A 12 Gbps DES Encryptor/Decryptor Core in an FPGA

Steve Trimberger[1], Raymond Pang[1], and Amit Singh[2]

[1] Xilinx, Inc, 2100 Logic Dr, San Jose, CA 95124, USA
Steve.trimberger@xilinx.com
[2] University of California, Santa Barbara, CA, USA

Abstract. This paper describes two implementations of a Data Encryption Standard (DES) encryptor/decryptor that operate at data rates up to 12 Gbps. The 12 Gpbs number is faster than any previously published design. In these DES implementations, the key can be changed and the core switched from encryption to decryption mode on a cycle-by-cycle basis with no dead cycles. The designs were synthesized from Verilog HDL and implemented in Xilinx XCV300 and XCV300E devices. This paper describes the optimizations used and the coding conventions required to direct the synthesis tools to map the design to achieve a high-speed implementation. No physical constraints were given to the tools.

1 Introduction

The rapid growth of virtual private networks has heightened demand for encryption hardware that can handle high data rates. The hardware-friendly DES algorithm is well-suited to this application. Concerns about the vulnerability of DES are driving further standardization efforts, so any encryption hardware that is deployed today may become obsolete in a few months. In contrast, if the encryption engine resides in an FPGA, it could be updated in the field with a new encryption algorithm when that algorithm is available.

The DES algorithm has a regular structure that lends itself to pipelining, and simple data manipulations that permit fast operations. Several high-speed DES hardware implementations have been reported in the literature. These implementations unroll the 16 rounds of encryption and pipeline them. Wilcox [8] describes an ASIC implementation that operates above 10Gbps. Patterson [5] compiles a key-dependent data path for encryption in an FPGA that runs over 12Gbps [6], but the latency to change keys is tens of milliseconds. The most directly comparable prior art design implemented in an FPGA has complete loop unrolling and encrypts at 3.05Gbps [3].

This paper describes the implementation and optimization of an FPGA core for DES encryption and decryption. The core achieves a data rate of 8.4 Gbps with 16 cycles of latency, and 12 Gpbs with 48 cycles of latency. The core takes a key an encrypt/decrypt signal, both of which may change on a cycle-by-cycle basis.

Ç.K. Koç and C. Paar (Eds.): CHES 2000, LNCS 1965, pp. 156 - 163, 2000.

Since the core is compiled Verilog, it is simple to concatenate multiple copies of the core in a larger FPGA to provide triple DES [1] at the same data rate. It is also straightforward to interface the core to data concentrators and different communication interfaces supported by the FPGA, such as LVDS or double data rate (DDR) RAM.

2 The DES Algorithm

DES [2][7] takes as input one 56-bit key and one 64-bit block of data, and produces one 64-bit block of encrypted data. The same basic design is used for both encryption and decryption. As shown in figure 1, the DES algorithm begins with an initial permutation (IP), encrypts in sixteen "rounds", followed by the inverse of the initial permutation (IP^{-1}). In each round, the right-side 32 bits of the block are transformed with the function labeled "*f* " and the key, then exclusive-ored with the left side 32 bits. The key for each round is a subset of the original 56-bit key with bits permuted. After each round, the two sides of the data block are swapped and the algorithm continues.

The *f* function expands the right side to 48 bits, exclusive-ors those bits with the key, and divides the resulting 48 bits into eight 6-bit fields. Those fields are used as addresses into 8 64-word by 4-bit memories called S-boxes. The eight 4-bit S-box outputs are re-assembled into the 32-bit word that is XORed with the left side of the block.

Decryption differs from encryption in the way the bits of the sub-keys (K_1-K_{16}) are selected from the encryption key. This selection leads to the key bits multiplexer in figure 2.

In summary, the DES algorithm consists of 16 identical encryption rounds. Each round contains a significant amount of bit movement, which is simple wire in a hardware implementation, 80 2-bit XORs, and 8 lookups in 64-word by 4-bit S-boxes. Each round uses a subset of the key bits with a particular permutation. The permutation depends on the round and on whether the operation is encrypt or decrypt. Consisting primarily of wiring, table lookups and bitwise operations, the algorithm fits nicely into an FPGA.

3 Implementation

We coded the design in Verilog and simulated with Cadence Verilog-XL 2.2.31. We synthesized with Synopsys Design Analyzer 1998.08, targeting Xilinx Virtex-6 speed grade. Physical design was done with Xilinx Design Manager 2.1I, C.22.

The original Verilog design was intended to be space efficient, and was implemented as a single instantiation of the encryption round that operated iteratively. A single block of data passed through the round 16 times to produce one block of output. We modified this original version in several ways to gain significant throughput and clock rate improvements. The following sections describe those improvements.

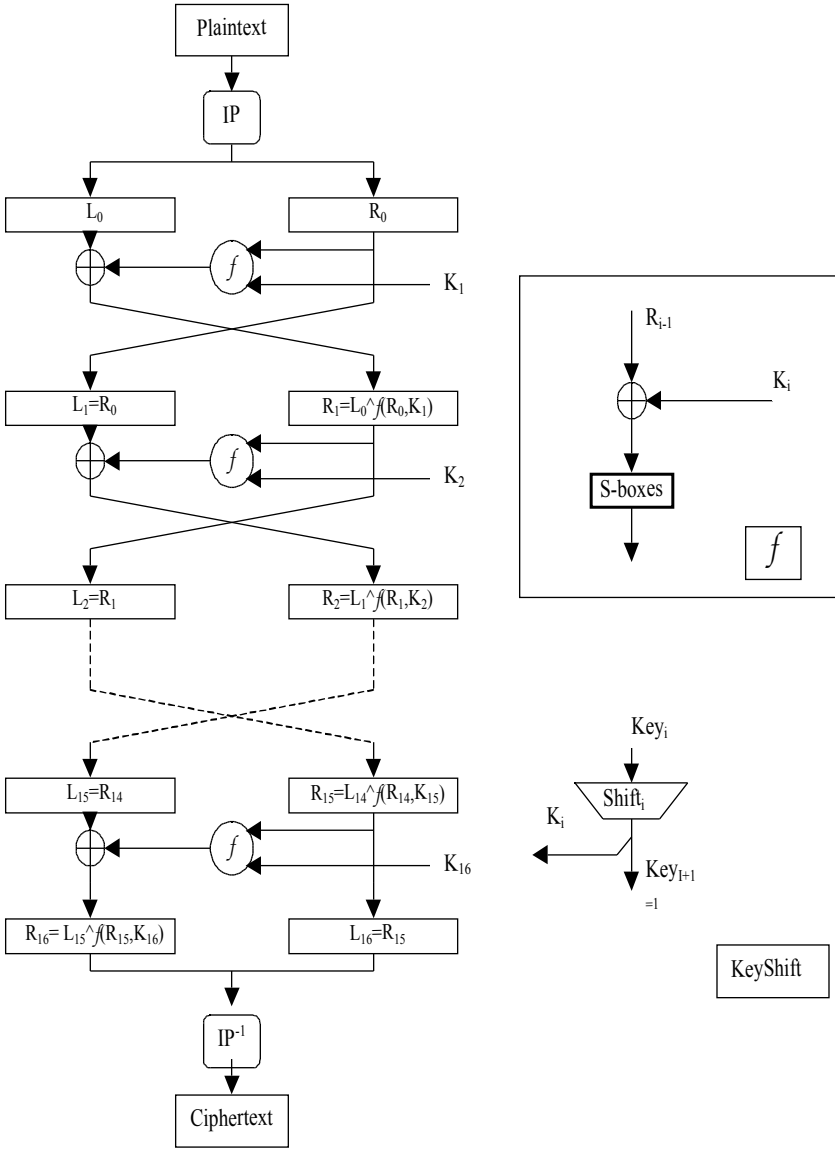

Fig. 1. DES Algorithm Overview

3.1 Loop Unrolling and Pipelining

To gain speed, we built 16 copies of the round and unrolled the loop, pipelining the data through the 16 stages. This increased the data rate by a factor of sixteen, but at the cost of approximately sixteen times as much logic. This design simulated fine, but logic synthesis, which we ran at medium effort, predicted a clock rate less than 25MHz. The critical path through the round is shown in figure 2. A multiplexer selects key bits depending on the round and on whether we are encrypting or decrypting. The resulting selected key bits are XORed with the right side of the data block (R_i) and 6-bit fields are used to address in to the S-boxes. One bit of one S-box is shown as 4LUTs, F5 and F6 MUXes. The resulting bits from the S-box are XORed with the left side bits from of the block (L_i) and stored in the pipeline register.

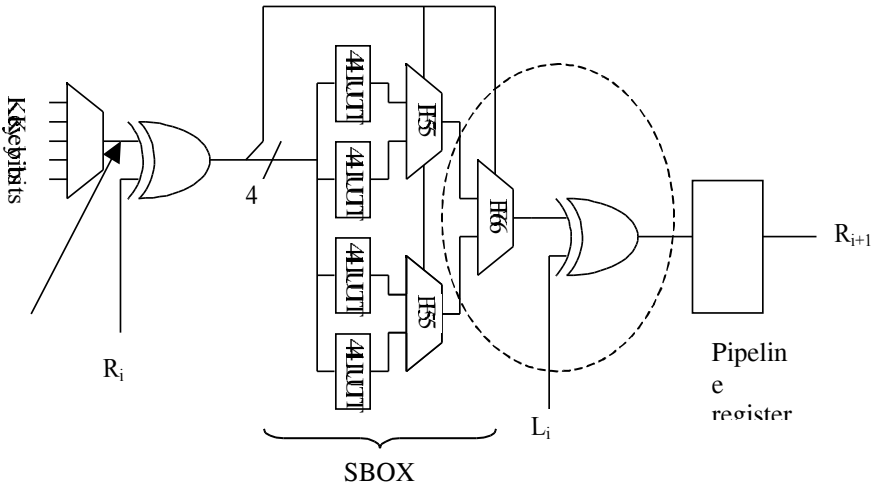

Fig. 2. Single Round Data Path.

3.2 Mapping to LUTs

Clearly, the critical path is through the logic of a single round. In our initial implementation, we used an SBOX expressed as 2-input functions[4] which produced an appallingly slow result, so we re-coded the SBOXes as 64x4 lookup tables. This design resulted in a critical path of eleven levels of LUTs which still performed rather poorly. We recognized that the Virtex CLB can implement one bit of an S-Box completely as a 64-word lookup table, which would reduce the logic to only three levels in the FPGA, but the synthesis tool did not implement the logic that way despite the Verilog description that looked like a lookup table. The stylized form of Verilog shown in figure 3 directs the synthesis tool to generate a 4LUT, but the form does not extend to 5LUTs or 6LUTs, which is why the original design was mashed into gates.

We changed the Verilog to build 4LUTs, and directly instantiated the Virtex F5 MUX in Verilog. We decided not to instantiate the F6 MUX because the F6 function could be merged with the following XOR gate in a single 4LUT (circled in figure 2), and the Virtex F5 function is slightly faster than the F6 function. The LUT following the SBOX is required in either case to implement the XOR with the left side of the data block that follows the SBOX lookup.

After these modifications, synthesis implemented the critical path in five levels of logic: two for the key MUX, one for the key XOR, one for the S-box through the 5LUT, and one for the F6 MUX plus the final XOR. Synopsys estimated the resulting speed at 50MHz.

```
always @(addr) begin
case (addr[2:5])
    0:  d2 = 4;
    1:  d2 = 1;
    2:  d2 = 14;
    3:  d2 = 8;
    4:  d2 = 13;
    5:  d2 = 6;
    6:  d2 = 2;
    7:  d2 = 11;
    8:  d2 = 15;
    9:  d2 = 12;
    10: d2 = 9;
    11: d2 = 7;
    12: d2 = 3;
    13: d2 = 10;
    14: d2 = 5;
    15: d2 = 0;
endcase
end
```

Fig 3. Verilog Case Statement that Generates a 4LUT.

3.3 Verilog Parameters

Focus turned now to the key multiplexer (on the left in figure 2). That multiplexer was rather wide because the Verilog code, which had been written for an iterative implementation, included a module for the key shift block. The module took as input signals that identified the round because the shift is different for different rounds. In the iterative version, those signals changed every cycle, but in the pipelined version,

those bits were tied to constants. However, since they were passed into a Verilog module, the multiplexers were not simplified by synthesis, so each of those MUXes had five or six inputs instead of two (one for encryption, one for decryption).

The solution was to implement these signals as Verilog parameters. To do this, the module must be declared a *template* in the synthesis tool, so the tool can create a separate module in the data base for each instance.

3.4 Decoupling the Key from the Data Path

In order to take the next step in performance improvement, we decided to take the key MUX off the critical path by pre-computing the key shift selection. This was rather straightforward. Since the key calculation must be pipelined along with the block of data, we moved the pipeline registers in the key calculation data path to the location marked with an arrow in figure 2. We added an additional pipeline stage in the first round at that location. The key must now be presented one cycle before the data it operates on. This modification is similar in concept to pre-computing the key schedule, which is common in software implementations, and which Patterson [5] did in software in his reconfigurable implementation. Since we compute the key in hardware anyway, we require no more logic to continually re-compute the key schedule one cycle ahead of when it is needed. This way, the decryptor is still able to switch keys on a cycle-by-cycle basis. After this modification, synthesis estimated a clock rate of about 70MHz. It was time for physical design.

3.5 Physical Design

We generated EDIF from the synthesized circuit, read the EDIF into the Xilinx Design Manager and set the target to XCV300-6. Without any constraints, placement and routing ran for about half an hour and produced a design that Design Manager reported used 4216 LUTs (about 69% capacity) and would run about 80MHz. We set a single timing constraint: the clock period should be 10ns, and set high placer and router effort (5). Placement and routing met the constraint after about four hours, yielding a circuit that encrypts or decrypts a 64-bit block every 10ns, a 6.4Gbps data rate. Further tightening of the clock period did not improve the resulting performance.

Next, we set the target to XCV300E-8 and tightened the clock rate constraint to 7.5ns. The resulting circuit ran at 132MHz, encrypting at 8.4Gbps.

We gave no placement constraints or hints. All performance numbers reported after physical design are post-layout, worst-case timing reports from the Xilinx Design Manager.

3.6 Deeper Pipeline

Finally, to wring a higher data rate, we inserted pipeline stages after the key XOR and after the F5 step in the S-Box lookup (and added two more pipeline stages in the key shift to maintain data alignment). This resulted in a 48-stage pipeline. Placement and routing with high effort, multiple-pass place-and-route and a tightening clock period

constraint yielded a designs that would operate at a data rate of 10.1Gbps in an XCV300-6 (6.3ns clock period), and 12Gbps in an XCV300E-8 device (5.3ns clock period).

4 Statistics

Both the 16-cycle and 48-cycle latency designs have these IO connections:

`Data_bus[1:64]`	Input data
`Data_out[1:64]`	Output data
`Key[1:56]`	Key data to be applied to the following data block
`Decrypt/encrypt`	The mode of operation on the current block
`E_data_rdy`	Input data is valid this cycle
`D_data_rdy`	Output data is valid this cycle
`Clk`	Clock

`Decrypt`, `E_data_rdy` and `D_data_rdy` are presented simultaneous with the data and are pipelined along with the data. The design can encrypt one cycle, and decrypt the next cycle with no dead cycles. `Key` must be presented one cycle before the data it operates on. Keys can also change every cycle with no dead cycles.

Here are implementation results for the two designs. Notice that the number of LUTs does not increase with increased pipeline depth. Pipeline registers in the data path require no additional logic, since the flip-flops in following the logic in the 16-stage pipeline were unused. The additional pipeline registers on the key bits required to maintain data alignment do add additional logic.

	16-stage pipeline	48-stage pipeline
Verilog Code Size (lines)	1106	1156
I/O	187	187
Design Size (LUTs)	4216	4216
Design Size (FFs)	1943	5573
Design Size estimated by Xilinx mapper (gates)	52936	72952
Data rate in XCV300-6 device (Gbps)	6.4	10.1
Data rate in XCV300E-8 device (Gbps)	8.4	12.0

5 Conclusion and Future Work

We designed a DES encryptor/decryptor core in Verilog and targeted it to an FPGA. The resulting design with 16 cycles of latency runs at 8.4Gbps. The design with 48

cycles of latency runs at 12Gbps. The speed of this design is approximately three times faster than the previous fastest comparable FPGA implementation. It is faster than an ASIC design reported only a year ago, and is comparable to a custom-key encryptor that requires tens of microseconds to change keys.

Part of the reason for executing this design was to determine the performance gained from various forms of optimizations to the design. We intend to use this information to drive design automation software development. In this design we were able to improve performance by more than a factor of two by applying an understanding of the algorithm to force a preferred mapping of the logic, and by changing the pipelining of the key. Although the former may be someday incorporated into logic synthesis software, many designers may not appreciate software that unilaterally changes the data alignment.

An observation of the delays in the final design shows that most of the delay in both implementations is due to interconnect routing. The next step in this investigation is to apply manual floorplanning and placement to reduce interconnect delay.

We are also interested in implementing additional encryption algorithms, with the intent that a system would load the algorithm of choice into the FPGA as needed. This strategy would permit, for example, fielding a system today that could be updated when the advanced encryption standard becomes available.

References

1. ANSI, "Triple Data Encryption Algorithm Modes of Operation", American National Standards Institute X9.52-1998, American Bankers Association, Washington DC, July 29, 1998
2. FIPS, "Data Encryption Standard", Federal Information Processing Standards Publication 46-2, 1993 December 30.
3. FreeIP, http://www.free-ip.com/DES/index.html
4. Kwan, M., "Bitslice DES", http://www.darkside.com.au/bitslice/ nonstd.c
5. Patterson, C., "High Performance DES Encryption in Virtex FPGAs using Jbits", *FCCM 2000*, IEEE Computer Society, 2000.
6. Patterson, C., private communication, 2000.
7. Schneier, B., *Applied Cryptography*, John Wiley and Sons, 1996.
8. Wilcox, D.C., et al., "A DES ASIC suitable for network encryption at 10Gbps and beyond", *First International Workshop on Cryptographic Hardware and Embedded Systems*, 1999.