5LIJ0
Assignment 1
PIL and HIL Simulation Tutorial

February 8, 2022

1 Installing and Configuring the Virtual Machine

Please read *Tutorial 1 The virtual machine* before this tutorial to set up your ECS virtual machine.

1.1 Allowing VM access to USB ports on Linux distributions

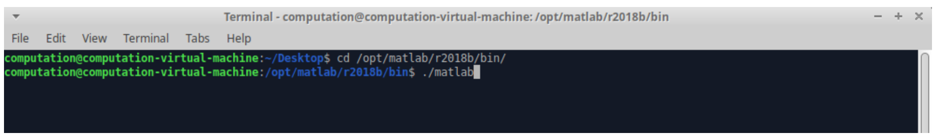
If you are running VirtualBox on a Linux distro, you have to add your username to the `vboxuser` group in order to access USB ports in the VM. You can simply do this by the following command(Replace the [username] by your own username).

```
    sudo usermod -a -G vboxusers [username]
```

2 Running MATLAB

In order to follow through the items in this tutorial, you need to run MATLAB R2018b. you can start MATLAB by opening a terminal and using the commands below.

```
     cd /opt/matlab/r2018b/bin/
    ./matlab.sh
```


Terminal - computation@computation-virtual-machine: /opt/matlab/r2018b/bin
File Edit View Terminal Tabs Help

```
computation@computation-virtual-machine:~/Desktop$ cd /opt/matlab/r2018b/bin/
computation@computation-virtual-machine:/opt/matlab/r2018b/bin$ ./matlab
```

Figure 1: Running MATLAB as `root` user.

3 Connecting to the board via `SSH`

Before continuing the tutorial it is expected that you already set up your board. if not, please follow the provided tutorial *Tutorial 2 The PYNQ board* provided in canvas.

SSH is a command interface to remotely access the board. While most of the necessary communication between the board and your Virtual Machine is carried out by Simulink, the board needs to be initialized to accept MATLAB connection. Therefore you need to establish a remote connection to your board via `SSH`. To do so, open two instances of terminal shells on your Virtual Machine and establish an `SSH` connection to the board on both terminal shells via the command:

```
    ssh student@pato-board.local
    student
```

The second line in above commands list is the password for the student user. For any other operation you can use the username and password as follows:

- Username: student
- Password: student

```
computation@computation-virtual-machine:~$ ssh student@pato-board.local
```

Figure 2: Connecting to the board via `SSH`.

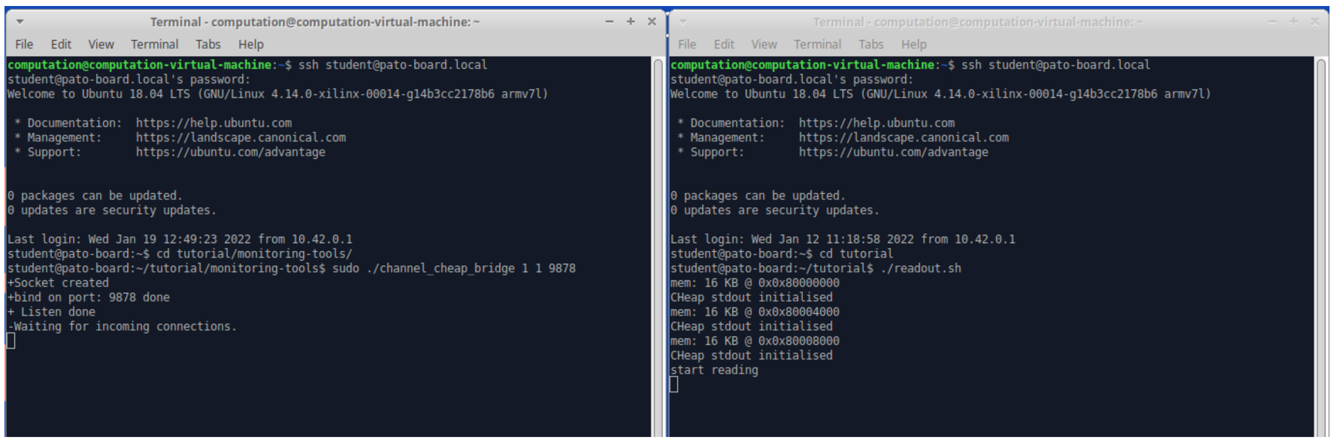
Next, execute the following command on one of the shells in order to prepare the board for connection via Simulink. The command below opens a TCP/IP port between your virtual machine and your PYNQ board.

```
    cd tutorial/monitoring-tools/
    sudo ./channel_cheap_bridge 0 1 9878
```

Next, on your second terminal, execute the following command on the other to initialize your hardware platform on the FPGA part of the board.:

```
    cd tutorial
    ./readout.sh
```

You should end up with the responses shown in Figure 3. If the commands are successfully executed and 'Waiting for incoming connections' statement is present, you can minimize both terminal shells and continue with the next steps on MATLAB and Simulink interfaces.


Terminal - computation@computation-virtual-machine: ~
File Edit View Terminal Tabs Help

```
computation@computation-virtual-machine:~$ ssh student@pato-board.local
student@pato-board.local's password:
Welcome to Ubuntu 18.04 LTS (GNU/Linux 4.14.0-xilinx-00014-g14b3cc2178b6 armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


0 packages can be updated.
0 updates are security updates.

Last login: Wed Jan 19 12:49:23 2022 from 10.42.0.1
student@pato-board:~$ cd tutorial/monitoring-tools/
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 1 1 9878
+Socket created
+bind on port: 9878 done
+ Listen done
-Waiting for incoming connections.
```

```
computation@computation-virtual-machine:~$ ssh student@pato-board.local
student@pato-board.local's password:
Welcome to Ubuntu 18.04 LTS (GNU/Linux 4.14.0-xilinx-00014-g14b3cc2178b6 armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


0 packages can be updated.
0 updates are security updates.

Last login: Wed Jan 12 11:18:58 2022 from 10.42.0.1
student@pato-board:~$ cd tutorial
student@pato-board:~/tutorial$ ./readout.sh
mem: 16 KB @ 0x0x80000000
CHeap stdout initialised
mem: 16 KB @ 0x0x80004000
CHeap stdout initialised
mem: 16 KB @ 0x0x80008000
CHeap stdout initialised
start reading
```

Figure 3: Connecting to the board via `SSH`.

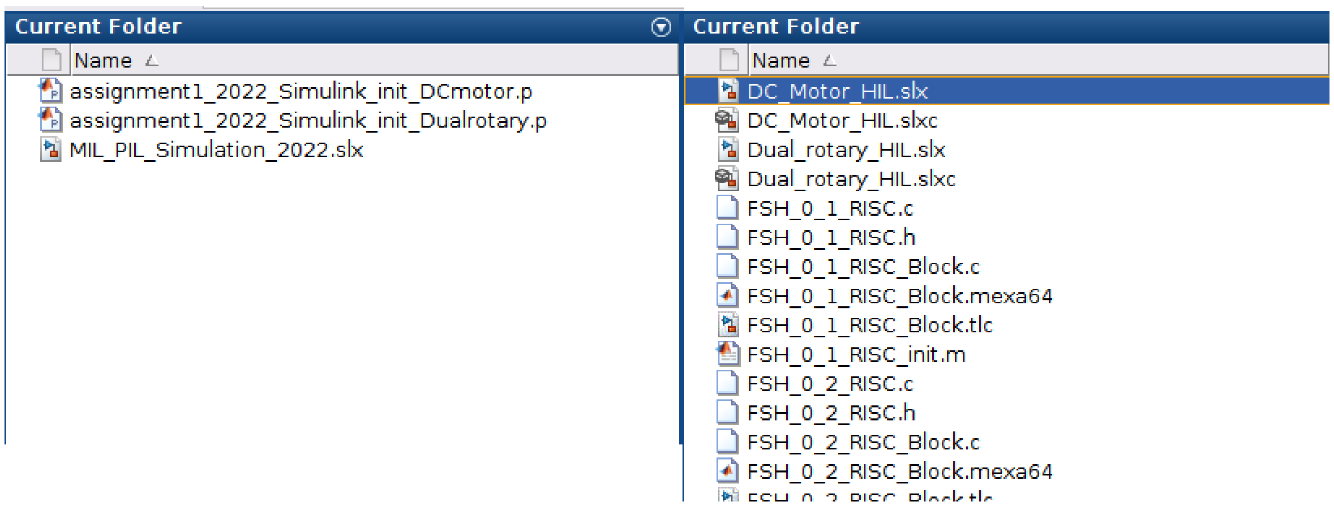
4 Preparing the MATLAB environment

All the necessary files and settings for Simulink connection and code generation are provided to you. The necessary files are already in your virtual machine. The necessary files are already in your virtual machine. The provided files are in the following directories:

- `/home/computation/MATLAB`$_Files/$

  The first directory contains two folders. The first folder includes two two sub folders names as:
  - MIL PIL 2022
  - HIL Simulations 2022

The first folder contains the necessary files for you to perform MIL and PIL simulations. The second folder contains the required files to perform HIL simulation. (More details about MIL,PIL, and HIL simulations can be found in assignment decription). The figure below represents the expected files to be present in each of these folders respectively.



assignment1_2022_Simulink_init_DCmotor.p
assignment1_2022_Simulink_init_Dualrotary.p
MIL_PIL_Simulation_2022.slx

DC_Motor_HIL.slx
DC_Motor_HIL.slxc
Dual_rotary_HIL.slx
Dual_rotary_HIL.slxc
FSH_0_1_RISC.c
FSH_0_1_RISC.h
FSH_0_1_RISC_Block.c
FSH_0_1_RISC_Block.mexa64
FSH_0_1_RISC_Block.tlc
FSH_0_1_RISC_init.m
FSH_0_2_RISC.c
FSH_0_2_RISC.h
FSH_0_2_RISC_Block.c
FSH_0_2_RISC_Block.mexa64

Figure 4: Suggested MATLAB path folder structure.

5 Simulink model and MIL Simulation

Opening the model `MIL_PIL_Simulation_2022.slx` in Simulink, you should see the view depicted in Figure 5.

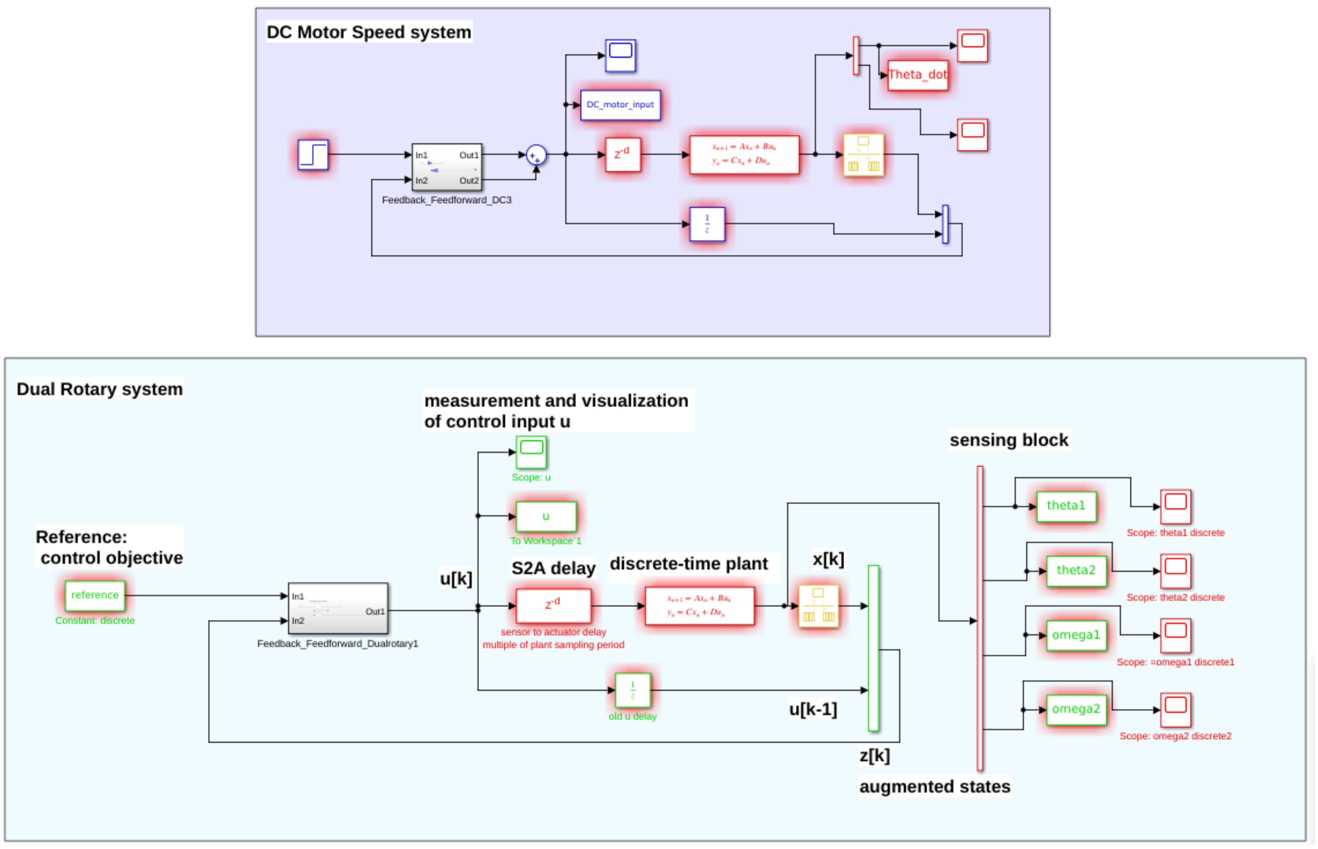


Figure 5: Provided Simulink model for PIL simulation.

In this model, two blocks are of interest to you. Namely, the feedback feedfowrad subsystems of the dual rotary and the DC motor. Since the goal of this assignment is to design controllers with consideration of scheduling and delay of the processors in the loop, you are expected to make your implementation on these blocks.

You can directly execute the MIL simulation by running this model. Before performing the simulation you need to initialize the parameters for the simulation. This is explained in the project description provided to you.

6 Building controller blocks for PIL Simulation

Once your MIL simulations result in proper control performance, you perform PIL simulations to extract the necessary execution times for your controllers. This will later help you in calculating the exact value of sensor to actuator delay. To perform PIL simulations, you need to build PIL blocks for each controller and perform PIL simulation. Building controller blocks for PIL simulation in Simulink is straightforward. Provided that you have successfully completed steps 2 to 4, the PIL simulation blocks can be built by right clicking the block and choosing `C/C++Code` $\longrightarrow$ `Build This Subsystem`. This is also depicted in Figure 6. You can directly use the given Simulink model without modification and follow the steps.

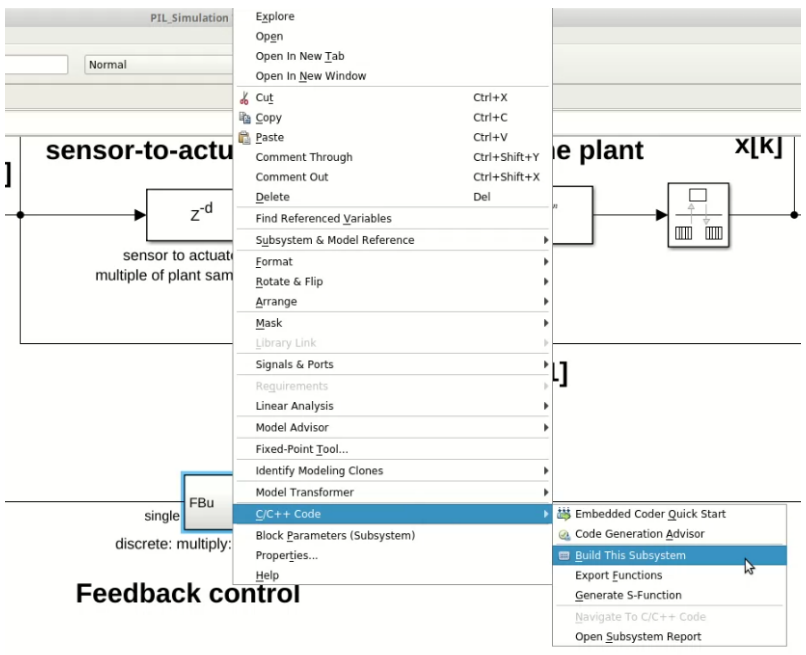
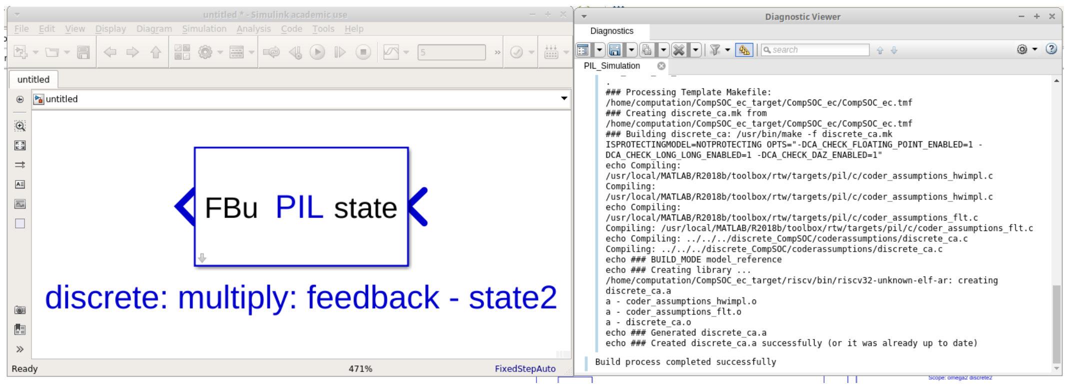


Figure 6: Building controller subsystem for PIL simulation.

Successful build process should result in the generation of a PIL simulation block in a new window and the diagnostics shown in Figure 7.



```
### Processing Template Makefile:
/home/computation/CompSOC_ec_target/CompSOC_ec/CompSOC_ec.tmf
### Creating discrete_ca.mk from
/home/computation/CompSOC_ec_target/CompSOC_ec/CompSOC_ec.tmf
### Building discrete_ca: /usr/bin/make -f discrete_ca.mk
ISPROTECTINGMODEL=NOTPROTECTING OPTS="-DCA_CHECK_FLOATING_POINT_ENABLED=1 -
DCA_CHECK_LONG_LONG_ENABLED=1 -DCA_CHECK_DAZ_ENABLED=1"
echo Compiling:
/usr/local/MATLAB/R2018b/toolbox/rtw/targets/pil/c/coder_assumptions_hwimpl.c
Compiling:
/usr/local/MATLAB/R2018b/toolbox/rtw/targets/pil/c/coder_assumptions_hwimpl.c
echo Compiling:
/usr/local/MATLAB/R2018b/toolbox/rtw/targets/pil/c/coder_assumptions_flt.c
Compiling: /usr/local/MATLAB/R2018b/toolbox/rtw/targets/pil/c/coder_assumptions_flt.c
echo Compiling: ../../../discrete_CompSOC/coderassumptions/discrete_ca.c
Compiling: ../../../discrete_CompSOC/coderassumptions/discrete_ca.c
echo ### BUILD_MODE model_reference
echo ### Creating library ...
/home/computation/CompSOC_ec_target/riscv/bin/riscv32-unknown-elf-ar: creating
discrete_ca.a
a - coder_assumptions_hwimpl.o
a - coder_assumptions_flt.o
a - discrete_ca.o
echo ### Generated discrete_ca.a
echo ### Created discrete_ca.a successfully (or it was already up to date)
Build process completed successfully
```

Figure 7: Result of a successful PIL block build.

Once the PIL simulation block is built, the only action you have to take in order to run the simulation is swapping the subsystem with the generated PIL block by cutting the generated from the new untitled window and pasting it to the model as shown in Figure 8.

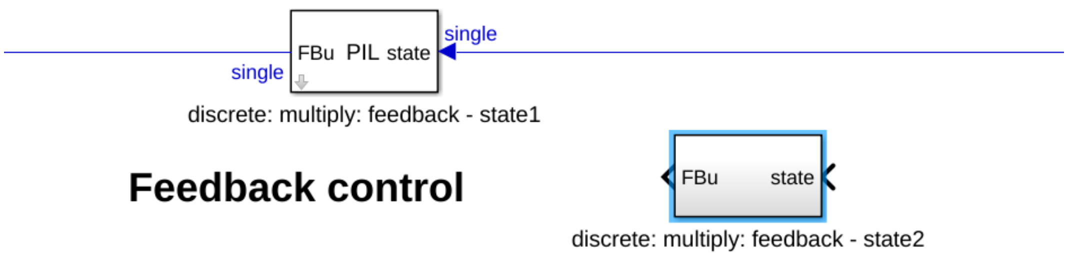
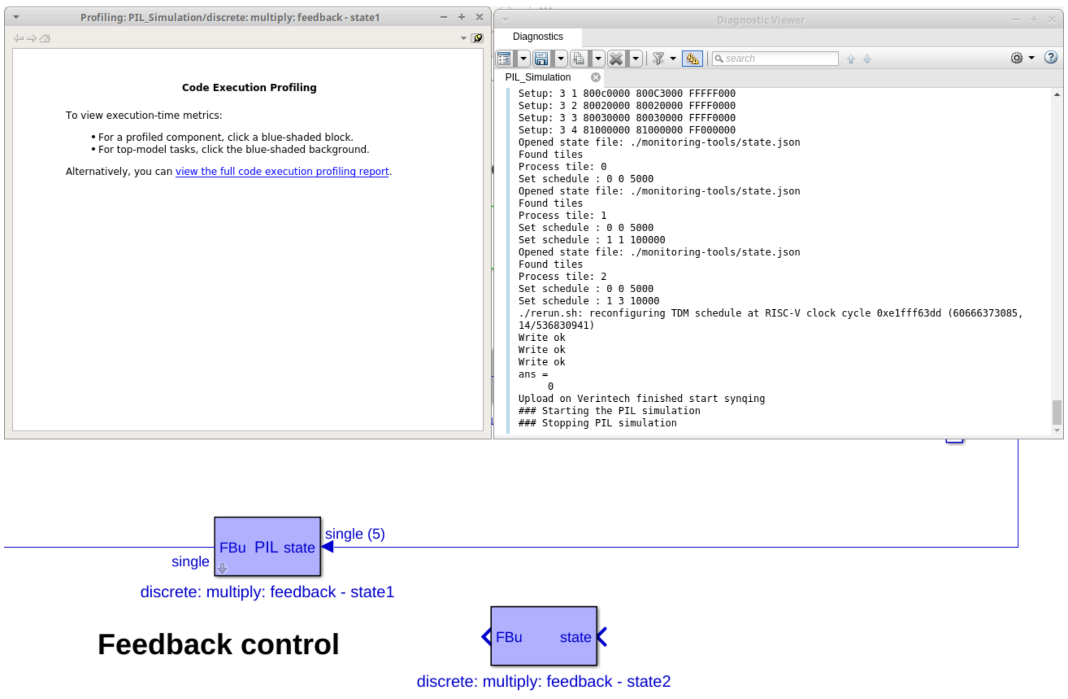

single
FBu PIL state
single
discrete: multiply: feedback - state1
Feedback control
FBu
state
discrete: multiply: feedback - state2

Figure 8: Result of a successful PIL block build.

NOTE: Please build PIL blocks and perform PIL simulation for each system (DC motor and the dual rotary) separately. DO NOT include two PIL blocks when you perform PIL simulations. This will result in an incorrect PIL result.

7 Running the PIL Simulation to determine execution times

Running the simulation with the PIL controller block in the loop should result in Figure 9.


Profiling: PIL_Simulation/discrete: multiply: feedback - state1
Diagnostic Viewer
Diagnostics
PIL_Simulation
single (5)
FBu PIL state
single
discrete: multiply: feedback - state1
Feedback control
FBu
state
discrete: multiply: feedback - state2

Code Execution Profiling

To view execution-time metrics:

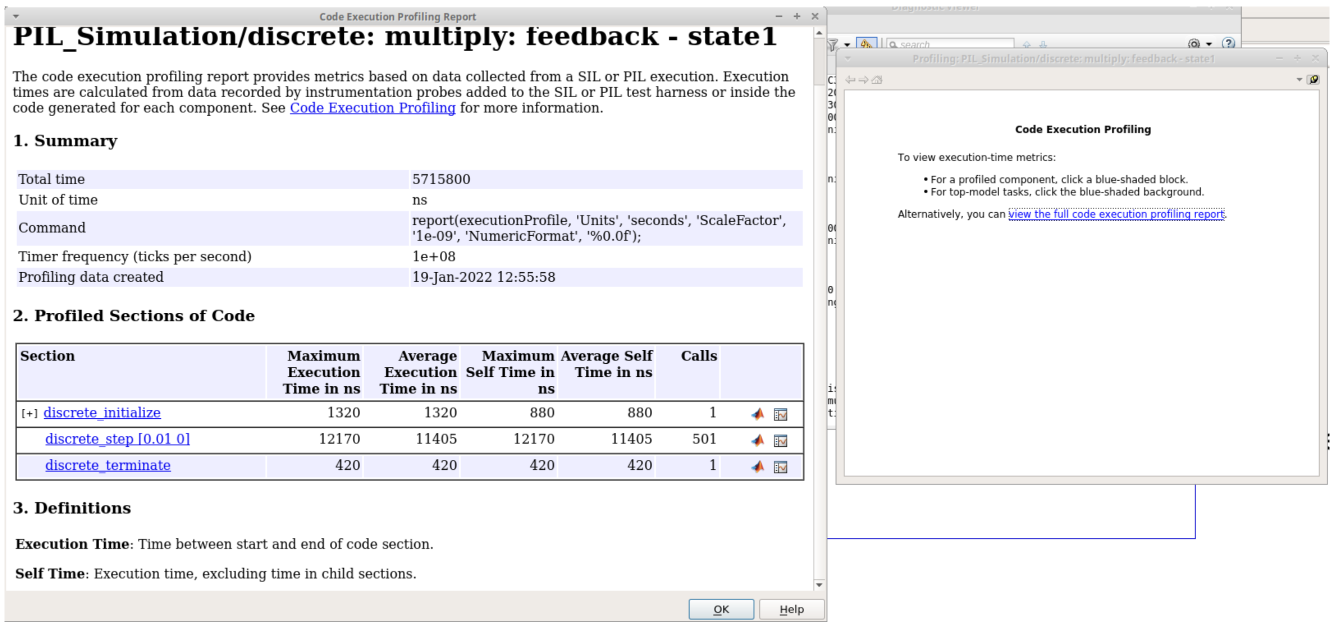
- For a profiled component, click a blue-shaded block.
- For top-model tasks, click the blue-shaded background.

Alternatively, you can view the full code execution profiling report.

```
Setup: 3 1 800c0000 800C3000 FFFFF000
Setup: 3 2 80020000 80020000 FFFF0000
Setup: 3 3 80030000 80030000 FFFF0000
Setup: 3 4 81000000 81000000 FF000000
Opened state file: ./monitoring-tools/state.json
Found tiles
Process tile: 0
Set schedule : 0 0 5000
Opened state file: ./monitoring-tools/state.json
Found tiles
Process tile: 1
Set schedule : 0 0 5000
Set schedule : 1 1 100000
Opened state file: ./monitoring-tools/state.json
Found tiles
Process tile: 2
Set schedule : 0 0 5000
Set schedule : 1 3 10000
./rerun.sh: reconfiguring TDM schedule at RISC-V clock cycle 0xe1fff63dd (60666373085,
14/536830941)
Write ok
Write ok
Write ok
ans =
     0
Upload on Verintech finished start synqing
### Starting the PIL simulation
### Stopping PIL simulation
```

Figure 9: Result of a successful PIL simulation.

By double clicking the view the full code execution profiling report link on `Profiling` window, you should obtain a similar window to the one shown in Figure 10.


Code Execution Profiling Report
OK
Help
Profiling: PIL_Simulation/discrete: multiply: feedback - state1

PIL_Simulation/discrete: multiply: feedback - state1

The code execution profiling report provides metrics based on data collected from a SIL or PIL execution. Execution times are calculated from data recorded by instrumentation probes added to the SIL or PIL test harness or inside the code generated for each component. See Code Execution Profiling for more information.

1. Summary

Total time	5715800
Unit of time	ns
Command	report(executionProfile, 'Units', 'seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	1e+08
Profiling data created	19-Jan-2022 12:55:58

2. Profiled Sections of Code

Section	Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls	
[+] discrete_initialize	1320	1320	880	880	1	
discrete_step [0.01 0]	12170	11405	12170	11405	501	
discrete_terminate	420	420	420	420	1	

3. Definitions

Execution Time: Time between start and end of code section.

Self Time: Execution time, excluding time in child sections.

Code Execution Profiling

To view execution-time metrics:

- For a profiled component, click a blue-shaded block.
- For top-model tasks, click the blue-shaded background.

Alternatively, you can view the full code execution profiling report.

Figure 10: Example 'full code execution profiling' report.

The execution time value of interest for the delay is the one labeled discrete_step [0.01 0], specifically the maximum execution time.

8 Configuring the TDMA scheduling

8.1 Hardware architecture

The Hardware layer of the CompSOC platform consists of three processor tiles. These tiles can communicate through 1-v-1 shared memories. Figure 11 demonstrates the hardware architecture:

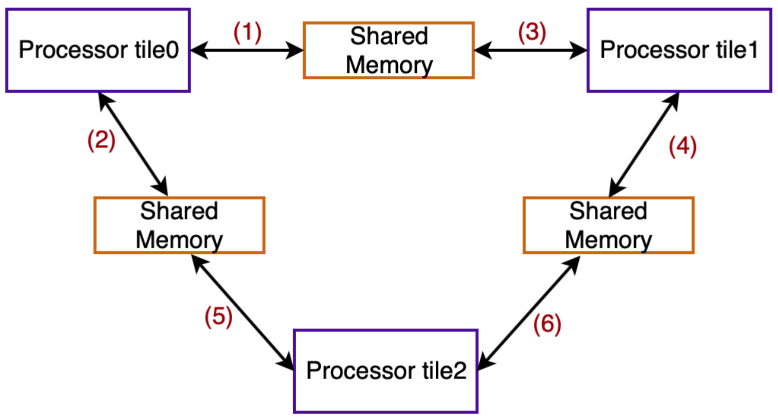


Figure 11: CompSOC hardware architecture.

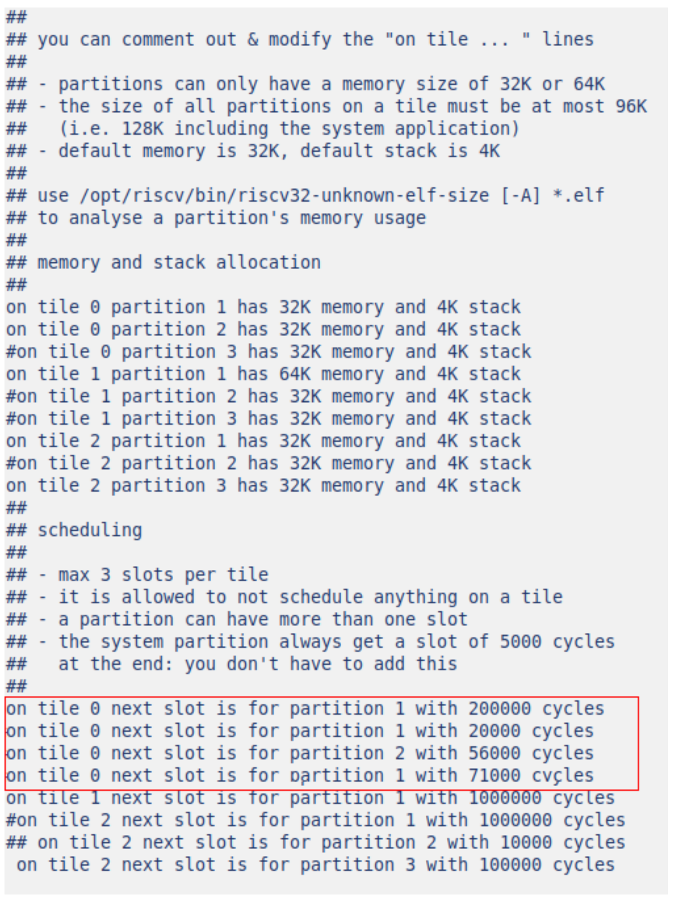
In order to perform HIL simulation with your designed TDMA scheduling, you need to modify the time slots allocated for your controller applications. You can define the scheduling of each processor tile to order the execution of applications on the tile. This can be done through the provided file called vep-config.txt. This file is responsible for defining the number of partition slots, size of each partition slots, and the application allocated to each partition slot. You can find the file via File Browser on the shown path.

```
    mousepad CompSOC_ec_target/CompSOC_ec/+CompSOC_ec/vep-config.txt
```

You can see the configuration file's structure on Figure 12. The red box shows the part which you should modify. Tile 0 is used for your controller application and therefore you should only modify the number of cycles partitioned for Tile 0. Note that # at the beginning of a line comments the line out.

Please note that:

1. Do not modify partitions for Tile 1 and Tile 2; they are reserved for the simulation of the plant! Do not modify memory partitions for any tile including Tile 0!
2. There is always a 5000 cycles partition slot for a system application at the end of the TDM table.
3. There is a context switching CoMik slot with a fixed size of 2000 clock cycles between each two partition slots.
4. Keep in mind that the system clock frequency is 40MHz .



```
##
## you can comment out & modify the "on tile ... " lines
##
## - partitions can only have a memory size of 32K or 64K
## - the size of all partitions on a tile must be at most 96K
##   (i.e. 128K including the system application)
## - default memory is 32K, default stack is 4K
##
## use /opt/riscv/bin/riscv32-unknown-elf-size [-A] *.elf
## to analyse a partition's memory usage
##
## memory and stack allocation
##
on tile 0 partition 1 has 32K memory and 4K stack
on tile 0 partition 2 has 32K memory and 4K stack
#on tile 0 partition 3 has 32K memory and 4K stack
on tile 1 partition 1 has 64K memory and 4K stack
#on tile 1 partition 2 has 32K memory and 4K stack
#on tile 1 partition 3 has 32K memory and 4K stack
on tile 2 partition 1 has 32K memory and 4K stack
#on tile 2 partition 2 has 32K memory and 4K stack
on tile 2 partition 3 has 32K memory and 4K stack
##
## scheduling
##
## - max 3 slots per tile
## - it is allowed to not schedule anything on a tile
## - a partition can have more than one slot
## - the system partition always get a slot of 5000 cycles
##   at the end: you don't have to add this
##
on tile 0 next slot is for partition 1 with 200000 cycles
on tile 0 next slot is for partition 1 with 20000 cycles
on tile 0 next slot is for partition 2 with 56000 cycles
on tile 0 next slot is for partition 1 with 71000 cycles
on tile 1 next slot is for partition 1 with 1000000 cycles
#on tile 2 next slot is for partition 1 with 1000000 cycles
## on tile 2 next slot is for partition 2 with 10000 cycles
 on tile 2 next slot is for partition 3 with 100000 cycles
```

Figure 12: File for configuring the TDMA scheduling.

Considering the mentioned remarks, the example in Figure 12, results in a TDM table illustrated in 13 and a sampling period of 362000 clock cycles or $9.05ms$.

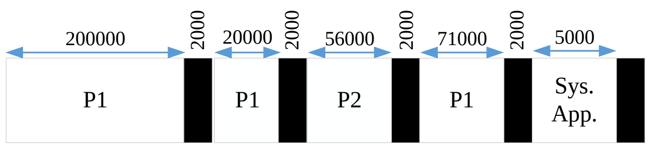


Figure 13: The resulted TDMA scheduling.

9 Performing HIL simulation on the platform

9.1 Implementing controllers for HIL Simulation

The necessary files to perform HIL simulations are provided in folder HIL simulations 2022 (Check 4). You are provided with a pair of Simulink models with names

- `DC_Motor_HIL.slx`
- `Dual_rotary_HIL.slx`

Both of these Simulink models Provide a somplete simulation model to perform an HIL simulations for each of the systems under study. Before using these models you need to update your feedback and feedforward values and also initiate the model sampling period. The models contain the blocks that send and receive information from the board. These blocks are packed in *Communication to Plant on Hardware* subsystem. Figure 14 shows the provided blocks inside this subsystem for DC Motor system and and Figure 15 shows them for Dual Rotary system.

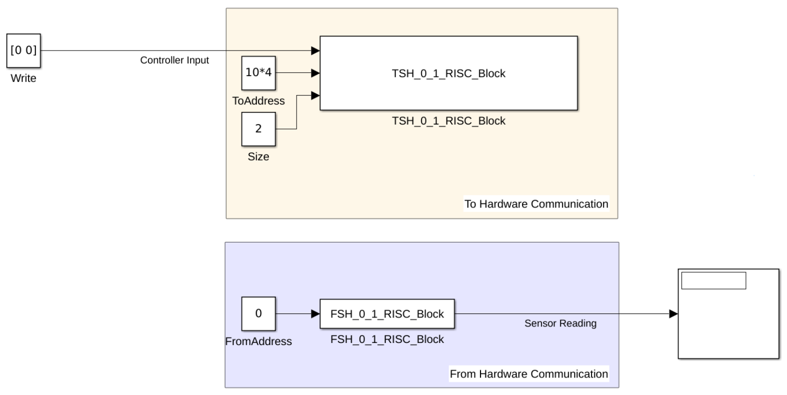


Figure 14: Provided communication blocks for DC Motor system.

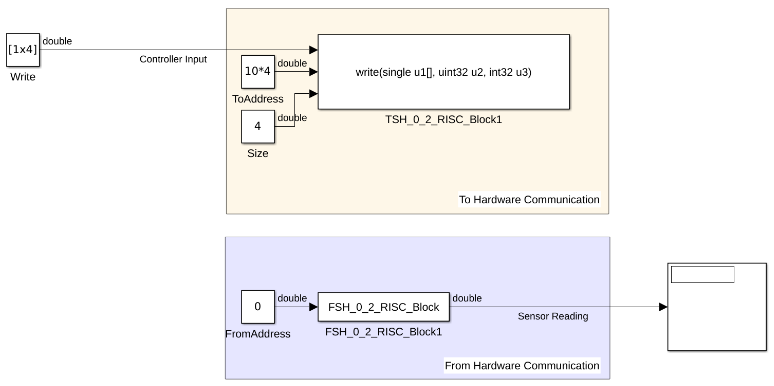


Figure 15: Provided communication blocks for Dual Rotary system.

Yellow areas in both models function to send data to respective applications. These blocks sends the control input to the plant under control. Conversely, the violet areas function to receive data from the plant, which corresponds to reading sensor outputs. Consequently, you are required to design your controllers to operate between these two blocks for both systems. As long as sensor outputs and controller actions are passed correctly between Simulink and the board, you can design the controller in any way you desire.

9.2 Establishing board connection for controller simulation

Connecting to the board for HIL simulation is similar to step 3. In the same fashion, after connecting to the board via `SSH` using command

and execute the following commands. The again, similar to Step 3, open a separate terminal shell and execute either one of the following pair of commands. The difference between them is the target application for the communication. Hence, in order to run simulations for dual-rotary system or DC motor system, execute the respective command.

```
    cd tutorial/monitoring-tools/
```

- DC Motor

```
    sudo ./channel_cheap_bridge 0 2 9879
```

- Dual-rotary

```
    sudo ./channel_cheap_bridge 0 1 9878
```

Once `readout.sh` is running and the connection is established as in Step 3, you can continue with the next step.

9.3 Building controller models and running the simulation

Building and running the simulation is straightforward once all the previous steps are correctly completed. Build the controller by clicking on `Build` button and start the simulation by clicking on `Connect to Target` button. You can record your variables by including a `ToWorkspace` block in Simulink. Figure 16 shows a sample situation when the model is successfully built and running.

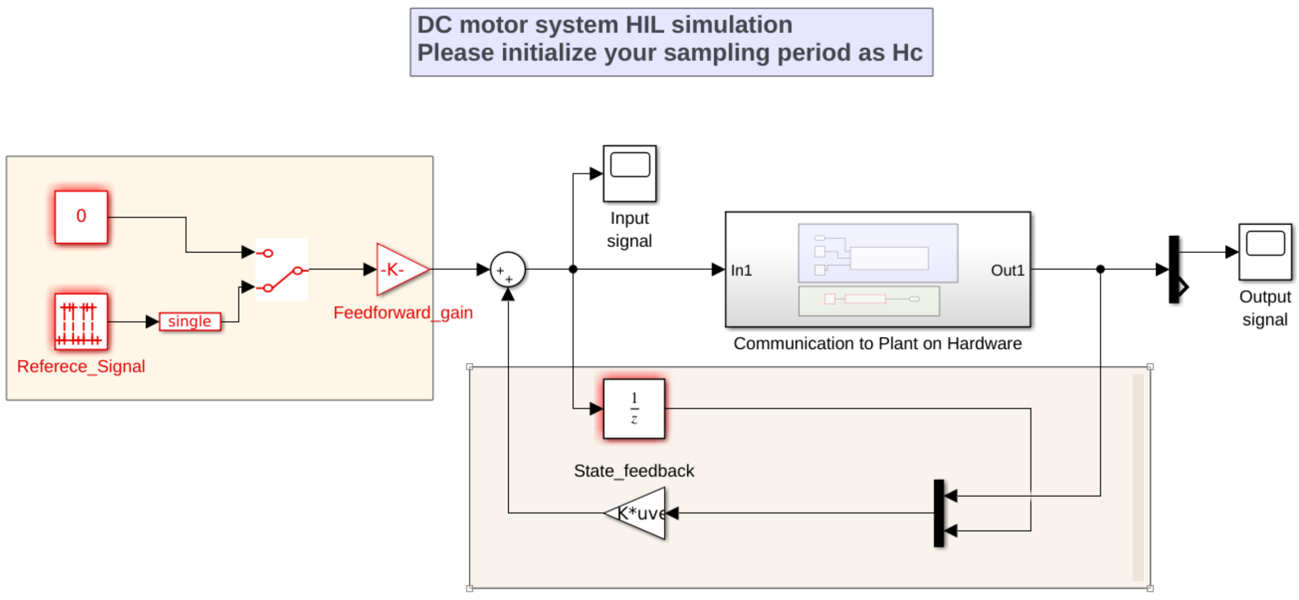

DC motor system HIL simulation
Please initialize your sampling period as Hc
0
single
Referece_Signal
-K-
Feedforward_gain
Input signal
In1
Out1
Communication to Plant on Hardware
Output signal
1/z
State_feedback
K*uvec

Figure 16: A correctly built and running HIL simulation.

If the simulation terminates abruptly *mid-execution* due to an error you have to rebuild and restart it with the previously mentioned buttons.