Interfacing with the TLV1571/78 Analog-to-Digital Converter to the TMS320C542 DSP

Lijoy Philipose

ABSTRACT

This application report presents a hardware solution for interfacing the TLV1571/ TLV1578 10-bit, 1.25 MSPS low-power analog-to-digital converter (ADC) to the 16-bit fixed-point TMS 320C542 digital signal processor (DSP). The report describes the interface hardware and C-callable software routines, which support communication between ADC and DSP. Project collateral discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/SLAA077.

Contents

List of Figures

List of Tables

1 Introduction

The TLV1571/ TLV1578 is a 10-bit data acquisition system that combines a 1/8-channel multiplexed input, a 10-bit ADC, and a parallel interface. Its maximum throughput of 1.25 MSPS at 5 V, 625KSPS at 3 V, can be achieved when clocked at 20 MHz and 10 MHz respectively.

Using the TLV1571/ TLV1578 with the TMS320C542 40 MHz DSP demonstrates the power and simplicity of this ADC-DSP interface. This DSP provides the high-frequency clock rates needed to run the TLV1571/ TLV1578 at its limits.

This application note begins by highlighting the various devices on the EVM, as well as the development tools used. The software interface sections begins with section 5. By the end of this report, the user will understand the software portion of the interface well enough to test all of features of the TLV1571/TLV1578 ADC.

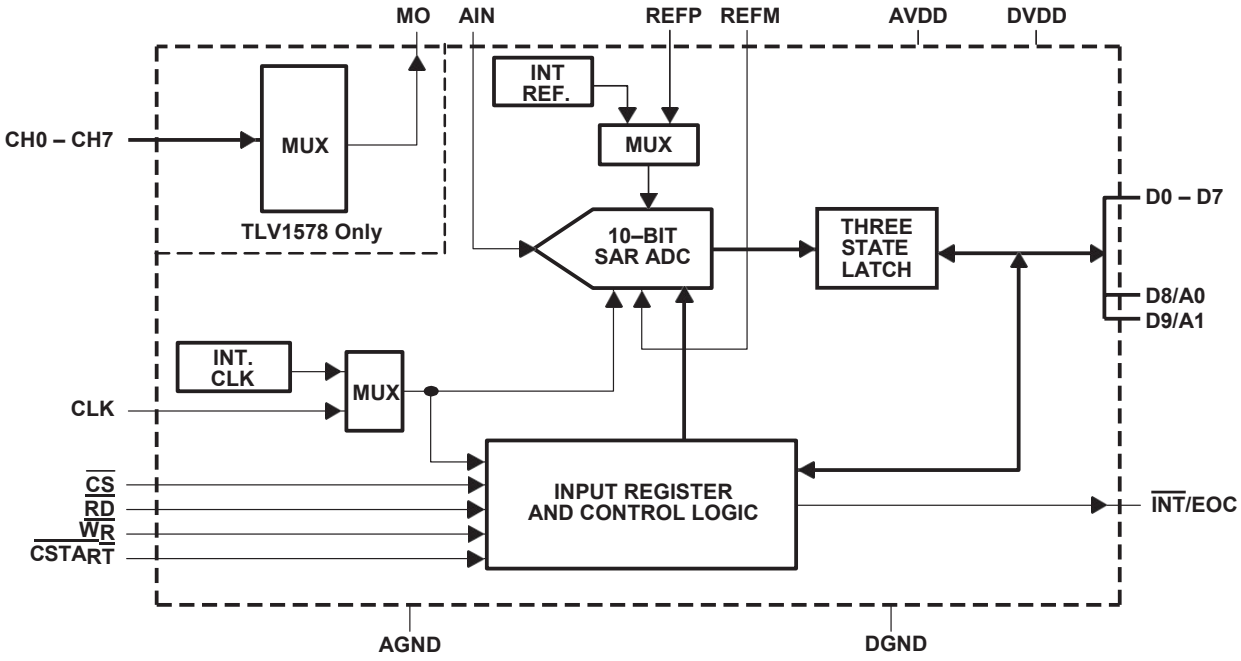
2 The Board

2.1 TMS320C54x DSKplus Starter Kit

TMS320C54x DSKplus software development tool is used extensively in both hardware and software testing. The ’C54x DSKplus, PC-linkable board, is the most powerful DSK development tool on the market. It is a low-cost development tool that enables designers to quickly start learning to use ’C54x DSPs. This Windows-based debugger makes the TMS320C54x DSKplus easy to use. It provides a visual environment that enables easier code development and reduces time-to-market.

2.2 ADC TLV1571/TLV1578 Overview

The TLV1571/TLV1578 is a 10-bit data acquisition system that combines 1/8-channel input multiplexer (MUX), a high-speed 10-bit ADC, and a parallel interface. The device contains two on-chip control registers allowing control of channel selection, software/hardware conversion start, and power down via the bi-directional parallel port. The MUX is independently accessible. This allows the user to insert a signal conditioning circuit such as an anti-aliasing filter or an amplifier, if required, between the MUX and the ADC. Therefore, one signal conditioning circuit can be used for all eight channels. The TLV1571 is a single channel analog input device with all the same functions as the TLV1578.



The TLV1571/ TLV1578 operates from a single 2.7 V to 5.5 V power supply. It accepts an analog input range from 0 V to AVDD and digitizes the input at a maximum 1.25 MSPS throughput rate at 5 V. The power dissipations are only 12 mW with a 3 V supply or 35 mW with a 5 V supply. The device features an auto-power down mode that automatically powers down to 1 mA 50 ns after conversion is performed. In software power-down mode, the ADC is further powered down to only 10 μA.

For more information see the TLV1571/ TLV1578 datasheet at the following URL:
http://www–s.ti.com/sc/psheets/slas170/slas170.pdf

2.3 System Development Features

This ADC has features that aid debugging of hardware and software problems during system development. Three self-test modes can be used to check whether the ADC is working properly; this can be done without having to supply an external signal. Register Readback modes can be used to determine whether the controls registers were initialized properly. The End-of-Conversion (EOC) signal can be used to determine whether the data is valid.

2.3.1 C54x to TLV1571/78 Interface

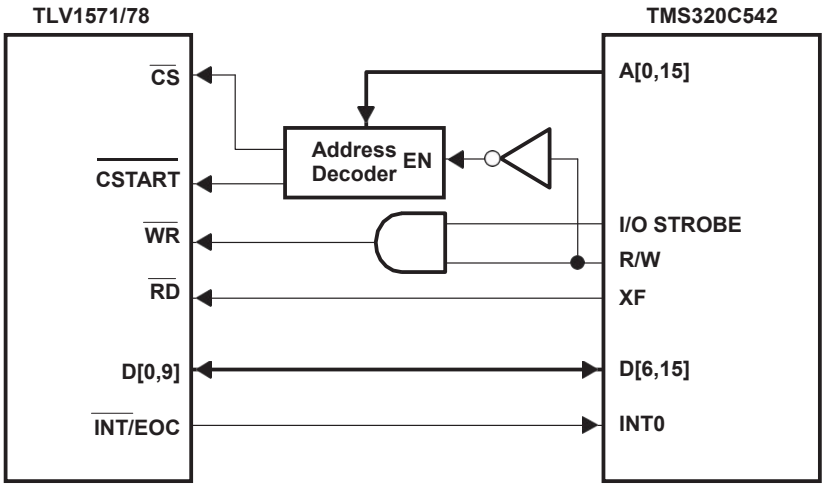


Figure 1. ADC-DSP Interface

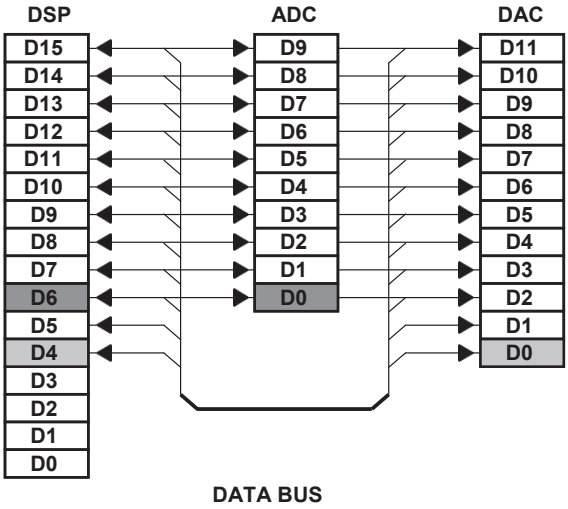


Figure 2. Data Bus Mapping for DSP-ADC-DAC

Figure 1 shows the simple interface with TLV1571/ TLV1578 and TMS320C542 DSP. The TMS320C542 DSP is a 16-bit device. The Data lines are mapped MSB-MSB on both ADC and DAC devices, See Figure 2. This becomes important when users write to these devices. when writing to ADC, DSP data bits 15 through 6 must contain the data to be sent. Likewise, data bits 15 through 4 must contain the data to be sent to DAC.

2.4 Onboard Components

The TLV1571/TLV1578 EVM contains three major devices. They are the TLV1571/TLV1578 ADC, TLV5619 DAC, and the TLV2771 op amp. The following sections are only a brief introduction to these devices. For a more detailed explanation on these devices refer to the *TLV1571/TLV1578 User's Guide*. The user's guide is located at: http://www–s.ti.com/sc/psheets/slau025/slau025.pdf

2.4.1 TLV5619 DAC

The TLV5619 is a 12-bit voltage output DAC with a TMS320 compatible parallel interface. The 12 data bits are double buffered so that the output can be updated asynchronously using the $\overline{\text{LDAC}}$ pin. During normal operation, the device dissipates 8 mW at a 5 V supply and 4.3 mW at a 3 V supply. The power consumption can be lowered to 50 nW by setting the DAC to power down mode. For more information on TLV5619 DAC access the following URL: http://www–s.ti.com/sc/psheets/slas172b/slas172b.pdf.

2.4.2 Operational Amplifier

One signal conditioning circuit can be used for all eight channels of the TLV1578. The TLV1571/TLV1578 EVM uses the TLV2771 operational amplifier to perform this task. The TLV2771 CMOS operational amplifier with its rail-to-rail output swing, high input impedance, excellent dc precision, and high output drive makes this device a good choice for driving the analog input of the ADC. The device provides 10.5 V/μs of slew rate and 5.1 MHz of bandwidth, while only consuming 1 mA of supply current. For more information on the TLV2771 Op Amp, access the following URL: http://www–s.ti.com/sc/psheets/slos209c/slos209c.pdf.

3 Operational Overview

The hardware interface must be understood before writing the software interface. The following chapter describes the connection between the DSP and the EVM.

3.1 Reference Voltage Inputs

The voltage difference between the VREFP and VREFM terminals determines the analog input range. For example with VREFM = 0 V, VREFP = 5 V, a dc input of 5 V would produce a full scale value (3FFh). Likewise a dc = 2.5 V will produce half-full scale output (1FFh). For external reference specifications refer to the TLV1571/TLV1578 datasheet at: http://www–s.ti.com/sc/psheets/slas170/slas170.pdf

3.2 Input Data Bits

The ADC contains two user-accessible registers, control register zero (CR0) and control register one (CR1). All user-defined features are programmed using CR0 and CR1. The data acquisition process must be started by first writing to these two registers. After which, the converter processes data in the same configuration until the registers' contents are changed.

3.3 Connections Between the DSP and the EVM

Table 1 provides interface connections between the C54x DSKplus board and TLV1571/ TLV1578 EVM.

3.4 DSP Memory Map

The C542 DSKplus board has additional reserved memory segments other than those described in the C54x reference set volume 1. The C54x has 10K of DRAM, the DSK reserves 1000h–100Ah for housekeeping functions. There is one block of physical memory on the board. Figure 3 shows the DSKplus memory mapping used in this application report.

Refer to the *TMS320C54x DSKplus DSP Starter Kit User's Guide* for a complete memory description.

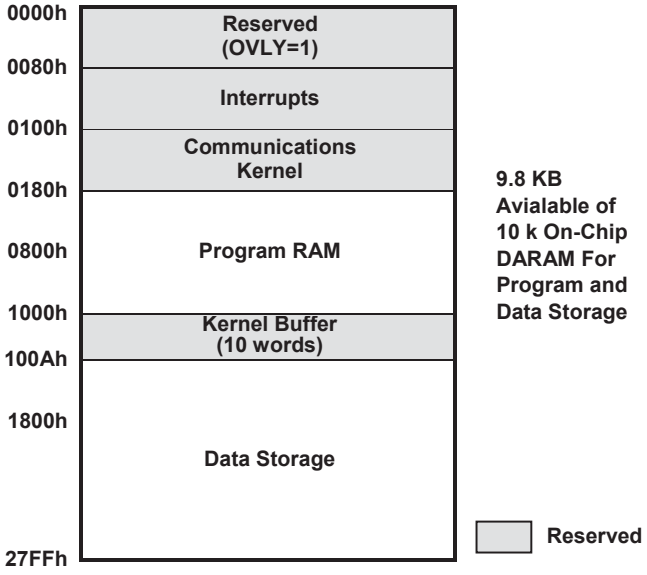


Figure 3. Memory Map Used in This Application Report

Table 1. Signal Connections

DSP SIGNAL	CONNECTOR/PIN ON THE DSKPLUS CIRCUIT BOARD	CONNECTOR/PIN ON THE TLV1571/78 EVM	ADC SIGNAL
General			
GND	Connector JP4: Pin 1, 10, 11, 12, 14, 15, 19, 20, 21, 27, 34, 35	Connector J6: Pin 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26	GND
	Connector JP5: Pin 6, 10, 11, 12	Connector J7: Pin 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26	GND
		Connector J10: Pin 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34	GND
Vcc	JP1/32	N/A	Vcc

Table 1. Signal Connections (Continued)

DSP SIGNAL	CONNECTOR/PIN ON THE DSKPLUS CIRCUIT BOARD	CONNECTOR/PIN ON THE TLV1571/78 EVM	ADC SIGNAL
Parallel Interface			
CLKOUT	JP3/2	J10/33	CLKOUT
$\overline{\text{INTO}}$	JP5/1	J7/17	$\overline{\text{INT}}$
XF	JP4/8	J7/23	$\overline{\text{RD}}$
R/W	JP4/30	J7/21	Decoded to the $\overline{\text{WR}}$ line
$\overline{\text{IO STRB}}$	JP4/36	J7/19	Decoded to the $\overline{\text{WR}}$ line
A0	JP5/34	J7/15	Address decoder to $\overline{\text{CS}}$ and $\overline{\text{CSTART}}$
A1	JP5/35	J1/13	Address decoder to $\overline{\text{CS}}$ and $\overline{\text{CSTART}}$
D6	JP3/17	J10/19	D0
D7	JP3/18	J10/17	D1
D8	JP3/20	J10/15	D2
D9	JP3/21	J10/13	D3
D10	JP3/23	J10/11	D4
D11	JP3/24	J10/9	D5
D12	JP3/26	J10/7	D6
D13	JP3/27	J10/5	D7
D14	JP3/29	J10/3	D8
D15	JP3/30	J10/1	D9

NOTE: DSP D[15,6] is tied to ADC D[9,0]

4 Communicating Between the TLV1571/TLV1578 and the DSP

The next few sections explain the interface with the DSP and TLV1571/TLV1578.

4.1 Writing to ADC

PORT(PA) = Smen

Writing to the I/O bus uses the port instruction. PA sets the ADDRESS bus permanently to that value. Smem is a value from memory being transferred to the data bus.

```
@CR0_Send = #040h    ;set the content of memory address CR0_Send to #040h
port(#2) = @CR0_Send ;set address bus to #2 and write #040h onto the Data bus.
```

The DSP automatically generates the $\overline{\text{WR}}$ pulse via the R/W pin.

4.2 Reading From ADC

Smen = PORT(PA)

Reading from the I/O bus. PA sets the ADDRESS bus. Smen is a memory cell. PA is the address on the bus. The above command can be used to clear $\overline{CS}/\overline{CSTART}$. The user must generate a $\overline{RD}$ pulse using the XF pin. There are two different ways to read data out.

This following method mirrors the datasheet, however, it takes needless DSP cycles:

1. PORT(#1)=Smen. Set DSP address bus to 1h. THis selects ADC $\overline{CS}$ on address decoder. It does not matter what the user chooses to write to the data bus.
2. Clear XF(=0), This causes read pulse to go clear.

3. Smen = PORT(PA). Read data out of ADC and store in variable Smen.
4. Set XF(=1), Read Pulse goes high. Data latched out of ADC.
5. Set DSP address bus to 0h. This causes the address decoder to select Zero, which sets ADC $\overline{CS}$ high.

The following method used in the attached source:

1. Clear XF(=0), Read pulse goes low.
2. Issue read command. Smen = PORT(PA). Selects ADC $\overline{CS}$ on address bus.
3. Set XF (=0) . Read pulse goes high.

This method takes advantage of the delay in the C542 DSK board. The board produces enough delay so that XF comes after ADC chip select is cleared. The attached code includes a NOP after a XF command is issued to account of this delay. The NOP resynchronize the $\overline{RD}$ and chip selects lines.

4.3 Initializing DSP

Before running your application, you must initialize the appropriate C542 DSP registers. The following registers are initialized to allow interrupts and proper hardware interface. The interrupt flag register (IFR) is a memory-mapped CPU register that identifies interrupts. This application uses INT0. When INT0 occurs, IFR is set. The interrupt mask register (IMR) individually masks off specific interrupts at the required times. INT0 is enabled when the respective bit in the IMR register is set. INTM is a bit in status register (ST1) that globally masks or enables all interrupts. This bit must be set, if interrupts are used at all. The software wait-state register (SWWSR) extends external bus cycles up to seven machine cycles. This is intended for use when interfacing with slower off-chip I/O devices, i.e., TLV1571/TLV1578. The attached source code assumes a wait-state of one.

For more information on wait-states refer to the following URL:
http://www–s.ti.com/sc/psheets/spru131f/spru131f.pdf

4.4 Data Page Pointer

```
DP = #0         ;Load DP with 0
DP = #variable ;Point with DP to the page, where variable is stored
DP = #register ;Error, this will not work. The DP is loaded with register content.
```

DP must point to the Data Page where variables are stored.

4.5 Generating the Chip Select Signal and the $\overline{CSTART}$ Signal

```
port(ADC) = @CR0_SEND   ;Clear CS (set Chip Select Low). Writing to port.
@temp = port(DEACTIVE)  ;Set CS and CSTART High. Reading from port.
@temp = port(ADC)       ;Set ADC CS low.
@temp = port(CSTART)    ;Set CSTART low.
```

The $\overline{CS}$ and $\overline{CSTART}$ signals are accessed using the address bus. The address decoder attached to DSP address bus sets the respectively signals high or low.

5 Software Overview

This application note consists of a C-callable assembler routine (c1571evm.asm) and its C program (c1571c.c). The assembler code is kept divided so the user can identify what source does which function. The assembly source code is divided into four segments.

1. Single channel mode with hardware start of conversion
2. Sweep channels mode with hardware start of conversion
3. Single channel mode with software start of conversion
4. Sweep channels mode with software start of conversion

The C program enables the user to specify register configurations. Users need only to set register variables to zero or one, similar to the control register map found in TLV1571/TLV1578 datasheet. In addition, the user can specify where to start storing samples, total number of samples to collect, and whether or not to send data to onboard DAC. The C program then calls the assembler function. The assembler program executes in the following steps:

1. Enable and reset interrupts
2. Format register variables to configure ADC
3. Sample and collect conversion data
4. Store collected data into DSP data memory
5. Collecting total numbers of data specified
6. Disable the ADC and return to C function

Table 2. Local and Global Variables and Corresponding Programs

PROGRAM	TYPE	VARIABLE	DESCRIPTION
C FILE	Global	_STARTSEL	Hardware or software start of conversion
	Global	_PROGEOC	Interrupt or end of conversion signal
	Global	_CLKSEL	Internal or external clock
	Global	_SWPDWN	Normal or power-down mode
	Global	_MODESEL	Single channel or sweep mode
	Global	_CHANNEL	Single or sweep mode
	Global	_OSCSPD	Internal OSC slow(10 MHz) or fast(20 MHz)
	Global	_OUTCODE	Binary or 2s complement code output
	Global	_OUTPUT	Normal conversion or self test[1,2,3] or CR[1,2] output
	Global	_NumSamples	Collect this many samples per assembly call.
	Global	_AtMemory	Store samples starting at this memory address
	Global	_SendDAC	Send sample to DAC or not
Assembly	Local	AD_DP	Label used to set data page
		ADSAMPLE	Store current sample read in from ADC
		CR0_SEND	Control word 1 written to CR0
		CR1_SEND	Control word 2 written to CR1
		NEXT_CH	Used to keep track of next data table to store sample.
		TABLE_7	Used to address next location in data table 7.
		TEMP	Used during dummy reads. i.e. toggle address decoder signals.

5.1 Configuration Cycle

TLV1571/ TLV1578 requires initialization of CR0 and CR1. The user must write to both control registers before accepting valid data.

```
@CR0_Send = #040h
port(#2) = @CR0_Send ;Set address bus to 2h (ADC_Chip Select) and write #040h
                     ;onto the Data bus.
@CR1_Send = #140h
port(#2) = @CR1_Send ;Set address bus to 2h (ADC_Chip Select) and write #140h
                     ;onto the Data bus.
```

Figure 4 is an example of using software start of conversion mode. It is important to note the second write pulse begins conversion process. It is recommended that the user issue a read pulse only after conversion is completed.

The user may start the conversion process anytime after the configuration cycle. Figure 5 is an example of using hardware start of conversion mode. In this case the conversion cycle process begins right after the configuration cycle.

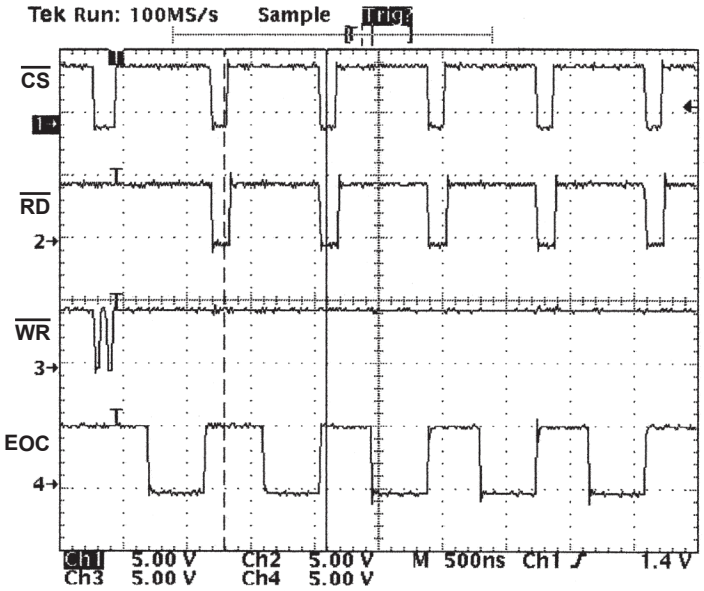


Figure 4. Software Start Configuration Cycle With EOC

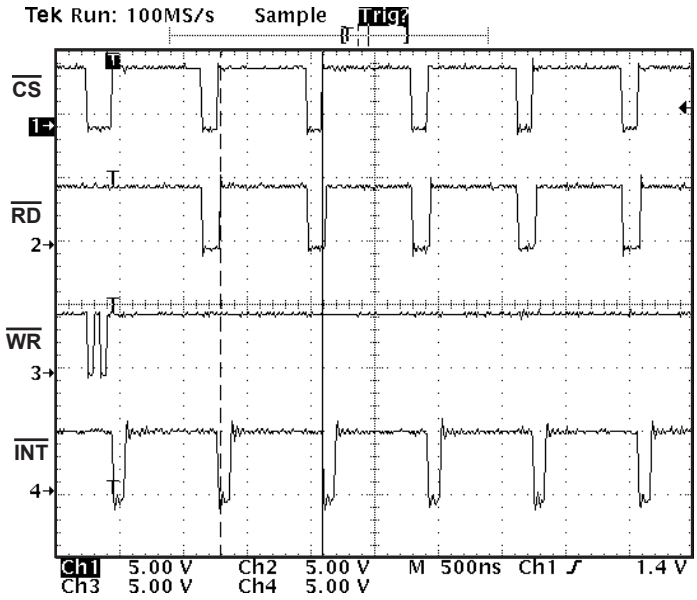


Figure 5. Software Start Configuration Cycle With $\overline{INT}$

5.2 Assemble Code Instruction Set

Assembly code in Chapter 7 an uses Algebraic Instruction set. The following link describes the Algebraic Instruction set for c54x DSP.

See http://www–s.ti.com/sc/psheets/spru179b/spru179b.pdf

5.2.1 Macros

Macros are text substitutions made at assembly time. The macrocode is literally dumped into the program with the parameter names substituted. Macros are useful when source code becomes tedious and repetitive, or when a branch routine would add too many clock cycles. Macros are used in the attached source code to help simplify program read.

For more information on writing and using macros, refer to
http://www–s.ti.com/sc/psheets/spru102c/spru102c.pdf

5.3 Loopback

Using the onboard DAC, the user can observe the data converted by the ADC. During development, it is useful to be able to compare the analog input signal and DAC output. The example code shows how this is done in software.

```
DAC .set #3
Port(DAC)= @ADSAMPLE  ;write 3h on address bus and write ADSAMPLE on data bus.
```

In source code, the conversion data is fed straight out to DAC in an attempt to emulate real-time processing.

To enable loopback mode, tie pin 1 and pin 2 together on W4.

5.4 Store Data

Sweep mode allows the user to collect data from various ADC channels and store them into DSP memory. The user has to specify the location to store the first sample and how many samples to collect. Using variables *At_Memory* and *NumSamples* the assembly program decides where to store the first sample and how many samples to collect.

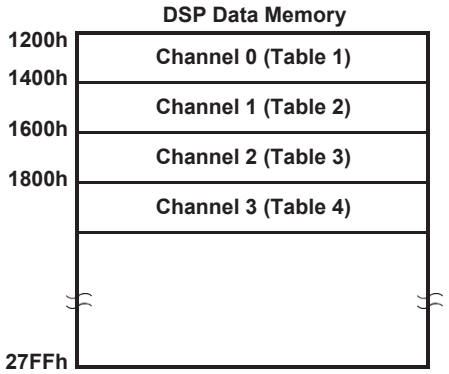


Figure 6. Sample Storage Format with At_Memory=1200h and NumSamples=200h

The ADC is programmed for four-channel sweep mode, data storage beginning at 1200h and 200h samples per channel would format memory like in Figure 7. In Figure 7, all samples collected from channel 0 are stored in at 1200h through 13FFh. The data tables are allocated in sequential memory addresses, therefore care must be taken to insure that sample tables fit in the available storage range (1200h–27FFh).

5.5 Optimization for a Specific Application

Allowing the user to input variables as one or zero adds extra cycles to the assembly program. This delays teh program from collecting and storing samples. The DSP spends several cycles to understand the control register variables, and then decide which program to call. The program flow chart is described in section 6. Users can decrease the function run time by bypassing the variable reformatting segments of the code. Also, by compiling only the program segments used in the application, program memory space can be decreased.

The attached source code is written so that programmers can easily extract pieces of code and modify it to their specific application.

The End-of-Conversion (EOC) mode gives the user valuable information; EOC pulse width gives the user conversion times and ADC activity. With this information, the user can tailor software and hardware to take advantage of that specific ADC's conversion characteristic.

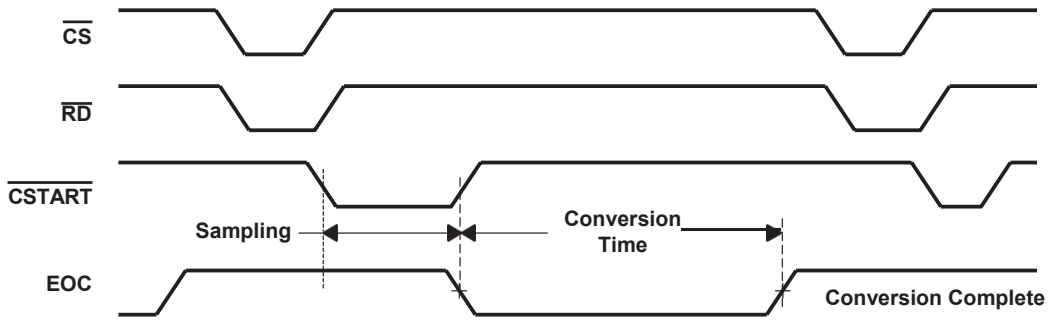


Figure 7. Tracking ADC Activity Using EOC Pulse

5.6 Flow Charts and Comments for All Software Modes

The source code included in this application note reads the data soon after each conversion is complete.

5.7 DSP INITIALIZATION

Before proceeding to program the ADC, the DSP must be set up. One of the things to be done is to define the DSP interrupt vector table. When interrupts occur, the DSP will refer to this table to determine the next course of action. This action often is branching to an interrupt service routine (ISR). In this application note, the ISR simply resets the interrupt.

For more information on interrupts see TMS320C54x CPU and Peripherals Reference Set Volume 1 at http://www–s.ti.com/sc/psheets/spru131f/spru131f.pdf

1. Using a DSP this fast with a slower external device requires using wait-states. A wait-state of ONE is used during write cycles.
2. External interrupt zero (INT0) is tied to the ADC interrupt pin. Reset any old interrupts on this pin.
3. Program the IMR register to allow INT0.
4. The debugger needs to do background interrupts, so maskable interrupts are enabled.

There are four different subroutines attached with this note. The programs fall into two categories, hardware start of conversion and software start of conversion. The following sections will explain the different start of conversions. Please refer to either the attached source (Chapter 7) or flow diagram (Figure 8). Every subroutine starts by configuring the ADC. The change comes with starting the conversion cycles.

NOTE: When using sweep mode, it is recommended that the op amp be bypassed.

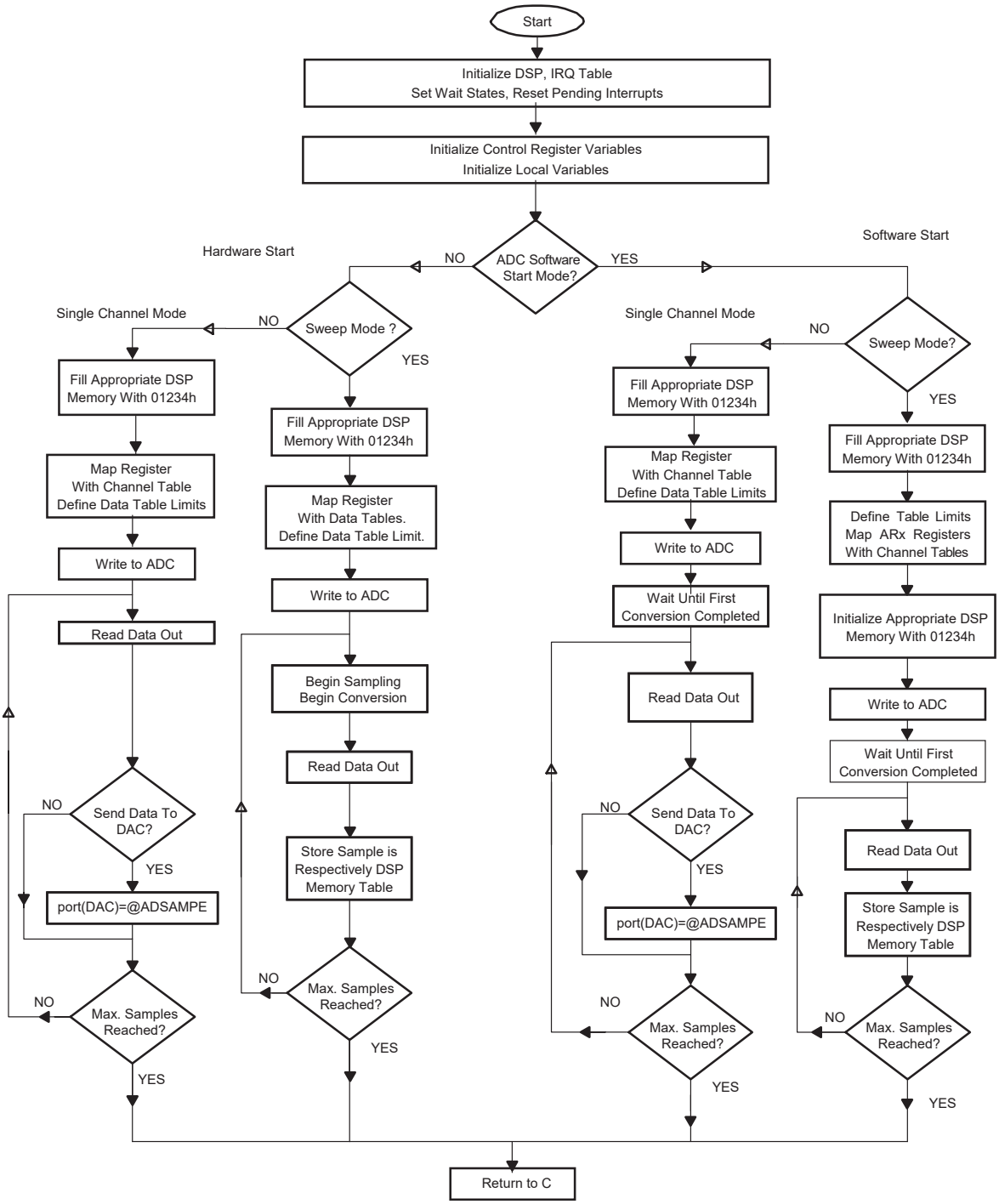


Figure 8. Software Flow Chart

5.7.1 Single and Sweep Channel Modes With Software Start of Conversion ($\overline{RD}$)

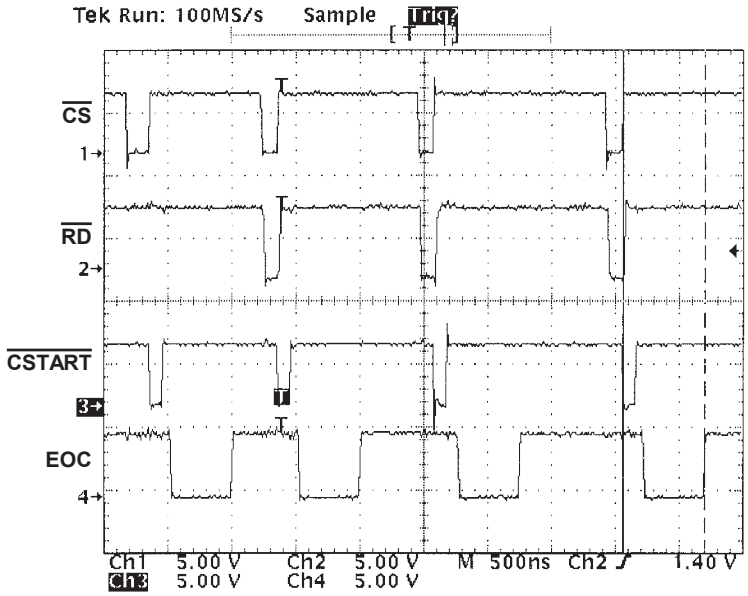


Figure 9. Software Start of Conversion With EOC Signal

Software start of conversion refers to using the $\overline{RD}$ pulse to begin sampling. In both sweep and single channel modes, sampling begins with low/high transition of $\overline{RD}$. In Sweep mode, the rising edge of the $\overline{RD}$ pulse begins sampling the next channel in the selected sweep sequence. During the configuration cycle, sampling begins with the rising edge of the second $\overline{WR}$ pulse. As a result, the first $\overline{RD}$ pulse must not come before the conversion cycle is completed. Thereafter the rising edge of $\overline{RD}$ begins sampling. Figure 9 is an example of what the user will see when running in the TLV1571/TLV1578 in software start mode.

5.7.2 Single and Sweep Channel With Hardware Start of Conversion (CSTART)

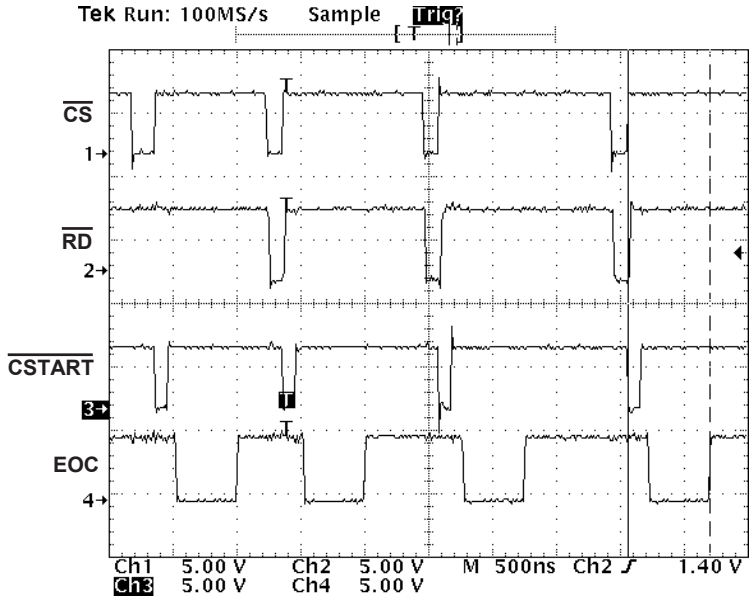


Figure 10. Hardware Start of Conversion Using EOC Pulse

Hardware start of conversion refers to using the $\overline{\text{CSTART}}$ pin to begin sampling and conversion. The user may begin a conversion cycle immediately following the configuration cycle. The falling edge of $\overline{\text{CSTART}}$ begins the sampling process, while the rising edge is used to begin the conversion process. This mode allows the user complete control over when sampling begins and how long it is sustained. It is important to remember, in sweep mode, $\overline{\text{CSTART}}$ begins the conversion cycle on the next channel in the sequence. If the $\overline{\text{RD}}$ pulse is not provided after conversion, but $\overline{\text{CSTART}}$ begins the next conversion cycle, the previous data will be lost. Figure 10 is an example of what the user will see when running in the TLV1571/TLV1578 in hardware start mode.

6 C-Callable

```
/*************************************************************************/
/* Title:      TLV1571 ADC C program main routine                        */
/* File:       C1571C.C                                                  */
/* Description:   In this c-program file the user select the             */
/*                Input channel(s), the Conversion Modes, the Memory     */
/*                start address, and the number of Samples. This code is */
/*                used with the ADC clocked at 20 MHz.                   */
/*-----------------------------------------------------------------------*/
/* TLV1571/78 Command Set(CMR):                                          */
/*                                                                       */
/*                Value:                                                 */
/* STARTSEL  { = 0x0001 Software Start                                   */
/*             = 0x0000 Hardware Start    }                              */
/* PROGEOC   { = 0x0000 Interrupt                                        */
/*             = 0x0001 End Of Conversion}                               */
/* CLKSEL    { = 0x0000 Internal Clock                                   */
/*             = 0x0001 External Clock    }                              */
/* SWPDWN    { = 0x0000 Normal Powerdown                                 */
/*             = 0x0001 Powerdown         }                              */
/* MODESEL   { = 0x0000 Single Channel Mode                              */
/*             = 0x0001 Sweep Mode        }                              */
/* CHANNEL   { = 0x0000 Channel 0 or Sweep: CH[O,1]                      */
/*             = 0x0001 Channel 1 or Sweep: CH[0,1,2,3]                  */
/*             = 0x0002 Channel 2 or Sweep: CH[0,1,2,3,4,5]              */
/*             = 0x0003 Channel 3 or Sweep: CH[0,1,2,3,4,5,6,7]          */
/*             = 0x0004 Channel 4                                        */
/*             = 0x0005 Channel 5                                        */
/*             = 0x0006 Channel 6                                        */
/*             = 0x0007 Channel 7 }                                      */
/* OSCSPD    { = 0x0000 Internal OSC Slow                                */
/*             = 0x0001 Internal OSC Fast }                              */
/* OUTCODE   { = 0x0000 Binary Output                                    */
/*             = 0x0001 2's Complement Output }                          */
/* OUTPUT    { = 0x0000 Normal Conversion                                */
/*             = 0x0001 Self Test 1                                      */
/*             = 0x0002 Self Test 2                                      */
/*             = 0x0003 Self Test 3                                      */
/*             = 0x0004 Readback Control Register 1                      */
/*             = 0x0005 Readback Control Register 2 }                    */
```

```
/*                                                                  */
/* NumSamples  = 0x0100                                             */
/* AtMemory    = 0x1200 Range: 1200h to 27FFh                       */
/* SendDAC   { = 0x0000 Do not Send Output to DAC                   */
/*               = 0x0001 Send Output to DAC }                      */
/*------------------------------------------------------------------*/
/*Note:  In Sweep Mode each Table will be NumSamples Long.          */
/* Memory Tables will be placed in increments of NumSamples.       */
/*                                                                  */

extern STARTSEL,PROGEOC, CLKSEL, SWPDWN, MODESEL, CHANNEL, OSCSPD;
extern OUTCODE, OUTPUT, NumSamples, SendDAC, AtMemory;
extern void c1571EVM();
main(void)
{

   STARTSEL    = 0x0001    ;
   PROGEOC     = 0x0000    ;
   CLKSEL      = 0x0000    ;
   SWPDWN      = 0x0000    ;
   MODESEL     = 0x0000    ;
   CHANNEL     = 0x0000    ;
   OSCSPD      = 0x0001    ;
   OUTCODE     = 0x0000    ;
   OUTPUT      = 0x0000    ;
   NumSamples  = 0x0100    ;
   AtMemory    = 0x1200    ;/*RANGE 1200h to 27FFh*/
   SendDAC     = 0x0001    ;

   c1571EVM();
}
```

7 Assembly Source Code

```
*************************************************************************
* TITLE        : TLV1571/78 Interface routine                           *
* FILE         : c1571evm.ASM                                           *
* FUNCTION     : MAIN                                                   *
* PROTOTYPE    : void MAIN ()                                           *
* CALLS        : N/A                                                    *
* PRECONDITION : N/A                                                    *
* POSTCONDITION: N/A                                                    *
* DESCRIPTION  : This program configures the ADC in specified modes,    *
*                Collects and stores the data at the required memory    *
*                location. This code is used when the ADC is clocked    *
*                at 20 MHz                                              *
* AUTHOR       : AAP Application Group, L. Philipose, Dallas            *
*                CREATED 1999(C) BY TEXAS INSTRUMENTS INCORPORATED      *
* REFERENCE    : TMS320C54x User's Guide, TI 1997                       *
*              : Data Aquisation Circuits, TI 1999                      *
*************************************************************************
         .title   "TLV1571/78 C Callable"
;         .mmregs
         .width   80
         .length  55
         .version 542
;        .setsect ".vectors",0x00200,0    ; sections of code
;        .setsect ".text",   0x00300,0    ; these assembler directives specify
;        .setsect ".data",   0x01100h,1   ; the absolute addresses of different
;        .setsect ".variabl",0x01100h,1   ; sections of code
    .sect ".vectors"
    .copy "vectors.asm"

*global Variables
 .global _c1571EVM
 .global _STARTSEL   ;Hardware or Software Start of Conversion
 .global _PROGEOC    ;Interrupt or End of Conversion
 .global _CLKSEL     ;Internal or External Clock
 .global _SWPDWN     ;Normal or Powerdown
 .global _MODESEL    ;Single Channel or Sweep Mode
 .global _CHANNEL    ;Select Channel(s) for single or sweep
 .global _OSCSPD     ;Internal OSC Slow(10MHz) or Fast(20MHz)
 .global _OUTCODE    ;Binary or 2's Complement Code Output
```

```
 .global _OUTPUT     ;Normal Coversion or Self Test[1,2,3] or CR[1,2] output
 .global _NumSamples ;Collect this many samples
 .global _AtMemory   ;store at them memory address
 .global _SendDAC    ;Collect this many samples
*Local Variables
AD_DP          .usect ".variabl",0 ;label
CR0_SEND  .usect  ".variabl",1  ;the  last  value,  sent  to  register  CR0
CR1_SEND    .usect  ".variabl",1  ;the  last  value,  sent  to  register  CR1
TEMP           .usect ".variabl",1 ;temporary variable
ADSAMPLE       .usect ".variabl",1 ;last readed sample of channel 2
NEXT_CH        .usect ".variabl",1 ; last readed sample of channel 1
TABLE_7        .usect ".variabl",1 ; last readed sample of channel 1
_STARTSEL      .usect ".variabl",1 ; Hardware or Software Start of Conversion
_PROGEOC       .usect ".variabl",1 ; Interrupt or End of Conversion
_CLKSEL        .usect ".variabl",1 ; Internal or External Clock
_SWPDWN        .usect ".variabl",1 ; Normal or Powerdown
_MODESEL       .usect ".variabl",1 ; Single Channel or Sweep Mode
_CHANNEL       .usect ".variabl",1 ; Select Channel(s) for single or sweep
_OSCSPD        .usect ".variabl",1 ; Internal OSC Slow(10MHz) or Fast(20MHz)
_OUTCODE       .usect ".variabl",1 ; Binary or 2's Complement Code Output
_OUTPUT        .usect ".variabl",1 ; Normal Coversion or Self Test[1,2,3] or
                                     CR[1,2] output
_NumSamples    .usect ".variabl",1 ; Collect this many samples
_AtMemory      .usect ".variabl",1 ; store at them memory address
_SendDAC       .usect ".variabl",1 ; Collect this many samples
* Address Decoder constants:
ADC            .set   00002h       ; activate A0 when TLV1571 is choosen
CSTART         .set   00001h       ; activate A1 when CSTART is choosen
DAC            .set   00003h       ; activate A2 when DAC1 is choosen
DEACTIVE       .set   00000h       ; deactivate the address lines A0, A1 and A2
    .sect ".text"

_c1571EVM:
_MAIN:
START:
    .copy "macros.asm"
*******************
*Save all Registers
*******************
   push(AR0)
   push(AR1)
```

```
   push(AR2)
   push(AR3)
   push(AR4)
   push(AR5)
   push(AR6)
   push(AR7)
        SXM     = 0                 ; no sign extension mode
* copy interrupt vector table to DSP IRQ Vector table:
        DP      = #1;
        AR7     = #00200h;
        repeat(#3h)
        data(0084h) = *AR7+         ; copy the NMI vector
        AR7     = #00240h
        repeat(#35)
        data(00C0h) = *AR7+         ; copy INT0, INT1,...
* initialize waitstates:
        DP      = #00000h           ; point to page zero
        @SWWSR  = #01000h           ; one I/O wait states
* reset pending IRQs
        IFR     = #1                ; reset any old interrupt on pin INT0
* enable Interrupt INT0
        @IMR   |= #01               ; allow INT0
* enable global interrupt (this is even required, if no IRQ routine is used
* by this program because the debugger needs to do its backgroud interrupts)
         INTM    = 0                ; enable global IRQ
         DP      = #AD_DP



*Intialize local variables
         A  = @_AtMemory            ; point to first date location of the storage
                                     table for channel A
         AR7 = A                    ; AR7 points to the first storage table
         @NEXT_CH=#0
         A = @_NumSamples
         B = @_AtMemory             ; AR0 points to the end of Storage Table for
                                   ; channel A.
         A = A+B
         AR0 =A
         B=#0
         A=#0
*Format register variables to be sent to CR0 and CR1
```

```
         Register_Bit @_STARTSEL,   #7    ; macro
         Register_Bit @_PROGEOC,    #6
         Register_Bit @_CLKSEL,     #5
         Register_Bit @_SWPDWN,     #4
         Register_Bit @_MODESEL,    #3
         Register_Bit @_CHANNEL,    #0
         @CR0_SEND=B                      ; Control Word for CR0 per data sheet.
         A=@CR0_SEND
         A=A<<<6    ;left shift
         @CR0_SEND=A                      ; Control Word for CR0 mapped MSB-MSB
                                          ; on the EVM.


         B=#100h
         Register_Bit @_OSCSPD,     #6
         Register_Bit @_OUTCODE,    #3
         Register_Bit @_OUTPUT,     #0
         @CR1_SEND=B                      ; Control Word for CR1.
         A=@CR1_SEND
         A=A<<<6     ;left shift          ; Left shifted before EVM is mapped
                                          ; MSB-MSB.
         @CR1_SEND=A
*****************************************
*This block of code determines whether the user wants a Hardware Start or a
 Software Start, then decides whether the user wants the Single channel Mode or the
 Sweep Mode.
*****************************************
   push(AR0)
   AR0=data(@_STARTSEL)
   AR6=#1h
   TC=(AR0==AR6)
   AR0=pop()
   if (TC) goto Software
*Hardware Sweep Mode
Hardware:
   push(AR0)
   AR0=data(@_MODESEL)
   AR6 =#1h
   TC=(AR0==AR6)
   AR0=pop()
   if (TC) goto HardSweep
   goto Single_Hard
Software:
```

```
   push(AR0)
   AR0=data(@_MODESEL)
   AR6=#1h
   TC=(AR0==AR6)
   AR0=pop()
   If (TC) goto SoftSweep
   goto Single_Soft
****************************************
*Hardware Start Sweep Mode
****************************************
HardSweep:
       .copy ”SweepH.asm”
   goto Return_to_C
****************************************
*Software Start Sweep Mode
****************************************
SoftSweep:
   .copy ”SweepS.asm”
   goto Return_to_C
****************************************
*Single Channel Hardware Start Mode
****************************************
Single_Hard:
   .copy ”SingleH.asm”
   goto Return_to_C
****************************************
*Single Channel Software Start Mode
****************************************
Single_Soft:
   .copy ”SingleS.asm”
   goto Return_to_C
****************************************
*Restore all Registers
****************************************
Return_to_C:

   AR7=pop()
   AR6=pop()
   AR5=pop()
   AR4=pop()
```

```
   AR3=pop()
   AR2=pop()
   AR1=pop()
   AR0=pop()
   return
ERROR_Go_Back:       ;if User inputs wrong configuration.
   A=#1
   return
*************************************************************
* IRQ_INT0:
*   Interrupt routine of the external interrupt input pin INT0
*************************************************************
IRQ_INT0:
   return_fast ; return fast from IRQ (wake up from the IDLE mode)
    .end
*****************************************************************************
* TITLE         : TLV1571/78 Interface routine                                *
* FILE          : SingleH.ASM                                                 *
* FUNCTION      : N/A                                                         *
* PROTOTYPE     : N/A                                                         *
* CALLS         : N/A                                                         *
* PRECONDITION  : N/A                                                         *
* POSTCONDITION : N/A                                                         *
* DESCRIPTION   : This assembly program is written for C54x. It is included   *
*  in c1571EVM.asm   Program configures and runs ADC in Hardware Start Single *
*  Channel Mode.                                                              *
* AUTHOR        : AAP Application Group, L. Philipose, Dallas                 *
*                 CREATED 1999(C) BY TEXAS INSTRUMENTS INCORPORATED.          *
* REFERENCE     : TMS320C54x User's Guide, TI 1997                            *
*               : Data Acquisition Circuits, TI 1999                          *
*****************************************************************************
***********************************
* Initialize ADC control Registers *
*   set ADC registers: CR0,CR1     *
***********************************
shADC_INI:
shSTEP1:
* write CR0:
         port(ADC) = @CR0_SEND         ;Address decoder sets CS low,
                                       ;WR- low and send CR1 value to the ADC
         NOP                           ;
         NOP                           ;wait for tW(CSH)=50ns
* write CR1
         port(ADC) = @CR1_SEND         ;send CR0 value to the ADC
         NOP                            ;
         NOP                           ;wait for tW(CSH)=50ns

*First conversion cycle
         @TEMP= port(CSTART)           ;clear CSTART- (CSTARTlow)
                                       ;begin sampling
*Begin Conversion
         @TEMP= port(DEACTIVE)         ;set CSTART- (CSTARThigh)
         repeat(#16)
         NOP
***************************************
```

```
* ADC_CStart Single Channel:
*   Read Sample
*   Send Sample to DAC
*   Store Sample into memory
****************************************

STEP3:
         XF        = 0                 ;clear RD
* read sample
STEP4:   DP        = #ADSAMPLE       ;point to ADSAMPLE
         @ADSAMPLE = port(ADC)       ;read the new sample into the DSP
         XF        = 1               ;set RD
         nop                         ;C542 DSK board introduces a delay of the RD
                                     ;signal (~30 ns). If a chip select is issued
                                     ;immediately after RD, then chip select goes
                                     ;low before read (RD) because of this delay. To
                                     ;remedy this problem a NOP is required.
*Begin Sample
STEP5:   @TEMP  = port(CSTART)       ;clear CSTART- (CSTARTlow)

*Begin Conversion
STEP6:   @TEMP = port(DEACTIVE)      ;set CSTART- (CSTARThigh)

         TC =(@_SendDAC == #1h)
         if (NTC) goto NO_DAC
*Send out to DAC
         port(DAC1) = @ADSAMPLE      ;Address Decoder selects DAC: CSz low, WRZ-low

NO_DAC:

*Store In Table
STEP7:   *AR7+   = data(@ADSAMPLE)   ;write last sample of channel into memory table

         TC       = (AR0 == AR7)     ;is AR7 = AR0? (table end reached?)
         if (TC) goto Return_to_C

CONTINUE:
         goto     STEP3              ;go back to receive next sample



 ***************************************************************************
* TITLE        : TLV1571/78 Interface routine                                *
* FILE         : SingleS.ASM                                                 *
* FUNCTION     : N/A                                                         *
* PROTOTYPE    : N/A                                                         *
* CALLS        : N/A                                                         *
* PRECONDITION : N/A                                                         *
* POSTCONDITION: N/A                                                         *
* DESCRIPTION  : This assembly program is written for C54x. It is included   *
*  in c1571EVM.asm Program configures and runs ADC in Software Start Single  *
*  Channel Mode: This source code is written for the ADC clocked at 20 MHz.  *
* AUTHOR       : AAP Application Group, L. Philipose, Dallas                 *
*                 CREATED 1999(C) BY TEXAS INSTRUMENTS INCORPORATED.         *
* REFERENCE    : TMS320C54x User's Guide, TI 1997                            *
*              : Data Acquisition Circuits, TI 1999                          *
***************************************************************************
*Configure ADC
* write CR0:
       port(ADC) = @CR0_SEND         ;Address decoder sets CS low,
                                     ;WR low and send CR1 value to the ADC
        NOP                          ;
```

```
        NOP                          ;wait for tW(CSH)=50ns

* write CR1
         port(ADC) = @CR1_SEND       ;send CR0 value to the ADC
         @TEMP = port(DEACTIVE)       ;send CR0 value to the ADC
         repeat(#22)                 ;TEST 800ns
         NOP                          ;wait for t(SAMPLE1)=100ns

*******************************************
* ADC_Software Single Channel:              *
*  read samples and store them into memory   *
*******************************************
ADC_Soft:
* read sample
SwSTEP4: XF          = 0             ;clear RD
SwSTEP6: DP          = #ADSAMPLE     ;point to ADSAMPLE
         @ADSAMPLE   = port(ADC)     ;read the new sample into the DSP
         XF          = 1             ;set RD
         nop                         ;C542 DSK board introduces a delay of the RD
                                     ;signal (~30 ns). If a chip select is issued
                                     ;immediately after RD, then chip select goes
                                     ;low before read (RD) because of this delay. To
                                     ;remedy this problem a NOP is required.
SwSTEP7: @TEMP     = port(DEACTIVE)
         repeat(#3)      ;wait for t(CONV1)
         NOP
         TC =(@_SendDAC==#1h)
         if (NTC) goto SwNO_DAC

*Send out to DAC
SwSTEP8: port(DAC1) = @ADSAMPLE     ;Address Decoder selects DAC: CSz low, WR low

SwNO_DAC:

*Store In Table
SwSTEP9: *AR7+   = data(@ADSAMPLE)  ;write last sample of channel into memory table

         TC     = (AR0 == AR7)      ;is AR7 = AR0? (table end reached?)
         if (TC) goto Return_to_C
         goto    SwSTEP4             ;go back to receive next sample
*************************************************************************
* TITLE        : TLV1571/78 Interface routine                           *
* FILE         : SweepS.ASM                                             *
* FUNCTION     : N/A                                                    *
* PROTOTYPE    : N/A                                                    *
* CALLS        : N/A                                                    *
* PRECONDITION : N/A                                                    *
* POSTCONDITION: N/A                                                    *
* DESCRIPTION   : This assembly program is written for C54x. It is included  *
*  in c1571EVM.asm.  Program configures and runs ADC in Software Start Sweep  *
*  Channels Mode. Note the source code is meant for the ADC clocked at 20 MHz *
* AUTHOR       : AAP Application Group, L. Philipose, Dallas            *
*                  CREATED 1999(C) BY TEXAS INSTRUMENTS INCORPORATED.   *
* REFERENCE    : TMS320C54x User's Guide, TI 1997                       *
*              : Data Acquisition Circuits, TI 1999                     *
*************************************************************************
* fill all locations between 1200h and 27FFh with 1234h:
         DP          = #AD_DP        ;
         @TEMP       = #01234h       ;


         A  = @_AtMemory
         INIT_TABLE AR7              ;initialize CH0 Table
```

```
         A = A + @_NumSamples       ;Start of Next Table
         INIT_TABLE AR6             ;initialize CH1 Table

         TC=(@_CHANNEL==#0h)     ;Sweep Sequence 0?
         if (TC) goto SwS_CONT

         A = A + @_NumSamples    ;Start of Next Table
         INIT_TABLE AR5          ;initialize CH2 Table

         A = A + @_NumSamples    ;Start of Next Table
         INIT_TABLE AR4          ;initialize CH3 Table

         TC=(@_CHANNEL== #1h)    ;Sweep Sequence 1?
         if (TC) goto SwS_CONT

         A = A + @_NumSamples    ;Start of Next Table
         INIT_TABLE AR3          ;initialize CH4 Table

         A = A + @_NumSamples    ;Start of Next Table
         INIT_TABLE AR2          ;initialize CH5 Table

         TC=(@_CHANNEL==#2h)     ;Sweep Sequence 2?
         if (TC) goto SwS_CONT

         A = A + @_NumSamples    ;Start of Next Table
         INIT_TABLE AR1          ;initialize CH6 Table

         push(AR0)
         A = A + @_NumSamples    ;Start of Next Table
         INIT_TABLE AR0          ;initialize CH7 Table
         AR0=pop()
         @TABLE_7  = A

SwS_CONT:

*********************************
* Initialize ADC control Registers
*   set ADC registers: CR0,CR1
*********************************
SwS_ADC_INI:
* write CR0:
         port(ADC) = @CR0_SEND         ;Address decoder sets CS low,
                                       ;WR low and send CR1 value to the ADC
         NOP                           ;wait for tW(CSH)=50 ns
         NOP
* write CR1
         port(ADC) = @CR1_SEND         ;send CR0 value to the ADC
SwS_STEP1:
         @TEMP=port(DEACTIVE)          ;deselect ADC (CShigh)
SwS_STEP2:
         repeat(#24)                   ;TEST 800ns
         NOP                           ;wait for t(SAMPLE1)=100ns

****************************************
* ADC Software Start Sweep Channels
*   read samples and store them into memory
****************************************
ADC_SSweep:
SwS_STEP4:
            XF       = 0              ;clear RD

* read sample
```

```
SwS_STEP6:
         DP       = #ADSAMPLE       ;point to ADSAMPLE
         @ADSAMPLE= port(ADC)       ;read the new sample into the DSP
         nop                        ;C542 DSK board introduces a delay of the RD
                                    ;signal (~30 ns). If a chip select is issued
                                    ;immediately after RD, then chip select goes
                                    ;low before read (RD) because of this delay. To
                                    ;remedy this problem a NOP is required.
         XF       = 1               ;set RD
         @TEMP    = port(DEACTIVE)


*********************************
* STORE:
*   saving the samples into memory
*********************************

      INIT_STORE  AR0               ;PASS AR0

* test for table end, set pointer back if true
        TC      = (AR0 == AR7)      ;is AR7 = AR0? (table end reached?)
        if (TC) goto Return_to_C     ;
        goto    SwS_STEP4            ;go back to receive next sample

************************************************************************
* TITLE        : TLV1571/78 Interface routine                          *
* FILE         : SweepH.ASM                                            *
* FUNCTION     : N/A                                                   *
* PROTOTYPE    : N/A                                                   *
* CALLS        : N/A                                                   *
* PRECONDITION : N/A                                                   *
* POSTCONDITION: N/A                                                   *
* DESCRIPTION  : This assembly program is written for C54x. It is included *
*  in c1571EVM.asm Program configures and runs ADC in Hardware Start Sweep *
*  Channels Mode. The following source code written for ADC clocked at     *
*                 20 MHz                                               *
* AUTHOR       : AAP Application Group, L. Philipose, Dallas           *
*                  CREATED 1999(C) BY TEXAS INSTRUMENTS INCORPORATED.  *
* REFERENCE    : TMS320C54x User's Guide, TI 1997                      *
*              : Data Acquisition Circuits, TI 1999                    *
************************************************************************
* fill all table locations between FFFFh:
        DP          = #AD_DP        ;
        @TEMP       = #0FFFFh       ;

* initialize storage table for the ADC samples

      A  = @_AtMemory

      INIT_TABLE AR7                ;initialize CH0 Table

      A = A + @_NumSamples          ;Start of Next Table
      INIT_TABLE AR6                ;initialize CH1 Table

      TC=(@_CHANNEL==#0h)           ;Sweep Sequence 0?
      if (TC) goto SwH_CONT

      A = A + @_NumSamples          ;Start of Next Table
      INIT_TABLE AR5                ;initialize CH2 Table

      A = A + @_NumSamples          ;Start of Next Table
      INIT_TABLE AR4                ;initialize CH3 Table
```

```
      TC=(@_CHANNEL== #1h)              ;Sweep Sequence 1?
      if (TC) goto SwH_CONT

      A = A + @_NumSamples              ;Start of Next Table
      INIT_TABLE AR3                    ;initialize CH4 Table

      A = A + @_NumSamples              ;Start of Next Table
      INIT_TABLE AR2                    ;initialize CH5 Table

      TC=(@_CHANNEL==#2h)               ;Sweep Sequence 2?
      if (TC) goto SwH_CONT

      A = A + @_NumSamples              ;Start of Next Table
      INIT_TABLE AR1                    ;initialize CH6 Table

      push(AR0)
      A = A + @_NumSamples              ;Start of Next Table
      INIT_TABLE AR0                    ;initialize CH7 Table

      AR0=pop()
      @TABLE_7  = A

SwH_CONT:

*********************************
* Initialize ADC control Registers
*   set ADC registers: CR0,CR1
*********************************
SwH_ADC_INI:
SwH_STEP1:
* write CR0:
         port(ADC) = @CR0_SEND          ;Address decoder sets CS low,
                                        ;WR low and send CR1 value to the ADC
         NOP                            ;
         NOP                            ;wait for tW(CSH)=50ns

* write CR1
         port(ADC) = @CR1_SEND          ;send CR0 value to the ADC
SwH_STEP1_5:
         @TEMP= port(CSTART)            ;clear CSTART (CSTARTlow)

*Begin Conversion
SwH_STEP2:
         @TEMP = port(DEACTIVE)         ;set CSTART (CSTARThigh)

         repeat(#24)                    ;
         NOP                            ;wait for t(CONV1)

  ****************************************
* ADC_CStart Single Channel:
*   Read Sample
*   Send Sample to DAC
*   Store Sample into memory
****************************************
* read sample

SwH_STEP3:
         XF = 0                    ;clear RD

SwH_STEP4:
         DP= #ADSAMPLE             ;point to ADSAMPLE
         @ADSAMPLE= port(ADC)      ;read the new sample into the DSP
         XF=1                      ;set RD
```

```
         nop                          ;C542 DSK board introduces a delay of the RD
                                      ;signal (~30 ns). If a chip select is issued
                                      ;immediately after RD, then chip select goes
                                      ;low before read (RD) because of this delay. To
                                      ;remedy this problem a NOP is required.
*Begin Sample
SwH_STEP5:
      @TEMP = port(CSTART)            ;clear CSTART (CSTARTlow)


*Begin Conversion
SwH_STEP6:
      @TEMP= port(DEACTIVE)           ;set CSTART (CSTARThigh)

*********************************
* STORE:
*   saving the samples into memory
*********************************
SwH_STORE:

      INIT_STORE  AR0                 ;pass AR0

* test for table end, set pointer back if true
      TC      = (AR0 == AR7)          ;is AR7 = AR0? (table end reached?)
      if (TC) goto Return_to_C
      goto    SwH_STEP3               ;go back to receive next channel data

************************************
 FILE: macros.asm                    *
*DESCRIPTION: Macro Routines         *
************************************
*Format register bits for ADC configuration
Register_Bit   .macro   Var, NUM
      A=Var
      A=A<<<NUM
      B=A|B
        .endm
*Initialize Memory Table
INIT_TABLE  .macro   ARx
   ARx= A      ;
   B=@_NumSamples
B=B-#1
   BRC=B
   NOP
   blockrepeat(End_Block?-1)
   *ARx+   = data(@TEMP)              ;fill table with FFFFh
End_Block?:
   ARx  = A
    .endm
* Initialize data storage
INIT_STORE  .macro   ARx              ;pass AR0

* store new sample
   TC=(@NEXT_CH==#0)                  ;NEXT_CH=0 channel 1, NEXT_CH=1 channel 1
   if (TC) goto CHANNEL_0?
   TC=(@NEXT_CH==#1)                  ;NEXT_CH=2 channel 2, NEXT_CH=3 channel 3
   if (TC) goto CHANNEL_1?
   TC=(@NEXT_CH==#2)
   if (TC) goto CHANNEL_2?
   TC=(@NEXT_CH==#3)
   if (TC) goto CHANNEL_3?
   TC=(@NEXT_CH==#4)
   if (TC) goto CHANNEL_4?
```

```
   TC=(@NEXT_CH==#5)
   if (TC) goto CHANNEL_5?
   TC=(@NEXT_CH==#6)
   if (TC) goto CHANNEL_6?
   TC=(@NEXT_CH==#7)
   if (TC) goto CHANNEL_7?

CHANNEL_0?:
   *AR7+   = data(@ADSAMPLE)        ;write last sample of channel 1 into memory
                                    ;table
   @NEXT_CH = #1
   goto CONTINUE?

CHANNEL_1?:
   *AR6+   = data(@ADSAMPLE)        ;write last sample of channel 2 into memory ;
                                    ;table
   @NEXT_CH = #2
   TC=(@_CHANNEL==#0h)              ;Sweep only CH0, CH1?
   if (TC) goto Reset_Sweep1?
   goto CONTINUE?

Reset_Sweep1?:
   @NEXT_CH = #0
   goto CONTINUE?

CHANNEL_2?:
   *AR5+   = data(@ADSAMPLE)        ;write last sample of channel 3 into memory
                                    ;table
   @NEXT_CH=#3
   goto CONTINUE?

CHANNEL_3?:
   *AR4+   = data(@ADSAMPLE)        ;write last sample of channel 2 into memory
                                    ;table
   @NEXT_CH=#4
   TC=(@_CHANNEL==#1h)              ;weep only CH0, CH1,ch2,ch3?
   if (TC) goto Reset_Sweep2?
   goto CONTINUE?

Reset_Sweep2?:
   @NEXT_CH = #0
   goto CONTINUE?

CHANNEL_4?:
   *AR3+   = data(@ADSAMPLE)        ;write last sample of channel 2 into memory
                                    ;table
   @NEXT_CH=#5
   goto CONTINUE?

CHANNEL_5?:
   *AR2+   = data(@ADSAMPLE)        ;write last sample of channel into memory table
   @NEXT_CH=#6
   TC=(@_CHANNEL==#2h)              ;weep only CH0, CH1,ch2,ch3,ch4,ch5?
   if (TC) goto Reset_Sweep3?
   goto CONTINUE?
Reset_Sweep3?:
   @NEXT_CH = #0
   goto CONTINUE?

CHANNEL_6?:
   *AR1+   = data(@ADSAMPLE)        ;write last sample of channel into memory table
   @NEXT_CH=#7
   goto CONTINUE?
```

```
CHANNEL_7?:
   push(ARx)
   B=@TABLE_7
   NOP                                  ;NEED BECAUSE OF PIPELINE DELAY
   AR0=B
   NOP                                  ;NEED BECAUSE OF PIPELINE DELAY
   NOP                                  ;NEED BECAUSE OF PIPELINE DELAY
   *ARx+   = data(@ADSAMPLE)            ;write last sample of channel into memory
                                        ;table
   @NEXT_CH=#0
   ARx=pop()
CONTINUE?:
   DP= #AD_DP                           ;TEST
   .endm
```

8 References

1. *TMS320C54X DSP CPU and Peripherals Reference Set Volume I*, Literature Number SPRU131F
2. *TMS320C54X DSP CPU and Peripherals Reference Set Volume 3*: *Algebraic Instruction*, Literature Number SPRU179B
3. *TMS320C54X Assembly Language Tools User's Guide*, Literature Number SPRU102C
4. *TMS320C54X DSKPLUS DSP Starter Kit User's Guide*, Literature Number SPRU191
5. *TLV1571/TLV1578 10-BIT 1.25 MSPS Parallel Analog-to-Digital Converter*, Literaure Number SLAS170
6. *Characteristics, Operation, and Use of the TLV157x EVM*, Literature Number SLAU025

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265