An Implementation of Adaptive Filters with the TMS320C25 or the TMS320C30

APPLICATION REPORT: SPRA116

Sen Kuo Northwestern Illinois University Chein Chen Digital Signal Processor Products Semiconductor Group Texas Instruments

Digital Signal Processing Solutions



IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Copyright © 1997, Texas Instruments Incorporated

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE (281) 274-2320

US TMS320 FAX (281) 274-2324

US TMS320 BBS (281) 274-2323

US TMS320 email dsph@ti.com

An Implementation of Adaptive Filters with the TMS320C25 or the TMS320C30

Abstract

Adaptive filtering techniques are necessary considerations when a specific signal output is desired but the coefficients of that filter cannot be determined at the outset. Sometimes this is because of changing line or transmission conditions. An adaptive filter is one which contains coefficients that are updated by an adaptive algorithm to optimize filter response to the desired performance criterion.

Two devices, the TMS320C25 and TMS320C30, combine the power, high speed, flexibility and architecture optimized for adaptive signal processing.

This book discusses the topic of adaptive filter implementation as they apply to these two processors.

The book begins with a description of the two parts of an adaptive filter: the filter and the adaptive algorithm. The book goes on to discuss:

- ☐ The applications of adaptive filters (including adaptive prediction, equalization, noise cancellation and echo cancellation).
- ☐ The implementation of adaptive structures and algorithms (including transversal structure with the LMS algorithm, symmetric transversal structure, lattice structure, and modified LMS algorithms)
- ☐ Implementation considerations (including dynamic range constraint, finite precision errors, and design issues)



	Software development (assembly function libraries, C function libraries, development process and environment)
The book also contains:	
	Tables showing transversal structure, symmetric transversal structure and lattice structure for both the TMS320C25 and TMS320C30 processors
	Extensive references
	Multiple appendices of sample code for both TMS320C25 and TMS320C30 processors



Product Support

World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. New users must register with TI&ME before they can access the data sheet archive. TI&ME allows users to build custom information pages and receive new product updates automatically via email.

Email

For technical issues or clarification on switching products, please send a detailed email to *dsph@ti.com*. Questions receive prompt attention and are usually answered within one business day.

Introduction

A filter selects or controls the characteristics of the signal it produces by conditioning the incoming signal. The coefficients of the filter determine its characteristics and output a priori in many cases. Often, a specific output is desired, but the coefficients of the filter cannot be determined at the outset. An example is an echo canceller; the desired output cancels the echo signal (an output result of zero when there is no other input signal). In this case, the coefficients cannot be determined initially since they depend on changing line or transmission conditions. For applications such as this, it is necessary to rely on adaptive filtering techniques.

An adaptive filter is a filter containing coefficients that are updated by an adaptive algorithm to optimize the filter's response to a desired performance criterion. In general, adaptive filters consist of two distinct parts: a filter, whose structure is designed to perform a desired processing function; and an adaptive algorithm, for adjusting the coefficients of that filter to improve its performance, as illustrated in Figure 1. The incoming signal, x(n), is weighted in a digital filter to produce an output, y(n). The adaptive algorithm adjusts the weights in the filter to minimize the error, e(n), between the filter output, y(n), and the desired response of the filter, d(n). Because of their robust performance in the unknown and time-variant environment, adaptive filters have been widely used from telecommunications to control.

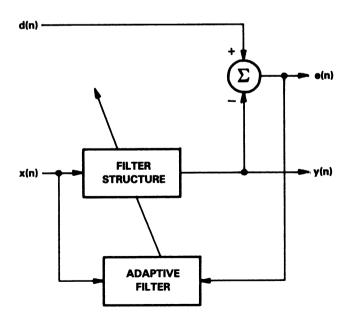


Figure 1. General Form of an Adaptive Filter

I I ... CALL FIL VIJ TMC220C25 J TMC220C22

Adaptive filters can be used in various applications with different input and output configurations. In many applications requiring real-time operation, such as adaptive prediction, channel equalization, echo cancellation, and noise cancellation, an adaptive filter implementation based on a programmable digital signal processor (DSP) has many advantages over other approaches such as a hard-wired adaptive filter. Not only are power, space, and manufacturing requirements greatly reduced, but also programmability provides flexibility for system upgrade and software improvement.

The early research on adaptive filters was concerned with adaptive antennas [1] and adaptive equalization of digital transmission systems [2]. Much of the reported research on the adaptive filter has been based on Widrow's well-known Least Mean Square (LMS) algorithm, because the LMS algorithm is relatively simple to design and implement, and it is well-understood and well-suited for many applications. All the filter structures and update algorithms discussed in this application report are Finite Impulse Response (FIR) filter structures and LMS-type algorithms. However, for a particular application, adaptive filters can be implemented in a variety of structures and adaptation algorithms [1, 3 through 9]. These structures and algorithms generally trade increased complexity for improved performance. An interactive software package to evaluate the performance of adaptive filters has also been developed [10].

The complexity of an adaptive filter implementation is usually measured in terms of its multiplication rate and storage requirement. However, the data flow and data manipulation capabilities of a DSP are also major factors in implementing adaptive filter systems. Parallel hardware multiplier, pipeline architecture, and fast on-chip memory size are major features of most DSPs [11, 12] and can make filter implementation more efficient.

Two such devices, the TMS320C25 and TMS320C30 from Texas Instruments [13, 14], have been chosen as the processors for fixed-point and floating-point arithmetic. They combine the power, high speed, flexibility, and an architecture optimized for adaptive signal processing. The instruction execution time is 80 ns for the TMS320C25 and only 60 ns for the TMS320C30. Most instructions execute in a single cycle, and the architectures of both processors make it possible to execute more than one operation per instruction. For example, in one instruction, the TMS320C25 processor can generate an instruction address and fetch that instruction, decode the instruction, perform one or two data moves (if the second data is from program memory), update one address pointer, and perform one or two computations (multiplication and accumulation). These processors are designed for real-time tasks in telecommunications, speech processing, image processing, and high-speed control, etc.

To direct the present research toward realistic real-time applications, three adaptive structures were implemented:

- 1. Transversal
- 2. Symmetric transversal
- 3. Lattice

Each structure utilizes five different update algorithms:

- 1. LMS
- 2. Normalized LMS
- 3. Leaky LMS
- 4. Sign-error LMS
- 5. Sign-sign LMS

Each structure with its adaptation algorithms is implemented using the TMS320C25 with fixed-point arithmetic and the TMS320C30 with floating-point arithmetic. The processor assembly code is included in the Appendix for each implementation. The assembly code for each structure and adaptation strategy can be readily modified by the reader to fit his/her applications and could be incorporated into a C function library as callable routines.

In this application report, the applications of adaptive filters, such as adaptive prediction, adaptive equalization, adaptive echo cancellation, and adaptive noise cancellation are presented first. Next, the implementation of the three filter structures and five adaptive algorithms with the TMS320C25 and TMS320C30 is described. This is followed by the practical considerations on the implementation of these adaptive filters. The remainder of the application report covers coding options, such as the routine libraries that support both assembly and C languages.

Applications of Adaptive Filters

The most important feature of an adaptive filter is the ability to operate effectively in an unknown environment and track time-varying characteristics of the input signal. The adaptive filter has been successfully applied to communications, radar, sonar, control, and image processing. Figure 1 illustrates a general form of an adaptive filter with input signals, x(n) and d(n), output signal, y(n), and error signal, e(n), which is the difference between the desired signal, d(n), and output signal, y(n). The adaptive filter can be used in different applications with different input/output configurations. In this section we briefly discuss several potential applications for the adaptive filters [15].

Adaptive Prediction

Adaptive prediction [16 through 18] is illustrated in Figure 2. In the general application of adaptive prediction, the signals are x(n) — delayed version of original signal, d(n) — original input signal, y(n) — predicted signal, and e(n) — prediction error or residual.

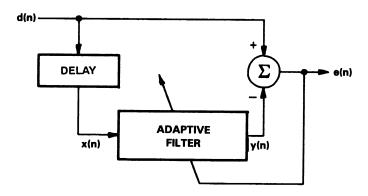


Figure 2. Block Diagram of an Adaptive Predictor

A major application of the adaptive prediction is the waveform coding of a speech signal. The adaptive filter is designed to exploit the correlation between adjacent samples of the speech signal so that the prediction error is much smaller than the input signal on the average. This prediction error signal is quantized and sent to the receiver in order to reduce the number of bits required for the transmission. This type of waveform coding is called Adaptive Differential Pulse-Code Modulation (ADPCM) [17] and provides data rate compression of the speech at 32 kb/s with toll quality. More recently, in certain online applications, time recursive modeling algorithms have been proposed to facilitate speech modeling and analysis.

The coefficients of the adaptive predictor can be used as the autoregressive (AR) parameters of the nonstationary model. The equation of the AR process is

$$u(n) = a_1^* u(n-1) + a_2^* u(n-2) + \dots + a_m^* u(n-m) + v(n)$$

where a_1, a_2, \ldots, a_m are the AR parameters. Thus, the present value of the process u(n) equals a finite linear combination of past values of the process plus an error term v(n). This adaptive AR model provides a practical means to measure the instantaneous frequency of input signal. The adaptive predictor can also be used to detect and enhance a narrow band signal embedded in broad band noise. This Adaptive Line Enhancer (ALE) provides at its output y(n) a sinusoid with an enhanced signal-to-noise ratio, while the sinusoidal components are reduced at the error output e(n).

Adaptive Equalization

Figure 3 shows another model known as adaptive equalization [2, 9, 15]. The signals in the adaptive equalization model are defined as x(n) — received signal (filtered version of transmitted signal) plus channel noise, d(n) — detected data signal (data mode) or pseudo random number (training mode), y(n) — equalized signal used to detect received data, and e(n) — residual intersymbol interference plus noise.

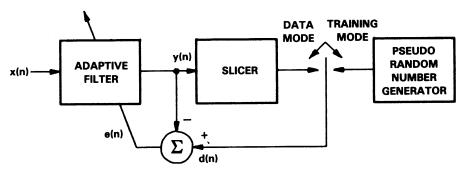


Figure 3. Block Diagram of an Adaptive Equalizer

The use of adaptive equalization to eliminate the amplitude and phase distortion introduced by the communication channel was one of the first applications of adaptive filtering in telecommunications [19]. The effect of each symbol transmitted over a time-dispersive channel extends beyond the time interval used to represent that symbol, resulting in an overlay of received symbols. Since most channels are time-varying and unknown in advance, the adaptive channel equalizer is designed to deal with this intersymbol interference and is widely used for bandwidth-efficient transmission over telephone and radio channels.

Adaptive Echo Cancellation

Another application, known as adaptive echo cancellation [20, 21] is shown in Figure 4. In this application, the signals are identified as x(n) – far-end signal, d(n) – echo of far-end signal plus near-end signal, y(n) – estimated echo of far-end signal, and e(n) – near-end signal plus residual echo.

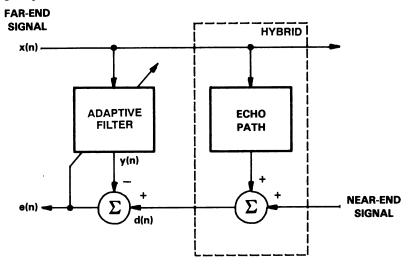


Figure 4. Block Diagram of an Echo Canceller

The adaptive echo cancellers are used in practical applications of cancelling echoes for long-distance telephone voice communication, full-duplex voiceband data modems, and high-performance audio-conferencing systems. To overcome the echo problem, echo cancellers are installed at both ends of the network. The cancellation is achieved by estimating the echo and subtracting it from the return signal.

Adaptive Noise Cancellation

One of the simplest and most effective adaptive signal processing techniques is adaptive noise cancelling [1, 22]. As shown in Figure 5, the primary input d(n) contains both signal and noise, where x(n) is the noise reference input. An adaptive filter is used to estimate the noise in d(n) and the noise estimate y(n) is then subtracted from the primary channel. The noise cancellation output is then the error signal e(n).

The applications of noise cancellation include the cancellation of various forms of interference in electrocardiography, noise in speech signals, noise in fighter cockpit environments, antennas sidelobe interference, and the elimination of 60-Hz hum. In the majority of these noise cancellation applications, the LMS algorithm has been utilized.

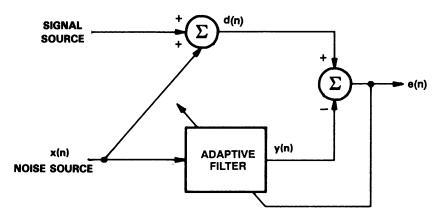


Figure 5. General Form of a Noise Canceller

Application Summary

The above list of applications is not exhaustive and is limited primarily to applications within the field of telecommunications. Adaptive filtering has been used extensively in the context of many other fields including, but not limited to, instantaneous frequency tracking, intrusion detection, acoustic Doppler extraction, on-line system identification, geophysical signal processing, biomedical signal processing, the elimination of radar clutter, beamforming, sonar processing, active sound cancellation, and adaptive control.

Implementation of Adaptive Structures and Algorithms

Several types of filter structures can be implemented in the design of the adaptive filters such as Infinite Impulse Response (IIR) or Finite Impulse Response (FIR). An adaptive IIR filter [1, 5], with poles as well as zeros, makes it possible to offer the same filter characteristics as the FIR filter with lower filter complexity. However, the major problem with adaptive IIR filter is the possible instability of the filter if the poles move outside the unit circle during the adaptive process. In this application report, only FIR structure is implemented to guarantee filter stability.

An adaptive FIR filter can be realized using transversal, symmetric transversal, and lattice structures. In this section, the adaptive transversal filter with the LMS algorithm is introduced and implemented first to provide a working knowledge of adaptive filters.

Transversal Structure with LMS Algorithm

Transversal Structure Filter

The most common implementation of the adaptive filter is the transversal structure (tapped delay line) illustrated in Figure 6. The filter output signal y(n) is

$$y(n) = \underline{w}^{T}(n)\underline{x}(n) = \sum_{i=0}^{N-1} w_{i}(n) x(n-i)$$
(1)

where $\underline{x}(n) = [x(n) \ x(n-1) \ \dots \ x(n-N+1)]^T$ is the input vector, $\underline{w}(n) = [w_0(n) \ w_1(n) \ \dots \ w_{N-1}(n)]^T$ is the weight vector, T denotes transpose, n is the time index, and N is the order of filter. This example is in the form of a finite impulse response filter as well as the convolution (inner product) of two vectors $\underline{x}(n)$ and $\underline{w}(n)$. The implementation of Equation (1) is illustrated using the following C program:

$$y[n] = 0.;$$

for (i = 0; i < N; i++) {
 $y[n] += wn[i]*xn[i];$

where wn [i] denotes wi(n) and xn[i] represents x(n-i).

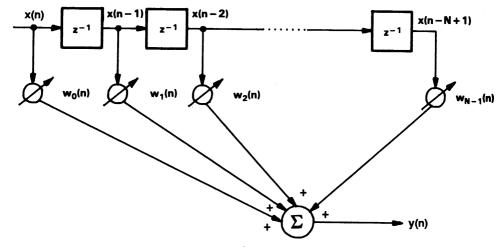


Figure 6. Transversal Filter Structure

TMS320C25 Implementation

The architecture of TMS320C25 [13] is optimized to implement the FIR filter. After execution of the CNFP (Configure Block B0 as Program Memory) instruction, the filter coefficients $w_i(n)$ from RAM block B0 (via program bus) and data x(n-i) from RAM block B1 (via data bus) are available simultaneously for the parallel multiplier (see Figure 7).

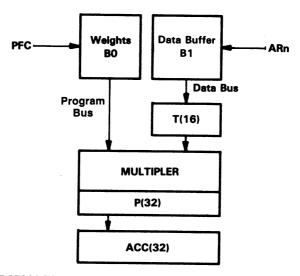


Figure 7. TMS320C25 Arithmetic Unit (after execute CNFP instruction)

The MACD instruction enables complete multiply/accumulate, data move, and pointer update operations to be completed in a single instruction cycle (80 ns) if filter coefficients are stored in on-chip RAM or ROM or in off-chip program memory with zero wait states. Since the adaptive weights $w_i(n)$ need to be updated in every iteration, the filter coefficients must be stored in RAM. The implementation of the inner product in Equation (1) can be made even more efficient with a repeat instruction, RPTK. An N-weight transversal filter can be implemented as follows [23]:

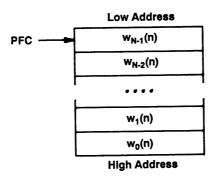
LARP ARn

LRLK ARn,LASTAP

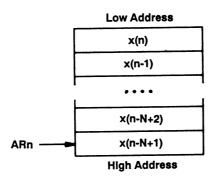
RPTK N-1

MACD COEFFP,*
(A)

Where ARn is an auxiliary address register that points to x(n-N+1), and the Prefetch Counter (PFC) points to the last weight $w_{N-1}(n)$ indicated by COEFFP. When the MACD instruction is repeated, the coefficient address is transferred to the PFC and is incremented by one during its operation. Therefore, the components of weight vector $\underline{w}(n)$ are stored in B0 as



The MACD in repeat mode will also copy data pointed to by ARn, to the next higher on-chip RAM location. The buffer memories of transversal filter are therefore stored as



In general, roundoff noise occurs after each multiplication. However, the TMS320C25 has a 16×16 -bit multiplier and a 32-bit accumulator, so there is no roundoff during the summing of a set of product terms in Program (A). All multiplication products are represented in full precision, and rounding is performed after they are summed. Thus y(n) is obtained from the accumulator with only one roundoff, which minimizes the roundoff noise in the output y(n). Since both the tapped delay line and the adaptive weights are stored in data RAM to achieve the fastest throughput, the highest transversal filter order for efficient implementation on the TMS320C25 is 256. However, if necessary, higher order filters can be implemented by using external data RAM.

TMS320C30 Implementation

The architecture of TMS320C30 [14] is quite different from TI's second generation processors. Instead of using program/data memory, it provides two data address buses to do the data memory manipulations. This feature allows two data memory addresses to be generated at the same time. Hence, parallel data store, load, or one data store with one data load can be done simultaneously. Such capabilities make the programming much easier and more flexible. Since the hardware multiplier and arithmetic logic unit (ALU) of TMS320C30 are separated, with proper operand arrangement, the processor can do one multiplication and one addition or subtraction at the same time. With these two combined features, the TMS320C30 can execute several other parallel instructions. These parallel instructions can be found in Section 11 of the *Third-Generation TMS320 User's Guide* [14]. Associating with single repeat instruction RPTS, an inner product in Equation (1) can be implemented as follows:

where auxiliary registers AR0 and AR1 point to x and w arrays. The addition in the parallel instruction sums the previous values of R1 and R2. Therefore, R1 is initialized with the first product prior to the repeat instruction RPTS.

Note that the implementation above does not move the data in the x array like MACD does in TMS320C25. For filter delay taps, the TMS320C30 uses a circular buffer method to implement the delay line. This method reserves a certain size of memory for the buffer and uses a pointer to indicate the beginning of the buffer. Instead of moving data to next memory location, the pointer is updated to point to the previous memory location. Therefore, from the new beginning of the buffer, it has the effect of the tapped delay line. When the value of the pointer exceeds the end of the buffer, it will be circled around to the other end of the buffer. It works just like joining two ends of the buffer together as a necklace. Thus, new data is within the circular queue, pointed to by ARO, replacing

the oldest value. However, from an adaptive filter point of view, data doesn't have to be moved at this point yet.

TMS320C30 has a 32-bit floating point multiplier and the result from the multiplier is put and accumulated into a 40-bit extended precision register. If the input from A/D converter is equal to or less than 16 bits, there is no roundoff noise after multiplication. Theoretically, the TMS320C30 can implement a very high order of adaptive filter. However, for the most efficient implementation, the limitation of filter order is 2K because the TMS320C30 external data write requires at least two cycles. If the filter coefficients are put in somewhere other than internal data RAM, the instruction cycles will be increased.

LMS Adaptation Algorithm

The adaptation algorithm uses the error signal

$$e(n) = d(n) - y(n), \tag{2}$$

where d(n) is the desired signal and y(n) is the filter output. The input vector $\underline{x}(n)$ and e(n) are used to update the adaptive filter coefficients according to a criterion that is to be minimized. The criterion employed in this section is the mean-square error (MSE) ϵ :

$$\epsilon = \mathrm{E}[\mathrm{e}^2(\mathrm{n})] \tag{3}$$

where E[.] denotes the expectation operator. If y(n) from Equation (1) is substituted into Equation (2), then Equation (3) can be expressed as

$$\epsilon = E[d^{2}(n)] + \underline{w}^{T}(n)R\underline{w}(n) - 2\underline{w}^{T}(n)\underline{p}$$
(4)

where $R = E[x(n)x^T(n)]$ is the N x N autocorrelation matrix, which indicates the sample-to-sample correlation within a signal, and $p = E[d(n) \underline{x}(n)]$ is the N x 1 cross-correlation vector, which indicates the correlation between the desired signal d(n) and the input signal vector $\underline{x}(n)$.

The optimum solution $w^* = [w_0^* \ w_1^* \ ... \ w_{N-1}^*]^T$, which minimizes MSE, is derived by solving the equation

$$\frac{\delta \epsilon}{\delta w(n)} = 0 \tag{5}$$

This leads to the normal equation

$$R w^* = p \tag{6}$$

If the R matrix has full rank (i.e., R-1 exists), the optimum weights are obtained by

$$\underline{\mathbf{w}}^* = \mathbf{R}^{-1} \, \underline{\mathbf{p}} \tag{7}$$

In Linear Predictive Coding (LPC) of a speech signal, the input speech is divided into short segments, the quantities of R and p are estimated, and the optimal weights corresponding to each segment are computed. This procedure is called a block-by-block data-adaptive algorithm [24].

A widely used LMS algorithm is an alternative algorithm that adapts the weights on a sample-by-sample basis. Since this method can avoid the complicated computation of R^{-1} and p, this algorithm is a practical method for finding close approximate solutions to Equation (7) in real time. The LMS algorithm is the steepest descent method in which the next weight vector w(n+1) is increased by a change proportional to the negative gradient of mean-square-error performance surface in Equation (7)

$$\underline{\mathbf{w}}(\mathbf{n}+1) = \underline{\mathbf{w}}(\mathbf{n}) - \mathbf{u}\underline{\nabla} \ (\mathbf{n}) \tag{8}$$

where u is the adaptation step size that controls the stability and the convergence rate. For the LMS algorithm, the gradient at the nth iteration, ∇ (n), is estimated by assuming squared error $e^2(n)$ as an estimate of the MSE in Equation (3). Thus, the expression for the gradient estimate can be simplified to

$$\underline{\nabla}(\mathbf{n}) = \frac{\delta[\mathbf{e}^2(\mathbf{n})]}{\delta \underline{\mathbf{w}}(\mathbf{n})} = -2 \ \mathbf{e}(\mathbf{n}) \ \underline{\mathbf{x}}(\mathbf{n}) \tag{9}$$

Substitution of this instantaneous gradient estimate into Equation (8) yields the Widrow-Hoff LMS algorithm

$$\underline{\mathbf{w}}(\mathbf{n}+1) = \underline{\mathbf{w}}(\mathbf{n}) + 2 \mathbf{u} \mathbf{e}(\mathbf{n}) \underline{\mathbf{x}}(\mathbf{n})$$
 (10)

where 2 u in Equation (10) is usually replaced by u in practical implementation.

Starting with an arbitrary initial weight vector $\underline{\mathbf{w}}(0)$, the weight vector $\underline{\mathbf{w}}(\mathbf{n})$ will converge to its optimal solution $\underline{\mathbf{w}}^*$, provided u is selected such that [1]

$$0 < u < \frac{1}{\lambda_{\text{max}}} \tag{11}$$

where λ_{max} is the largest eigenvalue of the matrix R. λ_{max} can be bounded by

$$\lambda_{\text{max}} < \text{Tr} [R] = \sum_{i=0}^{N-1} r(0) = N r(0)$$
 (12)

where Tr [.] denotes the trace of a matrix and $r(0) = E[x^2(n)]$ is average input power.

For adaptive signal processing applications, the most important practical consideration is the speed of convergence, which determines the ability of the filter to track nonstationary signals. Generally speaking, weight vector convergence is attained only when the slowest weight has converged. The time constant of the slowest mode is [1]

$$t = \frac{1}{u\lambda_{min}} \tag{13}$$

This indicates that the time constant for weight convergence is inversely proportional to u and also depends on the eigenvalues of the autocorrelation matrix of the input. With the disparate eigenvalues, i.e., $\lambda_{max} >> \lambda_{min}$, the setting time is limited by the slowest mode, λ_{min} . Figure 8 shows the relaxation of the mean square error from its initial value ϵ_0 toward the optimal value ϵ_{min} .

Adaptation based on a gradient estimate results in noise in the weight vector, therefore a loss in performance. This noise in the adaptive process causes the steady state weight vector to vary randomly about the optimum weight vector. The accuracy of weight vector in steady state is measured by excess mean square error (excess MSE = E [$\epsilon - \epsilon_{min}$]). The excess MSE in the LMS algorithm [1] is

excess MSE =
$$u Tr[R] \epsilon_{min}$$
 (14)

where ϵ_{min} is minimum MSE in the steady state.

Equations (13) and (14) yield the basic trade-off of the LMS algorithm: to obtain high accuracy (low excess MSE) in the steady state, a small value of u is required, but this will slow down the convergence rate. Further discussions of the characteristics and properties of the LMS algorithm are presented in [1, 3 through 9]. The implementations of LMS algorithm with the TMS320C25 and TMS320C30 are presented next.

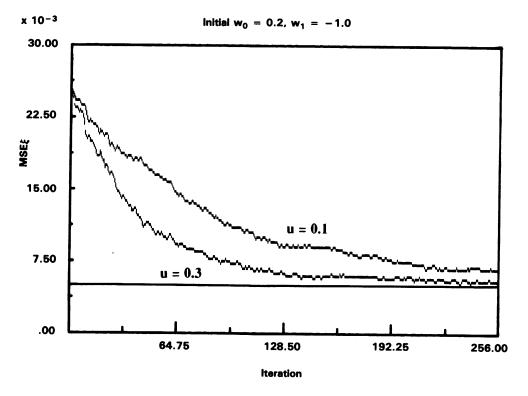


Figure 8. Learning Curve of an Adaptive Transversal Filter and an LMS Algorithm with Different Step Sizes

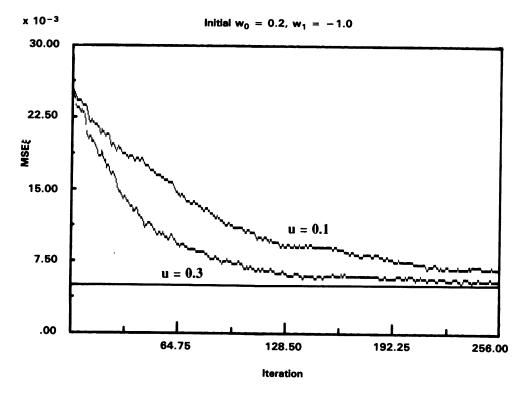


Figure 8. Learning Curve of an Adaptive Transversal Filter and an LMS Algorithm with Different Step Sizes

Since $u^*e(n)$ is constant for N weights update, the error signal e(n) is first multiplied by u to get ue(n). This constant can be computed first and then multiplied by x(n) to update w(n). An implementation method of the LMS algorithm in Equation (10) is illustrated as

```
ue(n) = u*e[n];

for (i=0; i<N; i++) {

    wn[i] += uen * xn[i];

}
```

TMS320C25 Implementation

The TMS320C25 provides two powerful instructions (ZALR and MPYA) to perform the update example in Equation (10).

- ZALR loads a data memory value into the high-order half of the accumulator while rounding the value by setting bit 15 of the accumulator to one and setting bits 0-14 of the accumulator to zero. The rounding is necessary because it can reduce the roundoff noise from multiplication.
- MPYA accumulates the previous product in the P register and multiplies the operand with the data in T register.

Assuming that ue(n) is stored in T and the address pointer is pointing to AR3, the adaptation of each weight is shown in the following instruction sequence:

```
; Initialize loop counter
       LRLK AR1,N-1
                                ; Point to w_{N-1}(n)
       LRLK AR2,COEFFD
                                ; Point to x(n-N+1), since MACD in (A)
       LRLK AR3,LASTAP+1
                                ; Already moved elements of current
                                ; x(n) to the next higher location
                                P = ue(n) * x(n-N+1)
       MPY *-,AR2
                                ; Load wi(n) and round
ADAP ZALR *,AR3
                                ; ACC=P+w_i(n) and P=ue(n) * x(n-i)
       MPYA *-,AR2
                                ; Store w_i(n+1)
       SACH *+,0,AR1
                                : Test loop counter, if counter not
       BANZ ADAP,*-,AR2
                                : Equal to 0, decrement counter,
                                : Branch to ADAP and select AR2 as
                                : Next pointer.
```

For each iteration, N instruction cycles are needed to perform Equation (1), 6N instruction cycles are needed to perform weight updates in Equation (10), and the total number of instruction cycles needed is 7N+28. An example of a TMS320C25 program implementing a LMS transversal filter is presented in Appendix A1. Note that BANZ needs three instruction cycles to execute. This can be avoided by using straight line code, which requires 4N+33 instruction cycles [25].

TMS320C30 Implementation

Although the TMS320C30 doesn't provide any specific instruction for adaptive filter coefficients update, it still can achieve the weight updating in two instructions because of its powerful architecture. The TMS320C30 has a repeat block instruction RPTB, which allows a block of instructions to be repeated a number of times without any penalty for looping. A single repeat mode, RM, in the status register, ST, and three registers – repeat start address (RS), repeat end address (RE), and repeat counter (RC) – control the block repeat. When RM is set, the PC repeats the instructions between RS and RE a number of times, which is determined by the value of RC. The repeat modes repeat a block of code at least once in a typical operation. The repeat counter should be loaded with one less than the desired number of repetitions. Assuming the error signal e(n) in Equation (10) is stored in R7, the adaptation of filter coefficients is shown as follows:

```
*AR0++(1)\%,R7,R1
       MPYF3
                                    R1 = u*e(n)*x(n)
       LDI
                order - 3.RC
                                    ; Initialize repeat counter
       RPTB
                LMS
                                    ; Do i = 0, N-3
       MPYF3
               *AR0++(1)\%,R7,R1
                                    ; Compute u*e(n)*x(n-i-1)
     ADDF3
                *AR1,R1,R2
                                    ; Compute wi(n) + u*e(n)*x(n-i)
LMS
       STF
               R2,*AR1++(1)\%
                                    ; Store wi(n+1)
       MPYF3
               *AR0,R7,R1
                                   ; For i = N-2
     | ADDF3
               *AR1,R1,R2
       STF
               R2,*AR1++(1)\%
                                   ; Store wN-2(n+1)
       ADDF3
               *AR1,R1,R2
                                   ; Include last w
       STF
               R2,*AR1++(1)\%
                                   ; Store wN-1(n+1)
```

where auxiliary register AR0 and AR1 point to x and w arrays. R1 is updated before loop since the accumulation in the parallel instruction uses the previous value in R1. In order to update x array pointer to the new beginning of the data buffer for next iteration (i.e., perform the data move), one of the loop instruction set has been taken out of loop and modified by eliminating the incrementation of AR0.

To perform an N-weight adaptive LMS transversal filter on TMS320C30 requires 3N+15 instruction cycles. There are N and 2N instruction cycles to perform Equations (1) and (10), respectively. The TMS320C30 example program is given in Appendix A2.

The LMS algorithm considerably reduces the computational requirements by using a simplified mean square error estimator (an estimate of the gradient). This algorithm has proved useful and effective in many applications. However, it has several limitations in performance such as the slow initial convergence, the undesirable dependence of its convergence rate on input signal statistics, and an excess mean square error still in existence after convergence.

Symmetric Transversal Structure [5]

A transversal filter with symmetric impulse response (weight values) about the center weight has a linear phase response. In applications such as speech processing, linear phase filters are preferred since they avoid phase distortion by causing all the components in the filter input to be delayed by the same amount. The adaptive symmetric transversal structure is shown in Figure 9.

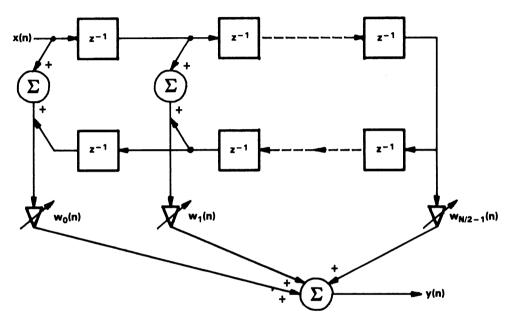


Figure 9. Symmetric Transversal Structure (even order)

This filter is actually an FIR filter with an impulse response that is symmetric about the center tap. The output of the filter is obtained as

$$y(n) = \sum_{i=0}^{N/2-1} w_i(n) [x(n-i) + x(n-N+i+1)]$$
 (15a)

where N is an even number. Note that, for fixed-point processors, the addition in the brackets may introduce overflow because the input signals x(n-i) and x(n-N+i+1) are in the range of -1 and $1-2^{-15}$. This problem can be solved by shifting x(n) to the right one bit. The update of the weight vector is

$$w_i(n+1) = w_i(n) + ue(n)[x(n-1) + x(n-N+i+1)]$$
 (15b)

for $i=0,1,...,(N/2^{-1})$, which requires N/2 multiplications and N additions. Theoretically, this symmetric structure can also reduce computational complexity since such filters require only half the multiplications of the general transversal filter. However, it is true only for the TMS320C30 processor. When a filter is implemented on the TMS320C25, the transversal structure is more efficient than the symmetric transversal structure due to the pipeline multiplication and accumulation instruction MACD, which is optimized to implement convolution in Equation (1).

TMS320C25 Implementation

For TMS320C25, in order to implement the instructions MAC, ZALR, and MPYA, we can trade memory requirements for computation saving by defining

$$z(n-i) = x(n-i) + x(n-N+i+1), i=0,1,...,N/2-1$$
 (16a)

Now, Equation (15) can be expressed as

$$y(n) = \sum_{i=0}^{N/2-1} w_i(n) \ z(n-i)$$
 (16b)

$$w_i(n+1) = w_i(n) + u e(n) z(n-i), i=0,1,...,N/2^{-1}$$
 (16c)

Equation (16a) can be implemented using the TMS320C25 as

```
LARK
                 AR1, N/2-1 ; Counter = N/2^{-1}
                 AR2,LAST_X; Point to x(n-N+1)
         LRLK
         LRLK
                 AR3,FIRST_X; Point to x(n)
                 AR4.FIRST_Z; Point to z(n)
         LRLK
         LARP
                 AR3
SYM
         LAC
                 *+,0,AR2
         ADD
                 *-,0,AR4
         SACL
                 *+,0,AR1
                 SYM,*-,AR3
         BANZ.
```

The instruction sequence to implement the LMS algorithm in Equations (1) and (10) can be used to implement Equations (16b) and (16c), except using MAC instead of MACD in Program (A). Therefore, N instruction cycles are needed to shift data in x(n), 3N instruction cycles are needed to implement Equation (16a), N/2 for Equation (16b), and 3N for Equation (16c). The total number of instruction cycles required to implement the symmetric transversal filter with the LMS algorithm is 7.5N+38. Where 7.5N is an integer because N is chosen as an even number. The 0.5N instruction cycles come from Equation (15a) since symmetric transversal structure folds the filter taps into half of the order N (see Figure 9). The maximum filter length for most efficient code, 256, is the

same as for the FIR filter. The use of the additional data memory can be obtained from the reduced data memory requirement for weights of the symmetric transversal filter. The complete TMS320C25 program is given in Appendix B1.

Note that instead of storing buffer locations x(n) contiguously, then using DMOV to shift data in the buffer memory (requiring N cycles) at the end of each iteration, we can use a circular buffer with pointers pointing to x(n) and x(n-N+1). Since pointer updating requires several instruction cycles, compared with N cycles using DMOV to update the buffer memory contents, the circular buffer technique is more efficient if N is large.

TMS320C30 Implementation

As mentioned above, the TMS320C30 uses a circular buffer instead of data move technique. Therefore, it does not have to implement tapped delay line separately as TMS320C25. Equations (1) and (16a) can be combined and implemented in the same loop. The advantage of this is that a parallel instruction reduces the number of the instruction cycles. The implementation is shown as follows:

```
LDF
                  0.0, R2
                                                : Clear R2
                  order/2-2,RC
                                                ; Set up loop counter
        LDI
                                               : Do i = 0, N/2^{-2}
        RPTB
                  INNER
        ADDF3 *AR4++(1)%,*AR5--(1)%,R1; z(i) = x(n-i) + x(n+N-i)
                  R1,*AR1++(1),R3
R1,*AR2++(1)
                                               : R3 = w[] * z[]
        MPYF3
      | STF
                                               ; Store z(i)
                                               ; Accumulate the result for y
INNER
        ADDF3
                 R3,R2,R2
                  *AR4 + +(1)\%, *AR5 - -(1)\%, R1; For i = N/2 - 1
        ADDF3
                  R1,*AR1 - -(IR0),R3
        MPYF3
                  R1.*AR2 - -(IR0)
      STF
                  R3,R2,R2
                                               ; Include last product
        ADDF3
```

where AR4 and AR5 point to x[0] and x[N-1]. AR1 and AR2 point to w and z array, respectively. IR0 contains value of $N/2^{-1}$. The same instruction codes of weight update of transversal filter can be used in symmetric transversal structure by changing the x array pointer to the z array pointer. Appendix B2 presents an example program. The total number of instructions needed is 2.5N+15, which is less than that of the transversal structure.

Lattice Structure [6]

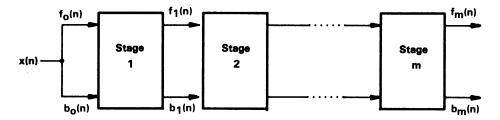
An alternative FIR filter realization is the lattice structure [26]. A discussion of the transversal filter with the LMS algorithm shows that the convergence rate of the transversal structure is restricted by the correlation of signal components; i.e., the eigenvalue spread, $\lambda_{max}/\lambda_{min}$. The lattice structure is a decorrelating transform based on a family of prediction error filters as illustrated in Figure 10. The recursive equations that describe the lattice predictor are

$$f_0(n) = b_0(n) = x(n)$$
 (17a)

$$f_m(n) = f_{m-1}(n) - k_m(n)b_{m-1}(n-1), 0 < m < M$$
 (17b)

$$b_m(n) = b_{m-1}(n-1) - k_m(n)f_{m-1}(n), 0 < m < M$$
 (17c)

where $f_m(n)$ represents the forward prediction error, $b_m(n)$ represents the backward prediction error, $k_m(n)$ is the reflection coefficients, m is the stage index, and M is the number of cascaded stages. The lattice structure has the advantage of being order-recursive. This property allows adding or deleting of stages from the lattice without affecting the existing stages.



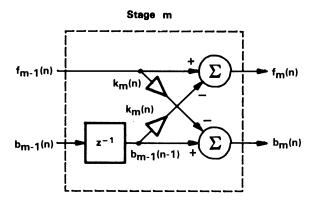


Figure 10. Lattice Structure

To implement the lattice filter for processing actual data, the reflection coefficients $k_m(n)$ are required. These coefficients can be computed according to estimates of the autocorrelation coefficients using Durbin's algorithm. However, it would be more efficient if these reflection coefficients could be estimated directly from the data and updated on a sample-by-sample basis, such as LMS algorithm [6]. The reflection coefficient $k_m(n+1)$ can be recursively computed [7]:

$$k_m(n+1) = k_m(n) + u[f_m(n)b_{m-1}(n-1) + b_m(n)f_{m-1}(n)], 0 < m < M$$
 (18)

For applications such as noise cancellation, channel equalization, line enhancement, etc., the joint-process estimation [3] illustrated in Figure 11 is required. This device performs two optimum estimations: the lattice predictor and the multiple regression filter. The following equations define the implementation of the regression filter

$$e_0(n) = d(n) - b_0(n)g_0(n)$$
 (19a)

$$e_m(n) = e_{m-1}(n) - b_{m-1}(n)g_{m-1}(n), 0 < m < M$$
 (19b)

$$g_m(n+1) = g_m(n) + u_{em}(n)b_m(n), \quad 0 <= m <= M$$
 (20)

where the LMS algorithm is used to update the coefficients of the regression filter. For noise cancellation application, $e_m(n)$ corresponds to the output e(n) in Figure 5. For applications such as adaptive line enhancer and channel equalizer, filter output y(n) is obtained as

$$y(n) = \sum_{m=0}^{M} g_m(n) b_m(n)$$
 (21)

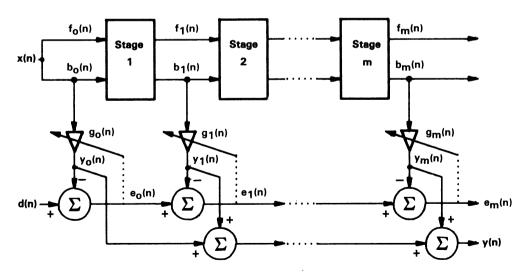


Figure 11. Lattice Structure with Joint Process Estimation

TMS320C25/TMS320C30 Implementation

There are five memory locations— $f_m(n)$, $b_m(n)$, $b_m(n-1)$, $k_m(n)$, and $g_m(n)$ —required for each stage. The limitation of on-chip data RAM is 544 words for the TMS320C25 and 2K words for the TMS320C30. A maximum of 102 stages can therefore be implemented on a single TMS320C25 for the highest throughput. Here, another advantage of TMS320C30 architecture design is shown. Since the operands of the mathematic operations can be either memory or register on the TMS320C30, and there is no need to preserve the values of f_m array for the next iteration (refer to Equations (17) and (18)), the f_m array can be replaced by an extended precision register. Thus, for the most efficient codes, the stage limitation of lattice structure for TMS320C30 is 512, or one-fourth of the 2K on-chip RAM.

Lattice structures have superior convergence properties relative to transversal structures and good stability properties; e.g., low sensitivity to coefficient quantization, low roundoff noise, and the ability to check stability by inspection. The disadvantages of lattice filter algorithms are that they are numerically complex and require mathematical sophistication to thoroughly understand their derivations. Furthermore, as shown in Appendixes C1 and C2, lattice structures cannot take advantage of the TMS320C25 and TMS320C30's pipeline architecture to achieve high throughput. The total number of instruction cycles needed is 33M+32 for TMS320C25 and 14M+4 for TMS320C30.

Modified LMS Algorithms [5]

The LMS algorithm described in previous sections is the most widely used algorithm in practical applications today. In this section, a set of LMS-type algorithms (all direct variants of the LMS algorithm) are presented and implemented. The motivation for each is some practical consideration, such as faster convergence, simplicity in implementation, or robustness in operation. The description of these algorithms is based on the transversal structure. However, these algorithms can be applied to the symmetric transversal structure and the lattice structure as well.

Normalized LMS Algorithm

The stability, convergence time, and fluctuation of the adaptation process is governed by the step size u and the input power to the adaptive filter. In some practical applications, you may need an automatic gain control (AGC) on the input to the adaptive filter. The normalized LMS algorithm is one important technique used to improve the speed of convergence. This is accomplished while maintaining the steady-state performance independent of the input signal power. This algorithm uses a variable convergence factor u(n), which represents a u that is a function of the time index,

$$\mathbf{u}(\mathbf{n}) = a / \mathbf{var}(\mathbf{n}) \tag{22}$$

$$\underline{\mathbf{w}}(\mathbf{n}+1) = \mathbf{w}(\mathbf{n}) + \mathbf{u}(\mathbf{n})\mathbf{e}(\mathbf{n})\mathbf{x}(\mathbf{n}) \tag{23}$$

where a is a convergence parameter, and var(n) is an estimate of the input average power at time n using the recursive equation

$$var(n) = (1 - b) var(n-1) + b x^{2} (n)$$
 (24)

where 0 < b < 1 is a smoothing parameter. In practice, a is chosen equal to b.

For fixed-point processors, there is a way to reduce the computation of power estimation. Since b in Equation (24) doesn't have to be an exact number, it is computationally convenient to make b a power of 2. If $b = 2^{-m}$, the multiplication of b can be implemented by shifting right m bits. Therefore, the var(n) in Equation (24) is computed by

$$var(n) = var(n-1) - b var(n-1) + b x^{2}(n)$$

= $var(n-1) - var(n-1) * 2^{-m} + x^{2}(n) * 2^{-m}$

Then, assuming the variance var(n) of input signal is stored in the data memory VAR and its initial value is 0.99997 (= $1-2^{-15}$), The implementation of this equation using TMS320C25 assembly code is

```
LARP
        AR3
LRLK
        AR3,FRSTAP; Point to input signal x
SQRA
                      ; Square input signal
SPH
        ERRF
ZALH
        VAR
                      ; ACC = var(n-1)
        VAR,SHIFT
                      ; ACC = (1-b) var(n-1)
SUB
        ERRF,SHIFT
                      ACC = (1-b) var(n-1) + b x^{2}(n)
ADD
SACH
        VAR
                      ; Store var(n)
```

The normalized LMS algorithm can be implemented as

```
var = b_1 * var + b * xn[0] * xn[0];

unen = e[n] * a / var;

for (i = 0; i < N; i++)

wn[i] += unen * xn[i];
```

where $b_1 = (1-b)$, xn[0] = x(n), and unen = u(n)*e(n). This normalized technique reduces the dependency of convergence speed on input signal power at the cost of increased computational complexity, especially the division in Equation (22). The algorithms of implementing the fixed-point and floating-point division on the TMS320C25 and

TMS320C30 can be found in the user's guide for each device [13, 14]. Since the power of input signal is always positive, those codes can be simplified to save computation time.

Since the power estimation in Equation (24) and step size normalization in Equation (22) are performed once for each sample x(n), the computation increase can be ignored when N is large. As shown in Appendixes D1 and D2, the total number of instruction cycles needed for the normalized LMS algorithm (7N+57 for the TMS320C25 and 3N+47 for the TMS320C30) is slightly higher than for the LMS algorithm (7N+34 and 3N+15) when N is large.

Sign LMS Algorithms

The LMS algorithm requires 2N multiplications and additions for each iteration; this amount is much lower than the requirements for many other complicated adaptive algorithms, such as Kalman and Recursive Least Square (RLS) [3]. However, there are three simplified versions of the LMS algorithm (sign-error LMS, sign-data LMS, and sign-sign LMS) that save the number of multiplications required and extend the real-time bandwidth for some applications [5, 27].

First, the sign-error LMS algorithm can be expressed as

$$\underline{\mathbf{w}}(\mathbf{n}+1) = \underline{\mathbf{w}}(\mathbf{n}) + \mathbf{u} \operatorname{sign}[\mathbf{e}(\mathbf{n})] \underline{\mathbf{x}}(\mathbf{n})$$

$$\mathbf{e} \quad \operatorname{sign}[\mathbf{e}(\mathbf{n})] = 1, \text{ if } \mathbf{e}(\mathbf{n}) \ge 0$$

$$-1, \text{ if } \mathbf{e}(\mathbf{n}) < 0$$
(25)

where

The C program implementation of sign-error LMS algorithm is

```
\begin{array}{l} tu = u; \\ if (e[n] < 0.) \{ \\ tu = -u; \} \\ for (i=0; i < N; i++) \{ \\ wn[i] += tu * xn[i]; \} \end{array}
```

As shown in Appendixes E1 and E2, the instruction sequence to implement weight update with the sign-error LMS algorithm is identical to that with the LMS algorithm. The difference is that the sign-error LMS algorithm uses the sign $[e(n)]^*u$ instead of $e(n)^*u$ before the update loop. Note that, for fixed-point processors, if u is chosen to be a power of two, the u x(n) can be accomplished by shifting right the elements in x(n). This algorithm keeps the same convergence direction as the LMS algorithm. Thus, the sign-error LMS algorithm should remain efficient, provided the variable gain u(n) is matched to this change. However, the use of constant step size u to reduce computation comes at the expense of a slow convergence rate since smaller u is normally used for stability reasons.

The programs in Appendixes E1 and E2 implement a transversal filter with sign-error LMS algorithm in looped code. The total number of instruction cycles needed for this algorithm using the TMS320C25 is 7N+26, which is slightly less than for the LMS algorithm's 7N+28. Computing u*e(n) takes 5 instruction cycles. The sign-error LMS algorithm determines the sign of the u by checking the sign of e(n), which takes only 3 instruction cycles. The total number of instruction cycles needed for the sign-error LMS algorithm using the TMS320C30 is 3N+16, which is slightly higher than for the LMS algorithm. This occurs because the TMS320C30 takes only one instruction cycle to compute u*e(n) and two instruction cycles to determine the sign of the u.

Secondly, the sign-data LMS algorithm is

$$\underline{\mathbf{w}}(\mathbf{n}+1) = \underline{\mathbf{w}}(\mathbf{n}) + \mathbf{u} \ \mathbf{e}(\mathbf{n}) \ \mathrm{sign}[\underline{\mathbf{x}} \ (\mathbf{n})] \tag{26}$$

This equation can be implemented as

$$w_i(n+1) = w_i(n) + ue(n)$$
, if $x(n-i) > 0$
= $w_i(n) - ue(n)$, if $x(n-i) < 0$

for i=0,1,...,N-1. Since the sign determination is required inside the adaptation loop to determine the sign of x(n-i), slower throughput is expected. The total number of instruction cycles needed is 11N+26 for the TMS320C25 and 5N+16 for the TMS320C30.

Finally, the sign-sign LMS algorithm is

$$\underline{\mathbf{w}}(\mathbf{n}+1) = \underline{\mathbf{w}}(\mathbf{n}) + \mathbf{u} \operatorname{sign}[\mathbf{e}(\mathbf{n})] \operatorname{sign}[\underline{\mathbf{x}}(\mathbf{n})]$$
 (27)

which requires no multiplications at all and is used in the CCITT standard for ADPCM transmission. As we can see from the above equations, the number of multiplications is reduced. This simplified LMS algorithm looks promising and is designed for VLSI or discrete IC implementation to save multiplications.

The sign-sign LMS algorithm can be implemented as

```
for (i=0; i<N; i++) {
    if (e[n] >= 0.) {
        if (xn[i] >= 0.)
            wn[i] += u;
        else
            wn[i] -= u; }
        else {
        if (xn[i] >= 0.)
            wn[i] -= u;
```

else
$$wn[i] += u;$$
}

When this algorithm is implemented on TMS320C25 and TMS320C30 with pipeline architecture and a parallel multiplier, the performance of sign-sign LMS algorithm is poor compared to standard LMS algorithm due to the determination of sign of data, which can break the instruction pipeline and can severely reduce the execution speed of the processors.

In order to avoid double branches inside the loop, the XOR instruction is utilized to check the sign bit of e(n) and x(n-i). The sign-sign LMS algorithm can be implemented as

```
w_i(n+1) = w_i(n) + u, if sign[e(n)] = sign[x(n-i)]
= w_i(n) - u, otherwise
```

The following TMS320C25 instruction sequence implements this algorithm without branching (assuming that the current address register used is AR3):

```
AR1.N-1
                                ; Set up counter
      LRLK
               AR2,COEFFD
                                ; Point to w<sub>i</sub>(n)
      LRLK
      LRLK
               AR3,LASTAP+1; Point to x(n-i)
                                ; Load x(n-i)
              *-.0.AR2
ADAP LAC
                                : XOR with e(n)
      XOR
               ERR
                                ; Save sign bit, sign = 0 if same signs
      SACL
               ERRF
                                : Sign = 1 if different signs
                                ; Sign extension to ACCH,
      LAC
               ERRF
                                ACCH = 0 If ERRF > 0
                                : ACCH = 0FFFFh if ERRF < 0
                                ; Take one's complement of m
      XORK
               MU,15
                                ; If sign = 1
                                ; Weight update
       ADD
               *,15
                                ; Save new weight
               *+,1,AR1
       SACH
       BANZ
               ADAP,*-,AR3
```

The one's complement of u is used instead of -u, because they are only slightly different and the step size does not require the exact number. The weight update with this technique requires 10N instruction cycles and FIR filtering requires N instruction cycles so that the total number of instruction cycles needed is 11N+21. The complete TMS320C25 assembly program is given in Appendix F1.

To determine whether a positive or negative u should be used without branching is trickier in the TMS320C30. Fortunately, the extended precision registers of TMS320C30 interpret the 32 most-significant bits of the 40-bit data as the floating-point number and the 32 least-significant bits of the 40-bit data as an integer. When a floating-point number

changes its sign, its exponent remains the same. Therefore, the sign of step size u can be determined by using XOR logic on its mantissa. The following code shows how the sign-sign LMS algorithm is implemented on the TMS320C30.

```
ASH
               -31,R7
                                ; R7 = Sign[e(n)]
        XOR3
               R0,R7,R5
                                ; R5 = Sign[e(n)] * u
               *AR0++(1)\%,R6; R6 = x(n)
        LDF
                                ; R6 = Sign[x(n-i)]
        ASH
               -31,R6
               R5,R6,R4
                                ; R4 = Sign[x(n-i)]*Sign[e(n)]*u
        XOR3
        ADDF3 *AR1,R4,R3
                                R3 = w_i(n) + R4
       LDI
               order-3,RC
                                ; Initialize repeat counter
        RPTB
                                ; Do i = 0, N-3
               SSLMS
       LDF
               *AR0++(1)\%,R6; Get next data
    | STF
               R3,*AR1++(1)\%; Update w_i(n+1)
                                ; Get the sign of data
        ASH
               -31,R6
        XOR3
               R5,R6,R4
                                ; Decide the sign of u
                                R3 = w_i(n) + R4
       ADDF3 *AR1,R4,R3
SSLMS
       LDF
               *AR0,R6
                                ; Get last data
    | STF
               R3,*AR1++(1)\%; Update w_{N-2}(n+1)
       ASH
               -31,R6
                                ; Get the sign of data
               R5,R6,R4
                                ; Decide the sign of u
        XOR3
        ADDF3 *AR1,R4,R3
                                ; Compute w_{N-1}(n+1)
                                ; Store last w(n+1)
        STF
               R3,*AR1++(1)\%
```

Here, R0, R4, and R5 contain the value of u before updating. AR0 and AR1 point to x array and w array, respectively. R7 contains the value of error signal e(n). The complete program is given in Appendix F2. The total number of instruction cycles is 5N+16, which is much higher than LMS algorithm.

The sign-sign LMS algorithm is developed to reduce the multiplication requirement of the LMS algorithm. Since DSPs provide the hardware multiplier as a standard feature, this modification does not provide any advantage when implementing this algorithm on the DSPs. On the contrary, it causes some disadvantages since decision instructions will destroy the instruction pipeline. If you use the XOR logic operation in order to avoid using the decision instructions, the complexity of the program will be increased and the total number of instruction cycles will be greater than the regular LMS algorithm.

Leaky LMS Algorithm

When adaptive filters are implemented on signal processors with fixed word lengths, roundoff noise is fed back to adaptive weights and accumulates in time without bound. This leads to an overflow that is unacceptable for real-time applications. One solution is

based upon adding a small forcing function, which tends to bias each filter weight toward zero. The leaky LMS algorithm has the form

$$\underline{\mathbf{w}}(\mathbf{n}+1) = \mathbf{r} \ \underline{\mathbf{w}}(\mathbf{n}) + \mathbf{u} \ \mathbf{e}(\mathbf{n}) \ \underline{\mathbf{x}}(\mathbf{n})$$
 (28a)

where r is slightly less than 1.

Since r can be expressed as 1-c and c <<1, the TMS320C25 can take advantage of the built-in shifters to implement this algorithm. Therefore, Equation (28a) can be changed to

$$\underline{\mathbf{w}}(\mathbf{n}+1) = \underline{\mathbf{w}}(\mathbf{n}) - \mathbf{c} \ \underline{\mathbf{w}}(\mathbf{n}) + \mathbf{u} \ \mathbf{e}(\mathbf{n}) \ \underline{\mathbf{x}}(\mathbf{n})$$
 (28b)

In order to achieve the highest throughput by using ZALR and MPYA, cw(n) can be implemented by shifting $w_i(n)$ right by m bits where 2^{-m} is close to c. Since the length of the accumulator is 32 bits and the high word (bits 16 to 31) is used for updating w(n), shifting right m bits of $w_i(n)$ can be implemented by loading $w_i(n)$ and shifting left 16-m bits. The sequence of TMS320C25 instructions to implement Equation (28b) is shown as

```
LRLK
               AR1,N-1
                               ; Set up counter
       LRLK
               AR2, COEFFD; Point to w_i(n)
       LRLK
               AR3,LASTAP+1
                               ; Point to x(n - i)
       LT
                                ; T = ERRF = u*e(n)
               ERRF
       MPY
               *-,AR2
ADAPT ZALR
               *,AR3
               *-.AR2
       MPYA
       SUB
               *,LEAKY
                                : LEAKY = 16 - m
       SACH
               *+.0.AR1
       BANZ
               ADAPT,*-,AR2
```

For each iteration, 7N instruction cycles are needed to perform the adaptation process (6N for the LMS algorithm). The total number of instruction cycles needed is 8N+28 (see Appendix G1 for the complete program). The leaky factor r has the same effect as adding a white noise to the input. This technique not only can solve adaptive weights overflow problem, but also can be beneficial in an insufficient spectral excitation and stalling situation [5].

The method used above is especially for the TMS320C25, which has a free shift feature. Since TMS320C30 is a floating-point processor, r can simply multiply to filter coefficient. However, in order to reduce the instruction cycles, this multiplication can combine with another instruction to be a parallel instruction inside the loop. The following code shows how to rearrange the instructions from the LMS algorithm to include this multiplication without an extra instruction cycle.

```
MPYF
                  @u__r,R7
                                      : R7 = e(n)*u/r
                   *AR0++(1)\%,R7,R1;R1=e(n)*u*x(n)/r
         MPYF3
                   *AR0++(1)\%,R7,R1;R1=e(n)*u*x(n-1)/r
         MPYF3
       | ADDF3
                   *AR1.R1.R2
                                      R2 = w_0(n) + e(n)*u*x(n)/r
         LDI
                  order-4,RC
                                      : Initialize repeat counter
                                      ; do i = 0, N-4
         RPTB
                  LLMS
         MPYF3
                                      R0 = r*w_i(n) + e(n)*u*x(n-i)
                   *AR2.R2.R0
       | ADDF3
                   *+AR1(1),R1,R2
                                      R2 = w_{i+1}(n) + e(n)*u*x(nz-i-1)/r
         MPYF3
                   *AR0++(1)\%,R7,R1;R1=e(n)*u*x(n-i-2)/r
LLMS
       STF
                  R0.*AR1++(1)\%
                                      : Store w_i(n+1)
         MPYF3
                  *AR2.R2.R0
                                      R0 = r^*w_{N-3}(n) + e(n)^*u^*x(n-N+3)
       | ADDF3
                  *+AR1(1),R1,R2
                                      R2 = w_{N-2}(n) + e(n)*u*x(n-N+2)/r
                                      R1 = e(n)*u*x(n-N+1)/r
         MPYF3
                  *AR0.R7.R1
       STF
                  R0.*AR1++(1)\%
                                      ; Store w_{N-3}(n+1)
         MPYF3
                  *AR2,R2,R0
                                      R0 = r^*w_i(n) + e(n)^*u^*x(n-N+2)
       | ADDF3
                  *+AR1(1).R1.R2
                                      R2 = w_{N-1}(n) +
                                            e(n)*u*x(n-N+1)/r
                  *AR2.R2.R0
                                      R0 = r*w_i(n) + e(n)*u*x(n-N+1)
         MPYF3
       | | STF
                  R0.*AR1++(1)\%
                                      ; Store w_{N-2}(n+1)
         STF
                  R0.*AR1++(1)\%
                                      ; Update last w
```

Auxiliary registers AR0 and AR1 point to x and w arrays. AR2 points to the memory location that contains value r. R7 contains the value of error signal e(n). R1 and R2 are updated before the loop because the parallel instructions inside the loop use the previous values in R1 and R2. Note that R1 is updated twice before the loop because the updating of R2 requires the previous value of R1. In order to update x array pointer to the new beginning of the data buffer for next iteration, two of the loop instruction sets have been taken out of loop and modified by eliminating the incrementation of AR0. The TMS320C30 assembly program of an adaptive transversal filter with the leakage LMS algorithm is listed in Appendix G2 as an example. The total number of instruction cycles for this algorithm is 3N+15, which is the same as the LMS algorithm. This example shows the power and flexibility of the TMS320C30.

Implementation Considerations

The adaptive filter structures and algorithms discussed previously were derived on the basis of infinite precision arithmetic. When implementing these structures and algorithms on a fixed integer machine, there is a limitation on the accuracy of these filters due to the fact that the DSP operates with a finite number of bits. Thus, designers must pay attention to the effects of finite word length. In general, these effects are input quantization, roundoff in the arithmetic operation, dynamic range constraints, and quantization of filter coefficients. These effects can either cause deviations from the original design criteria or create an effective noise at the filter output. These problems have been investigated extensively, and techniques to solve these problems have been developed [28, 29].

The effects of finite precision in adaptive filters is an active research area, and some significant results have been reported [30 through 32]. There are three categories of finite word length effects in adaptive filters:

- Dynamic Range Constraint (scaling to avoid overflow). Since this is not applicable for a floating-point processor, the TMS320C30 is not mentioned in this portion.
- Finite Precision Errors (errors introduced by roundoff in the arithmetic).
- Design Issues (design of the optimum step size u that minimizes system noise).

Dynamic Range Constraint

As shown in Figure 1, the most widely used LMS transversal filter is specified by the difference equations

$$y(n) = \sum_{i=0}^{N-1} w_i(n) x(n-i)$$
 (29)

and

$$w_i(n+1) = w_i(n) + u \cdot e(n) \cdot x(n-i), \text{ for } i = 0, 1, ..., N-1$$
 (30)

where x(n-i) is the input sequence and $w_i(n)$ are the filter coefficients.

If the input sequence and filter coefficients are properly normalized so that their values lie between -1 and 1 using Q15 format, no error is introduced into the addition. However, the sum of two numbers may become larger than one. This is known as overflow. The TMS320C25 provides four features that can be applied to handle overflow management [13]:

- A. Branch on overflow conditions.
- B. Overflow mode (saturation arithmetic).
- C. Product register right shift.
- D. Accumulator right shift.

One technique to inhibit the probability of overflow is scaling, i.e., constraining each node within an adaptive filter to maintain a magnitude less than unity. In Equation (29), the condition for |y(n)| < 1 is

$$x_{\text{max}} < 1 / \sum_{i=0}^{N-1} |w_i(n)|$$
 (31)

where x_{max} denotes the maximum of the absolute value of the input. The right shifter of the TMS320C25, which operates with no cycle overhead, can be applied to implement scaling to prevent overflow of multiply-accumulate operations in Equation (29). By setting the PM bits of status register ST1 to 11 using the SPM or LST1 instructions, the P register output is right-shifted 6 places. This allows up to 128 accumulations without the possibility of an overflow. SFR instruction can also be used to right shift one bit of the accumulator when it is near overflow.

Another effective technique to prevent overflow in the computation of Equation (29) is using saturation arithmetic. As illustrated in Figure 12, if the result of an addition overflows, the output is clamped at the maximum value. If saturation arithmetic is used, it is common practice [28] to permit the amplitude of x(n-i) to be larger than the upper bound given in Equation (31). Saturation of the filter represents a distortion, and the choice of scaling on the input depends on how often such distortion is permissible. The saturation arithmetic on the TMS320C25 is controlled by the OVM bit of status register ST0 and can be changed by the SOVM (set overflow mode), ROVM (reset overflow mode), or LST (load status register).

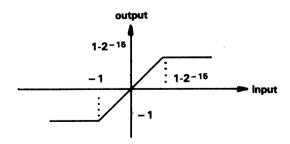


Figure 12. Saturation Arithmetic

Filter coefficients are updated using Equation (30). As illustrated in Figure 13, a new technique presented in reference 31 uses the scaling factor a to prevent filter's coefficients overflow during the weight updating operation. Suppose you use a=2-m. A right shift by m bits implements multiplication by a, while a left shift by m bits implements the scaling factor 1/a. Usually, the required value of a is not expected to be very small and depends on the application. Since a scales the desired signal, it does not affect the rate of convergence.

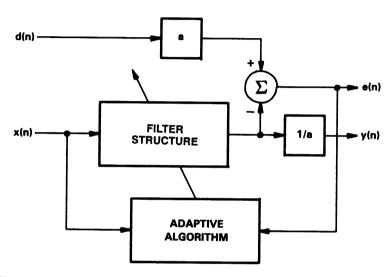


Figure 13. Fixed-Point Arithmetic Model of the Adaptive Filter

Finite Precision Errors

The TMS320C25 is a 16/32-bit fixed point processor. Each data sample is represented by a fractional number that uses 15 magnitude bits and one sign bit. The quantization interval

$$\delta = 2^{-b}, \tag{32}$$

(b = 15), is called the width of quantization since the numbers are quantized in steps of δ .

The products of the multiplications of data by coefficients within the filter must be rounded or truncated to store in memory or a CPU register. As shown in Figure 14, the roundoff error can be modeled as the white noise injected into the filter by each rounding operation. This white noise has a uniform distribution over a quantization interval and for rounding

$$\delta_{e^2} = (1/12) \, \delta^2 \tag{33b}$$

where δ_e^2 is the variance of the white noise.

In general, roundoff noise occurs after each multiplication. However, the TMS320C25 has a full precision accumulator, i.e., a 16×16 -bit multiplier with a 32-bit accumulator, so there is no roundoff when you implement a set of summations and multiplications as in Equation (29). Rounding is performed when the result is stored back to memory location y(n), so that only one noise source is presented in a given summation node.

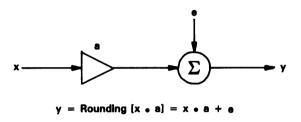


Figure 14. Fixed-Point Roundoff Noise Model

For floating-point arithmetic, the variance of the roundoff noise [31] is slightly different from Equation (33b),

$$\sigma_e^2 = 0.18 \,\delta^2 \tag{33c}$$

Since TMS320C30 has a 40/32-bit floating-point multiplier and ALU, the result from arithmetic operation has the mantissa of [31] bits plus one sign bit. Therefore, the δ in Equation (33c) is equal to 2^{-31} . Another roundoff noise is introduced when you restore the result back to memory. This noise has the power of 2^{-23} because the mantissa of TMS320C30 floating-point data is 23 bits plus one sign bit. Therefore, unless the filter order is high, the roundoff noise from arithmetic operation is relatively small.

The steady-state output error of the LMS algorithm due to the finite precision arithmetic of a digital processor was analyzed in reference [31]. It was found that the power of arithmetic errors is inversely proportional to the adaptation step size u. The significance of this result in the adaptive filter design is discussed next. Furthermore, roundoff noise is found to accumulate in time without bound, leading to an eventual overflow [32]. The leaky LMS algorithm presented in the previous section can be used to prevent the algorithm overflow.

Design Issues

The performance of digital adaptive algorithms differs from infinite precision adaptive algorithms. The finite precision LMS algorithm is given as

$$\underline{\mathbf{w}}(\mathbf{n}+1) = \underline{\mathbf{w}}(\mathbf{n}) + \mathbf{Q}[\mathbf{u}^*\mathbf{e}(\mathbf{n})^*\underline{\mathbf{x}}(\mathbf{n})] \tag{34}$$

where Q [.] denotes the operation of fixed point quantization. Whenever any correction term $u^*e(n)^*x(n-i)$ in the update of the weight vector in Equation (34) is too small, the quantized value of that term is zero, and the corresponding weight $w_i(n)$ remains unchanged. The condition for the ith component of the vector w(n) not to be updated when the algorithm is implemented with the TMS320C25 is

$$| u e(n) x(n-i) | < \delta/2$$
 (35a)

where $\delta = 2^{-15}$. The condition for TMS320C30 is

$$| u e(n) x(n-i) | < 2^{exp} * \delta/2$$
 (35b)

where exp is the exponent of $w_i(n)$ and $\delta = 2^{-23}$.

Since the adaptive algorithms are designed to minimize the mean squared value of the error signal, e(n) decreases with time. If u is small enough, most of the time the weights are not updated. This early termination of the adaptation may not allow the weight values to converge to the optimum set, resulting in a mean square error larger than its minimum value. The conditions for the adaptation to converge completely [30] is $u > u_{min}$ where

$$u^{2}_{\min} = \frac{\delta^{2}}{4\sigma_{\chi}^{2}\epsilon_{\min}}$$
 (36a)

for the TMS320C25 and the TMS320C30

$$u^{2}_{\min} = \frac{\delta^{2} * 2 \exp}{4\sigma_{X}^{2} \epsilon_{\min}}$$
 (36b)

where σ_x^2 is the power of input signal x(n) and ϵ_{min} is the minimum mean squared error at steady state.

In the Leaky LMS Algorithm section, it was mentioned that the excess MSE given in Equation (14) is minimized by using small u. However, this may result in a large quantization error since the most significant term in the total output quantization error is [31]

$$\frac{\cdot N\sigma_e^2}{2 \cdot n^2}$$
 (37)

The optimum step size u_0 reflects a compromise between these conflicting goals. The value of u_0 is shown to be too small to allow the adaptive algorithm to converge completely and also to give a slow convergence. In practice, $u > u_0$ is used for faster convergence. Hence, the excess MSE becomes larger, and the roundoff noise can typically be neglected when compared with the excess mean square error.

Finally, recall Equations (11) and (12). The step size u has an upper limit to guarantee the stability and convergence. Therefore, the adaptive algorithm requires

$$0 < u < \frac{1}{N\sigma_{\rm x}^2} \tag{38}$$

On the other hand, the step size u also has a lower limit. The optimum u_0 , which minimizes the sum of the excess MSE and roundoff noise, is smaller than u_{min} , i.e., too small to allow the adaptive weight to converge. For an algorithm implemented on the TMS320C25, the word-length of 16 bits is fixed, and the minimum step-size that can be used is given in Equation (36). The most important design issue is to find the best u to satisfy

$$u_{\min} < u < \frac{1}{N\sigma_{x}^{2}} \tag{39}$$

Therefore, in order to make the condition in Equation (39) valid, the initial values of filter coefficients are better close to zero for the floating-point processor if the situation in unknown.

Software Development

The TMS320C25 and TMS320C30 combine the high performance and the special features needed in adaptive signal processing applications. The processors are supported by a full set of software and hardware development tools. The software development tools include an assembler, a linker, a simulator, and a C compiler. The most universal software development tool available is a macro assembler. However, the assembly language programming for DSP can be tedious and costly. For adaptive filter applications, an assembly language programmer must have knowledge of adaptive signal processing. The challenge lies in compressing a great deal of complex code into the fairly small space and most efficient code dictated by the real-time applications typical of adaptive signal processing.

Recently, C compilers for the processors were developed to make DSP programming easier, quicker, and less costly compared with the work associated with programming in assembly language. Due to the general characteristics of a compiler, the code it generates is not the most efficient. Since the program efficiency consideration is important for adaptive filter implementation, the code generated from the C compiler has to be modified before implementing. Thus, two alternative ways, besides writing an assembly program, to implement adaptive signal processing on DSP are presented. First is the automatic adaptive filter code generator [12], which can be found on Texas Instruments TMS320 Bulletin Board Service (BBS), and second are the adaptive filter function libraries that support assembly and C programming languages.

In this report, two adaptive filter libraries have been developed: one can be called from an assembly main program; the other can be called from the C main program. Note that, for the TMS320C25 only, certain data memory locations have been reserved for storing the necessary filter coefficients, previous delayed signal, etc. In other words, these data memories are used as global variables.

Assembly Function Libraries

The basic concept of creating an assembly subroutine for an adaptive filter is to modify in module the assembly programs discussed above. Then, the user can implement the adaptive filter by writing his own assembly main program that calls the subroutine.

TMS320C25 Assembly Subroutine

The TMS320C25 has an eight-level deep hardware stack. The CALL and CALA subroutine calls store the current contents of the program counter (PC) on the top of the stack. The RET (return from subroutine) instruction pops the top of the stack back to the PC. For computational convenience, the processor needs to be set as follows before calling the assembly callable subroutine.

- 1. PM status bits equal to 01.
- 2. SXM status bit set to 1.
- 3. The current DP (data memory page pointer) is 0.

The following example is the TMS320C25 assembly main routine, which performs an adaptive line enhancement by calling the LMS algorithm subroutine. The filter order is 64, delay is equal to one, and the convergence factor u is 0.01.

DEFINE AND REFER SYMBOLS

.global ORDER,U,ONE,D,Y,ERR,XN,WN,LMS

DEFINE SAMPLING RATE, ORDER, AND MU

```
ORDER:
                 20
          .equ
                               ; mu = 0.01 in Q15 format
                 327
MU:
          .equ
                 0
PAGEO:
          .equ
     DEFINE ADDRESSES OF BUFFER AND COEFFICIENTS
X0:
                 "buffer", ORDER - 1
          .usect
                 "buffer",1
XN:
          .usect
                 "coeffs", ORDER
WN:
          .usect
     RESERVE ADDRESSES FOR PARAMETERS
*
ONE:
                 "parameters",1
          .usect
                 "parameters",1
U:
          .usect
ERR:
                 "parameters",1
          .usect
                 "parameters",1
Y:
          .usect
                 "parameters",1
D:
          .usect
                 "parameters",1
ERRF:
          .usect
     INITIALIZATION
START
          LDPK
                 PAGE0
                                : Set DP = 0
                                ; Set PM equal to 1
          SPM
                 1
          SSXM
                                ; Set sign extension mode
                                ; AR7 point to >300
          LRLK
                 AR7,X0
                                ; Initialize ONE = 1
          LACK
          SACL
                ONE
          LALK
                 MU
                                ; Initialize U = MU = 0.01
          SACL
     PERFORM THE PREDICTOR
*************************
INPUT:
          IN
                 D,PA2
                                ; Get the input
          CALL
                LMS
                                ; Call subroutine
OUTPUT:
          OUT
                 Y,PA2
                                ; Output the signal
                                ; Insert the newest sample
          LAC
                 D
          LARP
                 AR7
          SACL
                 INPUT
          В
          .end
```

The symbols, such as ORDER, U, ONE, D, LMS, Y, and ERR, are defined and referred to for the purpose of modular programming. The uninitialized sections specified by the directive .usect can be placed in any location of memory according to the linker command file. Note that MACD instruction requires the sources of the operands on program memory and data memory separately, and CNFP instruction configures RAM block 0 as program memory. Therefore, the coeffs section has to be in data RAM block 0, and the buffer has to be in RAM block 1. Appendix H1 contains the adaptive transversal filter with LMS algorithm subroutine using the TMS320C25, and Appendix H2 contains an example of a linker command file.

TMS320C30 Assembly Subroutine

Instead of a hardware stack, TMS320C30 uses a software stack, which is more flexible and convenient for a high-level language compiler. The stack memory location is pointed to by the stack pointer SP. In order to maintain the proper program sequence, the programmer must make certain that no data is lost and that the stack pointer always points to proper location. The PUSH, PUSHF, POP, POPF, CALL, CALLcond, RETIcond, and RETScond instructions will change the value of the stack pointer; in addition, writing data into it and using the interrupt will also change that value. It is the programmer's responsibility to initialize the stack pointer in the beginning of the program. The same adaptive line enhancer example above using TMS320C30 is listed below. The adapfltr int program that initializes the stack pointer and the data RAM is given in Appendix H3.

: R0 = 0.0

```
DEFINE GLOBAL VARIABLES AND CONSTANTS
                  "adapfltr.int"
           .copy
           .global LMS30,order,u,d,y,e
N
                  20
           .set
           .set
                  0.01
mu
      INITIALIZE POINTERS AND ARRAYS
           .text
begin
                   $
           .set
                  N,BK
                                     ; Set up circular buffer
           LDI
                                     ; Set data page
           LDP
                   @xn__addr
                  @xn_addr,AR0
                                     ; Set pointer for x[]
           LDI
                  @wn_addr,AR1
                                     ; Set pointer for w[]
           LDI
```

R0,*AR0++(1)% ; x[] = 0.

0.0, R0

N-1

LDF RPTS

STF

```
R0,*AR1++(1)\% ; w[] = 0.
         STF
                                     ; Set pointer for input ports
           LDI
                  @in_addr,AR6
                                     ; Set pointer for output ports
                  @out_addr,AR7
           LDI
*
*
      PERFORM ADAPTIVE LINE ENHANCER
*
input:
           LDF
                   *AR6,R7
                                     ; Input d(n)
         LDF
                                     ; Input x(n)
                   *+AR6(1),R6
           STF
                                     ; Insert d(n)
                   R7,@d
                  R6,*AR0
                                     ; Insert x(n) to buffer
           STF
      CALL ASSEMBLY SUBROUTINE
           CALL LMS30
      OUTPUT y(n) AND e(n) SIGNALS
           LDF
                   @y,R6
                                     ; Get y(n)
                  input
                                     ; Delay branch
           BD
                                     ; Get e(n)
           LDF
                   @e,R7
                                     ; Send out y(n)
           STF
                  R6.*AR7
                  R7,*+AR7(1)
                                     ; Send out e(n)
           STF
      DEFINE CONSTANTS
                   "buffer", N
           .usect
n
           .usect
                   "coeffs", N
wn
in__addr
                   "vars",1
           .usect
                   "vars",1
out__addr
           .usect
xn_addr
                   "vars",1
           .usect
                   "vars",1
wn_addr
           .usect
                   "vars",1
           .usect
u
                   "vars",1
order
           .usect
                   "vars",1
d
           .usect
                   "vars",1
y
           .usect
                   "vars",1
           .usect
                   ".cinit"
cinit
           .sect
                   6,in_addr
           .word
                   0804000h
           .word
                   0804002h
           .word
           .word
                   xn
           .word
                   wn
```

.float mu .word N−2 .end

In the above example, data memory order is initialized to N-2 for computation convenience. The linker command files and the subroutine that implements the LMS transversal filter can be found in Appendixes H4 and H5.

C Function Libraries

The TMS320C25 and TMS320C30 C language compilers provide high-level language support for these processors. The compilers allow application developers without an extensive knowledge of the device's architecture and instruction set to generate assembly code for the device. Also, since C programs are not device-specific, it is a relatively straightforward task to port existing C programs from other systems.

To allow fast development of efficient programs for adaptive signal processing applications, C function libraries have been developed. These libraries include functions for adaptive transversal, symmetric transversal, and lattice structures.

TMS320C25 C-Callable Subroutines

In a C program, the memory assignments are chosen by the compiler. There are two ways to use the most efficient instruction MACD:

- A. Use inline assembly code to assign memory locations for filter coefficients and buffers.
- B. Reserve the desired memory locations for them and do the assignment in the linker command file.

The latter method is used in this report.

For a C main program, the parameters passed to and returned from the subroutines are all within the parentheses following the subroutine name, as shown below:

lms(n,mu,d,x,&y,&e) n - Filter order

mu - Convergence factor

d - Desired signalx - Input signal

y - Address of output signal

e - Address of error signal

Since the TMS320C25 C compiler pushes the parameters from right to left into software stack pointed by AR1, the subroutine gets the parameters in reverse order, as shown below:

MAR *- ; Set pointer for getting parameters

LAC *- ; ACC = N

```
SUBK
        1
                      : ORDER = N - 1
SACL
        ORDER
                     ; Getting and storing the mu
LAC
        U
SACL
        *_
                     ; Getting and storing the D
LAC
SACL
        D
                     ; Insert the newest sample
        *-.0,A-R3
LAC
LRLK
        AR3, FRSTAP
SACL
```

The assembly subroutine returns the parameters y and e as follows:

```
AR1
LARP
        AR2,*-,AR2; Get the address of y in main
LAR
LAC
        Y
        *,0,AR1
                      ; Store y
SACL
                      : Get the address of e in main
        AR2,*,AR2
LAR
        ERR
LAC
SACL
        *,0,AR1
                      ; Store e
```

Therefore, the parameters should be entered in the order given above. If there are other parameters, they should be inserted right after the convergence factor mu. The leaky LMS algorithm subroutine is given as an example.

```
llms(n,mu,r,d,x,&y,&e)
```

the r is defined in Equation (28a). Note that the values of the AR registers, which will be used in subroutine, and the status registers must be saved at the beginning of the subroutine and restored right before returning to calling routine. An example of a C-callable program is given in Appendix I1. Memory locations 0200h to 0200h + N - 1 and 0300h to 0300h + N - 1 are reserved for filter coefficients and buffers, respectively. N denotes the filter order.

TMS320C30 C Subroutine

As previously mentioned, the TMS320C30 architecture has features designed for a high-level language compiler. Note that the callable word is dropped in this section title because the TMS320C30 is so flexible that the restrictions for the TMS320C25 no longer exist. Since the memory locations of filter buffers and coefficients are determined by the parameters that pass from the calling routine, the same subroutine can be used in different places. However, the only restriction is that the memory locations of filter buffers must align to the circular addressing boundary [14]. The features of TMS320C30 architecture that make a major contribution toward these improvements are dual data address buses, software stack, and flexible addressing mode. The parameters passed to subroutine are pushed into the stack. Therefore, after returning from the subroutine, the stack pointer, SP, must be updated to point to the location where SP pointed before pushing the parameters

into the stack. However, this will be done by the C compiler. The usage example of the C function subroutine is given as follows:

```
tlms(n,u,d,&w,&x,&y,&e) where

n - Filter order

u - Step size

d - Desired signal

&w - Filter coefficients

&x - Input signal buffers

&y - Addr of output signal

&e - Addr of error signal
```

The example below shows how the C subroutine receives and manipulates the parameters passed from the caller program and how the result is returned to the caller routine.

```
SET FRAME POINTER FP
FP
     .set
             AR3
     PUSH
             FP
     LDI
             SP,FP
*
     GET FILTER PARAMETERS
*
     LDI
             *-FP(2),R4; Get filter order
     LDI
             *-FP(6),AR0 ; Get pointer for x[]
             *--FP(5),AR1; Get pointer for w[]
     LDI
     COMPUTE ERROR SIGNAL e(n) AND STORE y(n) AND e(n)
     LDI
             *-FP(2),AR2
                          ; Get y(n) address
    SUBF3
            R2,*+FP(1),R7; e(n) = d(n) - y(n)
   STF
            R2,*AR2; Send out y(n)
            *-FP(3),AR2
    LDI
                          ; Get e(n) address
     STF
            R7.*AR2
                          ; Send out e(n)
    MPYF
            *+FP(2).R7
                          R7 = e(n) * u
    POP
            FP
```

Note that AR3 is used as the frame pointer in TMS320C30 C compiler. Appendix I2 contains the complete LMS transversal filter example subroutine program.

Development Process and Environment

Following a four stage procedure [33] to minimize the amount of finite word length effect analysis and real-time debugging, adaptive structures and algorithms are implemented

on the TMS320C25. Figure 15 illustrates the flowchart of this procedure. Since the implementation on TMS320C30 is done only by the simulator, the last stage, real-time testing, is not implemented.

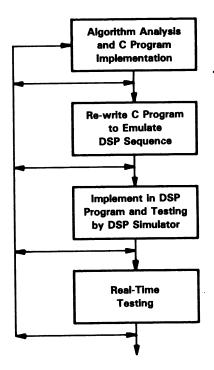


Figure 15. Adaptive Filter Implementation Procedure

In the first stage, algorithm design and study is performed on a personal computer. Once the algorithm is understood, the filter is implemented using a high-level C program with double precision coefficients and arithmetic. This filter is considered an ideal filter.

In the second stage, the C program is rewritten in a way that emulates the same sequence of operations with the same parameters and state variables that will be implemented in the processors. This program then serves as a detailed outline for the DSP assembly language program or can be compiled using TMS320C25 or TMS320C30 C compiler. The effects of numerical errors can be measured directly by means of the technique shown in Figure 16, where H(z) is the ideal filter implemented in the first stage and H'(z) is a real filter. Optimization is performed to minimize the quantization error and produce stable implementation.

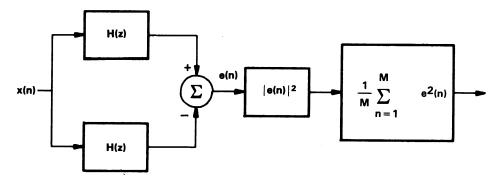


Figure 16. A Commutational Technique for Evaluating Quantization Effects

In the third stage, the TMS320C25 and TMS320C30 assembly programs are developed; then they are tested using the simulators with test data from a disk file. Note that the simulation of TMS320C25 can also be implemented on the SWDS with the data logging option. This test data is a short version of the data used in stage 2 that can be internally generated from a program or data digitized from a real application environment. Output from the simulation is compared against the equivalent output of the C program in the second stage. Since the simulation requires data files to be in Q15 format, certain precision is lost during data conversion. When a one-to-one agreement within tolerable range is obtained between these two outputs, the processor software is assured to be essentially correct.

The final stage is applied only to the TMS320C25. First, you download this assembled program into the target TMS320C25 system (SWDS) to initiate real-time operation. Thus, the real-time debugging process is constrained primarily to debugging the I/O timing structure of the algorithm and testing the long-term stability of the algorithm. Figure 17 shows an experimental setup for verification, in which the adaptive filter is configured for a one-step adaptive predictor illustrated in Figure 18. The data used for real-time testing is a sinusoid generated by a Tektronix FG504 Function Generator embedded in white noise generated by an HP Precision Noise Generator. The DSP gets a quantized signal from the Analog Interface Board (AIB), performs adaptive prediction routines, and outputs an enhanced sinusoid to the analog interface board. The corrupted input and predicted (enhanced) output waveforms are compared on the oscilloscope or on the HP 4361 Dynamic Signal Analyzer. The corresponding spectra of input and output can be compared on the signal analyzer. The signal-to-noise ratio (SNR) improvement can be measured from the analyzer, which is connected to an HP plotter.

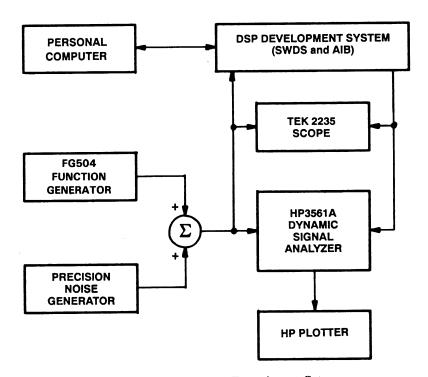


Figure 17. Real-Time Experiment Setup

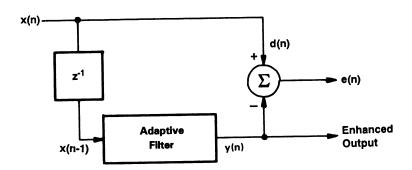


Figure 18. Block Diagram of a One-Step Adaptive Predictor

To illustrate the operation in a nonstationary environment, the adaptive predictor is implemented using a TMS320C25, and the following experiment is performed. The input signal is swept from 1287 Hz to 4025 Hz, then jumps back to 1287 Hz. The time for each sweep is one second. The input spectra at every second are shown in Figure 19a; the corresponding output spectra are shown in Figure 19b. From the observations on the

oscilloscope and signal analyzer, the significant SNR improvement, convergence speed, ability to track nonstationary signals, and long-term stability of the adaptive predictor are observed.

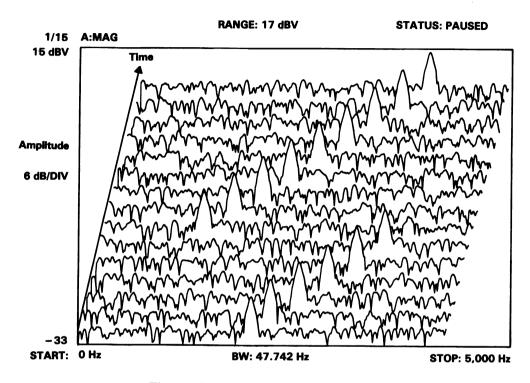


Figure 19(a). Spectrum of Input Signal

RANGE: 13 dBV STATUS: PAUSED

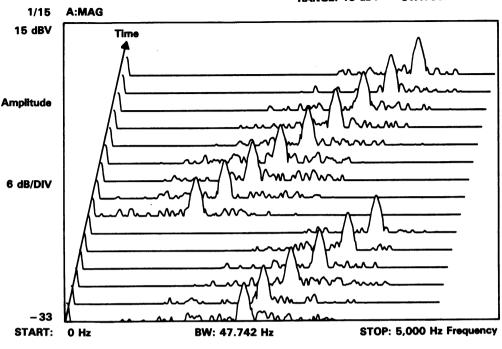


Figure 19(b). Spectrum of Enhanced Output Signal Summary

Three adaptive structures and six update algorithms are implemented with the TMS320C25 and TMS320C30. Applications of adaptive filters and implementation considerations have been discussed. Two subroutine libraries that support both C language and assembly language for two processors were developed. These routines can be readily incorporated into TMS320C25 or TMS320C30 users' application programs.

The advancements in the TMS320C25 and TMS320C30 devices have made the implementation of sophisticated adaptive algorithms oriented toward performing real-time processing tasks feasible. Many adaptive signal processing algorithms are readily available and capable of solving real-time problems when implemented on the DSP. These programs provide an efficient way to implement the widely used structures and algorithms on the TMS320C25 and TMS320C30, based on assembly-language programming. They are also extremely useful for choosing an algorithm for a given application. The performances of adaptive structures and algorithms that have been implemented using the TMS320C25 and TMS320C30 have been summarized in Tables 1 and 2.

Table 1. The Performance of Adaptive Structures and Algorithms of TMS320C25

		TMS320C25	
	LMS	Instruction Cycles	7N + 28
	LIVIS	Program Memory (Word)	33
	Leaky	Instruction Cycles	8N + 28
	LMS	Program Memory (Word)	34
	Sign-Data	Instruction Cycles	11N + 26
Transversal	LMS	Program Memory (Word)	41
Structure	Sign-Error	Instruction Cycles	7N + 26
	LMS	Program Memory (Word)	30
	Sign-Sign	Instruction Cycles	11N+21
	LMS	Program Memory (Word)	30
	Normalized	Instruction Cycles	7N + 57
	LMS	Program Memory (Word)	47
	LMS	Instruction Cycles	7.5N + 38
	LIVIS	Program Memory (Word)	50
	Leaky	Instruction Cycles	8N + 38
	LMS	Program Memory (Word)	51
Symmetric	Sign-Data	Instruction Cycles	9.5N + 36
Transversal	LMS	Program Memory (Word)	58
Structure	Sign-Error	Instruction Cycles	7.5N + 36
Structure	LMS	Program Memory (Word)	47
	Sign-Sign	Instruction Cycles	9.5N + 31
	LMS	Program Memory (Word)	47
	Normalized	Instruction Cycles	7.5N + 69
	LMS	Program Memory (Word)	66
	LMS	Instruction Cycles	33N + 32
	LIVIS	Program Memory (Word)	63
	Leaky	Instruction Cycles	35N + 32
Lattice	LMS	Program Memory (Word)	65
Structure	Sign-Error	Instruction Cycles	36N + 32
	LMS	Program Memory (Word)	65
	Normalized	Instruction Cycles	90N + 34
	LMS	Program Memory (Word)	92

Note: N represents filter order.

Table 2. The Performance of Adaptive Structures and Algorithms of TMS320C30

		TMS320C30	
	LMS	Instruction Cycles	3N + 15
	LIVIS	Program Memory (Word)	17
	Leaky	Instruction Cycles	3N + 15
\	LMS	Program Memory (Word)	19
	Sign-Data	Instruction Cycles	5N + 16
Transversal	LMS	Program Memory (Word)	24
Structure	Sign-Error	Instruction Cycles	3N + 16
	LMS	Program Memory (Word)	18
	Sign-Sign	Instruction Cycles	5N + 16
	LMS	Program Memory (Word)	24
	Normalized	Instruction Cycles	3N + 47
	LMS	Program Memory (Word)	49
	LMS	Instruction Cycles	2.5N + 15
	LIVIO	Program Memory (Word)	23
	Leaky	Instruction Cycles	2.5N + 19
	LMS	Program Memory (Word)	26
Symmetric	Sign-Data	Instruction Cycles	3.5N + 18
Transversal	LMS	Program Memory (Word)	30
Structure	Sign-Error	Instruction Cycles	2.5N + 18
Structure	LMS	Program Memory (Word)	24
	Sign-Sign	Instruction Cycles	3.5N + 17
	LMS	Program Memory (Word)	30
	Normalized	Instruction Cycles	2.5N + 50
	LMS	Program Memory (Word)	56
	LMS	Instruction Cycles	14N + 9
	LIVIS	Program Memory (Word)	20
	Leaky	Instruction Cycles	16N + 9
Lattice	LMS	Program Memory (Word)	22
Structure	Sign-Error	Instruction Cycles	16N + 9
	LMS	Program Memory (Word)	22
	Normalized	Instruction Cycles	67N+9
	LMS	Program Memory (Word)	73

Note: N represents filter order.

References

- [1] B. Widrow and S. Stearns, Adaptive Signal Processing, Prentice-Hall, 1985.
- [2] R. Lucky, J. Salz, and E. Weldon, *Principles of Data Communications*, McGraw-Hill, 1968.
- [3] S. Haykin, Adaptive Filter Theory, Prentice-Hall, 1986.
- [4] M. Honig and D. Messerschmit, Adaptive Filters: Structures, Algorithms, and Applications, Kluwer Academic, 1984.
- [5] J.R. Treichler, C.R. Johnson, and M.G. Larimore, *Theory and Design of Adaptive Filters*, Wiley, 1987.
- [6] T. Alexander, Adaptive Signal Processing, Springer-Verlag, 1986.
- [7] G. Goodwin and K. Sin, Adaptive Filtering Prediction and Control, Prentice-Hall, 1984.
- [8] M. Bellanger, Adaptive Digital Filters and Signal Analysis, Marcel Dekker, 1987.
- [9] J. Proakis, Digital Communications, McGraw-Hill, 1983.
- [10] C. Chen and S. Kuo, "An Interactive Software Package for Adaptive Signal Processing on an IBM Person Computer," 19th Pittsburgh Conference on Modeling and Simulation, May 1988.
- [11] S. Kuo, G. Ranganathan, P. Gupta, and C. Chen, "Design and Implementation of Adaptive Filters," *IEEE 1988 International Conference on Circuits and Systems*, June 1988.
- [12] S. Kuo, G. Ma, and C. Chen, "An Advanced DSP Code Generator for Adaptive Filters," 1988 ASSP DSP workshop, Sept. 1988.
- [13] Texas Instruments, Second-Generation TMS320 User's Guide, 1987.
- [14] Texas Instruments, Third-Generation TMS320 User's Guide, 1988.
- [15] S. Qureshi, "Adaptive Equalization," Invited Paper, *Proceedings of the IEEE*, Sept. 1985.
- [16] L. Rabiner and R. Schafer, Digital Processing of Speech Signals, Prentice-Hall, 1978.
- [17] N. Jayant and P. Noll, Digital Coding of Waveforms: Principles and Applications to Speech and Video, Prentice-Hall, 1984.
- [18] J. Makhoul, "Linear Prediction: A Tutorial Review," *Proceedings of the IEEE*, April 1975.
- [19] C. Cowan and P. Grant, Adaptive Filters, Prentice-Hall, 1985.
- [20] C. Gritton and D. Lin, "Echo Cancellation Algorithms," *IEEE ASSP Magazine*, April 1984.
- [21] D. Messerschmitt, et al, "Digital Voice Echo Canceller with a TMS32020," in *Digital Signal Processing Applications with the TMS320 Family*, Prentice-Hall, 1986.
- [22] B. Widrow, et al, "Adaptive Noise Cancelling: Principles and Applications," *Proceedings of the IEEE*, December 1975.
- [23] A. Lovrich and R. Simar, "Implementation of FIR/IIR Filter with the TMS32010/TMS32020," in *Digital Signal Processing Applications with the TMS320 Family*, Texas Instruments, 1986.

- [24] S. Orfanidis, Optimum Signal Processing, MacMillan, 1985.
- [25] G. Frantz, K. Lin, J. Reimer, and J. Bradley, "The Texas Instruments TMS320C25 Digital Signal Microcomputer," *IEEE Micro*, December 1986.
- [26] B. Friedlander, "Lattice Filters for Adaptive Processing," *Proceedings of the IEEE*, August 1982.
- [27] A. Gersho, "Adaptive Filtering with Binary Reinforcement," *IEEE Transactions on Information Theory*, March 1984.
- [28] A. Oppenheim and R. Schafer, *Digital Signal Processing*, Chap. 9, Prentice-Hall, 1975.
- [29] L. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Chap. 5, Prentice-Hall, 1975.
- [30] J. R. Gitlin et al, "On the Design of Gradient Algorithms for Digitally Implemented Adaptive Filters," *IEEE Transactions on Circuit Theory*, March 1973.
- [31] C. Caraiscos and B. Liu, "A Roundoff Error Analysis of the LMS Adaptive Algorithm," *IEEE Transactions on ASSP*, February, 1984.
- [32] J. Cioffi, "Limited-Precision Effects in Adaptive Filtering," *IEEE Transactions on Circuits and Systems*, July 1987.
- [33] R. Crochier, R. Cox, and J. Johnson, "Real-Time Speech Coding," *IEEE Transactions on Communications*, April 1982.

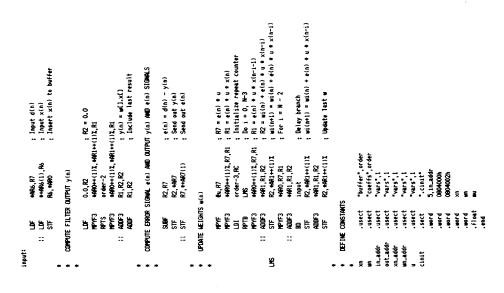
List of Appendices for Implementation of Adaptive Filters with the TMS320C25 and TMS320C30

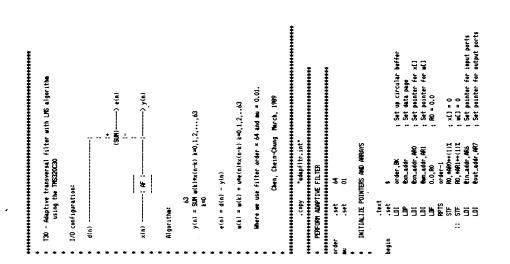
Appendix	Title
A 1	Transversal Structure with LMS Algorithm Using the TMS320C25
A2	Transversal Structure with LMS Algorithm Using the TMS320C30
B 1	Symmetric Transversal Structure with LMS Algorithm Using the
	TMS320C25
B2	Symmetric Transversal Structure with LMS Algorithm Using the
	TMS320C30
C1	Lattice Structure with LMS Algorithm Using the TMS320C25
C2	Lattice Structure with LMS Algorithm Using the TMS320C30
D1	Transversal Structure with Normalized LMS Algorithm Using the
	TMS320C25
D2	Transversal Structure with Normalized LMS Algorithm Using the
	TMS320C30
E1	Transversal Structure with Sign-Error LMS Algorithm Using the
	TMS320C25
E2	Transversal Structure with Sign-Error LMS Algorithm Using the
	TMS320C30
F1	Transversal Structure with Sign-Sign LMS Algorithm Using the TMS320C25
F2	Transversal Structure with Sign-Sign LMS Algorithm Using the TMS320C30
G1	Transversal Structure with Leaky LMS Algorithm Using the TMS320C25
G2	Transversal Structure with Leaky LMS Algorithm Using the TMS320C30
H1	Assembly Subroutine of Transversal Structure with LMS Algorithm Using
	the TMS320C25
H2	Linker Command File for Assembly Main Program Calling a TMS320C25
	Adaptive LMS Transversal Filter Subroutine
Н3	TMS320C30 Adaptive Filter Initialization Program
H4	Assembly Subroutine of Transversal Structure with LMS Algorithm Using
	the TMS320C30
H5	Linker Command/file for Assembly Main Program Calling the TMS320C30
	Adaptive LMS Transversal Filter Subroutine
I1	C Subroutine of Transversal Structure with LMS Algorithm Using the
	TMS320C25
I2	C Subroutine of Transversal Structure with LMS Algorithm Using the
	TMS320C30

Appendix A1. Transversal Structure with LMS Algorithm Using the TMS320C25

10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	title.	.title 'TLNS'	* .	ERVE ADDRE	RESERVE ADDRESSES FOR PARAMETERS		
0			• 6	•			
00Eusect 'parameters', 1 00Eusect 'pa	TLMS : Adapti	e Filter Using Transversal Structure	5 \$. usect	parameters, 1		
ONE 1 1985CT PARAMETER'S 1, 100E 1 1987CT PARAMETER'S 1, 110E 1987CT PARAME	il pue	S Alaberithm Langed Code	= 1	. usect	parameters , 1		
0.06:usect 'paraeters', 1 0.06:usect 'paraeters', 1 0.06:usect 'paraeters', 1 0.06:usect 'paraeters', 1 0.07:usect 'paraeters', 1 0.07:usect 'paraeters', 1 0.07:usect 'paraeters', 1 0.07:usect 'paraeters', 1 0.08:usect 'paraeters', 1 0.08:	,		ERR	.usect	parameters , 1		
0. LOWER PRESENT STATES OF PRE		-	:#6 6	.usect	parameters , 1		
0. LT EPRE PROPER TO ADDRESS 1 1			5	.usect	"parameters".		
### FESTINNIE THE SIGNAL Y ###################################			:3083	474311	- Parameters		
* FORTOR TR ADPTILE FILTER * * * ** ** ** ** ** ** ** ** ** ** **		*					
* FESTINAIR THE SIGNAL Y * ESTINAIR THE SIGNAL Y * LIAPP AR3 * COMPUTE THE SIGNAL Y * COMPUTE THE ESTONAL * SACH ESTONAL * COMPUTE THE ESTONAL * SACH ESTONAL * COMPUTE THE ESTONAL * SACH ESTONAL * COMPUTE THE ESTONAL *					***************************************		
### FERTINATE THE STONAL Y #### LAME ##### CONF.15 LAME ##### CONF.15 LAME ###################################			*	-08M THE A	MPTIVE FILTER		
* ESTIMATE THE SIGNAL Y * LUAPP ARC3 ONFT ORDER-1 * LUAP ARC3, IA ON PASS ARC4 * COMPUTE THE ESPON * COMPUTE THE ESPON * LUAPA ARC4 * LUAPA ARC4 * LUAPA ARC5, IA * LUAPA ARC5, IA * LUAPA ARC5, IA * LUAPA ARC6 * LUAPA ARC6, IA * LUAPA ARC7, IA * SACH * LARC7 *		<u>.</u>	******	***************************************	*************		
0. LT ENGEN-1				. text			
OFF AR3 OFF	x(u)		•				
0. IT BRR 1000-15 SACH 100-15		****	153 *	HATE THE	V WAST.		
0 has FIR HATCH OWE, 15 1.00			. *				
OFF 645 OFF	Algoriths:			8	ş		
0. LT ENR HALL GREEN-1 1. LAW, CORE, 15 1. LAW, CORES, 14 1. LAW, CORES, 14 1. LAW, CORES, 14 1. CORPUTE THE ENROR 1. LAW, ENROR 1. LAW, CORES-1 1. LAW, ARC, MAIL GREEN-1 1. CORPUTE THE BREEN-1 1. CORPUTE THE BRE					2	;	
# PY	Ş			Ì		; Configure BO as program memory	
1.00 CONE.15 1	1 (a)	67 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6		¥	0	; Clear the P register	
LIRIK MERS.1M FIR RPTK ORGEN-1 MACIA MAN-FORON, ← 1 APAC SACH Y APAC SACH Y BACH EBROR COMPOUTE THE METIONTS COMPOUTE THE METIONTS COMPOUTE THE EBROR COMPOUTE THE BROR COMPOUTE THE METIONTS COMPOUTE THE BROR COMPOUTE THE COMPOUTE THE METIONTS COMPOUTE THE COMPOUTE T	- (a)	11R/#X1R=K) K=U,1,2,,03		¥	OFE, 15	; Using rounding	
FIR NeTTK ORDER—1 MACO Merch (1900), ↑ SACH Y SACH Y SACH PER * UPDATE THE EPROP * UPDATE THE METONTS * SACH PERF * UPDATE THE METONTS	2			ž	AR3, XN	: point to the oldest sample	
NMCD WHYDON, +- ONED PRAC SACH Y COMPOUTE THE CHOOR COMPOUTE			FIR	RPTK	ORDER-1	: Repeat N times	
APAC 9404 Y SACH Y COMPUTE THE EMROR COMPUTE THE	(U) = (U)	- y(n)			MH-OF dooh, +-	: Estimate Y(n)	
#PAC 950H Y 1	V.17 V.17.			5		; Configure BO as data memory	
SACH Y * COMPUTE THE EMOR * COMPUTE THE EMOR * LEDATE THE MEIGHTS 0. LT EMBR * PROFITE THE MEIGHTS * LEDATE THE MEIGHTS * SACH ++ O. MRI * HRY *		+ u*e(n)*x(n-K) K=U,1,2,63		APAC			
# COMPUTE THE EPROR # UPDATE THE WEIGHTS 0.	1			58	*	; Store the filter output	
## COMPUTE THE CHROR ## UPDATE THE METONTS 0. LT EPR ## UPDATE THE METONTS 0. LT EPR ## UPDATE THE METONTS #	mere me us	TITLE OF OF END IN I U. U.	•				
MEG ADDH D SOCH EPR • UPDATE THE METGHTS 0. LT EPR HMY U PAC: ADDH CADHET LANK ARI, LONGER-1 LT EPR HMY HMY LANK ARI, LONGER-1 LT EPR HMY +- MRZ ADDH T ZAUR MRZ SOCH ++ O. MRI ETHISS ADDR T MRZ *	e: This so	nce program is the memoric version. I/A configuration has	# *	VIE THE EI	ROR		
ADDAT THE METGHTS 9. UPDATE THE METGHTS 1. UPDATE THE METGHTS 9. LIFE PROPERTY ADDAT THE METGHTS 1. LIFE PROPERTY 1. MARCH	not hee	set up. Hear has to modify the main routine for specific	*				
0. LT EPR + UPDATE THE METGHTS 0. LT EPR HEY U PRC AND ONE, 15 SACH EPR	300 100	ton		9		; ACC = - Y(n)	
9,004 EPR 1				B	0		
0. LT EPR FOR THE MEIGHTS 0. LT EPR FOR U FPG FOR END	ial conditi	::0		3	£	; EBR(n) = D(n) - Y(n)	
0. LT EPR PRY U PRY U PRY U PRY U PRY U PRY L AND ORE, 15 SACH EPRE 1. LARK ARI, ORGEN-1 LARK ARI, ORGEN-1 LT EPRE 1. TERE RY +	=	atus bit should be sens to 01	•				
0. LT EBR 197 U 19		# 10 CO	#G4n +	医医医	GHTS		
	2 SAT	tatus bit smould be set to 1. Hereof DP (data memory page pointer) should be page 0	•				
# 2003 + + 0.0461 *** SACS	4	described the state of the stat		5	Æ	: T = EMR(n)	
PRC AND AND AND AND BREF SACH BREF LOWK ANI, DREFI-1 LOWK ANI, DREFI-1 LOWK AND, DREFI-1 LOWK AND, DREFI-1 LOWK AND, DREF ADAPT ZALR 8-5, MN PPY +-, M2 RPY +-, M2 RPY, M3		remory unc should be 1.		ě	=	. P = U + E98(a)	
# 50CH 155 # 50CH 159 # 1-MAX MAX_MH 1 LIM K MAX_MH 1 LIM		memory U should be 32/.		8			
SACH EPPE 1-0400 ARI, 000 EPPE 1-0400 ARI,				8	06.15	. Round the result	
100 + 10000-1 100 + 10000-1 101 +		Chen, Chein-Chung Pebruary, 1989		9		· EB86 = II + EB8(a)	
LARK ARI, DRESE-1 REVI ← RRZ REVI ←			•				
URK R62, MA 100 MB 100				Š	AR1, ORDER-1	; Set up counter	
UBLK A803,1841 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	THE DODGETT	8		ž	MS2, LM	; Point to the coefficients	
11 5995 140.0FT 1/4.0F2 1 140.0FT 1/4.0FZ 1 140.	1	2		ž	AR3, XIN+1	; Point to the data sample	
Zeri-1 Lawre Zeri - 1840 - 1841	•	7		5	385	; Tregister = U + EMR(n)	
ADMPT 2/4.PR +, 46/3 ; 5 5/4/1 + , 10, 48/1 ; 5 5/4/2 ;	200	· -		È	+,442	; P = U * ENR(n) * X(n-k)	
ZW-+ NYW ZW		>	TANDA	ZACR.	*, AR3	: Load ACCH with A(k,n) & round	
1907 1040 -	THE ATTRECES	OF BEFFE AND PREFEITING		AY.	7₩,+	: W(k,n+1) = W(k,n) + P	
buffer, COORF-1 SACH ++, O, ARI ; SACH ++, O, ARI ; buffer*, 1 ** *coeffs*, COORFR ** *coeffs*, COORFR ** * FINISH		S of burief He Cuerticishis	•			: P = U + EPR(n) + X(n-k)	
**************************************	****	1-03000>54-		38	**.0, AR1	: Store W(k,n+1)	
Coeffs, 000ER	. usect	"Dutter", Ukulen-1		BANZ	ADAPT + ARZ		
PINIS HERE	. esect	Detrer', 1	+		-		
	.45601	"coetts", GROER	FINIS	Pu			

Appendix A2. Transversal Structure with LMS Algorithm Using the TMS320C30





Appendix B1. Symmetric Transversal Structure with LMS Algorithm Using the TMS320C25

.title 'Y25'	****	*****		***************************************	
***************************************	+ DEFINE P	DEFINE PARAMETERS			
25 : Adaptive Filter Ubing Symmetry Transversal Structure and US Algorithm, Looped Code	# ORDER: . ORDER2: .	\$ P	3.82		
(U)	# DEFINE A	SESSEC	STUGICIERS OF REFER SO SESSION PRIESO	STOLINGS	
+: + (u)^					
: A. F. :>(SLM)> e(n)			buffer ORDER2-1		
			Duffer, 1		
21(n) ; ; ; z1(n-k)			coeffs ONDER2		
	. FESTAL		Coeffs , UKUEK-1		
		. usect	CO@ffs", 1		
 <u>]</u> <u>]</u>	* RESERVE	ADDRESSES	RESERVE ADDRESSES FOR PARAMETERS		
x(n)					
	ä		"parameters", 1		
; <u> </u>		-	parameters", 1		
; Z : (MTS) (MTS) (MTS) (MTS)			parameters", 1		
<u>I</u>			parameters",1		
- / <u>-</u> / / / / / / / / /			"parameters", 1		
7 1 1	ERRF:	.usect	parameters",1		
I I	********************	=======================================	********		
	* PERFORM	PEDFORM THE ADMPTIVE FILTER	WE FILTER		
il gerithe:	*************************	******	***********		
	•	.text			
$21(n-k) \approx x(n-k) + x(n-63+k) k=0,1,,31$	* CYMECTRI	CVINCTIBIL DIESED ANDTION	MOTTON		
æ	***	5			
$y(n) = SIM w(k) + x(n-k) + w_0, 1, 2,, 31$		_	883		
· ·	_		981, ORDER2-1	; Set up the counter	
	<i>a</i>	_	MP2, LASDAT	; Point to oldest data	
e(u) = d(u) - y(u)	3		W3, FRSDAT	; Point to newest data	
		_	MA, FRSBUF	; Point to first buffer	
u(k) = u(k) + ube(n) + 21(n-k) k = 0, 1, 2, 31	I HAS	.	#,0,AR2		
	€ .		+,0,484		
Where we use filter order π 64 and $mu = 0.01$.	en a		**,0,4K1	$\frac{1}{2}$ Buffer(k) = DAI(n+k) + DAI(n-N+k)	
Mate: This source program is the paperic version: 1/0 configuration has	•		, , , un		
not been set up. User has to modify the main routine for specific	+ ESTIMATE	ESTIMATE THE SIGNAL Y	× 1		
application.	•	į		; ;	
	21	ì		; Contigure BU as program memory	
AITIS! CONSISTION:		.		; Clear the P register	
TO ALL THE PROPERTY OF THE PRO			Me, 13	Desire A. At. 114-4 bucks	
2) SAM STATUS DIT SHOULD DE SET TO 10GIC I.		ź è	MIS, USBUT	Description the oldest purrer	
3) The cufrent of toats memory page pointer should be page o.			MADE AT 1	; repeat NZ time	
S) Determinenty one secured by 1.			MATCH GOOIL, W	Configure 100 or data access:	
3) Date memory U should be 32/.				Configure to as oata many	
Chen, Chein-Chung February, 1989	: 05	- A		; Store the filter output	
	•				
	-	-			

```
: Load ACCH with A(k,n) & round
; W(k,n+1) = W(k,n) + P
                                                                                                                                                                                                                                                                                  ; Set pointer
; Repeat N-1 times
; Shift data for next iteration
                                                                                                                                              ; Point to the coefficients ; Point to the last buffer
                                                                                                                                                                    : T register = U * ERR(n)
                                                                                                                                                                              P = U + ERR(n) + X(n-k)
                                                                                                                                                                                                                 ; P = U + ERR(n) + X(n-k)
                       ; ERR(A) = D(A) - Y(A)
                                                                                                   ; EARF = U + EAR(n)
                                                                                                                                                                                                                            ; Store W(k,n+1)
                                                                              . P = U + EBR(n)
                                                                                                                                     ; Set up counter
; ACC = - Y(n)
                                                                   ; T = ERR(n)
                                                                                                                                                                                                                                                            UPDATE DATA POSTION FOR NEXT ITERATION
                                                                                                                                                                                                                                                                                  AR2, LASDAT-1
ORDER-2
                                                                                                                                    AR1, ORDER2-1
                                                                                                                                                                                                                                      ADAPT, 4-, AR2
                                                                                                                                                          AR3, LASBUF
                                                                                                                                                                                                                            **,0,AR1
                                                                                                   OME, 15
EPRF
                                                                                                                                                AR2, LE
                                                                                                                                                                                ±-,AR2
                                                                                                                                                                                                      +, AR2
            D, 15
                                              UPDATE THE METGHTS
  £ $ £
                                                                                                                                     ¥ = E E E
                                                                                                                                                                                           ZALR
                                                                                                                                                                                                                            # SEC
                                                                                                                                                                                                                                                                                   F F F
                                                                                                                                                                                                                                                                                                                                end.
                                                                                                                                                                                                                                                                                   FINISH
                                                                                                                                                                                                                                                                                             DATMOV
                                                                                                                                                                                            ADAPT
```

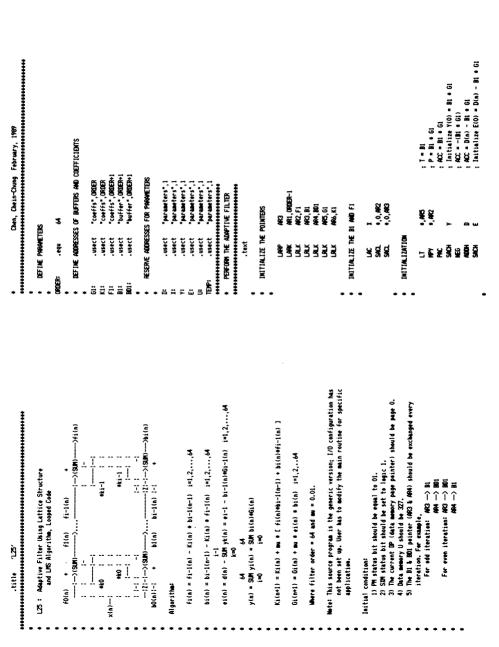
Appendix B2. Symmetric Transversal Structure with LMS Algorithm Using the TMS320C30

	ā	C#.0#	: Set backward bointer for x()
	5 E	order/2-2,RC INNER	
	ADDF3	+AR4++(1)Z, +AR5	-(1)%,R1 ; z(n) = x[n-i] + x[n+N-i]
.: . 89	MPYF3 STF ADDF3	RI, +4RI++(1), R3 RI, +4R2++(1) R3, R2, R2	; y(] = w[].z[] ; Store z(n) ; Accumulate the result
	ADDF3	#884++(1)Z,#885(-(1)Z,R1 ; z(n) = x[n-i] + x[n+N-i]
::	HPYF3 STF ADDF	R1, #4R1—(IRO), R3 R1, #4R2—(IRO) R3, R2	; y[] = w[].z[] ; Sfore z(n) ; Include last result
D. COMPUT	E EPROR S	COMPUTE ERROR SIGNAL e(n) AND OUTPUT	Ty(n) and e(n) SIGNOLS
	SEE.	R2,R7	; e(u) = q(u) - k(u)
==	SIF	R2, +487	Send out y(n)
UPDATE	MEIGHTS W(n)	((u)	
	#PYF	€u, R7	; R7 = e(n) + u
	F 1	#482++(1),R7,R1	R1 = e(n) + u + z(n)
	; E	UKS 1, 100	; Do i = 0, N+3
	MPYF3	##62++(1),R7,R1	
:: <u>¥</u>	6	**************************************	R2 = wi(n) + e(n) + u +
	PYF3	#482(IRO),R7,R1	$\frac{1}{2} \ln(0+1) = \ln(0) + e(0) + u + 2(0-1)$ $\frac{1}{2} \ln u - 2$
==	ADDF3	+4R1,R1,R2	•
	8	input	Delay branch
	ADDF3	KZ, #48(1++(1)	$\{ \{ \{ \{ \{ \{ \{ \} \} \} \} \} \} \} \} \} \} \} \} \}$
	¥.	R2, ##81-(IR0)	
DEFINE	CONSTANTS		
	. USect	"huffer" order	
	.usect	"coeffs", order/2	
	.usect	"coeffs", order/2	
in_addr	.usect	vars",1	
out_addr	.usect	vers , 1	
xn_addr	. usect	vars", 1	
m_addr	.usect	vers , 1	
Zn_206r	.usect	Vers .	
	. USect	vers , 1	

LMS algorithm using the TMS320C30	orithm: $z(n-k) = x(n-k-1) + x(n-k3+k) = 1 \dots 3$	31	y(n) = SUM w(k)+z(n-k) k=0,1,2,,31	2	e(n) = d(n) - y(n)	u(k) = u(k) + u = (n) + 2(n-k) = 0,1,2,31	Where we use filter order $= 64$ and mu $= 0.01$	PERCORN ADAPTIVE FILTR	***************************************	opy "adapfitr.int"	 35 .	3716 4316 1	INITIALIZE POINTERS AND AGRAYS	ext	I order.BK : Set us circular buffer	Em_addr ; Set	Exn. addr, ARO ; Set	Ben_adder, ARI ; Set	Ezn.addr, MC	i orgen/2-1, 1M ; Set insex pointer	5 order-1		order/2-2	RO, #4R1++(1) ; w[]	10, 48C+(1) ; z(1)	F NO. 4602—(180) ; W() = 0	ein addr. 1866 . Set	Cout_addr, AG7	 1486.R7	
6	Algorith a: z(n-k) = x	8	y(n) = SUM	?	e(n) = d(n	W(k) = W(k	Were we u	FIECHN ADAP	***************************************	. cepy	·set	í	TIAL 12E POI	.text	É	ŝ	ē	5 :	5 5	<u> </u>	<u>چ</u> د	SIF	P TS		<u></u>	# # ::		ē		

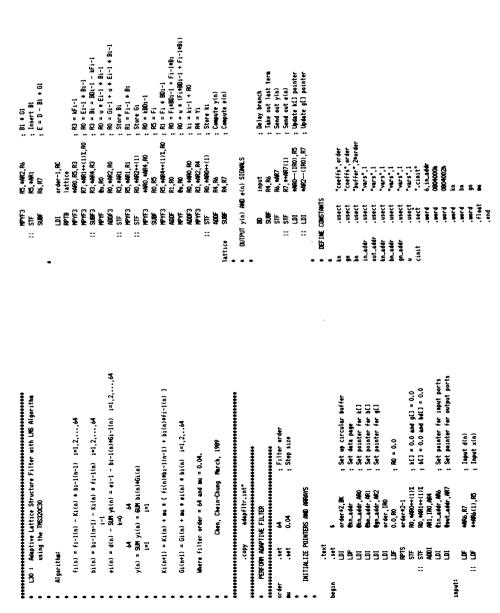


Appendix C1. Lattice Structure with LMS Algorithm Using the TMS320C25



).	CONFUTE THE FILL AND BILL)	(T) (B) (T)	
. [4]	9.00	, de	. ACF = F1-1
	-	790	-
	į		G 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4
	£		- L'I-INGALY - I-II-
	5	0	Store F1
	3	£.	-12
	ž	•	: ACC = 1801-1 - K1 + F1-1
	5	1	: Store Di
æ. •	COMPUTE GAIN G(n)	2	
	5	0	J. T.
	ě		: P = RU • E1-1
	£.	9	: Store MU + E1-1
	5	<u>10</u>	; T = MU + E1-1
	ě	£.	; P = MJ • E1-1 • B1-1
	ž	•,483	: ACC = 61(n)
	Ę	S.	; ACC = G1(n) + MJ+E1-1+B1-1, T = B1
EII	95	**,0,ARS	; Store G1(n+1)
- 00	COMPUTE E AND UPDATE KI	PDATE K1	
	Ě	. 18 2	
	ZALS	w	: E1-1
	S.A.	±, A €2	; ACC = E1-1 - B1 + G1, P = B1+F1-1
	Š	ш	
	4		; T = F1, ACC = B1+F1-1
	Ž	\$ ±	: P = F₁ + 180₁-1
	PPRC		: ACC = B1 #f1-1 + B01-1 + F1
	3	J. Die	
	5	963	: T = B1 of 1 - 1 + B01 - 1 + F1
	ě	0	; P = MJ + (F1+801-1+F1-1+81)
	ZALR		; ACC = K1(h)
	AP.		; ACC = K1(n) + MU
			; • (F1+801-1+F1-1+B1)
	58	186,0,1	; Store K1(n+1)
	Ž	LAT1, 4-, AR2	
100	COMPUTE Y		
	1		: Configure BO as program memory
	ž	•	
	SK SK		2
	ž	M2,81	: Set the pointer
æ	æ Æ	060ER-1	; Repeat M times
	¥	61+0f-d00h, ++	: Compute Y
	5		ě
	AP AC		; Include last data
	5	-	; Store the filter output
	es.		

Appendix C2. Lattice Structure with LMS Algorithm Using the TMS320C30



Appendix D1. Transversal Structure with Normalized LMS Algorithm Using the TMS320C25

.usect	EMR: .usect "parameters",1	OME: .usect "parameters",1	. usect		.0360	WRG: .usect "parameters",1	********************	. Detromate the Analysis on the	*****	text.			* ESTIMATE THE PUMER OF STUMPL		_	LPUX APC3, X0 ; Point to input signal X		100		E STATE OFFI	-	Coor, Stirl	(c)x * :	SACH WAR ; Store WAR(a)		A MODEL THE COMMAN	Table like Signer 1	••	MPW 0 : Clear the P register	1 36			ישנו מתרעו	##+01-000h, #-	CMFD ; Configure BO as data memory		SACH Y ; Store the filter output	And the state of t	* CAMPUTE THE EMACK		0	(u) - (u) = D(u) - \(u)	•	+ UPDATE THE LETIENTS	(*/883 ± L · · · · · · · · · · · · · · · · · ·	.		;	ADD ONE, 15 ; Round the result		* MORNA 175 COMMERCE EACTOR			PPTK 14 ; Repeat 15 times
ititie (1905)	- 3		And the state of t	INCS: Adaptive Filter Using Iransversal Structure	and Mormalized LMS Algorithm .Leoped Code			Algerithm:	S		y(n) = Sin e(k) ex(n-k) = k = 0, 1, 2,, 65	P		(a) = 4(a) - (b)			VEC(K) = (1T) = VEC(K-1) + T = X(B) = X(B)		$\mathbf{w}(k) = \mathbf{w}(k) + \mathbf{w} + \mathbf{w} + (\mathbf{n} + k) / \mathbf{w} \cdot (k) + \mathbf{w} \cdot (1, 2, 63)$		Where we use filter order = 64 and mu = 0.01.		with This course contrast is the essent warrion. 1/0 configuration has	The society of the base of the base of the training of the tra	not been set up. Oser has to modify the main rotating to specific	application.		ITTEL CONGILION:	1) FT status bit should be equal to U.	2) SXM status bit should be set to 1.	3) The current DP (data memory page pointer) should be page 0.	4) Data memory Off should be 1.	5) Data memory U should be 327.	4) Data access und aband he initialized to Offfth.		Charles Calendary 1999	Chen, Cheth-Chung regrowty, 1707			THE INC. DADORETTEDS			SHIFT: .equ 7	PAGE0: .equ 0	DEFINE ADDRESSES OF BUFFER AND COEFFICIBITS		neart "buffer" ORDER-1			.usect "coeffs", ONDER		RESERVE ADDRESSES FOR PARAMETERS		.usect "parameters",1

Appendix D2. Transversal Structure with Normalized LMS Algorithm Using the TMS320C30

	J.	86,86	; R6 = x2
	¥	er_1, R6	; R6 = (1-r) + x2
	ä	£,π	
	¥	Evar, R3	; R3 = r + var(n-1)
	8	177	
	CURTULE FILLER WOLFUL YIN	iroi yan	
	5	0.0,172	; R2 = 0.0
	IP YF3	###0++(1)X,###1++(1)X,R1	1) I, R1
==	AG	84,83	
	SIF	R3, Evar	; Restere var(n)
	RPTS	order-2	
•		,	ă
=		PLEASE DO DO(a)	114,N1
=		7,72,12	; y(n) = mil.Xii . Techuda last pasult
	į	2.	1000 100 100 100 100 100 100 100 100 10
TU-PUT	COMPUTE ERROR STOWN, e(n)	MAR. e(n)	
	300	R2,R7	; e(n) = d(n) - y(n)
•			
* OUTPUT	y(n) AMD 4	OUTPUT y(n) AND e(n) SIGNALS	
•	Ę	700** 00	(a): the Park
;		KZ, ##K/	
::	SIL.	R7, ++48(7(1)	; Send out e(n)
•	2000		
	UPDATE METURIS BITTY	2	
•	SHORE	2	. Cassute (Austra)
	8	2 &	. Car(a) II a 6 24
	3	-7 2	
	į į	2,5	
	9	. 2	. Most as bayes 2-s-1
	į	2 2	
	2	. 2	
	Ę.	! 2 2	; Now R2 = $x[0] = 1.0 + 2-e^{-1}$.
•			
	¥	82,83,80	*
		2.0,R0	= 2.0 - v • x[0]
	E AL	22,02	; R2 = x(11 = x(01 (2.0 - v + x(01))
•	ş	8	
	=	KZ, K3, N0	* ·
		2.0,80	= 2.0 - v + x(1)
	J.	22,02	R2 = x[2] = x[1] + (2.0 - v + x[1])
•	200	8	(C)
	9	0 000	. BO = 2 0 = x + y[2]
	T.	79°.62	: E
•			
	HPYF.	R2,R3,R0	; R0 = v # x[3]

* ESTIMATE THE POWER OF THE IMPUT SIGNAL.*

- Adaptive transversal	Adambient Appending									And the state of t
y(a) = SM w(k)*x(n-1) + () w(a) = SM w(k)*x(n-1) + () e(a) = (a) - y(n) w(b) = w(k) - y(n) w(c) = d(n) - y(n) Chen, Cheir Chen, Che	Using the TRESCOCO gorithm:	Using the TMS20020 Using the TMS200200 Using the TMS2002000000000000000000000000000000000	0. shaptive trainersal filter with Mormalized using the TMSS20020 (garithat: 6.3 y(a) = SiM with Px(n-t) Px(1,2,,63 re) x(a) = d(a) - y(a) with = with + we(a) Px(n-t) / y(a) Px(a) Habere we use filter order = 64 and mm = 0.01. Chen, Chein-Chang Murch, 1989 Historian Chang Murch, 1989 Historian Chen, Chein-Chang Murch, 1989 Historian Chen, Chen, 1989 Historian Chen, 1989	Solution Comparison Compa	Water Wate	Wing the TMS20020		Wing the INSCOCCO Section 1 Normalized using the INSCOCCO	0 - Adaptive transversal filter with Normalized using the TMS22020 (gorithm: 6(a) = SM with Pac(n-t) + (1-1)Pac(n)Pac(n) e(a) = fa(n) - y(n) with = with + une(n)Pac(n-t) + (1-1)Pac(n)Pac(n) (Chen, Chein-Chang March, 1989 copy "adaptive.int" SEGGN ADMPTINE FILTER set 64 ; Filter order .set 0.00 ; Step size	Š
using the TMS220C20 y(a) = SM utilex(n-t) + (1-1)tx(a)tx(a) e(a) = d(a) - y(a) u(t) = u(t) + uve(a)tx(n-t)/vur(a) t=0,1,2 Ghe, Chein-Chang Murch, 1989 .copy "adapfite.int" .copy "adapfite.int" .set 0.01 Sfets size .set 0.096 Sfets size .set 10	State Stat	Signature 1 Signature	Solution Compared	0 Adaptive transversal filter with Marmalized using the TMSS20200 (gorithm: 6(a) = SM wither(n-t) + (1-1) Pa(a) Pa(a) 7(a) = SM wither(n-1) + (1-1) Pa(a) Pa(a) 8(b) = with - y(n) 8(c) = d(n) - y(n) 8(c) = d(n) - y(n) 8(c) = d(n) - y(n) 100	Water Wate	Wing the TMS20020		Wing the INSCOCCO	0 - Adaptive transversal filter with Normalized using the TMS22020 (gorithm: 63 y(a) = SM with Px(n-t) + (1-1) Px(a) Px(a) x=(a) = chan(n-1) + (1-1) Px(a) Px(a) (a) = d(n) - y(n) with = w(t) + we(a) Px(a-t)/var(a) Px0,1,2 (b) = d(n) - y(n) with = w(t) + we(a) Px(a-t)/var(a) Px0,1,2 (c) = d(n) - y(n) SEGORA OMPTHE FILER 1.0 10 10 10 10 1.0 - alpha 1.14 IZE POINTESS AND ARRAYS 1.10 10 10 10 1.10 - alpha 1.14 IZE POINTESS AND ARRAYS 1.10 10 10 1.15 10 10 1.16 10 10 1.17 11 12 1.18 10 10 1.19 10 10 1.10 - alpha 1.11 10 10 1.11 10 10 1.12 10 10 1.13 10 10 1.14 10 10 1.15 10 10 1.16 10 10 1.17 11 12 1.17 12 13 1.18 13 14 1.19 14 15 1.10 11 11 1.11 12 13 1.11 13 14 1.11 15 15 1.11 15	1
using the TNS220C20 y(a) = SM u(t)**(n+t) k=0,1,2,,63 k=0 var(a) = r**uar(n-1) + (1-r)**(a)**c(a) u(t) = u(t) + u**e(a)**c(a+t)/var(a) k=0,1,2, Chen, Chein-Champ Murch, 1989 coy	gorithm:	using the TMSSACCO yerithm: 63 y(a) = SM with Par(n-t) k=0,1,2,,63 e(a) = e(a) - y(a) with = with + wet(a) k=(a+1) yer(a) k=0,1,2, libers we use filter order = 64 and mm = 0.01. Chen, Chein-Chang March, 1889 cop	Wing the TMSS2020 Using the TMSS2020 yerithm: e(n) = 5M u(t)=x(n-t) k=0,1,2,,63 e(n) = 4(n) - y(n) u(t) = u(t) + u*e(n)=x(n-t)/var(n) k=0,1,2, lakers we use filter order = 6M and mm = 0,01. Chen, Chein-Chang March, 1989 THE Chen, Chen, 1989 THE Chen, Chen, 1989 THE Chen, Chen, 1989 THE Chen, Chen, 1989 THE Che	0. Adaptive transversal filter with Marmalized using the TMSS20230 (garithms: 63 y(a) = SiM with Px(n-t) P=0,1,2,,63 e(n) = d(n) - y(n) with = with - wie(a) Px(n-t)/war(a) Px0,1,2, liners we use filter order = 64 and ma = 0,01. Chen, Chein-Chang Phrch, 1989 **********************************	War	Wing the TMS20C20	10 - Adaptive transversal filter with Normalized using the TMS220C20	On - Adaptive transversal filter with Marmalized using the INSZOGZO gorithm: 6.3 y(n) = SM wither(n+t) k=0,1,2,,63 war(n) = rewar(n-1) + (1-r)tx(n)tx(n) with = with + wither(n+t)/war(n) k=0,1,2, there we use filter order = 64 and mu = 0.01. Chen, Chein-Chang Murch, 1989 copy	0 - Adaptive transversal filter with Normalized using the TMSS20C20 y(a) = SiM with Pax(a+t) P=0,1,2,,63 x c	
using the TNS220C20 y(n) = SM uithex(n-t) k=0,1,2,,63 k=0 var(a) = revar(n-1) + (1-r)ex(a)ex(a) u(t) = u(t) + uve(a)ex(a-t)/var(a) l=0,1,2, Gen Gein - J(n) u(t) = u(t) + uve(a)ex(a-t)/var(a) l=0,1,2, Gen Gein-Chang Murch, 1989 Total - John - Jo	State Stat	Signature 1 Signature	Solution 1 Solu	10 Adaptive transversal filter with Marmalized using the TMS220200	War Marker War W	Wing the TMS20020		Wing the INSCOCCO Graph New Institute Wing the INSCOCCO	0 - Adaptive transversal filter with Normalized using the TMS2QC20	i
using the TNS220C20 y(a) = SM u(t)**(n+t) k=0,1,2,,63 k=0 var(a) = f*vaar(a-1) + (1-+)**(a)**(a) u(t) = u(t) + u**(a)**(a)**(a) k=0,1,2,,63 u(t) = d(a) - y(a) u(t) = u(t) + u**(a)**(a)**(a) k=0,1,2,,63 LOPE OF CHAIR CH	gorithm:	yerithm: 63 y(a) = SM wither(n-t) k=0,1,2,,63 k=0 var(a) = rewar(n-1) + (1-r)tx(a)tx(a) with = with + wer(a)tx(a-t)/var(a) k=0,1,2, with = with + wer(a)tx(a-t)/var(a) k=0,1,2, Chen, Chein-Chang March, 1989 coy	Wing the TMSSOCOO Using the TMSSOCOO yerithm: (c) = 54(n) - y(n) with = with + wee(n)+x(n+1) x=0,1,2,,63 with = with + wee(n)+x(n+1)/war(n) x=0,1,2, The construction order = 64 and mu = 0.01. Chee, Chein-Cheng Aurch, 1589 coy	00 - Adaptive transversal filter with Normalized using the TMSS20C20 (garithms: 63 y(a) = SiM with bx(n+t) k=0.1.263 k=0 var(a) = rever(n+1) + (1-r) bx(a) bx(a) e(n) = d(n) - y(n) with = with - wien(a) bx(n+1)/var(a) k=0.1.2 libers we use filter order = 64 and ma = 0.01. Chen, Chein-Chang Murch, 1599 cop	Solution Solution Solution Solution	10 - Adaptive transversal filter with Normalized using the TMS220C20		Using the TNSZOCOO goritha: y(a) = SM with Pec(n-t) k=0,1,2,,63 k=0, = SM with Pec(n-t) k=0,1,2,,63 war(a) = rewar(a-1) + (1-r)Pec(a)Pec(a) with = w(t) + wer(a)Pec(a-t)/war(a) k=0,1,2, Chen, Chein-Chang Murch, 1989coy	0 - Adaptive transversal filter with Normalized using the TMSS20230 y(a) = SAW with Pac(n+t) + Col.1,2,63 e Col. = d(a) - y(a) with = with + wet(a)Pac(n+t)/war(a) Pac.1,2, Chen, Chein-Chung March, 1989 coy	2
using the TNS220C20 y(a) = SM uit)=x(n-t) k=0,1,2,,63 k=0 var(a) = rewar(n-1) + (1-r)=x(n)=x(n) uit) = u(t) + uve(n)=x(n-t)/var(n) k=0,1,2, laker we use filter order = 64 and mu = 0.01. Chen, Chein-Chang Murch, 1989 Cony "adapfitr.int" SECON ADMPTIVE FILTER	using the TNGSZOGOO gorithm: (c) = SM wither(n-t) k=0,1,2,,63 y(a) = SM wither(n-t) k=0,1,2,,63 war(a) = rewar(n-1) + (1-r)tx(a)tx(a) with = with - with = with + wither(a)ty(a) (c) = d(a) - y(a) with = with + wither(a+t)/war(a) k=0,1,2, with = with + wither(a+t)/war(a) k=0,1,2, (c) = d(a) - y(a) (d) = d(a) - y(a)	using the TMSS20C20 yerithm: 63 y(a) = 5M u(t) > (a-1) + (1-1) > (a-1) > (a-1) = 5M u(t) > (a-1) + (a-1) >	Wing the TMSS2020 (garithm: 63 y(a) = 5M with Pac(n-t) k=0,1,2,,63 e(b) = 5M with Pac(n-t) k=0,1,2,,63 e(c) = d(a) - y(a) with = with + we(a) Pac(n-t)/Pac(n) k=0,1,2, (Chen, Chein-Chang Murch, 1989 .copy "adapfler.int" .set 64 ; Filter under .set 64 ; Filter under .set 64 ; Filter under .set 60 i Step size .set 1.0 ; Step size .set 0.00 ; Step size .set	00 - Adaptive transversal filter with Mormalized using the TMS220200 (gorithm: 63 y(a) = SM withor(n-t) h=0.1,2,63 ked) - y(n) with = with - whe(a) bx(n-t)/war(a) b=0.1,2 Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chang Murch, 1989 **********************************	00 - Adaptive transversal filter with Mormalized using the TNESQCEQ y(n) = SSM with Sx(n-k) k=0,1,2,,63 k=0	Wing the TMS20020		Using the TNG2XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	0 - Adaptive transversal filter with Normalized using the TMS220C20	Ē
y(n) = SLM u(k)=x(n-k) 1 (1) = (10) =	y(a) = SIM w(k)*x(n-k) y(a) = SIM w(k)*x(n-k) e(a) = ((a) - y(a) w(k) = w(k) - y(a) w(k) = w(k) - we(a)*x(i copy	Septime Trainersal	Wing the INSESSOCSO	Genithai 63 y(a) = SiM u(k) = k(n+1) + (1) e(a) = SiM u(k) = k(n+1) + (1) e(a) = sim u(k) + une(a) = k(n+1) + (1) u(k) = u(k) + une(a) = k(n+1) = (1) u(k) = u(k) + une(a) = k(n+1) = (1) c(a) = u(k) + une(a) = k(n+1) = (1) c(b) = u(k) + une(a) = k(n+1) = (1) c(c) = u(k) + une(a) = (1) c(c) = u(k) + un	yer:that San ut:bac(n-k) y(a) = SM ut:bac(n-k)			0 - Adaptive transverses 0 - Adaptive transverses 0 - 0 - 0 - 0 - 0 - 0 - 0 - 0 - 0 - 0	Goritha: 63 y(a) = SM w(k)*e(n+1) + (1 e(a) = 4(a) - y(a) w(k) = w(k) + we(a)*e(i Mere we use filter ord Mere we use filter ord Chen, Chein Chen, Chein Set 0.01 Set 0.09 Set 1.0 Set 0.09 Set 1.0 Set 0.00 Set 0.0	
y(n) = SLM w(k)*E(n-k)) + (1) (a) = (hn) - y(n) (b) = (hn) - y(n) (c) = d(n) - y(n) (d) = d(n) - y(n) (e) = d(n) - y(n) (e) = d(n) - y(n) (e) = d(n) - y(n) (f) = f(n) - y(n) (y(a) = SM w(k)*x(n+1) + (1) (e(a) = (h) - y(h) war(a) = rewar(a-1) + (1) (e(b) = (h) - y(h) with = u(k) - wer(a)*x(in+1) + (1) with = u(k) - wer(a)*x(in+1) + (1) with = u(k) - wer(a)*x(in+1) + (1) Chen, Cheir Chen, Chen, BK	y(a) = SIM with trainers at wing the INGSOGCO garithm: e(a) = SIM with trainers at the control of the control	Var (a) = SM u(k)*x(n-k)) + (1) var (a) = SM u(k)*x(n-k)) + (1) var (a) = f*var (n-1) + (1) var (b) = d(n) - y(n) var (c) = d(n) - y(n) var (a) = d(n) - y(n) var (b) = u(k) + u*ve(n)*x(n-k) + (1) Chen, Cheir Chen, Chen	Wing the Treatment of Samuel of the Treatment of Samuel	Mark			y(n) = SM w(k)+x(n+k)) y(n) = (n) = y(n) + y(n) = y(n) + y(n) = y(n) + y(n) = y(n) + y(n) = y(n) = y(n) + y(n) = y((goritha: (soritha:	
y(a) = SIM w(k)*x(n-k)) + (1) e(a) = (b) = (b) - y(a) war(a) = rewar(a-1) + (1) e(b) = d(n) - y(a) w(k) = w(k) + we(a)*x(1) w(k) = w(k) + we(a)*x(1) w(k) = w(k) + we(a)*x(1) Chen, Chein Chen, C	y(a) = SLM u(k)*x((n-k)) + (1 x x x x x x x x x		Wing the INSESSORESA	War (a) = 50 M w(k) sector=1) (gorithm: 63 y(a) = 50 M w(k) sector=1) + (1 (a) = 50 M w(k) sector=1) + (2 (a) M w(k) sector=1	10	V		10 10 10 10 10 10 10 10	(4-1) (4-1	
y(n) = SLM w(k) = K(n-k) 1 (n) = SLM w(k) = K(n-k) 1 (n) = K(n) - y(n) w(k) = rawer(n-1) + (1) w(k) =	y(a) = SIM w(t) > C(a+1) + (1)	yenithm: y(a) = SM w(k)*x(n+k) } (y(a) = SM w	yenithai y(a) = SIM withter(a+1) + (1) e(b) = SIM withter(a+1) + (1) e(a) = d(a) - y(a) with = w(b) - w(a) with = w(b) + we(a) bect copy - set for the condition of the c	Genithai 63 y(a) = SM w(k) = k(h-k) (i) = (ia) - y(n) w(a) = (ia) - y(n) w(b) = w(b) - y(n) w(c) = (ia) - y(n) w(d) = w(b) - y(n) w(d) = w(b) - y(n) w(d) = w(d) - y(n) w(d) =	y(a) = SM w(k) >c(a) = SM w(k) >c(b) = SM w(k) >c(b) = (b) = (c) = y(a) = revar(a-1) + (i) = revar(a-1) + (i			0 - Adaptive transversal using the INESZOCOO	0 - Adaptive transversal using the INSE20C20 using the INSE20C20 yeritha: e(a) = GA using(n=1) + (1 e(a) = e(a) - y(a) u(t) = u(t) + ue(a) > y(a) u(t) = u(t) + ue(a) + ue(a) > y(a) u(t) = u(t) + ue(a) + ue(a) > y(a) u(t) = u(t) + ue(a) + ue(a) u(t) = u(t) + ue(a) + ue(a) + ue(a) u(t) = u(t) + ue(a) + ue(a) u(t) = u(t) + ue(a) + ue(a) + ue(a) u(t) = u(t) + ue(a) + ue(a)	Ħ
y(a) = SIM w(k)*E(n-k) b y(a) = SIM w(k)*E(n-k) b e-0 var(a) = rewar(a-1) + (1 e(a) = d(a) - y(a) w(k) = w(b) + (we) a)*E(n-k) in the w use filter ord liber, Cheir Chen, Chen	y(a) = SM w(k)*x(n+k) } (c) y(a) = SM w(k)*x(n+k) } (c) y(a) = F*w(a-1) + (1) w(k) = w(a) = f*w(a-1) + (1) w(k) = w(a) = f*w(a-1) + (1) w(k) = w(a) = f*w(a)	y(a) = SM u(k)*x(a-k) + (1) var(a) = sM u(k)*x(a-k) + (1) var(b) = d(a) - y(a) u(k) = u(k) + uve(a)*x(a-k) + (1) coy SpeCon Aberlive Fill R set 0.00 set 0.004 set 0.004 text 2. text 2. text 3. text 3. text 3. text 4. text 6. text 6. text 7. text 6. text 7. text 6. text 7. text 6. text 7. text 7. text 6. text 7. text 6. text 7. text 7. text 6. text 7. text 6. text 7. text 7. text 7. text 6. text 7. text 6. text 7. text 7. text 7. text 6. text 7. text 6. text 7. text 7. text	(garitha: (a) (a) (b) (c) (c) (c) (c) (c) (d) (d) (d	(geritha: 63 (a) = Salk u(k) ex(n-k) 1 (c) = G(n) - y(n) (e) = G(n) - y(n) (e) = G(n) - y(n) (f) = u(k) + uve(n) ex(n-k) 1 (copy adaptive Chen, Cheir Chen,	yer:tha: 63 y(n) = SM w(k) *x(n-k) 1 (1) e(n) = 4(n) - y(n) w(k) = w(k) + we(n)*x(n-k) 1 + (1) e(n) = 4(n) - y(n) w(k) = w(k) + we(n)*x(n-k) 1 + (1) e(n) = 4(n) - y(n) w(k) = w(k) + we(n)*x(n-k) 1 + (1) Chen, Chein Chen,		y(a) = SM w(k) = k(a-k) (i) = (a) (i) = (b) (i) = (a) (i)	yeniths: y(a) = SM w(k)*x(n+k) } y(a) = SM w(k)*x(n+k) } e(a) = d(a) - y(a) war(a) = rewar(a-1) + (1 e(b) = d(b) - y(a) war(b) = u(k) + we(a)*x(i w(k) = u(k) + we(a)*x(i w(k) = u(k) + we(a)*x(i war(a) = d(a) - y(a) year(a) = cop year(a	(geritha: 63 (seritha: 63 (seritha: 63 (seritha: 63 (seritha: 63 (seritha: 63 (seritha: 64 (s	2
y(a) = SM w(k)*x(n-1) + (1) y(a) = SM w(k)*x(n-1) + (1) e(b) = (d(a) - y(n) w(c) = d(a) - y(n) Chen, Cheir Chen,	(a) (b) (c) (c) (c) (c) (c) (c) (c) (c) (c) (c		Wing the INSESSOCSO		yen:thai: 3 y(a) = SM w(k) > k(n+1) + (1) 4(a) = SM w(k) > k(n+1) + (1) 4(a) = (1) - y(a) 4(b) = w(k) + we(a) > k(n) 4(b) = w(k) + we(a) > k(n) 4(c) = w(c) - y(a) 4(d) = w(c) - w(c) 4(d) = w(10	(goritha: 63 y(a) = SUR u(k)=x(n-k) (e/n) = SUR u(k)=x(n-k) + (f/n) = y(n) =	9
y(a) = SIM w(k)*E(n+1) + (1) y(a) = SIM w(k)*E(n+1) + (1) e(a) = (a) - y(a) w(b) = w(b) - y(a) w(c) = d(a) - y(a) Con Control Con Control Set 0.00 Set 1.0 Set 0.00 Se	y(a) = SM w(k)+x(n+1) + (1) (a) = SM w(k)+x(n+1) + (1) (b) = SM w(k)+x(n+1) + (1) (c) = A(n) - y(n) w(k) = d(n) - y(n) w(k) = d(n) - y(n) w(k) = d(n) - y(n) w(k) = w(k) + w(k) + (1) Coopy "adaptity. "Set 64 "Set 64 "Set 64 "Set 1.0 "	y(a) = SIM with trainers at wing the INGSOGCO garitha: e(b) = SIM with trainers at were (a) + (i) + (i) e(b) - y(a) war(a) = ebag (a-1) + (i) e(b) - y(a) with = with - with a with trainers are filter ord liber, Cheir Chen,	Wing the INESCOCCO Using the INESCOCCO Using the INESCOCCO You'(a) = SIM u(k)*x(n+1) + (1 e(a) = d(a) - y(a) u(k) = u(k) + ue(a)*x(i u(k) =	Wing the The State of Wing the The State of Wing the The State of Wing	yerithm: 63 y(a) = SM w(k) *x(n+1) + () war(a) = rewer(n-1) + () e(n) = d(n) - y(n) w(k) = w(k) + wee(a) *x(n+1) + () e(n) = d(n) - y(n) w(k) = w(k) + wee(a) *x(n+1) + () cen Chen, Cheir Chen,		10 - Adaptive transversal 15 15 15 15 15 15 15 1	0 - Adaptive transversal using the INESZOCOO goritha: (a) = SIM u(t)+x(n+t) 1 + (1) (b) = SIM u(t)+x(n+t) 1 + (1) (c) = d(n) - y(n) (d) = d(n) - y(n) (e) = d(n) - y(n) (f) = f(n)	0 - Adaptive transversal using the INSE20C20 using the INSE20C20 y(a) = SM u(k) **(n+1) * (1) e(n) = (h) - y(n) u(k) = u(k) - u(k) + u(e(n) + y(n) Leo, Cheir Chen, Chen, Mannader Chen, Ben, adder Chen, B	Ė
	y(a) = SLM u(k)*x((n-k)) + (1 x x x x x x x x x		Wing the INSERCED Using the INSERCED Genithal		yer:tha: 63 y(a) = SM w(k) > k(n+1) + (1) e(a) = (a) - y(a) w(k) = w we filter ord Mere w we filter ord Chen, Chein Chen, Chein SEFORM ADMYTIVE FILTER "SEFORM ADMYTIVE FILTER "SEFORM ADMYTIVE FILTER "SEF CON "SEF IL	using the TRESCOCO using the TRESCOCO yeritha: 63 y(a) = SSM u(L) + (1) xer(a) = rever(a-1) + (1) xer(b) = d(a) - y(a) u(L) = u(L) + ure(a) bx(i later us use filter ord the color and filter Cheir Ch		10 10 10 10 10 10 10 10	(goritha: 63 y(a) = Sth u(t) ex(n-t) 1 k(a) = Sth u(t) ex(n-t) 1 k(a) = (in) - y(n) w(t) = u(t) + uve(a) ex(n) w(t) = u(t) + uve(a) ex(n) w(t) = u(t) + uve(a) ex(n) w(t) = u(t) - y(n) w(t) = (in) - y(n)	i
10 10 10 10 10 10 10 10	y(a) = SIM w(k)*x(n-k) } (b) = SIM w(k)*x(n-k) } (c) y(a) = SIM w(k)*x(n-k) } (c) y(a) = r**w(a)	yenithm: y(a) = SM withter(a-t)) + (1 + (1 + (1 + (1 + (1 + (1 + (1 +	yenithm: y(a) = SIM withter(a-t) 1 + (1 + (1 + (1 + (1 + (1 + (1 + (1 +	Oc-Adaptive transversal Using the TNSZ20C20 S y(a) = SM w(k)=k(n-k) i e(a) = d(a) - y(a) w(k) = w(k) + we(a)=k(i Mare we use filter ord Chen, Chein Chen, Che	100 - Adaptive transversal using the INSESCOCO 101 102 103 1	Using the TRSZOCZO Using the TRSZOCZO Using the TRSZOCZO Var (a) = SM u(k) **C(n-1) **() Var (a) = chau (k-1) **() Var (a) = chau	10 - Adaptive transversal 10 - Adaptive transversal 10 10 10 10 10 10 10 1	0 - Adaptive transversal using the INESZOCOO	0 - Adaptive transversal using the TNSZOCOO S y(a) = SM u(k) > k(h) = k - k var(a) = rhwar(a-1) + (i c(a) = d(a) - y(a) u(k) = u(k) + une(a) > k(i there we use filter ord Chen, Chein Chen,	Ξ
using the TRSESOCOO y(n) = SIM u(k)**c(n-k) k k-0 var(n) = revar(n-1) + (1 e(n) = d(n) - y(n) u(k) = u(k) + ure(n)**c(n-k) to be a given filter ord The cool order filter order filter set 0.00 set 0.004 set 0.004 text 1.00 set 0.004 text 1.00 set 0.004 text 1.00 set 0.004 text 1.00 set 0.004 text 1.00 set 0.004 text 1.00 set 0.004	y(a) = SM w(k)*x(n-k)) + (1 cc) = SM w(k)*x(n-k)) + (1 cc) = rewar(a-1) + (1 cc) = rew	y(a) = SIM u(k)*x(n-k) + (1) var(a) = SIM u(k)*x(n-k) + (1) var(b) = d(a) - y(a) u(k) = u(k) + uve(a)*x(i inter we use filter ord SECON ADMINISTREETING	Wing the TRESCOCCO Using the TRESCOCCO (garitha: (e(n) = SM u(k)*x(n-k) } + (1) (e(n) = (4n) - y(n) (e(n) = d(n) - y(n) (coy SECOM ADMPTIVE FILEN SECOM ADMPTIVE FILEN SECOM ADMPTIVE FILEN SECOM ADMPTIVE FILEN SECOM SECOM S	(geritha: 63 y(a) = SH w(t) ex(n+1) + (1) e(n) = G(n) - y(n) war(a) = revar(n-1) + (1) e(n) = d(n) - y(n) w(t) = u(t) + ure(a) ex(n) (coy	yeriths: 63 y(n) = SM w(k) *x(n-k) 1 e(n) = 4(n) - y(n) w(k) = w(k) + wee(a)*x(n-k) + (1 e(n) = 4(n) - y(n) w(k) = w(k) + wee(a)*x(n-k) + (2 e(n) = 4(n) - y(n) w(k) = w(k) + wee(a)*x(n-k) + (2 e(n) = 4(n) - y(n) w(k) = w(k) - y(n) w(k)	using the INSZOCCO using the INSZOCCO (geritha: 6.3 (A) = SIN u(k) bec(n-k) + (f k(n) = d(n) - y(n) u(k) = u(k) + uve(n)bec(n-k) u(k) = u(k) + uve(n)bec(n)b	y(a) = SM w(k) = (b) y(a) = SM w(k) = (b) y(b) = SM w(k) = (b) + (f) + (f) = (a) = (b) = (a) = (b) = (a) = (b) = (0 - Adaptive transversal using the INSEZOCOO	(geritha: (a) (geritha: (b) (c) (e(n) = SM u(k) ex(n+1) + (1 + (1 + (1 + (1 + (1 + (1 + (1 +	Ē
			wing the INESCOCCO ligarithm: A A A A A A A A A A A A A	Genithm: 63 y(a) = SUM u(k)=x(n-k) 1 e(n) = SUM u(k)=x(n-k) 1 e(n) = y(n) u(k) = u(k) + une(a)=x(n-k) u(k) = u(k) + une(a)=x(n-k) in the are use filter ord interes are interes Chen. Chein Chen. Chen	yerithm: 63 y(n) = SM w(k) > k(n+1) + (1) e(n) = (n) - y(n) w(k) = (n) - y(n) w(k) = w(k) + we(n) > e(n) w(k) = w(k) - y(n) w(k) = w(k)	Using the TRESCOCO	10 - Adaptive transversal 10 - Adaptive transversal 12 - 13 - 13 - 13 - 13 - 13 - 13 - 13 -	10	(goritha: 63 y(a) = SUR u(k)=x(n-k) (con) = SUR u(k)=x(n-k) (con) = y(n)	1
100 100	y(a) = SIM w(t) be(n-t) 1 y(a) = SIM w(t) be(n-t) 1 e(a) = (a) - y(a) w(b) = w(b) - y(a) w(t) = w(b) - y(a) w(t) = w(b) - we(a) be(i here we use filter ord copy "adaptive. "Set Copy "set 1.00 "Set 0.00 "Set 0.0	yenithm: y(a) = SM withter(a-t) + (1 eq.) (a) = SM withter(a-t) + (1 eq.) (b) = d(a) - y(a) with = with + wetanise(a-t) + (1 eq.) with = with + wetanise(a-t) + (1 eq.) with = with + wetanise(a-t) + (1 eq.) Comparine Filtraments Set 0.00	Varies transversal using the INGESCOCCO using the INGESCOCCO yearithm: (a) = SLM u(t)*c(n+1) + (1 (b) = d(n) - y(n) (c) = d(n) - y(n) (d) = d(n) - y(n) (e) = d(n) - y(n) (e) = d(n) - y(n) (f) = d(On - Adaptive transversal Using the TNSZ20230 Saline SaM u(k)=k(n-k) i e(n) = G(n) - y(n) u(k) = u(k) + une(n)-k(i there we use filter ord Chen, Chein Chen,	Water transversal	Using the TRESCOCO	10 - Adaptive transversal 10 - Adaptive transversal 10 10 10 10 10 10 10 1	0 - Adaptive transcerses	0 - Adaptive transversal using the INSE20C20 using the INSE20C20 y(a) = SM u(k)*x(n-1) + (1 e(a) = d(a) - y(a) u(k) = u(k) + ue(a)*x(i u(k) = ue(a)*x(i	Ė
y(a) = SIM w(k)*x(n-k) b y(a) = SIM w(k)*x(n-k) b x-0 yar(a) = r*yar(a-1) + (1 x(a) = d(a) - y(a) w(k) = w(b) + (w(a) + y(a) w(k) = w(k) + (w(k) + y(k) + y(a) w(k) = w(k) + (w(k) + y(y(a) = SM withter (raisers at a raing the INESCOCO) y(a) = SM withter(n-t) > (1) war(a) = fewar(a-1) + (1) with = with - with a raing at a r	yenithm: y(a) = SIM u(k)*x(n-k) } y(a) = SIM u(k)*x(n-k) } e(a) = fewar(n-k) + (i) e(b) = fewar(n-k) + (i) e(c) = d(a) - y(a) inter we use filter ord inter we use f	Wing the INESCOCCO Using the INESCOCCO	(geritha: 63 y(a) = SH w(k) ex(n-k) 1 e(n) = G(n) - y(n) war(a) = revar(n-1) + (1 e(n) = d(n) - y(n) w(k) = w(k) + wee(a)ex(n-k) (cey 'adepfile:	yerithm: 63 y(n) = SM w(k) *x(n-k) 1 e(n) = 4(n) - y(n) w(k) = w(k) + wee(n)*x(n-k) 1 e(n) = 4(n) - y(n) w(k) = w(k) + wee(n)*x(n-k) 1 e(n) = 4(n) - y(n) w(k) = w(k) + wee(n)*x(n-k) 1 e(n) = 4(n) - y(n) w(k) = w(k) -	using the TRS20C3O using the TRS20C3O (geritha: 63 (a) (b) (c) (a) (a) (b) (c) (b) (c) (c) (c) (c) (d) (d) (d) (d	0 - Adaptive transversal using the TRSZQCCO (5) (6) = 63 (8) = 51M u(k)*x(n-k) 1 + (1) (9) = 51M u(k)*x(n-k) 1 + (1) (10) = 1/2 (n-k) 1 + (1) (10	0 - Adaptive transversal using the INESZOC30 gorithm: (a) = SIM u(t)*x(n+t) 1 + (1) (c) = SIM u(t)*x(n+t) 1 + (1) (c) = d(n) - y(n) (d) =	(jeritha: 6	i !
	(a) (a) (b) (c)	using the INSESCICAD (gurithat: A y(a) = SM u(k)*x(n-k) } keO var(a) = cavar(n-1) + (1 e(n) = d(n) - y(n) u(k) = u(k) + uve(a)*x(r libers we use filter ord libers we use filter ord libers we use filter ord copy "adaptive THE POWERS ADAPTIVE FILTER set 0.01 set 0.004 set 0.004 set 0.004 set 0.004 set 1.00 set 0.004 set 0.004 set 1.10 set 0.004 set 2.004 set 2.0004	Wing the INSCOCCO (garitha: (a) (a) (b) (c) (c) (c) (c) (c) (c) (c	Genithm: 63 y(a) = SUM u(k)*x(n-k) 1 e(n) = SUM u(k)*x(n-k) 1 e(n) = f(n) - y(n) u(k) = u(k) + u*x(n-k)*x(i likere we use filter ord Chen, Chein Chen, Ch	yerithai 63 y(n) = SM w(k) > k(n+1) + (1) e(n) = (n) - y(n) war(n) = rewar(n-1) + (1) e(n) = d(n) - y(n) w(k) = w(k) + we(n) > e(n) w(k) = w(k) + we(n) > e(n) w(k) = w(k) + we(n) > e(n) w(k) = w(k) - y(n) Der, Chein	Using the TRESCOCO	10 - Adaptive transversal using the TRSZ0C3O	1	Gooritha: 63 y(a) = SM u(k)*x(n-k) k=0 var(a) = r*ver(a-1) + (1 e(n) = d(n) - y(n) u(k) = u(k) + u*x(a)*x(i likere we use filter ord likere we use filter ord copy "sef 0.01 -sef 0.09 -sef 0.09 -sef 0.09 -sef 0.09 -sef 0.09 -sef 0.09 -sef 1.0 -sef 0.09 -sef 1.0 -sef 0.09 -sef 1.0 -se	9
y(n) = SLM w(k)*x(n-k) 1 (2) y(n) = SLM w(k)*x(n-k) 1 (2) e(n) = d(n) - y(n) w(n) = w(n) - y(n) w(k) = w(k) + we(n)*x(n w(k) = w(k) + we(n)*x	y(a) = SLM with backer's is using the INESCOCOS y(a) = SLM with backer's is easy (a-1) + (1) e(a) = d(a) - y(a) with = with + wee(a)back Chen, Cheir Berry Chen Appriller FILER - Set 0.00 - Set	y(a) = SIM with refusion to anosative trainversal sign with refusion (a) = SIM with refusion (a) = SIM with refusion (a) = (a)	Wing the INESCOCCO Garitha: (a) = SIM with the (n-t.) + (1 + (1 + (1 + (1 + (1 + (1 + (1 + (Wing the TRESCOCO Using the TRESCOCO Using the TRESCOCO As SM w(t) = (t-1) + (1 e(n) = (4n) - y(n) wor(n) = rever(n-1) + (1 e(n) = (4n) - y(n) w(t) = u(t) + ue(n) **(i-1) + (1 e(n) = (4n) - y(n) TRESCOCOCOCOCOCOCOCOCOCOCOCOCOCOCOCOCOCOCO	Water transversal	Using the TRSZOCCO Using the Track Using the Tra	10 - Adaptive transversal using the TRSZ0030	0 - Adaptive transverses	0 - Adaptive transversal using the INSE20C20 using the INSE20C20 using the INSE20C20 y(a) = SM u(k)*x(n-1) + (1 e(a) = 4(a) - y(a) vor(a) = révar(n-1) + (1 e(a) = 4(a) - y(a) vor(b) = u(k) + uve(a)*x(i u(k) = u(k) + uve(a)*x(i u(k) = u(k) + uve(a)*x(i u(k) = u(k) + uve(a)*x(i liber us use filter ord Chen, Cheir Chen, Chen Chen Chen, Chen	•
y(a) = SIM w(k)*x(n-k)) b y(a) = SIM w(k)*x(n-k)) b e(a) = d(a) - y(a) w(b) = w(b) + we(a)*x(in-b) b in (b) = w(b) + we(a) b Chen, Cheir Chen, Chen, Cheir Chen, Chen, Chen, Cheir Chen, C	y(a) = SM w(k)*x(n+1) + (1 e(a) = GM w(k)*x(n+1) + (1 e(b) = d(a) - y(a) war(a) = d(a) - y(a) w(k) = w(k) + we(a)*x(i w(k) = w(k) + we(a)*x(i w(k) = w(k) + we(a)*x(i were we use filter ord Dee, Cheir Cosy "depfir. HILLIE POINTERS NO AND Set 0.00 Set 0.004 Set 0.004 Set 0.004 Set 0.004	yenithm: y(a) = SIM u(k)*x(n-k) + (1) vor(a) = revor(n-1) + (1) e(a) = d(a) - y(a) u(k) = u(k) + uve(a)*x(1 inter we use filter ord Chen, Cheir Chen, Chen, Cheir Chen, Chen, Cheir Chen, Che	No - deaptive trainversal using the INESZOGO geritha: And S SM with re(n-1) + (1 e(n) = G(n) - y(n) wit(n) = wit) + we(n) + y(n) with = with + we(n) + y(n) with = with + we(n) + y(n) Chen, Cheir Chen, Che	Wing the The State of Grant State of	90-1482 (S. 2002) 90-1183 (S. 2002) 90-1184 (S. 2002) 90-1184 (R.) = SLM u(k.) *x(n-k.) 1 (d. 2002) 90-1184 (R.) = W(k.) + une(a.) *x(n-k.) + (d. 2002) 90-1184 (M.) = W(k.) + une(a.) *x(n-k.) + (d. 2002) 90-1184 (M.) = W(k.) + (d. 2002) 90-1184 (M.) + (d. 2002) 90	using the TRS20C3O using the TRS20C3O (geritha: 63 (a) = 63 var(a) = 54b u(k) ex(n-k) + (f) e(n) = d(n) - y(n) u(k) = u(k) + uve(a) ex(n-k) u(k) = u(k) + uve(a) ex(n-k) Chen, Chein Chen, Chen, Chein Chen,	Using the TRSZQCCO using the TRSZQCCO 63 y(a) = SIM u(k)*x(n-k) + (i) k(a) = (i, a) + (i) u(k) = u(k) + uve(a)*x(i) u(k)	y(a) = SM w(k)*x(n+k) 1 (e(a) = SM w(k)*x(n+k) 1 + (1) (e(a) = d(a) - y(a) war(a) = r*vær(a-1) + (1) (e(a) = d(a) - y(a) w(k) = w(k) + we(a)*x(in + in +	(geritha: A y(a) = SH u(x) ex(a+1) + (1) e(a) = G(b) - y(b) war(a) = revar(a-1) + (1) e(b) = d(b) - y(b) w(k) = u(k) + ure(a)ex(4) (cey) 'adep(1r. SECON ADEPTIVE FILTER HIMLIE POINTERS AND ARRA Sect 0.004 Set 0.004	195
y(a) = SLM w(k)*x(n-k)) + (1) e(a) = SLM w(k)*x(n-k)) + (1) e(a) = d(a) - y(a) w(k) = w(k) + w*e(a)*x(i libere we use filter ord Chen, Cheir Chen, Chen, Cheir Chen, Chen	(a) (a) (b) (c)	(en) = (A) y(a) = (B) y(b) = (B) y(a) = (B)	Genitha: 63 9(a) = SH u(t) ex(n-t) 1 PO (a) = G(a) - y(b) u(t) = u(t) + uve(a) Ex(n-t) + (1 e(n) = d(a) - y(b) u(t) = u(t) + uve(a) Ex(n-t) (c) = d(a) - y(b) u(t) = u(t) + uve(a) Ex(n-t) (c) = d(a) - y(b) (c) = copy 100 100 100 100 100 100 100 1	Using the TMSZDOCOO Using the TMSZDOCOO Using the TMSZDOCOO Var(a) = SUM u(k)=k(n-k)) + () e(n) = SUM u(k)=k(n-k)) + () e(n) = d(n) - y(n) u(k) = u(k) + une(a)*e(i Mere we use filter ord Mere we use filter ord THE TER THE	yerithm: 63 y(n) = SM w(k) > k(n+1) + (1) e(n) = (n) - y(n) w(k) = (n) - y(n) w(k) = w(k) + we(n) > e(1) w(k) = w(k) + we(n) > e(1) w(k) = w(k) + we(n) > e(1) e(n) = w(k) + w(k) + w(k) e(n) = w(k) + w(k) e(Using the TRESCOCO	10 - Adaptive transversal using the TRS20C3O 10	10	0 - Adaptive transversal using the TMSZ20C30 S y(a) = SUR u(k)=k(n-k) } (e(n) = SUR u(k)=k(n-k) + (f) w(t) = w(k) + uwe(a)=k(f) w(t) = w(t) - vwe(a)=k(f) War(a) = w(t) - vwe(a)=k(f) War(b) = w(t) - vwe(a)=k(f) War(b) = w(t) - vwe(a)=k(f) War(a) =	
y(n) = SLM w(k)=K(n-k) 1 (n) = SLM w(k)=K(n-k) 1 (n) = SLM w(k)=K(n-k) 1 (n) = (n) = d(n) - y(n) w(k) = w(k) = w(k) + w(k)=k(n) = w(k)	y(a) = SIM w(k)*x(n-k) } (a) y(b) = SIM w(k)*x(n-k) } (b) y(a) = SIM w(k)*x(n-k) } (c) y(a) = r**w*x(a-1) * (f) y(a) = r**x(a-1) *	yenithm: (a) (b) (c) (c) (c) (d) (d) (d) (e) (e) (e) (e) (e	0 Modeltow frankersel (guritha: 3 y(a) = SLM u(k)*c(n+1) + (1 c(a) = d(a) - y(a) u(k) = u(k) + uve(a)*c(i u(k) = u(k) + uve(a)*c(i copy - Sed property - Set col - S	0 - Adaptive transversal using the TNSZ20230 S3 y(a) = SM u(k)*x(n-k) 1 e(a) = d(a) - y(a) u(k) = u(k) + uve(a)*x(i there we use filter ord there we use filter ord Chen, Chein Chen,	Wing the TRSZOC30	Using the TRSZOCZO Using the TRSZOCZO	10 - Adaptive transversal using the TRSZ0030	0 - Adaptive transversal using the INESZOCOO gerithm: (3) y(n) = SIM u(t)*E(n-t) } (t) +	10 Adaptive transversal using the TMSZ20C30 10 10 10 10 10 10 10	text
y(a) = SIM w(k)*x(n-k) b y(a) = SIM w(k)*x(n-k) b k=0 var(a) = rewar(a-1) + (1 e(a) = d(a) - y(a) w(k) = w(k) - w(e) b x(a) = d(a) - y(a) w(k) = w(k) - w(e) b x(a) = w(k) - w(e) b x(a) = w(k) - w(e) b x(a) = w(k) - w(e) b x(b) = w(k) - w(e) b x(c) = w(k) - w(e) b x(c) = w(e) - w(e) x(c) = w(e) - w	y(a) = SM w(k)*x(n+1) + (1 w(k) = SM w(k)*x(n+1) + (1 w(k) = w(k) - y(n) = w(k) = w(k) + w(k) + w(k) = w(k) + w(k)	Using the INGESCACO	0 Adda tive trainversal using the TRESZOCCO (guritha:	Wing the The State of State	Wing the TRESCOC30	10 - Adaptive transversal using the TRS20C3O	10 - Adaptive transversal using the TRSZ0C30	0 - Adaptive transversal using the INSEGOCO gorithm: 3 y(n) = SLM u(t)*x(n-t) 1 + (1 + (1 + (1 + (1 + (1 + (1 + (1 +	0 - Adaptive transversal using the TNS20000 using the TNS20000 A y(a) = SM u(t) **(a+1) + (1 e(n) = d(n) - y(n) u(t) = u(t) + u**(a+1)**(1 e(n) = d(n) - y(n) u(t) = u(t) + u**(a+1)**(1 e(n) = d(n) - y(n) u(t) = u(t) + u**(a+1)**(1 e(n) = d(n) - y(n) TNS-10 = to	
y(a) = SLM w(k)+x(n-k)) + (1) e(a) = SLM w(k)+x(n-k)) + (1) e(a) = d(a) - y(a) w(k) = w(k) + w(e (a)+x(i) w((a) SIM u(k) **(a+1) **(b) (b) SIM u(k) **(a+1) **(b) (c) (a) SIM u(k) **(a+1) **(b) (c) (a) (a) (a) (c) (a) (a) (a) (c) (a) (a) (c) (a) (a) (c) (a)	Wing the INSESCICAD	0 deaptive transversal (garitha: (a) (a) (b) (c) (c) (c) (c) (c) (c) (c	Wang the TRESCOCO Using the TRESCOCO S y(a) = SUM u(k) = x(n+1) + (1) e(n) = 4(n) - y(n) u(k) = u(k) + une(a) = x(n) u(k) = u(k) + une(a) = x(n) u(k) = u(k) - une(a) = x(n) Une - x = x(n) Section Admitted	yerithai 63 y(n) = SM w(k) **K(n+1) + (1) e(n) = (4(n) - y(n) wu(k) = w(k) + we(n) **K(n+1) + (1) e(n) = (4(n) - y(n) w(k) = w(k) + we(n) **K(n+1) + (1) e(n) = (4(n) - y(n) **K(n) = (4(n) - y(n) + (1) **K(n) + (1) **K(Using the TRESCOCO		00 - Adaptive transversal using the INESZOC30 garitha: 3 y(a) = SLM ul(x)*x(n-k) } wor(a) = r*var(n-1) + (1 e(a) = d(a) - y(a) u(k) = u(k) + u*e(a)*x(1 e(a) = d(a) - y(a) u(k) = u(k) + u*e(a)*x(1 e(a) = d(a) - y(a) con - y(a) set 0.09 set 0.09 set 0.09 set 0.00 set 0.00 set 0.00	(goritha: 63 y(a) = SM u(k) ex(n-k) 1 kd) y(a) = SM u(k) ex(n-k) 1 kd) w(b) = w(k) + uve(a) ex(n) w(c) = u(k) + uve(a) ex(n) w(c) = u(k) + uve(a) ex(n) w(c) = vec(a) ex(n) were we use filter ord were we use filter ord were we use filter ord were exception	
y(n) = SLM u(k)*x(n-k) 1 y(n) = SLM u(k)*x(n-k) 1 k(n) = (ln) - y(n) u(k) = u(k) - y(n) u(k) = u(k) - y(n) u(k) = u(k) - uve(n)*x(n-k) (Chen, Chein- Chen, Chein- Chen, Chein- STEOM ADDETIVE FILERE STEOM ADDETIVE FILERE	y(a) = SIM w(k)*E(n-k) b y(a) = SIM w(k)*E(n-k) b (b) = (la) - y(a) w(b) = w(b) - y(a) w(c) = w(b) - y(a) w(k) = w(k) + we(a)*E(n-k) Gen, Chein ***********************************	Signature transversal Signature Sign	0 Made the transversal 6.3 y(a) = SM w(k) ex(n-k) } (c) y(a) = SM w(k) ex(n-k) + (1) +	0 - Adaptive transversal using the TNSZ20C20 S3 y(a) = SM u(k) = k(n-k) + (1) e(a) = d(a) - y(a) u(k) = u(k) + une(a) > k(n) u(k) = u(k) + une(a) > k(n) Chen, Chein Chen,	Wing the TRSZOC30	Using the TRESCOCO	10 - Adaptive transversal using the TRSZ0030	0 - Adaptive transversal using the TREZZOCO geritha: 3 y(n) = SIM u(k)*E(n-k) i k-0 yar(n) = r*war(n-1) + (1) e(n) = d(n) - y(n) u(k) = u(k) + u*e(n)*E(1) u(k) = u(k) + u*e(10 Adaptive transversal using the TMS20020 Using the TMS2009 Using the TM	
using the TRSESOCOO y(n) = SLM u(k)*E(n-k) i k=0 var(n) = revar(n-1) + (1) e(n) = d(n) - y(n) u(k) = u(k) + uve(n)*E(n) u(k) = u(k) + uve(n)*E(n) Chen, Cheim ************************************	y(a) = SM with the (n=1) + (1)	y(a) = SLM u(k)*x(n-k) + (1) y(a) = SLM u(k)*x(n-k) + (1) k(b) = b(a) - y(a) war(a) = fewar(a-1) + (1) w(k) = u(k) + uve(a)*x(i w(k) = u(k) + uve(a)*x(i w(k) = u(k) + uve(a)*x(i where we use filter ord Dee, Cheir Cosy "adapfir. ************************************	0. shaptive transversal using the INESZOCZO (guritha: (a) = SM u(k)*x(n-k) + (1 + (n + k) + (n + k) + (1	0 Adaptive transversal using the TNSZOCZO (geritha: A3 y(a) = SH u(t) **(n+1) * (1) e(n) = (n) - y(n) u(t) = u(t) + ue(a) **(n+1) * (1) u(t) = u(t) + ue(a) **(n+1) * (1) u(t) = u(t) + ue(a) **(n+1) * (1) e(n) = (n) - y(n) u(t) = u(t) + ue(a) **(n+1) * (1) e(n) = (n) - y(n) TOTOM ADAPTIVE FILEN ***********************************	V - Adaptive transversal V - Adaptive transversal V - V - V - V - V - V - V - V - V - V	10 - Adaptive transversal using the TRS20C3O	10 - Adaptive transversal 10 - Adaptive transversal 15 15 15 15 15 15 15 1	0 - Adaptive transversal using the INSEZOCOS garithm: (a) = SIM u(t) > C(m+1) + (1)	0 - Adaptive transversal using the TNS20000 using the TNS20000 using the TNS20000 (a) = Sh u(s) **(n+1) **(1 (a) = Sh u(s) **(1) **(1 (b) = Sh u(s) **(1) **(1 (coy **) **(dapfir.**	A 176 P
y(a) = SLM w(k)+x(n+1) > (1) y(a) = SLM w(k)+x(n+1) > (1) e(a) = fan - y(a) w(k) = w(k) + we(a)+x(i w(k) = w(k) + w(k) w((a) (b) (c)	(a) 10 (a) 10 (b) 10 (a) 10 (b) 10 (b) 10 (c) 10 (c	0 Mangative transversal using the INSEZOCCO (geritha: 6.3 y(a) = SM w(k)*x(n-k) } kO (a) = SM w(k)*x(n-k) + (1	(goritha: 63 (goritha: 63 (a) (a) = SM u(k) ex(n-k) 1 (k) (a) = SM u(k) ex(n-k) 1 (c) (a) (a) = (a) - y(n) (a) =	yen:than Sala with PRSZOC30 y(n) = SM w(k) **E(n+1) * (1) war(a) = r*Awe'(a-1) * (1) w(a) = d(n) - y(n) w(b) = w(k) ** wee(a)**E(n+1) * (1) w(b) = w(k) ** wee(a)**E(n+1) * (1) w(c) = wee(a)**E(n+1) * (1) Before we use filter ord Mere we use filter ord Mere we use filter ord Mere we use filter ord West Cheir Before Admerlite Fil.TBR SEFORE A	Using the TRS20C3O	10 - Adaptive transversal using the TRSZ0CCCC 15	00 - Adaptive transversal using the INSESOCOS garithm: 63 y(n) = SLM u(k)*x(n+1) + (1) e(n) = f*var(n-1) + (1) e(n) = d(n) - y(n) u(k) = u(k) + u*e(n)*x(n+1) (Chen, Chein- Chen, Chen, Chein- Chen, Chen, Chen, Chein- Chen,	(goritha: 63 (goritha: 63 (a) = SM u(t)=t(n+1) + (1) (b) = SM u(t)=t(n+1) + (1) (c) = c(n) - v(n) (c) = d(n) - v(n) (c) = d(n) - v(n) (c) = d(n) - v(n) (c) = d(n) - v(n) (d) = d(n) - v(n) (e) = d(n) - v(n) (e) = d(n) - v(n) (f) =	1
y(a) = SM w(k)*x(n-k) 1 (2) y(a) = SM w(k)*x(n-k) 1 (2) xar(a) = rbwar(n-1) + (1) w(k) = w(h) - y(n) w(k) =	y(a) = SIM w(k)*E(n-k)) + (1) y(a) = SIM w(k)*E(n-k)) + (1) e(a) = d(a) - y(a) w(b) = w(b) + we(a)*E(i) w(c) = d(a) - y(b) w(c) = d(a) - y(a) w(c) = d(a) - y(a) w(c) = d(a) - y(a) Chen, Chein ***********************************	(geritha: 63 (s) (s) (s) (s) (s) (s) (s) (s	0 Magative transversal 63 y(a) = SiM u(k) ex(n-k) 1 (a) 7(a) = SiM u(k) ex(n-k) 1 (b) 84 (a) = 4(a) - y(a) 84 (b) = 4(b) - y(a) 84 (c) = 4(a) - y(a) 85 (c) = 4(a) - y(a) 86 (c) = 4(a) - y(a) 87 (c) = 4(a) 88 (c) = 46 88 (c) =	0 - Adaptive transversal using the TNSZ20230 S3 y(a) = SM u(k) *x(n-k) } +(i) e(a) = ((a) - y(a) u(k) = u(k) + uve(a) *x(i) u(k) = u(k) + uve(a) *x(i) Chen, Chein TOTOM ADAPTIVE FILER 1.54 1.54 1.50	Wing the TRSZ0C30	Using the TRESCOCO	10 - Adaptive transversal using the TRS20030	0 - Adaptive transversal using the TREZZOCO gerithat (3) (a) = SIM u(k)*E(n-k) i (b) = (b) = (b) + (i) (c) = d(n) - y(n) u(k) = u(k) + u*e(n)*E(i) u(k) = u(k) + u*e(n)*E(i) Chen, Chein Chen, Chein THERMITTER THERMIT	0 - Adaptive transversal using the TMS220C30 63 y(a) = SM u(k) *x(n-k) } t(d) ar(a) = favor(n-1) + t(d) ar(b) = u(k) + ure(a) *x(d) ar(b) = u(k) + ure(a) *x(d) bere us use filter ord there are use fil	
y(a) = SIM w(k)*E(n-k) b y(a) = SIM w(k)*E(n-k) b e(a) = d(a) - y(a) e(b) = w(b) - w(a) e(b) = w(b) - w(a) Chen, Chein Chen, Chen, Chein Chen,	y(a) = SM with reference (respective) (a) = SM with reference (a) = few (a-1) + (1 e(a) = d(a) - y(a) with = w(b) + we(a) reference we use filter ord with = w(b) + we(a) reference (a) = w(b) + we(a) reference (a) = w(b) + w(b) reference (a) = w(b) + w(b) reference (a) = w(b) + w(b) reference (a) = w(b)	y(a) = SLM u(k)*x(n-k) + (1) y(a) = SLM u(k)*x(n-k) + (1) war(a) = revar(a-1) + (1) e(a) = d(a) - y(a) u(k) = u(k) + uve(a)*x(i u(k) = u(k) + uve(a)*x(i u(k) = u(k) + uve(a)*x(i u(k) = u(k) + uve(a)*x(i libere us use filter ord libere us use filter ord Chen, Cheir Chen, Chen, C	0 - Anaptive francersal using the TMS220C20 using the TMS220C20 yearithat (a) = SM u(k)*x(n-k) + (1 e(n) = frow (n-1) + (1 e(n) = d(n) - y(n) (a) = d(n) - y(n) (b) = u(k) + uve(n)*x(n-k) (b) = u(k) + uve(n)*x(n-k) (coy 204) (coy 304) (co) 304 (co) 404 (co)	Wing the The State of State	Water transcersal Wate	10 - Mdaptive transversal using the TRS20C3O	0 - Adaptive transversal using the TRSZ0C30 63 y(a) = SIM u(k)=k(n-k) + (1) k=0 var(a) = ravar(n-1) + (1) u(k) = u(k) - vue(a) ke(n) u(k) = u(k) + vue(a)	0 - Adaptive transversal using the INSEZOCOO goritha: y(n) = SLM u(t)+x(n-t) 1 + (1 e(n) = d(n) - y(n) u(t) = u(t) + ure(n)+x(n-t) u(t) = u(t) + ure(n)+x(n)+x(n)+x(n)+x(n)+x(n)+x(n)+x(n)+x	0 - Adaptive transversal using the TNS20000 using the TNS20000 y(a) = SM u(s) x(n-t) + ((e(n) = sM u(s) x(n-t) + ((e(n) = d(n) - y(n) u(t) = u(t) + uve(a) x((inter we use filter ord Den, Cheir Cony "depfir. Cony "depfir. SECON ADAPTIVE FILER HIGH SECON ADAPTIVE FILER SECON ADAPTIVE FILER SECON ADAPTIVE FILER SECON ADAPTIVE FILER SECON SE	•
		0 Gaptive transversal ligarithm: A3 y(n) = SM w(k)*x(n-1) + (1 e(n) = d(n) - y(n) w(k) = w(k) + wee(n)*x(f-1) e(n) = d(n) - y(n) w(k) = w(k) + wee(n)*x(f-1) copy *adapfive Den, Chein FEFFORM ADMPTIVE FILTER ***********************************	00 - Admatrice transversal As y(n) = SM u(k)**(n-1) + (1 e(n) = d(n) - y(n) u(k) = u(k) + u**e**(n-1) + (1 e(n) = d(n) - y(n) u(k) = u(k) + u**e**(n-1) + (1 e(n) = d(n) - y(n) u(k) = u(k) + u**e**(n-1) + (1 e(n) = d(n) - y(n) e(n) = d(n) - y(n) EFFORM ADMITITE FILTER 1.54 1.54 1.55 1.5	00 - Adaptive transversal using the TMSZDGZO using the TMSZDGZO (a) = SLM u(k)=k(n+k) } (b) = SLM u(k)=k(n+k) } (c) = SLM u(k)=k(n+k) } (d) = SLM u(k)=k(n+k) } (e) = SLM u(k)=k(n+k) } (e) = SLM u(k)=k(n+k) } (f) =	00 - Adaptive transversal using the TMSZCOCO 63 y(a) = SM w(k)*c(n+1) + (1 e(a) = sM w(k)*c(n+1) + (1 e(a) = sM w(k) + uve(a)*c(n+1) + (1	00 - Adaptive transversal using the TRGSCOCO ligarithm: 63	1	190 - Adaptive transversal using the INESZOC20 190 - 150 50 50 50 50 190 - 150 50 50 50 50 190 - 150 50 50 50 50 190 - 150 50 50 50 190 - 150 50 50 50 190 - 150 50 50 190 - 150 50 190 - 150 50 50 190 - 150 50	00 - Adaptive transversal using the TRSZOCCO using the TRSZOCCO (a) = SLM u(t)=c(n+1) + (1) (a) = SLM u(t) = c(n+1) + (1) (a) = SLM u(t) = c(n+1) + (1) (b) = SLM u(t) = c(n+1) + (1) (c) =	set
(a) = (b) = (b)	y(n) = SIM w(k)**(n-k)) y(n) = SIM w(k)**(n-k)) war(n) = d(n) - y(n) w(k) = u(k) - we(n)**(in-k)) w(k) = u(k) - we(n)**(in-k)) Clen, Chein Clen, Chein Clen, Chein Simple W and will fire Clen, Chein Simple W and will fire Clen, Chein Simple W and will fire Simple W and will fir	Outsing the INESCOCCO Using the INESCOCCO (gorithm: (a) = SIM u(k) **c(n+1) + (1) (b) = SIM u(k) **c(n+1) + (1) (c) = (a) = (b) + (1) (d) = (b) = (b) + (b) + (1) (e) = (b) = (b) + (b)	On - Admitter trainersal using the INESCOCCO using the INESCOCCO Var(a) = SIN u(k)+c(n+1) + (1) e(n) = d(n) - y(n) u(k) = u(k) + ue(a)+c(n+1) + (1) Chen, Chein Chen, Ch	On - Adaptive transversal using the TMSZOCZO (a) (a) = SM u(k)*c(n+1) + (1 c(n) = (1n) - y(n) u(k) = u(k) + ue(n)*c(n there we use filter ord Chen, Cheir Chen,	00 - Adaptive transversal using the TRSZOCZO (goritha: 63 y(n) = SM w(t)rc(n+1) + (1) e(n) = (n) - y(n) w(t) = w(t) + we(n)rc(n+1) + (1) e(n) = (n) - y(n) w(t) = w(t) + we(n)rc(n+1) Chen, Chein Chen, Che	00 - Adaptive transversal using the TRG20C3O ligarithms 63 y(n) = SM u(k)=x(n-1) + (1 e(n) = d(n) - y(n) u(k) = u(k) - une(n)=x(n) u(k) = u(k) + une(n)=x(n) u(k) = u(k) - une(n) u(k) = u(k) - une(n)=x(n) u(k) = u(k) - une(n)	00 - Adaptive transversal using the TRSZ0030 using the TRSZ0030 (garitha: 63 (A) = 63 (A) = 64(A) - 94(A) u(B) = u(B) + uve(a)be(t libere us use filter ord libere ord l	00 - Adaptive transversal using the INSCOCCO lgoritha: (a) = SIM u(k)**c(n-k) i (c) (b) = SIM u(k)**c(n-k) i (c) (c) = d(n) - v(n) (c) = d(n) - v(n) (c) = d(n) - v(n) (c) = u(k) + une(n)**c(i libere we use filter ord Chen, Cheim Copy "adaptive ELLURE PHOTOM ADMPTIVE ELLURE 1.00 1.54 1.00 1.54 1.00	00 - Adaptive transversal using the TMSZOCOO using the TMSZOCOO (a) = SM u(k)*c(n+1) + (1 c(n) = c(n) - v(n) u(k) = u(k) + uec(n)*c(n there us use filter ord Chen, Chein Chen, Che	
19erithm: 63 19erithm: 19erithm: 63 19erithm: 63 19erithm: 63 19erithm: 64 19erithm:	yearing the TRESONCO lgorithm: (3) (a) (b) (a) (a) (b) (a) (b) (b	Using the TMSZXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	Ocabative fransversal using the MESSOCCO ligarithm: SAN (n) = SAN (s) = SAN	Ob - Addative transversal using the TMSZOCCO ligorithm: As y(a) = SH w(t) tx(n-1) + (1 c) And y(a) = Chew (n-1) + (1 c) And y(a) = (1 c) + (1 c) And y(a) = (1 c) + (1 c) And y(a) = (1 c)	100 - Additive transversal using the TMSZCOCOO 100 - 110 - 1	190 - Adaptive transversal using the TRESCOCO 190:1tha: A3	190 - Adaptive transversel	CO - Adaptive transversal using the TRSZOCZO Hyperitha: (a) = SIA u(k)rec(n+1) + (On - Adultive transversal using the TMS20020 As y(n) = SM with ex(n-1) + (1 e(n) = d(n) - y(n) wit(s) = wit(s) + we(a) = x(n) And wit(s) = wit(s) + we(a) = x(n) And wit(s) = wit(s) + we(a) = x(n) And wit(s) = wit(s) + wit(s) = x(n) And	1
	ligarithm: (a) S.M. (L) PAG(n-1) + (1 or (n) = SM u(L) PAG(n-1) + (1 or (n) = d(n) - y(n) and (n) = d(n) - y(n) (a) = d(n) - y(n) (b) = d(n) - y(n) (c) = d(n) - y(n) (d) = d(n) - y(n) (e) = d(n) - y(n) (e) = d(n) - y(n) (e) = d(n) - y(n) (f) = d(n) - y(n) (f) = f(n) - f(n) (f) = f(n) -	Using the Trianversa riller mith mormalized LDS algorithm Using the Trianversa riller mith mormalized LDS algorithm San with San	OS - Admatrice transversal Assay (a) = SLM u(k) ex(n-1) + (1) e(a) = d(a) - y(a) u(k) = u(k) + une(a) ex(n-1) + (1) e(b) = d(a) - y(a) u(k) = u(k) + une(a) ex(n-1) e(c) = d(a) - y(a) e(d) = d(a) - y(Ob - Adaptive transversal using the TMSZDGZO using the TMSZDGZO (A) = SUR u(k)+c(n-k) } (A) y(n) = SUR u(k)+c(n-k) } (A) = SUR u(k)+c(n-k) } (B) = SUR	Using the TRSZOCOO Using the TRSZOCOO Using the TRSZOCOO Varia SA Varia = SAM u(thr(n+1) + (1) varia = rawa(u-1) + (1) e(n) = d(n) - y(n) u(t) = u(t) + une(n)rac(n+1) + (1) e(n) = d(n) - y(n) Cen Cheir C	Using the TRESCOCO Using the TRESCOCO Using the TRESCOCO Using S S S S S S S S S S S S S S S S S S S	using the TRSZOCOO using the TRSZOCOO using the TRSZOCOO using the TRSZOCOO var(a) = SMR u(t)=x(n+1) + ((n+1) + (n+1)	On - Adaptive transversal using the TRESONCEO Ngaritha: SA y(a) = SA y(a)	CO - Adaptive transversal using the TMSZOCCO ligarithm: A3 y(n) = SUR u(t)-tc(n-t) + (1 e(n) = y(n) u(t) = u(t) + uec(n)-tc(n-t) there we use filter ord there we use the we use filter ord there we use the we use ord there we use ord	ž
Using the IMS20020 (3) y(n) = SUM u(k)*x(n-k) k=0,1,2,,63 y(n) = SUM u(k)*x(n-k) k=0,1,2,,63 y(n) = chon - y(n) u(k) = u(k) + u*x(n)*x(n-k)/var(n) k=0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chung Murch, 1989 Chen, Chein-Chung Murch, 1989 POPFORM ADMYTIVE FILTR		Using the TMSZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ	00 - Additive transversal filter with Mormalized LDS algorithm using the TMS200C20 y(n) = SiM wiki*Pekin+1) *e0,1,2,,63 y(n) = SiM wiki*Pekin+2) *e0,1,2,,63 war(n) = rewar(n-1) + (1-r)*Pekin)*Pekin) e(n) = d(n) - y(n) w(k) = wiki + wee(n)*Pe(n-k)*Var(n) *e0,1,2,63 Makere we use filter order = 64 and mu = 0.01. Chen, Chein-Chang March, 1889 **Copy "addifitrint"************************************	00 - Adaptive transversal filter with Normalized LNS algorithm using the TNG220C20 (5) (6) (7) (8) (8) (9) (9) (9) (9) (9) (9	00 - Adaptive transversal filter with Normalized LNS algorithm using the TNS20020 63 y(n) = SIM with Pacin-t, R=0,1,2,,63 k=0 var(a) = ravar(a-1) + (1-r)Pacin)Pacin) e(n) = d(n) - y(n) with = with + ure(n)Pacin-t, R=0,1,2,63 Nhere we use filter order = 64 and mu = 0.01. Chen, Chein-Champ March, 1989 .copy "adaptitr.int" .copy "adaptitr.int" .copy "adaptitr.int" .set 0.01 ; Step size .set 0.01 ; Step size	CO - Adaptive transversal filter with Marmalized LUS algorithm using the TMSZCC20 (a) (b) (c) (c) (c) (c) (d) (e) (e) (e) (e) (e) (e) (e	190 - Adaptive transversel 190 - 180 180 180 190 - 180 180 180 190 - 180 180 180 190 - 180 180 190 - 180 180 190 - 180 180 190 - 180 180 190 - 180 180 190 - 180 180 190 - 180 180 190 - 180 190	(0 - Adaptive transversal using the TREZOCOS) Algorithm: (3) = (3) (4) = (3) (4) = (4(n) - y(n) (6) = (4(n) - y(n) (7) = u(k) = uve(n)rect (Acc., Chein (Chen, Chen (Chen, Chen (Chen, Chen (Chen, Chen (Chen ((SO - Adaptive transversal using the TMS20020 (S) y(n) = SM u(t) = (n) y (n) = SM u(t) = (n) y (n) = (n) - y(n) = (n) - y	1
	using the TMS20C20 ligarithm: (a) = SLM ulk!ex(n-k) k=0,1,2,,63 y(n) = SLM ulk!ex(n-k) k=0,1,2,,63 y(n) = sLM ulk!ex(n-k) (1-r) ex(n) ex(n) e(n) = d(n) - y(n) u(k) = ulk) + ure(n) ex(n-k)/war(n) k=0,1,2,63 limbere we use filter order = 64 and mu = 0.01. Chen, Chein-Chang March, 1989 Septomen Appetite FLIER ***********************************	Using the TMSZXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	On - Adaptive transversal filter mith Normalized LDS algorithm using the TMSZOCZO y(a) = SM u(k)ex(n-1) + (1-1)ex(a)ex(a) e(a) = chew (n-1) + (1-1)ex(a)ex(a) e(b) = d(a) - y(b) u(k) = u(k) + ure(a)ex(n-k)/var(a) k=0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chemp March, 1969 Chen, Chein-Chemp March, 1969 *** *** *** *** *** *** ***	00 - Adaptive transversal filter with Normalized LNS algorithm using the TMS200C20 y(n) = SiN with Dev(n=1) + (1-7) Dev(n) Dev(n) e(n) = d(n) - y(n) u(k) = with + une (n) Dev(n=1) / var(n) Peo(1,2,63) Norman was filter order = 64 and mu = 0.01. Chen, Chein-Champ March, 1989 Chen, Chein-Champ March, 1989 **** **** **** **** **** **** ****	00 - Adaptive transversal filter with Marmalized LNS algorithm using the TMS20020 (3) y(a) = SM with Perior. (a) = SM with Perior. (b) = with - wire(a) Perior. (c) = d(a) - y(a) with = with + wire(a) Perior. (c) = d(a) - y(a) (d) = d(a) - y(a) (e) = d(a) - y(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) - f(a) - f(a) - f(a) - f(a) (f) = f(a) -	CO - Adaptive transversal filter with Marmalized LNS algorithm using the TMS20020 ilgorithm: A3 y(n) = SIM witi*Per(n+t) PeO,1,2,,63 k=0 var(n) = révar(n-t) + (1-r)*Per(n)*Ex(n) e(n) = d(n) - y(n) with = witi + use(n)*Ex(n-k)/var(n) PeO,1,2,63 Mhere we use filter order = 64 and mu = 0.01. Chen, Chein-Chung March, 1989 .copy "adapfilt.int"	using the TRS20020 using the TRS20020 ligorithm: 63 var(a) = SIA u(k)*e(n+1) t e(n) = (n) (n) var(a) = d(n) - e(n) u(k) = u(k) + ue(a)*e(i libere we use filter ord Then, Desire There we use filter ord Then Admitted Then Admitted The TRS2002000000000000000000000000000000000	CO - Adaptive transversal using the TRSZOCZO Hyperitha: (A) = SIA ultirec(n-t) 1 (1) + (OCO- Adultive transversal using the TMS2OCOO ligorithm: AS y(n) = SiM with ex(n-1) + (1 e(n) = d(n) - y(n) wit(s) = wit(s) + wee(n) = x(n) wit(s) = wit(s) + wee(n) = x(n) wit(s) = wit(s) + wee(n) wit(s) = wit(s	ž.
		Section Comparison Compar	So	CO - Adaptive transversal filter with Normalized LNS algorithm using the TNS220C20 (A) SM with Perform to TNS220C20 (A) Van SM with Perform to TNS220C20 (A) SM with Perform to TNS220C20 (A) A	00 - Adaptive transversal filter with Marsalized LNS algorithm using the TMS20020 43 y(n) = SM with Pecin+1 > Pa0,1,2,,63 y(n) = SM with Pecin+2 > Pa0,1,2,,63 war(n) = revar(n-1) + (1-r)Pecin)Pecin) e(n) = d(n) - y(n) with = with + when in Pecin > Pand mu = 0.01. Chen, Cheim-Chang March, 1989 **********************************	CO - Adaptive transversal filter with Normalized LDS algorithm using the TMSZCC20 (S) (S) (S) (A) (A) (A) (A) (A)	using the TRSZOCOO ligorithm: As y(m = 12 M u(t) = (12 M	19	CO - Adaptive transversal using the TMS20COO using the TMS20COO AS y(n) = SLM u(k)*c(n+1) + (1) c(n) = chou (n-1) + (1) u(k) = u(k) + uve(n)*c(n) u(k) = chou (n) u(k) = chou (1
using the IMSZ20020 Using the IMSZ20020 Sa	gorithm: 43 V(n) = SM with PK(n-t.) k=0,1,2,,63 y(n) = SM with PK(n-t.) k=0,1,2,,63 y(n) = G(n) - y(n) war(n) = d(n) - y(n) war(n) = with + with (n) + (1-r) PK(n) PK(n) war(n) = d(n) - y(n) war(n) = with + with (n) + (1-r) PK(n) PK(n) war(n) = d(n) - y(n) war(n) = with + with (n) + (1-r) PK(n) PK(n) war(n) = with + with (n) + (1-r) PK(n) PK(n) Chen, Chein-Chung Murch, 1989 war(n) = with PK(n) PK(n	Using the TMSZCCZO Using the TMSZCCZO Igorithm: (a) = SIM u(k) **X(n-k) k=0,1,2,,63 y(n) = SIM u(k) **X(n-k) (1-r) **X(n) **X(n) k=0 var(n) = r*var(n-1) * (1-r) **X(n) **X(n) e(n) = d(n) - y(n) u(k) = u(k) - y(n) likere we use filter order = 64 and mu = 0.01. Chen, Chein-Chung March, 1989 .copy "adapfitr.in" PEPFORM adaPTIVE FILTS ***THE COPY TRANSPORTIVE FILTS *	00 - Adaptive transversal filter mith Mormalized LDS algorithm using the TMS200C20 (garithm: 63 y(n) = SUM with Per(n+1) Pe0,1,2,,63 y(n) = SUM with Per(n+1) Pe0,1,2,,63 wir(n) = rewar(n-1) + (1-+)Pex(n)Pex(n) e(n) = d(n) - y(n) with = with + wee(n)Pex(n-k)/war(n) Pe0,1,2,63 Maker we use filter order = 64 and mu = 0.01. Chen, Chein-Chwang March, 1999	00 - Adaptive transversal filter with Normalized LNS algorithm using the TNS20020 (3) (4) (5) (6) (6) (7) (8) (8) (9) (9) (9) (9) (9) (9	00 - Adaptive transversal filter with Marmalized LNS algorithm using the TNS20020 63 y(n) = SM withex(n-t) + (1-7) ex(n) ex(n) exq var(n) = revar(n-1) + (1-7) ex(n) ex(n) ext(n) = d(n) - y(n) with = with + use(n) ex(n-t) / var(n) exq var(n) Chen, Chein-Chung Murch, 1989 POSFORM ADAPTIVE FILTR	CO - Adaptive transversal filter with Marmalized LNS algorithm using the TMSZCC20 (3) (4) (5) (6) (7) (8) (8) (9) (9) (9) (9) (9) (9	On - Adaptive transversal using the TRSZOCCO Nigorithai SA y(n) = SiN u(k) to(n-k) 1 var(n) = revar(n-1) + (f var(n) = d(n) - y(n) u(k) = u(k) + ute(n)te(n)te Dae, Dezin Chen, Chen Chen Chen, Chen Chen	CO - Adaptive transversal using the TRSZOC20 Algorithm: Algorith	CO - Adaptive transversal using the TMS220C20 Salay (n) = SAN u(t) = C(n+1) + (1) e(n) = d(n) - y(n) u(t) = u(t) + ue(n) > c(n) U(t) = u(t) + ue(n) > c(n) Chen, Chein Chen,	. set
Using the TMS20020 43 y(a) = SIM ultiPx(in-t) k=0,1,2,,63 k=0 var(a) = r+var(a-1) + (1-r)Px(in)Px(a) e(n) = d(n) - y(n) u(t) = ult) + ure(a)Px(in-t)/var(a) k=0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chang Murch, 1999	using the TMS20C20 ligarithm: (a) SIM u(k)ex(n-k) k=0,1,2,,63 y(n) = SIM u(k)ex(n-k) k=0,1,2,,63 y(n) = siM u(k)ex(n-k) (1-r)ex(n)ex(n) e(n) = d(n) - y(n) u(k) = u(k) + ure(n)ex(n-k)/var(n) k=0,1,2,63 likere use use filter order = 64 and am = 0.01. Chen, Chein-Champ flurch, 1989 .copy "adapfilt.int"	Using the Trianversa riller mith mormalized LDs algorithm Using the Trianversa riller mith mormalized LDs algorithm: S y(n) = SLM with Pex(n-t) k=0,1,2,,63 y(n) = SLM with Pex(n-t) k=0,1,2,,63 war(n) = rewar(n-t) + (1-r)Pex(n)Pex(n) e(n) = d(n) - y(n) with = with + wee (alPex(n-t)/war(n) k=0,1,2,63 likere we use filter order = 64 and mu = 0,01. Chen, Chein-Chung Phrch, 1989 .copy "adaptifr.int"	SO - Additive transversal filter with Mormalized LDS algorithm using the TMSZ20C20 y(n) = SUM ukt/scn+1 k=0,1,2,,63 y(n) = SUM ukt/scn+1 k=0,1,2,,63 var(n) = revar(n-1) + (1-r)sx(n)sx(n) e(n) = d(n) - y(n) u(k) = u(k) + use (n)sx(n+1)var(n) k=0,1,2,63 labere use use filter order = 64 and mu = 0,01. Chen, Chein-Chang Murch, 1989 .copy "addgift.int"	CO - Adaptive transversal filter with Normalized LNS algorithm using the TMSZ20C20 y(n) = SiM with Par(n+1) k=0,1,2,,63 y(n) = SiM with Par(n+2) k=0,1,2,,63 var(n) = revar(n-1) + (1-7)Par(n)Par(n) e(n) = d(n) - y(n) with = with + unetalPar(n+1)/var(n) k=0,1,2,63 Where we use filter order = 64 and mu = 0,01. Chen, Chein-Chung Murch, 1989 .copy "adaptifr.int"	00 - Adaptive transversal filter with Marmalized LNS algorithm using the TMS20020 63 y(n) = SuM with Pacin L + 0.1,2,63 y(n) = suM with Pacin L + (1-+) Pacin Pacin e(n) = d(n) - y(n) with = with + wretel Pacin L + (1-+) Pacin Pacin liberte we use filter order = 64 and mm = 0.01. Chen, Chein-Chang Murch, 1989 copy "adaptifrint"	CO - Adaptive transversal filter with Mormalized LNS algorithm using the TMS20C20 (1) (2) (3) (4) (4) (5) (6) (6) (6) (7) (7) (8) (9) (9) (9) (9) (9) (9) (9) (9) (9) (9	CO - Adaptive transversal filter with Normalized LMS algorithm using the TMS200200	CO - Adaptive transversal filter with Normalized LNS algorithm using the TMS200230 Migarithms: Sy(n) = SiM wilklev(n-t) k=0,1,2,,63 k=0 var(n) = r=var(n-t) + (1-†)**(n)**(n) e(n) = d(n) - y(n) with = with + whe(n)**(n+t)/var(n) k=0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Cheim-Chang March, 1989 **Copy "adaptitr.int" **Copy "adaptitr.int"	CO - Adaptive transversal filter with Normalized LNS algorithm using the TNSS20C20 Algorithm: CA Var(a) = SDN w(k) **C(n-k) k=0,1,2,,63 k=0 Var(a) = révar(a-1) + (1-r)**C(a)**C(a) k=0 Var(b) = w(k) + w**c(a)**C(a-k)**C(a) W(k) = w(k) + w**c(a)**C(a-k)**C(a) Where we use filter order = 64 and mu = 0,01. Chen, Cheim-Chang Murch, 1989 COPY **adapiltr.in** COPY **adapiltr.in** COPY **adapiltr.in**	
using the TMS20020 (3) (4) = SIM u(k)ex(n-t) k=0,1,2,,63 (a) = revar(n-t) + (1-r)ex(n)ex(n) (a) = d(n) - y(n) (b) = u(k) + ure(n)ex(n-k)/var(n) k=0,1,2,63 (b) = u(k) + ure(n)ex(n-k)/var(n) k=0,1,2,63 (b) = u(k) + ure(n)ex(n-k)/var(n) k=0,1,2,63 (c) = u(k) + ure(n)ex(n)ex(n)ex(n)ex(n)ex(n)ex(n)ex(n)e	garithm: Salan FMSZ8020 garithm: Salan FMSZ80200 y(n) = Silan FMSZ80200 var(n) = rebur(n-1) + (1-r) Px(n) Px(n) e(n) = d(n) - y(n) u(k) = u(k) + uve(n) Px(n-k) /var(n) Px(1,2,63) where we use filter order = 64 and wm = 0.01. Chen. Chein-Chung March, 1989 copy	Using the Trianversal riller mith mormalized LDs algorithm (3) (3) (4) Sign w(t) PK(20020 (4) Van) = Sign w(t) PK(n+t) Pe0,1,2,,63 (6) = d(n) - y(n) (7) = d(n) - y(n) (8) = w(t) + wee(n) PK(n+t) Var(n) Pe0,1,2,63 (8) = w(t) + wee(n) PK(n+t) Var(n) Pe0,1,2,63 (8) = w(t) + wee(n) PK(n+t) Var(n) Pe0,1,2,63 (9) = w(t) + wee(n) PK(n+t) Var(n) Pe0,1,2,63 (9) = w(t) + w(t) PK(n+t) PK(n) PE0,1,2,63 (1) = w(t) + w(t) PK(n)	00 - Additive transversal filter mith Mormalized LDS algorithm using the TMS200C20 (3) y(n) = Sulf with Pertn+T) = 0,1,2,,63 y(n) = Sulf with Pertn+T) = 0,1,2,,63 var(n) = rewar(n-1) + (1-r) Pertn) Pertn) e(n) = d(n) - y(n) with = with + wherin Pertn+T) with = with + wherin Pertn+T) Chen, Cheim-Chwang March, 1989 .copy "addefftr.int" .copy "addefftr.int"	CO - Adaptive transversal filter with Normalized LNS algorithm using the TNS220C20 (3) (4) (5) (6) (7) (8) (8) (9) (8) (9) (9) (9) (9	00 - Adaptive transversal filter with Mormalized LNS algorithm using the TNS20020 63 y(n) = SIM with Px(n-1) Px0,1,2,,63 y(n) = SIM with Px(n-1) + (1-r)Px(n)Px(n) e(n) = f(n) - y(n) with = with + whe(n)Px(n-1) frac(n) Px0,1,2,63 Makere we use filter order = 64 and mu = 0.01. Chen, Cheim-Chang Murch, 1989 .copy "adapfitr.int"	CO - Adaptive transversal filter with Normalized LUS algorithm using the TMSZCC20 (S) (A) (A) (A) (A) (A) (A) (A)	CO - Adaptive transversal filter with Normalized LNS algorithm using the TMSZCC20 Algorithm: Ca Algorithm: Chen, Chein-Champ March, 1999 **Adafitr.int** **Copy **Adafit	CO - Adaptive transversal filter with Normalized LNS algorithm using the TMSZCC20 Algorithm: Algorit	(SO - Adaptive transversal filter with Mormalized LMS algorithm using the TMS200200 (S) (A) = SAM with TMS200200 (A) = SAM with TMS2002000000000000000000000000000000000	*****
using the IMSZ20C20 (3) y(n) = SM wiki>x(n-k) k=0,1,2,,63 y(n) = SM wiki>x(n-k) k=0,1,2,,63 y(n) = GM wiki>x(n-k) (1-r) x(n) x(n) e(n) = d(n) - y(n) wiki = wiki + use (n) x(n-k)/wikin k=0,1,2,63 likere we use filter order = 64 and mu = 0.01. Chen, Cheim-Chung March, 1989 .copy "adapfite,int"		Using the Trianversal riller mith moralized LDs algorithm using the Trianversal riller mith moralized LDs algorithm: (a) SIM ulklex(n=1) + (1-7) Ex(n) Ex(n) (a) = 4(n) - y(n) u(k) = u(k) + uex(n) Ex(n-k)/var(n) Ex0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Cheim Chung Murch, 1989 .copy "adapfitr.int"	On - Adaptive transversal filter mith Normalized LDS algorithm using the TMSZOCZO y(n) = SIM u(k)ex(n-1) + (1-7)ex(n)ex(n) war(n) = r+war(n-1) + (1-7)ex(n)ex(n) w(n) = d(n) - y(n) u(k) = u(k) + ure(n)ex(n-k)/war(n) = 0.01. Chen, Cheim Chwag March, 1989 .copy "adapfiltrint"	On - Adaptive transversal filter with Normalized LNS algorithm using the TNS200C20 y(n) = SM y(n) = SM w(t) or(n-t) \ (1-1) fx(n) \ bx(n) e(n) = r twar (n-1) + (1-1) fx(n) bx(n) e(n) = d(n) - y(n) w(t) = w(t) + wre(n) bx(n-t) / wr(n) \ bx(n) w(t) = w(t) + wre(n) bx(n-t) / wr(n) \ bx(n) Chen, Chein-Chang March, 1989 .copy "adaptite: int" .copy "adaptite: int"	00 - Adaptive transversal filter with Mormalized LNS algorithm using the TNS20020 63 y(n) = SUM withex(n-t) k-0,1,2,,63 k=0 var(n) = revar(n-t) + (1-r)ex(n)ex(n) e(n) = d(n) - y(n) w(k) = w(k) + une(n)ex(n-k)/var(n) k=0,1,2,63 More we use filter order = 64 and mu = 0.01. Chen, Chein-Chang Murch, 1989 **********************************	CO - Adaptive transversal filter with Marmalized LNS algorithm using the TMSZOC20 ilgorithm: A3 y(n) = SLM witi*Px(n+t) k=0,1,2,,63 k=0 var(n) = révar(n-t) + (1-r)*Px(n)*x(n) e(n) = d(n) - y(n) with = witi + u*e(n)*x(n+t)/var(n) k=0,1,2,63 Makere we use filter order = 64 and mu = 0,01. Chen, Chein-Chung March, 1989 .copy "adapfite,int" .copy "adapfite,int"	CO - Adaptive transversal filter with Normalized LMS algorithm using the TMS2COC20 Algorithm: 63 y(n) = SAM with Pex(n+k) k=0,1,2,,63 k=0 var(n) = revar(n-l) + (1-r)Pex(n)Pex(n) e(n) = d(n) - y(n) with = with + une(n)Pex(n-k)/var(n) k=0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chung March, 1989 .copy "adapfite,int"	CO - Adaptive transversal filter with Narmalized LNS algorithm using the TMS200200 Algorithm: y(n) = SM with Px(n-t) l=0,1,2,,63 y(n) = SM with Px(n-t) l=0,1,2,,63 k=0 var(n) = r-war(n-1) + (1-r)Px(n)Px(n) e(n) = d(n) - y(n) with = with + wee(n)Px(n-t)/war(n) l=0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chung March, 1989 **********************************	OCO - Adaptive transversal filter with Hormalized LNS algorithm Using the TNSS20C20 Algorithm: Algorithm: Angle (a) = SiM with Par(n-t) k=0,1,2,,63 Angle (a) = c+war(n-1) + (1-r)Par(n)Par(n) E(n) = d(n) - y(n) With = with + uhe(n)Par(n+t)/war(n) k=0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chung March, 1989 Chen, Chein-Chung March, 1989	
using the TMS20020 43 y(n) = SIM u(k)ex(n-k) k=0,1,2,,63 y(n) = SIM u(k)ex(n-k) k=0,1,2,,63 k=0 var(n) = r+var(n-1) + (1-r)ex(n)ex(n) e(n) = d(n) - y(n) u(k) = u(k) + uve(n)ex(n-k)/var(n) k=0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chung Murch, 1989 .copy "adapfittinin"		Using the Trianversa riller mith mormalized LDs algorithm using the Trianversa riller mith mormalized LDs algorithm sold var(a) = SM with !nc(n+1) !-0,1,2,,63 var(a) = rowar(n-1) + (1-7) !nc(n) !nc(n) e(n) = d(n) - y(n) u(t) = with + unc(n) !nc(n-t) /var(a) !nc0,1,2,63 Where we use filter order = 64 and mm = 0.01. Chen, Cheim-Chang March, 1989 .cop, "adaptir.int"	00 - Additive transversal filter mith Mormalized LDS algorithm using the TMSZ20C20 (3) y(n) = SiN u(t) = 0.1,2,,63 y(n) = SiN u(t) = 0.1,2,,63 var(n) = rbvar(n-1) + (1-r) = 0.1,2,,63 war(n) = v(n) = v(n) u(t) = u(t) + ure(n) = v(n) = 0.1,2,,63 Where we use filter order = 64 and mm = 0.01. Chen, Chein-Chang March, 1989 .cop, "additiving March, 1989 .cop, "additiving March, 1989	CO - Adaptive transversal filter with Normalized LPS algorithm using the TMS220C20 (G) SM with PK(n+1) k=0,1,2,,63 y(n) = SM with PK(n+1) k=0,1,2,,63 y(n) = rawar(n-1) + (1-r) bK(n) bK(n) (e(n) = d(n) - y(n) with = with + waw(n) bK(n-k)/war(n) k=0,1,2,63 Where we use filter order = 64 and wn = 0,01. Chen, Chein-Chang March, 1989 ***Construction of the second se	00 - Adaptive transversal filter with Marmalized LNS algorithm using the TMS20020 1.2	CO - Adaptive transversal filter with Normalized LNS algorithm using the TMSZCC20 ligarithm: 63 y(n) = SIM wiki=x(n-t) k=0,1,2,,63 k=0 var(n) = r=var(n-t) + (1-r)=x(n)=x(n) e(n) = d(n) - y(n) w(k) = wiki + u=e(n)=x(n)=x(n)-x(n) liMere we use filter order = 64 and mu = 0.01. Chen, Cheim-Chung March, 1989 opy = "adaptive international i	CO - Adaptive transversal filter with Normalized LMS algorithm using the TMSZ0C20 Algorithm: Algorit	CO - Adaptive transversal filter with Normalized LNS algorithm using the TMSZGC20 Algorithm: C3 y(n) = SLM wur(n) = revor(n-t) + (1-1)*x(n)*x(n) k-0 var(n) = revor(n-1) + (1-1)*x(n)*x(n) e(n) = d(n) - y(n) w(t) = w(t) + w*e(n)*x(n-t)/var(n) k=0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chung Nurch, 1989 .copy = *adaptive international content international co	(SO - Adaptive transversal filter with Hormalized LPS algorithm Using the TMSZGCZG (SO - SOM with Part (n-1) + (1-1) Part (n) P	7000
using the IMSZ20020 (3) (4) (5) (6) (7) (8) (9) (9) (9) (9) (9) (9) (9		Os - Federice transversa riller mith Mormalized LDs algorithm 63 y(n) = SiM with Per(n-t) PeO,1,2,,63 y(n, = SiM with Per(n-t) PeO,1,2,,63 wur(n) = rever(n-1) + (1-r)Pex(n)Pex(n) e(n) = d(n) - y(n) with = with + wite(n)Pex(n-t)/var(n) PeO,1,2,63 Maker we use filter order = 64 and mu = 0.01. Chen, Chein-Chung Murch, 1989 **********************************	SO - Additive transversal filter mith Normalized LPS algorithm using the TMSZGCCO (3) y(n) = Suff with Fe(n+t) k=0,1,2,,63 y(n, = Suff with Fe(n+t) k=0,1,2,,63 yur(n) = rever(n-1) + (1-r) Px(n) Px(n) w(k) = w(k) + wPe(n) Px(n-k)/var(n) k=0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chomp Murch, 1999 .copy "addyfith.int"	CO - Adaptive transversal filter with Normalized LNS algorithm using the TNS220C20 SS y(n) = SM with Per(n-1) + (1-r) Per(n) Per(n) var(n) = revar(n-1) + (1-r) Per(n) Per(n) e(n) = d(n) - y(n) with = with + ure(n) Per(n-1) yar(n) = 0.1,2,63 Nhere we use filter order = 64 and mu = 0.01. Chen, Chein-Champ Nurch, 1989 .copy "adapfilt.int"	00 - Adaptive transversal filter with Marmalized LNS algorithm using the TNS20020 63 y(n) = SM withex(n-t) + k-0,1,2,,63 k-0 var(n) = revar(n-1) + (1-r)Px(n)Px(n) e(n) = d(n) - y(n) with = with + une(n)Px(n-t)/var(n) Px-0,1,2,63 Mhere we use filter order = 64 and mu = 0.01. Chen, Chein-Chung Murch, 1999 *********************************	CO - Adaptive transversal filter with Marmalized LUS algorithm using the TMSZCC20 (3) (4) (5) (6) (7) (8) (8) (9) (9) (9) (9) (9) (9	CO - Adaptive transversal filter with Normalized LNS algorithm using the TMS2CC20 Algorithm: Algorit	CO - Adaptive transversal filter with Narmalized LNS algorithm using the TMS200200 Algorithm: Algori	(SO - Adaptive transversal filter with Normalized LNS algorithm using the TNSZGCZO AS y(n) = SMR wiklex(n+k) k=0,1,2,,63 y(n) = SMR wiklex(n+k) k=0,1,2,,63 war(n) = revar(n+1) + (1-r)ex(n)ex(n) e(n) = d(n) - y(n) wikl = wikl + whe(n)ex(n+k)/war(n) k=0,1,2,63 Wikhere we use filter order = 64 and mu = 0.01. Chen, Chein-Chaung March, 1989 **********************************	***************************************
Using the TMS20020 43 y(a) = SIM u(k)*x(n-k) k=0,1,2,,63 y(a) = SIM u(k)*x(n-k) k=0,1,2,,63 y(a) = f=0 + y(a) e(a) = d(a) - y(a) u(k) = u(k) + u**e(a)*x(n-k)/v*a*(a) k=0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chang Murch, 1999	using the TMS20C20 ligarithm: (a) SIM u(k)ex(n-k) k=0,1,2,,63 y(n) SIM u(k)ex(n-k) k=0,1,2,,63 y(n) = n-war(n-1) + (1-r)ex(n)ex(n) e(n) = d(n) - y(n) u(k) = u(k) + ure(n)ex(n-k)/war(n) k=0,1,2,63 likere use use filter order = 64 and mu = 0.01. Chen, Chein-Champ March, 1989	Using the Trianversal riller mith mormalized LDs algorithm Using the Trianversal riller mith mormalized LDs algorithm (gerithm: k=0 var(m) = rever(m-1) + (1-r)kx(m)kx(m) k=0 var(m) = rever(m-1) + (1-r)kx(m)kx(m) e(n) = d(n) - y(m) u(k) = u(k) + use (m)kx(m-k)/var(m) k=0,1,2,63 likere we use filter order = 64 and mm = 0,01. Chen, Chein-Champ flurch, 1989	SO - Additive transversal filter mith Normalized LPS algorithm using the TMSZGCZO y(n) = SUN u(k)ex(n-k) k=0,1,2,,63 y(n) = SUN u(k)ex(n-k) k=0,1,2,,63 var(n) = revar(n-1) + (1-r)ex(n)ex(n) e(n) = d(n) - y(n) u(k) = u(k) + uee(n)ex(n-k)/var(n) k=0,1,2,63 lahere use use filter order = 64 and mu = 0,01. Chen, Chein-Chang March, 1989	CO - Adaptive transversal filter with Normalized LNS algorithm using the TNS200C20 y(n) = SM withouth + N=0,1,2,,63 y(n) = SM withouth + N=0,1,2,,63 var(a) = revar(a-1) + (1-r)Ex(n)Ex(n) e(n) = d(n) - y(n) with = with + unetalEx(a-t)/var(a) k=0,1,2,63 Where we use filter order = 64 and mu = 0,01. Chen, Chein-Chang Murch, 1989	00 - Adaptive transversal filter with Marsalized LNS algorithm using the TMS20020 63 y(a) = SM with Per(n-t) > A0,1,2,,63 var(a) = rewar(a-1) + (1-r) Per(a) Per(a) e(a) = d(a) - y(a) with = with + wire(a) Per(a-t) / var(a) > Pe0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chang Murch, 1989	CO - Adaptive transversal filter with Mormalized LNS algorithm using the TMS20020 (1) (3) (4) (5) (6) (7) (8) (9) (9) (9) (9) (9) (9) (1) (1	CO - Adaptive transversal filter with Normalized LMS algorithm using the TMSZCCZO Migorithm: 63 y(n) = SAM with Pextneth NeO, 1, 2,, 63 k=O var(n) = revar(n=1) + (1-r)Pex(n)Pex(n) e(n) = d(n) - y(n) with = with + use(n)Pex(n-k)/var(n) NeO, 1, 2,63 Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chung March, 1989	CO - Adaptive transversal filter with Normalized LNS algorithm using the TMS200230 Nigorithm: Sy(n) = SiM wilk!bec(n-k) k=0,1,2,,63 k=0 var(n) = r=var(n-1) + (1-r)bec(n)bec(n) e(n) = d(n) - y(n) wilk) = wilk) + whe(n)bec(n-k)/var(n) k=0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Cheim-Chang March, 1989	CO - Adaptive transversal filter with Normalized LNS algorithm using the TMSZGC20 (A) Signithm: (A) Signithm: (A) = Signithm: (A) = Signithm (A) + (1-1)	
using the TMS20020 43 y(n) = SUM u(k)ex(n-k) k=0,1,2,,63 y(n) = SUM u(k)ex(n-k) k=0,1,2,,63 y(n) = revar(n-1) + (1-r)ex(n)ex(n) e(n) = d(n) - y(n) u(k) = u(k) + une(n)ex(n-k)/var(n) k=0,1,2,63 Where we use filter order = 64 and mu = 0,01. Chen, Chein-Chung March, 1989	Using the TMSZ0C20 Igaritha: S SM w(k) = SM w(k) = V(n-k) V=0,1,2,,63 Var(a) = rbwar(a-1) + (1-r) = V(n)	Using the Trianversal riller mith mormalized LDs algorithm using the Trianversal riller mith mormalized LDs algorithm as y(n) = SiM with !m(n-1) !mo(1,1,2,,63 y(n) = T-bugr(n-1) !mo(1,1,2,,63 y(n) = d(n) - y(n) wur(n) = rebur(n-1) !mo(1,1,2,,63 wur(n) = with + une(n) !mo(n-1,1,2,,63 withere we use filter order = 64 and wm = 0.01. Chen, Cheim-Cheung Murch, 1989	00 - Additive transversal filter mith Normalized LPS algorithm using the TMSZ00C20 garithm: SA y(n) = SiN u(t) Px(n+t) k=0,1,2,,63 var(n) = rebur(n-1) + (1-r) Px(n) Px(n) e(n) = d(n) - y(n) u(t) = u(t) + ure(n) Px(n-t) /var(n) k=0,1,2,63 Where we use filter order = 64 and mm = 0,01. Chen, Cheim-Champ March, 1989	CO - Adaptive transversal filter with Normalized LPS algorithm using the TMS220C20 (G) (A) = SUM with Partner (A-1,2,,63 y(n) = SUM with Partner (A-1) + (1-+)Partner (A) var(n) = rewar(n-1) + (1-+)Partner (A) with = with + user(n)Partner (A) With + user(n)Part	00 - Adaptive transversal filter with Marmalized LNS algorithm using the TNS20020 43 y(n) = SM with PK(n-1) Pa0,1,2,,63 y(n) = SM with PK(n-1) + (1-r)PK(n)PK(n) var(n) = revar(n-1) + (1-r)PK(n)PK(n) e(n) = d(n) - y(n) with = with + wee(n)PK(n-1)/var(n) Pa0,1,2,63 Mhere we use filter order = 64 and wn = 0.01. Chen, Cheim-Champ March, 1989	CO - Adaptive transversal filter with Normalized LNS algorithm using the TMSZCC20 ligarithm: 63 y(n) = SIM wiklex(n-t) k=0,1,2,,63 k=0 var(a) = revar(n-t) + (1-r)tx(n)tx(n) e(n) = d(n) - y(n) wik) = wikl + upe(n)tx(n-k)/var(n) k=0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Cheim-Chang March, 1989	CO - Adaptive transversal filter with Marmalized LMS algorithm using the TMS20020 Algorithm: Algorit	CO - Adaptive transversal filter with Normalized LNS algorithm using the TMSZGC20 Algorithm: As y(n) = SLM wur(n) = Tevar(n-t) + (1-r) Px(n) Px(n) k=0 var(n) = revar(n-1) + (1-r) Px(n) Px(n) k=0 var(n) = revar(n-1) + (1-r) Px(n) Px(n) (n) = d(n) - y(n) w(t) = w(t) + une(n) Px(n-t) /var(n) k=0,1,2,63 Where we use filter order = 64 and mn = 0.01. Chen, Chein-Chung Murch, 1989	(SO - Adaptive transversal filter with Normalized LNS algorithm using the TMSZCC20 (S) (A) = SAM with Per(n+t) k=0,1,2,,k3 (An) = SAM with Per(n+t) k=0,1,2,,k3 (An) = shwar(n-1) + (1-t)Per(n)Per(n) (An) = u(t) + une(n)Per(n+t) k=0,1,2,k3	900
using the TMS200C30 (3) (4) (5) (6) (7) (8) (9) (9) (9) (9) (9) (9) (9	Using the TMSZGC20 Jgerithm: 63 y(n) = SM wikibx(n-t) k=0,1,2,,63 k=0 var(n) = rawar(n-1) + (1-r)bx(n)bx(n) e(n) = d(n) - y(n) wik) = wikib + wee(n)bx(n-k)/var(n) k=0,1,2,63 Wikhere we use filter order = 64 and mu = 0,01. Chen, Chein-Chung Murch, 1989	Os - federive transversa rilter mith mormalized LDs algorithm 63 y(n) = SM withPer(n-t) k=0,1,2,,63 y(n/s) = ravar(n-t) + (1-r)Per(n)Per(n) e(n) = d(n) - y(n) with = with + wre(n)Per(n-t)/var(n) k=0,1,2,63 likere we use filter order = 64 and mu = 0,01. Chen, Chein-Chung Murch, 1989	SO - Additive transversal filter mith Mormalized LDS algorithm using the TMSZGCCO (3) y(n) = Suff with Per(n-t) > Mo.1,2,,63 y(n) = Suff with Per(n-t) > (1-r) Per(n) Per(n) var(n) = revar(n-t) > (1-r) Per(n) Per(n) w(t) = with > the toler(n-t)/var(n) Per(1,2,63) Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chang Murch, 1989	CO - Adaptive transversal filter with Normalized LNS algorithm using the TNS220C20 SS y(n) = SM with Px(n-1) + (1-7) Px(n) Px(n) var(n) = revar(n-1) + (1-7) Px(n) Px(n) var(n) = revar(n-1) + (1-7) Px(n) Px(n) with = with + ure(n) Px(n-1) / var(n) Px(n) With = with + ure(n) Px(n-1) / var(n) Px(n) Chen, Chein-Chang Murch, 1999	00 - Adaptive transversal filter with Normalized LNS algorithm using the TNS20020 63 y(n) = SUM withex(n-t) k-0,1,2,,63 k-0 var(n) = revar(n-1) + (1-r) rex(n) rex(n) e(n) = d(n) - y(n) with = with + wee(n) rex(n-t) /var(n) k-0,1,2,63 Where we use filter order = 64 and mu = 0.01. Chen, Chein-Chang Nurch, 1989	CO - Adaptive transversal filter with Normalized LNS algorithm using the TMSZCC20 (3) (4) (5) (6) (7) (8) (8) (9) (9) (9) (9) (9) (9	CO - Adaptive transversal filter with Normalized LMS algorithm using the TMS2CC20 Algorithm: Algorit	CO - Adaptive transversal filter with Narmalized LNS algorithm using the TMS200200 Algorithm: Algorit	(SO - Adaptive transversal filter with Normalized LNS algerithm using the TMSZGCZO Algerithm: Algorithm: Algori	
using the TMSZ20C20 (a) = SUM u(k)ex(n-k)) an(a) = Fewar(n-1) + (1) (b) = d(n) - y(n) (k) = u(k) + ure(n)ex(i (k) = u(k) + ure(n)ex(i Chen, Chein-		- Ambative Transversal using the TMS220C20 (a) = SUM u(k)ex(n+1) + (1) k=0 ar(a) = revar(a-1) + (1) (a) = d(a) - y(a) (b) = u(k) + uee(a)ex(i (k) = u(k) + uee(a)ex(i Chen, Chein-	- Adaptive Transversal using the TMS220C20 (a) = SUM u(k)ex(n-t) 1 k0 ar(a) = Frequr(a-1) + (1 (a) = d(n) - y(n) (k) = u(k) + ure(a)ex(i (k) = u(k) + ure(a)ex(i Chen, Chein-	Adaptive transversal using the TMSS20C30 rithms: AS (a) = SIM u(k)ex(n-k) 1 kod ar(a) = revar(a-1) + (1 (n) = d(n) - y(n) (k) = u(k) + uee(a)ex(4 bere we use filter ord4 Chen, Chein-	Adaptive transversal using the TMSZ2CCO rithms: 63 63 S.M. a. S.M. wikibex(n-k)) ar(a) = rever(n-1) + (1) (n) = d(n) - y(n) (k) = wikib + une(a) bx(if here we use filter orde	-Adaptive transversal using the TMS220C30 rithm: 63 (a) = SM w(k)ex(n-k) 1 k=0 ar(a) = revor(n=1) + (1 (n) = d(n) - y(n) (k) = w(k) + use(a)ex(i (k) = w(k) + use(a)ex(i Chen Chein-	Adaptive transversal using the TMS220C30 rithm: 63 (a) = SM w(k)**x(n-k) } (b) = (b) = (c) + (c) = (d) - y(n) (d) = d(n) - y(n) (d) = d(n) - y(n) (d) = d(n) - w(n) *x(n) (d) = d(n) - w(n) (d) = d(n) - w(n) *x(n) (d) = d(n) - w(n) *x(n) (d) = d(n) - w(n) *x(n) (d) = d(n) - w(n) (d)	using the Thissoccas using the Thissoccas rithms (a) SUM u(k)ex(n+1) 1 k=0 ar(a) = revar(n+1) + (1 (a) = d(n) - y(n) (b) = d(n) - y(n) (c) = u(k) + ure(n)ex(CO - Adaptive transversal filter with Normalized LNS algorithm using the TNSS20C20 Algorithm: Algori	
using the TMS220C20 rithm: 63 (a) = SLM u(k)ex(n+1) + (1) ar(a) = revar(a-1) + (1) (b) = d(a) - y(a) (c) = u(k) + ume(a)ex(a) here we use filter ordel Chen, Cheim	using the THSESOCEO using the THSESOCEO ka ar(a) = SM u(k)ex(n-k) 1 ka ar(a) = revar(n-1) + (1 (a) = d(a) - y(n) (b) = u(k) + use(a) ex(i here we use filter ordal Chen, Cheim	- Acceptive Transversal using the TMS220C20 (a) = SIM u(k)ex(n-k)) k=0 ar(a) = revar(n-1) + (1) (a) = d(a) - y(a) (b) = u(k) + use(a) ex(i (c) = u(k) + use(a) ex(i Compared to the consequence of	- Adaptive Transversal using the TMSS20020 rithal AS (a) = SIM u(k)*x(n-k)) k=0 ar(a) = r*var(n-1) + (1) (a) = d(a) - y(a) (b) = u(k) + u*e(a)*x(i there we use filter ordal bere we use filter ordal Chen, Cheim	Adaptive transversal using the TMSS20C30 rithms: 63 (a) = SIM u(k)ex(n-k) 1 ar(a) = revoar(n-1) + (1 (n) = d(n) - y(n) (k) = u(k) + use(n)ex(n) k) (k) = u(k) + use(n)ex(n) Chen, Chen,	Adaptive transversal using the TMSZ2CCCO rithms: 63 (n) = SSM wiki>Px(n-k) 1 k=O ar(n) = revoar(n-1) + (1 (n) = d(n) - y(n) (k) = wiki + upe(n) Px(i) kb = wiki + upe(n) Px(i) Chen, Chen, Chen,	using the TMS220C30 rithm: 63 63 (a) = SUM u(k)ex(n+1) b k=0 an(a) = revar(n=1) + (1) (b) = d(n) - y(n) (c) = u(k) + u*e(a)ex(i) (k) = u(k) + u*e(a)ex(i) Chen, Chen, Chen	-Adaptive transversal using the TMS20020 ritha: 63 63 6.0 = SMH w(k)ex(n-k) 1 k=0 ar(a) = r*+var(a-1) + ((in) = d(n) - y(n) (k) = w(k) + w**e(a)*ex((here we use filter or**e	- Adaptive transversal using the TMSZ20C20 rithus (a) SM w(k)ex(n-k) 1 k= SM w(k)ex(n-k) + (1) ar(a) = revar(n-1) + (1) (n) = d(n) - y(n) (k) = w(k) + use(a)ex(i here we use filter orda here we use filter orda	(SO - Adaptive transversal filter with Normalized LNS algorithm Using the TMS20020 (S) (A) = SAM with Par(n+t) k=0,1,2,,63 (A) = SAM with Par(n+t) k=0,1,2,,63 (A) = (An - y(n)) (A(n) = with + une(n) Par(n+t) x=0,1,2,63 (A) = with + une(n) Par(n+t) x=0,1,2,63 (A) = with + une(n) Par(n+t) x=0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,	
using the TMSZ20C20 (a) = SIM u(k)*x(n-k) i (b) = (a) = r*var(n-1) + (i (c) = d(n) - y(n) (d) = u(k) + u*e(n)*x(i here we use filter ord Chen, Chen, Chen.		using the THSESCOCIO using the THSESCOCIO 63 (a) = SIM u(k)ex(n-k) 1 = Companie (a) = companie (a) + ((a) (b) = u(k) + urbe(n)ex((b) (c) = u(k) + urbe(n)ex((b)ex(- Adaptive Transversal using the TMSS20C20 (a) = SM wiki*ex(n-k) i k=0 ar(a) = revar(a-1) + (() (a) = d(n) - y(n) (k) = wiki + wee(n)*ex(n	Adaptive transversal using the TMSS20C30 rithms: 63 (n) = SSM wiki>ex(n-k) i k=0 ar(a) = revoar(n-1) + (() (n) = d(n) - y(n) (k) = wiki + ure(n)ex(f here we use filter ord Chen, Chein	- Adaptive transversal using the TMS200500 crithms: 63 63 63 63 64 65 65 66 66 67 68 68 68 68 68 68 68	using the TMS220C30 rithm: 63 63 64) = SLM u(k)ex(n-k) 1 k=0 ar(a) = revear(a-1) + (1) (n) = d(n) - y(n) (k) = u(k) + uer(a)ex(4 here we use filter ord4 Chen, Chen, Chen,	Adaptive transversal using the TMSZSCSO rithm: 63 (a) = SSH w(k)*x(n-k)) k=0 ar(a) = r*var(a-1) + (1) (a) = d(a) - y(a) (b) = w(k) + w*ve(a)*x(i) betwee use filter ord Chen, Chen, Chen,	using the Transversal using the TRSZOCCO rithm: 63 (a) = SIM u(k) ex(n-k) i k=0 (b) = d(n) - y(n) (c) = d(n) - y(n) (d) = u(k) + use(n) ex(i here we use filter ord	(SO - Adaptive transversal filter with Normalized LMS algorithm using the TMSZ20C20 Algorithm: Algor	
using the TMS220C30 (a) SUM u(k)*x(n+k) 1 k=0 ar(a) = F*var(n+1) + (1 (a) = d(n) - y(n) (b) = u(k) + u*e(n)*x(i (c) = u(k) + u*e(n)*x(i (c) = u(k) + u*e(n)*x(i	using the THS220C20 using the THS220C20 63 (a) = SUM u(k)ex(n-k)) k=0 an(a) = Fewar(n-1) + (1) (a) = d(a) - y(a) (b) = u(k) + ure(a)ex(f here we use filter orda	- Acceptive Transversal using the TMS220C20 (a) SMM u(k)ex(n+1) + (1) k=0 an(a) = free (n+1) + (1) (a) = d(a) - y(a) (b) = u(k) + uee(a)ex(f kere we use filter orda	- Adaptive Transversal using the TMS220C20 (a) = SUM u(k)ex(n-t) 1 k=0 ar(a) = Frenc(a-1) + (1 (a) = d(n) - y(n) (b) = u(k) + ure(a)ex(i (c) = u(k) + ure(a)ex(i	Adaptive transversal using the TMSS20C30 rithma: (a) = SLM u(k)ex(n-k) k0 ar(a) = Freque(a-1) + (1) (a) = d(n) - y(n) (b) = u(k) + upe(a) Px(f) (c) = u(k) + upe(a) Px(f) (d) = u(k) - y(n)	using the TMS20030 rithm: 63 63 (a) = SLM w(k)ex(n-k) 1 k=0 ar(a) = rever(n-1) + (1) (a) = d(a) - y(a) (b) = d(a) - y(a) (c) = we use filter order bere we use filter order	using the TMS220C30 using the TMS220C30 rithm: 63 (a) = SM u(k)ex(n-k) 1 (b) = (a) = revox(n-1) + (i) (c) = d(n) - y(n) (k) = u(k) + uec(n)ex(i) (k) = u(k) + uec(n)ex(i) (k) = u(k) + uec(n)ex(i)	Adaptive transversal using the TMS220C30 rithms: 63 (a) = SM w(k)*x(n-k) i k=0 ar(a) = revar(a-1) + (1 (b) = d(n) - y(n) (c) = w(k) + we*c(a)*x(i k=0)*x(i k=0)*	using the Theseoreal using the Theseocean tithas: 63 (a) SUM u(k)ex(n+1) + (i) an(a) = Fewar(n+1) + (i) (b) = d(n) - y(n) (c) = u(k) + uhe(n)ex(i) (d) = u(k) + uhe(n)ex(i) (e) = u(k) + uhe(n)ex(i)ex(i)ex(i)ex(i)ex(i)ex(i)ex(i)ex(i	CO - Adaptive transversal filter with Normalized LNS algerithm using the TNSS20C20 Algorithm: Algori	
using the TMS220C30 63 (a) SUM u(k)ex(n+1) 1 ke0 ar(a) = revar(a-1) + (1 (a) = d(a) - y(n) (b) = u(k) + use(a)ex(a) (c) = u(k) + use(a)ex(a)	using the THS220C20 using the THS220C20 k3 (a) = SIM u(k)ex(n-k) 1 k0 un(a) = revar(n-1) + (1 (a) = d(n) - y(n) (b) = u(k) + use(n)ex(n) (c) = u(k) + use(n)ex(n)ex(n)ex(n)ex(n)ex(n)ex(n)ex(n)e	- Acceptive Transversal using the TMS220C20 (a) = SIM u(k)ex(n-k)) k=0 ar(a) = rever(n-1) + (1) (a) = d(a) - y(n) (b) = u(k) + use(a)ex(i here we use filter orda	using the THESEOCEO using the THESEOCEO (a) = SIN u(k)*x(n-k)) k=0 ar(a) = revar(n-1) + (1) (a) = d(a) - y(a) (b) = u(k) + u**e(a)*x(i) (c) = u(k) + u**e(a)*x(i)	Adaptive transversal using the TMSS20C30 rithms 63 (a) = SIM u(k)ex(n-k) 1 km0 ar(a) = revoar(n-1) + (1 (n) = d(n) - y(n) (k) = u(k) + use(n)ex(n here we use filter orde	Adaptive transversal using the TMSZ2CCCO rithms: 63 (n) = SSM wiki>Px(n-k) i k-O ar(n) = revoar(n-1) + (1 (n) = d(n) - y(n) (k) = wiki + use(n) bx(i here we use filter orde	using the TMS220C30 rithm: 63 63 (a) = SUM u(k)ex(n+1) 1 k=0 ar(a) = revar(n=1) + (1 (a) = d(n) - y(n) (b) = u(k) + ues(a)ex(i (k) = u(k) + ues(a)ex(i	Adaptive transversal using the TMS220C30 ritha: 63 (a) SUM u(k)ex(n+1) 1 k=0 ar(a) = revar(a-1) + (1 (n) = d(n) - y(n) (k) = u(k) + use(a)ex(a here we use filter orda	- Adaptive transversal using the TMSZ20C20 rithm: (a) SUM u(k)ex(n-k) 1 k=0 ar(a) = revar(n-1) + (1 (a) = d(a) - y(n) (b) = d(a) - y(n) (c) = u(k) + use(a)ex(n) (d) = u(k) - use(a)ex(n)	ICO - Adaptive transversal filter with Normalized LNS algerithm Using the TNS220C20 G3 y(n) = SAN uttl=x(n+t) k=0,1,2,,63 k=0 var(a) = rewar(n-1) + (1-r)ex(a)ex(a) e(n) = d(n) - y(n) u(t) = u(t) + ure(a)ex(a-t)/var(a) k=0,1,2,63 Where we use filter order = 64 and mu = 0.01.	
using the INSZ20C20 (a) = SIM $u(k)$ ex($n+k$) $k=0$ ar(a) = five $u(k)$ ex($n+k$) + (i) (a) = $d(a)$ - $y(a)$ (b) = $u(k)$ + u ec(a) ex(i) (b) = $u(k)$ + u ec(a) ex(i) where we use filter ord		- Addative Transversal using the TMS220C20 (a) = SM w(k)*x(n-k) 1 k=0 ar(a) = r*var(a-1) + (() (a) = d(a) - y(a) (k) = w(k) + u**e(a)*x(() kere we use filter orde	- Adaptive Transversal using the TMSS20020 63 (a) = SM w(k)**x(n+k) 1 k=0 ar(a) = r**var(n+1) + (1 (a) = d(a) - y(a) (k) = w(k) + u**e(a)**x(f here we use filter orde	Adaptive transversal using the TMSS20C30 rithms: 63 (n) = SSM wiki>Px(n-k) 1 k=0 an(n) = revar(n-1) + (1 (n) = d(n) - y(n) (k) = wiki + uee(n) Px(i) here we use filter orde	using the TMS200500 rithm: 63 (a) = SUM u(k)ex(n-k)) k=0 ar(a) = rewar(a-1) + (() (a) = u(k) + uer(a) ex(i) (b) = u(k) + uer(a) ex(i) here we use filter orda	using the TMS220C30 rithm: A3 (a) = SLM u(k)ex(n-k) 1 k=0 ar(a) = rever(n-1) + (1) (b) = d(n) - y(n) (c) = u(k) + use(n)ex(n) then we use filter ord	Adaptive transversal using the TMS20020 rithm: 63 (a) = SM u(k)ex(n-k) 1 k=0 ar(a) = revar(a-1) + (1) (a) = d(a) - y(a) (b) = u(k) + uec(a) ex(i here we use filter ord	using the TMS200C30 using the TMS200C30 rithm: 63 (a) = SUM u(k)*x(n+k) i k=0 an(a) = reven(n+1) + (i (a) = d(a) - y(a) (b) = u(k) + u*e(a)*x(i here we use filter ord	(SO - Adaptive transversal filter with Normalized LNS algorithm using the TNSZ20C20 Algorithm: Algor	
using the TMS220C30 (a) = SUM u(k)*x(n+k)) (b) = K=0 un(a) = F*van(n-1) + (1) (c) = d(n) - y(n) (k) = u(k) + u*e(n)*x(i) (k) = u(k) + u*e(n)*x(i)		- Addative Transversal using the TMS220C20 (a) SUM u(k)ex(n+1) 1 k=0 an(a) = Fewar(a-1) + (1 (a) = d(n) - y(n) (k) = u(k) + ure(a)ex(i (k) = u(k) - use filter orda	- Adaptive Transversal using the TMS220C20 (a) = SUM u(k)ex(n+1) 1 k=0 an(a) = Fewar(a-1) + (1 (a) = d(n) - y(n) (k) = u(k) + ure(a)ex(i kere we use filter orde	Adaptive transversal using the TMSS20C30 rithm: (a) = SIM u(k)ex(n-k) 1 k=0 un(a) = revar(a-1) + (1 (a) = d(n) - y(n) (b) = u(k) + uee(a)ex(i (k) = u(k) + uee(a)ex(i	Adaptive transversal using the TMS200500 rithm: 6.3 6.3 S.M. w(k)ex(n-k) 1 k=0 k=0 ar(a) = reven(n-1) + (1) (n) = d(n) - y(n) (k) = w(k) + use(a)ex(i) (k) = w(k) + use(a)ex(i) kene we use filter orde	-Adaptive transversal using the TMS220C30 rithm: 63 (a) = SM w(k)*x(n-k) 1 k=0 ar(a) = r*var(a-1) + (1 (b) = d(a) - y(a) (c) = d(a) - y(a) (d) = d(b) - we(a)*x(if here we use filter orde	Adaptive transversal using the TMS220C30 rithms: 63 (a) = SM w(k)*x(n+k) 1 k=0	using the Theseorsal using the Theseorsal ritha: (a) SSM u(k)ex(n+1) 1 k=0 ar(a) = revar(n+1) + (1) (b) = d(n) - y(n) (k) = u(k) + ure(n)ex(i (k) = u(k) + ure(n)ex(i	NO - Adaptive transversal filter with Mermalized LNS algorithm using the TMS220C20 Algorithm: Algorithm: Red Var(a) = SM withex(n-t) k=0,1,2,,63 k=0 Var(a) = remar(n-1) + (1-r)ex(n)ex(n) e(n) = d(n) - y(n) with + wee(n)ex(n-t)/var(n) k=0,1,2,63 Midnere we use filter order = 64 and mu = 0,01.	
using the TNS220C30 (a) = SJM w(k)=x(n+k) 1 k=0 ar(a) = r=var(a-1) + (1 (a) = d(a) - y(n) (b) = w(k) + u=e(a)=x(i	modulive interpretation using the TMS20020 (a) = SIM u(k)ex(n-k) ko ko ko ko ko ko ko	- Medative Transversal using the TMS220C20 ritha: (a) = SJR u(k)*x(n-k)) ar(a) = revar(a-1) + (1) ar(a) = d(a) - y(a) (b) = u(k) + u**e(a)*x(i	- Addative Transversal using the TMSS20C20 rithm: 63 (a) = SJM u(k)ex(n-k) 1 ar(a) = revoar(n-1) + (1 (a) = d(n) - y(n) (b) = u(k) + use(a)ex(i	Adaptive transversal using the TMSS20C30 rithms: 63 (a) = SM w(k)*x(n-k) i = SM av(k)*x(n-k) i (a) = d(n) - y(n) (b) = w(k) + u**e(n)*x(i (k) = w(Adaptive transversal using the TNS220C30 rithms: 63 (a) = SM with Px(n-t.) 1 kr0 kr0 ar(a) = revar(a-1) + (() (a) = d(a) - y(a)	- Adaptive transversal using the TRS20020 ritha: 63 (a) = SM u(k)ex(n+1) k k0 an(a) = révar(n+1) + (1 (a) = d(n) - y(n) (b) = u(k) + ube(a)ex(f (k) = u(k) + ube(a)ex(f	Adaptive transversal using the TMS200200 ritha: 6.3 6.3 6.4 8.3 6.5 8.4 6.6 8.4 6.7 8.6 8.6 8.7 8.7 8.7 8.7 8.7	using the Transversal using the TRS20000 ritha: AS (A) = SIM u(k) Px(n-k)) R=0 R=0 ar(a) = revar(a-1) + (1) (A) = d(a) - y(a)	ISO - Adaptive transversal filter with Mermalized LNS algorithm using the TMSZ20C30 AS y(n) = SM witklex(n-t) k=0,1,2,,63 y(n) = Tever(n-t) + (1-+)ex(n)ex(n) e(n) = d(n) - y(n) witk) = witk) + wee(n)ex(n+k)/var(n) k=0,1,2,63	3
using the TMSZ20C30 (a) = SM u(k)*x(n+k) i k=0 ar(a) = r*van(n+1) + (i (n) = d(n) - y(n) (k) = u(k) + u*ve(n)*x(i		- Addative Cransversal using the TMS220C20 63 (a) = SUM u(k)*x(n+k) 1 k=0 ar(a) = revga(a-1) + (1 (n) = d(n) - y(n) (k) = u(k) + u*e(a)*x(i	- Adaptive Transversal using the TMSS20030 63 (a) = SM wiki>ex(n-t) i k=0 ar(a) = revar(a-1) + (((a) = d(n) - y(n) (b) = wiki + wee(a)ex(i	Adaptive transversal using the TNSS20C30 63 (a) = SJM u(k)ex(n-k) 1 k=0 ar(a) = revar(a-1) + (((n) = d(n) - y(n) (k) = u(k) + use(a)ex(i	Adaptive transversal using the TNS20030 ritha: 63 (a) = SMM u(k)ex(n-k) 1 k=0 ar(a) = rever(n-1) + (1 (n) = d(n) - y(n) (k) = u(k) + use(a)ex(i	- Adaptive transversal using the TMS220C30 ritha: 63 (a) = SM w(k)ex(n-k) 1 k=0 ar(a) = revar(a-1) + (1) (a) = d(a) - y(a) (b) = w(b) + wee(a)ex(i	Adaptive transversal using the TMS20020 rithm: 63 (a) = SIM u(k) ex(n-k) 1 k=0 ar(a) = revar(n-1) + (1 (n) = d(n) - y(n) (k) = u(k) + use(n) ex(i	- Adaptive transversal using the TMS20030 ritha: 63 (a) = SIM ulk)*x(n-k) 1 k=0 ar(a) = revar(a-1) + (1 (a) = d(a) - y(a) (b) = d(a) - y(a)	(SO - Adaptive transversal filter with Mornalized LUS algorithm using the TMS220C30 Algorithm: 63 y(n) = SLM wither(n-t) k=0,1,2,,63 yar(n) = revar(n-1) + (1-r)ex(n)ex(n) e(n) = d(n) - y(n) with = with + wee(n)ex(n-t)/var(n) k=0,1,2,63	
using the TRS220C30 (a) SUM u(k)ex(n+1) 1 k=0 ar(a) = revar(a-1) + (1 (a) = d(n) - y(n) (k) = u(k) + uee(a)ex(t		- Acceptive Transversal using the TMS220C20 (a) = SMH u(k)ex(n-t) 1 ke0 ar(a) = rever(a-1) + (1 (a) = d(n) - y(n) (k) = u(k) + uee(a)ex(e)	- Adaptive Transversal using the TMS220C20 (a) = SIM u(k)ex(n-k) 1 ke0 ar(a) = revar(a-1) + (1 (a) = d(n) - y(n) (k) = u(k) + uee(a)ex(e)	Adaptive transversal using the TMSS20C30 rithm A3 (A) SUM u(k) Px(n-k) 1 k-0 ar(a) = revar(a-1) + (1 (n) = d(n) - y(n) (k) = u(k) + uee(a) Px(i	Adaptive transversal using the TMSZ2CCO rithms: 63 SM u(k)ex(n-k) 1 RCO ar(a) = revar(n-1) + (1 (n) = d(n) - y(n) (t) = u(k) + uee(a)ex(e	- Adaptive transversal using the TMS20C20 ritha: 63 (a) = SM u(k)*x(n+k) i k=0 ar(a) = r*var(n+1) + (1 (a) = d(a) - y(a) (k) = u(k) + u*e(a)*x(t)	Adaptive transversal using the TNS220C30 ritha: 63 (a) = SIM u(k)*x(n+k) 1 k=0 an(a) = t*van(n-1) + (1 (a) = d(a) - y(a) (k) = u(k) + u*ve(a)*x(t)	using the Transversal using the TRSZ00C30 ritha: 63 (a) SMH u(k)ex(n+1) 1 k=0 ar(a) = revar(a-1) + (1 (a) = d(n) - y(n) (k) = u(k) + uee(a)ex(e)	NOO - Adaptive transversal filter with Normalized LNS algorithm using the TMS220C20 Algorithm: Algorithm: Red Var(a) = SM w(k) Px(n-k) k=0,1,2,,63 var(a) = revor(a-1) + (1-r)Px(a)Px(a) e(n) = d(n) - y(n) w(k) = w(k) + wee(a)Px(a-k)/var(a) k=0,1,2,63	
using the TMS200C30 63		- Medative Transversal using the TMS220C20 rithal (a) = SJM u(k)*x(n-k)) k=0 ar(a) = r*ver(n-k) + (i (n) = d(n) - y(n) (k) = u(k) + u**e(a)*x(i (k) = u(k) + u**e(a)*x(i	- Adaptive Transversal using the TMSS20020 in a SM wi(k)ex(n-k) i kn0 ar(a) = revoar(a-1) + (1 (n) = d(n) - y(n) (k) = wi(k) + upe(a)ex(i (k) = wi(k) + upe(a)ex(i	Adaptive transversal using the TMSS20C30 rithms 63 (a) = SM w(k)*x(n-k) 1 k=0 ar(a) = revoar(a-1) + (1 (a) = d(a) - y(a) (b) = w(k) + w**e(a)*x(f(k) = w(k) + w**e(a)*x(f(k) + w	Adaptive transversal using the TRS20030 crithms: 63 (n) = SSM wiki Pec(n-k) 1 k-0 ar(n) = revoar(n-1) + (() (n) = d(n) - y(n) (n) = d(n) - y(n) (k) = wiki + upe(n)Pec(n)	- Adaptive transversal using the TNS220C30 ritha: 63 (a) = SUN u(k)ex(n+t) 1 k=0 ar(a) = revar(n+1) + (1 (a) = d(n) - y(n) (b) = u(k) + ues(a)ex(i (k) = u(k) + ues(a)ex(i	Adaptive transversal using the TMS220C30 rithms 63 (a) = SLM w(k) ex(n+t) 1 k=0 ar(a) = rever(a-1) + (1) = u(k) + uer(a) ex(n) = (k) + uer(a) ex(k)	- Adaptive transversal using the TMS220C20 ritha: 63 (a) = SJM w(k)*x(n-k)) = SM w(k)*x(n-k)) ar(a) = revoar(a-1) + (1 (n) = d(n) - y(n) (k) = w(k) + u**e(a)*x(i(k)	ISO - Adaptive transversal filter with Mermalized LNS algorithm using the TNSZ20C30 AS y(n) = SM with Px(n-t) k=0,1,2,,63 y(n) = T-bwar(n-t) + (1-r)Px(n)bx(n) e(n) = d(n) - y(n) with = with + with (n)bx(n-t) / with p=0,1,2,63	
using the TMSZ20C30 rithas: 63 (a) = SM u(k)*x(n+t) 1 k=0 ar(a) = r*var(a-1) + (1 (a) = d(a) - y(a)		- Adaptive Transversal using the TMS220C20 ritha: 63 (a) = SUM u(k)*x(n+t) 1 k=0 ar(a) = revar(a-1) + (1 (n) = d(n) - y(n)	- Adaptive Transversal using the TMSS20030 rithms: 63 (a) = SJM w(k)*ex(n+1) 1 k=0 ar(a) = r*ver(a-1) + (1 (a) = d(a) - y(n)	Adaptive transversal using the TNSS20C30 rithms: 63 (a) = SM w(k)ex(n+1) k k*0 ar(a) = rever(e-1) + (() (n) = d(n) - y(n)	Adaptive transversal using the TMS200500 rithm: 63 (a) = SUN w(k)ex(n-k) 1 k=0 ar(a) = rever(n-1) + (1 (n) = d(n) - y(n)	- Adaptive transversal using the TMS220C30 ritha: 63 (a) = SUN w(k)*x(n-k) 1 k=0 ar(a) = revar(a-1) + (1 (a) = d(a) - y(a)	Adaptive transversal using the TMS20020 ritha: 63 (a) = SIM u(k)*x(n-k) i k=0 ar(a) = revar(a-1) + (i (a) = d(a) - y(a)	- Adaptive transversal using the TMS20030 rithm: 63 (n) = SUM ulk)*x(n+1) 1 k=0 ar(n) = revar(n-1) + (1 (n) = d(n) - y(n)	NGO - Adaptive transversal filter with Normalized LNS algorithm using the TNS20C30 Algorithm: 63 y(n) = SM wiklex(n-t) k=0,1,2,,63 k=0 var(n) = revar(n-1) + (1-r)ex(n)ex(n) e(n) = d(n) - y(n)) = (H)
using the TRS200C30 ritha: 63 (a) = SUM u(k)*x(n+1) 1 k=0 ar(a) = r*var(a-1) + (1 (a) = d(n) - y(n)		- Adaptive Transversal using the TMS220C20 c) c) c) c) c) c) c) c) c) c)	- Adaptive Transversal using the TMS220C20 c) (a) = SIM u(k)ex(n-k) 1 ke0 ar(a) = revar(a-1) + (1 (a) = d(n) - y(n)	Adaptive transversal using the TMSS20C30 rithm: AS (A) = SUM u(k) ex(n-k) 1 k=0 ar(a) = revar(a-1) + (1 (n) = d(n) - y(n)	using the TNS20030 ritha: 63 (a) = SUH u(k)ex(n-k) i k=0 ar(a) = revar(a-1) + (1 (a) = d(n) - y(n)	-Adaptive transversal using the TMS20020 ritha: (a) = SM u(k)*x(n+k) i k=0 an(a) = r*van(a-1) + (1 (a) = d(a) - y(a)	Adaptive transversal using the TNS220C30 ritha: 63 (a) = SM u(k)*x(n+1) 1 k=0 ar(a) = r*var(n-1) + (1 (a) = d(n) - y(n)	using the Thesecraturing the These of the Th	NOO - Adaptive transversal filter with Mormalized LNS algorithm using the TMSZ20C20 Algorithm: Algorithm: Red var(a) = rehow(a-1) + (1-+)ex(a)ex(a) e(n) = d(n) - y(n)	
using the TNS220C30 rithm: 63 63 SJR u(k)*x(n-k) 1 kn0 kn0 ar(a) = r*var(a-1) + (1		- Medative Transversal using the TMS220C20 is 5 (a) = SM u(k)*x(n-k) 1 k=0 ar(a) = r*vaar(a-1) + (1 (a) = d(n) - y(n)	- Adaptive Transversal using the TMSS20020 cithms: (a) = SM w(k)*x(n-k) 1 k=0 ar(a) = r*vaar(a-1) + (1 (a) = d(n) - y(n)	Adaptive transversal using the TMSS20C30 rithms: 63 (a) = SM w(k)*x(n+k) 1 k=0 ar(a) = r*vear(a-1) + (1 (a) = d(a) - y(a)	Adaptive transversal using the TNS20050 rithm: 63 (n) = SM wiki-bx(n-k) 1 k=0 k=0 an(a) = revar(n-1) + (1 (n) = d(n) - y(n)	- Adaptive transversal using the TNS220C30 ritha: 63 (a) = SM u(k)ex(n+1) k R0 ar(a) = révar(n-1) + (() (a) = d(n) - y(n)	Adaptive transversal using the TMS200200 ritha: 6.3 6.3 S.BH w(k)ex(n-t) 1 k=0 8.0 k=0 k=0 k=0 k=0 k=0 k=0 k=0 k=0 k=0 k=	- Adaptive transversal using the TMS220C30 ritha: 63 (a) = SM w(k)*ex(n-k) 1 k=0 k=0 ar(a) = revar(a-1) + (1 (a) = d(n) - y(n)	(SO - Adaptive transversal filter with Mornalized LNS algorithm using the TNSZ20C30 Algorithm: Algo	
using the TMSZ20C30 rithas: 63 (a) = SIM u(k)*x(n+t) 1 k=0 ar(a) = r*var(n-1) + (1)		- Medative Transversal using the TMSS20030 rithm: 63 (a) = SJM with Pec(n+1) 1 Peol and a - v(n) (i) = v(n) - v(n)	- Adaptive Transversal using the TMSS20030 rithms: 63 (a) = SJM wither(n+1) 1 km0 ar(a) = rever(n+1) + (i	Adaptive transversal using the TNSS20C30 rithms 63 (a) = SMM with ex(n+1) 1 ke0 ar(a) = rever(n+1) + (i)	Adaptive transversal using the TMS20030 rithm: 63 63 (A) = SUN w(k)ex(n+k) k R=0 ar(a) = rever(n+1) + (1) (b) = d(a) - v(a)	- Adaptive transversal using the TMS220C30 ritha: 63 (a) = SM w(k)*x(n-k) 1 k=0 ar(a) = r*var(a-1) + (1)	Adaptive transversal using the TMS20C30 rithm: 63 (a) = SIM w(k) ex(n-k) 1 k=0	- Adaptive transversal using the INS20030 rithm: 63 (a) = SUN u(k)*x(n+k) i k=0 k=0 k=0 mar(n+1) + (i m) = (i m) = (i m) + (i m) = (i	NSO - Adaptive transversal filter with Mornalized LNS algorithm using the TNS20C30 Algorithm: 63 y(a) = SM withex(a-t) k=0,1,2,,63 k=0 var(a) = rever(a-1) + (1-r)ex(a)ex(a)	
ritha: 63 (a) = SUM u(k) × (n+1) 1 k0 ar(a) = r+var(a-1) + (1	- modulive independent using the TMSZGOC3O rithd: 63 SMH u(k)ex(n+t) 1 k=0 ar(a) = r+var(a-1) + (1	- Medative Transversal using the TMS220C20 rithm: k3	- Meaptive Transversal using the TMS220C20 ritha: k3 (a) = SIM u(k)ex(n-k) 1 k0 ar(a) = rever(e-1) + (1	Adaptive transversal using the TMSS20C30 rithm: A3 (A) = SIM u(k) ex(n-k) 1 R=0 R=0 ar(a) = rever(e-1) + (1	using the TNS20020 vising the TNS20020 ritha: 63 (a) = SM w(k)*x(n+k) i k=0 ar(a) = r**var(a-1) + (i	- Adaptive transversal using the TMS20020 ritha: 63 (a) = SM u(k)+x(n+1) 1 k=0 ar(a) = r+war(a-1) + (1	Adaptive transversal using the TMS220C30 ritha: 63 (a) = SLM u(k) ex(n+1) 1 k0 an(a) = rewar(a-1) + (1	using the Thistocco using the Thistocco rithm: 63 (a) SUM u(k)ex(n+t) 1 k=0 ar(a) = revar(e-1) + (1	NOO - Adaptive transversal filter with Normalized LNS algorithm using the TMSS20C30 Algorithm: Algor	P = (0)
using the TNS200C30 ritha: (a) = SJM u(k)*x(n-k)) k=0 k=0 ar(a) = r*var(a-1) + ()	using the THS220C30 using the THS220C30 ritha: 63 (n) = SJN u(k)ex(n-k) 1 km0 ar(a) = revoar(a-1) + ()	- Meaptive Transversal using the TMS220C20 crithm: 63 = SM w(k)ex(n-k) 1 k=0 ar(a) = revour(n-1) + ((- Meaptive Transversal using the TMSS20030 cithms: (a) = SM u(k)*x(n+k) 1 k=0 ar(a) = r*vear(a-1) + ((Adaptive transversal using the TMSS20C30 rithm: 63 (a) = SM w(k)*x(n+k) 1 k=0 k=0 ar(a) = revar(a-1) + (1	Adaptive transversal using the TNS20050 rithms: 63 (a) = SIM with Px(n+t) 1 ke0 ke0 ar(a) = revar(a-1) + (1	-Adaptive transversal using the TMS200:30 ritha: 63 = SUN w(k)ex(n-t) 1 k=0 k=0 ar(a) = revor(n-1) + (1	Adaptive transversal using the TMSZ20C20 riths: 63 SJM w(k)ex(n-k) 1 k=0 k=0 ar(a) = revor(n-1) + (1	using the Transversal using the TRS200C30 rithas: 63 (a) = SJA u(k) ex(n-k) 1 k=0 k=0 ar(a) = revar(a-1) + (()	ISO - Adaptive transversal filter with Marmalized LNS algorithm using the TNSZ20C30 Algorithm: Algo	
using the TNS220C30 rithm: 63 (n) = SJM w(k)*x(n+k) k e=0 ar(n) = r*var(n=1) + (1	- modulive indiscenses using the TMSZ20C20 63 (a) = SJM u(k)ex(n+k) b k=0 ar(a) = rewar(a-1) + (1	- Meaptive Transversal using the TMSS20030 rithm: 63 (a) = SJM with Pec(n+1) 1 Prof. mr(a) = rever(n+1) + (1	- Adaptive Transversal using the TMSS20030 rithms: 63 (a) = SJM wither(n+1) 1 km0 ar(a) = rewar(a-1) + (1	Adaptive transversal using the TMSS20C30 rithms 63 (a) = SM u(k)ex(n+1) 1 k*0 ar(a) = rever(e-1) + (()	Adaptive transversal using the TMS200500 rithm: 63 63 60 = SUM w(k)ex(n+t) ke0 ke0 ar(n) = rewar(n-1) + ()	- Adaptive transversal using the TMS220C30 ritha: 63 (a) = SIM w(t)*x(n+t) 1 k=0 ar(a) = r*var(e-1) + (1)	Adaptive transversal using the TMS20030 rithm: 63 (a) = SM w(k)*x(n-k) 1 k=0 ar(a) = r*var(a-1) + (i)	using the TRSS20C30 using the TRSS20C30 rithm: 63 (a) = SJM u(k)*ex(n+1) 1 k=0 ar(a) = r*vear(a-1) + (1)	NGO - Adaptive transversal filter with Normalized LNS algorithm using the TNS20C30 Algorithm: 63 y(a) = SM wiklex(n-t) k=0,1,2,,63 k=0 var(a) = rever(n-1) + (1-r)ex(alex(a)	
using the THSS20030 rithm: 63 (a) = SM u(k)ex(n-k) k Re0 Re0	using the TMS220C30 rithms: 63 (a) = SIM w(k)ex(n-k) 1 R=0 R=0	- Meaptive Transversal using the TMSS20030 rithm: 63 (a) = SIM w(k)*x(n-k)) k=0 k=0	- Adaptive Transversal using the TMSS20030 rithm:	Adaptive transversal using the TMS220C3O rithm: 63 (a) = SUM w(k) ex(n-k) 1 k=0	Adaptive transversal using the TNS200530 ritha: 63 (a) = SM w(k)*x(n+k) i k=0 k=0 reval(n+k) i k=0	- Adaptive transversal using the TMS20020 ritha: 63 (n) = SM w(k) ex(n+k) t k=0	Adaptive transversal using the TMS220C30 riths: 63 (a) = SM w(k)*x(n+1) 1 kg	using the Thessocial using the Thessocial using the Thessocial rithms: 63 (a) = SLM u(k) ex(n+1) 1 kg	NOC - Adaptive transversal filter with Normalized LNS algorithm using the TMSZ20C20 Algorithm: Algorithm: Red Red Author(n-k) k-0,1,2,,63 k-0 Author(n-l) + (1-19x(n)k(n)	
using the TNS20030 rithms: 63 (n) = SM w(k)*x(n-k)	using the TMSS20C30 rithm: 63 (a) = SM w(k)*x(n-k) i k=0	- Modelive Transversal using the TMSSCOC30 rithm: 63 (a) = SM w(k)*x(n+t) 1 k=0	- Adaptive Transversal using the TMSSCOC3O rithm: 63 (a) = SLM w(k)*x(n+t) V	Adaptive transversal using the TNS220C30 ritha: 63 (a) = SJM u(k)ex(n+t) 1	Adaptive transversal using the TNS20020 rithm: 63 (a) = SM u(k)ex(n+t) 1 ke0	-Adaptive transversal using the TMSZOC30 ritha: 63 (a) = SLM w(k)ex(n+t) 1 k=0	Adaptive transversal using the TMSZ20C20 rithm: 63 63 SUM w(k)ex(n-k) 1 k=0	- Adaptive transversal using the TMSZOCCO ritha: 63 (a) = SJM w(k)*x(n+k) 1	NSO - Adaptive transversal filter with Marmalized LNS algorithm using the TNS220C30 Algorithm: AS y(n) = SM witklex(n-k) k=0,1,2,,63	
using the TNS20020 rithm: 63 (a) = SM u(k)*x(n+t) 1 k=0	using the TMSS2030 ritha: 63 (a) = SJM u(k)ex(n+t) 1 k=0	- Mountine Transversal using the TMSSCOC3O ritha: 63 (a) = SJM w(k)*x(n-t)) k=0	- Maptive Transversal using the TMS220C3O ritha: 63 63 = SUM w(k)ex(n-k) 1 k=0	Adaptive transversal using the TNS20030 rithm: 63 63 (a) = SIM w(k) ex(n-k) 1 km	- Adaptive transversal using the TNS20020 riths: 63 (a) = SIM w(k)*x(n+k) 1 k=0	Adaptive transversal using the TMS220C30 rithm: 63 63 (a) = SM w(k)ex(n-k) i	Adaptive transversal using the TMS220C30 rithm: 63 (n) = SM w(k)*x(n+k) i k=0	- Adaptive transversal using the TMSZ20C20 rithm: 63 (n) = SLM w(k)*x(n+k) k e=0	ICO - Adaptive transversal filter with Normalized LUS algorithm using the TMSZ20C30 Algorithm: Algo	
using the TNS2COC30 ritha: 63 (a) = SUM w(k)ex(n-k) 1 k=0	- mapping transversal using the TMSZCCCO rithm: 63 (a) = SLM w(k)ex(n-k) 1 k=0	- Medative Transversal using the TMSS20030 rithms 63 (A) = SAM w(k)ex(n-b) 1 k=0	- Adaptive Transversal using the TMSZ2CCCO rithm: 63 (n) = SJM w(k)+x(n+k) k k=0	Adaptive transversal using the TMS20020 rithm: 63 (n) = SJM w(k)ex(n+t) 1 k=0	- Adaptive transversal using the TMSZ0C30 rithm: 63 (a) = SJM w(k)ex(n+k) 1 k=0	Adaptive transversal using the TMS220C30 ritha: 63 (a) = SAM w(k)ex(n+k) k R=0	- Adaptive transversal using the TNS200C30 rithm: 63 (a) = SAM w(k)ex(n+t) 1 k=0	- Adaptive transversal using the TMSS20030 rithms: 63 (a) = SLM w(k)ex(n-k) 1 k=0	NGO - Adaptive transversal filter with Normalized LNS algorithm using the TNS20C30 Algorithms 63 y(n) = SMW w(k)tx(n-k) k=0,1,2,,63 k=0	
using the TMS2CC30 rithm: 63 (n) = SM w(k)*x(n-k)) k=0	using the TMS220C3O rithm: 63 (n) = SLM w(k) ex(n-k)) k=0	- Meaptive Transversal using the TMS220C3O rithm: 63 63 w(k) ex(n-k) i k=0M w(k) ex(n-k) i	- Adaptive fransversal using the TMS220C30 rithm: 63 (n) = SLM w(k) ex(n-k) b k=0.	- Adaptive transversal using the TMS220C30 rithm: 63 (n) = SLM w(k) ex(n-k) 1 k=0.00	- Adaptive transversal using the TMS220C30 ritha: 63 63 (a) = SUM w(k) ex(n-k) 1 k=0	ving the TMS20030 ving the TMS20030 rithm: 63 63 (a) = SM w(k)*x(n+k) k	using the TNS220C30 using the TNS220C30 rithm: 63 (a) = SM w(k)*x(n-k) i k=0	- Adaptive transversal using the TMS220C3O rithm: 63 (n) = SM w(k)*x(n+t) 1 k=0	NCO - Adaptive transversal filter with Normalized LNS algorithm using the TMSZ20C20 Algorithm: Algorithm: Algorithm (1) = SIM withex(n-t) k=0,1,2,,63 k=0	
using the TMSS20C30 rithm: 63 (a) = SIM w(k) ex(n-k) 1	using the TMS220C30 rithm: 63 = SM w(k) ex(n-k) 1 ben	- Memptive Transversal using the TMS220C30 rithm: 63 (a) = SM w(k) ex(n-k) i	- Adaptive fransversal using the TMS220C30 rithm: 63 (a) = SIM w(k) ex(n-k) i	- Adaptive transversal using the TNG320C30 ritha: 63 G1 (n) = SM w(k)ex(n-k))	- Adaptive transversal using the TMS220C30 ritha: 63 (a) = SM w(k) ex(n+k) i	- Adaptive transversal using the TMS220C30 rithe: 63 63 Bulklex(n-k) i	Adaptive transversal using the TMSZ0030 rithm: 63 63 (a) = SUM u(k) ex(n-k) 1 k=0	Adaptive transversal using the TMS20030 rithm: 63 63 (a) = SUM w(k) **C(n-k) 1 k-10 k-10 k-10 k-10 k-10 k-10 k-10 k-	ICO - Adaptive transversal filter with Normalized LNS algorithm using the TNSZCOCOO Algorithm: All y(n) = SM wikive(n+t) k=0,1,2,,63 y(n) = SM wikive(n+t) k=0,1,2,,63	•
using the TNS220C30 rithm: 63 (n) = SUM w(k) ex(n-k))	- momptive transversal using the TMS220C30 rithm: 63 (n) = SLM w(k) ex(n-k) 1	- Memptive Transversal using the TMS220C30 rithm: 63 (n) = SLM w(k) ex(n-k) l	- Adaptive fransversal using the TMS220C30 rithms: 63 (n) = SLM w(k)ex(n-k) l	- Adaptive transversal using the TNS220C30 rithm: 63 (n) = SJM w(k)**x(n-k))	- Adaptive transversal using the TMS320C30 ritha: 63 63 (a) = SJM u(k) (n-k) 1	- Adaptive transversal using the TMS220C30 ritha: 63 63 (a) = SIM w(k)**(n-k) b	- Adaptive transversal using the TNS2COC30 rithm: 63 (a) = SUM w(k)**x(n-k) b	- Adaptive transversal using the TMS220C30 rithm: 63 (A) = SJM w(k)*x(n+k) b	NGO - Adaptive transversal filter with Normalized LNS algorithm using the TNS20C30 Algorithms 63 y(n) = SUM withex(n-t) k=0,1,2,,63	드
using the TMS220C30 rithm: 63 63 = SUM w(k)*x(n=k))	- Modelive transversal using the TMS220C30 rithm: 63 63 (A) = SUM W(K)=K(n=K))	- Memptive transversal using the TMS220C30 rithm: 63 63 (A) = SUM W(K)=K(n=K))	- Mappive fransversal using the TNG320C30 ritha: 63 63 = SUM w(k)=K(n=k))	- Adaptive transversal using the TMSZ20C30 ritha: 63 63 = SUM w(k)=K(n=k))	- Adaptive transversal using the TMS320C30 ritha: 63 63 (n) = SM w(k)*x(n=k) i	using the TNS220C30 sitha: 63 63 Hulklex(n-k) i	using the TMS220C30 using the TMS220C30 rithm: 63 64 68 = SM w(k) ex(n-k) i	- Adaptive transversal using the TMS320C30 rithm: 63 63 M W(K) 94 M (K) 95 (m+k) 1	(CO - Adaptive transversal filter with Normalized LNS algorithm using the TNS20020 Algorithm: 63 V(n) = SUM w(k)ex(n+k) k=0.1.263	
using the TNS220C30 rithm:	using the TMS220C30 rithm:	- Mappive Transversal using the TMSZ20C30 ritha:	- Maptive fransversal using the TMSZ20C30 ritha:	- Adaptive transversal using the TMS220C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS220C30 rithm:	- Adaptive transversal using the TMS220C30 rithm:	- Adaptive transversal using the TMS220C30 rithm:	CO - Adaptive transversal filter with Normalized LNS algorithm using the TNSZCOC20 Algorithm:	, E
using the TMSZ20C30 rithm:	using the TMS220C20 rithm:	- Mosptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS220C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TNS320C30 rithm:	-Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS220C30 rithm:	CO - Adaptive transversal filter with Mormalized LUS algorithm using the TMSZ20C20 Algorithm:	D = (4)
using the TMSZ20C30 rithm:	moduling the TMS220C30 rithm:	- Momptive Transversal using the TMS220C30 rithm:	- Mdaptive transversal using the TMS220C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS320G30 rithe:	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS220C30 ritha:	IGO - Adaptive transversal filter with Normalized LNS algorithm using the TNS220C30 Algorithms 63	
using the TMS320C30 rithm:	using the TMS220C30 rithm:	- Momptive Transversal using the TMS320C30 rithm:	- Adaptive Transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS220C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TNS220C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	CO - Adaptive transversal filter with Normalized LNS algorithm using the TNS20C20	•
using the TMS320C30 rithm:	momptive transversal using the TMS320C30 rithm:	- Momptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS220C30 rithm:	- Adaptive transversal using the TNS320C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS220C30 rithm:	- Adaptive transversal using the TNS320C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	CO - Adaptive transversal filter with Normalized LNS algorithm using the TNSZCC20 Algorithm:	•
using the TMS320C30 rithm:	using the TMS320C30	- Momptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS320C30 rith a :	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	-Adaptive transversal using the TNG320C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	CO - Adaptive transversal filter with Normalized LNS algorithm using the TNSZCCC20 Algorithm:	•
using the TMS320C30 rithm:	using the TMS320C30 rithm:	- Modplive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS320C30 rithe:	- Adaptive transversal using the TMS320C30 rithe:	- Adaptive transversal using the TMS220C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS220C30 rithm:	CO - Adaptive transversal filter with Normalized LNS algorithm using the TMS220C20	
using the TMS320C30 rithm:	using the TMS320C30	 - Adaptive Transversal using the TMS320C30 rithe: 	- Adaptive transversal using the TMS320C30 rithe:	- Adaptive transversal using the TMS320C30 rithe:	- Adaptive transversal using the TNS320C30 rithe:	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	ICO - Adaptive transversal filter with Normalized LNS algorithm using the TNS200C30 Algorithm:	
using the TMS320C30 riths:	using the TMS320C30	 - Adaptive transversal using the TMS320C30 rithm: 	- Adaptive transversal using the TMS320C30 ritha:	- Adaptive transversal using the TMS320C30 rithe:	- Adaptive transversal using the TMS320C30 rithe:	- Adaptive transversal using the TMS320C30 riths:	- Adaptive transversal using the TNS320C30 rithm:	- Adaptive transversal using the TMS320C30 rithm:	NCO - Adaptive transversal filter with Normalized LNS algorithm using the TNS200200 Algorithm:	
using the TMS320C30	using the TMS320C30	 - Maptive transversal using the TMS320C30 rithm: 	 - Adaptive transversal using the TMS320C30 rithm: 	- Adaptive transversal using the TMS320C30 rithe:	- Adaptive transversal using the TMS320C30 rithe:	- Adaptive transversal using the TMS320C30 rithe:	- Adaptive transversal using the TMS320C30 rithe:	- Adaptive transversal using the TMS320C30 nithm:	NCO - Adaptive transversal filter with Normalized LNS algorithm using the TNS2COCO Allorithm:	
using the TMS320C30	using the TMS320C30	 - Adaptive transversal using the TMS320C30 	 - Adaptive transversal using the TMS320C30 	 Adaptive transversal using the TMS320C30 	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	NCO - Adaptive transversal filter with Marmalized LNS algorithm using the TNS200200	į
using the THS320C30	using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TNS320C30	ICO - Adaptive transversal filter mith Normalized LNS algorithm using the TNS220230	
using the TMS320C30	using the TMS320C30	- Maptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TNS320C30	- Adaptive transversal using the TMS320C30	ISO - Adaptive transversal filter with Normalized LNS algorithm using the TNS220C30	1
using the TNS320C30	using the TNS320C30	- Maptive Transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	NCO - Adaptive transversal filter with Normalized LNS algorithm using the TNS20C30	
using the TMS320C30	using the TMS320C30	- Maptive Transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	NCO - Adaptive transversal filter with Normalized LNS algorithm using the TNS20020	
using the TMS320C30	using the TMS320C30	using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	CO - Adaptive transversal filter with Normalized LNS algorithm using the TNSZCCZO	
using the TMS320C30	using the TMS320C30	- Adaptive Transversal using the TMS320C30	 Adaptive transversal using the TMS320C30 	 Adaptive transversal using the TMS320C30 	- Adaptive transversal using the TKS220C20	- Adaptive transversal using the THS320C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	CO - Adaptive transversal filter with Normalized LNS algorithm using the TNSZ20C20	
using the TMS320C30	using the THS320C30	- Adaptive transversal using the TMS320G30	- Adaptive transversal using the TMS320C30	 Adaptive transversal using the TMS320C30 	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS220C30	- Adaptive transversal using the TMS320C30	- Adaptive transversal using the TMS320C30	CO - Adaptive transversal filter with Normalized LNS algorithm using the TNS2CACO.	•
OCTOCOGE AND THE COLUMN	- MOMBELLIVE TECTORING	- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	CO - Adaptive transversal filter with Normalized LMS algorithm	5
	- MORDITAE ILEBAGESE	- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	CO - Adaptive transversal filter with Normalized LMS algorithm	•
		- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	- Adaptive transversal	CO - Adaptive transversal filter with Normalized LMS alocitim	
				A.A						
***************************************	***************************************	***************************************	***************************************	***************************************	+	***************************************				
***************************************	***************************************	***************************************	***************************************	***************************************	***************************************	***************************************				
stive transversal filter with Mereslived IMS alone	sails 30 handlessail dain sailed bearinged and								***********************************	

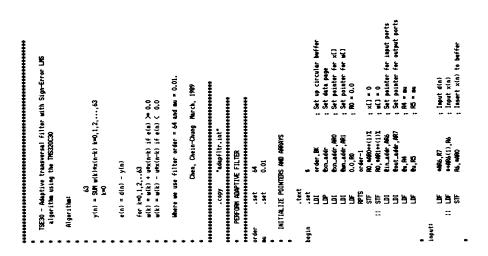
| Signe | 2,0,10 | 1,00 | 1,00 | 2,00 | 2,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00

Appendix E1. Transversal Structure with Sign-Error LMS Algorithm Using the TMS320C25

	:B8:	.usect	"parameters", 1	
***************************************	:300	.usect	parameters", 1	
19825 : Adaptive Filter Using Transversal Structure and Sim-frem (MS Alameithm) named Code	U: ENGE:	.usect	parameters , 1	
	MEGAU:	.usect	MEGNU: .usect "parameters",1	
	* PERFOR	H THE ADAP	* PERFORM THE ADAPTIVE FILTER	
	***	#		
y(n) = SLM w(k)+x(n-k) k=0,1,2,,63	*	. text		
	* ESTIMA	ESTINATE THE SIGNAL Y	MAL Y	
e(n) = d(n) - y(n)	-	8	8	
		8	1	· Configure BO as program memory
For k = 0,1,2,,63		¥.	0	Clear the P register
m(k) = m(k) + c+x(n-k) if e(n) >= 0		3	OME, 15	: Using rounding
$\mathbf{u}(\mathbf{k}) = \mathbf{u}(\mathbf{k}) - \mathbf{u}(\mathbf{k}) = \mathbf{u}(\mathbf{k})$		ž	AR3, XN	; Point to the oldest sample
Bank as men filless ander a to and an a 0 M	FIR	8	ORDER-1	; Repeat N times
			MH-0f-d00h, 4-	; Estimate Y(n)
Mote: This source program is the generic version: 1/0 configuration has		7 Jan		; Configure BO as data memory
not been set up. User has to modify the main routine for specific		æ æ æ	>	; Store the filter output
	•			
	+ OFECKI	CHECK THE SIGN OF ENROR	F ERROR	
1) PM status bit should be equal to 01.	•	-	=	I - retained I .
SXM status bit should be set to 1.		9		: ACT = - Y(n)
3) The current DP (data memory page pointer) should be page 0.		8	0	; ACC = D(n) - Y(n)
4) Data memory Offe should be 1.		BGE 2	NEXT	
5) Data memory U should be 32/.	•	5	MEGMU	; T register = →U
	TOUGH *	IPPATE THE LETCHTO	2	
Chen, Chein-Chung February, 1989		E 15	2	
	HEXT	Š	AR1, 0RDER-1	Set up counter
***************************************		ž	ARZ, IAN	; Point to the coefficients
		ž	AR3, XN+1	; Point to the data sample
		Ē	¥,	: b = 0 + X(n-k)
		E PC	5 4 1 2 4 1	: Load ACCH with W(k,n) & round
	•	! !		. P = [] = X(n-b)
		SACH	++,0,4R1	; Store W(k,n+1)
DEFINE ADDRESSES OF BUFFER AND COEFFICIENTS		BANZ	ADAPT, +-, AR2	
"buffer", OROER-1	FINIS	end.		
"buffer",1				
coeffs, ONDER				
reserve addresses for parameters				
"parameters",1				
Series de la company de la com				

Appendix E2. Transversal Structure with Sign-Error LMS Algorithm Using the TMS320C30

	; R2 = 0.0	u*	; y(n) = w[].x[] ; Include last result		; e(n) = d(n) - y(n)		Send out y(n)	Send out e(n)		Get Sign[e(n)]	(5 = S(e(n)) + u	R1 = S(e(n)] + u + x(n)	Initialize repeat counter		R1 = S[e(n)] + u + x(n-i-1)	R2 = ui(n) + S(e(n)) + u + x(n-i) $u(n+1) = ui(n) + S(e(n)) + u(n-i)$	Fer i = M - 2		Delay branch	WITHTT = WITH + SECTION PROFILE.	; Update last w													
PUT y(n)	0.0,R2 II 51,R	##RO++(1)%, ##R1++(1)%, R1 order-2 ##RO++(1)%, ##R1++(1)%, R1	RI,RZ,RZIIII ; y	M. e(n)	R2,87 ;	n) SIGNOLS	••	R7, *+4R7(1) ; S	2		 Se	1,155,R1	3,80	-	. 175,R1	_		•		RZ, ***(174, 174, 174, 174, 174, 174, 174, 174,	7(1		"buffer" order	"coeffs" order	vars , 1	"vers",1	"vars", 1	"wars", I	Chrit	S, in addr	40004080	0804002h	S.	\$
COMPUTE FILTER OUTPUT y(n)	Š	HPYF3 RPYF3 HPYF3	:: ADDF3	COMPUTE ETROR SIGNAL &(n)	SIBE	OUTPUT y(n) AND e(n) SIGNALS	STE	:: STF	UPDATE WEIGHTS w(n)	ð	EMOX.	HPYF3	ğ	er o		:: A00F3		:: A00F3	s !	SIF	STE	BEFINE CONSTANTS	1.5	. usect	in_addr .usect	out_addr .usect		m_addr .usect	cinit .sect		PJ09.	brow.	Pool.	208.



Appendix F1. Transversal Structure with Sign-Sign LMS Algorithm Using the TMS320C25

TSS : Adaptive Filter Using Transversal Structure and Sign-Sign LMS Algerithm , Looped Code Algerithm: 63 y(n) = SUM u(k)*x(n-k) k=0,1,2,,63 k=0	ë	.usect	perameters , 1	
Adaptive Filter Using Transversal Structure and Sign-Sign LMS Algorithm , Looped Code prithms Sign-Sign LMS Algorithm , Looped Code (A) (A) = SIM w(k) * k=0,1,2,,63 k=0 (A)				
i Addative Filter Using Transversal Structure and Sign-Sign LMS Algerithm, Looped Code prithm: 63 (A) = SUM with Px(n-k) k=0,1,2,,63 k=0	:	. esect	Parameters .	
and Sign-Sign LMS Algorithm , Looped Code prithm: (a) = SLM b(kl) ex(n-k) k=0,1,2,,63 k=0 k=0.1,2,,63	: 2005	.usect	"parameters"	
Algorithm: St St y(a) = SUR w(k) av(a-k) k=0.1,2,,63 k=0.10,1,2,,63	*******		***************************************	
writhm: 63 y(n) = SIM w(k)ex(n-t) k=0,1,2,,63	*	TOPH THE ADI	PERFORM THE ADAPTIVE FILTER	
63 y(n) = SIM w(k)ex(n-k) k=0,1,2,,63 k=0	*******	***************************************	*******************************	
y(n) = SIM u(k)ex(n-k) k=0,1,2,,63 k=0 k=0,1,2,,63 k=0		.text		
y(a) = SJM w(k)*x(n-t)	•			
, 1971 - 1984 - 1985	• ESTI	ESTIMATE THE SIGNAL Y	GWAL Y	
	•	3	2	
בנון - פנון / אנון		G G		: Cenfigure BO as program memory
		¥	•	Clear the Presister
For k = 0,1,2,,63		3	OME, 15	- Uking counding
u(k) = u(k) + u if $e(n) + x(n-k) > 0$		ž	AR3, XN	: Point to the oldest camele
$\mathbf{u}(\mathbf{k}) = \mathbf{u}(\mathbf{k}) - \mathbf{u}$ if $\mathbf{e}(\mathbf{n}) + \mathbf{x}(\mathbf{n} - \mathbf{k}) < 0$	FIR	AF PE	080ER-1	: Repeat N times
				: Estimate Y(n)
Where we use filter order = 64 and mu = 0.01.		0 4 20		; Configure BO as data memory
		PRC		
Note: Ints source program is the generic version; I/U configuration has	•	5	-	; Store the filter output
abolication.		Service are of the	É	
			25	
Initial conditions	•	Š	ARI. DROER-1	. Set us counter
1) PM status bit should be equal to 01.		ž	AR2, MM	: Point to the coefficients
2) SIM status bit should be set to 1.		ž	AR3, XIN+1	: Point to the data sample
3) The current DP (data memory page pointer) should be page 0.	*			
4) Data memory ONE should be 1.	90	CHECK THE STON OF EDROR	Y ENROR	
5) Data memory U should be 327.	•			
4		2		
Chen, Chein-Chung rebruary, 1989		5	_ {	; ACC = D(n) - Y(n)
***************************************	•	5	ž	
	•	UPDATE THE METCHTS	TIS	
DETINE PARAMETERS		•	;	
	į	3 9	±,0,#2	; ACC = X(n-k)
to min		5 6	1 6	; Get the Sign of EMR(n) + X(n-k)
		1 2		. Cat the sign with its sign setseemen
DEFINE ANDRESSES OF BRETED AND PREFETORING		Ì		Cot the sign atm 115 sign extension
IN MUNICIPALS OF BUTTER HOU CULTILIENTS		§ §	. 15	; Let the convergent factor AU or —AU . Undate M(k)
.usect "buffer".OBGER-1		Sec	# 1 AR	
		BONZ	ADAPT + AR3	
	•			
	FINISH	·end		
PESETNE ADDRESSES FOR PROMETERS				
.usect "parameters",1				

Appendix F2. Transversal Structure with Sign-Sign LMS Algorithm Using the TMS320C30

				į		
				RPTS RPVE	order=2	
TSS30 - Adapt	ive transversal filte	18830 - Adaptive transversal filter with Sign-Sign LMS	==		P1 P2 P2	[] []
algor	algorithm using the TMS320C30	00:30		9	81,72,18 21,72,18	i Include last result
Algoritha:			* COMPU	TE EPROR S	IGNAL e(n) AND OUTF	COMPUTE EPROR SIGNAL e(n) AND OUTPUT y(n) AND e(n) SIGNALS
•	3		•	SUB	R2.R7	: e(u) = d(u) - v(u)
y(n) = SUN	y(n) = SUM u(k) + x(n-k) k=0,1,2,,63	2,,63			R2, +4R7	; Send out y(n)
×	•			SIF	R7, *****(1)	; Send out e(n)
e(n) = d(e(n) = d(n) - y(n)		TADAU *	UPDATE NEIGHTS W(n)	n(u)	
for k=0,1	for k=0,1,2,,63		•	₹	-31,R7	; R7 = Sign[e(n)]
u(k)	u(k) = u(k) + u, if x(n-k)He(n) >= 0.0)#e(n) >= 0.0		XOR3	RO, R7, R5	; R5 = Sign[e(n)] + u
E(K)	w(k) = w(k) - u, if $x(n-k) + e(n) < 0.0$)*e(n) < 0.0		<u>5</u> 8	##R0++(1)%,R6	; R6 = x(n)
Where we	Where we use filter order π 64 and mu = 0.01.	4 and mu = 0.01.		X 083	51,76 75,76,74	: R4 = Sign[x(n-i)]#Sign[e(n)] # u
				ADDF3	#4R1, R4, R3	; R3 = wi(n) + R4
	Chen, Chein-Chung March, 1989	March, 1989		į		
				j į	order-3, RC	; Initialize repeat counter
************	"ter of Sache"	"adantito into		<u> </u>	SALIS MONTH IN BY	: Do 1 = 0, N-3
tan.			=		D 440144/117	The mext units
	5 2		=		20 DE	Cot the city of date
				200	51,78 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,	. Decide the sign of m
INITIALIZE PO	INITIALIZE POINTERS AND ARRAYS		SSURS	ADDF3	+AR1, R4, R3	; R3 = wi(n) + R4
				1		
. text			:	5 E	P2 = A01 + 1 (1)	; Det last data
	A series	e de la circulac buffer	=		-31 RA	. Set the gian of data
9	Pro addr	Set data mas		S	inout	. Delay branch
9	Em addr. ARO	Set pointer for x[]		20X	75.86.R4	. Decide the sign of u
5	Pen_addr, ARI	: Set pointer for w[]		ADDF3	**************************************	Compute wit-1(n+1)
5	0.0,00	0.0 = 0.0		STF	R3,+4R1++(1)Z	; Store last w(n+1)
RPTS			•			
STF	RO, ##RO++(1)%	; x(] = 0	+ DEF1N	DEFINE CONSTANTS	ęς	
:: STF	RO, #4R1++(1)%	0 = []# :	-		;	
5	æ.		¥	. usect	buffer, order	
Š	æ, æ	. R4 = B4	S	. usect	coeffs order	
Š	£.	3 = S2	in_addr	. usect	vars", 1	
Ē	Ein_addr, AR6	; Set pointer for input ports	out_addr	.usect	Vars	
5	Cout_addr, AR7	; Set pointer for output ports	Jppe-ux	.usect	vers .	
Imput:	;	•	1007-110		, , , ,	
	(H, 60)	i Input d(n)	,	nsect	Vers .	
5 8	**************************************	; Input x(n)	CINIC	5	. CINIT	
ż	19	; Insert x(n) to butter			OCONTON	
					COORDON	
COMPUTE FILL	CUMPUTE FILTER CUTPUT y(n)				ng/ma/yan	
1	8	8			5 1	
3	V.V,7k	1 KZ = 0.0		10 10	: 2	
HPYF3	+#80++(1)X,+#81++(1)X,R1	↔(1)I.Ri	pua.			

Appendix G1. Transversal Structure with Leaky LMS Algorithm Using the TMS320C25

Ui .usect 'parameters', J D96: .usect 'parameters', J	* FEFOR THE ALMPTINE FILTER ************************************	**************************************	+ ESTIMITE THE SIGNAL Y	LARP AR3	•	 S: ::	FIR DOTY CONTO.	MACD MAH-OF-GOOD, 4-	CMFD ; Configure BO as data memory	SACH Y Store the filter output		+ COMPUTE THE EDWOR		•	SACH EDR ; EDR(n) = D(n) - Y(n)		+ UPDATE THE WEIGHTS	LT EPR ; T = EPR(n)		ě	SAC ERROR : TRANSE : I + FRRIA.		AR1, ORDER-1	₩.	. A3,X#+1	••	ADAPT ZALR + AR3 . Lad ADAPT ZALR + AR3	IPYA + . BR2	+,LEAKY	SACH ++,0,ARI ; Store M(k,n+1)	734' + ' 1457Y 734G +	FINIS.			
.title 'T.Z'	N.25: Adaptive Filter Using Transversal Structure and Leaky-UNS Algoritha, Looped Code	Algorithms	S	y(n) = SLM w(k)+x(n-k) k=0,1,2,,63		$e(u) = d(u) - \lambda(u)$	m(k) m vin(k) + tine(n)tv(n-k) in 1 2 63		Where we use filter order $= 64$ and $m_0 = 0.01$.	Note: This source program is the generic version; I/O configuration has	not been set up. User has to modify the main routine for specific	application.	Initial condition:	1) PM status bit should be equal to 01.	2) SXM status bit should be set to 1.	3) The current IP (data memory page pointer) should be page 0.	4) Data memory UME should be 1. 5) Data memory U should be 327.		Chen, Chein-Chung February, 1989			DEFINE PROMETERS	** *** **** ****		Ober 1050		DEFINE ADDRESSES OF BUFFER AND COEFFICIENTS		· usect	.usect "coeffs", ONDER	RESERVE ADDRESSES FOR PARAMETERS		•	. usect	: .usect persmeters",1

Appendix G2. Transversal Structure with Leaky LMS Algorithm Using the TMS320C30

***************************************	***************************************	***************************************	==	ADDF3	R1,R2,R2	; y(n) = w[].x[]
+ TL30 - Adaptiv	Adaptive transversal filter using the TMS320C30	TL30 - Adaptive transversal filter with Leaky LMS algorithm using the TMS220C30		A00.	R1,R2	; include last result
			+ COMPUT	E EPROR S	(GNAL e(n) AND OUTPL	COMPUTE ERROR SIGNAL e(n) AND OUTPUT y(n) AND e(n) SIGNALS
* Algorithe.			•	*	18.187	: e(u) = q(u) - x(u)
E9 + 3(u) = STH	63 y(n) = SUM w(k)*x(n-k) k=0,1,2,,63	8,,	==	SIF	R2, +4R7 R7, ++4R7(1)	; Send out y(n) ; Send out e(n)
••	•		+ constr	(a) ~ (a) ~ (a)	(
• e(n) = d(n) - y(n)	n) - y(n)		*			
+ +	u(k) = relu(k) + ute(n)tx(n-k) k=0.1.263	FE 0.1.263				; R7 = e(n) bu/r . D1 = e(n) bubv(n) /r
				EVF3	•ARO++(1)Z,R7,R1	; R1 = e(n)=u=x(n-1)/r
there we	use filter order = 64	Where we use filter order $= 64$, $r = 0.995$ and $mu = 0.01$.	==	ADDF3	**R1,R1,R2	; R2 = u0(n) + e(n) +u+x(n)/r
• •	Chen, Chein-Chung March, 1989	March, 1989		3 E	order 4, R.	; Initialize repeat counter ; Do i = 0, N-4
*	+		=	PPYF3	**RZ, RZ, R0	$\frac{1}{2}R0 = rawi(n) + e(n) +$
. Copy	"adapfltr.int"		:: ¥	E CE	##R0#+(1), R1, R2	; KZ = W1+1(n) + e(n)*W*X(n-1-1)/r • R1 = o(n)*W*X(n-i-2)/r
***************************************	*****************************		== }	SIF	RO, +4R1 ++(1)%	; store wi(n+1)
+ PERFORM ADA	PERFORM ADAPTIVE FILTER			HPYF3	*AR2, R2, R0	; R0 = regh-3(n) + e(n) eufx(n-N+3)
order .set	order set 64		==	ADDF3	++4R1(1),R1,R2	; R2 = mM-2(n) + e(n)=uex(n-M+2)/r . D1 = e(e)=uex(n-M+1)/r
<u>~</u>		; mu / leaky	==	ST	RO. #4R1++(1)%	: Store and-3(n+1)
leaky .set	0.995			2	input	; Delay branch
•	STANDARY CALL SETTINGS THE SETTING			HPYF3	#AR2, R2, R0	; RO = r4u1(n) + e(n)4u6x(n-N+2)
• INI IME 12E F	UIRIERS HAD HERWITS		==	1	**************************************	$\frac{1}{2}$ R2 = uN-1(n) + e(n)4u4x(n-N+1)/r
. text			Ξ:	Ē	BO #481++(1)2	; NO = FWB1(n) + e(n)*U*X(n-N*1) . Store wM-2(n+1)
begin .set	-		=	, #S	RO. #4R1 ++ (1) Z	. Update last w
9	order, BK	; Set up circular buffer				
<u> </u>	Da_addra	s Set data page	# 1551KE	DEFINE CONSTANTS	"	
3	Exn. addr. PRO	s Set pointer for XLJ	•			
3	Esm400F, PK1	Cat pointer for BLJ	Ę,	.usect	"buffer", order	
i 🖰	0.0.0	. BO = 0.0	S	.usect	"coeffs", order	
RPTS		-	10-20dr		, 4619 , I	
STF			yn addr	100	vers.	
:: STF	RO, #4R1 ++(1)X		m addr	usect.	Vars	
5	Pin_addr, AR6	Set pointer for input ports	5	.usect	wars , 1	
5	Cout_addr, AR7	; Set peinter for output ports		.usect	vers , 1	
Input:	70 70V*	(a) Aura (a)	r_addr	.usect	vers , 1	
:: ::	##886(1) R6	· Input x(n)	cinit	· sect	.cinit	
: #5	R6, #4R0	; Insert x(n) to buffer			7, in addr	
•					0804002	
+ COMPUTE FIL.	COMPUTE FILTER OUTPUT y(n)			2	Ę	
•	8	8		Pod.	•	
•	0.0,62	: KZ = 0.0		float	au_leaky	
HPYE3	3 +480++(1)Z,+681++(1)Z,R1	++(1)1.R1		. float	leaky	
RPTS					-	
HPYF	3 +4RO++(1)I, +4R1++(1)I,R1	++(1)X,R1				

Appendix H1. Assembly Subroutine of Transversal Structure with LMS Algorithm Using the TMS320C25

tithe /BLMS/			
	25	-	: Set current register
	·		: Save register ARI
BLTS: HOSprive Filter subroutine using fransversal Structure	3 5	•	; Save register ARZ
* and LMS Algorithm, Looped Code	3 5	MC3,SAME3	: Save register AR3
•	8.00		. Configure BO as program memory
* Algoriths:	X.a	٥	. Clear the P cesister
•	<u> </u>		interface of the state of
3	5 3		: USING FOUNDING
			; Point to the oldest sample
# y(n) = (Jun w(x) + x(n - x) x=0,1,2,,n-1	FIR APTR	_	; Repeat M times
₽	MACD	10 MH+0Fd00h, +-	; Estimate Y(n)
-		e	; Configure BO as data memory
\bullet	APAC	ڀ	
	HOWS.	>	; Store the filter output
+ u(k) = u(k) + ute(n)tx(n-k) k=0,1,2,,N-1	•		
	* COMPUTE THE EPROR	E ERROR	
* Where we use filter order = N	•		
-	99		; ACC = - Y(n)
 Note: This subroutine performs Adaptive Filter using the LMS Algorithm. 	ADDH	-	
There are some initial conditions to meet before calling it.	35		; ERR(a) = D(n) - Y(n)
	*		
Initial Conditions:	, t Urbaile intermetionis	E CHIS	
1) Date memory over Should be equal to 1.			
2) Data memory U should be equal to MU (RIS format).	- i	Š :	: 1 = EM(n)
3) Fill status bit should be equal to 01.			; P = U + EMR(n)
4) SIM status bit should be set to logic 1.	£ !		
3) UVM status bit should be set to 1.	3		; round the result
b the current DP (data memory page pointer) should be page 0.	3 5	188 5	; ENGF = U + ENR(a)
COD at [[im material succitions the new and the code of the code o		X 481 000FB-1	. Set Campan
The section of the se			Delich to the
2) MAI MAG DEED USED IN This subfourthe.	5		Point to the Coefficients
	5 !		. Point to the data sample
Chen, Chein-Chung February, 1989	- i		; I register = U + ERR(n)
			: F = U + ERK(B) + X(B-K)
			; Load ACCH with A(k,n) & round
	•	Z#.	; W(K,n+1) = M(K,n) + P
EFINE AND REFER SYMBOLS	•		: P = U = EM(n) = X(n-k)
	SACH		; Store W(k, n+1)
.global LHS,0R0ER,U,D,0ME,Y,ERR,XM,MN	2	Z ADMPT,+-,AR2	
	•		
PESCINE ADDRESS FOR PARAMETER	5	_	Restore register ARI
	5		; Restore register ARZ
.usect	* 5	ARS, SAVE3	; Restore register AR3
.usect	•		
SAVE3: .usect "parameters",1	FINISH RET		
ERRF: .usect "parameters",1	*		
***************************************	· end	•	
FERFORM THE ADAPTIVE FILTER			

F ESTIMATE THE SIGNAL Y			

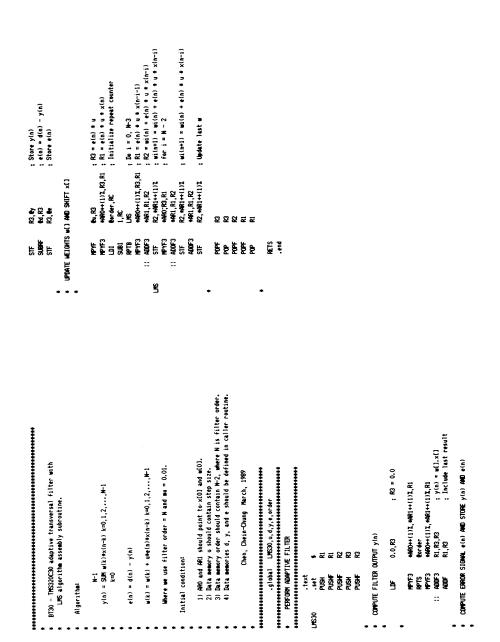
Appendix H2. Linker Command File for Assembly Main Program Calling a TMS320C25 Adaptive LMS Transversal Filter Subroutine

/+ ALTHALOYS - COMPAND FILE FOR LINKING A THS320C25 ASSENBLY PROGRAM	OR LINKING	A THS320C2	S ASSEMBLY	PROGRAM	•
	1989 Tex	as Instrume	nts Incorpo	rated	•
					•
/* Usage: dspink <obj -o="" <out="" file="" files?=""> -m Cmap file> c.cmd</obj>	les) -0	Cout file		the c.cm	•
					•
* Description: This file is a sample command file that can be used	sample co	mand file	that can be	e used	•
for linking the TMS320C25 assembly programs; use it	€ TMS320C2	5 assembly	programs; '	15e 1t 45 4	•
quideline. You may want to change the allocation	Ey Ent	to change t	he allocati	ue.1	•
is scheme according to the size of the program and the	ng to the	size of the	program 4	a the	•
(a memory configuration of your TMS320C25.	ration of	your TMS320	629		•
					•
** Notes: NEMORY SPECIFICATION	CATION				•
					•
in Block BO 15 configured as data memory (CMFD) and	nfigured a	s data men	ry (DED)	pue	÷
	croprocess	or mode).	La Benory	locations	•
6h SFh and 80h IFFh are not configured.	AIFFA ar	e not conf	gur ed.		•
`					ì
FENDRY					
PAGE 0 : Ints : origin Ext.Prog : origin	. 020 	length = 020h length = 0FEE0h	020h	/* Program */	•
PAGE 1: Regs : origin =	ŕ		\$	/e Date	•
Block_B2: origin =		length =	020h		
Int_ROM : origin	•	length =	0100h	æ •	•
Int_RAM! : origin	n = 0300h.	length =	01000	/* B1	-
1161 io - \$180" 173			3		
					7
/* SECTIONS ALLOCATION					•
/e- SECT10NS					Ì
vectors : () . Ints	PAGE	/* Inter	Interrupt vector table	table	•
0	PAG	/* Code			•
ters: ()	2 PAGE 1	/e Parameters	ters		•
ô	PAGE	/* Bleck BO	28		•
000			16		•

Appendix H3. TMS320C30 Adaptive Filter Initialization Program

	MO,RI do_init *ARO++,AR	UDI mAMON+, NO ; Get next first word SUBI 1, RI ; Count - 1 dons: RR begin										
.uidh 122	for TMSS20C30 adaptive	ing actions: ss the system stack. tition, which copies section to DAIN RAM.	; Size of system stack ; frame pointer	RESET	3118	; Address of stack ; Address of init tables : Midress of init tables : Midress of init tables : Midress of init tables		; Get page of stored address ; Load the address into SP ; And into FP tee		; Oct page of stored address ; Get address of init tables ; If NOM model, skip init	; Get first count If O, nothing to de ; Get dest address ; Get first word ; Count - 1	
.width 132	This is the initial beet reutine for TMSS20C30 adaptive filter Programs.	This module performs the fellowing actions: 1) Allocates and initializes the system stack. 2) Performs auto-initialization, which copies section 1, const* data from APM to DATA APM. 3) Prepare to start the user's assembly pregram.	STACK.SIE .set 4Ch ; Size of system stack FP .set AR3 ; Frame pointer *	FEST	.usect ".stack",STACK_SIZE .text	stack_addr .word stack ; Address of stack init_addr .word cinit ; Address of init table ***********************************	SET UP THE INITIAL STACK POINTER	LDP stack_addr LDI estack_addr,SP LDI SP,FP	* DO AUTOINITIALIZATION	LDP init_addr LDI @init_addr,ARO CMPI -1,ARO BEQ done	LDI +480++,RI BZD done LDI +480++,481 LDI +480++,RO SJBI 1,RI	o State
Ĭ			STACK_SIZE .set	RESET • ALLOCAT • . text T	stack •	stack_addr .word init_addr .word	90 T3C		+ DO AUTO			do_init:

Appendix H4. Assembly Subroutine of Transversal Structure with LMS Algorithm Using the TMS320C30



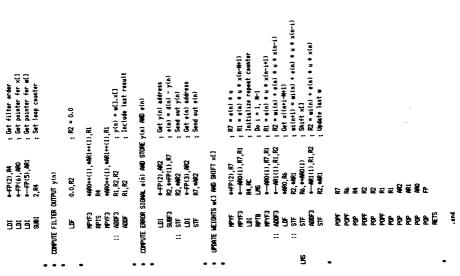
Appendix H5. Linker Command/file for Assembly Main Program Calling the TMS320C30 Adaptive LMS Transversal Filter Subroutine

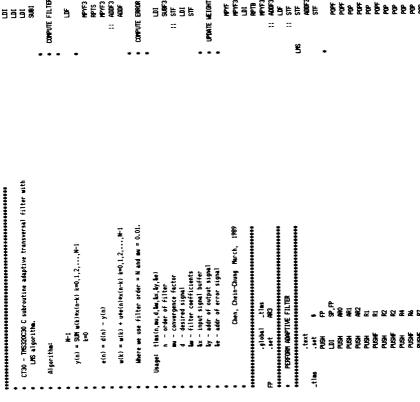
1 10 10 10 10 10 10 10	FROGRAMS I Ind 20 (Au) filter) -e cout riles Into filte a sende command filte Into filte a sende command filte All the adeltive riler program All the adeltive riler program All the EDING. ass section rile That the EDING. ass section r	Modes in this cas filter. The case filts addition to the case filts addition to the case filts addition to the case filts addition that site is a sample comment filter based for the case of the case	***********
10 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	30 (30) (1611.) - 5 \(\) (20) (161) (1611.) - 5 \(\) (161)	and file a das file adda.cod and file this can be used for stoods free this can be for stoods free this with the file stoods free to link with the file stoods free to link with the file stoods free this a single page. for file this a single page. for sail free this a single page.	* * * * * * * * * * * * * * * * * * * *
Description	So day filter. See State	and fits -s day fits address mand fits that can be used casedly respens regions in the that can be used settle many meets I may the full many meets I may the full many meets I may the full many meets I may full many meets I may full many meets I may full many meets full fits units a small page. It wall be small be the defined model res	* * * * * * * * * * * * *
Marciptima Tay 1978 Marciptima Tay 1978 Marciptima Control C	is the is a simple or limiting admitter that the admitter strice. The factors strice of the factors and the most strice has see a strice hist, were an effect of cross any der the cross and cro	mand file that can be used for expensively responsive to expensive to limb with the the arts_natialization. The arts_natialization is the arts_natialization arts_nation arts_nation arts_nation arts_nation_	* * * * * * * * * * * * *
1	This is a sample or in the sample of the adaptive filter Pilet Sample or interpretable of the adaptive to the adaptive to the BETIME, has see sample of the BETIME, has see sample of the Cross sample of the	manned file that can be used cassedly frequent requests where it into might the automatical states of the fault into a small page. The smaller than 64 meets and under ret.	• • • • • • • • • •
March 1997 (C) 1993 (the adaptive fill thing adaptive fill the fill things and the fill the do min using the mail (de min using the mail) (de min using the min u	for except forgets. The organization is the first and and the first and and and the first and	• • • • • • • • •
10 10 10 10 10 10 10 10	PINITASM files to de montage he man and man an	respirate to this sith the attachment of att	
Mark () 1997 () 1998 () 1998 () 1999 (PINIT.ASM file to do in using the small (de in using the small (de in using the small (de int cross any dat it not cross any dat it no cross a	the arts_natialization. Gailt manay model, be sure from offs units, a single page. It has not be n	• • • • • • •
100 - 100 -	in using the small (define Lass sec. arts and the DRIME Lass sec. arts and the DRIME Lass sec. arts and the Cross say 64% and the Cr	dell) memory model, be sure than fit author a single page. If be smaller than odd words and smaller than odd words and smaller than odd words and	
10	in using the small (de saftley thats, wars and saftley that, wars any GAK to the cross any GAK to the cross any GAK to the coxed the coxed the coxed	continue and the continue and c	
101 101 101 101 101 101 101 101 101 101	of the Diffile bis sec satisfy this, vars and that cross any dat be removed the constant En HENGRY MAP of ten = 0xc0	tion fils within a single page. It be smaller than off words and worder ret.	
1	satisfy this, wars and t not cross any 64k b ####################################	of be sailer than off, words and under res.	
PROFILE TO SECTION OF	th not cross any 648 b	Muldirite.	
9. 976/177 The SYSTI	EN FENGRY 149P +/- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1		• •
FORCET THE SYSTEM OF SYSTE	FENORY RE		
HERE OF BEACH OF SECULAR OF SECU	= =		
ROII: 09 = 0.4 STADO: 09 = 0.6 STADO: 09 = 0.6 SPECIFY THE SCITI	. 5		
Ramon org = 0.4 5.75(C): org = 0.4 6.94(E): org = 0			
. STACK: org = 0.00 . WARS: org = 0.00 . WARS: org = 0.00 . SPECIFY THE SCETT . CETTORS . VECTORS: 0.00 . CERT: 0.00 . CE	200 100	0 100 14 100 15	-
MANIT STORE OF BOAR STORES OF BOAR STORES OF BOAR STORES OF BOAR STORES OF STORES		A Curtan stand	•
**************************************		in plants and the	•
SPECIFY THE SCC11 ECTIONS	5	/* Mass block : 40% - 40 of Syr	• •
* SPECIFY THE SCETI ECTIONS vectors: (0 > VEC * * * * * * * * * * * * * * * * * * *			
ections	/* SPECIFY THE SCOTIONS ALLOCATION INTO HENCRY	ENORY +/	
vectors: 0 > .text: 0 > .contr. 0 > .			
00	•	Interrupt vectors	•
c	•/	Code	•
	•	Instiguization tables	•
\wedge	_STACK /+ S	System stack	•
ô	•	for variables	•
	*	for data buffer	ৃ
coeffs: () > ROM1	•	for filter coefficients	•
cains aliqu(32) : () > Romi		for lattice filter onion	

Appendix I1. C Subroutine of Transversal Structure with LMS Algorithm Using the TMS320C25

CXEFFP: equ 04f00h	***************************************	+ PERSON THE ADAPTIVE FILTER	-	* SAVE THE VALUES OF THE REGISTERS	***	Taks: IAW SANET	35	SAR AR3, SAVE3	SAR ARA, SANEA	ST DSTO	197 172	* OET THE ADAPTIVE FILTER PARAMETERS		Spr : Set P register shift more	••	••	•	MAR 4- ; Set pointer for getting papameter	N = 300 : → 3V7	-			1	_	1		LPLK AR3, FRSTAP	SACL + Insert newest sample		+ ESTIMATE THE STOWNER T	Canfigure BO as program memory		· ·	, 13 MP3 ATRI CT	9000	STATE OF THE PARTY	יייייייייייייייייייייייייייייייייייייי	••	,	SACH Y Stone the Filter output			MEG : 40C = - Y(n)	The state of the s
title 'QUS'	CLMS: Adaptive Filter C subroutine using Transversal Structure	and LMS Algorithm, Looped Code				y(n) = 2018 @(k) + x(n-k) k=0,1,4,,n-1		(u) = q(u) = x(u) .		$\mathbf{e}(\mathbf{k}) = \mathbf{e}(\mathbf{k}) + \mathbf{u} + \mathbf{e}(\mathbf{n}) + \mathbf{x} + \mathbf{n} + \mathbf{k} = 0, 1, 2, \dots, N-1$	blace we use filter order = N		+ Usage: Ins(n,mu,d,x,&y,&e)		* mu - convergence factor	* d - desired signal	* - Input signal	Ly - adds of output signal	and the state of t		* Note: Data memory 0200h 0200h+N-1 & 0300h 0300h+N-1 are reserved.	. Chen. Chein-Chung February, 1989		######################################	, def	A DESCRAIR ANDRESSES FOR PARAMETERS		noto: "sarameters".1	. usect	.usect	.usect	.usect	.usect	. usect	X: "usect "parameters",1	D: .usect "parameters",1	Us .usect "parameters", 1	y: .usect "parameters",1	n. usect	.usect	: .usect	*	+ DEFINE ADDRESSES OF BUFFER AND COEFFICIENTSS	

Appendix I2. C Subroutine of Transversal Structure with LMS Algorithm Using the TMS320C30





CET FILTER PARAMETERS