A Pipelined Camellia Architecture for Compact Hardware Implementation

Elif Bilge Kavun, Tolga Yalcin
Department of Cryptography
Institute of Applied Mathematics, METU
Ankara, Turkey
e-mail: e168522, tyalcin@metu.edu.tr

Abstract—In this paper, we present a compact and fast pipelined implementation of the block cipher Camellia for 128-bit data and 128-bit key lengths. The implementation is suitable for both Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC) platforms, and is targeted for low area and low power applications. To obtain a compact design, pipelining principles are exploited and platform specific optimizations are made. The design requires only 321 slices with a throughput of 32.96 Mbps based on Xilinx Spartan-S XC3S50-5 chip and 4.31K gates with a throughput of 81 Mbps based on 0.13-μm CMOS standard cell library.

Keywords-Camellia; block cipher; FPGA; ASIC; efficient implementation; cryptography; cryptographic hardware

I. INTRODUCTION

The 128-bit block cipher Camellia was developed by NTT (Nippon Telegraph and Telephone Corporation) and Mitsubishi Electric Corporation in 2000 [1]. Camellia was chosen as a recommended algorithm by NESSIE (New European Schemes for Signatures, Integrity and Encryption) [2] project in 2003 [3] to protect information. Camellia algorithm was selected with three other block ciphers: MISTY1 [4], SHACAL-2 [5] and AES (Advanced Encryption Standard) [6].

Camellia is also included in the list of cryptographic techniques selected by CRYPTREC (Cryptography Research and Evaluation Committees) [7] for Japanese e-Government systems. The algorithm has also been submitted to other evaluation projects and standardization organizations such as ISO/IEC JTC 1/SC 27 [8], IETF (Internet Engineering Task Force) [9] and TV-Anytime Forum [10]. It was certified as the IETF standard cipher for XML security URIs [11], SSL/TLS cipher suites [12] and IPSec in 2005 [13].

A compact and efficient hardware implementation of Camellia is attractive for low-cost embedded applications, which need low hardware complexity while high speed is not important.

In this paper, we propose a pipelined Camellia architecture utilizing an 8-bit datapath. The entire data path including a key scheduler is described.

The paper is organized as follows: Section II briefly describes the structure of Camellia algorithm. Section III presents a compact and fast pipelined Camellia design. Section IV is the hardware performance analysis for ASIC and FPGA implementations. Section V is the conclusion.

II. THE STRUCTURE OF CAMELLIA

Camellia [1] is a block cipher which processes data blocks of 128-bit with cipher key lengths of 128/192/256-bits. In our implementation, we use a 128-bit key.

The Camellia algorithm has an 18-round Feistel network structure with two 64-bit XOR operations before the first round and after the last round, which are called pre-whitening and post-whitening, respectively. After 6th and 12th rounds, FL and FL functions are inserted to introduce some non-regularity across rounds. The block diagram of the algorithm for 128-bit secret key and the inner 6-round structure can be seen in Figure 1. The encryption and decryption are done in the same way. There are no inverse functions. Only the subkeys should be presented in reverse order.

III. COMPACT CAMELLIA ARCHITECTURE

In our study, our main goal is to design a compact Camellia core with highest possible throughput, suitable for both FPGA and ASIC implementation. A fully parallel implementation requires 128-bit register to store the internal state, eight S-Boxes, and several logic gates to implement the other Camellia functions. While 128-bit memory is crucial to hold the internal state, choices of...
implementation of S-Boxes and permutation blocks give freedom to optimize the design for area or speed or both. In an area efficient design the number of S-Boxes can be reduced to one at the expense of a complex multiplexing logic and loss of throughput.

Use of a RAM seriously limits the pipelining capability of a design, resulting in a low throughput. For example, the design in [14] requires 875 clock cycles to complete all 18 rounds of a Camellia run as well as the 4 rounds required for the initial key scheduling. This yields a processing rate of 2.48 cycles per byte, whereas the theoretical limit is 1 cycle per byte for an 8-bit datapath.

In our approach, we use flip-flops to store the internal state, keys and temporary data. In order to minimize routing, we organize all flip-flops as 8-bit wide shift registers. As seen in Figure 2, the maximum data traffic width between registers is 8 bits. The datapath is tuned to routing, we organize all flip-flops as 8-bit wide shift registers. As seen in Figure 2, the maximum data traffic width between registers is 8 bits. The datapath is tuned to process an average of 1 byte every clock after the initial pipeline delay. The 7-byte extension register (X6-X0) is used to delay the output data so that it can be fed back to the main register (R7-R0) at the start of the next round.

The main penalty with our design is the additional two rounds added in order to implement the FL/FL-1 functions. Our design requires 20 rounds to process a 128-bit data block (Figure 3). It takes a total number of 400 clock cycles to encrypt 128-bit input block (including key expansion and pipeline delay), which corresponds to a processing rate of 1.14 cycles per byte [14].

In order to clarify the data flow in our pipelined architecture, we name the I/O and internal bytes for the regular and FL/FL-1 rounds as shown in Figures 4 and 5. In Figure 6, we plot the contents each register for three rounds (a regular round followed by an FL/FL-1 round, followed by a regular round). Circled register contents are used to generate the output of the corresponding cycle. We are able to implement a fully serial datapath and most of the Camellia functionality with a single S-Box and 8-bit wide logic gates. However, a 64-bit parallel P-function and two 32-bit parallel rotate functions are still needed.

### A. Regular Round Data Processing

In a regular round, data \( (x_0...x_7) \) enters the pipeline through the input register (IR) together with the round key \( (k_0...k_7) \) starting with the most significant byte \( (x_0) \), as shown in Figure 4. In the first 8 cycles, the output of IR is XORed with the key and passed through the S-Box. The output of the S-Box \( (s_i) \) enters the main processing register through \( R7 \), while the output of IR enters the backup registers \( B7-B0 \) for future processing.

At the same time, the rest of the P-function block output, \( p_i...p_7 \), are loaded in parallel to \( R6-R0 \), replacing the \( s_i \)'s; and backup register \( R0 \) output is routed to \( R7 \) input, restoring the original most significant input byte. In the following 7 cycles, the output of \( R0 \) is XORed with \( IR \) output and sent to the extension register as \( y_1...y_7 \), and the rest of \( x_1...x_7 \) are restored through \( R7 \).

At the end of the 16th cycle, output of \( R0 \), \( x_0 \), is fed into \( X6 \) as \( y_0 \); \( y_0 \) is at the output of \( X0 \) and fed into IR as the first input byte of the next round.

The last 8 cycles of the present round and the first 8 cycles of the next round take place in parallel. No cycles are wasted resulting in perfect pipelining at the expense of 8 backup registers, \( B7-B0 \).

### B. FL/FL-1 Round Data Processing

Data flow in these rounds is rather complex. Data \( (x_0...x_7) \) enters the pipeline through the input register \( (IR) \), together with 16 bytes of round key \( (k_0...k_7) \). Figure 5 shows the byte numbering for FL/FL-1 rounds.

The first 4 bytes of data are ANDed with the key bytes and sent to \( R7 \) as \( r_0...r_7 \), while being backed-
up inside B7-B4. At the end of the 4th cycle, 32-bits inside R4-R7 are left rotated by 1-bit, producing q0...q3. Output of IR, x0, is XORed with q0, ORed with the key byte, k8, and XORed with B4 output, x0, to generate y0, which is fed into R3 in the 5th cycle. At the same cycle, q1...q3 are loaded in parallel to R6-R4 and used to generate y1...y3 in the next 3 cycles. During cycles 5-8, output of the first XOR, y1...y3, are fed back through R7. Starting with the 9th cycle, y0...y7 are fed into the extension registers X6-X0 to be sent as inputs to the next round.

During the next 4 cycles, input bytes x8...x11 are pushed into main processing register, while the key bytes k8...k11 are stored in the backup registers B7-B4. Starting with the 12th cycle, output of IR (x12,...x15) is ORed with key input (k12...k15), and XORed with R4 output (x12...x15). The output of this operation is the 3rd quarter of the round output (y8...y11) and is sent to B7-B4. These bytes are also ANDed with B4 output (restored keys, k8...k11), XORed with the output of IR (x12...x15) to generate y12...y15, and fed through R7 into the processing register.

C. S-Box Design

Four different S-Boxes, S1-S4 used by the Camellia algorithm can be derived from a base design by rotating either the input or output. S1 is the base S-Box and its implementation is the main factor in the determination of the slice/gate count. A ROM look-up table based and a composite inverter based [1] S-Box are implemented for FPGA and ASIC implementation, respectively. The S2-S4 S-Boxes are derived from the base S1 S-Box via a multiplexer network.

D. P-Function Block

The structure of the P-function block is shown in Figure 7, which is slightly different than the P-block given in this standard. Our P-function block also incorporates the XOR logic on the right path of the Camellia round.

E. FL/FL^{-1} Function Block

Figure 8 shows the structure of the FL/FL^{-1} function. It is the equivalent circuit that performs the serial processing of FL/FL^{-1} functionality as explained in Section B. 32-bit rotations are performed in parallel.

Figure 6. Register contents for pipelined Camellia processing.

Figure 7. P-function block

Figure 8. FL/FL^{-1} function block

Figure 9. Key generation module
TABLE I. PERFORMANCE COMPARISON

<table>
<thead>
<tr>
<th>Design</th>
<th>Freq (MHz)</th>
<th>Clock cycles</th>
<th>Block size (bits)</th>
<th>Key size (bits)</th>
<th>Area (slices/gates)</th>
<th>T/put (Mbps)</th>
<th>T/put / area (Mbps/slice / Mgps/gates)</th>
<th>Device / Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camellia (Ours)</td>
<td>103</td>
<td>400</td>
<td>128</td>
<td>128</td>
<td>321</td>
<td>32.96</td>
<td>0.103</td>
<td>xc3s50-5</td>
</tr>
<tr>
<td>Camellia (Ours)</td>
<td>267</td>
<td>400</td>
<td>128</td>
<td>128</td>
<td>132</td>
<td>85.44</td>
<td>0.647</td>
<td>xc5vlx30-3</td>
</tr>
<tr>
<td>Camellia [14]</td>
<td>125</td>
<td>875</td>
<td>128</td>
<td>128</td>
<td>318</td>
<td>18.41</td>
<td>0.06</td>
<td>xc3s50-5</td>
</tr>
<tr>
<td>TinyXTEA-3 [16]</td>
<td>63</td>
<td>112</td>
<td>64</td>
<td>128</td>
<td>254</td>
<td>35.78</td>
<td>0.14</td>
<td>xc3s50-5</td>
</tr>
<tr>
<td>Camellia (Ours)</td>
<td>253</td>
<td>400</td>
<td>128</td>
<td>128</td>
<td>4313</td>
<td>81</td>
<td>0.02</td>
<td>130nm ASIC</td>
</tr>
<tr>
<td>AES[17]</td>
<td>290</td>
<td>160</td>
<td>128</td>
<td>128</td>
<td>3900</td>
<td>232</td>
<td>0.06</td>
<td>130nm ASIC</td>
</tr>
</tbody>
</table>

F. Key Expansion

Camellia key expansion is performed by the main core using the original key, \( K_o \), and the constant keys. The original key and the expanded key, \( K_A \), are stored inside the key registers \( L15-L0 \) and \( A15-A0 \), respectively. Each round uses a rotated version of one of these keys. We manage the rotations larger than 16-bits by rotating all 16 registers of the next round’s key by 1, 2, or 3 bytes during the idle cycles of each round. Sub-16 bit rotations are managed by the 16-bit rotators and the multiplexer network shown in Figure 9. The byte-serial nature of the design brings down the rotation circuit sizes down to 16-bits. The byte and bit rotation sizes depend on the round and cycle counts, and are determined by the control logic.

IV. PERFORMANCE

We implemented our design on the smallest XILINX Spartan device, smallest Virtex-5 device and on a 130 nm CMOS technology. Table I summarizes the results.

With the Spartan family, the slice count is 321 at a frequency of 103 MHz. For the Virtex-5 family, the slice count is 132 at a frequency of 267 MHz. Total cycle count for the key expansion and processing of a data block is 400 cycles, resulting in a throughput of 32.96 and 85.44 Mbps, respectively.

For the ASIC implementation, we chose both the area and cycle counts, and are determined by the control logic.

V. CONCLUSION

We have demonstrated that pipelining alone can drastically increase the performance of Camellia. We have achieved the highest throughput per area among the other compact Camellia cores. We have even gone below more compact AES cores in terms of power consumption. With its high throughput per area figures, and low power consumption, our Camellia core proves to be a strong candidate for resource-limited applications which demand lightweight cryptography.

REFERENCES