High Performance DES Encryption in Virtex™ FPGAs using JBits™

Cameron Patterson
Xilinx, Inc.
2300 55th Street
Boulder, Colorado 80301
Cameron.Patterson@xilinx.com

Abstract

A JBits implementation of the Data Encryption Standard (DES) algorithm in a Virtex FPGA is described. The Virtex architecture efficiently implements the DES primitive operations, and permits a high degree of pipelining. JBits provides a Java-based Application Programming Interface (API) for the run time creation and modification of the configuration bitstream. This allows dynamic circuit specialization based on a specific key and mode (encrypt or decrypt). The key schedule is computed entirely in software, and is part of the bitstream. As a result, all cryptographic key input and subkey generation logic are removed from the fully unrolled datapath. When combined with a speed efficient layout, the result is a throughput of over 10 Gigabits per second. This is sufficient bandwidth for SONET OC-192c (optical) networks, and exceeds the speed reported for a recently announced DES ASIC.

1 Introduction

Cryptographic algorithms such as the Data Encryption Standard (DES) are frequently implemented in Field-Programmable Gate Arrays [1]. Orders of magnitude speedup over software implementations are due to the following attributes of DES:

- It is a pure datapath.
- The primitive operations used are bit-level substitutions and permutations, which are inefficient in software. Fixed permutations are essentially free in hardware.
- DES has 16 rounds, which can be unrolled and pipelined in hardware.

The fastest software throughput reported in [2] is 13 Megabits/sec. Some speeds reported for recent academic and commercial FPGA implementations are given in Table 1. Note that only the first entry uses key-specific optimization of the circuitry.

It is generally assumed that ASIC implementations of DES provide the highest speed. The fastest DES ASIC known to the author was recently announced by Sandia National Laboratories [8]. It fully unrolls and pipelines the 16 rounds, and has been tested at 105 MHz (6.7 Gigabits/sec). Power consumption at 105 MHz is 6.5 watts. Simulation predicts a maximum speed of 145 MHz (9.28 Gigabits/sec).

A fully unrolled and pipelined FPGA implementation of the DES algorithm has been developed for which timing analysis predicts a maximum speed of 168 MHz, or 10.7 Gigabits/sec. This comfortably meets the bandwidth requirements for SONET OC-192c optical networks. Power consumption at 168 MHz is 3.2 watts. The remainder of this paper describes the technology, techniques and tools that enable this performance.

2 The DES Algorithm

In 1973, the U.S. National Bureau of Standards (now called the National Institute of Standards and Technology or NIST) solicited proposals for a standard cryptographic algorithm. IBM submitted a variant of their Lucifer algorithm, which utilized a 16-round substitution-permutation network. The U.S. National Security Agency evaluated the design, and recommended changes such as reducing the key size from 128-bits to 56-bits. In 1977, the Data Encryption Standard was adopted as a Federal Information Processing Standard for unclassified government communication [10].

DES encrypts 64-bit blocks with a 56-bit key[1]. The use of triple encryption with two different keys allevi-

[1] The key is transmitted as 64-bits, but this includes a parity bit for each byte.

FPGA(s) Used	Clock Rate (MHz)	Throughput (Megabits/sec)	Loop Unrolling	Design Capture	Reference
Xilinx 4013-4	7	26	none	VHDL	[3]
Xilinx 6216	23	57	none	schematics	[4]
Xilinx 4010E	43	172	none	VHDL/Verilog	[5]
Xilinx 4028EX-3	25	384	partial	VHDL	[6]
Xilinx V150-6	101	404	none	VHDL/Verilog	[7]
4 Altera 10K100-3	20	1280	full	AHDL	[8]
Xilinx V400-6	60	3656	full	VHDL	[9]

Table 1: Previous FPGA Implementations of DES

ates concerns of cryptanalysis using an exhaustive key search. The algorithm has 16 rounds or iterations, where each round consists of swapping the left and right 32-bits, as well as permutation, substitution and XOR operations. This is shown in Figure 1. Note that all permutation and expansion operations are implemented with wire crossings and fanout. A different subset of the key bits is used in each round. Decryption is identical to encryption, except that the subkeys are generated in the reverse order.

DES is a private or symmetric key algorithm, in which the same key is used for both encryption and decryption. A public key algorithm such as RSA may be used to encrypt and exchange this private key. Private key algorithms are generally much faster than public key algorithms (e.g. a hardware implementation of DES is about 1000 times faster than a hardware implementation of RSA). Programs such as PGP [11] typically use public key algorithms for key management, and private key algorithms for data encryption.

2.1 Implementing DES in the Virtex Architecture

In 1998, Xilinx introduced the Virtex architecture as the successor to the XC4000 family [12]. It uses a 2.5 volt, 0.22 micron, 5 metal layer process. Like the XC4000, it can be characterized as a symmetric array of CLBs surrounded by IOBs. Each CLB contains two slices, where each slice is roughly equivalent to an XC4000 CLB (i.e. it contains two 4-input LUTs, two flip flops, and a vertical carry path). System-level resources such as block RAM and delay-locked loops have also been added to Virtex.

Two features of the Virtex CLB make it especially well-suited to the DES algorithm:

- Each of the eight S-Boxes is specified as a table with a 6-bit input and a 4-bit output. Logic minimization algorithms find little structure in the S-Boxes, so it is reasonable to implement each S-Box as four independent 6-input single-output lookup tables. A single Virtex CLB can implement a 6-input LUT by combining four 4-input LUTs with CLB-resident multiplexers. The XC4000 implementation of a 6-input LUT is slower, since it requires two CLBs.

- A Virtex 4-input LUT can provide a 1 to 16-bit shift register function (SRL16). The LUT inputs define the position of the output tap on the SRL16. Beyond 3 bits, SRL16 power consumption is lower than connecting a series of flip flops. SRL16s allow a high degree of pipelining to be used in the DES datapath.

3 Run Time Reconfiguration

The run time reconfiguration (RTR) of FPGAs allows the customization of circuits to the problem instance at hand. This can have considerable speed and area advantages. For example, if one of the inputs to a multiplier is fixed for a given problem instance, then a general multiplier can be replaced with multiplication by a constant. The reconfiguration overhead must be acceptable to the application, however.

Xilinx SRAM-based FPGAs have always been reconfigurable. What has been missing, however, is software support for dynamic reconfiguration. The XACT, M1 and M2 products provide static (i.e. ASIC) design flows. Schematics and mainstream HDLs do not provide a way of specifying dynamic circuitry. The time and memory required by synthesis and implementation tools precludes their use in a run-time reconfiguration environment. The XC6200 family addressed partial and rapid reconfiguration at

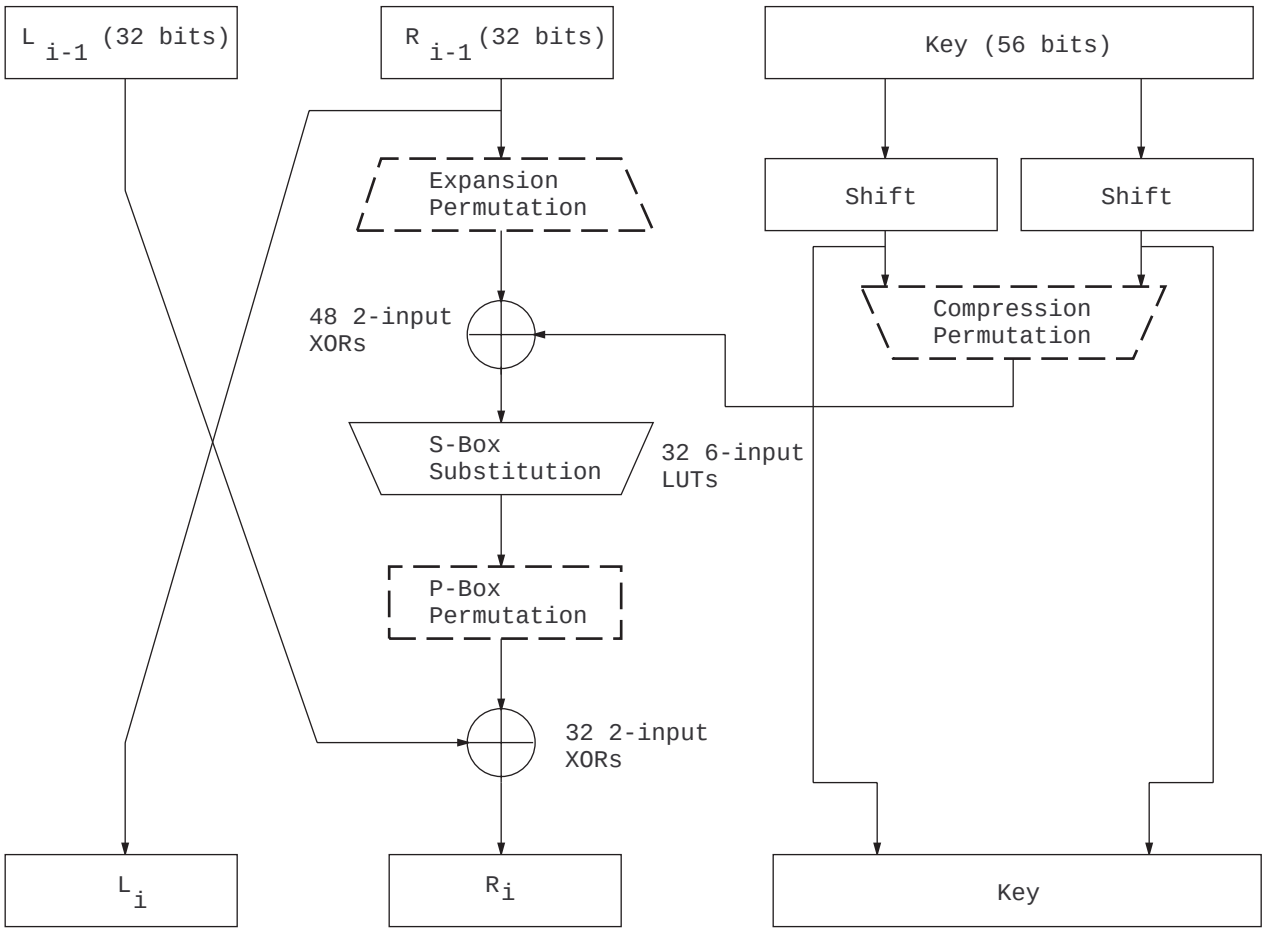

Figure 1: A Single DES Round

the architectural level, but the associated design flow started with EDIF input.

Netlist-based design and implementation flows are unsuited to the dynamic portion of RTR applications. An alternative model is to take a software-centric approach. A microprocessor-based application can directly create or modify the FPGA's configuration, provided an Application Programming Interface (API) is available for this purpose. The FPGA's configuration is not treated as data, but as state that is modified and queried by function calls. This abstraction is independent of whether the FPGA architecture supports partial reconfiguration, and works even if the entire FPGA bitstream must be rewritten when it is modified (as in the case of the XC4000).

This model also supports system level design, hardware/software co-design, and design reuse. The conventional approach is for the system architect to partition the design between software and hardware engineers. Integration of the hardware and software occurs late in the development effort, and may reveal interface problems. On the other hand, the configuration-API approach permits the software designer to incrementally define and test hardware functions using a library of run-time parameterized (RTP) FPGA cores.

3.1 JBits

JBits supports the configuration-API model in the Java programming language [13]. Currently, JBits is available from Xilinx for the XC4000 and Virtex architectures. JBits classes define the tiles used in the

architecture. For example, one of the tiles is a Configurable Logic Block and the associated General Routing Matrix. Each CLB instance is index by a row and column, and the configuration bits for the CLB can be queried or modified. Symbolic values are used to reference configurable structures (e.g. multiplexers) and their states. Figure 2 illustrates the JBits calls required to configure a CLB at (`row`,`col`) as a 6-input registered LUT. The code essentially defines paths through the CLB. Many of the multiplexer settings used are the power-up defaults.

For comparison, code which defines a netlist for a 6-input LUT in terms of Virtex primitive components (LUT4, MUXF5, MUXF6 and FD) is shown in Figure 3. The `placePrimitive` method specifies the CLB and slice coordinates for each component. Note that the netlist code is roughly equivalent in size to the JBits code. However, there is considerable time and memory overhead to map the netlist to the CLB configuration settings.

The JBits design approach clearly has run time advantages, which may be essential for the application. JBits coding requires a detailed knowledge of the architecture, though, and is not easily ported to other architectures. The netlist approach permits the use of mainstream simulation and implementation tools. However, there may not be complete control over mapping, placement and routing, which can make it difficult to exchange one logic block with another at run time. Our solution to this dilemma is to implement a DES core that generates a fully placed netlist. This netlist can be written out as EDIF for functional verification and processing by the M2 tools. In addition, part of the netlist is translated to JBits calls. This is done in the following situations:

- Whenever M2 does not permit the required degree of physical control (e.g. which LUT or flip flop is used in a slice, or the assignment of nets to LUT inputs).
- For portions of the circuit that are defined or modified at run time (e.g. LUT or block RAM contents, and connections to swapped in logic).
- For critical nets or logic that have no freedom of implementation.

The conversion of nets to JBits calls is assisted by a new utility called JRoute [14]. This will route a net with fanout that is specified in terms of endpoint locations. Prior to the development of JRoute, JBits users were required to specify the sequence of routing segments used for each net. For the regular connections arising in datapath circuits, JRoute provides a good trade off between run time and optimality. JBits calls can instead be used whenever JRoute does not produce the desired results.

4 DES Implementation

The objective was to find the maximum speed that can be obtained using a combination of Virtex and JBits technology. To this end, the 16 rounds of the DES algorithm were completely unrolled, and the optimal number of pipeline stages per round was investigated. The degree of pipelining must consider power consumption, since this may limit the maximum clock frequency. Several floorplans were also evaluated. Each of these aspects will be considered in turn.

4.1 Dynamic Circuit Specialization Using JBits

JBits permits computations to be easily partitioned between hardware and software. In DES, an obvious candidate for migration to software is precomputing the 48-bit subkey for each round. As shown in Figure 1, a subkey is XORed with the expanded right half of the data bits. Let d be a data bit and k be a subkey bit. Since k is a constant for a given encryption key, $\mathrm{XOR}(d, k)$ is either d or $\overline{d}$. An inversion on an S-Box input is equivalent to reordering the LUT contents. For example, inverting the least significant bit is the same as swapping adjacent entries in the LUT, and inverting the most significant bit can be achieved by swapping the two halves of the LUT.

The results of this optimization are shown in Figure 4. When compared with Figure 1, we see that the following have been removed:

- 48 XORs per round,
- all key input IOBs,
- all subkey generation and control logic.

The subkeys are now represented by permutations of the S-Box contents. The overall effect of this optimization is to make the S-Boxes both key and round specific.[2]

[2]Normally, the S-Box contents are independent of the key, and are the same in each round.

```
public void configureLUT6(int row, int col) {
    try {
        jBits.set(row, col, S0Control.BxInvert,  S0Control.OFF);
        jBits.set(row, col, S0Control.ByInvert,  S0Control.OFF);
        jBits.set(row, col, S0RAM.DUAL_MODE,     S0RAM.ON);
        jBits.set(row, col, S0RAM.LUT_MODE,      S0RAM.ON);
        jBits.set(row, col, S1Control.BxInvert,  S1Control.OFF);
        jBits.set(row, col, S1Control.ByInvert,  S1Control.OFF);
        jBits.set(row, col, S1RAM.DUAL_MODE,     S1RAM.ON);
        jBits.set(row, col, S1RAM.LUT_MODE,      S1RAM.ON);
        jBits.set(row, col, S1Control.Y.Y,       S1Control.Y.F6);
        jBits.set(row, col, S1Control.YDin.YDin, S1Control.YDin.BY);
        jBits.set(row, col, S1Control.LatchMode, S1Control.ON);
        jBits.set(row, col, S1Control.Sync,      S1Control.ON);
    } catch (ConfigurationException ce) {
        System.out.println("ERROR: unable to configure 6-input LUT at R" +
                           "row + "C" + col);
        System.out.println(ce);
    }
}
```

Figure 2: Implementing a 6-input Registered LUT with JBits

4.2 Pipelining

Two obvious places for the insertion of registers in Figure 4 are after the S-Box and XOR operations. This requires balancing of the left and right paths, as shown in Figure 5. Note that three pipeline stages are required in the left half, since that is the number of registers that the right half passes through before the two paths reconverge at the XOR of the next round. Prior to the first round, the left half requires an additional register. In the last round, the left half only requires two pipeline stages.

In terms of both density and power, the Virtex SRL16 primitives efficiently implement the three adjacent pipeline stages in the left path. A total of 32 SRL16s are needed, which utilizes 8 CLBs. If flip flops were used, then 24 CLBs would be required. Flip flops pipeline the right path.

The input and output IOBs are registered, and a pipeline stage is inserted prior to the first round. IOB nets that are single fanout connections from a register to a register permit greater freedom in placing the IOBs. Since the left half already required a pipeline stage before the first round, it will have two stages before the first round, and the right half will have one stage. The total number of pipeline stages from input to output pins is 35.

4.3 Floorplan

For each round, the resources required are:

- 8 CLBs for the 32 SRL16s.
- 8 CLBs for the 32 registered XORs.
- 32 CLBs for the 8 S-Boxes. Each S-Box consists of four 6-input LUTs with registered outputs.

The chosen arrangement for these CLBs is shown in Figure 6. This aspect ratio permits all sixteen rounds to fit within the 24×36 CLB array of an XCV150 device. Every LUT is used in the 768 CLBs that implement the 16 rounds.

An EPIC ratsnest view of the complete design is shown in Figure 7. All 64 connections between rounds are through the center 8 CLBs. These connections are equally efficient in either a left-to-right or right-to-left dataflow. The initial pipeline registers form the "handle" of the left. Slice utilization is 91% (1584 slices used from the 1728 available). The design requires 2560 combinatorial LUTs, 512 shift register LUTs, 1248 flip flops, and 129 IOBs (64 data input, 64 data output, and the clock input). There is no control logic; a useful purpose could not even be found for a reset signal. CLB and IOB resources are available for additional interface circuitry.

```
public void configureLUT6(int row, int col) {
    Net lut4aOut  = addNet("lut4aOut");
    Net lut4bOut  = addNet("lut4bOut");
    Net lut4cOut  = addNet("lut4cOut");
    Net lut4dOut  = addNet("lut4dOut");
    Net muxf5aOut = addNet("muxf5aOut");
    Net muxf5bOut = addNet("muxf5bOut");
    Net muxf6Out  = addNet("muxf6Out");

    LUT4  lut4a  = new LUT4(lut6In0, lut6In1, lut6In2, lut6In3, lut4aOut);
    LUT4  lut4b  = new LUT4(lut6In0, lut6In1, lut6In2, lut6In3, lut4bOut);
    LUT4  lut4c  = new LUT4(lut6In0, lut6In1, lut6In2, lut6In3, lut4cOut);
    LUT4  lut4d  = new LUT4(lut6In0, lut6In1, lut6In2, lut6In3, lut4dOut);
    MUXF5 muxf5a = new MUXF5(lut4aOut,  lut4bOut,  lut6In4, muxf5aOut);
    MUXF5 muxf5b = new MUXF5(lut4cOut,  lut4dOut,  lut6In4, muxf5bOut);
    MUXF6 muxf6  = new MUXF6(muxf5aOut, muxf5bOut, lut6In5, muxf6Out);
    FD    fd     = new FD(clock, muxf6Out, lut6Out);

    placePrimitive(lut4a,  row, col, "S0");
    placePrimitive(lut4b,  row, col, "S0");
    placePrimitive(lut4c,  row, col, "S1");
    placePrimitive(lut4d,  row, col, "S1");
    placePrimitive(muxf5a, row, col, "S0");
    placePrimitive(muxf5b, row, col, "S1");
    placePrimitive(muxf6,  row, col, "S1");
    placePrimitive(fd,     row, col, "S1");
}
```

Figure 3: Defining a Netlist for a 6-input Registered LUT

4.4 Validation and Performance

Functional verification was performed with the Model Technology VHDL simulator. An EDIF file with the S-Box LUTs configured for a particular key is first generated. This is converted to an NGD file using `ngdbuild`. The `ngd2vhdl` program creates a structural VHDL netlist and testbench. Encryption and decryption results were identical to the output from Schneier's DES software [2] and a second independent DES program.

When targeting an XCV150-6, static timing analysis indicates a maximum clock rate of 168 MHz, or 10.7 Gigabits/sec. Using a Virtex AFX board [15] with the data outputs connected to the data inputs,[3] less than 1 watt is consumed at 50 MHz. The power consumption at 168 MHz should therefore be about 3.2 watts.[4] For comparison, the Sandia Lab's DES ASIC, which is implemented in a 0.6 micron 5 volt process, consumes 6.5 watts at 105 MHz [8]. This demonstrates how the Virtex advanced process (i.e. 0.22 micron, 2.5 volt) more than compensates for the routing overheads associated with FPGAs. Only the highest volume ASICs may be able to justify using a process technology similar to Virtex.

The time required to switch keys or operating mode depends upon the speed of the host processor, operating system overheads and bus bandwidth to the FPGA. Assuming a 600 MHz Pentium III PC running NT 4.0 with the Virtex chip on a 33 MHz 32-bit PCI card, JBits can calculate and reconfigure the S-Box LUTs in tens of milliseconds. This is roughly equivalent to the time required for other system activities such as disk I/O. An embedded systems environment could use the Virtex SelectMap™ configuration interface directly, which is 8 bits wide and can run at up to 50 MHz (66 MHz if handshaking is used).

[3] This configuration forces many of the inputs to toggle every clock cycle.

[4] Power $= f_c V^2 C/2$, where f_c is the clock frequency, V is the power supply voltage, and C is the capacitive load.

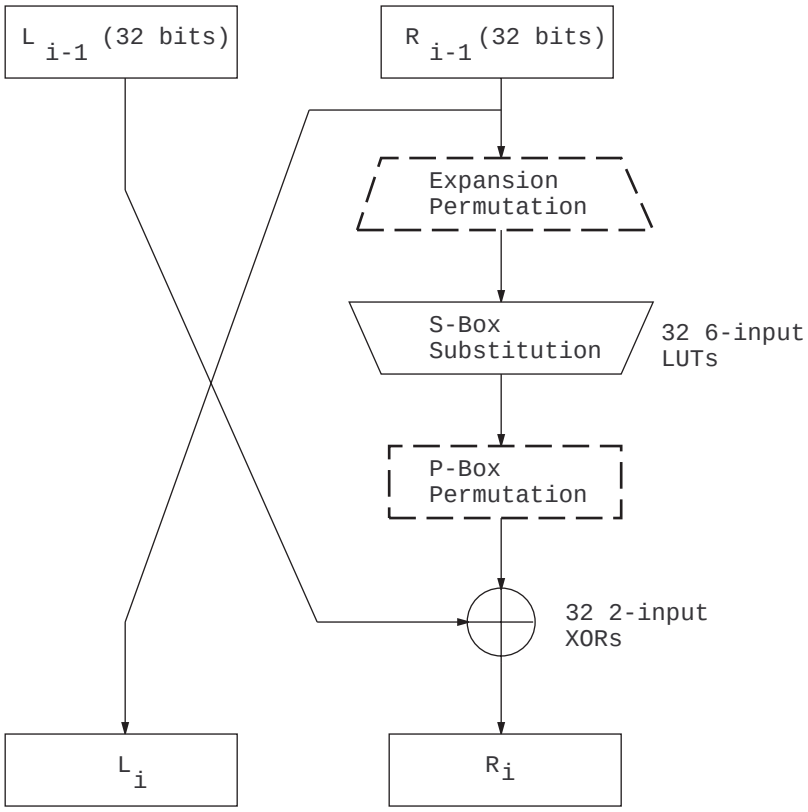

Figure 4: A JBits Optimized DES Round

5 Future Work

Additional functional verification and performance characterization is planned, including the use of Xilinx's production test equipment that can apply test vectors at the maximum clock rate. Independent validation will be performed prior to commercial availability.

The time required to reconfigure the S-Box LUTs when the key or operating mode is changed can be reduced by using partial reconfiguration of the Virtex device. Unlike the XC4000 family, Virtex does not require the entire configuration bit stream to be rewritten [16]. The existing S-Box floorplan minimizes the number of configuration frames that must be modified. Current JBits development includes the automatic determination and writing of partial bitstreams.

The DES design will be migrated to Virtex-E, which uses a 1.8 volt, 0.18 micron, 6 layer metal process. Maximum speed should be improved by about 20%. Alternatively, power consumption will be reduced at the same speed. No changes are required to the DES hardware or software. All that is needed is a port of JBits from Virtex to Virtex-E, which is straightforward.

Several candidates have been proposed for the Advanced Encryption Standard (AES), which is the successor to DES [17]. Some of the design requirements for AES that differ from DES are:

- A block size of 128 bits.
- A key size of 128 bits, 192 bits, or 256 bits.
- Efficient implementation in both hardware and software.

The majority of the AES candidates use the same primitive operations as DES, such as substitutions, permutations, and XORs. Most tables used to perform substitutions fit in the Virtex Block or LUT RAM. The algorithms have from 6 to 48 rounds, which can be partially or completely unrolled and pipelined. As with DES, the subkey for each round can be precomputed in software.

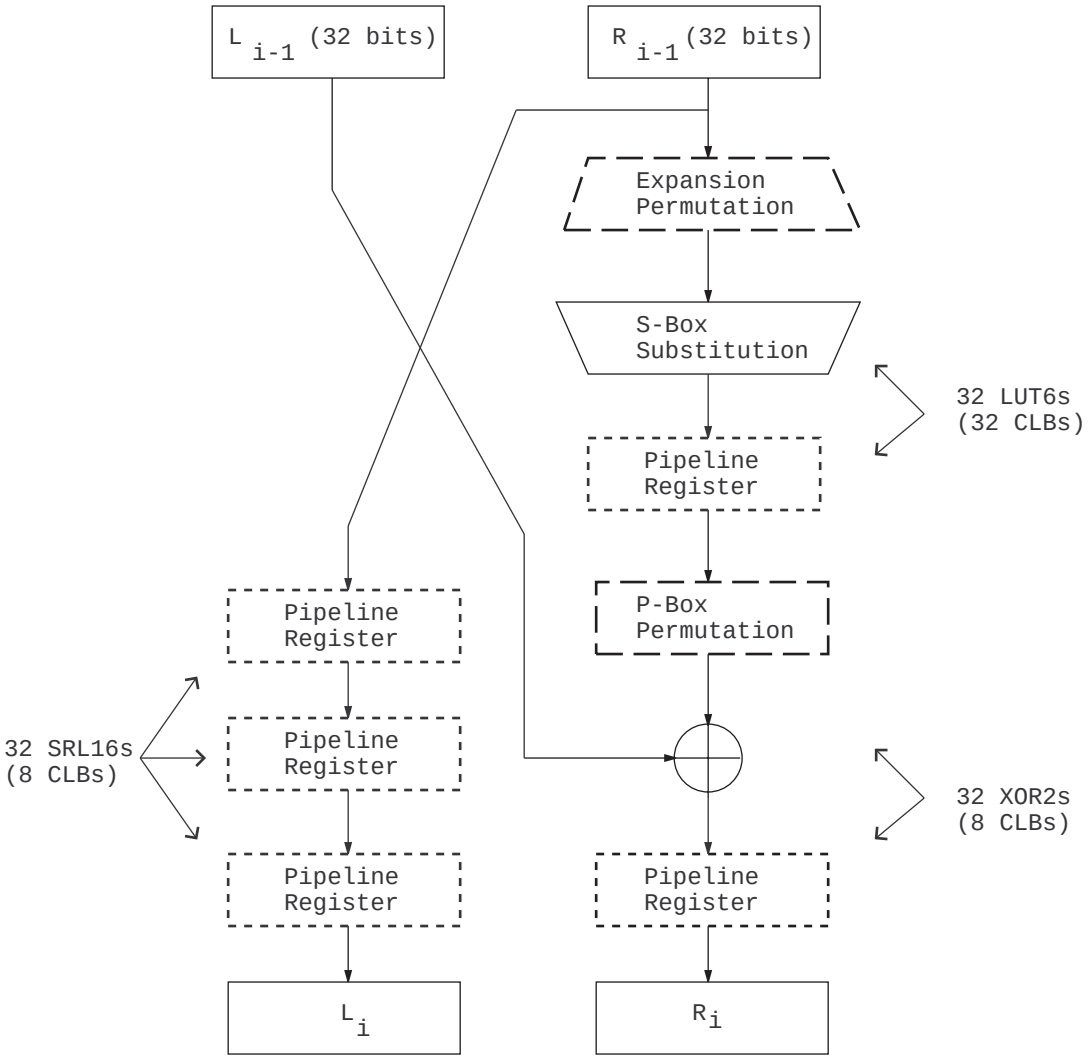

Figure 5: A Pipelined JBits DES Round

Virtex and JBits implementations of AES candidates such as Twofish [18] are underway. These algorithms are amenable to the same hardware / software partitioning as DES, and with comparable improvements in speed, density and power. The implementation results will add valuable information for the selection process [19].

Acknowledgements

Steve Guccione's development of JBits has made this work possible. Advice and assistance from the other members of the JBits project is greatly appreciated. This work was supported by the U.S. Defense Advanced Research Projects Agency, under contract DABT63-99-3-0004.

References

[1] Stephen Charlwood and Philip James-Roxby. Evaluation of the XC6200-series architecture for cryptographic applications. In Reiner W. Hartenstein and Andres Keevallik, editors, *Eighth International Workshop on Field-Programmable Logic and Applications (FPL'98)*, pages 218–227.

Springer-Verlag Lecture Notes in Computer Science, Volume 1482, Aug 1998.

[2] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., second edition, 1996.

[3] Jason Leonard and William H. Mangione-Smith. A case study of partially evaluated hardware circuits: Key-specific DES. In Wayne Luk, Peter Y.K. Cheung, and Manfred Glesner, editors, *Seventh International Workshop on Field-Programmable Logic and Applications (FPL'97)*, pages 151–160. Springer-Verlag Lecture Notes in Computer Science, Volume 1304, Sep 1997.

[4] Tom Kean and Ann Duncan. DES key breaking, encryption and decryption on the XC6216. In Kenneth L. Pocek and Jeffrey M. Arnold, editors, *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'98)*, pages 310–311, Apr 1998.

[5] http://www.memecdesign.com/product.

[6] Jens-Peter Kaps. High speed FPGA architectures for the Data Encryption Standard. Master's thesis, Department of Electrical Engineering, Worcester Polytechnic Institute, May 1998.

[7] http://www.cast-inc.com/cores/xdes.

[8] D. Craig Wilcox, Lyndon G. Pierson, Perry J. Robertson, Edward L. Witzke, and Karl Gass. A DES ASIC suitable for network encryption at 10 Gbps and beyond. In Çetin K. Koç and Christof Paar, editors, *First International Workshop on Cryptographic Hardware and Embedded Systems (CHES'99)*, pages 37–48. Springer-Verlag Lecture Notes in Computer Science, Volume 1717, Aug 1999.

[9] http://www.free-ip.com/DES/index.htm.

[10] National Bureau of Standards. *FIPS PUB 46, The Data Encryption Standard*. U.S. Department of Commerce, Jan 1977.

[11] P.R. Zimmermann. *PGP Source Code and Internals*. MIT Press, Boston, 1995.

[12] Xilinx, Inc., 2100 Logic Drive, San Jose, California. *Virtex 2.5V FPGA Series Data Sheet*, Oct 1999.

[13] Steve Guccione, Delon Levi, and Prasanna Sundararajan. JBits: Java based interface for reconfigurable computing. In *Second Annual Military and Aerospace Applications of Programmable Devices and Technologies (MAPLD'99)*, The Johns Hopkins University, Laurel, Maryland, Sep 1999.

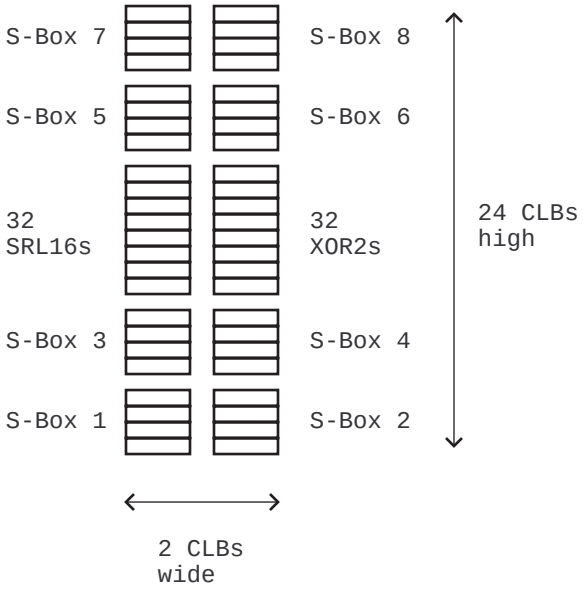

Figure 6: JBits DES Round Floorplan

[14] Eric Keller. JRoute: a run-time routing API for FPGA hardware. In *Seventh Reconfigurable Architectures Workshop (RAW 2000)*, Cancun, Mexico, May 2000. to appear.

[15] http://www.xilinx.com/products/virtex.htm.

[16] Steve Kelem. *Virtex Configuration Architecture Advanced User's Guide*. Xilinx, Inc., 2100 Logic Drive, San Jose, California, Jun 1999. Application Note 151.

[17] National Institute of Standards and Technology. Announcing request for candidate algorithm nominations for the Advanced Encryption Standard (AES). *Federal Register*, 62(117):48051–48058, Sep 1997.

[18] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. *The Twofish Encryption Algorithm*. John Wiley & Sons, Inc., 1999.

[19] http://www.nist.gov/aes.

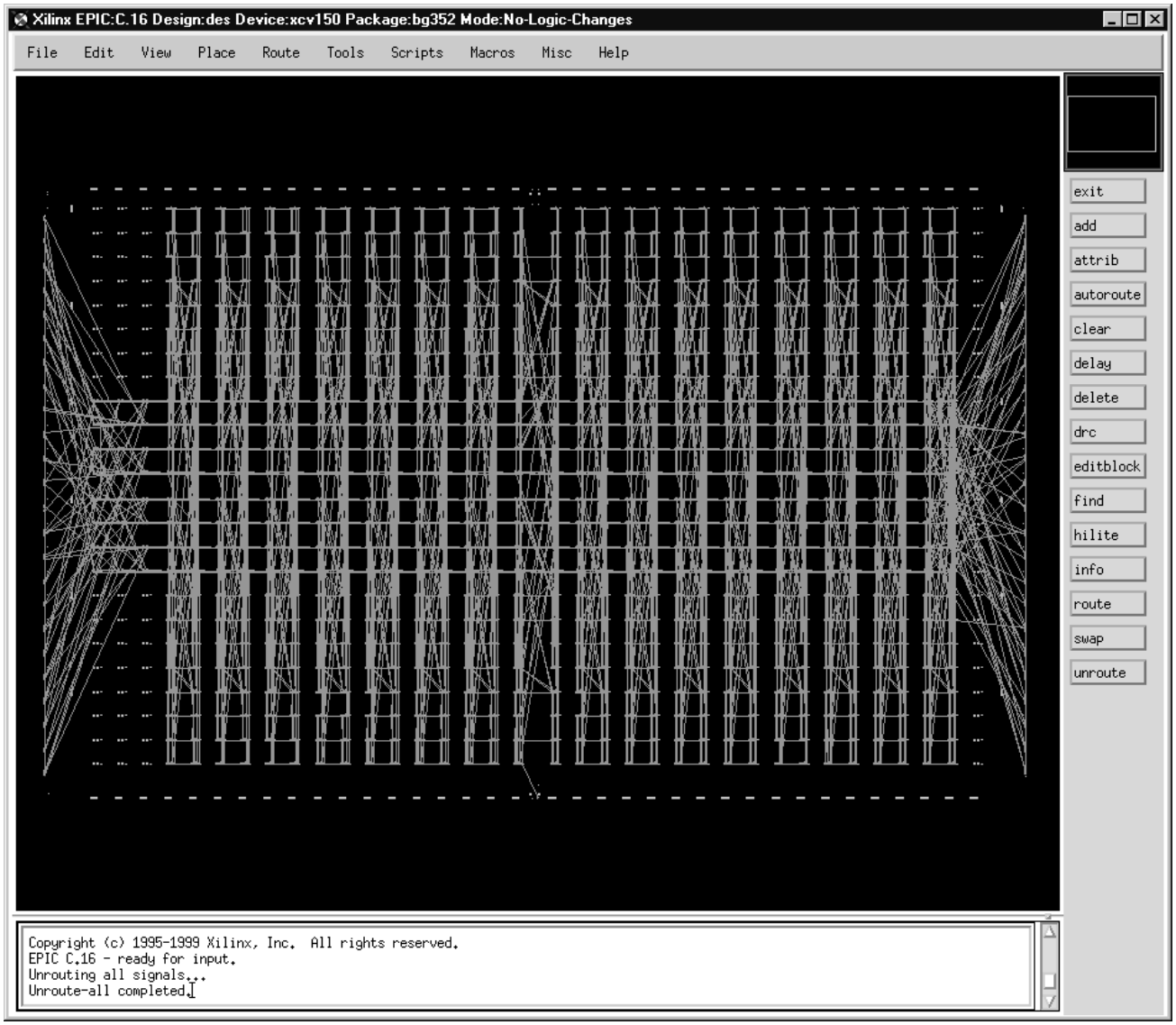

Figure 7: EPIC Ratsnest View of the 16 Rounds