"Implementation Trade-offs of Triple-DES in the SRC Reconfigurable Computing Environment"

Osman Devrim Fidanci ¹, Hatim Diab ¹, Tarek El-Ghazawi ¹, Kris Gaj ² and Nikitas Alexandridis ¹

¹: George Washington University
²: George Mason University
Outline

• Why Reconfigurable Computing?

• SRC-6E General Purpose Reconfigurable Computer
  – Hardware Architecture
  – Programming Model

• Triple DES Encryption Algorithm
  – Single DES Algorithm
  – Triple DES with 2 Keys

• Implementation Trades-offs of Triple-DES in SRC

• Conclusions
Why Reconfigurable Computing?

Performance

- Direct instantiation of hardware results in better efficiency
- Reduces I/O bandwidth requirements by elimination of Load/Store paradigm

Scalability

- Basic technology improving much faster than Moore’s law
- Takes advantage of parallelism found in many programs
SRC-6E Hardware Architecture
SRC MAP Compiler Architecture
Advantages of SRC Hardware

• Introduction of the SNAP card

  – Data feed to the FPGA though Memory bus with 800 MB/s peak (theoretical)
  – Eliminating the PCI bottleneck in traditional FPGA cards
SRC-6E System SW

• **System**
  – Linux
    • Red Hat 7.2
    • Driver and Library additions to support SNAP and MAP

• **Compilers**
  – Microprocessor and MAP

• **Tools**
  – WINE
  – FPGA
    • Synplicity Synplify Pro
    • Xilinx Integrated Software Environment
Macro Development for SRC

- User macros can be defined in Verilog or VHDL
- Three types of macros defined in SRC platform:
  - Functional
  - Stateful
  - External
- System clock frequency is 100MHz. So, User macros should be optimized for this speed. Otherwise FIFO must be employed
General Depiction of DES Encryption Algorithm

- 64-bit inputs (plaintext and keys)
- 64-bit output (ciphertext)
- 16-round operation plus initial permutations
- Pipelined, 17-clock cycles latency
- Output generated at every clock cycle
Single Round of DES Algorithm

四大操作：
1. 扩展/置换操作（E-table）
2. 逻辑XOR操作
3. 替换/选择操作（S-box）
4. 置换（P）
“Triple-DES with two keys” scheme is used for backward compatibility with DES (by setting two keys identical, provides single DES encryption/decryption).

- Triple DES requires 51-clock cycle to get ciphertext
Different Implementations of Triple DES in SRC

• **Case 1**  
  *Effect*: DES macro is called three times from Fortran main file

• **Case 2**  
  *Effect*: DES macro is called three times from Fortran subroutine file

• **Case 3**  
  *Effect*: Triple-DES macro is called once from Fortran main and subroutine files
Calling DES/Tri-DES macro from Fortran HLL

Main.f90

Subroutine1.mf
  DES.v

Subroutine2.mf
  DES.v

Subroutine3.mf
  Tri-DES.v

3 times

3 times
Implied Architectures

Fidanci

Case 1

17-clock cycles

3 times

DES

RAM

FPGA

Case 2

51-clock cycles

DES

DES

RAM

FPGA

Case 3

51-clock cycles

DES

Triple DES

RAM

FPGA

D3
Processing Time Measurement

• The timer macro is simply a counter running at 100 MHz

• There are two calls:
  - start_timer: zeros out the counter
  - read_timer: gets the counter’s current value
# Implementation Trades-Offs of Triple DES in SRC

<table>
<thead>
<tr>
<th></th>
<th>Experiment(1)</th>
<th>Experiment(2)</th>
<th>Experiment(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Maximum Clock Speed (MHz)</strong></td>
<td>102.3</td>
<td>100.8</td>
<td>101.8</td>
</tr>
<tr>
<td><strong>CLB Slices (Tot. equiv. Gate count)</strong></td>
<td>6,177 (163,835)</td>
<td>13,269 (382,927)</td>
<td>11,786 (359,635)</td>
</tr>
<tr>
<td><strong>Total Processing Time (91 blocks of data) (ns)</strong></td>
<td>4440</td>
<td>1820</td>
<td>1820</td>
</tr>
</tbody>
</table>
Timing Estimations

\[ \text{Total execution time} = \text{Transfer time} + \text{Total processing time} \]

\[ \mu P \text{ memory } \leftrightarrow \text{on-board memory} \]

Measured experimentally
Timing Estimations – cont.

Total processing time =

\[
\left( \text{Load time} + \#\text{pipeline\_stages} + (N-1) + \text{Store time} \right) \times t
\]

where

\[
N \rightarrow \text{number of data blocks being processed}
\]

\[
t \rightarrow \text{number of subroutine calls}
\]

Load time + Store time = Load/Store time = T_{LS} (unknown)
## Timing Estimations – cont.

<table>
<thead>
<tr>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>#_pipeline stages: 17</td>
<td>#_pipeline stages: 51</td>
<td>#_pipeline stages: 51</td>
</tr>
<tr>
<td>#_data blocks = N: 91</td>
<td>#_data blocks = N: 91</td>
<td>#_data blocks = N: 91</td>
</tr>
<tr>
<td>#_subroutine_calls = t: 3</td>
<td>#_subroutine_calls = t: 1</td>
<td>#_subroutine_calls = t: 1</td>
</tr>
<tr>
<td>Load/Store time: $T_{LS}$</td>
<td>Load/Store time: $T_{LS}$</td>
<td>Load/Store time: $T_{LS}$</td>
</tr>
<tr>
<td>Clock period: 10ns</td>
<td>Clock period: 10ns</td>
<td>Clock period: 10ns</td>
</tr>
<tr>
<td>Estimated # of clock cycles for processing 91 blocks: $[(17 + 90) + T_{LS}] \times 3 = 444$</td>
<td>Estimated # of clock cycles for processing 91 blocks: $[(51 + 90) + T_{LS}] = 182$</td>
<td>Estimated # of clock cycles for processing 91 blocks: $[(51 + 90) + T_{LS}] = 182$</td>
</tr>
<tr>
<td>Load/Store time: $T_{LS} = 41$</td>
<td>Load/Store time: $T_{LS} = 41$</td>
<td>Load/Store time: $T_{LS} = 41$</td>
</tr>
</tbody>
</table>

Fidanci
### I/O Overhead (T<sub>LS</sub>/Total time)

<table>
<thead>
<tr>
<th></th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>91 data blocks</td>
<td>27.7 %</td>
<td>22.5 %</td>
<td>22.5 %</td>
</tr>
<tr>
<td>501 data blocks</td>
<td>7.3 %</td>
<td>6.9 %</td>
<td>6.9 %</td>
</tr>
<tr>
<td>1001 data blocks</td>
<td>3.8 %</td>
<td>3.7 %</td>
<td>3.7 %</td>
</tr>
<tr>
<td>10001 data blocks</td>
<td>0.4 %</td>
<td>0.4 %</td>
<td>0.4 %</td>
</tr>
</tbody>
</table>
Ratio of processing times

\[
\frac{\text{Case 1 Total processing time}}{\text{Case 2&3 Total processing time}} = \frac{(17 + (N-1) + 41) \times 3}{(51 + (N-1) + 41)}
\]

<table>
<thead>
<tr>
<th>N</th>
<th>91</th>
<th>501</th>
<th>1,001</th>
<th>10,001</th>
</tr>
</thead>
<tbody>
<tr>
<td>ratio</td>
<td>2.44</td>
<td>2.83</td>
<td>2.91</td>
<td>2.99</td>
</tr>
</tbody>
</table>
Discussion

Case 2, i.e., calling DES macro three times from the Fortran subroutine, and
Case 3, i.e., calling 3DES macro once from the Fortran subroutine

almost equivalent

- the same execution time
- area 13% larger for Case 2 because of the default interface between single DES modules
Case 1, i.e., calling DES macro three times from the Fortran main file, and

Cases 2 and 3

very different

– approximately two times smaller area for Case 1
– over two times longer execution time caused by the larger number of iterations, larger I/O overhead (communication between FPGA and on-board memory) and smaller utilization of the pipeline
Conclusions

• SRC Programming Model enables flexible choice of the hardware architecture used to implement required function
• Implied architecture depends on
  – function (granularity) of the hardware description language macro
  – placement of macro calls in a high level language program
Conclusions – cont.

- Common features of all implemented architectures
  - deep pipelining
  - operational clock frequency 100 MHz
- Overhead associated with the run-time communication between FPGA (User Chip) and on-board memory can be made negligible for processing of large amounts of data