DCS Controller Loading Reduction for Sequence Logic - A Boiler Control Project Case Study

A technique was developed for programming DeltaV systems for boiler control that replaces SFCs with function blocks and may result in lower controller loading.

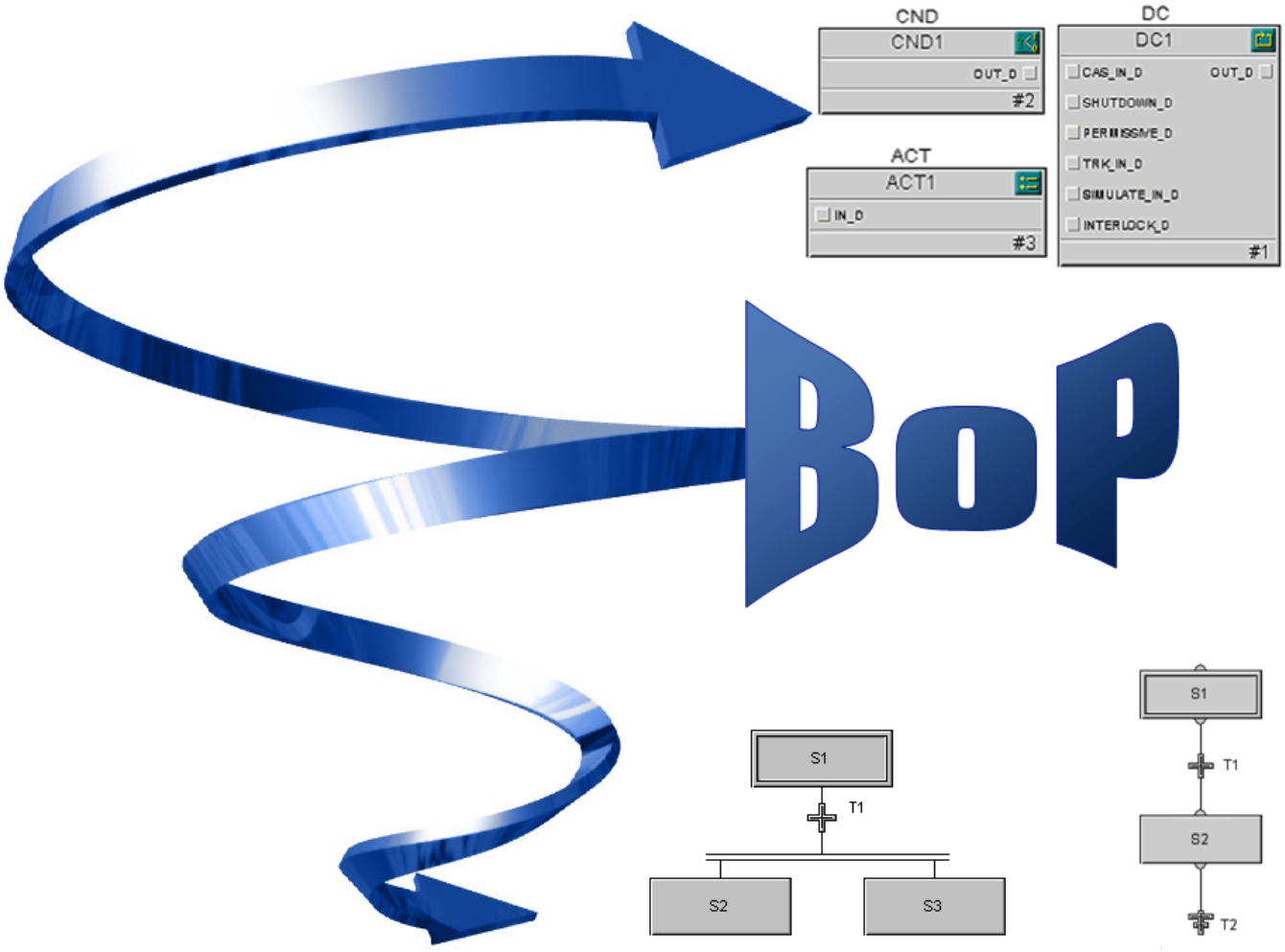

Table of Contents

Introduction

For many years the sequential control tasks in DCS and PLC control system programming have been handled using Sequential Function Charts (SFCs, defined in IEC 848). In DeltaV, for example, phase modules are generally set up using SFCs. These, which contain built in features derived from ISA S88, are often used in equipment modules. Phase classes have state transition diagram logic built in.

Yet while they make for convenient and easy-to-understand programming, often SFC implementations have a serious drawback: low free processor time in the controller(s) that contain the running SFC logic. However, if a higher scan time is used to use less free memory, the control logic may not execute fast enough to meet process requirements. This gives engineers unpleasant choices: put up with the low free time, create alternative control implementations (e.g. increase scan times or change the location of multiple modules), or purchase more computing power. Of course there are a few tricks available — in one case the fix was to use a CALC block — but that is abstract and makes troubleshooting more difficult for the customer.

A technique was developed for programming DeltaV systems for boiler control that replaces SFCs with function blocks and may result in lower overall processor consumption. This white paper will explain how this is done, using as an example a wood waste boiler project that used standard DeltaV function block tools in Control Studio to satisfy the sequence requirements outside the boiler combustion controls and flame safety system (hereafter referred to as the Balance of Plant or BoP module). This made the BoP module fit well with the common practice of using only function blocks in boiler controls.

Sequential Function Charts

A sequential function chart consists of a series of steps linked by transitions. A step is essentially a state of the system, and can be active or inactive (idle); the initial step is always active at start-up. Some steps have actions associated with them (open a valve, close a gate, etc.) that will be performed only if the step is active. Fig. 1 shows a simple linear SFC in generalized form, while Fig. 2 shows a specific application (note that in some SFC representations a transition is shown as a single heavy line segment, rather than the way it is shown here).

A transition is enabled when the step preceding it is active and the logical condition(s) for the transition are satisfied. When a transition is enabled the step preceding it becomes idle and the one following it becomes active.

It is possible for a single step to branch to two transitions (Fig. 3) by using a *sequence select divergence*, or for two transitions to lead to one step (Fig. 4) using a *sequence select convergence*. A single transition can lead to two simultaneous, or parallel, sequences by using a *parallel divergence*, as shown in Fig. 5.

In DeltaV programming, SFCs are deployed in phase modules, which contain built in features derived from ISA S88, and are often used in equipment modules.

Advantages of SFCs

Sequential function charts have a number of advantages. First of all, they are easy to understand, because they clearly and simply represent the steps of the process. They work well for high-level programming of control sequences. They are familiar to many people, because SFC is one of the languages specified in IEC 61131-3[1], and they are frequently used in programming PLCs. And SFCs are self-documenting and easy to troubleshoot and debug.

Disadvantages of SFCs

While SFCs are very useful in PLC programming, when used in a DCS they have several drawbacks. For one thing, in DeltaV applications an SFC can cause low free processor time in the controllers that contain the running SFC logic. Also, the effects of running multiple SFCs at a time need to be taken into account, especially with cases of using one controller for independent sequences or if recipe(s) may be executed.

Despite its seeming simplicity, an SFC requires a program structure that is not suitable for every application. Also, the effects of running multiple SFCs at a time need to be taken into account, especially in cases of using one controller for independent sequences or if recipe(s) may be executed.

Finally, it can take considerable preparation and planning before program writing can begin, and an SFC cannot be converted automatically to other languages, although a method[2] has been developed for converting SFCs to ladder diagrams (suitable for PLCs but not really applicable to DCSs).

Where SFCs are appropriate

SFCs are most useful in cases where equipment operation follows a definite sequence, where equipment operation is most easily understood as a series of states of operation, and where it's important for other people to be able to understand the programming quickly and easily.

Function Blocks

Function blocks, which are defined by IEC 61499, cover a broad range of applications. They are traditionally used for continuous control, but work quite well for discrete control, which is how they are used in the example presented in this paper.

Advantages of Function Blocks

Function blocks are easy to use, help reduce training requirements, allow reuse of programming elements, and help to reduce downtime[3]. And, as pointed out in the rest of this paper, they can also reduce processor loading compared to SFCs.

Disadvantages of Function Blocks

The main disadvantage cited for function blocks is that they cannot capture all the different aspects of today's distributed control applications[4].

Types of function blocks

Each function block contains standard process control algorithms (which can range from simple input conversions to complex control strategies) and parameters that customize the algorithm. A function blocks uses parameter data supplied by the user, by the function block itself, or by other function blocks to perform its calculations and

logic functions and to supply an output value to other function blocks or to field devices. Some function blocks also detect alarm conditions. The function blocks used in DeltaV systems are grouped into six categories:

- **Input/Output (I/O) Blocks** — Scale, convert, and filter input and output signals for use in other function blocks or field devices
- **Math Blocks** — Perform mathematical functions for conversion, integration, and totalizing
- **Timer/Counter Blocks** — Perform timing and counting functions for control and sequencing
- **Logical Blocks** — Perform logic functions for sequencing, scheduling, and interlocking
- **Analog Control Blocks** — Perform simple and complex algorithms for comprehensive analog control
- **Energy Metering Blocks** — Perform mathematical flow calculations for natural gases, steam, and other fluids
- **Advanced Control Blocks** — Perform complex algorithms for advanced process control

In this white paper we will limit the discussion of function blocks to the main ones deployed in the BoP module in the boiler example, all of which are classified as Logical Blocks. These include the Condition (CND) Function Block, Device Control (DC) Function Block, and Action (ACT) Function Block. The Boolean Fan Input (BFI) and OR (OR) Function Blocks are also used.

Condition Function Block

The CND block (Fig. 6) is used to confirm a condition as a prerequisite for further control system actions. In the case of the BoP module, CND blocks will confirm that sequence actions have completed, like transition confirmations in SFCs. If the CND belongs to an inactive sequence, the DISABLE parameter is set to true. Also, the timer function can be used to delay the next action from executing.

OUT_D is the discrete output value and status.

Parameters of the CND function block are shown in Table 1.

Table 1: Parameters of the CND Function Block

Parameter	Units	Description
ALGO_OPTS	None	Algorithm options. When selected, the expression algorithm will abort after a read error. The Algorithm option is AbortOnReadErrors. When the Condition block expression aborts, the value and status of PRE_OUT_D and OUT_D remain unchanged.
BLOCK_ERR	None	The summary of active error conditions associated with the block. The block error for the Condition function block is: • Configuration Error - The expression is empty

Parameter	Units	Description
ERROR_OPT	None	Specifies how the block behaves when a read error occurs. The value of PRE_OUT_D will be False (default), True, or the last value prior to the read error as defined in ERROR_OPT (unless 'Abort on Read Errors' is set in ALGO_OPTS, in which case ERROR_OPT does not apply). The status of PRE_OUT_D is BadNoComm when a read error occurs.
DESC	None	User-specified description of the expression.
DISABLE	None	Enables/disables the block logic (True = disable, False = enable).
OUT_D	None	The discrete output value and status.
PRE_OUT_D	None	The internal value set when the expression evaluates to True for the period defined by TIME_DURATION. PRE_OUT_D ignores the DISABLE value.
TIME_DURATION	Seconds	Specifies the time the expression must be True to set OUT_D.
TIMER	Seconds	The time the expression is True.

Device Control Function Block

The DC block (Fig. 7) is traditionally used to interpret a device state via discrete input(s), and set discrete output(s) in accordance with a setpoint value. The block compares the requested state (setpoint) to the actual state reported from the device and, after allowing time for the device to change state, detects alarm limits on any error. The basic functionality is augmented by an assortment of interlocks and device control options to customize the block's operation for a particular application.

The DC block supports mode control, setpoint tracking, simulation, and alarm limit detection. Options are available to specify the control strategy used in the block.

The setpoint requests the device to go to one of two or three supported states: Passive, Active 1 and Active 2 (optional). The Passive state is the power failure (safe) state, such as OFF or CLOSED. An Active state usually requires energy (or allows energy to flow), such as OPEN, RUN, FORWARD, or REVERSE or OFF/LOW/HIGH.

The DC block uses as many as eight discrete I/O channels to command a device to the requested setpoint state and to read back its confirmation contacts. Discrete I/O is associated with the Passive and Active states by means of a mask for each state that allows each bit to be defined as True (1), False (0), or not used. The user can configure four bits as outputs to the device and four bits as the contacts that confirm the device state. The confirm contacts must be maintained because the block is designed to alarm on loss of confirmation.

The DC block supports two modes: Automatic (Auto) and Cascade (Cas). Note that the Cas mode is not to be confused with CAS_IN_D, which is the discrete value and status of a setpoint from another block used when the block is in Cascade mode.

In the BoP module, the DC block is used for interface, permissive, and interlock support with no physical discrete I/O. The DC block is used in combination with the standard DeltaV faceplate DL_fp to allow the operator to change the sequence setpoint value and get feedback confirmation of the sequence status. And because the DC block supports the Auto and Cas modes, and DL_fp enables and disables the setpoint push buttons in Auto and Cas respectively, the operators are guided to not inadvertently stop a sequence. Table 2 shows a matrix used to determine the sequence's state:

Table 2: BoP Assessment of Status (two-state DC block)

DC Block Mode	DC Block Feedback	BoP Sequence State
Cas	RUNNING	Currently in a startup sequence
Cas	STOPPED	Currently in a shutdown sequence
Auto	RUNNING	Finished the startup sequence, in normal operation
Auto	STOPPED	Finished the shutdown sequence, in idle state

The DC Block Feedback value above is calculated based on the input(s) that go to the DC block, then the DC block displays if it is in the Active or Passive state. In the BoP case, the control module logically creates the input signals based on the actions in the ENABLE_START and ENABLE_STOP Action blocks, which are triggered by the START_ENABLED and STOP_ENABLED CND blocks respectively. The blocks are shown in Figure 11 and the expressions are shown in Table 3.

Action Function Block

The ACT block (Fig. 8) is used to complete an action expression, which usually involves writing a value to a parameter. In the case of the BoP module, ACT blocks will contain sequence actions and can be analogous to step actions in SFCs.

IN_D is the discrete input value and status that initiates expression evaluation.

Boolean Fan Input Function Block

The BFI block (Fig. 9) generates a discrete output based on the weighted binary sum, binary coded decimal (BCD) representation, transition state, or logical OR of one to sixteen discrete inputs. The block supports signal status propagation. It includes no modes or alarm detection.

Inputs and outputs are as follows:

- RESET_IN, when True (1), clears FIRST_OUT.
- IN_D1 through IN_D*n* are the discrete input values and statuses (as many as 16 inputs).
- OUT_INT is the unsigned 32-bit binary weighted output value that represents the bit combination of the inputs (IN_D*n*).
- OUT_D is the output value that represents the logical OR of the inputs (IN_D*n*).

- FIRST_OUT is the binary weighted output that represents the bit combination of the discrete input value or values (IN_D*n*) that are set when OUT_INT transitions from zero to non-zero and while ARM_TRAP is non-zero.

OR Function Block

The OR function block (Fig. 10) generates a discrete output value based on the logical OR of two to sixteen discrete inputs. When one or more of the inputs is True (1), the output is set to True. The block supports signal status propagation. It includes no modes or alarm detection.

IN_D1 through IN_D[n] are the discrete input values and statuses (as many as 16 inputs).

OUT_D is the discrete output value and status.

Description of BoP Implementation

The BoP module is broken into two main sections: the Sequence Control Initiation area shown in Fig. 11 and the Control Sequence Operation and Shutdown area shown in Fig. 12. The sections intend to clearly delineate where the sequence setpoint is determined and where the sequence actions and confirmations are in order to provide for better troubleshooting. We will focus on the B1_LIMEINJ_CNTL module for the description of the specific implementation.

Table 3: Configuration of Select Parameters in B1_LIMEINJ_CNTL

Name	Configuration
B1_LIMEINJ_CNTL	Module Scan Rate: 1 sec Faceplate Display: DL_fp Detail Display: DL_dt8 Parameter Download Behavior: Use configured values
DC1	DEVICE_OPTS: Permissive, SP Track, Interlock checked DC_STATE Named Set: states_mtr2 FV_D and PV_D Named Set: mtr2-pv SP_D Named Set: mtr2-sp STATE_MASKS: Active 1, Input 1 and Active 1, Output 1 checked
START_ENABLED	('^/DC1/PERMISSIVE_D.CV' = 1) AND ('^/DC1/SP_D.CV' = 'mtr2-sp:START') AND ('^/START_FLAG.CV' = FALSE)

Name	Configuration
ENABLE_START	'^/START_FLAG.CV' := TRUE; '^/DC1/MODE.TARGET' := CAS; '^/CAS_IN_D.CV' := 1; '^/DC1/F_IN_D1.CV' := 1; '^/ESHTDWN_CONFIRM1/DISABLE.CV' := 1; '^/CSHTDWN_CONFIRM1/DISABLE.CV' := 1; '^/CSHTDWN_CONFIRM2/DISABLE.CV' := 1; '^/CSHTDWN_CONFIRM3/DISABLE.CV' := 1; '^/STOP_ENABLED/DISABLE.CV' := 0; '^/START_ENABLED/DISABLE.CV' := 1
STOP_ENABLED	(('^/DC1/INTERLOCK_D.CV' = 0) OR ('^/DC1/SP_D.CV' = 'mtr2-sp:STOP')) AND ('^/START_FLAG.CV' = TRUE)
ENABLE_STOP	'^/START_FLAG.CV' := FALSE; '^/DC1/MODE.TARGET' := CAS; '^/CAS_IN_D.CV' := 0; '^/DC1/F_IN_D1.CV' := 0; '^/START_CONFIRM1/DISABLE.CV' := 1; '^/START_CONFIRM2/DISABLE.CV' := 1; '^/START_CONFIRM3/DISABLE.CV' := 1; '^/START_ENABLED/DISABLE.CV' := 0; '^/STOP_ENABLED/DISABLE.CV' := 1
START_STOP	'^/DC1/DC_STATE.CV' = 'states_mtr2:Confirmed Running'
START_ACTION4	'^/DC1/MODE.TARGET' := AUTO; START_MESSAGE := LOGEVENT("Limestone Injection Startup Complete"); '^/START_CONFIRM1/DISABLE.CV' := 1; '^/START_CONFIRM2/DISABLE.CV' := 1; '^/START_CONFIRM3/DISABLE.CV' := 1

The Sequence Control Initiation area comprises a motor control template with the added function blocks START_ENABLED, ENABLE_START, STOP ENABLED, and ENABLE_STOP to handle DC1 initial mode changing to CAS; DC1 feedback status of RUNNING or STOPPED; and initialization of the sequence CND function blocks.

The Control Sequence Operation and Shutdown area comprises the START_STOP block linked to the DC block running status; it starts a startup or shutdown sequence based on a positive edge trigger output and negative edge trigger output respectively. Then a progression of ACT blocks and CND blocks implement the sequence actions and confirmations. Note that in the case of the BoP modules created in this project positive edge triggers were used between the CND and ACT function blocks. The last ACT block of the sequence would then set the DC1 block to AUTO mode and disable the CND blocks of the sequence. An implementation might instead use expressions in each ACT block to disable the previous CND block. As a result, the implementation would be more analogous to SFCs and positive edge triggers wouldn't be needed.

As a way to make normal sequence troubleshooting easier when using the BoP module, the BoP implementer may choose to create top level Input Parameters in the BoP module and connect them to the respective sequence CND blocks' DISABLE parameters via Internal References. As a result, a BoP sequence troubleshooter will be able to easily see the DISABLE value of the CND blocks at the top level of the BoP CM rather than having to click on each CND block individually.

What's believed to be unique is the combination of the two areas that can handle failure detection (through the use of interlock and permissive functions of the DC block), using a standard interface of DL_fp, the connection between the two areas, and the sequence logic contained in a top-down format that's easy to read in comparison to a single CALC block with the action and confirmation expressions.

BoP Performance Results

The BoP module was originally created to avoid SFCs, which may or may not have merit depending on the end user requirements. What may have more value are the possible benefits to processor free time and sequence execution performance.

The BoP module may take up less space in DeltaV Controllers compared to traditional SFC implementation (i.e. command driven EM, phase). This effect is perhaps due to the smaller amount of background logic associated with SFCs vs. CND/ACT/DC function blocks, all of which need to be loaded on the controller at the beginning of the sequence regardless. As a result, the smaller amount of loading allows the memory to be used for essential communication between the controller and other computers on the DeltaV network, and decreases the risk of getting into a situation of adding another controller and changing control logic configuration.

A direct comparison study of the BoP module and a corresponding SFC implementation has not been completed. Instead, performance data of a controller (B1_FH) with 22 conveyance motors, 129 control modules, and four BoP modules has been collected. The significance of this example is:

Conveyance applications are traditionally covered by PLCs.

The control logic and function block number of the BoP modules is significant and comparable to a medium sized process.

The testing was done with DeltaV version 9.3.1.5251.xr and a MDPlus controller with hardware revision 5.50. The results may vary if any sequence is running. Also, the CAT completed with the B1_FH control modules executing at one second scan rates.

Table 4: Free Time vs. Scan Rate for BoP Modules

Scan Rate (ms)	Free Time (%)
100	53
200	73
500	86
1000	89
2000	91

If the B1_FH BoP modules and the motor control modules were to execute at 200 ms, the processor free time would decrease to 21-23%, which is generally considered a borderline acceptable free time. The result shows that the performance approaches PLC scan rates with complex sequencing requirements.

SWOT Analysis toward using BoP Modules on Projects

To illustrate a qualitative evaluation of the potential end state of using BoP modules below is a Strengths, Weaknesses, Opportunities, Threats (SWOT) Analysis:

Strengths	Weaknesses
Easy implementation learning curve due to use of well known function blocks in DeltaV. May have a competitive edge against traditional PLC applications. May allow for standardization on engineering training and expertise to function blocks alone depending on industry.	Programming is more complex, potentially resulting in longer project timelines and cost. SX/MX controller may make solution less appealing or unneeded. Function block approach deviates from increasing usage of SFCs (e.g. SYNCADE with recipe authoring). Presentation not as straightforward as SFCs.
Opportunities	**Threats**
Allows for better control system integrity, especially for controllers with multiple, independent sequences. Graphical interface is identical to that of a motor, allowing for faster Operator training on interface. Potentially uses less controller memory compared to a SFC for multiple, overlapping sequences	Potentially uses more controller memory compared to a SFC if multiple sequences are always executed on a single controller with no overlap. Users may view implementation as foreign depending on background with DeltaV.

Summary

The application of using the standards developed for the BoP module to possibly increase controller free times and the rate of sequence execution is being proposed.

We believe the most favorable use of this type of module is with controllers that require fast scan times on multiple units/independent sequences running on a single controller. However, there are numerous implementation disadvantages to the widespread use of this type of module. Actions may be done within the DeltaV software to streamline programming and make the presentation easier for Emerson integrators to interpret and configure. Also, further research needs to be conducted to determine if the hardware benefits with using this approach are worthwhile. Nevertheless, the BoP module has proved to be a robust method for the sequencing requirements found in the wood waste boiler project (i.e. nine Functional Descriptions involving different UO), and the controllers used on the project have good free processor times. As a result, the BoP module should be successful for select applications/projects with sequencing requirements and should help to reach the end goal of getting projects completed on time; on budget; and within quality specifications.

For further reading

More information on DeltaV configuration and other topics can be found at http://www.easydeltav.com/BOL/MasterBOL.htm. A non-javascript version is at http://www.easydeltav.com/bol/whnjs.htm. Also see http://www.easydeltav.com/bol/MasterBOL.htm.

For more information on IEC 61499, see http://knol.google.com/k/iec-61499#.

Figures

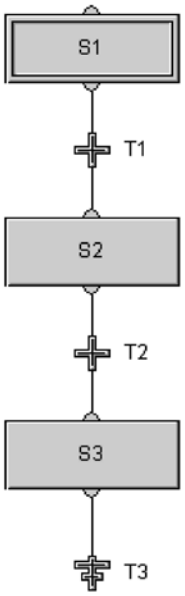

Figure 1: Linear SFC

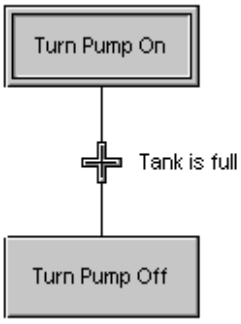

Figure 2: SFC Application

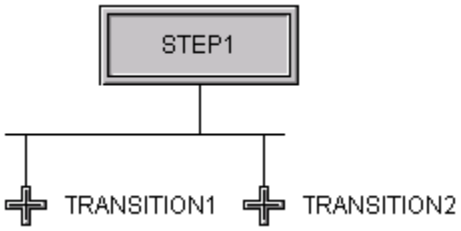

Figure 3: Sequence Select Divergence

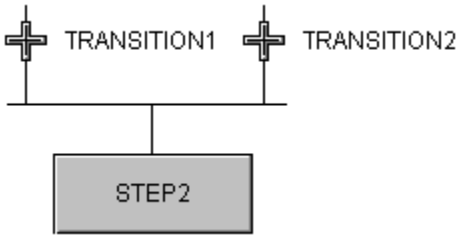

Figure 4: Sequence Select Convergence

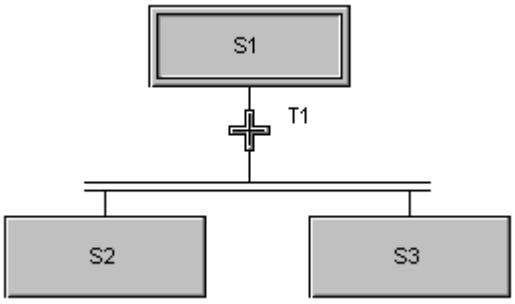

Figure 5: Parallel Divergence

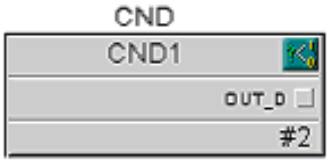

Figure 6: Condition Function Block

Figure 7: Device Control Function Block

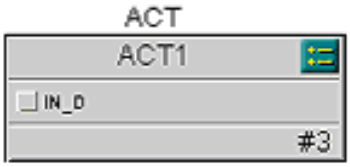

Figure 8: Action Function Block

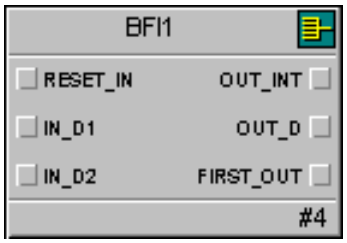

Figure 9: Boolean Fan Input Function Block

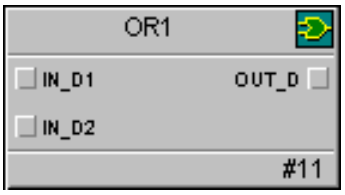

Figure 10: OR Function Block

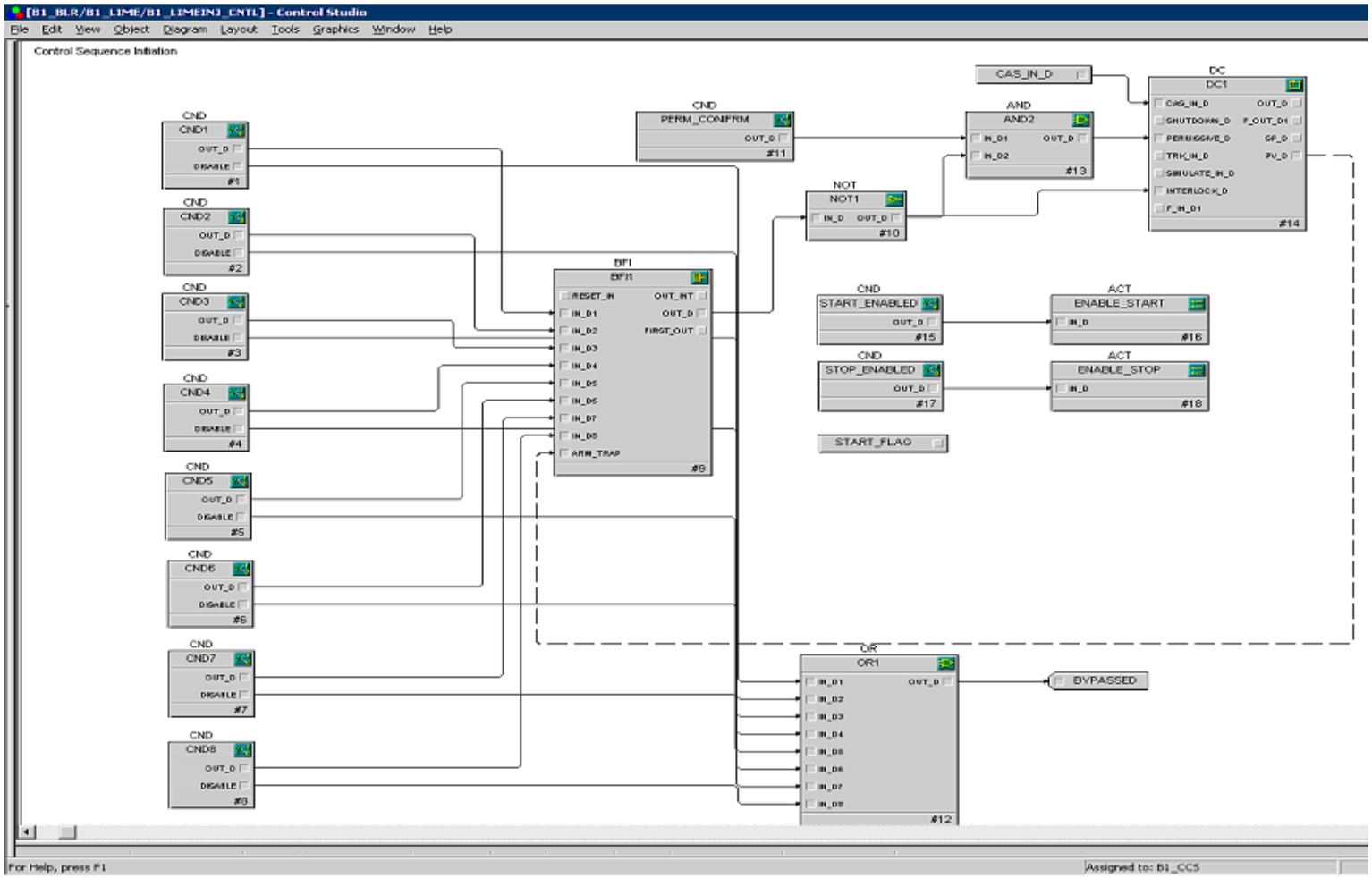

Figure 11: BoP Sequence Control Initiation Area

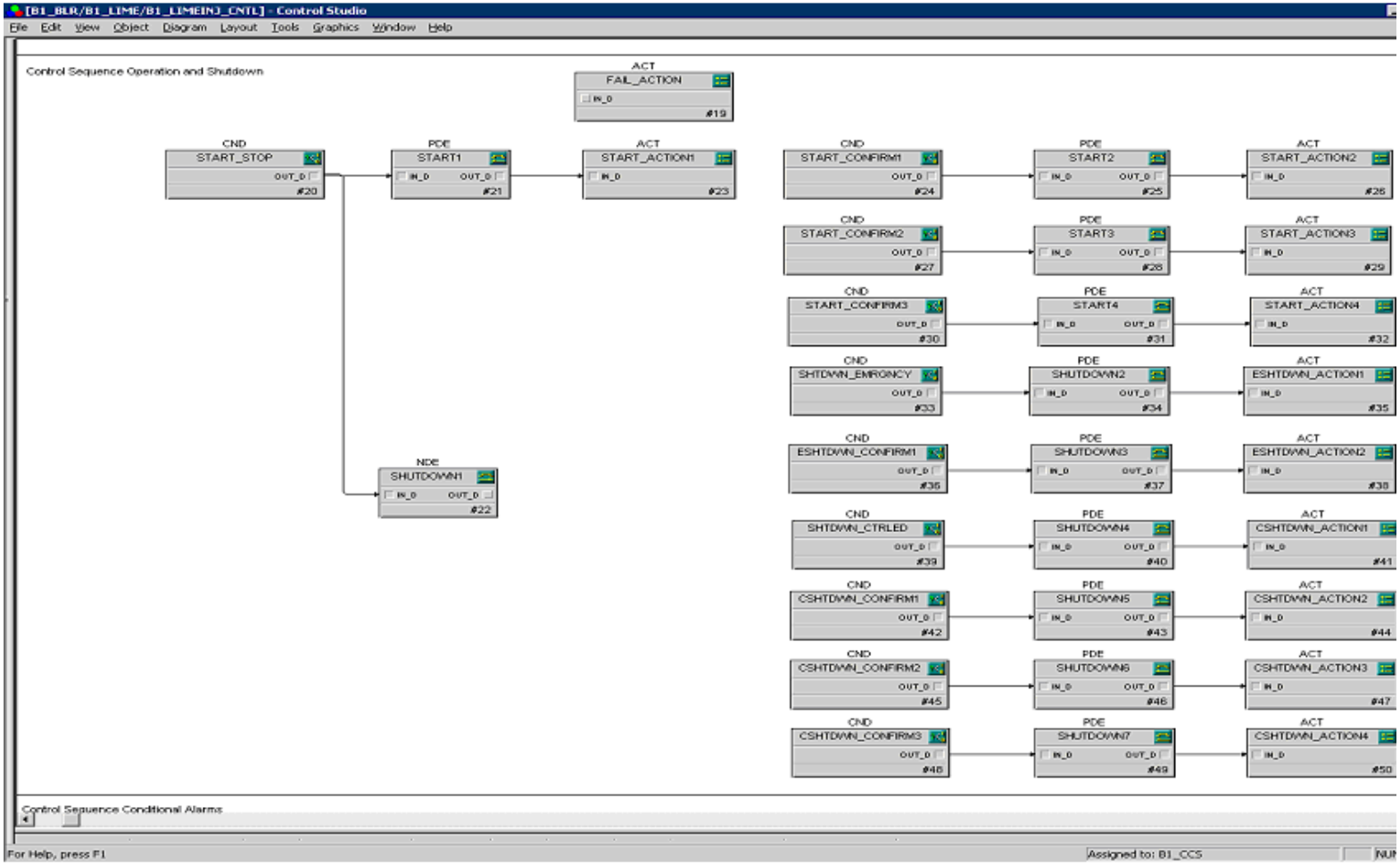

Figure 12: BoP Control Sequence Operation and Shutdown Area

Author, Contributors, and References

By Shawn Zadeh, Process Automation Engineer, Emerson Process Management

The following people contributed to development of the BoP Module:

Mark Durica, Sr. Principal Engineer, Emerson Process Management

Barbara Hamilton, Industrial Energy Consultant, Emerson Process Management

[1] Ted Thayer, "Speaking in Tongues: Understanding the IEC 61131-3 Programming Languages," Control Engineering, January, 2009

[2] Eric Anderson, white paper: "Sequential Function Chart to PLC Ladder Logic Translation", DMC Corp. (http://www.dmcinfo.com/Portals/0/State%20Transition%20Diagram%20to%20PLC%20Ladder%20Logic%20Translation%20Whitepaper.pdf), Sept 8, 2009.

[3] Rob Spiegel, "Quick & Simple PLC Programming", Automation World, April 2006 p.43

[4] Kleanthis Thramboulidis, "Design Alternatives in the IEC 61499 Function Block Model", IEEE Int. Conf. on Emerging Technologies and Factory Automation, (ETFA'06), Sept 2006.

To locate a sales office near you, visit our website at:
www.EmersonProcess.com/DeltaV
Or call us at:
Asia Pacific: 65.777.8211
Europe, Middle East: 41.41.768.6111
North America, Latin America: +1 800.833.8314 or +1 512.832.3774

For large power, water, and wastewater applications contact Power and Water Solutions at:
www.EmersonProcess-powerwater.com
Or call us at:
Asia Pacific: 65.777.8211
Europe, Middle East, Africa: 48.22.630.2443
North America, Latin America: +1 412.963.4000

© Emerson Process Management 2010. All rights reserved. For Emerson Process Management trademarks and service marks, go to: http://www.emersonprocess.com/home/news/resources/marks.pdf.

The contents of this publication are presented for informational purposes only, and while every effort has been made to ensure their accuracy, they are not to be construed as warrantees or guarantees, express or implied, regarding the products or services described herein or their use or applicability. All sales are governed by our terms and conditions, which are available on request. We reserve the right to modify or improve the design or specification of such products at any time without notice.