X2C

LINZ CENTER OF MECHATRONICS GMBH

Project Documentation Blinky_dsPIC33CK_Curiosity

July 24, 2020

Contents

Part I

X2C Model

1 Version Information

1.1 X2C

- X2C: Version 6.2.1984

1.2 Operating System

- OS: Windows 8 6.2

1.3 Scilab

- Scilab: Version 5.5.2.1427793548
- Java: Version 1.6.0_41

2 Model Structure

2.1 Xcos Model

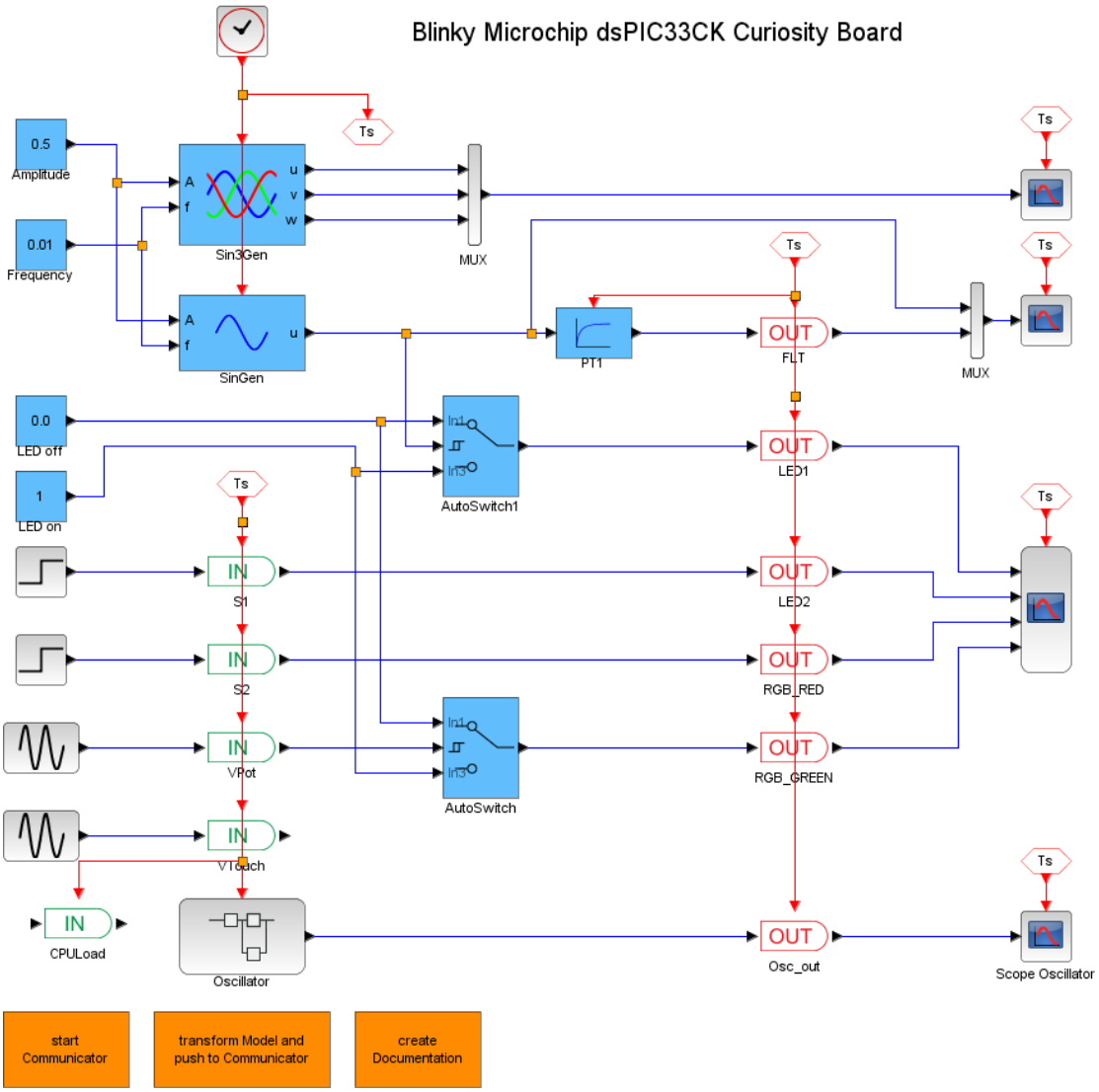


Figure 1: Blinky_dsPIC33CK_Curiosity

2.2 Subsystems

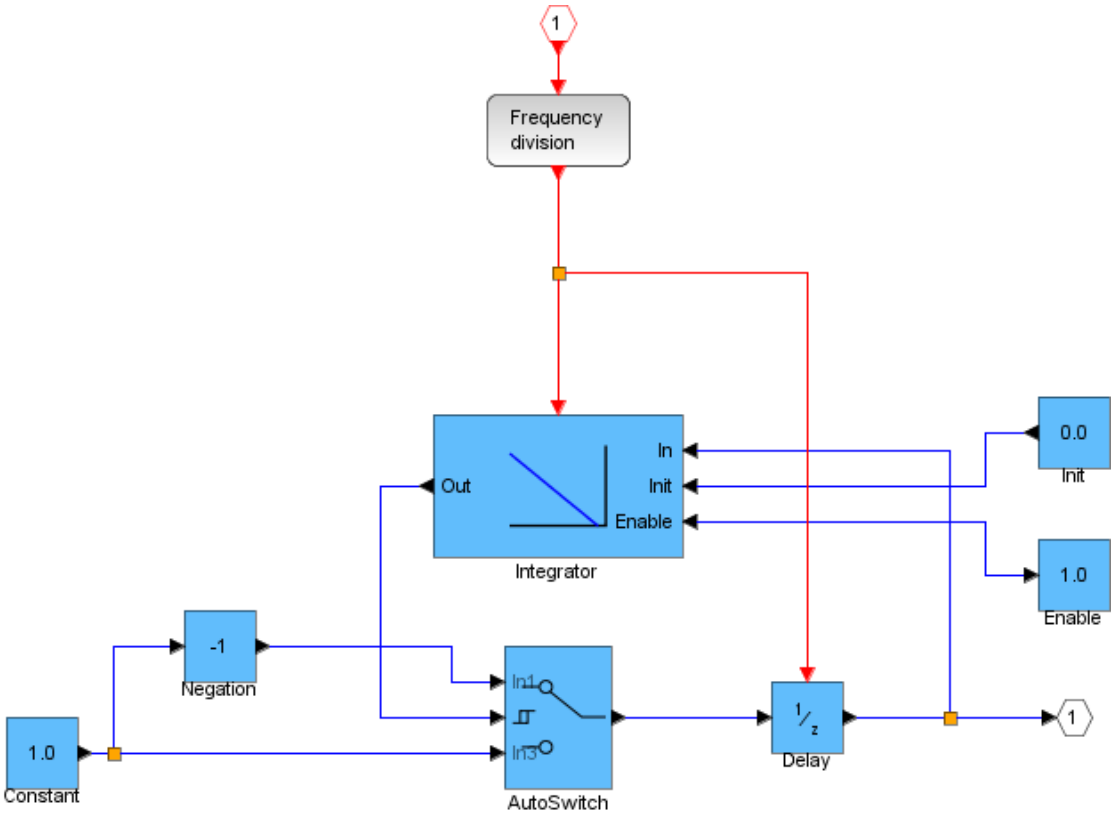


Figure 2: Blinky_dsPIC33CK_Curiosity_Oscillator

3 Model Parameter

3.1 Sample Time

Sample Time	
T_S	$100\mu s$

3.2 Scilab Parameter

```
// Scilab script file to store Model parameters
// This file is automatically executed by initProject.sce
// initScript.sce and this script is executed automatically, if model is opened from MPLAB X MCC

// Simulation settings
endTime      = 5;
stepSize     = 1.0E-2;
// X2C_sampleTime = 1.0E-4;

//****** User variables example *******//
// voltage = 24;
// maxCurrent = 10;
// powerLimit = voltage * maxCurrent;
```

Listing 1: ModelParameter.sce

4 Mask Parameter

Constant: Amplitude	
Value	0.5
Used Implementation	FiP16

AutoSwitch: AutoSwitch	
Thresh_up	0.3
Thresh_down	-0.3
Used Implementation	FiP16

AutoSwitch: AutoSwitch1	
Thresh_up	0.0
Thresh_down	0.0
Used Implementation	FiP16

Constant: Frequency	
Value	0.01
Used Implementation	FiP16

Constant: LED off	
Value	0.0
Used Implementation	FiP16

Constant: LED on	
Value	1.0
Used Implementation	FiP16

AutoSwitch: AutoSwitch	
Thresh_up	0.5
Thresh_down	-0.5
Used Implementation	FiP16

Constant: Constant	
Value	1.0
Used Implementation	FiP16

Delay: Delay	
ts_fact	3.0
Used Implementation	FiP16

Constant: Enable	
Value	1.0
Used Implementation	Bool

Constant: Init	
Value	0.0
Used Implementation	FiP16

I: Integrator	
Ki	30.0
ts_fact	3.0
Used Implementation	FiP16

Negation: Negation	
Used Implementation	FiP16

PT1: PT1	
V	1.0
fc	10.0
ts_fact	1.0
method	zoh
Used Implementation	FiP16

Sin3Gen: Sin3Gen	
fmax	1000.0
Offset	0.0
ts_fact	1.0
Used Implementation	FiP16

SinGen: SinGen	
fmax	1000.0
Offset	0.0
Phase	0.0
ts_fact	1.0
Used Implementation	FiP16

Part II
Frame Program Documentation

5 Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

6 Data Structure Documentation

6.1 _TMR_OBJ_STRUCT Struct Reference

6.1.1 Detailed Description

Section: Data Type DefinitionsTMR Driver Hardware Instance Object
@Summary Defines the object required for the maintenance of the hardware instance.
@Description This defines the object required for the maintenance of the hardware instance.
This object exists once per hardware instance of the peripheral.
Remarks: None.
The documentation for this struct was generated from the following file:

- tmr1.c

7 Example Documentation

7.1 C:/_bitbucket/x2c_demos/blinky_dspic33ck_curiosity/mcc_generated_files/reset.h

It handles the reset cause by clearing the cause register values. Its a weak function user can override this function.

Returns

None

```
RESET_CauseHandler();
/*
    (c) 2020 Microchip Technology Inc. and its subsidiaries. You may use this
    software and any derivatives exclusively with Microchip products.

    THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
    EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
    WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
    PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
    WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

    IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
    INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
    WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
    BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
    FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
    ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
    THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

    MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
    TERMS.
*/
#ifndef RESET_H
#define RESET_H
```

```
#include <stdint.h>
#include "reset_types.h"
uint16_t RESET_GetCause(void);
void __attribute__ ((weak)) RESET_CauseHandler(void);
void RESET_CauseClearAll();
#endif  /* RESET_H */
```

7.2 C:/_bitbucket/x2c_demos/blinky_dspic33ck_curiosity/mcc_generated_files/system.h

Initializes the CPU core control register.

```
SYSTEM_CORCONInitialize();
```

```
/*
    (c) 2020 Microchip Technology Inc. and its subsidiaries. You may use this
    software and any derivatives exclusively with Microchip products.

    THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
    EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
    WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
    PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
    WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

    IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
    INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
    WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
    BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
    FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
    ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
    THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

    MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
    TERMS.
*/
#ifndef _XTAL_FREQ
#define _XTAL_FREQ  100000000UL
#endif
#define WDT_CLR_KEY 0x5743
#include "xc.h"
#include "stdint.h"
#include "system_types.h"
#ifndef SYSTEM_H
#define SYSTEM_H
inline static void SYSTEM_CORCONInitialize()
{
    CORCON = (CORCON & 0x00F2) | CORCON_MODE_PORVALUES;    // POR value
}
inline static void SYSTEM_CORCONModeOperatingSet(SYSTEM_CORCON_MODES modeValue)
{
    CORCON = (CORCON & 0x00F2) | modeValue;
}
inline static void SYSTEM_CORCONRegisterValueSet(uint16_t value)
{
    CORCON = value;
}
inline static uint16_t SYSTEM_CORCONRegisterValueGet(void)
{
    return CORCON;
}
inline static uint32_t SYSTEM_DeviceIdRegisterAddressGet(void)
{
    return __DEVID_BASE;
}
void SYSTEM_Initialize(void);
#endif  /* SYSTEM_H */
```

7.3 C:/_bitbucket/x2c_demos/blinky_dspic33ck_curiosity/mcc_generated_files/watchdog.h

Enables Watch Dog Timer (WDT) using the software bit.

```
WATCHDOG_TimerSoftwareEnable();
```

```
/*
    (c) 2020 Microchip Technology Inc. and its subsidiaries. You may use this
    software and any derivatives exclusively with Microchip products.
```

```
    THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER
    EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED
    WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
    PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION
    WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

    IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE,
    INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND
    WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS
    BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
    FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN
    ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY,
    THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

    MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE
    TERMS.
*/
#ifndef WATCHDOG_H
#define WATCHDOG_H
#define WATCHDOG_CLR_KEY 0x5743
inline static void WATCHDOG_TimerSoftwareEnable(void)
{
    WDTCONLbits.ON = 1;
}
inline static void WATCHDOG_TimerSoftwareDisable(void)
{
    WDTCONLbits.ON = 0;
}
inline static void WATCHDOG_TimerClear(void)
{
    WDTCONH = WATCHDOG_CLR_KEY;
}
#endif  /* WATCHDOG_H */
```

Part III
Used X2C-Blocks

8 Project Specific Blocks

9 Internal Library Blocks

Block: AutoSwitch

Inports	
In1	Input #1
Switch	Input #2: Threshold signal
In3	Input #3

Outports	
Out	Either value of input #1 or input #3 dependent on value of input #2

Mask Parameters		
Name	**ID**	**Description**
Thresh_up	1	Threshold level for rising switch signal
Thresh_down	2	Threshold level for falling switch signal

Description:

Switch between In1 and In3 dependent on Switch signal:
Switch signal rising: Switch >= Threshold up –> Out = In1
Switch signal falling: Switch < Threshold down –> Out = In3

Implementations:

FiP16 16 Bit Fixed Point Implementation
FiP32 32 Bit Fixed Point Implementation
Float32 32 Bit Floating Point Implementation
Float64 64 Bit Floating Point Implementation

Implementation: FiP16

16 Bit Fixed Point Implementation

Inports Data Type	
In1	int16
Switch	int16
In3	int16

Outports Data Type	
Out	int16

Implementation: FiP32

32 Bit Fixed Point Implementation

Inports Data Type	
In1	int32
Switch	int32
In3	int32

Outports Data Type	
Out	int32

Implementation: Float32

32 Bit Floating Point Implementation

Inports Data Type	
In1	float32
Switch	float32
In3	float32

Outports Data Type	
Out	float32

Implementation: Float64

64 Bit Floating Point Implementation

Inports Data Type	
In1	float64
Switch	float64
In3	float64

Outports Data Type	
Out	float64

Block: Constant

Outports	
Out	Constant output

Mask Parameters		
Name	**ID**	**Description**
Value	1	Constant factor

Description:

Constant value.

Implementations:

Bool	Boolean Implementation
Int8	8 Bit Integer Implementation
Int16	16 Bit Integer Implementation
Int32	32 Bit Integer Implementation
FiP8	8 Bit Fixed Point Implementation
FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation
Float32	32 Bit Floating Point Implementation
Float64	64 Bit Floating Point Implementation

Implementation: Bool

Boolean Implementation

Outports Data Type	
Out	bool

Implementation: Int8

8 Bit Integer Implementation

Outports Data Type	
Out	int8

Implementation: Int16

16 Bit Integer Implementation

Outports Data Type	
Out	int16

Implementation: Int32

32 Bit Integer Implementation

Outports Data Type	
Out	int32

Implementation: FiP8

8 Bit Fixed Point Implementation

Outports Data Type	
Out	int8

Implementation: FiP16

16 Bit Fixed Point Implementation

Outports Data Type	
Out	int16

Implementation: FiP32

32 Bit Fixed Point Implementation

Outports Data Type	
Out	int32

Implementation: Float32

32 Bit Floating Point Implementation

Outports Data Type	
Out	float32

Implementation: Float64

64 Bit Floating Point Implementation

Outports Data Type	
Out	float64

Block: Delay

Inports	
In	Input In(k)

Outports	
Out	Output Out(k)=In(k-1)

Mask Parameters		
Name	**ID**	**Description**
ts_fact	1	Multiplication factor of base sampling time (in integer format)

Description:

Output delay by one sample time interval.

This block can be used to enable feedback loops in the model.

Implementations:

Bool	Boolean Integration
FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation
Float32	32 Bit Floating Point Implementation
Float64	64 Bit Floating Point Implementation

Implementation: Bool

Boolean Integration

Inports Data Type	
In	bool

Outports Data Type	
Out	bool

Implementation: FiP16

16 Bit Fixed Point Implementation

Inports Data Type	
In	int16

Outports Data Type	
Out	int16

Implementation: FiP32

32 Bit Fixed Point Implementation

Inports Data Type	
In	int32

Outports Data Type	
Out	int32

Implementation: Float32

32 Bit Floating Point Implementation

Inports Data Type	
In	float32

Outports Data Type	
Out	float32

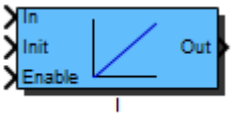
Implementation: Float64

64 Bit Floating Point Implementation

Inports Data Type	
In	float64

Outports Data Type	
Out	float64

Block: I



Inports	
In	Control error input
Init	Value which is loaded at initialization function call
Enable	Enable == 0: Deactivation of block; Out set to 0 Enable 0->1: Preload of integral part Enable == 1: Activation of block

Outports	
Out	Control value

Mask Parameters		
Name	**ID**	**Description**
Ki	1	Integral Factor
ts_fact	2	Multiplication factor of base sampling time (in integer format)

Description:

I controller:
G(s) = Ki/s = 1/(Ti*s)

Each fixed point implementation uses the next higher integer datatype for the integrational value storage variable.
A rising flank at the *Enable* inport will preload the integrational part with the value present on the *Init* inport.

Transfer function (zero-order hold discretization method):

$$G(z) = K_{\mathrm{i}} T_{\mathrm{s}} \frac{1}{z-1}$$

Implementations:

FiP8	8 Bit Fixed Point Implementation
FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation
Float32	32 Bit Floating Point Implementation
Float64	64 Bit Floating Point Implementation

Implementation: FiP8

8 Bit Fixed Point Implementation

Inports Data Type	
In	int8
Init	int8
Enable	bool

Outports Data Type	
Out	int8

Implementation: FiP16

16 Bit Fixed Point Implementation

Inports Data Type	
In	int16
Init	int16
Enable	bool

Outports Data Type	
Out	int16

Implementation: FiP32

32 Bit Fixed Point Implementation

Inports Data Type	
In	int32
Init	int32
Enable	bool

Outports Data Type	
Out	int32

Implementation: Float32

32 Bit Floating Point Implementation

Inports Data Type	
In	float32
Init	float32
Enable	bool

Outports Data Type	
Out	float32

Implementation: Float64

64 Bit Floating Point Implementation

Inports Data Type	
In	float64
Init	float64
Enable	bool

Outports Data Type	
Out	float64

Block: Negation

Inports	
In	Input

Outports	
Out	Negated input value

Description:

Negation of input signal.

Calculation:

$$\mathrm{Out} = -\mathrm{In}$$

Implementations:

FiP8	8 Bit Fixed Point Implementation
FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation
Float32	32 Bit Floating Point Implementation
Float64	64 Bit Floating Point Implementation

Implementation: FiP8

8 Bit Fixed Point Implementation

Inports Data Type	
In	int8

Outports Data Type	
Out	int8

Implementation: FiP16

16 Bit Fixed Point Implementation

Inports Data Type	
In	int16

Outports Data Type	
Out	int16

Implementation: FiP32

32 Bit Fixed Point Implementation

Inports Data Type	
In	int32

Outports Data Type	
Out	int32

Implementation: Float32

32 Bit Floating Point Implementation

Inports Data Type	
In	float32

Outports Data Type	
Out	float32

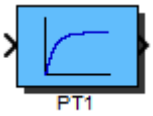
Implementation: Float64

64 Bit Floating Point Implementation

Inports Data Type	
In	float64

Outports Data Type	
Out	float64

Block: PT1



Inports	
In	Input In(k)

Outports	
Out	Output Out(k)

Mask Parameters		
Name	**ID**	**Description**
V	1	Gain
fc	2	Cut off frequency of low pass filter
ts_fact	3	Multiplication factor of base sampling time (in integer format)
method	4	Discretization method

Description:

First order low pass:
G(s) = V/(s/w + 1)

Due to limited value range in the 8 bit fixed point implementation rather high deviations from expected output values may occur.

9.0.0.1 Developer note:

The source code of block *TF1* is used.

Implementations:

FiP8 8 Bit Fixed Point Implementation
FiP16 16 Bit Fixed Point Implementation
FiP32 32 Bit Fixed Point Implementation
Float32 32 Bit Floating Point Implementation
Float64 64 Bit Floating Point Implementation

Implementation: FiP8

8 Bit Fixed Point Implementation

Inports Data Type	
In	int8

Outports Data Type	
Out	int8

Implementation: FiP16

16 Bit Fixed Point Implementation

Inports Data Type	
In	int16

Outports Data Type	
Out	int16

Implementation: FiP32

32 Bit Fixed Point Implementation

Inports Data Type	
In	int32

Outports Data Type	
Out	int32

Implementation: Float32

32 Bit Floating Point Implementation

Inports Data Type	
In	float32

Outports Data Type	
Out	float32

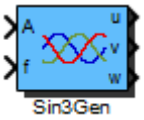
Implementation: Float64

64 Bit Floating Point Implementation

Inports Data Type	
In	float64

Outports Data Type	
Out	float64

Block: Sin3Gen



Inports	
A	Amplitude
f	Frequency

Outports	
u	Sine wave output phase u
v	Sine wave output phase v
w	Sine wave output phase w

Mask Parameters		
Name	**ID**	**Description**
fmax	1	Maximum Frequency in Hz
Offset	2	Offset
ts_fact	3	Multiplication factor of base sampling time (in integer format)

Description:

Generation of a 3 sine waves with amplitude (A) and frequency (f).

Calculation fixed point implementation:

$$
\begin{aligned}
u_k &= A_k \sin\left(2 f_k f_{\max} k T_\mathrm{s}\right) + A_\mathrm{offset} \\
v_k &= A_k \sin\left(2 f_k f_{\max} k T_\mathrm{s} - \frac{2\pi}{3}\right) + A_\mathrm{offset} \\
w_k &= A_k \sin\left(2 f_k f_{\max} k T_\mathrm{s} + \frac{2\pi}{3}\right) + A_\mathrm{offset}
\end{aligned}
$$

For sine calculation a lookup table with 256 entries is used. This results in a short computation time but with the downside of reduced accuracy for the FiP32 implementation.

Calculation floating point implementation (parameter *f_max* is ignored):

$$
\begin{aligned}
u_k &= A_k \sin\left(2\pi f_k k T_\mathrm{s}\right) + A_\mathrm{offset} \\
v_k &= A_k \sin\left(2\pi f_k k T_\mathrm{s} - \frac{2\pi}{3}\right) + A_\mathrm{offset} \\
w_k &= A_k \sin\left(2\pi f_k k T_\mathrm{s} + \frac{2\pi}{3}\right) + A_\mathrm{offset}
\end{aligned}
$$

Implementations:

FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation
Float32	32 Bit Floating Point Implementation
Float64	64 Bit Floating Point Implementation

Implementation: FiP16

16 Bit Fixed Point Implementation

Inports Data Type	
A	int16
f	int16

Outports Data Type	
u	int16
v	int16
w	int16

Implementation: FiP32

32 Bit Fixed Point Implementation

Inports Data Type	
A	int32
f	int32

Outports Data Type	
u	int32
v	int32
w	int32

Implementation: Float32

32 Bit Floating Point Implementation

Inports Data Type	
A	float32
f	float32

Outports Data Type	
u	float32
v	float32
w	float32

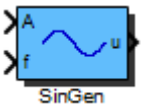
Implementation: Float64

64 Bit Floating Point Implementation

Inports Data Type	
A	float64
f	float64

Outports Data Type	
u	float64
v	float64
w	float64

Block: SinGen



Inports	
A	Amplitude
f	Frequency

Outports	
u	Sine wave output

Mask Parameters		
Name	**ID**	**Description**
fmax	1	Maximum Frequency in Hz
Offset	2	Offset
Phase	3	Phase [-Pi..Pi]
ts_fact	4	Multiplication factor of base sampling time (in integer format)

Description:

Generation of a sine wave with amplitude (A) and frequency (f).

Calculation fixed point implementation:

$$u_k = A_k \sin\left(2 f_k f_{\max} k T_\mathrm{s} + \phi_\mathrm{phase}\right) + A_\mathrm{offset}$$

For sine calculation a lookup table with 256 entries is used. This results in a short computation time but with the downside of reduced accuracy for the FiP32 implementation.

Calculation floating point implementation (parameter *f_max* is ignored):

$$u_k = A_k \sin\left(2\pi f_k k T_\mathrm{s} + \phi_\mathrm{phase}\right) + A_\mathrm{offset}$$

Implementations:

FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation
Float32	32 Bit Floating Point Implementation
Float64	64 Bit Floating Point Implementation

Implementation: FiP16

16 Bit Fixed Point Implementation

Inports Data Type	
A	int16
f	int16

Outports Data Type	
u	int16

Implementation: FiP32

32 Bit Fixed Point Implementation

Inports Data Type	
A	int32
f	int32

Outports Data Type	
u	int32

Implementation: Float32

32 Bit Floating Point Implementation

Inports Data Type	
A	float32
f	float32

Outports Data Type	
u	float32

Implementation: Float64

64 Bit Floating Point Implementation

Inports Data Type	
A	float64
f	float64

Outports Data Type	
u	float64