AN2428/D
Rev. 0, 01/2003

An Overview of the HCS12 ATD Module

by **Martyn Gallop,**
Applications Engineering
Freescale, East Kilbride

Introduction

Many of the HCS12 family of 16-bit microcontrollers include support for Analog-to-Digital conversion.

The HCS12 ATD converter is modular in design. At the time of publishing 8-channel and 16-channel implementations exist.

The ATD is available on devices with automotive, industrial and consumer temperature ranges. All of the examples in this document are based on VDDA = 5V but the module is also used on devices that can operate at 3V.

The HCS12 Analog-To-Digital converter (ATD) module is highly autonomous with an array of flexible conversion sequences. It provides all of the timing and controls for both sample and conversion periods for multiple conversions.

Programmable conversion sequences control at which analog source to start conversion, how many conversions to perform and whether these should be on the same or multiple input channels. Any conversion sequence can be selected to continuously repeat without MCU overhead.

Sequences can be started by a write to a single register or by a valid signal on an external trigger input (to synchronize the ATD conversion process with external events).

Conversions in each sequence can be configured for sample time, resolution and result data format.

The Digital-to-Analog Converter (DAC) is a unipolar, successive-approximation converter selectable for 8- or 10-bit resolution and accurate to ± 1 Least Significant Bit (LSB). Typical total unadjusted error is ± 2 counts. Monotonicity is guaranteed for both 8- and 10-bit conversions.

With a 2MHz module clock, the ADC can perform an 8-bit single sample in 6 μs or a 10-bit single conversion in 7 μs (including sample times).

A charge distribution technique is used, removing the need for external sample and hold circuits. An internal sample buffer amplifier minimises the effect of this charge sharing between consecutive conversions.

Unused analog inputs can be used as digital inputs. Each port pin has an associated digital input buffer that can be enabled on a pin-by-pin basis (which avoids any disturbance of analog signals when reading the digital port).

This document sets out to provide an overview of the HCS12 ATD module and its key features. It contains a functional overview, outlines the modes of operation and takes a look the key features. It is intended to be used in conjunction with an appropriate HCS12 Device User Guide and ATD Block User Guide for specific register and device information. For further hardware interfacing guidelines refer to Freescale Application Note AN2429, “Interfacing to the HCS12 ATD Module”.

All specification parameters used in this document should be validated against the current specification electrical parameters for any target device(s).

Figure 1 is a functional block diagram of the ADC module. The module is made up of an analog subsystem and a digital control subsystem.

Analog Subsystem

The analog subsystem consists of a multiplexer, an input sample amplifier, a resistor-capacitor digital-to-analog converter (RC DAC) array, and a high-gain comparator.

The multiplexer selects one of three internal or one of the external signal sources for conversion.

The sample amplifier pre-charges the sample capacitor before it gets connected to the input potential.

The RC DAC array provides the digital-to-analog comparison output necessary for successive approximation conversion.

The comparator indicates whether each successive output of the RC DAC array is higher or lower than the sampled input.

Digital Control Subsystem

The digital control subsystem contains registers and logic to control the conversion process and conversion sequences.

The register map for the HCS12 ATD module is shown in **Table 1**.

Control registers and associated logic select the conversion resolution (eight or ten bits), multiplexer input, and conversion sequencing mode, sample time, and ADC clock cycle.

As each input is converted, the digital control subsystem stores the result, one bit at a time, in the successive approximation register (SAR) and then transfers the result to one of the result registers.

Each result is available in three formats (right-justified unsigned, left-justified signed, and left-justified unsigned).

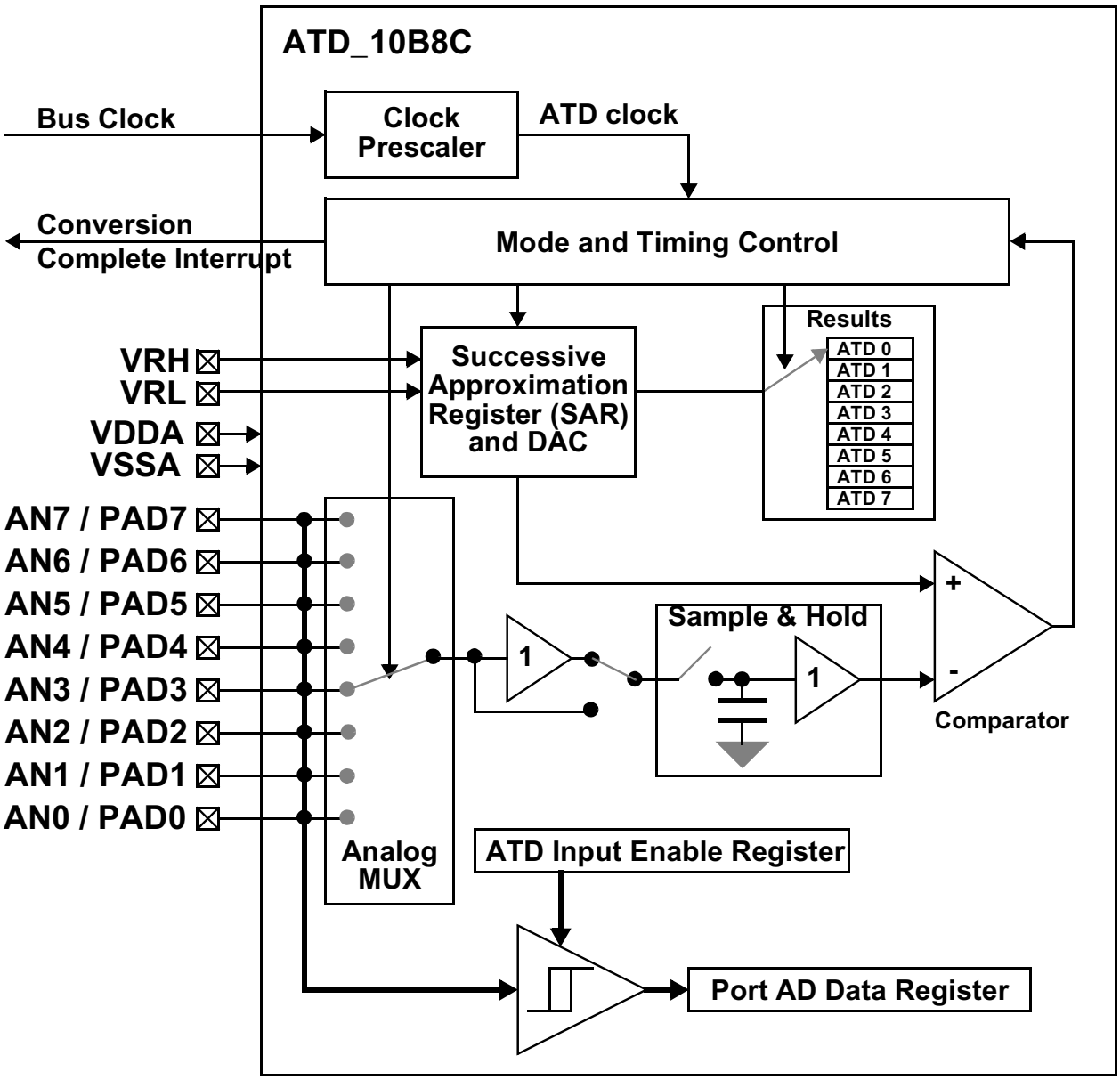

Figure 1. ATD 10B8C Block Diagram

Table 1. ATD Register Summary

Address Offset	Use	Name	Access
$_00	ATD Control Register 0	ATDCTL0	(1)
$_01	ATD Control Register 1	ATDCTL1	(1)
$_02	ATD Control Register 2	ATDCTL2	R/W
$_03	ATD Control Register 3	ATDCTL3	R/W
$_04	ATD Control Register 4	ATDCTL4	R/W
$_05	ATD Control Register5	ATDCTL5	R/W
$_06	ATD Status Register 0	ATDSTAT0	R/W
$_07	Reserved		
$_08	ATD Test Register 0	ATDTEST0	(1)
$_09	ATD Test Register 1	ATDTEST1	W (bit-0 only)
$_0A	**ATD Status Register 2**	**ATDSTAT2**	**R/W**
$_0B	ATD Status Register 1	ATDSTAT1	R/W
$_0C	**ATD Input Enable Register 0**	**ATDDIEN0**	**R/W**
$_0D	ATD Input Enable Register 1	ATDDIEN1	R/W
$_0E	**ATD Port Data Register 0**	**PORTAD0**	**R**
$_0F	ATD Port Data Register 1	PORTAD1	R
$_10, $_11	ATD Result Register 0	ATDDR0H, ATDDR0L	R/W
$_12, $_13	ATD Result Register 1	ATDDR1H, ATDDR1L	R/W
$_14, $_15	ATD Result Register 2	ATDDR2H, ATDDR2L	R/W
$_16, $_17	ATD Result Register 3	ATDDR3H, ATDDR3L	R/W
$_18, $_19	ATD Result Register 4	ATDDR4H, ATDDR4L	R/W
$_1A, $_1B	ATD Result Register 5	ATDDR5H, ATDDR5L	R/W
$_1C, $_1D	ATD Result Register 6	ATDDR6H, ATDDR6L	R/W
$_1E, $_1F	ATD Result Register 7	ATDDR7H, ATDDR7L	R/W
$_20, $_21	**ATD Result Register 8**	**ATDDR8H, ATDDR8L**	**R/W**
$_22, $_23	**ATD Result Register 9**	**ATDDR9H, ATDDR9L**	**R/W**
$_24, $_25	**ATD Result Register 10**	**ATDDR10H, ATDDR10L**	**R/W**
$_26, $_27	**ATD Result Register 11**	**ATDDR11H, ATDDR11L**	**R/W**
$_28, $_29	**ATD Result Register 12**	**ATDDR12H, ATDDR2L**	**R/W**
$_2A, $_2B	**ATD Result Register 13**	**ATDDR13H, ATDDR13L**	**R/W**
$_2C, $_2D	**ATD Result Register 14**	**ATDDR14H, ATDDR14L**	**R/W**
$_2E, $_2F	**ATD Result Register 15**	**ATDDR15H, ATDDR15L**	**R/W**

Bold indicates the additional registers of modules with more than 8 input channels. These locations are reserved and should not be written on modules with 8 or fewer input channels.

1. Factory test only.

Definitions used in this document

- PADxx – port pin used for general purpose digital input
- ANxx – port pin used as an analog signal input pin
- ADxx – analog converter input channel

ATD Configurations

Generally the following parameters apply for an ATD having an analog input multiplexer with *n* Analog Input Channels

- Sequence lengths 1 to *n* conversions
- External trigger on the last channel (ADn)
- The input channel selector wraps around after the last channel (ADn). See **Input Channel Wrap Around**

For example, the D family has two independent 8-channel, 10-bit ATD modules (ATD_10B8C) each with:

- Analog input multiplexer for 8 analog input channels
- Conversion sequence lengths 1 to 8
- External trigger on channel AD07. Pins AN07 (ATD0) & AN15 (ATD1)
- The input channel selector wraps around after AD07 to AD00. From input AN00 to AN07(ATD0) and from input AN15 to AN08 (ATD1)

Converter Functional Description

Analog Input Multiplexer

The analog input multiplexer selects one of the external analog inputs to convert.

The input analog signals are unipolar and must fall within the potential range of VSSA to VDDA (the analog sub-module supply potentials).

The input multiplexer includes protection circuitry to prevent crosstalk between channels when the applied input potentials are within specification.

Sample and Hold Stage

A Sample-and-Hold (S/H) stage accepts analog signals from the input multiplexer and stores them as charge on the sample capacitor. This charge redistribution method eliminates the need for external Sample-and-Hold circuitry.

The sample process has two stages:

1. Initially a sample amplifier (of unity gain) is used to buffer the input analog signal for two cycles to charge the sample capacitor almost to the input potential. This stage prevents charging and discharging the

sample capacitor via the source impedance

2. The sample buffer is then disconnected and the input signal is directly connected to the storage node for a programmable 2, 4, 8 or 16 cycles

Analog-to-Digital Converter Submodule

The Analog-to-Digital Machine uses a successive approximation ATD architecture to perform the analog to digital conversion. The resolution of the ATD converter is selectable as either 8- or 10-bits. It functions by comparing the stored analog sample potential on the sample capacitor with a series of digitally generated analog potentials. By following a binary search algorithm, the converter locates the potential that is closest to the sampled potential. At the end of the conversion process the Successive Approximation Register (SAR) contains the digital representation of the nearest approximation to the sampled signal (for the chosen resolution of the ATD converter) and is transferred to the appropriate results register in the selected format.

Modes of Operation

The analog to digital conversion process can be configured for:

- Converter resolution
- Sample time length
- Module clock frequency
- Result data format

The conversion settings are defined by three control bits:

- SRES8 (in ATDCTL4) controls whether conversions will be 8-bit or 10-bit resolution
- DSGN (in ATDCTL5) determines if the results should be treated as signed or unsigned
- DJM (in ATDCTL5) determines if the results should be left or right justified

and two control values:

- SMP[1:0] (in ATDCTL4) define the length of the sample time
- PR[4:0] (in ATDCTL4) define the ATD module clock frequency (prescaler)

Conversions are performed in a variety of different programmable sequences referred to as conversion modes. Each conversion mode is defined by:

- How many A/D conversions are performed in a sequence

- Which analog input channels are examined during a sequence
- If sequences are performed continuously or not
- Result register assignments

The modes are defined by the settings of five control bits:

- MULT (in ATDCTL5) controls whether the sequence examines a single analog input channel or scans a number of different input channels
- SCAN (in ATDCTL5) controls whether a sequence is to be repeated continuously
- FIFO (in ATDCTL3) controls whether the results register counter is reset at the end of each sequence

and two control values:

- CD/CC/CB/CA (in ATDCTL5) define the input channel at which a sequence should start
- S8C/S4C/S2C/S1C (in ATDCTL3) define the number of conversions in a sequence

Sequences are initiated by writing to control register ATDCTL5 and halted by writing to control registers ATDCTL2, 3, 4 and 5 (writing ATDCLT5 will halt any ongoing sequence and start a new one).

For the continuous sequence modes, conversions will not stop until

- Another non-continuous conversion sequence is initiated and finishes
- The ATD is powered down (by clearing the ADPU control bit)
- The ATD is reset (by a system reset)
- Wait mode is entered (if the AWAI bit is set)
- Full- or Pseudo-Stop mode is entered

The MCU can detect when result data is available in the result registers

- on an interrupt from the ATD Sequence Complete Interrupt Flag (ASCIF)
- by polling the Conversion Complete Flags (CCF)
- by polling the Sequence Complete Flag (SCF).

See **Using the ATD**.

NOTE: *ATD conversion modes should not be confused with MCU operating modes such as STOP, WAIT, IDLE, RUN, DEBUG, and SPECIAL (test) modes or with module-defined operating modes such as power down, fast flag clear and freeze modes*

Converting internal references

An additional control is also available in register ATDTEST1. The state of the SC bit determines whether to perform a 'special conversion' of the internal reference potentials i.e. converting VRH, VRL, (VRL+VRH)/2. This is usually used for test purposes but it may be of use for calibration in some applications.

Figure 2. ATD ATDxTEST1

0	0	0	0	0	0	0	SC

When writing the SC bit, all other bits in this register must be written to zero.

SC – Special Channel Conversion Mode

This selects the internal voltage references for conversion.
1 = Convert internal references.
0 = Convert external signals.

The CD, CC, CB & CA bits in ATDCTL5 are still used to select the channel for conversion:

To select VRH	CD:CC:CB:CA = 0:1:0:0
To select VRL	CD:CC:CB:CA = 0:1:0:1
To select (VRL+VRH)/2	CD:CC:CB:CA = 0:1:1:0

To convert all three references, a three channel, multiple conversion starting at VRH (CD:CC:CB:CA = 0:1:0:0) can be performed.

Using the ATD

The module is powered up and down by respectively setting and clearing the ADPU bit in the ATDCTL2 register. Each ATD module when powered takes ~2mA (VDDA=5V); where power management is a concern it is a good strategy to power down the ATD when it is not converting. Each time the module is powered up, a 20µs delay is required before the first conversion can be started.

To start a conversion sequence under software control, write to the ATDCTL2, ATDCTL3 and ATDCT4 registers to configure the next conversion sequence. The contents of these registers are not affected by executing a conversion sequence or by powering down the module, so it may not be necessary to write all of them each time.

Then write to the ATDCTL5 register to launch the sequence. When external trigger mode is enabled (in ATDCTL2) writing ATDCTL5 will not start a conversion – this will only happen on a valid input to the trigger channel.

Writing any of these four control registers will abort an ongoing sequence; writing to ATDCTL5 will also (re)start a new sequence.

Once a conversion is started:

- The appropriate CCFx flag is set as each conversion result becomes available
- At the end of each conversion sequence, the SCF flag gets set

- If the ATD interrupt is enabled (by setting the ASCIE bit in ATDCTL2) the ASCIF flag echoes the state of the SCF therefore an ATD interrupt service is requested at the end of each conversion sequence

NOTE: *As the interrupt is level sensitive, the SCF flag must be cleared before returning from the interrupt service routine in order to avoid the interrupt request being repeated immediately following the return from interrupt.*

Each CCFx flag can be cleared by

1. Starting a new conversion (all flags get cleared)
2. In 'normal flags' mode (AFFC=0), reading the ATDSTAT1 register with CCFx set, followed by a read of data from the associated result register
3. In 'fast flags' mode (AFFC=1), reading data from the associated result register

The SCF flag is cleared by

A. Starting a new conversion
B. Writing a '1' to it
C. By reading any result register in 'fast flags mode'

There are various ways of monitoring the status of a conversion sequence including:

1. Poll the SCF flag to detect the end of the conversion sequence and then read out the results registers
2. Poll the ATDSTAT0 register and as each bit gets set read the appropriate result register. Check the SCF flag before each read to see when the sequence has been completed
3. On interrupt by the ASCIF flag, read out the results registers

The 'fast flag' mode can be very useful when using an interrupt based strategy because it removes the need for any explicit action to clear the SCF flag (and hence the ASCIF) other than reading the results registers.

A simple example:

```
ATDCTL2 = (ADPU);                                       /* power up the ATD */
ATDCTL3 = (S2C | SC1);                                  /* sequence = 3 conversions */
ATDCTL4 = (PRS3 | PRS2 | PRS1 | PRS0 | SMP0 | SMP1);
/* Prescaler = 32, 16MHz bus = 500KHz ATDCLK,
   programmable sample clocks = 16 */

{Delay 20us from setting ADPU if first conversion}

ATDCTL5 = (DJM | MULT | CA | CB);                       /* start conversion sequence, left justify results,
                                                           single sequence, convert multiple channels,
                                                           starting at channel 3, SCF => 0 */


{wait until SCF => 1}

read result register 0                                  /* channel 3 result */
read result register 1                                  /* channel 4 result */
```

```
read result register 2                                          /* channel 5 result */

/* SCF remains set here but will be cleared on launch of next sequence */
```

Functional Details

ATD Module Clock

A clock prescaler allows the ATD module's internal clock to be configured to within the specified frequency range (500kHz to 2MHz) for different MCU bus clock frequencies. Register ATDxCTL4 controls this function.

For maximum accuracy, selecting a 500kHz clock is recommended where conversion and sample times are not critical.

The bus clock is divided by a programmable constant in order to generate the ATD module's internal clock. An additional benefit of this prescaled clock feature is that it allows further control over the sample period, as changing the module clock also affects the sample and conversion times.

The prescaler is based on a 5-bit modulus counter dividing the bus clock by an integer value between 1 and 32 with the final clock frequency obtained following a further division by 2. So the clock divider values are 2, 4, 6 …… 64 and the minimum bus frequency for ATD operation is 1MHz.

Example C header defines for the ATD clock

```
/**** MCU Defines ****/
#define OSCCLK_FREQ_KHZ 4000L                                                   /* oscillator frequency */
#define BUSCLK_FREQ_KHZ_OSC (OSCCLK_FREQ_KHZ/2)
#define PLL_DIVIDER                     4                                       /* default Pll divider value */
#define PLL_MULTIPLIER  16                                                      /* default Pll multiplier value */
#define BUSCLK_FREQ_KHZ_PLL (OSCCLK_FREQ_KHZ*(PLL_MULTIPLIER)/(PLL_DIVIDER)) /* PLL clock freq = 16MHz */

/* ATD defines */

#define ATD0_CLK_KHZ 2000                                                       /* should be between 500 - 2000 (KHz) */
#define ATD1_CLK_KHZ 500                                                        /* should be between 500 - 2000 (KHz) */

#define ATD0_PRESCALER_OSC (((BUSCLK_FREQ_KHZ_OSC/ATD0_CLK_KHZ)/2)-1)
#define ATD1_PRESCALER_OSC (((BUSCLK_FREQ_KHZ_OSC/ATD1_CLK_KHZ)/2)-1)

#define ATD0_PRESCALER_PLL (((BUSCLK_FREQ_KHZ_PLL/ATD0_CLK_KHZ)/2)-1)
#define ATD1_PRESCALER_PLL (((BUSCLK_FREQ_KHZ_PLL/ATD1_CLK_KHZ)/2)-1)
```

Conversion Timing

The time for a single conversion is the sum of the sample time and the successive approximation time for the resolution required. Sample time and resolution are configurable for each conversion sequence and will be specific for each analog signal and dependent on each application's requirements.

The sample time for a conversion consists of two parts

1. Two fixed cycles where the sample capacitor is charged via the sample buffer
2. Programmable 2 / 4 / 8 / 16 cycles where the input signal is connected directly to the storage capacitor.

$$\text{Conversion time} = \frac{(\text{number of bits resolution} + \text{number of programmed sample clocks} + 2)}{\text{module clock frequency}}$$

Table 2. Conversion timings

Module Clock Frequency	Resolution	Converter Time	2+2 Sample Clocks	2+4 Sample Clocks	2+8 Sample Clocks	2+16 Sample Clocks
2MHz	8-bit	4µs(1)	2µs(1)	3µs	5µs	9µs
2MHz	10-bit	5µs(2)				
500kHz	8-bit	16µs	4µs	12µs	20µs	36µs
500kHz	10-bit	20µs				

1. The fastest sample rate in 8-bit SCAN mode is 1 sample every 6µs = 166 ksamples/s.
2. The fastest sample rate in 10-bit SCAN mode is 1 sample every 7µs = 142 ksamples/s

NOTE: *There is one additional ATD module clock cycle required at the start of each new conversion sequence, i.e. when ATDCTL5 is written.*

For maximum accuracy, selecting sixteen programmable sample cycles is recommended where sample times is not critical.

For fast changing signals, sample time and source impedance should be kept low. Where accuracy is important or source impedance is relatively high, sample times should be increased.

It is possible to increase the effective sequential conversion time on devices with two ATDs by launching an identical conversion of the same signal on both ATDs with a short (software) delay between starting each ATD, i.e. writing the two ATDxCTL5 registers.

Using scan mode, it is possible to effectively double the sample rate of a single converter by using a start delay equal to half of the conversion time (of each converter).This will generate interleaved results across the two ATDs. Once launched, the two conversion sequences will remain synchronized.

Input Channel Wrap Around

On an ATD implementation of *n* input channels, in the case of a multiple conversion sequence (MULT bit = 1), when the input selector goes past AD*n,* it wraps to AD0.

For example, on the D family, input AD7 is connected to pin AN07 of ATD0. When the input selector goes past AD7, it wraps to AD0 (AN00) for the next conversion.

So, if the following pseudo-code is used to execute a five channel conversion sequence starting with AD6, the results registers will contain results as shown in Table 3.

```
ATD0CTL3 = (SC4 | SC1)                          /* 5 conversions in sequence */
ATD0CTL5 = (MULT | CC | CB)                     /* convert multiple channels starting at AN6 */
```

Table 3. ATD results for Example1

ATD0 (D_ATD)	Result
AD6	ATDDR0 / ATD**0**DR0
AD7	ATDDR1 / ATD**0**DR1
AD0	ATDDR2 / ATD**0**DR2
AD1	ATDDR3 / ATD**0**DR3
AD2	ATDDR4 / ATD**0**DR4

FIFO mode

FIFO mode allows the user to select between result register assignments in a conversion sequence.

In the default state, the results register counter is reset to zero at the start of each conversion sequence so the first result converted is stored in ATDDR**0**, the second result in ATDDR**1**, and so on.

In FIFO mode, the results register counter is unchanged by the start of a conversion sequence. Each result is stored in the next consecutive results register. The result register's counter wraps around when the end of the register file is reached. When used in continuous scan mode, this feature can increase the amount of time allowed to read out the result registers before they are overwritten by new data. The conversion counter bits in ATDSTAT can be used to track where the latest conversion result is stored.

External Trigger Source

An external triggering function allows the user to synchronize the ATD conversion process with external events. The device can be configured to trigger on edges or levels of different polarities. Several bits in the ATDxCTL2 register are used to configure this function.

The external trigger source is typically located on the highest input channel and is used to provide an external trigger input signal for the device when it is used in external trigger mode.

On the D family 8-channel converter, the external trigger source is on AD07. On ATD0, the external trigger source is on AN07 and on ATD1 it is on AN15.

Connecting the trigger channels together on devices with two ATDs allows synchronised conversions on the two converters. Examples of the trigger source could be external logic, software driving an output pin, timed conversions using a timer compare output, periodic conversions using a PWM channel.

Data justification

Data output can be right or left justified. Register ATDxCTL5 controls this function.

For 8-bit data, this only controls whether the data is stored at odd or even addresses in the results registers.

For 10-bit data, right justifying can be very helpful for fitting results data into a an appropriate structure that matches a high level compiler data type e.g. right justified data can be treated directly as C integers and the results registers can be treated directly as an array of integers.

Signed/Unsigned control

Conversion data can be signed or unsigned. Signed data is represented as 2's compliment. Register ATDxCTL5 controls this function.

A signed conversion sequence treats the value (VRH – VRL) / 2 as a zero reference. Input signals greater than this value result in positive results (VRH being the most positive). Input signals lower than this result in negative results (VRL being the most negative).

NOTE: *Signed mode cannot be used with right justified data.*

ATD Operational Modes

Power Down Mode

The ATD module can be powered down under program control. This results in minimum quiescent current draw.

The module can be powered in one of three ways.

1. By using the ADPU bit in control register ATDCTL2, the module can be powered down when this bit is cleared
2. When the STOP instruction is executed, the module will power down for the duration of the Stop function
3. If the module Wait enable bit (AWAI) is set and the WAI instruction is executed, the module will power down for the duration of the Wait function

The reset default for the ADPU bit is zero. Therefore, when this module is reset, it defaults into the power down state.

Once the command to power down has been received, the ATD module aborts any conversion sequence in progress and enters Power Down Mode. When the module is powered up again, the module requires a recovery time of ~20µs to stabilize the bias settings in the analog electronics before conversions can be performed.

Powering the module up and down does not affect the contents of the register file and in power down mode, the control and result registers are still accessible.

IDLE Mode

IDLE mode for the ATD module is defined as the state where the ATD module is powered up and ready to perform an A/D conversion, but not actually performing a conversion at the present time. Access to all control, status, and result registers is available. The module is consuming near maximum power.

NOTE: *When not active, the sample-and-hold and analog-to-digital sub-modules disable the clocks at their inputs to conserve power. The analog electronics still draw quiescent current.*

RUN Mode

RUN mode for the ATD module is defined as the state where the ATD module is powered up and currently performing an A/D conversion. Complete access to all control, status, and result registers is available. The module consumes maximum power.

ATD Operation In Different MCU Modes

STOP Mode

Asserting Stop causes the ATD module to power down.

The digital clocks to the module are disabled and the analog sub-module is turned off; this places the module into its power down state and is equivalent to clearing the ADPU control bit in ATDCTL2.

WAIT Mode

If the AWAI control bit in ATDCTL2 is set, then the ATD responds to WAIT mode. If the AWAI control bit is clear, then the ATD ignores the WAIT signal. The ATD response to the wait mode is to power down the module. In this mode, the MCU does not have access to the control, status or result registers.

Background Debug Mode

When the MCU enters Active Background Mode, the ATD module enters an ATD FREEZE Mode.

When debugging an application, it can be useful to have the ATD pause when a breakpoint is encountered. To accommodate this, there are two FREEZE bits in the ATDCTL3 register used to select one of three actions on a BGND request:

1. No effect – continue to execute the current conversion sequence
2. Finish the current conversion and 'freeze' before starting the next sample period
3. 'Freeze' immediately – internal leakage may compromise the accuracy of a frozen conversion depending on the length of the freeze period. When the BDM signal is negated, clock activity resumes

Module Reset

In the case of a system reset, the ATD module is placed into an initialized state. Any ongoing conversion sequence is terminated. The conversion complete flags are cleared and any pending interrupts are cancelled. Note that the control, test, and status registers are initialized on reset.

General Purpose Digital Input Port Operation

For each of the eight input channel there is a digital, 8-bit, input-only port associated with the ATD module. These digital ports are accessed through 8-bit Port Data Registers (PORTADx). Since the port pins are input-only, no data direction register is available for these ports.

An Input Enable Mask register allows each digital input buffer to be enabled / disabled on a pin-by-pin basis. This avoids reads of the digital port disturbing analog signals.

External Interface Considerations

Reference Potentials

The following constraint must be met to obtain full-scale, full range results:

$$VSSA \leq VRL \leq VIN \leq VRH \leq VDDA.$$

If the input level goes outside of this range it will be clipped.

Current Injection

Internal protection clamping diodes are present and the general current injection limits for I/O pins specified in the device user guide should be observed. The current at a pin should be limited to a maximum of 25 mA transient, 2.5mA steady state.

Current injection into an analog channel or neighboring pins can cause inaccuracies in the analog conversion.

Source Impedance

It is recommended that each analog input should have an external capacitor with good high frequency characteristics between the input pin and V_{SSA}.

The size of the external source capacitance will be application dependent but a good guideline is to keep the external capacitor greater than Cf as defined in the Electrical Characteristics section of the specific Device User Guide.

For a maximum 10-bit sampling error of the input voltage ≤ 1LSB, the external filter capacitor (C_f) should be ≥ 1024 * (C_{INS}-C_{INN}), i.e. ≥ 12 nF.

For an 8-bit conversion, 1LSB is 4 times larger so the minimum source capacitance for ≤ 1LSB error will be 256 * (C_{INS}-C_{INN}) i.e. ≥ 3nF.

The source impedance of the signal driver must also be considered when choosing the capacitor size. Optimizing the source impedance is often a compromise:

- External source impedance will form a low pass anti-aliasing filter with the input capacitor to reduce unwanted frequency components and noise. Avoid rolling off higher frequency components of interest in input signals.

The maximum external source impedance of an analog signal is limited by the I/O pin leakage, iin = ± 1μA.

A good guideline for minimising the effect of input leakage is described in the data sheets. When VREF = VRH – VRL = 5.12V, one 8-bit count = 20mV and one 10-bit count = 5mV

- For a maximum 10-bit error of <1/2 LSB, Rs should be ≤ 2.5KΩ (= 2.5mV / 1μA)
- For a maximum 8-bit error of <1/2 LSB, Rs should be ≤ 10KΩ (= 10mV / 1μA)

Larger R**S** values can be used with an associated reduction in accuracy.

For most applications the above component guidelines will be satisfactory.

However these values may require some optimization where

- the input to a channel changes drastically between successive samples in a sequence of conversions, with low sample times and conversion times. Typically, in multiple-channel, continuous-scan conversion sequences with two adjacent channels connected to grossly different potentials
- maximum accuracy is required.

- low-pass filter roll-off is an issue

For further detail on ATD hardware interfacing see Freescale Application Note AN2429, “Interfacing to the HCS12 ATD Module”.

Appendix A - ATD Bit Masks

```
/*ATDCTL2 bit masks */
#define   ASCIF                         0x01              /*atd sequence interrupt flag */
#define   ASCIE                         0x02              /*atd complete interrupt enable */
#define   ETRIGE                        0x04              /*external trigger mode enable */
#define   ETRIGP                        0x08              /*external trigger polarity */
#define   ETRIGLE                       0x10              /*external trigger level/edge control */
#define   AWAI                          0x20              /*atd stops in wait mode */
#define   AFFC                          0x40              /*atd fast flag clear all */
#define   ADPU                          0x80              /*atd enable */
/* function masks */
#define   ENABLE_ATD ADPU
#define   ENABLE_FAST_FLAGS AFFC
#define   STOP_ATD_IN_WAIT AWAI
#define   ENABLE_ATD_INT ASCIE

/*ATDCTL3 bit masks */
#define   FRZ0                          0x01              /*atd background mode */
#define   FRZ1                          0x02              /*atd background mode */
#define   FIFO                          0x04              /*results mapped to fifo mode */
#define   S1C                           0x08              /*sequence length = +1 */
#define   S2C                           0x10              /*sequence length = +2 */
#define   S4C                           0x20              /*sequence length = +4 */
#define   S8C                           0x40              /*sequence length =  8 */
/* freeze mode masks */
#define   FREEZE_NEVER                  0x00
#define   FREEZE_AFTER_CURRENT          0x02
#define   FREEZE_IMMEDIATE              0x03
/* conversion sequence length masks*/
#define   CONVERT_1                     0x08
#define   CONVERT_2                     0x10
#define   CONVERT_3                     0x18
#define   CONVERT_4                     0x20
#define   CONVERT_5                     0x28
#define   CONVERT_6                     0x30
#define   CONVERT_7                     0x38
#define   CONVERT_8                     0x40
#define   CONVERT_9                     0x48
#define   CONVERT_10                    0x50
#define   CONVERT_11                    0x58
#define   CONVERT_12                    0x60
#define   CONVERT_13                    0x68
#define   CONVERT_14                    0x70
#define   CONVERT_15                    0x78
#define   CONVERT_16                    0x00

/*ATDCTL4 bit masks */
                                                        /*all bits clear, clock divider = 2 */
#define   PRS0                          0x01              /* clock divider = +2 */
#define   PRS1                          0x02              /* clock divider = +4 */
#define   PRS2                          0x04              /* clock divider = +8 */
#define   PRS3                          0x08              /* clock divider = +16 */
#define   PRS4                          0x10              /* clock divider = +32 */
#define   SMP0                          0x20              /* programmable sample cycles */
#define   SMP1                          0x40              /* programmable sample cycles */
#define   RES8                          0x80              /* 8-bit conversion */
/* Programmable sample size masks */
#define   SMP2                          0x00
#define   SMP4                          0x20
#define   SMP8                          0x40
#define   SMP16                         0x60
```

```
/*ATDCTL5 bit masks */
#define   CA                          0x01              /* input channel select */
#define   CB                          0x02              /* input channel select */
#define   CC                          0x04              /* input channel select */
#define   MULT                        0x10              /* multiple channel conversion */
#define   SCAN                        0x20              /* continuous conversions sequence */
#define   DSGN                        0x40              /* signed conversion sequence */
#define   DJM                         0x80              /* data justification */
/* input channel select masks */
#define   IP_CH0                      0x00
#define   IP_CH1                      0x01
#define   IP_CH2                      0x02
#define   IP_CH3                      0x03
#define   IP_CH4                      0x04
#define   IP_CH5                      0x05
#define   IP_CH6                      0x06
#define   IP_CH7                      0x07
#define   IP_CH8                      0x08
#define   IP_CH9                      0x09
#define   IP_CH10                     0x0A
#define   IP_CH11                     0x0B
#define   IP_CH12                     0x0C
#define   IP_CH13                     0x0D
#define   IP_CH14                     0x0E
#define   IP_CH15                     0x0F
/* function masks */
#define   RIGHT_JUST                  DJM
#define   LEFT_JUST                   0x00
#define   SIGNED                      DSGN
#define   UNSIGNED                    0x00
#define   CONTINUOUS                  SCAN

/*ATDSTAT0 bit masks */
#define   FIFOR                       0x10              /*fifo overrun flag */
#define   ETORF                       0x20              /*external trigger overrun flag */
#define   SCF                         0x80              /*atd sequence complete flag */

/*ATDSTAT1 bit masks */
#define   CCF0                        0x01
#define   CCF1                        0x02
#define   CCF2                        0x04
#define   CCF3                        0x08
#define   CCF4                        0x10
#define   CCF5                        0x20
#define   CCF6                        0x40
#define   CCF7                        0x80

/*ATDSTAT2 bit masks */
#define   CCF8                        0x01
#define   CCF9                        0x02
#define   CCF10                       0x04
#define   CCF11                       0x08
#define   CCF12                       0x10
#define   CCF13                       0x20
#define   CCF14                       0x40
#define   CCF15                       0x80

/*ATDTEST1 bit masks */
#define   SC                          0x01              /*special channel conversion mode */

/*ATDDIEN/ATDIEN1 bit masks */
#define   DIEN0                       0x01              /*bit masks */
#define   DIEN1                       0x02
#define   DIEN2                       0x04
#define   DIEN3                       0x08
#define   DIEN4                       0x10
#define   DIEN5                       0x20
#define   DIEN6                       0x40
#define   DIEN7                       0x80

/*ATDDIEN0 bit masks */
#define   DIEN8                       0x01              /*bit masks */
#define   DIEN9                       0x02
#define   DIEN10                      0x04
#define   DIEN11                      0x08
#define   DIEN12                      0x10
```

```
#define    DIEN13                        0x20
#define    DIEN14                        0x40
#define    DIEN15                        0x80

/*PORTAD/PORTAD1 bit masks */
#define    PAD0                          0x01                /*bit masks */
#define    PAD1                          0x02
#define    PAD2                          0x04
#define    PAD3                          0x08
#define    PAD4                          0x10
#define    PAD5                          0x20
#define    PAD6                          0x40
#define    PAD7                          0x80

/*PORTAD0 bit masks */
#define    PAD8                          0x01                /*bit masks */
#define    PAD9                          0x02
#define    PAD10                         0x04
#define    PAD11                         0x08
#define    PAD12                         0x10
#define    PAD13                         0x20
#define    PAD14                         0x40
#define    PAD15                         0x80
```

[redacted]

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

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)
support@freescale.com

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 Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
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 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor