Abstract — This paper presents the design and implementation of a software (SW)-based receiver running on a single digital signal processor (DSP) for terrestrial digital multimedia broadcasting (TDMB). Implementing the functions of the physical layer such as synchronization and channel decoding as well as codec decoders for a TDMB receiver in software running a DSP raises numerous design and implementation issues. We present solutions to these issues including an efficient software receiver structure, efficient algorithms with low complexity, real-time scheduling, reduction of the execution cycle number, and data transfer and communication between function blocks in the receiver, with a focus on the physical layer. The SW-based TDMB receiver developed in the present work can change and upgrade its functions and adapt to new standards. The performance of the implemented receiver was verified through successful decoding of live audio and video signals.


I. INTRODUCTION

The digital audio broadcasting (DAB) system developed by the Eureka 147 project has been in operation in many European countries since 1995. In 2005, Terrestrial Digital Multimedia Broadcasting (TDMB) [1], based on the Eureka-147 standards [2], was launched in Korea and mobile TV service using TDMB has recently become popular in Korea and several other countries. DAB and TDMB systems use orthogonal frequency division multiplexing (OFDM) as a transmission technique on the merits of its robustness in multipath fading channels and bandwidth efficiency [3].

To date, diverse types of receivers in hardware and in software for DAB and TDMB have been developed [4]-[10]. Some receivers have implemented the physical (PHY) layer functions such as synchronization, OFDM demodulation, and channel decoding as well as codec decoders in hardware and/or on digital signal processors (DSPs) [4]-[7]. In one work, the PHY layer of a DAB receiver was implemented on a single DSP [8]. Personal computer (PC)-based DAB receivers have been developed, where the PHY layer or various application functions have been implemented in software running on a PC [9]-[10].

Many commercial TDMB receivers, which are standalone or running on mobile phones, portable media players (PMPs), or car navigation systems, are implemented using various types of chipsets under considerations of cost, size, and power consumption. A typical chipset is composed of a tuner chip, a baseband chip, and codec chips. However, it is difficult to change or upgrade the functions of TDMB receivers based on chipsets. Many PC-based TDMB receivers also use a tuner chip and a baseband chip, and codec decoding is performed on a PC. Hence, it is also difficult to change and upgrade the PHY functions of PC-based TDMB receivers.

In this work, the design and implementation of a software (SW)-based TDMB receiver running on a single DSP is presented, focusing on the PHY layer. We present solutions to a number of design and implementation issues that are raised when implementing a SW-based PHY layer running on a DSP, including the following: the need for an efficient SW-based receiver structure, efficient algorithms with low complexity, real-time scheduling, and reduction of the execution cycle number, as well as data transfer and communication problems between various function blocks in the receiver.

A brief description of the TDMB system is given in Section II. Section III describes the design of the proposed S/W-based TDMB receiver, focusing on the PHY layer. Details of the PHY layer implementation in the SW-based TDMB receiver are given in Section IV. Section V presents the performance of the SW-based TDMB receiver. Finally, conclusions are presented in Section VI.

II. TDMB SYSTEM DESCRIPTION

A. Overview of the TDMB System

The TDMB system is based on the Eureka-147 standards [2] and transmits video signals using MPEG-2 transport stream (TS) packets on the DAB stream mode data channel. Thus, the TDMB system can simultaneously provide audio, video, and data services to users moving at speeds of up to 200 Km/h. The audio service called MUSICAM makes it possible for TDMB users to listen to CD-quality stereo audio signal. The video service provides users with mobile TV service with the quality of BER of $10^{-8}$ on a seven inch liquid crystal display (LCD) screen. Such high quality video service becomes possible by applying the powerful Reed Solomon (RS) coding technique to the video signal. The data service includes a broadcasting web site (BWS) service and a transport protocol expert group (TPEG) service. Internet services are possible through the BWS service and traffic information can be received using the TPEG service on


1 This work was supported in part by the IT R&D program of MKE/IITA under Contract No. 2005-S-609-04, which is the International Joint Project of Texas Instruments and KAIST, in Korea.

M. Lee, B. Keum, J. H. Jeong, Y. S. Shim, and H. S. Lee are with the MMPC and the School of EEC, KAIST, Daejeon 305-701, Korea (e-mail: wildgoosemb@mmpc.kaist.ac.kr).
TDMB receivers. More information on the TDMB service can be found in [11]-[12].

To decode the contents of the aforementioned multimedia service, the following codec decoders have been developed in our center and have been used in the development of the SW-based TDMB receiver. However, the details of the developed decoders are not given here, as the focus of this paper is on the design and implementation of the PHY layer.

- H.264 SP decoder for video content
- MPEG-4 ER BSAC decoder for audio content
- MUSICAM decoder for MPEG audio layer II content
- BWS and TPEG decoders for data content

The TDMB system uses the band III (174 MHz ~ 216 MHz) and L-band (1,452 MHz ~ 1,492 MHz) as transmission frequency bands. These bands are divided into several sub-bands with 6 MHz bandwidth. Each 6 MHz sub-band is composed of three multiplex channels, and each occupies 2 MHz bandwidth (the actual bandwidth is 1.536 MHz). The TDMB system broadcasts audio, video, and data signals using transmission frames based on OFDM in each multiplex channel. The TDMB transmission frame has three different physical channels. The first channel called the synchronization channel is used to carry a Null symbol and a phase reference symbol (PRS), which are used for synchronization at the receiver. The second channel is a control channel called fast information channel (FIC) that uses three OFDM symbols. It carries control and system information such as channel multiplex configuration. The last channel, called the main service channel (MSC), is composed of four common interleaved frames (CIFs). One CIF, which corresponds to a source information interval of 24 ms, contains audio, video, and data service channels and is composed of 18 OFDM symbols. According to the channel multiplex configuration by the service provider, part or all of audio, video, and data channels can be transmitted. Furthermore, the TDMB system supports three transmission modes, modes I, II, and IV, as a function of the system operating conditions and each transmission mode provides identical service using different physical layer parameters [2]. Details of the TDMB transmission structure are found in [1]-[2].

B. Structure of the Conventional TDMB Receivers

The structure of the conventional TDMB receivers based on chipsets or a combination of chipsets and a PC is shown in Fig. 1. The tuner and the analog-to-digital converter (ADC) are normally implemented in hardware (H/W) chips. However, the PHY layer and the audio and video (A/V) decoders can be implemented using chips or a combination of chips and software running on a PC depending on the receiver structure. A separate controller or PC is used to control function blocks in the receiver. In addition to the conventional DAB channels such as a MUSICAM channel and a data channel called the program associated data (PAD) channel, the TDMB system has a video channel and a dedicated data channel called the non-PAD (NPAD) channel. Hence, the TDMB receiver needs additional function blocks (i.e., RS decoder, TS parser, NPAD decoder, A/V decoders) to decode video and NPAD channels compared to the DAB receiver part in Fig. 1. If the PHY layer and/or A/V decoders in a TDMB receiver are implemented using chipsets, it will be difficult to change or upgrade their functions.

III. DESIGN OF THE SW-BASED TDMB RECEIVER

A. Proposed SW-based TDMB Receiver Structure

To implement all the functions of the TDMB receiver in software running on a single DSP except for the tuner and the ADC, we propose the SW-based TDMB receiver structure illustrated in Fig. 2. The DSP is composed of a DSP core and an ARM core. The PHY layer, the A/V decoders, and the MUSICAM decoder are implemented on the DSP core. All the remaining functions including a system controller and drivers supporting a LCD screen, a speaker, and a graphic user interface (GUI) in a TDMB receiver are implemented on the ARM core, as shown in Fig. 2.

B. Design and Implementation Issues

The implementation of the SW-based TDMB receiver running a single DSP raises numerous design and implementation issues. Among them, several important issues that affect the receiver’s performance are listed below.

- Issues related to a target DSP chip
  - Clock speeds of DSP and ARM cores in a DSP
  - Number of multiple functional units in the DSP core
  - Size of on-chip memory and use of cache memory
- Use of direct memory access (DMA) to carry data from peripherals to off-chip memory
- Operating system (OS) to manage resources and real-time scheduling

- Issues related to receiver structure and its complexity
  - ADC sampling rate and sample resolution
  - Channel filtering and its complexity on a DSP
  - Type and complexity of I/Q demodulation
  - Complexity of synchronization algorithms
  - Frequency compensation scheme
  - Efficient algorithms for various functional blocks in terms of complexity and code size

- Issues related to real-time processing
  - Efficient software structure design for the PHY layer
  - Real-time scheduling using multiple threads
  - Efficient data transfer methods among ADC, DSP core, and ARM core
  - Efficient communication methods to transfer control and state signals between DSP and ARM cores

- Issues to reduce execution cycles on the DSP
  - Efficient utilization of on-chip memory by using stack memory and software interrupts
  - Code optimization for critical function blocks using SW pipelining, unrolling loops, and word-wide data.

To implement the SW-based TDMB receiver shown in Fig. 3, we used a DaVinci DSP with a DSP core clock speed of 513 MHz [13] as a target DSP that addresses the aforementioned DSP-related issues.

C. The Design of the PHY Layer

A block diagram of the PHY layer used in the SW-based TDMB receiver is shown in Fig. 3. When a small TDMB signal enters the receiver via a receive antenna, the tuner amplifies the small input signal and performs necessary frequency downconversion to an intermediate frequency (IF) or a low IF depending on the receiver structure. The ADC converts the analog IF signal to a digital signal and the digital signal is stored in the input buffer. The PHY control block, which communicates with the system controller on the ARM side, determines when and which service channel is decoded. The mode detection block finds the transmission mode type of the input signal.

The initial synchronization, which is composed of frame and symbol synchronization and integral and fractional frequency synchronization, finds a reference time that indicates the start point of data decoding. It also estimates a frequency offset (FOS) generated by the carrier frequency difference between the transmitter and the receiver. The time tracking estimates the time offset for every succeeding frame, which is caused by the sampling clock difference between the transmitter and the receiver, in order to find the exact start point of data decoding. After I/Q demodulation, frequency tracking is performed to estimate the FOS change for every OFDM symbol in each frame, which is generated by frequency stability of the local oscillator in the receiver.

Frequency compensation is then performed using FOSs obtained by the frequency tracking and the initial synchronization. OFDM demodulation using a fast Fourier transform (FFT), differential demodulation, and frequency deinterleaving are subsequently performed for the frequency compensated input data. The QPSK demapper produces soft decision values to be used in the Viterbi decoding.

From the tuner to the QPSK demapper, all function blocks are commonly used for the FIC channel and service channels. However, from time interleaving to RS decoding, different function blocks are necessary for different channels. In the case of a FIC channel that should carry control information as fast as possible, time deinterleaving prior to depuncturing is omitted, because it causes a time delay of 16 CIFs. The audio, data, and video channels need the same blocks of time deinterleaving, depuncturing, Viterbi decoding, and energy dispersal. However, the audio channel needs depuncturing with an unequal error protection level, whereas the data and video channels use an equal error protection level in depuncturing. The video channel needs convolutional deinterleaving and Reed Solomon (RS) decoding after the energy dispersal block. Even though audio, video, and data channels in a CIF carry data of different size, the PHY layer has been designed to share the depuncturing, Viterbi decoding, and energy dispersal blocks for different channel decoding in order to reduce the code size. But a different set of parameters corresponding to each channel is applied to the three common blocks when a channel is processed.

To implement the PHY layer in software running on a DSP, which meets the real-time deadline, we paid special attention to the design of the following critical blocks in the PHY layer with the aim of reducing the computational complexity.

C.1 Synchronization

Before the data decoding begins, the receiver always should be synchronized to the received signal. Furthermore, the state of synchronization should be maintained even at about 5 dB below the sensitivity of the receiver for fast recovery after the receiver escapes deep fading or shadowing. Hence, the synchronization performance is critical to maintain the receiver’s performance under a changing wireless channel.

Fig. 3. Block diagram of the PHY layer in the SW-based TDMB receiver running on a single DSP.
condition. The complexity of the synchronization algorithm is also important in terms of implementation. Therefore, synchronization is divided into two parts to meet the two goals: the initial synchronization and tracking. In the case of the initial synchronization, accuracy is more important than complexity, because it is only performed once after power turn-on of the receiver or the receiver’s escape from deep shadowing or fading. On the other hand, the complexity of the tracking algorithm should be as low as possible, because time and frequency tracking should be performed for every frame. The reference time and an integral FOS, which are obtained by the initial synchronization, are used for the time and frequency tracking as well as data decoding for succeeding frames.

The frame synchronization for the initial synchronization estimates the time offset $\Delta T$ within $\pm 10$ samples [9] by taking the power ratio of two energy windows for the received input signal as in [5]. The symbol synchronization finds the time offset $\Delta t$ within $\pm 1$ samples using the scheme in [9]. The reason for using this scheme in the initial synchronization is that even though its complexity is higher than that of the channel impulse response (CIR) method [5], it is not sensitive to frequency offsets. The fractional frequency synchronization attempts to estimate a fine FOS $\Delta f$ within $\pm 1$% of subcarrier spacing using the method described in [14]. The integral frequency synchronization estimates a rough FOS $\Delta F$ within $\pm 0.5$ times of the subcarrier spacing using the scheme in [15]. This partial correlation scheme is employed due to its robustness to time offsets.

For the time tracking, the CIR method [5] is used to estimate the time offset $\Delta t$ for each frame. In the case of frequency tracking, the fractional frequency synchronization scheme [14] that is used in the initial synchronization is again used. However, in this case, the received data OFDM symbol is applied to that scheme instead of the received PRS, because frequency tracking should be performed for every data symbol in a frame.

### 2.2 Channel Filtering and I/Q Demodulation

A channel filter in the receiver selects a desired channel and rejects unwanted channels to prevent the receiver from being saturated by large adjacent channel signals. If the channel filter for the TDMB receiver is implemented on a fixed-point DSP, huge computation power is necessary due to the filter’s sharp rejection characteristics [16]. Therefore, we do not implement the channel filter as a part of the PHY layer on the DSP, but instead have chosen a tuner that has a built-in channel filter with desired rejection characteristics.

If the I/Q demodulation is performed in the process of direct frequency conversion in the tuner, the DSP should receive two independent channel data with an I/Q imbalance problem from two ADCs. Hence, we use an I/Q demodulation scheme based on 4 times oversampling [17]. With this scheme, it is not necessary to multiply input data by sine and cosine values, and the I/Q imbalance problem does not occur due to exact phase control between sine and cosine values. Considering 4 times oversampling, an ADC sampling rate of 8.192 MHz was chosen. To reduce the buffer memory size and the execution cycle number on the DSP, the ADC samples were represented by 8 bits. Such low resolution of the ADC sample is allowed because the channel filter is not implemented on the DSP.

### 3.3 Frequency Compensation

The performance of OFDM-based receivers is sensitive to FOS because it causes intercarrier interference [15]. Therefore, precise FOS compensation as well as accurate FOS estimation is necessary. In the case of TDMB receivers based on chipsets, the estimated FOS value is sent to the local oscillator in the tuner via an external interface to compensate for FOS [5]. Data decoding begins after FOS compensation by the tuner. If the same FOS compensation scheme is applied to the SW-based receiver, the DSP or ARM has to send the estimated FOS value to the local oscillator for every OFDM symbol, whose length is 0.623 ms for transmission mode IV. When the DSP sends the estimated FOS value via a serial interface such as I2C, it has to be interrupted frequently, resulting in damage to SW pipelining used for code optimization. If the estimated FOS value is sent to the local oscillator by the ARM, accurate frequency compensation may not be guaranteed due to a time delay and the ARM being interrupted frequently. As a result, the aforementioned FOS compensation scheme is not suitable for the proposed SW-based receiver.

The SW-based TDMB receiver first stores all input data samples in the input buffer and decodes them later. Thus, FOS compensation should be independently performed for every symbol data in each frame stored in the buffer. The FOS compensated OFDM symbol data $z_{\nu}(n)$ is obtained by

$$z_{\nu}(n) = z_{\nu}(n)e^{-j2\pi(\Delta F+\Delta f)n/N}. \tag{1}$$

In (1), $z_{\nu}(n)$ is OFDM symbol input data with a FOS and $n$ is a time index. $N$ is a FFT size, and $\Delta F$ and $\Delta f$ are the integral and fine FOSs, respectively. According to (1), the different phase offset $e^{j2\pi(\Delta F+\Delta f)n/N}$ should be multiplied to $z_{\nu}(n)$ for each data sample within an OFDM data symbol, because the time index $n$ increases up to $N$. If the phase offset $e^{j2\pi(\Delta F+\Delta f)n/N}$ is compensated using sine and cosine tables in the SW-based TDMB receiver, large $\Delta F + \Delta f$ makes $2\pi(\Delta F+\Delta f)n/N$ values change rapidly over the sine and cosine tables as the time index $n$ increases. This causes cache misses to occur frequently during frequency compensation, resulting in increased execution cycle number for frequency compensation. Therefore, in this work, only $\Delta f$ is compensated using (1) and sine and cosine tables. On the other hand, $\Delta F$ is compensated by simply shifting the location of subcarriers by $\pm \Delta F$ directly after the FFT for the OFDM demodulation.

### 3.4 Channel Decoding

In this work, the Viterbi decoding algorithm using soft-decision value is used to decode the convolutionally encoded TDMB input signal, as it is known to be optimal [18]. In addition, the conventional algebraic RS decoding scheme with the following steps [19] is used to decode the received video
signal that is encoded by the RS code (204, 188, 8) for burst-error correction:

- Computation of the syndromes
- Determination of an error/erasure locator polynomial whose roots indicate locations of the errors and erasures (the Berlekamp-Massey algorithm was used in this work)
- Determination of the error and erasure values (Forney’s algorithm was used in this work)

IV. Implementation of the SW-based TDMB Receiver

A. Multithreaded Real-Time Scheduling

To process the TDMB signal in real-time, input data buffering by the DMA and data processing by the DSP should be performed at the same time. At least several input data frames should be stored in the input buffer by the slow DMA in advance before the fast DSP processes them. Therefore, depending on the availability of input data, the DSP should operate in different modes, that is, a processing mode if the input data is available or a standby mode if not. In addition, external events such as a request of service channel change from the GUI and resynchronization requests due to poor wireless channel conditions should be handled properly. Thus, to meet a real-time timing deadline in such a complicated environment, we adapted a multithreaded real-time scheduling scheme based on software interrupts in implementing the SW-based TDMB receiver. To manage real-time scheduling, we employed the DSP/BIOS, a small OS provided by the DSP chipmaker [20].

To implement the PHY layer in software running on a DSP, related source programs for the PHY layer are grouped into four functions called threads [20]-[21], each performing a specific task, and a priority is assigned to each thread. The real-time scheduling is then programmed to allow higher-priority threads to preempt lower-priority threads so that real-time data decoding is performed in response to external events. While hardware interrupts (HWIs) trigger HWI functions, they can damage SW pipeline loops (SPLOOPs) used for code optimization. Thus, HWI functions were not used in the PHY layer implementation. Tasks (TSKs) use their own private stacks and wait during execution until necessary resources are available. Thus, the use of many TSKs prevents critical codes or data from using the fast on-chip memory. Because of this property, in this work, TSKs are only used to implement codec threads that do not have strict real-time deadlines. On the other hand, software interrupts (SWIs) occur by just calling SWI functions in the program and SWI threads run to completion unless it interrupted by a hardware interrupt or preempted by a higher priority SWI. In addition, SWIs can have different priority levels. As a result, SWI threads can be programmed not to damage SPLOOPs. Furthermore, SWIs automatically perform context switching (i.e., storage/retrieval of the DSP status to the time the interrupt occurred) in a single system stack. Hence, if the system stack is placed on a fast on-chip memory, data decoding speed can be faster. Therefore, we implemented the PHY layer using several SWIs with proper size and priority in order to avoid damage of SPLOOPs and to quickly respond to external requests. To process input data frame-by-frame, a periodic (PRD) function is used to generate a series of periodic pulses whose period is 96 ms, a length of a data frame.

B. Real-Time Signal Processing

The four grouped functions in the TDMB PHY layer are PHY control, synchronization, symbol decoding, and channel decoding, as shown in Fig. 4. First, the PHY control function is activated by a start command from the system controller and it interprets the message about which channel to decode. It also checks the input buffer for data availability. If input data is ready, the initial synchronization function starts to perform the time and frequency synchronization process by the start request of the PHY control. At the steady state, time synchronization is performed to estimate the time offset for each frame. The symbol decoding function is then executed for the FIC channel or a service channel according to a request from the PHY control. In the case of a service channel, the symbol decoding process is performed for OFDM symbols in four CIF blocks, which belong to the requested service channel. For the FIC channel, the first three OFDM symbols within each frame are decoded. The channel decoding performs all or part of various function blocks in Fig. 4 in response to the service channel request. After channel decoding, the processed data is stored in the output buffer.

The aforementioned four functions are mapped to four threads whose names are PRD_frametrigger, Syntrack_SWI, Symbol_SWI, and Decoding_SWI, respectively, as listed in Table I, where Codec_TSK is also listed. The four threads are all implemented in SWIs and their priorities and properties are shown in Table I.

An execution graph of various threads to process data frames in the SW-based TDMB receiver is presented in Fig. 5. When the PHY layer is turned on by the system controller on the ARM, the initialization process of the PHY layer is performed by the main() function, which is assigned to the idle task. A series of periodic pulses, which are generated by counting ticks (a tick is a fourth of the DSP’s main clock in

![Fig. 4. Software structure of the PHY layer in the SW-based TDMB receiver.](image-url)
the codec processing is done. At the next periodic pulse, the PRD-frametrigger is triggered and the next frame is processed in the same manner as that described above.

C. Software Interface among DSP, ARM, and a Tuner

The input data, which is generated by the ADC at a given sampling rate, is transferred to the input buffer on the off-chip memory by the DMA on the ARM side, as shown in Fig. 6. The input buffer operates in a circular mode to support data writing by the DMA and data reading by DSP at the same time. When the DSP completes the requested channel decoding, the decoded data is stored in an output buffer called a common memory section (CMS). Different CMSs operating in the circular mode have been assigned for the storage and retrieval of different decoded channel data and for communication between the DSP and the ARM depicted in Fig. 6. Several other CMSs operating in a circular mode also have been assigned for codec decoding data.

D. Code Optimization

Besides code optimization techniques such as hand-coded SW pipelining, unrolling loops, and word-wide data [21]-[22], the following implementation techniques are used to reduce the execution cycle number for the Viterbi decoding with the highest complexity in the PHY layer.

To reduce the execution cycle number necessary to compute the branch metric, the following simplified Euclidean distance was used. Since the squared Euclidean distance between the $i$th received QPSK symbol $z_i$ and a reference QPSK symbol $s_j$ is $|z_i - s_j|^2$, the corresponding simplified Euclidean distance can be $dist(z_i | s_j) = z_i.x - z_j.x$, where $z_i = z_{i,x} + jz_{i,y}$ and $s_j = 1 + j1$. It is based on the fact that the terms $z_{i,x}^2 + z_{i,y}^2 + 2$ and the factor 2 are common in the squared Euclidean distance calculation for all QPSK symbols. Likewise, $dist(z_i | s_j) = z_{i,x} + z_{i,y}$, $dist(z_i | s_j) = z_{i,x} - z_{j,x}$, and $dist(z_i | s_j) = z_{i,x} + z_{i,y}$, where $s_j = 1 - j1$, $s_j = -1 + j1$, $s_j = -1 - j1$. The reference symbols $s_1$, $s_2$, $s_3$, and $s_4$ are assumed to be

---

**TABLE I**

<table>
<thead>
<tr>
<th>Thread name</th>
<th>Priority</th>
<th>Posting thread</th>
<th>Posted thread</th>
<th>Mapped functions</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRD_frametrigger</td>
<td>10</td>
<td>periodic pulses (96ms)</td>
<td>Synttrack_SWI</td>
<td>PHY_control</td>
<td>Periodic invoking of PHY control</td>
</tr>
<tr>
<td>Symbol_SWI</td>
<td>7</td>
<td>PRD_frametrigger</td>
<td>Symbol_SWI</td>
<td>Synchronization</td>
<td></td>
</tr>
<tr>
<td>Syntrack_SWI</td>
<td>8</td>
<td>Sync</td>
<td>Decoding_SWI</td>
<td>Symbol decoding</td>
<td>Time &amp; Freq. synchronization</td>
</tr>
<tr>
<td>Decoding_SWI</td>
<td>9</td>
<td>Symbol_SWI</td>
<td>None</td>
<td>Channel decoding</td>
<td>Channel decoding</td>
</tr>
<tr>
<td>Codec_TSK</td>
<td>3</td>
<td>None</td>
<td>None</td>
<td>Codec decoding</td>
<td>Video, Audio, MUSICAM decoding</td>
</tr>
</tbody>
</table>

**Fig. 5.** Execution graph of various threads to process data frames in the SW-based TDMB receiver.

**Fig. 6.** Software interface among DSP core, ARM core, and a tuner in the SW based TDMB receiver.
mapped to the bit sequence 00, 01, 10, 11. Hence, a branch metric corresponding to the codeword 0010 in the Viterbi decoding (the mother code rate is 1/4) for the TDMB receiver can be obtained by simply adding the two simplified Euclidean distances, as in \( \text{dist}(z_i | s_j) + \text{dist}(z_{i+1} | s_j) = z_{i,x} \cdot z_{i,y} + z_{i+1,x} \cdot z_{i+1,y} \). The rationale for this is the squared Euclidean minimum distance corresponding to a 4-bit codeword that is mapped to two independent QPSK symbols is a sum of two squared Euclidean minimum distances, which corresponds to each QPSK symbol, respectively.

To reduce the necessary memory space in the process of the ACS, only two arrays (AS1, AS2) with 64 short type elements are reserved to store 64 path metrics at the \( j \)th stage and the \( (j+1) \)th stage, respectively. The trellis for the Viterbi decoding in the TDMB receiver needs 64 states due to the constraint length \( K \) of the mother code used. Two different states (i.e., the \( j \)th and the \( m \)th states) at the \( j \)th stage always go to one state (i.e., the \( p \)th state) at the \( (j+1) \)th stage. Hence, the process of computing a branch metric and adding the computed branch metric to the related path metric stored in one array (AS1) can be performed for the \( j \)th and the \( m \)th states at the same time using multiple functional units in the DSP. The two accumulated path metrics are compared and the accumulated path metric with a smaller value by the minimum distance concept is stored in the \( p \)th element of the other array (AS2), which corresponds to the \( p \)th state at the \( (j+1) \)th stage. With this parallel operation, we could also considerably reduce the number of execution cycles necessary to load and save in-process data into the off-chip memory.

To store candidate bits of 0 or 1 for a desired codeword sequence at 64 states of the \( j \)th stage, where \( j = 1, 2, \ldots, 64 \) (decoding depth \( D \)), a history table composed of two arrays (AT1 and AT2) with 64 integer type (32 bits) elements was made. Once the history table is filled with candidate bits by the decoding process and the surviving path with the minimum accumulated path metric is decided, the traceback operation outputs the first codeword sequence of 32 bits from the array AT1 of the history table, which is filled first. The array AT1 is again filled with candidate bits by the succeeding decoding process. The second codeword sequence of 32 bits is then outputted from the array AT2 of the history table. Using this traceback scheme, considerable shift operation can be saved compared to the traceback scheme where the codeword sequence is outputted bit by bit. The reliability of the output codeword sequence is guaranteed by the characteristic that the decoding depth \( D \) satisfying the condition \( D \geq 5 \times (K-1) \) is sufficient to obtain almost optimum decoding performance of the Viterbi decoding algorithm [23].

To reduce the execution cycle number for the RS decoding, several code optimization techniques are used. Since the RS decoding process is performed byte by byte, we packed four bytes in a 32 bit register and used a special instruction to process four bytes at one time. To reduce the execution cycle number that is necessary in the process of load and store, data are processed in double word (8 bytes). In addition, we optimized a find root function, which is used to find roots of the error/erasure locator polynomial and takes considerable execution cycles, using the hand assembly technique.

### V. Performance of the SW-based TDMB Receiver

We have designed and implemented a SW-based TDMB receiver running on a single DSP using the aforementioned algorithms and optimization techniques. In order to verify its performance, we have built an evaluation board with the target DSP chip on which the implemented SW-based TDMB receiver is running. A photograph of the evaluation board and chip is shown in Fig. 7. A commercial antenna and a tuner were mounted on the evaluation board to receive the TDMB signal. The SW-based receiver on the DSP receives the 8-bits output of the ADC and video and data channel content are displayed on a seven-inch LCD screen on the evaluation board. The LCD screen is also used as a GUI to scan and select service channels. The evaluation board has an external speaker for audio channel content. Fig. 7 also shows a commercial TDMB generator to transmit the TDMB signal and a notebook to debug the receiver software and to monitor the status of the receiver.

#### A. Execution Cycle Performance

To measure how many execution cycles are required to run the SW-based TDMB receiver, a CLK_gettime() function [21] was used as it can measure the tick count difference between two points in the program while the DSP is running.

The execution cycle number in mega cycles per second (MCPS) is represented by

\[
MCPS = \frac{\text{Tick count difference}}{\text{Frame duration}} \times 10^4. \tag{2}
\]
In (2), the TDMB frame duration is 0.096 second. For MCPS measurement, a video signal with a rate of 544 kbps was transmitted by the TDMB generator at a carrier frequency of 201.008 MHz.

Table II shows the MCPSs for various function blocks in the PHY layer. As expected, the MCPS of the Viterbi decoding is the highest even if many optimization techniques were applied to reduce execution cycles. Since optimization techniques can not be applied to time deinterleaving, its MCPS is very high compared to those of other function blocks that were optimized to some degree. RS decoding has 10.5 MCPS, which is small compared to that of Viterbi decoding. The MCPS (14.6) of the FIC channel is very low compared to that (222.4) of the MSC channel, because the FIC channel is composed of only three OFDM symbols. The total MCPS of the PHY layer per frame at the steady state is 237. As expected, the initial synchronization has a high MCPS of 72.2. However, this is acceptable because no other function blocks in the PHY layer run while it is being executed.

Table III shows the MCPSs for the PHY layer and codec decoders, which are all running on the DSP core. If a video channel is selected, the PHY layer first runs and the H.264 and BSAC decoders are then executed for each frame within 96 ms. Hence, the total MCPS that the DSP core needs to support the PHY layer and two codec decoders is 410, which is smaller than the execution speed of the target DSP core (513 MHz). Accordingly, the implemented SW-based TDMB receiver will successfully decode the video signal with a rate of 544 Kbps. When an audio channel is selected, the PHY layer and the MUSICAM decoder are executed sequentially and the required MCPS is 247.

### B. Bit Error Rate Performance

To verify the bit rate error (BER) performance of the SW-based TDMB receiver, the TDMB signal generator was used to transmit all-zero data sequence. The starting input power level was -105 dBm over a channel bandwidth of 1.536 MHz and the measured noise figure (NF) of the tuner was 5 dB. The input power level was increased to change the carrier to noise ratio (CNR). Test bits of $2 \times 10^6$ were transmitted at each CNR to obtain reliable BER values.

Measured BER performance versus CNR of the SW-based TDMB receiver operating in transmission mode I has been plotted in Fig. 8. For comparison, simulated BERs versus CNR at the same conditions were plotted as well. The BER of $10^{-4}$ after the Viterbi decoder was obtained at a CNR of 7.7 dB, which is about 1.2 dB worse than the simulated value. This is due to the NF increase by the automatic gain control in the tuner as the input power increases. A BER of $10^{-8}$ after the RS decoder was obtained at a CNR of 6.2 dB, which is about 0.2 dB worse than the simulated value. We could also confirm that the receiver offers good decoding capability for the audio, data, and video channels on the LCD screen and through the speaker at a receiver sensitivity of -99 dBm.

Fig. 9 shows the BER performance versus CNR of the SW-based TDMB receiver as a function of transmission modes I, II, and IV. The BER measurement for each transmission mode was performed under the same conditions as those given above. Almost the same BER performance was obtained for each transmission mode. Furthermore, using the implemented SW-based TDMB receiver, we could enjoy live audio, video, and data services available in Daejeon, Korea.

---

**Table II**

<table>
<thead>
<tr>
<th>Function block name</th>
<th>MCPS</th>
<th>Function block name</th>
<th>MCPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHY control</td>
<td>3.1</td>
<td>Time tracking</td>
<td>7.9</td>
</tr>
<tr>
<td>FIC symbol decoding</td>
<td>4.2</td>
<td>FIC channel decoding</td>
<td>10.4</td>
</tr>
<tr>
<td>MSC symbol decoding</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I/Q demodulation</td>
<td>8.8</td>
<td>Time deinterleaving</td>
<td>62.3</td>
</tr>
<tr>
<td>Freq. tracking</td>
<td>0.5</td>
<td>Depuncturing</td>
<td>25.3</td>
</tr>
<tr>
<td>Frequency compensation</td>
<td>13.8</td>
<td>Viterbi decoding</td>
<td>70.0</td>
</tr>
<tr>
<td>OFDM demod.</td>
<td>9.2</td>
<td>Energy desp.</td>
<td>0.2</td>
</tr>
<tr>
<td>Diff. demod.</td>
<td>10.9</td>
<td>Convolutional deinterlv., RS decoding</td>
<td>10.5</td>
</tr>
<tr>
<td>Total MCPS for decoding per frame</td>
<td>237.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCPS for the initial synchronization</td>
<td>72.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A: MCPS was measured for 544 kbps video content

**Table III**

<table>
<thead>
<tr>
<th>Block name</th>
<th>Measurement condition</th>
<th>MCPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video service</td>
<td>Baseline profile level 3</td>
<td>QVGA, 30 fps</td>
</tr>
<tr>
<td></td>
<td>H.264 decoder</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MPEG-4 ER BSAC decoder</td>
<td>48KHz sampling, 64 kbps data rate</td>
</tr>
<tr>
<td>Audio service</td>
<td>MUSICAM decoder</td>
<td>48KHz sampling, 128kbps data rate</td>
</tr>
<tr>
<td>Total MCPS</td>
<td></td>
<td>420</td>
</tr>
</tbody>
</table>

Authorized licensed use limited to: Korea Advanced Institute of Science and Technology. Downloaded on January 4, 2009 at 23:51 from IEEE Xplore. Restrictions apply.
VI. CONCLUSION

We presented the design and implementation of a SW-based TDMB receiver running on a single DSP. The PHY layer including synchronization and channel decoding as well as A/V decoders was implemented on the DSP core and the other functions such as a TS parser, A/V players, and a controller were implemented on the ARM core. We built an evaluation board with the target DSP and verified the receiver’s decoding capability. The functions of the SW-based TDMB receiver can be easily changed and upgraded and adapted to new standards. Furthermore, this SW-based TDMB receiver can be readily applied to conventional mobile phones, PMP players, or car navigation systems that are already running on a DSP platform.

Fig. 9. BER performance versus CNR of the SW-based TDMB receiver for the transmission modes I, II, and IV.

ACKNOWLEDGMENT

The authors thank the members of the Mobile Media Platform Center (MMPC) in KAIST for participating in the development of the SW-based TDMB receiver.

REFERENCES


Moohong Lee received the B.S. degree in Electronics Engineering from Hanyang University, Seoul, Korea, in 1989 and the M.S. degree in EE Engineering from KAIST, Seoul, Korea, in 1991. He now works as a team leader for software modem development in the MMPC, KAIST and is currently pursuing the Ph.D. degree in EECs from KAIST. He is a student member of IEEE. His research interests include wireless communication modem and systems.

Byungik Keum received the B.S. degree in EE Engineering from Chungang University, Seoul, Korea, in 2005 and the M.S. degree in EE Engineering from KAIST, Daejeon, Korea, in 2007. He now works for software modem development in the MMPC, KAIST and is currently pursuing the Ph.D. degree in EECs from KAIST. His research interests include digital signal processing and mobile TV, and wireless communications.

Jeong Han Jeong received the B.S. degree in EE Engineering from Pusan National University, Korea, in 2005 and the M.S. degree in EE Engineering from KAIST, Daejeon, Korea, in 2007. He now works for software modem development in the MMPC, KAIST and is currently pursuing the Ph.D. degree in EECs from KAIST. His research interests include digital signal processing modem, mobile TV, and wireless communications.

Young Serk Shim received the B.S. in Electronics Engineering from Seoul National University, Seoul, in 1976 and his M.S.E. and Ph.D. in Electrical Engineering from KAIST, Korea, in 1978 and 1982, respectively. He is currently a director of the MMPC, KAIST. He is a member of IEEE. His research interests include image signal processing, wireless communication systems.

Hwang Soo Lee received the B.S. in Electronics Engineering from Seoul National University, Seoul, Korea, in 1975 and his M.S.E. and Ph.D. in Electrical Engineering from KAIST, Seoul, Korea, in 1978 and 1983, respectively. He is currently a professor in the Department of EECs, KAIST. He is a member of IEEE. His research interests include signal processing, digital communications, and next-generation convergence networks.