DESIGN OF A POWER AWARE ASYNCHRONOUS AES CIRCUIT

by

Salim Melih Arslan

B.S., Electronics and Telecommunication Engineering, Yıldız Technical University, 2007

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University

2011

DESIGN OF A POWER AWARE ASYNCHRONOUS AES CIRCUIT

APPROVED BY:

Assist Prof. Faik Başkaya .....
(Thesis Supervisor)

Assist. Prof. Gökay Saldamlı.....
(Thesis Co-supervisor)

Prof. Oğuzhan Çiçekoğlu .....

Assist. Prof. Şenol Mutlu .....

Assist. Prof. Alper Şen .....

DATE OF APPROVAL: 26.09.2011

ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to my supervisor, Assist. Prof. Faik Başkaya, and co-supervisor, Assist. Prof. Gökay Saldamlı, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject.

I also would like to thank other commitee members for their willingness to spend time being examiners of my thesis.

Finally, special thanks goes to my parents, my brother Semih and my dear Bilge since this thesis would not have been possible without their support, motivating talks and patience.

ABSTRACT

DESIGN OF A POWER AWARE ASYNCHRONOUS AES CIRCUIT

As the security and privacy awareness continuously arise, cryptographic services such as data confidentiality, integrity, authentication, and non-repudiation become an essential and indispensable part of digital systems and communication networks. Most of the digital circuits rely on synchronous design approach where a single system clock defines the timing of the communication between digital components. However, with the increasing complexity of digital circuits, it is becoming more and more difficult (and energy inefficient) to provide a global clock signal across the whole circuit synchronously. Hence, asynchronous designs emerge as a low power alternative in designing power aware systems urged in cryptographic engineering. In this thesis, it is managed to realize a power aware asynchronous AES (Advanced Encryption Standard) circuit that consumes much less power than similar synchronous implementations. While Balsa is used as a framework to synthesize asynchronous circuits, all low power measurements are carried on Xilinx FPGA families for a fair comparison with the existing literature.

ÖZET

ASENKRON AES DEVRESİNİN DÜŞÜK GÜÇLÜ TASARIMI

Güvenlik ve gizlilikle ilgili farkındalıkların artmasıyla beraber, kriptografinin ilgi alanına giren veri güvenliği, yetkilendirme, entegrasyon gibi servisler dijital sistemlerin ve iletişim ağlarının en gerekli ve vazgeçilemez bileşenleri haline gelmiştir. Günümüzde birçok dijital devre tek bir sistem saati tarafından devre elemanlarının birbiriyle haberleşme zamanlarını tayin ettiği senkron diye tabir edilen yaklaşımla tasarlanmıştır. Fakat dijital devrelerin gittikçe karmaşıklaşmasının sonucu olarak tek bir sistem saatinin tüm devre üzerinde senkron olarak dağıtılması hem zor olmaktadır hem de enerji tasarrufunu düşürmektedir. Bu yüzden asenkron tasarımlar kriptoloji mühendisliği alanında alternatif düşük güç harcayan tasarımlar olarak ortaya çıkmaktadır. Bu tez kapsamında gelişmis bir şifreleme standardı olarak kabul edilen AES devresinin senkron uygulamalarına nazaran daha düşük güç harcayan tasarımlarının asenkron olarak yapılabileği gösterilmiştir. Tasarlanan asenkron devrenin sentezlenmesinde Balsa platformu kullanılmış ve tüm güç ölçümleri adil bir karşılaştırma olanağı sunması açısından Xilinx firmasının FPGA ailesi için ölçülmüştür.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

AES	Advanced Encryption Standard
CSP	Communicating Sequential Processes
DES	Data Encryption Standard
DI	Delay Insensitive
LUT	Look Up Table
NIST	National Institue of Standards and Technology
QDI	Quasi Delay Insensitive
SAC	Strict Avalance Criteria
SI	Speed Independent
SPN	Substitution Permutation Network
VHDL	Very High Speed Integrated Circuit Hardware Description Language

1. INTRODUCTION

Cryptographic services such as data confidentiality, integrity, authentication, and non-repudiation become an essential and indispensable part of digital systems and communication networks. However, while delivering these services, sophisticated applications of sophisticated mathematics increase the complexity of digital circuits urged in cryptographic engineering; hence system models may become highly complicated that could take years to be realized. These new models are generally more secure but due to their complexity, they normally have low throughputs and higher energy consumption. While this situation is not a major concern for systems with average computing power such as desktop or notebook computers, it is definitely a big problem for constraint devices such as smart cards, RFID tags or most wireless gadgets. The power needs of the encryption unit could be so demanding that some smart card designs freeze all the activities of the device (including the processor and the interrupts) in order not to interrupt the encryption process. Such a solution could never be acceptable in terms of security measures. Our motivation is to address these kinds of problems for constraint devices by designing low power cryptographic primitives that do not have security flaws.

Today, most of the digital circuits are implemented synchronously where a single system clock defines timing of the communication between the digital components. However, with the increasing complexity of digital circuits, it is becoming more and more difficult to provide a global clock signal across the whole circuit synchronously. This causes the clock signal to be faster in order to suppress clock skew problems. As a result the high frequency microprocessors (and computing) once again bring back the tough cooling barrier to the technological development. Even though with the invention of CMOS circuits, it was thought that the power density and accompanying heat cooling problem would never be an obstacle anymore in the late 1960's.

Unfortunately, in the near future, a CMOS like technological leap is quite unlikely to happen; hence the current architectures are focused on the designs that could

decrease the power dissipation within the limits of CMOS. The first outcomes of this research are the low power processors and circuit elements lately launched in the market enabling the technological progress until the next major technological revolution. This vertical growth, providing time for the next heaps, is very likely to continue in the upcoming decades, meaning that the current hardware and software will continue to be redesigned with a low power emphasis.

In this sense, asynchronous design is an emerging technology that is a natural candidate to elaborate this vertical growth. Since there is no clock in an asynchronous circuit, clock distribution problems and more importantly, clock switching activity would be eliminated. Nevertheless, all signals have to be validated all times in asynchronous design which implies that at least the output signals that are seen by the environment must be free from all hazards. To achieve this, it is sometimes necessary to avoid hazards on internal signals as well. This is one of the reasons why the synthesis of asynchronous sequential circuits is difficult.

In this thesis, a power efficient asynchronous AES design is realized on an FPGA. In order to simplify the analysis the simplest S-box architecture that uses the lookup tables for Subbytes realization is picked. Secondly S-Box architecture is modified to be combinational circuit instead of lookup table. In order to have a fair comparison with the existing literature, measurements are carried on Xilinx FPGA families.

Rest of the thesis is organized as follows. In the next chapter, asynchronous circuit design is briefly introduced. Third chapter is dedicated to Balsa framework and syntactic rules of Balsa language. In Chapter 4, AES architecture is described. Additionally this section also includes the related work carried in the literature. Chapter 5 describes the proposed system model including asynchronous handshake graph, simulation, area estimation, synthesis and FPGA implementation and concludes with benchmarking and comparison of the proposed architectures with the existing ones in the literature. Finally Chapter 6 presents the final results with the future opportunities on the field of thesis.

2. ASYNCHRONOUS DESIGN TECHNIQUES

Vendors are revising an old concept -the clockless chip- as they look for new processor approaches to work with the growing number of number of cellular phones, PDAs, and other high performance, battery powered devices [1]. Clockless processors namely self timed or asynchronous, do not have oscillating crystal that provides regular ticking that makes the operation done as in the synchronous processors. Instead of waiting the clock cycle to be completed, asynchronous processors obtain the result of the operation instantly. The following sections describe main concepts of asynchronous design techniques required as a base for the upcoming chapters.

2.1. Comparison Of Synchronous And Asynchronous Circuits

As a never-ending battle, comparison between asynchronous and synchronous circuits still attracks a great interest. This section briefly asserts why a circuit is designed to be synchronous or asynchronous.

Most digital circuits designed and fabricated today are "synchronous" and in essence, they are based on two fundamental assumptions that greatly simplify their design: (i) all signals are binary, and (ii) all components share a common and discrete notion of time, as defined by a clock signal distributed throughout the circuit [2]. Asynchronous circuits, on the other hand, do not share common and discrete time. Instead, global clock is replaced by handshake components for the purpose of synchronization, communication and sequencing of operations. This synchronization is handled by means of localized synchronization protocol between the communicating blocks or neighboring components. The following items are the main advantages investigated in depth in [5]:

- Low power consumption
- High operating speed
- Less emission of electro-magnetic noise

- Robustness against variations in supply voltage, temperature, and fabrication process parameters
- Better composability and modularity
- No clock distribution and clock skew problems

In synchronous designs, as the data moves on every clock edge, voltage spikes occur. In asynchronous case, on the other hand, data does not all move at the same time, which spreads out current flow, thereby minimizing the strength and frequency of spikes and emitting less error-correction logic [1]. This really reduces the power consumption such that only active parts run in asynchronous chips however entire chip runs in synchronous chips as seen in Figure 2.1. Clockless chips offer an advantage over their

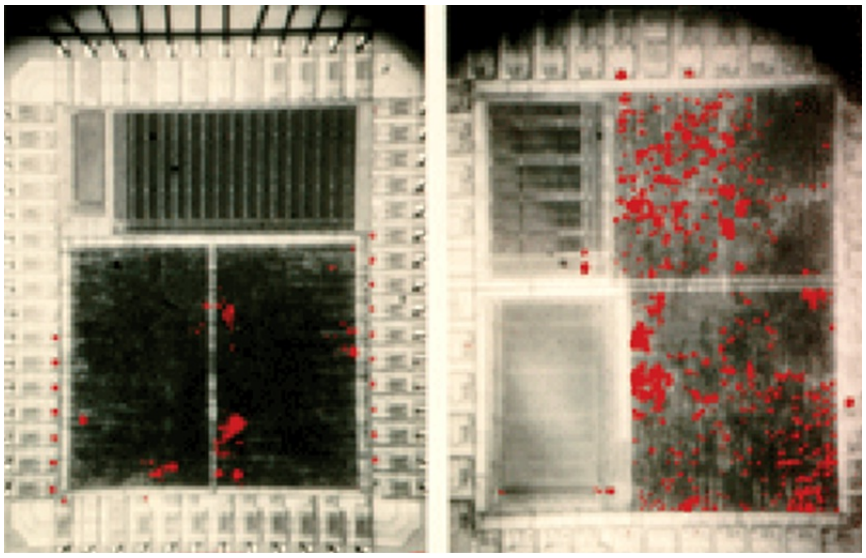

Figure 2.1. A View of Asycnhronous and Synchronous Processors.

synchronous counterparts because they efficiently use cycle times shown in Figure 2.2. Synchronous processors must make sure they can complete each part of a computation in one clock tick. Thus, in addition to running their logic, the chips must add cycle time to compensate for how much longer it takes to run some operations than to run average operations (worst case-average case) variations in clock operations (jitter and skew), and manufacturing and environmental irregularities [1].

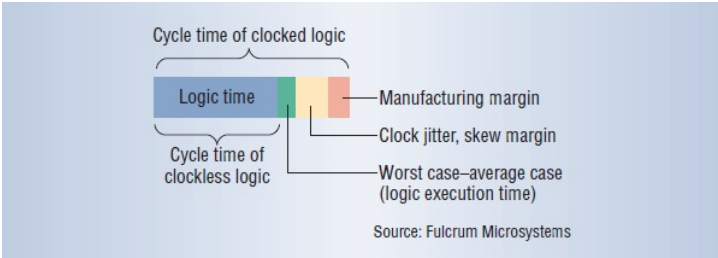

Figure 2.2. Cycle Time of Synchronous and Asynchronous Logics [1].

2.2. Handshake Signalling Protocols

This section covers various handshake protocols such as bundled data and dual rail protocols.

2.2.1. Bundled Data Protocol

Bundled Data is a term used to express a case that tha data signals are represented by Boolean levels to encode information. Besides there exist seperate request signal and acknowledge signal which are bundled with data signal as in Figure 2.3a. The term 4-phase means as is evident from it's name, handshaking is done in four step: (i) Data is sent and request signal set as high, (ii) As soon as receiver obtains the data, acknowledge signal is set as high, (iii) The sender takes request signal low which means data is no longer guarenteed to be valid, (iv) The receiver responds this by setting acknowledge signal as low. After forth phase, handshaking is completed and both handshake pairs are ready for next communication cycle. The 4-phase bundled data protocol is familiar to most digital designers, but it has a disadvantage in the superfluous return-to-zero transitions that cost unnecessary time and energy [2]. Two-phase one, however, does not suffer from this. Each signal transitions on the wires represent the information on request and acknowledge wires. So it does not matter if the transition is from zero to one or vice versa. Information is simply encoded as signal event.

Normally 2-phase bundled protocol makes circuits faster than that of 4-phase bundled protocol. However answer to which protocol is best changes as the complexity

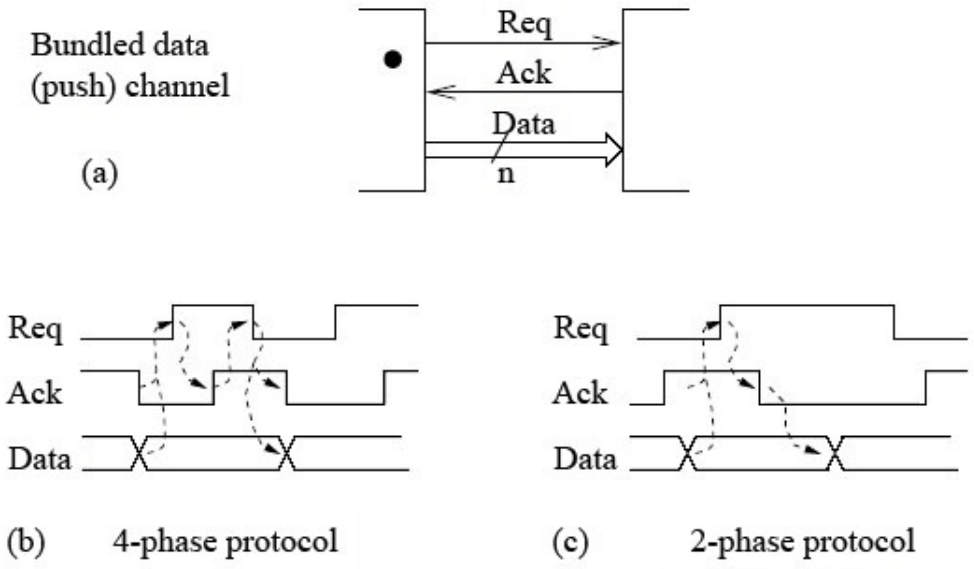

Figure 2.3. (a) A Bundled Data Channel. (b) 4-Phase Bundled-data Protocol. (c) 2-Phase Bundled-data Protocol [2].

and target of the circuit change. The term bundled data does not emphasize on timing aspect of communication. To be more explicatory,return-to-zero or level signalling are also used in some texts instead of 4-phase bundled data protocol, and non-return-to-zero instead of 2-phase bundled data protocol.

Two handshake components, one is sender and other is receiver, constitute a handshake pair. Sender starts the data communication as an active part. The wire where the communication takes place and which is initiated by sender is called push channel. Complementary part of the pair at the end absorbs the data like a receiver this is why the channel is also called pull channel. When there is no data transfer in a channel, it can be used for synchronisation purpose. If a channel transmits data bi-directionally request and acknowledge signals indicate exchanged validity. Each bundled protocols depend on delay matching to preserve signal events at the sender's end and receiver's end. On a push channel, if the request signal is set high data is no longer valid. This rule should be followed at the receiver's end and should be taken into account while implementing such circuits. Possible solutions are [2]:

- To control the placement and routing of the wires, possibly by routing all signals

in a channel as a bundle. This is trivial in a tile-based datapath structure

- To have a safety margin at the sender's end
- To insert and/or resize buffers after layout (much as is done in today's synthesis and layout CAD tools)

2.2.2. Dual Rail Protocols

2.2.2.1. 4-Phase Dual Rail Protocol. As illustrated in Figure 2.4, the 4-phase dual rail protocol represents the data signal by making use of two wires, each carries one bit of information. Usually request wire is represented by *d*. For logic 1 (true) *d.t* wire is used and *d.f* wire is used for logic 0 (false). Viewed together the {x.f,x.t} wire pair

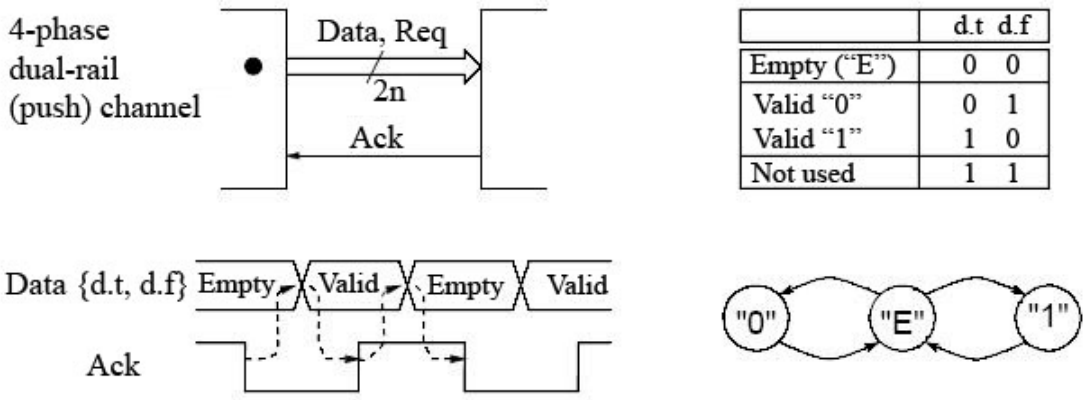

Figure 2.4. Four-Phase Dual Rail Protocol [2].

is a codeword; {x.f,x.t} = {1,0} and {x.f,x.t} = {0,1} represent valid data (logic 0 and logic 1 respectively) and {x.f,x.t} = {0,0} represents no data (or spacer or empty value or NULL) the codeword {x.f,x.t} = {1,1} is not used, and a transition from one valid codeword to another valid codeword is not allowed [2]. Handshaking of N-bit channel is shown if Figure 2.5. First receiver obtains the empty codeword, then intermediate codewords and lastly a valid codeword. Having received and acknowledged the codeword the receiver waits until next empty codeword and responds it by taking acknowledge low.

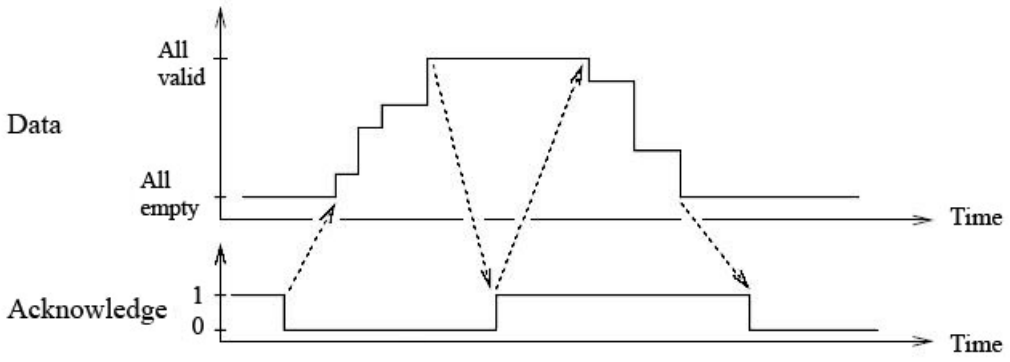

Figure 2.5. Illustration of Handshaking on a 4-phase Dual Rail Channel.

2.2.2.2. 2-Phase Dual Rail Protocol. Just as 4-phase dual rail protocol use 2 wires like $\{d.f, d.t\}$ per bit, so does 2-phase dual rail. The difference is that each signal transition indicates an information. For N-bit channel case, a new codeword is sensed by signal transition of one exact wire in each of the N wire pairs as illustrated in Figure 2.6.

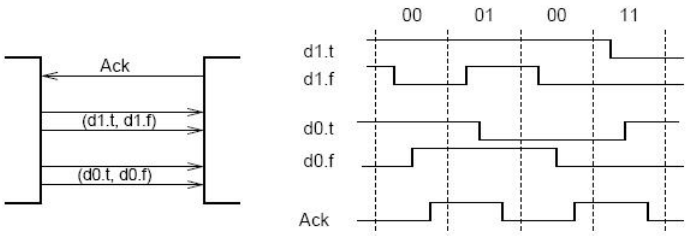

Figure 2.6. Illustration of Handshaking on a 2-phase Dual Rail Channel [2].

2.3. The Muller C Element

Main function of clock in synchronous circuits is to make clear in which time interval signals are stable, valid and meaningful. As the clocking frequency increases, hazards and glitching effects also raise. This brings the need for supressing hazards and race conditions as much as possible. The source of hazards are the signal transitions which are not indicated and acknowledged in other signal transitions. As obvious in Figure 2.7 Muller C element is used for preventing hazards. The output is if both inputs are 0 and output is 1 if both inputs are 1. In other cases, the output remains as is. This makes designer to ensure both inputs are now at 1 when the signal transition

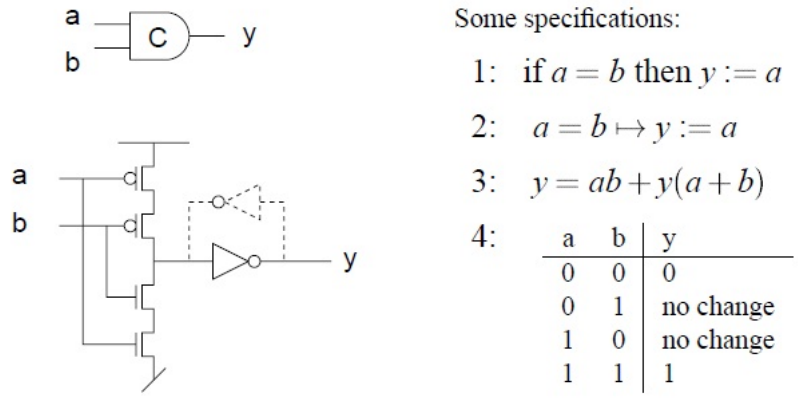

Figure 2.7. The Muller C Element [2].

happens from 0 to 1. Similary designer seeing the output change from 1 to 0 may conclude that both inputs are now 0. Consequently, the Muller C-element has very critical role in asynchronous circuits that is why it is widely used in many designs.

2.4. Delay Models

There are various delay models to identify the effects of delays on a system. One of these models is fixed delay model where delays are expected to have fixed values. Other model is bounded delay model where delays may have any value in a certain interval. Last one is unbounded delay model which is more common than others. This model has various timing assumptions as follows;

- *Delay Insensitive Circuit*: DI Circuits are the one in which circuit elements like gates and wires are completely insensitive to delays. However, very limited number of circuits conform to delay insensitivity. For this reason, delay-insensitivity is most often applied to larger, more coarsely grained units, constructed using other timing regimes in order to make them easier to compose [6].

- *Quasi Delay Insensitive Circuit*: If the difference between signal propagation delays in the branches of a set of interconnecting wires is negligible compared to the delays of the gates connected to these branches then the wires are said to form an isochronic fork [7]. This augmented version of DI is called Quasi Delay

Insensitive.

- *Speed Independent Circuit*: A circuit is called SI circuit if it operates independently from delays in any circuit element. In general, SI circuits and QDI circuits are regarded as equivalent from the modelling point of view [6].

2.5. Difficulties In Asynchronous Design

2.5.1. Metastability and Arbitration

In asynchronous circuits, a mutually exclusive components with two inputs and two outputs are usually preferred in order to enable *arbitration* between two contending inputs. To overcome arbitration a mutex element is usually chosen. Key property of a mutex is that at most one output is active at any time interval. A problem, however, occurs in case both inputs are activated at the same time, or very small time window. The mutex internal signals hover for an unbounded amount of time before reaching a stable state and selecting one of the outputs [6]. This problem is called metastability and non-determinism and race condition on the input signals is called arbitration.

2.5.2. Deadlocks

A deadlock is a situation where in two or more competing actions are each waiting for the other to finish, and thus neither ever does. Very common deadlock scenario is as follows [2];

(i) A (non-sequential) data transfer needs access to a particular RAM block.
(ii) This is prevented because an instruction fetch is already using the RAM array.
(iii) The instruction fetch cannot complete because the instruction decoder is still busy.
(iv) The processor pipeline is full and is blocked by the data fetch.
(v) Deadlock

Deadlock issues are very common in the field of asynchronous design. Without implicit global clock control, the control logic in an asynchronous design is more complex than in a synchronous equivalent design since each module of the design requires hardware to perform synchronisation, to wait for data, and to trigger other modules when it has produced its data [6]. Consequently explicit communicatations between modules make asynchronous design open to deadlocks by its nature.

3. BALSA FRAMEWORK AND PROGRAMMING LANGUAGE

Countless number of simulation tools have been used in the field of synchronous design. These tools are mostly advanced and commercial tools that are capable of synthesizing complete synchronous circuits. However, although synchronous languages and tools can and indeed have been used for asynchronous hardware too, their application in that context is proving awkward and inefficient [8]. For the case of asynchronous design, lack of sufficiently advanced commercial and non-commercial tools make research to develop their own tools. Some of these tools that have been used in asynchronous logic design are I-Nets [9], Petri-Nets [10], Signal Transition Graphs [11], State Transition Diagrams [12], and CCS [13]. Communicating Sequential Processes (CSP) [14], in particular, the concurrent process algebra developed by Tony Hoare for the specification of parallel systems, has been extensively advocated as a suitable means for describing asynchronous behaviour and several asynchronous modelling approaches and systems have been developed which use CSP-based notations [15]. However none of these tools are capable of synthesizing asynchronous circuits. One other tool which is able to both simulate and synthesize large asynchronous circuits is Tangram produced by Philips Research. But it is no longer available [16]. The tool set choosen for this thesis is Balsa which is non-commercial and created by Advanced Processor Technologies group in University of Manchester. It is not aimed at high performance or minimal area solutions, rather it is aimed at rapid implementation with optimizations being made possible at the language level by virtue of its transparent compilation technique [8].

3.1. Balsa Framework

Balsa is one of various asynchronous circuit design tool created by Advanced Processor Technologies group in University of Manchester [8]. The Balsa System is a Handshake Circuit based, macromodule synthesis tool-set. Design descriptions are

written in the proprietary language Balsa and synthesized into networks of handshake components in a similar manner to the Philips Tangram compiler [16]. Balsa is not only the name of synthesizer but also the name of asynchronous HDL.

3.1.1. Basic Concepts

Balsa compiler generates a circuit such that it is like a collection of communicating network of various handshake components. Channels are the links between components where the handshaking happens. Channels may have datapaths associated with them (in which case a handshake involves the transfer of data), or may be purely control (in which case the handshake acts as a synchronisation or rendez-vous point) [3]. Each channel conducts data or signal from one handshake component called active port to another handshake component called passive port. By sending request signal, active port initiates a communication. In return for request signal, passive port responds by acknowledge signal. Communicating via handshake channel is also called

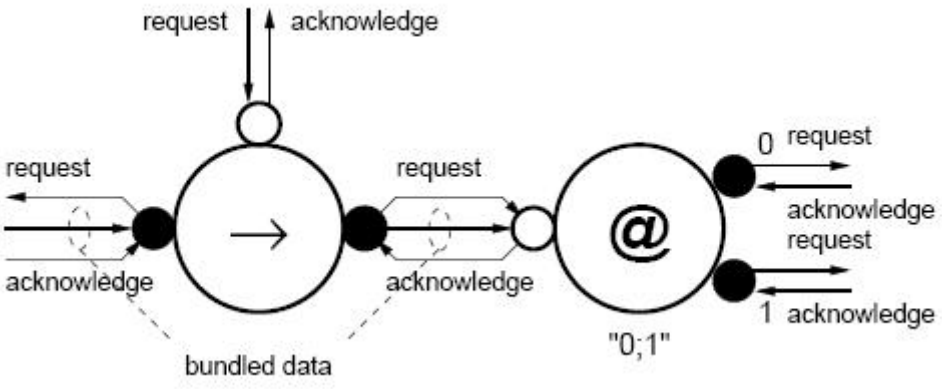

Figure 3.1. Two Connected Handshake Components [3].

as push&pull communication, where the direction of data flow and the direction of the request differentiates between the two types of communication. Data flows from passive port to active port in a pull channel. As illustrated in Figure 3.1 demands a transfer which in turn active port validates the data by acknowledge signal.

3.1.2. Balsa Tool Set And Design Flow

As mentioned in the previous section Balsa is not only a programming language but also a framework capable of simulating and synthesizing asynchronous circuits. This framework is a set of various some of which are listed as follows;

- *balsa-c* : Balsa-c is used to compile source codes written in Balsa language. The compiler generates an intermediate languate *breeze*
- *balsa-netlist* : This tool outputs a netlist depending on targeted technology. To generate netlist Breeze description obtained by compiling source file is used.
- *breeze2ps* : As is evident from it's name, this tool produces post script files of handshaking graph.
- *breeze-cost* : Breeze-cost is a kind of cost estimator.
- *balsa-mgr* : Balsa-mgr stands for Balsa manager which is the graphical front-end used as project navigator.
- *breeze-sim* : This tool simulates the Balsa descriptions at the handshake component level.

Figure 3.2 illustrates an overview of the Balsa design flow. As seen in the figure Balsa design flow starts with the compiling source file using *balsa-c* which in turn Breeze description is generated. Having generated Breeze file, design is ready to be simulated by *breeze-sim*. This simulator allows source level debugging, visualisation of the channel activity at the handshake circuit level as well as producing conventional waveform traces that can be viewed using the waveform viewer gtkwave [3]. Optionally, balsa source files have .balsa file extension. However this does not have special meaning or impact on compilation system.

3.1.3. Using Balsa-mgr

Balsa-mgr is project manager environment which acts as a front-end to the Balsa commands such as balsa-c and breeze2ps [3]. By running *balsa-mgr* command, the project manager environment shows up. Using pull-down menu "Project⇒New" is

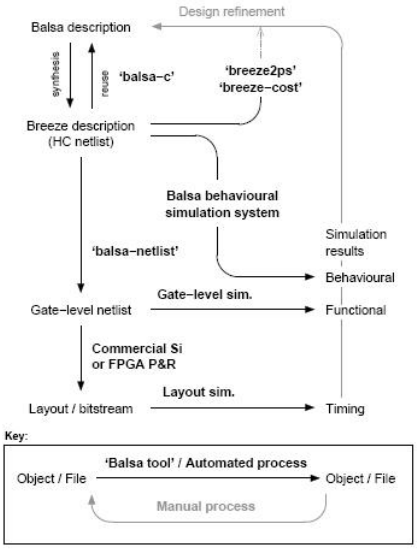

Figure 3.2. Balsa Design Flow [3].

selected to open a new project. Only one project is allowed per directory but each

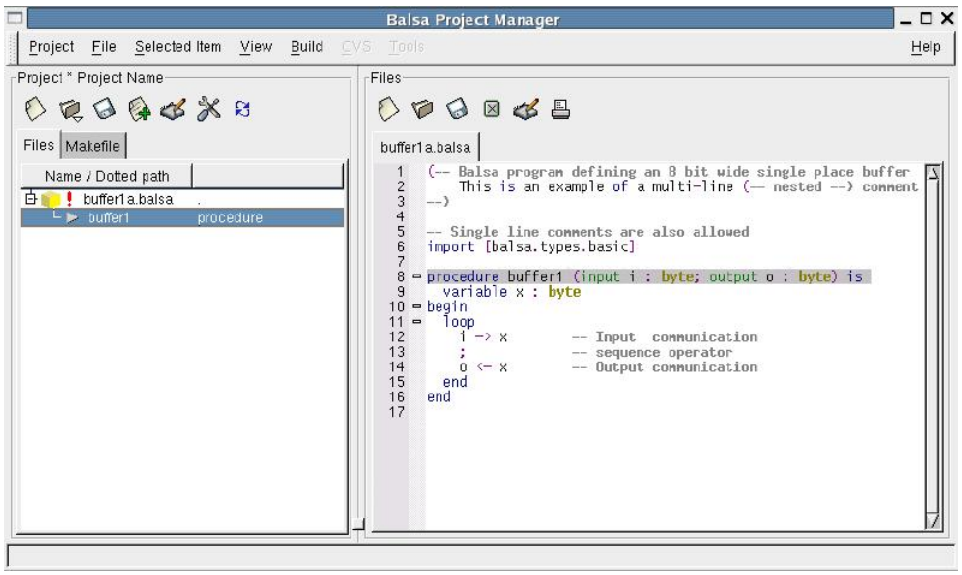

Figure 3.3. Balsa Project Navigator [3].

project may have several compilation targets [3]. To compile a circuit middle click can be used as a short cut or click on the Makefile tab on the left-hand pane. After the new menu appeared, click on the compile button. A save project box appears when there is unsaved changes. After clicking on the compile button another box shows up which is an execution window displaying steps of compilation process. While compiling the source file Balsa-mgr searches for the dependencies in the source files in the project,

generates a Makefiles and rules to enable invoking further balsa commands. If the initial Balsa description is syntactically incorrect in such a way as to make impossible the determination of dependencies, the Makefile will not be correctly generated [3]. By clicking on the "View" button opposite the label .ps file, the source file is compiled again in order to generate a post script file. By clicking on the "Run" button opposite

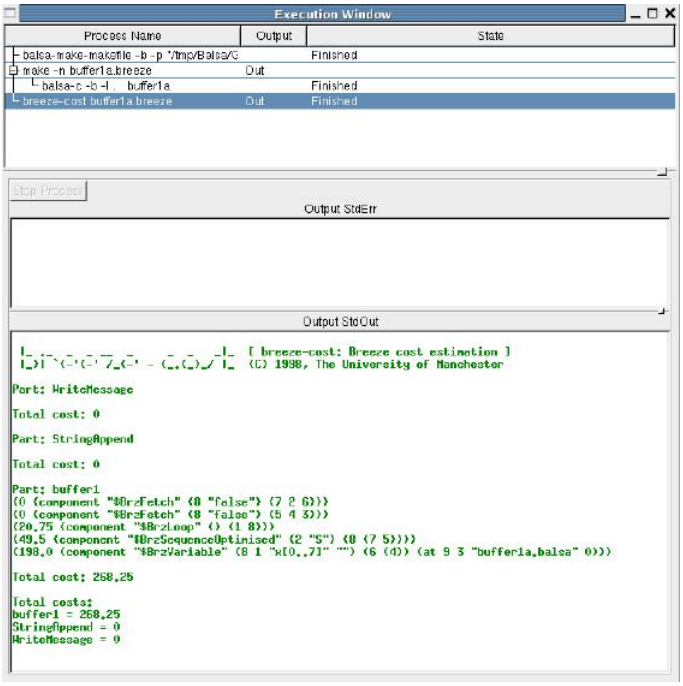

Figure 3.4. Balsa Cost Estimator [3].

the label another window appear as seen in Figure fig:balsacost. Behind this scene, area cost of the circuit is calculated and displayed in the execution window. The result is only to give an idea. Nevertheless, the cost figure is useful for gaining quick feedback on how changing the description of a circuit affects its size [3]. All errors and outputs can be displayed seen in "Output StdErr" and "Output StdOut" tabs respectively and can be saved to a file by right clicking.

3.1.4. Simulation

There are three ways to simulate balsa source files: default test harness, balsa test harness and custom LARD test harness. The default test harness exercises the target Balsa block by repeatedly handshaking on all external channels; input data channels receive auser defined value on each handshake, although it is possible to associate an

input channel with a data file [3]. Balsa test harness is rather complicated and capable of specifying more test sequences. LARD is no longer supported by Balsa. Simulation results in Balsa are displayed either in text format or visual format. Figure 3.5

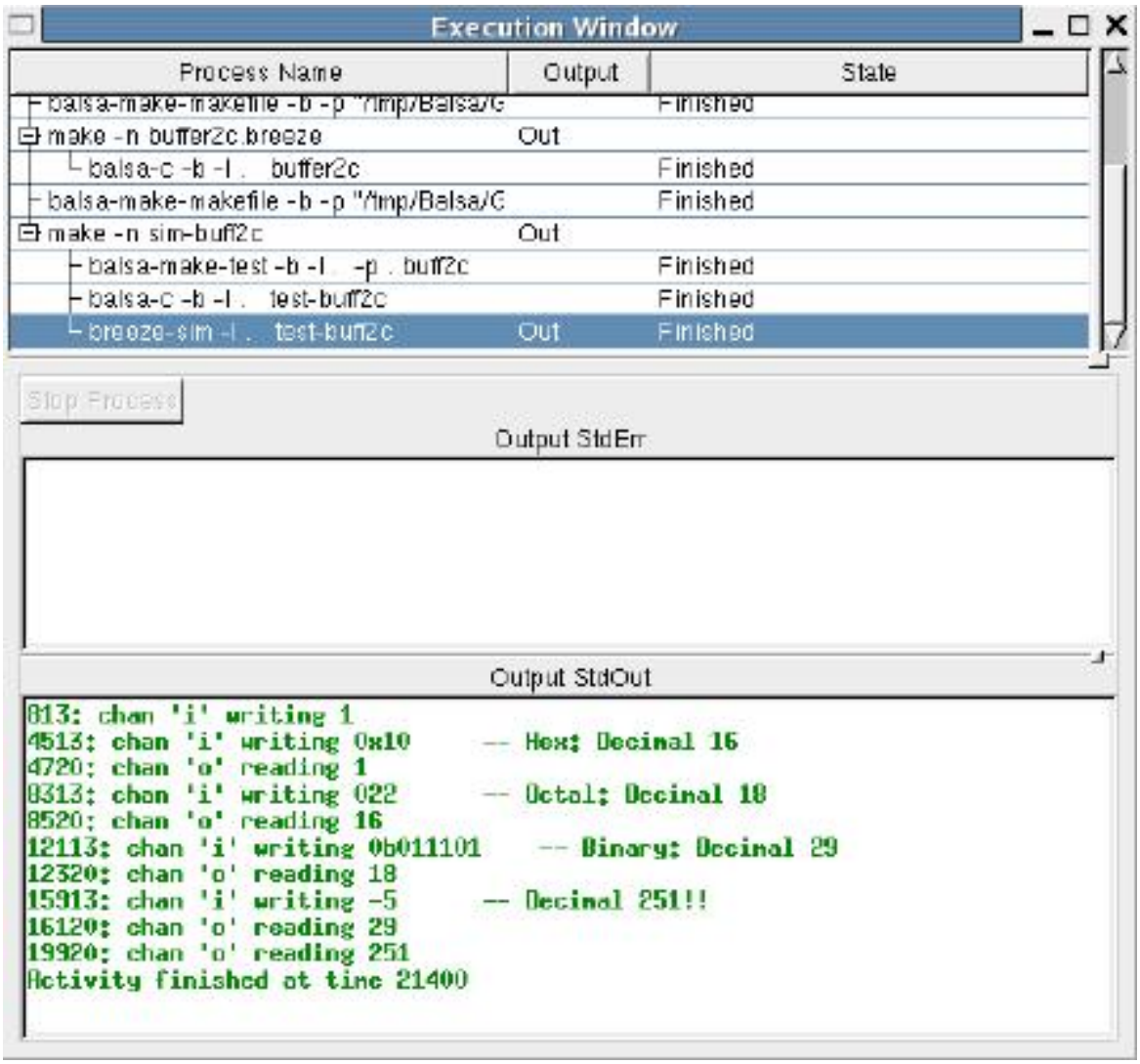

Figure 3.5. The Output of Text Only Simulation [3].

illustrates the output given by text-only simulation. Clicking on the "Run" button in the Tests section of Makefile tab, text only simulation starts. The numbers reported on the left hand side of each channel activity are simulation times - either the time at which data is presented at an input channel from the external environment or the time at which data is presented on an output channel to the external environment [3]. Second way to display simulation results in balsa is grapchical simulation. Controller for the graphical simulation is seen in Figure 3.6. Results may be presented in waveform viewer or in handshake circuit graph. GTKWave is the tool embedded in to balsa as

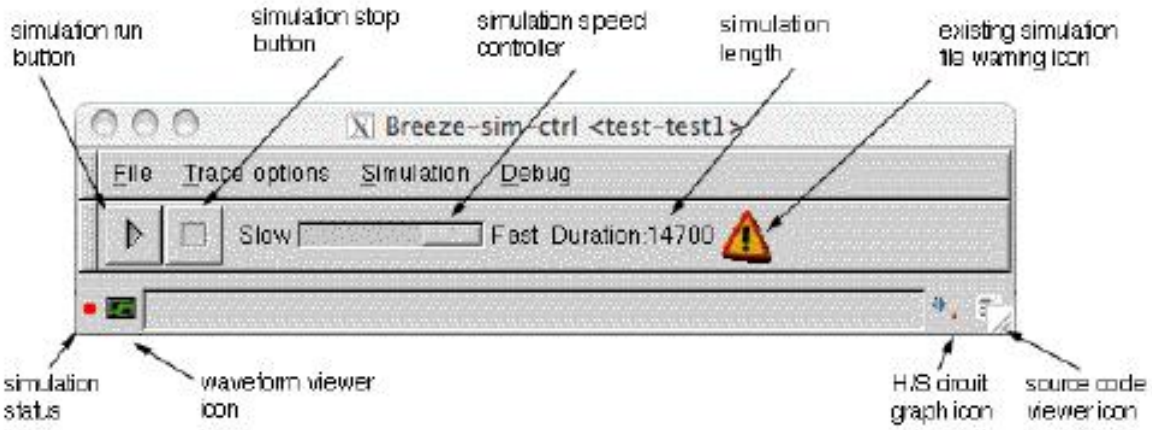

Figure 3.6. Graphical Simulator Controller [3].

a channel viewer or waveform viewer. By clicking the simulation run button in Figure 3.6 GTKWave pane appears. All channels are listed in the right-hand side pane as shown in Figure 3.7. Request signals are marked as red and acknowledge signals are marked as green. Values on the data bearing channels are displayed under the request/acknowledge channels. One other tool inside balsa-mgr is handshake circuit

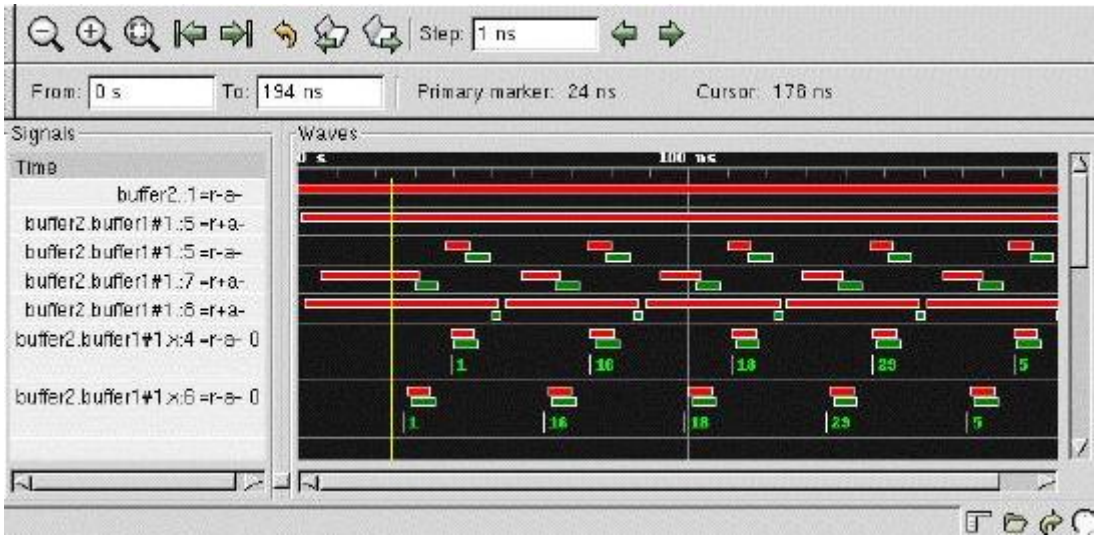

Figure 3.7. Channel Viewer Window [3].

animator as seen in Figure 3.8. With the aid of this feature, compiled handshake circuits can be represented as graphes so as to animate channel activities taking place on the various channels as the simulation runs. Animation can be started by clicking on the "Animate" button. Also speed of the animation can be adjusted by slider control right to the Animate button. To select a channel, click on it. Then a list of actions

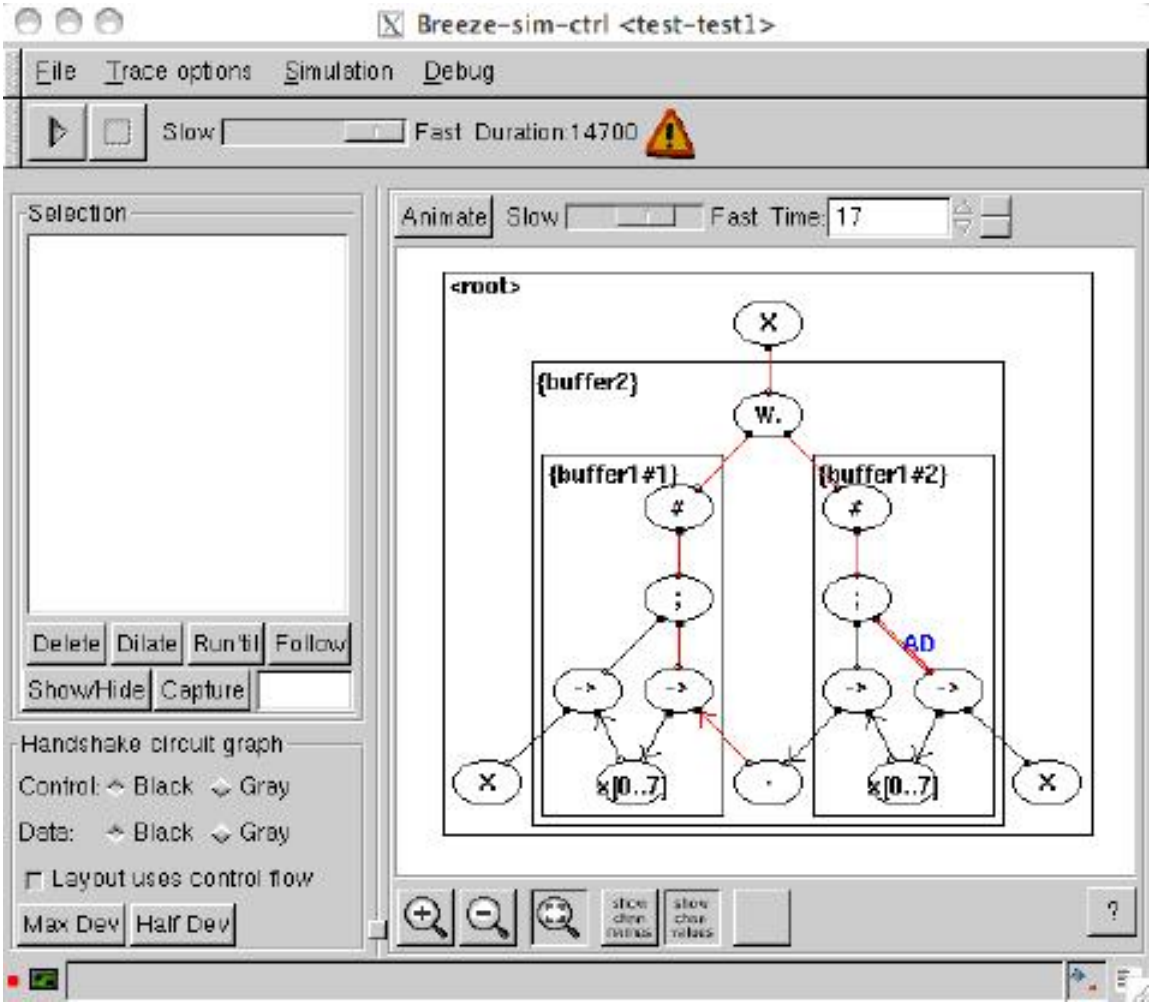

Figure 3.8. Channel Tree and Handshake Animation [3].

are available on the left-hand pane to observe on selected channel [3];

- *Delete*: Unselects the channel.
- *Dilate*: Expands the selection to the surrounding channels.
- *Run'til*: Runs the simulation until some activity appears on the selected channel(s).
- *Follow*: Runs until some activity on the channel and then looks for activity on the surrounding channels and expand the selection to those newly activated surrounding channels.

3.2. Balsa Programming Language

Previous section covers the detailed explanation of Balsa framework and supportive tools inside the framework. As mentioned, Balsa is not only a framework synthesizing clockless hardware sysyems but also a programming language for describing such systems. This section gives detailed syntactic structure of balsa programming language.

3.2.1. Data Types

Data types in balsa depends on bit vectors that is why results of any expression should be in the range of relevant bit vector representation.

3.2.1.1. Numeric Types. Numeric types can be named by aliases of the same range. Type decleration can be made as in Figure 3.9. Some predefined types can be found

```
type word is 16 bits
```

Figure 3.9. Type Decleration.

in `<BalsaInstallDir>/share/balsa/types/basic.balsa`. Byte, nibble, boolean and cardinals as well as the constants true and false are also included in

that directory. Users may contribute to this file if they need.

3.2.1.2. Enumerated Types. Enumeration is a set of named incrementing values. It starts at zero and increments by one from left to right.

```
type Colour is enumeration
Black, Brown, Red, Orange, Yellow, Green, Blue, Violet Purple=Violet, Grey,
Gray=Grey, White
end
```

Figure 3.10. Enumaration Decleration.

The Value of Violet is 7, as same as Purple.

In this case, 2 bits are enough to represent 3 possible values. The over keyword

```
type SillyExample is enumeration
e1=1, e2
over 4 bits
```

Figure 3.11. Sample Enumeration Decleration.

guarantees that enumerated type is exactly 4 bits.

3.2.1.3. Record Types. Records are very similar to "struct" data structure in conventional programming languages. However records are bitwise set of elements of various data types.

Resistor structure has 4 elements such that 3 of them are of type Colour and the last variable, Tolerance is of type ToleranceColour. However in this example Colour and ToleranceColours types are assumed to be declared previously. A record variable can be initialized by listing its elements within braces.

```
type Resistor is record
FirstBand, SecondBand, Multiplier : Colour
Tolerance : ToleranceColour
end
```

Figure 3.12. Record Decleration.

```
RK47 := {Yellow, Violet, Red, Gold}
```

Figure 3.13. Initialization of Record Variable.

3.2.1.4. Array Types. Array is a kind of data structure used to arrange same-typed values in numerically indexed format. Type decleration is as in Figure 3.14.

```
type RegBank_t : array 0..7 of byte
```

Figure 3.14. Array Decleration.

Following expression declares a new type which is an array composed of 8 elements indexed from 0 to 7, being of type byte. In balsa anonymous array types can also be defined as in Figure 3.15.

Array operations like array concatenation, array slicing are literally easy in balsa syntax as seen in Figure 3.16.

Array z4 is constructed by using list constructor. Array concatenation as done to construct z6 is made by means of "@". Array slicing on the other hand is exemplified by using square brackets. In z2 case, third and forth elements of newly concatenated array are extracted. An example for combination of array concatenation and array slicing can be seen in Figure 3.17. The result is also an array like {c,d}.

```
variable RegBank : array 0..7 of byte
```

Figure 3.15. Anonymous Array Decleration.

```
variable a, b, c, d, e ,f: byte
variable z2 : array 2 of byte
variable z4 : array 4 of byte
variable z6 : array 6 of byte
z4:= {a,b,c,d}
z6:= z4 @ {e, f}
z2:= (z4 @ {e, f}) [3..4]
```

Figure 3.16. Array Operations.

```
z2:= (({a, b, c, d} @ {e, f}) [1..4])[1..2]
```

Figure 3.17. Array Concatenation and Slicing.

3.2.1.5. Constants. Constants are the values which can not be reassociated with another value. Sting constants are not allowed in Balsa. Some examples can be seen in Figure 3.18. Other complex data structures that are arrays and records can also be constant as in Figure 3.19.

```
constant minx = 5
constant maxx = minx + 10
constant hue = Red : Colour
constant colour = Colour'Green
```

Figure 3.18. String Constants.

```
constant InitArray = {1, 2, 3, 4} : MyArrayType
constant R4K7 = {Yellow, Violet, Red, Gold} : Resistor
```

Figure 3.19. Constant Structures.

Another syntactic version of examples above are shown in Figure 3.20.

```
constant InitArray = MyArrayType {1, 2, 3, 4}
constant R4K7 = Resistor {Yellow, Violet, Red, Gold}
```

Figure 3.20. Example for Constant Structures.

3.2.2. Control Flow and Commands

Reserved words in Balsa are as follows; `active, also, and, arbitrate, array, as, begin, bits, case, channel, constant, continue, if`

, import, else, end, enumeration, for function, halt, in, input, parameter, passive, is, let, local, log, of, output, or, not, loop, new, multicast, over, print, procedure, pull, val, record, select, shared, signed, sizeof, sync, then, type, variable, xor, while, push. Some of these reserved words are explained in the following paragraps. "Sync" command hang off any handshake event on a named channel. Circuit never takes actions until handshaking is completed.
Channel assignment can be done in four different ways.

$< channel\_out > \leftarrow < expression >$

$< channel\_in > \rightarrow < variable >$

$< channel\_in > \rightarrow < channel\_out >$

$< channel\_in > \rightarrow then < command >$ end

Figure 3.21. Channel Assignment.

In the first assignment, result of the expression is directly output to the channel. Second example shows how to transfer data from a channel to a variable. Third, assignment is done from input channel to output. In the last way of assignment guarantess that data on the input channel remains valid as soon as the commands in the command block is processed. This makes data on the input channel enable to be read more than once and assigned to different channels.
Variable assignment is done like: `<variable> := <expression>`. Value of the expression is directly written in to variable. However, not to obtain compile error both types should match. Sequence Operator ";" has no syntactic importance. It simply indicates sequentiality. If semicolon is placed after the last statement in a block following error message is obtained as in Figure 3.22.

Parallel composition means that commands composed by "||" operates concurrently and independently. Beware of inadvertently introducing dependencies between the two commands so that neither can proceed until the other has completed [3]. Looping

```
expected one of tokens 'ident [ { sync local begin continue halt loop while if case
for select arbitrate print
```

Figure 3.22. Error Message.

constructs means execute a specific code block infinite times by using loop command. One way to construct loops in Balsa is using loop while command as in Figure 3.23.

```
loop while x < 10 then
x := (x+1 as byte)
end
```

Figure 3.23. Loop Construction.

Multiple guards are feasible in balsa as in Figure 3.2.2.

```
loop while
x < 10 then x := (x + 1 as byte)
| x >= 10 then x := 0
end
```

Figure 3.24. Multiple Guards.

If “also” keyword is used in the while construct, it is guaranteed that the line beginning with “also” is executed no matter which guard is satisfied. An example for “also” command can be seen in Figure 3.25.

More simplified version of loop construct is quiet similar to do...while loops in conventional programming languages;

```
loop while
x < 10 then x := (x + 1 as byte)
|x >= 10 then x := 0
also print “Value of x is ”, x
end
```

Figure 3.25. Also Command.

```
loop
print “value of x is: ”, x;
x := (x + 1 as 4 bits)
while x <= 10
end
```

Figure 3.26. Simplified While Loop.

Conditional execution is achieved by using “if” and “case” constructs. Evaluation of expressions is done at run-time. Syntax is as in Figure 3.27. The else clause is not mandatory. The case statement is a multi-way decision maker that tests whether an expression matches one or more possible values [3]. Case statement is works similarly as in the conventional programming languages. Differently, value range can be defined in the guard expression as Figure 3.28.

```
if condition1 then command
| condition2 then command
| condition3 then command
else CmdD
end
```

Figure 3.27. If Statement.

```
case x+y of
1 .. 4 then o ← x
| 5 .. 10 then o ← y
else o ← z
end
```

Figure 3.28. Case Statement.

For loops can be used to automate case statements as in Figure 3.29.

```
case s of
for j in 1 .. 3 then
o[j] ← i
| 0 then
print "Handling port 0 specially"
o[0] ← i-1
end
```

Figure 3.29. Automated Case Statement.

The case matches in the for loop can be any general expressions resolvable at compile time and only one for iteration variable is allowed per guard and the case matches must be disjoint from one another [3]. As used in the code above, balsa has for loop structure. It is similar to that in VHDL rather than conventional programming languages. It simply generates multiple structures. At this point for loop and while loop constructs are totally different from the view point of synthesized asynchronous circuit. For this reason inappropriate use of for command may cause undesired hardwares to be synthesized.

3.2.3. Binary And Unary Operators

In Table 3.1 [3], binary and unary operators available in Balsa are listed. Definitions can be found in the forth column of Table 3.1.

Table 3.1. Binary and Unary Operators in Balsa [3].

Symbol	**Operation**	**Valid Types**	**Notes**
.	record indexing	record	
#	smash	any	takes value from any type and reduces it to an array of bits
[]	array indexing	array	non-const index possible, can generate lots of hardware
∧	exponentiation	numeric	only constants
not, log, -(unary)	unary operators	numeric	log only works on constants, returns the ceiling: e.g log 15 returns 4 -returns a result 1 bit wider than the argument
*, /, %	multiply, divide, remainder	numeric	only applicable to constants
+, -	add, subtract	numeric	results are one or 2 bits longer than the largest argument
@	concatenation	array	
<, >, <=, >=	inequalities	numeric enums	
=, /=	equals, not equals	all	comparison is by sign extended value for signed numeric types
and	bitwise and	numeric	Balsa uses type 1 bits for if/while guards so bitwise and logical operators are the same
or, xor	bitwise or	numeric	

4. ADVANCED ENCRYPTION STANDARD

4.1. Basic Introduction To Cryptography

A cryptosystem consists of plain text, encryption algorithm, cipher text and key. Encryption algorithms are the most of important parts of these systems. Encryption algorithms are basically divided into 3 groups in terms of key used for encryption and decryption: symetric algorithms, asymetric algorithms and hash algorithms. Symetric algorithms uses same private key for both encryption and decryption. Block cipher and stream cipher are the two ways of symetric ciphering. Block cipher encrypts a message by breaking it down into blocks and encrypting data in each block. Stream ciphers consists of a state machine that outputs at each state transition one bit of information.

SPN (Substitution-Permutation Network), DES (Data Encryption Standard), AES (Advanced Encryption Standard), FEAL are some examples of block cipher. S-Boxes, number of rounds, XOR operation of keys, block length and key length become crucial when the security and performance of encryption algorithms come into question. It is commonly said that "*block length should be at least equal to the key length*". Furthermore, key used for encryption should be randomly generated. Resistance to attacks has became a criteria for the evaluation of the strength and performance of an algorithm. Some of these attacks are linear cryptanalysis [17], differantial cryptanalysis [18] and related-key cryptanalysis [19]. Also different varieties of these attacks are emerged by being developed under special conditions. One example for these special attacks is impossible-differential attack. Data, data domain and time are the three important parameters of a cryptanalysis. These parameters enables to emerge three different attacks special to block cipher: dictionary attack, codebook attack and exhaustive key search.

4.2. Block Cipher

Block cipher relies on some techniques such as confusion and diffusion explained in detail in the following sections. Confusion and diffusion is realized by shifting and linear transformation operations respectively. Two main block cipher architecture are Feistel Networks and Substitution and Permutation Networks. Both architectures are combitanitions of more than one different encryption operation. As mentioned in the previous section S-Boxes, number of rounds and key are of vital significance in encryption process.

4.2.1. Key

Length of the key or number of bits in block cipher algorithms must be reliable to the most common type of attack such as exhaustive key search. For instance, DES algorithm employs key of 56-bit length whereas AES has 128, 192 and 256-bit options. One important criteria for secure key is being generated randomly.

4.2.2. Number Of Rounds

Number of rounds should be chosen carefully in block cipher algorithms since the strength and performance of substitution and linear transformation operation heavily relies on the number of rounds. Eventhough there is no optimum value for number of rounds calculated theoretically, it should satisfy Equation 4.1 according to Lars Knudsen [20];

$$r \geq dn/w \tag{4.1}$$

In the expression above, r is the loop count, d is the maximum number of rounds it takes for one word to be input to a confusion stage, n is the block size and w is the minimum word size input to the confusion stage overall in the cipher. Optimum

number of rounds can be found in the Table 4.1.

Table 4.1. Number of Rounds for Various Encryption Algorithms.

Algorithm	**Number Of Rounds**	**Optimum Number According to [20]**
DES	16	21
IDEA	8	8
Blowfish	16	16
AES	10	16

4.2.3. S-Boxes

S-Boxes are the most important part of a block cipher since it is the only non-linear structure. Strict Avalance Criteria (SAC) is one of the three criterias developed for S-Boxes. According to SAC each changing the value of 1 bit at the input results in a change at each bits of output with 1/2 probability. Another criteria is the size of S-Boxes which is usually preferred as great as possible. To be protected, greater number of output bits for differential attacks, smaller number of input bits for linear attacks are required. Last criteria to create more secure S-Boxes is that outputs should not be linear to inputs.

4.3. Advanced Encryption Standard

Advanced Encryption Standard is an encryption standard officially approved by United States Of America. It is one of block ciphers also known as Rijndael. Since DES algorithm lose its reliability against emerging technology, a competition in order to determine a new encryption standard was organized by National Institue of Standards and Technology (NIST). A new algorithm developed by two Belgian researchers Joan Daemen and Vincent Rijmen was approved as a new standard. AES FIPS 197 standard was published on 26 November 2001 by NIST after long standardisation and verification process. Compared to DES algorithm, AES provides more secure algorithm and easy to implement. Eventhough AES algorithm supports various key sizes

and data block sizes, standards covers only 128-bit fixed data block size and 128, 192, 256-bit of key size. According to AES, data blocks of length 128-bit is assumed to consist of 4 words where each word is 32-bit length. At the beginning of the encryption process the plain text of 128-bit length or 4 words are written into state array. All required steps of algorithm is performed on state array in question. Having completed the last operation on state array, it is directly written on output array. For instance data block like $in_0 in_1 ... in_{15}$ is written into state array. Then all required operations are done on this state array. As soon as the last operation was done, encrypted data block like $out_0 out_1 ... out_{15}$ is given to output as byte array.

AES algorithm mainly divided into two parts. First part covers round transformations and second includes key generation block. Since AES algorithm has repetitive structure, depending on the key size of 128-bit, 192-bit or 256-bit length number of round can be 10, 12 or 14 respectively. Number of rounds based on key sizes are shown in Table 4.2.

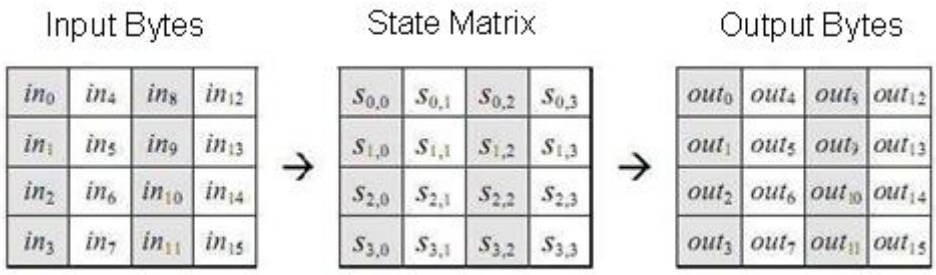

Figure 4.1. State Matrix and Input Output Bytes.

Table 4.2. Relationship Between Number of Rounds and Key Size.

AES	**Key Size (N_k)**	**Number Of Rounds (N_r)**
AES-128	4	10
AES-192	6	12
AES-256	8	14

At the beginning of the encryption process the plain text of 128-bit length or 4 words is written into state array as shown in Figure 4.1. Encryption process is started

by adding state array to input key. Depending on the key size, round transformations repeat 10, 12 or 14 times. Pseudo code is given in Figure 4.3 and block scheme is given in Figure 4.4.

```
AESEncrypt(InputState, Key)
R_Key[N_r : 1] = KeySchedule(Key);
State[0] = FirstKeyAddition(InputState, Key);
for (i = 1; i ¡ N_r; i++) do
   Round(State[i-1], R_Key[i]);
end for
FinalRound(State[N_r - 1], R_Key[N_r])
```

Figure 4.2. Pseudo Code for AES Algorithm.

During round transformation Sub-Byte Transformation, Shift-Row Transformation, Mix-Column Transformation and Add Round Key sub-operations are applied to state matrix. Output of each round added to key which is generated by key scheduler part of the system. At the end of final round state matrix is added to key to obtain encrypted data block. Final round differs from all previous rounds such that final round includes Sub-Byte Transformation, Shift-Row Transformation and Add Round Key but no more Mix-Column Transformation exists in this round. Pseudo code is given in Figure 4.3.

4.4. Round Transformations

As mentioned previously AES algorithm has a repetitive structure. Round transformation operations are repeated many times depending on the key size. Round Transformation phase covers Sub-Byte Transformation, Shift-Row Transformation, Mix-Column Transformation and Add Round Key operations. In Figure 4.5 block scheme of round transformation is illustrated.

```
Round(InputState, R_Key[i]
{
State1 = SubByteTransformation(InputState);
State2 = ShiftRowTransformation(State1);
State3 = MixColumnTransformation(State2);
OutputState = AddRoundKey(State3, R_Key[i])
}
FinalRound(InputState, R_Key[N_r])
{
State1 = SubByteTransformation(InputState);
State2 = ShiftRowTransformation(State1);
OutputState = AddRoundKey(State2, R_Key[N_r]
}
```

Figure 4.3. Pseudo Code for AES Round Transformations.

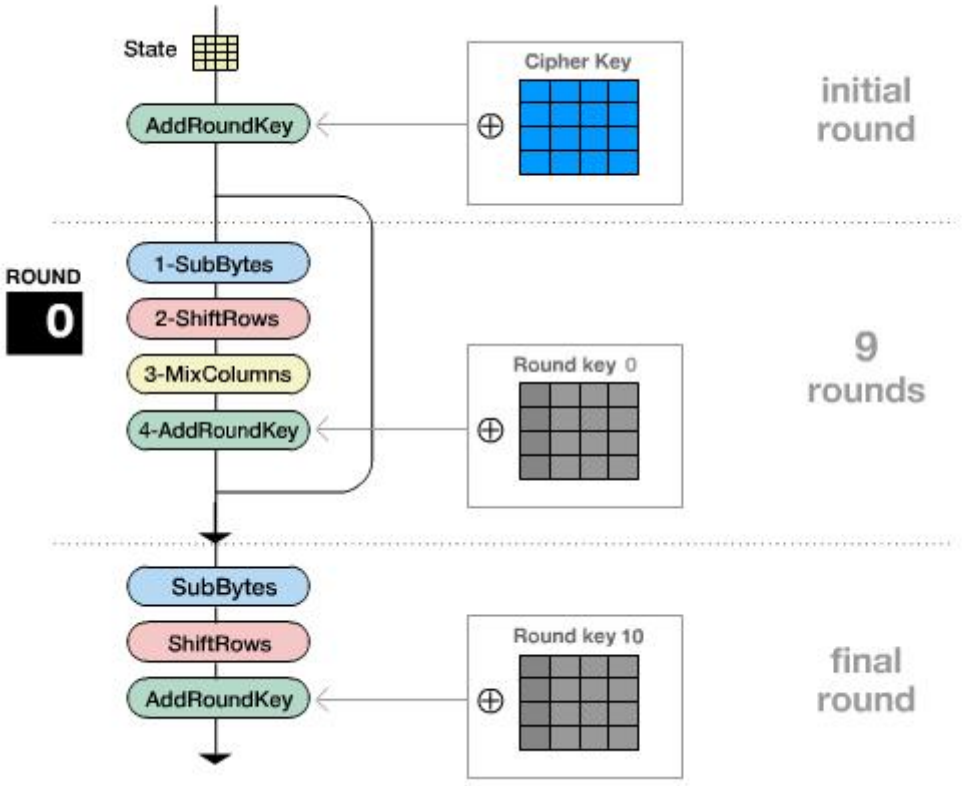

Figure 4.4. AES Block Scheme.

Figure 4.5. Round Transformation Block Scheme.

4.4.1. Sub-Byte Transformation

Sub-Byte transformation layer is non-linear byte substitution process which is composed of multiplicative inverse and affine transformation [21] as shown in Figure 4.6. This operation is performed on each bytes of 4x4 state matrix. Sub-byte transformation has two phases. First phase is multiplicative inverse operation which is realized in GF(2^8). The reducing polynom for this operation is shown in Equation 4.2.

$$P(x) = x^8 + x^4 + x^3 + x + 1 \tag{4.2}$$

In the second phase transition matrix is multiplied by inverted output in GF(2) and added by constant matrix. Constant matrix added by multiplicative inverse is given

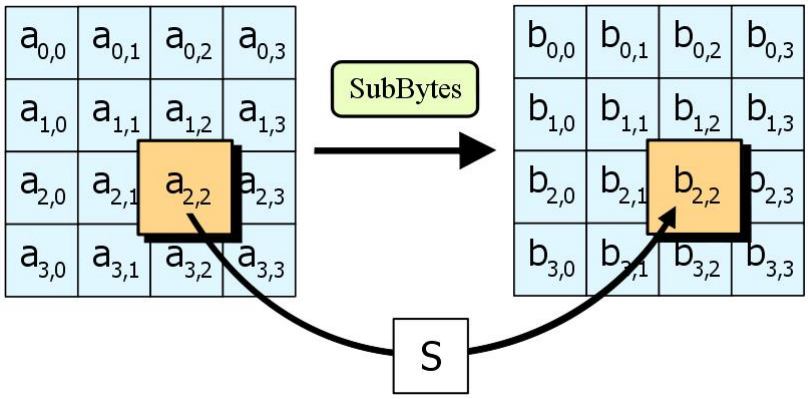

Figure 4.6. Sub-Byte Transformation.

with Equation 4.3 as follows;

$$\begin{bmatrix} \hat{b_0} \\ \hat{b_1} \\ \hat{b_2} \\ \hat{b_3} \\ \hat{b_4} \\ \hat{b_5} \\ \hat{b_6} \\ \hat{b_7} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{4.3}$$

It is possible to implement S-Box Transformation in several ways which are going to be explained in detail while mentioning about system model. At this point a more common method based on lookup table (LUT) is presented in Figure 4.3. Each byte has a pre-calculated value in the table.

Table 4.3. S-Box Look Up Table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7c	77	7b	f2	6b	6f	C5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
A	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
B	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
C	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
D	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
E	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
F	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

4.4.2. Shift-Row Transformation

In the Shift Row Transformation each row of state matrix except the first row are subjected to left shift operation by a certain offset. Second row is shifted one to the left. Two left for third row and forth row is shifted to left by three as shown in Figure 4.7.

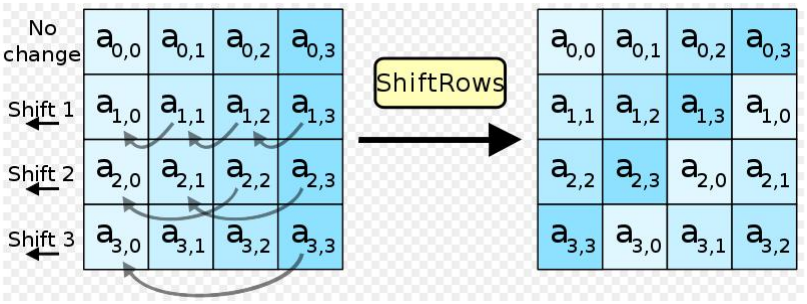

Figure 4.7. Shift-Row Transformation.

4.4.3. Mix-Column Transformation

Mix column transformation is performed on each column of state matrix independently as shown in Figure 4.8. Column vector is multiplied with fixed matrix where the bytes are treated as a polynomials rather than number [21]. Transformation is simply matrix multiplication over GF(2^8). At each row *a(x)* polynom is multiplied by elements in of each column in modulo $x^4 + 1$. Constant *a(x)* polynom and parametric definiton of Mix-Column Transformation satisfies Equation 4.4 and Equation 4.5 as follows;

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \tag{4.4}$$

$$\hat{s}(x) = a(x) * s(x) \tag{4.5}$$

Mix-Column operation over the rows is represented as multiplication of matrixes in the following Equation 4.6;

$$\begin{bmatrix} \hat{S_{0,c}} \\ \hat{S_{1,c}} \\ \hat{S_{2,c}} \\ \hat{S_{3,c}} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \tag{4.6}$$

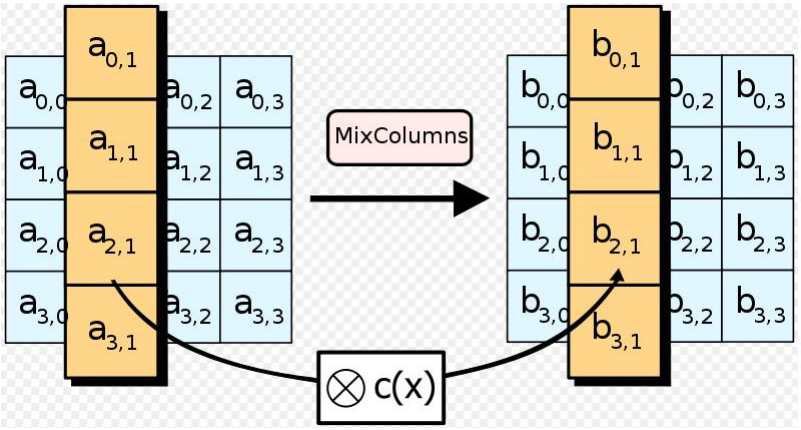

Figure 4.8. Mix Column Transformation.

4.4.4. Add Round Key

Add Round Key phase, as is evindent from it's name, is simply an XOR operation of state matrix with newly generated key array of 128-bits length as illustrated in Figure 4.9.

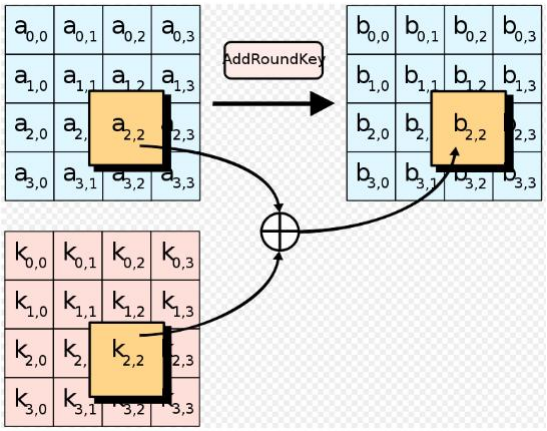

Figure 4.9. Add Round Key.

4.5. Key Scheduling

Let K be the key array. AES algorithm generates different key blocks required at next round by processing K array. AES algorithm supports various key sizes. However, generated key size is 128-bit length since data block size is 128-bit length. This generated key is used at Add Round Key phase which is the last sub-operation of round transformation. Key scheduling block, also known as key generator, firstly takes 128-bit key from input and stores in the form of 4x4 matrix whose each element is byte. Then the other 128-bit keys generated further rounds are used to expand the key matrix. By doing this a matrix of 4*(N_r + 1) dimension is obtained eventually. N_r is the number of round. Key generation operation is performed in such a way illustrated in Figure 4.10. A column which is about to be created is not a multiply (N_b) of

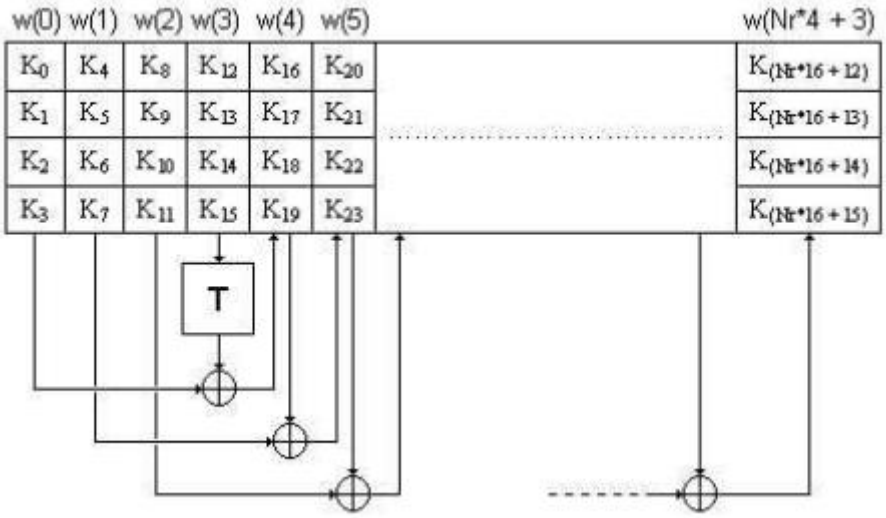

Figure 4.10. Key Generation For 128-bit Input.

word number of the key used at the input (N_b is 4 for 128-bits key size), that column is supposed to be obtained by XOR operation of previous column with the N_b times before the last column. If a column to be generated is a multiple of N_b, the previous column is transformed by series of operations and then put into XOR operation with N_b times before the last column. Key generation is scheduled to complete 3 phases. In the first phase shifting is performed on a column which includes 4 byte of data totally. Having completed the shifting, each byte is subjected to Sub-Byte transformation. In the next phase e-XOR of this column and round vector is calculated. First byte of

round vector depends on which round is currently being processed. Other bytes are always zero. Key generation phases and round vector is expressed in the following equations;

$$\{K_1, K_2, K_3, K_4\} = \{K_2, K_3, K_4, K_1\} \tag{4.7}$$

$$\{\hat{K_2}, \hat{K_3}, \hat{K_4}, \hat{K_1}\} = S - Box(\{K_1, K_2, K_3, K_4\}) \tag{4.8}$$

$$\{A_1, A_2, A_3, A_4\} = \{\hat{K_2}, \hat{K_3}, \hat{K_4}, \hat{K_1}\} \oplus \{Rcon, 00, 00, 00\} \tag{4.9}$$

$$Rcon(0) = \{01\} \tag{4.10}$$

$$Rcon(i) = x * Rcon(i-1) \tag{4.11}$$

Pseudo code for key scheduling can be found below in Figure 4.5;

4.6. Related Works

Several AES implementations has been implemented since the announcement of AES until now. Shortly after Rijndael was announced, designers mostly focused on feedback, high speed and high operation capability. Main concern in those implementations are throughput rate, power consumption and resistance to attacks. From the perspective of throughput rate, namely performance, significant improvements has been recorded such that megabits or even gigabit levels per second was

```
KeyScheduling(InputKey, AMatrix)
{
if i < Nb then
   AMatrix(i) = InputKey((32*i)-1:0);
   if i > Nk-1 then
      if i = 0 mod(Nb) then
         TempState = SBox(ShiftColum(AMatrix(i-1))) exor RCon
      else
         TempState = AMatrix(i);
      end if
      AMatrix(i) = TempState exor AMatrix(i-Nb))
   end if
end if
}
```

Figure 4.11. Pseudo Code for Key Generation.

reached [22, 23]. Also some FPGA implementations have considerably high performance [24–27]. In [27] AES has been implemented on FPGA which has 12.2Gbit/s throughput rate without any feedback.S-Box is hard to implement, most power dissipating, resource consuming and speed limiting part of AES. Most of the previous works therefore focused on improving S-Box [28–33].Table structure using LUT's of FPGA is one way to implement S-Box, while the other common method is combinational circuit implementation. Various implementations and results are investigated in [34].For power consumption, widespread solutions such as ASIC choice, Pass Transmission Gate (PTG) Logic or 3-input XOR depends on hardware modifications [35–37]. However all solutions published previously are synchronously implemented solutions except three [38–40]. In [38], Bousse presents an asynchronous crypto processor implemented with 0.13μm CMOS technology from STMicroelectronics. The results for 128-bits AES are crypto-processor are 910 ns ciphering time, 141 mbits/sec throughput with energy of 1.25nJ. Other asynchronous AES implementation published by D.Shang used 0.35μm CMOS technology. However this thesis focuses on how to increase resistance to attack of AES by making use of asynchronous techniques instead power reduction. The last one, [40] was implemented on a reconfigurable processor with total power consumption as 108.75mW.

5. SYSTEM MODEL

In this chapter design process is described besides results of power analysis. Since the main concern of this thesis is low-power AES design, asynchronous methodology was chosen. To realize asynchronous circuits, Balsa tool is used. This tool is not only capable of describing asynchronous circuits, but also capable of synthesizing them. With the aid of project navigator in Balsa framework, one can perform both text based or channel based simulation and calculate the total cost of the circuitry in terms of area as seen below. Having completed the design process of AES, next is power validation of circuit. For this purpose Xpower tool of Xilinx ISE is used.

This thesis covers two different architectures of asynchronous AES. Since the most power consuming part of AES is S-Box, all alternative design options focus on alternative ways to implement S-Box. One way depends on S-Box which is designed as a simple LUT as given in the previous chapters. Second way, however, makes use of combinationally implemented S-Box. LUT based S-Box implementations has a table of pre-calculated values to replace input bytes. However in combinational case, each counterpart of input values are calculated at run-time of algorithm. Brief information on how combinational transformation works is as follows [41].

The Subbyte transformation concludes with performing multiplicative inverse in $GF(2^8)$ and affine transformation respectively.

Matrix operation given by Figure 5.1 illustrates how Affine Transformation is applied. In this figure, the AT is the Affine transformation and the vector a is the input byte from the state array. According to the S-Box construction methodology used by E. Mui [41] Affine Transformation follows multiplicative inversion operation as illustrated in the Figure 5.2. The legends for the blocks used in the construction of multiplicative inversion module can be seen in Figure 5.3.

$$AT(a) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

Figure 5.1. Affine Transformation.

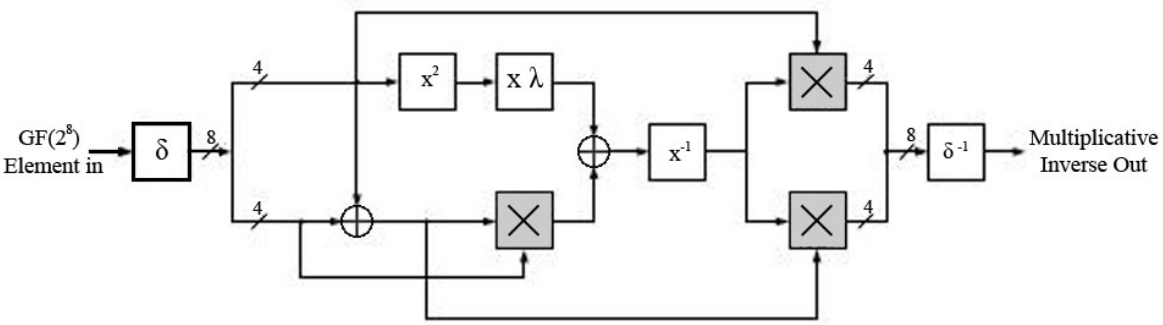

Figure 5.2. Multiplicative Inversion [4].

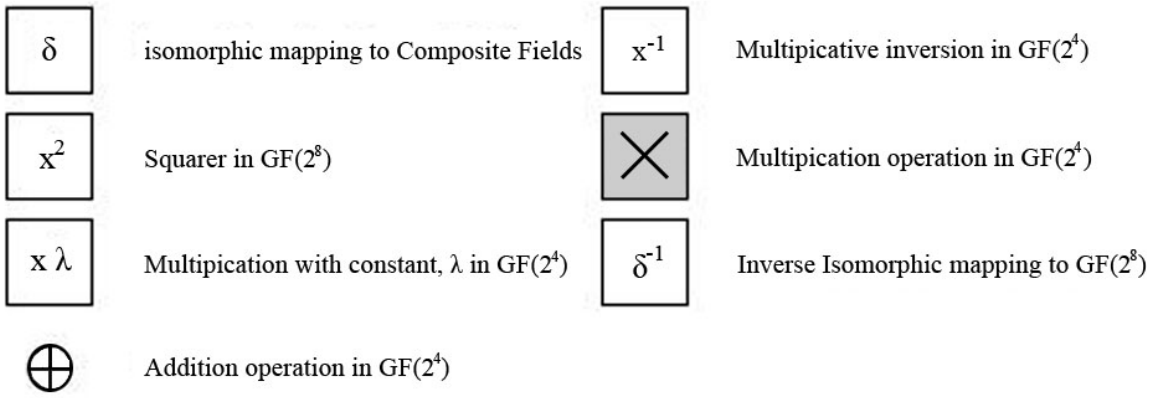

Figure 5.3. Legends for The Multiplicative Inversion Module.

5.1. Asynchronous AES Handshake Graph

Since the handshake graph of the entire AES circuitry is extremely big, Figure 5.4 shows AES handshake diagram partially. The part in the figure is Mix-Column block. Each element of handshake graph is called handshake component which are located in balsa library. The inputs of AES are i_0d, i_0r, i_0a which are data, request and acknowledge signals respectively. This also the case for output. First the data is requested from the previous part, as soon as the data is taken, acknowledge signal is set high. Shortly after calculation the result, output data is written on o_0d channel. If any request signal is activated on the output port o_0r, newly calculated data is pushed to communication channel.

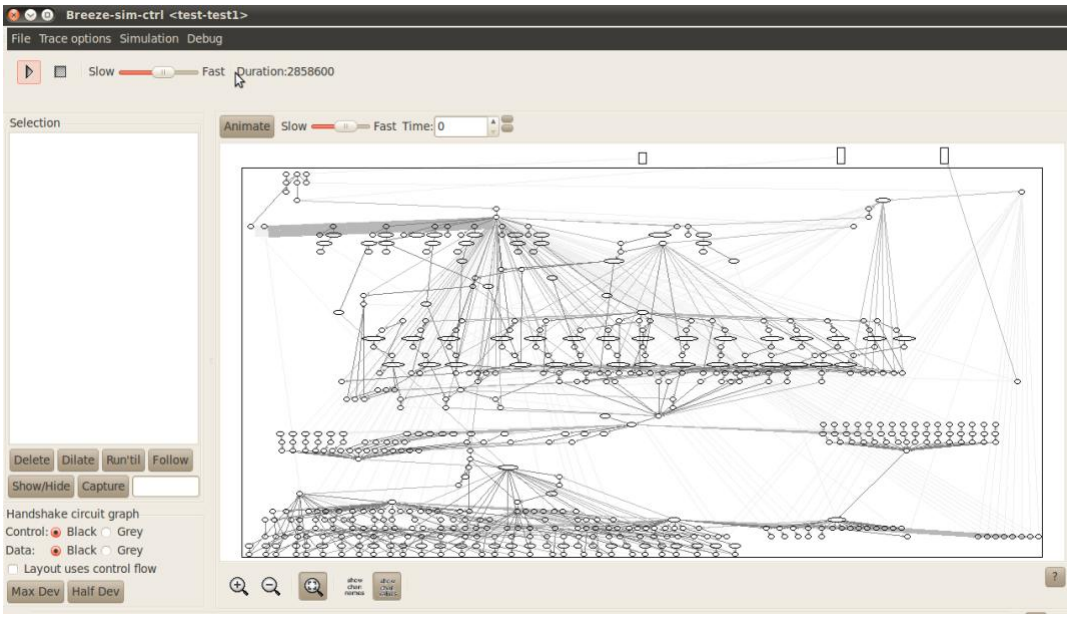

Figure 5.4. Handshake Graph of Mix-Column Transformation.

5.2. Area Estimation of Asynchronous AES

As mentioned previously, Balsa framework is a collection of various tools one of which provides cost estimation in terms of area. Total area cost in calculated by summation of each handshake components building an asynchronous AES system. Asynchronous AES design includes following handshake components; `$BrzContinuePush`, `$BrzFetch`, `$BrzConstant`, `$BrzWhile`, `$BrzBinaryFuncConstR`, `$Br-`

zSplitEqual, $BrzEncode, $BrzCombinEqual, $BrzSequence, $Brz-Call, $BrzBinaryFunc, $BrzCaseFetch, $BrzVariable, $BrzCallm-ux, $BrzConcur, $BrzCase. As seen in Figure 5.5 total estimated area for AES with LUT based S-Box is 2017283.25 unit whereas according to Figure 5.6 it is 238403.66 unit for AES with combinationally implemented S-Box.

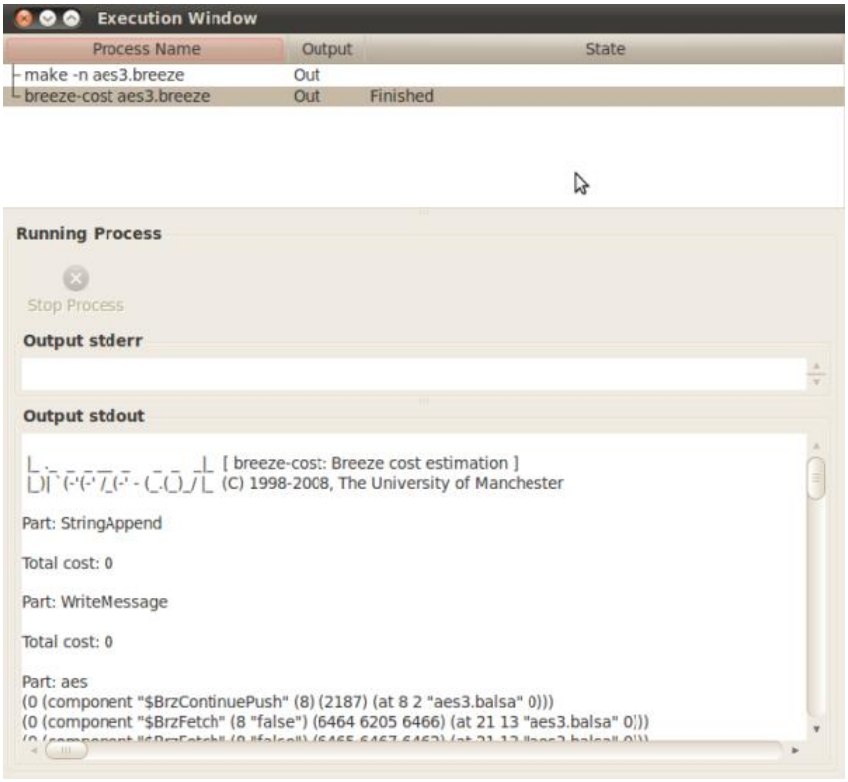

Figure 5.5. Cost Estimation of AES with LUT Based S-Box.

5.3. Asynchronous AES Simulations

- *Real Time Text Simulation*: One way to simulate balsa source file is text based simulation. By doing this the inputs and outputs of AES cipher block can be monitored in a text form. In Figure 5.7, 613 nanosecond takes for AES to initialize handshaking, then input data is taken. It is obvious that one input is for 128-bit of plain text. Plain text is the one which is going to be encrypted in accordance with Rijndael algorithm. Other input which is also 128-bit is initial key for the algorithm. This key is used to generate further keys. Ciphering process takes 49567420 nanosecond as seen in Figure 5.7. Text based simulation indicates that

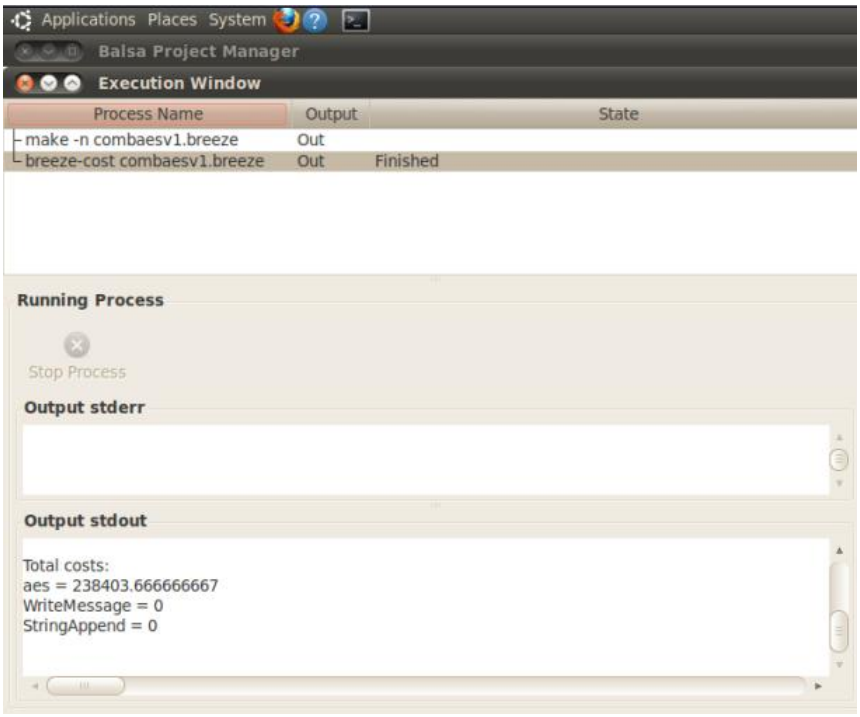

Figure 5.6. Cost Estimation of AES with Combinationally Implemented S-Box.

input channel is written with values 0x848F8E92A8DC69A2BE2F4A0BEE33D19 which is 128-bit plain text itself, and 0x3C4FCF098815F7ABA6D2AE2816157E2B which is initial key. The output channel is written with the 128-bit encrypted plain text value 0x320B6A19978511DCFB09DC021D842539. Since the thesis covers two different architectures to be compared, text based simulation is therefore presented with two figures where the second is Figure 5.8. According to the text based simulation results of AES with combinationally implemented S-Box, first initialization time does not differ. However total simulation time is 127272420 nanoseconds which is slightly better than that of AES with LUT based S-Box.

- *Real Time Channel Simulation*: Second way of simulating balsa source files is channel based simulation. This option is more visualized simulation rather then text simulation since all channel activities are indicated with certain colors. Figure 5.9 shows the channel simulation of sub-byte block. since the entire AES simulation picture requires more space, Sub-Byte block is depicted instead of whole circuitry. Each color has its own meaning such that red and green colors represent Request and Acknowledge signals respectively. Having completed

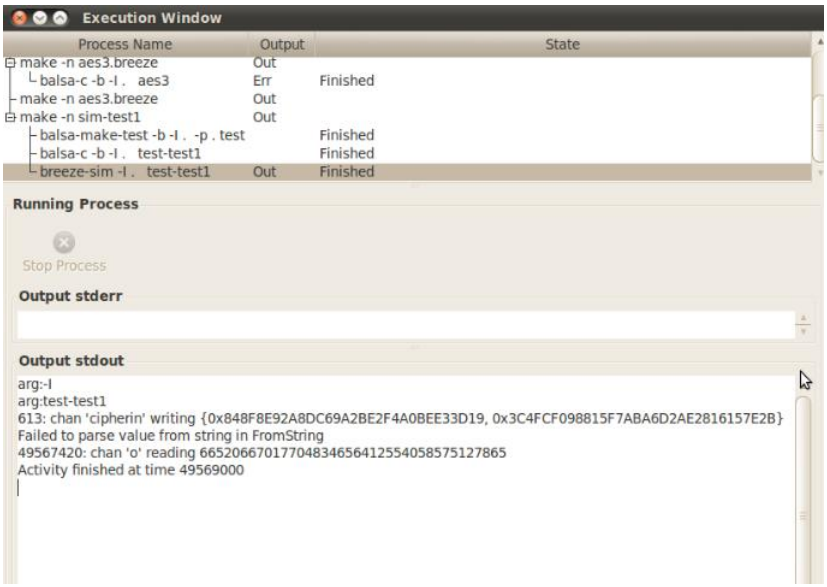

Figure 5.7. Text Based Simulation of AES with LUT Based S-Box.

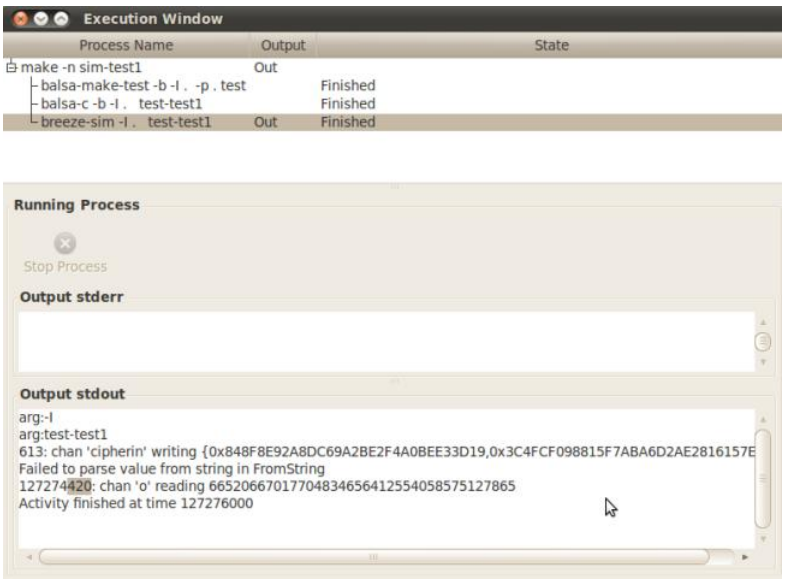

Figure 5.8. Text Based Simulation of AES with Combinationally Implemented S-Box.

every handshake, a new value is written on the input channel.

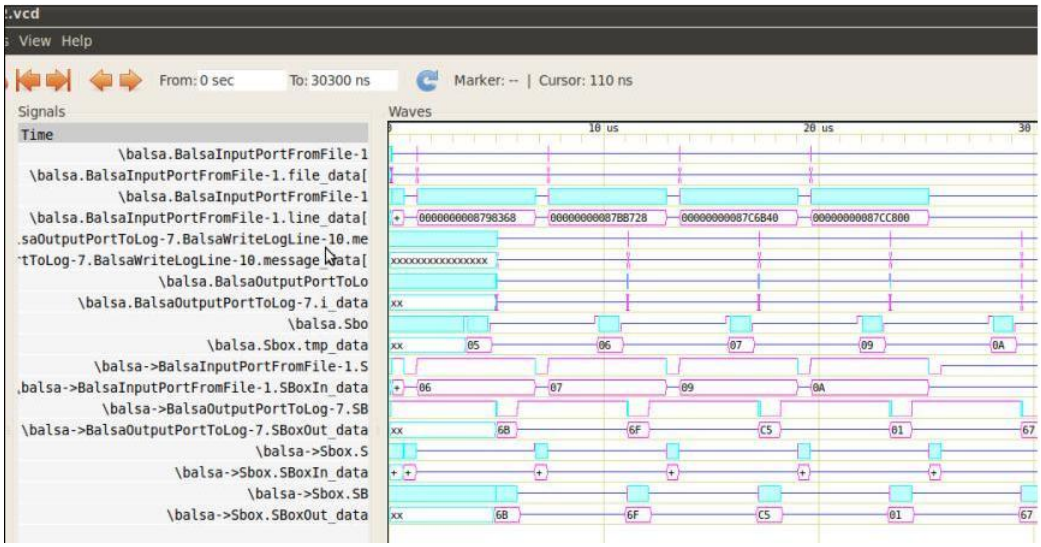

Figure 5.9. Real Time Channel Simulation.

5.4. Handshaking Animation

Balsa framework has a distinguishing feature that is animating whole channel activities depending on the design methodology. In Figure 5.10 shows how asynchronous Mix-Column block of AES communicates on handshake channels. Breeze-Sim Controller is the tool used for such animations. The handshake graph in the figure is obtained in accordance with 4-phase bundled data protocol. By its nature this protocol requires 4 colors for each phase. Red for request and green for acknowledge phase. Blue color indicates the Request Done phase and gray color indicates the Acknowledge Done phase. Breeze-Sim Controller tool is also very critical for detection of deadlocks. Deadlocks are usually the consequences of inefficient coding of Balsa source codes.

5.5. Asynchronous AES Design Synthesis

Having completed coding of asynchronous AES with Balsa in two alternative architecture, designs are now ready to be synthesized using Xilinx ISE. The link between Balsa framework and Xilinx ISE is constructed by a verilog netlist which is generated by Balsa. Figure 5.11 and Figure 5.12 show synthesized AES entity. Asynchronous

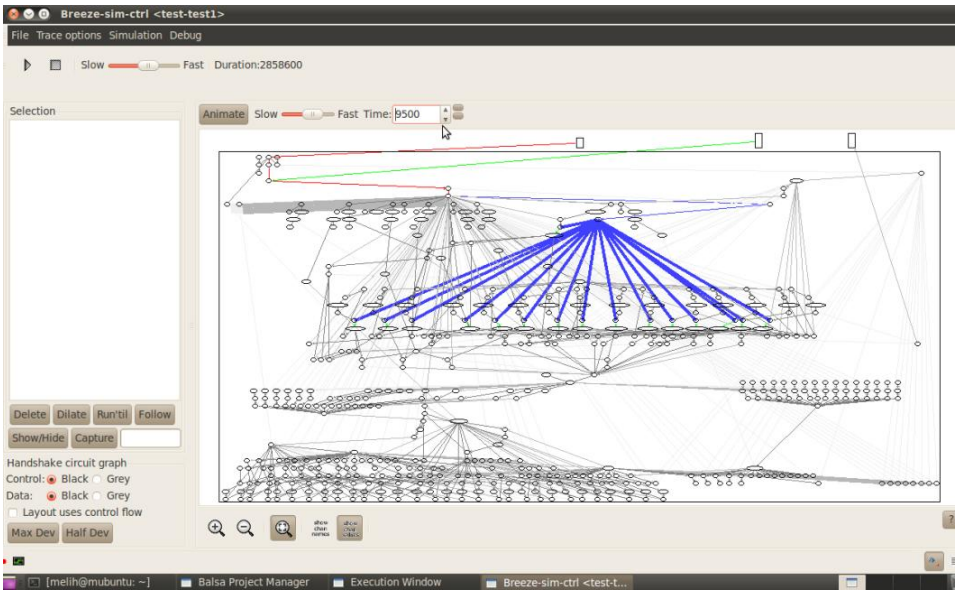

Figure 5.10. Handshaking Animation.

AES differs from synchronous one in that there is no global clock distributed among the whole circuit. Besides asynchronous one has additional I/O ports such that all data signals have request and acknowledge signal associated with them. As seen in Figure 5.11, before ciphering process AES block is initialized by initialize signal. Then circuit is activated by activate_0r signal. If activation is achieved then activate_0a is set high. Cipherin_0r indicates that AES request new plaint text and cipher key, then as soon as cipherin_0d is taken by AES, cipherin_0a signal is activated. After handshaking with the environment for transfering data, asynchronous computation phase runs until the output port $o\_0d(127:0)$ was loaded with the calculation result. Now, handshaking takes place in order to push output data to outer environment.

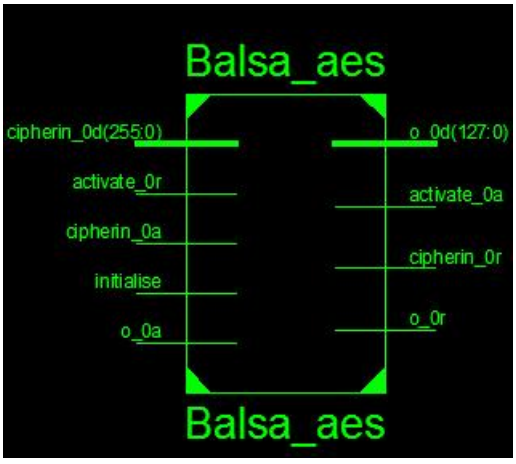

Figure 5.11. Asynchronous AES Entity.

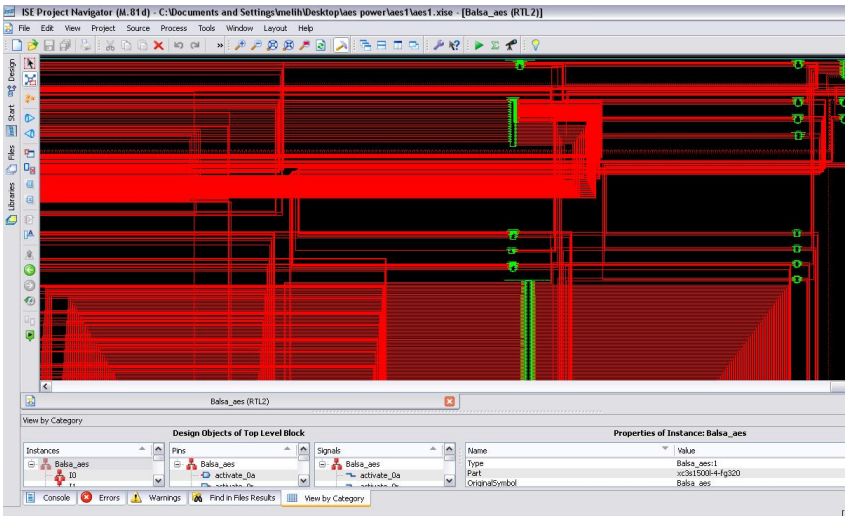

Figure 5.12. Asynchronous AES RTL Schematic.

5.6. FPGA Implementation

Having completed whole steps of designing an asynchronous AES, the circuit is synthesized using Xilinx ISE. In order to validate power consumption using Xpower tool, first the design should be implemented on a FPGA. Since the power consumption depends on routing, each FPGA family has its own power values. For this implementation XC4VLX200 family was chosen. After the design is placed and routed, testbench runs and the simulation result is obtained.

5.7. Benchmarking Between Synchronous And Asynchronous Implementations

Firstly, it should be mentioned that both designs presented in this thesis are the first asynchronous AES implementation on FPGA developed using Balsa. For this reason, further comparisons will be done by taking this distinguishing point into account. Also the S-Box Transformation is done using table structure which brings significant increase in speed. Since the main concern in this thesis is designing low-power AES, comparative analysis is done on the basis of power consumption. As mentioned above AES is designed asynchronously since it is expected to dissipate low-power. This expectation is examined in this section by benchmarking with synchronous

designs. XPower tool from Xilinx Inc. is used for dynamic power analysis. In

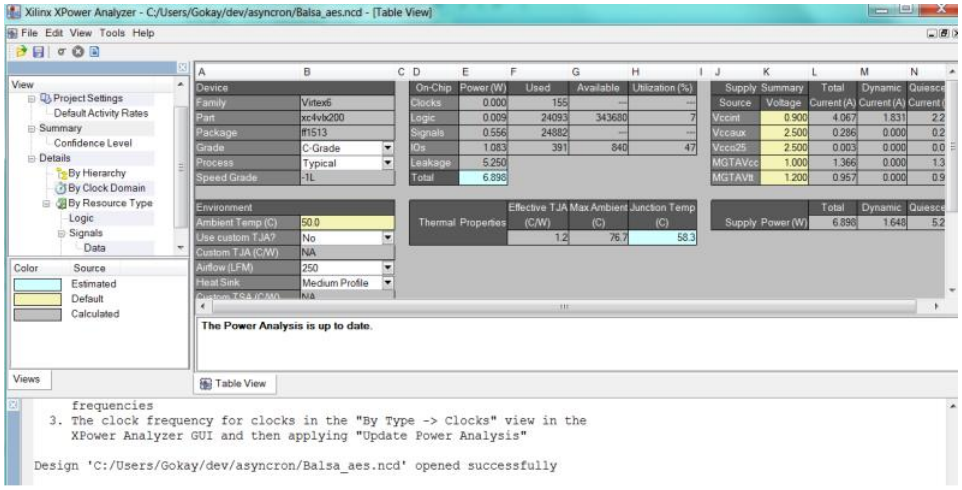

Figure 5.13. Power Analysizing of AES with LUT Based S-Box Using XPower.

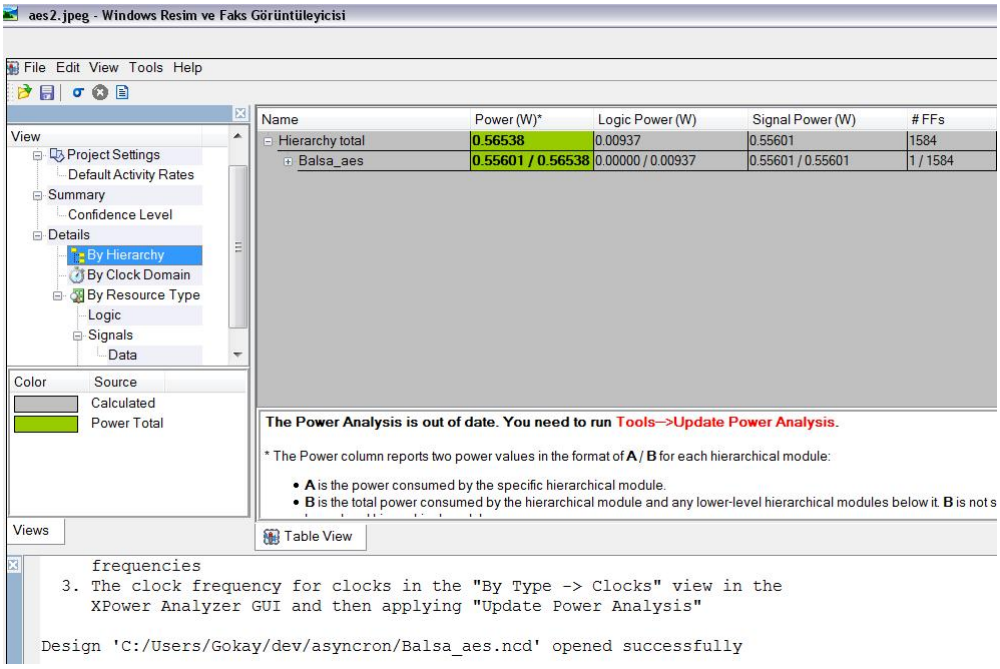

Figure 5.14. Power Consumption of Asynchronous AES with LUT Based S-Box.

order to calculate power using Xpower, at least two files are needed. First one is Native Circuit Description (.ncd) file. Having synthesized the verilog netlist that is generated by Balsa, Xilinx ISE maps the primitives onto the types of resources in the specific FPGA being targeted. The output of the Xilinx map tool is an .ncd file. Then the design is placed and routed. By doing this whole interconnection network is established. The output of Place&Route step is updated .ncd file. The other file

that is necessary for XPower analysis is Value Change Dump (.vcd) file. This file is obtained using $dumpfile("aes.vcd") command. As seen in Figure 5.13 and Figure 5.14 using .ncd and .vcd files, total power dissipated during switching activity is calculated. Further comparison is needed since there is two asynchronous design architectures. As mentioned previously the major power consuming part of AES is S-Box module. To reduce the power consumption, various design options for S-Box module come up. In this work one option is LUT based S-Box and the other is combinationally implemented S-Box. Figure 5.15 and Figure 5.16 illustrates how the power consumption reduces with the change in S-Box design.

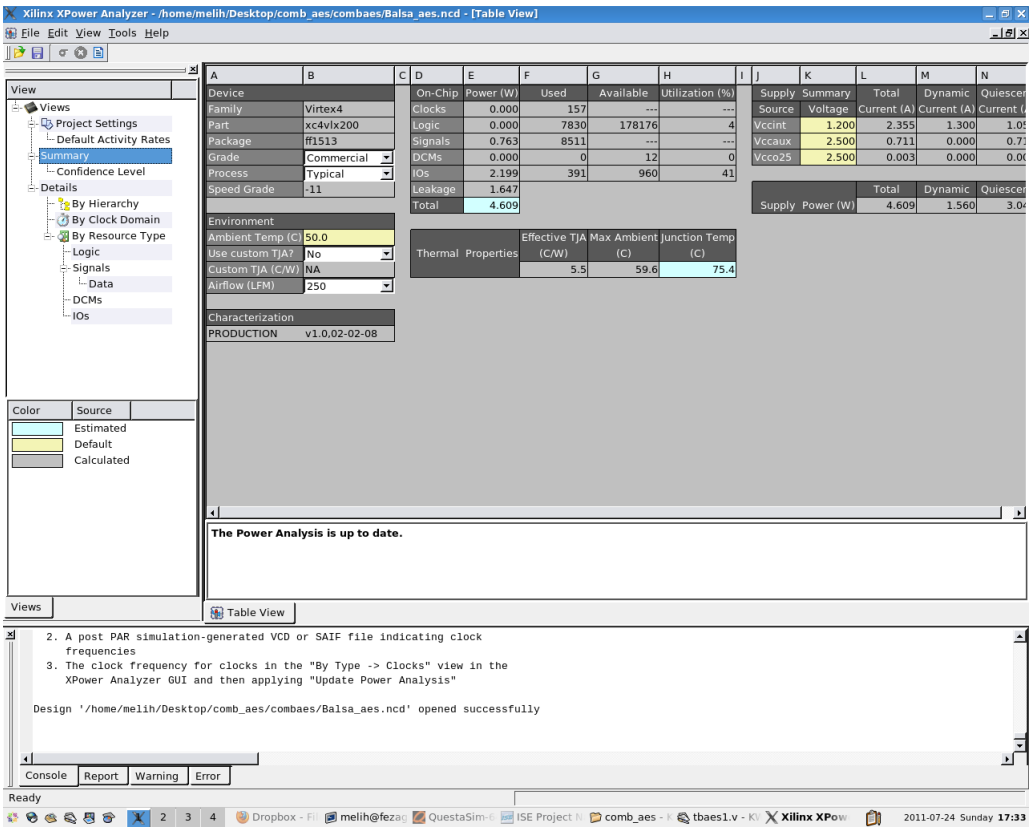

Figure 5.15. Power Analysizing of AES with Combinational S-Box Using XPower.

Results are presented in Table 5.1. First result belongs to [42], and both second and third designs are presented in this thesis. As per discussion in Section 4.6, [38] and [40] is not suitable to compare since they were not implemented on a FPGA target device and their S-Box implementations are not based on table structure. In [34] as mentioned earlier many S-Box implementation were compared. However results in [42] are more satisfying that is why [42] is treated as most comparable one. As seen

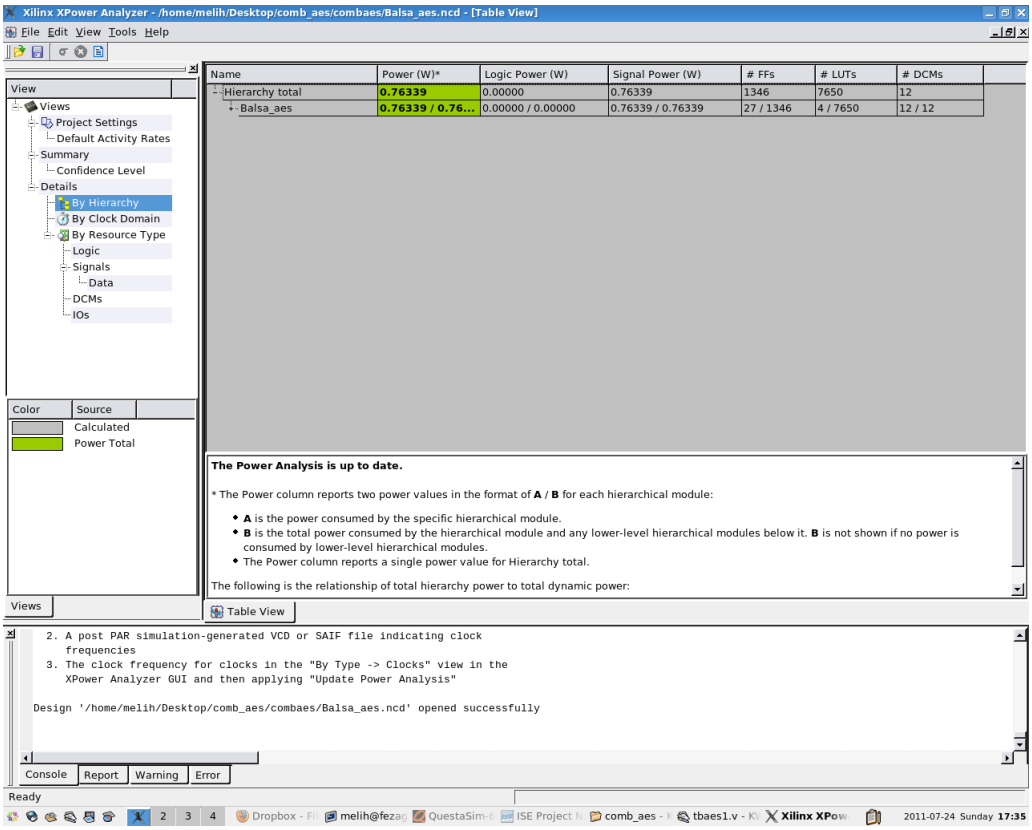

Figure 5.16. Power Consumption of Asynchronous AES with Combinational S-Box.

Table 5.1 asynchronous methodology brings almost 65% of power reduction compared to synchronous designs.

Table 5.1. Power Benchmark.

Design Methodology	**Power (mW)**	**Area (Slice, FF, LUT)**	**Critical Path (ns)**
Synchronous	1581	(2629 S, 931 F, 4932 L)	8.163
LUT Design	565	(20577 S, 1584 F, 33143 L)	9.438
Combinational Design	763	(4201 S, 1346 F, 7830 L)	15.101

Since there is two asynchronous design architectures proposed in this work, an internal comparison should be done. To make it clear, it should be mentioned that the LUT design refers to AES which includes S-Box module implemented as LUT. On the other hand, combinational design refers to those AES circuits which calculates the subbyte transormation values at each time using a combinational circuit. As

seen in Table 5.2 there is no difference between two architectures in terms of circuit initialization time. On the other hand second architecture is prior to the first, since the area is almost 10% of the first architecture. This is simply because of first architecture performs subbyte transformation using pre-calculated values. However area advantage brings another side effect by its nature which is increase in simulation time. Also this is expectable since it takes some amount of time to calculate sub-byte transformation counterparts of each values in the state matrix. This calculation load brings a new and interesting result such that eventhough combinational architecture occupies less resources in FPGA, it consumes more power. This is simply explained by the condense handshaking activity. The more the handshaking, the more the power consumption.

Table 5.2. Simulation Benchmark.

Design Methodology	**Simulation Time (ns)**	**Area (Balsa Unit)**	**Initialization Time (ns)**
LUT	49567420	2017283.25	613
Combinational	127272420	238403.66	613

Since asynchronous signaling is rather complicated then synchronous one, area overhead is inevitable. On the other side performance of synchronous designs are determined by the global clock frequency. The more the performance, the higher the clock frequency. Increase in clock frequency means more switching activity on various nets which cause power to be dissipated more. Obviously clock-free designs like asynchronous do not have such disadvantages as XPower result indicates.

6. CONCLUSION

Since the AES has a great range of application including mobile applications like smart cards, power dissipation becomes increasingly important. In this thesis how power consumption of an AES can be reduced has been investigated. For this purpose asynchronous methodology has been proposed to consume lower power in comparison to synchronous designs. To be more specific, there are two proposed designs dissipating 565mW and 763mW respectively where the compared design dissipates 1581mW. Power benchmarking at the end shows that expectations are consistent with the results provided by industry standard power analyzer tool.

REFERENCES

1. Geer, D., "Is it time for clockless chips? [Asynchronous processor chips]", *Computer*, Vol. 38, No. 3, pp. 18–21, 2005.

2. Sparso, J. and S. Furber, *Principles of Asynchronous Ciruit Design-A Systems Perspective*, Kluver Academic Publishers, Lyngby, 2001.

3. Edwards, D., A. Bardsley, L. Janin, L. Plana and W. Toms, "Balsa: A Tutorial Guide", ftp://ftp.cs.man.ac.uk/pub/amulet/balsa/3.5.1, 2006, accessed at October 2010.

4. Satoh, A., S. Morioka, K. Takano and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization", *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, Vol. 2248, No. 1, pp. 239–254, London, 2001.

5. Van Berkel, C., M. Josephs and S. Nowick, "Applications of asynchronous circuits", *Proceedings of the IEEE*, Vol. 87, No. 2, pp. 223–233, 1999.

6. Janin, L., *Simulation and Visualisation for Debugging Large Scale Asynchronous Handshake Circuits*, Ph.D. Thesis, University Of Manchester, 2004.

7. Van Berkel, K., F. Huberts and A. Peeters, "Stretching quasi delay insensitivity by means of extended isochronic forks", *Proceedings of the 2nd Working Conference on Asynchronous Design Methodologies*, Vol. 22, No. 16, pp. 99–106, Washington DC, 1995.

8. Edwards, D. A. and A. Bardsley, "Balsa: An Asynchronous Hardware Synthesis Language", *The Computer Journal*, Vol. 45, No. 1, pp. 12–18, 2002.

9. Zhang, Q. and G. Theodoropoulos, "Modelling SAMIPS: A Synthesisable Asyn-

chronous MIPS Processor", *Proceedings of the 37th annual symposium on Simulation*, Vol. 26, No. 3, pp. 205–212, Washington DC, 2004.

10. Cortadella, J., M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers", *IEICE Transactions on Information and Systems*, Vol. E80-D, No. 3, pp. 315–325, 1997.

11. Department of Computer Architecture at the Polytechnic University of Catalonia, "*VERSIFY Release 2.0*", http://www.ac.upc.es/vlsi/versify/, 1998, accessed at May 2011.

12. Birtwistle, G. and A. Davis, *Asynchronous Digital Circuit Design*, Springer-Verlang, Cleveland, 1995.

13. Stevens, K., J. Aldwinckle, G. Birtwistle and Y. Liu, "Designing parallel specifications in CCS", *In Proceedings of Canadian Conference on Electrical and Computer Engineering*, Vol. 2, No. 5, pp. 983–986, 1993.

14. Hoare, C. A. R., "Communicating Sequential Processes", *Communications of the ACM*, Vol. 21, No. 7, pp. 666–677, 1985.

15. Tsirogiannis, E., G. Theodoropoulos, D. Chen, Q. Zhang, L. Janin and D. Edwards, "A Framework for Distributed Simulation of Asynchronous Handshake Circuits", *Proceedings of the 39th annual Symposium on Simulation*, Vol. 72, No. 16, pp. 214–222, Washington DC, 2006.

16. Van Berkel, K., J. Kessels, M. Roncken, R. Saeijs and F. Schalij, "The VLSI-programming language Tangram and its translation into handshake circuits", *Proceedings of the European Conference on Design Automation*, Vol. 53, No. 7, pp. 384–389, 1991.

17. Matsui, M., "Linear cryptanalysis method for DES cipher", *Workshop on the*

theory and application of cryptographic techniques on Advances in cryptology, Vol. 15, No. 8, pp. 386–397, Lofthus, 1994.

18. Biham, E. and A. Shamir, "Differential Cryptanalysis of the Full 16-round DES", pp. 487–496, Springer-Verlag, Rehovot, 1993.

19. Schneier, B., *Applied cryptography: protocols, algorithms, and source code in C*, 2nd Edition, John Wiley & Sons, Inc., New York, 1995.

20. Knudsen, L. R., "The Number of Rounds in Block Ciphers", *Public Reports of Nessie Project*, 2000.

21. Zhong, Y. Q., J. M. Wang, Z. Zhao, D. Yu and L. Li, "A Low-cost and High Efficiency Architecture of AES Crypto-engine", *Proceedings of Second International Conference on Communications and Networking in China*, Vol. 13, N0.2, pp. 308–312, 2007.

22. Hodjat, A., D. D. Hwang, B. Lai, K. Tiri and I. Verbauwhede, "A 3.84 gbits/s AES crypto coprocessor with modes of operation in a 0.18μm CMOS technology", *Proceedings of the 15th ACM Great Lakes symposium on VLSI*, Vol. 21, No. 3, pp. 60–63, Illinois, 2005.

23. Diffie, W. and M. E. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory*, Vol. 22, No. 2, pp. 644–654, 1976.

24. Chodowiec, P. and K. Gaj, "Very Compact FPGA Implementation of the AES Algorithm", C. Walter, Ç. K. Koç and C. Paar (Editors), *Proceedings of the 5th International Workshop on Cryptographic Hardware and Embedded Systems*, Vol. 2779, No. 32, pp. 319–333, Cologne, 2003.

25. Pramstaller, N. and J. Wolkerstorfer, "A Universal and Efficient AES Co-Processor for Field Programmable Logic Arrays", *Proceedings of FPL*, Vol. 1361, No. 11, pp. 565–574, Leuven, 2004.

26. Aziz, A. and N. Ikram, "An FPGA-based AES-CCM Crypto Core For IEEE 802.11i Architecture", *International Journal of Network Security*, Vol. 5, No. 2, pp. 224–232, 2007.

27. Gaj, K. and P. Chodowiec, "Fast implementation and fair comparison of the final candidates for Advanced Encryption Standard using Field Programmable Gate Arrays", *Topics in Cryptology - The Cryptographers' Track at the RSA Conference*, Vol. 2020, No. 64, San Francisco, 2001.

28. Bertoni, G., M. Macchetti, L. Negri and P. Fragneto, "Power-efficient ASIC synthesis of cryptographic sboxes", *Proceedings of the 14th Great Lakes symposium on VLSI*, Vol. 3347, No. 26, pp. 277–281, Boston, 2004.

29. Canright, D., J. Rao and B. Sunar, "A Very Compact S-Box for AES", *Proceedings of the 7th International Workshop on Cryptographic Hardware and Embedded Systems*, Vol. 3659, No. 17, pp. 441 – 455, Edinburgh, 2005.

30. Macchetti, M. and G. Bertoni, "Hardware Implementation of the Rijndael SBOX:A Case Study", *ST Journal of System Research*, Vol. 769, No. 3, pp. 84–91, 2003.

31. Mentens, N., L. Batina, B. Preneel and I. Verbauwhede, "Systematic Evaluation of Compact Hardware Implemantations for Rijndael S-BOX", *The Cryptographers' Track at the RSA Conference*, Vol. 3376, No. 5, pp. 323–333, San Francisco, 2005.

32. Morioka, S. and A. Satoh, "An Optimized S-Box Circuit Architecture for Low Power AES Design", *Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, Vol. 2523, No. 11, pp. 172–186, Redwood Shores, 2002.

33. Satoh, A., S. Morioka, K. Takano and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization", *Proceedings of Advances in Cryptology of the 7th International Conference on the Theory and Application of Cryptology and*

Information Security, Vol. 2248, No. 16, pp. 239–254, Gold Coast, 2001.

34. Tillich, S., M. Feldhofer, T. Popp and J. Groschadl, "Area, Delay, and Power Characteristics of Standard-Cell Implementations of the AES S-Box", *Journal of Signal Processing Systems*, Vol. 50, No. 3, pp. 251–261, 2008.

35. Zhenglin, L., Z. Yonghong, Z. Xuecheng, H. Yu and C. Yicheng, "A High-Security and Low-Power AES S-Box Full-Custom Design for Wireless Sensor Network", *Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing*, Vol. 656, No. 8, pp. 2499–2502, Shanghai, 2007.

36. Hamalainen, P., T. Alho, M. Hannikainen and T. D. Hamalainen, "Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core", *Proceedings of the 9th Conference on Digital System Design*, Vol. 58, No. 12, pp. 577–583, Dubrovnik, 2006.

37. Kim, M., J. Kim and Y. Choi, "Low Power Circuit Architecture of AES Crypto Module for Wireless Sensor Network", *Proceedings of World Academy of Science, Engineering and Technology*, Vol. 8, No. 1, pp. 203–208, Dubai, 2005.

38. Bouesse, G., M. Renaudin, A. Witon and F. Germain, "A clock-less low-voltage AES crypto-processor", *Proceedings of the 31st European Solid-State Circuits Conference*, Vol. 384, No. 7, pp. 403–406, Grenoble, 2005.

39. Shang, D., F. Burns, A. Bystrov, A. Koelmans, D. Sokolov and A. Yakovlev, "High-security asynchronous circuit implementation of AES", *IEE Proceedings, Computers and Digital Techniques*, Vol. 153, No. 2, pp. 71–77, 2006.

40. Sun, K., X. Pan, J. Wang and J. Wang, "Design of A Novel Asynchronous Reconfigurable Architecture for Cryptographic Applications", *First International Multi-Symposiums on Computer and Computational Sciences*, Vol. 2, No. 5, pp. 751–757, Hangzhou, 2006.

41. Mui, E. N. C., *Practical implementation of rijndael S-Box using combinational logic*, http://www.xess.com/projects/Rijndael_SBox.pdf, 2007, accessed at May 2011.

42. Dogan, A., *Power Efficient FPGA Implementation of AES Algorithm*, M.S. Thesis, Istanbul Technical University, 2008.