Polling-based Real-time Software for MPEG2 System Protocol LSIs

Jiro NAGANUMA and Makoto ENDO
NTT Sytem Electronics Laboratories
3-1, Morinosato Wakamiya, Atsugi, Kanagawa, 243-01 JAPAN
Tel: +81 462 40 {2139, 2127} Fax: +81 462 40 4322
E-mail: {jiro, endo}@aecl.ntt.co.jp

Abstract— This paper proposes polling-based real-time software for MPEG2 System protocol LSIs, which is a typical embedded and real-time system on a chip, and demonstrates its performance and usefulness. The polling-based real-time software is designed and optimized by analyzing application specific function requirements and deciding scheduling intervals and the execution cycles of each task. It requires neither hardware for multiple interrupt handling nor software for heavy context switching. The polling-based approach provides sufficient performance without any hardware and software overhead for a real-time application like the MPEG2 System protocol.

I. Introduction

Recently, the MPEG2 standard has emerged as a method for effectively compressing video and audio streams, while maintaining their quality. Since the MPEG2 standard aims at retaining high quality, many transmission and storage applications are being considered : satellite broadcasting or cable TV (transmission), and CD-ROM (storage).

A typical MPEG2 environment consists of video and audio encoders, video and audio decoders, a multiplexor (MUX), and a de-multiplexor (DMUX). The "MPEG2 Video" standard [1] specifies the coded representation of video data and the decoding process, while the "MPEG2 Audio" standard [2] does the same for audio data. The "MPEG2 Systems" standard [3] specifies the system layer of the coding, and defines two protocol levels : the Transport Stream (TS) used in the environments where errors may occur ; and the Program Stream (PS) used in the error-free environments. The TS (resp. PS) specifies the format of a multiplexed stream consisting of several audio and video MPEG2 streams over lossy (resp. lossless) physical channels.

To implement the MPEG2 CODEC systems efficiently, the development of key component LSIs was desired. Video encoder [4][5] and decoder LSIs [6][7] meeting the "MPEG2 Video" standard, audio encoders [8] and decoders LSIs [9] meeting the "MPEG2 Audio" standard, and TS DMUX LSI [10] meeting the "MPEG2 Systems" standard have already been developed. We proposed and developed a new memory-based architecture [11] for MPEG2 MUX/DMUX LSIs implementing the MPEG2 System protocol. The architecture supports the full functionality of the "MPEG2 Systems" standard, for both the MUX and DMUX, and for both the TS and PS.

The MPEG2 MUX/DMUX LSI architecture consists of a core CPU, memories, and dedicated application-specific hardware, which is designed and optimized using hardware/software (HW/SW) co-design techniques. The MPEG2 MUX/DMUX LSI is a typical *on-chip* embedded and real-time system with hardware (HW) and software (SW) interactions. An embedded system [12] may provide sufficient flexibility and performance, however the software for real-time applications is especially difficult to develop.

There are two problems linked to the development of real-time applications on an embedded system such as the MPEG2 MUX/DMUX LSIs: *hardware* and *software* overhead for handling the frequent and asynchronous interactions between HW and SW via several dozen HW/SW interface registers. A interrupt-based approach between HW and SW is generally used for real-time applications but requires following expensive resources from the standpoints of both hardware and software:

- The hardware has to support the multiple and asynchronous interruptions from some HW/SW interface registers. A lot of gates are required for multiple interrupt handling (priority control), which complicates hardware design [13].
- The software has to support real-time kernels for scheduling each task. A lot of memory and time are required for heavy context switching. For instance, at least several tens of kilo bytes of memory is needed for most slight real-time kernel [14][15].

These requirements can be met in an on-board embedded system, but it is inefficient or impossible to do so as an on-chip embedded system such as our MPEG2 MUX/DMUX LSIs, which has only 4 kilo bytes of data memory.

To solve both problems (hardware and software overhead), we propose a polling-based real-time software for MPEG2 System protocol LSIs, and demonstrates its performance and usefulness. The polling-based real-time software is designed and optimized by analyzing application specific function requirements and deciding scheduling intervals and the execution cycles of each task. It

ASP-DAC '97
0-89791-851-7$5.00

requires neither hardware for multiple interrupt handling nor software for heavy context switching. The polling-based approach provides sufficient performance without any hardware and software overhead for a real-time application like the MPEG2 system protocol. The MUX/DMUX LSIs with their software are in use on MPEG2 CODEC systems [16] [17] for several multimedia communication and storage services.

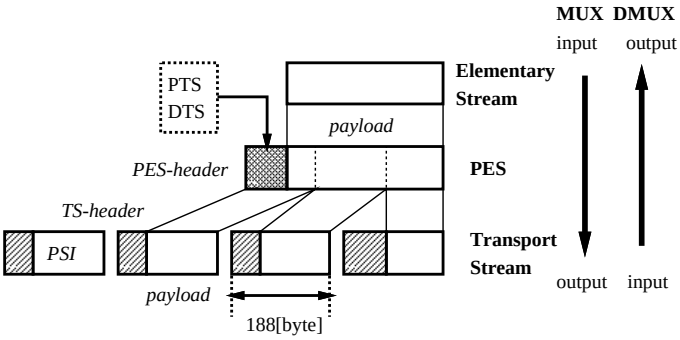
Structure of the paper. Section 2 describes the memory-based architecture for MPEG2 System protocol LSIs and the hardware and software interaction. Section 3 presents the concept and behavior of the polling-based real-time software for MPEG2 MUX and DMUX applications. Section 4 presents the results of an actual implementation and an evaluation of the polling-based real-time software implemented in the MUX and DMUX applications.

II. Memory-based Architecture

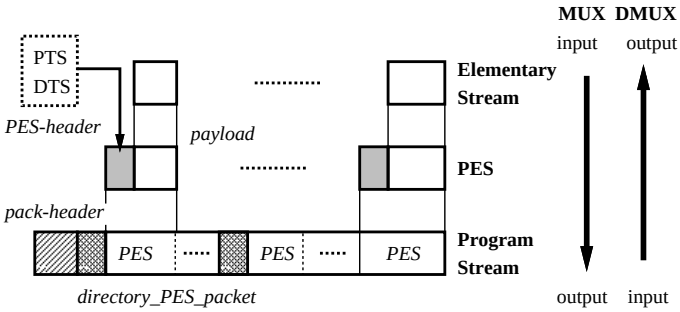
A. MPEG2 System Protocol Processing

Hierarchical Packetizing and Data Dependency: A hierarchical packetizing scheme is used in the MPEG2 System protocol. First, the Elementary Stream (ES) from the encoder is packetized to the Packetized Elementary Stream (PES). Next, the PES is further packetized to the TS or PS, depending on the application. As shown in Figures 1(a) and (b), the elementary stream from the encoder is written down in the PES payload (PES_packet_data_bytes), and the PES itself is written down in the TS or PS payload. Some values in the PES header are determined through the analysis of the elementary stream. Presentation Time Stamp (PTS) and Decoding Time Stamp (DTS) [3] which are used to synchronize the video with the audio, are examples of these values. The TS/PS syntax depends on the elementary stream, which complicates the packetization process.

Required Performance of MPEG2 System: The "MPEG2 Systems" standard does not specify the bit rates of video, audio and user data. In our design (MP@ML [1]), the maximum bit rates of video, audio and user data are respectively set to 15 Mbps, 384kbps, 192kbps, which are decided on the basis of the real-time use of MUX and DMUX. The hierarchical packetizing and the analysis of the elementary stream, which must be done in real-time, increase the complexity of the MUX/DMUX software.



(a) Composition of Transport Stream



(b) Composition of Program Stream

Fig. 1. Hierarchical Packetizing and Data Dependency

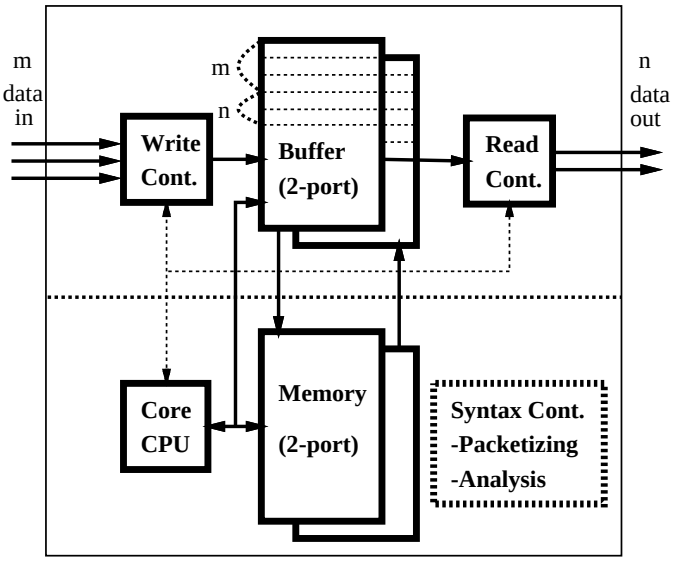


Fig. 2. Memory-based Architecture for MPEG2 Systems

B. Memory-based Architecture

The memory-based architecture [11] for LSIs implementing MPEG2 System Protocol consists of a core CPU, memories, and dedicated application-specific hardware as shown in Figure 2. It is designed and optimized by hardware/software co-design techniques. This architecture features the good performance of the hardware-oriented model and the high flexibility of the software-oriented model. Our MPEG2 MUX/DMUX LSIs based on this architecture provide sufficient performance and flexibility for real-time applications of the MPEG2 System protocol.

The main functions and features of our MPEG2 MUX/DMUX LSIs are listed in Table I. The MUX and DMUX LSIs support the full functionality of the MPEG2 System protocol. These LSIs were fabricated using a 0.5

TABLE I
FUNCTIONS AND FEATURES OF MUX/DMUX LSIs

	MUX	DMUX
Profile & Level	MP@ML	
Input Stream	Video/Audio/User	TS or PS
Output Stream	TS or PS	Video/Audio/User
Sync. of STC	PCR or SCR	
Sync. of Video/Audio	PTS and DTS	
Technology	0.5μm CMOS Embedded Gate Array	
Clock	27MHz	
Register Files Memories	32b×32 8kB Dual-port RAM	
Area Size	$14 \times 14mm^2$	
Power	1.85W @ 3.3V	
Package	304-lead CQFP	

μm CMOS Embedded Gate Array technology. A photograph of the MUX LSI is shown in Figure 3. The floorplan of the DMUX LSI is the same and the commonality rate between the MUX and DMUX reaches 60% in terms of chip area.

C. HW/SW Interactions

All of the interactions between hardware and software are modeled as shown in Figure 4. There are three kinds of interactions:

- Hardware to Software
  - External interrupts
  - Writing to registers or I/O ports
- Software to Hardware
  - Issuing special instructions
  - Writing to registers or I/O ports
- CPU to Software
  - Internal interrupts when software error occurs
  - Internal interrupts in the debug mode

Basically, a memory mapped I/O technique as almost all interactions between hardware and software is used for implementing MPEG2 MUX/DMUX real-time application. Interruptions are only used when a external interruption from a external port is needed in real-time.

III. POLLING-BASED REAL-TIME SOFTWARE

A. Real-time Application of MPEG2 MUX

The MPEG2 MUX multiplexes in real-time one or more elementary streams (ES) of video, audio and user data into one program and transmits them on the network as a transport stream. Moreover, Program Clock Reference (PCR), which is used for the time management of MPEG2 system, Program Specific Information (PSI), which is the environmental information of MPEG2 system, and Null packet, which carries no information, are also multiplexed on a transport stream.

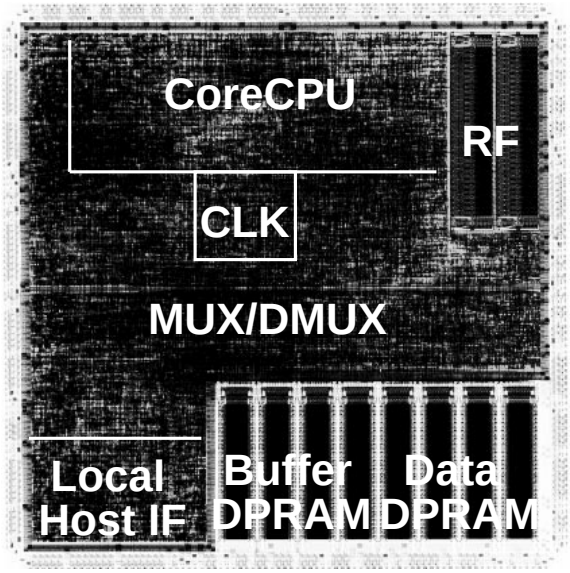


Fig. 3. Photograph of MUX LSI

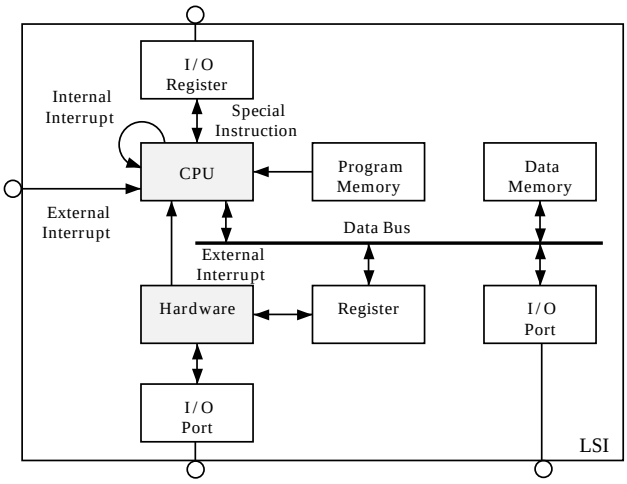


Fig. 4. Hardware/Software Interactions

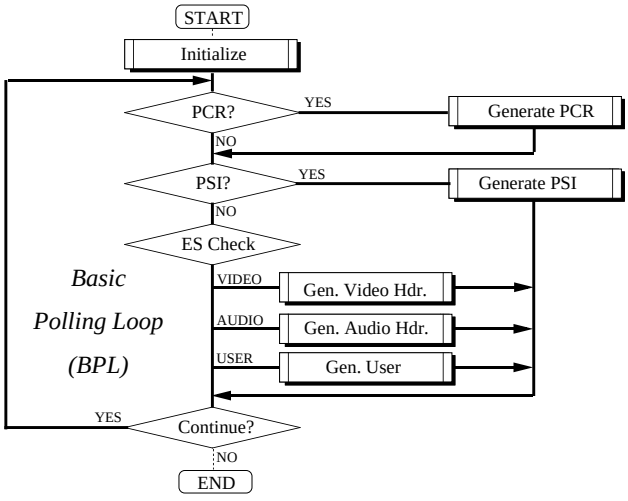


Fig. 5. Flowchart of MPEG2 Multiplexing for Transport Stream

The basic flowchart of MPEG2 Multiplexing for transport stream processing is shown in Figure 5. Each packet mixing on a transport stream has to keep each elementary stream's required bit rate; i.e. 5 Mbps for Video, 256 kbps for Audio, and 64 kbps for User, with PCR of more than 10 times per second, PSI of once per second, and a total transport stream rate of 6.144 Mbps. Therefore, each task (rectangles), which generates each packet, included in the basic polling loop (BPL) in Figure 5 should be scheduled and invoked in real-time in order to maintain each elementary stream's required bit rate.

B. Concept of Polling-based Real-time Software

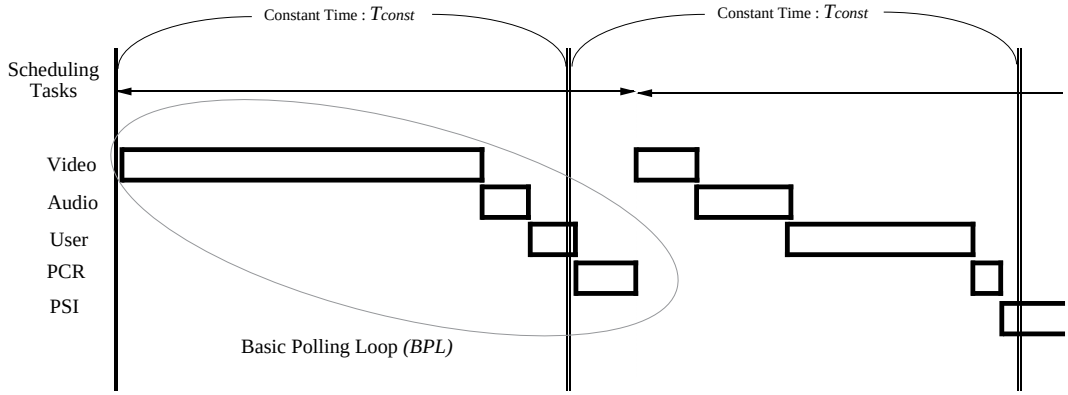
Each task in the Basic Polling Loop (BPL) in Figure 5, which is iteratively invoked and executed, requires variable execution time (execution cycles) to finish themselves. For example, the "Gen. Video Hdr" task generates the video header of the PES when the ES data is buffered enough as the next TS packet and the picture header data is searched for in the ES buffered. The number of execution cycles (CoreCPU cycles) vary from several to several hundred depending on the conditions of buffer and the results of header search. On the other hand, the number of possible cycles for one transport stream packet (188 bytes) processing is given from the total bit rate. For example, for a total transport stream rate of 15 Mbps (≈2MB/sec) and a system clock of 27 MHz, the number of possible execution cycles is about 2500 ($27 \times 10^6 \times 188/2 \times 10^6$).

To invoke each task in real-time, the original task is split into subtasks and the number of schedulings of each task is increased in one transport stream packet processing. The basic idea and the behavior of splitting an original task into subtasks are described in the Appendix. Each split subtask execution cycle can be modified and fitted to the suitable execution cycle that is less than the original task one.

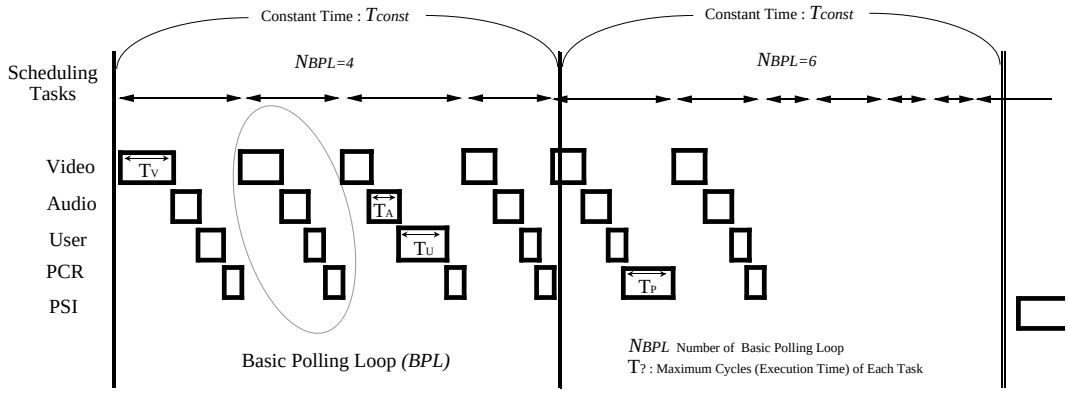
The concept of polling-based real-time software is show in Figure 6. The Polling-based real-time scheduler is design and optimized carefully in order to guarantee the following constraints:

- suppress the longest execution cycles of each task, i.e. video, audio, user, PCR, and PSI, by splitting each task into suitable subtasks, and
- increase and maintain the number of minimum schedulings for the basic polling loop (BPL) in one transport stream packet processing at more than a proper constant value. In other words, the maximum scheduling interval of the BPL (or each task) is kept at less than a proper constant time (cycles).

"Non-optimized and Failure Scheduling," which does not satisfy these constraints, and "Optimized and Successful Scheduling," which satisfies them, are show in Figures 6(a) and (b), respectively.



(a) Non-optimized and Failure Scheduling



(b) Optimized and Successful Scheduling

Fig. 6. Concept of Polling-based Real-time Software

C. Behavior of Polling-based Real-time Software

The behavior of polling-based real-time software in a MPEG2 multiplexing application is shown in Figure 7. The timing of the polling in one transport stream packet processing changes with every loop as depicted by the random interval spiral line in the Figure. In one transport stream packet processing (a constant interval), T_{const}, the basic polling loop (BPL) or each task is scheduled several times, N_{BPL}. The polling-based scheduler guarantees the minimum N_{BPL} will be more than a proper constant value ($Min.\ N_{BPL}$). Therefore, the maximum scheduling interval of the BPL (or each task) in T_{const} is given as dividing T_{const} by $Min.\ N_{BPL}$.

IV. Implementation and Evaluation Results

A. Implemented Polling-based Software

In implementing MPEG2 MUX/DMUX applications such as (1) MUX in the TS (mentioned as the previous section), (2) DMUX in the TS, and (3) MUX/DMUX in the PS, the proposed polling-based real-time software is used as the basic programming technique in our MPEG2 MUX/DMUX LSIs. The software is written in C-language. Table II shows the size of the C-language code for the BPL (Basic Polling Loop) of TS/PS and the common parts of MUX/DMUX. The coding size of each BPL is a small part of the total, but the BPLs work well and efficiently. The sophisticated BPL by considering polling-based scheduler makes it possible to invoke each task in real-time without any interruptions between hardware and software.

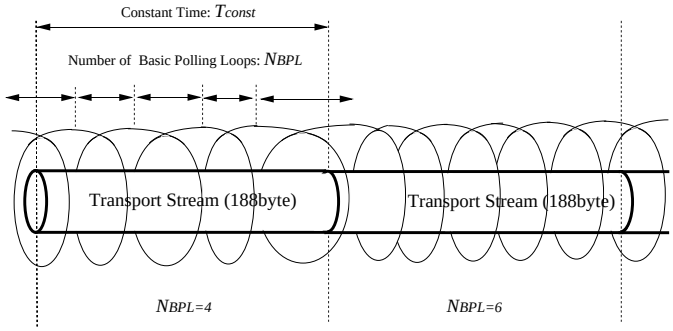


Fig. 7. Behavior of Polling-based Real-time Software

B. Validation and Evaluation Results

To validate and evaluate the polling-based real-time software, we developed MPEG2 CODEC Systems [16] [17] using our MPEG2 MUX/DMUX LSIs with the polling-based real-time software and extracted the real transport streams. The streams were then analyzed using an MPEG2 transport analyzer [18]. The each elementary stream's required bit rates are 5 Mbps for Video, 256 kbps for Audio, 64 kbps for User, and 6.144 Mbps of total transport stream. This parameter set is a digital CATV application in Japan [16].

The transition of output packets, which are generated by each scheduled task, in actual transport stream is shown in Figure 8. Each task such as video, audio, user, PCR, PSI is scheduled based on the polling-based scheduler within each elementary stream's required bit rate in real-time. The statistics on the transport stream are shown in Table III. The actual (extracted) bit rate of each packet in the transport stream agrees with that of the required bit rate specifications.

These results demonstrate the sufficient performance and usefulness of the polling-based real-time software in implementing the full functionality of the MPEG2 System protocol. The MUX/DMUX LSIs with their software are in use on MPEG2 CODEC systems [16] [17] for several multimedia communication and storage services.

TABLE II
Lines of Polling-based Real-time Software (C-Language)

	BPL(TS)	BPL(PS)	Common	Total
MUX	2.9k	1.6k	7.1k	11.6k
DMUX	1.5k	1.5k	6.3k	9.3k

TABLE III
Statistics on Transport Stream Processing ($\approx$ 4.5 sec.)

TS Packets	# of TS	(Percentage)	Req. Bit Rate
Video TS	15458	(83.59 %)	$\approx$ 5 Mbps
Audio TS	1225	(6.62 %)	$\approx$ 256 kbps
User TS	490	(2.65 %)	$\approx$ 64 kbps
Null TS	1265	(6.84 %)	—
PCR	46	(0.25 %)	—
PSI	8	(0.04 %)	—
Total TS	18492	(100.00 %)	= 6.144 Mbps

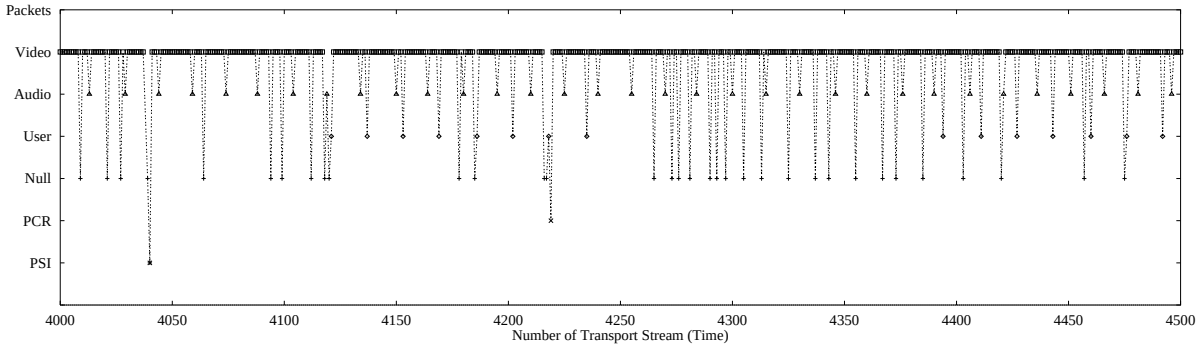


Fig. 8. Transition of Output Packets in Transport Stream Processing ($\approx$ 120 msec.)

V. Conclusion

This paper has proposed polling-based real-time software for MPEG2 System protocol LSIs which are a typical on-chip embedded and real-time systems. It demonstrates the performance and usefulness of the polling-based real-time software, which was designed and optimized by analyzing application specific function requirements and deciding scheduling intervals and the execution cycles of each task. The polling-based approach provides sufficient performance without any hardware and software overhead for a real-time application like the MPEG2 System protocol. The MUX/DMUX LSIs with their software are in use on MPEG2 CODEC systems [16] [17] for several multimedia communication and storage services. In the near future, we will study an automatic task splitting, and investigate ways to expand it to very light weight real-time kernels for an on-chip embedded and real-time system.

Acknowledgments

The authors would like to thank Dr. Osamu Karatsu of the NTT System Electronics Laboratories and Tamio Hoshino of the NTT Multimedia Service Promotion Headquarters for supporting this work. Thanks are also due to Takaaki Izuoka of the NTT Human Interface Laboratories, Minoru Inamori of the NTT Optical Network Systems Laboratories, and the members of the Advanced LSI Laboratory for useful discussions.

References

[1] *Video - Generic Coding of Moving Pictures and Associated Audio ISO/IEC 13818-2 International Standard.* 11, November, 1994.

[2] *Audio - Generic Coding of Moving Pictures and Associated Audio - ISO/IEC 13818-3 International Standard.* 11, November, 1994.

[3] *Systems - Generic Coding of Moving Pictures and Associated Audio - ISO/IEC 13818-1 International Standard.* 11, November, 1994.

[4] K. Ishihara et al. A Half-pel Precision MPEG2 Motion-Estimation Processor with Concurrent Three-Vector Search. *IEEE International Solid-State Circuits Conference digest of technical papers*, pp. 288–289, 1995.

[5] T. Kondo et al. A Two-Chip Realtime MPEG2 Video Encoder with wide range motion estimation. *Symposium Record HOT Chips VII*, pp. 95–101, 1995.

[6] T. Demura et al. A Single-Chip MPEG2 Video Decoder LSI. *IEEE International Solid-State Circuit Conference*, pp. 72–73, 1994.

[7] M. Toyokura et al. A Video DSP with a Macroblock-Level-Pipeline and a SIMD Type Vector-Pipeline Architecture for MPEG2 CODEC. *IEEE International Solid-State Circuit*, pp. 74–75, 1994.

[8] Texas Instruments. TMS320C3x Users Manual. 1994.

[9] Texas Instruments. TMS320AVX users Manual. 1993.

[10] C-Cube Microsystems. Data sheet of CL9110 - Transport Layer Demultiplexer. 1994.

[11] M. Inamori, J. Naganuma, H. Wakabayashi, and M. Endo. A Memory-based Architecture for MPEG2 System Protocol LSIs. *European Design & Test Conference*, 1996.

[12] Daniel D. Gaiski, Frank Vahid, Sanjiv Narayan, and Jie Gong. *Specification and Design of Embedded Systems.* Prentice Hall, 1994.

[13] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, Inc., 1990.

[14] M. Accetta et al. Mach: A New Kernel Foundation for UNIX Development. *Proc. Summer 1986 USENIX*, pp. 93–112, 1986.

[15] J. K. Ousterhout et al. The Sprite Network Operating System. *IEEE Computer*, Vol. 21, No. 2, pp. 23–36, 1988.

[16] N. Terada et al. A MPEG2-Based Digital CATV and VOD System using ATM-PON Architecture. *Proc. of IEEE MULTIMEDIA'96 Conference*, pp. 522–531, 1996.

[17] Yutaka Tashiro et al. MPEG2 Video and Audio CODEC Board Set for a Personal Computer. *IEEE Global Telecommunications Conference*, Vol. 1, pp. 483–487, 1995.

[18] *Software MUX/DMUX System (Ver 1.0).* NTT Internal Reports, 1994.

Appendix: Splitting Task into Subtasks

The basic idea and its behavior of splitting original task into subtasks are described here.

An original task can be split into subtasks as shown in Figure 9. The variable "part_flag_n" has to be controlled exclusively. The variables between each part has to maintain globally.

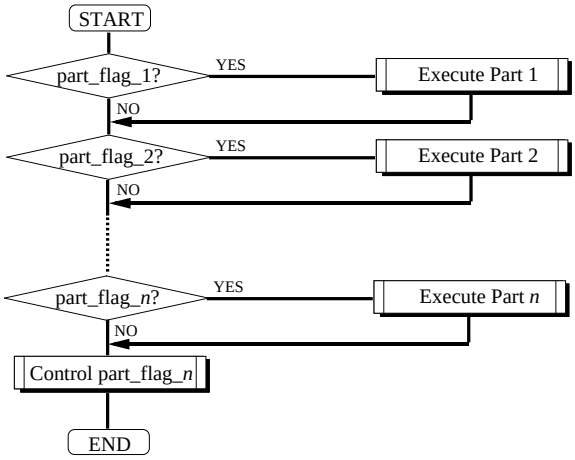


Fig. 9. Basic Idea of Splitting Task into Subtasks

Each split subtask execution cycle can be modified and fitted to the suitable execution cycle that is less than the original task one as shown in Figure 10.

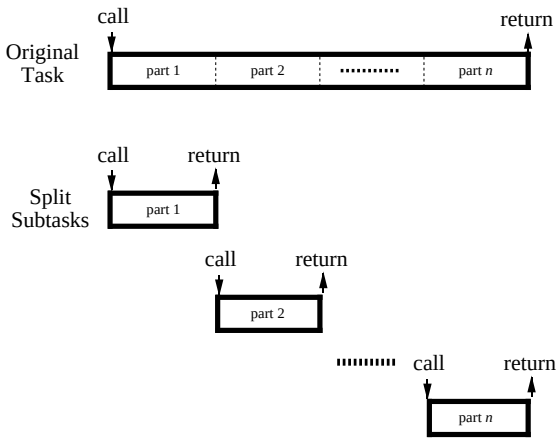


Fig. 10. Subtask Execution Behavior