Sindh Univ. Res. Jour. (Sci. Ser.) Vol.44 (2) 131-142 (2012)

SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES)

Real Time Video Steaming and Region of Interest Transmission System

M. FRAZ, Y. A. MALKANI*, L. D. DHOMEJA**, M. A.ELAHI

Department of Electrical Engineering, COMSATS Institute of Information Technology, Lahore

Corresponding author: L. D. DHOMEJA Email: lachhman123@yahoo.com Cell. No. +92 3223077286

Received 03rd September 2011and Revised 23rd January 2012)

Abstract: Recently, video streaming has been extensively used for information broadcasting. Availability of improved and enhanced transmission facilities make it possible to use video streaming in fast real time applications. Real time video transmission is widely used in surveillance, conferencing, media broadcasting and applications that include remote assistance. Though a lot of work has already been done in the field of video streaming and many systems are already available, most of them either use a dedicated physical medium for data transmission or rely on general purpose computer systems as servers; which make these systems less efficient and un-suitable for such real time applications that require high speed dedicated video servers. This issue can be resolved by deploying special purpose high speed dedicated embedded processors. In this paper, we present the design and implementation of a real-time video streaming system by using an embedded media processing platform. The system deals with the issues of real-time protocol implementation on captured video and serving it to a client system via Ethernet which plays it in real-time. Further, in this piece of work we also deal with 'Region of Interest' (ROI) transmission to improve processing efficiency of the system, which allows video fragmentation and communication together in real-time.

Keywords: Streaming, Region of Interest, DSP Platform.

1. INTRODUCTION

Video technology is continuously growing and is being used in many sophisticated applications. It is not only replacing the previously implemented solutions but is also providing facilities for developers and engineers to enhance features of video applications by including video communication. It is the need of time to enhance the efficiency of video processing systems for advance and time critical applications. Video processing is quite hectic job for processing systems due to high throughput of video data. When processing combines with transmission of video from one place to other, then general purpose processing systems fail to efficiently handle video processing and communication together in real-time. This problem makes video processing less feasible for real-time applications.

The main objective of this work is the implementation of a real time video streaming system on an embedded Digital Signal and Media Processor (DSP). The processor works as server that takes video data from camera, converts that data into digital format, packetized it in real time using streaming protocol (i.e. RTP) and broadcast these packets over the Ethernet by using user datagram protocol (UDP) over internet protocol (IP). To ensure correct transmission of data, a client application is developed, which runs on a computer system (client) that is also connected to Ethernet. Client receives the packets, extracts data from them, reforms that data into video frames and display them on screen.

This paper is the extended version of the paper (Fraz, 2009) published in the conference proceedings of IEEE 2nd International Conference on Computer, Control and Communication (IC4).

1. Background

Currently, video streaming systems are going through transition where more and more traditional analog solutions and systems are being replaced by newly available digital systems. The reason is that traditional systems are not easily expandable and have low video resolution with little or no signal processing. However, Internet Protocol (IP)-based digital systems having high speed dedicated processors not only offer fast processing but also

* Institute of Mathematics and Computer Science, University of Sindh, Jamshoro.

**Institute of Information and Communication Technology, University of Sindh, Jamshoro.

provide ease in implementing advance features such as motion detection, facial recognition, object tracking and cyber security (Austerberry, 2005).

Video streaming has significant requirements of bandwidth, delay, and loss (Wu, 2001). Currently internet technology does not offer any QoS guarantees to streaming video over the internet. In addition, the diversity of internet makes it really difficult to support video broadcasting while providing flexibility to meet QoS requirements (Wang, 2002). Thus video streaming over the internet offers many challenges such as video compression, application-layer QoS control for streaming video, continuous media distribution services, streaming servers, media synchronization mechanisms and protocols for streaming media.

Video compression techniques effectively reduce the bandwidth required to transmit digital video via terrestrial broadcast, via cable, or via satellite services (Hanzo, 2007). Though video compression improves processing efficiency, it also affects the quality of the image, since all compression techniques work on the principle of eliminating redundant data. Therefore, Video compression has a trade-off between processing efficiency, disk space, video quality and the cost of system required to decompress the video in a reasonable time. In some applications, we are not interested in processing of whole frame, instead we want to work on particular regions of the captured video, which are termed as "regions of interest" (ROI). Towards this, captured video is fragmented to separate ROI from the whole frame, and then that region is transmitted only over the physical layer. This method not only makes system more application specific by omitting unwanted data, but also reduces the load on physical layer.

To handle the above discussed challenges in real time video streaming, advance digital signal and media processors (DSP) are the latest available systems that can meet the requirements of an efficient real time video server. These dedicated processors are fully capable of handling high throughputs and can provide fast enough processing to ensure less delay and better quality as compared to other available systems based on general-purpose computer systems.

2. Hardware design

A. *Digital Signal Processor (DSP)*

A 'Digital Signal Processor' is a special purpose CPU (Central Processing Unit) that provides ultra-fast instruction sequences, such as 'shift and add' and 'multiply and add', which are commonly used in math-intensive signal processing applications.

DSPs are not same as common microprocessors. Microprocessors are typically general purpose devices that run large blocks of software. They are not often called upon for real-time computation and they work at a slower pace, choosing a course of action, and then waiting to finish the present job before responding to the next user command. A DSP, on the other hand, is often used as a type of embedded processor that is built into another piece of equipment and is dedicated to a single group of tasks. In this environment, the DSP assists the general purpose host microprocessor.

B. *Da-Vinci TMS320DM6437 Digital Media Processor*

The TMS320C64x+™ DSPs (including the TMS320DM6437 device) are the highest-performance fixed-point DSP generation in the TMS320C6000™ DSP platform. The DM6437 device is based on the third-generation high-performance, advanced VelociTI™ very-long-instruction-word (VLIW) architecture developed by Texas Instruments (TI), making these DSPs an excellent choice for digital media applications (Texas Instruments, 2008). Further, the DM6437 comes with a complete set of development tools. These include C compilers, a DSP assembly optimizer to simplify programming and scheduling, and a Windows™ debugger interface for visibility into source code execution.

The detailed description of DM6437 is out of the scope of this paper, however interested readers can read its further detail from (Texas Instruments, 2008).

C. *Video Port Interpolated Controller*

Da-vinci platform offers a fully functional video port controller with lots of functionalities that help in developing advance application. The video port peripheral can operate as video input port, video display port or transport stream interface (TSI) capture port (Texas Instruments, 2006).

The high level block diagram of video port interpolated controller is shown in **(Fig. 1)**.

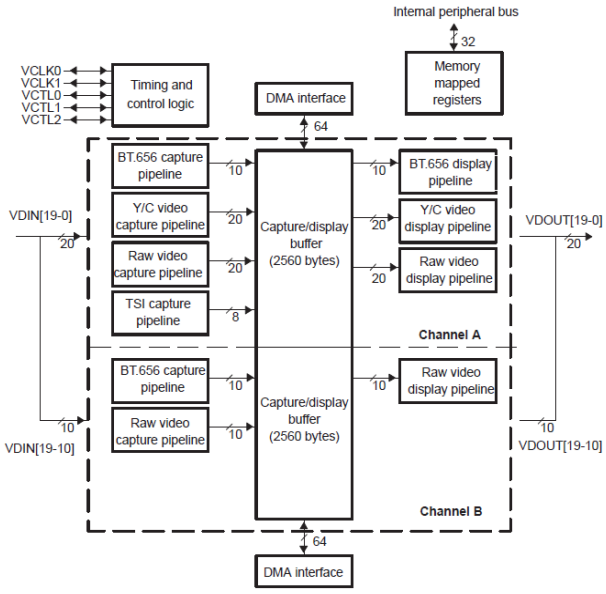

Fig.1: High level architecture diagram of video port

The port consists of two channels: A and B. We can split a 5120-byte capture/display buffer between the two channels. The entire port (both channels) is always configured for either video capture or display only. Separate data pipelines control the parsing and formatting of video capture or display data for each of the BT.656, Y/C, raw video, and TSI modes.

For video capture operation, the video port may operate as two 8/10-bit channels of BT.656 or raw video capture; or as a single channel of 8/10-bit BT.656, 8/10-bit raw video, 16/20-bit Y/C video, 16/20-bit raw video, or 8-bit TSI.

For video display operation, the video port may operate as a single channel of 8/10-bit BT.656, 8/10-bit raw video, 16/20 bit Y/C video, or 16/20-bit raw video. It may also operate in a two channel 8/10-bit raw mode in which the two channels are locked to the same timing. Channel B is not used during single channel operation (Texas Instruments, 2006).

Overall video processing system is divided into two sections **(as shown in Fig. 2)**:

- Video Port Front End (VPFE): VPFE is an interface for external imaging peripherals such as image sensors, video decoders etc.
- Video Port Back End (VPBE): VPBE is an interface for display devices, such as analog SDTV displays, digital LCD panels and HDTV video encoders, etc.

In addition to these peripherals, there is DMA and set of common buffer memory that ensures the efficient use of DDR2 bandwidth. The buffer provide interface between VPFE and VPBE components via a 128-bit wide bus (Texas Instruments, 2008).

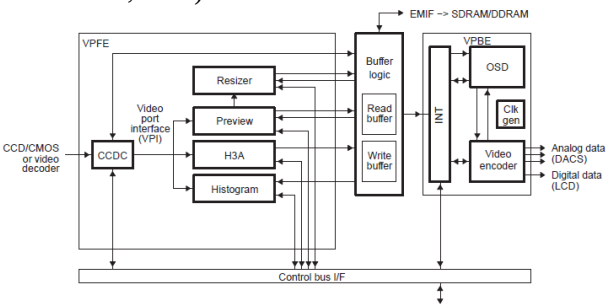

Fig. 2: Video Processing Sub System (VPFE & VPBE)

In VPFE, CCDC controller is responsible to receive video data from the image sensor (i.e. camera). It then forwards this data to pre processing modules which process the input data and then either store it in memory buffer or pass it on to VPBE for display. VPBE contains encoder that converts data into required display driver format (i.e. TV, LCD).

2. MATERIAL AND METHODS

1). *Ethernet Media Access Controller*

Ethernet media access controller module provides an efficient interface between core DSP processor and Ethernet. The EMAC supports both 10Base-T (10Mbits/sec) and 100BaseTX (100Mbits/sec), in either half or full duplex, with hardware flow control and quality-of-service (QoS) support (Texas Instruments, 2004).

The basic feature set of the EMAC module is:

- EMAC acts as DMA master to either internal or external DSP memory space.
- Standard Media Independent Interface (MII) to physical layer device (PHY).
- Eight receive channels with VLAN tag discrimination for receive quality of service (QOS) support.
- Eight transmit channels with round-robin or fixed priority for transmit quality of service (QOS) support.
- Synchronous 10/100 Mbit operation.
- Ether-Stats and 802.3-Stats statistics gathering.
- Transmit CRC generation selectable on a per channel basis
- Broadcast frames selection for reception on a single channel.

- Multicast frames selection for reception on a single channel.
- Promiscuous receive mode frames selection for reception on a single channel (all frames, all good frames, short frames, error frames).
- Hardware flow control.

The EMAC module contains the following logical functions:

- Receive DMA engine
- Receive FIFO
- MAC receiver
- Transmit DMA engine
- Transmit FIFO
- MAC transmitter
- Statistics logic and RAM
- Control registers and logic

All logic is clocked synchronously with the CPUclk/4 peripheral clock except for the Ethernet MII synchronization logic. All the logical blocks and their interconnection are shown in **(Fig. 3)**.

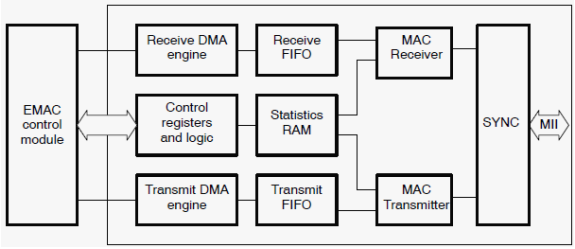

Fig. 3: EMAC Module Block Diagram (Texas Instruments, 2004)

D. *EMAC Control Module*

The main interface between EMAC module and DSP processor is provided by EMAC control module.

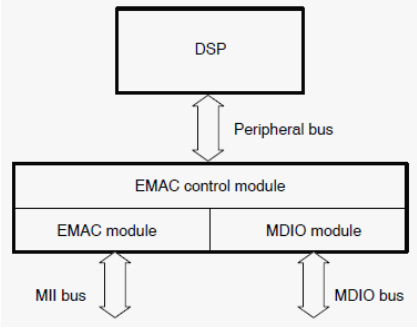

Fig. 4: EMAC Control module

The EMAC control module contains required components to allow the EMAC to make efficient use of DSP memory, plus it controls device reset, interrupts and memory interface priority. Interested readers can read the detailed features of EMAC control module from (Texas Instruments, 2004). However, in **(Fig. 4)**, we have depicted the role of EMAC control module in the communication of DSP processor and EMAC module.

E. *MDIO Module*

The management data input/output (MDIO) module implements the 802.3 serial management interface to interrogate and control Ethernet PHY(s) using a shared two-wire bus. Host software uses the MDIO module to configure the auto-negotiation parameters of each PHY attached to the EMAC, retrieve the negotiation results, and configure required parameters in the EMAC module for correct operation. The MDIO module interfaces to the outside world through two MDIO pins (MDCLK and MDIO), and to the DSP core through the EMAC control module. The MDIO module consists of the following logical components:

- MDIO clock generator
- Global PHY detection and link state monitoring
- Active PHY monitoring
- PHY register user access

Logical functions and their interconnections are shown in **(Fig. 5)** below:

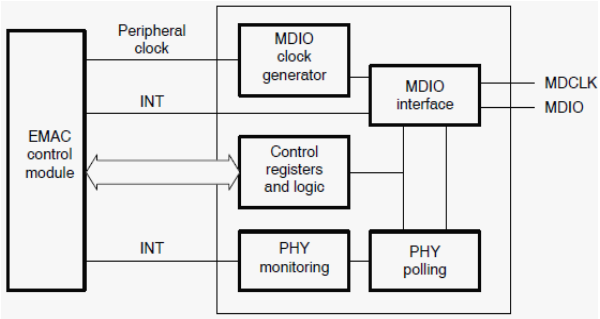

Fig. 5: MDIO Module block diagram

F. *Architectural Overview of EMAC and MDIO Modules*

(Fig. 6) shows the three main functional modules of the EMAC/MDIO peripheral: EMAC control module, EMAC module, and MDIO module. The main interface between the EMAC control module and the DSP core is also shown. The following connections are made to the DSP:

- The peripheral bus connection from the EMAC control module allows the EMAC module to read and write both internal and external memory through the DSP.s memory transfer controller (similar to an EDMA) (Texas Instruments, 2004).
- The EMAC control module, EMAC, and MDIO all have control registers.

- These registers are memory mapped into DSP memory space via the DSP config bus. Along with these registers, the control modules internal RAM is mapped into this same range (Texas Instruments, 2004).

The EMAC and MDIO interrupts are combined into a single interrupt within the control module. The interrupt from the control module then goes to the DSPs interrupt MUX. The EMAC and MDIO interrupts are combined within the control module, so only the control module interrupt needs to be monitored by the application software or device driver. The interrupt is mapped to a specific DSP interrupt through the use of the interrupt MUX. The interrupt selection number of the combined EMAC/MDIO interrupt for use with the MUX is 11000b (Texas Instruments, 2004).

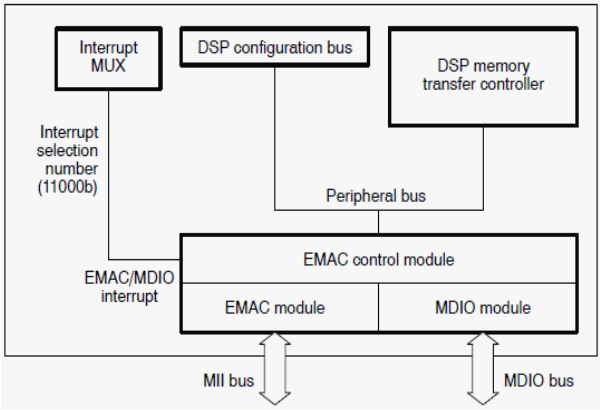

Fig. 6: High level architecture view of EMAC and MDIO module

2) SYSTEM DESIGN AND IMPLEMENTATION

G. *Code Composer Studio IDE*

Code composer studio is a development environment designed for the Texas Instruments (TI) high performance TMS320C6000(C6000) and other digital signal processor (DSP) platforms, the Code Composer Studio IDE is a development environment that tightly integrates the tools needed to create advance DSP applications. The IDE environment offers following major components (Code Composer Studio, 2004):

- Tuning tools for optimizing applications
- C/C++ Compiler, Assembly Optimizer and Linker (Code Generation Tools)
- Real-Time Operating System (DSP/BIOS)
- Ability to dynamically connect and disconnect from targets
- Real-Time Data Exchange between host and target (RTDX)
- Update Advisor
- Instruction Set Simulator
- Advanced Event Triggering
- Data Visualization

In our DSP platform, applications are designed to run separately in the form of threads so that efficiency of complex applications remains better as compare to super-loop approach. To support multithreading, a real-time kernel is necessarily available that could allocate applications to threads. The DSP/BIOS kernel provides an efficient set of kernel, real-time analysis, and peripheral configuration services, eliminating the need to develop and maintain custom DSP operating systems. DSP/BIOS is an integral part of the Code Composer Studio IDE. The kernel object browser displays the state of operating system objects (such as tasks and semaphores), and provides data on stack usage and stack overflow/underflow. These capabilities make it easier to debug applications and optimize usage of OS and system resources (Code Composer Studio, 2004).

Apart from this, Code composer studio provides a lot of configuration tools, project support tools, libraries and API's for drivers configuration and application development that ease in progress of advance applications. Figure 7 shows the user interface of Code Composer Studio IDE.

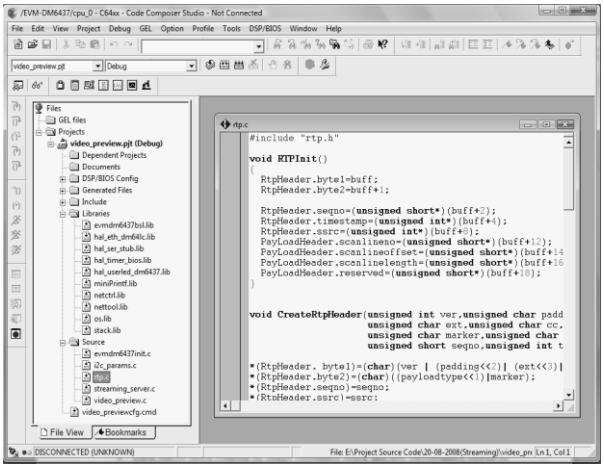

Fig. 7: Code Composer Studio IDE Interface

H. *System Architecture*

(Fig. 8) illustrates the generic architecture of the streaming system. The process of video streaming starts with image sensor that captures video data and transfers it to the server. In this case, DSP platform works as real-time video server that takes data, pre-process it and performs RTP packetization. These RTP packets are then transferred to Ethernet using

UDP. The client develops communication with video server using server's unique IP address and starts receiving RTP packets. These packets are further processed for extracting video data from them. The recovered data is encoded to RGB format, which is played on screen by the media player. The detailed design and implementation process is discussed in subsequent sections.

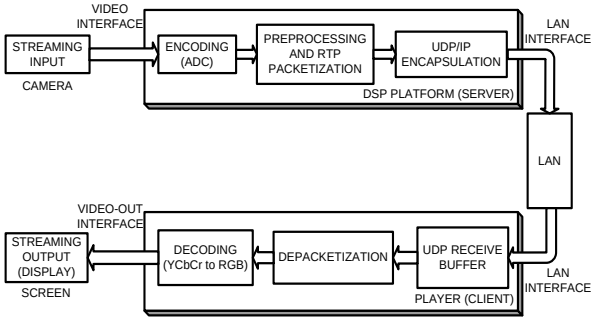

Fig. 8: High-level system architecture

I. *Server Side System Design and Implementation*

1) Capturing and Conversion

Camera gives analog video signal to the input video port of the server. First thing that server performs on incoming analog data is to convert it into digital format so that it can be processed with digital signal processor. The platform supports a number of digital formats. We have used 8-bit YCbCr 4:2:2 format. A wire, having BNC connector at one end (camera side) and RCA phono (analog composite) at the other end (server side) provides interconnection between camera and server, as shown in **(Fig. 9)**.

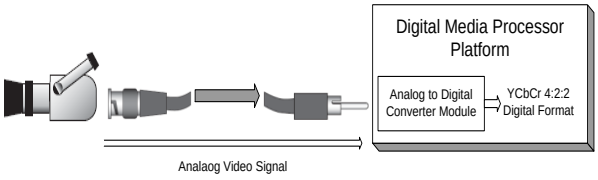

Fig. 9: Video transfer from camera to DSP

Analog-to-Digital converter stores digital data to a temporary buffer that stores this data frame by frame in a one dimensional matrix form. We then process sampled data in line-by-line form from each frame. The line length depends upon frame resolution.

2) Data Organization in Memory

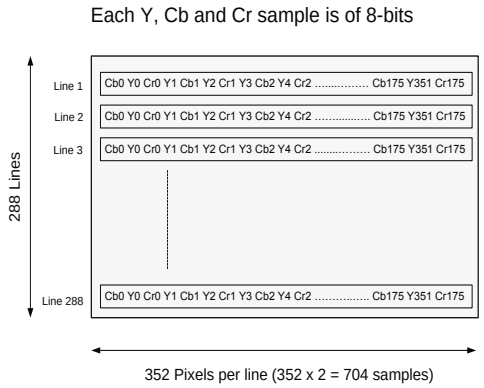

Fig. 10: A 352x288 frame.

The frame resolution deployed here is 352 x 288. It means that each frame comprises of 288 lines and each line has 352 pixels in it. Since, we have used 4:2:2 format, thus, for every 4 pixels there are 4 Y samples, 2 Cb samples and 2 Cr samples (Cb and Cr samples are shared among consecutive Y sample). Overall, each pixel becomes 16 bits in size and total number of samples in each line becomes 352*2 = 704 samples or bytes. Figure 10 depicts the information contained within a video frame. The arrangement of samples is also notable, as each line starts with a Cb sample instead of a Y. Actually, the frame resides in memory in the form of a long 1D matrix as shown in **(Fig. 11)**.

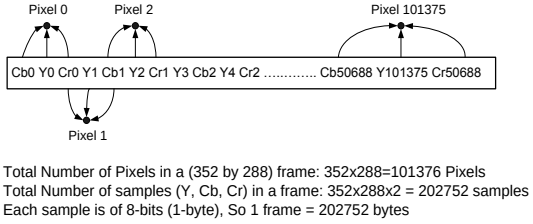

Fig. 11: Memory map of a (352x288, 4:2:2 formated) frame

3) Packtization

Once data is buffered, the process of RTP packetization comes next. In RTP packetization, data samples are picked from buffer and placed in the data field of RTP packet. In this way, memory pointer keeps on proceeding according to the size of a line (704 bytes) and collects next line of data for every new RTP packets. This process is shown in **(Fig. 12)**.

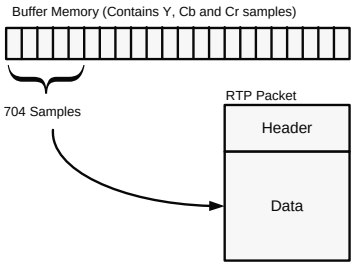

Fig. 12: RTP packetization

Each packet has a header that holds specific information about arrangement of data, its format and encoding information. The information contained in a packet's header is helpful for client to reform received video data into its original form. Here, uncompressed data is being packetized according to RFC-4175 payload format (Perkins, 2005), which describes the standard way of forming an RTP packet from uncompressed video data. According to the standard, for first line of data, the frame contains the following **(Table 1)** information in each field.

Table 1: RTP Packet

Bytes	0-1	2	3	4-7	8	9-15	16-31
0 – 3	10	0	0	1000	0	0000000	0000000000000000 (for unicast)
4 – 7	000000000000001 (Line no, is used as timestamp)						
8 – 11	00000000000000000000000000000000 (for unicast streaming)						
12 – 15	0000000000000000						0000001011000000(Decimal= 704)
16 - 19	0	000000000000001 (for line 1)				0	000000000000000 (no offset)
20 – 723	704 Cb,Y, Cr Octets						

The maximum amount of data that a packet can hold depends upon maximum size supported by the protocol. However, we can send as much data as we want (up to maximum allowed limit) depending on the requirements of application. The process of packetization affects several things. For instance, if we want to send very less amount of data in a single packet, then more packets will be needed for single frame, which will require extra processing. On the other hand, if too much data will be held in a single packet then the receiver efficiency will be affected, as it has to receive and process large amount of data at a time.

As soon as a packet is formed, it is then forwarded for UDP implementation. UDP develops communication with the physical port through which it broadcasts data. UDP header is inserted with RTP packetized data. Then by using Internet Protocol, packet is conveyed to physical medium through which it travels to client. The organization of this procedure is shown in **(Fig. 13)**.

4) UDP/IP Implementation

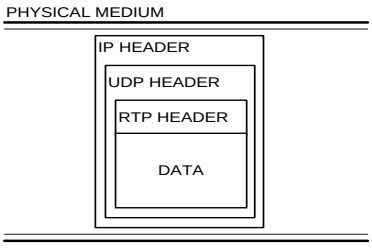

Fig. 13: Protocol hierarchy in a data packet

As UDP is being used here for streaming over IP which only allows sending a maximum of 8192 bytes at a time in one packet; therefore we have to keep our data up to or less than that limit. One complete frame cannot be transferred via one packet as each frame is of 202752 bytes. For the purpose of simplicity, one line per packet is being transmitted. Each line is of 704 bytes. Therefore, it is easy for the client to process them in real time according to the server's transmission speed. Using this organization, one complete video frame transmits in 352 RTP packets.

5) Server's Software Implementation

The developed program for server side implementation handles multiple jobs simultaneously. The presence of real-time kernel makes it easy to handle the execution of multiple tasks at a time through parallel processing. It has provided an efficient way to grip both video processing and network communication in a well manner. When server starts, it first initializes video port by configuring video port driver with the provided parameters, then it configures network according to the supplied parameters, i.e. IP address and default gateway. After initialization, it boots the network so that it can start its working. Then, video capture process starts in a separate thread, which receives data from the video port, converts it into digital format (provided at port initialization time), and buffers it in the volatile memory available at the hardware platform. Along with capturing of incoming video, the routine also transfers buffered data to the video output port for preview purpose at server side.

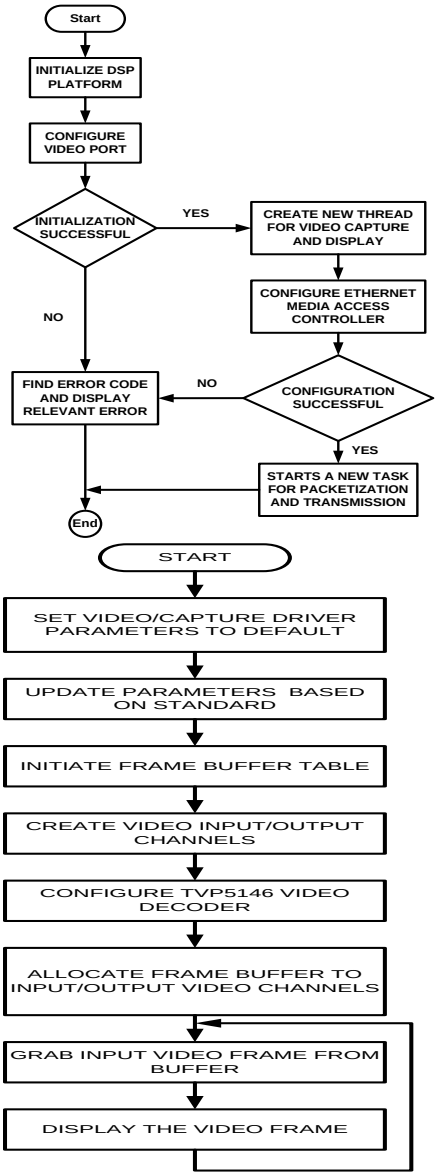

Fig. 14: Flow chart of server initialization program (left) and video capture and preview routine (right)

Apart from video capture and preview, another process executes in parallel that performs the complete working of a video server. It creates a socket, binds it to a port using port number and starts listening for a request from client. As soon as it receives a request for data, it acknowledges the request with a short message. It then starts packetizing video samples by picking them from temporary buffer and forward those packets to Ethernet Media Access Controller (EMAC) using specific socket functions. The socket function API supported by the stack libraries consistent with the standard Berkeley sockets API (Texas Instrument, 2007). The flowchart shown in **(Fig. 14)** (left) depicts the significant jobs that server's main initialization program performs.

The flowchart of video preview routine is shown in figure 14 (right). It shows all the major steps that are performed to capture video from image sensor and to display it at server side. It includes initialization of driver with updated parameters according to in use video standard–i.e. PAL (Poynton, 1996) – creation of software channels that interacts with the physical video ports, allocation of buffer to those channels for storage of incoming video and delivering that video from buffer to video display port.

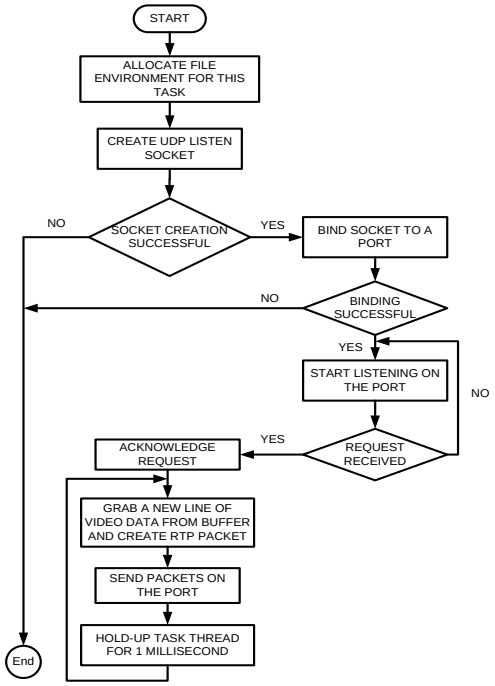

Fig. 15: Flow chart of streaming server thread

The major thread that performs the actual working of a video streaming task has several jobs to do. The flow chart given in figure 15 is showing all major steps of a streaming server's main thread. The thread starts by creating a software socket and binds it to a port. After socket binding it starts listening on that port for an access request. As soon as a request is received from client, it acknowledges the request and start calling RTP routines. RTP procedures create video packets by grabbing data (line by line) from buffer and place it in data field of RTP packets. RTP functions also initializes header of each packet accordingly. RTP routines after creating each packet send its pointer to the main stream server thread from where these packets are delivered to Ethernet using UDP socket routines.

J. *Client Side System Design and Implementation*

1) Architecture

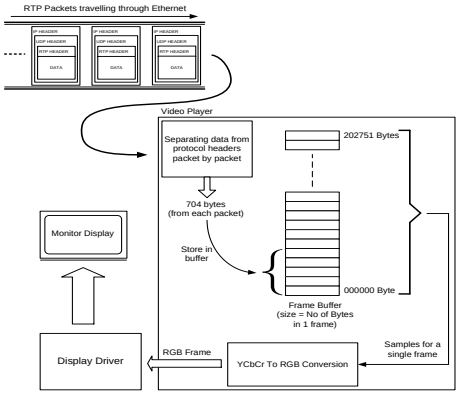

Fig. 16: System architecture for client application

The client side system is a Windows application that plays the received video data in real-time. The client program has to perform some processing on received packets in order to separate data from protocol headers and plays it. The block diagram in **(Fig.16)** elaborates the working of client application.

The client side application uses the same technique for network communication, as it is deployed on the server side. The client creates a socket and binds it to a port. It then sends data request to server by using server's IP address and start waiting for an acknowledgment. The server acknowledges with the data packets so that receiver begins to take video packets and stores the data into memory buffer. As soon as a frame completes, the whole buffer is moved to conversion routine, which converts YCbCr values to RGB (Texas Instruments, 2007). The RGB format is compatible with display driver. Therefore, RGB data is passed to display driver which shows it on screen. As a result, real time video gets played on client side.

Fig. 17: Client's application

The client application's interface is shown in **(Fig. 17)**. It is kept very simple, and does not have any ambiguous options or input fields. It takes two inputs: Server's IP address and Port Number to bind socket. When 'connect and start capturing' button is pressed, after entering a valid server IP address and port number in respective fields, the program sends data request to server and starts receiving video samples, which it displays frame by frame on its 352x288 display field.

2) Region of Interest (ROI) Implementation

With innovation in the field of video streaming, many new features have been introduced. These features not only diversify the use of video streaming in real-time applications but also help in improving the efficiency of serving systems. ROI is one of these features. ROI is a region that human eyes tend to put more attention than the remainder region in an image or video frame. By using this technique, selected regions of interest of digital video can be transmitted at a specified resolution.

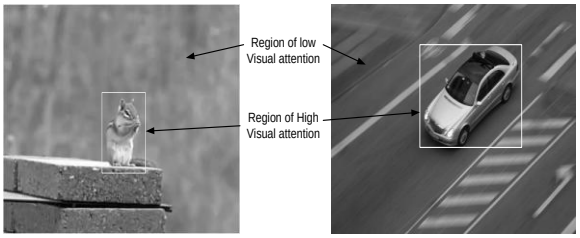

Fig. 18: Examples of regions of high and low visual attention

We also included this capability in our streaming system. Our system is capable of transmitting an ROI from captured frames in real-time. However, a very sophisticated approach has not been developed yet; a simple approach is used to enhance the functionality of our real-time streaming system.

As discussed earlier, for normal video streaming a pointer to memory buffer is used for accessing data to place it in RTP packets. All the video samples are separately accessed and their values are changed through that pointer. Therefore, while making RTP packets, the values are kept same for those video samples (or pixels) that lie in ROI, and the values for all other samples (or pixels) are changed to zero. Thus, the data lied in ROI remains unchanged, and rest of the frame becomes blank. In this way, when these frames arrive on receiving side (i.e. client), only ROI is appeared to be played on screen.

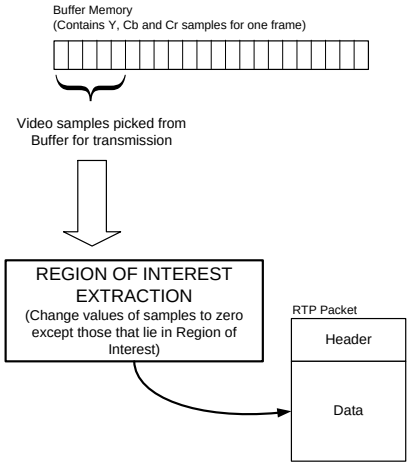

Fig. 19: ROI Implementation

3) Efficiency Improvement

The efficiency of the system can be improved by modifying the above used technique in such a way that instead of transmitting null values (in place of those pixels that do not lay in ROI), we only transmit those pixels that lay in ROI. Other pixels should be omitted from the packet. In this way, reduced packet size will result an improvement in processing efficiency, and accommodate more data from ROI in one frame. Though this technique can improve efficiency, it also raises some other issues, such as packet size will vary at runtime with the size of desired ROI. Thus, server and client need to be intelligent enough for dynamic packet processing. It is out of the scope of this paper and left for the future work.

3. SYSTEM EVALUATION AND RESULTS

A system remains incomplete until it goes through an exhausted series of tests and produces desired results. The streaming system, which has been implemented in our project, works remarkably well. It is capable to stream and play video in real-time with high quality. However, few problems occurred during design and implementation, which are resolved to meet the project requirements. The performance of the system depends on various criterions, such as network connectivity, video quality, delay in real-time display of video, and packet loss.

(Fig. 20) shows a screen shot of video display by the client side player. It depicts that the quality of video is acceptable. As it is video, so it is not possible to demonstrate the amount of delay and real-time working of system in this paper. However, delay of this streaming system depends upon several factors, such as network speed and amount of data in a single packet. We did not face any problems regarding communication medium as the available network was not congested heavily, and it did not cause any packet loss during streaming. However, second point needs some consideration as it directly affects the delay of overall system. We have tried several data amounts to be transferred through a single packet, and all of them had their own trade-off.

Fig. 20: Player displaying real-time video

The least delay occurs in video display, if four lines per packet are sent to the client. The video quality remains very good, and no any significant delay occurs at client application. Thus, it gives the best impression of a real-time system. However, the disadvantage of sending this much data (four lines) comes into play when network do not perform well and packet loss is likely to occur. In that case, one lost packet will miss four lines of data from one frame; hence, multi-packet loss will result in a display of unacceptable quality.

On the other hand if half line is sent through a single packet, it introduces a delay in real-time display but it works well in circumstances where chances of packet loss are high. As every packet contains only half line of data from a frame of 288 lines, so if a packet gets lost during transmission, even then it will not cause much affect on quality of overall frame. The optimized way is to transmit one line per packet that keeps both constraints up to an acceptable level **(Fig 21)**.

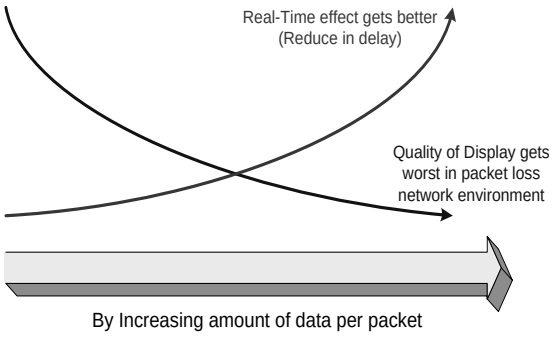

Fig. 21: Effect of increasing data per packet

By changing the amount of data per packet creates a problem at client side player display. The display screen starts flickering by changing the amount of data per packet. Display frames starts to move upwards by an amount of single line with each frame. The problem gets even worst by increasing the amount of data up to four lines per packet and resolves by reducing the amount of data per packet (i.e. up to half line per packet). This problem actually occurs at client side due to mismatch of receiving rate between data processing and refreshing rate of video display driver, since display driver has to refresh screen periodically. The refresh rate and data updating procedure of buffer must synchronize with each other so that display remains stable and smooth for the viewer. Here, refresh rate of video display driver is fast as compared to received amount of data from Ethernet to fill one buffer. As a result, display driver keeps on displaying data before letting display buffer filled with new data.

The receiving delay of data occurs due to suspension of streaming server thread for a specific duration in order to synchronize it with comparatively low speed network. The available 'task suspend function' (TSF) can suspend a task for minimum of one millisecond, which is too high for a streaming server. The ideal delay should be 114.53 microseconds for every line. Calculations are shown as under:

As, Video display driver refreshes the screen at a rate of 30 frames per second, thus, time for 1 frame = 1/30 = 0.0333 seconds or 33.3 milliseconds.
Time for 1 line in a frame of 288 lines = 33/288 = 0.114533 milliseconds or 114.533 microseconds.

Therefore, a function is developed that suspends the streaming task for 114.53 microseconds after transmission of each packet, which results in flawless video display.

(Fig. 22) shows some results for ROI program execution of the implemented system. ROI results are remarkably well with regard to the technique used. All the frames are eliminated successfully during the transmission excluding those that lay in ROI.

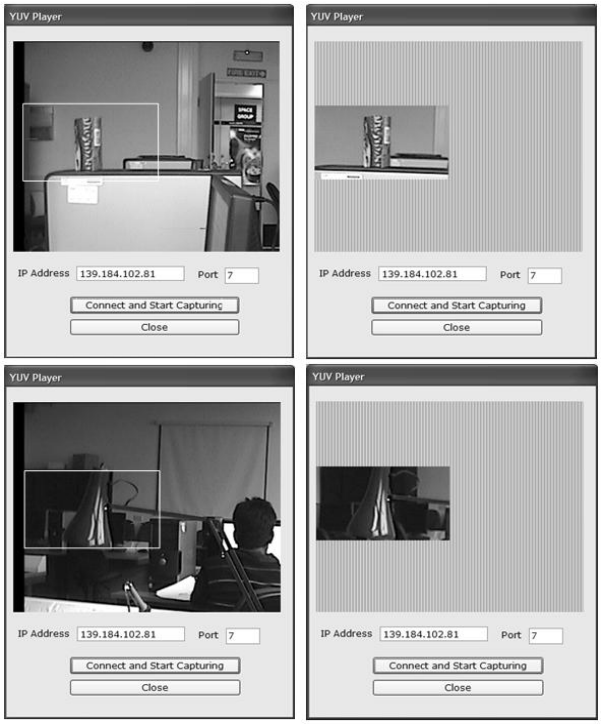

Fig. 22: ROI demonstration

4. CONCLUSION

The implementation of a real-time video streaming system on an embedded media processor has raised opportunities for enhancements at both server and client sides in order to develop productive applications. The piece of work presented in this paper has provided a base for time critical applications to make use of special purpose embedded systems in order to meet tight deadlines. The work has also magnified the importance of certain techniques, such as dynamic RTP packetization, that can improve the performance of any real-time video streaming system. Apart from 'raw' video streaming the ROI transmission also provides a way to enhance the

efficiency of the system. Although, a very sophisticated method is not used to extract ROI, it offers a good working prototype, and much work still needs to be done in this area. However, a number of functionalities can easily be added to the implemented system to make it more efficient and application specific, especially in the context of embedded applications.

REFERENCES:

Austerberry, D., (2005) The Technology of Video and Audio Streaming. 2nd Edition. Oxford, Focal Press.

Code Composer Studio, (2004) Texas Instruments White Paper. Literature Number: SPRAA08.

Fraz, M., Y.A. Malkani, and M. A. Elahi., (2009) Proc. of the 2nd IEEE International Conference on Computer, Control and Communication (IC4), Karachi, Pakistan.

Hanzo, L., (2007) Video Compression and Communications. 2nd Edition. West Sussex, Wiley and Sons.

Perkins, C., (2005) RTP payload for uncompressed video. Internet Engineering Task Force, RFC 4175.

Poynton, C. A., (1996). A Technical Introduction to Digital Video, John Wiley and Sons, Inc., West Sussex, 175Pp.

Texas Instruments, (2004) TMS320C6000 DSP Ethernet Media Access Controller (EMAC)/Management Data Input/Output (MDIO) Module. Literature Number: SPRU628A.

Texas Instruments, (2006) Video port interpolated controller. Literature Number: SPRU629E.

Texas Instruments, (2007) TMS320C6000 Network Developer's Kit (NDK) Software Programmer's: Literature Number: SPRU524C and SPRU198G .

Texas Instruments, (2008) TMS320DM6437 Digital media processor. Literature Number: SPRS345D and SPRU977A.

Wang, Y. (2002). Video processing and Communication. Prentice Hall.

Wu, D., Y. T. Hou, W. Zhu, Y. Q. Zhang, and J. M. Peha, (2001) Streaming video over the Internet: approaches and directions. IEEE Trans. Circuits Syst. Video Technol., vol. **(11)**: 282-300.