# *Interfacing the TLV1544 and TLV1548 A/D Converters to Digital Processors*

Author: Heinz-Peter Beckemeyer
Nicholas Holland
Richard Oed

Literature Number: SLAA022
Date: December, 97

## 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 applications 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.

# Contents

## List of Figures



## Software Listings



## Tables

# *Interfacing the TLV1544 and the TLV1548 A/D Converters to Digital Processors*

**ABSTRACT**

This Application Report describes the hardware and software requirements for interfacing an A/D converter to a DSP and to a MCU. The 10-bit A/D converter TLV1544 (4 analog input channels) and the TLV1548 (8 analog input channels) from Texas Instruments have been used to develop such interface. Example software code has been written showing how to program the DSP and the MCU to control the A/D converter and to acquire samples. This is shown and explained methodically in the Application Report.

## 1. Introduction

As we enter the Digital Age, more and more measurement and control processes are using Digital Signal Processors and Microcontrollers to manage entire systems. However, all the variables in the "real world" which sensors are used to measure are analog in their physical nature. Such signals could come from light, temperature and pressure sensors etc.

Before these signals can be treated, they must first be converted from analog into digital data format so that the digital systems can acquire the equivalent analog values and modify them appropriately.

The following (Figure 1) shows a typical block diagram for a digital system, using an ADC to acquire the analog signal and convert it into digital form. The DSP or MCU manipulates the information into the desired form and then sends the result to the DAC to provide an analog signal, which may be used for control or observation purposes.

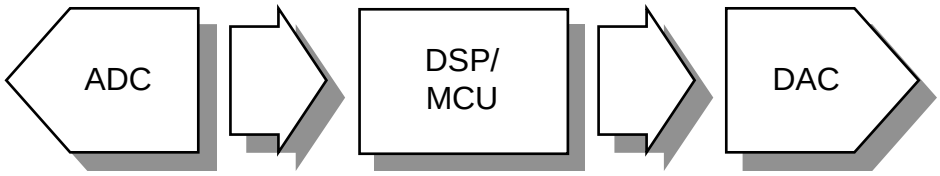


***Figure 1: Typical Digital System Interface***

This Application report is broken into two sections describing the method used to interface the Texas Instruments 10-bit TLV1544 and TLV1548 Analog-to-Digital converters to the :

- Texas Instruments TMS320C542 Digital Signal Processor.
- Motorola MC68B11E9 Microcontroller.

Throughout this report, the process of interfacing the ADC will be methodically explained by means of circuit diagrams, timing diagrams and by descriptions of the assembly code used to control them.

# 2. The Analog-to-Digital Converter

The ADC used in this Application Report is the 10-bit TLV1544 from Texas Instruments. The TLV1544 and the TLV1548 are almost identical except for the fact that the TLV1548 has eight analog inputs rather than four.

## 2.1 Introducing the TLV1544/8

The TLV1544/8 is a CMOS 10-bit switched-capacitor successive-approximation (SAR) analog-to-digital (A/D) converter. This device has an on-chip seven/eleven channel multiplexer that can select any of four/eight analog inputs or any of three test voltages. These test voltages can be used to check that the ADC is working correctly and that data can be send to and from the converter. The three test voltages provide a 000h, 200h or a 3FFh result.

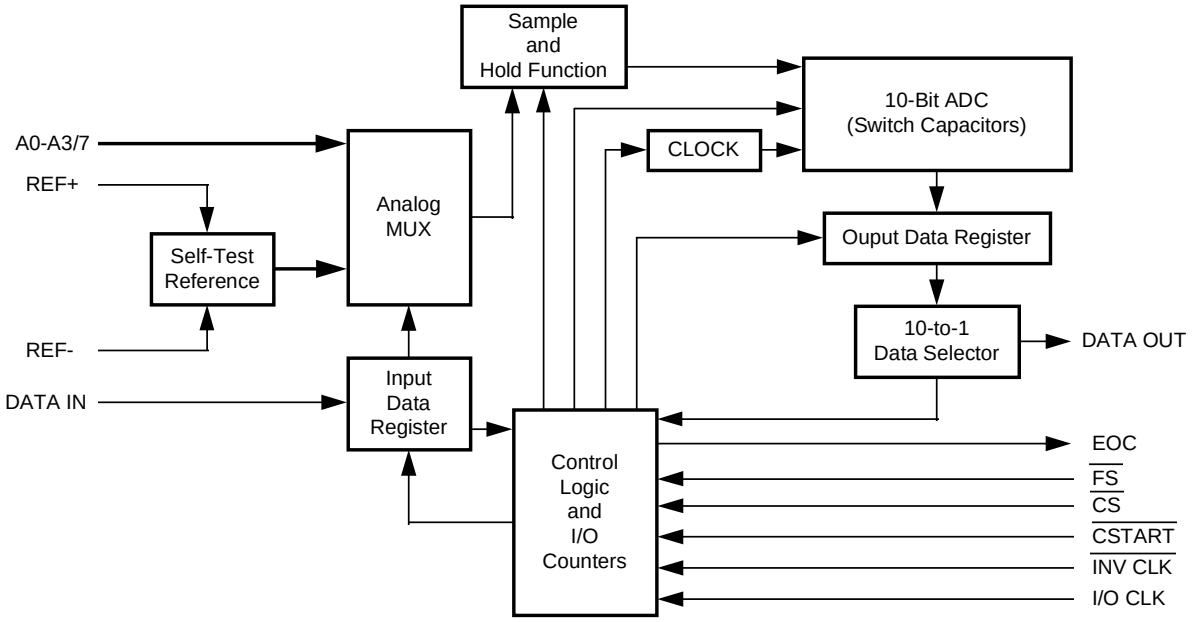


***Figure 2: TLV1544/8 Functional Block Diagram***

## 2.2 Serial Port Interface Pin Description

In order to connect the A/D converter to a DSP or Microcontroller, a brief description of the main serial port signals will be given.

### *2.2.1 Chip Select*

A high-to-low transition on $\overline{CS}$ resets the internal counters and controls and enables DATA IN, DATA OUT and I/O CLK. A low-to-high transition disables DATA IN, DATA OUT and I/O CLK.

### 2.2.2 I/O CLK

This is the input/output clock signal and is required to send the converted values back to the digital processor and receive the software-programmable operation mode data from the digital processor via serial port on the A/D converter.

### 2.2.3 DATA IN

The 4-bit serial data selects the desired analog input or test voltage to be converted next in a normal cycle. These bits can also set the conversion rate, the channels, the test voltages and enable the power-down mode.

After the four input data bits have been read into the input data register, DATA IN is ignored for the remainder of the current conversion period.

### 2.2.4 DATA OUT

Three-state serial output of the A/D conversion result. DATA OUT is in the high-impedance state when $\overline{CS}$ is high and active when $\overline{CS}$ is low. With a valid $\overline{CS}$ signal, DATA OUT is removed from the high-impedance state and is driven to the logic level corresponding to the MSB or LSB value of the previous conversion result.

### 2.2.5 FS

DSP frame synchronization input. FS indicates the start of a serial data frame into and out of the device.

### 2.2.6 EOC

End of conversion flag. This signal goes low once the last conversion has been read out of the A/D converter and remains low until the next conversion is complete. Once the new data is ready for transfer it returns to logic high. EOC can also indicate that the converter is busy.

## 2.3 Features of the TLV1544/8

All the features that can be software-programmed are listed below in Table 1. These values are sent to the A/D converter via the DATA IN signal.

| FUNCTION SELECT | INPUT DATA BYTE | | COMMENT |
|---|---|---|---|
| | BINARY | HEX | |
| Analog channel A0 for TLV1548 selected | 0000b | 0h | Channel 0 for TLV1544 |
| Analog channel A1 for TLV1548 selected | 0001b | 1h | |
| Analog channel A2 for TLV1548 selected | 0010b | 2h | Channel 1 for TLV1544 |
| Analog channel A3 for TLV1548 selected | 0011b | 3h | |
| Analog channel A4 for TLV1548 selected | 0100b | 4h | Channel 2 for TLV1544 |
| Analog channel A5 for TLV1548 selected | 0101b | 5h | |
| Analog channel A6 for TLV1548 selected | 0110b | 6h | Channel 3 for TLV1544 |
| Analog channel A7 for TLV1548 selected | 0111b | 7h | |
| Software power down set | 1000b | 8h | No conversion result (cleared by any access) |
| Fast conversion rate (10 ms) | 1001b | 9h | No conversion result (cleared by setting to fast) |
| Slow conversion rate (40 ms) | 1010b | Ah | No conversion result (cleared by setting to slow) |
| Self-test voltage | 1011b | Bh | Output result = 200h |
| Self-test voltage | 1100b | Ch | Output result = 000h |
| Self-test voltage | 1101b | Dh | Output result = 3FFh |
| Reserved | 1110b | Eh | No conversion result |
| Reserved | 1111b | Fh | No conversion result |

***Table 1: Software-Programmed Operation Modes Values***

Now that the TLV1544/8 has been described, the next sections of this report deal with the way in which it is possible to interface the converter to a DSP or Microcontroller.

# 3. The ADC to TMS320C542 DSP Interface

The TLV1544/8 was interfaced to the 16-bit Fixed-Point DSP TMS320C542. It has one Time-Division Multiplexed (TDM) Serial Port and one Buffered Serial Port (BSP), of which the TDM port was used to control the ADC.

## 3.1 Hardware Interface

The following circuit diagram (Figure 3) shows the configuration that was used to interface the A/D converter to the DSP.

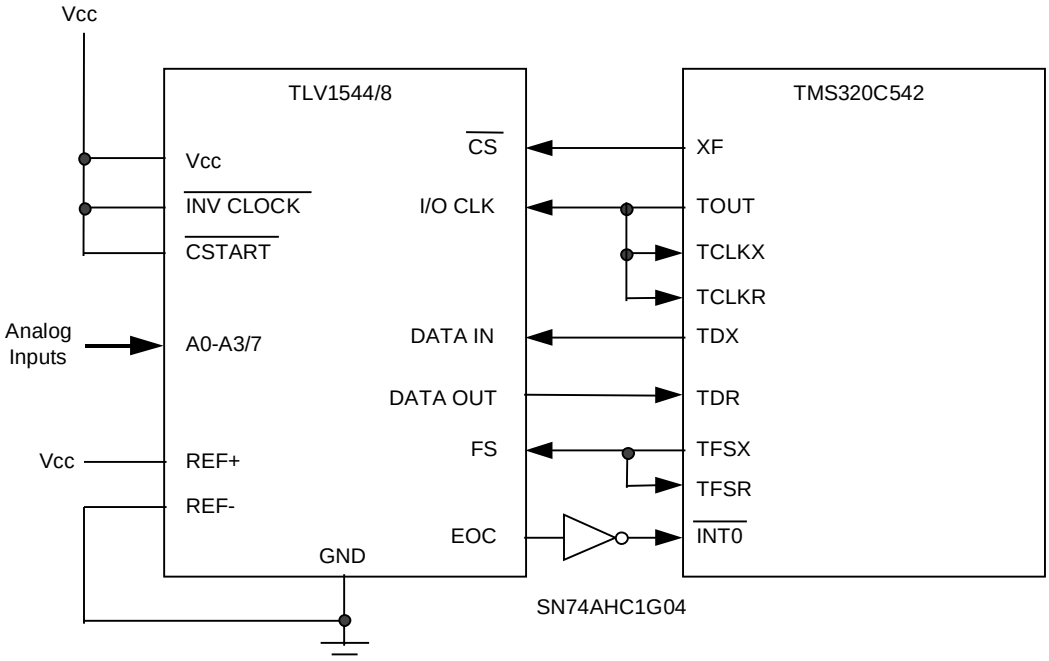


***Figure 3: ADC to 'C542 DSP Circuit Diagram***

| Pin | Description |
|---|---|
| INT0 | External user interrupt inputs |
| TCKLR | TDM port receive clock input |
| TCLKX | TDM port transmit clock (input or output) |
| TDR | TDM port serial data receive input |
| TDX | TDM port serial data transmit output |
| TFSR | TDM port receive frame synchronization |
| TFSX | TDM port transmit frame synchronization |
| TOUT | Timer output. |
| XF0 | External flag output (latched software-programmable signal) |

***Table 2: TMS320C542 Signal Description***

## 3.2 Software Interface

In order to interface the DSP and the TLV1544/8 together, a basic understanding of the A/D converter is required. The best way to visualize how the converter works is by using a timing diagram. Such a diagram can be seen below in Figure 4.

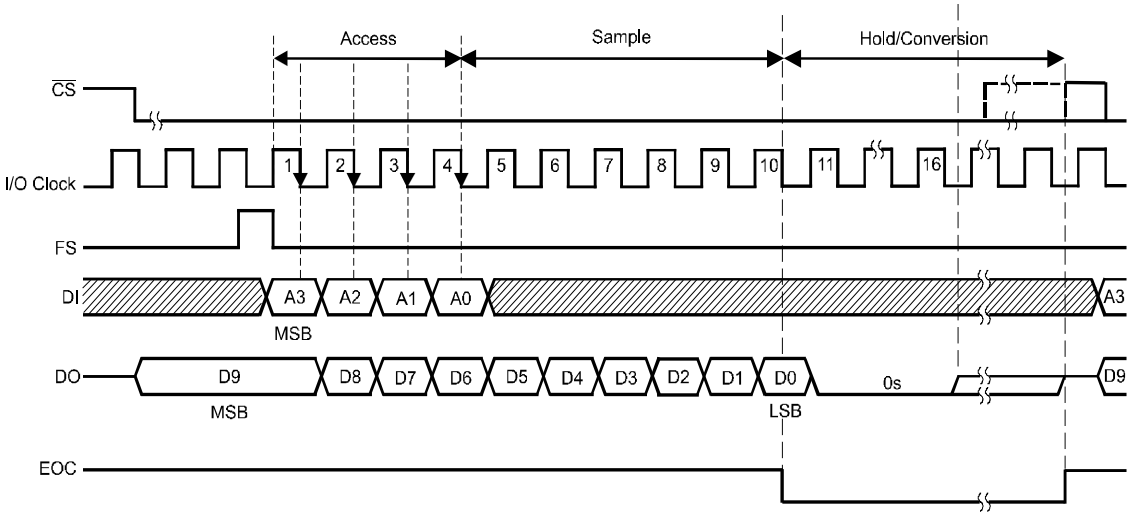


***Figure 4: Timing Diagram for the TLV1544/8***

As can been seen above, the I/O Clock signal (generated by the DSP) oscillates continuously. When $\overline{CS}$ is low and a Frame Sync (FS) signal is received on the FS pin, the TLV1544/8 starts simultaneously to receive the next operation mode byte and to send the last converted value. Once the first four input bits have been received, any more data to the input is ignored. After the LSB of the converted value has been sent, the EOC signal goes low for whatever conversion time has been previously set and returns to high once the conversion has been completed. Once the conversion is complete, the EOC signal goes back to high, and $\overline{CS}$ may return to a high state.

Now that the timing diagram has been shown, the DSP must be correctly configured so that the control software may accurately interface with the TLV1544/8.

### *3.2.1 Source Code*

Because the DSP software assembler can understand certain #include and .copy commands, the program was broken into various distinct files. This improves the readability and the understanding of the source code. The five files are as follows :

| | |
|---|---|
| ***Go.asm*** | Main program |
| ***Vectors.asm*** | Vector table |
| ***Values.asm*** | Variable setup |
| ***Timer.asm*** | Timer initialization function |
| ***TDM.asm*** | TDM Serial Port initialization function |

The best way to understand the operation of the software is to show the program in its entirety and then break it down into concise segments. The whole program using the algebraic assembler syntax is shown below in Listing 1.

```
;***********************************************************************
;       (C) COPYRIGHT TEXAS INSTRUMENTS, INC. 1997                      *
;***********************************************************************
;*                                                                     *
;* File: GO.ASM  Main routine for the TMS320C54x                       *
;*                                                                     *
;***********************************************************************
         .width   80
         .length  55
         .title  "TLV1544C ADC Interface routine"
;***********************************************************************
;This routine allows the ’C54x DSKplus to interface with an ADC on
;the TDM port of the DSP.

           .mmregs
           .setsect ".data", 0x500,1  ;place data section in data
                                      ;memory starting at address
           ;0x500
           .setsect ".text", 0x500,0  ;place code in program memory
                                      ;starting at address 0x500h
           .setsect "vectors",0x180,0 ;loads vectors section into
           ;absolute address 0x180h

           .sect "vectors"
           .copy "vectors.asm"        ;copy the interrupt vector
           ;table              from              the              file
                                      ;vectors.asm

           .sect ".data"
           .copy "values.asm"         ;copy the constants needed for
                                      ;the ADC to the data section

           .sect ".text"

start:
;Initialize the DSP control registers
           INTM = 1                   ;disable global interrupts
           XF = #high                 ;set CS to logic level high
           PMST = #01A0h              ;setup the PMST register
           SP = #0ffah                ;setup the SP register
```

```
;Initialize the memory and peripherals
           CALL init_DSP             ;initialise the DSP variables
           ;and memory
           CALL init_timer           ;initialise the TOUT Pin for 8
           ;MHz I/O Clock on ADC
           CALL init_TDM             ;Set up the TDM port
;Configure the interrupt operation of the DSP
           IMR = #241h               ;allow RINT and INT0
           INTM = 0                  ;enable global interrupts
;Initialize the ADC
           CALL power_up             ;initialises the ADC for the
           ;first time

;******************
;* Read in values *
;******************

start_read:
           AR6 = #location           ;pointer to data memory
                                     ;location
           AR7 = #num_samples        ;number of samples

samples_in:
           XF = #low                 ;set CS logic level low
           TDXR = #channel4_3        ;send            Software-Programmed
                                     ;operational mode to ADC
           CALL wait                 ;wait for EOC signal
           B = B << -6               ;shift   6   places   right   since
                                     ;ADC uses only 10 bit
           *AR6+ = B                 ;store value in data memory and
                                     ;pointer++
           if (*AR7- != 0) GOTO samples_in  ;continue  if  all  samples
                                          ;are  read  in,  else  branch
                                          ;back

function1:
           ;
           ;a function requiring all sampled values
           ;could go here e.g. FFT...
           ;
```

```
;*********************************************************************
;*                Program finished and simply looping                *
;*********************************************************************

end_loop:
          nop
          nop
          nop
          nop
          goto end_loop

;*********************************************************************
;*                         Here are the ISR's                        *
;*********************************************************************

;Interrupt service routine for the TDM port receiver interrupt
RINT:
          B = TRCV        ;load acc b with output from ADC
                          ;a function requiring single sampled
                          ;values could go here e.g. FIR, IIR.
                          ;However function must be shorter than
                          ;the conversion rate of the ADC

          return_enable   ;return from interrupt with interrupt
                          ;enable

;Interrupt service routine for the external interrupt 0, which is
connected to the EOC signal of the ADC
CS_CLEAR:

          XF = #high      ;set CS back to logic level high
          AR1 = #valid    ;set the EOC flag
          return_enable   ;return from interrupt with interrupt
                          ;enable
```

```
;*********************************************************************
;*                    Here go the callable functions                 *
;*********************************************************************

init_DSP:
          CALL clear_memory      ;clear the data memory location
          AR1 = #invalid         ;reset flag for EOC
          AR6 = #location        ;pointer to data memory location
          AR7 = #num_samples     ;number           of           samples
                                 ;pointers   used   in   the   sample
                                 ;collect loop
          return                 ;return from function call

clear_memory:
          AR6 = #location        ;pointer to data memory
                                 ;location
          A = #0h                ;acc zero
          REPEAT(#num_samples)   ;store 0 in memory "location" +
          ;"num_samples"
          *AR6+ = A
          return                 ;return from function call


;Initialize the ADC
power_up:
          XF = #low                ;bring CS logic level low
          TDXR = #fast_conv        ;set conversion rate. fast_conv
                                   ;is defined in the values.asm
          ;file
          CALL wait                ;wait   until   EOC   signal   is
                                   ;received
          return_enable            ;return to main program


wait:
          nop                      ;no operation
          nop                      ;no operation
          nop                      ;no operation
          if (*AR1 != 0) GOTO wait ;loop until EOC flag set
          AR1 = #invalid           ;reset the EOC flag
          return                   ;return from function call
```

```
;***************************************************************
;*                  Here go the copied file list               *
;***************************************************************

;Copy the timer initalization routine
          .copy "timer.asm"

;Copy the TDM port initialization routine
          .copy "TDM.asm"

;Here is the end of the main listing
          .end
```

***Listing 1: Main Program Source Code***

The following sections will deal with the description of the main program by defining and explaining each individual segment.

#### 3.2.1.1 Setting up the I/O Clock Signal using the On-Chip Timer

The I/O Clock signal can be generated externally by using a crystal or by the DSP. In the case of this Application Report, the I/O Clock was generated by the DSP using the TOUT pin. The timer consists of three registers:

| | | |
|---|---|---|
| TIM | Timer Register | The 16-bit memory-mapped timer register is loaded with period register value and decremented |
| PRD | Timer Period Register | The 16-bit memory-mapped timer period register is used to reload the timer register |
| TCR | Timer Control Register | The 16-bit memory-mapped timer control register contains the control and status bits of the timer (see the TMS320C54x User's Guide for details). |

***Table 3: TMS320C542 Timer Registers***

In order to generate an 8 MHz signal from TOUT, which will drive the TCLKR and TCLKX signals on the DSP, the on-chip timer must be configured as follows:

```
;This initialization routine sets up the timer
;You can interrupt the CPU by enabling the IMR=8 location.
;Set the TDDRreg and PRDreg below to configure the timer as defined
;by the equation.
;                            1
; TOUT cycle = ------------------------------
;              25ns * (TDDRreg+1) * (PRDreg+1)

;Therefore, by setting the respective registers to the following values ;a
TOUT cycle of 125 ns is achieved. This is equivalent to 8.0 MHz TOUT ;signal
;                          1
; TOUT cycle = --------------------  = 8 MHz
;              25ns * (4+1) * (0+1)


PRDreg     .set  0h
TDDRreg    .set  4h

;*****************************************************************
           .eval TDDRreg | 20h, TDDRval   ;set timer reload bit

init_timer:
           prd = #PRDreg               ;set prd register to value
           tcr = #TDDRval              ;set prd register to value
           return                      ;return from function call
```

***Listing 2: Timer.asm File Source Code***

Once the timer has been set to operate at the desired frequency, the serial port must be initialized to allow data to be passed to and from the ADC.

#### 3.2.1.2 Setting up the TDM Serial Port

For this interface, the Time-Division Multiplexed (TDM) Serial Port was used. This port allows the TMS320C542 to communicate serially with up to seven other devices. However, for this application the TDM port was configured as stand-alone, in which case it behaves exactly like the standard serial port.

The TDM port consists of eight registers:

| | | |
|---|---|---|
| TRCV | TDM data receive register | Holds the incoming TDM serial data. |
| TDXR | TDM data transmit register | Holds the outgoing TDM serial data. |
| TSPC | TDM serial port control register | The TSPC contains control bits, which configure the operation of the serial port. |
| TCSR | TDM channel select register | Selects in which time slot each TMS320C54x device is to transmit. |
| TRTA | TDM receive/transmit address register | Specifies in the eight LSBs the receive address of the TMS320C54x device and in the eight MSBs the transmit address of the TMS320C54x device. |
| TRAD | TDM receive address register | contains various information regarding the status of the TDM address line. |
| TRSR | TDM data receive shift register | controls the storing of the data from the input pin to the TRCV. |
| TXSR | TDM data transmit shift register | controls the transfer of the outgoing data from the TDXR and holds the data to be transmitted on the data-transmit pin (TDX). |

***Table 4: TMS320C542 TDM Port Registers***

For more information on these registers, please refer to the chapter 9 of the TMS320C54x DSP CPU and Peripherals Reference Set (see Appendix for literature).

Because the TDM serial port is being used in stand-alone mode, only the TSPC register must be configured for correct operation with the TLV1544/8.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Free | Soft | RSRFULL | XSREMPTY | RXDY | RRDY | IN1 | IN0 | RRST | XRST | TXM | MCM | FSM | FO | DLB | TDM |

***Figure 5: TSPC Register Layout***

In Figure 5, the layout of the TSPC register is shown. For the serial port to work correctly, bit 0 (TDM) must be set to 0. This places the TDM serial port into stand-alone mode. Additionally, since the FS signal must be generated by the DSP bit 5 (TXM) must be set to 1. Lastly, Burst Mode must be selected for correct data transfer, which means that bit 3 (FSM in stand-alone mode) must be set to 1. Additional information on these bits can be found at table 9-5 of the TMS320C54x DSP CPU and Peripherals Reference Set.

After that, an initialization routine must be run to initialize the TDM serial port. A typical function to do this is shown below.

```
;This initialization routine sets up the TDM to work as the standard
;serial port. The various bit are setup in order to be able to ;interface a
Texas Instruments TLC1544 ADC to the TDM port

init_TDM:
            intm = 1          ;disable all int service routines
            ifr = #0c0h       ;clear XINT and RINT flags in IFR
            a = imr           ;load interrupt mask register into
                              ;accumulator A
            a= a | #280h      ;wakeup from idle when TDM trns int
            imr = a           ;write it back to interrupt mask
                              ;register
            tspc = #028h      ;stop TDM serial port
            tdxr = #0h        ;send 0 as first xmit word
            tspc = #0E8h      ;reset and start TDM serial port
            return            ;return from function call
```

***Listing 3: TDM.asm File Source Code***

#### 3.2.1.3 Setting up the DSP Vector Table

When using interrupts, the vector table must be correctly configured. Since this application uses interrupts, the vector table should be as follows :

```
;The vectors in this table has been configured for processing 2 main
;interrupts.

;As you can see below, when a TRINT occurs, the vector has been set to ;goto
RINT and when an external INT0 occurs, the vector goes to ;CS_CLEAR

            .mmregs

reset       goto #80h                 ;00;RESET
            nop
            nop
nmi         return_enable             ;04;non-maskable external
                                      ;interrupt
            nop
            nop
            nop
trap2       goto #88h                 ;08;trap2
            nop
            nop
            .space 52*16              ;0C-3F;vectors for software
                                      ;interrupts 18-30
int0        goto CS_CLEAR             ;40;external interrupt int0
            nop
            nop
```

```
int1        return_enable             ;44;external interrupt int1
            nop
            nop
            nop
int2        return_enable             ;48;external interrupt int2
            nop
            nop
            nop
tint        return_enable             ;4C;internal timer interrupt
            nop
            nop
            nop
brint       return_enable             ;50;BSP receive interrupt
            nop
            nop
            nop
bxint       return_enable             ;54;BSP transmit interrupt
            nop
            nop
            nop
trint       goto RINT                 ;58;TDM receive interrupt
            nop
            nop
txint       return_enable             ;5C;TDM transmit interrupt
            nop
            nop
            nop
int3        return_enable             ;60;external interrupt int3
            nop
            nop
            nop
hpiint      goto #0e4h                ;64;HPIint
            nop
            nop
            .space 24*16              ;68-7F;reserved area
```

***Listing 4: Vectors.asm File Source Code***

**Note :** If an interrupt has been declared, one nop must be removed, since the goto instruction is a double word instruction and the memory location for this nop is therefore already used.

#### 3.2.1.4 Variables and the Software-Programmable Operation Mode Bytes

Lastly, the Software-Programmable Operation Mode bytes and some variables have been defined in the *values.asm* file.

```
;*************************************************************
;*                    Memory and sample setup values                 *
;*************************************************************
location    .set 2000h  ;start location for storing of sampled values
```

```
num_samples .set 50     ;samples   to   be   taken   (Remember   first   value
                        ;invalid) change this line for other arrays
valid       .set 0      ;a result is valid
invalid     .set 1      ;a result is invalid
low         .set 0      ;low logic value
high        .set 1      ;high logic value
;*********************************************************************
;*               ADC Software-Programmed Operation Modes             *
;*********************************************************************
;These values are derived from the TLV1548 and TLV1544 data sheet
;TLV1548
;*******
channel8_0  .set        0000h
channel8_1  .set        1000h
channel8_2  .set        2000h
channel8_3  .set        3000h
channel8_4  .set        4000h
channel8_5  .set        5000h
channel8_6  .set        6000h
channel8_7  .set        7000h

;TLV1544
;*******
channel4_0  .set        0000h
channel4_1  .set        2000h
channel4_2  .set        4000h
channel4_3  .set        6000h
```

```
;***********************************************************************
;*       Following values are used for both TLV1544 and TLV1548        *
;***********************************************************************

power_down  .set 8000h
fast_conv   .set 9000h
slow_conv   .set 0A000h
test_200    .set 0B000h
test_000    .set 0C000h
test_3FF    .set 0D000h
```

***Listing 5: Values.asm File Source Code***

**Note :** One important point to remember is that since the A/D converter only reads the first four bits of the operation mode word and the transmit register is 16-bits long, the mode byte must be placed in the most significant four bits of the transmit register. Therefore, all software-programmable operation mode bytes are (0)x000h.

| **MSB** | | | | | | | | | | | | | | | **LSB** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **1** | **0** | **1** | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

***Figure 6: TDM Transmit Register***

## 3.3 Software Flowchart

Now that each segment has been described, the following flowchart (Figure 7) shows the exact program operation.

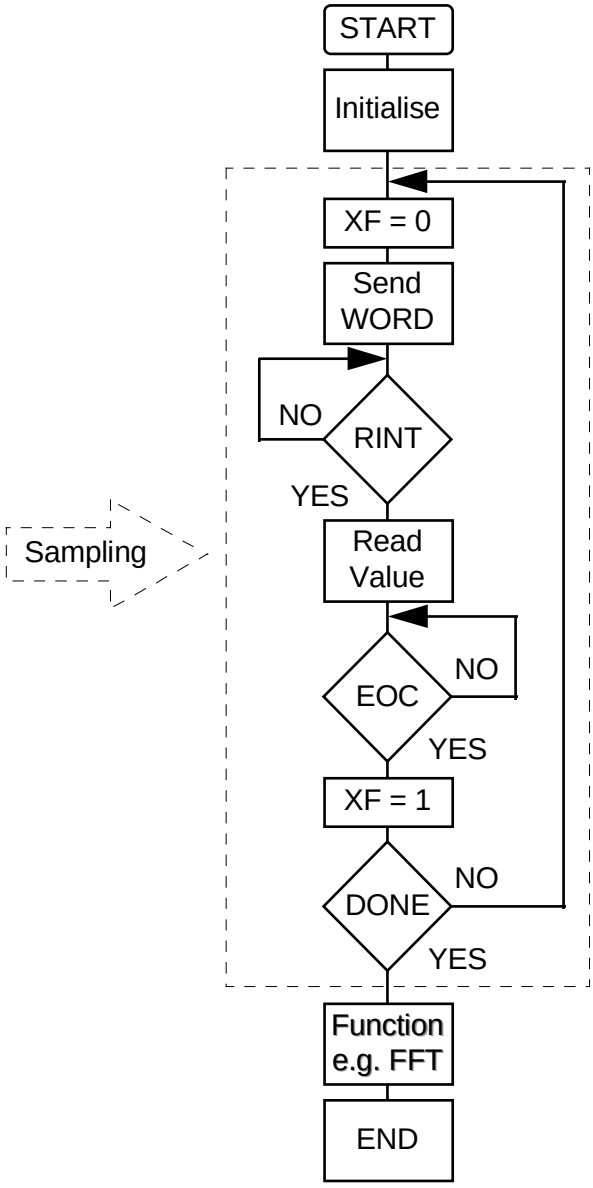


***Figure 7: ADC Interface Program Flowchart***

From Figure 7, the first routine initializes various variables, memory locations, the timer and the TDM serial port. Once this has been done, the DSP writes the first word to the A/D converter. This first word must contain the conversion time to be set for the TLV1544/8. Thereafter, the sampling process can start.

The XF ($\overline{CS}$) pin goes low to initialize the counter and state Machine on the A/D converter. Then a 16-bit (first 4-bit only read by ADC) operation mode word is sent to the TLV1544/8 and at the same time the DSP starts receiving the last 10-bit converted value. When the LSB (16-bit word) has been sent from the converter, the TDM serial port generates a Receive Interrupt (RINT). This indicates that the converted value is ready to be read from the receive register and then stored in memory.

Then the EOC signal goes low, indicating that the converter is busy.

Finally, once the conversion process is complete, the EOC return to high, causing an interrupt on $\overline{INT0}$, which brings the XF pin ($\overline{CS}$) back high. The process is now ready to repeat again, until all the samples have been collected.

## 3.4 Measured Timing Diagram

To show the user what should be expected, the waveform was recorded using a digital oscilloscope. The following (Figure 8) shows the $\overline{CS}$, I/O CLK, FS and EOC signals.

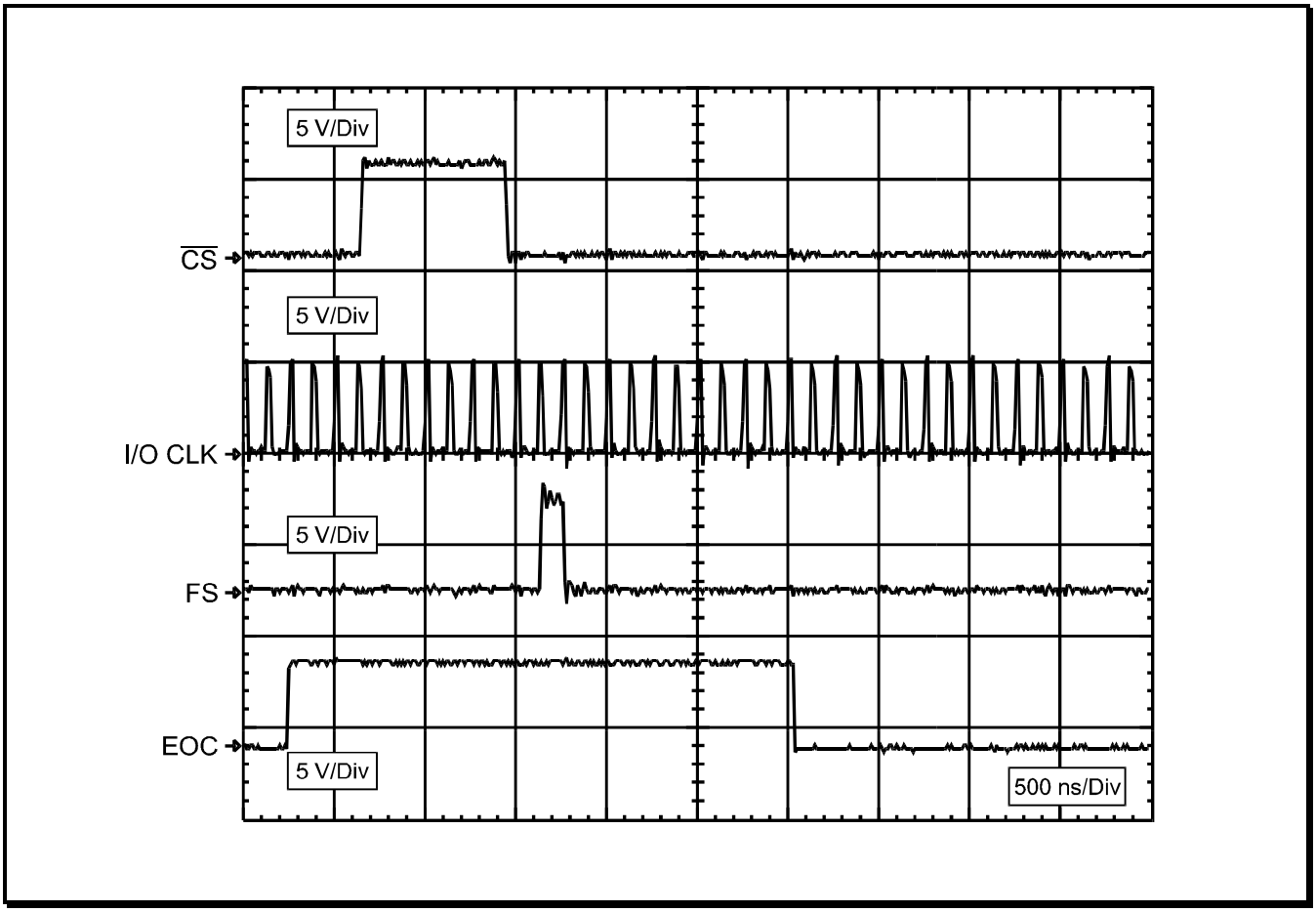


***Figure 8: Timing Waveforms from the Digital Oscilloscope***

# 4. The ADC to MC68B11E9 EBLP Interface

The TLV1544/8 was also interfaced to the MC68B11E9 EBLP Microcontroller by using the Serial Peripheral Interface SPI™.

## 4.1 Hardware Interface

The ADC was connected to the Microcontroller in the following way:

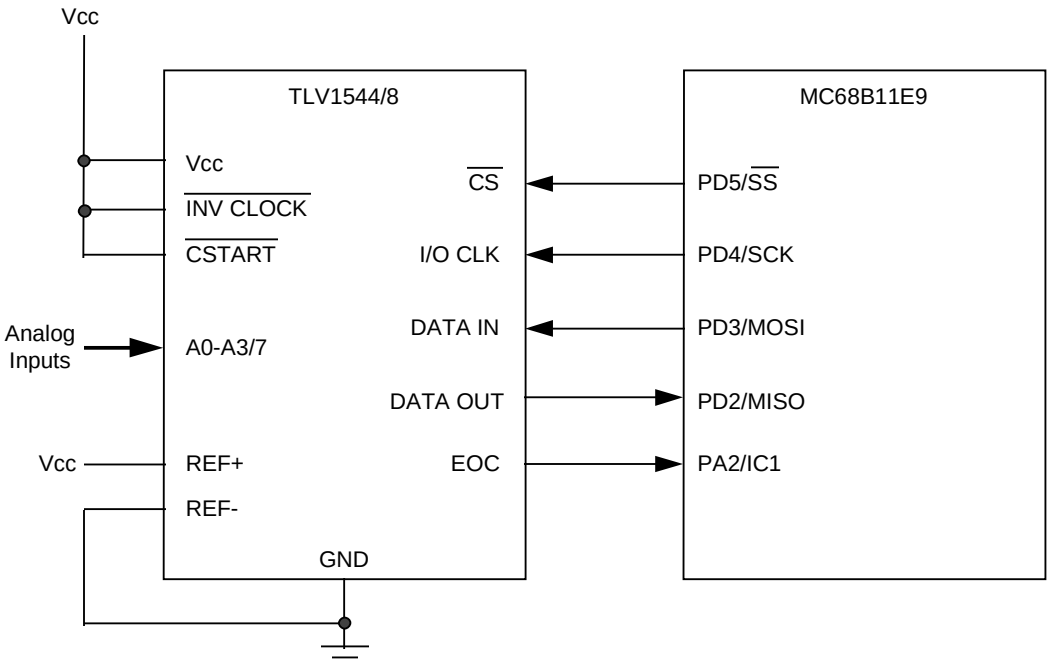


***Figure 9: ADC to MC68B11E9 Circuit Diagram***

## 4.2 Software Interface

The timing diagram for the interface to the Microcontroller is slightly different to that of the DSP. Because the TLV1544/8 uses the SPI™ of the microprocessor there is no need for the FS (Frame Sync) signal. The different timing diagram can be seen below in Figure 10.

SPI is a registered trademark of Motorola, Inc.

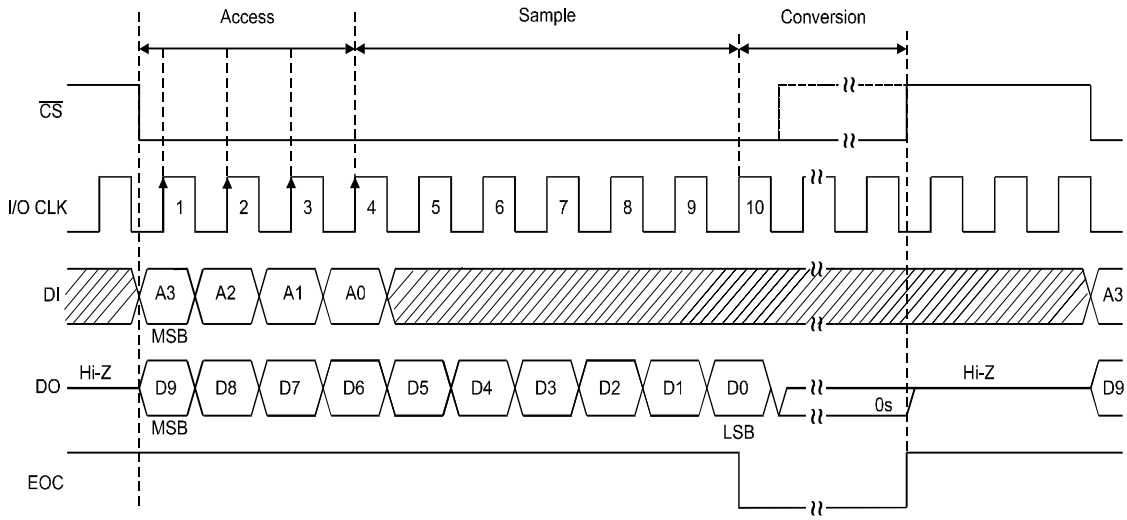


***Figure 10: Timing Diagram for TLV1544/8***

Once the CS signal goes low and the MCU sends the first bit to the A/D converter, the whole process of data transmission and reception simultaneously starts. Once the first four input bytes have been received, any more data to the input is ignored. After the LSB of the converted value has been received, the EOC signal goes low for whatever conversion time has been previously set and returns to high once the conversion has been completed.

**Note :** The I/O Clock signal on the SPI™ behaves differently to the TDM port on the DSP. The I/O Clock signal from the DSP was continuously generated by the TOUT pin. However, the SPI™ I/O Clock signal only generates eight clock pulses for each byte that is sent via the SPI™ MOSI. This means that to recover all 10-bit from DOUT, two bytes must be sent to the ADC (16 cock cycles in total). Figure 15 at the end of this section shows this characteristic.

### *4.2.1 Source Code*

To interface the MC68B119E with the TLV1544/8, the following file was created:

***ADCIO.asm*** The main program

Since the AS11 compiler does support the #include declaration, all the functions are included within this one file. As a result, the main source code will be shown first and then discussed in segments.

```
*****************************************************************
*          (C) COPYRIGHT TEXAS INSTRUMENTS, INC. 1997           *
*****************************************************************
* File: ADCIO.ASM  Software for the Motorola MC68HCE9 Starter Kit *
*****************************************************************
* Program entry point at routine "main".  The entry point       *
* is address $B600 (e.g. "G B600" from BUFFALO)                  *
```

```
*****************************************************************
* Equate statements for the below registers are commonly
* used as offsets to the x index register
* which contains the register block base address, i.e.
* $1000 for the "A" series, "E" series, and "L" series, and $0000 for the
* "D" series of 6811's
portd       EQU $08
ddrd        EQU $09
tctl2       EQU $21
tmsk1       EQU $22
tflg1       EQU $23
spcr        EQU $28
spsr        EQU $29
spdr        EQU $2a
*****************************************************************
* The following table of addresses corresponds to the memory map of  *
* the MC68L11E9; consult a memory map of your device if using any      *
* other 6811 device in this socket                                    *
*****************************************************************
ramlow      EQU $0000     ;<- will use this area for variables
bufstck     EQU $0041
rammid      EQU $0100
ramhi       EQU $01ff     ;<- will use this area for samples
regbas      EQU $1000
eeprom      EQU $b600
eprom       EQU $d000
buffalo     EQU $e000
*****************************************************************
* The following table of addresses corresponds to the pseudo vector  *
* map of the MC68L11E9                                                *
*****************************************************************
pvic1     EQU    $00e8     ;vector for EOC interrupt
*****************************************************************
* This area is for declaring certain variables for the ADC       *
*****************************************************************
***********
* TLV1548 *
***********
channel8_0  EQU $00
channel8_1  EQU $20
channel8_3  EQU $30
channel8_4  EQU $40
channel8_5  EQU $50
channel8_6  EQU $60
```

```
channel8_7  EQU $70
***********
* TLV1544 *
***********
channel4_0  EQU $00
channel4_1  EQU $20
channel4_2  EQU $40
channel4_3  EQU $60
*************************************************************************
* Following values are used for both TLV1544 and TLV1548                *
*************************************************************************
power_down  EQU $80
fast_conv   EQU $90
slow_conv   EQU $A0
test_200    EQU $B0
test_000    EQU $C0
test_3FF    EQU $D0
*************************************************************************
* Here are some variables used in the program                           *
*************************************************************************
num_samples EQU $80     ;number of samples to be taken
dummy       EQU $00     ;dummy value

            ORG ramlow

MSByte      RMB 1       ;MSByte from ADC
LSByte      RMB 1       ;LSByte from ADC
Counter1    RMB 1       ;Counter for # samples to go

*************************************************************************
* This is the main routine and also the entry point to the program  *
*************************************************************************

            ORG eeprom    ;Start at eeprom so that all code segments
                          ;can be started with the Buffalo monitor
                          ;command 'G B600', because the main routine
                          ;resides in EEPROM at $B600.

main
            LDS #bufstck  ;initialize the stack pointer
            LDX #regbas   ;initialize the index register

            BSR initPRTD  ;initialize port D for SPI
            BSR initPRTA  ;initialize port A for interrupt
```

```
            BSR initADC   ;initialize the ADC on the SPI
            BSR samples   ;get samples from the ADC

*********************************************************************
*                  These are the called functions                   *
*********************************************************************

initPRTD
            LDAA #$08
            STAA portd,x    ;initialize before turning on drivers
            LDAA #$38       ;let MOSI and SCK pins be output pins
            STAA ddrd,x
            LDAA #$50       ;let SPI be master
            STAA spcr,x
            BSET portd,x#$20  ;set CS to logic level high
            RTS

initPRTA
            LDAA #$7e       ;this is the op code for jump extended
            STAA pvic1      ;store in pseudo vector for IC1
            LDD #cs_clear   ;get start address of IC1 service routine
            STD pvic1+1
            LDAA #$10       ;IC1 capture on rising edge (EOC signal)
            STAA tctl2,x
            LDAA #$04       ;clear flag on FLG1 for IC1
            STAA tflg1,x
            STAA tmsk1,x    ;enable interrupts on IC1
            RTS

initADC
            SEI
            BCLR portd,x#$20  ;set CS to logic level low
            LDAA #slow_conv ;set ADC to fast conversion
            STAA spdr,x     ;send value to ADC

wait1       BRCLR spsr,x#$80 wait1  ;loop until data sent and received
*                                   ;i.e. If SPIF=0 --> wait
            LDAA spdr,x       ;load converted data in accumulator
            LDAA #dummy       ;send a dummy value
            STAA spdr,x       ;send value to ADC
wait2        BRCLR spsr,x#$80 wait2 ;loop until data sent and received
*                                   ; i.e. If SPIF=0 --> wait
            LDAA spdr,x       ;load converted data in accumulator
            CLI               ;enable global interrupts
```

```
*           WAI               ;wait for the interrupt on IC1
            RTS

samples
            LDAB #num_samples ;load the acc B with # samples to go
            STAB Counter1     ;store value

get_samples
            SEI
            BCLR portd,x#$20  ;set CS to logic level low
            LDAA #test_200    ;set value to send ADC
            STAA spdr,x       ;send value to ADC

wait3       BRCLR spsr,x#$80 wait3;loop until data sent and received
*                             ;i.e. If SPIF=0 --> wait
            LDAA spdr,x       ;load converted data in accumulator
            STAA MSByte       ;store MSByte of sample
            LDAA #dummy       ;send a dummy value
            STAA spdr,x       ;send value to ADC
wait4       BRCLR spsr,x#$80 wait4;loop until data sent and received
*                             ;i.e. If SPIF=0 --> wait
            LDAA spdr,x       ;load converted data in accumulator
            STAA LSByte       ;store LSByte of sample

            LDAA  MSByte      ;load acc A with MSB
            LDAB  LSByte      ;load acc B with LSB
            LSRD              ; ACCD right shift 6 times
            LSRD              ;since ADC 10 and 16 bits
            LSRD              ;received in total
            LSRD
            LSRD
            LSRD
            STAA MSByte
            STAB LSByte       ;store values again

            CLI               ;enable global interrupts
            WAI               ;wait for EOC signal from ADC
            nop
            nop

            LDAB Counter1     ;load counter value
            DECB              ;decrement acc B by one
            STAB Counter1     ;store counter value
```

```
            CMPB #$00          ;compare ACCB to 0
            BHI get_samples    ;if ACCB > 0 then get more

end_loop
            nop                ;program finished
            nop
            BSR end_loop
*************************************************************************
*                 This is the EOC interrupt routine                     *
*************************************************************************
cs_clear
            BSET portd,x#$20   ;Set CS logic level high
            ldaa #$04          ;clear flag so that
            staa tflg1,x       ;not constantly interrupted
            RTI
```

***Listing 6: Main Program Source Code***

The following sections will deal with the description of the main program by defining and explaining each individual segment.

### *4.2.2 Setting up the Register List*

The first section of the assembly code deals with the equating of each register with its offset address. The address for the registers starts at 1000h. Normally, the Index register is used to point at the base address (1000h) and then the appropriate register can be read or modified by offsetting the equated value from the Index register.

### *4.2.3 Setting up the Memory Map*

The next section equates the memory map to the respective memory addresses where the various sections of memory should be placed.

### *4.2.4 Setting up the Vector Table*

This segment simply equates the IC1 vector variable to the vector memory address. It will be used later to store the location where the program should jump to when an interrupt on IC1 occurs.

### *4.2.5 Defining Variables and the Software-Programmable Operation Mode Bytes*

As previously, the Operation Mode bytes have been declared for both the TLV1544 and TLV1548 with some extra variables.

**Note:** This time the Microprocessor transmit register is 8-bits long. Again the operation mode byte must be placed in the first four bits of the transmit register. Hence the reason for all Software-Programmable Operation Mode bytes are x0h.

| MSB | | | | | | | LSB |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **1** | **0** | **1** | **1** | 0 | 0 | 0 | 0 |

***Figure 11: Microcontroller Transmit Register***

## 4.3 The Program

As described above, the first part of the code sets up certain variable and register values The main program is located in the EEPROM area of memory and is described as follows. After initializing the stack pointer and the index register, there are 4 branches, the first two configure the ports and the latter two set up the ADC.

### *4.3.1 Setting up PortD*

PortD is used for the SPI™ to interface the ADC with the MCU. This is a four-wire link using the following pins:

$\overline{SS}$

SCK

MOSI

MISO

In order to configure this port, the DDRD and the SPCR registers must be modified.

As can be seen below in Figure 12, the DDRD is an 8-bit register. Bits 5,4,3 and 2 are used by the SPI system when the SPI enable (SPE) control bit is one. For the port to be set as Master i.e. the ADC is the Slave and is controlled by the MCU, bit 5,4 and 3 must be set to logic high.

| - | - | DDRD5 | DDRD4 | DDRD3 | DDRD2 | DDRD1 | DDRD0 |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **0** | **0** | **1** | **1** | **1** | **0** | **0** | **0** |
| - | - | $\overline{SS}$ | SCK | MOSI | MISO | TxD | RxD |

***Figure 12: DDRD Register Configuration***

Additionally, the SPCR register must be configured before SPI transfers can occur. Figure 13 (below) shows the SPCR register. The SPE and MSTR bits must be set to logic high for the port to be fully set up as Master.

| SPIE | SPE | DWOM | MSTR | CPOL | CPHA | SPR1 | SPR0 |
|---|---|---|---|---|---|---|---|

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | **1** | **0** | **1** | **0** | **0** | **0** | **0** |

***Figure 13: SPCR Register Configuration***

Once that has been done, the EOC interrupt must be set up.

### *4.3.2 Setting up the EOC Interrupt on PORTA*

This is done by using the Input Capture pin (IC1) on port A, The first thing to do is to write the starting address of the routine that deals with the interrupt to the PVIC1 location in the vector table. Once that has been done the following three registers must be configured:

TCTL2

TFLG1

TMSK1

The TCTL2 register controls on which edge (rising or falling) the particular input-capture pin causes a interrupt. Since the EOC signal from the ADC has a rising edge upon completion of conversion, the input-capture pin IC1 was set up to cause an interrupt on a rising-edge. This was done by setting bit 4 to logic high, as can be seen below in Figure 14.

| - | - | EDG1B | EDG1A | EDG2B | EDG2A | EDG3B | EDG3A |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** |

***Figure 14: TCTL2 Register Configuration***

TMSK1 and TFLG1 are the Input Capture Interrupt Enable an Input Capture Flag registers respectively. They need to be configured to set IC1 as Input Capture pin for the EOC signal. This is done by setting the IC1I bit on TMSK1 to one and clearing the flag related to that interrupt on TFLG1, as shown in Figure 15 and Figure 16 below.

| OC1I | OC2I | OC3I | OC4I | OC5I | IC1I | IC2I | IC3I |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** |

***Figure 15: TMSK1 Register Configuration***

| OC1F | OC2F | OC3F | OC4F | OC5F | IC1F | IC2F | IC3F |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** |

***Figure 16: TFLG1 Register Configuration***

Now that the ports have been correctly configured, the MCU program can start sending and receiving data to and from the ADC. As this is described in depth in the code listing, it will not be developed further in this section.

**Note:** The program shows how to collect *n* samples from the A/D converter on the SPI™ and store the sample in two memory locations, since the converted value is 10-bits and memory on the MCU is only 8-bit.

## 4.4 Software Flowchart

Now that each segment has been described, the following flowchart shows the exact program operation.

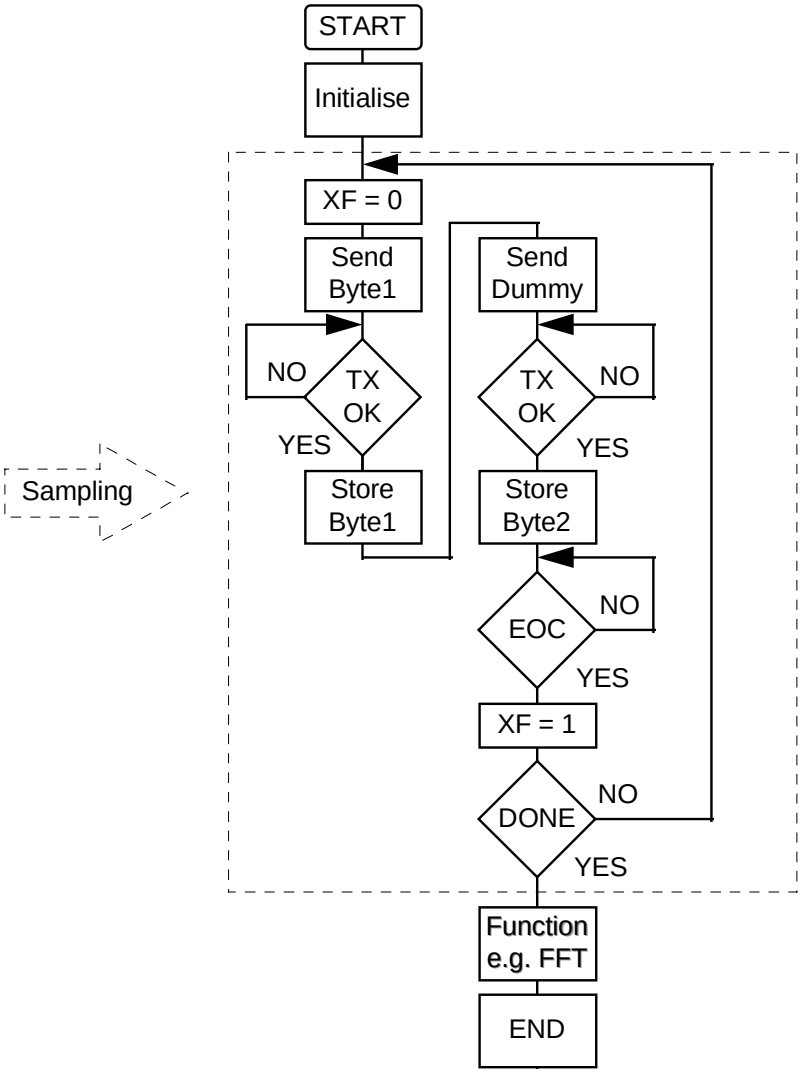


***Figure 17: Main Program Flowchart***

The first routine initializes the two ports that will be used in this application. PORTA is configured so that the IC1 pin on port A is the rising-edge triggered interrupt for the EOC signal from the A/D converter and PORTD is setup for the SPI™, so that the MCU is master and the TLV1544/8 is the slave.

Once the configuration is complete, the MC68B119E writes the first byte to the A/D converter. As before, this first byte must contain the conversion time to be set for the TLV1544/8. Because the transmit register is only 8-bits long, a dummy value must be sent after the first byte to allow the I/O CLK to reach 10 clock cycles. Once the conversion rate has been fixed, the process of getting samples can start.

The PD5/$\overline{SS}$ ($\overline{CS}$) pin goes low to initialize the counter and state Machine on the A/D converter. Then an 8-bit (first 4 bit only read by ADC) operation mode byte is sent to the TLV1544/8 and at the same time the MCU starts receiving the last converted value. The 8-bit dummy byte is sent to retrieve the last two bits from the converter.

When the LSB has finally been sent from the A/D converter, the EOC signal goes low, indicating that the converter is busy and two values that have been retrieved from the converter can be stored in two separate memory locations.

Once the conversion process is complete, the EOC returns to high, causing an interrupt on IC1, the software then brings the PD5/$\overline{SS}$ pin ($\overline{CS}$) back high. The process is now ready to repeat again, until all the samples have been collected.

## 4.5 Measured Timing Diagram

Just to show the user what should be expected, the waveform was recorded using a digital oscilloscope. The following Figure 18 shows the $\overline{CS}$, I/O CLK, DATA IN and EOC signals.

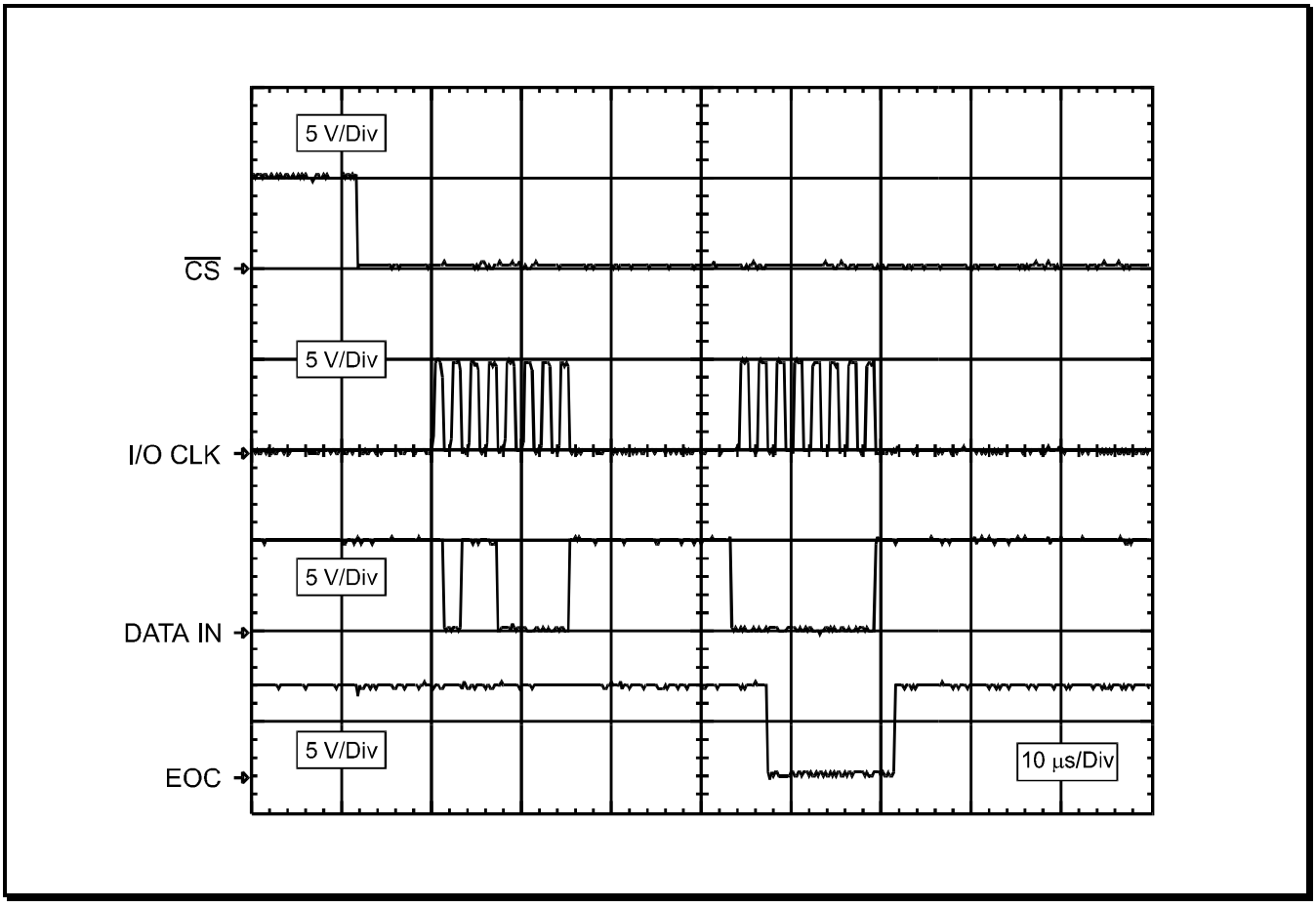


***Figure 18: Timing Waveforms from the Digital Oscilloscope***

# 5. Summary

This Application Report describes the hardware and software requirements for interfacing an A/D converter to a DSP and to a MCU. The 10-bit A/D converter TLV1544 (4 analog input channels) and the TLV1548 (8 analog input channels) from Texas Instruments have been used to develop such interface. Example software code has been written showing how to program the DSP and the MCU to control the A/D converter and to acquire samples. This is shown and explained methodically in the Application Report.

# 6. Appendix

| | |
|---|---|
| TLV1544/8 Datasheet | SLAS139A |
| Data Acquisition Data Book | SLAD001 |
| Data Converter Selection Guide | SLABE05B |
| Operational Amplifiers Data Book Volume A | SLYD011A |
| Operational Amplifiers Data Book Volume B | SLYD012A |
| Rail-to-Rail Operational Amplifier Selection Guide | SLOBE02 |
| Single Supply Operational Amplifier Selection Guide | SLOBE03 |
| Mixed Signal Analog CD-ROM | SLYC005A |
| TMS320C54x CPU and Peripherals | SPRU131C |
| TMS320C54x Algebraic Instruction Set | SPRU179 |
| TMS320C54x DSKplus User's Guide | SPRU191 |

Much useful software is available from the TI Internet site. The main TI Web site is at
`http://www.ti.com/`

Information on the TMS320C54x is at
`http://www.ti.com/sc/docs/dsps/tools/c54x/c54xdskp.htm`

TMS320C54x software can be downloaded from
`http://www.ti.com/sc/docs/dsps/tools/c54x/softsupp.htm`