29U, 158U, 225U, 248U,
682     152U, 17U, 105U, 217U, 142U, 148U, 155U, 30U, 135U, 233U, 206U, 85U, 40U,
683     223U, 140U, 161U, 137U, 13U, 191U, 230U, 66U, 104U, 65U, 153U, 45U, 15U,
684     176U, 84U, 187U, 22U };
685
686   static const uint8_T b_0[256] = { 82U, 9U, 106U, 213U, 48U, 54U, 165U, 56U,
687     191U, 64U, 163U, 158U, 129U, 243U, 215U, 251U, 124U, 227U, 57U, 130U, 155U,
688     47U, MAX_uint8_T, 135U, 52U, 142U, 67U, 68U, 196U, 222U, 233U, 203U, 84U,
689     123U, 148U, 50U, 166U, 194U, 35U, 61U, 238U, 76U, 149U, 11U, 66U, 250U, 195U,
690     78U, 8U, 46U, 161U, 102U, 40U, 217U, 36U, 178U, 118U, 91U, 162U, 73U, 109U,
691     139U, 209U, 37U, 114U, 248U, 246U, 100U, 134U, 104U, 152U, 22U, 212U, 164U,
692     92U, 204U, 93U, 101U, 182U, 146U, 108U, 112U, 72U, 80U, 253U, 237U, 185U,
693     218U, 94U, 21U, 70U, 87U, 167U, 141U, 157U, 132U, 144U, 216U, 171U, 0U, 140U,
694     188U, 211U, 10U, 247U, 228U, 88U, 5U, 184U, 179U, 69U, 6U, 208U, 44U, 30U,
695     143U, 202U, 63U, 15U, 2U, 193U, 175U, 189U, 3U, 1U, 19U, 138U, 107U, 58U,
696     145U, 17U, 65U, 79U, 103U, 220U, 234U, 151U, 242U, 207U, 206U, 240U, 180U,
697     230U, 115U, 150U, 172U, 116U, 34U, 231U, 173U, 53U, 133U, 226U, 249U, 55U,
698     232U, 28U, 117U, 223U, 110U, 71U, 241U, 26U, 113U, 29U, 41U, 197U, 137U,
699     111U, 183U, 98U, 14U, 170U, 24U, 190U, 27U, 252U, 86U, 62U, 75U, 198U, 210U,
700     121U, 32U, 154U, 219U, 192U, 254U, 120U, 205U, 90U, 244U, 31U, 221U, 168U,
701     51U, 136U, 7U, 199U, 49U, 177U, 18U, 16U, 89U, 39U, 128U, 236U, 95U, 96U,
```

```
    81U, 127U, 169U, 25U, 181U, 74U, 13U, 45U, 229U, 122U, 159U, 147U, 201U,
    156U, 239U, 160U, 224U, 59U, 77U, 174U, 42U, 245U, 176U, 200U, 235U, 187U,
    60U, 131U, 83U, 153U, 97U, 23U, 43U, 4U, 126U, 186U, 119U, 214U, 38U, 225U,
    105U, 20U, 99U, 85U, 33U, 12U, 125U };

  real_T w[240];
  real_T rtb_Out_g[16];
  real_T rtb_Out_j[16];
  real_T c[4];
  int32_T k;
  int32_T rtb_Out_j_tmp;
  int32_T rtb_Out_j_tmp_0;
  int32_T rtb_Out_j_tmp_1;
  char_T rtb_DataTypeConversion2_0[32];
  uint8_T rtb_DataTypeConversion2[32];
  for (k = 0; k < 32; k++) {
    /* MATLAB Function: '<S3>/MATLAB Function' incorporates:
     *  DataTypeConversion: '<S3>/Data Type Conversion2'
     *  MATLAB Function: '<S3>/MATLAB Function1'
     */
    rtb_DataTypeConversion2_0[k] = (int8_T)k;

    /* DataTypeConversion: '<S3>/Data Type Conversion2' incorporates:
     *  MATLAB Function: '<S3>/MATLAB Function1'
     */
    rtb_DataTypeConversion2[k] = (uint8_T)k;
  }

  /* MATLAB Function: '<S3>/MATLAB Function' incorporates:
   *  Constant: '<S1>/Constant4'
   *  DataTypeConversion: '<S3>/Data Type Conversion'
   *  Inport: '<Root>/PlainText'
   */
  KeyExpansion(rtb_DataTypeConversion2_0, w);
  memset(&rtb_Out_g[0], 0, 15U * sizeof(real_T));
  rtb_Out_g[15] = rtU.PlainText;
  for (k = 0; k < 4; k++) {
    rtb_Out_j_tmp = k << 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp]);
    rtb_Out_g[rtb_Out_j_tmp + 1] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp + 1], w[rtb_Out_j_tmp + 1]);
    rtb_Out_g[rtb_Out_j_tmp + 2] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp + 2], w[rtb_Out_j_tmp + 2]);
    rtb_Out_g[rtb_Out_j_tmp + 3] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp + 3], w[rtb_Out_j_tmp + 3]);
  }

  for (k = 0; k < 16; k++) {
    rtb_Out_g[k] = b[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
  }

  c[0] = rtb_Out_g[1];
  c[1] = rtb_Out_g[5];
  c[2] = rtb_Out_g[9];
  c[3] = rtb_Out_g[13];
  circshift_h(c);
  rtb_Out_g[1] = c[0];
  c[0] = rtb_Out_g[2];
```

```
761   rtb_Out_g[5] = c[1];
762   c[1] = rtb_Out_g[6];
763   rtb_Out_g[9] = c[2];
764   c[2] = rtb_Out_g[10];
765   rtb_Out_g[13] = c[3];
766   c[3] = rtb_Out_g[14];
767   circshift_h5(c);
768   rtb_Out_g[2] = c[0];
769   c[0] = rtb_Out_g[3];
770   rtb_Out_g[6] = c[1];
771   c[1] = rtb_Out_g[7];
772   rtb_Out_g[10] = c[2];
773   c[2] = rtb_Out_g[11];
774   rtb_Out_g[14] = c[3];
775   c[3] = rtb_Out_g[15];
776   circshift_h5f(c);
777   rtb_Out_g[3] = c[0];
778   rtb_Out_g[7] = c[1];
779   rtb_Out_g[11] = c[2];
780   rtb_Out_g[15] = c[3];
781   memcpy(&rtb_Out_j[0], &rtb_Out_g[0], sizeof(real_T) << 4);
782   MixColumns(rtb_Out_j, rtb_Out_g);
783   for (k = 0; k < 4; k++) {
784     rtb_Out_j_tmp = k << 2;
785     rtb_Out_j_tmp_0 = (k + 4) << 2;
786     rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
787       (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
788     rtb_Out_j_tmp = (k << 2) + 1;
789     rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
790       (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 1]);
791     rtb_Out_j_tmp = (k << 2) + 2;
792     rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
793       (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 2]);
794     rtb_Out_j_tmp = (k << 2) + 3;
795     rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
796       (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 3]);
797   }
798
799   for (k = 0; k < 16; k++) {
800     rtb_Out_g[k] = b[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
801   }
802
803   c[0] = rtb_Out_g[1];
804   c[1] = rtb_Out_g[5];
805   c[2] = rtb_Out_g[9];
806   c[3] = rtb_Out_g[13];
807   circshift_h(c);
808   rtb_Out_g[1] = c[0];
809   c[0] = rtb_Out_g[2];
810   rtb_Out_g[5] = c[1];
811   c[1] = rtb_Out_g[6];
812   rtb_Out_g[9] = c[2];
813   c[2] = rtb_Out_g[10];
814   rtb_Out_g[13] = c[3];
815   c[3] = rtb_Out_g[14];
816   circshift_h5(c);
817   rtb_Out_g[2] = c[0];
818   c[0] = rtb_Out_g[3];
819   rtb_Out_g[6] = c[1];
```

```
820   c[1] = rtb_Out_g[7];
821   rtb_Out_g[10] = c[2];
822   c[2] = rtb_Out_g[11];
823   rtb_Out_g[14] = c[3];
824   c[3] = rtb_Out_g[15];
825   circshift_h5f(c);
826   rtb_Out_g[3] = c[0];
827   rtb_Out_g[7] = c[1];
828   rtb_Out_g[11] = c[2];
829   rtb_Out_g[15] = c[3];
830   memcpy(&rtb_Out_j[0], &rtb_Out_g[0], sizeof(real_T) << 4);
831   MixColumns(rtb_Out_j, rtb_Out_g);
832   for (k = 0; k < 4; k++) {
833     rtb_Out_j_tmp = k << 2;
834     rtb_Out_j_tmp_0 = (k + 8) << 2;
835     rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
836       (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
837     rtb_Out_j_tmp = (k << 2) + 1;
838     rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
839       (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 1]);
840     rtb_Out_j_tmp = (k << 2) + 2;
841     rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
842       (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 2]);
843     rtb_Out_j_tmp = (k << 2) + 3;
844     rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
845       (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 3]);
846   }
847
848   for (k = 0; k < 16; k++) {
849     rtb_Out_g[k] = b[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
850   }
851
852   c[0] = rtb_Out_g[1];
853   c[1] = rtb_Out_g[5];
854   c[2] = rtb_Out_g[9];
855   c[3] = rtb_Out_g[13];
856   circshift_h(c);
857   rtb_Out_g[1] = c[0];
858   c[0] = rtb_Out_g[2];
859   rtb_Out_g[5] = c[1];
860   c[1] = rtb_Out_g[6];
861   rtb_Out_g[9] = c[2];
862   c[2] = rtb_Out_g[10];
863   rtb_Out_g[13] = c[3];
864   c[3] = rtb_Out_g[14];
865   circshift_h5(c);
866   rtb_Out_g[2] = c[0];
867   c[0] = rtb_Out_g[3];
868   rtb_Out_g[6] = c[1];
869   c[1] = rtb_Out_g[7];
870   rtb_Out_g[10] = c[2];
871   c[2] = rtb_Out_g[11];
872   rtb_Out_g[14] = c[3];
873   c[3] = rtb_Out_g[15];
874   circshift_h5f(c);
875   rtb_Out_g[3] = c[0];
876   rtb_Out_g[7] = c[1];
877   rtb_Out_g[11] = c[2];
878   rtb_Out_g[15] = c[3];
```

```
879   memcpy(&rtb_Out_j[0], &rtb_Out_g[0], sizeof(real_T) << 4);
880   MixColumns(rtb_Out_j, rtb_Out_g);
881   for (k = 0; k < 4; k++) {
882     rtb_Out_j_tmp = k << 2;
883     rtb_Out_j_tmp_0 = (k + 12) << 2;
884     rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
885       (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
886     rtb_Out_j_tmp = (k << 2) + 1;
887     rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
888       (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 1]);
889     rtb_Out_j_tmp = (k << 2) + 2;
890     rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
891       (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 2]);
892     rtb_Out_j_tmp = (k << 2) + 3;
893     rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
894       (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 3]);
895   }
896
897   for (k = 0; k < 16; k++) {
898     rtb_Out_g[k] = b[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
899   }
900
901   c[0] = rtb_Out_g[1];
902   c[1] = rtb_Out_g[5];
903   c[2] = rtb_Out_g[9];
904   c[3] = rtb_Out_g[13];
905   circshift_h(c);
906   rtb_Out_g[1] = c[0];
907   c[0] = rtb_Out_g[2];
908   rtb_Out_g[5] = c[1];
909   c[1] = rtb_Out_g[6];
910   rtb_Out_g[9] = c[2];
911   c[2] = rtb_Out_g[10];
912   rtb_Out_g[13] = c[3];
913   c[3] = rtb_Out_g[14];
914   circshift_h5(c);
915   rtb_Out_g[2] = c[0];
916   c[0] = rtb_Out_g[3];
917   rtb_Out_g[6] = c[1];
918   c[1] = rtb_Out_g[7];
919   rtb_Out_g[10] = c[2];
920   c[2] = rtb_Out_g[11];
921   rtb_Out_g[14] = c[3];
922   c[3] = rtb_Out_g[15];
923   circshift_h5f(c);
924   rtb_Out_g[3] = c[0];
925   rtb_Out_g[7] = c[1];
926   rtb_Out_g[11] = c[2];
927   rtb_Out_g[15] = c[3];
928   memcpy(&rtb_Out_j[0], &rtb_Out_g[0], sizeof(real_T) << 4);
929   MixColumns(rtb_Out_j, rtb_Out_g);
930   for (k = 0; k < 4; k++) {
931     rtb_Out_j_tmp = k << 2;
932     rtb_Out_j_tmp_0 = (k + 16) << 2;
933     rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
934       (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
935     rtb_Out_j_tmp = (k << 2) + 1;
936     rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
937       (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 1]);
```

```
    rtb_Out_j_tmp = (k << 2) + 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 2]);
    rtb_Out_j_tmp = (k << 2) + 3;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 3]);
  }

  for (k = 0; k < 16; k++) {
    rtb_Out_g[k] = b[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
  }

  c[0] = rtb_Out_g[1];
  c[1] = rtb_Out_g[5];
  c[2] = rtb_Out_g[9];
  c[3] = rtb_Out_g[13];
  circshift_h(c);
  rtb_Out_g[1] = c[0];
  c[0] = rtb_Out_g[2];
  rtb_Out_g[5] = c[1];
  c[1] = rtb_Out_g[6];
  rtb_Out_g[9] = c[2];
  c[2] = rtb_Out_g[10];
  rtb_Out_g[13] = c[3];
  c[3] = rtb_Out_g[14];
  circshift_h5(c);
  rtb_Out_g[2] = c[0];
  c[0] = rtb_Out_g[3];
  rtb_Out_g[6] = c[1];
  c[1] = rtb_Out_g[7];
  rtb_Out_g[10] = c[2];
  c[2] = rtb_Out_g[11];
  rtb_Out_g[14] = c[3];
  c[3] = rtb_Out_g[15];
  circshift_h5f(c);
  rtb_Out_g[3] = c[0];
  rtb_Out_g[7] = c[1];
  rtb_Out_g[11] = c[2];
  rtb_Out_g[15] = c[3];
  memcpy(&rtb_Out_j[0], &rtb_Out_g[0], sizeof(real_T) << 4);
  MixColumns(rtb_Out_j, rtb_Out_g);
  for (k = 0; k < 4; k++) {
    rtb_Out_j_tmp = k << 2;
    rtb_Out_j_tmp_0 = (k + 20) << 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
    rtb_Out_j_tmp = (k << 2) + 1;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 1]);
    rtb_Out_j_tmp = (k << 2) + 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 2]);
    rtb_Out_j_tmp = (k << 2) + 3;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 3]);
  }

  for (k = 0; k < 16; k++) {
    rtb_Out_g[k] = b[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
```

```
997   }
998
999   c[0] = rtb_Out_g[1];
1000  c[1] = rtb_Out_g[5];
1001  c[2] = rtb_Out_g[9];
1002  c[3] = rtb_Out_g[13];
1003  circshift_h(c);
1004  rtb_Out_g[1] = c[0];
1005  c[0] = rtb_Out_g[2];
1006  rtb_Out_g[5] = c[1];
1007  c[1] = rtb_Out_g[6];
1008  rtb_Out_g[9] = c[2];
1009  c[2] = rtb_Out_g[10];
1010  rtb_Out_g[13] = c[3];
1011  c[3] = rtb_Out_g[14];
1012  circshift_h5(c);
1013  rtb_Out_g[2] = c[0];
1014  c[0] = rtb_Out_g[3];
1015  rtb_Out_g[6] = c[1];
1016  c[1] = rtb_Out_g[7];
1017  rtb_Out_g[10] = c[2];
1018  c[2] = rtb_Out_g[11];
1019  rtb_Out_g[14] = c[3];
1020  c[3] = rtb_Out_g[15];
1021  circshift_h5f(c);
1022  rtb_Out_g[3] = c[0];
1023  rtb_Out_g[7] = c[1];
1024  rtb_Out_g[11] = c[2];
1025  rtb_Out_g[15] = c[3];
1026  memcpy(&rtb_Out_j[0], &rtb_Out_g[0], sizeof(real_T) << 4);
1027  MixColumns(rtb_Out_j, rtb_Out_g);
1028  for (k = 0; k < 4; k++) {
1029    rtb_Out_j_tmp = k << 2;
1030    rtb_Out_j_tmp_0 = (k + 24) << 2;
1031    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1032      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
1033    rtb_Out_j_tmp = (k << 2) + 1;
1034    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1035      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 1]);
1036    rtb_Out_j_tmp = (k << 2) + 2;
1037    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1038      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 2]);
1039    rtb_Out_j_tmp = (k << 2) + 3;
1040    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1041      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 3]);
1042  }
1043
1044  for (k = 0; k < 16; k++) {
1045    rtb_Out_g[k] = b[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
1046  }
1047
1048  c[0] = rtb_Out_g[1];
1049  c[1] = rtb_Out_g[5];
1050  c[2] = rtb_Out_g[9];
1051  c[3] = rtb_Out_g[13];
1052  circshift_h(c);
1053  rtb_Out_g[1] = c[0];
1054  c[0] = rtb_Out_g[2];
1055  rtb_Out_g[5] = c[1];
```

```
  c[1] = rtb_Out_g[6];
  rtb_Out_g[9] = c[2];
  c[2] = rtb_Out_g[10];
  rtb_Out_g[13] = c[3];
  c[3] = rtb_Out_g[14];
  circshift_h5(c);
  rtb_Out_g[2] = c[0];
  c[0] = rtb_Out_g[3];
  rtb_Out_g[6] = c[1];
  c[1] = rtb_Out_g[7];
  rtb_Out_g[10] = c[2];
  c[2] = rtb_Out_g[11];
  rtb_Out_g[14] = c[3];
  c[3] = rtb_Out_g[15];
  circshift_h5f(c);
  rtb_Out_g[3] = c[0];
  rtb_Out_g[7] = c[1];
  rtb_Out_g[11] = c[2];
  rtb_Out_g[15] = c[3];
  memcpy(&rtb_Out_j[0], &rtb_Out_g[0], sizeof(real_T) << 4);
  MixColumns(rtb_Out_j, rtb_Out_g);
  for (k = 0; k < 4; k++) {
    rtb_Out_j_tmp = k << 2;
    rtb_Out_j_tmp_0 = (k + 28) << 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
    rtb_Out_j_tmp = (k << 2) + 1;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 1]);
    rtb_Out_j_tmp = (k << 2) + 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 2]);
    rtb_Out_j_tmp = (k << 2) + 3;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 3]);
  }

  for (k = 0; k < 16; k++) {
    rtb_Out_g[k] = b[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
  }

  c[0] = rtb_Out_g[1];
  c[1] = rtb_Out_g[5];
  c[2] = rtb_Out_g[9];
  c[3] = rtb_Out_g[13];
  circshift_h(c);
  rtb_Out_g[1] = c[0];
  c[0] = rtb_Out_g[2];
  rtb_Out_g[5] = c[1];
  c[1] = rtb_Out_g[6];
  rtb_Out_g[9] = c[2];
  c[2] = rtb_Out_g[10];
  rtb_Out_g[13] = c[3];
  c[3] = rtb_Out_g[14];
  circshift_h5(c);
  rtb_Out_g[2] = c[0];
  c[0] = rtb_Out_g[3];
  rtb_Out_g[6] = c[1];
  c[1] = rtb_Out_g[7];
```

```
1115  rtb_Out_g[10] = c[2];
1116  c[2] = rtb_Out_g[11];
1117  rtb_Out_g[14] = c[3];
1118  c[3] = rtb_Out_g[15];
1119  circshift_h5f(c);
1120  rtb_Out_g[3] = c[0];
1121  rtb_Out_g[7] = c[1];
1122  rtb_Out_g[11] = c[2];
1123  rtb_Out_g[15] = c[3];
1124  memcpy(&rtb_Out_j[0], &rtb_Out_g[0], sizeof(real_T) << 4);
1125  MixColumns(rtb_Out_j, rtb_Out_g);
1126  for (k = 0; k < 4; k++) {
1127    rtb_Out_j_tmp = k << 2;
1128    rtb_Out_j_tmp_0 = (k + 32) << 2;
1129    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1130      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
1131    rtb_Out_j_tmp = (k << 2) + 1;
1132    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1133      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 1]);
1134    rtb_Out_j_tmp = (k << 2) + 2;
1135    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1136      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 2]);
1137    rtb_Out_j_tmp = (k << 2) + 3;
1138    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1139      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 3]);
1140  }
1141
1142  for (k = 0; k < 16; k++) {
1143    rtb_Out_g[k] = b[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
1144  }
1145
1146  c[0] = rtb_Out_g[1];
1147  c[1] = rtb_Out_g[5];
1148  c[2] = rtb_Out_g[9];
1149  c[3] = rtb_Out_g[13];
1150  circshift_h(c);
1151  rtb_Out_g[1] = c[0];
1152  c[0] = rtb_Out_g[2];
1153  rtb_Out_g[5] = c[1];
1154  c[1] = rtb_Out_g[6];
1155  rtb_Out_g[9] = c[2];
1156  c[2] = rtb_Out_g[10];
1157  rtb_Out_g[13] = c[3];
1158  c[3] = rtb_Out_g[14];
1159  circshift_h5(c);
1160  rtb_Out_g[2] = c[0];
1161  c[0] = rtb_Out_g[3];
1162  rtb_Out_g[6] = c[1];
1163  c[1] = rtb_Out_g[7];
1164  rtb_Out_g[10] = c[2];
1165  c[2] = rtb_Out_g[11];
1166  rtb_Out_g[14] = c[3];
1167  c[3] = rtb_Out_g[15];
1168  circshift_h5f(c);
1169  rtb_Out_g[3] = c[0];
1170  rtb_Out_g[7] = c[1];
1171  rtb_Out_g[11] = c[2];
1172  rtb_Out_g[15] = c[3];
1173  memcpy(&rtb_Out_j[0], &rtb_Out_g[0], sizeof(real_T) << 4);
```

```
1174  MixColumns(rtb_Out_j, rtb_Out_g);
1175  for (k = 0; k < 4; k++) {
1176    rtb_Out_j_tmp = k << 2;
1177    rtb_Out_j_tmp_0 = (k + 36) << 2;
1178    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1179      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
1180    rtb_Out_j_tmp = (k << 2) + 1;
1181    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1182      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 1]);
1183    rtb_Out_j_tmp = (k << 2) + 2;
1184    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1185      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 2]);
1186    rtb_Out_j_tmp = (k << 2) + 3;
1187    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1188      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 3]);
1189  }
1190
1191  for (k = 0; k < 16; k++) {
1192    rtb_Out_g[k] = b[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
1193  }
1194
1195  c[0] = rtb_Out_g[1];
1196  c[1] = rtb_Out_g[5];
1197  c[2] = rtb_Out_g[9];
1198  c[3] = rtb_Out_g[13];
1199  circshift_h(c);
1200  rtb_Out_g[1] = c[0];
1201  c[0] = rtb_Out_g[2];
1202  rtb_Out_g[5] = c[1];
1203  c[1] = rtb_Out_g[6];
1204  rtb_Out_g[9] = c[2];
1205  c[2] = rtb_Out_g[10];
1206  rtb_Out_g[13] = c[3];
1207  c[3] = rtb_Out_g[14];
1208  circshift_h5(c);
1209  rtb_Out_g[2] = c[0];
1210  c[0] = rtb_Out_g[3];
1211  rtb_Out_g[6] = c[1];
1212  c[1] = rtb_Out_g[7];
1213  rtb_Out_g[10] = c[2];
1214  c[2] = rtb_Out_g[11];
1215  rtb_Out_g[14] = c[3];
1216  c[3] = rtb_Out_g[15];
1217  circshift_h5f(c);
1218  rtb_Out_g[3] = c[0];
1219  rtb_Out_g[7] = c[1];
1220  rtb_Out_g[11] = c[2];
1221  rtb_Out_g[15] = c[3];
1222  memcpy(&rtb_Out_j[0], &rtb_Out_g[0], sizeof(real_T) << 4);
1223  MixColumns(rtb_Out_j, rtb_Out_g);
1224  for (k = 0; k < 4; k++) {
1225    rtb_Out_j_tmp = k << 2;
1226    rtb_Out_j_tmp_0 = (k + 40) << 2;
1227    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1228      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
1229    rtb_Out_j_tmp = (k << 2) + 1;
1230    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1231      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 1]);
1232    rtb_Out_j_tmp = (k << 2) + 2;
```

```
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 2]);
    rtb_Out_j_tmp = (k << 2) + 3;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 3]);
  }

  for (k = 0; k < 16; k++) {
    rtb_Out_g[k] = b[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
  }

  c[0] = rtb_Out_g[1];
  c[1] = rtb_Out_g[5];
  c[2] = rtb_Out_g[9];
  c[3] = rtb_Out_g[13];
  circshift_h(c);
  rtb_Out_g[1] = c[0];
  c[0] = rtb_Out_g[2];
  rtb_Out_g[5] = c[1];
  c[1] = rtb_Out_g[6];
  rtb_Out_g[9] = c[2];
  c[2] = rtb_Out_g[10];
  rtb_Out_g[13] = c[3];
  c[3] = rtb_Out_g[14];
  circshift_h5(c);
  rtb_Out_g[2] = c[0];
  c[0] = rtb_Out_g[3];
  rtb_Out_g[6] = c[1];
  c[1] = rtb_Out_g[7];
  rtb_Out_g[10] = c[2];
  c[2] = rtb_Out_g[11];
  rtb_Out_g[14] = c[3];
  c[3] = rtb_Out_g[15];
  circshift_h5f(c);
  rtb_Out_g[3] = c[0];
  rtb_Out_g[7] = c[1];
  rtb_Out_g[11] = c[2];
  rtb_Out_g[15] = c[3];
  memcpy(&rtb_Out_j[0], &rtb_Out_g[0], sizeof(real_T) << 4);
  MixColumns(rtb_Out_j, rtb_Out_g);
  for (k = 0; k < 4; k++) {
    rtb_Out_j_tmp = k << 2;
    rtb_Out_j_tmp_0 = (k + 44) << 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
    rtb_Out_j_tmp = (k << 2) + 1;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 1]);
    rtb_Out_j_tmp = (k << 2) + 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 2]);
    rtb_Out_j_tmp = (k << 2) + 3;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 3]);
  }

  for (k = 0; k < 16; k++) {
    rtb_Out_g[k] = b[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
  }
```

```
1292
1293    c[0] = rtb_Out_g[1];
1294    c[1] = rtb_Out_g[5];
1295    c[2] = rtb_Out_g[9];
1296    c[3] = rtb_Out_g[13];
1297    circshift_h(c);
1298    rtb_Out_g[1] = c[0];
1299    c[0] = rtb_Out_g[2];
1300    rtb_Out_g[5] = c[1];
1301    c[1] = rtb_Out_g[6];
1302    rtb_Out_g[9] = c[2];
1303    c[2] = rtb_Out_g[10];
1304    rtb_Out_g[13] = c[3];
1305    c[3] = rtb_Out_g[14];
1306    circshift_h5(c);
1307    rtb_Out_g[2] = c[0];
1308    c[0] = rtb_Out_g[3];
1309    rtb_Out_g[6] = c[1];
1310    c[1] = rtb_Out_g[7];
1311    rtb_Out_g[10] = c[2];
1312    c[2] = rtb_Out_g[11];
1313    rtb_Out_g[14] = c[3];
1314    c[3] = rtb_Out_g[15];
1315    circshift_h5f(c);
1316    rtb_Out_g[3] = c[0];
1317    rtb_Out_g[7] = c[1];
1318    rtb_Out_g[11] = c[2];
1319    rtb_Out_g[15] = c[3];
1320    memcpy(&rtb_Out_j[0], &rtb_Out_g[0], sizeof(real_T) << 4);
1321    MixColumns(rtb_Out_j, rtb_Out_g);
1322    for (k = 0; k < 4; k++) {
1323      rtb_Out_j_tmp = k << 2;
1324      rtb_Out_j_tmp_0 = (k + 48) << 2;
1325      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1326        (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
1327      rtb_Out_j_tmp = (k << 2) + 1;
1328      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1329        (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 1]);
1330      rtb_Out_j_tmp = (k << 2) + 2;
1331      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1332        (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 2]);
1333      rtb_Out_j_tmp = (k << 2) + 3;
1334      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1335        (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 3]);
1336    }
1337
1338    for (k = 0; k < 16; k++) {
1339      rtb_Out_g[k] = b[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
1340    }
1341
1342    c[0] = rtb_Out_g[1];
1343    c[1] = rtb_Out_g[5];
1344    c[2] = rtb_Out_g[9];
1345    c[3] = rtb_Out_g[13];
1346    circshift_h(c);
1347    rtb_Out_g[1] = c[0];
1348    c[0] = rtb_Out_g[2];
1349    rtb_Out_g[5] = c[1];
1350    c[1] = rtb_Out_g[6];
```

```
1351  rtb_Out_g[9] = c[2];
1352  c[2] = rtb_Out_g[10];
1353  rtb_Out_g[13] = c[3];
1354  c[3] = rtb_Out_g[14];
1355  circshift_h5(c);
1356  rtb_Out_g[2] = c[0];
1357  c[0] = rtb_Out_g[3];
1358  rtb_Out_g[6] = c[1];
1359  c[1] = rtb_Out_g[7];
1360  rtb_Out_g[10] = c[2];
1361  c[2] = rtb_Out_g[11];
1362  rtb_Out_g[14] = c[3];
1363  c[3] = rtb_Out_g[15];
1364  circshift_h5f(c);
1365  rtb_Out_g[3] = c[0];
1366  rtb_Out_g[7] = c[1];
1367  rtb_Out_g[11] = c[2];
1368  rtb_Out_g[15] = c[3];
1369  memcpy(&rtb_Out_j[0], &rtb_Out_g[0], sizeof(real_T) << 4);
1370  MixColumns(rtb_Out_j, rtb_Out_g);
1371  for (k = 0; k < 4; k++) {
1372    rtb_Out_j_tmp = k << 2;
1373    rtb_Out_j_tmp_0 = (k + 52) << 2;
1374    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1375      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
1376    rtb_Out_j_tmp = (k << 2) + 1;
1377    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1378      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 1]);
1379    rtb_Out_j_tmp = (k << 2) + 2;
1380    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1381      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 2]);
1382    rtb_Out_j_tmp = (k << 2) + 3;
1383    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1384      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 3]);
1385  }
1386
1387  for (k = 0; k < 16; k++) {
1388    rtb_Out_g[k] = b[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
1389  }
1390
1391  c[0] = rtb_Out_g[1];
1392  c[1] = rtb_Out_g[5];
1393  c[2] = rtb_Out_g[9];
1394  c[3] = rtb_Out_g[13];
1395  circshift_h(c);
1396  rtb_Out_g[1] = c[0];
1397  c[0] = rtb_Out_g[2];
1398  rtb_Out_g[5] = c[1];
1399  c[1] = rtb_Out_g[6];
1400  rtb_Out_g[9] = c[2];
1401  c[2] = rtb_Out_g[10];
1402  rtb_Out_g[13] = c[3];
1403  c[3] = rtb_Out_g[14];
1404  circshift_h5(c);
1405  rtb_Out_g[2] = c[0];
1406  c[0] = rtb_Out_g[3];
1407  rtb_Out_g[6] = c[1];
1408  c[1] = rtb_Out_g[7];
1409  rtb_Out_g[10] = c[2];
```

```
  c[2] = rtb_Out_g[11];
  rtb_Out_g[14] = c[3];
  c[3] = rtb_Out_g[15];
  circshift_h5f(c);
  for (k = 0; k < 4; k++) {
    rtb_Out_j_tmp = k << 2;
    rtb_Out_g[rtb_Out_j_tmp + 3] = c[k];
    rtb_Out_j_tmp_0 = (k + 56) << 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
    rtb_Out_g[rtb_Out_j_tmp + 1] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp + 1], w[rtb_Out_j_tmp_0 + 1]);
    rtb_Out_g[rtb_Out_j_tmp + 2] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp + 2], w[rtb_Out_j_tmp_0 + 2]);
    rtb_Out_g[rtb_Out_j_tmp + 3] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp + 3], w[rtb_Out_j_tmp_0 + 3]);
  }

  /* MATLAB Function: '<S4>/MATLAB Function1' */
  for (k = 0; k < 32; k++) {
    rtb_DataTypeConversion2_0[k] = (int8_T)rtb_DataTypeConversion2[k];
  }

  KeyExpansion(rtb_DataTypeConversion2_0, w);
  for (k = 0; k < 4; k++) {
    rtb_Out_j_tmp = k << 2;
    rtb_Out_j_tmp_0 = (k + 56) << 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0]);
    rtb_Out_j_tmp = (k << 2) + 1;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp_0 + 1]);
    rtb_Out_j_tmp_1 = (k << 2) + 2;
    rtb_Out_g[rtb_Out_j_tmp_1] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp_1], w[rtb_Out_j_tmp_0 + 2]);
    rtb_Out_j_tmp_1 = (k << 2) + 3;
    rtb_Out_g[rtb_Out_j_tmp_1] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp_1], w[rtb_Out_j_tmp_0 + 3]);
    c[k] = rtb_Out_g[rtb_Out_j_tmp];
  }

  circshift_h5f(c);
  rtb_Out_g[1] = c[0];
  c[0] = rtb_Out_g[2];
  rtb_Out_g[5] = c[1];
  c[1] = rtb_Out_g[6];
  rtb_Out_g[9] = c[2];
  c[2] = rtb_Out_g[10];
  rtb_Out_g[13] = c[3];
  c[3] = rtb_Out_g[14];
  circshift_m1s(c);
  rtb_Out_g[2] = c[0];
  c[0] = rtb_Out_g[3];
  rtb_Out_g[6] = c[1];
  c[1] = rtb_Out_g[7];
  rtb_Out_g[10] = c[2];
  c[2] = rtb_Out_g[11];
  rtb_Out_g[14] = c[3];
  c[3] = rtb_Out_g[15];
```

```
1469  circshift_h(c);
1470  rtb_Out_g[3] = c[0];
1471  rtb_Out_g[7] = c[1];
1472  rtb_Out_g[11] = c[2];
1473  rtb_Out_g[15] = c[3];
1474  for (k = 0; k < 16; k++) {
1475    rtb_Out_g[k] = b_0[(int32_T)(rtb_Out_g[k] + 1.0) - 1];
1476  }
1477
1478  for (k = 0; k < 4; k++) {
1479    rtb_Out_j_tmp = k << 2;
1480    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1481      (rtb_Out_g[rtb_Out_j_tmp], w[(k + 52) << 2]);
1482    rtb_Out_j_tmp = (k << 2) + 1;
1483    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1484      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 52) << 2) + 1]);
1485    rtb_Out_j_tmp = (k << 2) + 2;
1486    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1487      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 52) << 2) + 2]);
1488    rtb_Out_j_tmp = (k << 2) + 3;
1489    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1490      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 52) << 2) + 3]);
1491  }
1492
1493  ara_func(rtb_Out_g);
1494  for (k = 0; k < 4; k++) {
1495    rtb_Out_j_tmp = k << 2;
1496    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1497      (rtb_Out_g[rtb_Out_j_tmp], w[(k + 48) << 2]);
1498    rtb_Out_j_tmp = (k << 2) + 1;
1499    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1500      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 48) << 2) + 1]);
1501    rtb_Out_j_tmp = (k << 2) + 2;
1502    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1503      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 48) << 2) + 2]);
1504    rtb_Out_j_tmp = (k << 2) + 3;
1505    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1506      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 48) << 2) + 3]);
1507  }
1508
1509  ara_func(rtb_Out_g);
1510  for (k = 0; k < 4; k++) {
1511    rtb_Out_j_tmp = k << 2;
1512    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1513      (rtb_Out_g[rtb_Out_j_tmp], w[(k + 44) << 2]);
1514    rtb_Out_j_tmp = (k << 2) + 1;
1515    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1516      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 44) << 2) + 1]);
1517    rtb_Out_j_tmp = (k << 2) + 2;
1518    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1519      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 44) << 2) + 2]);
1520    rtb_Out_j_tmp = (k << 2) + 3;
1521    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1522      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 44) << 2) + 3]);
1523  }
1524
1525  ara_func(rtb_Out_g);
1526  for (k = 0; k < 4; k++) {
1527    rtb_Out_j_tmp = k << 2;
```

```
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[(k + 40) << 2]);
    rtb_Out_j_tmp = (k << 2) + 1;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 40) << 2) + 1]);
    rtb_Out_j_tmp = (k << 2) + 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 40) << 2) + 2]);
    rtb_Out_j_tmp = (k << 2) + 3;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 40) << 2) + 3]);
  }

  ara_func(rtb_Out_g);
  for (k = 0; k < 4; k++) {
    rtb_Out_j_tmp = k << 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[(k + 36) << 2]);
    rtb_Out_j_tmp = (k << 2) + 1;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 36) << 2) + 1]);
    rtb_Out_j_tmp = (k << 2) + 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 36) << 2) + 2]);
    rtb_Out_j_tmp = (k << 2) + 3;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 36) << 2) + 3]);
  }

  ara_func(rtb_Out_g);
  for (k = 0; k < 4; k++) {
    rtb_Out_j_tmp = k << 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[(k + 32) << 2]);
    rtb_Out_j_tmp = (k << 2) + 1;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 32) << 2) + 1]);
    rtb_Out_j_tmp = (k << 2) + 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 32) << 2) + 2]);
    rtb_Out_j_tmp = (k << 2) + 3;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 32) << 2) + 3]);
  }

  ara_func(rtb_Out_g);
  for (k = 0; k < 4; k++) {
    rtb_Out_j_tmp = k << 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[(k + 28) << 2]);
    rtb_Out_j_tmp = (k << 2) + 1;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 28) << 2) + 1]);
    rtb_Out_j_tmp = (k << 2) + 2;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 28) << 2) + 2]);
    rtb_Out_j_tmp = (k << 2) + 3;
    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 28) << 2) + 3]);
```

```
1587  }
1588
1589  ara_func(rtb_Out_g);
1590  for (k = 0; k < 4; k++) {
1591    rtb_Out_j_tmp = k << 2;
1592    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1593      (rtb_Out_g[rtb_Out_j_tmp], w[(k + 24) << 2]);
1594    rtb_Out_j_tmp = (k << 2) + 1;
1595    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1596      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 24) << 2) + 1]);
1597    rtb_Out_j_tmp = (k << 2) + 2;
1598    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1599      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 24) << 2) + 2]);
1600    rtb_Out_j_tmp = (k << 2) + 3;
1601    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1602      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 24) << 2) + 3]);
1603  }
1604
1605  ara_func(rtb_Out_g);
1606  for (k = 0; k < 4; k++) {
1607    rtb_Out_j_tmp = k << 2;
1608    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1609      (rtb_Out_g[rtb_Out_j_tmp], w[(k + 20) << 2]);
1610    rtb_Out_j_tmp = (k << 2) + 1;
1611    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1612      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 20) << 2) + 1]);
1613    rtb_Out_j_tmp = (k << 2) + 2;
1614    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1615      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 20) << 2) + 2]);
1616    rtb_Out_j_tmp = (k << 2) + 3;
1617    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1618      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 20) << 2) + 3]);
1619  }
1620
1621  ara_func(rtb_Out_g);
1622  for (k = 0; k < 4; k++) {
1623    rtb_Out_j_tmp = k << 2;
1624    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1625      (rtb_Out_g[rtb_Out_j_tmp], w[(k + 16) << 2]);
1626    rtb_Out_j_tmp = (k << 2) + 1;
1627    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1628      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 16) << 2) + 1]);
1629    rtb_Out_j_tmp = (k << 2) + 2;
1630    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1631      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 16) << 2) + 2]);
1632    rtb_Out_j_tmp = (k << 2) + 3;
1633    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1634      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 16) << 2) + 3]);
1635  }
1636
1637  ara_func(rtb_Out_g);
1638  for (k = 0; k < 4; k++) {
1639    rtb_Out_j_tmp = k << 2;
1640    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1641      (rtb_Out_g[rtb_Out_j_tmp], w[(k + 12) << 2]);
1642    rtb_Out_j_tmp = (k << 2) + 1;
1643    rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1644      (rtb_Out_g[rtb_Out_j_tmp], w[((k + 12) << 2) + 1]);
1645    rtb_Out_j_tmp = (k << 2) + 2;
```

```
1646      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1647        (rtb_Out_g[rtb_Out_j_tmp], w[((k + 12) << 2) + 2]);
1648      rtb_Out_j_tmp = (k << 2) + 3;
1649      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1650        (rtb_Out_g[rtb_Out_j_tmp], w[((k + 12) << 2) + 3]);
1651    }
1652
1653    ara_func(rtb_Out_g);
1654    for (k = 0; k < 4; k++) {
1655      rtb_Out_j_tmp = k << 2;
1656      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1657        (rtb_Out_g[rtb_Out_j_tmp], w[(k + 8) << 2]);
1658      rtb_Out_j_tmp = (k << 2) + 1;
1659      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1660        (rtb_Out_g[rtb_Out_j_tmp], w[((k + 8) << 2) + 1]);
1661      rtb_Out_j_tmp = (k << 2) + 2;
1662      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1663        (rtb_Out_g[rtb_Out_j_tmp], w[((k + 8) << 2) + 2]);
1664      rtb_Out_j_tmp = (k << 2) + 3;
1665      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1666        (rtb_Out_g[rtb_Out_j_tmp], w[((k + 8) << 2) + 3]);
1667    }
1668
1669    ara_func(rtb_Out_g);
1670    for (k = 0; k < 4; k++) {
1671      rtb_Out_j_tmp = k << 2;
1672      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1673        (rtb_Out_g[rtb_Out_j_tmp], w[(k + 4) << 2]);
1674      rtb_Out_j_tmp = (k << 2) + 1;
1675      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1676        (rtb_Out_g[rtb_Out_j_tmp], w[((k + 4) << 2) + 1]);
1677      rtb_Out_j_tmp = (k << 2) + 2;
1678      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1679        (rtb_Out_g[rtb_Out_j_tmp], w[((k + 4) << 2) + 2]);
1680      rtb_Out_j_tmp = (k << 2) + 3;
1681      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1682        (rtb_Out_g[rtb_Out_j_tmp], w[((k + 4) << 2) + 3]);
1683    }
1684
1685    ara_func(rtb_Out_g);
1686    for (k = 0; k < 4; k++) {
1687      rtb_Out_j_tmp = k << 2;
1688      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1689        (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp]);
1690      rtb_Out_j_tmp = (k << 2) + 1;
1691      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1692        (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp]);
1693      rtb_Out_j_tmp = (k << 2) + 2;
1694      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1695        (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp]);
1696      rtb_Out_j_tmp = (k << 2) + 3;
1697      rtb_Out_g[rtb_Out_j_tmp] = function_handle_parenReference
1698        (rtb_Out_g[rtb_Out_j_tmp], w[rtb_Out_j_tmp]);
1699    }
1700
1701    /* End of MATLAB Function: '<S4>/MATLAB Function1' */
1702
1703    /* Outport: '<Root>/DecryptedText' incorporates:
1704     *  DataTypeConversion: '<S4>/Data Type Conversion'
```

```
1705    */
1706   rtY.DecryptedText = (uint8_T)rtb_Out_g[15];
1707 }
1708
1709 /* Model initialize function */
1710 void Encryption_initialize(void)
1711 {
1712   /* Registration code */
1713
1714   /* initialize non-finites */
1715   rt_InitInfAndNaN(sizeof(real_T));
1716 }
1717
1718 /*
1719  * File trailer for generated code.
1720  *
1721  * [EOF]
1722  */
```

Listing A.1: Simulink Produced Code for the AES Model

```
1  function Out = Cipher_aes(key,In)
2  key=char(key);
3  Nk=length(key)/4;
4
5  state=reshape(In,4,[]);%reshapes input into state matrix
6  state=AddRoundKey(state,w(:,1:4));%conducts first round
7  % Rounds begin
8
9      state=ara_func2(state);%per standard
10
11     state=AddRoundKey(state,w(:,5:8));%per standard
12
13   state=ara_func2(state);%per standard
14     state=AddRoundKey(state,w(:,9:12));%per standard
15
16      state=ara_func2(state);%per standard
17     state=AddRoundKey(state,w(:,13:16));%per standard
18
19   state=ara_func2(state);%per standard
20     state=AddRoundKey(state,w(:,17:20));%per standard
21
22
23
24     state=ara_func2(state);%per standard
25     state=AddRoundKey(state,w(:,21:24));%per standard
26
27
28
29
30   state=ara_func2(state);%per standard
31     state=AddRoundKey(state,w(:,25:28));%per standard
32
33
34
35       state=ara_func2(state);%per standard
36     state=AddRoundKey(state,w(:,29:32));%per standard
37
38
```

```
39
40      state=ara_func2(state);%per standard
41      state=AddRoundKey(state,w(:,33:36));%per standard
42
43
44
45      state=ara_func2(state);%per standard
46      state=AddRoundKey(state,w(:,37:40));%per standard
47
48
49
50       state=ara_func2(state);%per standard
51      state=AddRoundKey(state,w(:,41:44));%per standard
52
53
54
55     state=ara_func2(state);%per standard
56      state=AddRoundKey(state,w(:,45:48));%per standard
57
58
59
60
61       state=ara_func2(state);%per standard
62      state=AddRoundKey(state,w(:,49:52));%per standard
63
64
65    state=ara_func2(state);%per standard
66      state=AddRoundKey(state,w(:,53:56));%per standard
67
68  state=SubBytes(state);
69  state=ShiftRows(state);
70  state=AddRoundKey(state,w(:,4*(Nk+6)+1:4*(Nk+7)));
71  Out=state(:);%changes output to column vector 1x16 double
72  end
```

Listing A.2: AES Cipher Function

```
1   function Out = Decipher_aes(key,In)
2
3   key=char(key);
4   Nk=length(key)/4;
5
6   w=KeyExpansion(key,Nk);
7   state=reshape(In,4,[]);
8   state=AddRoundKey(state,w(:,4*(Nk+6)+1:4*(Nk+7)));
9
10
11    state=InvShiftRows(state);%per standard
12        state=InvSubBytes(state);%per standard
13      state=AddRoundKey(state,w(:,53:56));%per standard
14
15
16
17
18  state=ara_func(state);
19
20        state=AddRoundKey(state,w(:,49:52));%per standard
21
22
```

```
23
24 state=ara_func(state);
25     state=AddRoundKey(state,w(:,45:48));%per standard
26
27
28 state=ara_func(state);
29         state=AddRoundKey(state,w(:,41:44));%per standard
30
31
32 state=ara_func(state);
33        state=AddRoundKey(state,w(:,37:40));%per standard
34
35
36
37 state=ara_func(state);
38     state=AddRoundKey(state,w(:,33:36));%per standard
39
40
41
42 state=ara_func(state);
43   state=AddRoundKey(state,w(:,29:32));%per standard
44
45
46 state=ara_func(state);
47        state=AddRoundKey(state,w(:,25:28));%per standard
48
49
50
51 state=ara_func(state);
52       state=AddRoundKey(state,w(:,21:24));%per standard
53
54
55
56 state=ara_func(state);
57     state=AddRoundKey(state,w(:,17:20));%per standard
58
59
60
61 state=ara_func(state);
62     state=AddRoundKey(state,w(:,13:16));%per standard
63
64
65 state=ara_func(state);
66       state=AddRoundKey(state,w(:,9:12));%per standard
67
68
69
70 state=ara_func(state);
71     state=AddRoundKey(state,w(:,5:8));%per standard
72
73 state=ara_func(state);
74 state=AddRoundKey(state,w(:,1:4));
75 Out=state(:)';
76
77 end
```

Listing A.3: AES Decipher Function

CURRICULUM VITAE

Name Surname: Fregis SALA
Place and Date of Birth: Tirana, Albania 25.06.1997
E-Mail: fregissala@gmail.com
EDUCATION:

- **B.Sc.:** 2016-2021, Istanbul Technical University, Electrical and Electronics Engineering Faculty, Electronics and Communication Department
- **High School:** 2012-2016, Albanian Qatar College

PROFESSIONAL EXPERIENCE:

- 02.2021-03.2021, Intern, İTÜ Arı Teknokent, Abe Teknoloji
- 07.2020-08.2020, Summer Intern, TÜBİTAK, Integrated Circuit Design and Training Laboratory(TÜTEL)
- 07.2019-08.2019, Summer Intern, İTÜ, VLSI Duran Leblebici Labs

CURRICULUM VITAE

Name Surname: Tolga KARAKAYA
Place and Date of Birth: Salihli, Turkey 08.09.1997
E-Mail: tfkarakaya@gmail.com
EDUCATION:

- **B.Sc.:** 2015-2021, Istanbul Technical University, Electrical and Electronics Engineering Faculty, Electronics and Communication Department
- **High School:** 2011-2015, Sekine Evren Anatolian High School

PROFESSIONAL EXPERIENCE:

- 07.2020-08.2020, Summer Intern, TÜBİTAK, Integrated Circuit Design and Training Laboratory(TÜTEL)
- 07.2019-08.2019, Summer Intern, İTÜ, Medical Device Research Development and Application Laboratory