Technical Note PLC-EOTKN104U-EN

AC500 V3 PLC – Address operators

ADR, ^ & BITADR Operators and Examples

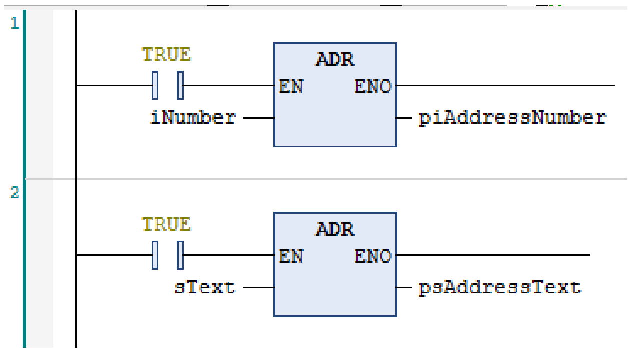
The Address operators are used for pointer access. Mainly an access by value is used by the most POUs which means the value of a variable is given inside another block. With a pointer access not the value but the whole variable is inputted which allows to change the input values during runtime. Pointer access with the ADR operator is mainly used when working with bigger arrays or structures.

ADR

The address operator ADR yields the 16-bit address of its argument. This address can be stored in a pointer or inputted to any function block or function in the project. Assuming the following 5 variables are declared.

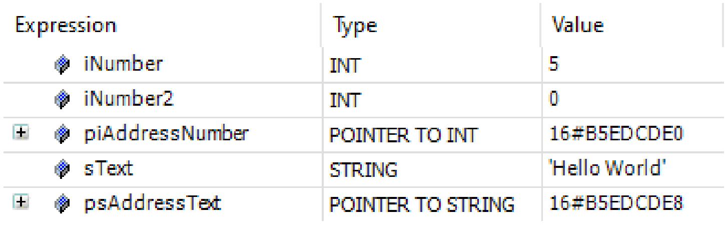
```
VAR
        iNumber:                INT                 := 5;
        iNumber2:[AF1]          INT                 := 0;
        piAddressNumber:        POINTER TO INT;
        sText:                  STRING              := 'Hello World';
        psAddressText:          POINTER TO STRING;
END_VAR
```

The following example shows how the ADR operator is used in ladder diagram and structured text.



```
piAddressNumber:= ADR(iNumber) ;
psAddressText:= ADR(sText) ;
```

After running the code, the variables piAddressNumber and psAddressText will contain the physical memory address where the two variables are stored.

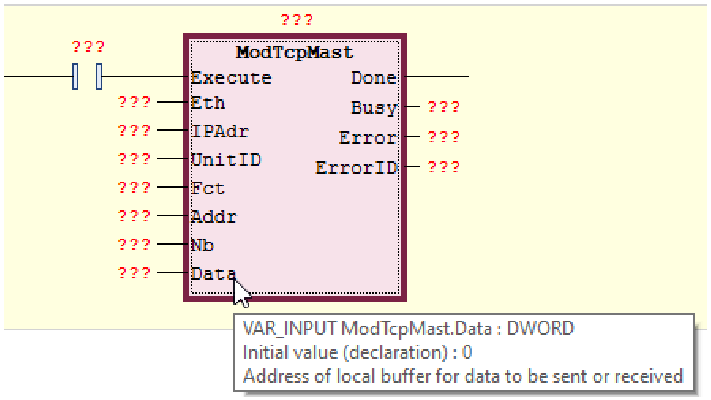


Expression	Type	Value
iNumber	INT	5
iNumber2	INT	0
piAddressNumber	POINTER TO INT	16#B5EDCDE0
sText	STRING	'Hello World'
psAddressText	POINTER TO STRING	16#B5EDCDE8

Caution! During an online change memory address can shift. Use the ADR operator cyclic to ensure that the pointer is correct. Local memory in functions and methods will become invalid after the call. Do not return pointer to variables.

How do I detect when the ADR operator must be used?

There are some function blocks and functions, which require an address as input. One of these is for example the ModTcpMast function block. The address to the data for sending or receiving is entered here. The type of these inputs is usually DWORD to represent a 16 bits number. As visible in the comment it also says that here an address must be entered.

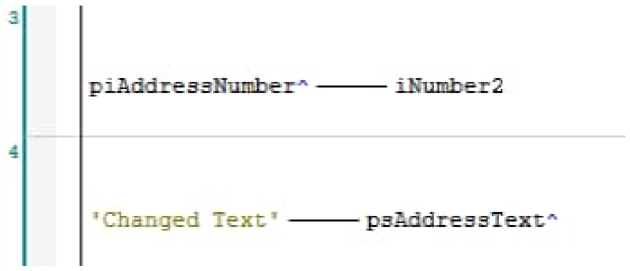


VAR_INPUT ModTcpMast.Data : DWORD
Initial value (declaration) : 0
Address of local buffer for data to be sent or received

Content Operator ^

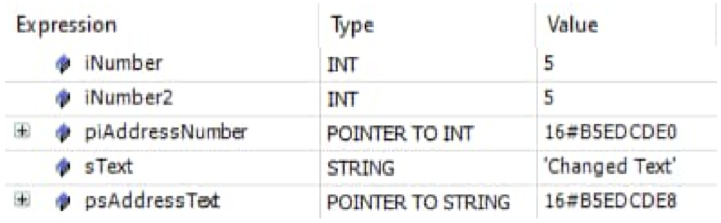
The content operator ^ is the opposite of the ADR operator. It is used to dereference pointers to get back the value. The variables from the ADR operator are used again.

The existing two lines of code remain. The following example shows how to get back the value from a pointer in ladder diagram and structured text.



```
iNumber2 := piAddressNumber^;
psAddressText^:= 'Changed Text';
```

With the first assignment the pointer piAddressNumber is dereferenced. The value 5 is assigned to the variable iNumber2. In the next line the value Changed Text is assigned to the dereferenced pointer psAddressText.



Expression	Type	Value
iNumber	INT	5
iNumber2	INT	5
piAddressNumber	POINTER TO INT	16#B5EDCDE0
sText	STRING	'Changed Text'
psAddressText	POINTER TO STRING	16#B5EDCDE8

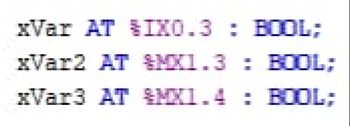
As visible in the screenshot above the Value of iNumber2 is now also 5 and the value of sText has been changed to 'Changed Text'.

Caution! When dereferencing a pointer please make sure before, that the pointer is valid. When dereferencing an invalid pointer wrong variables might be overwritten or the PLC might stop because of an invalid access exception.

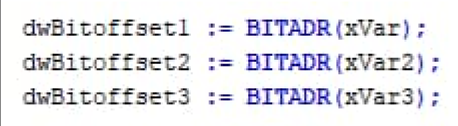
BITADR

The BITADR operator is a very specific operator which is only used for special cases. The BITADR operator can only be applied on Boolean values which have a direct memory access like shown below the addresses can be in %I, %O or %M

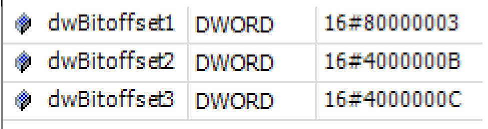
BITADR returns a DWORD showing the offset inside the memory in bits. The returned DWORD value needs to be interpreted as hexadecimal. The first digit indicates which type of memory is used. The last digits show the offset in bits.



```
xVar AT %IX0.3 : BOOL;
xVar2 AT %MX1.3 : BOOL;
xVar3 AT %MX1.4 : BOOL;
```



```
dwBitoffset1 := BITADR(xVar);
dwBitoffset2 := BITADR(xVar2);
dwBitoffset3 := BITADR(xVar3);
```



dwBitoffset1	DWORD	16#80000003
dwBitoffset2	DWORD	16#4000000B
dwBitoffset3	DWORD	16#4000000C

On the left side the declaration is visible. In the middle the call of the BITADR operator in ST. On the right side the resulting DWORD values.

IX0.3 is in the input range. The value starts with 8 and has an offset of 3, which is the last digit.

MX1.3 is in the Modbus range. The value starts with 4 and has an offset of 11 which is one byte (8 bits) and 3 bits.

MX1.4 is similar to MX1.3 above also in the Modbus range and with one bit more offset so 0xC = 12d.