Document Number: MC33816SIMUG
Rev. 1.0, 6/2014

MC33816 PSC Simulator

A Software Tool for Simulating the MC33816 Device

Contents

1 Overview

The PSC (Programmable Solenoid Controller) Simulator is an integrated design/simulation environment for the firmware required by a family of Freescale Programmable Solenoid Controllers (PSC). In this chapter, the main functions of the PSC simulator are listed, along with a suggested development cycle.

1.1 Development Cycle

A software developer follows a general development process:

- Begin with the specification for new/updated software
- Create a new project and classify it, according to application, hardware used, etc
- Implement the specification in firmware suited to the smart driver
- Have the PSC simulator compile the code into machine code
- Specify simulation commands and feedback
- Simulate the firmware behavior
- Correct errors (debug)
- Release the compiled code for the inclusion into system software

Each of the steps, except the first one, is briefly described hereinafter and in a dedicated chapter.

1.1.1 Specification

This step is entirely assigned to the user.

1.1.2 Creation and Classification

The first step is to create a new project and to classify it. The new project can start either from scratch (a new application) or from a copy of an existing project (a new version of an existing project). The PSC is a very flexible device, so it can adapt to a number of different applications where solenoids must be driven. Even considering a single application, a number of different firmware projects are designed to address either bug fixes or new specifications. Therefore, classification of the project is mandatory. The PSC simulator uses a side tool, called "Version Manager", which is used during this step to create a new project. The classification of the project is first specified upon creation, but it can be modified later.

1.1.3 Implementation

From the version manager, it is possible to launch the PSC Simulator together with any project that has been previously classified. When the PSC simulator has started, it is not possible to change the project to which it is referring (loading a new project). To select a new project, the PSC simulator must be launched again from the version manager, selecting the desired project. After launching the PSC simulator, it is possible to enter the firmware needed by the PSC, according to the specification.

1.1.4 Compile

From the PSC simulator it is possible to compile the code designed by the user into machine code. An internal compiler is available, but it is recommended to link the simulator with a certified compiler. This link can be set up using the version manager and it is global for every project using the same PSC. The simulator has a log window where the compiler output is shown. The first level of debug is performed during this step, to obtain a compilation without errors.

1.1.5 Simulation Setup

The second level of debug is simulation. The firmware for the PSC is almost never enough to perform a simulation: the PSC in the real application interacts with other devices, like the loads (through the power stage) and an external controller. To achieve a meaningful simulation it is necessary to emulate, at least roughly, the devices that interact with the PSC. The simulator supports emulation of:

- Time base stimulus on digital pin of the device (useful to emulate commands received from an external controller)
- Feedback from a power stage, commonly used for diagnosis
- Loads

1.1.6 Simulation

Once the simulation setup is complete, a subset of the signals of the device (input, output and internal) can be selected and the simulation can be run. The PSC simulator is able to show a time diagram of the selected signals, as well as the current/voltage on the specified loads. It is possible to simulate for a fixed time, to insert breakpoints or execute code step by step.

1.1.7 Debug

After the simulation is complete, the user can compare the result with the starting specification and modify the firmware and the simulation setup until the desired result is achieved.

1.1.8 Release

The files used by the PSC simulator can be saved in different formats; one of these formats (hexadecimal) has a syntax which is compatible with ANSI C, so it is ready to be included in the software of the system into which the PSC is used.

2 Creation and Classification

Using the "Version Manager" tool, the PSC firmware is categorized for easier retrieval and modifications, according to the following five parameters:

- The PSC used (e.g. MC33816)
- The Hardware platform (ECU) where the configuration is used (e.g. FreescaleEVB)
- The application into which the configuration is integrated (e.g. DieselInjector)
- The ID of the device on the Hardware platform (e.g. ID0). This is useful in case of hardware platforms with more than one PSC onboard
- The Version of the configuration (e.g. 1.0)

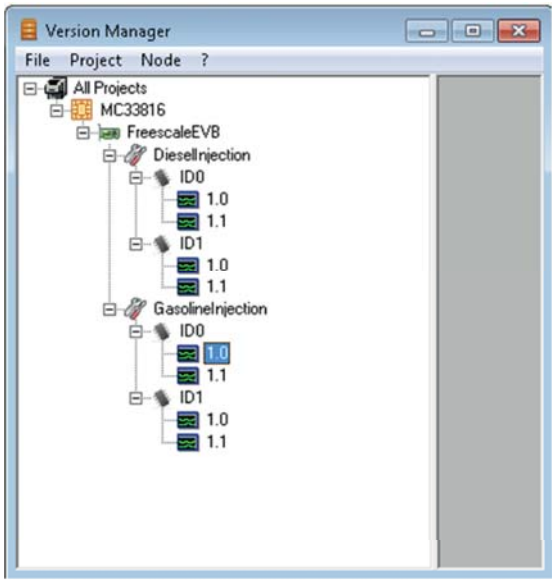


Figure 1. Classification Example

Each project needs all the five classification parameters. All of the parameters are assigned at the moment of its creation. They can be modified after the creation. The Version Manager cannot classify two projects that have exactly the same five classification parameters. It is not possible to add a project to the classification (either new or already existing) for which the five classification parameters have the same values as one of the project already classified.

2.1 New Project

To create a new project from the very beginning, select "Project->Add New" from the main menu. The following form prompts the user for the five classification parameters and the directory where the project is saved.

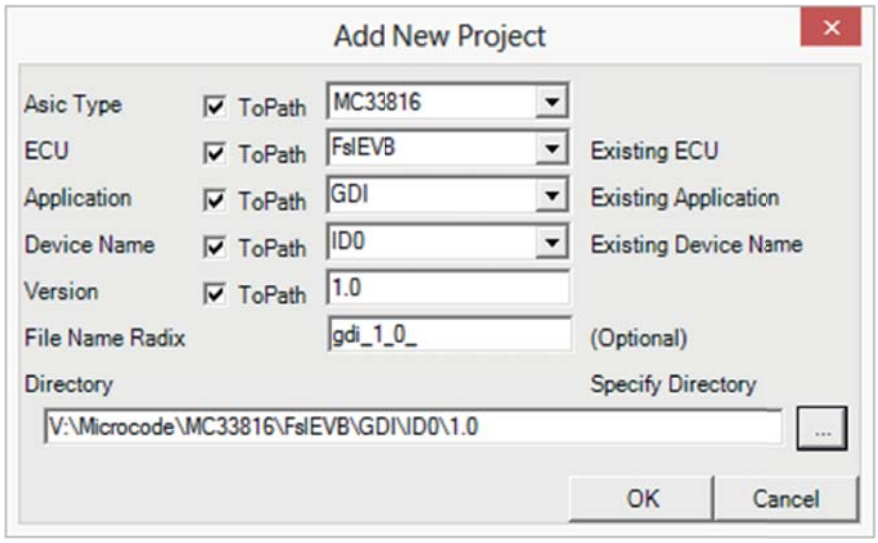


Figure 2. Creation of a New Project

Since it is a new project, a template is copied to the specified directory. This template contains all the files necessary for the firmware of the selected PSC; the template contains no code (code files are empty) and all the PSC registers are set to their default values. The template contains no simulation file (models, stimuli, etc….). The File Name Radix prepends the names of all of the microcode and register files that are part of the project.

2.2 Copy Project

It is far more common to create a new project as a copy of an existing project than creating it from scratch. It could be a new version containing bug fixes or additional specifications, or it is a new application that is not too different from an existing one (e.g. creating a project for a Diesel application starting from a Gasoline project).

To copy a project:

- First select the source project from the classification tree
- Select "Copy" from the context menu or "Node->Copy" from the main menu
- The following form prompts the user for the five classification parameters of the copied project and the directory where the copied project is saved

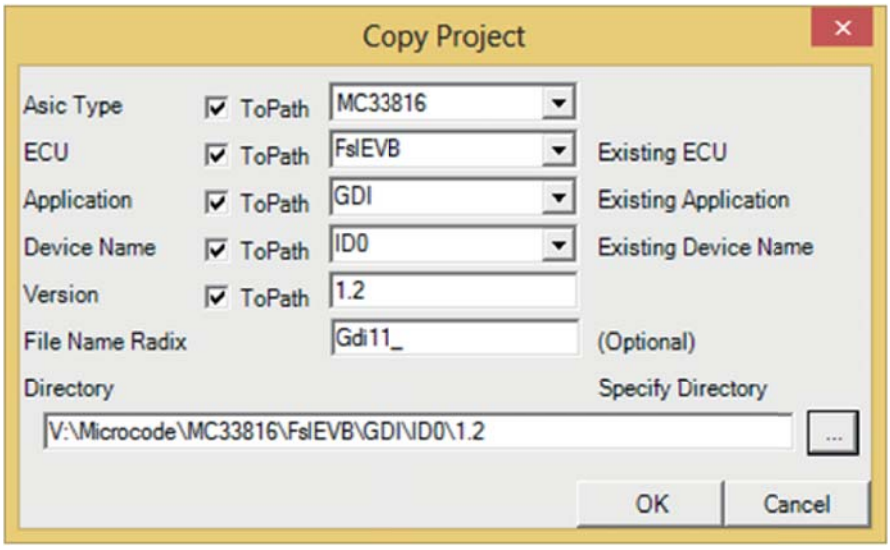


Figure 3. Copy of a Project

2.3 Add Existing Project

It is possible to classify an already existing project. This operation is useful in the case of projects developed by other designers, which need to be used as reference or as a starting point for new designs.

To classify an existing project select "Project->Add Existing". A file dialog opens, prompting for selecting the project main file, "project.xml". The project file already contains the five classification parameters, which are used to classify the existing project. The operation fails if the PSC used for the existing project is not already included in the "Version Manager".

3 Implementation

After the project has been created, it can be used by selecting it from the Version Manager and:

- Double clicking on the project
- Right-clicking on the project and selecting "Run" from the context menu
- Selecting "Node->Run" from the main menu

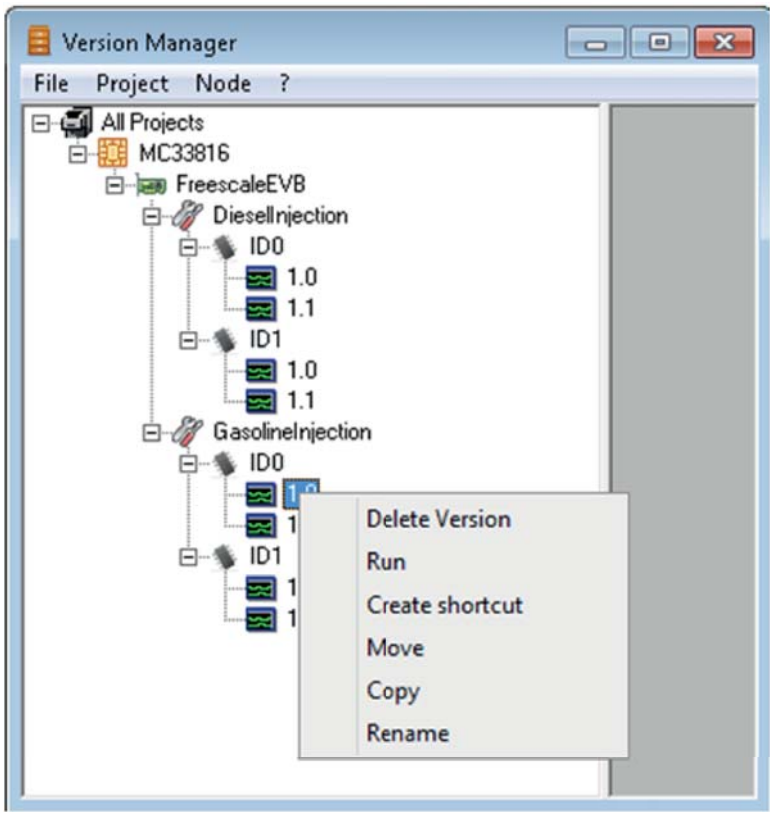


Figure 4. Start a Project

When the project is run, the PSC simulator starts loading the following:

- The PSC model appropriate to the project selected (MC33816 as seen in Figure 4, Start a Project)
- All the firmware and simulation files relating to the project

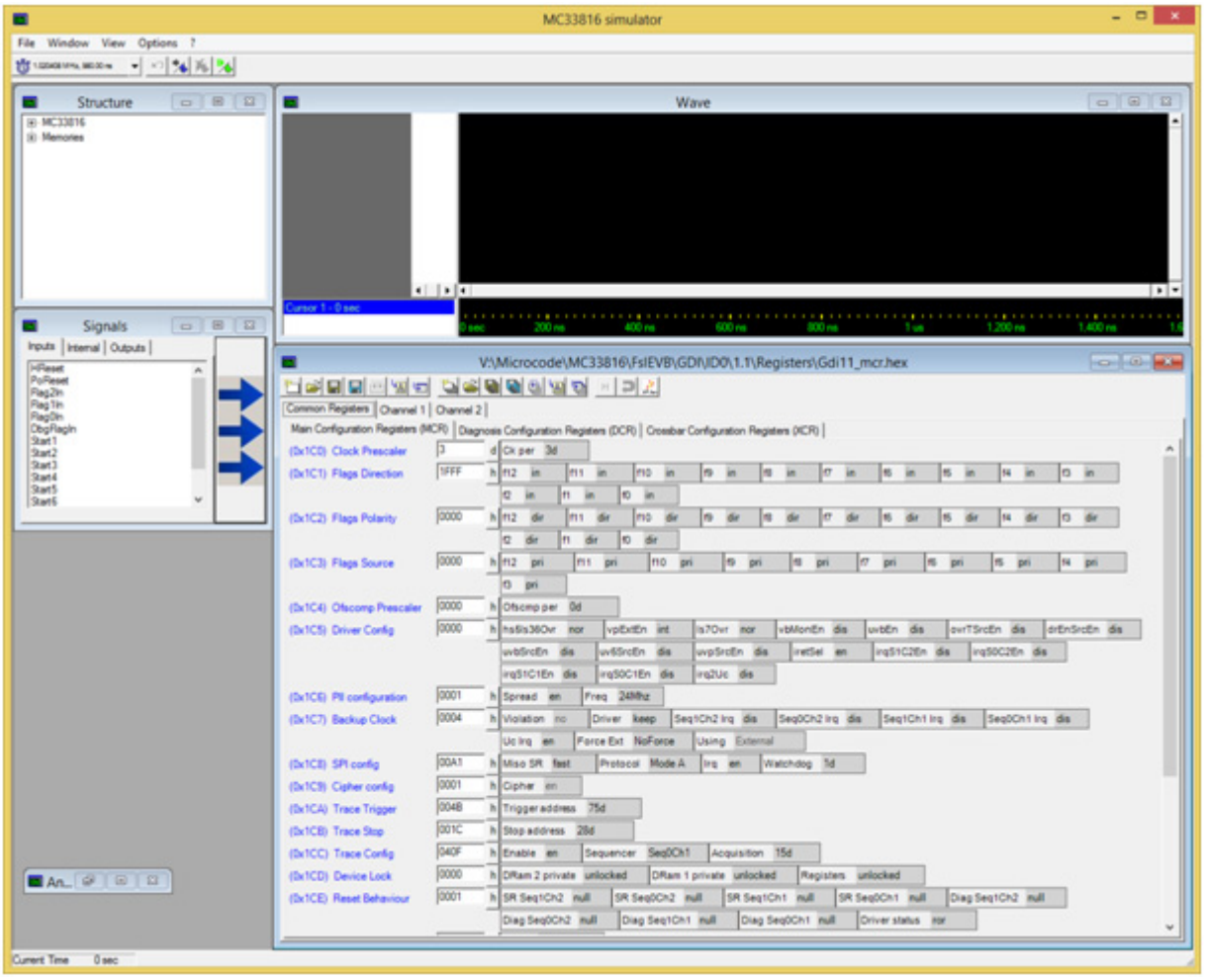


Figure 5. PSC Simulator

The PSC simulator is composed of nine sub-windows. Each window can be opened (or brought to front if already opened) by selecting the appropriate command from the "Windows" main menu (e.g. "Show MicroCode").

The "MicroCode" window contains all the tools used to implement the PSC firmware. The layout of the MicroCode window is loaded from the PSC model, so its appearance is different for different devices. The window contains a variable number of tabs, one for each of the memory areas of the PSC. The simulator supports three types of memory areas:

- Code RAM: these are RAM areas which contain compiled code. For each of these areas a "Code" tab is created
- Data RAM: these are RAM areas which contain application-specific data. For each of these areas a "Data" tab is created
- Registers: these are groups of memory elements that have a fixed function. For each of these areas a "Register" tab is created

By setting the desired values on these tabs, the operational specification can be implemented.

The window is also supplied with a toolbar (buttons described in order) to perform basic file operations on the firmware files. The description also indicates if the buttons are used for implementation (I), compilation (C) or simulation (S).

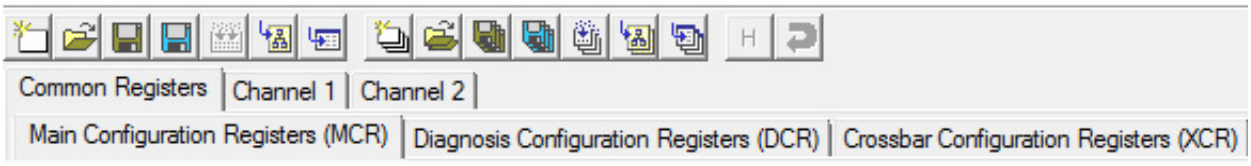


Figure 6. Microcode Window Toolbar

- (I) New: resets the content of the currently selected tab to its reset value
- (I) Open: loads the value of the items in the currently selected tab (code, data or registers) from a selected file

- (I) Save: saves the value of the items in the currently selected tab (code, data or registers) to a file
- (I) Save As: saves the value of the items in the currently selected tab (code, data or registers) to a new file
- (C) Compile: enabled only if the selected tab refers to Code RAM. It compiles the code contained in the currently selected tab
- (S) Load to model: loads the value of the items in the currently selected tab (code, data or registers) to the simulation. Normally this operation is executed automatically once at simulation time 0
- (S) Load from model: all the items in the currently selected tab (code, data or registers) are copied from the simulation to the tab
- (I) New All: resets the content of all the tabs to their default value
- (I) Open All: loads the values of the items in all the tabs (be it code, data or registers) from a selected file (one for each tab)
- (I) Save All: saves the values of the items in all the tabs (be it code, data or registers) to a file (one for each tab)
- (I) Save All As: saves the value of the items in all the tabs (be it code, data or registers) to a new file (one for each tab)
- (C) Compile All: compiles the code contained in the Code RAM tabs
- (S) Load All to model: loads the value of the items in all the tabs (code, data or registers) to the simulation. Normally this operation is executed automatically once at simulation time 0
- (S) Load All from model: copies data from the simulation to all of the tabs (code, data and registers)
- (I) Syntax Help: enabled only if the selected tab refers to Code RAM. It enables syntax help for code entry
- (I) Undo: enabled only if the selected tab refers to Code RAM. Undo the last text entry operation

3.1 Code Entry

For each of the Code RAMs in the PSC, a code tab is available in the MicroCode window.

Figure 7. Code Entry

The tab allows code entry, with basic features (cut, paste, find, etc.) plus the following advanced features:

- Custom Syntax: the simulator is supplied with a syntax file, specific to the PSC, when it is launched. This syntax file, which describes the instruction set used by the PSC, is used for:
  - Syntax highlighting: instructions are highlighted in blue, comments in green and numbers in violet
  - Content-aware auto-completion: advanced auto-completion feature that proposes values for instructions and parameters by analyzing the code
- Column editing: by pressing the Alt key, it is possible to select multiple columns, to cut, paste and edit in column mode
- Row Numbering: for each row, the row text number (basic feature) and the code line number (advanced feature) are shown. The code line number is the address of the specific instruction in the Code RAM

In the code, it is possible to use aliases to refer to a Data RAM cell, instead of the address. For additional information on the aliases, see 3.3 Register Entry on page 12.

3.2 Data Entry

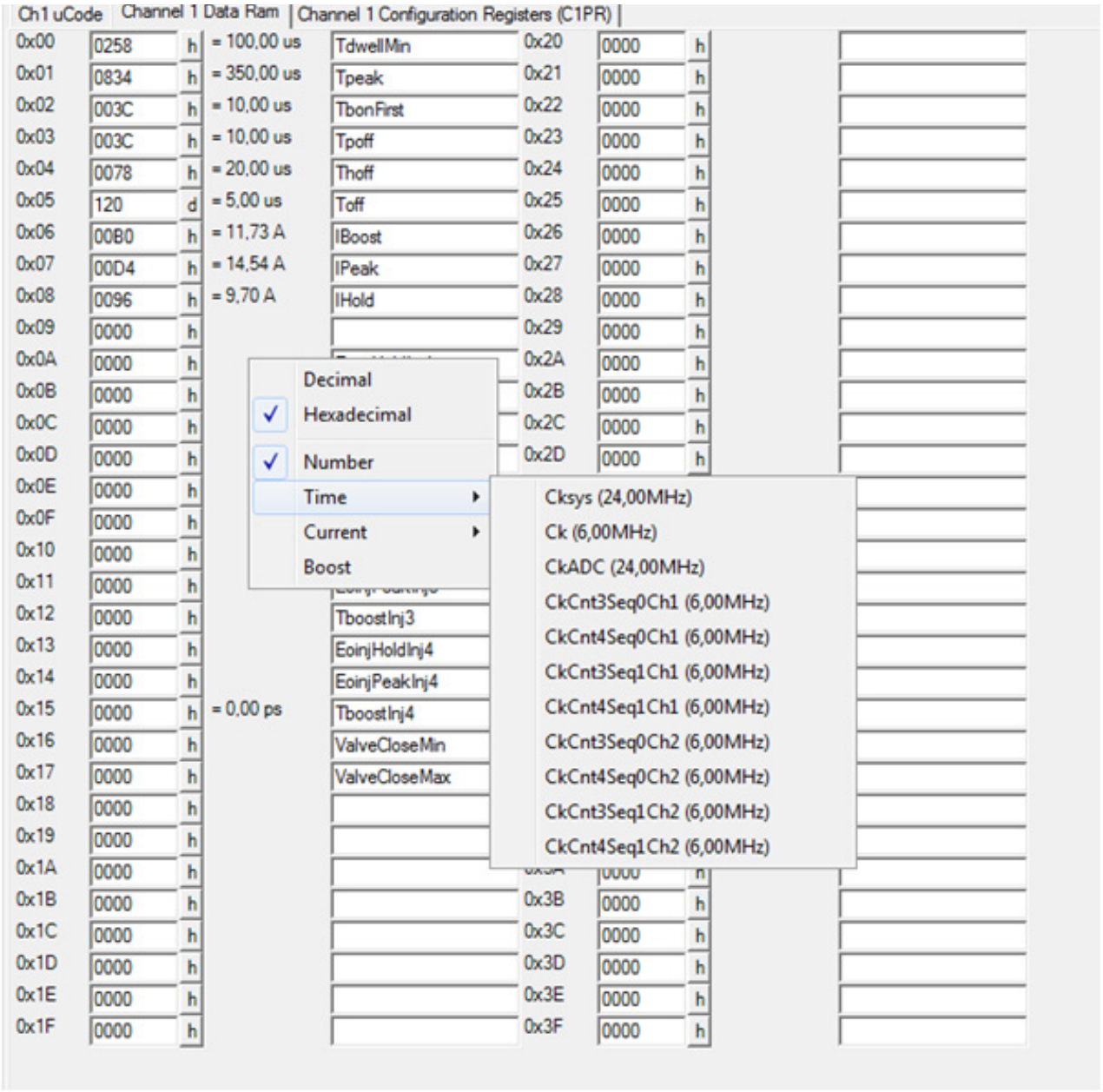


Figure 8. Data Entry

The values contained in the Data RAM are application-dependent. In this tab there are a number of entries, one for each element of the Data RAM. For each of these entries it is possible to specify:

- Value: the number that is contained in the cell of the Data RAM
- Decoding: a formula can be specified, to translate the number in the cell into its corresponding physical value. The formula can be specified using the context menu:
  - Number: no formula, since the cell contains a pure number
  - Time: the number in the cell contains a time value. Specify from the sub-menu which of the clocks of the device must be used for the conversion
  - Current: the number in the cell contains a current value. Specify from the sub-menu which current measurement block and which gain must be used for the conversion

- Boost: the number in the cell contains a voltage value
- Alias: this is an alternate name of the parameter contained in the Data RAM cell. It can be used in the code when reading/writing this cell in place of its address

3.3 Register Entry

Registers are memory elements of the device that have a fixed function. Registers are commonly grouped by function in the PSC. For each group of registers in the PSC, a register tab is available in the MicroCode window.

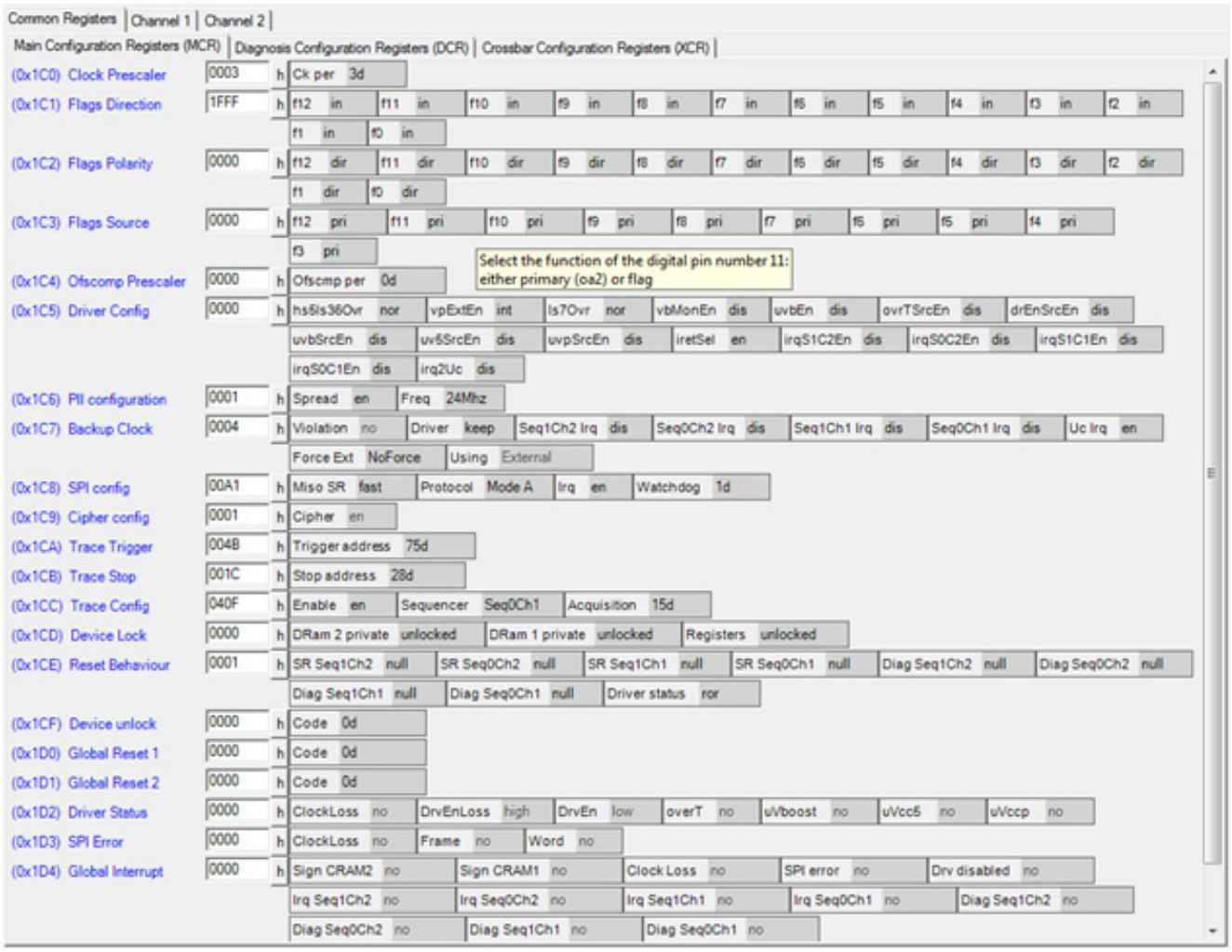


Figure 9. Register Entry

For each register a horizontal row is present that contains:

- Name and address of the register
- Register value: it is possible to read and enter the complete (all bitfields) value of the register. When the value is changed, the bitfields are updated, and vice versa
- Bitfields: all the bitfields are presented to the user, in a human-readable interface. Each bitfield has a description of the field in its tooltip. When a bitfield value is changed, the register value is updated, and vice versa. There are three types of bitfields:
  - Boolean: to change (toggle) its value, double-click the bitfield
  - Numeric: when clicked, it is possible to enter a value. The simulator checks to see that the number is accepted by the model
  - Enumerative: when clicked, a drop-down menu allows the user to choose between the possible values for the field

4 Compile

When the implementation is complete, the assembler code must be compiled to be translated into machine code. For this operation, both the Microcode window (already used for the implementation) and the Log window are necessary.

To compile the code for one of the channels, select the tab containing the desired code from the Microcode window and press the "Compile" button on the Microcode window toolbar. To compile all the code, press the "Compile all" button in the Microcode window toolbar (no need to select a specific tab).

The result of the compilation is listed in the Log Window.

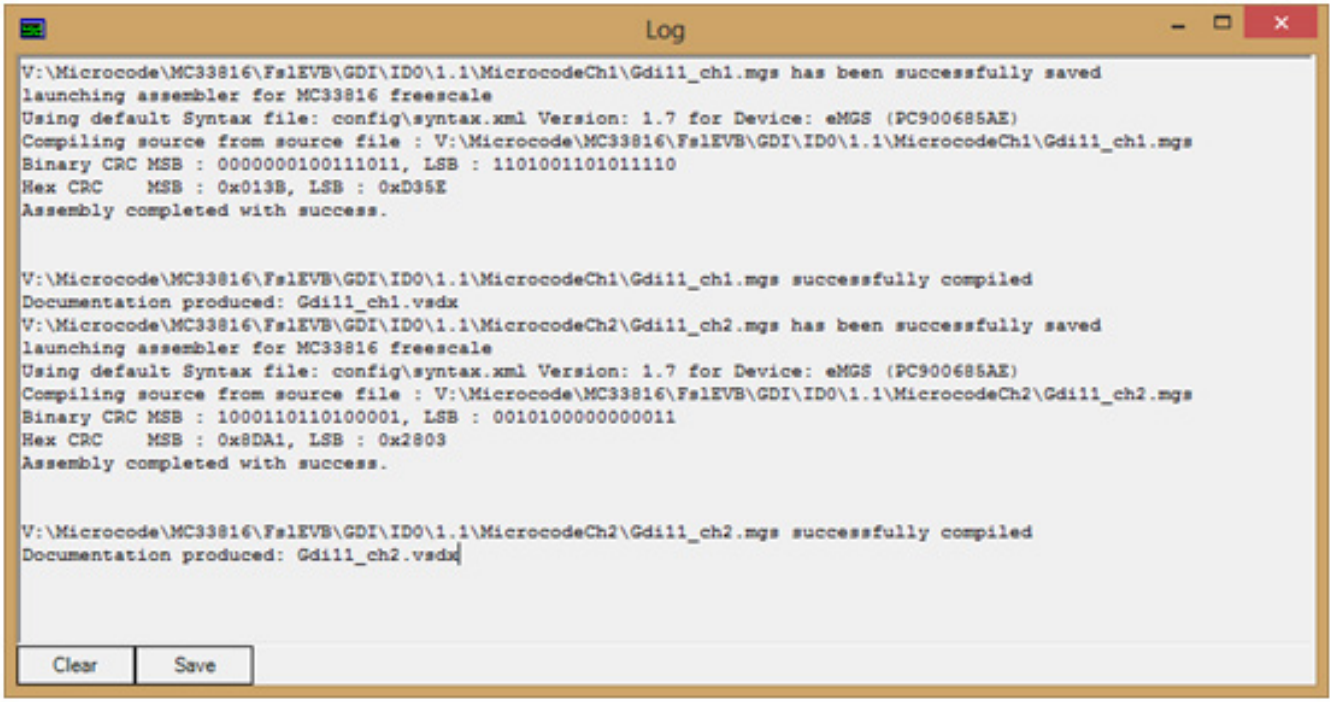


Figure 10. Successful Compilation

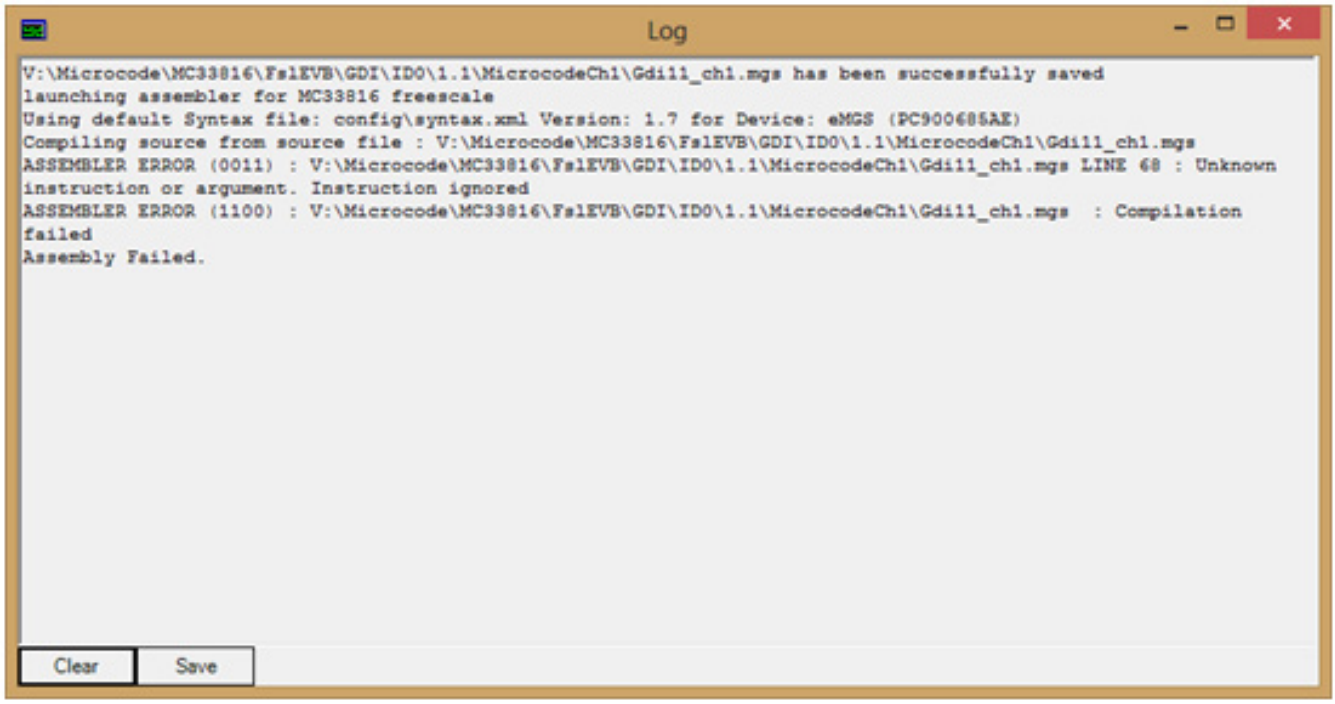


Figure 11. Failed Compilation

By default, the PSC simulator uses an internal compiler, which is not certified. It is possible to use an external compiler by "linking" such compiler to a PSC in the version manager. Once this link is established, all the projects using that PSC uses the same external compiler. If no such link exists, the internal compiler is used.

To link a PSC to an external compiler:

- Launch the version manager
- Right-click on the desired PSC ( ) and select the "set paths" entry
- The "PSC program paths" form is displayed. The content of this form depends on the PSC selected. If the PSC supports an external compiler, it is listed in this form. Supply the path to the compiler executable and select "OK". The external compiler may use a key to cipher the compiled code. In this case, the path to the cipher key file can be specified in the same form
- To use this newly created link, you have to launch the PSC simulator after the link is created. This link affects already existing projects

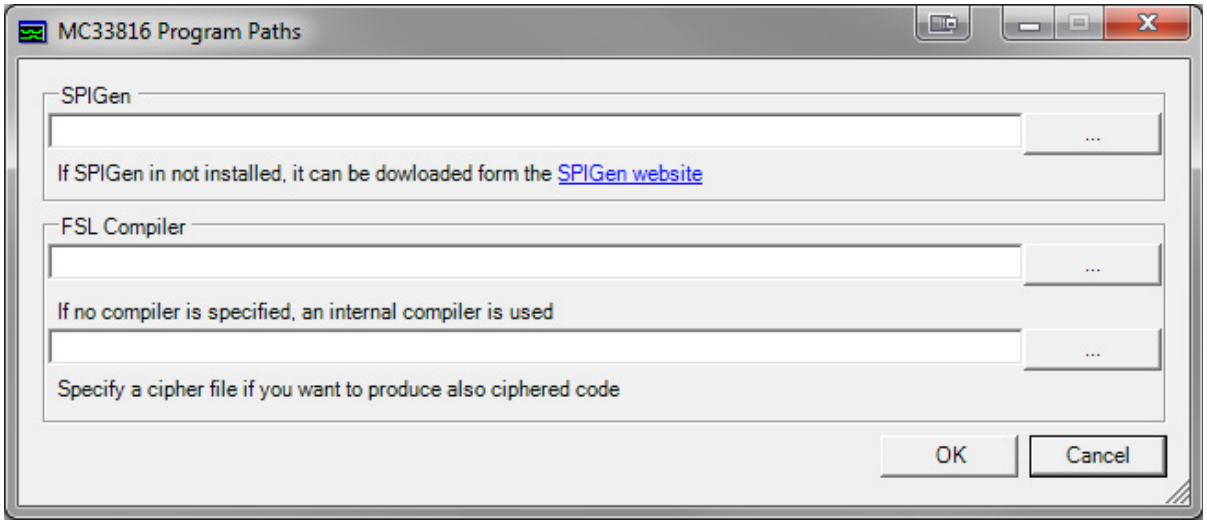


Figure 12. PSC Program Paths

The output in the Log window either confirms a successful compilation or lists the errors found. The internal compiler output includes:

- A description of the error
- Text line where the error is located (e.g. the 12th line of the file)
- Code line where the error is located (e.g. the code line, counting from zero and counting only code lines, not comments, or defines)

If using an external compiler, please refer to its documentation for details about its output.

5 Simulation Setup

The first level of debug of the code is compilation. When compilation completes successfully, all syntax errors have been removed.

The second level of debug is simulation. Before starting a simulation, some setup steps must be performed, according to the type of firmware designed:

- Stimuli: the firmware may need commands from a higher level controller. For example if the application is fuel injection, it is very likely the request to inject fuel is received from an external micro-controller. This step may be skipped if the application can be tested without higher level commands
- Loads: it is very likely the PSC controls part of its load in a closed control loop (e.g. driving of injectors by controlling the current in the load). In this case no simulation can be achieved if the device receives no "control" feedback from the driven load (e.g. the current flowing in the load read back using a shunt resistor). This step may be skipped if the PSC is controlling everything in open loop (e.g. some pumps)
- Diagnosis: the PSC may have some diagnosis feedback, used to detect faults on the loads it drives. To prevent false diagnosis, and to test the diagnosis routines, it must be specified which kind of signals are supplied to the diagnosis feedback of the device. This step may be skipped if the firmware does not implement diagnosis routines (very common in the first drafts of firmware)

The following picture shows an example of the PSC (MC33816) in which the pins concerning the three steps of the simulation setup are highlighted.

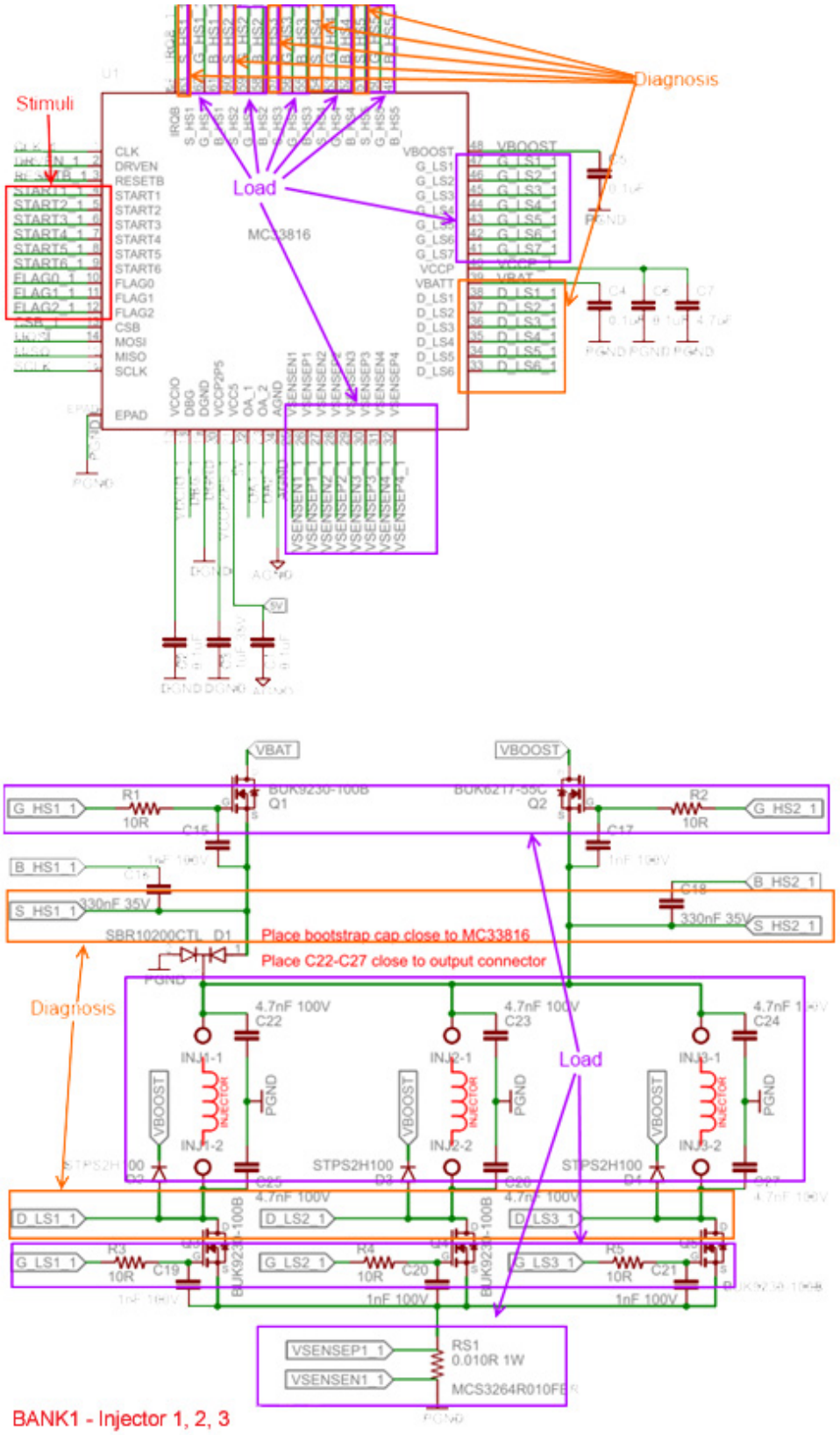


Figure 13. Simulation Setup

5.1 Stimuli

Using the Stimulus window, it is possible to specify some stimuli on the digital pins of the device. Those stimuli can be based only on time, not on the outputs of the device, so they are not optimized to simulate feedback. The main purpose of these stimuli is to emulate an external device communicating with the PSC.

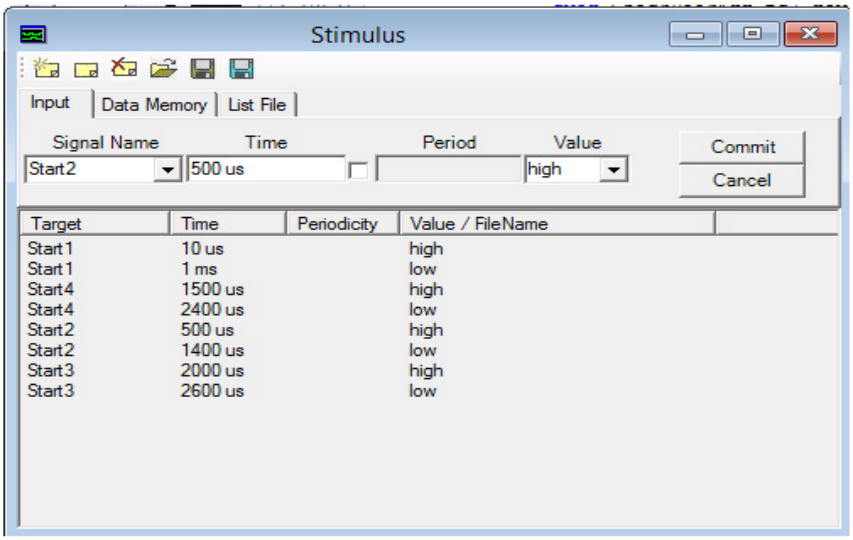


Figure 14. Stimulus Window

Operations:

- Create a new stimulus: click on the first button of the toolbar (Add) to enter edit mode; enter the details of the stimulus in the top part of the form. Click the commit button to save the stimulus to the list, or cancel to close editing mode
- Modify a stimulus: select the desired stimulus from the stimuli list (middle low part of the window), this loads the stimulus details in the top part of the form, then click the second button of the toolbar (Edit) to enter edit mode. Modify the stimulus details in the top part of the form. Click the commit button to update the stimulus to the list, or cancel to discard the changes and close editing mode
- Delete a stimulus: select the desired entry (or entries) from the stimuli list (middle low part of the window) and press the "Delete" button (third button of the toolbar)

There are two types of stimuli, selectable from the tab control just below the toolbar:

- Input Stimuli: these stimuli are applied to single pin/input of the device
  - Signal Name: the name of the signal/pin to which apply the stimulus
  - Time: the time at which the stimulus must be applied. In case of periodic stimulus, it is the time when the stimulus is first applied
  - Period Check: if unchecked, the stimulus is applied only one time. If checked, the stimulus is applied when the time is ["Time" + n * "Period"], where n is an integer and non-negative
  - Period: used only if the "Period check" is set. If the stimulus is periodic, it specifies the period of stimulus
  - Value: the value applied to the selected signal at the specified time

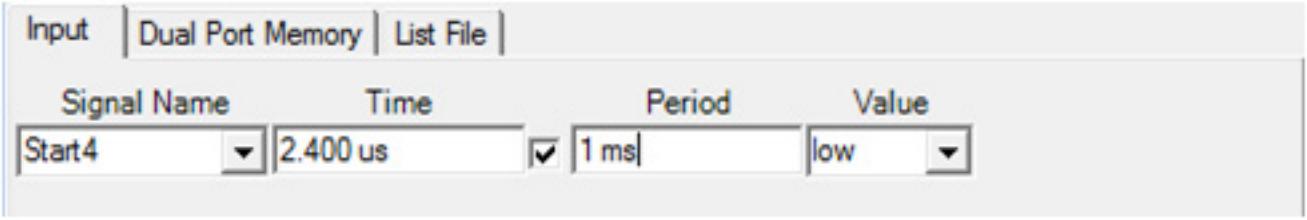


Figure 15. Signal Stimulus

- Memory Stimuli: these stimuli are applied to the Data RAM of the ASIC:
  - Channel: select which Data RAM of the device is the target of the stimulus
  - Time: the time at which the stimulus must be applied. In case of periodic stimulus, it is the time when the stimulus is first applied

- Period Check: if unchecked, the stimulus is applied only one time. If checked, the stimulus is applied when the time is ["Time" + n * "Period"], where n is an integer and non-negative
- Period: used only if the "Period check" is set. If the stimulus is periodic, it specifies the period of stimulus
- File Name: a file containing the list of values to be written into the selected Data RAM at the specified time

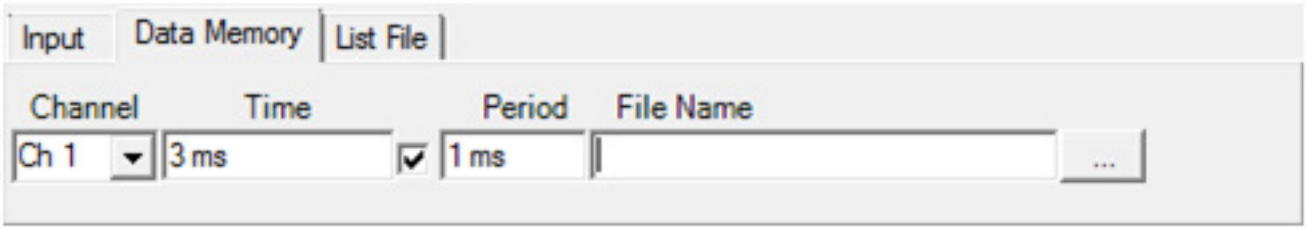


Figure 16. Memory Stimulus

5.2 Loads

The loads can be specified using the Actuator window. The actuator window has a variable number of tabs, one for each load that has been defined.

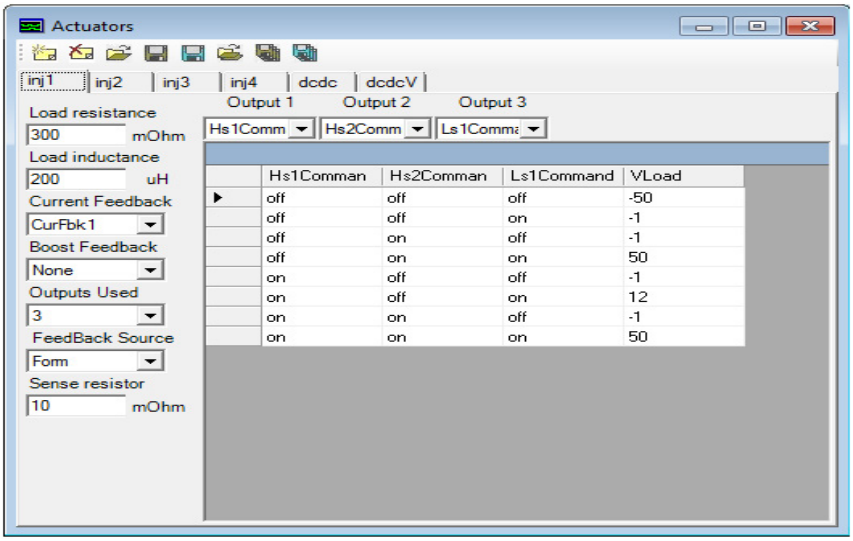


Figure 17. Actuator Window With Ideal L-R Load

The toolbar of the window is used to create/import/delete/save the definition of each actuator. Each actuator is saved into a separate file. In order, the icons on the toolbar allow the user to:

- Create a new load: this command opens a dialog which asks the user to select a name for the new load and select a file to where save its data. If an existing file is selected, the data in the file is imported in the tab
- Delete a load: this command removes the load whose tab is selected at the moment
- Open a load: this command imports the data of a load which has been previously created into the currently selected tab. A dialog prompts the user for the actuator file, from where the data is to be imported
- Save a load: this command saves the data of the actuator whose tab is currently selected
- Save a load As: This command saves the data of the actuator whose tab is currently selected into a new file. A dialog prompts the user for the name of the file where the data is saved
- Open all the loads: this command applies the "Open a load" command on all the tabs instead of only on the selected one
- Save all the loads: this command applies the "Save a load" command on all the tabs instead of only on the selected one
- Save all the loads As: this command applies the "Save a load As" command on all the tabs instead of only on the selected one

5.2.1 Ideal Model

The PSC simulator provides an ideal R-L series load as a model of the actuator. This can be used as a basic model for most solenoid loads (injectors, valves). If this model is not adequate, it is possible to define a custom model using the C# language.

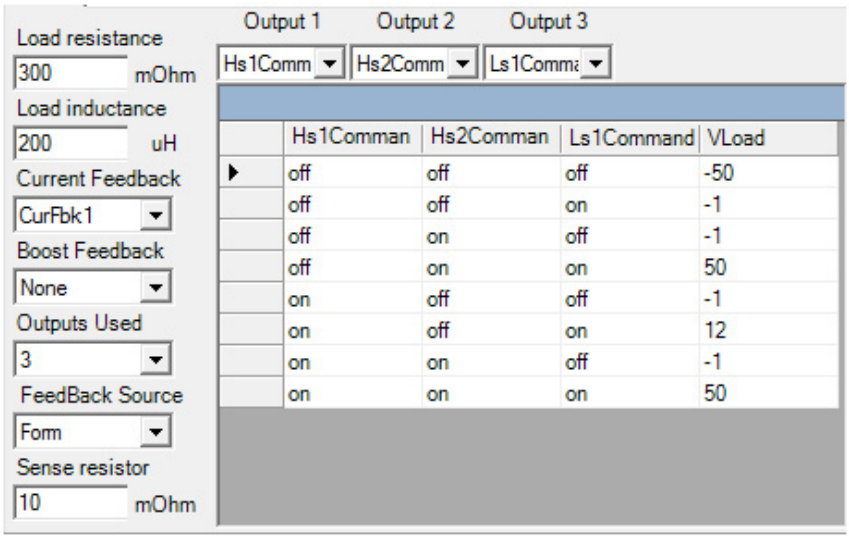


Figure 18. Ideal R-L Model

The ideal model must be supplied with the following parameters:

- Feedback Source: select "Form" to use the ideal R-L load
- Current Feedback: select to which input of the device the feedback for this load is sent to. It can be set to "None" if needed; in that case no feedback is set on the PSC (a load driven in open loop)
- Boost Feedback: set to "None"
- Sense resistor: this is the value of the shunt resistor used to read the current flowing in the load
- Load Resistance: the resistance of the load expressed as mOhm
- Load Inductance: the inductance of the load expressed as uH
- Outputs Used: this control is used to select the number of the outputs of the PSC which affect the driving of this load (e.g. for a GDI/DDI injector, High-side, Boost and Low-side MOSFETs are needed, three outputs in total). When this value is changed to a number, the same number of combo-boxes is displayed in the top part of the tab. From these comboboxes, it must be selected which are the outputs that affect the state of the load (choosing amongst the outputs of the PSC) must be selected. The middle-right section of the tab is occupied by the VLoad table. This table has a row for each possible combination of the states of all the selected outputs (the ones affecting the load) (e.g. eight rows for three outputs). The user must specify the voltage applied over the load for each row, which represents a possible state of the outputs of the device, the voltage applied over the load

Using all the parameters supplied, it is possible to simulate the current in the ideal load (starting from resistance, inductance and the VLoad table). Using the "sense resistor" and "current feedback", the correct feedback is sent back to the PSC.

5.2.2 Custom Model

In some simulations, the ideal model is not acceptable:

- More details are needed: A model of parasitic effects is needed or maybe variable parameters
- A completely different type of load is driven, for example a DC-DC converter

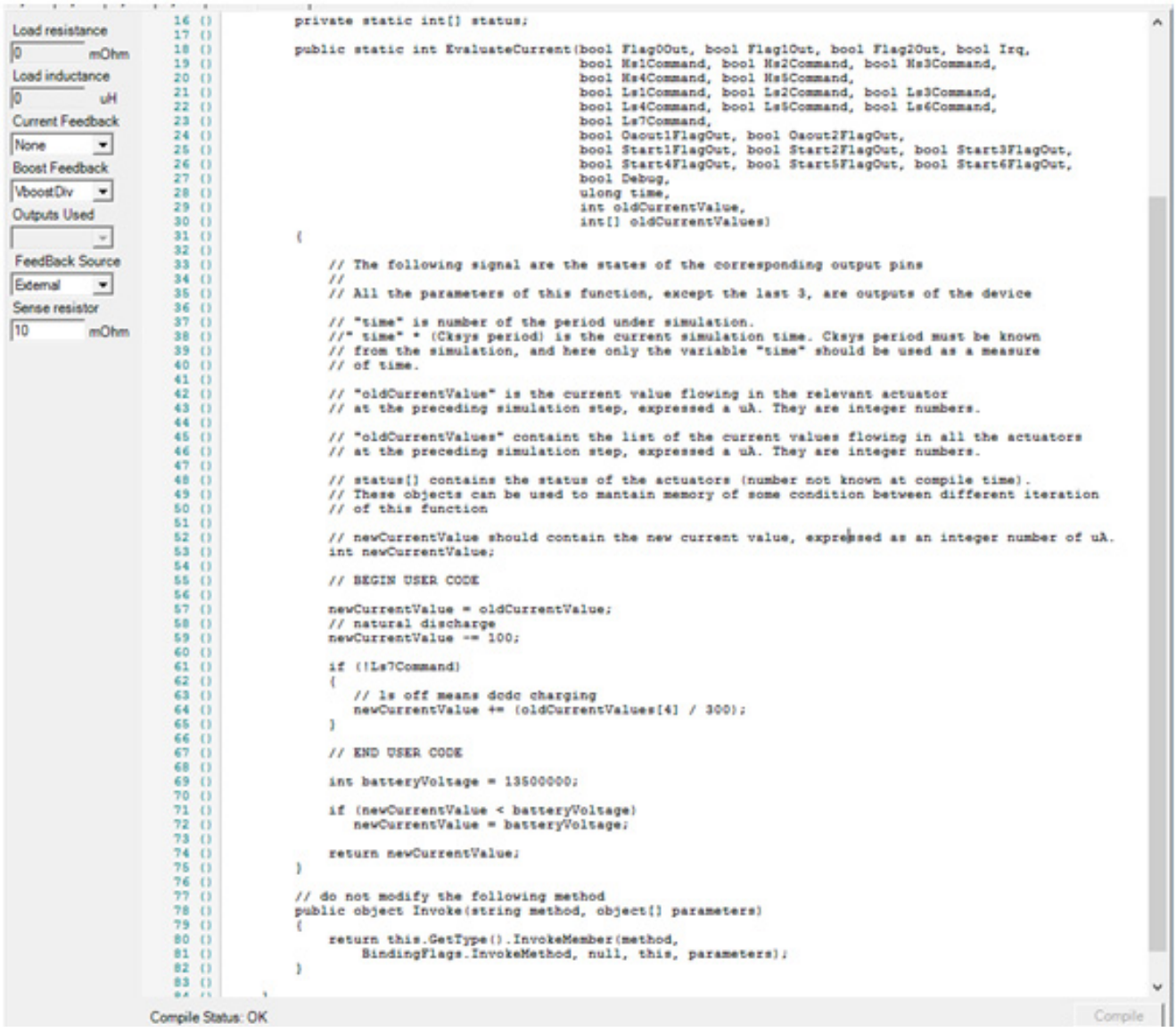
The simulator supports two types of feedbacks for closed loop:

- Current: the current closed loop controls are commonly used to drive solenoid loads. In this case, the important load parameter is a current
- Voltage: the voltage closed loop control is useful to drive some particular loads (e.g. DC-DC converters). In this case, the important load parameter is voltage

Some loads may need multiple closed loops to be driven (e.g. common DC-DC converters have a low-level current closed loop for the current in the inductance, plus a high-level voltage closed loop for the voltage in the capacitor). In this case, multiple "virtual" loads need to be defined, one for each closed loop needed.

When the ideal model is not acceptable, a custom model can be described:

- Feedback Source: select "External" to use a custom model
- Current Feedback or Boost Feedback: select to which input of the device the feedback for this load is sent to. Only one of these controls can be set to a value different from "None". Both can be set to "None" if needed; in that case, no feedback is set on the PSC
- Sense resistor: this is the value of the shunt resistor used to read the current flowing in the load. Used only if a "current feedback" value different from "None" is selected




```
        private static int[] status;

        public static int EvaluateCurrent(bool Flag0Out, bool Flag1Out, bool Flag2Out, bool Irq,
                                          bool Hs1Command, bool Hs2Command, bool Hs3Command,
                                          bool Hs4Command, bool Hs5Command,
                                          bool Ls1Command, bool Ls2Command, bool Ls3Command,
                                          bool Ls4Command, bool Ls5Command, bool Ls6Command,
                                          bool Ls7Command,
                                          bool Oscout1FlagOut, bool Oscout2FlagOut,
                                          bool Start1FlagOut, bool Start2FlagOut, bool Start3FlagOut,
                                          bool Start4FlagOut, bool Start5FlagOut, bool Start6FlagOut,
                                          bool Debug,
                                          ulong time,
                                          int oldCurrentValue,
                                          int[] oldCurrentValues)
        {

            // The following signal are the states of the corresponding output pins
            //
            // All the parameters of this function, except the last 3, are outputs of the device

            // "time" is number of the period under simulation.
            //" time" * (Cksys period) is the current simulation time. Cksys period must be known
            // from the simulation, and here only the variable "time" should be used as a measure
            // of time.

            // "oldCurrentValue" is the current value flowing in the relevant actuator
            // at the preceding simulation step, expressed a uA. They are integer numbers.

            // "oldCurrentValues" containt the list of the current values flowing in all the actuators
            // at the preceding simulation step, expressed a uA. They are integer numbers.

            // status[] contains the status of the actuators (number not known at compile time).
            // These objects can be used to mantain memory of some condition between different iteration
            // of this function

            // newCurrentValue should contain the new current value, expressed as an integer number of uA.
            int newCurrentValue;

            // BEGIN USER CODE

            newCurrentValue = oldCurrentValue;
            // natural discharge
            newCurrentValue -= 100;

            if (!Ls7Command)
            {
               // ls off means dodc charging
               newCurrentValue += (oldCurrentValues[4] / 300);
            }

            // END USER CODE

            int batteryVoltage = 13500000;

            if (newCurrentValue < batteryVoltage)
               newCurrentValue = batteryVoltage;

            return newCurrentValue;
        }

        // do not modify the following method
        public object Invoke(string method, object[] parameters)
        {
            return this.GetType().InvokeMember(method,
                BindingFlags.InvokeMethod, null, this, parameters);
        }
```

Figure 19. Custom Model

The user is presented with a C# file describing a function. The user can modify the body of the function, to describe the load behavior. The function is called at every simulation step (the exact duration depends on the system clock of the PSC used). Starting from:

- The output of the PSC (all the function input parameters except the last three) at that simulation step
- "time": the time (supplied as the number of periods already simulated)
- "oldCurrentValue": the value of the state variable (current or voltage) of this load at the previous simulation step
- "oldCurrentValues": the value of the state variables (current or voltage) of all the loads at the previous simulation step

The function must return the new value of the state variable ("newCurrentValue").

After the custom model is complete, it must be compiled, using the "Compile" button in the lower-right corner of the form. The Compilation result is reflected both in the form (lower-left) and in the Log window. The Simulation cannot start if the custom models are not compiled successfully.

5.3 Diagnosis

Most of the applications perform diagnosis by reading currents/voltages from the rest of the system and comparing them with the state of the device (e.g. reading the Drain-Source voltage of a MOSFET and expecting it to be low when the MOSFET is turned on). The "feedback" is normally tightly related to the outputs of the device itself. The feedback values can be specified in the "feedback" window. This window is composed of a toolbar and a variable number of tabs, one for each input of the device.

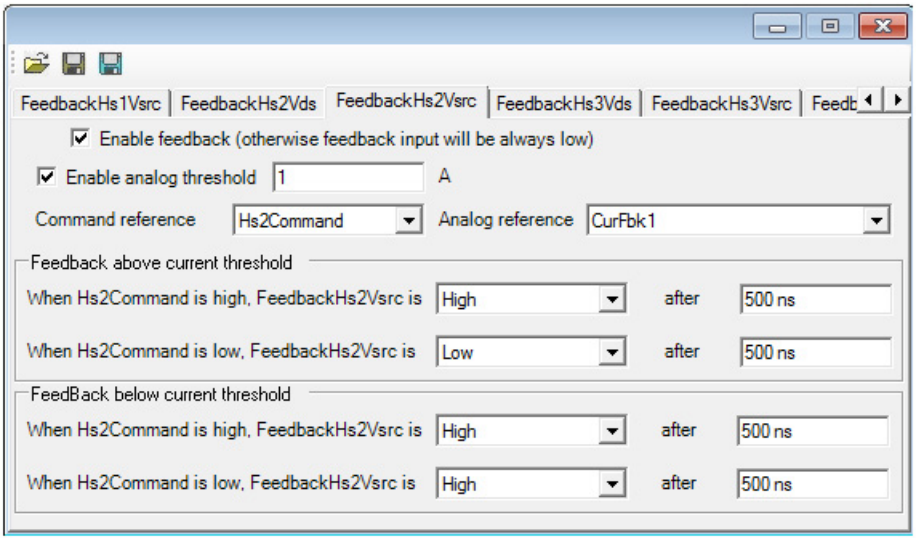


Figure 20. Diagnosis Feedback Window

The toolbar of the window is used to import/save the feedback definitions. The data for all the tabs (feedback for all the inputs) is saved in the same file. The icons on the toolbar (left to right) allow the user to:

- Import the feedback configuration: this command imports the existing feedback data. A dialog prompts the user for the feedback file, from where its data is imported
- Save the feedback configuration: This command saves the feedback data
- Save the feedback configuration As: This command saves the feedback data into a new file. A dialog prompts the user for the name of the file to where the data is saved

Each feedback can be configured into one of three possibilities, by selecting values on the tab with its name:

- Disabled: if the topmost checkbox ("Enable feedback") is disabled, the related input of the device is always low. This setting is used with inputs which are not used in the application
- Normal: if the topmost checkbox ("Enable feedback") is enabled, but the second one ("Enable analog threshold") is not, the feedback is in normal mode. Its value is determined by a certain formula applied on one of the outputs of the device
- Double: if both the topmost checkbox ("Enable feedback") and the second one ("Enable analog threshold") are enabled, the feedback is in double mode. There are two different formulas to determine the value of the feedback, starting from one of the outputs of the device (the same for both formulas); one formula is applied when a selected analog signal is above a specified threshold, another formula when the same analog signal is below the threshold (e.g. use a certain formula when more than 1A is flowing into the injector, another formula when less than 1A is flowing)

According to the mode selected, only a number of fields needs to be supplied. All the unneeded fields are grayed out when the desired mode is selected.

- Analog Threshold: this field in enabled only in Double mode. If the reference analog signal is above this threshold, the top formula is used, otherwise the low one is used
- Command Reference: this reference is used as input for the formula to calculate the feedback. Through a combobox, it must be selected one of the outputs of the ASIC must be selected
- Analog Reference: this field is enabled only in Double mode. Through a combobox, one of the current feedbacks of the ASIC or one of the loads must be selected (to each load corresponds an analog signal, e.g. current in an injector or boost voltage)
- Feedback: the formula used to generate the feedback is simple: when the selected reference has a transition, the feedback has a transition as well, after a specified time. Which type of transition (rising or falling) and the delay between the transition of the reference and the transition of the feedback can be specified in the formula. It is possible to specify one formula (if normal mode is selected) or two (if double mode is selected)

6 Simulation

After the appropriate simulation setup has been specified, according to the needs of the application, it is possible to run a simulation. The PSC simulator is able to run a cycle accurate simulation, using a model of the PSC. The user can probe:

- The state variable of the defined loads (either current or voltage, depending on the type of load)
- The signals of the PSC, both input, output and internal

Since the number of loads are few (possibly 10-15 in a worst case scenario) while there are a large number of internal signals of the device (possibly a thousand), all the state variables of the loads are recorded during the simulation, while only a part of the signals of the PSC are recorded. The user has to specify this PSC signals subset (called "format" by PSC simulator), otherwise none of the PSC signals are recorded.

After the format has been specified, the simulation can be run, using the commands available on the main toolbar.

6.1 Format

The format and the list of PSC signals that are recorded during the simulation can be specified using the "Structure" and "Signals" windows.

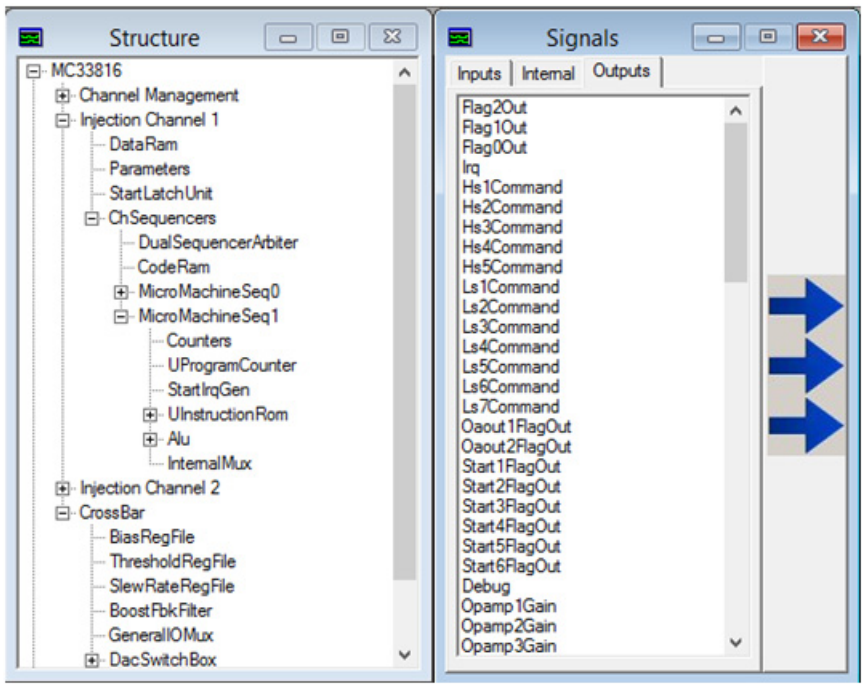


Figure 21. Structure and Signals Window

The "Structure" window contains the structure of the model of the PSC (e.g. MC33816). The top item is the whole device; each item is further divided into sub-blocks, according to the PSC model. When a block is selected in the structure window, its signals are shown in the "Signals" window, divided between Inputs (of the block), Internal (to the block) or outputs (from the block).

The top block contains:

- Inputs: inputs of the PSC
- Internal: signals going from the PSC analog section to the PSC digital section
- Outputs: outputs of the PSC and signals going from the PSC digital section to the PSC analog section

To add signals to the format, select the block containing the signals in the "Structure" window, and then select the desired signals from the "Signals" window, and then press the arrows button. The format can be saved to/loaded from file to avoid creating it for every simulation.

From the main menu select "File->Save format" to save the format to file, select "File->Load format" to load the format from a file.

6.2 Run Simulation

The toolbar in the main window contains four buttons used to control the progress of the simulation:

Figure 22. Main Window Toolbar

- Clock: the first large button, allows the user to specify the clock supplied to the PSC. To be used only if the ASIC requires an externally supplied clock
- Restart: if selected, clears the current simulated data and reverts simulation time to 0. Disabled when simulation time is already 0
- Run: simulates a fixed amount of time or a fixed number of clock cycles, starting from the current simulation time
- Break: breaks the simulation in progress, showing the partial result achieved. Enabled only when the simulation is in progress
- Step: simulates a single clock cycle

The current simulation time is shown in the low left corner of the main window.

All the details defined during simulation setup (actuator/diagnosis/loads) are not automatically used, but must be enabled: when the "Run" command is selected, the Auto Load window is shown to the user, requesting which parts of the project must be loaded into the simulation.

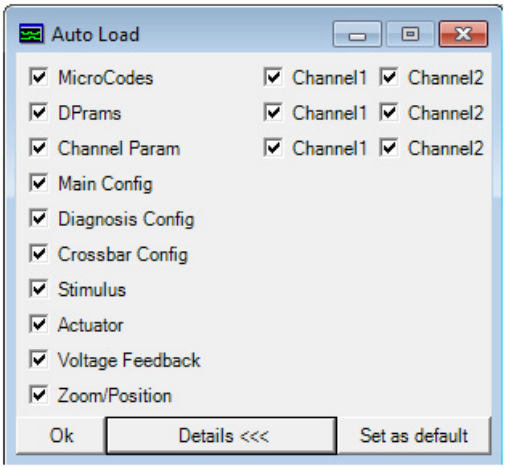


Figure 23. Auto Load Window

- PSC Data: there is a number of PSC data (code/registers/data), depending on the model of the PSC itself (first six rows in the above picture, MC33816). Data which is not loaded displays the default values for the simulation
- Stimulus: load all the stimuli specified using the Stimulus window
- Actuator: load all the load models specified using the Actuator window
- Voltage Feedback: load the feedback formulas specified using the Feedback window
- Zoom/Position: if selected, the current zoom level and position in time of the Wave window are kept, otherwise are reset to the default values

By pressing the button "Set as default", the displayed configuration of checks can be saved to a file (specific to the project).

It is possible to place breakpoints in the code, using the Code tabs of the microcode window. When the chosen code line is executed, the simulation halts, regardless of the simulation time requested by the user.

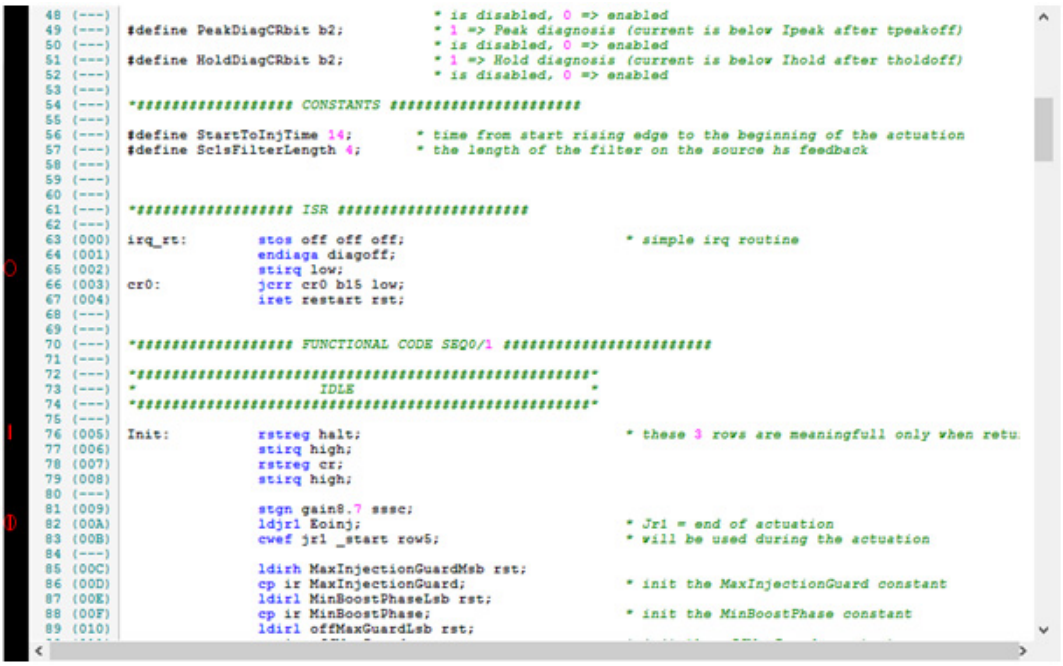


Figure 24. Breakpoints

To place a breakpoint, double click on the black bar on the left side of the code. Each subsequent double click cycles the breakpoint to the next state in the following list:

- No breakpoint (default) => no symbol
- Breakpoint of the sequencer 0 executing that code => red '0' symbol
- Breakpoint of the sequencer 1 executing that code => red 'I' symbol
- Breakpoint of the sequencer 0 and sequencer 1 executing that code => red '0' and 'I' symbol overlapped

If a breakpoint is placed as specific to a sequencer, simulation stops only if that sequencer executes the chosen instruction.

6.3 Simulation Results

The simulation results are available into three windows:

- The actuator currents/voltages are available into the Analog window
- The PSC signals are available into the Wave window
- The program counters are available through the Microcode window

The results are accessible only when the simulation is not running

All the result windows share a common cursor system. When a new cursor is placed in the Current or Wave window (cursors cannot be placed in the Microcode window), a cursor of the same color is created into the other windows, tracking the cursor placed by the user. At any time, the windows contain the same cursors, placed at the same time (e.g. all the windows have a green cursor at time 250 us, a gray cursor placed at time 920 us and so on).

Figure 25. Tracking Cursors

6.3.1 Analog

The Analog window is divided into a variable number of tabs, one for each actuator that has been defined in the actuator window. For each actuator only one signal is represented, the one used as state variable during the description. In case of the ideal R-L model, it is the current flowing into the inductance. In case of a custom model, it can be either a current or a voltage, depending on the feedback selected:

- If Boost Feedback is not set to None, the state variable is a voltage
- If Current Feedback is not set to None, the state variable is a current

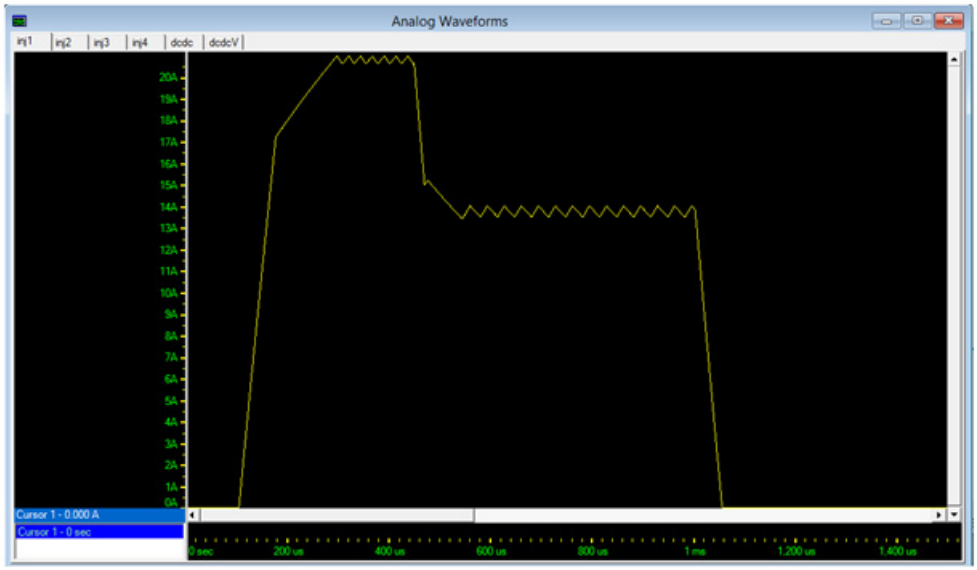


Figure 26. Simulation Result, Current in Actuator

The Analog window allows the user to zoom the displayed waveform, both horizontally and vertically. Zoom can be performed:

- Middle mouse button: only zoom in is possible; to zoom drag a rectangle on the part of the waveform to be zoomed
- Context menu: right-clicking on the picture opens a context menu, where the zoom level can be chosen

A new cursor can be placed by right-clicking in the lower-left corner of the window, on the list of currently present cursors. To select a cursor, left-click on its name in the lower left corner of the window.

6.3.2 Wave

In the Wave window, it is possible to see the behavior of all the signals of the "format", the subset of PSC signals selected to be recorded before the simulation.

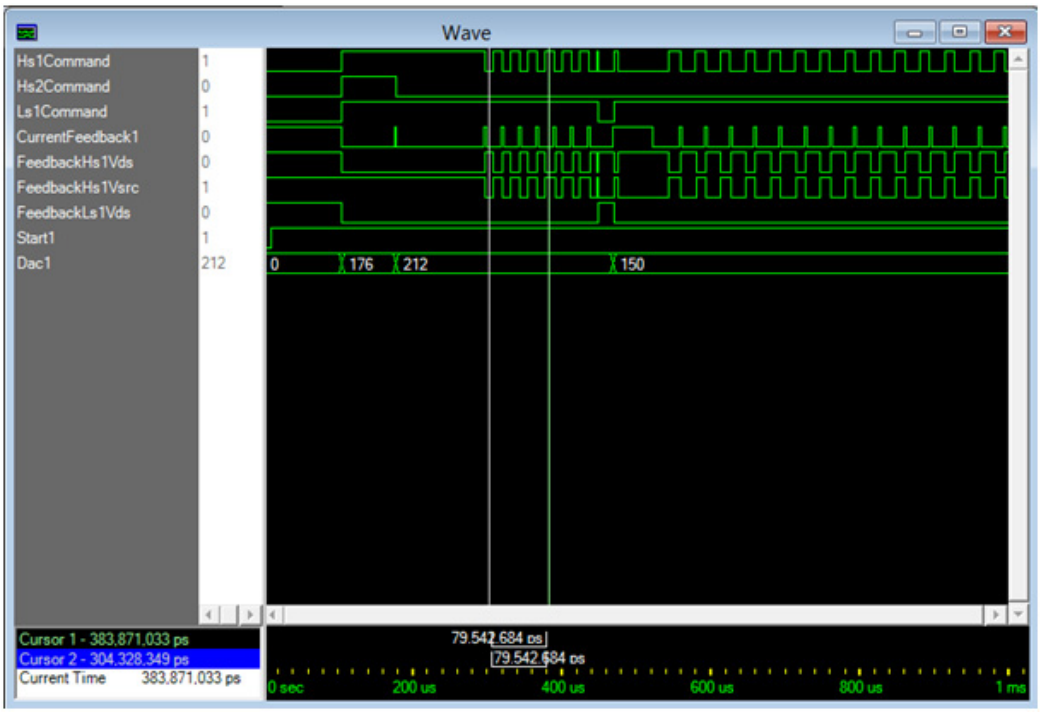


Figure 27. Simulation Result, PSC Signals

The Wave window allows the user to zoom the displayed waveform horizontally. Zoom can be performed:

- Middle mouse button: drag the cursor, with the middle mouse pressed, from left to right to zoom in, from right to left to zoom out
- Context menu: right-clicking on the wave display opens a context menu, where the zoom level can be chosen

A new cursor can be placed by right-clicking in the lower-left corner of the window, on the list of currently present cursors. To select a cursor, left-click on its name in the lower left corner of the window.

In case of a numeric signal, by right-clicking on the signal name it is possible to change the representation of the signal (decimal/binary/hexadecimal).

It is possible to re-order the signals by dragging their names.

6.3.3 Code

When a cursor is placed in the wave or analog window, a tracking cursor is placed also in the Microcode window. The cursor is a mark placed in the black bar in the left side of the window, with the same color of the cursor in the other windows.

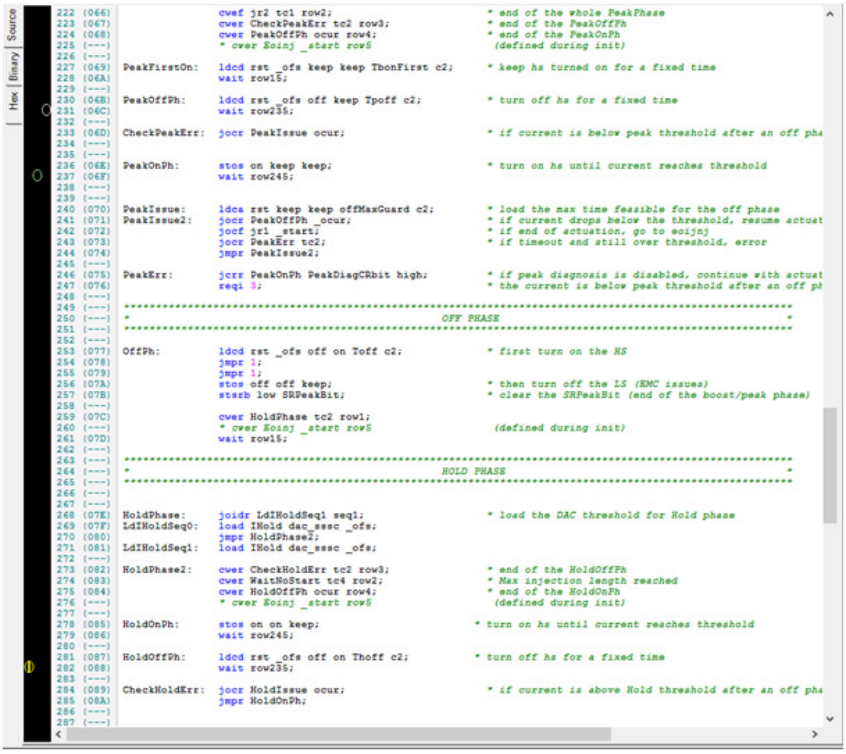


Figure 28. Cursor in Code Window

In the previous picture, in the time instant where the green cursor is placed, the instruction "wait row245" is under execution; in the time instant where the gray cursor is placed, the instruction "wait row235" is under execution.

A yellow mark is added to the last instruction executed at the end of the simulation ("wait row235" as seen in **Figure 28**).

7 Debug

After the simulation is complete, the user can verify the results and check if they are compliant with the specification. If they are not, the simulation results allow the user to find the issue, which could lie in the code/data/registers or in the simulation setup.

The user can modify and re-run the simulation until the result is satisfactory.

8 Release

All the firmware can be saved by using the microcode window. Each memory area is saved into a different file. The file formats supported are:

- Decimal values: one register/memory cell for row, represented in decimal format
- Hexadecimal: one register/memory cell for row, represented in hexadecimal format, followed by a comma. The file also contains comments starting with "//". The file format is compatible with the initialization of an array in ANSI C. This format is commonly used to be included into C code
- Decimal values: one register/memory cell for row, represented in binary format. This format can be read easily by digital simulation languages (e.g. VHDL) and by Freescale SPIGen

All the files needed by the PSC are created when the project is created (or copied): to save the values used in the GUI to files, select the Save (or Save All) button from the microcode button toolbar in the MicroCode window.

8.1 Automatic Documentation

Every time the code is successfully compiled, the PSC simulator tries to produce a Visio block diagram documentation of the code. To obtain this documentation, the user must also specify all the addresses relative to code routines. The exact list of addresses to specify may vary with the PSC, but includes:

- Entry points: starting addresses for code execution
- Interrupt routine addresses

Providing incorrect values for those fields may lead either to incorrect documentation or to a failure to create the documentation.

Other than these addresses, also some "conceptual" errors in the code may lead to failing documentation: if a path exists in the code where the content of a register is used as a target for a jump or wait operation without being initialized, the documentation process fails. In this case, the user is supplied with an error message detailing the "error" path.

Figure 29. Failed Documentation Due to “Error” Path in the Code

The documentation includes, other than the code, a selection of the comments present in the code: comments marked with a single "*" are ignored for documentation purposes, while comments marked with a triple "***" are reported in the documentation.

```
peak_on1:   stos on off on;                        *** Turn VBAT on, BOOST off, LS on
            wait row124;                           * Wait
peak_off1:  ldcd rst ofs keep keep Tpeak_off c2;   *** Load in the counter 2
            stos off off on;                       *** Turn VBAT off, BOOST off, LS on
            wait row123;                           * Wait
```

Figure 30. Documentation Comments Example

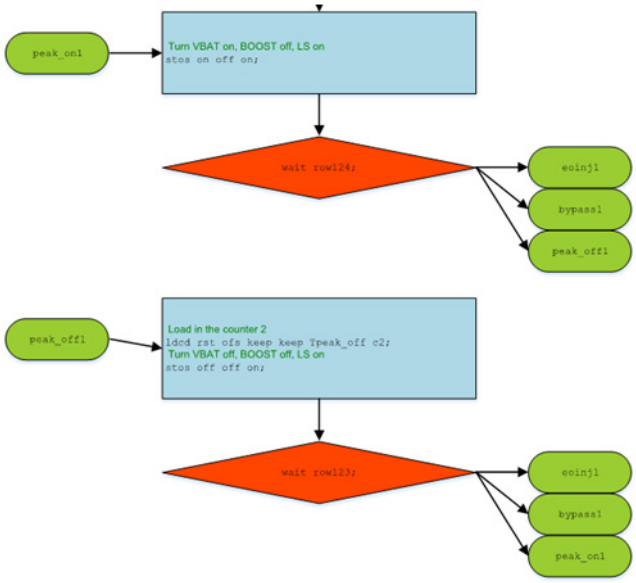


Figure 31. Comments in Documentation

In the previous example, three comments are reported in the documentation (in green), while two are not.

It is possible to fragment the documentation in "pages" to produce a number of small diagrams instead of a single large diagram. It is possible to insert some tags, where the desired name of the page is marked with the "***###" tag.

```
***###PeakPhase
peak_on1:   stos on off on;                     *** Turn VBAT on, BOOST off, LS on
            wait row124;                        * Wait

peak_off1:  ldcd rst ofs keep keep Tpeak_off c2;    *** Load in the counter 2
            stos off off on;                        *** Turn VBAT off, BOOST off, LS on
            wait row123;                            * Wait

***###BypassPhase
bypass1:    ldcd rst ofs keep keep Tbypass c3;      * Load in the counter 3
            stos off off off;                       * Turn VBAT off, BOOST off, LS off
            cwer hold1 tc3 row4;                    * Jump to hold when tc3
            wait row14;                             * Wait
```

In the example just shown, the produced documentation contains two pages, one named "PeakPhase" with the code of the first five lines, the second named "BypassPhase" with the code of the last four lines.

8.2 Acronyms

Acronym	Definition
ECU	Electronic Control Unit
IC	Integrated Circuit
I&M	Ideas & Motion s.r.l.
PSC	Programmable Solenoid Controller

9 References

Following are URLs where you can obtain information on related Freescale products and application solutions:

Document Number and Description	URL
MC33816 Data Sheet	http://cache.freescale.com/files/analog/doc/data_sheet/MC33816.pdf
Freescale.com Support Pages	**URL**
MC33816 Product Summary Page	http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MC33816
KIT33816AEEVM Tool Summary Page	http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=KIT33816AEEVM
Analog Home Page	http://www.freescale.com/analog
Automotive Home Page	http://www.freescale.com/automotive

10 Revision History

Revision	Date	Description of Changes
1.0	6/2014	• Initial Release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Document Number: MC33816SIMUG
Rev. 1.0
6/2014