SETIT 2005
3rd International Conference: Sciences of Electronic,
Technologies of Information and Telecommunications
March 27-31, 2005 – TUNISIA

Hardware based Security for Wireless Devices

Engr. Junaid Majeed
FEST Hamdard University
junaidmajeedhu@yahoo.com

Abstract Digital communication has become prevalent over the last few decades. The increasing popularity of the Internet and the proliferation of wireless networks have allowed people to conduct business and to communicate “online." Naturally, as the popularity of the medium increases, so do the concerns. Businesses and individuals alike are demanding security guarantees from the applications and services that they use. The most basic of these requirements are privacy and authenticity of data in transmission. Portable devices, including cell phones, Personal Digital Assistants (PDAs), and laptop computers are playing an increasing role in the lives of consumers. As these devices become more integrated with the daily routines of the end-users, they are more likely to contain more personal information, including telephone numbers, birthdates, contact lists, and perhaps even credit card, bank account, or other billing information. Loss of a device containing this type of information is not only inconvenient; it can easily lead to identity theft. As the theft of devices like cellular phones becomes more common, end-users would like to see products that not only contain the latest features, but also provide some assurance of security. For example, a cellular telephone that allows a user to change providers and technologies without buying new hardware would certainly find a place in the pockets of consumers. The objective of this effort is to demonstrate the viability of a secured run-time reconfigurable processor for streaming-data communication applications, and to develop structured methods for assembling and reconfiguring this type of processor.

Keywords Security ,VLSI , Algorithms

Introduction

Network processors are becoming a predominant feature in the field of network hardware. As new network protocols emerge and data speeds increase, contemporary general-purpose network processors are entering their second generation and academic research is being actively conducted into new techniques for the design and implementation of these systems. At the same time, systems ranging from secured military communications equipment to consumer devices such as cellular telephones and PDAs are being updated to provide network connectivity. Many of these devices require, or would benefit from, the inclusion of device security in addition to data security. Whether it is a top-secret encryption scheme that must be concealed or a personal device that needs protection against unauthorized use, security of the device itself is becoming an important factor in system design. Unfortunately, current network processor solutions were not developed with device security in mind. A secure network processor can provide the means to fill this gap while continuing to provide full support for emerging communication protocols.

Portable devices, including cell phones, Personal Digital Assistants (PDAs), and laptop computers are playing an increasing role in the lives of consumers. As these devices become more integrated with the daily routines of the end-users, they are more likely to contain more personal information, including telephone numbers, birthdates, contact lists, and perhaps even credit card, bank account, or other billing information. Loss of a device containing this type of information is not only inconvenient; it can easily lead to identity theft [1]. As the theft of devices like cellular phones becomes more common [2], end-users would like to see products that not only contain the latest features, but also provide some assurance of security. For example, a cellular telephone that allows a user to change providers and technologies without buying new hardware would certainly find a place in the pockets of consumers. These same users would also derive comfort from the idea that this leading-edge adaptable communication hardware is keyed to them personally and will not function should it be lost or stolen.

As this evolution in communication systems takes place, network hardware options are evolving as well. Older network systems relied upon application-specific integrated circuits (ASICs) to provide processing of network traffic. These ASICs provide the high throughput required, but lack the flexibility needed to support changing network protocols.

From a security standpoint, ASICs also fall short, as they can be disassembled at any time to discover their contents. This type of an attack on a system not only causes a loss of system integrity, but also results in a loss of the intellectual property contained within the device.

Newer network systems require programmable hardware resources and some have turned to general-purpose processors. In addition, processor-based systems are susceptible to security attacks based upon known processor functions and operational monitoring. Exploitation of known processor weaknesses might allow attackers to interfere with system operation. In an effort to provide better support for modern network systems, researchers have begun developing specialized network processors. The resulting processors range from pattern processors and Ethernet classifiers [20,21,22] to systems consisting of general purpose RISC (Reduced Instruction Set Computer) cores [15,16,17,18] or reconfigurable hardware with additional special-purpose coprocessors designed to assist in common network packet processing tasks [19,23,24,26,27]. Network processors also serve to prolong networking hardware lifetimes beyond ASIC solutions since they can be programmed to support alternate protocols or modifications to existing protocols.

Current devices are designed primarily for efficient packet routing and high throughput, and remain somewhat limited in their flexibility and lack native data processing (encryption, compression, etc.) capabilities. Those devices that do provide some security or data processing support provide only limited support for data encryption. Additional processing support must be added in the form of a coprocessor, while no attempt to secure the device itself against unwanted use or reverse engineering is made. In addition, these processors often are based upon an existing microprocessor core that does not include a direct hardware implementation of instructions that are fundamental to the task at hand.

2 Background

As network data rates and processing requirements continue to increase, specialized network processors are being developed as an alternative to the general-purpose processors and Application-Specific Integrated Circuits (ASICs) traditionally found in network data handling systems. These network processors are designed to provide systems with both ASIC-like speeds and microprocessor-like flexibility. Although the presently available commercial devices are primarily designed much like augmented RISC machines, current research is investigating the use of reconfigurable logic to provide additional speed and flexibility to end product designs.

As network devices become more prolific, applications requiring enhanced device security will arise. Military use of network systems for encrypted data traffic might depend on algorithms that are not publicly available. It is desirable for the structure of these systems to remain hidden. In the commercial realm, it is easily within the foreseeable future that applications such as cellular telephones customize themselves on a per-user basis. Reconfigurable processing provides an excellent basis for this, and enhanced device security can serve to protect corporate investment in the intellectual property that goes in to the technology. From a consumer viewpoint, it is also desirable to protect these new devices from unauthorized use. Enhanced security within the device itself provides user specific customization without creating a device that is easily modified for unauthorized use.

2.1 Network Processing

Traditional network data-handling systems (switches, routers, edge devices, etc.) use either ASICs for performance [3,4] or general-purpose processors for flexibility and cost reduction [5,6]. Unfortunately, these devices do not offer the combination of speed and flexibility sought in current network systems [7,8]. ASICs provide high-speed data processing, but do not contain sufficient flexibility to adapt to new protocols and features [7]. Additionally, ASIC development is a costly and time-consuming process, requiring significant numbers of ASIC designers and millions of dollars in foundry fees just to produce a single device [7,8,9]. This level of investment in a device is a limiting factor to innovation, and can cause even the most well established players to hesitate before updating an established technology or pursuing a new direction. General-purpose processors are quite flexible, but do not offer the performance required for some network system tasks [7]. At the current rate of growth for network interface and general purpose processor speeds, this performance gap is continuing to widen [7]. These issues have created a demand for specialized network processors.

To meet this demand, several network processor solutions have been developed. These hardware solutions range from full single-chip network processors to individual special purpose programmable network processing devices, all falling into one of the following general categories: *general-purpose* processors and *focused-function* devices.

A *network processor* is defined here as a re-programmable processing device that is designed specifically for use in a networking environment. It must be able to provide computing services for the continuous data flows that comprise a networking environment.

2.2 Embedded Device Security

Device security becomes a concern as network-attached devices become more ubiquitous and competition among vendors increases. For some applications, such as military communications, it is important to protect the network hardware itself from attempts to reverse engineer or replicate its operation (or from incompetence). From the manufacturer's standpoint, it is also important to protect the intellectual property (IP) investment contained in a network product. Gathering the core system function into a single device limits the number of exposed interfaces that can be monitored to determine system function. Organizations involved in proprietary network data manipulation can take advantage of this by encapsulating all IP functionality within a network processor.

Unfortunately, current network processors are not designed to meaningfully provide this type of security. Solutions based upon conventional ASIC and processor techniques fall short of providing the full protection for the algorithms they contain. ASICs retain their functionality even when the system is not operating. As a result, an ASIC can be opened and studied to determine its internal structure [10]. A conventional processor may not retain information regarding tasks when not in operation, but it is vulnerable to bus monitoring attacks while running [11]. In a processor-based system, the IP investment is in the code fed to the processor at run-time.

A reconfigurable logic device, such as a Field Programmable Gate Array (FPGA), does not need to retain its function while the system is not in operation. When un-powered, the FPGA is a generic reconfigurable device containing no user-specific function. While operating, it can remain a generic piece of hardware running a simple user discovery application until a valid user presents credentials to the system. Once a user is validated by the system, the device can be configured to include functionality tailored to that user. While common FPGA configuration streams are clear text, protection of the information using a standard cryptographic protocol can provide for a measure of security as this data is transferred to the device [12,13].

In both processor-based and FPGA-based systems, program information is often held in static random-access memory (SRAM). In both cases, a secure application must take care to eradicate itself from the RAM, since it may be possible to detect the residual information in SRAM cells after power has been removed [14]. Since many FPGA technologies hold configuration information in SRAM, this means care must be taken when a user is done with the system.

2.3 Background Summary

The current design techniques used to build these network processors do not, however, fully conceal the internal algorithms, nor do they provide support for security of the hardware itself. When the reputation of a network company is based upon its ability to process data more efficiently than the competition, it is important to protect the intellectual property of a design. When that design is critical to the security of the user, it is important that that security not be breached. As these systems find their way into applications in which the algorithms themselves need to be provided the utmost security (military encryption, etc.), it becomes more important to conceal the device function and control user access. Toward this end, a new scheme is needed to maintain the speed and flexibility of network processors while providing an additional level of device security. An ideal device provides an efficient means of run-time reconfiguration to support leading-edge network protocols, adapts as these protocols change, and implements an efficient method for concealing the internal operation and configuration/programming data presented at its ports.

3 Structure

The prototype system described in this chapter distributes the network processor function across several configurable devices to allow for run-time reconfiguration and to provide a modular design structure that promotes independent development and validation as well as access points for integration testing (Figure 3.1). Separate configurable elements are used for system configuration control and user processing. Support for both the initial user specific function and the run-time modification of that function is contained within a single device. A second configurable device contains the interfaces for user hardware and the basic packet handling systems. A third device provides system configuration control and authentication support through during both the user discovery and the user function modes of operation. This division allows for external device (wireless radio-frequency hardware and wired network physical interface) connection establishment during initial user configuration. It also permits run-time updates to the processing channels that do not disturb these external connections.

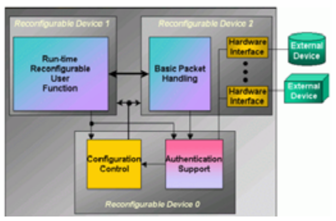


Figure 3.1 Basic System Block Diagram

The use of multiple devices simplifies the initial development and debugging process while allowing for a lower cost prototype. Although the operation has been divided among three components for demonstration, the functionality provided by these three configurable elements could be folded into a single larger reconfigurable device. Additional support for partial run-time reconfiguration must be developed to support single-device packaging, however. Along these lines, configurable logic vendors now provide technologies such as the internal *SelectMAP* [30] interface (*ICAP*) in the Xilinx *Virtex-II* [31] FPGA family. These interfaces allow a device to partially reconfigure itself while it is running. Fong, Harper, and Athanas describe in [43] the current state of ongoing research into this concept of dynamic self-reconfiguration, the results of which will permit the full integration of the prototype.

3.1 Configuration Management

The secure network processor operates in two distinct modes. When a valid user is present, it provides individualized processing of the user data. When no user is present, a discovery application is run on the hardware. This section describes the system operation as it transitions from user discovery mode to user function mode.

3.1.1 Authorization

In this particular prototype system, a user is identified by a token and a biometric. The user inserts a token into a receptacle attached to the prototype platform and then allows the system to identify them via a biometric interface. At the final stage of authorization, information used in assembling the embedded user function is downloaded from the token in the form of a *Bit stream Cache Table*.

This prototype uses a custom-designed *Bit stream Cache Table* to manage the encrypted configuration bitstreams. Each table entry corresponds to an individual bitstream required for current user operation. As shown in Figure 3.2, a table entry contains an identification tag, a decryption key, bitstream memory address ranges, a partial bitstream indicator, and status flags. This format was chosen to allow for modular development of the prototype. As the primary communication point between the authentication system and the configuration engine, it allowed the two systems to be developed independently, and then easily merged into a whole system. The table itself is flexible enough to allow for modification of this interface as needed. Without a valid set of table entries, the encrypted bitstream data cannot be retrieved, decrypted, and loaded into processing elements.

The *Start Address* and *Stop Address* fields indicate where the encrypted bitstream is stored in local memory after retrieval from a remote server (Section 3.1.3). Entries with the *Default* flag set are to be programmed in as the initial user function. Entries that do not have this flag set represent configuration data that may be swapped in while the network processor is in User Function mode.

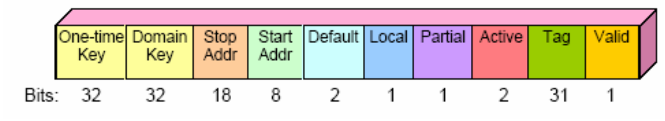


Figure 3.2: *128-bit cache table entry for configuration management*

The *Active* flag is used to mark which configurations are currently being used. After a bitstream is retrieved from the remote server to local memory, it is marked as *Local*. The *Tag* is used to uniquely identify a configuration bitstream and its domain. For *Valid* entries, the previously mentioned fields represent a user application bitstream. Invalid table entries simply represent slots that are not yet filled by the current application. Finally, the *One-time Key* and *Domain Key* fields contain the keys associated with a doubly encrypted configuration entry. The one-time key is determined by reconstruction of the one-time key used by a remote Configuration Server. The domain key is retrieved from the token and used in conjunction with the one-time key to produce a clear-text configuration bitstream. Once these keys are in the table, they are kept inside the secure network processor and used only by the configuration engine (see Figure 3.5).

The Bitstream Cache Table can only be modified by two sources. The first source is the User Discovery application, which transfers the original table data upon user verification. The second source is a *Control Register* in Reconfigurable Device 0 that provides an interface for run-time modification of the system. This register allows the user application to request reconfiguration in addition to providing a debugging interface for the prototype system. Restricting cache table access in this manner allows for internal control of reconfiguration events. Although table access could be restricted further for other applications, the chosen method provides a useful combination of control and accessibility in the prototype system. Support for user discovery mode is provided by a security management system running inside of the Configuration Engine in *Reconfigurable Device 0* and by an additional *Authentication System* running in *Reconfigurable Device 2*. Additional details of user discovery mode operation are provided in [29].

3.1.2 Stream Management

User bitstreams are retrieved from a network connection based upon the tags placed in the Bitstream Cache Table during the user authentication process. Conversion from User Discovery Mode to User Function Mode is outlined in Figure 3.3. It involves retrieving the appropriate encrypted configuration bitstreams for reconfigurable devices *0* and *1* from an external network and placing them in local storage. The bitstreams are then read by a Configuration Engine, decrypted, and used to place the user application in the network processor.

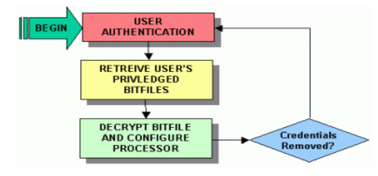

BEGIN
USER AUTHENTICATION
RETREIVE USER'S PRIVLEDGED BITFILES
DECRYPT BITFILE AND CONFIGURE PROCESSOR
Credentials Removed?

Figure 3.3: *Conversion from user discovery to user function mode*

The security management system employed during this operation is shown in Figure 3.4. It consists of a device programmer, a user authentication unit (described in [29]), a network interface, and a configuration manager. The *Device Interface* shown in the picture is a simple register-based connection that directly controls the configuration pins of *Reconfigurable Device 1* and *Reconfigurable Device 2* [32]. Bitstream data is directly written to this interface in unencrypted form to configure the reconfigurable devices that comprise the system. The *Ethernet Interface* may be either a direct (local wired crossover) connection to a display device such as a tablet computer, or a standard connection to a full network system. In the case of a direct connection, the Configuration Store is the connected device.

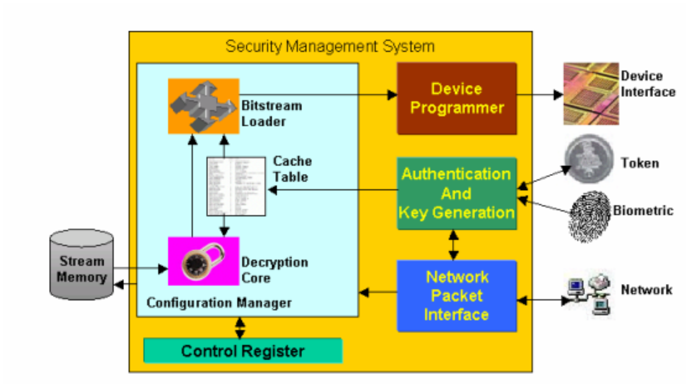


Figure 3.4: *Configuration management*

The *Configuration Manager* is comprised of three major entities: a Decryption Core, a Cache Table, and a Bitstream Loader. In this implementation, The *Bitstream Loader* consults the cache table to locate encrypted configuration bitstreams in memory and pass them through the decryption unit to the Device Programmer. The *Decryption Core* decodes the encrypted bitstream using the associated key (provided by the cache table) and returns the result to the Bitstream Loader for forwarding to the Device Programmer.

3.1.3 Configuration Stream Requests

Configuring *Reconfigurable Device 1* and *Reconfigurable Device 2* as a user-specific network processing system changes the platform from a stand-by state to an operational device. As described above, this process is based on the bitstream cache table and handled by the Configuration Manager. When the bitstream cache table is first transferred to the network processor, the Configuration Manager determines if the required encrypted bitstreams are stored locally. For non-local bitstreams, a request is sent over the network interface to a remote Configuration Store to retrieve the needed information as depicted in Figure 3.5.

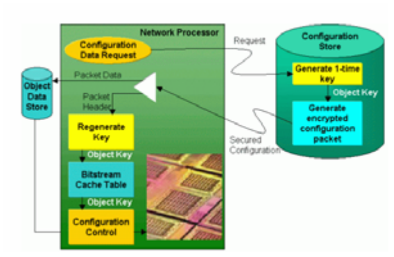


Figure 3.5: *Configuration key handling*

The transmitted request used in this example structure is in clear text and the assumption is made that it does not need to be secured. The request contains only a bitstream identifier. As such, any outside party intercepting it would gain information regarding the bitstream identifiers requested by a particular device. While they would not gain any knowledge of the content of the configuration data, this might provide some advantage in attempting to decode the returned data. In a non-prototype system, this request could be protected using the same scheme that is used here to encrypt returned configuration data. This additional protection of the request would conceal the identifiers, but would also require additional resources. The marginal gain in security was not considered worthwhile for the prototype system, as it does not demonstrate anything outside of the already established protection scheme.

In response to a request, the network Configuration Store returns a packet containing an encrypted bitstream. This bitstream is secured in a packet conforming to the format. Upon receipt of this packet, the network processing system decrypts the packet header and uses a combiner located in *Reconfigurable Device 0* to produce the decryption key for the bitstream. The encrypted bitstream is moved into external storage at the location specified by the key table, while the internal key table is updated with the configuration data key and the bitstream is marked as local.

When all configuration streams are local, the bitstream loader reads those marked as *Default* from local storage, retrieves the one-time key from the Bitstream Cache Table and the domain key from the token, and uses a hardware encryption unit to decrypt the configuration information and forward the results to the appropriate programming interface.

3.1.4 User-based Reconfiguration

Once the initial configuration is complete, the user-specific system may itself request device reconfiguration. Reconfiguration requests of this type may result from user input or data flowing through the system. The process by which these requests are handled is similar to that of the initial reconfiguration. The bitstream loader is notified by the user application that a configuration update is needed. The update may consist of a bitstream placed in memory by the user application or a tag for a bitstream that must be retrieved from a remote server. In the first case, only the Bitstream Cache Table is updated to reflect the location of the new bitstream and its decryption key. In the later case, the bitstream is retrieved as described above for the initial configuration. In both cases, once the bitstream is local and the table is updated, the bitstream loader decrypts the configuration data and feeds it to the appropriate device configuration interface.

The Bitstream Cache Table keeps track of recently used bitstreams using identification tags to refer to bitstream entries (see Figure 3.2). Bitstream entries may reside anywhere in the table and the table may be extended as needed with the only theoretical limitation being the number of unique identifiers it can contain (231 entries using the current table format). In practice, table access may be restricted to initial configuration only or extended to the user or application for additional device flexibility.

3.2 Structure Summary

It provides the flexibility needed to test user applications within the network processor framework and allows for the run-time system modifications that are the basis of the framework. It is not, however, an ideal representation of the network processor. Several steps were taken to produce a realizable system in currently available (and relatively inexpensive) hardware including separation of the functionality across several programmable devices. While all of these are reasonable goals in a production system, and certainly worthy of a second prototype, they do not add significantly to the preliminary investigative utility provided by the described prototype platform.

4 Implementation

The prototype network processor system has been implemented on a SLAAC1-V reconfigurable computing platform [33]. The SLAAC1-V, as shown in Figure 4.1, has three Xilinx *Virtex XCV1000* FPGAs [25] labeled *X0*, *X1*, and *X2*. There are ten independent on-board SRAM memories providing a total capacity of 11.5Mbytes, and a significant amount of inter-FPGA connectivity. In the default SLAAC1-V system, the Configuration Control FPGA (a Xilinx *Virtex XCV300* [25] device) governs device configuration.

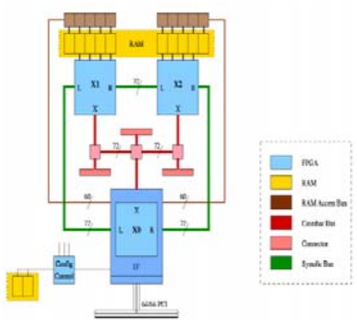


Figure 4.1: *SLAAC1-V block diagram* [29]

The network processor functionality is divided among the primary FPGAs as defined by the structure described in Section 3.1. Given that the target operations of the prototype all involve data movement between wired and wireless networks, this division is made as shown in Figure 4.2. Here, the reconfigurable device *X0* contains the primary system control function, *X1* contains user-based processing functions, and *X2* contains device interfaces along with associated data processing.

All permanent configuration and control functions are embedded in device *X0* since this device has direct access to the Configuration Control FPGA (Figure 4.1). In the prototype system, however, the Configuration Control FPGA consists of nothing more than a data path to the configuration interfaces of the three primary FPGAs. The contents of the *X0* device remain unchanged as users arrive (authenticate) and depart (deactivate). Since it is controlling the configuration operation, the SLAAC1-V design does not allow it to be configured itself.

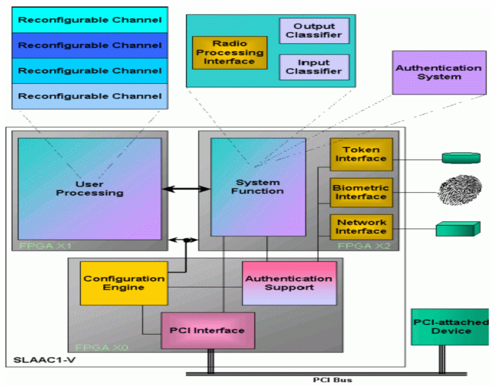


Figure 4.2: *Network processor embedding*

If no valid user is present, device *X2* contains only the User Discovery System and authorization device interfaces. After a valid user arrives, *X2* contains the primary user data routing functions as well as connections to user devices. Add-on modules have been developed to provide this FPGA with physical connections to support hardware, including wired and wireless network, user identification token, and biometric identification device interfaces [29]. These interfaces also perform any packet handling (encapsulation, header processing, error checking) issues associated with an attached device, allowing the data to interact with the user processing system. Additional support for PCI-attached devices is routed through *X0* since it is physically connected to the PCI bus of the prototyping card. This transceiver interacts with the PCI core in *X0*, while user-configurable data processing for the device takes place in *X2*. Device *X1* is empty when no valid user is present. It is configured to contain the Reconfigurable Processing Channels of the network processor user function. When a valid user is detected. This division of functionality allows for run-time reconfiguration of the Reconfigurable Processing Channels in *X1* without disturbing the hardware interfaces contained in *X2*.

4.1 Platform Reconfiguration

Due to limitations in the token processing power, however, the configuration process does deviate somewhat from the sequence outlined in Figure 3.3. Rather than transferring encrypted configuration stream header information to the token for key assembly, the prototype system transfers the user credentials via a secured link from the token to the network processor platform. The key assembly mechanism is located in FGPA *X0* of the prototype. This modification creates a slightly less secure system, as credentials are exposed to the platform, yet was necessary due to limitations in the iButton token processing capabilities. The FPGA contains more processing resources than the selected token device and provides design access to ease algorithm development. As technology progresses and configuration stream key assembly algorithms are more thoroughly investigated, the assembly function may be moved back into the token.

A user fingerprint template is also downloaded from the token to the platform. This template is forwarded to a Secugen [34] fingerprint reader where a match routine authenticates a user. The result of the match is returned to the network processor as a pass/fail indication. The template is a proprietary format developed by Secugen and cannot be used to reconstruct the fingerprint from which it was derived. The FDA01 device, however, can use it to verify a user using internal hardware to match a current user to the template. It is impractical to perform this algorithm in the FPGA due to its proprietary, undisclosed, nature. In a full implementation of the secure network

processor, however, the hardware to perform this match function should be moved into a token similar to [28].
Upon user verification encrypted configuration data is read from a Configuration Store device. The basis of the encryption used in this system is the Blowfish algorithm [35]. Blowfish was chosen because it supports a flexible key size and maps well to hardware [36]. In addition, the open-source algorithm is similar to other symmetric cryptographic algorithms and is credible in the security industry [37]. Blowfish supports key sizes ranging from 32 bits to 448 bits, although the prototype restricts itself to 32-bit keys to reduce the size and complexity of the embedded hardware. The hardware implementation of blowfish used in the prototype encrypts or decrypts up to 35.5 Mbits/sec (using a 20 MHz clock), regardless of key length. This allows for an entire *X1* or *X2* configuration (approximately 6 Mbits in length) to be decrypted in 170 ms. Once configured for user operation, the user functions of the current prototype embed wired and wireless network interfaces into the user processing elements. An arbiter combines output data from the four processing channels, preserving packet boundaries, and forwards the data to an output device.

4.2 Wireless Network Data Interface

Data destined for the transmission from the wireless interface is packetized within its processing channel before being presented to an arbiter for multiplexing onto the wireless output channel. This encapsulation of data into a wireless packet is performed by the *packetizer*. This entity adds an appropriate wireless protocol header to outgoing data and calculates an attached CRC if it is needed. The wireless packet format is shown in Figure 4.3.

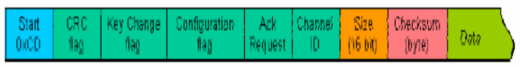


Figure 4.3: *Wireless packet header*

A *Start* marker in the wireless packet header allows for receiver synchronization and a *Checksum* [38] is provided to verify both header integrity and the validity of the Start marker. A control byte, identical to that used on the wired network input is included to provide packet content and routing information. Also in keeping with the wired network input format, the *Size* word is presented LSB-first. If a CRC is present, it is appended to the data and is calculated over the full packet beginning with the Start marker.
This wireless packet format was constructed to minimize header overhead while allowing for lossy connections between wireless stations. As indicated above, the wireless interface for the first prototype was implemented using a commercial wireless transceiver.
This device provides internal data integrity checking for its own smaller communication packets. Packets seen by the prototype at the wireless interface consist of several of these smaller transceiver packets. Data errors at the wireless interface therefore consist of either a lost portion of a packet or reception of data beginning in the middle of a packet. Lost portions are readily identified by the CRC, while the checksum provides some assurance that a detected packet start marker is indeed the start of a packet. The checksum and CRC employed are the same as those used in the Ethernet protocol, and should provide similar data integrity assurance. The chosen Ethernet techniques were primarily selected for ease of hardware implementation, and have been carried over to the second prototype in which a user-configurable wireless system is used [39]. The data integrity checks provided in the current system may very well be replaced as the wireless interface continues to evolve, and new data transfer protocols are developed.
Data received by the wireless transceiver is classified for channel processing by the *classifier*. This entity routes data to the correct processing channel, performs any needed CRC check, and signals an acknowledgement unit when the sender requests an acknowledgement of packet receipt. In this case, the classifier also acts to detect valid packet starting points by searching for a packet start marker and verifying the header checksum. Wireless packet information is reduced to the control byte as data is forwarded to a channel for processing. Data flowing from the wireless to the wired interface is placed into outgoing FIFOs after channel-specific processing is complete. A separate Control FIFO is used to indicate the availability of data in these outgoing data FIFOs. This data is retrieved by an outbound Ethernet interface for placement on the wired network.

4.3 Wired Network Data Interface

A header is required for all data entering the wired network side of the processor, as shown in Figure 4.4. This header allows for classification and processing of packetized data before it is sent to the wireless interface for transmission. The *Control Byte* determines the processing channel to which the packet will be routed as well as providing some packet processing control information. The *Size* fields specify the size of the data set in bytes, where the size is a 16-bit value provided Least Significant Byte (LSB) first. A network protocol-specific module translates from the external network packet format to this internal format.

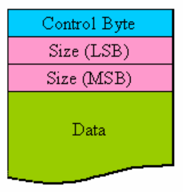
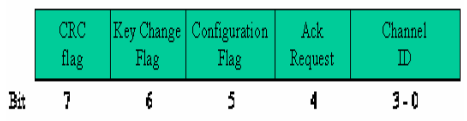


Figure 4.4: *Wired network input packet*

The *user_classifier* parses the control byte (Figure 4.5) to determine packet destination. An optional 16-bit Cyclic Redundancy Check (CRCCCITT [40]) may be attached to the data for error checking. If that is the case, the *CRC* flag is set in the Control Byte, the Size field is expanded to include the CRC, and the user_classifier performs a CRC check over the data set as it is read in. The result of the CRC check is passed to the appropriate channel following the data. The *Key Change* flag in the Control Byte is set to indicate the fact that the data in the packet is a new key to be used for following transmissions on the channel. The *Configuration* flag indicates the packet data is a new configuration for the indicated processing channel. An *Ack Request* field allows the sender to request an acknowledgement of packet receipt from the recipient. If an acknowledgement is requested, the user classifier signals a separate acknowledgement unit to reply. Finally, the *Channel ID* field provides a mask for routing this packet, where each bit is used to enable a processing channel. Note that data may be broadcast to more than one channel by setting multiple bits of the Channel ID.



	CRC flag	Key Change Flag	Configuration Flag	Ack Request	Channel ID
Bit	7	6	5	4	3-0

Figure 4.5: *Incoming data control byte*

4.4 Secured Network Gateway

Making use of alternate hardware for the network and radio connections, the new system increases potential throughput. In addition, the user processing space of *X1* is used for reconfigurable processing channels that are used to encrypt and decrypt data traffic for the wireless network. The user function provided by this prototype is a Secured Network Gateway. A Secured Network Gateway is a device providing a link between two network systems that is adapted to a particular user and active only in that user's presence. Applications for this type of device include any high-security installations with a private network that have a controlled external access point. This type of installation might include the radio room of a naval vessel or the communications center of a military installation. While many users may not need this level of security, the adaptable nature of the system does provide some benefit even the security features are not essential. For example, simply indicating the presence of a particular user may modify the features of a gateway. Real benefit for all users can be seen in the adaptable nature of the system as it is deployed in devices like cellular telephones where the interface between voice and radio may be adapted to suit the services used by an individual. In this situation, run-time adaptability of the system also comes in to play as the user moves from one location to another, potentially changing radio communication protocols along the way. It requires the presence of a valid user to provide a link between a wired network and a wireless system. A block diagram of the system is shown in Figure 4.6.

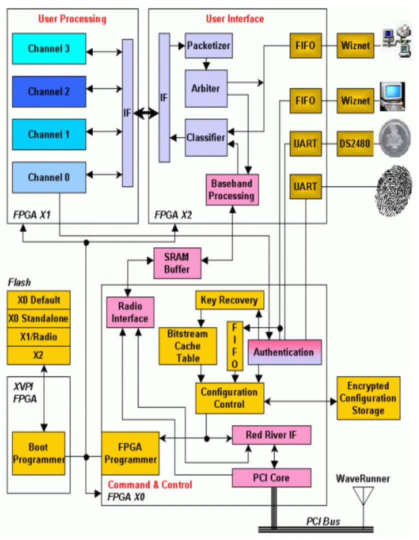


Figure 4.6: *Secure network gateway prototype*

4.5 Network Gateway Structure

Configuration of the gateway is done in a secure manner using a remote Configuration Store. Once a user is authenticated, the user requirements and credentials are transferred from the token to the network processor. The network processor then initiates a configuration request sequence to retrieve user settings for the *X1* and *X2* FPGAs as well as the connected radio hardware. These requests and the return data are handled with the exception that the information contained within the Configuration Store is not pre-encrypted and does not require a domain configuration key for decoding. While the addition of this security level could easily be added, it is more convenient to deal with unencrypted bitstreams in the prototype environment. The encrypted user configurations stored in local SRAM are therefore encrypted only with the one-time key used for the transfer. A local security agent processes the attached header data to create keys for each configuration and writes the results to the Bitstream Cache Table. The Configuration Manager then parses the table and configures attached devices as needed.

The wireless network still operates on a serialized FIFObased interface with an arbiter feeding data to it and a classifier reading from it. The wired network still feeds data to a classifier, but it now uses an output arbitration unit to feed multi-channel data through a single path. A second wired connection is included as well to allow for a directly connected user device (PDA, laptop PC) for control and monitoring in addition to a local area network (LAN) connection for data flow. Wiznet IIM7010 Ethernet modules [41,42] are used for the wired network interfaces. These modules read and write data using a FIFO interface and provide raw Ethernet, UDP/IP (User Datagram Protocol over IP), or TCP/IP network protocol handling. For this prototype, data exchange with the LAN is done at the Ethernet level, while data is exchanged with the local user device using TCP/IP via a crossover cable connection. The Configuration Store is contained within the user device to simplify testing and demonstration.

5 Conclusion

Several approaches to the design of a network processor currently exist, each of which provides some means of efficiently handling high-speed network traffic and adapting to new protocols. As these devices find their way into applications in which the very algorithms need to be protected, it becomes more important to conceal the device function, control user access, and otherwise limit the possibility of interacting with, participating in, or interfering with the operation of a privileged communications system. Most design techniques currently used to build these network processors do not, however, attempt to conceal the internal algorithms or to provide support for security of the hardware itself. Those that do provide some security support typically add hardware co-processing for common security functions (RSA, SHA, etc.) or provide internal storage for program streams. When the reputation of a network company is based upon its ability to process data more efficiently than the competition, it is important to protect the intellectual property of a design. When that design is critical to the security of the user, it is important that device security not be breached.

Fixed-function ASIC devices are inherently insecure, as they not only require external devices for production of a useful network processing solution, but their hardware nature leaves them susceptible to offline reverse engineering. Processor-based systems have the potential for slightly increased security since more functionality might be contained in a single device. Unfortunately, their reliance on external instruction streams means that they are susceptible to instruction bus monitoring attacks while operational. While offline observation of the devices themselves might reveal very little about these processor-based systems, attacks that determine the contents of instruction storage are particularly effective. Devices like the Intel IXP 2850 reduce the effectiveness of operational bus monitoring for applications that can fit in the internal instruction storage space. They do, however, still remain vulnerable to offline analysis of this instruction store. Finally, reconfigurable systems like the USC/ISI GRIP are particularly vulnerable to IP theft, as they provide the hardware functionality in the form of an unsecured configuration stream that could easily be used in other devices to provide the same function. These systems could easily increase security somewhat by using current vendor bitstream encryption, or more fully by adapting the interface structure.

The flexible hardware basis for the secure network processor permits the construction of systems fully self-contained within a single device. It is also a consideration when attempting to minimize components in a network processing design. The least self-contained systems, such as the IDT PAXport 2500 classifiers are designed to simply provide a portion of a larger solution. Devices based upon processing cores provide more self-contained solutions, as a wider variety of system function can be performed in a single component. Many of these devices do not, however, contain basic network interfaces and still require off-chip storage of instruction streams during system operation. The on-chip storage and MAC support of the IBM PowerNP2G allows for a somewhat more self-contained solution. Reconfigurable systems provide the most potential for self-contained systems. The Chameleon CS2112 does not require external instruction data, and provides a reconfigurable processing base that can implement a variety of network functions. In a system requiring user authentication, it may fall short; however, as no internal security functions are included. Finally, reconfigurable systems like the USC/SI GRIP are designed using several components, and are inherently multi-device solutions. It would certainly be possible, however, to incorporate the function that they provide into a single system like the secure network processor to make them a truly single-device solution. Lastly, the dynamic nature of this system allows for an increased level of dynamic user adaptability. Clearly, reconfigurable solutions like the USC/ISI GRIP could be as adaptable as the secure network processor. Their current lack of user identification and authorization means that in their current incarnations they are not inherently suited to user adaptation. Some additional support hardware and minor internal modification could remedy this, however. General-purpose processing solutions are likewise not currently user-adaptable. They to could benefit from additional support in this area, but remain somewhat limited in their potential by the restriction to a fixed instruction set. Finally, ASIC devices designed for a specific purpose like the Agere APP750NP may provide some flexibility within their intended function, are clearly not adaptable to provide other functions. As reconfigurable hardware technology continues to improve, the size of standard reconfigurable devices will continue to grow. The space available within current reconfigurable systems allows for custom hardware and standard processor cores to be mixed together on a single device. Newer technology will allow even more functionality within a device, making single-chip solutions even more practical.

6 Bibliography

[1] Thompson, J.F., J. Bernstein, and J. Crane, "*Identity Theft*," Presentation to the President's Information Technology Advisory Committee (PITAC) 12th meeting, February 7, 2001.

[2] Evers, J., "Dutch police fight cell phone theft with SMS bombs," IDG News Service \Amsterdam Bureau, March 27, 2001.

[3] Juniper Networks, *T-Series Routing Platform Data Sheet*, Part Number 100051-0006, Juniper Networks Inc., Sunnyvale, CA, February 2003.

[4] Austrian Aerospace, "*Spacewire Router ASIC Development*," ESM-006, Presentation to the European Space Agency (ESA) Spacewire Working Group, September 18, 2001.

[5] May, M.D., P.W. Thompson, and P.H. Welch, ed., *Networks, Routers and Transputers: Function, Performance and Applications*, IOS Press, Headington Burke, VA 1993.

[6] Smitt, E. L. and R.J. Collins, "Microprocessor based control and switching device," United States Patent no. 4,685,124, assigned to Data General Corporation, Westboro, MA, April 30, 1985.

[7] Lawson, S., "Network processors enter new generation," IDG News Service, *Network World Fusion*, June 19, 2002

[8] Husak, D., "Programmable NPUs 'edge' out ASICs," *EE Times*, November 18, 2002.

[9] Foremski, T., "Network chips: battleground for the big boys," *Financial Times Survey*, FT.com, April 15, 2002.

[10] Blythe, S., B. Fraboni, S. Lall, H. Ahmed, and U. de Riu, "Layout reconstruction of complex silicon chips," *IEEE Journal of Solid-State Circuits*, volume 28, pp.138 – 145, February 1993.

[11] Huang, A., "Keeping Secrets in Hardware, the Microsoft XBoxTM Case Study," *CHES 2002*, August 2002.

[12] Erickson, C.R., D. Tavana, and V. A. Holen, "Encryption of Configuration Stream," United States Patent no. 6,212,639 B1, assigned to Xilinx Inc., San Jose, CA, April 3, 2001.

[13] Batinic, I., L. Kraus and M. P. Loranger, "Field Programmable Gate Array With Program Encryption," United States Patent no. 6,351,814 B1, assigned to Credence Systems Corporation, Fremont, CA, February 26, 2002.

[14] Anderson, R. and M. Kuhn, "Tamper Resistance – a Cautionary Note," Proceedings of *the Second USENIX Workshop on Electronic Commerce*, Oakland, CA, November 18-21, 1996, pp 1-11.

[15] Intel Corp., *Intel IXP1250 Network Processor Data Sheet*, Intel Corporation, Part number 278371-006, December 2001.

[16] Chandra, V., "Selecting a network processor architecture," IBM Microelectronics Technology Group, August 2002.

[17] C-port Corp., *C-5 Network Processor Architecture Guide*, C-port technical library C5NPD0-AG/D, C-port Corporation, North Andover, MA, May 2001.

[18] Vitesse Semiconductor, *IQ2200 Product Brief*, Vittesse Semiconductor Corp, Camarillo, CA, 2002.

[19] Wirbel, L., "Network processors take reconfigurable approach," *EE Times*, May 22, 2000.

[20] Agere Systems, *Smart Processing for Terabit Networks, PayloadPlus Processor Family Overview*, Agere Systems Incorporated, Allentown, PA, 1999.

[21] Agere Systems, *10G Network Processor Chip Set (APP750NP and APP750TM) Product Brief*, Agere Systems Incorporated, Allentown, PA, November 2002.

[22] PMC-Sierra, *RM9000x2 Integrated Multiprocessor Data Sheet (preliminary)*, PMCSierra

Incorporated, Burnaby, B.C., 2001.
[23] Solidum Systems, *PAX.port 2500 Product Brochure*, IDT Canada Inc., Ottawa, Ontario, 2002.
[24] Bellows, P., V. Bhaskaran, J. Flidr, T. Lehman, B. Schott, K. Underwood, "GRIP: A Reconfigurable Architecture for Host-Based Gigabit-Rate Packet Processing," *IEEE Symposium on Field-Programmable Custom Computing Machine (FCCM) '02*, Napa, CA, April 2002, pp. 121-130.
[25] Xilinx, Inc., *Virtex Data Sheet*, Xilinx Inc., San Jose, CA, February 2000.
[26] Lockwood, J.W., "An Open Platform for Development of Network Processing Modules in Reprogrammable Hardware," *IEC DesignCon '01*, Santa Clara, CA, January 2001, paper WB-19.
[27] Brebner, G., "Single-chip Gigabit Mixed-version IP Router on Virtex-II Pro," *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)'02*, Napa, CA, April 2002, pp. 35-44.
[28] Hansen, S. "Rugged, Biometric "Match-On-Device" Token In R&D at Datakey Electronics, Inc.," http://www.datakeyelectronics.com/PR6.htm, Datakey Electronics, Inc., Burnsville, MN, August 14, 2002, last accessed April 3, 2003.
[29] Abraham, A., *It is I, An Authentication System for a Reconfigurable Radio*, M.S. thesis, Virginia Polytechnic Institute and State University, August 2, 2002.
[30] Xilinx Inc., *Virtex Series Configuration Architecture User Guide*, Xilinx, Inc, San Jose, CA, February, 2000.
[31] Xilinx Inc., *Virtex-II Handbook*, Xilinx, Inc., San Jose, CA, November 2002.
[32] Sundararajan, P., and S. A. Guccione, "XVPI: A Portable Hardware / Software Interface for Virtex," *Reconfigurable Technology: FPGAs for Computing and Applications II, Proc. SPIE 4212*, Bellingham, WA, November 2000, pp. 90-95.
[33] ISI/East, *SLAAC1-V VHDL Users Guide, Release 0.1.3*, University of Southern California Information Sciences Institute/East, 2001.
[34] SecuGen Corporation, *SecuGen FDA01 Developer's Guide, DC1-0001B Rev A*, SecuGen Corporation, Milpitas, CA, April 24, 2001.
[35] Schneier, B., "*Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)*," Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp. 191-204.
[36] Landaker, W.J., "Free Blowfish VHDL Code," SourceForge, http://sourceforge.net/projects/blowfishvhdl/, registered October 18, 2000, development status 4-Beta, last accessed April 3, 2003.
[37] Counterpane Labs, "Counterpane Labs: Blowfish," http://www.counterpane.com/blowfish.html, last accessed April 3, 2003.
[38] RFC 760, "DOD Standard Internet Protocol," prepared by Information Sciences Institute, University of Southern California, January 1980, p. 14.
[39] Red River Engineering, *WaveRunner PMC Model 301 Datasheet*, Red River Engineering, Richardson, TX.
[40] Tanenbaum, A. S., *Computer Networks, 2nd Edition*, Prentice Hall, Englewood Cliffs, NJ, 1988, p. 212.
[41] Wiznet Inc., *IIM7010 Product Description*, Wiznet Incorporated, Kangnam-ku, Seoul, Korea.
[42] Wiznet Inc., *i2Chip W3100A Technical Datasheet v1.3*, Wiznet, Incorporated, Kangnam-ku, Seoul, Korea.
[43] Fong, R.J., S.J. Harper, and P.M. Athanas, "A Versatile Framework for FPGA Field Updates: An Application of Partial Self-Reconfiguration," *Proceedings of the 14th IEEE International Workshop on Rapid System Prototyping*, San Diego, CA, June 2003.