

# **Interfacing the TLV2544/TLV2548 ADC to the TMS320C5402 DSP**

*Lijoy Philipose*
*AAP Data Conversion*

## **ABSTRACT**

This application report presents a hardware and software solution for interfacing the TLV2544/TLV2548 12-bit, 200-ksps, 4/8-channel, low-power, serial analog-to-digital converters to the 16-bit fixed-point TMS320C5402 DSK.

## **Contents**

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Hardware Interface</b>	<b>2</b>
2.1	TMS320C5402 DSK Starter Kit	2
2.2	TLV2544/TLV2548 EVM	3
<b>3</b>	<b>The Software</b>	<b>4</b>
3.1	DSP	4
3.1.1	Interrupts	4
3.2	CPLD	5
3.2.1	CPLD Registers Setting	5
3.2.2	DSP CNTL1 Register (I/O Address 0x0000)	5
3.2.3	DSP CNTL2 Control Register (I/O Address 0x0004)	6
3.3	McBSP	6
3.4	TLV2544/TLV2548 Configuration	12
3.5	TLV2544/TLV2548 Conversion Modes	13
3.5.1	Sweep Mode	14
3.5.2	Repeat Mode	16
3.6	Repeat Sweep Mode and Single Shot Mode	17
<b>4</b>	<b>References</b>	<b>19</b>
<b>Appendix A</b>	<b>TLV2548 ADC C Program Main Routine</b>	<b>20</b>
<b>Appendix B</b>	<b>Single Shot Conversion Mode</b>	<b>21</b>
<b>Appendix C</b>	<b>Repeat Mode Conversion Mode</b>	<b>27</b>
<b>Appendix D</b>	<b>Sweep Mode Conversion Mode</b>	<b>34</b>
<b>Appendix E</b>	<b>Repeat Sweep Conversion</b>	<b>41</b>
<b>Appendix F</b>	<b>McBSP Memory Mapped Register Definition</b>	<b>50</b>
<b>Appendix G</b>	<b>Interrupt Vector Table</b>	<b>52</b>
<b>Appendix H</b>	<b>Program and Data Memory Mapping</b>	<b>56</b>

## List of Figures

1	Interrupt Vector Mapping in Program Memory Space .....	4
2	McBSP Programming .....	8
3	McBSP Control Register Bit Features .....	11
4	TLV254x Configuration and Conversion Cycle .....	14
5	TLV254x Conversion and FIFO Read Cycles .....	14
6	Sweep Mode Flowchart .....	15
7	Repeat Sweep Mode Flow Chart .....	17
8	Repeat Sweep Mode Flow Chart .....	18
9	Single Shot Mode Flow Chart .....	19

## List of Tables

1	DSP CPLD CNTL Register Bit Definitions .....	5
2	DSP CPLD CNTL 2 Control Register Bit Definitions .....	6
3	McBSP0 Registers Modified in This Application Report .....	7
4	TLV2544/TLV2548 Command Set .....	12
5	TLV2544/TLV2548 Configuration Register Bit Definitions .....	13

## 1 Introduction

Texas Instruments provides a wide variety of digital signal processors (DSP). This application focuses on the TMS320C5402 DSP. The TMS320C5402 offers 16 kilowords of on-chip memory, up to 100 MIPS of performance, and two McBSP (multichannel buffered serial port) interfaces. The TMS320C5402 DSK gives new DSP designers access to the industry's most powerful DSP, specifically optimized for applications that need the best combination of power and performance. This affordable DSK is used in this report to interface to the TLV2544/TLV2548 ADC.

The TLV2544/TLV2548 devices offer a wide range of programming capabilities. These 12-bit analog-to-digital (ADC) converters operate at low power and provide multiple analog inputs, wide single-supply range, hardware controlled and programmable sampling period, and hardware and software control of power-down features. These features make the ADCs able to adapt to many application needs.

This report provides an overview of the hardware interface, and an in-depth look at the software interface. The sample programs attached will allow the user to quickly get familiar with this family of data converters.

## 2 Hardware Interface

The hardware interface consists of the TMS320C5402 DSK and the TLVX544 /2548 EVM.

### 2.1 TMS320C5402 DSK Starter Kit

The 'C5402 DSK is designed specially for digital communications applications and comes complete with a TMS320C5402-based target board, DSK-specific code composer studio debug tools, 32K application size limited C compiler/assembler/linker, parallel port interface, power supply, and cables.

The 'C5402 device features a 100-MHz clock, a 40-bit ALU, 16K x 16-bit dual access on chip RAM, 4K x 16-bit on chip ROM, advanced multibus architecture with three separate 16-bit data memory busses, and one program memory bus. The onboard parallel port controller allows the host PC to use the parallel port for emulation, or for direct access to the host port interface of the 'C5402. The 'C5402 also provides an onboard standard JTAG interface connection for optional emulation, expansion connectors for add-on accessories. Texas Instruments now provides expansion connectors or adapter boards for all data converter EVMs to interface to the 'c5402 DSK.

The enhanced peripheral of particular interest in this report is the McBSP serial port. The McBSP is explained in Chapter 3.

The 'c5402 DSK supports a TMS320VC5402 DSP which can operate at a frequency of up to 100 MHz with a core voltage of 1.8 V and an I/O voltage of 3.3 V. The DSK provides support for all the DSP interfaces and control signals. The JTAG emulation interface is used to support both embedded and external JTAG emulations. The control interface is used to reset the device and to provide external interrupts. The McBSP0 is, by default, used to interface to the telephone DAA circuit. This port is also available to the daughterboard via an onboard multiplexer. The McBSP1 is, by default, used for the microphone/speaker interfaces, and brought to the peripheral expansion connector for daughterboard use. The complex programmable logic device (CPLD) controls the input of McBSP0 and McBSP1.

## 2.2 TLV2544/TLV2548 EVM

The TLV2544/TLV2548 evaluation module (EVM) provides a platform for evaluating the TLV2544 and TLV2548 12-bit analog-to-digital converters under various signal, reference, and supply conditions.

The TLV2544 has four analog inputs: A0, A1, A2, and A3. While the TLV2548 has eight analog input channels: A0 through A7. The applied signal connects to these inputs as follows:

- A0: a TLV2272 single-supply op amp is used to scale the input analog signal. The dynamic range of the output is compressed to a level between 0 V and 5 V over a wide range of input levels (0 V to  $\pm 10$  V, and zero dc input offset). Note that there is a 2.5-V offset voltage produced by the node voltage divider (R5/R6) multiplied by the amplifier gain. The offset is used to convert the bipolar input to a single-ended input required by the ADC. Therefore, it is assumed that the analog input signal is free of dc offset. If the incoming signal has a dc offset, then the ADC REF<sup>-</sup> voltage should be adjusted to the same value as the input signal dc offset voltage, and the REF<sup>+</sup> should be raised by the same amount. A second TLV2772 op amp is used to buffer the A0 input signal.
- A1: a TLE2142 dual-supply op amp is used to scale the wide ranging (0 V to  $\pm 10$  V) signal connected to the A1 input. The op amp receives external input through J2.
- A2–A7: these are available for user-defined inputs. The external inputs to these channels are applied through J1 and J4 through J7, respectively. It is important to note that each input channel has a low-pass filter. The filter's RC components can be adjusted to suit the application.

The user should refer to the TLVX544/2548EVM user's guide for more detailed explanations of the various circuits contained in this EVM.

### 3 The Software

The DSP, McBSP, and CPLD devices must be correctly initialized before attempting to read or write to the data converter. The following sections explain the sample code included in the appendixes.

#### 3.1 DSP

The 'VC5402 DSK provides the DSP with a single 20-Mhz frequency reference via the DSP built-in crystal oscillator. The DSP clock-mode pins are configured via DIP switch settings to allow for a number of different frequencies, up to the part's maximum rate of 100 MHz. The CLKMD register can also be changed after reset to vary the DSP's operating frequency.

```
CLKMD = #4007 ;Sets the CPU CLOCK FREQ = 100 MHz
```

##### 3.1.1 Interrupts

Interrupts are hardware or software driven signals that cause the DSP to suspend its main program and execute another function called an interrupt service routine. The TLV2544/TLV2548 devices can be programmed to generate an interrupt when a conversion is completed, or when the FIFO is filled. Using interrupt service routines (ISR) allows the DSP to perform other functions, until an off-chip device needs attention.

When an interrupt is detected, the DSP checks to see if any interrupts are enabled; if it finds that a particular interrupt is enabled, it looks in the interrupt vector table for the next instruction, which is typically a jump instruction to an interrupt service routine.

The interrupt vectors table can be remapped to the beginning of any 128-word page in program memory, except in the reserved areas. At reset, the interrupt vector pointer (IPTR) bits are set to 1 (IPTR = 1FFh); this value maps the vectors to page 511 in program-memory space (see Figure 1). Therefore, the reset vector for hardware reset always resides at location 0FF80h. The interrupt vectors can be mapped to another location by loading the IPTR with a value other than 1FFh. In this report, the vector table is stored at 0080h in program-memory space. Therefore, IPTR is set to 0001h, which remaps the interrupt vector table to begin at 0080h in program-memory space. IPTR is a 7-bit field in the processor mode status register (PMST).

```
PMST=#00F8h ;The Interrupt Vector Table is remapped to 0x0080.
;User-defined interrupt vector table is stored starting at 0x0080.
```

Program Memory Address

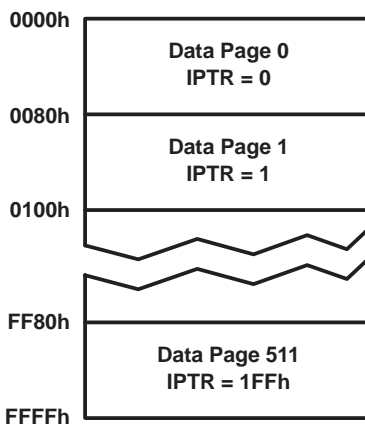


Figure 1. Interrupt Vector Mapping in Program Memory Space

The EVM design maps ADC interrupt to external interrupt 3 ( $\overline{\text{INT3}}$ ) on the DSP. In this report, interrupt 3 is the only interrupt.

```
IMR=#0100h ;Enable interrupt 3 only.
```

Interrupts and interrupt service routines are explained further in *TMS320C54X DSP CPU and Peripherals Reference Set*, Volume I.

## 3.2 CPLD

Seven DSP CPLD registers are mapped to the DSP's lower I/O address space starting at address 0x0000. Only CNTL1 and CNTL2 are of interest in this application report.

### 3.2.1 CPLD Registers Setting

Control register 1 (CNTL1) should be set to enable the daughterboard signal  $\overline{\text{INT3}}$  so that it is transferred to the DSP. This is accomplished by setting CNTL1 = #0008h. This register is mapped in I/O space 0h; therefore, the port(#0) instruction is used to access the register.

```
For example: port(#0) = #0008h
```

Control register 2 (CNTL2) needs to be set to enable the daughterboard as the source for McBSP0. Therefore, set CNTL2 = #0001h. This register is mapped to I/O space 4h; therefore, the port(#4) instruction is used to write to this register.

```
For example: port(#04) = #0001h
```

### 3.2.2 DSP CPLD CNTL1 Register (I/O Address 0x0000)

Control register 1 controls daughterboard signals, interrupts, and user LED's. INT3SEL selects between the host (= 0, default) and the daughtercard as source of the DSP's  $\overline{\text{INT3}}$  signal. This bit needs to be set so that the daughterboard (EVM) is used as the source of the DSP  $\overline{\text{INT3}}$  signal. Table 1 summarizes the function of each bit in the CNTL1 register.

**Table 1. DSP CPLD CNTL1 Register Bit Definitions (bold indicates default)**

BIT	NAME	R/W	DESCRIPTION
7	INT2SEL	RW	Select bootstrap or daughterboard as source for INT2 ( <b>0 = boot</b> , 1 = daughterboard)
6	DB_RST	RW	Daughterboard reset ( <b>0 = no reset</b> , 1 = reset)
5	DB_INT	RW	Daughterboard reset ( <b>0 = no interrupt</b> , 1 = interrupt)
4	NMIEN	RW	DSP NMI enable ( <b>0 = disable NMI</b> , 1 = enable NMI)
3	INT3SEL	RW	Select host or daughterboard as source for $\overline{\text{INT3}}$ ( <b>0 = host</b> , 1 = daughterboard)
2	USERLED2	RW	User-defined LED 2 control ( <b>0 = off</b> , 1 = on)
1	USERLED1	RW	User-defined LED 1 control ( <b>0 = off</b> , 1 = on)
0	USERLED0	RW	User-defined LED 0 control ( <b>0 = off</b> , 1 = on)

### 3.2.3 DSP CPLD CNTL2 Control Register (I/O Address 0x0004)

Control register 2 allows the software to control the source of data for both McBSPs, which defaults to the onboard sources. In this application report, McBSP0 is set to use the daughterboard as its source of data. Table 2 shows the register bit definitions.

**Table 2. DSP CPLD CNTL 2 Control Register Bit Definitions**

BIT	NAME	R/W	DESCRIPTION
7	DAAOH	RW	DAA off-hook control ( <b>0 = on hook</b> , 1 = off hook)
6	DAACID	RW	DAA caller ID enable ( <b>0 = disabled</b> , 1 = enabled)
5	FLASHENB	RW	Select <b>FLASH (=1)</b> or SRAM(=0) for external memory ( <b>1</b> )
4	INT1SEL	RW	Interrupt 1 source selection ( <b>0 = UART</b> , 1 = daughterboard)
3	FC1CON	RW	MIC/speaker AD50 FC control bit ( <b>0</b> )
2	FC0CON	RW	DAA AD50 FC control bit ( <b>0</b> )
1	BSPSEL1	RW	McBSP1 select control ( <b>0 = Mic/Speaker</b> , 1 = daughterboard)
0	BSPSEL0	RW	McBSP0 select control ( <b>0 = TelSet DAA</b> , 1 = daughterboard)

## 3.3 McBSP

The McBSP is based on the standard serial-port interface found on earlier DSPs from Texas Instruments. The McBSP is a high-speed, full-duplex, multichannel buffered serial port that allows direct interface to 'c54xx devices, codecs, data converters, and other devices in a system. In addition, the McBSP offers double-buffered registers that allow continuous data stream and independent framing and clocking for receiver and transmitter. This flexible device gives the user unprecedented control over the serial interface.

To successfully use the McBSP, the user must become familiar with the sample rate generator, the transmitter, and the receiver.

The sample rate generator is composed of a three-stage clock divider that allows programmable data clocks (CLKG) and framing signals (FSG); these are McBSP internal signals that can be programmed to drive receive and/or transmit clocking (CLKR/X) and framing (FSR/X). The sample rate generator can be driven by an internal clock source, or by an internal clock derived from an external clock source.

The sample rate generator registers (SRGR[1,2]) control the operation of the various functions of the sample rate generator. These registers are used to control the width of the frame-sync pulse, whether the frame-sync is an external input driven by the sample rate generator, or a signal that indicates that data from DXR[1,2] to XSR[1,2] has been copied. These registers control whether the sample rate generator clock is derived from the CPU clock or from the CLKS pin, and the division factor necessary to produce the desired serial clock (CLKX/R).

The transmitter and receiver on the McBSP can be programmed for multiple-frame lengths and word lengths. The transmit-control registers (XCR[1,2]) and receive-control registers (RCR[1,2]) determine the mode of the transmitter and receiver. Frame length can be defined as the number of serial words transferred per frame. A serial-word length can be 8-bit, 12-bit, 16-bit, 20-bit, 24-bit, or 32-bit long. Table 3 represents a map of all the McBSP0 registers used in this application report. The DSP contains two McBSP ports: McBSP0 and McBSP1.

**Table 3. McBSP0 Registers Modified in This Application Report**

McBSP0	SUBADDRESS	ACRONYM	REGISTER NAME
0x0021		DRR1	McBSP data-receive register 1
0x0023		DXR1	McBSP data-transmit register 1
0x0038		SPSA	McBSP subaddress register
0x0039	0x0000	SPCR1	McBSP serial port Control Register 1
	0x0001	SPCR2	McBSP serial port Control Register 2
	0x0002	RCR1	McBSP receive Control Register 1
	0x0003	RCR2	McBSP receive Control Register 2
	0x0004	XCR1	McBSP transmit Control Register 1
	0x0005	XCR2	McBSP transmit Control Register 2
	0x0006	SRGR1	McBSP sample rate generator register 1
	0x0007	SRGR2	McBSP sample rate generator register 2

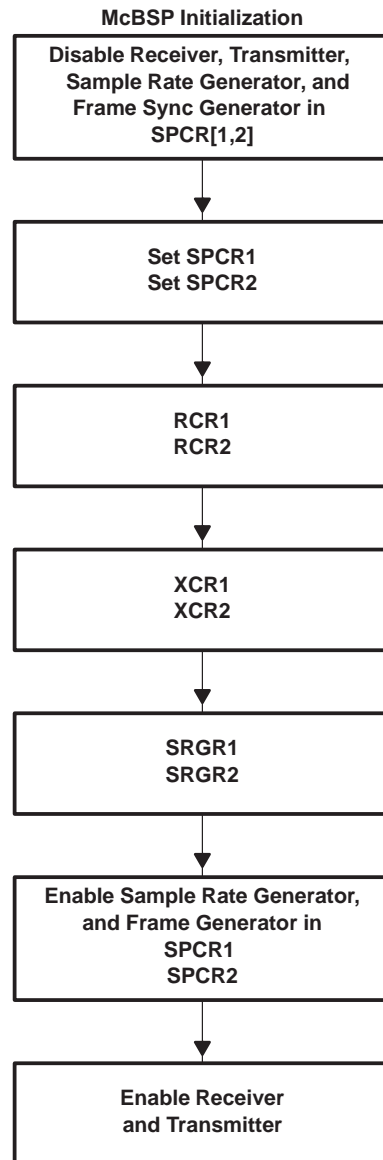
The following sections demonstrate the process used to configure the McBSP for this application.

Modifying a McBSP0 register requires writing its subaddress to the McBSP0 subaddress register (SPSA). The subaddress register for McBSP0 is located at address 0x0038 in program memory. The user must then write the control word for that register to address 0x0039 in program memory.

```
MMR(#0038h) = #0000h           ;Load subaddress of SPCR1 in subaddress register
                                ;in McBSP0
MMR(#0039h)=#0801h           ;Load value in SPCR1
```

With the receiver and transmitter in reset, the McBSP registers can be filled with the desired values.

The next sections demonstrate the procedure used in this report to program the McBSP0. Figure 2, McBSP Programming, is a flowchart of the procedure for setting up the McBSP. The following sections describe this procedure.



**Figure 2. McBSP Programming**

**Set XRST\* = RRS\* = FRST \* = GRST\* = 0 in SPCR[1,2].** This step disables the transmitter, the receiver, the frame-sync generator, and the sample rate generator. The free-running mode is enabled in SPCR2; in this mode, the serial port clock continues to run even after a software breakpoint occurs.

Program only the McBSP configuration registers as required while the serial port is in the reset state.

The following are the pin control register (PCR) settings required for this report:

- DX, FSX, CLKX are serial-port pins.
- DX is the data-transmit pin.
- FSX is the frame-sync pin, and is driven by the sample rate generator.

- FSR is an input pin driven by an external source.
- CLKX is an output pin driven by the internal sample rate generator.
- CLKR is an input pin driven by an external clock.
- FSX and FSR are active high (FSR is sent back on the EVM).
- The transmit data is sampled on the rising edge of CLKX.
- The receive data is sampled on the falling edge of CLKR.

PCR is set equal to #0A00h.

**Receive Control Register 1 (RCR1).** The following are the receive control register 1 (RCR1) settings required for this report:

- Receive one 16-bit word per frame.
- Single phase, meaning only one data word per frame-sync (FSR).
- Receive one 16-bit word per frame. The ADC sends out 16 bits per frame-sync. Since this is a 12-bit ADC, the last four LSBs are zero-padded.
- Data transfer to start with the MSB. The ADC sends out the MSB bit first.
- 1-bit data receive delay. Data sent out by ADC immediately after FSR falling edge.

RCR1 is set equal to #0040h.

**Receive Control Register 2 (RCR2).** This register can be ignored, since this report only deals with transfers that are less than 16 bits wide. However, this register should be configured the same as RCR1 for consistency.

RCR2 is set equal to #0041h.

**Transmit Control Register 1 (XCR1).** The following are the transmitter settings required in this report:

- Transmit one 16-bit word per frame. The TLV2544/TLV2548 accepts a 16-bit word per frame-sync transferred to it.
- Single phase, meaning only one data word per FSX
- Transmit one 16-bit word per frame
- Data transfer to start with MSB first
- 1-bit data transmit delay, meaning data sent out by ADC immediately after FSX/R

XCR1 is set equal to #0040h.

**Transmit Control Register 2. (XCR2).** This register can be ignored, since this report only deals with transfers that are less than 16 bits wide. However, this register should be configured the same as XCR1 for consistency.

XCR2 is set equal to #0041h.

**Sample Rate Generator Register 1. (SRGR1).** The following settings are required:

- Frame-sync pulse width (FWID) = 2 CLK periods. The smallest frame-sync width is two clock periods.
- $SCLK = (CPU\ clock / (1 + CLKGDV))$
- Desired SCLK = 20 Mhz. Since the CPU clock is equal to 100 Mhz, divide the number by four ( $CLKGDV = 4$ ) to generate the required sample rate generator (SRG) clock frequency.

SRGR1 is set equal to #0104h.

**Sample Rate Generator Register 2 (SRGR2).** The register should be configured as follows:

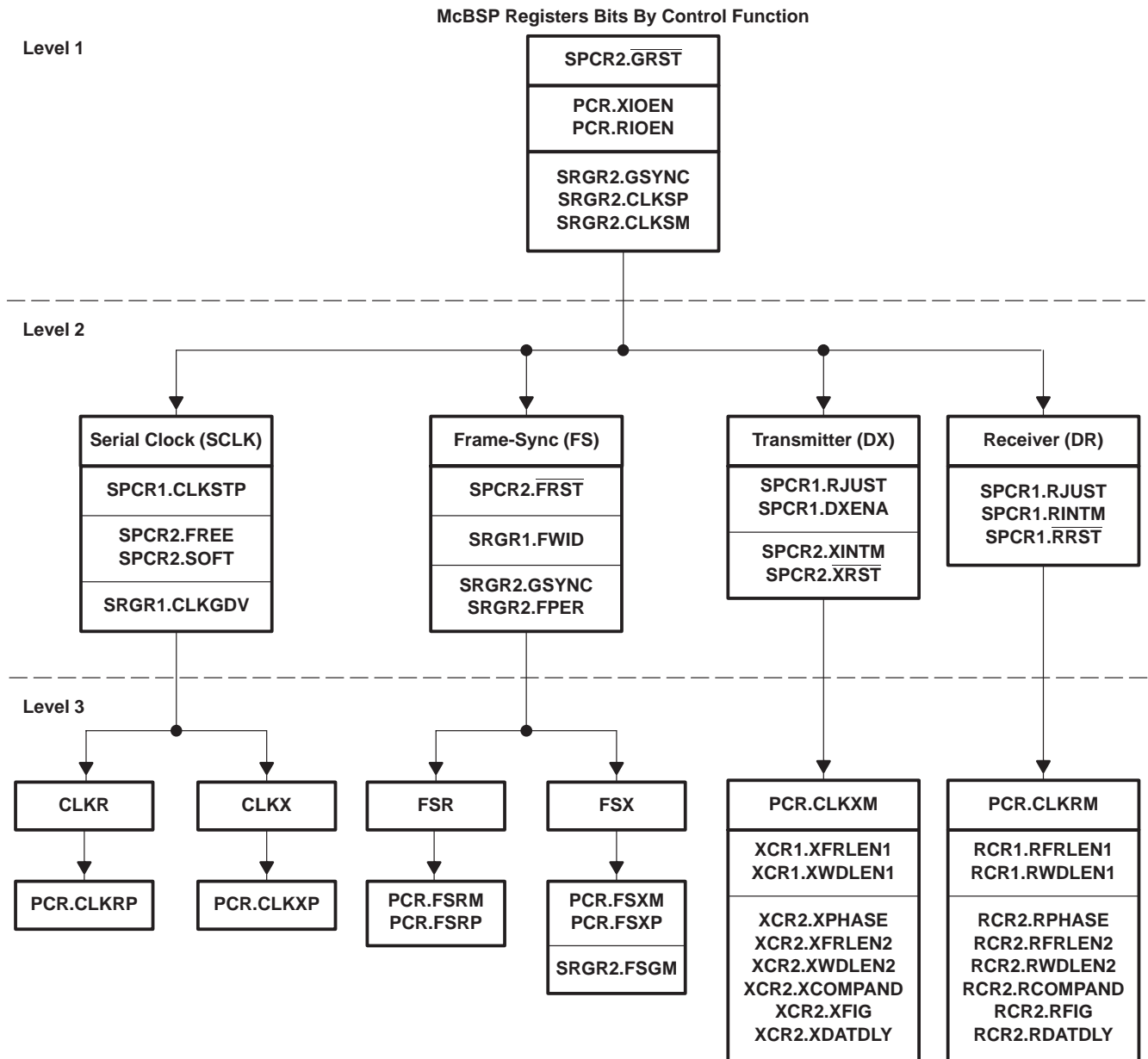
- $CLKSM = 1$ , sample rate generator (SRG0) clock, derived from the CPU clock.
- Free running SRG clock
- Transmit frame-sync (FSX) generated by copying DXR[1,2] to XSR[1,2]

SRGR2 is set equal to #2000h.

**Set XRST\* = RRST\* = FRST\* = GRST\* = 1 in SPCR[1,2].** This step enables the transmitter, the receiver, the frame-sync generator, and the sample rate generator. Wait two bit clocks for receiver and transmitter to become active.

The serial port interface consists of transmit and receive clocks, transmit and receive frame-sync, and transmit and receive data lines. Figure 3 takes the McBSP registers and categorizes its bits by related functions. Level one is the general McBSP setup, such as setting pins DX/R, FSX/R, and CLKX/R on the McBSP as serial port pins. Level 2 deals with the bits that affect a particular serial line. Level 3 bits deals with the specifics of the serial line, that is, the polarity of frame-sync, clocks, and the length of the transmitter and receiver words.

For more information on McBSP see TMS320C54X DSP Enhanced Peripherals Reference Set, Volume 5.



**Figure 3. McBSP Control Register Bit Features**

### 3.4 TLV2544/TLV2548 Configuration

The TLV2544/TLV2548 uses the first 4 bits as command bits (see Table 4). The device determines what mode to enter based on the first four bits received.

**Table 4. TLV2544/TLV2548 Command Set**

SDI D(15–12) BINARY		TLV2548 COMMAND	TLV2544 COMMAND
0000b	0h	Select analog input channel 0	Select analog input channel 0
0001b	1h	Select analog input channel 1	N/A
0010b	2h	Select analog input channel 2	Select analog input channel 1
0011b	3h	Select analog input channel 3	N/A
0100b	4h	Select analog input channel 4	Select analog input channel 2
0101b	5h	Select analog input channel 5	N/A
0110b	6h	Select analog input channel 6	Select analog input channel 3
0111b	7h	Select analog input channel 7	N/A
1000b	8h	SW power down (analog + reference)	
1001b	9h	Read CFR register data shown as SDO D(11–0)	
1010b	Ah plus data	Write CFR followed by 12-bit data, e.g., 0A100h means external reference, short sampling, SCLK/4, single shot, INT	
1011b	Bh	Select test, voltage = (REFP+REFM)/2	
1100b	Ch	Select test, voltage = REFM	
1101b	Dh	Select test, voltage = REFP	
1110b	Eh	FIFO read, FIFO contents shown as SDO D(15–4), D(3–0) = 0000	
1111b	Fh plus data	Reserved	

If the first four bits received by the ADC are 0xA, then the following 12 bits are interpreted by the ADC as configuration data. Likewise, if the first four bits received are 0xE, the ADC proceeds to output the contents the first level of the FIFO. See Table 5 for configuration register map and modes available.

**Table 5. TLV2544/TLV2548 Configuration Register Bit Definitions**

BIT	DEFINITION	
D11	Reference select 0: External      1: internal	
D10	Internal reference voltage select 0: Internal ref = 4 V   1: internal ref = 2 V	
D9	Sample period select 0: Short sampling 12 SCLKs (1x sampling time) 1: Long sampling 24 SCLKs (2x sampling time)	
D(8–7)	Conversion clock source select 00: Conversion clock = internal OSC 01: Conversion clock = SCLK 10: Conversion clock = SCLK/4 11: Conversion clock = SCLK/2	
D(6,5)	Conversion mode select 00: Single shot mode [FIFO not used, D(1,0) has no effect.] 01: Repeat mode 10: Sweep mode 11: Repeat sweep mode	
D(4,3) <sup>†</sup>	<b>TLV2548</b>	<b>TLV2544</b>
	Sweep auto sequence select 00: 0–1–2–3–4–5–6–7 01: 0–2–4–6–0–2–4–6 10: 0–0–2–2–4–4–6–6 11: 0–2–0–2–0–2–0–2	Sweep auto sequence select 00: N/A 01: 0–1–2–3–0–1–2–3 10: 0–0–1–1–2–2–3–3 11: 0–1–0–1–0–1–0–1
D2	EOC/ $\overline{\text{INT}}$ – pin function select 0: Pin used as $\overline{\text{INT}}$ 1: Pin used as EOC	
D(1,0)	FIFO trigger level (sweep sequence length) 00: Full ( $\overline{\text{INT}}$ generated after FIFO level 7 filled) 01: 3/4 ( $\overline{\text{INT}}$ generated after FIFO level 5 filled) 10: 1/2 ( $\overline{\text{INT}}$ generated after FIFO level 3 filled) 11: 1/4 ( $\overline{\text{INT}}$ generated after FIFO level 1 filled)	

<sup>†</sup> These bits only take effect in conversion modes 10 and 11.

### 3.5 TLV2544/TLV2548 Conversion Modes

The TLV2544/TLV2548 supports different conversion modes. Single-shot mode is the traditional method, that is, convert/sample, and immediately read the data out of the data converter. This method prevents the DSP from being used by other devices. The FIFO conversion modes (repeat, sweep, and repeat-sweep) allow the DSP to service the ADC less frequently. The repeat and sweep conversion modes are analyzed more closely in this report. Sample code for all the conversion modes is included in the appendixes.

Every program, regardless of desired conversion mode, must begin by properly initializing the DSP, the CPLD, and the McBSP. Two things need to be done to ensure that the DSP branches to the interrupt service routine: first, in the DSP, interrupt 3 must be enabled in the interrupt mask register (IMR), and the global interrupt bit (INTM) must be set; second, the CPLD must be configured so that signal  $\overline{\text{INT3}}$  from the daughtercard is transferred to the DSP. Since the CPLD controls the input of McBSPs, it must be set to allow the daughtercard to be the input to the McBSP0. Once these two steps have been completed, the next step is to program the McBSP0.

McBSP initialization procedure is shown in Figure 2. The procedure and settings are explained in Section 3.3. Once the McBSP is programmed, it is then time to configure the ADC.

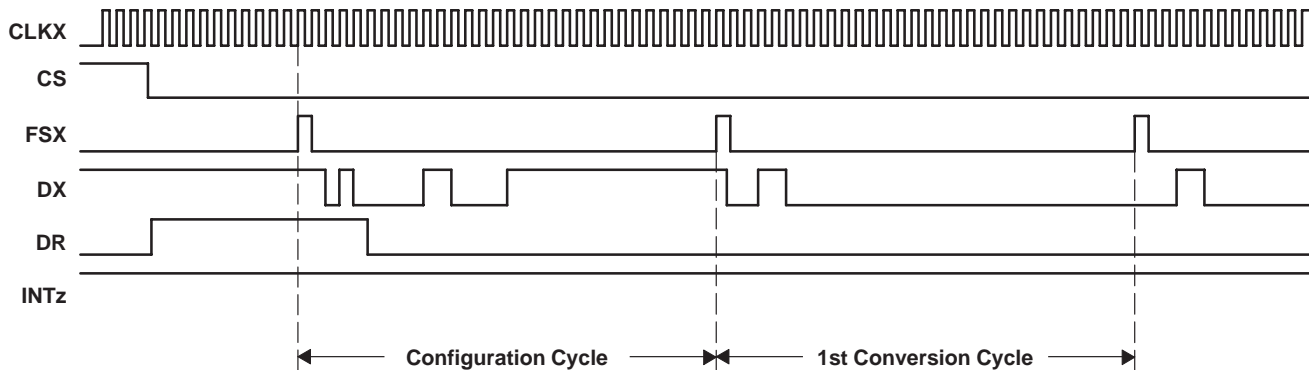
### 3.5.1 Sweep Mode

The TLV254x ADC is configured by writing the configuration word to the McBSP transmit register. When the ADC decodes the first four MSBs in the serial stream as 0xA, it expects the following 12 bits as configuration bits. With the TLV2544/TLV2548 properly configured, the user only needs to provide a frame-sync pulse to begin the conversion. The conversion results are then automatically stored in the FIFO. When the FIFO levels are filled, the ADC generates an interrupt pulse and wait for the DSP to service it.

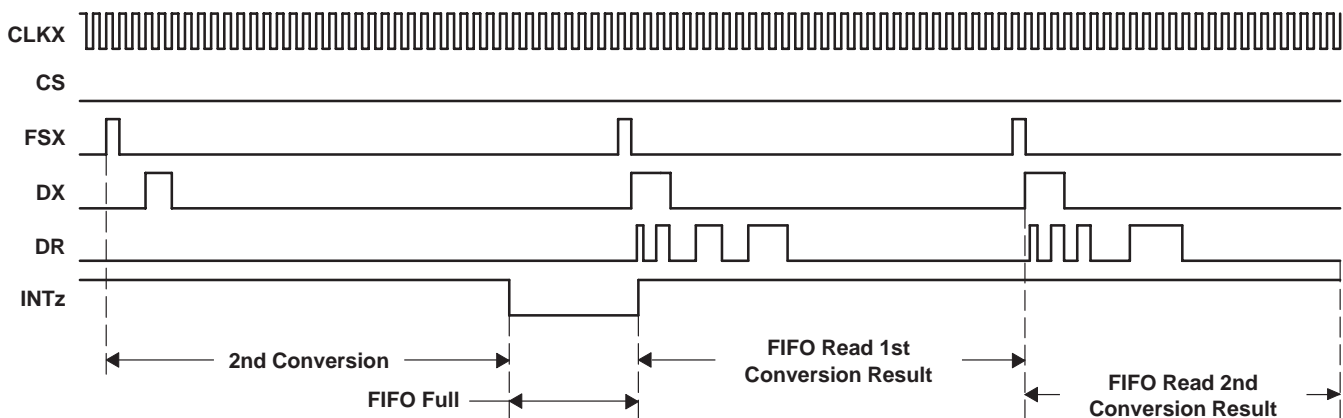
As an example, let us consider the TLV2548 configured for sweep mode and FIFO level equal to two (see Figure 4). Providing a frame-sync pulse to the data converter starts the conversion cycle (dummy writes are used to generate this frame-sync). In this example, two conversion cycles will fill-up the FIFO, causing it to send an interrupt to the DSP (see Figure 5). The user now has the options of reading the data out of the FIFO or starting another conversion. If another conversion cycle is begun before reading the FIFO out, the current samples will be lost.

When sweep sequence 00 is selected, a sample from channel 0 and a sample from channel 1 are stored in the FIFO. When the user issues a FIFO read command to the ADC, the sample from channel 0 is put on the DR line. A second FIFO read command to the TLV2548 puts the channel 1 sample on the DR line to be read out.

It is recommended that the dummy writes previously mentioned be channel select commands. This ensures that the ADC is not inadvertently reconfigured.



**Figure 4. TLV254x Configuration and Conversion Cycle**



**Figure 5. TLV254x Conversion and FIFO Read Cycles**

The flowchart for the sweep mode is presented in Figure 6.

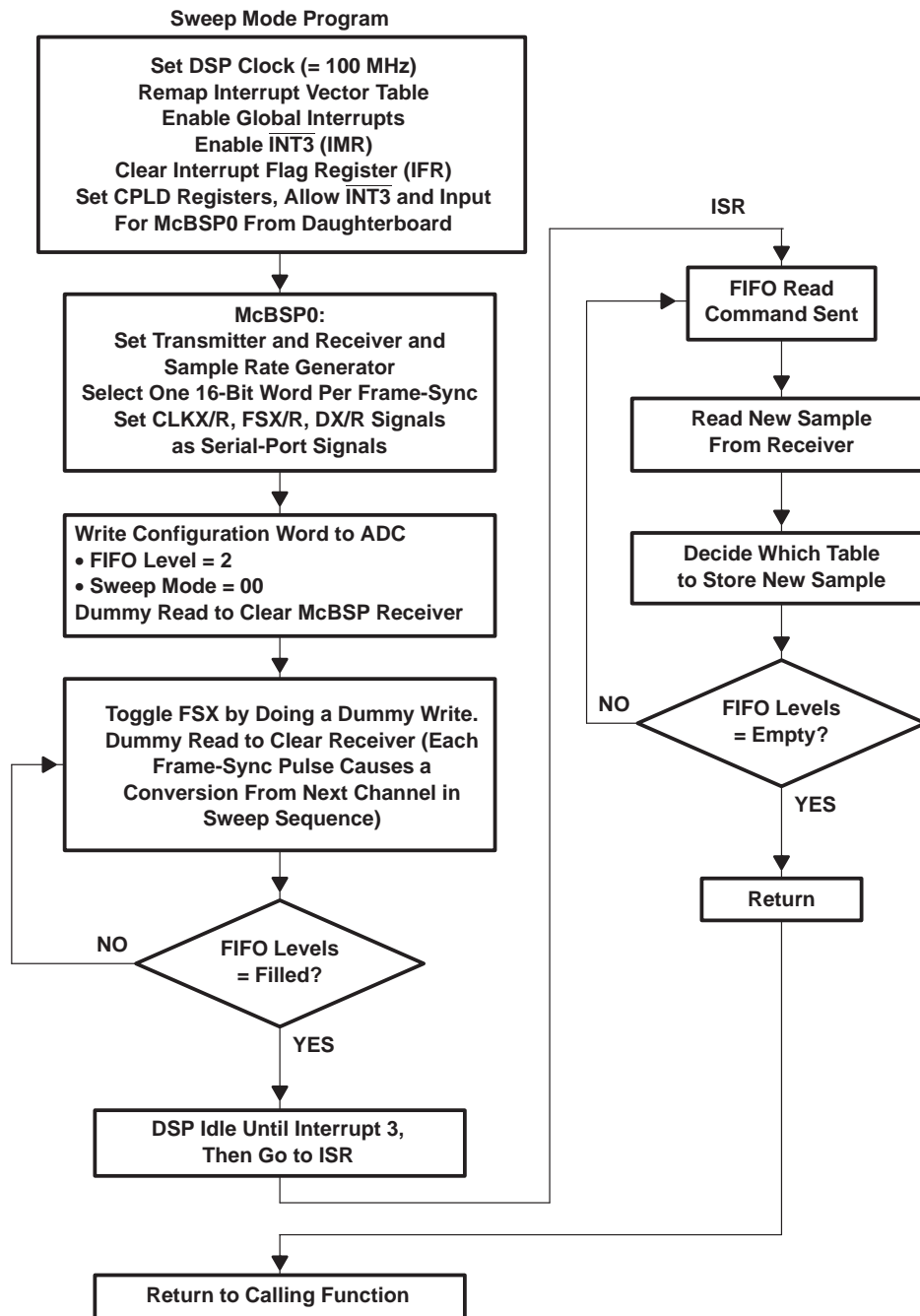


Figure 6. Sweep Mode Flowchart

### 3.5.2 *Repeat Mode*

The flowchart for repeat mode is presented in Figure 7. Repeat mode is similar to the sweep mode in that the interrupt is only triggered after the FIFO is filled. For example, if the FIFO level is set to 8 during configuration, the ADC converts eight samples and store them in the FIFO. When all eight levels of the FIFO are filled, an interrupt signal is sent to the DSP. The repeat mode differs from the sweep mode in that it allows the user to arbitrarily choose a combination of up to eight channels to sample from the TLV2548, or up to four channels from the TLV2544.

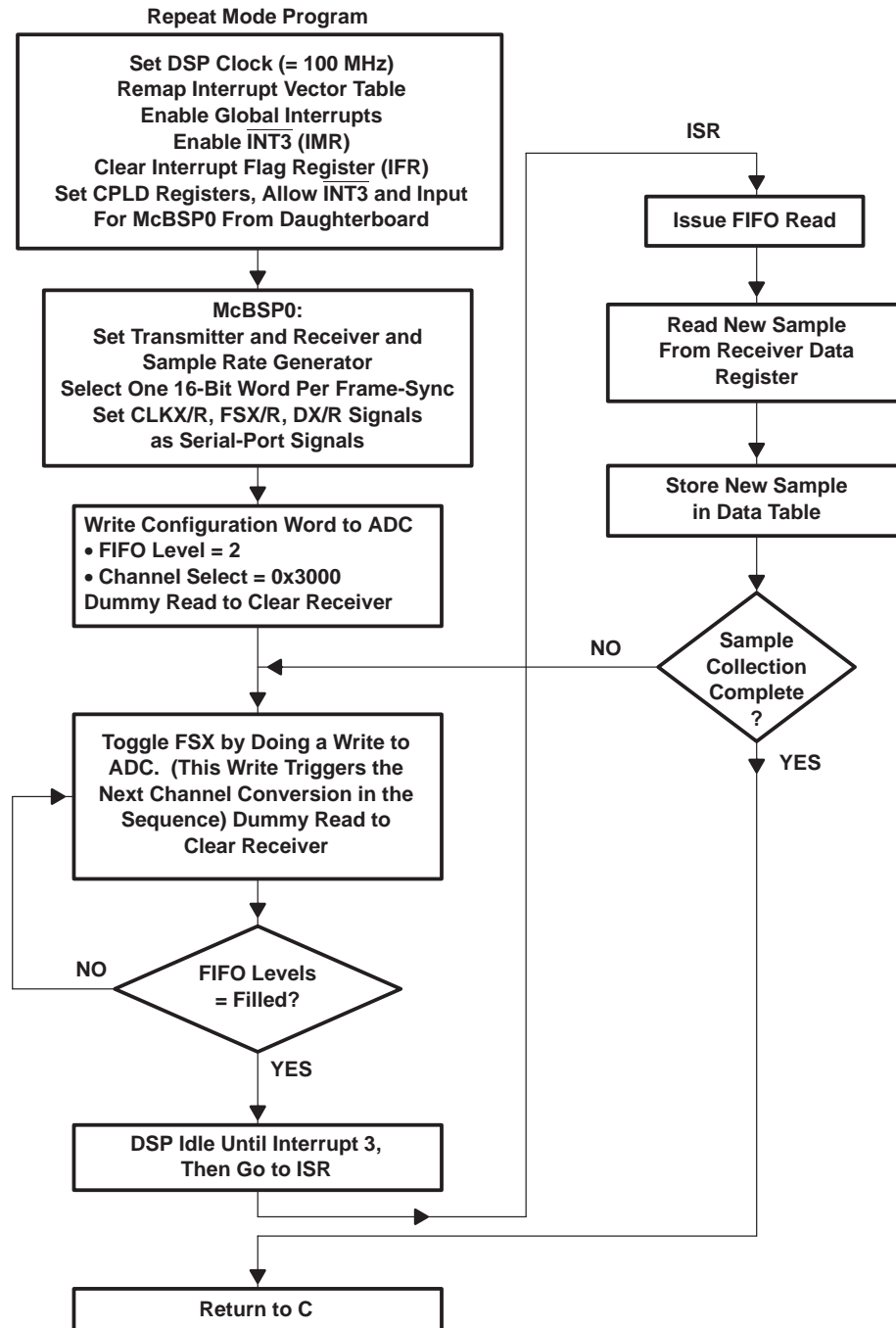
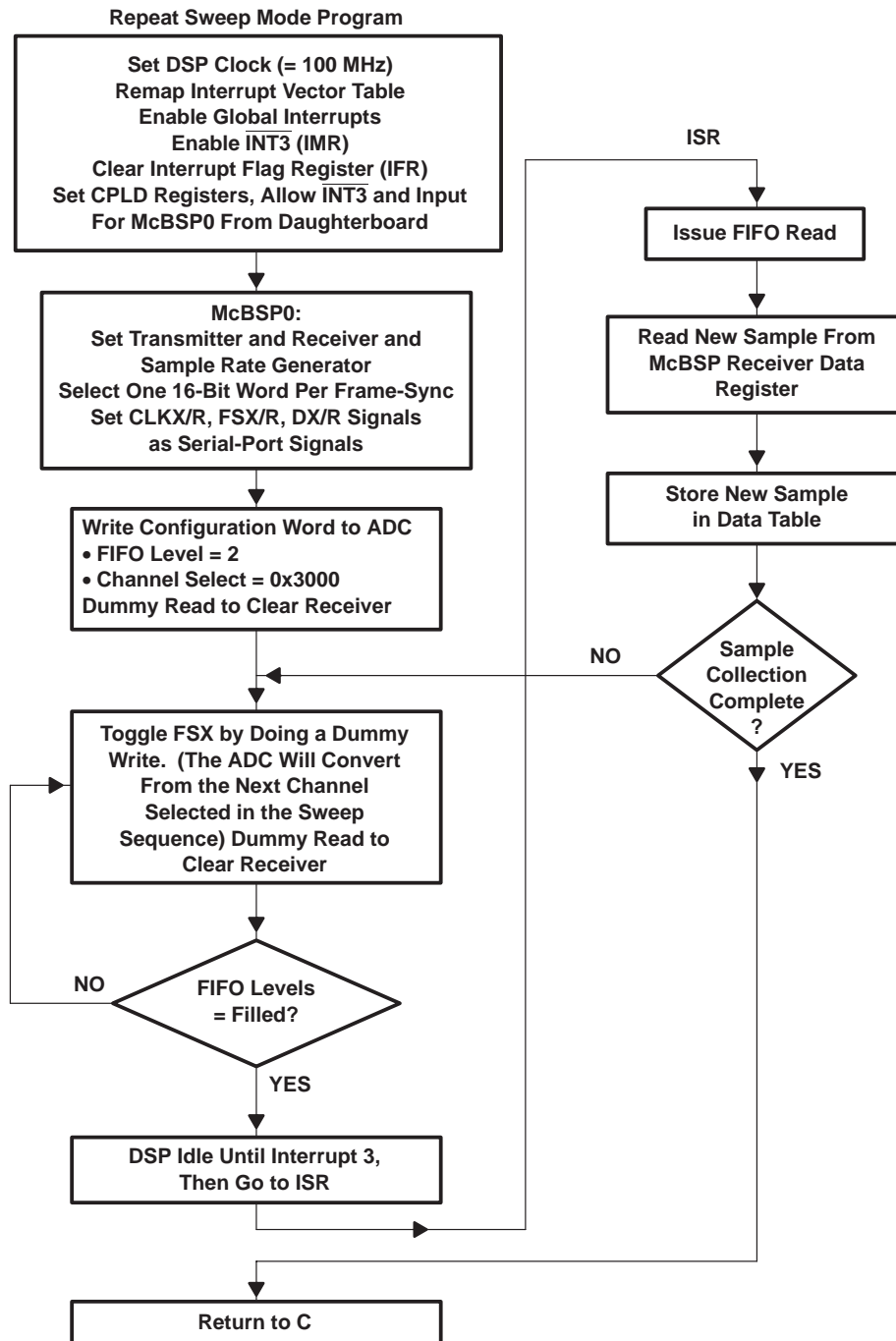


Figure 7. Repeat Mode Flow Chart

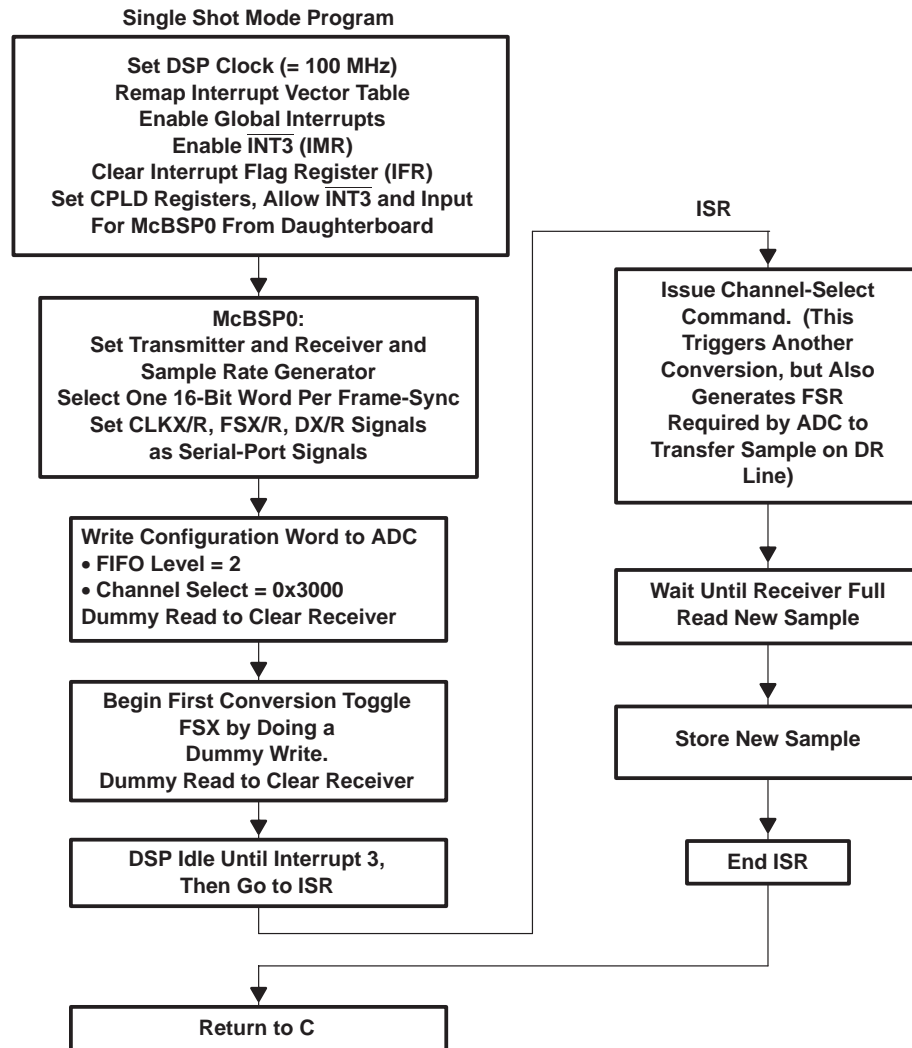
### 3.6 Repeat Sweep Mode and Single Shot Mode

Repeat sweep mode is almost identical to sweep mode. In repeat sweep mode, the ADC continuously samples and converts the particular sweep sequence selected during configuration. The flow chart is presented in Figure 8.



**Figure 8. Repeat Sweep Mode Flow Chart**

Single shot mode is the traditional method used for data conversion. The DSP must read the data out of the data converter before beginning another conversion cycle. This flow chart is presented in Figure 9.



**Figure 9. Single Shot Mode Flow Chart**

## 4 References

1. *TMS320C54x DSP CPU and Peripherals Reference Set Volume I*, literature number SPRU131F
2. *TMS320C54x DSP Set Reference Set Volume 3: Algebraic Instruction*, literature number SPRU179B
3. *TMS320C54x Assembly Language Tools User's Guide*, literature number SPRU102C
4. *TMS320C54xx DSP Enhanced Peripherals Reference Set, Volume 5*, literature number SPRU302
5. TLV2548 and TLV2544 data sheet, literature number SLAS198A
6. *TLVX544/2548EVM Evaluation Module for the TLVX544/TLV2548 10-Bit and 12-Bit ADC*, literature number SLAU029
7. *TMS320C54x DSP CPU and Peripherals Reference Set Volume I*, literature number SPRU131F

## Appendix A TLV2548 ADC C Program Main Routine

```
*****
* FILE           : TLV2548C.C                               *
* DESCRIPTION    *
* This program is the C file used to call assembly language files. The user *
* stores the Configuration Word in Config_Command, the channel command in *
* Conv_Command, the memory location to store sample in StoreAt, and the *
* number of samples to store in NumToStore. Once these variables are loaded, *
* the user can call the assembly function written for that particular *
* conversion mode to the calling function. *
*
* AUTHOR        : AAP Application Group, L. Philipose, Dallas, Tx *
*               : CREATED 2000(C) BY TEXAS INSTRUMENTS INCORPORATED. *
* REFERENCE     : TMS320C54x User's Guide, TI 1997 *
*               : :TMS320C54x Optimizing C Compiler User's Guide, TI 1998 *
*****
```

```
extern unsigned int StoreAt,CLKGDV, NumToStore, Config_Command;
extern unsigned int Conv_Command;
```

```
extern void SingeShotMode();
extern void RepeatSweepMode();
extern void SingleShotMode();
extern void SweepMode();
```

```
main(void)
```

```
{
    StoreAt=0x3200;
    NumToStore=0x400;
    // Config_Command=0xA000; /*Single shot Mode */
    // Config_Command=0xA0A3; /*Repeat Mode */
    Config_Command=0xA0C3; /*Sweep Mode */
    // Config_Command=0xA0E3; /*Repeat Sweep Mode */
    Conv_Command=0x3000;
    CLKGDV=0x0009;

    SweepMode();
}
```

## Appendix B Single Shot Conversion Mode

```
*****
* TITLE           : TLV2544/48 Interface routine                               *
* FILE            : tlv2548ss.ASM                                             *
* FUNCTION        : MAIN                                                       *
* PROTOTYPE       : void SingleShotMode()                                     *
* CALLS           : N/A                                                        *
* PRECONDITION    : N/A                                                        *
* POSTCONDITION   : N/A                                                        *
* DESCRIPTION     :                                                              *
* This program is to be used for single shot conversion mode. The program    *
* expects the calling program to store the configuration word in              *
* Config_Command, the channel Conversion Command in _Conv_Command, and the    *
* memory location to store samples in _StoreAt. The code starts off by       *
* programming the DSP, CPLD, McBSP0, and then the TLV2548 analog-to-digital   *
* converter. Since a McBSP transmit operation generates the FSR used to read  *
* data into the receiver. The function ADC_Write_Read is used to both read    *
* and write to the device. The McBSP0_init function configures the McBSP     *
* serial port. This program is designed to gather one sample and then return  *
* to the calling function.                                                    *
* AUTHOR          : AAP Application Group, L. Philipose, Dallas, Tx           *
*                 : CREATED 2000(C) BY TEXAS INSTRUMENTS INCORPORATED.      *
* REFERENCE       : TMS320C54x User's Guide, TI 1997                          *
*                 : Data Acquisition Circuits, TI 1999                          *
*****
```

```

        .title    "TLV2544/48 ADC"
        .mmregs
        .width    80
        .length   55

        .sect    ".vectors"
        .copy    "C5402vec.asm"
        .sect    ".data"
        .include regs.h
*Global Variables*
        .global  _CLKGDV
        .global  _SingleShotMode
        .global  _StoreAt
        .global  _Config_Command
        .global  _Conv_Command
AD_DP          .usect ".varibl", 0      ;Data page label for variables.
ADWORD        .usect ".varibl", 1      ;
Temp          .usect ".varibl", 1      ;control word for the ADC
EndOfTable    .usect ".varibl", 1      ;End of Table
_Config_Command .usect ".varibl", 1    ;Configuration Word
_Conv_Command  .usect ".varibl", 1    ;Command word to ADC
_CLKGDV       .usect ".varibl", 1    ;Variable used to set CLKX/R on McBSP0
_StoreAt       .usect ".varibl", 1    ;Address to begin Storing Samples
        .sect    ".text"
_SingleShotMode:

```

```

_main:
start:

***Initialize DSP***
DP=#0;                ;Memory Mapped Registers in Data Page 0
CLKMD = #4007h        ;Set c5402 DSP clock to 100MHz.

INTM=#1               ;Disable maskable interrupts.
SXM=#0                ;No Sign Extension
CPL=0                 ;DP used for relative addressing
PMST = #00F8h;        ;Re-map Interrupt Vector Table to 0x0080h in Program Memory
IMR = #0104h;         ;Enabled Interrupt #3
IFR = #0FFFFh;        ;Clear all Pending Interrupts

SWWSR = #2000h;       ;Need Wait-State of at least 2.

DP=#ADWORD            ;Change Data Page to where Variables are stored.

@Temp=#0088h          ;Enable INT3# From Daughtercard.
port(DSP_CPLD_CNTL1) =@Temp ;

@Temp=#0001h
port(DSP_CPLD_CNTL2) =@Temp ;Select daughterboard as source for McBSP0
XF = 1                ;Set ADC Chip Select High

call McBSP0_init      ;Set McBSP0 registers
INTM=#0               ;enable global interrupts
DP=#AD_DP             ;
XF = 0                ;SET CSz. ADC Chip Select is held low throughout
                    ;this program.

A=@_StoreAt
AR7=A                 ;Address to store Samples at.

*Configure the ADC *  ;Transmit Word Store in Accum. A. Receive Word Stored in
                    ; Accum. B
A=#0A080h            ;Configuration Word, Choose external SCLK, and Single shot
                    ; mode.
call ADC_Write_Read  ;Write and Read from McBSP0. Write the Configuration Data
                    ; to ADC, and read out receiver data register. Receiver
                    ; data is stored
                    ;in Accumulator B. In this case we don't care what was
                    ; received.

*Select channel and begin conversion cycle*
A=#3000h              ;Select Channel 3 to sample and convert
call ADC_Write_Read  ;Write it to McBSP0. Data received is invalid.

                    ;DSP sits in idle mode until ADC generates an Interrupt.
idle (1)              ;Once INT3# occurs, the DSP will power-up and launch
                    ; ISR_int03.

```

```

        nop                ;When complete, the DSP will sit idle once again waiting
        nop                ; for next. interrupt.
        RETURN            ;Return to C calling function

**Interrupt #3 Service Routine **
ISR_int03:                ;This ISR is loaded into Program Counter after
                        ;Interrupt 3 occurs.

        DP = #AD_DP      ;Set Data Page
        A=#3000h         ;Generate Frame-Sync by providing a dummy
                        ; write. This dummy write selects
        call ADC_Write_Read ;Channel #3, as the next channel to sample and
                        ; convert from.

        @ADWORD=B       ;Store converted data in this variable temporarily
        *AR7+ = data(@ADWORD) ;Store Sample at Memory Location pointed to by
                        ; AR7, then increment AR7.

Quit:
        return_enable

**Function Configures McBSP0**

McBSP0_init:

        mmr(McBSP0_SPSA) = #SPCR1      ;Reset McBSP0 Receiver
        A=mmr(McBSP0_SPSD)            ;
        A = #0FFFEh & A                ;RRST*=0
        mmr(McBSP0_SPSD) =A

        repeat(#5)
            NOP

        mmr(McBSP0_SPSA) = #SPCR2      ;Reset McBSP0 Receiver
        A=mmr(McBSP0_SPSD)            ;
        A= A & #0FF7Fh                  ;FRST*=0
        A= A & #0FFBFh                  ;GRST*=0
        A= A & #0FFFEh                  ;XRST*=0

        A=A|#0200h                      ;enable free running mode
        mmr(McBSP0_SPSD) = A            ;

        repeat(#5)
            NOP

        mmr(McBSP0_SPSA) = #PCR        ;FSXM & CLKXM output pin driven
        mmr(McBSP0_SPSD) = #0A00h     ;by sample-rate generator
                                        ;FSRM & CLKRM input pins
                                        ;driven by external source

        repeat(#6)
            NOP
    
```

```

mmr(McBSP0_SPSA) = #RCR1      ;
mmr(McBSP0_SPSD) = #0040h    ;One 16-bit Word per frame

repeat(#6)
  NOP

mmr(McBSP0_SPSA) = #RCR2      ;One 16-bit Word per frame
mmr(McBSP0_SPSD) = #0041h    ;one bit receive bit data delay

repeat(#6)
  NOP

mmr(McBSP0_SPSA) = #XCR1      ;
mmr(McBSP0_SPSD) = #0040h    ;One 16-bit Word per frame
repeat(#6)
  NOP
mmr(McBSP0_SPSA) = #XCR2      ;One 16-bit Word per frame
mmr(McBSP0_SPSD) = #0041h    ;one bit transmit data delay

repeat(#6)
  NOP
mmr(McBSP0_SPSA) = #SRGR1     ;FS width=one clock
A=#0;
A= #0100h | A                 ;
B=@_CLKGDV
A=A|B
mmr(McBSP0_SPSD) = A         ;SRG clock divider=9 (CPU Clock /1+CLKGDV)=10 MHz

repeat(#6)
  NOP

mmr(McBSP0_SPSA) = #SRGR2     ;SRG clock derived from CPU clock
mmr(McBSP0_SPSD) = #2000h    ;FSX due to transfer DXR->XSR

repeat(#6)
  NOP

mmr(McBSP0_SPSA) = #SPCR2     ;
A=mmr(McBSP0_SPSD)
A= A | #0040h                 ;GRST*=1
mmr(McBSP0_SPSD) = A         ;Frame Generator and sample
                               ;sample-rate generator enabled

repeat(#6)
  NOP

A= A | #0001h                 ;XRST*=1
mmr(McBSP0_SPSD) = A         ;Transmitter enabled

repeat(#6)
  NOP

```

```

    mmr(McBSP0_SPSA) = #SPCR1      ;RRST*=1
A= mmr(McBSP0_SPSD)
A= A | #0001h
mmr(McBSP0_SPSD) = A              ;Receiver enabled

repeat(#6)
    NOP

RETURN
**Function Write to Transmitter and Read from Receiver of McBSP0**
**Data to be written out must be stored in Accum. A,          **
**Data received in stored in Accumulator B                   **
ADC_Write_Read:
    PUSH (AR2)          ;input in A
    PUSH (AR1)          ;Save Registers
    PUSH (AR0)
Wait_On_Transmitter:
    DP=#0
    mmr(McBSP0_SPSA) = #SPCR2      ;Check to see if transmitter is ready to send
    B=mmr(McBSP0_SPSD)             ;new data.

    B=B & #0002h;
    AR1=B
        nop
    nop
    AR0=#2
    nop
    nop
    TC      = (AR0 == AR1)
    if (NTC) goto Wait_On_Transmitter ;If transmitter is full wait until it is not.
    AR2=#McBSP0_DXR1
    nop
    *AR2 = A ;Send out to ADC

Wait_On_Receiver:
    DP=#0
    mmr(McBSP0_SPSA) = #SPCR1      ;
    B=mmr(McBSP0_SPSD)             ;Check to see if Receiver is FULL with Data
    AR0=#2
    B=B & #0002h;
    AR1=B
        nop
    nop
    TC = (AR0 == AR1)
    if (NTC) goto Wait_On_Receiver ;If receiver is not ready with new data, then
wait.
    DP=#AD_DP
    AR2=#McBSP0_DRR1
    NOP
    B=*AR2                          ;Read out ADC
    AR0 =POP()

```

SLAA093A

---

```
AR1 =POP()  
AR2 =POP()  
RETURN  
.end
```

## Appendix C Repeat Mode Conversion Mode

```

*****
* TITLE           : TLV2540/48 Interface routine           *
* FILE            : tlv2548rep_m.ASM                       *
* FUNCTION        : MAIN                                    *
* PROTOTYPE       : void SweepMode ( )                    *
* CALLS           : N/A                                    *
* PRECONDITION    : N/A                                    *
* POSTCONDITION   : N/A                                    *
* DESCRIPTION     :                                         *
* This routine is written to demonstrate sweep mode of the TLV2548 ADC. *
* The program expects the calling program to store the configuration word *
* Config_Command, the channel Conversion Command in _Conv_Command, the *
* memory location to store sample in _StoreAt, and the number of samples *
* to gather in _NumtoStore. The code starts off by setting the DSP, CPLD, *
* and McBSP0 registers for this interface. The ADC is then configured, and *
* conversions triggered. Once the FIFO is filled, the DSP sits idle until *
* an interrupt is received from the ADC. When an interrupt is received, *
* the DSP jumps to Interrupt Service Routine (ISR). The DSP reads the data *
* out of the FIFO, checks to see if it has collected enough samples, then *
* returns to the main program.                               *
*                                                         *
* AUTHOR          : AAP Application Group, L. Philipose, Dallas, Tx *
*                 : CREATED 2000(C) BY TEXAS INSTRUMENTS INCORPORATED. *
* REFERENCE       : TMS320C54x User's Guide, TI 1997      *
*                 : Data Acquisition Circuits, TI 1999    *
*****
    
```

```

        .title    "TLV2544/48 ADC"
        .mmregs
        .width    80
        .length   55

        .sect ".vectors"
        .copy "C5402vec.asm"

        .sect ".data"
        .include regs.h
**Global Variables**
        .global _RepeatMode
        .global _StoreAt
        .global _NumToStore
        .global _Config_Command
        .global _Conv_Command
        .global _CLKGDV
**Variables**
AD_DP           .usect ".varibl", 0 ;Data page label for variables.
_StoreAt        .usect ".varibl", 1 ;Address to begin Storing Samples
_NumToStore    .usect ".varibl", 1 ;Total Number of Samples to Collect
_Config_Command .usect ".varibl", 1 ;Configuration Word
_Conv_Command  .usect ".varibl", 1 ;Command word to ADC
_CLKGDV       .usect ".varibl", 1 ;Variable used to set CLKX/R on McBSP0
    
```

```

ADWORD          .usect ".varibl", 1 ;Converted Data stored t
Temp            .usect ".varibl", 1 ;
FIFO_LEVEL      .usect ".varibl", 1 ;FIFO set to Trigger Interrupt at this Level
EndOfTable      .usect ".varibl", 1 ;End of Table
Next_Table      .usect ".varibl", 1 ;Next Table to store Sample
Quit_Now        .usect ".varibl", 1 ;Next Table to store Sample
Input_CH_0      .set 00000h
Input_CH_1      .set 01000h
Input_CH_2      .set 02000h
Input_CH_3      .set 03000h
Input_CH_4      .set 04000h
Input_CH_5      .set 05000h
Input_CH_6      .set 06000h
Input_CH_7      .set 07000h
SW_PWR_DWN      .set 08000h
Read_CFR        .set 09000h
Write_CFR       .set 0A000h
Self_Test_Half  .set 0B000h
Self_Test_MINUS .set 0C000h
Self_Test_PLUS  .set 0D000h
FIFO_Read       .set 0E000h

.sect ".text"
_RepeatMode:
_MAIN:
start:

***Initialize DSP***
CPL=#0          ;Using DP for relative addressing mode
NOP
NOP
DP=#0;          ;Memory Mapped Registers in Data Page 0
CLKMD = #4007h ;Set c5402 DSP clock to 100MHz.

INTM=#1        ;Disable maskable interrupts.
SXM=#0         ;No Sign Extension
PMST = #00F8h; ;Re-map Interrupt Vector Table to 0x0080h in
               ; Program Memory
IMR = #0100h;  ;Enabled Interrupt #3
IFR = #0FFFFh; ;Clear all Pending Interrupts

SWWSR = #2000h; ;Need Wait-State of at least 2.

DP=#AD_DP      ;Change Data Page to where Variables are stored.

@Temp=#0088h   ;Enable INT3# From Daugthercard.
port(DSP_CPLD_CNTL1) =@Temp ;

@Temp=#0001h
port(DSP_CPLD_CNTL2) =@Temp ;Select daughterboard as source for McBSP0
call McBSP0_init ;Set McBSP0 registers

```

```

INTM=#0                ;enable global interrupts

DP=#AD_DP              ;
XF = 0                 ;SET CSz. ADC Chip Select is held low throughout this
                       ; program.

A=@_StoreAt
AR7=A                  ;Address to begin storing Samples at.
B=@_NumToStore         ;Add number of samples requires to table starting location.
A=B+A
@EndOfTable=A         ;End of Sample Table

call FIFO_LEVEL_SET   ;Determine FIFO Trigger Level

*Configure the ADC *   ;Transmit Word Store in Accum. A. Receive Word Stored in
                       ; Accum. B
A=@_Config_Command    ;Configuration Word, Choose external SCLK, and Single shot
                       ; mode.
call ADC_Write         ;Write and Read from McBSP0. Write the Configuration Data
                       ; to
call ADC_Read          ;ADC, and read out receiver data register. Receiver data is
                       ; stored in Accumulator B. In this case we don't care what
                       ; was received.

@Quit_Now=#0          ;0->Continue sampling
                       ;1->Return to C

Wait:
TC = (@Quit_Now == #1)
if (TC) goto Return_To_C

A=@FIFO_LEVEL
BRC = A
NOP
NOP
blockrepeat(continueCYCLE-1)

A=@_Conv_Command
call ADC_Write
call ADC_Read
repeat(#6)
nop
continueCYCLE:        ;end_block repeat

idle(#1)              ;DSP power-down until INT3# occurs
nop
nop
goto Wait

Return_To_C:

RETURN
McBSP0_init:
    
```

```

mmr(McBSP0_SPSA) = #SPCR1      ;Reset McBSP0 Receiver
A=mmr(McBSP0_SPSD)            ;
A = #0FFFEh & A                ;RRST*=0
mmr(McBSP0_SPSD) =A

mmr(McBSP0_SPSA) = #SPCR2      ;Reset McBSP0 Receiver
A=mmr(McBSP0_SPSD)
A= A & #0FF7Fh                 ;FRST*=0
A= A & #0FFBFh                 ;GRST*=0
A= A & #0FFFEh                 ;XRST*=0

A=A|#0200h                     ;enable free running mode
mmr(McBSP0_SPSD) = A          ;

mmr(McBSP0_SPSA) = #PCR        ;FSXM & CLKXM output pin driven
mmr(McBSP0_SPSD) = #0A00h      ;by sample-rate generator
                                ;FSRM & CLKRM input pins driven by external source

mmr(McBSP0_SPSA) = #RCR1       ;
mmr(McBSP0_SPSD) = #0040h      ;One 16-bit Word per frame

mmr(McBSP0_SPSA) = #RCR2       ;One 16-bit Word per frame
mmr(McBSP0_SPSD) = #0041h      ;one bit receive bit data delay

mmr(McBSP0_SPSA) = #XCR1       ;
mmr(McBSP0_SPSD) = #0040h      ;One 16-bit Word per frame
mmr(McBSP0_SPSA) = #XCR2       ;One 16-bit Word per frame
mmr(McBSP0_SPSD) = #0041h      ;one bit transmit data delay

mmr(McBSP0_SPSA) = #SRGR1      ;FS width=one clock
A= #0100h                      ;
B=@_CLKGDV                     ;CLKGDV=9->10 MHz
A=A|B                           ;CLKGDV=4->20 MHz
mmr(McBSP0_SPSD) = A           ;SRG clock divider=9 (CPU Clock /1+CLKGDC)=10 MHz
mmr(McBSP0_SPSA) = #SRGR2      ;SRG clock derived from CPU clock
mmr(McBSP0_SPSD) = #2018h      ;FSX due to transfer DXR->XSR

mmr(McBSP0_SPSA) = #SPCR2      ;
A=mmr(McBSP0_SPSD)
A= A | #00080h                 ;FRST*=1
A= A | #0040h                 ;GRST*=1
mmr(McBSP0_SPSD) = A          ;Frame Generator and sample
                                ;sample-rate generator enabled

A= A | #0001h                 ;XRST*=1
mmr(McBSP0_SPSD) = A          ;Transmitter enabled

repeat(#6)                     ;wait for transmitter to become active
NOP

mmr(McBSP0_SPSA) = #SPCR1      ; RRST*=1
A= mmr(McBSP0_SPSD)

```

```

A= A | #0001h
mmr(McBSP0_SPSD) = A           ;Receiver enabled

repeat(#6)                     ;wait for receiver to become active
    NOP

RETURN

**Function Write to Transmitter and Read from Receiver of McBSP0**
**Data to be written out must be stored in Accum. A,           **
**Data received in stored in Accumulator B                   **
ADC_Write:                     ;input in A
    PUSH (AR2)                ;Save Registers
    PUSH (AR1)
    PUSH (AR0)
Wait_On_Transmitter:
    DP=#0
    mmr(McBSP0_SPSA) = #SPCR2 ;Check to see if transmitter is ready to send
    B=mmr(McBSP0_SPSD)        ;new data.

    B=B & #0002h;
    AR1=B
    nop
    nop
    AR0=#2
    nop
    nop
    TC      = (AR0 == AR1)
    if (NTC) goto Wait_On_Transmitter ;If transmitter is full wait until it is not.
    AR2=#McBSP0_DXR1
    nop
    *AR2 = A ;Send out to ADC
    AR0 =POP()
    AR1 =POP()
    AR2 =POP()
    RETURN

ADC_Read:                       ;Output in B
    PUSH (AR2)                   ;Save Registers
    PUSH (AR1)
    PUSH (AR0)
Wait_On_Receiver:
    mmr(McBSP0_SPSA) = #SPCR1    ;
    B=mmr(McBSP0_SPSD)          ;Check to see if Receiver is FULL with Data
    AR0=#2
    B=B & #0002h;
    AR1=B
    nop
    nop
    TC = (AR0 == AR1)
    if (NTC) goto Wait_On_Receiver ;If receiver is not ready with new data, then
    ; wait.
    
```

```

DP=#AD_DP
AR2=#McBSP0_DRR1
NOP
B=*AR2                ;Read out ADC
AR0 =POP()
AR1 =POP()
AR2 =POP()
RETURN
**Function determines the Level FIFO is set to trigger interrupt.**
**FIFO Level is saved in variable FIFO_Level **
FIFO_LEVEL_SET:

A=@_Config_Command    ;Load ADC configuration Word and determine
A=A & #0003h          ;FIFO Trigger Level

@Temp=A
TC=(@Temp == #0)      ;If it set to FIFO level 7?
if (NTC) goto NOT_FIFO__LEVEL_7
A=#7
@FIFO_LEVEL=A

goto FIFO_SET
NOT_FIFO__LEVEL_7:    ;Not Level 7, if FIFO set to trigger when
TC=(@Temp == #1)      ;level 5 is filler?
if (NTC) goto NOT_FIFO__LEVEL_5
A=#5
@FIFO_LEVEL=A

goto FIFO_SET
NOT_FIFO__LEVEL_5:    ;Not set to trigger at level 7 or 5. Maybe
                    ;Level 3?
TC=(@Temp == #2)
if (NTC) goto NOT_FIFO__LEVEL_3
A=#3
@FIFO_LEVEL=A

goto FIFO_SET
NOT_FIFO__LEVEL_3:    ;Not Set to Trigger at Level 7,5,3. Then has to be 1.
A=#1                  ;FIFO has Two Levels 0,1. When these levels are filled
@FIFO_LEVEL=A        ;TLV254x will generate an Interrupt.
FIFO_SET:

RETURN

ISR_int03:
DP = #AD_DP
A=@FIFO_LEVEL        ;Load in Block Repeat Counter Register the number of
BRC = A              ;FIFO levels to Read.
NOP
NOP
blockrepeat(Readout_FIFO-1)

```

```

    repeat(#5)
    nop
    A=#FIFO_Read           ;Write Command FIFO Read.
    call ADC_Write        ;This prompts the ADC to shift
    call ADC_Read         ;FIFO contents out.
    A=@EndOfTable        ;Load into Accum. A, the end of Storage table.
    AR0=A
    @ADWORD=B            ;Store converted data in this variable
                        ;temporarily
    TC      = (AR0 == AR7) ;Total Samples Collect = Total Requested?
    if (TC) goto Quit

Table_0:
    *AR7+  = data(@ADWORD) ;point to first date location of the storage
                        ;table

Readout_FIFO:
    goto Continue
Quit:
    @Quit_Now=#1

Continue:

    return_enable
.end

```

## Appendix D Sweep Mode Conversion Mode

```

*****
* TITLE           : TLV2544/48 Interface routine          *
* FILE            : tlv2548sweep_m.ASM                   *
* FUNCTION        : MAIN                                  *
* PROTOTYPE       : void SweepMode ( )                   *
* CALLS           : N/A                                   *
* PRECONDITION    : N/A                                   *
* POSTCONDITION   : N/A                                   *
* DESCRIPTION     :                                       *
* This routine is written to demonstrate sweep mode of the TLV2548 ADC. *
* The program expects the calling program to store the configuration word *
* Config_Command, the channel Conversion Command in _Conv_Command, the *
* memory location to store sample in _StoreAt, and the number of samples *
* to gather in _NumtoStore. The code starts off by setting the DSP, CPLD, *
* and McBSP0 registers for this interface. The ADC is then configured, and *
* conversions triggered. Once the FIFO is filled, the DSP sits idle until *
* an interrupt is received from the ADC. When an interrupt is received, *
* the DSP jumps to Interrupt Service Routine (ISR). The DSP reads the data *
* out of the FIFO, stores the data in the respective data table, checks to *
* see if it has collected enough samples, then returns to the main *
* program. A FIFO level of 2 is assumed, so only two data tables are *
* initialized.                                           *
*                                                         *
* AUTHOR          : AAP Application Group, L. Philipose, Dallas, Tx *
*                 CREATED 2000(C) BY TEXAS INSTRUMENTS INCORPORATED. *
* REFERENCE       : TMS320C54x User's Guide, TI 1997 *
*                 : Data Acquisition Circuits, TI 1999 *
*****

        .title    "TLV2544/48 ADC"
        .mmregs
        .width    80
        .length   55

        .sect    ".vectors"
        .copy    "C5402vec.asm"

        .sect    ".data"
        .include regs.h
**Global Variables**
.global _SweepMode
.global _StoreAt
.global _NumToStore
.global _Config_Command
.global _Conv_Command
.global _CLKGDV
**Variables**
AD_DP           .usect ".varibl", 0 ;Data page label for variables.
_StoreAt         .usect ".varibl", 1 ;Address to begin Storing Samples
_NumToStore     .usect ".varibl", 1 ;Total Number of Samples to Collect
_Config_Command .usect ".varibl", 1 ;Configuration Word
_Conv_Command   .usect ".varibl", 1 ;Command word to ADC
_CLKGDV         .usect ".varibl", 1 ;Variable used to set CLKX/R on McBSP0

```

```

ADWORD          .usect ".varibl", 1 ;Converted Data stored t
Temp            .usect ".varibl", 1 ;
FIFO_LEVEL      .usect ".varibl", 1 ;FIFO set to Trigger Interrupt at this Level
EndOfTable      .usect ".varibl", 1 ;End of Table
Next_Table      .usect ".varibl", 1 ;Next Table to store Sample
Quit_Now        .usect ".varibl", 1 ;Next Table to store Sample
Input_CH_0      .set 00000h
Input_CH_1      .set 01000h
Input_CH_2      .set 02000h
Input_CH_3      .set 03000h
Input_CH_4      .set 04000h
Input_CH_5      .set 05000h
Input_CH_6      .set 06000h
Input_CH_7      .set 07000h
SW_PWR_DWN      .set 08000h
Read_CFR        .set 09000h
Write_CFR        .set 0A000h
Self_Test_Half  .set 0B000h
Self_Test_MINUS .set 0C000h
Self_Test_PLUS  .set 0D000h
FIFO_Read       .set 0E000h

    .sect ".text"
_SweepMode:
_MAIN:
start:

***Initialize DSP***
    NOP
    NOP
    DP=#0;                ;Memory Mapped Registers in Data Page 0
    CLKMD = #4007h        ;Set c5402 DSP clock to 100MHz.

    INTM=#1               ;Disable maskable interrupts.
    CPL=#0                ;Using DP for relative addressing mode
    SXM=#0                ;No Sign Extension
    PMST = #00F8h;        ;Re-map Interrupt Vector Table to 0x0080h in
                        ; Program Memory
    IMR = #0100h;         ;Enabled Interrupt #3
    IFR = #0FFFFh;       ;Clear all Pending Interrupts

    SWWSR = #2000h;       ;Need Wait-State of at least 2.

    DP=#AD_DP             ;Change Data Page to where Variables are stored.

    @Temp=#0088h          ;Enable INT3# From Daugthercard.
    port(DSP_CPLD_CNTL1) =@Temp ;

    @Temp=#0001h
    port(DSP_CPLD_CNTL2) =@Temp ;Select daughterboard as source for McBSP0
    call McBSP0_init      ;Set McBSP0 registers

    INTM=#0               ;enable global interrupts
    
```

```

DP=#AD_DP          ;
XF = 0             ;SET CSz. ADC Chip Select is held low throughout
                   ; this program.

A=@_StoreAt
AR7=A              ;Address to begin storing Samples at.
@Quit_Now=#0       ;0->Continue sampling
                   ;1->Return to C

call FIFO_LEVEL_SET ;Determine FIFO Trigger Level
@Next_Table=#0

*Configure the ADC * ;Transmit Word Store in Accum. A. Receive Word Stored in
                   ; Accum. B
A=@_Config_Command ;Configuration Word, Choose external SCLK, and Single shot
                   ; mode.
call ADC_Write_Read ;Write and Read from McBSP0. Write the Configuration Data
                   ; to ADC, and read out receiver data register. Receiver
                   ; data is stored in Accumulator B. In this case we don't
                   ; care what was received.

Wait:
A=@FIFO_LEVEL
BRC = A
NOP
NOP
blockrepeat(continueCYCLE-1)

A=@_Conv_Command
call ADC_Write_Read
repeat(#6)
nop
continueCYCLE:      ;end_block repeat

idle(#1)           ;DSP power-down until INT3# occurs
RETURN             ;Sweep Sequence Completed Return to calling Function

**Initialize McBSP0 for 16-bit transfers, SCLK derived from CPU clock **
**Set SCLK frequency using @_CLKGDV **
McBSP0_init:

mmr(McBSP0_SPSA) = #SPCR1      ;Reset McBSP0 Receiver
A=mmr(McBSP0_SPSD)             ;
A = #0FFFEh & A                ;RRST*=0
mmr(McBSP0_SPSD) =A

mmr(McBSP0_SPSA) = #SPCR2      ;Reset McBSP0 Receiver
A=mmr(McBSP0_SPSD)             ;
A= A & #0FF7Fh                 ;FRST*=0
A= A & #0FFBFh                 ;GRST*=0
A= A & #0FFFEh                 ;XRST*=0

A=A|#0200h                    ;enable free running mode
mmr(McBSP0_SPSD) = A           ;

mmr(McBSP0_SPSA) = #PCR        ;FSXM & CLKXM output pin driven

```

```

mmr(McBSP0_SPSD) = #0A00h      ;by sample-rate generator
                                ;FSRM & CLKRM input pins
                                ;driven by external source

mmr(McBSP0_SPSA) = #RCR1      ;
mmr(McBSP0_SPSD) = #0040h      ;One 16-bit Word per frame

mmr(McBSP0_SPSA) = #RCR2      ;One 16-bit Word per frame
mmr(McBSP0_SPSD) = #0041h      ;one bit receive bit data delay

mmr(McBSP0_SPSA) = #XCR1      ;
mmr(McBSP0_SPSD) = #0040h      ;One 16-bit Word per frame
mmr(McBSP0_SPSA) = #XCR2      ;One 16-bit Word per frame
mmr(McBSP0_SPSD) = #0041h      ;one bit transmit data delay

mmr(McBSP0_SPSA) = #SRGR1      ;FS width=one clock
A= #0100h                        ;
NOP
NOP
B=@_CLKGDV                        ;CLKGDV=9->10 Mhz
NOP
NOP
A=A|B                              ;CLKGDV=4->20 Mhz
mmr(McBSP0_SPSD) = A              ;SRG clock divider=9 (CPU Clock /1+CLKGDC)=10 MHz
mmr(McBSP0_SPSA) = #SRGR2        ;SRG clock derived from CPU clock
mmr(McBSP0_SPSD) = #2018h        ;FSX due to transfer DXR->XSR

mmr(McBSP0_SPSA) = #SPCR2        ;
A=mmr(McBSP0_SPSD)
A= A | #00080h                    ;FRST*=1
A= A | #0040h                    ;GRST*=1
mmr(McBSP0_SPSD) = A              ;Frame Generator and sample
                                ;sample-rate generator enabled

A= A | #0001h                    ;XRST*=1
mmr(McBSP0_SPSD) = A              ;Transmitter enabled

    repeat(#6)                    ;wait for transmitter to become active
    NOP

    mmr(McBSP0_SPSA) = #SPCR1      ; RRST*=1
A= mmr(McBSP0_SPSD)
A= A | #0001h
mmr(McBSP0_SPSD) = A              ;receiver enabled

    repeat(#6)                    ;wait for receiver to become active
    NOP

RETURN

**Function Write to Transmitter and Read from Receiver of McBSP0**
**Data to be written out must be stored in Accum. A,          **
**Data received in stored in Accumulator B                   **
ADC_Write_Read:          ;input in A
    PUSH (AR2)           ;Save Registers
    
```

```

    PUSH (AR1)
    PUSH (AR0)
Wait_On_Transmitter:
    DP=#0
    mmr(McBSP0_SPSA) = #SPCR2      ;Check to see if transmitter is ready to send
    B=mmr(McBSP0_SPSD)              ;new data.

    B=B & #0002h;
    AR1=B
        nop
    nop
    AR0=#2
    nop
    nop
    TC      = (AR0 == AR1)
    if (NTC) goto Wait_On_Transmitter ;If transmitter is full wait until it is not.
    AR2=#McBSP0_DXR1
    nop
    *AR2 = A ;Send out to ADC
Wait_On_Receiver:
    DP=#0
    mmr(McBSP0_SPSA) = #SPCR1      ;
    B=mmr(McBSP0_SPSD)              ;Check to see if Receiver is FULL with Data
    AR0=#2
    B=B & #0002h;
    AR1=B
    nop
    nop
    TC = (AR0 == AR1)
    if (NTC) goto Wait_On_Receiver  ;If receiver is not ready with new data,
                                     then wait.

    DP=#AD_DP
    AR2=#McBSP0_DRR1
    NOP
    B=*AR2                          ;Read out ADC
    AR0 =POP()
    AR1 =POP()
    AR2 =POP()
    RETURN
**Function determines the Level FIFO is set to trigger interrupt. **
**FIFO Level is saved in variable FIFO_Level **
FIFO_LEVEL_SET:

    A=@_Config_Command              ;Load ADC configuration Word and determine
    A=A & #0003h                    ;FIFO Trigger Level

    @Temp=A
    TC=(@Temp == #0)                ;If it set to FIFO level 7?
    if (NTC) goto NOT_FIFO__LEVEL_7
    A=#7
    @FIFO_LEVEL=A

```

```

        goto FIFO_SET
NOT_FIFO__LEVEL_7:           ;Not Level 7, if FIFO set to trigger when
        TC=(@Temp == #1)      ;level 5 is filler?
        if (NTC) goto NOT_FIFO__LEVEL_5
        A=#5
        @FIFO_LEVEL=A

        goto FIFO_SET
NOT_FIFO__LEVEL_5:           ;Not set to trigger at level 7 or 5.  Maybe
                               ;Level 3?
        TC=(@Temp == #2)
        if (NTC) goto NOT_FIFO__LEVEL_3
        A=#3
        @FIFO_LEVEL=A

        goto FIFO_SET
NOT_FIFO__LEVEL_3:           ;Not Set to Trigger at Level 7,5,3. Then has to be 1.
        A=#1                   ;FIFO has Two Levels 0,1.  When these levels are filled
        @FIFO_LEVEL=A         ;TLV254x will generate an Interrupt.
FIFO_SET:
        RETURN

ISR_int03:
        DP = #AD_DP
        A=@FIFO_LEVEL         ;Load in Block Repeat Counter Register the number of
        BRC = A               ;FIFO levels to Read.
        NOP
        NOP
        blockrepeat(Readout_FIFO-1)
        repeat(#5)
        nop
        A=#FIFO_Read          ;Write Command FIFO Read.
        call ADC_Write_Read    ;This prompts the ADC to shift
                               ;FIFO contents out.

        TC = (@Next_Table = #0) ;Decide where to store FIFO Contents
        if (TC) goto Table_0 ;Either in Table One or Table Two.
        TC = (@Next_Table = #1) ;This code is written for a FIFO trigger level set to 1.
        if (TC) goto Table_1
        goto Read_Next_FIFO
Table_0:
        *AR7+ = data(@ADWORD) ;point to first date location of the storage table
        @Next_Table =#1
        goto Read_Next_FIFO
Table_1:
        *AR6+ = data(@ADWORD) ;point to first date location of the storage table
        @Next_Table =#0

Read_Next_FIFO:
        nop
Readout_FIFO:
    
```

```
    goto Continue
Continue:
    return_enable
.end
```

## Appendix E Repeat Sweep Conversion

```
*****
* TITLE           : TLV2544/48 Interface routine                *
* FILE            : TLV2548Rep_Sweep_M.ASM                    *
* FUNCTION        : MAIN                                       *
* PROTOTYPE       : void RepeatSweepMode ( )                  *
* CALLS           : N/A                                         *
* PRECONDITION    : N/A                                         *
* POSTCONDITION   : N/A                                         *
* DESCRIPTION     :
* This routine is written to demonstrate sweep mode of the TLV2548 ADC.
* The program expects the calling program to store the configuration word
* Config_Command, the channel Conversion Command in _Conv_Command, the
* memory location to store sample in _StoreAt, and the number of samples
* to gather in _NumtoStore. The code starts off by setting the DSP, CPLD,
* and McBSP0 registers for this interface. The ADC is then configured, and
* conversions triggered. Once the FIFO is filled, the DSP sits idle until
* an interrupt is received from the ADC. When an interrupt is received,
* the DSP jumps to Interrupt Service Routine (ISR). The DSP reads the data
* out of the FIFO, stores the data in the respective data table, checks to
* see if it has collected enough samples, then returns to the main
* program. There are only two data tables to store the converted data. The
* user must initialize additional tables.
*
* AUTHOR          : AAP Application Group, L. Philipose, Dallas, Tx
*                  CREATED 2000(C) BY TEXAS INSTRUMENTS INCORPORATED.
* REFERENCE       : TMS320C54x User's Guide, TI 1997
*                  : Data Acquisition Circuits, TI 1999
*****
```

```

        .title    "TLV2544/48 ADC"
        .mmregs
        .width    80
        .length   55

        .sect    ".vectors"
        .copy    "C5402vec.asm"

        .sect    ".data"
        .include regs.h
**Global Variables**
        .global  _RepeatSweepMode
        .global  _StoreAt
        .global  _NumToStore
        .global  _Config_Command
```

```

.global _Conv_Command
.global _CLKGDV
**Variables**
AD_DP          .usect ".varibl", 0 ;Data page label for variables.
_StoreAt        .usect ".varibl", 1 ;Address to begin Storing Samples
_NumToStore    .usect ".varibl", 1 ;Total Number of Samples to Collect
_Config_Command .usect ".varibl", 1 ;Configuration Word
_Conv_Command  .usect ".varibl", 1 ;Command word to ADC
_CLKGDV        .usect ".varibl", 1 ;Variable used to set CLKX/R on McBSP0
ADWORD         .usect ".varibl", 1 ;Converted Data stored t
Temp           .usect ".varibl", 1 ;
FIFO_LEVEL     .usect ".varibl", 1 ;FIFO set to Trigger Interrupt at this Level
EndOfTable     .usect ".varibl", 1 ;End of Table
Next_Table     .usect ".varibl", 1 ;Next Table to store Sample
Quit_Now       .usect ".varibl", 1 ;Next Table to store Sample
Input_CH_0     .set 00000h
Input_CH_1     .set 01000h
Input_CH_2     .set 02000h
Input_CH_3     .set 03000h
Input_CH_4     .set 04000h
Input_CH_5     .set 05000h
Input_CH_6     .set 06000h
Input_CH_7     .set 07000h
SW_PWR_DWN     .set 08000h
Read_CFR       .set 09000h
Write_CFR      .set 0A000h
Self_Test_Half .set 0B000h
Self_Test_MINUS .set 0C000h
Self_Test_PLUS .set 0D000h
FIFO_Read      .set 0E000h

.sect ".text"
_RepeatSweepMode:
_MAIN:
start:

***Initialize DSP***
    CPL=#0                ;Using DP for relative addressing mode
    NOP
    NOP

```

```

DP=#0;                ;Memory Mapped Registers in Data Page 0
CLKMD = #4007h        ;Set c5402 DSP clock to 100MHz.

INTM=#1                ;Disable maskable interrupts.
SXM=#0                ;No Sign Extension
PMST = #00F8h;        ;Re-map Interrupt Vector Table to 0x0080h in
                    ;Program Memory
IMR = #0100h;         ;Enabled Interrupt #3
IFR = #0FFFFh;        ;Clear all Pending Interrupts

SWWSR = #2000h;        ;Need Wait-State of at least 2.

DP=#AD_DP              ;Change Data Page to where Variables are
                    ;stored.

@Temp=#0088h           ;Enable INT3# From Daugthercard.
port(DSP_CPLD_CNTL1) =@Temp ;

@Temp=#0001h
port(DSP_CPLD_CNTL2) =@Temp ;Select daughterboard as source for McBSP0
call McBSP0_init        ;Set McBSP0 registers

INTM=#0                ;enable global interrupts

DP=#AD_DP              ;
XF = 0                 ;SET CSz. ADC Chip Select is held low throughout this program.
A=@_StoreAt
AR7=A                 ;Address to begin storing Samples at.
B=@_NumToStore        ;Add number of samples requires to table starting location.
A=B+A
@EndOfTable=A         ;End of Sample Table
AR6=A;

call FIFO_LEVEL_SET    ;Determine FIFO Trigger Level
@Next_Table=#0         ;Initialize Data Table Counter
*Configure the ADC *   ;Transmit Word Store in Accum. A. Receive Word Stored
                    ; in Accum. B
A=#_Config_Command    ;Configuration Word, Choose external SCLK, and Single
                    ; shot mode.
call ADC_Write         ;Write and Read from McBSP0. Write the Configuration
                    ; Data to
call ADC_Read          ;ADC, and read out receiver data register. Receiver
                    ; data is stored
    
```

```

;in Accumulator B. In this case we don't care what
; was received.
@Quit_Now=#0          ;0->Continue sampling
                    ;1->Return to C

Wait:
TC = (@Quit_Now == #1)
  if (TC) goto Return_To_C

A=@FIFO_LEVEL
BRC = A
NOP
NOP
blockrepeat(continueCYCLE-1)

A=@_Conv_Command
call ADC_Write
call ADC_Read
repeat(#6)
nop
continueCYCLE:      ;end_block repeat

idle(#1)           ;DSP power-down until INT3# occurs
goto Wait

Return_To_C:

RETURN
McBSP0_init:

mmr(McBSP0_SPSA) = #SPCR1 ;Reset McBSP0 Receiver
A=mmr(McBSP0_SPSD)      ;
A = #0FFFEh & A        ;RRST*=0
mmr(McBSP0_SPSD) =A
mmr(McBSP0_SPSA) = #SPCR2 ; Reset McBSP0 Receiver
A=mmr(McBSP0_SPSD)
A= A & #0FF7Fh        ;FRST*=0
A= A & #0FFBFh        ;GRST*=0
A= A & #0FFFEh        ;XRST*=0

A=A|#0200h           ;enable free running mode

```

```

mmr(McBSP0_SPSD) = A          ;

mmr(McBSP0_SPSA) = #PCR      ;FSXM & CLKXM output pin driven
mmr(McBSP0_SPSD) = #0A00h    ;by sample-rate generator
                                ;FSRM & CLKRM input pins
                                ;driven by external source

mmr(McBSP0_SPSA) = #RCR1     ;
mmr(McBSP0_SPSD) = #0040h    ;One 16-bit Word per frame

mmr(McBSP0_SPSA) = #RCR2     ;One 16-bit Word per frame
mmr(McBSP0_SPSD) = #0041h    ;one bit receive bit data delay

mmr(McBSP0_SPSA) = #XCR1     ;
mmr(McBSP0_SPSD) = #0040h    ;One 16-bit Word per frame
mmr(McBSP0_SPSA) = #XCR2     ;One 16-bit Word per frame
mmr(McBSP0_SPSD) = #0041h    ;one bit transmit data delay

mmr(McBSP0_SPSA) = #SRGR1    ;FS width=one clock
A= #0100h                    ;
NOP
NOP
B=@_CLKGDV                   ;CLKGDV=9->10 Mhz
NOP
NOP
A=A|B                         ;CLKGDV=4->20 Mhz
mmr(McBSP0_SPSD) = A          ;SRG clock divider=9 (CPU Clock /1+CLKGDC)=10 Mhz
mmr(McBSP0_SPSA) = #SRGR2    ;SRG clock derived from CPU clock
mmr(McBSP0_SPSD) = #2018h    ;FSX due to transfer DXR->XSR

mmr(McBSP0_SPSA) = #SPCR2    ;
A=mmr(McBSP0_SPSD)
A= A | #00080h                ;FRST*=1
A= A | #0040h                 ;GRST*=1
mmr(McBSP0_SPSD) = A          ;Frame Generator and sample
                                ;sample-rate generator enabled

A= A | #0001h                 ;XRST*=1
mmr(McBSP0_SPSD) = A          ;Transmitter enabled
    
```

```

repeat(#6)                ;wait for transmitter to become active
NOP

mmr(McBSP0_SPSA) = #SPCR1    ;RRST*=1
A= mmr(McBSP0_SPSD)
A= A | #0001h
mmr(McBSP0_SPSD) = A        ;receiver enabled

repeat(#6)                ;wait for receiver to become active
NOP

RETURN

**Function Write to Transmitter and Read from Receiver of McBSP0 **
**Data to be written out must be stored in Accum. A,           **
**Data received in stored in Accumulator B                     **
ADC_Write:                ;input in A
    PUSH (AR2)            ;Save Registers
    PUSH (AR1)
    PUSH (AR0)
Wait_On_Transmitter:
    DP=#0
    mmr(McBSP0_SPSA) = #SPCR2 ;Check to see if transmitter is ready to send
    B=mmr(McBSP0_SPSD)      ;new data.

    B=B & #0002h;
    AR1=B
    nop
    nop
    AR0=#2
    nop
    nop
    TC      = (AR0 == AR1)
    if (NTC) goto Wait_On_Transmitter ;If transmitter is full wait until it is not.
    AR2=#McBSP0_DXR1
    nop
    *AR2 = A                ;Send out to ADC
    AR0 =POP()
    AR1 =POP()
    AR2 =POP()

```

```

    RETURN

ADC_Read:                                ;Output in B
    PUSH (AR2)                            ;Save Registers
    PUSH (AR1)
    PUSH (AR0)
Wait_On_Receiver:
    mmr(McBSP0_SPSA) = #SPCR1 ;
    B=mmr(McBSP0_SPSD)          ;Check to see if Receiver is FULL with Data
    AR0=#2
    B=B & #0002h;
    AR1=B
    nop
    nop
    TC = (AR0 == AR1)
    if (NTC) goto Wait_On_Receiver ;If receiver is not ready with new data,
                                    ; then wait.

    DP=#AD_DP
    AR2=#McBSP0_DRR1
    NOP
    B=*AR2                            ;Read out ADC
    AR0 =POP()
    AR1 =POP()
    AR2 =POP()
    RETURN

**Function determines the Level FIFO is set to trigger interrupt. **
**FIFO Level is saved in variable FIFO_Level **
FIFO_LEVEL_SET:

    A=@_Config_Command                ;Load ADC configuration Word and determine
    A=A & #0003h                        ;FIFO Trigger Level

    @Temp=A
    TC=(@Temp == #0)                    ;If it set to FIFO level 7?
    if (NTC) goto NOT_FIFO__LEVEL_7
    A=#7
    @FIFO_LEVEL=A

    goto FIFO_SET
NOT_FIFO__LEVEL_7:                       ;Not Level 7, if FIFO set to trigger when
    TC=(@Temp == #1)                    ;level 5 is filler?
    
```

```

if (NTC) goto NOT_FIFO__LEVEL_5
A=#5
@FIFO_LEVEL=A

goto FIFO_SET
NOT_FIFO__LEVEL_5:           ;Not set to trigger at level 7 or 5.  Maybe
                             ;Level 3?

TC=(@Temp == #2)
if (NTC) goto NOT_FIFO__LEVEL_3
A=#3
@FIFO_LEVEL=A

goto FIFO_SET
NOT_FIFO__LEVEL_3:           ;Not Set to Trigger at Level 7,5,3.  Then has to be 1.
A=#1                         ;FIFO has Two Levels 0,1.  When these levels are filled
@FIFO_LEVEL=A                ;TLV254x will generate an Interrupt.
FIFO_SET:

RETURN

ISR_int03:
DP = #AD_DP
A=@FIFO_LEVEL                 ;Load in Block Repeat Counter Register the number of
BRC = A                       ;FIFO levels to Read.
NOP
NOP
blockrepeat(Readout_FIFO-1)
repeat(#5)
nop
A=#FIFO_Read                 ;Write Command FIFO Read.
call ADC_Write                ;This prompts the ADC to shift
call ADC_Read                 ;FIFO contents out.
A=@EndOfTable                 ;Load into Accum. A, the end of Storage table.
AR0=A
@ADWORD=B                     ;Store converted data in this variable temporarily
TC = (AR0 == AR7)             ;Total Samples Collect = Total Requested?
if (TC) goto Quit

TC = (@Next_Table = #0) ;Decide where to store FIFO Contents
if (TC) goto Table_0 ;Either in Table One or Table Two.

```

```

    TC = (@Next_Table = #1) ;This code is written for a FIFO trigger level set to 1.
    if (TC) goto Table_1
    goto Read_Next_FIFO
Table_0:
    *AR7+      = data(@ADWORD) ;point to first date location of the storage table
    @Next_Table =#1
    goto Read_Next_FIFO
Table_1:
    *AR6+      = data(@ADWORD) ;point to first date location of the storage table
    @Next_Table =#0

Read_Next_FIFO:
    nop
Readout_FIFO:
    goto Continue
Quit:
    @Quit_Now=#1

Continue:

    return_enable
    end
    
```

## Appendix F McBSP Memory Mapped Register Definition

```

*=====
*  FILENAME:  Reg.h
*
*  TMS320VC5402 & 5402 DSK memory mapped reg definition
*
*=====
*----  include the 54xx register map defined under .mmreg directive ----
        .mmregs
*-----  TIMER2 Registers  -----
TIM1      .set      0030h    ; Timer1
PRD1      .set      0031h    ; Timer1 Period Reg
TCR1      .set      0032h    ; Timer1 Ctrl Reg
*-----  McBSP0 Registers  -----
McBSP0_DRR2x .set      0020h    ; McBSP0 Data Rx Reg2
McBSP0_DRR1 .set      0021h    ; McBSP0 Data Rx Reg1
McBSP0_DXR2 .set      0022h    ; McBSP0 Data Tx Reg2
McBSP0_DXR1 .set      0023h    ; McBSP0 Data Tx Reg1
McBSP0_SPSA .set      0038h    ; McBSP0 Sub Bank Addr Reg
McBSP0_SPSD .set      0039h    ; McBSP0 Sub Bank Data Reg
*-----  McBSP1 Registers  -----
McBSP1_DRR2 .set      0040h    ; McBSP1 Data Rx Reg2
McBSP1_DRR1 .set      0041h    ; McBSP1 Data Rx Reg1
McBSP1_DXR2 .set      0042h    ; McBSP1 Data Tx Reg2
McBSP1_DXR1 .set      0043h    ; McBSP1 Data Tx Reg1
McBSP1_SPSA .set      0048h    ; McBSP1 Sub Bank Addr Reg
McBSP1_SPSD .set      0049h    ; McBSP1 Sub Bank Data Reg
*-----  McBSP0 & McBSP1 Sub-Bank Addressed Registers  -----

SPCR1     .set      0000h    ; McBSP Ser Port Ctrl Reg1
SPCR2     .set      0001h    ; McBSP Ser Port Ctrl Reg2
RCR1      .set      0002h    ; McBSP Rx Ctrl Reg1
RCR2      .set      0003h    ; McBSP Rx Ctrl Reg2
XCR1      .set      0004h    ; McBSP Tx Ctrl Reg1
XCR2      .set      0005h    ; McBSP Tx Ctrl Reg2
SRGR1     .set      0006h    ; McBSP Sample Rate Gen Reg1
SRGR2     .set      0007h    ; McBSP Sample Rate Gen Reg2
MCR1      .set      0008h    ; McBSP Multichan Reg1
MCR2      .set      0009h    ; McBSP Multichan Reg2
RCERA     .set      000Ah    ; McBSP Rx Chan Enable Reg Partition A

```

```

RCERB      .set      000Bh    ; McBSP Rx Chan Enable Reg Partition B
XCERA      .set      000Ch    ; McBSP Tx Chan Enable Reg Partition A
XCERB      .set      000Dh    ; McBSP Tx Chan Enable Reg Partition B
PCR        .set      000Eh    ; McBSP Pin Ctrl Reg
*----- General Purpose I/O Registers -----
GPIOCR     .set      003Ch    ; GP I/O Pins Control Reg
GPIOSR     .set      003Dh    ; GP I/O Pins Status Reg
*----- onboard I/O Memory Mapped Register -----
DSP_CPLD_CNTL1 .set      0000h    ; Control Reg1
DSP_CPLD_STAT .set      0001h    ; Status Reg
DSP_CPLD_DMCNTL .set      0002h    ; Data Memory Control Reg
DSP_CPLD_DBIO .set      0003h    ; Daughterboard / GPIO Reg
DSP_CPLD_CNTL2 .set      0004h    ; Control Reg2
DSP_CPLD_SEM0 .set      0005h    ; Semaphore 0
DSP_CPLD_SEM1 .set      0006h    ; Semaphore 1
    
```

## Appendix G Interrupt Vector Table

```

*****
*      FILENAME: 54xVECS.ASM
*      This routine initializes the 54xDSK Interrupt vector table.
*****
        .title "54xxDSK Vector Table Initialization"
        .global _c_int00, _main
        .algebraic
        .ref ISR_int03
        .sect ".vectors"
RESET:  dgoto _main          ; Reset (Hardware and software Reset)
        nop
        nop
NMI:    dgoto NMI           ; Nonmaskable Interrupt
        nop
        nop
*****
*      S/W Interrupts
*****
SINT17  return_enable       ;Software Interrupt #17
        nop
        nop
        nop
SINT18  return_enable       ;Software Interrupt #18
        nop
        nop
        nop
SINT19  return_enable       ;Software Interrupt #19
        nop
        nop
        nop
SINT20  return_enable       ;Software Interrupt #20
        nop
        nop
        nop
SINT21  return_enable       ;Software Interrupt #21
        nop
        nop
        nop
SINT22  return_enable       ;Software Interrupt #22

```

```

        nop
        nop
        nop
SINT23  return_enable      ;Software Interrupt #23
        nop
        nop
        nop
SINT24  return_enable      ;Software Interrupt #24
        nop
        nop
        nop
SINT25  return_enable      ;Software Interrupt #25
        nop
        nop
        nop
SINT26  return_enable      ;Software Interrupt #26
        nop
        nop
        nop
SINT27  return_enable      ;Software Interrupt #27
        nop
        nop
        nop
SINT28  return_enable      ;Software Interrupt #28
        nop
        nop
        nop
SINT29  return_enable      ;Software Interrupt #29
        nop
        nop
        nop
SINT30  return_enable      ;Software Interrupt #30
        nop
        nop
        nop
*****
*      Rest of the Interrupts
*****
INT0:  return_enable      ;External user interrupt #0
        nop
    
```

```

        nop
        nop
INT1:  return_enable      ;External user interrupt #1
        nop
        nop
        nop
INT2:  return_enable      ;External user interrupt #2
        nop              ;Only 2 NOPs
        nop
        nop
TINT:  return_enable      ;Timer0 Interrupt
        nop
        nop
        nop
BRINT0: return_enable     ;McBSP#0 Receiver Interrupt
        nop
        nop
        nop
BXINT0: return_enable     ;McBSP#0 Transmit Interrupt
        nop
        nop
        nop
DMACO:  return_enable     ;DMA Channel 0 interrupt
        nop
        nop
        nop
TINT1:  return_enable     ;Timer1 Interrupt(default) or DMA channel 1 interrupt
        nop
        nop
        nop
INT3:  dgoto    ISR_int03 ;External user interrupt #3
        nop              ;only TWO NOPs
        nop
HPINT:  return_enable     ;HPI Interrupt
        nop
        nop
        nop
BRINT1: return_enable     ;McBSP#1 Receive Interrupt (Default) or DMA Channel 2
                          ;interrupt
        nop
        nop

```

```
    nop
BXINT1:  return_enable      ;McBSP#1 transmit interrupt (Default) or DMA Channel
                                ;3interrupt
    nop
    nop
    nop
DMAC4:   return_enable      ;DMA channel 4 interrupt
    nop
    nop
    nop
DMAC5:   return_enable      ;DMA channel 5 interrupt
    nop
    nop
    nop
```

## Appendix H Program and Data Memory Mapping

```

/*****/
/*    TMS320C54x DSK Plus Linker Command File
/* 16K words on chip DARAM is shared by Prog
/* and Data sections
/*****/
MEMORY
{
    PAGE 0:                /* Pgm space */
    VECS      : origin = 0080h, length = 0080h /* vector table space */
    PROG      : origin = 0100h, length = 1EFFh /* Pgm mem space */

    PAGE 1:                /* Data space */
    DAT0      : origin = 0060h, length = 0020h /* Scratch Pad mem space */
    DAT1      : origin = 2000h, length = 1C00h /* 7K words for Data */
    STK       : origin = 3C00h, length = 0400h /* 1K words for Stack */
}

SECTIONS
{
    .vectors : {} > VECS    PAGE 0 /* Interrupt Vector table */
    .coeffs  : {} > PROG    PAGE 0 /* */
    .data    : {} > PROG    PAGE 0 /* */
    .text    : {} > PROG    PAGE 0 /* Program code goes here */
    .varibl  : {} > DAT1    PAGE 1 /* uninitialized variables */
    .bss     : {} > DAT1    PAGE 1 /* uninitialized variables */
    .stack   : {} > STK     PAGE 1 /* software stack section */
}

```