FAQ January 2013

Migration of STL Programs to S7-1500

STEP 7 (TIA Portal)

https://support.industry.siemens.com/cs/ww/en/view/67655405

This entry is from the Siemens Industry Online Support. The general terms of use (http://www.siemens.com/terms_of_use) apply.

Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, solutions, machines, equipment and/or networks. They are important components in a holistic industrial security concept. With this in mind, Siemens' products and solutions undergo continuous development. Siemens recommends strongly that you regularly check for product updates.

For the secure operation of Siemens products and solutions, it is necessary to take suitable preventive action (e.g. cell protection concept) and integrate each component into a holistic, state-of-the-art industrial security concept. Third-party products that may be in use should also be considered. For more information about industrial security, visit http://www.siemens.com/industrialsecurity.

To stay informed about product updates as they occur, sign up for a product-specific newsletter. For more information, visit http://support.industry.siemens.com.

Caution

The functions and solutions described in this article confine themselves predominantly to the realization of the automation task. Furthermore, please take into account that corresponding protective measures have to be taken in the context of Industrial Security when connecting your equipment to other parts of the plant, the enterprise network or the internet. Further information can be found in Entry ID: !50203404!.

http://support.automation.siemens.com/WW/view/de/50203404

Table of contents

1 Migration of STL Programs to S7-1500

1.1 Information on Migrating STL Programs

Passing values using registers or the status word

In order to achieve the highest program processing performance in CPUs of the S7-1500 series, the transfer of values between blocks is only possible using the block interface, global data blocks or PLC tags.

In LAD and FBD you do not have the option of passing values using registers (accumulators, AR1, AR2, DB, DI, for example) or the status word. Value passing is possible but restricted in STL. However, please be aware that the program processing slows down if you use these areas to pass values between different blocks.

The rules below apply for STL.

- The contents of the registers, accumulators and the status word are available only in STL networks. If a LAD or FBD network follows an STL network, you cannot access the register contents previously set in STL from the LAD or FBD network. The register contents become available again in a subsequent STL network.

  The RLO bit is an exception: with a language change it is set to "undefined" and is no longer available in subsequent networks.

- The values from registers, accumulators and the status word are generally not transferred to called blocks. The "CC" and "UC" instructions are the only exceptions. If you use "UC" or "CC" and would like to pass parameters to the called block using registers, the status word or accumulators, you must enable the "Supply parameters using registers" option in the Properties of the called block. Note that this option is only available for STL blocks with standard access and the block must not have any formal parameters. When the option is enabled, you can pass register contents between blocks.

  The RLO bit is an exception here too: with a language change it is set to "undefined" and is no longer available in subsequent networks.

- You can use the BIE bit to pass an error message to the calling block. You must first use the "SAVE" instruction in the called block to save the error message in the BIE bit. Then you can read the BIE bit in the calling block. There are also jump commands like SPBNB that set the BIE.

- The data block register DB is set to "0" after each access to the data block with the specification of a fully qualified address (%DB10.DBW10, "MyDB.Component", for example). A subsequent partially qualified access leads to an error during compiling.

- In S7-1500, if you address a local formal parameter from the block interface of an FB (with the L #myIn instruction, for example), you always access the data block that you specified as instance when calling the block. The AUF DI, L AR2, +AR2, TDB and TAR instructions change the content of the DI or address register AR2, but the registers are no longer evaluated with the addressing of local formal parameters.

Programming examples are given in the following.

Master Control Relay
The Master Control Relay is not available in the S7-1500. Migration reports a fault. Change the program manually. Define the conditions in the form of block parameters, for example, to execute the instructions or networks.

LEAVE and ENT
The "LEAVE" and "ENT" instructions are not provided in the S7-1500 because only two accumulators are available. Migration reports a fault. Change the program manually. Use temporary variables, for example, to store interim results.

Block parameters of the "Block_DB" type
The "Block_DB" parameter type is not available in the S7-1500. Migration changes these parameters and assigns them the "DB_Any" data type instead. However, in the S7-1500 it is not possible to assign library instructions when called an instance in the form of a parameter of the "DB_Any" type. The example below shows how a library block is called with a variable instance on a CPU of the S7-300/400 family. This sequence cannot be migrated to S7-1500.

STL	**Explanation**
CALL GET, #myBlock_DB	// The "GET" library block is called with the
REQ:= <Operand>	instance block
ID:= <Operand>	that is currently at the
NDR:= <Operand>	"myBlock_DB" block parameter.
Error:= <Operand>	

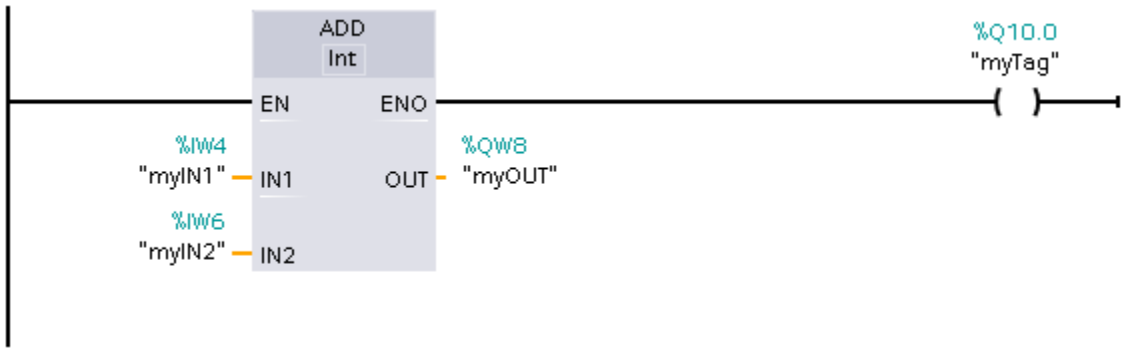
1.2 Passing Values Using Registers in Case of Language Change

Introduction
The contents of the registers, accumulators and the status word are available only in STL networks. If a LAD or FBD network follows an STL network, you cannot access the register contents previously set in STL from the LAD or FBD network. The register contents become available again in a subsequent STL network. The RLO bit is an exception: with a language change it is set to "undefined" and is no longer available in subsequent networks.

Migration of passing values using registers in case of language change
If registers are accessed in a migrated LAD or FBD network, a fault is reported during compilation. Change the program so that registers can only be set and read in STL networks.

Example
The example below shows the migration of an accumulator access. The first two figures show the program before migration.

Network 1: The "myIN1" operand is loaded into Accu 1. Then "myIN2" is loaded into Accu 1, which transfers "myIN1" to Accu 2. Now both values are added. The result is stored in Accu 1 and from there it is assigned to the "myOUT" operand.



Network 2: The "myIN3" operand is loaded into Accu 1. This transfers the "myOUT" operand that is still there to Accu 2 and can be added immediately to the "myIN3". The "myOUT" operand does not have to be reloaded explicitly.

```
1    L     "myIN3"     %IW12
2    +I
3    T     "myResult"  %IW14
4
```

After migration Network 2 is defective, because no more values can be written to the accumulators through the LAD network. You must change the program manually so that both values are loaded into the accumulators explicitly in STL. The figure below shows Network 2 after the fault has been cleared.

Network 2: Both values are loaded into the accumulators in STL before they are added.

```
1     L     "myIN3"     %IW12
2     L     "myOUT"     %QW8
3     +I
4     T     "myResult"  %QW14
5
```

1.3 Passing Values Using Registers or the Status Word with a Block Call

Migration of passing values using registers or the status word with a block call

The values from registers, accumulators and the status word are set to "0" upon change of block or receive the "undefined" status. This prevents them from being passed to called blocks. The "CC" and "UC" instructions are the only exceptions. If you use "UC" or "CC" and would like to pass parameters to the called block using registers, the status word or accumulators, you must enable the "Supply parameters using registers" option in the Properties of the called block. Note that this option is only available for STL blocks with standard access and the block must not have any formal parameters. When the option is enabled, you can pass register contents between blocks. The RLO bit is an exception: Upon block transfer "undefined" is always set and is no longer available after a block call. You can use the BIE bit to pass an error message to the calling block. You must first save the error message initially in the BIE bit of the called block. You use the "SAVE" or "SPBNB" instruction for this. Then you can read the BIE bit in the calling block. Migration reports a fault if after a block call you access register content that has been set in the called block. Change the program manually. Use tags in data blocks or PLC tags, for example, to return values to the calling block.

Example
The example below shows how to change your program to pass values to a calling block using registers.

The first two tables show the program before migration.

STL	**Explanation**
CALL "MyFB", "MyFB_DB" = #MyBit	The RLO of the "MyFB" block is assigned to the "MyBit" operand after processing.

The second table shows how you must change the program.

STL	Explanation
CALL "MyFB", "MyFB_DB"	In the called block "MyFB" you use the "SAVE" instruction to write the current RLO at any point into the BIE bit.
U BIE = #MyBit	The BIE bit is read in the calling block. The value of the BIE bit is assigned to the "MyBit" operand.

1.4 Passing Values Using Registers with the CC and UC Instructions

Introduction

With the S7-300/400 you can use the UC and CC instructions to program block calls. The parameters are transferred to the calling block in this case not over the interface but by using registers (like AR1, AR2, DB, DI, for example), the accumulators or the status word.

These calls slow down the program processing and are therefore no longer possible as presettings. The CALL instruction is used instead in the S7-1500. However, using the CALL instruction means that you cannot have any indirect block calls. If you want to use UC or CC for indirect block calls, you must enable the "Supply parameters using registers" option in the Properties of the called block. You then have the possibility of passing register contents between different blocks. The RLO bit is an exception: with a block transfer it is set to "undefined" and is no longer available after a block call.

Note that this option is only available for STL blocks with standard access and the block must not have any formal parameters. When the option is enabled, you can pass register contents between blocks. The RLO bit is an exception here too: with a block transfer it is set to "undefined" and is no longer available after a block call.

Migration of block calls using "UC" or "CC"

Migration handles block calls using "UC" or "CC" as follows.

- The "UC FC" instruction with specification of a block number is replaced by the "CALL" instruction.
- The "CC FC" instruction with specification of a block number is replaced by the "CALL" instruction. A jump command that implements the conditional call is inserted in addition.
- The "UC FC" and "CC FC" instructions with indirect specification of the block numbers remain unchanged.
- The "UC FB" and "CC FB" instructions with direct or indirect specification of a block number remain unchanged.

Example
The example below shows the migration of block calls using "UC".

The first two tables show the program before migration.

STL
UC FC 10
UC FC [#temp0]
UC FB 10
UC FB [temp0]

The table below shows the program after migration.

STL	Explanation
CALL FC 10	
UC FC [#temp0]	The "Supply parameters using registers" option must be set on the called block.
UC FB 10	The "Supply parameters using registers" option must be set on the called block.
UC FB [#temp0]	The "Supply parameters using registers" option must be set on the called block.

1.5 Fully Qualified Addresses in STL

Introduction

The addressing of DB variables with specification of the DB name or the DB number is called fully qualified addressing. The DB register is set to "0" after each access to a DB with specification of a fully qualified address. If you want to access the DB register again after a fully qualified address access, you must first give it a value again using the AUF DB instruction.

Migration of fully qualified addresses
If necessary, migration inserts the "AUF" instruction after a qualified address access in order to load the current data block once again into the data block register.

Example
The example below shows the migration of a fully qualified address.

The first two tables show the program before migration.

STL	Explanation
AUF "MyDB"	"MyDB" is loaded into the data block register.
L %DBW1	Data elements from "MyDB" are addressed partially qualified.
L "Global_DB".Data1	Elements from a global DB are then addressed fully qualified. This loads the "Global_DB" implicitly into the data block register.
L "Global_DB".DBW2	
T DBW[AR1, P#0.0]	A subsequent access addresses the "Global_DB", because it is in the DB register.

The table below shows the program after migration.

STL	Explanation
AUF "MyDB"	
L "MyDB".DBW1	The partially qualified access is converted into a fully qualified access.
L "Global_DB".Data1	

	The DB register is reset after a fully qualified access.
L "Global_DB".DBWord2	
AUF "Global_DB"	The "Global_DB" must be loaded again with AUF into the DB register.
T DBW[AR1, P#0.0]	

1.6 Partially Qualified Addresses in STL

Introduction

The addressing of DB variables without specification of the DB name or the DB number is called partially qualified addressing. Partially qualified addressing accesses a defined value in the data block that is currently in the DB register.

The following restrictions hold for partially qualified addresses in S7-1500:

- In S7-1500 partially qualified addresses are permitted only if the DB register has been set explicitly in the current block. You set the DB register with the "AUF" instruction, for example. You can partially address only variables in data blocks with standard access.
- When the block is called, the data block register is set to "0" in S7-1500. It is thus not possible to open a data block in a block and partially qualified address data elements from the data block in a lower-level block. Set the DB register in the current block before you address partially qualified a DB variable.
- The data block register DB is set to "0" also after each fully qualified access (%DB10.DBW10, for example). After a fully qualified access set the DB register again before you address partially qualified a DB variable.

Migration of partially qualified addresses
Migration handles partially qualified addresses in STL as follows.

- The partially qualified address is converted as far as possible into a fully qualified address.
- If the data block has been opened in the calling code block and the DB cannot be specified uniquely, the migration inserts a parameter of the "DB_Any" data type in the interface of the called block. The data block name is transferred to this parameter. Migration inserts the "AUF" instruction in the called block in order to open the data block.

Example
The example below shows the migration of a partially qualified address.

The first two tables show the program before migration.

STL	**Explanation**
L DBW10	A data block has been opened in the calling
L DBW12	block and transferred to the DB register. In the
+I	current block the values "DBW10", "DBW12"
T DBW14	and "DBW14" are transferred from the data

block that is currently in the DB register.

The table below shows the program after migration.

STL	Explanation
AUF "PlcmigTempBlockDB"	
L DBW10	Migration inserts a parameter of the "DB_Any"
L DBW12	data type in the interface of the called block.
+I	The data block name is transferred to this
T DBW14	parameter.
	Migration inserts the "AUF" instruction in the called block in order to open the data block transferred at the interface.

Note

Migration of partially qualified block parameters

Migration also converts partially qualified DB parameters into fully qualified parameters. Note that this can change the type of parameter transfer to the called block: It might be that using the fully qualified address the called block no longer accesses the actual parameter directly, but works with a copy transferred when the block is called. Therefore you must check that the semantics of the migrated program still match that of the initial program.

For more information, go to: Access to block parameters during processing of the program. If you do not want this change, you can use a parameter with a structured data type, for example, instead of an elementary block parameter. Define the PLC data type (UDT) as formal parameter and transfer a variable of this type or a DB derived from the PLC data type (UDT).

Examples:

```
CALL "MyFC"
InStruct :="DBofUDT"
or
CALL "MyFC"
InStruct := "DBArrayOfUDT".a[#i]
```

1.7 Access to the Instance DB in STL

Introduction

The "AUF DI" and "TDB" instructions create a data block in the DI register. In S7-300/400 the block opened there is an instance data block. The subsequent symbolic addressing of a local formal parameter from the block interface of an FB (IN, OUT, InOut or Static) no longer addresses the data block that was specified as instance when the block was called, but the data block that is in the DI register. The instance DB must be loaded into the DI register in order subsequently to address a local formal parameter symbolically from the block interface.

Also after the "L AR2", "+ AR" and "TAR" instructions it is not possible to address a formal parameter symbolically from the block parameter, because the access base for parameter access has been destroyed by the instructions.

This behavior has been corrected in S7-1500. In S7-1500, if you address a local formal parameter from the block interface (with the L #myIn instruction, for example), you always access the data block that you specified as instance when calling the block. The AUF DI, L AR2, +AR2, TDB and TAR instructions change the content of the DI or address register, but the registers are no longer evaluated with the addressing of local formal parameters.

Migration of access to local variables in the instance DB

Migration does not change the programmed access. If you have used the AUF DI, L AR2, +AR2, TDB, TAR, etc. instructions in the initial program, the semantics of the program might however change. You must change the program manually in order to regain the original semantics. If is often no longer necessary to address data using registers. Instead you can use the option of using ARRAYs in the instance DB and indirectly indexing the ARRAY elements.

Example 1
The example below shows the changed semantics of the "AUF DI" instruction.

STL	Explanation
L #MyIn1 L #MyIn2 +I T #MyOut3	The "L" and "T" instructions address local variables declared in the block interface. The values of the variables are in the instance DB specified when the block was called.
AUFDI "MyDB" L #MyIn1 L #MyIn2 +I T #MyOut3	The global data block "MyDB" is written into the DI register. Prior to migration the "L" and "T" instructions address variables declared in "MyDB". After migration, the "L" and "T" instructions address variables declared in the block interface. The DI register is not evaluated for the access in S7-1500.

Example 2
The example below shows the changed semantics of the "LAR2" instruction.

STL	Explanations
L P#M23.0	
LAR2	Prior to migration, the assignment to AR2 destroys the access base for parameter access.
L #MyIn1	Prior to migration, it is not possible to access

	"MyIn1" or an error occurs.
L EW [AR2, P#1.0]	
	After migration, access to the formal parameter
U [AR2, P#0.4]	"MyIn1" is executed correctly.
L EW [AR2. P#1.0]	Access to %EW24.0
U [AR2, Ü#0.4]	Access to %EW23.4

Example 3
The example below shows how you can address a DB variable directly in S7-1500 without using the address register.

STL	**Explanations**
AUF "MyDB"	Prior to migration, the addressing used is indirect to the register and internal to the area.
L #index	This loads a variable value (#index) into the address register 1. Depending on this value, a
LAR1	data word is loaded from "MyDB" into the accumulator 1.
L %DBW [AR1, P#10.0]	
	After migration, you can store the data values in "MyDB" in an ARRAY. The individual
L "MyDB".MyArray1[#index]	ARRAY elements can be indexed variably using the "#index" input parameter.