**MSP430FR2355 I/O PINS:**

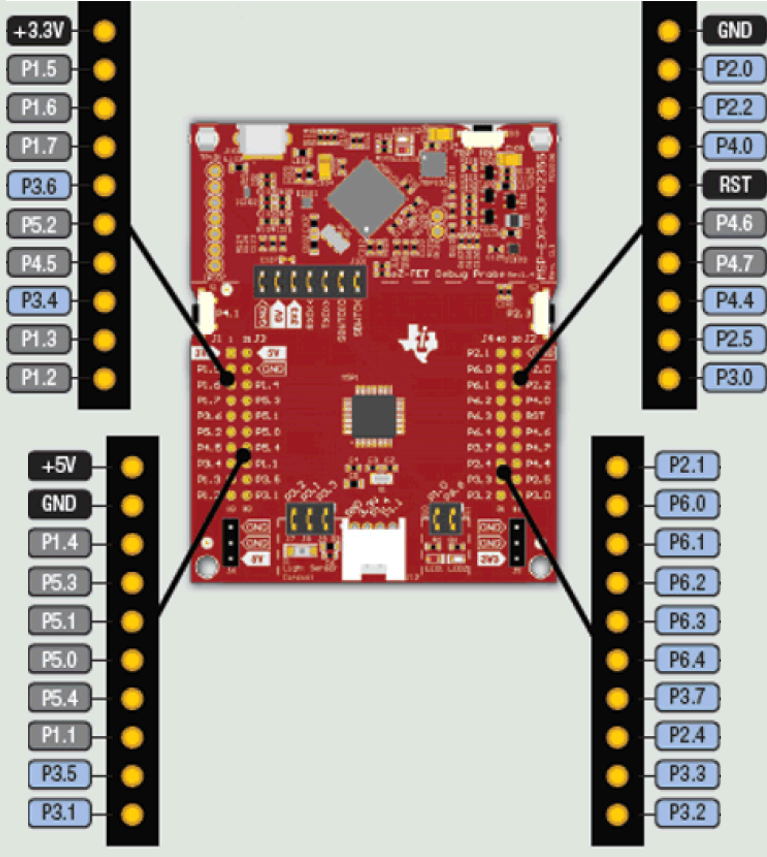
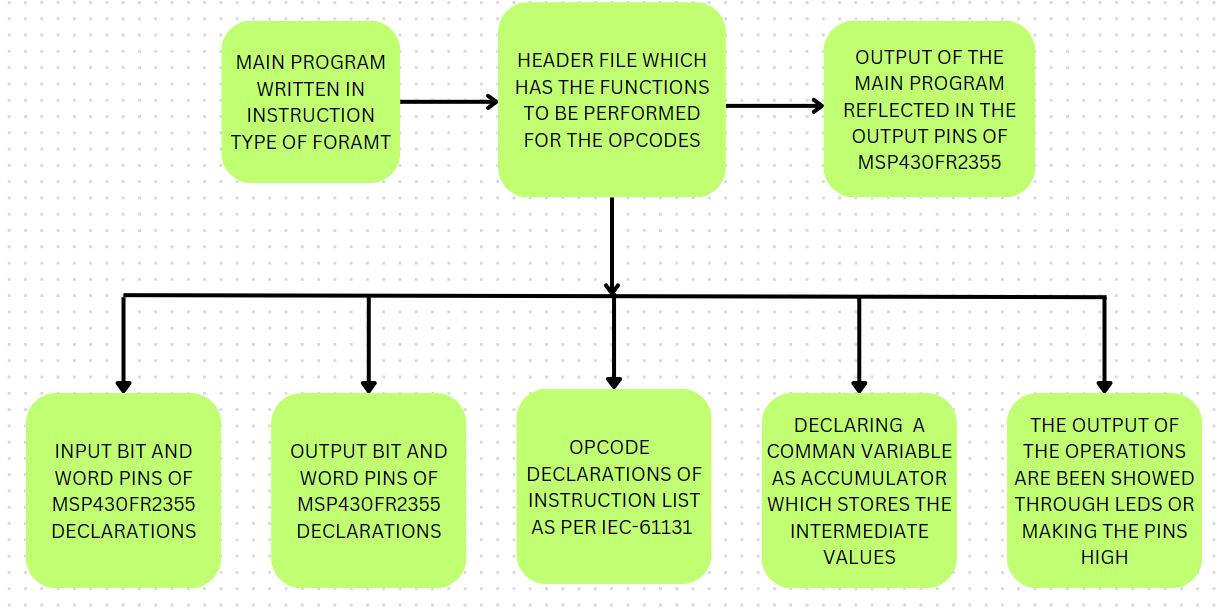


FIGURE 1: THE PINS CONFIGURED AS INPUT AND OUTPUT **I\*=INPUT , O\*=OUTPUT,NC=NOT CONFIGURED**

**BLOCK DIAGRAM:**

****

**INSTRUCTIONS AND THEIR FUNCTIONS:**

|  |  |  |
| --- | --- | --- |
| **OPCODE IN INSTRUCTION LIST (IEC-61131-3)** | **OPCODE USED IN C PROGRAM FOR MSP430** | **FUNCTIONS PERFORMED** |
| LD %IX2.1 | LD IX(2,1) | Loading the particular input value to the accumulator. And IX21() means input port 2.1 value. |
| ST %QX1.0 | ST(1,0) | Store it sends the logic value(0/1) of the accumulator to that particular O/P pin here P1.0. |
| AND %IX2.1 | AND IX(2,1) | AND is used to perform bitwise AND operation with the accumulator value and operand. |
| OR %IX2.1 | OR IX(2,1) | OR is used to perform bitwise OR operation with the accumulator value and operand. |
| XOR %IX2.1 | XOR IX(2,1) | XOR is used to perform bitwise XOR operation with the accumulator value and operand. |
| ANDN %IX2.1 | ANDN(IX(2,1)) | ANDN is used to perform bitwise AND operation with the accumulator value and negated(inverted) operand value. |
| ORN %IX2.1 | ORN(IX(2,1)) | ORN is used to perform bitwise OR operation with the accumulator value and negated(inverted) operand value. |
| XORN %IX2.1 | XORN(IX(2,1)) | XORN is used to perform bitwise XOR operation with the accumulator value and negated(inverted) operand value. |
| S %QX1.0 | SET(1,0) | SET operation is done to set that is logic 1 to a output pin |
| R %QX1.0 | RESET(1,0) | RESET operation is done to reset that is logic 1 to a output pin |
| OUT TR0 | \_OUT(TR0) | \_OUT operation is used to store the accumulator value to a temporary register. In this case TR0. |
| ADD A | ADD(EN,A,B) | ADD operation has 3 inputs one enable (EN) and other 2 are operands ,only if EN is high then only addition will happen and is stored in accumulator. [accumulator=A+B] |
| SUB B | SUB(EN,A,B) | SUB operation has 3 inputs one enable (EN) and other 2 are operands ,only if EN is high then only subtraction will happen and is stored in accumulator. [accumulator=A-B] |
| MOV C | MOV(EN,A,B) | MOV operation has 3 inputs and one enable (EN) and other 2 are operands ,only if EN is high then only move operation will happen in which the A variable value will be stored in the B variable. [B=A] |

**INITAIL CONFIGURATIONS:**

* Include the MSP PLC file “#include <msp430plc.h>” in the main program.
* Do copy the header file in the project folder where your main program is written.

**INSTRUCTION LISTS IN DETAIL:**

***LOAD OPERATION:***

The load operation used in the c program is in the format.

Syntax:

LD IX(port number, pin number); // IX meaning a input pin

LD IX(3,1);

Which means take the input value of port 2.1 and load that bit value in the accumulator.

Example:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(3,1);

}

**return** 0;

}

If that particular bit value is high then logic 1 will be stored in the accumulator else logic 0 will be stored in the accumulator.

***STORE OPERATION:***

The store operation used in c program is in the format.

Syntax:

ST(port number, pin number);

ST(1,0);

Which means store the accumulator value to that corresponding output pin. If the accumulator is high then that pin is also high. Here the output is stored in port 1 pin 0 (P1.0).

Example 1:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

ST(1,0);

}

**return** 0;

}

Example 2:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

ST(4,5);

}

**return** 0;

}

In the first example if the accumulator value is true then this pin P1.0 will be energized.

In the second example if the accumulator value is true then this pin P4.5 will be energized.

***AND OPERATION:***

The AND operation used in c program is in the format.

Syntax:

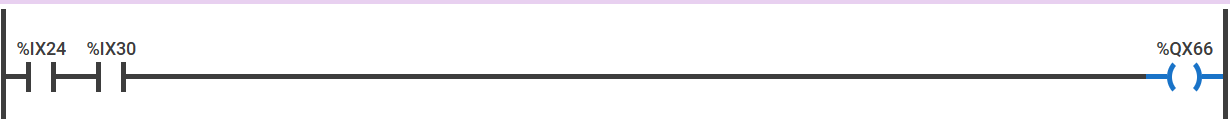
AND IX(port number, pin number); // IX meaning a input pin

AND IX(2,1);

Which means do bitwise AND operation with the accumulator value and the operand value given in the instruction and store that value in the accumulator.

Example:

Ladder logic :



Corresponding C program code to run in MSP430:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(2,4);

AND IX(3,0);

ST(6,6);

}

**return** 0;

}

In this example the load operation loads that value of pin 2.4 to the accumulator and AND operation takes the input from P3.0 and does bitwise AND operation with accumulator value and stores the result in the accumulator and the ST opcode stores the value of the accumulator to the output pin P6.6.

***OR OPERATION:***

The OR operation used in c program is in the format.

Syntax:

OR IX(port number, pin number); // IX meaning a input pin

OR IX(3,7);

Which means do bitwise OR operation with the accumulator value and the operand value given in the instruction and store that value in the accumulator.

Example:

Ladder logic :



Corresponding C program code to run in MSP430:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(3,7);

OR IX(3,0);

ST(6,6);

}

**return** 0;

}

In this example the load operation loads the value of P3.7 to the accumulator and the OR operand does bitwise OR operation with the operand value and the accumulator and stores the value in the accumulator and the ST opcode stores the value of the accumulator to the output pin P6.6.

***XOR OPERATION:***

The XOR operation used in c program is in the format.

Syntax:

XOR IX(port number, pin number); // IX meaning a input pin

XOR IX(3,7);

Which means do bitwise XOR operation with the accumulator value and the operand value given in the instruction and store that value in the accumulator.

Example:

Ladder logic :



Corresponding C program code to run in MSP430:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(3,7);

XOR IX(3,0);

ST(6,6);

}

**return** 0;

}

In this example the load operation loads the value of P3.7 to the accumulator and the XOR operand does bitwise XOR operation with the operand value and the accumulator and stores the value in the accumulator and the ST opcode stores the value of the accumulator to the output pin P6.6.

***ANDN OPERATION:***

The ANDN operation used in c program is in the format.

Syntax:

ANDN IX(port number, pin number); // IX meaning a input pin

ANDN IX(2,1);

Which means do bitwise AND operation with the accumulator value and the negation/inverted operand value given in the instruction and store that value in the accumulator.

Example:

Ladder logic :



Corresponding C program code to run in MSP430:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(2,4);

ANDN IX(3,0);

ST(6,6);

}

**return** 0;

}

In this example the load operation loads that value of pin 2.4 to the accumulator and ANDN operation takes the input from P3.0 and inverts the input and does bitwise AND operation with accumulator value and stores the result in the accumulator and the ST opcode stores the value of the accumulator to the output pin P6.6.

***ORN OPERATION:***

The ORN operation used in c program is in the format.

Syntax:

ORN IX(port number, pin number); // IX meaning a input pin

ORN IX(2,1);

Which means do bitwise OR operation with the accumulator value and the negation/inverted operand value given in the instruction and store that value in the accumulator.

Example:

Ladder logic :



Corresponding C program code to run in MSP430:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(2,4);

ORN IX(3,0);

ST(6,6);

}

**return** 0;

}

In this example the load operation loads that value of pin 2.4 to the accumulator and ORN operation takes the input from P3.0 and inverts the input and does bitwise OR operation with accumulator value and stores the result in the accumulator and the ST opcode stores the value of the accumulator to the output pin P6.6.

***XORN OPERATION:***

The XORN operation used in c program is in the format.

Syntax:

XORN IX(port number, pin number); // IX meaning a input pin

XORN IX(2,1);

Which means do bitwise XOR operation with the accumulator value and the negation/inverted operand value given in the instruction and store that value in the accumulator.

Example:

Ladder logic :



Corresponding C program code to run in MSP430:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(2,4);

XORN IX(3,0);

ST(6,6);

}

**return** 0;

}

In this example the load operation loads that value of pin 2.4 to the accumulator and XORN operation takes the input from P3.0 and inverts the input and does bitwise XOR operation with accumulator value and stores the result in the accumulator and the ST opcode stores the value of the accumulator to the output pin P6.6.

***SET OPERATION:***

SET operation used in the C program is in the format:

Syntax:

SET (port number, pin number); // IX meaning a input pin

SET(1,0);

Which means make the corresponding output pin high.

Example 1:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

SET(1,0);

}

**return** 0;

}

This operation will set the output pin P1.0 as high.

Example 2:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

SET(6,6);

}

**return** 0;

}

This operation will set the output pin P6.6 as high.

***RESET OPERATION:***

RESET operation used in the C program is in the format:

Syntax:

RESET (port number, pin number); // IX meaning a input pin

RESET(1,0);

Which means make the corresponding output pin low.

Example 1:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

RESET(1,0);

}

**return** 0;

}

This operation will set the output pin P1.0 as low.

Example 2:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

RESET(6,6);

}

**return** 0;

}

This operation will set the output pin P6.6 as low.

***\_OUT OPERATION:***

This operation is to store accumulator value in a temporary register.

“**There are 16 temporary registers starting from TR0 till TR15 each of 16 bit”.**

**Where TR0-TR9 are unsigned integer and TR10-TR20 are signed integer registers.**

Syntax:

\_OUT(variable/register);

\_OUT(TR2);

This instruction will store the accumulator value to the TR2 temporary register.

Example:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(2,4);

\_OUT(TR0);

}

**return** 0;

}

In this example the load instruction takes the value from the input pin P2.4 and the \_OUT opcode stores the accumulator value in the TR0 temporary register. This can be used while doing complex operation.

***ADD OPERATION:***

This operation is to adds the value of the 2 values only if the EN value is true and stores it in a variable named **output**.

Syntax:

ADD(enable pin,val1,val2);

ADD(accumulator,20,30); // here accumulator is the enable pin , you can give 1 directly also instead of giving accumulator.

ADD(1,20,30); // the first argument is to enable the addition operation.

Example:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(2,4);

ADD(accumulator,30,40);

}

**return** 0;

}

In this example the load instruction takes the value from the input pin P2.4 and the ADD function takes 3 arguments the enabler,value1,value2 and the result is stored in the output variable.

Example 2:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(2,4);

ADD(1,30,40);

MOV(output,TR10)

}

**return** 0;

}

Here instead of giving accumulator as enabler pin 1 is given directly and the addition operation of 2 operands will happen and the result is stored in the output. And with the MOV operand (from,to) which will copy the value from output to TR10 register.

***SUB OPERATION:***

This operation takes the difference of the 2 values only if the EN value is true and stores it in a variable named **output**.

Syntax:

SUB(enable pin,val1,val2);

SUB(accumulator,20,30); // here accumulator is the enable pin , you can give 1 directly also instead of giving accumulator.

SUB(1,20,30); // the first argument is to enable the addition operation.

“**NOTE THE OPERATION PERFORMED IS 20-30”** for the above example.

Example 1:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(2,4);

SUB(accumulator,30,40);

}

**return** 0;

}

In this example the load instruction takes the value from the input pin P2.4 and the SUB function takes 3 arguments the enabler,value1,value2 and the result is stored in the output variable. ”**NOTE: output=value1-value2”.**

Example 2:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(2,4);

SUB(1,30,40);

MOV(output,TR10)

}

**return** 0;

}

Here instead of giving accumulator as enabler pin 1 is given directly and the difference of 2 operands will happen and the result is stored in the output. And with the MOV operand (from variable ,to variable) which will copy the value from output to TR10 register.

***MOV OPERATION:***

This operation is used to move the value from one variable/register to another only if EN value is true.

Syntax:

MOV( from variable, to variable);

MOV(TR0,TR15); // this will move the data from TR0,TR15 only if accumulator is true.

Example 1:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(2,4);

MOV(TR0,TR10)

}

**return** 0;

}

In this example the value of TR0 is copied to TR10.

Example 2:

**#include** <msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(2,4);

MOV(accumulator,TR10)

}

**return** 0;

}

In this example the accumulator value of TR0 is copied to TR10 register.

THE HEADER FILE <msp430plc.h>

**#ifndef** MSP430PLC\_H\_

**#define** MSP430PLC\_H\_

**#include** <msp430.h>

**unsigned** **int** accumulator;

**unsigned** **int** TR0=0;

**unsigned** **int** TR1=0;

**unsigned** **int** TR2=0;

**unsigned** **int** TR3=0;

**unsigned** **int** TR4=0;

**unsigned** **int** TR5=0;

**unsigned** **int** TR6=0;

**unsigned** **int** TR7=0;

**unsigned** **int** TR8=0;

**unsigned** **int** TR9=0;

**int** TR10=0;

**int** TR11=0;

**int** TR12=0;

**int** TR13=0;

**int** TR14=0;

**int** TR15=0;

**int** TR16=0;

**int** TR17=0;

**int** TR18=0;

**int** TR19=0;

**int** TR20=0;

**int** output;

**int** BIT[15]={0x0001,0x0002,0x0004,0x0008,0x0010,0x0020,0x0040,0x0080,0x0100,0x0200,0x0400,0x0800,0x1000,0x2000,

0x4000,0x8000};

**unsigned** **int** inter=0;

**inline** **int** **IX**(a,b)

{

PM5CTL0 &= ~LOCKLPM5;

**if**(a==1)

{

P1DIR &=~BIT[b];

P1REN |=BIT[b];

P1OUT |=BIT[b];

**if**(P1IN & BIT[b]){

**return** 1;}

**else**{

**return** 0;}}

**if**(a==2)

{

P2DIR &=~BIT[b];

P2REN |=BIT[b];

P2OUT |=BIT[b];

**if**(P2IN & BIT[b]){

**return** 1;}

**else**{

**return** 0;}}

**if**(a==3)

{

P3DIR &=~BIT[b];

P3REN |=BIT[b];

P3OUT |=BIT[b];

**if**(P3IN & BIT[b]){

**return** 1;}

**else**{

**return** 0;}}

**if**(a==4)

{

P4DIR &=~BIT[b];

P4REN |=BIT[b];

P4OUT |=BIT[b];

**if**(P4IN & BIT[b]){

**return** 1;}

**else**{

**return** 0;}}

**if**(a==5)

{

P5DIR &=~BIT[b];

P5REN |=BIT[b];

P5OUT |=BIT[b];

**if**(P5IN & BIT[b]){

**return** 1;}

**else**{

**return** 0;}}

**if**(a==6)

{

P6DIR &=~BIT[b];

P6REN |=BIT[b];

P6OUT |=BIT[b];

**if**(P6IN & BIT[b]){

**return** 1;}

**else**{

**return** 0;}}

}

**inline** **void** **ANDN**(**int** (\*x)(**int**))

{

inter =(\*x);

inter=~inter;

accumulator= accumulator & inter;

}

**inline** **void** **ORN**(**int** (\*x)(**int**))

{

inter =(\*x);

inter=~inter;

**if**(inter==65534)

{

inter=0;

accumulator= accumulator | inter;

}

**else** **if**(inter==65535)

{

inter=1;

accumulator= accumulator | inter;

}

}

**inline** **void** **XORN**(**int** (\*x)(**int**))

{

inter =(\*x);

inter=~inter;

**if**(inter==65534)

{

inter=0;

accumulator= accumulator ^ inter;

}

**else** **if**(inter==65535)

{

inter=1;

accumulator= accumulator ^ inter;

}

}

**inline** **void** **ST**(a,b)

{

PM5CTL0 &= ~LOCKLPM5;

**if**(a==1)

{

P1DIR |=BIT[b];

**if**(accumulator)

{

P1OUT |= BIT[b];

}

**else**

{

P1OUT &=~ BIT[b];

}

}

**if**(a==2)

{

P2DIR |=BIT[b];

**if**(accumulator)

{

P2OUT |= BIT[b];

}

**else**

{

P2OUT &=~ BIT[b];

}

}

**if**(a==3)

{

P3DIR |=BIT[b];

**if**(accumulator)

{

P3OUT |= BIT[b];

}

**else**

{

P3OUT &=~ BIT[b];

}

}

**if**(a==4)

{

P4DIR |=BIT[b];

**if**(accumulator)

{

P4OUT |= BIT[b];

}

**else**

{

P4OUT &=~ BIT[b];

}

}

**if**(a==5)

{

P5DIR |=BIT[b];

**if**(accumulator)

{

P5OUT |= BIT[b];

}

**else**

{

P5OUT &=~ BIT[b];

}

}

**if**(a==6)

{

P6DIR |=BIT[b];

**if**(accumulator)

{

P6OUT |= BIT[b];

}

**else**

{

P6OUT &=~ BIT[b];

}

}

}

**inline** **void** **SET**(a,b)

{

PM5CTL0 &= ~LOCKLPM5;

**if**(a==1)

{

P1OUT|=BIT[b];

}

**if**(a==2)

{

P2OUT|=BIT[b];

}

**if**(a==3)

{

P3OUT|=BIT[b];

}

**if**(a==4)

{

P4OUT|=BIT[b];

}

**if**(a==5)

{

P5OUT|=BIT[b];

}

**if**(a==6)

{

P6OUT|=BIT[b];

}

}

**inline** **void** **RESET**(a,b)

{

PM5CTL0 &= ~LOCKLPM5;

**if**(a==1)

{

P1OUT&=~BIT[b];

}

**if**(a==2)

{

P2OUT&=~BIT[b];

}

**if**(a==3)

{

P3OUT&=~BIT[b];

}

**if**(a==4)

{

P4OUT&=~BIT[b];

}

**if**(a==5)

{

P5OUT&=~BIT[b];

}

**if**(a==6)

{

P6OUT&=~BIT[b];

}

}

**#define** \_OUT(x) x=accumulator

**#define** LD accumulator=

**inline** **void** **LDN**(**int** (\*x)(**int**))

{

inter =(\*x);

inter=~inter;

**if**(inter==65534)

{

inter=0;

accumulator=inter;

}

**else** **if**(inter==65535)

{

inter=1;

accumulator=inter;

}

}

**#define** AND accumulator=accumulator &

**#define** OR accumulator=accumulator |

**#define** XOR accumulator=accumulator ^

**inline** **void** **ADD**(EN,A,B)

{

**if**(EN)

{

output=A+B;

}

}

**inline** **void** **SUB**(EN,A,B)

{

**if**(EN)

{

output=A-B;

}

}

**#define** MOV(A,B) **do**{\

**if**(accumulator)\

{\

B=A;\

}\

}**while**(0)

**#endif** /\* MSP430PLC\_H\_ \*/

**Examples:**

****

**#include**<msp430plc.h>

**int** **main**()

{

**while**(1)

{

LD IX(1,1);

OR IX(1,2);

OR IX(1,3);

ORN(IX(1,4));

AND IX(1,5);

ST(1,0);

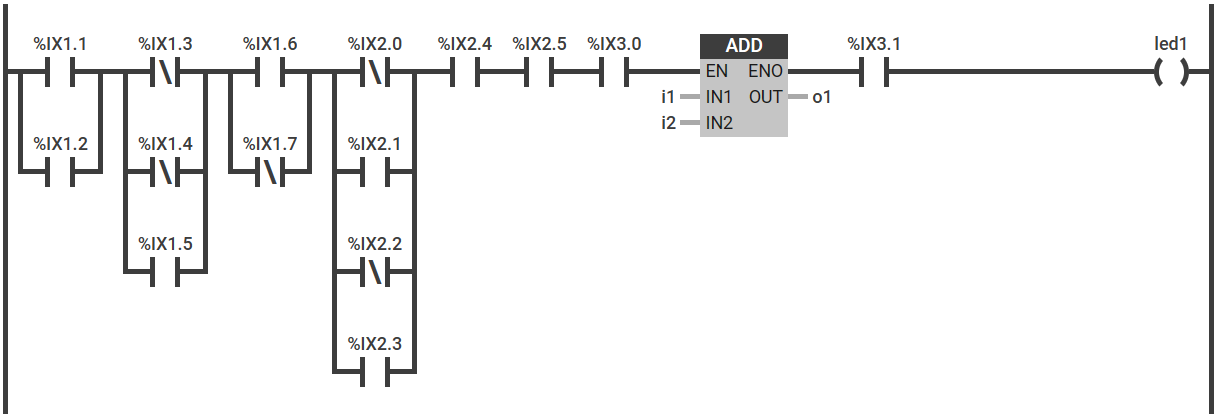
}

**return** 0;

}

**Example 2:**

%QX6.6

****

**#include** <msp430plc.h>

/\*\*

\* main.c

\*/

**int** **main**(**void**)

{

WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

**while**(1)

{

LD IX(1,1);

OR IX(1,2);

\_OUT(TR0);

LDN(IX(1,3));

ORN(IX(1,4));

OR IX(1,5);

AND TR0;

\_OUT(TR1);

LD IX(1,6);

ORN(IX(1,7));

AND TR1;

\_OUT(TR0);

LDN(IX(2,0));

OR IX(2,1);

ORN(IX(2,2));

OR IX(2,3);

AND TR0;

AND IX(2,4);

AND IX(2,5);

AND IX(3,0);

ADD(accumulator,10,20);

AND IX(3,1);

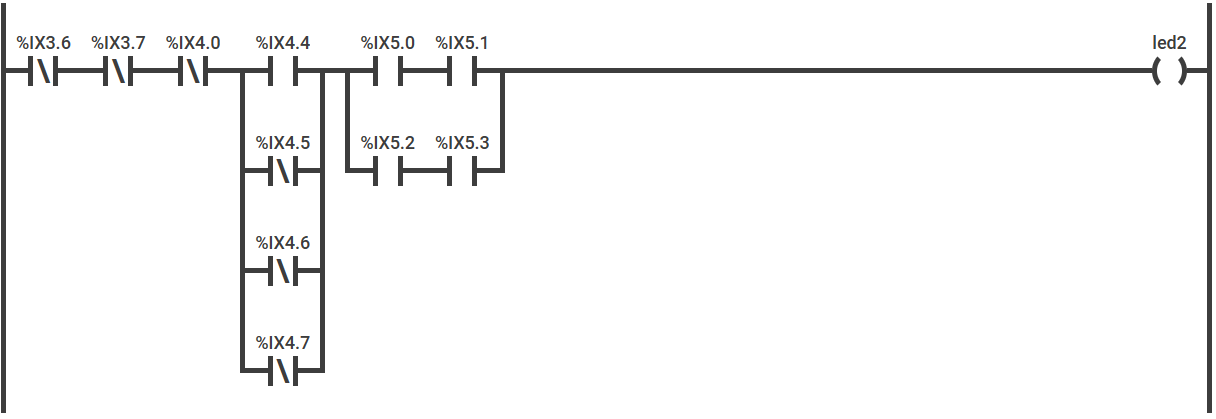
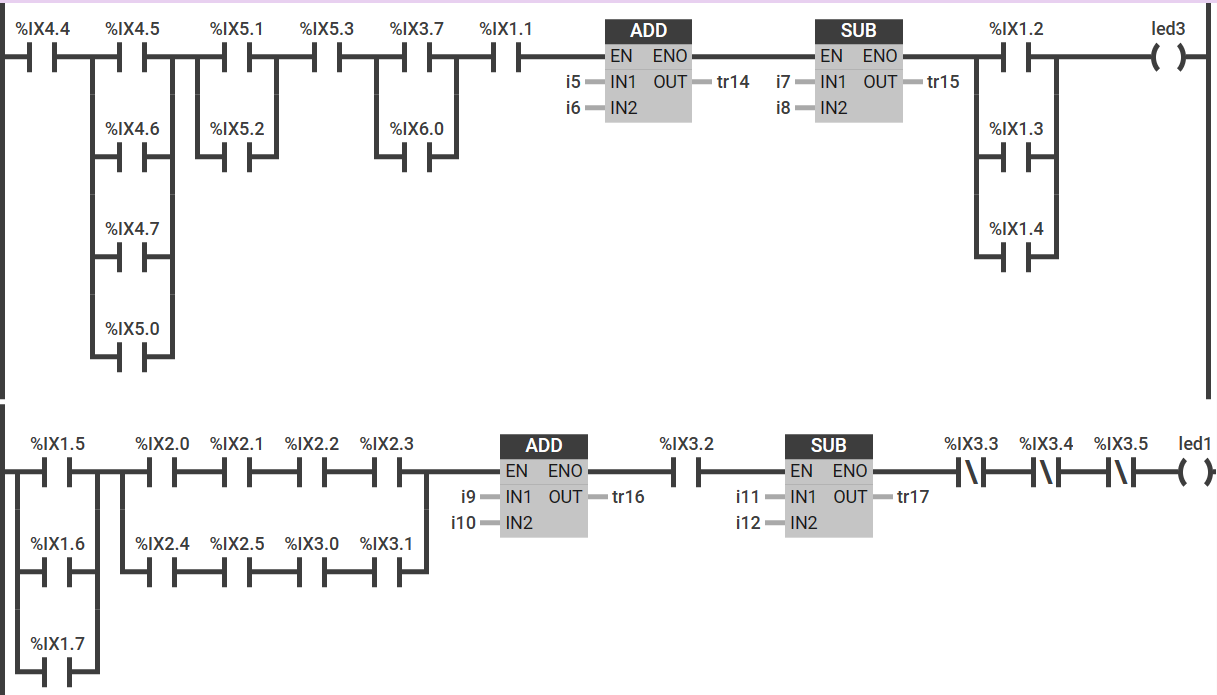
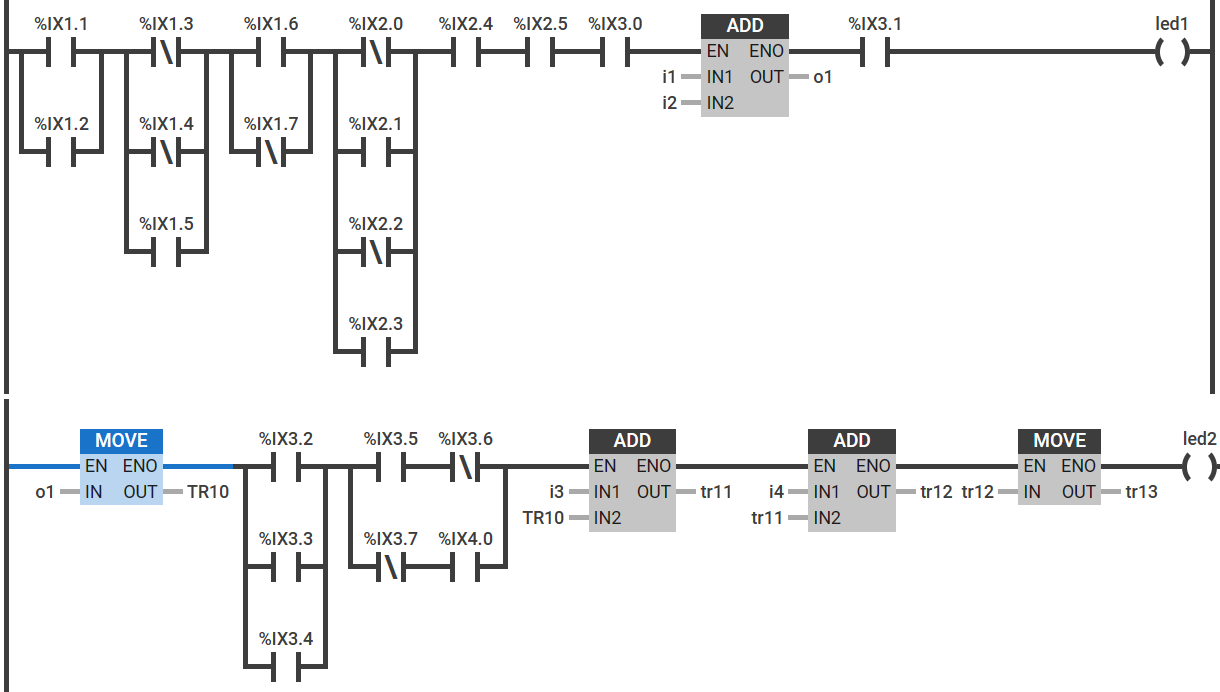
ST(6,6);

}

return 0;

}

**Example 3:**

****

Code:

**#include** <msp430plc.h>

/\*\*

\* main.c

\*/

**int** **main**(**void**)

{

WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

**while**(1)

{

LD IX(1,1);

OR IX(1,2);

\_OUT(TR0);

LDN(IX(1,3));

ORN(IX(1,4));

OR IX(1,5);

AND TR0;

\_OUT(TR1);

LD IX(1,6);

ORN(IX(1,7));

AND TR1;

\_OUT(TR0);

LDN(IX(2,0));

OR IX(2,1);

ORN(IX(2,2));

OR IX(2,3);

AND TR0;

AND IX(2,4);

AND IX(2,5);

AND IX(3,0);

ADD(accumulator,10,20);

AND IX(3,1);

ST(6,6);

MOV(output,TR10);

LD IX(3,2);

OR IX(3,3);

OR IX(3,4);

\_OUT(TR2);

LD IX(3,5);

ANDN(IX(3,6));

\_OUT(TR3);

LDN(IX(3,7));

AND IX(4,0);

OR TR3;

AND TR2;

ADD(accumulator,1000,2000);

MOV(output,TR11);

ADD(accumulator,7000,TR11);

MOV(output,TR12);

MOV(TR12,TR13);

ST(1,0);

LD IX(4,4);

\_OUT(TR4);

LD IX(4,5);

OR IX(4,6);

OR IX(4,7);

OR IX(5,0);

AND TR4;

\_OUT(TR5);

LD IX(5,1);

OR IX(5,2);

AND TR5;

AND IX(5,3);

\_OUT(TR6);

LD IX(5,4);

OR IX(6,0);

AND TR6;

AND IX(1,1);

ADD(accumulator,10,20);

MOV(output,TR14);

SUB(accumulator,output,1000);

MOV(output,TR15);

\_OUT(TR7);

LD IX(1,2);

OR IX(1,3);

OR IX(1,4);

AND TR7;

ST(6,4);

LD IX(1,5);

OR IX(1,6);

OR IX(1,7);

\_OUT(TR8);

LD IX(2,0);

AND IX(2,1);

AND IX(2,2);

AND IX(2,3);

\_OUT(TR9);

LD IX(2,4);

AND IX(2,5);

AND IX(3,0);

AND IX(3,1);

OR TR9;

AND TR8;

ADD(accumulator,1000,3000);

MOV(output,TR16);

AND IX(3,2);

SUB(accumulator,100,500);

MOV(output,TR17);

ANDN(IX(3,3));

ANDN(IX(3,4));

ANDN(IX(3,5));

ST(6,6);

LDN(IX(3,6));

ANDN(IX(3,7));

ANDN(IX(4,0));

\_OUT(TR0);

LD IX(4,4);

ORN(IX(4,5));

ORN(IX(4,6));

ORN(IX(4,7));

AND TR0;

\_OUT(TR1);

LD IX(5,0);

AND IX(5,1);

\_OUT(TR2);

LD IX(5,2);

AND IX(5,3);

OR TR2;

AND TR1;

ST(1,0);

}

**return** 0;

}

**Ways by which we can use this header file for other MSP microcontrollers:**

There are some minor changes which is needed to be done in the msp430plc.h file those are:

* Firstly the programmer should have an idea about how many ports are there in the particular MSP board which he is trying to implement PLC.
* Next step is to open the header file and scroll down to the line no 35 where is a function named IX in that there are 6 ports defined by default if in case we want to increase the number of ports then we can increase by writing the following code below line no 96 in the header file.

For example I have taken port 7:

**if**(a==7)

{

P7DIR &=~BIT[b];

P7REN |=BIT[b];

P7OUT |=BIT[b];

**if**(P7IN & BIT[b])

{

**return** 1;

}

**else**

{

**return** 0;

}

}

Like this if there many ports we can increase by using this method. This is specifically for input.

* Same way we have to write for output for that in the header file the function named ST and then go to line no 224 and write the following code below if the board has more than 6 ports .

**if**(a==7)

{

P7DIR |=BIT[b];

**if**(accumulator)

{

P7OUT |= BIT[b];

}

**else**

{

P7OUT &=~ BIT[b];

}

}

For example, I have taken port 7 and we can do the same if we have more ports.

**Things to note for optimum working of MSP430 as a PLC:**

* Make sure you have the header file msp430plc.h file saved in the same project folder where you are using the header file.
* Make sure about the number of ports your particular MSP board has if it has less than or equal to 6 ports then the same header file can be used if in a case there are more then 6 ports then follow the steps above and do the required changes in the header file and then proceed.

**Limitations:**

* It can perform only BIT operations but cannot perform / evaluate word operations that is basically performed through the analog pins.
* It performs only limited operations like addition, subtraction, move, store, load.