Application Report
SPRA831 - July 2001

# Electronic Shock Protection (ESP) for CD Players That Use a TMS320C54x

*Nara Won*
*IA CD Team*

*Image and Audio Group*

**ABSTRACT**

Compact disc (CD) media is used for storage of audio files because of its relatively inexpensive cost and its large capacity. Today, people want to be able to play popular compressed audio files (MP3, WMA, AAC, etc.) with the same player that can play legacy audio CDs. In order to have skip-free playback, these players need to provide protection from external shocks, especially in the case of portable or car CD players. Traditionally, this external shock protection was achieved with mechanical buffers or external electronic shock protection chipsets.

This application report describes how to implement an electronic shock protection system with a TI digital signal processor (DSP). This solution removes any external shock protection chipset from the printed circuit board (PCB). This is a practical implementation used by many CD players that are already available on the market.

This document details the overall electronic shock protection (ESP) algorithm, including real implementation tactics, and the whole software structure of a legacy audio CD player using a TI DSP.

Also explained in this application report are many issues related to real-time programming, and good examples of how to use the following DSP/BIOS™ features: the hardware interrupt manager (HWI), software interrupt manager (SWI), periodic function manager (PRD), and buffered pipe manager (PIP). This document also provides an example of how to use real-time analysis using graphs.

**Contents**

**List of Figures**



**List of Tables**



# 1 Introduction

The TI DSP has been popular in the MP3 market since the market first emerged, and it has shown stronger benefits as the market moves toward multi-format audio players. A key segment of the market is using CD media storage because of the low cost and large capacity. A requirement of these CD players is to play legacy audio CDs as well as multi-format audio files. This creates a new issue for protecting audio sound from external shock in legacy CD audio on a multi-format audio CD player.

A portable CD player or car CD player suffers a noise problem because of external shocks. To get clean sound, a shock protection system is needed mechanically or electrically. A mechanical shock protection system is more expensive and more difficult than an electronic shock protection (ESP) system, making the ESP system more popular than the mechanical shock protection system.

The traditional approach for implementing an ESP system was by using a separate chipset. The most popular ESP chipsets are the Nippon Precision Circuits (NPC) SM590XX series. This approach was reasonable because a DSP was not needed to play only legacy audio CD. But today, multi-format audio CD players are popular. The players can play multi-format audio (MP3, WMA, AAC, etc.), including legacy audio CDs. To play multi-format audio, a DSP is necessary. In this case, if the DSP can do the ESP functionality, then a separate chipset for ESP can be removed, reducing circuit complexity and costs. The TMS320C54x DSP is already widely used to implement multi-format audio players, and it has sufficient millions of instructions per second (MIPS) to perform the ESP function while playing legacy audio CD.

This application report overviews the ESP system and describes an ESP implementation on the TI TMS320C54x DSP. It describes practical implementations that can be directly used to develop a multi-format audio CD player. The implementation described was executed on both a Rejina and Shruthi evaluation board. These are evaluation boards designed for IA CD solutions.

# 2 ESP System Overview

This section briefly describes issues related to ESP, to give an understanding to the basic concept about ESP and related terminologies. Detailed implementations will be described in section 3.

## 2.1 Responsibilities and Limitations of ESP

An ESP system tries to get clean sound output while external shocks exist. ESP does not clear noises caused by scratched or dirty CDs.

An ESP system cannot clear noises if the external shock continues too long. The time limitation is dependent on the dynamic random-access memory (DRAM) buffer size and the data-compression ratio.

## 2.2 Principle of ESP

Figure 1 shows a simplified example of the workflow of a real-time ESP.

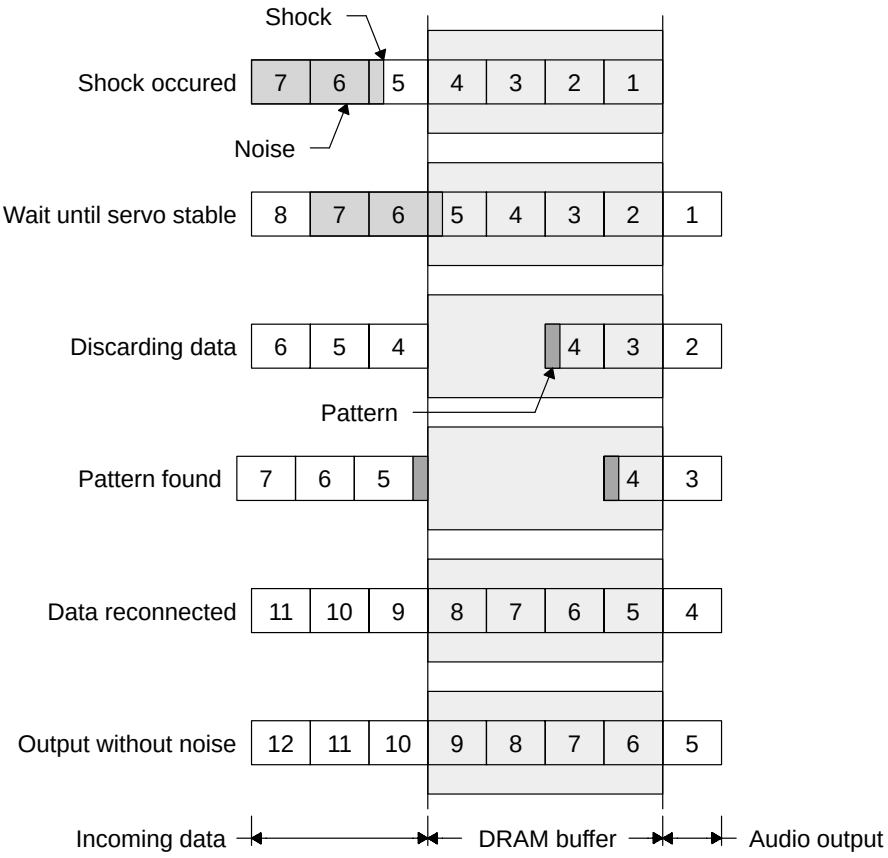


**Figure 1. Real-Time ESP Workflow**

In Figure 1, an external shock occurs at the 5th frame, causing servo to become unstable and resulting in noises during the 5th, 6th and 7th frames. After the servo becomes stable, an appropriate pattern is selected, and data after the pattern is discarded. The CD jumps back to the start of the 4th frame, and the pattern is compared with the incoming data stream.

The pattern matches with the data right before the 5th frame, so new data is saved starting from the 5th frame and is placed right after the 4th frame in the DRAM buffer. Now the 5th frame in the DRAM buffer is noise-free data, resulting in noise-free audio output of the 5th frame, even though there had been an external shock in this frame.

In a real system, there is no sign for the frame border, so pattern matching generally occurs in the middle of a frame. Also, note that a shock cannot be detected immediately.

## 2.3 CD Loader Speed

With a 1x speed CD loader, only mechanical shock protection is possible. To implement ESP, the CD loader has to provide at least 2x speed. Audio data comes faster than 44.1 kHz if a CD loader runs at more than 2x speed, but the audio output should be synchronized to the 44.1-kHz rate of the audio output clock. The DSP does the synchronization process with the DRAM buffer. The Philips M3 CD loader of the Rejina and Shruthi boards provide a maximum of 2x speed.

## 2.4 Data Buffering

Preloaded audio data is buffered in the DRAM buffer. The CD audio data sampling rate is 44.1 kHz, corresponding to 176.4k bytes in one second. Therefore, the theoretical shock protection time can be a maximum of about 12 seconds with 16M bits of DRAM. However, the practical shock protection time is less than 12 seconds with 16M-bit DRAM because of overheads related with data processing. The Rejina board has 16M bits of SDRAM, and the Shruthi board has 64M bits of SDRAM that are used for the ESP data buffer.

## 2.5 Data Compression/Decompression

To expand the shock protection time with the same DRAM buffer size, audio data is compressed before buffering and decompressed before it is output. This compression and decompression should not be disturbed to make real-time audio output. Therefore, lossy but simple adaptive differential pulse code modulation (ADPCM) algorithms are generally used. Because ADPCM is a lossy algorithm, sound quality degradation is not avoidable when ADPCM is applied.

The issues related to choosing a compression algorithm and their respective implementations, as well as sound quality degradations, will not be discussed in this application report. The implementation described supports only a 4:1 compression ratio with an ADPCM algorithm.

## 2.6 Pattern

Some continuous bytes of the audio data stream are used as a pattern to reconnect audio data. The pattern data is saved regularly when there are no external shocks. The position of the pattern data in the DRAM buffer, and the absolute pattern position in the CD (Min:Sec:Frame) are also saved.

## 2.7 Data Reconnection

After detecting an external shock, the DSP requests for the CD loader to rewind to the position of the saved pattern in the CD, and discards all data in the DRAM buffer that exist after the position of the saved pattern. It also prohibits incoming audio data to be written in the DRAM buffer. The DSP then searches for the saved pattern within the incoming audio data. If the DSP finds the pattern, then it permits incoming audio data located after the matching point to be written into the DRAM buffer. Buffered data in the DRAM buffer is transferred to the audio output during this process. To avoid mute audio, sufficient data is needed in the DRAM buffer during the audio reconnection process.

## 2.8 DRAM Buffer Full Condition

Even though there are no external shocks, the audio reconnection process is needed when the DRAM buffer becomes full because incoming data comes faster than 44.1 kHz. The DRAM buffer full condition is treated just like the external shock condition. To prevent a too frequent DRAM full condition from occurring, the DSP can request for the CD loader to change the speed to 1x after the DRAM buffer is full. If the remaining data is less than the predefined amount because of external shocks, the DSP will make another request for the CD loader to increase the speed back to 2x.

## 2.9 Shock Detection

To know if there were external shocks, the CD loader should provide shock signals. It is beyond the scope of this application report to determine which signal the CD loader should use for error detection. A signal, or a combination of some signals, should notify the DSP after the shock as soon as possible and to as great a degree of sensitivity as possible. The implementation described uses the Rejina and Shruthi boards with the Philips M3 loader, which use the Philips SAA7234 servo control chip. SAA7234 provides 2 shock signals: CFLAG and EF. The Rejina and Shruthi boards use only the EF signal because the M3 loader provides only the EF signal in its connector.

## 2.10 ESP Modes

The time limitation of the ESP system is dependent on the DRAM buffer size and the data compression ratio. If data compression is used, then sound quality will be degraded because the ADPCM algorithm used for the data compression is a lossy compression algorithm. This is why several ESP modes are generally provided, to allow the user several options balancing between the amount of shock-protection time and the sound quality.

There are 3 modes: Buffering ESP mode, Compressed ESP mode, and Disable ESP mode.

*Buffering ESP* mode does not use data compression, so there is no sound-quality degradation. The time limitation of *Buffering ESP* mode is shorter than that for the *Compressed ESP* mode.

*Compressed ESP* mode uses data compression and decompression, but the sound quality is degraded because of the lossy ADPCM algorithm. This mode can be divided into several sub-modes, according to the compression ratio of the algorithm. The implementation described in this application report supports only a 4:1 compression ratio.

ESP can be disabled in some cases, such as in boom boxes or stationary players. The *Disable ESP* mode does not do any tasks related with ESP. It is the legacy audio CD player mode. See Table 1.

**Table 1. ESP Modes**

| ESP Modes | Compression | DRAM Buffering | Data Reconnection |
|---|---|---|---|
| Compressed ESP | O | O | O |
| Buffering ESP | X | O | O |
| Disable ESP | X | X | X |

# 3 ESP System Implementation

This section describes an ESP implementation in detail, and a real ESP implementation.

## 3.1 Data Flow and Interrupts

Figure 2 shows the data flow and interrupts while a legacy audio CD is playing.

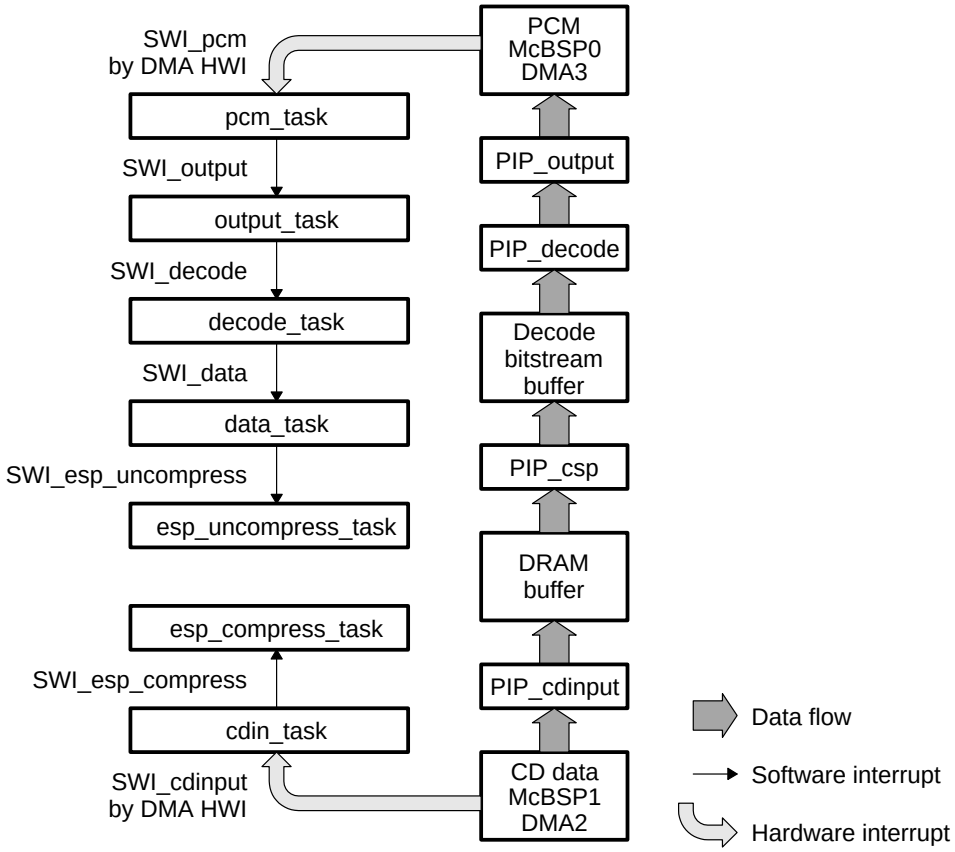


**Figure 2. Data Flow and Interrupts**

McBSP1 and DMA2 receive incoming CD data. DMA2 makes a hardware interrupt after collecting 1 frame of data. DMA2 HWI service routine posts *SWI_cdinput*.

*SWI_cdinput* runs *cdin_task()*. *cdin_task()* copies CD data to *PIP_cdinput* and posts *SWI_esp_compress*.

*SWI_esp_compress* runs *esp_compress_task()*. *esp_compress_task()* does compression and saves the results in the DRAM buffer when in normal playing time. It also does the pattern-matching process after shock.

DMA3 HWI is issued after consuming data in the DMA3 buffer, and the DMA3 HWI service routine posts *SWI_pcm*.

*SWI_pcm* runs *pcm_task(). pcm_task()* copies data in *PIP_output* to the DMA3 buffer, and posts *SWI_output* after consuming data in *PIP_output*.

*SWI_output* runs *output_task()*. *output_task()* applies volume and equalizer functions to data from *PIP_decode* and copies the results to *PIP_output*. It posts *SWI_decode* after consuming data in *PIP_decode*.

*SWI_decode* runs *decode_task()*. *decode_task()* separates left sound and right sound from the data in the decode buffer, and saves them in *PIP_decode*. It posts *SWI_data* when the decode buffer does not have sufficient data.

*SWI_data* runs *data_task()*. *data_task()* controls data flow according to key inputs. It consumes data in *PIP_esp* and posts *SWI_esp_uncompress*. FF/REW/Pause functions are implemented in *data_task()*.

*SWI_esp_uncompress* runs *esp_uncompress_task()*. *esp_uncompress_task()* decompresses data in the DRAM buffer, and saves the result in *PIP_esp*.

## 3.2 SWI Priorities

SWI priorities are set as shown in Figure 3. *PRD_swi* and SWIs related to HWI have the highest priority. *SWI_user* manages user key input and the LCD display; it has the lowest priority.

To avoid mute audio output caused by the system load, *SWI_decode* should have the highest priority among the remaining SWIs, and *SWI_data* should have lowest priority. However, this configuration wastes processing cycles because it results in too frequent *SWI_decode* posts. The compress/decompress algorithm requires a somewhat long processing time; therefore, if *decode_task()* has a higher priority than decompress task, then decompress task will be blocked by *decode_task().*

Current priority assignments in Figure 3 work well, and without any noise caused by the system load.

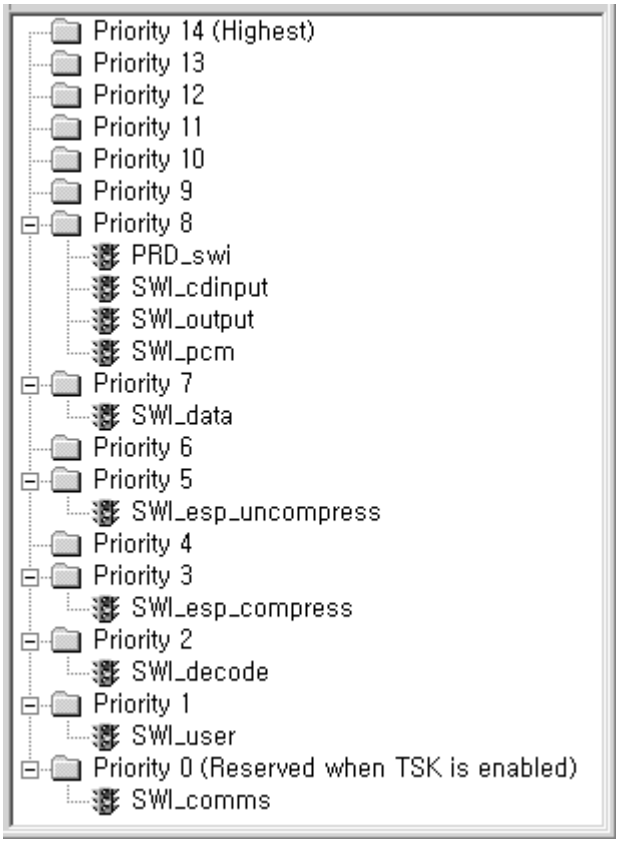


**Figure 3. SWI Priorities**

## 3.3 PIPE

Figure 2 shows all PIPEs. Two frames are sufficient for all PIPEs (excluding *PIP_cdinput*). *PIP_cdinput* has 4 frames, to avoid missing frames. The compression task requires a long processing time, so it consumes data in *PIP_cdinput* slowly. It can cause missed frames because of the PIPE frame becoming full while CD data comes in continuously. As a result, the number of frames for *PIP_cdinput* has been increased to 4 frames.

## 3.4 DRAM Buffer Management

A ring-buffer mechanism is used to manage the DRAM buffer in word units.

Additional ring-buffer functions are needed to implement ESP, including basic ring buffer management functions. These are functions to allow or disallow writing data to the DRAM buffer, and to discard data after the indicated DRAM buffer address.

When shocks are detected or if the DRAM buffer becomes full, incoming data should be discarded, and data after a valid pattern should be discarded. To discard incoming data, DRAM buffer writing permission will be clear. After data reconnection, DRAM buffer writing permission will be set.

These functions are implemented in *"buffman_cdda.c"* and *"buffman_cdda.h"*

## 3.5 Shock Detection

The McBSP2 port is connected to the EF pin on the Philips M3 CD loader. The M3 CD loader sets the EF signal if there are errors in the CD data stream. *McBSP2Config()* in *"shock.c"* configures McBSP2 as a GPIO port.

*CheckForShock()* in *"esp_task.c"* detects shocks while observing the McBSP2 port. PRD_Shock calls it at every tick cycle, to detect shocks as soon as possible. See Figure 4.

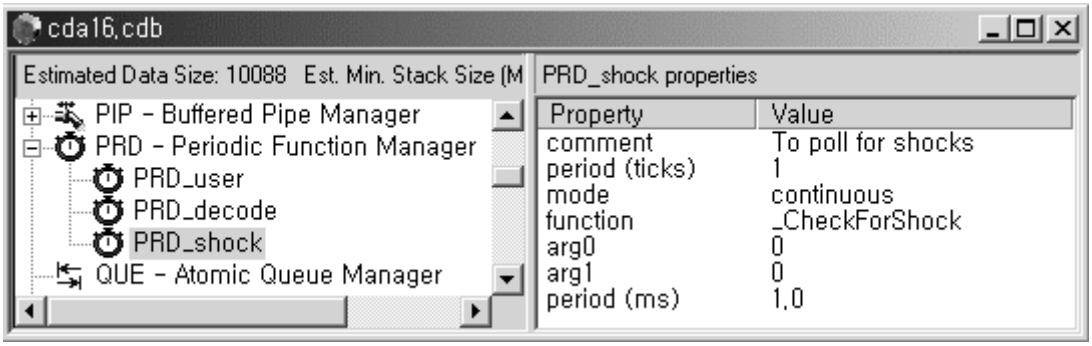


**Figure 4. PRD Object for Shock Detection**

The main shock detection algorithm is implemented in *checkPort()* in *"shock.c". checkPort()* has a state machine, which is described in Figure 5.

The initial state is *Normal* state. If there is a shock signal at McBSP2 port and pattern assignment succeeds, then the state goes to *Shock Detected* state. If the DRAM buffer becomes full and pattern assignment succeeds, then the state goes to *Shock Detected* state. Even though there are shocks at the port or the DRAM buffer becomes full, the state remains in *Normal* state if the pattern assignment fails. This means that there is no available saved pattern nor saved noise-free data in the DRAM buffer, so it is impossible to reconnect audio data.

While changing state to *Shock Detected* state, DRAM buffer writing is disabled, and one of the saved patterns is chosen for the pattern-finding process. Saved data after the pattern in the DRAM buffer is discarded because it is noise.

In *Shock Detected* state, it waits until servo becomes stable. There should be no new shocks detected during a pre-defined time, which is defined as *ESP_SHOCK_STABLE_CNT*. If *ESP_SHOCK_STABLE_CNT* becomes longer, then it helps stabilize the servo but reduces shock protection time. After checking that the servo is stable, the pattern-finding process is started, and the state goes to the *Pattern Finding* state.

In the *Pattern Finding* state, *esp_compress_task()* in *"esp_task.c"* compares the pattern with incoming data. If the pattern is found in the incoming data or is not found after several retries, then the state goes to *Normal* state. If there are shocks before pattern finding in the *Pattern Finding* state, then the retry count is reset, and it extends the number of retry times.

The REWIND command is sent to the servo controller before starting to compare pattern. It causes some shock signals, so *ESP_SHOCK_IGNORE_CNT* is used to define the number of shock times ignored in the *Pattern Finding* state.

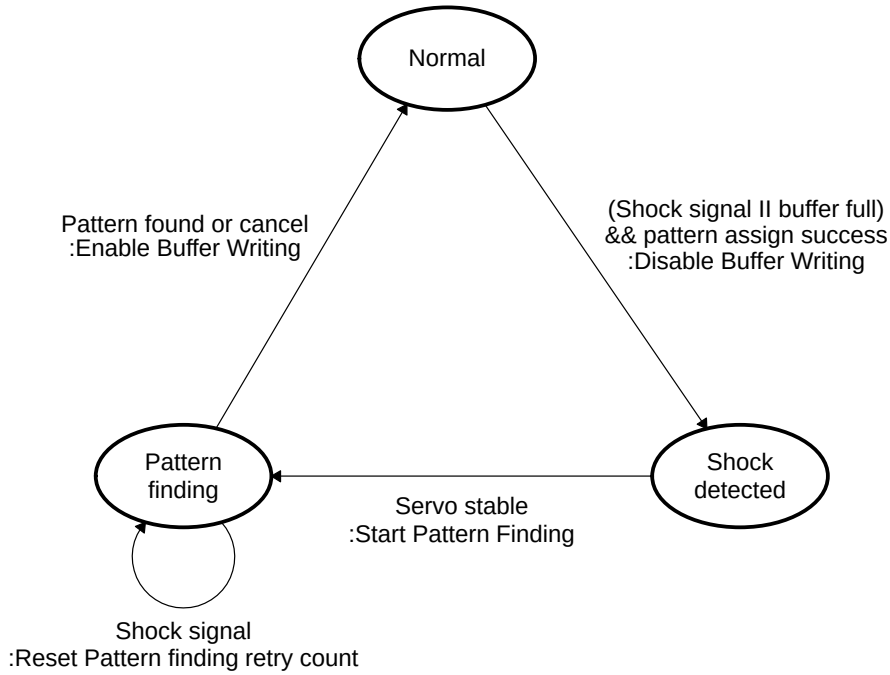


**Figure 5. Shock Detection State Diagram**

## 3.6 DRAM Buffer Full

Because CD speed is faster than 1x, the DRAM-buffer-full condition cannot be avoided. This condition can be treated just like the shock-detected condition. However, the DRAM-buffer-full condition occurs frequently if the CD speed stays faster than 1x. To prevent this, the CD speed should be changed to 1x, to reduce the DRAM-buffer-full condition occurrences. CD speed should be increased faster than 1x if data in the DRAM buffer is less than the predefined amount. The current implementation recovers CD speed when data in the DRAM buffer is less than 50% of the DRAM buffer size.

*esp_after_full_process()* in *"esp_task.c"* is called after every DRAM buffer writing. It checks to see if the DRAM buffer is full. If the DRAM buffer is full, then it changes the CD speed to 1x and starts the pattern-finding process. The servo becomes unstable after sending the CD speed change command to the servo controller. Therefore, *esp_after_full_process()* waits for a response from the servo controller so that the CD speed change is done. This starts the pattern-finding process after getting the response.

Increasing CD speed faster than 1x when there is not sufficient data in the DRAM buffer is implemented in *esp_uncompress_task()* in *"esp_task.c"*. It is the consumer of data in the DRAM buffer.

## 3.7 Pattern Saving

The last bytes written into the DRAM buffer become a pattern when incoming data is written into the DRAM buffer at every DRAM buffer writing time.

A pattern saved just before shock can be used to reconnect data if shock can be detected immediately. It is possible that the pattern can also be noise because the shock cannot be detected immediately.

By waiting until the servo is stable in the shock-detection routine, contiguous shocks can be treated. It is possible that there will be no valid pattern if shock occurs again after data reconnection.

Having only a saved pattern is not sufficient, and some pattern histories are needed to remove noise because of shock detection delay.

An array *eps_data[ ]* in *"esp_task.c"* is used to manage pattern history with the ring buffer mechanism. To do data reconnection, you need the pattern data, the address of the pattern in the DRAM buffer, and the absolute position of the pattern on the CD. The position on the CD is represented by Min:Sec:Frame information. 75 frames correspond to 1 second.

*Esp_data[ ]* is managed by some global variables. The current saving position in esp_data[ ] is saved in *esp_data_index*. The current number of valid patterns is saved in *esp_data_num*. The position of the pattern used to reconnect data is saved in *nShockedIndex*.

Patterns are captured from *PIP_cdinput* because data in the pipe is not compressed.

## 3.8 Discarding Data After Shock

Shocks cannot be detected immediately in a real system. Servo-control chipsets have some delays to set error flags. The shock-detection algorithm in the DSP also has some delay, to recognize the error signal. During the delays, noise data is being saved in the DRAM buffer. Because of this, some amounts of data should be discarded from the DRAM buffer to remove the noise data. Under this condition, it is hard to say whether the saved pattern is not noise, and some amount of saved patterns should be discarded as well.

After discarding recently saved patterns, the last pattern is used to find the reconnection point. The DRAM buffer address of the pattern is saved with the pattern, and the data after the address is discarded. Then the DRAM buffer has noise-free data, and the pattern locates at the end of the data.

Discarding the recently saved pattern is implemented in *ShockPatternAssign()*, and discarding the DRAM buffer data is implemented in *DRAMMoveWritePostion()*.

**NOTE:** The explanations given are normal implementations for data discarding. The current implementation is slightly different because of the ADPCM library behavior. This will be discussed in section 4.1.

### 3.9 Processes After Shock

*esp_start_pattern_finding()* rewinds the CD to the front of the pattern position and sets the shock-detection state to the pattern-finding state.

If the shock-detection state is in the pattern-finding state, then *esp_compress_task()* does not save data in the DRAM buffer, and runs *esp_pattern_find_task()* in *"esp_task.c"*. The routine compares the pattern with incoming data frames. *ESP_PATTERN_FIND_FRAME_NUM* defines the maximum number of frames to compare. If there is no data matched to the pattern, then the DSP requests to rewind the CD again, and retries to find the pattern in incoming data. The limitation of retries is *ESP_PATTERN_FIND_RETRY_NUM*.

After retrying up to the predefined limit, the DSP stops reconnecting the data, rewinds the CD to the position where the shocks are detected, and starts to save incoming data in the DRAM buffer. *PatternFindCancel()* does this function.

### 3.10 Saving Data in DRAM Buffer

*esp_compress_task()* runs *esp_write2dram()* while in normal play. *esp_write2dram()* checks ESP mode, compresses data in *Compressed ESP* mode, and saves the pattern data.

The ADPCM compression library requires a minimum of 8 words, but it is possible that there is not sufficient data in the current frame of *PIP_cdinput*. If the remaining data is not sufficient for the ADPCM compression library, then data is supplemented from next frame. Remaining data which was not able to be compressed is saved in *resBuf[ ],* and *resBufPtr* remembers the position of empty space in *resBuf[ ].* The data in *resBuf[ ]* is compressed before the data in the next frame of *PIP_cdinput* is compressed.

## 4 Other Issues

This section describes implementation-specific issues and important issues to make ESP complete.

### 4.1 Noise Effect in ADPCM Library

Theoretically, the reconnection point is right after the pattern, as explained earlier (the result is shown in Figure 6). The top graph in Figure 6 is the input sine wave, and the bottom graph is the sine wave after reconnection. With the sine wave file and the graph, you can clearly see the noise at the reconnection point. The method to get this graph will be described in Appendix A.

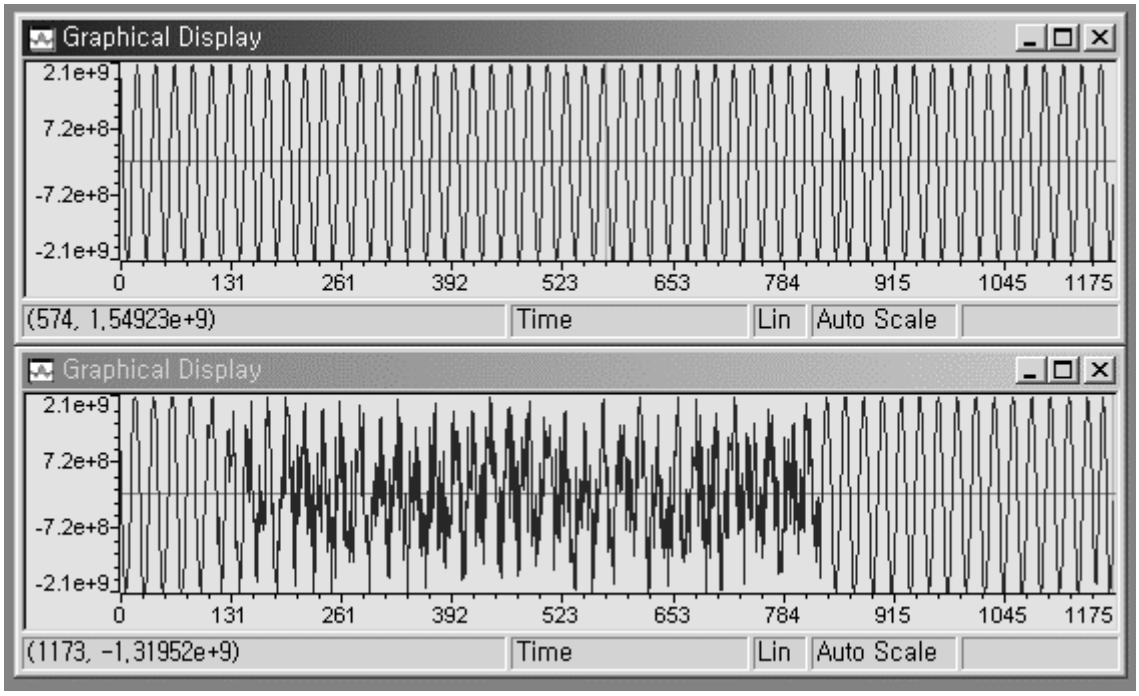


**Figure 6. Sine Wave After Reconnection (Right After Position of Pattern)**

This noise was caused by the noises being fed to the ADPCM library before shock detection. The ADPCM library uses past data to make the current output. If noise were fed in past inputs, then the current output would be affected. The noise is not removed by initializing the ADPCM library after the shock.

To remove accumulated past noise effects, some data in the DRAM buffer is left after the pattern. After finding matched data, the same amount of data is fed only to the ADPCM library, and is not saved in the DRAM buffer. After consuming the same amount of data, incoming data is saved into the DRAM buffer.

The assumption of this method is that some amount of data after the pattern has no noise after discarding sufficient data in the DRAM buffer, and that incoming data contains no noise after finding pattern match.

The current implementation includes this method. Figure 7 shows the result. The phase mismatch in the output graph is not noise because the frames of PIP_esp are used as a ping-pong buffer so the order of frames can be swapped.

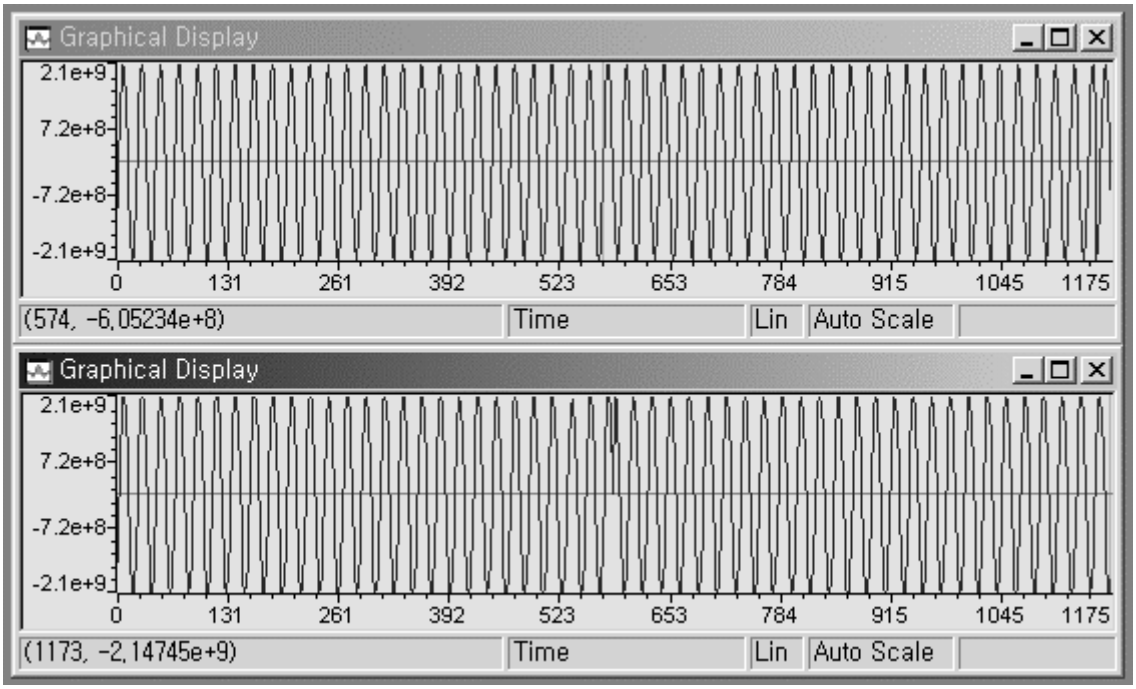


**Figure 7. Sine Wave After Reconnection With Delayed Connection**

### 4.2 Scratched CD

Most servo chipsets cannot distinguish between an external shock and scratches on the CD surface. Servo chipsets will set error flags in both cases. The DSP software should determine if the error flag is from a scratch or an external shock by observing the position of shocks. If shocks occur continuously at the same position, then it is determined to be a scratch on the CD surface. To do this, the DSP needs to have correct and detailed CD position information. The current implementation does not include codes for scratches on the CD surface because the M3 CD loader does not provide detailed Min:Sec:Frame information.

### 4.3 Soft-Shock Detection

Every servo chipsets can detect heavy shocks very well; however, servo chipsets cannot detect soft shocks. The soft shock can be tested by shaking the CD loader very softly. To overcome the soft-shock problem, some chipsets separate the *data reading* mode from the *audio playing* mode. Under the *audio playing* mode, noises caused by soft shocks are removed by the servo chipset.

It is very important to choose a servo chipset that provides the *audio playing* mode. There is no method to remove noise with software if shocks are not detected.

## 5 Conclusion

The TI DSP is used in many applications and products. This application report shows a good, practical example of an application with a TI DSP. The multi-format audio CD player using the ESP implementation described in this document is available on the market now.

Due to the ESP implementation on a TI DSP, the CD products using a TI DSP can have strong adaptability under many environments.

# 6 References

1. *Real-Time DSP Software Design for Portable MP3 Player on TMS320C54x DSP Using DSP/BIOS* (SPRA695).
2. SAA7324; Digital servo processor and Compact Disc decoder with integrated DAC (CD10 II) data sheet, Philips Semiconductors.
3. SM5903BF compression and non compression type shock–proof memory controller data sheet, Nippon Precision Circuits, Inc.

# Appendix A How to Obtain Sine Wave After Reconnection, Right After the Position of Pattern (Shown in Figure 6)

To get the sine wave after reconnection right after the position of pattern, as shown in Figure 6, you must disable the solution to remove the noise at reconnection point, and add some codes to know the time-playing reconnection point. Modify *"esp_task.c"* and *"buffman_cdda.c"* as follows:

*esp_pattern_find_task()* in *"esp_task.c"*

```
unsigned long ulRelinkAdd; // global variable to save reconnection point

void esp_pattern_find_task(void)
{
        . . .

       if( nRelinkIndex >= 0 )
       {
            . . .

           /* Write data to DRAM starting from where shock had occured */
           if(LoadData.ESPFlag == ESP_MODE_COMP)
               nNoWriteSize = ESP_NO_WRITE_DATA;
           else
               nNoWriteSize = 0;

           nNoWriteSize = 0;  // to do no delayed reconnection
           ulRelinkAdd = DRAMBufGetWritePosition(); // to save reconnection point

           esp_write2dram();

        }
```

*DRAMMoveWritePosition()* in *"buffman_cdda.c"*

```
int DRAMMoveWritePostion(unsigned long ulAdd)
{
/*     ulStartAddWrite = ulAdd;    */

       if( LoadData.ESPFlag == ESP_MODE_COMP )
           //ulAdd = ( ulAdd + (unsigned long)ESP_NO_WRITE_DATA ) %
DRAM_BUF_SIZE_WORD;
           ulAdd = ( ulAdd ) % DRAM_BUF_SIZE_WORD; // to do no delayed reconnection

       if( ulStartAddRead >= ulStartAddWrite )
       . . .
```

To set the break point, modify *esp_uncompress_task()* as follows:

*esp_uncompress_task()* in *"esp_task.c"*

```
    if( nReadDataSize )
    {

        if (LoadData.ESPFlag != ESP_MODE_COMP || FFREWFlag) {
              memmove((int*)out->writerAddr + nESPUncompDataOffset, aBufDec, nReadDa-
taSize);
              nSizeUncompressed = nReadDataSize;
        }
        else
              esp_uncompress( (int*)out->writerAddr + nESPUncompDataOffset, &nSizeUn-
compressed, aBufDec, (int)nReadDataSize );
        nESPUncompDataOffset += nSizeUncompressed;
        if( ulReadAdd <= ulRelinkAdd && DRAMBufGetReadPosition() >= ulRelinkAdd )
        {
             static int k = 0;
             k++;                 // break point to get graph : code has no meaning
        }
    }
```

Set a break point at the indicated line, and run CD Audio project with a sine wave CD. After the break, select *"View/Graph/Time Frequency…"* in Code Composer Studio™ v1.2. Then the property-setting window will be appear. In the property-setting window, set *Start Address* to the start address of the frame buffers of PIP_esp, *Acquisition Buffer Size* to 1176, and *Display Data Size* to 1176. Then press the OK button.

Code Composer Studio is a trademark of Texas Instruments.

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265