ADC_Filtering_1 for KIT_AURIX_TC397_TFT

ADC filtering

AURIX™ TC3xx Microcontroller Training
V1.0.1

Please read the Important Notice and Warnings at the end of this document

Scope of work

Four EVADC channels are used to convert the same analog signal with different filters enabled.

The Enhanced Versatile Analog-to-Digital Converter (EVADC) module is configured to convert four channels. The data resulting from the conversions of three channels is automatically modified: one channel computes an average on 4 results, another channel applies a 3rd order Finite Impulse Response (FIR) filter and another channel applies a 1st order Infinite Impulse Response (IIR) filter. Finally, the last channel measures the same signal without Data Modification. The channels are continuously converted and, for each of them, the maximum and minimum values are stored, which are then sent through UART in order to be compared.

Introduction

- The Enhanced Versatile Analog-to-Digital Converter module (EVADC) of the AURIX™ TC39x comprises 12 independent analog to digital converters (EVADC groups) with up to 16 analog input channels each.

- Each channel can convert analog inputs with a resolution of up to 12-bit.

- Analog/Digital conversions can be requested by several request sources:
  - **Queued request source**, specific to a single group
  - **Synchronization source**, synchronized conversion request from another ADC master kernel

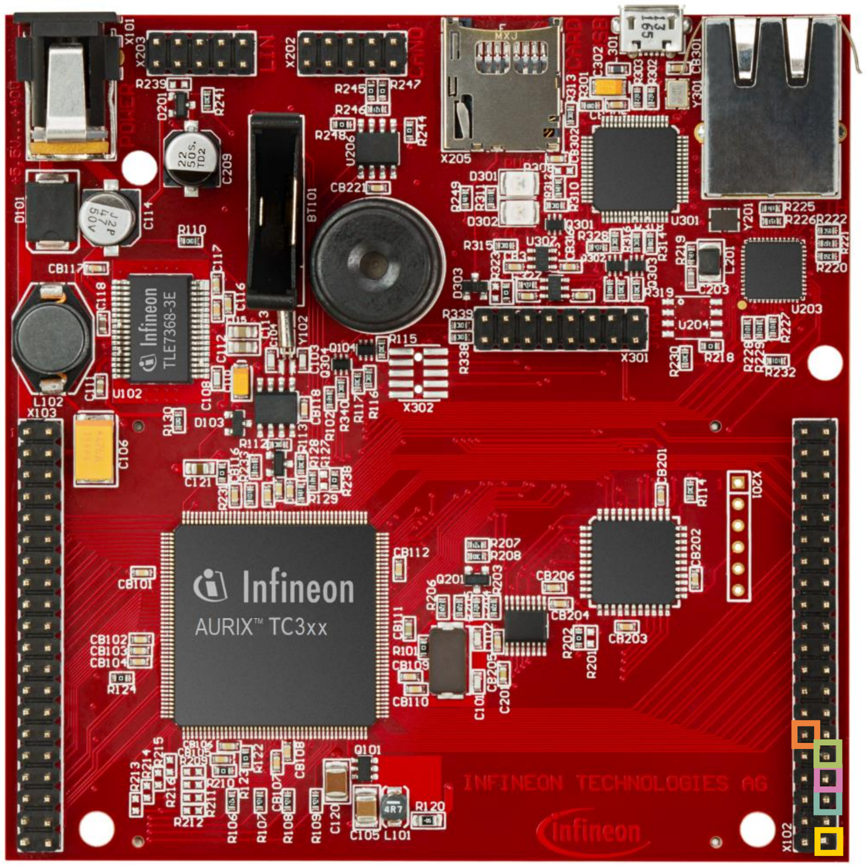
- A queued request source provides several buffer stages building a queue and can handle application-specific arbitrary conversion sequences up to the queue size.

- The trigger for the conversion can be sent:
  - Once (by another external module)
  - On a regular time base (by an external timer)
  - Permanently (by using the refill option)

Introduction

- The data resulting from conversions can be automatically modified before being used by an application to reduce the required CPU/DMA load to process the conversion.
- Three types of data modification are supported:
  - **Standard Data Reduction Mode**
  - **Result Filtering Mode**
  - **Difference Mode**
- With **Standard Data Reduction Mode**, the EVADC accumulates up to 16 values before generating a result interrupt. This mode can be used on any result register of any group GxRES0..GxRES15, where x is the number of the group.
- When **Result Filtering Mode** is enabled, depending on the configuration, the EVADC can apply either a 3rd order Finite Impulse Response (FIR) filter with selectable coefficients, or a 1st order Infinite Impulse Response (IIR) filter with selectable coefficients to the conversion results. This mode can be applied on the result registers GxRES7 and GxRES15 of any group, where x is the number of the group.
- The **Difference Mode** subtracts the content of the result register GxRES0 from the conversion results. This mode can be used on the result registers GxRES1..GxRES15 of any group, where x is the number of the group.

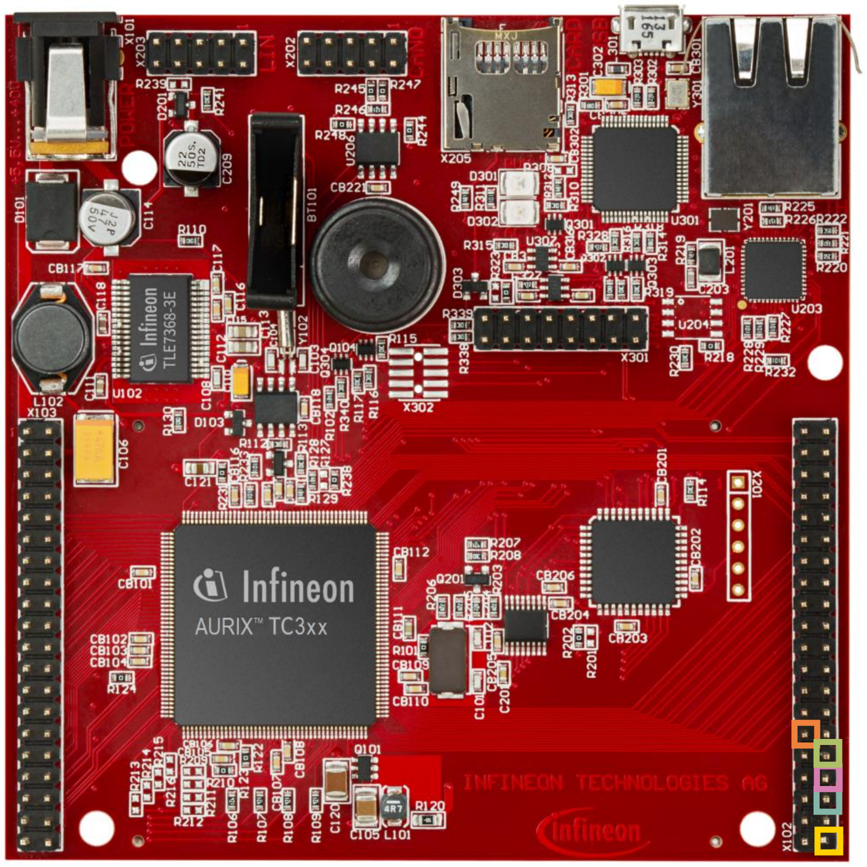
Hardware setup



This code example has been developed for the board KIT_A2G_TC397_5V_TFT. In this example, the port pins AN0, AN3, AN13 and AN21 are used, connected to VCC_IN. For using the VCC_IN, the resistor R113 must be soldered.

	X102		
P14.5	40	39	P14.4
P33.10	38	37	P20.9
P15.7	36	35	P15.6
P15.5	34	33	P15.4
P15.3	32	31	P15.2
P22.3	30	29	P22.2
P22.1	28	27	P22.0
P33.11	26	25	P23.4
P23.3	24	23	P23.2
P23.1	22	21	P23.0
P33.6	20	19	P33.8
P33.12	18	17	P33.1
P33.2	16	15	P33.3
P33.4	14	13	P33.5
AN0	12	11	AN8
AN2	10	9	AN3
AN11	8	7	AN13
AN20	6	5	AN21
GND	4	3	GND
V_UC	2	1	VCC_IN

Hardware setup



- **Note**: VCC_IN supplies the power provided to the board when the resistor R113 is soldered. If the board is supplied via the USB port, VCC_IN supplies ~3,3V, if the board is supplied with an external power supply, VCC_IN can reach 40V, thus connect the 4 analog pins to a DC signal between 0 and 5V instead.
- **Note**: The channels can be HW filtered by the board, depending on which capacitor/resistors couples are soldered. Consult the Application Kit’s Manual to check which channels are filtered by HW.

Implementation

Configuration of the EVADC

The configuration of the EVADC is done in the ***initADC()*** function in four different steps:

- Configuration of the **EVADC module**
- Configuration of the **EVADC groups**
- Configuration of the **EVADC channels**
- Configuration of the **data modification**

Configuration of the EVADC module

The default configuration of the EVADC module, given by the iLLDs, can be used for this example.
This is done by initializing an instance of the ***IfxEvadc_Adc_Config*** structure and applying default values to its fields through the function ***IfxEvadc_Adc_initModuleConfig()***.
Then, the configuration can be applied to the EVADC module with the function ***IfxEvadc_Adc_initModule()***.

Implementation

Configuration of the EVADC groups

The configuration of the EVADC groups is done by initializing an instance of the ***IfxEvadc_Adc_GroupConfig*** structure with default values through the function ***IfxEvadc_Adc_initGroupConfig()*** and modifying the following fields:

› ***arbiter*** – a structure that represents the enabled request sources, which can be one of the three queue sources. In this example, ***arbiter.requestSlotQueue0Enabled*** is set to ***TRUE***, thus enabling the request queue 0.
› ***queueRequest[0]*** – a structure that allows to configure the queue request source 0 by setting:
  – ***triggerConfig*** – a parameter that specifies the trigger configuration
› ***master*** – to indicate which converter is the master
› ***groupId*** – to select which converter to configure

While using multiple converters, they can have the same or different settings. In this example, the same configuration is applied to all of the used converters by changing the ***groupId*** parameter in a ***for*** loop.

The configuration is applied through the function ***IfxEvadc_Adc_initGroup()*** inside the ***for*** loop.

Implementation

Configuration of the EVADC channels

The configuration of each channel is done by initializing an instance of the ***IfxEvadc_Adc_ChannelConfig*** structure with default values through the function ***IfxEvadc_Adc_initChannelConfig()*** and modifying the following fields:

- ***channelId*** – to select the channel to configure
- ***resultRegister*** – to indicate the register where the A/D conversion value is stored

Then, the configuration is applied with the function ***IfxEvadc_Adc_initChannel()*** and the channel is added to the queue through the function ***IfxEvadc_Adc_addToQueue()***.

Finally, the result registers used for storing the conversion results can be configured to use data modification, in order to enable the filtering.

Configuration of the data modification

The data modification is configured in the ***applyFiltering()*** function.

The iLLDs for AURIX™ TC3xx do not support the EVADC data modification, thus it is needed to directly modify the Group Result Control Registers (GxRCRy, with x indicating the Group number and y indicating the result register where to apply the filtering).

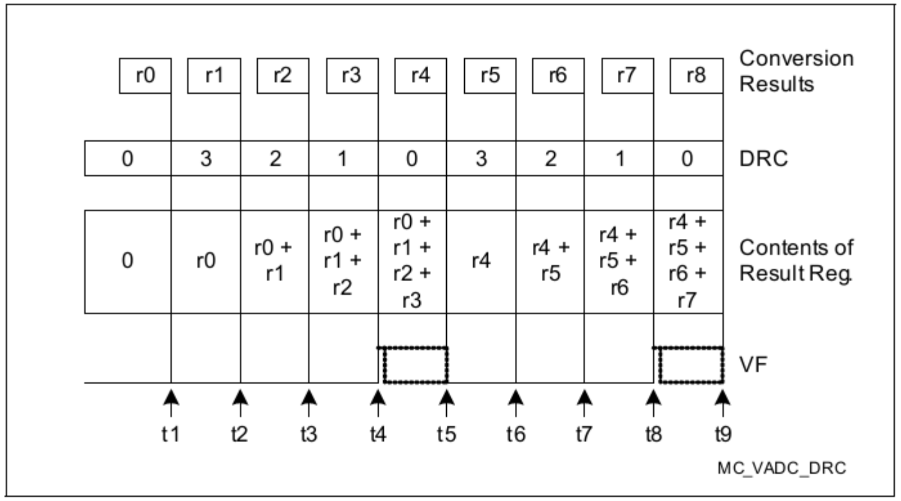
Implementation

Configuration of the data modification

To enable the **Standard Data Reduction Mode** on a specific result register, the Data Modification Mode (DMM) bit field of the associated GxRCRy register must be set to ***IfxEvadc_DataModificationMode_standardDataReduction*** (00_B) and the Data Reduction Control (DRCTR) bit field of the same register can be set to one of the values presented in table 1.

When the conversion is ready, depending on the configuration of the DRCTR bit field, the result register contains the sum of up to 16 result values, thus it is needed to divide the content of the result register GxRESy by the number of the accumulated values in order to obtain an average of the measurements.

Note: Using Standard Data Reduction Mode, the final result must be read before the next data reduction sequence starts (before t5 or t9 in the example), otherwise the Valid Flag (**VF**) bitfield **will not be cleared**.
In order to read a correct measurement, VF must be 1 and Data Reduction Counter (DRC) bitfield must be 0.



Implementation

Configuration of the data modification: Table 1

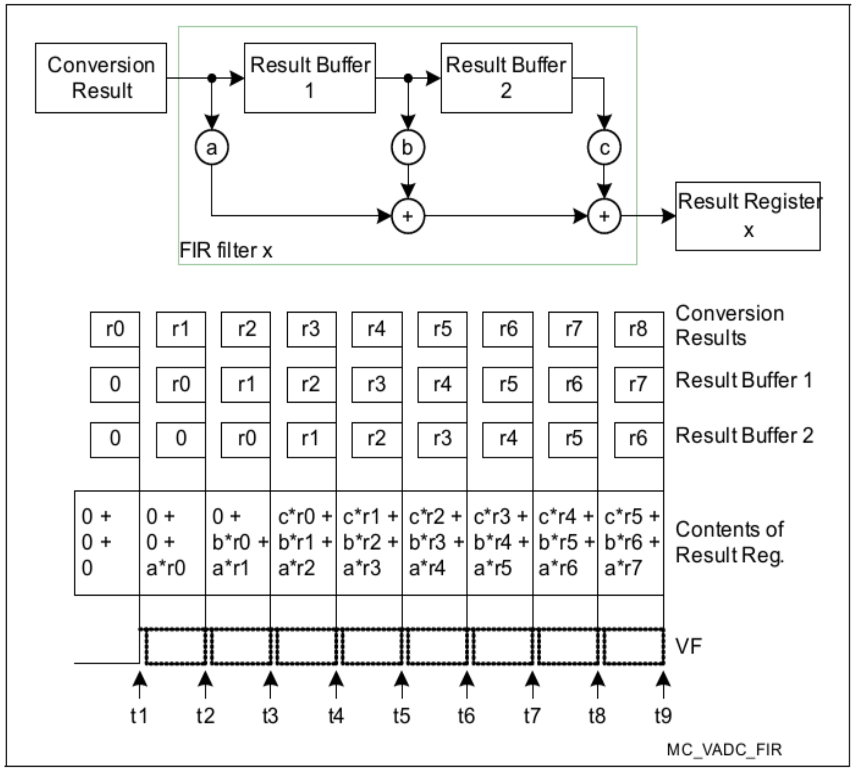
DMM	DRCTR	Filter coefficients
00_B	0000_B = ***IfxEvadc_DataReductionControlMode_0***	Data Reduction disabled
00_B	0001_B = ***IfxEvadc_DataReductionControlMode_1***	Accumulate 2 result values
00_B	0010_B = ***IfxEvadc_DataReductionControlMode_2***	Accumulate 3 result values
00_B	0011_B = ***IfxEvadc_DataReductionControlMode_3***	Accumulate 4 result values
00_B	0100_B = ***IfxEvadc_DataReductionControlMode_4***	Accumulate 5 result values
00_B	0101_B = ***IfxEvadc_DataReductionControlMode_5***	Accumulate 6 result values
00_B	0110_B = ***IfxEvadc_DataReductionControlMode_6***	Accumulate 7 result values
00_B	0111_B = ***IfxEvadc_DataReductionControlMode_7***	Accumulate 8 result values
00_B	1000_B = ***IfxEvadc_DataReductionControlMode_8***	Accumulate 9 result values
00_B	1001_B = ***IfxEvadc_DataReductionControlMode_9***	Accumulate 10 result values
00_B	1010_B = ***IfxEvadc_DataReductionControlMode_10***	Accumulate 11 result values
00_B	1011_B = ***IfxEvadc_DataReductionControlMode_11***	Accumulate 12 result values
00_B	1100_B = ***IfxEvadc_DataReductionControlMode_12***	Accumulate 13 result values
00_B	1101_B = ***IfxEvadc_DataReductionControlMode_13***	Accumulate 14 result values
00_B	1110_B = ***IfxEvadc_DataReductionControlMode_14***	Accumulate 15 result values
00_B	1111_B = ***IfxEvadc_DataReductionControlMode_15***	Accumulate 16 result values

Implementation

Configuration of the data modification

To enable the **Result Filtering Mode** on a specific result register, the Data Modification Mode (DMM) bit field of the associated GxRCRy register must be set to ***IfxEvadc_DataModificationMode_resultFilteringMode*** (01_B) and the Data Reduction Control (DRCTR) bit field of the same register can be set to enable either a 3rd order Finite Impulse Response (FIR) filter or a 1st order Infinite Impulse Response (IIR) filter, both with selectable coefficients, according to the values in table 2.

When a **FIR filter** is enabled, depending on the selected coefficients, a gain of 3 or 4 (the DC gain of a FIR filter is equal to the sum of its coefficients) is applied to the ADC result, producing a 14-bit value.
Therefore, in order to obtain the filtered measurement, it is needed to divide the content of the result register by the sum of the selected coefficients.



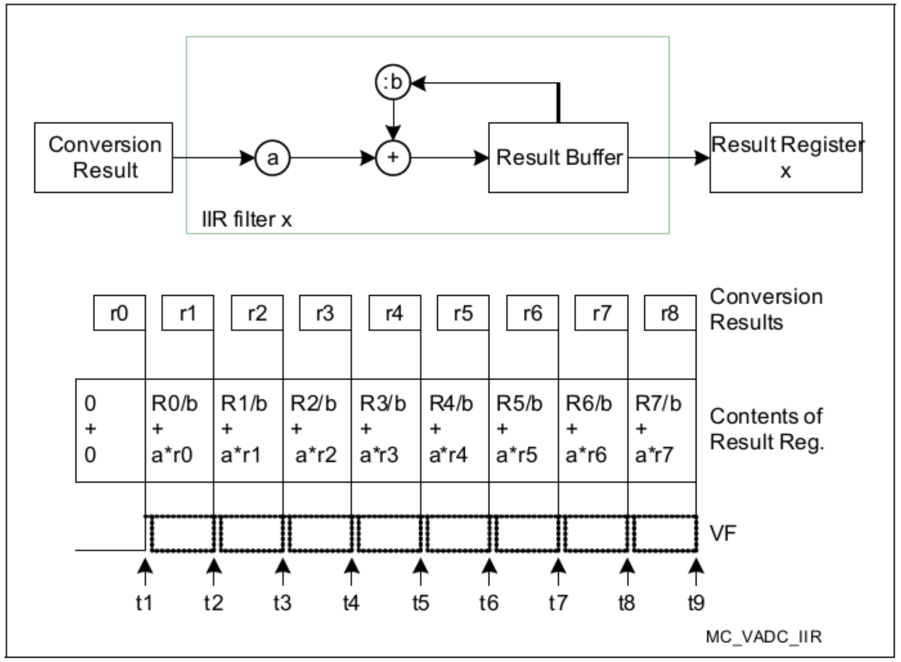
Implementation

Configuration of the data modification

The selectable coefficients for an **IIR filter** lead to a gain of 4 to the ADC result, producing a 14-bit value. Consequently, in order to obtain the filtered measurement, the content of the result register needs to be divided by 4.

All the measurement's divisions are carried out in the ***Cpu0_main.c*** file, after reading the conversion result from the result register.

The FIR and IIR filters needs to be initialized, otherwise the first values are incorrect (see the figures for the two filters).



Note: In this example, a delay before starting to read the conversion results is needed. This ensures that reading spikes in the VCC_IN supply due to the initialization of the device are avoided and that incorrect values are not read due to the filters not being yet at full speed.

Implementation

Available coefficients for FIR and IIR filters: Table 2

DMM	DRCTR	Filter coefficients
01_B	0000_B = ***IfxEvadc_DataReductionControlMode_0***	FIR filter: a=2, b=1, c=0
01_B	0001_B = ***IfxEvadc_DataReductionControlMode_1***	FIR filter: a=1, b=2, c=0
01_B	0010_B = ***IfxEvadc_DataReductionControlMode_2***	FIR filter: a=2, b=0, c=1
01_B	0011_B = ***IfxEvadc_DataReductionControlMode_3***	FIR filter: a=1, b=1, c=1
01_B	0100_B = ***IfxEvadc_DataReductionControlMode_4***	FIR filter: a=1, b=0, c=2
01_B	0101_B = ***IfxEvadc_DataReductionControlMode_5***	FIR filter: a=3, b=1, c=0
01_B	0110_B = ***IfxEvadc_DataReductionControlMode_6***	FIR filter: a=2, b=2, c=0
01_B	0111_B = ***IfxEvadc_DataReductionControlMode_7***	FIR filter: a=1, b=3, c=0
01_B	1000_B = ***IfxEvadc_DataReductionControlMode_8***	FIR filter: a=3, b=0, c=1
01_B	1001_B = ***IfxEvadc_DataReductionControlMode_9***	FIR filter: a=2, b=1, c=1
01_B	1010_B = ***IfxEvadc_DataReductionControlMode_10***	FIR filter: a=1, b=2, c=1
01_B	1011_B = ***IfxEvadc_DataReductionControlMode_11***	FIR filter: a=2, b=0, c=2
01_B	1100_B = ***IfxEvadc_DataReductionControlMode_12***	FIR filter: a=1, b=1, c=2
01_B	1101_B = ***IfxEvadc_DataReductionControlMode_13***	FIR filter: a=1, b=0, c=3
01_B	1110_B = ***IfxEvadc_DataReductionControlMode_14***	IIR filter: a=2, b=2
01_B	1111_B = ***IfxEvadc_DataReductionControlMode_15***	IIR filter: a=3, b=4

Implementation

Configuration of the data modification

In this example, the converted channels are configured as it follows:

Table 3

Channel	Data Modification Mode enabled
AN0	Standard Data Reduction Mode
AN3	Result Filtering Mode: IIR filter
AN13	Result Filtering Mode: FIR filter
AN21	No Data Modification Mode enabled

The channel AN21 has no data modification enabled in order to use it as a comparison.

Implementation

Configuration of the EVADC

When the EVADC module, its groups and channels are configured together with the Data Modification registers, the scan sequence is started with the function ***IfxEvadc_Adc_startQueue()*** in a ***for*** loop.

Read the EVADC measurements

Finally, to read a conversion, the function ***readADCValue()*** is used, which calls the ***IfxEvadc_Adc_getResult()*** function from iLLDs until a new measurement is returned (a new measurement is considered correct only when both the Valid Flag and the Data Reduction Counter bitfield are set to 1 and respectively 0, the latter is needed because the Standard Data Reduction Mode is enabled on the AN0 pin).

All the functions used to get a conversion and configuring the EVADC module, its group and channels can be found in the iLLD header ***IfxEvadc_Adc.h***.

Implementation

Configuration of the UART

In this example, the UART connection is used to make the debugging more convenient and easier to understand. The configured EVADC channels are continuously read, but the maximum and minimum values, together with the computed V_{pp} are printed using UART communication only when the user requests them.

The ***initUART()*** function initializes the UART communication.

The iLLD function ***IfxAsclin_Asc_initModuleConfig()*** fills the configuration structure ***ascConf*** with default values. Then, the parameters used to configure the module are set, depending on the needed connection: baudrate, Tx and Rx buffers, Tx and Rx pin configuration etc.

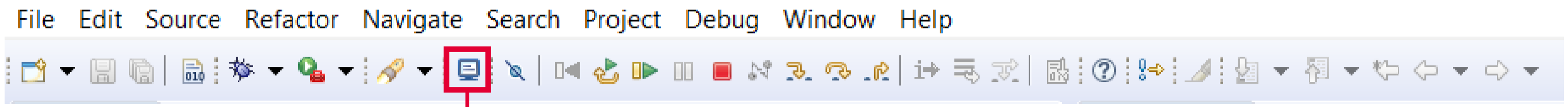
Finally, ***IfxAsclin_Asc_initModule()*** initializes the module with the user configuration and ***IfxAsclin_Asc_stdIfDPipeInit()*** initializes the standard interface to use the ASCLIN module.

The functions ***isDataAvailable()*** and ***receiveData()*** are used to interface with the ASCLIN module to check if new data is available through the function ***IfxAsclin_Asc_getReadCount()*** and, respectively, to receive data over the UART communication through the function ***IfxAsclin_Asc_read()***.
The function ***IfxStdIf_DPipe_print()*** is used to print the stored processed values.

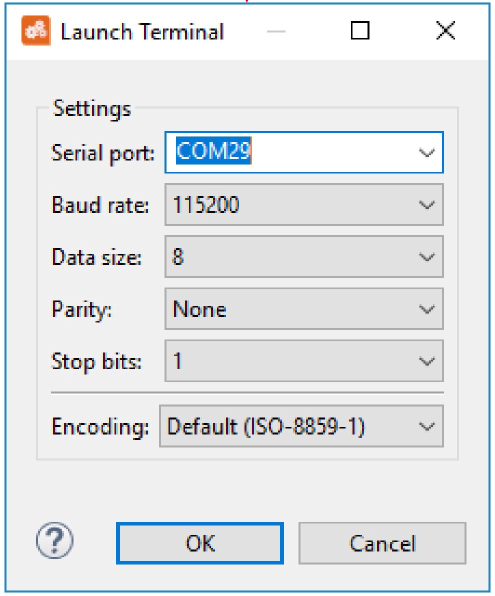
The functions used to interface and initialize the ASCLIN module can be found in the iLLD header ***IfxAsclin_Asc.h***, while the latter can be found in the iLLD header ***IfxStdIf_DPipe.h***.

Run and Test

- For this training, a serial monitor is required for visualizing the values. The monitor can be opened inside the AURIX™ Development Studio using the following icon:


File Edit Source Refactor Navigate Search Project Debug Window Help

- The serial monitor must be configured with the following parameters to enable the communication between the board and the PC:
  - Speed (baud): 115200
  - Data bits: 8
  - Stop bit: 1



Serial port: COM29
Baud rate: 115200
Data size: 8
Parity: None
Stop bits: 1
Encoding: Default (ISO-8859-1)

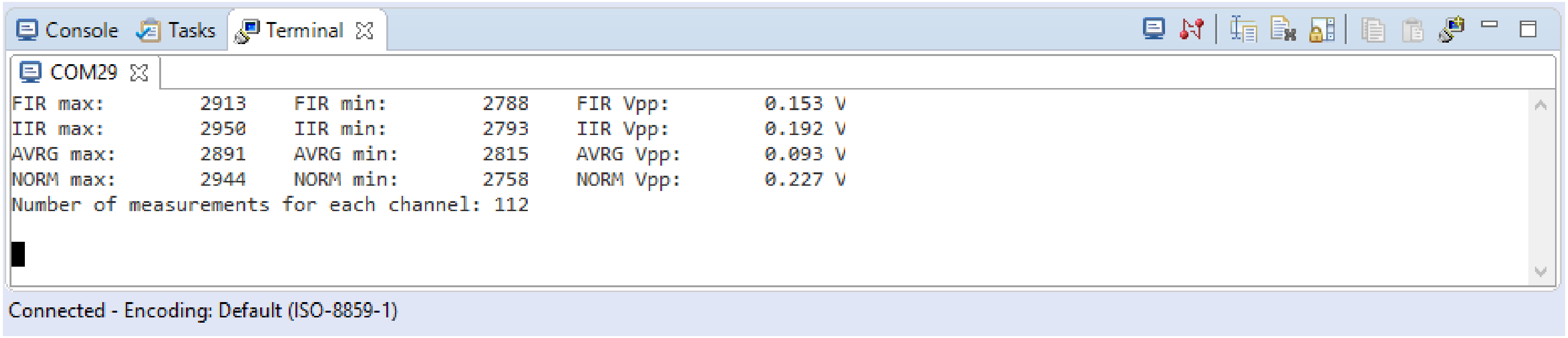
Run and Test

After code compilation and flashing the device, perform the following steps:

› Connect the channels AN0, AN3, AN13 and AN21 to the VCC_IN pin (~3.3V), or to any DC signal between 0 and 5V

› Open the serial monitor and start the serial communication, linked with the appropriate COMx port (this can be checked in the Device Manager)

› After a few seconds, send the character “1” to print the maximum and minimum values read by the channels, together with the computed V_{pp}



```
FIR max:        2913    FIR min:        2788    FIR Vpp:        0.153 V
IIR max:        2950    IIR min:        2793    IIR Vpp:        0.192 V
AVRG max:       2891    AVRG min:       2815    AVRG Vpp:       0.093 V
NORM max:       2944    NORM min:       2758    NORM Vpp:       0.227 V
Number of measurements for each channel: 112
```

Connected - Encoding: Default (ISO-8859-1)

The maximum and minimum values are expressed as a 12-bits integer value, in decimal format (0 - 4095 range), while the V_{pp} is expressed in Volts

It can be noticed that for this signal, the filter applying an average is the most effective one to reduce the V_{pp} range.

References

- AURIX™ Development Studio is available online:
- https://www.infineon.com/aurixdevelopmentstudio
- Use the „*Import...*“ function to get access to more code examples.

- More code examples can be found on the GIT repository:
- https://github.com/Infineon/AURIX_code_examples

- For additional trainings, visit our webpage:
- https://www.infineon.com/aurix-expert-training

- For questions and support, use the AURIX™ Forum:
- https://www.infineonforums.com/forums/13-Aurix-Forum

Revision history

Revision	**Description of change**
V1.0.1	Update of version to be in line with the code example’s version
V1.0.0	Initial version

Edition 2020-12
Published by
Infineon Technologies AG
81726 Munich, Germany

Do you have a question about this document?
Email: erratum@infineon.com

Document reference
ADC_Filtering_1_KIT_TC397_TFT