An Implementation Comparison of an IDEA Encryption Cryptosystem on Two General-Purpose Reconfigurable Computers

Allen Michalski[1], Kris Gaj[1], Tarek El-Ghazawi[2]

[1] ECE Department, George Mason University
4400 University Drive, Fairfax, VA 22030, U.S.A.
[2] ECE Department, The George Washington University
801 22nd Street NW, Washington DC 20052, U.S.A.
{emichals, kgaj}@gmu.edu, tarek@seas.gwu.edu

Abstract. The combination of traditional microprocessors and Field Programmable Gate Arrays (FPGAs) is developing as a future platform for intensive computational computing, combining the best aspects of traditional microprocessor front-end development with the reconfigurability of FPGAs for computation-intensive problems. Several prototype PC-FPGA machines have demonstrated significant speedups compared to standalone PC workstations for computationally intensive problems. Cryptographic applications are a clear candidate for this type of platform, due to their computational intensity and long operand lengths. In this paper, we demonstrate an efficient implementation of IDEA encryption, using two of the leading reconfigurable computers available, SRC Computers' SRC-6E and Star Bridge Systems' HC-36. We compare the hardware architecture and programming model of these reconfigurable computers, and the implementation of a common IDEA encryption architecture in both platforms. Detailed analyses of FPGA resource utilization for both systems, data transfer and reconfiguration overheads for the SRC system, and a comparison between SRC and a public domain software implementation are given in the paper.

1. Introduction

The need for faster computational processing methods has grown along with the desire to process large amounts of data in shorter periods of time. Approaches to this problem have typically involved microprocessor-based solution, utilizing concurrent processing with multiple processors within a single machine or in a cluster of workstations over a network. Since their introduction, FPGAs have emerged as a low cost complement to traditional microprocessor software and hardware solutions for computationally intensive applications, due to their hardware reconfigurability [1].

Several workstation-based FPGA systems are available today for commercial applications. These "reconfigurable computers" make use of workstation microprocessor(s) for front-end processing, and provide one or more FPGAs for computations less suited for a microprocessor. The choice of how a design can be divided between a microprocessor and an FPGA depends on the development environment available for the system. In this paper, we explore the efficient implementation of pipelined IDEA encryption within two such available reconfigurable computers: the SRC-6E from SRC Computers, and the HC-36 from Star Bridge Systems. This paper discusses the hardware architectures and development tools of both systems in detail, and provides FPGA timing and utilization results for both systems. A summary including the state of development of both systems is also presented.

2. FPGAs and Reconfigurable Computing

An FPGA is a regular structure of basic modules called Configurable Logic Blocks (CLBs), which can be interconnected to provide hardware implementations of algorithms required by a designer. FPGA interconnects between modules are under the designer's complete control [2]. The FPGAs used in both the SRC and Star Bridge systems are the Xilinx Virtex II 6000 series, which have a capacity of six million system gates, and provides dedicated logic for fast carry propagation for addition operations, multipliers that handle operand sizes up to 18 bits, and block RAMs for local memory access. FPGAs are reconfigurable, meaning that the FPGA device can be configured to carry out a specific function, and can be reconfigured to carry out a different function at a later time.

The term "reconfigurable computing" is used to describe a combination of microprocessor systems with FPGAs to provide a reconfigurable hardware environment. Both the SRC and Star Bridge systems are examples of reconfigurable computing platforms. Reconfigurable computers offer benefits over microprocessor-based hardware solutions because FPGAs can more easily exploit computational precision and operand sizes required by the design, and can implement operation pipelining and parallelism specific to the needs of the application being developed. Microprocessor instruction sets have fixed operand lengths that may not match operand sizes specific to the design, and the implementation of algorithms typically involves multiple-instruction executions for one algorithmic operation. In addition, much of the microprocessor's capability is unutilized, since a general-purpose microprocessor is designed to implement a wide variety of operations specific to a workstation computing environment, whereas most design requirements for cryptographic systems only use a subset of the full capabilities of a microprocessor. All of the resources of an FPGA can be dedicated to the needs of a design, which provides a more efficient implementation versus a single or multi-processor-based design solution.

3. The SRC-6E

3.1 SRC-6E Hardware Overview

The SRC-6E system architecture frontend consists of two dual Intel processor motherboards. Each motherboard contains two Intel P3 Xeon processors and 1.5 GB of memory. Each system hosts a multi-processor version of the Linux operating system, and provides two distinct Linux-based microprocessor-FPGA reconfigurable computers.

An SRC MAP® processor is attached to each Intel motherboard, as shown in Fig. 1. Each MAP processor consists of two Xilinx Virtex II 6000 FPGA chips available for user logic, and a control processor, which is also a Xilinx Virtex II 6000 FPGA, all running at a clock rate of 100 MHz. The control processor implements fixed control logic, and is responsible for direct memory access (DMA) transfers between Intel system memory and the onboard memory of the MAP processor. The user logic and control FPGAs have access to six banks of dual port 512k x 64 bit static RAM providing a total of 24MB of memory external to the FPGAs. The MAP control processor communicates with the Intel processors through a SNAP interconnect. The SNAP interconnect is a high speed, low latency interface which functions as a Double Data Rate (DDR) memory interface, and plugs into a DDR SDRAM slot on the motherboard. SNAP provides higher data throughput between the Intel processor and the MAP processor versus component interfacing using the PCI-X bus.

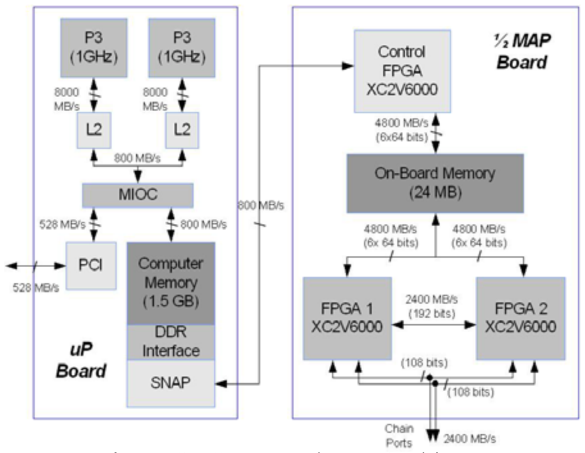


Fig. 1. SRC-6E Hardware Architecture

3.2 SRC-6E Programming Environment

SRC has created a development environment that uses traditional programming paradigms in addition to hardware description language (HDL) development for implementing a design with the MAP FPGAs, as shown in Fig. 2. The SRC environment provides the ability to implement FPGA user logic using either C or FORTRAN sourcecode alone, or in combination with HDL sourcecode. GNU compilers are provided for C or FORTRAN sources that target the Intel processors, and SRC provides its own C or FORTRAN compilers that target the MAP processors. The MAP processor compilers produce Verilog code which is then synthesized using Synplify Pro, and Xilinx tools perform map, place and route. The GNU compilers allow the use of ANSI-compliant C or FORTRAN code and libraries, which allows for integration with other existing UNIX applications.

The design environment is hosted in each Linux platform within the SRC-6E. The compilation process consists of compilation of user logic HDL files, if present, that will execute on the MAP processor, compilation of C or FORTRAN code that will also execute on the MAP processor, and compilation of the C or FORTRAN code that will execute on the Intel processors. The compilation process places wrapper code around logic that resides within the FPGAs to facilitate data transfer and control synchronization between the microprocessor and MAP processor. The binary files produced by each compile process are combined into one single executable for the host Intel platform, which is responsible for loading the Intel and MAP processors with the code compiled for each.

3.3 Designing within the SRC-6E

Designs in SRC can have required operations performed on either the Intel or MAP processors. Within the MAP processor, FPGA designs can be implemented using a C or FORTRAN source alone, a C or FORTRAN source in combination with HDL, or completely in HDL, using a C or FORTRAN source to provide data transfer services. A simple API is available in the high-level language (HLL) to provide control functionality and data transfer functions between the Intel and MAP processors, and the passing of data to and from included user HDL designs. MAP C sourcecode data types are limited to 32 and 64 bit types.

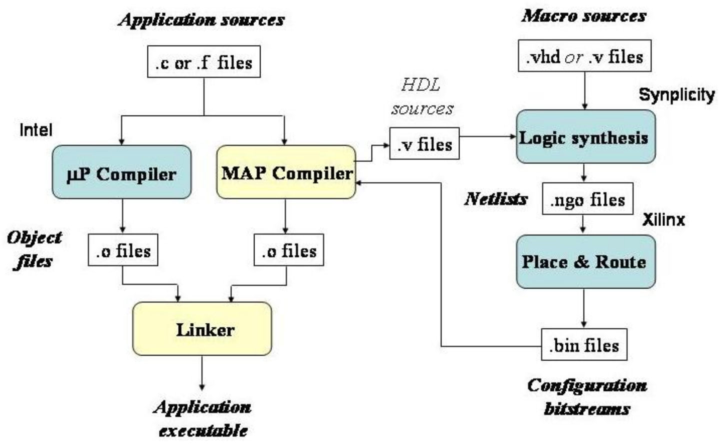


Fig. 2. Overview of the SRC Compilation Process

HDL source that is targeted for the MAP processor is called a macro. SRC supports either VHDL or Verilog sources within macros. A macro's primary characteristics are defined as functional, stateful, or external. Additional characteristics define latency of the design and whether the design is pipelined. An "info" file is used to describe these characteristics. A functional macro is one that carries no state information, and therefore doesn't require the reset of a state machine. A stateful macro is one that caries state information and may need to be reset during a data cycle. An external macro is one that needs direct hardware access to memory, versus using HLL calls to read memory and supply data to the design. Latency defines how many clock cycles are needed before a result is available, and the pipeline attribute defines whether the macro can take data on each clock cycle. The combination of all these attributes defines the HDL macro supplied by the user.

If a C or FORTRAN source is used alone or in combination with HDL source to implement the design within the MAP processor, the compiler attempts to extract the maximum parallelism from the code and generate pipelined hardware logic for instantiation in the MAP FPGAs [3].

3.4 Debugging within the SRC-6E

Debugging within SRC can be attained by traditional functional testing of HDL outside of SRC and traditional HLL debugging techniques, or using SRCs MAP debugger. The MAP debugger provides hardware debugging of code compiled for the MAP processor, and emulation debugging of emulation or simulation code. Emulation is based on specially formatted "data flow graph" C routines, provided by the compiler from existing HLL source if that is used to implement logic, or by the user in an additional file if HDL is also used, that emulates the functionality of the design being implemented in the MAP processor. Simulation mode actually simulates the HDL produced by the compiler, which requires a license for a Verilog simulator not included in the development environment.

4. The Star Bridge HC-36

4.1 HC-36 Hardware Overview

The Star Bridge HC-36a consists of a Tyan S2720 dual Intel P4 Xeon processor motherboard with 4 GB of memory. Attached to the motherboard through the PCI-X bus is a PCI card containing two Virtex II 4000 FPGAs for dedicated PCI and hardware control, one Virtex II 6000 FPGA allocated for user-designed FPGA control functions, and 4 Virtex II 6000 FPGAs available for user logic, as in Fig. 3. FPGA clocks are set to the PCI-X clock speed, with the FPGA digital clock manager used to provide clock speeds other than the PCI-X clock speed. Star Bridge uses the name "HyperComputer®" to refer to this combination of FPGAs, an Intel-based frontend, and Star Bridge's VIVA development environment [4]. The environment is hosted in Microsoft's Windows 2000 Server operating system.

FPGAs available for user logic are referred to as PE1 to PE4. These four PEs (Processing Elements) have dedicated 50-bit connections to each other. Each PE, along with the XPoint FPGA mentioned below, has four banks of memory, each bank having 512 MB of RAM with a 64-bit PE interface. This gives a total of 10 GB of memory with 20 64-bit independent memory channels within the user FPGAs.

Control FPGAs available to the user consist of an FPGA known as XPoint and a router FPGA. XPoint connects to the four user FPGAs using dedicated 32-bit data connections, and is allocated to provide user-defined control interfacing to the user FPGAs. The router FPGA has dedicated 94-bit connectivity to the user FPGAs, and provides additional user FPGA connectivity.

The base grouping of four PEs is called a Quad Element. Only one Quad Element is available in the HC-36a, however each XPoint control FPGA can be used to control two Quad Elements. This structure of one XPoint control FPGA along with two Quad Elements can be replicated to create larger FPGA arrays, and is used to create larger Star Bridge platforms [4].

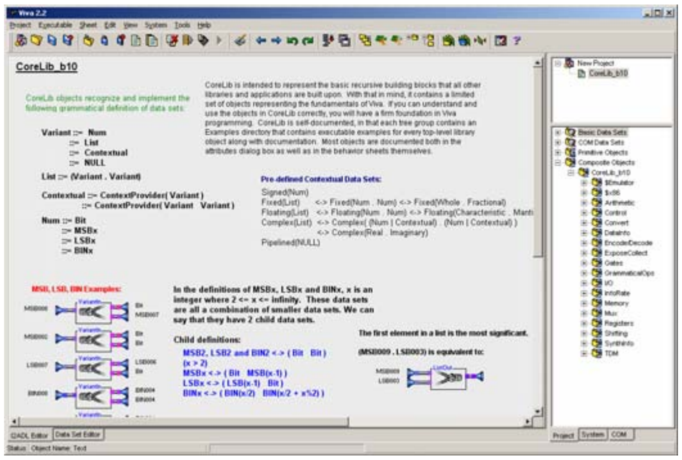
4.2 The Star Bridge Programming Environment

VIVA® is Star Bridge's proprietary development environment. It provides a graphical user interface based on "drag and drop" design principles. VIVA's design language, "Implementation Independent Algorithm Description Language" or IIADL, is object-based and uses object attributes to specify different options within the design. Fig. 4 shows the VIVA Integrated Development environment (IDE).

Designs within VIVA can be allocated to execute on either the Intel platform or the FPGA array. Objects that execute on the Intel processor are implemented using pre-built VIVA libraries within Windows. VIVA is able to use Windows COM (Component Object Model) to communicate with other Windows applications, allowing Windows applications that make use of COM methods to be integrated in with VIVA FPGA designs.

VIVA provides its own proprietary synthesis tool for designs that will run on user FPGAs. The target of the synthesis tool is defined by a system description. VIVA includes system descriptions for the Pentium x86, which allows simulation of the design in an x86, and for specific PEs in the FPGA array. VIVA's system descriptions allow VIVA to abstract out the target hardware description from the design to be implemented, allowing VIVA to target hardware other than its own.

If the design is targeted for a PE, the synthesis EDIF output is fed into Xilinx place and route tools to produce a final binary, which VIVA then loads onto the Intel processors and the FPGA array. After initial execution, the VIVA binary can be saved in a file format that can be loaded using VIVA command-line tools or within the VIVA development environment.



CoreLib_b10

CoreLib objects recognize and implement the following grammatical definition of data sets:

```
Variant ::= Num
        ::= List
        ::= Contextual
        ::= NULL
List ::= (Variant . Variant)
Contextual ::= ContextProvider( Variant )
           ::= ContextProvider( Variant  Variant )
Num ::= Bit
    ::= MSBx
    ::= LSBx
    ::= BINx
```

CoreLib is intended to represent the basic recursive building blocks that all other libraries and applications are built upon. With that in mind, it contains a limited set of objects representing the fundamentals of Viva. If you can understand and use the objects in CoreLib correctly, you will have a firm foundation in Viva programming. CoreLib is self-documented, in that each tree group contains an Examples directory that contains executable examples for every top-level library object along with documentation. Most objects are documented both in the attributes dialog box as well as in the behavior sheets themselves.

Pre-defined Contextual Data Sets:

```
Signed(Num)
Fixed(List)     <-> Fixed(Num . Num) <-> Fixed(Whole . Fractional)
Floating(List)  <-> Floating(Num . Num) <-> Floating(Characteristic . Mant
Complex(List)   <-> Complex( (Num | Contextual) . (Num | Contextual) )
                <-> Complex(Real . Imaginary)
Pipelined(NULL)
```

MSB, LSB, BIN Examples:

In the definitions of MSBx, LSBx and BINx, x is an integer where 2 <= x <= infinity. These data sets are all a combination of smaller data sets. We can say that they have 2 child data sets.

Child definitions:

```
MSB2, LSB2 and BIN2 <-> ( Bit  Bit )
(x > 2)
MSBx <-> ( Bit  MSB(x-1) )
LSBx <-> ( LSB(x-1)  Bit )
BINx <-> ( BIN(x/2)  BIN(x/2 + x%2) )
```

The first element in a list is the most significant.

(MSB009 . LSB003) is equivalent to:

Fig. 4. Star Bridge IDE

4.3 Designing within VIVA

VIVA defines basic data types, from which objects can be built. The basic types are Bit, Variant, Vector, NULL, LSB, MSB and BIN. Bit is the only non-abstract fundamental data set. Within VIVA, data sets are basically recursive combinations of Bits. Each type therefore has two basic objects associated with it: an Exposer and a Collector. A data set Exposer takes the data set and outputs its two children data sets. A collector does the opposite: it takes two child data sets and combines them into one parent data set. For instance, a BIN016 (16-bit binary) data set, when exposed, will produce two BIN008 (8-bit binary) data sets. A MSB016, when Exposed, will produce a MSB001 and a MSB015. This is equivalent to splitting out the most-significant bit of a std_logic_vector(15 downto 0), and returning a std_logic and a std_logic_vector(14 downto 0).

The above definition of types allows for recursion of large data sets into small data sets, which is useful when a large operation is comprised of smaller operations of the same type. For instance, a 16-bit

AND gate can be built using a tree of 2-bit AND gates, which can be built recursively. In order for recursion of an operation to be properly implemented, two objects for the operation must be created: a base object of the smallest acceptable operand size, and a recursive version of the object, which tells VIVA how to recurse down to the base operand object. Operands can be overloaded by simply copying the operand object design and changing the input data types, effectively overloading the operand type to handle different data sets. Objects can incorporate other types of objects, which allows for the hierarchical building of basic objects.

VIVA provides basic operands such as AND, OR, and INVERT, which are standard to VIVA. These basic objects can be implemented in multiple system descriptions. VIVA objects, in general, can be designed to be implemented in multiple system descriptions or in a specific system description only. VIVA provides a library of additional operands, and data types built from the standard data types. This library, Corelib, has objects ranging from registers through state machines used for design control to file I/O objects for data transfer from and to a Windows file (see Fig. 5). This library is meant to be the basis for designs built in VIVA.

4.4 Debugging within VIVA

Debugging a design within VIVA is performed by loading the x86 system description into a design simulate design components within the x86 environment. There is no in-place hardware debugger, although one is planned for future release. The simplest method of input and output testing is displaying inputs and outputs using widgets, which can be displayed using scrollbars, textboxes, graphics, and a number of other selections, as in Fig. 5. The clock can be manually single-stepped or continuously run. Windows COM objects can be integrated in to provide advanced debugging of the data output of a design.

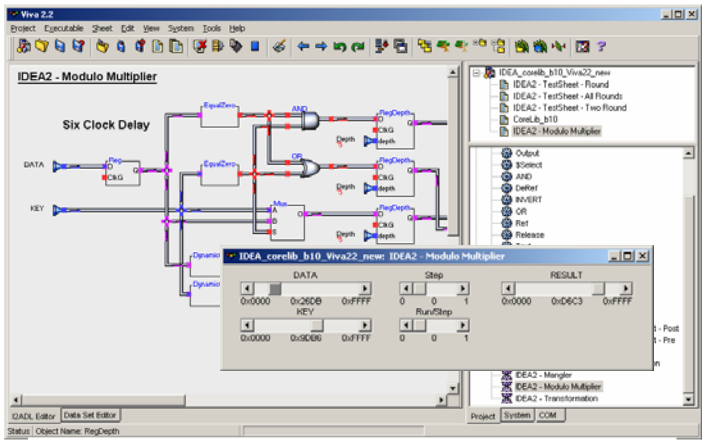


Fig. 5. VIVA Synthesis Output using the X86 System

5. IDEA

5.1 The IDEA Algorithm

The International Data Encryption Algorithm (IDEA) is a symmetric block cipher developed by Xuejia Lai and James Massey of the Swiss Federal Institute of Technology and published in 1990[5]. At that time it was suggested as a candidate to replace DES, however its widest adoption has been in PGP, which has insured widespread use of the algorithm.

IDEA uses a 128-bit key to encrypt data blocks of 64 bits. IDEA consists of eight rounds follow by a Transformation half-round that provides a 64-bit encrypted output. A round consists of a Transformation half-round and a Mangler half-round, of which an excellent description is provided by William Stallings in the book “Cryptography and Network Security: Principles and Practice” [6]. IDEA makes use of three basic operations to carry out encryption and decryption: a 16-bit XOR operation, a 16-bit modulo addition operation (mod 2^{16}), and a 16-bit modulo multiplication operation (mod $2^{16} + 1$). In addition, an all-zero operand input to a modulo multiplier within IDEA equals 2^{16} for internal calculations.

Each round requires six keys: four for the Transformation half-round and two for the Mangler half-round. For the total 8.5 rounds required for IDEA encryption, 52 16-bit keys are required. The first eight keys are provided by the input key. Each additional set of eight keys is generated by performing a circular left shift of 25 bits of the previous eight key set.

5.2 A Common Design Choice for IDEA Implementation within SRC and Star Bridge

The XOR and modulo addition operations of IDEA can be easily implemented. The most difficult operation, multiplication mod ($2^{16} + 1$) with the input of $0 = 2^{16}$, can be broken down into three cases:

multiplication of two nonzero inputs, multiplication where one input is zero and multiplication where both inputs are zero. For multiplication of two nonzero numbers, the following rule is used [6]:

$ab \bmod (2^n + 1) = (ab \bmod 2^n) - (ab \operatorname{div} 2^n)$
$\quad$ if $(ab \bmod 2^n) \geq (ab \operatorname{div} 2^n)$
$ab \bmod (2^n + 1) = (ab \bmod 2^n) - (ab \operatorname{div} 2^n) + 2^n + 1$
$\quad$ if $(ab \bmod 2^n) \leq (ab \operatorname{div} 2^n)$

For the remaining two cases, zero tests are used to mux in appropriate nonzero results and constants to provide the correct answer. To minimize latency, the use of a Virtex II hardware multiplier was desired to implement the two equations above. Since a 16x16 multiply operation using a single Virtex II 6000 block multiplier requires over 10 ns, a two-level three pipeline "divide-and-conquer" strategy was used to meet the 10 ns timing constraint. This method requires four 8x8 multiplies along with column additions [7].

Design choices for IDEA centered on making each half-round modular to create a repetitive instantiation, therefore key scheduling is broken into a unit that can be implemented in a modular round. The solution for key scheduling requires the generation of six keys for each round. After eight keys have been consumed, though, a 25-bit rotate left operation is required. To accommodate the above two constraints, a unique constant is input to each round's key scheduler, which selects a mux that determines where the key rotate operation occurs within that round, if required.

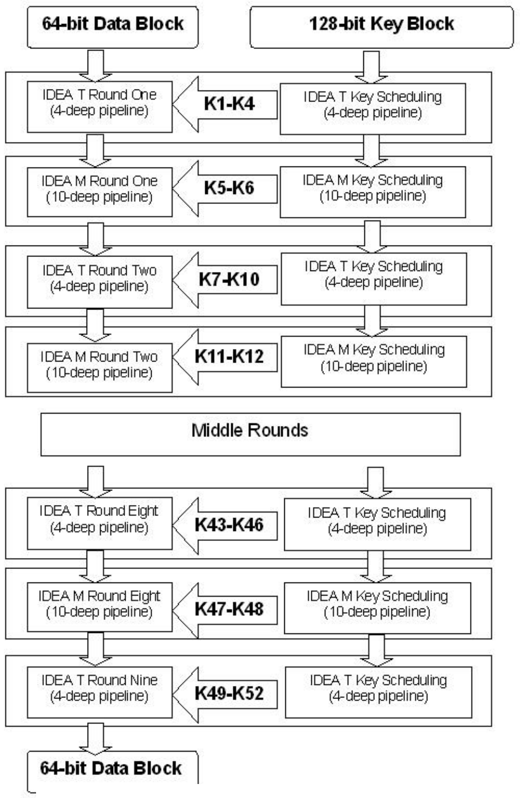

64-bit Data Block
128-bit Key Block
IDEA T Round One (4-deep pipeline)
K1-K4
IDEA T Key Scheduling (4-deep pipeline)
IDEA M Round One (10-deep pipeline)
K5-K6
IDEA M Key Scheduling (10-deep pipeline)
IDEA T Round Two (4-deep pipeline)
K7-K10
IDEA T Key Scheduling (4-deep pipeline)
IDEA M Round Two (10-deep pipeline)
K11-K12
IDEA M Key Scheduling (10-deep pipeline)
Middle Rounds
IDEA T Round Eight (4-deep pipeline)
K43-K46
IDEA T Key Scheduling (4-deep pipeline)
IDEA M Round Eight (10-deep pipeline)
K47-K48
IDEA M Key Scheduling (10-deep pipeline)
IDEA T Round Nine (4-deep pipeline)
K49-K52
IDEA T Key Scheduling (4-deep pipeline)
64-bit Data Block

Fig. 6. IDEA Block Diagram

The design was pipelined in order to achieve high throughput. Both data and key scheduling were pipelined, which allows this core to be used in IDEA breaking or encryption since different data blocks can be introduced with either the same or a different key at each clock cycle. Pipeline placement was chosen based on synthesized VHDL and Xilinx place and route results for a target Xilinx II 6000 FPGA and a timing constraint of 10 ns (100 MHz). The final design has a pipeline latency of 116 clocks: each Transformation half-round requires four clock cycles and each Mangler half-round requires ten clock cycles. Fig. 6 shows a block diagram of the design.

6. IDEA within SRC

6.1 Design Implementation within SRC

To implement an algorithm within SRC, at least two source files are required. *main.c* is responsible for reading in data and calling a user-defined function that loads a user-logic bitstream into the MAP processor. The second source *IDEA_test.mc* implements the MAP function that is called from *main.c*, and describes a bitstream to be loaded into the MAP processor. This MAP function calls SRC functions that control the data transfer between the Intel platform and the MAP processor, and uses a C for-loop structure for passing data to C commands that implement data processing within the MAP processor and to user HDL macros (see Table 2) that are also loaded within the MAP. In addition, an "info" file is required to specify attributes of the IDEA VHDL instantiation for the MAP compiler.

Two test cases were implemented within SRC, as shown in Tables 1 and 2. The first test case instantiates the whole IDEA algorithm within VHDL, and uses the MAP C function to pass data. The SRC info file for case one defines the VHDL user macro as a functional pipelined macro with a 116-clock latency. The second case instantiates half-rounds within VHDL, and uses the MAP C function to instantiate the 8.5 rounds required by IDEA. The SRC info file for case two describes two VHDL macros, both being defined as functional pipelined macros with a 4 or 10-clock latency. These test cases are representative of SRC design options, and allow for an FPGA resource and timing comparison between the two methods.

To test for latency results, a Unix high-resolution time structure was used to time of the HLL function call within *main.c*, while SRC timer calls were used within *IDEA_test.mc* to time the input and output DMA data transfers and the time to process the data. HLL time reads are in *us*, while timer reads within SRC provide the current clock tick. A conversion to ms was performed to arrive at the final timing calculations.

Table 1. Portions of IDEA Rounds Instantiated using VHDL

```
  -- Round 8.
  signal outkeyhigh_half_round8,   outkeylow_half_round8,
         outblockdata_half_round8, outkeyhigh_round8,
         outkeylow_round8,         outblockdata_round8: std_logic_vector(63 downto 0);
begin
  -- Round 1.
  round1_T: component IDEAroundT
    port map (clk => clk, inkeyhigh => inkeyhigh, inkeylow => inkeylow,
              mux_sel => X"00000000", inblockdata => inblockdata,
              outkeyhigh => outkeyhigh_half_round1, outkeylow => outkeylow_half_round1,
              outhalfblockdata => outblockdata_half_round1);

  round1_M: component IDEAroundM
    port map (clk => clk, inkeyhigh => outkeyhigh_half_round1,
              inkeylow => outkeylow_half_round1, mux_sel => X"00000000",
              inhalfblockdata => outblockdata_half_round1,
              outkeyhigh => outkeyhigh_round1, outkeylow => outkeylow_round1,
              outblockdata => outblockdata_round1);
```

Table 2. Portions of IDEA Rounds Instantiated using C

```
  /* Round 8 */
  uint64_t outkeyhigh_half_round8, outkeylow_half_round8, outblockdata_half_round8,
           outkeyhigh_round8, outkeylow_round8, outblockdata_round8;
  ……
  int i;
  ……
  cm2obm_c(c, data_in, size*8);
  wait_server_c();
   read_timer(start_loop);

  for (i=0; i<size; i++) {
    keyhigh = a[i];
    keylow  = b[i];
    data    = c[i];

    /* Round 1 */
    IDEAroundT(keyhigh, keylow, (uint32_t)0x0, data, &outkeyhigh_half_round1,
               &outkeylow_half_round1, &outblockdata_half_round1);
    IDEAroundM(outkeyhigh_half_round1, outkeylow_half_round1, (uint32_t)0x0,
               outblockdata_half_round1, &outkeyhigh_round1, &outkeylow_round1,
               &outblockdata_round1);
```

6.2 Results of SRC Timing

Results of the SRC implementation of IDEA are summarized in Tables 3 and 4. The timing results were independent of which of the two SRC implementations of IDEA described in Section 6.2 was used for the measurements.

In Table 3, the execution times are provided for four input data sizes ranging from 5 MB to 20 MB. In each case, the encryption was accomplished by ten calls to the MAP function with a proportional amount of input data to the total data processed. An end-to-end time includes a single MAP configuration time in the range of 102 ms, and the total MAP processing time, which is proportional to the amount of data being encrypted. Since the reconfiguration of the MAP FPGA can be performed before any input data becomes available for processing, this configuration may be treated as a part of a one-time set up routine.

It is worth noting that the time spent for processing data inside of the MAP FPGA (MAP FPGA Processing Time) constitutes less than 8% of the Total MAP Processing time. Instead, the majority of the time is spent for transferring data to and from the MAP using the DMA transfer between the microprocessor's Computer Memory and the MAP's On-Board Memory. The MAP Transfer In Time is greater than the MAP Transfer Out Time because in our implementation of IDEA, each input consists of both a data block (64 bits) and the corresponding key (128 bits), while an output includes only an encrypted data block (64 bits).

In Table 4, the corresponding MAP Processing and Data Throughputs are calculated. All throughputs are for the most part independent of the amount of data being processed. The MAP FPGA Processing Throughput approaches the theoretical maximum of 64 bits per clock cycle = 800 MB/s. The only reasons for a slightly smaller value of this throughput are the latency of the pipelined architecture of IDEA (116 clock cycles) and a control overhead associated with implementing a loop structure within the MAP function, measured to be equal to 47 clock cycles [13]. The MAP FPGA Processing time expressed in the number of clock cycles is equal to:

Encryption Unit Latency + (Number of Data Blocks Processed – 1) + Loop Control Overhead (1)

The MAP Transfer In Throughput is greater than the MAP Transfer Out Throughput because of the larger number of 64-bit words being processed without changing the OBM address. Finally, the Total MAP Processing Throughput (not including reconfiguration) is in the range of 60 MB/s, which is only 7.5% of the theoretical maximum of 800 MB/s. The limited data transfer bandwidth and a lack of overlapping between data transfers and data computations inside of the MAP FPGA contribute to this considerable slow down.

Table 3. FPGA Processing and Configuration Times

Data Size	PC End-to-End Time (ms)	MAP FPGA Conf Time (ms)	MAP Total Processing Time (ms)	MAP Data Transfer In (ms)	MAP FPGA Processing (ms)	MAP Data Transfer Out (ms)
5 MB	185.6	102.0	83.6	53.0	6.6	24.0
10 MB	269.7	102.0	167.7	104.4	13.1	50.1
15 MB	350.8	102.2	248.7	153.5	19.7	75.4
20 MB	432.6	101.8	330.8	203.0	26.2	101.5

Table 4. FPGA Throughput Results

Data Size	MAP Total Processing Throughput (MB/s)	MAP Transfer In Throughput (MB/s)	MAP Transfer Out Throughput (MB/s)	MAP FPGA Processing Throughput (MB/s)
5 MB	59.8	282.9	207.9	761.1
10 MB	29.8	287.3	199.4	762.0
15 MB	20.1	293.1	198.8	762.3
20 MB	15.1	295.6	196.9	762.5

6.3 Results of SRC FPGA Timing and Resource Utilization

Table 5. SRC PAR Results

Test Case	Slices	Slice FFs	LUTs	Mult 18x18	Clock Period
VHDL Only	9006 (26%)	13221 (19%)	10111 (15%)	136 (94%)	11.579 ns
VHDL-C	11442 (33%)	18460 (27%)	10435 (15%)	136 (94%)	11.379 ns

SRC v1.4 compilers were used for MAP compilation. Synopsys v7.2 was used for systhesis, and Xilinx v5.2 was used for map, place and route. For the VHDL-only case, synthesis times averaged around two minutes, map times averaged around three minutes, while place and route (PAR) times averaged approximately 66 minutes. For the VHDL-C case, synthesis times averaged around two minutes, map times averaged around four minutes, while place and route (PAR) times averaged approximately 94 minutes. Table 5 shows Xilinx PAR results for both the SRC VHDL and VHDL-C instantiation cases. Using a VHDL-C combination to instantiate the higher level hierarchy of IDEA shows an 8% increase in the number of FPGA slice flip-flops used. This is due to SRC control logic that is inserted in its instantiation of the IDEA rounds. Clock timing differences were negligible.

7. IDEA within VIVA

VIVA was used to build up a design that matches the VHDL implementation. Fig. 7 shows portions of the Mangler half-round, and Fig. 8 shows portions of an IDEA test sheet. The VIVA PE system descriptions are not complete; a limitation of this is that the full I/O capabilities of the FPGA were not available. As a result, a limitation in this design is that the key inputs are constant, as the minimum number of inputs required is 192 bits (64 data + 128 key), while only 128 bits are available for inputs in the system description. Functional verification was performed using the same IDEA data file as in SRC.

Measuring of timing results of data transfer or data processing on the Intel side requires using a COM wrapper around a Windows timer, as a standard VIVA timer object is not available within the current library. A COM object was developed for timing; however, the CoreLib library's beta state of development prevented its integration with CoreLib's I/O objects.

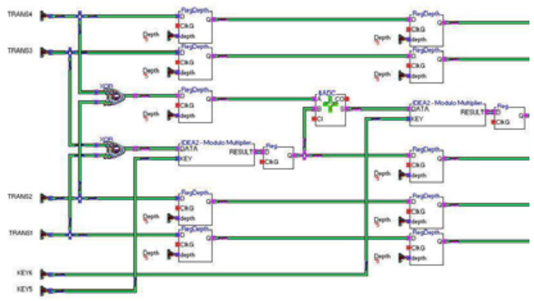


Fig. 7. VIVA Half Round Mangler Design

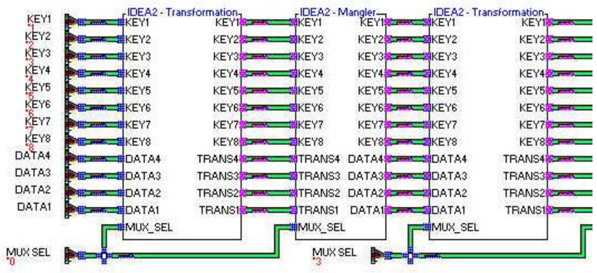


Fig. 8. IDEA in VIVA

7.1 Results of Star Bridge FPGA Timing and Resource Utilization

Table 6. Star Bridge PAR Results

Test Case	Slices	Slice FFs	LUTs	Mult 18x18	Clock Period
VIVA	17292 (51%)	16118 (23%)	7639 (11%)	136 (94%)	12.191 ns

VIVA v2.2 was used for synthesis, and Xilinx v4.2 was used for map and PAR. VIVA synthesis and PAR times scaled exponentially depending on the number of rounds synthesized: one round took 25 minutes, five rounds took approximately 7 hours, and the full 8.5 rounds took approximately 36 hours, due to the alpha state of their synthesis algorithms. The Xilinx map time for the full round implementation was 14 minutes, and PAR for the full round implementation was 85 minutes.

Table 6 shows Xilinx PAR results. Compared to the SRC VHDL-only design, the number of Slices used was 24% greater, while the number of Slice Flip Flops used was only 3% greater. An explanation for this discrepancy is that VIVA may be using only one LUT per CLB slice. The total number of LUTs used was 4% smaller. Clock timing was slightly longer. This comparison is tentative since the key inputs are constant versus the SRC design.

8. Crypto++ Software Implementation versus SRC

Crypto++ v5 was used to implement IDEA encryption within software on SRC's Linux-based workstation, to normalize the microprocessor capability between the software and the software-FPGA implementations. 20 MB of data was processed using ten calls to the Crypto library's IDEA encryption algorithm, using the same input data as SRC and VIVA. Timing measurements for IDEA encryption processing of the data were taken using a UNIX high resolution timer.

The time required to process 20 MB of data was approximately 8 seconds, which corresponds to a throughput of 2.5 MB/s. When compared to SRC's MAP Processing Throughput for IDEA, the SRC design is 24 times faster than the software implementation. When compared to the time including the FPGA configuration time, the SRC design was 18.5 times faster. Given continuous processing of large data blocks, the throughput advantage of SRC will approach the 24-times speed increase maximum.

9. SRC and Star Bridge Limitations

Both systems have limitations, which should be noted before conclusions can be understood.

SRC's compiler can only handle certain HLL constructs within a MAP processor subroutine. It is not a limitation in this design, and SRC has been adding significant functionality with each release.

Star Bridge's synthesis technology is in an alpha state. Significant improvements have been seen with the last few product releases; however the product is still in its early development stages. Star Bridge's CoreLib library is in a beta state, and design elements within the library are not final.

Both systems plan on implementing multiple FPGA use within their next software releases. SRC and Star Bridge currently cannot make use of more than one FPGA.

10. Conclusion

Reconfigurable computing is a platform that can provide larger and more efficient computing resources for implementing computationally intensive design solutions Both SRC and VIVA have unique reconfigurable computer environments that provide complimentary ways achieving this goal. This paper shows how the designer can use SRC's development environment to implement design objectives using different combinations of traditional HLL development with HDL development, and how Star Bridge's VIVA environment provides a new paradigm for FPGA design entry and synthesis. An IDEA encryption algorithm is implemented in both environments for comparison.

Both SRC and Star Bridge take a different approach in their design environment. The SRC system is based on traditional programming languages, builds upon known development paradigms, and can incorporate existing HLL source code for designs targeted at FPGAs. Star Bridge's VIVA provides a new development paradigm that has the potential to provide an easier method of design entry, and can target several different architectures in addition to its own.

The combination of a PC with FPGAs provides a high-speed design alternative to designs implemented solely in PC software. This paper shows that the FPGA's maximum throughput can be significantly reduced due to time required to configure the FPGA and data transfer between PC memory and memory available to the FPGA. While the effect of reconfiguration can be reduced with large data processing, the effect of data transfer is a consistent bottleneck to maximum throughput if large amounts of data need to be transferred.

References

1. Singleterry, R., Sobieszczanski-Sobieski, J. Brown, S., "Field-Programmable Gate Array Computer in Structural Analysis: An Initial Exploration", available at http://www.starbridgesystems.com
2. Parnell, K., Mehta, N., "Programmable Logic Design Quick Start Handbook", available at www.xilinx.com
3. SRC Inc. Web Page, http://www.srccomp.com
4. Star Bridge Systems Web Page, http://www.starbridgesystems.com
5. Lai, X., Massey, J., "A Proposal for a New Block Encryption Standard", Proceedings, EUROCRYPT '90, 1990.
6. Stallings, W., "Cryptography and Network Security: Principles and Practice", 2nd Edition, pgs. 102-109, 128, 1999.
7. Parhami, B., "Computer Arithmetic Algorithms and Hardware Designs", pgs 191-192, 2000.
8. Dai, W., Crypto++ v. 5, http://www.cryptopp.com, Sep. 2002.
9. VIVA User's Guide, Version 2.2
10. The SRC-6E C Programming Environment Guide, v1.2, 2003, available from SRC Computers.
11. The SRC-6E MAP Hardware Guide, 2003, available from SRC Computers.
12. 'Reconfigurable, Inherently Parallel "Hypercomputing" ', PowerPoint presentation, 2002, available from Star Bridge Systems.
13. Fidanci, O.D., Diab, H., El-Ghazawi, T., Gaj, K., Alexandridis, N., "Implementation trade-offs of Triple DES in the SRC-6e Reconfigurable Computing Environment," 2002 MAPLD International Conference, Sep. 2002.
14. Fidanci, O.D., Poznanovic, D., Gaj, K., El-Ghazawi, T., Alexandridis, N.,"Performance and Overhead in a Hybrid Reconfigurable Computer," Reconfigurable Architecture Workshop, RAW 2003, Apr. 2003.