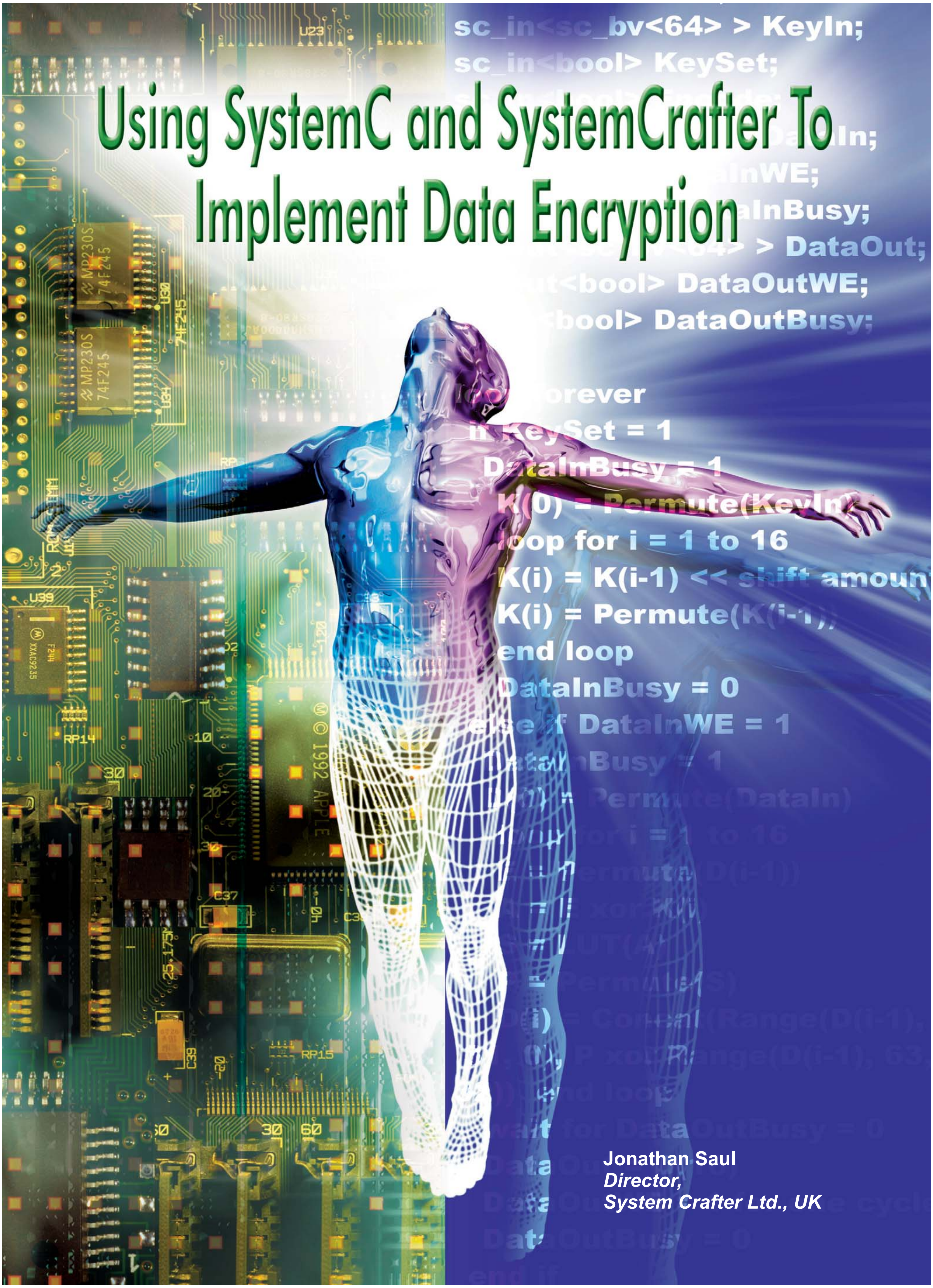
Using SystemC and SystemCrafter To Implement Data Encryption



Jonathan Saul
Director,
System Crafter Ltd., UK

SystemCrafter provides a simple route for high-level FPGA design

For many years people have been using C or C++ as a starting point for developing their hardware and systems. This is because these languages are widely known, quick to write, and give an executable specification, which allows very fast simulation. C or C++ versions of standard algorithms are widely available, which allows easy reuse of legacy and publicly available code. For system-level design, they allow hardware and software descriptions to be described in a single framework.

However there have been two drawbacks. Firstly, C and C++ don't support the description of some important hardware concepts, such as timing and concurrency. This has led to the development of proprietary C-like languages, which haven't been popular because they tied the user to a single software supplier. Secondly, C and C++ have to be translated manually to a hardware description language, such as VHDL or Verilog, for hardware implementation. This step requires specialist resources, is time-consuming, and often introduces errors that are difficult to find.

The first of these problems was solved by the development of SystemC, which is now a widely-accepted industry standard that adds hardware concepts to C++.

The second of these problems is solved by the development of tools like SystemCrafter SC, which allows SystemC descriptions to be automatically translated to VHDL or Verilog.

This article illustrate the use of SystemCrafter SC and SystemC by describing an implementation of the popular DES encryption algorithm. It first describes SystemC, SystemCrafter and DES. Then it illustrates the use of SystemCrafter in a typical design flow, using DES as an example.

SystemC

SystemC provides an industry standard means of modeling and verifying hardware and systems using standard software compilers. All the material required to simulate SystemC using a standard C++ compiler, such as Microsoft Visual C++ or GNU GCC, can be downloaded free of charge from the SystemC website (www.systemc.org)
.

SystemC consists of a set of class libraries for C++ that describe hardware constructs and concepts. This means that you can develop cycle-accurate models of hardware, software and interfaces, which can be simulated and debugged within your existing C++ development environment.

SystemC allows the initial design, debugging and refinement to be performed using the same test benches, which eliminates translation errors, and allows fast, easy verification.

Since SystemC uses standard C++, the productivity benefits offered to software engineers for years are now available to hardware and system designers. SystemC is more compact than VHDL or Verilog and, as a result, is faster to write and more maintainable and readable. It can be compiled into a fast, executable specification.

SystemCrafter SC

SystemC was originally developed as a system modeling and verification language, and still required manual translation to a hardware description language to produce hardware.

SystemCrafter SC automates this process, by quickly synthesizing SystemC to RTL VHDL or Verilog. It will also generate a SystemC description of the synthesized circuit, which can be used to verify the synthesized code using your existing test harness.

SystemCrafter SC gives the designer control of the critical steps of scheduling (clock cycle allocation) and allocation (hardware reuse). Thus, the results are always predictable, controllable and match the designer's expectations.

SystemCrafter SC allows you to develop, refine, debug and synthesize hardware and systems within your existing C++ compiler's development environment. You can run fast, executable SystemC specifications to verify your design. You can configure your compiler so that SystemCrafter SC is automatically run when you specify that you want to generate hardware.

SystemCrafter SC can be used for:

» Synthesizing SystemC to Hardware

» System-level Design and Co-design

» Custom FPGA Co-processing and Hardware Acceleration

The DES Algorithm

The Data Encryption Standard (DES) algorithm encodes data using a 64 bit key. Refer Figure 1. The same 64 bit key is required to decode the data at the receiving end. It is a well-proven, highly secure means of transmitting sensitive data. DES is particularly suitable for hardware implementation as it requires only simple operations such as bit permutation (which is particularly expensive in software), exclusive OR and table look up operations.

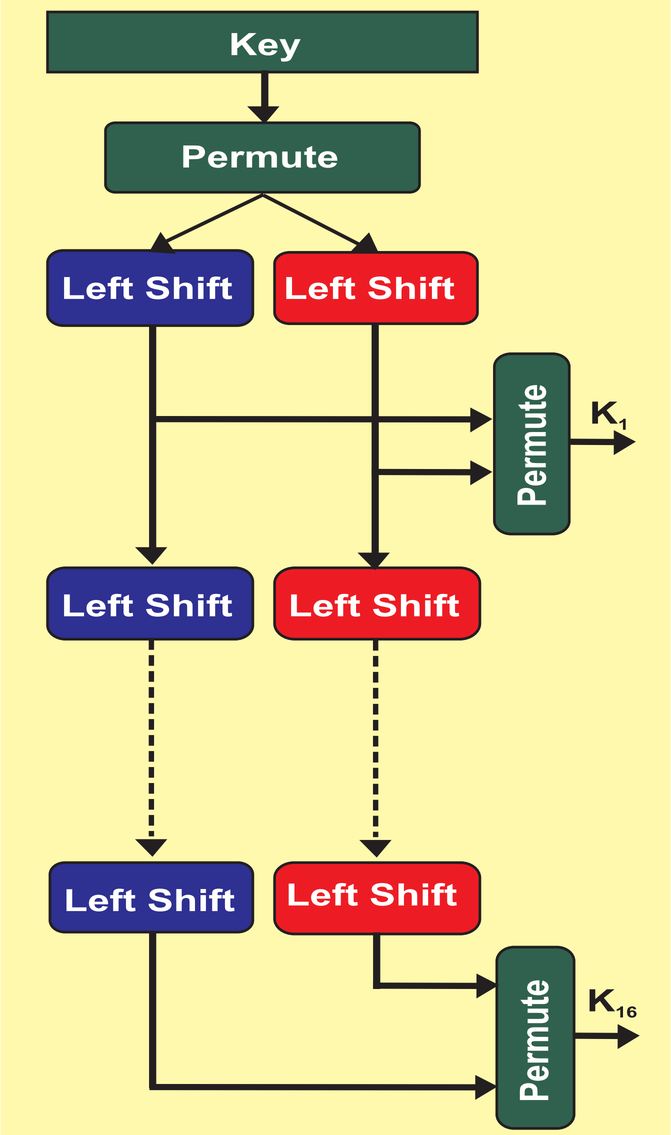


Figure 1: Circuit for pre-calculating the DES key values.

An implementation of DES consists of two stages. During the first stage, 16 intermediate values are pre-computed based on the initial key. Figure 1 shows the calculations used during this phase. These 16 values are fixed for a particular key value and may be reused for many blocks of data. To calculate the key values, the bits of the initial key are first re-ordered. This first permutation also drops the 8 bits in the key used for parity checking resulting in 56 active key bits. Each intermediate key value is then computed by first splitting the output of the previous stage into two halves of 28 bits each and shifting each half left by 1 or 2 bits depending on the iteration number. The shifted value is permuted once again resulting in a 48 bit value for use during the encryption/decryption stage.

The second computation stage involves 16 iterations of a circuit where each iteration uses one of the pre-computed key values. Figure 2 shows the calculations used during this phase. Encryption is based on 64 bit 'blocks' of data with 64 bits of input data encoded for each group of 16 iterations resulting in 64 bits of output data. The 64 bits of input data are first re-ordered by a permutation. The data is then split into two halves of 32 bits each. At each stage, 32 bits of data are permuted and expanded to 48 bits before being exclusive OR-ed with one of the 48 bit key values from stage 1. The result of the exclusive OR is split into eight 6 bit values which are used to look up eight 4 bit values from eight different look up tables (called S-blocks). The outputs of the look up operations are permuted once again and exclusive OR-ed with the other half of the input data. The two halves of the data are then reversed before starting the next iteration.

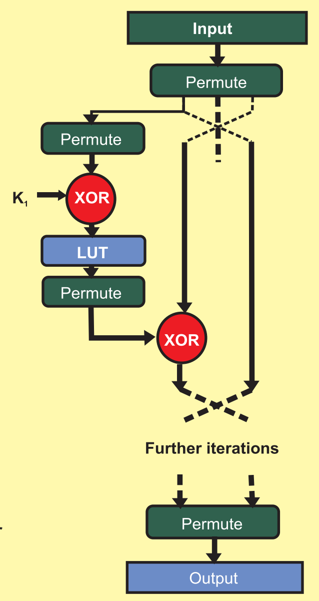


Figure 2: Circuit for encoding/decoding data.

Decryption consists of using exactly the same stage 2 calculations but with the 16 key values from the first stage used in reverse order.

A full description of the algorithm can be found at www.systemcrafter.com.

Design Flow

The design flow using SystemC and SystemCrafter is shown in Figure 3. An important benefit of this design flow is that the development of the initial SystemC description, partitioning, and system- and gate-level simulation can all be carried out in the same framework. In this case Microsoft's Visual C++ design environment was used.
The target platform was the ZestSC1, an FPGA board containing a Xilinx Spartan-3 FPGA and some SRAM, communicating with a host PC via a USB interface.

Hardware/Software Partitioning

The first step is to write an initial system-level description. It may be appropriate to decide on a hardware/software partitioning at this point, or to defer this decision until a working system-level description has been written.

In the case of DES the hardware/software partitioning decision is easy and intuitive. The core of the algorithm involves high-bandwidth computation using operators unsuitable for software implementation, and so these will be implemented on the FPGA.

A hardware description of the DES core was written in SystemC, using Microsoft's Visual C++ design environment. The use of Visual C++ allowed the programmer to use an environment that he was familiar with. Calls to SystemCrafter are made as “Custom Build Steps”, which allows simulation and synthesis from within the Visual C++ design environment. Alternatively the user could have used the SystemCrafter GUI to manage his project.

The look-up tables were implemented as CoreGen modules, to show how complex third party IP blocks can be used in a SystemCrafter design. SystemCrafter treats SystemC modules with missing method definitions as black boxes, which allows IP blocks written in other languages to be used. A model of the look-up tables was written in SystemC for use during simulation.

Simulation and Synthesis

The SystemCrafter flow allows the design to be simulated at a number of levels.

First a test harness was written, which fed standard DES test patterns through the hardware engine and displayed the results on the console.

Two configurations were specified in Visual C++: System-level and Gate-level.

The System-level configuration was used first. This compiles the original SystemC design, to produce a simulation model at the behavioral-level. This model is an executable program that can be run to produce a behavioral-level simulation of the DES engine.

Once the behavioral model produced the desired results, the second configuration was used. This automatically calls SystemCrafter synthesis as part of the build process, which produces two descriptions of the synthesized circuit: a SystemC description, and a VHDL description.

As part of the build process the gate-level SystemC description is compiled into a gate-level simulation model, which can be run to produce a gate-level simulation of the DES engine. This verifies that the synthesis process is correct. The VHDL description was then simulated using ModelSim.

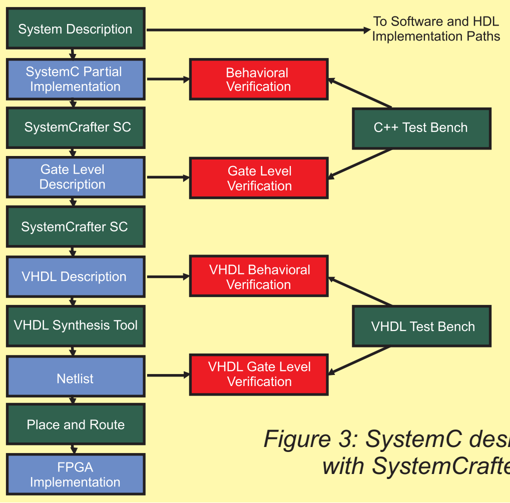


Figure 3: SystemC design flow with SystemCrafter.

Implementation

When KeySet is high for a cycle, the value of KeyIn is used to pre-compute the key values which are stored in a RAM table.

When DataInWE is high, DataIn is used as the next 64 bit input data block. DataInBusy will be high during execution of the encryption/decryption loop to hold off further data.

When DataOutWE is high, the encrypted or decrypted data appears on the DataOut port. DataOutBusy can be held high to hold off the new result.

The pseudo code for the design is shown in Figure 4.

Figure 5 shows the structure of the final design. The look up tables are implemented as CoreGen modules to illustrate the process of including complex IP blocks in a SystemCrafter design. SystemCrafter treats SystemC modules with missing method definitions as black boxes and this is how the SBlock class is defined in the DES source code.

During behavioural simulation, it is important that the correct behaviour of the CoreGen blocks is simulated. For this reason, a SystemC model of the look up tables is provided (SBlock***n***::LUT() method) which is removed for the compilation to hardware. Using these techniques, models of any IP blocks can be created to verify behaviour at an early stage.

The VHDL files produced by SystemCrafter is the core of the implementation. Support modules supplied with the board helped with the interface between the PC and ZestSC1 FPGA board.

A VHDL module simplifies the connection to the USB bus on the board. A small piece of VHDL was written to connect the 16 bit USB bus to the 64 bit DES input and output ports.

The complete DES project was then compiled using the standard Xilinx tools (XST for the VHDL and CoreGen for the ROMs, followed by place and route) to generate an FPGA configuration file. The device driver and C library contained in the ZestSC1 support package were then used to develop a simple GUI that configures the board with the FPGA configuration file. It then loads an image, sends it to ZestSC1 over the USB bus, and displays the encrypted result. The data can then be decrypted to retrieve the original image.

```
loop forever
    if KeySet = 1
        DataInBusy = 1
        K(0) = Permute(KeyIn)
        loop for i = 1 to 16
            K(i) = K(i-1) << shift amount
            K(i) = Permute(K(i-1))
        end loop
        DataInBusy = 0
    else if DataInWE = 1
        DataInBusy = 1
        D(0) = Permute(DataIn)
        loop for i = 1 to 16
            E = Permute(D(i-1))
            A = E xor K(i)
            S = LUT(A)
            P = Permute(S)
            D(i) = Concat(Range(D(i-1), 31,
            0), P xor Range(D(i-1), 63, 32))
        end loop
        wait for DataOutBusy = 0
        DataOut = D(16)
        DataOutWE = 1 for one cycle
        DataOutBusy = 0
    end if
end loop
```

Figure 4: Pseudo code for DES implementation.

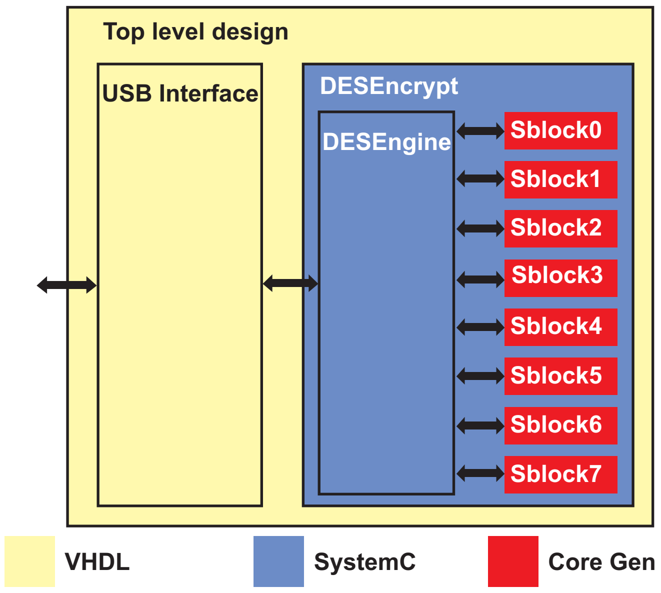


Figure 5: FPGA implementation of DES.

A sample output from the GUI is shown in Figure 6.

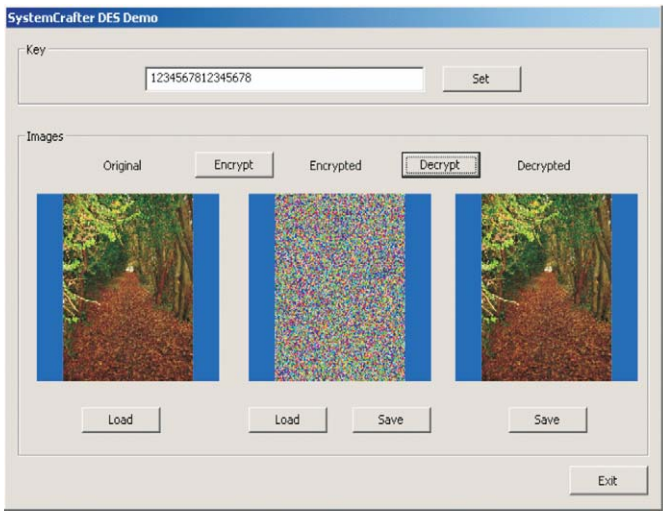


Figure 6: DES Image Encryption GUI.

Conclusion

SystemCrafter can be used to provide a simple route from system-level design down to Xilinx FPGAs.

It is suitable for programmers, scientists, systems engineers and hardware engineers. It enables developing hardware to be viewed as a higher level activity than writing an HDL, and allows the user to focus on the algorithm, rather than the details of the implementation. This can improve time to market, reduce design risk, and allow programmers and scientists to design complex systems without learning HDLs or electronics.

Both SystemCrafter SC and the DES implementation, including working source files and a more detailed description can be downloaded from the SystemCrafter website (www.systemcrafter.com).