

# Using SystemC and SystemCrafter to Implement Data Encryption

SystemCrafter provides a simple route for high-level FPGA design.

by Jonathan Saul, Ph.D.  
CEO  
SystemCrafter  
[jon.saul@systemcrafter.com](mailto:jon.saul@systemcrafter.com)

C and C++ have been a popular starting point for developing hardware and systems for many years; the languages are widely known, quick to write, and give an executable specification, which allows for very fast simulation. C or C++ versions of standard algorithms are widely available, so you can easily reuse legacy and publicly available code. For system-level design, C and C++ allow you to describe hardware and software descriptions in a single framework.

Two drawbacks exist, however. First, C and C++ do not 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 aren't popular because they tie users to a single software supplier. Second, you must manually translate C and C++ to a hardware description language such as VHDL or Verilog for hardware implementation. This time-consuming step requires hardware experts and often introduces errors that are difficult to find.

The first problem has been solved by the development of SystemC, which is now a widely accepted industry standard that adds hardware concepts to C++.

The second problem has been solved by the development of tools like SystemCrafter SC, which allows SystemC descriptions to be automatically translated to VHDL.

In this article, I'll explain how to use SystemCrafter SC and SystemC by describing an implementation of the popular DES encryption algorithm.

## SystemC

SystemC provides an industry-standard means of modeling and verifying hardware and systems using standard software compilers. You can download all of the material required to simulate SystemC using a standard C++ compiler, such as Microsoft Visual C++ or GNU GCC, from the SystemC website free of charge ([www.systemc.org](http://www.systemc.org)).

SystemC comprises 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 for simulation and debugging within your existing C++ development environment.

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

And because 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; as a result, it is faster to write and more maintainable and readable. It can be compiled quickly into an executable specification.

## SystemCrafter SC

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

SystemCrafter SC automates this process by quickly synthesizing SystemC to RTL VHDL. It will also generate a SystemC description of the synthesized circuit, allowing you to verify the synthesized code with your existing test harness.

You can use SystemCrafter SC for:

- Synthesizing SystemC to hardware
- System-level design and co-design
- Custom FPGA co-processing and hardware acceleration

SystemCrafter SC gives you control of the critical steps of scheduling (clock cycle allocation) and allocation (hardware reuse). Thus, the results are always predictable, controllable, and will match your 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 also configure the compiler so that SystemCrafter SC will automatically run when you want to generate hardware.

### The DES Algorithm

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

A DES implementation comprises two stages. During the first stage, 16 intermediate values are pre-computed based on the initial key. These 16 values are fixed for a particular key value and can be reused for many blocks of data. Calculation of the key values involves repeated shifts and reordering of bits.

The second computation stage involves 16 iterations of a circuit using one of the pre-computed key values. 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. Each iteration involves permutations, exclusive-OR operations, and the look-up of eight 4-bit values in 8 look-up tables.

Decryption works exactly the same way as the second computation stage, but with the 16 key values from the first stage used in reverse order.

For a full description of the DES algorithm, go to <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>.

### Design Flow

The design flow using SystemC and SystemCrafter is shown in Figure 1. An important benefit of this design flow is that you can carry out the development of the initial SystemC description, partitioning, and system- and gate-level simulation all in the same framework. The designer

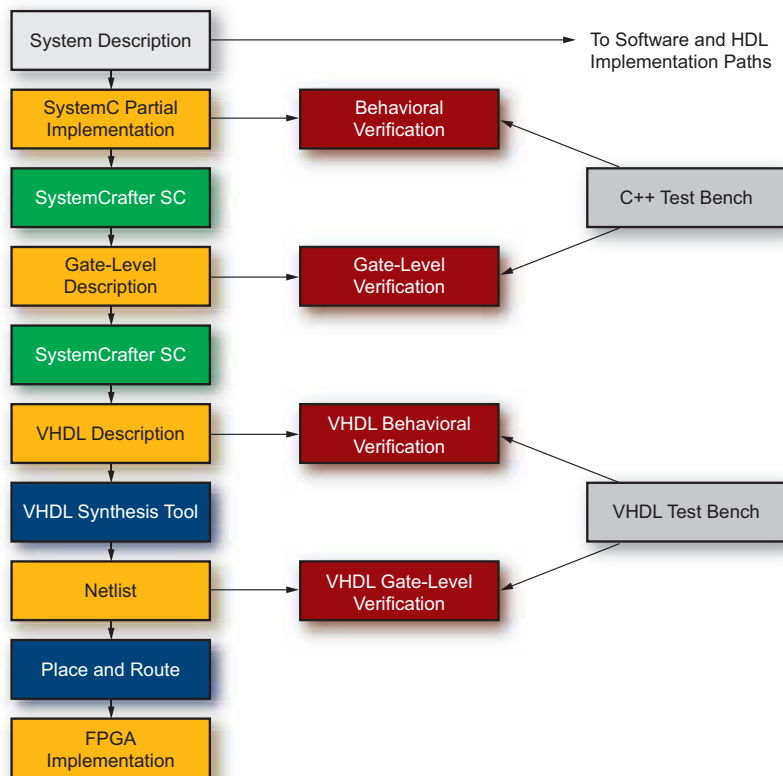


Figure 1 – Design flow

implementing the DES algorithm used his normal design environment, Microsoft's Visual C++. The target platform was the ZestSC1, an FPGA board containing a Xilinx® Spartan™-3 FPGA and some SRAM communicating with a host PC through a USB interface.

### Hardware/Software Partitioning

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

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

As I stated previously, the designer wrote a hardware description of the DES core in SystemC using Microsoft's Visual C++ design environment. Pseudo code is shown in Figure 2. Calls to SystemCrafter

are made as “custom build steps,” which allows simulation and synthesis from within the Visual C++ design environment. You can also use the SystemCrafter GUI to manage projects.

Implementing look-up tables as CoreGen modules shows how complex third-party IP blocks are usable in a SystemCrafter design. SystemCrafter treats SystemC modules with missing method definitions as black boxes, allowing you to use IP blocks written in other languages. For simulation, the designer wrote a model of the look-up tables in SystemC for use during simulation.

### Simulation and Synthesis

The SystemCrafter flow allows you to simulate the design at a number of levels.

In the DES example, the designer wrote a test harness, which fed standard DES test patterns through the hardware engine and displayed the results on the console. He specified two configurations in Visual C++: system level and gate level.

The system-level configuration was first. It compiles the original SystemC design to

```

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 2 – Pseudo code for the DES implementation

produce a simulation model at the behavioral level. This model is an executable program that can produce a fast behavioral-level simulation of the DES engine.

Once the behavioral model produced the desired results, it was time for the gate-level configuration. It 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 produce a fast gate-level simulation of the DES engine. This verifies that the synthesis process is correct.

The designer used Mentor Graphics's ModelSim to simulate the VHDL description.

### Implementation

The VHDL 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 simplified the connection to the USB bus. The designer wrote a small piece of VHDL to connect the 16-bit USB bus to the 64-bit DES input and output ports.

The complete DES project was then compiled using standard Xilinx tools (XST for the VHDL and CORE Generator™ software 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 integral in developing 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 for encryption and then decryption, and displays the resulting images.

Figure 3 shows a sample output from the GUI.



Figure 3 – DES image encryption GUI

### Discussion

The DES application was written by a VHDL expert who was new to SystemC. The complete design, including both hardware and software, went from concept to final version in only three days, including a number of explorations of different design implementations.

The SystemCrafter flow was flexible enough so that the designer could use a mixture of SystemC for the algorithmic part of the design, CORE Generator blocks for the look-up tables, and VHDL for low-level interfacing.

The flow allowed him to use his existing Visual C++ design environment for code development, simulation, and synthesis. The SystemC description was more concise than an equivalent VHDL design would have been, and this level of description allowed the development to focus on algorithmic issues.

### Conclusion

SystemCrafter SC offers a simple route from system-level design down to Xilinx FPGAs.

It is suitable for programmers, scientists, systems engineers, and hardware engineers. It enables you to view developing hardware as a higher-level activity than writing HDL and allows you to focus on the algorithm rather than on the details of the implementation. This can improve time to market, reduce design risk, and allow you 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, are available as downloads from the SystemCrafter website, [www.systemcrafter.com](http://www.systemcrafter.com).