Hardware Implementation of Session Initiation Protocol Servers and Clients

Raymond Peterkin, Mohamed Abou-Gabal, Fadi El-Hassan and Dan Ionescu
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada
{peterkin,elhassan,ionescu}@site.uottawa.ca

Abstract—This paper presents a reconfigurable architecture for the session initiation protocol (SIP). SIP is a protocol used primarily to establish point-to-point sessions between users for multimedia communication. Furthermore it increases the performance of Internet applications and overall efficiency. However, most existing SIP solutions are entirely software based. SIP performance can be enhanced by executing core tasks in hardware while allowing other tasks to be executed in software to guard against performance degradation. This paper proposes an architecture of SIP implementable on reconfigurable platforms (FPGAs) for increased performance and efficiency.

Index Terms—SIP, Reconfigurable, Hardware, FPGA, Embedded

I. INTRODUCTION

The Internet combined with the proliferation of wireless protocols (Wi-Fi) provide a widespread communications platform increasingly accessible through embedded devices like cell phones and PDAs. However the processing capabilities of embedded devices are normally insufficient for high performance multimedia communications. If multimedia protocols were implemented directly into hardware, then embedded devices could be used for audio and video communications with the Internet instead of proprietary networks. The session initiation protocol (SIP) [1] is a protocol widely used to establish sessions for applications like VoIP and video conferencing. Therefore a hardware implementation of SIP would be required for a meaningful and widely accepted mechanism to enable multimedia communications through embedded devices.

Various SIP implementations exist today. However, almost all commercial implementations of SIP are software based, the consequence of which is a significant cost with respect to performance. Open source SIP implementations like oSIP [2] are also entirely implemented in software. Lower level protocols can take in the order of several milliseconds to correctly process information. SIP requires significantly more processing time because it is an ASCII-based protocol. That kind of delay can lead to substantial performance degradation and lost data in high speed networks. To minimize those phenomena, hardware implementations have been researched.

An embedded software implementation for VoIP is presented in [3]. Signaling control functions of SIP are integrated in the proposed embedded system platform. Additional features like voice conferences, call waiting and call on-hold can be built into the system based through SIP but no facilities exist to implement any functionality in hardware.

A microcontroller based phone is described in [4] with SIP based signaling. The microcontroller includes a real-time operating system to optimize the execution of software but provided no facilities to implement any functionality directly in hardware. Similar work was performed in [5] with the development of a VoIP gateway name the Enhanced Multi Media Adaptor (EMMA). EMMA was implemented as a system on a programmable chip (SOPC) with an Altera FPGA. With this implementation, numerous hardware blocks were used to directly implement accelerated voice packetization. However with both projects session establishment was provided through a multi threaded, real-time operating system implementation of SIP.

The work in [6] describes a reconfigurable hardware implementation of SIP. It describes a hardware processor that parses SIP headers and includes checksums for IP and TCP packets. However the processor lacks the functionality to interpret SIP content. No description of an interface is presented to allow for the processor’s integration in a software environment. Therefore, the processor cannot be used directly to implement SIP for robust multimedia applications

The absence of a robust hardware implementation of SIP in the literature and commercial products presents an opportunity to improve multimedia communications through a unique product. Consequently this work is partially described in [7], a recently submitted patent illustrating a comprehensive reconfigurable multimedia system.

The paper is organized into the following sections. The first section provides an overview of SIP and explains its basic characteristics including information exchanged to establish sessions and extensions for other functionality used in high level communications. Then an overview of the architecture is given with great emphasis on the specific components and their tasks. Results achieved to date are then summarized and a conclusion to the paper is provided in the end.
II. SIP OVERVIEW

This section describes the SIP functionality implemented in the architecture described later in this paper. SIP is an application level protocol used to invite users to participate in multimedia sessions. It is similar to HTTP in its request-response model and its ASCII-based packets. Each line in the packet header contains a label (To, From, etc.) and its corresponding information separated by a colon and space. Every line is terminated by a carriage return and line feed. The packet header is terminated by an additional carriage return and line feed before the payload begins.

SIP communications are initiated through end systems (PCs and embedded devices) but messages pass through intermediate machines like proxy servers. These servers are used to perform functions like authentication, authorization and call routing. Users may also announce their current location through SIP registration, which is largely performed through proxy participation.

As this architecture is intended to allow SIP functionality to be performed efficiently on embedded devices, only end system functionality is considered for implementation. Three categories of functionality necessary for comprehensive multimedia communications on end systems are session management, presence and instant messaging. Therefore, the following subsections describe how each of these functions are performed with SIP. Descriptions include relevant message exchanges and SIP packet formats.

A. Session Management

Fig. 1 illustrates the operation of SIP in establishing a multimedia session. One participant transmits an INVITE packet to another participant. Session information is described in the payload of the INVITE message using the Session Description Protocol (SDP) [8]. If the second participant wishes to accept the request a session a 200 OK message is sent back to the first participant. The first participant responds with an ACK (acknowledgment) message and the session is now established. Either participant may terminate the session by sending a BYE packet that must be acknowledged with a 200 OK packet.

Fig. 2 provides an example of an INVITE packet with an SDP payload. A 200 OK response to an INVITE contains the same SDP payload. The ACK, BYE and other 200 OK packets do not contain payloads.

B. Presence

Presence information describes the status of a participant while communicating with other clients. It is often useful to exchange this information so a user can be reassured that sent data will likely be received in a timely manner. SIP provides the facility to exchange presence information among participants [9]. Fig. 3 illustrates how presence information is exchanged with SIP. A SUBSCRIBE packet is transmitted by an individual wanting to know the presence information of another SIP user. If the SUBSCRIBE request is accepted a 200 OK message is transmitted back to the user. Every time the presence is updated a NOTIFY message is transmitted to all individuals who successfully sent SUBSCRIBE messages. All NOTIFY messages must be acknowledged with a 200 OK message.

Fig. 4 provides an example of a SUBSCRIBE packet used...
to initiate session notification. No payloads are transmitted with SUBSCRIBE packets, however NOTIFY packets contain payloads describing presence information. Presence is described with Presence Information Data Format [10], an XML based standard.

C. Instant Messaging

The ability to exchange instant messages (IM) is available with SIP through an extension [11]. Fig. 5 demonstrates how instant messages are exchanged with SIP using MESSAGE packets. Every MESSAGE packet must be acknowledged with a 200 OK message. Participants may exchange IMs at any time with MESSAGE commands in the absence of a dialog.

An example of a MESSAGE command is shown in Fig. 6. The payload is unformatted text directly interpreted as an IM. No payload is included with a 200 OK response.

III. HARDWARE ARCHITECTURE

A high level architecture of an embedded system for multimedia communications is described in Fig. 7. The software component of the system is used to communicate with a SIP client for exchanging information regarding configuration and feedback. Sockets are used by the embedded system to establish connections with remote devices and exchange data through the Internet.

The hardware component of the embedded system is used for performing the majority of processing and generation with respect to SIP packets. Data and status registers comprise the interface of the hardware component. Information is read from both status and data registers while data is written to the embedded system through data registers alone. The functionality of the hardware component is divided into three modules. The SIP and XML engines perform the functionality for SIP and PIDF respectively to generate and
process the appropriate information. Both engines share a common interface so data can be successfully transferred to and from software. This interface is implemented as a separate module and it is described as the XML-SIP Interface (XSI). The XSI manipulates the data and status registers used to communicate with the software component. Separate memory components are used to store SIP, IM and SDP data and they are manipulated directly by the XSI and SIP engine. The remainder of this section will describe elements of the hardware component used to implement SIP functionality, including a detailed description of the SIP engine.

Fig. 8 describes the high level architecture of the XSI. It is divided into a control unit and data path responsible for high level logic and holding data respectively. The control unit is a collection of finite state machines containing all the logic of the XSI. The interface is used to interact directly with software by receiving and interpreting high level commands. Those commands are translated into lower level equivalents and sent to a command module. The command module acts as the main coordination module of all functionality. It is used to generate status values after every operation is complete and concurrently interact with all other modules to ensure that commands are properly executed. A parser module is implemented as a separate component of the control unit and it is used for verifying data before it is sent to the SIP and XML engines. Certain errors may be detected by the parser before data is sent to other modules, saving processing time.

The data path is divided into three major components. Data and status registers are stored in the Registers component of the data path. When data is received from software it is directed to the Packet Data component and subsequently to the Parser Data component if necessary.

The command module of the XSI is divided into several modules as shown in Fig. 9. The packet processor is used to manipulate the XSI parser module, SIP engine and XML engine so a SIP packet can be processed as efficiently as possible. The SIP presence reader is used to read a PIDF document from the XML engine while the SIP presence configuration module is used to write configuration data into the XML engine.

The architecture of the SIP Engine is shown in Fig. 10. The XSI interacts with the SIP engine by manipulating the Interface, which is a finite state machine coordinating all other modules in the SIP Engine. The interface enables the SIP Packet Generator and Parser to perform those high level functions in addition to manipulating the SIP Data Module to store configuration information.
All the logic to generate SIP packets with the correct sequence of data in all scenarios is implemented in the SIP Packet Generator. When a command to generate a SIP packet is received from the XSI, the Interface enables the SIP Generator to create and store the requested packet in SIP memory. The SIP packets supported by this implementation are ACK, BYE, INVITE, MESSAGE, NOTIFY, OK and SUBSCRIBE to provide functionality for session establishment, presence and instant messaging. The SIP Packet Generator is a finite state machine optimized for all the supported packets given the similarities among the different types. The SIP Data module is controlled to generate all the packet data and the Transmitted Data module is controlled to save specific data required by the SIP Parser if a response packet is received.

The SIP Data Module, illustrated in Fig. 11, contains all the information required to generate SIP packets and the means to search for information when packets are parsed. Numerous databases are implemented to provide the ability to store and search many types of data including packet labels (To, From, Via, etc.), users and packet types. Fig. 12 illustrates the databases implemented with the Data module using a 200 OK packet to demonstrate the information that can be generated or searched. The Header Type database is used to identify and generate the type of SIP packet (INVITE, MESSAGE, NOTIFY, etc.). SIP user identifiers are held in the user database, which is used primarily with information in the 'To' and 'From' fields. The command sequence (cseq) database contains information to determine out-of-sequence requests while the type of payload data is determined through the Content Type database. Every field name is held in the field database. The integer-to-ASCII converter is used to convert integer data from various databases into an equivalent ASCII string when SIP packets are being generated. No converter is required when SIP data is parsed.

The database components in Fig. 11 have two search functions. A database may be searched for a string given a unique identifier (index) and conversely it can be searched for the index corresponding to a given string. The data path of these databases is shown in Fig. 13. Strings are stored in the Data ROM. The string length and offset in data ROM are stored in the Length ROM and Offset ROM components of the data path. Various counters, registers and comparators are used to methodically search data as required. The status of the most recent search is saved in a register after an operation is complete.

The database control unit is illustrated in Fig. 14. Basic states are used to write a search index, search string or reset the search string before a major search is performed. When an index-based search is performed, the necessary values from Length and Offset ROM are read and saved before Data ROM
is accessed. Once the Data ROM component has been read the control unit occupies its idle state. This operation must be performed until the entire string has been read. When the last part of the string is read, an index-based search will not produce further data until the index has been reset.

SIP data processed by the XSI must be parsed one character at a time so errors can be detected as quickly as possible. However, SIP characters should be written to SIP memory in parallel as much as possible. The amount of data that can be written depends on the width of the data bus (in hardware and software) and the amount of data available at any given time. The SIP Character Generator contains the functionality to write data to SIP memory in parallel given the aforementioned issues. It is assumed that the architecture has a data bus width of 32 bits. Therefore, either one, two, three or four characters can be written into SIP memory at once depending on the sequence of characters being written at a given time.

The data path for the SIP character generator is shown in Fig. 15. Inputs to all registers (denoted R) are multiplexed so that any combination of four characters can be written at any time. The correct combination of characters to be written is determined by the last character saved and the number of characters to be written (one, two, three or four). Data must be saved into the registers before it is written to SIP memory.

Fig. 16 demonstrates a high level description of the SIP character generator control unit. Its functionality is divided into five finite state machines. One finite state machine acts as an interface while the others implement the functionality to write one, two, three or four characters of SIP data. The interface merely activates the appropriate finite state machine to generate data and waits until it is complete before occupying its idle state.
After the XSI has processed a received SIP packet for syntactical errors it is processed by the SIP Parser in the SIP Engine. The SIP parser has two primary tasks once it has been determined that a packet is syntax error free. First, it must be determined if the packet has the correct number and sequence of SIP fields given the packet type. Second, if the packet is a response then certain information must correspond to the originally transmitted packet. For example, if an OK packet is received in response to an INVITE packet then certain contents of the OK packet must be the same as the original message like the To, From and Call-ID fields. This information is verified by checking the contents of the ‘Transmitted Data’ module, which is modified every time a packet is generated. Any inconsistencies in the packet contents cause an error to be generated by the SIP Parser and returned to the XSI through the SIP Interface. Like the SIP Packet Generator, the SIP Parser is an optimized finite state machine optimized to perform the aforementioned tasks.

IV. RESULTS

The hardware implementation of SIP is currently in development. Functionality exists to correctly process SIP packets in the XSI and generate packets for Instant Messaging in hardware. The software component in Fig. 7 in addition to a graphical user interface (GUI) to interact with the embedded system through the Ethernet. Multimedia and instant messaging abilities are implemented directly in the GUI. The hardware and software component are both implemented on the Altera EP1S40F780C5 FPGA with a thirty-two bit wide data bus. Communication between both components occurs through the Avalon bus architecture [12]. The GUI is implemented with Java while the multimedia functionality is provided through the Java Multimedia Framework (JMF) [13].

As many as four bytes are written to SIP memory when a packet is generated due to the thirty-bit data bus. Depending on various operations that may take place, either one, two or three or four bytes of data are written to SIP memory at any time in the sequence of writing an entire SIP packet. However, when a SIP data is received it is first be parsed one character at a time by the XSI Parser Module (Fig. 8) for error detection purposes. This process also organizes a SIP packet into separate fields so it may be read by the SIP packet parser (Fig. 10) four characters at a time.

In the absence of errors, reading and processing SIP packets each require several cycles per character of data. SIP packets generally do not exceed 1000 characters. The SAMPLE sip packets in this papers are consistent with typical use of SIP and are substantially smaller, especially when payloads are not present. The Altera EP1S40F780C5 FPGA uses a 50 MHZ clock, thereby requiring 50 cycles to examine one character if a 1000 character SIP packet is processed within one millisecond. Processing a character should not require more than approximately ten to fifteen cycles per character accounting for miscellaneous operations that may take place during packet parsing or generation, most notably database searches. Considering that SIP software implementations require several milliseconds of time to perform operations, the potential for increased performance and efficiency with a reconfigurable implementation of SIP is extremely significant.

V. CONCLUSION

This paper presented a reconfigurable architecture for SIP. End system functionality like session establishment, presence and instant messaging were integrated into the hardware component of the architecture while other functions were delegated to software. The implemented system to date has instant messaging completed and allows for a video session to be established through a Java GUI interacting with the hardware component through Ethernet.

Future work includes the development of the XML engine and its integration into the hardware component for PIDF payloads. A completed hardware component would also allow for other high level functions to be implemented directly in hardware like SIP P2P networking[14] or IP Multimedia Systems (IMS)[15].

REFERENCES