# ARM TX Event to trigger ADC

The doc discusses how to use ARM TX Event to trigger ADC based on LPC55S69-EVK board and MCUXPresso IDE and SDK.
As the Table 753, the “ARM tx event” is the one of the ADC triggering source, although it is a hardware triggering, but the mechanism is the same as software triggering, after ADC configuration, after you execute the instruction asm(“SEV”); the ARM tx event will be generated and trigger ADC to sample.
The ADC of LPC55xx support hardware trigger mode, each hardware trigger source corresponds to a trigger register, as the Table 753, the ARM tx event ASC triggering source index is 11, so you have to initialize the ADC trigger control register TCTRL11. In the TCTRL11 register, you can assign the command ID, select the FIFO0 or FIFO1 to save the ADC result, enable hardware triggering.

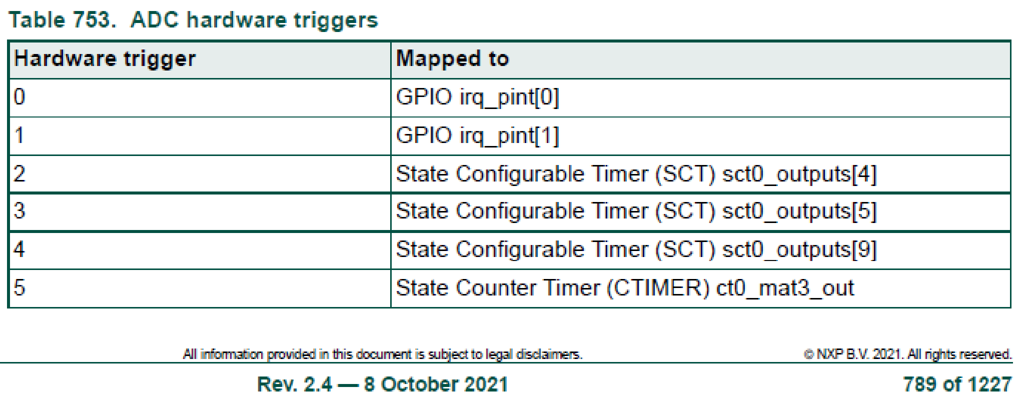
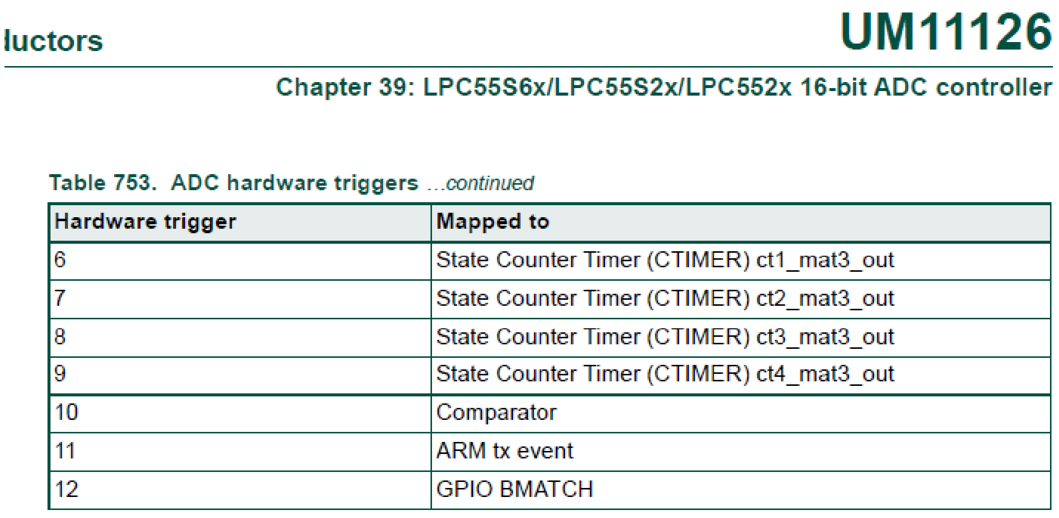


Table 753. ADC hardware triggers

| 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 |

After you execute asm(“SEV”); the ADC triggering source 11 will generate event to trigger ADC.
Based on the lpadc_interrupt project in SDK package for LPC55S69-EVK board, I modify two

location.

1) the original code initializes the TCTRL0 register, because it uses software triggering mode, any TCTRLx register can be used. But for ARM tx event, you have to use TCTRL11 register and enable hardware triggering mode.

```
    mLpadcTriggerConfigStruct.enableHardwareTrigger = true;
    LPADC_SetConvTriggerConfig(DEMO_LPADC_BASE, 11U,
&mLpadcTriggerConfigStruct); /* Configurate the trigger11. */
```

2) for each sampling, execute the asm("SEV");, after the instruction, ADC will convert, after the conversion is completed, ADC ISR(Interrupt Service Routine) will be executed, `g_LpadcConversionCompletedFlag will be set.`

   This is the code:

```
    while (1)
    {
        GETCHAR();
        //LPADC_DoSoftwareTrigger(DEMO_LPADC_BASE, 1U); /* 1U is trigger0
mask. */
        asm("SEV");
        while (!g_LpadcConversionCompletedFlag)
        {
        }
        PRINTF("ADC value: %d\r\nADC interrupt count: %d\r\n",
               ((g_LpadcResultConfigStruct.convValue) >>
g_LpadcResultShift), g_LpadcInterruptCounter);
        g_LpadcConversionCompletedFlag = false;
    }
```

ADC code after modification:

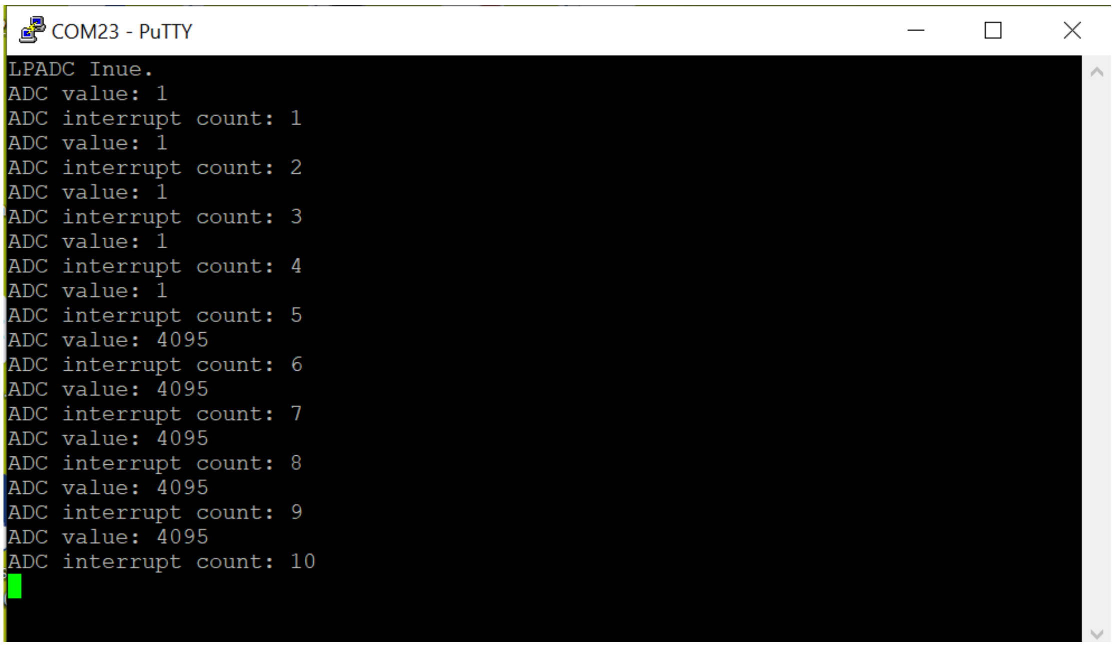
```
int main(void)
{
    lpadc_config_t mLpadcConfigStruct;
    lpadc_conv_trigger_config_t mLpadcTriggerConfigStruct;
    lpadc_conv_command_config_t mLpadcCommandConfigStruct;

    /* set BOD VBAT level to 1.65V */
    POWER_SetBodVbatLevel(kPOWER_BodVbatLevel1650mv,
kPOWER_BodHystLevel50mv, false);
    /* attach main clock divide to FLEXCOMM0 (debug console) */
    CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);

    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();
    /* Set clock source for ADC0 */
    CLOCK_SetClkDiv(kCLOCK_DivAdcAsyncClk, 8U, true);
    CLOCK_AttachClk(kMAIN_CLK_to_ADC_CLK);

    /* Disable LDOGPADC power down */
    POWER_DisablePD(kPDRUNCFG_PD_LDOGPADC);

    ANACTRL_Init(ANACTRL);
    ANACTRL_EnableVref1V(ANACTRL, true);

    PRINTF("LPADC Interrupt Example\r\n");

    LPADC_GetDefaultConfig(&mLpadcConfigStruct);
    mLpadcConfigStruct.enableAnalogPreliminary = true;
#if defined(DEMO_LPADC_VREF_SOURCE)
    mLpadcConfigStruct.referenceVoltageSource = DEMO_LPADC_VREF_SOURCE;
#endif /* DEMO_LPADC_VREF_SOURCE */
#if defined(FSL_FEATURE_LPADC_HAS_CTRL_CAL_AVGS) &&
FSL_FEATURE_LPADC_HAS_CTRL_CAL_AVGS
    mLpadcConfigStruct.conversionAverageMode = kLPADC_ConversionAverage128;
#endif /* FSL_FEATURE_LPADC_HAS_CTRL_CAL_AVGS */
    LPADC_Init(DEMO_LPADC_BASE, &mLpadcConfigStruct);

#if defined(FSL_FEATURE_LPADC_HAS_CTRL_CALOFS) &&
FSL_FEATURE_LPADC_HAS_CTRL_CALOFS
#if defined(FSL_FEATURE_LPADC_HAS_OFSTRIM) && FSL_FEATURE_LPADC_HAS_OFSTRIM
    /* Request offset calibration. */
#if defined(DEMO_LPADC_DO_OFFSET_CALIBRATION) &&
DEMO_LPADC_DO_OFFSET_CALIBRATION
    LPADC_DoOffsetCalibration(DEMO_LPADC_BASE);
#else
    LPADC_SetOffsetValue(DEMO_LPADC_BASE, DEMO_LPADC_OFFSET_VALUE_A,
DEMO_LPADC_OFFSET_VALUE_B);
#endif /* DEMO_LPADC_DO_OFFSET_CALIBRATION */
#endif /* FSL_FEATURE_LPADC_HAS_OFSTRIM */
    /* Request gain calibration. */
    LPADC_DoAutoCalibration(DEMO_LPADC_BASE);
#endif /* FSL_FEATURE_LPADC_HAS_CTRL_CALOFS */

#if (defined(FSL_FEATURE_LPADC_HAS_CFG_CALOFS) &&
FSL_FEATURE_LPADC_HAS_CFG_CALOFS)
    /* Do auto calibration. */
    LPADC_DoAutoCalibration(DEMO_LPADC_BASE);
#endif /* FSL_FEATURE_LPADC_HAS_CFG_CALOFS */
```

```
    /* Set conversion CMD configuration. */
    LPADC_GetDefaultConvCommandConfig(&mLpadcCommandConfigStruct);
    mLpadcCommandConfigStruct.channelNumber = DEMO_LPADC_USER_CHANNEL;
#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);

    /* Set trigger configuration. */
    LPADC_GetDefaultConvTriggerConfig(&mLpadcTriggerConfigStruct);
    mLpadcTriggerConfigStruct.targetCommandId       = DEMO_LPADC_USER_CMDID; 
/* CMD15 is executed. */
    mLpadcTriggerConfigStruct.enableHardwareTrigger = true;
    LPADC_SetConvTriggerConfig(DEMO_LPADC_BASE, 11U, 
&mLpadcTriggerConfigStruct); /* Configurate the trigger0. */

/* Enable the watermark interrupt. */
#if (defined(FSL_FEATURE_LPADC_FIFO_COUNT) && (FSL_FEATURE_LPADC_FIFO_COUNT 
== 2U))
    LPADC_EnableInterrupts(DEMO_LPADC_BASE, 
kLPADC_FIFO0WatermarkInterruptEnable);
#else
    LPADC_EnableInterrupts(DEMO_LPADC_BASE, 
kLPADC_FIFOWatermarkInterruptEnable);
#endif /* FSL_FEATURE_LPADC_FIFO_COUNT */
    EnableIRQ(DEMO_LPADC_IRQn);

    PRINTF("ADC Full Range: %d\r\n", g_LpadcFullRange);
#if defined(FSL_FEATURE_LPADC_HAS_CMDL_CSCALE) && 
FSL_FEATURE_LPADC_HAS_CMDL_CSCALE
    if (kLPADC_SampleFullScale == 
mLpadcCommandConfigStruct.sampleScaleMode)
    {
        PRINTF("Full channel scale (Factor of 1).\r\n");
    }
    else if (kLPADC_SamplePartScale == 
mLpadcCommandConfigStruct.sampleScaleMode)
    {
        PRINTF("Divided input voltage signal. (Factor of 30/64).\r\n");
    }
```

```
#endif

    /* When the number of datawords stored in the ADC Result FIFO is
greater
     * than watermark value(0U), LPADC watermark interrupt would be
triggered.
     */
    PRINTF("Please press any key to get user channel's ADC value.\r\n");
    while (1)
    {
        GETCHAR();
        //LPADC_DoSoftwareTrigger(DEMO_LPADC_BASE, 1U); /* 1U is trigger0
mask. */
        asm("SEV");
        while (!g_LpadcConversionCompletedFlag)
        {
        }
        PRINTF("ADC value: %d\r\nADC interrupt count: %d\r\n",
               ((g_LpadcResultConfigStruct.convValue) >>
g_LpadcResultShift), g_LpadcInterruptCounter);
        g_LpadcConversionCompletedFlag = false;
    }
}
```

ADC result:

For each time you Click any button on PC, the ADC will convert once.



```
LPADC Inue.
ADC value: 1
ADC interrupt count: 1
ADC value: 1
ADC interrupt count: 2
ADC value: 1
ADC interrupt count: 3
ADC value: 1
ADC interrupt count: 4
ADC value: 1
ADC interrupt count: 5
ADC value: 4095
ADC interrupt count: 6
ADC value: 4095
ADC interrupt count: 7
ADC value: 4095
ADC interrupt count: 8
ADC value: 4095
ADC interrupt count: 9
ADC value: 4095
ADC interrupt count: 10
```