CS577-C Based VLSI DESIGN

BLOWFISH

Submitted by –

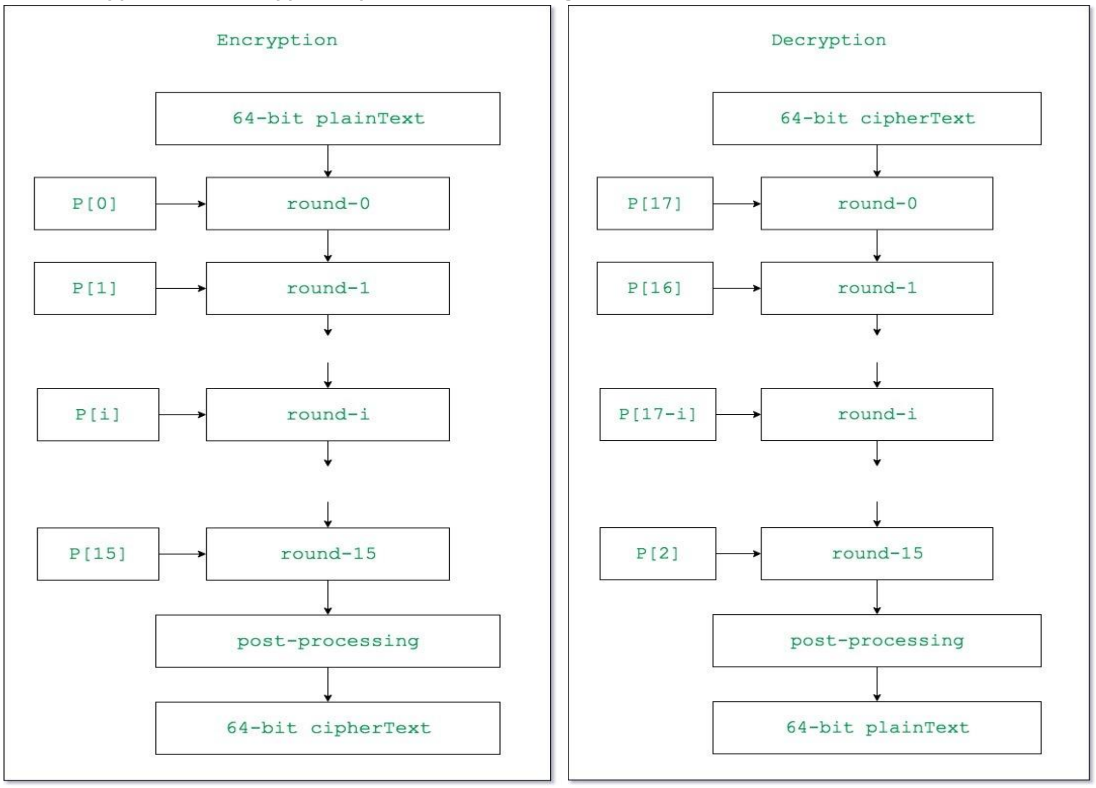
(Group no -10)

Group Members	Roll No.
Bhaskar Sharma	204101018
Farhan Mohammad	204101022
Harshwardhan Dubey	204101026
Kunal Wanikar	204101070

Link for Git-Hub Repo : https://github.com/kunalwanikar

Introduction

- **Blowfish** is an encryption technique designed by **Bruce Schneier** in 1993 as an alternative to DES Encryption Technique. It is significantly faster than DES and provides a good encryption rate with no effective cryptanalysis technique found to date. It is one of the first, secure block cyphers not subject to any patents and hence freely available for anyone to use.
- **The Encryption & Decryption process of this algorithm is as follows :**



Files Included –

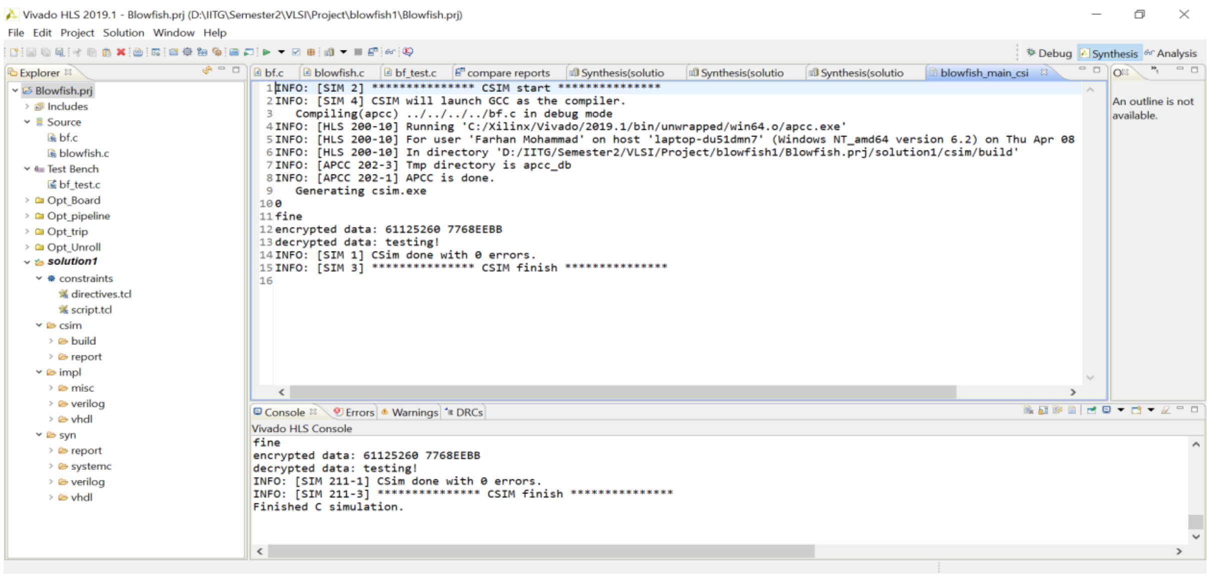
a) Source File -
   - bf.c
   - blowfish.c (To show the output on screen)

b) Test Bench –
   - bf_test.c

Running the algorithm using vivado -

1) C- Simulation :

- It is a pre-synthesis validation that the C program correctly implements the required functionality.
- Before synthesis, the function to be synthesized should be validated with a test bench using C simulation.
- Our test bench includes the **top function(blowfish_main())** that calls the function to be synthesized. C-simulation also shows the error if any, in our case it is showing the output as encrypted and decrypted data (testing!).The screenshot is mentioned below-



```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling(apcc) ../../../../bf.c in debug mode
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2019.1/bin/unwrapped/win64.o/apcc.exe'
INFO: [HLS 200-10] For user 'Farhan Mohammad' on host 'laptop-du51dmn7' (Windows NT_amd64 version 6.2) on Thu Apr 08
INFO: [HLS 200-10] In directory 'D:/IITG/Semester2/VLSI/Project/blowfish1/Blowfish.prj/solution1/csim/build'
INFO: [APCC 202-3] Tmp directory is apcc_db
INFO: [APCC 202-1] APCC is done.
   Generating csim.exe
0
fine
encrypted data: 61125260 7768EEBB
decrypted data: testing!
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

```
fine
encrypted data: 61125260 7768EEBB
decrypted data: testing!
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
Finished C simulation.
```

2) Synthesis :

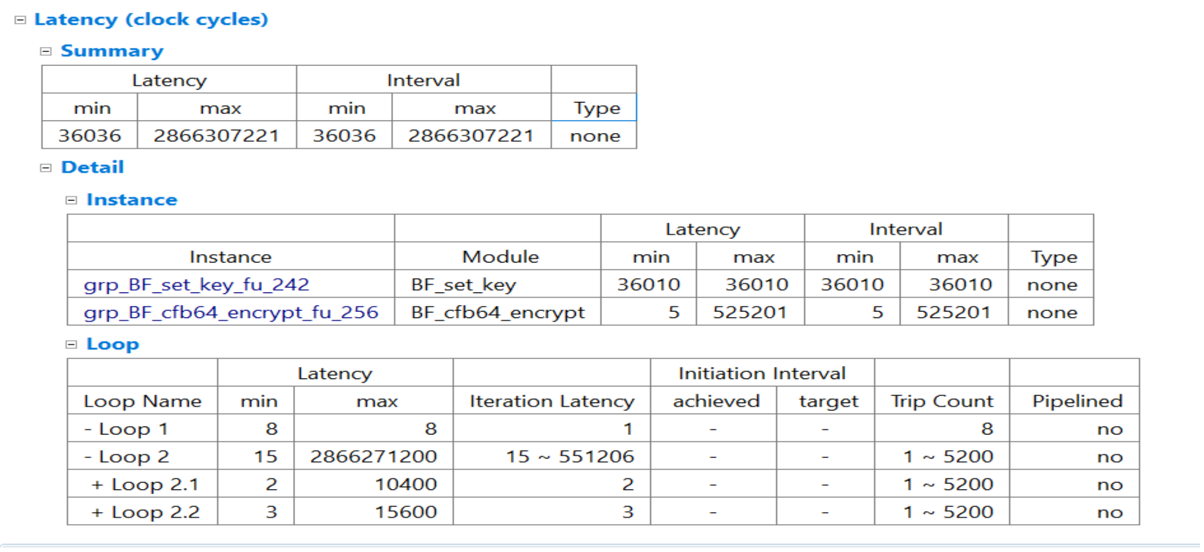
It is the process of transforming an RTL-specified design into a gate-level representation. The screenshots of synthesis of baseline is as follows :

- **Performance Estimate :**

⊟ **Timing (ns)**

⊟ **Summary**

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.750	1.25



⊟ **Latency (clock cycles)**

⊟ **Summary**

Latency		Interval		
min	max	min	max	Type
36036	2866307221	36036	2866307221	none

⊟ **Detail**

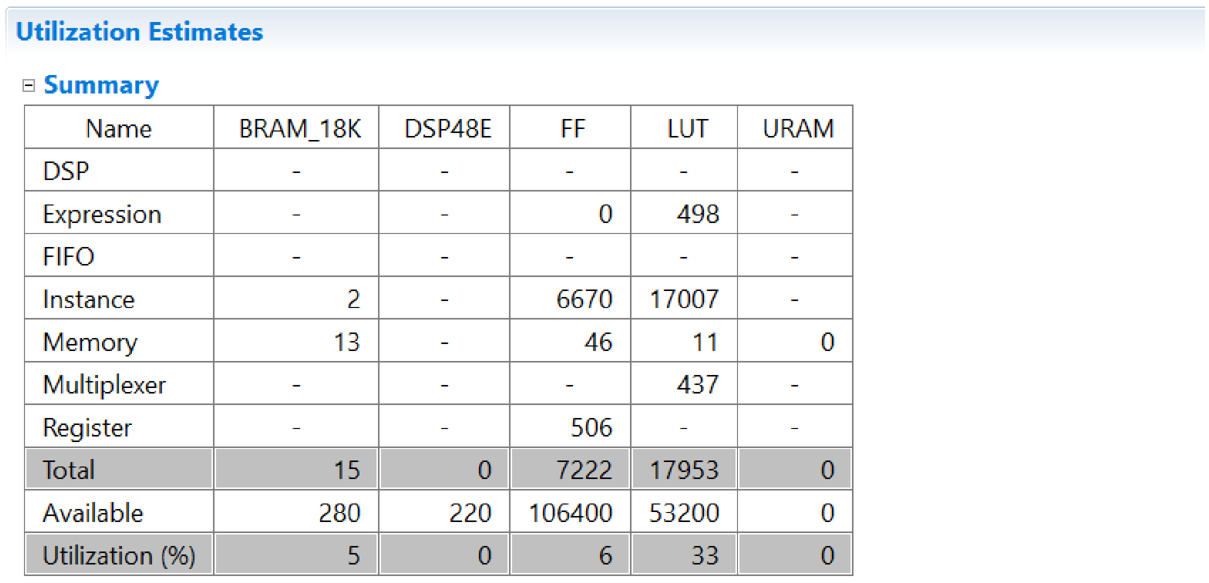
⊟ **Instance**

		Latency		Interval		
Instance	Module	min	max	min	max	Type
grp_BF_set_key_fu_242	BF_set_key	36010	36010	36010	36010	none
grp_BF_cfb64_encrypt_fu_256	BF_cfb64_encrypt	5	525201	5	525201	none

⊟ **Loop**

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	8	8	1	-	-	8	no
- Loop 2	15	2866271200	15 ~ 551206	-	-	1 ~ 5200	no
+ Loop 2.1	2	10400	2	-	-	1 ~ 5200	no
+ Loop 2.2	3	15600	3	-	-	1 ~ 5200	no

- **Utilization :**



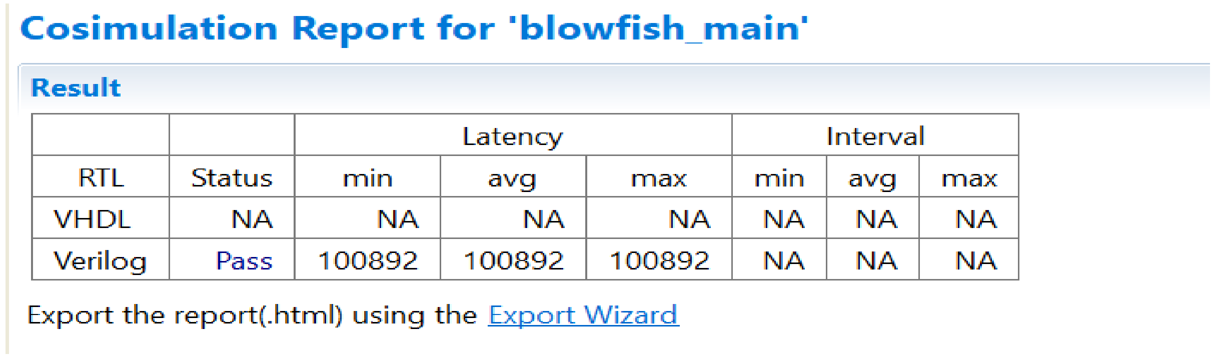
Utilization Estimates

⊟ **Summary**

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	498	-
FIFO	-	-	-	-	-
Instance	2	-	6670	17007	-
Memory	13	-	46	11	0
Multiplexer	-	-	-	437	-
Register	-	-	506	-	-
Total	15	0	7222	17953	0
Available	280	220	106400	53200	0
Utilization (%)	5	0	6	33	0

3) **C/RTL Co-Simulation Report :** It verifies that the RTL is functionally identical to the C source code.

a) **Co-Simulation report for BASELINE/OPT_LATENCY**



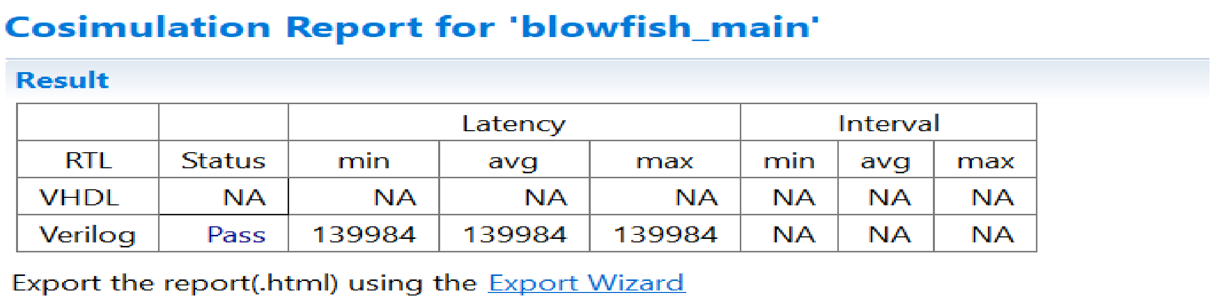
Cosimulation Report for 'blowfish_main'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	100892	100892	100892	NA	NA	NA

Export the report(.html) using the Export Wizard

b) Co-Simulation report for OPT_AREA



Cosimulation Report for 'blowfish_main'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	139984	139984	139984	NA	NA	NA

Export the report(.html) using the Export Wizard

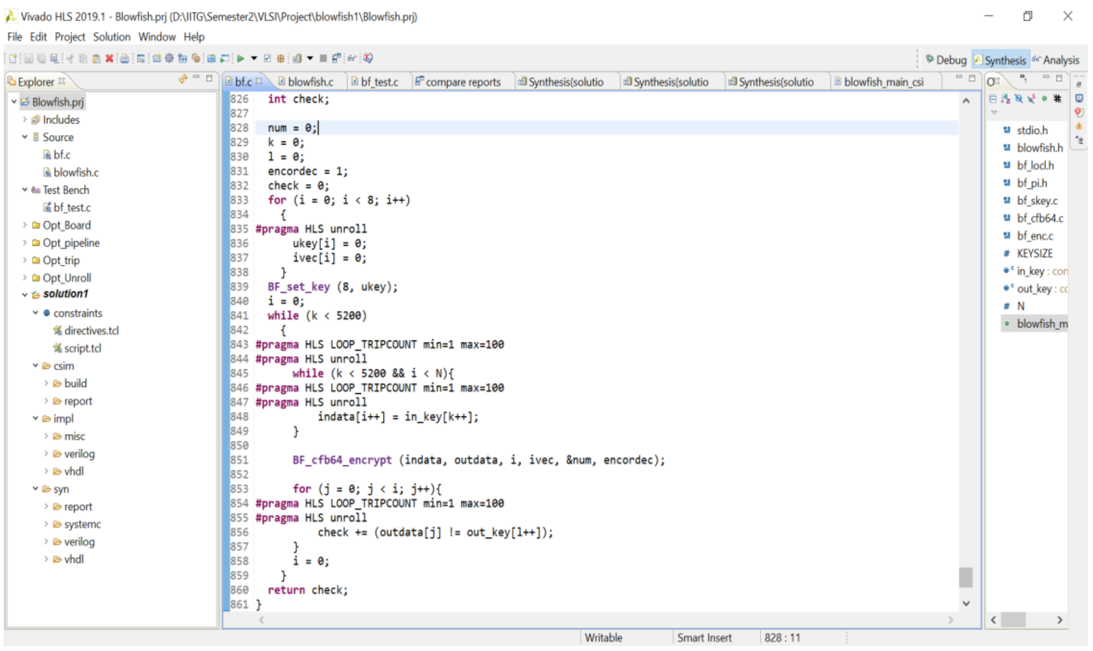
4) Result :

FPGA Board	Name of Top Module	FF	LUT	Latency
ZedBoard	Blowfish_main	7222	17953	2866307221

➤ **Here without any optimization we are getting very high latency and the number of resources (area) used are also high.**

Problems and it's solution –

1) **Removed ? mark –** We were getting symbol (?) in place of minimum and maximum latency. So we removed it using the **trip count.** Check the below attached screenshot-



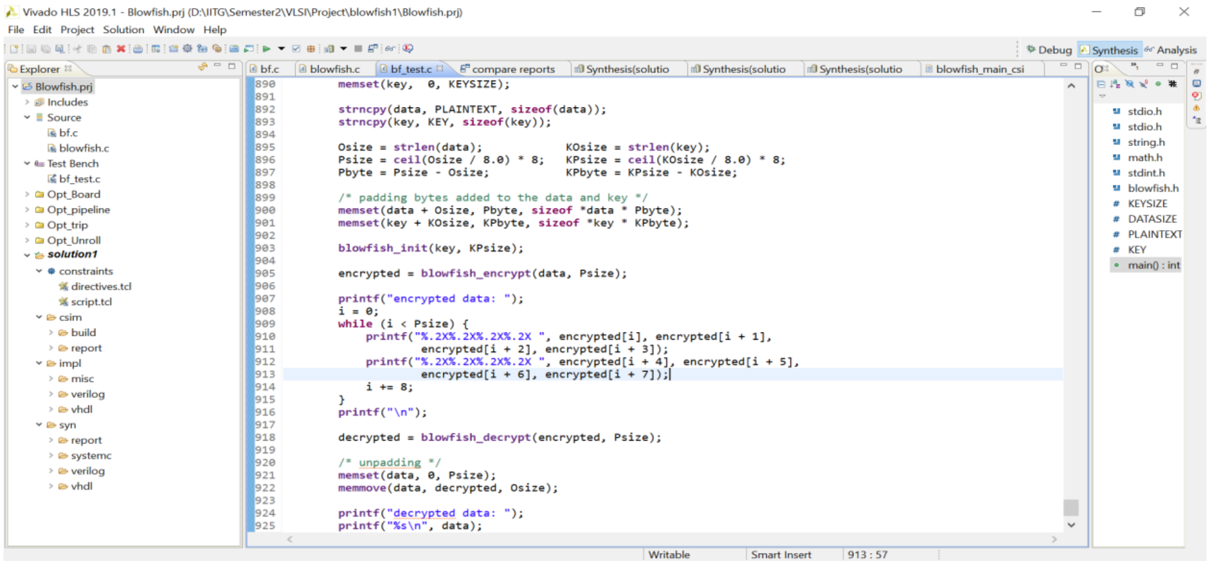
```c
  int check;

  num = 0;
  k = 0;
  l = 0;
  encordec = 1;
  check = 0;
  for (i = 0; i < 8; i++)
    {
#pragma HLS unroll
      ukey[i] = 0;
      ivec[i] = 0;
    }
  BF_set_key (8, ukey);
  i = 0;
  while (k < 5200)
    {
#pragma HLS LOOP_TRIPCOUNT min=1 max=100
#pragma HLS unroll
      while (k < 5200 && i < N){
#pragma HLS LOOP_TRIPCOUNT min=1 max=100
#pragma HLS unroll
        indata[i++] = in_key[k++];
      }

      BF_cfb64_encrypt (indata, outdata, i, ivec, &num, encordec);

      for (j = 0; j < i; j++){
#pragma HLS LOOP_TRIPCOUNT min=1 max=100
#pragma HLS unroll
        check += (outdata[j] != out_key[l++]);
      }
      i = 0;
    }
  return check;
}
```

2) **Encryption/Decryption –** We have written the function which shows the encrypted and decrypted output. The code is as follows :



```c
        memset(key,  0, KEYSIZE);

        strncpy(data, PLAINTEXT, sizeof(data));
        strncpy(key, KEY, sizeof(key));

        Osize = strlen(data);               KOsize = strlen(key);
        Psize = ceil(Osize / 8.0) * 8;      KPsize = ceil(KOsize / 8.0) * 8;
        Pbyte = Psize - Osize;              KPbyte = KPsize - KOsize;

        /* padding bytes added to the data and key */
        memset(data + Osize, Pbyte, sizeof *data * Pbyte);
        memset(key + KOsize, KPbyte, sizeof *key * KPbyte);

        blowfish_init(key, KPsize);

        encrypted = blowfish_encrypt(data, Psize);

        printf("encrypted data: ");
        i = 0;
        while (i < Psize) {
            printf("%.2X%.2X%.2X%.2X ", encrypted[i], encrypted[i + 1],
                    encrypted[i + 2], encrypted[i + 3]);
            printf("%.2X%.2X%.2X%.2X ", encrypted[i + 4], encrypted[i + 5],
                    encrypted[i + 6], encrypted[i + 7]);
            i += 8;
        }
        printf("\n");

        decrypted = blowfish_decrypt(encrypted, Psize);

        /* unpadding */
        memset(data, 0, Psize);
        memmove(data, decrypted, Osize);

        printf("decrypted data: ");
        printf("%s\n", data);
```

Pragma's Used

- **#pragma HLS inline region :** This applies the pragma to the region or the body of the function it is assigned in. It applies downward, inlining the contents of the region or function, but not inlining recursively through the hierarchy.
- **#pragma HLS unroll :** The UNROLL pragma transforms loops by creating multiples copies of the loop body in the RTL design, which allows some or all loop iterations to occur in parallel.
- **#pragma HLS loop_TRIPCOUNT min=1 max=5200** : The TRIPCOUNT pragma can be applied to a loop to manually specify the total number of iterations performed by a loop.
- **#pragma HLS pipeline** : The PIPELINE pragma reduces the initiation interval for a function or loop by allowing the concurrent execution of operations.

Table after Optimization :

Benchmarks	Type	Resource Utilization				Latency		Major Optimization
		LUT	FF	DSP	BRAM	Min Latency	Max Latency	
Baseline	Baseline	17953	7222	0	15	36036	2866307221	**No Optimization**
Optimization 1	Area	9517	4161	0	15	35998	2595901984	**Area(LUT,FF)**
Optimization 2	Latency	18171	6939	0	15	36037	1514322822	**Latency**

Optimization Explanation -

- **Here we have applied two optimizations on the Baseline (**We have used the board- **" ZedBoard Zynq Evaluation and Development Kit (xc7z020clg484-1)").**

1.) Optimization-1 (Area) :

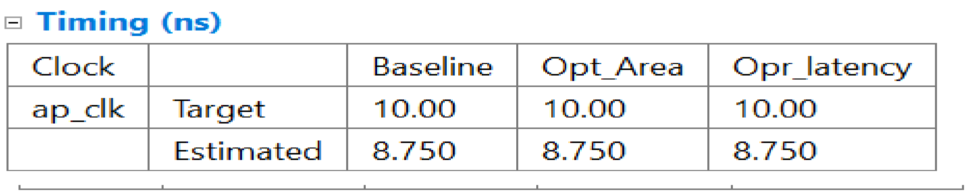
- After analyzing the code, we have found that many functions are repeatedly called and by default every call has a different memory map, in other words, LUT consumption is higher in this case and the flip-flops used are also high. So we have used the **INLINE** pragma to limit/optimize the LUT consumption from (17953 to 9517) and flip-flops(from 7222 to 4161).

2.) Optimization-2 (Latency) :

- To reduce the latency ,we have used the **PIPELINE and UNROLL** pragmas. The idea was used because there were many functions which have multiple nested loops operations with constant variables thus it created a scope of improvement for latency. Similar functions were identified and pipelined. Pipelining a function allows operations to be implemented in a concurrent manner. This has the potential to reduce requirement of number of clock cycles significantly.
- As a result our latency has decreased from (2866307221 to 1514322822) and LUT's have increased from (17953 to 18171).

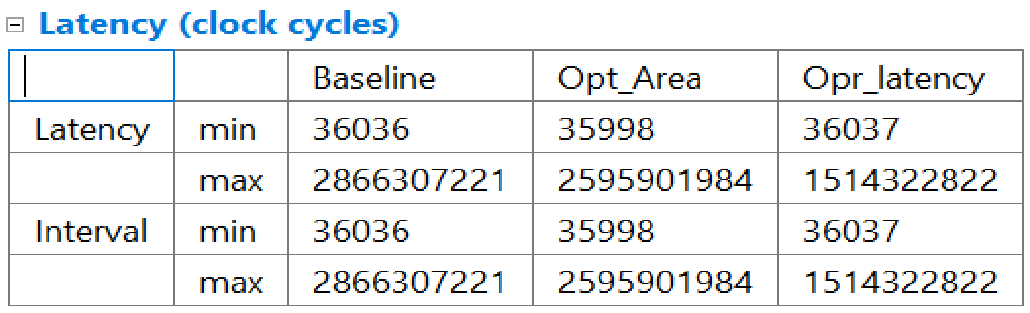
Comparison report

- We have compared the optimization-1(OPT_Latency) and optimization-2(OPT_Area) along with the baseline solution after enhancing some part of the code. Comparison report is as follows –



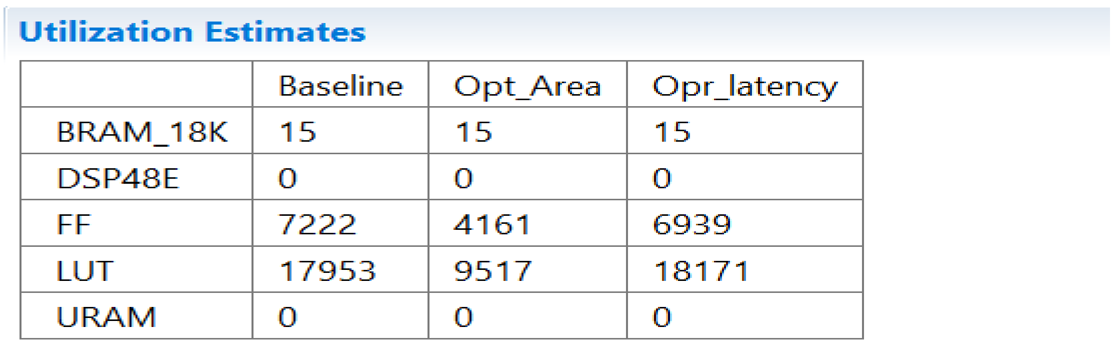
⊟ Timing (ns)

Clock		Baseline	Opt_Area	Opr_latency
ap_clk	Target	10.00	10.00	10.00
	Estimated	8.750	8.750	8.750



⊟ Latency (clock cycles)

		Baseline	Opt_Area	Opr_latency
Latency	min	36036	35998	36037
	max	2866307221	2595901984	1514322822
Interval	min	36036	35998	36037
	max	2866307221	2595901984	1514322822



Utilization Estimates

	Baseline	Opt_Area	Opr_latency
BRAM_18K	15	15	15
DSP48E	0	0	0
FF	7222	4161	6939
LUT	17953	9517	18171
URAM	0	0	0

Conclusion

- We had given the code for blowfish, where we have added the code for two things-
  I. To remove the ? mark.
  II. To show the output on screen for the encrypted and decrypted data.
- Initially we were getting very high latency (2866307221) and very high utilization of LUT's & flip-flop's.As we cannot reduce the latency and area simultaneously, we considered the baseline and optimized both the things one by one.
- To optimize the area we applied the **INLINE** pragma to reduce the number of LUTs **(From 17953 to 9517)** and Flip-Flops **(From 7222 to 4161)**.
- To optimize the latency we applied **PIPELINE** and **UNROLL**, the latency has been reduced from **2866307221** to **1514322822** and LUT got increased in this case from **17953** to **18171**.