# S32K3XX
# SAR-ADC, BCTU AND TRGMUX

Automotive Systems & Applications Engineering Team
GPIS-BL, Automotive Processing

**FEB 2024**

SECURE CONNECTIONS
FOR A SMARTER WORLD


**PUBLIC**

# AGENDA

1. ADC overview
2. Body Cross-triggering Unit (BCTU)
3. Trigger Multiplexer
4. RTD MCAL Example
5. RTD Low Level Drivers Example

# ADC Overview

SECURE CONNECTIONS
FOR A SMARTER WORLD

# ADC – OVERVIEW

| Instance | S32K388/S32K358/S32K348/S32K338/ S32K328/S32K344/S32K324/S32K314 | S32K342/S32K322/S32K341/S32K312/ S32K311 |
|---|---|---|
| ADC_0 | YES | YES |
| ADC_1 | YES | YES |
| ADC_2 | YES | NO |

This chip has up to 3 instances of ADC. ADC channels are divided into 3 groups - Precision, Standard and External:

| Feature | ADC_0 | ADC_1 | ADC_2 |
|---|---|---|---|
| No. of precision channels | 8 | 8 | 8 |
| No. of standard channels | 16 | 16 | 16 |
| No. of special internal channels | 1 | 0 | 0 |
| No. of external channels | 32 | 32 | 0 |
| BCTU trigger support | YES | YES | YES |
| DMA support | YES | YES | YES |
| Hardware interleaving | YES | YES | YES |
| No. of watchdogs | 4 | 4 | 4 |
| No. of Hardware trigger | 3 | 3 | 3 |

# S32K3 ADC FEATURES OVERVIEW

- 8/10/12/14-bit resolution (conversion result is always 15 bit wide)
- Sampling and conversion time: ~1uS (80 MHz conversion clock) Up to 1M samples/sec
- Accuracy: TUE (+/- 6LSB)
- Three conversion modes:
  - Normal conversion supports one-shot and scan conversions
  - Injected conversion supports one-shot conversion only
  - BCTU conversion supports single and list conversions
- Independent data registers per channel
- Programmable DMA enable per channel
- Programmable pre-sampling enable per channel
- Configurable analog watchdogs for different channels
- Individual interrupt flags for the following conditions:
  - Single channel, chain or BCTU end-of-conversion
  - Watchdog thresholds violation
- Software initiated calibration
- Hardware interleaving
- Hardware average function
- Self-Test function

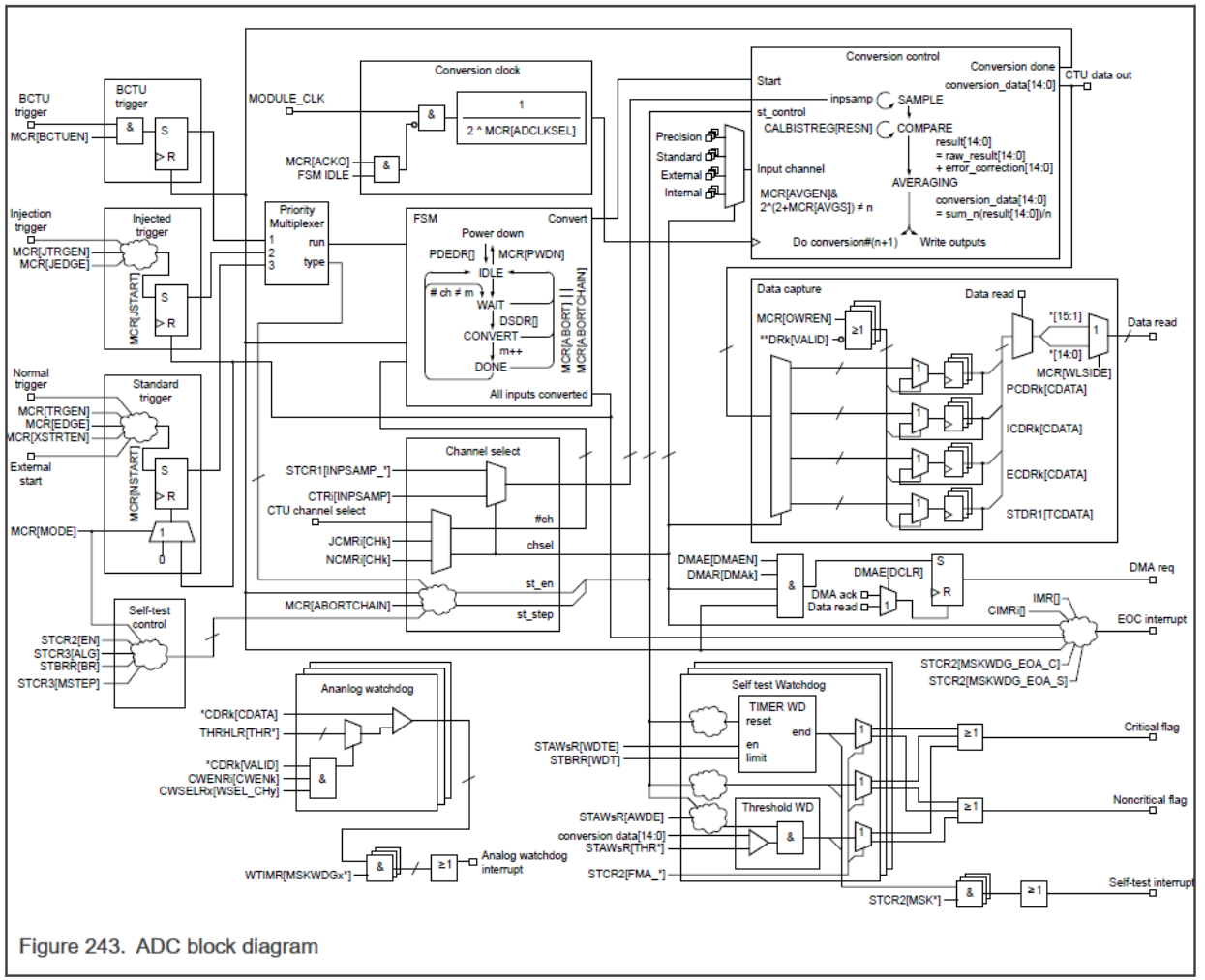


Figure 243. ADC block diagram

- SAR-ADC doesn’t work in the chip STANDBY mode
- In the chip RUN mode:
  - IDLE ( MCR[PWDN] = 0, MSR[ADCSTATUS] = 000b )
    - ADC is waiting for the conversion triggers (software, hardware, BCTU)
  - Power-down ( MCR[PWDN] = 1, MSR[ADCSTATUS] = 001b )
    - This is the default mode out of reset
  - Wait ( MCR[PWDN] = 0, MSR[ADCSTATUS] = 010b )
    - Pausing for settling time of the external analog multiplexer
  - Calibrate ( MCR[PWDN] = 0, MSR[ADCSTATUS] = 011b )
    - Calibration is in progress
  - Convert ( MCR[PWDN] = 0, MSR[ADCSTATUS] = 100b )
    - SAR conversion is in progress
  - Done ( MCR[PWDN] = 0, MSR[ADCSTATUS] = 110b )
    - SAR conversion completed

# ADC CHANNEL MAPPING

| Channel Type | Control Registers [bits] | Pin Names | Result Registers |
|---|---|---|---|
| “Precision” | N/JCMR0 [7:0] | ADCn_P[7:0] | PCDR7 – PCDR0 |
| | N/JCMR0 [9:8] | n/a | PCDR9 – PCDR8 (not used) * |
| | N/JCMR0 [31:10] | n/a | n/a |
| “Standard” or “Internal” | N/JCMR1 [23:0] | ADCn_S[23:8] | ICDR55 – ICDR32 |
| | N/JCMR1 [31:24] | n/a | ICDR63 – ICDR56 (not used) * |
| “External” | N/JCMR2 [31:0] | ADCn_X[3:0] & ADCn_MA[2:0] | ECDR95 – ECDR64 |

**ADC Pins mapped per register**

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NCMR0, JCMR0 | | | | | | | | | | | | | | | | | | | | | | | | | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 | ADCn_P[7:0] |
| NCMR1, JCMR1 | | | | | | | | | S31 | S30 | S29 | S28 | S27 | S26 | S25 | S24 | S23 | S22 | S21 | S20 | S19 | S18 | S17 | S16 | S15 | S14 | S13 | S12 | S11 | S10 | S9 | S8 | ADCn_S[23:8] |
| NCMR2, JCMR2 | X3 | | | | | | | | X2 | | | | | | | | X1 | | | | | | | | X0 | | | | | | | | ADCn_X[3:0] |
| *** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ADCn_MA[2:0] |

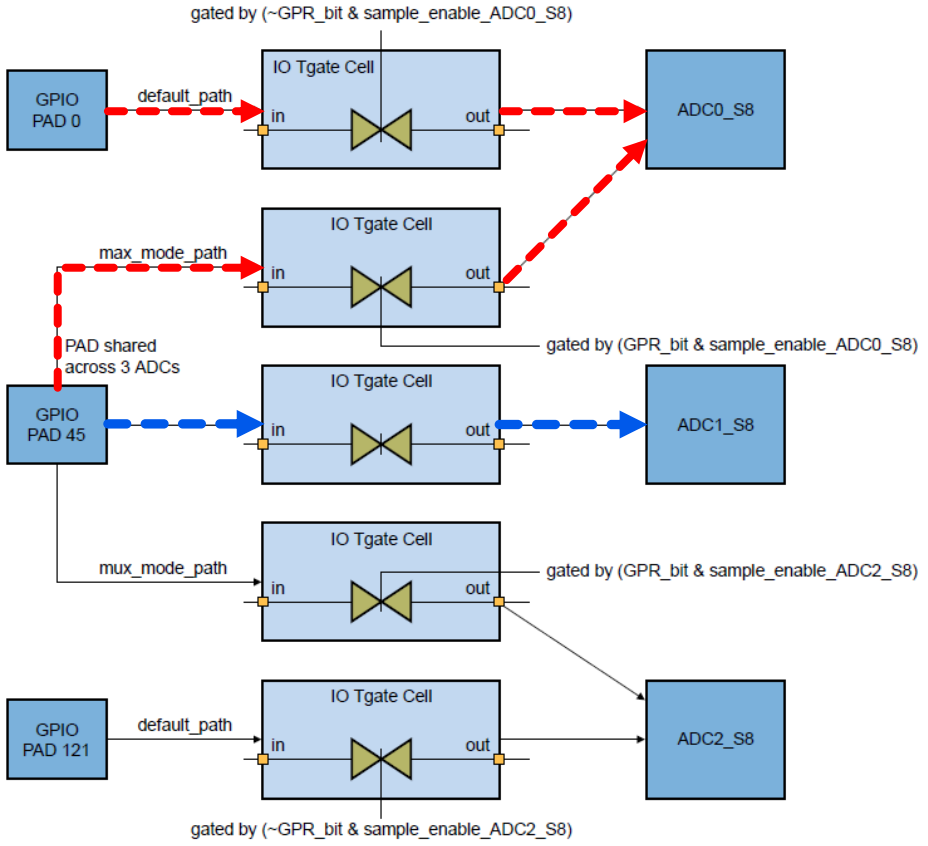
# EXTERNAL CHANNELS

- Each ADC provides 3 external decode signals (MA) to be used to select 1 channel out of 8 in the external analog multiplexers.
- There can be maximum 4 of such multiplexers to connect 32 external channels.
- The ADC automatically sets the decode signals (MA) to control these external analog multiplexers, based on the current channel selected for conversion.
- Depending on the Mask Register bits, the corresponding "X" pin is sampled, and the result is stored in the matching location for the "MA" & "X" combination.
- The Decode Signal Delay (DSD) register is provided to program the delay between the decoding signal (MA) selection and the actual start of conversion.
- **Note**: ADCx_MA is gray-encoded, so it counts in the order of 0, 1, 3, 2, 6, 7, 5, 4.

| Enable Register | Data Register | Input pin | "MA" encoding |
|---|---|---|---|
| N/JCMR2 [7:0] | CDR71 – CDR64 | ADCx_X0 | ADCx_MA[2:0] = (4, 5, 7, 6, 2, 3, 1, 0) |
| N/JCMR2 [15:8] | CDR79 – CDR72 | ADCx_X1 | ADCx_MA[2:0] = (4, 5, 7, 6, 2, 3, 1, 0) |
| N/JCMR2 [23:16] | CDR87 – CDR80 | ADCx_X2 | ADCx_MA[2:0] = (4, 5, 7, 6, 2, 3, 1, 0) |
| N/JCMR2 [31:24] | CDR95 – CDR88 | ADCx_X3 | ADCx_MA[2:0] = (4, 5, 7, 6, 2, 3, 1, 0) |

- There are some channels in SAR_ADC which are driven by multiple pads.
- GPR bits are used to distinguish between default and mux_mode path for a particular channel.

| No. | Channel | GPR bit | Description | |
|---|---|---|---|---|
| 1 | ADC0_S8 | DCM.DCMRWF4[1] | 0: GPIO_PAD0 | 1: GPIO_PAD45 |
| 2 | ADC0_S9 | DCM.DCMRWF4[2] | 0: GPIO_PAD1 | 1: GPIO_PAD46 |
| 3 | ADC1_S14 | DCM.DCMRWF4[3] | 0: GPIO_PAD47 | 1: GPIO_PAD32 |
| 4 | ADC1_S15 | DCM.DCMRWF4[4] | 0: GPIO_PAD48 | 1: GPIO_PAD33 |
| 5 | ADC1_S16 | DCM.DCMRWF4[5] | 0: GPIO_PAD71 | 1: GPIO_PAD114 |
| 6 | ADC1_S17 | DCM.DCMRWF4[6] | 0: GPIO_PAD3 | 1: GPIO_PAD115 |
| 7 | ADC1_S25 | DCM.DCMRWF4[7] | 0: GPIO_PAD147 | 1: GPIO_PAD145 |
| 8 | ADC1_S26 | DCM.DCMRWF4[8] | 0: GPIO_PAD148 | 1: GPIO_PAD146 |
| 9 | ADC2_S8 | DCM.DCMRWF4[9] | 0: GPIO_PAD121 | 1: GPIO_PAD45 |
| 10 | ADC2_S9 | DCM.DCMRWF4[10] | 0: GPIO_PAD203 | 1: GPIO_PAD46 |



# ADC CONVERSION TIME

- The total conversion time depends on the conversion clock frequency (AD_clk), the conversion clock frequency is configured by MCR[ADCLKSEL] bit.

- The components of conversion time and affecting configurations are:
  - Pre-Sample phase time (PST, equal to ST with 1 AD_clk cycle delay)
  - Sample phase time (ST, configurable - the minimum value is 8 AD_clk cycle)
  - Compare phase time (CT, fixed - 4 AD_CLK cycle per bit)
  - Data Processing time (DP, fixed - 2 AD_CLK cycle)
  - Trigger Processing time (TPT, fixed – 1 AD_CLK cycle)

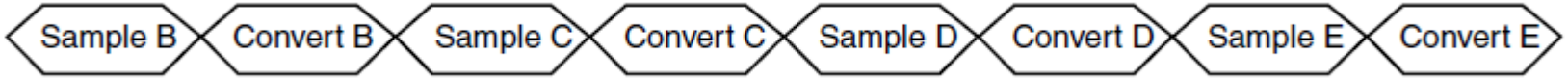
- Total_conversion_time = ([(PST + ST + CT + DP)*chain_length] + TPT) * TAD_clk

# ADC CONVERSION TIME EXAMPLE

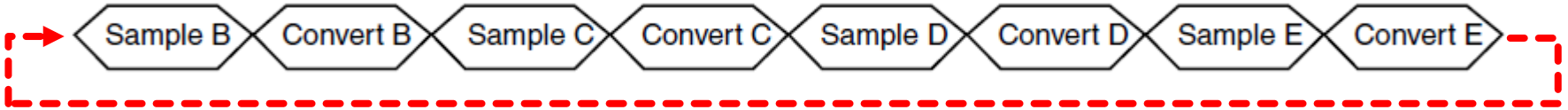
- Example:
  - The conversion clock frequency (AD_clk) is 80 MHz (T = 12.5 ns)
  - ADC resolution 12 bit + 1 bit for a special capacitor (CS)
  - 3 channels are programmed in NCMRx register (chain_length = 3)
  - No pre-sampling is selected (PST, 0 cycle)
  - Sample time kept default (ST, 22 cycle)
  - Conversion time (CT, 4 cycle per bit)
  - Data Processing time is fixed (DP, 2 cycle)
  - Trigger Processing time is fixed (TPT, 1 cycle)

- Total_conversion_time = [(0 + 22 + (4*13) + 2) * 3] + 1 = 229 cycles ~= 2.862 µs

- Start Conversion
  - After programming the Normal Conversion Mask Registers(NCMRn) and the Main Configuration Register(MCR), a conversion can be started via software by setting the NSTART bit.
  - Any conversion process involves sampling and conversion phases shown as below.

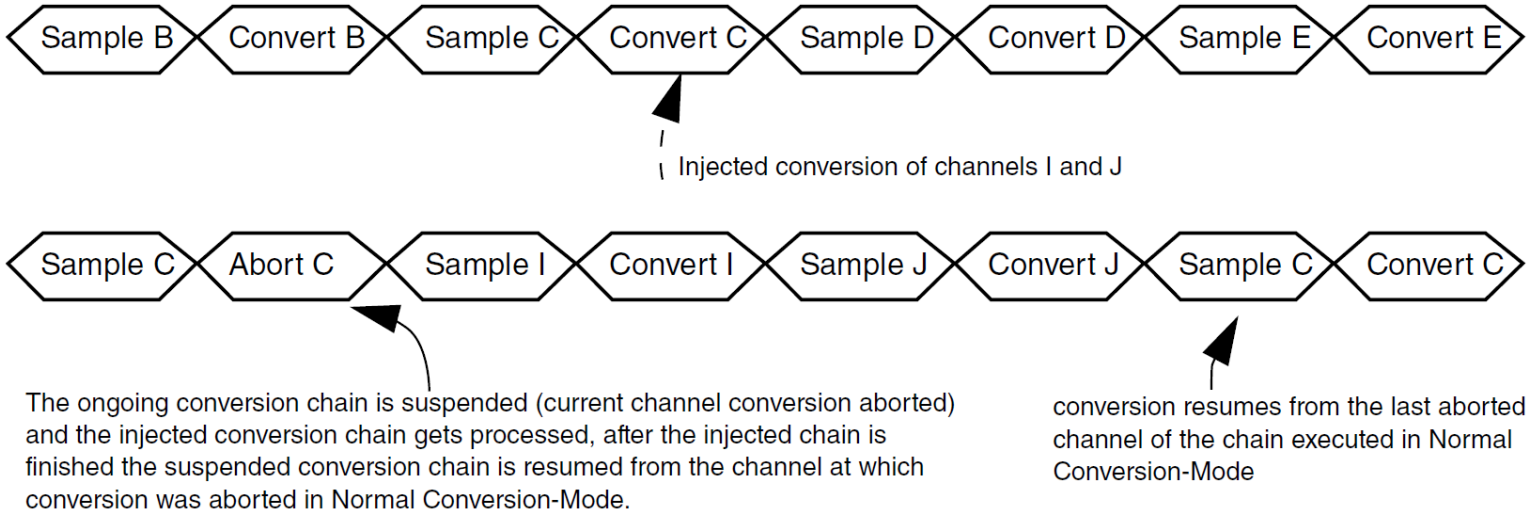



- Operation Mode
  - One-Shot operation mode(MODE = 0): a sequential conversion is performed only once.
  - Scan operation mode(MODE = 1): a sequential conversion is performed continuously shown as below.



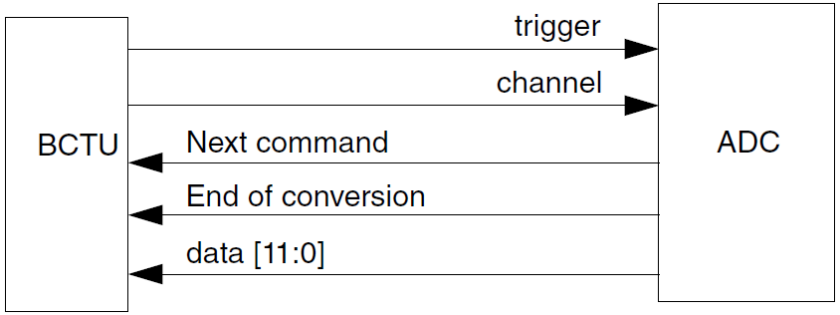

- End of Conversion
  - At the end of each conversion, the result is stored in corresponding register(PCDRn/ICDRn/ECDRn), the EOC flag is set, and End of Conversion(EOC) interrupt is issued(if enabled in MSKEOC).
  - After the last conversion in the chain is complete, the ECH flag is set, and End of Chain(ECH) interrupt is issued(if enabled in MSKECH).

- Start Conversion
  - After programming the Injected Conversion Mask Registers(JCMRn), a conversion can be started via software by setting the JSTART bit.
  - It can only be inserted into an ongoing conversion chain processed in Normal conversion mode.
  - It can only be run in One-Shot operation mode.
- End of Conversion
  - At the end of each conversion, an End of Injected Conversion(JEOC) interrupt is issued.
  - At the end of the Injected chain, an End of Injected Chain(JECH) interrupt is issued.



## • BCTU Interface

- The BCTU interface enhances the Injected conversion capability of SAR_ADC.
- The BCTU generates the trigger and channel number to be converted.
- The ADC returns the result together with 2 output signals:
  - The assertion of bctu_push means conversion is finished, and data is available.
  - The assertion of bctu_nextcmd means ADC is ready to accept next trigger.




- The BCTU and ADC are always connected to the same clock domain and therefore are synchronous.

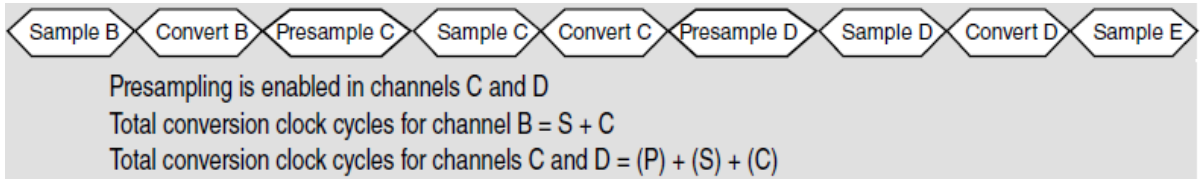
## • Priority among 3 Conversion Types

- If a BCTU conversion is requested during Injected mode, the Injected chain will be aborted immediately and only the BCTU triggered Injected conversion mode is processed.
- If a BCTU conversion is requested during Normal mode, the Normal chain will be suspended and the BCTU triggered Injected conversion mode is processed.
- If a Normal conversion is requested during BCTU mode, the Normal chain will start after the BCTU conversion is completed.
- If an Injected conversion is requested during BCTU mode, the Injected chain will be discarded immediately.

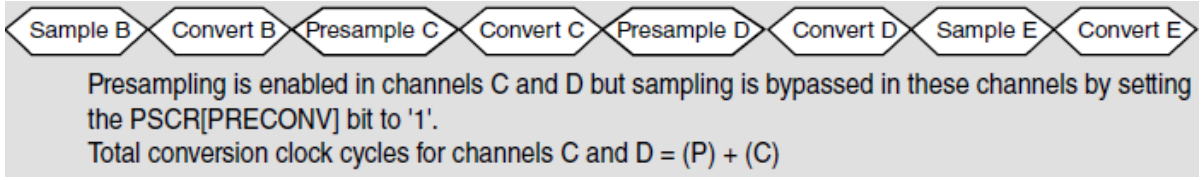
- This feature allows to precharge or discharge the SAR_ADC internal sample capacitor node to a defined level before sampling of the selected analog input channel starts.
  - Useful for avoiding a memory effect in the comparison due to a previous conversion.
- It is possible to select between 2 internally generated voltages as given in the following table.

| No. | PREVALn | Presampling voltage |
|---|---|---|
| 1 | 0 | Presample voltage is VREFL |
| 2 | 1 | Presample voltage is VREFH |

- After enabling the presampling for a channel, the sequence of operation will be Presampling+Sampling+Conversion.
- The ADC samples the internally generated voltage during presampling instead of analog input coming from pads.

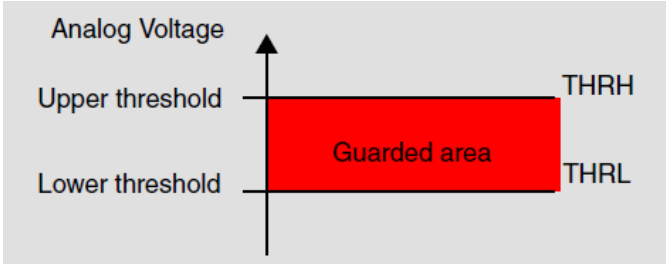



- Sampling of all channels can be bypassed by setting PRECONV bit, and then the sampled and stored internal voltage applied during pre-sampling phase will be converted.

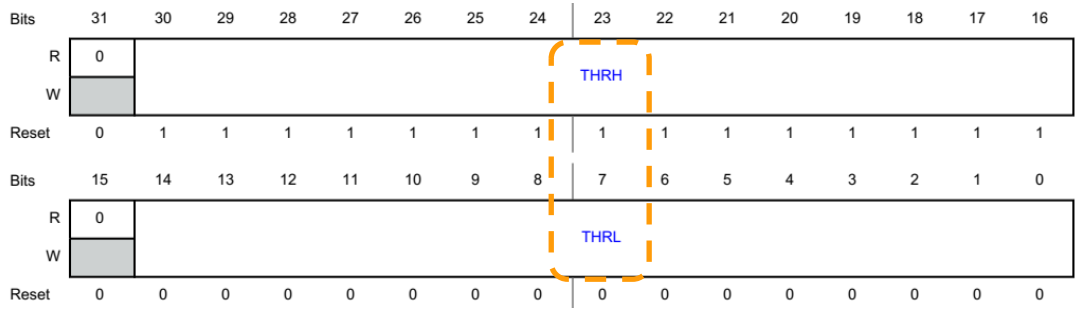


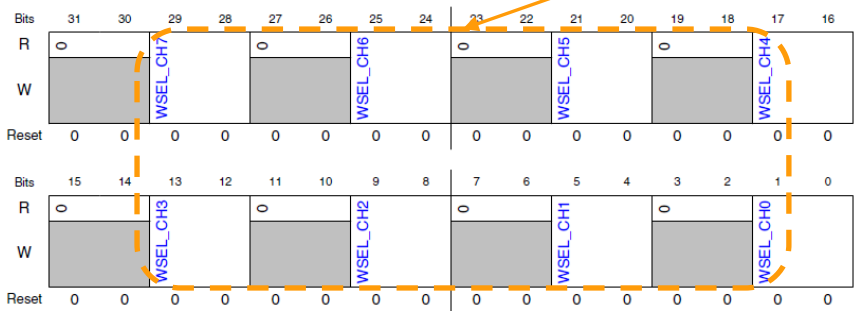
- Analog watchdogs are used to monitor the conversion result being inside defined limits specified by an upper and a lower threshold value named THRH and THRL respectively.
- The comparison result is stored as WDGxH and WDGxL fields in WTISR register as explained in the following table.

| No. | WDGxH | WDGxL | Conversion result |
|---|---|---|---|
| 1 | 1 | 0 | Conversion result > THRH |
| 2 | 0 | 1 | Conversion result < THRL |
| 3 | 0 | 0 | THRH ≥ conversion result ≥ THRL |

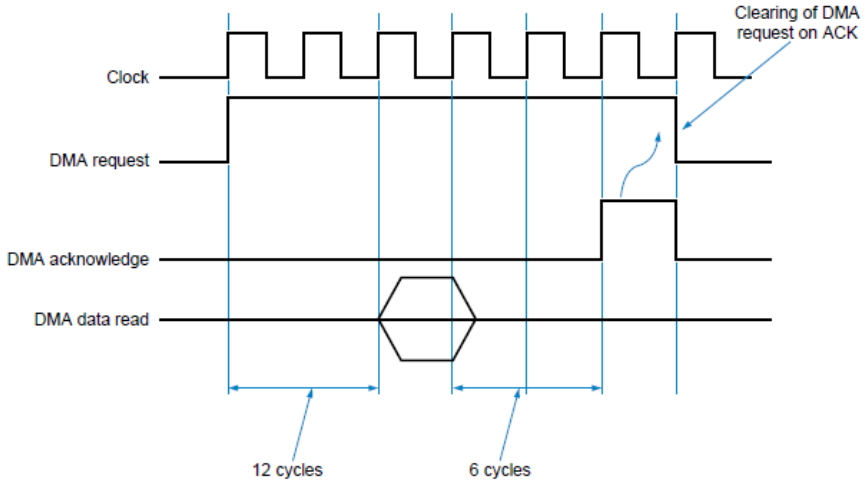



- SAR_ADC can support up to 4 sets of threshold values: THRHLR0 – THRHLR3.
- Each channel can be enabled/disabled for comparison function independently.
- Each channel can select 1 set of threshold values from 4 THRHLRn registers by programming WSEL_CHn field.

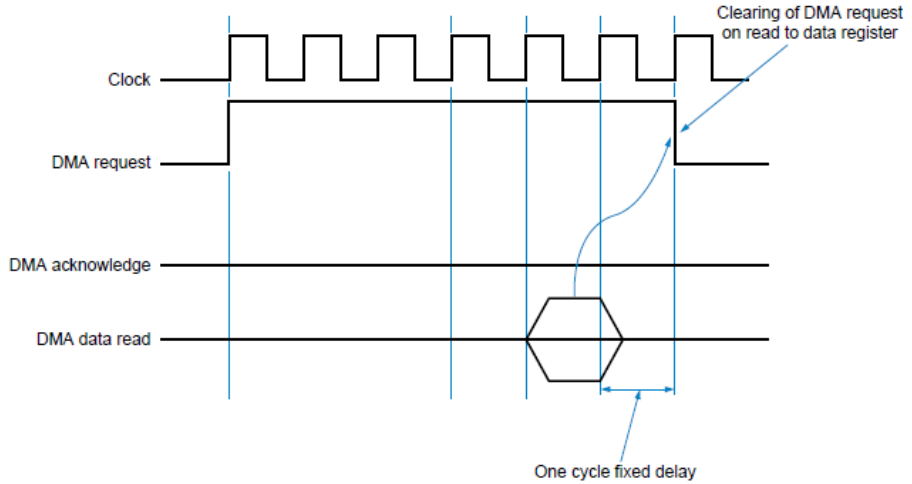


- Conversion Data of any channel can be transferred from register to system memory via DMA.
- Once enabled, the on-chip DMA controller can get DMA request after the conversion of every channel by setting the respective bit in the DMAR0|DMAR1|DMAR2 registers.
- The DMA request can be cleared at different times in 2 modes:
  - MODE-1: Clearing of DMA request on Acknowledgement from DMA controller (DMAE[DCLR] = 0)
  - MODE-2: Clearing of DMA request on Read to data registers (DMAE[DCLR] = 1)



DMA operation MODE - 1 ( DMAE [DCLR] = 0 )



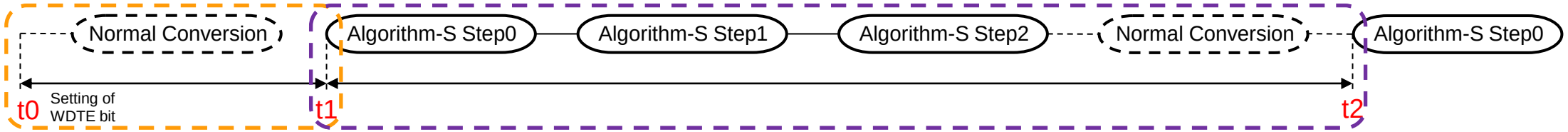
DMA operation MODE - 2 ( DMAE [DCLR] = 1 )

## • Self-Test Analog Watchdog

- The ADC digital interface also provides monitors to determine whether the result of conversion for self-test algorithms are in the range of a particular guard area.
- If the converted value does not lie between the threshold values specified by the particular algorithm, the corresponding error flag bit is set and the step number in which error occurred is updated(Algorithm C).

## • Watchdog Timer

- Safe Time Checking: Ensure the selected algorithm is started(t1-t0) and completed(t2-t1) within a safe time period.
- Sequence Checking: Monitor the steps of a particular algorithm are implemented in correct order.
  - The safe time is measured starting from Step0 of the algorithm (including all Normal chain conversions in between) to the point where Step0 of the same algorithm starts.




## • Baud Rate Control for Test Channel

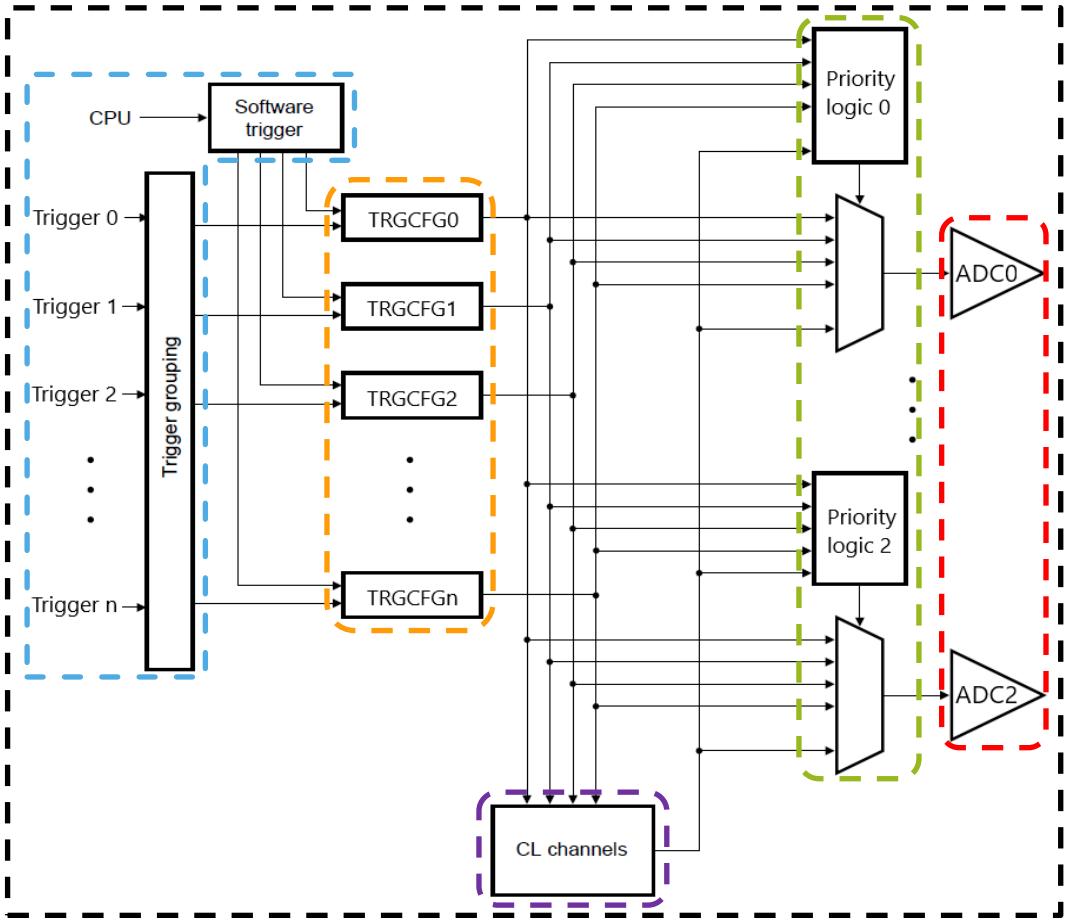
- The BR field provides flexibility by scheduling the test channel conversion to be performed not at the end of every Normal chain but at the end of BR+1 number of Normal chains.

# Body Cross-triggering Unit (BCTU)

SECURE CONNECTIONS
FOR A SMARTER WORLD

The Body Cross Triggering Unit(BCTU) is a module that is used to trigger the on-chip ADCs.

- Trigger sources
  - eMIOS timer channels
  - TRGMUX inputs
  - Software
- Trigger outputs to 3 on-chip ADCs
- 1 priority selection logic block per ADC
- A Conversion LIST of 32 ADC channels
- Up to 72 conversion configuration registers TRGCFG
- MPC(Multiple Parallel Conversions) functionality
- Optional halting of the LIST execution till next trigger
- 3 result registers for 3 ADC modules respectively
- 2 FIFOs for reading ADC results




BCTU High-level Block Diagram

- The S32K3xx chip has only 1 instance of BCTU module and 2 FIFOs:

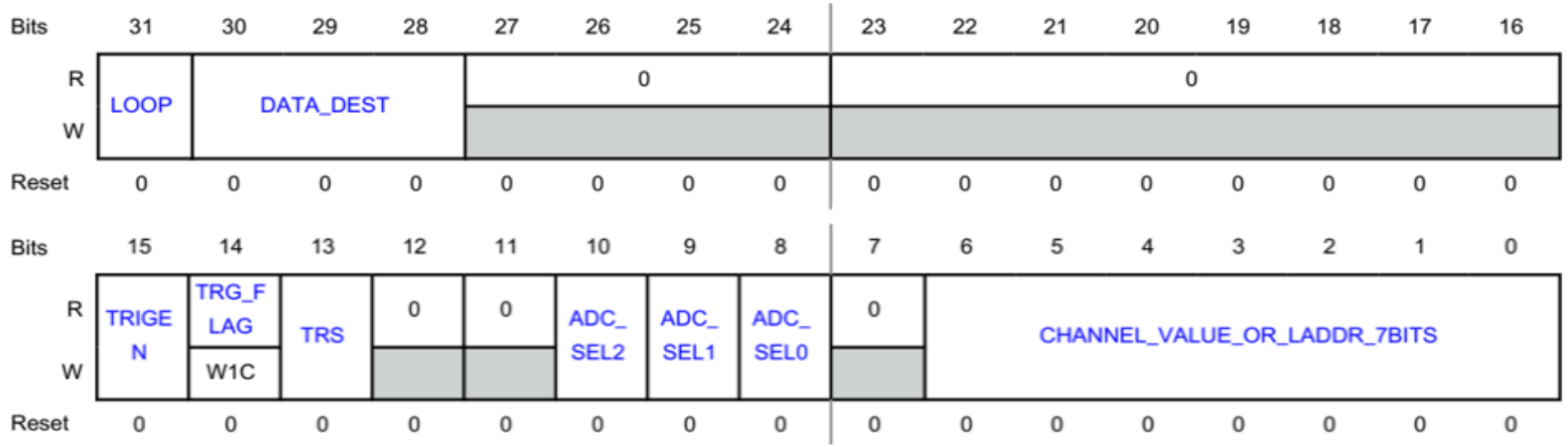
| Instance | S32K388,S32K358/S32K348/S32K338/S32K328,S32K344/ S32K324/S32K314 |
|---|---|
| BCTU | Yes |
| **Feature** | **Configuration** |
| FIFO | 16 words |
| FIFO2 | 8 words |

- BCTU has up to 72 trigger sources as listed below, 23 channels are trtigger for each eMIOS instance :

| BCTU trigger Number | Module | Source | Applicability |
|---|---|---|---|
| 0 | eMIOS_0 | Channel_0 | All |
| | ••• | | |
| 23 | Any of the TRGMUX inp | BCTU_Trg23 | All |
| 24 | eMIOS_1 | Channel_0 | All |
| | ••• | | |
| 47 | Any of the TRGMUX inp | BCTU_Trg47 | All |
| 48 | eMIOS_2 | Channel_0 | S32K388, S32K358/S32K348/S32K338/S32K328, S32K344/S32K324/S32K314 |
| | ••• | | |
| 71 | Any of the TRGMUX inp | BCTU_Trg71 | S32K388, S32K358/S32K348/S32K338/S32K328, S32K344/S32K324/S32K314 |

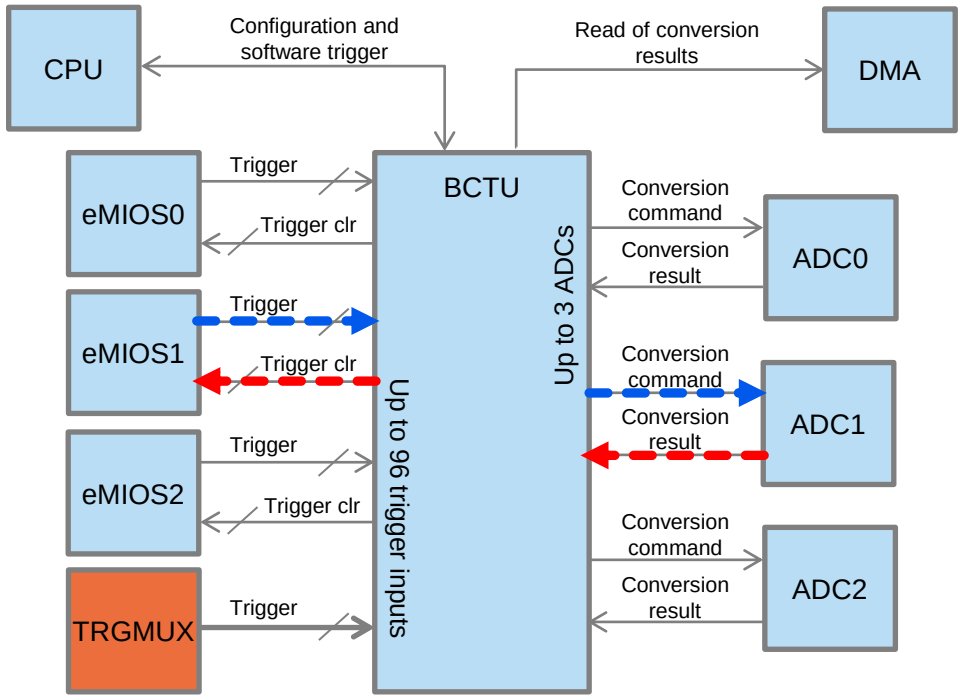
- BCTU doesn’t work in the chip STANDBY mode
- In the chip RUN mode:
  - Configuration mode ( MCR[GTRGEN] = 0 )
    - This is the default mode out of reset
    - Doesn’t generate conversion triggers to the ADCs
    - TRGCFG registers are programmed by software
    - Trigger inputs are disabled
  - Normal mode ( MCR[GTRGEN] = 1 )
    - Generate triggers for the ADCs based on trigger inputs from on chip modules
  - Low Power mode ( MCR[MDIS] = 1 )
    - All trigger inputs are disregarded, no triggers are issued to the ADCs
    - Current ongoing conversion is finished before the low power mode is entered and pending active triggers remain active (software needs to clear the pending active triggers)

# TRIGGER CONFIGURATION REGISTER (TRGCFG_0~71)




- LOOP: specifies that the current trigger executes in a loop
- DATA_DEST: route ADC results to ADC-specific Data registers or FIFO1/FIFO2
- TRIGEN: enable / disable this trigger
- TRG_FLAG: indicates an ADC conversion is ongoing / completed
- TRS: trigger single channel or a list of channels
- ADC_SELn: select ADC n for conversion
- CHANNEL_VALUE_OR_LADDR_7BITS: select a channel or the first address of LIST for conversion

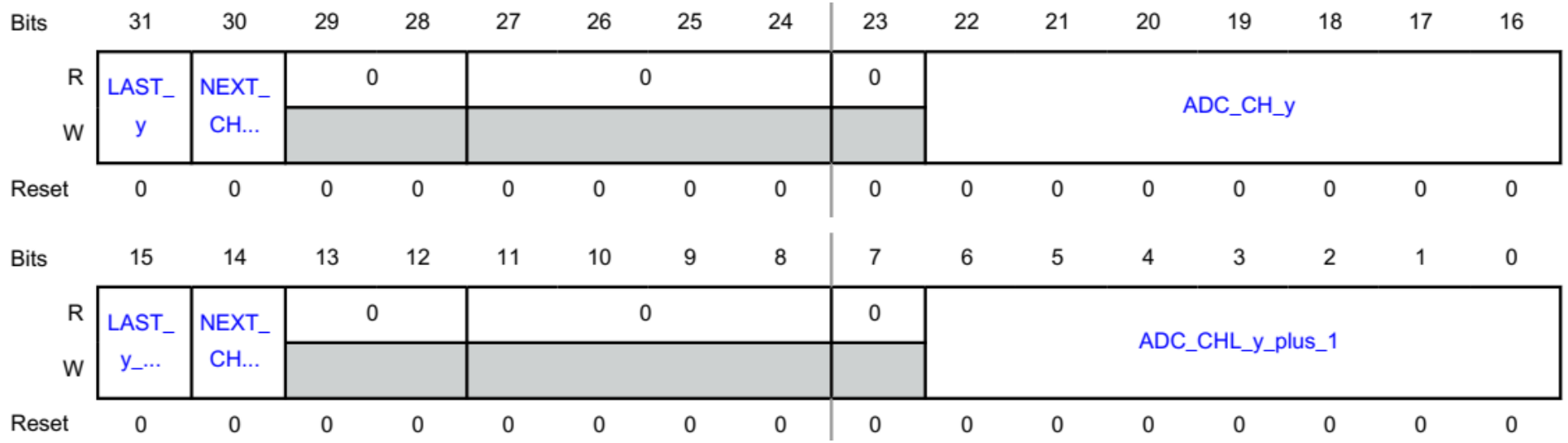
The figure below shows an example of how the BCTU interacts with other on-chip peripherals.




| Trigger Number | Module | Source |
|---|---|---|
| 0-22 | eMIOS0 | CH0-CH22 |
| 23 | TRGMUX | BCTU_Trg23 |
| 24-31 | eMIOS0 | CH24-CH31 |
| 32-54 | eMIOS1 | CH0-CH22 |
| 55 | TRGMUX | BCTU_Trg55 |
| 56-63 | eMIOS1 | CH24-CH31 |
| 64-86 | eMIOS2 | CH0-CH22 |
| 87 | TRGMUX | BCTU_Trg87 |
| 88-95 | eMIOS2 | CH24-CH31 |

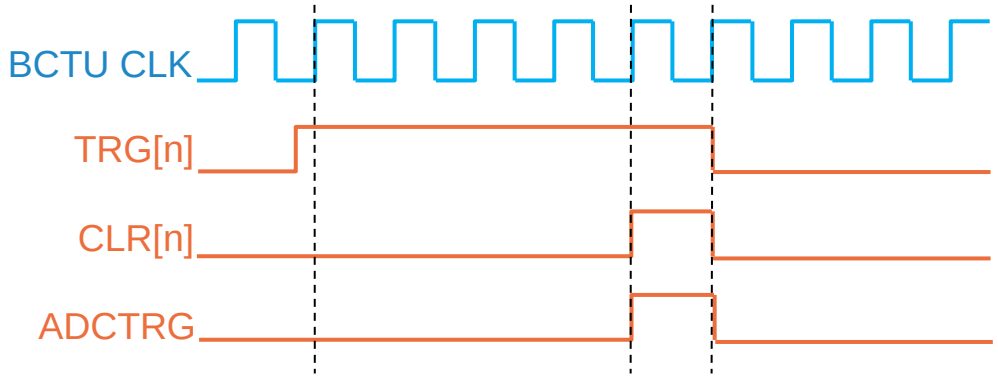
- When a timer output asserts a BCTU trigger, BCTU passes the trigger to an ADC input and stores the trigger assertion until the corresponding ADC begins the requested conversion.
- After the ADC completes the conversion, BCTU asserts a trigger clear signal to clear both the internally stored trigger and the external trigger source if they are still asserted.

# LIST CHANNEL ADDRESS REGISTER (LISTCHR_0~15)

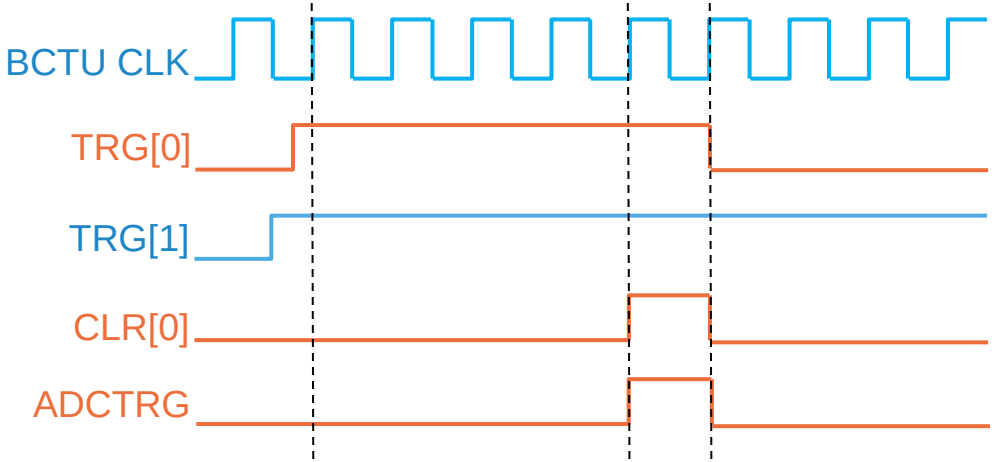



- LAST_y: the last channel in the CL
- NEXT_CH_WAIT_ON_TRIG_y: stop after executing the current ADC channel
- ADC_CH_y: select ADC channel for the y element of the CL
- LAST_y_plus_1: the next-to-last channel in the CL
- NEXT_CH_WAIT_ON_TRIG_y_plus_1 : stop after executing the current ADC channel
- ADC_CH_y_plus_1 : select ADC channel for the (y+1) element of the CL

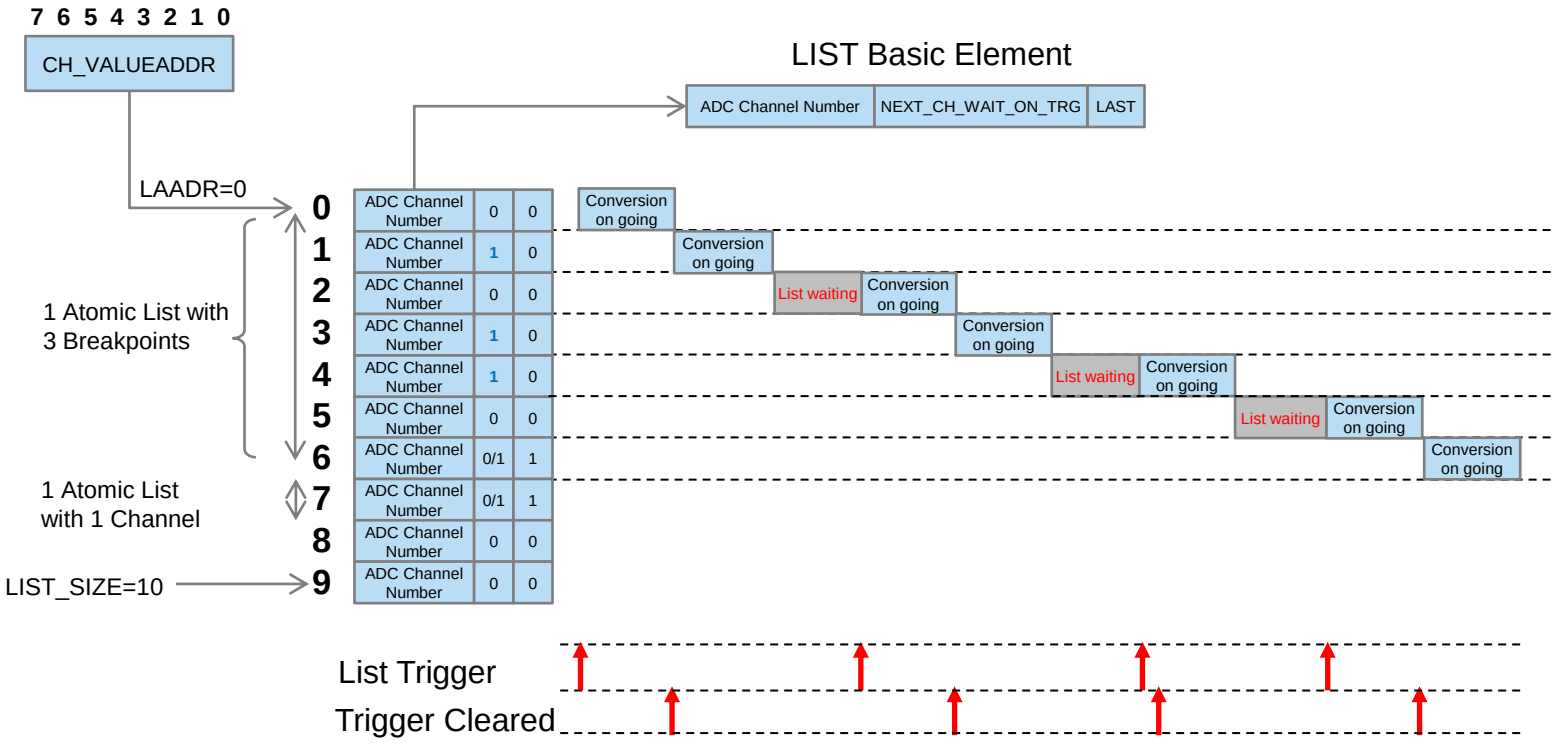
- Trigger Timing
  - There are 4 clock cycles from assertion of a BCTU trigger input to assertion of the trigger output to ADC.
  - This figure also shows the CLR[n] (clear) signal generated by the BCTU which clears the input trigger signal.

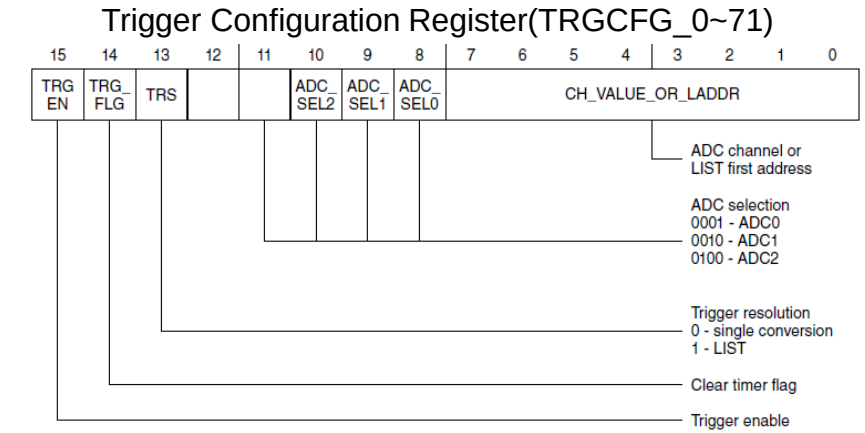



- Priority Management
  - ADC triggers are prioritized based on the trigger number (lower the trigger input number the higher is the priority)
  - Trigger input priority scheme applies when the triggers occur at the same time or when the ADC is busy in a conversion.



# BCTU CONVERSION MODES





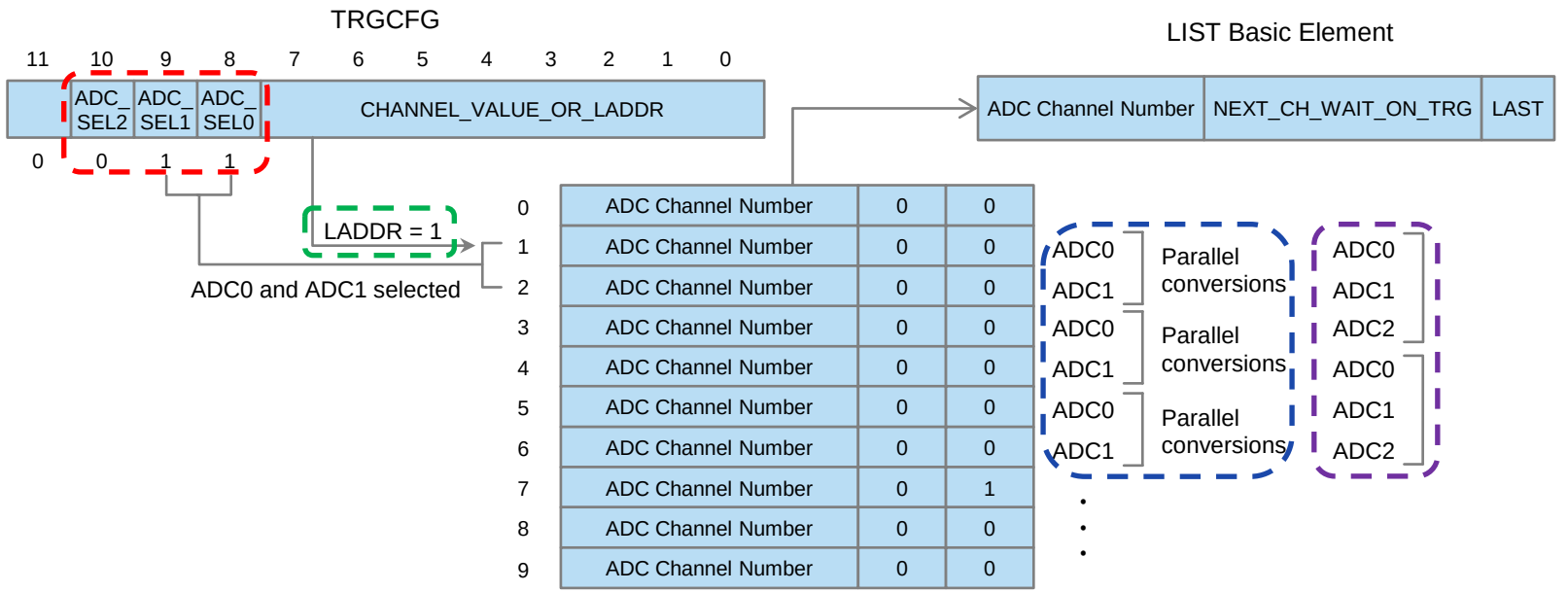
- Single(TRS = 0): The selection of ADC instances and the channel number must be configured in the TRGCFGn registers.
- LIST(TRS = 1): The selection of ADC instances and the LIST first address must be configured in the TRGCFGn registers.

  The conversion elements of LIST must be configured in the LISTCHRn registers.

  The sequence of conversions executes until the LAST field is 1.

  If the LIST is waiting due to a breakpoint, then BCTU will clear the trigger source and the LIST execution pauses after the current conversion is completed; The LIST execution resumes when the same trigger asserts again.

# MULTIPLE PARALLEL CONVERSIONS (MPC)




- The Multiple Parallel Conversions functionality is implemented using the LIST of channels.
- More than 1 ADC should be selected in the TRGCFG register and the TRS bit must be set.
- In this case the CHANNEL_VALUE_OR_LADDR field points to an element in the LIST.
- The channels in the LIST are interpreted as referring to the ADCs selected in the TRGCFG register
- If 3 ADCs are selected, the LIST is interpreted in groups of 3 elements, being the order from ADC0 through ADC2

# ADC CONVERSION RESULTS ACCESS - ADCNDR

- BCTU ADC result data register
  - Select ADCnDR to store the conversion results
  - 3 registers (for every ADC instance)
  - Additional information (trigger source, channel, LIST status)
- DMA transfer
  - Enable DMA requests
  - select 1 channel for DMA per ADC
  - New data in ADCnDR generates a DMA request
- Available events to generate interrupt
  - Data is available from ADC0/1/2
  - Data is overwritten from ADC0/1/2
  - ADC was triggered

ADCn Result Data (ADC0DR - ADC2DR)

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | TRG_SRC | | | | | | | | CH | | | | | | LIST | LAST |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 0 | ADC_DATA | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Trigger Configuration (TRGCFG_0~71)

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | LOOP | DATA_DEST | | | 0 | | | | 0 | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | TRIGEN | TRG_FLAG | TRS | 0 | 0 | ADC_SEL2 | ADC_SEL1 | ADC_SEL0 | 0 | CHANNEL_VALUE_OR_LADDR_7BITS | | | | | | |
| W | | W1C | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

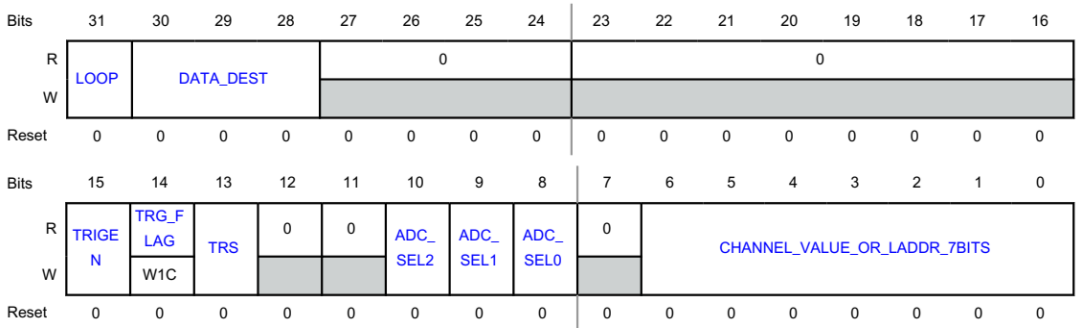
Module Configuration (MCR)

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Softwar... | MDIS | FRZ | 0 | | GTRGEN | 0 | | | | | | | DMA2 | DMA1 | DMA0 |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 0 | | | | | | | | TRGEN | 0 | LIST_IEN | 0 | | IEN2 | IEN1 | IEN0 |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

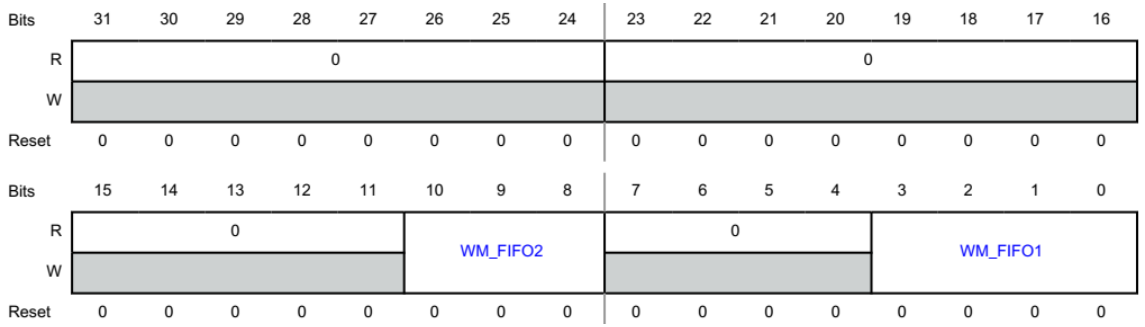
# ADC CONVERSION RESULTS ACCESS - FIFONDR

## Trigger Configuration (TRGCFG_0~71)



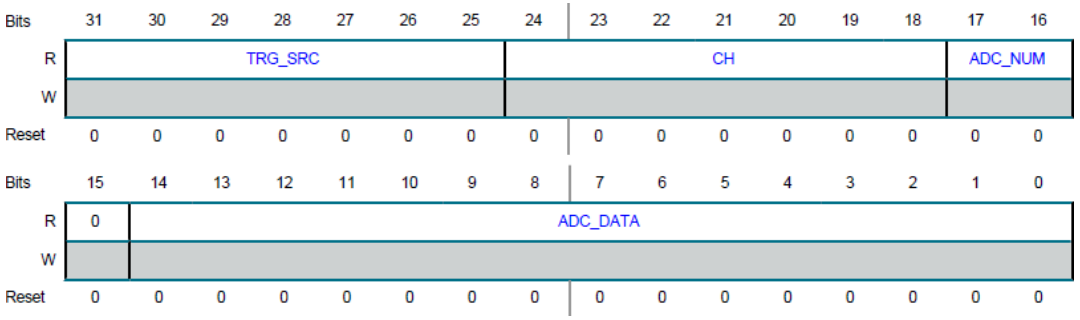

DATA_DEST set destination to FIFO1/FIFO2.

## FIFO Watermark Configuration (FIFOWM)



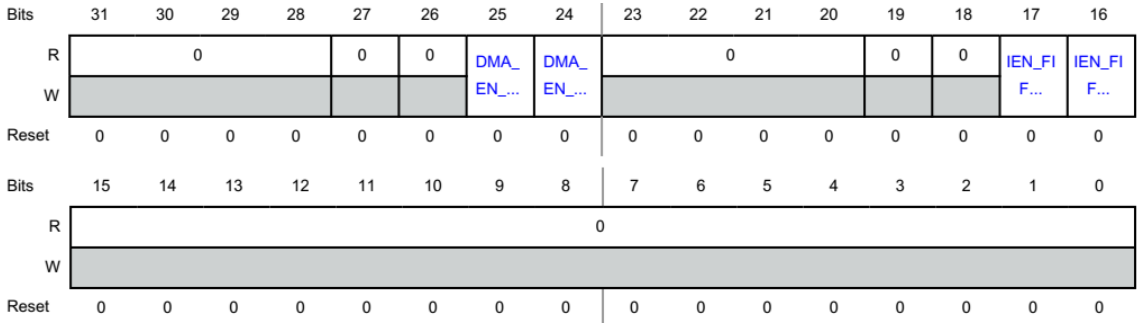

Define FIFO watermark level to trigger DMA/IRQ.

## FIFO Result Data (FIFO1DR - FIFO2DR)




Conversion results and other information in FIFOnDR.

## FIFO Control (FIFOCR)




Enable DMA or IRQ for handling FIFO data.

# Trigger Multiplexer (TRGMUX)

SECURE CONNECTIONS
FOR A SMARTER WORLD

- The S32K3xx chip has only 1 instance of TRGMUX module:

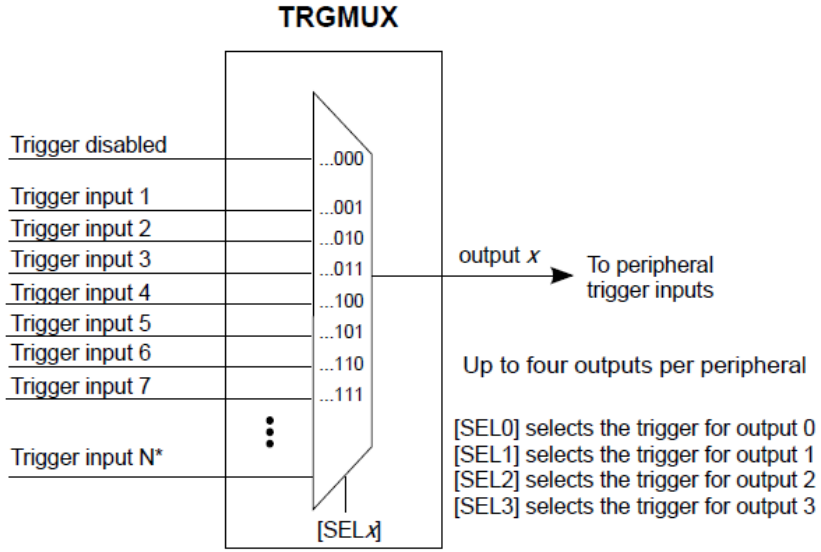
| Instance | S32K388, S32K358/S32K348/S32K338/S32K328, S32K344/S32K324/S32K314 |
|---|---|
| TRGMUX | Yes |

- What is TRGMUX:
  - Flexible mechanism for connecting various trigger sources to multiple peripherals.
  - The signal interconnection can be programmed by software.
  - For the supported trigger sources and destination, see the TRGMUX connectivity file attached to the Reference Manual document.

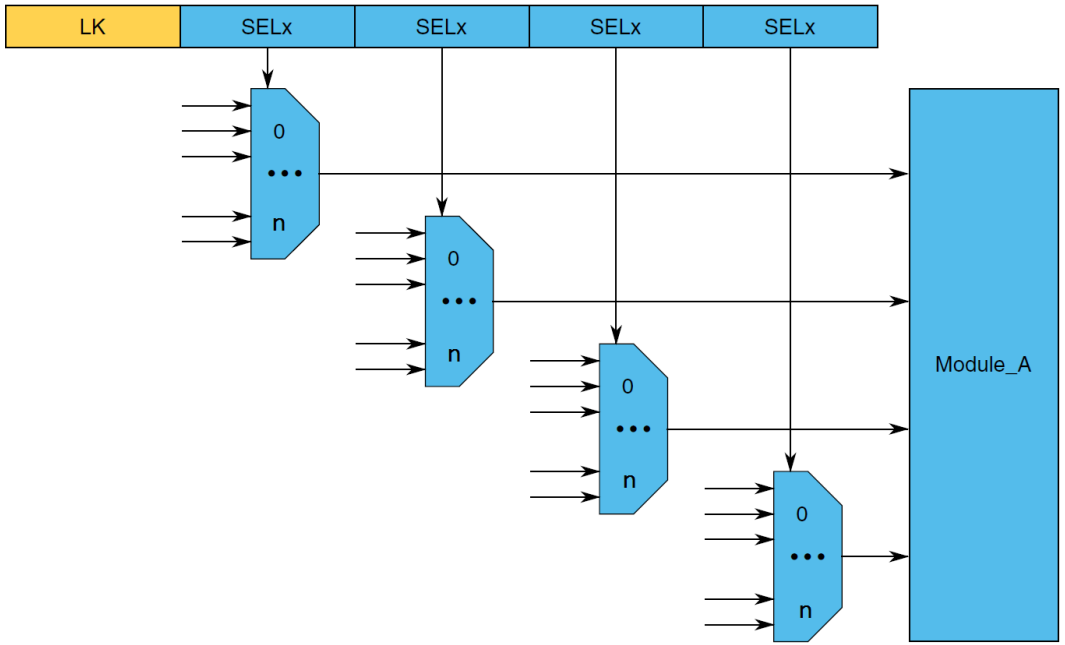

*Note: S32K344 supports 128 x 112 signal routing paths*

- Each peripheral has its own specific 32-bit trigger control register, each control register support up to 4 trigger sources.

- Up to 255 trigger inputs may be available for SEL0, SEL1 and SEL2. For SEL3, up to 127 trigger inputs may be available.




* Up to 255 trigger inputs may be available for SEL0, SEL1, and SEL2. For SEL3, up to 127 trigger inputs may be available. When the number of trigger inputs is 255, SEL3 is not available and becomes reserved. See the chip-specific TRGMUX information for the maximum number of trigger inputs supported on this device.

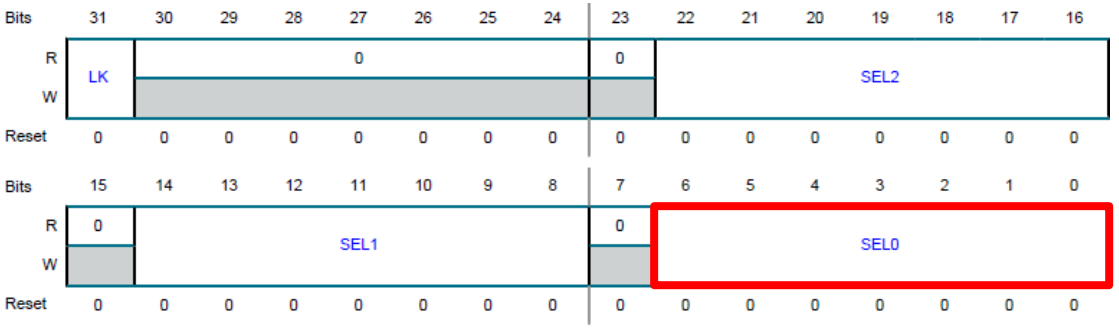



| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | LK | SEL3 | | | | | | | 0 | SEL2 | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 0 | SEL1 | | | | | | | 0 | SEL0 | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# TRIGGER MULTIPLEXER (TRGMUX)

| TRGMUX output | | ADC12_External trigger for **normal** conversion | ADC12_External trigger for **injected** conversion | ADC12_External triggerto sync the start pulses | LPCMP_Sample/Window input | BCTU_Trg | eMIOS_emios_odis | eMIOS_ipp_ind_emios | FlexIO_External Input Trigger | SIUL_TRGMUX_OUT | LPI2C_Trigger input | LPSPI_Trigger input | LPUART_Trigger input | LCU_sync | LCU_force | LCU_lc_inp | CM7_0_RXEV | CM7_1_RXEV | CM7_2_RXEV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **TRGMUX input** | **#Signals** | | 3 | 3 | 3 | 3 | 4 | 28 | 4 | 16 | 1 | 3 | 4 | 4 | 6 | 24 | 1 | 1 | 1 |
| Logic 0 (VSS) | 1 | NA | NA | NA | | NA | NA | NA | | | NA | NA | NA | | | | NA | NA | NA |
| Logic 1 (VDD) | 1 | NA | NA | NA | | NA | NA | NA | | | NA | NA | NA | | | | NA | NA | NA |
| ADC12_EOC | 3 | | | | NA | NA | NA | | | | NA | NA | NA | | NA | | NA | NA | NA |
| LPCMP_COUT output | 3 | | | | NA | | NA | | | | NA | NA | | | | | NA | NA | NA |
| eDMA_eDMA_Done | 4 | | | | NA | NA | NA | | NA | | NA | NA | NA | NA | NA | | NA | NA | NA |
| eMIOS_reload_out | 8 | | | | | | | | | | | | NA | | NA | | NA | NA | NA |
| eMIOS_ipp_do_emios | 36 | | | | | NA | | | | | | | | | | | NA | NA | NA |
| FlexIO_External Output Trigger | 4 | NA | NA | NA | NA | NA | | | NA | | NA | NA | NA | | NA | | NA | NA | NA |
| SIUL_TRGMUX_IN | 16 | | | | | | | | | NA | | | | | | | | | |
| LPI2C_Master trigger output | 1 | NA | NA | NA | NA | NA | NA | | | | NA | NA | NA | NA | NA | | NA | NA | NA |
| LPI2C_Slave trigger output | 1 | NA | NA | NA | NA | NA | NA | | | | NA | NA | NA | NA | NA | | NA | NA | NA |
| LPSPI_End of frame trigger | 3 | NA | NA | NA | NA | NA | NA | | | | NA | NA | NA | NA | NA | | NA | NA | NA |
| LPSPI_Receive data trigger | 3 | NA | NA | NA | NA | NA | NA | | | | NA | NA | NA | NA | NA | | NA | NA | NA |
| LPUART_trg_txword | 3 | NA | NA | NA | NA | NA | NA | | | | NA | NA | NA | NA | NA | | NA | NA | NA |
| LPUART_trg_rxword | 3 | NA | NA | NA | NA | NA | NA | | | | NA | NA | NA | NA | NA | | NA | NA | NA |
| LPUART_trg_rxidle | 3 | NA | NA | NA | NA | NA | NA | | | | NA | NA | NA | NA | NA | | NA | NA | NA |
| LCU_lc_out | 24 | | | | | | | | | | | | | NA | NA | NA | | | |
| PIT_CH | 8 | | | | | | | | | | | | NA | | NA | | | | |
| PIT_CH_RTI | 1 | | | | | | | | | | | | NA | | NA | | | | |
| CM7_0_TXEV | 1 | | | | NA | NA | NA | | | | | | NA | | NA | | NA | | |
| CM7_1_TXEV | 1 | | | | NA | NA | NA | | | | | | NA | | NA | | | NA | |




- You can flexibly route any trigger from input to any output.
- Where #Signals is number of similar signals in TRGMUX for the same IP blocks. (For example, 2 PIT module take 8 PIT_CH triggers.)
- Where NA, this route is not available.
- PIT_CH is routed to ADC_External_Normal_Conversion.
- In TRGMUX ADC12_n register set appropriate SELx to appropriate value of PIT_CH.

# ADC Component Configuration using RTD 3.0.0 IP config

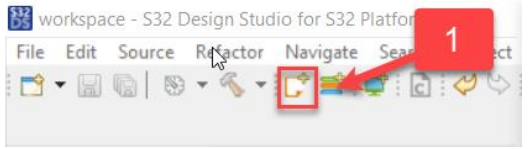
SECURE CONNECTIONS
FOR A SMARTER WORLD

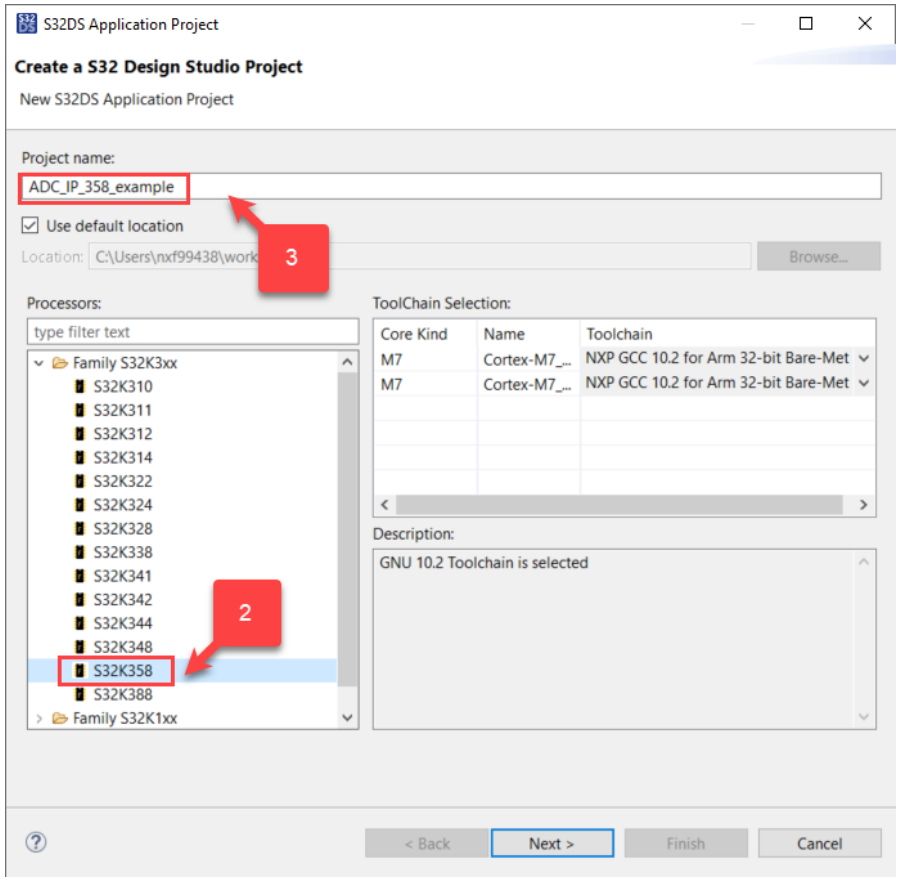
# HOW TO CREATE THE ADC PROJECT INTO S32DS

1.- Click on S32DS Application project.

2.- Select the Microcontroller family you need, for this **S32K358** is set up.

3.- Give a Project name, then click next**.**



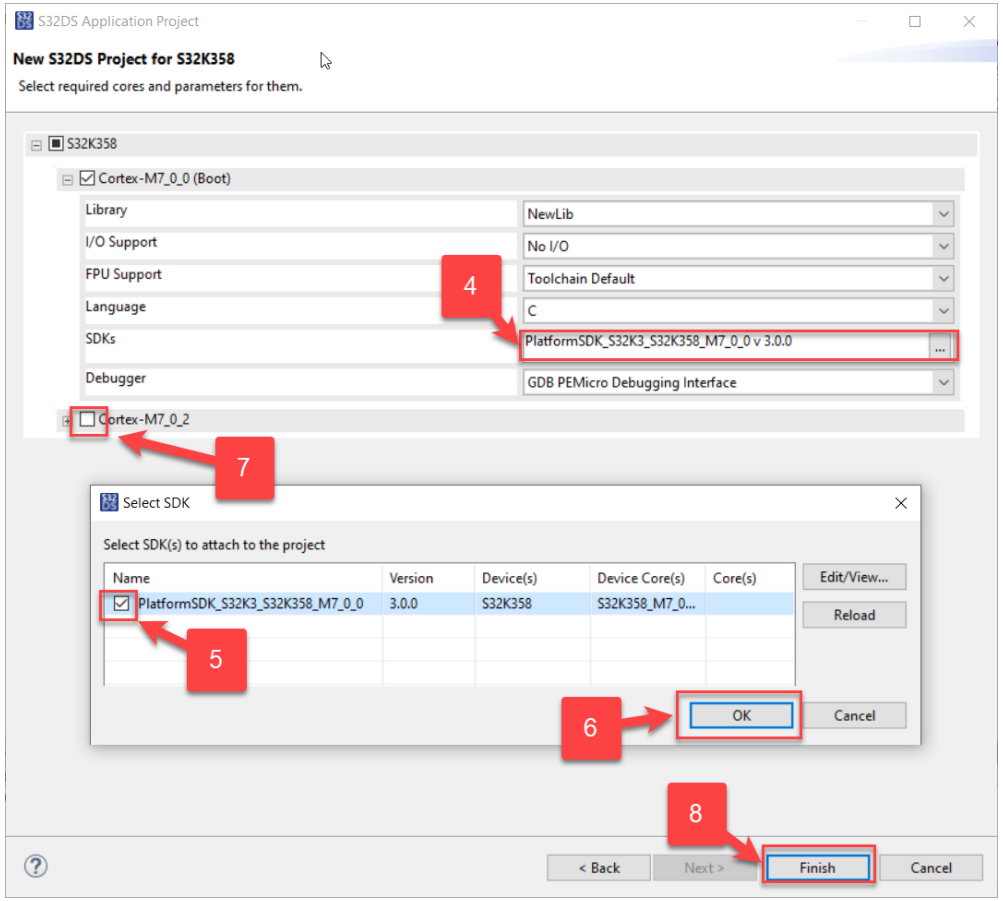
4.- Select the SDK, and a prompt is going to be shown.

5.- Check the box "PlatformSDK_S32K3_S32K358".

6.- Click OK button.

7.Uncheck/DIselect the box for "Cortex-M7_0_2", because we only need one core.

8. Click on the “Finish” button

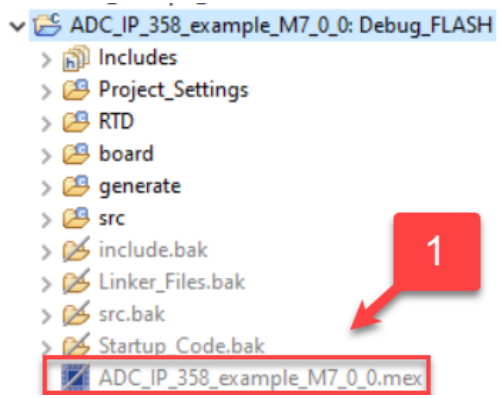


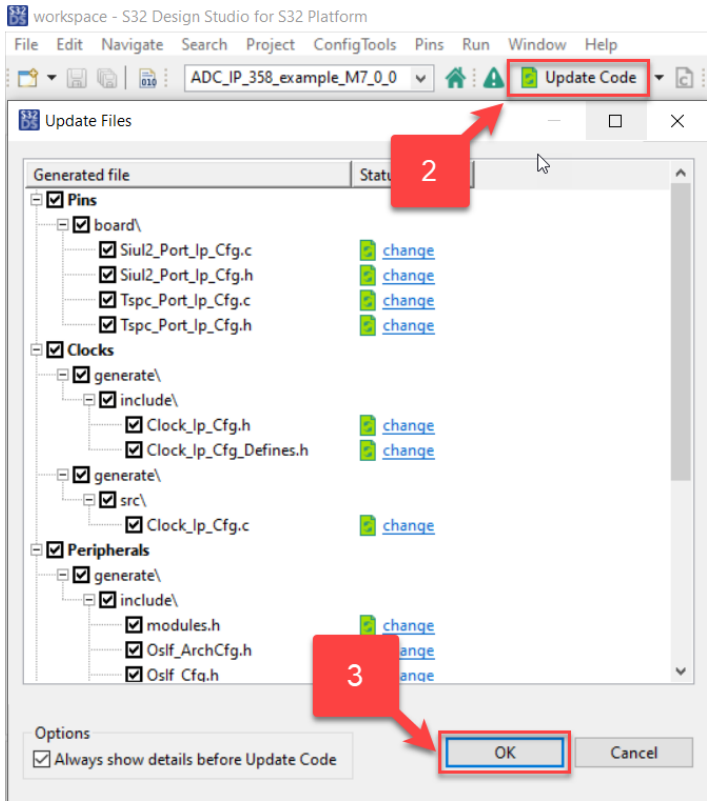
# HOW TO CREATE THE ADC PROJECT INTO S32DS

1.- Double click on ".mex" file located in the Project Explorer tree displayed in the left side of S32DS. You will get switched to pins view. Please see next slide.

2.- Click on "Update Code".

3.- Click "OK" to confirm the change in files.



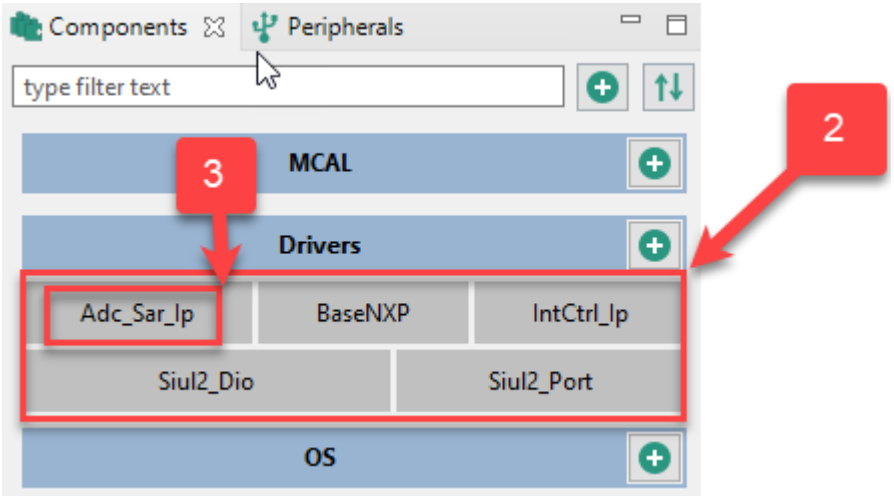



# HOW TO CREATE THE ADC PROJECT INTO S32DS

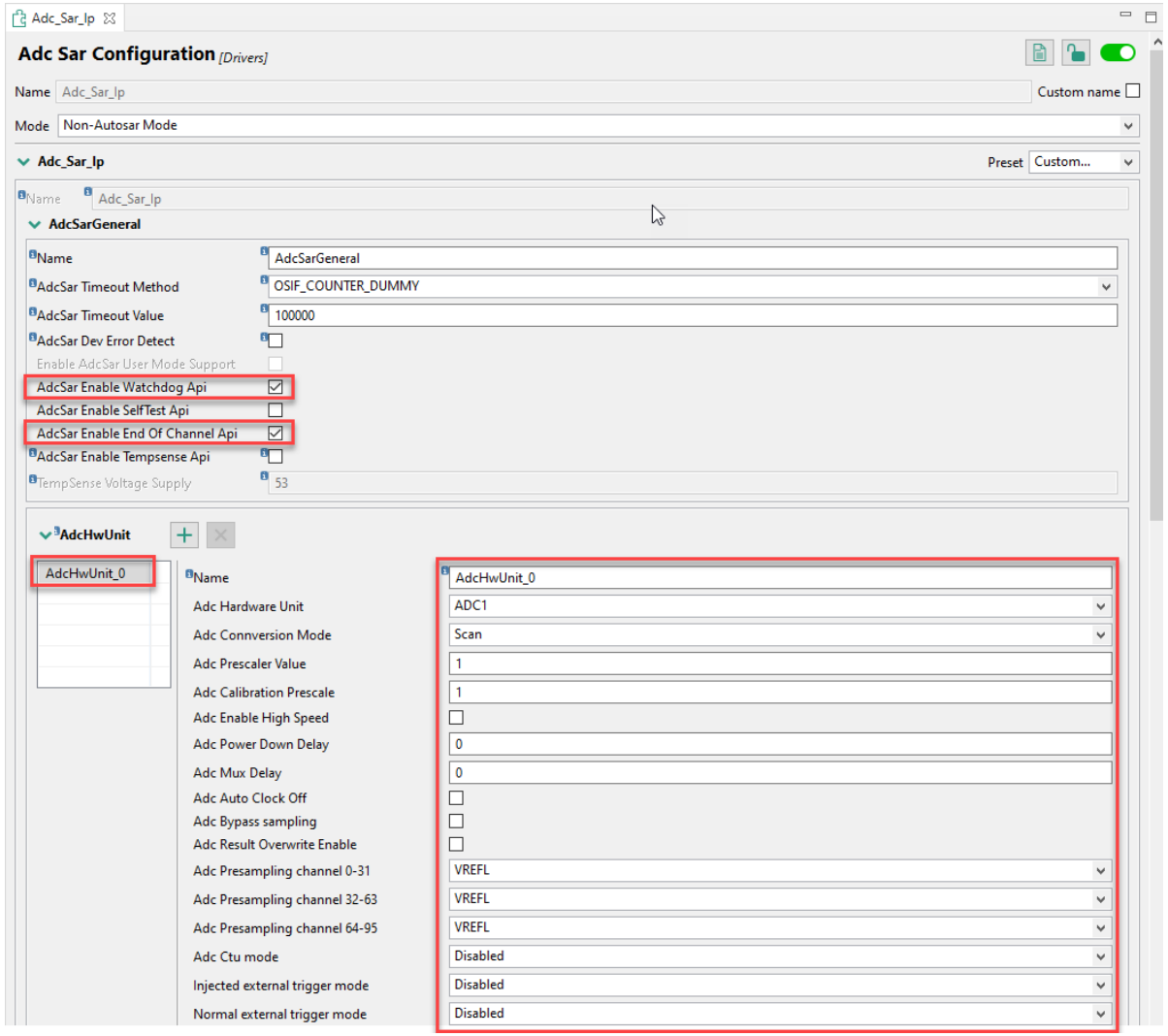
1.- From the top bar, change the perspective view to “Peripherals”


2.- Ensure all the drivers shown in the image are included. If some of them are not included click on the “+” icon to add the driver.

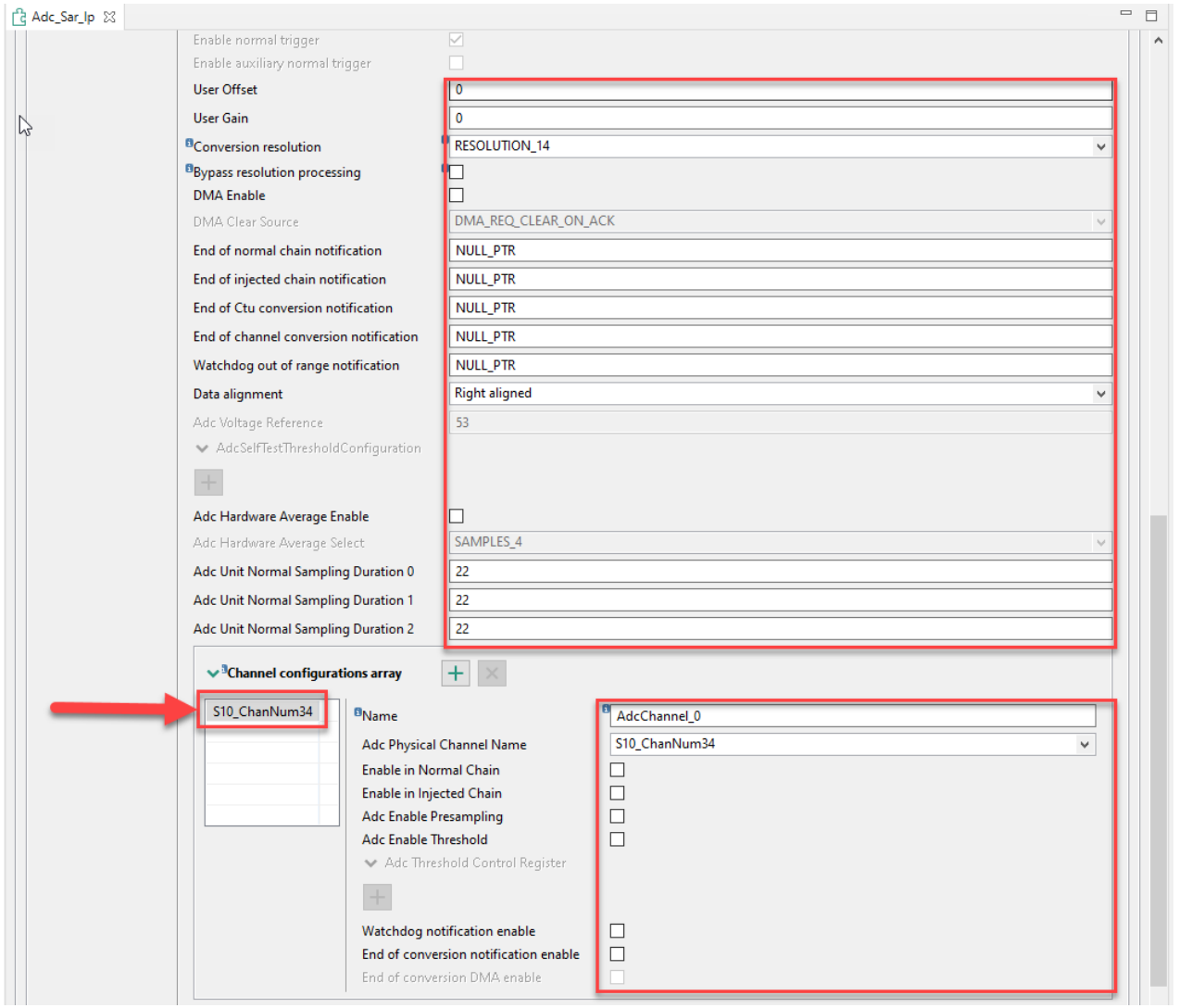
3.-Doucle click in the “ADC_Sar_Ip” driver



In the “Adc_Sar_Ip” module ensure the following configuration for instance 0:



# Configure S10 ADC the channel number 34 as shown

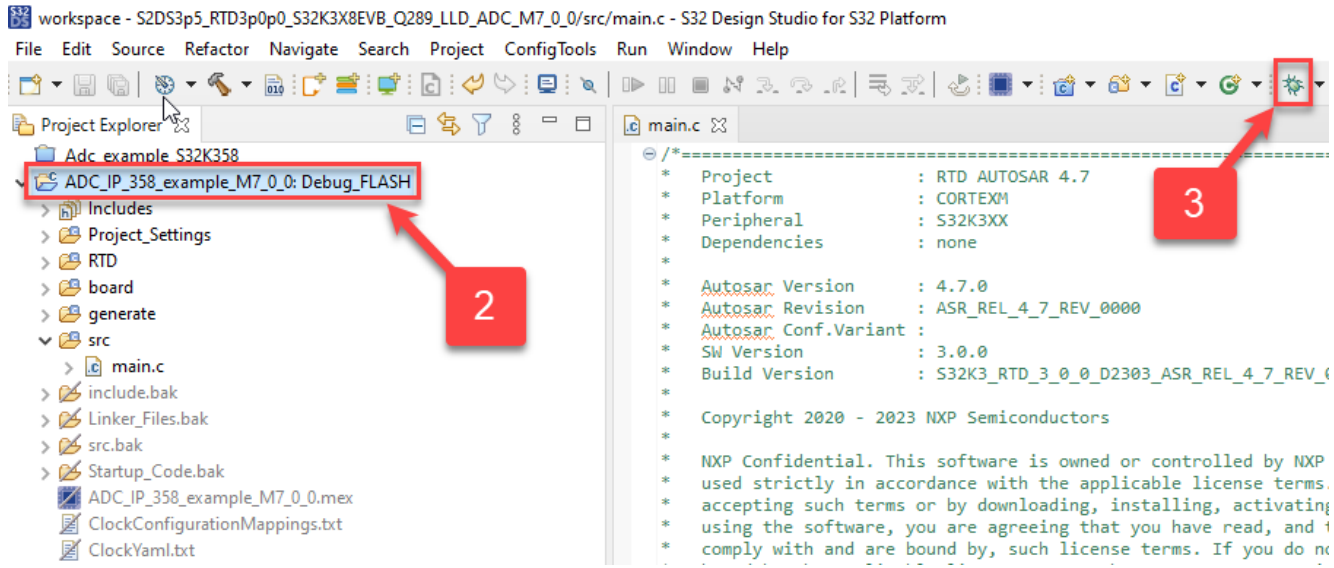


1.- Switch to C perspective view.

2. - Click over the folder name.

3.- Debug code. For this action board need to be plugged to source power and USB cable.

4.- Run the project

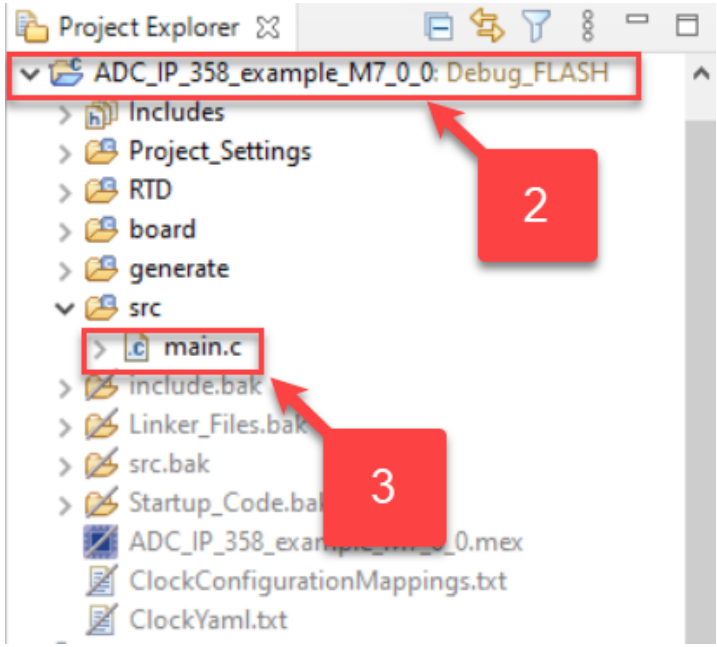





1.- Change to C perspective view.

2.- Open the folder which contains the project.

3.-Open the “main” file and erase its content to paste the following code:



# HOW TO CREATE THE ADC PROJECT INTO S32DS

```
/*==================================================================================================
*                                        INCLUDE FILES
* 1) system and project includes
* 2) needed interfaces from external units
* 3) internal and external interfaces from this unit
==================================================================================================*/
#include "Clock_Ip.h"
#include "IntCtrl_Ip.h"
#include "Adc_Sar_Ip.h"
#include "Siul2_Dio_Ip.h"
#include "Siul2_Port_Ip.h"

/*==================================================================================================
*                                      EXTERN DECLARATIONS
==================================================================================================*/
/* By default S32K342, K358 and K311 boards will have 3.3V voltage reference selected while S32K312, S32K344 and S32K396 will have 5V selected.
 * If you have S32K342 with default pin configuration or if you've manually selected 3.3V reference, please uncomment the following line: */
#define ADC_3V3_VREF_SELECTED
#define ADC_BANDGAP         5980U /* Vbandgap ~ 1.2V on 14 bits resolution, 3.3V VrefH */
volatile int exit_code = 0;
```

# HOW TO CREATE THE ADC PROJECT INTO S32DS

```
/* User includes */

#define ADC_VREFH           16383U /* 14 bits resolution */
#define NUM_RESULTS         (3u)
#define RESULT_BUFF_VAL     (0xaaaa)
#define ADC_RESULT_BUFF_VAL (0xbbbb)
#define ADC_TOLERANCE(x,y)  (((x > y) ? (x - y) : (y - x)) > 200U) /* Check that the data is within tolerated range */
#define ADC_CHANNEL 34u
#define ADC_INSTANCE 1u
volatile uint16 data;
extern void Adc_Sar_0_Isr(void);
/*!
  \brief The main function for the project.
  \details The startup initialization sequence is the following:
 * - startup asm routine
 * - main()
*/
int main(void)
{
StatusType status;
uint16_t resultConv = 0;
uint8 Index;

Siul2_Port_Ip_Init(NUM_OF_CONFIGURED_PINS0,g_pin_mux_InitConfigArr0);

Adc_Sar_Ip_Init(1u, &AdcHwUnit_0);
Adc_Sar_Ip_EnableChannel(ADC_INSTANCE, ADC_SAR_IP_CONV_CHAIN_NORMAL, ADC_CHANNEL);
```

# HOW TO CREATE THE ADC PROJECT INTO S32DS

```
/* Install and enable interrupt handlers */
IntCtrl_Ip_InstallHandler(ADC0_IRQn, Adc_Sar_0_Isr, NULL_PTR);
IntCtrl_Ip_EnableIrq(ADC0_IRQn);

/* Call Calibration function multiple times, to mitigate instability of board source */
for(Index = 0; Index <= 5; Index++)
{
   status = (StatusType) Adc_Sar_Ip_DoCalibration(1u);
   if(status == E_OK)
   {
      break;
   }
}

/*This function starts a conversion channel (normnal or injected)*/
Adc_Sar_Ip_StartConversion(1u, ADC_SAR_IP_CONV_CHAIN_NORMAL);

/*This function gets the conversion results for the selected Conversion Chain.*/
Adc_Sar_Ip_GetConvData(ADC_INSTANCE, ADC_CHANNEL);

for(;;)
{
resultConv = Adc_Sar_Ip_GetConvData(ADC_INSTANCE, ADC_CHANNEL);

if (resultConv <= RED_LOW_LED1)
{
Siul2_Dio_Ip_WritePin(LEDRED_1_PORT, LEDRED_1_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDGREEN_1_PORT, LEDGREEN_1_PIN, OFF);
Siul2_Dio_Ip_WritePin(LEDBLUE_1_PORT, LEDBLUE_1_PIN, OFF);
}
```

```
if(resultConv  >= GREEN_LOW_LED1 && resultConv  <=  GREEN_HIGH_LED1)
{
Siul2_Dio_Ip_WritePin(LEDRED_1_PORT, LEDRED_1_PIN, OFF);
Siul2_Dio_Ip_WritePin(LEDGREEN_1_PORT, LEDGREEN_1_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDBLUE_1_PORT, LEDBLUE_1_PIN, OFF);
}
if(resultConv  >= BLUE_LOW_LED1 && resultConv  <=  BLUE_HIGH_LED1)
{
Siul2_Dio_Ip_WritePin(LEDRED_1_PORT, LEDRED_1_PIN, OFF);
Siul2_Dio_Ip_WritePin(LEDGREEN_1_PORT, LEDGREEN_1_PIN, OFF);
Siul2_Dio_Ip_WritePin(LEDBLUE_1_PORT, LEDBLUE_1_PIN, ON);
}
if(resultConv  >= PURPLE_LOW_LED1 && resultConv  <=  PURPLE_HIGH_LED1)
{
Siul2_Dio_Ip_WritePin(LEDRED_1_PORT, LEDRED_1_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDGREEN_1_PORT, LEDGREEN_1_PIN, OFF);
Siul2_Dio_Ip_WritePin(LEDBLUE_1_PORT, LEDBLUE_1_PIN, ON);
}
if(resultConv  >= YELLOW_LOW_LED1 && resultConv  <=  YELLOW_HIGH_LED1)
{
Siul2_Dio_Ip_WritePin(LEDRED_1_PORT, LEDRED_1_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDGREEN_1_PORT, LEDGREEN_1_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDBLUE_1_PORT, LEDBLUE_1_PIN, OFF);
}
if(resultConv  >= CYAN_LOW_LED1 && resultConv  <= CYAN_HIGH_LED1)
{
Siul2_Dio_Ip_WritePin(LEDRED_1_PORT, LEDRED_1_PIN, OFF);
Siul2_Dio_Ip_WritePin(LEDGREEN_1_PORT, LEDGREEN_1_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDBLUE_1_PORT, LEDBLUE_1_PIN, ON);
}
```

```
if(resultConv  >= WHITE_LOW_LED1 && resultConv  <= WHITE_HIGH_LED1)
{
Siul2_Dio_Ip_WritePin(LEDRED_1_PORT, LEDRED_1_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDGREEN_1_PORT, LEDGREEN_1_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDBLUE_1_PORT, LEDBLUE_1_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDRED_0_PORT, LEDRED_0_PIN, OFF);

}
if(resultConv  >= RED_LOW_LED2 && resultConv  <= RED_HIGH_LED2)
{
Siul2_Dio_Ip_WritePin(LEDRED_0_PORT, LEDRED_0_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDGREEN_0_PORT, LEDGREEN_0_PIN, OFF);
Siul2_Dio_Ip_WritePin(LEDBLUE_0_PORT, LEDBLUE_0_PIN, OFF);

}
if(resultConv  >= GREEN_LOW_LED2 && resultConv  <= GREEN_HIGH_LED2)
{
Siul2_Dio_Ip_WritePin(LEDRED_0_PORT, LEDRED_0_PIN, OFF);
Siul2_Dio_Ip_WritePin(LEDGREEN_0_PORT, LEDGREEN_0_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDBLUE_0_PORT, LEDBLUE_0_PIN, OFF);

}
if(resultConv  >= BLUE_LOW_LED2 && resultConv  <= BLUE_HIGH_LED2)
{
Siul2_Dio_Ip_WritePin(LEDRED_0_PORT, LEDRED_0_PIN, OFF);
Siul2_Dio_Ip_WritePin(LEDGREEN_0_PORT, LEDGREEN_0_PIN, OFF);
Siul2_Dio_Ip_WritePin(LEDBLUE_0_PORT, LEDBLUE_0_PIN, ON);

}
```

```
if(resultConv  >= PURPLE_LOW_LED2 && resultConv  <= PURPLE_HIGH_LED2)
{
Siul2_Dio_Ip_WritePin(LEDRED_0_PORT, LEDRED_0_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDGREEN_0_PORT, LEDGREEN_0_PIN, OFF);
Siul2_Dio_Ip_WritePin(LEDBLUE_0_PORT, LEDBLUE_0_PIN, ON);

}
if(resultConv  >=  YELLOW_LOW_LED2 && resultConv  <= YELLOW_HIGH_LED2)
{
Siul2_Dio_Ip_WritePin(LEDRED_0_PORT, LEDRED_0_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDGREEN_0_PORT, LEDGREEN_0_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDBLUE_0_PORT, LEDBLUE_0_PIN, OFF);

}
if(resultConv  >= WHITE_LOW_LED2 && resultConv  <= WHITE_HIGH_LED2)
{
Siul2_Dio_Ip_WritePin(LEDRED_0_PORT, LEDRED_0_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDGREEN_0_PORT, LEDGREEN_0_PIN, ON);
Siul2_Dio_Ip_WritePin(LEDBLUE_0_PORT, LEDBLUE_0_PIN, ON);
}
}
}
```

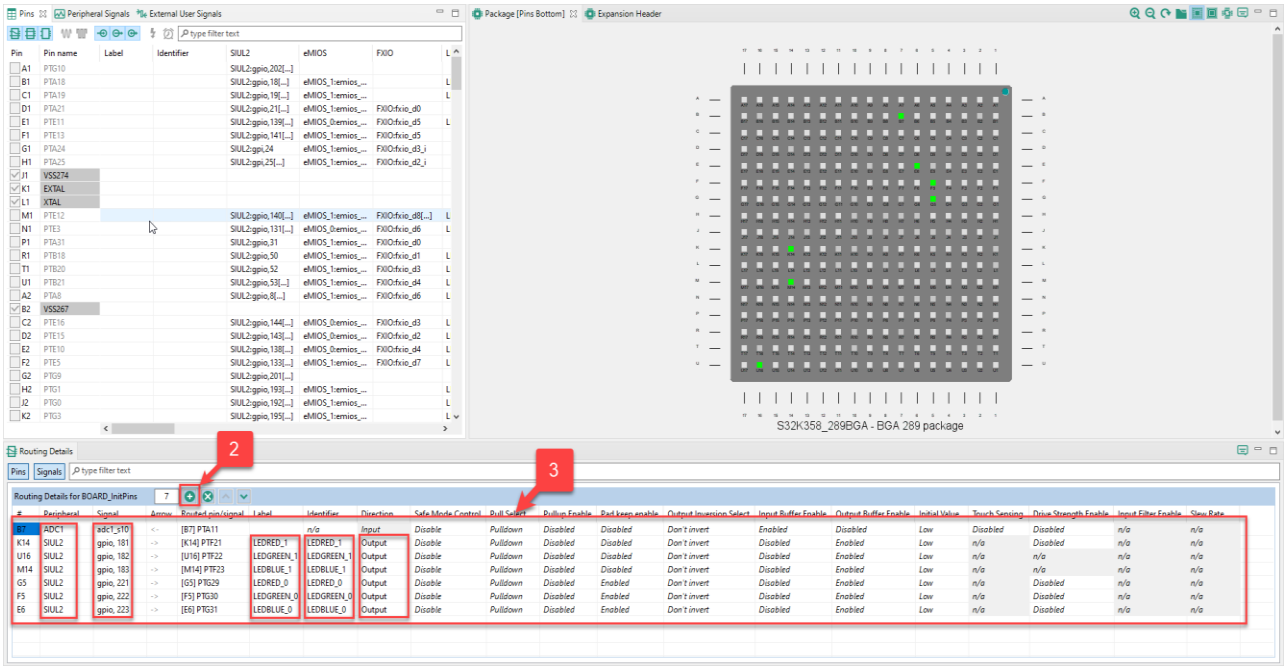
# HOW TO CREATE THE ADC PROJECT INTO S32DS


1.- Change the perspective to Pins view.

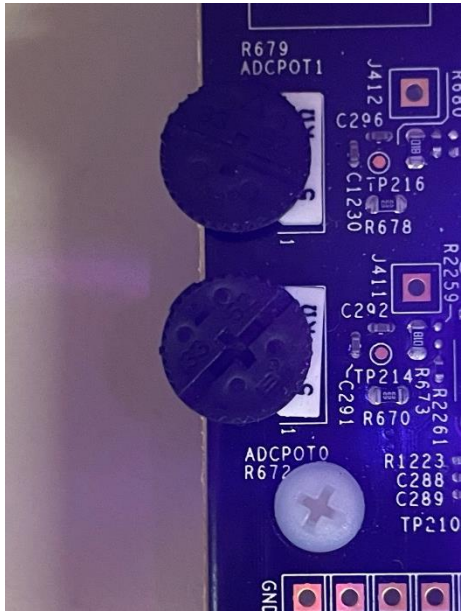
2.- Set up the pins adding a new row, 7 pins are required for this project.

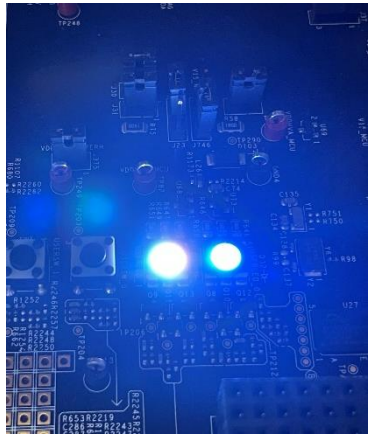
3.- The Configuration is the one we show in this image.

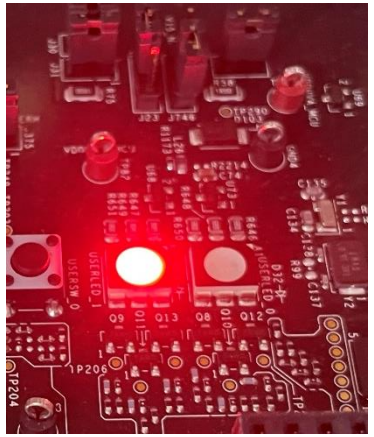
4.- Update Code clicking the button.

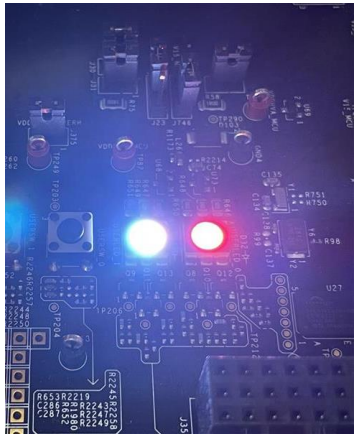


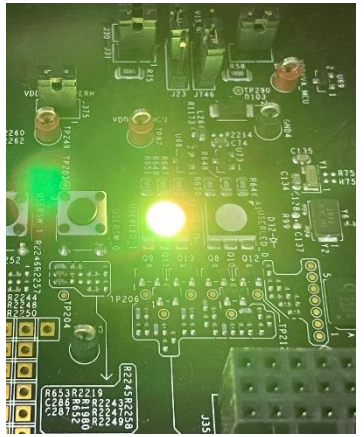

The Project is a visual demostration of ADC SAR driver. A value is read from Potentiometer ADCPOT0 and a color is shown in the USERLED_1 and USERLED_0.









# How to import an example using RTD MCAL

SECURE CONNECTIONS
FOR A SMARTER WORLD

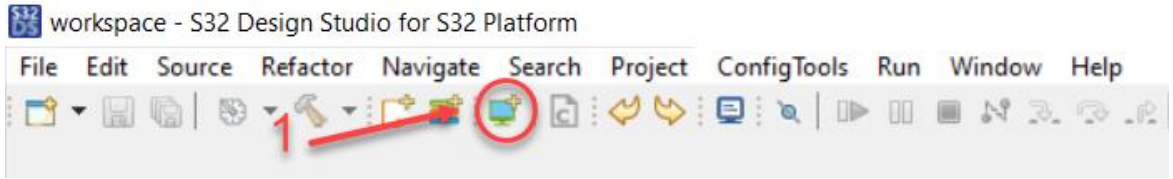
# HOW TO IMPORT AN EXAMPLE USING RTD MCAL

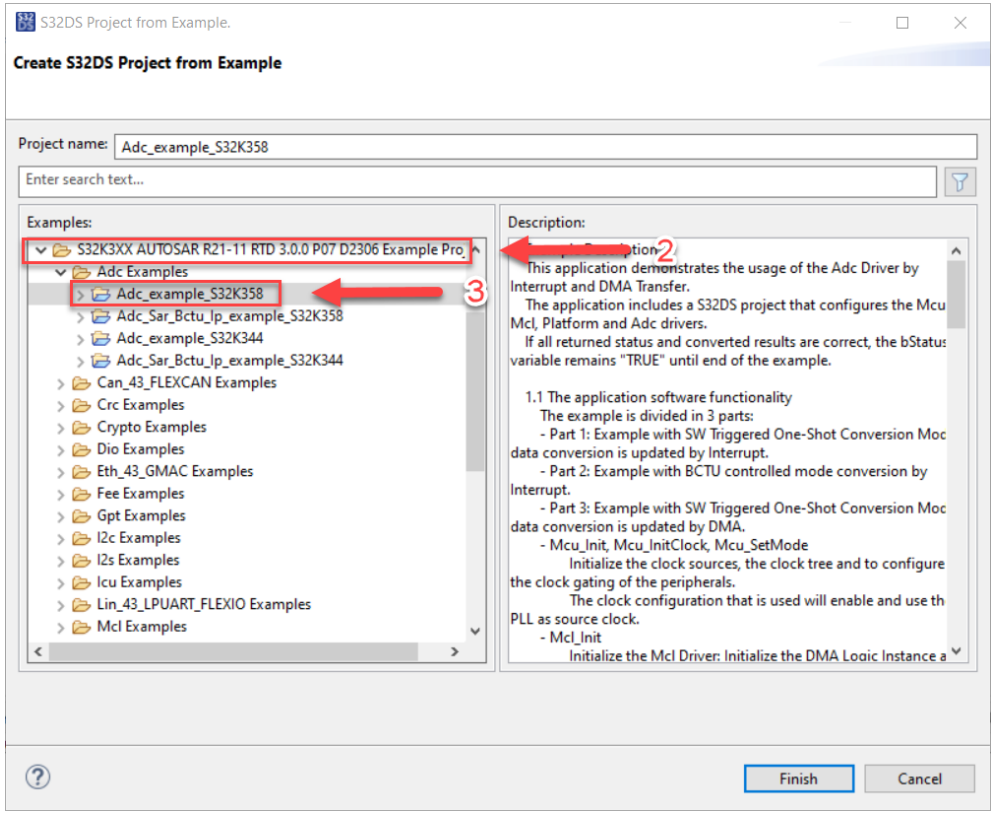
1.- Click on “S32DS Project from example”.

2.- Choose the RTD version you need.

3.- Open the ADC Example

4.- Choose the device you need, then click on Finish. **For this training we used the S32K3X8EVB-Q289. SCH-54870 REV B2.**

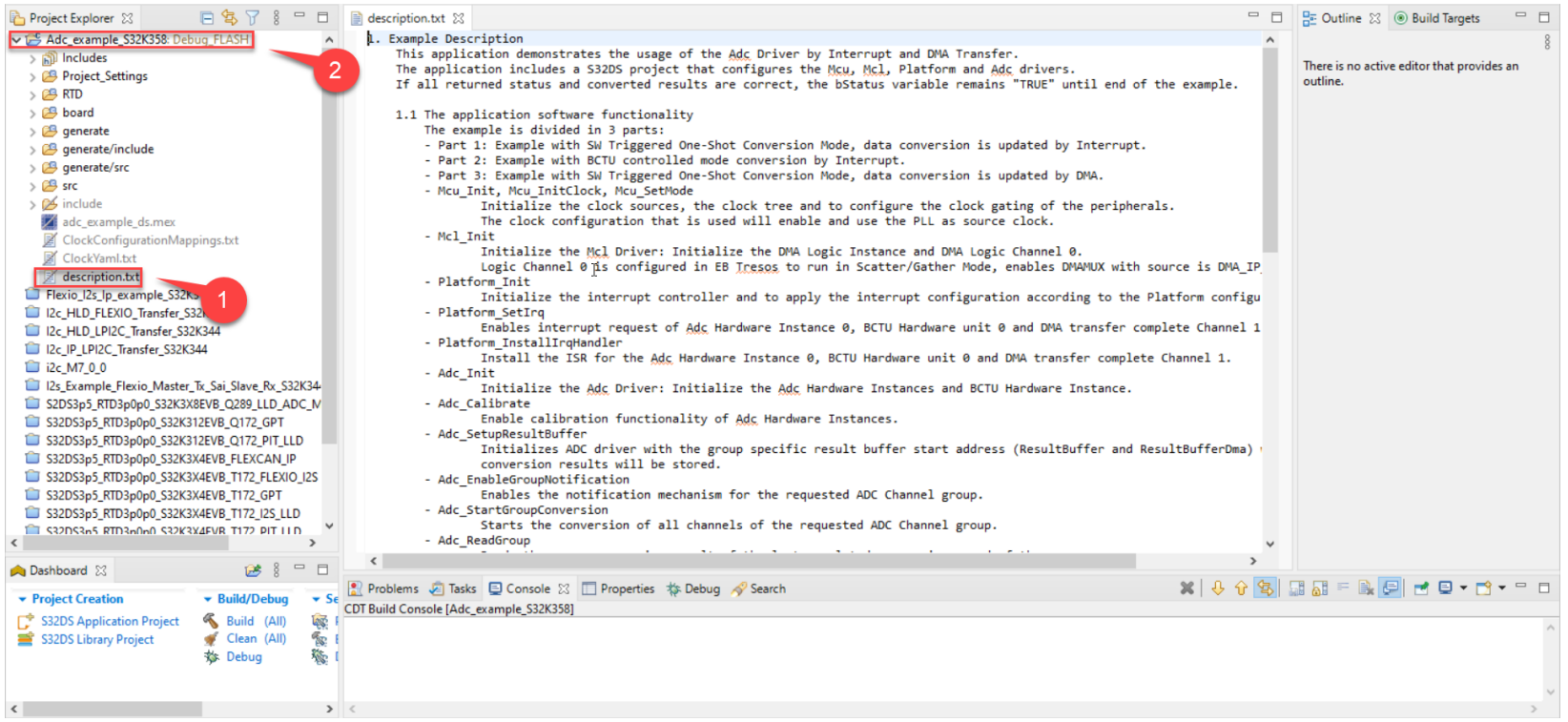


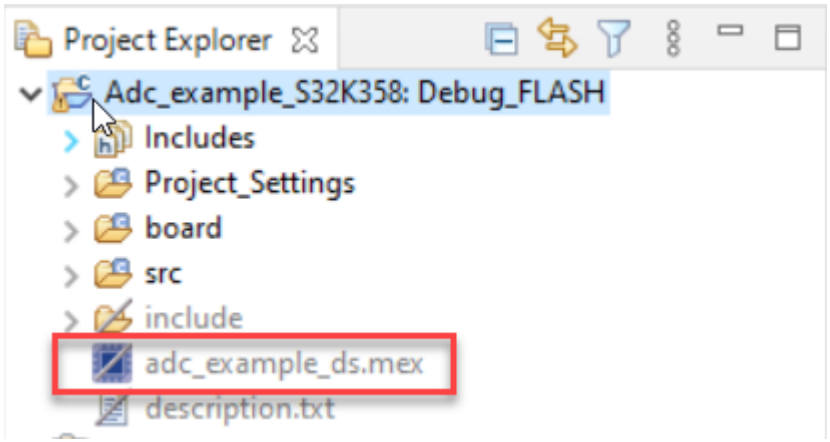
1.- Double click on “description.txt” to read the Example description.

It is important to check if there is special connections or conditions to run the example.

2.- Click just one time over the folder name



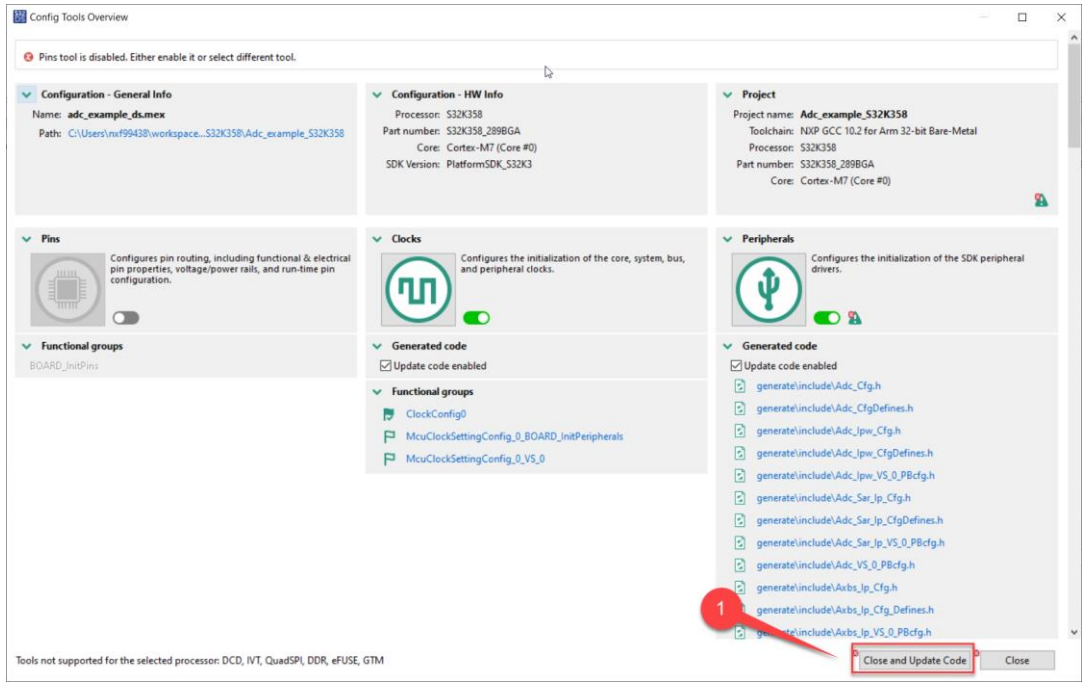
1.- Double click on ".mex" file located in the Project Explorer tree displayed in the left side of S32DS. You will get switched to pins view. Please see next slide.

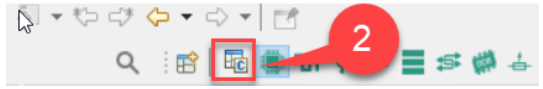


# HOW TO IMPORT AN EXAMPLE USING RTD MCAL

1.- Click in the “Close and Update Code” button.

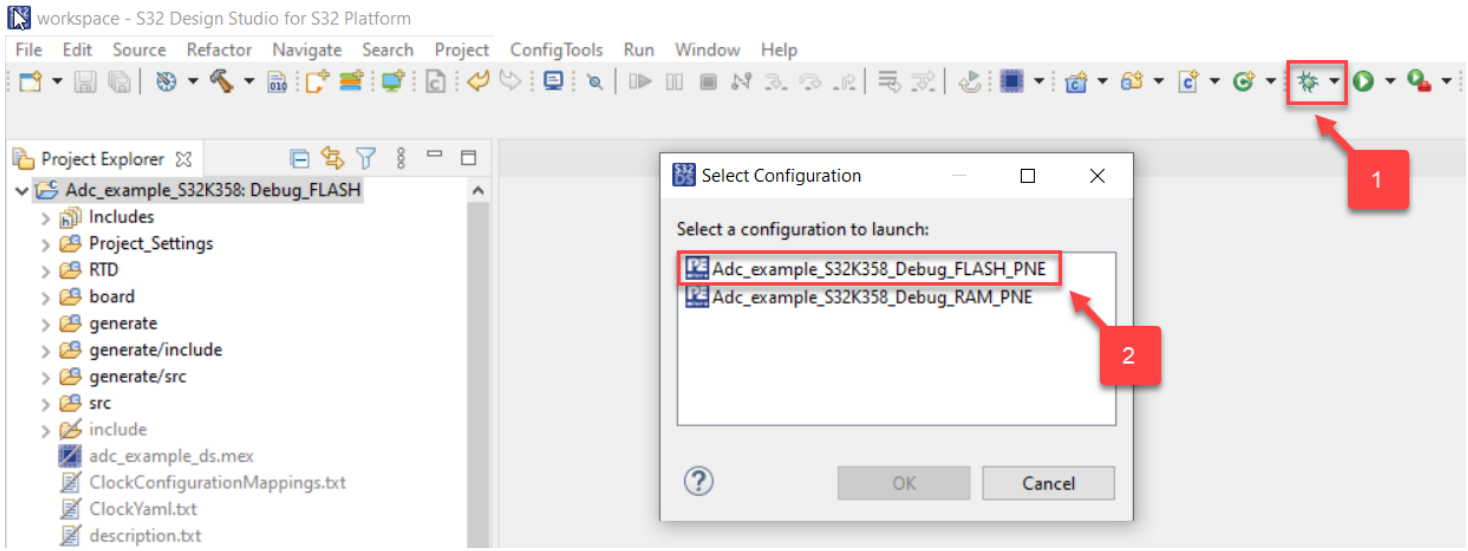
2.- Then, switch to the C perspective view.



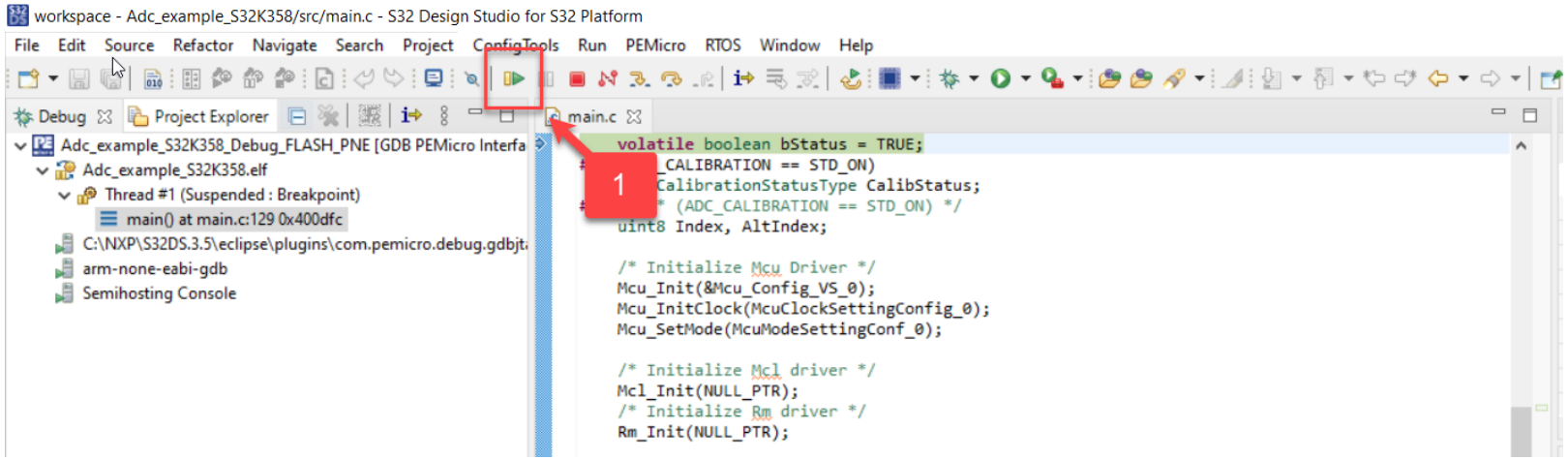



1.- Click on Debug

2.- Select the “Adc_example_S32K358_Debug_FLASH_PNE” as the configuration to launch.



1.- Run the Example.



- For further information regarding the ADC RTD driver, please consult the documentation in the following path on default installation RTD:

- C:\NXP\S32DS.3.5\S32DS\software\PlatformSDK_S32K3\RTD\Adc_TS_T40D34M30I0R0\doc\RTD_ADC_UM.pdf

SECURE CONNECTIONS
FOR A SMARTER WORLD