Rabbit Cipher Implementation

For this exercise I have implemented the Rabbit cipher encryption algorithm in C#. I followed the specification in the original paper. I also used some resources from wikipedia. The process is pretty straight forward. I followed the steps in the paper implementing each part.
The code is organized in a few classes:

- Rabbit - The main class that contains the encryption logic and holds the current state (since this is a stream cipher). It is used to encrypt a block of plaintext into ciphertext. Takes the bytes of the plaintext and returns the bytes of the ciphertext. Block size is 128bit.
- State - A class that encapsulates the state of the Rabbit Algorithm. This class holds the 8 state variables, the 8 counters and contains most of the logic for the whole algorithm. There are initialization methods for the state variables, counters and configuration with the initialization vector. Also this class contains the logic for iterating the state.
- RabbitUtils - This class contains a few helper methods that I required for the project. Such as converting a byte array to a hex string and the inverse. Also left and right bitwise rotation.
- QByte - A data structure representing a 4 byte variable. Helper class that aids in the containment of the state variables.

Implementation

The algorithm consists of a few stages. The first stage is the **Key Setup.** In this stage the 8 state variables and 8 counter variables are initialised from the key. This is Very simple and required dividing the key into 2 byte sections and concatenating them in a specific order for each 4 byte variable and counter.
Next comes the **IV Setup.** This stage is not required and can be skipped, but if an initialization vector is provided it provides a way to manipulate the counters in order to make retrieval of the key more complex. This stage is also simple since it only requires XOR-ing the current values of the counter variables with different sections of the 64bit initialization vector. The exact scheme can be found in the original paper that I have linked above.
State Iteration Is the next stage. This is a scheme that describes how in each iteration the state of the algorithm changes because it is a stream cipher and must evolve with each iteration. The scheme is simple and it includes a few nonlinear combinations of the previous values of the state variables and the current value of the counter variables. This is again very well described in the paper.

In a separate stage the iteration of the counters is described. I will call this section the **Counter Iteration**. This section explains how each counter is incremented based on a few predefined constants. This continues and a form of overflow (carry) bits are involved in order to increase the minimum cycle period of the counter variables. This stage was the simplest for me because each counter is increased by a constant and the carry bit if present.
Finally after each iteration (apart from a few in the initialization part) we can generate a streamkey that can be used to encrypt a block of plaintext. The generated stream key is 128bits long. It is acquired by XOR-ing sections of the 8 state variables and concatenate them together. Again a pretty simple process, just care has to be given to the order of indices.
With this streamkey we are able to encrypt a 128bit block of plaintext by XOR-ing it with the streamkey. Because the XOR operation is reversible, the same stream key is used for decryption. After a block is encrypted, the state and counter variables should be iterated using the schemes described in **State** and **Counter Iteration.**