

# IMACS User Guide for Processor-in-the-Loop (PiL) Simulation

## Section I: PiL implementation for sequential configuration

- open MATLAB: To open MATLAB in the IMACS virtual machine, run the following command in the new terminal.

**Note:** If MATLAB is already opened, then no need to reopen it.

```
$ matlab
```

```
computation@computation-virtual-machine: ~
computation@computation-virtual-machine:~$ matlab
MATLAB is selecting SOFTWARE OPENGL rendering.
```

- open CoppeliaSim: to open the CoppeliaSim, run the following command in a new terminal.

**Note:** If CoppeliaSim is already opened, then no need to reopen it.

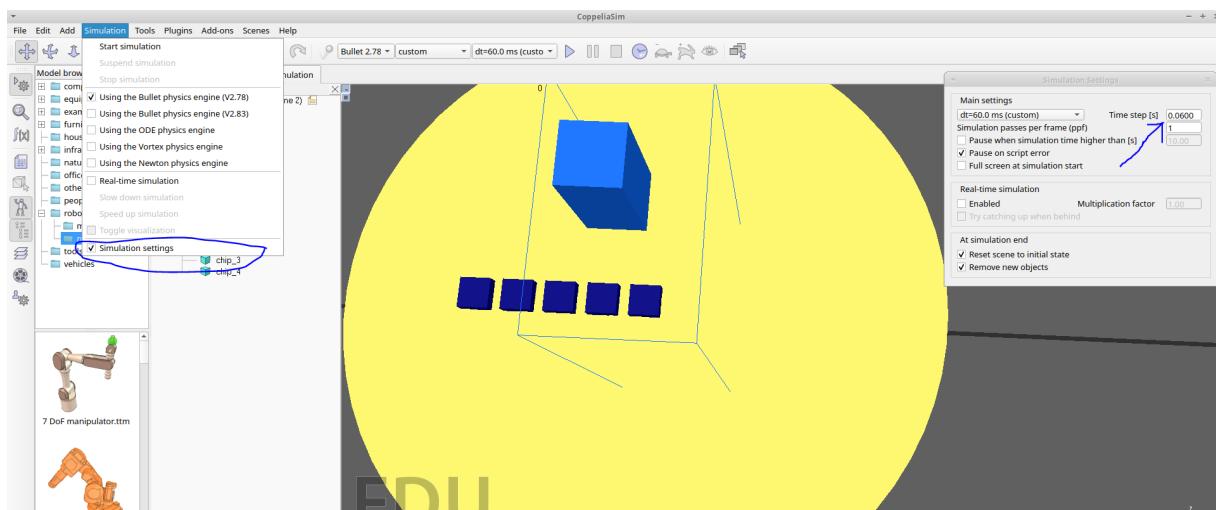
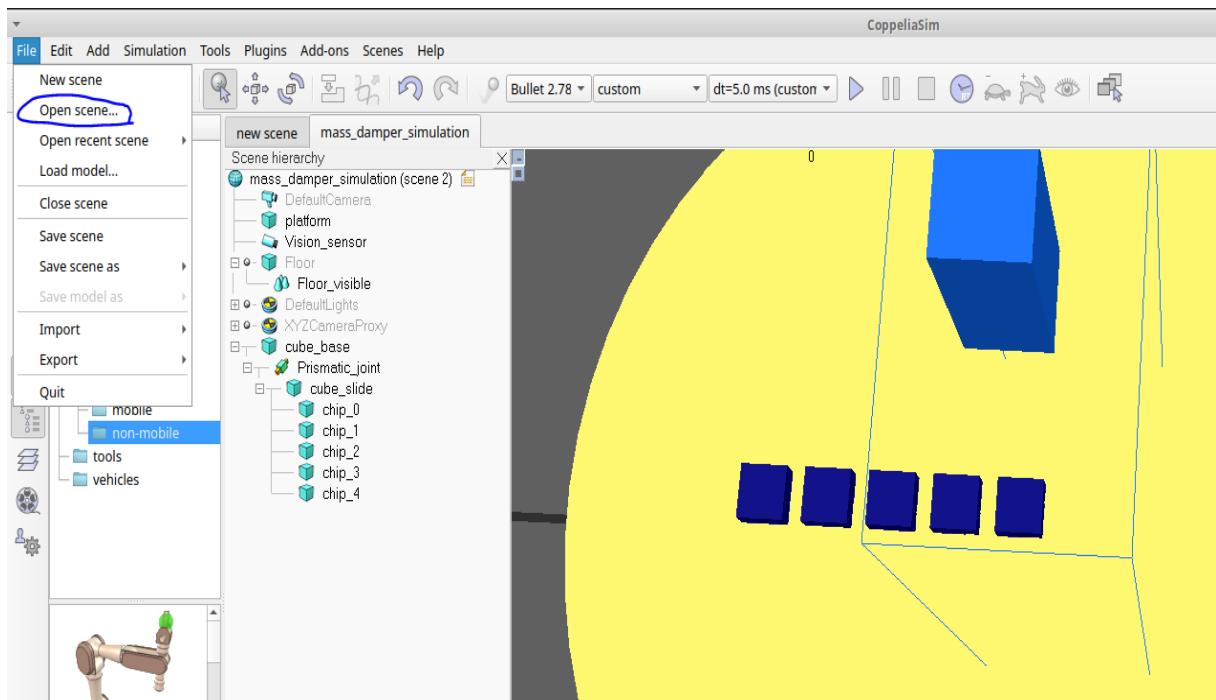
```
$ vrep
```

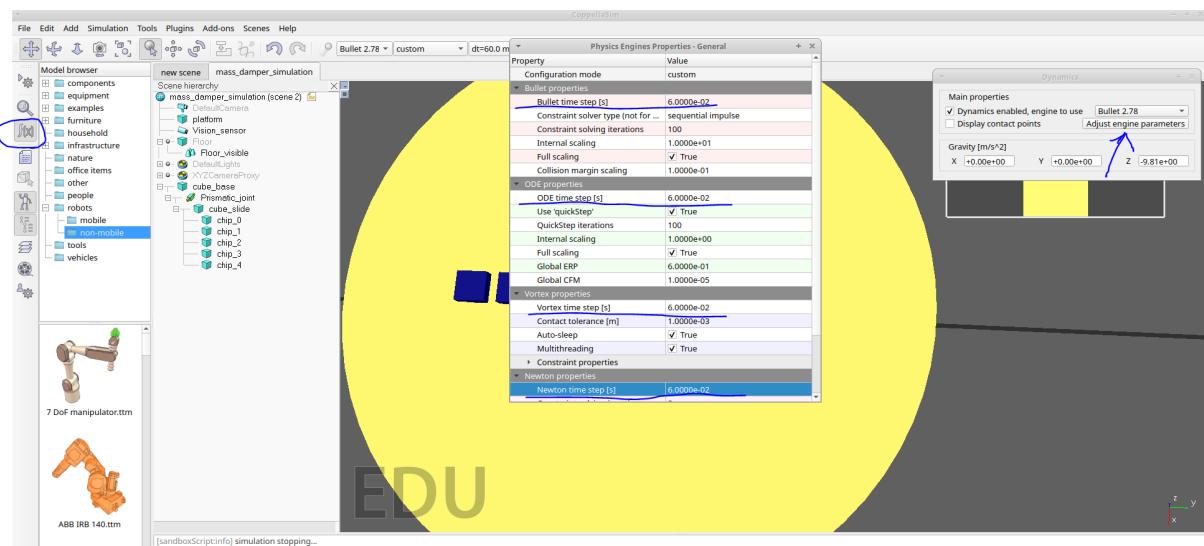
```
File Edit View Terminal Tabs Help
computation@computation-virtual-machine:~$ vrep
[CoppeliaSimClient]    loading the CoppeliaSim library...
[CoppeliaSimClient]    done.
[CoppeliaSimClient:loadinfo]  launching CoppeliaSim...
[CoppeliaSim:loadinfo]  CoppeliaSim V4.2.0., (rev. 5), flavor: 1
[CoppeliaSim:loadinfo]  Legacy machine ID: 5000-B6EB-FFC4-9C4A-F7E3-D41D
[CoppeliaSim:loadinfo]  Machine ID: 67BA-736C-64E4-0000-713E-0101
[CoppeliaSim:loadinfo]  using the default Lua library.
[CoppeliaSim:loadinfo]  loaded the video compression library.
```

- open CoppeliaSim scene for the mass-damper system.
- go to File → Open scene → select the following directory  
`/home/computation/5lij0_ECS_Course_2023/PIL_Implementation/PIL_sequential_impl`

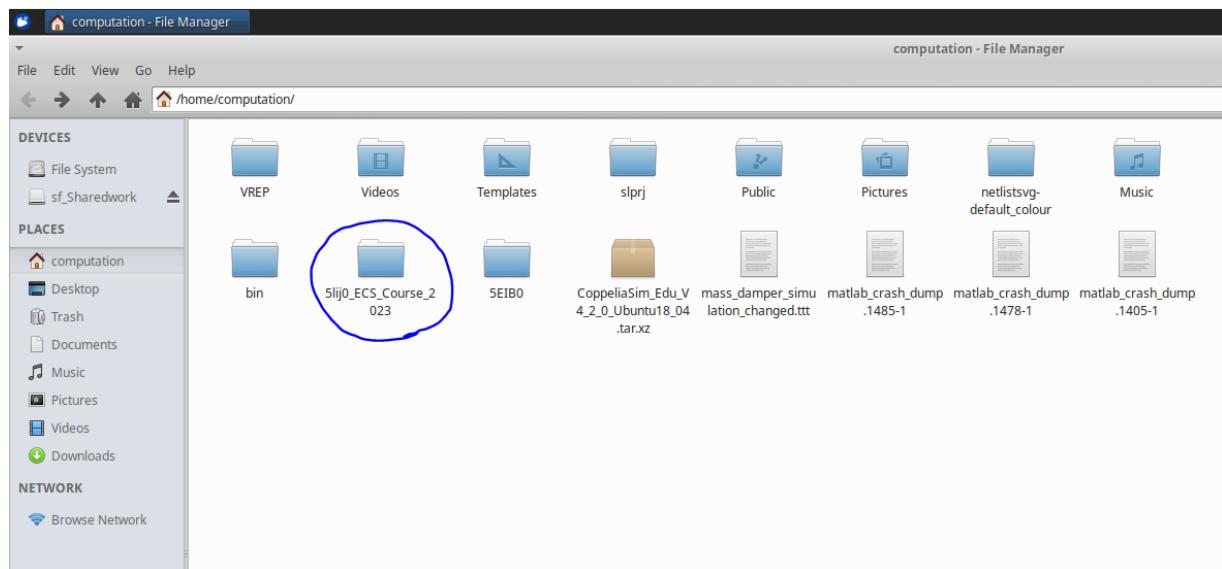
open **mass\_damper\_simulation.ttt**, check the following screenshot for reference.

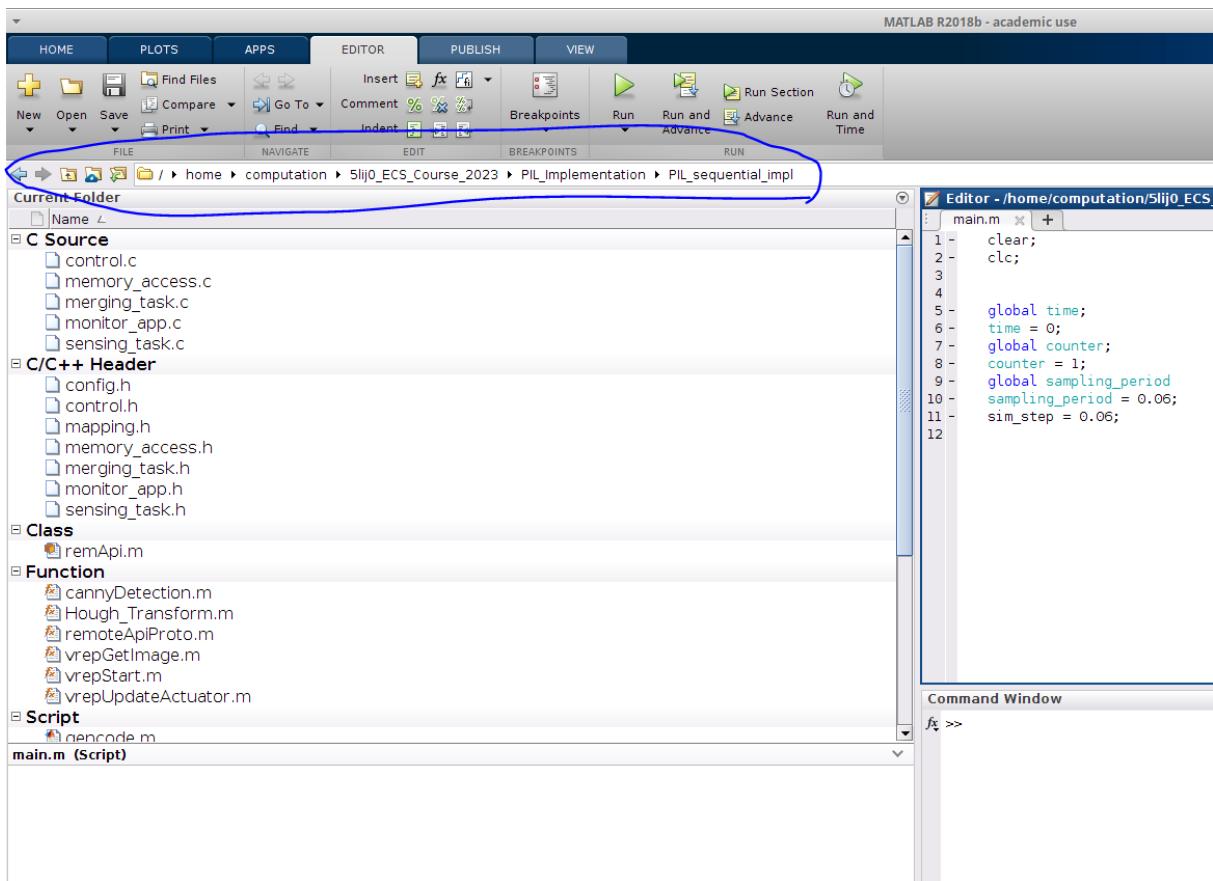
We have implemented the PIL of sequential configuration considering the 60ms sample period. The same sample period should be used in CoppeliaSim. In the case of different sample periods, students can make the changes in the CoppeliaSim, as explained below. For example, the following screenshot shows how to change the time step in the CoppeliaSim scene.





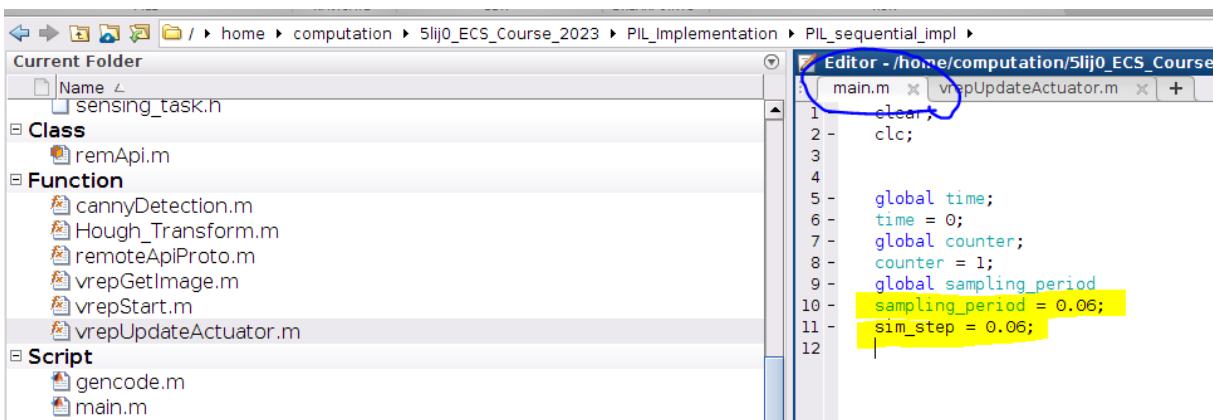
- open the folder for PiL sequential implementation.
  - go to the following folder path and open in MATLAB
   
computation → 5lij0\_ECS\_Course\_2023 → PIL\_Implementation → PIL\_sequential\_impl





- open the main.m file and make changes for the highlighted parameters, as shown in the following screenshot.

Note: In this screenshot, we have explained for the 60ms sample period, students can use different sample periods according to the implementation design and make the changes for the following parameters. Once the changes are done, run the main.m script. This file includes the initialization, which is required for the PiL simulation.



- open the gencode.m file and run the script. This file uses legacy code tool functionality, integrating the existing C or C++ functions into MATLAB Simulink. Once this gencode.m runs, it creates the S-function block inside the Simulink model.

The screenshot shows the MATLAB IDE interface. The current folder path is `/home/computation/Slijo_ECS_Course_2023/PIL_Implementation/PIL_sequential_impl`. The left pane displays a file tree with various MATLAB files like `sensing_task.h`, `remApi.m`, and `vrepUpdateActuator.m`. The right pane shows the MATLAB Editor with the code for `genode.m`. A blue circle highlights the first few lines of the code:

```

1 %> Monitor app
2 def_monitor = legacy_code('initialize');
3 def_monitor.SourceFiles = {'monitor_app.c', 'memory_access.c'};
4 def_monitor.HeaderFiles = {'monitor_app.h', 'memory_access.h', 'co
5 def_monitor.SFunctionName = 'monitor_app_block';
6 def_monitor.IncPaths = {'.', '..', 'CompSOC_ec_target/files'};
7 def_monitor.OutputFcnSpec = 'void monitor(uint8 uil[12496], single
8 legacy_code('sfcn_cmex_generate', def_monitor);
9 legacy_code('sfcn_tlc_generate', def_monitor);
10 legacy_code('compile', def_monitor);
11 % legacy_code('slblk_generate', def_monitor);
12
13 %% Sensing task
14 def_sensing = legacy_code('initialize');
15 def_sensing.SourceFiles = {'sensing_task.c', 'memory_access.c'};
16 def_sensing.HeaderFiles = {'sensing_task.h', 'memory_access.h', 'c
17 def_sensing.SFunctionName = 'sensing_block';
18 def_sensing.IncPaths = {'.', '..', 'CompSOC_ec_target/files'};
19 def_sensing.OutputFcnSpec = 'void hough_transform()';
20 legacy_code('sfcn_cmex_generate', def_sensing);
21 legacy_code('sfcn_tlc_generate', def_sensing);
22 legacy_code('compile', def_sensing);
23 % legacy_code('slblk_generate', def_sensing);
24
25 %% MERGING task
26

```

- open the **vrepUpdateActuator.m** file. Make the necessary changes in the file, as shown below in the screenshot. This file is required for the actuation.

The screenshot shows the MATLAB IDE interface. The current folder path is `/home/computation/Slijo_ECS_Course_2023/PIL_Implementation/PIL_sequential_impl`. The left pane displays a file tree with various MATLAB files like `sensing_task.h`, `remApi.m`, and `vrepUpdateActuator.m`. The right pane shows the MATLAB Editor with the code for `vrepUpdateActuator.m`. A blue circle highlights the line `function output = vrepUpdateActuator(input)`. The code also includes several global variable declarations at the bottom:

```

1 function output = vrepUpdateActuator(input)
2
3 % vrepParam = [
4 %   1 clientID      -> vrepParam(1)
5 %   2 cam           -> vrepParam(2)
6 %   3 motor          -> vrepParam(3)
7 %   4 chip           -> vrepParam(4)
8 %   5 base           -> vrepParam(5) ]
9
10 vrepParam = input(4:end);
11 force_lock = input(2);
12 force = input(1);
13
14 damping_constant = 1;
15 sim_step = 0.06;
16
17 global vrep;
18 global prev_sign;
19 global total_force;
20 global time;
21 global counter;
22 global prev_pos;
23 global sampling_period;

```

- open the **config.h** file. Make the necessary changes in the file, as shown below in the screenshot. This file contains all the required parameters used in the PiL simulation.

```

1 #ifndef CONFIG_H
2 #define CONFIG_H
3
4 // Sensing
5 #define WAITING_TIME ((uint32_t)200000) // Cycles ~ 5 ms
6 #define IMG_WIDTH (88) // Pixels
7 #define HALF_WIDTH (44) // Pixels
8 #define IMG_HEIGHT (142) // Pixels
9 #define IMG_SIZE (12496) // Pixels
10 #define HALF_IMG (6248)
11 #define HOUGH_THRESHOLD (25)
12 #define PADDING_SIZE_THETA (3) // Pixels
13 #define PADDING_SIZE_RHO (3) // Pixels
14 #define MAX_PEAK_NUMBER (2)
15 #define DESIRED_POS (44) // Pixels
16 #define PIXEL_RATIO (0.001) // 1 millimeter
17 #define THETA_neg_5_pos_5
18
19
20 // Controller
21 #define DAMPING_CONST ((float)1.0)
22 #define SAMPLING_TIME ((float)0.060)
23 #define INV_SAMPLING_TIME ((float)16.66)
24 #define K_P ((float)60)
25 #define K_D ((float)10)
26 #define K_I ((float)0)
27 #define PI (3.1416)
28 #endif

```

- for tuning the PID controller gains, students can change the gains in the **config.h** file, check the below screenshot.

```

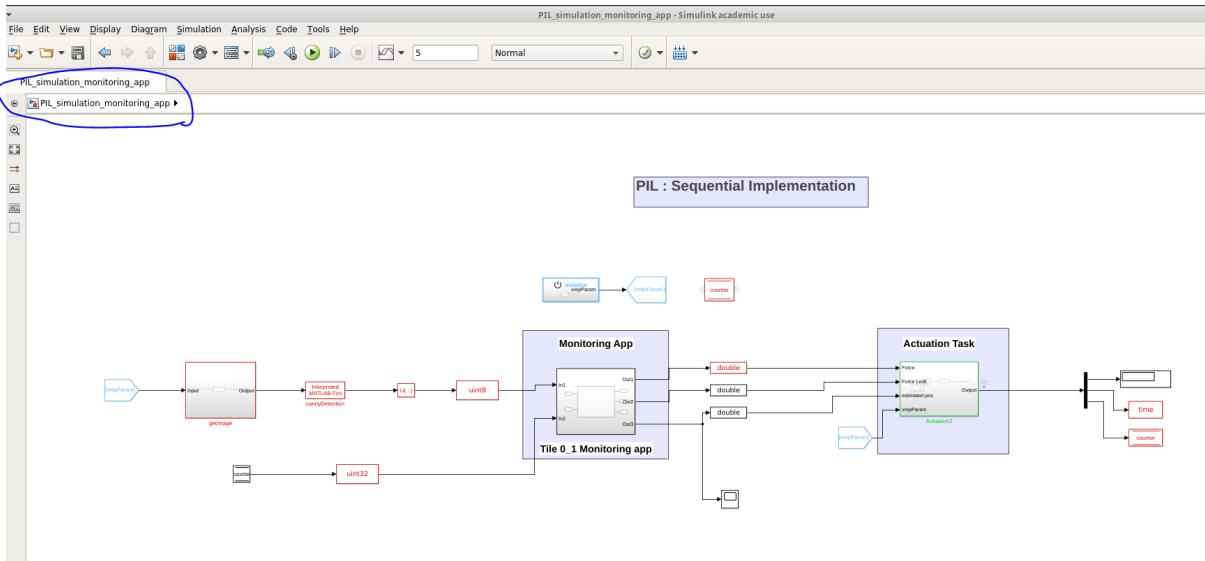
1 #ifndef CONFIG_H
2 #define CONFIG_H
3
4 // Sensing
5 #define WAITING_TIME ((uint32_t)200000) // Cycles ~ 5 ms
6 #define IMG_WIDTH (88) // Pixels
7 #define HALF_WIDTH (44) // Pixels
8 #define IMG_HEIGHT (142) // Pixels
9 #define IMG_SIZE (12496) // Pixels
10 #define HALF_IMG (6248)
11 #define HOUGH_THRESHOLD (25)
12 #define PADDING_SIZE_THETA (3) // Pixels
13 #define PADDING_SIZE_RHO (3) // Pixels
14 #define MAX_PEAK_NUMBER (2)
15 #define DESIRED_POS (44) // Pixels
16 #define PIXEL_RATIO (0.001) // 1 millimeter
17 #define THETA_neg_5_pos_5
18
19
20 // Controller
21 #define DAMPING_CONST ((float)1.0)
22 #define SAMPLING_TIME ((float)0.060)
23 #define INV_SAMPLING_TIME ((float)16.66)
24 #define K_P ((float)60) // previous values
25 #define K_D ((float)10)
26 #define K_I ((float)0)
27 #define PI (3.1416)
28 #endif

```

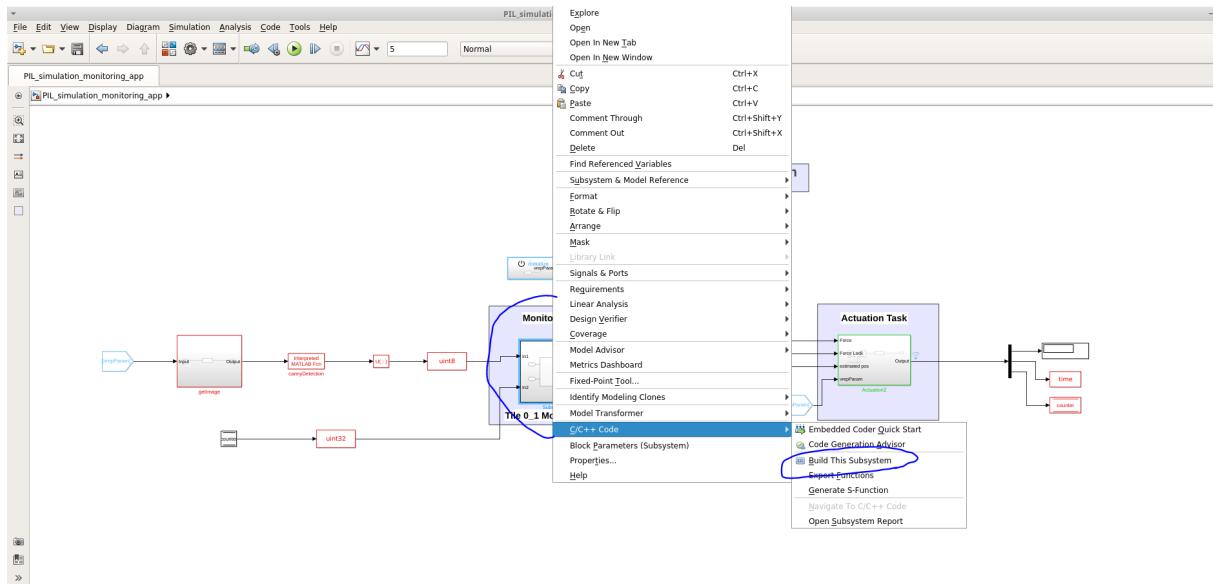
- once the steps mentioned above are done, open the PiL simulation file named **PIL\_simulation\_monitoring\_app.slx**

Students can find this file in the following directory:

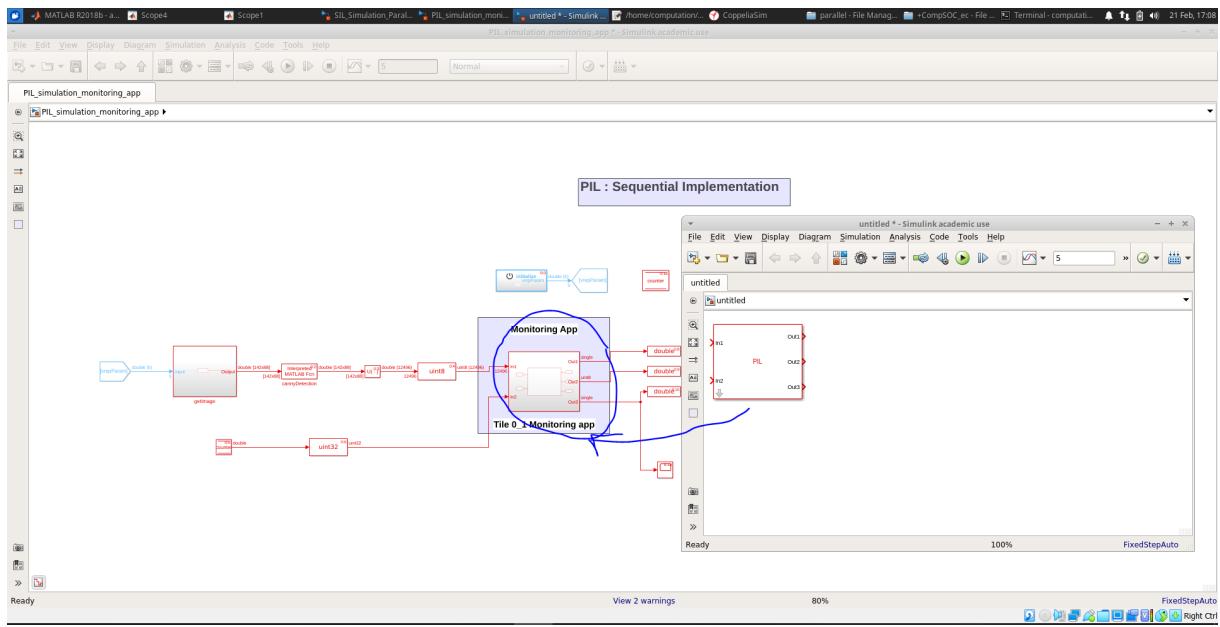
computation→5lij0\_ECS\_Course\_2023→PIL\_Implementation→PIL\_sequential\_Impl



- right-click on the block, select the C/C++ code option, and click on build this subsystem. Check the following screenshot for reference.



- once, the build process is completed successfully; it will generate the PIL block; check the following screenshot:



- after completing the above steps, check the vep\_config.txt file. In this example, we have considered the 60ms sample period. Therefore, in the vep\_config.txt, Time-division multiplexing (TDM) scheduling is given according to the 60ms sample period.
  - we have provided the vep\_config.txt file in the following folder  
`/home/computation/5lij0_ECS_Course_2023/PIL_Implementation/PIL_sequential_Impl`
  - students need to copy the contents from the above vep\_config file into the main **vep-config.txt** file, which is inside the following folder path :  
or the student can enter the TDM scheduling calculation directly in the vep-config.txt file, which is stored in the following location.

/home/computation/CompSOC\_ec\_target/CompSOC\_ec/+CompSOC\_ec/vep-config.txt

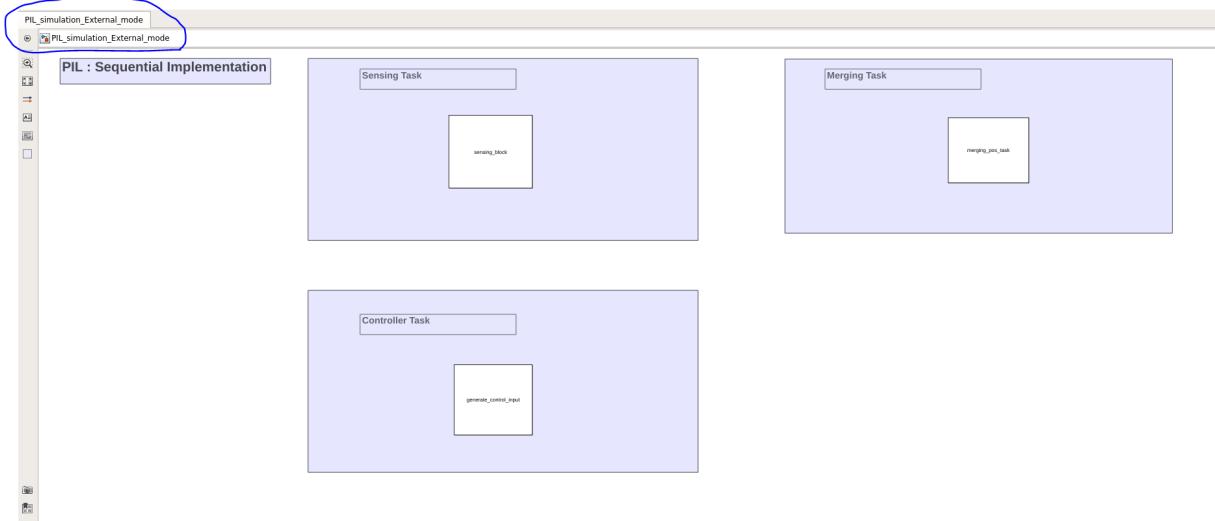
- the contents in the vep\_config file should be as follows:

```

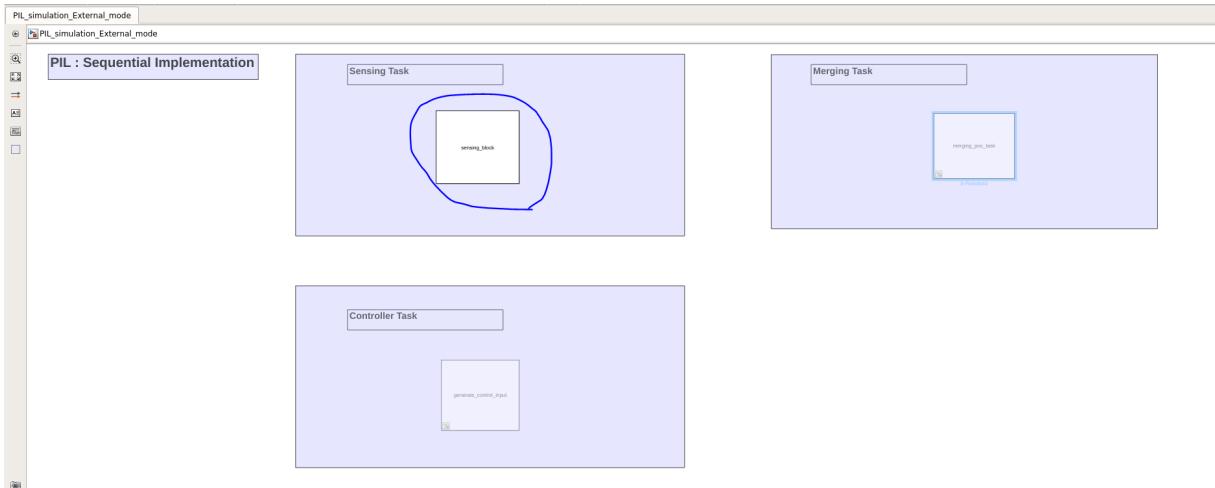
## you can comment out & modify the "on tile ... " lines
##
## - partitions can only have a memory size of 32K or 64K
## - the size of all partitions on a tile must be at most 96K
##   (i.e. 128K including the system application)
## - default memory is 32K, default stack is 4K
##
## use /opt/riscv/bin/riscv32-unknown-elf-size [-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 64K 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 1600000 cycles
#on tile 0 next slot is for partition 2 with 10000 cycles
#on tile 0 next slot is for partition 3 with 777000 cycles
#on tile 1 next slot is for partition 1 with 1600000 cycles
#on tile 1 next slot is for partition 2 with 789000 cycles
#on tile 1 next slot is for partition 3 with 372000 cycles
#on tile 2 next slot is for partition 1 with 700000 cycles
#on tile 2 next slot is for partition 2 with 289000 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

```

- open the PIL\_simulation\_External\_mode.slx. The folder path to this Simulink model is as follows:  
`/home/computation/5lij0_ECS_Course_2023/PIL_Implementation/PIL_sequential_impl`
- PIL\_simulation\_External\_mode.slx file has three different tasks, i.e., sensing, merging, and controller, which will run in the external mode on CompSOC based on the TDM scheduling. To run each task independently in the external mode, follow the below steps and check the screenshot for reference:

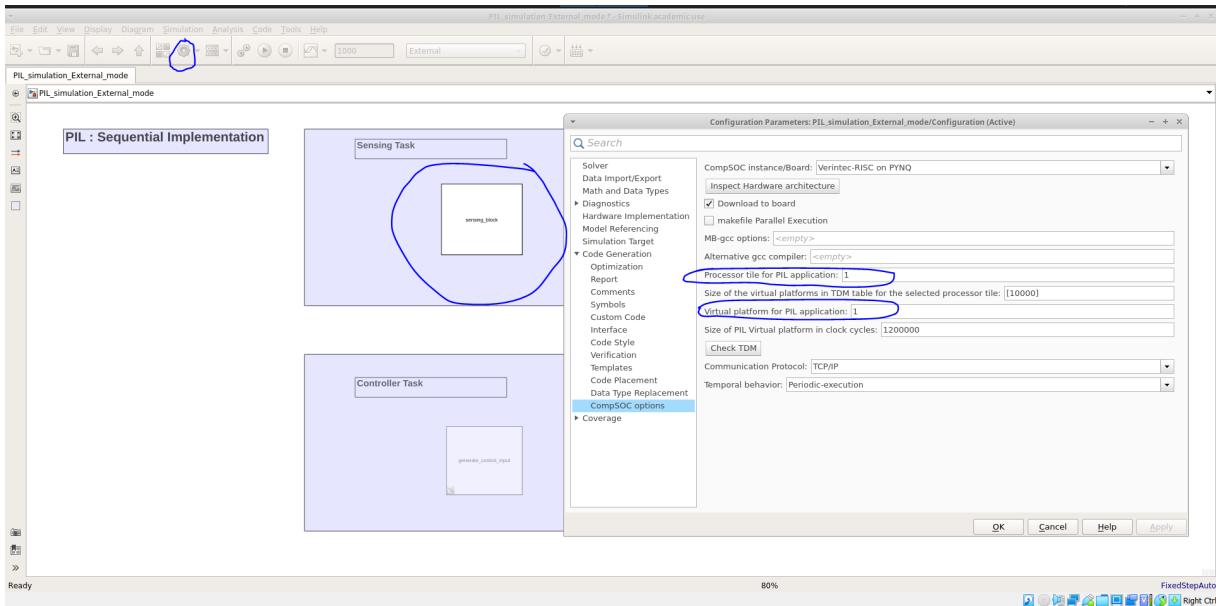


- to run the sensing task in the external mode, comment the other Simulink block and only uncomment the block which needs to be run. In the below screenshot, we consider the sensing task block first, and other blocks kept commenting.



- open the configuration setting and check the required setting for the particular task. In this example, we are running a sensing task on processor 1 and partition 1. Therefore, consider the following changes :

- Processor tile for PIL application: 1
- Virtual platform for PIL application: 1



- open the two new terminal windows, connect the PYNQ board, and use the following command:
  - ensure that vep-config.txt is updated in the main CompSOC folder as per the required TDM scheduling. This step is essential before running the task in external mode on CompSOC.
  - vep-config file path:  
`/home/computation/CompSOC_ec_target/CompSOC_ec/+CompSOC_ec/vep-config.txt`
  - commands for running the sensing task on processor 1 and partition 1

- Terminal 1 :
  - `cd tutorial`
  - `cd monitoring-tools/`
  - `sudo ./channel_cheap_bridge 1 1 9878`
- Terminal 2 :
  - `cd tutorial`
  - `./readout.sh`

```

computation@computation-virtual-machine:~ 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.

student@pato-board:~/tutorial$ ./readout.sh
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

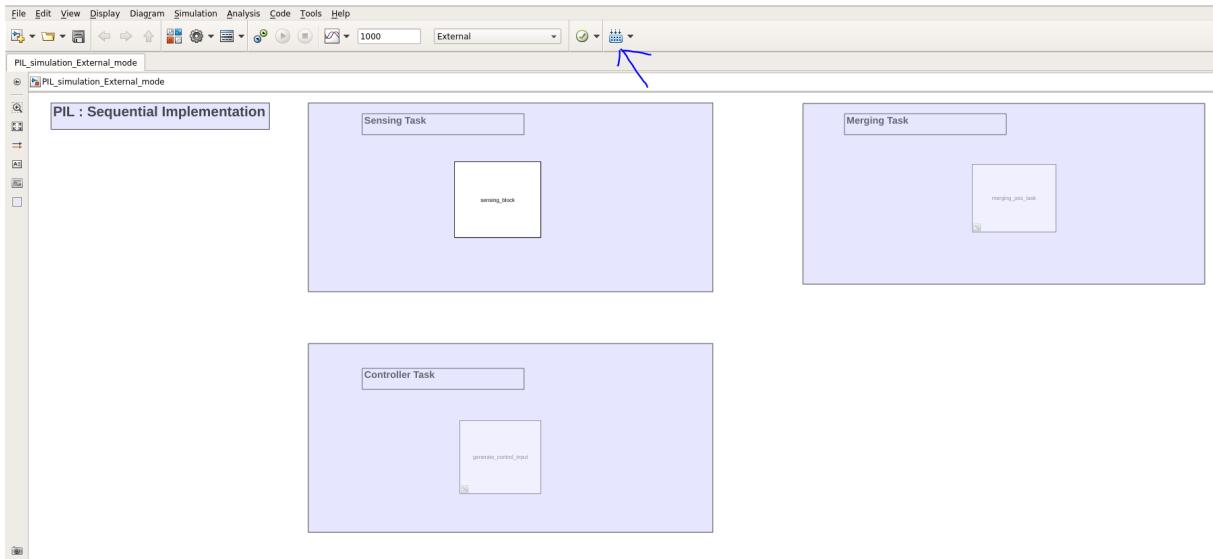
0 packages can be updated.
0 updates are security updates.

New release '20.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

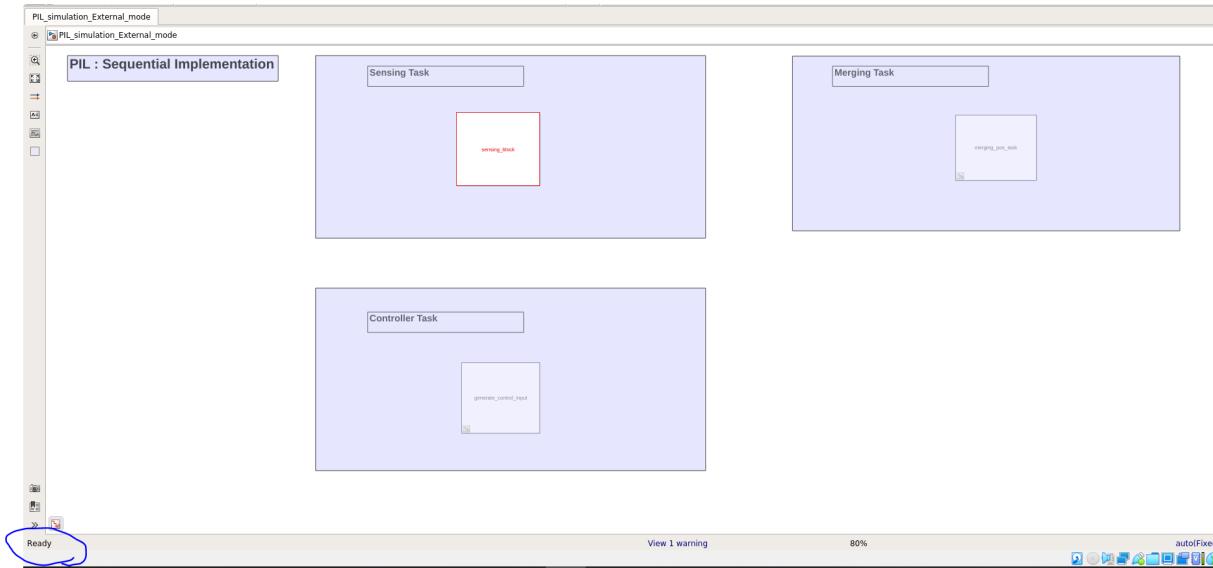
Last login: Tue Feb 21 04:59:55 2023 from 10.42.0.1
student@pato-board:~$ cd tutorial
student@pato-board:~/tutorial$ ./readout.sh
mem: 16 KB @ 0x0x80000000
CHeap stdout initialised
mem: 16 KB @ 0x0x80004000
CHeap stdou initialised
mem: 16 KB @ 0x0x80008000
CHeap stdou initialised
start reading

```

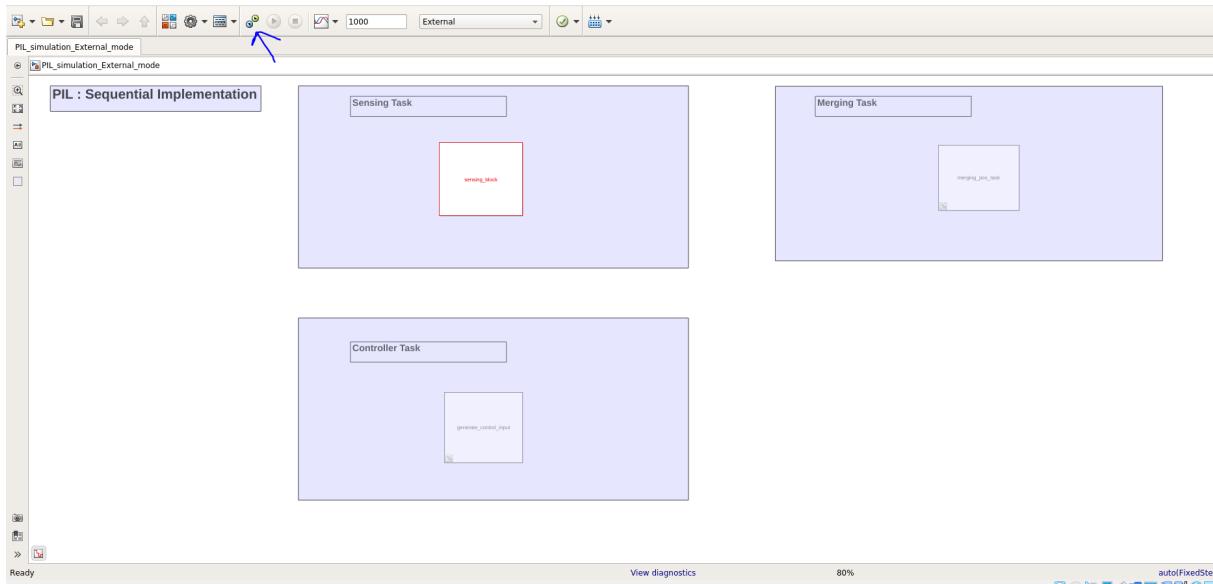
- click on the build button, as shown by the arrow in the below screenshot.



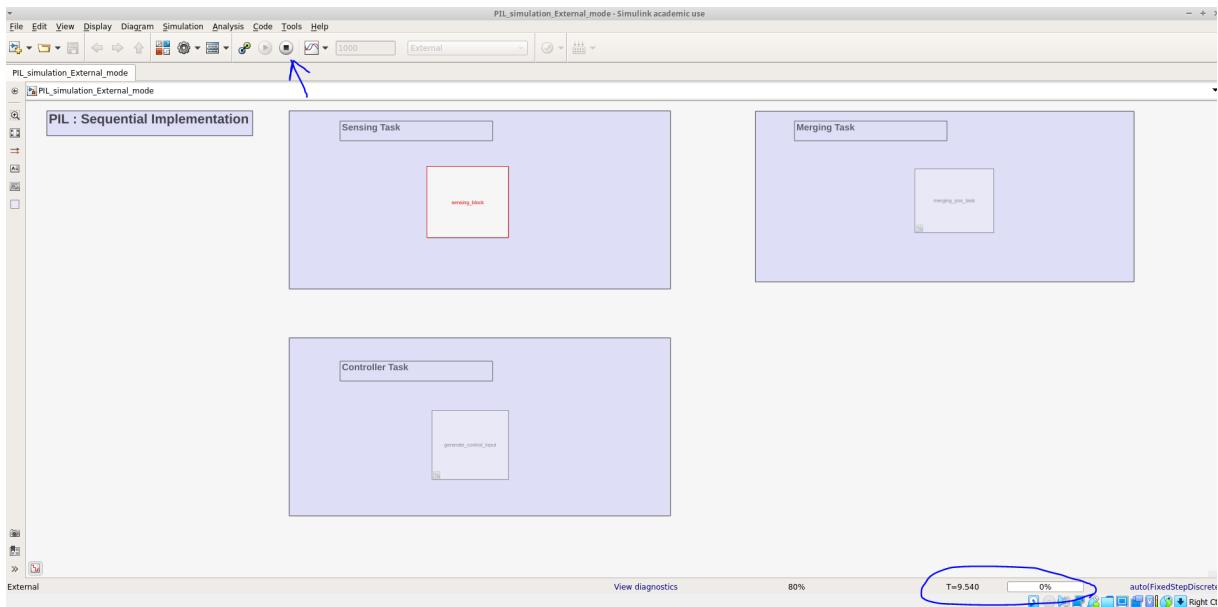
- after the build finishes, the Simulink model shows ready status; check the circle in the following screenshot.



- if the build finishes successfully, ready status is available in the model. After that, run the task in external mode by clicking the connect to target button. Check the following screenshot for reference.

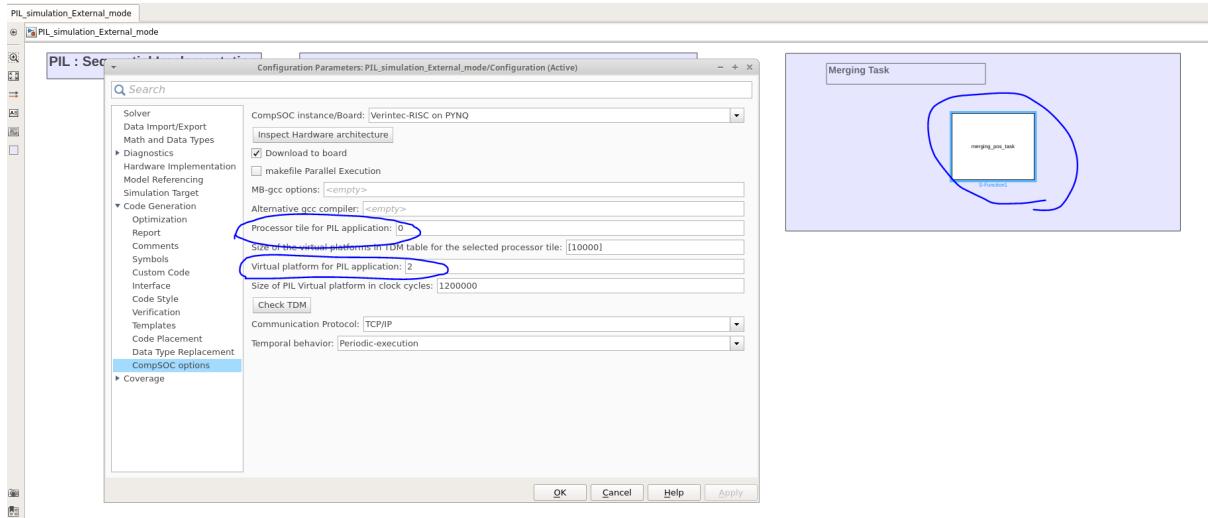


- after the above step, the sensing task connects to the target and starts running on the CompSOC; let it starts and runs for some time; check the highlighted circle in the below screenshot, then stop the running of the task by clicking on the stop button, check the arrow in the screenshot. No need to run the task for a long time.



- move to the next task and comment all blocks except the one you want to run in external mode. Check the following screenshot; we have considered merging task after finishing the sensing task. Only the merging task is uncommented. Other blocks kept commenting.
- check the configuration setting. We are running the merging task on processor 0 and partition 2. So, the following setting should be considered,

- Processor tile for PIL application : 0
- Virtual platform for PIL application: 2



- in the same two terminal windows opened for the previous task, you can use the following highlighted commands from the screenshot.

- Terminal 1 :
  - cd tutorial
  - cd monitoring-tools/
  - sudo ./channel\_cheap\_bridge 0 2 9878
- Terminal 2 :
  - cd tutorial
  - ./readout.sh

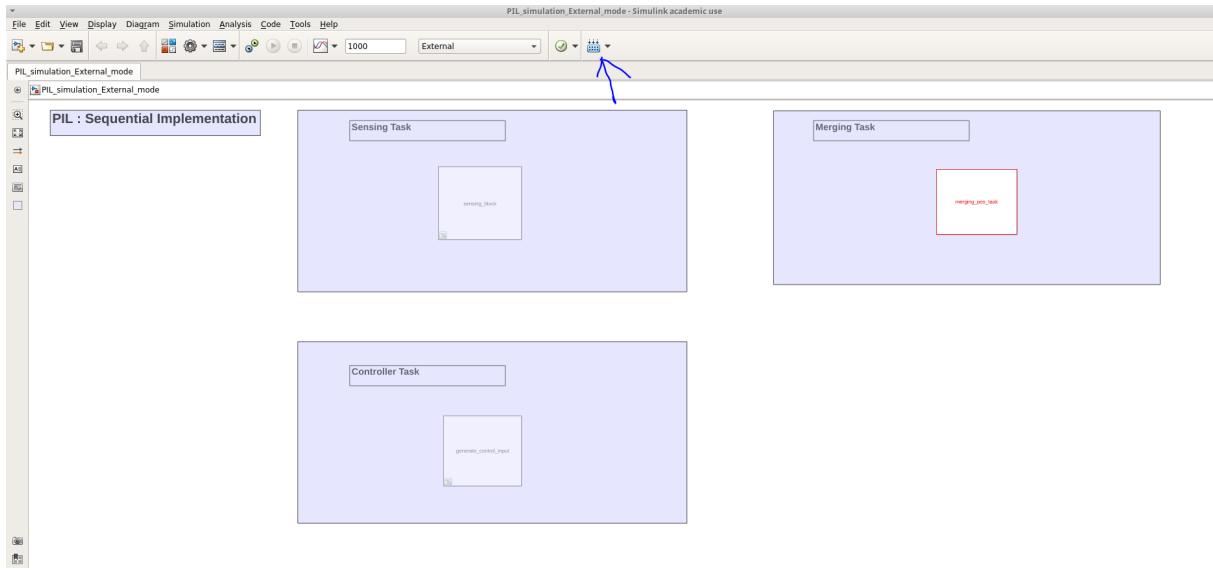
```

computation@computation-virtual-machine:~ computation@computation-virtual-machine:~ computation@computation-virtual-machine:~
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.
- Accepting socket
+Connection accepted
+Connection closed
0.000011
0.000029
+Waiting for incoming connections.
^C
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 0 2 9878

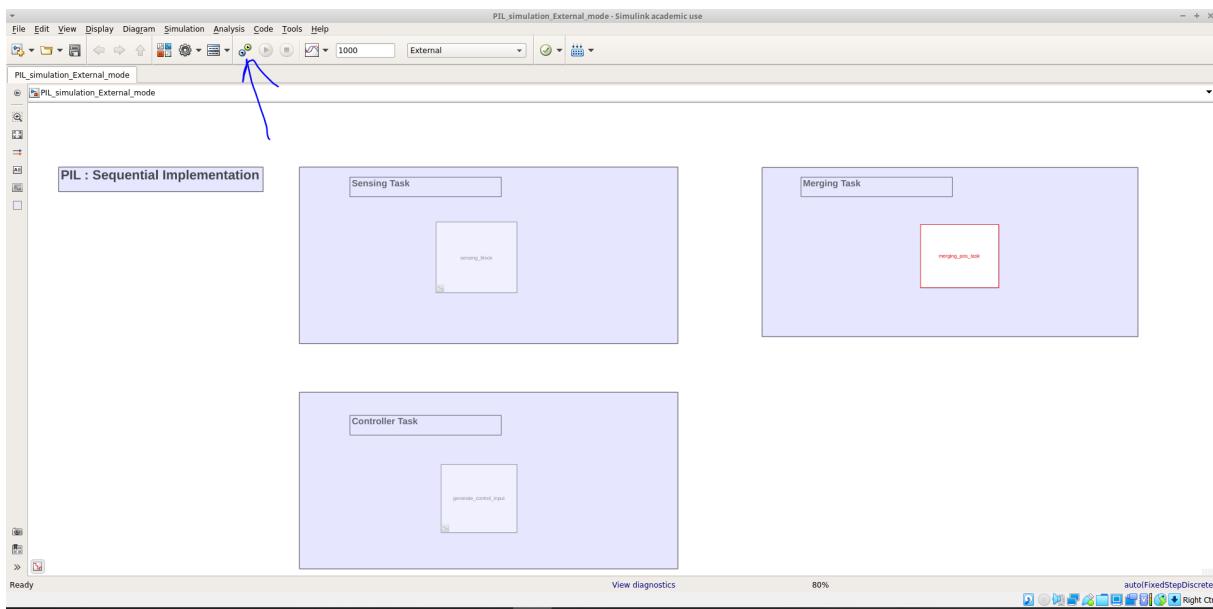
Terminal - computation@computation-virtual-machine:-
File Edit View Terminal Tabs Help
Last login: Tue Feb 21 04:59:55 2023 from 10.42.0.1
student@pato-board:~$ cd tutorial
student@pato-board:~/tutorial$ ./readout.sh
mem: 16 KB @ 0x0000000000000000
CHeap stdout initialised
mem: 16 KB @ 0x0000000000000000
CHeap stdout initialised
mem: 16 KB @ 0x0000000000000000
CHeap stdout initialised
start reading
01 01: Open stream
00 01: Open stream
00 01: Open stream
01 02: Close stream
01 01: Close stream
00 02: Close stream
01 01: Open stream
00 01: Open stream
01 02: Open stream
00 02: Open stream
00 03: Open stream
01 01: Close stream

```

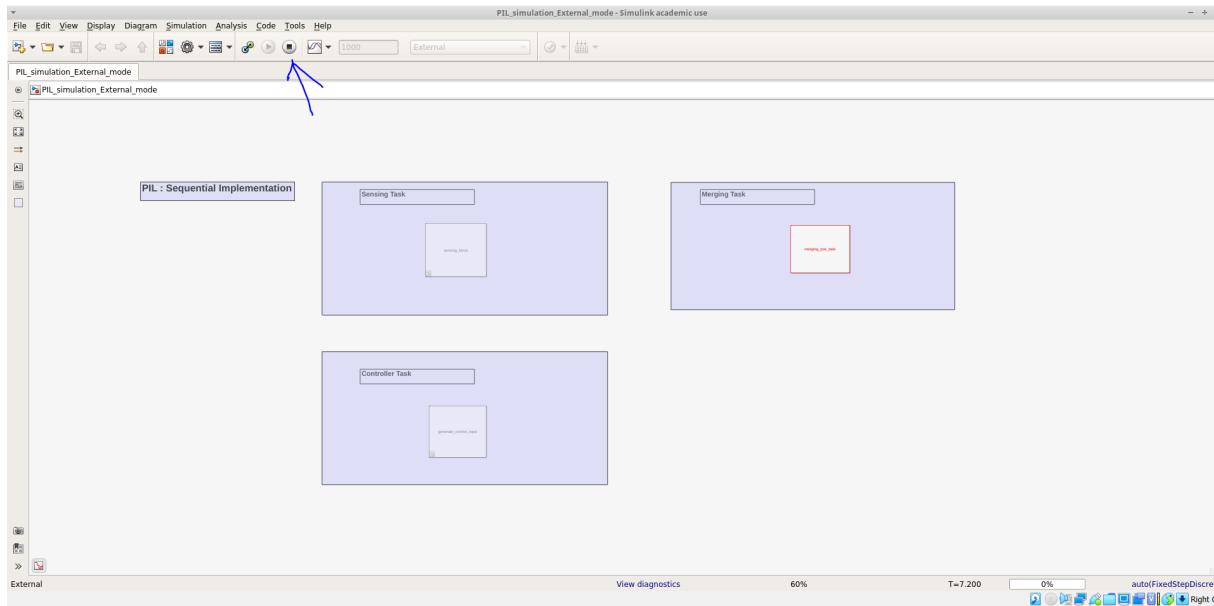
- click on the build button, as shown by the arrow in the below screenshot.



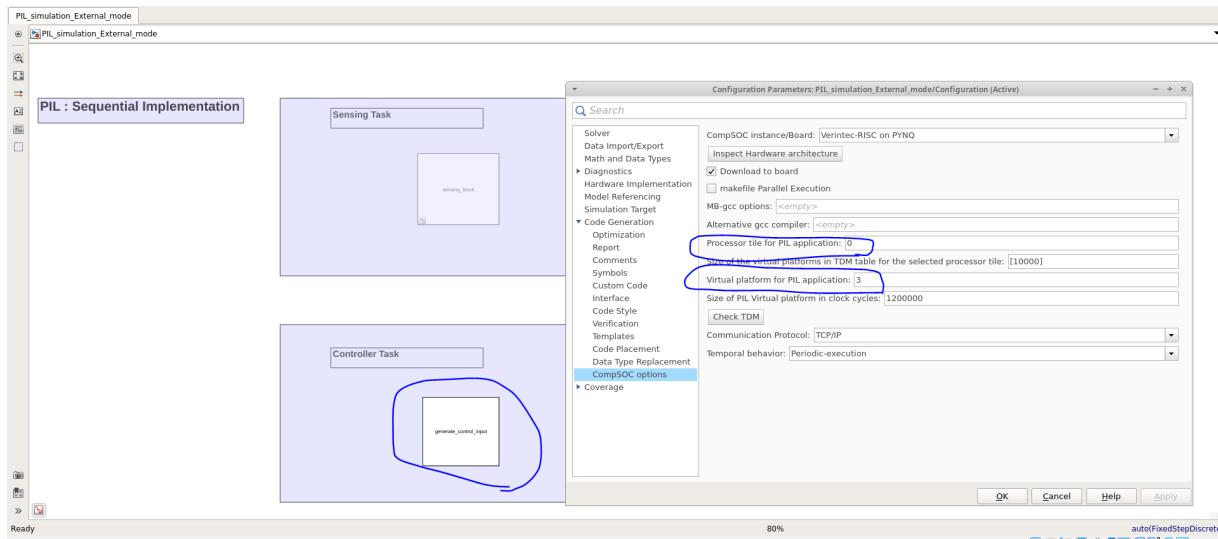
- if the build finishes successfully, ready status is available in the model. After that, run the task in external mode by clicking on the connect to target button. Check the following screenshot for reference.



- after the above step, the merging task connects to the target and starts running on the CompSOC; let it starts and runs for some time, then stop the running of the task by clicking on the stop button. Check the arrow in the screenshot. No need to run the task for a long time.



- move to the next task and comment all blocks except the one you want to run in external mode. Check the following screenshot; we have considered the controller task after finishing the merging task. Only the controller task is uncommented. Other blocks kept commenting.
- check the configuration setting. We are running the controller task on processor 0 and partition 3. So, the following setting should be considered,
  - Processor tile for PIL application : 0
  - Virtual platform for PIL application: 3

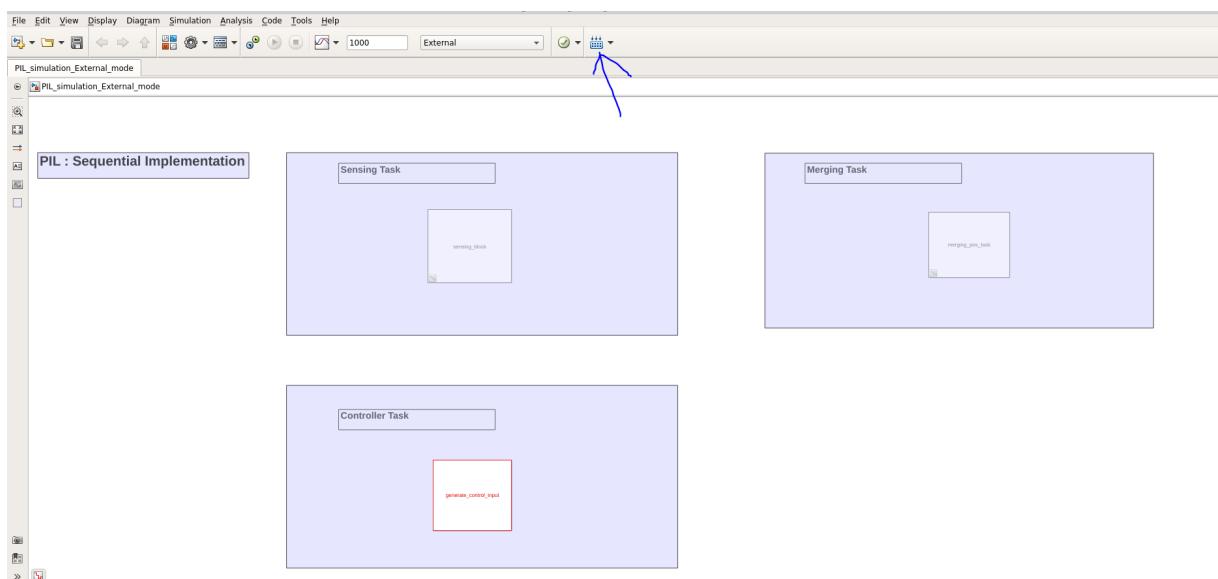


- in the same two terminal windows opened for the previous task, you can use the following highlighted commands from the screenshot.

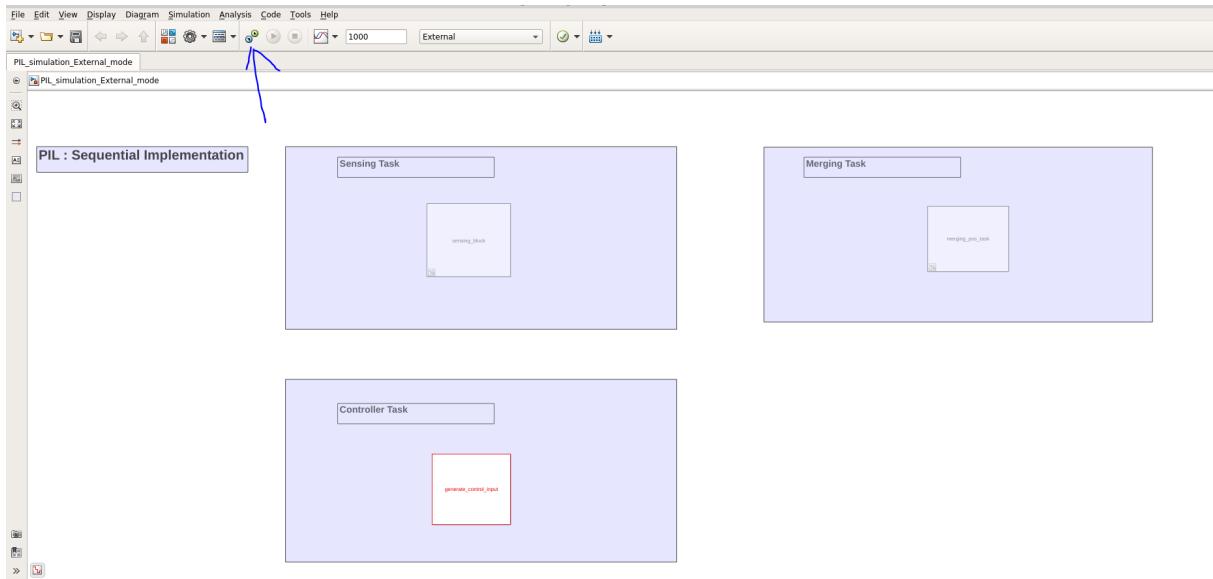
- Terminal 1 :
  - cd tutorial
  - cd monitoring-tools/
  - sudo ./channel Cheap\_Bridge 0 3 9878
- Terminal 2 :
  - cd tutorial
  - ./readout.sh

The screenshot shows two terminal windows side-by-side. The left terminal window has the title "computation@computation-virtual-machine:~". It displays the command "student@pato-board:~/tutorial/monitoring-tools\$ sudo ./channel Cheap\_Bridge 1 1 9878" followed by several lines of log output related to socket creation, binding, listening, and accepting connections. The right terminal window also has the title "computation@computation-virtual-machine:~". It displays the command "student@pato-board:~/tutorial/monitoring-tools\$ sudo ./channel Cheap\_Bridge 0 2 9878" followed by similar log output. Both terminals show a series of stream operations (open, close, read, write) from the readout script.

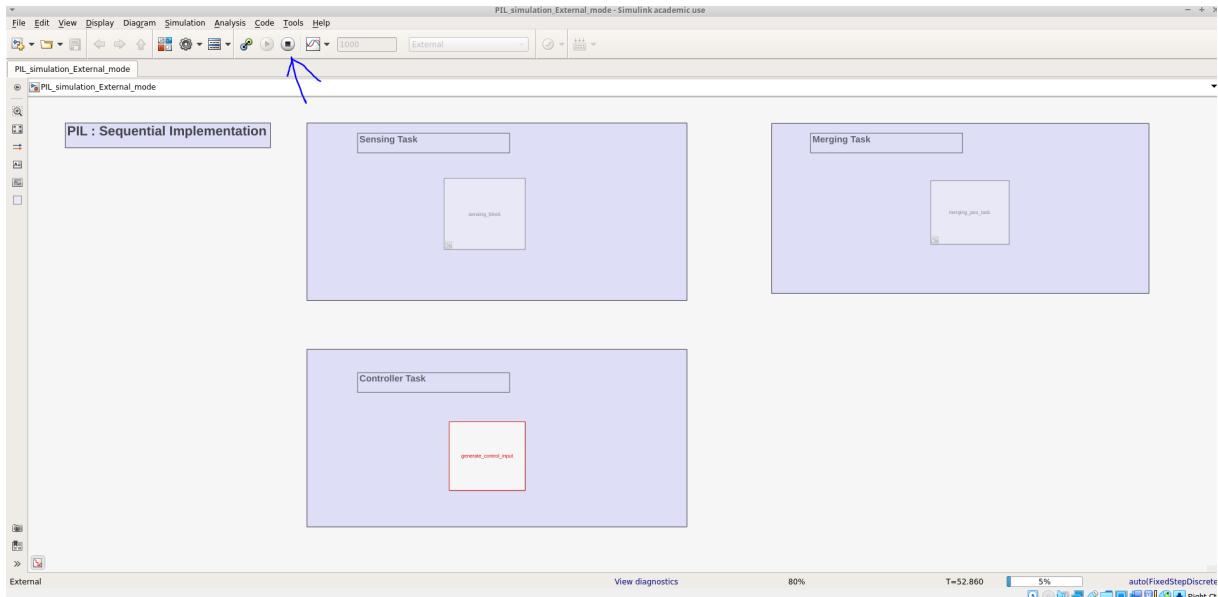
- click on the build button, as shown by the arrow in the below screenshot.



- if the build finishes successfully, ready status is available in the model. After that, run the task in external mode by clicking the connect to target button. Check the following screenshot for reference.



- after the above step, the controller task connects to the target and starts running on the CompSOC; let it starts and runs for some time, then stop the running of the task by clicking on the stop button; check the arrow in the screenshot. No need to run the task for a long time.



- after completing all the steps, run the following highlighted command in the same terminal window opened for the previous steps. This step is required for running the final PIL simulation, which will be run from **PIL\_simulation\_monitoring\_app.slx** Simulink model.

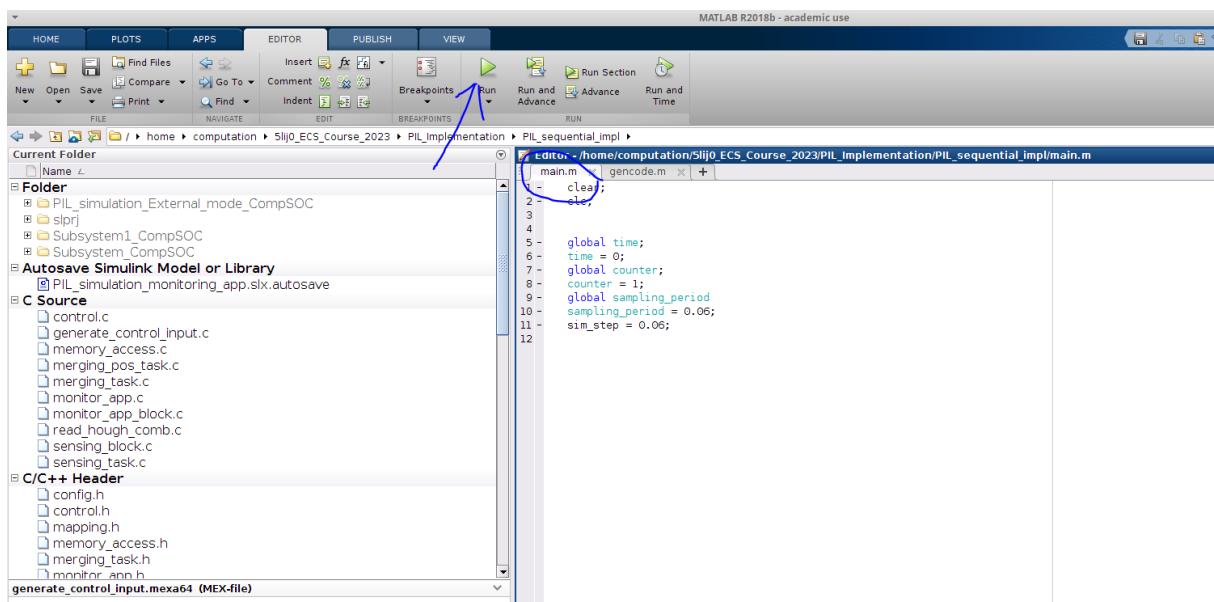
- Terminal 1 :
  - cd tutorial
  - cd monitoring-tools/
  - sudo ./channel Cheap\_Bridge 0 1 9878
- Terminal 2 :
  - cd tutorial
  - ./readout.sh

```

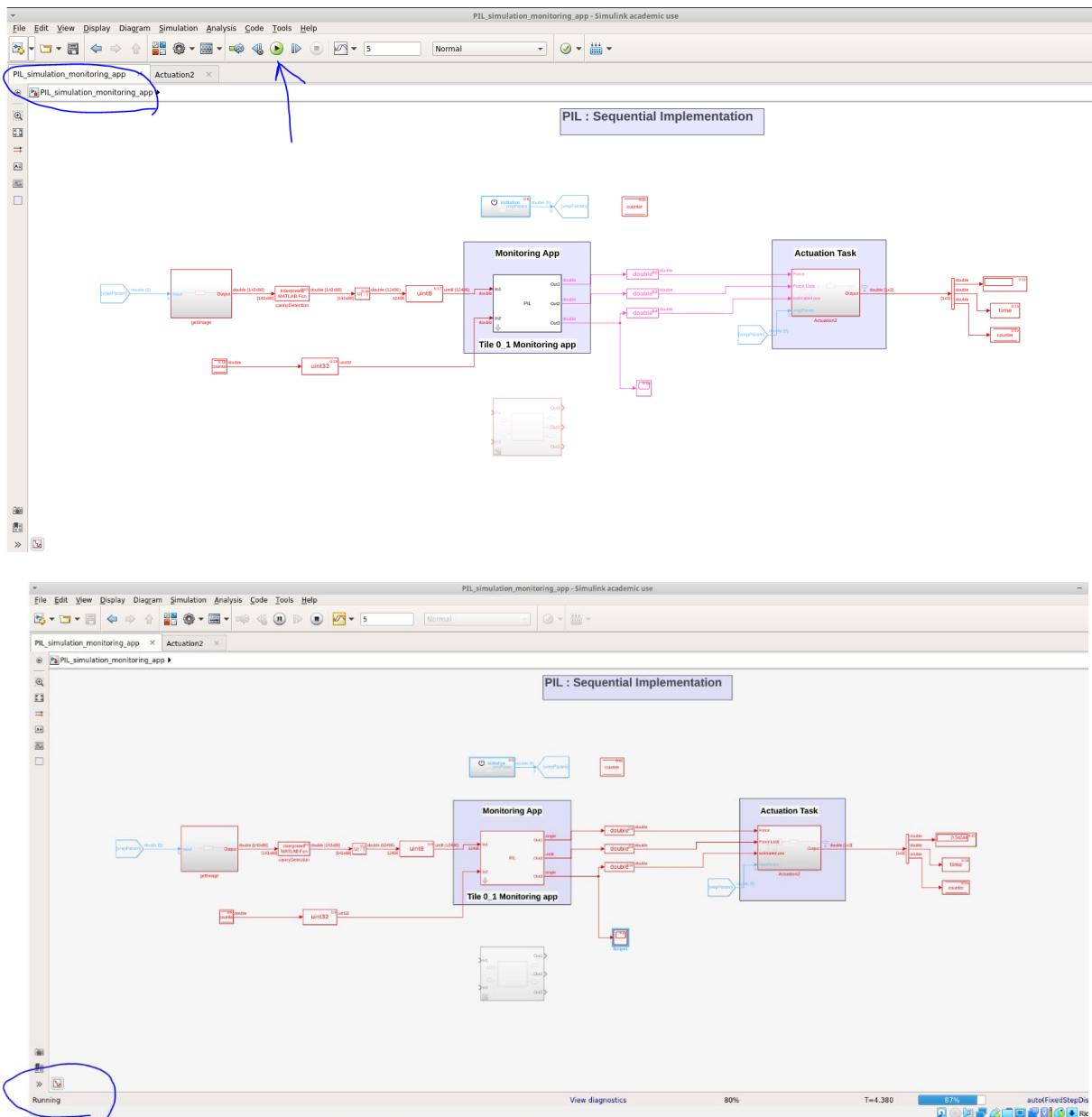
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 1 1 9878
+Socket created
+bind on port: 9878 done
+ Listen done
-Waiting for incoming connections.
-Accepting socket
+Connection accepted
-Connection closed
0.000011
0.000029
-Waiting for incoming connections.
-
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 0 2 9878
+Socket created
+bind on port: 9878 done
+ Listen done
-Waiting for incoming connections.
-Accepting socket
+Connection accepted
-Connection closed
0.000010
0.000027
-Waiting for incoming connections.
-Accepting socket
+Connection accepted
-Connection closed
0.000011
0.000029
-Waiting for incoming connections.
-
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 0 1 9878
+Socket created
+bind on port: 9878 done
+ Listen done
-Waiting for incoming connections.
-Accepting socket
+Connection accepted
-Connection closed
0.000010
0.000025
-Waiting for incoming connections.
-

```

- next step is to run the main.m file and check the CoppeliaSim software; the mass-damper scene should be opened and ensure it is not running automatically; if it is running, stop it.



- open the **PIL\_simulation\_monitoring\_app.slx** Simulink model, and click on the Run button. This will start running the simulation. It will take some time to simulate as it initializes all the required settings for running the simulation. So wait until you see the running status. Check the following screenshots for reference.
- check the current position returned from the monitoring app using the scope linked to the monitoring app PIL block. Also, visualize the output in CoppeliaSim.

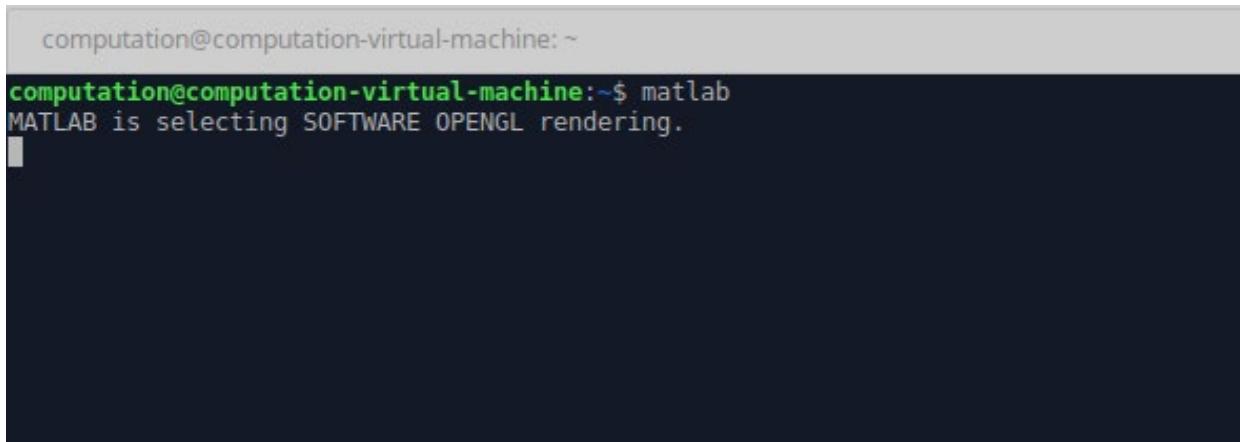


## Section II: PiL implementation for parallel configuration

- open MATLAB: To open MATLAB in the IMACS virtual machine, run the following command in the new terminal.

**Note:** If MATLAB is already opened, then no need to reopen it.

```
$ matlab
```

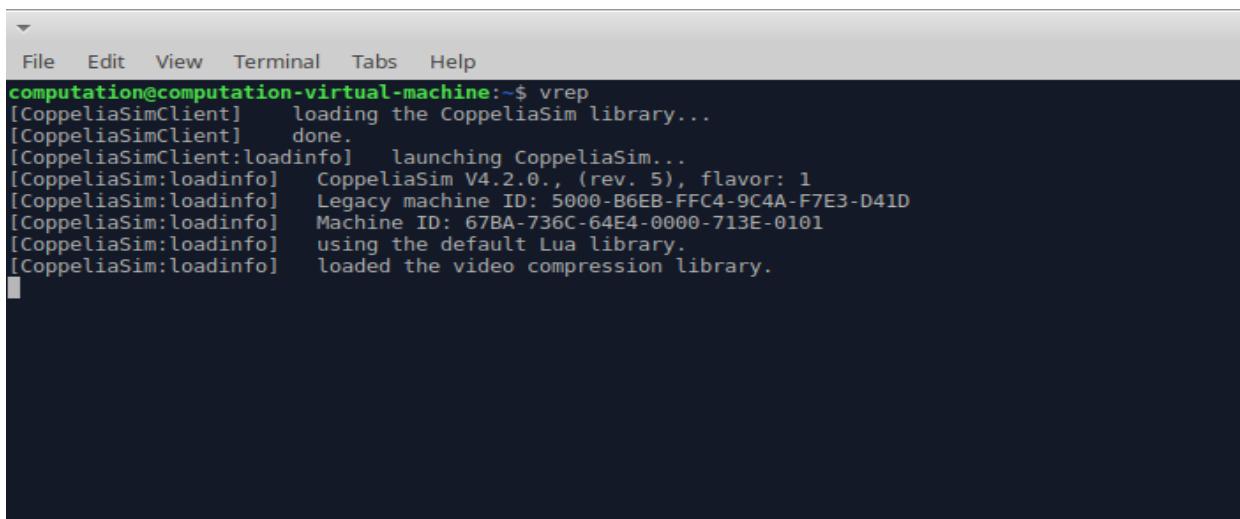


A terminal window titled "computation@computation-virtual-machine: ~". It shows the command \$ matlab being run, followed by the message "MATLAB is selecting SOFTWARE OPENGL rendering." The window has a light gray header bar and a dark gray body.

- open CoppeliaSim: to open the CoppeliaSim, run the following command in a new terminal.

**Note:** If CoppeliaSim is already opened, then no need to reopen it.

```
$ vrep
```



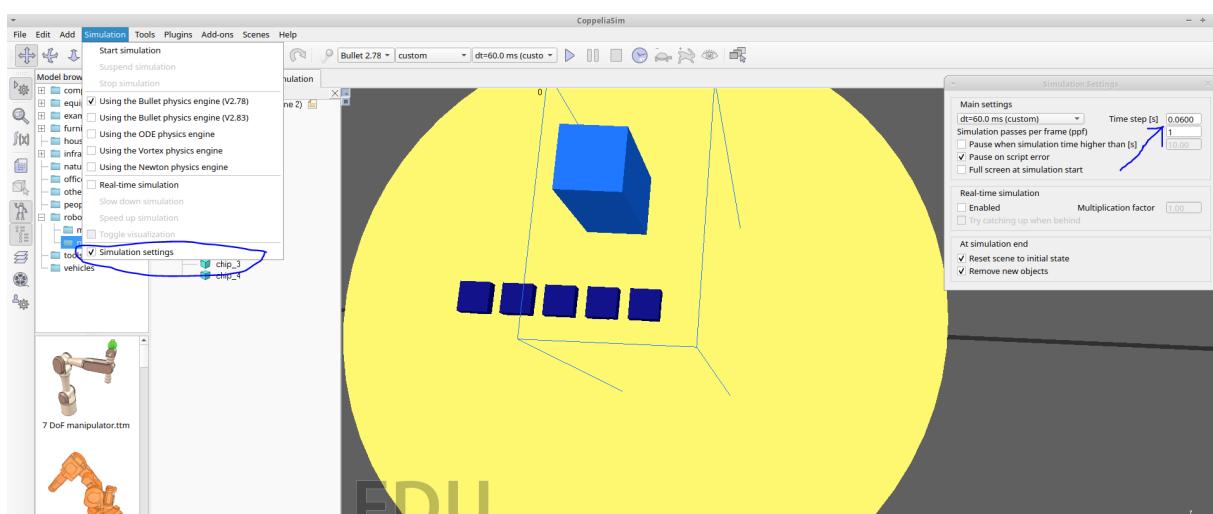
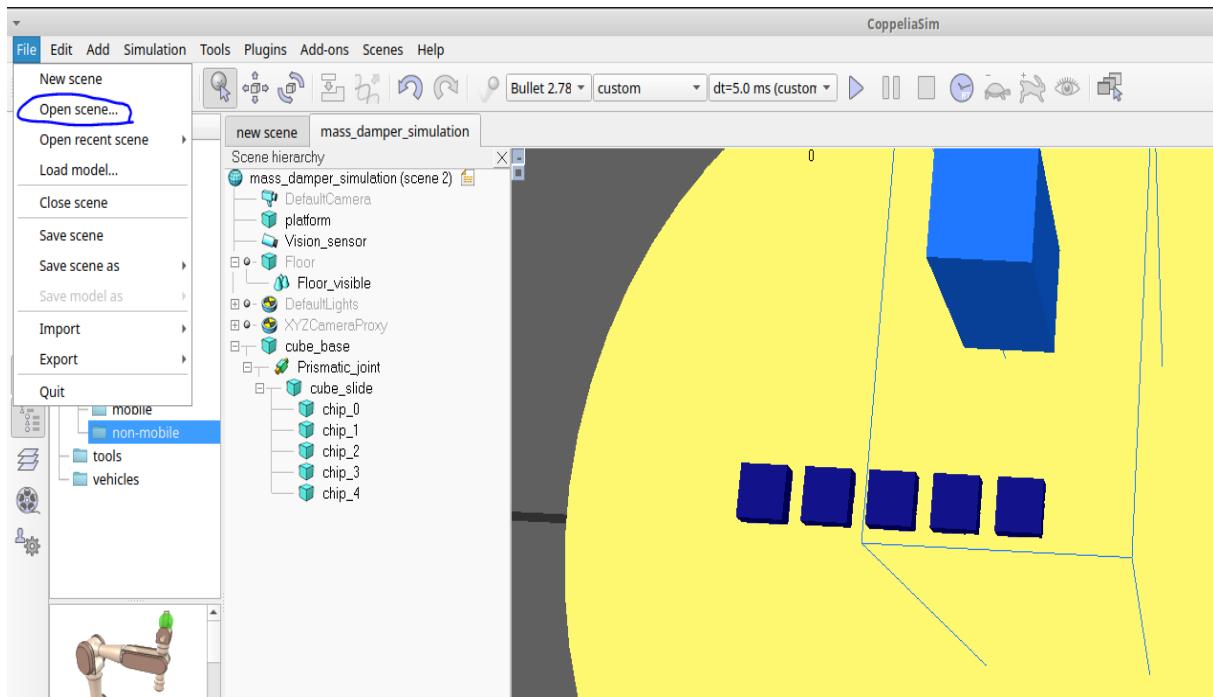
A terminal window titled "computation@computation-virtual-machine: ~\$ vrep". The menu bar includes File, Edit, View, Terminal, Tabs, and Help. The window shows the command \$ vrep being run, followed by several log messages from CoppeliaSimClient and CoppeliaSimClient:loadinfo indicating the loading of the library, launching of the simulation, and providing machine IDs. The window has a light gray header bar and a dark gray body.

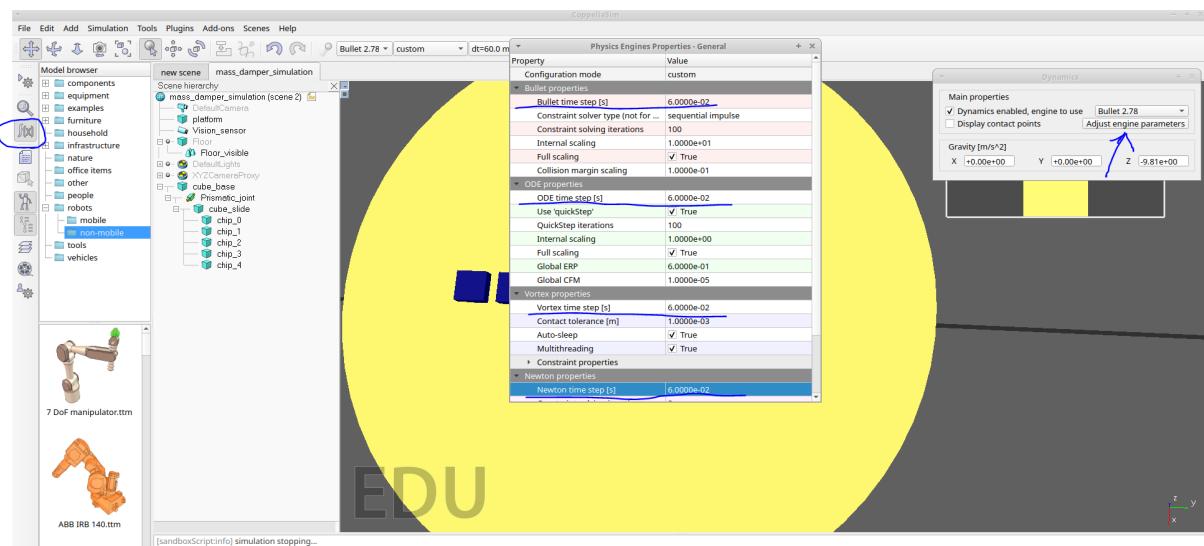
- open CoppeliaSim scene for the mass-damper system.

- go to File → Open scene → select the following directory  
/home/computation/5lij0\_ECS\_Course\_2023/PIL\_Implementation/PIL\_parallel\_i  
mpl

open **mass\_damper\_simulation.ttt**, check the following screenshot for reference.

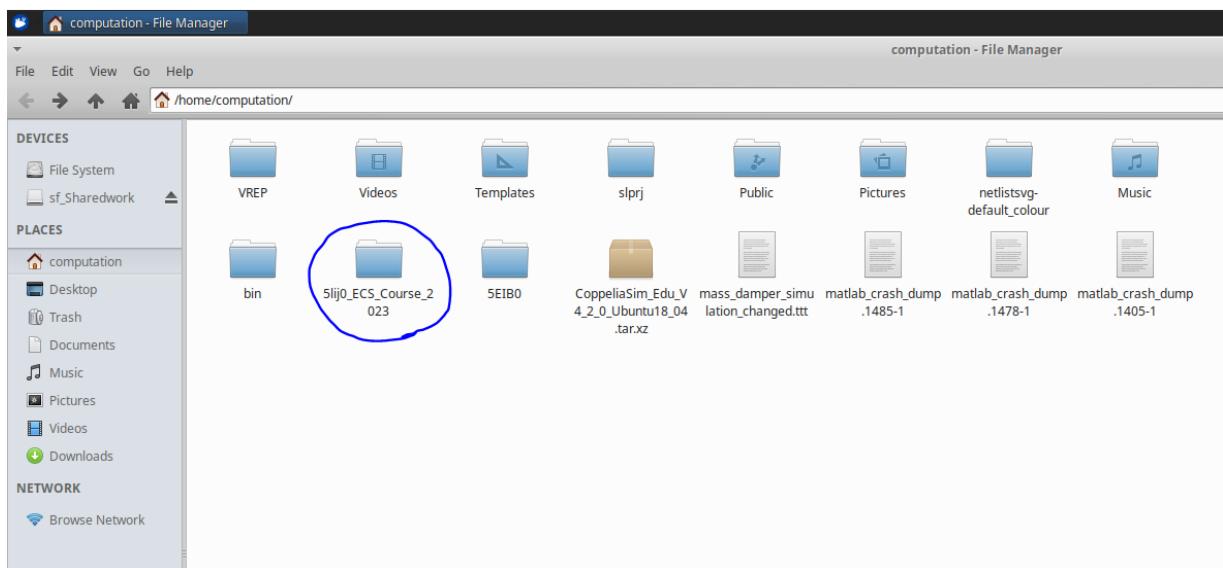
We have implemented the PIL of sequential configuration considering the 60ms sample period. The same sample period should be used in CoppeliaSim. In case of different sample periods, students can make the changes in the CoppeliaSim as explained below. For example, the following screenshot shows how to change the time step in the CoppeliaSim scene.

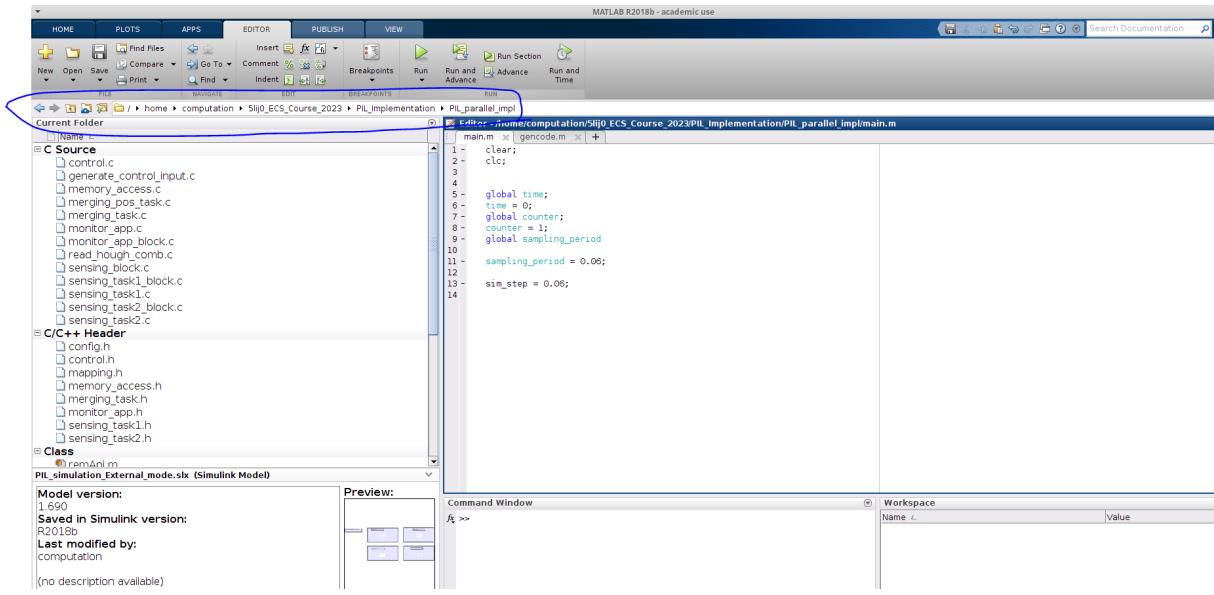




- open the folder for PiL parallel implementation.
  - go to the following folder path and open in MATLAB
 

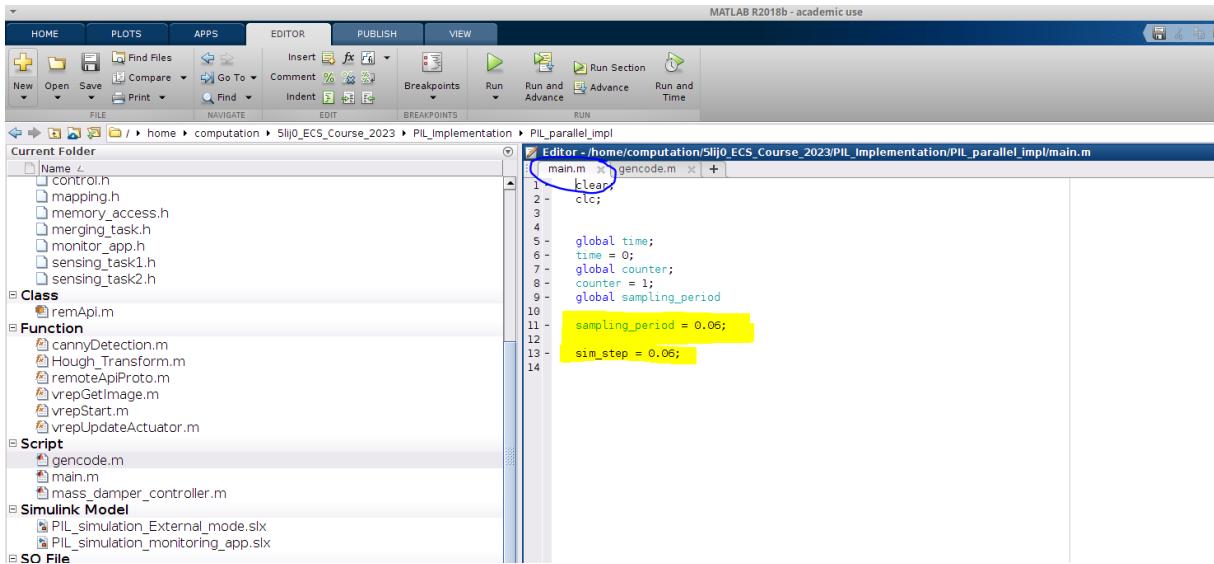
computation → 5lij0\_ECS\_Course\_2023 → PIL\_Implementation → PIL\_parallel\_impl





- open the main.m file and make changes for the highlighted parameters, as shown in the following screenshot.

Note: In this screenshot, we have explained for the 60ms sample period, students can use different sample periods according to the implementation design and make the changes for the following parameters. Once the changes are done, run the main.m script. This file includes the initialization, which is required for the PiL simulation.



- open the gencode.m file and run the script. This file uses legacy code tool functionality, integrating the existing C or C++ functions into MATLAB Simulink. Once this gencode.m runs, it creates the S-function block inside the Simulink model.

```

1 %> %% Monitor app
2 def_monitor = legacy_code('initialize');
3 def_monitor.SourceFiles = {'monitor_app.c', 'memory_access.c'};
4 def_monitor.HeaderFiles = {'monitor_app.h', 'memory_access.h', 'config.h', 'mapping.h'}
5 def_monitor.SFunctionName = 'monitor_app_block';
6 def_monitor.IncPaths = {'../../../../CompSOC_ec_target/files'};
7 def_monitor.OutputFcnSpec = 'void monitor(uint8 ui[12496], single y1[1], uint8 y2[1] )';
8 legacy_code('sfcn_cmex_generate', def_monitor);
9 legacy_code('sfcn_tlc_generate', def_monitor);
10 legacy_code('compile', def_monitor);
11 legacy_code('slblock_generate', def_monitor);
12
13 %% Sensing task1
14 def_sensing1 = legacy_code('initialize');
15 def_sensing1.SourceFiles = {'sensing_task1.c', 'memory_access.c'};
16 def_sensing1.HeaderFiles = {'sensing_task1.h', 'memory_access.h', 'config.h', 'mapping.h'}
17 def_sensing1.SFunctionName = 'sensing_task1_block';
18 def_sensing1.IncPaths = {'../../../../CompSOC_ec_target/files'};
19 def_sensing1.OutputFcnSpec = 'void hough_transform()';
20 legacy_code('sfcn_cmex_generate', def_sensing1);
21 legacy_code('sfcn_tlc_generate', def_sensing1);
22 legacy_code('compile', def_sensing1);
23 legacy_code('slblock_generate', def_sensing1);
24
25 %% Sensing task2
26 def_sensing2 = legacy_code('initialize');
27 def_sensing2.SourceFiles = {'sensing_task2.c', 'memory_access.c'};
28 def_sensing2.HeaderFiles = {'sensing_task2.h', 'memory_access.h', 'config.h', 'mapping.h'}
29 def_sensing2.SFunctionName = 'sensing_task2_block';
30 def_sensing2.IncPaths = {'../../../../CompSOC_ec_target/files'};
31 def_sensing2.OutputFcnSpec = 'void hough_transform()';
32 legacy_code('sfcn_cmex_generate', def_sensing2);
33 legacy_code('sfcn_tlc_generate', def_sensing2);
34 legacy_code('compile', def_sensing2);
35 legacy_code('slblock_generate', def_sensing2);
36

```

- open the **vrepUpdateActuator.m** file. Make the necessary changes in the file, as shown below in the screenshot. This file is required for the actuation.

```

1 function output = vrepUpdateActuator(input)
2
3 % vrepParam = [
4 %   1 clientId      -> vrepParam(1)
5 %   2 cam           -> vrepParam(2)
6 %   3 motor          -> vrepParam(3)
7 %   4 chip           -> vrepParam(4)
8 %   5 base           -> vrepParam(5) ]
9
10 vrepParam = input(4:end);
11 force_lock = input(2);
12 force = input(1);
13
14 damping_constant = 1;
15 sim_step = 0.06;
16
17 global vrep;
18 global prev_sign;
19 global total_force;
20 global time;
21 global counter;
22 global prev_pos;
23 global sampling_period;
24
25 % Initial Data
26 if (time == 0)
27     [returnCode, chip_pos] = vrep.simxGetObjectPosition(vrepPar
28     chip_pos = double((chip_pos(2)-1.4 + 0.025/2)) + 0.001;
```

- open the **config.h** file. Make the necessary changes in the file, as shown below in the screenshot. This file contains all the required parameters used in the PiL simulation.

The screenshot shows a software interface with a file tree on the left and a code editor on the right. The file tree under 'C Source' includes files like control.c, generate\_control\_input.c, memory\_access.c, merging\_pos\_task.c, merging\_task.c, monitor\_app.c, monitor\_app\_block.c, read\_hough\_comb.c, sensing\_block.c, sensing\_task1\_block.c, sensing\_task1.c, sensing\_task2\_block.c, and sensing\_task2.c. Under 'C/C++ Header', it lists config.h, control.h, mapping.h, memory\_access.h, merging\_task.h, monitor\_app.h, sensing\_task1.h, and sensing\_task2.h. The 'Class' section contains remAnim.h. The code editor displays config.h with the following content:

```

1 #ifndef CONFIG_H
2 #define CONFIG_H
3
4 // Sensing
5 #define WAITING_TIME ((uint32_t)200000) // Cycles ~ 5 ms
6 #define IMG_WIDTH (88) // Pixels
7 #define HALF_WIDTH (44) // Pixels
8 #define IMG_HEIGHT (142) // Pixels
9 #define IMG_SIZE (12496) // Pixels
10 #define HALF_IMG (6248)
11 #define HOUGH_THRESHOLD (25)
12 #define PADDING_SIZE_THETA (3) // Pixels
13 #define PADDING_SIZE_RHO (3) // Pixels
14 #define MAX_PEAK_NUMBER (2)
15 #define DESIRED_POS (44) // Pixels
16 #define PIXEL_RATIO (0.001) // 1 millimeter
17 #define THETA_neg_5_pos_5
18
19
20 // Controller
21 #define DAMPING_CONST ((float)1.0)
22 #define SAMPLING_TIME ((float)0.06)
23 #define INV_SAMPLING_TIME ((float)16.66)
24 #define K_P ((float)0.06)
25 #define K_D ((float)0.01)
26 #define K_I ((float)0.00)
27 #define PI (3.1416)
28#endif

```

- for tuning the PID controller gains, students can change the gains in the **config.h** file, check the below screenshot.

This screenshot is identical to the one above, except the code editor highlights a different set of lines related to the PID controller gains (K\_P, K\_D, K\_I). The highlighted lines are:

```

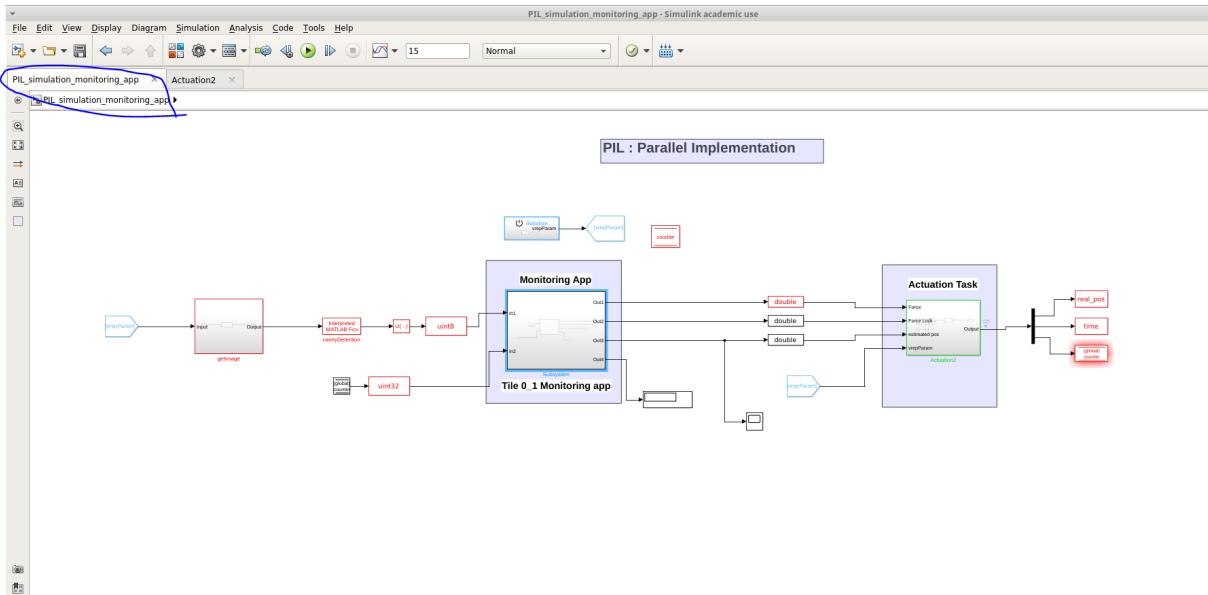
24 #define K_P ((float)0.06)
25 #define K_D ((float)0.01)
26 #define K_I ((float)0.00)

```

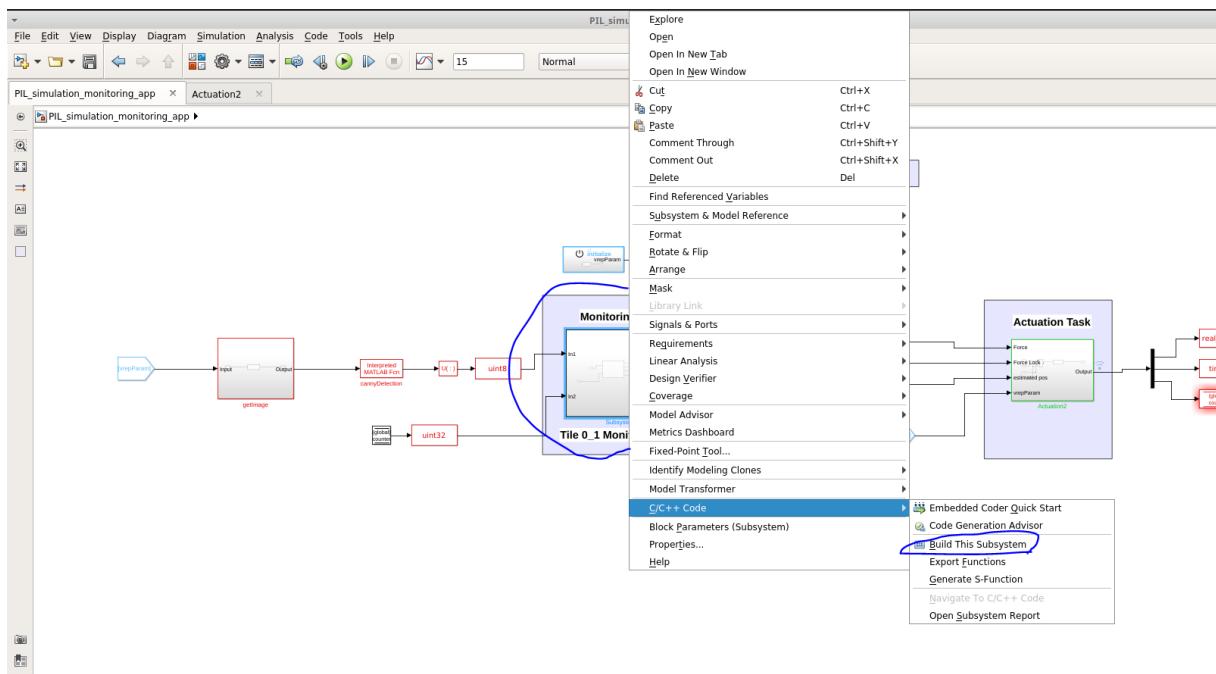
- once the steps mentioned above are done, open the PiL simulation file named **PIL\_simulation\_monitoring\_app.slx**

Students can find this file in the following directory:

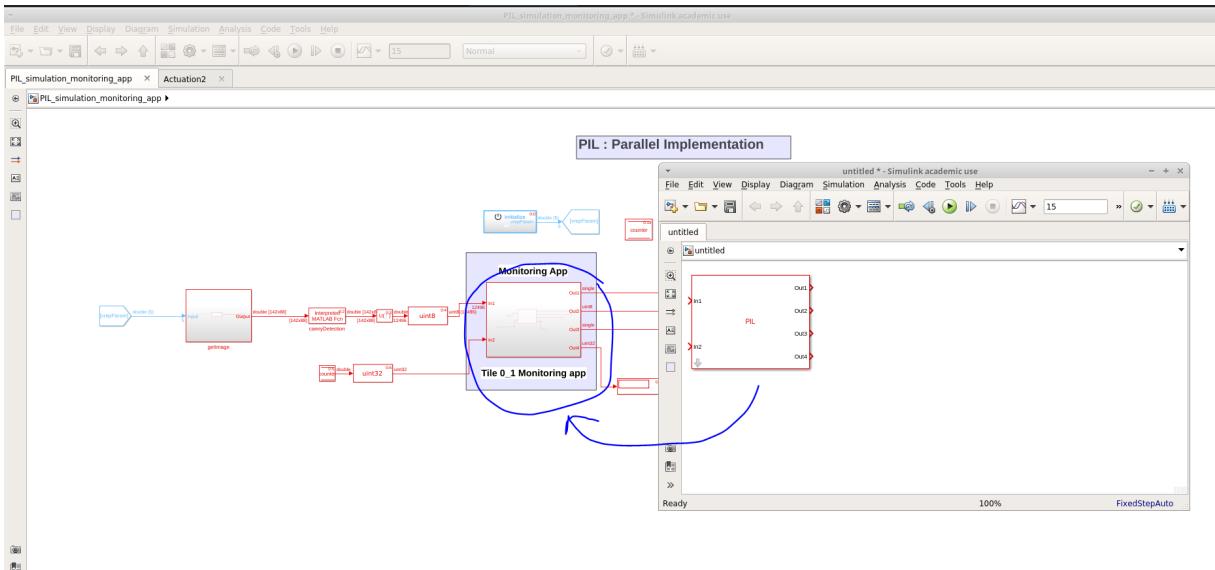
computation → 5lij0\_ECS\_Course\_2023 → PIL\_Implementation → PIL\_parallel\_impl



- right-click on the block, select the C/C++ code option, and click on build this subsystem. Check the following screenshot for reference.



- once, the build process is completed successfully; it will generate the PIL block; check the following screenshot:



- after completing the above steps, check the vep\_config.txt file. In this example, we have considered the 60ms sample period. Therefore, in the vep\_config.txt, scheduling is given according to the 60ms sample period.
- we have provided the vep\_config.txt file in the following folder  
`/home/computation/5lij0_ECS_Course_2023/PIL_Implementation/PIL_parallel_impl`
- students need to copy the contents from the above vep\_config file into the main **vep-config.txt** file, which is inside the following folder path :  
 or the student can enter the TDM scheduling calculation directly in the vep-config.txt file, which is stored in the following location.

`/home/computation/CompSOC_ec_target/CompSOC_ec/+CompSOC_ec/vep-config.txt`

- the contents in the vep\_config file should be as follows:

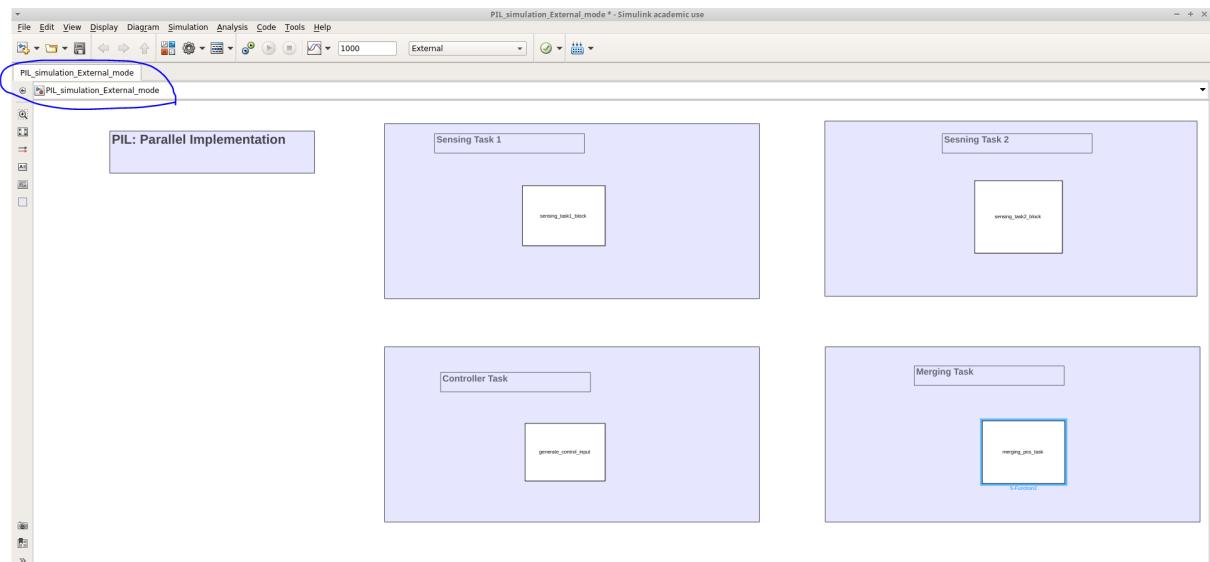
```
File Edit Search View Document Help

##  
## 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 64K 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 800000 cycles  
on tile 0 next slot is for partition 2 with 10000 cycles  
on tile 0 next slot is for partition 3 with 1577000 cycles  
#on tile 0 next slot is for partition 1 with 71000 cycles  
on tile 1 next slot is for partition 1 with 800000 cycles  
on tile 1 next slot is for partition 2 with 1589000 cycles  
on tile 2 next slot is for partition 1 with 800000 cycles  
on tile 2 next slot is for partition 2 with 1589000 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|
```

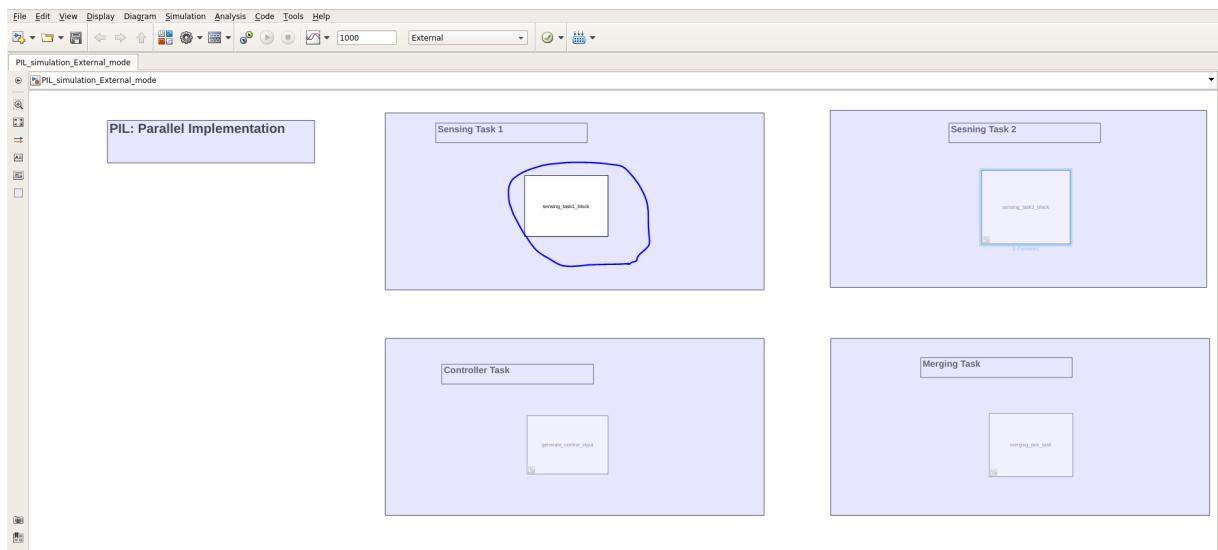
- open the PIL\_simulation\_External\_mode.slx. The folder path to this Simulink model is as follows:

/home/computation/5lij0\_ECS\_Course\_2023/PIL\_Implementation/PIL\_parallel\_impl

- PIL\_simulation\_External\_mode.slx file has four different tasks, i.e., sensing task 1, sensing task 2, merging, and controller, which will run in the external mode on CompSOC based on the TDM scheduling. To run each task independently in the external mode, follow the below steps and check the screenshot for reference:

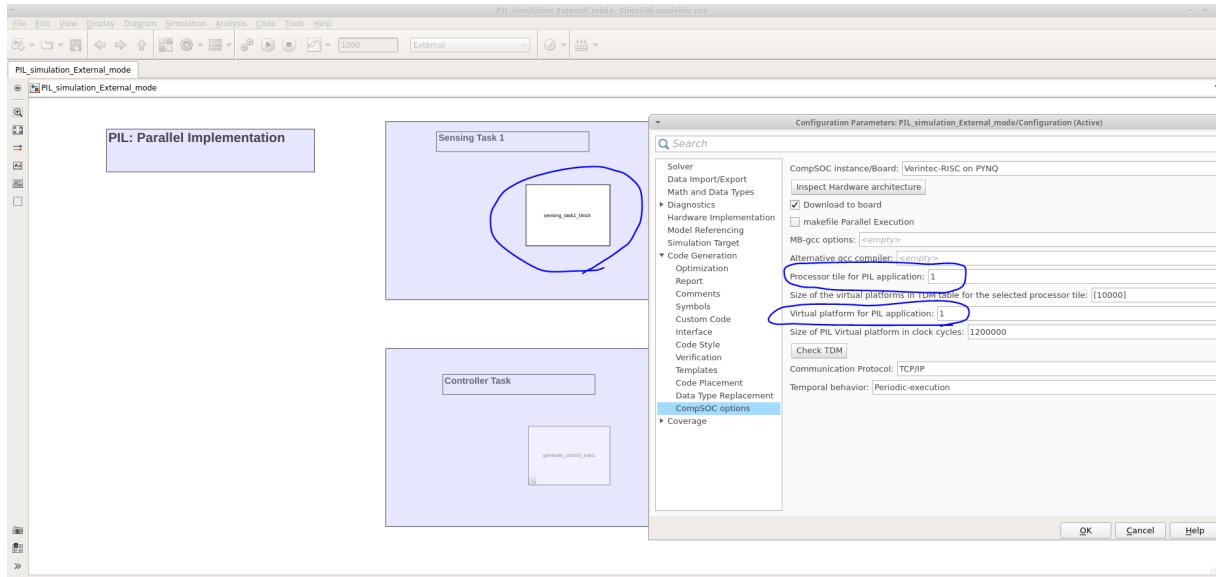


- to run sensing task 1 in the external mode, comment the other Simulink block and only uncomment the block which needs to be run. In the below screenshot, we consider the sensing task 1 block first, and other blocks kept commenting.



- open the configuration setting and check the required setting for the particular task. In this example, we run a sensing task 1 on processor 1 and partition 1. Therefore, consider the following changes :

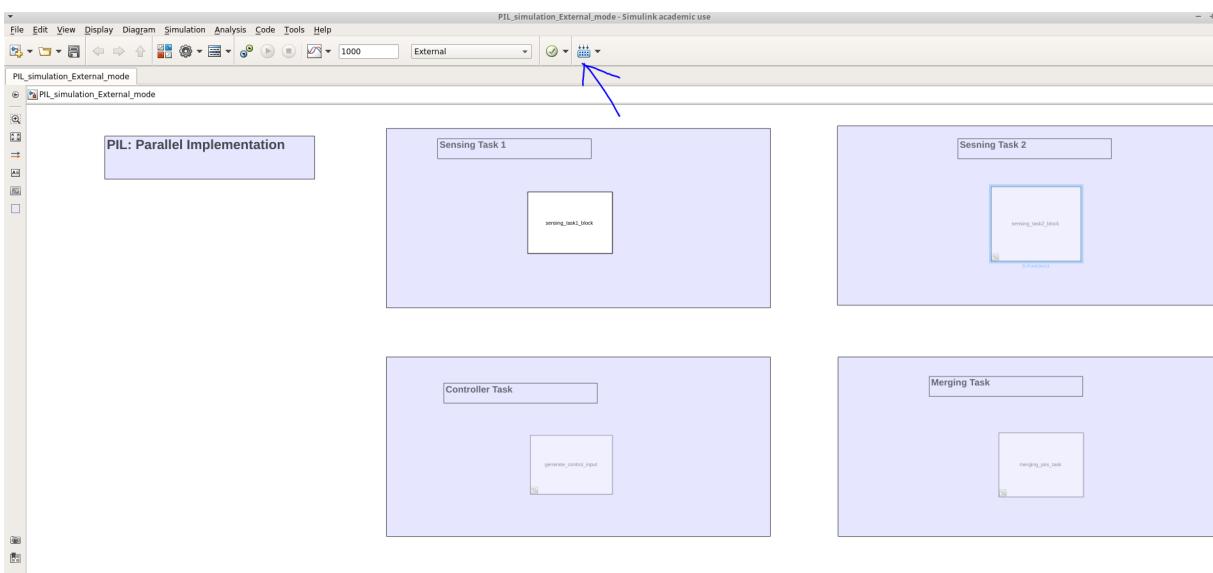
- Processor tile for PIL application: 1
- Virtual platform for PIL application: 1



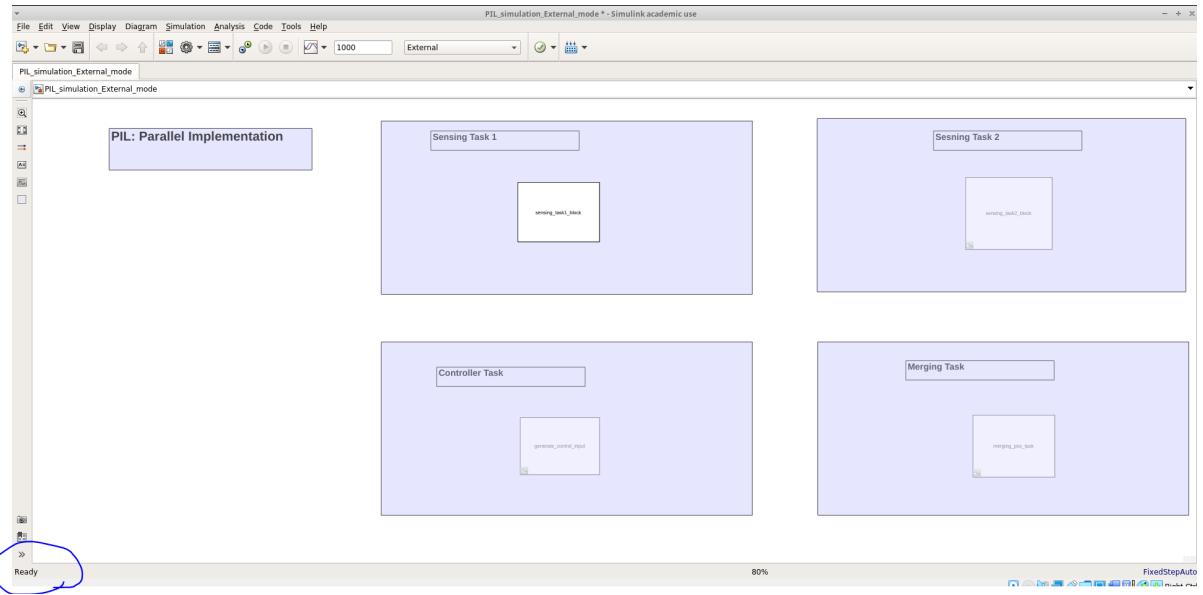
- open the two new terminal windows, connect the PYNQ board, and use the following command:
  - ensure that vep-config.txt is updated in the main CompSOC folder as per the required TDM scheduling. This step is essential before running the task in external mode on CompSOC.
  - vep-config file path:  
`/home/computation/CompSOC_ec_target/CompSOC_ec/+CompSOC_ec/vep-config.txt`
  - commands for running the sensing task on processor 1 and partition 1
    - Terminal 1 :
      - `cd tutorial`
      - `cd monitoring-tools/`
      - `sudo ./channel Cheap_Bridge 1 1 9878`
    - Terminal 2 :
      - `cd tutorial`
      - `./readout.sh`

The screenshot shows three terminal windows side-by-side. The left window is titled 'computation@computation-virtual-machine:~' and contains the command: 'student@pato-board:~/tutorial/monitoring/tools\$ sudo ./channel Cheap\_Bridge 1 1 9878'. The middle window is also titled 'computation@computation-virtual-machine:~' and shows a series of log entries: '00 03: Open stream', '00 02: close stream', '01 01: Open stream', '00 03: close stream', '01 02: Open stream', '00 02: Open stream', '00 03: Open stream', '00 02: close stream', '00 01: Open stream', '00 02: Open stream', '00 03: Open stream', '01 01: Open stream', '01 02: Open stream', '00 03: close stream', '01 01: close stream', '00 03: close stream', '00 02: Close stream'. The right window is titled 'computation@computation-virtual-machine:~' and is mostly blank.

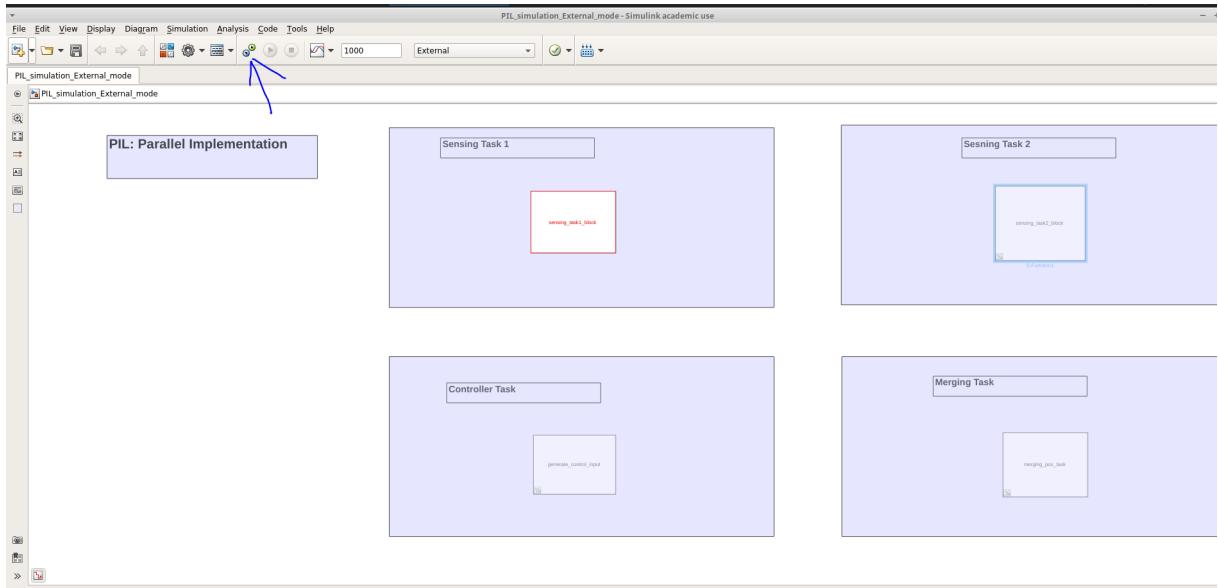
- click on the build button, as shown by the arrow in the below screenshot.



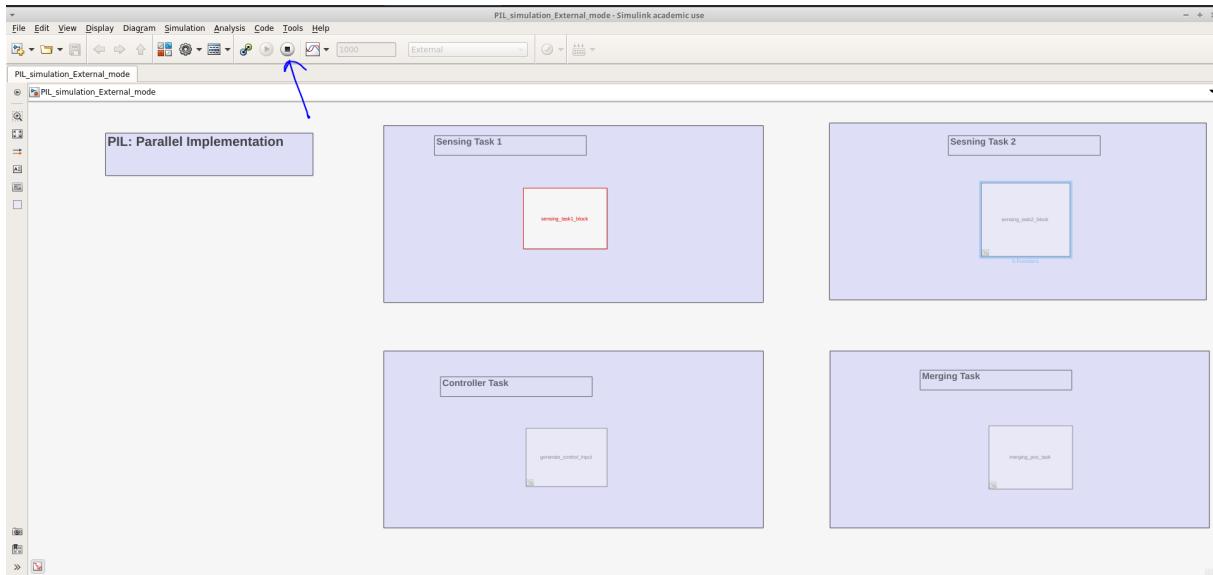
- after the build finishes, the Simulink model shows ready status; check the circle in the following screenshot.



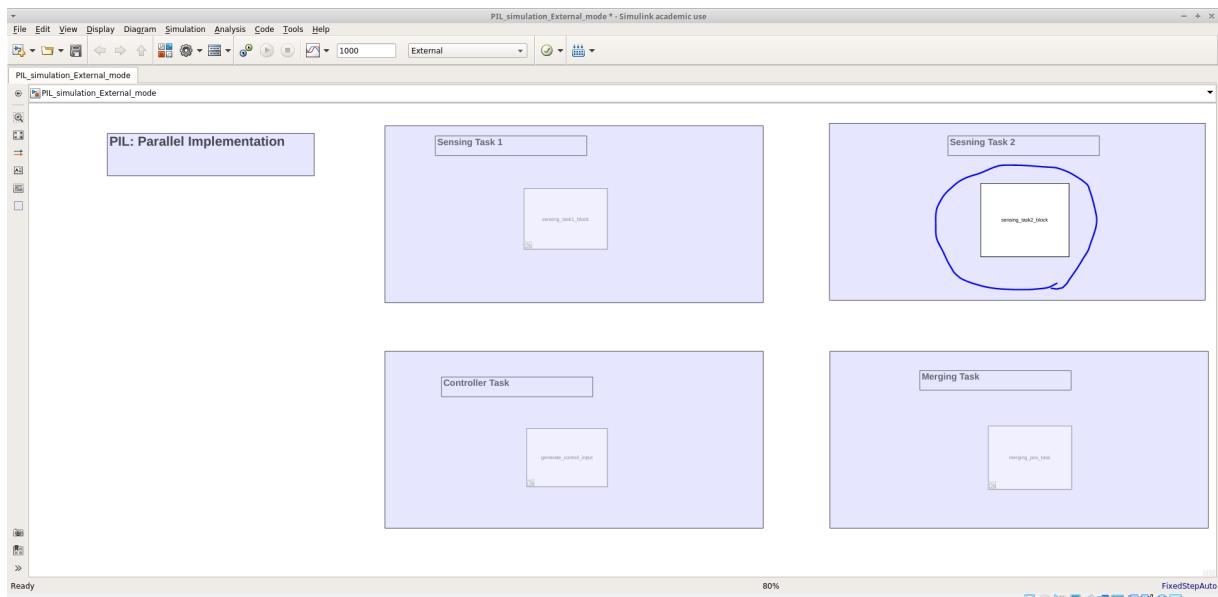
- if the build finishes successfully, ready status is available in the model. After that, run the task in external mode by clicking the connect to target button. Check the following screenshot for reference.



- after the above step, the sensing task connects to the target and starts running on the CompSOC; let it starts and runs for some time; check the highlighted circle in the below screenshot, then stop the running of the task by clicking on the stop button, check the arrow in the screenshot. No need to run the task for a long time.

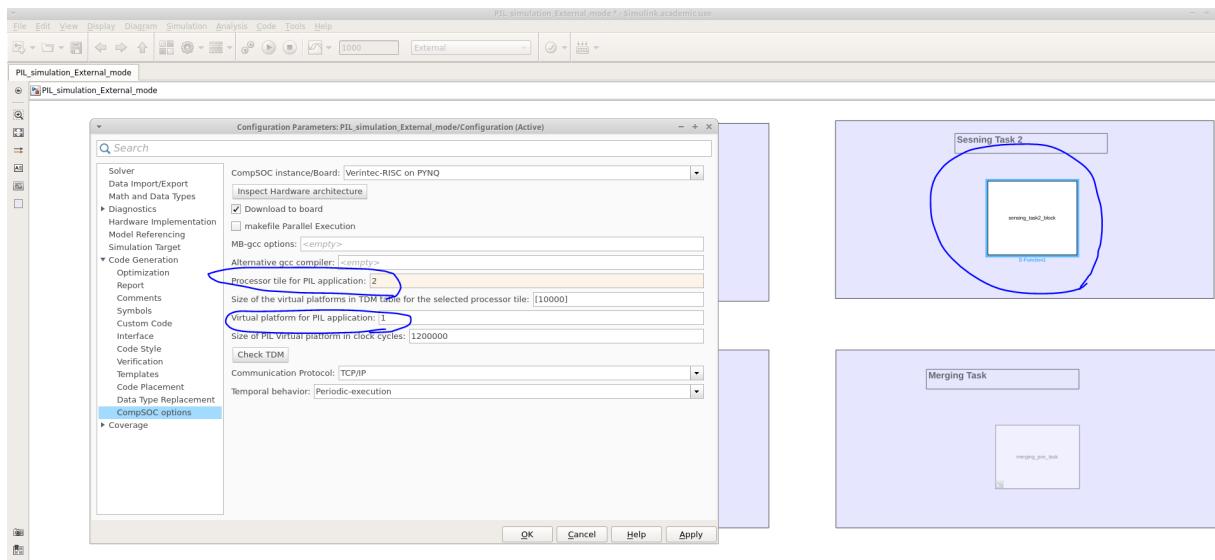


- move to the next task and comment all blocks except the one you want to run in external mode. Check the following screenshot; we have considered sensing task 2 after finishing sensing task 1. Only sensing task 2 is uncommented. Other blocks kept commenting.

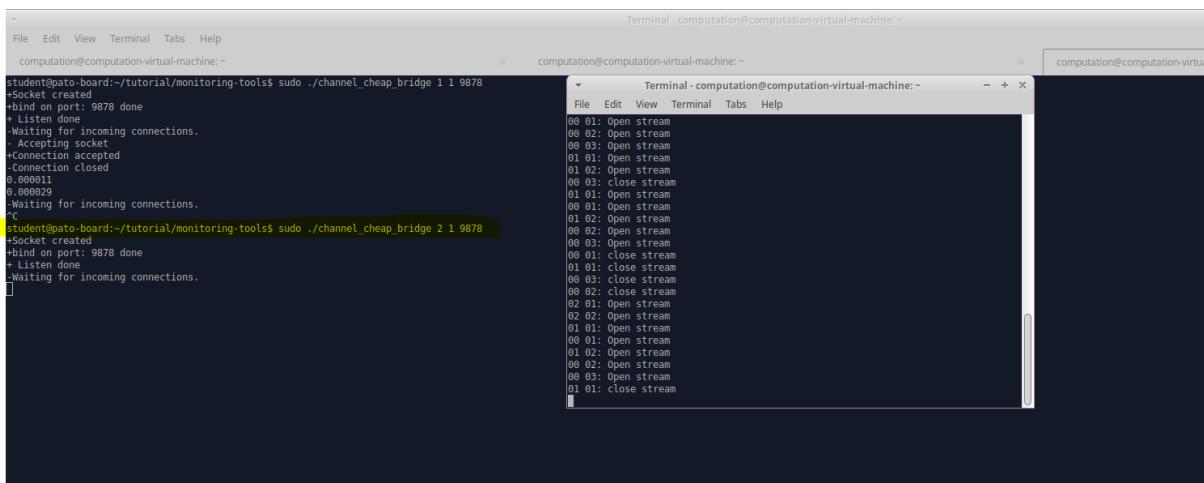


- open the configuration setting and check the required setting for the particular task. In this example, we run sensing task 2 on processor 2 and partition 1. Therefore, consider the following changes :

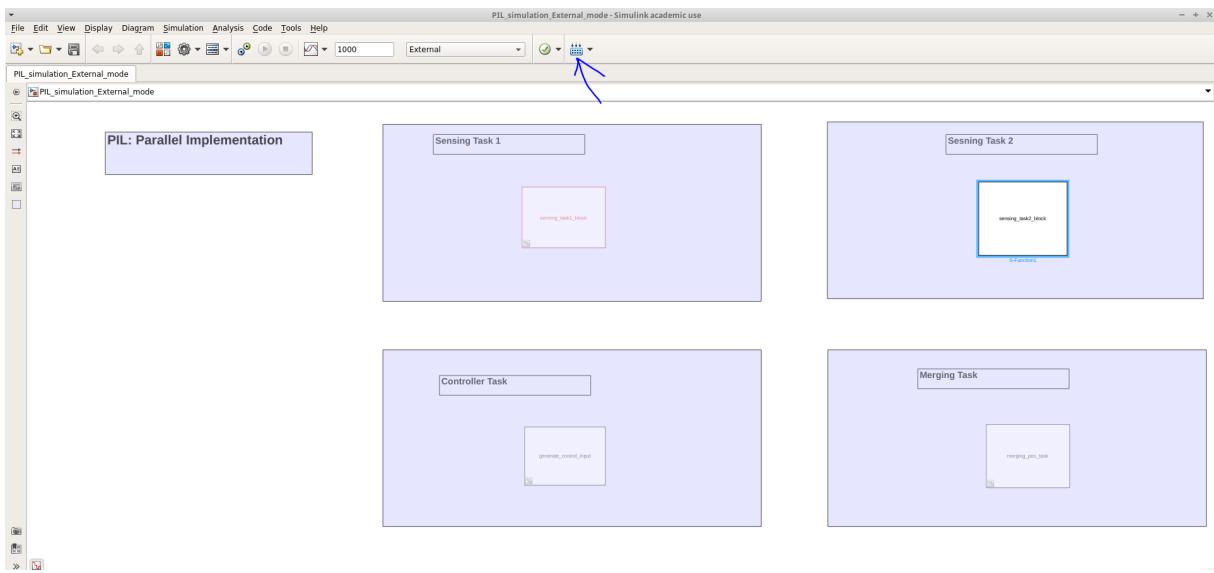
- Processor tile for PIL application: 2
- Virtual platform for PIL application: 1



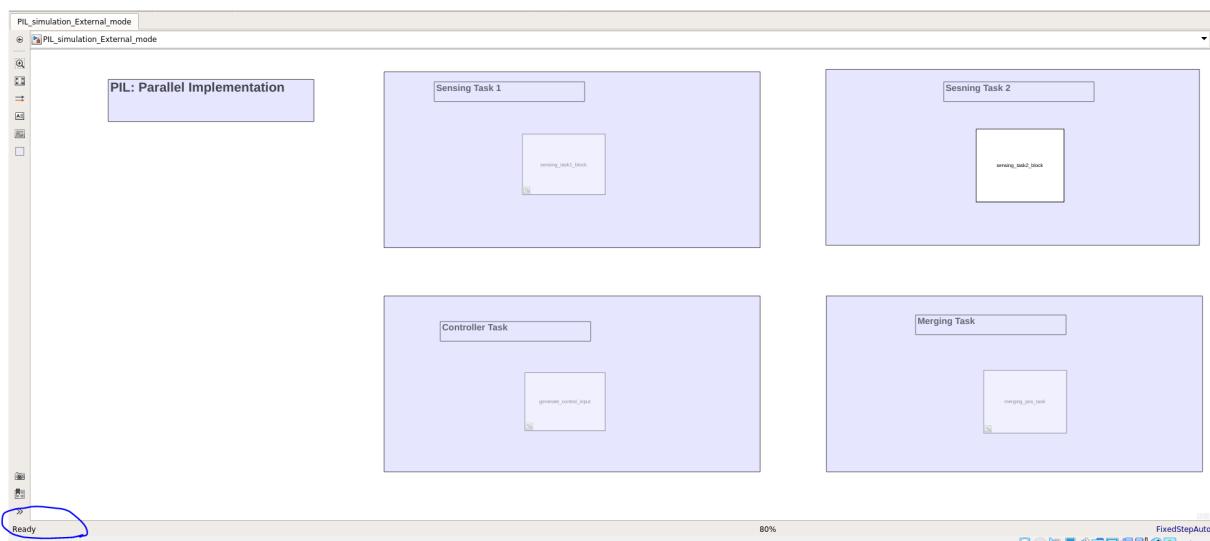
- open the two new terminal windows, connect the PYNQ board, and use the following command:
    - ensure that vep-config.txt is updated in the main CompSOC folder as per the required TDM scheduling. This step is essential before running the task in external mode on CompSOC.
    - vep-config file path:  
`/home/computation/CompSOC_ec_target/CompSOC_ec/+CompSOC_ec/vep-config.txt`
    - commands for running the sensing task on processor 2 and partition 1
      - Terminal 1 :
        - `cd tutorial`
        - `cd monitoring-tools/`
        - `sudo ./channel_cheap_bridge 2 1 9878`
      - Terminal 2 :
        - `cd tutorial`
        - `./readout.sh`



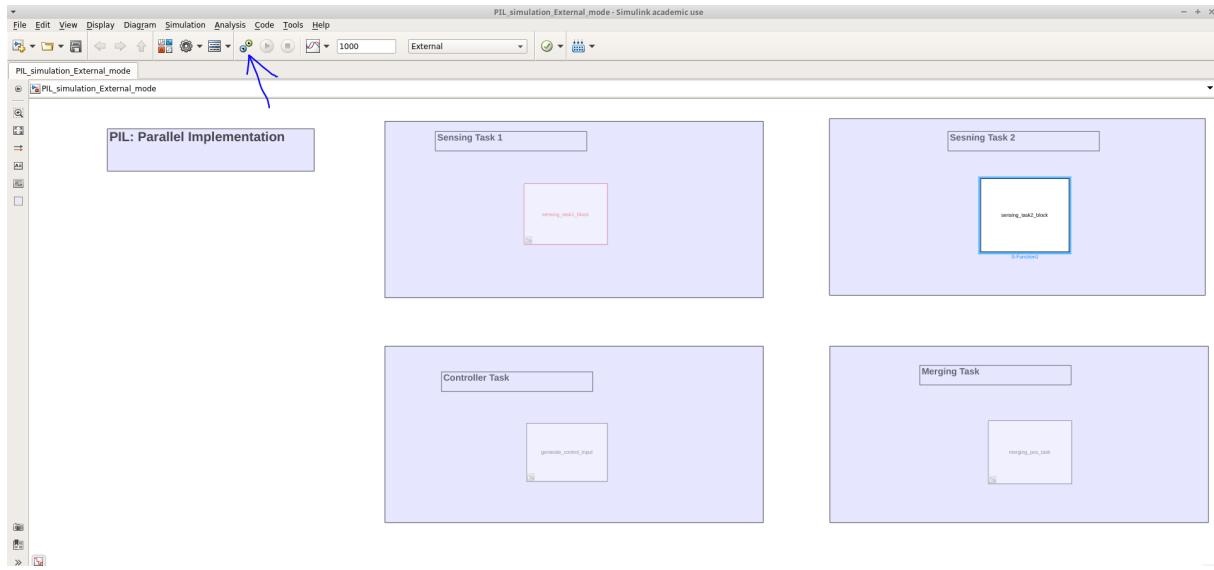
- click on the build button, as shown by the arrow in the below screenshot.



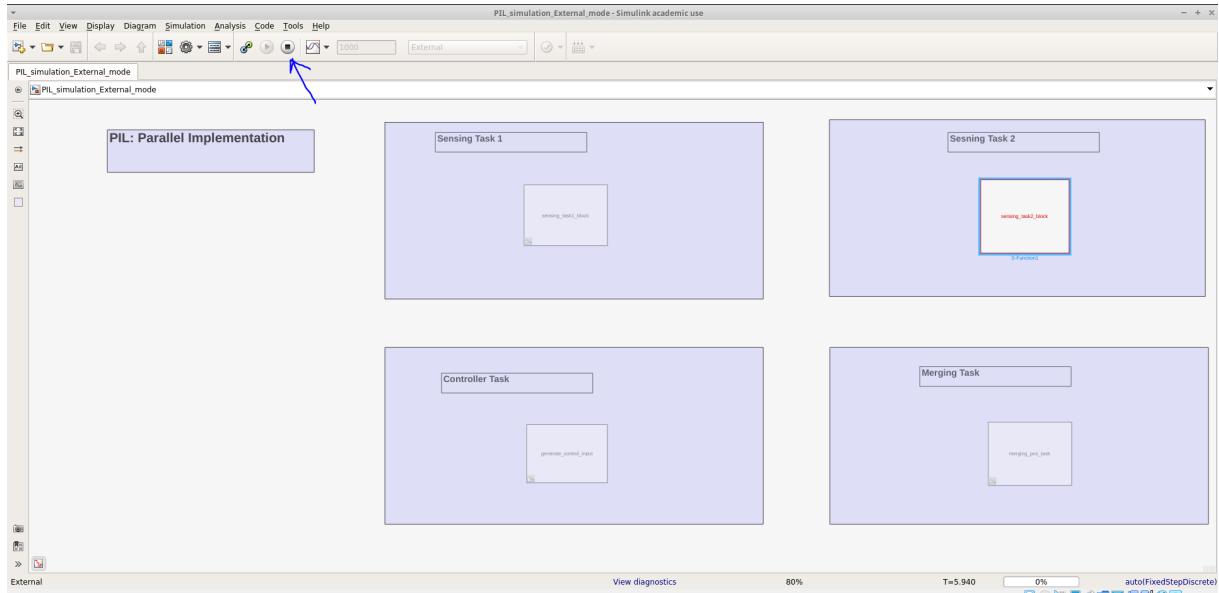
- after the build finishes, the Simulink model shows ready status; check the circle in the following screenshot.



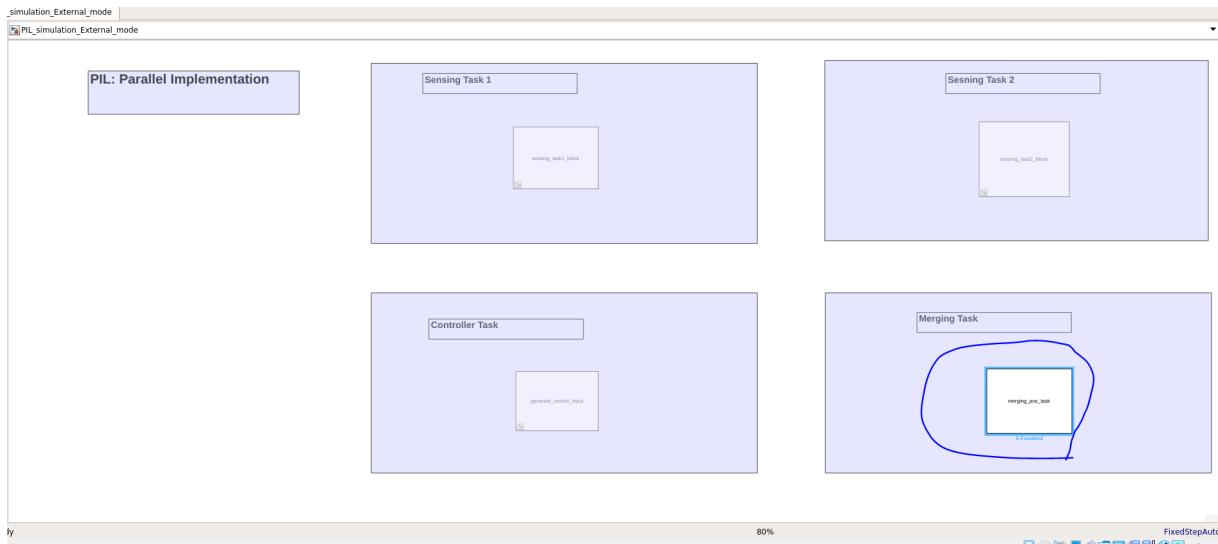
- if the build finishes successfully, ready status is available in the model. After that, run the task in external mode by clicking the connect to target button. Check the following screenshot for reference.



- after the above step, the sensing task connects to the target and starts running on the CompSOC; let it starts and runs for some time; check the highlighted circle in the below screenshot, then stop the running of the task by clicking on the stop button, check the arrow in the screenshot. No need to run the task for a long time.

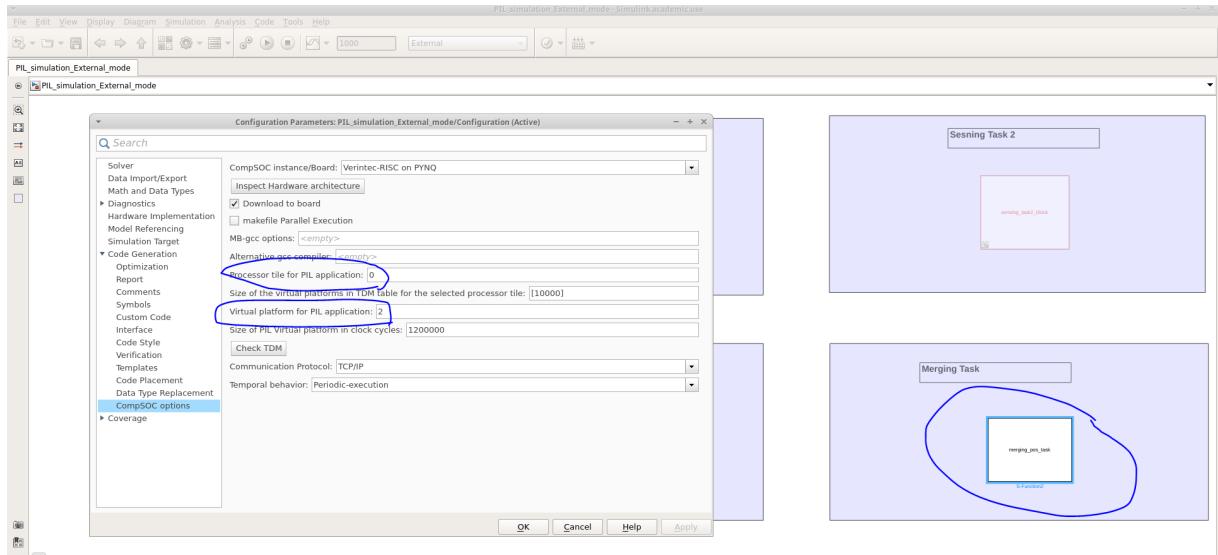


- move to the next task and comment all blocks except the one you want to run in external mode. Check the following screenshot; we have considered merging task after finishing the sensing task. Only the merging task is uncommented. Other blocks kept commenting.



- check the configuration setting. We are running the merging task on processor 0 and partition 2. So, the following setting should be considered,

- Processor tile for PIL application : 0
- Virtual platform for PIL application: 2



- in the same two terminal windows opened for the previous task, you can use the following highlighted commands from the screenshot.

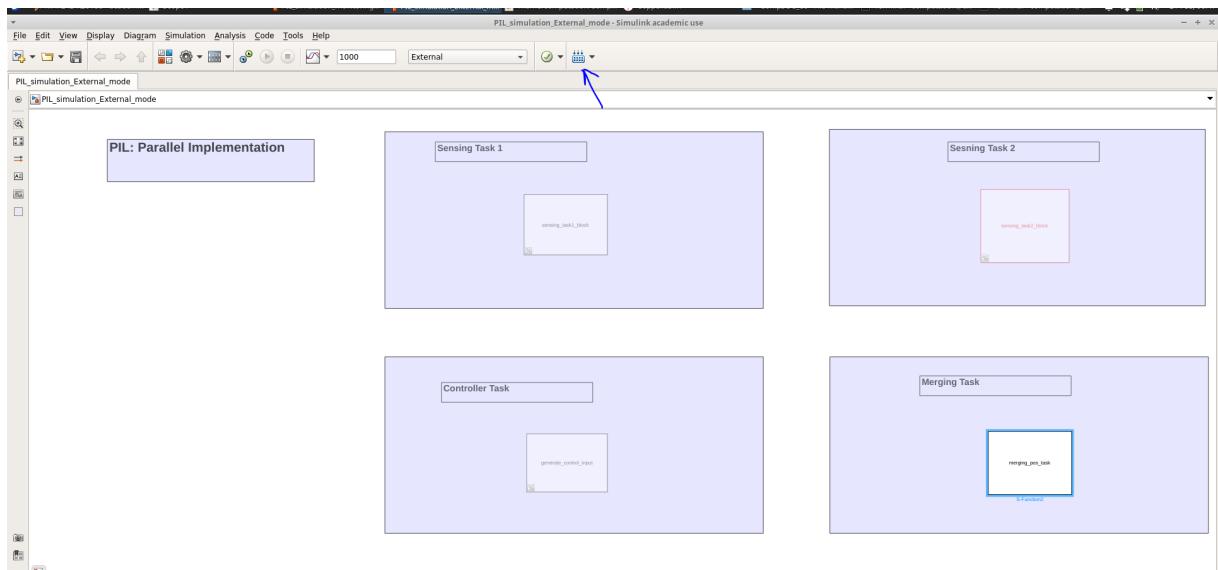
- Terminal 1 :
  - cd tutorial
  - cd monitoring-tools/
  - sudo ./channel Cheap\_bridge 0 2 9878
- Terminal 2 :
  - cd tutorial
  - ./readout.sh

```

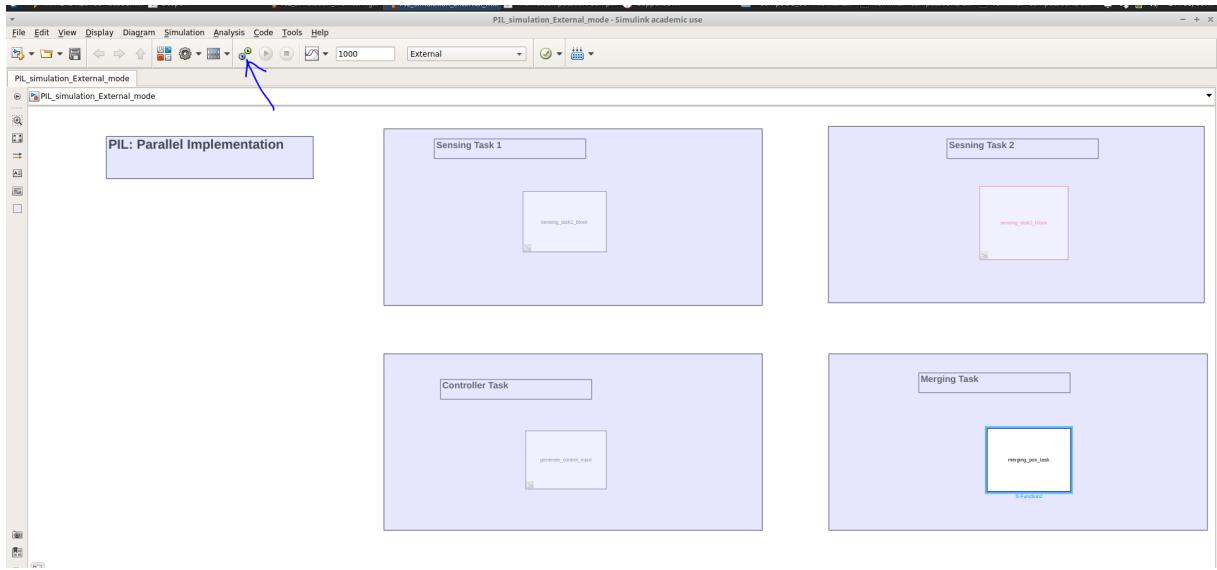
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 1 1 9878
+Socket created
+bind on port: 9878 done
+Listen done
-Waiting for incoming connections.
-Accepting socket
+Connection accepted
+Connection closed
0.000011
0.000029
-Waiting for incoming connections.
^C
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 2 1 9878
+Socket created
+bind on port: 9878 done
+Listen done
-Waiting for incoming connections.
-Accepting socket
+Connection accepted
+Connection closed
0.000011
0.000031
-Waiting for incoming connections.
^C
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 0 2 9878
+Socket created
+bind on port: 9878 done
+Listen done
-Waiting for incoming connections.

```

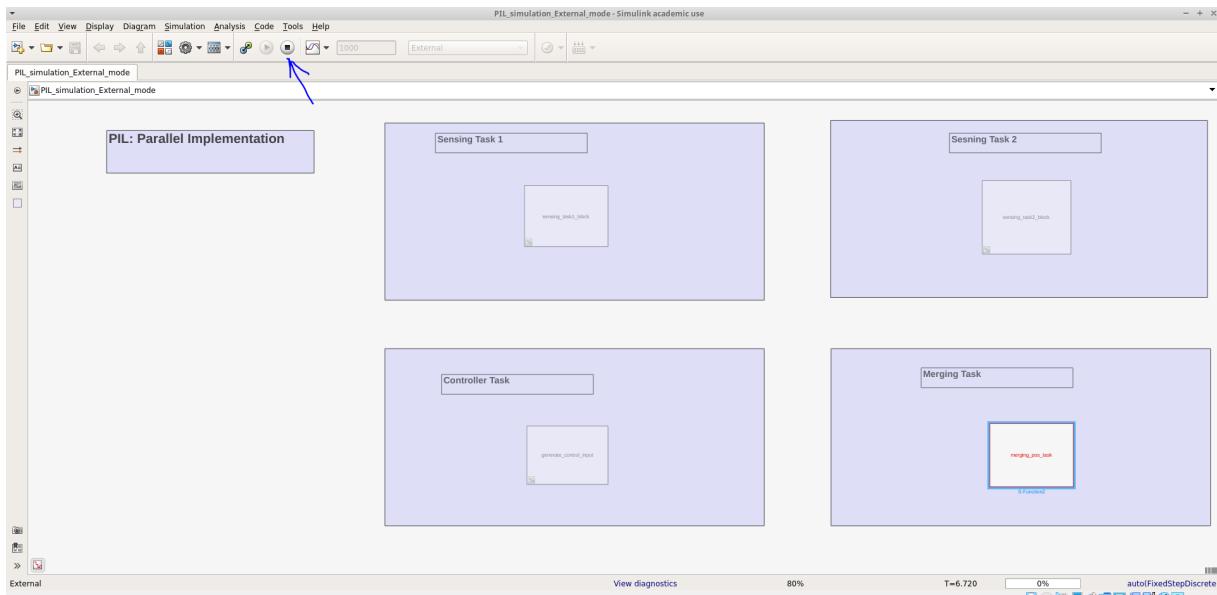
- click on the build button, as shown by the arrow in the below screenshot.



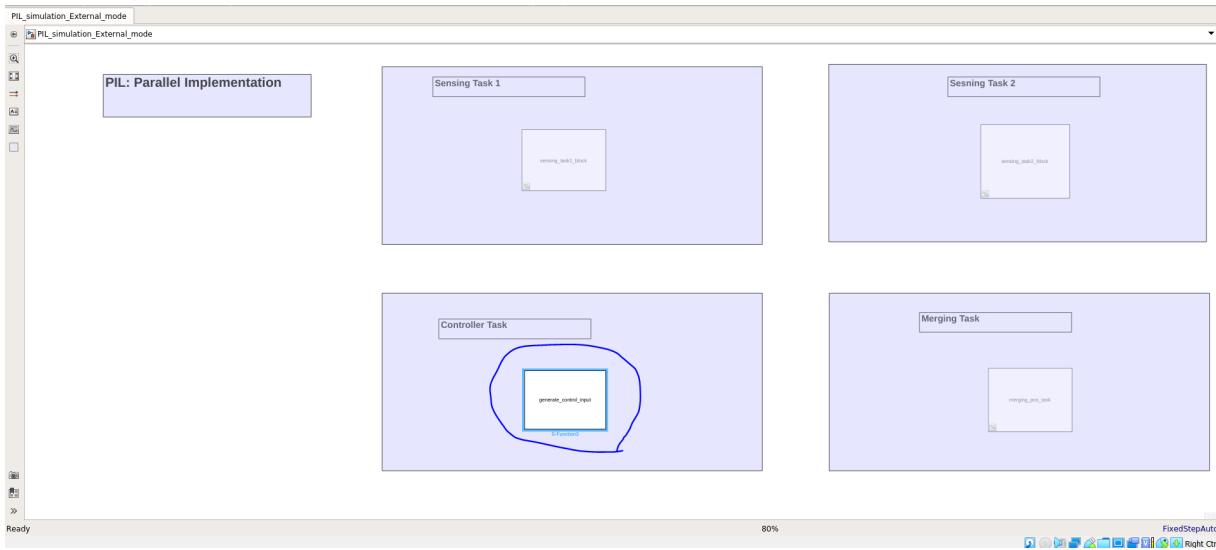
- if the build finishes successfully, ready status is available in the model. After that, run the task in external mode by clicking on the connect to target button. Check the following screenshot for reference.



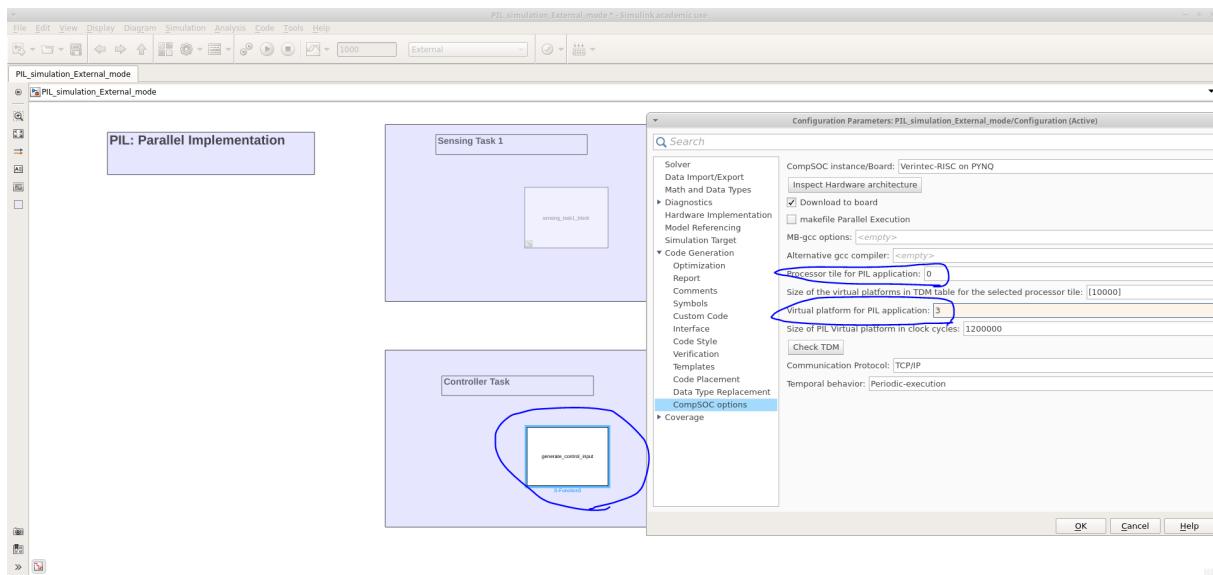
- after the above step, the merging task connects to the target and starts running on the CompSOC; let it starts and runs for some time, then stop the running of the task by clicking on the stop button. Check the arrow in the screenshot. No need to run the task for a long time.



- move to the next task and comment all blocks except the one you want to run in external mode. Check the following screenshot; we have considered the controller task after finishing the merging task. Only the controller task is uncommented. Other blocks kept commenting.



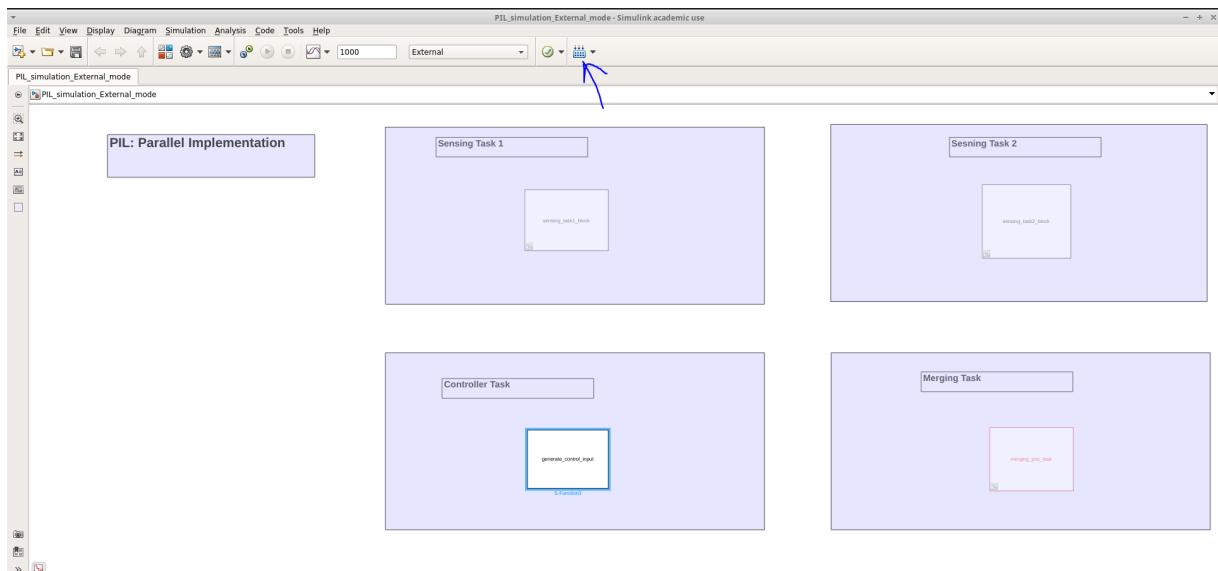
- check the configuration setting. We are running the controller task on processor 0 and partition 3. So, the following setting should be considered,
  - Processor tile for PIL application : 0
  - Virtual platform for PIL application: 3



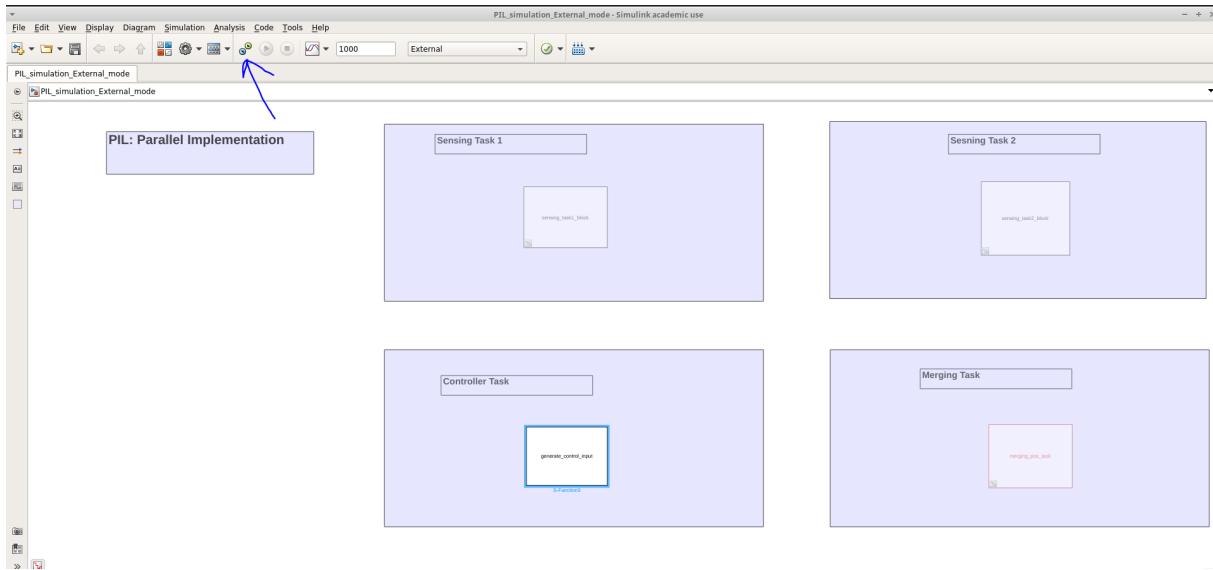
- in the same two terminal windows opened for the previous task, you can use the following highlighted commands from the screenshot.
  - Terminal 1 :
    - cd tutorial
    - cd monitoring-tools/
    - sudo ./channel Cheap\_Bridge 0 3 9878
  - Terminal 2 :
    - cd tutorial
    - ./readout.sh

The screenshot shows two terminal windows side-by-side. Both windows have the title 'computation@computation-virtual-machine:~'. The left terminal window displays the command 'student@pato-board:~/tutorial/monitoring-tools\$ sudo ./channel\_cheap\_bridge 1 1 9878' followed by several lines of log output related to socket creation, binding, and connection handling. The right terminal window displays the command 'student@pato-board:~/tutorial/monitoring-tools\$ sudo ./channel\_cheap\_bridge 2 1 9878' with similar log output. Both terminals show a series of stream operations (open, close) and connection events (accept, connect, closed) over time.

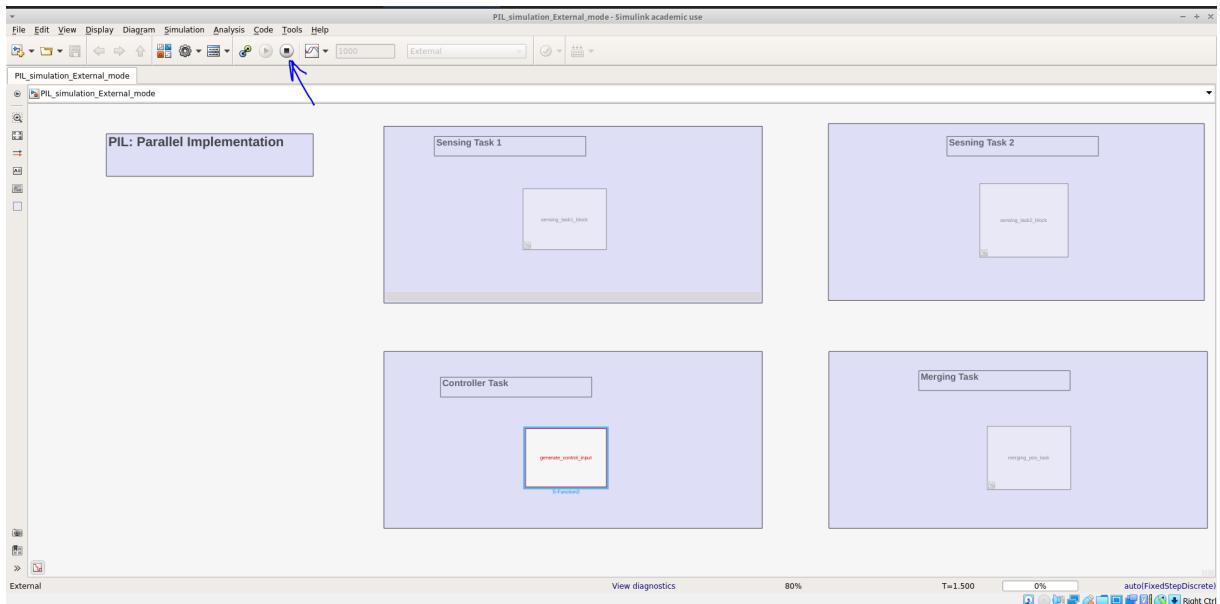
➤ click on the build button, as shown by the arrow in the below screenshot.



➤ if the build finishes successfully, ready status is available in the model. After that, run the task in external mode by clicking the connect to target button. Check the following screenshot for reference.



- after the above step, the controller task connects to the target and starts running on the CompSOC; let it starts and runs for some time, then stop the running of the task by clicking on the stop button; check the arrow in the screenshot. No need to run the task for a long time.



- after completing all the steps, run the following highlighted command in the same terminal window opened for the previous steps. This step is required for running the final PIL simulation, which will be run from **PIL\_simulation\_monitoring\_app.slx** Simulink model.

- Terminal 1 :
  - cd tutorial
  - cd monitoring-tools/
  - sudo ./channel Cheap\_Bridge 0 1 9878
- Terminal 2 :
  - cd tutorial
  - ./readout.sh

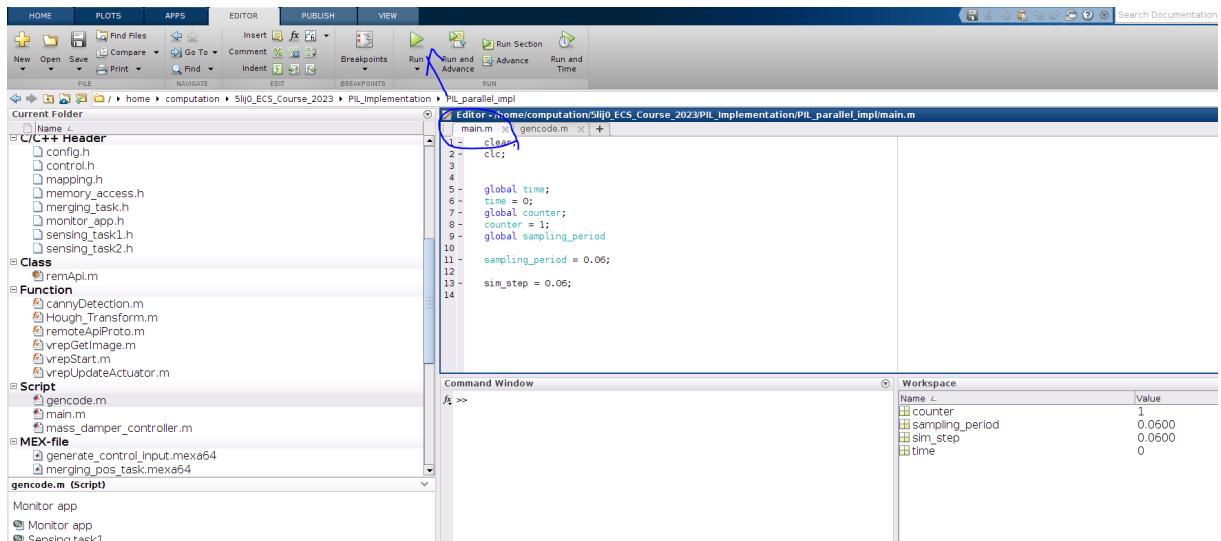
The image shows three terminal windows side-by-side, all running on the same host (computation@computation-virtual-machine). Each window displays the output of a different script named `channel_cheap_bridge` with varying parameters (1, 2, or 3) and port numbers (9878 or 9879). The scripts perform socket operations like binding, listening, accepting connections, and reading from streams. The output is identical across all three windows, indicating they are running simultaneously.

```

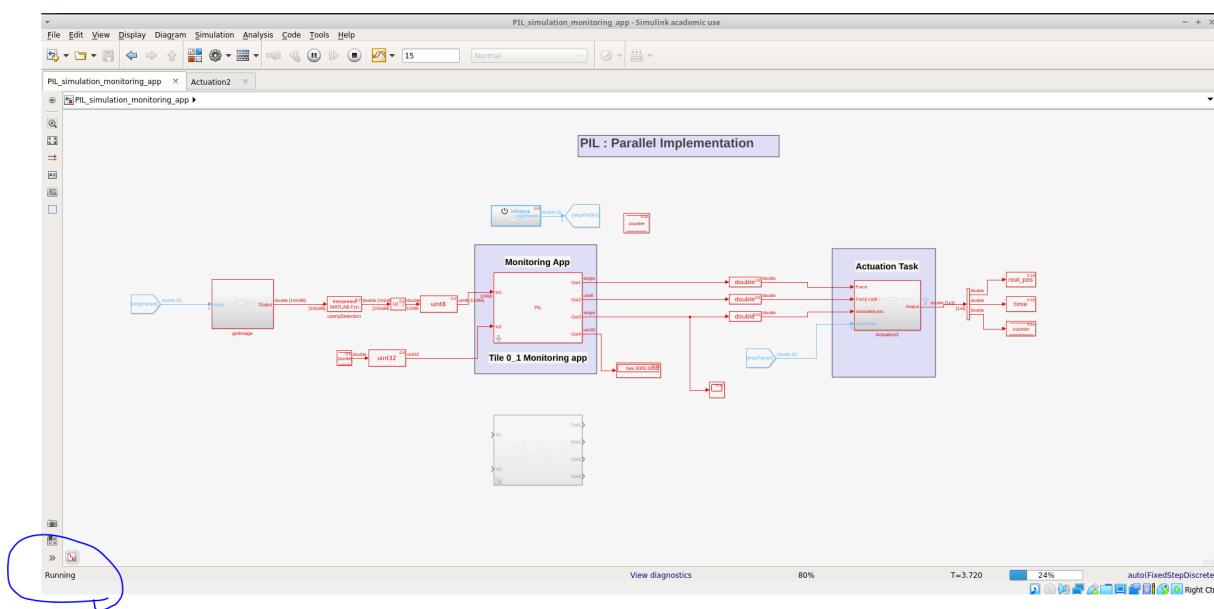
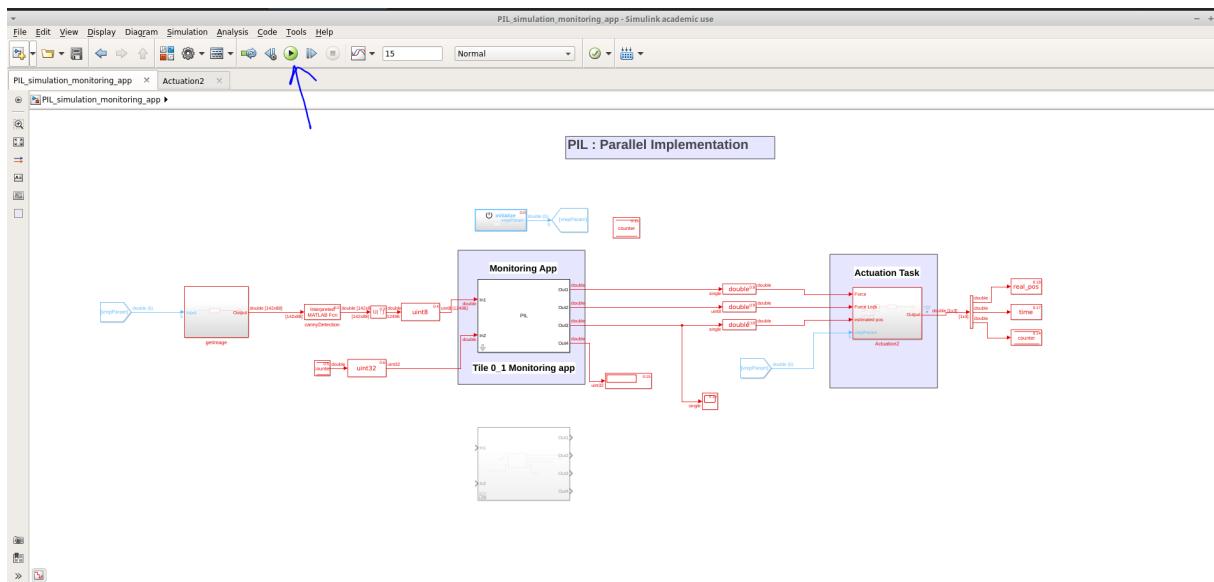
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 1 9878
+Socket created
+bind on port: 9878 done
+ Listen done
+ Waiting for incoming connections.
+ Accepting socket
+Connection accepted
+Connection closed
0.000001
0.000029
+Waiting for incoming connections.
^C
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 2 1 9878
+Socket created
+bind on port: 9878 done
+ Listen done
+ Waiting for incoming connections.
+ Accepting socket
+Connection accepted
+Connection closed
0.000011
0.000031
+Waiting for incoming connections.
^C
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 0 3 9878
+Socket created
+bind on port: 9878 done
+ Listen done
+ Waiting for incoming connections.
+ Accepting socket
+Connection accepted
+Connection closed
0.000010
0.000031
+Waiting for incoming connections.
^C
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 0 1 9878
+Socket created
+bind on port: 9878 done
+ Listen done
+ Waiting for incoming connections.

```

- next step is to run the main.m file and check the CoppeliaSim software; the mass-damper scene should be opened and ensure it is not running automatically; if it is running, stop it.



- open the **PIL\_simulation\_monitoring\_app.slx** Simulink model, and click on the Run button. This will start running the simulation. It will take some time to simulate as it initializes all the required settings for running the simulation. So wait until you see the running status. Check the following screenshots for reference.
- check the current position returned from the monitoring app using the scope linked to the monitoring app PIL block. Also, visualize the output in CoppeliaSim.

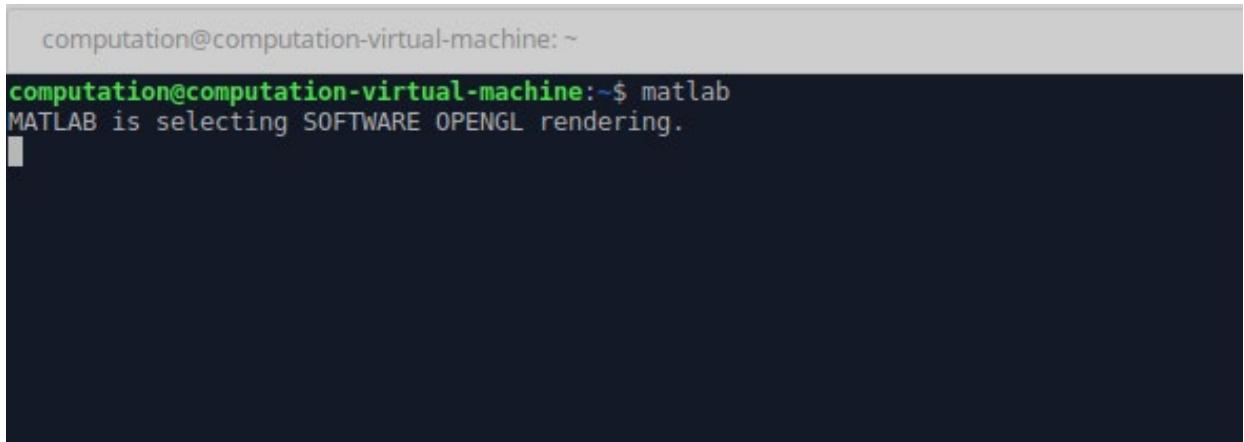


### Section III: PiL implementation for pipeline configuration

- open MATLAB: To open MATLAB in the IMACS virtual machine, run the following command in the new terminal.

**Note:** If MATLAB is already opened, then no need to reopen it.

```
$ matlab
```

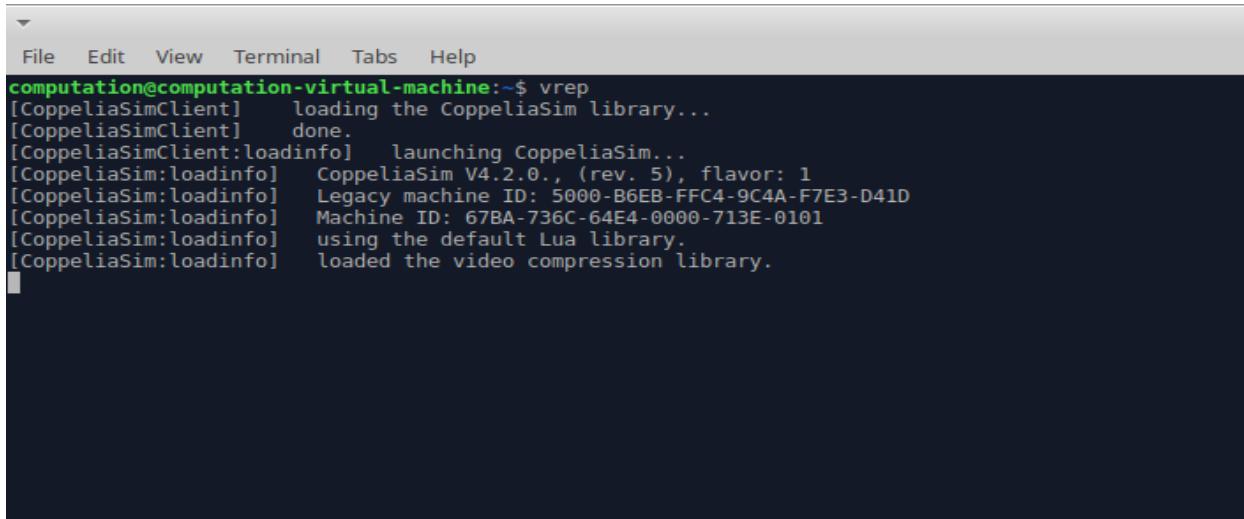


A terminal window showing the MATLAB startup process. The prompt is computation@computation-virtual-machine:~. The command \$ matlab is entered, followed by the message "MATLAB is selecting SOFTWARE OPENGL rendering.".

- open CoppeliaSim: to open the CoppeliaSim, run the following command in a new terminal.

**Note:** If CoppeliaSim is already opened, then no need to reopen it.

```
$ vrep
```



A terminal window showing the CoppeliaSim startup process. The prompt is computation@computation-virtual-machine:~\$ vrep. The command is entered, followed by several log messages indicating the loading of the CoppeliaSim library, launching the simulation, and providing machine IDs.

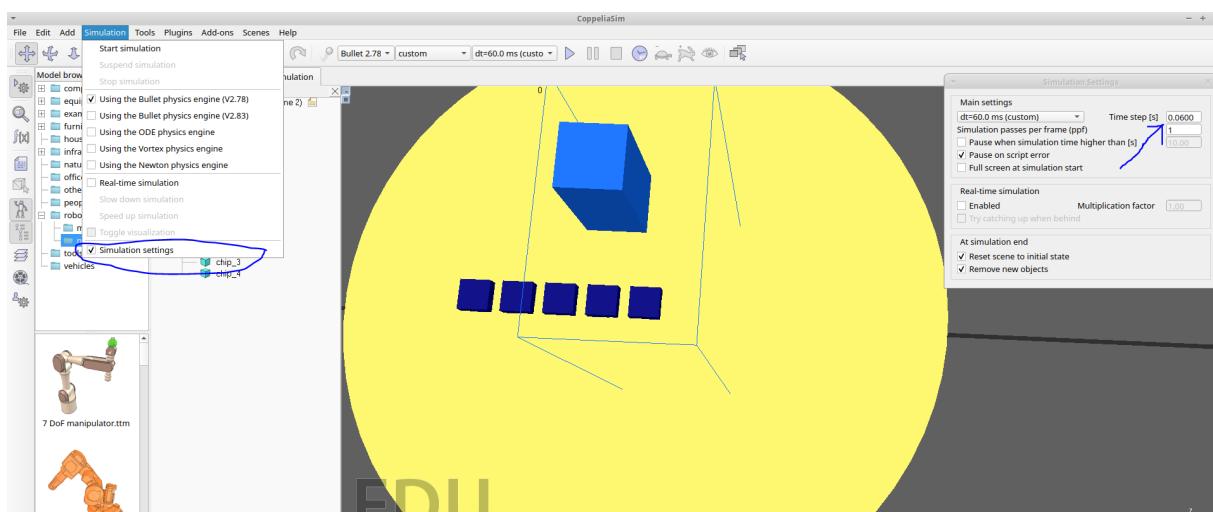
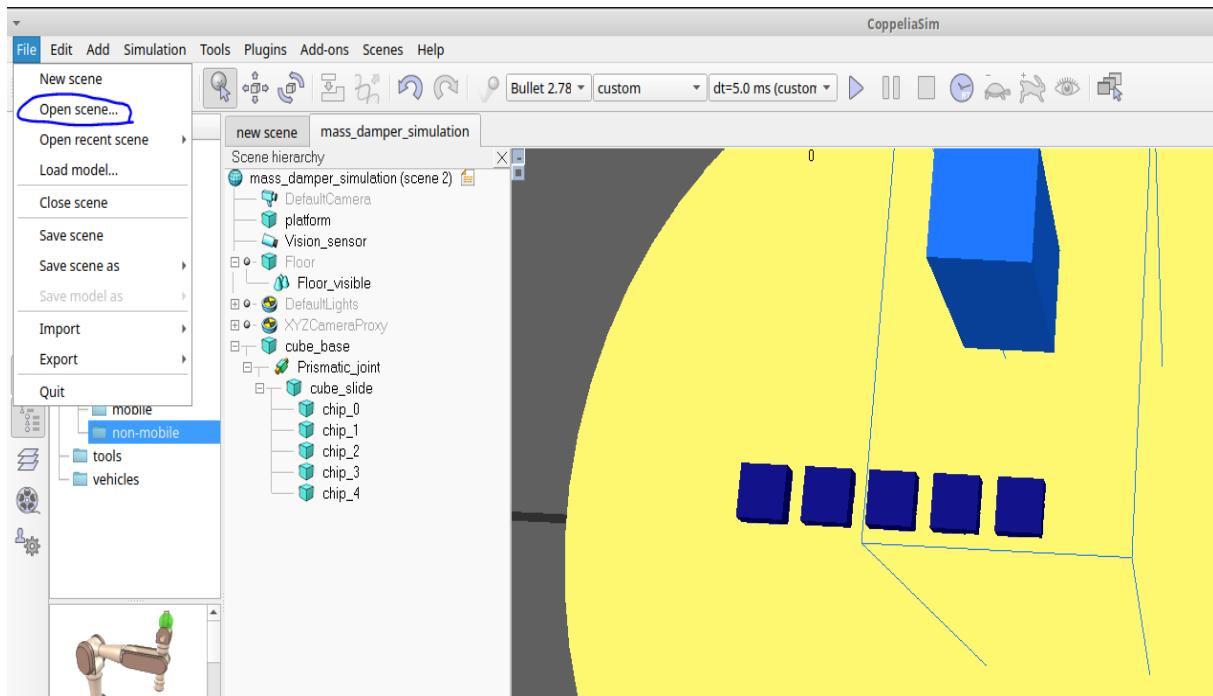
```
[CoppeliaSimClient]    loading the CoppeliaSim library...
[CoppeliaSimClient]    done.
[CoppeliaSimClient:loadinfo]  launching CoppeliaSim...
[CoppeliaSim:loadinfo]  CoppeliaSim V4.2.0., (rev. 5), flavor: 1
[CoppeliaSim:loadinfo]  Legacy machine ID: 5000-B6EB-FFC4-9C4A-F7E3-D41D
[CoppeliaSim:loadinfo]  Machine ID: 67BA-736C-64E4-0000-713E-0101
[CoppeliaSim:loadinfo]  using the default Lua library.
[CoppeliaSim:loadinfo]  loaded the video compression library.
```

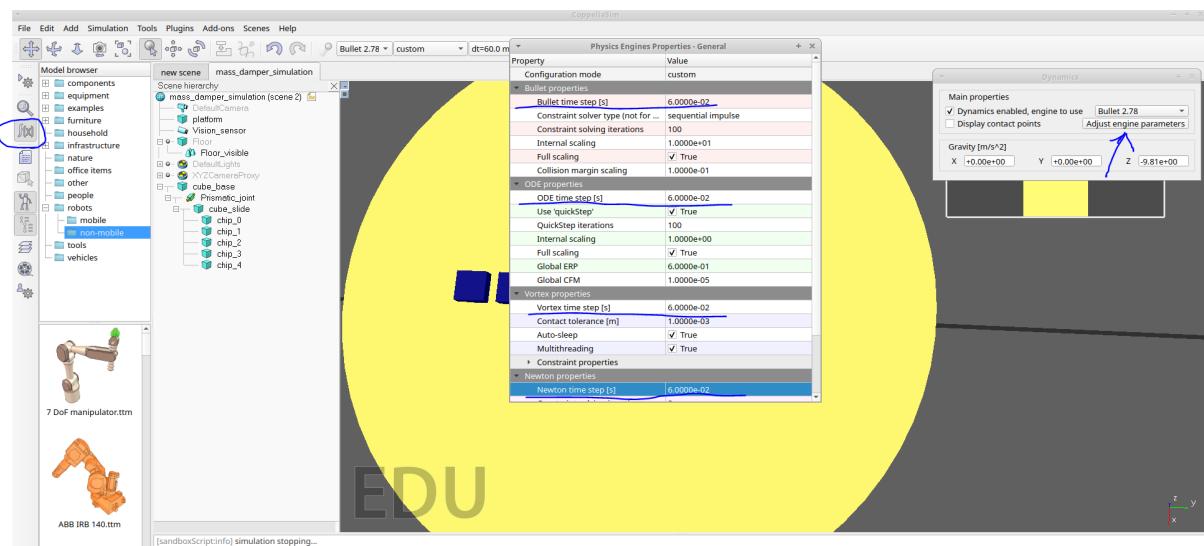
- open CoppeliaSim scene for the mass-damper system.

- go to File→Open scene→ select the following directory  
/home/computation/5lij0\_ECS\_Course\_2023/PIL\_Implementation/PIL\_pipeline\_i  
mpl

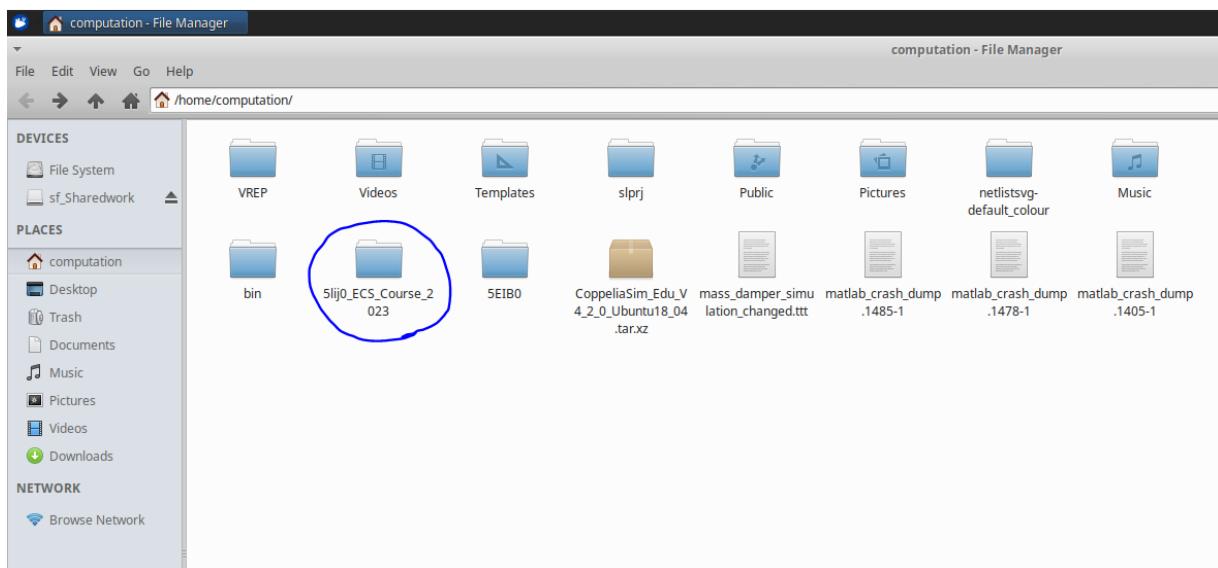
open **mass\_damper\_simulation.ttt**, check the following screenshot for reference.

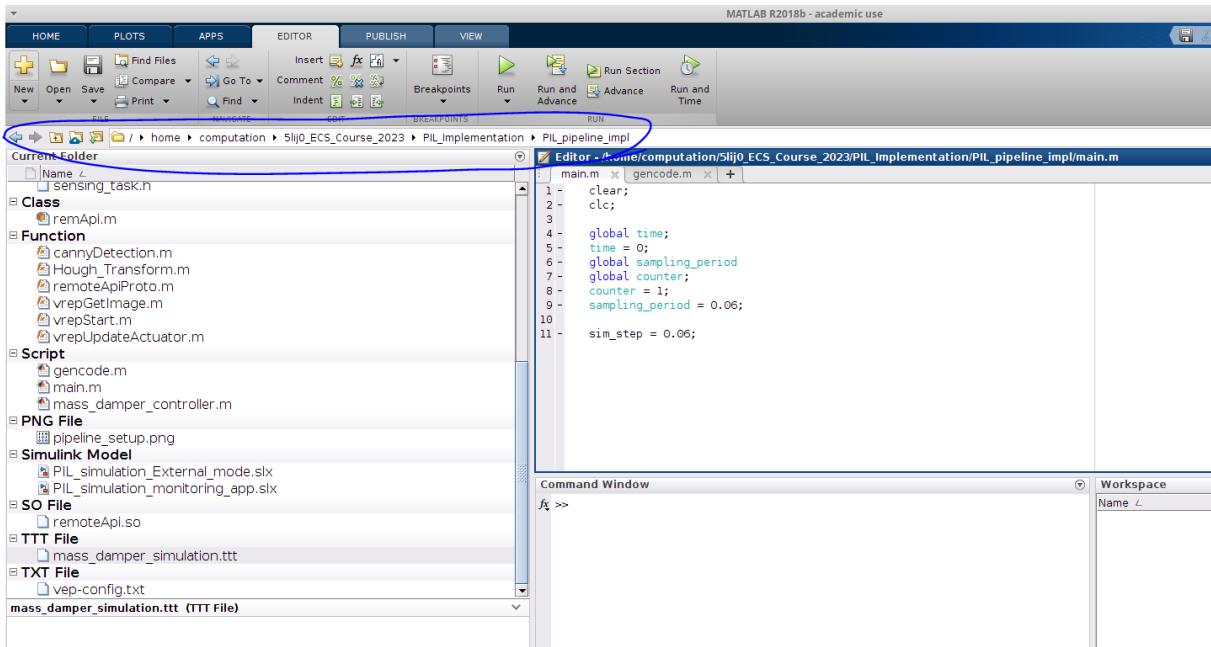
We have implemented the PIL of sequential configuration considering the 60ms sample period. The same sample period should be used in CoppeliaSim. In the case of different sample periods, students can make the changes in the CoppeliaSim, as explained below. For example, the following screenshot shows how to change the time step in the CoppeliaSim scene.





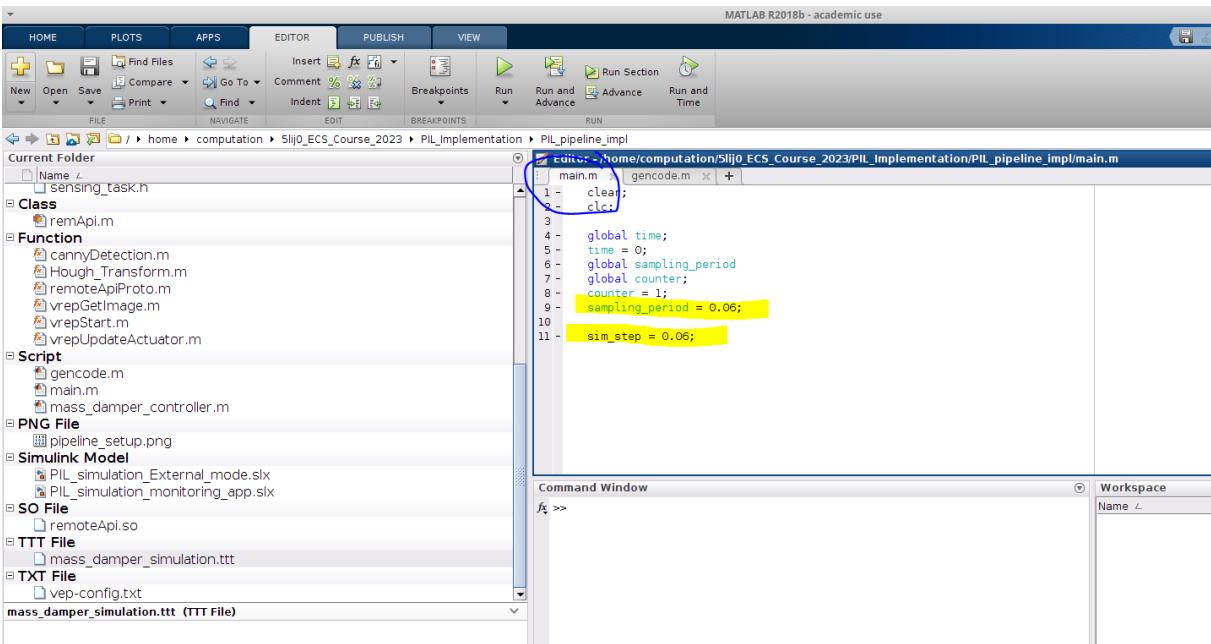
- open the folder for PiL pipeline implementation.
  - go to the following folder path and open in MATLAB
   
computation → 5lij0\_ECS\_Course\_2023 → PIL\_Implementation → PIL\_pipeline\_impl





- open the main.m file and make changes for the highlighted parameters, as shown in the following screenshot.

Note: In this screenshot, we have explained for the 60ms sample period, students can use different sample periods according to the implementation design and make the changes for the following parameters. Once the changes are done, run the main.m script. This file includes the initialization, which is required for the PiL simulation.



- open the gencode.m file and run the script. This file uses legacy code tool functionality, integrating the existing C or C++ functions into MATLAB Simulink. Once this gencode.m runs, it creates the S-function block inside the Simulink model.

```

Editor - /home/computation/Slij0_ECS_Course_2023/PIL_Implementation/PIL_pipeline_impl/genode.m
main.m genode.m + [1]
1 %> Monitor app
2 def_monitor = legacy_code('initialize');
3 def_monitor.SourceFiles = {'monitor_app.c', 'memory_access.c'};
4 def_monitor.HeaderFiles = {'monitor_app.h', 'memory_access.h', 'config.h', 'mapping.h'};
5 def_monitor.SFunctionName = 'monitor_app_block_pipe';
6 def_monitor.IncPaths = {'.', './CompSOC_ec_target/files'};
7 def_monitor.OutputFcnSpec = 'void monitor(uint8 ui[12496], single yi[1], uint8 y2[1], uint32 u2[1], si
8 legacy_code('sfcn_cmex_generate', def_monitor);
9 legacy_code('sfcn_tlc_generate', def_monitor);
10 legacy_code('compile', def_monitor);
11 legacy_code('slblock_generate', def_monitor);
12
13 %> Sensing task 1
14 def_sensing = legacy_code('initialize');
15 def_sensing.SourceFiles = {'sensing_task_1.c', 'memory_access.c'};
16 def_sensing.HeaderFiles = {'sensing_task.h', 'memory_access.h', 'config.h', 'mapping.h'};
17 def_sensing.SFunctionName = 'sensing_block_1';
18 def_sensing.IncPaths = {'.', './CompSOC_ec_target/files'};
19 def_sensing.OutputFcnSpec = 'void hough_transform()';
20 legacy_code('sfcn_cmex_generate', def_sensing);
21 legacy_code('sfcn_tlc_generate', def_sensing);
22 legacy_code('compile', def_sensing);
23
24 %> Sensing task 2
25 def_sensing = legacy_code('initialize');
26 def_sensing.SourceFiles = {'sensing_task_2.c', 'memory_access.c'};
27 def_sensing.HeaderFiles = {'sensing_task.h', 'memory_access.h', 'config.h', 'mapping.h'};
28 def_sensing.SFunctionName = 'sensing_block_2';
29 def_sensing.IncPaths = {'.', './CompSOC_ec_target/files'};
30 def_sensing.OutputFcnSpec = 'void hough_transform()';
31 legacy_code('sfcn_cmex_generate', def_sensing);
32 legacy_code('sfcn_tlc_generate', def_sensing);
33 legacy_code('compile', def_sensing);

```

- open the **vrepUpdateActuator.m** file. Make the necessary changes in the file, as shown below in the screenshot. This file is required for the actuation.

```

Editor - /home/computation/Slij0_ECS_Course_2023/PIL_Implementation/PIL_pipeline_impl/vrepUpdateActuator.m
main.m vrepUpdateActuator.m + [1]
1 function output = vrepUpdateActuator(input)
2
3 vrepParam = [
4     1 clientID    -> vrepParam(1);
5     2 cam         -> vrepParam(2);
6     3 motor        -> vrepParam(3);
7     4 chip        -> vrepParam(4);
8     5 base        -> vrepParam(5) ];
9
10 vrepParam = input(3:end);
11 force_lock = input(2);
12 force = input(1);
13
14 damping_constant = 1;
15 sim_step = 0.06;
16
17 global vrep;
18 global prev_sign;
19 global total_force;
20 global time;
21 global counter;
22 global prev_pos;
23 global sampling_period;
24
25 % Initiate Data
26 if (time == 0)
27     [returnCode, chip_pos] = vrep.simxGetObjectPosition(vrepParam(1), vrepParam(4));
28     chip_pos = double((chip_pos(2)-1.4 + 0.025/2)) + 0.001;
29     prev_pos = chip_pos;
30     total_force = 0;

```

- open the **config.h** file. Make the necessary changes in the file, as shown below in the screenshot. This file contains all the required parameters used in the PiL simulation.

```

FILE
New Open Save Compare Go To Insert Comment Breakpoints
FILE NAVIGATE EDIT BREAKPOINTS
Current Folder Name ↴
C Source
control.c
dump.c
generate_control_input.c
memory_access.c
monitor_app.c
monitor_app_block_pipe.c
read_hough_comb.c
sensing_block_1.c
sensing_block_2.c
sensing_task_1.c
sensing_task_2.c
C/C++ Header
config.h
control.h
mapping.h
memory_access.h
monitor_app.h
sensing_task.h
Class
remApI.m
Function
cannyDetection.m
Hough_Transform.m
remoteApIProto.m
config.h (C/C++ Header)
Editor - /home/computation/5lij0_ECS_Course_2023/PIL_Implementation/PIL_pipelineImpl/config.h
main.m config.h
1 //include "config.h"
2
3 #define IMAGE_WIDTH (400) // Pixels
4 #define IMAGE_HEIGHT (142) // Pixels
5 #define IMAGE_SIZE (12496) // Pixels
6 #define HALF_IMG (6248)
7 #define HOUGH_THRESHOLD (25)
8 #define PADDING_SIZE_THETA (3) // Pixels
9 #define PADDING_SIZE_RHO (3) // Pixels
10 #define MAX_PEAK_NUMBER (2)
11 #define DESIRED_POS (40) // Pixels
12 #define PIXEL_RATIO (0.001) // 1 millimeter
13 #define THETA_neg_5_pos_5
14 // #define THETA_0_pos_5
15 // #define THETA_0_pos_30
16 #define ms_35 ((uint32_t)1400000)
17 #define ms_36 ((uint32_t)1440000)
18 #define ms_37 ((uint32_t)1480000)
19 #define ms_38 ((uint32_t)1520000)
20 #define ms_39 ((uint32_t)1560000)
21
22
23
24
25
26
27
28 // Controller
29 #define DAMPING_CONST ((float)1.0)
30 #define SAMPLING_TIME ((float)0.060)
31 #define INV_SAMPLING_TIME ((float)16.66)
32 #define K_P ((float)0.02) //0.02
33 #define K_D ((float)0.01) //0.01
34 #define K_I ((float)0)
35 #define PI (3.1416)
36 #endif

```

Command Window

>>

Workspace

- for tuning the PID controller gains, students can change the gains in the **config.h** file, check the below screenshot.

```

FILE
New Open Save Compare Go To Insert Comment Breakpoints
FILE NAVIGATE EDIT BREAKPOINTS
Current Folder Name ↴
C Source
control.c
dump.c
generate_control_input.c
memory_access.c
monitor_app.c
monitor_app_block_pipe.c
read_hough_comb.c
sensing_block_1.c
sensing_block_2.c
sensing_task_1.c
sensing_task_2.c
C/C++ Header
config.h
control.h
mapping.h
memory_access.h
monitor_app.h
sensing_task.h
Class
remApI.m
Function
cannyDetection.m
Hough_Transform.m
remoteApIProto.m
config.h (C/C++ Header)
Editor - /home/computation/5lij0_ECS_Course_2023/PIL_Implementation/PIL_pipelineImpl/config.h
main.m config.h
1 //include "config.h"
2
3 #define IMAGE_WIDTH (400) // Pixels
4 #define IMAGE_HEIGHT (142) // Pixels
5 #define IMAGE_SIZE (12496) // Pixels
6 #define HALF_IMG (6248)
7 #define HOUGH_THRESHOLD (25)
8 #define PADDING_SIZE_THETA (3) // Pixels
9 #define PADDING_SIZE_RHO (3) // Pixels
10 #define MAX_PEAK_NUMBER (2)
11 #define DESIRED_POS (40) // Pixels
12 #define PIXEL_RATIO (0.001) // 1 millimeter
13 #define THETA_neg_5_pos_5
14 // #define THETA_0_pos_5
15 // #define THETA_0_pos_30
16 #define ms_35 ((uint32_t)1400000)
17 #define ms_36 ((uint32_t)1440000)
18 #define ms_37 ((uint32_t)1480000)
19 #define ms_38 ((uint32_t)1520000)
20 #define ms_39 ((uint32_t)1560000)
21
22
23
24
25
26
27
28 // Controller
29 #define DAMPING_CONST ((float)1.0)
30 #define SAMPLING_TIME ((float)0.060)
31 #define INV_SAMPLING_TIME ((float)16.66)
32 #define K_P ((float)0.02) //0.02
33 #define K_D ((float)0.01) //0.01
34 #define K_I ((float)0)
35 #define PI (3.1416)
36 #endif

```

Command Window

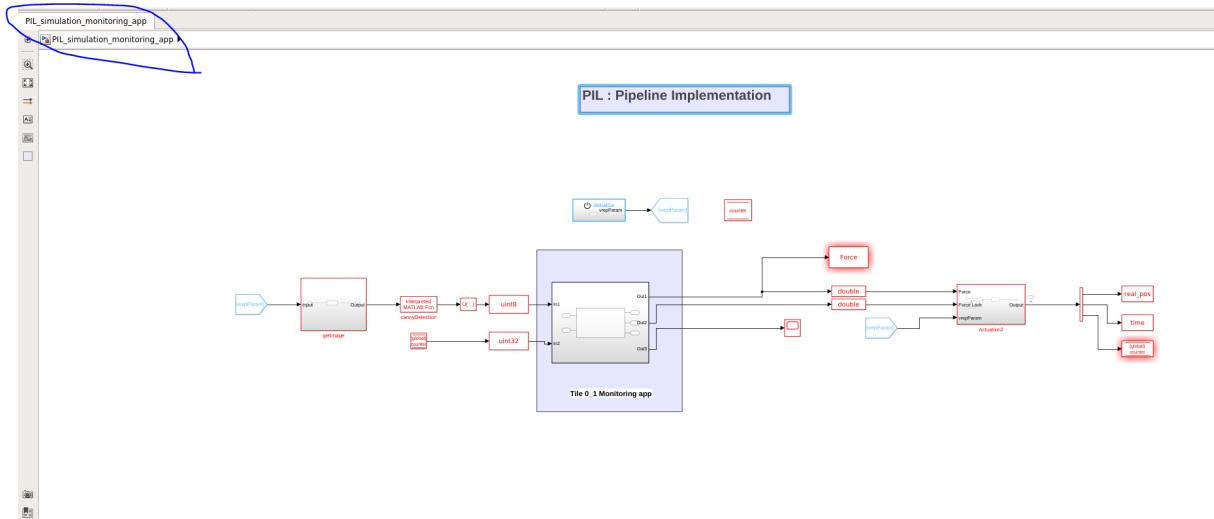
>>

Workspace

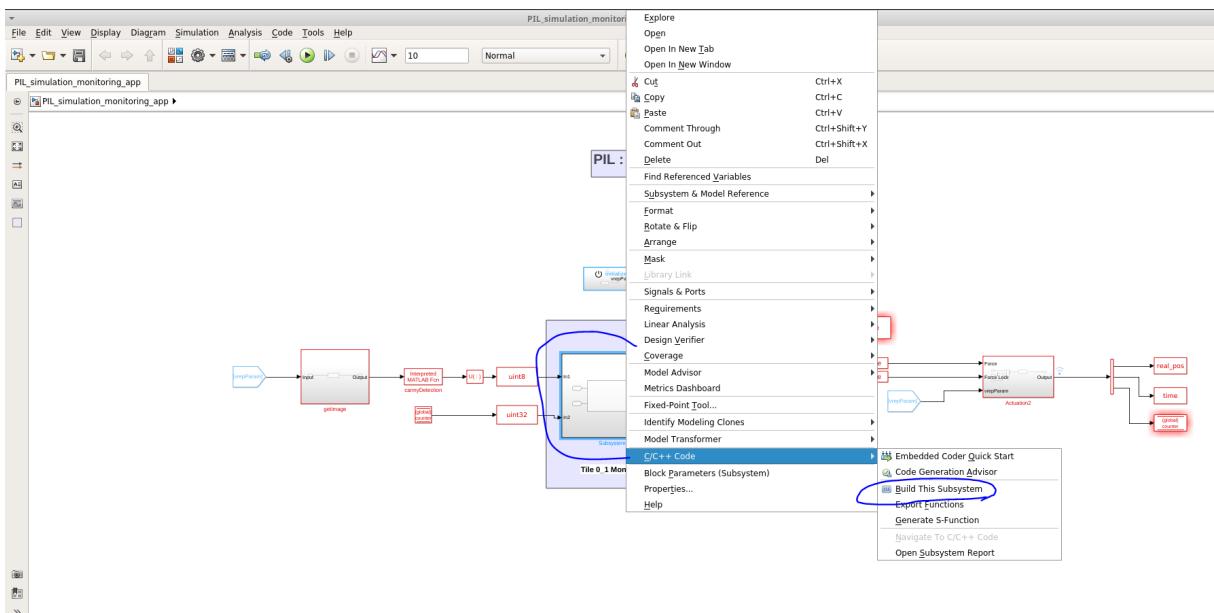
- once the steps mentioned above are done, open the PiL simulation file named **PIL\_simulation\_monitoring\_app.slx**

Students can find this file in the following directory:

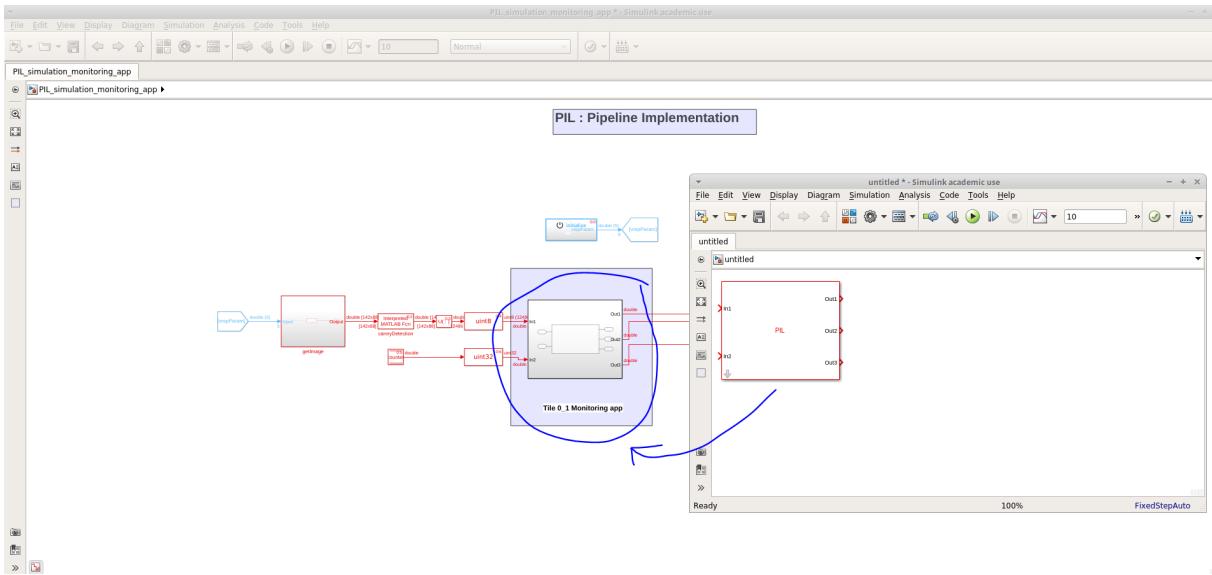
computation → 5lij0\_ECS\_Course\_2023 → PIL\_Implementation → PIL\_pipelineImpl



- right-click on the block, select the C/C++ code option, and click on build this subsystem. Check the following screenshot for reference.



- once, the build process is completed successfully; it will generate the PIL block; check the following screenshot:



- after completing the above steps, check the vep\_config.txt file. In this example, we have considered the 60ms sample period. Therefore, in the vep\_config.txt, scheduling is given according to the 60ms sample period.
- we have provided the vep\_config.txt file in the following folder  
`/home/computation/5lij0_ECS_Course_2023/PIL_Implementation/PIL_pipeline_impl`
- students need to copy the contents from the above vep\_config file into the main **vep-config.txt** file, which is inside the following folder path :  
 or the student can enter the TDM scheduling calculation directly in the vep-config.txt file, which is stored in the following location.

`/home/computation/CompSOC_ec_target/CompSOC_ec/+CompSOC_ec/vep-config.txt`

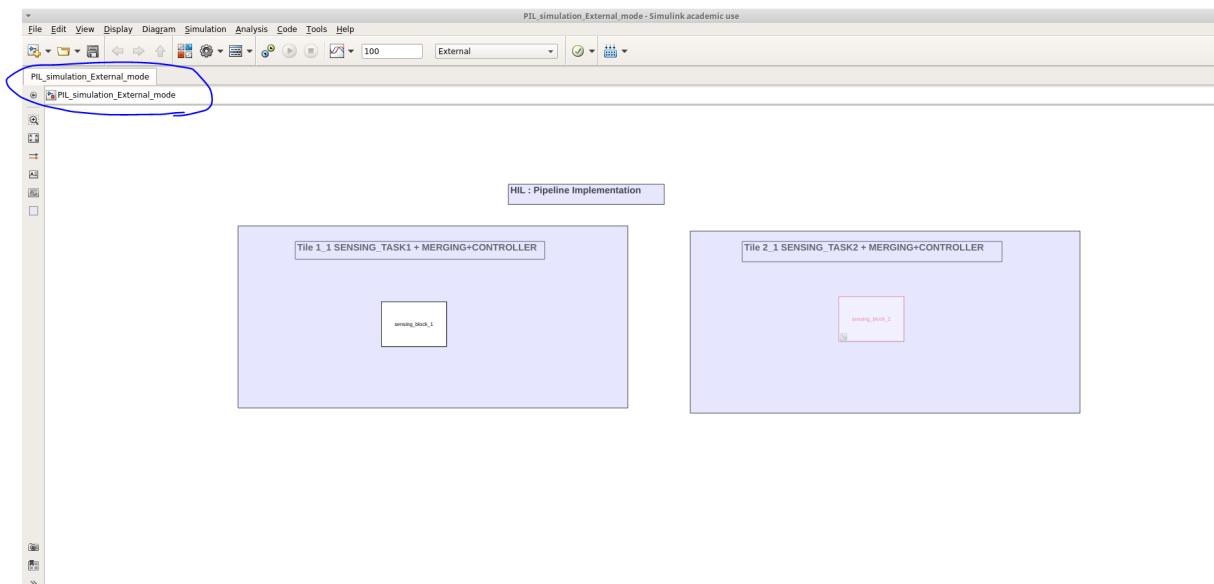
- the contents in the vep\_config file should be as follows:

```

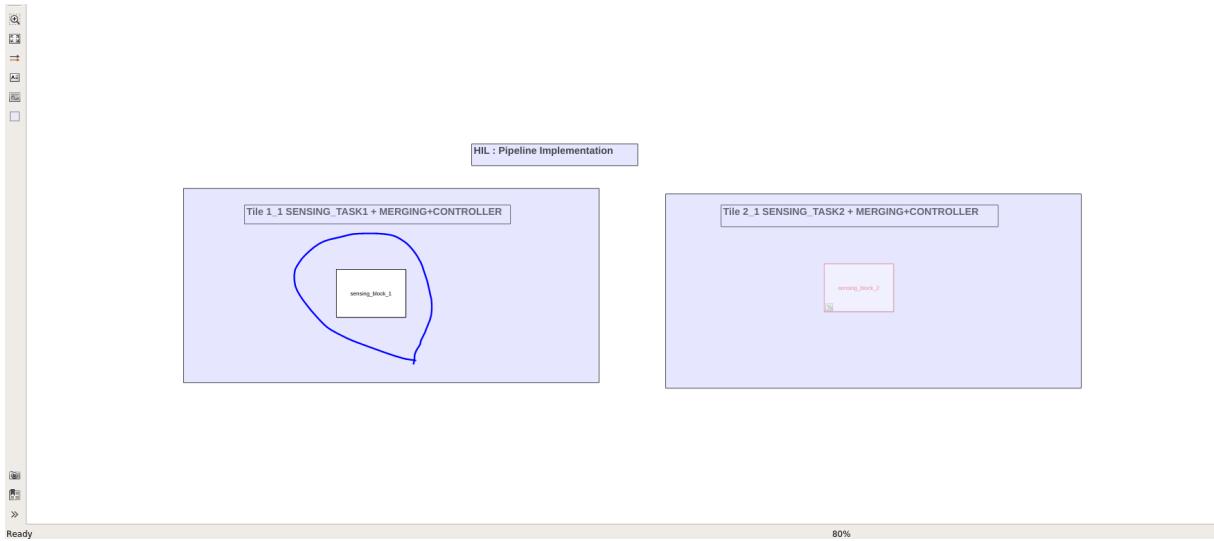
## you can comment out & modify the "on tile ... " lines
##
## - partitions can only have a memory size of 32K or 64K
## - the size of all partitions on a tile must be at most 96K
##   (i.e. 128K including the system application)
## - default memory is 32K, default stack is 4K
##
## use /opt/riscv/bin/riscv32-unknown-elf-size [-A] *.elf
## to analyse a partition's memory usage
##
## memory and stack allocation
##
on tile 0 partition 1 has 64K memory and 4K stack
#on tile 0 partition 2 has 64K 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 64K 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 2391000 cycles
#on tile 0 next slot is for partition 2 with 30000 cycles
#on tile 0 next slot is for partition 1 with 71000 cycles
on tile 1 next slot is for partition 1 with 2391000 cycles
#on tile 1 next slot is for partition 2 with 789000 cycles
on tile 2 next slot is for partition 1 with 2391000 cycles
#on tile 2 next slot is for partition 2 with 800000 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

```

- open the PIL\_simulation\_External\_mode.slx. The folder path to this Simulink model is as follows:  
`/home/computation/5lij0_ECS_Course_2023/PIL_Implementation/PIL_pipeline_impl`
- PIL\_simulation\_External\_mode.slx file has two blocks, i.e., one block for (sensing task 1 + merging + controller tasks) and other blocks for (sensing task 2 + merging + controller tasks), which will run in the external mode on CompSOC based on the TDM scheduling. To run the task independently in the external mode, follow the below steps and check the screenshot for reference:

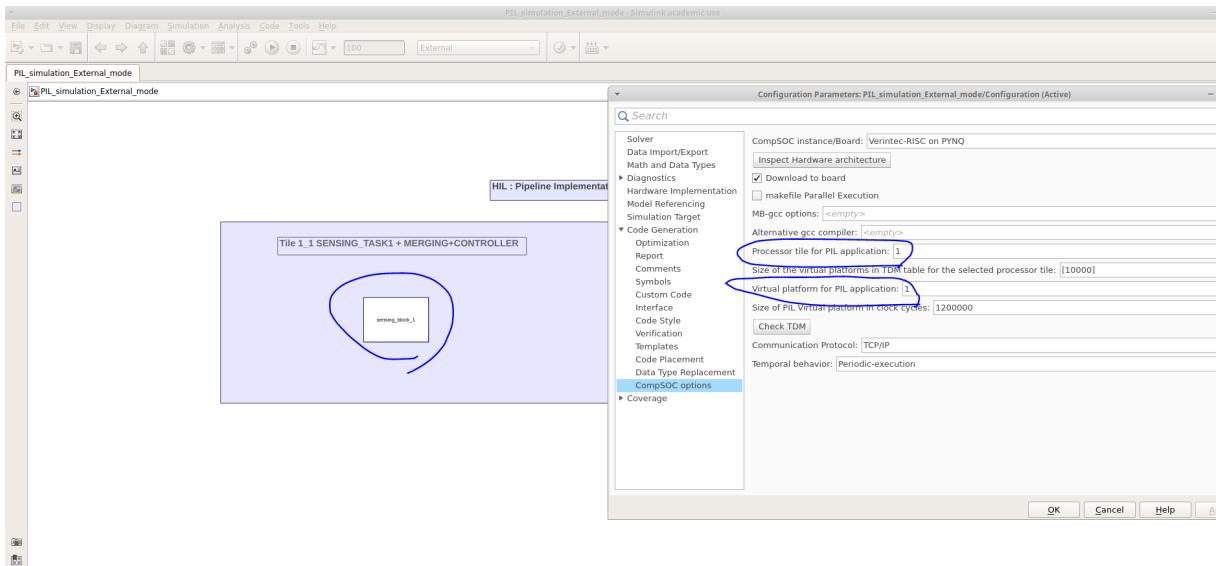


- to run the (sensing task 1 + merging + controller tasks) in the external mode, comment the other Simulink block and only uncomment the block which needs to be run. In the below screenshot, we consider the (sensing task 1 + merging + controller tasks) block first, and other blocks kept commenting.

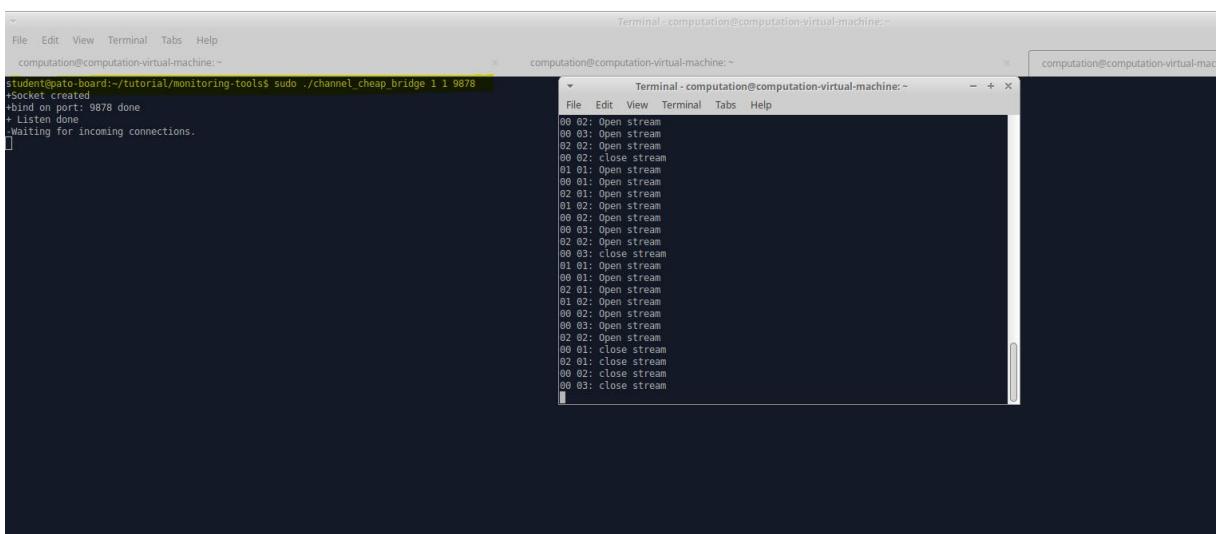


- open the configuration setting and check the required setting for the particular task. In this example, we run (sensing task 1 + merging + controller tasks) on processor 1 and partition 1. Therefore, consider the following changes :

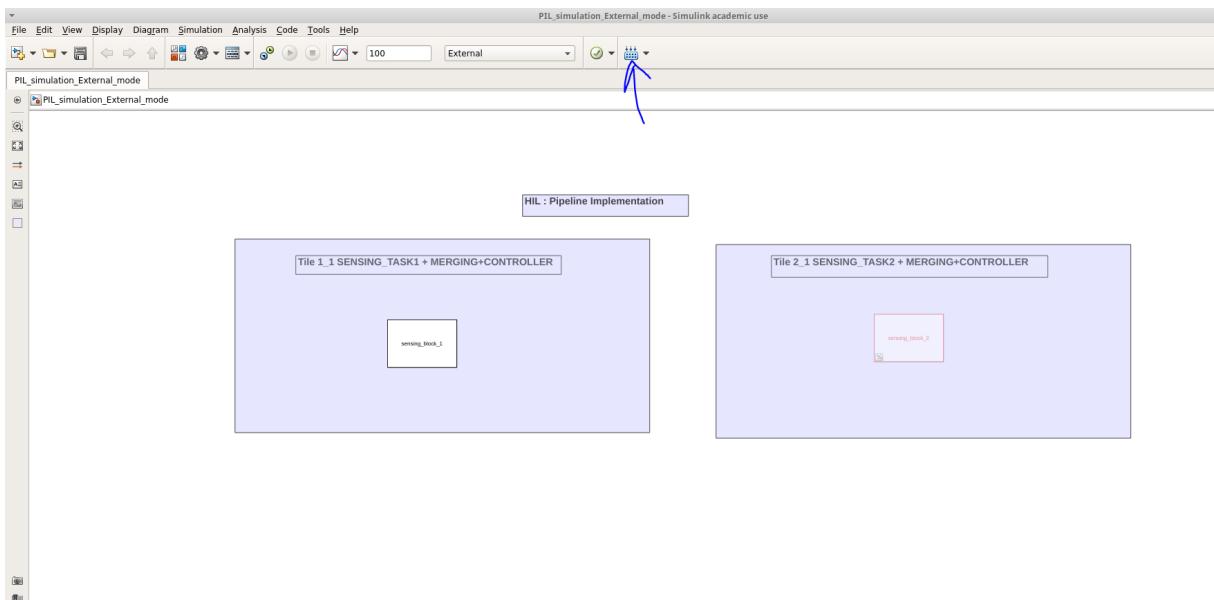
- Processor tile for PIL application: 1
- Virtual platform for PIL application: 1



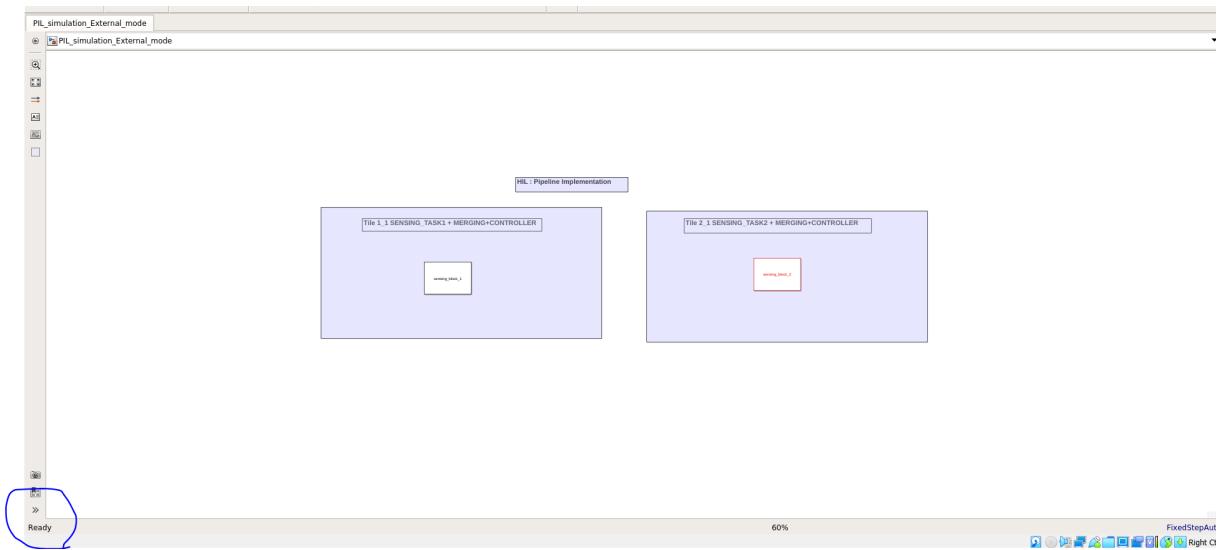
- open the two new terminal windows, connect the PYNQ board, and use the following command:
  - ensure that vep-config.txt is updated in the main CompSOC folder as per the required TDM scheduling. This step is essential before running the task in external mode on CompSOC.
  - vep-config file path:  
/home/computation/CompSOC\_ec\_target/CompSOC\_ec/+CompSOC\_ec/vep-config.txt
  - commands for running the sensing task on processor 1 and partition 1
    - Terminal 1 :
      - cd tutorial
      - cd monitoring-tools/
      - sudo ./channel Cheap\_Bridge 1 1 9878
    - Terminal 2 :
      - cd tutorial
      - ./readout.sh



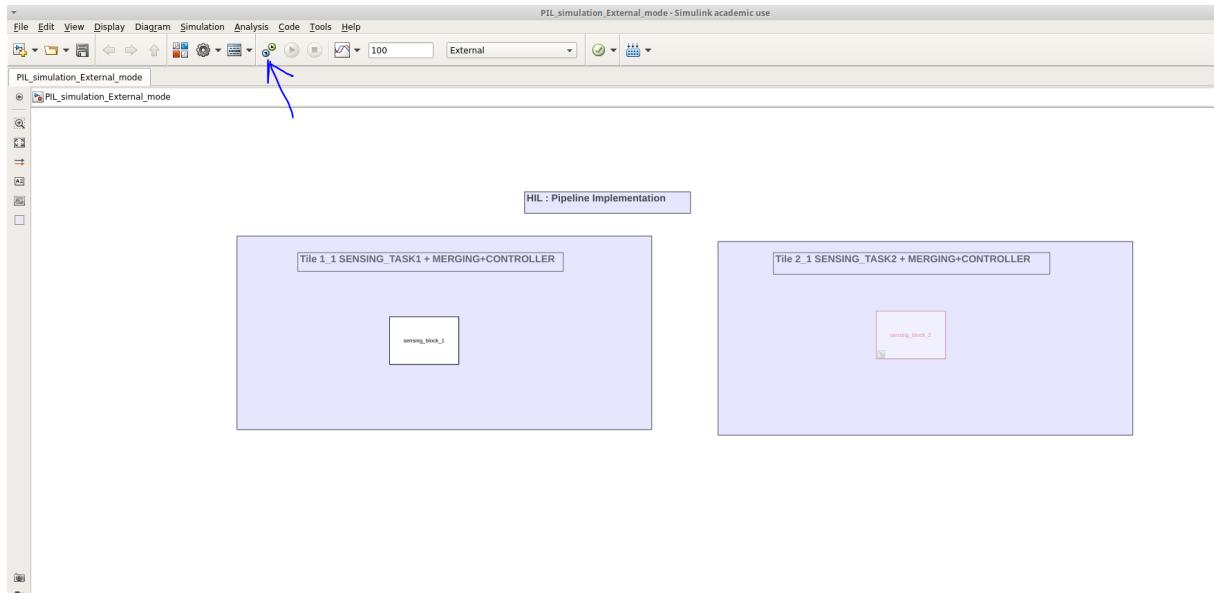
- click on the build button, as shown by the arrow in the below screenshot.



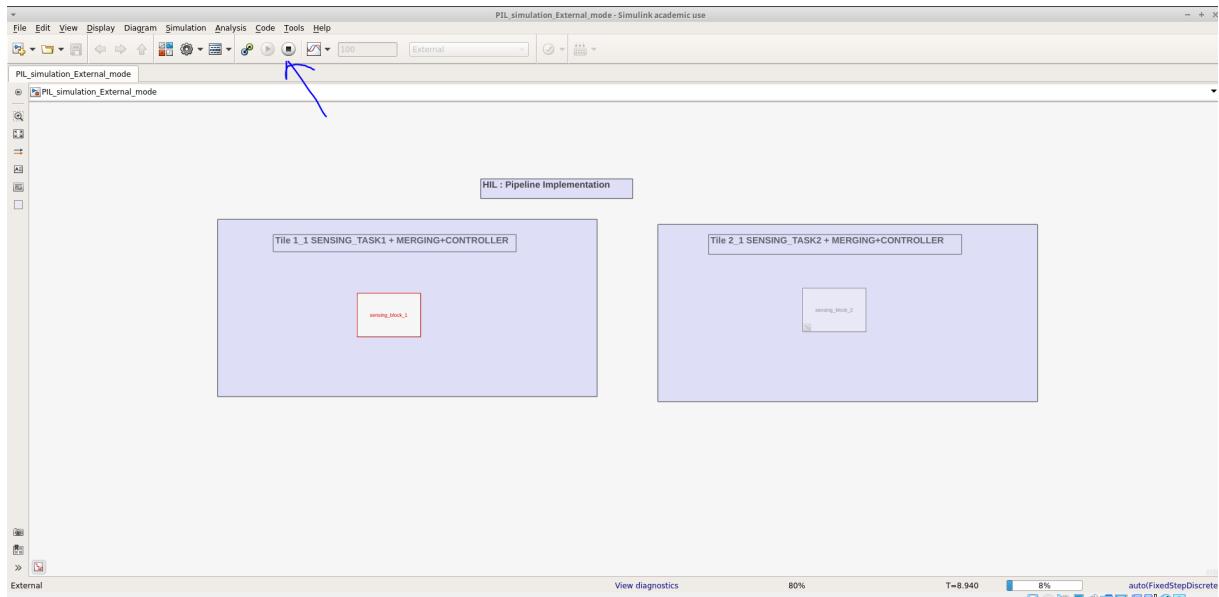
- after the build finishes, the Simulink model shows ready status; check the circle in the following screenshot.



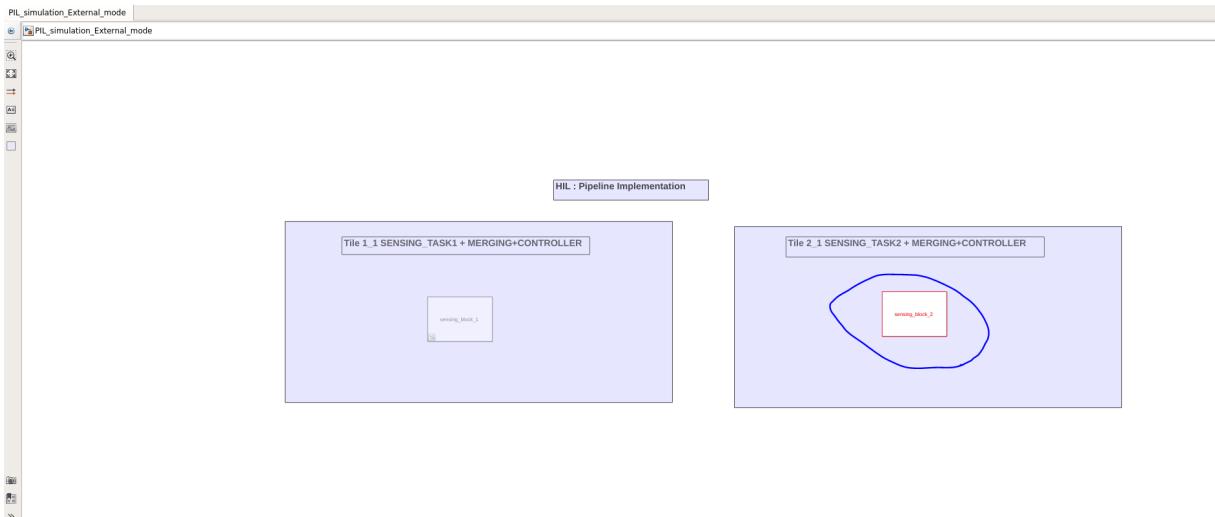
- if the build finishes successfully, ready status is available in the model. After that, run the task in external mode by clicking the connect to target button. Check the following screenshot for reference.



- after the above step, the (sensing task 1+merging+controller) connects to the target and starts running on the CompSOC; let it starts and runs for some time; check the highlighted circle in the below screenshot, then stop the running of the task by clicking on the stop button, check the arrow in the screenshot. No need to run the task for a long time.

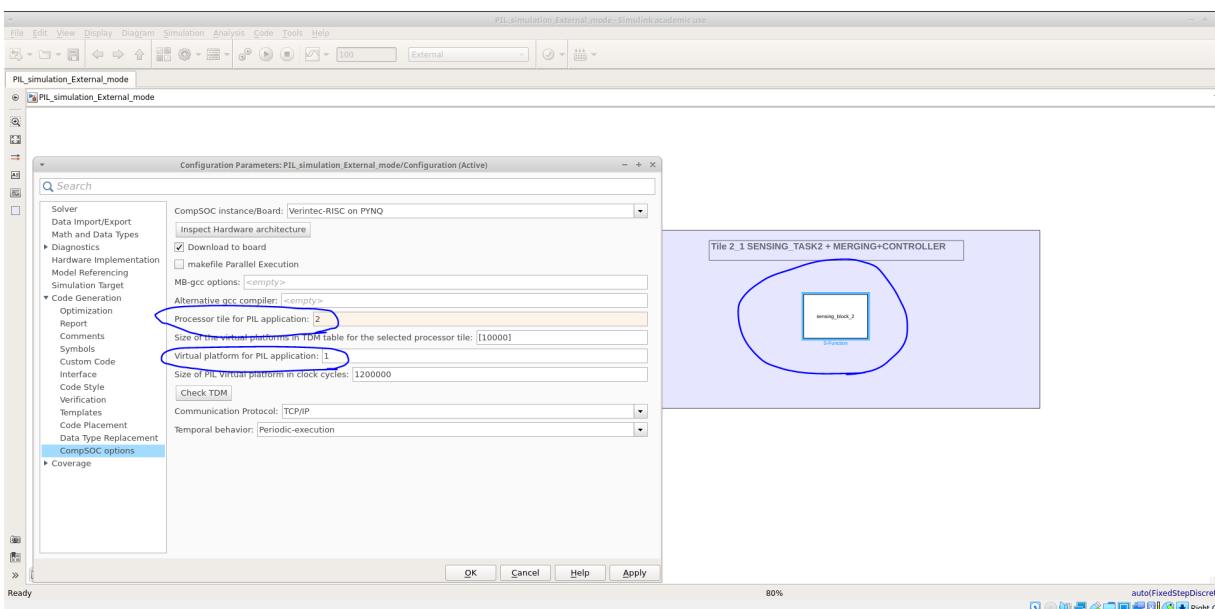


- move to the next task and comment all blocks except the one you want to run in external mode. Check the following screenshot; we have considered the (sensing task 2 + merging + controller tasks). Only the (sensing task 2 + merging + controller tasks) block is uncommented. Other block kept commenting.



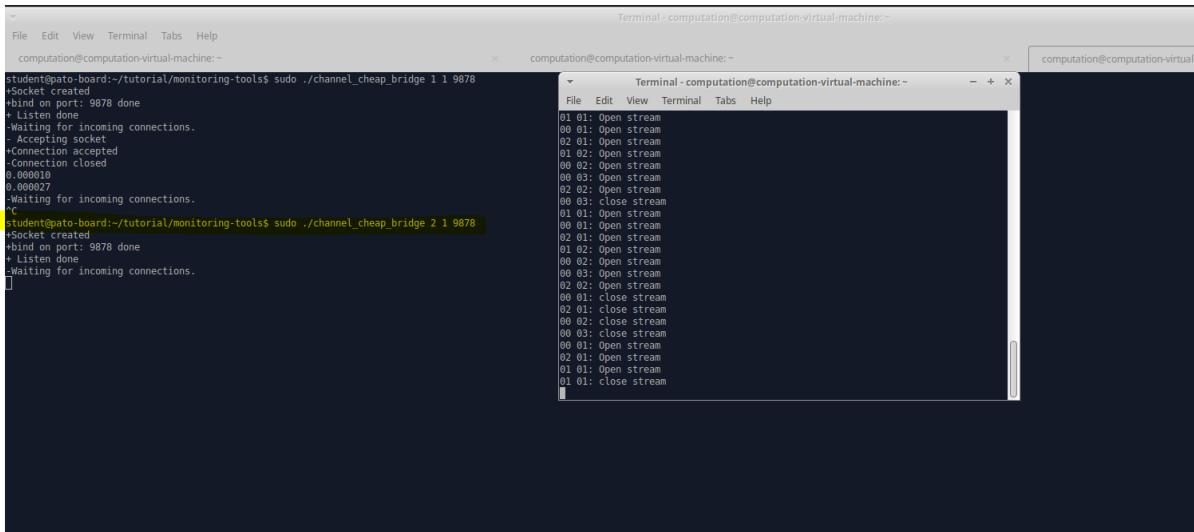
- open the configuration setting and check the required setting for the particular task. In this example, we run (sensing task 2 + merging + controller tasks) on processor 2 and partition 1. Therefore, consider the following changes :

- Processor tile for PIL application: 2
- Virtual platform for PIL application: 1



➤ open the two new terminal windows, connect the PYNQ board, and use the following command:

- ensure that vep-config.txt is updated in the main CompSOC folder as per the required TDM scheduling. This step is essential before running the task in external mode on CompSOC.
- vep-config file path:  
/home/computation/CompSOC\_ec\_target/CompSOC\_ec/+CompSOC\_ec/vep-config.txt
- commands for running the sensing task on processor 2 and partition 1
  - Terminal 1 :
    - cd tutorial
    - cd monitoring-tools/
    - sudo ./channel\_cheap\_bridge 2 1 9878
  - Terminal 2 :
    - cd tutorial
    - ./readout.sh



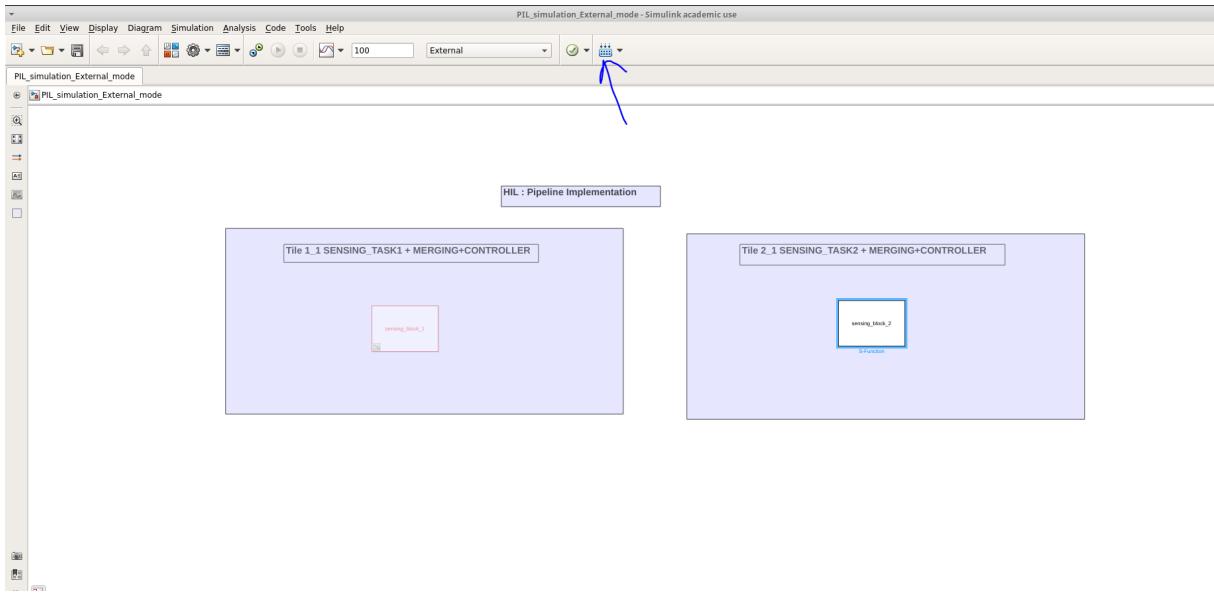
The screenshot shows two terminal windows side-by-side. The left terminal window is titled 'computation@computation-virtual-machine:~' and contains the following command output:

```
student@pto-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 1 1 9878
+Socket created
+bind on port: 9878 done
+Listen done
-Waiting for incoming connections.
- Accepting socket
+Connection accepted
-Connection closed
0.000010
0.000027
-Waiting for incoming connections.
r
student@pto-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 2 1 9878
+Socket created
+bind on port: 9878 done
+Listen done
-Waiting for incoming connections.
```

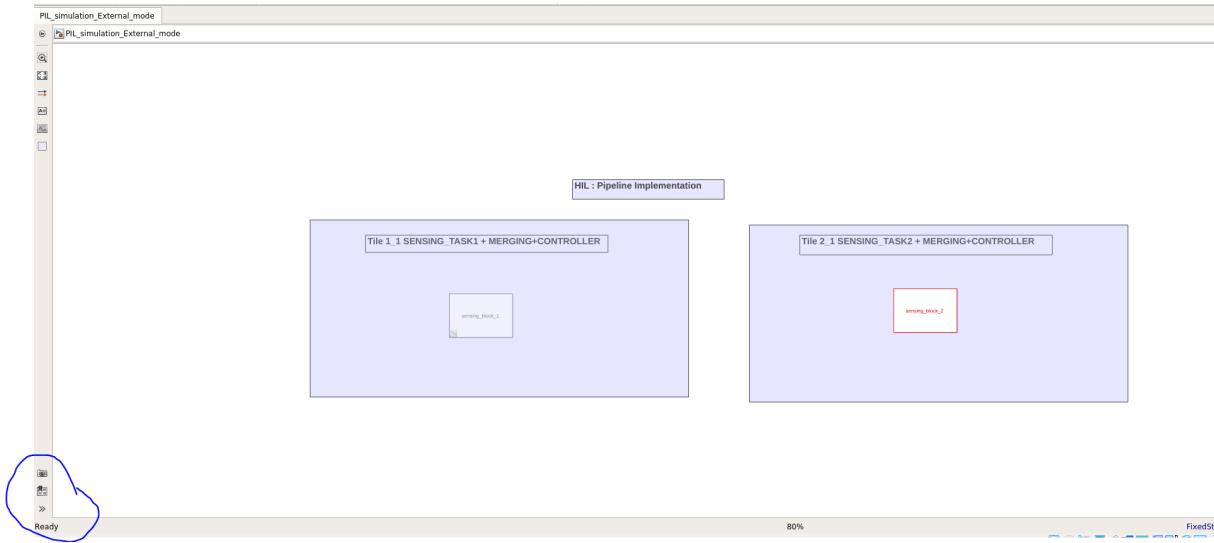
The right terminal window is also titled 'computation@computation-virtual-machine:~' and displays a continuous stream of log entries:

```
01 01: Open stream
00 01: Open stream
02 01: Open stream
01 02: Open stream
00 02: Open stream
00 03: Open stream
02 02: Open stream
01 01: Close stream
01 01: Open stream
00 01: Open stream
01 02: Open stream
00 02: Open stream
00 03: Open stream
02 02: Open stream
01 01: Close stream
02 01: Close stream
00 02: Close stream
00 03: Close stream
00 01: Open stream
02 01: Open stream
01 01: Open stream
01 01: Close stream
```

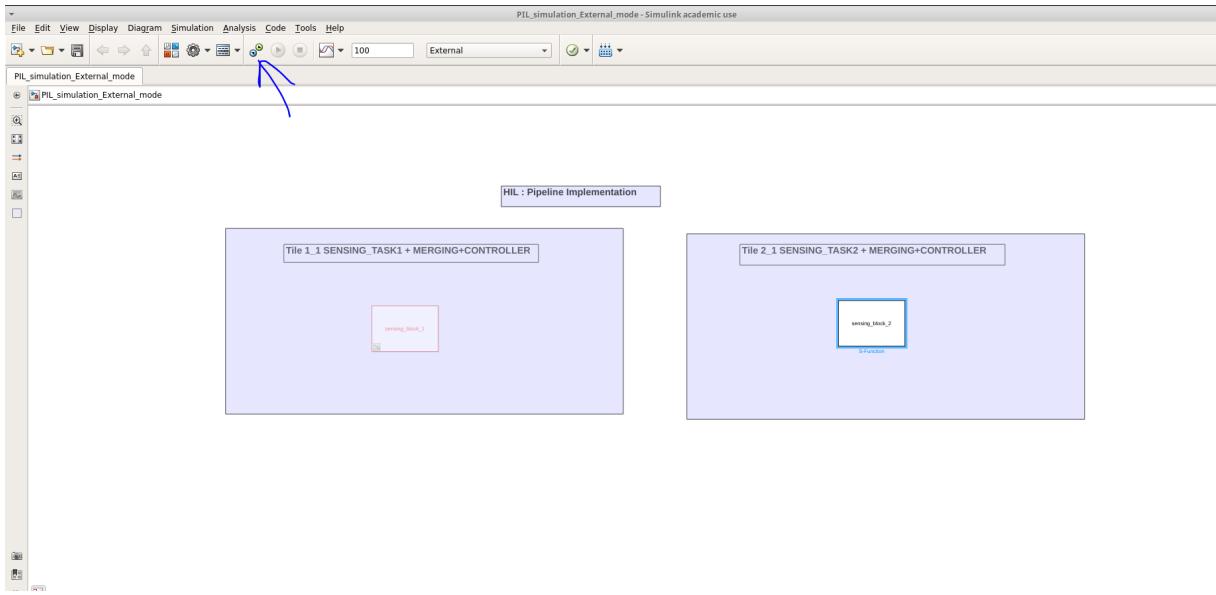
➤ click on the build button, as shown by the arrow in the below screenshot.



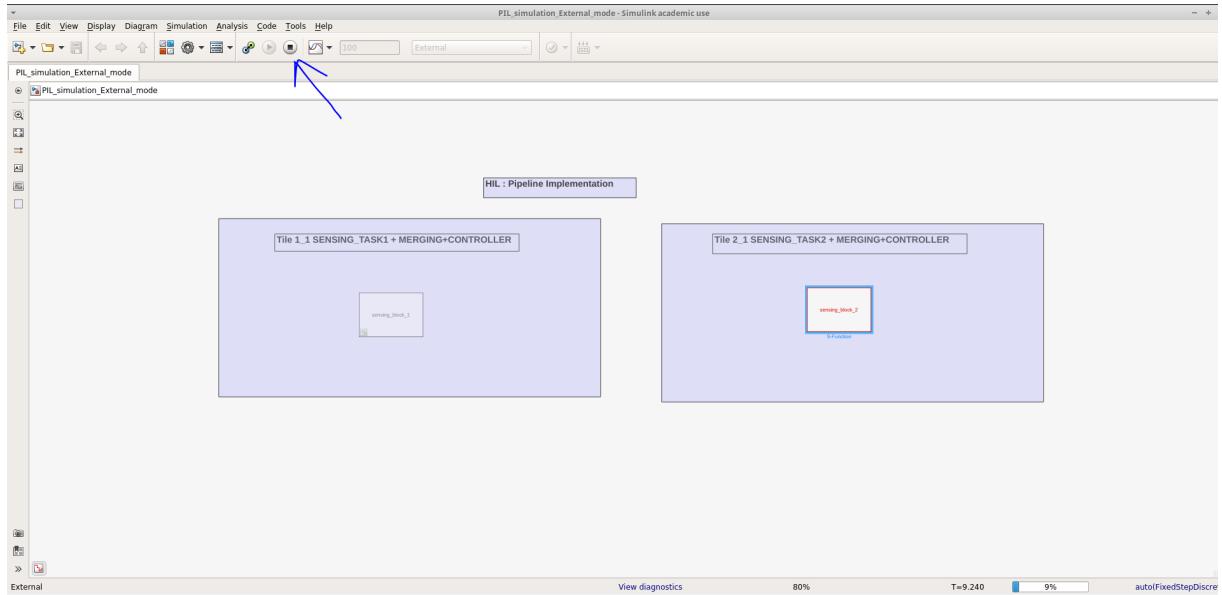
- after the build finishes, the Simulink model shows ready status; check the circle in the following screenshot.



- if the build finishes successfully, ready status is available in the model. After that, run the task in external mode by clicking the connect to target button. Check the following screenshot for reference.



- after the above step, the (sensing task 2 + merging + controller) connects to the target and starts running on the CompSOC; let it starts and runs for some time; check the highlighted circle in the below screenshot, then stop the running of the task by clicking on the stop button, check the arrow in the screenshot. No need to run the task for a long time.



- after completing all the steps, run the following highlighted command in the same terminal window opened for the previous steps. This step is required for running the final PIL simulation, which will be run from **PIL\_simulation\_monitoring\_app.slx** Simulink model.

- Terminal 1 :
  - cd tutorial
  - cd monitoring-tools/
  - sudo ./channel\_cheap\_bridge 0 1 9878
- Terminal 2 :
  - cd tutorial
  - ./readout.sh

The screenshot shows three terminal windows side-by-side. The leftmost terminal has the following output:

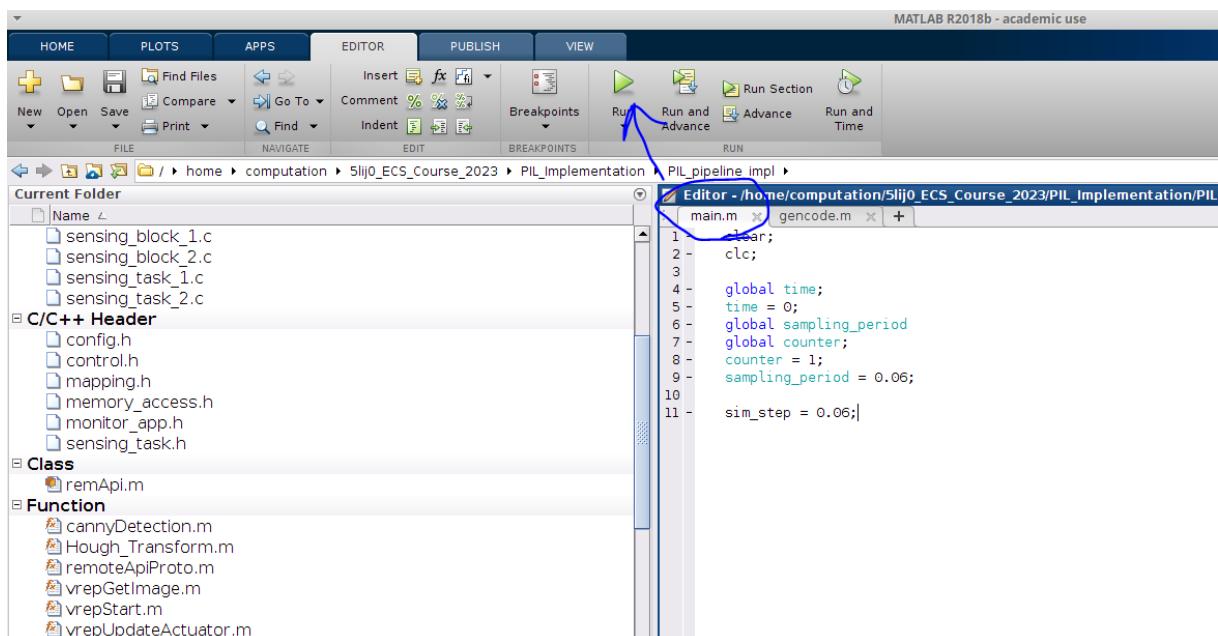
```
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 1 1 9878
+Socket created
+bind on port: 9878 done
+ Listen done
+Waiting for incoming connections.
- Accepting socket
+Connection accepted
-Connection closed
0.000010
0.000027
+Waiting for incoming connections.
:c
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 2 1 9878
+Socket created
+bind on port: 9878 done
+ Listen done
+Waiting for incoming connections.
- Accepting socket
+Connection accepted
-Connection closed
0.000011
0.000028
+Waiting for incoming connections.
:c
student@pato-board:~/tutorial/monitoring-tools$ sudo ./channel_cheap_bridge 0 1 9878
+Socket created
+bind on port: 9878 done
+ Listen done
+Waiting for incoming connections.
```

The middle terminal has the following output:

```
computation@computation-virtual-machine: ~
Terminal - computation@computation-virtual-machine: ~
File Edit View Terminal Tabs Help
00 02: Open stream
00 03: Open stream
02 02: Open stream
02 03: Open stream
00 03: close stream
01 01: Open stream
00 01: Open stream
02 01: Open stream
01 01: Open stream
00 03: Open stream
00 03: Open stream
02 02: Open stream
00 01: close stream
02 01: close stream
00 03: close stream
00 03: close stream
00 01: Open stream
02 01: Open stream
01 01: Open stream
01 01: close stream
01 01: Open stream
00 01: Open stream
02 01: Open stream
02 01: close stream
```

The rightmost terminal is empty.

- next step is to run the main.m file and check the CoppeliaSim software; the mass-damper scene should be opened and ensure it is not running automatically; if it is running, stop it.



- open the **PIL\_simulation\_monitoring\_app.slx** Simulink model, and click on the Run button. This will start running the simulation. It will take some time to simulate as it initializes all the

required settings for running the simulation. So wait until you see the running status. Check the following screenshots for reference.

- check the current position returned from the monitoring app using the scope linked to the monitoring app PIL block. Also, visualize the output in CoppeliaSim.

