Managing HIL, SIL and MIL Simulation with SIMulation Workbench™

A software solution for creation, control, communication and I/O connection of handwritten or autocoded models

By: Mark Slater
Concurrent Systems Analyst
January 2018
Overview

An increasing number of commercial and military systems now contain one or more ‘black boxes’ — devices that take input signals, make the necessary calculations, and then produce signal outputs. The logic that drives these black boxes is designed by engineers using common computer languages like C/C++ or modeling tools like Mathworks’ Simulink®. The resulting programs or models represent the code destined to run in the black box, other simulated components of the system, and often the real world. The final testing environment will contain some combination of simulated models, actual hardware, and possibly human interaction.

Testing code that will eventually reside in a black box without the actual hardware is called Software-in-the-Loop (SIL). Installing software in an actual device and testing that device in a virtual environment is called Hardware-in-the-Loop (HIL). If human interaction is desired or required, the test also includes a Man-in-the-Loop (MIL) component. Hosting and managing such a virtual environment requires an operating system with real-time performance and a full-featured executive. Concurrent Real-Time’s RedHawk™ Linux® and SIMulation Workbench provide the real-time Linux environment and full-featured executive required to construct tests using vehicle and aircraft software models, data acquisition I/O and hardware devices that may be part of a given test.

Introduction

SIimulation Workbench (SimWB) was designed by Concurrent Real-Time to satisfy a growing need within our customer base. Concurrent has traditionally supplied tools necessary to help developers create, debug and execute complicated multi-model, multi-rate, multi-threaded hand-written models in a real-time environment. Today, however, many engineers are using tools such as Mathworks’ MATLAB/Simulink for model creation. Simulink allows engineers to focus on their specialty, easily creating high-fidelity mathematical models from drop-down menus of functions, operators, busses and connectors. The models can then be exercised in a non-real-time, controlled environment to verify functionality and accuracy. Although this paper focuses on Simulink and hand-written models, SimWB also supports other popular modeling tools such as VI-grade, SIMPACK, AMESim, Dymola, veDyna, MapleSim GT-Suite and CarSim.

A challenge arises when it comes time to move these models to a real-time environment and connect their signal inputs and outputs to actual I/O in the real-time system. SimWB was designed with this requirement in mind. SimWB provides interfaces for all aspects of model integration, I/O mapping, test creation and execution, data visualization, recording and
playback, and provides a scripting language for test control. Existing customers of Concurrent have successfully utilized SimWB to combine legacy, handwritten models with models produced in Simulink and other tools, and to map model variables to I/O points provided by cards in the system’s native bus, or cards in one or more expansion chassis.

**SIMulation Workbench**

SimWB offers a number of components that support all levels of test creation and execution. There is one for creation, selection and maintenance of its Real-Time Database (RTDB); one for importing models and hand-coded models; one for controlling tests and recording and playback of data; and one for monitoring model variables during a test or creating a Human Machine Interface (HMI) GUI for visualization and control.

**RTDB**

The RTDB is actually two shared memory areas that serve as the heart of the execution environment. Any model variable that will be connected to an I/O channel or accessed for recording, playback or visualization should be part of the RTDB. One of the two shared memory areas that make up the RTDB is called the Current Value Table or CVT. As the name implies, the CVT will have pointers to the current value of each defined RTDB variable. The other shared memory area contains all meta-data associated with each RTDB variable. This meta-data includes items like variable type, description, recordability, engineering conversion units and I/O mapping.

**Using the RTDB**

Customers can use the RTDB in basically two ways. The most common case is where RTDB items are actual variable names from the Simulink model or the handwritten code. Names like `eng2_N1`, `eng2_oil_temp`, `weight_on_wheels` and `primary_bus_voltage` are all representative of model variables, some of which may drive or be driven by actual I/O in the test system. In fact, SimWB can build the RTDB automatically from the signal sources and signal sinks found in the Simulink model.

The other philosophy in RTDB creation is to have the RTDB represent all of the I/O points by name. This RTDB would have names like `ai_brd1_ch1`, `ai_brd1_ch2`, `di_brd2_ch5` and `do_brd3_ch7`. This case is more common in HIL test stands where there are no models per se and SimWB is being used to cycle the reading and writing of I/O points. These systems typically have expansion chassis full of the I/O cards necessary to exercise the unit under test, (UUT) and additional racks of relays and signal conditioning cards. SimWB’s scripting language is then used to drive system I/O to stimulate the UUT and monitor the outputs from the UUT to
measure performance. Concurrent can supply scripts that will automatically produce an RTDB based on the I/O cards that populate the system.

**Import of Simulink Models**

Development of Simulink models can be done natively on the RedHawk Linux host as long as the appropriate MathWorks licenses are available. It is not uncommon, however, for MATLAB® and Simulink development to be done on Windows®-based systems that are dedicated to that purpose. SimWB is designed to accommodate this design philosophy and provides a small piece of software for the Windows system that will add a SimWB pulldown to the Simulink menu (Figure 1). Regardless of the development system, the Simulink model window will provide a SimWB pulldown Configuration Tool that will enable the user to select either the ‘localhost’ or a RedHawk system on the local network as the target system where the compiled model will run under the control of SimWB in a real-time environment (Figure 2).

**Figure 1: Simulink Model**

The Configuration Tool also provides a dialogue that lets the user associate the Simulink model with a previously created RTDB, or have the tool automatically create the RTDB from the signals
it finds in the Simulink model. Even when using the auto-create feature, the user can be selective about signals that end up in the RTDB. The tool will give the user the option of prefixing symbol names with an ‘SW’ and then creating the RTDB from only those symbols. The option to append the Simulink variables onto an existing RTDB is also provided.

![Figure 2: SimWB Real-Time Target Configuration Tool in Simulink](image_url)

**How to Import Simulink Models**

Importing Simulink models into SimWB control is straightforward. The Configuration Tool above provides a button that will engage the MathWorks’ MATLAB/Simulink Coder to translate the Simulink model to C or C++ code, push that code to the host system and compile it to an executable for inclusion in a SimWB test. The only real decision a user must make is how to handle RTDB creation. The contents of the RTDB will vary by project. For projects consisting of only Simulink models, the auto-creation feature should work well to place all variables of interest into the RTDB. Other projects, however, may utilize hand-coded models with unique variables of their own. For these cases, the RTDB creation dialogue within SimWB can be used to create RTDB variables to match those in the handwritten models.

When importing user code into SimWB, a boilerplate is provided with all relevant initialization and memory mapping functions already in place. For example, if the user code contains the variables `target_range` and `ownship_height`, these two variables can be added to the RTDB with the RTDB creation dialogue. When the user’s code is integrated into the boilerplate created by
SimWB, the user can simply modify his code to rename the variables to `cvtTable.target_range` and `cvtTable.ownship_height`. All variables preceded by the `cvtTable` structure heading will be referring to the RTDB shared memory area containing the Current Value Table. Ultimately it is not necessary to use the boilerplate provided by SimWB. User code can utilize the provided API to access the RTDB and to synchronize itself with models being controlled by SimWB.

**I/O Mapping**

One of the most useful aspects of SimWB is the ease with which model variables can be connected to I/O points. When defining RTDB variables, input and output variables are specified as Analog, Digital or String. SimWB provides an interface for each I/O type and lists the RTDB variables that the user may choose to connect to the available channels. Figure 3 is an example of the interface for an ARINC 429 card.

![Figure 3: ARINC 429 I/O Mapping Interface](image)

On the extreme right, is a list of the RTDB variables that the user might choose to connect to an ARINC 429 channel. Notice that only variables of type Analog Out are listed here. Variables defined as Digitals or Strings will not appear in the list as they are not supported by this particular I/O card. The interface page for each I/O device is unique and presents the user with all controllable options associated with the specific type of I/O. Connecting an I/O channel to an RTDB variable is as simple as clicking on the channel and then clicking on the variable.
How model variables are mapped to I/O points is a saved state of a named RTDB and becomes part of that RTDB’s metadata. Any number of RTDBs can be created and named. This means that a given I/O point can be connected to one model variable in a particular test case and a completely different variable in a different model in a different test case. There is no contradiction here because only one test case, and therefore only one RTDB, can be run at a time. This provides great flexibility in how the I/O available in a system is used. One set of channels can be used in one test and another, possibly overlapping, set of channels can be used in another.

**Test Creation/Execution**

All models destined to run under the control of SimWB are first compiled to an executable. Whether the model was created in Simulink or hand-coded, SimWB provides mechanisms to compile the model and put the resulting executable in a directory of selectable models. SimWB provides a test creation dialogue where the user selects the RTDB upon which the test will be based. The user then selects, from all the available executables, those that will be part of the test. Graphical representations of these models are placed on a palette (Figure 4). From this palette, scheduling parameters can be set for each of the models, i.e. scheduling policy, priority, CPU mask, frequency period and order of execution.

**Figure 4: Test Creation**
Test Session

The ultimate runnable entity of SimWB is called a Test Session. In creating a Test Session, the user can fully define the features and parameters of a test run. The user selects a test name and description, whether or not the test will log data, whether the test will use direct or buffered IO, the duration of run, what to do in the case of overruns, and the set of initial conditions that apply to the session.

The ability to create a set of initial conditions is a powerful feature. In addition to being able to set the initial value of any RTDB-defined variable, the user also has access to any parameter defined in the Simulink model. This means that model parameters such as multipliers and lookup tables can be changed on a session-by-session basis without the need to modify the Simulink model and recompile it to code and executable. In defining the Test Session, the user also has the option of assigning a set of environment variables, choosing a scheduler other than the default scheduler, and specifying whether debugging or kernel tracing will be part of the test run.

A Test Session’s step size reflects the rate at which SimWB will cycle and is defined automatically by the fastest rate found in any Simulink model involved in the test. If there are no Simulink models involved in a test, then the “Fixed Step” field is editable and the user can specify the rate at which SimWB will cycle.

Simulation-Only Mode

SimWB can also test models in non-real-time without connection to any physical I/O. This feature is useful for verifying models early in a development cycle when hardware components are not available. Users can create and configure an RTBD and perform simulation testing on multiple models and evaluate their interaction. In simulation-only mode, the synchronous simulation loop is dispatched with no timing source. Overrun detection is disabled as the loop is restarted as soon as it finishes. Only models, user models, and scripts can be run with this scheduler option. Users can run their models as fast as possible to produce data for later analysis.

Data Recording and Playback

SimWB provides a data recording and playback capability. When an RTDB variable is defined, the user can select whether this variable should be subject to recording or not, and at what frame rate the recording should take place. Whether or not recording of any RTDB variables will take place during a given test is controlled via a checkbox on the test control dialogue. Recorded data can be stored locally on the test system or streamed over a network to a system dedicated to the collection and storing of recorded data.
Using Data Recording and Playback

Any RTDB variable, input or output, can be tagged for recording. Not all inputs or outputs need to be recorded for a particular test. In Playback mode (Figure 5), any unrecorded inputs will be fed to the models as they normally would be from actual, current I/O values, while recorded inputs will override the current values of the I/O channels associated with the input. In Playback mode with the models active, any outputs that were not recorded will be generated from the executing models while recorded outputs will override current calculations. The Playback dialogue offers a “Data only” check box. When this box is checked, I/O tasks do not run and the models do not cycle. Only those RTDB variables that have been recorded will be played back and be available for data visualization using the RTViewer or an HMI GUI.

Scripting

SimWB supports two types of scripting -- input cycle scripting and the SWm scripting language. Input cycle scripts can be written in C or Python. SimWB provides an interface (Figure 6), which will create a script template.
Script templates provide the framework for linking in a synchronized script module that will be executed after the input tasks read the I/O and before inputs are passed to the executing models that make up a test.

The SWm scripting language provides a traditional, comprehensive scripting experience. SWm offers a range of commands for manipulation of RTDB values, standard I/O interactivity for a test operator, string manipulation and logical and conditional operators for test control (Figure 7).
Using Scripting

C or Python input cycle scripts can be used to control the value of I/O inputs regardless of the actual value of the hardware input. With these scripts, the user can manipulate inputs precisely as desired in order to replicate them exactly over repeated runs of a test. The user may have actual data from a hardware system that is being modeled that requires, for example, that with these given inputs, on a particular time scale, the model should produce certain outputs. A user can then run a script that produces the inputs, and then tweak the model until it produces the desired outputs. The SWm scripting language can be used to run tests under script control for setting inputs, watching outputs with conditional statements and logging results to an HTML test log. The script can also dialogue with a test operator, ask the operator to perform functions, manipulate hardware and verify results before continuing. These results are also logged to an HTML test log (Figure 8).
In some HIL test stands designed by Concurrent for customers, the SWm script even replaces the concept of running models in a test. In these cases, the system is populated with numerous types of input and output boards, relays, power supplies and signal conditioning boards. The I/O points are wired to an interface panel that can be connected to the customer UUTs. The script can then manipulate test stand relays and outputs that are fed to the UUT. Outputs from the UUT are read as inputs to the test stand and evaluated by the running script for verification of results and passed tests.

**HMI GUI Builder**

SimWB does provide several ways of monitoring or controlling RTDB variables through the RTViewer. Here, the user can select RTDB variables for monitoring or modification on a command line, or double click for a graph. On the other hand, a key feature of SimWB is the built-in GUI Builder for constructing HMIs for control and monitoring of running tests. The HMI GUI builder (Figure 9) does not require any third-party software and is an integral part of SimWB. It provides a large collection of widgets that are easily connected to the RTDB variables.
in a given test. In addition to the variety of knobs, gauges, sliders, toggles, LEDs, etc., action buttons are provided that can be tied to SimWB control functions like test starting, stopping, pausing and resuming.

Figure 9: HMI Builder

Using the HMI GUI Builder

One of the most useful features of the HMI GUI Builder is the ability to import digital images that can be incorporated into the HMI GUI. Images can be used in strictly a cosmetic fashion to improve the look and feel of the GUI or, as in Figure 9, be used as part of the functional display. Above, a digital image of an aircraft control panel is used as a background starting point. Some of the instruments, such as the ADI and the slip indicator, are widgets provided by SimWB that were simply dropped on and sized properly for the background. Others, like the airspeed indicator and the climb rate indicator, are simply needles. These widgets use the actual image of the instrument as a background with the real needle removed and a SimWB gauge widget dropped on top of it. The gauge widget can then be properly scaled and edited so that the needle is the right size and color and all but the needle is transparent. In this way, custom HMI GUIs can be realistically constructed to maximize the connection between SimWB tests and the actual hardware involved in a test.
Conclusion

SIMulation Workbench was created to fill a gap in the toolsets needed by Concurrent Real-Time’s simulation customers. Users required an easy, intuitive way to integrate legacy handwritten models with newly created Simulink and other models, connect them to system I/O, and run them at real-time frequencies in a deterministic environment. RedHawk Linux is a hard real-time, deterministic operating system that supplies the user with the tools necessary to control process, processor and interrupt affinities. System resources can be protected from nominal Linux activity and devoted to the users’ real-time applications. SIMulation Workbench was designed to run on RedHawk Linux systems and take advantage of that ability to shield resources. SimWB grants the user complete control over the resources used by the tool itself (I/O tasks and the scheduler), as well as those used by the executing models. Users are free to shield specific cores from OS activity and assign test components (models) to run on those shielded cores. In short, SimWB is the perfect tool for any project involving HIL, SIL or MIL simulation or testing.

About Concurrent

Concurrent Real-Time is the industry’s foremost provider of high-performance real-time Linux® computer systems, solutions and software for commercial and government markets. The Company focuses on hardware-in-the-loop and man-in-the-loop simulation, data acquisition, and industrial systems, and serves industries that include aerospace and defense, automotive, energy and financial. Concurrent Real-Time is located in Pompano Beach, Florida with offices through North America, Europe and Asia. For more information, please visit Concurrent Real-Time at www.concurrent-rt.com

©2018 Concurrent Real-Time, Inc.. Concurrent Real-Time and its logo are registered trademarks of Concurrent Real-Time. All other Concurrent Real-Time product names are trademarks of Concurrent Real-Time while all other product names are trademarks or registered trademarks of their respective owners. Linux® is used pursuant to a sublicense from the Linux Mark Institute.