Application Note, V1.0, November. 2008

AP16155

XC2000/XE166 family

ADC Result Handling on XC2000/XE166 family of Microcontrollers

Microcontrollers

Never stop thinking

Edition 2008-11-11

Published by
Infineon Technologies AG
81726 München, Germany

© Infineon Technologies AG 2008.
All Rights Reserved.

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

AP16155

Revision History:	2008-11	V1.0
Previous Version:	none	

Page	Subjects (major changes since last revision)

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:

mcdocu.comments@infineon.com

Table of Contents **Page**

1 Introduction

1.1 Overview

This application note provides some detailed information on the result handling mechanism of the ADC module and the programming aspects of it for XC2000/XE166 family of 16-bit Microcontrollers. The configuration options for the result handling of ADC module are discussed with examples.

The example programs and the detailed description on the same will help the users to program the ADC for their application needs. The following key result handling operations of the ADC are explained in detail.

- Storage of the conversion results
- Wait-for-read mode
- Result Event Interrupts
- Result FIFO buffer
- Data reduction or anti-aliasing Filter

1.2 Hardware arrangement

The application examples use the XC2267 starter kit and an optional potentiometer circuit for analog input voltage tuning. For all examples the potentiometer available with the start kit is used whereas for the Multiple FIFO structure example external potentiometer circuit is used.

A typical arrangement of the hardware is shown in Figure 1.

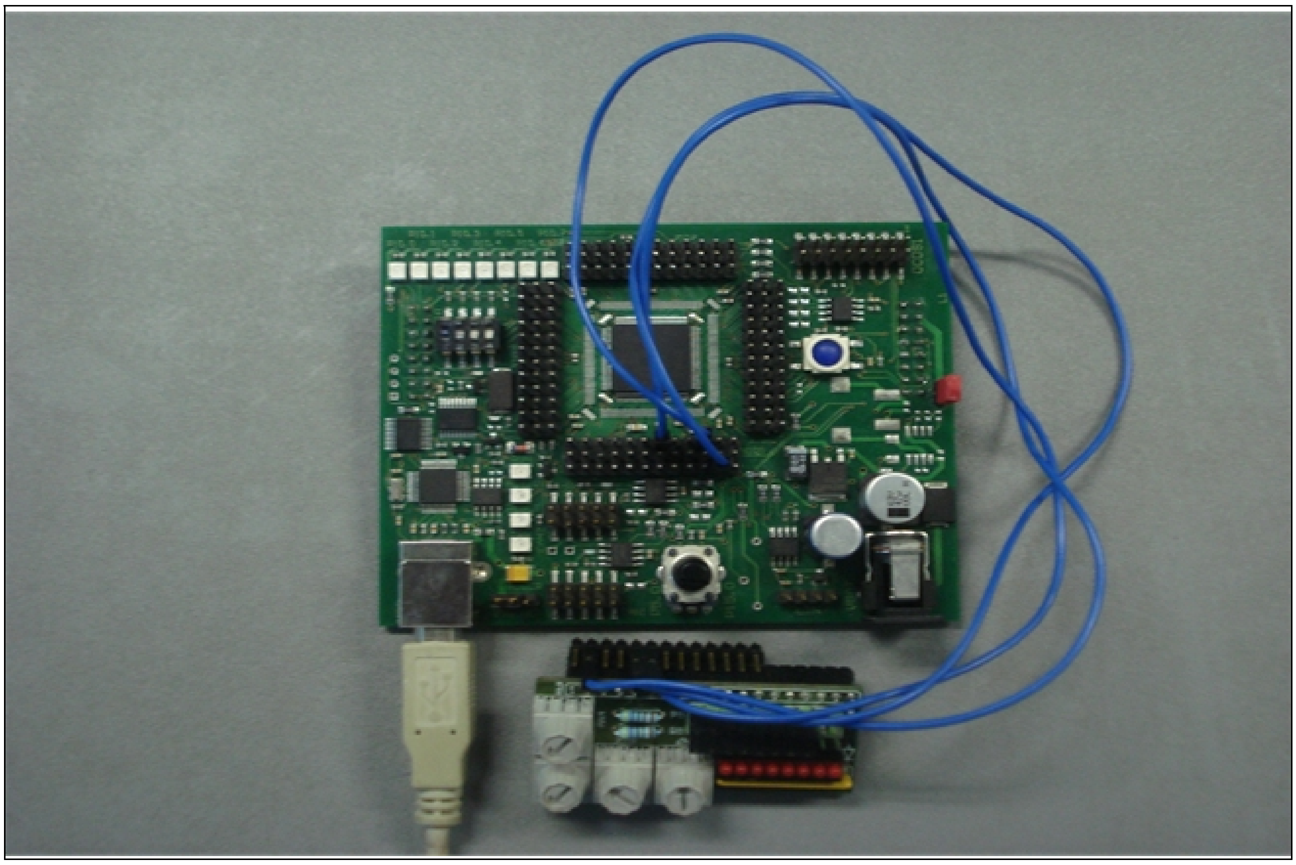


Figure 1 An example hardware setup

1.3 Software requirements

DAvE 2.1 r24 (Digital Application virtual Engineer) is used for the peripheral driver configuration settings.

Keil uVision 3 C166 V6.1 is used for source code addition, compilation and hex file generation.

Memtool V4.01.01 used for downloading the hex file to target.

Docklight V1.6 is used for target communication.

1.4 Conversion result handling

The ADC module contains two ADC kernels that can operate autonomously or can be synchronized to each other. An ADC kernel is a unit used to convert an analog input signal into a digital value and provides means for triggering, data handling and storage.

The conversion result handling is shown in Figure 2.

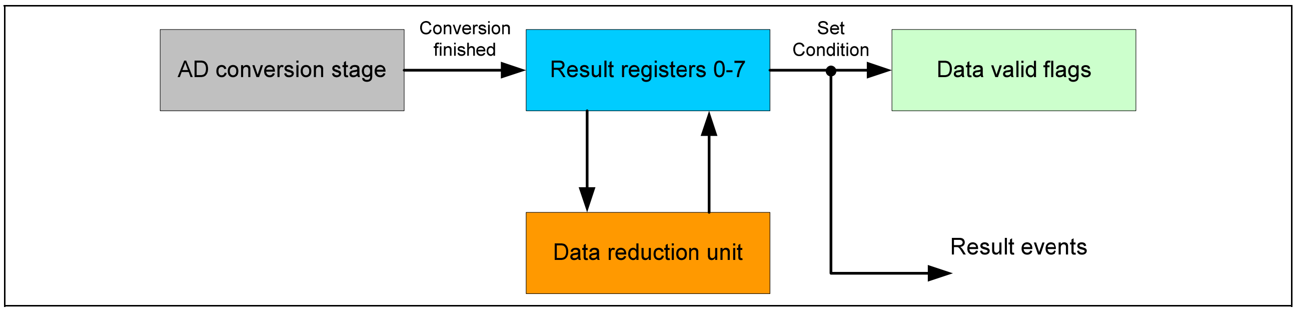


Figure 2 Conversion result handling

The result handling unit of each ADC kernel has 8 independent result registers. The conversion result of each analog input channel can be directed to one of the result registers to be stored there. The result handling block also supports data reduction (e.g. for digital anti-aliasing filtering) by automatically adding up to 4 conversion results before informing the CPU that new data is available.

Additionally, the results registers can be concatenated to FIFO structures to provide storage capability for more than one conversion result without overwriting previous data. This feature also helps to handle CPU latency effects.

2 Storage of the conversion results

Each analog input channel has an associated channel control register CHCTRx (x = 0 -15) contains a pointer bitfield RESRSEL defining the result register to store the conversion result of this channel.

For example to map the result register 7 for channel 0, the RESRSEL bitfield of CHCTR0 register should be set with “111” and is shown in Figure 3.

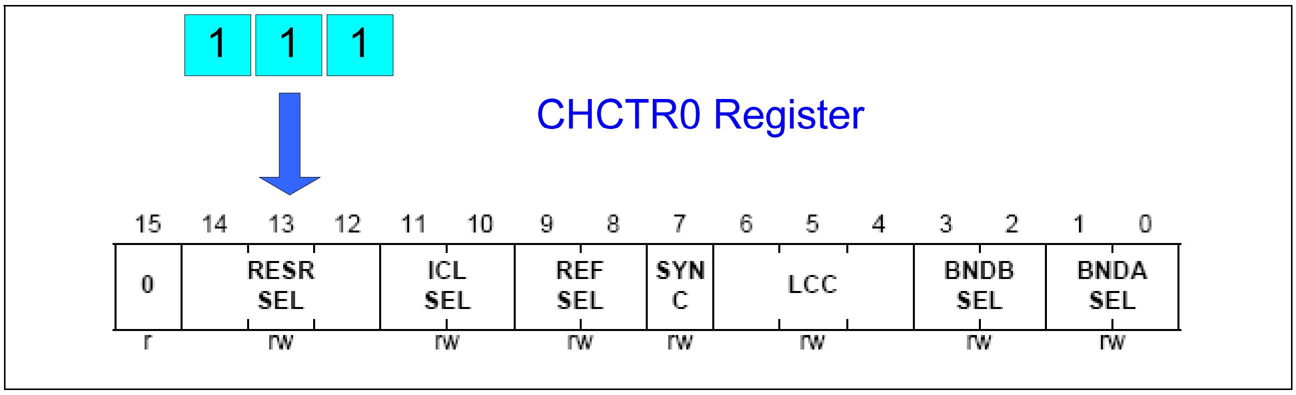


Figure 3 Result register selection for channel 0

Once the conversion is completed and result is stored in the result register, an individual data valid flag VF7 will be set in the VFR register. This indicates that a new data is stored in the result register and can be read out.

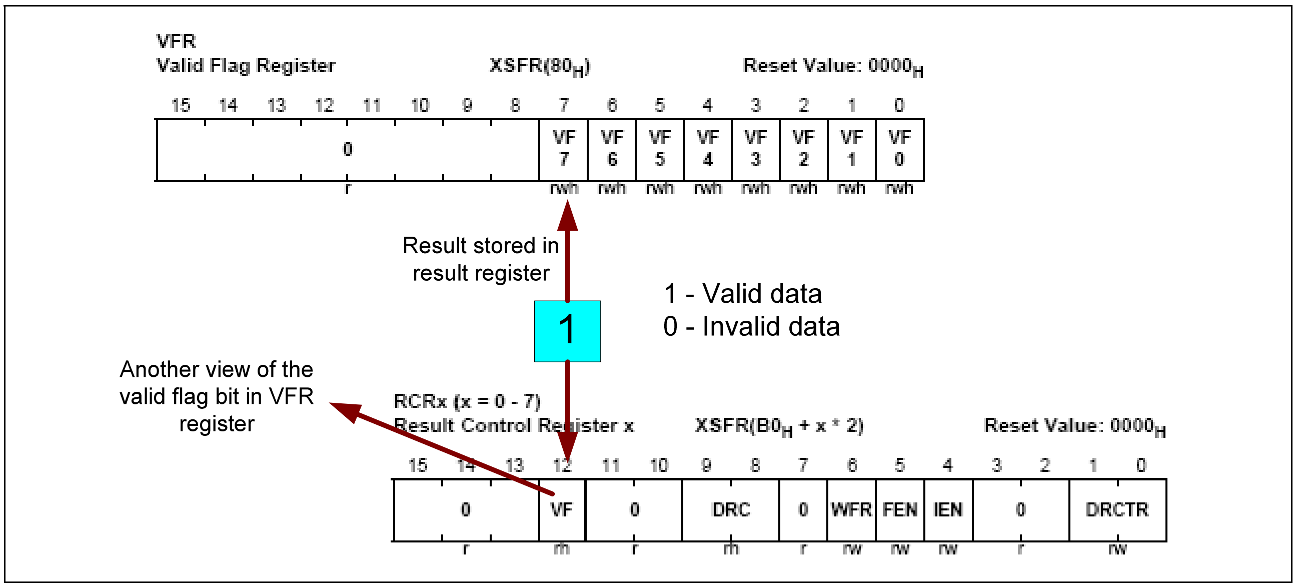
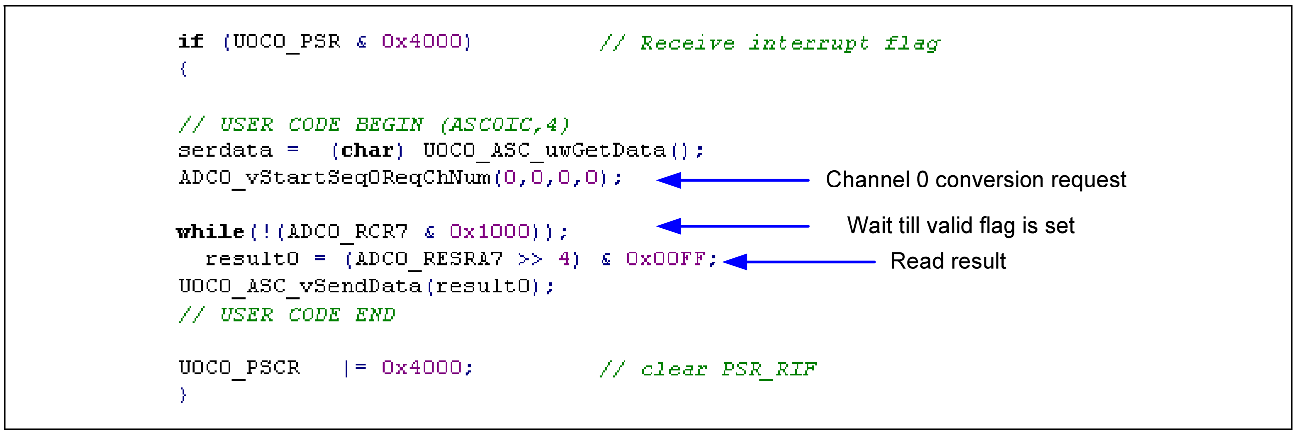

VFR
Valid Flag Register
XSFR(80H)
Reset Value: 0000H
0
VF7 VF6 VF5 VF4 VF3 VF2 VF1 VF0
r rwh
Result stored in result register
1
1 - Valid data
0 - Invalid data
Another view of the valid flag bit in VFR register
RCRx (x = 0 - 7)
Result Control Register x
XSFR(B0H + x * 2)
Reset Value: 0000H
0 VF 0 DRC 0 WFR FEN IEN 0 DRCTR
r rh r rh r rw rw rw r rw

Figure 4 Data valid flag of result register 7

Another view of the valid flag bit in VFR register is from VF bit in RCR7 register. Only read operation is possible in RCR7 register whereas in VFR register both read and write is possible.

In VF7bit of VFR register, writing a 0 has no effect, whereas writing a 1 clears the written bit position. If a hardware event triggers the setting of a bit VF7 and SW writes 1 to the same bit position, the bit VF7 is cleared (software overrules hardware).

An example program ADC_Storage_result0 is given with the application note. On UART receive interrupt the ADC conversion is requested and the result is read.



```
if  (U0C0_PSR & 0x4000)          // Receive interrupt flag
{

// USER CODE BEGIN (ASC0IC,4)
serdata =   (char) U0C0_ASC_uwGetData();
ADC0_vStartSeq0ReqChNum(0,0,0,0);

while(!(ADC0_RCR7 & 0x1000));
  result0 = (ADC0_RESRA7 >> 4)  & 0x00FF;
U0C0_ASC_vSendData(result0);
// USER CODE END

U0C0_PSCR   |= 0x4000;           // clear PSR_RIF
}
```

Figure 5 Result data valid flag checking

In the above example, channel 0 conversion is requested first. Then polling requested until the result valid flag is set (bitfield VF) in the result register. Finally the result (8bit) is read from the result register and then transmitted using U0C0_ASC_vSendData function.

Few things should be taken care with respect to the conversion start and result reading.

Check the data valid flag of the result register before reading the result. Also make sure whether correct data has been read for the conversion requested through a particular channel (incase same result register shared by two channels). The channel number corresponding to the latest result data update can be read from the result register and is shown in Figure 6.

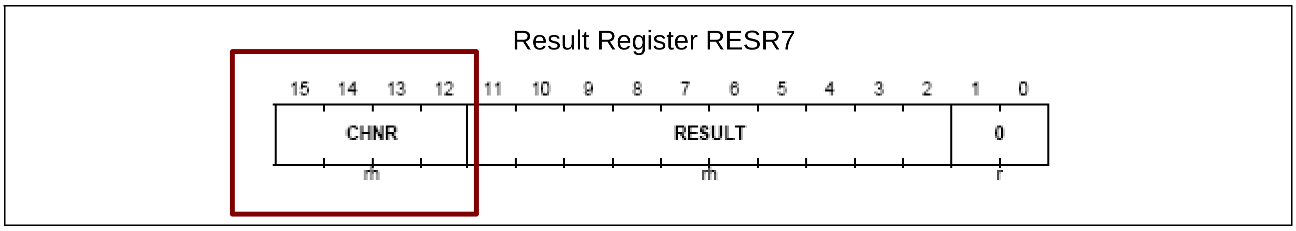


Figure 6 Channel number bitfield of result register for the latest data

2.1 Representation of conversion data

The application requirements for results with enabled or disabled data reduction filter being different and debugger accesses can occur, four different scenarios with different result register read views are supported. The four read views refer to the same result register contents, but show a different behavior according to the address that has been read.

2.1.1.1 Standard read view RESRx (x = 0-7)

The data reduction filter has to be disabled, the channel number is included to identify which channel has been converted, and a read action clears the corresponding valid bit. This representation is compatible to the ADC result register in XC16x devices.

2.1.1.2 Read view RESRAx (x = 0-7)

The data reduction filter can be enabled, the channel number is not included, and a read action clears the corresponding valid bit.

2.1.1.3 Read view RESRVx (x = 0-7) for debugger

The data reduction filter has to be disabled, the channel number is included, but a read action does not clear the corresponding valid bit.

2.1.1.4 Read view RESRAVx (x = 0-7) for debugger

The data reduction filter can be enabled, the channel number is not included, but a read action does not clear the corresponding valid bit.

Due to different result handling mechanisms, the conversion data can be represented in different ways:

2.1.2 Data reduction filter disabled

The conversion result is maximum 10 bits wide with the MSB of the conversion result being always at bit position 11 and the remaining LSBs filled with 0. For a 10bit conversion the valid data is available at bit field 11:2 and for an 8bit conversion the valid data is available at bitfield 11:4. This is shown in Figure 7. The result data can be read from the RESR7 register or from the RESRA7 register.

2.1.3 Data reduction filter enabled

The conversion result is maximum 10 bits wide with the MSB of the conversion result being always at bit position 11 and the remaining LSBs filled with 0. The additional bits [13:12] show the MSBs of the data accumulation. If the data reduction filter is enabled then the result should be read from the RESRA7 register and is shown in Figure 7.

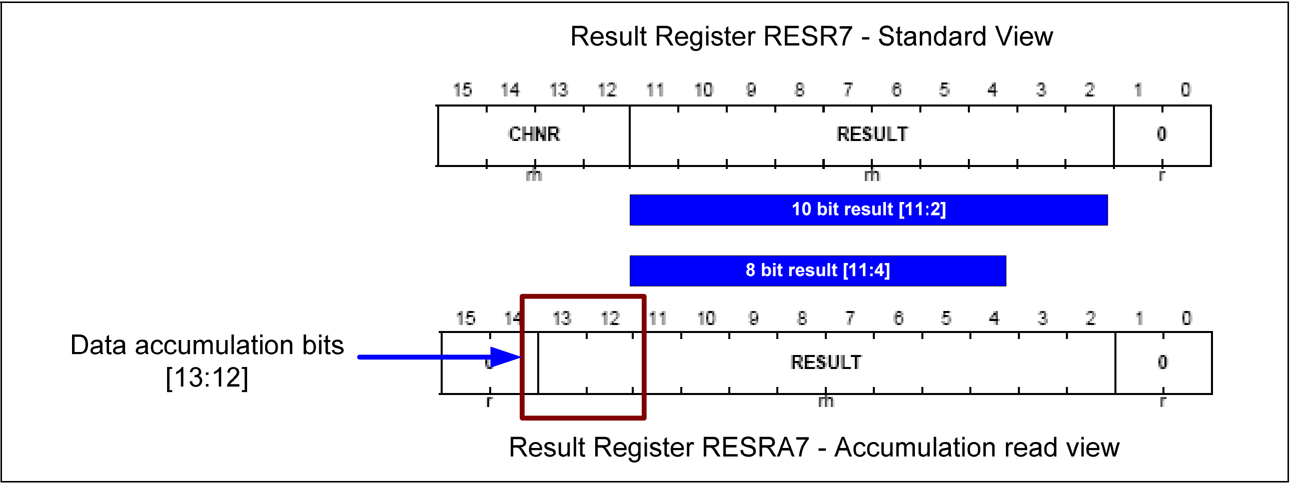


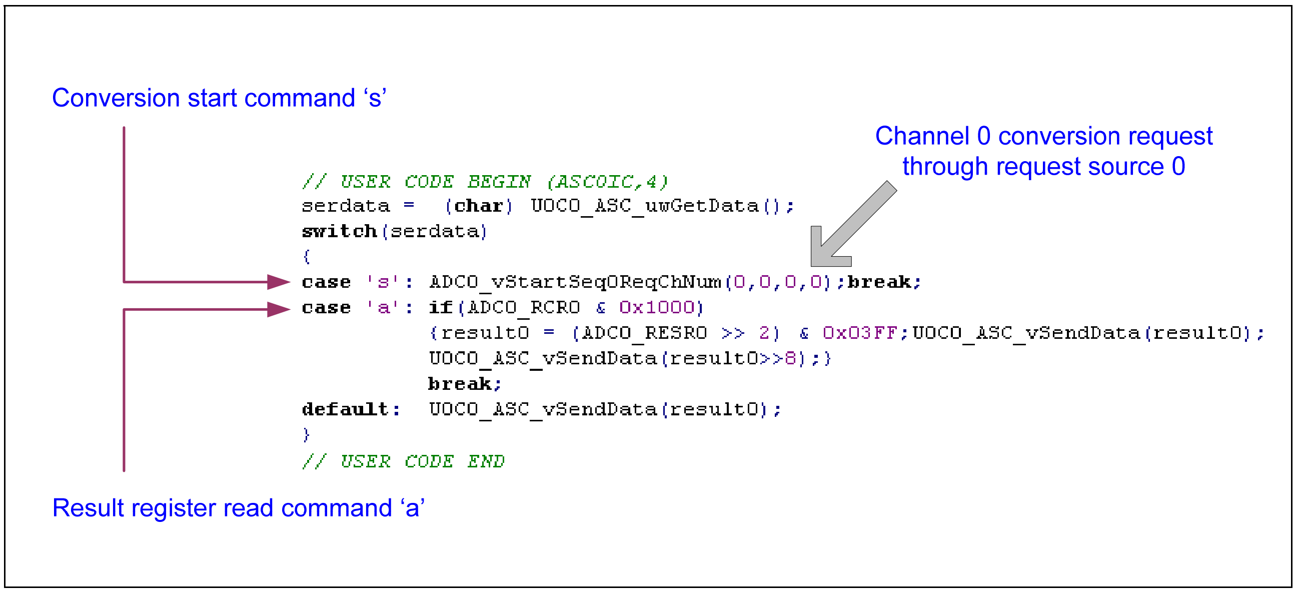
Figure 7 Result bit field for 8 and 10 bit result data

3 Wait-for-read mode

The wait-for-read mode is a feature of a result register allowing the CPU (or PEC) to treat each conversion result independently without the risk of data loss. Data loss could occur if the CPU does not read a conversion result from a result register before a new result overwrites the previous one.

If wait-for-read mode is enabled for a result register by setting bit WFR in register **RCRx (x = 0 - 7)**, a request source does not generate a conversion request while the targeted result register contains valid data (indicated by the valid flag VFx = 1) or if a currently running conversion targets the same result register.

Refer the example program ADC_result0_WFR given with this application note.

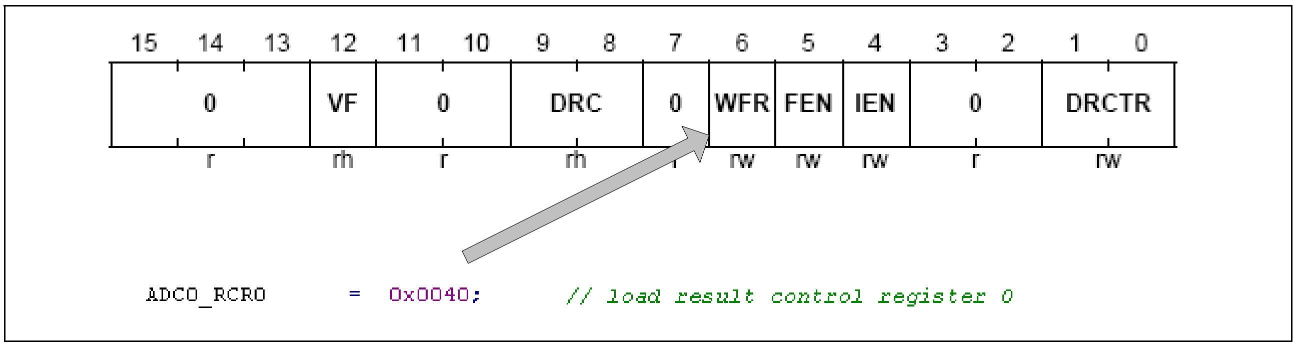

Conversion start command ‘s’
Channel 0 conversion request through request source 0
Result register read command ‘a’

```
// USER CODE BEGIN (ASC0IC,4)
serdata =  (char) U0C0_ASC_uwGetData();
switch(serdata)
{
case 's': ADC0_vStartSeq0ReqChNum(0,0,0,0);break;
case 'a': if(ADC0_RCR0 & 0x1000)
          {result0 = (ADC0_RESR0 >> 2) & 0x03FF;U0C0_ASC_vSendData(result0);
          U0C0_ASC_vSendData(result0>>8);}
          break;
default:  U0C0_ASC_vSendData(result0);
}
// USER CODE END
```

Figure 8 Conversion start and result read example

The conversion start and read operations can be performed by sending the commands ‘s’ and ‘a’ through UART. When you send ‘s’ through Docklight, a conversion start of channel 0 will be initiated through request source 0. When you send ‘a’ through Docklight the result data will be transmitted back to Docklight using result0 variable.

Also see the result register 0 WFR (wait for read) bit enable option in the Figure 9.



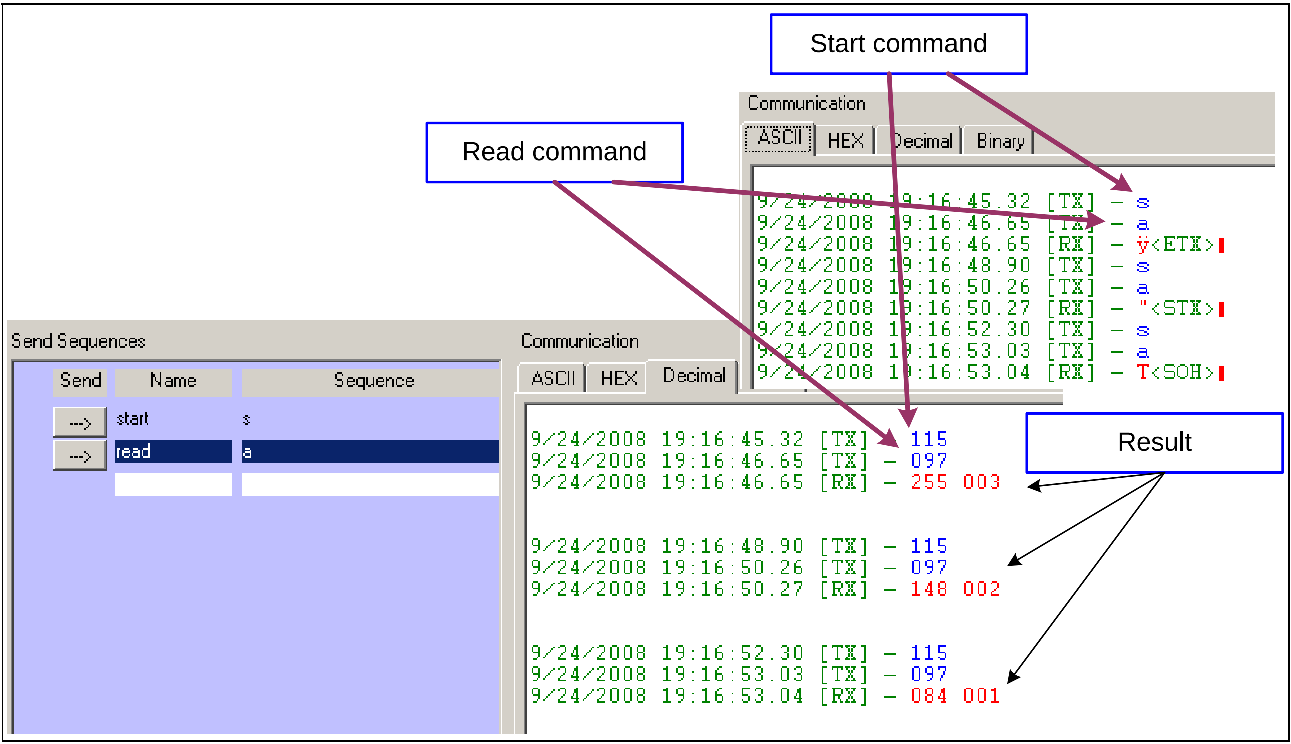
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			VF	0		DRC		0	WFR	FEN	IEN	0		DRCTR	
r			rh	r		rh		r	rw	rw	rw	r		rw	

```
ADC0_RCR0        =  0x0040;        // load result control register 0
```

Figure 9 Result register 0 wait-for-read mode enable setting

For this example, two send sequences are configured in Docklight. A start sequence which will send 's' to the target controller and a read sequence which will send 'a' to the target controller. Refer Figure 8 for start conversion and read result mechanism in the UART interrupt service routine.

Refer Figure 10 for the detailed configuration in Docklight for communication.



Send	Name	Sequence
--->	start	s
--->	read	a

```
9/24/2008 19:16:45.32 [TX] - s
9/24/2008 19:16:46.65 [TX] - a
9/24/2008 19:16:46.65 [RX] - ÿ<ETX>
9/24/2008 19:16:48.90 [TX] - s
9/24/2008 19:16:50.26 [TX] - a
9/24/2008 19:16:50.27 [RX] - "<STX>
9/24/2008 19:16:52.30 [TX] - s
9/24/2008 19:16:53.03 [TX] - a
9/24/2008 19:16:53.04 [RX] - T<SOH>
```

```
9/24/2008 19:16:45.32 [TX] - 115
9/24/2008 19:16:46.65 [TX] - 097
9/24/2008 19:16:46.65 [RX] - 255 003


9/24/2008 19:16:48.90 [TX] - 115
9/24/2008 19:16:50.26 [TX] - 097
9/24/2008 19:16:50.27 [RX] - 148 002


9/24/2008 19:16:52.30 [TX] - 115
9/24/2008 19:16:53.03 [TX] - 097
9/24/2008 19:16:53.04 [RX] - 084 001
```

Figure 10 Docklight communication with start conversion and read result commands

3.1 WFR test scenario 1

Step 1: Keep the potentiometer in the maximum analog input voltage position.

Step 2: Start a conversion by sending the start sequence from Docklight.

Step 3: Read the result.

Step 4: Tune the potentiometer to change the analog input voltage.

Step 5: Start a conversion by sending the start sequence from Docklight.

Step 6: Read the result.

You will get the exact result for the analog input given at the channel 0. As the result of the conversion requested in Step 1 is read in Step 3, the conversion request in Step 5 will be handled and the result will be stored in the result register.

3.2 WFR test scenario 2

Step 1: Keep the potentiometer in the maximum analog input voltage position.

Step 2: Start a conversion by sending the start sequence from Docklight.

Step 3: Tune the potentiometer to change the analog input voltage.

Step 4: Start a conversion by sending the start sequence from Docklight.

Step 5: Read the result.

Step 6: Read the result.

Though the conversion is requested in Step 4, this conversion request will be pending as the result for the Step 2 conversion request was not read. When you request the result in Step 5, you will get the result of the Step 2 conversion request (which is the maximum 10bit value). And now the pending conversion request of Step 4 will be handled and result will be stored in result register. In Step 6, the result for the Step 4 conversion request will be read from the result register.

The triggering mechanism and result reading should be handled carefully. Say if there is a Step 4a (between Step 4 and 5), in which the potentiometer is kept at its minimum analog input position. For the conversion requested in Step 4, we will be getting the result as 0x0000 (result of Step 6 and potentiometer position as in Step4a instead of the desired data as in Step 3). This is because the conversion request was pending and handled after the result was read. This is explained in Test scenario 3.

3.3 WFR test scenario 3

Step 1: Keep the potentiometer in the maximum analog input voltage position.

Step 2: Start a conversion by sending the start sequence from Docklight.

Step 3: Tune the potentiometer to the minimum analog input voltage position.

Step 4: Start a conversion by sending the start sequence from Docklight.

Step 5: Tune the potentiometer in between the min and max position.

Step 6: Read the result.

Step 7: Read the result.

In Step 1, the potentiometer is kept in its maximum analog input position and a conversion is requested in Step2. Now this conversion request will be handled and the result of this conversion will be stored in result register 0. In Step 3, the potentiometer is kept in its minimum analog input position and a conversion is requested in Step 4. Now as the result of the previous conversion is not read from the result register, the conversion will be pending until the result was read. Now remember, the conversion is requested in Step 4, when the input analog voltage is in minimum (zero position). In Step 5, the analog input is changed to a value in between the minimum and maximum position. In Step 6, the result was read from the result register. The result for the Step 2 conversion and for the maximum analog input voltage will be displayed. At this time the pending conversion request of Step 4 will be handled and now the analog input voltage is between the minimum and maximum. The result of this conversion will be stored in the result register and this result was read in Step 6. But the conversion is requested when the analog input voltage value is zero.

3.4 Important things to remember for wait-for-read mode

When writing complex algorithms the programmer should be careful when to start a conversion and read the conversion result.

If two request sources target the same result register with wait-for-read selected, a lower priority request started before the higher priority source has requested its conversion can not be interrupted by the higher priority request. If a higher priority request targets a different result register, the lower priority conversion can be cancelled and repeated afterwards.

4 Result FIFO buffer

The result FIFO buffer mechanism allows to store measurement results and to read them out later with a "relaxed" CPU access timing. It is possible to set up more than one FIFO buffer structure with the available result registers.

A FIFO structure can be built by at least two "neighbor" result registers with the indices x and z = x+1, where result register z represents the input and result register x represents the output of the FIFO buffer. If more than two result neighbor registers are concatenated to a FIFO buffer (from result register z to result register x, with z > x), the one with the highest index (z) is always the input and the one with the lowest index (x) is always the output. All intermediate result registers y (x < y < z) are used as intermediate FIFO stages without data input or data output functionality.

Features of the FIFO buffer

1. Wait-for-read mode and Data reduction is allowed and can be enabled only for input stage.
2. Result event interrupt generation is allowed and can be enabled only for the output stage.
3. Both the input and intermediate buffer stage must not be read at a read view modifying the valid bit.

The FIFO buffer function is enabled by setting bit FEN in result control registers RCRx (x=0-7).

Point to remember:

The FEN bitfield of the input stage of the result register should be set with '0'. The functionality remains the same for a single FIFO buffer configuration even if the FEN bitfield of the input stage of the result register is set with '1'. But with multiple FIFO configurations this may cause problems. Hence it is always recommended to set the FEN bitfield of the input stage with '0'. This is explained in section 4.2.

4.1 Single FIFO structure

In the example provided with this application note (ADC0_Result_FIFO), result registers 3…0 are concatenated to form a FIFO buffer. Here the result register 3 is the input stage and the result register 0 is the output stage of the FIFO buffer. Result registers 1 and 2 will be treated as the intermediate stages.

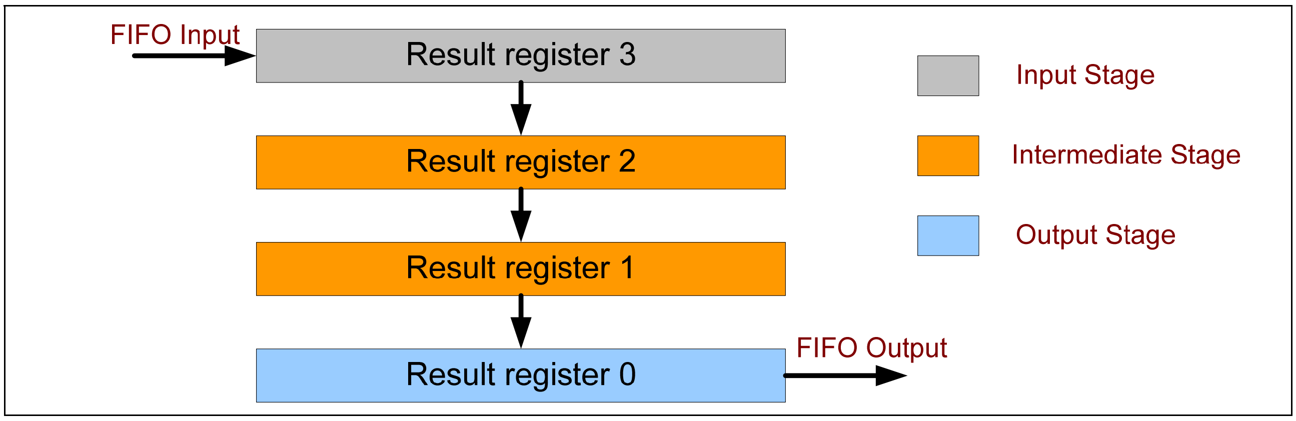
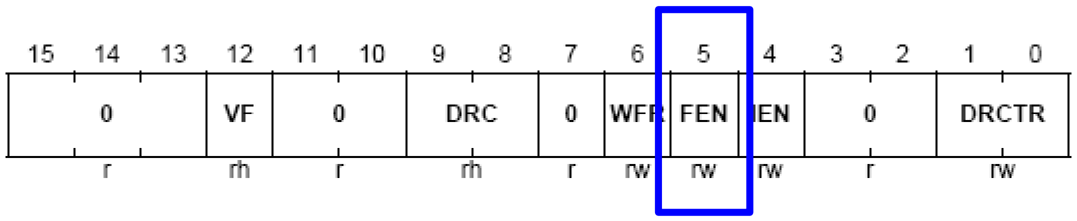


Figure 11 FIFO buffer formation with result registers

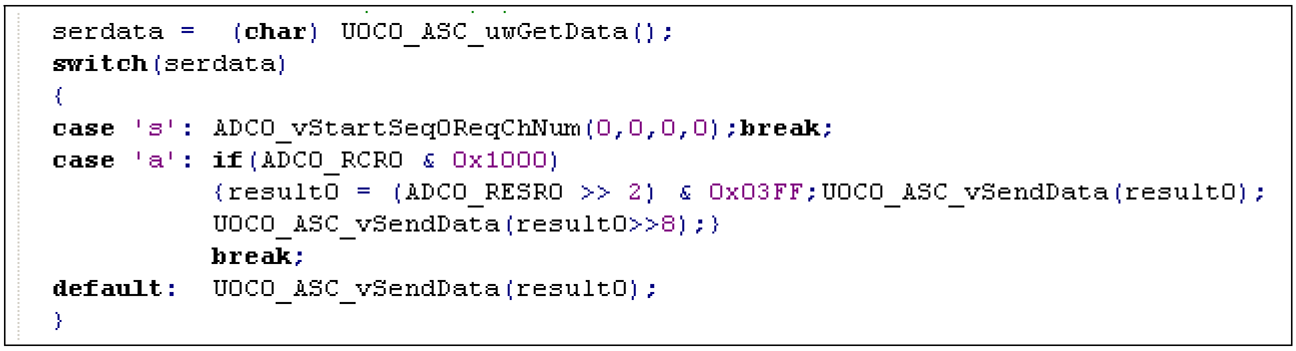
The FIFO buffer function is enabled by setting bit FEN in result control registers RCRx (x=0-2). The software settings are shown in Figure 12. In the DAvE setting the FIFO functionality is not enabled whereas the registers ADC0_RCRx (x=0, 2) is configured manually as shown in Figure 12.



```
ADC0_RCR0     =  0x0020;
ADC0_RCR1     =  0x0020;
ADC0_RCR2     =  0x0020;
ADC0_RCR3     =  0x0000;         //  (FEN bitfield is set with ‘0’ for the input stage)
```

Figure 12 FIFO enable configuration in result control registers RCR3…RCR0

In this example, two send sequences (start conversion ‘s’ and read result ‘a’) are configured in Docklight. Refer Figure 13 for start conversion and read result mechanism in the UART interrupt service routine.



```
serdata =  (char) U0C0_ASC_uwGetData();
switch(serdata)
{
case 's': ADC0_vStartSeq0ReqChNum(0,0,0,0);break;
case 'a': if(ADC0_RCR0 & 0x1000)
          {result0 = (ADC0_RESR0 >> 2) & 0x03FF;U0C0_ASC_vSendData(result0);
          U0C0_ASC_vSendData(result0>>8);}
          break;
default:  U0C0_ASC_vSendData(result0);
}
```

Figure 13 FIFO buffer example (start conversion and read result functionality in ISR)

4.1.1 FIFO buffer test scenario 1

Step 1: Keep the potentiometer in the maximum analog input voltage position.

Step 2: Start a conversion by sending the start sequence from Docklight.

Step 3: Keep the potentiometer within min and max analog input voltage position (close to maximum).

Step 4: Start a conversion by sending the start sequence from Docklight.

Step 5: Keep the potentiometer within min and max analog input voltage position (close to minimum).

Step 6: Start a conversion by sending the start sequence from Docklight.

Step 7: Keep the potentiometer in the minimum analog input voltage position.

Step 8: Start a conversion by sending the start sequence from Docklight.

Step 9: Read the result.

Step 10: Read the result.

Step 11: Read the result.

Step 12: Read the result.

You will get the result in sequence for the conversions requested in Step2, Step 4, Step 6 and Step 8.

4.1.2 FIFO buffer test scenario 2

Execute the Steps upto 8 as in FIFO buffer Test Scenario 1.

Step 9: Keep the potentiometer in maximum analog input voltage position.

Step 10: Start a conversion by sending the start sequence from Docklight.

Step 11: Read the result.

Step 12: Read the result.

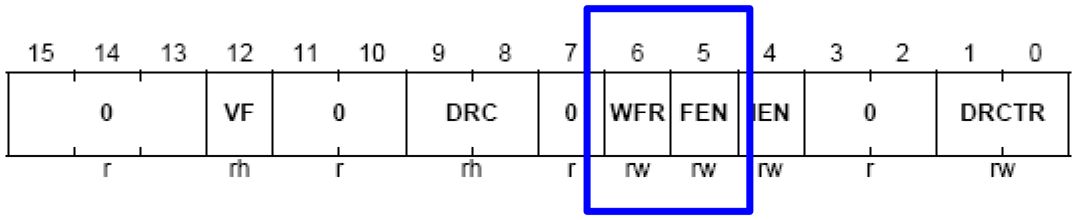
Step 13: Read the result.

Step 14: Read the result.

You will get the results for Step 11, 12 and 13 for the conversions requested in Step 2, 4 and 6. But for the Step 14 the result of conversion requested in Step 10 will be shown. The result of the conversion requested in Step 8 is overwritten by the conversion requested in Step 10. Hence if the FIFO buffer is full and the input result register is not configured for wait-for-read mode then any further conversions will overwrite the input register again and again. The contents of result register in intermediate and output stage will not be affected.

4.1.3 FIFO buffer test scenario 3

In the above example, only the result register 3 (input register) configurations are changed for wait-for-read result. Refer ADC0_Result_FIFO_WFR example. In the DAvE setting the FIFO functionality is not enabled whereas the registers ADC0_RCRx (x=0, 2) is configured manually as shown in Figure 14.



```
ADC0_RCR0    = 0x0020;
ADC0_RCR1    = 0x0020;
ADC0_RCR2    = 0x0020;
ADC0_RCR3    = 0x0040;    // (FEN bitfield is set with ‘0’ for the input stage)
```

Figure 14 FIFO buffer input register configuration for wait-for-read mode

Execute the FIFO buffer test scenario 2.

You will get the results for Step 11, 12, 13 and 14 for the conversions requested in Step 2, 4, 6 and 8. But the conversion requested in Step 10 will be pending as the FIFO is full and the result is not read. The pending conversion will be started again after the result is read in Step 11.

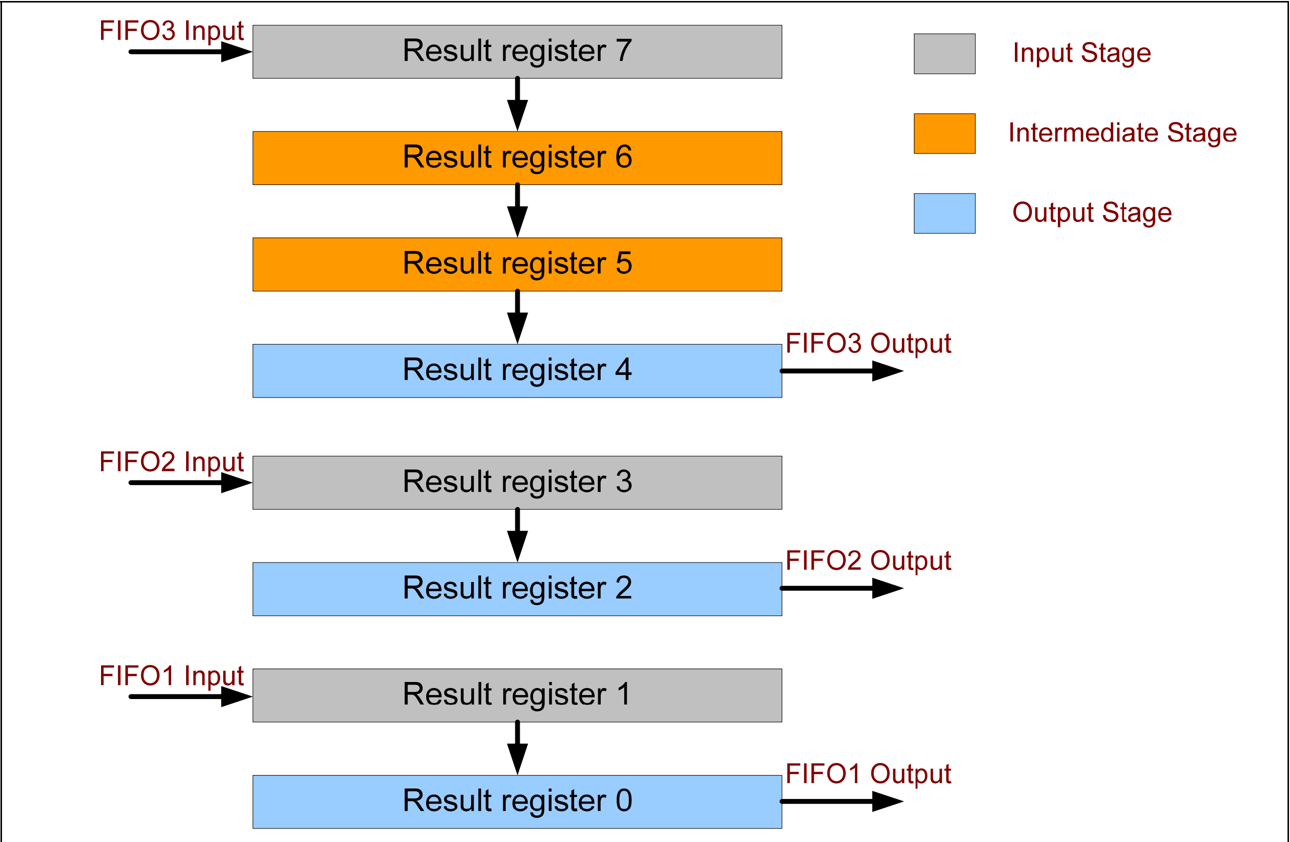
As wait-for-read mode is selected, refer section 3.4 for the important points to be taken into account.

4.1.3.1 FIFO buffer test scenario 4

Now execute all the previous three test scenarios after setting the FEN bitfield of the input stage of the result register (RCR3 register) with ‘1’. The results remain the same. But this will cause problem with multiple FIFO structure and is explained in section 4.2.

4.2 Multiple FIFO structure

Here three FIFO structures, FIFO1 [Result registers 1 to 0], FIFO2 [Result registers 3 to 2] and FIFO3 [result registers 7 to 4 are formed. The FIFO structure and the stages are shown in Figure 15.



```
ADC0_RCR0    = 0x0020;
ADC0_RCR1    = 0x0000;    // (FEN bitfield is set with ‘0’ for the input stage of FIFO1)
ADC0_RCR2    = 0x0020;
ADC0_RCR3    = 0x0000;    // (FEN bitfield is set with ‘0’ for the input stage of FIFO2)
ADC0_RCR4    = 0x0020;
ADC0_RCR5    = 0x0020;
ADC0_RCR6    = 0x0020;
ADC0_RCR7    = 0x0000;    // (FEN bitfield is set with ‘0’ for the input stage of FIFO3)
```

Figure 15 Multiple FIFO structure

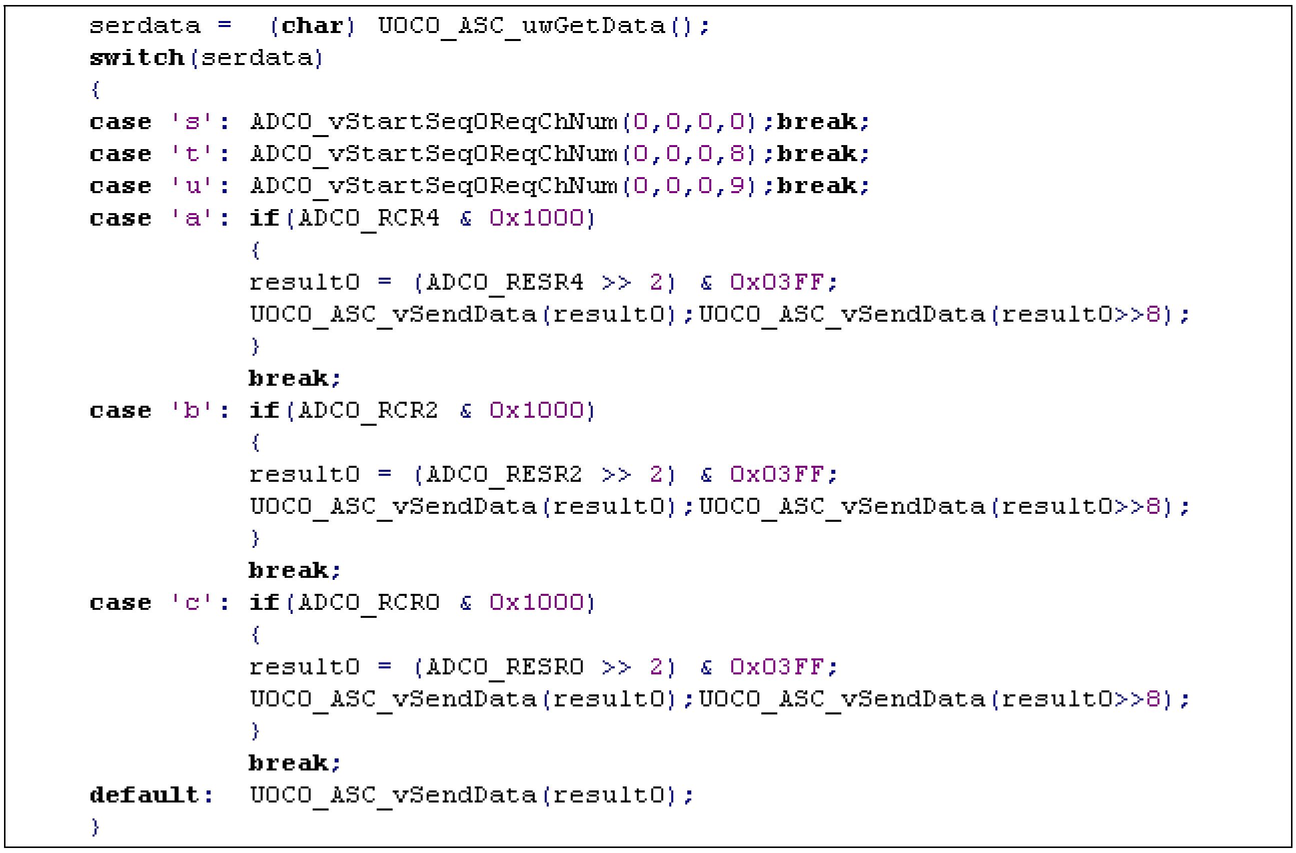
The FEN bitfield of output and intermediate stages are set with ‘1’ whereas for the input stage FEN bitfield is set with ‘0’. For example if the FEN bitfield of RCR3 and RCR1 is set with ‘1’ and the FIFO3 structure will be formed with result registers 7 to 0 and the result data will be stored from result register 0. But the user will be expecting data from the result register 4. Similarly for the FIFO2 the FIFO structure will be formed with result registers 3 to 0. By setting the FEN bitfield with ‘0’ for the input stage, FIFO buffer becomes well defined in case of multiple FIFO structure.

Refer the application example ADC0_Result_Multiple_FIFO.

Channel 0, Channel 8 and Channel 9 are configured with output register 7, 3 and 1 respectively.

In the DAvE setting the FIFO functionality is not enabled whereas the registers ADC0_RCRx (x=0, 7) is configured manually as shown in Figure 15.

Refer Figure 16 for implementation of conversion start of channels 0, 8 and 9 and result reading mechanism from output stages of FIFO structures in the UART interrupt service routine.



```
serdata =  (char) U0C0_ASC_uwGetData();
switch(serdata)
{
case 's': ADC0_vStartSeq0ReqChNum(0,0,0,0);break;
case 't': ADC0_vStartSeq0ReqChNum(0,0,0,8);break;
case 'u': ADC0_vStartSeq0ReqChNum(0,0,0,9);break;
case 'a': if(ADC0_RCR4 & 0x1000)
          {
          result0 = (ADC0_RESR4 >> 2) & 0x03FF;
          U0C0_ASC_vSendData(result0);U0C0_ASC_vSendData(result0>>8);
          }
          break;
case 'b': if(ADC0_RCR2 & 0x1000)
          {
          result0 = (ADC0_RESR2 >> 2) & 0x03FF;
          U0C0_ASC_vSendData(result0);U0C0_ASC_vSendData(result0>>8);
          }
          break;
case 'c': if(ADC0_RCR0 & 0x1000)
          {
          result0 = (ADC0_RESR0 >> 2) & 0x03FF;
          U0C0_ASC_vSendData(result0);U0C0_ASC_vSendData(result0>>8);
          }
          break;
default:  U0C0_ASC_vSendData(result0);
}
```

Figure 16 Multiple FIFO channel conversion and result reading

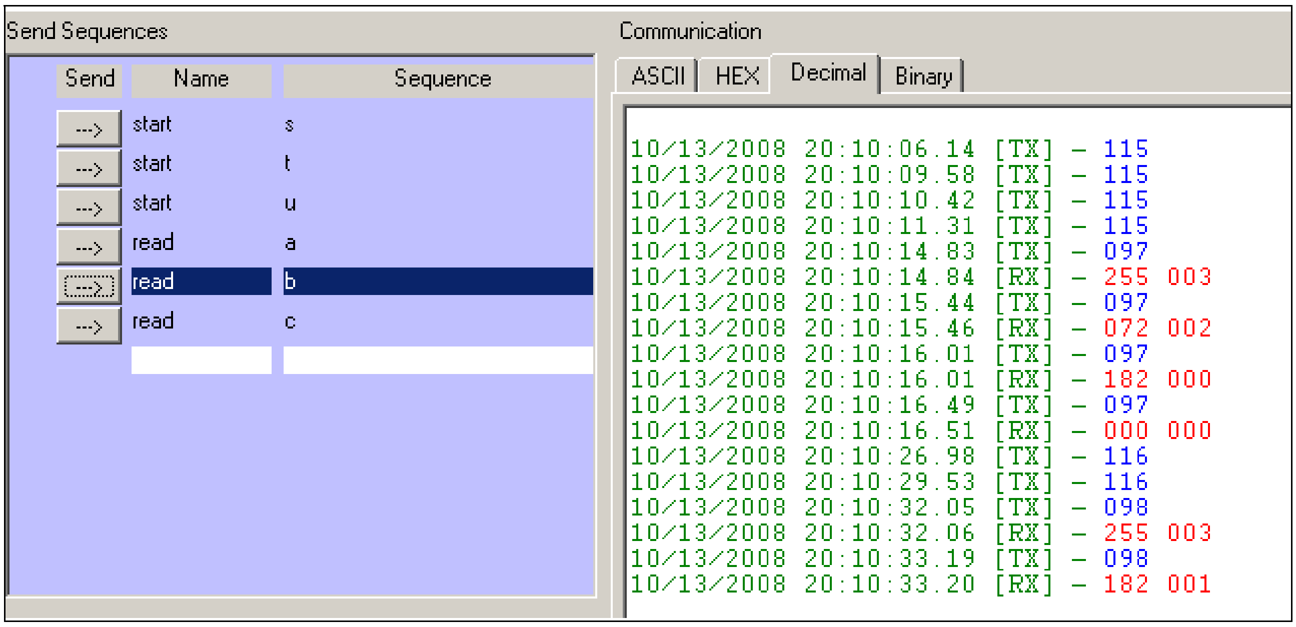
In this example, three conversion request sequences and three result reading sequences are configured in Docklight.

Conversion requests:

‘s’ for channel 0, ‘t’ for channel 8 and ‘u’ for channel 9

Results reading:

‘a’ for FIFO3 (result register 4), ‘b’ for FIFO2 (result register 2) and ‘c’ for FIFO1 (result register 0).



Send	Name	Sequence
--->	start	s
--->	start	t
--->	start	u
--->	read	a
--->	read	b
--->	read	c

```
10/13/2008 20:10:06.14 [TX] - 115
10/13/2008 20:10:09.58 [TX] - 115
10/13/2008 20:10:10.42 [TX] - 115
10/13/2008 20:10:11.31 [TX] - 115
10/13/2008 20:10:14.83 [TX] - 097
10/13/2008 20:10:14.84 [RX] - 255 003
10/13/2008 20:10:15.44 [TX] - 097
10/13/2008 20:10:15.46 [RX] - 072 002
10/13/2008 20:10:16.01 [TX] - 097
10/13/2008 20:10:16.01 [RX] - 182 000
10/13/2008 20:10:16.49 [TX] - 097
10/13/2008 20:10:16.51 [RX] - 000 000
10/13/2008 20:10:26.98 [TX] - 116
10/13/2008 20:10:29.53 [TX] - 116
10/13/2008 20:10:32.05 [TX] - 098
10/13/2008 20:10:32.06 [RX] - 255 003
10/13/2008 20:10:33.19 [TX] - 098
10/13/2008 20:10:33.20 [RX] - 182 001
```

Figure 17 Conversion start and result read with multiple FIFO mechanism in Docklight

The user can set the FEN bitfield of input stages of FIFO1, FIFO2 and FIFO3 with ‘1’ and test the same. The user will see things happening as explained in this section earlier.

5 Data reduction filter

The data reduction filter can be used as digital filter for anti-aliasing or decimation purposes. It can accumulate a maximum of 4 conversion results to generate a final result. Each result register can be individually enabled for data reduction.

The feature is controlled by bit field DRCTR in registers **RCRx (x = 0 - 7)**. The actual status is given by bit field DRC (data reduction counter) in the same register.

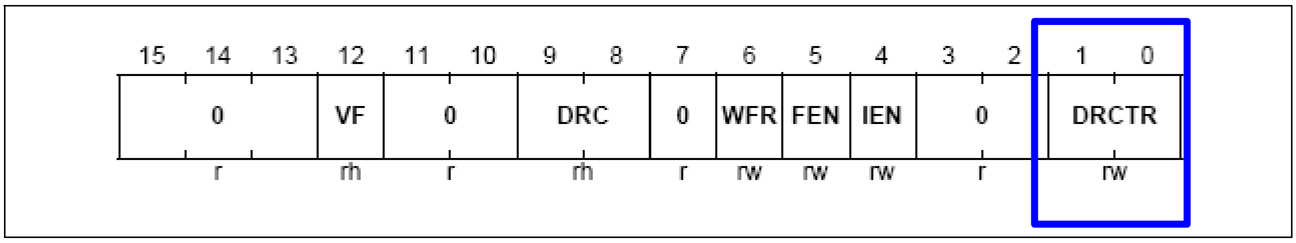


Figure 18 Data reduction enable setting of a result register in DRCTR bifield of RCRx register

In the example provided with this application note (ADC0_Data_Reduction_Filter), data reduction filter is enabled for result register 0. Also the reload value for DRC is selected as 3.

The result control register 0 is configured as

ADC0_RCR0 = 0x0003;

In this example, two send sequences (start conversion 's' and read result 'a') are configured in Docklight. Refer Figure 19 for start conversion and read result mechanism in the UART interrupt service routine. The result is read from the data reduction read view register RESRA0.

```
serdata =  (char) U0C0_ASC_uwGetData();
switch(serdata)
{
case 's': ADC0_vStartSeq0ReqChNum(0,0,0,0);break;
case 'a': if(ADC0_RCR0 & 0x1000)
          {result0 = (ADC0_RESRA0 >> 2) & 0x0FFF;U0C0_ASC_vSendData(result0);
          U0C0_ASC_vSendData(result0>>8);}
          break;
default:  U0C0_ASC_vSendData(result0);
}
```

Figure 19 Data reduction filter example (Conversion start and result read functionality in UART ISR)

5.1 Data reduction test scenario 1

Step 1: Keep the potentiometer in the maximum analog input voltage position.

Step 2: Start a conversion by sending the start sequence from Docklight.

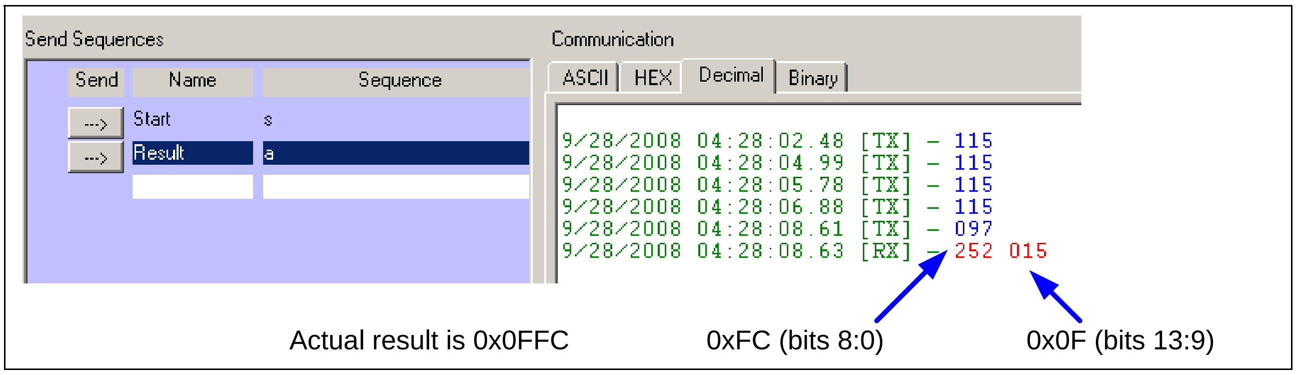
Step 3: Start a conversion by sending the start sequence from Docklight.

Step 4: Start a conversion by sending the start sequence from Docklight.

Step 5: Start a conversion by sending the start sequence from Docklight.

Step 6: Read the result.

The accumulated result will be displayed in the Docklight. As the potentiometer is adjusted to give the maximum input voltage, the 10 bit result will be maximum (1023). The conversion is requested four times and the accumulated value should be 4092 (4*1023). The hexadecimal equivalent of this is 0x0FFC. This is shown in Figure 20.



```
9/28/2008 04:28:02.48 [TX] - 115
9/28/2008 04:28:04.99 [TX] - 115
9/28/2008 04:28:05.78 [TX] - 115
9/28/2008 04:28:06.88 [TX] - 115
9/28/2008 04:28:08.61 [TX] - 097
9/28/2008 04:28:08.63 [RX] - 252 015
```

Figure 20 Data reduction accumulated result in Docklight

Without reading the result if the conversion requested more than the reload value count (here accumulation of 4), then the counter will reset and the data accumulation will start again overwriting the previous accumulated result (data loss).

5.2 Data reduction test scenario 2

All the configurations remain same except the wait-for-read mode is enabled for the result register 0. Refer ADC0_Data_Reduction_Filter_WFR example.

Step 1: Keep the potentiometer in the maximum analog input voltage position.

Step 2: Start a conversion by sending the start sequence from Docklight.

Step 3: Start a conversion by sending the start sequence from Docklight.

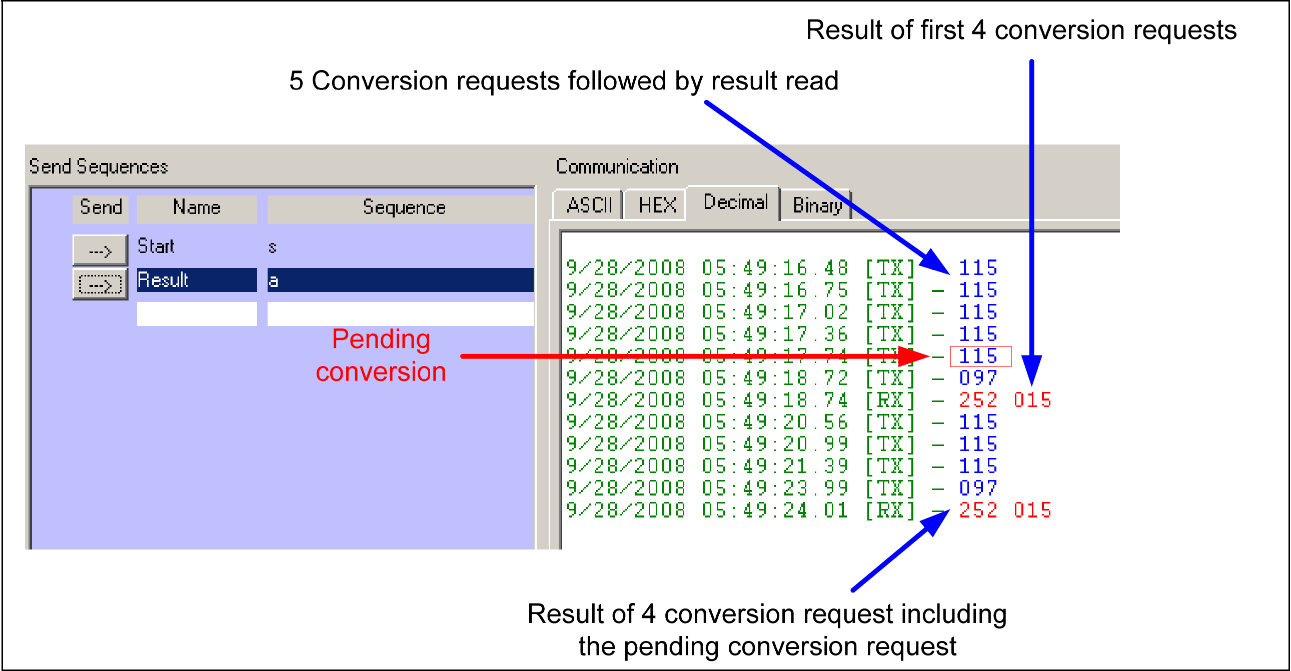
Step 4: Start a conversion by sending the start sequence from Docklight.

Step 5: Start a conversion by sending the start sequence from Docklight.

Step 6: Start a conversion by sending the start sequence from Docklight.

Step 7: Read the result.

Upto Step 5 the conversion will happen and the result will be accumulated in result register 0. The conversion requested by Step 6 will be pending as the result is not read. In Step 7, the result will be read from the result register and pending conversion requested by Step 5 will be handled. This is shown in Figure 21.


Result of first 4 conversion requests
5 Conversion requests followed by result read
Send Sequences
Send
Name
Sequence
-->
Start
s
Result
a
Pending conversion
Communication
ASCII
HEX
Decimal
Binary
Result of 4 conversion request including the pending conversion request

```
9/28/2008 05:49:16.48 [TX] - 115
9/28/2008 05:49:16.75 [TX] - 115
9/28/2008 05:49:17.02 [TX] - 115
9/28/2008 05:49:17.36 [TX] - 115
9/28/2008 05:49:17.74 [TX] - 115
9/28/2008 05:49:18.72 [TX] - 097
9/28/2008 05:49:18.74 [RX] - 252 015
9/28/2008 05:49:20.56 [TX] - 115
9/28/2008 05:49:20.99 [TX] - 115
9/28/2008 05:49:21.39 [TX] - 115
9/28/2008 05:49:23.99 [TX] - 097
9/28/2008 05:49:24.01 [RX] - 252 015
```

Figure 21 Data reduction with wait-for-read mode enabled for result register 0

As wait-for-read mode is selected, refer section 3.4 for the important points to be taken into account.

6 Data reduction filter with result FIFO

The final result of the data reduction sequence has to be read out from result register x (where x is the register configured for data reduction, x = 0-7) before the next data sequence starts. If the interval is too short for result reading and the start of the next sequence, then the FIFO mechanism can be used.

In the example given with this application note (ADC0_DRF_FIFO), the FIFO buffer is formed using result register RCR3…RCR0. Here result register 3 acts as the input register, result registers 2 to 1 acts as the intermediate registers and result register 0 acts as the output register. Also result register 3 (only input register is allowed for data reduction mechanism) is enabled for data reduction filter.

In the DAvE, the FIFO functionality is not enabled whereas the registers ADC0_RCRx (x=0, 2) is configured manually as shown in Figure 22.

```
ADC0_RCR0    = 0x0020;
ADC0_RCR1    = 0x0020;
ADC0_RCR2    = 0x0020;
ADC0_RCR3    = 0x0003;       // (FEN bitfield is set with ‘0’ for the input stage)
```

Figure 22 Result control register RCR3…RCR0 configuration for data reduction with result FIFO

In this example, two send sequences (start conversion ‘s’ and read result ‘a’) are configured in Docklight. Refer Figure 23 for start conversion and read result mechanism in the UART interrupt service routine. The result is read from the data reduction read view register RESRA0.

```
serdata =  (char) U0C0_ASC_uwGetData();
switch(serdata)
{
case 's': ADC0_vStartSeq0ReqChNum(0,0,0,0);break;
case 'a': if(ADC0_RCR0 & 0x1000)
          {result0 = (ADC0_RESRA0 >> 2) & 0x0FFF;U0C0_ASC_vSendData(result0);
          U0C0_ASC_vSendData(result0>>8);}
          break;
default:  U0C0_ASC_vSendData(result0);
}
```

Figure 23 Data reduction filter with result FIFO conversion start and result read in UART ISR

6.1 Data reduction filter with result FIFO test scenario 1

Step 1: Keep the potentiometer in the maximum analog input voltage position.

Step 2: Start the conversion four times by sending the start sequence from Docklight.

Step 3: Keep the potentiometer in the maximum analog input voltage position.

Step 4: Start the conversion four times by sending the start sequence from Docklight.

Step 5: Keep the potentiometer in the minimum analog input voltage position.

Step 6: Start the conversion four times by sending the start sequence from Docklight.

Step 7: Keep the potentiometer in the maximum analog input voltage position.

Step 8: Start the conversion four times by sending the start sequence from Docklight.

Step 9: Read the result.

Step 10: Read the result.

Step 11: Read the result.

Step 12: Read the result.

During Step 2, after each conversion completion, the converted result will be accumulated in RESRA3 register and will be moved to RESRA0 at the end of last conversion (at the end of fourth conversion).

During Step 4, after each conversion completion, the converted result will be accumulated in RESRA3 register and will be moved to RESRA1 at the end of last conversion (at the end of fourth conversion).

During Step 6, after each conversion completion, the converted result will be accumulated in RESRA3 register and will be moved to RESRA2 at the end of last conversion (at the end of fourth conversion).

During Step 8, after each conversion completion, the converted result will be accumulated in RESRA3 register and will be moved to RESRA3 at the end of last conversion (at the end of fourth conversion).

Now the FIFO buffer is full.

When we read the results four times we will get the results as 0x0FFC, 0x0FFC, 0x0000 and 0x0FFC.

6.2 Data reduction filter with result FIFO test scenario 2

Execute the steps upto 8 as described in data reduction filter with result FIFO test scenario 1.

Then execute the following.

Step 9: Keep the potentiometer in the minimum analog input voltage position.

Step 10: Start the conversion four times by sending the start sequence from Docklight.

Step 11: Read the result.

Step 12: Read the result.

Step 13: Read the result.

Step 14: Read the result.

Upto step 8, the functionality remains the same as in the previous test scenario. Now the FIFO is full. But because of the Step 9 and Step 10 actions, the contents of RESRA3 register will be modified (accumulated) for each converted result (accumulation of four result data).

If the user read the result at any stage, the accumulated result data of that instant will be available.

As RESRA3 (input stage) is not configured for wait-for-read mode, the data will be overwritten if the FIFO result data was not read.

When we read the results four times we will get the results as 0x0FFC, 0x0FFC, 0x0000 and 0x0000.

Points to remember:

1. Each converted result will be accumulated (upto four) in RESRA3 register and after accumulation the final result will be moved to lowest empty register.
2. A result read operation will shift the contents of the FIFO in the order RESRA3…RESRA1 to RESRA2…RESRA0.
3. When the FIFO buffer is full and wait-for-read mode is not selected then after each conversion the new result data will be overwritten (accumulated) in the RESRA3 register.
4. When the FIFO buffer is full and wait-for-read mode is selected then the new conversion requests will be pending and will be handled later after a read operation from the FIFO. This is explained in the next section 6.3

6.3 Data reduction filter with result FIFO test scenario 3

Refer ADC0_DRF_FIFO_WFR example.

Now the result register 3 is enabled for wait-for-read mode option. Also in DAvE, the FIFO functionality is not enabled whereas the registers ADC0_RCRx (x=0, 2) is configured manually as shown in Figure 24.

```
    ADC0_RCR0     = 0x0020;
    ADC0_RCR1     = 0x0020;
    ADC0_RCR2     = 0x0020;
    ADC0_RCR3     = 0x0043;
```

Figure 24 Data reduction filter with result FIFO wait-for-read mode

Now execute the same steps as described in data reduction filter with result FIFO test scenario 1.

Then execute the following.

Step 9: Keep the potentiometer in the minimum analog input voltage position.

Step 10: Start the conversion four times by sending the start sequence from Docklight.

Step 11: Read the result.

Step 12: Start the conversion three times by sending the start sequence from Docklight.

Step 13: Read the result.

Step 14: Read the result.

Step 15: Read the result.

Step 16: Read the result.

In Step 10 - as only one queue stage is available with request source 0, one conversion will be pending (as the result is not read) and other conversion requests will be ignored.

In Step 11- the content of RESRA0 is read hence the contents from RESRA3…RESRA1 will be moved to RESRA2…RESRA0. The user will get the result of 0x0FFC for this read operation.

Now the pending conversion request will be handled. For the completion of data accumulation three more completed conversion result is required.

In Step 12 - three more conversions requested and this will be handled and the data will be accumulated in RESRA3 register. Now the FIFO buffer is full.

In Step 13…16 - the user will be getting the following results in sequence 0x0FFC, 0x0000, 0x0FFC and 0x0000.

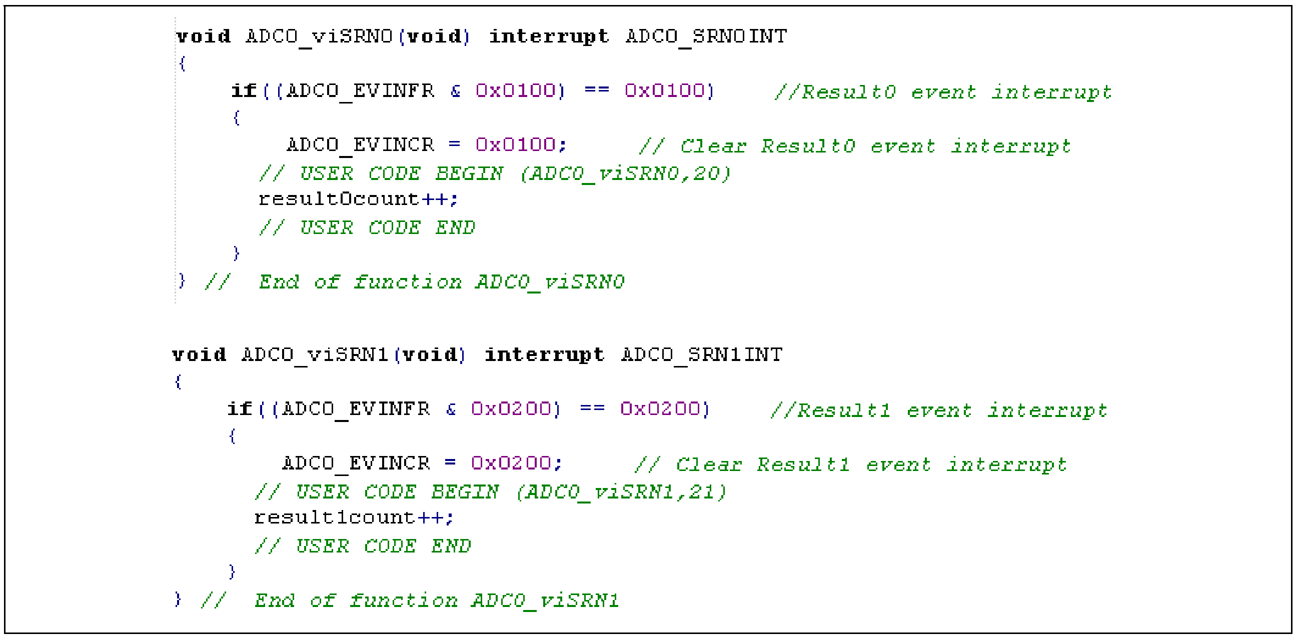
As wait-for-read mode is selected, refer section 3.4 for the important points to be taken into account.

7 Result event interrupt

The bitfield IEN in RCRx enables the result event interrupt if a result event is detected for result register x where x = 0-7. Any of the service request output line SRx (x=0 - 3) can be selected for the result register event.

In the example ADC0_Result_Event, given with this application note, the channel 0 and channel 8 are selected with the result registers 0 and 1 respectively. The result event of channel 0 and channel 8 use the service request output lines SR0 and SR1 respectively.

In the result event interrupt service routines of result register 0 and result register 1 event the variable result0count and result1count incremented by 1 respectively. This is shown in Figure 25.

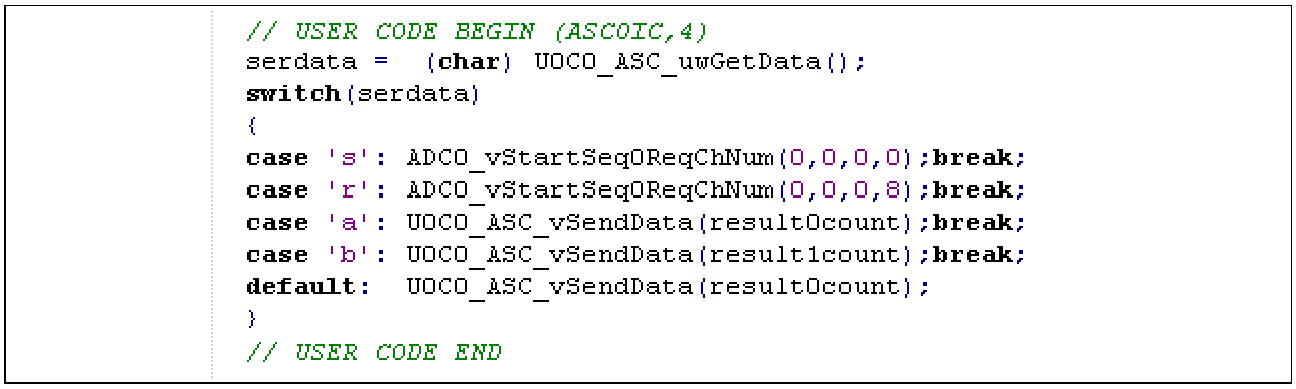


```
void ADC0_viSRN0(void) interrupt ADC0_SRN0INT
{
    if((ADC0_EVINFR & 0x0100) == 0x0100)    //Result0 event interrupt
    {
        ADC0_EVINCR = 0x0100;     // Clear Result0 event interrupt
      // USER CODE BEGIN (ADC0_viSRN0,20)
      result0count++;
      // USER CODE END
    }
} //  End of function ADC0_viSRN0

void ADC0_viSRN1(void) interrupt ADC0_SRN1INT
{
    if((ADC0_EVINFR & 0x0200) == 0x0200)    //Result1 event interrupt
    {
        ADC0_EVINCR = 0x0200;     // Clear Result1 event interrupt
      // USER CODE BEGIN (ADC0_viSRN1,21)
      result1count++;
      // USER CODE END
    }
} //  End of function ADC0_viSRN1
```

Figure 25 Result event interrupt service routine

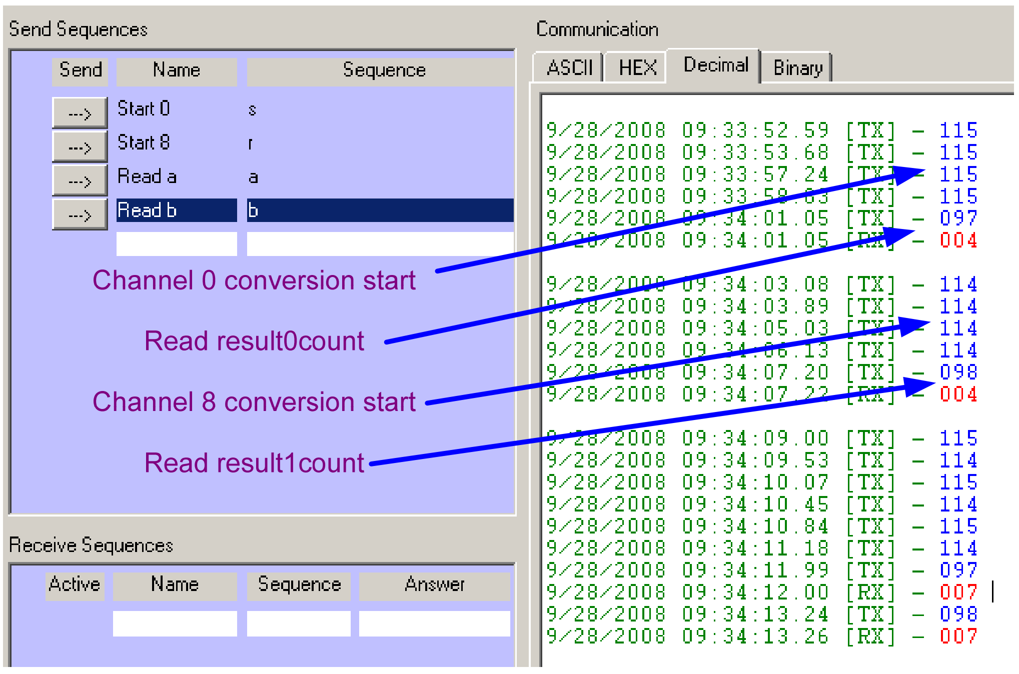
In UART interrupt service routine, the conversion start request of channel 0 and 8 will take place respectively when the UART communication command of ‘s’ and ‘r’ is received. Also the result0count and result1count value will be transmitted when the UART communication command of ‘a’ and ‘b’ is received. This is shown in Figure 26.



```
// USER CODE BEGIN (ASC0IC,4)
serdata =  (char) U0C0_ASC_uwGetData();
switch(serdata)
{
case 's': ADC0_vStartSeq0ReqChNum(0,0,0,0);break;
case 'r': ADC0_vStartSeq0ReqChNum(0,0,0,8);break;
case 'a': U0C0_ASC_vSendData(result0count);break;
case 'b': U0C0_ASC_vSendData(result1count);break;
default:  U0C0_ASC_vSendData(result0count);
}
// USER CODE END
```

Figure 26 UART interrupt service routing for result event interrupt example

The docklight command sequences for conversion start and to read the result event count are shown in Figure 27.



Send	Name	Sequence
--->	Start 0	s
--->	Start 8	r
--->	Read a	a
--->	Read b	b

Active	Name	Sequence	Answer

```
9/28/2008 09:33:52.59 [TX] - 115
9/28/2008 09:33:53.68 [TX] - 115
9/28/2008 09:33:57.24 [TX] - 115
9/28/2008 09:33:58.83 [TX] - 115
9/28/2008 09:34:01.05 [TX] - 097
9/28/2008 09:34:01.05 [RX] - 004

9/28/2008 09:34:03.08 [TX] - 114
9/28/2008 09:34:03.89 [TX] - 114
9/28/2008 09:34:05.03 [TX] - 114
9/28/2008 09:34:06.13 [TX] - 114
9/28/2008 09:34:07.20 [TX] - 098
9/28/2008 09:34:07.22 [RX] - 004

9/28/2008 09:34:09.00 [TX] - 115
9/28/2008 09:34:09.53 [TX] - 114
9/28/2008 09:34:10.07 [TX] - 115
9/28/2008 09:34:10.45 [TX] - 114
9/28/2008 09:34:10.84 [TX] - 115
9/28/2008 09:34:11.18 [TX] - 114
9/28/2008 09:34:11.99 [TX] - 097
9/28/2008 09:34:12.00 [RX] - 007
9/28/2008 09:34:13.24 [TX] - 098
9/28/2008 09:34:13.26 [RX] - 007
```

Figure 27 Docklight command for conversion start and read the result event count

http://www.infineon.com

Published by Infineon Technologies AG