Freescale Semiconductor
Application Note

Document Number: AN3749
Rev.0, 10/2008

Using ADC and QADC Modules with ColdFire Microcontrollers

The MCF5211/12/13 and MCF522xx ADC Module

The MCF5214/16 and MCF528x QADC Module

by: David Tosenovjan
Technical Information Center, Roznov

1 Introduction

This application note helps understand how to use the ColdFire MCF528x, MCF5214/16 microcontrollers queued analog-to-digital converter and the ColdFire MCF5211/12/13, MCF522xx microcontrollers analog-to-digital converter. It provides basic examples for different ADC and QADC configurations. QADC examples were created for evaluation board M5282EVB and ADC examples for evaluation board M52233DEMO.

The M5282EVB does not include any devices connected to the QADC inputs. It needs to have a signal generator or voltage source (preferably both).

The M52233DEMO includes a potentiometer and accelerometer connected to a few analog pins. No external equipment is needed.

Contents

2 Queued Analog-to-Digital Converter

2.1 Brief Description

- 10-bit, unipolar, and successive approximation converter
- Up to eight analog input channels if internal multiplexing is used
- Up to 18 analog input channels if external multiplexing is used
- Possibility of using as a GPIO
- Conversion command word table with 64 command words
- 64 right-justified unsigned result registers
- 64 left-justified signed result registers
- 64 left-justified unsigned result registers
- Two independent various length queues (can be cut to various numbers of subqueues except software triggered and externally gated modes)

2.2 Terms

Queue 1 (Q1) — Sequence of commands that begin with the first entry of CCW and ends with one of the following options:

- An entry that contains the first EOQ code. Using the EOQ code is strongly recommended.
- One entry before the entry determined by BQ2.

Queue 2 (Q2) — Sequence of commands that begins with the entry the BQ2 points to and ends with second EOQ code or ends at the end of the CCW table if the second EOQ is not used.

Subqueue (SQxx) — Sequence of commands between the beginning and the CCW entry with pause bit set, two entries with pause bit set, or entry with pause bit set, and the end of queue.

Pause — If pause bit in the CCW entry is set the conversion is paused after this command execution and waits for another trigger event. The queue is in a pause state except both externally gated modes and both software-triggered modes where pause state is ignored.

End of queue (EOQ) — If the command has set channel number 63, the whole command is interpreted as end of queue notification. The rest of this entry is not important.

Beginning of queue 2 (BQ2) — An entry of the CCW table pointed to by QACR2[BQ2]. The entry prior can contain the EOQ code.

2.3 Conversion Command Word (CCW) Table

The conversion command word table defines Q1 and Q2. Every row (command, CCW entry) contains an input channel number and input sample time (usually zero) that can be set to a pause state after conversion and to a sample amplifier bypass (not usually used).

Table 1. Example of CCW Configuration with Market Queues and Subqueues

	Command (entry)				QUEUE	SUBQUEUE
	Input channel number	Pause	Sample amplifier bypass	Input sample time		
00	0	No	No	0	Q1	SQ11
01	1	No	No	0		
02	2	No	No	0		
03	3	Yes	No	0		
04	0	No	No	0		SQ12
05	1	No	No	0		
06	2	No	No	0		
07	3	No	No	0		
08	63 (EOQ)	x	x	x		
09	52	No	No	0	Q2	SQ21
10	53	Yes	No	0		
11	52	No	No	0		SQ22
12	53	Yes	No	0		
13	55	No	No	0		SQ23
14	56	No	No	0		
15	63 (EOQ)	x	x	x		

2.4 Modes of Operation

2.4.1 Software Triggered Single-Scan Mode

The Q1/Q2 scan is initiated by asserting the QACR1[SSE1]/QACR2[SSE2] bit. The CCW entry with the pause bit set does not cause the pause state although the PF*n* flag is set and the interrupt is invoked if allowed. After the appropriate queue scan completion the SSE1/SSE2 bit is negated. Another scan may be initiated by another SSE1/SSE2 bit assertion.

2.4.2 Externally Triggered Single-Scan Mode

Triggering can be set for rising or falling edge for the ETRIG*n* signal. The SSE bit has to be asserted to allow queue triggering. After the appropriate queue scan completion, the SSE1/SSE2 bit is negated.

2.4.3 Interval Timer Single-Scan Mode

The timer interval between two triggering events can be set from 2^7 to 2^{17} QCLK cycles in binary multiples. The Q1/Q2 scan is initiated by asserting the SSE1/SSE2 bit. After the appropriate queue scan completion the SSE1/SSE2 bit is negated. The conversion can continue after the SSE1/SSE2 bit is set again by the software. For example, it is useful if the interrupt service routine takes more time than the time between a scan completion and another triggering event. This approach ensures coherent results. For more information go to *MCF5282UM — MCF5282 ColdFire Microcontroller User's Manual*.

2.4.4 Externally Gated Single-Scan Mode

This mode can only be used with Q1. High level on the ETRIG1 (or ETRIG2 if QACR0[TRG] bit is set) input allows scan initialization by asserting bit SSE1 and vice versa. In this mode the pause bit set in CCW entry is ignored during the queue scan. The queue pause flag is redefined and set if the gate closes during the scan. After the scan is complete, the SSE1 bit is negated.

2.4.5 Software Triggered Continuous-Scan Mode

Queue execution begins immediately after the QACR*n*[MQ*n*] settings for this mode. The CCW entry with pause bit set does not cause pause state. The PF*n* flag is set (and interrupt is invoked if allowed). This mode is usually used only for Q2. If used in Q1, Q2 never executes. In this mode, interrupts are not usually used. After the scan completion the queue execution continues immediately from the beginning.

2.4.6 Externally Triggered Continuous-Scan Mode

Triggering can be set for rising or falling edge of the ETRIG*n* signal. After the scan completion another trigger event on the ETRIG*n* is needed for scan continuance. No software involvement is required.

2.4.7 Periodic Timer Continuous-Scan Mode

The timer interval between two triggering events can be set from 2^7 to 2^{17} QCLK cycles in binary multiples. In comparison to the interval timer single-scan mode the SSE1/SSE2 bit does not have any influence on queue scan behavior. After scan completion, the queue execution immediately continues from the beginning.

2.4.8 Externally Gated Continuous-Scan Mode

This mode can only be used with Q1. High level on ETRIG1 (or ETRIG2 if QACR0[TRG] bit is set) input causes cyclic execution of the queue. The pause bit set in the CCW entry is ignored during the queue scan in this mode. The queue pause flag is redefined and is set if the gate closes during the queue scan.

2.5 QADC Programming Examples

2.5.1 Evaluation Tools Needed

Unfortunately the evaluation boards (M5282EVB and M5282LITE) used in this application note do not have a device connected to the QADC inputs. A voltage source/function waveform generator is needed. An oscilloscope is a great benefit.

Connecting the UART0 to the terminal window with these settings is needed:

- 115200 baud
- 8-bits
- No parity
- 1 stop bit
- No flow control

Table 2. Signal Inputs can be Connected to these Pins

Signal name	AN0	AN1	AN2	AN3	AN52	AN53	AN55	AN56	VSSA
Alternate	ANW	ANX	ANY	ANZ	MA0	MA1	ETRIG1	ETRIG2	–
GPIO	PQB0	PQB1	PQB2	PQB3	PQA0	PQA1	PQA3	PQA4	–
EVB pin no.[1]	5	16	17	19	3	14	11	13	1
LITE pin no.[2]	14	13	12	11	10	9	8	7	50

[1] J5 header of MCF5282EVB

[2] MCU_PORT header of MCF5282LITE

When using an evaluation board a few jumpers influence the QADC behavior.

Table 3. Analogue Power Selector

JP24	Selected VDDA and VDDH reference
1–2*	3.3 V
3–4	5 V
5–6	On pin J5–18

Table 4. Analogue Ground Selector

JP23	Selected VSSA reference
1–2*	GND
2–3	Pin J5–1

Table 5. Voltage Reference High Selector

JP19	Selected VRH reference
1–2*	VDDA
2–3	Pin J5–20

Table 6. Voltage Reference Low Selector

JP18	Selected VRL reference
1–2*	VSSA
2–3	Pin J5–2

*Default settings

NOTE

The MCF5282 is a 5 V tolerant device and has up to a 5 V digital output driven by a VDDH input.

In case of using a LITE board, fixed analog references are used:

- VDDA = VRH = +3.3 V
- VSSA = VRL = 0 V

2.5.2 General Programing Scheme

All examples are based on the CodeWarrior 6.4 stationery and are built as separate targets of this project.

Intended to prepare the code example for easy configuration with minimum possible system mistakes. Headers with operating modes and a template for the CCW easy editing are prepared. The BQ2 parameter can be automatically extracted from the CCW table. Interrupt service routines know what results are needed to read.

2.5.2.1 Files Modified or Added in Comparison with the Standard CodeWarrior 6.4 Stationery

Files modified:

- vectors.s

Files added:

- QADC_CCW_table.h
- QADC_opmodes.h

Modified and differs for a particular example:

- main_x.c
- int_handlers_x.c

Added and differs for a particular example:

- QADC_CCW_table_x.c

2.5.2.2 File Description

2.5.2.2.1 vectors.s

Added vectors for the following interrupt service routines:

- EPORT_IRQ7_button_pressing
- QADC_Q1_conversion_pause
- QADC_Q2_conversion_pause
- QADC_Q1_conversion_complete
- QADC_Q2_conversion_complete

2.5.2.2.2 int_handlers_x.c

Added interrupt service routines for the sources mentioned above.

2.5.2.2.3 main_x.c

This file includes a description of the example by displaying a notice on the terminal window, initialization part of EPORT module, interrupt controller module, and the QADC module.

Functions:

- void QADC_init(void) — Initializes the QADC module and its CCW table. This function calls functions set_CCW and get_BQ2.
- void cpu_pause(int usecs) — For the required time in μs, this function can be used for a wait in the loop. It uses the DMA timer 3.

2.5.2.2.4 QADC_opmodes.h

The file includes headers with the list of Q1 and Q2 operating modes for easy configuration of QACR1[MQ1] field and QACR2[MQ2].

2.5.2.2.5 QADC_CCW_table_x.c

This file includes two functions and a template for creating the CCW table.

The table is prepared as a one-dimensional unknown size data field. Every item (row) is composed of macros and is an entry to the CCWtable.

Functions:

- void set_CCW (void) — Copies the CCW table from the data field CCWtable[] located in the file QADC_CCW_table.c to the appropriate registers.
- int32 get_BQ2 (void) — Identifies and returns the beginning of Queue 2. The return value is the index of the first EOQ entry increased by one. The CCW table can contain two entries with the EOQ code. If the CCW table includes only one EOQ code all entries below are considered as Queue 2 and the end of the CCW table is considered as end of Queue 2. If the CCW table does not include any EOQ code default value, 127 is returned. It is the same as the default QACR[BQ2] value and it means that the trigger event for Q2 does not cause any conversion.

2.5.2.2.6 QADC_CCW_table.h

Headers for configuring the CCW table.

2.5.2.3 How Initialization is Performed

The initialization has to be executed after beginning the main routine in these configuration phases:

1. Edge port module, this is because of the used IRQ7 button.
2. Interrupt controller that allows interrupts from the QADC and EPORT — The interrupt sources for the INTC0 are set in such a way that the IRQ7 button has higher priority therefore the conversion pause interrupts:

Table 7. Interrupt Levels and Priorities of Added Interrupt Sources

Interrupt source	Priority	Level
EPORT_IRQ7_button_pressing	Mid (between 3 and 4)	7 (fixed)
QADC_Q1_conversion_pause	3	4
QADC_Q2_conversion_pause	2	4
QADC_Q1_conversion_complete	1	4
QADC_Q2_conversion_complete	0	4

3. QADC Conversion Command Table — The following function is used for copying the CCWtable[] data field to the CCW registers:

```
void set_CCW (void)
{
  int32 i;
  int32 CCWtable_length = sizeof(CCWtable)/2; // length of CCW is 16 bit words

  /**** initialize Conversion Command Word Table ****/

  for (i = 0; i < CCWtable_length; i++)
  {
   MCF5282_QADC_CCW(i) = CCWtable[i];         // copy CCW table to CCW registers
  }
}
```

4. The rest of the QADC — Settings vary according to the given example. All examples use a function for getting the BQ2 parameter from the CCW table. Call of this function is placed to the MCF5282_QADC_QACRx_BQ macro instead of a direct number parameter:

```
int32 get_BQ2 (void)
{
  int32 i;
  int32 CCWtable_length = sizeof(CCWtable)/2; // length of CCW

  /**** identify beginning of Queue 2 ****/

  for (i = 0; i < CCWtable_length; i++)
  {
    if ((MCF5282_QADC_CCW(i) & EOQ) == EOQ)   // identify end of Q1
    {
      return (i+1);                           // return line after first EOQ code
    }
  }
  return (127);                               // no EOQ code (whole CCW is assigned for Q1)
}
```

2.5.2.4 How Event Servicing is Performed

The software waits in the infinite loop after initialization. All examples use only interrupt service routines for the QADC event servicing.

2.5.2.5 Determining what Results are Read

The QADC does not contain any sample ready status registers. For determining results that are supposed to be read only command word pointers QASR1[CWPQ1], QASR1[CWPQ2], and the CCW table can be used.

All QADC interrupt handlers use global static variables previous_CWPQ1 and previous_CWPQ2. These are used for remembering the last read result (by the previous QADC interrupt service routine). The current CWPQ*n* are found during the interrupt service routine and all samples between these two pointers must be read. Information about channel to result register assignments can be found in the appropriate entry of the CCW table. The EOQ must be excluded.

```
for (i = 1 + previous_CWPQ1; i <= CWPQ1; i++)
{
  channel = MCF5282_QADC_CCW(i) & 0x003F;       // get channel number
  if (channel != EOQ)                           // exclude End-of-Queue code
  {
    printf("%d",channel);
    printf(":");
    printf("%6d",(int16) MCF5282_QADC_LJSRR(i)); // signed results
    printf("\n\r");
  }
}
```

The variable previous_CWPQ2 by default has to be set as a pointer to a command prior the beginning of queue 2 (variable BQ2). The BQ2 is read during runtime from the QACR2[BQ2]. The conversion pause interrupt service routine checks variable first_entry_flag and if it is set the variable previous_CWPQ2 is initialized by function first_entry, and the first_entry_flag is cleared.

```
void first_entry (void) /* shared function for all QADC interrupt handlers */
{
  int32 BQ2;                                              // beginning of Q2
  previous_CWPQ1 = -1;                                    // set pointer before
                                                             beginning of Q1
  BQ2 = MCF5282_QADC_QACR2 & MCF5282_QADC_QACRx_BQ(0x7F); // beginning of Q2
  previous_CWPQ2 = BQ2 - 1;                              // set pointer before beginning of Q2
  first_entry_flag = 0;                                   // this function will
                                                             no longer be called
}
```

2.5.3 Example 1 – Using the QADC as General Purpose Output

Both the QA and QB port can be used for general purpose input/output. There are no pin assignment registers. Both functions GPIO and QADC can be used simultaneously. The DDRQ*n* registers determine the data direction and PORTQ*n* registers reflect the current pin states (if configured for input), or store the data to be driven on the corresponding pins (if configured for output).

NOTE

If the output direction is set and a value is assigned, the output data is stored on corresponding pins. If such pins are used afterwards for an analog function, the analog state of the stored digital value is read.

The digital state of the QB port (AN[3:0]) is 0000b after reset and pressing the IRQ7 button increases this state by one. This means that all QB port pins are used for digital output. The state of the QB port is displayed on the terminal window.

The MCF5282 allows using a 5 V digital output to be driven by a VDDH input. If an EVB is used try settings jumper JP24 to position 3–4.

2.5.4 Example 2 – Interval Timer Single-scan Mode of One Channel

The analog input signal can be connected to pin AN0. The voltage level of this signal can be between VRL and VRH. See the above mentioned jumper position.

Pressing the IRQ7 button starts scanning the AN0 in regular intervals set by the clock divider QACR0[QPR] and by the choice interval/periodic timer period from 2^7 to 2^{17} QCLK cycles (by the QACR*n*[MQ]). After each scan the converted value is displayed on the terminal window. The signed results are used.

Pressing the IRQ7 button again stops the scan (start/stop is done by asserting/negating of SSE1 bit). After stop one, more results are displayed because of the interrupt service routine from Q1. Conversion complete is executed once more.

2.5.4.1 Used CCW Table

```
int16 CCWtable[] =
{
/* channel | time | pause | bypass ,
   CHAN(x) | IST(x) | P   | BYP    , */
   CHAN(0) | IST(0)               , // 00
   EOQ1                           // 01
}
```

2.5.4.2 Used Configuration

```
void QADC_init(void)
{
  set_CCW(); // copy CCW table to CCW registers

  MCF5282_QADC_QADCMCR = MCF5282_QADC_QADCMCR_QDBG;  // freeze in debug mode
  MCF5282_QADC_QACR0 = MCF5282_QADC_QACR0_QPR(127);  // clock divider
  MCF5282_QADC_QACR1 =
     MCF5282_QADC_QACRx_CIE |                        // Q1 completion interrupt enable
     MCF5282_QADC_QACRx_PIE |                        // Q1 pause interrupt enable
     MCF5282_QADC_QACRx_MQ(Q1_INTERVAL_TIMER_SINGLE_SCAN_4096); // operating mode
  MCF5282_QADC_QACR2 =
     MCF5282_QADC_QACRx_MQ(Q2_DISABLED) |            // operating mode
     MCF5282_QADC_QACRx_BQ(get_BQ2())   ;            // identify BQ2 parameter
}
```

2.5.5 Example 3 – External-Trigger Continuous-Scan Mode of Multiple Channels

The analog input signal can be connected to pins AN0–AN3 and AN52–AN53. The voltage level of these signals can be between VRL and VRH. Selected edge of signal ETRIG1 (pin AN55) triggers the Q1 scan and the selected edge of signal ETRIG2 (pin AN56) triggers Q2 scan. The queue operation mode selects whether rising or falling edge is used. After each scan the converted value is displayed on the terminal window. The signed results are used. Q1 results are displayed in the first column and Q2 results are in the second column.

2.5.5.1 QADC Low-Power Stop Mode

Pressing the IRQ7 button is used for forcing the QADC to a low power stop mode and restarting from an idle state to a normal operation. The global variable QADC_activity is used to differentiate (condition branch) between these two possibilities.

After restart, wait for the T_{SR} recovery time and reload the QACR1 and QACR2 control registers prior stored.

```
__interrupt__
void EPORT_IRQ7_button_pressing (void)
{

 static vuint16 QACR1, QACR2;

 MCF5282_EPORT_EPFR = MCF5282_EPORT_EPFR_EPF7        // clear IRQ7 flag
 if (QADC_activity == 0)
```

```
 {

  printf("\n\rRESTART QADC\n\r");
  QADC_activity = 1;                          // set flag for QADC active state
  MCF5282_QADC_QADCMCR &= ~MCF5282_QADC_QADCMCR_QSTOP; // restart QADC
  cpu_pause(10);                              // wait 10us to stabilize the analog circuits
  MCF5282_QADC_QACR1 = QACR1;
  MCF5282_QADC_QACR2 = QACR2;                 // store control registers
 }
 else
 {
  printf("\n\rLOW POWER STOP MODE\n\r");
  QADC_activity = 0;                          // set flag for QADC inactive state
  QACR1 = MCF5282_QADC_QACR1;
  QACR2 = MCF5282_QADC_QACR2;                 // reload control registers
  MCF5282_QADC_QADCMCR |= MCF5282_QADC_QADCMCR_QSTOP; // stop QADC
 }
}
```

2.5.5.2 Used CCW Table

```
};int16 CCWtable[] =
{
/* channel  | time   | pause | bypass,
   CHAN(x)  | IST(x) | P     | BYP   , */

   CHAN(0)  | IST(0)                , // 00
   CHAN(1)  | IST(0) | P            , // 01
   CHAN(2)  | IST(0)                , // 02
   CHAN(3)  | IST(0)                , // 03
   EOQ1                             , // 04

   CHAN(52) | IST(0)                , // 05
   CHAN(53) | IST(0)                , // 06
   EOQ2                               // 07
};
```

2.5.5.3 Used Configuration

```
void QADC_init(void)
{
 set_CCW(); MCF5282_QADC_QADCMCR = MCF5282_QADC_QADCMCR_QDBG; //freeze in debug mode

 MCF5282_QADC_QACR0 = MCF5282_QADC_QACR0_QPR(0);             //clock divider value
 MCF5282_QADC_QACR1 =
 MCF5282_QADC_QACRx_CIE |                                   //Q1 completion interrupt enable
 MCF5282_QADC_QACRx_PIE |                                    //Q1 pause interrupt enable
 MCF5282_QADC_QACRx_MQ(Q1_EXT_TRIGGER_CONTINUOUS_SCAN_RISING);//opmode

 MCF5282_QADC_QACR2 =
 MCF5282_QADC_QACRx_CIE |                                   //Q2 completion interrupt enable
 MCF5282_QADC_QACRx_PIE |                                    //Q2 pause interrupt enable
 MCF5282_QADC_QACRx_RESUME |                                 //after suspension begin with
                                                               aborted CCW
 MCF5282_QADC_QACRx_BQ(get_BQ2()) |                          //identify BQ2 parameter
 MCF5282_QADC_QACRx_MQ(Q2_EXT_TRIGGER_CONTINUOUS_SCAN_RISING);//opmode
}
```

2.5.6 Example 4 – Different Modes for Both Queues

- Q1 — Externally gated single-scan mode of multiple channels (AN0–AN3), ETRIG1 signal (pin AN55) is the gate.

  After each scan the converted value is displayed on the terminal window. The signed results are used and the results are displayed in the first column.

  The interval between two scans is defined only by the clock divider QACR0[QPR]. The interval can be extended by using function cpu_pause prior to re-assertion of the SSE1 bit in conversion complete interrupt. Prior to the cpu_pause other interrupts can be enabled by interrupt unmasking in the core status register.

  Gated modes ignore the pause bit set in the CCW entry. If the gate closes during a scan the QASR0[PF1] is set and a pause interrupt occurs. Queue conversion pause interrupt service routine in this example clears the queue pause flag and re-asserts the SSE1 bit.
- Q2 — Software-triggered single-scan mode of multiple channels (AN52, AN53, and AN56). Pressing the IRQ7 button initiates one Q2 scan.

  After each scan the converted value is displayed on the terminal window. The signed results are used and results are displayed in the second column.

2.5.6.1 Used CCW Table

```
int16 CCWtable[] =
{
/* channel | time   | pause | bypass ,
   CHAN(x) | IST(x) | P     | BYP    , */

   CHAN(0) | IST(0)                  ,// 00
   CHAN(1) | IST(0) | P              ,// 01 – pause is ignored here
   CHAN(2) | IST(0)                  ,// 02
   CHAN(3) | IST(0)                  ,// 03
   EOQ1                              ,// 04

   CHAN(52) | IST(0) | P             ,// 05
   CHAN(53) | IST(0)                 ,// 06
   CHAN(56) | IST(0)                 ,// 07
   EOQ2                               // 08
};
```

2.5.6.2 Used Configuration

```
void QADC_init(void)
{
  set_CCW(); // copy CCW table from header file QADC_CCWtable to CCW registers

  MCF5282_QADC_QADCMCR = MCF5282_QADC_QADCMCR_QDBG;          // freeze in debug mode
  MCF5282_QADC_QACR0 = MCF5282_QADC_QACR0_QPR(127);          // clock divider
  MCF5282_QADC_QACR1 =
    MCF5282_QADC_QACRx_CIE |                                  // Q1 completion interrupt enable
    MCF5282_QADC_QACRx_PIE |                                  // Q1 pause interrupt enable
    MCF5282_QADC_QACRx_SSE |                                  // single scan enable
    MCF5282_QADC_QACRx_MQ(Q1_EXT_GATED_SINGLE_SCAN);          // operating mode
  MCF5282_QADC_QACR2 =
```

```
    MCF5282_QADC_QACRx_CIE |                                    // Q2 completion interrupt enable
    MCF5282_QADC_QACRx_PIE |                                    // Q2 pause interrupt enable
    MCF5282_QADC_QACRx_RESUME |                                 // after suspension begin with
                                                                   aborted CCW
    MCF5282_QADC_QACRx_BQ(get_BQ2()) |                          // identify BQ2 parameter
    MCF5282_QADC_QACRx_MQ(Q2_SOFTWARE_TRIGGERED_SINGLE_SCAN);// operating mode
}
```

3 Analog-to-Digital Converter

3.1 Brief Description

- 12-bit resolution
- Two separate and complete ADCs (4 channels per ADC)
- Ability to simultaneously sample and hold two inputs
- Ability to sequentially scan and store up to eight measurements
- Single ended or differential inputs
- Optional interrupts at the end of a scan if an out of range limit is exceeded (high or low), or at zero crossing
- Optional sample correction by subtracting a pre-programmed offset value also used to indicate signed or unsigned results.
- Power saving modes

3.2 Operation Modes

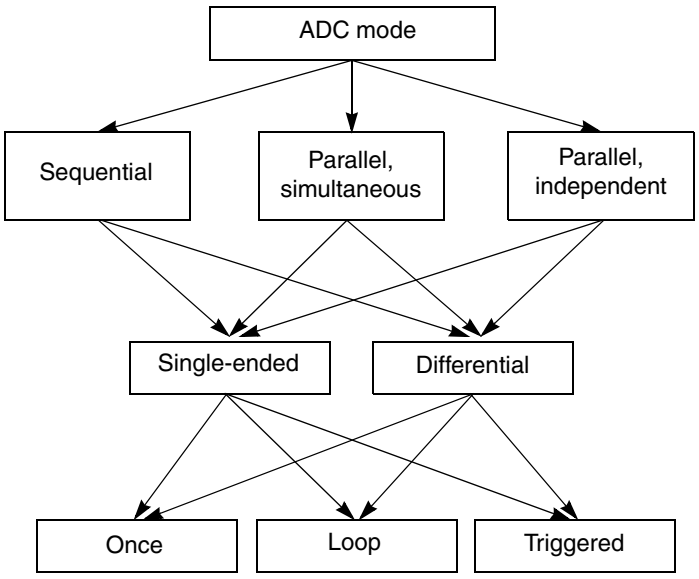

Figure 1. Operation Modes

3.2.1 Sequential Scan Mode

The ADC scan sequencing is defined by registers ADLST1 and ADLST2 and performed in order from SAMPLE0 to SAMPLE7. Any input can be assigned to any result register.

Scan can be initiated by asserting CTRL1[START0] bit or by the rising edge of the SYNCA signal.

3.2.2 Parallel, Simultaneous Scan Mode

The ADCA scan sequencing is defined by register ADLST1 and performs in order from SAMPLE0 to SAMPLE3. The ADCB scan sequencing is defined by registers ADLST2 and performs simultaneously with the ADCA from SAMPLE4 to SAMPLE7.

Starting the scan (both converters simultaneously) can be initiated by asserting the CTRL1[START0] bit or by rising edge of the SYNCA signal.

3.2.3 Parallel, Independent (Non-Simultaneous) Scan Mode

The ADCA scan sequencing is defined by register ADLST1 and performs in order from SAMPLE0 to SAMPLE3. Scan sequencing ADCB is defined by registers ADLST2 and performs simultaneously with the ADCA from SAMPLE4 to SAMPLE7.

Starting the ADCA scan can be initiated by asserting CTRL1[START0] bit or by rising edge of the SYNCA signal.

Starting the ADCB scan can be initiated by asserting CTRL2[START1] bit or by rising edge of SYNCB signal.

3.2.4 Singe-Ended Sample Mode

AN*n* inputs are used as plus terminal and the VREFL as a minus terminal for the A/D core.

3.2.5 Differential Sample Mode

Even AN*n* inputs are used as a plus terminal and odd AN*n* inputs as a minus terminal to the A/D core. Both references (even and odd) in the ADLST*n* registers cause differential measurement.

3.2.6 Once Scan Mode

The single scan mode is initiated once by asserting the START*n* bit or by rising edge of the SYNC*n* signal. Subsequent rising edges of the SYNC*n* signal are ignored until SYNC*n* input is re-armed by CTRL*n*[SYNC*n*] bit.

3.2.7 Triggered Scan Mode

This scan is initiated by asserting the START*n* bit or by rising edge of the SYNC*n* signal. Any other rising edge of the SYNC*n* signal starts another scan.

3.2.8 Loop Scan Mode

A scan automatically restarts after the end of the previous one.

3.3 ADC Programming Examples

3.3.1 Evaluation Tools

Examples are prepared for the demo board M52233DEMO. A potentiometer to analog input AN0 and an accelerometer to analog inputs AN4, AN5, and AN6 (X, Y, and Z axis) are connected to this board. The software can be used with the evaluation board M52235EVB. This board does not have an accelerometer connected and a signal needs to be connected from the external voltage source or waveform generator.

NOTE

There are differences between switch connections in particular the M52233DEMO board revisions. There are only two possibilities Rev.C and lower use of IRQ4/IRQ7 pins for SW1/SW2 switches connection, and Rev.D and upper use of IRQ7/IRQ1 pins for SW1/SW2 switches connection.

If using Rev.C or lower, include mcf5xxx_vectors_REV_ABC.s file to the project and remove mcf5xxx_vectors.s file from the project.

Connect UART0 to the terminal window with these settings:

- 115200 baud
- Eight bits
- No parity
- One stop bit
- No flow control

Both boards use fixed analog references:

- VDDA = VRH = +3.3 V
- VSSA = VRL = 0 V

3.3.2 General Programming Scheme

All examples are based on CodeWarrior 6.4 stationery and built as separate targets of this project.

Example codes are prepared for maximum universality and simplicity for using. All examples share the same interrupt handlers that take care of all the possible interrupt services.

3.3.2.1 Modified or Added Files in Comparison with the Standard CodeWarrior 6.4 Stationery

Modified generally:

- mcf5xxx.vectors.s
- int_handlers.c

Modified and differs for a particular example:

- main_x.c

3.3.2.2 File Description

3.3.2.2.1 mcf5xxx_vectors.s

Added vectors for the following interrupt service routines:

- EPORT_SW1_button_pressing
- EPORT_SW2_button_pressing
- ADCA_conversion_complete
- ADCB_conversion_complete
- ADC_zero_crossing_or_limit_error

3.3.2.2.2 int_handlers.c

Added interrupt service routines for the sources mentioned above.

3.3.2.2.3 main_x.c

This file includes the example description that appears in the terminal window, initialization part of general purpose I/O module, interrupt controller module, and ADC module.

Functions:

- void ADC_init(void) — initializes ADC module

3.3.2.3 How Initialization is Performed

After beginning the main routine all initialization is done in the following configuration phases:

1. Interrupt controller to allow interrupts from ADC and EPORT

Interrupt sources for INTC0 are set in such a way that higher priority has buttons and conversion complete interrupts. See Table 8:

Table 8. Interrupt Levels and Priorities of Interrupt Sources

Interrupt source	Priority	Level
EPORT_SW2_button_pressing	Mid (between 3 and 4)	Fixed
EPORT_SW1_button_pressing	Mid (between 3 and 4)	Fixed
ADC_zero_crossing_or_limit_error	3	1
ADCB_conversion_complete	2	1
ADCA_conversion_complete	1	1

2. A general purpose module to assign all analog pins to their primary function
3. ADC initialization — Settings vary according to the example

3.3.2.4 How the Event Servicing is Performed

Software waits in the infinite loop after initialization. All examples use only interrupt service routines for ADC event servicing.

3.3.2.5 Determining what Results are Supposed to be Read

This type of ADC contains the status register ADSTAT that determines what samples are ready and also contains information about possible zero or limits crossing:

```
ADC_status = MCF_ADC_ADSTAT;        // read status register

for (i=0; i<8; i++)
{
 if ((ADC_status >> i) & 0x0001)
 {
  printf("%d",i);
  printf(":");
  printf("%6d",(int16) MCF_ADC_ADRSLT(i));
  printf(" ");
 }
}
```

Interrupt service routines are written for universal service of all modes.

ADCA_conversion_complete routine has two main branches. First, the sequential or parallel simultaneous mode and the second for the parallel independent mode. In case parallel independent sampling, read half of the result registers in the interrupt service routine initiated by the ADCA. Read the other half in ISR initiated by ADCB.

Sequential mode is possible to determine by CTRL1[SMODE0]:

```
!(MCF_ADC_CTRL1 & 0x0001)
```

Parallel simultaneous mode can be determined by CTRL2[SIMULT]:

```
(MCF_ADC_CTRL1 & 0x0001) && (MCF_ADC_CTRL2 & MCF_ADC_CTRL2_SIMULT)
```

Similar testing is used in routines EPORT_SWx_button_pressing and used for the following condition:

```
!(MCF_ADC_CTRL1 & 0x0006)
```

This determines whether ADC uses the mode once or not. If yes, pressing the button starts the conversion. If no, it means loop or triggered mode is used and pressing the button alternately starts/stops conversion.

3.3.3 Example 1 – Mode Once, One Channel Scan, (Single Ended Input)

This example uses a parallel independent scan. It does not matter what mode is chosen in this case because behavior is the same.

Pressing the SW1 button starts scanning channel AN0 where the demo board has the potentiometer connected. Jumper POT_EN has to be on.

3.3.3.1 ADC Power-Up Sequence

These examples use a normal power mode. First power-up the voltage reference circuit and both converters wait for POWER[PUDELAY] ADC clock cycles until PSTS0, PSTS1, and PSTS2 are cleared:

```
MCF_ADC_POWER &= ~(MCF_ADC_POWER_PD0 | MCF_ADC_POWER_PD1 | MCF_ADC_POWER_PD2);
// power-up converter A, converter B and voltage reference circuit

while (MCF_ADC_POWER & (MCF_ADC_POWER_PSTS0 |
                        MCF_ADC_POWER_PSTS1 | MCF_ADC_POWER_PSTS2)) {};
// stay here as long as converter A, B and voltage reference circuit are power-down
```

3.3.3.2 Clock Divisor Select

The clock divisor is set for maximum conversion clock but not to exceed 5 MHz. It is closely involved with the value of the peripheral system clock that equals ½ of the core clock according to the reference manual by the following equation:

$$\text{DIV} = \text{round up } ((\text{peripheral clock} / 2 \times 5\text{MHz}) - 1) \qquad \textit{Eqn. 1}$$

```
if (CORE_CLOCK % 20000)
  MCF_ADC_CTRL2 = MCF_ADC_CTRL2_DIV(CORE_CLOCK/20000) |     // divison remainder!=0
                  MCF_ADC_CTRL2_STOP1;                      // stop ADCB
else
  MCF_ADC_CTRL2 = MCF_ADC_CTRL2_DIV((CORE_CLOCK/20000)-1) | // divison reminder=0
                  MCF_ADC_CTRL2_STOP1;                      // stop ADCB
```

3.3.3.3 Offset Settings

Offset registers (ADOFS*n*) are used for determining whether the result is signed or unsigned.

0 means positive unsigned results, 2047 means signed results, and 4095 means negative unsigned results:

```
MCF_ADC_ADOFS0 = MCF_ADC_ADOFS_OFFSET(2047);
```

3.3.3.4 Disabling Unused Sample Slots

Setting the ADSDIS[DS1] causes disabling sample slot 1 and all higher sample slots, this means from slot one to slot seven in case of sequential sampling and from slot one to slot three in case of parallel sampling. Similarly ADSDIS[DS4] disables sample slots from four to seven (entire ADCB converter) in case of parallel sampling. In case of a sequential sampling no action is taken.

```
MCF_ADC_ADSDIS = MCF_ADC_ADSDIS_DS1 | // disable SAMPLE1 slot and higher
                 MCF_ADC_ADSDIS_DS4;  // disable SAMPLE4 slot and higher
```

3.3.3.5 Used Configuration

```
MCF_ADC_CTRL1 = MCF_ADC_CTRL1_EOSIE0    | // ADCA end of scan interrupt enable
                MCF_ADC_CTRL1_CHNCFG(0) | // all inputs single ended
                MCF_ADC_CTRL1_SMODE(1)  | // once parallel mode
                MCF_ADC_CTRL1_STOP0;      // stop until button is pressed
```

3.3.4 Example 2 – One Channel Loop (Differential Input)

This example uses a parallel independent scan. It does not matter what mode is chosen in this case because behavior is the same.

Pressing the SW1 button causes start/stop of the scan of the differential input. This consists of channel AN0 where the potentiometer is connected to the demo board and AN1 where you can connect the DC voltage source. Do not exceed 3.6 V. This example uses zero crossing (both edges) and limits interrupts (low limit approximately 0.5 V, high limit 2.8 V).

After each scan the converted value is displayed on the terminal window. The signed results are used. Limits or zero crossing events also cause notices on the terminal window and has a higher priority then the results displayed.

The ADC power-up sequence, clock divisor select, offset settings, and disabling unused sample slots are essentially the same parts as Section 3.3.3, “Example 1 – Mode Once, One Channel Scan, (Single Ended Input)”.

3.3.4.1 Used Configuration

```
MCF_ADC_CTRL1 = MCF_ADC_CTRL1_EOSIE0    | // ADCA end of scan interrupt enable
                MCF_ADC_CTRL1_HLMTIE    | // high limit interrupt enable
                MCF_ADC_CTRL1_LLMTIE    | // low limit interrupt enable
                MCF_ADC_CTRL1_ZCIE      | // zero crossing interrupt enable
                MCF_ADC_CTRL1_CHNCFG(1) | // (AN0 +, AN1 -), the rest single-ended
                MCF_ADC_CTRL1_SMODE(3)  | // loop parallel mode
                MCF_ADC_CTRL1_STOP0;      // stop until button is pressed
```

3.3.5 Example 3 – Loop Sequential Scan of Multiple Channels (Single Ended Inputs)

Pressing the SW1 button causes the scan to start/stop channel AN0 (connected to the potentiometer) and AN4–6 (connected to the accelerometer).

After each scan the converted value is displayed on the terminal window. The signed results are used. First column shows potentiometer value, second, third, and fourth shows X, Y, and Z axis of accelerometer outputs.

The ADC power-up sequence, clock divisor select, offset settings, and disabling unused sample slots are essentially the same parts as Section 3.3.3, “Example 1 – Mode Once, One Channel Scan, (Single Ended Input)”.

3.3.5.1 Channels to Sample Slots Linking

This example does not use default assignments between channels and sample slots. Because of using sequential scan mode any channel to any sample slot can be assigned:

```
MCF_ADC_ADLST1 = MCF_ADC_ADLST1_SAMPLE0(0) | // sample slot 0 for channel 0
                 MCF_ADC_ADLST1_SAMPLE1(4) | // sample slot 1 for channel 4
                 MCF_ADC_ADLST1_SAMPLE2(5) | // sample slot 2 for channel 5
                 MCF_ADC_ADLST1_SAMPLE3(6) ; // sample slot 3 for channel 6
```

3.3.5.2 Used Configuration

```
MCF_ADC_CTRL1 = MCF_ADC_CTRL1_EOSIE0    | // ADCA end of scan interrupt enable
                MCF_ADC_CTRL1_CHNCFG(0) | // all inputs as single ended
                MCF_ADC_CTRL1_SMODE(2)  | // loop sequential mode
                MCF_ADC_CTRL1_STOP0;      // stop until button is pressed
```

3.3.6 Example 4 – Triggered Parallel Independent Scan of Multiple Channels (Single Ended Inputs)

Pressing SW1/SW2 carries out one conversion and starts/stops the ADCA/ADCB converter to accept rising edge triggering signal on the input SYNCA/SYNCB.

After each scan the converted value is displayed on the terminal window. The ADCA conversion scan complete shows potentiometer value and the ADCB conversion scan complete shows the X, Y, and Z axis of the accelerometer outputs. The signed results are used.

The ADC power-up sequence, clock divisor select, offset settings, and disabling unused sample slots are essentially the same parts as Section 3.3.3, “Example 1 – Mode Once, One Channel Scan, (Single Ended Input)”.

3.3.6.1 Used Configuration

```
MCF_ADC_CTRL1 = MCF_ADC_CTRL1_EOSIE0    | // ADCA end of scan interrupt enable
                MCF_ADC_CTRL1_SYNC0     | // rising edge of SYNCA can initiate scan
                MCF_ADC_CTRL1_CHNCFG(0) | // all inputs as single ended
                MCF_ADC_CTRL1_SMODE(5)  ; // triggered parallel
MCF_ADC_CTRL2|= MCF_ADC_CTRL2_EOSIE1    | // ADCB end of scan interrupt enable
                MCF_ADC_CTRL2_SYNC1     ; // rising edge of SYNCB can initiate scan
```

4 References

MCF52235 ColdFire® Integrated Microcontroller Reference Manual, MCF52235RM

MCF52235 ColdFire Microcontroller Data Sheet, MCF52235DS

MCF52235 Device Errata, MCF52235DE

MCF5282 ColdFire® Microcontroller User's Manual, MCF5282UM

MCF5282 Device Errata, MCF5282DE

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN3749
Rev. 0
08/2008