TMS320 DSP
DESIGNER'S NOTEBOOK

Using the Circular Buffers on the TMS320C5x

APPLICATION BRIEF: SPRA264

Thomas G. Horner
Digital Signal Processing Products
Semiconductor Group

Texas Instruments
October 1995

TEXAS INSTRUMENTS

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.

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

Contents

Figures

Examples

Using the Circular Buffers on the TMS320C5x

Abstract

The TMS320C5x family has special memory-mapped registers and associated circuitry for two circular buffers. One of the eight auxiliary registers is used as a pointer into the circular buffer. Two memory-mapped registers are associated with each circular buffer that need to be initialized with the start and end address.

This document discusses how to use the circular buffers on the TMS320C5x to implement an input sample delay buffer for filtering, FFTs, and control system transfer function calculations.

Design Problem

How do I use the circular buffers on the TMS320C5x to implement an input sample delay buffer for filtering, FFTs, and control system transfer function calculations?

Solution

The TMS320C5x family has special memory-mapped registers and associated circuitry for two circular buffers. One of the eight auxiliary registers is used as a pointer into the circular buffer. Two memory-mapped registers are associated with each circular buffer that need to be initialized with the start and end addresses.

Once the circular buffers are initialized, the question becomes: How do I use these circular buffers in a filter, etc. calculation? Two different scenarios will be outlined in this Designer's Notebook page.

1) The first is a standard filter implementation where the input array and coefficient array are the same size and only a single output is kept.

2) In the second, there is pure delay in the system as well as a filter, transfer function, etc. This type of system can have input or output arrays that are larger than the coefficient arrays.

Examples of these types of systems are telephone networks, remote control systems, etc. A block diagram of the system and a time response is shown in Figure 1,
where:
t_t = Total time delay through the system
t_h = Time delay through filter, etc.
t_1 = Time delay at system input
t_2 = Time delay at system output
F_s = Sample rate

In standard filter implementations, $t_1 = t_2 = 0$, which makes the input sample and coefficient arrays equal length. This is the most straight forward use of a circular buffer. Only a single circular buffer is required for this type of problem. It is used as the pointer to the input array which changes with time. The equation to be solved is of the form:

```
y(n) = a(0)*x(n) + a(1)*x(n-1) + ... + a(N-1)*x(n-N+1)
```
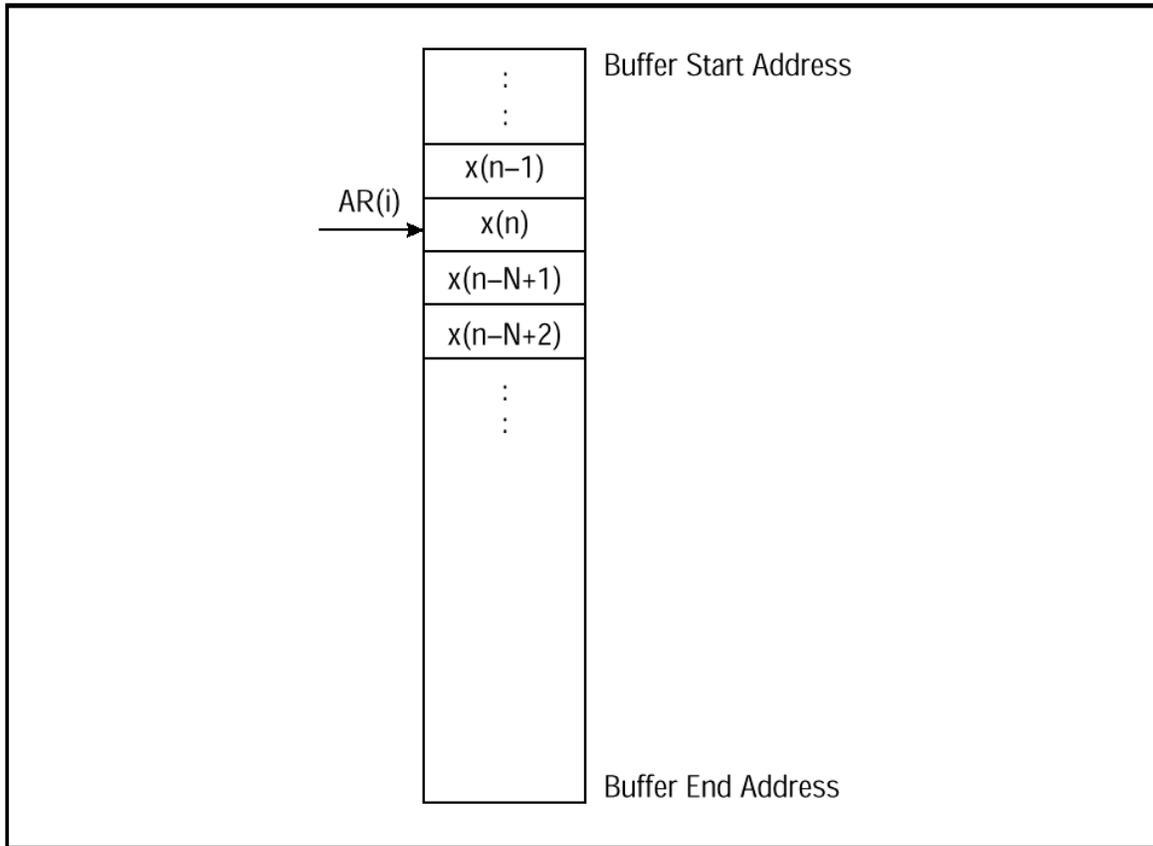
where $N = t_h * F_s$

Figure 1. System Block Diagram
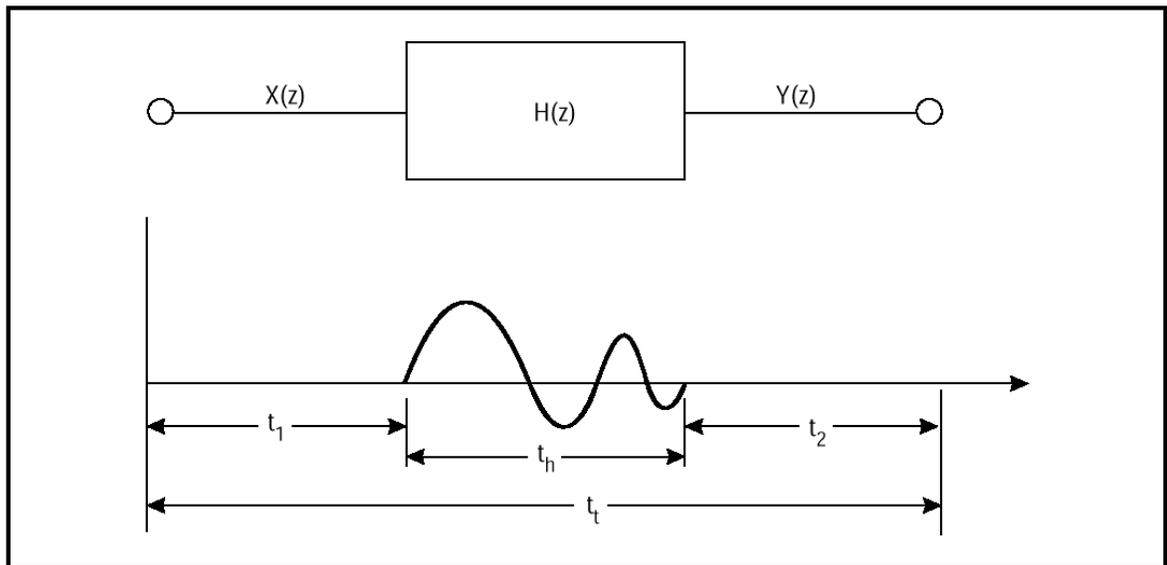
At any given time the input sample circular buffer would look like the following figure.

Figure 2. Input Sample Circular Buffer

Before the circular buffers can be used, they need to be set up. This is done as part of the initialization routine. Once the circular buffer is setup, it can be used in filter calculations. At the beginning of each sample period, a new sample will be read into the circular buffer, overwriting the oldest sample. The newest sample x(n) will be stored at the memory location pointed at by auxiliary register AR(i). Then the filter calculation will be performed and the pointer adjusted for the next new input sample. An example of this code is shown below.

Note that the example calculation starts at the oldest sample, x(n–N+1), and proceeds to the newest sample, x(n). As a result, the coefficients need to be accessed in reverse order in the array (aN–1, aN–2, ...). Using the MAC command requires that the coefficients be stored in reverse order. This is because the PreFetch Counter (PFC) is used to point to the coefficients stored in PROGRAM memory space and the PFC only has an increment function.

Example 1. Code Listing

```
;———————— INITIALIZE CIRCULAR BUFFERS
splk   #0feh,CBCR  ;Initialize circular buffer #1
                   ; CB#1 pointer = AR6

lacc   #XRAM       ;Initialize buffer start address
samm   CBSR1       ; CB#1 Start Register
samm   AR6         ;Initialize AR6 to start address
add    #Th-1       ;Initialize buffer end address
samm   CBER1       ; CB#1 End Register
   :   :
   :   :


;———————— N TAP FILTER CALCULATION ISR FILTER
ldp    #0          ;Setup DP and ARP
mar    *,AR6

lacc   output      ;Read latest output from memory
samm   DXR         ; and write to serial port
lamm   DRR         ;Read latest input from serial port
sacl   *+      ; store to memory and increment pointer

rptz   Th          ;Compute filter response
mac    COEF,*+     ; Use Th for extra APAC + pointer
                   ; realignment
sach   output      ;Write output to memory

rete               ;Return to MAIN w/ interrupt enable
```

A more complicated system results when there is some pure delay in the system (i.e., where t_1 or t_2 do not equal zero). An example of this would be a telephone 2:4 wire hybrid simulation which could be used to develop echo cancellation applications. There is a delay from the time the signal enters the telephone line to the time it reaches the hybrid (t_1). Then the hybrid acts on the signal reflecting part of the energy back towards the sender (t_h). Finally, there is the pure delay of the signal traveling back to the sender from the hybrid (t_2).

In this case there is pure delay before and after the filter transfer function. It would seem to be necessary to setup coefficient, input, and output buffers long enough to cover the entire time period:

$$N = (t_1+t_h+t_2) * F_s$$

However, with a little planning, it is possible to save $(t_1+t_2)*F_s$ memory locations for the coefficient array, t_2*F_s memory locations for the input buffer, and t_1*F_s memory locations for the output buffer.

To reduce the requirements for the coefficient array, you need to take into account the fact that, during time periods t_1 and t_2, there is no modification to the signal's frequency content or magnitude. It is only during time period "t_h" that the 68-4 signal is affected by the system. If a coefficient array were to be setup to implement an N tap filter of the form:

```
y(n) = a(0)*x(n) + a(1)*x(n-1) + ... + a(N-1)*x(n-N+1)
```

covering the time period $t_t = t_1 + t_h + t_2$
      with:

            $N = (t_1 + t_h + t_2) * Fs = N_1 + N_h + N_2$ and,
            $N_1 = t_1 * F_s$
            $N_h = t_h * F_s$
            $N_2 = t_2 * F_s$
      then,
            $a(0)$ to $a(N1-1) = 0$ and,
            $a(N_1 + N_h)$ to $a(N-1) = 0$

Using this fact, the difference equation can be simplified to:

```
y(n) = a(N1)*x(n-N1) + a(N1+1)*x(n-N1-1) + ... +

a(N1+Nh-1)*x(n-N1-Nh+1)
```

which saves N1+N2 memory locations for coefficients plus N1+N2 multiplies and adds for the filter calculation. Since coefficients a(N1+Nh) to a(N−1) are equal to zero, then corresponding input samples x(n−N1−Nh) to x(n−N1+1) don't need to be stored in the input sample buffer because they will not be used in the filter calculation. This saves N2 memory locations in the input sample buffer. Input samples from x(n) to x(n−N1+1) are required, however, to provide the pure delay during time period t_1. The output buffer needs $t_h * F_s$ locations to represent the delay through the filter and an additional $t_2 * F_s$ locations for the pure delay during time period t_2. There can never be any system response during time period t_1 because the signal has not reached the filter yet, so it is not necessay to save outputs for this time period. This saves $t_1 * F_s$ memory locations for the ouptput array.

A key element in being able to take advantage of these memory savings is correctly maintaining the pointers to the input and output buffers. It is these pointers which will allow the time delays to be represented correctly. In both buffers, the newest and oldest values are in consecutive memory locations, with the oldest following the newest. When a new sample period begins, the oldest output value is read out of the memory and sent to an I/O port. The output buffer is now ready to receive a new computed filter response which will replace the oldest value that is no longer needed. Once the new output value is stored to memory, the pointer will be incremented to point at the oldest value to be used in the next sample period.

The procedure for the input buffer is slightly different. When a new sample period begins, the newest sample is stored at the pointer location and then the pointer is incremented to point to the oldest remaining sample in the buffer. It is the oldest t_h*F_s samples that will be used in the filter calculation, starting with the oldest sample. As a result, the coefficients need to be stored in reverse order as in the simple filter example. In addition, the input buffer pointer must be restored to its beginning value because the pointer does not wrap all the way back around to the beginning of the buffer during the calculation as it does in the standard filter implementation. This is because the first t_1*F_s samples are not used to compute the filter response.

The picture below shows a long buffer (no memory savings) plus the memory reduced input, x(n); output, y(n); and coefficient, a(n) arrays with the corresponding pointers. An example of how to implement this technique in software follows:

Figure 3. Software Implementation
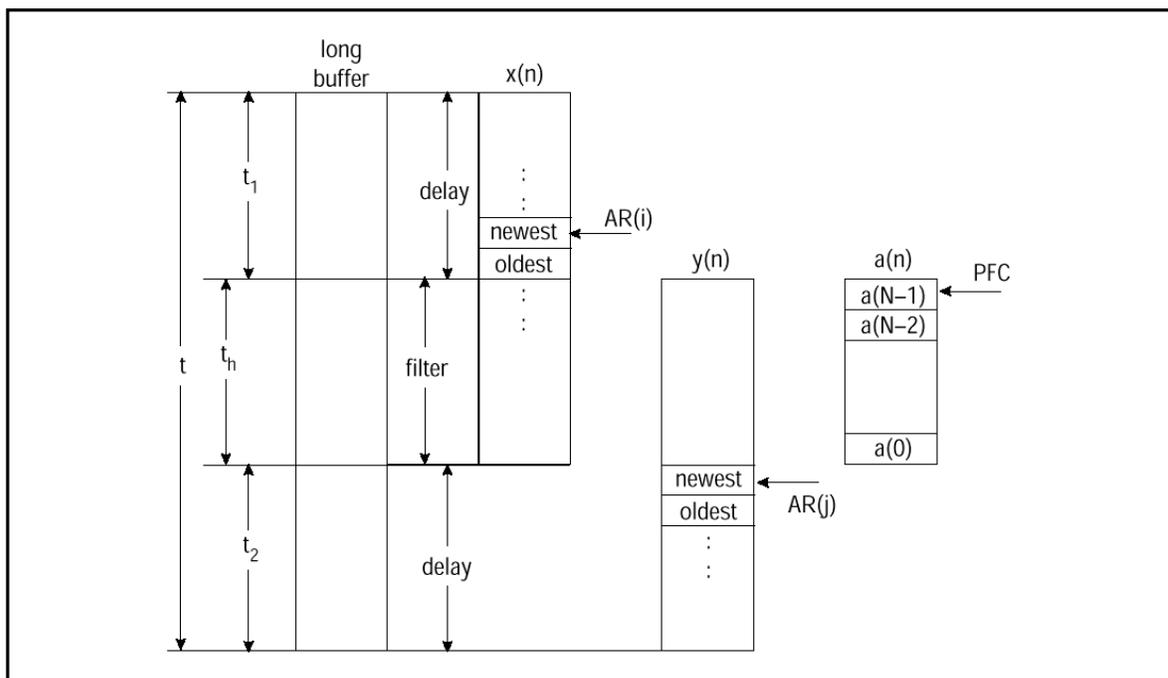
Example 2. Software Implementation Code

```
;————— INITIALIZE CIRCULAR BUFFERS
;Initialize circular buffers
   splk           #0feh,CBCR     ; CB#1 pointer = AR6
                                 ; CB#2 pointer = AR7


;Initialize CB#1 = Input data
   lacc           #XRAM          ;Buffer start address
```

```
    samm            CBSR1               ; Initiailize Start register
    samm            AR6                 ;Initialize AR6 to CB#1 start address
    add             #(T1+Th-1)          ;Compute end address
    samm            CBER1               ; Initialize End register

;Initialize CB#2 = Output data
    lacc            #YRAM               ;Buffer start address
    samm            CBSR2               ; Initialize Start register
    samm            AR7                 ;Initialize AR7 to CB#2 start address
    add             #(Th+T2-1)          ;Compute end address
    samm            CBER2               ; Initialize End register
        :               :
        :               :

;——————- N TAP FILTER CALCULATION W/ DELAY ISR FILTER
    ldp             #0                  ;Setup DP and ARP
    mar             *,AR7

    lacc            *,AR6               ;Read delayed output from CB#2
    samm            DXR                 ; and output to serial port
    lamm            DRR                 ;Read latest input from serial port
    sacl            *+,4                ; and store to CB#1 (XRAM array)
    smmr            AR6,#cb1            ;Store start of x(n) array address

    rptz            Th                  ;Compute filter response
    mac             COEF,*+             ; Use Th (N+1) to get extra APAC
    mar             *,AR7               ;Store result in y(n) array in CB#2
    sach            *+,AR6              ; and increment pointer

    lmmr            AR6,#cb1            ;Restore x(n) array start address.
    rete                                ;Return to MAIN w/ interrupt enable
```