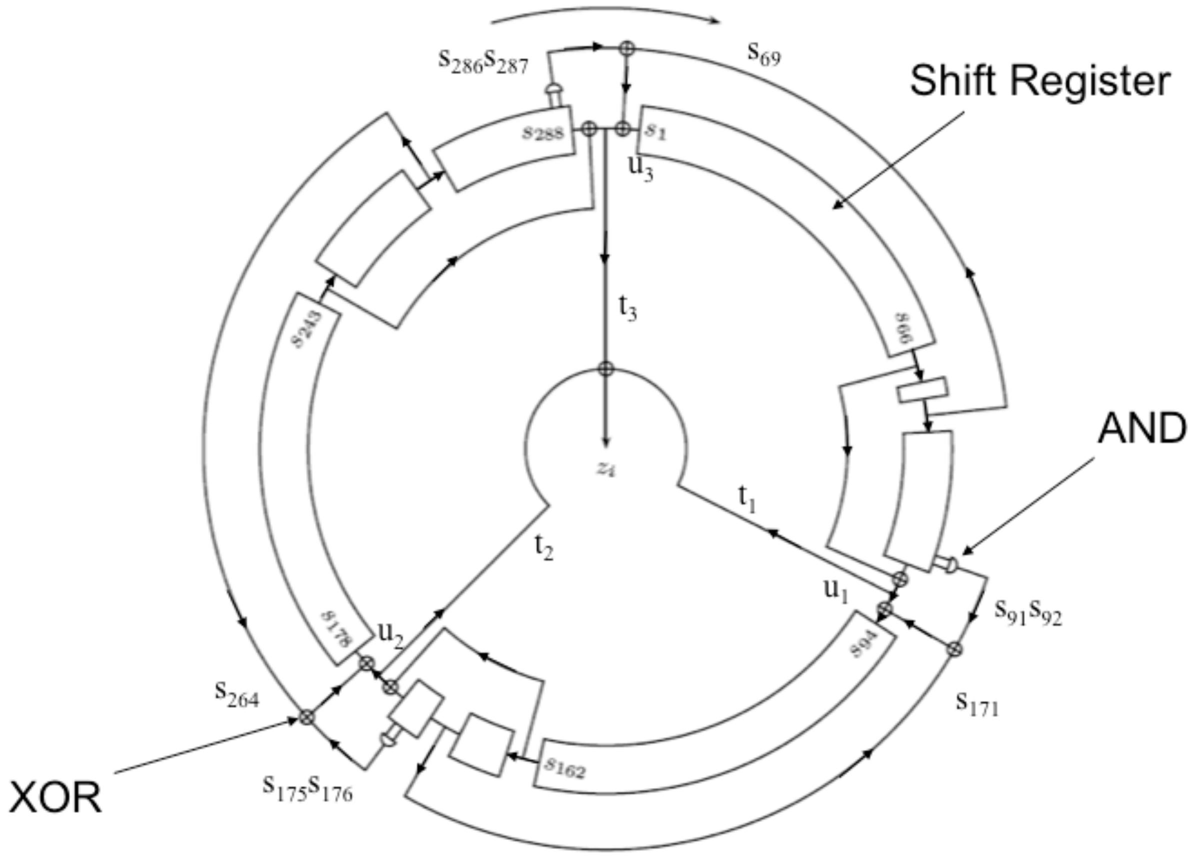
Experiment 2

Part 1 (5 points)

You are to implement the stream cipher Trivium. The cipher is described in detail in http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf

The following block diagram and pseudocode describe the keystream generator part of the cipher. The pseudocode describes also the way of calculating ciphertext bits based on the message and keystream bits.



```
for i = 1 to N do
  t1 ← s66 + s93
  t2 ← s162 + s177
  t3 ← s243 + s288
  zi ← t1 + t2 + t3
  ci ← mi + zi
  u1 ← t1 + s91 · s92 + s171
  $u2 \leftarrow t2 + s175 \cdot s176 + s264$
  $u3 \leftarrow t3 + s286 \cdot s287 + s69$
  $(s1, s2, ... , s93) \leftarrow (u3, s1, ... , s92)$
  $(s94, s95, ... , s177) \leftarrow (u1, s94 , ... , s176)$
  $(s178, s279 , ... , s288) \leftarrow (u2, s178 , ... , s287)$
end for
```

+ represents xor, N is the number of bits of the message,

mi denotes an i-th bit of the message, ci – an i-th bit of the ciphertext, zi – an i-th bit of the keystream.

s1-s288 are the output values of the individual bits in the shift register. The last 3 lines within the for loop in the above pseudocode represent the actual shifting.

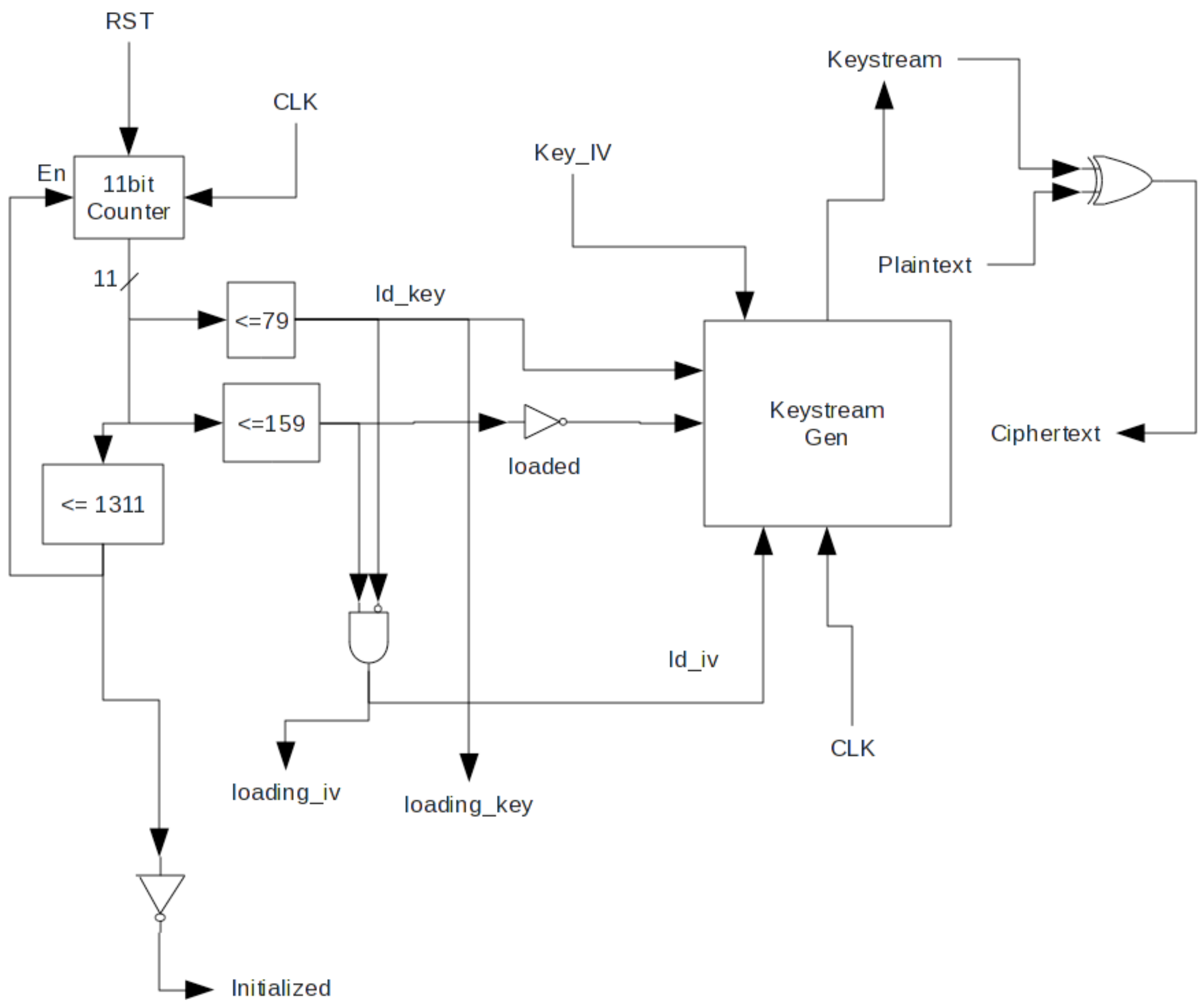
Everything in the loop occurs every clock cycle.

In order to initialize the circuit, the first 80 bits starting at s1 are loaded with the key. The IV is loaded into another 80 bits starting from s94 to keep the keystream unique, and the remaining bits are 0 except for 286-288 which are 1. Then, pseudocode above is run for 4 complete cycles (288*4 clock cycles).

You are to use 0x00000000000000000000000000000000 for the test key, IV, and plaintext for initial testing. The expected ciphertext for this should be 0xFBE0BF265859051B517A2E4E239FC97F.

http://www.ecrypt.eu.org/stream/triviumpf.html provides a link to a reference implementation in C, which you should use to generate additional test vectors.

The interface and control unit of the circuit you are to build are shown below:



The testbench should first send the key, then send the IV based on the loading_key and loading_iv signals, via the key_iv signal 1 bit at a time.

The signal named 'loading' determines whether or not to enable all registers or only the ones enabled via ld_key or ld_iv . It also controls muxes that are to be placed between the xor gate before s1 and s94 in the diagram above (not shown) in the diagram to load the key.

Part 2 (1 bonus point)

Redesign the above circuit, so it generates two bits of an output in a single clock cycle, as described in the lab slides.

Part 3 (1 bonus point on top of Part 2, 2 bonus points without separate Part 2)

Redesign the above circuit, so it generates k bits of an output in a single clock cycle, where k is a divisor of 288, smaller or equal to 64. The value of k should be passed to the design entity as a generic.

Tasks and Deliverables:

As a part of the design process for the dataflow description:

1. Draw a complete block diagram of your circuit, including components necessary to switch between initialization mode and encryption mode.
2. Translate your block diagram into a synthesizable VHDL code.
3. Develop a comprehensive testbench for your circuit, and test it using at least two different test vectors.

In the lab report include:

1. Complete block diagram of your circuit, including the names of all signals and ports used in your VHDL code.
2. VHDL source codes for the circuit description (electronic versions submitted using Blackboard)
3. Testbench and waveforms from the functional simulation **using ModelSim** (electronic versions submitted using Blackboard).

Important Dates

	Monday section	Tuesday section	Wednesday section	Thursday section
Hands-on Session and Introduction to the Experiment	02/01/2010	02/02/2010	02/03/2010	02/04/2010
Demonstration and Deliverables Due	**02/15/2010**	**02/16/2010**	**02/17/2010**	**02/18/2010**