ADC Driver

TM4C123GH6PM Micro-Controller

Due to AUTOSAR standard

SWS 4.3.1

Author: Mohamed Ahmed Gebril Awad.

I’m a student at the faculty of engineering Alexandria University expected graduation is in 2019.
I’m interested in embedded system field especially automotive industry looking for a challenging position in a leading company where I can enrich and increase my experience in this interesting field.

This document purpose is to show how to configure the ADC driver for the best use.

This document isn’t responsible for explaining the APIs and the interfaces provided as all this information can be found in the AUTOSAR_ADC_SWS_4.3.1 document chapter 8 specifically.

Table of Contents

Introduction and functional overview:

This document describes the functionality, API and the configuration of the ADC driver for TM4C123GH6PM microcontroller.

This document is written for the graduation project purpose only it's used by other members in the team to facilitate working with the adc driver in other layers that uses it.

TM4C123GH6PM ADC overview:

The TM4C123GH6PM microcontroller provides two ADC modules with each having the following features:

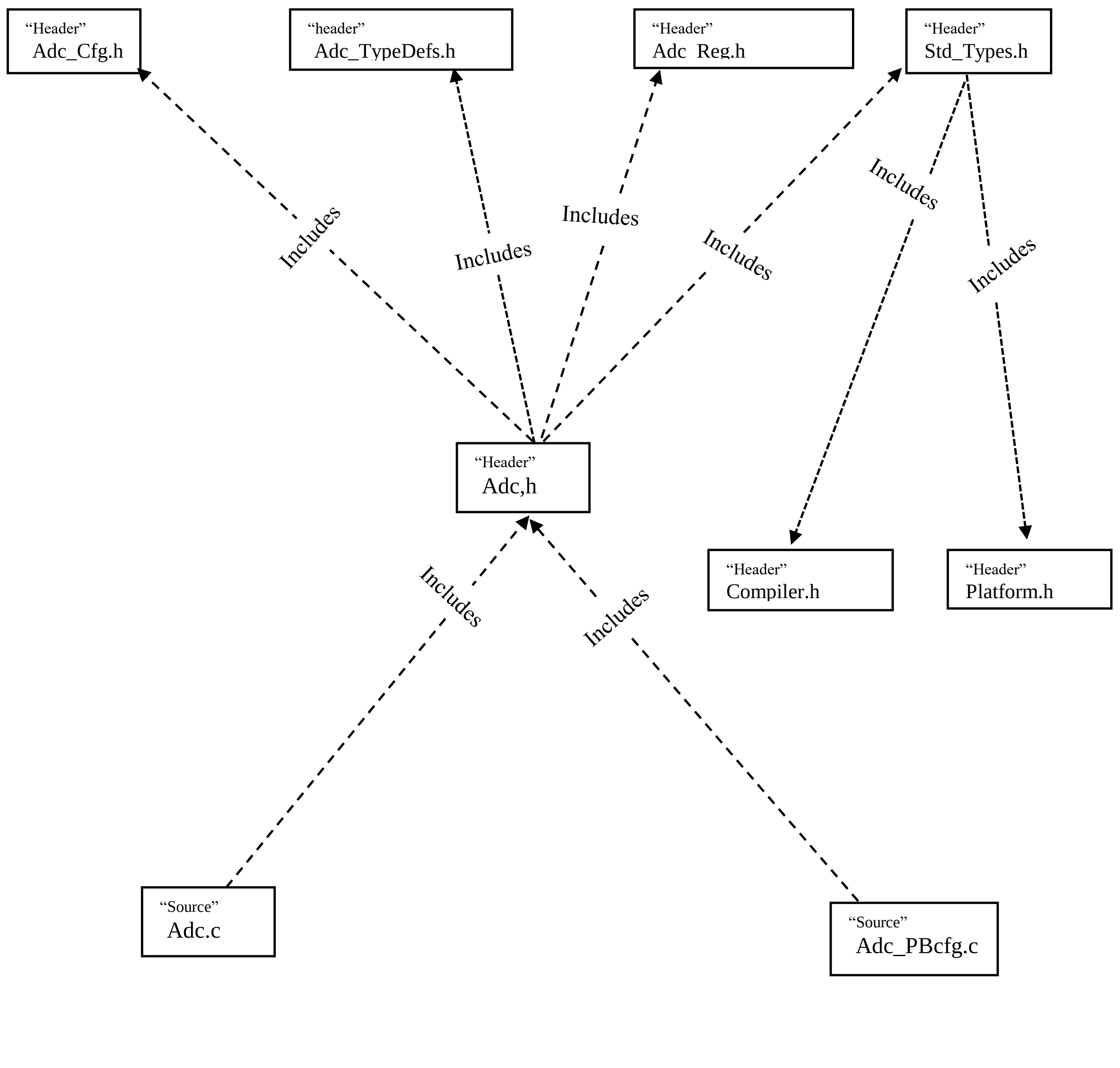
- Twelve shared analog input channels.
- 12-bit precision ADC.
- Single-ended and differential-input configuration.
- On-chip internal temperature sensor.
- Maximum sample rate of one million sample per second
- Optional phase shift in sample time programmable from 22.5º to 337.5º
- Four programmable sample conversion sequencers from one to eight entries long, with corresponding conversion results FIFOs
- Flexible trigger control
  - Controller (SW)
  - Timers
  - Analog Comparators
  - PWM
  - GPIO
- Hardware averaging of up to 64 samples
- Digital comparison unit of up to 64 samples
- Converter uses VDDA and GNDA as the voltage reference
- Power and ground for the analog circuitry is separate from the digital power and ground
- Efficient transfers using Micro Direct Memory Access Controller (uDMA)
  - Dedicated channel for each sample sequencer.
  - ADC modules uses burst requests for DMA.

ADC driver overview:

This driver is written with respect to ADC_SWS_4.3.1 AUTOSAR standard for the purpose of using it in graduation project.

All the work in this driver is an individual effort with some help from engineers working in the field of automotive embedded system.

Driver File structure:



Functional overview:

This driver works with respect to SWS.4.3.1, The ADC driver initializes and controls the internal Analogue Digital Converter unit(s) of the microcontroller. It provides services to start and stop a conversion respectively to enable and disable the trigger source of a conversion. Furthermore, it provides services to enable and disable a notification mechanism and routines to query the status and result of a conversion.

The ADC Driver shall work on so called ADC Channels. An ADC channel combines an analogue input pin, the needed ADC circuitry itself and a conversion result register into an entity that can be individually controlled and accessed via the ADC Driver.

This Adc Driver works with concept of ADC channel Group: which is a group of ADC channels linked to the same ADC hardware unit (e.g. one Sample&Hold and one A/D converter). The conversion of the whole group is triggered by one trigger source.

The result of the conversion of one ADC channel group is stored on what is called with respect to SWS 4.3.1 ADC result buffer which is: The user of the ADC Driver has to provide a buffer for every group. This buffer can hold multiple samples of the same channel group if streaming access mode is selected. If single access mode is selected one sample of each group channel is held in the buffer.

Expressions used with in ADC driver:

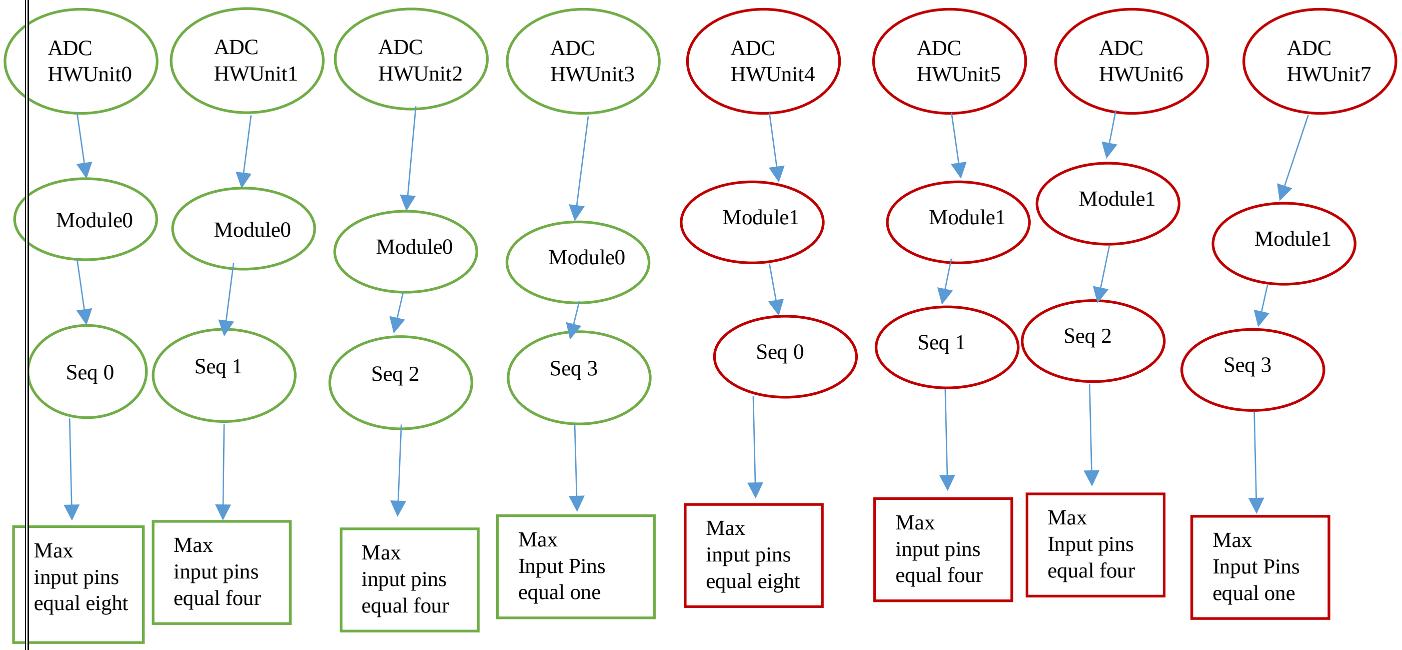
- HW Unit:
  Represents a microcontroller input electronic device that includes all parts necessary to perform an "analogue to digital conversion".
- ADC channel:
  Represents a logical ADC entity bound to one port pin. Multiple ADC entities can be mapped to the same port pin.
- ADC channel group:
  A group of ADC channels linked to the same ADC hardware unit (e.g. one Sample&Hold and one A/D converter). The conversion of the whole group is triggered by one trigger source.
- ADC result buffer:
  The user of the ADC Driver has to provide a buffer for every group. This buffer can hold multiple samples of the same channel group if streaming access mode is selected. If single access mode is selected one sample of each group channel is held in the buffer.
- Trigger Source: Source Events that starts a single conversion or a continues series of conversions.
- Conversion Mode: One shot:
  The conversion of an ADC channel group is performed once after a trigger and result is written to assigned buffer.
  A trigger can be a software API call or hardware event.
  Continuous: the conversion of an ADC channel group are performed continuously after a software API call(start) the conversion themselves are running automatically (hardware/ interrupt controlled). The continuous conversions can be stopped by a software API call(stop).
- Sample Sequencer:
  The sampling control and data capture is handled by the sample sequencers. All of the sequencers are identical in implementation except for the number of samples that can be captured and the depth of the FIFO where the conversion results are stored in HW.
  You can refer to AUTOSAR_SWS_ADC_4.3.1 and TM4C123GH6PM data sheet for more information

Main blocks if the ADC driver and hardware limitations:

I divided the ADC driver into eight HW units where each HW is implemented as a software C structure. Each HW unit contains one ADC module and one ADC sample sequencer and one ADC channel group.
Every ADC channel group has different number of maximum input pins that can be assigned to because that number depends on the HW unit you'll assign the ADC channel group to.

According to the TM4C123GH6PM data sheet every sample sequencer can be assigned a maximum number of input channel that's why ADC channel group has different number of maximum input pins.

ADC driver block diagram:



Adc_Typedefs.h header file:

This header contains the typedefs needed for the use in the ADC driver, this is useful to abstract the upper layers e.g. sensor and actuator layer from the Micro-Controller input/output layer.
All the information needed to know about the typedefs can be found in AUTOSAR_ADC_SWS_4.3.1.

Adc.h header file:

This header contains all the interfaces and typedefs that anyone want to use this driver will need, so any one intends to use this driver has to include this file in his project.

API Specifications:

Chapter eight in AUTOSAR_ADC_SWS_4.3.1 gives a detailed description about the interfaces used in this ADC driver although not all APIs are implemented as they are not needed in my graduation project, - anyone who can add and implement rest functions in the standard is welcome -.

Adc_Init

Void Adc_Init(const Adc_ConfigType* ConfigPtr);

This API is used to initialize the the ADC driver, you ‘ll not able to use any other API without this API called at the beginning of your program.

Parameters: Pointer to a struct of type Adc_ConfigType, you have to pass the struct that is defined in Adc_PBCfg.c, this struct is seen as it’s extern in Adc.h.

Errors: If DET (Development Error Tracer module) is implemented one error will be arisen If you called the function and you’ve called it before **the error** ADC_E_ALREADY_INITIALIZED.

Setup Result Buffer:

Std_ReturnType Adc_SetupResultBuffer(
 Adc_GroupType Group,
 Adc_ValueGroupType* DataBufferPtr
 );

Initializes ADC driver with the group specific result buffer start address where the conversion results will be stored. The application has to ensure that the application buffer, where DataBufferPtr points to, can hold all the conversion results of the specified group.
The initialization with Adc_SetupResultBuffer is required after reset, before a group conversion can be started.

Parameters: Adc_ChannelGroup index.
pointer to result data buffer.

Errors:
1- ADC_E_PARAM_GROUP wrong group index.
2- ADC_E_BUSY the Adc_ChannelGroup status is not idle.
3- ADC_E_UNINIT Adc_Init() wasn’t called.

Software Trigger Conversion:

void Adc_StartGroupConversion(Adc_GroupType Group); Used in starting software conversion you can call it for different channel groups as Hw priority mechanism is impelemented.

Parametes: the Adc_ChannelGroup index.

Errors: If DET (Development Error Tracer module) is implemented

1- ADC_E_PARAM_GROUP if you passed a wrong group index.
2- ADC_E_WRONG_TRIGG_SRC if the passed group index trigger source is HW not Software.

Read_Group:

Std_ReturnType Adc_ReadGroup(
Adc_GroupType Group,
Adc_ValueGroupType* DataBufferPtr
);
Reads the group conversion result of the last completed conversion round of the requested group
and stores the channel values starting at the DataBufferPtr address. The group channel values are stored in ascending channel number order.

Parameters: 1- Adc_ChannelGroup Index.
2- Pointer to DataBuffer

Errors: If DET (Development Error Tracer module) is implemented
1-ADC_E_PARAM_GROUP
2- ADC_E_UNINIT

Group Status:

Adc_StatusType Adc_GetGroupStatus(Adc_GroupType Group);
Returns the conversion status of the requested ADC Channel group.

Parameters: Adc_ChannelGroup index.

Errors: If DET (Development Error Tracer module) is implemented
1- ADC_E_PARAM_GROUP
2- ADC_E_UNINIT

Return Value: Conversion status for the requested group.
ADC_IDLE: The conversion of the specified group has not been started. - No result is available.
ADC_BUSY: The conversion of the specified group has been started and is still going on. - So far no result is available.
ADC_COMPLETED: A conversion round (which is not the final one) of the specified group has been finished. - A result is available for all channels of the group.
ADC_STREAM_COMPLETED: The result buffer is completely filled - For each channel of the selected group the number of samples to be acquired is available

Enable Notification:

void Adc_EnableGroupNotification(Adc_GroupType Group); Enables the notification mechanism for the requested ADC Channel group.
Parametes: Group Index.
Errors: **Errors: If DET (Development Error Tracer module) is implemented**
ADC_E_PARAM_GROUP, ADC_E_UNINIT, ADC_E_NOTIF_CAPABILITY

Disable Notification:

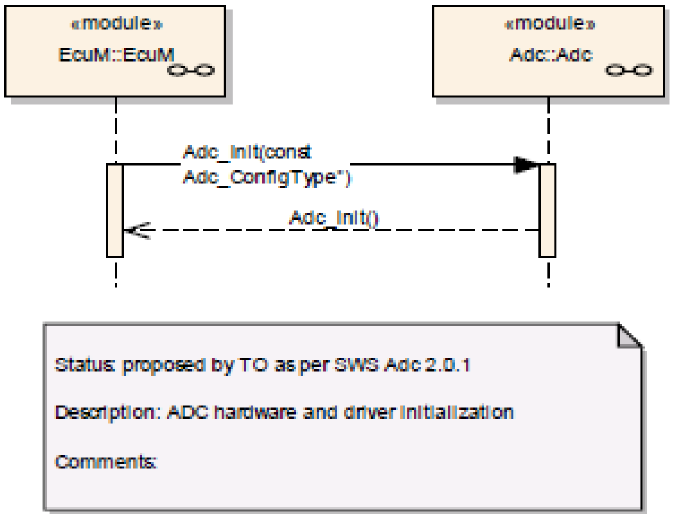
Adc_DisableGroupNotification(Adc_GroupType Group); Disables the notification mechanism for the requested ADC Channel group.
Parametes: Group Index.
Errors: **Errors: If DET (Development Error Tracer module) is implemented**
ADC_E_PARAM_GROUP, ADC_E_UNINIT, ADC_E_NOTIF_CAPABILITY

ADC Driver Modes:

You can refer to AUTOSAR_ADC_SWS_4.3.1 chapter 9 for more figures and examples.

Initializing the driver and provide the result buffer for each channel group.
First you have to initialize the driver using void **Adc_Init(const Adc_ConfigType* ConfigPtr)** API.
e.g. **Adc_Init(&Adc_Config);** Always pass &Adc_Config.

Then you have to setup the result buffer which will store the result of the conversion of your adc channel group using API **Std_ReturnType Adc_SetupResultBuffer(**
Adc_GroupType Group,
Adc_ValueGroupType* DataBufferPtr
);
Example: Adc_SetupResultBuffer(0, &Result_Buffer);
Note: DataBuffer should be large enough to hold the number of samples specified by user for every channel It's the responsibility of the application to make sure of the validation of the size of the data buffer.



Status: proposed by TO as per SWS Adc 2.0.1

Description: ADC hardware and driver initialization

Comments:

AUTOSAR_ADC_SWS_4.3.1 Chapter 9

Software triggered One-Shot conversion without notification:

Using APIs:

a. Adc_StartGroupConversion(Adc_GroupType Group);

b. Adc_GetGroupStatus(Adc_StatusType, Adc_GroupType); till Adc_StatusType return ADC_STREAM_COMPLETED

c. Adc_ReadGroup(Adc_GroupType, Adc_ValueGroupType*);

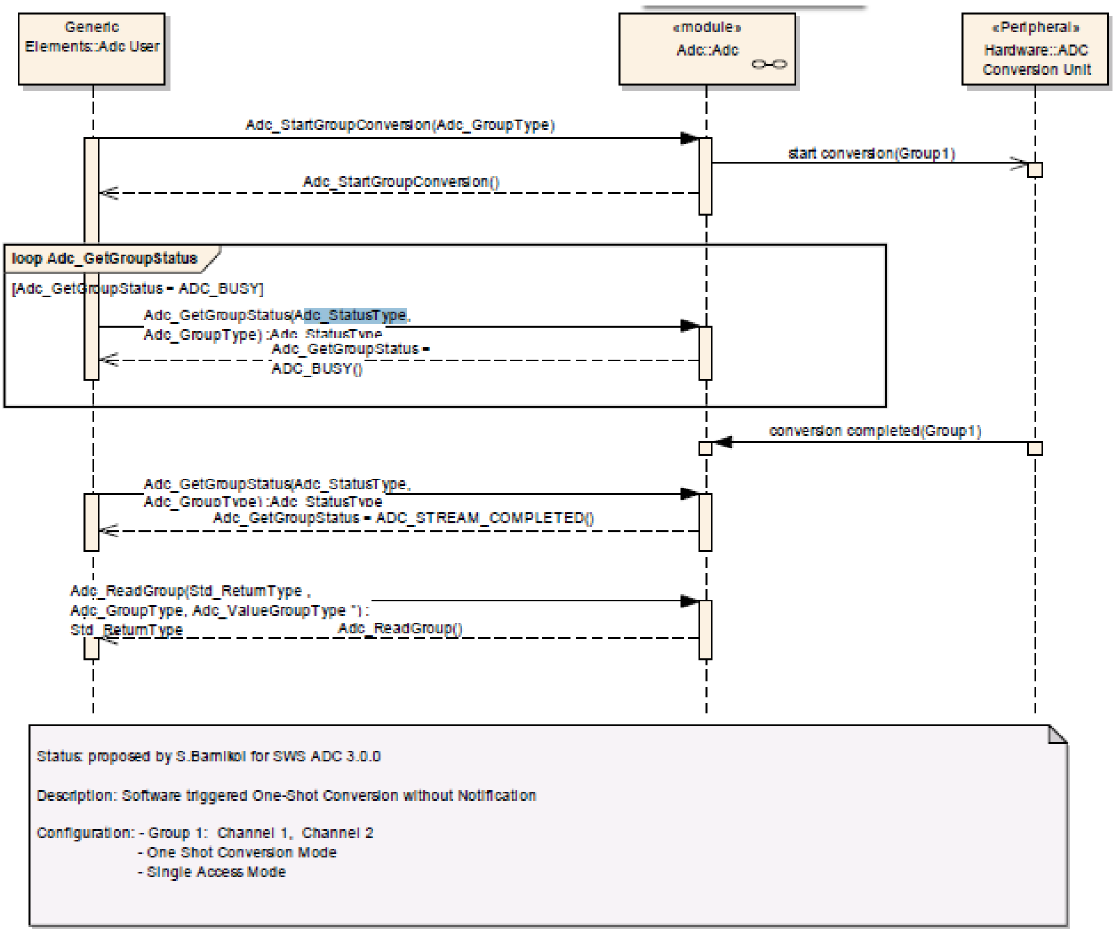

Generic Elements::Adc User
«module» Adc::Adc
«Peripheral» Hardware::ADC Conversion Unit
Adc_StartGroupConversion(Adc_GroupType)
start conversion(Group1)
Adc_StartGroupConversion()
loop Adc_GetGroupStatus
[Adc_GetGroupStatus = ADC_BUSY]
Adc_GetGroupStatus(Adc_StatusType, Adc_GroupType) :Adc_StatusType
Adc_GetGroupStatus = ADC_BUSY()
conversion completed(Group1)
Adc_GetGroupStatus(Adc_StatusType, Adc_GroupType) :Adc_StatusType
Adc_GetGroupStatus = ADC_STREAM_COMPLETED()
Adc_ReadGroup(Std_ReturnType , Adc_GroupType, Adc_ValueGroupType *) : Std_ReturnType
Adc_ReadGroup()
Status: proposed by S.Bamlkol for SWS ADC 3.0.0
Description: Software triggered One-Shot Conversion without Notification
Configuration: - Group 1: Channel 1, Channel 2
- One Shot Conversion Mode
- Single Access Mode

Figure 15: Software triggered one-shot conversion without notification

AUTOSAR_ADC_SWS_4.3.1 Chapter 9

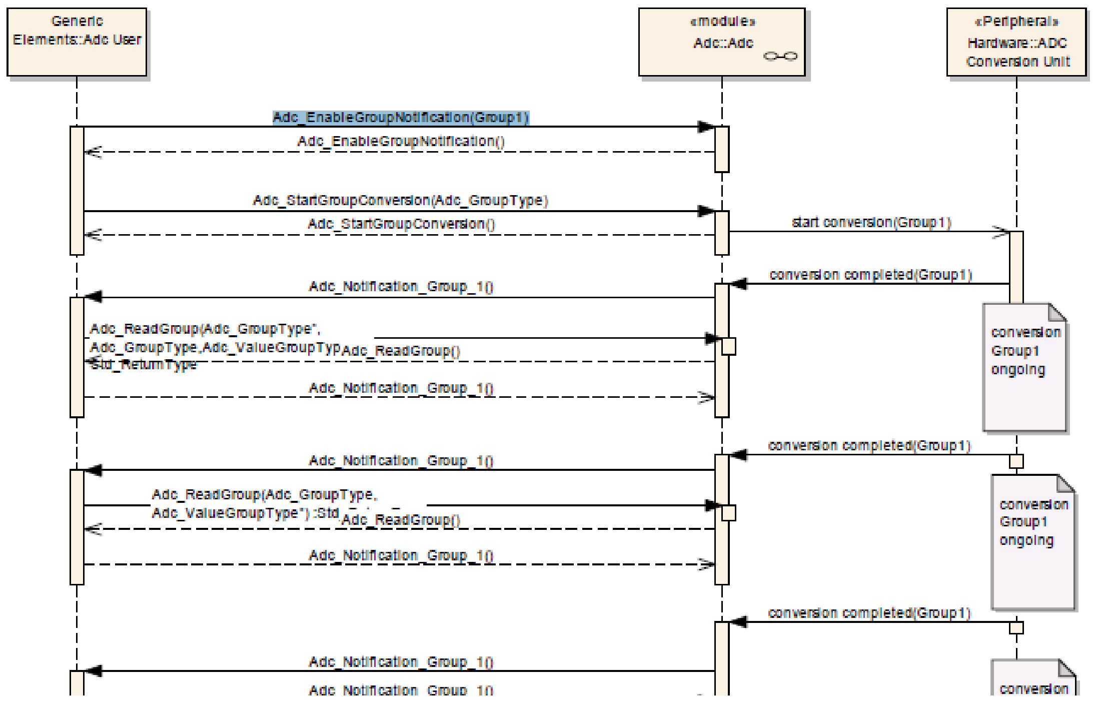
Software triggered continuous conversion with notification:

First you have to enable the Group Notification which I implemented as the enabling for the ADC interrupt for the given group, however you have to provide a pointer to function to be called as a notification that the conversion has finished.

Enable Group notification using API: Adc_EnableGroupNotification(Group);

Then start the Group Conversion using
Adc_StartGroupConversion(Group);

9.4 Software triggered continuous conversion with notification


Generic Elements::Adc User
«module» Adc::Adc
«Peripheral» Hardware::ADC Conversion Unit
Adc_EnableGroupNotification(Group1)
Adc_EnableGroupNotification()
Adc_StartGroupConversion(Adc_GroupType)
Adc_StartGroupConversion()
start conversion(Group1)
conversion completed(Group1)
Adc_Notification_Group_1()
Adc_ReadGroup(Adc_GroupType*, Adc_GroupType,Adc_ValueGroupTyp Std_ReturnType
Adc_ReadGroup()
Adc_Notification_Group_1()
conversion Group1 ongoing
conversion completed(Group1)
Adc_Notification_Group_1()
Adc_ReadGroup(Adc_GroupType, Adc_ValueGroupType*) :Std
Adc_ReadGroup()
Adc_Notification_Group_1()
conversion Group1 ongoing
conversion completed(Group1)
Adc_Notification_Group_1()
Adc_Notification_Group_1()
conversion

AUTOSAR_ADC_SWS_4.3.1 Chapter 9

Hardware trigger:

Hardware trigger is not yet implemented as I don't need it in my graduation project anyone can implement it and add it to the project.

Priority mechanism:

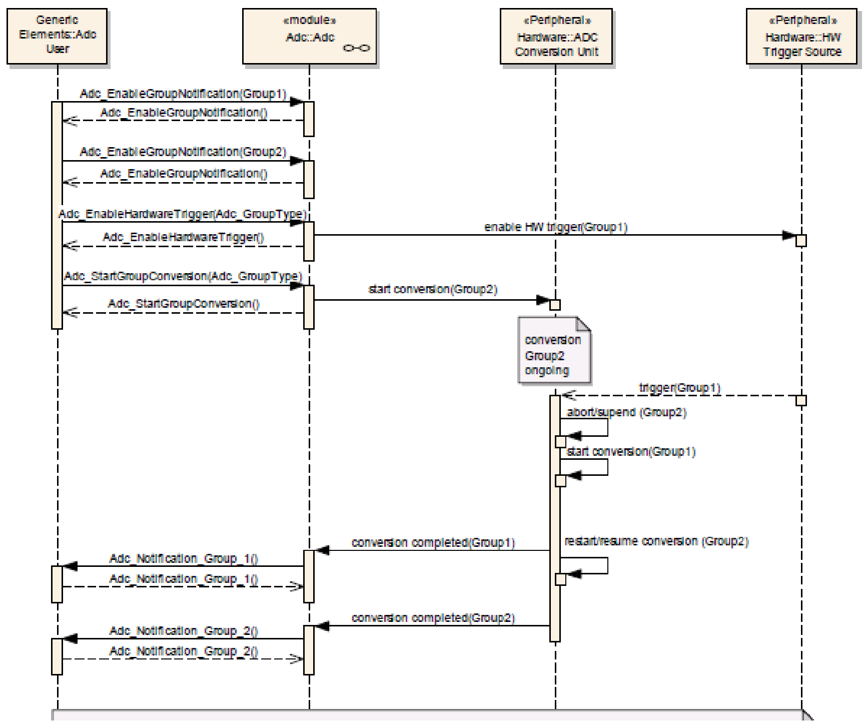
HW priority mechanism the priority mechanism that used as it's already implemented on the TM4C123GH6PM.

Every Adc_ChannelGroup is contained within one HWUnit, Every module has 4 sequencers 4 HWUnit each HWUnit takes a priority from 0 to 3 and every HWunit have to take different value or a strange behavior will be encountered.

If a higher priority group interrupted the conversion of a lower group the higher group will be served then the lower priority group conversion will be resumed.

9.10 HW Priority Mechanism – HW Queuing

More Tools


Generic Elements::Adc User
«module» Adc::Adc
«Peripheral» Hardware::ADC Conversion Unit
«Peripheral» Hardware::HW Trigger Source
Adc_EnableGroupNotification(Group1)
Adc_EnableGroupNotification()
Adc_EnableGroupNotification(Group2)
Adc_EnableGroupNotification()
Adc_EnableHardwareTrigger(Adc_GroupType)
Adc_EnableHardwareTrigger()
enable HW trigger(Group1)
Adc_StartGroupConversion(Adc_GroupType)
Adc_StartGroupConversion()
start conversion(Group2)
conversion Group2 ongoing
trigger(Group1)
abort/supend (Group2)
start conversion(Group1)
conversion completed(Group1)
restart/resume conversion (Group2)
Adc_Notification_Group_1()
Adc_Notification_Group_1()
conversion completed(Group2)
Adc_Notification_Group_2()
Adc_Notification_Group_2()

AUTOSAR_ADC_SWS_4.3.1 Chapter 9

ADC driver Configuration:

Adc_Cfg.h header file:

Adc_Cfg.h contains the static configuration which will shape your driver e.g. you can remove or add APIs just change STD_ON to STD_OFF or STD_OFF by STD_ON, also you specify the number of **Adc_ChannelGroups which has a maximum of eight channel group** besides the priority of each sample sequencer inside the same module (0→3) which is the hardware priority of the Adc_Channelgroup also you specify the interrupt priority for all sample sequencer inside the whole controller (0→7).

First you specify the total number of ADC_ChannelGroup by changing the value in front of
#define NUMBER_OF_CHANNEL_GROUPS (put the value here a value from 1 to 8 will be valid)

changing the priority of the sample sequencers which in turn affects the priority of the Adc_ChannelGroup assigned to them by changing the number in front of
#define ADC_0_SS_0_MODULE_PRI (put the priority here a value from 0 to 3 will be valid).

You change the interrupt service priority from the sample sequencers by changing the number in front of
#define ADC_0_SS0_HANDLER_PRI (A value from 0 to 7 will be considered valid where 0 is the highest priority)

Finally you can use or discard any API that has this feature by using STD_ON for add and STD_OFF for removing mainly this is used to control the size of code where there is no need to all features that is provided by the standard for example:

#define ADC_READ_GROUP_API STD_ON

This adds the API to the driver if you changed it to STD_OFF make it unavailable and removes its entire implementation to save memory space.

There is one last thing which is
#define ADC_PRIORITY_IMPLEMENTATION ADC_PRIORITY_HW
and this is used to specify the priority mechanism in case you’re using multiple ADC channel group at the same time there are three options you can choose one from.
ADC_PRIORITY_HW uses the ADC hardware features for prioritization of the software conversion requests and hardware trigger signals for groups with trigger source hardware.
The mixed hardware and software prioritization mechanism (ADC_PRIORITY_HW_SW) uses the ADC hardware features for prioritization of ADC hardware trigger for groups with trigger source hardware and a software implemented prioritization mechanism for groups with trigger source software.
Although hardware priority is implemented only and configured as mentioned previously within the same file Adc_Cfg.h.

Adc_PBCfg.c Source File:

This file contains the main blocks that this driver consists of, I will try to explain them briefly and gives a detailed example in configuring the driver.

Adc_ChannelGroupConfig:

Array of type Adc_ChannelGroupConfigType structure used in defining the Adc_channelGroups properties used within your program. It has eleven elements you've to fill them all let's start one by one:

1- Group Id: you have to give the Adc_ChannelGroup an ID which preferable to be the index of the structure within the array of structure (.Adc_ChannelGroupConfigType).

2- Group Access mode: has two options single access ADC_ACCESS_MODE_SINGLE or streaming access ADC_ACCESS_MODE_STREAMING,
ADC_ACCESS_MODE_SINGLE used with single shot mode.
While streaming access is used with continuous and hardware trigger mode.

3- Group Conversion mode:
ADC_CONV_MODE_ONESHOT → every time you want to start conversion you'll call Adc_StartGroupConversion(Group) API.
ADC_CONV_MODE_CONTINUOUS → Adc_StartGroupConversion is called once.
Options: ADC_CONV_MODE_ONESHOT, ADC_CONV_MODE_CONTINUOUS.

4- Adc_GroupPriority: It's used for the SW priority mechanism which isn't supported by this driver.

5- AdcGroupTriggSrc: ADC_TRIGG_SRC_SW using API Adc_StartGroupConv().
ADC_TRIGG_SRC_HW using HW trigger and there are various options for HW trigger.
HW trigger is implemented but not tested, SW trigger implemented and tested with multiple ADC_ChannelGroup.
Options: ADC_TRIGG_SRC_SW, ADC_TRIGG_SRC_HW.

6- Adc_HwTriggerSource: Options to choose from if ADC_TRIGG_SRS_HW is choosed in AdcGroupTriggSrc.
Options: ANALOG_COMPARATOR_0, ANALOG_COMPARATOR_1, EXTERNAL, TIMER, PWM_GENERATOR_0, PWM_GENERATOR_1, PWM_GENERATOR_2, PWM_GENERATOR_3.

7- Adc_HwTriggerSignal: Type for configuring on which edge of the hardware trigger signal the driver should start conversion.
Options: ADC_HW_TRIG_RISING_EDGE, ADC_HW_TRIG_FALLING_EDGE and ADC_HW_TRIG_BOTH_EDGES.

8- Adc_StreamBufferMode: StreamBufferMode Linear buffer mode is supported by this driver, **circular is not implemented.**
Options: ADC_STREAM_BUFFER_LINEAR, ADC_STREAM_BUFFER_CIRCULAR.

9- AdcStreamingNumSamples: Number of samples for every channel in the group before the group state become ADC_STREAM_COMPLETED.

Note: this element should be one in ADC_CONV_MODE_ONESHOT.

10- AdcNotification: Pointer to function will be called (If Adc_EnableGroupNotification() has been called) When the conversion of the Adc_ChannelGroup finishs.
Default value: NULL.

11- Adc_GroupReplacement: used in SW priority mechanism (Not Supported).
Options: ADC_GROUP_REPL_ABORT_RESTART , ADC_GROUP_REPL_SUSPEND_RESUME.

12- AdcGroupDefinition[ADC_MAXIMUM_CHANNELS_PER_GROUP]; here you give the list of input pins for this ADC_ChannelGroup starting from ADC_CHANNEL_0 to ADC_CHANNEL_11.
where

ADC_CHANNEL_0	PE3
ADC_CHANNEL_1	PE2
ADC_CHANNEL_2	PE1
ADC_CHANNEL_3	PE0
ADC_CHANNEL_4	PD3
ADC_CHANNEL_5	PD2
ADC_CHANNEL_6	PD1
ADC_CHANNEL_7	PD0
ADC_CHANNEL_8	PE5
ADC_CHANNEL_9	PE4
ADC_CHANNEL_10	PB4
ADC_CHANNEL_11	PB5

<u>Example of configuring Adc_ChannelGroupConfig:</u>

```
Adc_ChannelGroupConfigType Adc_ChannelGroupConfig[NUMBER_OF_CHANNEL_GROUPS]=
{
   {
0,                                                /* Group_Id */
ADC_ACCESS_MODE_SINGLE,   /* Group_Access_Mode */
ADC_CONV_MODE_ONESHOT,            /* Group_Conv_Mode           */
0,
ADC_TRIGG_SRC_SW,                 /* Group Event Trigger */
EXTERNAL,
ADC_HW_TRIG_RISING_EDGE,
ADC_STREAM_BUFFER_CIRCULAR,       /* Software buffer mode */
1,
NULL_PTR,
ADC_GROUP_REPL_ABORT_RESTART,
{
ADC_CHANNEL_0
}};
```

You can copy and paste this example and modify it as you want and use it in your code

Adc_HwUnitConfig:

An array of structures of type Adc_HwUnitConfigType of constant size eight you only need to modify three members, if you want to configure and use the current HWUnit you change The forth member from FALSE to TRUE

Second you enter the number of input pins in the Adc_ChannelGroup.

Third you add the Index of The Adc_ChannelGroup in Adc_ChannelGroupConfig array of structure.

Example of configuring the Adc_HwUnitConfig:

This example show using HWUnit0, 1, 2, 3 with four Adc_ChannelGroup with indexes 0, 1, 2, 3 and one input pins in each Adc_ChannelGroup.

```
Adc_HwUnitConfigType Adc_HwUnitConfig[NUMBER_OF_HW_UNITS]=
{
  {
          0, /* Numeric ID of the HW Unit. */
         ADC_MODULE_0,      /*  Numeric Id of the Adc_Module */
         ADC_SEQUENCER_0, /* Numberic Id of the Adc_Sequencer */
          TRUE, /* True: use Adc_Sequencer false: don't configure */
        /* Hold the number of input channels per hw unit*/
        /* maximum for this hw unit is 8 */
        1,
        0          /* Index of the Adc_ChannelGroup*/
       },
       {
        1, /* Numeric ID of the HW Unit. */
        ADC_MODULE_0,        /*  Numeric Id of the Adc_Module */
        ADC_SEQUENCER_1, /* Numberic Id of the Adc_Sequencer */
        TRUE, /* True: use Adc_Sequencer false: don't configure */
        /* Hold the number of input channels per hw unit*/
        /* maximum for this hw unit is 4 */
        1,
        1          /* Index of the Adc_ChannelGroup*/
       },
       {
        2, /* Numeric ID of the HW Unit. */
        ADC_MODULE_0,        /*  Numeric Id of the Adc_Module */
        ADC_SEQUENCER_2, /* Numberic Id of the Adc_Sequencer */
        TRUE, /* True: use Adc_Sequencer false: don't configure */
        /* Hold the number of input channels per hw unit*/
        /* maximum for this hw unit is 4 */
        1,
        2          /* Index of the Adc_ChannelGroup*/
       },
       {
        3, /* Numeric ID of the HW Unit. */
        ADC_MODULE_0,        /*  Numeric Id of the Adc_Module */
        ADC_SEQUENCER_3, /* Numberic Id of the Adc_Sequencer */
        TRUE, /* True: use Adc_Sequencer false: don't configure */
        /* Hold the number of input channels per hw unit*/
        /* maximum for this hw unit is 1 */
        1,
               3          /* Index of the Adc_ChannelGroup*/
```

```
	},
	{
	 4, /* Numeric ID of the HW Unit. */
	 ADC_MODULE_1,		/*  Numeric Id of the Adc_Module */
	 ADC_SEQUENCER_0, /* Numberic Id of the Adc_Sequencer */
	 FALSE, /* True: use Adc_Sequencer false: don't configure */
	 /* Hold the number of input channels per hw unit*/
	 /* maximum for this hw unit is 8 */
	 0,
	 255	/* Index of the Adc_ChannelGroup*/
	},
	{
	 5, /* Numeric ID of the HW Unit. */
	 ADC_MODULE_1,		/*  Numeric Id of the Adc_Module */
	 ADC_SEQUENCER_1, /* Numberic Id of the Adc_Sequencer */
	 FALSE, /* True: use Adc_Sequencer false: don't configure */
	 /* Hold the number of input channels per hw unit*/
	 /* maximum for this hw unit is 4 */
	 0,
	 255	/* Index of the Adc_ChannelGroup*/
	},
	{
	 6, /* Numeric ID of the HW Unit. */
	 ADC_MODULE_1,		/*  Numeric Id of the Adc_Module */
	 ADC_SEQUENCER_2, /* Numberic Id of the Adc_Sequencer */
	 FALSE, /* True: use Adc_Sequencer false: don't configure */
	 /* Hold the number of input channels per hw unit*/
	 /* maximum for this hw unit is 4 */
	 0,
	 255 /* Index of the Adc_ChannelGroup*/
	},
	{
	 7, /* Numeric ID of the HW Unit. */
	 ADC_MODULE_1,		/*  Numeric Id of the Adc_Module */
	 ADC_SEQUENCER_3, /* Numberic Id of the Adc_Sequencer */
	 FALSE, /* True: use Adc_Sequencer false: don't configure */
	 /* Hold the number of input channels per hw unit*/
	 /* maximum for this hw unit is 1 */
	 1,
	 255/* Index of the Adc_ChannelGroup*/
	},
};
```

Adc_Config

Adc_Config is a structure of type Adc_ConfigType has five members uses to choose which ADC module you will use 0, 1 or both and specify the sampling speed of the module.

Example:

```
const Adc_ConfigType Adc_Config=
{
TRUE, /* True: Configure and enable clock for module 1, False: Don't configure */
ADC_500_K_SAMPLING_SPEED,		/* 500 kilo sample per second. */
TRUE, /* True: Configure and enable clock for module 1, False: Don't configure */
ADC_500_K_SAMPLING_SPEED,
&Adc_HwUnitConfig[0]
};
```