CS 577: Project Report

Project Number :	P21
Group Number:	13
Name of the top modules:	KeyExpansion, Cipher, InvCipher
Link for GitHub Repo:	github.com/deepraj88/project21

Group Members	Roll Numbers
Sujoy Ghosh	160101073
Shubham Goel	160101083
Rishabh Jain	160101088
Balbir Singh	160108013

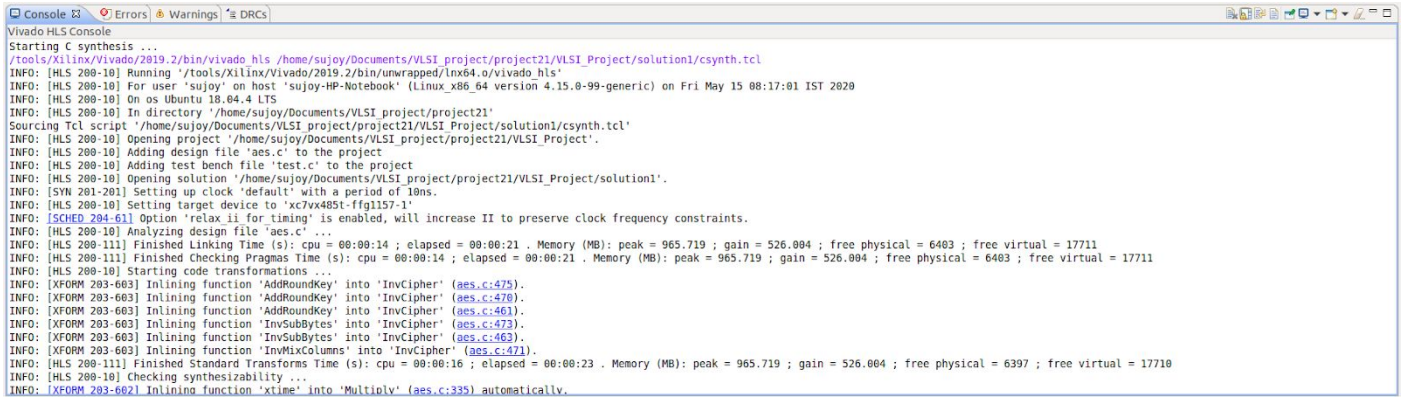
Date: 15.05.2020

INTRODUCTION

PHASE-1

1. Running the algorithm

1.1 Simulation screenshot



1.2 Synthesis screenshot

Synthesis Report for 'InvCipher'

General Information

Date:	Fri May 15 08:17:30 2020
Version:	2019.2 (Build 2704478 on Wed Nov 06 22:10:23 MST 2019)
Project:	VLSI_Project
Solution:	solution1
Product family:	virtex7
Target device:	xc7vx485t-ffg1157-1

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.680 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
86	86	0.860 us	0.860 us	86	86	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	200	-
FIFO	-	-	-	-	-
Instance	-	-	0	1488	-
Memory	8	-	0	0	-
Multiplexer	-	-	-	1713	-
Register	-	-	393	-	-
Total	8	0	393	3401	0
Available	2060	2800	607200	303600	0
Utilization (%)	~0	0	~0	1	0

All our loops ran a constant number of times. Hence, we don't see any '?' in the latency table.

1.3 C/RTL co-simulation screenshot

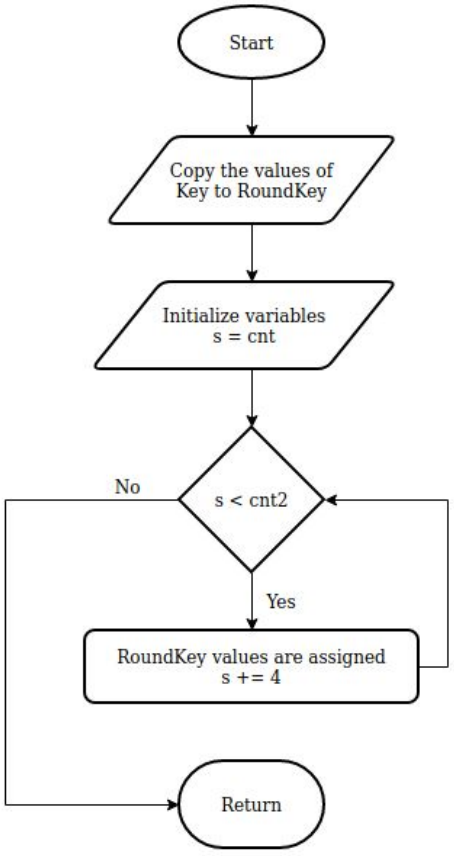
Cosimulation Report for 'InvCipher'

Result

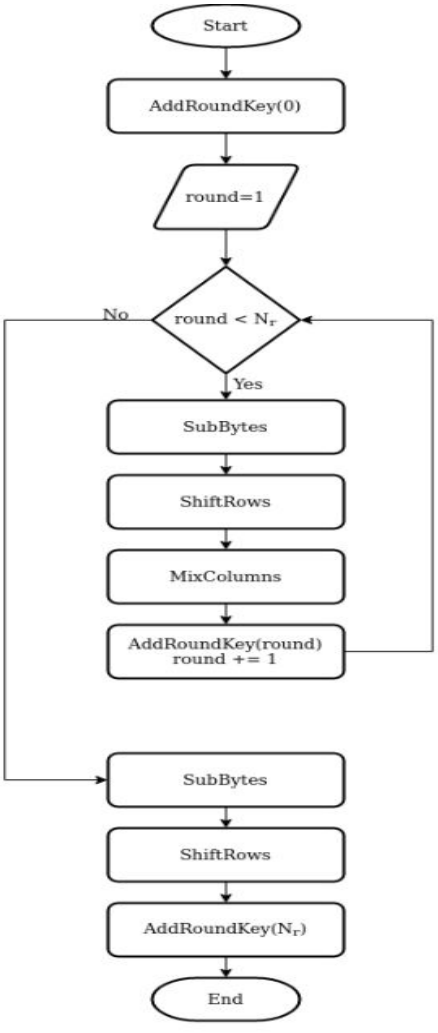
		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	86	86	86	87	87	87

2. Flowchart (Give the flowchart of the function used. Describe basic understanding of the algorithm)

We have three top modules to work with, namely KeyExpansion, Cipher, InvCipher. All these modules are present in aes.c file.



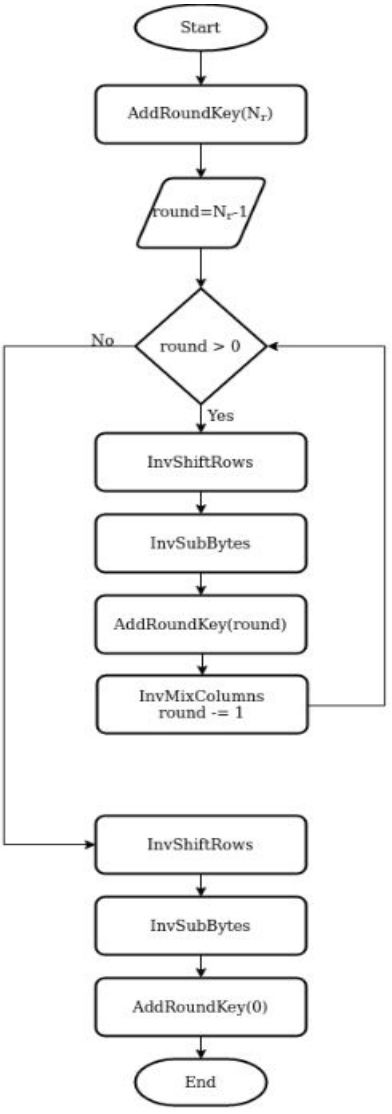
KeyExpansion Module



Cipher Module

The AES encryption algorithm defines numerous transformations that are to be performed on data stored in an array. The first step of the cipher is to put the data into an array -- after which, the cipher transformations are repeated over multiple encryption rounds.

The first transformation in the AES encryption cipher is substitution of data using a substitution table; the second transformation shifts data rows, and the third mixes columns. The last transformation is performed on each column using a different part of the encryption key. Longer keys need more rounds to complete.



InvCipher Module

3. Result

FPGA Part	Name of Top Module	FF	LUT	BRAM	DSP	Latency	II
xc7vx485tffg1157-1	KeyExpansion	121	543	1	0	254	-

Explanation:

This module is used extensively. The area utilisation of the module is less. However, the latency is high, owing to the fact that the BRAMs single port operation act as the bottleneck.

Problems and Solutions:

We need to decrease the latency, without significant increase in the area (LUTs/FFs) utilisation. We cannot go on decreasing the latency without taking note of LUT usage, since this file (aes.c) acts as the base for many other approaches. Hence, we should take care of this fact. So, we shall be using pipelining and unrolling techniques to make many statements run in parallel. We shall also use array reshape/partition to increase the number of ports of RAM access.

FPGA Part	Name of Top Module	FF	LUT	BRAM	DSP	Latency	II
xc7vx485tffg1157-1	Cipher	194	952	1	0	1316	

Explanation:

This module is also used extensively while converting normal text to cipher text. The area utilisation of the module is less. However, the latency is pretty high, owing to the fact that we have many function calls and the BRAMs single port operation act as the bottleneck.

Problems and Solutions:

We need to decrease the latency, without significant increase in the area (LUTs/FFs) utilisation. We cannot go on decreasing the latency without taking note of LUT usage, since this file (aes.c) acts as the base for many other approaches. Hence, we should take care of this fact. So, we shall be using pipelining and unrolling techniques to make many statements run in parallel. We shall also use array reshape/partition to increase the number of ports of RAM access. We shall also try to limit the number of operators using **ALLOCATION** technique

FPGA Part	Name of Top Module	FF	LUT	BRAM	DSP	Latency	II
xc7vx485tffg1157-1	InvCipher	809	3078	1	0	485	

Explanation:

This module is used extensively while converting cipher-text to normal text. The area utilisation of the module is more than the Cipher **module**. However, the latency is pretty high, owing to the fact that we have many function calls and the BRAMs single port operation act as the bottleneck.

Problems and Solutions:

We need to decrease the latency, without significant increase in the area (LUTs/FFs) utilisation. We cannot go on decreasing the latency without taking note of LUT usage, since this file (aes.c) acts as the base for many other approaches. Hence, we should take care of this fact. So, we shall be using pipelining and unrolling techniques to make many statements run in parallel. We shall also use array reshape/partition to increase the number of ports of RAM access.

We shall be using **Loop Unrolling, Pipelining, Function Inlining, Array Partition** and **Allocation** pragmas. Allocation pragma is usually used for area reduction, while others have impact on both area and latency. It depends upon the code that we are running on.

PHASE-2

The target FPGA board is **Virtex-7 board**

(Since no board was specified in the assignment initially, we used the default board provided while creating the project).

Target Clock = 10ns for all

Top Module- KeyExpansion

Benchmark	Type (Area/ Latency/ Both)	Resource Utilization				Latency		Major Optimizations
		LUT	FF	DSP	Block RAM	No of Clock cycle/ latency	Clock period (in ns)	
Baseline	--	543	121	0	1	254	5.328	----
Commit-1 KeyExpansion revisit	Latency	1635	75	0	4	130	8.344	**Applied ARRAY Partitioning/ Reshape** Array partitioning was applied to the module to increase the no. of ports for the array access. It leads to increase in area, as expected.
Commit-2 KeyExpansion revisit	Latency	16291	1366	0	1	23	8.239	**Pipelining allowed** Pipelining is introduced in the module to reduce the latency. But it increases the area a lot, Since it was applied in the method itself.
Commit-3 KeyExpansion revisit	Latency	16291	1366	0	1	23	8.239	**Code Cleanup** The code was made more cleaner by removing Common

								Subexpressions etc. It didn't help though.
Commit-4 Partition changed to reshape	Area	14898	1430	0	1	23	8.239	**Array Partition changed To Array Reshape** Everything was changed to Array Reshape so that it could reduce the Number of LUTs used. It did decrease the LUT count.
Commit-5 KeyExpansion Tradeoff between latency and area	Area	1851	322	0	0	43	8.430	**Array Reshape changed To Array Partition and Pipelining inside loop Loop Unrolling** We soon realised that Array Reshape was the reason behind so much LUT usage, using Analysis Tools. Hence, we changed to Array Partition and it helped to reduce the LUT usage by a factor of 7. We also used the loop unrolling and pipeline to maximize the con--currency.
Commit-6 KeyExpansion area reduction	Area	1587	452	0	0	63	8.299	**Code Rewrite and factor Reduction** The factor 16 was changed to 8 to reduce area. Also, the extra arrays were replaced by Variables, and the code was rewritten. All these increased the latency a bit, but also reduced the area. We also limited the number of ICMP instances to be used, which helped in further area reduction.

Commit-7 KeyExpansion area reduction	Area	1527	247	0	0	103	8.299	**Allocation limit on Select instances** It increased the latency But there was less area reduction. We checked many values, and found 22 to be the best of all.
Commit-8 KeyExpansion major changes	Latency + Area	1905	285	0	0	33	7.820	**Unroll factor increase and limit AND instance** Unrolling the loop more Helped to get more con--currency. But the area Util was also high. Many AND instances were present. So, we limit the number of AND instances to reduce the area.

Top Module- Cipher

Benchmark	Type (Area/ Latency/ Both)	Resource Utilization				Latency		Major Optimizations
		LUT	FF	DSP	Block RAM	No of Clock cycle/ latency	Clock period (in ns)	
Baseline	--	952	194	0	1	1316	4.927	----

Commit-1 AddRoundKey Analysis done	Latency	2809	1074	0	1	506	5.112	**Array Partition and Loop Unrolling (AddRoundKey method)** The loop is completely unrolled to increase parallelism. But to achieve that, we need to partition the array itself.
Commit-2 SubBytes Analysis done	Latency	2380	928	0	8	106	5.329	**Array Partition and Loop Unrolling (SubBytes method)** The loop is completely unrolled to increase parallelism. But to achieve that, we need to partition the array itself.
Commit-3 Cipher Analysis done	Latency + Area	1923	140	0	8	13	7.405	**Function Inlining Pipelining Code rewrite (MixColumns method)** Here, we do the function inlining for the subfunctions used in Cipher module. This lead to reduction in Both area and time. We also added Loop Pipelining, which decreased the latency further, but area util increased. We noted that the XOR the count was huge. MixColumns itself had 17 XOR in the loop. Hence, we rewrite the code to decrease the XOR to 13. It helped a lot in reducing the area consumption.

Here, we used **ALLOCATION pragma** on many values for different instances. But it was eventually **increasing both Latency and LUT/FF usage**. Hence, **its commit is not made.**

Top Module- InvCipher

Benchmark	Type (Area/ Latency/ Both)	Resource Utilization				Latency		Major Optimizations
		LUT	FF	DSP	Block RAM	No of Clock cycle/ latency	Clock period (in ns)	
Baseline	--	3078	809	0	1	485	6.780	----
Commit-1 InvCipher changed to multiply module	--	4378	809	0	1	485	7.821	Just changed the multiply to function as an experiment.
Commit-2 InvCipher corrected	Latency + Area	3001	546	0	1	458	7.050	**Function Inlining Array Partitioning Loop unroll** We did array partition to increase the number of ports, and hence parallelism. The loop was unrolled to allow better concurrency. (full unroll gave time error critical path length was greater than clock period) Function Inlining was done so that the code can do Some local optimisations.
Commit-2 InvCipher changed the InvMultiply module	Latency + Area	2305	237	0	8	48	6.390	**Loop Unroll (InvMixColumns)** The loop was completely unrolled to allow better concurrency and resource sharing.
Commit-4 InvCipher code rearranged	Area	2179	275	0	8	58	5.811	**Code Restructure Remove Extra XOR Inlining** The code in InvCipher

								was restructured to allow better resource sharing. This leads to reduction in area. We also removed extra XORs for optimizing area. Inlining was also used and it gave better latency at very small Increase of FFs.
Commit-5 InvCipher pipelined	Latency	2869	340	0	8	33	8.340	**Pipelining** The loop in InvCipher is pipelined to allow Better concurrency, leading to better latency, at the cost of extra area.

Here, we used **ALLOCATION pragma** on many values for different instances. But it was eventually **increasing both Latency and LUT/FF usage**. Hence, **its commit is not made.**