A Dynamic FPGA Implementation of the Serpent Block Cipher

Cameron Patterson

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

Abstract. A JBits implementation of the Serpent block cipher in a Xilinx FPGA is described. JBits provides a Java-based Application Programming Interface (API) for the run-time modification of the configuration bitstream. This allows dynamic circuit specialization based on a specific key and mode (encrypt or decrypt). Subkeys are computed in software and treated as constants in the Serpent datapath. The resulting logic optimization produces a circuit that is half the size and twice the speed of a static, synthesized implementation. With a throughput of over 10 Gigabits per second, the JBits implementation has sufficient bandwidth for SONET OC-192c (optical) networks.

1 Introduction

The United States Department of Commerce defines a standard cryptographic algorithm for non-classified government use, and the Data Encryption Standard (DES) has fulfilled this role since 1977 [1]. DES was intended to be used for no more than 10 years, but the absence of a new standard means that it is still the workhorse private key encryption algorithm. There are, however, compelling reasons to replace DES:

- Exhaustive key search has been a serious threat to DES for several years. Triple DES extends the effective key size, but incurs a significant performance penalty.
- Cryptographic algorithms must provide high performance in software as well as hardware. DES is hardware-oriented. When implemented in software, an n-bit processor performs operations on small subsets of the bits in a word. The fastest publically available DES software that encrypts a single block at a time has a throughput of 15 Mbits/sec on a 200 MHz Pentium [2]. If 64 blocks are encrypted in parallel, then software throughput can be increased to 137 Mbits/sec [3].
- The DES 64-bit block size is insufficient for high bandwidth applications.

In 1997, the National Institute of Standards and Technology (NIST) solicited candidates for a successor to DES, which is to be called the Advanced Encryption Standard (AES) [4]. AES, like DES, will be a private key algorithm (i.e. it uses

© Springer-Verlag Berlin Heidelberg 2000

the same key for both encryption and decryption). However, AES defines key sizes of 128, 192, or 256 bits, and doubles the block size to 128 bits. Fifteen algorithms were submitted to the First AES Candidate Conference in August 1998 [5]. One year later, NIST announced the five finalists: MARS, RC6™, Rijndael, Serpent and Twofish.

Field-Programmable Gate Arrays are frequently used to implement cryptographic algorithms [6]. The author previously implemented DES in the Xilinx Virtex™ family [7]. An electronic codebook (ECB) mode throughput in excess of 10 Gbits/sec was achieved using JBits to perform key-specific dynamic circuit specialization [8]. According to the National Security Agency, high speed network encryption may require the use of non-feedback modes such as ECB [9].

Key-specific circuit specialization removes a logic level and the associated routing from the critical path in a DES round. Compared with a static design that uses the same Virtex process and degree of pipelining, speed is increased by over 50%. The reduction in circuit size also decreases power consumption and permits the use of smaller devices. Cheaper packages may be used, since key input pins are no longer required. The resulting implementation has better throughput and lower power than the fastest reported DES ASIC [10]. High volume cost in the Spartan™-II family should be competitive with ASICs.

The author is developing dynamic implementations of the AES finalists, which can provide additional data for the selection process. Serpent was chosen first for several reasons:

– It uses many of the same primitive operations as DES. This allows some of the DES building blocks to be reused.
– The simple round structure means that high clock rates can be achieved with only one pipeline stage per round.
– Decryption requires inverse permutations and linear transformations, which would normally increase the size of a static implementation. A dynamic implementation, however, simply reconfigures the same FPGA resources when either the key or mode changes.
– Speed and size comparisons can be made with a static Virtex design [11].

A dynamic Serpent implementation is twice the speed and half the size of the static design. If both encryption and decryption are required, then the dynamic circuit requires an even smaller fraction of the FPGA resources. ECB mode data throughput is over 10 Gbits/sec. This is the same bandwidth as the dynamic DES design, and is achieved with a similar degree of pipelining, twice the data path width and half the clock rate. Power consumption for Serpent should be about 4 watts, compared with 3.2 watts for DES. Hence, the JBits approach is providing the same performance and cost benefits for Serpent as it did for DES. The remainder of this paper applies dynamic circuit specialization to Serpent.

2 The Serpent Algorithm

Serpent is a substitution-permutation (SP) network that uses 32 rounds. The output of round i is the input to round $(i + 1)$. The algorithm has two different modes of implementation: standard and bitslice. The standard mode operates on individual bits or groups of four bits, while the bitslice mode improves software efficiency by operating on entire 32-bit words. Serpent software using the bitslice optimization encrypts at roughly 32 Mbits/sec on a 200 MHz Pentium [2].

The bitslice mode was chosen for both software and hardware implementation, since:

– It does not increase the number of lookup tables (LUTs) in the critical path.
– Layout considerations encourage the partitioning of 128-bit data paths into four 32-bit words.
– Like DES, the standard Serpent mode requires permuting the input bits to the first round, and permuting the output bits from the final round. Subkey permutations are also required. Permutations on 128-bit buses consume significant routing resources in an FPGA device, and can increase the routing delay.

Serpent's bitslice mode is to be assumed in the following exposition.

2.1 The Round Function

The rounds are numbered from 0 to 31. Round i has input B_i and output B_{i+1}. The round function is defined as:

$$\begin{aligned} B_{i+1} &= L(S_i(B_i \oplus K_i)) \qquad \text{for } i = 0, \ldots, 30 \\ B_{32} &= (S_{31}(B_{31} \oplus K_{31})) \oplus K_{32} \end{aligned}$$

S_i applies 32 copies of a 4-bit permutation called an S-box. Eight different S-boxes are used, which are derived from the DES S-boxes. Round i applies S-box (i mod 8), so that each S-box is used in four different rounds. The generation of the 128-bit round subkeys K_i is described in Section 2.2. L is a linear transformation. In bitslice mode, it applies left rotations (<<<), left shifts (<<) and XORs ($\oplus$) to the 32-bit operands X_0, X_1, X_2, and X_3:

$$\begin{aligned} X_0, X_1, X_2, X_3 &:= S_i(B_i \oplus K_i) \\ X_0 &:= X_0 <<< 13 \\ X_2 &:= X_2 <<< 3 \\ X_1 &:= X_1 \oplus X_0 \oplus X_2 \\ X_3 &:= X_3 \oplus X_2 \oplus (X_0 << 3) \\ X_1 &:= X_1 <<< 1 \\ X_3 &:= X_3 <<< 7 \\ X_0 &:= X_0 \oplus X_1 \oplus X_3 \\ X_2 &:= X_2 \oplus X_3 \oplus (X_1 << 7) \\ X_0 &:= X_0 <<< 5 \\ X_2 &:= X_2 <<< 22 \\ B_{i+1} &:= X_0, X_1, X_2, X_3 \end{aligned}$$

2.2 The Key Schedule

If required, the user-supplied key is first padded to 256 bits. This is done by assigning a 1 to the most significant bit, and a 0 to the remaining bits. The key is stored as eight 32-bit words $w_{-8}, \ldots, w_{-1}$. This is used to generate the prekeys $w_0, w_1, \ldots, w_{131}$ with the recurrence:

$$w_i := (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \phi \oplus i) <<< 11$$

where ϕ is the hexadecimal constant 9E3779B9.[1] Finally, round subkeys are computed from the prekeys by applying the S-boxes as follows:

$$\begin{aligned}
K_0 &:= S_3(w_0, w_1, w_2, w_3) \\
K_1 &:= S_2(w_4, w_5, w_6, w_7) \\
K_2 &:= S_1(w_8, w_9, w_{10}, w_{11}) \\
&\vdots \\
K_{31} &:= S_4(w_{124}, w_{125}, w_{126}, w_{127}) \\
K_{32} &:= S_3(w_{128}, w_{129}, w_{130}, w_{131})
\end{aligned}$$

The structure of the rounds and the application of the subkeys are shown in Figures 1 and 2.

2.3 Decryption

Feistel networks such as DES decrypt by simply applying the round subkeys in reverse order. Serpent, however, is not a Feistel network. It also requires applying the inverse of the S-boxes in reverse order, and inverting the linear transformation. Although this represents an area overhead for ASICs and static FPGAs, the same logic and routing resources can simply be reconfigured with a dynamic implementation.

3 Run-Time Reconfiguration

The run-time optimization of FPGAs to the problem instance at hand can have considerable speed and area advantages. For example, a dynamic implementation of the DES algorithm exceeds the performance of the fastest known DES ASIC. In this case, the reduction in circuit complexity more than compensates for the routing overheads associated with FPGAs.

However, most systems do not exploit the run-time reconfiguration (RTR) of SRAM-based FPGAs. This is primarily because there is no support for RTR in the standard design capture, verification and implementation tools. The conventional netlist-based FPGA design flow is the same as for ASICs, and has far too much time and memory overhead to be used in real-time or embedded environments.

[1] This is the fractional part of the golden ratio $(1 + \sqrt{5})/2$.

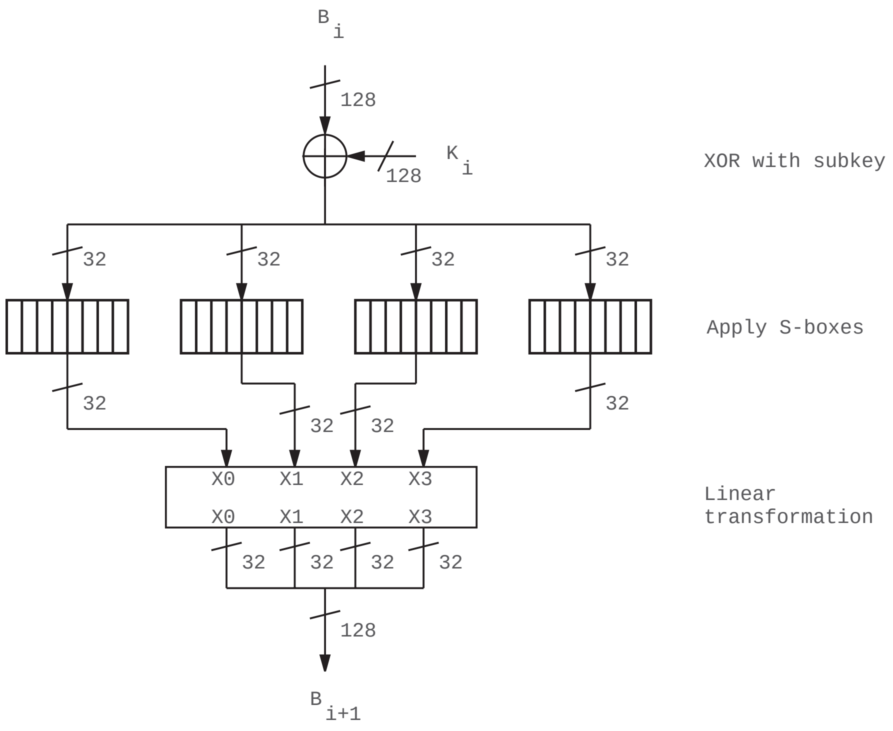


Fig. 1. Structure of Serpent Rounds 0 to 30

RTR is most easily controlled with a microprocessor. Many systems already make use of one or more microprocessors for those operations that do not require hardware speeds. Software can directly create or modify the FPGA's configuration with a suitable Application Programming Interface (API). This model readily supports hardware/software co-design, since the integration of hardware and software occurs early in the development effort.

3.1 JBits

Xilinx provides a Java-based configuration API for the XC4000 and Virtex architectures called JBits [12]. The tiles used in an architecture are defined as JBits classes. For example, Virtex has a Configurable Logic Block (CLB) tile that includes the associated General Routing Matrix (GRM). A CLB instance is treated as an object that is referenced by a row and column. The state of the configurable structures in the CLB can be queried or modified. Figure 3 illustrates the JBits calls required to configure a CLB at (`row`,`col`) as a 4-input 4-output

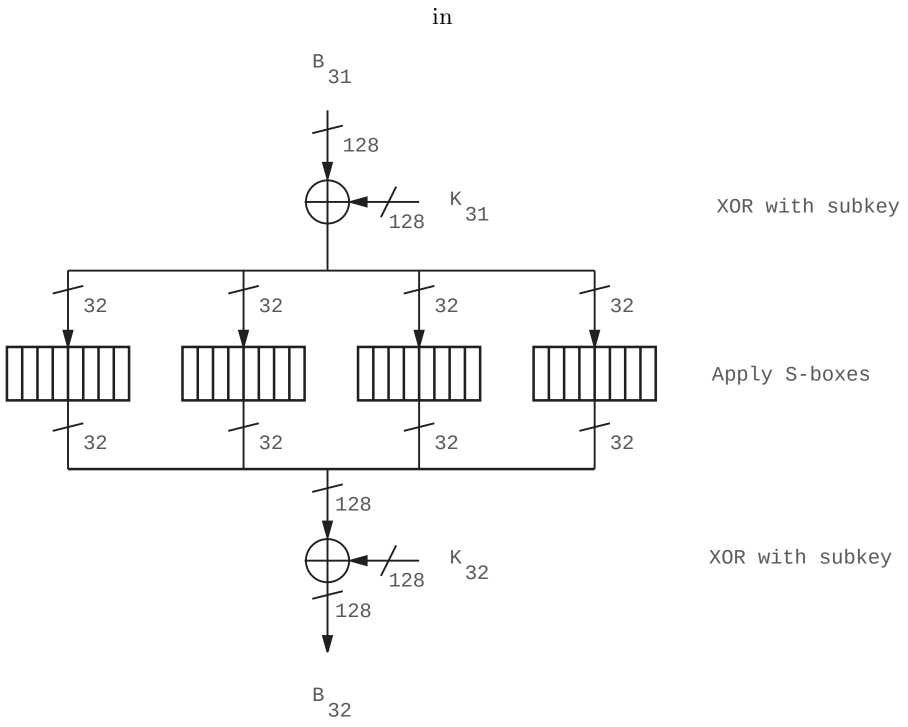


Fig. 2. Structure of Serpent Round 31

registered S-box. There is less code than the equivalent constrained structural netlist of Virtex primitives defined in an HDL. Many of the configuration settings used are the power-up defaults.

Generating a configuration bitstream with JBits generally takes on the order of seconds, compared with minutes or even hours for the Xilinx M2.1 implementation tools. JBits is a physical design tool, and avoids the optimization problems that arise during the logical to physical transformation in conventional CAD flows. All mapping, placement and routing is fully specified in a JBits design.

4 Serpent Implementation

In 1998, Xilinx introduced the Virtex architecture as the successor to the XC4000 family [13]. 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 carry path). System-level resources such as block RAM and delay-locked loops have also been added. Unlike the XC4000, Virtex supports packet-based partial reconfiguration [14].

```
public void configureSBox(int row, int col) {
  try {
    jbits.set(row, col, S0RAM.DUAL_MODE,     S0RAM.ON);
    jbits.set(row, col, S0RAM.LUT_MODE,      S0RAM.ON);
    jbits.set(row, col, S0Control.X.X,       S0Control.Cin.FOUT);
    jbits.set(row, col, S0Control.Y.Y,       S0Control.Y.GOUT);
    jbits.set(row, col, S0Control.XDin.XDin, S0Control.XDin.X);
    jbits.set(row, col, S0Control.YDin.YDin, S0Control.YDin.Y);
    jbits.set(row, col, S0Control.LatchMode, S0Control.ON);
    jbits.set(row, col, S0Control.Sync,      S0Control.ON);
    jbits.set(row, col, S1RAM.DUAL_MODE,     S1RAM.ON);
    jbits.set(row, col, S1RAM.LUT_MODE,      S1RAM.ON);
    jbits.set(row, col, S1Control.X.X,       S1Control.Cin.FOUT);
    jbits.set(row, col, S1Control.Y.Y,       S1Control.Y.GOUT);
    jbits.set(row, col, S1Control.XDin.XDin, S1Control.XDin.X);
    jbits.set(row, col, S1Control.YDin.YDin, S1Control.YDin.Y);
    jbits.set(row, col, S1Control.LatchMode, S1Control.ON);
    jbits.set(row, col, S1Control.Sync,      S1Control.ON);
  } catch (ConfigurationException ce) {
    System.out.println("S-box configuration error at R" +
                       row + "C" + col);
    System.out.println(ce);
  }
}
```

Fig. 3. Implementing a Registered S-box with JBits

The Virtex CLB is well-suited to the Serpent algorithm. An S-box is specified as a table with a 4-bit input and a 4-bit output. Logic minimization algorithms find little structure in the S-boxes, so it is reasonable to implement an S-box as a set of four single-output LUTs that are driven by the same four inputs. A single Virtex CLB can implement all four LUTs, while the XC4000 architecture would require two CLBs.

In addition, the Virtex segmented routing structure efficiently implements the shift and rotation operations in Serpent's linear transformation stages. Each Virtex horizontal and vertical channel has 96 hex-length and 24 single-length segments, which provides high speed and high density wire permutations.

The same approach is used for Serpent as for DES, namely to investigate the speed, size and power optimizations achievable with dynamic circuit specialization. Two additional considerations also influenced the design:

– An effort was made to make a fair comparison with a static Virtex implementation of Serpent. Both designs use only one pipeline stage per round. The static design includes support for cipher block chaining mode [15], and the dynamic design has resources available for the required XOR gates.
– Wherever practical, the Serpent core was designed to be compatible with the JBits DES core. This was largely achieved for throughput, degree of

pipelining and power consumption, despite the fact that Serpent has twice the datapath width and number of rounds compared with DES. Note that Serpent's larger key size does not increase the pin requirement, since the JBits approach does not require any key-handling circuitry or IOBs.

4.1 Dynamic Circuit Specialization Using JBits

Serpent's key scheduling logic is more complicated than DES, but the datapath is simpler. This results in even greater benefits when the datapath is implemented in hardware and the key schedule is precomputed in software. The logical operations performed in the datapath are permutations on groups of 4 bits, and XORs. As shown in Figure 1, the first 31 rounds require 128 XORs between the data and subkey bits. Let b be a data bit from B_i and k be the corresponding subkey bit from K_i. Since k is a constant for a given encryption key, XOR(b, k) is either b or $\overline{b}$. 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 result of the optimization is shown in Figure 4.

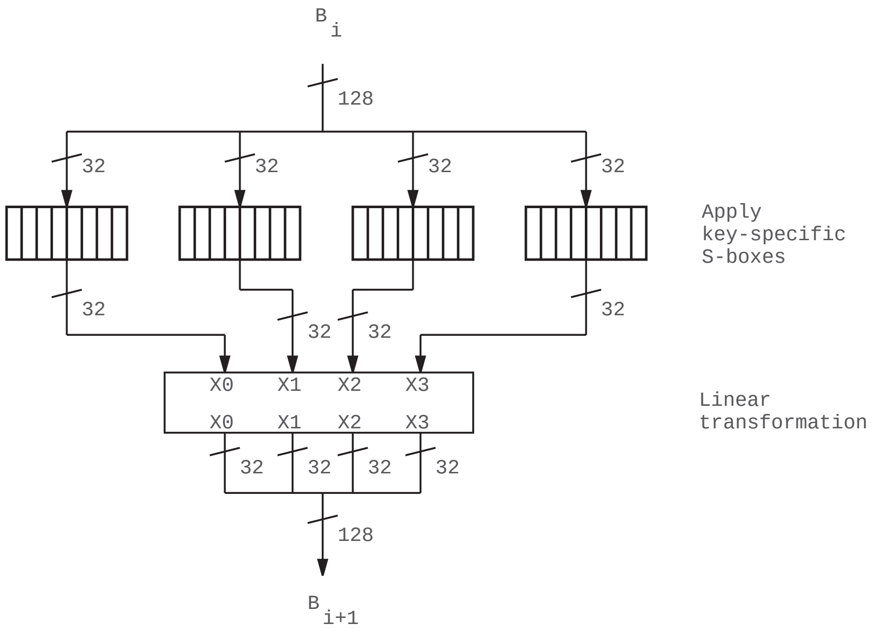


Fig. 4. Optimization of Serpent Rounds 0 to 30

The final round, shown in Figure 2, replaces the linear transformation with an additional XOR between the datapath and subkey K_{32}. These 128 XOR gates can also be folded into the round's S-boxes. Again let s be an output bit from the S-boxes, and k be the corresponding subkey bit form K_{32}. Each XOR results in either s or $\overline{s}$. An inversion on an S-box output is effected by inverting the contents of the LUT. As shown in Figure 5, the optimized final round contains only S-boxes.

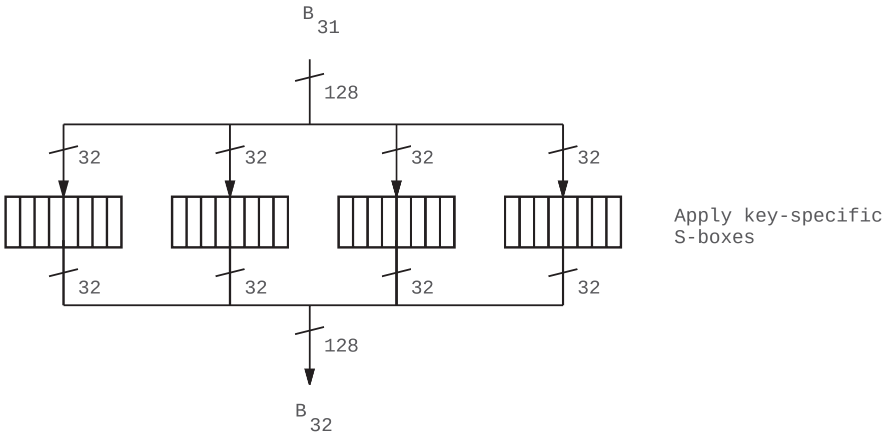


Fig. 5. Optimization of Serpent Round 31

Dynamic specialization has removed logic, register and routing resources for 4224 two-input XOR gates and 4224 subkey bits. This accounts for the two-fold reduction in circuit area compared with the static design. IOBs are no longer required for subkey loading. One of the four logic levels and the associated routing has also been removed from the critical path between pipeline registers, which is largely responsible for the speed improvement.

The static design uses an XCV1000BG560 Virtex part, although the authors indicate that it should fit in an XCV800. By contrast, the dynamic design targets the significantly less expensive XCV400BG432. Routing delays are not reduced by using a larger part, because the mapping and placement is fixed. Subkey precomputation is performed outside the FPGA in both designs, and could be performed by similar software environments.

4.2 High-Level JBits Code

The Serpent design is defined as a JBits run-time parameterized core (RTPCore). An RTPCore can generate a bitstream directly. It can also generate EDIF, for integration with logic simulators and the NGD/NCD-based Xilinx implementation tools. The EDIF flow was used for validation and timing analysis of the Serpent design.

An RTPCore supports hierarchy with ports and subcores. The subcore's ports are connected with nets and buses. Depending on whether a bitstream or EDIF is being generated, a primitive RTPCore either makes JBits calls to configure logic and routing resources, or creates a netlist of components from the Virtex SIMPRIM library.

The constructor for the Serpent module is shown in Figure 6. The arguments are the hexadecimal value of the encryption key, and the external nets and buses to be connected to the clock, B_0 and B_{32} ports. For simplicity, encryption is assumed. An initial key is required to support the static EDIF flow. Note that run-time modification of the encryption key does not require the core to be recreated, since the key only affects the contents of LUTs.

```
public class Serpent extends RTPCore {
  public Serpent(String hexKey, Net clk, Bus din, Bus dout) {
    byte[]   key    = Key.fromHexString(hexKey);
    int[]    preKey = Key.computePreKey(key);
    byte[][] subKey = Key.computeSubKey(preKey);

    Port clkPort  = newInputPort("clk", clk);
    Port dinPort  = newInputPort("din", din);
    Port doutPort = newOutputPort("dout", dout);
    Net clkInt = newNet("clk");
    Bus[] b = new Bus[NUM_ROUNDS+1];
    for (int i = 0;  i < b.length;  i++)
      b[i] = newBus("b" + i, DATAPATH_WIDTH);
    clkPort.setIntSig(clkInt);
    dinPort.setIntSig(b[0]);
    doutPort.setIntSig(b[NUM_ROUNDS]);

    SerpentRound[] round = new SerpentRound[NUM_ROUNDS];
    for (int i = 0;  i < LAST_ROUND;  i++)
      round[i] = new SerpentRound(i, subKey[i], clkInt, b[i], b[i+1]);
    round[LAST_ROUND] = new SerpentRound(subKey[LAST_ROUND],
                                         subKey[LAST_ROUND+1], clkInt,
                                         b[LAST_ROUND], b[LAST_ROUND+1));
  }
}
```

Fig. 6. Serpent Constructor

The key is first converted to binary, then to the prekey array of 132 32-bit words, and finally to a subkey array of 33 128-bit words. Ports are defined and connected to the external and internal signals. An array is created to reference the 32 `SerpentRound` submodules, which are instantiated with the computed subkey(s) and connected to the clock net and the data buses. The `configureSBox` method (shown in Figure 3) is called 32 times by each `SerpentRound` instance.

The ports, nets and buses are used by both JRoute [16] and the EDIF generator. JRoute is a routing API built upon JBits, and selects and configures the routing resources necessary to make connections between CLBs. The core need only specify the end points of the connections as logical ports, which are translated to physical CLB pins once the module is placed. JRoute provides a significant abstraction layer for physical routing, since the core need not specify the detailed sequence of routing resources needed to complete a route. This can still be done, however, if JRoute does not select the required path.

4.3 Layout

Rounds 0 to 30 require 32 CLBs for the S-boxes, and 32 CLBs for the linear transformation. The final round does not perform a linear transformation, and is implemented entirely with S-boxes. As shown in Figure 7, an 8×8 array of CLBs is used for the non-final rounds. All 32 rounds fit within the 40×60 CLB array of an XCV400 device. Every LUT is used in the 2016 CLBs that implement the 32 rounds.

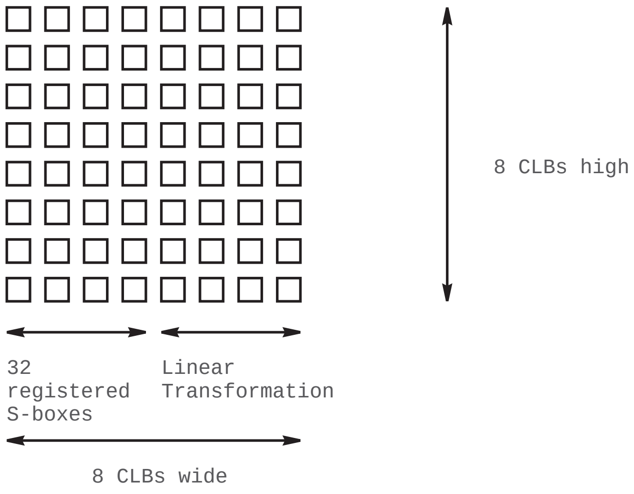
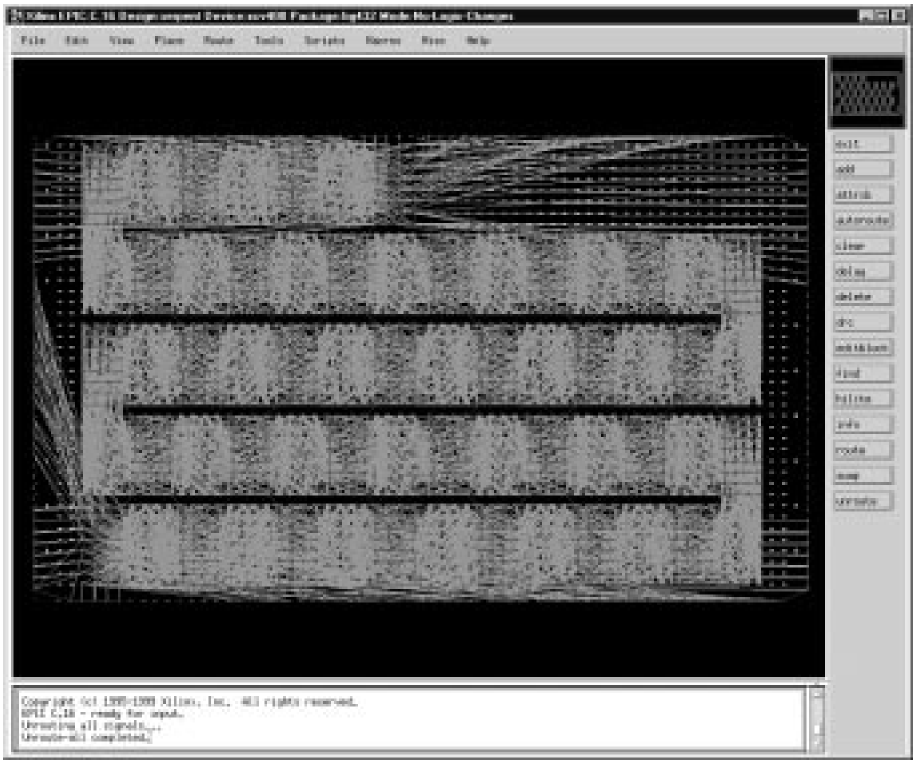


Fig. 7. Floorplan for Rounds 0 to 30

RTPCores have a mechanism similar to placement directives [17] for assigning coordinates to relocatable modules. This is used to create the floorplan in Figure 7, and to define the folded datapath for the 32 rounds. Beginning in the lower left corner, the rounds are placed horizontally in a zigzag pattern with alternating left-to-right and right-to-left dataflows. This snake-like layout can be seen in Figure 8.



```
Copyright (c) 1995-1999 Xilinx, Inc. All rights reserved.
EPIC C.16 - ready for input.
Unrouting all signals...
Unroute-all completed.
```

Fig. 8. EPIC Ratsnest View of the 32 Rounds

The design uses 8064 LUTs, 4352 flip flops, and 257 IOBs. All of the S-box CLBs and data IOBs are registered. Hence the total number of pipeline stages from input to output pins is 34. Slice utilization is 84% (4032 slices used from the 4800 available in an XCV400 device). A total of 384 CLBs and 60 IOBs are available for additional interface circuitry.

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 a reference software implementation [2].

Table 1 reports the speed achieved for fully pipelined Serpent implementations. Both the static and dynamic FPGA designs use the same implementation technology (Virtex -4 speed grade) and tools (M2.1). The ASIC performance is estimated by the National Security Agency for a MOSIS 0.5 micron process [9]. High volumes are required before an ASIC can justify using a process technology similar to Virtex.

The bulk of the performance improvement is achieved with dynamic circuit specialization, although floorplanning also contributes. Regular floorplans make reconfiguration more efficient. Power consumption for the XCV400-4 dynamic design is estimated at 4 watts, and is not reported for the static design. Note that the highest speed grade XCV400 in a BG432 package may be less expensive than the lowest speed grade XCV1000 or XCV800 in a BG560 package.

Table 1. Performance Results

Technology Used	Package	Design Methodology	Floorplanned	Clock Rate (MHz)	Throughput (Gbits/sec)
Xilinx V1000-4	BG560	static	no	37.97	4.86
ASIC		static		62.73	8.03
Xilinx V400-4	BG432	dynamic	no	75.48	9.66
Xilinx V400-4	BG432	dynamic	yes	80.27	10.27
Xilinx V400-6	BG432	dynamic	yes	101.44	12.98
Xilinx V400E-8	BG432	dynamic	yes	137.15	17.55

In order to change the key or switch between encryption and decryption, 56 of the CLB columns have to be reconfigured (i.e. about 85% of the 2,546,080 configuration bits in an XCV400). Using the 8-bit wide Virtex SelectMap™ port running at 50 MHz, this reconfiguration is accomplished in under 10 milliseconds. Assuming the use of a high-end microprocessor, computing new subkeys and updating the LUTs with JBits calls also requires about 10 milliseconds. This is roughly the same as the time required for other system operations such as disk I/O. If there is no switching between encryption and decryption, an offset folding of the Serpent datapath can reduce the number of reconfigured CLB columns by 50%.

5 Conclusions

A dynamic implementation of the Serpent block cipher in a Virtex FPGA has been presented. It achieves a throughput of over 10 Gbits/sec, which is about 100 times faster than software implementations on high-end microprocessors. When compared with a static Virtex implementation, the dynamic circuit is over twice the speed and half the size. Power consumption and the number of package pins required is also reduced. This combination of factors result in significant cost savings.

Creating the key schedule is much more complicated for the AES finalists than for DES, and suits software implementation. The size of the key, and the time required to compute the subkeys, makes a high degree of key agility—such as changing the key on every clock cycle—difficult to achieve. Given this characteristic of the AES algorithms, FPGA reconfiguration overhead is less significant compared with DES.

As shown by code fragments in this paper, JBits provides several levels of abstraction for defining circuits. The lowest level JBits class provides complete configuration control, while the RTPCore class allows connectivity to be specified in terms of ports, nets and buses. Placement directives and JRoute help to bridge these levels. For systems that are partitioned between hardware and software, this single-language approach greatly reduces the integration effort.

Acknowledgements

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

References

1. National Bureau of Standards. *FIPS PUB 46, The Data Encryption Standard.* U.S. Department of Commerce, Jan 1977.
2. Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A proposal for the Advanced Encryption Standard. `http://www.cl.cam.ac.uk/~rja14/serpent.html`.
3. Eli Biham. A fast new DES implementation in software. In *Fourth International Workshop on Fast Software Encryption (FSE'97)*, pages 260–271. Springer-Verlag Lecture Notes in Computer Science, Volume 1267, 1997.
4. 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.
5. `http://www.nist.gov/aes`.
6. 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.
7. Cameron Patterson. High performance DES encryption in Virtex FPGAs using JBits. In Kenneth L. Pocek and Jeffrey M. Arnold, editors, *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2000)*, Apr 2000.
8. 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.

9. Bryan Weeks, Mark Bean, Tom Rozylowicz, and Chris Ficke. Hardware performance simulations of round 2 Advanced Encryption Standard algorithms. In *Third AES Candidate Conference (AES3)*, Apr 2000.
10. 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.
11. A.J. Elbirt and C Paar. An FPGA implementation and performance evaluation of the Serpent block cipher. In *ACM Eighth International Symposium on Field Programmable Gate Arrays (FPGA 2000)*, pages 33–40, Feb 2000.
12. 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.
13. Xilinx, Inc., 2100 Logic Drive, San Jose, California. *Virtex 2.5V FPGA Series Data Sheet*, Oct 1999.
14. Steve Kelem. *Virtex Configuration Architecture Advanced User's Guide.* Xilinx, Inc., 2100 Logic Drive, San Jose, California, Jun 1999. Application Note 151.
15. Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, Inc., second edition, 1996.
16. Eric Keller. JRoute: a run-time routing API for FPGA hardware. In *Seventh Reconfigurable Architectures Workshop (RAW 2000)*, pages 874–881. Springer-Verlag Lecture Notes in Computer Science, Volume 1800, May 2000.
17. James Hwang, Cameron Patterson, S. Mohan, Eric Dellinger, Sujoy Mitra, and Ralph Wittig. Generating layouts for self-implementing modules. In Reiner W. Hartenstein and Andres Keevallik, editors, *Eighth International Workshop on Field-Programmable Logic and Applications (FPL'98)*, pages 525–529. Springer-Verlag Lecture Notes in Computer Science, Volume 1482, Aug 1998.