Graphical Programming for PIC® Microcontrollers.

---BitCraft-----

Introduction

Most electronic enthusiast/hobbyist will agree that microcontrollers are very versatile and powerful marvels of technology. They are also inexpensive and it is easy to build a working circuit. Generating and testing software for them is another story, especially if you are not reasonably familiar with any of the programming languages and your application is a bit more involved than just flashing an LED. In which case there is a good chance of your ideas remaining just ideas.

This document can maybe help, not by teaching you the use of one of the popular programming languages, but by describing an alternative approach and a graphics tool for generating and debugging software without writing a single line of code

The first part of this document is designed to introduce you to the use of function blocks to produce a software design for your microcontroller. The concept is simple, so those of you expecting some brand new technology, you will be disappointed. Function blocks are very successfully used for programming industrial automation controllers

Where to Start with a Project

A method used by many when starting with a project is to first make a basic functional diagram of the general workings of the application, nothing complicated, just to show the basics. You will be amazed how this first step can serve to generate ideas. Just a few blocks on paper can say a lot; after all, a picture is still worth a thousand words.

Our example application's function is to measure the ambient temperature and to warn us if any alarm condition is detected. Here is a bit more detail of what we want our system to do.

> Alarm1 LED on when Temperature > 40 DegC
> Alarm2 LED on when Temperature < 20 DegC

Each active alarm condition will be indicated by an LED, and every time any one of the alarms become active it will switch on a buzzer, which will remain on (even if the alarm condition goes away) until it is silenced by operating a push button.
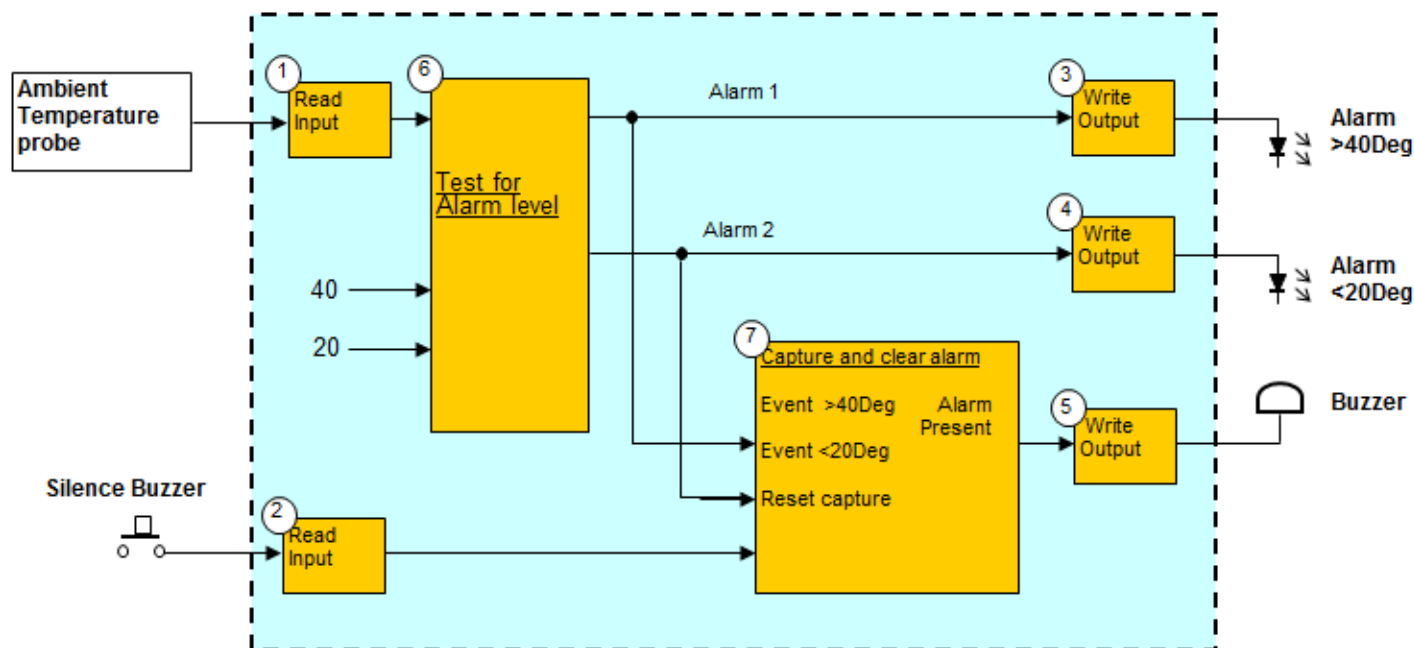

Figure 1 Application Basic Functional Diagram

sometimes referred to as PLC's. If you think about it most of our microcontroller applications do some sort of control, like reading one or more inputs, do some processing, and then use the results to drive some output device(s), just like the industrial controller. So the idea here is to show how the use of function blocks can simplify things considerably when making software for microcontrollers, even if it is just to make an LED flash! We will use a simple application to show how to arrive at a software design based on function blocks.

The second part of this document describes the basics of a PC application that will allow you to generate the software for your PIC® microcontroller application. A CAD based approach is used where graphic symbols, representing function blocks, are dropped on one or more pages and then inter-connected by lines to form your complete application on screen. For testing your project live values from your microcontroller can be displayed on the 'drawing'.

There is a lot of information available on the internet about the technicalities and anatomy of function blocks as used in programming, so in this document we will rather concentrate on the practical application of this design approach to generate software for our application.

Figure 1 is my version of the basic functional diagram for our project. Stick to a few general rules when drawing the basic functional diagram:

1. Inputs into the microcontroller chip are located on the left side of the diagram and outputs from the chip on the right side of the diagram.
2. Function/Processing blocks receive input signals on their left side and their result(s) appear on their right side. This means a signal/value propagate through the diagram from left to right
3. The description of a Function/Processing block only states what it is doing, not how it is doing it.

Figure 1 show how the signals from the input devices are transferred by 'Read Input' Blocks 1 and 2 to the processing Blocks 6 and 7, and also how the 'Write Output' Blocks 3 to 5 drive the output devices. Take note that Figure 1 is not a circuit diagram, it only shows the basics.

The workings of Blocks 2 to 5 are straight forward; they handle signals that can only be ON or OFF. Block 1 on the other hand handles a numeric value representing the temperature. The specification for the temperature probe

states that it will produce 10mVolt for every 1 DegC above zero degrees. (Refer to LM35). For example if the ambient temperature is 20 DegC the probe will supply 200mVolt. For our software to 'read' the temperature value the voltage signal from the temperature probe must be connected to a pin on the microcontroller equipped with an Analog-to-Digital (A/D) converter. In our case the A/D converter of the microcontrollers will convert an input signal ranging from 0 to 5Volt, into a numeric count ranging from 0 – 1023, that is for a 10bit A/D converter. With a bit of mathematic manipulation you can show that the ambient temperature (in DegC) is given by:

Ambient temperature = 2.046 * (A/D converter count)

The functionality required for Block 1 in Figure 1 can now be expressed by using simple function blocks as shown in Figure 2. Remember you don't need to now how a function block is working; only what it is doing.

From Block 1a, the A/D converter, we obtain an 'integer' in the range 0 – 1023 representing the ambient temperature. With the aid of Block 1b this integer value is converted into a 'real' (floating point) value which is then multiplied by 2.046 in Block 1c to produce the ambient temperature in DegC.
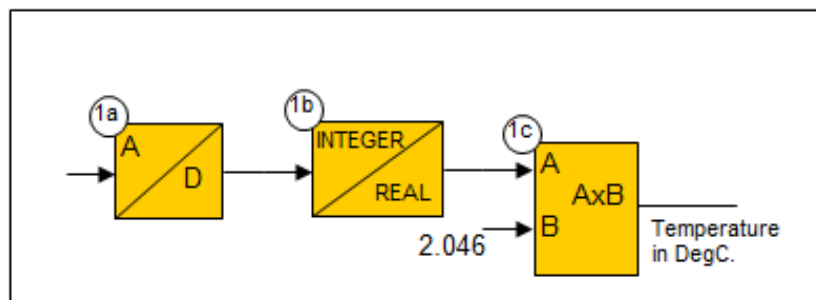

Figure 2 Ambient Temperature Input Conversion.

At this stage you might ask: What function blocks are available to use in the design? As far as I know there is no fixed standard, but there seem to be general consensus that at least the following should be provided for in a controller:

Logic functions
(AND, OR, XOR, SR-LATCH)

Timing functions
(ON-DELAY, OFF-DELAY, MONOSTABLE.)

Mathematical function
(ADD, SUBTRACT, MULTIPLY, DIVIDE)

Comparator functions
(>, <, =)

Input and Output functions
(ANALOG-IN, ANALOG-OUT, DIGITAL-IN, DIGITAL-OUT)

In more involved designs it is not unusual to have one or more intermediate stages of functional designs, with the last stage using the function blocks provided by the specific controller.

Let us get back to our project by looking at Block 6 in Figure 1. This block must generate the 2 alarm conditions

depending on the current temperature value. Comparator function blocks are used as shown in Figure 3, and I have taken the liberty of including in the figure the Blocks 3 and 4 to show how the alarm signals Alarm 1 and Alarm 3 are used to drive the output pins to which are connected the LED's.
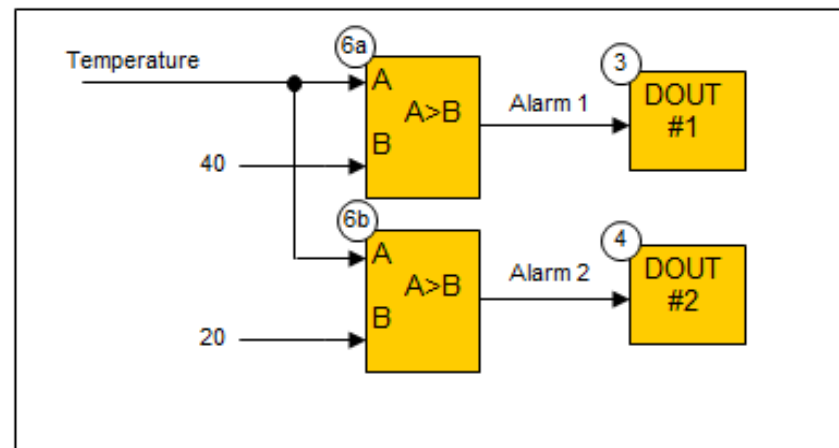

Figure 3 Test for Alarm Condition.

Block 7 in Figure 1 requires that when any of the alarm conditions occur the buzzer is to sound continuously until it is silenced by the *Silence Buzzer* push button, ready to be triggered by the next alarm occurrence. In Figure 4 the Blocks 7a and 7b will each output a short pulse when they detect a 0-to-1 transition at their respective inputs. Any of these pulses will be transferred via Block 7c to the S(et) input of the SR-Latch (Block 7d). The output of the latch, driving Block 5, will remain set until the Reset Alarm push button is momentarily operated, which will convey a Logic 1 signal to the R(eset) input of Block 7d, causing it to clear its output, thus silencing the buzzer.
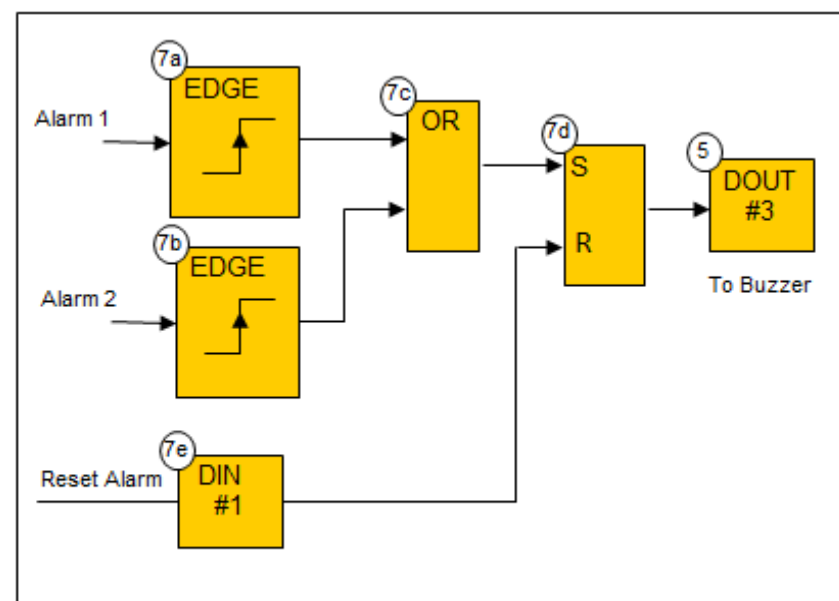
Figure 4 Capture and clear Alarm

That concludes our function block design. As a bonus you now also have your application documented; something which is usually left for last, or worse, never done.

Next step is to generate the software that implements our functional design. You can of course now proceed to write code for the function blocks in Figures 2, 3, and 4, using your favorite programming language, but there is an easy way out, just read the next section.

Software Development with VPS_P18

In the section above was described a method of using function blocks to arrive at a design for the software of a microcontroller project. If now you write the code for each function block then after a few projects you will probably have quite a handy 'library' of function blocks that you can re-use in new projects. In actual fact this is how VPS_P18 started.

VPS_P18 is a PC CAD application that will allow you to 'draw' the application in much the same form as that what we ended up with in the first section of this document. The function blocks available are listed at the end of this document. Courtesy of Microchip™, VPS_P18 produces a .HEX file, complete with an operating system, ready to be programmed into your microcontroller. Once the program is 'burned' into the microcontroller it is time for debugging and testing. Using the microcontroller's serial port VPS_P18 will display on your application drawing live values (100mSec update rate, if your PC is fast enough) for the inputs and outputs of each function block. This means, for instance, that you can follow the effect a pushbutton input signal has 'inside' the application and not just if it produces the desired result at some output pin. Debugging and testing is further made easier when using the 'trend view' in VPS_P18. With this 'oscilloscope type' tool you can view 400 values of up to 4 variables sampled at an adjustable rate of multiples of 100 mSec.

Communication between PIC® and PC is via RS232. You can use a USB/RS232 converter if your PC does not have a serial port.

VPS_P18 generates complete, ready-to-run HEX code for applications for the following PIC® microcontrollers from Microchip™.

PIC18F242
PIC18F2420
PIC18F252
PIC18F2520
PIC18F442
PIC18F4420
PIC18F452
PIC18F4520

The following figure is a screenshot from a code-page of VPS_P18 showing the implementation of our temperature alarm project. The sections for Figures 2, 3, and 4 are identified by relevant comment blocks. For those interested in this kind of information, the project occupies 426* bytes of program memory and takes on average 584 processor cycles, which translates to 73uSec execution time.
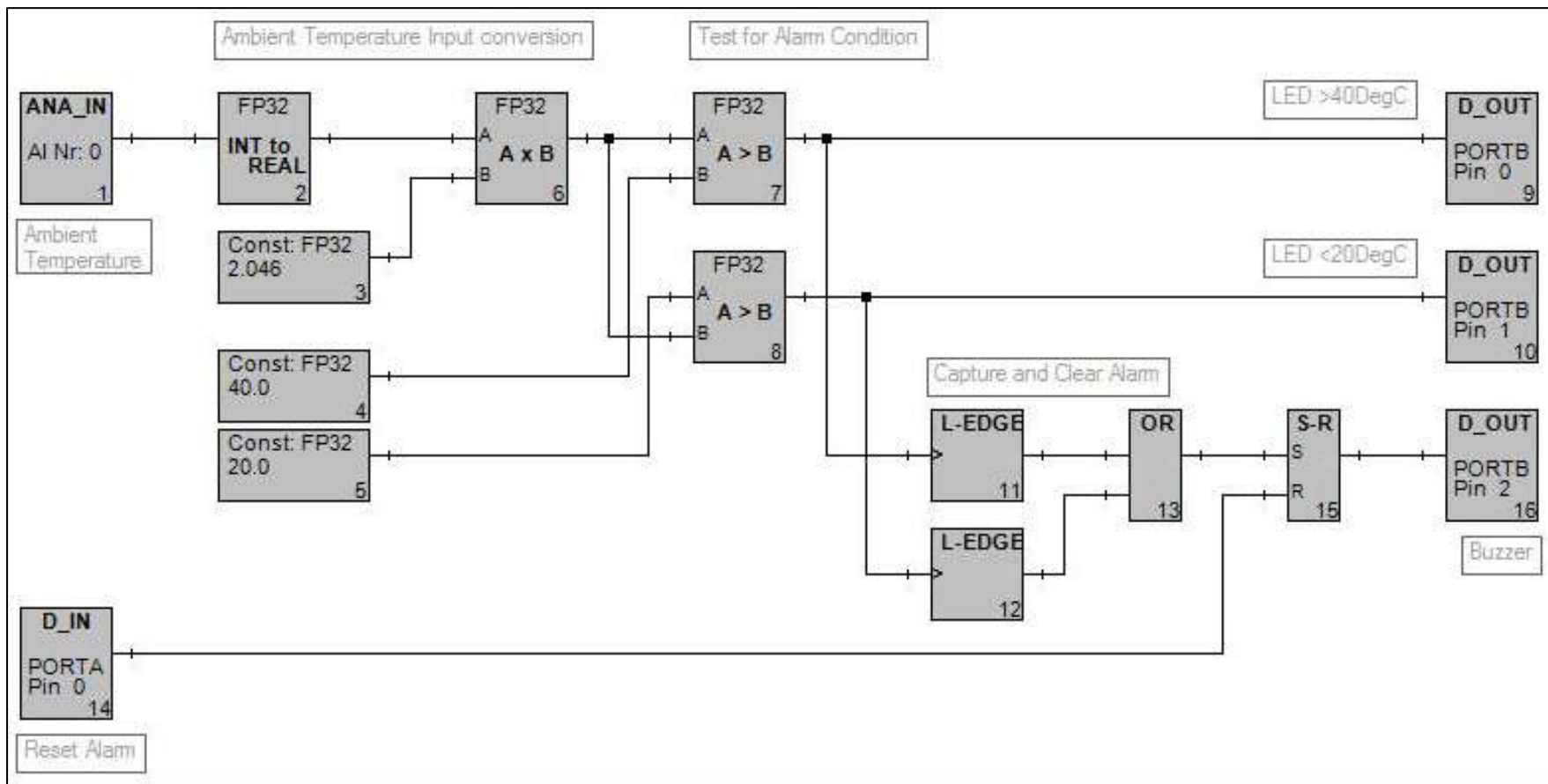
VPS_P18 was made by a hobbyist for hobbyists, so I welcome any suggestions, questions and critic. For those who want more information or wants to use the program it is available at no charge when used for non-commercial purposes. Contact Lourens at bitcraft@global.co.za.

Happy PicKing.

*This is of course without the memory required for the base software and operating system.

VPS_P18 Function Block library

The following are the Function Blocks available in VPS_P18 Ver 1.5

Logic Functions:

Reference	Description
FB1	AND Gate 2-Input
FB2	AND Gate 3-Input
FB3	AND Gate 4-Input
FB7	AND Gate 8-Input
FB8	OR Gate 2-Input
FB9	OR Gate 3-Input
FB10	OR Gate 4-Input
FB14	OR Gate 8-Input
FB15	SR Latch
FB16	XOR Gate 2-Input
FB17	D-LATCH with Reset
FB18	TRAILING EDGE Detect
FB19	LEADING EDGE Detect
FB138	Byte AND 2-Input
FB139	Byte OR 2-Input
FB140	Byte XOR 2-Input
FB141	Byte INVert

Timing Functions:

Reference	Description
FB30	ON-Delay Timer
FB31	OFF-Delay Timer
FB32	MONO-Stable Timer
FB33	MONO-Stable Timer retriggerable

Mathematical Functions:

Reference	Description
FB120	ADD INT Values
FB102	ADD FLOATING POINT Values
FB122	SUBTRACT INT Values
FB103	SUBTRACT FLOATING POINT Values
FB124	MULTIPLY INT Values
FB101	MULTIPLY FLOATING POINT Values
FB126	DIVIDE INT Values
FB100	DIVIDE FLOATING POINT Values
FB105	ABSOLUTE VALUE of FLOATING POINT Value
FB129	ABSOLUTE VALUE of INT Value
FB170	INTEGRATOR for INT Values
FB171	DIFFERENTIATOR for INT Values

Comparator Functions:

Reference	Description
FB59	COMPARE for BYTE Values >, =, <
FB92	COMPARE for INT Values >, =, <
FB60	TEST if A>=B for INT Values
FB61	TEST if A=B for INT Values
FB64	TEST if A<B for FLOATING POINT Values
FB65	TEST if A>B for FLOATING POINT Values

Variable Test Functions:

Reference	Description
FB93	TEST INT Value for +ve, =0, -ve
FB98	TEST FLOATING POINT Value for +ve, =0, -ve

Counter Functions:

Reference	Description
FB39	8BIT UP-DOWN COUNTER with Limits
FB40	16BIT UP-DOWN COUNTER with Limits

Selector/Multiplexor Functions:

Reference	Description
FB20	CHANGE-OVER-SWITCH for BOOLEAN values
FB56	CHANGE-OVER-SWITCH for BYTE values
FB72	CHANGE-OVER-SWITCH for INT values
FB78	CHANGE-OVER-SWITCH for FLOATING POINT values

FB41	MUX for 1-of-8 BYTE Literals
FB42	MUX for 1-of-8 INT Literals
FB43	MUX for 1-of-8 UINT Literals
FB44	MUX for 1-of-8 FLOATING POINT Literals
FB79	SELECT 1-of-4 FLOATING POINT Values
FB70	SELECT MAXIMUM of 2 INT Values
FB71	SELECT MINIMUM of 2 INT Values
FB76	SELECT MAXIMUM of 2 FLOATING POINT Values
FB77	SELECT MINIMUM of 2 FLOATING POINT Values
FB108	SELECT 1-of-8 BYTE Literal values
FB109	SELECT 1-of-8 INT Literal values
FB110	SELECT 1-of-8 UINT Literal values
FB111	SELECT 1-of-8 FLOATING POINT Literal values

Limiter Functions:

Reference	Description
FB83	HIGH LIMITER for INT Values
FB84	LOW LIMITER for INT Values
FB90	HIGH LIMIT DETECT for INT Values
FB91	LOW LIMIT DETECT for INT Values
FB94	HIGH LIMIT DETECT for FLOATING POINT Values
FB95	LOW LIMIT DETECT for FLOATING POINT Values

Table Look-up (Function Generator) Functions:

Reference	Description
FB168	FUNCTION GENERATOR INT Values (Input 0…1024, Output -32768…32767)
FB169	FUNCTION GENERATOR INT Values (Input 0…32767, Output -32768…32767)

Communication Functions:

Reference	Description
FB163	I2C WRITE (to Slave, 7Bit address)
FB164	I2C READ (from Slave, 7Bit address)

Input and Output Functions:

Reference	Description
FB25	ANALOG INPUT
FB26	FREQUENCY COUNTER INPUT (max 32000Hz)
FB27	PWM OUTPUT
FB28	DIGITAL OUTPUT
FB29	DIGITAL INPUT
FB142	DISPLAY BYTE Variable on LCD
FB143	DISPLAY INT Variable on LCD
FB145	DISPLAY FLOATING POINT Variable on LCD
FB146	DISPLAY STRING on LCD

Pulse Generator Functions:

Reference	Description
FB21	LOW FREQUENCY PULSE GENERATOR
FB211	FLASHER BITS (System Resource)

Data Pack/Unpack Functions:

Reference	Description
FB112	PACK 8 Bits into BYTE
FB113	PACK 2 BYTES into INT
FB114	PACK 2 BYTES into UINT
FB115	PACK 4 BYTES into FLOATING POINT
FB116	UNPACK BYTE into 8 Bits
FB117	UNPACK INT into 2 BYTES
FB118	UNPACK UINT into 2 BYTES
FB119	UNPACK FLOATING POINT into 4 BYTES

Read Data EEPROM Functions:

Reference	Description
FB45	READ BYTE Value from EEPROM
FB47	READ INT Value from EEPROM

FB49	READ UINT Value from EEPROM
FB51	READ FLOATING POINT Value from EEPROM

Data Type Conversion Functions:

Reference	Description
FB150	CONVERT INT to FLOATING POINT
FB151	CONVERT FLOATING POINT to INT
FB152	RANGE TRANSFORM INT/ FLOATING POINT
FB153	BINARY Byte to packed BCD
FB154	BINARY 2 Bytes to 1 Byte packed BCD
FB155	1 Byte packed BCD to BINARY Byte
FB156	1 Byte packed BCD to 2 Bytes BINARY

Sequence Control Functions:

Reference	Description
FB157	SEQUENCE CONTROLLER
FB158	SEQUENCE STEP HEAD
FB159	SEQUENCE STEP TAIL

Control Functions:

Reference	Description
FB66	INCREMENT/DECREMENT RATE LIMITER for INT Values
FB67	INC/DEC RAMP CONTROLLER
FB173	FILTER, (very) LOW PASS for INT Values
FB174	FILTER, LOW PASS for INT Values
FB175	LEAD FUNCTION for INT Values
FB176	LAG FUNCTION for INT Values
FB177	DEAD-TIME FUNCTION for INT Values
FB181	PID CONTROLLER standard interactive using velocity algorithm

Constants:

Reference	Description
FB195	DEFINE FLOATING POINT Constant
FB196	DEFINE UINT Constant
FB197	DEFINE INT Constant
FB198	DEFINE BYTE Constant
FB210	DEFINE BOOLEAN Constant (System Resource)

PIC® Device Functions and utilities:

Reference	Description
FB188	LOAD INDICATOR
FB189	SHOW MICROCONTROLLER DEVICE ID
FB190	SFR BYTE Read
FB191	SFR BIT Read
FB192	SFR BYTE Write
FB193	SFR BIT Write

Code-Page Functions:

Reference	Description
FB200	SIGNAL READ Connector (read signal from another, or same, Code Page)
FB201	SIGNAL WRITE Connector (write signal for use on another, or same, Code Page)
FB202	TEXT COMMENT BOX
FB203	SIGNAL WRITE Connector (for Sequence Output Signals)