6.375: Complex Digital System Design
Final Proposal, High-level design and Test Plan

Lauren De Meyer - Candace Ross

1 Lightweight cryptography & SPECK

In the last few years, the need for small devices with very low computing power has awoken an interest for lightweight cryptography. Consider for example wireless sensors and the rapidly growing number of Internet of Things devices, becoming smaller and smarter.

SPECK [1] is a family of lightweight block ciphers that was introduced by the NSA in 2013. It was designed for flexibility and optimized for hardware implementations. Its round function only requires modular addition (A), rotations (R) and exclusive OR (X). The use of these three efficient operations is popular and the schemes that use them are called ARX ciphers. The round function is shown in Figure 1.

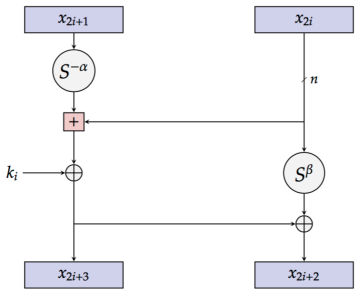

Figure 1: The SPECK Round function

One block of plaintext or ciphertext is $2n$ bits long with n the word size. As a family of block ciphers, SPECK allows for different choices for n, ranging from 16 to 64. For each choice of word size, there are one or two possible key sizes mn. For each pair of parameters (n, m), the designers have specified the number of rounds to use and the rotation parameters α and β (See Table 1).

The round function depicted in Figure 1 is applied T times, starting with x_1 and x_0 respectively the most significant and least significant n bits of a plaintext block.

$$R_k(x, y) = \big((S^{-\alpha}x + y) \oplus k, S^{\beta}y \oplus (S^{-\alpha}x + y) \oplus k\big)$$

Each round transforms (x_{2i+1}, x_{2i}) to (x_{2i+3}, x_{2i+2}) with R_{k_i} and after the last round, we obtain the 2 words (x_{2T+1}, x_{2T}) which form the ciphertext block.

The round keys k_i are determined from the master key K by a key schedule, which requires exactly the same hardware as the round function. We split the key K in its m words:

$$K = (l_{m-2}, \ldots, l_0, k_0)$$

word size n	key words m	block size $2n$	key size mn	α	β	# rounds T
16	4	32	64	7	2	22
24	3	48	72	8	3	22
	4		96			23
32	3	64	96	8	3	26
	4		128			27
48	2	96	96	8	3	28
	3		144			29
64	2	128	128	8	3	32
	3		192			33
	4		256			34

Table 1: Parameters for SPECK

The first round key k_0 is thus the least significant word of K. Subsequent keys are calculated as follows:

$$l_{i+m-1} = (S^{-\alpha} l_i + k_i) \oplus i$$
$$k_{i+1} = S^{\beta} k_i \oplus l_{i+m-1}$$

This is equivalent to applying the round function with round key i:

$$(l_{i+m-1}, k_{i+1}) = R_i(l_i, k_i)$$

For decryption, we must use the inverse of the round function

$$R_k^{-1}(x, y) = \big(S^{\alpha}(x \oplus k - S^{-\beta}(x \oplus y)), S^{-\beta}(x \oplus y)\big)$$

and we must invert the order of the round keys.

2 High-level design

Apart from the fact that it uses the same hardware, another important aspect of the key schedule is that it can be executed "on the fly". There is no need to generate all round keys before the start of encryption. During each round of encryption, we can calculate the next round key in parallel. As a result, we only need to keep the current key k_i in memory and don't need to remember the complete array of round keys. We do however need a register for the array l.

When many blocks are encrypted with the same key, it might seem unefficient to recalculate the round keys for every encryption. The alternative however is to store all round keys (Tn bits) in memory. We prefer only storing the encryption key (mn bits) as the cost of the round key calculation is negligible: the hardware requirements of the round function are minimal and as the key schedule is performed parallel to encryption, we don't lose any time efficiency.

An essential property of a lightweight cipher is its low-area design. We will therefore opt for a folded pipeline rather than a linear pipeline. For many lightweight applications, throughput is not the top priority. The pipeline will contain the hardware for two round functions such that we can perform encryption and key schedule in parallel. In addition,

we will need a register for the array l, inputFIFO's for the plaintext and key and and outputFIFO for the ciphertext.
The implementation will use polymorphism to allow for different choices of parameters and we will choose one or two specific sets for synthesis on the FPGA.

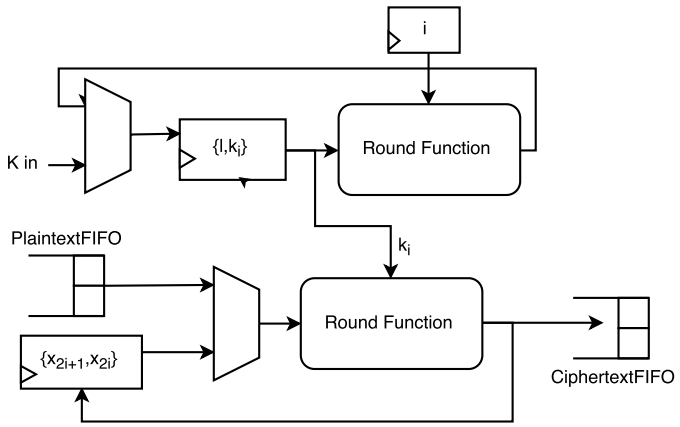

Figure 2: High level design

3 Why embedded cryptography?

Why is it useful to create cryptographic hardware instead of using software implementations? Firstly and most obviously, a dedicated hardware implementation achieves better performance. More importantly, a separate cryptography processor creates an explicit physical barrier around sensitive information. Cryptographic software performed in an environment with other applications running concurrently can leak information through for example timing variations or the cache. It is better for secret keys to be handled in their own environment with a private cache. Furthermore, many devices in the Internet of Things have limited resources or even no generalized core at all, in which case cryptography in hardware is the only option.

4 Test Plan and Interaction with Host processor

We have a working C implementation and a set of test vectors to check for correctness from the original paper [1]. Since the paper only provides one test for each parameter set, we can use the reference implementation to generate more test vectors for the parameter sets that we use on the actual FPGA.
We use a testing infrastructure similar to that of the Audio Pipeline, using Sce-Mi. A software test bench in c++ feeds the test vectors to port proxies and creates two outputfiles (encryption results and decryption results). We use these to compare the received encryptions to our reference ciphertext and to check that the decryption results match the initial plaintexts. We wrap our SPECK implementation in the `SceMiLayer` to connect it to the Sce-Mi ports. There are three types of communiation between the host processor and the FPGA:

1. Host processor sets key

2. Host processor passes one block of plaintext or ciphertext
3. FPGA passes output to host processor

While there are two separate hardware modules for encryption and decryption, we use the same in- and outputports for both. Our 'Device Under Test' (DUT) combines the two modules and decides which one to use based on an enumeration flag that is sent with the inputs.

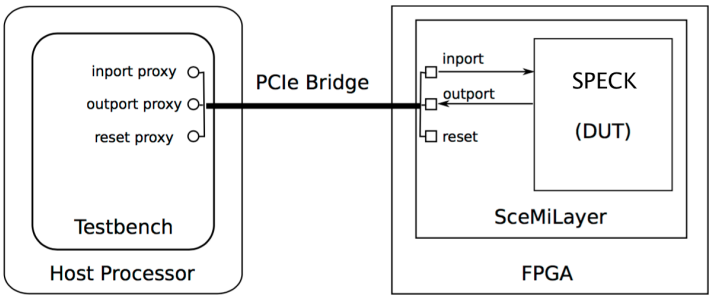

Figure 3: Test infrastructure with Sce-Mi

5 Microarchitectural Description

On the FPGA, our design consists of two modules. The modules, one for encryption and one for decryption, have identical interfaces and nearly identical implementations. They only differ in the definition of the round function (see Figure 2). As their names imply, the encryption module receives plaintext and transforms it into ciphertext. The decryption module receives ciphertext and produces plaintext.

5.1 Interface

SPECK can have a variable word size n, key size mn and number of rounds used for encryption T. The parameters (n, m, T) are used as numeric types for the `EncryptDecrypt` interface, which is implemented by both modules. The interface has three methods:

1. an `Action` method `setKey()` for setting the encryption/decryption key
2. an `Action` method `inputMessage()` for receiving the block of plaintext or ciphertext
3. an `ActionValue` method `getResult()`, returning the encryption/decryption result

These methods were chosen such that a user must only set his key once, when encrypting or decrypting multiple blocks.

5.2 Implementation

Each module has one rule implementing the folded pipeline shown in Figure 2. In every cycle, the round function is executed twice (once for the plaintext/ciphertext block and once for the keys) and the round indicator i is incremented. In the first round, when $i = 0$, the plaintext/ciphertext is taken from the inputFIFO instead of a register. In the last round, when $i = T - 1$, the result is put in the outputFIFO, the inputFIFO is dequeued, the round indicator i is reset to 0 and the round key k_i is reset to k_0.

In order to avoid conflicts with the `setKey()` method (both are writing to the key registers k_i and l), we introduce mutually exclusive guards on the two, using the state of the inputFIFO. It is only logical, that the key should only be allowed to changed when the inputFIFO is empty.

The `setKey()` method receives a key or mn bits or m words. The last three words are stored in registervector l. The round key is initialized with the first word k_0, which is also stored in a separate register for reuse of the key.

The `inputMessage()` and `getResult()` methods are straightforward. They respectively receive a block to enqueue into the inputFIFO or dequeue and return the first element of the outputFIFO. They don't conflict with any other method or rule.

6 Synthesis

We synthesized for the parameterset $(n, m) = (24, 4)$. With a clock period of 10ns, we obtain a critical path with worst negative slack equal to 2.29ns. Both the decrypt and encrypt module require a total of 941 flip-flops. There are 1253 LUT's for the decrypt module and 1263 for encryption. The current folded implementation achieves a throughput of 33 333 48-bit blocks per second ($\Leftrightarrow$ 1.6Mbps).

6.1 Planned exploration

Next, we want to unfold the design by including more than one round of encryption per cycle and see how this affects area and throughput. However, unlike with for example an FFT implementation, unfolding the design will not come without extra overhead costs. As can be seen in Table 1, the number of rounds used for Speck is seldom a power of 2 and even frequently odd. This means that apart from the repeated hardware from unfolding, we will need extra control and multiplexers to bypass some round functions in the last cycle.

References

[1] R. Beaulieu et al. The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. `http://eprint.iacr.org/`.