LINZ CENTER OF MECHATRONICS GMBH

Documentation

March 3, 2017

Contents

Part I
Installation

1 Software versions

Following software versions were tested for full *X2C* functionality:

Software	Version
Required:	
Scilab (www.scilab.org)	5.5.x
Java Runtime Environment	6
Optional (for documentation):	
MiKTeX (www.miktex.org)	2.9
Doxygen (www.doxygen.org)	1.8.10
Graphviz (www.graphviz.org)	2.38
Optional (for programming):	
Texas Instruments Code Composer Studio	5.5.x
Texas Instruments Code Generation Tools	6.1.6
Keil μVision	4.x
Microchip MPLAB X IDE	3.x
Microchip Compiler XC16	1.25

Different versions of these programs may work but without warranty.

2 Setup with *Scilab/Xcos* support

2.1 Installation

1. Open *Scilab/Xcos* and with the *File Browser* navigate to <X2C_ROOT>\System\Scilab\Scripts. Right click on **setup.sce** and click *Execute in Scilab*.
2. Restart *Scilab/Xcos*
3. The setup command creates a *X2C* configuration file which will automically load *X2C* libraries and palettes at startup of *Scilab/Xcos* .

2.2 Deinstallation

1. Open *Scilab/Xcos* and execute the command `initX2C(%f)` in the *Scilab/Xcos* console.
2. Restart *Scilab/Xcos*
3. Once above command was exeeuted, the *X2C* configuration file is deleted and *Scilab/Xcos* will not load any *X2C* libraries or palettes anymore.

For the unlikely event that Scilab freezes at startup and remains in a deadlock state, the deinstallation can be done manually by deleting the file **scilab.ini** located in the Scilab home directory (for Windows typically C:\Users\<your user name>\AppData\Roaming\Scilab\scilab-5.x.x).

3 Configuration of *Code Composer Studio*

3.1 Install the TI v6.1.6 compiler

It is necessary to use the compiler version TI v6.1.6 in *Code Composer Studio* in combination with *X2C* . Navigate to **Project** → **Properties** click **General** and in the **Advanced settings** area see what compiler versions are available. It is necessary to use the compiler version **TI v6.1.6**. If this version is not selectable go to **Help** → **Install new Software** and in the **Work with** drop down menu choose **Code Generation Tools Update**. In the section **TI Compiler Updates** find *C2800 Compiler Tools Version 6.1.6* and mark it as seen in figure 1. Klick **Next** and install the update. Now go back to the Project Properties and change the compiler.

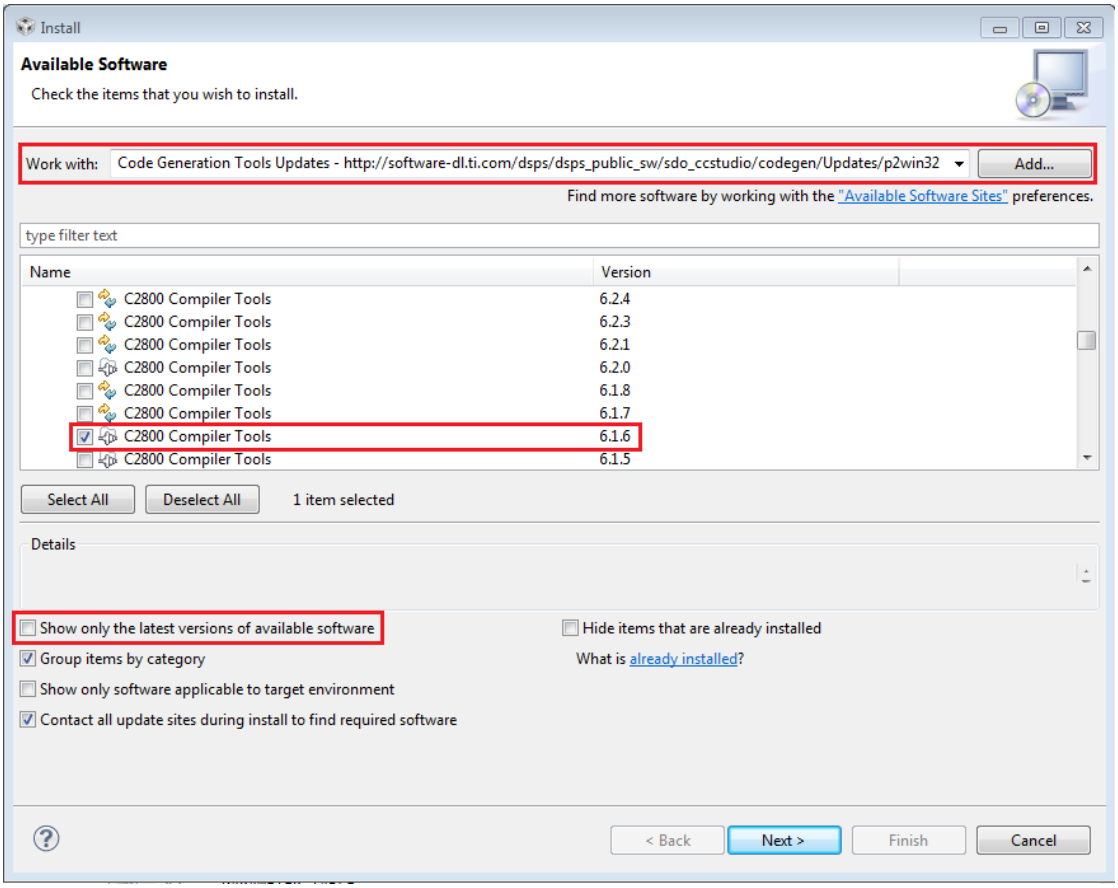


Figure 1: *Code Composer Studio* Compiler Download

3.2 *Texas Instruments* target processor types

3.2.1 Supported processors families

Currently the following *Texas Instruments* processor families are supported by *X2C* .

- TI C28x 32-Bit CPU
- TI TM4C12x 32-Bit CPU (ARM Cortex-M4 core)

3.2.2 Change target processor in *Code Composer Studio*

In the following section file names may vary with different processor types.

1. Import the *Blinky* demo application in *Code Composer Studio* (see section 14 for more information).

2. Change the *Predefined symbols* (see section 3.3) suitable for the used processor type.

3. With the *OS* file browser navigate to the *controlSUITE* subdirectory \device_support and search for your processor type
(e.g. C:\ti\controlSUITE\device_support\f2806x\v130\F2806x_headers).

4. Copy the folders *cmd, include* and *source* into the project directory <PROJECT_DIRECTORY>\TexasInstruments and replace the existing folders from the processor used in the *Blinkdy* demo application.

5. In *Code Composer Studio* open the **F28xxx_Device.h** file in the folder <PROJECT_DIRECTORY>\TexasInstruments\include. In the section *User To Select Target Device* search for your processor and change the 0 to `TARGET`. An example is shown in figure 2

```
57 #define   DSP28_28067P     0
58 #define   DSP28_28067UP    0
59 #define   DSP28_28067PZ    0
60 #define   DSP28_28067UPZ   0
61
62 #define   DSP28_28068P     0
63 #define   DSP28_28068UP    0
64 #define   DSP28_28068PZ    0
65 #define   DSP28_28068UPZ   0
66
67 #define   DSP28_28069P     0
68 #define   DSP28_28069UP    0
69 #define   DSP28_28069PZ    0
70 #define   DSP28_28069UPZ   TARGET
```

Figure 2: Change processor type in the *device.h* file

6. In *Code Composer Studio* open the files *Hardware.h* and *X2cDataTypes.h* and adapt the file names in the *include* directives for the *F28xxx_Device.h* file.

3.3 Change predefined Symbols

The *X2C* project uses predefined symbols to give the preprocessor information before compiling the project. Navigate to **Project** → **Properties** open **Build** → **C2000 Compiler** → **Advance Options** → **Predefined Symbols**.
Currently three different processor families can be choosen

- `__GENERIC_TI_C28X__` for *Texas Instruments* Processors
- `__GENERIC_ARM_ARMV7__` for *ARM* Processors
- `__GENERIC_MICROCHIP_DSPIC__` for Microchip Processors

The definition

- `__CUSTOM_DATATYPE_DEFINITIONS__`

is needed to avoid compiler warnings caused by multiple *typedefs*.

In addition the definition

- `SCOPE_SIZE=8000`

like seen in figure 3 needs to be made. The value of *Scope Size* is changeable and depends on the intended application and the used target processor. In the *Blinky* demo applications these values are already defined.

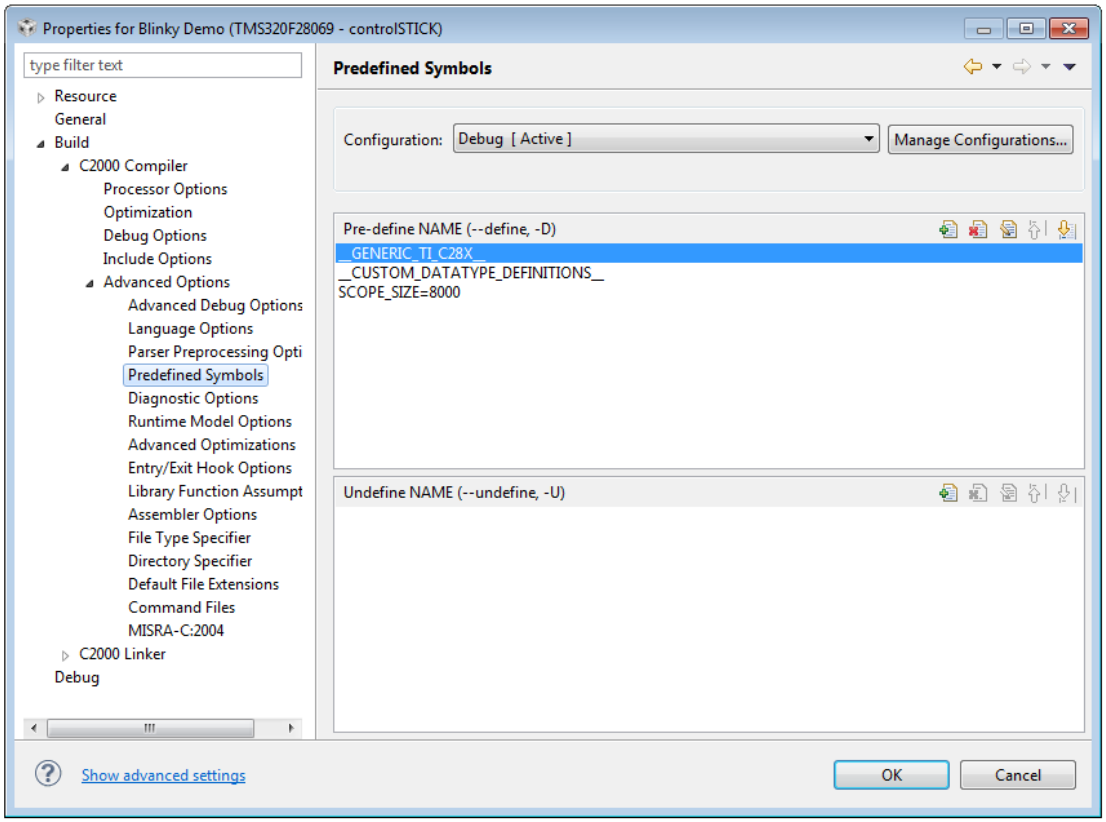


Figure 3: Predefined symbols for generic processor type in *Code Composer Studio*

4 Configuration of *MPLAB X*

4.1 Install the XC16 compiler

When working with *MPLAB X*the compiler to build the project has to be installed manually. Which compiler is needed depends on the used processor type. In the demo application *Blinky* the *xc 16 v1.21* compiler from the *Microchip* web page (http://www.microchip.com/pagehandler/en_us/devtools/mplabxc/) can be used.

4.2 *Microchip* target processor types

4.2.1 Supported processors families

Currently the following *Microchip* processor families are supported by *X2C* .

- dsPIC 16-Bit CPU

4.2.2 Change target processor in *MPLAB X*

Right click on the **Project** → **Properties**. In the *Configuration* area *Devices* can be picked in the drop down menu. Click **OK** to save the changes.

4.3 Change predefined Symbols

The *X2C* project uses *predefined Symbols* to give the preprocessor information before compiling the project. In the the sample project these symbols are already defined. Right click on the **Projectname** → **Properties** → **XC16 Global Options** → **xc16-gcc** to eventually change them. In the section *Define C macros* a list of defines is available as seen in figure 4. Depending on the used target processor three different processor families can be choosen

- `__GENERIC_TI_C28X__` for *Texas Instruments* Processors
- `__GENERIC_ARM_ARMV7__` for *ARM* Processors
- `__GENERIC_MICROCHIP_DSPIC__` for Microchip Processors

In addition the definition

- `SCOPE_SIZE=5000`

like seen in figure 4 needs to be made. The value of *Scope Size* is changeable and depends on the intended application and the used target processor. In the *Blinky* demo applications these values are already defined.

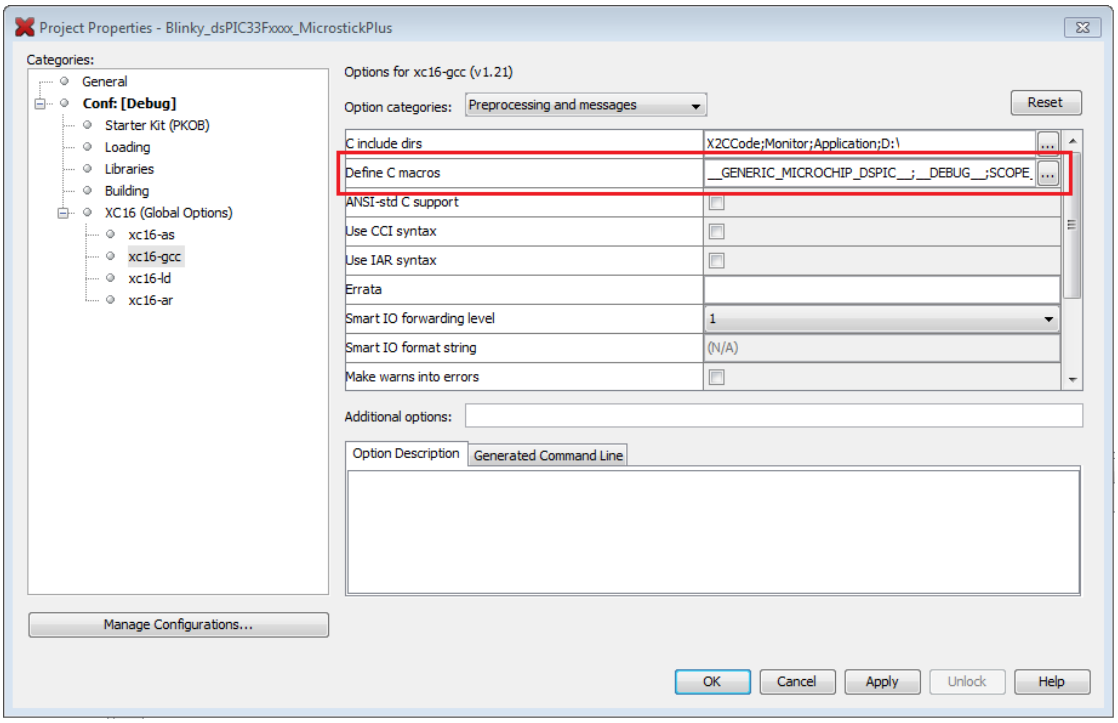


Figure 4: Predefined Symbols in *MPLAB X*

Part II
General

5 Introduction to X2C

5.1 Fixed point data representation

5.1.1 Standard signals

Standard signals are symmetrically scaled about zero and their scaling range is]-1...1[. In a case of an overflow the signal will be limited to 1 (or -1 respectively). For example a subtraction of a signal with value 0.5 from a signal with value -0.7 will lead to a signal with value -1.
Depending on the chosen implementation the values are handled in one of the following formats:

FiP8	
Implementation type	8-bit fixed point
Format	Q7
Minimum value	$-0.992\,187\,500$
Maximum value	$0.992\,187\,500$
Resolution	$0.007\,812\,500$

FiP16	
Implementation type	16-bit fixed point
Format	Q15
Minimum value	$-0.999\,969\,482\,421\,875$
Maximum value	$0.999\,969\,482\,421\,875$
Resolution	$0.000\,030\,517\,578\,125$

FiP32	
Implementation type	32-bit fixed point
Format	Q31
Minimum value	$-0.999\,999\,999\,534\,339$
Maximum value	$0.999\,999\,999\,534\,339$
Resolution	$0.000\,000\,000\,465\,661$

5.1.2 Unlimited/Unbalanced signals

The scaling of unlimited/unbalanced signals is [-1...1[. While the standard signals omit one value to achieve a symmetrical value range, the unlimited (or also called unbalanced) signals utilize the full value range which leads to a slightly unbalanced value range. As the name implies unlimited signals won't be limited. In fact unlimited signals utilize wrapping/overflow functions of the DSP. For example a subtraction of a signal with value 0.5 from a signal with value -0.7 will lead to a signal with value 0.8.

So the primary use of the unlimited signal is as angular signal where (-1...1) corresponds to (-π...π).
Depending on the chosen implementation the values are handled in one of the following formats:

FiP8	
Implementation type	8-bit fixed point
Format	Q7
Minimum value	$-1.000\,000\,000$
Maximum value	$0.992\,187\,500$
Resolution	$0.007\,812\,500$

FiP16	
Implementation type	16-bit fixed point
Format	Q15
Minimum value	$-1.000\,000\,000\,000\,000$
Maximum value	$0.999\,969\,482\,421\,875$
Resolution	$0.000\,030\,517\,578\,125$

FiP32	
Implementation type	32-bit fixed point
Format	Q31
Minimum value	$-1.000\,000\,000\,000\,000$
Maximum value	$0.999\,999\,999\,534\,339$
Resolution	$0.000\,000\,000\,465\,661$

5.2 Floating point data representation

5.2.1 Standard signals

The standard signals in floating point format are not restricted, the full value range according to the IEEE 754 standard is available.

Float32	
Implementation type	32-bit floating point
Format	IEEE 754
Minimum value	$-3.40282346638528859811704183484 52e+38$
Maximum value	$3.40282346638528859811704183484 52e+38$
Resolution	$\pm 1.17549435082228750796873653722 22e-38$ (normalized)

Float64	
Implementation type	64-bit floating point
Format	IEEE 754
Minimum value	$-1.797693134862315708145274237317e+308$
Maximum value	$1.797693134862315708145274237317e+308$
Resolution	$\pm 2.225073858507201383090232717332 4e-308$ (normalized)

5.2.2 Unlimited/Unbalanced signals

Contrary to their names the unlimited/unbalanced signals in floating point format are limited to $[-\pi, +\pi]$. All other properties are similar to the unlimited signals in fixed point format.

Float32	
Implementation type	32-bit floating point
Format	IEEE 754
Minimum value	$-3.14159265358979323846264338327 95$
Maximum value	$3.14159265358979323846264338327 95$
Resolution	$\pm 1.175494350822287507968736537222 2e-38$ (normalized)

Float64	
Implementation type	64-bit floating point
Format	IEEE 754
Minimum value	$-3.14159265358979323846264338327 95$
Maximum value	$3.14159265358979323846264338327 95$
Resolution	$\pm 2.225073858507201383090232717332 4e-308$ (normalized)

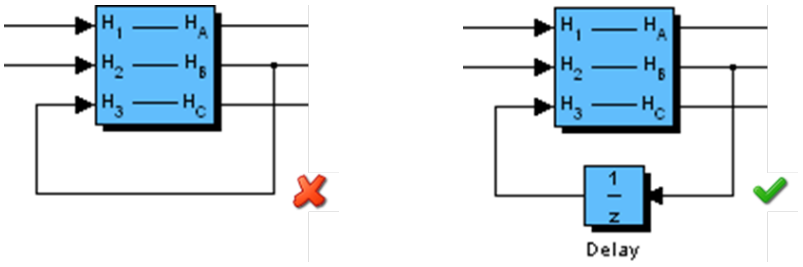
5.3 Restrictions

5.3.1 Algebraic loops

Algebraic loops as depicted in figure 5a are not possible due to a execution order problem. Therefore a block is required which breaks the loop at a specific position. This can be achieved by inserting a block with no direct feedthrough functionality, e.g. Block: LoopBreaker from the *General*-library or Block: Delay from the *Control*-library (see figure 5b).

5.3.2 Connection of blocks with different implementations

Though blocks with different implementations are allowed (and computed correctly) in the same model, connections of ports with different datatypes are not permitted. The conversion blocks Block: TypeConv, Block: Int2Real and Block: Real2Int can be used to resolve datatype incompatibilities.



(a) forbidden (b) allowed

Figure 5: Algebraic loops

6 Basic structure of the C Code

When setting up a new project with *X2C* code generation it is necessary to configure the hardware on the target system. The following section provides basic information how the *Blinky* demo applications are structured. With this understanding one should be able to adapt hardware configuration for further projects.
In the following the *Blinky* demo application in combination with the *TI Piccolo F28069 ControlSTICK* and the *TMS320F28069* Processor is used for demonstration.
Info: The *.c files listed below need to be updated in case of changes in the *Scilab/Xcos* model configuration.

6.1 Main.c

- **initInterruptVector()** defined in *Hardware.c* configures the target specific interrupts.
- **initSerial()** defined in *Hardware.c* initializes the serial interface.
- **initHardware()** defined in *Hardware.c* here the peripheral devices such as *Watchdog, GPIO Ports, ADCs, Timers* and others are defined.
- **X2C_init()** defined in *X2C.c* calls the initialization functions of the *X2C* blocks.
- The **while(1)** loop is mainly used for the serial communication.
- The **mainTask()** function is the key structure of the project. Here the connection between *Outports* and *Inports* are defined and the *X2C_Update()* function (defined in *X2C.c*) is called. The basic structure of the *mainTask()* is

  1. Assign Inports
  2. Call *X2C_Update()*
  3. Update Outports

  The *mainTask()* function is usually called by an *Interrupt Service Routine (Isr)* which can be triggered by multiple sources.
  Example: In the *Blinky Demo Application* after each *ADC* conversion cycle the *ADCIsr* calls the *mainTask()* function.
- **KICK_DOG** resets the *Watchdog* timer periodically. During operation this timer continuously counts a certain time span (configured in *Hardware.c*). If the application has an unexpected failure *KICK_DOG* can not be called and the *Watchdog* timer exceeds its limit and therefore the target reboots. If the operation executes as expected the *Watchdog* timer is within the limit and can be reseted by *KICK_DOG* without any further actions.

6.2 Hardware.c

In this file all connections and peripheral device settings should be made.

- In **initHardware()** all peripheral function should be initialized.
  - Watchdog timer
  - GPIO Ports
  - Interrupt initialization
  - ADC, Timer, PWM and all the other peripheral devices
- **initSerial()** initializes the serial interface on the target. The settings made here should match with the setting made in the *Communicator* described in section 10.

7 Testing

7.1 JUnit tests

To minimize the risk of software bugs most parts of X2C are tested. The Java core of X2C is tested wit JUnit tests.

7.2 CUnit tests

Much care is also taken of testing the C-code of the blocks. These so called CUnit tests are conducted directly on the target. In figure 6 the test setting can be seen.

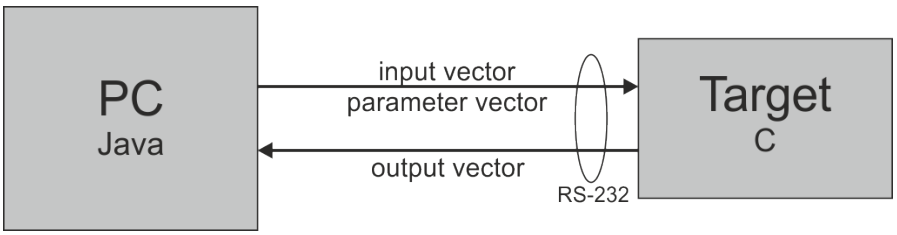


Figure 6: CUnit test setting

In the test environment on the PC a input vector and a parameter vector, if necessary, are defined and sent to the target via serial interface. On the target the update function of the block under test is executed. The resulting output vector is transferred back to the PC where it is compared with a reference output vector. If the difference between the actual and the reference output vector is below a specified limit the test is marked as passed. Otherwise an error is reported.
Basically, the generation of the test vectors are done with one or a combination of these methods:

Equivalence class testing To reduce the number of test items they are grouped into classes with same behavior. Only one member of each equivalence class is used as entry for the test vector. Two simple equivalence classes could be positive and negative numbers.

Boundary testing The entries for the test vectors are chosen in such a way that they lie below and above critical boundaries. For example, typical values for a FiP16 implementation would be -32768, -32767, -1, 0, 1, 32767.

Back-to-back testing For complex blocks this method is used. The vectors are generated by simulating a block or model with the same functionality in Matlab/Simulink or Scilab/Xcos.

Several different targets are used for testing. The test reports can be found in the library documentation *Library.source.pdf* in the directory *<X2C_ROOT>\Library*.

8 Coding Conventions

8.1 Language

The native language of X2C is English. Hence all documentation, file names, variables, comments in source files, etc. should be in English.

8.2 General naming conventions

- Unless otherwise stated, all names should use the camel case notation. A definition of camel case can be found on http://en.wikipedia.org/wiki/CamelCase. The type of camel case (upper or lower) depends on the type of name, see sections below.

  Examples: `ThisIsUpperCamelCasing.java, showLowerCamelCaseExample()`

- Non-ASCII characters should be avoided. Also the space character schould not be used.
- Due to a character limitation in Scilab, names with more than 27 characters should be avoided.
- Names should not start with a number.
- Hungarian notation should not be used.

8.3 Naming of files

In general files should have a meaningful name. If abbreviations are used, easy understandable ones should be used. Upper camel case is recommended.
Examples: `Hardware.c`, `SystemControl.c`, `GlobalDefines.h`

8.4 Naming of functions and methods

Function and method names should contain a verb to describe the action of the function. The verb is placed first and is in lower case and subsequent nouns start with a capital letter (lower camel case).
Examples: `readADC()`, `setPWM()`

8.5 Naming of macros

Macros should be written in capital letters. Macro names which contain mutiple words should use a underscore as seperator (screaming snake case).
Examples: `DISABLE_PWM, NO_ERROR`

8.6 Naming of variables

Based on the proposed convention from Sun Microsystems following guidelines should be considered:
Variables are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.
Examples: `int16 i; float32 myWidth;`

8.7 Naming of model parameters

Parameter and variable names in Matlab or Scilab should follow lower snake case notation. This means the first word can either start with a lower or upper case letter, all subsequent words have to start with a lower case letter, and the words are seperated by underscores.
Examples: `i_ref`, `n_max`, `k_T`, `U_dc_max`

8.8 Naming of X2C blocks

Block and Subsystem/Superblock names in Matlab or Scilab should follow upper camel case notation. Exceptions are words with abbreviations, then a underscore character as seperator is allowed to increase readability.
Examples: `CurrentController`, `OffsetAngle`, `AnIn1`, `DigOut1`, `I_Phase1`

8.9 Source and header files

Every *.c source file should have a corresponding *.h header file. A minimalistic header with prototype definitions is sufficient.
Due to MISRA rule 8.1 (see MISRA-C 2004 compliance) it is required to have prototypes for every function, including static ones. To avoid conflicts between global and static function prototypes when including header files, the following rule shall apply:

- Global function prototypes should be located in its header file
- Static function prototypes should be located at the beginning of its source file

8.10 Global definitions

Definition of macros which are used in more than one source file should be placed in `GlobalDefines.h`. Macros only used in one file should be defined in the source file in which they are used.
Globally needed variables should be defined in `GlobalDefines.c` and declared in `GlobalDefines.h`. This way all global variables are at one place and can be referenced from every file which has the `GlobalDefines.h` header file included.
Example:

Listing 1: GlobalDefines.c

```c
#include "GlobalDefines.h"

/*****************************************************************************/
/* Global Variables                                                          */
/*****************************************************************************/
uint16 errorstate  = NO_ERROR;            /* Error message */
uint32 modulestate = NO_ERROR;            /* Module status */
FIStates FIState = RESET_STATE;           /* State of frequency inverter */
```

Listing 2: GlobalDefines.h

```c
#ifndef _GLOBALDEFINES_H_
  #define _GLOBALDEFINES_H_

  #include "Target.h"
  #include "X2C.h"

  /*****************************************************************************/
  /* Global Variables                                                          */
  /*****************************************************************************/
  extern uint16 errorstate;           /* Error message */
  extern uint32 modulestate;            /* Module status */
  extern FIStates FIState;            /* State of frequency inverter */

#endif
```

8.11 Template files

For an easy orientation standard file names should be used in X2C projects:

- `Main.c`: Frame program main file.
- `Hardware.c`: Hardware configuration and initialization.
- `GlobalDefines.*`: Files with globally needed definitions and variables.
- `SystemControl.c`: File with startup sequence of power electronics and error handling.
- `InterruptControl.c` Interrupt handling, especially interrupt vector table and interrupt service routines.
- `InputControl.c`: Handling of analog and digital inputs.
- `OutputControl.c`: Handling of analog and digital outputs.
- `CANControl.c`: Configuration, initialization and functions of a CAN interface.

Of course, if files contain a lot of code it is recommended to split the file into several ones to maintain comprehensibility.
Example: Splitting of `InputsControl.c` in `AnalogInputControl.c`, `DigitalInputControl.c` and `HallSensorControl.c`

8.12 Include order of header files

To avoid conflicts and missing dependencies header files should be included in following order:

1. System headers
2. Application headers
3. Header of current source file

Example:

Listing 3: Main.c

```
1 #include "VersionInfo.h"
2 #include "GlobalDefines.h"
3 #include "Hardware.h"
4 #include "SystemControl.h"
5 #include "InputControl.h"
6 #include "OutputControl.h"
7 #include "Main.h"
```

8.13 Hardware registers

To maintain comprehensibility and to allow interchangeability of source files hardware (DSP) registers should be accessed via macros.
Example:

```
#define SET_LED     (GPBSET = 0x00000004)

/* some code */
SET_LED;
/* some more code */
```

instead of

```
/* some code */
GPBSET = 0x00000004;
/* some more code */
```

9 MISRA-C 2004 compliance

The rules of the Motor Industry Software Reliability Association MISRA should be followed as much as possible. Some major rules are:

- **MISRA-C:2004 2.2/R:** Source code shall only use /* ... */ style comments.
- **MISRA-C:2004 19.4/R:** C macros shall only expand to a braced initialiser, a constant, a string literal, a parenthesised expression, a type qualifier, a storage class specifier, or a do-while-zero construct.
- **MISRA-C:2004 8.12/R:** When an array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialisation.
- **MISRA-C:2004 18.4/R:** Unions shall not be used.
- **MISRA-C:2004 16.3/R:** Identifiers shall be given for all of the parameters in a function prototype declaration.
- **MISRA-C:2004 19.15/R:** Precautions shall be taken in order to prevent the contents of a header file being included twice.
- **MISRA-C:2004 19.1/A:** #include statements in a file should only be preceded by other preprocessor directives or comments.
- **MISRA-C:2004 8.1/R:** Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call.

9.1 Applied rules

```
1  /* Enable MISRA-C:2004 checking (all rules) */
2  --check_misra="all"
3
4  /* Rule violation handling */
5  --misra_advisory="warning"
6  --misra_required="warning"
7
8  /* Exceptions */
9  --check_misra="-1.1"  /* (MISRA-C:2004 1.1/R)   Ensure strict ANSI C mode (-ps)
       is enabled */
10 --check_misra="-12.7" /* (MISRA-C:2004 12.7/R)  Bitwise operators shall not be
      applied to operands whose underlying type is signed */
11 --check_misra="-19.7" /* (MISRA-C:2004 19.7/A)  A function should be used in
      preference to a function-like macro */
12 --check_misra="-5.7"  /* (MISRA-C:2004 5.7/A)   No identifier name should be
      reused */
```

Listing 4: MISRA.opt

Part III
Utilities

10 Communicator

The *Communicator* is the interface between the target system and the model in *Scilab/Xcos* . It is used to create the C code in the *X2C.c* and *X2C.h* files out of the model. Furthermore it is used to transfer data between the computer and the target. When started the *Communicator* is connected with the model via *RMI* interface and via serial interface with the target (DSP).

10.1 *Scilab/Xcos Communicator* start

As described in section 13 the *Communicator* is started out of an open *Scilab/Xcos* model with the buttons *start Communicator*. The button *Transform model and push to Communicator* loads the model file (*.xml*) into the *Communicator*. Changes in the model structure can be made and pushed to the *Communicator* by double clicking on the button *Transform model and push to Communicator*.

10.2 Standalone *Communicator* start

If there are no intended changes in the model structure it is possible to start the *Communicator* without *Scilab/Xcos* . In <X2C_ROOT>\System\Java double click on **Communicator.jar**. In the open *Communicator* go to **Model** $\rightarrow$ **Load Model** and browse to your project directory. In the *X2CCode* folder choose the model (*.xml*) file and open it. In the *Status* tab check the *Log* area if the model has been loaded successfully.

10.3 Basic functions of the *Communicator*

The *Communicator* is structured into the menu bar (1) the basic function buttons (2) and three main tabs (3).

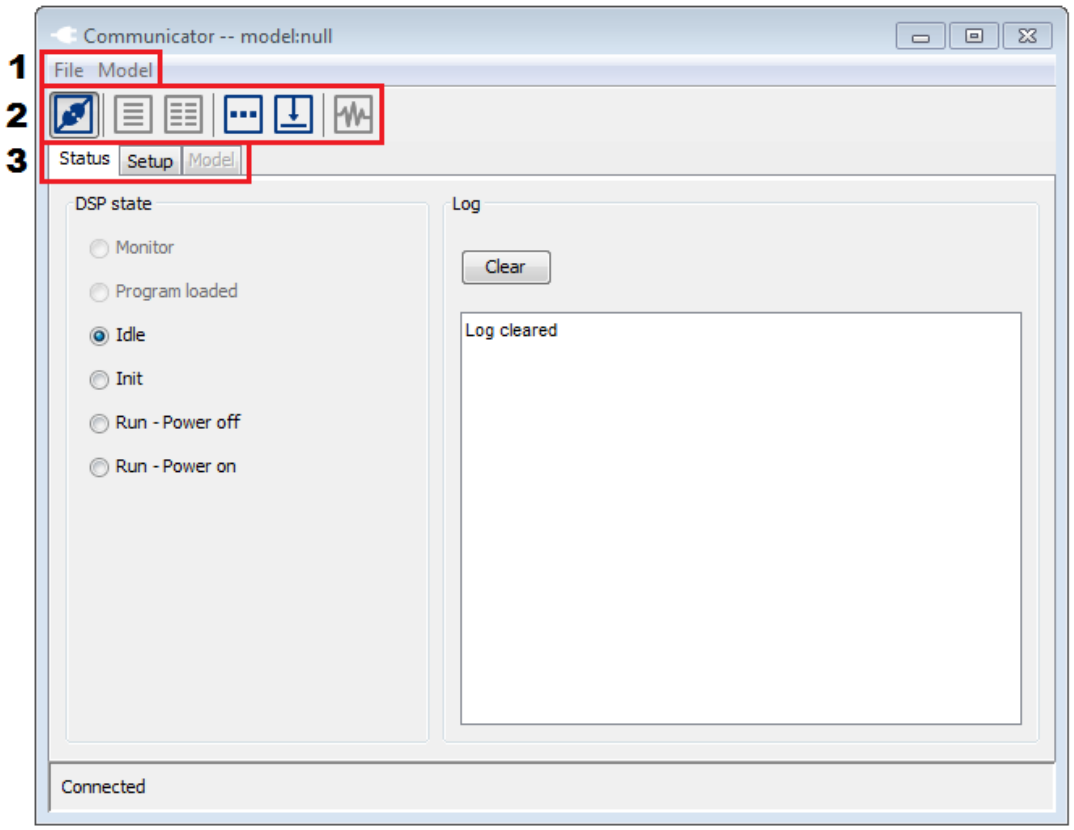


Figure 7: Basic structure of the *Communicator*

1. Menu bar
   - In the **File** menu the settings can be modified, loaded and saved.
   - In the **Model** menu a new *.xml* file can be loaded and saved.
2. Function buttons
   - With the button **Connect to Target** the *Communicator* can be connected respectively disconnected from the target.
   - **Create Code** generates the *X2C.c* and *X2C.h* files out of the *X2C* model. Changes in the *Model* tab like *Sample time* require new code creation.
   - **Create RTOS Code** was moved to *Settings*. See 10.4 for details.
   - In the **Download settings** the *.hex* and *.map* files out of the C code build process can be loaded. These files are needed for two functions:
     (a) In the full version of *X2C* the *Communicator* needs these files for flashing the code on the target through the serial interface. This function is not available in the free version.
     (b) In the full version and the free version of *X2C* these files (especially the *.map* file) are needed for block data transfer. For more information see number 3.
   - The **Download application to target** function is only available in the full version of *X2C* . This function provides program flashing via the serial interface without any use of external programing devices.
   - The **Scope** button starts an oscilloscope like environment for plotting signals and variables of the running target. For more information see section 11.
3. Tabs

- In the **Status** tab there are two main areas as seen in figure 8. In *DSP state* the current status of the connected target is shown. The following states are possible:
  - The **Monitor** state is only active before code flashing (full version of *X2C*). In the free version this state is only active when the target reboots after an application error.
  - The **Program loaded** is active after code flashing.
  - In the **Idle** state only the communication between target and computer is active. All controller functions are inactive furthermore the *Outports* values are static.
  - The **Init** state calls the initialization functions of all *X2C* blocks. In this state all signals an variables are reset to their initial values.
  - **Run - Power off** means the application is running normal but the power supply of the power electronics (e.g. frequency converter) is off.
  - In **Run - Power on** the system is fully active.

  Info: In the *Blinky* demo application the last four states cannot be changed because they are only useful for engine control applications.
  The *Log* area shows status updates and error messages and can be cleared with the *Clear* button.

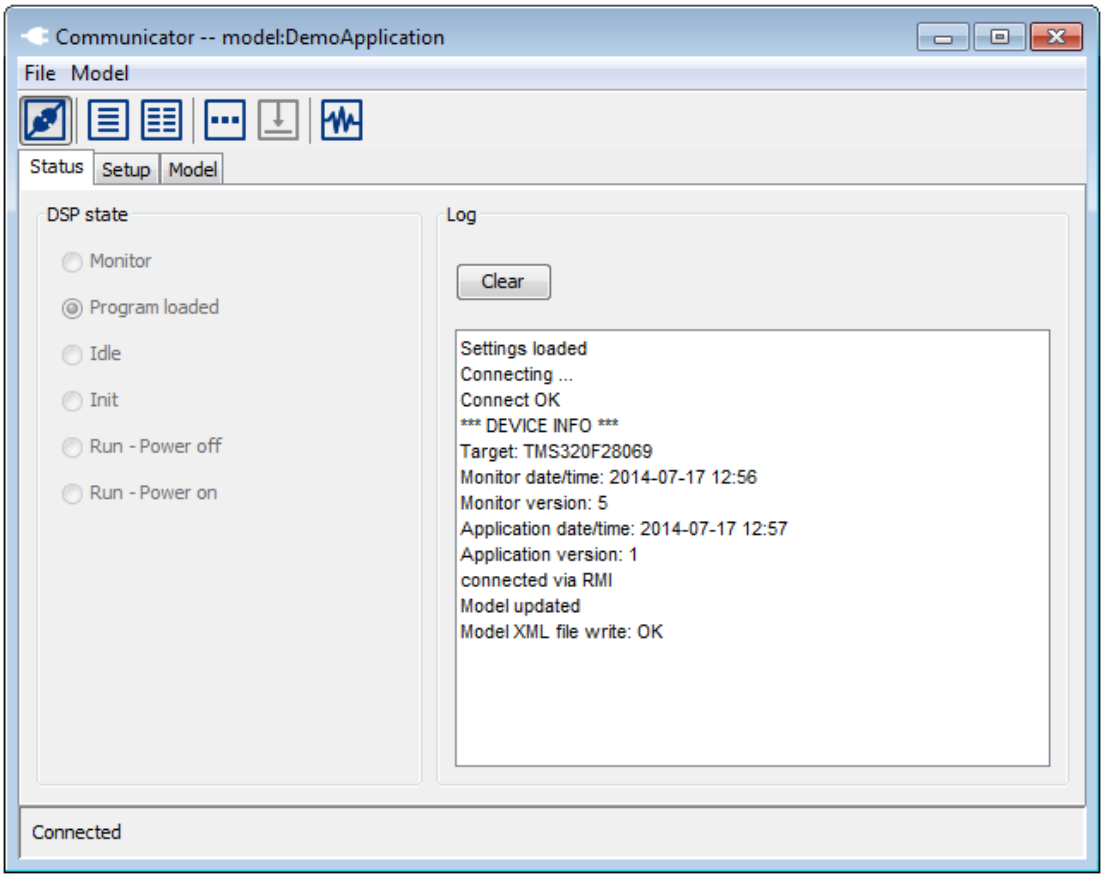


Figure 8: *Communicator* status

- The **Setup** tab is for the interface configuration. To change the settings the *Communicator* needs to be disconnected from the target.
  There are a few ways to connect with the target. One can choose between *Serial*, *USB* and *PCAN* interface. The setting made here need to be compatible with the

target configurations.
In the *Protocol* area the *LNet Node ID* can be set. By default this value is set to 1. Since the *Communicator* can be used with more than one target each target is defined with an unique *LNet Node ID*.

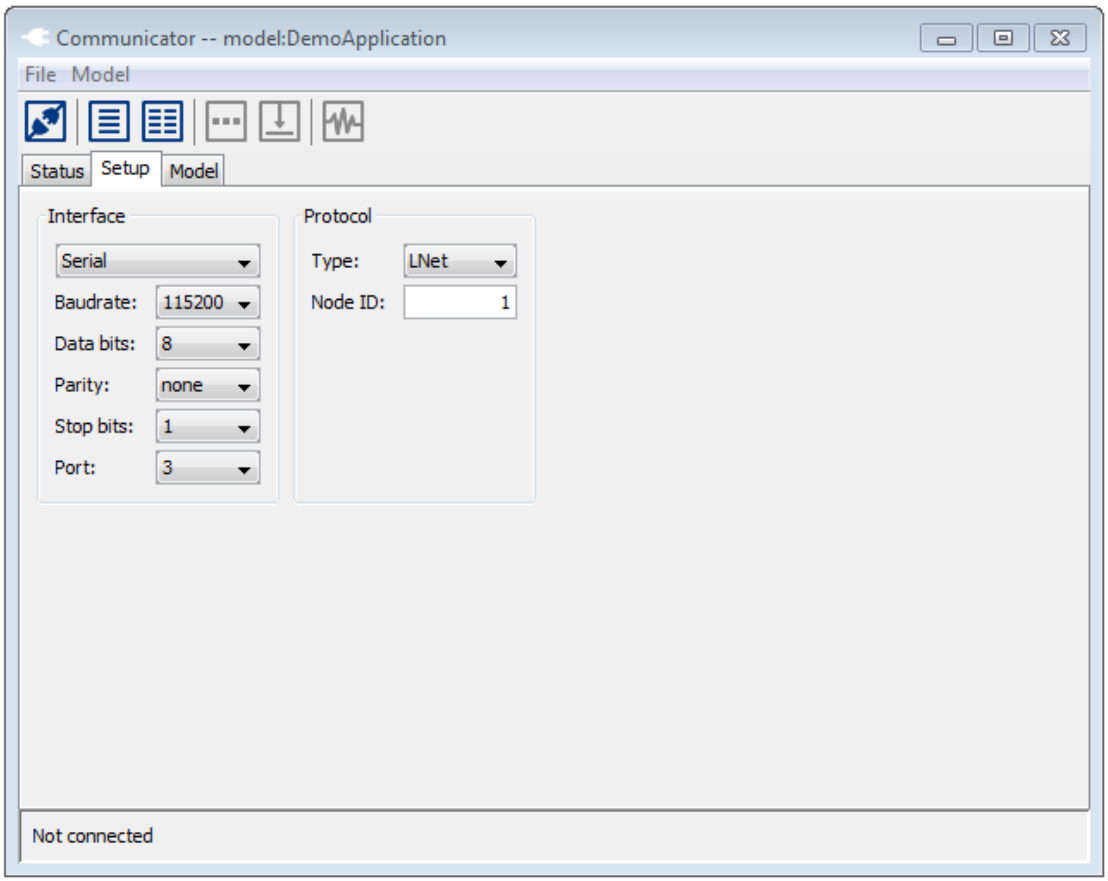


Figure 9: *Communicator* setup

- In the **Model** tab the *Model structure* area provides a list of the used blocks in the *Scilab/Xcos* model. Jump to section 10.5 to see how variables can be changed through the *Model structure* settings.
  The *Model properties* settings need to be made before code generation.
  - The *Sample time* can be changed in the *Scilab/Xcos* model by changing the values in the *CLOCK* block. After double clicking on *transform model and push to Communicator* the sample time in the *Communicator* is updated. After clicking on *Analyze* the sample in the *Communicator* is updated.
    Note: Changes of the sample time made in the model need to be compatible with the defined sample time on the target.
  - *Use Scope* was moved to *Settings*. See 10.4 for details.
  - *Use Parameter ID for block data transfer* was moved to *Settings*. See 10.4 for details.

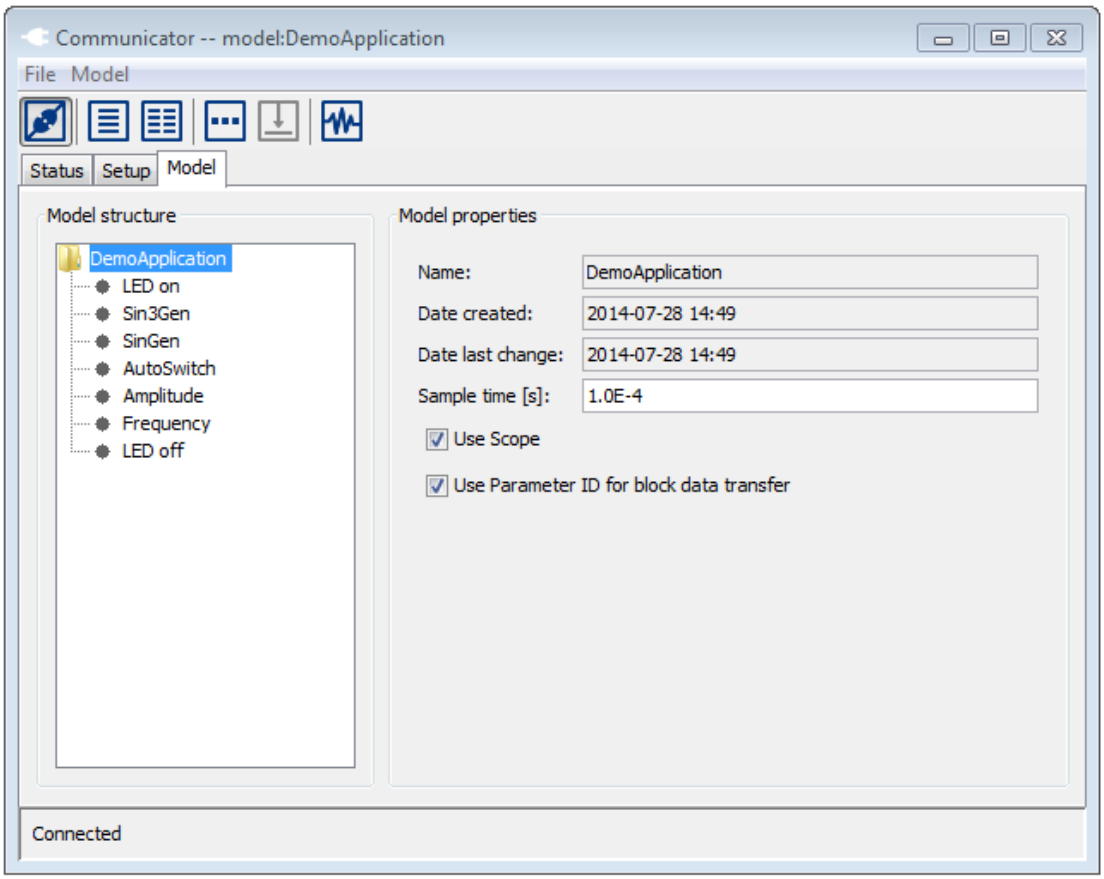


Figure 10: *Communicator Model*

10.4 Settings

Settings can be found in the 'File' menu. These are split into **Common** and **Advanced** options. Following options are available for configuration: (angular brackets = default state)

- **Use Scope** [enabled]

  Use Scope defines if the scope application (see section 11) can be used when connected with the target. When disabled the target processor is relieved due to less communication effort.

- **Create RTOS code** [disabled]

  Generates code which is optimized for real time operations.

- **Connect to target on startup** [enabled]

  Tries to connect to the target when the Communicator starts. If this option is enabled, the previously used communication setup is being used.

- **Use Parameter-ID for block data transfer** [enabled]

  When *Use Parameter ID for block data transfer* is enabled the *Communicator* generates an identification number for each block in the *Scilab/Xcos* model during code generation. As an result only signals at *X2C* blocks can be observed with the *Scope*. When disabled the *Communicator* uses the generated *.map* file out of C code building for block data transfer. In this file the register addresses of the *X2C* block signals and furthermore the addresses of global variables used on the target processor are stored. The *.map* file can be loaded with the button *Download settings*. With this setting it is possible to observe *X2C* block signals as well as global variables with the scope.

- **Create Signals code** [disabled]

  Creates a file containing lists with internal *X2C* signals. These signals are Inports, Outports and Block Outports.

- **Create & compile HotInt code** [disabled]

  Generates HotInt specific files. After successful generation, the HotInt project (Microsoft®Visual Studio) is being compiled. The latest version being found is used for compilation. Supported Visual Studio versions:

  – Visual Studio 2013
  – Visual Studio 2012

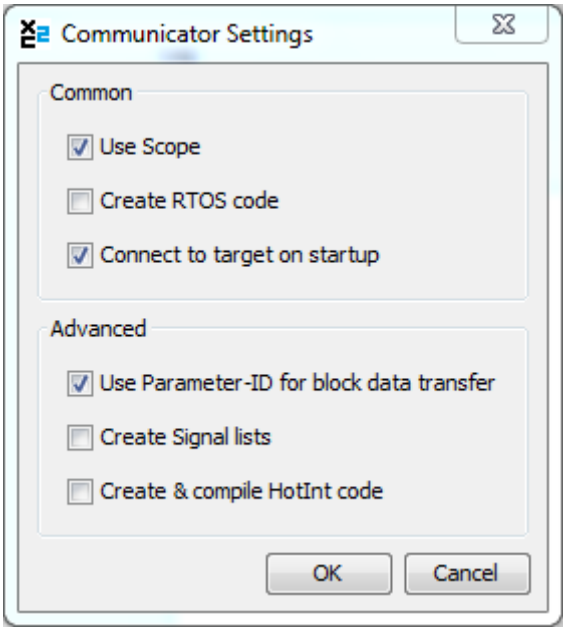


Figure 11: *Communicator Settings*

10.5 Change parameters on the target with the *Communicator*

If all connection are set up properly there are two ways of changing the parameter values on the running target.

1. In the *Communicator* click on **Model**. In the Model structure area all the Block properties are listed and can be changed by double clicking on them.

2. In the *Scilab/Xcos* model double click on the blocks and change the values.

11 Scope

The *Scope* application is a very useful device for monitoring signals and variables on the running target. It allows an easy observation in an oscilloscope like environment. The *Scope* is structured into four main areas as seen in figure 12.

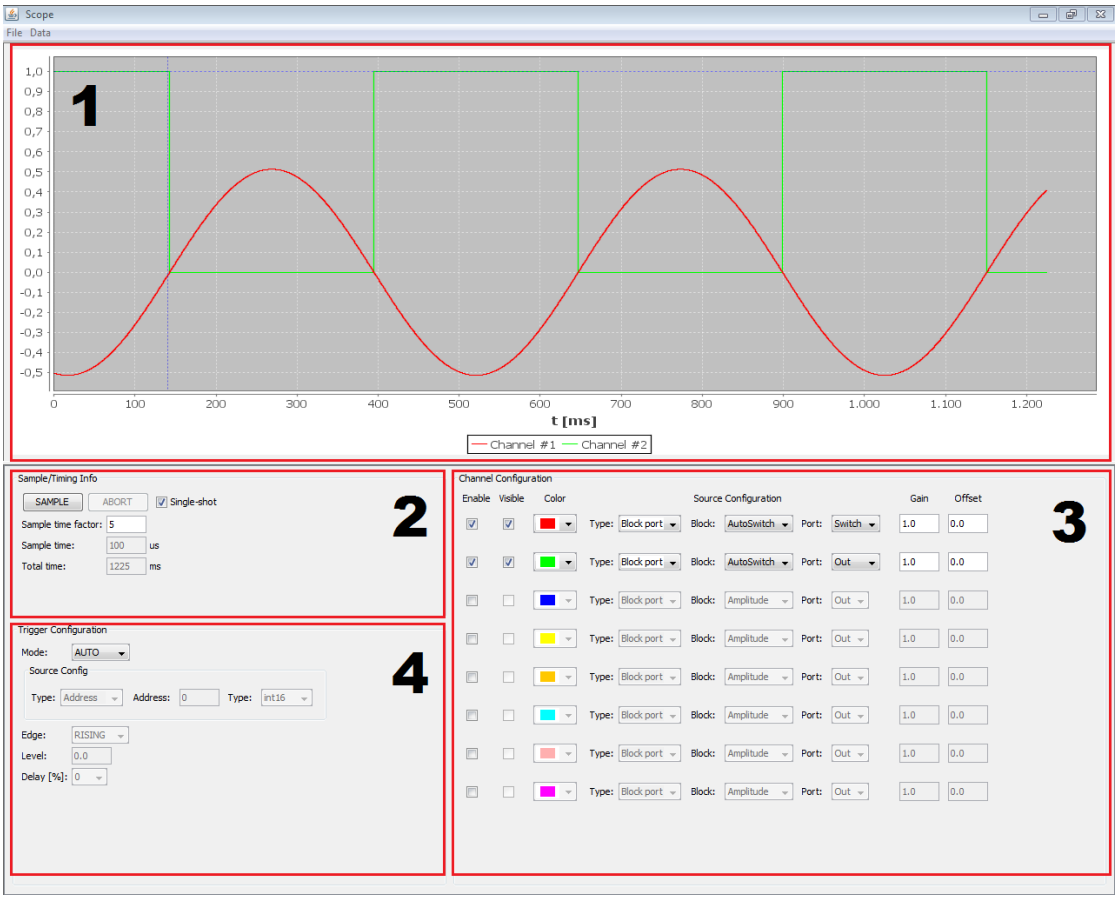


Figure 12: Communicator Scope

1. The **Plot area** shows the selected values at a time based abscissa (x-coordinate) in milliseconds and scaled from -1.0 to 1.0 at the ordinate (y-coordinate). Furthermore there is a legend a the bottom of the plot area showing the color of each channel.

2. In the **Sample/Timing Info** section options of the time axis can be made. The oscilloscope is started with the button *SAMPLE*. When the option *Single-shot* is marked only one time period (*see Total time*) is shown in the plot area. When unmarked the plot area continuously plots the received values from the target. Due to time delay in data transfer it is possible that there a missing values between two plot cycles.
   The plot process can be stopped with the button *ABORT*.
   With the option *Sample time factor* the time axis can be scaled. Factor 1 means every value (Time between two values is *Sample Time*) is plotted in the plot area. As example factor 5 means every fifth value is used, therefore a longer time span can be plotted at the time axis.

3. The **Channel Configuration** configures which signal is shown at the plot area. There are eight channels that can be plotted simultaneously. Mark *Enable* to configure one

channel. In the *Type* menu *Address, Block Port* and *I/O port* can be chosen. *I/O ports* are the links between the target peripheries and the *X2C* model. The *Block Port* are signals used in the *X2C* model.
When fixed point data representation is selected all signal are scaled to values between -1.0 and 1.0, therefore one might use the option *Gain* or *Offset* to plot the signal in real scale.

4. The **Trigger Configuration** is divided in the options *NORMAL* and *AUTO*. When option *AUTO* is chosen no specific trigger is set. In this configuration signal values are continuously transfer and plotted. This can lead to moving graphs especially when periodic signals are observed.

   Choose *NORMAL* to set up a trigger.

   (a) In *Source Config* choose a signal which should work as trigger source.

   (b) With *Edge* the trigger only checks rising respectively falling edges of the source signal.

   (c) The *Level* and *Delay* options move the trigger point in vertical and time direction.

   Example: Trigger the harmonic sine wave *u* from a *SinGen* block.
   As trigger source the signal itself is used. When the trigger is delayed in time a vertical marker indicates the position. The effect of the settings *Level* with a value of 0.2 and *Edge* for *FALLING* can be seen in figure 13.

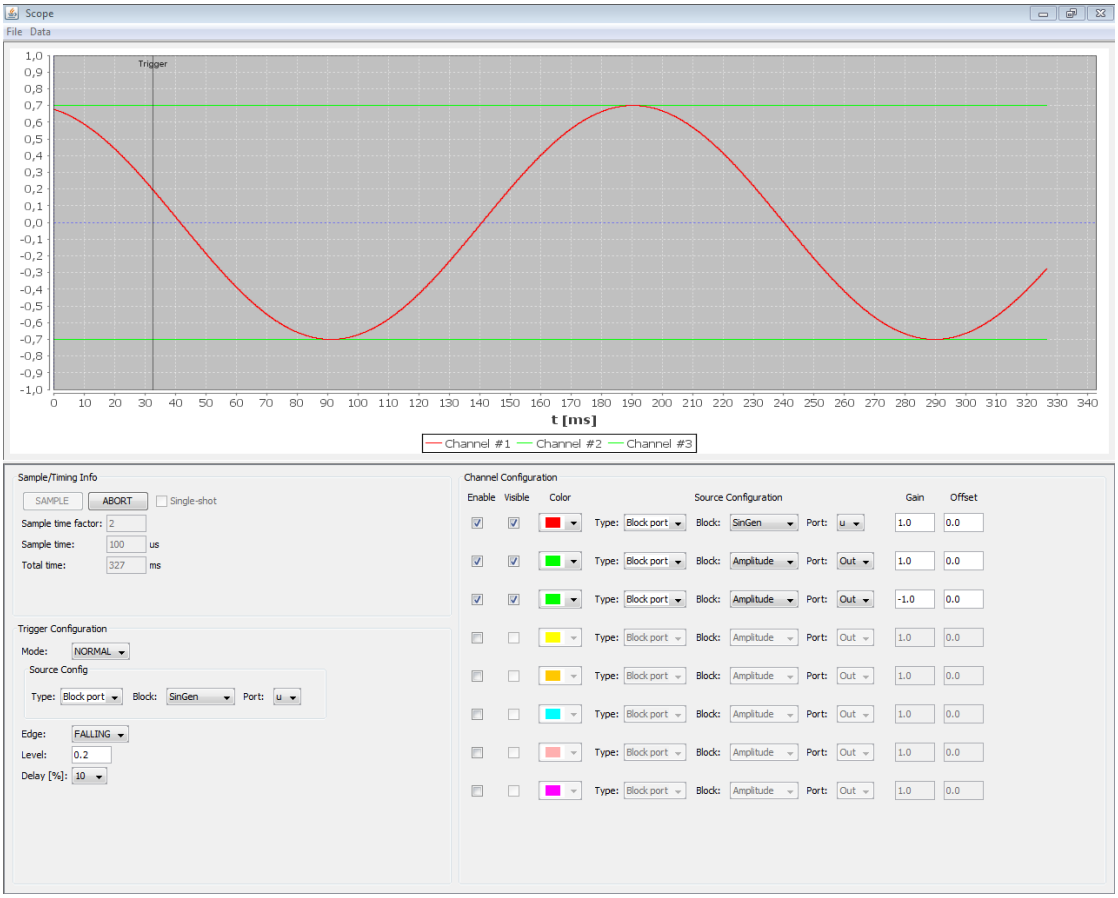


Figure 13: Scope Trigger using Example

12 Block Generator

The Block Generator is used to create new blocks, load and/or edit previously saved blocks. The following, essential parameters define the block function:

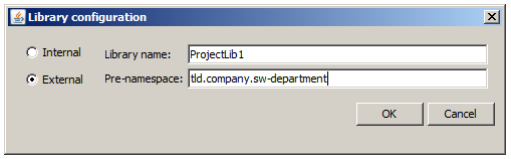
12.1 Block properties

Name

Each block within a library must have a unique name.

Library type

The library type selection is done via the 'Change configuration' button.
An internal library block will be stored within the *X2C* structure where only the library name is required. The internal library name can be selected via a dropdown menu.
When the block is saved, the files will be automatically saved into the correct directories.
An external (or project specific) block is stored within its project structure and requires the user to enter a library name and pre-namespace identifier. Both, the library name & pre-namespace, is enterd via text fields.
When the block is saved, a window will appear, which allows to select the project directory for this project specific block (only directory selection is possible) The Block Generator tries to save the files in the following structure (directories will be created automatically if they don't exist):
<selected directory>\Library\<library name>\



Identifier

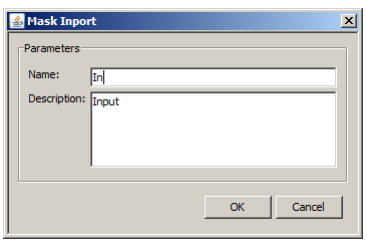
Every block needs an unique identifier (ID) across all libraries to ensure proper functionality if a project uses blocks from different libraries. *ID* should be a value < 4000 for internal blocks and a value $\geq$ 4000 for external blocks.

Additional LaTeX information file

In case of having a LaTeX file with additional block information the name of the file can be set.

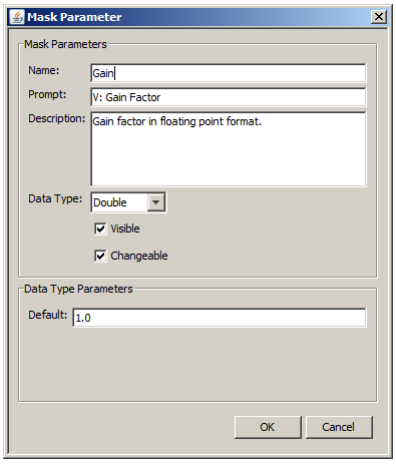
Mask in- & outports

Every block can have several inports & outports. Each in-/outport must have an unique name.



Mask parameters

Every block can have several mask parameters. Each mask parameter must have an unique name. *Prompt* will be displayed next to the value input of this parameter. *Data type* decides between an input field for type *Double* or a dropdown menu for type *ComboBox*.
In case of type *Double* you can select the *Default value* for this parameter.
Type *ComboBox* lets you add/remove items which can be selected by the user if this parameter value should be changed. The default value is defined by selecting one out of the entries.
Visible makes this parameter visible, *Changeable* en- or disables this parameter.



Visualizations

Visualizations are used to represent the block within a model. *Command* contains the language specific commands to represent the block.

12.2 Implementation properties

Every block must have at least 1 Implementation. Each Implementation has its Init-, Update-, Save- and Load functions (C) and Conversion functions (Java/Python/JavaScript).

Name

Each implementation must have a unique name within a block. The Implementation name is used for C- & Java code file name.

Identifier

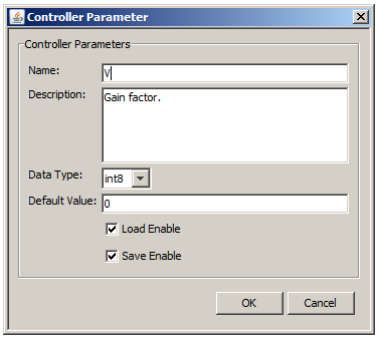
Every Implementation needs an unique idendifier (ID) within its block. This ID can have values in the range from 0 to 15.

Controller In- & Outports

The Controller In- & Outport names can be selected but not edited. The names are defined by the block's Mask In- & Outports. Only the data type must be selected for each In-/Outport.

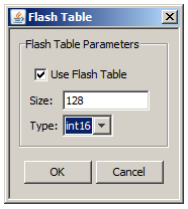
Controller Parameter

Each Controller Parameter must have an unique name within its Implementation. The data type and default value can be selected. Also the ability to download/upload the parameter can be defined by using *Load Enable* and *Save Enable* checkboxes.



Flash table

A flash table can be enabled for this block by using the *Change* button. If *Use Flash Table* is selected, the size and data type of the flash table can be selected.



Conversion function type

The Conversion function type can be selected by using the dropdown menu. If *Java*, *Python* or *JavaScript* is selected, the Block Generator will generate a conversion function file for this block, otherwise no conversion function file will be created.

Update enable

If checked, an Update function is generated when saving the block.

12.3 Save or load a block

Saving & loading is done via the *File* menu in the menu bar.

Saving

If the selected block is member of an internal library, the Block Generator automatically uses the correct library root directory.
In case of an external library block type, the user is prompted a directory selection window, in which the project directory can be selected. The library root directory is now located in: <user directory selection>\<Library>\<library name>.
Each library is organized in this structure:

- Controller: Directory with the C-code source files (*.c, *.h).
- Conversion: Directory with the Java, Python or JavaScript conversion files.
- Doc: Directory with files needed for the (auto-generated) documentation.
- Scilab: This directory contains the *Scilab/Xcos* library files as well as the interfaces functions and the files need for simulation in *Scilab/Xcos* .
- XML: Configuration files (*.xml) contain all block parameters and are located in this directory.

Part IV
How-To

13 *X2C* code generation with *Scilab/Xcos*

The following section describes *X2C* code generation of a *Scilab/Xcos* model based on the *Blinky* demo application.

1. Open *Scilab/Xcos* and in the file browser navigate to your project directory (e.g. <X2C_ROOT>\DemoApplication\Blinky_TI_TMS320F28069_controlSTICK\X2CCode).

2. Double click on **DemoApplication.zcos**. The example project contains a few blocks used to demonstrate the basic function of *X2C* (see figure 14). The *Inport* and *Outport* blocks define the interface between the generated *X2C* code and the peripheral functions (e.g. ADC or GPIO Pins) on the target. For details about each block function read *X2Copen.Doc.pdf* in the documentation folder of the *X2C* directory.

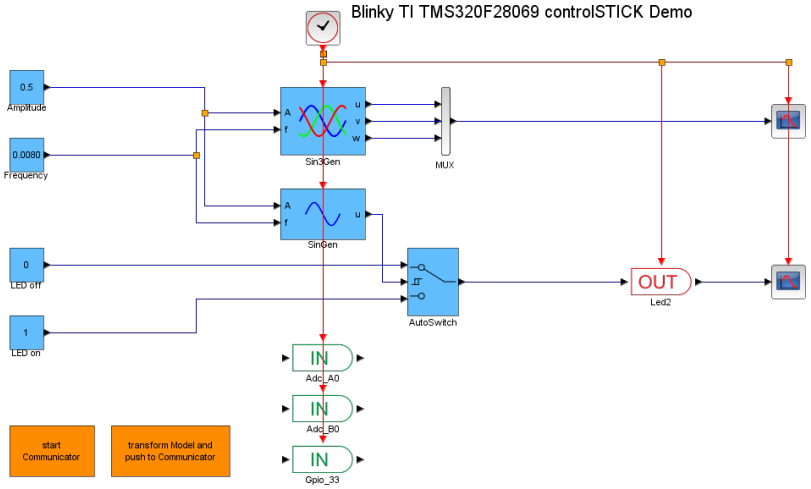


Figure 14: *Blinky* demo application in *Scilab/Xcos*

3. Double click on **start Communicator** (for more information about the *Communicator* see section 10). Some details of the current actions of the *Communicator* are shown in the *Log* area of the *Communicator* window and the *Scilab/Xcos* command line:

```
1        Starting Communicator
2        done
3        Successfully connected to Communicator
```

4. Double click on **Transform model and push to Communicator** and check the pop-up window for the end of the transformation process.

5. Click **Create Code** in the *Communicator*. Now the files *X2C.h* and *X2C.c* are generated in the <PROJECT_ROOT>\X2CCode directory and the Log screen should contain the lines:

```
1        [...]
2        Model updated
3        Model XML file write: OK
4        Create code successful.
```

6. The *C* code for the *X2C* application has been created. Depending on the used target start the programming tool (e.g. *Code Composer Studio* or *MPLAB X*) and import the *Blinky* demo application project as described in section 14, or 15 respectively. Follow the instructions on how to configure and flash the project on the target.

14 Loading and building the demo application Blinky in *Code Composer Studio*

The demo application *Blinky* is intended to be used with *TI F28069 Piccolo controlSTICK*.

1. Connect the *TI F28069 Piccolo controlSTICK* with the computer.
2. Open *Code Composer Studio* (choose workspace directory as you like). Now click **Project** → **Import Existing CCS Eclipse Project**. Browse to the location of the *Blinky* project (<X2C_ROOT>\DemoApplication\Blinky_TI_TMS320F28069_controlSTICK). Click **Finish** to import the project.
3. In the *Code Composer Studio* file structure of the *Blinky* demo project there are two virtual folders *Blocks* and *Core*, which should be linked directly to the *X2C* directory. To ensure this go to **Project** → **Properties** drop down **Resource** and click **Linked Resources**. Double click on folder **X2C_ROOT** and set the correct link to your X2C installation directory (<X2C_ROOT>). After hitting **OK** two times there should not be any warning signs (like shown in figure 15) at the icons for the linked files in the Blocks and Core folders.

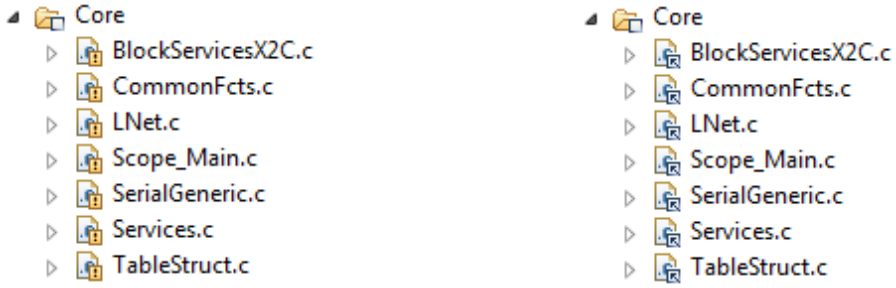


Figure 15: *Code Composer Studio* invalid (left) and valid (right) *X2C* root directory

4. The generated code from *X2C* is located in the folder <X2C_ROOT>\DemoApplication\Blinky_TI_TMS320F28069_controlSTICK\X2CCode. To check if code generation went fine go to the X2CCode folder and open *X2C.c*. Make sure time and date of code generation is plausible.
5. Build the project in *Code Composer Studio* by clicking **Project** → **Build all** or by clicking on the **Hammer** symbol as seen in figure 16 at the top of the screen. Check for errors while building in the console at the bottom of the screen.

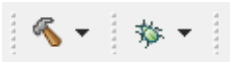


Figure 16: *Code Composer Studio* build and debug buttons

6. If your target is connected to the computer click **Run** → **Debug** or click on the *Bug* symbol as seen in figure 16 at the top. The program is now transferred to the target and can be started with the **green arrow** button at the top.
7. After starting the program the on-board LED of the *TI F28069 Piccolo controlSTICK* should be blinking!

15 Loading and building the demo application Blinky in *MPLAB X*

The demo application *Blinky* is build for the combination of the *Microstick II* with the *dsPIC33FJ128MC802* processor and the *MicrostickPlus* developer board (for details see `www.microstick.com`).
Info: While flashing new code only the *Microstick II* needs to be connected with the computer.

1. Connect the *Microstick II* with the computer.

2. Open *MPLAB X* and click **File** → **Open Project**. Browse to the location of the *Blinky* demo application in the *X2C* directory <X2C_ROOT>\DemoApplication\. . . \Blinky_Microchip_dsPIC33Fxxxx_MicrostickPlus. Click **Open Project**.

3. In the case the demo application is copied/moved to a different location, the include paths have to be adapted. To ensure the compiler uses the correct path variables right click on the **Projectname** → **Properties** → **XC16 Global Options** → **xc16-gcc**. In the drop down menu **Option categories** choose **Preprocessing and messages**. Click on the dots beside *C include dirs*. There are relative paths to the needed include files listed as seen in figure 17. Correct the links by double clicking on the path variables.
Info: Only the links to the *Library* and *Controller* path need to be updated.

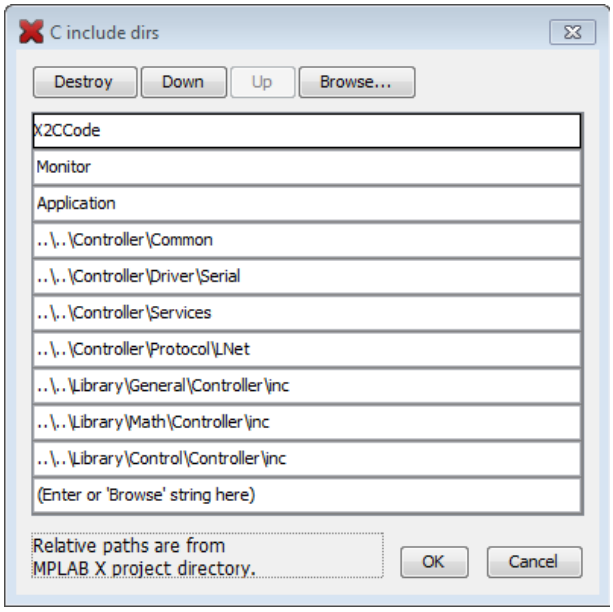


Figure 17: Default path variables for the include files

4. Go to **Run** → **Clean and Build Main Project** or click the *hammer with brush* button as seen in figure 18. After building there should be a message `BUILD SUCCESSFUL` in the message area at the bottom of the screen.

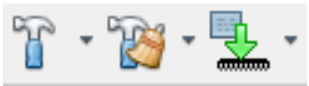


Figure 18: *MPLAB X Clean and Build Main Project* button

5. If the build process was successful go to **Run** → **Run Main Project** or click the *Green Arrow* button as seen in figure 18. If there is a message similar to *MICROSTICK not Found* try to select the *Starter Kits (PKOB)* item which represents your board.

6. After starting the program the LED (RB12) on the *MicrostickPlus Board* should be blinking!

16 *X2C* block generation

This section describes the creation of an external (or project specific) *X2C* block. Basically, block creation is done in three steps as described below.

16.1 Generation of block structure

Generate the general block structure with the BlockGenerator tool described in section 12. The BlockGenerator can be startet by executing the BlockGenerator.jar file located in <X2C_ROOT>\System\Java\.

16.2 Coding

Enter the functionality of the block in its C-code source file. The C-code source file can be found in <ProjectDir>\Library\<LibraryName>\Controller\src\.
If the block uses mask parameters, the conversion function, which converts the mask parameters into controller parameters, has to be coded as well. The conversion function can be found in <ProjectDir>\Library\<LibraryName>\Conversion\<ConversionType>\.

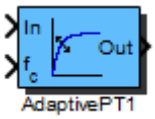
16.3 Finishing of block in Scilab

In Scilab execute the command `createXcosBlock('<LibraryName>', '<BlockName>', '<ProjectDir>')`.

Part V
Libraries

17 Control

Block: AdaptivePT1



Inports	
In	Input In(k)
fc	Cutoff frequency

Outports	
Out	Output Out(k)

Mask Parameters	
V	Gain
fmax	Maximum frequency [Hz] (not used in floating point implementations)
ts_fact	Multiplication factor of base sampling time (in integer format)
method	Discretization method

Description:

First order low pass with adaptive cut off frequency:
G(s) = V/(s/(2*pi*fc) + 1)

Transfer function (zero-order hold discretization method):

$$G(z) = V\frac{1-e^{-2\pi f_c T_s}}{z-e^{-2\pi f_c T_s}}$$

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

Name	FiP8
ID	3408
Revision	0.1
C filename	AdaptivePT1_FiP8.c
H filename	AdaptivePT1_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
w_scale	Calculation base for wc: -2*pi*Ts*fmax
gain	Gain
sfr	Shift factor for gain
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            *fc;
      int8            Out;
      int8            w_scale;
      int8            gain;
      uint8           sfr;
      int8            in_old;
} ADAPTIVEPT1_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	3409
Revision	1
C filename	AdaptivePT1_FiP16.c
H filename	AdaptivePT1_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
w_scale	Calculation base for wc: -2*pi*Ts*fmax
gain	Gain
sfr	Shift factor for gain
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
```

```
    int16          *In;
    int16          *fc;
    int16          Out;
    int16          w_scale;
    int16          gain;
    uint8          sfr;
    int16          in_old;
} ADAPTIVEPT1_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	3410
Revision	0.1
C filename	AdaptivePT1_FiP32.c
H filename	AdaptivePT1_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
w_scale	Calculation base for wc: -2*pi*Ts*fmax
gain	Gain
sfr	Shift factor forgain
in_old	In(k-1)

Data Structure:

```
typedef struct {
    uint16         ID;
    int32          *In;
    int32          *fc;
    int32          Out;
    int32          w_scale;
    int32          gain;
    uint8          sfr;
    int32          in_old;
} ADAPTIVEPT1_FIP32;
```

Implementation: Float32

Name	Float32
ID	3411
Revision	0.1
C filename	AdaptivePT1_Float32.c
H filename	AdaptivePT1_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
w_scale	Calculation base for wc: -2*pi*Ts*fmax
gain	Gain
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
      float32         *fc;
      float32         Out;
      float32         w_scale;
      float32         gain;
      float32         in_old;
} ADAPTIVEPT1_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	3412
Revision	0.1
C filename	AdaptivePT1_Float64.c
H filename	AdaptivePT1_Float64.h

64 Bit Floating Point Implementation

Controller Parameters	
w_scale	Calculation base for wc: -2*pi*Ts*fmax
gain	Gain
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In;
      float64         *fc;
      float64         Out;
      float64         w_scale;
      float64         gain;
      float64         in_old;
} ADAPTIVEPT1_FLOAT64;
```

Block: Delay

Inports	
In	Input In(k)

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

Mask Parameters	
ts_fact	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:

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

Name	FiP16
ID	3425
Revision	0.1
C filename	Delay_FiP16.c
H filename	Delay_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
In_old	Input value from previous cycle

Data Structure:

```
typedef struct {
    uint16        ID;
```

```
    int16          *In;
    int16          Out;
    int16          In_old;
} DELAY_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	3426
Revision	0.1
C filename	Delay_FiP32.c
H filename	Delay_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
In_old	Input value from previous cycle

Data Structure:

```
typedef struct {
    uint16         ID;
    int32          *In;
    int32          Out;
    int32          In_old;
} DELAY_FIP32;
```

Implementation: Float32

Name	Float32
ID	3427
Revision	0.1
C filename	Delay_Float32.c
H filename	Delay_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
In_old	Input value from previous cycle

Data Structure:

```
typedef struct {
    uint16         ID;
    float32        *In;
    float32        Out;
    float32        In_old;
} DELAY_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	3428
Revision	0.1
C filename	Delay_Float64.c
H filename	Delay_Float64.h

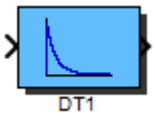
64 Bit Floating Point Implementation

Controller Parameters	
In_old	Input value from previous cycle

Data Structure:

```
typedef struct {
     uint16         ID;
     float64        *In;
     float64        Out;
     float64        In_old;
} DELAY_FLOAT64;
```

Block: DT1



Inports	
In	Input In(k)

Outports	
Out	Output Out(k)

Mask Parameters	
V	Gain
fc	Cut off frequency of low pass filter
ts_fact	Multiplication factor of base sampling time (in integer format)
method	Discretization method

Description:

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

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

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

Name	FiP8
ID	3328
Revision	0.1
C filename	DT1_FiP8.c
H filename	DT1_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
b0	
b1	
a0	
sfrb	
sfra	
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            Out;
      int8            b0;
      int8            b1;
      int8            a0;
      int8            sfrb;
      int8            sfra;
      int8            in_old;
} DT1_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	3329
Revision	0.1
C filename	DT1_FiP16.c
H filename	DT1_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
b0	
b1	
a0	
sfrb	
sfra	
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In;
      int16           Out;
```

```
    int16           b0;
    int16           b1;
    int16           a0;
    int8            sfrb;
    int8            sfra;
    int16           in_old;
} DT1_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	3330
Revision	0.1
C filename	DT1_FiP32.c
H filename	DT1_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
b0	
b1	
a0	
sfrb	
sfra	
in_old	In(k-1)

Data Structure:

```
typedef struct {
    uint16          ID;
    int32           *In;
    int32           Out;
    int32           b0;
    int32           b1;
    int32           a0;
    int8            sfrb;
    int8            sfra;
    int32           in_old;
} DT1_FIP32;
```

Implementation: Float32

Name	Float32
ID	3331
Revision	0.1
C filename	DT1_Float32.c
H filename	DT1_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
b0	Coefficient b0
b1	Coefficient b1
a0	Coefficient a0
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
      float32         Out;
      float32         b0;
      float32         b1;
      float32         a0;
      float32         in_old;
} DT1_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	3332
Revision	0.1
C filename	DT1_Float64.c
H filename	DT1_Float64.h

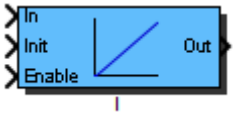
64 Bit Floating Point Implementation

Controller Parameters	
b0	Coefficient b0
b1	Coefficient b1
a0	Coefficient a0
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In;
      float64         Out;
      float64         b0;
      float64         b1;
      float64         a0;
      float64         in_old;
} DT1_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	
Ki	Integral Factor
ts_fact	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):

$$G(z) = K_I T_s \frac{1}{z-1}$$

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

Name FiP8
ID 3200
Revision 1.0
C filename I_FiP8.c
H filename I_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
sfr	Shift factor for I coefficient b0
i_old	Integrator value from previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            *Init;
      int8            *Enable;
      int8            Out;
      int8            b0;
      int8            sfr;
      int16           i_old;
      int8            enable_old;
} I_FIP8;
```

Implementation: FiP16

Name FiP16
ID 3201
Revision 1.0
C filename I_FiP16.c
H filename I_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
sfr	Shift factor for I coefficient b0
i_old	Integrator value from previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
```

```
    uint16          ID;
    int16           *In;
    int16           *Init;
    int8            *Enable;
    int16           Out;
    int16           b0;
    int8            sfr;
    int32           i_old;
    int8            enable_old;
} I_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	3202
Revision	1.0
C filename	I_FiP32.c
H filename	I_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
sfr	Shift factor for I coefficient b0
i_old	Integrator value from previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
    uint16          ID;
    int32           *In;
    int32           *Init;
    int8            *Enable;
    int32           Out;
    int32           b0;
    int8            sfr;
    int64           i_old;
    int8            enable_old;
} I_FIP32;
```

Implementation: Float32

Name	Float32
ID	3203
Revision	0.1
C filename	I_Float32.c
H filename	I_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
b0	Integral coefficient
i_old	Integrator value from previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
     uint16          ID;
     float32         *In;
     float32         *Init;
     int8            *Enable;
     float32         Out;
     float32         b0;
     float32         i_old;
     int8            enable_old;
} I_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	3204
Revision	0.1
C filename	I_Float64.c
H filename	I_Float64.h

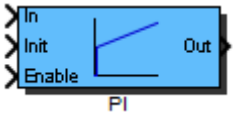
64 Bit Floating Point Implementation

Controller Parameters	
b0	Integral coefficient
i_old	Integrator value from previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
     uint16          ID;
     float64         *In;
     float64         *Init;
     int8            *Enable;
     float64         Out;
     float64         b0;
     float64         i_old;
     int8            enable_old;
} I_FLOAT64;
```

Block: PI



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	

Mask Parameters	
Kp	Proportional Factor
Ki	Integral Factor
ts_fact	Multiplication factor of base sampling time (in integer format)

Description:

PI controller:
G(s) = Kp + Ki/s

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

Transfer function (zero-order hold discretization method):

$$G(z) \quad = \quad K_P + K_I T_s \frac{1}{z-1}$$

Developer note: For the fixed point implementations the source code of block Block: PILimit 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

Name	FiP8
ID	3216
Revision	2.0
C filename	PI_FiP8.c
H filename	PI_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
sfrb0	Shift factor for PI coefficient b0
sfrb1	Shift factor for PI coefficient b1
i_old	Integrator value from previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
     uint16          ID;
     int8            *In;
     int8            *Init;
     int8            *Enable;
     int8            Out;
     int8            b0;
     int8            b1;
     int8            sfrb0;
     int8            sfrb1;
     int16           i_old;
     int8            enable_old;
} PI_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	3217
Revision	2.0
C filename	PI_FiP16.c
H filename	PI_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
sfrb0	Shift factor for PI coefficient b0
sfrb1	Shift factor for PI coefficient b1
i_old	Integrator value from previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
     uint16          ID;
     int16           *In;
     int16           *Init;
     int8            *Enable;
     int16           Out;
     int16           b0;
     int16           b1;
     int8            sfrb0;
     int8            sfrb1;
     int32           i_old;
     int8            enable_old;
} PI_FIP16;
```

Implementation: FiP32

Name FiP32
ID 3218
Revision 2.0
C filename PI_FiP32.c
H filename PI_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
sfrb0	Shift factor for PI coefficient b0
sfrb1	Shift factor for PI coefficient b1
i_old	Integrator value from previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
     uint16          ID;
     int32           *In;
     int32           *Init;
```

```
    int8            *Enable;
    int32           Out;
    int32           b0;
    int32           b1;
    int8            sfrb0;
    int8            sfrb1;
    int64           i_old;
    int8            enable_old;
} PI_FIP32;
```

Implementation: Float32

Name	Float32
ID	3219
Revision	2.0
C filename	PI_Float32.c
H filename	PI_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
i_old	Integrator value of previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
    uint16          ID;
    float32         *In;
    float32         *Init;
    int8            *Enable;
    float32         Out;
    float32         b0;
    float32         b1;
    float32         i_old;
    int8            enable_old;
} PI_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	3220
Revision	2.0
C filename	PI_Float64.c
H filename	PI_Float64.h

64 Bit Floating Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
i_old	Integrator value of previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
     uint16          ID;
     float64         *In;
     float64         *Init;
     int8            *Enable;
     float64         Out;
     float64         b0;
     float64         b1;
     float64         i_old;
     int8            enable_old;
} PI_FLOAT64;
```

Block: PID

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	

Mask Parameters	
Kp	Proportional Factor
Ki	Integral Factor
Kd	Derivative Factor
fc	Cutoff frequency of realization low pass
ts_fact	Multiplication factor of base sampling time (in integer format)

Description:

PID controller:
G(s) = Kp + Ki/s + Kd*s/(s/(2*pi*fc) + 1)

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):

$$G(z) = K_P + K_I T_s \frac{1}{z-1} + K_D \omega_c \frac{z-1}{z - e^{-\omega_c T_s}}$$

FiP8 bug: When using the TI compiler the step response of the derivative part doesn't return to zero, but generates an overflow at zero crossing if the derivative parameter value is too high.

Developer note: For the fixed point implementations the source code of block *PIDLimit* 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

Name	FiP8
ID	3248
Revision	1.0
C filename	PID_FiP8.c
H filename	PID_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
b0d	Derivational coefficient b0
b1d	Derivational coefficient b1
a0d	Derivational coefficient a0
sfrb0	Shift factor for PI coefficient b0
sfrb1	Shift factor for PI coefficient b1
sfrd	Shift factor for D coefficients b0d and b1d
in_old	Input value of previous cycle
i_old	Integrator value of previous cycle
d_old	Derivative value of previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
    uint16          ID;
    int8            *In;
    int8            *Init;
    int8            *Enable;
    int8            Out;
    int8            b0;
    int8            b1;
    int8            b0d;
    int8            b1d;
    int8            a0d;
    int8            sfrb0;
    int8            sfrb1;
    int8            sfrd;
    int8            in_old;
```

```
    int16           i_old;
    int8            d_old;
    int8            enable_old;
} PID_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	3249
Revision	1.0
C filename	PID_FiP16.c
H filename	PID_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
b0d	Derivational coefficient b0
b1d	Derivational coefficient b1
a0d	Derivational coefficient a0
sfrb0	Shift factor for PI coefficient b0
sfrb1	Shift factor for PI coefficient b1
sfrd	Shift factor for D coefficients b0d and b1d
in_old	Input value of previous cycle
i_old	Integrator value of previous cycle
d_old	Derivative value of previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
    uint16          ID;
    int16           *In;
    int16           *Init;
    int8            *Enable;
    int16           Out;
    int16           b0;
    int16           b1;
    int16           b0d;
    int16           b1d;
    int16           a0d;
    int8            sfrb0;
    int8            sfrb1;
    int8            sfrd;
    int16           in_old;
    int32           i_old;
    int16           d_old;
    int8            enable_old;
```

```
} PID_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	3250
Revision	1.0
C filename	PID_FiP32.c
H filename	PID_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
b0d	Derivational coefficient b0
b1d	Derivational coefficient b1
a0d	Derivational coefficient a0
sfrb0	Shift factor for PI coefficient b0
sfrb1	Shift factor for PI coefficient b1
sfrd	Shift factor for D coefficients b0d and b1d
in_old	Input value of previous cycle
i_old	Integrator value of previous cycle
d_old	Derivative value of previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
     uint16          ID;
     int32           *In;
     int32           *Init;
     int8            *Enable;
     int32           Out;
     int32           b0;
     int32           b1;
     int32           b0d;
     int32           b1d;
     int32           a0d;
     int8            sfrb0;
     int8            sfrb1;
     int8            sfrd;
     int32           in_old;
     int64           i_old;
     int32           d_old;
     int8            enable_old;
} PID_FIP32;
```

Implementation: Float32

Name	Float32
ID	3251
Revision	0.1
C filename	PID_Float32.c
H filename	PID_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
b0d	Derivational coefficient b0
b1d	Derivational coefficient b1
a0d	Derivational coefficient a0
in_old	Input value of previous cycle
i_old	Integrator value of previous cycle
d_old	Derivative value of previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
     uint16          ID;
     float32         *In;
     float32         *Init;
     int8            *Enable;
     float32         Out;
     float32         b0;
     float32         b1;
     float32         b0d;
     float32         b1d;
     float32         a0d;
     float32         in_old;
     float32         i_old;
     float32         d_old;
     int8            enable_old;
} PID_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	3252
Revision	0.1
C filename	PID_Float64.c
H filename	PID_Float64.h

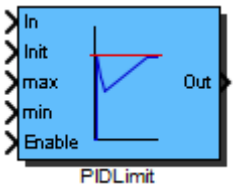
64 Bit Floating Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
b0d	Derivational coefficient b0
b1d	Derivational coefficient b1
a0d	Derivational coefficient a0
in_old	Input value of previous cycle
i_old	Integrator value of previous cycle
d_old	Derivative value of previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
     uint16          ID;
     float64         *In;
     float64         *Init;
     int8            *Enable;
     float64         Out;
     float64         b0;
     float64         b1;
     float64         b0d;
     float64         b1d;
     float64         a0d;
     float64         in_old;
     float64         i_old;
     float64         d_old;
     int8            enable_old;
} PID_FLOAT64;
```

Block: PIDLimit



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

Outports	
Out	

Mask Parameters	
Kp	Proportional Factor
Ki	Integral Factor
Kd	Derivative Factor
fc	Cutoff frequency of realization low pass
ts_fact	Multiplication factor of base sampling time (in integer format)

Description:

PID Controller with Output Limitation:
G(s) = Kp + Ki/s + Kd*s/(s/(2*pi*fc) + 1)

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):

$$G(z) = K_P + K_I T_s \frac{1}{z-1} + K_D \omega_c \frac{z-1}{z - e^{-\omega_c T_s}}$$

FiP8 bug: When using the TI compiler the step response of the derivative part doesn't return to zero, but generates an overflow at zero crossing if the derivative parameter value is too high.

Developer note: The fixed point implementation source code of this block is used for block *PID*.

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

Name	FiP8
ID	3264
Revision	1.0
C filename	PIDLimit_FiP8.c
H filename	PIDLimit_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
b0d	Derivational coefficient b0
b1d	Derivational coefficient b1
a0d	Derivational coefficient a0
sfrb0	Shift factor for PI coefficient b0
sfrb1	Shift factor for PI coefficient b1
sfrd	Shift factor for D coefficients b0d and b1d
in_old	Input value of previous cycle
i_old	Integrator value of previous cycle
d_old	Derivative value of previous cycle
enable_old	Enable(k-1)

Data Structure:

```
typedef struct {
    uint16          ID;
    int8            *In;
    int8            *Init;
    int8            *max;
    int8            *min;
    int8            *Enable;
    int8            Out;
    int8            b0;
    int8            b1;
    int8            b0d;
```

```
    int8            b1d;
    int8            a0d;
    int8            sfrb0;
    int8            sfrb1;
    int8            sfrd;
    int8            in_old;
    int16           i_old;
    int8            d_old;
    int8            enable_old;
} PIDLIMIT_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	3265
Revision	1.0
C filename	PIDLimit_FiP16.c
H filename	PIDLimit_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
b0d	Derivational coefficient b0
b1d	Derivational coefficient b1
a0d	Derivational coefficient a0
sfrb0	Shift factor for PI coefficient b0
sfrb1	Shift factor for PI coefficient b1
sfrd	Shift factor for D coefficients b0d and b1d
in_old	Input value of previous cycle
i_old	Integrator value of previous cycle
d_old	Derivative value of previous cycle
enable_old	Enable(k-1)

Data Structure:

```
typedef struct {
    uint16          ID;
    int16           *In;
    int16           *Init;
    int16           *max;
    int16           *min;
    int8            *Enable;
    int16           Out;
    int16           b0;
    int16           b1;
    int16           b0d;
    int16           b1d;
```

```
    int16           a0d;
    int8            sfrb0;
    int8            sfrb1;
    int8            sfrd;
    int16           in_old;
    int32           i_old;
    int16           d_old;
    int8            enable_old;
} PIDLIMIT_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	3266
Revision	1.0
C filename	PIDLimit_FiP32.c
H filename	PIDLimit_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
b0d	Derivational coefficient b0
b1d	Derivational coefficient b1
a0d	Derivational coefficient a0
sfrb0	Shift factor for PI coefficient b0
sfrb1	Shift factor for PI coefficient b1
sfrd	Shift factor for D coefficients b0d and b1d
in_old	Input value of previous cycle
i_old	Integrator value of previous cycle
d_old	Derivative value of previous cycle
enable_old	Enable(k-1)

Data Structure:

```
typedef struct {
    uint16          ID;
    int32           *In;
    int32           *Init;
    int32           *max;
    int32           *min;
    int8            *Enable;
    int32           Out;
    int32           b0;
    int32           b1;
    int32           b0d;
    int32           b1d;
    int32           a0d;
```

```
    int8            sfrb0;
    int8            sfrb1;
    int8            sfrd;
    int32           in_old;
    int64           i_old;
    int32           d_old;
    int8            enable_old;
} PIDLIMIT_FIP32;
```

Implementation: Float32

Name	Float32
ID	3267
Revision	0.1
C filename	PIDLimit_Float32.c
H filename	PIDLimit_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
b0d	Derivational coefficient b0
b1d	Derivational coefficient b1
a0d	Derivational coefficient a0
in_old	Input value of previous cycle
i_old	Integrator value of previous cycle
d_old	Derivative value of previous cycle
enable_old	Enable(k-1)

Data Structure:

```
typedef struct {
    uint16          ID;
    float32         *In;
    float32         *Init;
    float32         *max;
    float32         *min;
    int8            *Enable;
    float32         Out;
    float32         b0;
    float32         b1;
    float32         b0d;
    float32         b1d;
    float32         a0d;
    float32         in_old;
    float32         i_old;
    float32         d_old;
    int8            enable_old;
} PIDLIMIT_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	3268
Revision	0.1
C filename	PIDLimit_Float64.c
H filename	PIDLimit_Float64.h

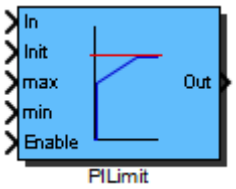
64 Bit Floating Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
b0d	Derivational coefficient b0
b1d	Derivational coefficient b1
a0d	Derivational coefficient a0
in_old	Input value of previous cycle
i_old	Integrator value of previous cycle
d_old	Derivative value of previous cycle
enable_old	Enable(k-1)

Data Structure:

```
typedef struct {
     uint16          ID;
     float64         *In;
     float64         *Init;
     float64         *max;
     float64         *min;
     int8            *Enable;
     float64         Out;
     float64         b0;
     float64         b1;
     float64         b0d;
     float64         b1d;
     float64         a0d;
     float64         in_old;
     float64         i_old;
     float64         d_old;
     int8            enable_old;
} PIDLIMIT_FLOAT64;
```

Block: PILimit



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

Outports	
Out	

Mask Parameters	
Kp	Proportional Factor
Ki	Integral Factor
ts_fact	Multiplication factor of base sampling time (in integer format)

Description:

PI controller with output limitation:
G(s) = Kp + Ki/s

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

Transfer function (zero-order hold discretization method):

$$G(z) = K_P + K_I T_s \frac{1}{z-1}$$

Developer note: The fixed point implementation source code of this block is used for block Block: PI.

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

Name	FiP8
ID	3232
Revision	2.0
C filename	PILimit_FiP8.c
H filename	PILimit_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
sfrb0	Shift factor for PI coefficient b0
sfrb1	Shift factor for PI coefficient b1
i_old	Integrator value of previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
    uint16          ID;
    int8            *In;
    int8            *Init;
    int8            *max;
    int8            *min;
    int8            *Enable;
    int8            Out;
    int8            b0;
    int8            b1;
    int8            sfrb0;
    int8            sfrb1;
    int16           i_old;
    int8            enable_old;
} PILIMIT_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	3233
Revision	2.0
C filename	PILimit_FiP16.c
H filename	PILimit_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
b0	Integrall coefficient
b1	Proportional coefficient
sfrb0	Shift factor for PI coefficient b0
sfrb1	Shift factor for PI coefficient b1
i_old	Integrator value of previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
     uint16          ID;
     int16           *In;
     int16           *Init;
     int16           *max;
     int16           *min;
     int8            *Enable;
     int16           Out;
     int16           b0;
     int16           b1;
     int8            sfrb0;
     int8            sfrb1;
     int32           i_old;
     int8            enable_old;
} PILIMIT_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	3234
Revision	2.0
C filename	PILimit_FiP32.c
H filename	PILimit_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
b0	Integrall coefficient
b1	Proportional coefficient
sfrb0	Shift factor for PI coefficient b0
sfrb1	Shift factor for PI coefficient b1
i_old	Integrator value of previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In;
      int32           *Init;
      int32           *max;
      int32           *min;
      int8            *Enable;
      int32           Out;
      int32           b0;
      int32           b1;
      int8            sfrb0;
      int8            sfrb1;
      int64           i_old;
      int8            enable_old;
} PILIMIT_FIP32;
```

Implementation: Float32

Name Float32
ID 3235
Revision 2.0
C filename PILimit_Float32.c
H filename PILimit_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
i_old	Integrator value of previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
      float32         *Init;
      float32         *max;
```

```
    float32         *min;
    int8            *Enable;
    float32         Out;
    float32         b0;
    float32         b1;
    float32         i_old;
    int8            enable_old;
} PILIMIT_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	3236
Revision	2.0
C filename	PILimit_Float64.c
H filename	PILimit_Float64.h

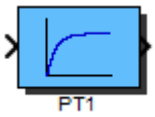
64 Bit Floating Point Implementation

Controller Parameters	
b0	Integral coefficient
b1	Proportional coefficient
i_old	Integrator value of previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
    uint16          ID;
    float64         *In;
    float64         *Init;
    float64         *max;
    float64         *min;
    int8            *Enable;
    float64         Out;
    float64         b0;
    float64         b1;
    float64         i_old;
    int8            enable_old;
} PILIMIT_FLOAT64;
```

Block: PT1



Inports	
In	Input In(k)

Outports	
Out	Output Out(k)

Mask Parameters	
V	Gain
fc	Cut off frequency of low pass filter
ts_fact	Multiplication factor of base sampling time (in integer format)
method	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.

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

Name	FiP8
ID	3312
Revision	0.1
C filename	PT1_FiP8.c
H filename	PT1_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
b0	
b1	
a0	
sfrb	
sfra	
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            Out;
      int8            b0;
      int8            b1;
      int8            a0;
      int8            sfrb;
      int8            sfra;
      int8            in_old;
} PT1_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	3313
Revision	0.1
C filename	PT1_FiP16.c
H filename	PT1_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
b0	
b1	
a0	
sfrb	
sfra	
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In;
      int16           Out;
```

```
    int16          b0;
    int16          b1;
    int16          a0;
    int8           sfrb;
    int8           sfra;
    int16          in_old;
} PT1_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	3314
Revision	0.1
C filename	PT1_FiP32.c
H filename	PT1_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
b0	
b1	
a0	
sfrb	
sfra	
in_old	In(k-1)

Data Structure:

```
typedef struct {
    uint16         ID;
    int32          *In;
    int32          Out;
    int32          b0;
    int32          b1;
    int32          a0;
    int8           sfrb;
    int8           sfra;
    int32          in_old;
} PT1_FIP32;
```

Implementation: Float32

Name	Float32
ID	3315
Revision	0.1
C filename	PT1_Float32.c
H filename	PT1_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
b0	Coefficient b0
b1	Coefficient b1
a0	Coefficient a0
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
      float32         Out;
      float32         b0;
      float32         b1;
      float32         a0;
      float32         in_old;
} PT1_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	3316
Revision	0.1
C filename	PT1_Float64.c
H filename	PT1_Float64.h

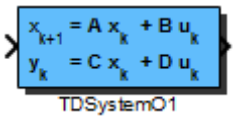
64 Bit Floating Point Implementation

Controller Parameters	
b0	Coefficient b0
b1	Coefficient b1
a0	Coefficient a0
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In;
      float64         Out;
      float64         b0;
      float64         b1;
      float64         a0;
      float64         in_old;
} PT1_FLOAT64;
```

Block: TDSystemO1



Inports	
In	Input #1

Outports	
Out	Output #1

Mask Parameters	
A	State matrix A
B	Input matrix B
C	Output matrix C
D	Feedthrough matrix D

Description:

1st order time discrete system with one input and one output.

Calculation:

$$\begin{aligned} x_{1,k+1} &= a_{11}x_{1,k} + b_{11}u_{1,k} \\ y_{1,k} &= c_{11}x_{1,k} + d_{11}u_{1,k} \end{aligned}$$

or short

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k \end{aligned}$$

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

Name	FiP8
ID	3344
Revision	1
C filename	TDSystemO1_FiP8.c
H filename	TDSystemO1_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
a11	Coefficient a11
b11	Coefficient b11
c11	Coefficient c11
d11	Coefficient d11
sfra11	Shift factor for coefficient a11
sfrb11	Shift factor for coefficient b11
sfrc11	Shift factor for coefficient c11
sfrd11	Shift factor for coefficient d11
x1	State x1

Data Structure:

```
typedef struct {
     uint16          ID;
     int8            *In;
     int8            Out;
     int8            a11;
     int8            b11;
     int8            c11;
     int8            d11;
     uint8           sfra11;
     uint8           sfrb11;
     uint8           sfrc11;
     uint8           sfrd11;
     int8            x1;
} TDSYSTEMO1_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	3345
Revision	1
C filename	TDSystemO1_FiP16.c
H filename	TDSystemO1_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
a11	Coefficient a11
b11	Coefficient b11
c11	Coefficient c11
d11	Coefficient d11
sfra11	Shift factor for coefficient a11
sfrb11	Shift factor for coefficient b11
sfrc11	Shift factor for coefficient c11
sfrd11	Shift factor for coefficient d11
x1	State x1

Data Structure:

```
typedef struct {
     uint16          ID;
     int16           *In;
     int16           Out;
     int16           a11;
     int16           b11;
     int16           c11;
     int16           d11;
     uint8           sfra11;
     uint8           sfrb11;
     uint8           sfrc11;
     uint8           sfrd11;
     int16           x1;
} TDSYSTEMO1_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	3346
Revision	1
C filename	TDSystemO1_FiP32.c
H filename	TDSystemO1_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
a11	Coefficient a11
b11	Coefficient b11
c11	Coefficient c11
d11	Coefficient d11
sfra11	Shift factor for coefficient a11
sfrb11	Shift factor for coefficient b11
sfrc11	Shift factor for coefficient c11
sfrd11	Shift factor for coefficient d11
x1	State x1

Data Structure:

```
typedef struct {
     uint16          ID;
     int32           *In;
     int32           Out;
     int32           a11;
     int32           b11;
     int32           c11;
     int32           d11;
     uint8           sfra11;
     uint8           sfrb11;
     uint8           sfrc11;
     uint8           sfrd11;
     int32           x1;
} TDSYSTEMO1_FIP32;
```

Implementation: Float32

Name Float32
ID 3347
Revision 0.1
C filename TDSystemO1_Float32.c
H filename TDSystemO1_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
a11	Coefficient a11
b11	Coefficient b11
c11	Coefficient c11
d11	Coefficient d11
x1	State x1

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
      float32         Out;
      float32         a11;
      float32         b11;
      float32         c11;
      float32         d11;
      float32         x1;
} TDSYSTEMO1_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	3348
Revision	0.1
C filename	TDSystemO1_Float64.c
H filename	TDSystemO1_Float64.h

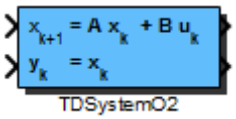
64 Bit Floating Point Implementation

Controller Parameters	
a11	Coefficient a11
b11	Coefficient b11
c11	Coefficient c11
d11	Coefficient d11
x1	State x1

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In;
      float64         Out;
      float64         a11;
      float64         b11;
      float64         c11;
      float64         d11;
      float64         x1;
} TDSYSTEMO1_FLOAT64;
```

Block: TDSystemO2



Inports	
In1	Input #1
In2	Input #2

Outports	
Out1	Output #1
Out2	Output #2

Mask Parameters	
A	State matrix A
B	Input matrix B

Description:

2nd order time discrete system with two inputs and two outputs.

Calculation:

$$\begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \begin{bmatrix} u_{1,k} \\ u_{2,k} \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_{1,k} \\ u_{2,k} \end{bmatrix}$$

or short

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\ \mathbf{y}_k &= \mathbf{x}_k \end{aligned}$$

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

Name	FiP8
ID	3360
Revision	1
C filename	TDSystemO2_FiP8.c
H filename	TDSystemO2_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
a11	Coefficient a11
a12	Coefficient a12
a21	Coefficient a21
a22	Coefficient a22
b11	Coefficient b11
b12	Coefficient b12
b21	Coefficient b21
b22	Coefficient b22
sfra11	Shift factor for coefficient a11
sfra12	Shift factor for coefficient a12
sfra21	Shift factor for coefficient a21
sfra22	Shift factor for coefficient a22
sfrb11	Shift factor for coefficient b11
sfrb12	Shift factor for coefficient b12
sfrb21	Shift factor for coefficient b21
sfrb22	Shift factor for coefficient b22
x1	State x1
x2	State x2

Data Structure:

```
typedef struct {
    uint16          ID;
    int8            *In1;
    int8            *In2;
    int8            Out1;
    int8            Out2;
    int8            a11;
    int8            a12;
    int8            a21;
    int8            a22;
    int8            b11;
    int8            b12;
    int8            b21;
    int8            b22;
    uint8           sfra11;
    uint8           sfra12;
    uint8           sfra21;
```

```
    uint8           sfra22;
    uint8           sfrb11;
    uint8           sfrb12;
    uint8           sfrb21;
    uint8           sfrb22;
    int8            x1;
    int8            x2;
} TDSYSTEMO2_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	3361
Revision	1
C filename	TDSystemO2_FiP16.c
H filename	TDSystemO2_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
a11	Coefficient a11
a12	Coefficient a12
a21	Coefficient a21
a22	Coefficient a22
b11	Coefficient b11
b12	Coefficient b12
b21	Coefficient b21
b22	Coefficient b22
sfra11	Shift factor for coefficient a11
sfra12	Shift factor for coefficient a12
sfra21	Shift factor for coefficient a21
sfra22	Shift factor for coefficient a22
sfrb11	Shift factor for coefficient b11
sfrb12	Shift factor for coefficient b12
sfrb21	Shift factor for coefficient b21
sfrb22	Shift factor for coefficient b22
x1	State x1
x2	State x2

Data Structure:

```
typedef struct {
    uint16          ID;
    int16           *In1;
    int16           *In2;
    int16           Out1;
    int16           Out2;
```

```
    int16          a11;
    int16          a12;
    int16          a21;
    int16          a22;
    int16          b11;
    int16          b12;
    int16          b21;
    int16          b22;
    uint8          sfra11;
    uint8          sfra12;
    uint8          sfra21;
    uint8          sfra22;
    uint8          sfrb11;
    uint8          sfrb12;
    uint8          sfrb21;
    uint8          sfrb22;
    int16          x1;
    int16          x2;
} TDSYSTEMO2_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	3362
Revision	1
C filename	TDSystemO2_FiP32.c
H filename	TDSystemO2_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
a11	Coefficient a11
a12	Coefficient a12
a21	Coefficient a21
a22	Coefficient a22
b11	Coefficient b11
b12	Coefficient b12
b21	Coefficient b21
b22	Coefficient b22
sfra11	Shift factor for coefficient a11
sfra12	Shift factor for coefficient a12
sfra21	Shift factor for coefficient a21
sfra22	Shift factor for coefficient a22
sfrb11	Shift factor for coefficient b11
sfrb12	Shift factor for coefficient b12
sfrb21	Shift factor for coefficient b21
sfrb22	Shift factor for coefficient b22
x1	State x1
x2	State x2

Data Structure:

```
typedef struct {
     uint16          ID;
     int32           *In1;
     int32           *In2;
     int32           Out1;
     int32           Out2;
     int32           a11;
     int32           a12;
     int32           a21;
     int32           a22;
     int32           b11;
     int32           b12;
     int32           b21;
     int32           b22;
     uint8           sfra11;
     uint8           sfra12;
     uint8           sfra21;
     uint8           sfra22;
     uint8           sfrb11;
     uint8           sfrb12;
     uint8           sfrb21;
     uint8           sfrb22;
     int32           x1;
     int32           x2;
} TDSYSTEMO2_FIP32;
```

Implementation: Float32

Name	Float32
ID	3363
Revision	0.1
C filename	TDSystemO2_Float32.c
H filename	TDSystemO2_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
a11	Coefficient a11
a12	Coefficient a12
a21	Coefficient a21
a22	Coefficient a22
b11	Coefficient b11
b12	Coefficient b12
b21	Coefficient b21
b22	Coefficient b22
x1	State x1
x2	State x2

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In1;
      float32         *In2;
      float32         Out1;
      float32         Out2;
      float32         a11;
      float32         a12;
      float32         a21;
      float32         a22;
      float32         b11;
      float32         b12;
      float32         b21;
      float32         b22;
      float32         x1;
      float32         x2;
} TDSYSTEMO2_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	3364
Revision	0.1
C filename	TDSystemO2_Float64.c
H filename	TDSystemO2_Float64.h

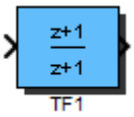
64 Bit Floating Point Implementation

Controller Parameters	
a11	Coefficient a11
a12	Coefficient a12
a21	Coefficient a21
a22	Coefficient a22
b11	Coefficient b11
b12	Coefficient b12
b21	Coefficient b21
b22	Coefficient b22
x1	State x1
x2	State x2

Data Structure:

```
typedef struct {
     uint16         ID;
     float64        *In1;
     float64        *In2;
     float64        Out1;
     float64        Out2;
     float64        a11;
     float64        a12;
     float64        a21;
     float64        a22;
     float64        b11;
     float64        b12;
     float64        b21;
     float64        b22;
     float64        x1;
     float64        x2;
} TDSYSTEMO2_FLOAT64;
```

Block: TF1



Inports	
In	Input In(k)

Outports	
Out	Output Out(k)

Mask Parameters	
b1	b1
b0	b0
a0	a0
ts_fact	Multiplication factor of base sampling time (in integer format)

Description:

First order transfer function:
G(z) = (b1.z + b0) / (z + a0)

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

Developer note: The source code of this block is used for blocks *DT1* and *PT1*.

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

Name	FiP8
ID	3280
Revision	0.1
C filename	TF1_FiP8.c
H filename	TF1_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
b0	Coefficient b0
b1	Coefficient b1
a0	Coefficient a0
sfrb	Shift factor for coefficient b0 and b1
sfra	Shift factor for coefficient a0
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            Out;
      int8            b0;
      int8            b1;
      int8            a0;
      int8            sfrb;
      int8            sfra;
      int8            in_old;
} TF1_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	3281
Revision	0.1
C filename	TF1_FiP16.c
H filename	TF1_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
b0	Coefficient b0
b1	Coefficient b1
a0	Coefficient a0
sfrb	Shift factor for coefficient b0 and b1
sfra	Shift factor for coefficient a0
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In;
      int16           Out;
```

```
    int16           b0;
    int16           b1;
    int16           a0;
    int8            sfrb;
    int8            sfra;
    int16           in_old;
} TF1_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	3282
Revision	0.1
C filename	TF1_FiP32.c
H filename	TF1_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
b0	Coefficient b0
b1	Coefficient b1
a0	Coefficient a0
sfrb	Shift factor for coefficient b0 and b1
sfra	Shift factor for coefficient a0
in_old	In(k-1)

Data Structure:

```
typedef struct {
    uint16          ID;
    int32           *In;
    int32           Out;
    int32           b0;
    int32           b1;
    int32           a0;
    int8            sfrb;
    int8            sfra;
    int32           in_old;
} TF1_FIP32;
```

Implementation: Float32

Name	Float32
ID	3283
Revision	0.1
C filename	TF1_Float32.c
H filename	TF1_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
b0	Coefficient b0
b1	Coefficient b1
a0	Coefficient a0
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
      float32         Out;
      float32         b0;
      float32         b1;
      float32         a0;
      float32         in_old;
} TF1_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	3284
Revision	0.1
C filename	TF1_Float64.c
H filename	TF1_Float64.h

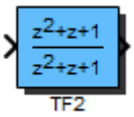
64 Bit Floating Point Implementation

Controller Parameters	
b0	Coefficient b0
b1	Coefficient b1
a0	Coefficient a0
in_old	In(k-1)

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In;
      float64         Out;
      float64         b0;
      float64         b1;
      float64         a0;
      float64         in_old;
} TF1_FLOAT64;
```

Block: TF2



Inports	
In	Input In(k)

Outports	
Out	Output Out(k)

Mask Parameters	
b2	b2
b1	b1
b0	b0
a1	a1
a0	a0
ts_fact	Multiplication factor of base sampling time (in integer format)

Description:

Second order transfer function:
G(z) = (b2.z^2 + b1.z + b0) / (z^2 + a1.z + a0)

Implementations:

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

Implementation: FiP16

Name	FiP16
ID	3297
Revision	0.1
C filename	TF2_FiP16.c
H filename	TF2_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
b0	
b1	
b2	
a0	
a1	
sfrb	
sfra	
in_old	In(k-1)
in_veryold	In(k-2)
out_old	Out(k-1)
out_veryold	Out(k-2)

Data Structure:

```
typedef struct {
     uint16          ID;
     int16           *In;
     int16           Out;
     int16           b0;
     int16           b1;
     int16           b2;
     int16           a0;
     int16           a1;
     int8            sfrb;
     int8            sfra;
     int16           in_old;
     int16           in_veryold;
     int16           out_old;
     int16           out_veryold;
} TF2_FIP16;
```

Implementation: FiP8

Name	FiP8
ID	3296
Revision	0.1
C filename	TF2_FiP8.c
H filename	TF2_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
b0	
b1	
b2	
a0	
a1	
sfrb	
sfra	
in_old	In(k-1)
in_veryold	In(k-2)
out_old	Out(k-1)
out_veryold	Out(k-2)

Data Structure:

```
typedef struct {
     uint16          ID;
     int8            *In;
     int8            Out;
     int8            b0;
     int8            b1;
     int8            b2;
     int8            a0;
     int8            a1;
     int8            sfrb;
     int8            sfra;
     int8            in_old;
     int8            in_veryold;
     int8            out_old;
     int8            out_veryold;
} TF2_FIP8;
```

Implementation: FiP32

Name	FiP32
ID	3298
Revision	0.1
C filename	TF2_FiP32.c
H filename	TF2_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
b0	
b1	
b2	
a0	
a1	
sfrb	
sfra	
in_old	In(k-1)
in_veryold	In(k-2)
out_old	Out(k-1)
out_veryold	Out(k-2)

Data Structure:

```
typedef struct {
     uint16          ID;
     int32           *In;
     int32           Out;
     int32           b0;
     int32           b1;
     int32           b2;
     int32           a0;
     int32           a1;
     int8            sfrb;
     int8            sfra;
     int32           in_old;
     int32           in_veryold;
     int32           out_old;
     int32           out_veryold;
} TF2_FIP32;
```

Implementation: Float32

Name	Float32
ID	3299
Revision	0.1
C filename	TF2_Float32.c
H filename	TF2_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
b0	
b1	
b2	
a0	
a1	
in_old	In(k-1)
in_veryold	In(k-2)
out_old	Out(k-1)
out_veryold	Out(k-2)

Data Structure:

```
typedef struct {
     uint16         ID;
     float32        *In;
     float32        Out;
     float32        b0;
     float32        b1;
     float32        b2;
     float32        a0;
     float32        a1;
     float32        in_old;
     float32        in_veryold;
     float32        out_old;
     float32        out_veryold;
} TF2_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	3300
Revision	0.1
C filename	TF2_Float64.c
H filename	TF2_Float64.h

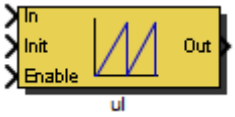
64 Bit Floating Point Implementation

Controller Parameters	
b0	
b1	
b2	
a0	
a1	
in_old	In(k-1)
in_veryold	In(k-2)
out_old	Out(k-1)
out_veryold	Out(k-2)

Data Structure:

```
typedef struct {
     uint16         ID;
     float64        *In;
     float64        Out;
     float64        b0;
     float64        b1;
     float64        b2;
     float64        a0;
     float64        a1;
     float64        in_old;
     float64        in_veryold;
     float64        out_old;
     float64        out_veryold;
} TF2_FLOAT64;
```

Block: ul



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

Outports	
Out	Integrator output

Mask Parameters	
Ki	Integral Factor
ts_fact	Multiplication factor of base sampling time (in integer format)

Description:

Integrator for angle signals:
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):

$$G(z) = K_I T_s \frac{1}{z-1}$$

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

Name FiP8
ID 3376
Revision 1.0
C filename ul_FiP8.c
H filename ul_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
sfr	Shift factor for I coefficient b0
i_old	Integrator value from previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            *Init;
      int8            *Enable;
      int8            Out;
      int8            b0;
      int8            sfr;
      int16           i_old;
      int8            enable_old;
} UI_FIP8;
```

Implementation: FiP16

Name FiP16
ID 3377
Revision 1.0
C filename ul_FiP16.c
H filename ul_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
sfr	Shift factor for I coefficient b0
i_old	Integrator value from previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
```

```
    uint16          ID;
    int16           *In;
    int16           *Init;
    int8            *Enable;
    int16           Out;
    int16           b0;
    int8            sfr;
    int32           i_old;
    int8            enable_old;
} UI_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	3378
Revision	1.0
C filename	ul_FiP32.c
H filename	ul_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
b0	Integral coefficient
sfr	Shift factor for I coefficient b0
i_old	Integrator value from previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
    uint16          ID;
    int32           *In;
    int32           *Init;
    int8            *Enable;
    int32           Out;
    int32           b0;
    int8            sfr;
    int64           i_old;
    int8            enable_old;
} UI_FIP32;
```

Implementation: Float32

Name	Float32
ID	3379
Revision	1.0
C filename	ul_Float32.c
H filename	ul_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
b0	Integral coefficient
i_old	Integrator value from previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
     uint16          ID;
     float32         *In;
     float32         *Init;
     int8            *Enable;
     float32         Out;
     float32         b0;
     float32         i_old;
     int8            enable_old;
} UI_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	3380
Revision	1.0
C filename	ul_Float64.c
H filename	ul_Float64.h

64 Bit Floating Point Implementation

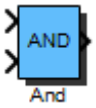
Controller Parameters	
b0	Integral coefficient
i_old	Integrator value from previous cycle
enable_old	Enable value of previous cycle

Data Structure:

```
typedef struct {
     uint16          ID;
     float64         *In;
     float64         *Init;
     int8            *Enable;
     float64         Out;
     float64         b0;
     float64         i_old;
     int8            enable_old;
} UI_FLOAT64;
```

18 General

Block: And



Inports	
In1	
In2	

Outports	
Out	

Description:

Logical AND block.

Implementations:

FiP8	8 Bit Fixed Point Implementation
FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation

Implementation: FiP8

Name	FiP8
ID	240
Revision	0.1
C filename	And_FiP8.c
H filename	And_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int8            *In1;
     int8            *In2;
     int8            Out;
} AND_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	241
Revision	0.1
C filename	And_FiP16.c
H filename	And_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int16           *In1;
     int16           *In2;
     int16           Out;
} AND_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	242
Revision	0.1
C filename	And_FiP32.c
H filename	And_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int32           *In1;
     int32           *In2;
     int32           Out;
} AND_FIP32;
```

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	
Thresh_up	Threshold level for rising switch signal
Thresh_down	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:

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

Name	FiP8
ID	128
Revision	0.1
C filename	AutoSwitch_FiP8.c
H filename	AutoSwitch_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
Thresh_up	Threshold level for rising switch signal
Thresh_down	Threshold level for falling switch signal
Status	Current hysteresis state

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In1;
      int8            *Switch;
      int8            *In3;
      int8            Out;
      int8            Thresh_up;
      int8            Thresh_down;
      int8            Status;
} AUTOSWITCH_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	129
Revision	0.1
C filename	AutoSwitch_FiP16.c
H filename	AutoSwitch_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
Thresh_up	Threshold level for rising switch signal
Thresh_down	Threshold level for falling switch signal
Status	Current hysteresis state

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In1;
      int16           *Switch;
      int16           *In3;
      int16           Out;
      int16           Thresh_up;
      int16           Thresh_down;
      int8            Status;
} AUTOSWITCH_FIP16;
```

Implementation: FiP32

Name FiP32
ID 130
Revision 0.1
C filename AutoSwitch_FiP32.c
H filename AutoSwitch_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
Thresh_up	Threshold level for rising switch signal
Thresh_down	Threshold level for falling switch signal
Status	Current hysteresis state

Data Structure:

```
typedef struct {
     uint16          ID;
     int32           *In1;
     int32           *Switch;
     int32           *In3;
     int32           Out;
     int32           Thresh_up;
     int32           Thresh_down;
     int8            Status;
} AUTOSWITCH_FIP32;
```

Implementation: Float32

Name Float32
ID 131
Revision 0.1
C filename AutoSwitch_Float32.c
H filename AutoSwitch_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
Thresh_up	Threshold level for rising switch signal
Thresh_down	Threshold level for falling switch signal
Status	Current hysteresis state

Data Structure:

```
typedef struct {
     uint16          ID;
     float32         *In1;
     float32         *Switch;
     float32         *In3;
     float32         Out;
     float32         Thresh_up;
```

```
    float32         Thresh_down;
    int8            Status;
} AUTOSWITCH_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	132
Revision	0.1
C filename	AutoSwitch_Float64.c
H filename	AutoSwitch_Float64.h

64 Bit Floating Point Implementation

Controller Parameters	
Thresh_up	Threshold level for rising switch signal
Thresh_down	Threshold level for falling switch signal
Status	Current hysteresis state

Data Structure:

```
typedef struct {
    uint16          ID;
    float64         *In1;
    float64         *Switch;
    float64         *In3;
    float64         Out;
    float64         Thresh_up;
    float64         Thresh_down;
    int8            Status;
} AUTOSWITCH_FLOAT64;
```

Block: Constant

Outports	
Out	Constant output

Mask Parameters	
Value	Constant factor

Description:

Constant value.

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

Name	FiP8
ID	48
Revision	0.3
C filename	Constant_FiP8.c
H filename	Constant_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
K	Constant factor

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            Out;
      int8            K;
} CONSTANT_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	49
Revision	0.3
C filename	Constant_FiP16.c
H filename	Constant_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
K	Constant factor

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           Out;
      int16           K;
} CONSTANT_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	50
Revision	0.3
C filename	Constant_FiP32.c
H filename	Constant_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
K	Constant factor

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           Out;
      int32           K;
} CONSTANT_FIP32;
```

Implementation: Float32

Name	Float32
ID	51
Revision	0.1
C filename	Constant_Float32.c
H filename	Constant_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
K	Constant factor

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         Out;
      float32         K;
} CONSTANT_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	52
Revision	0.1
C filename	Constant_Float64.c
H filename	Constant_Float64.h

64 Bit Floating Point Implementation

Controller Parameters	
K	Constant factor

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         Out;
      float64         K;
} CONSTANT_FLOAT64;
```

Block: Gain

Inports	
In	Input

Outports	
Out	Amplified input

Mask Parameters	
Gain	Gain factor in floating point format

Description:

Amplification of input by gain factor.

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

Name	FiP8
ID	16
Revision	1.0
C filename	Gain_FiP8.c
H filename	Gain_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
V	Gain factor
sfr	Shift factor

Data Structure:

```
typedef struct {
    uint16          ID;
    int8            *In;
```

```
    int8            Out;
    int8            V;
    int8            sfr;
} GAIN_FIP8;
```

Implementation: FiP16

Name FiP16
ID 17
Revision 1.0
C filename Gain_FiP16.c
H filename Gain_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
V	Gain factor
sfr	Shift factor

Data Structure:

```
typedef struct {
    uint16          ID;
    int16           *In;
    int16           Out;
    int16           V;
    int8            sfr;
} GAIN_FIP16;
```

Implementation: FiP32

Name FiP32
ID 18
Revision 1.0
C filename Gain_FiP32.c
H filename Gain_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
V	Gain factor
sfr	Shift factor

Data Structure:

```
typedef struct {
    uint16          ID;
    int32           *In;
```

```
    int32           Out;
    int32           V;
    int8            sfr;
} GAIN_FIP32;
```

Implementation: Float32

Name	Float32
ID	19
Revision	0.1
C filename	Gain_Float32.c
H filename	Gain_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
V	Gain factor

Data Structure:

```
typedef struct {
    uint16          ID;
    float32         *In;
    float32         Out;
    float32         V;
} GAIN_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	20
Revision	0.1
C filename	Gain_Float64.c
H filename	Gain_Float64.h

64 Bit Floating Point Implementation

Controller Parameters	
V	Gain factor

Data Structure:

```
typedef struct {
    uint16          ID;
    float64         *In;
    float64         Out;
    float64         V;
} GAIN_FLOAT64;
```

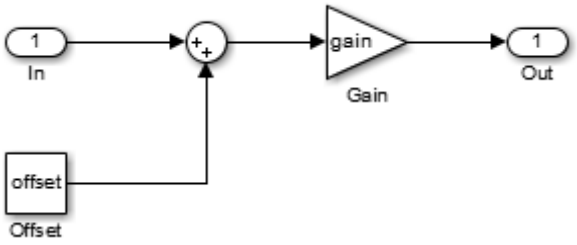
Block: Inport

Inports	
IN	Signal from frame program

Mask Parameters	
ts_fact	Multiplication factor of base sampling time (in integer format)
Gain	Gain value used in simulation
Offset	Offset value used in simulation

Description:

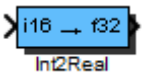
Serves as interface to the frame program. The input of this block is intended for simulation purposes and can be left unconnected if not used. Also the parameters *Gain* and *Offset* are only used during simulation. The schematic for simulation can be seen in the figure below.



Data Types:

int8	8 Bit Fixed Point
int16	16 Bit Fixed Point
int32	32 Bit Fixed Point
float32	32 Bit Floating Point
float64	64 Bit Floating Point

Block: Int2Real



Inports	
In	Integer input

Outports	
Out	Real output

Mask Parameters	
Scale	Scaling factor from integer to real

Description:

Conversion block from integer (fixed point) datatypes to real (floating point) datatypes.
Out = In * Scale

Implementations:

FiP8_Float32	8 Bit Fixed Point to 32 Bit Floating Point Implementation
FiP16_Float32	16 Bit Fixed Point to 32 Bit Floating Point Implementation
FiP32_Float32	32 Bit Fixed Point to 32 Bit Floating Point Implementation
FiP8_Float64	8 Bit Fixed Point to 64 Bit Floating Point Implementation
FiP16_Float64	16 Bit Fixed Point to 64 Bit Floating Point Implementation
FiP32_Float64	32 Bit Fixed Point to 64 Bit Floating Point Implementation

Implementation: FiP8_Float32

Name	FiP8_Float32
ID	192
Revision	0.1
C filename	Int2Real_FiP8_Float32.c
H filename	Int2Real_FiP8_Float32.h

8 Bit Fixed Point to 32 Bit Floating Point Implementation

Controller Parameters	
scale	Scaling factor

Data Structure:

```
typedef struct {
    uint16        ID;
```

```
    int8            *In;
    float32         Out;
    float32         scale;
} INT2REAL_FIP8_FLOAT32;
```

Implementation: FiP16_Float32

Name	FiP16_Float32
ID	193
Revision	0.1
C filename	Int2Real_FiP16_Float32.c
H filename	Int2Real_FiP16_Float32.h

16 Bit Fixed Point to 32 Bit Floating Point Implementation

Controller Parameters	
scale	Scaling factor

Data Structure:

```
typedef struct {
    uint16          ID;
    int16           *In;
    float32         Out;
    float32         scale;
} INT2REAL_FIP16_FLOAT32;
```

Implementation: FiP32_Float32

Name	FiP32_Float32
ID	194
Revision	0.1
C filename	Int2Real_FiP32_Float32.c
H filename	Int2Real_FiP32_Float32.h

32 Bit Fixed Point to 32 Bit Floating Point Implementation

Controller Parameters	
scale	Scaling factor

Data Structure:

```
typedef struct {
    uint16          ID;
    int32           *In;
    float32         Out;
    float32         scale;
} INT2REAL_FIP32_FLOAT32;
```

Implementation: FiP8_Float64

Name	FiP8_Float64
ID	195
Revision	0.1
C filename	Int2Real_FiP8_Float64.c
H filename	Int2Real_FiP8_Float64.h

8 Bit Fixed Point to 64 Bit Floating Point Implementation

Controller Parameters	
scale	Scaling factor

Data Structure:

```
typedef struct {
    uint16          ID;
    int8            *In;
    float64         Out;
    float64         scale;
} INT2REAL_FIP8_FLOAT64;
```

Implementation: FiP16_Float64

Name	FiP16_Float64
ID	196
Revision	0.1
C filename	Int2Real_FiP16_Float64.c
H filename	Int2Real_FiP16_Float64.h

16 Bit Fixed Point to 64 Bit Floating Point Implementation

Controller Parameters	
scale	Scaling factor

Data Structure:

```
typedef struct {
    uint16          ID;
    int16           *In;
    float64         Out;
    float64         scale;
} INT2REAL_FIP16_FLOAT64;
```

Implementation: FiP32_Float64

Name	FiP32_Float64
ID	197
Revision	0.1
C filename	Int2Real_FiP32_Float64.c
H filename	Int2Real_FiP32_Float64.h

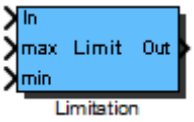
32 Bit Fixed Point to 64 Bit Floating Point Implementation

Controller Parameters	
scale	Scaling factor

Data Structure:

```
typedef struct {
     uint16          ID;
     int32           *In;
     float64         Out;
     float64         scale;
} INT2REAL_FIP32_FLOAT64;
```

Block: Limitation



Inports	
In	Input signal
max	Upper limit
min	Lower limit

Outports	
Out	Limited input signal

Description:

Limits the input signal to min and max_sci.
Caution: For correct computation the upper limit max has to be greater than the lower limit min!

Calculation:

$$Out = \begin{cases} max & In > max \\ In & \\ min & In < min \end{cases}$$

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

Name	FiP8
ID	384
Revision	0.1
C filename	Limitation_FiP8.c
H filename	Limitation_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            *max;
      int8            *min;
      int8            Out;
} LIMITATION_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	385
Revision	0.1
C filename	Limitation_FiP16.c
H filename	Limitation_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In;
      int16           *max;
      int16           *min;
      int16           Out;
} LIMITATION_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	386
Revision	0.1
C filename	Limitation_FiP32.c
H filename	Limitation_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In;
      int32           *max;
      int32           *min;
      int32           Out;
} LIMITATION_FIP32;
```

Implementation: Float32

Name	Float32
ID	387
Revision	0.1
C filename	Limitation_Float32.c
H filename	Limitation_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
      float32         *max;
      float32         *min;
      float32         Out;
} LIMITATION_FLOAT32;
```

Implementation: Float64

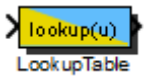
Name	Float64
ID	388
Revision	0.1
C filename	Limitation_Float64.c
H filename	Limitation_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In;
      float64         *max;
      float64         *min;
      float64         Out;
} LIMITATION_FLOAT64;
```

Block: LookupTable



Inports	
In	Table index

Outports	
Out	Table output

Mask Parameters	
Lookup	Look-up Table

Description:

Look-up Table with 256+1 values.
Note: 257th value is used for preventing index overflow during interpolation.
-> for periodic signals the 257th value should be set equal to 1st value
-> for non-periodic signals the 257th value should be set equal to 256th value

Implementations:

FiP8 8 Bit Fixed Point Implementation
FiP16 16 Bit Fixed Point Implementation
FiP32 32 Bit Fixed Point Implementation

Implementation: FiP8

Name FiP8
ID 160
Revision 0.1
C filename LookupTable_FiP8.c
H filename LookupTable_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
Table	Lookup table content

Data Structure:

```
typedef struct {
    uint16          ID;
    int8            *In;
```

```
    int8            Out;
    const int8      *Table;
} LOOKUPTABLE_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	161
Revision	0.1
C filename	LookupTable_FiP16.c
H filename	LookupTable_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
Table	Lookup table content

Data Structure:

```
typedef struct {
    uint16          ID;
    int16           *In;
    int16           Out;
    const int16     *Table;
} LOOKUPTABLE_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	162
Revision	0.1
C filename	LookupTable_FiP32.c
H filename	LookupTable_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
Table	Lookup table content

Data Structure:

```
typedef struct {
    uint16          ID;
    int32           *In;
    int32           Out;
    const int32     *Table;
} LOOKUPTABLE_FIP32;
```

Block: LoopBreaker

Inports	
In	Input In(k)

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

Description:

Block to break algebraic loops.

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

Name	FiP16
ID	481
Revision	0.1
C filename	LoopBreaker_FiP16.c
H filename	LoopBreaker_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int16           *In;
     int16           Out;
} LOOPBREAKER_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	482
Revision	0.1
C filename	LoopBreaker_FiP32.c
H filename	LoopBreaker_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In;
      int32           Out;
} LOOPBREAKER_FIP32;
```

Implementation: Float32

Name	Float32
ID	483
Revision	0.1
C filename	LoopBreaker_Float32.c
H filename	LoopBreaker_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
      float32         Out;
} LOOPBREAKER_FLOAT32;
```

Implementation: Float64

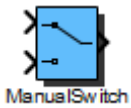
Name	Float64
ID	484
Revision	0.1
C filename	LoopBreaker_Float64.c
H filename	LoopBreaker_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In;
      float64         Out;
} LOOPBREAKER_FLOAT64;
```

Block: ManualSwitch



Inports	
In1	Input #1
In2	Input #2

Outports	
Out	

Mask Parameters	
Toggle	Toggle

Description:

Toggling between inputs by double-clicking on block.

Doubleclicking of the *ManualSwitch* block changes the routing of the input signals and doesn't open the *Function Block Parameters* dialog. So if changing the implementation is required, one has to open the dialog via *Mask Parameters* command of the context menu.

Developer note: To get the double-click feature the callback function of *OpenFnc* in *Block Properties* is manually altered to

```
if get_param(gcb,'Toggle') == '0'
    set_param(gcb,'Toggle', '1');
else
   set_param(gcb,'Toggle', '0');
end
setBlockData(gcs, gcb);
initSFunction(gcb);
```

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

Name	FiP8
ID	144
Revision	1
C filename	ManualSwitch_FiP8.c
H filename	ManualSwitch_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
Toggle	Toggle info

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In1;
      int8            *In2;
      int8            Out;
      int8            Toggle;
} MANUALSWITCH_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	145
Revision	1
C filename	ManualSwitch_FiP16.c
H filename	ManualSwitch_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
Toggle	Toggle info

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In1;
      int16           *In2;
      int16           Out;
      int8            Toggle;
} MANUALSWITCH_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	146
Revision	1
C filename	ManualSwitch_FiP32.c
H filename	ManualSwitch_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
Toggle	Toggle info

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In1;
      int32           *In2;
      int32           Out;
      int8            Toggle;
} MANUALSWITCH_FIP32;
```

Implementation: Float32

Name	Float32
ID	147
Revision	0.1
C filename	ManualSwitch_Float32.c
H filename	ManualSwitch_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
Toggle	Toggle info

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In1;
      float32         *In2;
      float32         Out;
      int8            Toggle;
} MANUALSWITCH_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	148
Revision	0.1
C filename	ManualSwitch_Float64.c
H filename	ManualSwitch_Float64.h

64 Bit Floating Point Implementation

Controller Parameters	
Toggle	Toggle info

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In1;
      float64         *In2;
      float64         Out;
      int8            Toggle;
} MANUALSWITCH_FLOAT64;
```

Block: Maximum

Inports	
In1	Input #1
In2	Input #2

Outports	
Out	Maximum of Input #1 and Input #2

Description:

Outputs the greater value of the two input signals.

Calculation:

$$Out = max(In_1, In_2)$$

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

Name	FiP8
ID	368
Revision	0.1
C filename	Maximum_FiP8.c
H filename	Maximum_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In1;
      int8            *In2;
      int8            Out;
} MAXIMUM_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	369
Revision	0.1
C filename	Maximum_FiP16.c
H filename	Maximum_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In1;
      int16           *In2;
      int16           Out;
} MAXIMUM_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	370
Revision	0.1
C filename	Maximum_FiP32.c
H filename	Maximum_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In1;
      int32           *In2;
      int32           Out;
} MAXIMUM_FIP32;
```

Implementation: Float32

Name	Float32
ID	371
Revision	0.1
C filename	Maximum_Float32.c
H filename	Maximum_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In1;
      float32         *In2;
      float32         Out;
} MAXIMUM_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	372
Revision	0.1
C filename	Maximum_Float64.c
H filename	Maximum_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In1;
      float64         *In2;
      float64         Out;
} MAXIMUM_FLOAT64;
```

Block: Minimum

Inports	
In1	Input #1
In2	Input #2

Outports	
Out	Minimum of Input #1 and Input #2

Description:

Outputs the lesser value of the two input signals.

Calculation:

$$Out = min(In_1, In_2)$$

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

Name	FiP8
ID	352
Revision	0.1
C filename	Minimum_FiP8.c
H filename	Minimum_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In1;
      int8            *In2;
      int8            Out;
} MINIMUM_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	353
Revision	0.1
C filename	Minimum_FiP16.c
H filename	Minimum_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In1;
      int16           *In2;
      int16           Out;
} MINIMUM_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	354
Revision	0.1
C filename	Minimum_FiP32.c
H filename	Minimum_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In1;
      int32           *In2;
      int32           Out;
} MINIMUM_FIP32;
```

Implementation: Float32

Name	Float32
ID	355
Revision	0.1
C filename	Minimum_Float32.c
H filename	Minimum_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In1;
      float32         *In2;
      float32         Out;
} MINIMUM_FLOAT32;
```

Implementation: Float64

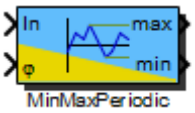
Name	Float64
ID	356
Revision	0.1
C filename	Minimum_Float64.c
H filename	Minimum_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In1;
      float64         *In2;
      float64         Out;
} MINIMUM_FLOAT64;
```

Block: MinMaxPeriodic



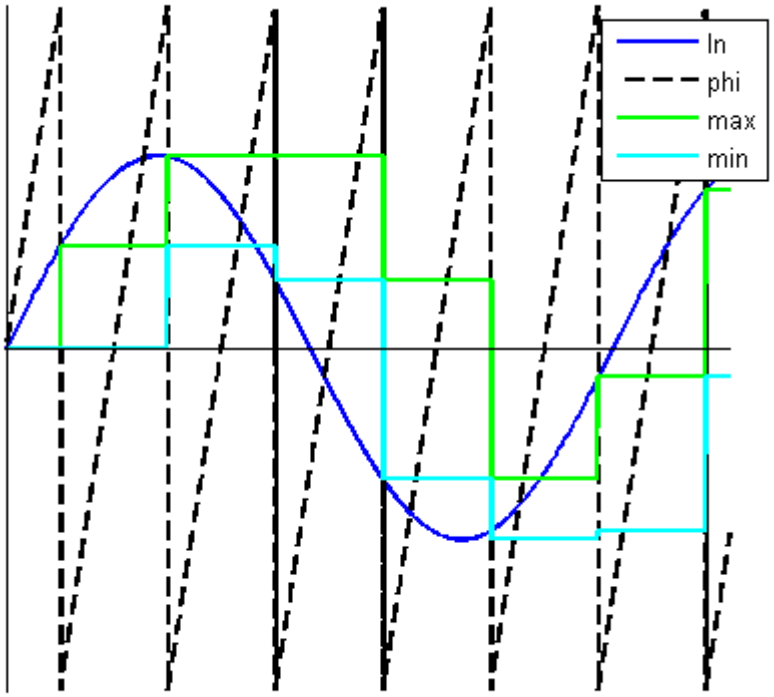
Inports	
In	Input signal
phi	Angle signal

Outports	
max	Maximum of input signal
min	Minimum of input signal

Description:

Outputs the minimum and maximum of the input signal over one period of the (angle) signal phi.

Exemplary signal waveforms:



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

Name	FiP8
ID	464
Revision	0.1
C filename	MinMaxPeriodic_FiP8.c
H filename	MinMaxPeriodic_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
min_act	Current minimum
max_act	Current maximum
phi_old	Angle signal from previous cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            *phi;
      int8            max;
      int8            min;
      int8            min_act;
      int8            max_act;
      int8            phi_old;
} MINMAXPERIODIC_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	465
Revision	0.1
C filename	MinMaxPeriodic_FiP16.c
H filename	MinMaxPeriodic_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
min_act	Current minimum
max_act	Current maximum
phi_old	Angle signal from previous cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In;
      int16           *phi;
      int16           max;
      int16           min;
      int16           min_act;
      int16           max_act;
      int16           phi_old;
} MINMAXPERIODIC_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	466
Revision	0.1
C filename	MinMaxPeriodic_FiP32.c
H filename	MinMaxPeriodic_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
min_act	Current minimum
max_act	Current maximum
phi_old	Angle signal from previous cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In;
      int32           *phi;
      int32           max;
      int32           min;
      int32           min_act;
      int32           max_act;
      int32           phi_old;
} MINMAXPERIODIC_FIP32;
```

Implementation: Float32

Name	Float32
ID	467
Revision	0.1
C filename	MinMaxPeriodic_Float32.c
H filename	MinMaxPeriodic_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
min_act	Current minimum
max_act	Current maximum
phi_old	Angle signal from previous cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
      float32         *phi;
      float32         max;
      float32         min;
      float32         min_act;
      float32         max_act;
      float32         phi_old;
} MINMAXPERIODIC_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	468
Revision	0.1
C filename	MinMaxPeriodic_Float64.c
H filename	MinMaxPeriodic_Float64.h

64 Bit Floating Point Implementation

Controller Parameters	
min_act	Current minimum
max_act	Current maximum
phi_old	Angle signal from previous cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In;
      float64         *phi;
      float64         max;
      float64         min;
      float64         min_act;
```

```
    float64         max_act;
    float64         phi_old;
} MINMAXPERIODIC_FLOAT64;
```

Block: Not

Inports	
In	

Outports	
Out	

Description:

Logical inverter block.

Implementations:

FiP8	8 Bit Fixed Point Implementation
FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation

Implementation: FiP8

Name	FiP8
ID	224
Revision	0.1
C filename	Not_FiP8.c
H filename	Not_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int8            *In;
     int8            Out;
} NOT_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	225
Revision	0.1
C filename	Not_FiP16.c
H filename	Not_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int16           *In;
     int16           Out;
} NOT_FIP16;
```

Implementation: FiP32

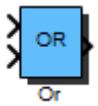
Name	FiP32
ID	226
Revision	0.1
C filename	Not_FiP32.c
H filename	Not_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int32           *In;
     int32           Out;
} NOT_FIP32;
```

Block: Or



Inports	
In1	
In2	

Outports	
Out	

Description:

Logical OR block.

Implementations:

FiP8	8 Bit Fixed Point Implementation
FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation

Implementation: FiP8

Name	FiP8
ID	256
Revision	0.1
C filename	Or_FiP8.c
H filename	Or_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int8            *In1;
     int8            *In2;
     int8            Out;
} OR_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	257
Revision	0.1
C filename	Or_FiP16.c
H filename	Or_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In1;
      int16           *In2;
      int16           Out;
} OR_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	258
Revision	0.1
C filename	Or_FiP32.c
H filename	Or_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In1;
      int32           *In2;
      int32           Out;
} OR_FIP32;
```

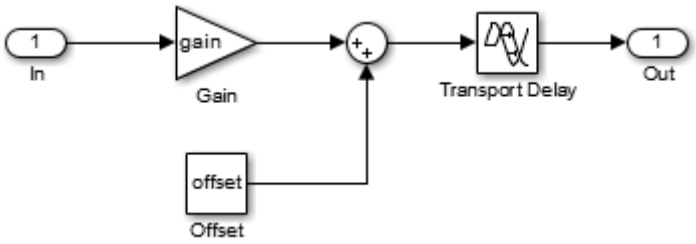
Block: Outport

Outports	
OUT	Signal to frame program

Mask Parameters	
ts_fact	Multiplication factor of base sampling time (in integer format)
Gain	Gain value used in simulation
Offset	Offset value used in simulation
Delay	Time delay of signal used in simulation

Description:

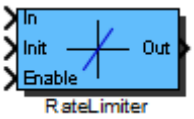
Serves as interface to the frame program. The output of this block is intended for simulation purposes and can be left unconnected if not used. Also the parameters *Gain*, *Offset*, and *Delay* are only used during simulation. The schematic for simulation can be seen in the figure below.



Data Types:

int8	8 Bit Fixed Point
int16	16 Bit Fixed Point
int32	32 Bit Fixed Point
float32	32 Bit Floating Point
float64	64 Bit Floating Point

Block: RateLimiter



Inports	
In	
Init	Value which is loaded at rising flanke of enable signal
Enable	Enable == 0: Deactivation of block; Out is set to In. Enable != 0: Activation of block; Out is rate limited. Enable 0->1: Preloading of output; Out is set to value of Init input

Outports	
Out	

Mask Parameters	
Tr	Rising time in seconds. Slew rate will be 1/Tr
Tf	Falling time in seconds. Slew rate will be 1/Tf
ts_fact	Multiplication factor of base sampling time (in integer format)

Description:

Limitation of rising and falling rate.
Function of Enable:
0: rate limiting disabled, signal is passed through
1: rate limiting enabled, signal is rate limited
0->1: preload of output with value from init input

Rising and falling time refer to a step from 0 to 1. Entries for *Tr: Rising time* and *Tf: Falling time* smaller than the actual sample time will be limited to the sample time internally.
The 16- and 32-Bit fixed point implementations are based on an internal 32-Bit wide slew-rate variable while the 8-Bit fixed point implementation uses a 16-Bit wide slew-rate variable.

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

Name FiP8
ID 96
Revision 1.0
C filename RateLimiter_FiP8.c
H filename RateLimiter_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
out_old	Output value from last cycle in int16 format
enable_old	Enable value from last cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            *Init;
      int8            *Enable;
      int8            Out;
      int16           RateUp;
      int16           RateDown;
      int16           out_old;
      int8            enable_old;
} RATELIMITER_FIP8;
```

Implementation: FiP16

Name FiP16
ID 97
Revision 1.0
C filename RateLimiter_FiP16.c
H filename RateLimiter_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
out_old	Output value from last cycle in int32 format
enable_old	Enable value from last cycle

Data Structure:

```
typedef struct {
```

```
    uint16          ID;
    int16           *In;
    int16           *Init;
    int8            *Enable;
    int16           Out;
    int32           RateUp;
    int32           RateDown;
    int32           out_old;
    int8            enable_old;
} RATELIMITER_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	98
Revision	1.0
C filename	RateLimiter_FiP32.c
H filename	RateLimiter_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
enable_old	Enable value from last cycle

Data Structure:

```
typedef struct {
    uint16          ID;
    int32           *In;
    int32           *Init;
    int8            *Enable;
    int32           Out;
    int32           RateUp;
    int32           RateDown;
    int8            enable_old;
} RATELIMITER_FIP32;
```

Implementation: Float32

Name	Float32
ID	99
Revision	0.1
C filename	RateLimiter_Float32.c
H filename	RateLimiter_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
enable_old	Enable value from last cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
      float32         *Init;
      int8            *Enable;
      float32         Out;
      float32         RateUp;
      float32         RateDown;
      int8            enable_old;
} RATELIMITER_FLOAT32;
```

Implementation: Float64

Name Float64
ID 100
Revision 0.1
C filename RateLimiter_Float64.c
H filename RateLimiter_Float64.h

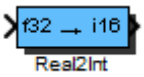
64 Bit Floating Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
enable_old	Enable value from last cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In;
      float64         *Init;
      int8            *Enable;
      float64         Out;
      float64         RateUp;
      float64         RateDown;
      int8            enable_old;
} RATELIMITER_FLOAT64;
```

Block: Real2Int



Inports	
In	Real input

Outports	
Out	Integer output

Mask Parameters	
Scale	Scaling factor from real to integer

Description:

Conversion block from real (floating point) datatypes to integer (fixed point) datatypes.
Out = In / Scale

Implementations:

Float32_FiP8	32 Floating Point to 8 Bit Fixed Point Implementation
Float32_FiP16	32 Floating Point to 16 Bit Fixed Point Implementation
Float32_FiP32	32 Floating Point to 32 Bit Fixed Point Implementation
Float64_FiP8	64 Floating Point to 8 Bit Fixed Point Implementation
Float64_FiP16	64 Floating Point to 16 Bit Fixed Point Implementation
Float64_FiP32	64 Floating Point to 32 Bit Fixed Point Implementation

Implementation: Float32_FiP8

Name	Float32_FiP8
ID	208
Revision	0.1
C filename	Real2Int_Float32_FiP8.c
H filename	Real2Int_Float32_FiP8.h

32 Floating Point to 8 Bit Fixed Point Implementation

Controller Parameters	
scale	Scaling factor

Data Structure:

```
typedef struct {
    uint16        ID;
```

```
    float32         *In;
    int8            Out;
    float32         scale;
} REAL2INT_FLOAT32_FIP8;
```

Implementation: Float32_FiP16

Name	Float32_FiP16
ID	209
Revision	0.1
C filename	Real2Int_Float32_FiP16.c
H filename	Real2Int_Float32_FiP16.h

32 Floating Point to 16 Bit Fixed Point Implementation

Controller Parameters	
scale	Scaling factor

Data Structure:

```
typedef struct {
    uint16          ID;
    float32         *In;
    int16           Out;
    float32         scale;
} REAL2INT_FLOAT32_FIP16;
```

Implementation: Float32_FiP32

Name	Float32_FiP32
ID	210
Revision	0.1
C filename	Real2Int_Float32_FiP32.c
H filename	Real2Int_Float32_FiP32.h

32 Floating Point to 32 Bit Fixed Point Implementation

Controller Parameters	
scale	Scaling factor

Data Structure:

```
typedef struct {
    uint16          ID;
    float32         *In;
    int32           Out;
    float32         scale;
} REAL2INT_FLOAT32_FIP32;
```

Implementation: Float64_FiP8

Name	Float64_FiP8
ID	211
Revision	0.1
C filename	Real2Int_Float64_FiP8.c
H filename	Real2Int_Float64_FiP8.h

64 Floating Point to 8 Bit Fixed Point Implementation

Controller Parameters	
scale	Scaling factor

Data Structure:

```
typedef struct {
    uint16          ID;
    float64         *In;
    int8            Out;
    float64         scale;
} REAL2INT_FLOAT64_FIP8;
```

Implementation: Float64_FiP16

Name	Float64_FiP16
ID	212
Revision	0.1
C filename	Real2Int_Float64_FiP16.c
H filename	Real2Int_Float64_FiP16.h

64 Floating Point to 16 Bit Fixed Point Implementation

Controller Parameters	
scale	Scaling factor

Data Structure:

```
typedef struct {
    uint16          ID;
    float64         *In;
    int16           Out;
    float64         scale;
} REAL2INT_FLOAT64_FIP16;
```

Implementation: Float64_FiP32

Name	Float64_FiP32
ID	213
Revision	0.1
C filename	Real2Int_Float64_FiP32.c
H filename	Real2Int_Float64_FiP32.h

64 Floating Point to 32 Bit Fixed Point Implementation

Controller Parameters	
scale	Scaling factor

Data Structure:

```
typedef struct {
     uint16         ID;
     float64        *In;
     int32          Out;
     float64        scale;
} REAL2INT_FLOAT64_FIP32;
```

Block: Saturation

Inports	
In	Input

Outports	
Out	Limited output

Mask Parameters	
max	Upper Limit
min	Lower Limit

Description:

Saturation of output to adjustable upper and lower limit.

If the entry for *Upper Limit* is lower than the entry for *Lower Limit* then the limits will be swapped internally.

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

Name	FiP8
ID	80
Revision	1.0
C filename	Saturation_FiP8.c
H filename	Saturation_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
max	Upper limit
min	Lower limit

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            Out;
      int8            max;
      int8            min;
} SATURATION_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	81
Revision	1.0
C filename	Saturation_FiP16.c
H filename	Saturation_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
max	Upper limit
min	Lower limit

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In;
      int16           Out;
      int16           max;
      int16           min;
} SATURATION_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	82
Revision	1.0
C filename	Saturation_FiP32.c
H filename	Saturation_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
max	Upper limit
min	Lower limit

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In;
      int32           Out;
      int32           max;
      int32           min;
} SATURATION_FIP32;
```

Implementation: Float32

Name	Float32
ID	83
Revision	0.1
C filename	Saturation_Float32.c
H filename	Saturation_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
max	Upper limit
min	Lower limit

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
      float32         Out;
      float32         max;
      float32         min;
} SATURATION_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	84
Revision	0.1
C filename	Saturation_Float64.c
H filename	Saturation_Float64.h

64 Bit Floating Point Implementation

Controller Parameters	
max	Upper limit
min	Lower limit

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In;
      float64         Out;
      float64         max;
      float64         min;
} SATURATION_FLOAT64;
```

Block: SaveSignal

Inports	
In	Input signal to be saved

Description:

Makes the incoming signal accessible for reading with parameter numbers.

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

Name	FiP8
ID	320
Revision	0.1
C filename	SaveSignal_FiP8.c
H filename	SaveSignal_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int8            *In;
} SAVESIGNAL_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	321
Revision	0.1
C filename	SaveSignal_FiP16.c
H filename	SaveSignal_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int16           *In;
} SAVESIGNAL_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	322
Revision	0.1
C filename	SaveSignal_FiP32.c
H filename	SaveSignal_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int32           *In;
} SAVESIGNAL_FIP32;
```

Implementation: Float32

Name	Float32
ID	323
Revision	0.1
C filename	SaveSignal_Float32.c
H filename	SaveSignal_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     float32         *In;
} SAVESIGNAL_FLOAT32;
```

Implementation: Float64

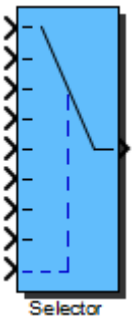
Name	Float64
ID	324
Revision	0.1
C filename	SaveSignal_Float64.c
H filename	SaveSignal_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     float64         *In;
} SAVESIGNAL_FLOAT64;
```

Block: Selector



Inports	
In0	Input #0
In1	Input #1
In2	Input #2
In3	Input #3
In4	Input #4
In5	Input #5
In6	Input #6
In7	Input #7
Select	Input select

Outports	
Out	Selected input signal

Description:

Passing through of input signal selected by the select inport:
Select = 0 (DSP): Out = In0
Select = 1 (DSP): Out = In1
...
Select = 7 (DSP): Out = In7

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

Name	FiP8
ID	400
Revision	1.0
C filename	Selector_FiP8.c
H filename	Selector_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In0;
      int8            *In1;
      int8            *In2;
      int8            *In3;
      int8            *In4;
      int8            *In5;
      int8            *In6;
      int8            *In7;
      int8            *Select;
      int8            Out;
} SELECTOR_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	401
Revision	1.0
C filename	Selector_FiP16.c
H filename	Selector_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In0;
      int16           *In1;
      int16           *In2;
      int16           *In3;
      int16           *In4;
      int16           *In5;
      int16           *In6;
      int16           *In7;
      int8            *Select;
      int16           Out;
} SELECTOR_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	402
Revision	1.0
C filename	Selector_FiP32.c
H filename	Selector_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In0;
      int32           *In1;
      int32           *In2;
      int32           *In3;
      int32           *In4;
      int32           *In5;
      int32           *In6;
      int32           *In7;
      int8            *Select;
      int32           Out;
} SELECTOR_FIP32;
```

Implementation: Float32

Name	Float32
ID	403
Revision	1.0
C filename	Selector_Float32.c
H filename	Selector_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In0;
      float32         *In1;
      float32         *In2;
      float32         *In3;
      float32         *In4;
      float32         *In5;
      float32         *In6;
      float32         *In7;
      int8            *Select;
      float32         Out;
} SELECTOR_FLOAT32;
```

Implementation: Float64

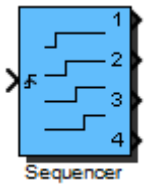
Name	Float64
ID	404
Revision	1.0
C filename	Selector_Float64.c
H filename	Selector_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     float64         *In0;
     float64         *In1;
     float64         *In2;
     float64         *In3;
     float64         *In4;
     float64         *In5;
     float64         *In6;
     float64         *In7;
     int8            *Select;
     float64         Out;
} SELECTOR_FLOAT64;
```

Block: Sequencer



Inports	
Start	Start signal. Rising flank triggers sequence

Outports	
Out1	Output #1
Out2	Output #2
Out3	Output #3
Out4	Output #4

Mask Parameters	
Delay1	Time delay for output 1
Delay2	Time delay for output 2
Delay3	Time delay for output 3
Delay4	Time delay for output 4
ts_fact	Multiplication factor of base sampling time (in integer format)

Description:

Generation of time delayed (enable) sequence.

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

Name	FiP8
ID	448
Revision	1.0
C filename	Sequencer_FiP8.c
H filename	Sequencer_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
delay1	Time delay for output 1
delay2	Time delay for output 2
delay3	Time delay for output 3
delay4	Time delay for output 4
cnt	Timer value
start_old	Start value from previous cycle

Data Structure:

```
typedef struct {
     uint16          ID;
     int8           *Start;
     int8            Out1;
     int8            Out2;
     int8            Out3;
     int8            Out4;
     uint16          delay1;
     uint16          delay2;
     uint16          delay3;
     uint16          delay4;
     uint16          cnt;
     int8            start_old;
} SEQUENCER_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	449
Revision	1.0
C filename	Sequencer_FiP16.c
H filename	Sequencer_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
delay1	Time delay for output 1
delay2	Time delay for output 2
delay3	Time delay for output 3
delay4	Time delay for output 4
cnt	Timer value
start_old	Start value from previous cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *Start;
      int16           Out1;
      int16           Out2;
      int16           Out3;
      int16           Out4;
      uint16          delay1;
      uint16          delay2;
      uint16          delay3;
      uint16          delay4;
      uint16          cnt;
      int16           start_old;
} SEQUENCER_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	450
Revision	1.0
C filename	Sequencer_FiP32.c
H filename	Sequencer_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
delay1	Time delay for output 1
delay2	Time delay for output 2
delay3	Time delay for output 3
delay4	Time delay for output 4
cnt	Timer value
start_old	Start value from previous cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *Start;
```

```
    int32           Out1;
    int32           Out2;
    int32           Out3;
    int32           Out4;
    uint16          delay1;
    uint16          delay2;
    uint16          delay3;
    uint16          delay4;
    uint16          cnt;
    int32           start_old;
} SEQUENCER_FIP32;
```

Implementation: Float32

Name	Float32
ID	451
Revision	0.1
C filename	Sequencer_Float32.c
H filename	Sequencer_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
delay1	Time delay for output 1
delay2	Time delay for output 2
delay3	Time delay for output 3
delay4	Time delay for output 4
cnt	Timer value
start_old	Start value from previous cycle

Data Structure:

```
typedef struct {
    uint16          ID;
    float32         *Start;
    float32         Out1;
    float32         Out2;
    float32         Out3;
    float32         Out4;
    uint16          delay1;
    uint16          delay2;
    uint16          delay3;
    uint16          delay4;
    uint16          cnt;
    float32         start_old;
} SEQUENCER_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	452
Revision	0.1
C filename	Sequencer_Float64.c
H filename	Sequencer_Float64.h

64 Bit Floating Point Implementation

Controller Parameters	
delay1	Time delay for output 1
delay2	Time delay for output 2
delay3	Time delay for output 3
delay4	Time delay for output 4
cnt	Timer value
start_old	Start value from previous cycle

Data Structure:

```
typedef struct {
     uint16          ID;
     float64         *Start;
     float64         Out1;
     float64         Out2;
     float64         Out3;
     float64         Out4;
     uint16          delay1;
     uint16          delay2;
     uint16          delay3;
     uint16          delay4;
     uint16          cnt;
     float64         start_old;
} SEQUENCER_FLOAT64;
```

Block: Sin2Limiter

Inports	
In	

Outports	
Out	

Mask Parameters	
Tr	Rising time in seconds. Slew rate will be 1/Tr
Tf	Falling time in seconds. Slew rate will be 1/Tf
ts_fact	Multiplication factor of base sampling time (in integer format)

Description:

Limitation of rising and falling rate with sin^2 characteristic.
Note: A running limitation process can not be interrupted!

Rising and falling time refer to a step from 0 to 1. Entries for *Tr: Rising time* and *Tf: Falling time* smaller than the actual sample time will be limited to the sample time internally.

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

Name	FiP8
ID	112
Revision	0.2
C filename	Sin2Limiter_FiP8.c
H filename	Sin2Limiter_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
Scaled_RateUp	To step height scaled rising time parameter
Scaled_RateDown	To step height scaled falling time parameter
Out_end	Desired target value
Level	Current level of internal ramp from 1 to 0
Step_Height	Active step height
State	Current state of limitation

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            Out;
      int16           RateUp;
      int16           RateDown;
      int16           Scaled_RateUp;
      int16           Scaled_RateDown;
      int8            Out_end;
      uint16          Level;
      int16           Step_Height;
      int8            State;
} SIN2LIMITER_FIP8;
```

Implementation: FiP16

Name FiP16
ID 113
Revision 0.2
C filename Sin2Limiter_FiP16.c
H filename Sin2Limiter_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
Scaled_RateUp	To step height scaled rising time parameter
Scaled_RateDown	To step height scaled rising time parameter
Out_end	Desired target value
Level	Current level of internal ramp from 1 to 0
Step_Height	Active step height
State	Current state of limitation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In;
      int16           Out;
      int32           RateUp;
      int32           RateDown;
      int32           Scaled_RateUp;
      int32           Scaled_RateDown;
      int16           Out_end;
      uint32          Level;
      int32           Step_Height;
      int8            State;
} SIN2LIMITER_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	114
Revision	0.2
C filename	Sin2Limiter_FiP32.c
H filename	Sin2Limiter_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
Scaled_RateUp	To step height scaled rising time parameter
Scaled_RateDown	To step height scaled rising time parameter
Out_end	Desired target value
Level	Current level of internal ramp from 1 to 0
Step_Height	Active step height
State	Current state of limitation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In;
      int32           Out;
      int32           RateUp;
      int32           RateDown;
      int32           Scaled_RateUp;
      int32           Scaled_RateDown;
      int32           Out_end;
      uint32          Level;
      int32           Step_Height;
      int8            State;
} SIN2LIMITER_FIP32;
```

Implementation: Float32

Name	Float32
ID	115
Revision	0.1
C filename	Sin2Limiter_Float32.c
H filename	Sin2Limiter_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
Scaled_RateUp	To step height scaled rising time parameter
Scaled_RateDown	To step height scaled falling time parameter
Out_end	Desired target value
Level	Current level of internal ramp from pi/2 to 0
Step_Height	Active step height
State	Current state of limitation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
      float32         Out;
      float32         RateUp;
      float32         RateDown;
      float32         Scaled_RateUp;
      float32         Scaled_RateDown;
      float32         Out_end;
      float32         Level;
      float32         Step_Height;
      int8            State;
} SIN2LIMITER_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	116
Revision	0.1
C filename	Sin2Limiter_Float64.c
H filename	Sin2Limiter_Float64.h

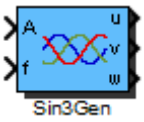
64 Bit Floating Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
Scaled_RateUp	To step height scaled rising time parameter
Scaled_RateDown	To step height scaled falling time parameter
Out_end	Desired target value
Level	Current level of internal ramp from pi/2 to 0
Step_Height	Active step height
State	Current state of limitation

Data Structure:

```
typedef struct {
     uint16          ID;
     float64         *In;
     float64         Out;
     float64         RateUp;
     float64         RateDown;
     float64         Scaled_RateUp;
     float64         Scaled_RateDown;
     float64         Out_end;
     float64         Level;
     float64         Step_Height;
     int8            State;
} SIN2LIMITER_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	
fmax	Maximum Frequency in Hz
Offset	Offset
ts_fact	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:

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

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):

$$\begin{aligned} u_k &= A_k \cdot \sin\left(2\pi f_k \cdot k T_S\right) + A_{Offset} \\ v_k &= A_k \cdot \sin\left(2\pi f_k \cdot k T_S - \frac{2\pi}{3}\right) + A_{Offset} \\ w_k &= A_k \cdot \sin\left(2\pi f_k \cdot k T_S + \frac{2\pi}{3}\right) + A_{Offset} \end{aligned}$$

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

Name	FiP8
ID	432
Revision	1.0
C filename	Sin3Gen_FiP8.c
H filename	Sin3Gen_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
delta_phi	Angle increment
offset	Amplitude offset
phi	Current angle

Data Structure:

```
typedef struct {
    uint16          ID;
    int8            *A;
    int8            *f;
    int8            u;
    int8            v;
    int8            w;
    int8            delta_phi;
    int8            offset;
    int8            phi;
} SIN3GEN_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	433
Revision	1.0
C filename	Sin3Gen_FiP16.c
H filename	Sin3Gen_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
delta_phi	Angle increment
offset	Amplitude offset
phi	Current angle

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *A;
      int16           *f;
      int16           u;
      int16           v;
      int16           w;
      int16           delta_phi;
      int16           offset;
      int16           phi;
} SIN3GEN_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	434
Revision	1.0
C filename	Sin3Gen_FiP32.c
H filename	Sin3Gen_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
delta_phi	Angle increment
offset	Amplitude offset
phi	Current angle

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *A;
      int32           *f;
      int32           u;
      int32           v;
      int32           w;
      int32           delta_phi;
      int32           offset;
      int32           phi;
} SIN3GEN_FIP32;
```

Implementation: Float32

Name	Float32
ID	435
Revision	0.1
C filename	Sin3Gen_Float32.c
H filename	Sin3Gen_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
delta_phi	Angle increment
offset	Amplitude offset
phi	Current angle

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *A;
      float32         *f;
      float32         u;
      float32         v;
      float32         w;
      float32         delta_phi;
      float32         offset;
      float32         phi;
} SIN3GEN_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	436
Revision	0.1
C filename	Sin3Gen_Float64.c
H filename	Sin3Gen_Float64.h

64 Bit Floating Point Implementation

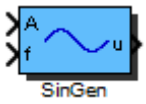
Controller Parameters	
delta_phi	Angle increment
offset	Amplitude offset
phi	Current angle

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *A;
      float64         *f;
```

```
    float64         u;
    float64         v;
    float64         w;
    float64         delta_phi;
    float64         offset;
    float64         phi;
} SIN3GEN_FLOAT64;
```

Block: SinGen



Inports	
A	Amplitude
f	Frequency

Outports	
u	Sine wave output

Mask Parameters	
fmax	Maximum Frequency in Hz
Offset	Offset
Phase	Phase [-Pi..Pi]
ts_fact	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:

$$u_k = A_k \cdot \sin\left(2f_k \cdot f_{max} \cdot kT_S + \phi_{Phase}\right) + A_{Offset}$$

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):

$$u_k = A_k \cdot \sin\left(2\pi f_k \cdot kT_S + \phi_{Phase}\right) + A_{Offset}$$

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

Name FiP8
ID 416
Revision 1.0
C filename SinGen_FiP8.c
H filename SinGen_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
delta_phi	Angle increment
phase	Angle offset
offset	Amplitude offset
phi	Current angle

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *A;
      int8            *f;
      int8            u;
      int8            delta_phi;
      int8            phase;
      int8            offset;
      int8            phi;
} SINGEN_FIP8;
```

Implementation: FiP16

Name FiP16
ID 417
Revision 1.0
C filename SinGen_FiP16.c
H filename SinGen_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
delta_phi	Angle increment
phase	Angle offset
offset	Amplitude offset
phi	Current angle

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *A;
      int16           *f;
```

```
    int16           u;
    int16           delta_phi;
    int16           phase;
    int16           offset;
    int16           phi;
} SINGEN_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	418
Revision	1.0
C filename	SinGen_FiP32.c
H filename	SinGen_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
delta_phi	Angle increment
phase	Angle offset
offset	Amplitude offset
phi	Current angle

Data Structure:

```
typedef struct {
    uint16          ID;
    int32           *A;
    int32           *f;
    int32           u;
    int32           delta_phi;
    int32           phase;
    int32           offset;
    int32           phi;
} SINGEN_FIP32;
```

Implementation: Float32

Name	Float32
ID	419
Revision	0.1
C filename	SinGen_Float32.c
H filename	SinGen_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
delta_phi	Angle increment
phase	Angle offset
offset	Amplitude offset
phi	Current angle

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *A;
      float32         *f;
      float32         u;
      float32         delta_phi;
      float32         phase;
      float32         offset;
      float32         phi;
} SINGEN_FLOAT32;
```

Implementation: Float64

Name Float64
ID 420
Revision 0.1
C filename SinGen_Float64.c
H filename SinGen_Float64.h

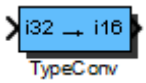
64 Bit Floating Point Implementation

Controller Parameters	
delta_phi	Angle increment
phase	Angle offset
offset	Amplitude offset
phi	Current angle

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *A;
      float64         *f;
      float64         u;
      float64         delta_phi;
      float64         phase;
      float64         offset;
      float64         phi;
} SINGEN_FLOAT64;
```

Block: TypeConv



Inports	
In	

Outports	
Out	

Description:

Data Type Conversion

Implementations:

FiP8_16	8 to 16 Bit Fixed Point Implementation
FiP8_32	8 to 32 Bit Fixed Point Implementation
FiP16_8	16 to 8 Bit Fixed Point Implementation
FiP16_32	16 to 32 Bit Fixed Point Implementation
FiP32_8	32 to 8 Bit Fixed Point Implementation
FiP32_16	32 to 16 Bit Fixed Point Implementation

Implementation: FiP8_16

Name	FiP8_16
ID	176
Revision	0.1
C filename	TypeConv_FiP8_16.c
H filename	TypeConv_FiP8_16.h

8 to 16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int8            *In;
     int16           Out;
} TYPECONV_FIP8_16;
```

Implementation: FiP8_32

Name	FiP8_32
ID	177
Revision	0.1
C filename	TypeConv_FiP8_32.c
H filename	TypeConv_FiP8_32.h

8 to 32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
    uint16          ID;
    int8            *In;
    int32           Out;
} TYPECONV_FIP8_32;
```

Implementation: FiP16_8

Name	FiP16_8
ID	178
Revision	0.1
C filename	TypeConv_FiP16_8.c
H filename	TypeConv_FiP16_8.h

16 to 8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
    uint16          ID;
    int16           *In;
    int8            Out;
} TYPECONV_FIP16_8;
```

Implementation: FiP16_32

Name	FiP16_32
ID	179
Revision	0.1
C filename	TypeConv_FiP16_32.c
H filename	TypeConv_FiP16_32.h

16 to 32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
    uint16          ID;
    int16           *In;
```

```
    int32           Out;
} TYPECONV_FIP16_32;
```

Implementation: FiP32_8

Name	FiP32_8
ID	180
Revision	0.1
C filename	TypeConv_FiP32_8.c
H filename	TypeConv_FiP32_8.h

32 to 8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
    uint16          ID;
    int32           *In;
    int8            Out;
} TYPECONV_FIP32_8;
```

Implementation: FiP32_16

Name	FiP32_16
ID	181
Revision	0.1
C filename	TypeConv_FiP32_16.c
H filename	TypeConv_FiP32_16.h

32 to 16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
    uint16          ID;
    int32           *In;
    int16           Out;
} TYPECONV_FIP32_16;
```

Block: uConstant

Outports	
Out	Constant output

Mask Parameters	
Value	Constant factor

Description:

Constant value.

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

Name	FiP8
ID	64
Revision	0.2
C filename	uConstant_FiP8.c
H filename	uConstant_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
K	Constant factor

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            Out;
      int8            K;
} UCONSTANT_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	65
Revision	0.2
C filename	uConstant_FiP16.c
H filename	uConstant_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
K	Constant factor

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           Out;
      int16           K;
} UCONSTANT_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	66
Revision	0.2
C filename	uConstant_FiP32.c
H filename	uConstant_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
K	Constant factor

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           Out;
      int32           K;
} UCONSTANT_FIP32;
```

Implementation: Float32

Name	Float32
ID	67
Revision	0.1
C filename	uConstant_Float32.c
H filename	uConstant_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
K	Constant factor

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         Out;
      float32         K;
} UCONSTANT_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	68
Revision	0.1
C filename	uConstant_Float64.c
H filename	uConstant_Float64.h

64 Bit Floating Point Implementation

Controller Parameters	
K	Constant factor

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         Out;
      float64         K;
} UCONSTANT_FLOAT64;
```

Block: uGain

Inports	
In	Input

Outports	
Out	Amplified input

Mask Parameters	
Gain	Gain factor in floating point format

Description:

Amplification of input by gain factor with output wrapping.

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	32 Bit Floating Point Implementation

Implementation: FiP8

Name	FiP8
ID	32
Revision	1.0
C filename	uGain_FiP8.c
H filename	uGain_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
V	Gain factor
sfr	Shift factor for gain value

Data Structure:

```
typedef struct {
    uint16        ID;
    int8          *In;
```

```
    int8            Out;
    int8            V;
    int8            sfr;
} UGAIN_FIP8;
```

Implementation: FiP16

Name FiP16
ID 33
Revision 1.0
C filename uGain_FiP16.c
H filename uGain_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
V	Gain factor
sfr	Shift factor for gain value

Data Structure:

```
typedef struct {
    uint16          ID;
    int16           *In;
    int16           Out;
    int16           V;
    int8            sfr;
} UGAIN_FIP16;
```

Implementation: FiP32

Name FiP32
ID 34
Revision 1.0
C filename uGain_FiP32.c
H filename uGain_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
V	Gain factor
sfr	Shift factor for gain value

Data Structure:

```
typedef struct {
    uint16          ID;
    int32           *In;
```

```
    int32           Out;
    int32           V;
    int8            sfr;
} UGAIN_FIP32;
```

Implementation: Float32

Name	Float32
ID	35
Revision	0.1
C filename	uGain_Float32.c
H filename	uGain_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
V	Gain factor

Data Structure:

```
typedef struct {
    uint16          ID;
    float32         *In;
    float32         Out;
    float32         V;
} UGAIN_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	36
Revision	0.1
C filename	uGain_Float64.c
H filename	uGain_Float64.h

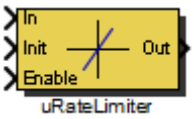
32 Bit Floating Point Implementation

Controller Parameters	
V	Gain factor

Data Structure:

```
typedef struct {
    uint16          ID;
    float64         *In;
    float64         Out;
    float64         V;
} UGAIN_FLOAT64;
```

Block: uRateLimiter



Inports	
In	
Init	Value which is loaded at rising flanke of enable signal
Enable	Enable == 0: Deactivation of block; Out is set to In. Enable != 0: Activation of block; Out is rate limited. Enable 0->1: Preloading of output; Out is set to value of Init input

Outports	
Out	

Mask Parameters	
Tr	Rising time in seconds. Slew rate will be 1/Tr
Tf	Falling time in seconds. Slew rate will be 1/Tf
ts_fact	Multiplication factor of base sampling time (in integer format)

Description:

Limitation of rising and falling rate.
Function of Enable:
0: rate limiting disabled, signal is passed through
1: rate limiting enabled, signal is rate limited
0->1: preload of output with value from init input

Rising and falling time refer to a step from 0 to 1. Entries for *Tr: Rising time* and *Tf: Falling time* smaller than the actual sample time will be limited to the sample time internally.
The 16- and 32-Bit fixed point implementations are based on an internal 32-Bit wide slew-rate variable while the 8-Bit fixed point implementation uses a 16-Bit wide slew-rate variable.

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

Name FiP8
ID 288
Revision 1.0
C filename uRateLimiter_FiP8.c
H filename uRateLimiter_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
out_old	Output value from last cycle in int16 format
enable_old	Enable value from last cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            *Init;
      int8            *Enable;
      int8            Out;
      int16           RateUp;
      int16           RateDown;
      int16           out_old;
      int8            enable_old;
} URATELIMITER_FIP8;
```

Implementation: FiP16

Name FiP16
ID 289
Revision 1.0
C filename uRateLimiter_FiP16.c
H filename uRateLimiter_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
out_old	Output value from last cycle in int32 format
enable_old	Enable value from last cycle

Data Structure:

```
typedef struct {
```

```
    uint16          ID;
    int16           *In;
    int16           *Init;
    int8            *Enable;
    int16           Out;
    int32           RateUp;
    int32           RateDown;
    int32           out_old;
    int8            enable_old;
} URATELIMITER_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	290
Revision	1.0
C filename	uRateLimiter_FiP32.c
H filename	uRateLimiter_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
enable_old	Enable value from last cycle

Data Structure:

```
typedef struct {
    uint16          ID;
    int32           *In;
    int32           *Init;
    int8            *Enable;
    int32           Out;
    int32           RateUp;
    int32           RateDown;
    int8            enable_old;
} URATELIMITER_FIP32;
```

Implementation: Float32

Name	Float32
ID	291
Revision	0.1
C filename	uRateLimiter_Float32.c
H filename	uRateLimiter_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
enable_old	Enable value from last cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
      float32         *Init;
      int8            *Enable;
      float32         Out;
      float32         RateUp;
      float32         RateDown;
      int8            enable_old;
} URATELIMITER_FLOAT32;
```

Implementation: Float64

Name Float64
ID 292
Revision 0.1
C filename uRateLimiter_Float64.c
H filename uRateLimiter_Float64.h

64 Bit Floating Point Implementation

Controller Parameters	
RateUp	Rising time parameter
RateDown	Falling time parameter
enable_old	Enable value from last cycle

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In;
      float64         *Init;
      int8            *Enable;
      float64         Out;
      float64         RateUp;
      float64         RateDown;
      int8            enable_old;
} URATELIMITER_FLOAT64;
```

Block: uSaveSignal

Inports	
In	Input signal to be saved

Description:

Makes the incoming signal accessible for reading with parameter numbers.

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

Name	FiP8
ID	336
Revision	0.1
C filename	uSaveSignal_FiP8.c
H filename	uSaveSignal_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16         ID;
     uint8          *In;
} USAVESIGNAL_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	337
Revision	0.1
C filename	uSaveSignal_FiP16.c
H filename	uSaveSignal_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     uint16          *In;
} USAVESIGNAL_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	338
Revision	0.1
C filename	uSaveSignal_FiP32.c
H filename	uSaveSignal_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     uint32          *In;
} USAVESIGNAL_FIP32;
```

Implementation: Float32

Name	Float32
ID	339
Revision	0.1
C filename	uSaveSignal_Float32.c
H filename	uSaveSignal_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     float32         *In;
} USAVESIGNAL_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	340
Revision	0.1
C filename	uSaveSignal_Float64.c
H filename	uSaveSignal_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     float64         *In;
} USAVESIGNAL_FLOAT64;
```

Block: Xor

Inports	
In1	
In2	

Outports	
Out	

Description:

Logical XOR block.

Implementations:

FiP8	8 Bit Fixed Point Implementation
FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation

Implementation: FiP8

Name	FiP8
ID	272
Revision	0.1
C filename	Xor_FiP8.c
H filename	Xor_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int8            *In1;
     int8            *In2;
     int8            Out;
} XOR_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	273
Revision	0.1
C filename	Xor_FiP16.c
H filename	Xor_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In1;
      int16           *In2;
      int16           Out;
} XOR_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	274
Revision	0.1
C filename	Xor_FiP32.c
H filename	Xor_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In1;
      int32           *In2;
      int32           Out;
} XOR_FIP32;
```

19 Math

Block: Abs

Inports	
In	Input u

Outports	
Out	Absolute value of u

Description:

Calculation of absolute value of input.

Calculation:

$$Out = |In|$$

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

Name	FiP8
ID	4912
Revision	0.1
C filename	Abs_FiP8.c
H filename	Abs_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16         ID;
     int8           *In;
     int8           Out;
} ABS_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	4913
Revision	0.1
C filename	Abs_FiP16.c
H filename	Abs_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int16           *In;
     int16           Out;
} ABS_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	4914
Revision	0.1
C filename	Abs_FiP32.c
H filename	Abs_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int32           *In;
     int32           Out;
} ABS_FIP32;
```

Implementation: Float32

Name	Float32
ID	4915
Revision	0.1
C filename	Abs_Float32.c
H filename	Abs_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     float32         *In;
```

```
    float32         Out;
} ABS_FLOAT32;
```

Implementation: Float64

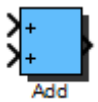
Name	Float64
ID	4916
Revision	0.1
C filename	Abs_Float64.c
H filename	Abs_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
    uint16          ID;
    float64         *In;
    float64         Out;
} ABS_FLOAT64;
```

Block: Add



Inports	
In1	Addend 1
In2	Addend 2

Outports	
Out	Sum

Description:

Addition of input 1 and input 2.

Calculation:

$$Out = In_1 + In_2$$

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

Name	FiP8
ID	4960
Revision	0.3
C filename	Add_FiP8.c
H filename	Add_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int8            *In1;
     int8            *In2;
     int8            Out;
} ADD_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	4961
Revision	0.3
C filename	Add_FiP16.c
H filename	Add_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int16           *In1;
     int16           *In2;
     int16           Out;
} ADD_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	4962
Revision	0.3
C filename	Add_FiP32.c
H filename	Add_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int32           *In1;
     int32           *In2;
     int32           Out;
} ADD_FIP32;
```

Implementation: Float32

Name	Float32
ID	4963
Revision	0.1
C filename	Add_Float32.c
H filename	Add_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In1;
      float32         *In2;
      float32         Out;
} ADD_FLOAT32;
```

Implementation: Float64

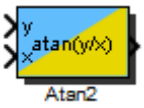
Name	Float64
ID	4964
Revision	0.1
C filename	Add_Float64.c
H filename	Add_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In1;
      float64         *In2;
      float64         Out;
} ADD_FLOAT64;
```

Block: Atan2



Inports	
y	
x	

Outports	
Out	Result of atan2(y/x)

Description:

Computation of the angle between the inputs x and y.

Calculation:

$$Out = \begin{cases} arctan\left(\frac{y}{x}\right) & x > 0 \\ arctan\left(\frac{y}{x}\right) + \pi & x < 0, y \geq 0 \\ arctan\left(\frac{y}{x}\right) - \pi & x < 0, y < 0 \\ +\frac{\pi}{2} & x = 0, y > 0 \\ -\frac{\pi}{2} & x = 0, y < 0 \\ 0 & x = 0, y = 0 \end{cases}$$

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

Name	FiP8
ID	4880
Revision	1.0
C filename	Atan2_FiP8.c
H filename	Atan2_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *y;
      int8            *x;
      int8            Out;
} ATAN2_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	4881
Revision	1.0
C filename	Atan2_FiP16.c
H filename	Atan2_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *y;
      int16           *x;
      int16           Out;
} ATAN2_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	4882
Revision	1.0
C filename	Atan2_FiP32.c
H filename	Atan2_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *y;
      int32           *x;
      int32           Out;
} ATAN2_FIP32;
```

Implementation: Float32

Name	Float32
ID	4883
Revision	0.1
C filename	Atan2_Float32.c
H filename	Atan2_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *y;
      float32         *x;
      float32         Out;
} ATAN2_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	4884
Revision	0.1
C filename	Atan2_Float64.c
H filename	Atan2_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *y;
      float64         *x;
      float64         Out;
} ATAN2_FLOAT64;
```

Block: Average

Inports	
In	Input value

Outports	
Out	Averaged value

Mask Parameters	
n	Number of points to be averaged over
ts_fact	Multiplication factor of base sampling time (in integer format)

Description:

Calculation of moving average value over n numbers.

Calculation:

$$Out_k = \frac{1}{n} \sum_{i=k-n}^{k} In_i$$

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

Name	FiP8
ID	5024
Revision	1
C filename	Average_FiP8.c
H filename	Average_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
n	Average window size
sfrn	Shift factor for computation of average value sfrn = ld(n)
sum	Temporary sum
count	Index counter
avg	Array with data values

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            Out;
      uint16          n;
      uint8           sfrn;
      int16           sum;
      uint16          count;
      int8            *avg;
} AVERAGE_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	5025
Revision	1
C filename	Average_FiP16.c
H filename	Average_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
n	Average window size
sfrn	Shift factor for computation of average value sfrn = ld(n)
sum	Temporary sum
count	Index counter
avg	Array with data values

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In;
      int16           Out;
      uint16          n;
      uint8           sfrn;
      int32           sum;
      uint16          count;
      int16           *avg;
} AVERAGE_FIP16;
```

Implementation: FiP32

Name FiP32
ID 5026
Revision 1
C filename Average_FiP32.c
H filename Average_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
n	Average window size
sfrn	Shift factor for computation of average value sfrn = ld(n)
sum	Temporary sum
count	Index counter
avg	Array with data values

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In;
      int32           Out;
      uint16          n;
      uint8           sfrn;
      int64           sum;
      uint16          count;
      int32           *avg;
} AVERAGE_FIP32;
```

Implementation: Float32

Name Float32
ID 5027
Revision 0.1
C filename Average_Float32.c
H filename Average_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
n	Average window size
sum	Temporary sum
count	Index counter
avg	Array with data values

Data Structure:

```
typedef struct {
```

```
    uint16          ID;
    float32         *In;
    float32         Out;
    uint16          n;
    float32         sum;
    uint16          count;
    float32         *avg;
} AVERAGE_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	5028
Revision	0.1
C filename	Average_Float64.c
H filename	Average_Float64.h

64 Bit Floating Point Implementation

Controller Parameters	
n	Average window size
sum	Temporary sum
count	Index counter
avg	Array with data values

Data Structure:

```
typedef struct {
    uint16          ID;
    float64         *In;
    float64         Out;
    uint16          n;
    float64         sum;
    uint16          count;
    float64         *avg;
} AVERAGE_FLOAT64;
```

Block: Cos

Inports	
In	Input u

Outports	
Out	Result of cos(u)

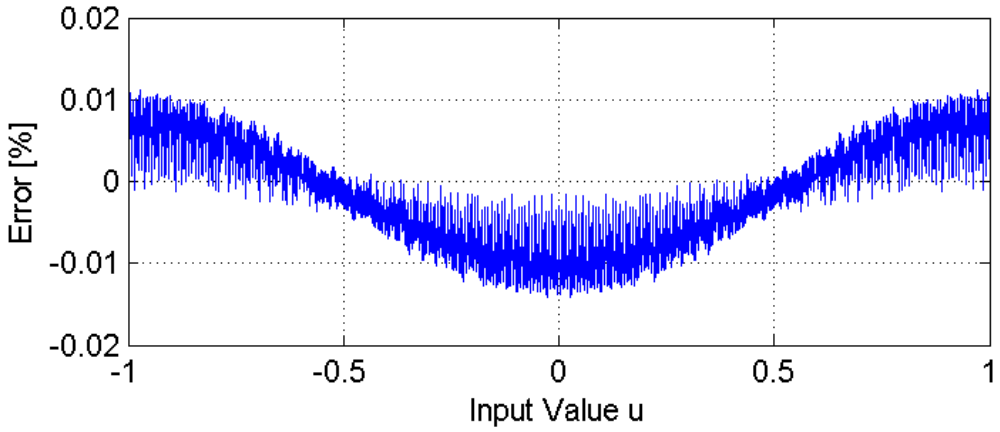
Description:

Cosine computation of input value.

Calculation:

$$Out = cos(In)$$

Error for 16 Bit Fixed Point Implementation:



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

Name	FiP8
ID	4864
Revision	0.1
C filename	Cos_FiP8.c
H filename	Cos_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            Out;
} COS_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	4865
Revision	0.1
C filename	Cos_FiP16.c
H filename	Cos_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In;
      int16           Out;
} COS_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	4866
Revision	0.1
C filename	Cos_FiP32.c
H filename	Cos_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In;
      int32           Out;
} COS_FIP32;
```

Implementation: Float32

Name	Float32
ID	4867
Revision	0.1
C filename	Cos_Float32.c
H filename	Cos_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
     uint16         ID;
     float32        *In;
     float32        Out;
} COS_FLOAT32;
```

Implementation: Float64

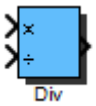
Name	Float64
ID	4868
Revision	0.1
C filename	Cos_Float64.c
H filename	Cos_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
     uint16         ID;
     float64        *In;
     float64        Out;
} COS_FLOAT64;
```

Block: Div



Inports	
Num	Dividend (Numerator)
Den	Divisor (Denominator)

Outports	
Out	Quotient

Description:

Division of input Num by input Den.

Calculation:

$$Out = \begin{cases} 0 & Num = 0, Den = 0 \\ maxVal & Num > 0, Den = 0 \\ minVal & Num < 0, Den = 0 \\ \frac{Num}{Den} & \text{otherwise} \end{cases}$$

Note: *maxVal* and *minVal* refer to the maximum/minimum representable value of the implementation.

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

Name	FiP8
ID	4928
Revision	0.1
C filename	Div_FiP8.c
H filename	Div_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *Num;
      int8            *Den;
      int8            Out;
} DIV_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	4929
Revision	0.1
C filename	Div_FiP16.c
H filename	Div_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *Num;
      int16           *Den;
      int16           Out;
} DIV_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	4930
Revision	0.1
C filename	Div_FiP32.c
H filename	Div_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *Num;
      int32           *Den;
      int32           Out;
} DIV_FIP32;
```

Implementation: Float32

Name	Float32
ID	4931
Revision	0.1
C filename	Div_Float32.c
H filename	Div_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *Num;
      float32         *Den;
      float32         Out;
} DIV_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	4932
Revision	0.1
C filename	Div_Float64.c
H filename	Div_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *Num;
      float64         *Den;
      float64         Out;
} DIV_FLOAT64;
```

Block: Exp

Inports	
In	Input u

Outports	
Out	Result of exp(u)

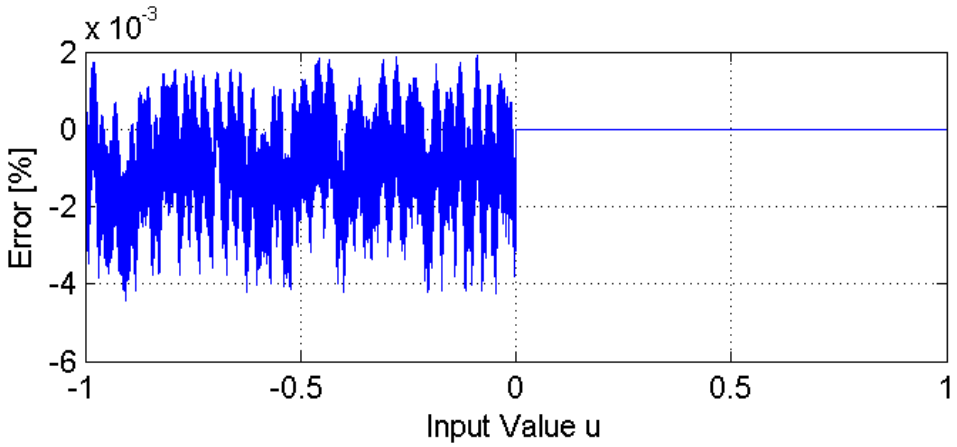
Description:

Computation of the exponential of the input.

Calculation:

$$Out = \begin{cases} e^{In} & In \leq 0 \\ 1 & In > 0 \end{cases}$$

Error for 16 Bit Fixed Point Implementation:



Implementations:

FiP8	8 Bit Fixed Point Implementation
FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation

Implementation: FiP8

Name	FiP8
ID	4848
Revision	0.1
C filename	Exp_FiP8.c
H filename	Exp_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            Out;
} EXP_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	4849
Revision	0.1
C filename	Exp_FiP16.c
H filename	Exp_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In;
      int16           Out;
} EXP_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	4850
Revision	0.1
C filename	Exp_FiP32.c
H filename	Exp_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In;
      int32           Out;
} EXP_FIP32;
```

Block: L2Norm

Inports	
u1	Input u1
u2	Input u2

Outports	
Out	Euclidean norm of u1 and u2

Description:

Calculation of L2-norm (euclidean norm).

Calculation:

$$Out = \|u\| = \sqrt{u_1^2 + u_2^2}$$

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

Name	FiP8
ID	5056
Revision	0.1
C filename	L2Norm_FiP8.c
H filename	L2Norm_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
    uint16          ID;
    int8            *u1;
    int8            *u2;
    int8            Out;
} L2NORM_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	5057
Revision	0.1
C filename	L2Norm_FiP16.c
H filename	L2Norm_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *u1;
      int16           *u2;
      int16           Out;
} L2NORM_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	5058
Revision	0.1
C filename	L2Norm_FiP32.c
H filename	L2Norm_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *u1;
      int32           *u2;
      int32           Out;
} L2NORM_FIP32;
```

Implementation: Float32

Name	Float32
ID	5059
Revision	0.1
C filename	L2Norm_Float32.c
H filename	L2Norm_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     float32         *u1;
     float32         *u2;
     float32         Out;
} L2NORM_FLOAT32;
```

Implementation: Float64

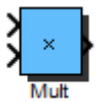
Name	Float64
ID	5060
Revision	0.1
C filename	L2Norm_Float64.c
H filename	L2Norm_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     float64         *u1;
     float32         *u2;
     float64         Out;
} L2NORM_FLOAT64;
```

Block: Mult



Inports	
In1	Multiplicand 1
In2	Multiplicand 2

Outports	
Out	Product

Description:

Multiplication of input 1 with input 2.

Calculation:

$$Out \quad = \quad In_1 \cdot In_2$$

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

Name	FiP8
ID	4944
Revision	0.1
C filename	Mult_FiP8.c
H filename	Mult_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int8            *In1;
     int8            *In2;
     int8            Out;
} MULT_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	4945
Revision	0.1
C filename	Mult_FiP16.c
H filename	Mult_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int16           *In1;
     int16           *In2;
     int16           Out;
} MULT_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	4946
Revision	0.1
C filename	Mult_FiP32.c
H filename	Mult_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int32           *In1;
     int32           *In2;
     int32           Out;
} MULT_FIP32;
```

Implementation: Float32

Name	Float32
ID	4947
Revision	0.1
C filename	Mult_Float32.c
H filename	Mult_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In1;
      float32         *In2;
      float32         Out;
} MULT_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	4948
Revision	0.1
C filename	Mult_Float64.c
H filename	Mult_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In1;
      float64         *In2;
      float64         Out;
} MULT_FLOAT64;
```

Block: Negation

Inports	
In	Input

Outports	
Out	Negated input value

Description:

Negation of input signal.

Calculation:

$$Out \quad = \quad -In$$

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

Name	FiP8
ID	5040
Revision	0.1
C filename	Negation_FiP8.c
H filename	Negation_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
    uint16          ID;
    int8            *In;
    int8            Out;
} NEGATION_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	5041
Revision	0.1
C filename	Negation_FiP16.c
H filename	Negation_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In;
      int16           Out;
} NEGATION_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	5042
Revision	0.1
C filename	Negation_FiP32.c
H filename	Negation_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In;
      int32           Out;
} NEGATION_FIP32;
```

Implementation: Float32

Name	Float32
ID	5043
Revision	0.1
C filename	Negation_Float32.c
H filename	Negation_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
```

```
    float32         Out;
} NEGATION_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	5044
Revision	0.1
C filename	Negation_Float64.c
H filename	Negation_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
    uint16          ID;
    float64         *In;
    float64         Out;
} NEGATION_FLOAT64;
```

Block: Sign

Inports	
In	Input u

Outports	
Out	Value corresponding to sign of u

Description:

Signum function.

Calculation:

$$Out = sign(In) = \begin{cases} +1 & In \geq 0 \\ -1 & In < 0 \end{cases}$$

Implementations:

FiP8	8 Bit Fixed Point Implementation
FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation

Implementation: FiP8

Name	FiP8
ID	4896
Revision	0.1
C filename	Sign_FiP8.c
H filename	Sign_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16         ID;
     int8           *In;
     int8           Out;
} SIGN_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	4897
Revision	0.1
C filename	Sign_FiP16.c
H filename	Sign_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In;
      int16           Out;
} SIGN_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	4898
Revision	0.1
C filename	Sign_FiP32.c
H filename	Sign_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In;
      int32           Out;
} SIGN_FIP32;
```

Block: Sin

Inports	
In	Input u

Outports	
Out	Result of sin(u)

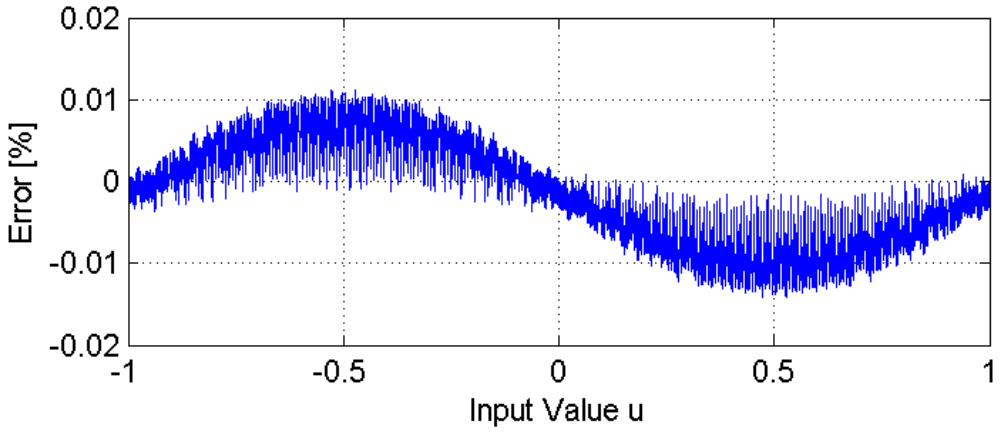
Description:

Sine computation of input value.

Calculation:

$$Out = sin(In)$$

Error for 16 Bit Fixed Point Implementation:



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

Name	FiP8
ID	4832
Revision	0.1
C filename	Sin_FiP8.c
H filename	Sin_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            Out;
} SIN_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	4833
Revision	0.1
C filename	Sin_FiP16.c
H filename	Sin_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In;
      int16           Out;
} SIN_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	4834
Revision	0.1
C filename	Sin_FiP32.c
H filename	Sin_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In;
      int32           Out;
} SIN_FIP32;
```

Implementation: Float32

Name	Float32
ID	4835
Revision	0.1
C filename	Sin_Float32.c
H filename	Sin_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
     uint16         ID;
     float32        *In;
     float32        Out;
} SIN_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	4836
Revision	0.1
C filename	Sin_Float64.c
H filename	Sin_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
     uint16         ID;
     float64        *In;
     float64        Out;
} SIN_FLOAT64;
```

Block: Sqrt

Inports	
In	Input u

Outports	
Out	Result of sqrt(|u|)

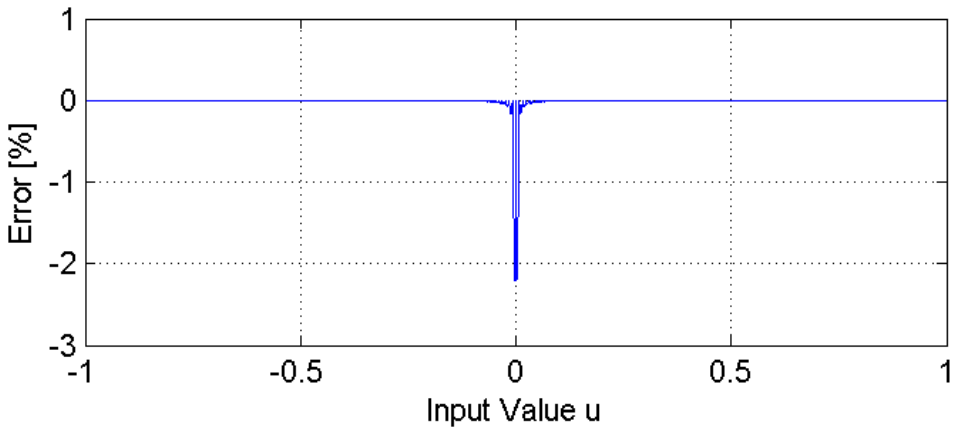
Description:

Square root computation of absolute input value.

Calculation:

$$Out = \sqrt{|In|}$$

Error for 16 Bit Fixed Point Implementation:



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

Name	FiP8
ID	4816
Revision	0.1
C filename	Sqrt_FiP8.c
H filename	Sqrt_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In;
      int8            Out;
} SQRT_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	4817
Revision	0.1
C filename	Sqrt_FiP16.c
H filename	Sqrt_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In;
      int16           Out;
} SQRT_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	4818
Revision	0.1
C filename	Sqrt_FiP32.c
H filename	Sqrt_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In;
      int32           Out;
} SQRT_FIP32;
```

Implementation: Float32

Name	Float32
ID	4819
Revision	0.1
C filename	Sqrt_Float32.c
H filename	Sqrt_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In;
      float32         Out;
} SQRT_FLOAT32;
```

Implementation: Float64

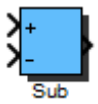
Name	Float64
ID	4820
Revision	0.1
C filename	Sqrt_Float64.c
H filename	Sqrt_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In;
      float64         Out;
} SQRT_FLOAT64;
```

Block: Sub



Inports	
Plus	Minuend
Minus	Subtrahend

Outports	
Out	Difference

Description:

Subtraction of input Minus from input Plus.

Calculation:

$$Out \quad = \quad Plus - Minus$$

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

Name	FiP8
ID	4992
Revision	0.1
C filename	Sub_FiP8.c
H filename	Sub_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
    uint16          ID;
    int8            *Plus;
    int8            *Minus;
    int8            Out;
} SUB_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	4993
Revision	0.1
C filename	Sub_FiP16.c
H filename	Sub_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *Plus;
      int16           *Minus;
      int16           Out;
} SUB_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	4994
Revision	0.1
C filename	Sub_FiP32.c
H filename	Sub_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *Plus;
      int32           *Minus;
      int32           Out;
} SUB_FIP32;
```

Implementation: Float32

Name	Float32
ID	4995
Revision	0.1
C filename	Sub_Float32.c
H filename	Sub_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *Plus;
      float32         *Minus;
      float32         Out;
} SUB_FLOAT32;
```

Implementation: Float64

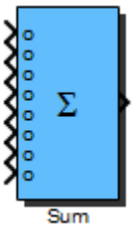
Name	Float64
ID	4996
Revision	0.1
C filename	Sub_Float64.c
H filename	Sub_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *Plus;
      float64         *Minus;
      float64         Out;
} SUB_FLOAT64;
```

Block: Sum



Inports	
In1	Input #1
In2	Input #2
In3	Input #3
In4	Input #4
In5	Input #5
In6	Input #6
In7	Input #7
In8	Input #8

Outports	
Out	Result

Mask Parameters	
In1	Input #1
In2	Input #2
In3	Input #3
In4	Input #4
In5	Input #5
In6	Input #6
In7	Input #7
In8	Input #8

Description:

Sum of inputs:
+ ... Input will be added to result.
- ... Input will be subtracted from result.
0 ... Input will be ignored.

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

Name	FiP8
ID	4800
Revision	0.1
C filename	Sum_FiP8.c
H filename	Sum_FiP8.h

8 Bit Fixed Point Implementation

Controller Parameters	
sign	Bitfield with sign information of inputs

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In1;
      int8            *In2;
      int8            *In3;
      int8            *In4;
      int8            *In5;
      int8            *In6;
      int8            *In7;
      int8            *In8;
      int8            Out;
      uint16          sign;
} SUM_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	4801
Revision	0.1
C filename	Sum_FiP16.c
H filename	Sum_FiP16.h

16 Bit Fixed Point Implementation

Controller Parameters	
sign	Bitfield with sign information of inputs

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *In1;
      int16           *In2;
      int16           *In3;
      int16           *In4;
      int16           *In5;
      int16           *In6;
      int16           *In7;
      int16           *In8;
      int16           Out;
      uint16          sign;
} SUM_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	4802
Revision	0.1
C filename	Sum_FiP32.c
H filename	Sum_FiP32.h

32 Bit Fixed Point Implementation

Controller Parameters	
sign	Bitfield with sign information of inputs

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *In1;
      int32           *In2;
      int32           *In3;
      int32           *In4;
      int32           *In5;
      int32           *In6;
      int32           *In7;
      int32           *In8;
      int32           Out;
      uint16          sign;
} SUM_FIP32;
```

Implementation: Float32

Name	Float32
ID	4803
Revision	0.1
C filename	Sum_Float32.c
H filename	Sum_Float32.h

32 Bit Floating Point Implementation

Controller Parameters	
sign	Bitfield with sign information of inputs

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In1;
      float32         *In2;
      float32         *In3;
      float32         *In4;
      float32         *In5;
      float32         *In6;
      float32         *In7;
      float32         *In8;
      float32         Out;
      uint16          sign;
} SUM_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	4804
Revision	0.1
C filename	Sum_Float64.c
H filename	Sum_Float64.h

64 Bit Floating Point Implementation

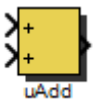
Controller Parameters	
sign	Bitfield with sign information of inputs

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In1;
      float64         *In2;
      float64         *In3;
      float64         *In4;
      float64         *In5;
      float64         *In6;
      float64         *In7;
      float64         *In8;
```

```
    float64         Out;
    uint16          sign;
} SUM_FLOAT64;
```

Block: uAdd



Inports	
In1	Addend 1
In2	Addend 2

Outports	
Out	Sum

Description:

Addition of input 1 and input 2 with output wrapping.

Calculation:

$$Out \quad = \quad In_1 + In_2$$

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

Name	FiP8
ID	4976
Revision	0.1
C filename	uAdd_FiP8.c
H filename	uAdd_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *In1;
      int8            *In2;
      int8            Out;
} UADD_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	4977
Revision	0.1
C filename	uAdd_FiP16.c
H filename	uAdd_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int16           *In1;
     int16           *In2;
     int16           Out;
} UADD_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	4978
Revision	0.1
C filename	uAdd_FiP32.c
H filename	uAdd_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
     uint16          ID;
     int32           *In1;
     int32           *In2;
     int32           Out;
} UADD_FIP32;
```

Implementation: Float32

Name	Float32
ID	4979
Revision	0.1
C filename	uAdd_Float32.c
H filename	uAdd_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *In1;
      float32         *In2;
      float32         Out;
} UADD_FLOAT32;
```

Implementation: Float64

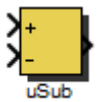
Name	Float64
ID	4980
Revision	0.1
C filename	uAdd_Float64.c
H filename	uAdd_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *In1;
      float64         *In2;
      float64         Out;
} UADD_FLOAT64;
```

Block: uSub



Inports	
Plus	Minuend
Minus	Subtrahend

Outports	
Out	Difference

Description:

Subtraction of input Minus from input Plus with output wrapping.

Calculation:

$$Out \quad = \quad Plus - Minus$$

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

Name	FiP8
ID	5008
Revision	0.1
C filename	uSub_FiP8.c
H filename	uSub_FiP8.h

8 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int8            *Plus;
      int8            *Minus;
      int8            Out;
} USUB_FIP8;
```

Implementation: FiP16

Name	FiP16
ID	5009
Revision	0.1
C filename	uSub_FiP16.c
H filename	uSub_FiP16.h

16 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int16           *Plus;
      int16           *Minus;
      int16           Out;
} USUB_FIP16;
```

Implementation: FiP32

Name	FiP32
ID	5010
Revision	0.1
C filename	uSub_FiP32.c
H filename	uSub_FiP32.h

32 Bit Fixed Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      int32           *Plus;
      int32           *Minus;
      int32           Out;
} USUB_FIP32;
```

Implementation: Float32

Name	Float32
ID	5011
Revision	0.1
C filename	uSub_Float32.c
H filename	uSub_Float32.h

32 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float32         *Plus;
      float32         *Minus;
      float32         Out;
} USUB_FLOAT32;
```

Implementation: Float64

Name	Float64
ID	5012
Revision	0.1
C filename	uSub_Float64.c
H filename	uSub_Float64.h

64 Bit Floating Point Implementation

Data Structure:

```
typedef struct {
      uint16          ID;
      float64         *Plus;
      float64         *Minus;
      float64         Out;
} USUB_FLOAT64;
```