

Implementation of the DES Algorithm Using SystemCrafter SC

APP 001

1. Introduction

This application note is intended as an introduction to the SystemCrafter SC synthesis tool. The DES encryption algorithm is used as an example of a complete design flow from high level specification through to final implementation in an FPGA. Simulation of the design at all levels is illustrated.

The chosen implementation mixes VHDL, SystemC and Xilinx CoreGen modules to illustrate mixed language design and tool interoperability. The chosen implementation platform is the Orange Tree Technologies ZestSC1 board (available from the SystemCrafter website).

2. The DES Algorithm

The Data Encryption Standard (DES) algorithm encodes data using a 64 bit key. 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.

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 re-used 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.

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 [1].

3. Design Flow

A typical design flow consists of an initial implementation which must be verified at many stages in the conversion to final hardware. The initial implementation phase involves choosing a suitable partition of the design between hardware and software and then sub-partitioning the hardware and software components based on the suitability of implementation in various languages and with various tools.

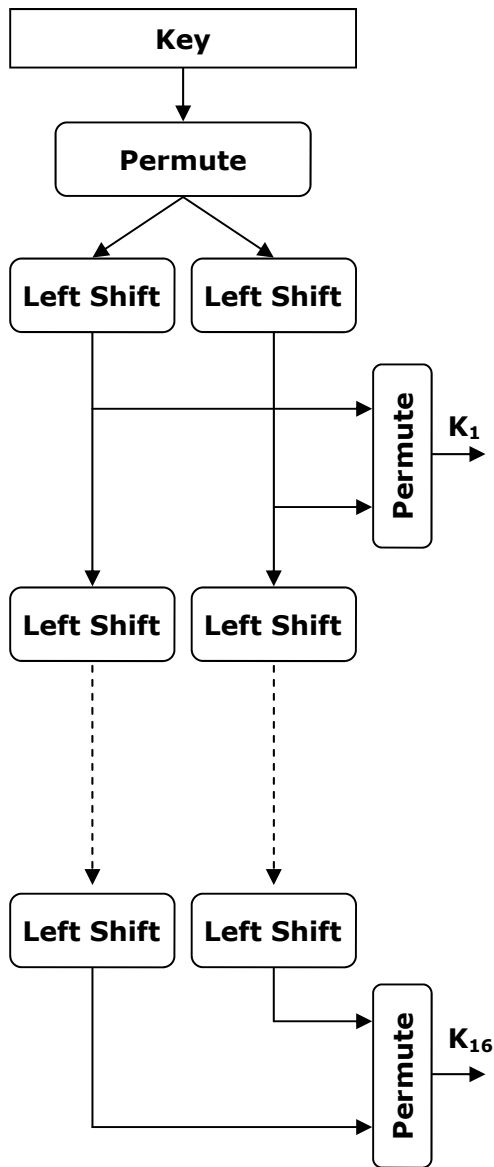


Figure 1. Circuit for pre-calculating the DES key values

For example, the core DES engine involves high bandwidth computation using operators unsuitable for software implementation so hardware implementation is a sensible choice for this block.

Inside the DES engine, the main iteration loop is algorithmic and therefore suitable for description in SystemC. However, the look up operation should take advantage of special constructs in the FPGA for area and speed efficiency so macros are a sensible implementation choice for these tables. Xilinx provides the CoreGen utility to generate area and speed efficient ROM constructs ideal for this area of the design.

Another design choice may be made based on

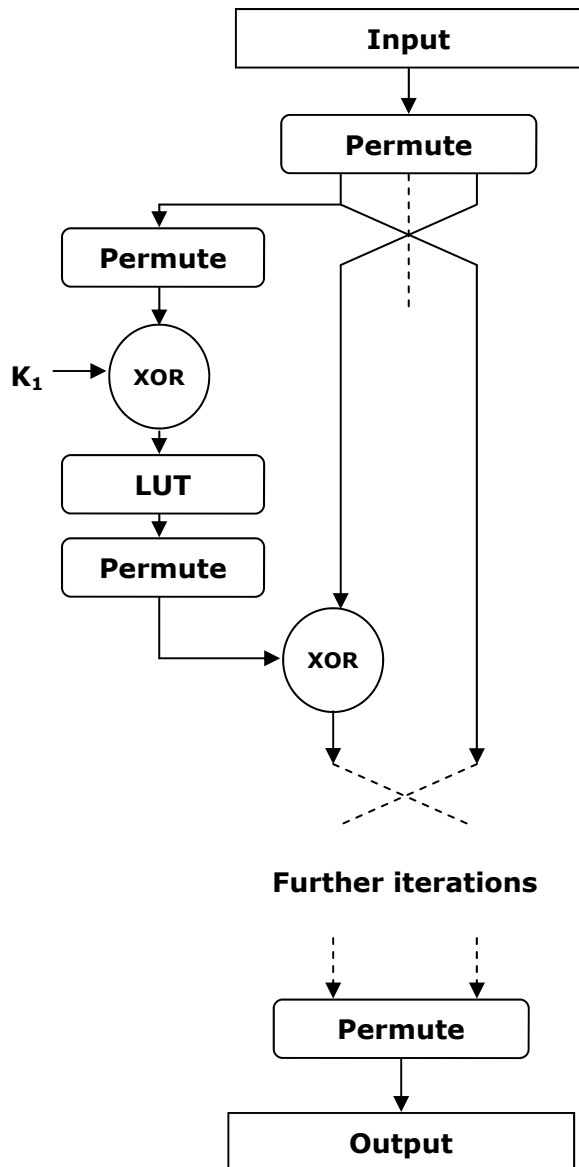


Figure 2. Circuit for encoding/decoding data

re-use of existing code blocks. The chosen implementation platform for the DES algorithm is provided with VHDL descriptions for the external interfaces which will be used to reduce implementation time.

Once partitioned, each component can be implemented simultaneously based on definitions of the interfaces between components.

Verification should be performed at different stages throughout the implementation chain. Figure 3 shows a typical implementation flow illustrating verification performed at behavioural level, gate level, post SYstemC synthesis and post place and route stages. In this way, problems caused by implementation specifics and compiler choices can be discovered

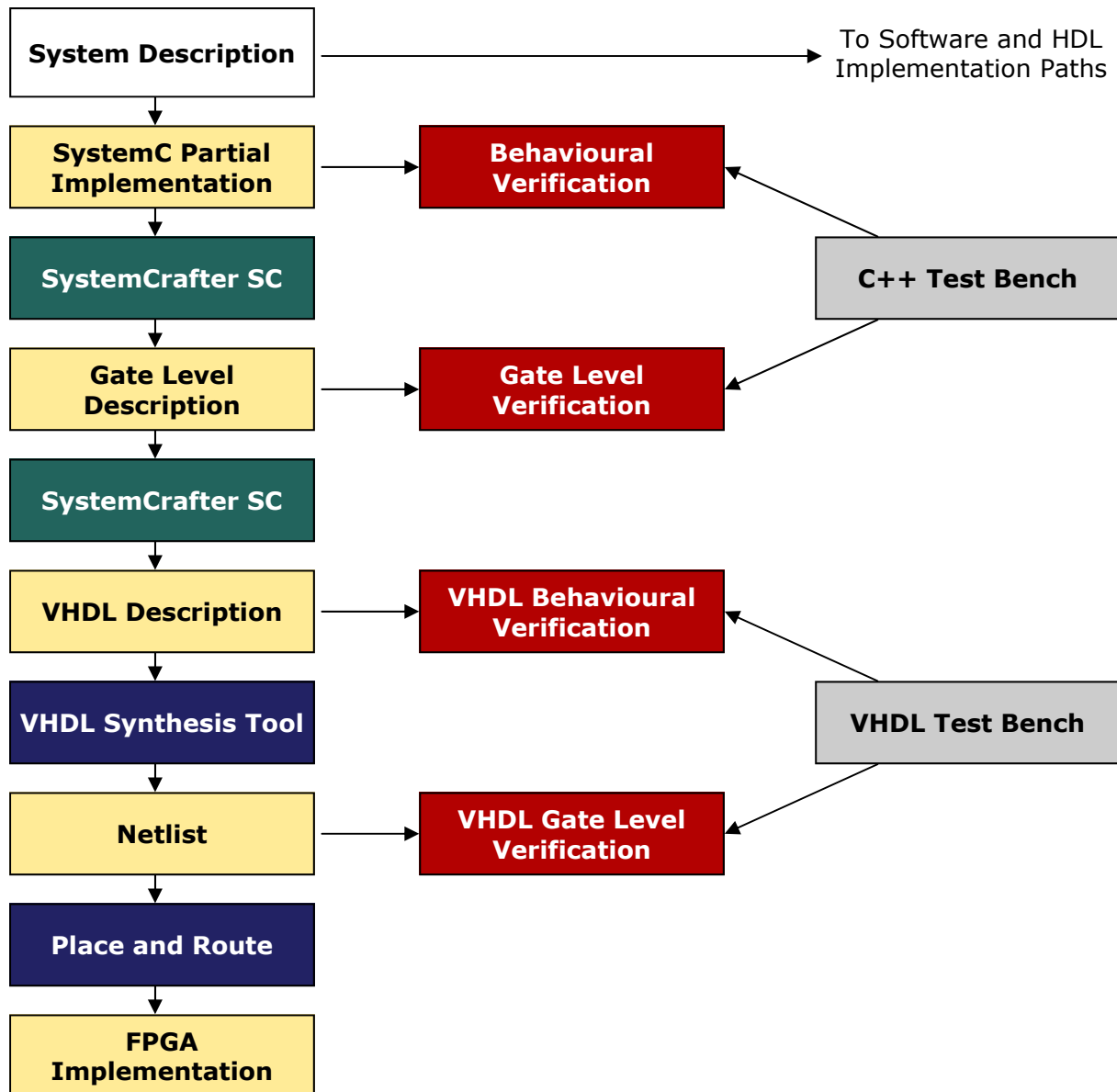


Figure 3. Typical design/verification flow

at an early stage.

4. SystemC Implementation

The hardware implementation of the DES encoder/decoder consists of a single block with the following ports:

```

sc_in<bool>          CLK;
sc_in<bool>          RST;

sc_in<sc_bv<64> >   KeyIn;
sc_in<bool>          KeySet;

sc_in<bool>          Encode;

sc_in<sc_bv<64> >   DataIn;
sc_in<bool>          DataInWE;
sc_out<bool>         DataInBusy;
  
```

```

sc_out<sc_bv<64> >   DataOut;
sc_out<bool>         DataOutWE;
sc_in<bool>          DataOutBusy;
  
```

The Encode port must be set to 1 for DES encryption or 0 for DES decryption. This value simply reverses the order of the pre-computed key values.

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.

```

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

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 de-

sign. 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

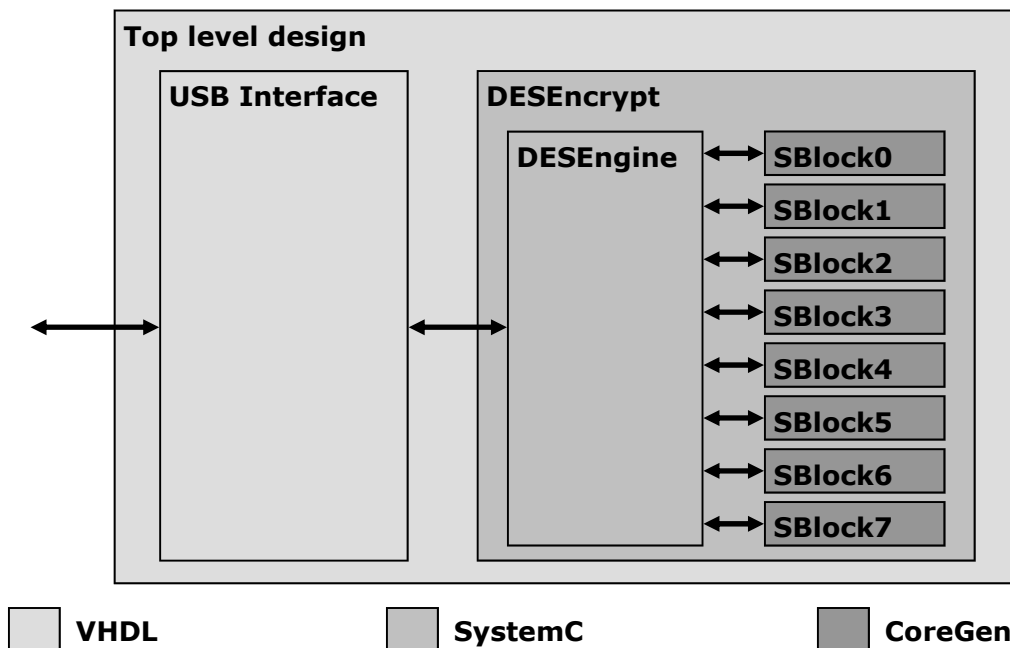


Figure 5. Hierarchy of FPGA implementation of DES

File	Description
SystemC\SystemC.sln	Main Visual Studio solution workspace
SystemC\DESEncrypt.cpp	DES encryption engine
SystemC\DESEncrypt.h	Top level 'wires only' definition
SystemC\SBlocks.cpp	SystemC model of CoreGen ROM blocks
SystemC\TestBench.cpp	SystemC test bench
VHDL\DES.npl	Xilinx Project Navigator file
VHDL\TestBench.vhd	VHDL test bench
VHDL\TopLevel.vhd	Top level 'glue' VHDL file
Host\Host.sln	Visual Studio solution workspace for GUI

Table 1. Main files in the source code package

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.

A top level block consisting solely of wires is used to instantiate the 8 look up tables and the main DES engine and connect the components together.

Table 1 lists the main files in the associated source code package.

5. Simulating the Design

The first stage in verification is executing the behavioural description of the circuit. To achieve this, a test harness is provided which feeds standard DES test patterns through the engine and displays the results on the console.

The source code package contains a Microsoft Visual Studio .NET solution workspace with configurations for behavioural simulation (Debug and Release) and gate level simulation (SystemCrafter). To execute a behavioural simulation, compile the workspace for Debug or Release and run the resulting SystemC.exe file. The testbench included in the workspace will feed standard DES test vectors through the design and display actual and expected results for comparison.

Once behavioural simulation has been completed, SystemCrafter can be used to generate a gate level simulation model. This model accurately reflects the hardware that will be generated and can be used to confirm the correctness of the synthesis.

Gate level simulation models can use the same test benches as the behavioural description, so reducing the amount of work required during testing. In fact, the gate level model is a direct, drop-in replacement for the behavioural description.

To execute a gate level simulation, compile the SystemCrafter configuration in the Visual Studio workspace. This generates both gate level SystemC and VHDL from the behavioural SystemC description. Running the resulting SystemC.exe file should produce the same results as the behavioural simulation.

At the same time as the gate level SystemC description is generated, VHDL is generated ready for compilation by the Xilinx tools.

It is also possible to link the software and HDL arms of the design flow in to the SystemC simulation. This requires models of the HDL blocks (which can be written in SystemC) and some interfacing code between the software and SystemC blocks. Whilst outside of the scope of this document it should be noted that SystemC allows simulation of the complete system before final implementation.

6. FPGA Implementation

The target hardware platform is the Orange Tree Technologies ZestSC1 FPGA card. The ZestSC1 consists of a Xilinx Spartan-3 FPGA, some SRAM (unused in this design) and a USB interface.

Compiling the SystemCrafter configuration of the Visual Studio solution workspace from the associated source code package generates VHDL from the behavioural SystemC description. This VHDL can then be simulated using

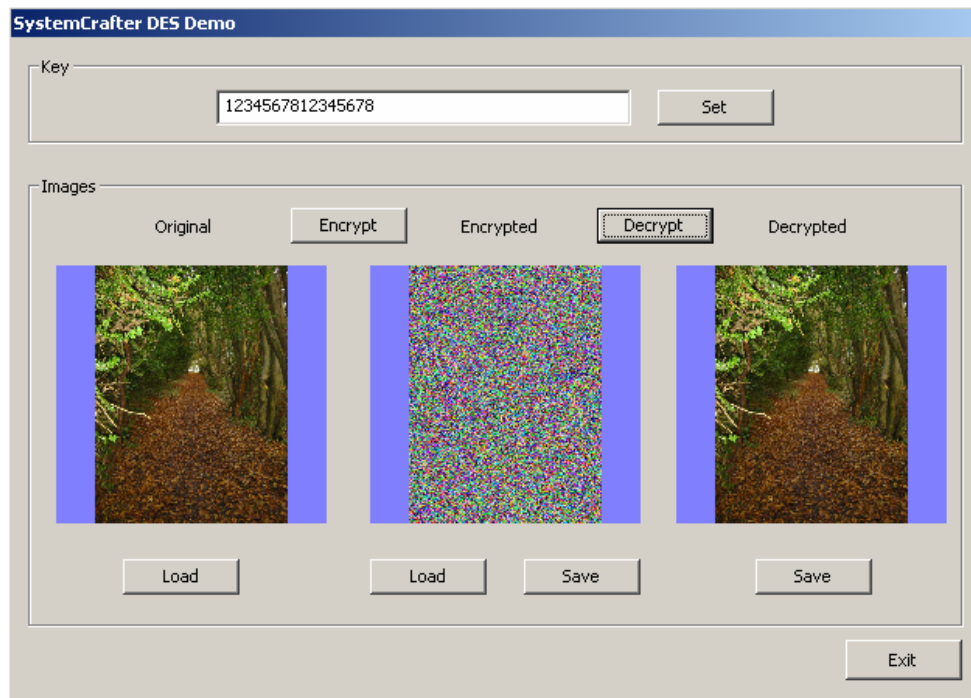


Figure 6. DES image encryption GUI

(for example) ModelSim to confirm correctness. A VHDL test bench is used to supply the same DES test patterns used during behavioural simulation.

A Xilinx project file (.npl file) is included in the source code package with all the correct settings for the ZestSC1 board. It also includes the ModelSim test bench to feed the standard DES test vectors through the DES engine at behavioural and gate level stages. Executing this test bench in ModelSim should produce the results seen in the SystemC simulations on the DataOut port when DataOutWE is high.

Once simulated, the DES engine must be connected to the ports on the target hardware platform. A VHDL module is provided in the ZestSC1 support package [2] which simplifies the connection to the USB bus on the board. The USB interface provides a 16 bit streaming data bus in either direction and a microprocessor bus interface for control registers inside the FPGA. A small piece of VHDL (TopLevel.vhd) is used to connect the 16 bit USB bus to the 64 bit DES input and output ports.

Figure 5 shows the structure and implementation language of the various modules in the complete system.

Compiling the complete DES project with the Xilinx tools (XST for the VHDL and CoreGen for the ROMs followed by place and route) generates a FPGA configuration (.bit) file.

7. Host Application

The ZestSC1 support package also contains a device driver and associated C library for developing applications. These are used by a simple GUI which loads an image, sends it to the ZestSC1 card over the USB bus and displays the encrypted result. The data can then be decrypted to retrieve the original image. Figure 6 shows the GUI.

Running the GUI detects the presence of ZestSC1 boards and configures the first one it finds with the configuration file generated by the Xilinx tools.

To encrypt an image, enter a 64 bit key as 16 hexadecimal digits and click 'Set'. This sends the key to the FPGA and runs the pre-compute stage of the DES algorithm.

Load an image by clicking 'Load' under the 'Original' box and selecting a JPEG or BMP file. Click 'Encrypt' to encrypt the image. The encrypted image should look like garbage to the human eye. Click 'Decrypt' to reverse the operation. Decryption should only be successful if the same key is used. If the key is altered (and 'Set' is clicked) then the decryption should also produce garbage. (Note, however, that every eighth bit of the key is a parity bit which is not used for encryption/decryption and so can be changed without effect).

8. Conclusion

This application note has shown how a real-world application can be developed using the SystemCrafter tools. As in most complex FPGA designs, a mixture of techniques have been used where they exhibit strengths to produce a complete system design.

9. References

- [1] Data Encryption Standard (DES)
<http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>
- [2] ZestSC1 User Guide
<http://www.orangetreetech.com>
- [3] SystemC User Guide
<http://www.systemc.org>
- [4] SystemC Functional Specification
<http://www.systemc.org>
- [5] SystemCrafter User Guide
<http://www.systemcrafter.com>
- [6] Application Note 001 Source Code
<http://www.systemcrafter.com>