

An Affordable Platform for Learning Real-Time Adaptive Signal Processing

Miguel Alonso, Jr., and Armando Barreto

Abstract—This paper presents a practical, economical, and useful way of enhancing an existing Real-Time Digital Signal Processing (DSP) learning kit, the Texas Instruments TMS320C3x DSK, towards the development of Adaptive Signal Processing (ADSP) algorithms. Many DSP educators use this kit to provide a basic platform for learning the real-time implementation of DSP algorithms. The kit includes an assembler and a debugger for software development. In its original configuration, the kit can be connected to just one signal source and provides a single analog output, which is sufficient for the prototype implementation of basic DSP algorithms. In the last few years, however, the sub-field of Adaptive Digital Signal Processing has become increasingly important in scientific and industrial applications. As such, real-time implementation of these algorithms, which typically require more than one independent input and output, is highly desirable. This paper outlines how to develop such training platform by enhancing the popular TMS320C3x DSK with a stereo input/output module, and provides a software template and an example to aid in the software development for this enhanced training platform.

Index Terms—DSP Starter Kit, Real-Time DSP training kit, Adaptive Digital Signal Processing, Digital Signal Processing, Adaptive Noise Canceller, Least Mean Squares.

I. INTRODUCTION

The continued enhancement of Digital Signal Processors (DSPs), in terms of low cost, availability, size and efficiency, has fueled the need for engineers with the knowledge to design and implement systems using DSP technology. From toys to portable MP3 players, to cell phones, DSP systems can be found in many electronic products on the market. Some of the skills that a DSP professional must possess range from DSP algorithm development to real-time implementation of DSP designs.

In the past, DSP systems were designed based on observations or assumptions on the behavior or characteristics of the signal of interest to be processed by the system. So, for example, a DSP system that implements a narrowband filter to remove polluting noise centered around 60 Hz will only be effective if the assumption about the frequency spectrum of the polluting noise is correct. But imagine a situation in which the characteristics of the polluting noise are not known or vary. It is evident that under these circumstances a fixed DSP solution is not effective. This is where Adaptive Digital Signal Processing (ADSP) comes into play.

During the 1980's, a new approach to Digital Signal Processing, called Adaptive Digital Signal Processing, emerged. The basis for this new approach is self-adaptation, that is, the DSP system adapts or changes its nature to improve its performance. ADSP systems are not completely designed in advance. The basic structure of the system is set, but the parameters governing the ADSP are flexible and are adjusted according to what is referred to as an "adaptation algorithm". Many classical DSP systems (e.g., digital filters) normally process one input signal and generate one output signal. ADSP systems, on the other hand, read not only the "Primary Input", which contains the signal

of interest, but also a second input called the “Reference Input”, which is usually a sample of the signal that is to be removed, i.e. a sample of only noise. The final result is a system that removes from the “Primary Input”, a signal that is correlated, or similar to the signal present at the “Reference Input”.

Through a very ingenious analysis of these adaptive systems, a very simple yet powerful algorithm, called the Least Mean Squares (LMS) adaptation algorithm [6], was developed to be the driving force that molds the adaptive system for optimal removal of unwanted noise. The LMS algorithm has repeatedly proven its effectiveness in industry, being implemented in numerous real-time industrial applications. Thus, it is of utmost importance to provide students with a practical understanding of ADSP systems, as well as knowledge of real-time implementations of these systems. This need has been partially addressed by the development of software tools to aid in the simulation and development process of adaptive digital signal processing systems [4],[5]. These tools, however, do not expose students to the obstacles that real-time implementations must overcome in industrial applications.

Recently, information on how to set up a practical and affordable real-time DSP learning station based on the Texas Instruments TMS320C3x Digital Signal Processing Starter Kit (DSK) and a Personal Computer has been made available [1]. The kit used in that setup, identified here as the “C3x DSK”, is very affordable, containing a target board designed around the TMS320C31 DSP chip, an assembler, used to convert source files into DSK executables, and a DOS-based debugger, which allows a host PC to communicate and download the executables via the parallel port to the DSK. This setup is sufficient for traditional real-time DSP implementations, where only a one-

channel input is needed. But, for most ADSP applications, a minimum of two input channels (“Primary Input” and “Reference Input”) is necessary. This paper presents a method to overcome this limitation while keeping the cost of the training platform down, and using the original software development tools, which are already familiar to many educators and students.

The setup described here is suggested for student laboratory exercises in an advanced undergraduate or graduate real-time DSP course. Students require the fundamental DSP background typically provided in an introductory DSP course, as shown in Figure 1. If students are to modify the programs provided, they will also need familiarity with microprocessor assembly programming. However, instructors could simply show the experiments, as provided, as a supporting demonstration for the corresponding theoretical concepts. Within the real-time DSP course, the real-time ADSP experiments would naturally follow exercises in implementation of fixed DSP systems, such as Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) digital filters.

In the absence of a dual-input real-time platform, such as the one proposed in this paper, students would be constrained to experience real-time implementations of a narrow subset of ADSP systems that employ only a single input signal, such as the predictor configuration used as a self-tuning filter, or an adaptive line enhancer [6].

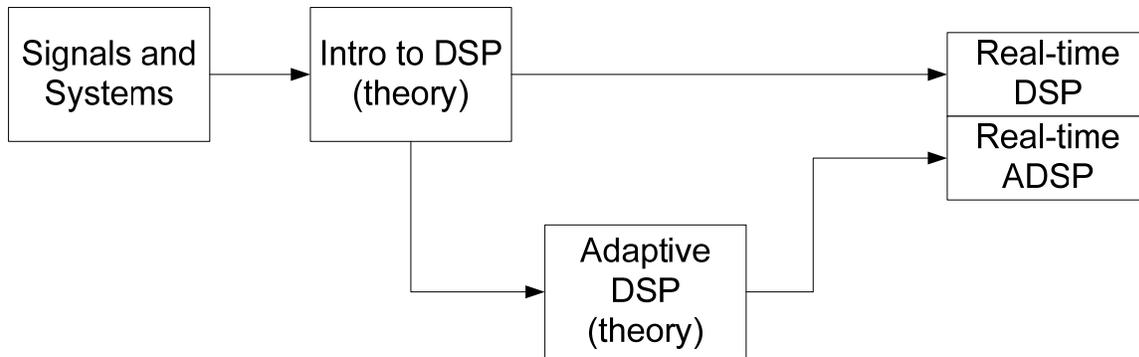


Figure 1. Typical course sequence that may lead to the use of the setup described here for experimentation in Real-Time Adaptive Digital Signal Processing (ADSP).

II. HARDWARE SETUP

The DSP processor in the C3x DSK board communicates to the analog world through a single-channel Analog Interface Circuit, or AIC (TLC32040), which is connected to its serial port. Fortunately, the designers of the C3x DSK board had the foresight to establish the connection of the DSP chip serial port to the AIC through a jumper header, JP1 (see Figure 2). This allows the user to easily disconnect the AIC from the DSP by removing the jumpers [2]. In order to experiment with adaptive signal processing systems, requiring not only a primary input, but also a reference noise input, the serial port of the DSP chip has to be connected to an analog input/output device capable of providing two independent input/output channels. Fortunately, stereo codecs (coder-decoders) with standard serial interfaces and capable of sampling two input channels at sampling rates appropriate for digital audio (e.g., 441000 Hz.) have proliferated and subsequently become affordable. One prime example of this kind of devices is the Crystal Semiconductor CS4216 stereo codec. This chip meets all the requirements for an analog interface that would enable the implementation of adaptive signal processing systems with the C3x DSK.

To effectively act as analog front-end for the TMS320C31 DSP chip, the CS4216 needs to be configured with a few analog components. The application note, “CDB4216: CS4216 Evaluation Board” included in Crystal Semiconductor’s Databook, [3], provides all the necessary details and a printed circuit board layout to build a stereo codec board based on this device. Similarly, Chassaing [2] shows the schematics to develop a stereo codec board based on the CS4216/18 (along with a basic program to verify its functionality). Furthermore, there are some ready-made stereo codec boards, which can be utilized directly as dual-channel analog input/output systems for the C3x DSK. In particular, the platform described here uses the “DSProto Codec Board”, based on the CS4216, and manufactured by Digital Control Labs (<http://digitalcontrollab.com>). This affordable board contains a 12 MHz clock IC, which acts as master clock for the codec. A bank of switches determines the effective clock rate delivered to the codec, thereby adjusting the sampling frequency used for analog-to-digital and digital-to-analog conversions. The board connects to a main processor through a serial port available via a male DB-15 edge connector. This DB15 connector must be connected to selected pins in the JP1 jumper header of the C3x DSK board, as indicated in Figure 2, to establish the hardware connection between the two systems. The actual connection can be made using a female DB15 ribbon connector to attach to the DSProto Codec board (Digi-Key Part No.CFP15T-ND), and a 26-pin, dual-row, socket ribbon connector (Digi-Key Part No.CSC26T-ND), to attach to the jumper header (JP1) on the C3x DSK. The jumper header JP1 in the C3x DSK board has only 22 pins. However, a 26-pin connector is the smallest standard size available for the connection and, fortunately, the C3x board has enough space to accommodate the slightly larger connector. Because of

the disparity in the number of contacts on each of the devices, the ribbon cables from each connector will have to be spliced and matched on a conductor-by-conductor basis to achieve the connections shown in Figure 2.

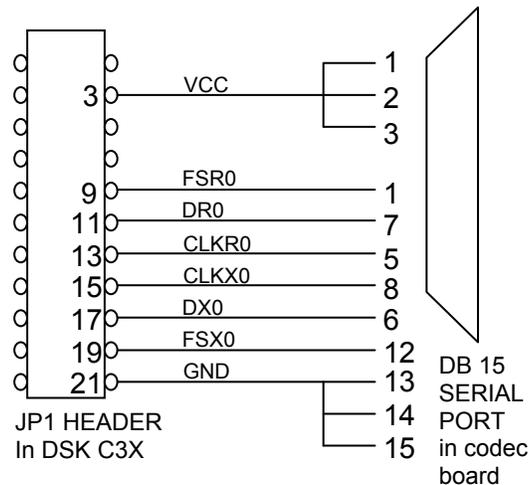


Figure 2- Hardware connections from the DSK to the serial port in the codec board

It should be noted that in this configuration, the DSPProto codec board acts as a somewhat autonomous module, with its own timing signals determined by the on-board clock circuit and the switches that control its frequency division. The interaction between the DSPProto Codec board and the C3x DSK is controlled by the sampling rate used in the codec. Every time an analog-to-digital conversion is completed in the codec, a serial port interrupt is issued to the DSP chip in the C3x DSK board.

The overall configuration of the training station, showing the connection of the host PC with the C3X DSK board through the parallel port cable supplied with the DSK, and the connection of the C3X DSK board with the codec board, detailed in Figure 2, is shown in Figure 3, for reference.

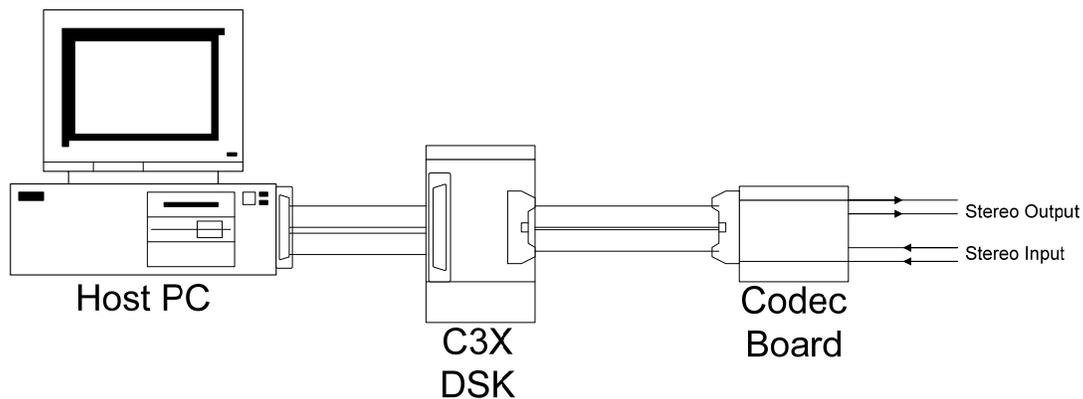


Figure 3 - Overall component connections in the real-time ADSP training station.

III. SOFTWARE SETUP

Once the hardware connections between the codec board and the DSK board are established, the codec board will convert pairs of samples and exchange data with the DSP upon reset through the serial port. The basic assembly programming template, shown in the following sub-section, initializes the DSK board and establishes an interrupt service routine to receive new input samples and send output samples to the codec.

A. Template Assembly Program

```

; 2-CHANNEL IN/OUT PROGRAMMING TEMPLATE
;-----
; Link sections to memory locations
;-----
        .start  "intsect",0x809FC5
        .start  ".text",0x809900
        .start  ".data",0x809C00
;-----
; Immediate constants (If needed, use .set)
;-----
;-----
; Interrupt vector for XINT0 ser port int.
;-----
        .sect   "intsect"
        BR      INT_SER
;-----
; Data section
;-----
        .data
PER_BASE      .word  0x0808000
SP_CTRL_WD    .word  0x0EBC0000
;serial port set-up data
IN_SAMPLE_L   .word  0           ;In Sample Left
IN_SAMPLE_R   .word  0           ;In Sample Right
OUT_SAMPLE_L  .word  0           ;Out Sample Left
OUT_SAMPLE_R  .word  0           ;Out Sample Right
;-----
; [A] Text section (entry point) MAIN
;-----
        .entry BEGIN      ; code start
        .text
BEGIN      LDP    SP_CTRL_WD ;init dpp
          CALL   PER_INIT
WAIT       IDLE   ; wait for interrupt
          BR     WAIT
;-----
; [B] Peripheral Initialization Procedure
;-----
PER_INIT   NOP
;-----
;1. Preserve AR0 and R0 Registers
;-----
        PUSH   AR0
        PUSH   R0
;-----
;2. Load Peripheral Base Address
;-----
        LDI    @PER_BASE,AR0
;-----
;3. Set up serial port 0(to talk to codec)
;-----
        LDI    0x131,R0
        STI    R0,*+AR0(0x42)
        STI    R0,*+AR0(0x43)
        LDI    @SP_CTRL_WD,R0
        STI    R0,*+AR0(0x40)
        LDI    0,R0          ; R0 = 0

```

```

        STI    R0, *+AR0(0x48)
;-----
        POP    R0
        POP    AR0
;-----
;4. Set up interrupt(for serial transfer)
;-----
        LDI    0x0,IF      ; clear IF
        OR     0x10,IE     ;en EXINT0
        OR     0x2000,ST   ;glob. int
        RETS
;-----
; [C] CS CODEC I/O Transfer Procedure
;-----
CS_IO      PUSH   AR0
           PUSH   R0
           PUSH   R1
           LDI    @PER_BASE,AR0
           LDI    @OUT_SAMPLE_L,R0
           LDI    @OUT_SAMPLE_R,R1
           LSH    16,R0
           AND    -1,R1
           OR     R0,R1
           STI    R1, *+AR0(0x48)
           LDI    *+AR0(0x4C),R1
           LDI    R1,R0
           LSH    16,R1
           ASH    -16,R1
           ASH    -16,R0
           STI    R0,@IN_SAMPLE_L
           STI    R1,@IN_SAMPLE_R
           POP    R1
           POP    R0
           POP    AR0
           RETS
;-----
; [D] Interrupt Service Procedure
;-----
INT_SER    LDI    @IN_SAMPLE_L,R3
           LDI    @IN_SAMPLE_R,R4

; **** Insert DSP algorithm here ****
           MPYI   4,R3
           MPYI   4,R4

           STI    R3,@OUT_SAMPLE_L
           STI    R4,@OUT_SAMPLE_R

           CALL   CS_IO
           RETI

.end

```

B. Explanation of Template

In this template assembly program, the first few lines define the position in memory of the three assembly sections, “data”, “text” (for program instructions), and “intsect”

(pointer to interrupt service routine). The data section allocates a few memory locations that will be used to receive and send left and right pairs of values to and from the codec.

The actual assembled program, in the text section, comprises 4 modules, as identified with square brackets in the program listing. Module [A] defines the “main” program, which initializes the data page pointer, calls the peripheral initialization routine, PER_INIT, and then just idles, waiting to receive an interrupt. When an interrupt occurs, it is serviced by the interrupt routine, Module [D]. Upon completion of this module, the program control returns to the idling loop.

Module [B], the peripheral initialization routine, PER_INIT, is critical, for it sets up the serial port to enable communication with the codec board by writing appropriate values to the serial port global control register, the serial port pin-configuration registers, and finally, enabling the interrupt for the serial port.

Module [D] (INT_SER) is a very simple interrupt service routine for the serial port interrupt. As shown in this template, it brings in the “newest” two input values, from memory locations IN_SAMPLE_L and IN_SAMPLE_R, to registers R3, and R4, respectively, as 16-bit values. At this point the samples are available for any form of processing that the programmer may want to implement. In the template the samples are just multiplied by four and then sent back to memory locations OUT_SAMPLE_L and OUT_SAMPLE_R. Then the interrupt routine calls the codec I/O transfer procedure, which effectively exchanges data with the codec board and returns flow control to the main program (Module [A]).

The effective transfer of values between the codec board and the four designated memory locations is performed by subroutine CS_IO, which is module [C] in the

template program. This routine appends the two 16-bit output values, taken from memory locations `OUT_SAMPLE_L` and `OUT_SAMPLE_R`, into a single 32-bit word and writes it out to the data transmit register of serial port 0 (mapped at address 808048h). It then reads the latest samples converted by the codec from the data receive register of serial port 0 (mapped at 80804Ch), breaks this 32-bit received word into two 16-bit sample values from each channel, and makes them available to the real-time interrupt routine in memory locations `IN_SAMPLE_L` and `IN_SAMPLE_R`.

Assembling and downloading the template file described above to the C3x DSK will allow the user to listen, via amplified speakers, to each one of the stereo channels of a tape or CD player plugged into the inputs of the DSPProto Codec board. In fact, by changing the factor used to multiply the contents of R3 and R4 in the interrupt service routine, the user should be able to appreciate different amounts amplification at the output of the codec board.

IV. EXAMPLE IMPLEMENTATION: ADAPTIVE NOISE CANCELLER

The combination of hardware integration of the C3x DSK board with the DSPProto Codec board, and the development of the software template discussed above, provides a complete PC-based environment in which the implementation of real-time adaptive signal processing systems can be learned.

We will demonstrate the practical use of this learning environment through the implementation and real-time verification of a classic Adaptive Noise Canceler (ANC) system, as described in the pioneering book by Widrow and Stearns [6]. The block diagram of the ANC is shown in Figure 4. This figure illustrates how the primary input sequence, $d(n)$, is applied to the upper path in the ANC, while the reference noise

sequence (“reference input”), $x(n)$, is processed by an Adaptive Transversal Filter (ATF) in the lower path of the diagram, Figure 4.

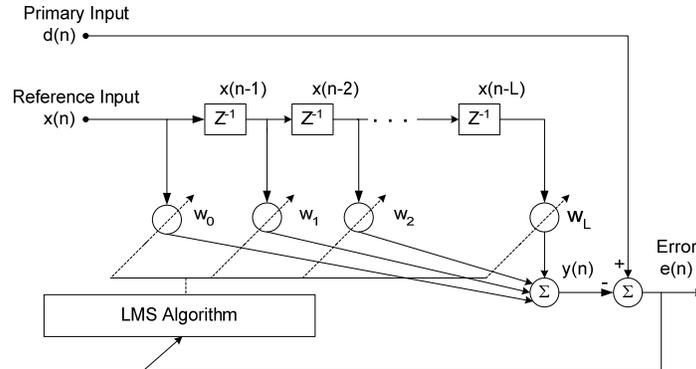


Figure 4- Block Diagram of the Adaptive Noise Canceller

The ATF is a non-recursive filter that implements the sum of products of a set of coefficients or “weights”, w_0, \dots, w_L , with the most recent reference input sample $x(n)$, and L past samples, $x(n-1), \dots, x(n-L)$, respectively. So, at each sampling instant, the output of the ATF, $y(n)$ is calculated as:

$$y(n) = \sum_{j=0}^L w_j x(n-j) \quad (1)$$

The “error signal”, $e(n)$, in the ANC is calculated as the difference between the current primary input sample, $d(n)$, and the current output from the ATF, $y(n)$:

$$e(n) = d(n) - y(n) \quad (2)$$

The concept behind the ANC is that if the primary input is polluted with some noise component, and the noise is available for sampling somewhere else where it is not mixed with our signal of interest, the ATF will adapt to a configuration in which the reference noise is transformed into a signal, $y(n)$, that matches the polluting component in $d(n)$. Therefore, the “adjusted noise” $y(n)$ is subtracted from the polluting noise

component in the primary input, $d(n)$, leaving only the (“clean”) signal of interest in the resulting signal, $e(n)$. Effective removal of the polluting noise component will cause a reduction in the power of $e(n)$. That is why a common adaptation procedure seeks to obtain the “Least Mean Square” (LMS) value of the error signal, $e(n)$, achievable through the adjustment of the ATF. The LMS adaptation algorithm iteratively changes the weights according to reference input samples present in the ATF, $x(n) \dots x(n-L)$, and the error signal, $e(n)$, which is the overall output of the ANC. The LMS speed of convergence is regulated by a fixed parameter, β :

$$(w_i)_{\text{NEW}} = (w_i)_{\text{OLD}} + \beta e(n) x(n-i) \quad i = 0, 1, \dots, L \quad (3)$$

To perform as a real-time adaptive noise canceller, our system must implement equations 1, 2 and 3 above, in sequence, every time a new sample is acquired from the primary and reference input signals, that is, every time the interrupt service routine for the serial port is executed. In order to achieve this, three modifications have to be made to the simple template file provided before:

1. The following instructions must be added to the assembly of the “.data” section, immediately after the instruction `OUT_SAMPLE_R .word 0:`

```

BETA      .float      2.5e-14
ER        .float      0
LENGTH   .word       64
WN        .float      0
          .loop       254
          .float      0
          .endloop
          .brstart   "XN_BUFF", 256
XN        .sect       "XN_BUFF"
          .loop       255
          .float      0
          .endloop
XN_ADDR   .word       XN
WN_ADDR   .word       WN
ER_ADDR   .word       ER
BETA_ADDR .word       BETA

```

These instructions allocate memory and initialize to zero arrays for the samples of the reference input, $x(n)$, and for the adaptable weights, w_i .

2. The following two instructions must be added in the “main” program (Module [A]), right after the call to the subroutine PER_INIT:

```
LDI          @LENGTH, BK
LDI          @XN_ADDR, AR1
```

These instructions establish the length of a circular buffer for the samples of the reference input and initialize the pointer to the beginning of the circular buffer.

3. The following lines of code must be placed in the interrupt service routine for the serial port 0 (Module [D]), *in substitution* of the two instructions:

STI R3 ,@OUT_SAMPLE_L, and STI R4 ,@OUT_SAMPLE_R

```

                                FLOAT      R3, R5
                                FLOAT      R4, R6
                                STF         R6, *AR1++%
                                LDI         @WN_ADDR, AR0
                                LDI         @ER_ADDR, AR2
                                LDF         0, R0
                                LDF         0, R2
                                LDI         @LENGTH, R3
                                SUBI        1, R3
                                RPTS        R3
                                MPYF3      *AR0++, *AR1++%, R0
||                                ADDF3     R0, R2, R2
                                ADDF        R0, R2
                                SUBF3      R2, R5, R7
                                MPYF       2, R7
                                FIX         R7, R0    ;Error
                                FIX         R2, R1    ;Out ATF
                                STI         R0, @OUT_SAMPLE_L
                                STI         R1, @OUT_SAMPLE_R
                                MPYF       @BETA, R7
                                STF         R7, *AR2
                                LDI         @LENGTH, RC
                                SUBI        1, RC
                                LDI         @WN_ADDR, AR0
                                RPTB       LMS_LOOP
                                MPYF3      *AR2, *AR1++%, R0
                                LDF         *AR0, R1
                                ADDF        R1, R0
LMS_LOOP                        STF         R0, *AR0++

```

This sequence of instructions implements equations 1, 2, and 3 above to calculate the output of the ATF, evaluate the error and use it to update the ATF weights according to the LMS algorithm, before the next interrupt.

Figure 5 shows a flow chart representation of the ANC assembly program. It should be noted that the modules outlined in the description of the template ([A], [B], [C], [D]) remain present. The main enhancement has been performed on module [D], the interrupt service routine, where the 3 equations required by the LMS algorithm have been implemented. Moreover, it should also be noted that the template provided is general enough that only module [D] would need to be modified to experiment with alternative ADSP algorithms.

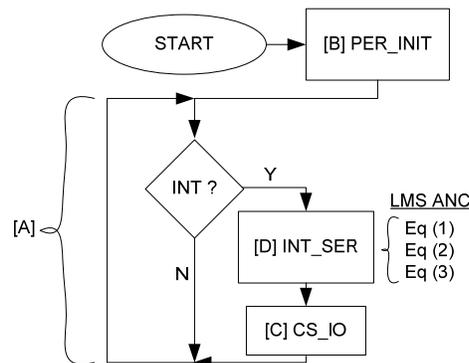


Figure 5- Flow chart for the ANC implementation. The letters in brackets correspond to the four modules identified in the assembly template (Please see section III for a description of these modules.)

V. RESULTS

We have verified the functionality of the ANC described above by processing a stereo recording which contained in the left channel a mixture of low-amplitude vocal music and large amplitude 440 Hz sinusoidal interference and in the right channel a 440 Hz sinusoid at different phase and amplitude. These signals were created as discrete

sequences in Matlab[®], recorded to a CD-ROM as a stereo wave file, and played back into the left and right input connectors of the DSProto Codec Board from a portable CD-Player. By monitoring the signal obtained from the LEFT output connector in the DSProto Codec board we can see (Figure 6) the effect of adaptation. The “error” signal in the ANC system is initially very similar to the “primary input” signal acquired from the left channel of the recording, containing both the music and the large-amplitude sine interference, which is responsible for most of the amplitude of $e(n)$, at this point in time. As the LMS algorithm adapts the weights in the ANC to minimize the power of the error, the sinusoidal component is progressively removed from the error signal. After adaptation is complete, the error signal is only the music without the large sinusoidal interference. The noise cancellation effect can also be verified by connecting the LEFT output of the DSProto Codec Board to an amplified (PC) speaker. In this case the adaptation will progressively remove the strong tone component that the left input channel has included in it. Students can experiment with different values of the parameter “BETA” and the length of the ATF in the program and verify the effect that those changes have on the convergence of the adaptation process.

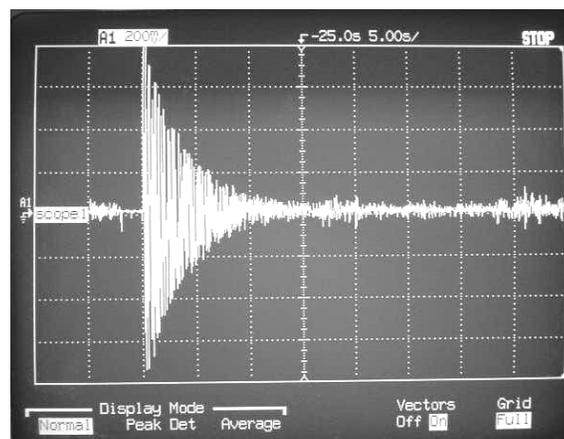


Figure 6- ANC error signal showing progressive elimination. The system was restarted at the second time division (10 s) from the left.

VI. CONCLUSION

This paper has demonstrated how to enhance a popular and economic PC-based DSP teaching kit to make it a learning platform for the real-time implementation of adaptive signal processing algorithms. A detailed description is provided of the hardware modifications and the software required to implement a classical example of adaptive systems: an Adaptive Noise Canceller (ANC), using this learning tool. Fully commented versions of the template and ANC assembler files, as well as their DSK executables, can be downloaded from http://dsplab.eng.fiu.edu/ASP_DSK/. This web site also contains the C-source and DOS-executable versions of a host (PC) program that allows the user to change the value of the adaptation rate parameter, β , change the number of ATF weights, and save the weights of the adaptive transversal filter to a disk file. A simple README (text) file provides step-by-step instructions to run the DOS-executable for demonstration purposes. Also available at this site is the sine_voice.wav file, which can be played out with appropriate PC software or burnt into a CD and then played with a CD-player, to test the real-time implementation described.

The setup suggested and the sample software freely available to students and educators will enable them to experience first-hand alternative real-time DSP solutions that can be achieved through the use of adaptive systems. Additionally, the set of files provided on the web site may motivate interested instructors of (theoretical) DSP courses to implement a single hardware setup as described and use the ready-to-run executable programs provided at the web site as a demonstration for their classes. We are convinced that this will lend added credibility and a sense of practicality to the theoretical concepts taught in the classroom.

For the reader's convenience, Table I, below, summarizes the WWW resources that have been mentioned in this paper, including a very brief description for each of them.

Table I. WWW resources referred to in the paper.

URL	Description
http://digitalcontrollab.com	Digital Control Lab - manufacturers of the "DSPProto Codec Board"
http://digikey.com	Digikey - Electronic component suppliers
http://dsplab.eng.fiu.edu/ASP_DSK/	Fully commented versions of the template and ANC assembly files, DSK executables, and host PC program to control the DSK.

VII. REFERENCES

- [1] Barreto, A. B., Yen, K. K., and Aguilar, C. D., "PC-based DSP Training Station", Proc. ASEE 1998 Annual Conference, Seattle, WA, June 1998. Session 1220: Digital Signal Processing, (CD-ROM format)
- [2] Chassaing, R., "Digital Signal Processing: Laboratory Experiments Using C and the TMS320C31 DSK", Wiley Interscience, 1999.
- [3] Crystal Semiconductor Corp., "Crystal Semiconductor Audio Databook", Austin, TX, 1994.
- [4] Harteneck, M., Stewart, R. W., "Adaptive Signal Processing JAVA applet", IEEE Transactions on Education, Vol. 44, Issue 2, May 2001, pp. 200

- [5] Stewart, R. W., Harteneck, M., Weiss, S., “Interactive Teaching of Adaptive Signal Processing”, Engineering Science and Education Journal, Vol. 9 Issue 4, Aug. 2000, pp. 161-168
- [6] Widrow, B., and Stearns, S., “Adaptive Signal Processing”, Prentice-Hall, 1985.

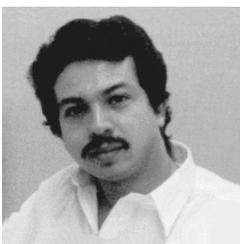
VIII. ACKNOWLEDGEMENTS

This work was sponsored by NSF grant EIA-9906600 and ONR grant N00014-99-1-0952. Mr. Miguel Alonso Jr. is the recipient of a Graduate Research Fellowship from the National Science Foundation.



Miguel Alonso Jr. was born in Miami, FL in 1979. Mr. Alonso received a B.S. degree in Computer Engineering from Florida International University, Miami, FL, in 2001. He is currently a research assistant in the Digital Signal Processing Laboratory at Florida International University while pursuing a M.S. degree in Computer Engineering. His interests are in improvements in human computer interaction, signal processing, robotics and A.I.

Mr. Alonso is a member of Tau Beta Pi and Eta Kappa Nu.



Armando Barreto was born in Mexico City, Mexico, in 1963. Dr. Barreto obtained the degree of Ingeniero Mecanico-Electricista, from the National Autonomous University of Mexico (UNAM), in 1987. Dr. Barreto received his Master's Degree in Electrical Engineering from Florida International University in 1989 and the Ph. D. Degree from the University of Florida, in 1993. In 1994, after completing an appointment as Postdoctoral Fellow at the University of Florida, Dr. Barreto joined the faculty of the Electrical and Computer Engineering Department at Florida International University, where he established the Digital Signal Processing Laboratory. Dr. Barreto continues to lead the activities of the FIU DSP Lab, as an associate professor.

Dr. Barreto is a member of Eta Kappa Nu.