Chapter 14

RECIPES

Edition 3

Recipe Overview

These blocks provide a means of implementing recipe systems within the PC3000. Recipes may created, edited, stored and loaded locally or from a supervisory system. The library comprises a control function block, the 'recipe manager', which is responsible for a number of `recipe slaves' which are assigned to it.

The recipe manager provides the recipe selection mechanism and provides the control interface.

The recipe slaves provide the recipe data storage; they are similar to 2 dimensional arrays but with fixed dimensions, 16 values wide by 128 values deep. A range of data types is supported:

Boolean, Real, Integer and String

A mechanism for time sequencing of a pre-defined list of recipes is also provided. This is called the Stage Manager.

Usage

There are two concepts behind this PC3000 recipe system.

The first is to provide a store of recipes without the need of a supervisory system. In this case the recipes are stored in the LCM RAM and are used when requested. The blocks allow for off-line editing of any recipe even when another one is selected.

The second use is with a supervisory system where the problem is the communications overhead downloading from one recipe to the next. All the recipes are downloaded from the supervisory system before the process starts. During the process the Stage_Manager is used to sequence the requested recipes in the correct order without the need of further communications.

In both of these cases the recipe data is stored in the LCM RAM and is therefore lost whenever the LCM is reset; for example after a rebuild and download. Until recipe data is entered there will be no valid recipes.

To store recipes over a reset, or cold start, the recipe data has to be stored somewhere. This is not a problem in the PC3000 with the supervisory system, containing a hard disc, but it could be a problem with the stand alone PC3000. In this case the File Store System is the only choice.

The Prog8 function blocks have their own in-built recipe system which incorporates the File Store and this should be used where appropriate.

The only other option is to generate a Macro in the application program which stores the recipe away regularly during normal operation and, on cold start, retrieves the data from the File System before resuming its normal functions.

In this case store the data in strings iusing the COMPACT functions REP_REAL_IN_STR and EXT-REAL-FROM-STR etc)

Definitions

Recipe:	A recipe is the state defined by all the outputs associated with a single recipe manager or stage manager.
Manager:	Manager on its own is used in this document to describe either a stage manager or a recipe manager function block.
Sequence:	The sequence is the list of recipes stored in the stage manager, including loops and end points.
Stage:	A stage is a single element of a sequence, consisting of a recipe number and loop information.
On-line/Live Recipe:	The on-line recipe is the set of values which affect the process.
Off-line Recipe:	The off-line recipe is the set of values used for changing the contents of the recipe without affecting the process.
Snapshot:	Snapshot is the term used to describe loading the live recipe values back into a recipe.

Example Application

A typical control strategy will be governed by a number of parameters of different types. As an example system, we will consider an analog control loop coupled with a digital output. The analog loop contains a ramp function block, the digital output is triggered by a timer. In a typical application, this set up will be repeated a number of times (e.g. the zones of an extruder). A single loop is shown in figure 14-1.

In this example, the parameters which control an individual batch are as follows:

Ramp Setpoint

Ramp Rate

Digital Output state

Batch Time

The function of a recipe system is to set up these parameters, and to synchronise the start/end of a batch. If a supervisory system were used, before the start of each batch, these parameters for each loop would have to be downloaded via a communications link to the PC3000. Once present in the PC3000, the supervisory system would then signal the start of the batch.

To implement the recipe system shown in figure 14-1 two types of function block are required. These are Recipe Managers and Recipe Slaves .

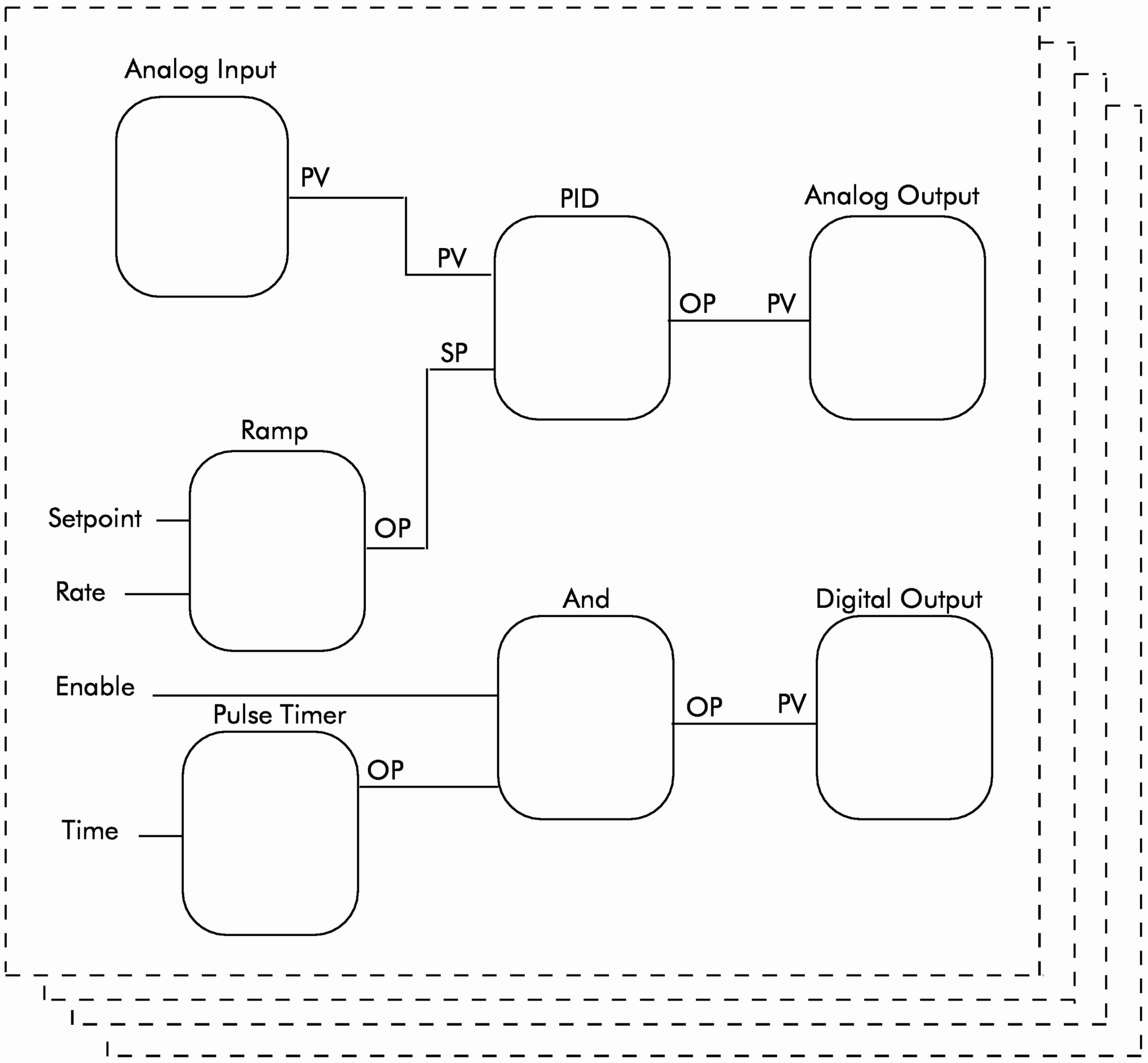


Figure 14-1 Recipe System Example

Recipe Slaves

The recipe slave function blocks form the data store of the recipe system. Recipe slaves are defined by their data type (e.g. Dint, Real), the number of outputs per block, and the number of recipes which can be stored. For example, a recipe block of type Real which supports 16 parameters and can store 128 recipes, is named Real _16 x 128

The recipe slave function blocks also provide the communications interface into the recipe system, by acting as communications slave variables.

Recipe Managers

The recipe manager controls recipe slaves. It governs which recipe they output. It also controls some recipe editing facilities.

In some applications, it is necessary to have more than one recipe system running at the same time. For example, combining three fluids in varying amounts may need temperature setpoints for the three fluids, which rarely change, and valve settings for the three fluids, which would change in every batch. Rather than having the temperature setpoints repeated in each recipe, we can split this into two recipes, one for the temperatures, another for the flow rates.

To facilitate this there is an identifier character, or ID, for each recipe system present in the PC3000. In each recipe system there will be one recipe manager and a number of recipe slaves, all of them with the same ID.

Each recipe manager must therefore have a unique Manager ID, in order to distinguish between the recipes.

In most applications, only one manager will be present. All recipe slaves will then have the same ID as that manager.

Recipe Managers

The recipe manager performs tasks associated with the global set of recipe slaves that it controls.

The recipe manager performs tasks associated with the global set of slaves associated with it. These include 'snapshotting' of the current recipe values (i.e.. loading the live values of the system into a recipe), clearing down individual recipes or the entire recipe store, changing the recipe currently output, and inhibiting changes to the recipe. The recipe manager has an ID used to associate particular recipe slave function blocks with specific managers, and a recipe number.

The recipe number input sets the recipe on the associated recipe slave function blocks' outputs. At first execution, the recipe slave function blocks declare themselves to the recipe managers. When the recipe manager recipe number input changes, the manager declares this new recipe to all of its associated slaves.

The set of tasks which the recipe manager can perform are as follows:

The manager can initiate a change to the current live recipe. This is done by either setting the Force input/output, or changing the recipe number. Setting the Force boolean re-outputs the current recipe.

The manager can initiate a load from the current live recipe back into the recipe store. In this way, the user can make modifications to the current recipe, and store them in a recipe on the PC3000. This is known as 'snapshotting'

The manager can declare an individual recipe as being invalid. A recipe is only deemed valid if there has been a recipe set up in it. An attempt to change to an invalid recipe will fail. This is done by the recipe manager requesting from each recipe slave the new recipe. On the recipe slave's next execution, it checks whether the recipe is valid and signals this to the manager. On the manager's next execution, if all the associated recipe slaves have valid data for this recipe, it will signal them all to change to it. Otherwise, none of the recipe slaves are signalled to change recipe.

The manager can output a recipe to the off-line values. The off-line values are used for recipe editing from the PC3000. They provide a mechanism of outputting, viewing, and editing recipes without affecting the live recipe.

The manager can load the off-line values into a recipe.

The manager can inhibit changes to the stored recipes.

Recipe Slaves

The recipe slaves perform the data handling for the recipe system. Internally they consist of a Communications buffer and a data storage array.

The recipe parameters are output to the user program through recipe slave function blocks. These each have an internal array of data relating to the individual recipes. Each is associated with an individual recipe manager, and has a comms address for configuration and monitoring purposes. Extra facilities are provided to allow a copy from one recipe to another, and to specify which values change in a particular recipe. Also they have a set of values related to the ``off-line" recipe, which can be used to edit the contents of the array without affecting the process.

On-line Recipe Values

The on-line recipe values are the parameters which should be connected to the system.

Off-line Recipe Values

The off-line values are parameters which can be changed without affecting the currently running process. This provides a simple facility for editing recipes through the PC3000 itself.

The Recipe Mask

There are some applications where values are set up at the start of a process run, but are altered manually throughout that process, either to fine tune the process prior to 'snapshotting' the live values, or because manual control is required. In this situation, it is necessary to output a recipe value once, at the start of the run, but not to overwrite the value as the recipe subsequently changes.

To provide this functionality, there is a system of recipe masks.

The value for a recipe will ONLY change if the mask bit is SET for that value.

For instance, a Real function block has a recipe mask for recipe number 20 of 15 (binary 0000000000001111). This means that when changing to recipe 20, the values on outputs 5 through 16 will remain unchanged. The values on outputs 1 to 4 will change.

Valid and Invalid Recipes

At cold start, all the recipes in the recipe slave will be invalid -they have no data in them, and it is not safe to output their values as a recipe. Once a recipe has been written to, either via comms, from loading the current On-line recipe, or loading the current Off-line recipe, that recipe in that slave is considered valid. If all of the slaves associated with a manger have a valid recipe at the same number, then that recipe as a whole is considered to be valid. To invalidate a given recipe, the Clear must be asserted, together with the required recipe number, by the associated manager.

Communications Buffer

Internally the recipe slave contains a 19 parameter multi-element slave variable. This is interpreted by the block as a load pointer, a destination pointer, a mask and 16 values.

Load Pointer

Writing a recipe number to the load pointer causes the recipe slave to copy that recipe (i.e. 16 values and a mask) into the recipe pointed at by the destination pointer for destination pointer values between 1 and 128. Anything greater than 128 or less than 0 is invalid. 0 is used as a special number to indicate that this transaction is a load from the value input/outputs.

Destination Pointer

The destination pointer decides which recipe is the subject of reads and writes. Recipe 0 is a special denoting the value input/outputs. In order to read the values of a specific recipe, the recipe number must first be written to the destination pointer.

Mask

This defines the output change mask for this particular recipe. The least significant bit relates to value 1, the most significant to value 16. (e.g. a mask value of 1000000000000001 binary will only allow updates of value 16 and value 1 for this recipe)

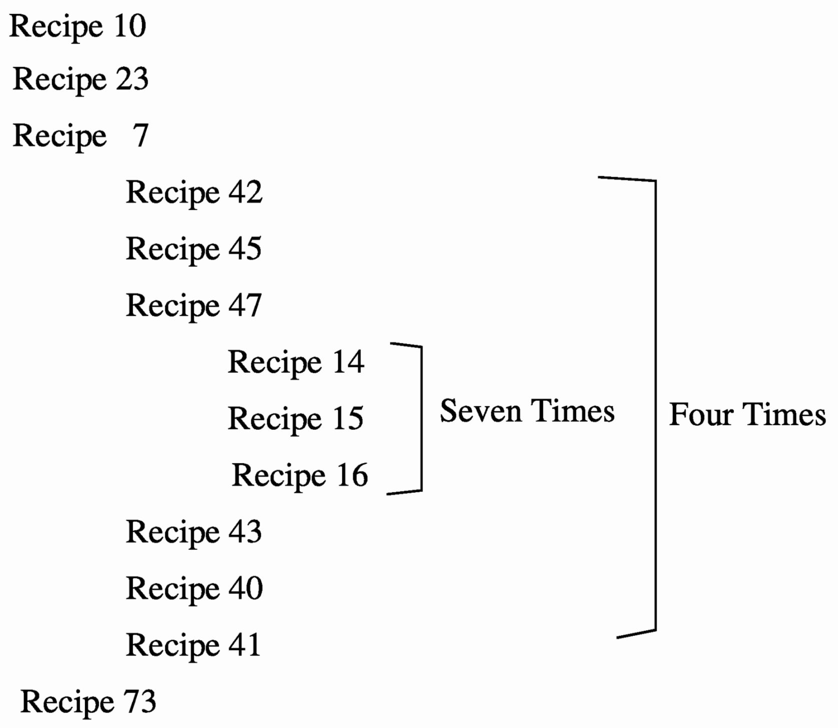
Values

After the mask come the 16 values for the current recipe. These can be left as default values, or overwritten. Defaults are null string for string types, and zero for real, boolean and integer types.

Stage Managers

The stage manager has an internal sequence list which is accessed over comms or through user program. When in auto mode, a rising edge on the clock input causes the internal recipe manager to move on to the next recipe from the sequence list. This list also supports up to four nested levels of looping.

In some systems, there is a requirement to pass through a number of recipes sequentially. For example, consider the following list of recipes:



With the stage manager function block, it is possible to specify a sequence of recipes in this way.

The function block view of the RecipeMan function block is shown in figure 14-7

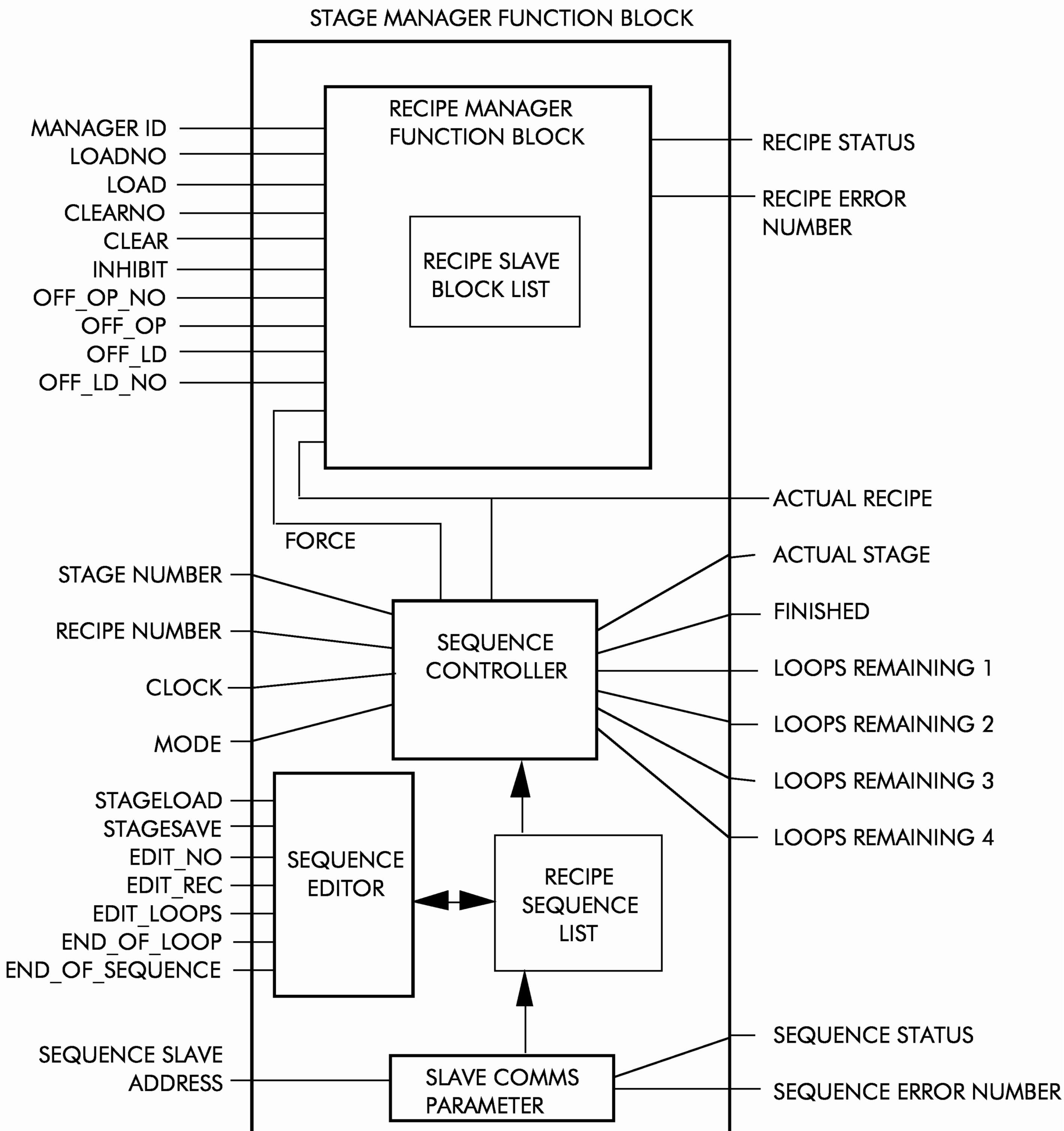


Figure 14-2 Stage Manager Function Block

Internally the stage manager has two functional parts. There is a recipe manager function block contained within the stage manager, and there is the sequence handler. The sequence handler's task is to decide the next recipe to output according to the sequence list. The sequence list is accessible over comms - see Chapter 3 Appendix C.

Each stage of the list contains the following information:

Stage Number:	The stage number is this stage's position in the list of stages. The sequence handler in the stage manager works from stage 1 through in ascending numerical order, apart from when it deals with loops.
Recipe Number:	The recipe number is the recipe to output for this stage.
Loop Count:	If no loop starts on this stage, the loop count is set to 0. If a loop starts on this stage, then the number of loops is set up in this field.
End Of Loop:	If this stage is the end of the current loop, then the end of loop flag is set.
End Of Sequence:	If this stage is the last in the sequence, then this flag is set.

For example, to implement the sequence shown previously:

Stage	**Recipe**	**Loops**	**Start of Loop**	**End of Sequence**
1	10	0	Off	Off
2	23	0	Off	Off
3	7	0	Off	Off
4	42	4	Off	Off
5	45	0	Off	Off
6	47	0	Off	Off
7	14	7	Off	Off
8	15	0	Off	Off
9	16	0	On	Off
10	43	0	Off	Off
11	40	0	Off	Off
12	41	0	On	Off
13	73	0	Off	On

Table 14-1 Stage Information

It is possible to jump to a stage in the sequence by setting StageNo parameter and setting mode to Jump(3). If the jump is to the middle of a loop, the end of loops will be ignored. For instance, if you jump to stage 8 in the above sequence, the stage manager will step through stage 8 to 13 in sequence, ignoring the end of loop at stage 9 and stage 12.

An Example using the PC3000 Recipe System

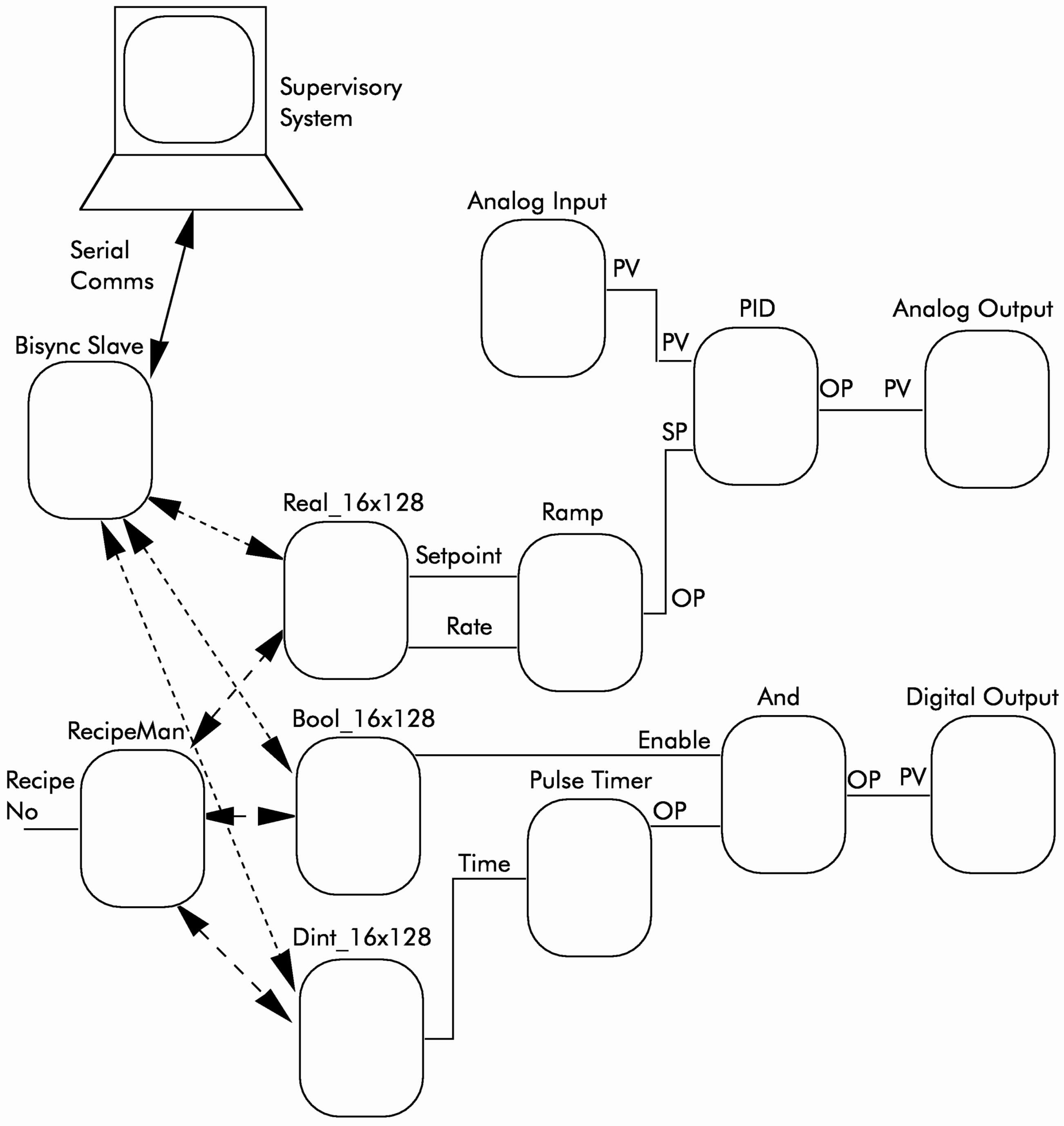


Figure 14-3 PC3000 Recipe System Example

Using the example system outlined on page 14-iv as an example, we need a single recipe manager, and recipe slaves of type Dint, Real and Bool. The recipe slaves all have the same Manager ID, so when the recipe number on the manager changes, the recipe on all the slave blocks changes.

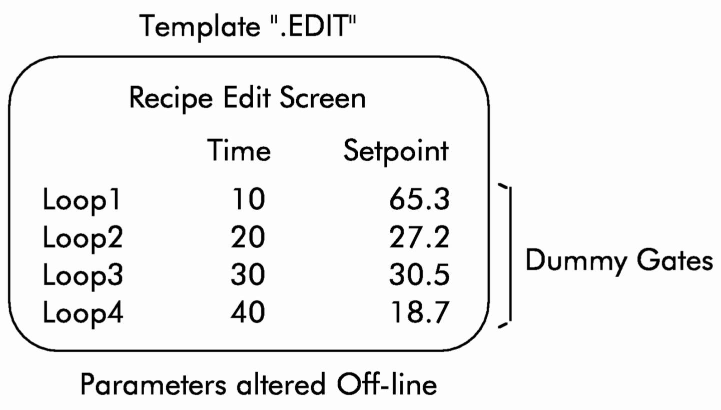
To facilitate communications to a supervisory system, an instance of a communications driver is required, for example the EI function block.

We give the RecipeMan function block an ID of '1', and all the recipe slave blocks a ManagerID of '1'.

For the slave block addresses, we use EI Bisync addresses. We give the Real an address of 'EBd00' the Bool an address of 'EBb00' and the Dint an address of 'EBd00'

The function block wiring is shown in figure 14-3. In a four loop system, we may use Val_1 and Val_2 for the first loop's setpoint and ramp rate respectively, Val_3 and Val_4 for the second loop etc.

Without the connection of a supervisory system, we immediately have a certain amount of functionality in this system. For instance, we could set up the four setpoints, the four ramp rates, the boolean and the process time manually. Then by setting the Load input on the RecipeMan function block to 20, and setting the Load input/output to load, the values set on the outputs will loaded into recipe number 20. If we then alter the values, we can reassert the previous values by setting RecipeNo to 20. If RecipeNo is already 20, the block will not be able to detect any change, and so it is necessary to set the Force Input/Output, which forces the recipe block to re-output the recipe.



Recipe Edit Screen

	Time	Setpoint
Loop1	10	65.3
Loop2	20	27.2
Loop3	30	30.5
Loop4	40	18.7

Figure 14-4 Recipe Edit Screen

Integrating ESP and PC3000 Recipe Systems

The PC3000 recipe system can be integrated with the Eurotherm Supervisory Package (ESP) recipe system, and ESP recipe edit, save and download can be used to transfer recipes to the PC3000.

The following example takes the previous example and shows how to add an ESP recipe interface to it.

Introduction

The interface to be outlined here consists of two main screens, a recipe edit screen and a recipe download screen. The recipe edit screen allows the user to alter recipe values off-line, and save them as ESP recipe files. The on-line screen shows an off-line recipe and a PC3000 recipe simultaneously, and allows download from ESP and on-line editing of the PC3000 recipe.

Gate Parameter Set-up

The gate parameters are set up with addresses as shown in Chapter 3 Appendix B. The gates which must be present are the recipe number for each block, the recipe values, and the recipe mask.

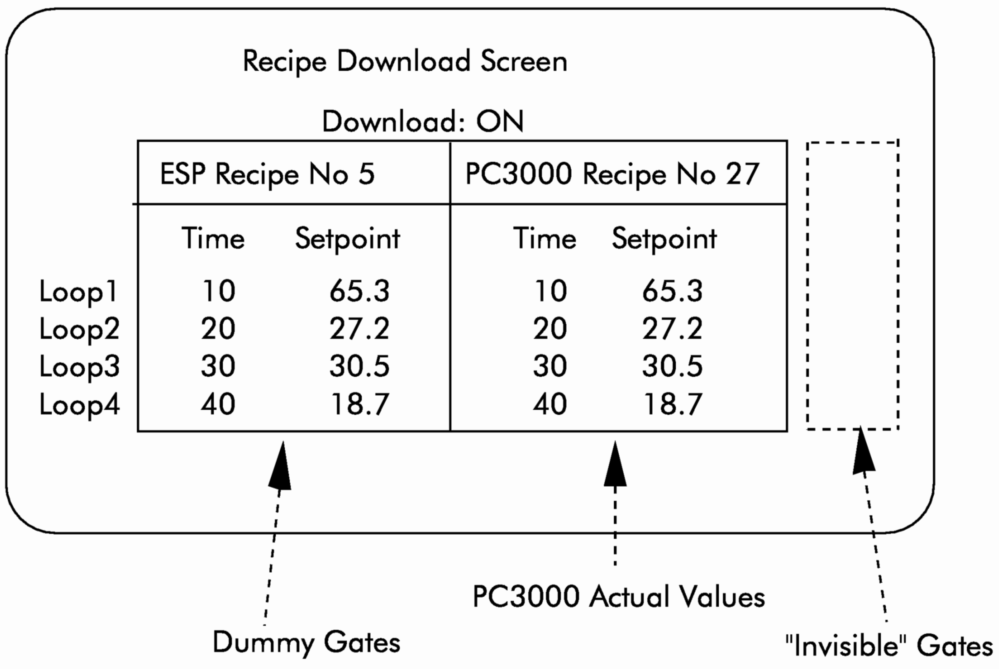
If we take an integer recipe block, with address 'EBA90' the gates

required will be as in the table in Chapter 3 Appendix B.

In addition to the recipe slave parameters, the function block parameters of the recipe manager will be needed. These can be generated by marking the gates in the PC3000 Programming Station (PS) and saving the gates.

Recipe Edit Screens

The edit screen (see fig edit) consists solely of dummy gates. We create a dummy gate to represent each of the recipe parameters. These can then be modified, and saved as an ESP recipe.



Recipe Download Screen

Download: ON

	ESP Recipe No 5		PC3000 Recipe No 27	
	Time	Setpoint	Time	Setpoint
Loop1	10	65.3	10	65.3
Loop2	20	27.2	20	27.2
Loop3	30	30.5	30	30.5
Loop4	40	18.7	40	18.7

Fig 14-5 Recipe Download Screen

Recipe Download Screens

The recipe download screen (see fig download) uses the same recipe files as the recipe edit screen.

It consists of three parts. There are the dummy gates which appear in the edit template - these are used to load the ESP recipe. This is done by detecting a change in the ESP recipe number gate, and loading the new recipe using Wizcon language - this allows a recipe file of a different name to be loaded. The PC3000 recipe gates give a view onto the recipe as stored in the PC3000.

There are a set of 'invisible' gates (foreground colour set to background colour). These are the destination pointer and recipe mask for the individual recipe slaves. The destination pointers are updated using Wizcon language to be the same as the visible recipe number gate. The recipe mask should be set to binary 1111111111111111 (i.e. always update all 16 values).

The recipe download mechanism is implemented in Wizcon language. It detects the ``Download'' gate changing, then copies the contents of the dummy gates to the PC3000 recipe gates.

Using EI Bisync to communicate to a Recipe Slave

This example shows communications to a recipe slave integer function block.

Assuming a Dint function block has the address 'EBA90'.

Assuming the EI function block has a   Gid of '0'.

The block is configured over the comms. either using a single composite parameter or using 19 basic parameters. These are :

Address	
00 A90	Composite parameter
00 A91	Destination
00 A92	Load From
00 A93	Mask
00 A94	Value 1
00 A95	Value 2
00 A96	Value 3
00 A97	Value 4
00 A98	Value 5
00 A99	Value 6
00 A9:	Value 7
00 A9;	Value 8
00 A9 <	Value 9
00 A9	Value 10
00 A9 >	Value 11
00 A9?	Value 12
00 A9@	Value 13
00 A9A	Value 14
00 A9B	Value 15
00 A9C	Value 16

Table 14-2 Basic Parameters

The fields within the composite parameter are in the same order as the individual parameters in the above table.

Execution, Timing and Synchronisation

Execution

When a command is issued to the recipe manager, a number of actions take place. Firstly, the command (e.g. a request to change to a new recipe) is broadcast to all of the manager's associated slaves. When each slave executes, it decides whether the command is valid (e.g. it will check to see if the new recipe is valid). On the manager's next execution, it checks the replies from all of its associated slaves. If any of the slaves can not perform the action, it will abort the attempt.

If all the slaves can perform the action, the manager broadcasts a command to perform the action to each of its associated slaves. Then as each slave next executes, it updates its outputs as necessary.

Timing

Latency from a change of recipe number to a change on the recipe slave outputs will be a maximum of three ticks. This worst case comes when the slave executes immediately before the manager, and the change to the recipe number comes immediately after the manager executes, as shown below.

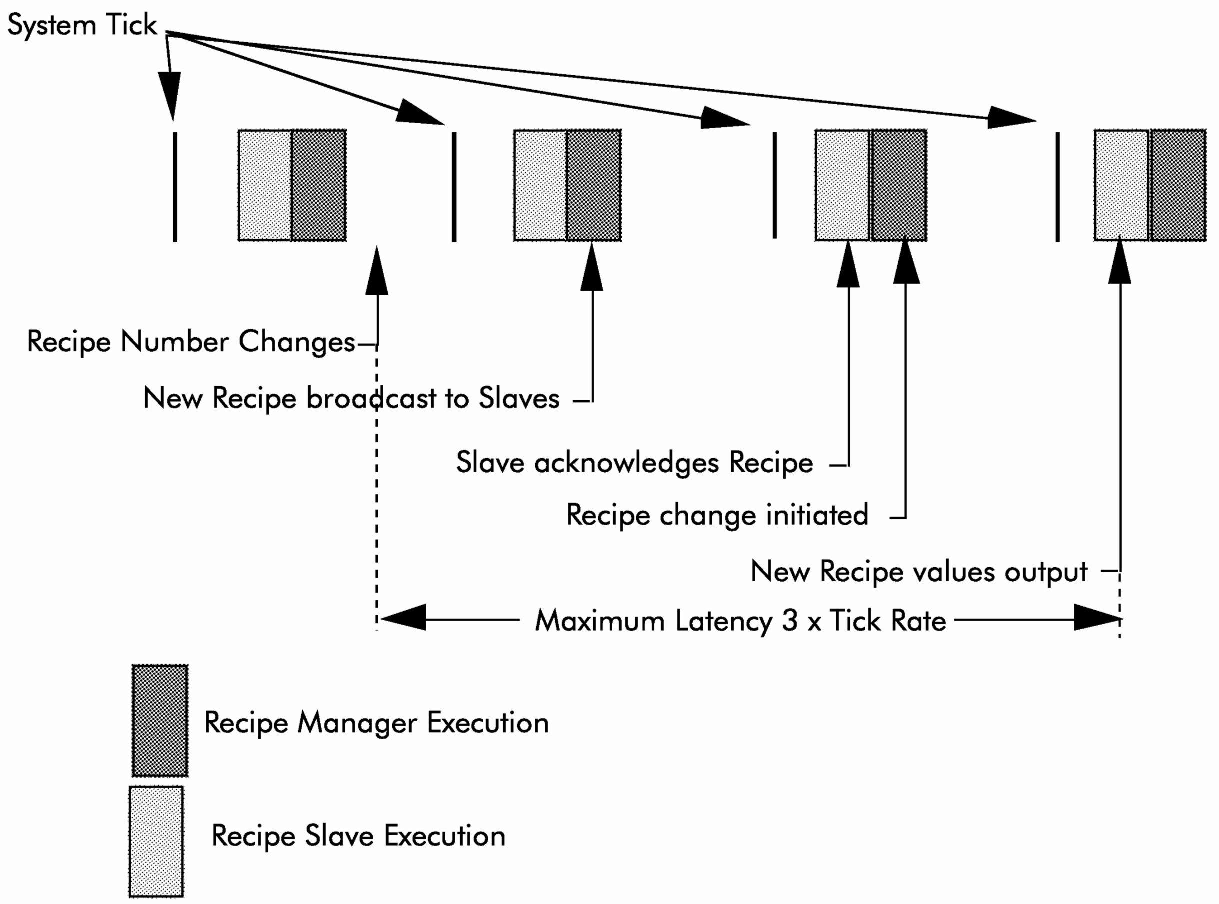


Fig 14-6 Recipe Manager and Slave Timing

Synchronisation

There is no guarantee of the execution order of function blocks executing on the same task - the programming station decides the order of execution dependent upon the wiring between the blocks. Because of this, the change of values in a recipe will no necessarily fall on the same tick. However, information is available in the form of the Changed flag on the recipe slaves. This is set by the block, and only reset by the userware.

Communications Timings

When the recipe salves are being accessed by EI Bisync as single element parameters, the multiple writes which occur to the same parameter can cause comms errors. This happens because the slave must process each transaction individually, and it is write protected until it has processed the last transaction.

There are two ways to overcome this - either move the recipe slaves to a faster task, or increase the amount of time between each transaction to a particular slave. This can be done by interleaving writes to a number of different recipe slaves.

RECIPEMAN FUNCTION BLOCK

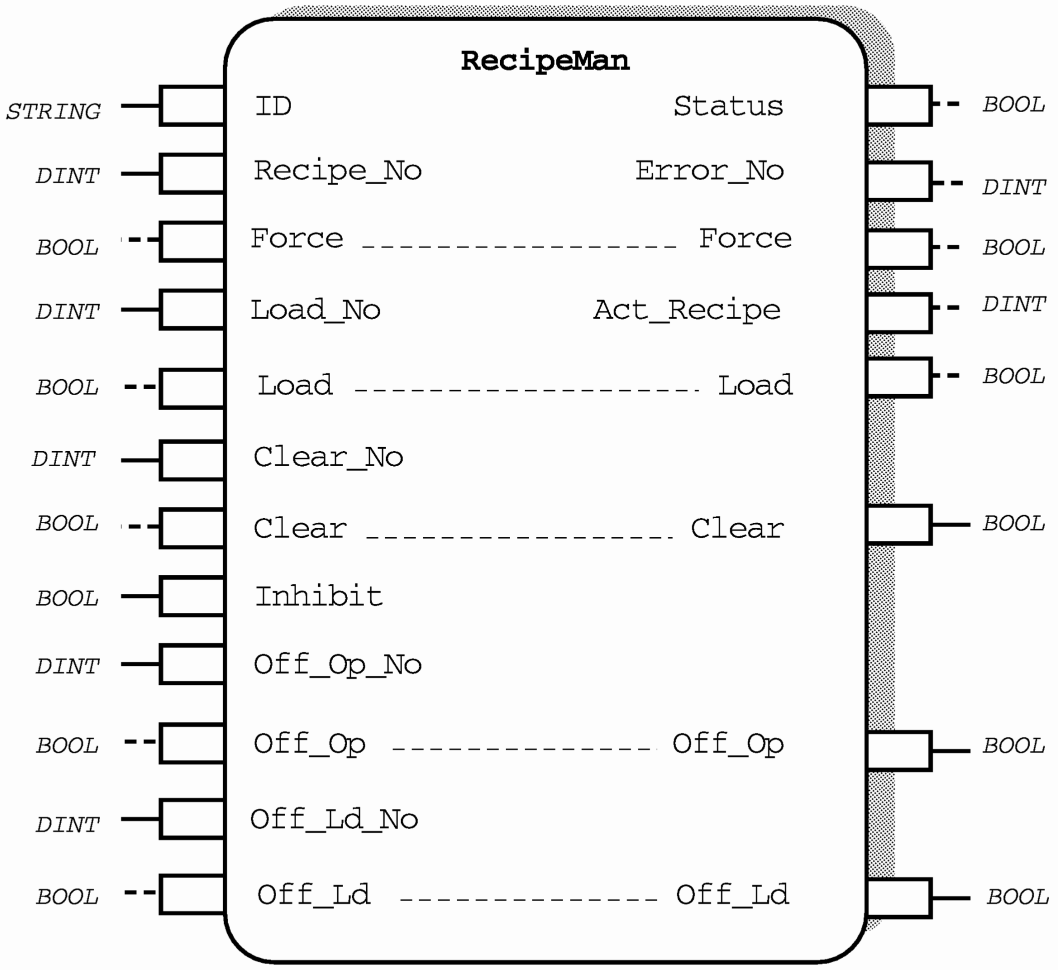


Figure 14-7 RecipeMan Function Block Diagram

Functional Description

The Recipe Manager function block is used to co-ordinate a number of Recipe Slave function blocks. Facilities are provided for selecting recipes, editing recipes, clearing recipes, etc.

Function Block Attributes

Type:	3E00
Class:	RECIPE
Default Task:	Task_2
Short List:	ID, Recipe No, Status
Memory Requirements:	66 Bytes

Parameter Descriptions

ID (ID)

Is a one character string which associates the Recipe Manager function block with a number of Recipe Slaves. All the Recipe Slave function blocks with the same ID will be controlled by this Recipe Manager.

Recipe_No(RN)

Used to set the required recipe number. A new recipe will be selected whenever Recipe_No changes. To re-select the same recipe no Force should be used.

Force (F)

Used to force the current recipe to be re-selected. It could be used following a recipe edit to force the latest values to be used.

Load_No (LN)

Defines the recipe slot that the current outputs will be loaded into when Load is set to Load.

Load (L)

When set to Load the current recipe outputs will be loaded into the recipe slot defined by Load_No.

Clear_No (CN)

Defines the recipe slot that will be cleared (deleted) when Clear is set to Clear.

Clear (C)

When set to Clear the contents of the recipe slot defined by Clear_No will be cleared (deleted).

Inhibit (I)

When set to Inhibit the recipe system will be write protected and no further data will be written to the recipe slots.

Off_Op_No (OON)

Defines the recipe slot whose contents will be transferred to the off-line editing pins when Off_Op is set to Output.

Off_Op (OO)

When set to Output the contents of the recipe slot defined by Off_Op_No will be transferred to the off-line editing pins.

Off_Ld_No (OLN)

Defines the recipe slot that the values of the off-line editing pins will be written to when Off_Ld is set to Load.

Off_Ld (OL)

When set to Load the values on the off-line editing pins will be written to the recipe slot defined by Off_Ld_No.

Status (S)

Set to Go if the recipe system is operating normally. If set to NOGO Error_No will indicate the cause of the error.

Error_No (EN)

Defines the error condition if Status is NOGO

100 = Undefined Recipe

101 = Load_No out of range

102 = Clear_No out of range

Act_Recipe (AR)

The recipe number which is currently being used. This may be different to Recipe_No if an invalid recipe number has been requested.

Parameter Attributes

Name	**Type**	**Cold Start**	**Read Access**	**Write Access**	**Type Specific Information**	
ID	**STRING**		Oper	Oper		
Recipe_No	**DINT**	1	Oper	Super	High Limit Low Limit	128 1
Force	**BOOL**	Idle (0)	Oper	Oper	Senses	Force (1) Idle (0)
Load_No	**DINT**	1	Oper	Oper	High Limit Low Limit	128 1
Load	**BOOL**	Idle (0)	Oper	Oper	Senses	Load (1) Idle (0)
Clear_No	**DINT**	1	Oper	Super	High Limit Low Limit	128 1
Clear	**BOOL**	Idle (0)	Oper	Oper	Senses	Clear (1) Idle (0)
Inhibit	**BOOL**	Enable (0)	Oper	Oper	Senses	Inhibit (1) Enable (0)
Off_Op_No	**DINT**	1	Oper	Super	High Limit Low Limit	128 1
Off_Op	**BOOL**	Idle (0)	Oper	Oper	Senses	Output 1) Idle (0)
Off_Ld_No	**DINT**	1	Oper	Super	High Limit Low Limit	128 1
Off_Ld	**BOOL**	Idle (0)	Oper	Oper	Senses	Load (1) Idle (0)
Status	**BOOL**	NOGO (0)	Oper	Block	Senses	NOGO (0) Go (1)
Error_No	**DINT**	0	Oper	Block	High Limit Low Limit	255 0
Act_Recipe	**DINT**	0	Oper	Block	High Limit Low Limit	128 0

Table 14-3 RecipeMan Parameter Attributes

STAGEMAN FUNCTION BLOCK

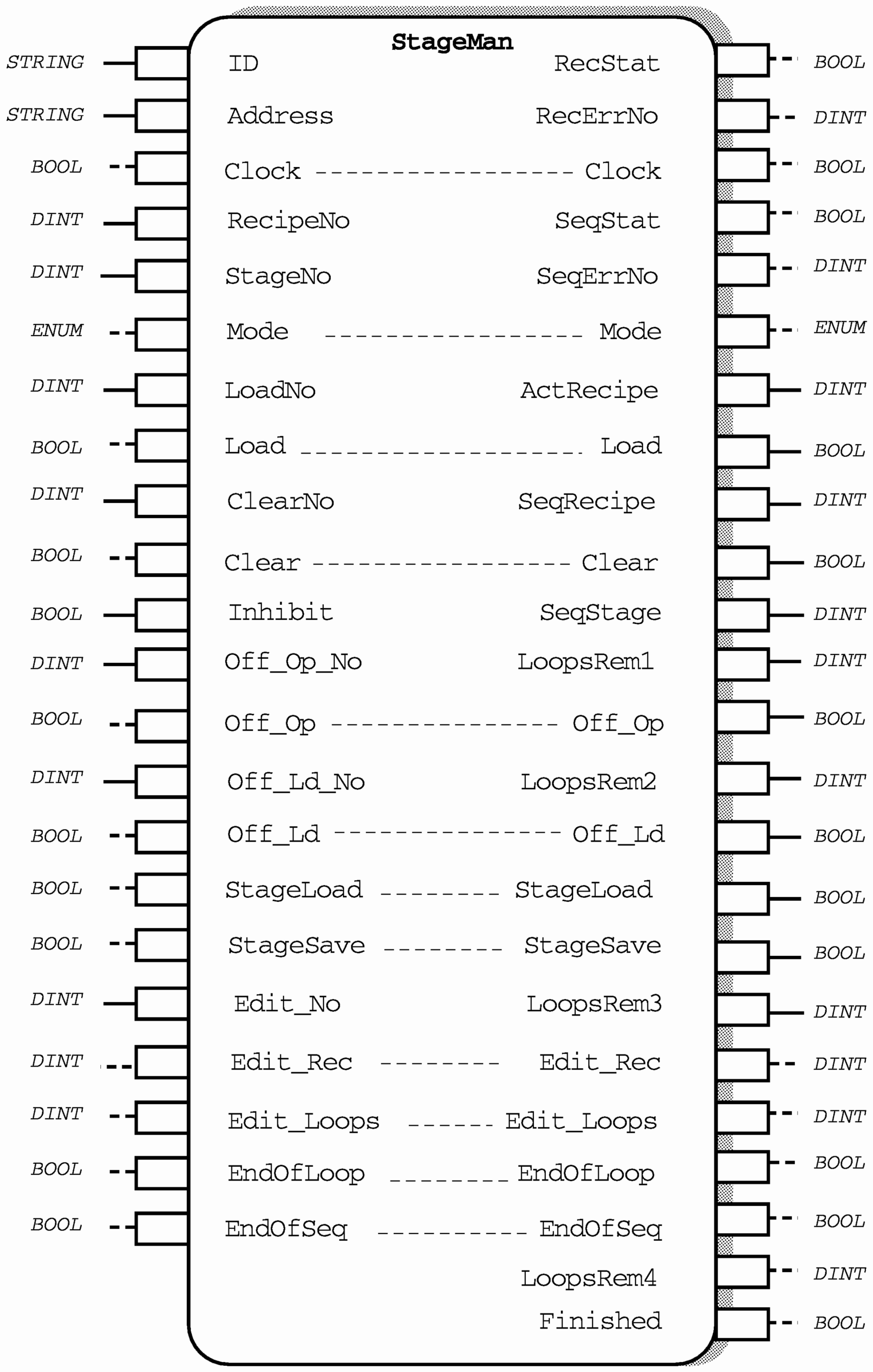


Figure 14-8 StageMan Function Block

Functional Description

The Stage Manager function block is an extension to the Recipe Manager function blocks which is designed to support the sequencing and looping of recipes.

The recipe sequence is pre-loaded into the Stage Manager function block either via the communications interface or by the user program.

Function Block Attributes

Type:	3E08
Class:..................................	RECIPE
Default Task:	Task_2
Short List:	ID, SeqStage, Act Recipe
Memory Requirements:	3408 Bytes

Parameter Descriptions

ID (ID)

Is a single character string used to associate recipe slave function blocks with the Stage Manager E.g. A Stage Manager with ID = 'A' will control all recipe slave function blocks which also have an ID = 'A'.

Address (A)

Address defines the communications address for accessing the internal data arrays for the storage of sequence information. E.g. If set to 'EBx00', the address of the multi-element parameter will be x00, the first individual parameter x01 etc.

Clock (C)

If Mode = Auto setting Clock to Tick will cause the next recipe in the sequence to be output. When the new recipe has been output, Clock will revert to Tock.

RecipeNo (RN

If Mode = Manual, this input specifies the recipe number which will be selected. A new recipe will be selected each time RecipeNo changes. The Stage Manager will indicate an error if an invalid recipe number is selected.

Stage No(SN)

When Mode is set to Jump, the Stage Manager will jump to the sequence stage defined by StageNo.

Mode (M)

The normal operating mode is Auto. In this case recipes will be controlled by the sequence and the Clock input.

If Mode is set to Manual the Stage Manager will operate a Recipe Manager. RecipeNo is used to select a specific recipe.

If Mode is set to Reset the sequence will be reset to the first stage of the sequence.

If Mode is set to Jump, the sequence will jump to the sequence stage specified by StageNo.

LoadNo (LN)

Defines the destination for loading the current recipe when Load is set to Load.

ClearNo (CN)

Defines the recipe slot which will be cleared (deleted) when Clear is set to Clear.

Clear(C)

When set to Clear the contents of the recipe slot defined by ClearNo will be cleared (deleted).

Inhibit (I)

When set to inhibit the recipe system will be write protected and no changes to the stored recipe data will be allowed.

Off_Op_No (OON)

Defines the recipe slot whose contents will be transferred to the off-line editing pins when Off_Op is set to Output.

Off_Op (OO)

When set to Output the contents of the recipe slot defined by Off_Op_No will be transferred to the off-line editing pins.

Off_Ld_No (OLN)

Defines the recipe slot that the values of the off-line editing pins will be written to when Off_Ld is set to Load.

Off_Ld (OL)

When set to Load the values on the off-line editing pins will be written to the recipe slot defined by Off_Ld_No.

RecStat (RS)

When set to Go the recipe system is operating normally. If set to NOGO an error occurred during the last recipe transaction. E.g. output, load clear, etc. RecErrNo will indicate the cause of the error.

RecErrNo (REN)

Defines the error condition if RecStat is NOGO

100 = Undefined Recipe

101 = Load_No out of range

102 = Clear_No out of range

SeqStat (SS)

When set to Go the last communications transaction with the Stage Manager was successful. If set to NOGO an error occurred during the last communications transaction. SeqErrNo will indicate the cause of the error.

SeqErrNo (SEN)

Defines the error condition if SeqStat is NOGO. Refer to the appropriate communications slave driver in chapter 3 for error number explanation.

ActRecipe (AR)

This is the recipe number which is currently being output by the recipe slave function block.

SeqRecipe (SR)

This is the recipe number that is being requested by the Stage Manager sequence. It may differ from ActRecipe if an invalid recipe selected.

SeqStage (SS)

This output indicates the current stage in the Stage Manager recipe sequence.

LoopsRem1 (LR)

This output indicates the number of loops remaining for loop 1.

LoopsRem2 (LR)

This output indicates the number of loops remaining for loop 2.

LoopsRem3 (LR)

This output indicates the number of loops remaining for loop 3.

LoopsRem4 (LR)

This output indicates the number of loops remaining for loop 4.

Finished (F)

Will be set to Done when the end of the recipe sequence has been reached.

StageLoad (SL)

When set to Load the contents of the sequence stage defined by Edit_No will be written to the editing pins Edit_Rec, Edit_Loops, EndOfLoop and EndOfSeq.

StageSave (SS)

When set to Save the values of the editing pins Edit_Rec, Edit_Loops, EndOfLoop and EndOfSeq will be written to the sequence stage defined by Edit_No.

Edit_No (EN)

Specifies the Stage Manager stage no that will be read from or written to when StageLoad and StageSave are set.

Edit_Rec (ER)

The recipe number which will be selected for this stage.

Edit_Loops (EL)

Defines the number of loops for this stage. The number of loops is defined for the first stage of a loop. Up to four nested loops are supported.

EndOfLoop (EOL)

Set to End if this stage is the last stage of a loop.

EndOfSeq (EOS)

Set to End if this stage is the last stage in the sequence.

Parameter Attributes

Name	**Type**	**Cold Start**	**Read Access**	**Write Access**	**Type Specific Information**	
ID	**STRING**	Null	Oper	Oper	Single character	
Address	**STRING**	Null	Oper	Oper		
Clock	**BOOL**	Tock (0)	Oper	Oper	Senses	Tick (1) Tock (0)
Recipe_No	**DINT**	1	Oper	Super	High Limit Low Limit	128 1
StageNo	**DINT**	1	Oper	Super	High Limit Low Limit	512 1
Mode	**ENUM**	Auto (0)	Oper	Oper	Senses	Auto (0) Manual (1) Reset (2) Jump (3)
Load_No	**DINT**	1	Oper	Super	High Limit Low Limit	128 1
Load	**BOOL**	Idle (0)	Oper	Oper	Senses	Load (1) Idle (0)
ClearNo	**DINT**	1	Oper	Super	High Limit Low Limit	128 1
Clear	**BOOL**	Idle (0)	Oper	Oper	Senses	Clear(1) Idle (0)
Inhibit	**BOOL**	Enable (0)	Oper	Oper	Senses	Inhibit (1) Enable (0)
Off_Op_No	**DINT**	0	Oper	Super	High Limit Low Limit	128 0
Off_Op	**BOOL**	Idle (0)	Oper	Oper	Senses	Output (1) Idle (0)
Off_Ld_No	**DINT**	1	Oper	Super	High Limit Low Limit	128 1
Off_Ld	**BOOL**	Idle (0)	Oper	Oper	Senses	Load (1) Idle (0)
StageLoad	**BOOL**	Idle (0)	Oper	Oper	Senses	Load (1) Idle (0)

Table 14-4 StageMan Parameter Attributes (Continued)

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
StgeSave	**BOOL**	Idle (0)	Oper	Oper	Senses	Save (1) Idle (0)
Edit_No	**DINT**	1	Oper	Super	High Limit Low Limit	512 1
Edit_Rec	**DINT**	1	Oper	Super	High Limit Low Limit	128 1
Edit_Loops	**DINT**	0	Oper	Super	High Limit Low Limit	65535 0
EndOfLoop	**BOOL**	Off (0)	Oper	Oper	Senses	End (1) Off (0)
EndOfSeq	**BOOL**	Off (0)	Oper	Oper	Senses	End (1) Off (0)
RecStat	**BOOL**	NOGO (0)	Oper	Block	Senses	G0 (1) NOGO (0)
RecErrNo	**DINT**	0	Oper	Block	High Limit Low Limit	255 0
SeqStat	**BOOL**	NOGO (0)	Oper	Block	Senses	G0 (1) NOGO (0)
SeqErrNo	**DINT**	0	Oper	Block	High Limit Low Limit	255 0
ActRecipe	**DINT**	0	Oper	Block	High Limit Low Limit	128 0
SeqRecipe	**DINT**	0	Oper	Block	High Limit Low Limit	128 0
SeqStage	**DINT**	0	Oper	Block	High Limit Low Limit	512 0
LoopsRem1-4	**DINT**	0	Oper	Block	High Limit Low Limit	255 0
Finished	**BOOL**	Running (0)	Oper	Block	Senses	Done (1) Running (0)

Table 14-4 StageMan Parameter Attributes

BOOL_16X128FUNCTION BLOCK

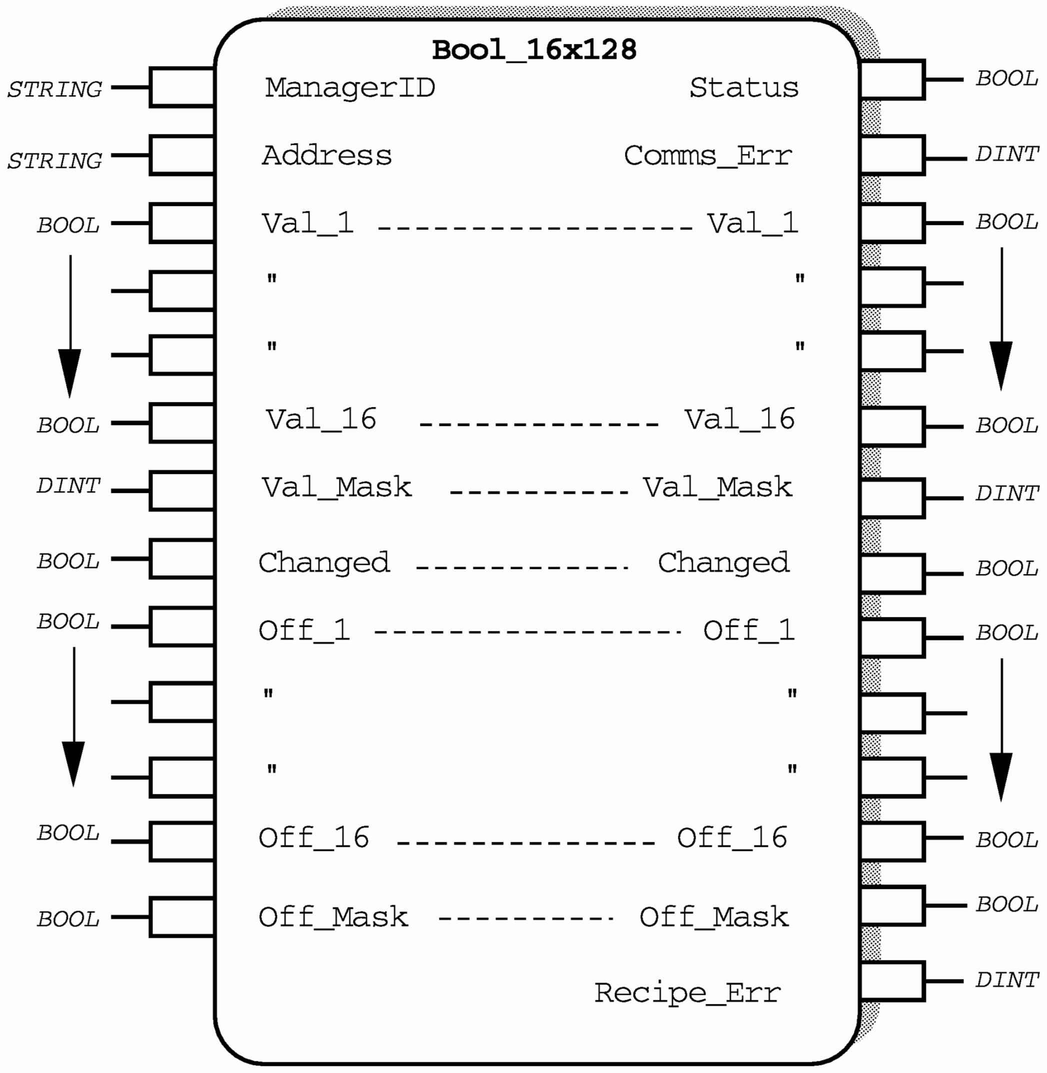


Figure 14-9 Bool_16x128 Function Block Diagram

Functional Description

The Bool _16x128 is used for boolean recipe data. Each block can support up to 16 recipe variables and up to 128 recipes. A Recipe Manager or a Stage Manager function block must also be created to control the recipe system.

Function Block Attributes

Type:	3E90
Class:	RECIPE
Default Task:	Task_2
Short List:	Manager ID, Status
Memory Requirements:	3088 Bytes

Parameter Descriptions

ManagerID (ID)

ID is a single character which is used to associate the Recipe Slave function block with a Recipe Manager or Stage Manager. E.g. A Recipe Slave with an ManagerID = 'A' will be associated with the Recipe or Stage Manager which also has an ID = 'A'

Address (A)

Address defines the communications address for accessing the internal data arrays for direct read and write of recipe data.

Status (S)

Set to Go if the recipe system is operating normally. If an error has occurred either due to a recipe transaction or a communications transaction Status will be set to NOGO and the cause of the error will be indicated by Comms_Err or Recipe_Err.

Comms_Err (CE)

Gives the error number for a communications error when Status is NOGO. Refer to the appropriate communications slave driver in chapter 3 for error number explanation.

Recipe_Err (RE)

Gives the error number for a recipe error when Status is NOGO.

1 = Recipe out of range

2 = Recipe undefined

3 = Default recipe undefined

4 = Load recipe out of range

5 = Clear recipe out of range

Val_1(V1) to Val_16(V16)

This is the actual recipe value for parameters 1 to 16. It can be written to change the value of the output, but this change will be permanent unless the recipe is reloaded using the Load pin on the Recipe or Stage manager.

Val_Mask (VM)

The mask is used to specify which values to change when the recipe is output. Each parameter is represented by a bit in the mask. E.g. a recipe slot with mask 15 (binary 0000000000001111) will only affect values 1 to 4.

Changed (C)

Set to Changed when a new recipe has been output. This parameter must be reset by the program.

Off_1(O1) to Off_16(O16)

The off-line editing pins are used for editing a recipe. Writing and reading parameters to and from the off-line pins is controlled by the Off-Op and Off_Ld pins on the Recipe or Stage Manager.

OffMask(OM)

This is the mask value for the off-line recipe.

Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
ManagerID	**STRING**	Null	Oper	Oper		
Address	**STRING**	Null	Super	Super		
Val_1 to Val_16	**BOOL**	Off (0)	Oper	Oper	Senses	On (1) Off (0)
Val_Mask	**DINT**	0	Oper	Super	High Limit Low Limit	65536 0
Changed	**BOOL**	Off (0)	Oper	Oper	Senses	Changed (1) Off (0)
Off_1 to Off_16	**BOOL**	Off (0)	Oper	Oper	Senses	On (1) Off (0)
OffMask	**DINT**	0	Oper	Super	High Limit Low Limit	65536 0
Status	**BOOL**	NOGO (0)	Oper	Block	Senses	NOGO (1) Go (0)
Comms_Err	**DINT**	0	Oper	Block	High Limit Low Limit	255 0
Recipe_Err	**DINT**	0	Oper	Block	High Limit Low Limit	255 0

Table 14-5 Bool_16x128 Parameter Attributes

REAL_16X128 FUNCTION BLOCK

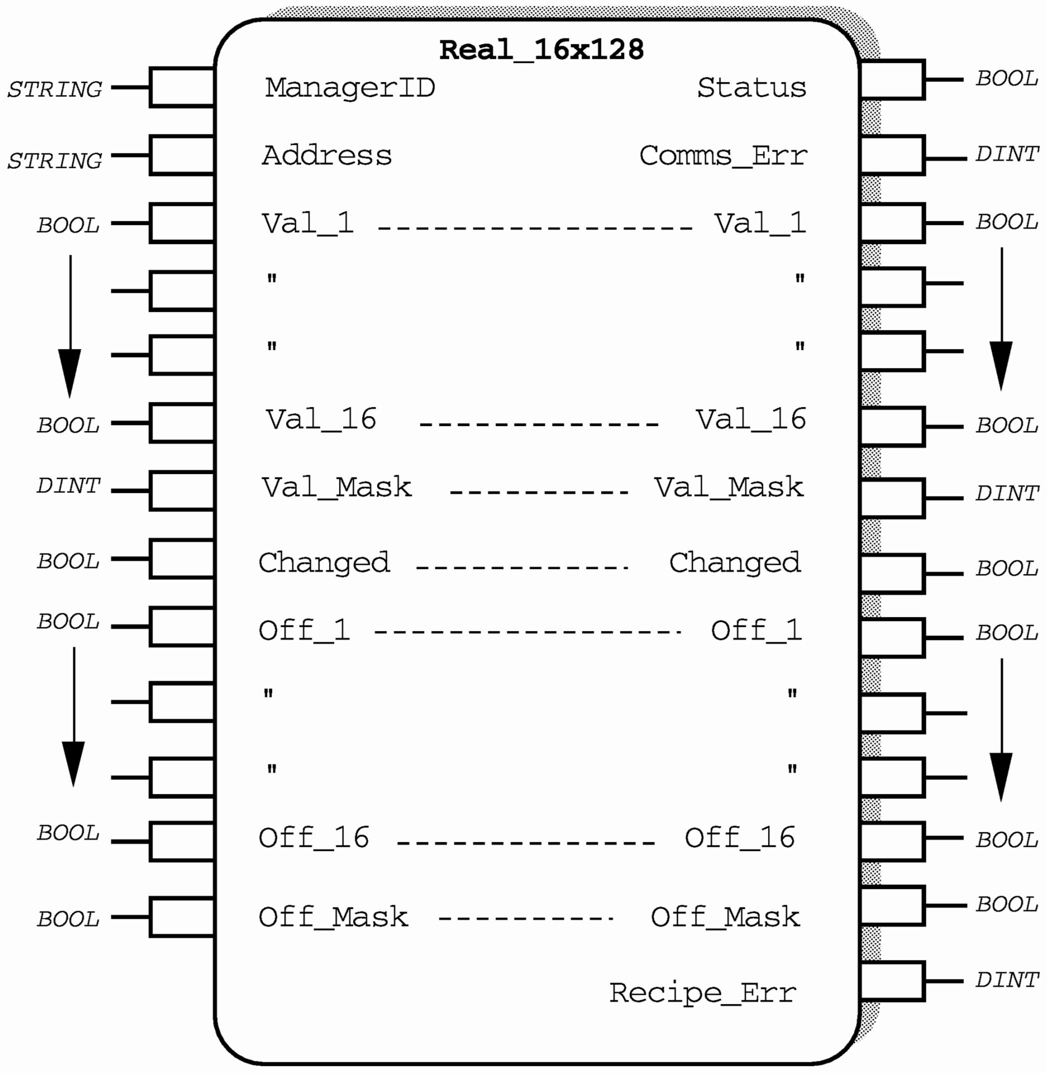


Figure 14-10 Real_16x128 Function Block Diagram

Functional Description

The Real_16x128 is used for real recipe data. Each block can support up to 16 recipe variables and up to 128 recipes. A Recipe Manager or a Stage Manager function block must also be created to control the recipe system.

Function Block Attributes

Type:.................................. 3EA0

Class: ....RECIPE

Default Task: ....Task_2

Short List: ....Manager ID, Status

Memory Requirements: ......9376 Bytes

Parameter Descriptions

See Bool_16x128 parameter descriptions

Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
ManagerID	**STRING**	Null	Oper	Oper		
Address	**STRING**	Null	Super	Config		
Val_1 to Val_16	**REAL**		Oper	Oper	High Limit Low Limit	1000000 -1000000
Val_Mask	**DINT**	0	Oper	Super	High Limit Low Limit	65536 0
Changed	**BOOL**	Off (0)	Oper	Oper	Senses	Changed (1) Off (0)
Off_1 to Off_16	**REAL**		Oper	Oper	High Limit Low Limit	1000000 -1000000
OffMask	**DINT**	0	Oper	Super	High Limit Low Limit	65536 0
Status	**BOOL**	NOGO (0)	Oper	Block	Senses	NOGO (0) Go (1)
Comms_Err	**DINT**	0	Oper	Block	High Limit Low Limit	255 0
Recipe_Err	**DINT**	0	Oper	Block	High Limit Low Limit	255 0

Table 14-6 Real_16x128 Parameter Attributes

DINT_16X128 FUNCTION BLOCK

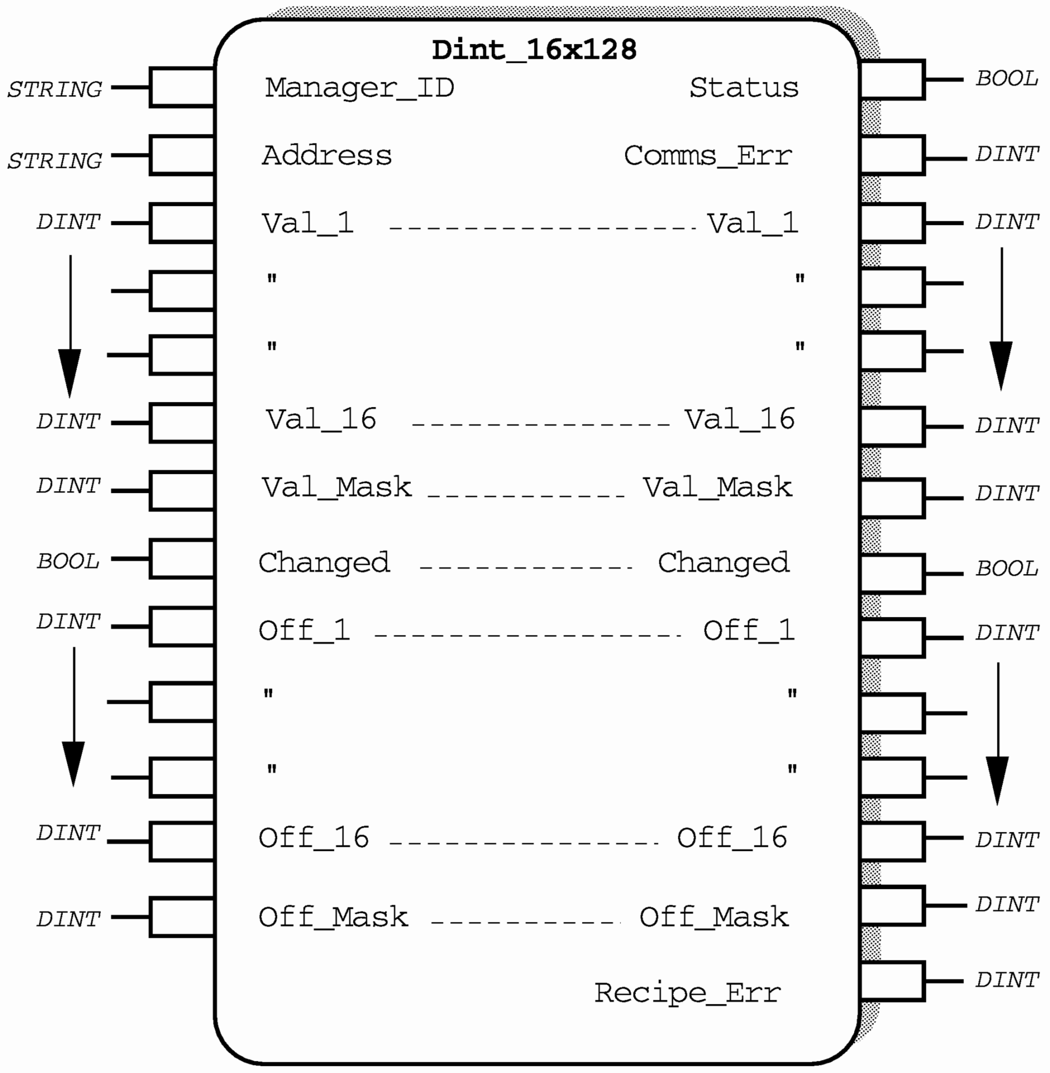


Figure 14-11 Dint _16x128 Function Block Diagram

Functional Description

The Dint_16x128 is used for integer recipe data. Each block can support up to 16 recipe variables and up to 128 recipes. A Recipe Manager or a Stage Manager function block must also be created to control the recipe system.

Function Block Attributes

Type:	3EB0
Class:	RECIPE
Default Task:	Task_2
Short List:	Manager ID, Status
Memory Requirements:	9424 Bytes

Parameter Descriptions

See Bool_16x128 parameter descriptions

Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
ManagerID	**STRING**	Null	Oper	Oper		
Address	**STRING**	Null	Super	Config		
Val_1 to Val_16	**DINT**	0	Oper	Oper	High Limit Low Limit	1000000 -1000000
Val_Mask	**DINT**	0	Oper	Super	High Limit Low Limit	65536 0
Changed	**BOOL**	Off (0)	Oper	Oper	Senses	Changed (1) Off (0)
Off_1 to Off_16	**DINT**	0	Oper	Oper	High Limit Low Limit	1000000 -1000000
OffMask	**DINT**	0	Oper	Super	High Limit Low Limit	65536 0
Status	**BOOL**	NOGO (0)	Oper	Block	Senses	NOGO (0) Go (1)
Comms_Err	**DINT**	0	Oper	Block	High Limit Low Limit	255 0
Recipe_Err	**DINT**	0	Oper	Block	High Limit Low Limit	255 0

Table 14-7 Dint _16x128 Parameter Attributes

STR_1X128 FUNCTION BLOCK

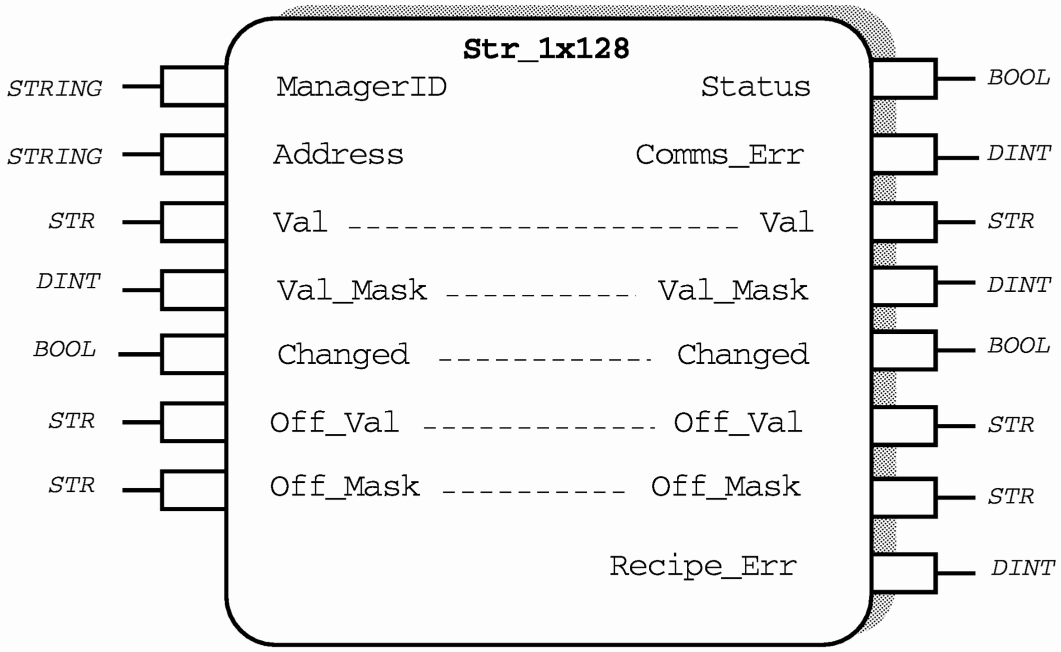


Figure 14-12 Str_1x128 Function Block Diagram

Functional Description

The Str_1x128 is used for String recipe data. Each block supports one recipe variable and up to 128 recipes. A Recipe Manager or a Stage Manager function block must also be created to control the recipe system.

Function Block Attributes

Type:	3EC0
Class:	RECIPE
Default Task:	Task_2
Short List:	Manager ID, Status, Value
Memory Requirements:	4704 Bytes

Parameter Descriptions

ManagerID (ID)

ID is a single character which is used to associate the Recipe Slave function block with a Recipe Manager or Stage Manager. E.g. a Recipe Slave with an ManagerID

= ’A’ will be associated with the Recipe or Stage Manager which also has an ID = ’A’.

Address (A)

Address defines the communications address for accessing the internal data arrays for direct read and write of recipe data.

Status (S)

Set to Go if the recipe system is operating normally. If an error has occurred either due to a recipe transaction or a communications transaction Status will be set to NOGO and the cause of the error will be indicated by Comms_Err or Recipe_Err.

Comms_Err (CE)

Gives the error number for a communications error when Status is NOGO. Refer to the appropriate communications slave driver in chapter 3 for error number explanation.

Recipe_Err (RE)

Gives the error number for a recipe error when Status is NOGO.

1 = Recipe out of range

2 = Recipe undefined

3 = Default recipe undefined

4 = Load recipe out of range

5 = Clear recipe out of range

Val(V)

This is the actual recipe value. It can be written to change the value of the output, but this change will be permanent unless the recipe is reloaded using the Load pin on the Recipe or Stage manager.

ValueMask (VM)

The mask is used to specify whether the output should change when the recipe is output.

Changed (C)

Set to Changed when a new recipe has been output. This parameter must be reset by the program.

Off_Val (OV)

The off-line editing pin is used for editing a recipe. Writing and reading parameters to and from the off-line pin is controlled by the Off-Op and Off_Ld pins on the Recipe or Stage Manager.

Off_Mask(OM)

This is the mask value for the off-line recipe.

Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
ManagerID	**STRING**	Null	Oper	Oper		
Address	**STRING**	Null	Super	Oper		
Val	**STRING**	Null	Oper	Oper		
Mask	**BOOL**	Off (0)	Oper	Oper	Senses	On (1) Off (0)
Changed	**BOOL**	Off (0)	Oper	Oper	Senses	Changed (1) Off (0)
Off_Val	**STRING**	Null	Oper	Oper		
Off_Mask	**BOOL**	Off (0)	Oper	Oper	Senses	On (1) Off (0)
Status	**BOOL**	NOGO (0)	Oper	Block	Senses	NOGO (0) Go (1)
Comms_Err	**DINT**	0	Oper	Block	High Limit Low Limit	255 0
Recipe_Err	**DINT**	0	Oper	Block	High Limit Low Limit	255 0

Table 14-8 Str_1x128 Parameter Attributes