Assignment Homework 4

Due date: Check Canvas

Student Version – "AES T-Tables (BRAM)"

Please team up with a fellow student and start working on the following tasks (100 pts total). In addition, you *must* consider the following:

- Vivado creates a default template including a comment block with:
  - "Company" – insert team name/number
  - "Engineer" – put team member names here
  - "Design Name" – based on task description (*do not leave empty*)
  - "Project Name" – Assignment 4
  - "Description" – based on task description (*do not leave empty*)
- To prepare your assignment for submission through Canvas, you must use (in Vivado): `File > Project > Archive`
- Preparing your HDL project through this process is especially important when working with IP cores such as DSPs and BRAMs!
- You work as a team. Please alternate between one student creating the testbench, the other student creating the implementation, or other aspects of the implementation.

We will start this homework assignment together through Zoom as a lab. Please try to follow along and ask questions if you get stuck somewhere. Your homework assignment includes all tasks mentioned here (even those, that you manage to complete during our joint lab time).

1 Task

In class, we learned a little bit about the Advanced Encryption Standard (AES)[*], a cryptographic algorithm for encryption/decryption of data, more specifically, a symmetric block cipher. While we do not have the time to appreciate its full academic beauty, we do appreciate all the work by previous researchers that made this achievement possible. Most notably, AES is an open standard that was the result of a transparent competition, breaking with the long tradition that the R&D of cryptographic algorithms was primarily a secretive government task.

Some students might feel that this assignment is more difficult than the others. However, aside from the fact that you are implementing AES, this is really just "a bunch of table lookups" combined with XORs. Please also note the following:

[*] `https://en.wikipedia.org/wiki/Advanced_Encryption_Standard`

- You will only be implementing AES encryption with a 128 bit key (resulting in 10 rounds of the AES). The input to your AES is the plaintext message, the output of AES is the encrypted ciphertext. You will not be implementing the decryption even though that would have been a nice team exercise (Student 1 implements encryption, Student 2 implements decryption).
- Even though AES is considered mathematically secure this is not true for all of its implementations. Implementations of the AES can suffer from timing-dependencies, or show specific patterns in the power consumption or electromagnetic emanation that makes it vulnerable to so called side-channel analysis that can extract the secret key (this is one of my research areas).
- Never implement your own AES in an industry/business context as this is likely not secure from an implementation point of view (unless you have multiple years of experience in this field and understand the potential security limitations better).

Your AES module will follow the basic AES entity/module description.

Listing 1: AES entity / module description.

```
module AES (
    input clk,
    input rst,
    input [127:0] plain, // plaintext
    input start,
    output reg [127:0] cipher, // ciphertext
    output reg valid
);
```

For properly designing your AES circuit, different perspectives are provided to you. The AES-128 algorithm structure is shown in Figure 1. As inputs, you need to consider the plaintext, as the the keys will all be stored in BRAM (because of this, they are not part of the module description). The output of the AES is the encrypted ciphertext block.

Figure 2 provides a more detailed view of the previous figure, as it shows the internals of the round function. Since you just want to get the job done, you ignore that SubBytes is an inversion in a Galois Field (you still spend a second to commemorate the great work by Galois). All the other parts of the round function – except the AddRoundKey – can also be merged into a lookup table. These lookup tables will also be stored in BRAM as indicated in the figure.

Consequently, "*the only thing*" that needs to be done is to implement a round-iterating structure (alternatively, a fully unrolled version with pipelining). Please note that implementing the AES using T-Tables is the fastest option both in software and in FPGAs, i.e., this is a state-of-the-art technique and when using BRAMs and DSPs for the XOR (AddRoundKey) then this is the fastest known method of implementing AES in an FPGA. This structure (BRAM and DSP) is illustrated in Figure 3 from [*]. However, to keep things simple, you are not required to implement the XOR with DSPs (=less work for you).

There is one piece missing though: selecting the right inputs from the AES state (128 bit). For a T-table implementation, we need to look at a "quarter-round" of the AES which implies, processing 32 bit as one. To just focus on the loop-iteration of the full AES round, you are tasked to process the four quarter-rounds in parallel (in AES terminology, this is processing all four columns C of the AES state). For a single column and quarter-round, this is shown in the following Figure 4.

[*] https://informatik.rub.de/wp-content/uploads/2021/11/phd_gueneysu.pdf

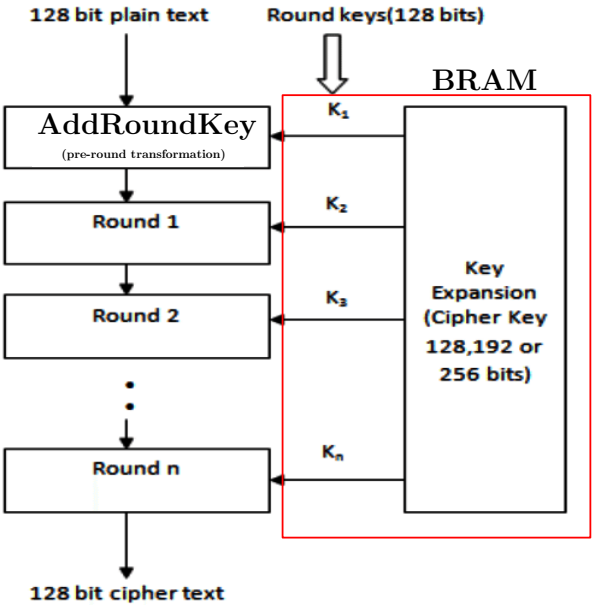

Figure 1: AES-128 (Very High Level Structure)

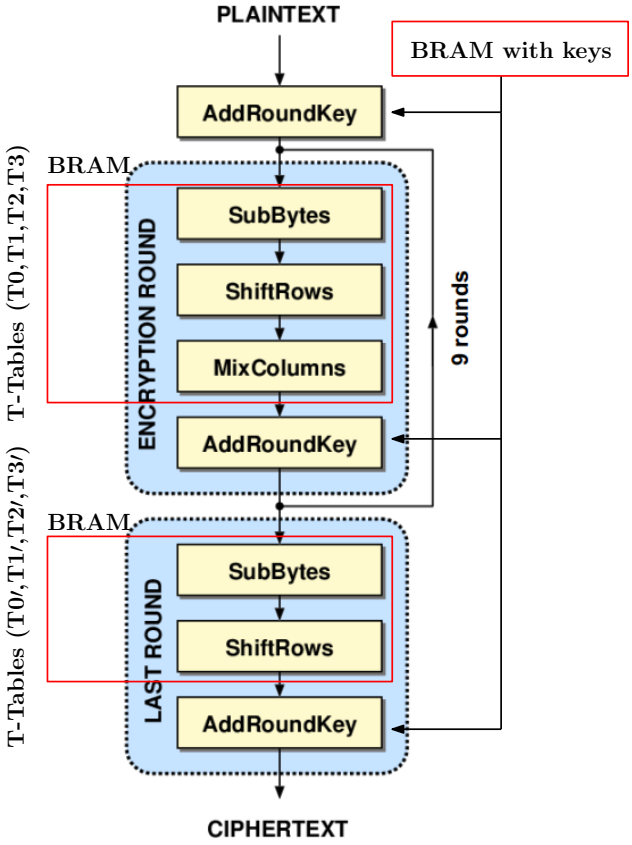

Figure 2: AES-128 (very high level structure with rounds)

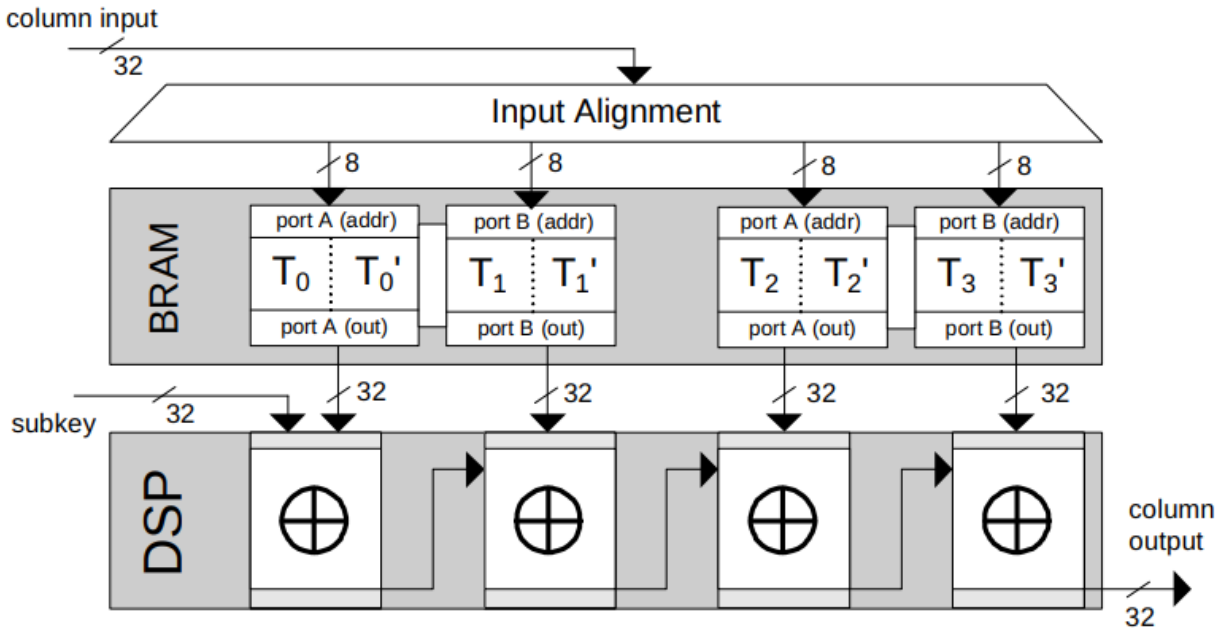

Figure 3: AES-128 (implementation-oriented structure of a "quarter-round"). Please note that this figure shows the idea of using T0/T1 and T2/T3 combined in a single BRAM and 8-bit address per port. You will receive initialization vectors for the BRAM for T0/T0', T1/T1', T2/T2', T3/T3' where the final round data is 'stacked' on top. Therefore, you will use a 9-bit address per port. The MSB of the address will be your signal `finalround`.

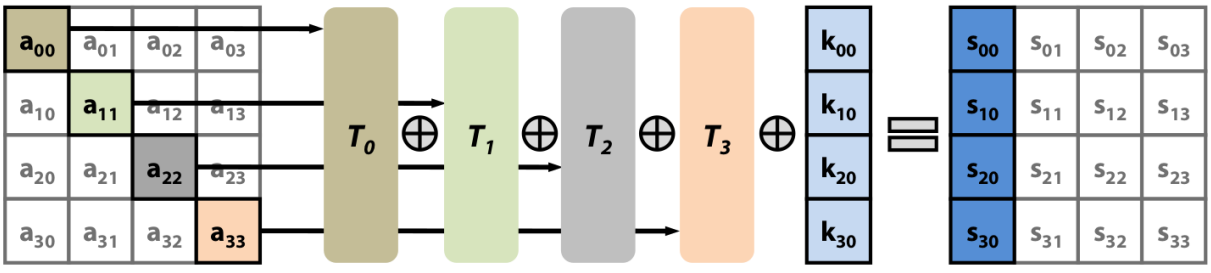

Figure 4: AES-128 (algorithm-oriented structure of a "quarter-round").

To simplify the design process, you quickly devise a strategy to implement it:

1. Start with reading all keys from the BRAM, i.e., write a counter that you use as address. Try reading the keys.
2. Afterwards, you should try to get the first round of the AES working.
3. Only then, you should try to make the loop-iterated design.

The grading will reflect these steps (i.e., should you be unable to get the full AES working).

2 Output

Create two (System-)Verilog files `AES.v` and `tb_AES.v`. The test vector to check that AES works is given to you. Again, please try to use good habits to write your code and the test benches. This includes creating a nice wave form configuration. In your submission, please include all modules you use, the test documentation, and ideally, the wave form configuration in the archive.

3 Hints

Please check the following simulation results. This is the result of the following test vectors:

- AES-128 key(s) from BRAM: `000102030405060708090a0b0c0d0e0f`
- AES-128 plaintext input: `00102030405060708090a0b0c0d0e0f0`
- Note how key XOR plaintext will result in: `00112233445566778899aabbccddeeff`
- The signals $\{e0, e1, e2, e3\}$ are the 32 bit inputs for each round
- The signals $\{a0, a1, a2, a3\}$ are the 32 bit outputs of the T-tables
- In other words: $\{a0, a1, a2, a3\}$ with additional AddRoundKey will form the new round input
- The final output is: `76d0627da1d290436e21a4af7fca94b7` (ciphertext)
- Alternatively, you can use the test vectors from FIPS-197 Appendix B [*].

The variable notation of using e and a follows this paper[†]. Please see Section 1 and 2 of this paper. You can stop reading on page 3 (before 2.1 Decryption).

[*] `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf`
[†] `https://saardrimer.com/sd410/papers/aes_dsp.pdf`

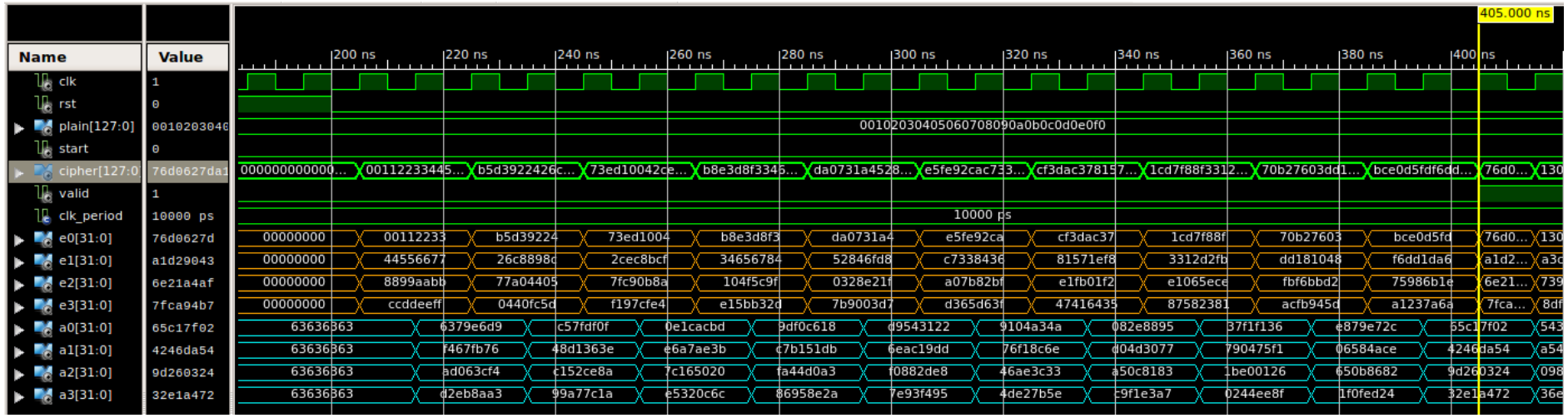

Figure 5: Simulation results.