ISSP Update: Delivering Performance
by Christoph Hecker, Product Marketing Engineer, NEC Electronics (Europe) GmbH and Steven Kawamoto, St. Program Manager, NEC Electronics America, Inc.

Over the past year, more than twenty-five customers have leveraged NEC Electronics’ experience with deep submicron design automation and intellectual property integration to develop high-performance, low-cost ISSP silicon for their system designs. One reason ISSP technology is being chosen is because it has a flow that frees designers from having to perform design tasks such as testing, making the design process faster and the number of tools needed fewer. Another reason is that having fewer tools means NEC Electronics can work with an EDA partner, such as Synplicity®, to develop customized ISSP EDA technology that makes it faster, easier, and more automated to achieve high performance. For example, the Synplify ASIC® solution is now being used in ISSP designs whose performance levels (up to 300 MHz) would be difficult to achieve with non-optimized tools.

ISSP devices have a five-metal layer architecture, in which the upper two layers can be customized easily to meet individual design requirements. The bottom three metal layers are predefined with NEC Electronics intellectual property and design-for-test (DFT) blocks, and include circuitry to reduce signal integrity effects and clock skew issues. This configuration facilitates high-fault coverage testing and eliminates the need to perform DFT steps. Testing methodologies, including scan, BIST, BSCAN, and TestBus, are embedded in the masters of the architecture.

Creative Uses for Spartan-3 Dedicated Resources
by Marc Baker, Staff Applications Engineer, Xilinx, Inc.

The Spartan™-3 FPGA logic resources include not only the four-input look-up tables that have become the standard building block of FPGAs, but also dedicated resources that provide fast and efficient implementation of special functions. These resources are automatically selected and optimized by the Synplicity® synthesis tools and Xilinx implementation tools. However, the user should consider creative ways of applying these resources when the design is created. Block RAM is highlighted there, with non-standard uses of the multipliers, multiplexers, and SRL16 shift registers briefly described. Each resource is actually much more flexible and applicable to wider variety of functions than their names would imply. Use the examples provided here to spur your own imagination.

Block RAM

Flexibility in Port Sizes

Each block RAM contains 18,432 bits of fast static RAM, 16K bits of which is allocated to data storage. In some memory configurations, an additional 2K bits is allocated to “parity” bits, although they can be used for any type of additional data. Each port of the true dual-port block memory is completely independent and supports multiple aspect ratios. Cascade multiple block RAMs to create deeper and wider memory organizations.
One traditional interpretation of automation is the use of machinery to perform large numbers of repeated tasks. It is also commonly accepted that the quality of machine generated product is lower than a carefully hand-crafted one. Automation is therefore often viewed as a productivity benefit, at the cost of sub-optimum results.

In fact, the above definition matches mechanization instead of automation. Mechanization is the lower layer of actions that encompasses all possible transformations to generate the final result. Synthesis tools feature hundreds of such mechanical actions: decomposition, factoring, sizing, buffering, cloning, merging, splitting, load shielding, restructuring, to name a few. Yet, the synthesis process is more complex than serially sequencing all possible transformations. Traditional synthesis tools rely on user input to converge to a good result. It is not unusual to have hundreds of commands from which the user must compose rather complex scripts.

Automation actually goes well beyond mechanization. A truly automated process makes intelligent decisions on whether and where to apply selected actions. It requires a different set of heuristics, which analyze the problem in order to make these decisions. From a software development perspective, human intelligence is actually embedded in the heuristics, which try to mimic human brains. The mechanical transformations are in fact similar to the muscles. The holy grail of such automated process is to deliver results of quality equal to the hand-crafted ones.

On top of the productivity gains that come from mechanization, automation provides several other benefits. First, it greatly reduces the effort required to achieve optimum results. Indeed, very minimal user intervention is required. Moreover, users benefit from all the methodology that has been embedded into the automation layer. One can truly focus on his value-added task and grow design expertise rather than tool expertise.

A second benefit lies in reliability. Let's illustrate this point through a simple example. Consider flying an airplane that has 30 simple on-off switches. How many different combinations are there? Well, $2^{30}$ or more than a billion possibilities. The space to explore is in fact so immense that regardless of the testing effort from the manufacturer, the coverage is very likely to be low. In other words, you are on your own. Automation, however, significantly reduces the exploration space. For example, conflicting and redundant actions are eliminated by construction. Quickly, the coverage of the tool goes up and much less failures are to be seen.

Finally, automation allows organizations to stay on the leading edge. As technologies and methodologies progress, they are incorporated into new releases of tools. These transformations are automatically applied when necessary, without any need to re-train users on new, improved, obsolete, or conflicting features.

Synplicity's mission is to contribute to your success. That is why automation is an essential piece of our philosophy.
“ISSP Update” continued from page 1

ISSP Standard Family
The ISSP standard family (ISSP-STD) covers densities up to 1.5M usable gates and up to 3.7 megabits (Mb) of embedded configurable memory.

High-Speed Interface Family
Taking direct aim at the high-speed interfacing needs of networking and telecom applications, NEC Electronics has developed a high-speed interface family (ISSP-HSI) that leverages process-proven SerDes core technology. Frequencies range from 622 Mbps through 1.25 Gbps and 2.5 Gbps to 3.125 Gbps to fulfill individual application requirements. Interfaces supported include InfiniBand, XAU1, Gigabit Ethernet, and 3G10. NEC Electronics has also kept power dissipation in mind. With 220 mW per channel at 3.125 Gbps, ISSP truly is a low-power ASIC. With up to 16 SerDes cores, four analog phase-locked loops, and 32 prediffused delay line loops, ISSP-HSI can offer the high complexity of an ASIC with radically lower costs and shorter turnaround times.

Tool Flow For Performance
ISSP designers can leverage NEC Electronics’ sophisticated OpenCAD™ design environment that includes physical floorplanning, timing-driven layout, hierarchical design, and other leading-edge technologies. Furthermore, NEC Electronics and Synplicity’s partnership in customized synthesis with the Synplify ASIC solution has been in production since February this year, and is integrated with the OpenCAD system. The Synplify ASIC product has been shown to deliver significantly better timing performance, run time, and area utilization than non-optimized synthesis flows, making the completion of high-performance ISSP designs faster and more automated as well. Users of Synplify ASIC software have the convenience of a built-in, performance-optimized, custom ISSP synthesis engine; simply loading an ISSP library invokes the custom optimizations and features.

Availability
Engineering samples of ISSP-STD and the ISSP-HSI device libraries, as well as a reference design kit including a hardware board and software development kit, can be obtained by contacting your local NEC Electronics sales office. Volume production of ISSP-HSI will be in Q4 2003. NEC Electronics targets turnaround time for devices to be as fast as four weeks from design to engineering samples. Non-recurring engineering costs for the ISSP products start at less than $75,000.

FPGA Solutions

“Creative Uses for Spartan-3” continued from page 1

Two Independent Single-Port RAMs Using One Block RAM
A simple trick allows a single block RAM to behave as if it were two completely independent single-port memories, effectively doubling the number of RAM blocks on the device. Tie the most significant address bit of one port High and the most significant address bit of the other port Low, evenly splitting the available RAM between them. Both ports are independent, each with its own memory organization, data inputs and outputs, clock input, and control signals. For example, Port A could be 256x36 while Port B is 2Kx4.

A 256x72 Single-Port RAM Using One Block RAM
A wider RAM can be created by again tying the MSB High on one port and Low on the other, but sharing the rest of the address lines and the control inputs in common. In this way a 256-deep by 72-bit wide single-port RAM can use a single block RAM. One half contains the lower 36 bits and the other half stores the upper 36 bits.

“Creative Uses for Spartan-3” continued on page 4
Fast Complex State Machines

A 128-state, 8-way branch finite state machine with 38 total state outputs fits in a single block RAM. Port A is configured as 2Kx9 but used as a 1Kx9 single-port ROM. Seven outputs step through the 128 states while the remaining three address inputs determine the eight-way branch. Port B is configured as 512x36 and used as a 256x36 single-port ROM. It receives the same 7-bit current-state value from Port A, and drives 36 outputs that can be arbitrarily defined for each state. This same basic architecture can be modified to form a 256-state finite state machine with four-way branch, or a 64-state state machine with 16-way branch. The advantages of this design are low cost (a single block RAM), high performance (over 125 MHz), absence of lay-out or routing issues, and complete design freedom.

Fast, Long Counters Using RAM

A 20-bit binary up counter, with clock enable and synchronous reset, can be implemented in a single block RAM. The 20-bit counter can be constructed from two identical 10-bit binary counters, with the lower 10-bit counter enabling the upper 10-bit counter every 1024 clock cycles. Both ports are configured as a 1Kx18 ROM with the 10 LSBs connected directly to the ten address inputs. The eleventh data bit of the lower order counter forms the terminal count and drives the enable of the upper counter.

Four-Port Memory

Each block RAM is physically a dual-port memory. However, due to the block RAM’s fast access performance, it is possible to create multi-port memories by time-division multiplexing the signals in and out of the memory. A block RAM with some additional logic easily supports up to four ports, as shown in XAPP228.

Content-Addressable Memory

Content-Addressable Memory (CAM) is used in a variety of networking and data processing applications. In CAM applications, the content is the driving input and the output indicates whether or not the content exists in memory and, if so, provides a reference to its location. See the CORE Generator System’s CAM module and XAPP260.

Implementing Logic Functions Using Block RAM

Block RAM can be used as a larger version of the basic four-input look-up table. A 16Kx1 block RAM has 14 inputs and a single output. Consequently, block RAM is capable of implementing any possible arbitrary logic function of up to 14 inputs, regardless of the complexity. As another example, a 2Kx9 block RAM can implement nine separate arbitrary Boolean logic functions of 11 inputs, as long as the inputs are shared. There cannot be any asynchronous feedback paths in the logic, such as those that create latches. The logic output must be synchronized to a clock input, since Block RAM does not support asynchronous reads.

Mapping Logic into Block RAM

The Xilinx ISE software can map logic functions into block RAM when using the MAP -bp option. A final flip-flop output is required as block RAMs have a synchronous, registered output. The mapping software packs the flip-flop with whatever LUT logic is driving it. To specify which register outputs will be converted to block RAM outputs, create a file containing a list of the net names connected to the register outputs, and set the environment variable XIL_MAP_BRAM_FILE to the file name.

Function Tables

Sometimes it is faster to look up an answer than deriving it. For example, instead of creating the CLB logic that implements the polynomial equation \( Y = A x^3 + B x^2 + C x + D \), the function can be pre-computed for each input value of \( x \) and stored in a block RAM. A 512x36 block RAM implements the above equation for input values between 0 and 511. Some other look-up functions possible in a single block RAM include complex arithmetic functions of a single input such as log and square root, binary/BCD converters, sine/cosine lookup tables, and seven-segment LED converters. See XAPP463 for more information on these and other applications for block RAM.

Multipliers

Dedicated 18x18 multipliers are of obvious value when multiplying two binary numbers of up to 18 bits. But there are other possible applications to consider. Since binary multiplication by 2\(^n\) is the same as shifting the value \( n \) places, a multiplier can be used as a shifter of 0 to 16 positions. A magnitude return or absolute value generator can be implemented by multiplying by a value with one as the LSB and the MSB of the input value in all the other bit positions. Both complex multiplication (including imaginary components) and floating-point multiplication (including exponents) can be implemented simply by using more than one multiplier.

“Creative Uses for Spartan-3” continued on page 5
“Creative Uses for Spartan-3” continued from page 4

Color conversion can be described as a $3 \times 3$ matrix multiplication by a constant, which requires nine multiplies. The high speed of a Spartan-3 device allows the user to “time share” the multiplier. Instead of nine multipliers, the design feeds nine sets of inputs resulting in nine sets of results at nine times the clock rate of the system, reducing the multiplier count to one. See XAPP284 for more information on matrix math, and XAPP467 for more information on Spartan-3 multipliers.

**Multiplexers**

The Spartan-3 architecture includes dedicated multiplexers that improve the performance and density of not just wide multiplexers but almost any wide-input function. Using these resources, a 32:1 multiplexer fits in just one level of logic, as do some Boolean logic functions of up to 79 inputs. For applications like comparators, encoder-decoders, or case statements, these resources provide an optimal solution. See XAPP466 for more details.

**Shift Registers**

The SRL16 is an alternative mode for the look-up tables where they become 16-bit shift registers, efficient for delay lines or latency compensation. They can also be used as a Linear Feedback Shift Register, or LFSR. LFSRs sequence through $2^n - 1$ states, where $n$ is the number of flip-flops. The sequence is created by feeding specific bits back through an XNOR gate, which can be conveniently located in the non-memory half of a Spartan-3 logic block. Maximal-length LFSRs need taps taken from specific positions within the shift register, and there are multiple ways these taps can be accessed, as described in XAPP210. LFSRs can replace conventional binary counters in performance critical applications where the count sequence is not important (e.g., FIFOs), and are also used as pseudo-random number generators. Another unusual application of the SRL16 is in a non-standard counter or state machine. Any desired repeated sequence of 16 states can be achieved by feeding each output with an SRL16. Cascading the SRL16 allows even longer arbitrary count sequences. See XAPP465 for more on using the SRL16.

**Conclusion**

Block RAM, multipliers, multiplexers, and shift registers are all special resources in the Spartan-3 architecture. But do not limit your imagination to only using them for the basic functions they are named for. With creative design techniques you can use these powerful features for a wide variety of functions, improving density and performance of your Spartan-3 designs.

---

**Extensible Network Security Device Built with the Amplify Solution**

by John Lockwood, Washington University in Saint Louis

This year, the demand for Internet security has significantly increased. Internet connected hosts are now frequently attacked by malicious machines located throughout the world. Users are now bombarded daily with tens to hundreds of SPAM messages. Internet worms and viruses are now rapidly spreading from machine to machine. Networks can be made more secure by filtering traffic with content-aware, extensible network security devices. By actively dropping harmful packets and rate-limiting unwanted traffic flows, the damage caused by such attacks can be reduced.

While some types of attacks can be thwarted solely by examination of packet headers, other types of attacks – such as network intrusion, Internet worm propagation, and SPAM proliferation – require that network security devices process entire data content. Few existing devices have the capability to scan entire packet payloads. Of those that do, most are software-based and cannot process packets at the high-speed rates used by modern networks.

**System-on-Chip Firewall Technology**

A single-chip, Internet security device has been developed that protects high-speed networks from present and future threats. The device was implemented on a single Xilinx Field Programmable Gate Array (FPGA) using the Synplicity Amplify Physical Optimizer solution. In order to protect networks against current threats, the baseline system parses Internet protocol headers, scans packet payloads, classifies traffic based on both the content and header files, and then performs per-flow queuing with selective transmission. In order to protect against future threats, the device allows new hardware components to be integrated as modules into FPGA hardware. The top-level architecture of the device is shown in Figure 1.
Extensible Network continues from page 5

When a packet first enters the device, it is processed by a set of layered protocol wrappers. These wrappers segment and reassemble frames; verify and compute checksums; and read and write the headers of the Internet packet. Once the packet has been parsed by the wrappers, a payload scanner searches the entire content of the packet for keywords and regular expressions. A set of bits are set to identify which regular expressions were matched in the payload. Next, a Ternary Content Addressable Memory (TCAM) classifies the packet based on the value of the headers and the payload match bits. The result is a flow identifier (Flow ID) that is forwarded to an extensible module.

Extensible modules enable the network security device to perform additional, customized packet processing or functions. Extensible modules are implemented as a VHDL or Verilog component and are placed within a region of the FPGA device. Modules can examine, modify, add, queue, or drop packets as they pass through the system. A memory interface is provided to the module so that it can access off-chip Synchronous Dynamic Random Access Memory (SDRAM), if needed. The extensible modules can also modify the Flow ID, which is useful for implementing modules that aggregate flows together to protect a network from a Distributed Denial of Service (DDoS) attack.

The resulting flow identifier is output to a queue manager, which either drops the packet or schedules it for transmission on the outgoing network link. The packet itself is stored in a flow buffer. In order to buffer large amounts of traffic, the network security device stores packets in off-chip SDRAM. Once the packet scheduler determines it is time for the packet to be sent, data is read from SDRAM, processed by the layered protocol wrappers, and then transmitted on either a Gigabit Ethernet or SONET line card.

Synthesis Flow

An automated design flow enables the network security device to easily be reprogrammed to implement additional functionality. As shown in Figure 2, the process begins when a new feature is uploaded to an on-line database. Next, the set of modules that implement the desired feature set is extracted from the database. For the payload scanner module, VHDL code is automatically generated to create finite state machines that search for the regular expressions specified by the administrator. The resulting VHDL is then synthesized into logic using Synplicity tools. In order to constrain where modules were placed, the Amplify solution is used to constrain the placement of logic. Likewise, the I/O are constrained to match the pads used by the FPGA on the implementation platform. This resulting netlist is placed and routed using FPGA tools and a bitstream is generated. The bitstream is then deployed over network into Field programmable Port Extender (FPX) platform.

Results

Using an AMD Athlon 2400MP to perform all of the steps of the design flow shown in Figure 2, a complete network security device with the protocol wrappers, TCAM, forwarder, and queue manager was synthesized in 9 minutes. Most of this time required to build the new circuit was used by the Xilinx tools to place and route the circuit. The logic of the network security device occupies 43% of the logic and 39% of the block RAMs of the Xilinx Virtex XCV2000E part used on the FPX. This circuit synthesized to operate at 62.5 MHz. Since each of these components process 32 bits of data in every cycle, the SOC firewall achieves a throughput of 32*62.5MHz = 2 Gigabits/second. A view of the resulting placed and routed FPGA is shown in Figure 3. Note that the center region of the chip was left available for insertion of extensible modules. This area of the chip can be used to implement new features.

In-system Testing

The Field Programmable Port Extender (FPX) platform was used to implement the network security device and process real Internet traffic passing between a host and the Internet. The payload scanner was programmed to find regular expressions that contained signatures of computer viruses. The payload scanner examined the full payloads of all packets passing through the device and the TCAM was programmed to drop traffic that contained the virus. As expected, the virus-infected traffic was dropped and other traffic flows received their fair share of network bandwidth.
**Synplify® and Synplify Pro® Software**

**Q:** How can I view only the logic associated with selected instances/pins of my design and remove the rest?

**A:** The HDL Analyst® product has a feature called “Isolate Paths.” As the name suggests this option isolates those paths associated with the highlighted instance/pin. If this option is used on an instance then the resulting view contains this instance and the logic associated with each of its pins. Each pin would be traced back until the tool finds a register or a port connected to the pin directly or through combinatorial logic and all this will be displayed.

If the Isolate Paths option is used on a pin, then the pin will be traced back until the tool finds a register or port and the entire logic along this path will be displayed. If the pin feeds several registers through some logic, then all these paths associated with the pin will be displayed.

Example: To observe a specific register to register path alone, in the HDL Analyst® editor you can select the output pin (Q pin) of the first register then right click ➔ Isolate Paths. The result would be a filtered view containing the register whose output pin was selected and all the logic it feeds and the destination flop/flops.

In the Schematic in Figure 1, if you are interested in only the register to register path circled, then select the output pin of the starting register as shown below and the right click ➔ Isolate Path, the result will be the schematic shown in Figure 2.

![Figure 1](image1.png)

![Figure 2](image2.png)

**Note:** The difference between Isolate Path and Expand Paths:

The Isolate Path feature is always used in an unfiltered schematic to filter and observe logic associated with selected instance or pins. The Expand Path feature is however used in a filtered schematic to observe more logic associated with the selected pin. So, the Isolate Path feature reduces the logic displayed in the schematic, whereas the Expand Path feature adds extra logic to the existing filtered schematic.

**Amplify® Physical Optimizer™**

**Q:** How do I constrain wide datapaths for an efficient, high-performance implementation?

**A:** If the critical path in your design includes a datapath that is too wide for a single region in your floorplan, the Amplify Physical Optimizer product can help. For example, if you have a 16-bit multiplexer that doesn’t fit in a region, then:

- Use the Amplify software's bit-slicing feature to divide the datapath.
- Replicate the register and place with the common logic of the critical path.
- Place half of the critical path in one region (R1) and the other half of the critical path in another region (R2) as shown in Figure 3.

![Figure 3](image3.png)

**Identify™ RTL debugger**

**Q:** How do I use the simultaneous debugging capability of the Identify 1.3 product?

**A:** When multiple Identify debugger licenses are available, multiple FPGAs residing on a single board can now be debugged concurrently through a single cable. This capability is based on semaphores that allow more than one debugger to share the port. Earlier versions of the Identify solution were restricted in that only a single debugger could be actively polling for a trigger. In the Identify 1.3 product, multiple debuggers can poll for trigger conditions on a single JTAG chain.

There are two software events that control a debugger: a local event specific to the running debugger and a global event common to all debuggers. These events allow cross triggering between debuggers through the software as opposed to a trigger physically wired between two FPGAs (Figure 4).
Software triggering controls are used to stop data collection and download the data. The software triggering controls are available from the command-line interface:

```
remote_trigger [-all | <process_id>]
```

Using the “-all” flag will cause all other Identify debuggers on the system to “trigger”. Otherwise only the debugger with the specified “process_id” will trigger. Use the command “pid” in the TCL command window to find the process id of the current debugger.

Additionally, the software trigger can be specified from the run command, so that the remote debugger will be triggered whenever originating debugger hits a trigger condition:

```
run -remote_trigger <process_id>
```

Certify® Software

Q: Why aren’t the gated clocks inside a Compile Point converted while the gated clocks outside are successfully converted into clocks and enables?

A: FPGAs have a limited number of low skew clock resources and logic on clock lines will easily cause clock skew problems. ASICs frequently use gated clocks to save power. Certify software has an option that will convert gated clocks into direct clocks and enable signals for clocked elements to make a design FPGA friendly.

A module specified as a compile point by the Certify software will lock down its port definition. A gated clock conversion will create both a clock and an enable signal. Since the compile point cannot accept the new enable signal because its port definition is locked down, then no conversion can be made inside the compile point.

To avoid this problem, move the clocked elements outside the compile point module.

Training Courses That Fit Your Needs

by Celine Sanie, Corp. Training Dept. Manager, Synplicity, Inc.

Whether you are a new or experienced user, Synplicity offers a variety of focused product training courses to help ensure your productivity in your daily job. With the flexibility of attending formal training classes, purchasing your own copy of the training material, or registering for the on-line, self-paced courses, you choose the option that best fits into your schedule and learning style.

Up-to-date Classroom and On-site Training Classes

All courses are taught by Synplicity Corporate Application Engineers that are experts in each of our products. A description of each course is available on our web page at www.synplicity.com/training. All training classes are given monthly at Synplicity’s headquarters and feature formal instruction, labs, and plenty of time to ask questions to your instructor. Classes can also be arranged at your facility.

On-line Self-paced Courses

Any user who is evaluating or owns the Amplify® Physical Optimizer™ solution can register for our on-line, self-paced course for the Altera or Xilinx technology. This course comes at no additional charge. The on-line course for the Synplify Pro® tool for Xilinx FPGAs takes the designer through an introduction of the Synplify Pro tool as well as Xilinx-specific and advanced features. All Xilinx FPGA designers will surely find all the tips they need in these pages!

To view the current schedule of available classes, get more information on Synplicity’s Training Courses, or to register, just go to www.synplicity.com/training! If you have any questions regarding our Training Program, simply e-mail them to training@synplicity.com.