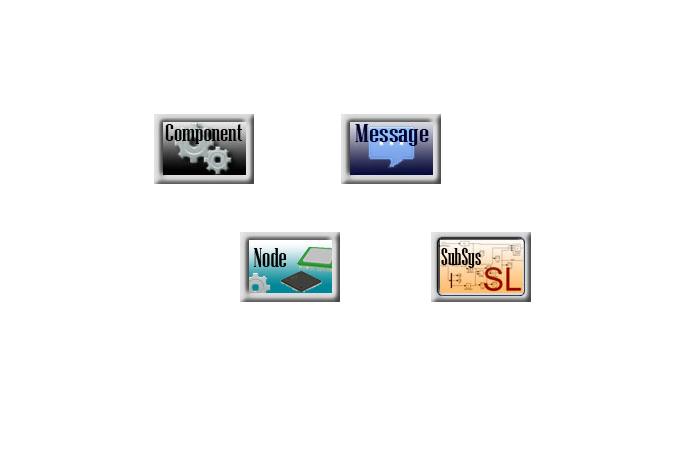
October 2010

Version 1.2

The ESMoL Toolchain User’s Manual

A helpful little guide.



by

Graham Hemingway, [Gabor Karsai](mailto:gabor.karsai@vanderbilt.edu), Nicholas Kottenstette, Harmon Nine, Joe Porter, [Janos Sztipanovits](https://wiki.isis.vanderbilt.edu/hcddes/index.php/Janos_Sztipanovits) (PI), [Chris vanBuskirk](mailto:Chris.vanBuskirk@vanderbilt.edu) and [project alumni](https://wiki.isis.vanderbilt.edu/hcddes/index.php/VandyAlumni)

**Table of Contents**

[Introduction 4](#_Toc275172933)

[Prerequisite Software 4](#_Toc275172934)

[ESMoL Installation & Configuration 4](#_Toc275172935)

[The Windows Installer 4](#_Toc275172936)

[Co-Simulations 5](#_Toc275172937)

[Initial TrueTime Setup 5](#_Toc275172938)

[The Toolchain Components and File Formats 6](#_Toc275172939)

[mdl2mga 7](#_Toc275172940)

[Stage1 8](#_Toc275172941)

[Stage2 8](#_Toc275172942)

[SchedTool 9](#_Toc275172943)

[SchedResults 9](#_Toc275172944)

[Schedule Input File Format 10](#_Toc275172945)

[Schedule Output File Format 11](#_Toc275172946)

[Deployment Platforms 14](#_Toc275172947)

[FRODO Time-Triggered Architecture 14](#_Toc275172948)

[Contact Information 15](#_Toc275172949)

[Appendix I: Provided Example Datasets 16](#_Toc275172950)

[Example 1: The Quad Integrator Model 16](#_Toc275172951)

[Example 2: The Quad Rotor Model 34](#_Toc275172952)

[Appendix II: Developing ESMoL 34](#_Toc275172953)

[Prerequisite Software 35](#_Toc275172954)

[Source Code Installations 35](#_Toc275172955)

[3rd Party Software Libraries 36](#_Toc275172956)

[Environment Variable Settings 37](#_Toc275172957)

[Application Compilation 38](#_Toc275172958)

# Introduction

Welcome to the ESMoL (Embedded Systems Modeling Language) tool suite. If at any point you need assistance, please first double-check your configuration. Configuration help beyond this document can be contacted via the contact information listed at the end of this document.

# Prerequisite Software

Several applications and toolkits must be installed by the user before the HCDDES environment can be configured. These applications are:

* Windows XP Service Pack 3
* Visual Studio 9.0 (a.k.a. 2008) – Professional Edition *(*not *tested with Express Edition)*
* MATLAB with Simulink *(tested with edition R2010a)*
* GME version 10.2.9

The following packages are provided by the installer, and are listed here only for reference:

* UDM Toolkit, Release 3.2.6
* GREAT Toolkit, Release 1.7.4
* Python 2.6.5 w/ the Cheetah template engine
* Boost python version 1.42.0

Please read and follow the installation and configuration instructions that come with each of these packages.[[1]](#footnote-1)

# ESMoL Installation & Configuration

## The Windows Installer

As a convenience, we provide a binary installer, which automates the bulk of the tedious installation and configuration steps. Unfortunately, the current edition of the Windows installer has not been tested with Windows Vista or with Windows 7, and it is expected that there could be some minor configuration issues on these platforms. Future versions of the installer are expected to support Windows 7, additionally.

**Step 1: Run the installer.**

The installer filename usually has the form Setup\_ISIS-ESMoL\_v10.10.18.1611.zip. Rename it to Setup\_ISIS-ESMoL\_v10.10.18.1611.exe, and then run the installer.

**Step 2: GME version warning**

The installer may complain about versions of GME which are higher than 10.2.9. It’s OK to continue, unless your version is older than 10.2.9.

**Step 3: Introduction, license agreement, and information.**

These are fairly standard.

**Step 4: Select Destination Location.**

The installer allows the user to select a destination location, but alternate locations have not been tested. The default is strongly recommended.

**Step 5: Install**

Copies files and establishes the configuration.

When installing a binary release, the toolchain will be placed in C:\Program Files\ISIS\ESMoL\ by default. The minimal footprint of related, 3rd party dependencies will reside in the ‘lib’ subdirectory therein.

## Co-Simulations

The ESMoL toolchain supports simulation of platform effects such as limited computational and communication resources by way of the TrueTime hardware/software co-simulation toolbox for Simulink. TrueTime does require some initial, minimal setup procedures, however, which is summarized here. For a more complete reference, please consult %HCDDES\_PATH%\lib\truetime-2.0-beta4\docs\report.pdf.

### Initial TrueTime Setup

1. The HCDDES installer has placed the TrueTime library in %HCDDES\_PATH%\lib and defined the required %TTKERNEL% variable to point to this toolbox for you.
2. Open MATLAB, and add the following relevant TT directories to your path[[2]](#footnote-2).

* addpath([getenv(’TTKERNEL’)])
* addpath([getenv(’TTKERNEL’) ’\matlab\help’])
* addpath([getenv(’TTKERNEL’) ’\matlab’])
* mex –setup (make sure to select ‘Microsoft Visual C++ 2008 SP1…”)
* truetime (this opens the TrueTime library elements)

1. **Make the MATLAB path changes permanent.**

* From the Matlab prompt, type userpath.
* Select one of the listed paths (there may only be one), and in that directory create a file called ‘startup.m’.
* Copy the three “addpath([getenv(…” commands above into the startup.m file. Save startup.m and close it.
* Restart Matlab, and type path to check that the three TrueTime paths are now in the list.
* Execute the truetime command to make sure that the library is accessible.

# Using the Tools

In this section we discuss the practical use of the ESMoL tools.

## Executing TrueTime Models

Once a TrueTime model has been generated and compiled (see sections 10, 11 and 12 in Example 1 below), you can execute the model to simulate possible platform effects introduced by deploying your controller onto a time-triggered architecture. You execute the model just like any other Simulink model, by setting the simulation’s duration and pressing the play button. In order for the model to execute effectively, are few simple rules-of-thumb should be followed:

1. Execute the model in Simulink with Accelerator enabled. TrueTime models have been tested using both Normal and Accelerated modes without issue.
2. Do not alter the default parameters for zero-threshold crossing. While changing these parameters may speed up the execution of the model they will alter the behavior and may cause incorrect phenomena to be introduced into the simulation.

Once your model is done executing you can view the results using standard Simulink scopes and blocks. Key scopes to look at are those attached to the sched output of the TrueTime kernel blocks. These scopes will visualize the execution trace of all tasks created and executed on that kernel. It is straightforward to understand the execution schedule by looking at the scope.

## Debugging TrueTime Models

It may be necessary at times to gain a deeper understanding of how the TrueTime model is operating than what is available via the normal Simulink model. Since we are using compiled C-code to manifest the internal operations of the TrueTime blocks we can use the Visual Studio debugger to step through the C-code as needed. Here are the steps necessary to debug TrueTime using Visual Studio:

1. Get your model ready to run in Simulink, but don’t run it quite yet.
2. Start Visual Studio and open the KERNELNAME\_init.cpp file for the TrueTime kernel(s) you want to debug.
3. Under the Tools menu in Visual Studio, select “Attach to Process” and choose Matlab from the list that appears.
4. Set some breakpoints in the \*.cpp file so that once execution begins it will halt at those breakpoints.
5. Finally, run the model simulation in Simulink. When the model get to the point of executing the code you want to debug it will hit the breakpoints you set in Step 4 and you can begin debugging the model in Visual Studio.

To finish execution of the simulation just disable or delete any breakpoints that might halt execution and click the Run button in Visual Studio.

# The Toolchain Components and File Formats

This section introduces the functional components of ESMoL and summarizes the overall workflow for the ESMoL toolchain. The toolchain is designed to serve as a vehicle for integrating the Simulink/Stateflow environment with various V&V tools and for the model-based development of embedded software, including generation of executable code from the graphical and textual models.

ESMoL is based on the MIC toolset, including the metaprogrammable Generic Modeling Environment (GME), as well as other tools, such as UDM (Universal Data Model), and GReAT (Graph Rewrite and Transformation). GME is a generic, programmable model editor tool, UDM is a package that provides access to models stored in various formats (e.g. GME binaries, called .mga files, or XML files) using various ‘back-ends’, and GReAT is tool for developing model transformation tools using graph transformation techniques. These MIC tools have been instantiated and configured to support the domain-specific language(s) used in the ESMoL toolchain, but through their open API-s, further extensions to the toolchain are also possible.

The toolchain includes the following major elements.

1. A visual modeling language and dedicated editing environment (based on GME) for storing and manipulating models imported from the Simulink/Stateflow environment. The visual modeling language is a superset of the Simulink/Stateflow and it has extensions for the componentization of functional models, for the modeling of software architecture that will utilize the functional blocks, as well as for the allocation of components to tasks on a real-time platform.
2. An import translator that allows importing the Simulink/Stateflow models into the GME tool. This tool is the gateway for Simulink/Stateflow, as it builds a GME representation of those models.
3. Code generators that generate executable code (e.g. C, Java and MATLAB) from the visual models. This code can then be subjected to various V&V analysis and/or deployed onto target hardware. The generators have an open back-end and the generation of executable code in other language(s) (e.g. Ada) is feasible. Also, a separate code generator that generates code from Embedded MATLAB scripts (using the same back-end) is available.
4. A number of utility tools for converting models between various (intermediate) formats, as well as for performing other analysis and model manipulation activities.

## mdl2mga

The mdl2mga.exe utility imports and updates individual Simulink models into larger ESMoL models.

Usage:

**mdl2mga [options] <mdl-file> [ <mga-file> | <xml-file> ]**

Allowed Options:

* --help : Produce help message
* -n [ --newfile ] : Create new MGA/XML
* -s [ --substitute ] arg : Substition table spec
* -t [ --toplevel ] : Only translate top-level Simulink blocks
* -L [ --libdir ] arg : Specify directory in which to search for MATLAB libraries
* -h [ --host ] arg : Specify host on which to launch MATLAB for translation

Required Arguments:

* <mdl-file> : the name of a Matlab .mdl file

If the second argument is not specified, it defaults to a file that has the same name as the .mdl file, but ending in .xml -- see below for a description of this argument.

Optional Arguments:

* <mga-file> is the name of an GME-based .mga model file that will be created or overwritten, and will conform to the ESMoL paradigm in GME.
* <xml-file> is the name of an XML-based (.xml) model file that will be created or overwritten, and will conform to the ESMoL paradigm as specified in the ESMoL.xsd XML schema file. This schema file is embedded in this program and so does not have to be explicitly specified.
* <directory> is the name of a directory that is given to Matlab to search for SL/SF library files. Such library files may be used by the <mdl-file> in specification of its SL/SF model. As many of these directories may be specified as needed, though each with its own '-L' option specifier.

*NOTE*: the directory in which the <mdl-file> resides is always implicitly used by this program as a library search path to MATLAB.

*NOTE*: If Matlab is slow to start, then MDL2MGA may time out. An easy solution is to start Matlab completely once to get it into memory before running MDL2MGA.

* <host> is the hostname or ip-address of a machine with a Matlab installation. MDL2MGA will direct this machine to process the <mdl-file>, and will then query this same machine about the model therein. By default, MDL2MGA uses the local machine.

## Stage1

Stage1.exe first transforms the original model from a domains-specific language, which is targeted at embedded systems hardware and software engineers, into an abstract representation more amenable to parsing by computerized model reasoners and translators.

The supported command-line switches are as follows:

* -h [ --help ] : Usage information
* -f [ --input-file ] <<model.xml>> : Input file
* -o [ --output-file ] <<arg>> : Output filename base
* -c [ --context ] : Show translation context trace
* -d [ --debug ] : Show some debugging information
* --graph : Generate the dataflow graph of the model

## Stage2

Initially, the Stage2.exe tool is used to generate a scheduler input file from an ESMoL\_Abstract model. This input file is processed by SchedTool to perform schedulability analysis (i.e. next section). The results which are finally merged back into the original model(s) in a subsequent execution phase by the SchedResults utility.

Additionally, Stage2 is used again further down the processing pipeline with different –translate options in order to perform other translation tasks such as generating code from the ESMoL\_Abstract representation.

In general, the supported options for Stage2 are as follows:

* -h [ --help ] : Usage information
* -r [ --res ] <<arg>> (default=5us) : Scheduler resolution
* -f [ --input-file ] <<model.xml>> : Input file
* -o [ --output-file ] <<arg>> : Output filename base
* -p [ --directory-name ] <<arg>> : Generated files directory
* -t [ --translate ] <<arg>> : Translator(s) to use
* -l [ --list ] : List available translators (and exit)
* -d [ --debug ] : Print debugging information.

## SchedTool

The scheduler is a command line tool called SchedTool.exe. This scheduler takes as input a file (typically denoted with a .scs suffix) created in the previous section and from that generates a valid schedule. The schedule is output into a second file (typically denoted with a .rslt suffix).

SchedTool.exe has a few command-line switches. They are:

* -h : This switch asks the tool to output its command-line options, just like is being done now. If used, this optional switch overrides all other switches.
* -i <<filename>> : This switch specifies which file should be used as the input file. This switch is optional. If not specified, the tool looks for a file named “sched.scs”.
* -o <<filename>> : This switch specifies what the output filename should be. This switch is optional. If not specified, the output file will be called “result.rslt”.
* -d <<list of entities to debug>> : This switch enables debugging information from the scheduler for the entities listed. As the scheduler encounters these entities it will provide additional debugging information via STDOUT. This is an optional switch. Note (10/2010) debugging functionality may not function as expected.
* -v <<bool>> : This switch enables verbose output during schedule generation. This switch is optional but enabled by default.

An example invocation of this tool may look like this:

**SchedTool.exe –i sched.scs –o output.rslt –v false –d entity1 entity2**

Additional information about the HCDDES scheduler and its detailed operation are provided in the included reference materials located at %HCDDEs\_PATH%\doc\papers\.

## SchedResults

As mentioned previously, the resulting schedule information should be inserted back into the ESMoL and ESMoL\_Abstract models from which the scheduling information was derived. Therefore, the SchedResults.exe command-line tool, is used to weave these computed schedules back into the original model(s).

SchedResults.exe also has a few command-line switches. They are:

* -h : This switch asks the tool to output its command-line options, just like is being done now. If used, this optional switch overrides all other switches.
* -r <<filename>> : This switch specifies which file should be used as the input file. This switch is optional. If not specified, the tool looks for a file named “result.scs”.
* -e <<filename>> : This switch specifies which ESMoL model file should have these results added to it. This switch is optional. If not specified, the tool will not look for an ESMoL model to annotate.
* -a <<filename>> : This switch specifies which ESMoL\_Abstract model file should have these results added to it. This switch is optional. If not specified, the tool will not look for any ESMoL\_Abstract model to annotate.
* -d : This switch enables verbose output during execution of the tool. This switch is optional and enabled by default for the Debug build and disabled for the Release build.

The SchedResults tool is quite straightforward and little else needs to be said about its execution or use. An example invocation of this tool may look like this:

**SchedResults.exe –r results.scs –e quad\_rotor.mga –a quad\_rotor.xml –d**

## Schedule Input File Format

The input file to the scheduler must be in a specific format in order to be useable by SchedTool. This format is formally described in a format Antlr3 grammar and can be found in src\SchedTool\sched.g. To augment this description, this document outlines the primary components of any schedule input file. Additionally, a number of example scheduler input files can be found in archive\Scheduler\Test.

The possible components of the scheduler input file are:

1. The % character denotes the start of a comment which continues until the end of the line. It cannot be used in the name of any identified object. Object names must start with a letter (‘a’…’z’ and ‘A’…’Z’) and be followed with any further combination of letter, number or the ‘\_’ character.
2. The first non-comment line must be the resolution declaration:

Ex: **Resolution 1us**

In this example, the resolution is one microsecond. All time values in the scheduler input file can have one of the following units: ms, us, ns, ps, fs, or s. The resolution time can be any integer or floating point value followed by a time unit. The resolution declaration is a global value and is used across all schedulable objects within the model.

1. The next declaration is for a processor and its associated components or component groups. Typically there are one or more Processor blocks in a scheduler input file depending on the number of processors in the system being modeled.

Ex: **Proc GS 100MHz 1us 1us**

**Comp InnerLoop =10Hz 10ms**

**Comp OuterLoop =20Hz 5ms**

**CompGrp DataHandling =10Hz 5ms**

**CompGrp SensorData =20Hz 5ms**

In this example the processor is named GS, runs at 100MHz, has a XXX of 1us, and a XXX of 1us. The GS processor has four schedulable objects associated with it named: InnerLoop, OuterLoop, DataHandling, and SensorData. The first two, IA and IB, are individual software components, denoted with by Comp. The second two are groups of components that all share the same schedule. Such groups are denoted by CompGrp. Each schedulable object is given a unique name, a frequency and a worst-case execution time (WCET). All frequency values in the scheduler input file can have one of the following units: GHz, MHz, kHz, or hz. The frequency is composed of any integer or floating-point value followed by one of those frequency units.

1. The next declaration is for a bus and its associated messages or message groups. Typically there are one or more Bus blocks in a scheduler input file depending on the number of busses in the system being modeled.

Ex: **Bus I2C 10Mb 1us**

**Msg M1 2kb GS/InnerLoop RS/OtherLoop**

**Msg M2 10kb RS/OtherLoop GS/OuterLoop**

**MsgGrp M3 3kb GS/DataHandling RS/GPSMgr**

In this example the bus is named I2c, has 10Mb of bandwidth, and a latency of 1us. There are three schedulable objects on this bus named: M1, M2, and M3. Each object is given a unique name, a message size, and sending and receiving components. The name of both the sender and the receiver is composed of the processor name and component (or component group) separated by a ‘/’.

1. The final possible declaration block in a scheduler input file is for latency.

Ex: **Latency 1us GS/OuterLoop RS/OtherLoop**

**Latency 5us RS/OtherLoop GS/DataHandling**

Each latency declaration is independent from any other. Each declaration is given a time value and two fully named components.

## Schedule Output File Format

The output file from the scheduler contains all of the scheduling information for all of the components, component groups, messages, and message groups. Because of its simpler structure as compared to the input file, there is no Antlr3 grammar for the input file. Similar to the input file, a number of sample output files can be found in the archive\Scheduler\Test directory.

The structure of the scheduler output file is:

1. The first line must be the hyperperiod declaration.

Ex: **Hyperperiod 20ms**

In this example, the hyperperiod for the entire system is 20ms. In the output file, all time value formats and units are identical to those found in the input file.

1. The remaining blocks are all similarly composed. Each block corresponds to either a processor or bus declared in the input file.

Ex: **GS:**

**GS/OuterLoop\_0 3.0**

**GS/InnerLoop\_0 4.0**

**GS/SensorData\_0 5.0**

**GS/InnerLoop\_1 10.0**

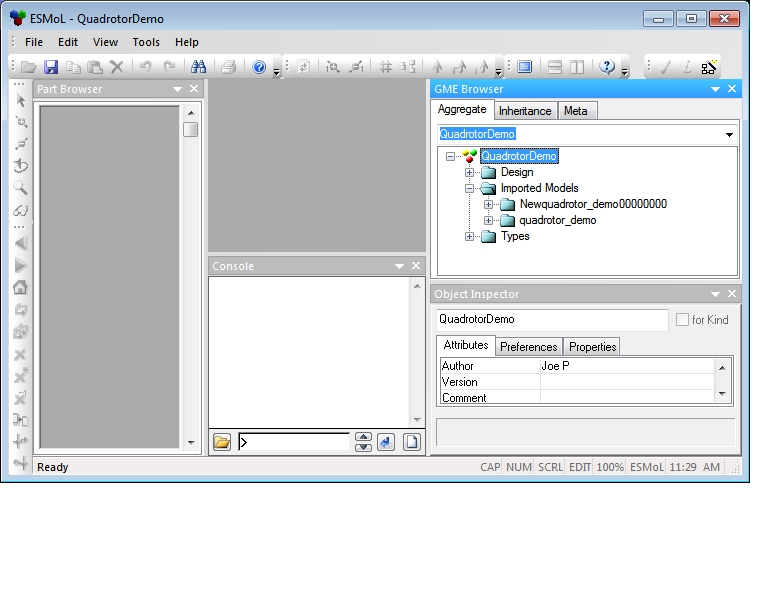
**GS/InnerLoop\_2 18.0**

This example uses the GS processor discussed in the input file section. The block begins with the name of the processor (or bus in such a case) followed by a colon. Then each line after that is one execution instance of a component (or message). The full name of the component (prefixed with the name of the processor) is used but an additional suffix is also added. An underscore and number are appended to each component name. The number in the suffix corresponds to the number of times that component has been invoked. For example GS/OuterLoop\_2 is the second instance of the OuterLoop component. A time value is also given to each instance and should be in ascending order throughout the overall block.

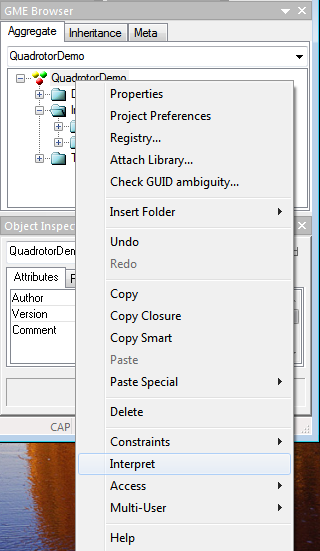
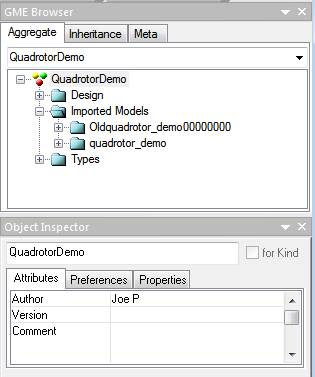
There are example models included in the %HCDDES\_PATH%\examples directory. Please explore these and consult the appendices of this document for further instructions about how to use the HCDDES tool chain with the provided examples[[3]](#footnote-3).

## ESMoLUpdate

MDL2MGA may be used to create a new model or to import a Simulink specification into an existing model. The toolchain provides a tool (ESMoLUpdate) for migrating the design model references from an older Simulink dataflow to a newly imported and updated dataflow model. The tool is integrated into the graphical modeling environment as shown here:



In the figure the Imported Models folder contains the original Simulink design model (quadrotor\_demo) and a newly imported model (Newquadrotor\_demo00000000). Invoking the interpreter (as shown in the pop-up menu below) will update all of the references from the ESMoL design that point into quadrotor\_demo to point to Newquadrotor\_demo00000000. After running the updater, Newquadrotor\_demo00000000 will be renamed to quadrotor\_demo and quadrotor\_demo will be renamed to Oldquadrotor\_demo00000000. To get to the interpreter, right-click the QuadrotorDemo entry in the model tree.

# Deployment Platforms

## FRODO Time-Triggered Architecture

In a typical experimental setup, the deployment hardware for our models consists of a Gumstix Verdex embedded processing module, with a Robostix I/O expansion board. The Gumstix is a PXA-based processing board with both wired and wireless Ethernet as well as expansion connectors. The Robostix has an ATmega 128 microcontroller with numerous ports for collecting sensor data or providing actuation signals. One Robostix UART is dedicated to the ’plant’ data connection (see below). An I2C bus links the Gumstix with the Robostix, which exchange sensor, state, and control data. The ESMoL model of the platform is appropriately simple (Fig. 7).

On this hardware, we realize a simple, portable software implementation of the common *time-triggered architecture* (TTA) in the form of a layer code called FRODO[[4]](#footnote-4). The FRODO virtual machine (VM) implements the time- triggered execution of tasks, and has a communications layer which handles the buffering and timed transfers of data messages. The current version runs on generic Linux (on the Gumstix) as well as on FreeRTOS (for the AVR). If a new processor or OS is used, the VM may be easily ported to that new platform[[5]](#footnote-5).

# Contact Information

The Vanderbilt Institute for Software Integrated Systems provides limited support for the HCDDES toolkit. Please contact us if you encounter issues or questions during the installation or configuration of your environment:

The HCDDES Project Team

c/o: VUSE-ISIS Building

1025 16th Ave. South

Nashville, TN 37212 (615) 343-7472

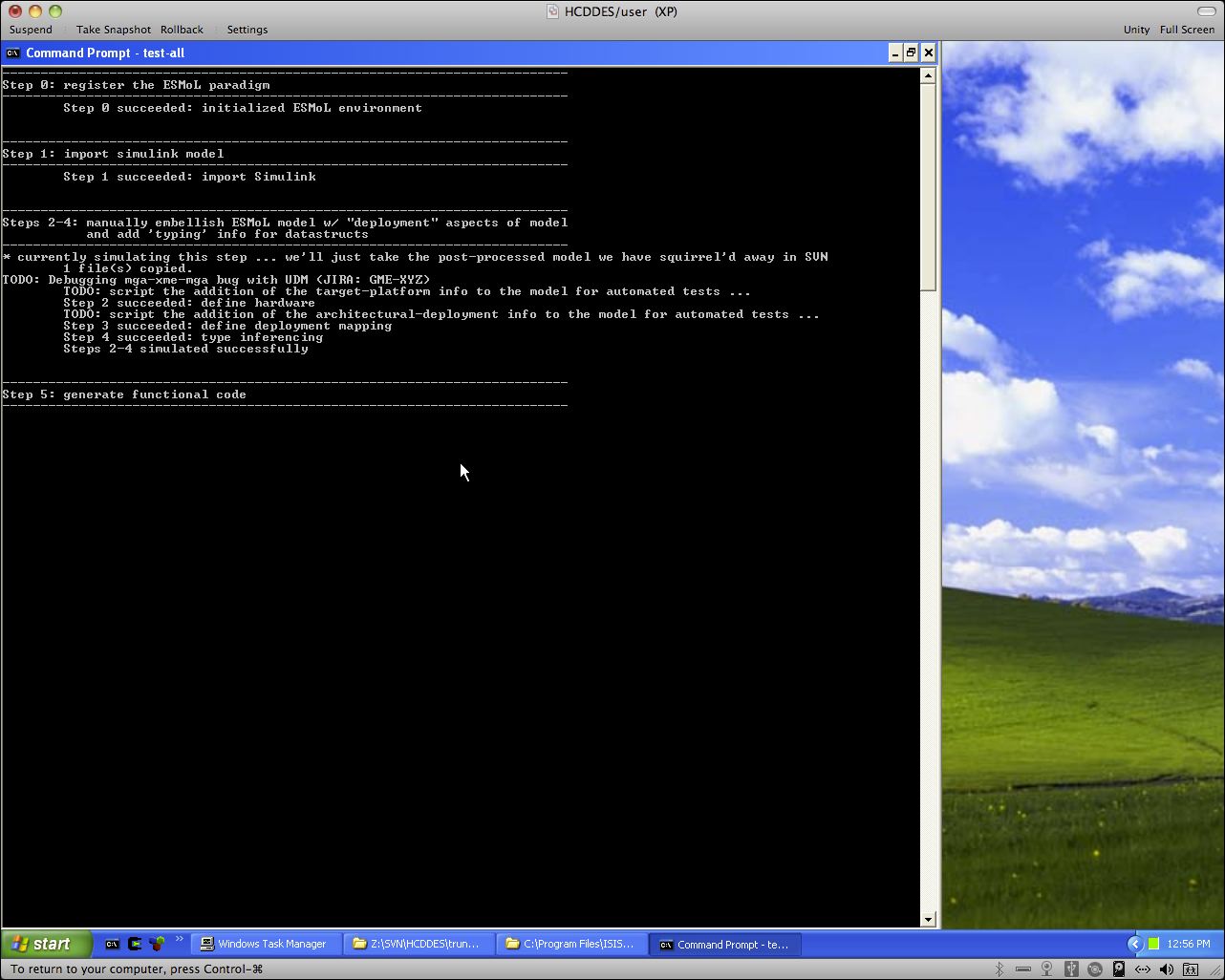
escher@isis.vanderbilt.edu

<https://wiki.isis.vanderbilt.edu/hcddes/index.php/Main_Page>

# Appendix I: Provided Example Datasets

## Example 1: The Quad Integrator Model

A single script named %HCDDES\_PATH%\examples\example1\tests\test-all.cmd automates the entire procedure that follows.

**Figure**: the `test-all` script in progress

**Step 1** – Import the Simulink model into ESMoL

Find the quad\_integrator\_subsys.mdl example Simulink model in this example’s directory structure[[6]](#footnote-6). From the command line, use the MDL2MGA tool to translate this Simulink model into an ESMoL model.

**`MDL2MGA quad\_integrator\_subsys.mdl QuadIntegrator.mga`**

A new file, called QuadIntegrator.mga, will be created. If QuadIntegrator.mga already exists then the Simulink model will just be incorporated into the existing file. Go ahead and open QuadIntegrator.mga in GME. If QuadIntegrator.mga was just created you should see only two folders beneath the model’s Root Folder.

1. The first folder is Imported Models
2. The second folder is Types

**Step 2** – Define the *Components* and *Message Types* in ESMoL

The first step in creating a fully specified ESMoL model is to create all of the Component and Message types that will be used throughout the model. Component types are typically derived from Simulink Subsystems.

Insert a DesignFolder and call it “Component Definitions”.

Add a SystemTypes sheet and call it “Component Types”.

Add five *messages* to “Component Types”: pos\_msg, ang\_ref, thrust\_commands, sensor\_data, pos\_ref

In pos\_msg add two values: pos and pos2

In ang\_ref add two values; vel and ang

In thrust\_commands add three values: ang1, ang2, ang3

In sensor\_data add one value: sensor\_data

In pos\_ref add one value: pos\_ref

Add three *components* to “Component Types”: InnerLoop, OuterLoop, and DataHandling

In *OuterLoop* add reference to the Simulink Subsystem called ‘OuterLoop’ (click drag with Control+Shift). Also add references to pos\_msg and ang\_ref. Wire everything together. Pos\_ref goes to OuterLoop and OuterLoop goes to ang\_ref:

pos\_msg.pos -> OuterLoop.pos\_ref

pos\_msg.pos2 -> OuterLoop.pos

OuterLoop.vel\_err -> ang\_ref.vel

OuterLoop.ang\_ref -> ang\_ref.ang

In *InnerLoop* add references to the Simulink Subsystem named ‘InnerLoop’. Also add references to ang\_ref and thrust\_commands: ang\_ref goes to InnerLoop and InnerLoop goes to thrust\_commands:

Ang\_ref.ang -> InnerLoop.ang\_ref

Ang\_ref.vel -> InnerLoop.angle

InnerLoop.ang\_err -> thrust\_commands.ang1

InnerLoop.Torque -> thrust\_commands.ang2

InnerLoop.ang\_vel -> thrust\_commands.ang3

*DataHandling* is a C-code based component. Add a CCode block into the component. Also add references to pos\_ref, sensor\_data, and pos\_msg. Inside of the CCode block add two input ports (pos\_ref and sensor\_data) and two output ports (pos and pos2). Wire the pos\_ref message into the appropriate port on the CCode block. Sensor\_data is also an input. The CCode block is wired out to pos\_msg.

**Step 3** – Define the *Physical Platform*

Add a DesignFolder under the Root Folder and call it “Platform”. Inside the Platform folder add a HardwareUnit and call it “RS\_GS”. This stands for RoboStix/GumStix. It is a simple platform with two processors connected via an *I2C bus*.

Into RS\_GS add two Nodes, called RS and GS, and add a TTBus, called TT\_I2C. The RS node needs two IChan (named ADC and Serial), one BChan, and one OChan (named Serial) and an OS block. The GS node needs one BChan and an OS block. The OS block in the RS node needs its Resolution value set to 1ms. The OS block in the GS node is also set to 1ms.

Connect the RS and GS nodes to the TTBus via the BChan ports. Add three IODevices to the Platform, named SerialIn, ADC and SerialOut. Connect SerialIn to the Serial input port on the RS node. Connect ADC to the ADC input port on the RS node. And connect the Serial output port on the RS node to the SerialOut IODevice.

Add a Plant block to Platform. Connect the ADC, SerialIn, and SerialOut IODevices to the Plant.

**Step 4** – Create a *Deployment* of Components onto the Platform

Add a new DesignFolder and call it “Deployment“. Inside add a System and call is “RS-GS Deployment”. Into this System add references to the InnerLoop, the OuterLoop, and the DataHandling component types.

There are three *aspects* used in creating a deployment. Make sure that you are in the LogicalArchitectureView. Connect together the three components in the following manner:

DataHandling.pos\_msg -> OuterLoop.pos\_msg

OuterLoop.ang\_ref -> InnerLoop.ang\_ref

Now switch to the DeploymentView aspect. Add references to the RS and GS nodes. We want to deploy DataHandling and InnerLoop onto the RS node and OuterLoop onto the GS node. Connect from the Components to the Nodes to achieve this. There should be dashed lines pointing to the Nodes nows.

We also need to associate hardware ports on Nodes with message ports on Components. Connect Component ports and Node ports as follows:

DataHandling.pos\_msg -> RS.ADC

DataHandling.sensor\_data -> RS.Serial (Kind IChan)

InnerLoop.ang\_ref -> RS.BChan

InnerLoop.thrust\_commands -> RS.Serial (Kind OChan)

OuterLoop.pos\_msg -> GS.BChan

OuterLoop.ang\_ref -> GS.BChan

Finally, switch to the ExecutionView aspect. We need to add timing information to all objects that need to be scheduled explicitly. Add five TTExecInfo blocks. To each component, connect one TTExecInfo. The messages that get passed via the TTBus also need to get scheduled. Connect one TTExecInfo to the pos\_msg ports on both DataHandling and OuterLoop. Likewise, connect the last TTExecInfo to the ang\_ref ports on both OuterLoop and InnerLoop. All TTExecInfo blocks need to have ExecPeriod set to 20ms and WCDuration set to 1ms.

Congratulations! Your ESMoL model is now fully designed. In the following steps we will generate functional code, perform some analysis, and finally generate all of the glue code and virtual machine you will need to actually execute your model.

**Step 5** – Generate the functional code

From the command line run the SL\_CodeGen and SF\_CodeGen tools against your .mga file. Like this[[7]](#footnote-7):

**`cd %HCDDES\_PATH%\examples\example1\tests`**

**`mkdir Generated\c\_codegen`**

**`mkdir Generated\schedtool`**

**`mkdir Generated\schedresults`**

**`mkdir Generated\sl\_codegen`**

**`mkdir Generated\sf\_codegen`**

**`mkdir Generated\stage1`**

**`mkdir Generated\stage2-sched`**

**`mkdir Generated\stage2-frodo`**

**`mkdir Generated\stage2-truetime`**

**`mkdir Generated\target-frodo`**

**`mkdir Generated\target-truetime`**

**`SL\_CodeGen ..\model\QuadIntegrator.mga -p Generated\sl\_codegen`[[8]](#footnote-8)**

**`SF\_CodeGen ..\model\QuadIntegrator.mga -t Generated\sf\_codegen`**

**`C\_CodeGen.exe -f ..\model\QuadIntegrator.mga -p Generated\c\_codegen`**

All .c files are generated inside the tests\Generated\ directory structure.

**Step 6** – Transform the model into its abstract description

In order to carry out various types of complex analysis on your model we first need to transform it from the ESMoL modeling language to the ESMoL\_Abstract language. The Stage1 tool is used to do this. It is run like this[[9]](#footnote-9):

**`Stage1 –f ..\model\QuadIntegrator.mga -o Generated\stage1\QuadIntegrator.xml`**

A new file, QuadIntegrator.xml, is created. Now your model is ready to have various types of analysis.

**Step 7** – Generate the Scheduler’s input file

**`Stage2 -f Generated\stage1\QuadIntegrator.xml -o Generated\stage2-sched\sched.scs -t Sched -r 1ms`**

**Step 8** – Derive a valid time-triggered schedule

**`SchedTool -f Generated\stage2-sched\sched.scs -o Generated\schedtool\results.rslt`**

**Step 9** – Import resultant schedule back into the original models

**`SchedResults -r Generated\schedtool\results.rslt -e ..\model\QuadIntegrator.mga -a Generated\stage1\QuadIntegrator.xml`**

When populating the ESMoL schedule, the tool complains about some sensors having ‘no valid channels.’ This is the expected behavior; you may safely ignore these messages.

**Step 10** – Generate *Glue Code*, *Virtual Machine*, and *TrueTime Simulation* artifacts

**`Stage2 -t FRODO -f Generated\stage1\QuadIntegrator.xml -o Generated\stage2-frodo`**

**`Stage2 -t TrueTime -f Generated\stage1\QuadIntegrator.xml -p Generated\stage2-truetime -o QuadIntegrator`**

**Step 11** – Copy generated code to working directories for various target platforms

**copy Generated\sl\_codegen\\*.c Generated\target-frodo  
copy Generated\sl\_codegen\\*.h Generated\target-frodo  
copy Generated\c\_codegen\\*.c Generated\target-frodo  
copy Generated\c\_codegen\\*.h Generated\target-frodo  
copy Generated\stage2-frodo\GS.c Generated\target-frodo  
copy Generated\stage2-frodo\RS.c Generated\target-frodo  
;  
copy Generated\sl\_codegen\\*.c Generated\target-truetime  
copy Generated\sl\_codegen\\*.h Generated\target-truetime  
copy Generated\c\_codegen\\*.c Generated\target-truetime  
copy Generated\c\_codegen\\*.h Generated\target-truetime  
copy Generated\stage2-truetime\GS\_init.cpp Generated\target-truetime  
copy Generated\stage2-truetime\RS\_init.cpp Generated\target-truetime  
copy Generated\stage2-truetime\QuadIntegrator\_gen.m Generated\target-truetime**

**Step 12** – Execute the generated TrueTime co-simulation

Copy over some pre-staged utility scripts/models.

**`pushd Generated\target-truetime`**

**`copy ..\..\..\targets\truetime\QuadIntegrator\_post-processed.mdl .\QuadIntegrator.mdl`**

**`copy ..\..\..\targets\truetime\init\_ttenv.m`**

Next, open MATLAB and add the TrueTime toolkit to your MATLAB PATH.

**`MATLAB.exe`**

**`init\_ttenv` # this executes a provided .m script**

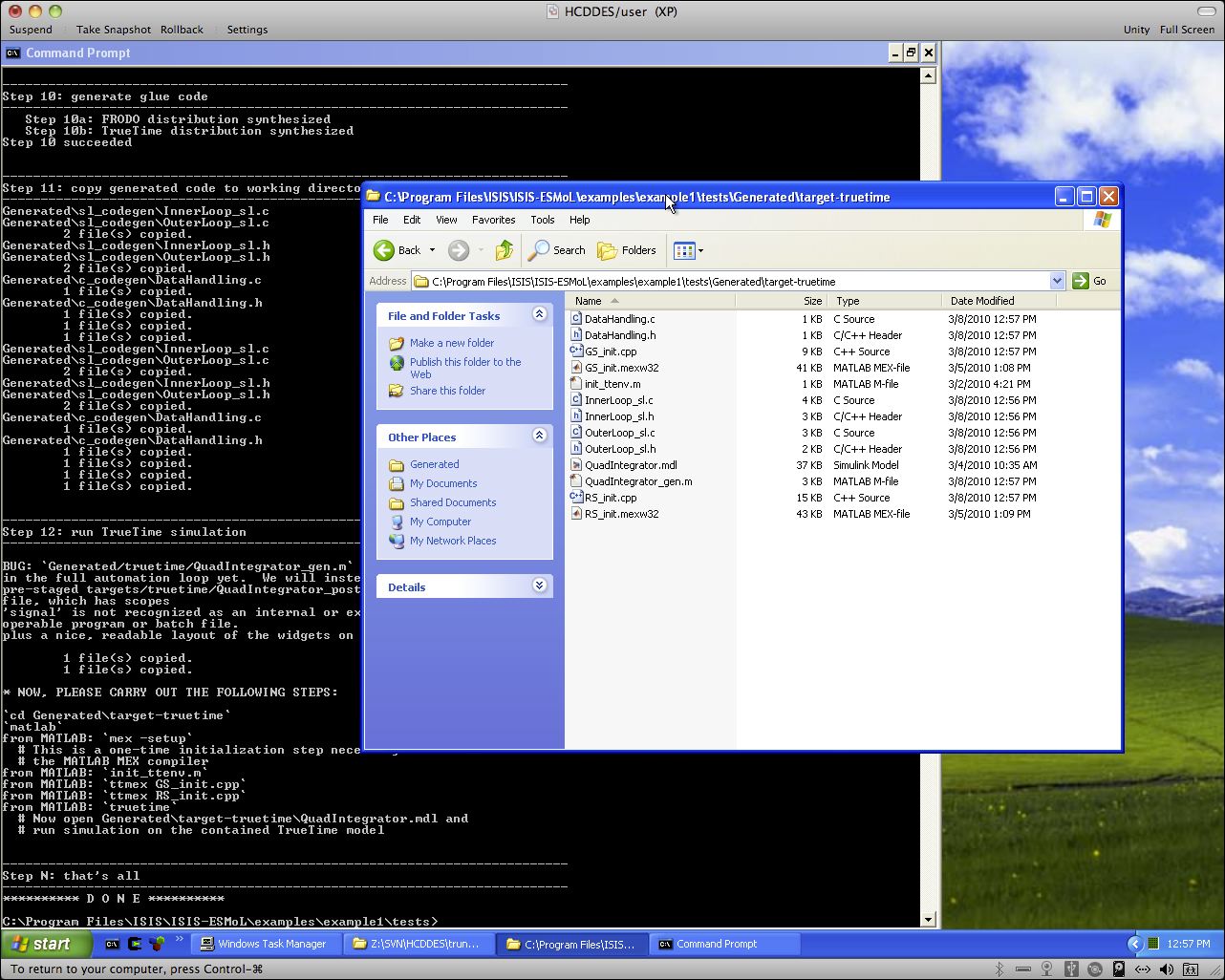
Compile the simulation model (i.e. from within MATLAB).

**`ttmex GS\_init.cpp`**

**`ttmex RS\_init.cpp`**

Finally, from the resultant TrueTime window, open the model (viz. QaadIntegrator\_post-processed.mdl)[[10]](#footnote-10). Now open one or more of the oscilliscope blocks and execute the Simulink/TrueTime simulation[[11]](#footnote-11).

\* Examine the contents of tests/Generated/target-truetime/. The files generated here by the above process should be quite similar to the example files shipped in targets/truetime/.

**Figure**: a directory listing of the synthesized TrueTime co-simulation

Also, please consult the additional reference material provided at %HCDDES\_PATH%\doc\papers\RSP10.pdf for more thorough coverage of the role that these TrueTime co-simulations play in the overall development, testing and debugging workflow for embedded systems design.

**To go to hardware-in-the-loop testing…**

**Step 13** – Deploy the generated system to hardware

Given that the TrueTime co-simulation produces satisfactory results, we are finally ready to deploy the actual control system on microprocessors. In the previous step, we were working out of the tests/Generated/target-truetime/ directory. Similarly, there is a tests/Generated/target-frodo/ directory containing the synthesized C code targeted at the Gumstix/Robostix execution platform. In this section, we will compile that code and deploy it onto a running system, where the Plant dynamics are simulated by a Mathworks xPC system[[12]](#footnote-12).

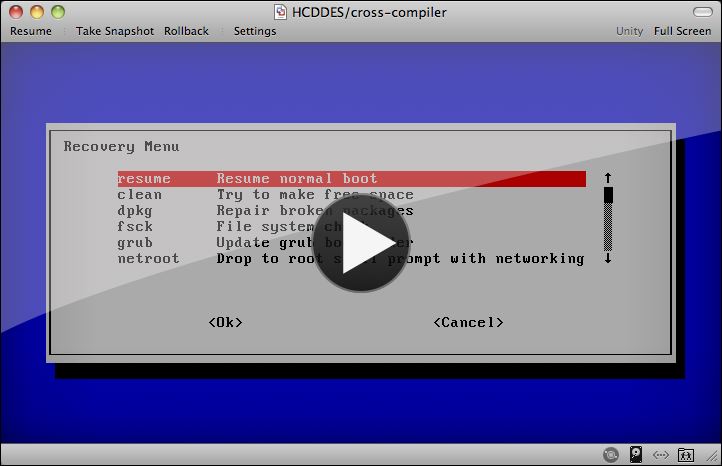
**Step 13a** – copy over the synthesized system into the cross-compiler staging area

**`pushd %HCDDES\_PATH%\examples\example1\targets\hardware\`  
`deploy.cmd`**

**Step 13b** – compile the sources for both the Gumstix and the Robostix hardware.

The gumstix/robostix cross-compiler is not supported under the Microsoft Windows environment, but rather runs under Linux only. OpenEmbedded.org provides a free cross-compiler ready to be downloaded. Relevant instructions are provided at <http://wiki.openembedded.net/index.php/Main_Page>.

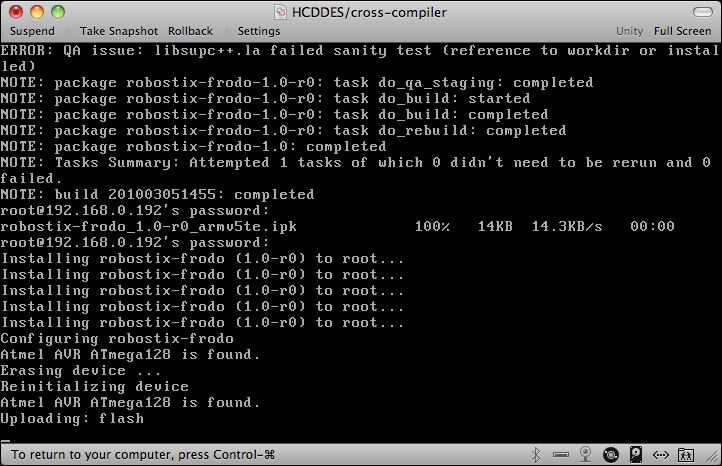
Alternately, as a convenience we provide a pre-configured cross-compiler environment in the form of a VMware virtual machine, which is located at <http://129.59.129.100/ubuntu904server.tar.gz>. This URL is ephemeral, meaning we do not expect to maintain it for long periods of time. If at the time of reading, this link is broken, feel free to contact the development team for up to date instructions.

**Figure**: the Ubuntu based cross-compiler VM

Log into the Linux machine[[13]](#footnote-13) that houses your cross-compilation environment and set the current working directory to the equivalent of the path labeled %HCDDES\_PATH%\targets\hardware\build in your Windows platform above. We recommend you use VMware’s “Sharing” feature to enable this directory to automatically appear underneath your Linux VM. Alternately, you could arrange some other form of shared filesystem, or your preferred file transfer mechanism.

Now, let’s compile and package up the runtime executables:

`**cd /mnt/hgfs/Programs/ISIS/ESMoL/examples/example1/targets/hardware/build/`  
`bin/gf`  
`bin/rf` [[14]](#footnote-14)**

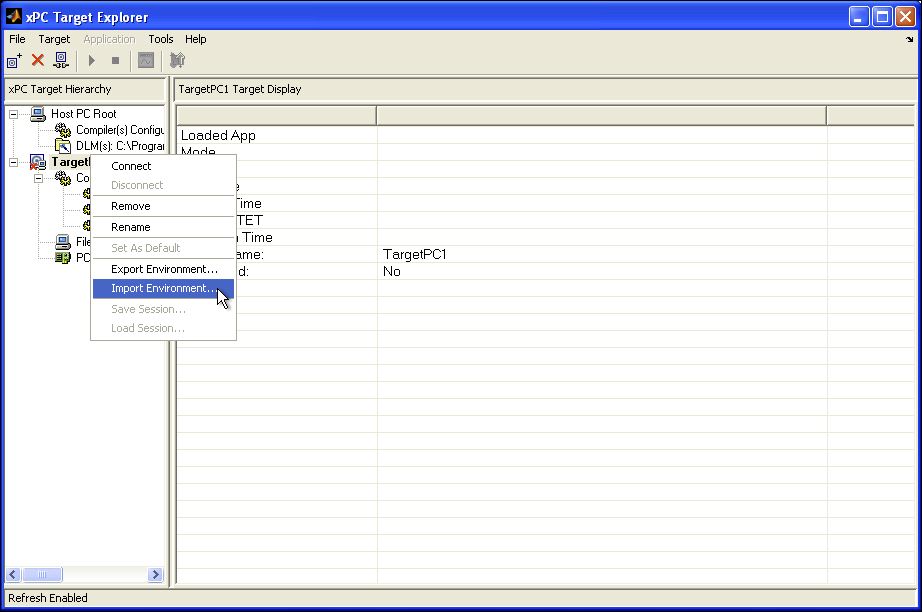
**Figure**: the cross-compilation procedures

**Step 13c** – Initiate real-time simulation of the Plant dynamics

At this stage of the process, we will still be simulating the behavior of the system under control, however the actual controls will be deployed on real computing and networking hardware[[15]](#footnote-15). For this, we will rely on an xPC target to simulate the physical system’s dynamics. Included in %HCDDES\_PATH%\targets\hardware\xPC\ is a version of the original Simulink model (viz. %HCDDES\_PATH%\model\simulink\quad\_integrator\_subsys.mdl) that has been supplemented with the specifications for a particular configuration of an xPC platform (e.g. particular digital I/O, ADC and DAC boards). This section describes the steps necessary to initialize the plant simulation.

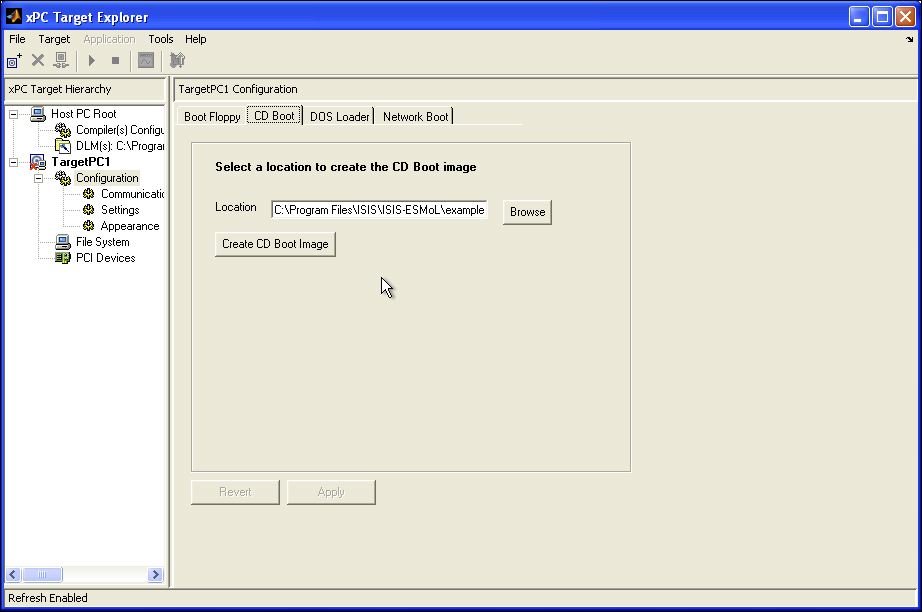
**`cd %HCDDES\_PATH%\examples\example1\targets\hardware\xPC`   
`MATLAB.exe`  
`xpcexplr`  
`load TargetPC1\_Env`**

The last command above loaded your MATLAB workspace with some pre-configured settings for the xPC target. Next, we will load the xPC configuration with theses values. While many of the values will be correct, there may be some xPC configuration settings that you will be required to set manually. For example, if your Microsoft Visual Studio compiler is located in a non-standard location, these settings will result in compile time errors when you attempt to process the xPC model. To load the values from your MATLAB workspace into your xPC machine configuration, right-click on the TargetPC1 node in the tree browser widget of your xPC Target Explorer window.



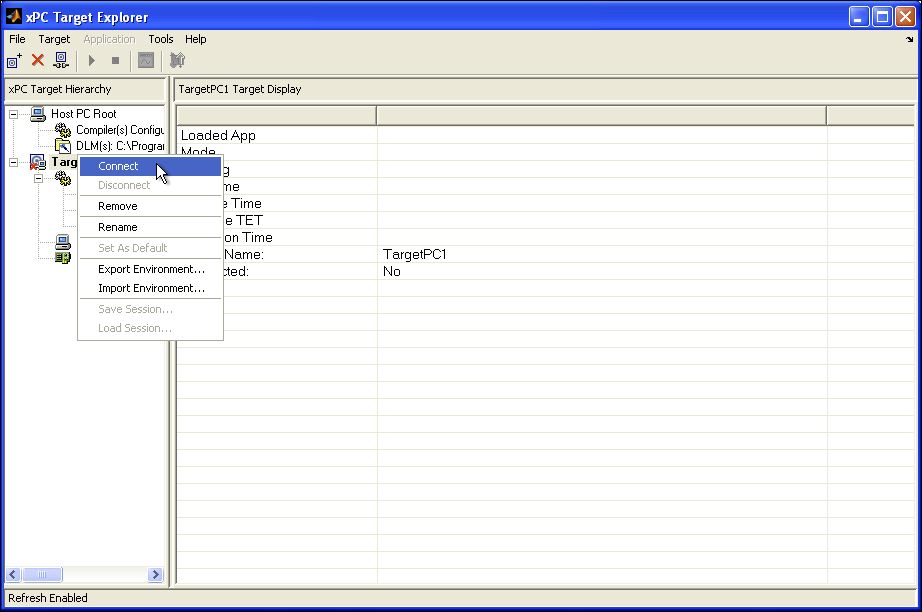
**Figure**: import xPC machine settings

At this point, you may wish to carefully examine each of the settings for the TargetPC1 configuration. When satisfied, you may next create the boot CD image for booting up your xPC simulation. This operation is available form the following screen. Alternately, you may choose to use the %HCDDES\_PATH%\examples\example1\targets\hardware\xPC\cdboot.iso that is pre-packaged in our binary distribution. This ISO, when pressed to a CD will be used to boot the xPC target.

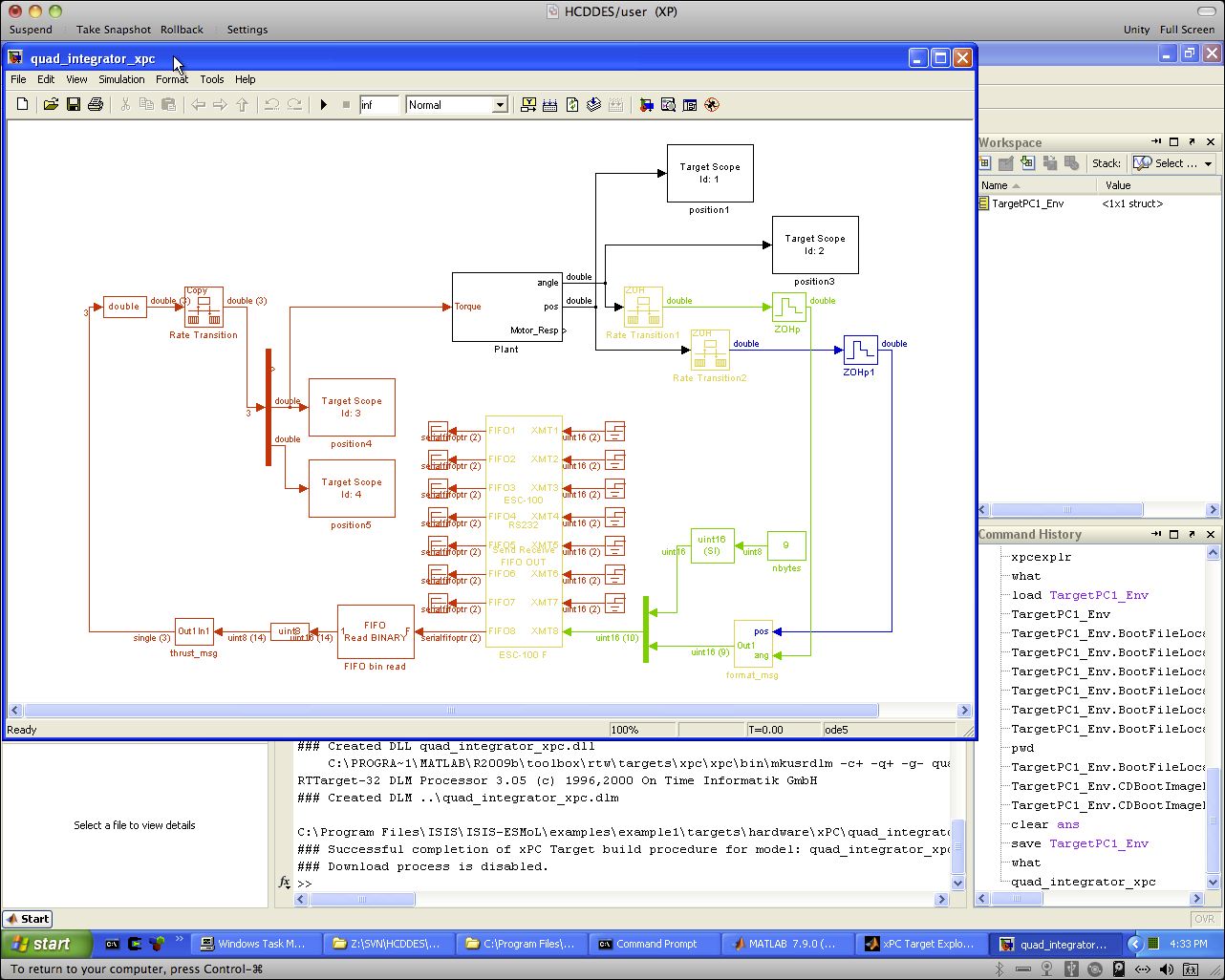


**Figure**: creating the xPC boot image

After using this image to boot the xPC target, one should now be able to *Connect* to the xPC device from your xPC Target Explorer window.

**Figure**: connecting to the xPC target

After connecting to the xPC, open the quad\_integrator\_xpc model, compile it (i.e. <CTRL-B.>) and download the simulation to the xPC target.



**Figure**: the compiled xPC model

**Figure**: xPC after initial bootup sequence



**Figure**: xPC with downloaded model

Finally, you may now begin the simulation.



**Figure**: xPC executing the simulation, but before gumstix/robostix are started

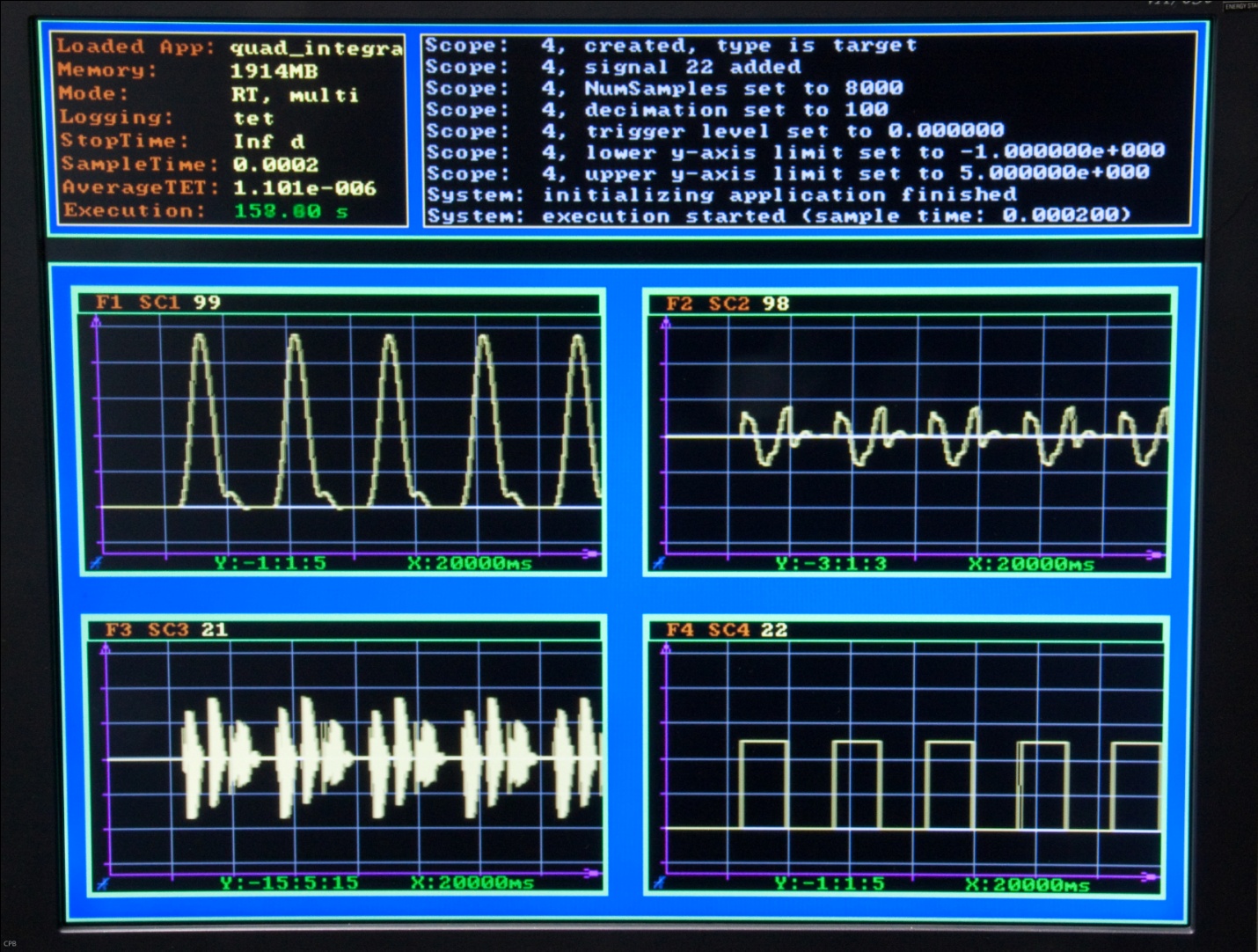
**Step 13d** – start the gumstix/robostix control software

Now we are finally ready for the last step in this process, which is to execute the software that we deployed onto the gumstix/robostix microcontrollers in Step 13b.

First log into the gumstix device, then execute the script that bootstraps the control code.

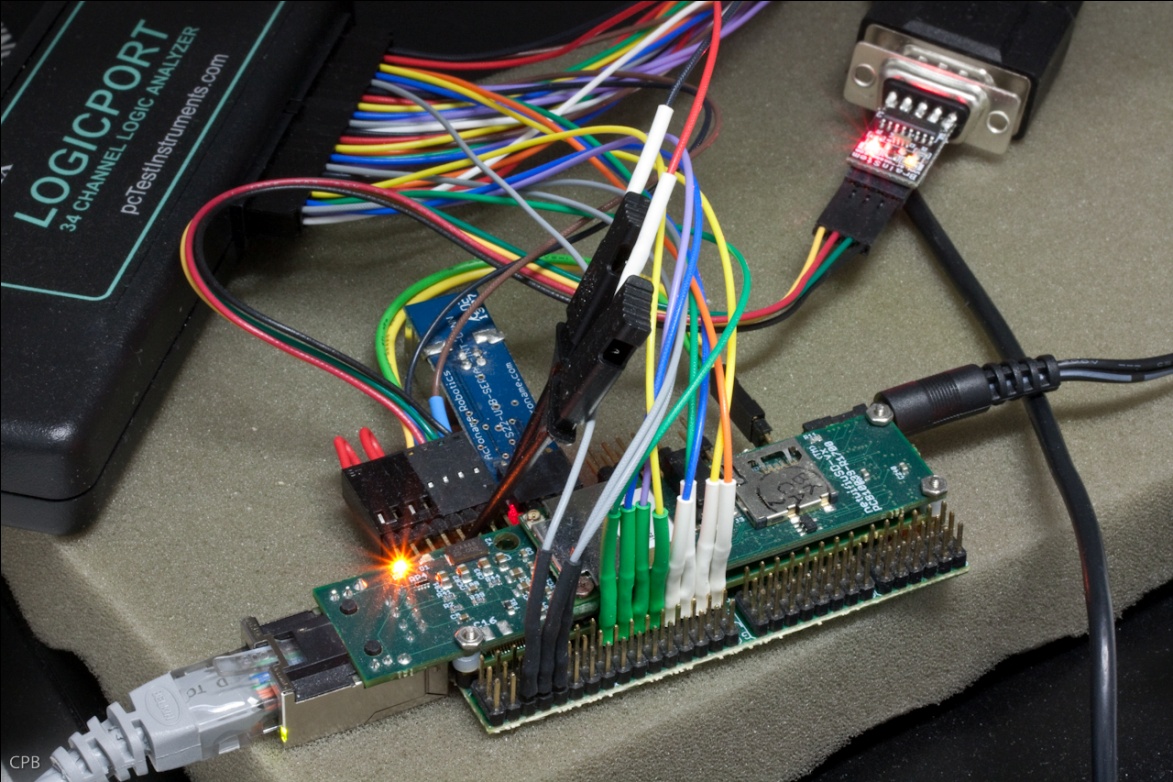
**`ssh** [**root@192.160.0.192`**](mailto:root@192.160.0.192%60) **`./run\_controller`**

At this point, the four graphs on the xPC target should start updating with valid signals. The plot in the upper-left quadrant is the plant’s position signal, which is tracking the command signal shown in the lower-right quadrant.

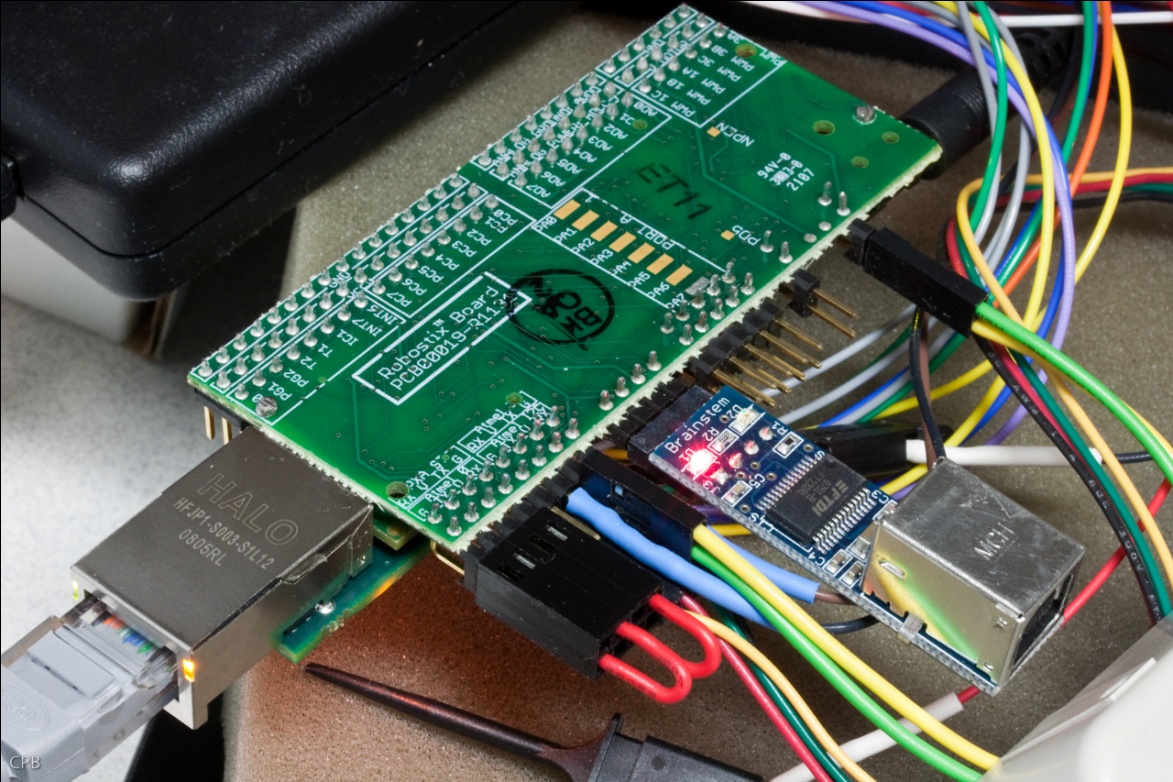


**Figure**: xPC executing the simulation,after gumstix/robostix are started

We close this section with a couple of snapshots of the gumstix/robostix hardware and ﬂassociated wiring.



**Figure**: gumstix side of the assembly



**Figure**: robostix side of the assembly

## Example 2: The Quad Rotor Model

The development flow for the QuadRotor is the same as for the Quad Integrator, so we provide fewer details. Run the steps in this order:

**Step 1** – Import the Simulink model into ESMoL

The model can be found at examples\example2\model\simulink\quadrotor\_demo.mdl.

**Step 2** – Define the *Components* and *Message Types* in ESMoL

For comparison, import the model examples\example2\model\quadrotor\_demo.xme into GME using the “Import XML” entry under the File menu. Take a look at the SysTypes model.

**Step 3** – Define the *Physical Platform*

The platform model contains the details.

**Step 4** – Create a *Deployment* of Components onto the Platform

**Step 5** – Generate the functional code

As per the steps above (Example 1), but using the appropriate paths.

**Step 6** – Transform the model into its abstract description

**Step 7** – Generate the Scheduler’s input file

**Step 8** – Derive a valid time-triggered schedule

**Step 9** – Import resultant schedule back into the original models

**Step 10** – Generate *Glue Code*, *Virtual Machine*, and *TrueTime Simulation* artifacts

**Step 11** – Copy generated code to working directories for various target platforms

**Step 12** – Execute the generated TrueTime co-simulation

**To go to hardware-in-the-loop testing…**

**Step 13** – Deploy the generated system to hardware (again, as above).

# Appendix II: Developing ESMoL

This appendix covers details for setting up a development environment to contribute to the ESMoL project, starting with a release of the source code. The automated installer does not provide source code, so source must be obtained separately. Please contact the ESMoL development team for source packages.

This document describes of the prerequisites, installation procedures, and configuration steps necessary to create an operational *ESMoL* (Embedded Systems Modeling Language) development environment. The overall environment is not overly complex, but it is important to carefully follow the steps laid out in this document.  **If you are installing a binary release of the software by way of a *Windows Installer*, these details have been automated for you, except where noted otherwise. Nevertheless, it is still recommended that you review the following procedures in order to have a working familiarity with some of the underlying details.**

# Prerequisite Software

There are a number of external software tools and packages required by the ESMoL environment. These third-party dependencies are open source and most are relatively small. They are contained within the project’s source code repository in the ‘3rdparty’ directory directly under the repository root. We will go into more detail about these in a moment. Several applications and toolkits must be installed by the user before the HCDDES environment can be configured. These applications are:

* Windows XP Service Pack 3, or Windows 7
* Visual Studio 9.0 (a.k.a. 2008) – Standard Edition *(*not *tested with Express Edition) (note 2005 and 2010 are known to not work presently 10/2010)*
* MATLAB with Simulink *(tested with edition R2010a and R2009b)*
* GME version 10.2.9 **or GME version 10.8.18**
* *The following are required for source code, but not binary drops* 
  + UDM Toolkit, Release 3.2.6 (or higher)
  + GREAT Toolkit, Release 1.7.4
  + Java Development Kit *(tested with JDK SE 6)*
  + Python 2.6.5 w/ Cheetah
  + Boost python version 1.42.0

Please read and follow the installation and configuration instructions that come with each of these packages.[[16]](#footnote-16)

## Source Code Installations

If you prefer to install a *source code* release of ESMoL, please contact the development team for Subversion (SVN) repository access details or for details about obtaining a packaged source code drop (i.e. zip file). Once you have obtained access to the HCDDES sources, the next step in the installation and configuration process is to setup the HCDDES root directory. The root directory may be placed in a location of your choosing. A typical location is either at the root of a drive, for example “C:\HCDDES”, or in your documents folder, for example “C:\Documents and Settings\user1\My Documents\HCDDES”.

If you are deploying from a compress ZIP file place all unzipped items into this root directory. If you are deploying from the SVN repository just direct SVN to place all documents into the HCDDES folder. In either case it is important to structure the HCDDES directories properly. The overall directory should be the same as this:

* HCDDES
  + 3rdparty
  + trunk
    - archive
    - bin
    - doc
    - examples
    - include
    - lib
    - src
    - tests

There are a number of other directories present in the hierarchy, but the ones listed above are the most important. Please make sure that all three levels of the hierarchy are present as represented.

## 3rd Party Software Libraries

Contained in the 3rd party directory of the distribution are a number of necessary libraries upon which HCDDES builds. Each of these libraries needs to be unzipped. Below is the list of libraries, what it is used for and the version corresponding to this release:

|  |  |  |
| --- | --- | --- |
| **Library** | **Version** | **Use** |
| Antlr | 2.7.7 and 3.1.3 | Used to create parsers for many input files and to parse portions of MATLAB/Simulink models |
| Boost | 1.42.0 | Boost is used widely throughout HCDDES. A number of the headers are used, but several of its libraries are also compiled and used.  Compiled Libraries:   * filesystem * graph * program options * regex * system * thread * python |
| CTemplate | 0.96 | Used to generate code from standardized templates |
| Python | 2.6.5 | Also used primarily for many code generation templates |
| Gecode | 3.3.1 | Used in schedule analysis and generation |
| Truetime | 2.0 beta 4 | Used to simulate platform timing effects |

Unzip each of these archives directly into the 3rdparty directory. Once all of these libraries have been unzipped--or *installed* in the case of the Gecode .msi file (and Python) --you are ready to continue setting up the overall environment[[17]](#footnote-17). Python has to be installed in the following manner:

1. Install python-2.6.5.msi normally, by running the installer. It should install in C:\Python26.
2. Once done, unzip the 3rdparty\Python26.zip (unzip –u Python26.zip –d c:\ ). If unzip asks whether you want to replace files, select “Replace All”. This will add the Cheetah template engine to your Python installation.

## Environment Variable Settings

A number of environment variables need to be created and initialized. These variables help the toolchain understand where all of its necessary pieces and parts are located[[18]](#footnote-18).

The method to set an environment variable does not vary. On Windows XP go to the Control Panel and open the System utility if you are in “Classic” view. If you are in “Category” view then select the Performance and Maintenance choice, next select the System utility near the bottom of the window.

In the System utility select the “Advanced” tab and then click Environment Variables. We want to create or set a number of System variables and so will only be dealing with the lower portion of this window. Clicking on new (the lower new button, not the upper one) will let you create a new system variable, while edit will allow modification of the selected, previously existing variable.

Listed in the table below, are all of the environment variables needed for the HCDDES environment. Many of them rely upon the location of the root HCDDES directory. For example, %ANTLR\_PATH% need to be set to %HCDDES\_PATH%\3rdparty\antlr\2.7.7. If you see %HCDDES\_PATH% substitute in the actual path of the HCDDES root directory – so for our example %ANTLR% might be set to C:\HCDDES\3rdparty\antlr\2.7.7.

|  |  |
| --- | --- |
| **System Variable** | **Value** |
| ANTLR\_PATH | %HCDDES Root%\3rdparty\antlr\2.7.7 |
| ANTLR3\_PATH | %HCDDES Root%\3rdparty\antlr\3.1.3 |
| BOOST\_PATH | %HCDDES Root%\3rdparty\boost-1\_42\_0 |
| CTEMPLATE\_PATH | %HCDDES Root%\3rdparty\ctemplate-0.96 |
| GECODE\_PATH | C:\Program Files\Gecode |
| GME\_PATH | C:\Program Files\GME |
| GREAT\_PATH | C:\Program Files\ISIS\GReAT |
| HCDDES\_PATH | %HCDDES Root% |
| MATLAB\_PATH | C:\Program Files\MATLAB\R2010a |
| PYTHON\_HOME | C:\Python26 |
| TTKERNEL | %HCDDES Root%\3rdparty\truetime-2.0-beta4\kernel |
| UDM\_PATH | C:\Program Files\ISIS\UDM |
| XERCES\_PATH | C:\Program Files\ISIS\UDM\3rdparty\xerces-c\_2\_8\_0 |
| VCP\_PATH | %HCDDES Root%\trunk |

Obviously, the specific values required for these variables will depend upon where you have actually installed the dependent frameworks and applications. To aid in debugging, a make-checkenv.cmd script is provided in the trunk directory (i.e. for source distributions), which will identify any mis-configured environment variables. Especially take note of the %MATLAB\_PATH% variable, as its path is also dependent upon the version of MATLAB that you have installed. Additionally, TrueTime requires that the MATLAB startup script be modified to add the TrueTime library’s directories into the default MATLAB path. Follow the TrueTime installation directions for how to properly configure your installation (also, more details are reviewed in a later section).

## Application Compilation

Before you may begin working with models, several applications must be compiled[[19]](#footnote-19). This is also the ultimate test to ensure that your environment has been properly configured. For each of the applications listed below, follow the instructions on how to compile the project. All compiled applications should be automatically placed into the %HCDDES\_PATH%\trunk\bin directory. A “Release” and a “Debug” version of each application should also be available. The Debug version’s name will end in a capital D. All projects are designed and tested to work with Visual Studio 2008.

* **VCP (MDL2MGA, SF\_CodeGen, SL\_CodeGen, C\_CodeGen)**

VCP is a collection of many subprojects and applications that all are built together. The VCP solution is located at %HCDDES\_PATH%\trunk\VCP.sln. Open this solution, select the type of build (Release or Debug) and build the solution.

There usually are several warnings, but no errors should occur during the lengthy compilation. If any errors do happen, recheck your installation and configuration and try again.

* **Stage1**

Stage1 translates models from the ESMoL paradigm into the ESMoL\_Abstract paradigm. The Stage1 solution is located at %HCDDES\_PATH%\trunk\src\Stage1\Stage1.sln. Open this solution, select the type of build, and build the solution.

* **Stage2**

The Stage2 application has several uses depending on the command-line switches provided at run-time. The Visual Studio solution for Stage2 is located at %HCDDES\_PATH%\trunk\src\Stage2\Stage2.sln. Open this solution, select the type of build, and build the solution.

* **SchedTool**

All schedule analysis and generation is performed by the SchedTool application. The SchedTool solution is located at %HCDDES\_PATH%\trunk\src\SchedTool\SchedTool.sln. Open this solution, select the type of build, and build the solution.

* **SchedResults**

Once a schedule has been generated, its results must be fed back into the ESMoL and ESMoL\_Abstract models using the SchedResults application. The SchedResults solution is located at %HCDDES\_PATH%\trunk\src\SchedResults\SchedResults.sln. Open this solution, select the type of build, and build the solution.

* **SchedViz**

Once the schedule has been loaded back into the ESMoL\_Abstract model it can be visualized using the SchedViz application. The SchedViz solution is located at %HCDDES\_PATH%\trunk\src\SchedViz\SchedViz.sln. Open this solution, select the type of build, and build the solution.

Other applications and utilities are contained within the HCDDES source tree, but these applications are the primary ones used most frequently by end users. If all of these applications build successfully, your ESMoL environment is fully configured and ready for use.

The following section covers usage of these individual components at runtime, including for example their supported command-line options.

1. Installation into a non-standard location should also work but has not been thoroughly tested. [↑](#footnote-ref-1)
2. For binary distributions of the toolchain, a convenience script named init\_ttenv.m is provided to aid in the automation of step #2. [↑](#footnote-ref-2)
3. For the binary release of the HCDDES toolchain, example models are distributed in the %HCDDES\_PATH%\examples\ sub-directory. An included appendix describes working with these individual, sample datasets. [↑](#footnote-ref-3)
4. See Also: *The Specification and Implementation of a Model of Computation*. Master’s thesis, Vanderbilt University, by R. Thibodeaux (May 2008). [↑](#footnote-ref-4)
5. Alternative ports do already exist. For example, Windows and Mac targets are commonly used for development and debugging. [↑](#footnote-ref-5)
6. For a binary distribution, this example is installed under %HCDDES\_PATH%\examples\example1\model\simulink\. Additionally, all toolchain executables will be found under %HCDDES\_PATH%\bin\. [↑](#footnote-ref-6)
7. Warning: cutting and pasting commands from a PDF document into a command prompt can sometimes introduce invisible control characters, which cause mysterious errors. [↑](#footnote-ref-7)
8. At the time of this writing, SL\_CodeGen.exe may fail complaining of ‘Udm: XMLException: Could not open file: … \_SFC.xml, of type: IOException … DomDataNetwork::CloseWithUpdate()’ For our purposes, this error condition may safely be ignored. [↑](#footnote-ref-8)
9. Presently, the ESMoL utilities assume that any ‘-o’ flag refers only to directory structures that already exist. The utilities will not create missing directory structure hierarchy. [↑](#footnote-ref-9)
10. During the code generation steps above, a MATLAB script named tests/Generated/target-truetime/QuadIntegrator\_gen.m was synthesized from the original ESMoL model. Executing this script from within MATLAB will initialize the associated TrueTime .mdl file based upon the contents of the ESMoL model. However, there is still some basic wiring that one needs to define manually after generating the TrueTime mdl. For example, the user must still specify which signals are to be visualized on Simulink scope blocks, as that information is not encoded in the ESMoL environment. As a convenience, we provide examples/example1/targets/truetime/QuadIntegrator\_post-processed.mdl, in which these post-processing steps have already been recorded. [↑](#footnote-ref-10)
11. We recommend a minimum of 2Gb of physical memory when working with this example. [↑](#footnote-ref-11)
12. http://www.mathworks.com/products/xpctarget/ [↑](#footnote-ref-12)
13. For the VM provided by ISIS, the username is ‘user’ and password is ‘chrysaor.info’. [↑](#footnote-ref-13)
14. The IP address of the gumstix device is embedded within the gf and the rf scripts. If your subnet defines an address other than 192.168.0.192 for this embedded device, then you will need to edit these two scripts before running them. [↑](#footnote-ref-14)
15. If unfamiliar with the xPC product by Matworks, please consult <http://www.mathworks.com/products/xpctarget/> for relevant background material. [↑](#footnote-ref-15)
16. Installation into a non-standard location should work but has not been thoroughly tested. [↑](#footnote-ref-16)
17. When installing the HCDDES toolchain by way of a Windows Installer, the minimal subset of Gecode (<http://www.gecode.org/>) is packaged and redistributed along with the ESMoL application. [↑](#footnote-ref-17)
18. When installing by way of the Windows Installer, all necessary environment variables will automatically be configured to refer to the newly installed ESMoL tool suite. [↑](#footnote-ref-18)
19. This section is only relevant for source code distributions. Of course, no compilation steps are necessary for binary releases of the software. Also, please take note that there is a %VCP\_PATH%\src\make-all.cmd script, which automates the entire procedure described in this section. [↑](#footnote-ref-19)