Multiple ADC channels conversion with CTimer triggering once for LPC55S69

The ADC of LPC55xx supports scan mode, in scan mode, once ADC triggering (either hardware or software) can convert multiple analog channels. The document gives an example that the CTImer2 module triggers ADC and ADC converts two analog channels for each triggering.
The doc introduces the CTimer configuration, ADC triggering control register configuration, and ADC Command buffer chain and ADC result reading , in this way, the CTimer can trigger ADC, the ADC can convert multiple channels.
The ADC of LPC55xx supports hardware trigger mode, each hardware trigger source corresponds to a trigger register, as the Table 753 copied from UM11126.pdf, the ct2_mat3_out triggering source index is 7, so you have to initialize the ADC trigger control register TCTRL7. In the TCTRL7 register, you can assign the command buffer, select the FIFO0 or FIFO1 to save the ADC result, enable hardware triggering. In order to implement the scan mode to convert multiple ADC analog channels, you have to initialize multiple Command Buffers so that multiple Command Buffers can be chained each other, each Command Buffers specify one or two ADC analog channels.

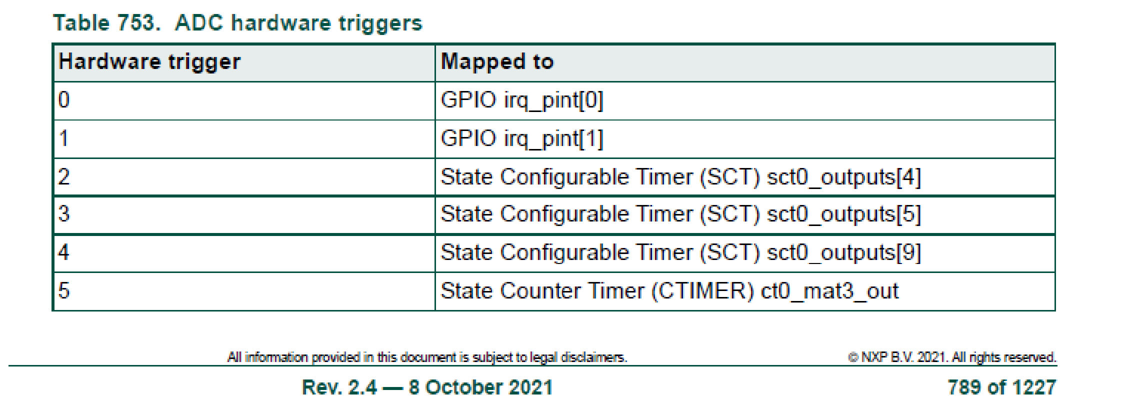
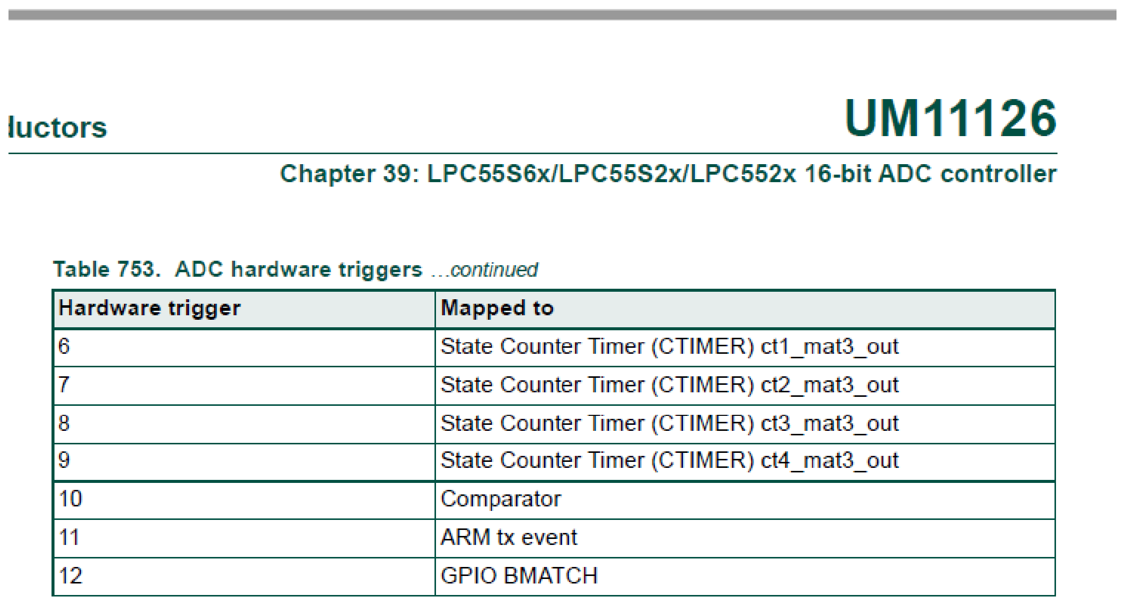


Table 753. ADC hardware triggers

Hardware trigger	Mapped to
0	GPIO irq_pint[0]
1	GPIO irq_pint[1]
2	State Configurable Timer (SCT) sct0_outputs[4]
3	State Configurable Timer (SCT) sct0_outputs[5]
4	State Configurable Timer (SCT) sct0_outputs[9]
5	State Counter Timer (CTIMER) ct0_mat3_out

All information provided in this document is subject to legal disclaimers. © NXP B.V. 2021. All rights reserved.

Rev. 2.4 — 8 October 2021 789 of 1227



luctors **UM11126**

Chapter 39: LPC55S6x/LPC55S2x/LPC552x 16-bit ADC controller

Table 753. ADC hardware triggers *…continued*

Hardware trigger	Mapped to
6	State Counter Timer (CTIMER) ct1_mat3_out
7	State Counter Timer (CTIMER) ct2_mat3_out
8	State Counter Timer (CTIMER) ct3_mat3_out
9	State Counter Timer (CTIMER) ct4_mat3_out
10	Comparator
11	ARM tx event
12	GPIO BMATCH

1. CTimer configuration.

```
void CTimerInit(void)
{
ctimer_config_t config;
 ctimer_match_config_t matchConfig;
    /* Use 12 MHz clock for some of the Ctimers */
    CLOCK_AttachClk(kFRO_HF_to_CTIMER2);
    CTIMER_GetDefaultConfig(&config);

    CTIMER_Init(CTIMER2, &config);

    matchConfig.enableCounterReset = true;
    matchConfig.enableCounterStop = false;
    matchConfig.matchValue = 12000000; //the CTimer frequency is
12MHz/12000000=1Hz
    matchConfig.outControl = kCTIMER_Output_Toggle;
    matchConfig.outPinInitState = true;
    matchConfig.enableInterrupt = false;
    CTIMER_SetupMatch(CTIMER2, kCTIMER_Match_3, &matchConfig);
    CTIMER_StartTimer(CTIMER2);
}
```

The `void CTimerInit(void) api function initializes CTimer2, in the function,` the CTimer driving clock source is FRO-HF, which is 12MHz. Because the CTimer2_mat3 signal is toggled once the CTimer2 counter reaches up to the match3 value 12000000, so the CTimer2_mat3 signal frequency is 12MHz/(2*12000000)=0.5Hz.

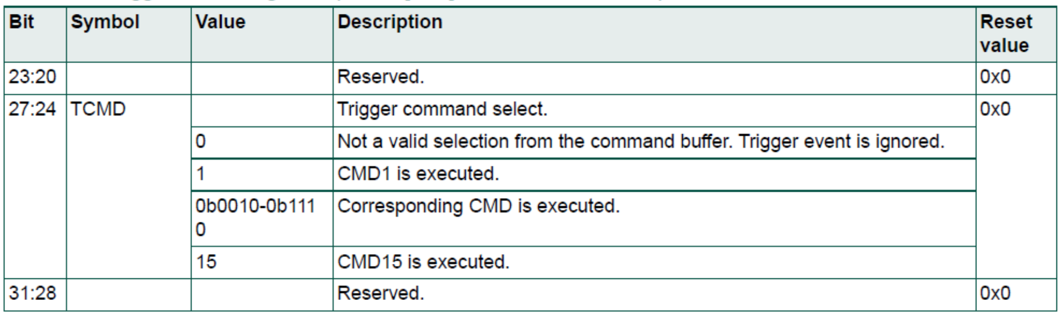
The "Table 735 ADC Hardware triggers" describes the CTimer incorrectly, the CTimer is standard counter/timers, the "state counter timer" should have been "standard counter/timers" or use only "CTimer".

The ADC triggering index for ct2_mat3_out signal is 7, so user has to configure the TCTRL7 register, especially the command buffer in the TCTRL7 register.

Based on the lpadc_interrupt project in SDK package for LPC55S69-EVK board, I develop the code

2. Initialize the ADC trigger control register

Table 740. Trigger control registers (TCTRL[0:15], offsets 0xA0 to 0xDC) *...continued*



Bit	Symbol	Value	Description	Reset value
23:20			Reserved.	0x0
27:24	TCMD		Trigger command select.	0x0
		0	Not a valid selection from the command buffer. Trigger event is ignored.	
		1	CMD1 is executed.	
		0b0010-0b1110	Corresponding CMD is executed.	
		15	CMD15 is executed.	
31:28			Reserved.	0x0

The original ADC example for LPC5569 uses software triggering mode, in software triggering mode, any TCTRLx register can be used. But for ct2_mat3_out signal triggering, it corresponds to trigger register 7 by hardware connection, so you have to initialize only TCTRL7 register and enable hardware triggering mode. In the TCTRLx register, there is TCMD bits(Trigger command select) as above table 740, with which the trigger source is bonded to a specific command buffer, in the command buffer, the analog channel and conversion type can be defined.

```
#define DEMO_LPADC_USER_CMDID            1U /* CMD1 */
#define DEMO_LPADC_USER_CMDID_2          2U /* CMD2 */
#define CTIMER2_TRIGGER_INDEX 7
```

```
/* Set trigger configuration. */
    LPADC_GetDefaultConvTriggerConfig(&mLpadcTriggerConfigStruct);
    mLpadcTriggerConfigStruct.targetCommandId       = DEMO_LPADC_USER_CMDID;
    mLpadcTriggerConfigStruct.enableHardwareTrigger = true;
    LPADC_SetConvTriggerConfig(DEMO_LPADC_BASE, CTIMER2_TRIGGER_INDEX,
&mLpadcTriggerConfigStruct);
```

There are total 15 command buffers ranging from 1 to 15, note that the command buffer 0 does not exist, it means the command buffer chain has terminated if the Next bits in command buffer is assigned as 0.

The assigned command buffer is `DEMO_LPADC_USER_CMDID or 1.`

`the CTIMER2_TRIGGER_INDEX is 7.`

`The above code initialize the TCTRL7 register, the TCTRL7[TCMD]= DEMO_LPADC_USER_CMDID=1`

3. ADC analog channel configuration:

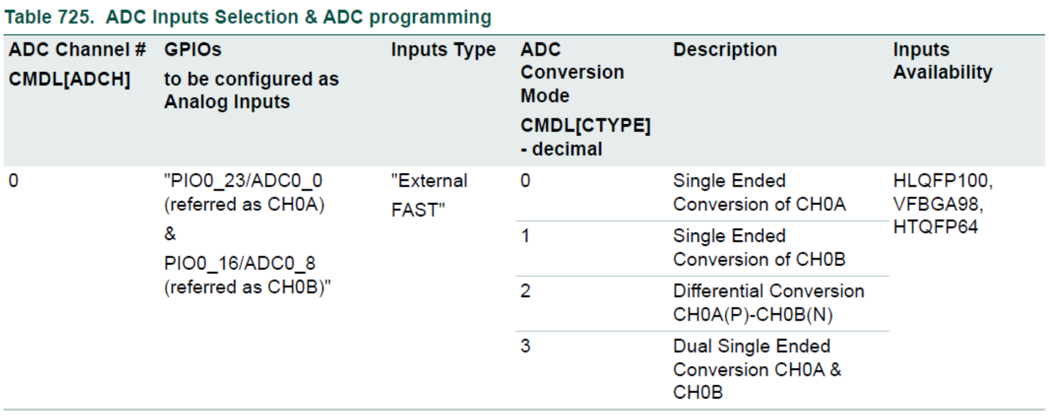


Table 725. ADC Inputs Selection & ADC programming

ADC Channel # CMDL[ADCH]	GPIOs to be configured as Analog Inputs	Inputs Type	ADC Conversion Mode CMDL[CTYPE] - decimal	Description	Inputs Availability
0	"PIO0_23/ADC0_0 (referred as CH0A) & PIO0_16/ADC0_8 (referred as CH0B)"	"External FAST"	0	Single Ended Conversion of CH0A	HLQFP100, VFBGA98, HTQFP64
			1	Single Ended Conversion of CH0B	
			2	Differential Conversion CH0A(P)-CH0B(N)	
			3	Dual Single Ended Conversion CH0A & CH0B	

In the example, two channels ADC0_CH0A(PIO0_23 pin) and ADC0_CH0B(PIO0_16 pin) are measured.

The PIO0_23 is ADC0_CH0A, it is ADC channel 0 with CTYPE bits as 0, which means single-ended mode and A side channel.

The following code used to initialize the command buffer1 and command buffer2. The command buffer2 is chained by command buffer1. Each command buffer includes the components: ADC analog channel, conversion type such as single-ended side A or single-ended side B to select different ADC pin with the same analog channels such as ADC0_0A or ADC0_0B, next chained command buffer.

3.1 initialize Commend buffer1

```
#define DEMO_LPADC_USER_CHANNEL 0
/* Set conversion CMD1 configuration. ADC channel is ADC0_0A PIO0_23*/
    LPADC_GetDefaultConvCommandConfig(&mLpadcCommandConfigStruct);
    mLpadcCommandConfigStruct.channelNumber = DEMO_LPADC_USER_CHANNEL;
    mLpadcCommandConfigStruct.sampleChannelMode
=kLPADC_SampleChannelSingleEndSideA;
    mLpadcCommandConfigStruct.chainedNextCommandNumber =
DEMO_LPADC_USER_CMDID_2;
#if defined(DEMO_LPADC_USE_HIGH_RESOLUTION) &&
DEMO_LPADC_USE_HIGH_RESOLUTION
    mLpadcCommandConfigStruct.conversionResolutionMode =
kLPADC_ConversionResolutionHigh;
#endif /* DEMO_LPADC_USE_HIGH_RESOLUTION */
    LPADC_SetConvCommandConfig(DEMO_LPADC_BASE, DEMO_LPADC_USER_CMDID,
&mLpadcCommandConfigStruct);
```

3.2 3.1 initialize Commend buffer2

The PIO0_16 is ADC0_CH0B, it is ADC channel 0 with CTYPE bits as 1, which means single-ended mode and B side channel.

```
LPADC_GetDefaultConvCommandConfig(&mLpadcCommandConfigStruct);
    mLpadcCommandConfigStruct.channelNumber = DEMO_LPADC_USER_CHANNEL;
    mLpadcCommandConfigStruct.sampleChannelMode
=kLPADC_SampleChannelSingleEndSideB;
    mLpadcCommandConfigStruct.chainedNextCommandNumber=0;
#if defined(DEMO_LPADC_USE_HIGH_RESOLUTION) &&
DEMO_LPADC_USE_HIGH_RESOLUTION
    mLpadcCommandConfigStruct.conversionResolutionMode =
kLPADC_ConversionResolutionHigh;
#endif /* DEMO_LPADC_USE_HIGH_RESOLUTION */
        LPADC_SetConvCommandConfig(DEMO_LPADC_BASE,  DEMO_LPADC_USER_CMDID_2,
    &mLpadcCommandConfigStruct);
```

In other words, for the two analog channels ADC0_CH0A(PIO0_23 pin) and ADC0_CH0B(PIO0_16 pin), the channel index is the same as “0”, but the CType mode is different, ADC0_CH0A(PIO0_23 pin) analog channel is A side Single-ended mode, ADC0_CH0B(PIO0_16 pin) is B side Single-ended mode.

For the pin configuration, you have to configure the IOCON register with the code:

```
/ADC0_0 function as ADC0_0A
    const uint32_t port0_pin23_config = (/* Pin is configured as ADC0_0 */
                                         IOCON_PIO_FUNC0 |
                                         /* No addition pin function */
                                         IOCON_PIO_MODE_INACT |
                                         /* Standard mode, output slew rate
control is enabled */
                                         IOCON_PIO_SLEW_STANDARD |
                                         /* Input function is not inverted */
                                         IOCON_PIO_INV_DI |
                                         /* Enables analog function */
                                         IOCON_PIO_ANALOG_EN |
                                         /* Open drain is disabled */
                                         IOCON_PIO_OPENDRAIN_DI |
                                         /* Analog switch is closed (enabled)
*/
                                         IOCON_PIO_ASW_EN);
    /* PORT0 PIN23 (coords: 20) is configured as ADC0_0A */
    IOCON_PinMuxSet(IOCON, 0U, 23U, port0_pin23_config);

    //ADC0_8 function as ADC0_0B
    const uint32_t port0_pin16_config = (/* Pin is configured as ADC0_8 */
                                         IOCON_PIO_FUNC0 |
                                         /* No addition pin function */
                                         IOCON_PIO_MODE_INACT |
                                         /* Standard mode, output slew rate
control is enabled */
                                         IOCON_PIO_SLEW_STANDARD |
                                         /* Input function is not inverted */
                                         IOCON_PIO_INV_DI |
                                         /* Enables analog function */
                                         IOCON_PIO_ANALOG_EN |
                                         /* Open drain is disabled */
                                         IOCON_PIO_OPENDRAIN_DI |
                                         /* Analog switch is closed (enabled)
*/
                                         IOCON_PIO_ASW_EN);
    /* PORT0 PIN16  is configured as ADC0_0B */
        IOCON_PinMuxSet(IOCON, 0U, 16U, port0_pin16_config);
```

4. **Set up the ADC channel chain feature so that multiple analog channels can be converted with one trigger.**
   The ADC of LPC55xx supports scan mode, in other words, multiple ADC analog channels can be converted with one ADC trigger, the trigger can be either software trigger or

hardware trigger.
The command buffer specify the analog channel and conversion type, in the command buffer register, there is a “Next” bits as the following figure, which specifies the chained command buffer, if the Next bits is assigned for example 5, the next chained command buffer index is 5, so the command buffer 5 will define the chained analog channel. With the method, you can convert multiple analog channels.

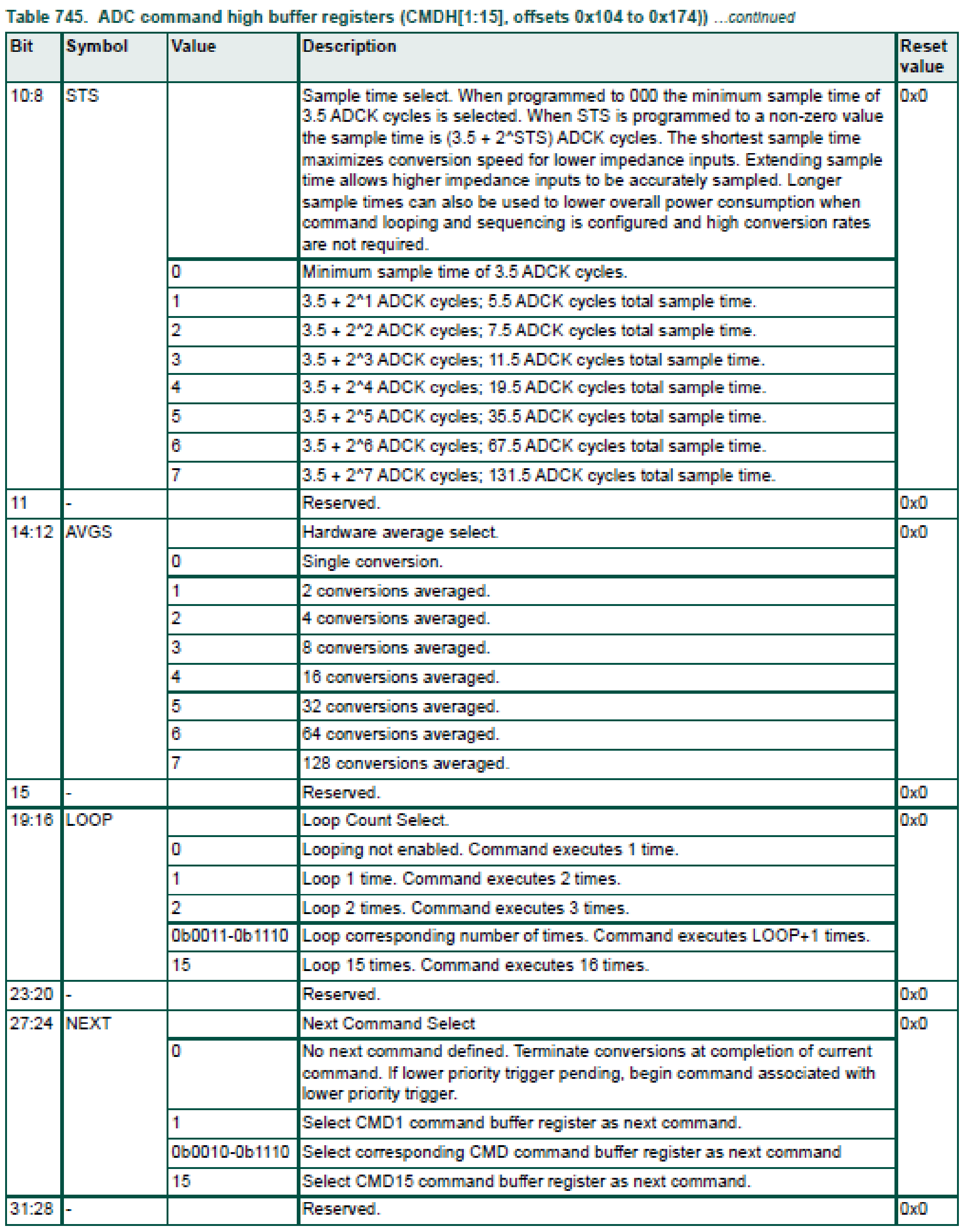
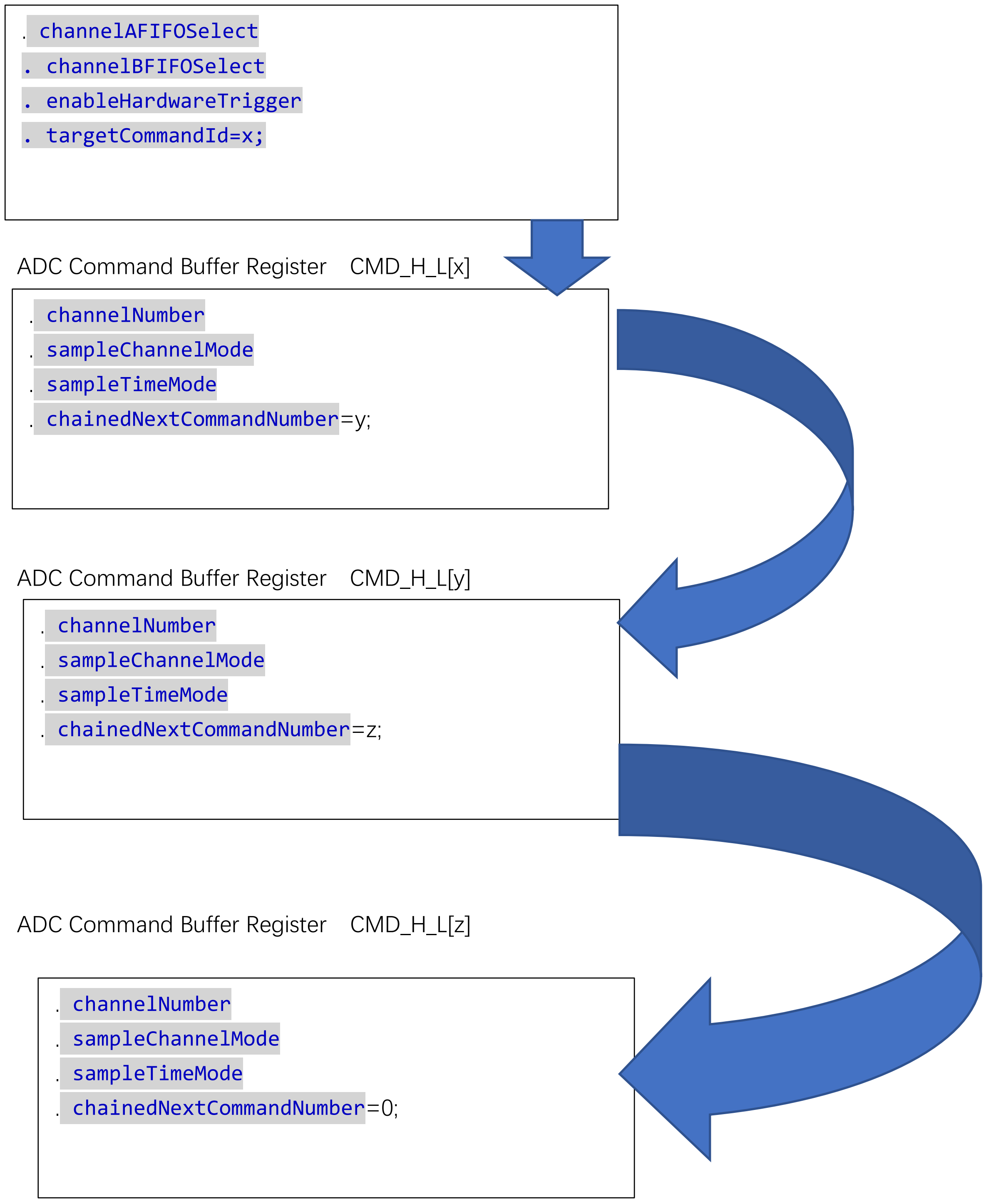


Table 745. ADC command high buffer registers (CMDH[1:15], offsets 0x104 to 0x174)) *...continued*

Bit	Symbol	Value	Description	Reset value
10:8	STS		Sample time select. When programmed to 000 the minimum sample time of 3.5 ADCK cycles is selected. When STS is programmed to a non-zero value the sample time is (3.5 + 2^STS) ADCK cycles. The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.	0x0
		0	Minimum sample time of 3.5 ADCK cycles.	
		1	3.5 + 2^1 ADCK cycles; 5.5 ADCK cycles total sample time.	
		2	3.5 + 2^2 ADCK cycles; 7.5 ADCK cycles total sample time.	
		3	3.5 + 2^3 ADCK cycles; 11.5 ADCK cycles total sample time.	
		4	3.5 + 2^4 ADCK cycles; 19.5 ADCK cycles total sample time.	
		5	3.5 + 2^5 ADCK cycles; 35.5 ADCK cycles total sample time.	
		6	3.5 + 2^6 ADCK cycles; 67.5 ADCK cycles total sample time.	
		7	3.5 + 2^7 ADCK cycles; 131.5 ADCK cycles total sample time.	
11	-		Reserved.	0x0
14:12	AVGS		Hardware average select.	0x0
		0	Single conversion.	
		1	2 conversions averaged.	
		2	4 conversions averaged.	
		3	8 conversions averaged.	
		4	16 conversions averaged.	
		5	32 conversions averaged.	
		6	64 conversions averaged.	
		7	128 conversions averaged.	
15	-		Reserved.	0x0
19:16	LOOP		Loop Count Select.	0x0
		0	Looping not enabled. Command executes 1 time.	
		1	Loop 1 time. Command executes 2 times.	
		2	Loop 2 times. Command executes 3 times.	
		0b0011-0b1110	Loop corresponding number of times. Command executes LOOP+1 times.	
		15	Loop 15 times. Command executes 16 times.	
23:20	-		Reserved.	0x0
27:24	NEXT		Next Command Select	0x0
		0	No next command defined. Terminate conversions at completion of current command. If lower priority trigger pending, begin command associated with lower priority trigger.	
		1	Select CMD1 command buffer register as next command.	
		0b0010-0b1110	Select corresponding CMD command buffer register as next command	
		15	Select CMD15 command buffer register as next command.	
31:28	-		Reserved.	0x0

For the software, you can use the scheme to implement the command buffer chain so that you can convert multiple analog channels.
When the .`chainedNextCommandNumber=0,` it means that the command buffer chain is terminated.

This is the software flow chart to implement chained command buffer in SDK



```
. channelAFIFOSelect
. channelBFIFOSelect
. enableHardwareTrigger
. targetCommandId=x;
```

ADC Command Buffer Register CMD_H_L[x]

```
. channelNumber
. sampleChannelMode
. sampleTimeMode
. chainedNextCommandNumber=y;
```

ADC Command Buffer Register CMD_H_L[y]

```
. channelNumber
. sampleChannelMode
. sampleTimeMode
. chainedNextCommandNumber=z;
```

ADC Command Buffer Register CMD_H_L[z]

```
. channelNumber
. sampleChannelMode
. sampleTimeMode
. chainedNextCommandNumber=0;
```

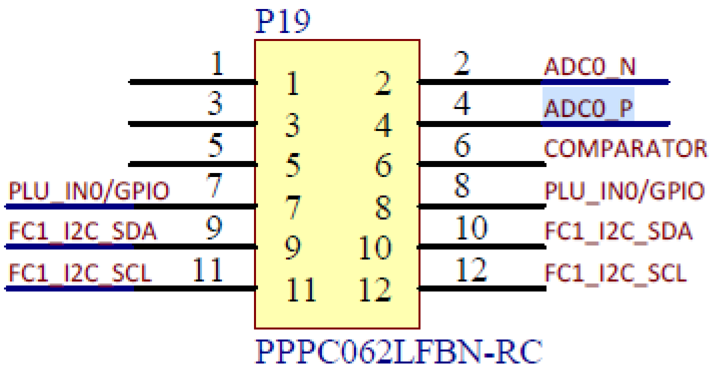
5. The multiple ADC channels reading

In the ADC ISR(Interrupt Service Routine), first of all, read the FCOUNT bits in FCTRL[0] **register to get the** number of entries stored in each FIFO0 if you use the FIFO0 to save sample results, then based on the number of ADC result entries, read the ADC results from FIFO0. The ADC result in FIFO includes ADC sample, command buffer index, trigger sources, with the commend buffer index, you can know the analog channel the ADC result corresponds.

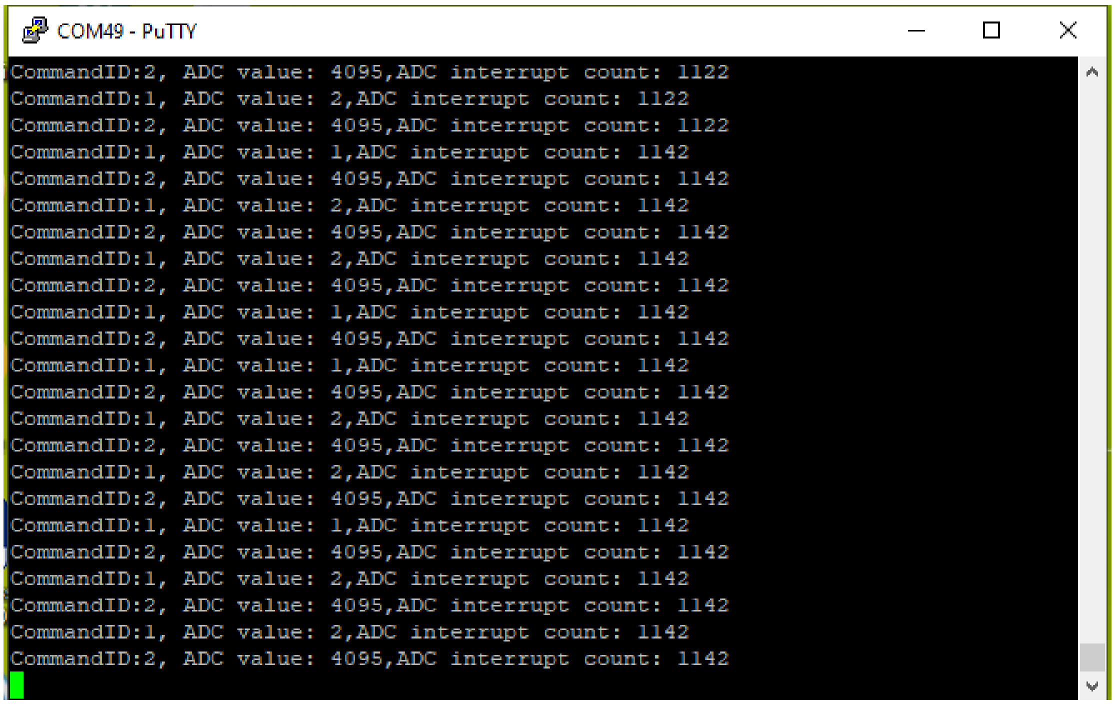
```
lpadc_conv_result_t g_LpadcResultConfigStruct[SAMPLE_NUMBER];
uint32_t channelNumber, channelNumberIndex, displayIndex;
void DEMO_LPADC_IRQ_HANDLER_FUNC(void)
{

 channelNumber=LPADC_GetConvResultCount(DEMO_LPADC_BASE,0);
    g_LpadcInterruptCounter++;
    GPIO_PortToggle(GPIO,1,1<<4);
    for(uint32_t i=0; i<channelNumber; i++)
    {
    LPADC_GetConvResult(DEMO_LPADC_BASE,
&g_LpadcResultConfigStruct[channelNumberIndex], 0U);
    channelNumberIndex++;
    }
    if(channelNumberIndex>=SAMPLE_NUMBER)
    {
        g_LpadcConversionCompletedFlag = true;
        channelNumberIndex=0;
    }
#if 0
#if (defined(FSL_FEATURE_LPADC_FIFO_COUNT) &&
(FSL_FEATURE_LPADC_FIFO_COUNT == 2U))

    if (LPADC_GetConvResult(DEMO_LPADC_BASE, &g_LpadcResultConfigStruct,
0U))
#else
    if (LPADC_GetConvResult(DEMO_LPADC_BASE,
&g_LpadcResultConfigStruct))
#endif /* FSL_FEATURE_LPADC_FIFO_COUNT */
   {
        g_LpadcConversionCompletedFlag = true;
    }
#endif
    SDK_ISR_EXIT_BARRIER;
}
```

6. **ADC conversion result:**

The ADC example uses ct2_mat3_out signal to trigger ADC with hardware mode, the example convert two analog channels with once trigger. The first channel is PIO0_23, which is connected to pin4 of P19 connector on LPC55S69-EVK as ADC0_P, the second channel is PIO0_16, which is connected to pin2 of P19 connector on LPC55S69-EVK as ADC0_N.



When the PIO0_23 pin is connected to GND, the PIO0_16 is connected to 3.3V, this is the result. Command ID 1 corresponds to ADC0_CH0A(PIO0_23 pin, pin4 of P19), Command ID 2 corresponds to ADC0_CH0B(PIO0_16, pin2 of P19).
You can see that two analog channels are converted and reading are interleaved ADC0_CH0A/ADC0_CH0B/ ADC0_CH0A/ADC0_CH0B/…………..



```
CommandID:2, ADC value: 4095,ADC interrupt count: 1122
CommandID:1, ADC value: 2,ADC interrupt count: 1122
CommandID:2, ADC value: 4095,ADC interrupt count: 1122
CommandID:1, ADC value: 1,ADC interrupt count: 1142
CommandID:2, ADC value: 4095,ADC interrupt count: 1142
CommandID:1, ADC value: 2,ADC interrupt count: 1142
CommandID:2, ADC value: 4095,ADC interrupt count: 1142
CommandID:1, ADC value: 2,ADC interrupt count: 1142
CommandID:2, ADC value: 4095,ADC interrupt count: 1142
CommandID:1, ADC value: 1,ADC interrupt count: 1142
CommandID:2, ADC value: 4095,ADC interrupt count: 1142
CommandID:1, ADC value: 1,ADC interrupt count: 1142
CommandID:2, ADC value: 4095,ADC interrupt count: 1142
CommandID:1, ADC value: 2,ADC interrupt count: 1142
CommandID:2, ADC value: 4095,ADC interrupt count: 1142
CommandID:1, ADC value: 2,ADC interrupt count: 1142
CommandID:2, ADC value: 4095,ADC interrupt count: 1142
CommandID:1, ADC value: 1,ADC interrupt count: 1142
CommandID:2, ADC value: 4095,ADC interrupt count: 1142
CommandID:1, ADC value: 2,ADC interrupt count: 1142
CommandID:2, ADC value: 4095,ADC interrupt count: 1142
CommandID:1, ADC value: 2,ADC interrupt count: 1142
CommandID:2, ADC value: 4095,ADC interrupt count: 1142
```