People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering

Department of Electronics

Final Year Project Report Presented in Partial Fulfilment of the Requirementsfor the Degree of

MASTER

In **Electronics**

Option: **Computer Engineering**

Title:

Interfacing Simulink to NI-7851R Card using SIT and Veristand for Hardware-In-The-Loop testing

Presented by:

- **ABDERRAHMANE Saad**
- **BELHAOUI Salaheddine**

Supervisor:

Dr. MAACHE

Registration Number:………/2019

Dedication

First and foremost, we are thankful to Allah, the most gracious, the most merciful for helping us finish this modest work. Thanks to my beloved parents who have pushed me to overpass all obstacles and hitches I faced in my entire life, their love and guidance are with me in whatever I pursue. I wish to express my sincere appreciation to Dr. MAACHE for his guidance and encouragement throughout this project. To my friend, BELHAOUI Salaheddine, I thank him for helping me to achieve this work. Last but not least, I thank all my family and my friends.

ABDERRAHMANE Saad

Dedication

Alhamdulillah. Thanks to Allah, whom with His willing giving us the opportunity to complete this Final Year Project. Firstly, I would like to express my deepest thanks to, Dr. MAACHE , my supervisor who had guided me through the path of the completion of this project. I also want to thank the lecturers and staffs of IGEE for the great journey of 5 years in the institute. To my friend, ABDERRAHMANE Saad, I thank him for working with me to complete this project. Deepest thanks and appreciation to my parents, because without them, I could not reach that far. Deepest thanks and appreciation to my family, colleagues , and everyone for their cooperation, encouragement, constructive suggestion and full of support for the report completion, from the beginning till the end.

BELHAOUI Salaheddine

Abstract

The project studies two software approaches to generate real-time output signals from a Simulink model in order to allow for Hardware-in-the-loop testing. These two tools are NI SIT (Simulation Interface Toolkit) and NI Veristand. As a case study, a Simulink model of "Fault Detection" was tested using NI PCIe-7851R Reconfigurable Multifunction FPGA-based card. The Project discusses how to interconnect and interface LabView or Veristand with Simulink, in a way that we could easily deploy and debug Matlab model using NI R Series hardware. This is particularly useful for power systems' researchers as it allows them to directly use the NI PCIe-7851R Reconfigurable card without translating their Simulink models to LabView.

Contents

List of Figures

Acronyms

A/D Analog to Digital

AI Analog input

AO Analog Output

D/A Digital to Analog

DAQ Data acquisition

DIO Digital Input Output

DLL Dynamic Link Library

DMA Direct Memory Access

FIFO First In First Out

FPGA Field-Programmable Gate Array

HIL Hardware-in-the-loop

IP Internet Protocol

MIL Model-in-the-loop

NI National Instruments

RIO Reconfigurable Input/Output

RT Real Time

SCB Shielded Connector Block

SIL Software-in-the-loop

SIT Simulation Interface toolkit

TCP Transmission Control Protocol

VI Virtual Instrument

Introduction

Hardware In The Loop (HIL) simulation is becoming a significant tool in prototyping complex, highly available system, especially when a portion of the given system is simulation and a portion of the same system is a hardware implement, with proven results in industrial application. The Research Laboratory at our Institute uses NI R series Multifunction FPGA-based cards in performing HIL tests. This type of cards are compatible only with LabView tool and not with Simulink. Hence, any researcher who would like to use this type of card must translate their Simulink models to LabView. The principal idea behind this project is to overcome this problem. Matlab Simulink and NI R series Multifunction RIO can be linked together through the LabVIEW Simulation Interface Toolkit (SIT) or Veristand. This allow researchers and engineers to interactively verify SIMULINK models and easily deploy these models to real time hardware for control prototyping and Hardware in the loop testing. The main aim of this project is to investigate the most efficient ways to allow researchers to use their -already existing- Simulink models to access the R series cards without re-designing their models using LabView.

At first, the solution of writing a low level driver for the PCIe bus in order to control the card directly from Simulink was taken into consideration. However, the closed box controller on the card prevented us from doing so.

One of the objectives of this project in to implementation/test a Fault detection Simulink model using the two different environments: LabVIEW SIT and Veristand.

This report is organized as follows: Chapter one dives into background and essentials about NI 7851R card and introduction to Hardware in the loop. In chapter two, the first technique LabVIEW simulation interface toolkit is explained with a background about its functionality is presented. the third chapter deals with the second interfacing method Veristand by providing concepts on how it works. The last chapter discusses a case study and comparison between the two interfacing solution.

Chapter 1

Background Materials

I Chapter I: Background Materials

I.1 Hardware-in-the-loop/Software-in-the-loop/Model-in-the-loop

I.1.1 Hardware in the loop testing

Hardware-in-the-loop (HIL) testing is a technique where real signals from a controller are connected to a test system that simulates reality, tricking the controller into thinking it is in the assembled product[1]. Test and design iteration take place as though the real-world system is being used. You can easily run through thousands of possible scenarios to properly exercise your controller without the cost and time associated with actual physical tests. You use HIL simulation to test your controller design, You can also use HIL to determine if your physical system (plant) model is valid. In HIL simulation, you use a real-time computer as a virtual representation of your plant model and a real version of your controller. Figure 1.1 shows a typical HIL simulation setup. The desktop computer (development hardware) contains the real-time capable model of the controller and plant. The development hardware also contains an interface with which to control the virtual input to the plant. The controller hardware contains the controller software that is generated from the controller model. The real-time processor (target hardware) contains code for the physical system that is generated from the plant model.

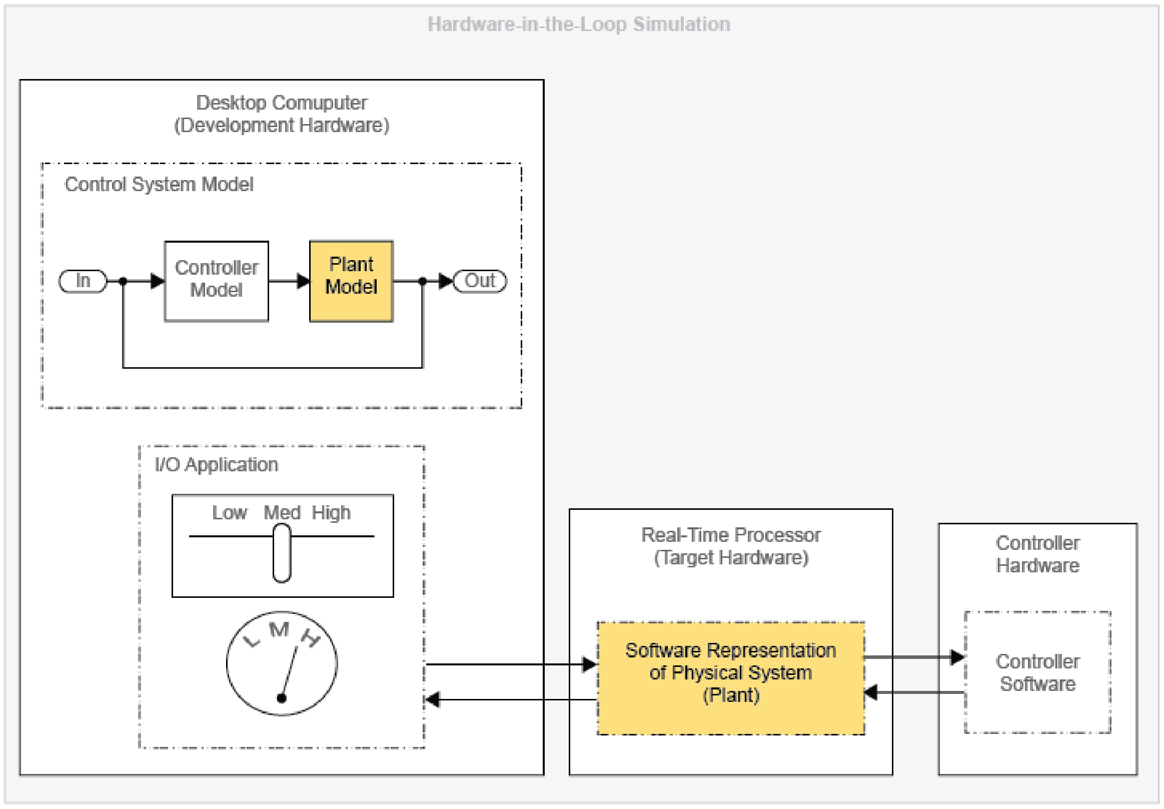


Figure 1.1: Real time Hardware in the loop simulation

I.1.2 Model in the loop testing

Model-in-the-loop testing (MIL) and simulation is a technique used to abstract the behaviour of a system or sub-system in a way that this model can be used to test, simulate and verify the desired system [2]. By using Simulink for model definition you can test and refine that model within a desktop environment, allowing a complex system to be managed efficiently.

I.1.3 Software in the loop testing

SIL is testing any software/firmware/algorithm/control system in such a way that a piece of software simulating a piece of hardware, or simulating a physical component, or a physical system, including possibly its response or other characteristics in a system. The system can be either open-ended (feed-forward only), or with feedback [3].

I.1.4 Why Perform Hardware-In-The-Loop Simulation?

Use HIL simulation to test the design of your controller when you are performing Model-Based Design (MBD). Validation involves using actual plant hardware to test your controller in real-life situations or in environmental proxies (for example, a pressure chamber). In HIL simulation, you do not have to use real hardware for your physical system (plant). You also do not have to rely on a naturalistic or environmental test setup. By allowing you to use your model to represent the plant, HIL simulation offers benefits in cost and practicality. There are several areas in which HIL simulation offers cost savings over validation testing. HIL simulation tends to be less expensive for design changes. You can perform HIL simulation earlier than validation in the MBD workflow so you can identify and redesign for problems relatively early the project. Finding problems early includes these benefits:

- Your team is more likely to approve changes.
- Design changes are less costly to implement.

In terms of scheduling, HIL simulation is less expensive and more practical than validation because you can set it up to run on its own. HIL simulation is more practical than validation for testing controller's response to unusual events. It can also be used to observe controller's responses to stimuli that occur in inaccessible environments like deep sea or deep space [1].

I.2 NI PCIe-7851 RIO card

I.2.1 Overview of Reconfigurable I/O

R Series Multifunction RIO devices are based on a reconfigurable FPGA core surrounded by fixed I/O resources for analog and digital input and output. Its behavior can be configured to match the requirements of the measurement and control system. You can implement this user-defined behavior as an FPGA VI to create an application-specific I/O device

Figure 1.2 shows an FPGA connected to fixed I/O resources and a bus interface. The fixed I/O resources include A/D converters, D/A converters, and digital I/O lines [4].

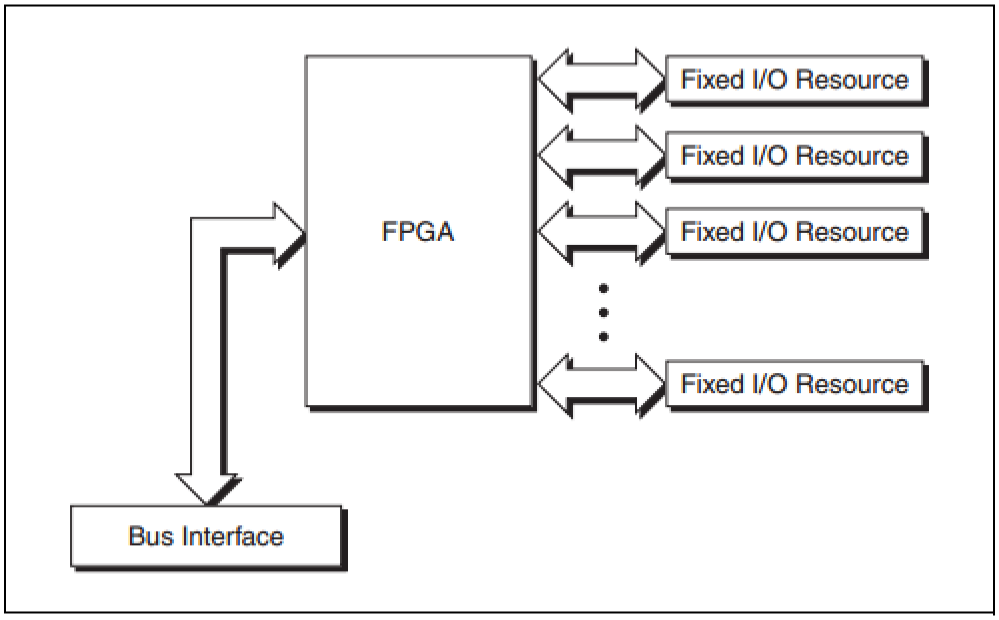


Figure 1.2: High-Level FPGA Functional Overview

Software accesses the R Series device through the bus interface, and the FPGA connects the bus interface and the fixed I/O to make possible timing, triggering, processing, and custom I/O measurements using the LabVIEW FPGA Module. The FPGA logic provides timing, triggering, processing, and custom I/O measurements. The bus interface provides software access to the device. The remaining FPGA logic is available for higher-level functions such as timing, triggering, and counting. The functions use varied amounts of logic.

I.2.2 NI 7851R Overview

The NI 7851R card shown in Figure 1.3 have eight independent, 16-bit Analogue In (AI) channels; eight independent, 16-bit Analogue Out (AO) channels; and 96 bidirectional Digital IO (DIO) lines that you can configure individually for input or output.

The NI 7851R has a Xilinx Virtex-5 LX30 FPGA, it has three connectors [4].

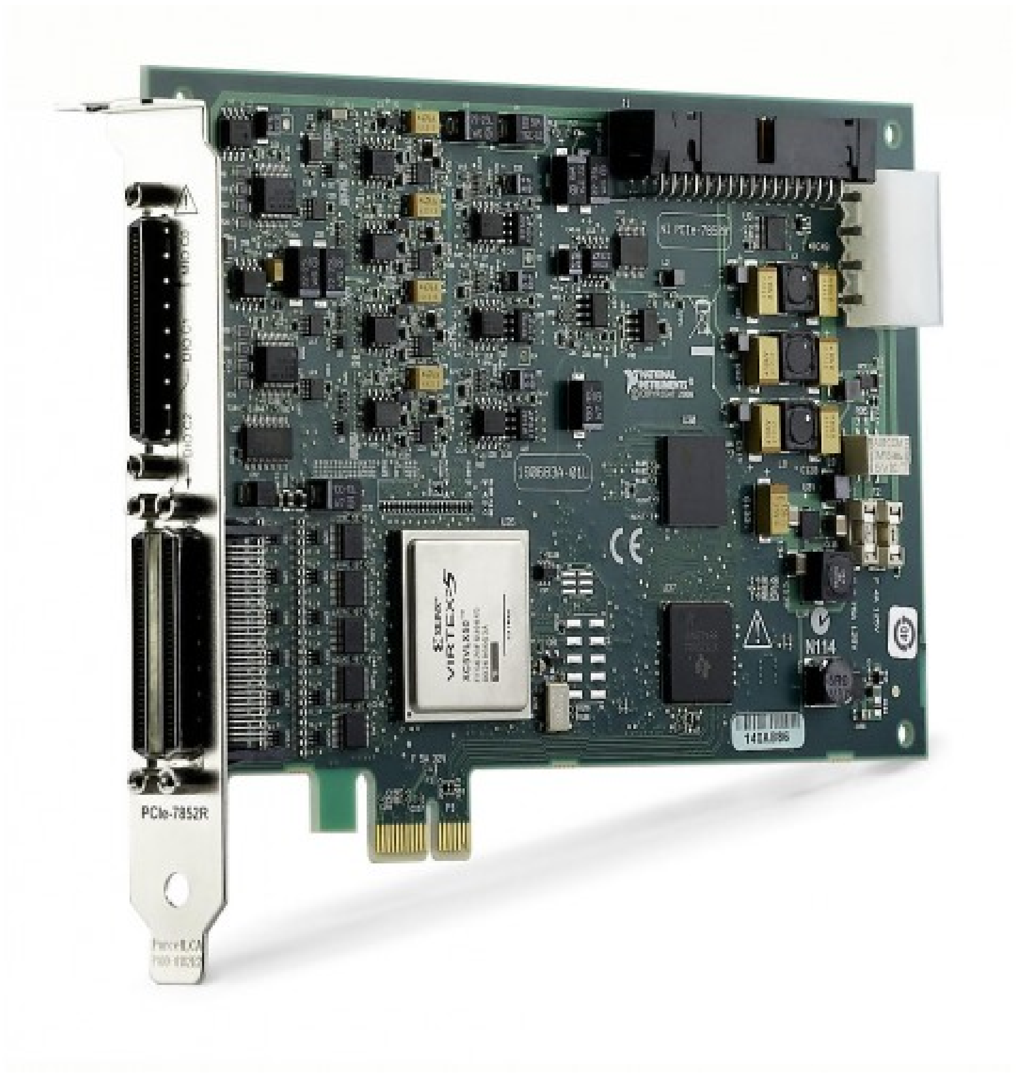


Figure 1.3: NI PCIe 7851R

The Virtex-5 FPGA device used has the architecture presented as in Figure 1.4

I.2.3 SCB-68 Connector

The SCB-68 shown in Figures 1.5 has 68 screw terminals for I/O signal connections. To use the SCB-68 with the NI 78xxR, you must configure the SCB-68 as a general-purpose connector block [4].

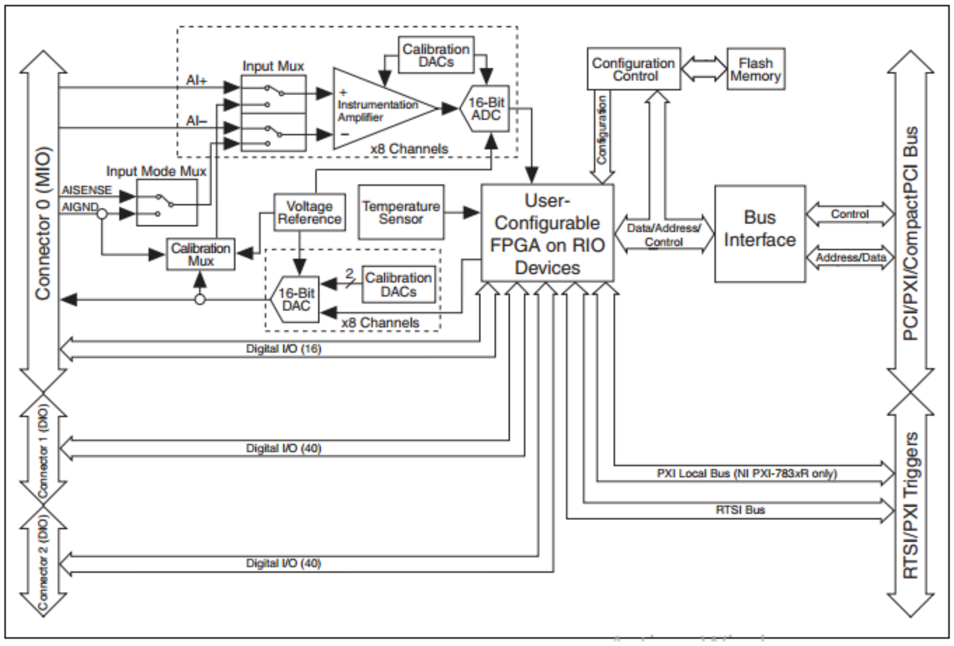


Figure 1.4: NI 7831R-7833R-784xR-785xR Block Diagram

Figure 1.5: SCB 68A Connector

Chapter 2

Working With Simulation Interface toolkit

II Chapter II: Working With Simulation Interface toolkit

II.1 SIT Functionality

The LabVIEW Simulation Interface Toolkit interfaces LabVIEW with Simulink application software and Real-Time Workshop application software in a way that enables you to develop, prototype, and test control systems using models developed in the Simulink simulation environment. You use the Simulation Interface Toolkit to create a LabVIEW user interface for a model developed in the Simulink simulation environment and run a simulation on a real-time (RT) target.

II.1.1 Components of a Simulation

A simulation consists of the following components [5]:

- **Model**: a simulation block diagram in graphical form, another source code form (e.g., C code), or compiled form. Models contain inputs and outputs that send and receive data. Models contain parameters you can manipulate and signals whose values you can view.
- **Host VI**: The VI that you use to manipulate a model. The host VI consists of a front panel and a block diagram. You use front panel controls to manipulate model parameters. The block diagram of the host VI contains the code that defines mappings between front panel controls/indicators and model parameters/signals.
- **SIT Server**: The server that uses a TCP/IP connection to transmit data between the host VI and the model. You must launch the SIT Server, which starts automatically when you launch MATLAB application software, before running a simulation. By default, the SIT Server runs on port 6011.
- **Host Computer**: The computer on which you run the host VI. The host computer must be a PC running Windows 7/Vista/XP/Server 2008 (64-bit)/Server 2003 (32-bit).
- **Execution Host**: The computer on which you run the MATLAB application software, the SIT Server, and the simulation itself. The execution host can be the host computer or a Windows computer on the same TCP/IP network as the host computer.

The following Figure 2.1 shows how these components work together:

When you run the host VI, the diagram code initializes the simulation and defines the relationship between host VI controls/indicators and model parameters/signals. As you change the values of front panel controls, the simulation executes the following steps:

1. The host VI block diagram uses TCP/IP to send the new parameter values to the SIT Server.

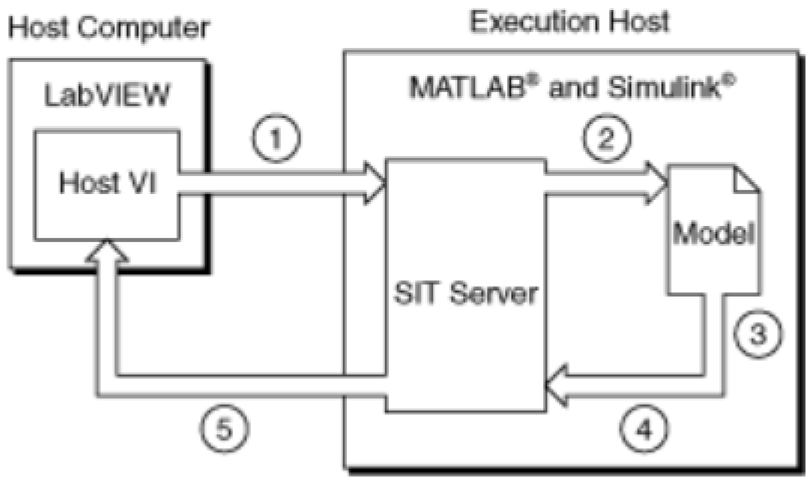


Figure 2.1: Components of a Simulation

2. The SIT Server transmits these new parameter values to the model.
3. The model uses these new parameter values to execute the blocks, which update the appropriate signal values.
4. The SIT Server probes the model signals for which you created mappings.
5. The SIT Server transmits the new signal values to the host VI, which updates the front panel indicators.

II.1.2 Real-Time Simulations

LabVIEW Simulation Interface Toolkit can be used to run a simulation on a real-time (RT) target [5]. The Simulation Interface Toolkit supports certain types of National Instruments RT Series hardware [6] it also can be used to execute an RT simulation on a Windows computer [7]. RT simulations involve the following additional components:

- **LabVIEW Real-Time Module**: this module is needed for running a deterministic RT simulation.
- **LabVIEW FPGA Module**: (Optional) If a mappings is created between National Instruments FPGA targets and a model DLL, any specified FPGA target must have an associated FPGA lvbit file. The Simulation Interface Toolkit provides FPGA bitfiles for the NI PXI-7831R, NI PCI-7831R, and NI PXI-7811R. If it is needed to customize these FPGA VIs or create an FPGA VI for another FPGA target, FPGA Module is required.
- **National Instruments Driver Software**: an appropriate National Instruments driver software is required to communicate with hardware installed in an RT target. For FPGA targets, NI-RIO3.0 or later is desired. For CAN interfaces, NI-CAN 2.6 or later. For DAQ devices, NI-DAQmx 8.7.2 or later.

- **Driver VI**: The VI that manipulates the model DLL. The driver VI and the model DLL run on an RT target. The driver VI is manipulated with a host VI. Driver VI is created using the SIT Connection Manager dialog box.
- **Model DLL**: a model compiled to run on an RT target. A model is enabled to run on an RT target by using The Real-Time Workshop and Microsoft Visual `C++`. To enable a model to run on an RT target, Real-Time Workshop converts the model and any subsystems into a C code version of the model. Note this`C`code version of the model is the same Simulink block diagram model, just in a`C`code form. Microsoft Visual `C++` then compiles the`C`code model into a model DLL named `ModelName.dll,`where ModelName is the name of the model. Real-Time Workshop places the model DLL into the current working directory. The following Figure 2.2 shows this conversion process.

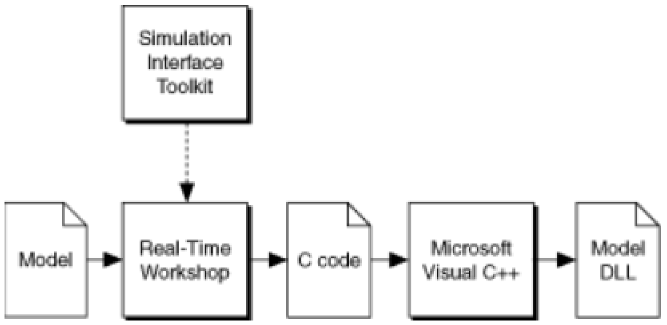


Figure 2.2: conversion process of model into DLL

- **SIT Server**: The SIT Server transmits data between the host VI and the driver VI. If it is not needed to communicate with an RT simulation, an RT simulation can run without the SIT Server[8].It is not needed to launch the MATLAB application software to run a simulation on an RT target.

Figure 2.3 shows how these components work together:

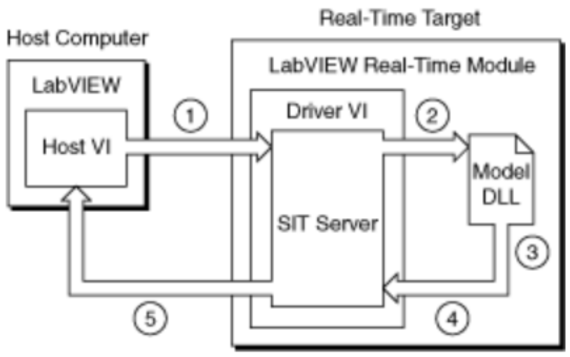


Figure 2.3: RT simulation process

II.2 Understanding the Driver VI

The driver VI contains the code that communicates between the host VI and the target on which a simulation is running [9]. This VI consists of a top-level driver VI, `modelname_driver.vi`, and a support library, `modelname_IO.llb` modelname is the name of the model DLL. These VIs and LLBs exist in the project directory specified in the SIT Connection Manager dialog box. After setting SIT Connection Manager dialog box, the LabVIEW Simulation Interface Toolkit creates a driver VI unique to that simulation. The driver VI contains values specified for the model, such as timing information, and the mappings created between the model and any I/O hardware on the RT target. The Simulation Interface Toolkit uses a template, located in the `labview\vi.lib\addons\Simulation Interface\_ConnectionManager\ scriptdriver\templates` directory, to create this driver VI. This template creates a top-level driver VI that executes the following steps:

1. Initializes the model DLL.
2. Starts the SIT Server on the RT target.
3. Initiates data logging and playback.
4. Configures timing parameters.
5. Calls the IO Base Rate Loop VI, which transfers data to and from the hardware inputs and outputs.
6. Finalizes the model DLL.

II.2.1 The Front Panel of the Driver VI

The front panel of the driver VI contains the following controls that affect the simulation.

- **number of signals**: Specifies the maximum number of signals that the simulation probes.
- **num data points**: Specifies the maximum width of the data buffer that probes the model signals. The total width of all the signals you want to probe must be less than or equal to the width of this data buffer.
- **circular buffer size**: Specifies the depth of the data buffer.
- **Start Server?**: Specifies if the SIT Server starts when the simulation run. If this switch is set to FALSE, the host VI cannot be used to communicate with the simulation. In this situation, the driver VI starts the simulation immediately on execution. The default value of this switch is TRUE, which allows to use the host VI to communicate with the simulation. In this situation, the driver VI waits for the Run command from the host VI before starting the simulation.

II.2.2 The IO Base Rate VI

The block diagram of the driver VI includes the IO Base Rate Loop VI, which is circled in the following Figure 2.4.

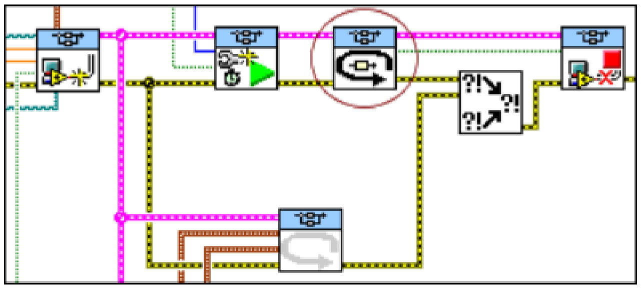


Figure 2.4: the IO Base Rate Loop VI

Like the driver VI, the Simulation Interface Toolkit creates the IO Base Rate Loop VI from a template in the `labview\vi.lib\addons\Simulation Interface\_ConnectionManager\scriptdriver\templates` directory. The IO Base Rate template is sit Base Rate Loop.vi. This template creates a VI named `modelname_IO` IO Base Rate Loop.vi.

The IO Base Rate Loop VI contains the following four custom VIs:

- `modelname_IO IO Init.vi`
- `modelname_IO IO Read.vi`
- `modelname_IO IO Write.vi`
- `modelname_IO IO Close.vi`

The following Figure 2.5 shows these VIs, circled from left to right, on the block diagram of the IO Base Rate Loop VI.

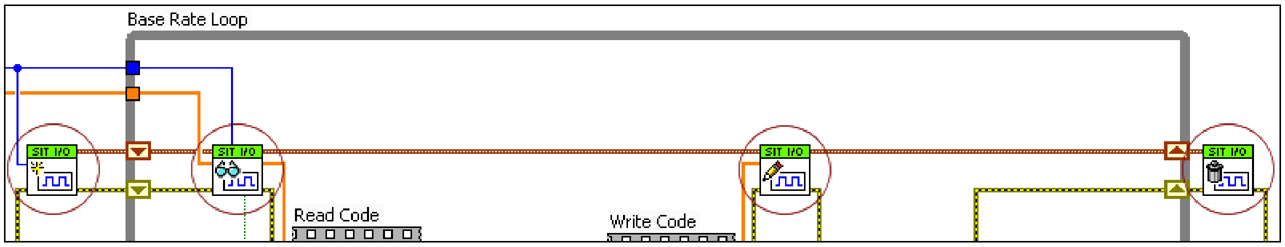


Figure 2.5: the block diagram of the IO Base Rate Loop VI

The IO Base Rate Loop VI uses these four VIs to execute the following steps:

1. Initialize the simulation.
2. Execute the following steps for the duration of the simulation.

   (a) Read data from the hardware inputs.
   (b) Run the SIT Scheduler VI, which takes the hardware input values, runs a single step of the simulation, and receives output values from the simulation.
   (c) Probe the values of the model DLL and send these values to the SIT Server.
   (d) Write the values the model DLL returns to the hardware outputs.
   (e) Perform any commands the host VI sends.

3. Free hardware resources after the simulation finishes.

II.2.3 How the Driver VI Schedules Simulations

Single-rate simulations use only the IO Base Rate Loop, located on the block diagram, to control the timing of the simulation. However, multirate simulations use additional Timed Loops. Because each model task might have a different time step, the Scheduler Loop runs at the base rate of the simulation, which is the greatest common divisor of all the discrete time steps in the simulation. For example, if a multirate simulation contains two discrete tasks, one with a time step of 100 microseconds and one with a time step of 500 microseconds, the base rate is 100 microseconds. The Scheduler Loop then uses 100 microseconds as the time step of the simulation. The Simulation Interface Toolkit uses rate-monotonic scheduling to prioritize different model tasks in the same model. Rate-monotonic scheduling gives higher priority to model tasks that have shorter time steps. A higher-priority task can interrupt a lower-priority task.

II.3 Requirements and Installation

Before the connection of Matlab with LabView, all programs must be installed correctly. Programs that have to be installed are:

- LabVIEW.
- Simulation Interface Toolkit (SIT).
- Microsoft Visual studio
- LabVIEW Real-Time Module.
- LabVIEW FPGA Module (Optional).
- LabVIEW FPGA Compile Farm (Optional).
- NI LabVIEW FPGA Xilinx
- RIO drivers

refer to appendix A for more details about setting up and initialization of the Host Computer and Real-Time Target and versions of programs used in this project

II.4 Configuring a Simulation on a Real-Time Target

[10] The LabVIEW Simulation Interface Toolkit supports hardware-in-the-loop (HIL) testing using supported National Instruments RT Series hardware [6] Note You also can execute a model DLL on a Windows computer[7].

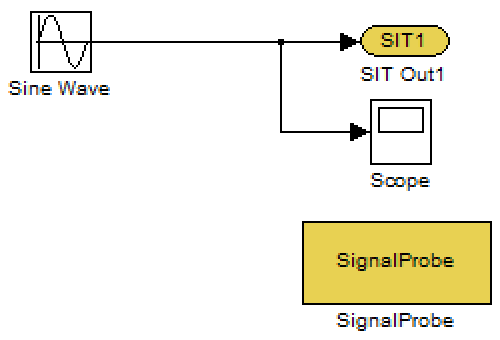
II.4.1 Configuring a simulation on a real-time (RT) target involves the following steps:

1. Build a model. **Note**: In order to communicate with RT Series hardware, the model must have at least one input port and one output port.
2. Create a host VI with controls and indicators.
3. Convert the model to a model DLL.
4. Specify the model, model DLL, execution host and mapping HW I/O: This step may involve the creation of Custom FPGA Bitfile which defines the analog, digital, and pulse width modulation (PWM) inputs and outputs of the FPGA device.
5. Create a driver VI.
6. Run the simulation.

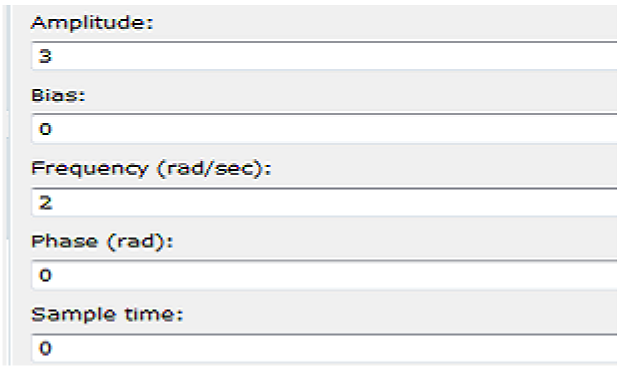
Refer to appendix A, for a detailed step by step guide on how to create a model and simulate it on real-time (RT) target.

II.5 Simple case study "sine wave model"

- Consider the Simulink model in Figure 2.6:



(a) Sinewave model



Amplitude:
3
Bias:
0
Frequency (rad/sec):
2
Phase (rad):
0
Sample time:
0

(b) Sinewave parameters

Figure 2.6: Simulink model

- Configure the communication between Simulink and SIT:[11] [12]

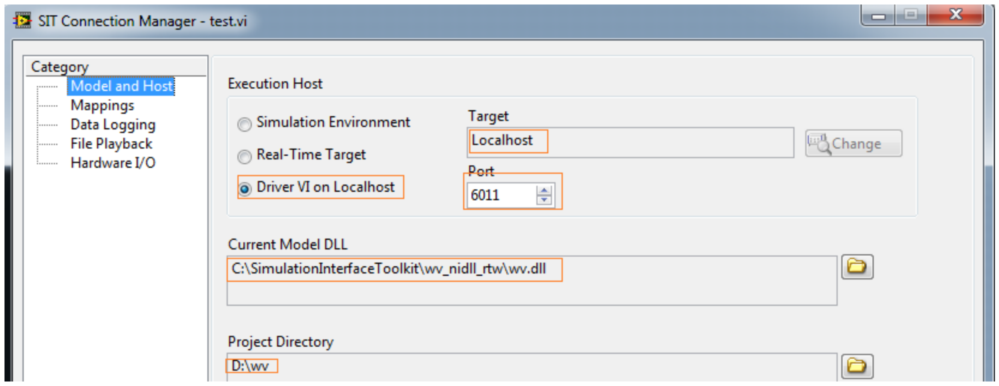
  This part involves the creation of DLL file with appropriate settings for both the **Solver** and **Real Time Workshop** . After all the following message is displayed in the MATLAB command window, which indicates that Real-Time Workshop has completed building the model DLL.

```
###Successful completion of Real−Time Workshop build procedure for model:ModelName
```

- Creating the Front Panel of the Host VI: The front panel of the host VI.
- Executing sine wave model DLL on a Windows Computer:

  Follow steps in apppendix A to execute a model DLL on a Windows computer:

  1. In the **Model and Host section** from **SIT Connection Manager** dialog box choose the commendable settings as in Figure 2.7



SIT Connection Manager - test.vi

Category: Model and Host, Mappings, Data Logging, File Playback, Hardware I/O

Execution Host: Simulation Environment; Real-Time Target; [✓ Driver VI on Localhost]

Target: Localhost

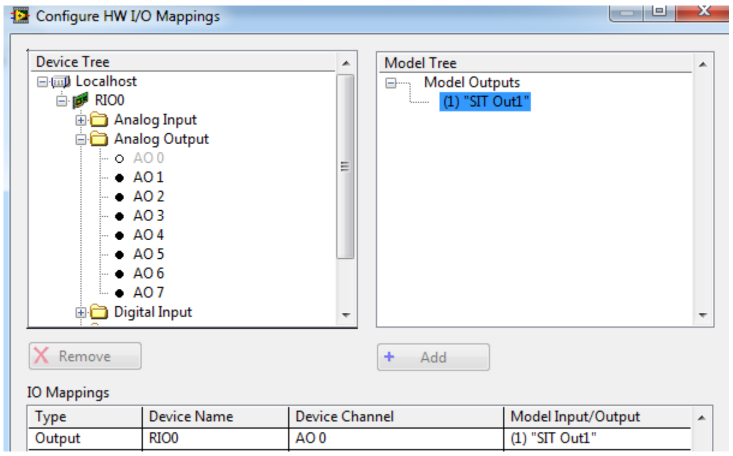
Port: 6011

Current Model DLL: C:\SimulationInterfaceToolkit\wv_nidll_rtw\wv.dll

Project Directory: D:\wv

Figure 2.7: Set fixed-step as a type of sampling

  2. In **Mappings** map indicator **Sine Wave** to **Sum1** parameter.
  3. In the **Configure HW I/O** dialog box, map between the FPGA target and a model DLL by adding the NI-FPGA and specifying its bitfile which will allow you to map HW I/O.
  4. In the **Hardware I/O** tab map **HW I/O** correctly Figure 2.8



Configure HW I/O Mappings

Device Tree: Localhost > RIO0 > Analog Input; Analog Output (AO 0, AO 1, AO 2, AO 3, AO 4, AO 5, AO 6, AO 7); Digital Input

Model Tree: Model Outputs > (1) "SIT Out1"

IO Mappings

Type	Device Name	Device Channel	Model Input/Output
Output	RIO0	AO 0	(1) "SIT Out1"

Figure 2.8: Add IO Mapping

The LabVIEW Simulation Interface Toolkit places the driver VI, `<ModelName>\_driver.vi`, in the same directory as the model DLL you specified.

5. Close the simulation environment or manually stop the SIT Server by echoing the following command NISITServer('stop') in Matlab command line; also, ensure that no other driver VIs are running on the Windows computer. If you do not stop the SIT Server before continuing, LabVIEW returns an error when you try to run the VIs.
6. After running the driver VI and the host VI, the generated sinewave signal is shown in Figure 2.9.

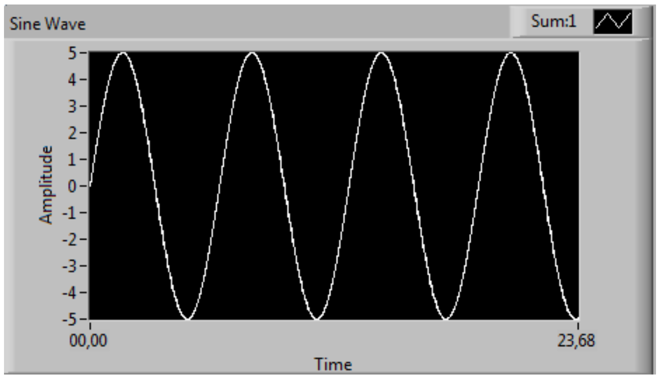


Figure 2.9: The generated sinewave signal after running the host VI

7. From the oscilloscope Figure 2.10.

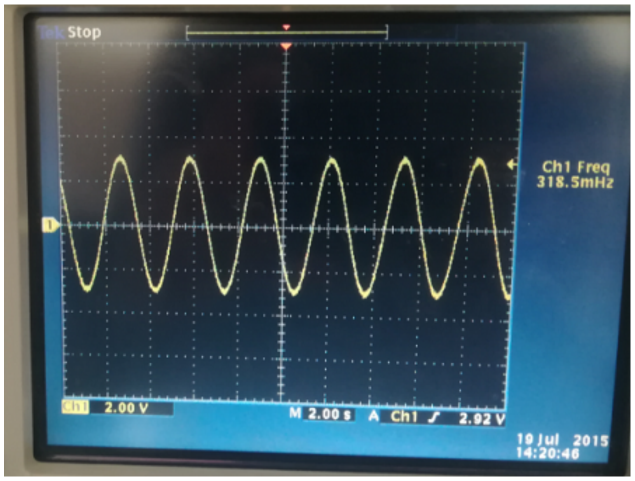


Figure 2.10: Sinewave in oscilloscope

Chapter 3

Working With NI Veristand

III Chapter III: Working With NI Veristand

III.1 What is NI Veristand?

You can use LabVIEW and VeriStand to run compiled models (DLLs) created using MathWorks Simulink and Simulink CoderTM. The models generally include fixed-step numerical solvers from the MathWorks. A fixed-step solver (ode 1) was chosen in Simulink so that the models can also run on real-time targets.

This interfacing approach between compiled models from Simulink and LabVIEW or VeriStand can be used to implement algorithms, filters, control systems, estimators, and real-time models in your applications using NI hardware [13].

The NI VeriStand is a software environment for configuring real-time testing applications. It provides a powerful framework for embedded software validation and real-time control and monitoring of mechanical and electrical test applications.
NI VeriStand can also import control algorithms, simulation models, and other tasks from NI LabVIEW software and third-party environments. It can be used to interact with models from a variety of modeling environments and programming languages. It can run compiled models created in any supported modeling environment as well as uncompiled models (*.mdl files) created using MATLAB/Simulink software [14].

NI Veristand has the following features [15]:

- Direct Model Integration with Matlab, Simulink, Python and .DLL
- Test Sequence Builder
- Runtime GUI Editing
- Sensor Simulation with FPGA
- Extensive Utilities:
  - Calibration
  - Error Reporting
  - Channel Configuration
  - Test Reporting
- Vehicle Bus Communications
- Real-time Sequencing

III.2 Components of NI VeriStand Project

The following illustration and sections describe the roles and locations of the major components of an NI VeriStand project [16]. Some components operate internally in the system when you run a project. Other components are user-visible features you create and configure in the NI VeriStand environment as shown in Figure 3.1.

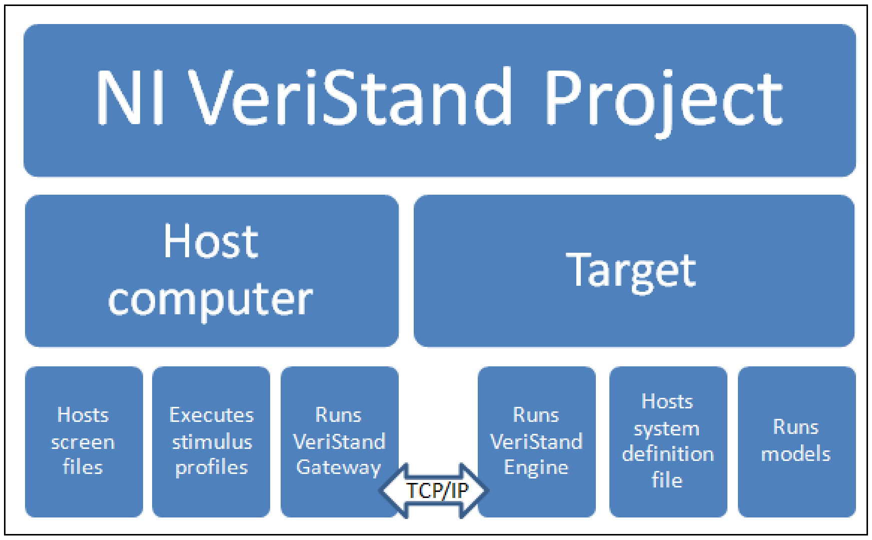


Figure 3.1: VeriStand Environment

- **Host Computer**

  A host computer hosts the screen files that serve as the user interface for operators. This computer also runs the VeriStand Gateway. The host computer must be a PC running a supported version of Windows.

  - **Internal Features**
    - **VeriStand Gateway:** Creates a TCP/IP communication channel that facilitates communication with the VeriStand Engine over the network. The VeriStand Gateway receives channel values from the VeriStand Engine and stores these values in a table that can be viewed using the **Channel Data Viewer**, available in the **Tools** menu of the **Workspace** window.

      If you run a project on a desktop PC, the VeriStand Gateway initiates the VeriStand Engine. If you run a project on an RT target, the VeriStand Gateway synchronizes with the system definition file that is running on the RT target. If the system definition file currently running on the VeriStand Engine does not match the system definition file that the VeriStand Gateway expects, then the VeriStand Gateway does not synchronize with the system definition file running on the RT target.
    - **NI VeriStand Model Simulation Server:** If your system includes an uncompiled Simulink model, this server uses a TCP/IP connection to transmit data between NI VeriStand and the uncompiled model. By default, the Model Simulation Server starts automatically when you launch the Matlab software and runs on port 6012. However, if this server does not automatically start, you can manually launch the Model Simulation Server.
  - **Features You Interact With**

    * **Project File:** The `.nivsproj` file that defines high-level settings, such as:
        - The screen and system definition files to run
        - The list of tools you can launch from the **Tools** menu of the Workspace window
        - Which services run when you deploy a project to the target
        - The IP address of the VeriStand Gateway
        - Stimulus profiles and real-time sequences
    * **Screen File:** Defines the configuration and settings for the screens and display items you view in the **Workspace** window.
    * **Stimulus profile:** A test executive that can call real-time sequences, open and close NI VeriStand projects, and perform data-logging and pass/fail analysis. It also connects real-time sequences to system definition files to bind channel data within the system definition file to variables in the real-time sequence. Stimulus profiles execute on the host computer. You create and run a stimulus profile using the Stimulus Profile Editor.
    * **National Instruments Driver Software:** You need the appropriate National Instruments driver software to communicate with hardware installed on a target.

- **Deployment Target**

  The target in an NI VeriStand system is a desktop PC or RT target on which you run the system definition file and VeriStand Engine.

  - **Internal Features**
    * **VeriStand Engine:** The non-visible execution mechanism that controls the timing of the entire system as well as the communication between the target and the host computer. It consists of multiple timed loops that use RT FIFOs to transfer data between the loops.
  - **Features You Interact With**
    * **System Definition File:** The `.nivssdf` file you configure in the System Explorer window. A system definition file contains the configuration settings of the VeriStand Engine, including:
        - The rate at which the system runs.
        - For each device the task and channel configurations for each.
        - Simulation models to execute, and the rate at which they execute.
        - The list of active alarms. You can use alarms to trigger actions on the target, such as procedures, or to display dialog boxes that alert the user of an event.
        - The list of procedures that can execute on the target. A procedure is a script of commands that define a set of actions in the VeriStand Engine.

        · The list of channels for data objects in the system. Channel types include:
        · Hardware I/O channels (DAQ, FPGA, etc.)
        · Model channels (inputs, outputs, parameters, signals)
        · User channels (used to store or map user-defined values in the system)
        · Calculated channels (channels that represent the result of a user-defined calculation of other channels in the system)
        · The system mappings that determine how channels are connected.
    ∗ **Model:** A mathematical representation of a real-world system. A model responds to stimuli by producing outputs in a way that emulates the behavior of the modeled item. Models contain inputs and outputs, called inports and outports, that communicate with other parts of the control system.
      You can build models using several different modeling environments, and then integrate the model into a system definition file.

III.3 Integrating a model to NI VeriStand

Models run on hardware targets and are typically used to respond to stimuli from other parts of the system by producing outputs in a way that simulates the modeled item. Models also can serve the functions of signal generation, signal analysis, and control.

Models can contain the following components that connect to other parts of the system or allow you to interact with the model:

- **Inports and Outports:** To communicate with other parts of the control system, models contain inputs and outputs, called inports and outports. You can map inports and outports directly to hardware inputs and outputs, other models in the system, system channels, and so on. Inports and outports are dynamic values the simulation updates each time the model executes.
- **Parameters:** Parameters act like variables in the model. Unlike inports, whose values come from elsewhere in the system and change frequently, users typically manipulate parameters infrequently to tune the behavior of the simulation.
- **Signals:** Signals serve as probes, or test points, of a model as it executes.

Consider a system that contains a physical motor controller and a model that represents a DC motor. The model runs on a hardware target. Such a model might contain the following components:

- An inport that accepts the motor command from the motor controller.
- An outport that returns the motor speed from the model.

- Parameters that adjust the load on the motor. You might set parameter values once per test rather than updating them frequently during the test.
- A signal that returns internal data that aids in debugging.

NI VeriStand uses direct memory access (DMA) FIFOs to transfer data between the host computer and FPGA target. The `DMA_READ FIFO` sends data read from the FPGA inputs to the host computer. The `DMA_WRITE FIFO` transfers data from the host computer to the FPGA outputs. The data is stored in packets that each can contain up to 64 bits. Values of different data types can packed together in the same packet. If a channel is added to the FPGA VI, it must be added to a packet that is written to the FIFO.[17].

Models can be built using several different modeling environments. Typically, a model must be compiled in the modeling environment before it can run on an RT target[18]. In this project the process of preparing a model in Simulink is illustrated in details in the Appendices.

The process for preparing to add a Simulink `.mdl` to a system definition file depends on the target you want to run the model:

- **RT target:** NI VeriStand requires that a Simulink model must be compiled into a DLL or .out file.
- **Desktop PC:** On Windows computers with MATLAB and Simulink installed, the model can be compiled or remain uncompiled. You can add uncompiled models directly to a system definition.

Note: If your system definition file contains compiled models created in the Simulink software, NI VeriStand does not use the Model Simulation Server because compiled models execute in the VeriStand Engine [13].

Running uncompiled models on a desktop computer can be useful when you perform rapid model changes that require you to switch between Simulink and NI VeriStand frequently, such as during model-in-the-loop testing.

In order to convert a model, Simulink performs the following process (as seen in Figure 3.2) [19]:

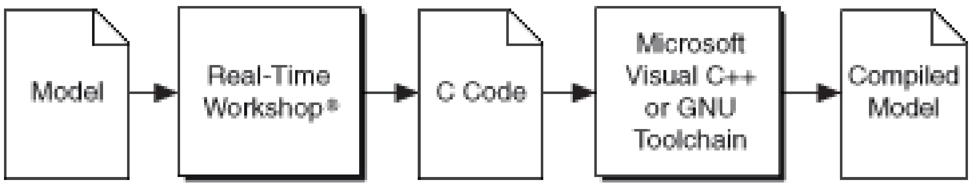


Figure 3.2: Conversion Process for Models from Simulink

- The Real-Time Workshop tool converts the model and any submodels into a `C/C++` code version of the same model.
- A compiler, Microsoft Visual `C++` or the `Wind River GNU Toolchain`, compiles the `C/C++` code model into a .dll or .out files.

Chapter 4

Case Study "Three Phase Fault"

IV Chapter IV: Case Study "Three Phase Fault"

In this chapter, a three phase fault model is implemented using both SIT and VeriStand methods. Results of both approaches are then compared.

An abnormal electric current is referred to as a fault current. Typical causes of a fault include lightning strikes, insulation contamination, equipment insulation failure, and animal spanning two lines. The two types of faults are unsymmetrical faults and symmetrical faults. A symmetrical fault is also known as a balanced fault and is divided into two types, which are line-to-line-to-line-to ground (L-L-L-G) and line-to-line-to-line (L-L-L). These faults may cause thermal damage to the equipment. The unsymmetrical faults are more common and less severe. It consists of single line-to-ground, line-to-line, double line-to-ground, and balanced three phase faults [20]. An example of a fault scenario of a three-phase system is displayed in Figure 4.1 and implemented in MATLAB/Simulink. The system outputs 3-phase current in a scope, a fault occurs at 10 seconds that stops normal system operation [20].

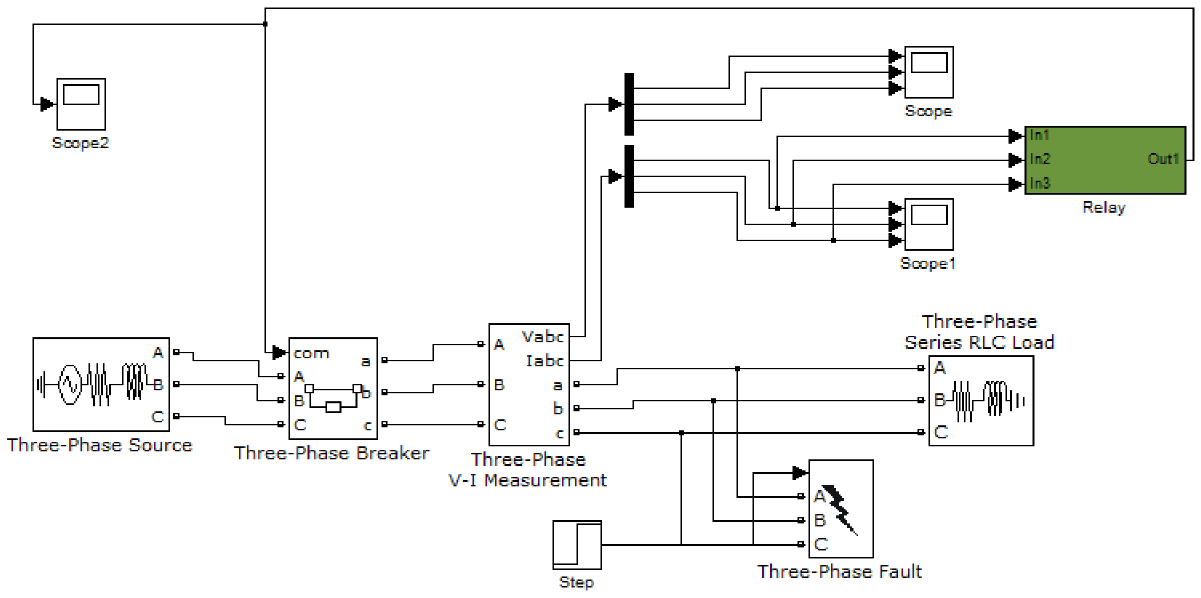


Figure 4.1: Phase system Fault analysis model

The internal design of the relay used in the above circuit is shown in Figure 4.2. This relay detects overcurrent using Relation Operator that compares the actual current to a maximum value 30 kA. When current exceed the maximum value the SR Flip-Flop generate a set value hence $\overline{Q} = 0$ that turned the AND logic result signal to zero 'False'. This is supplied to COM terminal of the Three-Phase Breaker which stops normal system operation.

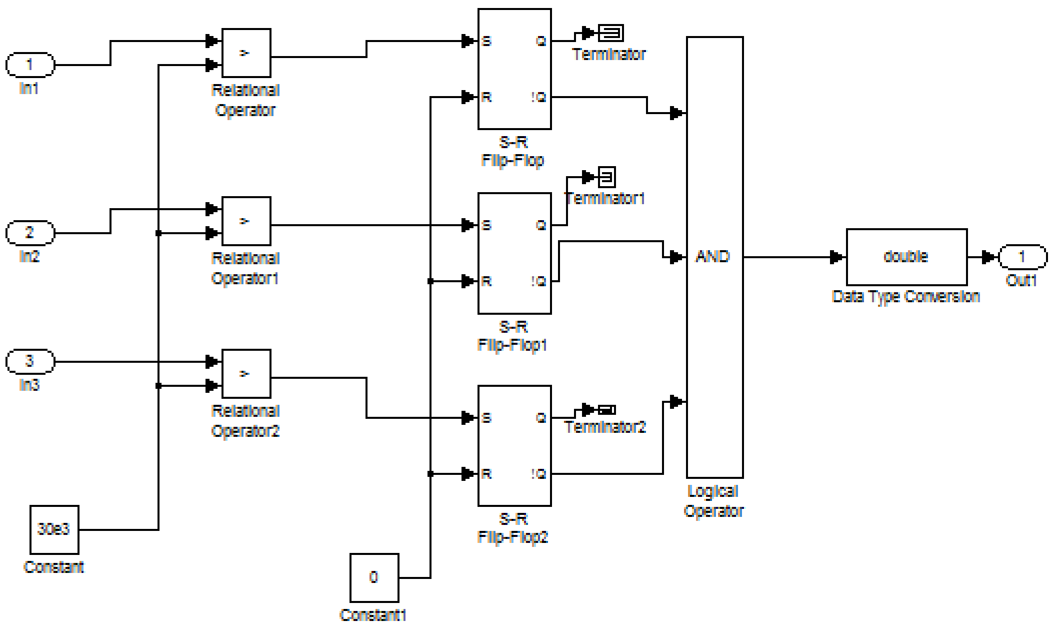


Figure 4.2: Relay block model

The current is captured during a fault from simulink environment using a scope as demonstrated in Figure 4.3.

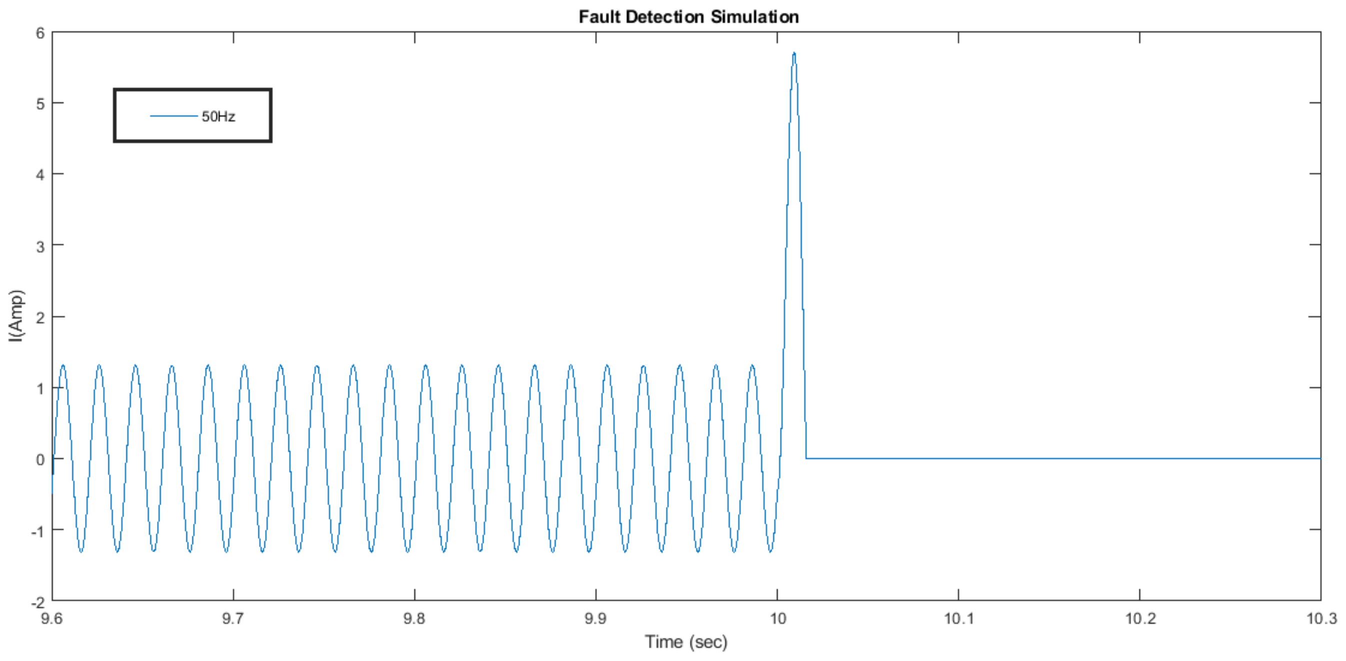


Figure 4.3: Fault Detection Simulation

To test the three phase fault model, the connections shown in Figure 4.4 were made in order to observe the circuit outputs.

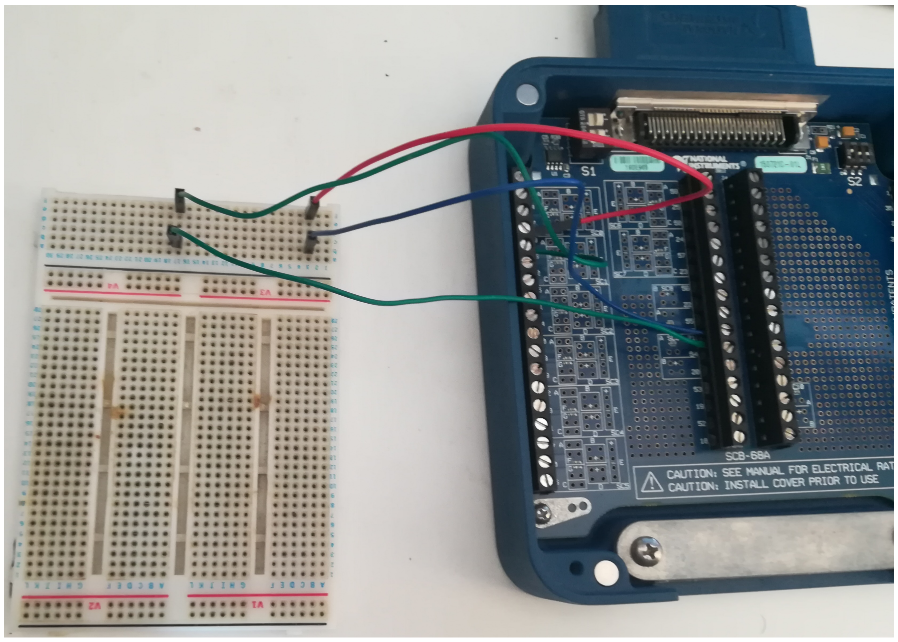


Figure 4.4: Connections implemented to test the model

`AO1(pin 54)` is connected back to `AI1(pin 66)`, the ground related to both are connected to each other(pin 20 is connected to pin 33). Now it's time to test the model in both environments SIT and VeriStand then demonstrate the major differences between the two approaches and state the difficulties when performing each of them.

IV.1 Using SIT

In Figure 4.5, The Simulink model is redesigned so it can communicate with SIT.Also, the current is lowered to a value that PCIe-7851R can generate. The model sends the output current value through `SIT Out1` block to LabVIEW VI or hardware(`AO1` pin) , then the output current is feeded back to `AI1` to the model through `SIT In 1`, the `SIT Out3` is used to see the difference between the supplied output current and feedback input current, `SIT Out4` is used to see the current error rate(relative error) which is giving in the following equation:

$$RelativeError = (OutputCurrent - InputCurrent)/InputCurrent$$

.

The steps demonstrated in Appendix A and chapter 2 are followed here in order to configure a simulation on a Real-Time Target.

- Create the host VI with needed controls and indicators.
- Convert the model to a model DLL.

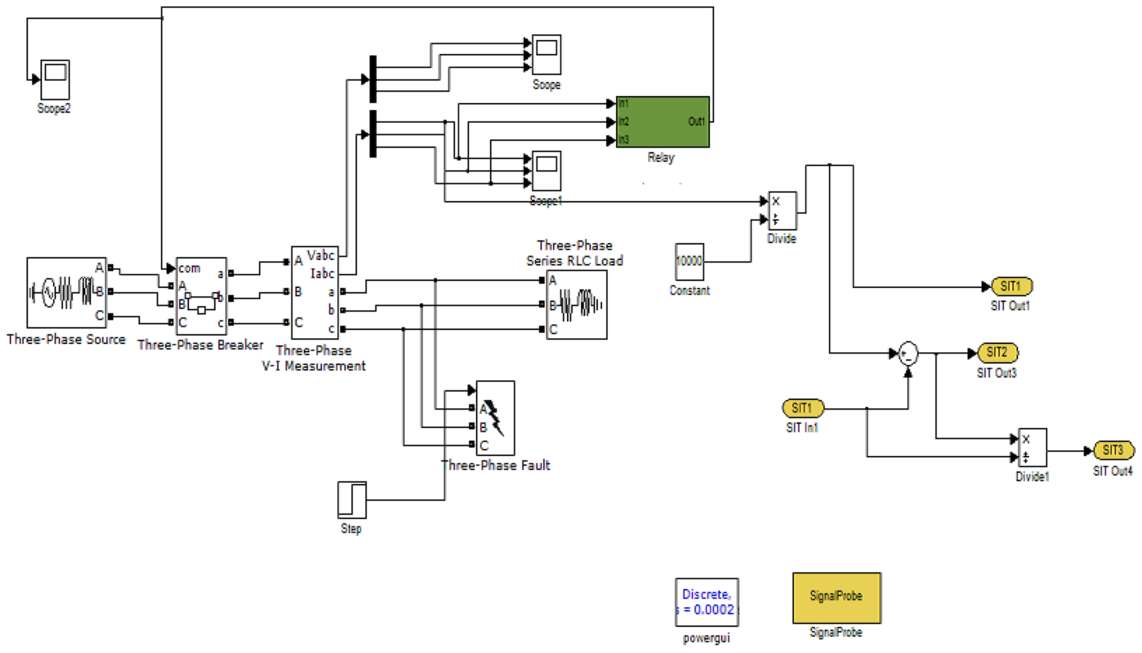
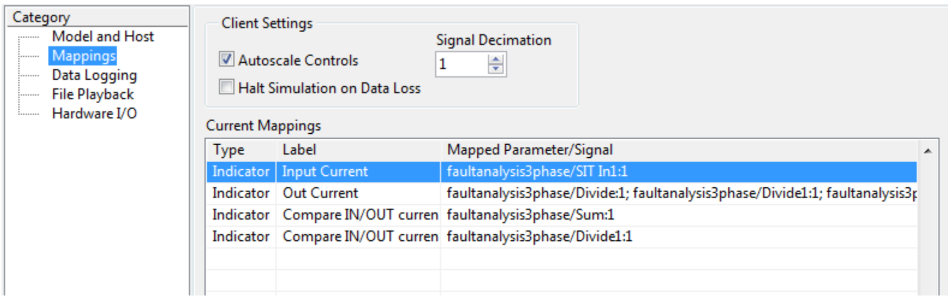


Figure 4.5: Simulink model with SIT I/O

- Specify the model DLL, execution host and mapping HW I/O.
  - Mappings are shown in Figure 4.6.



Signal Decimation: 1

[✓] Autoscale Controls

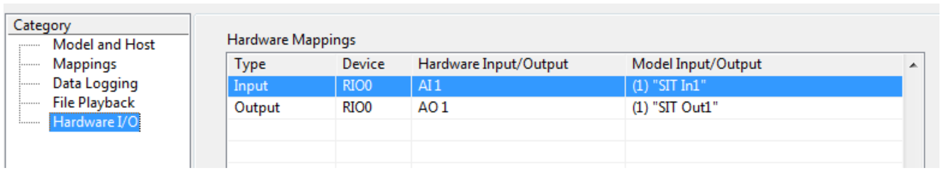
[ ] Halt Simulation on Data Loss

Current Mappings

Type	Label	Mapped Parameter/Signal
Indicator	Input Current	faultanalysis3phase/SIT In1:1
Indicator	Out Current	faultanalysis3phase/Divide:1; faultanalysis3phase/Divide1:1; faultanalysis3p
Indicator	Compare IN/OUT curren	faultanalysis3phase/Sum:1
Indicator	Compare IN/OUT curren	faultanalysis3phase/Divide1:1

Figure 4.6: Three phase fault Mappings

  - Hardware I/O mappings are shown in Figure 4.7 below.



Hardware Mappings

Type	Device	Hardware Input/Output	Model Input/Output
Input	RIO0	AI 1	(1) "SIT In1"
Output	RIO0	AO 1	(1) "SIT Out1"

Figure 4.7: Three phase fault Hardware Mappings

- Create the driver VI.
- Run the simulation.

Obtained results:

- In the Host VI (see Figure 4.8).

  From the first graph in the top left corner of Figure 4.8a entitled “In, Out, Relative error” ,it can be noticed that there is a small delay between the supplied Output ’Out:1’ and the read Input ’In:1’ and this difference is shown in “Compare IN/OUT current (difference)” graph in the top right corner. From the “Relative error” graph in the bottom left corner, it can be seen that the relative error is approximately zero except when the current’s amplitude is close to zero, this is due to delay and division by small factor.

  From Figure 4.8b, we can see that the fault occurs exactly at the time set in the step function 10 seconds into the simulation. In this fault, an abnormal overcurrent is generated to which the relay reacts by sending a stop signal to the Three-Phase Breaker. The latter stops the normal operation of the system. From this demonstration, it can be seen that signals are generated in real-time using simulation.

- In the Oscilloscope Figure 4.9.

  from Figure 4.9a it is noticed that the frequency of the Current signal is approximately 50 Hz which tends to be the frequency of the three phase signal hence the properties of the signal is preserved. Figure 4.9b shows when the overcurrent fault occurs and the system is finally stopped.

  It can be seen that the results shown by the graphs in LabView Host VIs and the oscilloscope are the same hence the interface technique works properly.

IV.2 Using VeriStand:

First, we need to apply the needed changes explained in the Appendix B to the Three Phase Fault Simulink model , so it can communicate with NI VeriStand as shown in Figure 4.10.

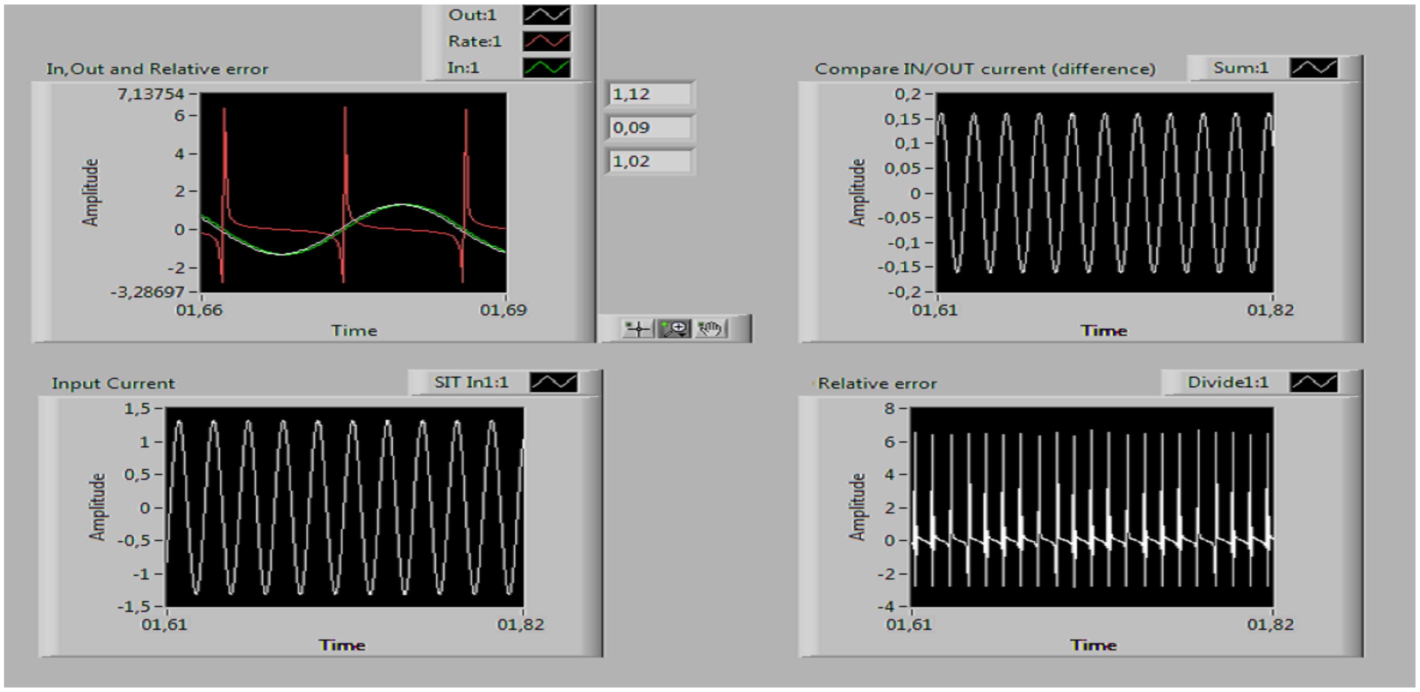
As with SIT, the current is lowered to a value that NI PCIe-7851R can generate and receive later on.

NI PCIe-7851R `AO1` pin is mapped to `NIVeriStand Out` Block, so it can output the divided current through Analog Output `AO1` of the NI PCIe-7851R. Also `AO1` is connected physically back to `AI1` which is mapped to `NIVeriStand In1`, to receive the output current that have been generated from the model itself.

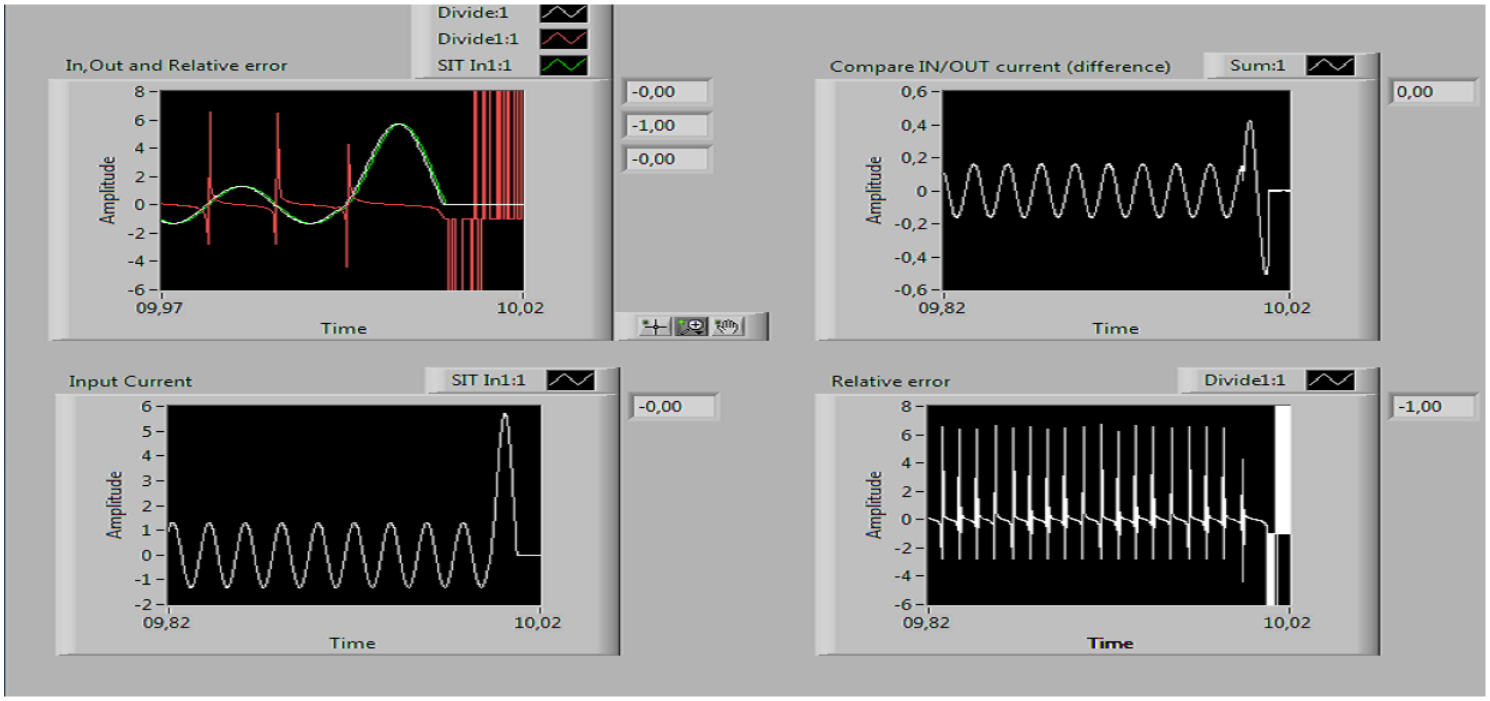
`NIVeriStand Out2` is used to see the difference between the actual current that have been outputted and the input current.

`NIVeriStand Out3` is used to see the error rate(relative error) described by the following equation:

$$ErrorRate = (OutputCurrent - InputCurrent)/InputCurrent$$



(a) Before Fault

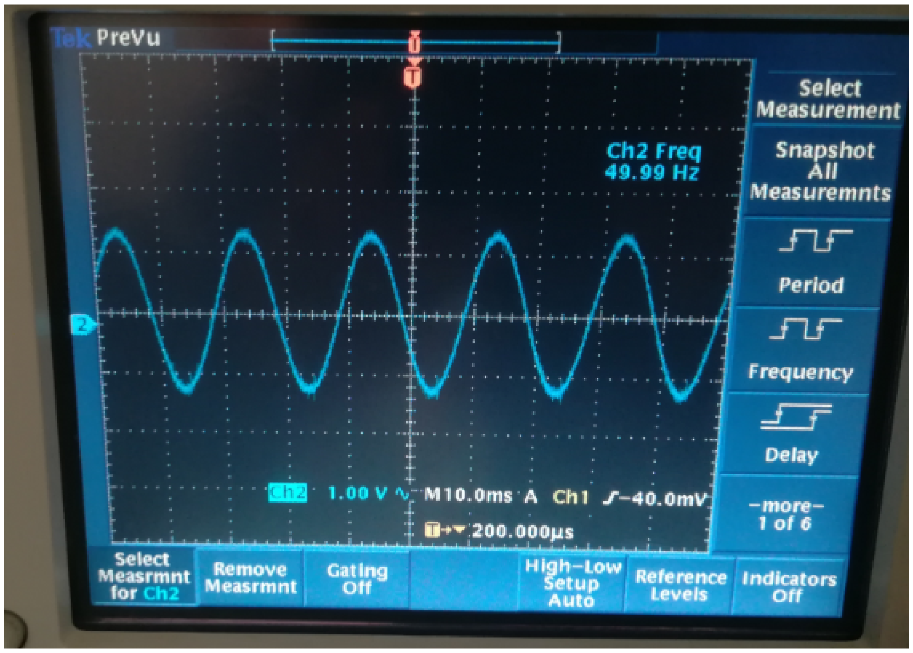


(b) During fault

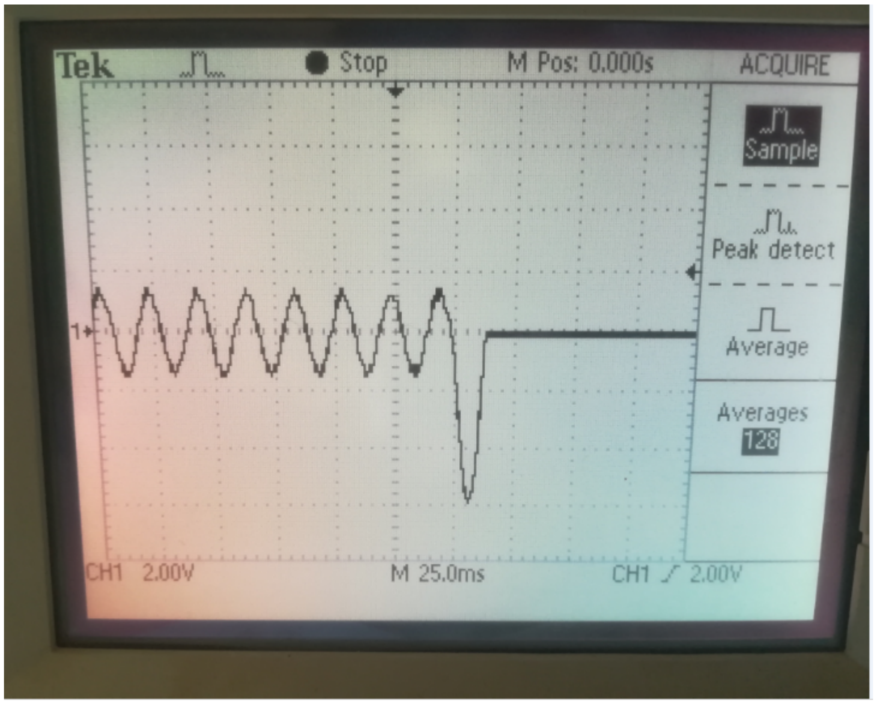
Figure 4.8: Fault occurrence as seen in LabView

. Steps in Appendix B must be followed to:

- Compile the model to DLL.



(a) Before Fault



(b) During fault

Figure 4.9: Fault occurrence as seen in an oscilloscope

- Configure the System Definition file with the specific hardware,model and their mappings which should be similar to that in Figure 4.11.
- Run and Deploy the project to NI device

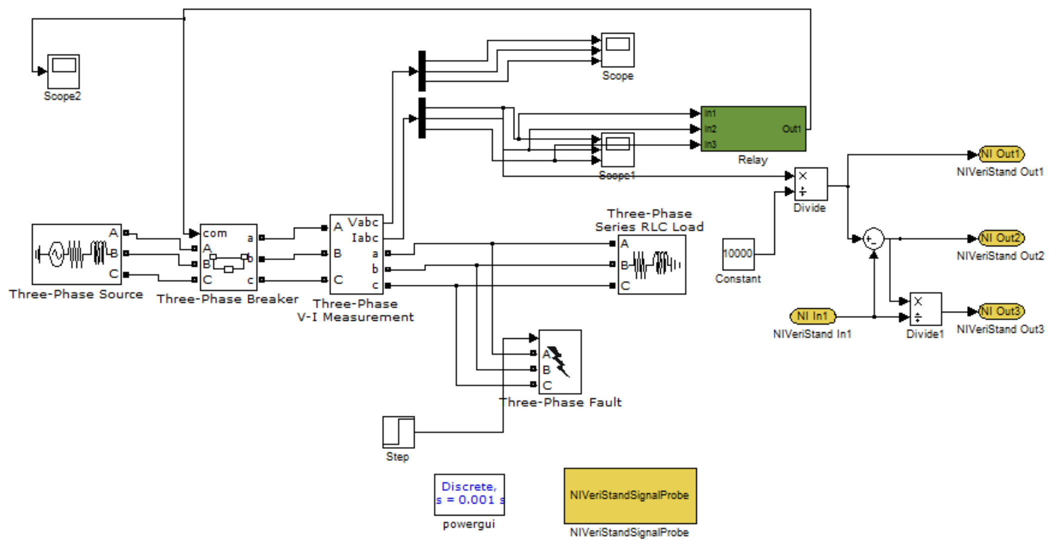
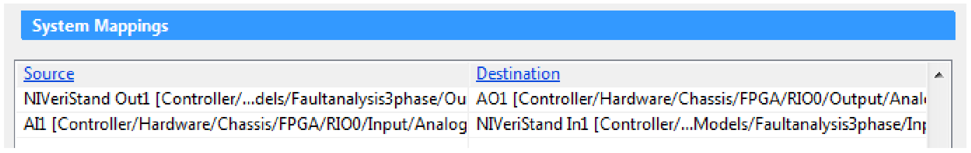


Figure 4.10: Simulink model with VeriStand I/O


System Mappings

Source	Destination
NIVeriStand Out1 [Controller/...dels/Faultanalysis3phase/Ou	AO1 [Controller/Hardware/Chassis/FPGA/RIO0/Output/Anal
AI1 [Controller/Hardware/Chassis/FPGA/RIO0/Input/Analog	NIVeriStand In1 [Controller/...Models/Faultanalysis3phase/Inp

Figure 4.11: Three phase fault mappings in NI VeriStand

- Customize the WorkSpace.

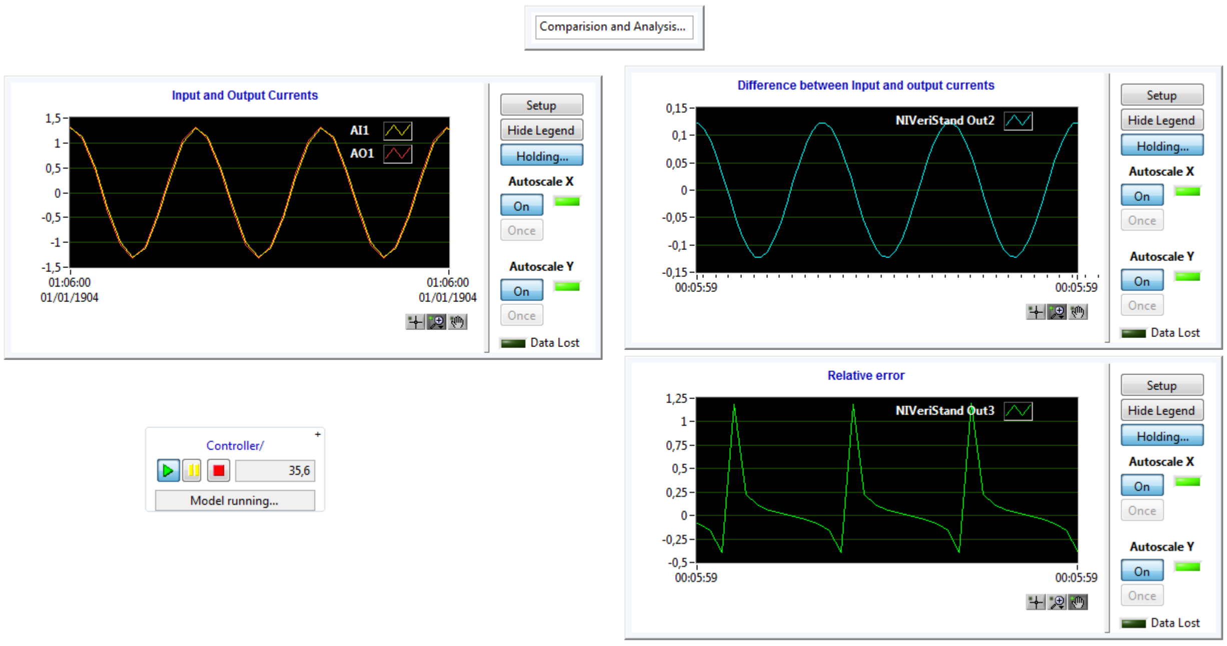
Obtained Results:

- In the WorkSpace (see Figure 4.12).

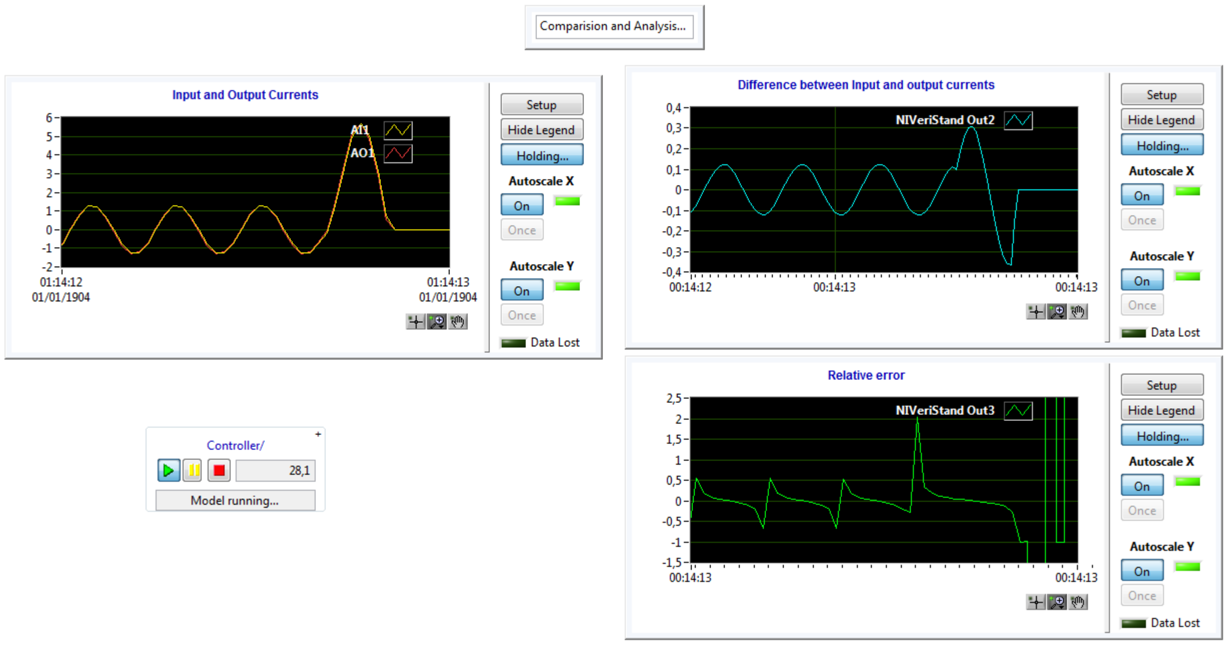
  In Figure 4.12a, the tiny phase shift between the input and output currents in **the Input and Output Currents graph** takes place due to the sampling rate chosen and the latency accumulated because of the signal propagation from DACs,FPGA Logic,ADCs and the wire used to connect `AO1` to `AI1`. The very little small difference between the input and output currents noticed in **Difference between input and output currents graph** in the right upper corner of Figure 4.12a occurs because of the delay between them caused by the reasons stated before. The relative error supports what is stated previously. In Figure 4.12b, The fault happens 10s after the execution of the Simulation then the circuit breaker blocks the normal work of the power system.

- In the Oscilloscope (see Figure 4.13).

  In Figure 4.13a, the signal seen is identical to that in WorkSpace of VeriStand. The frequency is approximately 50 Hz which is the frequency that characterizes power systems. In Figure 4.13b, the Oscilloscope demonstrate the signal when the fault is happening.



(a) Before Fault

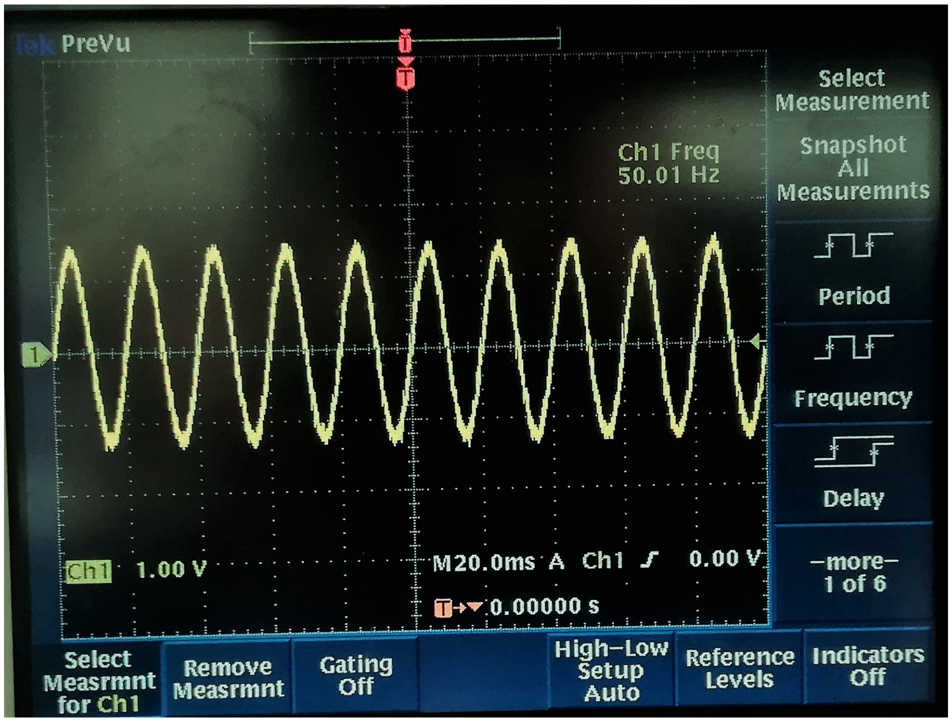


(b) During fault

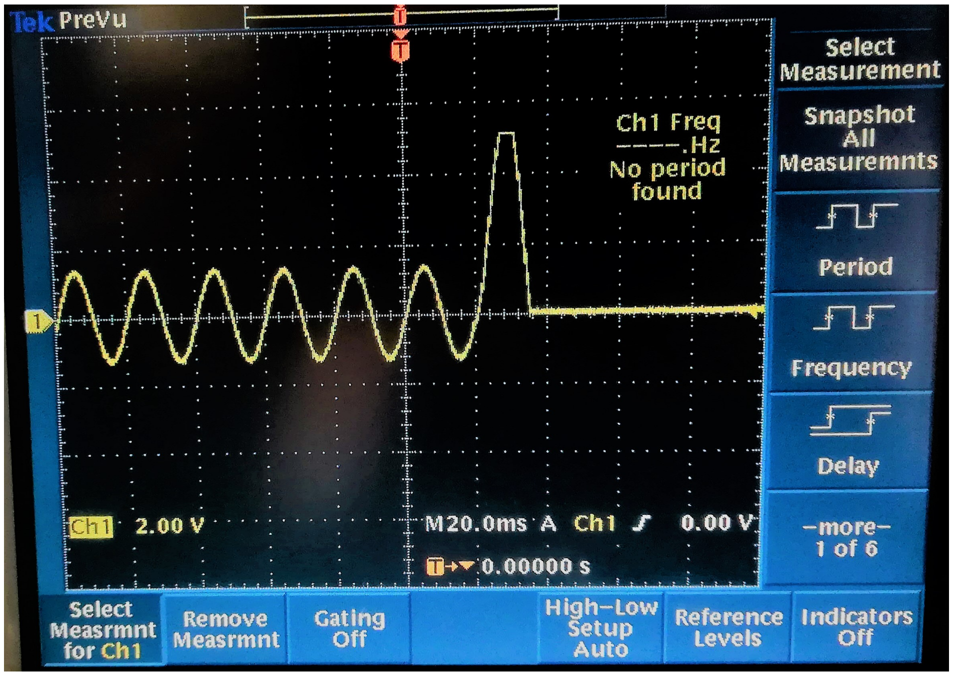
Figure 4.12: Fault Detection in WorkSpace

IV.3 Comparison and Discussion:

The core difference between the Simulation Interface Toolkit and NI VeriStand is that NI VeriStand is a configuration based tool that allows you to import simulation models, DAQ/FPGA devices, etc. and map everything together out of the box. The Simulation Interface Toolkit requires you to build LabVIEW code around your sim-



(a) Before Fault



(b) During fault

Figure 4.13: Fault Detection in Oscilloscope

ulation model to perform additional functions. NI VeriStand is preferred, because applications that involve simulation are typically quite complex, and NI VeriStand can significantly reduce software development time.

In addition, Veristand is more flexible and allow for managing and editing Workspace during execution

Moreover, Veristand provides the ability for freezing the system through the rate target frequency which allow for deep analysis and control.

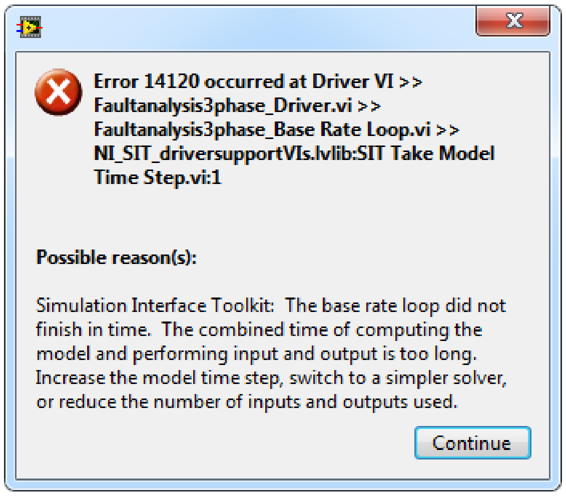
Furthermore, Simulation interface toolkit has a lot of limitation by supporting only 32-bit Operating System and it is not anymore developed and it has been replaced by Model Interface Toolkit which is integrated as part of Veristand.

Besides, Veristand is very simple and easy to use however SIT is very complex and crashes a lot.

IV.4 Difficulties

Difficulties faced in this study:

- Sampling rate: should be selected approperly hence it can freeze the system the system when going above certain value of frequencies for Veristand and the signal can be lost if small value of frequency are selected for both techniques
- SIT toolkit was unable to load and integerating the bitfile into the labview diagram for hardware initialization and I/O definition so it is adjusted manually.
- SIT crashes:
  - Base Rate loop crash 4.14: due to very small sampling time and a the number of probed signals which causes rate to miss data and not finish in time.

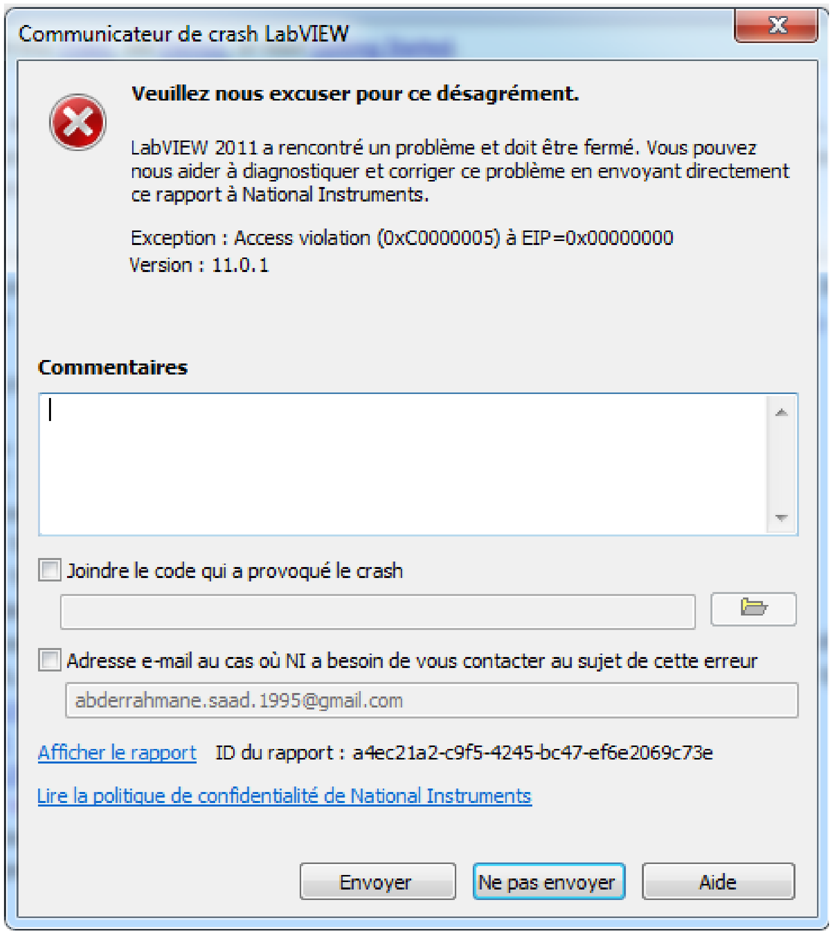

Error 14120 occurred at Driver VI >> Faultanalysis3phase_Driver.vi >> Faultanalysis3phase_Base Rate Loop.vi >> NI_SIT_driversupportVIs.lvlib:SIT Take Model Time Step.vi:1

Possible reason(s):

Simulation Interface Toolkit: The base rate loop did not finish in time. The combined time of computing the model and performing input and output is too long. Increase the model time step, switch to a simpler solver, or reduce the number of inputs and outputs used.

Figure 4.14: Base Rate loop crash

  - Access violation crach 4.15 due to bad memory allocation which cause overriding EIP Extended Instruction Pointer register.


Communicateur de crash LabVIEW

Veuillez nous excuser pour ce désagrément.

LabVIEW 2011 a rencontré un problème et doit être fermé. Vous pouvez nous aider à diagnostiquer et corriger ce problème en envoyant directement ce rapport à National Instruments.

Exception : Access violation (0xC0000005) à EIP=0x00000000
Version : 11.0.1

Commentaires

☐ Joindre le code qui a provoqué le crash

☐ Adresse e-mail au cas où NI a besoin de vous contacter au sujet de cette erreur

abderrahmane.saad.1995@gmail.com

Afficher le rapport ID du rapport : a4ec21a2-c9f5-4245-bc47-ef6e2069c73e

Lire la politique de confidentialité de National Instruments

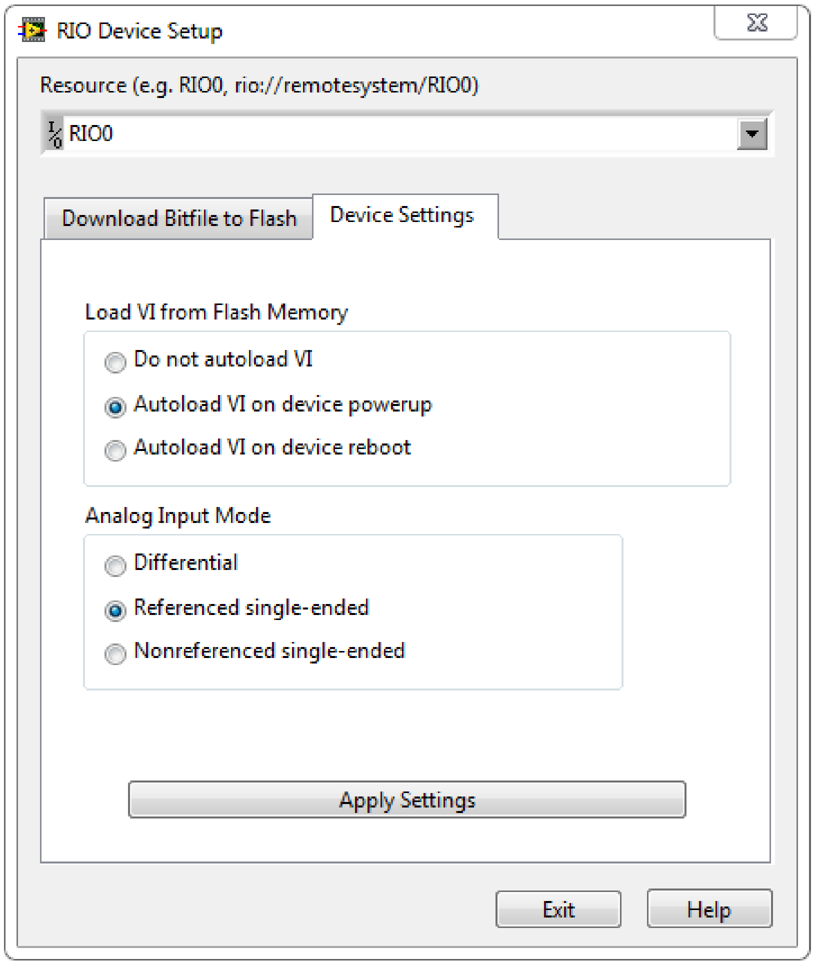
Figure 4.15: Access violation crach

- Analog input mode (see Figure 4.16):
  Ufortunately, measuring analog signals with a data acquisition device is not always as simple as wiring the signal source leads to the data acquisition device. Knowledge of the nature of the signal source, a suitable configuration of the data acquisition device. A voltage source can be grouped into one of two categories-grounded or ungrounded (floating). Similarly, a measurement system can be grouped into one of two categories-grounded or ground-referenced, and ungrounded (floating) [21].

  – Grounded or Ground-Referenced Signal Source: a grounded source is one in which the voltage signal is referenced to the building system ground.

  – Ungrounded or Nonreferenced (Floating) Signal Source: a floating source is a source in which the voltage signal is not referred to an absolute reference, such as earth or building ground.

  – Differential or Nonreferenced Measurement System:A differential, or nonreferenced, measurement system has neither of its inputs tied to a fixed reference such as earth or building ground.


RIO Device Setup

Resource (e.g. RIO0, rio://remotesystem/RIO0)

RIO0

Load VI from Flash Memory

- Do not autoload VI
- [✓ Autoload VI on device powerup]
- Autoload VI on device reboot

Analog Input Mode

- Differential
- [✓ Referenced single-ended]
- Nonreferenced single-ended

Figure 4.16: Analog input mode

- Unwanted results sometimes take place if the complexity of the model is increased mainly with the sampling rate which should be decreased if the complexity increased to obtain accurate results.

Conclusion

In this project, a Fault Detection Simulink model was tested using two different techniques: “Simulation Interface toolkit and Veristand”. The two techniques allowed for Simulink models to access NI 7851R FPGA-based card without using LabView. the The first chapter provides a background about the NI 7851R series and an introduction to different simulation environments with their properties. Whereas, the second and third chapters discuss the working principle of the two used techniques. The case study of Fault Detection was demonstrated in the last chapter by deploying its model using the two methods and compare their results with that in Simulink scope. Moreover, the differences between the two techniques are stated and discussed. The results obtained show the importance of Hardware in the loop testing for real time simulation and give preference to Veristand over Simulation Interface Toolkit because VeriStand is more flexible, reliable and easy to use compared with SIT for deployment of this kind of simulation. As further work, it is suggested to extend these techniques, deploy more complex systems to test their limitations and try with a newer versions of software that will provide more properties and features. In addition, These two techniques could be tested on a real-time target such as CompactRIO targets. These approaches allow model execution to happen on the target, not on the host computer.

References

[1] MathWorks, "What is hardware-in-the-loop simulation?." `https://www.mathworks.com/help/physmod/simscape/ug/what-is-hardware-in-the-loop-simulation.html`. Accessed on 2019-06-11.

[2] add2 Limited: Embedded Systems Testing Development, "Model-in-the-loop testing applications." `https://www.add2.co.uk/applications/model-in-the-loop-testing-applications`, june 2018. Accessed on 2019-06-11.

[3] E. R. A. Guy, "What is hardware in the loop (hil) and software in the loop (sil) testing?." `https://www.electricrcaircraftguy.com/2018/06/hil-and-sil.html`. Accessed on 2019-06-11.

[4] N. Instruments, *NI R Series Multifunction RIO User Manual.* National Instruments, 2009.

[5] N. Instruments, "Components of a simulation (simulation interface toolkit)." `https://zone.ni.com/reference/en-XX/help/371504F-01/lvsitconcepts/sit_c_components_of_a_simulation/`, june 2010. Accessed on 2019-03-26.

[6] N. Instruments, "Supported national instruments real-time hardware (simulation interface toolkit)." `http://zone.ni.com/reference/en-XX/help/371504F-01/lvsitconcepts/sit_c_supported_ni_hardware/`, june 2010. Accessed on 2019-03-28.

[7] N. Instruments, "Executing a model dll on a windows computer (simulation interface toolkit)." `http://zone.ni.com/reference/en-XX/help/371504F-01/lvsithowto/sit_h_dll_windows/`, june 2010. Accessed on 2019-03-28.

[8] N. Instruments, "Running a simulation (simulation interface toolkit)." `https://zone.ni.com/reference/en-XX/help/371504F-01/lvsithowto/sit_h_running_a_simulation/`, june 2010. Accessed on 2019-03-28.

[9] N. Instruments, "Understanding the driver vi (simulation interface toolkit)." `https://zone.ni.com/reference/en-XX/help/371504F-01/lvsitconcepts/sit_c_understanding_the_driver_vi/`, june 2010. Accessed on 2019-03-28.

[10] N. Instruments, "Configuring a simulation on a real-time target (simulation interface toolkit)." http://zone.ni.com/reference/en-XX/help/371504F-01/lvsithowto/sit_h_configuring_sim_rt/, june 2010. Accessed on 2019-03-28.

[11] N. Instruments, "Converting a model into a model dll (simulation interface toolkit)." http://zone.ni.com/reference/en-XX/help/371504F-01/lvsithowto/sit_h_convert_model_to_dll/, june 2010. Accessed on 2019-06-11.

[12] N. Instruments, "Specifying the model dll and execution host (simulation interface toolkit)." http://zone.ni.com/reference/en-XX/help/371504F-01/lvsithowto/sit_h_specify_dll_and_host/, june 2010. Accessed on 2019-03-28.

[13] N. Instruments, "Running compiled simulink models in labview and in veristand." https://forums.ni.com/t5/Example-Programs/Running-Compiled-Simulink-Models-in-LabVIEW-and-in-VeriStand/ta-p/3889270?profile.language=en, April 2019. Accessed on 2019-05-28.

[14] F. M'zoughi, S. Bouallegue, and M. Ayadi, "Modeling and sil simulation of an oscillating water column for ocean energy conversion," in *IREC2015 The Sixth International Renewable Energy Congress*, pp. 1–6, IEEE, 2015.

[15] W. T. Incorporated, "Hardware-in-the-loop with veristand." https://www.winemantech.com/hubfs/Wineman_Tech_Dec2015/Pdf/VeriStand_Service_Sheet.pdf?hsLang=en-us. Accessed on 2019-05-28.

[16] N. Instruments, "Components of a project." http://zone.ni.com/reference/en-XX/help/372846M-01/veristand/comp_of_project/, May 2018. Accessed on 2019-06-02.

[17] N. Instruments, "Creating a custom fpga bitfile." http://zone.ni.com/reference/en-XX/help/372846M-01/veristandmerge/creating_custom_fpga/, May 2018. Accessed on 2019-06-02.

[18] N. Instruments, "Integrating and executing models." http://zone.ni.com/reference/en-XX/help/372846M-01/veristand/models_overview/, May 2018. Accessed on 2019-06-02.

[19] N. Instruments, "Conversion process for models from the mathworks, inc. simulink software (model interface toolkit)." http://zone.ni.com/reference/en-XX/help/374160C-01/vsmithelp/mit_model_compiling/, May 2017. Accessed on 2019-06-03.

[20] M. Peck, G. Alvarez, B. Coleman, H. Moradi, M. Forest, and V. Aalo, "Modeling and analysis of power line communications for application in smart grid," *arXiv preprint arXiv:1709.06883*, 2017.

[21] N. Instruments, "Field wiring and noise considerations for analog signals." http://www.ni.com/product-documentation/3344/en, august 2018. Accessed on 2019-05-10.

[22] S. ideas for a big world, "Configure matlab to use visual 2010 c/c++ compiler." http://smallideasforabigworld.blogspot.com/2011/03/configure-matlab-to-use-visual-2010-cc.html, March 2011. Accessed on 2019-05-20.

[23] N. Instruments, "Creating a custom fpga bitfile (simulation interface toolkit)." https://zone.ni.com/reference/en-XX/help/371504F-01/lvsithowto/sit_h_custfpga/, June 2010. Accessed on 2019-06-11.

[24] N. Instruments, "Rules for naming custom fpga bitfiles and vis (simulation interface toolkit)." https://zone.ni.com/reference/en-XX/help/371504F-01/lvsitconcepts/sit_c_custfpgarules/, June 2010. Accessed on 2019-06-11.

[25] N. Instruments, "Sit connection manager dialog box." https://zone.ni.com/reference/en-XX/help/371504F-01/lvsit/sit_connection_manager_db/, June 2010. Accessed on 2019-06-11.

[26] N. Instruments, "Creating mappings (simulation interface toolkit)." https://zone.ni.com/reference/en-XX/help/371504F-01/lvsitconcepts/sit_c_creating_mappings/, June 2010. Accessed on 2019-06-11.

[27] N. Instruments, "Ni-fpga property dialog box." http://zone.ni.com/reference/en-XX/help/371504F-01/lvsit/sit_config_fpga_wiz/, June 2010. Accessed on 2019-06-11.

[28] N. Instruments, "Real-time controllers and real-time operating system compatibility." http://www.ni.com/product-documentation/53636/en/?OpenDocument, May 2019. Accessed on 2019-06-11.

[29] N. Instruments, "Creating a custom fpga bitfile." zone.ni.com/reference/en-XX/help/372846M-01/verisNIVeriStandCreatingaCustomFPGABitfiletandmerge/creating_custom_fpga/, May 2018. Accessed on 2019-06-12.

[30] N. Instruments, "Creating a custom fpga configuration file." http://zone.ni.com/reference/en-XX/help/372846M-01/veristandmerge/creating_custom_fpga_configuration_file/, May 2018. Accessed on 2019-06-12.

Appendix A.Simulation Interface Toolkit

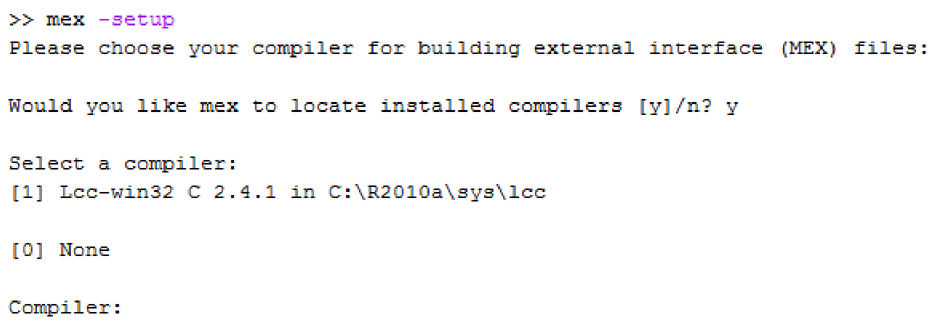
Requirements and Installation:

Before the connection of Matlab with LabVIEW, all programs must be installed correctly. Programs that have to be installed are:

- The MathWorks MATLAB and The MathWorks Simulink R2010a (32-bit)
- LabVIEW 2011 (32-bit)
- LabVIEW Simulation Interface Toolkit (SIT) (32-bit)
- Microsoft Visual Studio 2010
- Microsoft Visual studio 2008 SP1
- Microsoft Windows SDK for Windows 7 (7.1)
- LabVIEW Real-Time Module
- LabVIEW FPGA Module
- LabVIEW FPGA Compile Farm
- NI LabVIEW FPGA Xilinx 12.4 SP1 Tools
- RIO drivers

Matlab, LabVIEW, SIT (`C:\SimutlationInterfaceToolkit\`) and Microsoft Visual Studio are installed by a standard way to the root directory (`C:\`).

The next step is the interconnection of these programs and their settings, in Matlab the compiler is selected and Matlab is connected to the SIT.
Selecting the compiler is done by command `mex -setup` in Matlab shell as depicted in Figure 17.

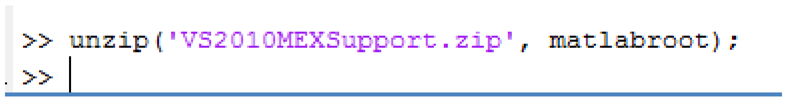


```
>> mex -setup
Please choose your compiler for building external interface (MEX) files:

Would you like mex to locate installed compilers [y]/n? y

Select a compiler:
[1] Lcc-win32 C 2.4.1 in C:\R2010a\sys\lcc

[0] None

Compiler:
```

Figure 17: Compilers Menu

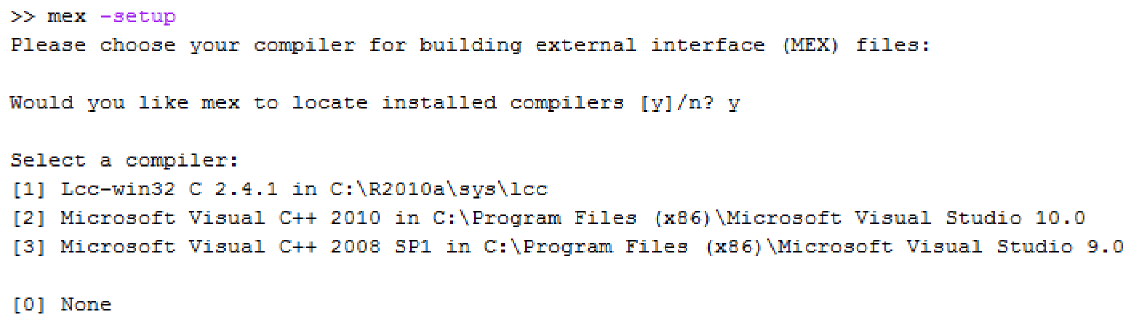
In our case we use Microsoft Visual studio 2010 , it can be seen from Figure 17 that Microsoft Visual studio 2010 is not listed in the compilers Menu due to the incompatibility of Matlab R2010a with Microsoft Visual Studio 2010 to solve this problem we decide to install a patch named `VS2010MEXsupport` alongside with Microsoft Windows SDK for Windows 7 (7.1) and Visual Studio 2008 SP1 to fix the incompatibility issue. Unzip the contents of the patch into your MATLAB installation. This can be done from within MATLAB itself with the command: `unzip('path_to_zip_file', matlabroot);` [22].

Now the patch is installed as in Figure 18, execute the command `mex -setup` again in Matlab shell Figure as shown in 19



```
>> unzip('VS2010MEXSupport.zip', matlabroot);
>>
```

Figure 18: The patch is installed

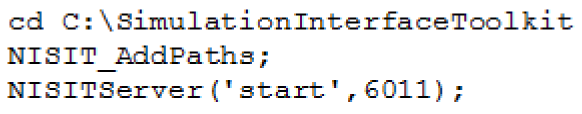


```
>> mex -setup
Please choose your compiler for building external interface (MEX) files:

Would you like mex to locate installed compilers [y]/n? y

Select a compiler:
[1] Lcc-win32 C 2.4.1 in C:\R2010a\sys\lcc
[2] Microsoft Visual C++ 2010 in C:\Program Files (x86)\Microsoft Visual Studio 10.0
[3] Microsoft Visual C++ 2008 SP1 in C:\Program Files (x86)\Microsoft Visual Studio 9.0

[0] None
```

Figure 19: Visual studio 2010 appear on the list

Note: If you installed the MATLAB application software files as read-only, the Simulation Interface Toolkit does not configure the SIT Server to start automatically and the SignalProbe block does not appear in the Simulink Library Browser window.

To see these changes, add the following lines to the matlabrc.m file which is located at `C:\ProgramFiles\MATLAB\R2010b\toolbox\local\matlabrc.m`. At the end of the file we write what is in Figure 20.

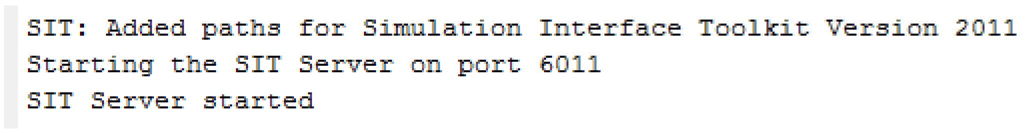


```
cd C:\SimulationInterfaceToolkit
NISIT_AddPaths;
NISITServer('start',6011);
```

Figure 20: The lines that must be added to matlabrc.m file

Note: you may have to run matlabrc.m as administrator to have the writing access. The last change is in the directory of Simulation Interface Toolkit (SIT), in `C:\SimulationInterfaceToolkit\2010\ModelInterface\tmw\R2007b\` the version is changed from `SIT_VERSION=5.0` to current version in our case `SIT_VERSION=2010`.

Creating DLL file from Simulink model:



```
SIT: Added paths for Simulation Interface Toolkit Version 2011
Starting the SIT Server on port 6011
SIT Server started
```

Figure 21: SIT server is successfully connected to Matlab

Now everything is set and a Simulink model of our application can be created and then converted into a DLL. If we can see the following text during start as in Figure 21, the Matlab is set correctly [11].

Running Simulink, the NI SIT Blocks Library must appear in the Simulink Library browser like in Figure 22. Simulink model must contain SignalProbe block and the

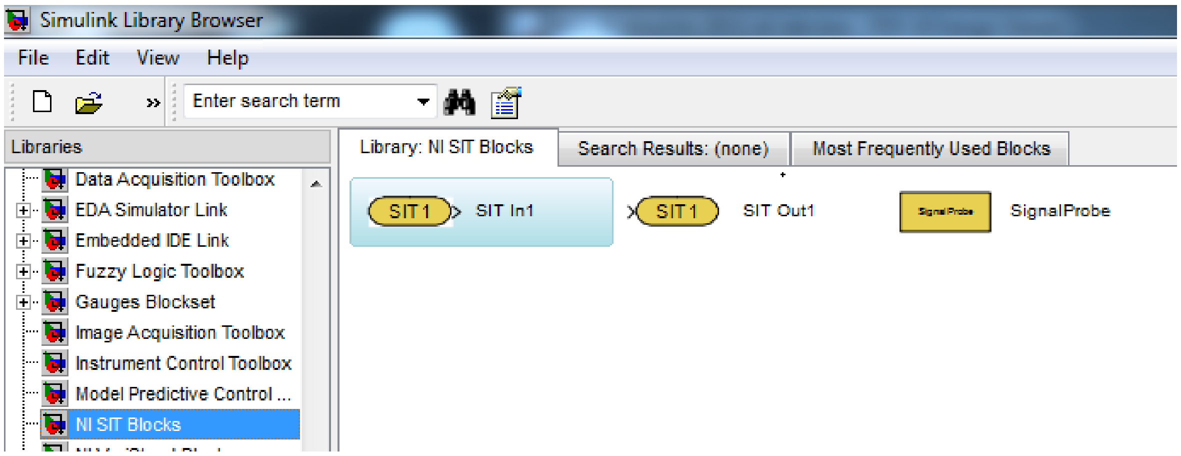


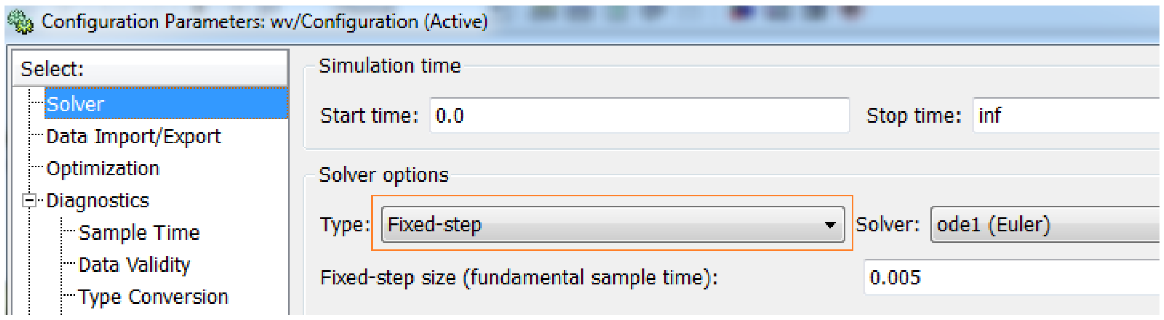
Figure 22: NI SIT Blocks Library

LabView is connected via inport and outport. Unconnected inports can be used for monitoring waveforms in the processor of NI PCIe-7851R. The DLL file and other files, that are generated together, are needed for connecting to LabVIEW environment. This interconnection provides Simulation interface toolbox. For the simulation to work properly, in Matlab Simulink from Tools $\rightarrow$ $Real - TimeWorkshop$ $\rightarrow$ $Options$ ... select Fixed-step as a type in Solver options as shown in Figure 23.

If we don't want to run the model in VXworks OS in seperated execution host, we need only DLL file which is generated by `nidll.tlc`. These files are generated in Matlab Simulink. The different versions of Matlab Simulink may have different paths to the generator.

In Matlab R2010b Simulink path is Tools $\rightarrow$ $Real - Timeworkshop$ $\rightarrow$ $Options$ $\rightarrow$ $Real - TimeWorkshop$ where `nidll.tlc` is selected as System target file and then we click on build as in Figure 24.

Connection of DLL file with LABVIEW:



Configuration Parameters: wv/Configuration (Active)

Simulation time

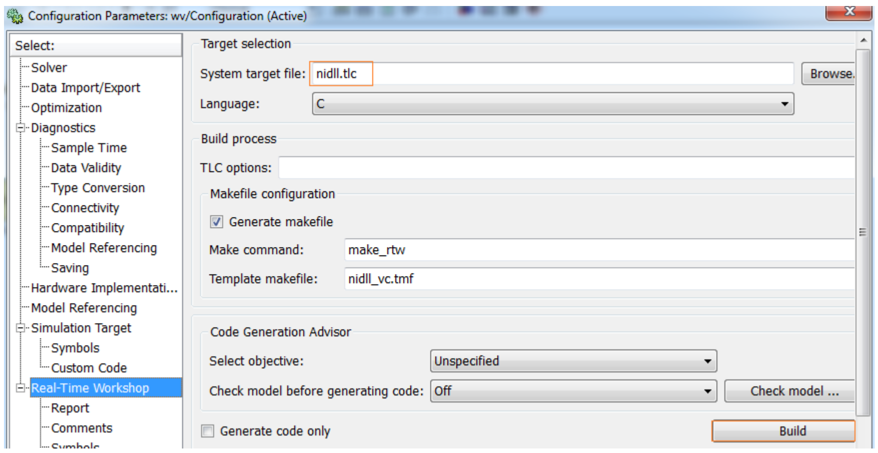
Start time: 0.0 Stop time: inf

Solver options

Type: Fixed-step Solver: ode1 (Euler)

Fixed-step size (fundamental sample time): 0.005

Figure 23: Configure fixed-step to run the simulation properly



Configuration Parameters: wv/Configuration (Active)

Target selection

System target file: nidll.tlc

Language: C

Build process

TLC options:

Makefile configuration

☑ Generate makefile

Make command: make_rtw

Template makefile: nidll_vc.tmf

Code Generation Advisor

Select objective: Unspecified

Check model before generating code: Off

☐ Generate code only

Figure 24: Compiling and generating the DLL file

The first thing to know is what type of connections of DLL and LABVIEW and NI R series will be needed, Here it is not needed to change the data sent or received from the R series card which means that the Matlab model is in the form of DLLs, directly connected to the input and output of the cards.

Creating a custom FPGA bitfile [23]

- Make a copy of the Sample FPGA VI and Projects:
  For R series Devices:

  1. Browse to "`LabVIEW2011\vi.lib\addons\SimulationInterface\_IOTypes\Plugins\NI-FPGA\FPGAIOSource`" directory.

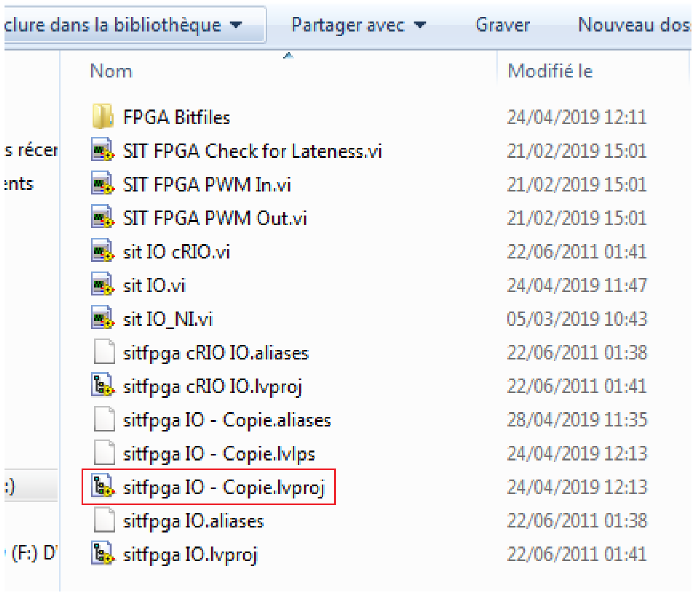
  2. Create a copy of "sitfpga IO.lvproj" and place this copy in this same directory as in Figure 25.

  3. Open this copy in LABVIEW.

  4. Expand the **My computer** item in the project Explorer window.

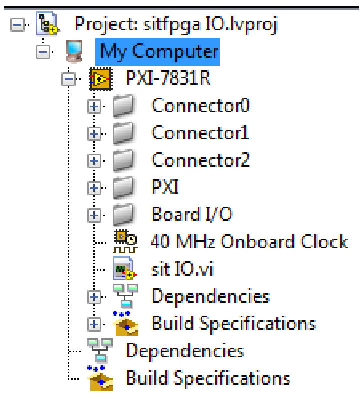
  5. Expand the **PXI-7831R** item in the Project Explorer window Figure 26.

     This example project defines a PXI-7831R as the target, since PCIe 7851R is the target for this case, complete the following:



Nom	Modifié le
FPGA Bitfiles	24/04/2019 12:11
SIT FPGA Check for Lateness.vi	21/02/2019 15:01
SIT FPGA PWM In.vi	21/02/2019 15:01
SIT FPGA PWM Out.vi	21/02/2019 15:01
sit IO cRIO.vi	22/06/2011 01:41
sit IO.vi	24/04/2019 11:47
sit IO_NI.vi	05/03/2019 10:43
sitfpga cRIO IO.aliases	22/06/2011 01:38
sitfpga cRIO IO.lvproj	22/06/2011 01:41
sitfpga IO - Copie.aliases	28/04/2019 11:35
sitfpga IO - Copie.lvlps	24/04/2019 12:13
sitfpga IO - Copie.lvproj	24/04/2019 12:13
sitfpga IO.aliases	22/06/2011 01:38
sitfpga IO.lvproj	22/06/2011 01:41

Figure 25: Make a copy of sitfpga IO.lvproj



```
Project: sitfpga IO.lvproj
  My Computer
    PXI-7831R
      Connector0
      Connector1
      Connector2
      PXI
      Board I/O
      40 MHz Onboard Clock
      sit IO.vi
      Dependencies
      Build Specifications
    Dependencies
    Build Specifications
```

Figure 26: sitfpga IO.lvproj opened in LabVIEW

(a) Right click on the **My computer** item and select **New** » **Targets and Devices** from the shortcut menu.

(b) Select **Existing target or device** » **discover an existing target(s) or device(s)** » expand **FPGA Target** » **R Series**.

(c) Select **RIO0 (PCIe-7851R)** as in Figure 27.

(d) In the **Project Explorer** window, click and drag **sit IO.vi** from the **PXI-7831R (PXI-7831R)** target to the new target **RIO0 (PCIe-7851R)**.

(e) Since **PXI-7831R** target has no use after, it can be deleted.

6. Double-click “sit IO.vi” in the **Project Explorer** window to open it.

7. Right-click **FPGA Target (RIO, PCIe-7851R)** » **New** » **FPGA I/O** then add **Available Resources** to **New FPGA I/O** as in Figure 28.

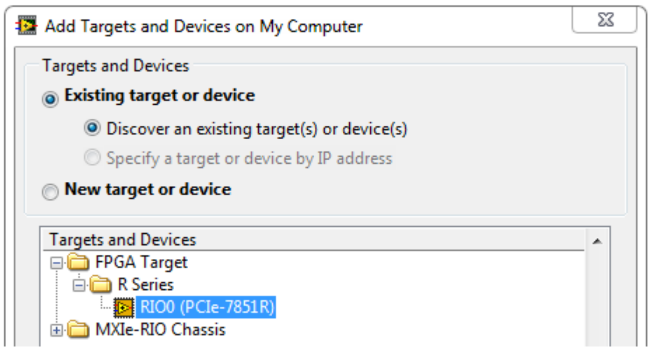
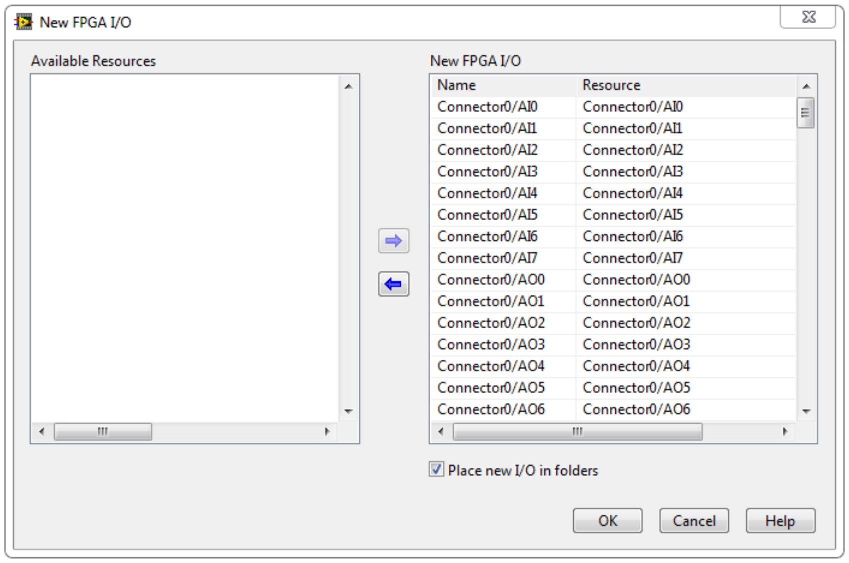

Add Targets and Devices on My Computer
Targets and Devices
Existing target or device
Discover an existing target(s) or device(s)
Specify a target or device by IP address
New target or device
Targets and Devices
FPGA Target
R Series
RIO0 (PCIe-7851R)
MXIe-RIO Chassis

Figure 27: Selecting RIO0 (PCIe-7851R)



Name	Resource
Connector0/AI0	Connector0/AI0
Connector0/AI1	Connector0/AI1
Connector0/AI2	Connector0/AI2
Connector0/AI3	Connector0/AI3
Connector0/AI4	Connector0/AI4
Connector0/AI5	Connector0/AI5
Connector0/AI6	Connector0/AI6
Connector0/AI7	Connector0/AI7
Connector0/AO0	Connector0/AO0
Connector0/AO1	Connector0/AO1
Connector0/AO2	Connector0/AO2
Connector0/AO3	Connector0/AO3
Connector0/AO4	Connector0/AO4
Connector0/AO5	Connector0/AO5
Connector0/AO6	Connector0/AO6

Figure 28: Adding Available Resources to RIO0 (PCIe-7851R)

8. Select **File** » **Save As** from the pull-down menu of the sit IO VI window.

9. Ensure the **Substitute copy for original** option is selected and click the **Continue** button.

10. Rename the VI and save it to the
“`LabVIEW2011\vi.lib\addons\SimulationInterface\_IOTypes\Plugins`
`\NI-FPGA\FPGAIOSource"`directory.

11. Save the project by clicking the **Save** button in the **Project Explorer** window, then the **Project Explorer** should be similar to what is in Figure 29.

- Customize the FPGA VI:
The process of creating the custom FPGA VI differs depending on the hardware devices you are using.
Refer to the FPGA Module documentation for information about creating FPGA

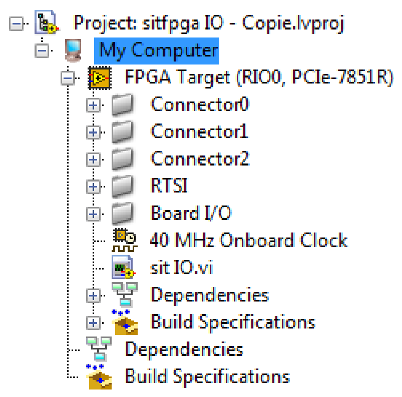


Figure 29: SIT IO.vi in PCIe-7871R target

VIs and Bitfiles for an FPGA target.
The default project defines the following FPGA I/O items for the PXI-7831R device: analog input channels 0-7, analog output channels 0-7, and digital lines 0-39 on both connectors 1 and 2 and digital line 0-15 on connector 0.
You can add or remove FPGA I/O items depending on the device and the needs of the simulation.
So here for the PCIe card which have different connectors and RTSI signals instead of TRIG signals so FPGA I/O should be modified and TRIG signals should be replaced by RTSI ones. "**instruction and pictures about the process**"
While creating or modifying the FPGA VI, pay attention to the following guidelines to ensure the SIT Connection Manager dialog box recognizes this FPGA VI.

- Do not modify, remove, or rename any objects in the gray areas of the sample FPGA VI.
- As you create controls and indicators to represent FPGA I/O connections, do not begin the name of these new objects with an underscore or asterisk.
- National Instruments recommends that you start control names with AI for analog in, AO for analog out, and so on.

- Compile the FPGA VI into a Bitfile.
  Complete the following steps to compile the custom FPGA VI and create the Bitfile.

  1. Display the **Project Explorer** window.
  2. Right-click the "sit IO.vi" in the tree and select **Create build specification** as shown in Figure 30.
  3. Expand **build specification** tree then Right-click the **FPGA VI** in the tree and select **Build** from the shortcut menu to compile the FPGA VI Figure 31, LabVIEW then creates a Bitfile for this VI.

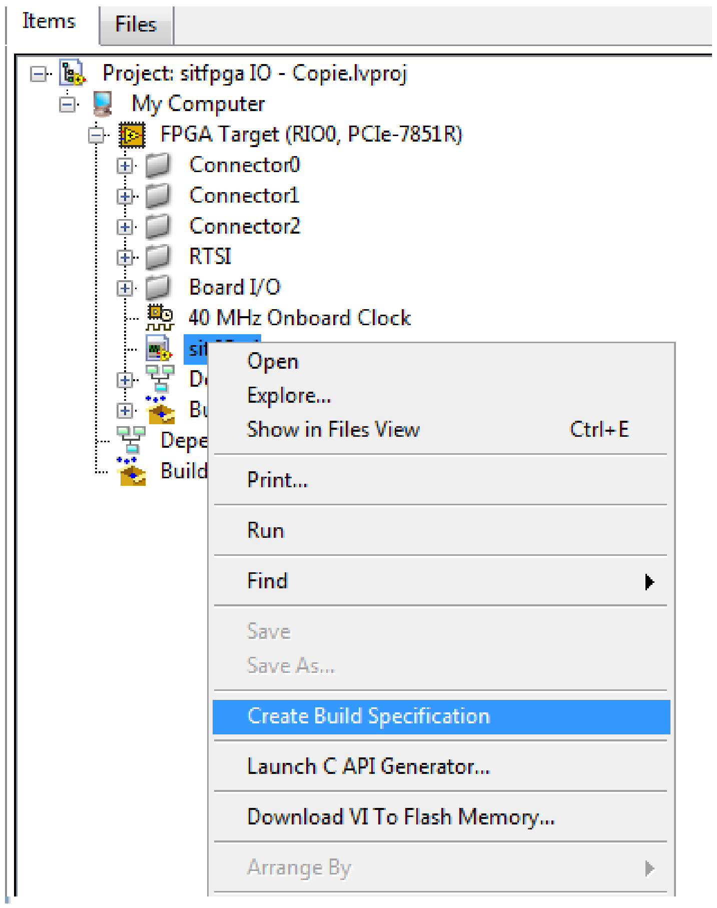

Items
Files
Project: sitfpga IO - Copie.lvproj
My Computer
FPGA Target (RIO0, PCIe-7851R)
Connector0
Connector1
Connector2
RTSI
Board I/O
40 MHz Onboard Clock
Open
Explore...
Show in Files View Ctrl+E
Print...
Run
Find
Save
Save As...
Create Build Specification
Launch C API Generator...
Download VI To Flash Memory...
Arrange By

Figure 30: Create build specification

But here the **LABVIEW FPGA compile farm** and **LABVIEW Xilinx 12.4 SP1 build specification** must be installed to compile it using local compiler server as in Figure 32.

4. Copy the resulting Bitfile to the "`labview\vi.lib\addons\SimulationInterface\_IOTypes\Plugins \NI-FPGA\FPGABitfiles''` directory.
   The compiler places the bitfile in a subdirectory, FPGA Bitfiles, relative to the project file directory.
   By default, the bitfile name is name of project_name of FPGA VI .lvbitx as in Figure 33.

5. Rename the bitfile according to the rules for naming custom bitfiles [24] similar to Figure 34.

Connection manager [25]

After the creation of the blank VI in LABVIEW we can create control and indicators by which we want to control MATLAB model and display a variety of waveforms. Connection is done using the SIT connection Manager by selecting **Tools » SIT**

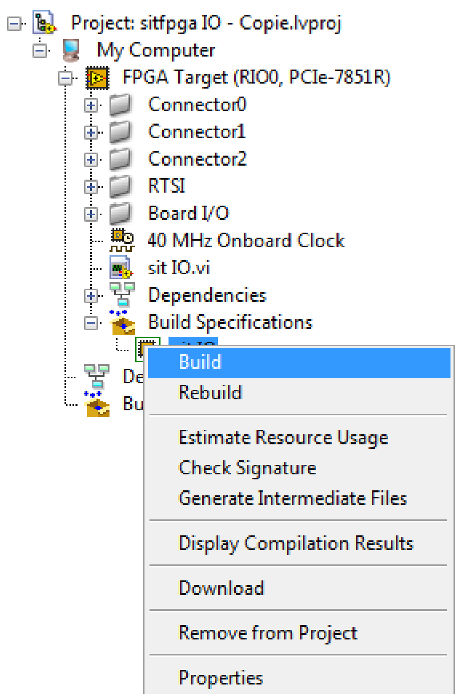
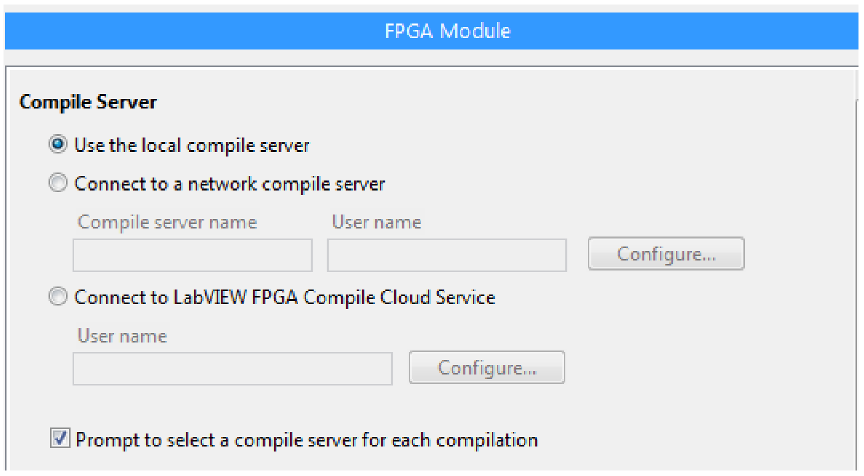


Figure 31: Build sit IO


FPGA Module

Compile Server

(●) Use the local compile server

( ) Connect to a network compile server

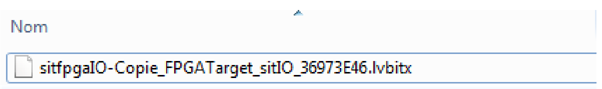
Compile server name | User name | Configure...

( ) Connect to LabVIEW FPGA Compile Cloud Service

User name | Configure...

[✓] Prompt to select a compile server for each compilation

Figure 32: Create build specification

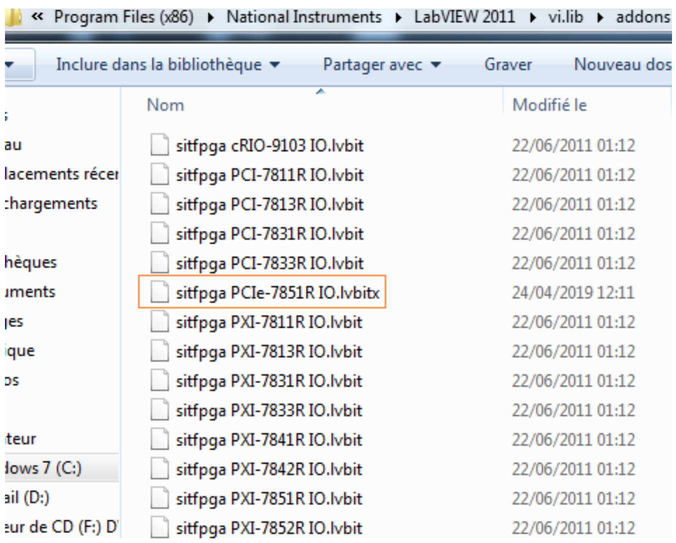

Nom

sitfpgaIO-Copie_FPGATarget_sitIO_36973E46.lvbitx

Figure 33: Created bitfile

Connection Manager, this dialog box create mappings between a host VI, model and certain Real-Time series hardware and generate block diagram code of the host VI or driver VI. This dialog box includes the following components:

- Model and Host: shows options related to the model and the execution host



Nom	Modifié le
sitfpga cRIO-9103 IO.lvbit	22/06/2011 01:12
sitfpga PCI-7811R IO.lvbit	22/06/2011 01:12
sitfpga PCI-7813R IO.lvbit	22/06/2011 01:12
sitfpga PCI-7831R IO.lvbit	22/06/2011 01:12
sitfpga PCI-7833R IO.lvbit	22/06/2011 01:12
sitfpga PCIe-7851R IO.lvbitx	24/04/2019 12:11
sitfpga PXI-7811R IO.lvbit	22/06/2011 01:12
sitfpga PXI-7813R IO.lvbit	22/06/2011 01:12
sitfpga PXI-7831R IO.lvbit	22/06/2011 01:12
sitfpga PXI-7833R IO.lvbit	22/06/2011 01:12
sitfpga PXI-7841R IO.lvbit	22/06/2011 01:12
sitfpga PXI-7842R IO.lvbit	22/06/2011 01:12
sitfpga PXI-7851R IO.lvbit	22/06/2011 01:12
sitfpga PXI-7852R IO.lvbit	22/06/2011 01:12

Figure 34: Final bitfile

- Execution Host: Use this section to specify the location of the SIT Server.
  1. Simulation Environment: specifies that the SIT Server is running on a non-RT target.
  2. Simulation IP Address: specifies the IP address of the computer on which the SIT Server is running.
  3. Real-Time Target: specifies that the SIT Server is running on an RT target.
  4. Driver VI on Localhost: specifies that the SIT server is running in a driver VI on the local machine.
  5. Target: specifies the IP address of the RT target on which the simulation is running.
  6. Port: specifies the port on which the SIT Server is running.
- Current Model or Current Model DLL: allows you to select a model resource file.
- Project Directory: specifies where the Simulation Interface Toolkit stores files such as data logging configurations, replay configurations, and the driver VI.

• Mappings: Use this page to configure mappings between host VI controls/indicators and model parameters/signals.
Here we should follow a valid mapping [26].

• Data Logging: use this page to configure data logging options for the simulation.

• File Playback: use this page to configure file playback options for the simulation.

- Hardware I/O: use this page to configure mappings between RT Series hardware and a model DLL.

Simple case study "sine wave model"

- Consider the Simulink model in Figure 35:

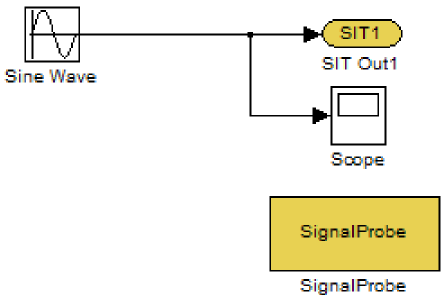
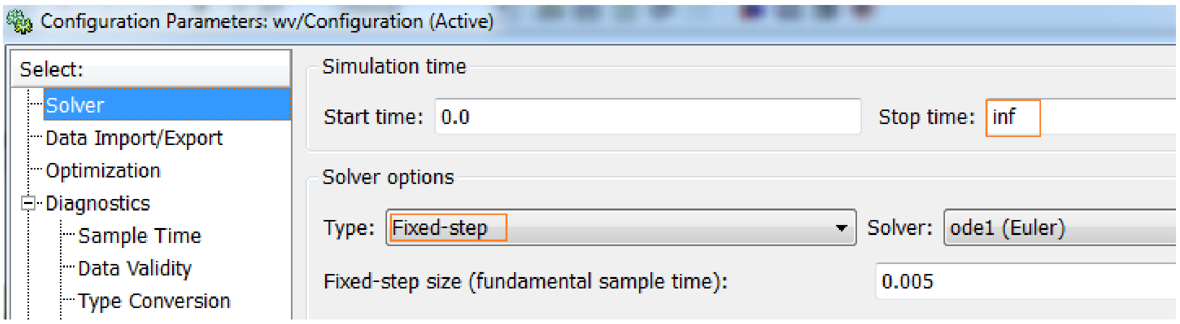


Figure 35: Sinwave Simulink model

- Configure the communication between Simulink and SIT:
  From the menu bar select **Simulation** » **Configuration Parameters** to launch the Simulation Parameter dialog box, make simulation **Fixed-step** and set **Stop time to inf** on **Solver** category as in Figure 36.

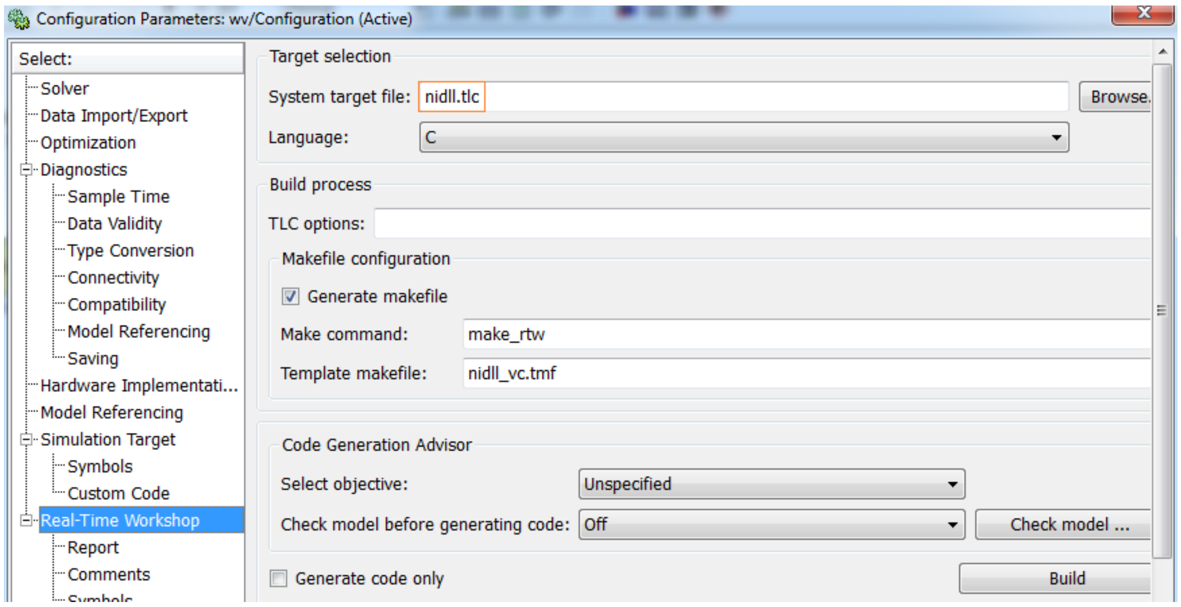


Start time: 0.0
Stop time: inf
Type: Fixed-step
Solver: ode1 (Euler)
Fixed-step size (fundamental sample time): 0.005

Figure 36: Set fixed-step as a type of sampling

Select the **Real-Time Workshop** tab, click **Browse** button to launch the **System Target File Browser** dialog box and select **nidll.tlc** from the list as shown in Figure 37.
Click the **Build** button in the **Category** section to begin building the model DLL. The following message in the MATLAB command window indicates that Real-Time Workshop has completed building the model DLL.

```
###Successful completion of Real−Time Workshop build procedure for model:ModelName
```



Target selection

System target file: nidll.tlc

Language: C

Build process

TLC options:

Makefile configuration

[✓] Generate makefile

Make command: make_rtw

Template makefile: nidll_vc.tmf

Code Generation Advisor

Select objective: Unspecified

Check model before generating code: Off

[ ] Generate code only

Figure 37: Building the DLL file

- Creating the Front Panel of the Host VI: Complete the following steps to create the front panel of the host VI for the sinewave model:

  1. Launch LABVIEW and create new blank VI.
  2. Add a waveform chart indicator to the front panel and label the indicator Sine Wave. The front panel of the host VI resembles the following Figure 38.

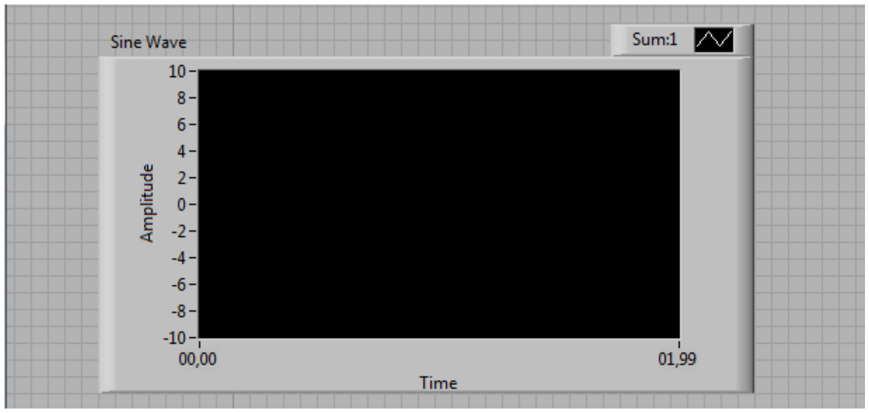
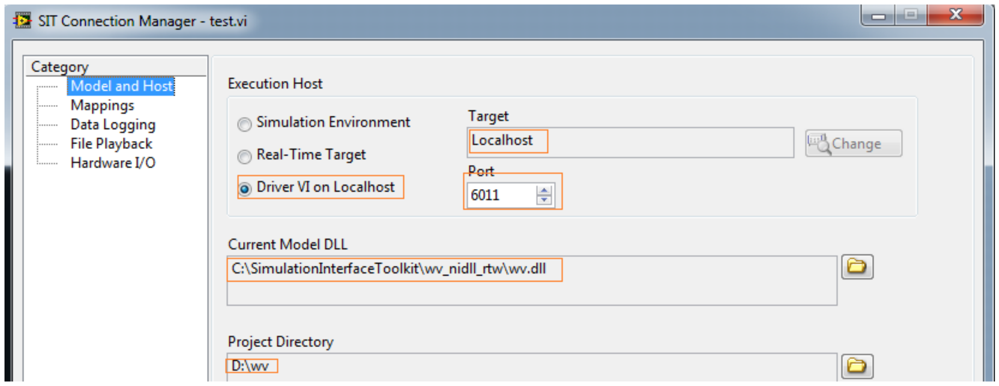


Figure 38: LabVIEW sinwave VI

- Executing sine wave model DLL on a Windows Computer [7]:
  Complete the following steps to execute a model DLL on a Windows computer:

  1. From the menu bar of the VI select **Tools** » **SIT Connection Manager**

2. In the **Model and Host section**:
   - Select the **Driver VI on Localhost** option and **port 6011**.
   - Specify **Current Model DLL** and **Project directory** in Figure 39.



Execution Host

Simulation Environment
Real-Time Target
Driver VI on Localhost (selected)

Target: Localhost

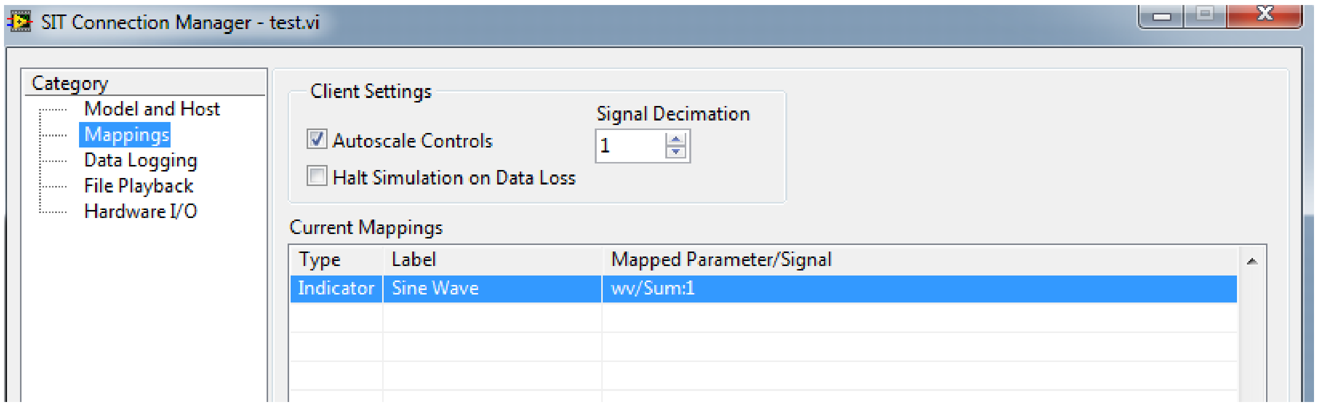
Port: 6011

Current Model DLL: C:\SimulationInterfaceToolkit\wv_nidll_rtw\wv.dll

Project Directory: D:\wv

Figure 39: Set fixed-step as a type of sampling

3. In **Mappings** map indicator **Sine Wave** to **Sum1** parameter Figure 40.



Client Settings

Autoscale Controls ✓

Halt Simulation on Data Loss

Signal Decimation: 1

Current Mappings

Type	Label	Mapped Parameter/Signal
Indicator	Sine Wave	wv/Sum:1

Figure 40: Create mappings

4. Click the **Hardware I/O** tab then **Configure HW I/O** as in Figure 41.
5. In the **Configure HW I/O** dialog box, complete the following steps to configure mappings between an FPGA target and a model DLL.
   (a) Right-click the IP address or target name in the **Device Tree** and select **Add Device » NI-FPGA** as shown Figure 42 .
   (b) Configure the NI-FPGA target [27] by checking **Select from found devices** which displays any FPGA targets available on the RT target and the bitfile will be specified if correctly follow how to create and name bitfile Figure as in 43.

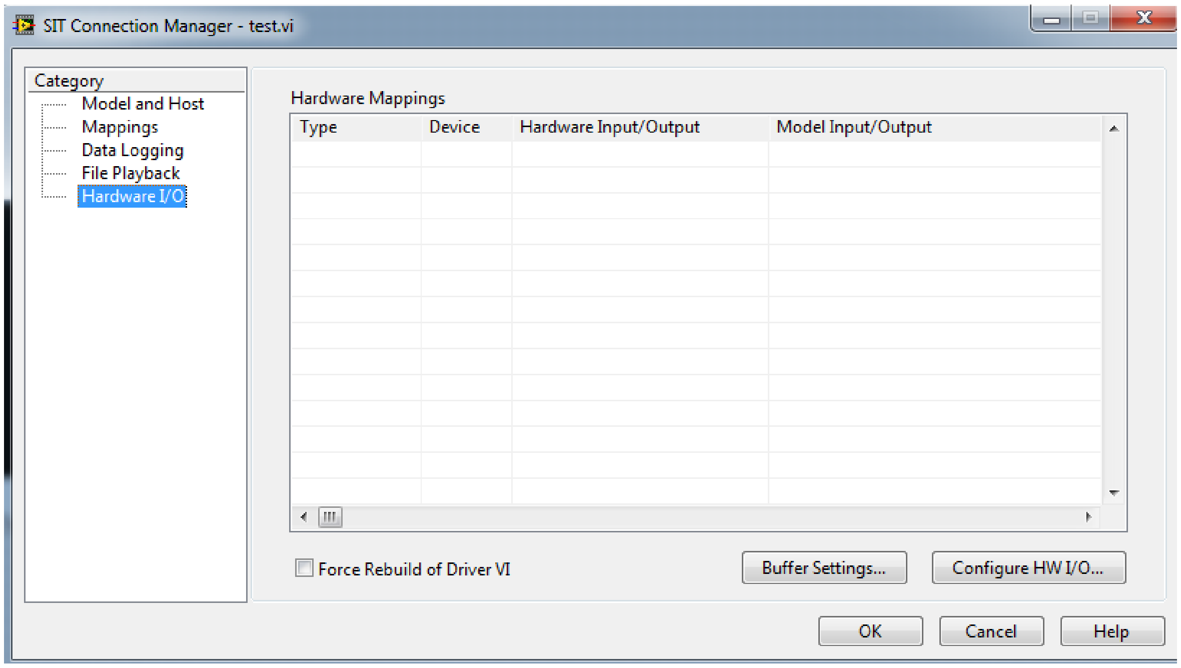


Figure 41: Configure HW I/O

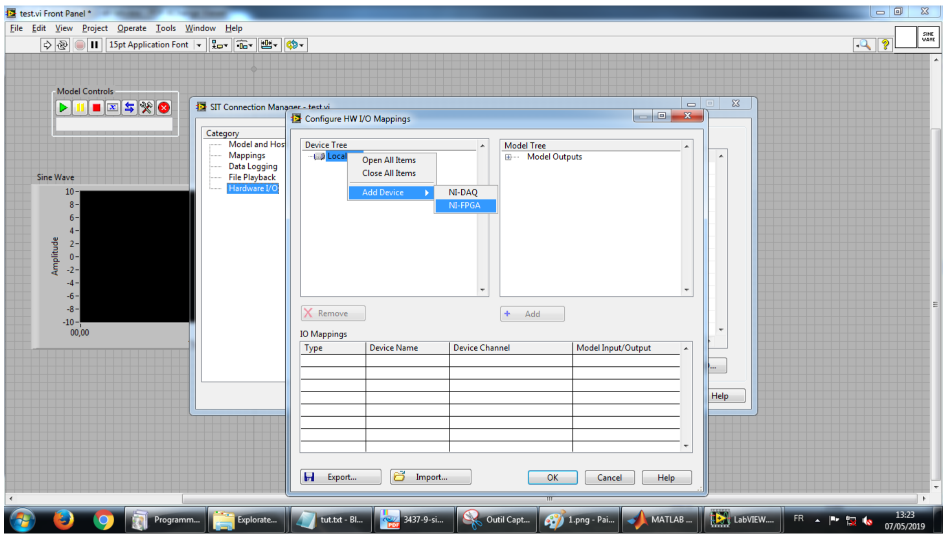
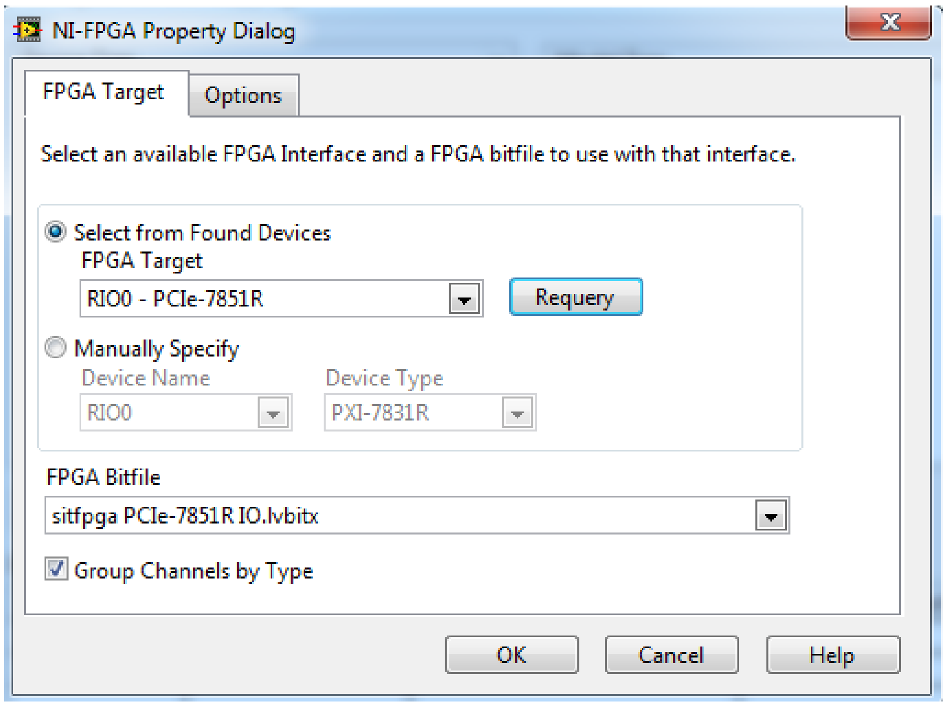


Figure 42: Add FPGA Target

(c) Select the appropriate signal pairs in the **Device Tree** and the **Model Tree** here we map **AO1** to **Sum1** and click the **Add** button as shown in Figure 44.

(d) Click the **OK** button to exit the **Configure HW I/O Mappings** dialog box.
The newly defined mappings appear in the **Hardware Mappings** table of the **Hardware I/O** page in the **SIT Connection Manager** as in Figure 45.


NI-FPGA Property Dialog

Select an available FPGA Interface and a FPGA bitfile to use with that interface.

(●) Select from Found Devices

FPGA Target: RIO0 - PCIe-7851R

( ) Manually Specify

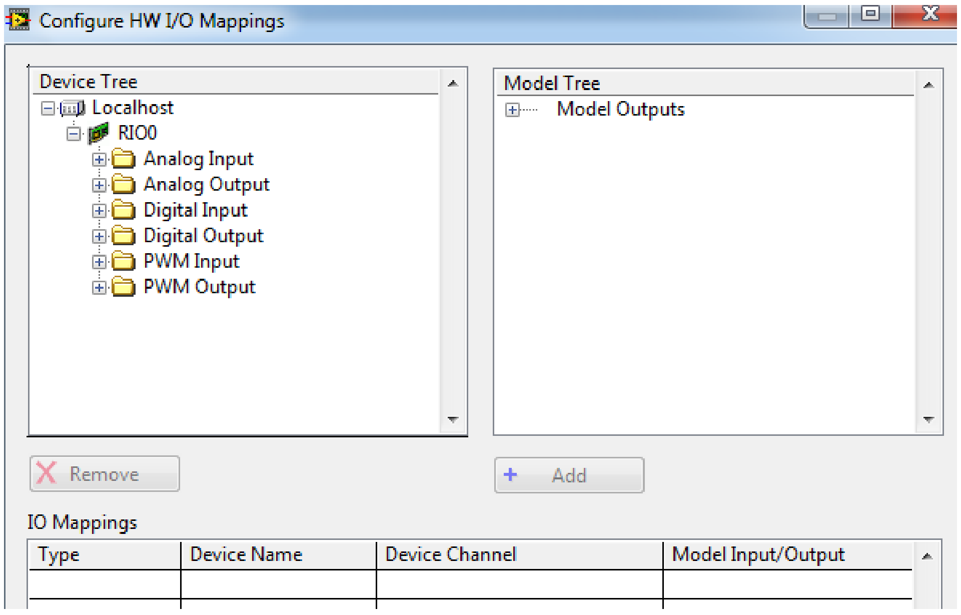
Device Name: RIO0

Device Type: PXI-7831R

FPGA Bitfile: sitfpga PCIe-7851R IO.lvbitx

[✓] Group Channels by Type

Figure 43: Select the FPGA target and FPGA Bitfile

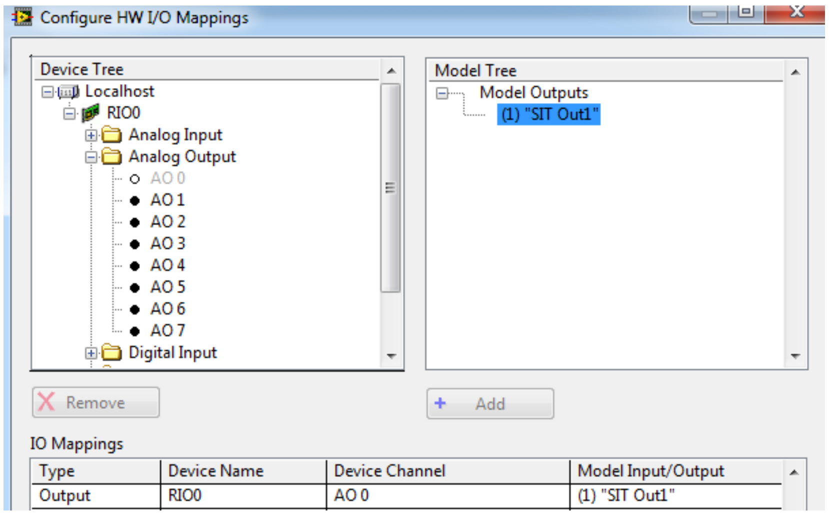

Configure HW I/O Mappings

Type	Device Name	Device Channel	Model Input/Output

Figure 44: The FPGA target is added

(e) Click the **OK** button to exit the **SIT Connection Manager** and rebuild the host VI with the new mappings.

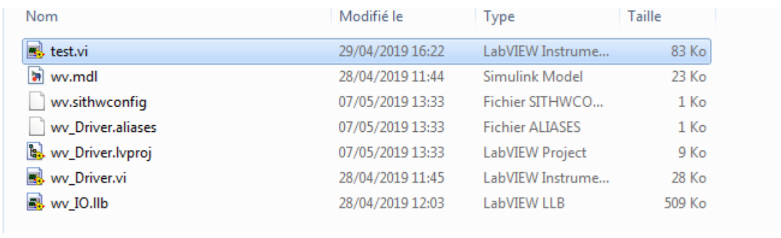
6. Click the **Build Model Files** button to generate the driver VI. The LabVIEW Simulation Interface Toolkit places the driver VI, <ModelName>_driver.vi, in the same directory as the model DLL you specified, as in Figure 46.



IO Mappings

Type	Device Name	Device Channel	Model Input/Output
Output	RIO0	AO 0	(1) "SIT Out1"

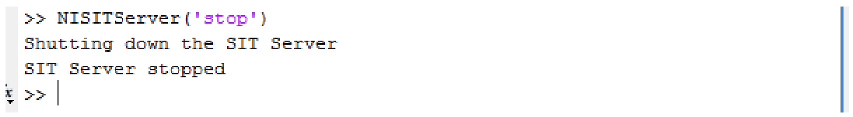
Figure 45: Add IO Mapping



Nom	Modifié le	Type	Taille
test.vi	29/04/2019 16:22	LabVIEW Instrume...	83 Ko
wv.mdl	28/04/2019 11:44	Simulink Model	23 Ko
wv.sithwconfig	07/05/2019 13:33	Fichier SITHWCO...	1 Ko
wv_Driver.aliases	07/05/2019 13:33	Fichier ALIASES	1 Ko
wv_Driver.lvproj	07/05/2019 13:33	LabVIEW Project	9 Ko
wv_Driver.vi	28/04/2019 11:45	LabVIEW Instrume...	28 Ko
wv_IO.llb	28/04/2019 12:03	LabVIEW LLB	509 Ko

Figure 46: Driver VI appear in the same directory as the model DLL

7. Close the simulation environment or manually stop the SIT Server by echoing the following command NISITServer('stop') in Matlab command line as in Figure 47; also, ensure that no other driver VIs are running on the Windows computer. If you do not stop the SIT Server before continuing, LabVIEW returns an error when you try to run the VIs.



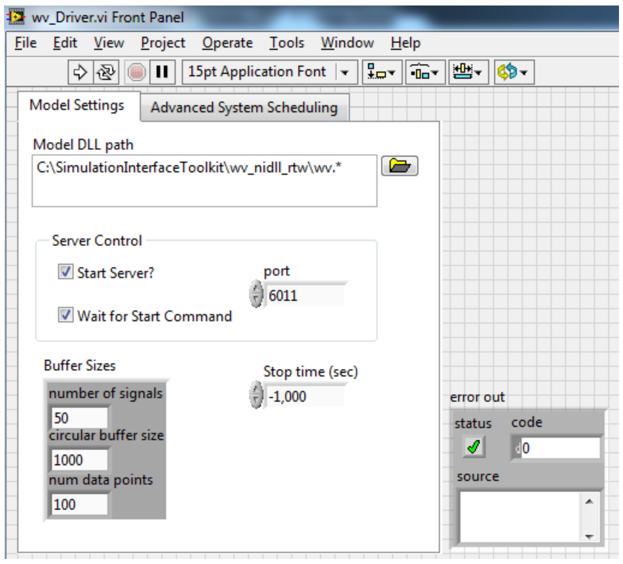
```
>> NISITServer('stop')
Shutting down the SIT Server
SIT Server stopped
>> |
```

Figure 47: Stop the SIT server port 6011

8. Open and run the driver VI,as in Figure 48.

9. Open and run the host VI,as shown Figure 49.

10. From the oscilloscope as seen in Figure 50.


wv_Driver.vi Front Panel

File Edit View Project Operate Tools Window Help

15pt Application Font

Model Settings | Advanced System Scheduling

Model DLL path: C:\SimulationInterfaceToolkit\wv_nidll_rtw\wv.*

Server Control

☑ Start Server? port: 6011

☑ Wait for Start Command

Buffer Sizes

number of signals: 50

circular buffer size: 1000

num data points: 100

Stop time (sec): -1,000

error out

status: ✓ code: 0

source:

Figure 48: The Driver VI

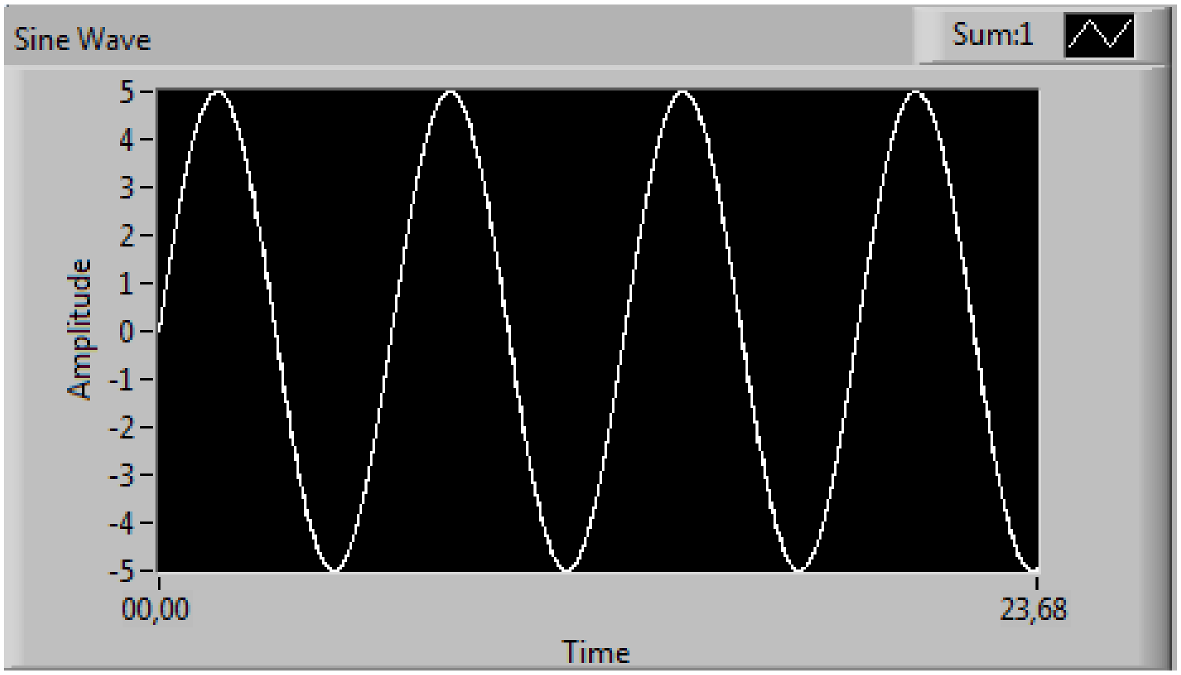


Figure 49: The host VI is running

Appendix B.NI Veristand

Requirements and Installation:

The computer on which you develop an NI VeriStand project might be different from the host computer in the system. The development computer contains the NI VeriStand software. To extend the functionality of NI VeriStand, you might also use the following NI products on the development computer [16]:

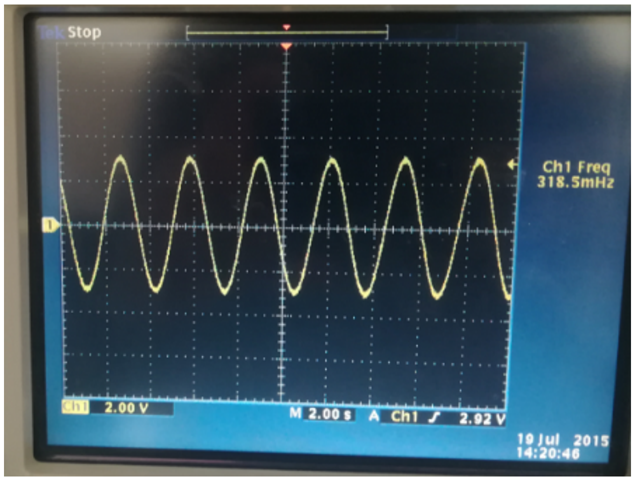


Figure 50: Sinewave in oscilloscope

- **LabVIEW Development System:** If you want to create custom devices, workspace controls/indicators, timing devices, and/or **Tools** menu utilities, you need the LabVIEW Development System.
- **LabVIEW Real-Time Module:** You need this module to use RT functions in custom device VIs.
- **LabVIEW FPGA Module:** If you add a National Instruments FPGA target to a project, it must have an associated FPGA bitfile. NI VeriStand provides FPGA bitfiles for certain FPGA devices. If you want to customize these FPGA bitfiles or create a custom FPGA bitfile for another FPGA target, you need the FPGA Module.

In this project NI VeriStand 2013 SP1 is used, to make sure everything will work properly, before the connection of Matlab with Veristand, all the following programs must be installed correctly:

- MATLAB and Simulink R2010a (32-bit)
- LabVIEW 2011 (32-bit)
- Veristand 2013 SP1
- Microsoft Visual Studio 2010
- Microsoft Visual studio 2008 SP1
- Microsoft Windows SDK for Windows 7 (7.1)
- LabVIEW Real-Time Module
- LabVIEW FPGA Module
- LabVIEW FPGA Compile Farm

- NI LabVIEW FPGA Xilinx 12.4 SP1 Tools
- RIO drivers

The next step is to ensure that you have the correct compiler since it is needed to create a compiled model that runs on Windows, the compiler is Microsoft Visual studio 2010. For other RT targets and real-time operating system (RTOS) that each runs refer to Real-Time Controllers and Real-Time Operating System Compatibility [28].

Migrating Simulation Interface Toolkit Applications to Veristand and creating DLL file:

The process for building a Simulink DLL model that has been followed in Appendix A, remains the same for NI VeriStand Environment with some changes highlighted here in this section. Migrating Models:
After the installation of NI Veristand 2013 SP1, replace any SIT input and output blocks in the Simulink model with NI VeriStand input and output blocks.
The needed blocks are located in the `NI VeriStand Blocks` simulink library as can be seen in Figure 51.

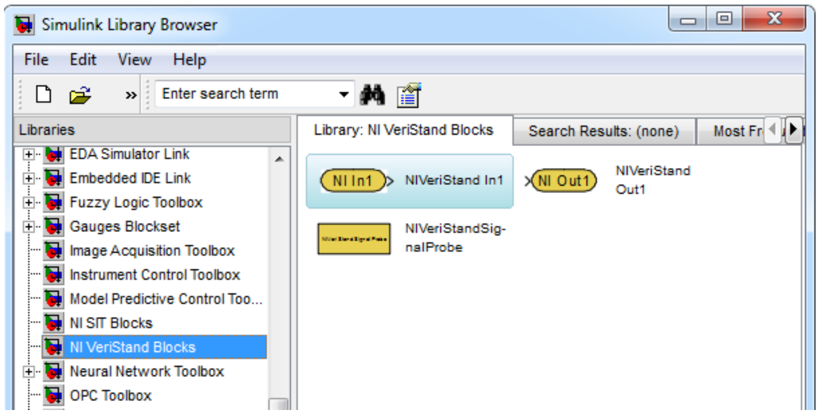


Figure 51: NI VeriStand Blocks Library

The Simulink model should be similar to that in Figure 52:

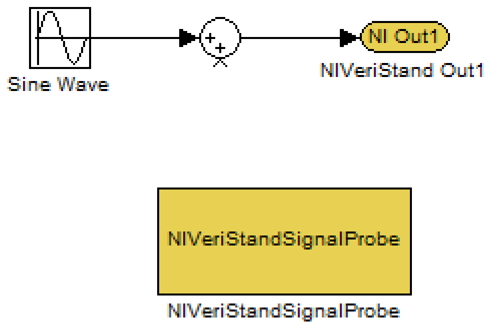


Figure 52: Sinewave Simulink Model that can be connected to VeriStand

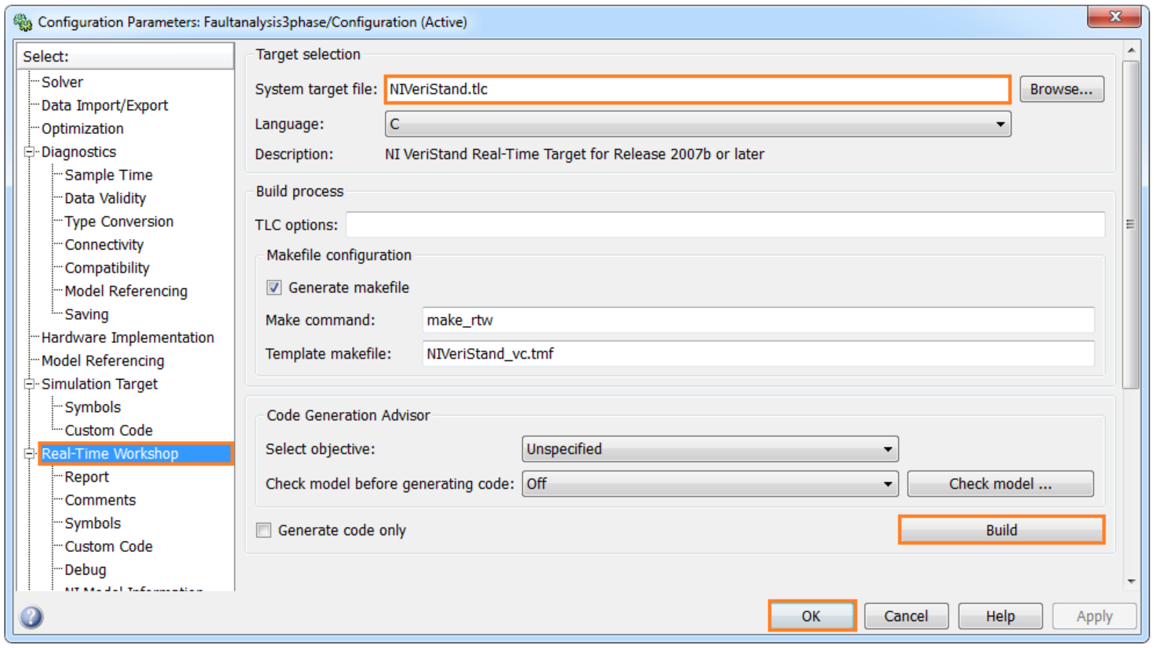
Build the Compiled Model in Simulink: After setting up and selecting the needed compiler and creating the simulink model, assure the selection of appropriate settings to compile the model by following these steps:

1. Select Simulation $\rightarrow$ *ConfigurationParameters* to launch the Model Configuration Parameters dialog box.

2. Click the Solver tab and configure the following options:

    - Stop time: inf
    - Type: Fixed-step
    - Solver: discrete (no continuous states)

3. Click the Real-Time Workshop tab(see Figure 53).



Configuration Parameters: Faultanalysis3phase/Configuration (Active)

Target selection

System target file: NIVeriStand.tlc

Language: C

Description: NI VeriStand Real-Time Target for Release 2007b or later

Build process

TLC options:

Makefile configuration

[✓] Generate makefile

Make command: make_rtw

Template makefile: NIVeriStand_vc.tmf

Code Generation Advisor

Select objective: Unspecified

Check model before generating code: Off

[ ] Generate code only

Figure 53: Real-Time Workshop window

.

4. Click the Browse button to launch the System Target File Browser dialog box

5. Select the correct option for your target from the list for this case (Windows) select `NIVeriStand.tlc-NI Real-Time` Target for other RTOS.
   Note: If the appropriate .tlc is not visible, the MATLAB software files might be read-only, and the NI VEriStand installer is not able to provide this option. To display the option, add the following lines to the matlabrc.m file, a file that is installed by the MATLAB software:

```
addpath('X:\VeriStand');
NIVeriStandAddPaths;
```

where X is the drive letter on which you installed the NI VeriStand Model Framework.

6. Click the OK button.

7. Click the Build button in the Category section to begin building the compiled model.

   The MATLAB software command window displays the status of the build process. The following message in the MATLAB command window indicates that the Simulink Coder software has completed building the compiled model.

   ```
   ### Successful completion of build procedure for model:ModelName
   ```

Connection of DLL file with Veristand:

At minimum, an NI VeriStand project consists of the following files that you use to configure, deploy, and interact with your system [16]:

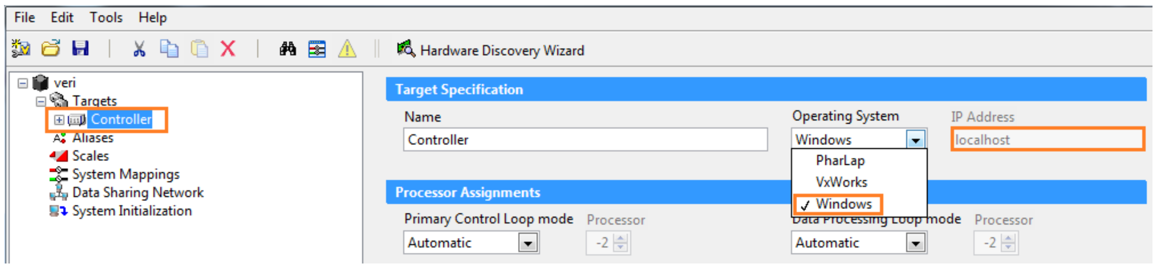
- One project file (`.nivsproj`)
- One system definition file (`.nivssdf`)
- One screen file (`.nivsscreen`)

Now that everything is set from configuring Mathworks Matlab and creating the DLL model that VeriStand support, the next step is to connect the model to the VeriStand through the following:

1. Run VeriSTAND 2013 and create a new project(named veri.nivsproj in our case) see Figure 54:

2. From Project Explorer open the System Definition File(veri.nivssdf) see Figure 55.

3. From Targets in System Explorer select Controller as demonstrated in Figure 56.



Name: Controller
Operating System: Windows
IP Address: localhost
Primary Control Loop mode: Automatic
Processor: -2
Data Processing Loop mode: Automatic
Processor: -2

Figure 56: Selecting Operating System and IP Address

Select Windows as an Operating System and localhost(127.0.0.0) as an IP Address because the device is in the same development system.

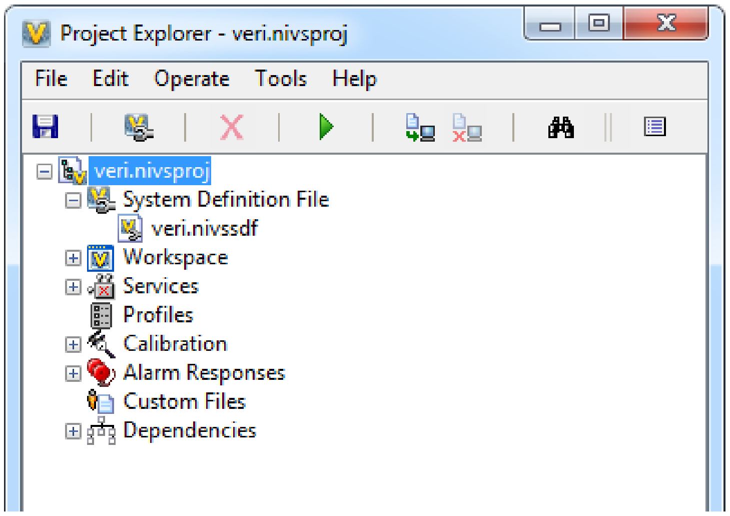
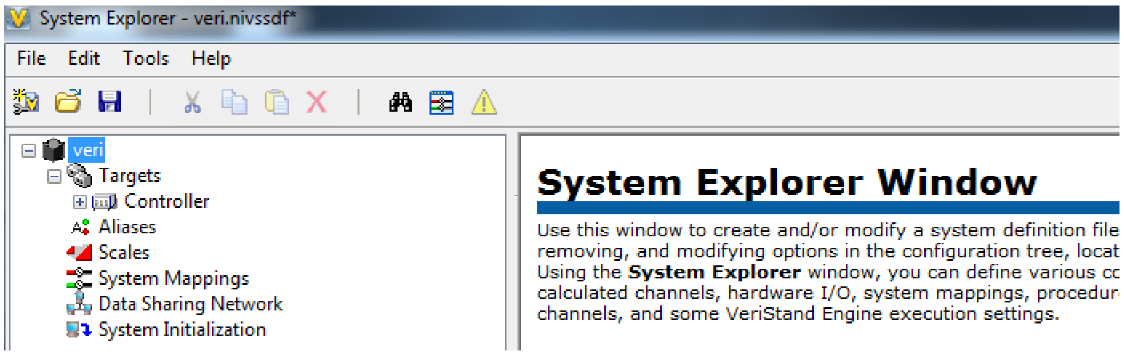


Figure 54: VeriStand Project Explorer

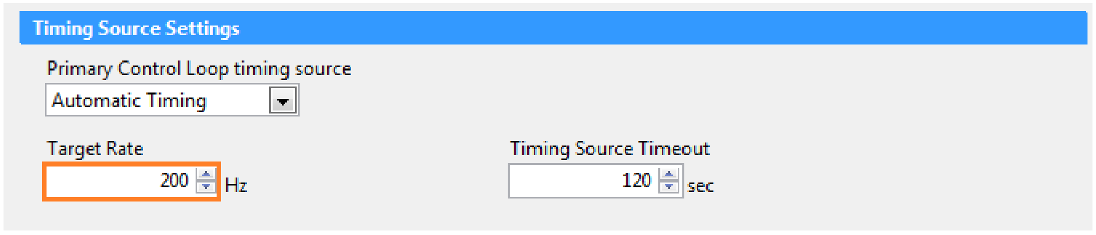


System Explorer Window

Use this window to create and/or modify a system definition file
removing, and modifying options in the configuration tree, locat
Using the **System Explorer** window, you can define various cc
calculated channels, hardware I/O, system mappings, procedur
channels, and some VeriStand Engine execution settings.

Figure 55: System Definition file

4. In the same window, set the `Target rate` to a frequency equivalent to the model sample time you choose in `Solver` tab in Simulink,in this simple case study, the target rate is set to 200 Hz which is equivalent to 0.005 sample time in Simulink `Solver` tab,as it can be seen in Figure 57


Timing Source Settings

Primary Control Loop timing source: Automatic Timing

Target Rate: 200 Hz

Timing Source Timeout: 120 sec

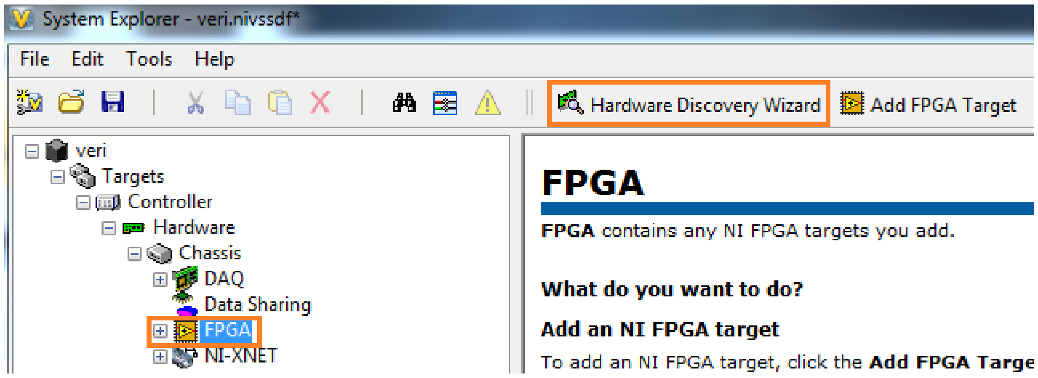
Figure 57: Add FPGA target using Hardware Discovery Wizard

The target rate is related to sample time through the following equation:

$$TargetRate = 1/SampleTime$$

Where:
TargetRate is corresponding to the Target rate in VeriStand and SampleTime is corresponding to the fundamental sample time in `Solver` tab in Simulink.

5. Expand Controller → *Hardware* and Select FPGA then Add FPGA target using Hardware Discovery Wizard,as shown in Figures 58 and 59:



FPGA

FPGA contains any NI FPGA targets you add.

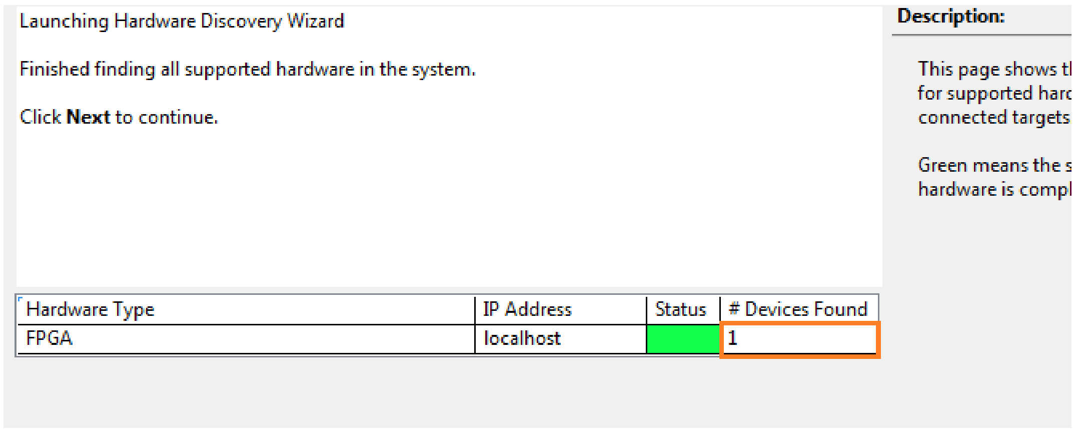
What do you want to do?

Add an NI FPGA target

To add an NI FPGA target, click the **Add FPGA Targe**

Figure 58: Add FPGA target using Hardware Discovery Wizard

The wizard starts searching for FPGA devices in the localhost(the IP Address set in Controller window before), it should find the FPGA device connected to the PC,if it is correctly installed.



Launching Hardware Discovery Wizard

Finished finding all supported hardware in the system.

Click **Next** to continue.

Description:

This page shows tl
for supported harc
connected targets

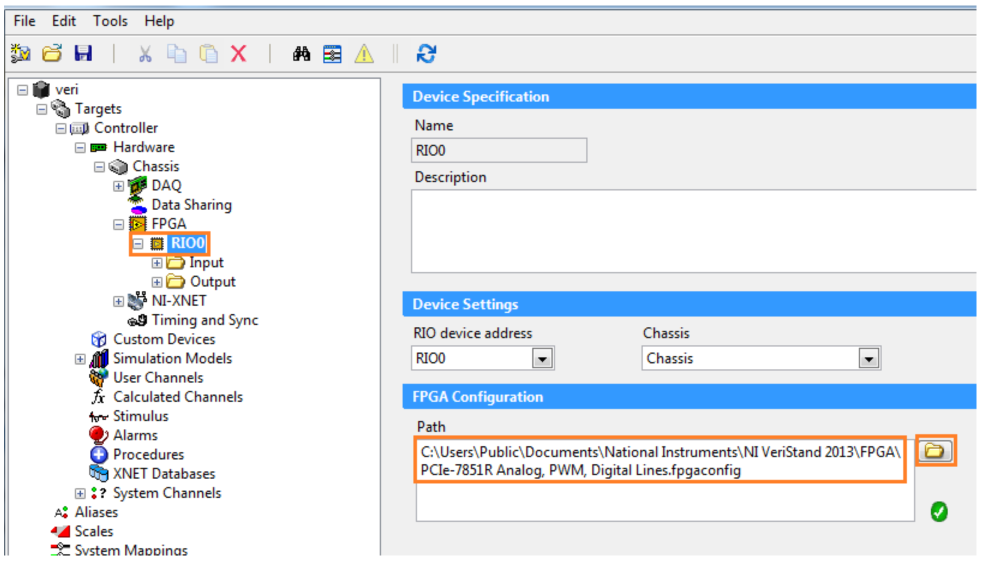
Green means the s
hardware is compl

Hardware Type	IP Address	Status	# Devices Found
FPGA	localhost	🟩	1

Figure 59: Hardware Discovery Wizard discovers the FPGA device

Then follow the handy instructions to add the detected FPGA device to the System Definition file which is in this simple case study (`RIO0`).

6. Select your FPGA Target(`RIO0`) and from FPGA Configuration (see Figure 60), select the corresponding FPGA bitfile. If you cannot find your device bitfile, you can create your own device bitfile and its accompanying FPGA configuration file.Refer to Creating a Custom FPGA Bitfile [29] and Creating a Custom FPGA Configuration File [30] on how to do it.



Name: RIO0

Description:

RIO device address: RIO0

Chassis: Chassis

Path: C:\Users\Public\Documents\National Instruments\NI VeriStand 2013\FPGA\PCIe-7851R Analog, PWM, Digital Lines.fpgaconfig

Figure 60: FPGA Configuration

7. After configuring the device in NI VeriStand, the next step is to add a model that had been created previously and map it to hardware.

   (a) Add a model to the System Definition file by right-clicking Simulation Models→ *Models* from `System Explorer` then add model,(see Figure 61).

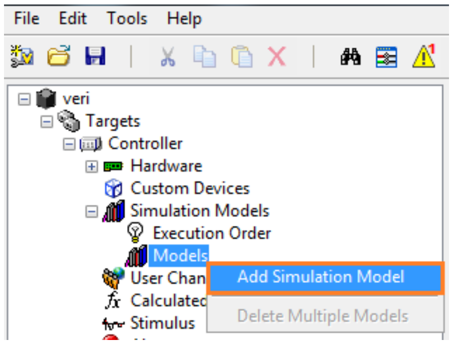
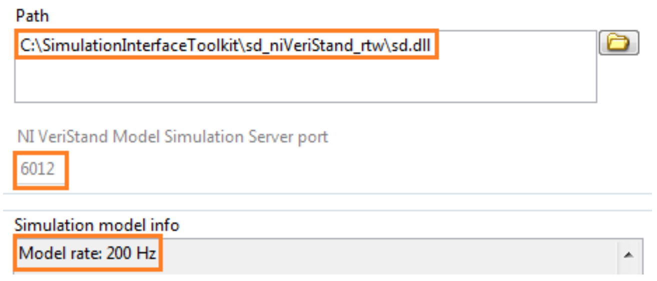


Figure 61: Add a model

   (b) Browse to the compiled model DLL created before. When you select your model's path you can observe the model rate in the Simulation info (see Figure 62), this model rate should be identical to the Target rate seen before to run your model properly.



Path
C:\SimulationInterfaceToolkit\sd_niVeriStand_rtw\sd.dll

NI VeriStand Model Simulation Server port
6012

Simulation model info
Model rate: 200 Hz

Figure 62: Browse a model

(c) After the model is added successfully, click configure mappings in tools menu, see Figure 63.

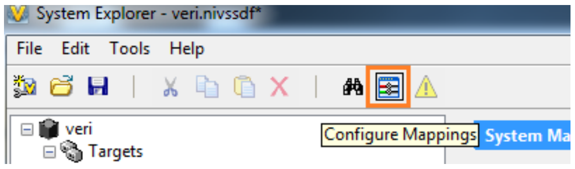
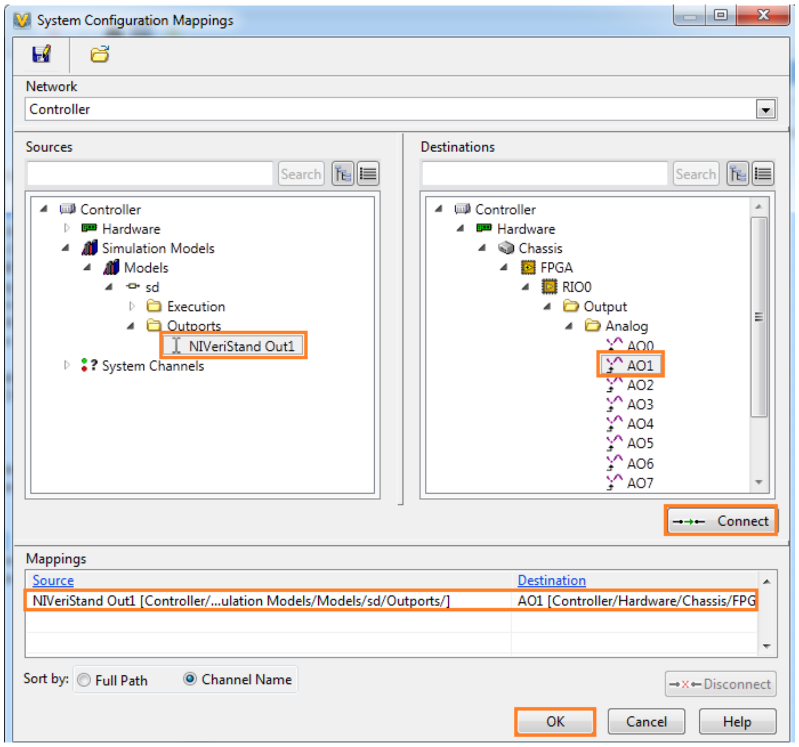


Figure 63: Select System mappings

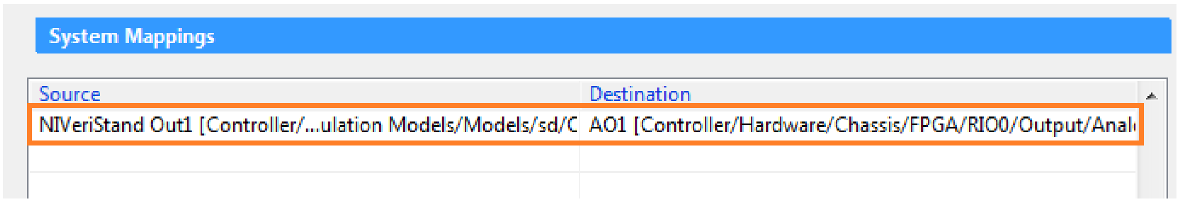
(d) Connect from the model the `NIVeriStand Out1` as a Source to `AO1` as a Destination hardware analog port as shown in Figure 64.



Source	Destination
NIVeriStand Out1 [Controller/...ulation Models/Models/sd/Outports/]	AO1 [Controller/Hardware/Chassis/FPG

Figure 64: Connect Mappings

(e) The mapping created should appear in the System mappings as can be seen in Figure 65:



System Mappings

Source	Destination
NIVeriStand Out1 [Controller/...ulation Models/Models/sd/C	AO1 [Controller/Hardware/Chassis/FPGA/RIO0/Output/Anal

Figure 65: mapping added successfully

8. From the project Explorer deploy then run your project by clicking RUN command as shown in Figure 66:

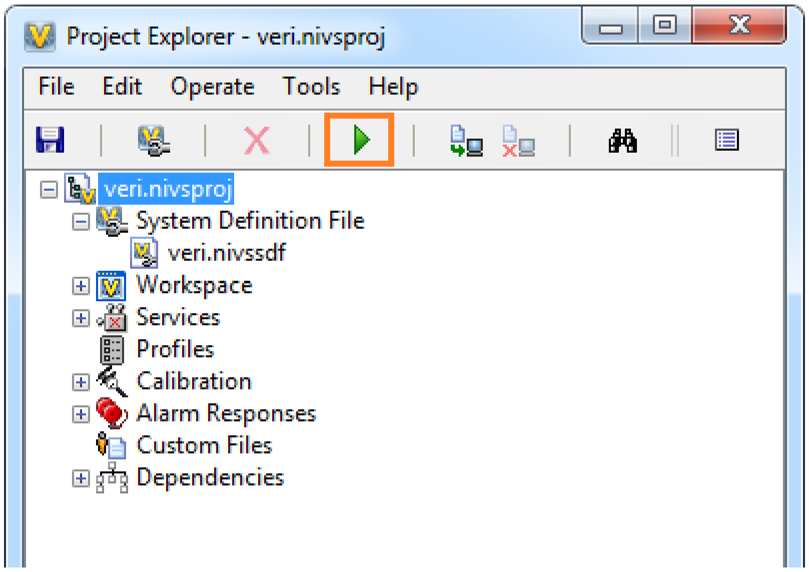
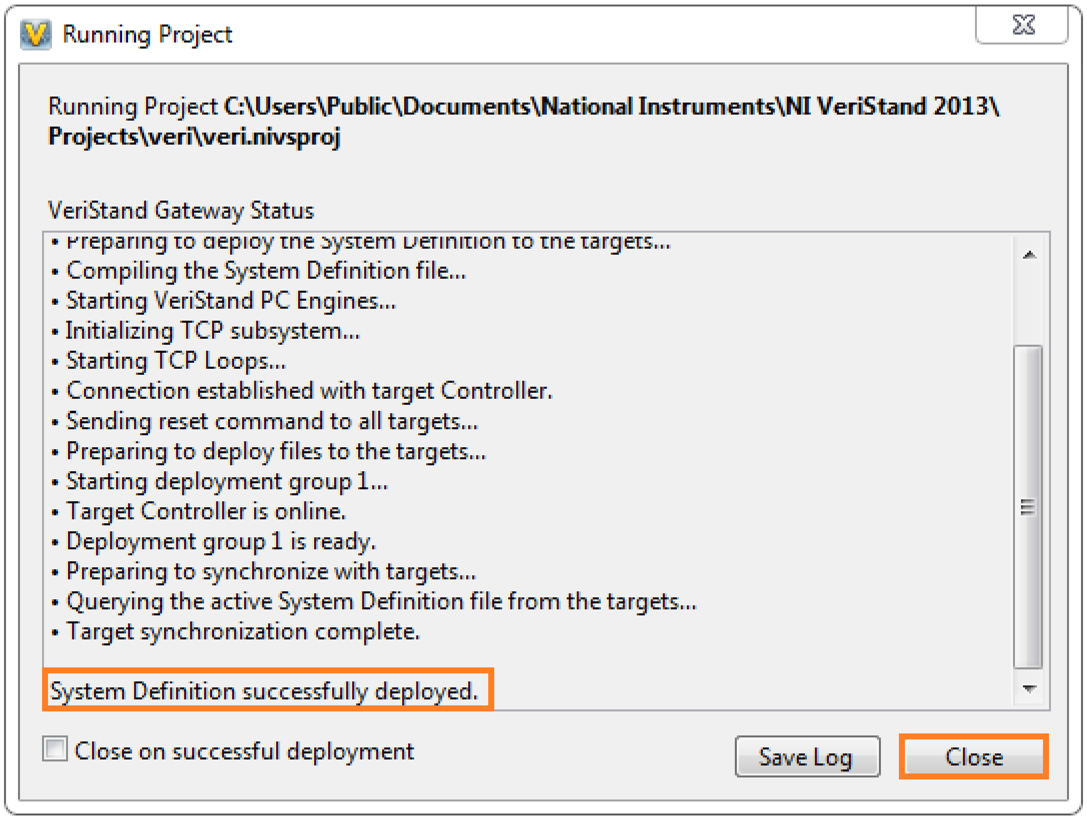


Figure 66: Run the project

Note: To deploy a system definition file to an RT target, you must first download support files for NI VeriStand to the target.

9. VeriStand starts automatically the deployment process of the System Definition file to the hardware as depicted in Figure 67:


Running Project

Running Project **C:\Users\Public\Documents\National Instruments\NI VeriStand 2013\Projects\veri\veri.nivsproj**

VeriStand Gateway Status

```
• Preparing to deploy the System Definition to the targets...
• Compiling the System Definition file...
• Starting VeriStand PC Engines...
• Initializing TCP subsystem...
• Starting TCP Loops...
• Connection established with target Controller.
• Sending reset command to all targets...
• Preparing to deploy files to the targets...
• Starting deployment group 1...
• Target Controller is online.
• Deployment group 1 is ready.
• Preparing to synchronize with targets...
• Querying the active System Definition file from the targets...
• Target synchronization complete.

System Definition successfully deployed.
```

Figure 67: Deployment process of the System Definition file

As can be seen from Figure 67, The System Definition is successfully deployed to the hardware(RIO0 FPGA target),By clicking Close button an interesting window will pop-up and this is the discussion of next subsection.

Customize The Workspace:

After Running the project a blank workspace appear automatically, to see and analyze the results, it should be edited by adding graphs,simulation control panel and other tools if needed, this can be done through the following:

1. Enter the edit mode, so you can have privileges to edit the workspace, see Figure 68.

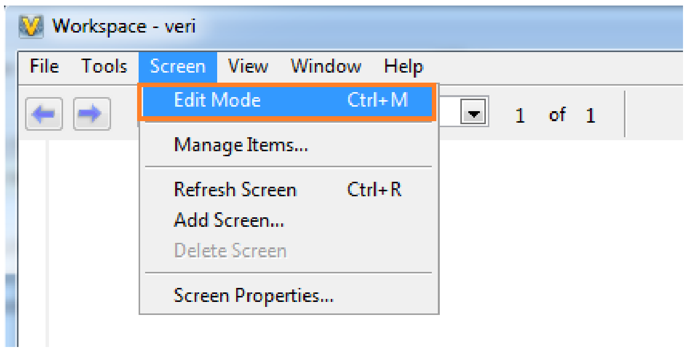


Figure 68: Switch to edit mode

2. Open the `WorkSpace Controls` panel from the right side of the screen, see Figure 69:

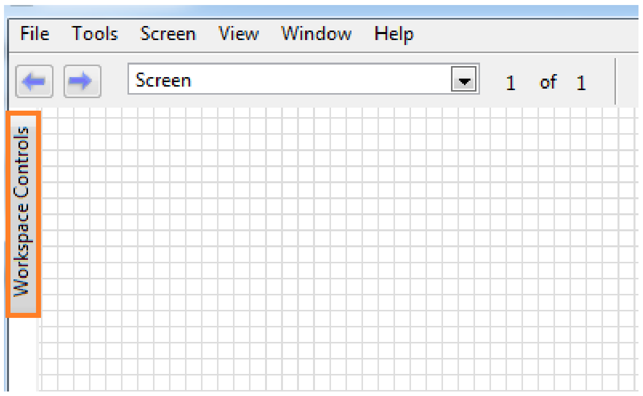


Figure 69: Show WorkSpace Controls

3. Drag a simple Graph to the screen as shown in Figure 70.

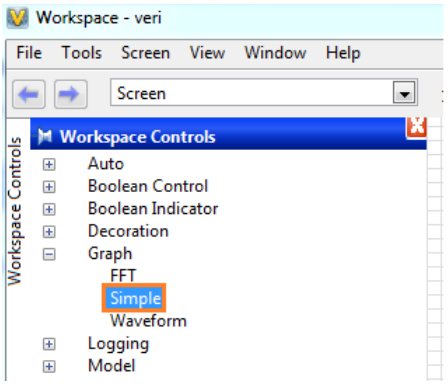


Figure 70: Simple Graph type

4. A window pop-up to select and configure channels for the graph as shown in Figure 71, expand the project to the channel that you want to add to the graph,

then add arrow.you can add as much channels as you want to see them in the same graph.

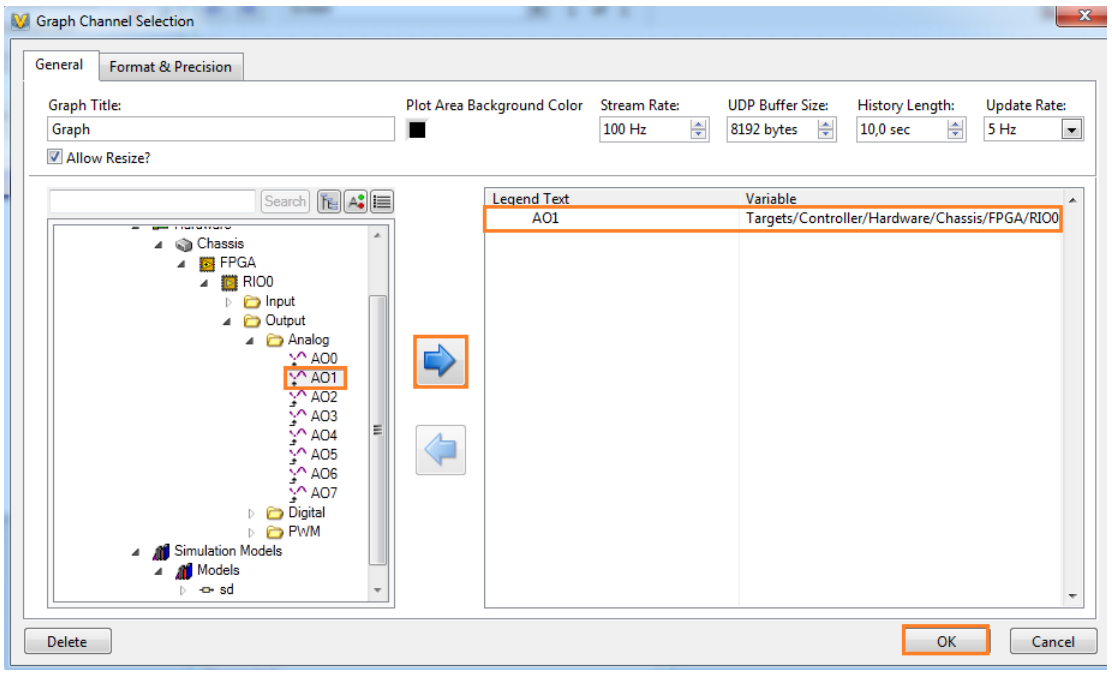


Figure 71: Adding channels

5. You can drag Model Control to the screen, in order to control the simulation and track it, you can do that by:

   (a) Drag the Model Control from the `WorkSpace Controls` to the screen as seen in Figure 72:

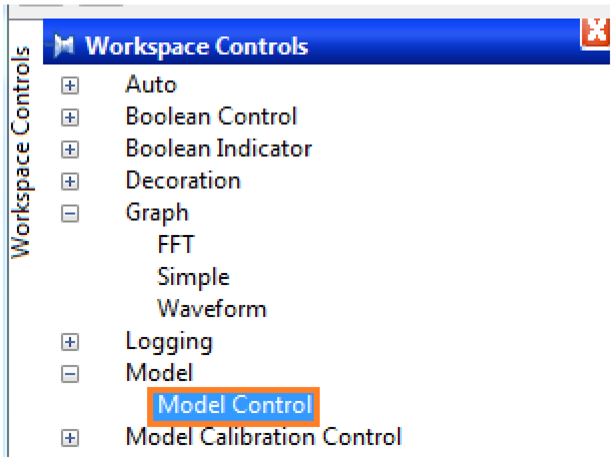


Figure 72: Model Control

   (b) Choose your model from the pop-up window, see Figure 73:

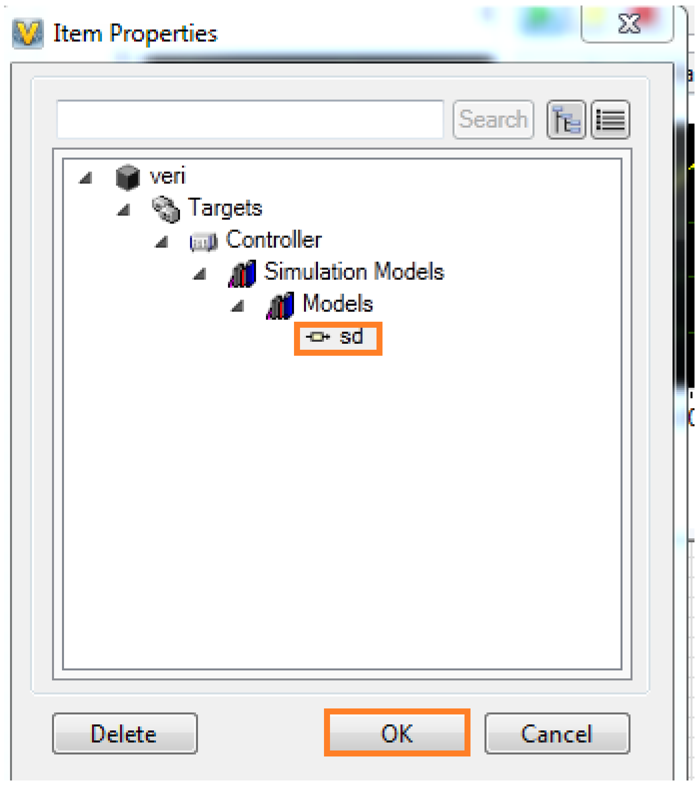
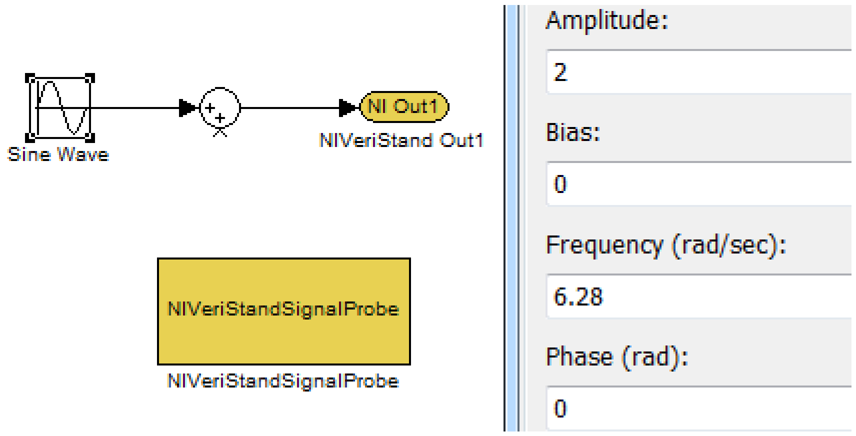


Figure 73: Choosing the model

6. After setting everything described previously, you can see the sinewave generated from the Simulink Model whose values depicted in Figure 74:



Amplitude:
2
Bias:
0
Frequency (rad/sec):
6.28
Phase (rad):
0

Figure 74: sinewave values in Simulink Model

The frequency 6.28 rad/sec is equivalent to 1 Hz.
The generated Simulink sinewave appear in the graph as can be seen in Figure 75, and also appear in the Oscilloscope as shown in Figure 76.

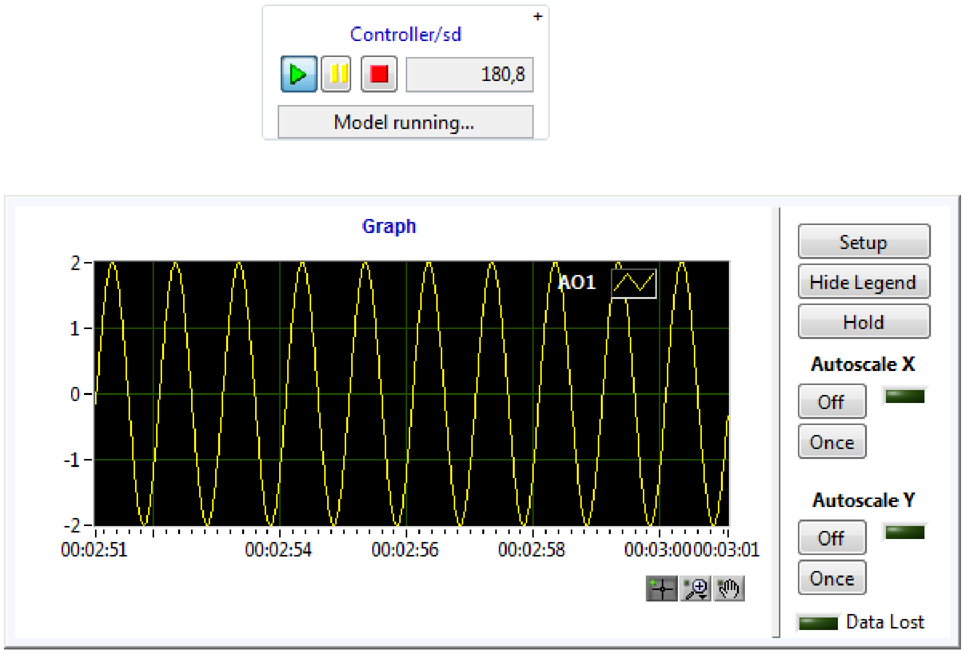


Figure 75: sinewave generated from Simulink Model appear in graph

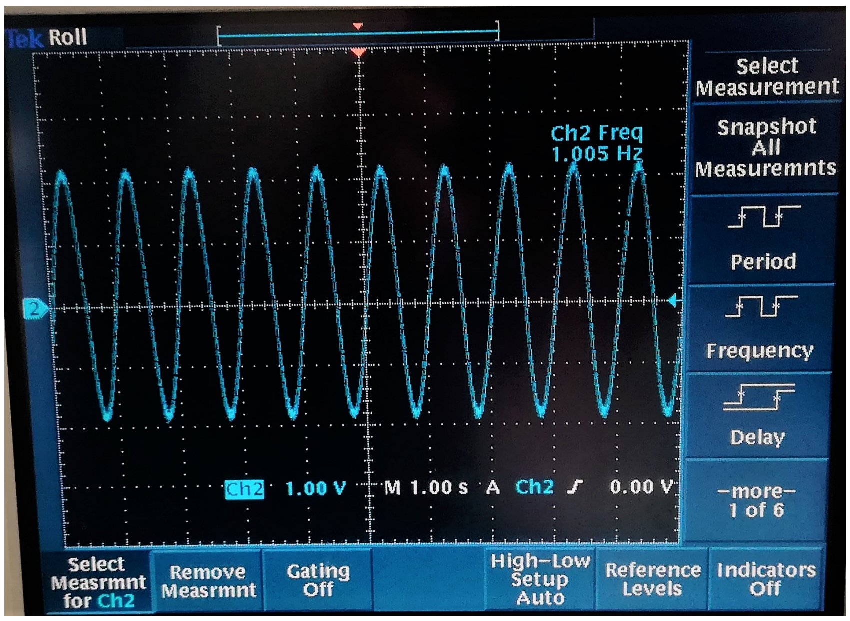


Figure 76: sinewave generated from Simulink Model in Oscilloscope