Implementation of an Embedded H.264 Live Video Streaming System

N Vun, M Ansary

School of Computer Engineering
Nanyang Technological University
Singapore

***Abstract*—This paper presents the methodologies taken to integrate open-source LIVE555 based data streamer with a baseline H.264 encoder running on the Texas Instruments's DaVinci based embedded platform. The system implemented is able to stream live H.264 encoded video over the network, and be displayed on remote stations using the VLC media player. The system developed provides an embedded platform to implement a smart surveillance camera system, whereby video analysis can be performed locally on the embedded platform to minimize the streaming throughput by making use of the H.264 motion vector information.**

Keywords- Real time Streaming, H.264, LIVE555, Embedded

I. INTRODUCTION

Previous ISCE papers [1] [2] presented the development of an H.264 embedded encoder with the aim to implement a smart surveillance camera system by incorporating video analysis routines locally in the encoder running on the embedded platform. The operating scenario is to have the camera be able to selectively transfer the video to remote station(s) upon the detection of relevant incidents. By only transmitting important motion vectors or macroblocks that have changed, the transmission bandwidth can be further reduced while remain H.264 compatible. A logical extension of such a system is to further include a video streaming capability on the embedded platform to provide a real time live video feed as and when is necessary. This paper hence focuses on the implementation of a H.264 media streaming embedded system that is based on an open-source media streamer and a software based H.264 encoder, running on the TI's DaVinci based DM6446 DVEVM evaluation board [3] [4] with video camera that provides live video frames.

The embedded streaming system makes use of the LIVE555 media server[5], which is modified and cross-compiling to run natively on the DaVinci board. A DaVinci H.264 encoder demo program is also modified such that the encoded video is streamed to the Live555 media server for multicast over IP network. With this system setup, the video stream can then be viewed on remote stations using the open-source VLC media player, which contains the relevant H.264 codec for such purpose. Figure 1 illustrates the typical setup of system implemented.

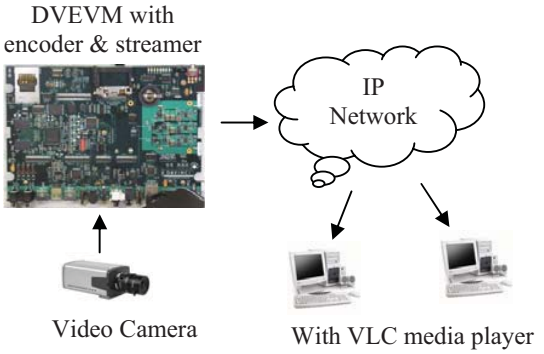

Figure 1. System Setup

The remainder of the paper is organized as follows. Section II examines the relevant real-time streaming protocols used in the system. Section III presents the LIVE555 media server, and the modifications that are done to support multicasting of H.264 video frames. Section IV describes the operation of the DM6446 DVEVM and its encoder program. Section V discusses the issues faced in the integration of the LIVE555 with the encoder program on the embedded platform, and some concluding remarks made in Section VI.

II. STREAMING PROTOCOLS

The key protocols used to perform real-time data streaming the systems are based on the following three standards: Real-Time Streaming Protocol (RTSP), Real-time Transport Protocol (RTP) and the Session Description Protocol (SDP).

A. RTSP

Real-Time Streaming Protocol (RTSP)[7] acts a control protocol to media streaming servers. It establishes connection between two end points of the system and manage the media sessions. Clients issue VCR-like commands like play and pause to the server to facilitate the control of real-time

978-1-4244-6673-3/10/$26.00 ©2010 IEEE

playback of media streams from the servers. Some of the commands commonly used are as follows:

- *OPTIONS*
- *DESCRIBE*
- *SETUP*
- *PLAY*
- *PAUSE*
- *RECORD*
- *TEARDOWN*

These commands are sent as message through a RTSP URL. An example of such a client request using *OPTIONS* is as follows (assuming a port number of 554):

OPTIONS rtsp://155.69.148.136:554/sample.264 RTSP/1.0
CSeq: 1\r\n
User-agent: VLC media Player

The *CSeq* parameter keeps track of the number of request sent to the server, and is incremented every time a new request is issued. The *User-agent* refers to the client making the request. Below is another example using the *Describe* command, where the *Accept* header is used to describe the formats understood by the client (which in this case, is the SDP used by the VLC media player).

DESCRIBE rtsp://155.69.148.138:554/test.264 RTSP/1.0
CSeq: 2\r\n
Accept: application/sdp\r\n
User agent: VLC media Player

In response to the client's request, the server will response with ASCII based status code similar to HTTP. Examples of the status codes with their meaning are as follows.

- *200: OK*
- *301: Redirection*
- *405: Method Not Allowed*
- *451: Parameter Not Understood*
- *454: Session Not Found*
- *457: Invalid Range*
- *461: Unsupported Transport*
- *462: Destination Unreachable*

B. RTP

The Real-time Transport Protocol (RTP)[8] defines the structure of the packet which is used to transport media streamed over the network. It also manages jitter compensation and detection of incorrect sequence arrival of data, which could occur for transmission over IP network. Due to the high latency of TCP protocol in establishing connections, RTP is often built on top of the UDP. In addition, RTP also supports multicast transmission of data.

The packet structure contains a header containing fields that describe the payload appended. For example, the 7-bit *Payload Type* field with value "60h" would indicate that the payload is a H.264 video encoded at 90KHz. A 16-bit *Sequence Number* is incremented for every RTP packet sent, which is used to detect packet loss and out of sequence packet arrival. A 32-bit *Time Stamp* is to be used by the receivers to play the media samples at correct intervals of time, and a 16-bit *Length* indicates the size of the payload.

C. SDP

The Session Description Protocol (SDP) [9] is a standard to describe the initialization parameters for the media session. In the system implemented, the understanding of SDP is important in streaming as the client such as the VLC Media Player expects a SDP description from the server in order to setup the session and facilitate the playback of the streaming media using the RTSP protocol.

The SDP contains description of the session, time and media, which are used for session announcement, session invitation and parameter negotiation. This protocol can be used together with RTSP, as in the *DESCRIBE* RTSP message example earlier, to get session's media initialization parameters.

III. LIVE555 MEDIASERVER

The Live555 Media Server [5] is a well defined complete RTSP open-source server application, and uses RTSP, RTP and SDP protocols for streaming media, and is compatible with media player like VLC and QuickTime. However it does not support H.264 media file, and is not as binary for ARM processor based platform like DaVinci. However, as it is an open-source application, its source code is readily available and can be modified to suit specific requirement, which is the main reason to be used for implementing the streaming system described here. The followings describe how the LIVE555 is modified for this purpose.

As LIVE555 is a C++ implementation, support of H.264 is done by implementing classes to encapsulate the streaming of the H.264 video media file. Five classes are used to achieve the task as follows

i. *VideoFileSink*: This adds a 4-byte start code (0x000001) to the media file, and continues to write the rest of the data of the media file into this output file

ii. *VideoFileServerMediaSubsession*: This creates a dynamic session for streaming the data over the network

iii. *VideoRTPSink*: This facilitates in the packetization of NALs to be sent over the network

iv. *VideoStreamFramer*: This classifies the input video data into frames. This is done by continuously reading the input file and identifying the data which is contained in each frame. The frame size is computed and the frame rate of the video is set appropriately by setting the presentation time of each frame.

v. *VideoStreamParser*: This is invoked by the VideoStreamFramer in order to correctly parse the data into frames. It checks for the 4 bytes start code before parsing the frames.

Figure 2 illustrate the interactions of these classes implemented to support the H.264 media format.

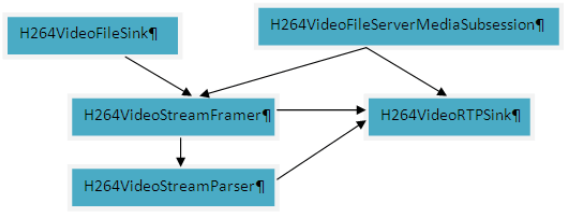

Figure 2. Interaction of classes for streaming H.264

As a networked based video surveillance system, multicast over the IP network is used to allow the video stream to be monitored simultaneously at multiple stations. The multicast support in LIVE555 media server is available through the *PassiveServerMediaSubsession* class, which makes use of broadcast address to stream data to multiple clients. The address is generated at random and it uses the range of [232.0.1.0, 232.255.255.255]. To support multiple connections that start at different instances, *reuseFirstSource* parameter is set in the class.

In order to eventually integrate with the H.264 Encoder on the embedded platform, the LIVE555 is also modified to enable communication using socket based IPC, which will be further explained in later section.

The last step involves the cross-compilation of the LIVE555 source code, which is written for the GNU C++ compiler running on Linux x86 based station. To port it to the ARM based DaVinci board, it needs to be recompiled using the MontaVista embedded Linux[10] based tool chain that make uses of the Arm C and C++ cross compiler. This process involves the modification of the relevant Makefiles.

IV. DM6446EVM ENCODER PLATFORM

The TI DM6446EVM [3] platform is based on the TI DaVinci technology which is designed for multimedia applications together with networking capability. It consists of three subsystems:

i. ARM9 processor Subsystem which manage the overall control operation of the platform. For the system implemented, it also handles the LIVE555 media server that stream the video frames over the network.

ii. DSP Subsystem supported by the Video and Imaging Coprocessor (VICP) and its signal processing library. It is responsible for performing the H.264 encoding in this system.

iii. Video Processing Subsystem which perform the processing of the video frames.

The DM6446DVM is effective a dual processor platform, with the ARM processor running on MontaVista Linux OS[10] while the DSP runs using TI's DSP/BIOS RTOS[11]. communication between the two processors is performed using the DSP/BIOS Link [12]. This is done by sharing certain part of the on-board memory as buffers using the Contiguous Memory Allocator (CMEM) kernel module, which is configurable by user.

Of particular interest of the CM6446EVM platform is the A/V encoder, Encode Demo[13], supplied as a demo application with the platform. This encoder program is used as the module that perform the capturing of the video frame, and subsequent encoding of the H.264 media file. This program is run on the Linux OS, with the encoding algorithm processing passed to the DSP for execution.

The Encode Demo application is implemented as a multi-threads C programs. The actual program consists of six POSIX threads what are configured to be preemptive and priority based scheduled. The program uses the individual threads to perform specific functions, with data passed between threads using buffer. Figure 3 shows the interaction of three threads that are relevant to the streaming system implementation.

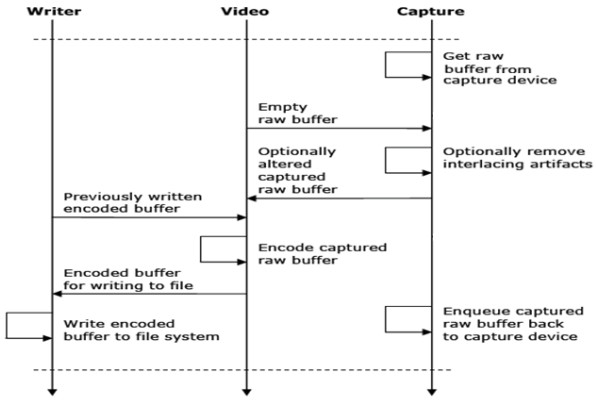

Figure 3. Encode Demo Thread Interactions[13]

The Capture thread first gets hold of the buffer that contains the video frame that is captured by the camera. It then fetches an empty buffer from the Video thread, and fills it with the captured video frame, after doing an optional data pre-conditioning. The buffer is then released back to the Video thread.

When the Video thread receives the buffer with the captured video frame, it fetches the I/O buffer from the Writer thread. It then executes the encoding of the video frame (done by DSP) and stores the encoded frame into the I/O buffer.

When the Writer thread receives the encoded frame in the buffer, it is then written to the Linux file system. The Encode Demo application then repeat with the Capture Thread waiting for the next frame to be supplied by the camera through the VPSS device driver that managed the captured device.

The synchronization of thread initialization, execution and buffer sharing is handled by the Rendezvous utility module.

For the streaming system, the Writer thread is modified to work with the LIVE555 media server, as described in the next section.

V. System Integration

To implement the video streaming system on the embedded platform, both the LIVE555 media server program and the encoder program have to be integrated together in order to achieve live streaming. However, the LIVE555 media server is implemented in C++ codes based on event-driven while the encoder program is implemented using C codes written as a multi-threaded program. To avoid making major changes to either of the programs, the system run the two programs as two separate processes on the Linux. Data transfer is performed through inter-process communication (IPC) using socket programming. Figure 4 shows the basic operation of the two processes using this approach.

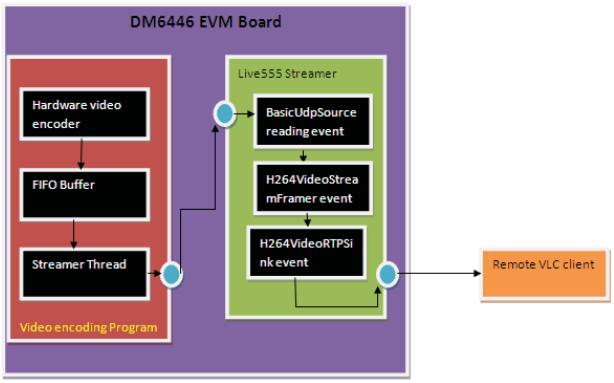

Figure 4. IPC between Live555 events and the Encoder Threads

Several adaptations are done in order to achieve the live streaming of the video frames. The Writer thread in the encoder is replaced by a Streaming thread, while the LIVE555 streamer now reads directly from a socket. UDP sockets are used as video data are loss tolerant but delay intolerant. In this set up, the encoder program acts as the server and live555 as the client

The Streamer thread opens a socket that is bind to the LIVE555 media server as shown. The video encoder program hence operates similar to the original program; captures raw video frame and encodes the video frame into H.264 format. Instead of writing the encoded frame to a file (which can be optionally performed if needed), the Streamer thread writes the encoded video frame to the socket. The LIVE555 media server, which listening to the socket, receives the video frame and streams it over the network. The reading of the data from the socket in LIVE555 is modeled as an event.

VI. Concluding Remark

This paper describes the implementation of a H.264 video network streaming system on the DM6446 DVEVM board, an embedded board based on ARM9 processor running on Linux. The system makes use of the A/V encoder program, Encode Demo provided with the DM6446 DVEVM board, and the open-source RTSP server program, the Live555 Media Server. By using the open-source program, it allows the application to be modified and cross-compiled to run natively on the embedded platform.

Several modifications are made to both programs in order to implement the system. As the LIVE555 streamer does not support H.264, several class are created to encapsulate the H.264 data stream to the format that is compatible with the VLC media player. The encoder program is also modified to include a streaming thread that send the encoded video frame to the LIVE555 media server directly. Because of the different program methodologies used by the two programs (C and C++), the programs are run as two separate processes on the Linux OS environment. Inter Process communication is then achieved through socket programming to allow the passing of the video frame between the two programs. In this way, video frames captured by the DVEVM board can be streamed directly, in both unicast and multicast fashions, over the network and be viewed at remote station using open-source VLC media player capable of playing H.264 video format.

References

[1] N. Vun and T. N. A. Nguyen, “Development of H.264 encoder for a DSP based embedded system”, International Symposium of Consumer Electronics (ISCE) 2007

[2] N. Vun and Y. J. Cai, “Optimization Techniques for a DSP Based H.264 Embedded Systems”, International Symposium of Consumer Electronics (ISCE) 2009

[3] Texas Instrument, TMDSEVM6446: DM6446 Digital Video Evaluation Module, http://focus.ti.com/docs/toolsw/folders/print/tmdsevm6446.html

[4] Texas Instrument, “DVEVM Getting Started Guide”, SPRUE66B, October 2006

[5] LIVE555 Media Server, http://www.live555.com/mediaServer/

[6] VLC Media Player, http://www.videolan.org/vlc/

[7] H. Schulzrinne, A. Rao and R. Lanphier, “Real Time Streaming Protocol (RTSP)”, April 1998. http://www.ietf.org/rfc/rfc2326.txt

[8] H. Schulzrinne, S. Casner, R. Fredernick, and V. Jacobson, “RFC-1889 RTP: A Transport Proctocol for Real-Time Applications”, January 1996. http://www.faqs.org/rfcs/rfc1889.html

[9] Handley, M. and V. Jacobson, "RFC 2327 SDP: Session Description Protocol", April 1998. http://tools.ietf.org/html/rfc2327

[10] MontaVista Embeded Linux, http://www.mvista.com/

[11] Texas Instrument, DSP/BIOS 5.x Real-Time Operating System http://focus.ti.com/docs/toolsw/folders/print/dspbios5.html

[12] Texas Instrument, “DSP/BIOS LINK User Guide”, LNK 058 USR, April 2006

[13] Texas Instrument, “Encode Demo for the DVEVM/DVSDK 1.2.”, SPRAA96A, April 2007