

Chapter 18 – Planning Tasks

Introduction

As an engineer in controls work, one bumps into problems with timing and task planning on a regular basis. It is never expected but there are times when the problems of computer timing seem to beg to be considered. Usually, the problem is simple and only takes a few minutes of our day. Other times, the problems of tasks being shared and I/O read are a problem worth giving much time to. The lyrics of the song below tell of a poor guy who didn't plan well and was stuck. Give a listen on the youtube version if led. See why poor Charlie was forever stuck on the MTA.

“The M.T.A.

[The Kingston Trio](#)

These are the times that try men's souls

In the course of our nation's history the people of Boston have rallied bravely whenever the rights of men have been threatened

Today a new crisis has arisen

The Metropolitan Transit Authority, better known as the M.T.A.

Is attempting to levy a burdensome tax on the population in the form of a subway fare increase

Citizens, hear me out, this could happen to you!

Well, let me tell you of the story of a man named Charlie

On a tragic and fateful day

He put ten cents in his pocket, kissed his wife and family

Went to ride on the MTA

Well, did he ever return?

No he never returned and his fate is still unlearned (what a pity)

He may ride forever 'neath the streets of Boston

He's the man who never returned

Charlie handed in his dime at the Kendall Square station

And he changed for Jamaica Plain

When he got there the conductor told him, "one more nickel"

Charlie couldn't get off of that train!

But did he ever return?

No he never returned and his fate is still unlearned (poor old Charlie)

He may ride forever 'neath the streets of Boston

He's the man who never returned

Now, all night long Charlie rides through the station

Crying, "what will become of me?"

How can I afford to see my sister in Chelsea

Or my cousin in Roxbury?"

But did he ever return?

No he never returned and his fate is still unlearned (shame and scandal)

He may ride forever 'neath the streets of Boston

He's the man who never returned

Charlie's wife goes down to the Scollay Square station

Every day at quarter past two

And through the open window she hands Charlie a sandwich

As the train comes rumbling through!

But did he ever return?

No he never returned and his fate is still unlearned (he may ride forever)
He may ride forever 'neath the streets of Boston
He's the man who never returned
Pick it Davey
Kinda hurts my fingers
Now, you citizens of Boston, don't you think it's a scandal
How the people have to pay and pay?
Fight the fare increase, vote for George O'Brian
Get poor Charlie off the MTA!
Or else he'll never return
No he'll never return and his fate is still unlearned (just like Paul Revere)
He may ride forever 'neath the streets of Boston
He's the man who never returned
He's the man who never returned
He's the man who never returned
Et tu, Charlie?"
Source: [LyricFind](#)

Hopefully you will never experience the same fate as Charlie! The song is old but not planning especially regarding computer timing, can lead to some serious catastrophic problems.

An Anecdotal Concerning Timing

One simple problem encountered some years ago had to do with a high-speed counter card. The card was purchased to count the pulses from a pulse output flow meter dealing with flow to a mixer. The flow was to turn off within .1 gallon of total flow from the liquid's feed line. This was not happening. Every time the ingredient was called, the accuracy was off by .7 or .8 gallon, either plus or minus and never predictable. The time spent on this was of great concern because there was an obvious waste of each batch that had to be scrapped since the liquid was so far out of specification. It was also pointed out that the original flow controller was able to fill the batch to within .05 gallon repeatedly.

Everything looked to be in order. The pulses were coming into the high-speed card accurately and the PLC was scanning the card often enough to read and turn on an output every 10 ms or so. So where was the problem?

Not until a program was written to show the delay between the update from the high-speed counter card to the CPU was the source of the problem revealed. The high-speed counter card was updating but not updating the count to the CPU more than about every .75 seconds, enough to cause the .7 or .8 gallon error. The program, while simple to write, revealed the problem to the engineer. The card was obviously flawed and needed to be replaced. The question became one of was this for this card alone or was the problem with all cards of this design. That question was answered by calling the manufacturer and asking that they put the card on a bench to test for the same problem. When this occurred, it was found that the problem was across the board and the card was essentially not able to control the flow of liquid accurately enough to accurately fill our batch. Another solution would need to be found.

A solution was found that was accurate enough and the problem solved. The manufacturer was embarrassed by this obvious flaw in the design of his I/O card and the problem was only exposed by writing a program to show the delay between updates of the pulsed input to the cpu. He

admitted that the card was purchased from a third party (not associated with the manufacturer) and that the design had never been thoroughly checked out (or they would have found the problem themselves).

The solution was to create a frequency divider for the pulsed input from the flow controller and read the inputs into the cpu directly at a slower rate that was able to be updated by the discrete I/O card on the PLC. To slow down the pulse rate was acceptable in this case since the accuracy of the cpu's scan rate exceeded that needed to turn off the flow valve after a setpoint had been reached. Problem solved!

The following shows a description of the original pulse followed by a frequency divider and then a second frequency divider. It was tempting to build this frequency divider using chips but that is not typically acceptable in a factory environment. A Red Lion counter gave the desired output of a 4:1 count with four input pulses giving a toggled output. This output was slow enough to be read successfully by the digital input on the PLC. The PLC was a Modicon.

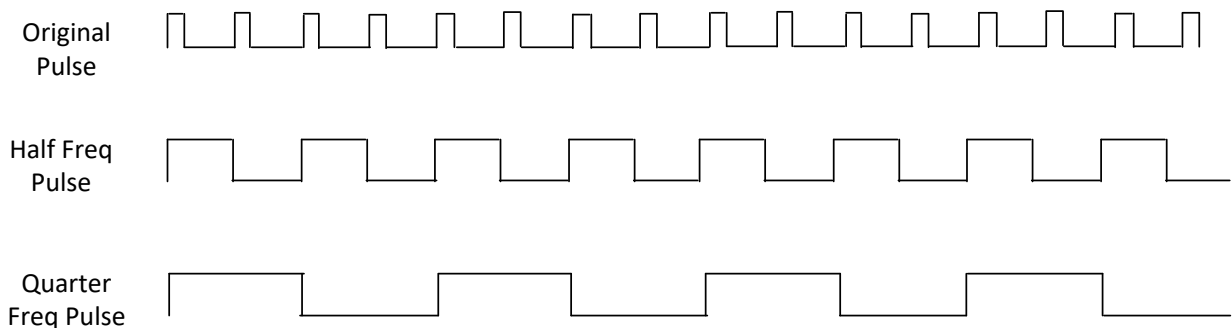


Fig. 18-1

Both A-B and Siemens have methods of solving problems such as the one above. Both A-B and Siemens have aggressive methods for dealing with timing issues.

Both use a method to read I/O on an immediate basis rather than from the scan's own I/O update. If necessary, they allow an instruction to have an immediate read from an input or an immediate write to an output. This is necessary for the processor to guarantee some of the simple high-speed problems of a process. The rest of the chapter outlines some of the more sophisticated methods for problem solving of timing issues and diagnosis of problems when they arise. In Chapter 19, it will be obvious that a scheduled program is needed to help in providing a best possible performance from the PID loops being programmed.

Allen-Bradley - Managing Tasks

The default RSLogix 5000 project provides a single task for all logic. While this is sufficient for many applications, some need more than a single task to perform the requirements of the project.

A Logix5000 controller supports multiple tasks to schedule and prioritize the execution of your programs based on specific criteria. This balances the processing time of the controller.

- The controller executes only one task at one time
- A different task can interrupt a task that is executing and take control

- In any given task, only one program executes at one time
- A Logix5000 controller supports three types of tasks:

Continuous
Periodic
Event

The continuous task runs in the background. It is the Main program and its sub-programs. It contains most of the code for the program. The periodic task runs on a clock. Programs such as a PID algorithm needs a constant clock execution time to correctly calculate a new output for the PID block. Also included in the PID algorithm is the requirement that the analog values be collected in a timely manner. That is, the analog values cannot be from about a scan ago. They must be refreshed just before needed in the clocked program. An event task runs based on an interrupt. These programs must be guarded in that not too many programs must be created or their frequency must be low in order to be run successfully.

To assign a priority to a task, use these guidelines:

“

If you want	Then	Notes
This task to interrupt another task	Assign a priority number that is less than (higher priority) the priority number of the other task	<ul style="list-style-type: none"> • A higher priority task interrupts all lower priority tasks • A higher priority task can interrupt a lower priority task multiple times
Another task to interrupt this task	Assign a priority number that is greater than (lower priority) the priority number of the other task	
This task to share controller time with another task	Assign the same priority number to both tasks	The controller switches back and forth between each task and executes each one for 1 ms

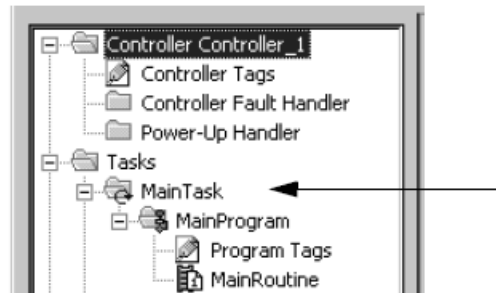
”

Leave enough time for unscheduled communication. Unscheduled communication occurs only when a periodic or event task is not running. If you use multiple tasks, make sure that the scan times and execution intervals leave enough time for unscheduled communication. Use these methods to plan enough unscheduled communication time.

1. Verify that the execution time of a highest priority task is significantly less than its specified period.
2. Verify that the total execution time of all tasks is significantly less than the period of the lowest priority tasks.

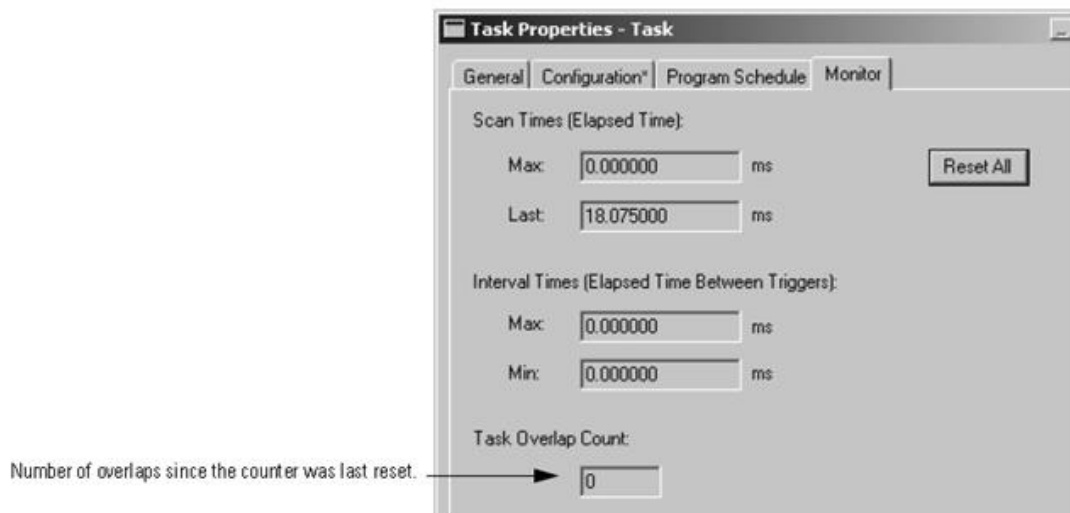
Manually check for overlaps using the following steps:

1. In the Controller Organizer, right-click Main Task and choose Properties.



The Task Properties dialog box appears.

2. Click the Monitor tab.



3. Click OK.

Fig. 18-2

Module Input Data State Change Trigger:

To trigger an event task based on data from an input module, use the module Input Data State Change trigger shown below:

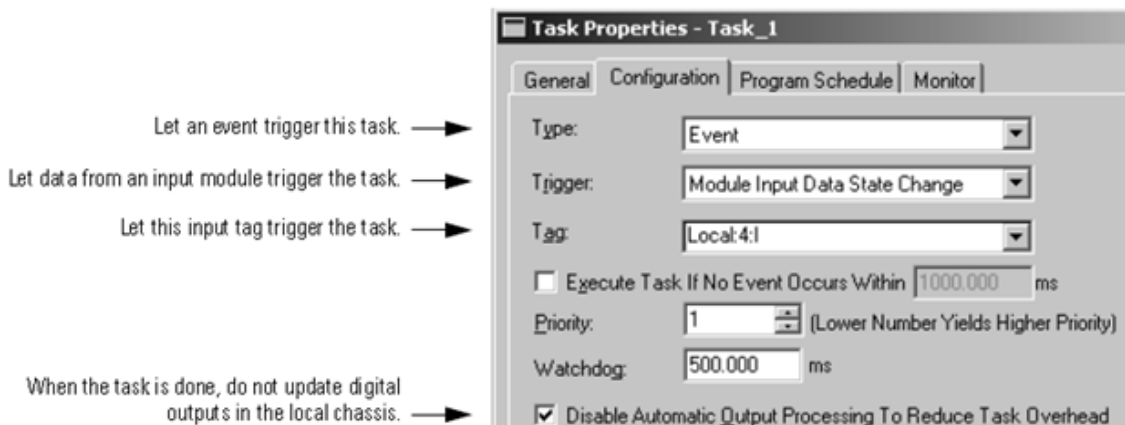


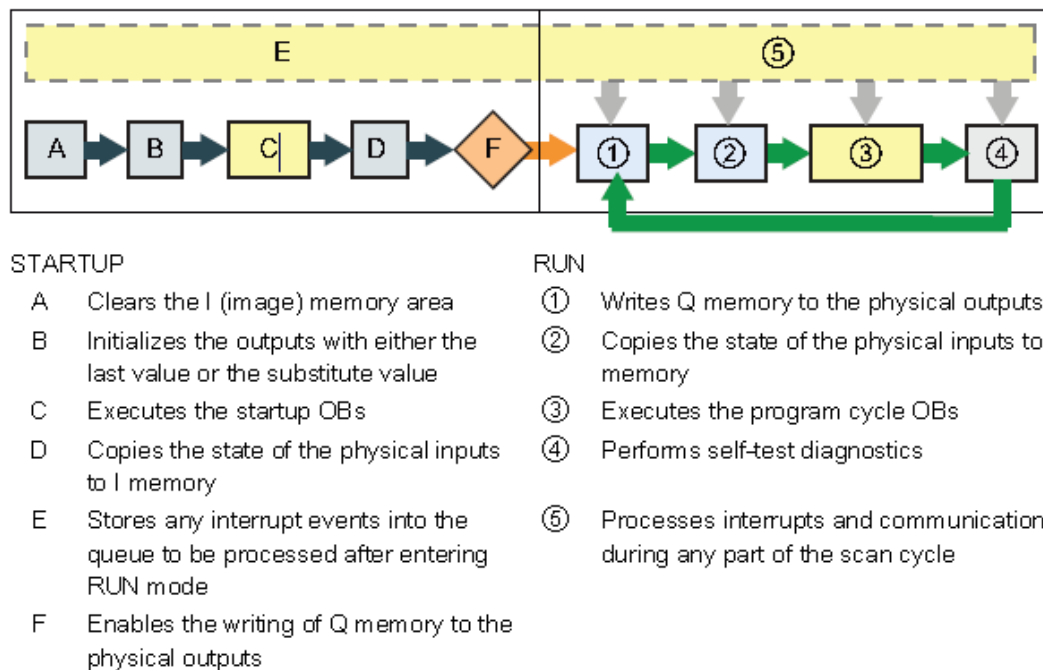
Fig. 18-3

Some A-B I/O has the ability to send the I/O status to multiple processors. This is referred to as an I/O Module Trigger to an event Task. We will not discuss this topic further here since the I/O is not present in the lab and we do not need this capability to process any information. However, do remember that the more sophisticated processors have this capability and it should be considered when planning a project.

Siemens

Siemens, like Allen-Bradley, has a sequence of events that occur when the processor starts up and then services the I/O and solves logic during the scan of the processor. It is roughly described in the following diagram:

“



”

Fig. 18-4

Operating modes of the CPU

The CPU has three modes of operation: STOP mode, STARTUP mode, and RUN mode.

“The system guarantees that the scan cycle will be completed in a time period called the maximum cycle time; otherwise a time error event is generated.

- Each scan cycle begins by retrieving the current values of the digital and analog outputs from the process image and then writing them to the physical outputs of the CPU, SB, and SM modules configured for automatic I/O update (default configuration). When a physical output is accessed by an instruction, both the output process image and the physical output itself are updated.
- The scan cycle continues by reading the current values of the digital and analog inputs from the CPU, SB, and SMs configured for automatic I/O update (default configuration), and then writing these values to the process image. When a physical input is accessed by

an instruction, the value of the physical input is accessed by the instruction, but the input process image is not updated.

- After reading the inputs, the user program is executed from the first instruction through the end instruction. This includes all the program cycle OBs plus all their associated FCs and FBs. The program cycle OBs are executed in order according to the OB number with the lowest OB number executing first.”

Interrupt latency

The interrupt event latency (the time from notification of the CPU that an event has occurred until the CPU begins execution of the first instruction in the OB that services the event) is approximately 175 µsec, provided that a program cycle OB is the only event service routine active at the time of the interrupt event.

Understanding time error events

The occurrence of any of several different time error conditions results in a time error event. The following time errors are supported:

- Maximum cycle time exceeded
- Requested OB cannot be started
- Queue overflow occurred

The maximum cycle time exceeded condition results if the program cycle does not complete within the specified maximum scan cycle time. See the section on "Monitoring the cycle time in the S7-1200 System Manual" (Page 80) for more information regarding the maximum cycle time condition, how to configure the maximum scan cycle time, and how to reset the cycle timer.

The requested OB cannot be started condition results if an OB is requested by a cyclic interrupt, a time-delay interrupt, or a time-of-day interrupt, but the requested OB is already being executed. The queue overflow occurred condition results if the interrupts are occurring faster than they can be processed. The number of pending (queued) events is limited using a different queue for each event type. If an event occurs when the corresponding queue is full, a time error event is generated. All time error events trigger the execution of OB 80 if it exists. If an OB 80 is not included in the user program, then the device configuration of the CPU determines the CPU reaction to the time error:

- The default configuration for time errors, such as starting a second cyclic interrupt before the CPU has finished the execution of the first, is for the CPU to stay in RUN.
- The default configuration for exceeding the maximum time is for the CPU to change to STOP.

Recording of measured values with the trace function:

The trace and logic analyzer function can be called in the device folder in the project navigator under the name “Traces”. You record device tags and evaluate the recordings with the trace and logic analyzer function. Tags are for example drive parameters or system and user tags of a CPU. The maximum recording duration is limited by the memory size. How much memory is available for the recording depends on the hardware used.

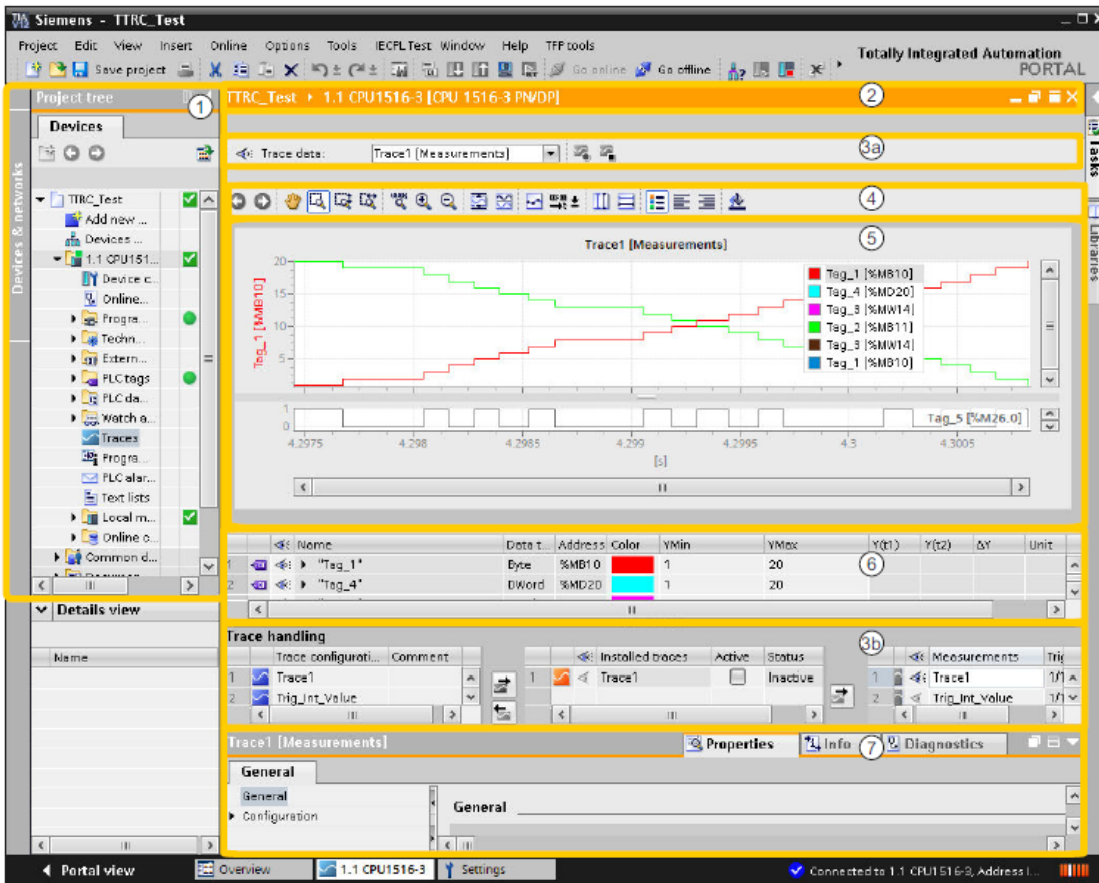


Fig. 18-5

While all this may seem rather bothersome, the exercise to make sure the scan has enough time and there is sufficient memory to accomplish the task is very important. It is too late if the project is nearing start-up to find out these problems. Then they are a catastrophe.

Timing issues can definitely stop a good project in its tracks. There are a number of software issues that can be hard to conquer but timing issues are one of the worst.

A Timed Interrupt Program

The following program is an example of a timed interrupt program saved in OB30 (Siemens). This program is found in Chapter 19 since the program uses the PID block. However, the use of a timed interrupt program is essential for proper execution of the program. Back in Chapter 10, a timed interrupt program was used to count calories. What was needed was a .1 second or .01 second interrupt that calculated the number of calories in the last 100 or 10 msec and added them to those already expended. The result was a graph of calories that grew with time and the load on the bike measured in watts from the light bulbs and by the ache in the legs.

The following program has a problem in that the program could be executed in any time range from .1 second to .001 second. It is primarily determined by the accuracy of the speed which is a function of the number of pulses received in the time period.

First, 10 msec was picked and found that the maximum number of pulses in the time period was 12, not enough to accurately control the PID algorithm. Who wants a speed control that can only be controlled to 1 part in 12? That is not a good outcome. However, if one extends the time period, we lose the ability to make quick changes in the function. The function is the control of a simple dc motor with feedback control through an encoder. The encoder was simply not accurate enough to allow an increase in the update speed of the program. So, for 100 msec time period, we would get about 120 pulses for an extremely low accuracy encoder.

The following figure shows the configuration of the system including the cyclic interrupt (OB30).

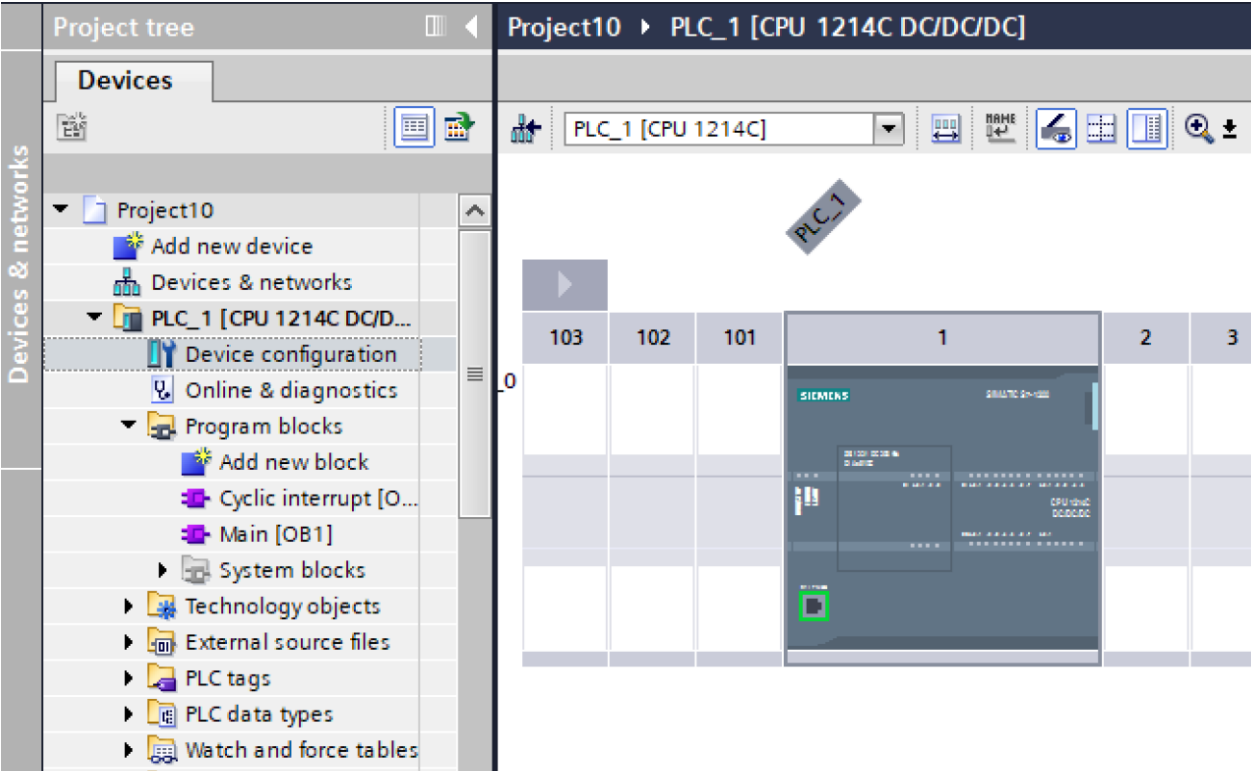


Fig. 18-6

The following figure describes the input waveform configuration. The input must be allowed to be read very fast – in this case, 20 microseconds.

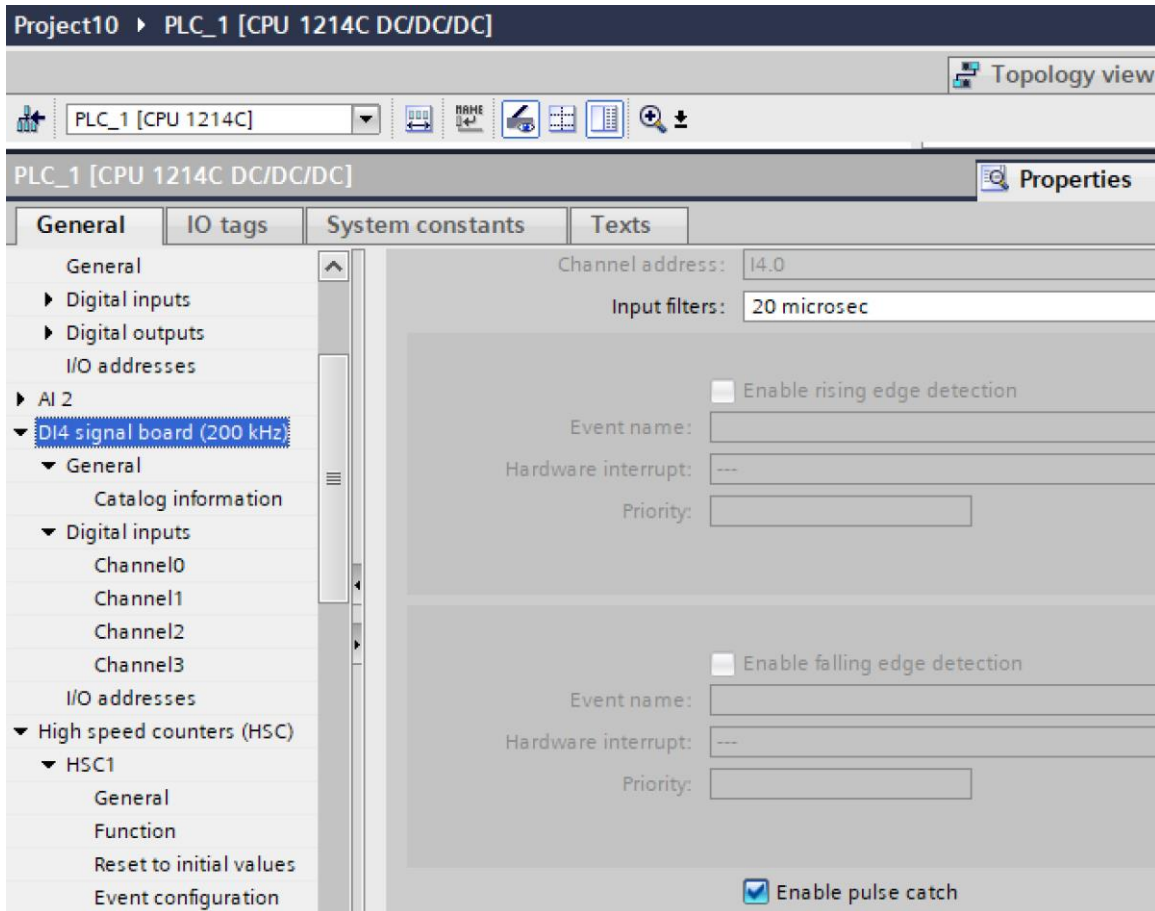


Fig. 18-7

The following describes the high-speed counter configuration. The input is HSC_1. Wiring terminal and address is found in the next figure:

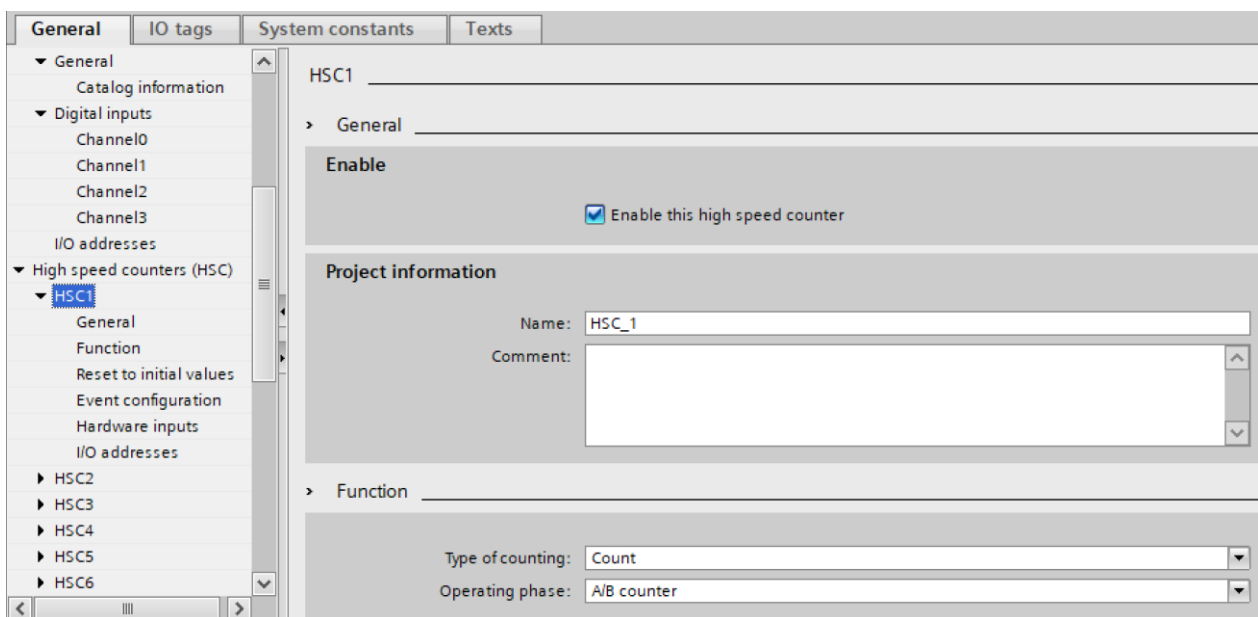


Fig. 18-8

The input to pulse A is wired to I4.1:

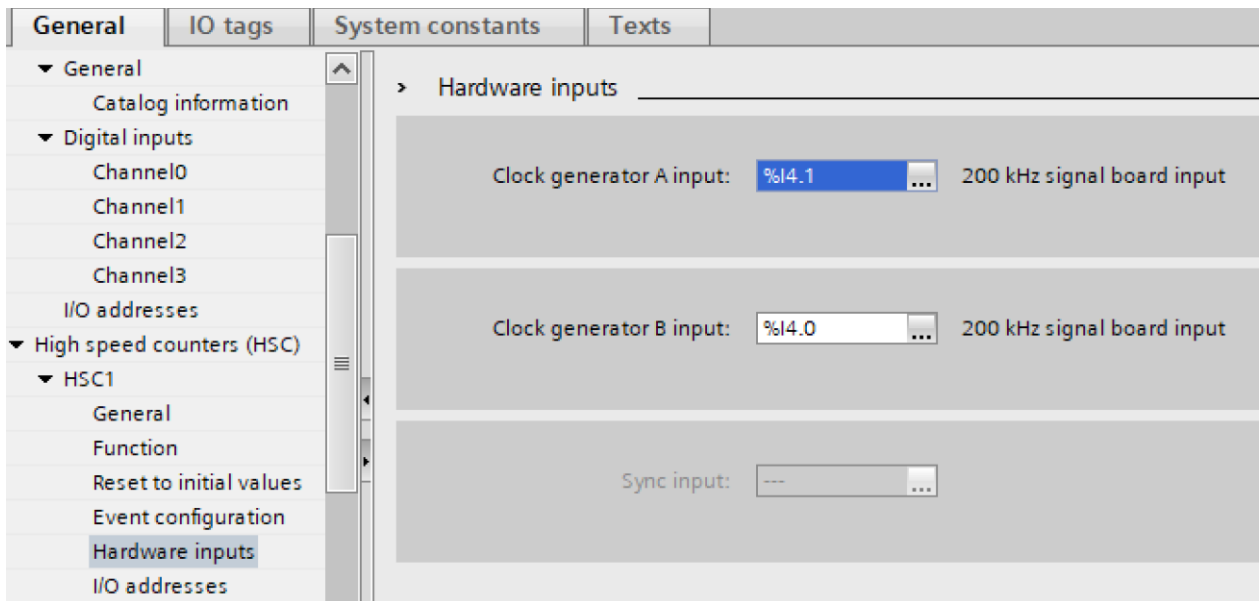


Fig. 18-9

The input is addressed in the program as ID1000. This address appears in the program statement in the logic below:

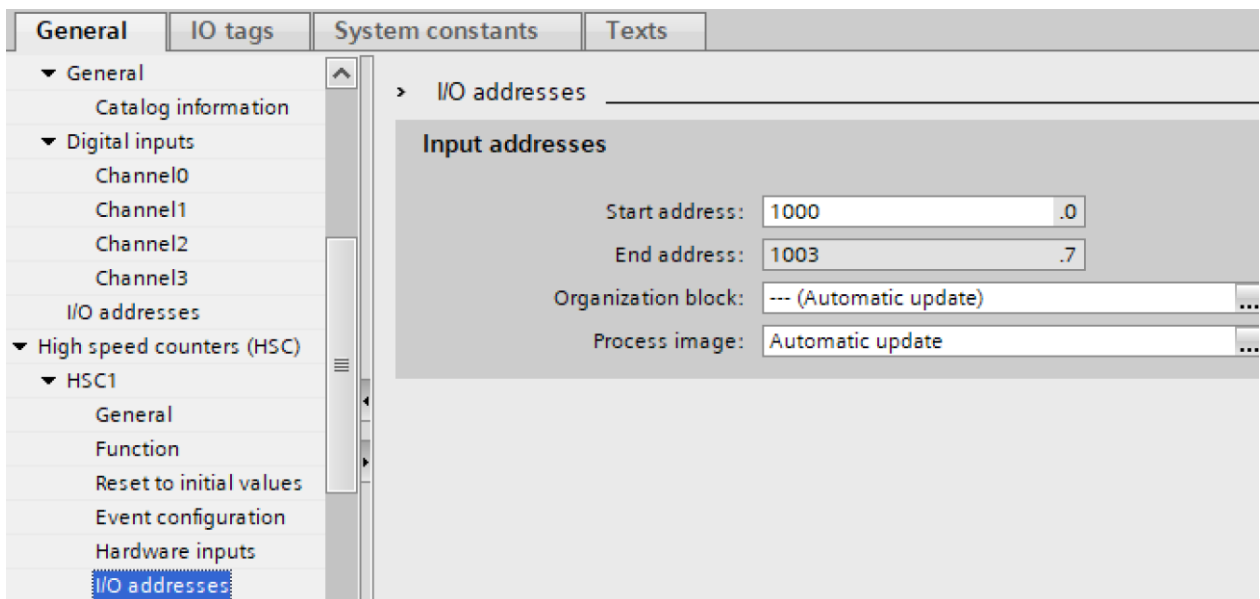


Fig. 18-10

The following program reads the input and determines the number of pulses since the last scan. In this case, the pulse count is the count in the last 100 msec.

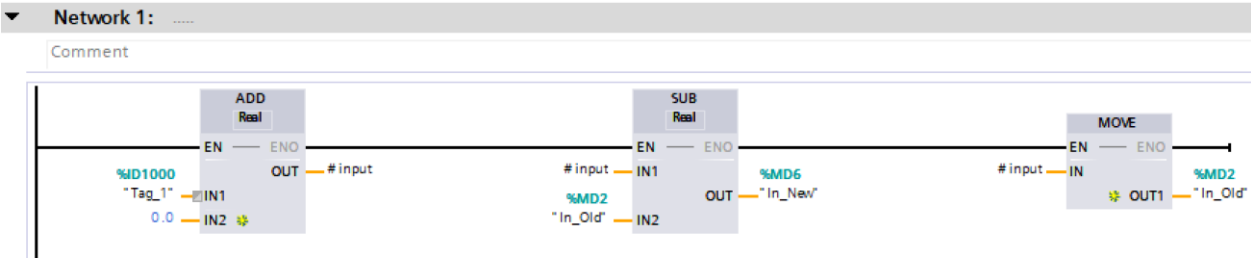


Fig. 18-11

A Problem Using Timed Interrupt Programming

For the process below, the roll conveyor carries boxes from left to right. If a box is too close to the one before it, the spacing bar comes up (arrow) to hold it back until a constant spacing is achieved. If a box is spaced in excess of the minimum, the box is allowed to pass on with no blocking. Write a program in ladder logic to control the blocking bar based on the photo-eye and an input from a pulse tachometer. Assume the pulse tach is a dint word with a constantly increasing number of pulses each time read.

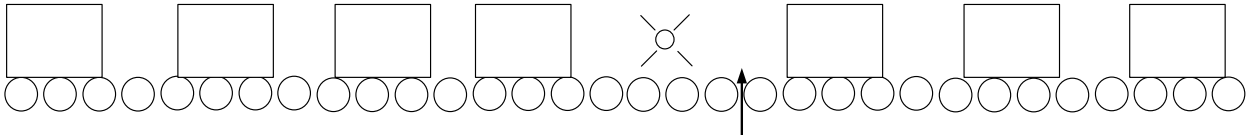


Fig. 18-12

This problem uses the pulse counter described in the program above to find the movement of the conveyor and the boxes per unit of time. The problem should be run about every 10 msec since the movement of the boxes probably will be able to move less than 1/4 inch in that time and the control of the spacing boxes would be adequate based on this 1/4 inch requirement.

The photo-eye would see the leading edge and the trailing edge of the previous box can be tracked. When the box's leading edge moves to the blocking bar, a decision is made whether to activate it. If activated, then the tracking continues until the distance from the last trailing edge is exceeded and the blocking bar is de-activated or blocked.

This problem is assigned in the Problems section.

Another Anecdotal:

In 1972, this author was placed on a project team that was responsible for the automatic tracking of glass on a cutting machine for a major glass manufacturer. The team leader was experienced and confident the project would proceed well. The other team member was experienced as well. I was the newbie.

The team leader in early 1973 quit. And the other team member, while willing to help, was relegated to the bench due to competition between plants. The new plant was purported to be non-union and the plant that he worked for was union. Thus, I was the last one standing.

The project was to track glass on 66 sections of conveyor. The prior job for which the team leader and other member had worked had only 6. They used interrupts for each pulse that came in to the computer which represented each 1.5 inches of travel of the conveyor. I looked at the interrupt light for this interrupt category and it was on pretty bright. What would happen if the number of interrupts increased by a factor of 10? I had no idea. And I was alone. So what did I do?

I changed the program to a scanned program that would execute each 8 msec. In this format, multiple pulse inputs would possibly change in each execution of the program but there would be only one interrupt each 8 msec. This change, while not monumental, was significant. I just implemented it. I did not ask. I just went ahead.

Later, my boss was discussing something along the lines of the portion of the program in question and I happened to mention that I had changed it to a scanned program. I thought I would be fired from the look on his face. I wasn't. He reviewed with me the change and went along with it.

The program worked. It collected data from all 66 conveyors and tracked, not at 1.5 inch accuracy but at .75 inch accuracy since I executed on both the rising and trailing edge of the signal. So, it was twice as accurate. The light was also not as intense as the light on the prior program. It seemed to run 'cooler'.

I don't know if this change 'saved' the project. I don't care. I just know that it worked the way I programmed it.

And, I became a firm believer in the expression "it is better to ask for forgiveness than to ask for permission." You see, I doubt if my boss would have gone along with a 'theory' that the program would fall apart once moved to 66 conveyors using interrupts. It would have been hard to prove one way or the other and he would have reverted to the way it had previously been done. I was responsible for the project and was conservative in the use of time. Neither could prove absolutely the right way to go. It worked the new way and, in the process, saved money on the future purchases of computers used in the work in that the original computer cost about \$100K and the ones that could now be purchased would cost only about \$30K, a savings of \$70K. Not bad.

Finally

The following table is found in the S7-1200 Programmable Controller System Manual. The table gives execution times for instructions in the PLC. It will be used in a problem at the end of the chapter. It is useful to calculate the execution time for your programs. Use it!

Table A- 49 Performance

Type of instruction		Execution speed	
		Direct addressing (I, Q and M)	DB accesses
Boolean		0.08 μ s/instruction	
Move	Move_Bool	0.3 μ s/instruction	1.17 μ s/instruction
	Move_Word	0.137 μ s/instruction	1.0 μ s/instruction
	Move_Real	0.72 μ s/instruction	1.0 μ s/instruction
Real Math	Add Real	1.48 μ s/instruction	1.78 μ s/instruction

Note

Many variables affect measured times. The above performance times are for the fastest instructions in this category and error-free programs.

The following pages are from the Siemens Text:
Programming Guideline for S7-1200/1500 Entry ID: 81318674, V1.6, 12/2018

They summarize changes and upgrades to the Siemens Portal Language from Version 14 – TIA and later:

“

2.14 STOP mode in the event of errors

In comparison to S7-300/400 there are fewer criteria with the S7-1200/1500 that lead to the "STOP" mode.

Due to the changed consistency check in the TIA Portal, the "STOP" mode for S7-1200/1500 controllers can already be excluded in advance in most cases. The consistency of program blocks is already checked when compiling in the TIA Portal. This approach makes the S7-1200/1500 controllers more "fault tolerant" to errors than their predecessors.

Advantages

There are only three fault situations that put the S7-1200/1500 controllers into the STOP mode. This makes the programming of the error management clearer and easier.

Properties

Table 2-18: Response to errors of S7-1200/1500

	Error	S7-1200	S7-1500
1.	Cycle monitoring time exceeded once	RUN	STOP (when OB80 is not configured)
2.	Cycle monitoring time exceeded twice	STOP	STOP
3.	Programming error	RUN	STOP (when OB121 is not configured)

Error OBs:

- OB80 "Time error interrupt" is called by the operating system when the maximum cycle time of the controller is exceeded.
- OB121 "Programming error" is called by the operating system when an error occurs during program execution.

For every error, in addition, an entry is automatically created in the diagnostic buffer.

Note

For S7-1200/1500 controllers there are other programmable error OBs (diagnostic error, module rack failure, etc.).

More information on error responses of S7-1200/1500 can be found in the online help of the TIA Portal under "Events and OBs".

2.4 Optimized machine code

TIA Portal and S7-1200/1500 enable an optimized runtime performance in every programming language. All languages are compiled directly in machine code in the same way.

Advantages

- All programming languages have the same level of performance (for the same access types)
- No reduction of performance through additional compilation with interim step via STL

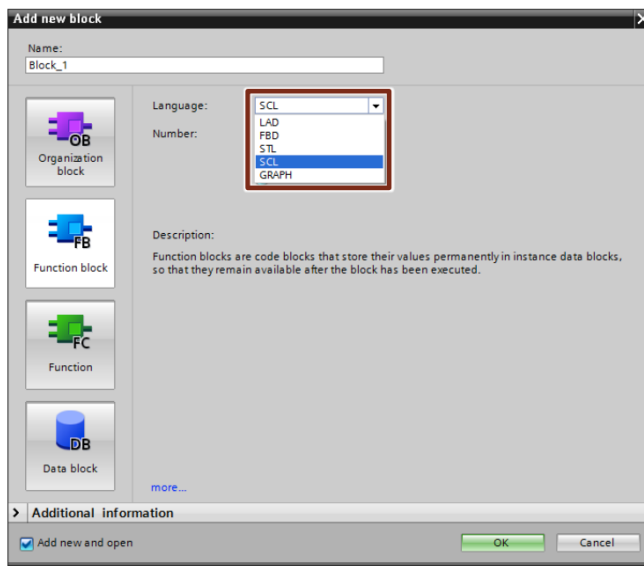
Properties

In the following figure, the difference in the compilation of S7-programs in machine code is displayed.

2.5 Block creation

All blocks such as OBs, FBs and FCs can be directly programmed in the desired programming language. Therefore no source has to be created for SCL programming. Only select the block and SCL as programming language. You can then program the block directly.

Figure 2-4: Dialog "Add new Block"



2.6 Optimized blocks

S7-1200/1500 controllers have an optimized data storage. In optimized blocks all tags are automatically sorted according to their data type. The sorting ensures that data gaps between the tags are reduced to a minimum and that the tags are stored access-optimized for the processor.

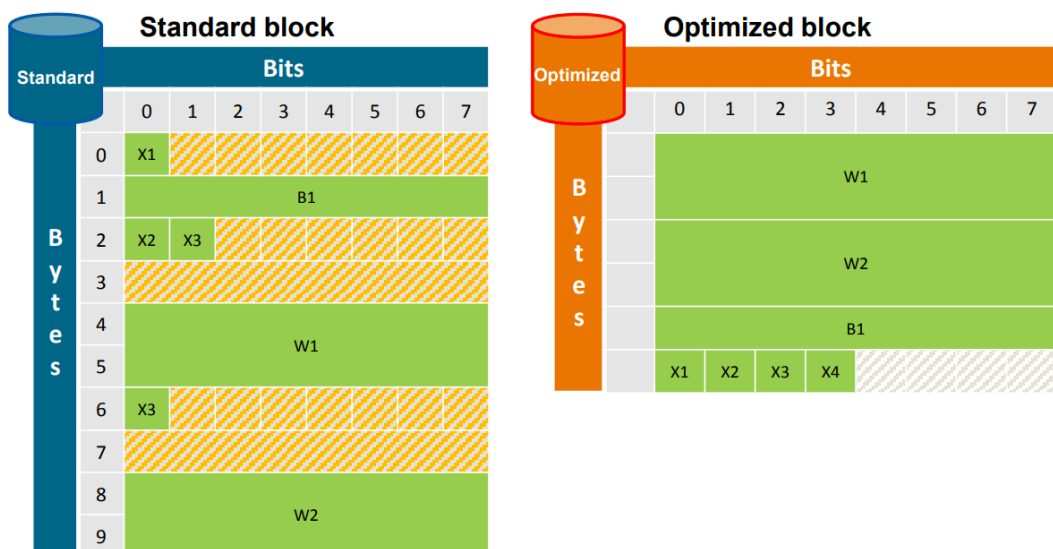
Non-optimized blocks are only available for compatibility reasons in S7-1200/1500 controllers.

Advantages

- Access always takes place as quickly as possible since the data storage is optimized by the system and independent of the declaration.
- No danger of inconsistencies due to faulty, absolute access, since access is generally symbolic
- Declaration changes do not lead to access errors since, for example, HMI access is symbolic.
- Individual tags can be specifically defined as retentive.
- No settings required in the instance data block. Everything is set in the assigned FB (for example, retentivity).
- Storage reserves in the data block enables changes without loss of current values (see chapter [3.2.8 Downloading without reinitialisation](#)).

2.6.1 S7-1200: Structure of optimized blocks

Figure 2-5: Optimized blocks for S7-1200



Properties

- No data gaps are formed since larger tags are located at the start of a block and smaller ones at the end.
- There is only symbolic access for optimized blocks.


Example: Setting optimized block access

By default, the optimized block access is enabled for all newly created blocks for S7-1200/1500. Block access can be set for OBs, FBs and global DBs. For instance DBs, the setting derives from the respective FB.

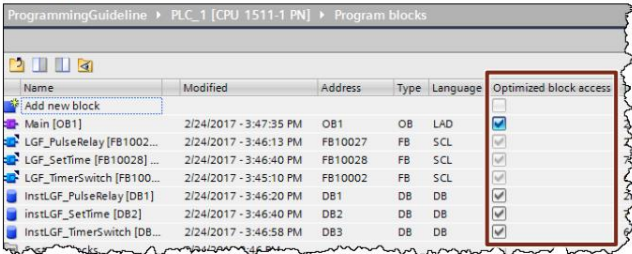
Block access is not automatically reset if a block is migrated from a S7-300/400 controller to a S7-1200/1500. You can later change the block access to "Optimized block access". After changing the block access, you have to recompile the program. If you change FBs to "Optimized block access", the assigned instance data blocks are automatically updated.

Follow the instructions to set the optimized block access.

Table 2-4: Setting optimized block access

Step	Instruction
1.	<p>Click the "Maximizes/minimizes the Overview" button in the project tree.</p> 
2.	<p>Navigate to "Program blocks".</p>

2.6 Optimized blocks

Step	Instruction
3.	<p>Here, you see all blocks in the program and whether they are optimized or not. In this overview the status "Optimized block access" can be conveniently changed.</p>  <p>Note: Instance data blocks (here "Function_block_1_DB") inherit the status "optimized" from the associated FB. This is why the "optimized" setting can only be changed on the FB. After the compilation of the project, the DB takes on the status depending on the associated FB.</p>

Display of optimized and non-optimized blocks in the TIA Portal

In the two following figures the differences between an optimized and a non-optimized instance DB can be seen.

For a global DB there are the same differences.

Figure 2-9: optimized data block (without offset)

InstLGF_PulseRelay			
	Name	Data type	Start value
1	▼ Input		
2	trigger	Bool	false
3	set	Bool	false
4	reset	Bool	false
5	▼ Output		
6	out	Bool	false

Figure 2-10: non-optimized data block (with offset)

InstLGF_PulseRelay				
	Name	Data type	Offset	Start value
1	▼ Input			
2	trigger	Bool	0.0	false
3	set	Bool	0.1	false
4	reset	Bool	0.2	false
5	▼ Output			
6	out	Bool	2.0	false

Table 2-5: Difference: Optimized and non-optimized data block

Optimized data block	Non-optimized data block
Optimized data blocks are addressed symbolically. Therefore no "offset" is shown.	For non-optimized blocks the "offset" is shown and can be used for addressing.
In the optimized block you can declare each tag individually with "Retain".	In non-optimized blocks only all or no tag can be declared with "Retain".

2.6 Optimized blocks

The retentivity of tags of a global DB is directly defined in the global DB. By default, non-retain is preset.

Define the retentivity of tags in an instance in the function block (not the instance DB). These settings are therefore valid for all instances of this FB.

Access types for optimized and non-optimized data blocks

In the following table all access types for blocks are displayed.

Table 2-6: Access types

Access type	Optimized block	Non-optimized block
Symbolic	yes	yes
Indexed (fields)	yes	yes
Slice access	yes	yes
AT instruction	no (Alternative: slice access)	yes
Direct absolute	no (Alternative: ARRAY with INDEX)	yes
Indirect absolute (pointer)	no (Alternative: VARIANT / ARRAY with index)	yes
Load without reinitialization	yes	no

2.6.4 Conversion between optimized and non-optimized tags

It is generally recommended to work with optimized tags. However, if in individual cases, you want to keep your programming so far, there will be a mix of optimized and non-optimized data storage in the program.

The system knows the internal storage of each tag, irrelevant whether structured (derived from an individually defined data type) or elementary (INT, LREAL, ...).

For assignments with the same type between two tags with different memory storage, the system converts automatically. This conversion requires performance for structured tags and should therefore be avoided, if possible.

2.6.5 Parameter transfer between blocks with optimized and non-optimized access

When you transfer structures to the called block as in/out parameters (InOut), they are transferred by default as reference (see [chapter 3.3.2 Call-by-reference](#)).

However, this is not the case if one of the blocks has the property "Optimized access" and the other block the property "Default access". In this case, all parameters are generally transferred as copy (see [chapter 3.3.1 Call-by-value](#)).

In this case the called block always works with the copied values. During block processing, these values may be changed and they are copied back to the original operand, after processing of the block call.

This may cause problems if the original operands are changed by asynchronous processes, for example, by HMI access or interrupt OBs. If the copies are copied back to the original operands after the block processing, the asynchronously performed changes on the original operands are overwritten.

Recommendation

- Always set the same access type for the two blocks that communicate with each other.

2.7 Block properties

2.7.1 Block sizes

For S7-1200/1500 controllers the maximum size of blocks in the main memory was noticeably enlarged.

Table 2-7: Block sizes

Max. size and number (without consideration of memory size)		S7-300/400	S7-1200	S7-1500
DB	Max. size	64 kB	64 kB	64 kB 16 MB (optimized CPU1518)
	Max. number	16.000	65.535	65.535
FC / FB	Max. size	64 kB	64 kB	512 kB
	Max. number	7.999	65.535	65.535
FC / FB / DB	Max. number	4.096 (CPU319) 6.000 (CPU412)	1.024	10.000 (CPU1518)

Recommendation

- Use DBs for S7-1500 controllers as data container of very large data volumes.
- You can store data volumes of > 64 kB with S7-1500 controllers in an optimized DB (max. size 16 MB).

2.7.2 Number of organization blocks (OB)

With OBs a hierarchical structure of the user program can be created. There are different OBs available for this.

Table 2-8: Number of organization blocks

Organization block type	S7-1200	S7-1500	Benefits
Cyclic and startup OBs	100	100	Modularization of the user program
Hardware interrupts	50	50	Separate OB for each event possible
Delay interrupts	4 *	20	Modularization of the user program
Cyclic interrupts		20	Modularization of the user program
Clocked interrupts	no	20	Modularization of the user program

* As of firmware V4, 4 delay interrupts and 4 cyclic interrupts are possible.

Recommendation

- Use OBs in order to structure the user program hierarchically.
- Further recommendations for the use of OBs can be found in chapter [3.2.1 Organization blocks \(OB\)](#).

2.7.3 Block interface – hide block parameters (V14 or higher)

When calling the block, block parameters can be specifically displayed or hidden. Here, you have three options that you can configure individually for each formal parameter.

- "Show"
- "Hide"
- "Hide if no parameter is assigned"

Advantages

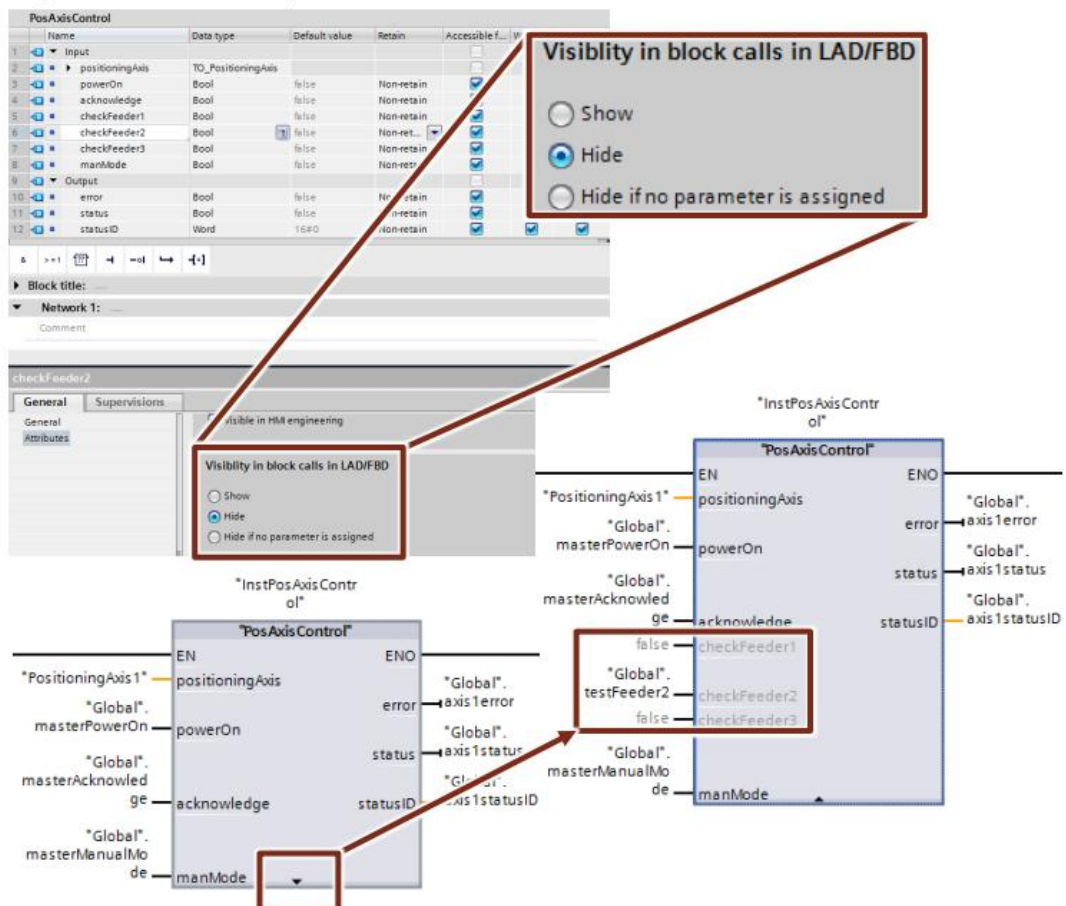
- Better overview for blocks with many optional parameters

Properties

- Can be used for:
 - FCs, FBs
 - In, Out, InOut

Example

Figure 2-12: Hide block parameters



2.8 New data types for S7-1200/1500

S7-1200/1500 controllers support new data types to make programming more convenient. With the new 64 bit data types, considerably larger and more precise values can be used.

Note

More information can be found in the following entry:

In STEP 7 (TIA Portal), how do you convert the data types for the S7-1200/1500?

<https://support.industry.siemens.com/cs/ww/en/view/48711306>

2.8.1 Elementary data types

Table 2-9: Integer data types

Type	Size	Value range
USint	8 bit	0 .. 255
SInt	8 bit	-128 .. 127
UInt	16 bit	0 .. 65535
UDInt	32 bit	0 .. 4.3 Mio
ULInt*	64 bit	0 .. 18.4 Trio (10^{18})
LInt*	64 bit	-9.2 Trio .. 9.2 Trio
LWord	64 bit	16#0000 0000 0000 0000 to 16# FFFF FFFF FFFF FFFF

* only for S7-1500

Table 2-10: Floating-point data types

Type	Size	Value range
Real	32 bit (1 bit prefix, 8 bit exponent, 23 bit mantissa), precision 7 places after the comma	-3.40e+38 .. 3.40e+38
LReal	64 bit (1 bit prefix, 11 bit exponent, 52 bit mantissa), precision 15 places after the comma	-1.79e+308 .. 1.79e+308

Note

More information can be found in the following entries:

Why, in STEP 7 (TIA Portal), is the result of the DInt Addition in SCL not displayed correctly?

<https://support.industry.siemens.com/cs/ww/en/view/98278626>

2.8.2 Data type Date_Time_Long

Table 2-11: Structure of DTL (Date_Time_Long)

Year	Month	Day	Weekday	Hour	Minute	Second	Nanosecond
------	-------	-----	---------	------	--------	--------	------------

DTL always reads the current system time. Access to the individual values is by the symbolic names (for example, `My_Timestamp.Hour`)

Advantages

- All subareas (for example, Year, Month, ...) can be addressed symbolically.

Recommendation

Use the new data type DTL instead of LDT and address it symbolically (for example `My_Timestamp.Hour`).

Note

More information can be found in the following entries:

In STEP 7 (TIA Portal), how can you input, read out and edit the date and time for the CPU modules of S7-300/S7-400/S7-1200/S7-1500?

<https://support.industry.siemens.com/cs/ww/en/view/43566349>

Which functions are available in STEP 7 V5.5 and in TIA Portal for processing the data types DT and DTL?

<https://support.industry.siemens.com/cs/ww/en/view/63900229>

2.8.3 Other time data types

Table 2-12: Time data types (only S7-1500)

Type	Size	Value range
LTime	64 Bit	LT#-106751d23h47m16s854ms775us808ns to LT#+106751d23h47m16s854ms775us807ns
LTIME_OF_DAY	64 Bit	LTOD#00:00:00.000000000 to LTOD#23:59:59.999999999

2.8.4 Unicode data types

With the help of the data types WCHAR and WSTRING Unicode characters can be processed.

Table 2-13: Time data types (only S7-1500)

Type	Size	Value range
WCHAR	2 Byte	-
WSTRING	(4 + 2*n) Byte	Preset value: 0 ..254 characters Max. Value: 0 ..16382

n = length of string

Properties

- Processing of characters in, for example, Latin, Chinese or other languages.
- Line breaks, form feed, tab, spaces
- Special characters: Dollar signs, quotes

Example

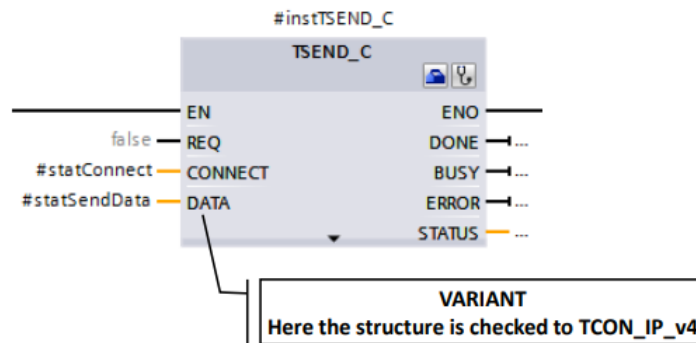
- `WCHAR# 'a '`
- `WSTRING# 'Hello World! '`

2.8.5 Data type VARIANT (S7-1500 and S7-1200 from FW4.1)

A parameter from the type VARIANT is a pointer that can point to tag of different data types. In contrast to the ANY pointer, VARIANT is a pointer with type test. This means that the target structure and source structure are checked at runtime and have to be identical.

VARIANT, for example, is used for communication blocks (TSEND_C) as input.

Figure 2-13: Data type VARIANT as input parameters for instruction TSEND_C



Advantages

- Integrated type test prevents faulty access.
- The code can be more easily read through the symbolic addressing of the variant tags.
- Code is more efficiently and within a shorter time.
- Variant pointers are clearly more intuitive than ANY pointers.
- The right type of variant tags can be used directly with the help of system functions.
- Flexible and performant transfer of different structured tags is possible.

Properties

In a comparison between ANY and variant, the properties can be seen.

Table 2-14: Comparison ANY and variant

ANY	Variant
Requires 10 byte memory with defined structure	Does not require a main memory for the user
Initialization either via assignment of the data area or by filling the ANY structure	Initialization by assigning the data area or system instruction
Non-typed – type of an interconnected structure cannot be recognized	Typed – interconnected type and for arrays the length can also be determined
Partly typed – for arrays the length can also be determined	VARIANT can be evaluated and also created via system instructions

Recommendation

- Check where before you had to use the ANY pointer. In many cases a pointer is no longer necessary (see following table).
- Use the data type VARIANT only for indirect addressing when the data types are only determined at program runtime.
 - Use the data type VARIANT as InOut formal parameter to create generic blocks that are independent from the data type of the actual parameters (see example in this chapter).
 - Use the VARIANT data type instead of the ANY pointer. Errors are detected early on due to the integrated type test. Due to the symbolic addressing, the program code can be easily interpreted.
 - Use the variant instruction, for example, for type identification (see following example and chapter [2.9.2 VARIANT instructions](#).)
- Use the index for arrays instead of addressing the array elements via ANY (see chapter [3.6.2 ARRAY data type and indirect field accesses](#)).

Table 2-15: Comparison ANY pointer and simplifications

What are ANY pointers used for?		Simplification with S7-1200/1500
Programming functions that can process different data types	→	Functions with variant pointer as InOut parameter for blocks (see following example)
Processing of arrays <ul style="list-style-type: none"> • for example, reading, initializing, copying of elements of the same type 	→	Default array functions <ul style="list-style-type: none"> • Reading and writing with #myArray[#index] (see chapter 3.6.2 ARRAY data type and indirect field accesses) • Copying with MOVE_BLK (see chapter 2.9.1 MOVE instructions)
<ul style="list-style-type: none"> • Transferring structures and performant processing via absolute addressing for example, transferring user-defined structures via ANY pointer to functions 	→	Transferring structures as InOut parameters <ul style="list-style-type: none"> • see chapter 3.3.2 Call-by-reference

Note

If values of non-structured VARIANT tags are to be copied, you can also use VariantGet instead of MOVE_BLK_VARIANT (chapter [2.9.2 VARIANT instructions](#)).

Example

With the data type VARIANT it is possible to identify data types in the user program and to respond to them accordingly. The following code of the FCs "MoveVariant" shows a possible programming.

- The InOut formal parameter "InVar" (data type VARIANT) is used to show a tag independent from the data type.
- The data type of the actual parameter is detected with the "Type_Of" instruction
- Depending on the data type, the tag value is copied with the "MOVE_BLK_VARIANT" instruction to the different output formal parameters.
- If the data type of the actual parameter is not detected, the block will output an error code.

Figure 2-14: Formal parameter of the FC "MoveVariant"

MoveVariant			
	Name	Data type	Default value
1	▶ Input		
2	▼ Output		
3	▶ outInteger	Int	
4	▶ outReal	Real	
5	▶ outTypeCustom	"typeCustom"	
6	▼ InOut		
7	▶ inOutVariant	Variant	
8	▶ Temp		
9	▶ Constant		
10	▶ Return		

```

CASE TypeOf(#inOutVariant) OF // Check datatypes
  Int: // Move Integer
    #MoveVariant := MOVE_BLK_VARIANT(SRC := #inOutVariant,
                                     COUNT := 1,
                                     SRC_INDEX := 0,
                                     DEST_INDEX := 0,
                                     DEST => #outInteger);

  Real: // Move Real
    #MoveVariant := MOVE_BLK_VARIANT(SRC := #inOutVariant,
                                     COUNT := 1,
                                     SRC_INDEX := 0,
                                     DEST_INDEX := 0,
                                     DEST => #outReal);

  typeCustom: // Move outTypeCustom
    #MoveVariant := MOVE_BLK_VARIANT(SRC := #inOutVariant,
                                     COUNT := 1,
                                     SRC_INDEX := 0,
                                     DEST_INDEX := 0,
                                     DEST => #outTypeCustom);

ELSE // Error, no sufficient datatype
  #MoveVariant := WORD_TO_INT(#NO_CORRECT_DATA_TYPE);
  // 80B4: Error-Code of MOVE_BLK_VARIANT: Data types do
  not correspond
END_CASE;

```

2.9.3 RUNTIME

The "RUNTIME" instruction measures the runtime of the entire program, individual blocks or command sequences. You can call this instruction in LAD, FBD, SCL and in STL (only S7-1500).

Note More information can be found in the following entry:

With S7-1200/S7-1500, how do you measure the total cycle time of an organization block?

<https://support.industry.siemens.com/cs/ww/en/view/87668055>

3.7 Libraries

With the TIA Portal you can establish independent libraries from different project elements that can be easily reused.

Advantages

- Simple storage for the data configured in the TIA Portal:
 - Complete devices (controller, HMI, drive, etc.)
 - Blocks, tag tables, PLC data types, watch tables, etc.
 - HMI screens, HMI tags, scripts, etc.
- Cross-project exchange via libraries
- Central update function of library elements
- Versioning of library elements
- Fewer error sources when using control blocks through system-supported consideration of dependencies

Recommendations

- Create the master copies for easy reusability of blocks, hardware configurations, HMI screens, etc.
- Create the types for the system-supported reusability of library elements:
 - Versioning of blocks
 - Central update function of all usage locations
- Use the global library for the exchange with other users or as central storage for the simultaneous use of several users.
- Configure the storage location of your global library so it can automatically be opened when starting the TIA Portal.
Further information is available at:
<https://support.industry.siemens.com/cs/ww/en/view/100451450>

Note More information can be found in the following entries:

Which elements of STEP 7 (TIA Portal) and WinCC (TIA Portal) can you store in a library as Type or as Master Copy?

<https://support.industry.siemens.com/cs/ww/en/view/109476862>

How can you open a global library with write access rights in STEP 7 (TIA Portal)?

<https://support.industry.siemens.com/cs/ww/en/view/37364723>

3.7.1 Types of libraries and library elements

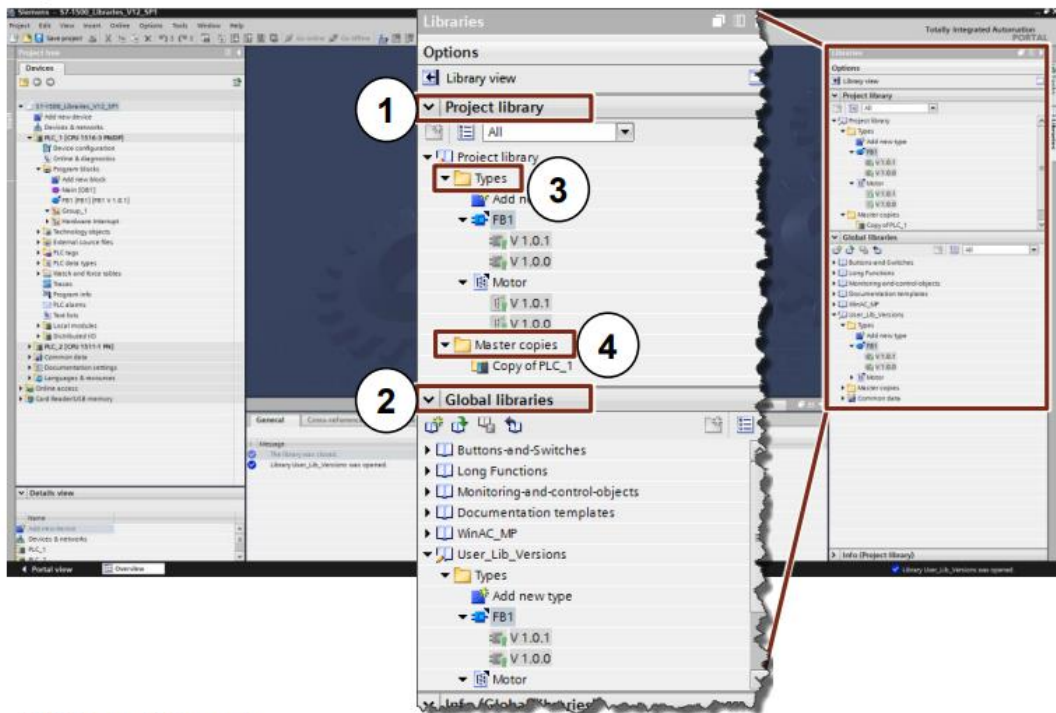
Generally there are two different types of libraries:

- "Project library"
- "Global library"

The content consists of two storage types each:

- "Types"
- "Master Copies"

Figure 3-38: Libraries in the TIA Portal



(1) "Project library"

- Integrated in the project and managed with the project
- Allows the reusability within the project

(2) "Global library"

- Independent library
- Use within several projects possible

A library includes two different types of storage of library elements:

(3) "Master copies"

- Copies of configuration elements in the library (e.g. blocks, hardware, PLC tag tables, etc.)
- Copies are not connected with the elements in the project.
- Master copies can also be made of several configuration elements.

(4) "Types"

- Types are connected with your usage locations in the project. When types are changed, the usage locations in the project can be updated automatically.

3.7 Libraries

- Supported types are control blocks (FCs, FBs), PLC data types, HMI screens, HMI faceplates, HMI UDT, scripts).
- Subordinate elements are automatically typified.
- Types are versioned: Changes can be made by creating a newer version.
- There can only be one version of a used type within a controller.

3.7.2 Type concept

The type concept allows the creation of standardized automation functions that you can use in several plants or machines. The type concept supports you with versioning and updating functions.

You can use types from the library in the user program. This offers the following advantages:

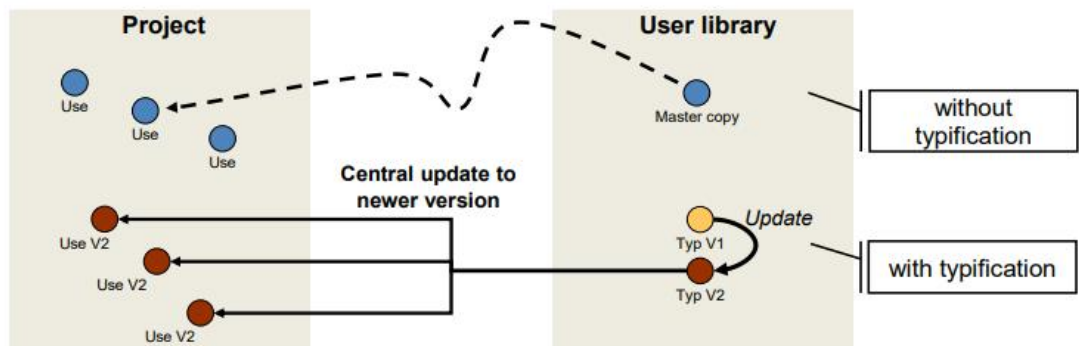
Advantages

- Central update of all usage locations in the project
- Unwanted modifications of usage locations of types are not possible.
- The system guarantees that types always remain consistent by hindering unwanted delete operations.
- If a type is deleted, all usage locations in the user program are deleted.

Properties

By using types you can make the changes centrally and update them in the complete project.

Figure 3-39: Typifying with user libraries



- Types are always marked for better identification

3.7.3 Differences between the typifiable objects for CPU and HMI

There are system-related differences between the typifiable objects for controllers and HMI:

Table 3-9: Differences of types for controller and HMI

Controller	HMI
Subordinate control elements are typified.	Subordinate HMI elements are not typified.
Subordinate control elements are instanced.	Subordinate HMI elements are not instanced.
Control elements are edited in a test environment .	HMI images and HMI scripts are edited in a test environment. Faceplates and HMI - UDTs are directly edited in the library without test environment .

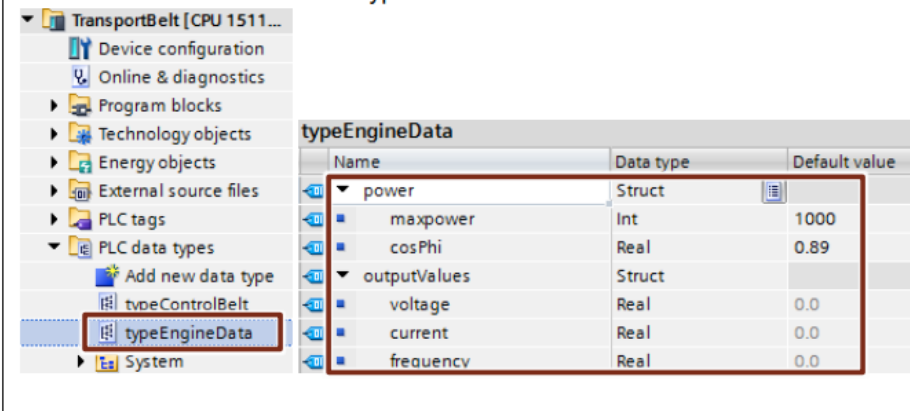
Further information on the handling of libraries can be found in the following example.

3.7.4 Versioning of a block

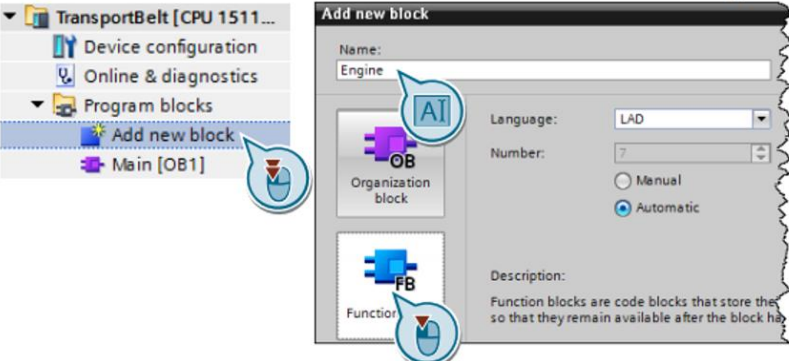
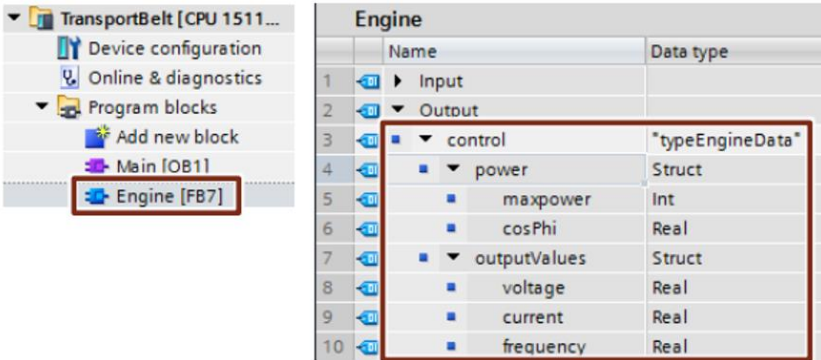
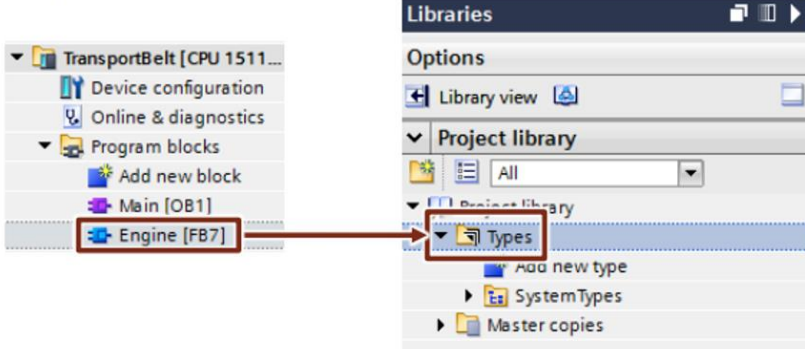
Example: Creating a type

The following example shows you how the basic functions of the libraries are used with types.

Table 3-10: Creating a type

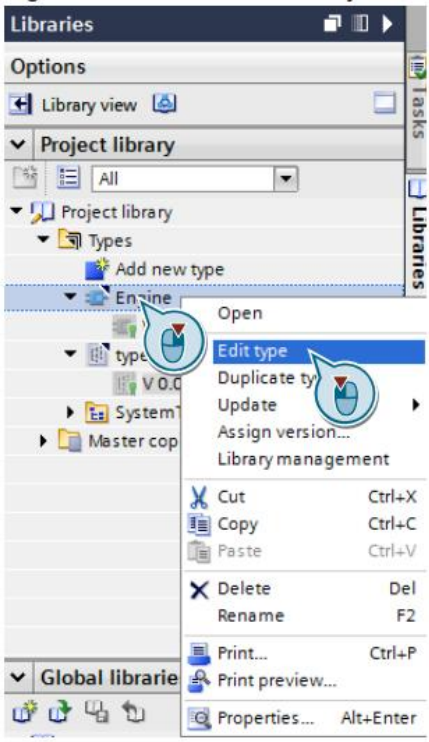

Step	Instruction																								
1.	<p>Create a new PLC data type with "Add new data type" and create some tags. Later on this is the subordinate type.</p>  <p>The screenshot shows the SIMATIC Manager interface. On the left, the project tree is expanded to 'PLC data types', where 'typeEngineData' is highlighted. On the right, the 'typeEngineData' structure is displayed in a table:</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Data type</th> <th>Default value</th> </tr> </thead> <tbody> <tr> <td>power</td> <td>Struct</td> <td></td> </tr> <tr> <td> maxpower</td> <td>Int</td> <td>1000</td> </tr> <tr> <td> cosPhi</td> <td>Real</td> <td>0.89</td> </tr> <tr> <td> outputValues</td> <td>Struct</td> <td></td> </tr> <tr> <td> voltage</td> <td>Real</td> <td>0.0</td> </tr> <tr> <td> current</td> <td>Real</td> <td>0.0</td> </tr> <tr> <td> frequency</td> <td>Real</td> <td>0.0</td> </tr> </tbody> </table>	Name	Data type	Default value	power	Struct		maxpower	Int	1000	cosPhi	Real	0.89	outputValues	Struct		voltage	Real	0.0	current	Real	0.0	frequency	Real	0.0
Name	Data type	Default value																							
power	Struct																								
maxpower	Int	1000																							
cosPhi	Real	0.89																							
outputValues	Struct																								
voltage	Real	0.0																							
current	Real	0.0																							
frequency	Real	0.0																							

3.7 Libraries

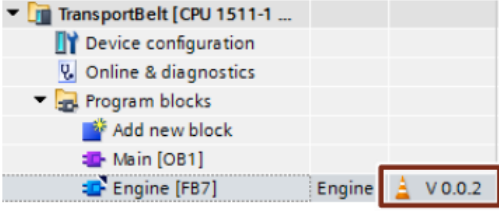
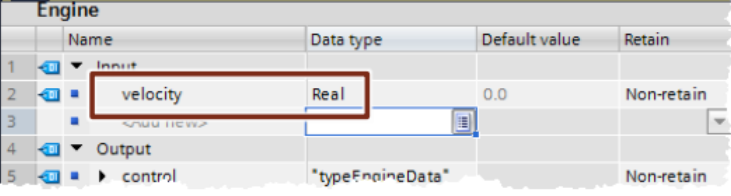

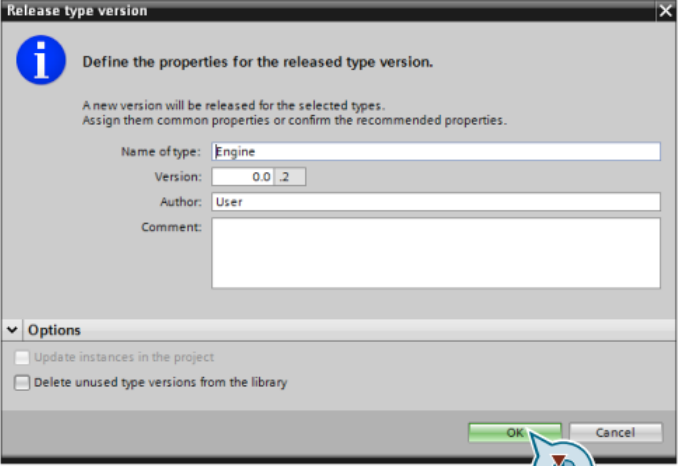
Step	Instruction																																	
2.	<p>Create a new function block with "Add new Block". This is the higher-level type.</p> 																																	
3.	<p>Define an output tag of the data type you have created. The PLC data type is therefore subordinate to the function block.</p>  <table border="1" data-bbox="805 806 1316 1164"> <thead> <tr> <th></th> <th>Name</th> <th>Data type</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Input</td> <td></td> </tr> <tr> <td>2</td> <td>Output</td> <td></td> </tr> <tr> <td>3</td> <td>control</td> <td>"typeEngineData"</td> </tr> <tr> <td>4</td> <td> power</td> <td>Struct</td> </tr> <tr> <td>5</td> <td> maxpower</td> <td>Int</td> </tr> <tr> <td>6</td> <td> cosPhi</td> <td>Real</td> </tr> <tr> <td>7</td> <td> outputValues</td> <td>Struct</td> </tr> <tr> <td>8</td> <td> voltage</td> <td>Real</td> </tr> <tr> <td>9</td> <td> current</td> <td>Real</td> </tr> <tr> <td>10</td> <td> frequency</td> <td>Real</td> </tr> </tbody> </table>		Name	Data type	1	Input		2	Output		3	control	"typeEngineData"	4	power	Struct	5	maxpower	Int	6	cosPhi	Real	7	outputValues	Struct	8	voltage	Real	9	current	Real	10	frequency	Real
	Name	Data type																																
1	Input																																	
2	Output																																	
3	control	"typeEngineData"																																
4	power	Struct																																
5	maxpower	Int																																
6	cosPhi	Real																																
7	outputValues	Struct																																
8	voltage	Real																																
9	current	Real																																
10	frequency	Real																																
4.	<p>Drag the function block via drag-and-drop into the "Types" folder in the project library.</p> 																																	

Example: Changing a type

Table 3-11: Changing a type

Step	Instruction
1.	<p>Right-click the block in the "Project library" and select "Edit type"</p> 
2.	<p>Select which controller is to be used as test environment and confirm the dialog with "OK".</p>  <p>If several controllers in the project use the selected block, a controller has to be selected as test environment.</p>

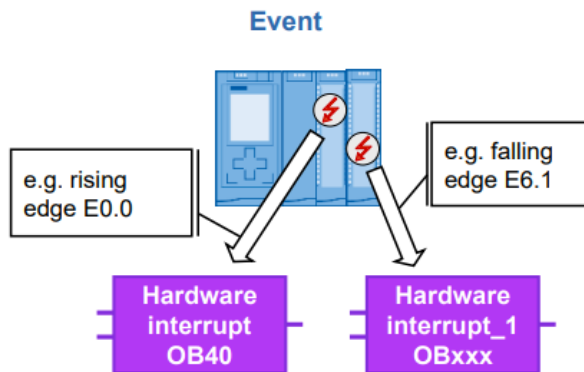
3.7 Libraries

Step	Instruction																														
3.	<p>The block opens. A new version of the block is created.</p> 																														
4.	<p>Add an input tag.</p> <p>In this place you have the option to test the change on the block by loading the project onto a controller. When you have finished testing the block, continue with the following steps.</p> <p>! This object is connected with a type in the library and is currently in the "in test" state. Any change to this test instance is mirrored in the version of the type in the test state:</p>  <table border="1" data-bbox="496 786 1230 981"> <thead> <tr> <th></th> <th>Name</th> <th>Data type</th> <th>Default value</th> <th>Retain</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Input</td> <td></td> <td></td> <td></td> </tr> <tr> <td>2</td> <td>velocity</td> <td>Real</td> <td>0.0</td> <td>Non-retain</td> </tr> <tr> <td>3</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>4</td> <td>Output</td> <td></td> <td></td> <td></td> </tr> <tr> <td>5</td> <td>control</td> <td>*typeEngineData*</td> <td></td> <td>Non-retain</td> </tr> </tbody> </table>		Name	Data type	Default value	Retain	1	Input				2	velocity	Real	0.0	Non-retain	3					4	Output				5	control	*typeEngineData*		Non-retain
	Name	Data type	Default value	Retain																											
1	Input																														
2	velocity	Real	0.0	Non-retain																											
3																															
4	Output																														
5	control	*typeEngineData*		Non-retain																											
5.	<p>Click the "release the version" button.</p> <p>! This object is connected with a type in the library and is currently in the "in test" state. Any change to this test instance is mirrored in the version of the type in the test state: You can release the version or discard the changes and delete the version.</p> 																														
6.	<p>A dialog box opens. Here you can store a version-related comment. Confirm the dialog with "OK".</p>  <p>Release type version</p> <p>Define the properties for the released type version.</p> <p>A new version will be released for the selected types. Assign them common properties or confirm the recommended properties.</p> <p>Name of type: Engine</p> <p>Version: 0.0.2</p> <p>Author: User</p> <p>Comment:</p> <p>Options</p> <p><input type="checkbox"/> Update instances in the project</p> <p><input type="checkbox"/> Delete unused type versions from the library</p> <p>OK Cancel</p> <p>If there are several usage locations of the block in different controllers of the project, you can update them all at the same time: "Update instances in the project".</p> <p>If older versions of the element are no longer required you can delete them by clicking "Delete unused type versions from library"</p>																														

3.8 Increased performance for hardware interrupts

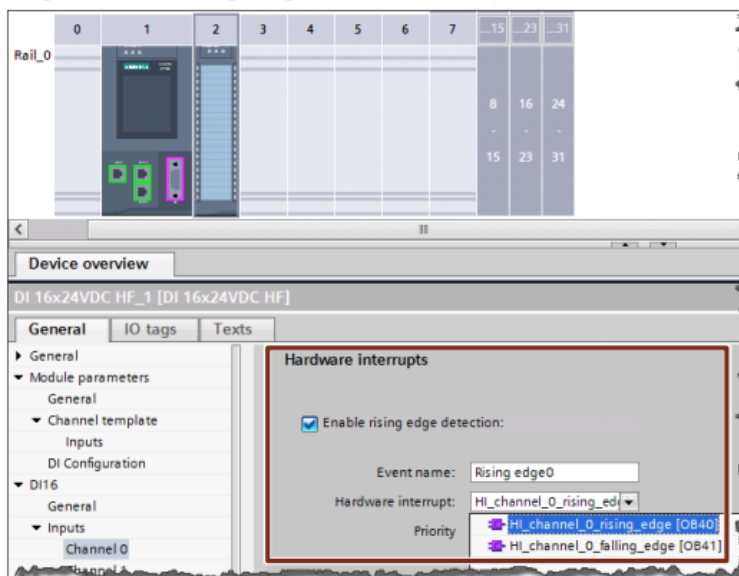
The processing of the user program can be influenced by events such as hardware interrupts. When you need a fast response of the controller to hardware events (e.g. a rising edge of a channel of a digital input module), configure a hardware interrupt. For each hardware interrupt a separate OB can be programmed. This OB is called by the operating system of the controller in the event of a hardware interrupt. The cycle of the controller is therefore interrupted and continued after processing the hardware interrupt.

Figure 3-40: Hardware interrupt is calling OB



In the following figure you can see the configuration of a "hardware interrupt" in the hardware configuration of a digital input module.

Figure 3-41: Configuring hardware interrupt



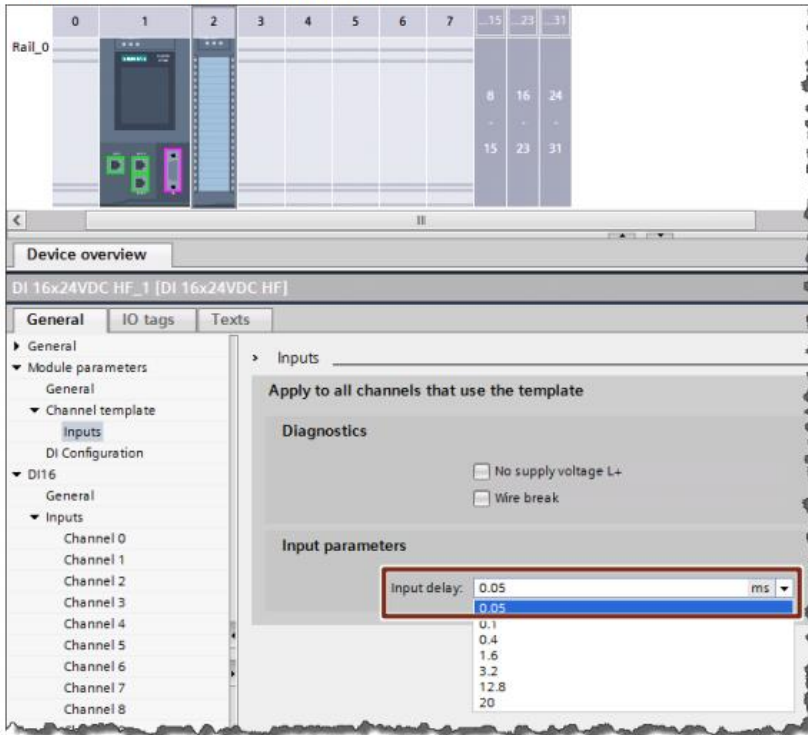
Advantages

- Fast system response to events (rising, falling edge, etc.)
- Each event can start a separate OB.

Recommendation

- Use the hardware interrupts in order to program fast responses to hardware events.
- If the system response is not fast enough despite programming a hardware interrupt, you can still accelerate the response. Set as small an "Input delay" as possible in the module. A response to an event can always only occur if the input delay has lapsed. The input delay is used for filtering the input signal in order to, for example, compensate faults such as contact bounce or chatter.

Figure 3-42: Setting input delay



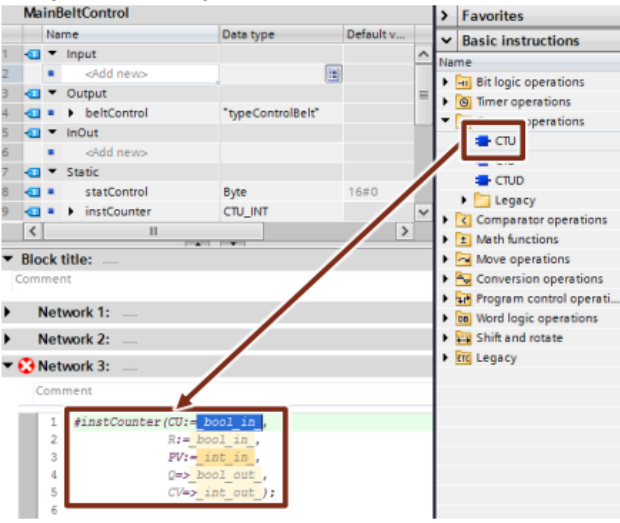
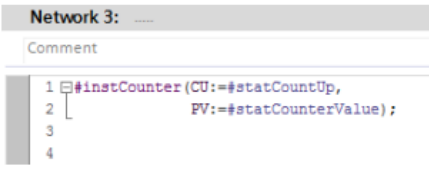
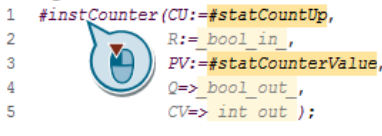
3.10 SCL programming language: Tips and Tricks

3.10.1 Using call templates

Many instructions of the programming languages offer a call template with a list of existing formal parameters.

Example

Table 3-12: Easy expanding of the call template

Step	Instruction
1.	<p>Drag an instruction from the library into the SCL program. The editor shows the complete call template.</p> 
2.	<p>Now fill in the required parameters "CU" and "PV" and finish the entry with the "Return" button.</p>
3.	<p>The editor automatically reduces the call template.</p> 
4.	<p>If you want to edit the complete call later on again, proceed as follows. Click into the call at any place and then click "CTRL+SHIFT+SPACE". You are now in the "Call Template" mode. The editor expands the call again. You can navigate with the "TAB" button through the parameters.</p> 
5.	<p>Note: In the "Call Template" mode the writing is in italics.</p>


3.10.2 What instruction parameters are mandatory?

If you are expanding the call template, the color coding will show you straight away what formal parameters of an instruction are optional and which ones are not. Mandatory parameters are marked dark.

3.10.3 Drag-and-drop with entire tag names

In the SCL editor you can also use drag-and-drop functions. For tag names you are additionally supported. If you want to replace one tag by another, proceed as follows.

Table 3-13: Drag-and-drop with tags in SCL

Step	Instruction
1.	<p data-bbox="501 689 1390 748">Drag the tag via drag-and-drop to the tag in the program that is to be replaced. Hold the tag for more than 1 second before releasing it.</p> <div data-bbox="501 757 1125 1153"><p>The screenshot shows the 'MainBeltControl' variable declaration table with columns 'Name' and 'Data type'. The 'statCountUp' variable is highlighted with a red box. Below the table, the 'Network 1' code snippet is shown with 'statCountUp' also highlighted. A mouse cursor is positioned over the code with a tooltip indicating a 1-second hold.</p></div> <p data-bbox="501 1167 836 1202">The complete tag is replaced.</p>

3.10.4 Structuring with the keyword REGION (V14 or higher)

The SCL code can be divided in areas with the keyword REGION. These areas can be given a name and can be collapsed and expanded.

Advantages

- Better overview
- Easy navigation even in large blocks
- Ready code fragments can be collapsed.

Properties

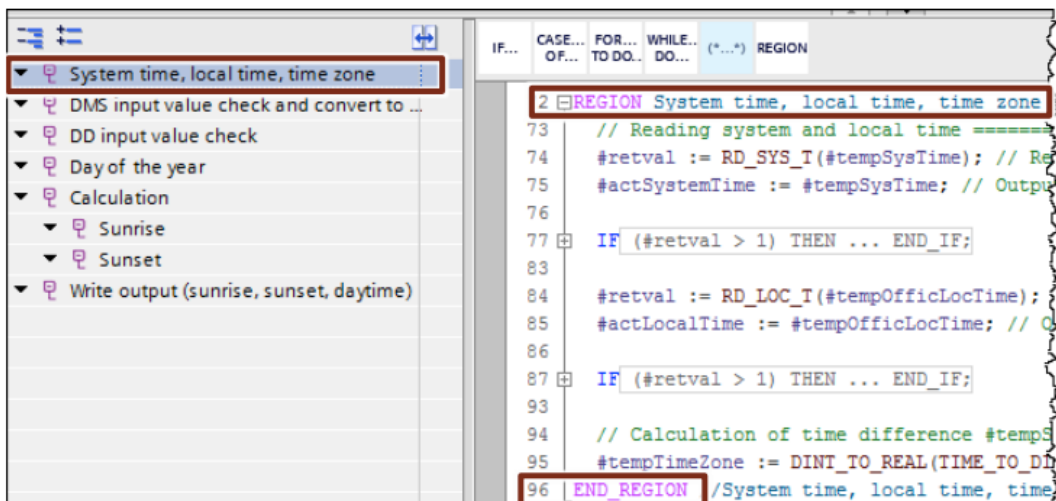
REGIONS can be nested.

Recommendation

Use the keyword REGION for the structuring of your SCL blocks.

Example

Figure 3-43: SCL editor



3.10.5 Correct use of FOR, REPEAT and WHILE loops

There are different versions and applications for the use of loops. The following examples show the differences.

Properties: FOR loop

The FOR loop runs through a **defined number of runs**. The loop variable is assigned a start value at the beginning. Afterwards it is incremented up to the end value in each loop run with the specified step size.

For reasons of performance, the start as well as the end value is calculated once at the beginning. Consequently, the loop variable can no longer influence the loop code.

Syntax

```
FOR statCounter := statStartCount TO statEndCount DO
    // Statement section ;
END_FOR;
```

With the EXIT command the loop can be interrupted at any time.

Properties: WHILE loop

The WHILE loop is ended by a termination condition. The **termination condition** is checked **before the start** of the loop code. I.e., the loop is not executed, if the condition is not instantly fulfilled. Each variable can be adjusted for the next run in the loop code.

Syntax

```
WHILE condition DO
    // Statement section ;
END_WHILE;
```

Properties: REPEAT loop

The REPEAT loop is ended by a termination condition. The **termination condition** is checked **at the end** of the loop code. This means the loop is **run through at least once**. Each variable can be adjusted for the next run in the loop code.

Syntax

```
REPEAT
    // Statement section ;
UNTIL condition
END_REPEAT;
```

Recommendation

- Use FOR loops if the loop variable is clearly defined.
- Use the WHILE or REPEAT loop if a loop variable has to be adjusted during the loop processing.

3.10.6 Using CASE instruction efficiently

With the CASE instruction in SCL, it will be exactly jumped to the selected CASE block condition. After executing the CASE block the instruction is finished. This allows you, for example, to check frequently required value ranges more specifically and easily.

Example

```
CASE #myVar OF
    5:
        #Engine(#myParam);
    10,12:
        #Transport(#myParam);
    15:
        #Lift(#myParam);
    0..20:
        #Global(#myParam);
// Global is never called for the values 5, 10, 12 or 15!
ELSE
END_CASE;
```

Note

CASE instructions also work with CHAR, STRING data types, as well as with elements (see example in chapter [2.8.5 Data type VARIANT](#)).

3.10.7 No manipulation of loop counters for FOR loop

FOR loops in SCL are pure counter loops, i.e. the number of iterations is fixed when the loop is entered. In a FOR loop, the loop counter cannot be changed. With the EXIT instruction a loop can be interrupted at any time.

Advantages

- The compiler can optimize the program better, since it does not know the number of iterations.

Example

```
FOR #statVar := #statLower TO #statUpper DO
    #statVar := #statVar + 1; // no effect, compiler warning
END_FOR;
```

3.10.8 FOR loop backwards

In SCL you can also increment the index of FOR loops backwards or in another step width. For this, use the optional "BY" key word in the loop head.

Example

```
FOR #statVar := #statUpper TO #statLower BY -2 DO

END_FOR;
```

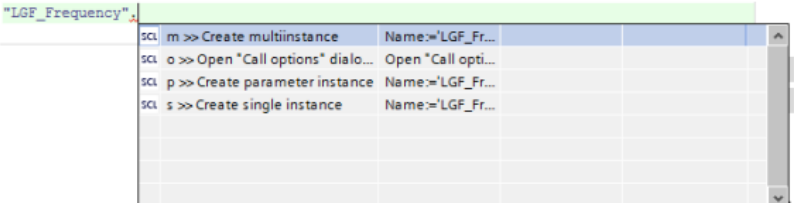
If you are defining "BY" as "-2", as in the example, the counter is lowered by 2 in every iteration. If you omit "BY", the default setting for "BY" is 1

3.10.9 Easy creation of instances for calls

If you prefer to work with the keyboard, there is a simple possibility to create instances for blocks in SCL.

Example

Table 3-14: Easy creation of instances

Step	Instruction															
1.	<p>Give the block a name, followed by a "." (dot). The automatic compilation now shows you the following.</p>  <p>The screenshot shows a table with the following content:</p> <table border="1"> <thead> <tr> <th>Shortcut</th> <th>Description</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>SCl m >></td> <td>Create multiinstance</td> <td>Name:'LGF_Fr...</td> </tr> <tr> <td>SCl o >></td> <td>Open "Call options" dialo...</td> <td>Open "Call opti...</td> </tr> <tr> <td>SCl p >></td> <td>Create parameter instance</td> <td>Name:'LGF_Fr...</td> </tr> <tr> <td>SCl s >></td> <td>Create single instance</td> <td>Name:'LGF_Fr...</td> </tr> </tbody> </table>	Shortcut	Description	Name	SCl m >>	Create multiinstance	Name:'LGF_Fr...	SCl o >>	Open "Call options" dialo...	Open "Call opti...	SCl p >>	Create parameter instance	Name:'LGF_Fr...	SCl s >>	Create single instance	Name:'LGF_Fr...
Shortcut	Description	Name														
SCl m >>	Create multiinstance	Name:'LGF_Fr...														
SCl o >>	Open "Call options" dialo...	Open "Call opti...														
SCl p >>	Create parameter instance	Name:'LGF_Fr...														
SCl s >>	Create single instance	Name:'LGF_Fr...														
2.	<p>On the top you can see the already existing instances. In addition, you can directly create a new single instance or multi-instance. Use the shortcuts "s" or "m" to go directly to the respective entries in the automatic compilation window.</p>															

3.10.10 Handling of time tags

You can calculate the time tags in SCL just as with normal numbers i.e. you do not need to look for additional functions, such as, e.g. T_COMBINE but you can use simple arithmetic. This approach is called "overload of operands". The SCL compiler automatically uses the suitable functions. You can use a reasonable arithmetic for the time types and can therefore program more efficiently.

Example

```
time difference := time stamp_1 - time stamp_2;
```

The following table summarizes the overloaded operators and the operations behind it:

3.10 SCL programming language: Tips and Tricks

Table 3-15: Overloaded operands for SCL

overloaded operand	Operation
ltime + time	T_ADD LTime
ltime – time	T_SUB LTime
ltime + lint	T_ADD LTime
ltime – lint	T_SUB LTime
time + time	T_ADD Time
time - time	T_SUB Time
time + dint	T_ADD Time
time - dint	T_SUB Time
ldt + ltime	T_ADD LDT / LTime
ldt – ltime	T_SUB LDT / LTime
ldt + time	T_ADD LDT / Time
ldt – time	T_SUB LDT / Time
dtl + ltime	T_ADD DTL / LTime
dtl – ltime	T_SUB DTL / LTime
dtl + time	T_ADD DTL / Time
dtl – time	T_SUB DTL / Time
ltod + ltime	T_ADD LTOD / LTime
ltod – ltime	T_SUB LTOD / LTime
ltod + lint	T_ADD LTOD / LTime
ltod – lint	T_SUB LTOD / LTime
ltod + time	T_ADD LTOD / Time
ltod – time	T_SUB LTOD / Time
tod + time	T_ADD TOD / Time
tod – time	T_SUB TOD / Time
tod + dint	T_ADD TOD / Time
tod – dint	T_SUB TOD / Time
dt + time	T_ADD DT / Time
dt – time	T_SUB DT / Time
ldt – ldt	T_DIFF LDT
dtl – dtl	T_DIFF DTL
dt – dt	T_DIFF DT
date – date	T_DIFF DATE
ltod – ltod	T_DIFF LTOD
date + ltod	T_COMBINE DATE / LTOD
date + tod	T_COMBINE DATE / TOD

3.10.11 Unnecessary IF instruction

Programmers often think in IF-THEN-ELSE instructions. This frequently leads to unnecessary constructs in programs.

Example

```
IF (statOn1 = TRUE AND statOn2 = TRUE) THEN
    statMotor := TRUE;
ELSE
    statMotor := FALSE;
END_IF
```

Recommendation

Remember that for Boolean requests a direct assignment is often more effective. The entire construct can be programmed with one line.

Example

```
statMotor := statOn1 AND statOn2;
```

4 Hardware-independent programming

To make sure that a block can be used on all controllers without any further adjustments, it is important not use hardware-dependent functions and properties.

4.1 Data types of S7-300/400 and S7-1200/1500

Below is a list of all elementary data types and data groups.

Recommendation

- Only use the data types that are supported by the controllers on which the program is to run.

Table 4-1: Elementary data types correspond to standard EN 61131-3

	Description	S7-300/400	S7-1200	S7-1500
Bit data types	<ul style="list-style-type: none"> • BOOL • BYTE • WORD • DWORD 	yes	yes	yes
	<ul style="list-style-type: none"> • LWORD 	no	no	yes
Character type	<ul style="list-style-type: none"> • CHAR (8 bit) 	yes	yes	yes
Numerical data types	<ul style="list-style-type: none"> • INT (16 bit) • DINT (32 bit) • REAL (32 bit) 	yes	yes	yes
	<ul style="list-style-type: none"> • SINT (8 bit) • USINT (8 bit) • UINT (16 bit) • UDINT (32 bit) • LREAL (64 bit) 	no	yes	yes
	<ul style="list-style-type: none"> • LINT (64 bit) • ULINT (64 bit) 	no	no	yes
Time types	<ul style="list-style-type: none"> • TIME • DATE • TIME_OF_DAY 	yes	yes	yes
	<ul style="list-style-type: none"> • S5TIME 	yes	no	yes
	<ul style="list-style-type: none"> • LTIME • L_TIME_OF_DAY 	no	no	yes

4.1 Data types of S7-300/400 and S7-1200/1500

Table 4-2 Data groups that are made up of other data types

	Description	S7-300/400	S7-1200	S7-1500
Time types	• DT (DATE_AND_TIME)	yes	no	yes
	• DTL	no	yes	yes
	• LDT (L_DATE_AND_TIME)	no	no	yes
Character type	• STRING	yes	yes	yes
Feld	• ARRAY	yes	yes	yes
Structure	• STRUCT	yes	yes	yes

Table 4-3: Parameter types for formal parameters that are transferred between blocks

	Description	S7-300/400	S7-1200	S7-1500
Pointer	• POINTER • ANY	yes	no	yes ¹⁾
	• VARIANT	no	yes	yes
Blocks	• TIMER • COUNTER	yes	yes ²⁾	yes
	• BLOCK_FB • BLOCK_FC	yes	no	yes
	• BLOCK_DB • BLOCK_SDB	yes	no	no
	• VOID	yes	yes	yes
PLC data types	• PLC DATA TYPE	yes	yes	yes

¹⁾ For optimized access, only symbolic addressing is possible

²⁾ For S7-1200/1500 the TIMER and COUNTER data type is represented by IEC_TIMER and IEC_Counter.

4.2 No bit memory but global data blocks

Advantages

- Optimized global DBs are clearly more powerful than the bit memory address area that is not optimized only for reasons of compatibility.

Recommendation

- Dealing with bit memory (system and clock flags also) is problematic since the size of the flag area of each controller has is different. Do not use bit memory for the programming but always global data blocks. This is how the program can always be used universally.

4.3 Programming of "Cycle bits"

Recommendation

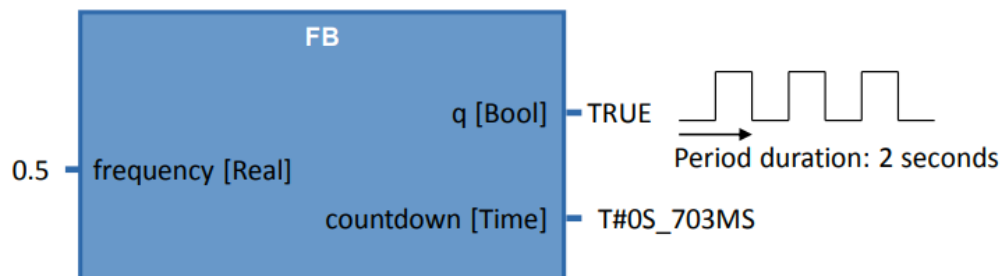
For the programming of clock memory, the hardware configuration always has to be correct.

Use a programmed block as clock generator. Below, you can find a programming example for a clock generator in the SCL programming language.

Example

The programmed block has the following functions. A desired frequency is preset. The "Q" output is a Boolean value that toggles in the desired frequency. The "countdown" output outputs the remaining time of the current state of "q".

If the desired frequency is smaller or equal 0.0, then the output q = FALSE and Countdown = 0.0.



Note

The complete programming example can be found in the following entry:

<https://support.industry.siemens.com/cs/ww/en/view/109479728>

”

Summary

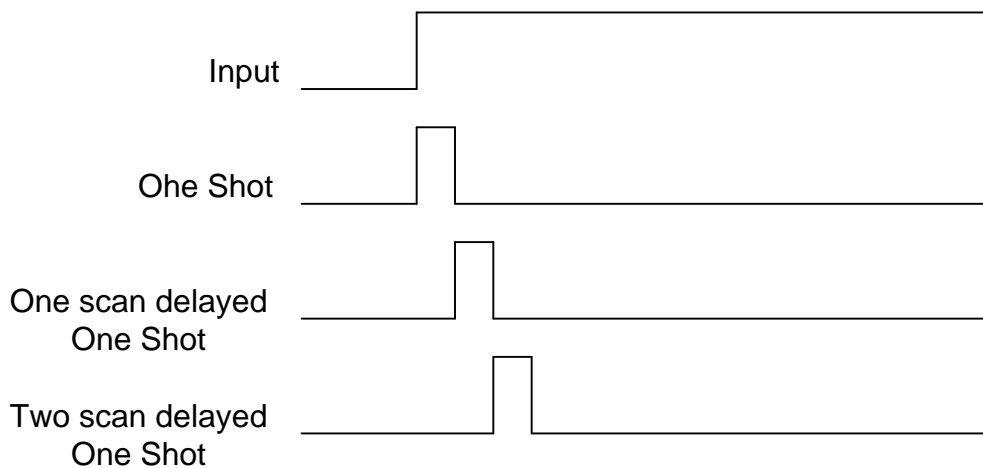
From the beginning of this chapter, the discussion has been one of planning to not fail by running out of something. In the case of Charlie and the MTA, it was a nickel that was needed. During the remainder of the chapter, the quantity is time. The plan should include enough time to guarantee execution of all programs without failing. The conservative planner usually allows extra time and he is more successful in his implementation of the overall system. Don't run out of CPU time!

Problems

- For each example, list whether the task would be best programmed as a continuous, periodic or event (interrupt driven) task.

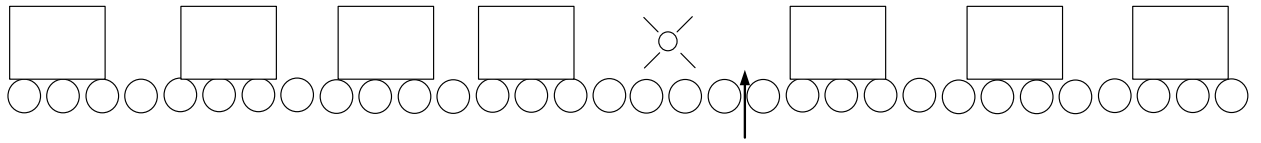
Fill a tank to its maximum level and then open a drain valve	↓
Read the thickness of a paper roll every 20 ms	
A gluing station must adjust the amount of glue it applies to compensate for changes in the speed of the axis. After the motion planner executes, check the command speed of an axis and vary the amount of glue, if needed.	
Your system must check the position of a field arm each 0.1 s and calculate the average rate of change in its position. This is used to determine braking pressure	
In a production line, if any of the programs detect an unsafe condition the entire line must shut down. The shutdown procedure is the same regardless of the unsafe condition	

- For the following figure, provide logic for each one-shot. Identify in your program each signal labeled below. You may use either Siemens or A-B one-shot logic or you may create the logic entirely without one-shot instructions:



- Write the program that would accomplish the task described in the chapter that was solved by using a frequency divider (prove that the update between the card and the CPU was happening only about every $\frac{3}{4}$ second).
- Pick a commercial wheel with toothed gears and a proximity switch wired to a PLC input. Describe analytically how fast the wheel is allowed to turn in order for the program in your PLC to be guaranteed to not miss any pulses without using a high-speed pulse counter input.
- For the process below, the roll conveyor carries boxes from left to right. If a box is too close to the one before it, the spacing bar comes up (arrow) to hold it back until a constant spacing

is achieved. If a box is spaced in excess of the minimum, the box is allowed to pass on with no blocking. Write a program in ladder logic to control the blocking bar based on the photo-eye and an input from a pulse tachometer. Assume the pulse tach is a dint word with a constantly increasing number of pulses each time read.



- Use a program previously written and using the table in the 'Finally' section of the chapter to calculate the execution time of the program. Do you think you will ever do this in the real world? If you do not, think again!



This work is licensed under a Creative Commons Attribution 4.0 International License.