Table of Contents

This is the all-inclusive guide to our setup. (Work in Progress)

HCDDES Tutorial

Step 1 – Import Simulink modeling into ESMoL.

Find quad_integrator_subsys.mdl example Simulink model in the trunk/examples/QuadIntegrator directory. From the command line use the MDL2MGA tool to translate this Simulink model into an ESMoL model. Assuming MDL2MGA.exe is in the trunk/bin directory, execute the following:

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 just two folders beneath the Root Folder.

The first folder is Imported Models.

The second folder is Types.

Step 2 – Create Component 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 usually derived from Simulink Subsystems.

Insert DesignFolder and call it Component Definitions.

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

Add five messages: 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: 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 -> ang_ref.vel
OuterLoop.ang -> ang_ref.ang

In InnerLoop add references to the Simulink Subsystem called InnterLoop. 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.ang
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 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 ACD 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 the System add references to the InnerLoop, OuterLoop, and 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 -> OuterLoop.pos
OuterLoop.ang -> InnerLoop.ang

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 -> RS.ADC
DataHandling.sen -> RS.SerialIn
InnerLoop.ang -> RS.BChan
InnerLoop.thrust_commands -> RS.SerialOut
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. Connect one to each Component. The messages that get passed via the TTBus also need to get scheduled. Connect one TTExecInfo to the pos_msg port on both DataHandling and OuterLoop. Likewise, connect the last TTExecInfo to the ang_ref port 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 – Add Typing Information and Generate the Functional Code

From the command line, run the AddTypes tool. It runs directly against your .mga file. It should be involked like this:

AddTypes QuadIntegrator.mga

AddTypes analyzes your model and adds some typing information needed in order to generate functional code. It does perform some updates to your model, but they should only appear in the Types folder which you really never need to mess with.

Now we can generate the functional code for your model. From the command line run the SL_CodeGen and SF_CodeGen tools against your .mga file. Like this:

SL_CodeGen QuadIntegrator.mga
SF_CodeGen QuadIntegrator.mga

A folder is created with all .c files inside.

Step 6 – Transform the Model into the Abstract Description

In order to various types of 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:

Stage1 –f QuadIntegrator.mga –o QuadIntegrator.xml

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

Step 7 – Generate Scheduler Input File

Stage2 –f QuadIntegrator.xml –o sched.scs –t Sched

Step 8 – Generate Schedule using Input File

SchedTool –f sched.scs –o results.rslt

Step 9 – Import Scheduler Results in Models

SchedResults –r results.rslt –e QuadIntegrator.mga –a QuadIntegrator.xml

Step 10 – Generate Glue Code and Virtual Machine

Stage2 –f QuadIntegrator.xml –t FRODO

Goals:

--How to take our XPPro vmz file and get it up and running, ready to test and connect to an XPC Target machine.
--How to take the file that our toolchain spits out and load it onto the gumstix and have it control the XPC Target machine

Part 1: From VMX to ready XPC Target PC

Simply take our .vmz file and open it up via VMWare's Workstation. All tools should be ready and fully capable of running as is.

See our troubleshooting section for help, should it be needed.

You will need a separate machine to run the model for you; to set it up, open MATLAB. Type in *xpcexplr* in the command prompt. What should open is the XPC Explorer program. In this window, add a target pc, configure it as necessary. Create the bootable diskette from the same window.

Part 2: Cross compiler

Just make sure you have a working cross-compiler. OpenEmbedded has a free cross-compiler ready to be downloaded, all you have to do is follow their instructions on the wiki, located here: http://wiki.openembedded.net/index.php/Main_Page

Part 3: IPK Files generated

If you are using a brand new gumstix device, follow these instructions. If not, skip down to the next bold portion. The full version can be found in HCDDES/trunk/FRODOs

From existing documentation:

Check out the main repository, navigate to the HCDDES/trunk/FRODOs/ folder

1) invoke bin/gumstix-kermit
   reboot the gumstix board & hit a button to enter the uBoot prompt

2) install new root filesystem image:

   'loadb a2000000'
   press CTRL-\ then 'c'
   'cd ~/gumstix/gumstix-oe/tmp/deploy/glibc/images/gumstix-custom-verdex/'
   'send gumstix-basic-image-gumstix-custom-verdex.jffs2'
   wait for the end of the transfer (~20 mins)
   'connect'
   'protect on 1:0-1'
   'erase all'
   'cp.b a2000000 40000 ${filesize}'

3) install kernel:

   'loadb a2000000'
   press CTRL-\ then 'c'
   'send uImage-2.6.21-r1-gumstix-custom-verdex.bin'
   wait for the end of the transfer (~2 mins)
   'connect'
   'katinstall 100000'
   'katload 100000'

'bootm'

4) setup wireless link (essid) in /etc/network/interfaces

```
------------------------------------------------
auto wlan0
iface wlan0 inet dhcp
        wireless_mode managed
>>>         wireless_essid  hcddes <<<
------------------------------------------------
```

5) bring up the wireless interface:

'ifup wlan0'

6) add the following lines to /etc/hosts

```
------------------------------------------------
192.168.0.100   ampere
192.168.0.101   gumstix gs
192.168.0.150   ampere-wifi
192.168.0.151   gumstix-wifi gsw
------------------------------------------------
```

7) install uisp:

'ipkg update'
'ipkg install uisp'

8) install robostix support:

'ipkg install robostix-cmdline'
'ipkg install robostix-module'

9) remove things you don't need (only if you really don't need them!):

'ipkg remove alsa*' -- sound (ignore errors regarding dependencies)
'ipkg remove bluez*' -- bluetooth
'ipkg remove boa' -- web server
'ipkg remove dbus*' -- message bus (used by bluetooth)
'ipkg remove kernel-module-bluetooth kernel-module-bnep kernel-module-hci-uart kernel-module-hci-usb kernel-module-hidp kernel-module-l2cap kernel-module-rfcomm kernel-module-sco' -- more bluetooth (multi-step)
'ipkg remove task-base-gumstix task-base-gumstix-bluetooth task-base-gumstix-sound'
'ipkg remove kernel-module-l2cap' - more bluetooth
'ipkg remove kernel-module-bluetooth kernel-module-gumstix-bluetooth'
'ipkg remove task-base-gumstix-touchscreen' -- touchscreeen
'ipkg remove kernel-module-tsc2003' -- more touchscreen
'ipkg remove tslib-calibrate tslib-tests libts-1.0-0 pointercal' - final ts

10) add ssh public keys to authoried_hosts. We have used the passphrase located in

HCDDES/trunk/doc

11) replace dropbear to openssh on the gumstix (in a minicom session :)):
'ipkg remove dropbear'
'ipkg install openssh'
'/etc/init.d/sshd start'

If your gumstix device already has an operating system set up:
If you have the gumstix devices that have already been kindly set up. All you must do is edit the app.c files found in each of the *-frodo/files folder.

Following these instructions, you should find that, in the HCCDES/trunk/FRODOs/bin folder, there is a pair of scripts. rf and gf.

Power on the gumstix device, use ifconfig to make sure its networking is set up as desired.

Edit these scripts as necessary. Make sure to change the scp targets to whatever the gumstix has been assigned on your router.

Once you execute these scripts, they will use the cross compiler to build all the .c files necessary for the gumstix to control the XPC Target PC.

These scripts will also dump the IPK files onto the gumstix you have set up in the rf and gf scrips.

Part 4: Building the IPK files on the gumstix

If, for some reason, your rf and gf scripts failed to report the successful build of the ipk files, follow the instructions below.

ssh into your device, the default username and password is root:gumstix

From there, you should see the two IPK files in root's home folder.

Use the command ipkg -force-reinstall install robo* && ipkg -force-reinstall install gum*

Part 5: Demo

For the quad_integrator demo, you should have the XPC Target PC loaded with the quad_integrator model. Start the model, then run the gumstix-frodo command on the gumstix device.

Optionally, you may find that there is a script called gumstix-perpetual in the home directory. This script will keep the gumstix program running to work around I2C errors, should they arise.

Troubleshooting: (WIP, updated as necessary)

Cross compiler issues:

During the bitbake build of your cross-compiler (if you are using openembedded), we have found it sometimes necessary to downgrade and upgrade your C and C++ compilers to work around issues.

VMware Problems:

Sometimes, while creating (or rather trying to) a bootable diskette, XPC will complain and fail to write to the CD. Make sure your MATLAB directory has no spaces in it.

Gumstix Problems: