EUROPEAN CONFERENCE ON PLANT & PROCESS SAFETY

HIMA SMART SAFETY.

HIMA Model based safety solutions - 14.09.2022

Bernd Schaefer – Product Manager Programming Software

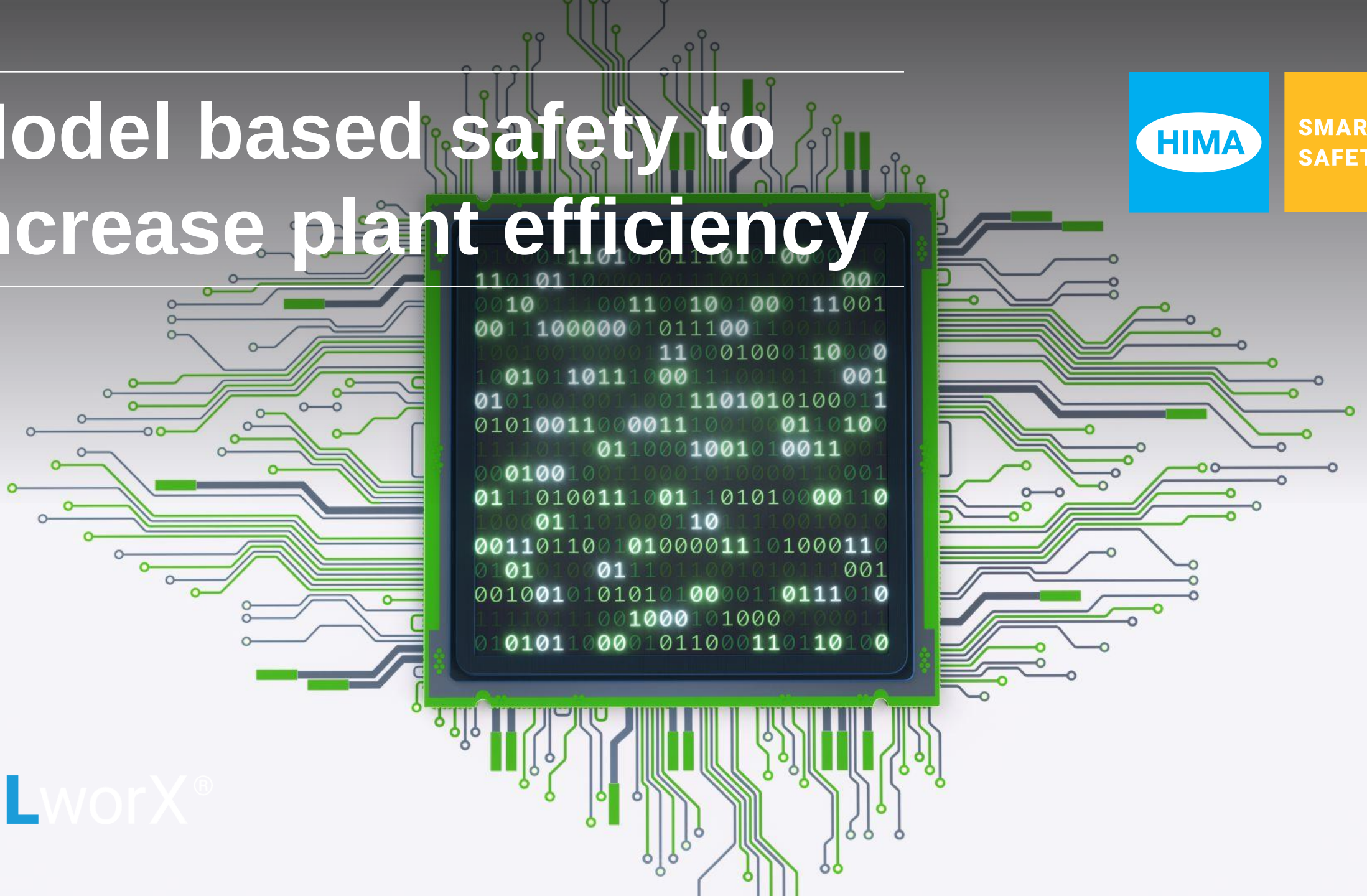
Bernd Schaefer

Product Manager – Programming Software

HIMA Paul Hildebrandt
Albert-Bassermann-Straße 28
68782 Brühl

Phone (0 62 02) 709 - 453
E-Mail b.schaefer@hima.com

Model based safety to increase plant efficiency

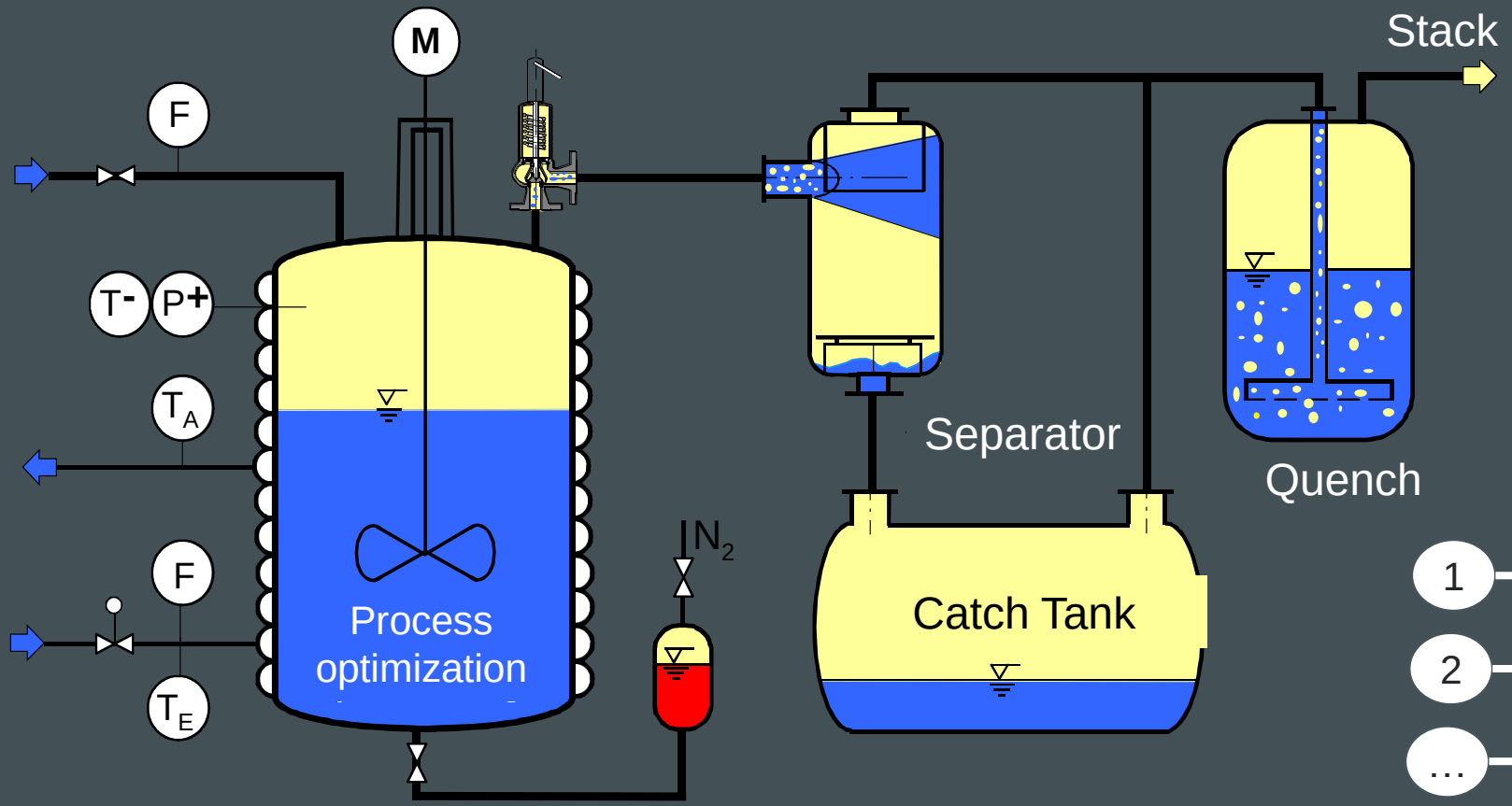


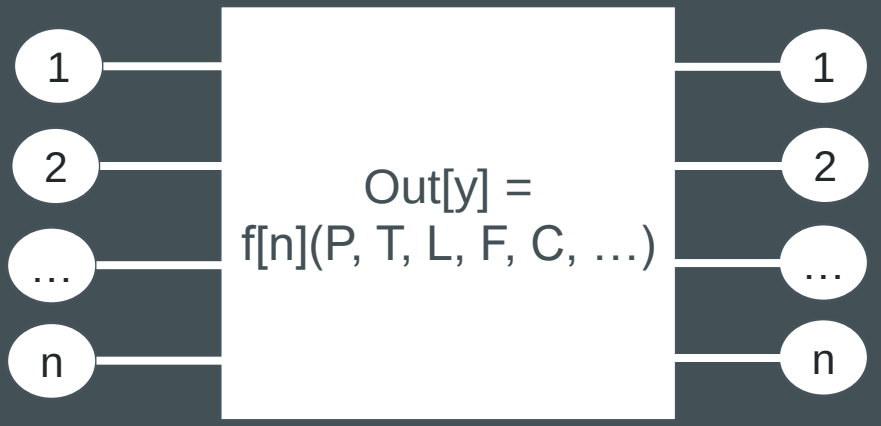
HIMA

SMART SAFETY.

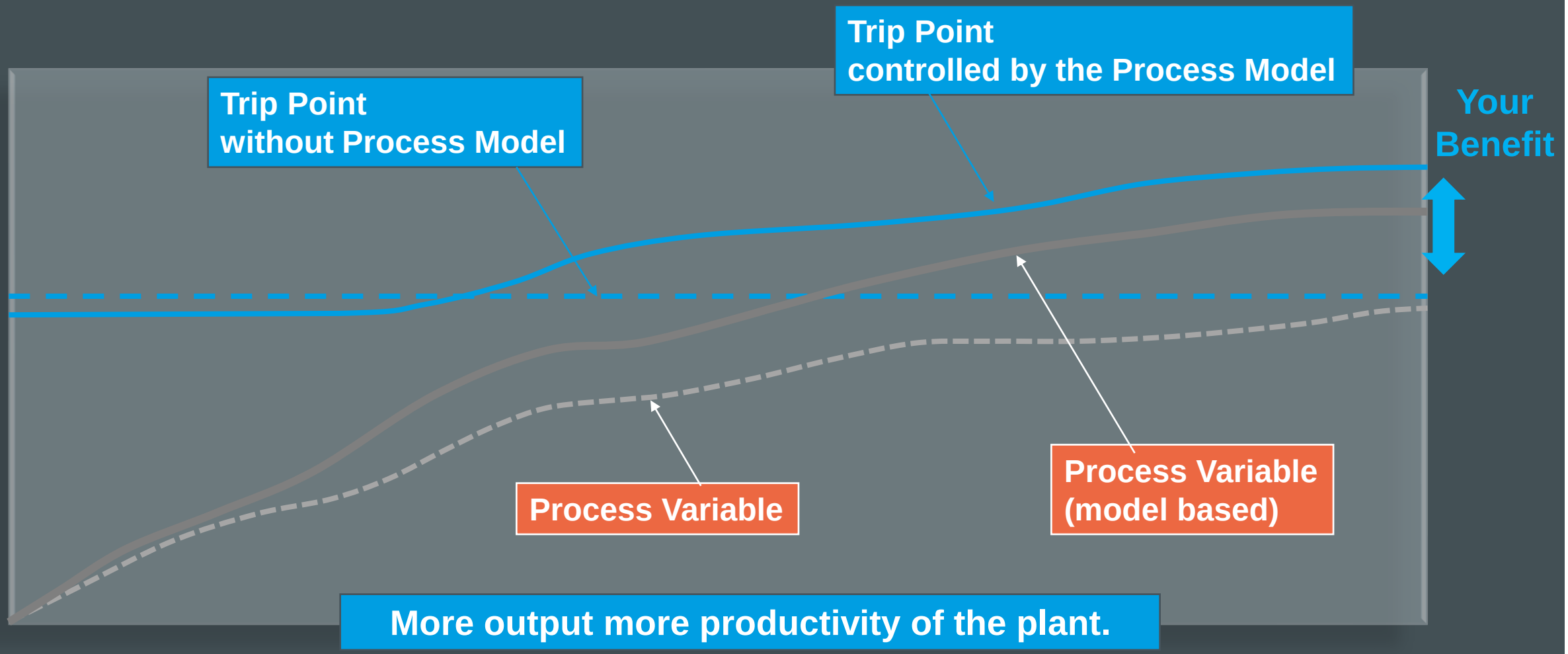
SILworX®

Motivation for model based safety solutions



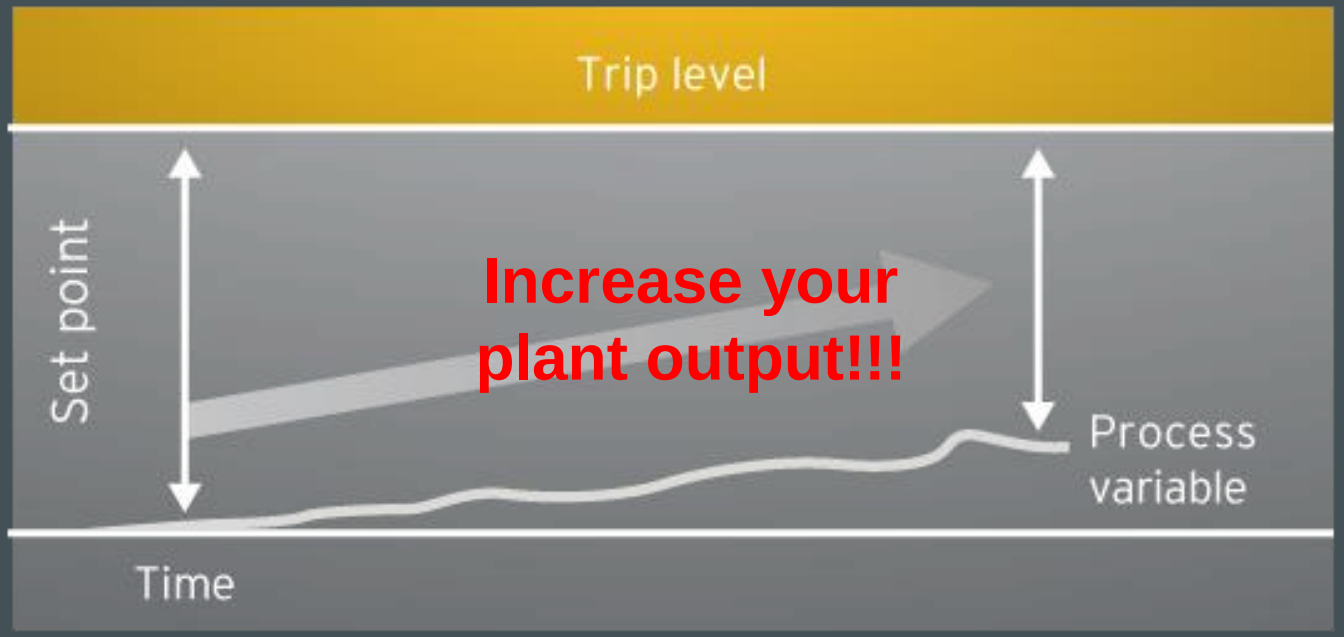


Benefits of Model based safety solutions



Benefits of Model based safety solutions

- Process optimization possible – allows new kind of processing
- Process safety optimization possible – availability instead of shut-down
- Enables additional safety for complex processing – lower risk / less effort

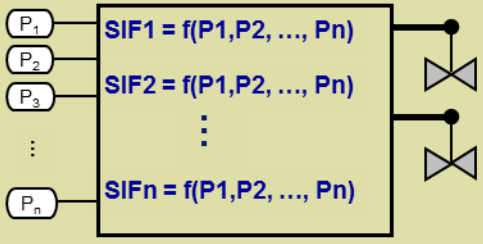


You know better, what it means to increase plant output by 20% to 30% in terms of money

Derived requirements for the Safety system

- High performance for mathematical operations
- Possibility for the integration of mathematical & statistical models for dynamically process control
- Simple implementation of complex mathematical functions
  - Differential equations, complex numbers, matrix operations, series expansions, …
- Easy implementation of models from simulation software (MATLAB Simulink)

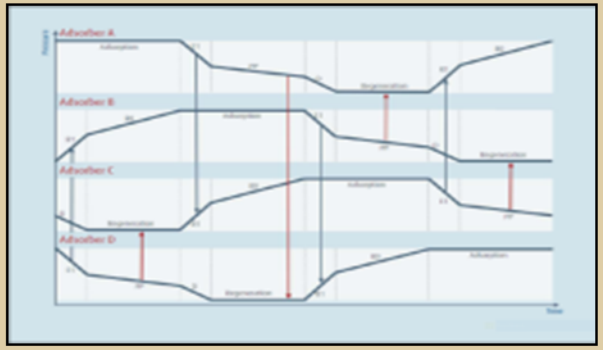
Complex mathematical functions



Recursive function development

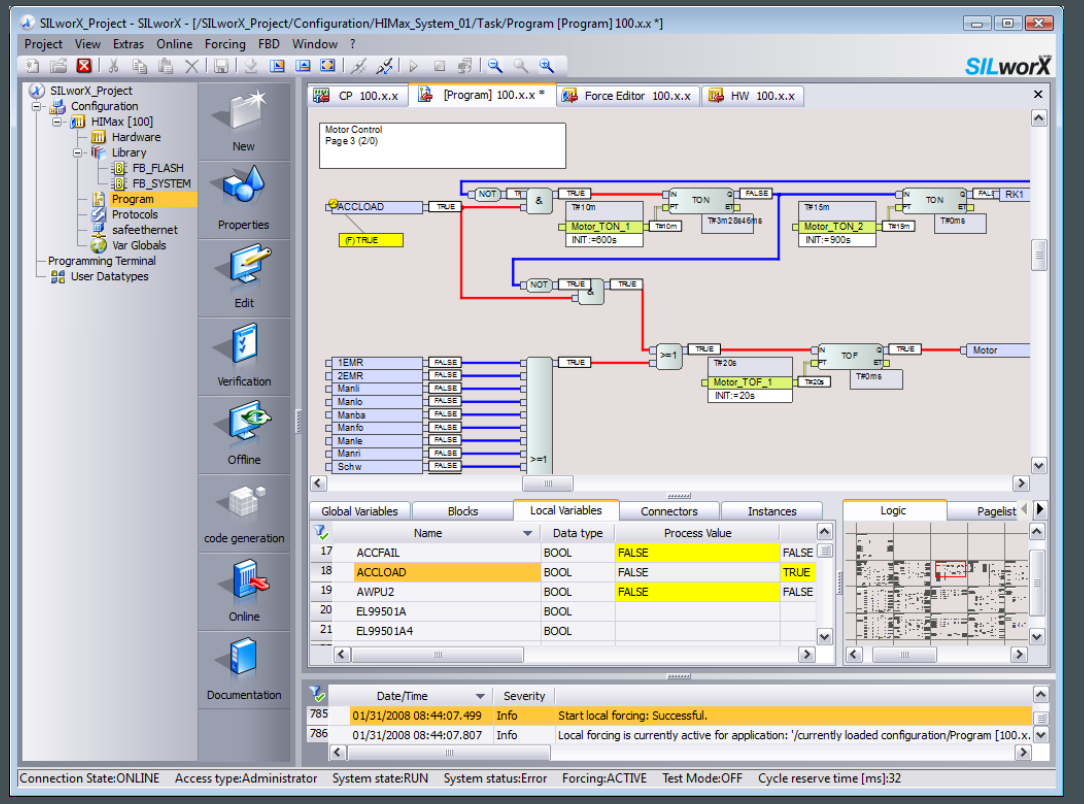
$$\sum_{n=0}^{\infty}(-1)^n \frac{z^{2n+1}}{(2n+1)!}$$

Complex status and error Dependent process sequences

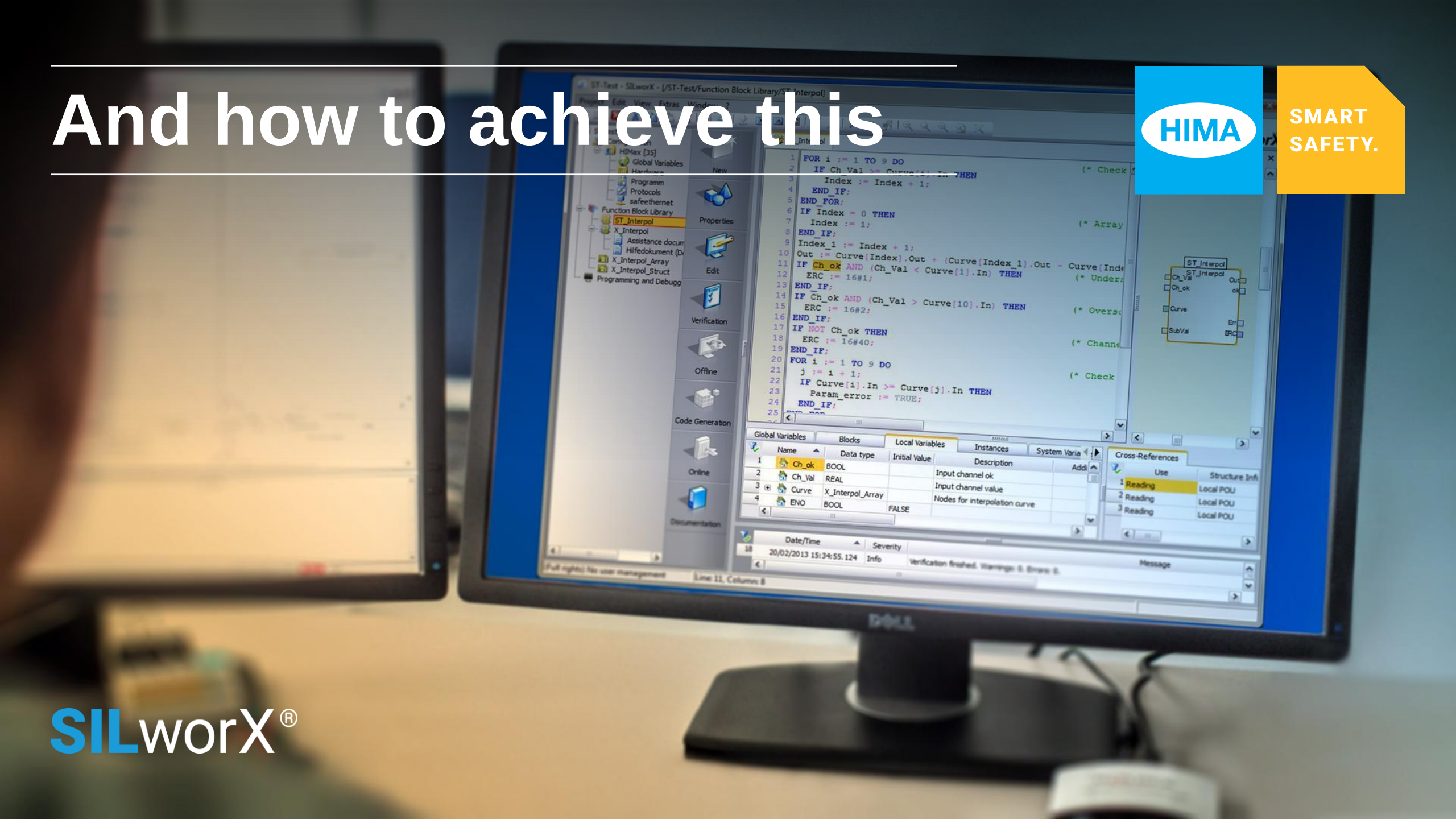


SILworX® High-End Safety Engineering

- One engineering tool for HIMax®, HIMatrix® and HIQuad X® Systems
- Fully integrated configuration, programming and diagnostic environment
- Offline simulation and online test
- SIL3 comparator for hardware and logic changes
- Supports reload functionality for hardware, communication and logic changes
- Multitasking for up to 32 independent programs
- Password and user management protection for projects and controller access
- IEC 61131-3 programming:
  - function block diagrams (FBD),
  - sequential function charts (SFC),
  - structured text (ST)
  - C++ interface – the base for model based safety



And how to achieve this



SILworX C++ FBs: Extend Your Flexibility

- Allow the user to embed their own C++ programs into the SILworX
- Inputs and outputs of C++ FBs serve as an interface to the functions of the embedded C++ programs (only elementary data types allowed!).
- Compared to function block diagrams or structured text: More flexible programming and higher performance.
- C++ source code from third-party tools (such as mathematics and simulation software) can be embedded into the SILworX
  -> Model based safety solutions

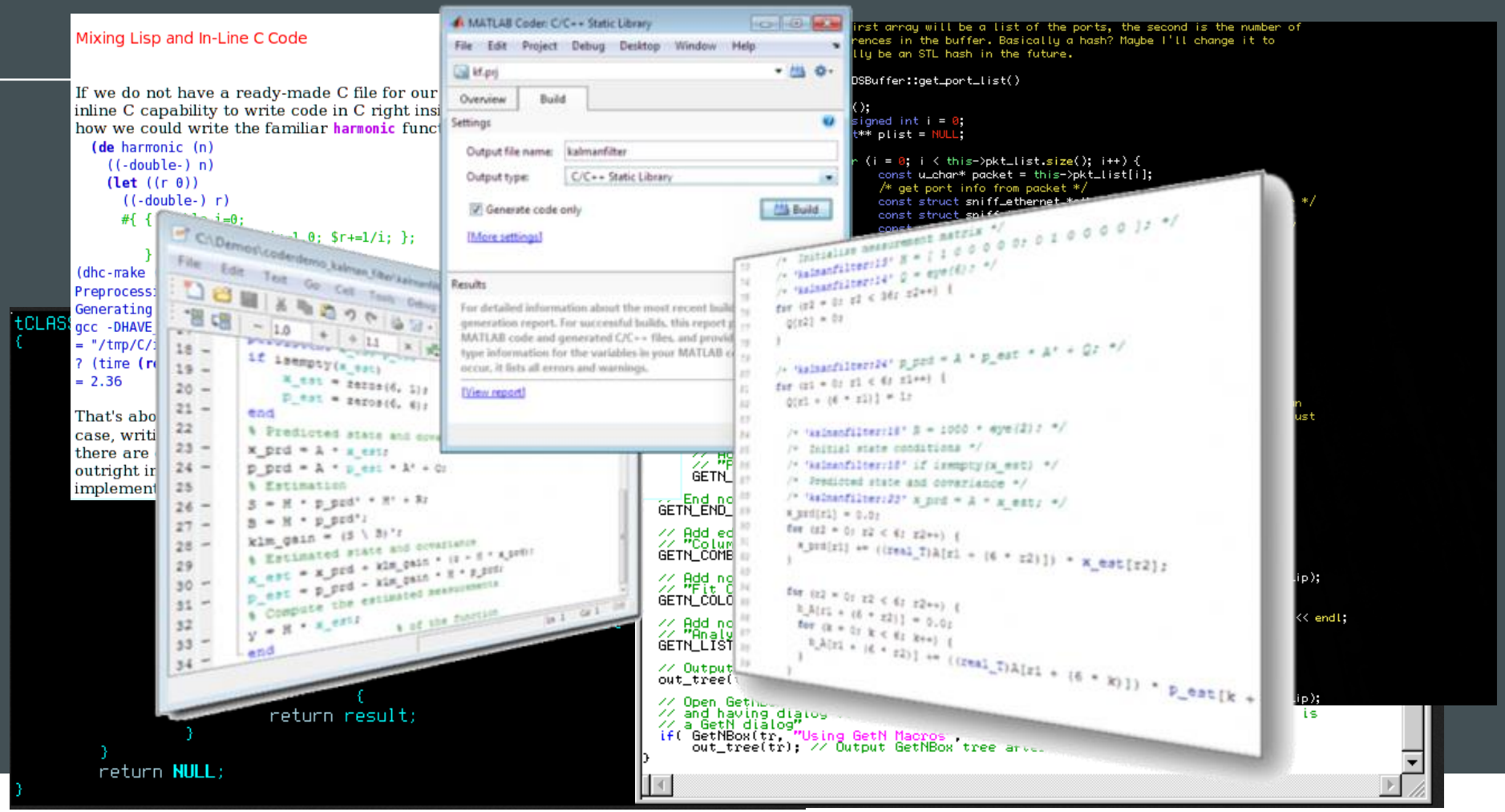
But……

- There are still some restrictions in c++ language usage (e.g. dynamic memory allocation, ……..
- C++ is a FVL (Full variability language), thus
  - C++ code imported in a SILworX function block to be developed according the V-model of IEC 61508 / EN 50128
  - Code must be completely pre-validated (tested)
  - Necessary documentation for this process to be created
  - No possibility to correctly and completely validate that code inside SILworX

don‘t forget…

It‘s a safety system!!!

Model based safety based on a C++ Function Block Interface – A new approach to process safety



Are there any Questions?

Simple things can be difficult!

Thank You.

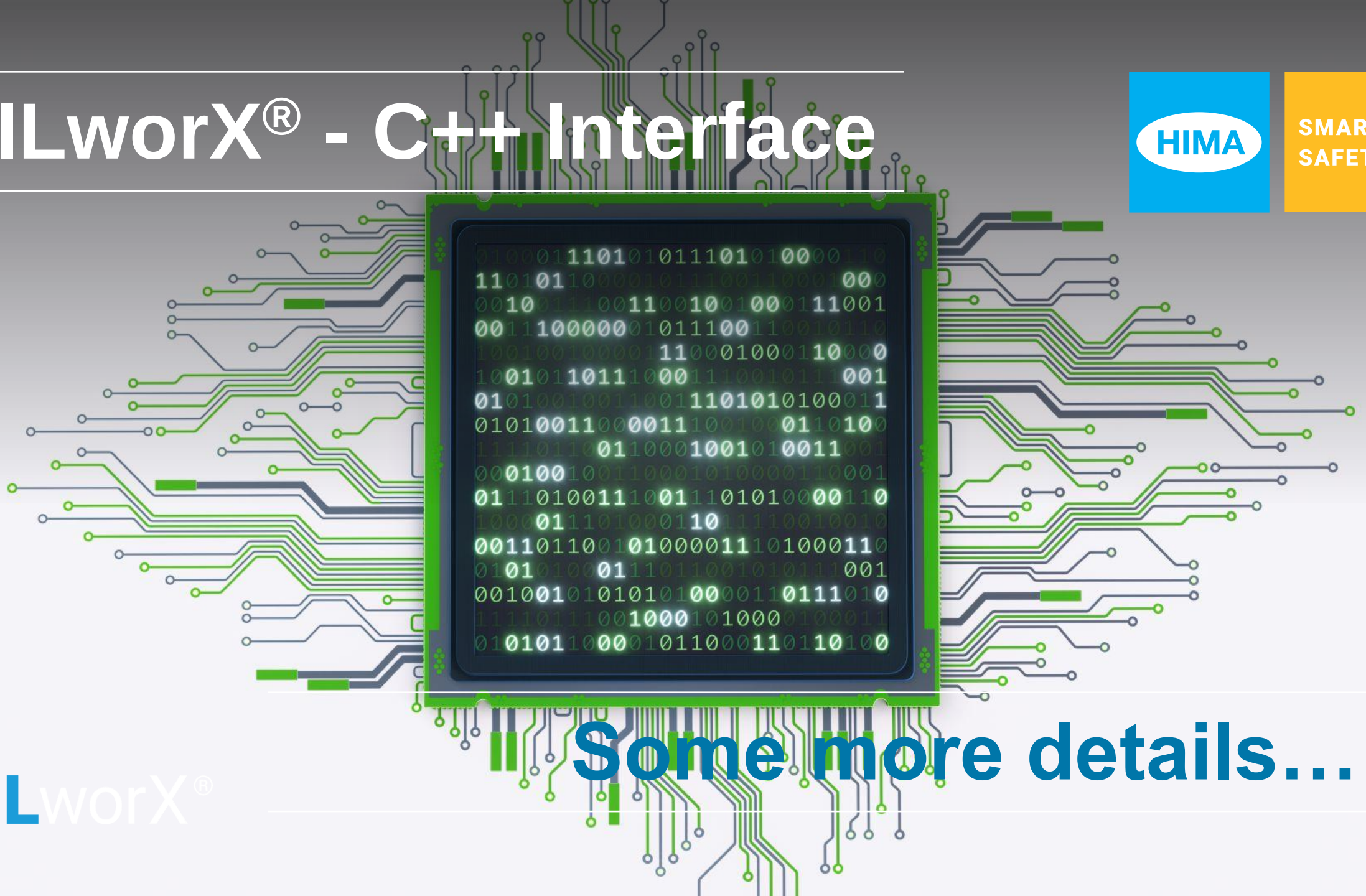
HIMA Paul Hildebrandt GmbH

Albert-Bassermann-Str. 28
68782 Brühl, Germany

Phone: +49 (0) 6202 / 709-0
Fax: +49 (0) 6202 / 709-107

Email: info@hima.com
Website: www.hima.com

SILworX® - C++ Interface



Some more details……

SILworX®

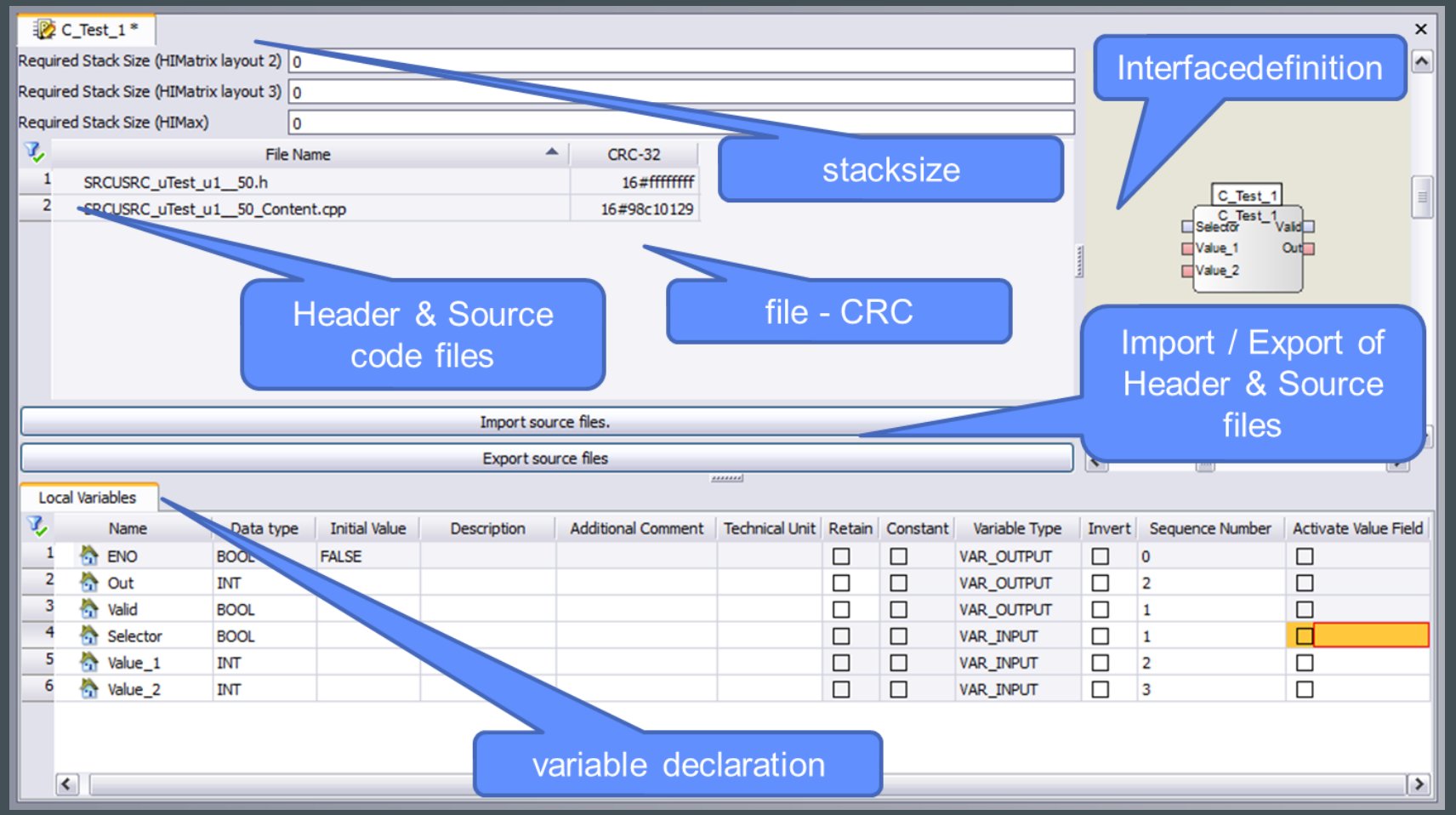
SILworX® C++ Function Block Interface - Usability

- In general all SILworX ® C++ function blocks can be used and implemented like all standard function blocks.
- SILworX ® C++ function block standard handling in SILworX ®
  - archive, restore
  - Cut, copy, paste
- SILworX ® C++ function block attributes
  - Normal
  - read-only
  - know-how protected
- RELOAD is possible
- processing in separate program recommended
  - Runtime monitoring
  - Time slice limitation

C++ Function Block Interface - Handling

1. create C++ function block and define properties
   - name
   - height, wideness, extendibility
   - variant
2. Define In-/ and Output variables
   - VAR_Input
   - VAR_Output
   - VAR
3. export C++ function block source code
4. edit C++ function block source code
5. import C++ function block source code and additional header and source code files

SILworX® C++ Function Block Interface - Editor



C++ Function Block Interface - Restrictions

- global variables can‘t be used in SILworX ® C++ function block
- SILworX ® standard function blocks can‘t be used in C++ function blocks
- user defined data types like arrays or structs can‘t be used in C++ function blocks
- limited C++ language features
- class structure of C++ function block has to be changed in the interface editor
- no offline-/ online simulation of the internal source code in SILworX ® possible
- after a RELOAD of a changed C++ function block the initial values are used for one cycle
- C++ function blocks can be only tested as a “blackbox”
- customer is responsible for internal safety relevant source code
  => FVL programming language => code qualification according to IEC 61508 Ed. 2 necessary
- C++ code development isn’t a HIMA focus

C++ Function Block Interface – stack & data

- automatically 4kByte stack reserved
- additional stack size has to be calculated or estimated

SILworX-Datentyp	Alias-Namen	C++-Datentyp	Größe in Byte
BOOL	-	bool	1
BYTE und USINT	ubyte, uint8	unsigned char	1
WORD und UINT	uword, uint16	unsigned short	2
DWORD und UDINT	udword, uint32	unsigned long	4
LWORD und ULINT	uldword, uint64	unsigned long long	8
SINT	int8	char	1
INT	int16	short	2
DINT	int32	long	4
LINT und TIME	int64, time	long long	8
REAL	real	float	4
LREAL	lreal	double	8

- Maximum stack size

Parameter	Beschreibung
Stack-Bedarf für HIMatrix Layout 2	Wertebereich: 0…1 048 064 Byte Standard: 0
Stack-Bedarf für HIMatrix Layout 3	Wertebereich: 0…5 177 344 Byte Standard: 0
Stack-Bedarf für HIMax	Wertebereich: 0…10 481 664 Byte Standard: 0

Details in SILworX C++ Function Block manual

SILworX® C++ Function Block Interface – language

5.3.1 General C++ Language Elements

- Constants with supported data types
- Local variables with supported data types
- C++ global variables with supported data types
- Literals
- Comments
- if .. else
- switch
- while, do..while
- for
- function declarations and function calls
- sizeof
- assignments
- return
- break
- continue
- external
  for C++ functions and variables only, but no external declarations with specification of a programming language, e.g., external "C" typedef void (*CFunction)(void) is illeagal.
- #include
- #ifdef, #if .. #elif .. #else .. #endif
- #define
- #undef (for self-defined elements only)

SILworX® C++ Function Block Interface – language

5.3.1 General C++ Language Elements

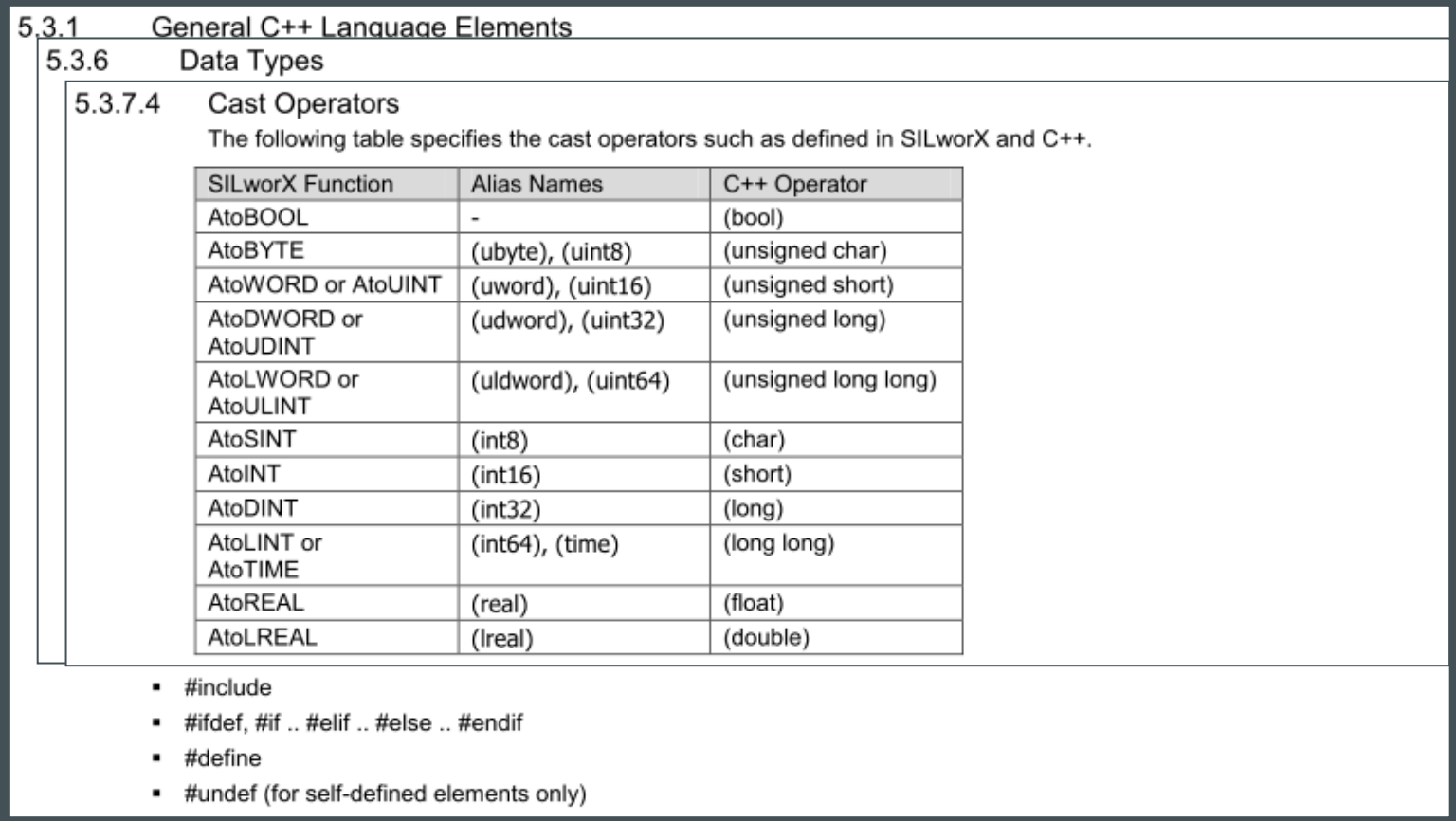
5.3.6 Data Types

The following table shows the matching data types as they are used in SILworX and C++, respectively. The corresponding alias names are also allowed in C++ source code.

SILworX Data Type	Alias Names	C++ Data Type
BOOL	-	bool
BYTE and USINT	ubyte, uint8	unsigned char
WORD and UINT	uword, uint16	unsigned short
DWORD and UDINT	udword, uint32	unsigned long
LWORD and ULINT	uldword, uint64	unsigned long long
SINT	int8	char
INT	int16	short
DINT	int32	long
LINT and TIME	int64, time	long long
REAL	real	float
LREAL	lreal	double

- #include
- #ifdef, #if .. #elif .. #else .. #endif
- #define
- #undef (for self-defined elements only)

SILworX® C++ Function Block Interface – language



5.3.1 General C++ Language Elements

5.3.6 Data Types

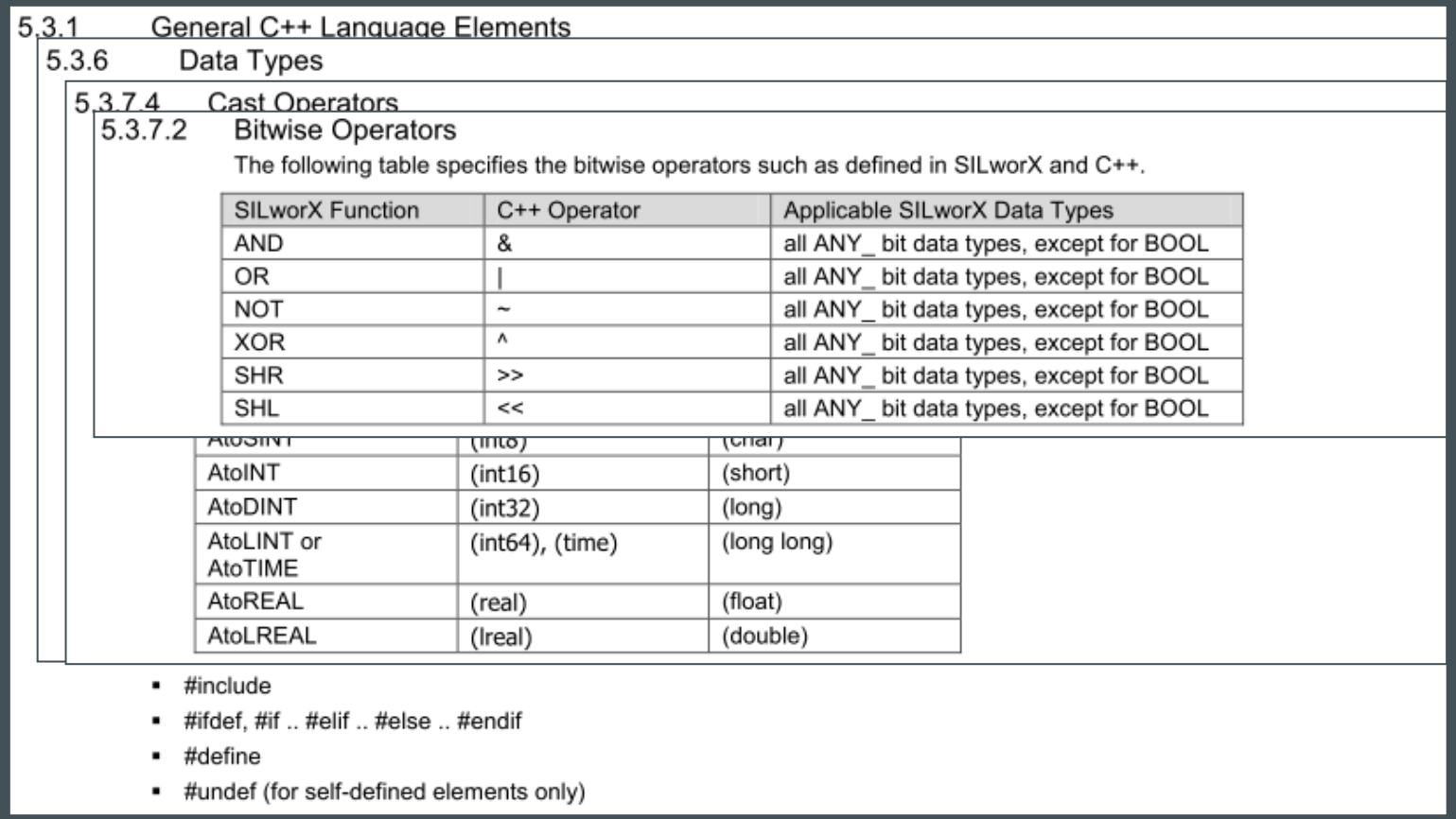
5.3.7.4 Cast Operators

The following table specifies the cast operators such as defined in SILworX and C++.

SILworX Function	Alias Names	C++ Operator
AtoBOOL	-	(bool)
AtoBYTE	(ubyte), (uint8)	(unsigned char)
AtoWORD or AtoUINT	(uword), (uint16)	(unsigned short)
AtoDWORD or AtoUDINT	(udword), (uint32)	(unsigned long)
AtoLWORD or AtoULINT	(uldword), (uint64)	(unsigned long long)
AtoSINT	(int8)	(char)
AtoINT	(int16)	(short)
AtoDINT	(int32)	(long)
AtoLINT or AtoTIME	(int64), (time)	(long long)
AtoREAL	(real)	(float)
AtoLREAL	(lreal)	(double)

- #include
- #ifdef, #if .. #elif .. #else .. #endif
- #define
- #undef (for self-defined elements only)

SILworX® C++ Function Block Interface – language



5.3.1 General C++ Language Elements

5.3.6 Data Types

5.3.7.4 Cast Operators

5.3.7.2 Bitwise Operators

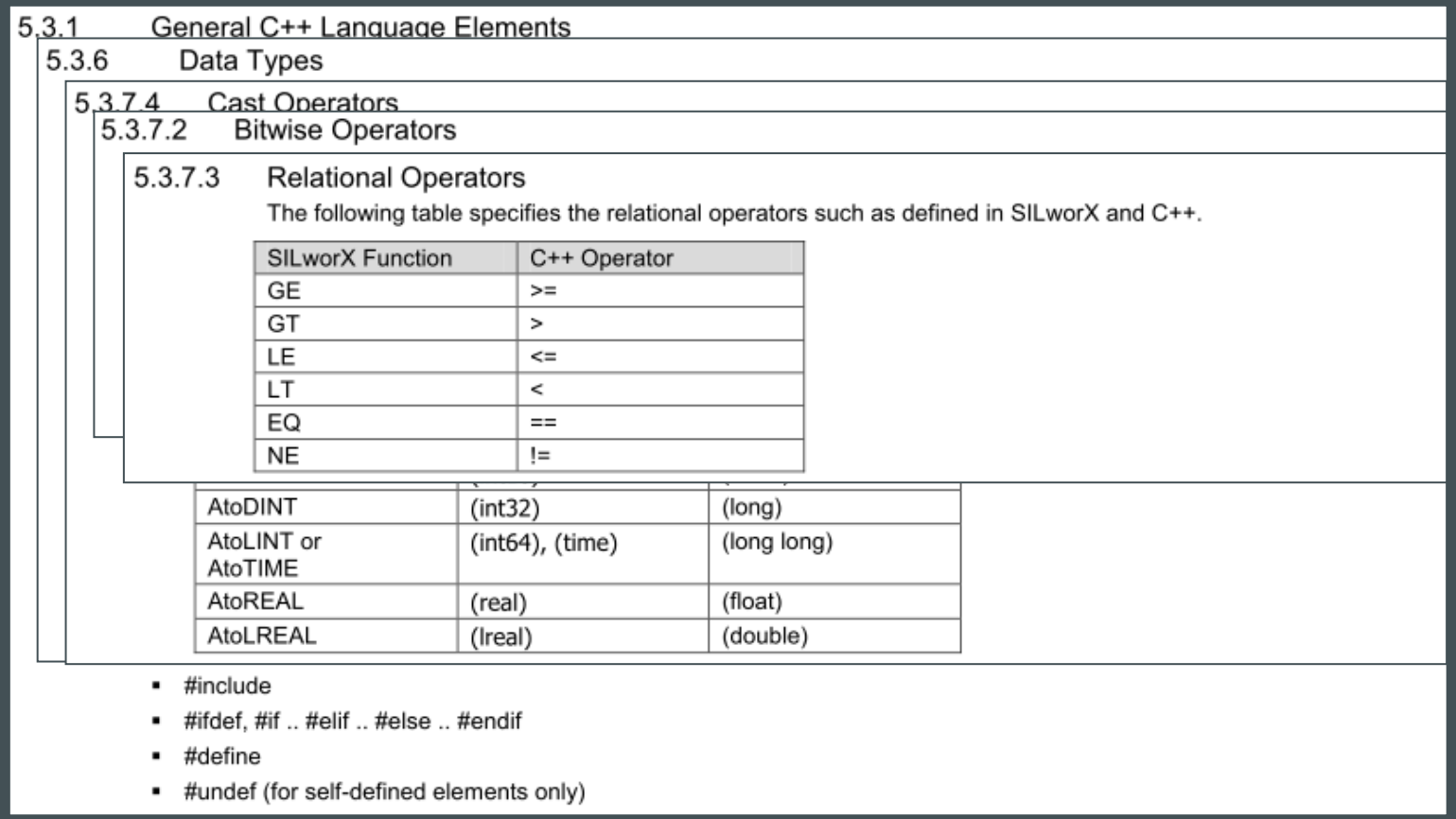
The following table specifies the bitwise operators such as defined in SILworX and C++.

SILworX Function	C++ Operator	Applicable SILworX Data Types
AND	&	all ANY_ bit data types, except for BOOL
OR	|	all ANY_ bit data types, except for BOOL
NOT	~	all ANY_ bit data types, except for BOOL
XOR	^	all ANY_ bit data types, except for BOOL
SHR	>>	all ANY_ bit data types, except for BOOL
SHL	<<	all ANY_ bit data types, except for BOOL

AtoSINT	(int8)	(char)
AtoINT	(int16)	(short)
AtoDINT	(int32)	(long)
AtoLINT or AtoTIME	(int64), (time)	(long long)
AtoREAL	(real)	(float)
AtoLREAL	(lreal)	(double)

- #include
- #ifdef, #if .. #elif .. #else .. #endif
- #define
- #undef (for self-defined elements only)

SILworX® C++ Function Block Interface – language



5.3.1 General C++ Language Elements

5.3.6 Data Types

5.3.7.4 Cast Operators

5.3.7.2 Bitwise Operators

5.3.7.3 Relational Operators

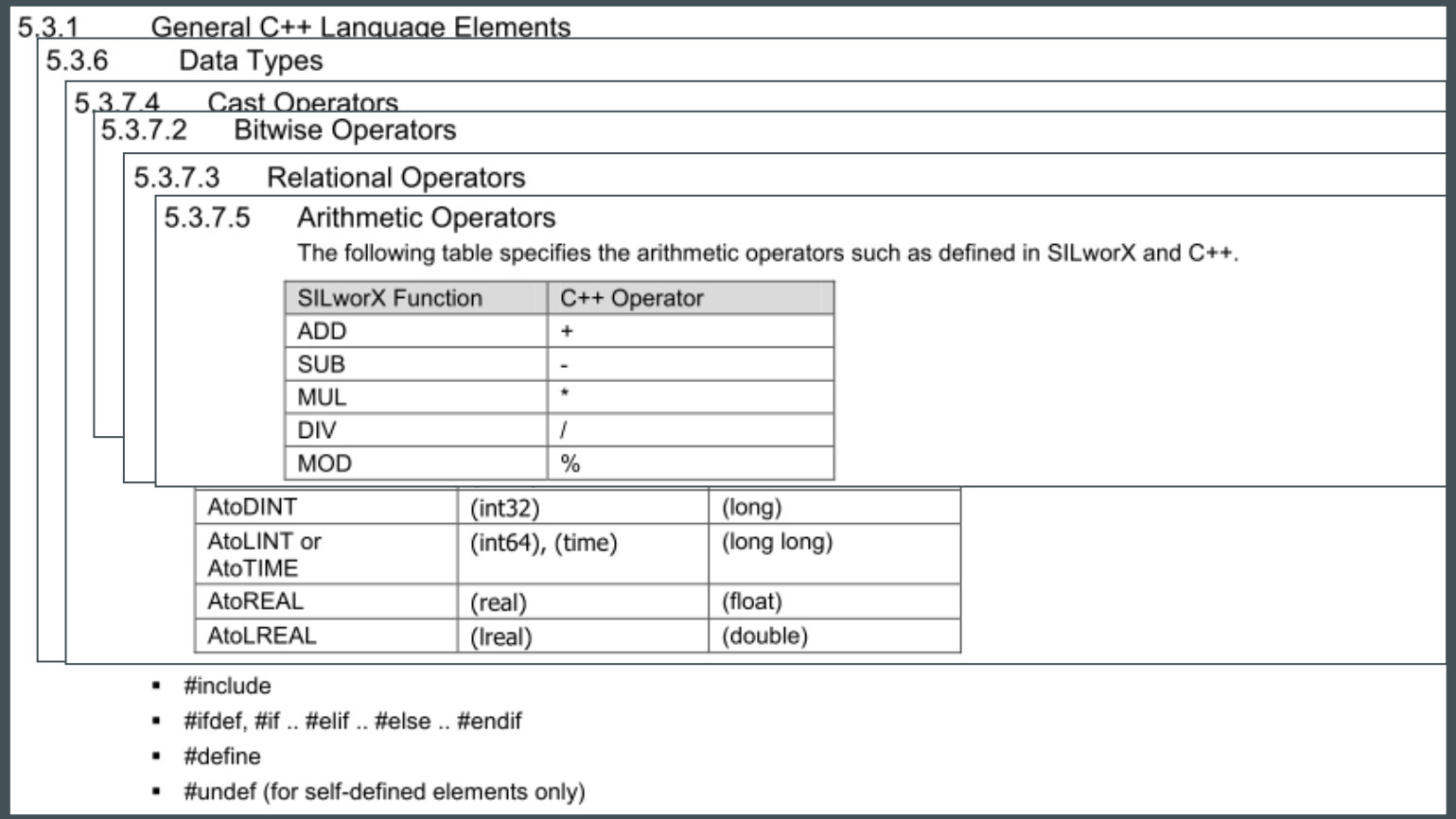
The following table specifies the relational operators such as defined in SILworX and C++.

SILworX Function	C++ Operator
GE	>=
GT	>
LE	<=
LT	<
EQ	==
NE	!=

AtoDINT	(int32)	(long)
AtoLINT or AtoTIME	(int64), (time)	(long long)
AtoREAL	(real)	(float)
AtoLREAL	(lreal)	(double)

- #include
- #ifdef, #if .. #elif .. #else .. #endif
- #define
- #undef (for self-defined elements only)

SILworX® C++ Function Block Interface – language



5.3.1 General C++ Language Elements

5.3.6 Data Types

5.3.7.4 Cast Operators

5.3.7.2 Bitwise Operators

5.3.7.3 Relational Operators

5.3.7.5 Arithmetic Operators

The following table specifies the arithmetic operators such as defined in SILworX and C++.

SILworX Function	C++ Operator
ADD	+
SUB	-
MUL	*
DIV	/
MOD	%

AtoDINT	(int32)	(long)
AtoLINT or AtoTIME	(int64), (time)	(long long)
AtoREAL	(real)	(float)
AtoLREAL	(lreal)	(double)

- #include
- #ifdef, #if .. #elif .. #else .. #endif
- #define
- #undef (for self-defined elements only)

SILworX® C++ Function Block Interface – language

5.3.1 General C++ Language Elements

5.3.6 Data Types

5.3.7.4 Cast Operators

5.3.7.2 Bitwise Operators

5.3.7.3 Relational Operators

5.3.7.5 Arithmetic Operators

The following table specifies the arithmetic operators such as defined in SILworX and C++.

5.3.7.6 Numeric Functions

The following table specifies the numeric functions such as defined in SILworX and C++.

SILworX Function	C++ Function
SQRT	sqrtf(float), sqrt(double)
ABS	abs(char), abs(short), labs(long), llabs(long long), fabsf(float), fabs(double)
SIN	sinf(float), sin(double)
ASIN	asinf(float), asin(double)
COS	cosf(float), cos(double)
ACOS	acosf(float), acos(double)
TAN	tanf(float), tan(double)
ATAN	atanf(float), atan(double)
EXPT	powf(float, float), pow(double, double)
EXP	expf(float), exp(double)
LN	logf(float), log(double)
LOG	log10f(float), log10(double)

C++ Function Block Interface – source code

- Declaration of global C++ variables possible
  - valid for all instances of the function block
- standard header files
  - iec_types.h
  - iec_std_types.h
- additional header files
  - float.h
  - limits.h
  - math.h
  - stddef.h

SILworX® – source code

- no dynamic memory allocation e.g. [malloc(Sizeof …)]
- no recursive C++ function block call
- no interrupts
- no GoTo commands
- no inline assembler
- no pragmas
  - #pragma name
- no runtime type information (RTTI)
  - dynamic_cast
  - typeid
  - exception handling
- no class destructor (~classname)

Again!!
It is still a safety system!!!

SILworX® – header & cpp

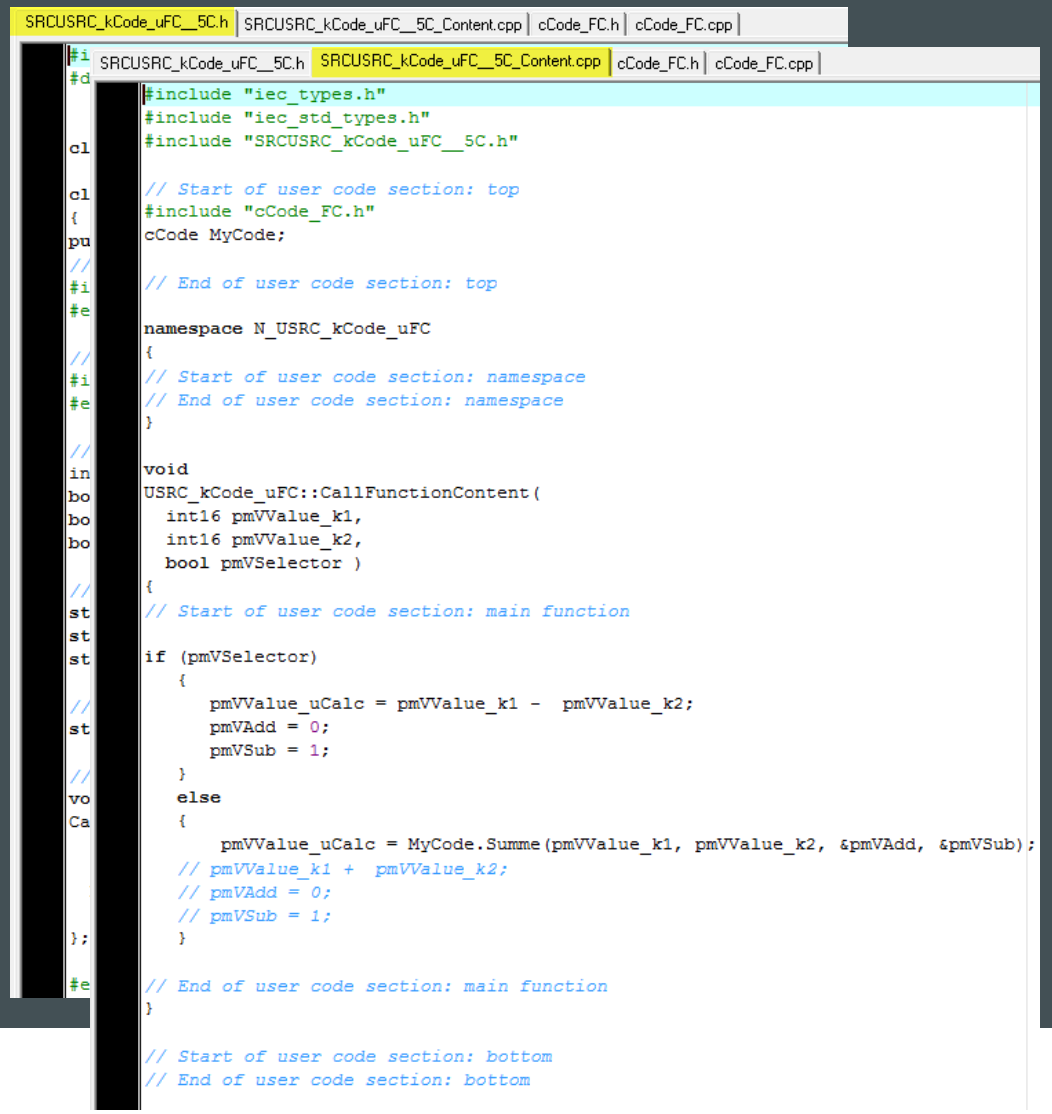
- header and sourcecode file name depending on C++ function block name
- basic structure must not be changed
  - SRCUSRC_***.h
  - SRCUSRC_***.cpp
- additional header and source code files can be implemented, e.g.
  - cCode_FC.h
  - cCode_FC.cpp

SRCUSRC_kCode_uFC__5C.h | SRCUSRC_kCode_uFC__5C_Content.cpp | cCode_FC.h | cCode_FC.cpp

```
#ifndef SRCUSRC_kCode_uFC__5C
#define SRCUSRC_kCode_uFC__5C


class USRC_kCode_uFC;

class USRC_kCode_uFC
{
public:
// force switches
#if X_WITH_LOCAL_FORCE_DATA
#endif

// force values
#if X_WITH_LOCAL_FORCE_DATA
#endif

// process values, instances
int16 pmVValue_uCalc;
bool pmVAdd;
bool pmVENO;
bool pmVSub;

// VAR_INPUT initial values
static int16 const InitmVValue_k1;
static int16 const InitmVValue_k2;
static bool const InitmVSelector;

// POU initial values
static USRC_kCode_uFC const Init;

// Logic
void
CallFunctionContent(
  int16 pmVValue_k1 = InitmVValue_k1,
  int16 pmVValue_k2 = InitmVValue_k2,
  bool pmVSelector = InitmVSelector );

};

#endif
```

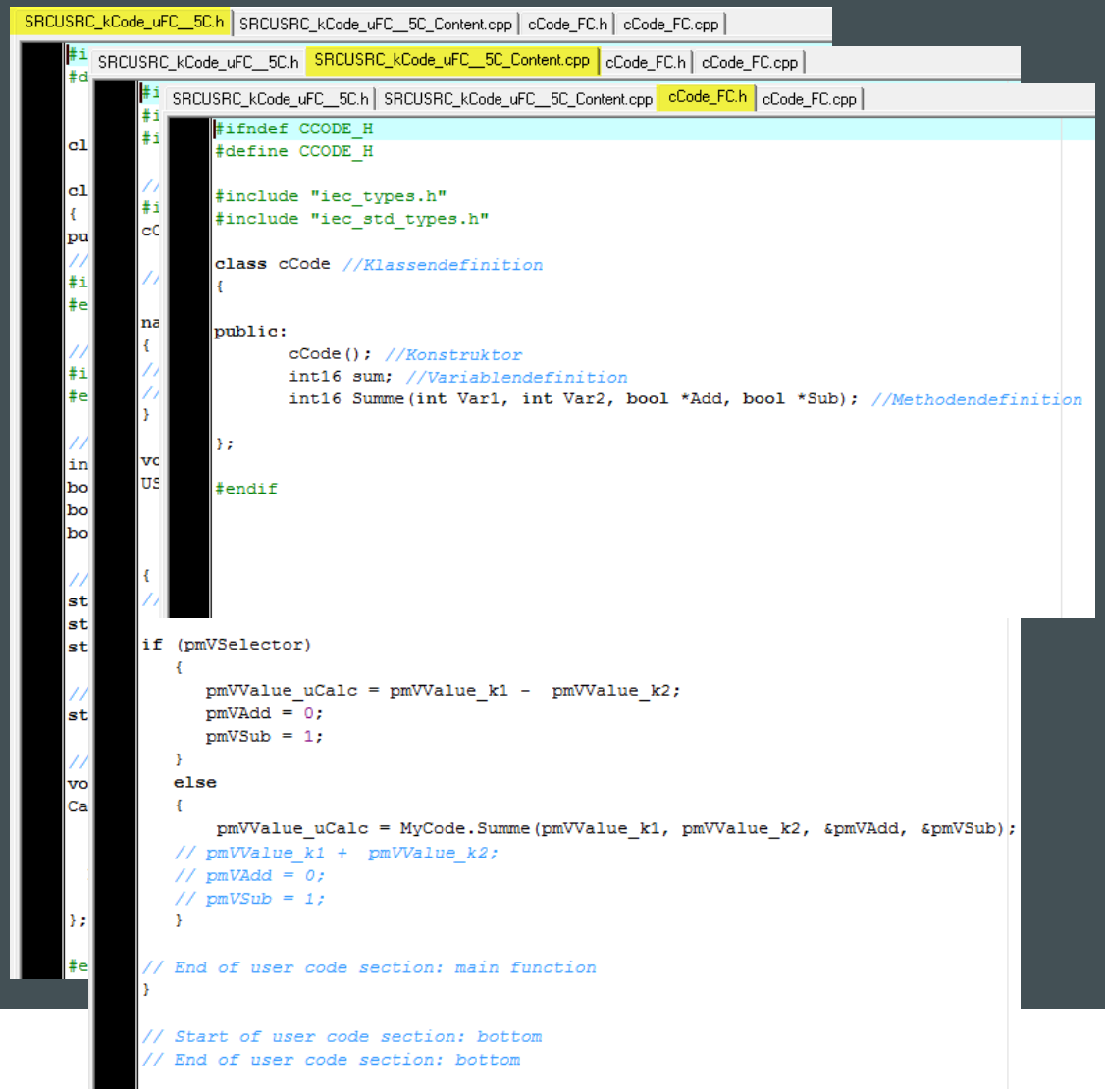
SILworX® – header & cpp

- header and sourcecode file name depending on C++ function block name
- basic structure must not be changed
  - SRCUSRC_***.h
  - SRCUSRC_***.cpp
- additional header and source code files can be implemented, e.g.
  - cCode_FC.h
  - cCode_FC.cpp



SILworX® – header & cpp

- header and sourcecode file name depending on C++ function block name
- basic structure must not be changed
  - SRCUSRC_***.h
  - SRCUSRC_***.cpp
- additional header and source code files can be implemented, e.g.
  - cCode_FC.h
  - cCode_FC.cpp



SILworX® – header & cpp

- header and sourcecode file name depending on C++ function block name
- basic structure must not be changed
  - SRCUSRC_***.h
  - SRCUSRC_***.cpp
- additional header and source code files can be implemented, e.g.
  - cCode_FC.h
  - cCode_FC.cpp