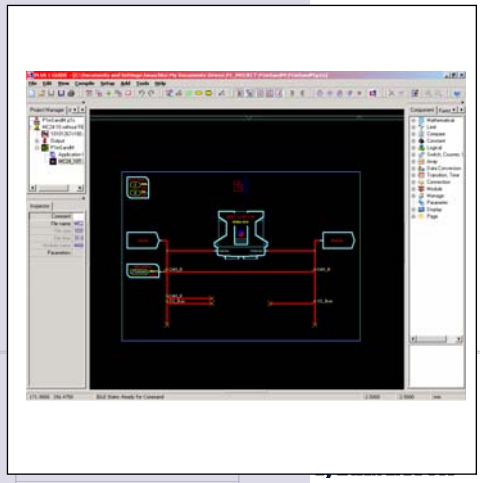
SAUER DANFOSS

PLUS+1™

Development Guidelines

Revision B



Changes

Revision History

Revision date	Page	Change	Author	Remarks
22.10.2006	All	Release	Guidelines Team: T. Waschkowski T. Juul K. Lorenscheit J. Wandersee P. Sundh S. Niska T. Braun	Revision A
18.01.2007	16	Do … on Buses reworked	T. Waschkowski	Rev A 0.1
22.01.2007	Many	Rewording	L. Culbert	Rev A 0.2
30.01.2007	All	Release	C. Trende	Rev. B

Preface

The Guidelines Team and many other people contributed to these Guidelines by providing material, reviewing documents, editing documents as well as joining many meetings and discussions. Thank you to everybody!

This document starts with its first edition. The project will continue and there will be future opportunities and extensions. Please let us know your opinion.

Sauer-Danfoss welcomes suggestions for improving our documentation. If you have suggestions for improving this document, please contact Sauer-Danfoss at doc.eh.de.nms@sauer-danfoss.com.

Contents

Contents

Prefix

The **PLUS+1 GUIDE Tool** is an easy to learn and easy to use development tool to develop software for machinery control.

The GUIDE graphical programming language is a powerful and flexible programming language designed for rapid programming as well as complex software development.
The simplicity and speed of developing a system with GUIDE can hide the reality that graphical programming with GUIDE can be complex in case of complex problems.

The language is, in fact, a complete programming language, suitable for handling the biggest and most complex applications that engineers can realize on machines.

In particular, programmers creating networking applications for embedded control systems cannot afford to introduce errors by misinterpretation of the specification.

A **structured development design is crucial** to ensure quality and reusability of the code developed!

Many engineers are facing the challenge of building more and more complex systems to keep up with the growing complexity of a changing specification. Small and lean implementations are growing in functionality and complexity.

It is easy for the customer to specify that he has “just the small change.” “It’s only software.”

A structured programming process is required to master the generic requirement for state of the art technology and quality expectations of a supplier.

In reality, a structured process for developing code – in any software programming language – is independent of the language or tool used.

This document outlines some common practices for following a structured programming process and shows how GUIDE should be used in these situations.

Overview

The purpose of this document is to promote consistency in PLUS+1 GUIDE™ applications and functions.

This document contains guidelines for:

1. General Software Design
2. General Graphical Coding
3. Coding Checklist
4. Page levels
5. Signal Flow
6. Descriptions and Comments
7. Application Structure
8. The naming of variables used in PLUS+1 GUIDE applications such as user-defined wires, checkpoints, and sub-buses.
9. The mapping of discrete and status signals to data types.
10. The formatting of the names of Sauer-Danfoss functions that appear in the Function tab.
11. The appearance of Sauer-Danfoss functions in the Function tab's Preview pane.

General

Software Design explains the software solution also to non-programmers.
A software design can be understood as a block diagram representation of a high abstraction level.

Software Design is not only direct programming with GUIDE.
Software Design means to create a solution plan which achieves the best compromise in regards to the total cost of ownership during the product life cycle.

It is crucial for safety, quality and success to develop software on which is easy to do maintenance.

Mobile machinery exists for many years and so does the software for embedded systems in mobile machinery.

This truly points out the large benefit of the effort spent in a well defined specification. The specification should also be under version control, which can be found inside the software design.

If there are still some doubts as to what a software design means, try to paint the complete system with software on a piece of paper and consider how add-ons can be planned as well. When the simplification of the system is also considered, the design is on a good path. After that, make it nice looking with the help of a drawing tool (E.g. Visio, GUIDE).

Note: "Think about what you want to do before you do it."

Example of a System Block Diagram

This picture shows an example block diagram of a system.

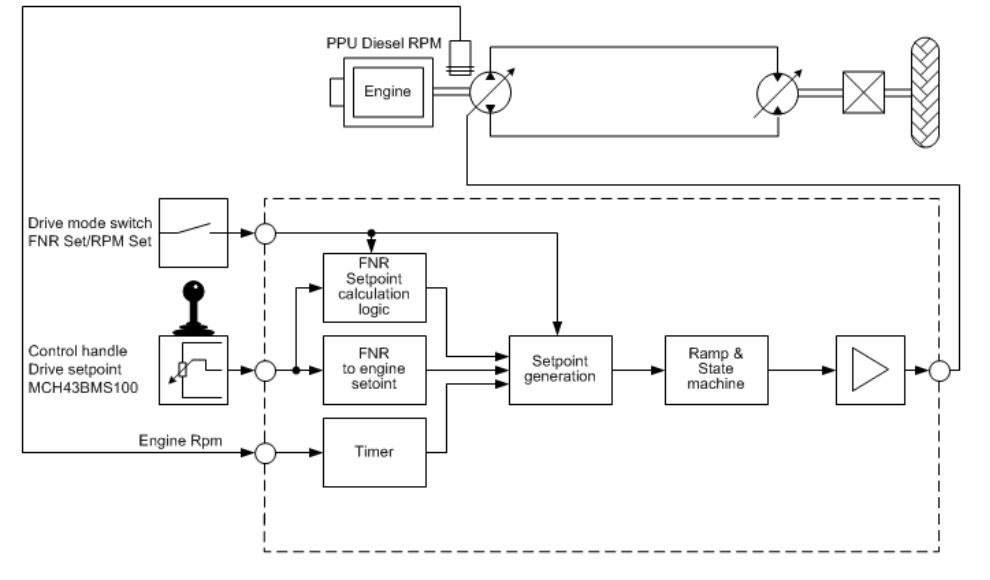


Figure 001

General (continued)

Example of a Software Block Diagram

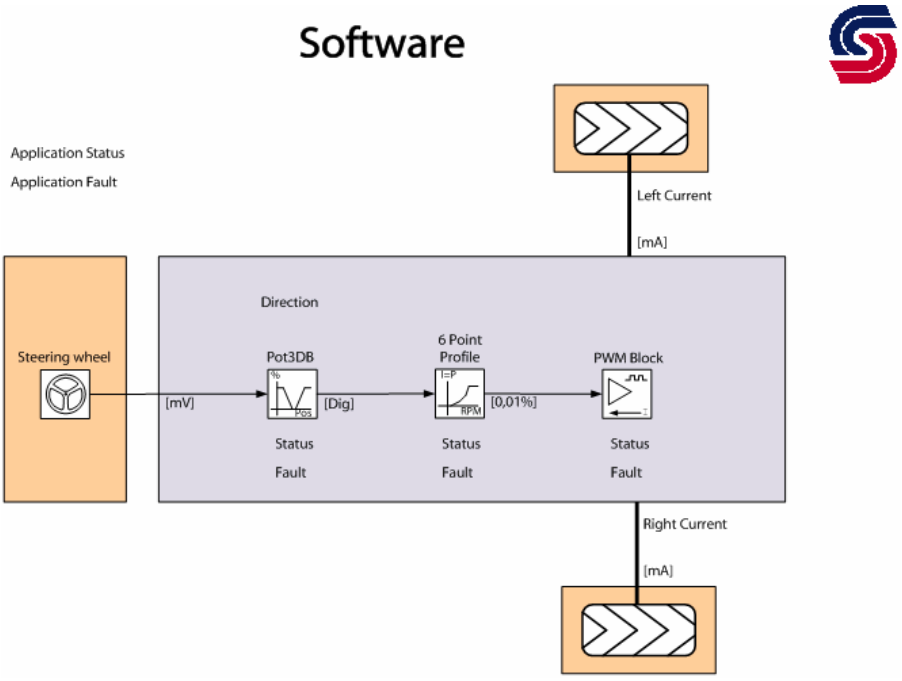


Figure 002

The focus here is on the structure and modules. If this is well understood, further thoughts can go into the software details.

Example of a Complex Software Block Diagram

This example shows a larger diagram; the details are intentionally ambiguous in order to focus on the logic of the blocks.

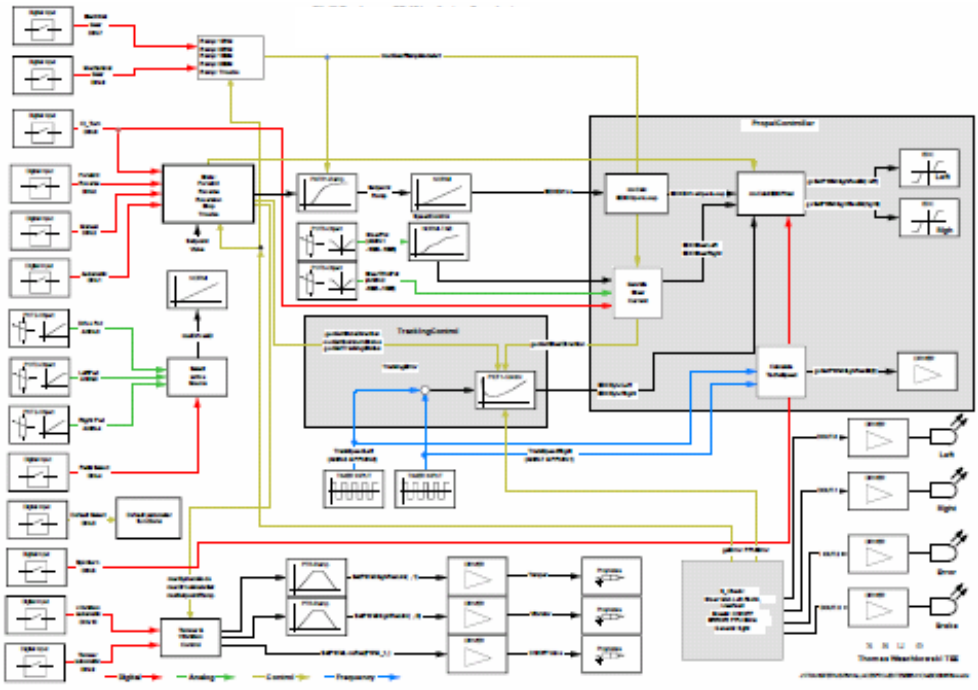


Figure 003

It is recommended dividing bigger problems into smaller ones. For example, the gray box above includes a logical control. Lines can be wiped out and only one box can represent the control instead of the 7 boxes shown in the diagram.

General

Many publications have been made about graphical software implementation. The general idea is that one can be more efficient with graphical implementations. To achieve this statement, a specific discipline is required.

Coding

Coding is a method to implement the specification. Coding itself is not a software design. Coding is an implementation of a software design.

The success of software solutions compared to hardware solutions is in the area of the implementation flexibility. The software can be easily changed. Many different solutions can be implemented in many different ways.

Many different coding implementations can be delivered. It should be the goal of every SW engineer or developer to create code in the same style.

This document tries to bring good practices to a broader public.

Diligent Editing

Keep your source code clean. This chapter begins with an example.

Negative Examples

The drawing space of GUIDE is free format, but you should try to avoid these types of drawings:

Here, the Blocks are placed First and the Lines have Steps:

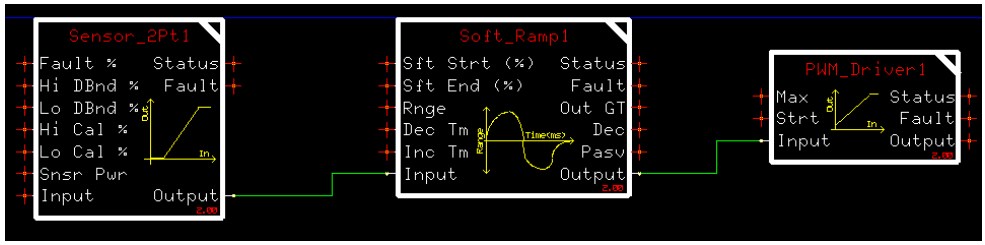


Figure 004

After Aligning the Blocks, the Wires look like this:

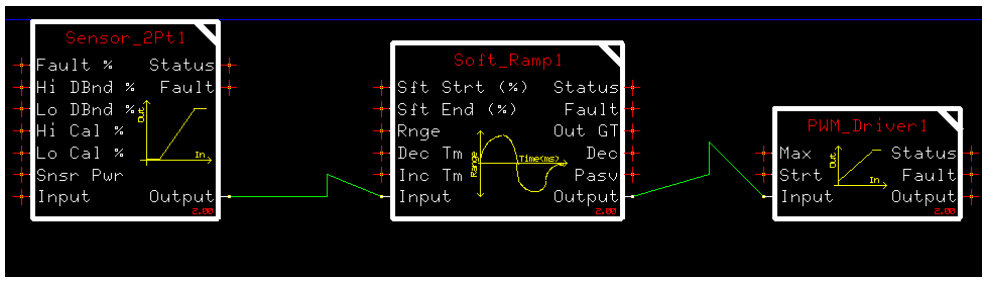


Figure 005

Wire runs over a Block

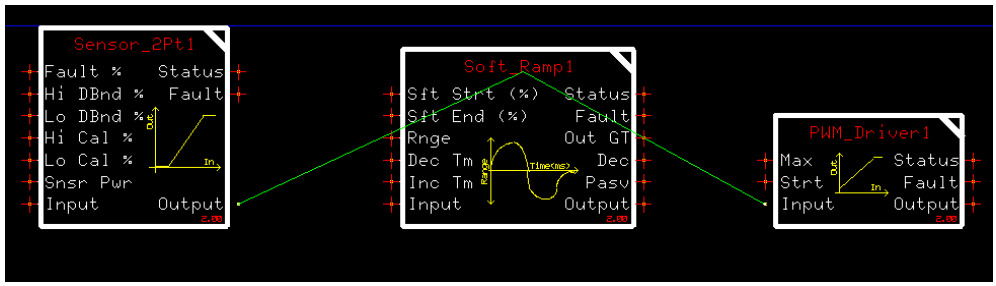


Figure 006

Negative Examples (continued)

What looks ugly is not recommend:

Phantom Wiring

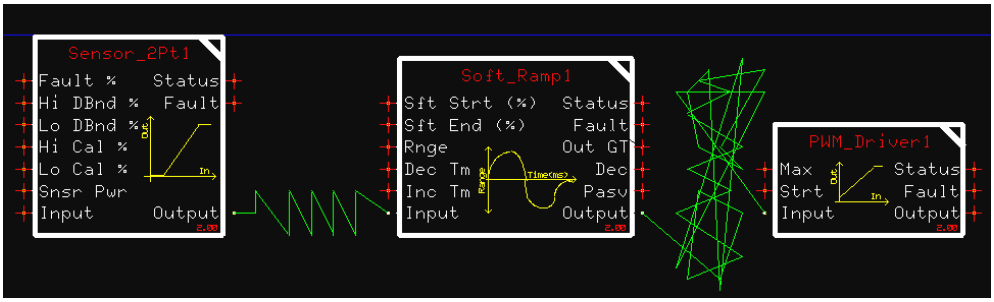


Figure 007

Incorrect Order: Inputs should be on the left Side; outputs should be on the right Side. (The output gets its input one loop time too late).

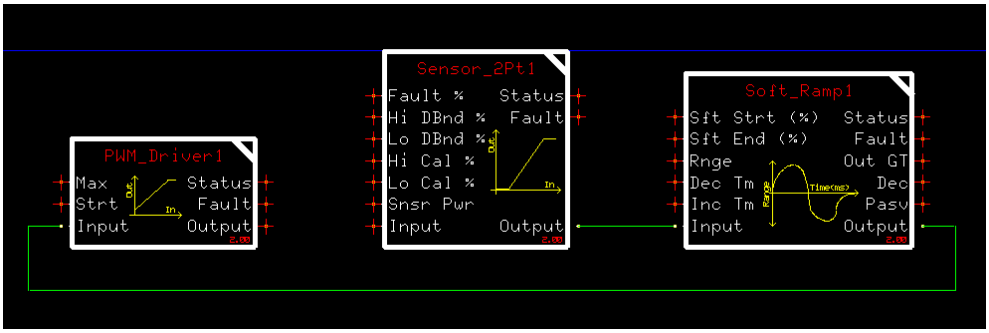


Figure 008

Positive Example

And finally: This is the correct way to do it.

It is better to complete the Step and Align the Wires again.

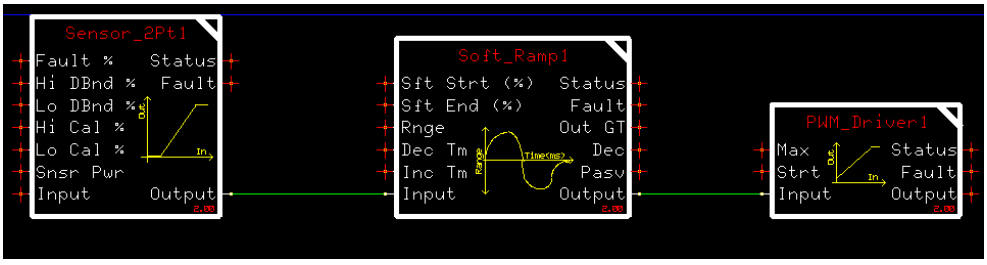


Figure 009

General

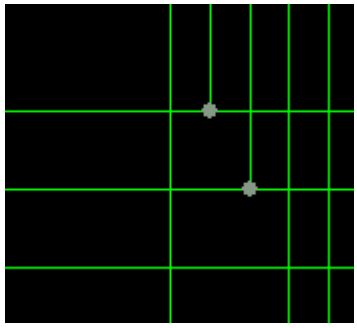


Figure 048

Except for the exceptions described below, try to make all routes either vertical or horizontal.

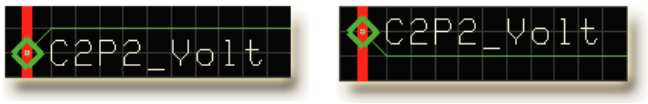


Figure 042

Avoid tangled routing. Tangled routing makes it difficult to understand and update an application.

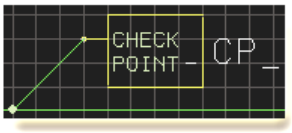


Figure 043

Checkpoints, being part of the application, should be connected as any other symbol. If using checkpoints only for debugging, place them outside of the page border. Consequently you will get a warning which allows you to find these points very easily.

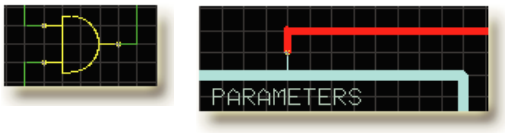


Figure 044

For legibility, route buses and wires at least one grid away from pins before changing the routing direction.

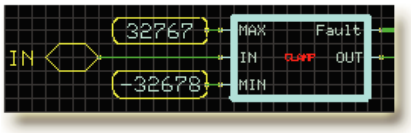


Figure 045

Align all application elements on the 2.5 mm x 2.5 mm grid.
For legibility, keep a one horizontal grid distance between components and page pins.

General (continued)

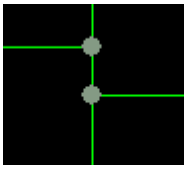
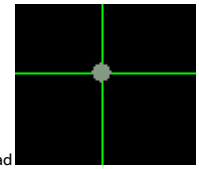
Good  Bad 

Figure 051

For legibility, separate connections to a route, as shown in the “Good” example. When zoomed out, connections made to a single point, as shown in the “Bad” example, look like routes crossing.

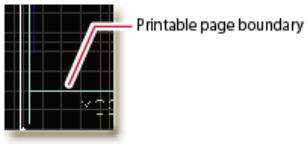


Figure 046

Keep all application elements inside the thin light blue line that defines the boundary of the printable page. Elements placed outside this boundary will cause compilation warnings. (The warning was implemented to help avoid someone putting all symbols on one page.)

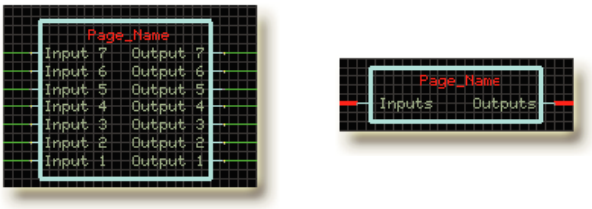


Figure 047

Try to use buses to reduce the number of wire ports on a page.

Example for an Easy Solution

This example demonstrates the advantage of simplifying an implementation.

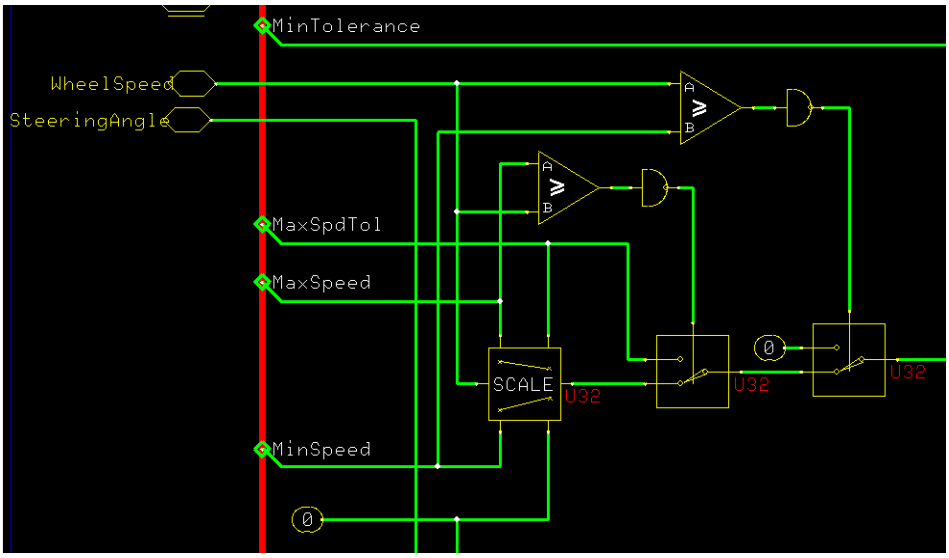


Figure 018

Function:
The scaling of a value with upper and lower limits for the input is shown in figure 018. The Outputs are limited in this example to “MaxSpinTol” and “0”.
Software Code Reviews are recommended and rethinking the source code comes up with: If “WheelSpeed” is bigger than “MinSpeed”, then (not) the output is set to zero? Using the inverse logic is no problem for experienced programmers, but it is much more difficult for others to understand.

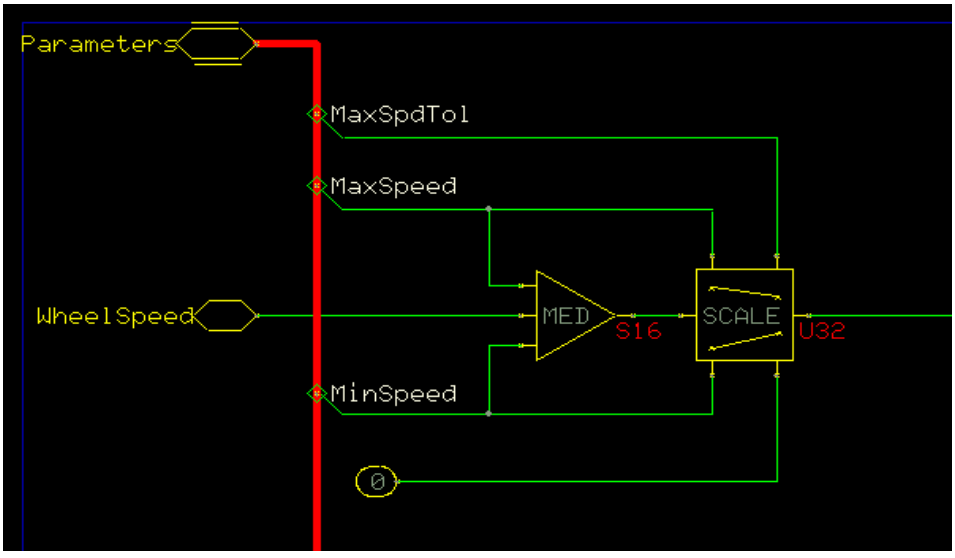


Figure 019

In the revised example shown in figure 019 above, the functionality is much easier to understand. “A WheelSpeed” is normalized between minimum and maximum, the output is between 0 and MaxSpdTol. The code quality is better by eliminating crossing lines, using only two symbols instead of seven and using nine green lines instead of twenty one.

Example for Simplification in using Guidelines

Simplification is very important for code quality and this Guideline will help.
Here is another example:

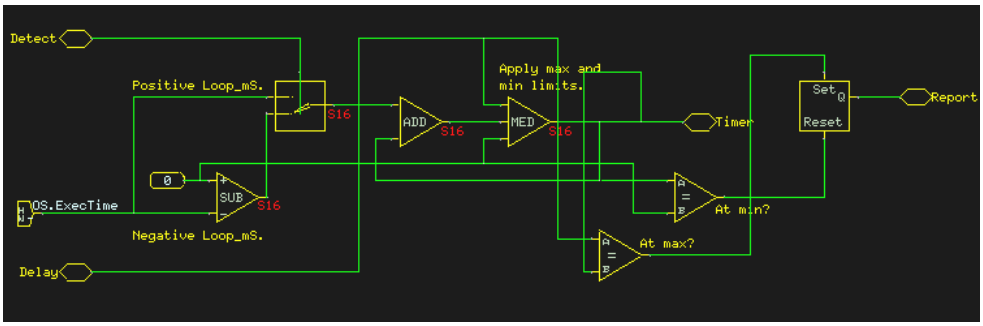


Figure 020

The developer of the example above did not follow the Guidelines: more lines are crossed than needed and the same function quickly becomes complicated and difficult to review.

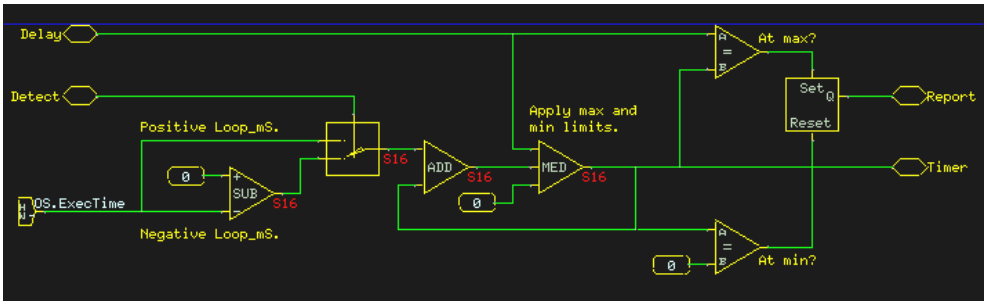


Figure 021

One function of the example above is: If a delay is at max, a report is generated.

Redundant Code

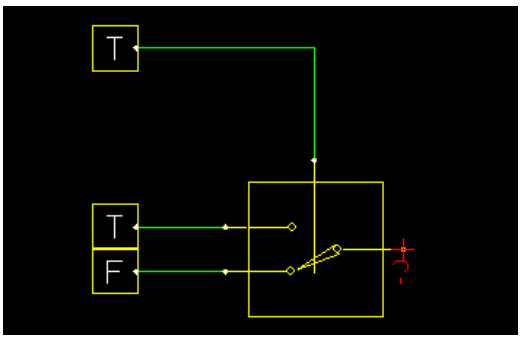


Figure 022

Avoid redundant code. This code is doing nothing other than keeping True as True and such logic should be avoided.

Checklist

Do

1. Flow data from left to right. Wires enter from the left and exit to the right.
2. Align and distribute functions, terminals, and constants.
3. Version control of important functions should be implemented.
4. Consider the reusability of the implementation.
5. Make sure the program can deal with error conditions and invalid values.
6. Rely on the correctness of a library function block; therefore not every output needs to be double checked.
7. Arrange the logical solution on a dedicated area.
   Centralize topics and information within the dedicated page.
8. Review the code for efficiency and accuracy.
9. Use the error signals of the function blocks.
10. Use a bus for “defines” (Type definition, compare enumerate, transition into Boolean with name) when there is repeated use.
11. Use the bus only for dedicated meaning.
12. Use a separate bus for parameters.
13. Use a separate bus for working variables.
14. Do not mix working variables and parameters.
15. Use a unidirectional bus for working variables.
16. Bi-directional use is only acceptable for parameters.
17. Add comments liberally to aid understanding.
18. Use text comments to indicate the purpose of various parts of the page.
19. Document the implementation.
20. Explain abbreviations (standard list in GUIDE).
21. Use a glossary when possible.
22. Use variable sizes applicable to the requirement.
23. Names and variable names should be clear in meaning and function.
24. Names should be defined with consistent logic.
25. Notation of naming should be followed strictly.
26. Meaningful page names reflect their function.
27. Graphic implemented to illustrate the function of the page.
28. Use error handling consolidated in a separate block.

Avoid (do not)

1. Avoid placing blocks and pages on top of wires.
2. Avoid creating pages with too much code on one page.
3. Do not duplicate code.
4. Avoid redundant code.
5. Do not use short names when not needed. Longer names are more easily understood.

General

Textual programming can have a little different way of thinking. Here is an example from the C-Programming world to show the difference:

The example shows a graphical implementation of how to consider an "If / else statement":

```
If (In > 0)
{
    Dir = 1;   /* Right */
}
else if (In < 0)
{
    Dir = -1;  /* Left  */
}
else
{
    Dir = 0;   /* Neutral  */
}
```

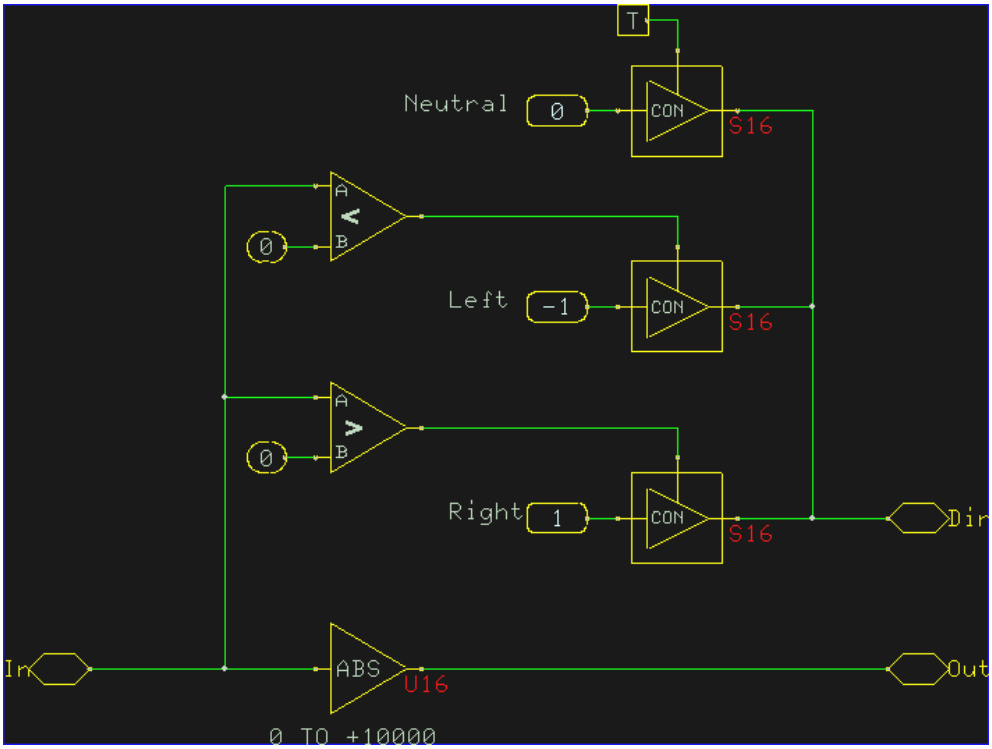


Figure 023

Note: The implementation above does not exactly reproduce the c-code. The focus is for the c-code and the GUIDE code to have the same functionality.

Before

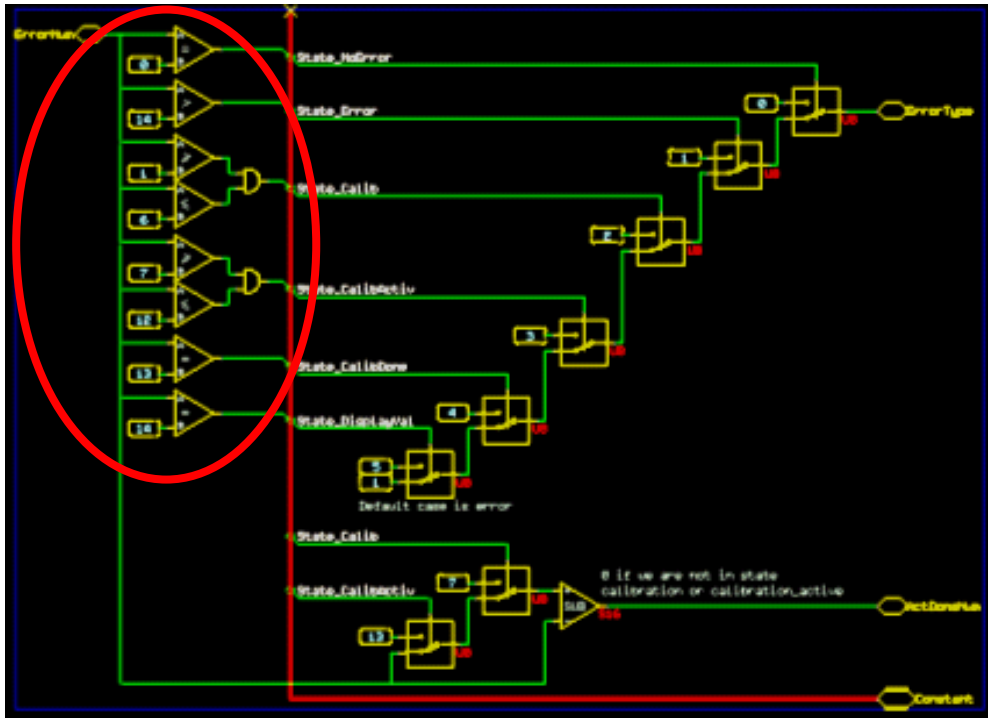


Figure 024

General statements about the example:

1. A single instance of a port (same name) gives quick visual information about the inputs/outputs.
   Only ports present on the top view should be used since a port indicates that it is an input/output to the page.
2. A state should be a bus containing the different states. With Boolean states true or false, this makes it possible to use the state directly. This makes the code more readable.
3. Place the input ports to the left and the output ports to the right. This makes the code more readable.

Simplification on this page:
The red ellipse on the screen shot is the focus for this example:
According to the Guidelines defined earlier, there is too much logic on one page view.
Please refer to the next figure, where the simplification is discussed.

After

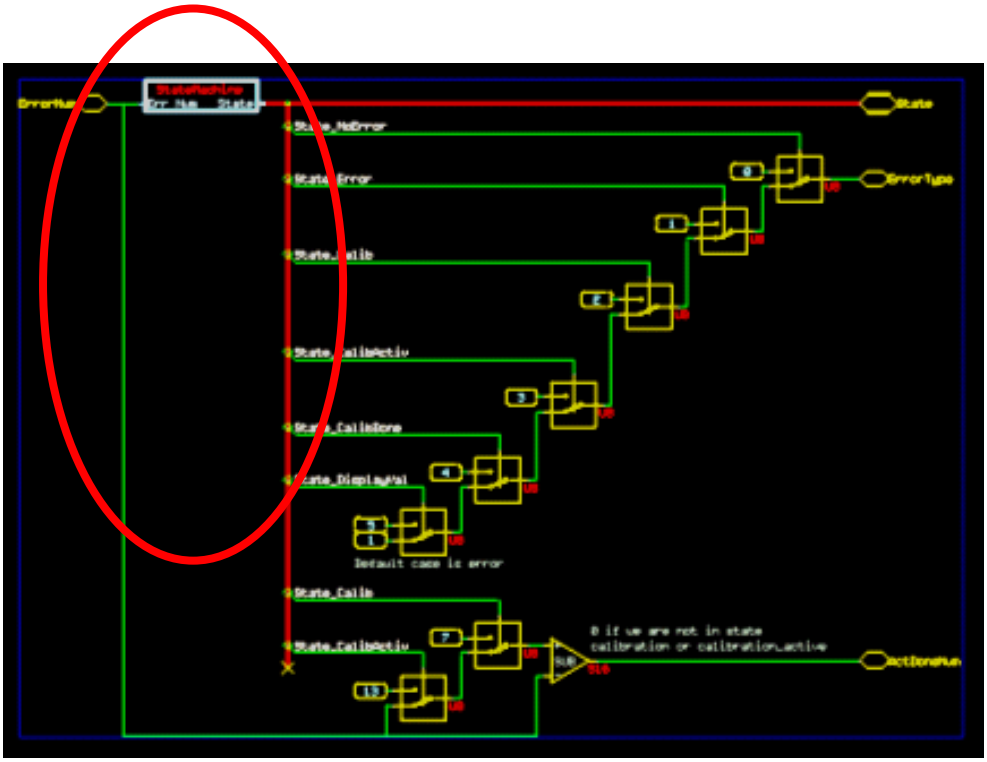


Figure 025

The simplification is done by moving the details into a sub-page.

This is the Content of the Sub-Page:

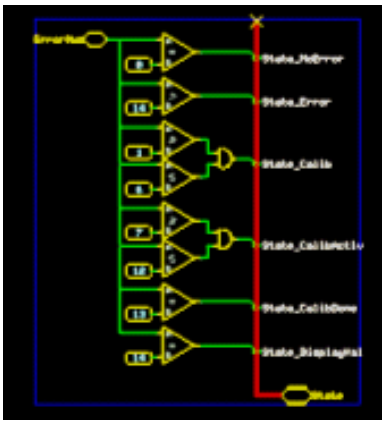


Figure 026

Page Levels

When creating an application, the nesting levels of the implementation should be consistent.

Top Level

This is Level 1

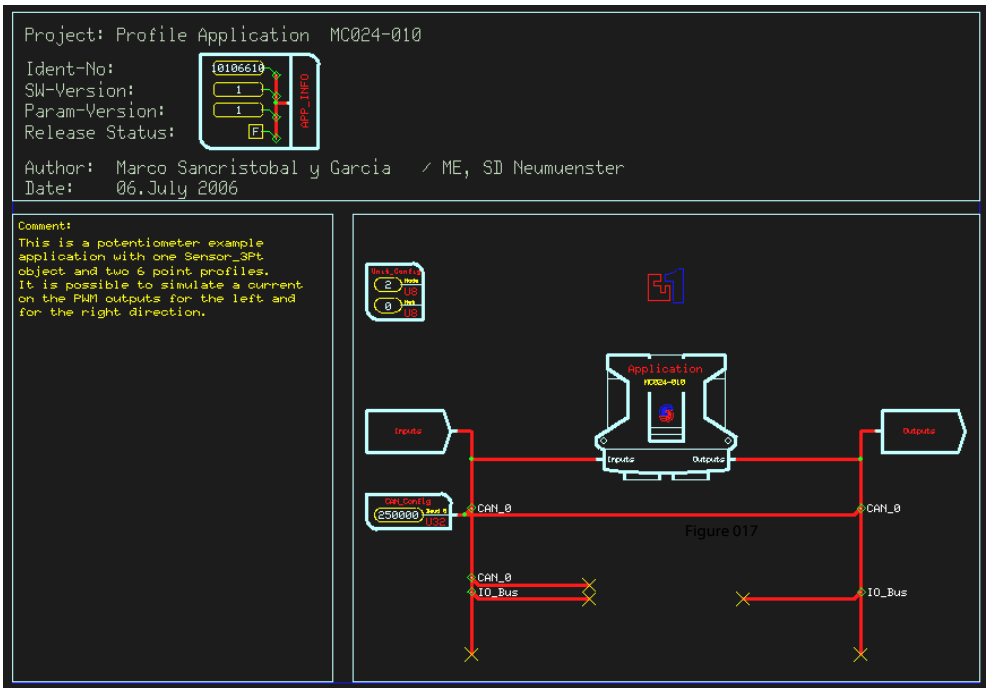


Figure 027

The top level is the top level of the template.

Page Levels (continued)

One Level Below

This is Level 2.

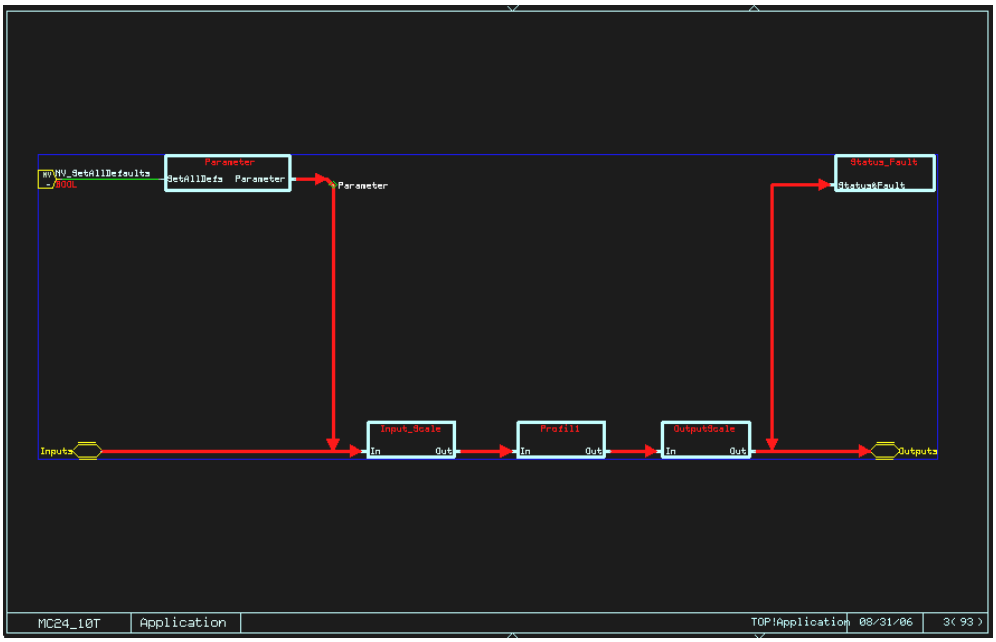


Figure 028

One level below, the application overview should be visible.

Detail Levels

This Example shows Level 3.

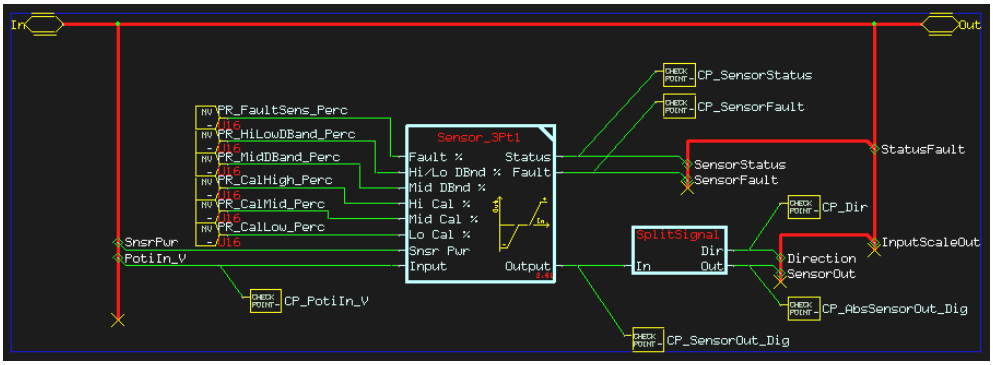


Figure 029

More detailed implementations should be done in level 3.
If there are too many requirements to address at this level, a deeper level can be implemented. This would be level 4.

The deeper the level, the more detailed is the functionality.

Page Structure

The structure of a page is crucial for the design.

These are the general recommendations:

- Organize your page structure in a way that reflects the logic of your application.
- Give your page a meaningful name that reflects its function.
- When possible, create a graphic to illustrate the function of the page.
- Use text comments to indicate the purpose of various parts of the page.
- If an experienced PLUS+1 programmer can understand the purpose and function of the page within ten seconds, you have created a successful page design.
- If non-experienced programmers can understand the purpose as well, you have implemented a good design!
- Carry hardware signals to and from the pages that you place within the Application page. Use the buses connected to the Hardware Inputs and Hardware Outputs supplied with the template.
- Divide functions into pages to make them easier to understand.
- Do not use hardware connectors to carry signals to and from pages. Using hardware connectors within pages makes applications more hardware-specific and less portable.
- Avoid “pass-through” signals when possible. This type of signal passes through a page but is not used within the page or its sub-pages. Creating entry and exit points for these signals clutters the layout of the page.
- Avoid creating multiple entry and exit points in a page for the same signal or bus. Use care when rotating functions and components. Rotation may cause problems in the execution order.

Signal Flow View

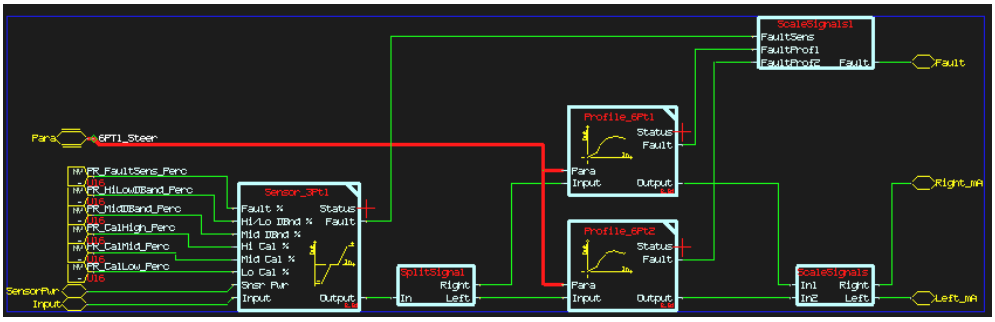


Figure 031

It makes absolute sense to consider the logic of your signal flow.

This example shows a top view where you can clearly understand the signal flow of your software. The signal entries are on the left side, the signal flow remains straight forward from left to right, crossing is avoided by using sub pages, and the implementation is complete on the page.

Buses are used with clear structure. Indeed, this is not always possible.

Each block is designed with a graphic which explains the functionality at just a glance. This illustrates one of the main advantages of graphical programming - it guides you through the code and tells you something about the software design.

Specifically during a code review, a person who has not seen the code before, will be able to understand more quickly what the software is supposed to do.

It is recommend to implement code in this way even if the total memory usage might be higher.

Confusing Flow

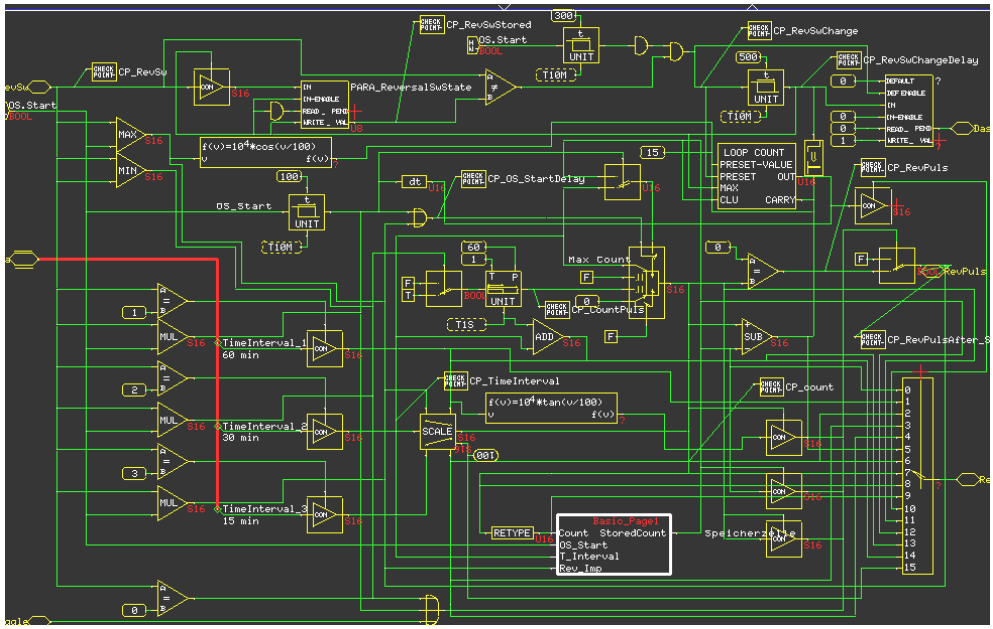


Figure 032

This example shows the opposite of a clear implementation - the page is simply confusing.

This style is sometimes referred to as **green spaghetti** and should be avoided.

Specifically, “new” PLUS+1 programmers will have difficulties in understanding the logic of the implementation just by looking at the “picture”.

It may take a long time to understand the functionality of a page. It may take even more time to implement a small specification change.

Handing over software to a colleague will definitely take a longer time than in the first example.

Position of Ports in a Page

Inputs should be on the left Side, outputs on the right Side.

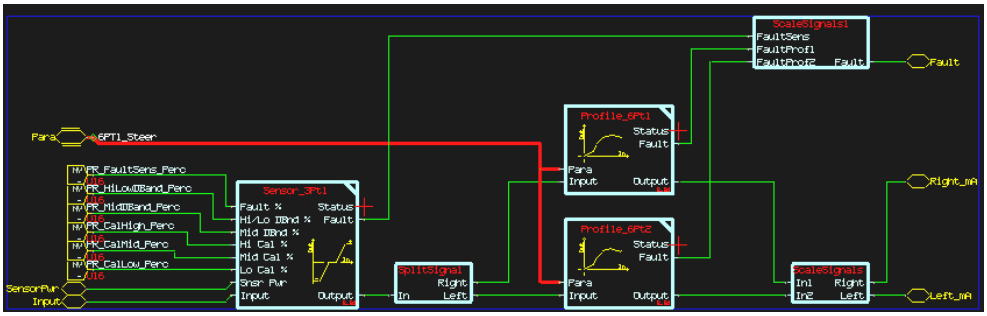


Figure 033

The overview of a page sometimes needs a compromise. A port sometimes needs to be in the middle of the page.

Data Consistency

The consistency of values must be equal. Mixing up physical dimensions is like comparing apples and bananas.

Please add only ***current with current*** or AD converter digits if they have the ***same scale.***
Closed loop controls must operate with the correct scale. Please do not substitute the scale by just using bigger parameter values.

Example: current controller –
It should be
Delta Current [mA] = Current Setpoint [mA] – Current Measure [mA]

And not
Delta Current [A] = Current Setpoint [mA] – Current Measure [Digits]

Please consider that closed control outputs are used in the same direction and meaning as discussed for the variables above.
A closed loop output of RPM should be scaled into a current before being used in a summantion for that current.

Useful Comments

Comments are helpful for Reviews and Maintenance

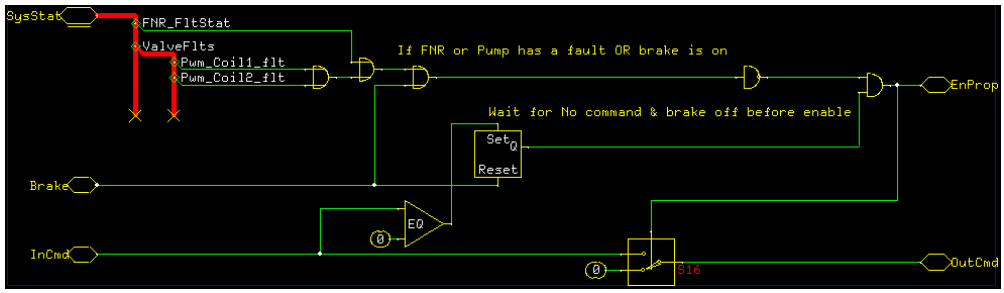


Figure 034

The comments are very helpful in this example. The two lines of text explain what the logic is doing.

Combing comment number 2 with comment number 1 allows us to make the following statement regarding the function:
"The input command is only passed through to the output if there are no FNR or Pump faults AND the input command has been zero since the brake was released. A subsequent brake command will set the output to zero."

Comments in a Box can contain General Explanations.

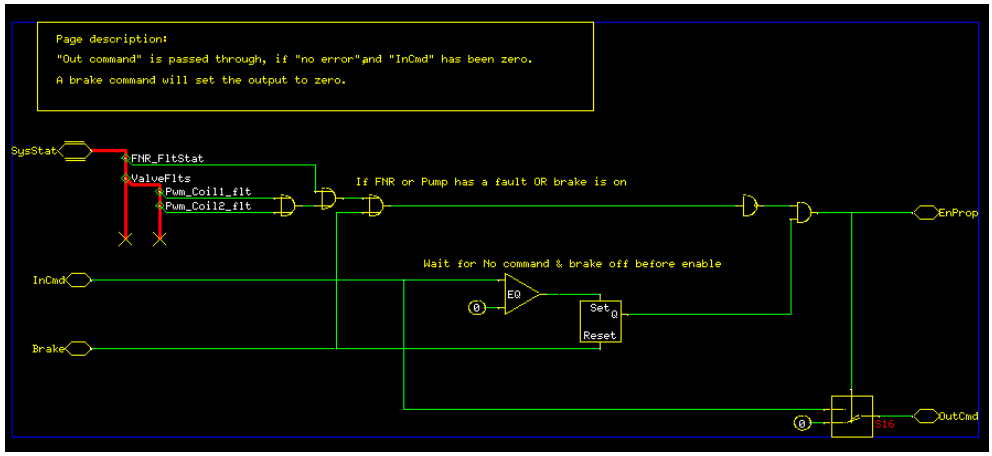


Figure 035

This page has been cleaned up a bit now. What is different?

- A page comment has been added.
- The switch has been moved further to the right.
- The result of the "Brake" will be executed in the same loop instead of the loop after as in the previous example.
- Although we have two crossing lines instead of one, the flow is clearer.

Missing Descriptions

Example Implementation That Doesn't Conform to Guidelines

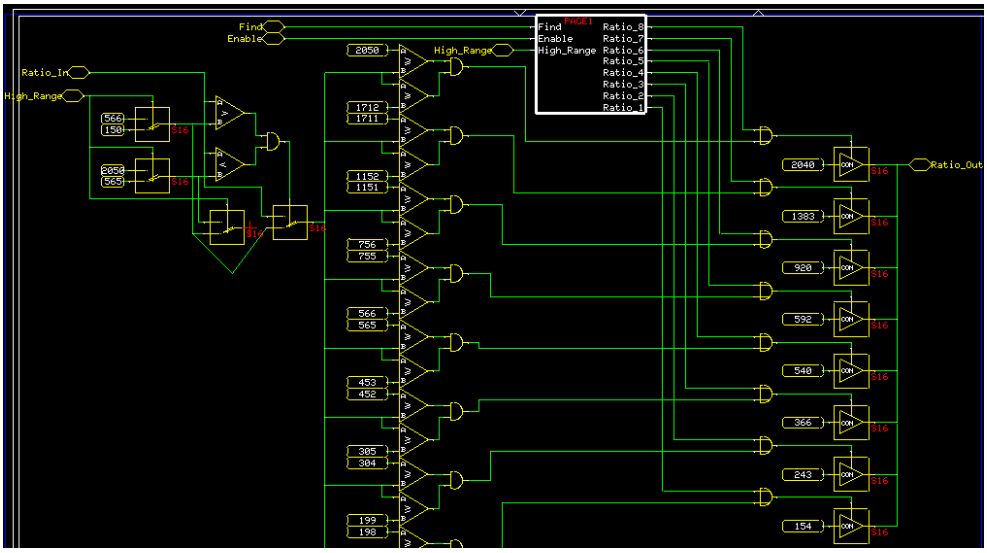


Figure 036

There is no description at all.

Missing descriptions of parameters and the numbers used for calculations caused confusion during reviews and software maintenance.

Using a number like "2040" in this example is easy to type, but there should be a ***name explaining the meaning of this number***, specifically if you want to reuse it in other pages.
Using just the number makes it difficult to understand the objective of an implementation.
A better implementation is to use a define Bus where the number 2040 is linked to a name e.g. "MaxGearRatio".

Possible improvements:

- Add page name as a comment
- Add a short description of the page functionality
- Add references to external documentation, definitions and requirements
- Explain numbers if they are used

Please refer to the chapter on variable naming convention. This is a very important factor for the clarity of an implementation.

Function Block Top View

Function Block Top View - Level 1

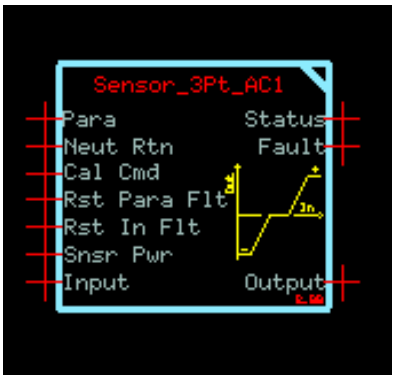


Figure 037

Function Blocks should be designed with a clear structure as well. An example function block is shown above.
The design is more or less self-explaining. The symbol is lean, but shows the block function. The title is clear and a version number is included. (0.00 = draft).

Underscores are not used in variable names, the graph is yellow, the page name is red, and any added text should be yellow.

Function Block Lower Level

Function detail view - Level 2

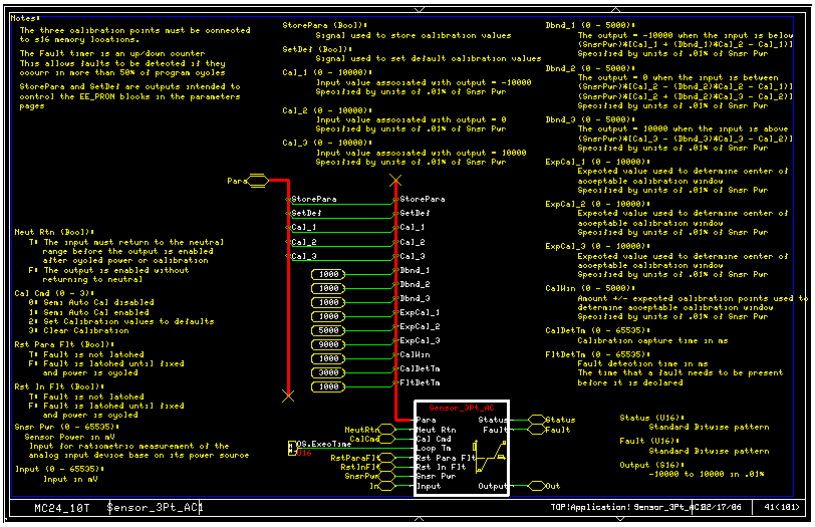


Figure 038

The level below explains the details of a function block.

Yellow test describes the possible configurations of the block. In the ideal case, a user manual of the function block is not needed if the description is complete.

Error Handling of Applications

Use a standardized P1D outline logic.

Flash Codes for Error Handling

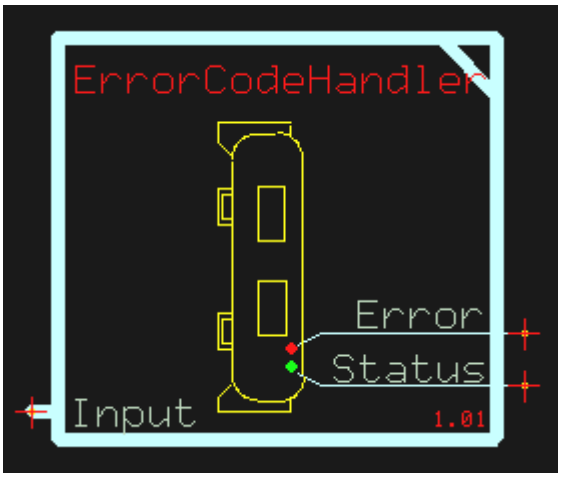


Figure 039

Use a standard error code function block to automatically follow the global standard for flash codes.

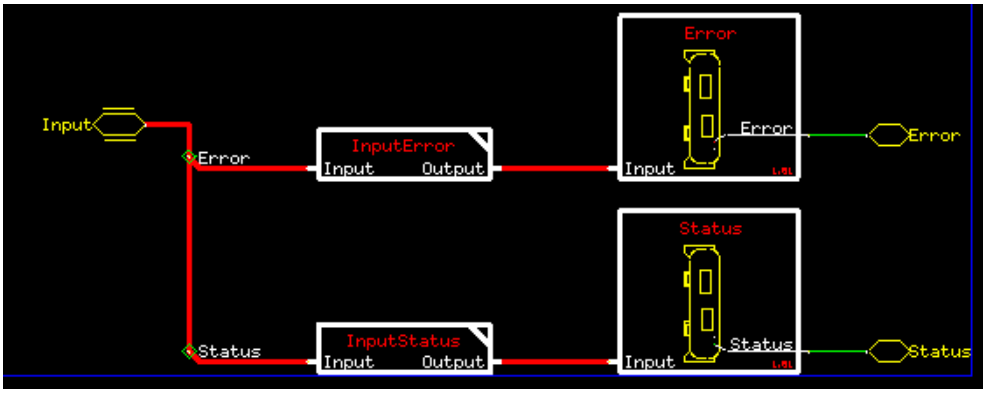


Figure 040

This is the inner view. Errors are shown using the red LED. Status is shown using the yellow LED.

Flash Codes for Error Handling (continued)

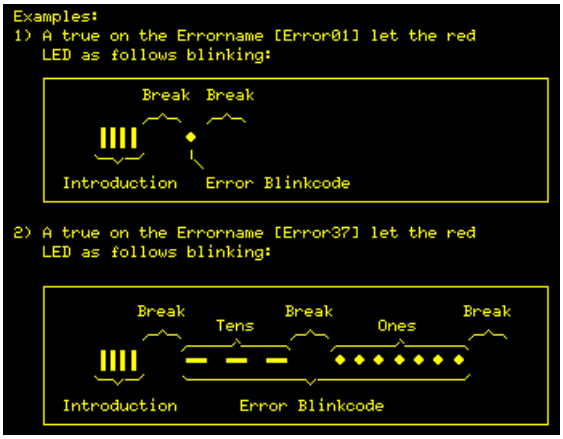


Figure 041

The flash code follows the logic shown above.

Information about the Software is needed to identify the software version and the Status of the software.

Material Number

Material Numbers (also known as Part Numbers) are used to define a unique system.

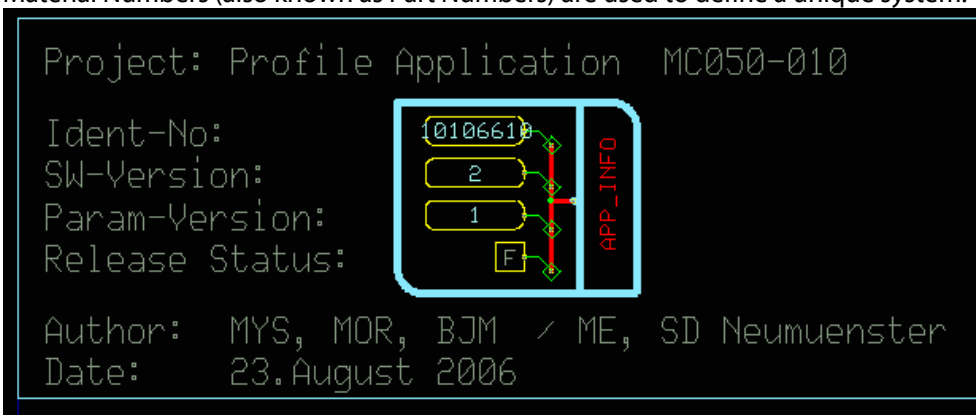


Figure 60

Application Screen

An application screen on the service tool is the first screen visible when connecting to an application and loading its P1D file. General information like part number should be displayed here.

Place the following information:

- Ident Number
- Software Version
- Parameter Version
- Release Status

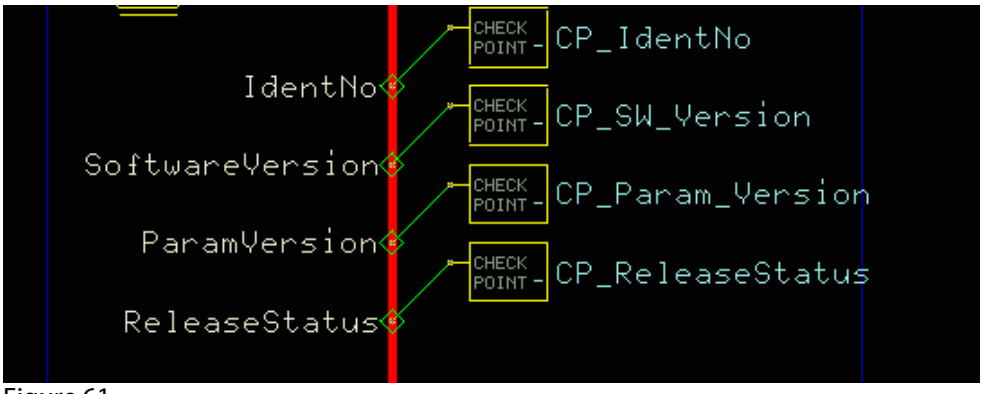


Figure 61

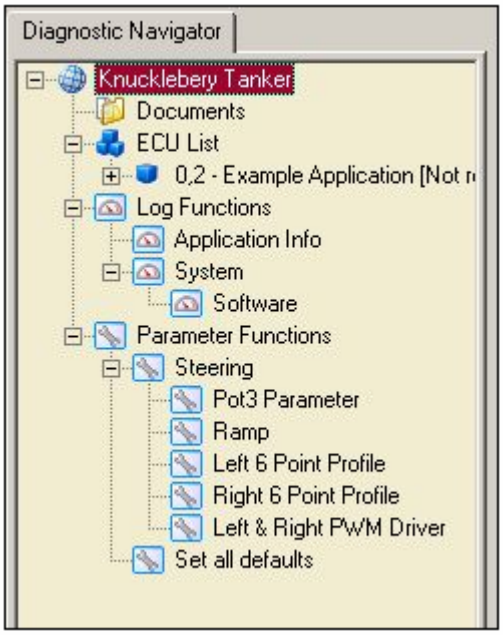
Guide Service Outline

Service tool standardization is important for service and maintenance. The cost savings are immense if an organization with engineering, service and customer support does not need to retrain personnel for each new application development. If the structure and the outline of available screens follows a consistent style and guidelines, those screens are mostly self explaining.

Recommendation - use a standardized outline and logic for the screens.

Structure on Diagnostic Navigator

Service Tool Screens should be arranged in this structure:



Application Screen

An application screen is the first screen visible when connecting to an application. At this position basic and general information like part number should be displayed.

Application Screen Example

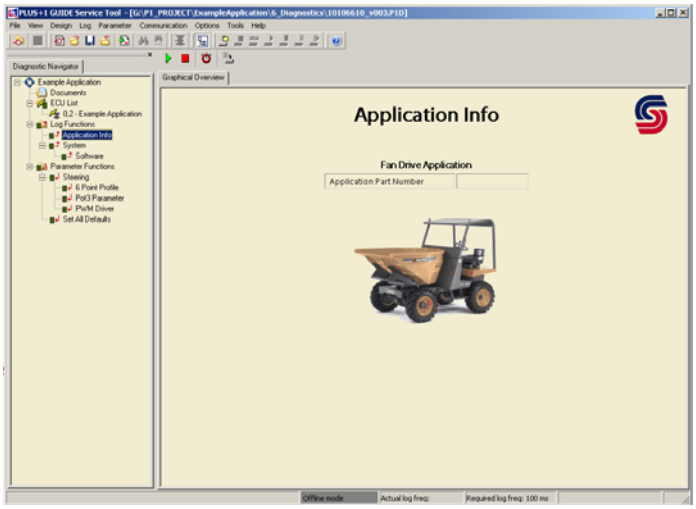


Figure 042

System Information

After the application Info the next screen is the System Screen.

System Screen Example 1

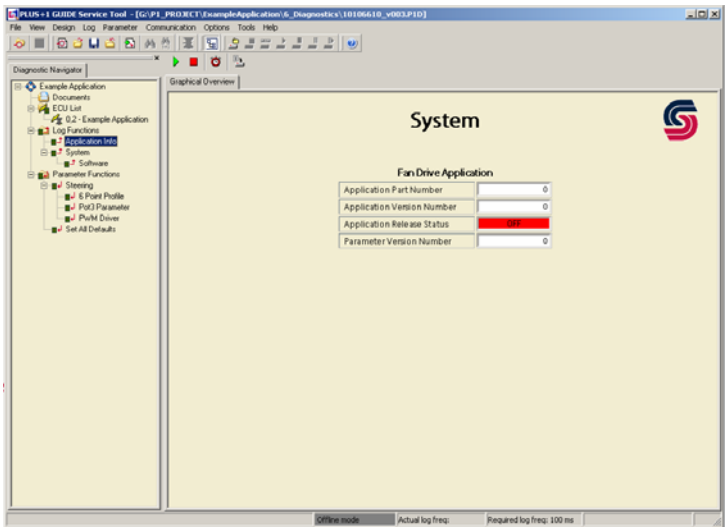


Figure 043

System Screen Example 2

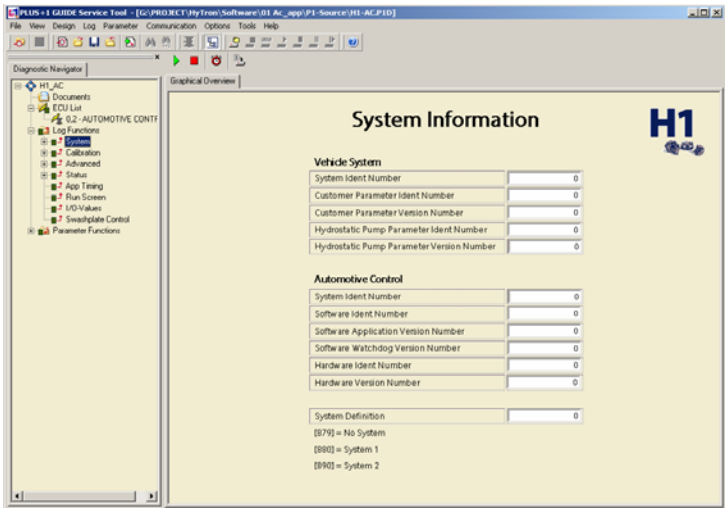
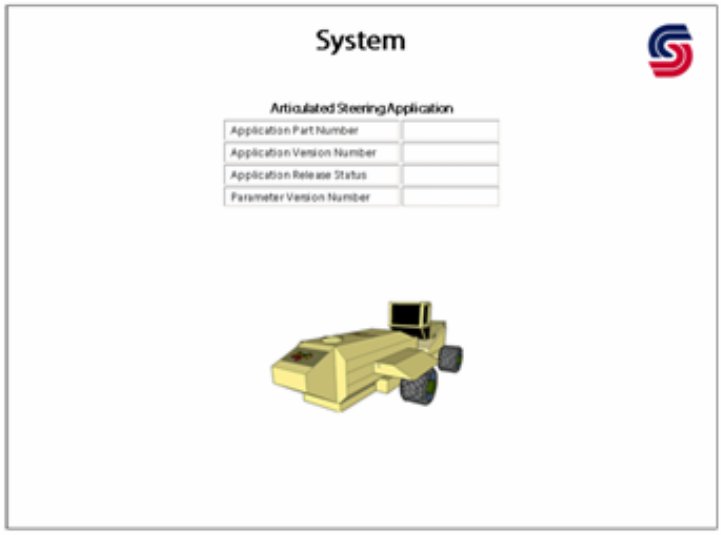


Figure 044

System Screen Example 3



The appearance of the screens should be consistent for each application.

It is important is to keep the corporate Identity and information compatible.

Software Screen

A software screen is the main view into the application. The general logic of the application should be displayed here. It is good style if the “software design” can be recognized on this page.

Software Screen Example 1

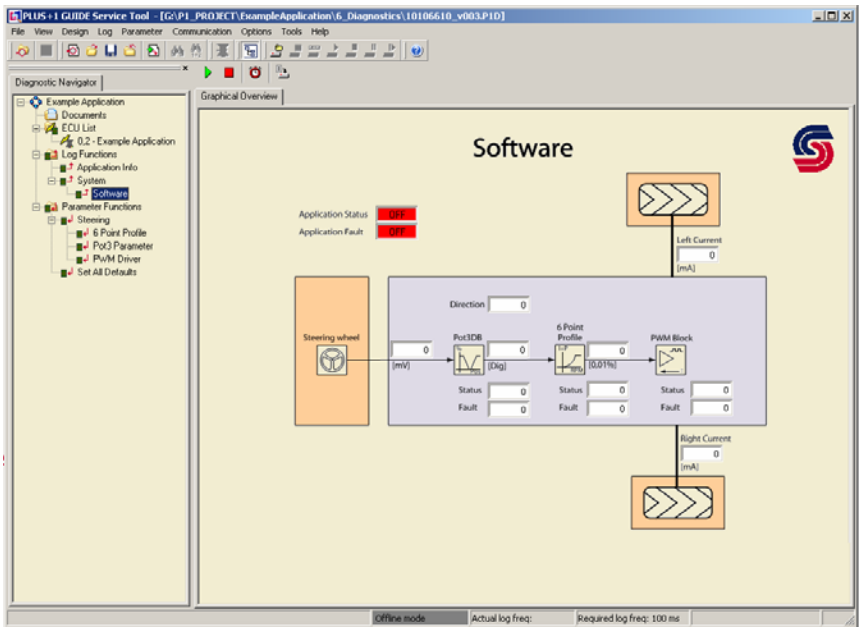


Figure 045

Software Screen Example 2

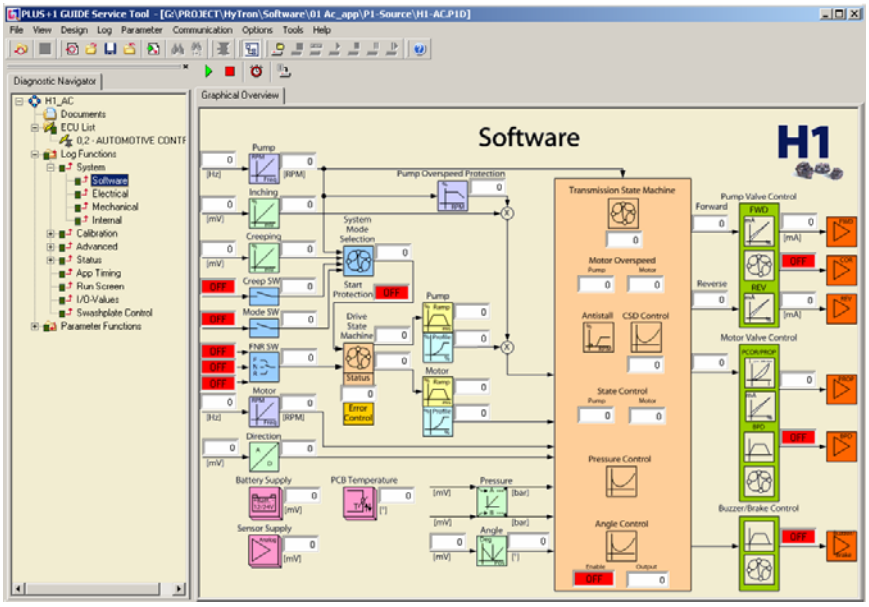


Figure 046

Parameters Screen

A parameter screen is used to modify values that are used inside of applications. For example, if the time used in a ramp function needs to be adjustable, these values can be accessible via parameters.
The clarity of the service tool page has a large influence on the fault tolerance of a system. If the service tool page is clear, there will be less faulty adjustments done by the user.

Parameters Screen Example 1

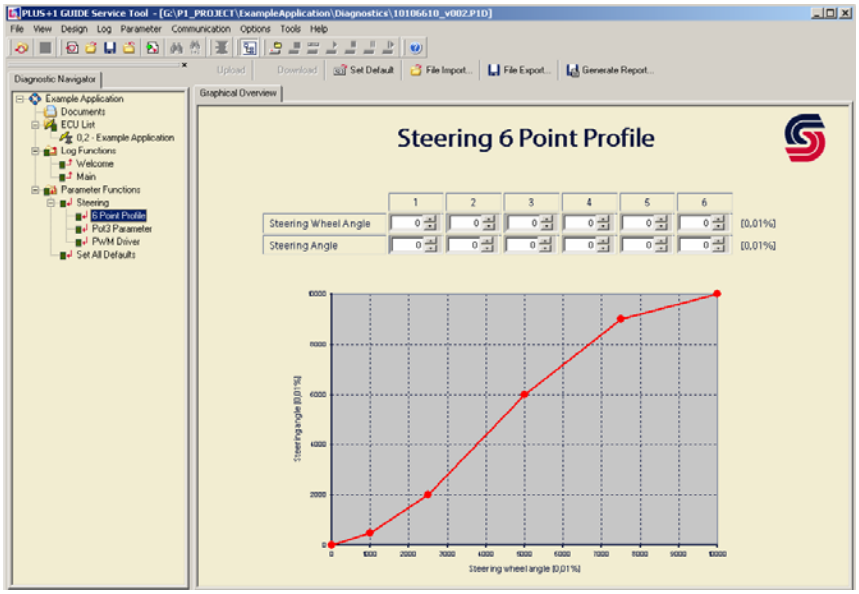


Figure 047

Parameters Screen Example 2

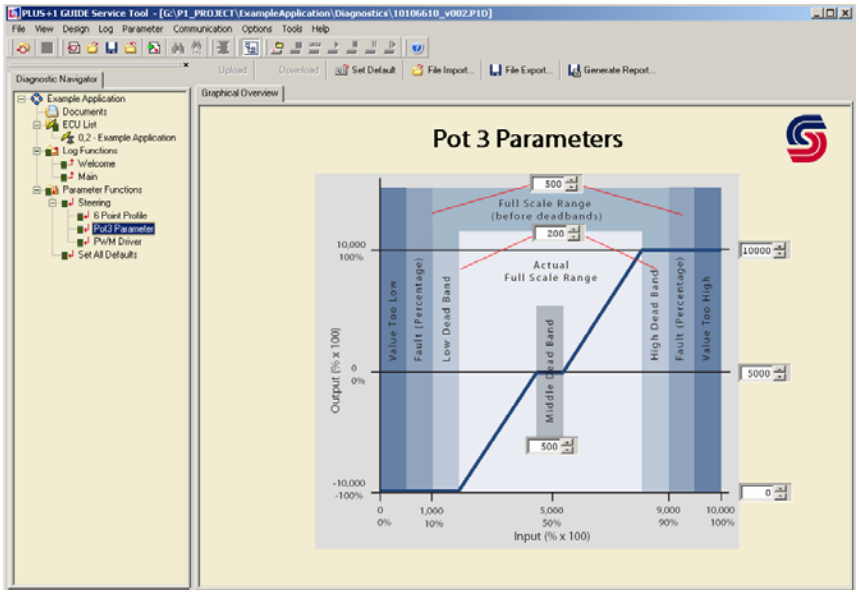
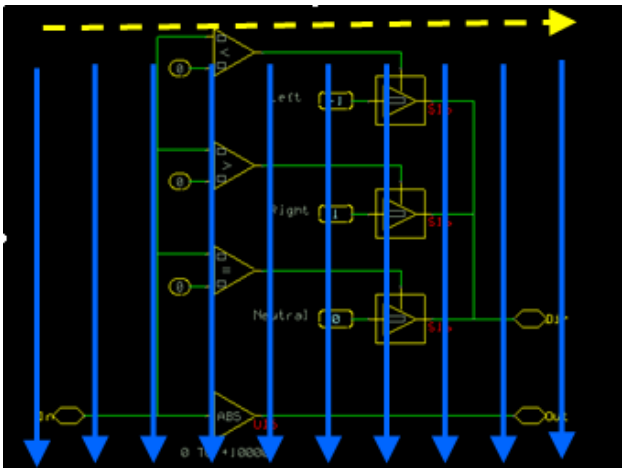


Figure 048

Execution Flow

It is important to understand the order of execution in a GUIDE application. If one calculation is based on the result of another, the result should be ready before it is used.

The execution flow is from top left of the page to the bottom left of the page, continuing with the next top pixel and moving right until the last one (referred to as *pixel scanning*).



The blue arrows in the pictures above show the execution order.

Loop Time

Check if the loop time is too long.

The PLUS+1 software kernel automatically checks the runtime of the active task (loop time). If the actual loop time exceeds the application setting, a longer run time is set by the kernel.
Ensure the task time has the correct setting.
The correct loop time is set in milliseconds. If the run time is the same as the set time, then add one millisecond to the set time to give some headroom.

Number of Sub Pages

Number of sub pages:
There is no limit. Use as many levels as the best solution requires.

Safety critical functions

There is no ISO 61508 implementation defined. Use a system FMEA (Failure Mode Effects Analysis) and review with the customer.

Certified for Service

A code review and design review can result in a process to certify software for Service. In this case Sauer-Danfoss can do field service on machines equipped with Sauer-Danfoss software (cheaper).

Certified for Maintenance

A code review and design review can result in a process to certify software for Maintenance. In this case Sauer-Danfoss can develop changes or modifications to customer released software (cheaper).

Parameters inside the Application

Application software should verify the status of the parameters being used.
If, for example, after downloading in production, no parameter set is valid, an applicable default parameter set should be loaded and stored. This will ensure the correct functionality of the application.

The application needs to ensure that the behavior of a machine is safe even without adjusted parameters.

Startup Behavior Considerations

Application software normally is specified for a running system. Often the requirements are different in the scenario where a system is being started.

An engine might be already running and the embedded control is starting. It would be an unexpected behavior if the machine immediately moves ahead. These situations are addressed with specific functions for specific cases.
For example, a START Protection system can prevent movement during system startup.

The application needs to ensure that the behavior of a machine is safe even during system start.

It is recommended that the developer fully consider the startup sequence of the controller. For example:

- Is the Digital Output to battery plus or to ground?
- Is the initialization of variables consistent with the connection diagram?

General

Variable names can explain many details about the functionality of the software. The following pictures will explain how variable names are structured.

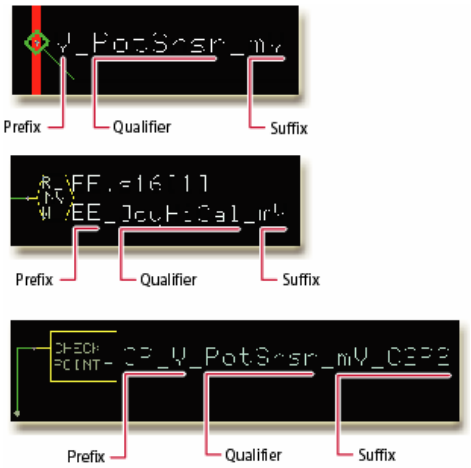


Figure 049

Variables include user-defined signal wires, sub-buses, checkpoints, and EEPROM aliases. These guidelines do not apply to variables that are part of the application templates.

User-Defined Variable Names

Variables consist of a prefix, qualifier, and suffix:

Prefix

Prefix - Identifies the variable type, such as Boolean (B) or checkpoint (CP).

1. The prefix helps identify variable types in sorted lists.
2. Not all variables have enough space for a prefix.
3. Capitalize all prefixes.
4. Separate prefixes from the qualifier with an underscore (_).

Qualifier

Qualifier - Describes how the application uses the variable.

1. Use mixed case formatting to make the qualifier easier to read. Capitalize only the first letter of each word or abbreviation.
2. Concatenate (run together) abbreviated words used in the qualifier.
3. Avoid letter combinations, such as lH and lD, that are hard to read on the screen:
4. Use descriptions that indicate the true state for Boolean signals, such as SetDefAll (Set Defaults All), Fnd (Found), and Pass.

   Preferred

   JoyHiCal, SetDefAll

   Avoid

   CalHiJoy, SetAllDef

5. Limit new qualifier abbreviations to no more than four letters.

User-Defined Variable Names (continued)

Suffix

Suffix— Identifies the variable units.

1. Separate the suffix from the qualifier with an underscore (_).
2. Not all variables have enough space for a suffix.

Signal names can be up to 15 characters in length. Checkpoint and alias names can be up to 100 characters in length. Keep these names short—diagnostic screens may not be able to display all the characters in long names.

Identify sub-buses with a qualifier. These labels do not have prefixes or suffixes.

Prefixes and Suffixes

The following table lists recommended prefixes and suffixes for PLUS+1 variables.

Prefixes and Suffixes				
Type	**Pin Name**	**Prefix**	**Suffix**	**Example**
DIGITAL_INPUTS				
	DIGIN	B	——	B_SetDefAll
	PINSTATUS	S	——	
ANALOG_INPUTS				
	ANIN	AD	——	
	VOLT	V	mV	V_PotSnsr_mV
	OHM	O	ohm	
	PINSTATUS	S	——	
MULTIFUNC_INPUTS				
	DIGIN	B	——	
	ANIN	AD	——	
	VOLT	V	mV	
	FREQ	F	Hz	
	PINSTATUS	S	——	
OUTPUT_STATUS				
	ACTPWM	FP	pct2*	
	FEEDBVALUE	FV	pct1†	
	PINSTATUS	FS	——	
	ACTFREQ	FF	Hz	
PARAMETER				
	——	——	Units (where applicable)	HiCal_mV
CHECKPOINT				
	——	CP and variable type	Units, connector, and pin location (where applicable)	CP_B_InRngHiCal__c2p2
EE				
	ALIAS	EE	Units (where applicable)	EE_MidJoyCal_mV
Subbus	——	——	——	Pot1Para

* 10,000 = 100% (100 x 10 x 10)

† 1000 = 100 % (100 x 10)

Qualifier/Port Label Abbreviations

The following table lists qualifier abbreviations for use in PLUS+1 variables. Also use these abbreviations when labeling ports on a function.

Qualifier/Port Label Abbreviations					
Qualifier	**Abbreviation**	**Comment**	**Qualifier**	**Abbreviation**	**Comment**
2 wheel	2Whl		Loop Time	LpTm	
4 wheel	4Whl		Low	Lo	
Accelerate	Acel		Maximum	Max	
Alarm	Alrm		Middle, Mid	Mid	
All	All		Minimum	Min	
Angle	Ang		Negative	Neg	
Average	Avg		Neutral	Neut	
Buzzer	Buzz		Not	Not	
Bypass	Bpas		Number	Nmbr	
Calibrate	Cal		Operator	Op	
Clockwise	Cw	CW*	Output	Out	
Command	Cmd		Pack	Pck	
Control, Controller	Ctrl		Parameters	Para	
Coordinate	Cord		Pass	Pass	
Counterclockwise	Ccw	CCW*	Passive	Pasv	
Deadband	Dbnd		Point	Pt	
Decelerate	Dcel		Positive	Pos	
Decrease, Decrement	Dec		Potentiometer	Pot	
Default	Def		Presence	Prs	
Diameter	Dia		Propel	Prpl	
Digital	Dig		Pulse	Puls	
Direction	Dir		Pulse pickup unit	Ppu	PPU*
Displacement	Disp		Pulse width mod. n	Pwm	PWM*
Done	Done		Range	Rnge	
Down	Dn		Read	Rd	
Drive	Drv		Reset	Rst	
Enable	Enbl		Reverse	Rvs	
Equal	Eq		Revolution	Rev	
Error	Err		Right	R	
Fail	Fail		Right front	Rf	RF*
Fault	Flt		Right rear	Rr	RR*
Feedback	Fdbk		Sample	Smpl	
Forward	Fwd		Select	Slct	
Found	Fnd		Sensor	Snsr	
Four wheel	4Whl		Set	Set	
Frequency	Freq		Setpoint	Stpt	
Ground	Gnd		Soft	Sft	
Guard	Grd		Solenoid	Sol	
Handle	Hndl		Speed	Spd	
High	Hi		Start	Strt	
Hold	Hld		Status	Stat	
Horn	Hrn		Steer, Steering	Str	
Hysteresis	Hyst		Stop	Stop	
Increase, Increment	Inc		Sweep	Swp	
Initialize	Init		Switch	Sw	
Input*	In		Time	Tm	
Left	L		Value	Val	
Left front	Lf	LF*	Wheel	Whl	
Left rear	Lr	LR*	Width	Wdth	
Length	Lgth		Work	Wrk	
Light Emitting Diode	LED		Write	Wr	
Loop	Lp				

*Use all caps when labeling ports on a function.

Suffix Abbreviations

The following table lists suffix unit abbreviations. Use these abbreviations when your variable has a suffix.

Unit Abbreviations			
Item	**Abbreviation**	**Item**	**Abbreviation**
100 = 100 %	pct	kilometers per hour	kph
1000 = 100 %	pct1*	meter	m
10,000 = 100 %	pct2*	miles per hour	mph
ampere	A	millimeter	mm
centimeter	cm	minute (time)	min
connector	c	ohm	ohm
degree	Deg	pascal	Pa
degree Celsius	C	pin	p
degree Fahrenheit	F	pounds per square inch	psi
foot	ft	radian	rad
hertz	Hz	revolutions per minute	rpm
hour	h	second (time)	s
inch	in	volt	V
kilogram	kg	watt	W

* 1000 = 100 % (100 x 10)
† 10,000 = 100% (100 x 10 x 10)

General

This section contains guidelines for mapping discrete and status signals. These signals indicate digital states and directions.

Discrete Signals

Discrete Signals				
Value	**Data Type**	**Value**	**Meaning**	**Comments**
Digital input	BOOL	0	Low/Off	Direct mapping from BOOL type
		1	High/On	
Digital output	BOOL	0	Low/Off	Direct mapping from BOOL type
		1	High/On	
Direction	S16	0	Neutral	
		+1	Up, right, forward, extend	
		-1	Down, left, reverse, retract	
		2 – 32767	Unassigned	
		-32768 – -3		

Status Signals

These signals indicate status values.

State Signals				
Value	**Data Type**	**Value**	**Meaning**	**Comments**
Calibrate	S8	0	Auto calibration disabled	Normal operation
		1	Auto calibration enabled	
		2	Apply default values	
		3	Force to uncalibrated values	Maximum and minimum values are both 0
Fault	S16	0	No fault	
		1	Input value too low	
		2	Input value too high	
		3	Input value at 0	
		4	Input value at maximum	
Status	U16	0	No status	
		1	Not calibrated	
		2	In calibration	
		3	Corrupt parameters	
		4–100	Reserved for system-wide status	
		101–65535	Status specific to the application	
Errors	U16	0	No error	
		1	Value too low	
		2	Value too high	
		3	Value at zero	
		4	Value at maximum	
		5–100	Reserved for system-wide errors	
		101–65535	Errors specific to the application	

SAUER DANFOSS

Our Products

Hydrostatic transmissions

Hydraulic power steering

Electro-hydraulic power steering

Electric power steering

Closed and open circuit axial piston pumps and motors

Gear pumps and motors

Bent axis motors

Orbital motors

Transit mixer drives

Planetary compact gears

Proportional valves

Directional spool valves

Cartridge valves

Hydraulic integrated circuits

Hydrostatic transaxles

Integrated systems

Fan drive systems

Electrohydraulics

Microcontrollers and software

Electric motors and inverters

Joystick and control handles

Displays

Sensors

Sauer-Danfoss Hydraulic Power Systems - Market Leaders Worldwide

Sauer-Danfoss is a comprehensive supplier providing complete systems to the global mobile market.

Sauer-Danfoss serves markets such as agriculture, construction, road building, material handling, municipal, forestry, turf care, and many others.

We offer our customers optimum solutions for their needs and develop new products and systems in close cooperation and partnership with them.

Sauer-Danfoss specializes in integrating a full range of system components to provide vehicle designers with the most advanced total system design.

Sauer-Danfoss provides comprehensive worldwide service for its products through an extensive network of Authorized Service Centers strategically located in all parts of the world.

Local address:

Sauer-Danfoss (US) Company
3500 Annapolis Lane North
Minneapolis, NM 55447, USA
Phone: +1 763 509 2084
Fax: +1 763 559 0108

Sauer-Danfoss ApS
DK-6430 Nordborg, Denmark
Phone: +45 7488 4444
Fax: +45 7488 4400

Sauer-Danfoss GmbH & Co. OHG
Postfach 2460, D-24531 Neumünster
Krokamp 35, D-24539 Neumünster
Phone: +49 4321 871 0
Fax: +49 4321 871 284

Sauer-Danfoss-Daikin LTD
Sannomiya Grad Bldg. 8F
2-2-21 Isogami-dori, Chuo-ku
Kobe, Hyogo 651-0086, Japan
Phone: +81 78 231 5001
Fax: +81 78 231 5004

10106114 • Rev. B • Jan 2007

www.sauer-danfoss.com