Programming (some of) Microchip's PIC18F Processors with Pictures.

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 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 to generating software for your microcontroller. Might I add that this programming approach is not just for the novice – those well familiar with writing software can get a lot of 'horsepower' from a controller using the methods described here.

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 controllers used in industrial automation. Microcontrollers are also mainly use for receiving some input, do some processing, and then use the results to drive some output device(s).The use of function blocks can simplify things considerably when it comes to making software for them, 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 microcontroller application without 'writing' any code. A CAD based approach is used where graphic symbols, representing function blocks, are used to 'draw' your application on screen. For testing your project live values from your microcontroller is 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.

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 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 > 50 DegC
Alarm2 LED on when Temperature > 40 DegC
Alarm3 LED on when Temperature < 20 DegC

Each active alarm 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.

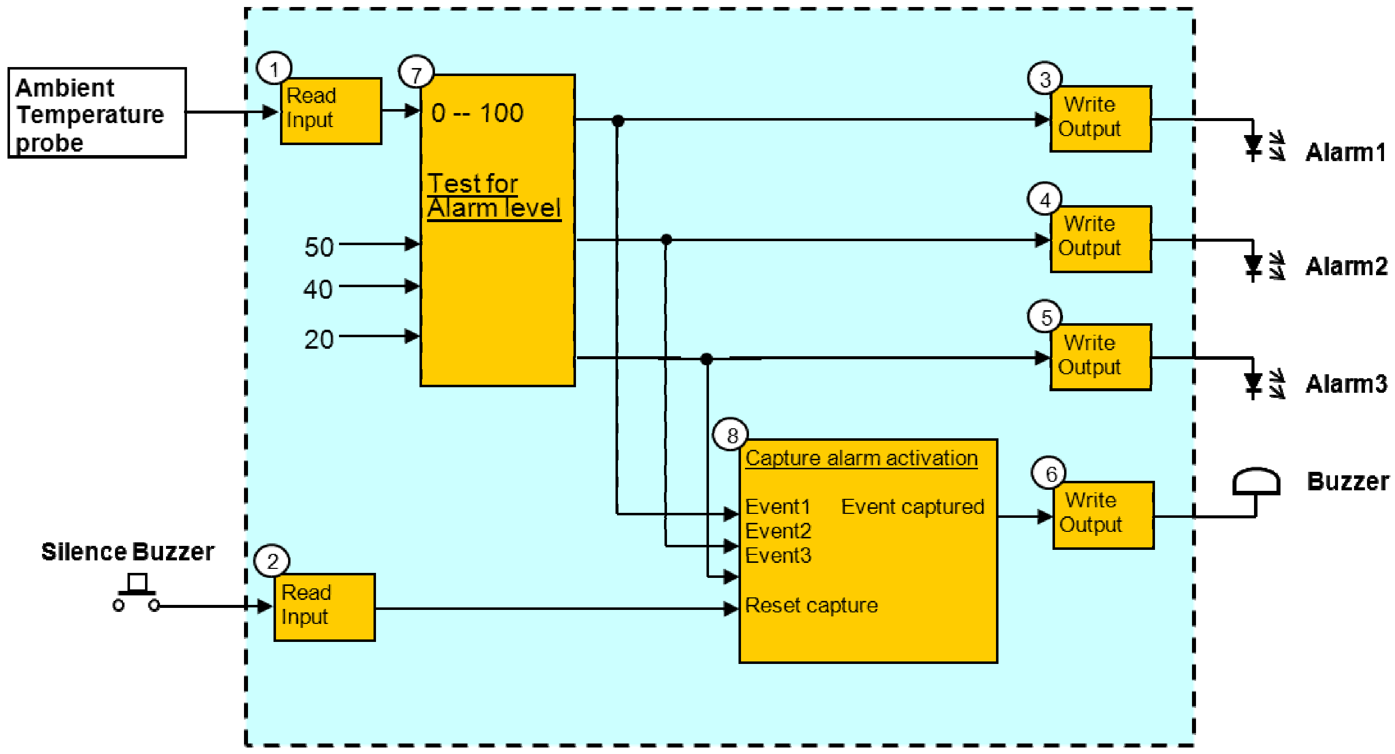


Fig. 1 Application Basic Functional Diagram

Figure 1 is my version of the basic functional diagram for our project. I 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. Signal/Data/Information flow directions are indicated with arrows.
4. 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 7 and 8, and also how the 'Write Output' Blocks 3 to 6 drive the output peripherals. Take note that **Figure 1** is not a circuit diagram, it only shows the basics.

The workings of Blocks 2 to 6 are straight forward; they handle signals that can 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. For example if the ambient temperature is 20 DegC the probe will supply 200mVolt. For our software to 'read' the temperature value the voltage 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 – 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.

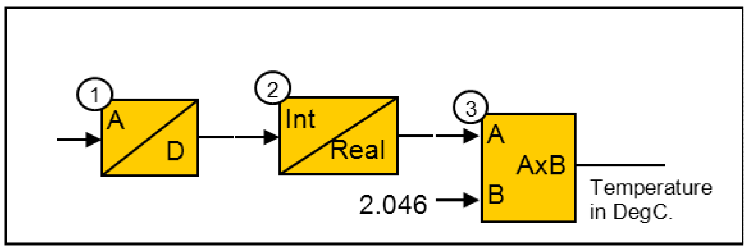


Fig. 2 Ambient Temperature Input Conversion.

From Block 1, the A/D converter, we obtain an 'integer' in the range 0 – 1023 representing the ambient temperature. With the aid of Block 2 this integer value is converted into a 'real' (floating point) value which is then multiplied by 2.046 in Block 3 to produce the ambient temperature in DegC.

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)

Also note that 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 7 in **Figure 1**. This block must generate the 3 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 4, 5, and 6 to show how the alarm signals Alarm 1, 2, and 3 are used to drive the output pins to which are connected the LED's.

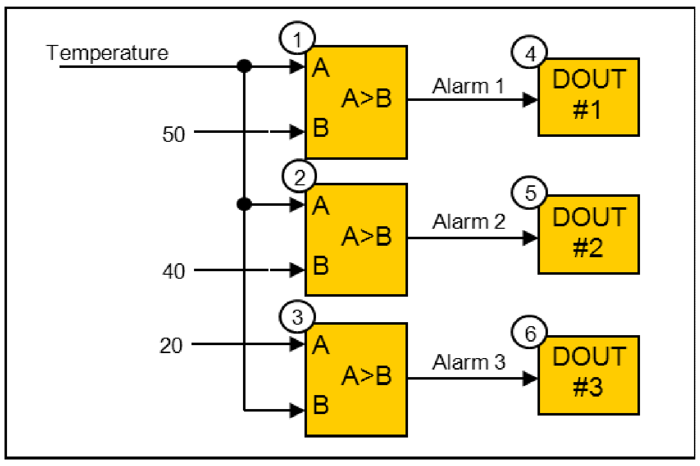


Fig. 3 Test for Alarm Condition.

Block 8 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 1, 2, and 3 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 5 to the S(et) input of the SR-Latch (Block 6). The latch output, driving the buzzer, will remain set until the Reset Alarm line connected to the R(eset) input of Block 6 is at Logic 1.

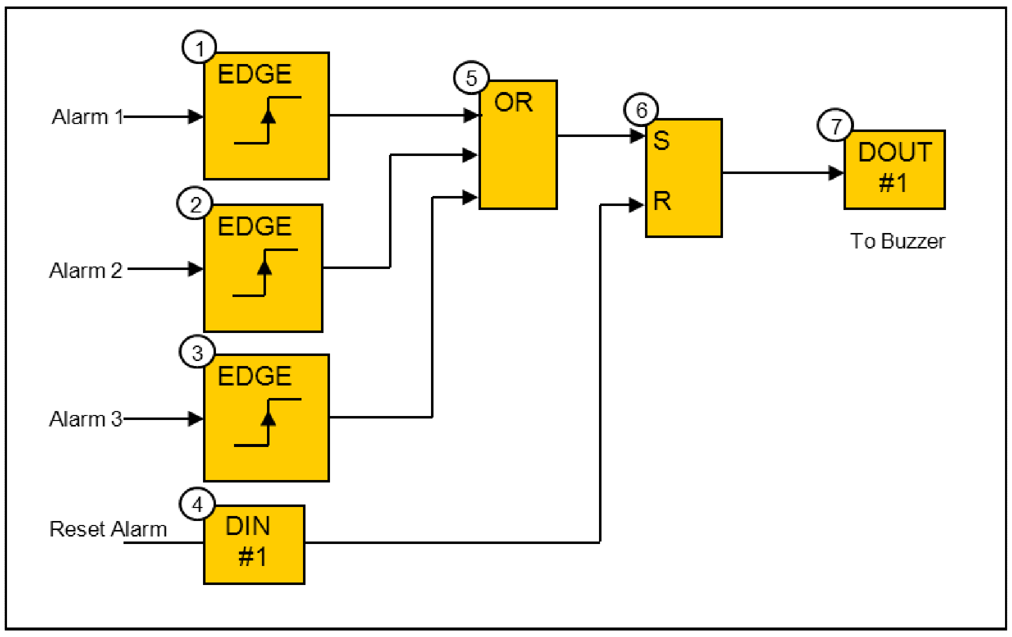


Fig. 4 Capture Alarm Activation

That concludes our function block design, with the added bonus that the documentation, which is usually left for last, or worse, never done, is available.

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 PIC01_CB

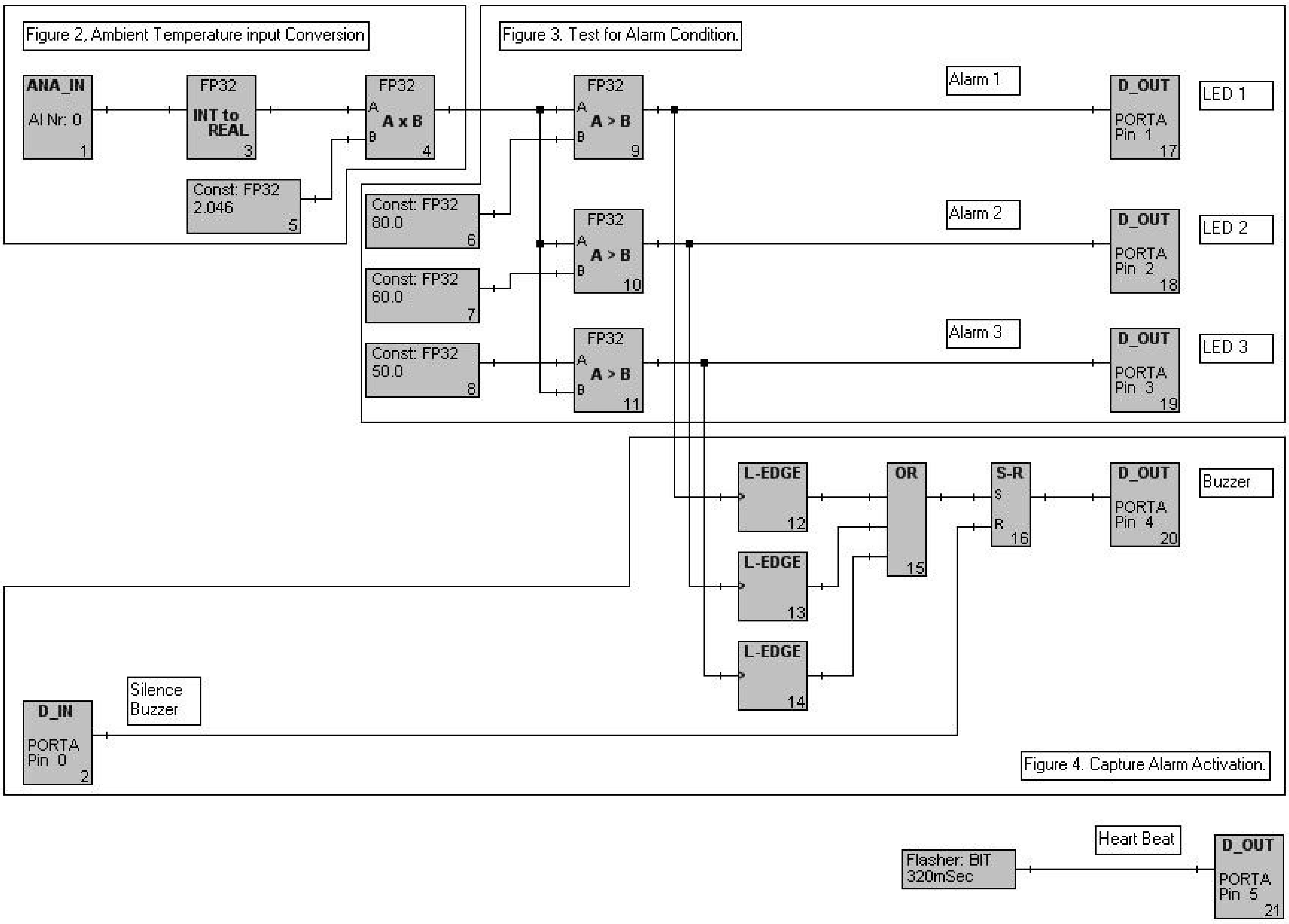
In the section above was described a method of using function blocks to arrive at a design for the software of a microcontroller project. If now you write the code for each function block then after a few projects you will probably have quite a handy 'library' of function blocks that you can re-use in new projects. In actual fact this is how PIC01_CB came into being.

PIC01_CB 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. PIC01_CB uses a library (see last section) of function block source code to compile your drawing(s) into the final software code, 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. Here PIC01_CB will update your application drawing with live values 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 PIC01_CB. With this tool you can display 400 values of up to 5 variables sampled at a rate of 500 mSec.

PIC01_CB generates complete, ready-to-run HEX code for applications for the following microcontrollers from Microchip.

PIC18F242
PIC18F2420
PIC18F252
PIC18F2520
PIC18F442
PIC18F4420
PIC18F452
PIC18F4520

The following figure is a screenshot from a code-page of PIC01_CB showing the implementation of our temperature alarm project. The sections for **Figures 2**, **3**, and **4** are identified by relevant comment blocks. You might have noticed the two additional function blocks at the right-hand bottom with the comment 'Heart Beat'. This is something I have added, firstly because I find it very reassuring to see a software-driven blinking light, and secondly to show you how easy it is to add a flashing LED to your project. By the way these two blocks contains the full functionality of my first PIC project of many moons ago.



Function Blocks, a link between requirement and solution.

PIC01_CB was made for hobbyists by a hobbyist, so I welcome any suggestions, questions and even critic. For those who want more information or wants to test-drive the program it is available at no charge. Contact Lourens at bitcraft@global.co.za.

Happy PicKing.

PIC01_CB Function Block library

The following are the Function Blocks available in PIC01_CB ver1.2.2

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

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

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
FB179	PID CONTROLLER (non interactive)

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:

Reference	Description
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)