How to autocode a HEV model block?

In this document, you will find how to autocode one block of the HEV global model. To make understand how it is done, we took as an example the Engine Representation block. You will find this block by opening HEV model→Rear Axle→Electrical Machine. Copy this block on a new Simulink file. Replace all the “from” by Inputs and “goto” by Outputs. Beware of the order and the name of every data. So the original block which is shown Figure 2: Engine_Representation become on your new Simulink file Figure 1: New simulink file

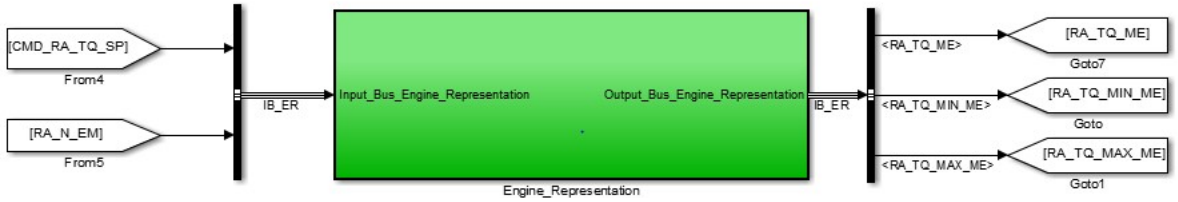


Figure 2: Engine_Representation

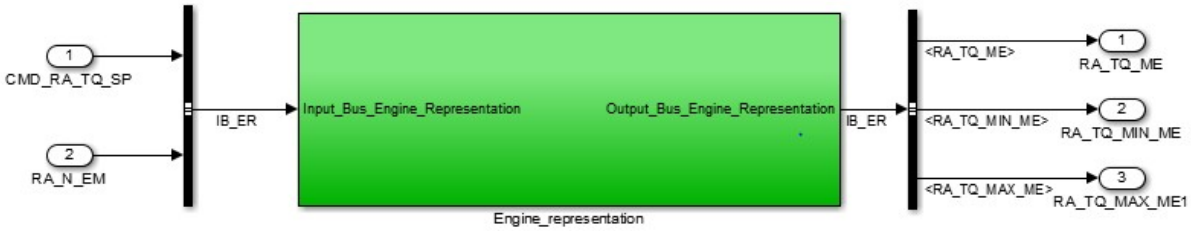
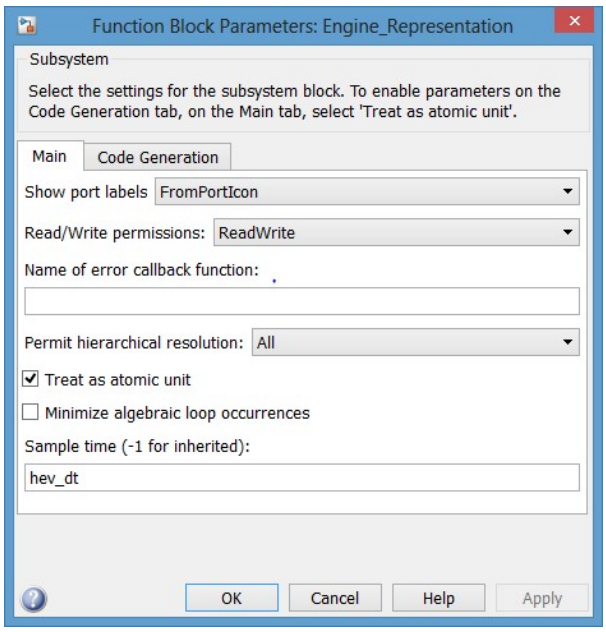


Figure 1: New simulink file

When you follow the Code Generation Tutorial to autocode this block you will get some errors saying that: the block should be treat as atomic.

1. Atomic Unit

This happened because in Simulink all the blocks can be treated at the same time. So when Simulink is simulating a block, if the treatment is not finished in this block, it can go and treat another block. But to simulate the HEV behaviour, we want a step by step (block after block) simulation. To do so we have to set all the blocks that are going to be autocoded as “atomic” so they will be treated one after another. For example with the Engine Representation block, you have to: Click Right on the block → Go on Block parameters →Tick “treat as atomic bloc”.



You also have to specify the sample time as the same you put in the Solver parameters.

Figure 3: Atomic block

2. Data Export/Import

In the Tutorial Code Generation, while autocoding we specified that there were data (inputs and outputs) which were coming from and going to the Matlab workspace. It was because we were autocoding the only block of the Simulink file. But now, we've got a lot of blocks for the HEV and we want to autocode the small ones to recreate the global model behaviour, you don't have to take in count this part, just continue the Tutorial. There is nothing which is ticked.

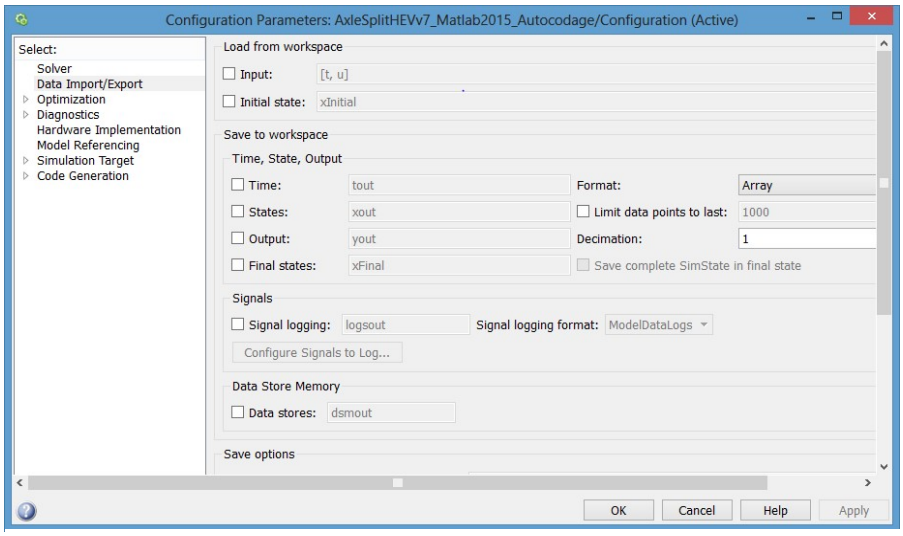
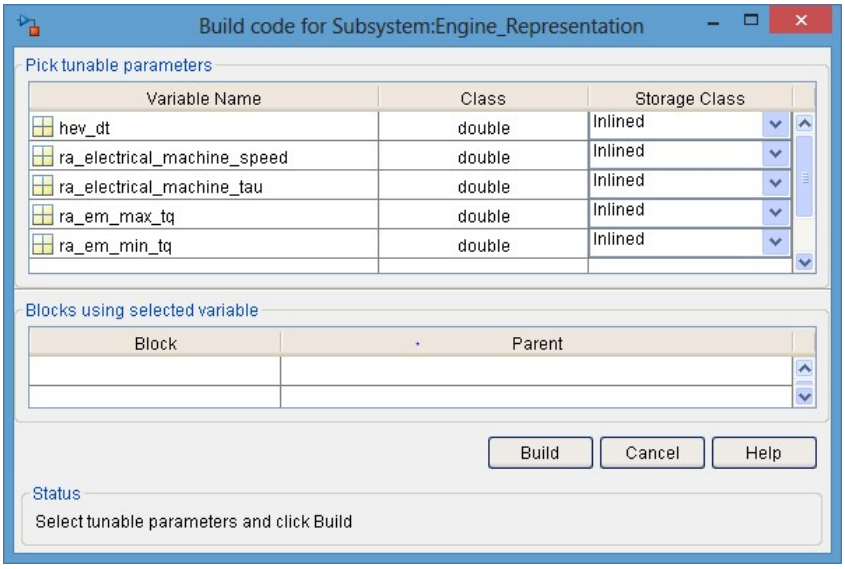


Figure 4: Data Import/Export

3. Code Generation

When all the Code Generation parameters are set, to generate the code of a block you have to:

Click Right on the block→C/C++ Code→Build this subsystem.



Then the following window appears. Finally click on "Build" to generate the block code.

Figure 5: Build code for subsystem

4. Test the autocode

To validate the generated code you have to follow these instructions step by step.

4.1. The data recovery

To test the code you have to compare it with the Simulink model results so you have to recover and save them. To do so, you have to put in your model on input and output a "To workspace" in order to recover all the data on the workspace.

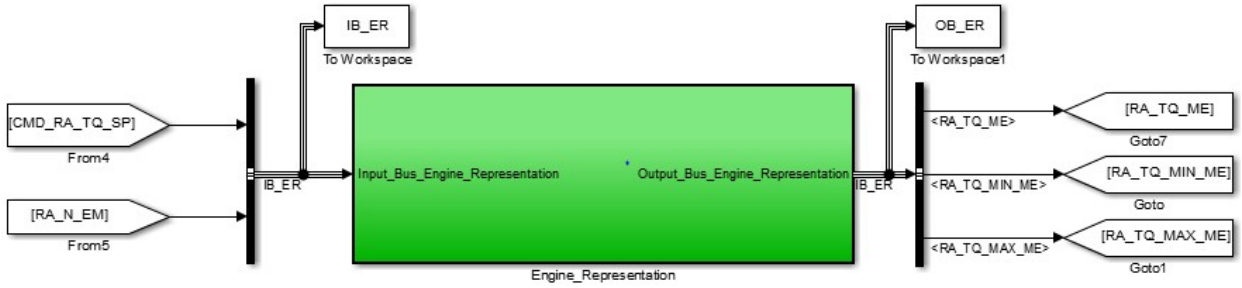


Figure 6: Data recovery in the matlab workspace

Then "Run" the global model to find your data in the workspace.

Finally, for every single input or output, create a text file with the input or output name to save all the data. While saving, **beware of replacing the "comma" by "point".** For example with the Engine Representation, we created 5 files: 2 inputs and 3 output as you can see on the previous figure. Here the text file of the 3rd output RA_TQ_MAX_ME with the points in numbers.

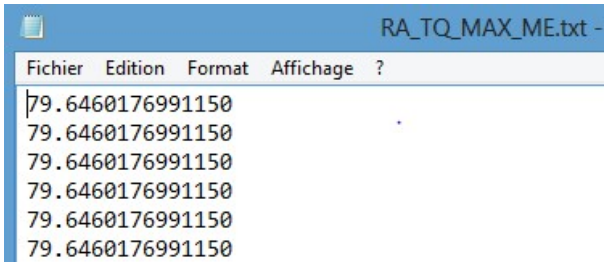


Figure 7: Text File with points

To be more organized, we create the C++ test project the following way.

4.2. C++ PROJECT

In the "Autocodage" folder, create a folder with the name of the block that you are autocoding.

All the work will be done in this folder. For our example is "Engine_Representation".

Create a C++ project in this folder and add all the .cpp and .h files generated by the autocode.

Create a matlab script to test the generated code in this folder.

Finally for the text files, create 3 folders: One for the Inputs, one for the Code Outputs and one for the Simulink outputs.

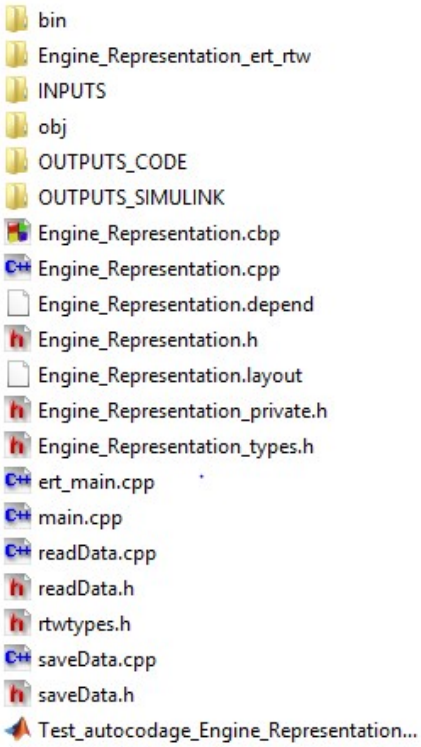


Figure 8: Block folder

You should have a folder like this one for the Engine Representation block.

4.3. THE MAIN

To be more understandable, we will explain the code directly by putting comments on every useful lines. These comment will be in red.

```
// File: ert_main.cpp
// Code generated for Simulink model 'Engine_Representation'. The model we autocoded
// Model version                  : 1.1798
// Simulink Coder version         : 8.8 (R2015a) 09-Feb-2015
// C/C++ source code generated on : Tue Jan 23 15:26:15 2018
// Target selection: ert.tlc
// Embedded hardware selection: 32-bit Generic
// Code generation objectives: Unspecified
// Validation result: Not run
#include <stddef.h>
#include <stdio.h>                // This ert_main.c example uses printf/fflush
#include "Engine_Representation.h"    Model's header file
#include "rtwtypes.h"
```

```
#include "saveData.h"  The saveData library
#include "readData.h"   The readData library

static Engine_RepresentationModelClass Engine;// Creating of an Engine_Representation object named “Engine”

// Associating rt_OneStep with a real-time clock or interrupt service routine
// is what makes the generated code "real-time".  The function rt_OneStep is
// always associated with the base rate of the model.  Subrates are managed
// by the base rate from inside the generated code.  Enabling/disabling
// interrupts and floating point context switches are target specific.  This
// example code indicates where these should take place relative to executing
// the generated code step function.  Overrun behavior should be tailored to
// your application needs.  This example simply sets an error status in the
// real-time model and returns from rt_OneStep.
//
void rt_OneStep(void);
void rt_OneStep(void)
{
  static boolean_T OverrunFlag = false;
  // Disable interrupts here
  // Check for overrun
  if (OverrunFlag) {
    rtmSetErrorStatus(Engine.getRTM(), "Overrun"); Beware to put the correct object name
    return;
  }
  OverrunFlag = true;
  // Save FPU context here (if necessary)
  // Re-enable timer or interrupt here
  // Set model inputs here
  // Step the model
  Engine.step();

  // Get model outputs here
```

```
    // Indicate task complete
    OverrunFlag = false;
    // Disable interrupts here
    // Restore FPU context here (if necessary)
    // Enable interrupts here
  }
  // The example "main" function illustrates what is required by your
  // application code to initialize, execute, and terminate the generated code.
  // Attaching rt_OneStep to a real-time clock is target specific.  This example
  // illustates how you do this relative to initializing the model.

  int_T main(int_T argc, const char *argv[])
  {
    // Unused arguments
    (void)(argc);
    (void)(argv);
    // Initialize model
    Engine.initialize();
    int taille= 20001; The size of the tabs

   //Creation of the Inputs tabs
    long double CMD_RA_TQ_SP[taille]; 1st input of Engine_Representation block
    long double RA_N_EM[taille]; 2nd  input of Engine_Representation block

    //Creation of the Outputs tabs
    long double RA_TQ_ME[taille]; 1st output of Engine_Representation block
    long double RA_TQ_MIN_ME[taille]; 2nd output of Engine_Representation block
    long double RA_TQ_MAX_ME[taille]; 3rd output of Engine_Representation block

    //initialization of the inputs with readData
    readData(CMD_RA_TQ_SP,"INPUTS/CMD_RA_TQ_SP.txt"); Taking data from the correct text
file
    readData(RA_N_EM,"INPUTS/RA_N_EM.txt");
```

```
    // Attach rt_OneStep to a timer or interrupt service routine with
    //  period 0.01 seconds (the model's base sample time) here.  The
    //  call syntax for rt_OneStep is
    //   rt_OneStep();
    for (int i=0; i < 20001;i++)
    {
      Engine.Engine_Representation_U.CMD_RA_TQ_SP=   CMD_RA_TQ_SP[i];//Putting  the  first
input in the object Engine
      Engine.Engine_Representation_U.RA_N_EM= RA_N_EM[i]; ];//Putting the second input in the
object Engine
      rt_OneStep();//processing in the block for the ith inputs
      //Filling the output tabs with the correct output from the code
      RA_TQ_ME[i]=Engine.Engine_Representation_Y.RA_TQ_ME;
      RA_TQ_MIN_ME[i]=Engine.Engine_Representation_Y.RA_TQ_MIN_ME;
      RA_TQ_MAX_ME[i]=Engine.Engine_Representation_Y.RA_TQ_MAX_ME;
    }
   //Saving all the code outputs in the correct text file with saveData
    saveData(taille,RA_TQ_ME,"OUTPUTS_CODE/RA_TQ_ME_code.txt");
    saveData(taille,RA_TQ_MIN_ME,"OUTPUTS_CODE/RA_TQ_MIN_ME_code.txt");
    saveData(taille,RA_TQ_MAX_ME,"OUTPUTS_CODE/RA_TQ_MAX_ME_code.txt");
    printf("autocoding done"); //Display this message when the test is finished

    fflush((NULL));
    while (rtmGetErrorStatus(Engine.getRTM()) == (NULL)) {
      //  Perform other application tasks here
    }
    // Disable rt_OneStep() here
    // Terminate model
    Engine.terminate();//The block function is finished
    return 0;
   }
   //
   // File trailer for generated code.
```

Finally we test all the results on Matlab with the script. This script is explained in another technical document.

Here is the results for the 3 outputs of the Engine Representation block.

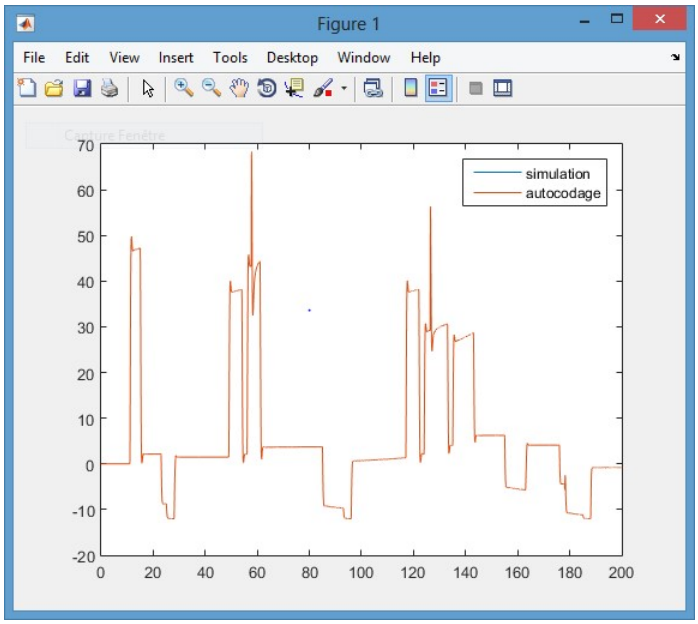


Figure 10: RA_TQ_ME

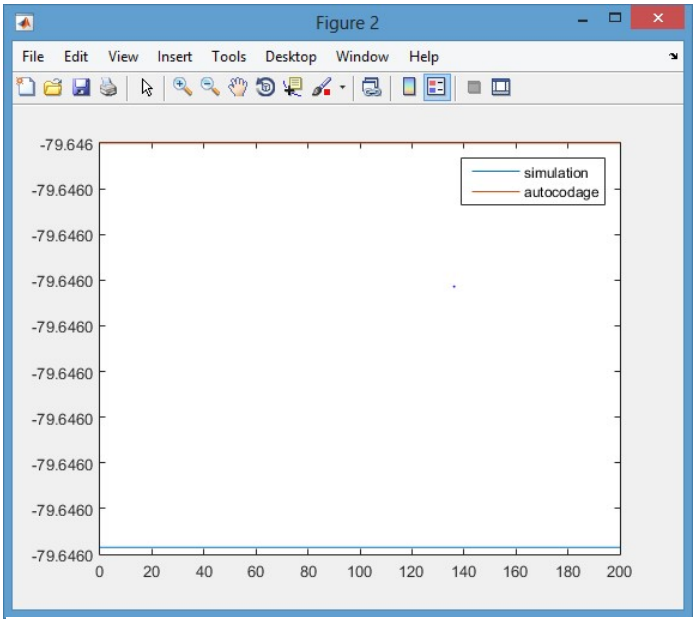


Figure 11:RA_TQ_MIN_ME

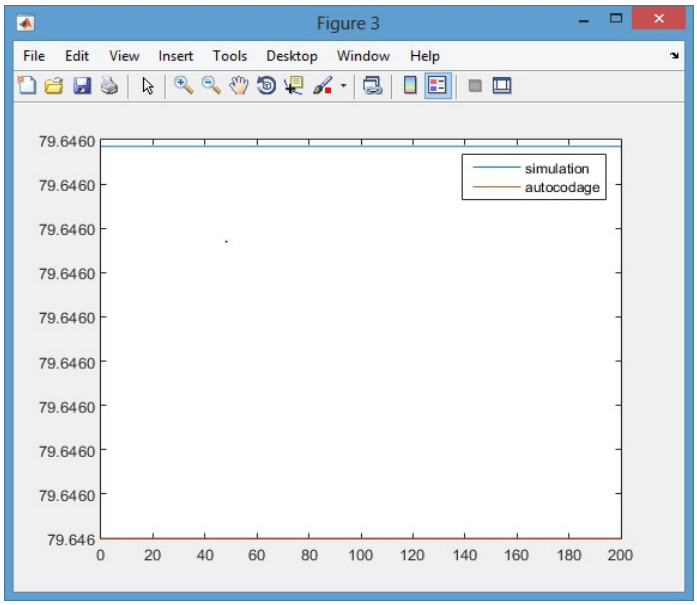


Figure 9: RA_TQ_MAX_ME

We can see that the generated code results are the same than the Simulink ones. To be more precise, we also plotted the error between each output and we obtained the following figure:

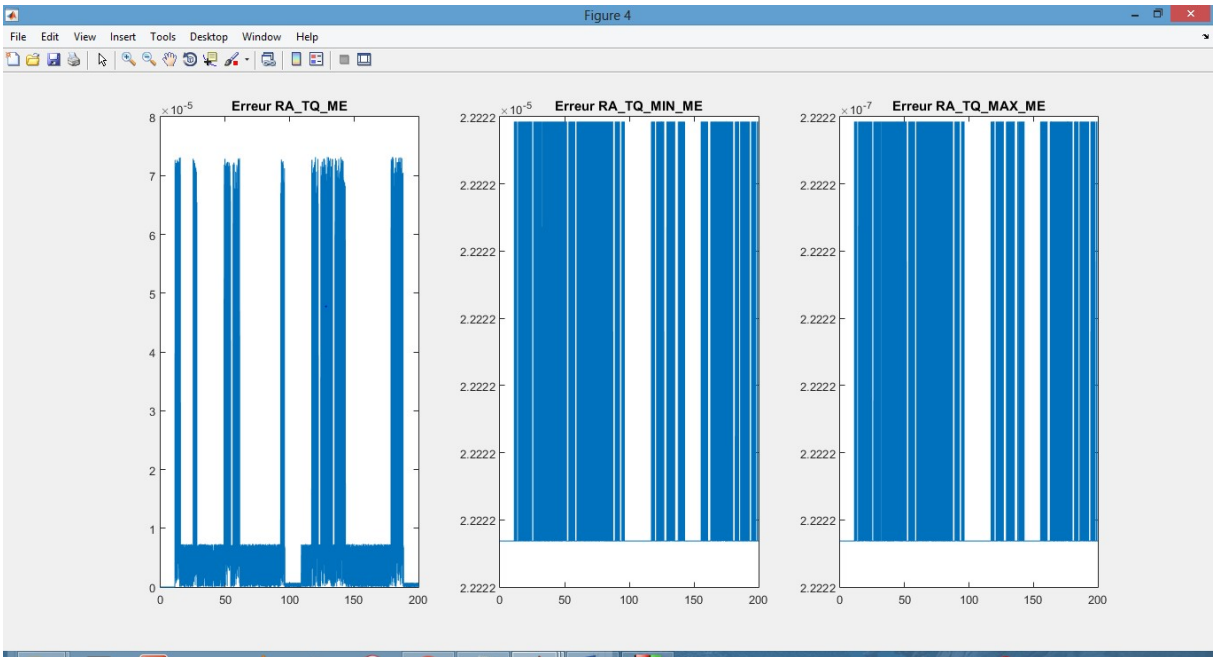


Figure 12: Output errors

We've got a little errors (10e-5) because of it's not possible to copy the numbers from Matlab to the text files with all the digits.

5. How to solve running errors on CodeBlocks

Sometimes when you launch your C++ project on CodeBlocs, the executable file crashes. This error is due to memory assignation on tabs. To solve this error you need to implement the following instructions:

- Add at the beginning of the program the following line "#include <vector>"
- Replace all the outputs by the following line : "vector<long double> MY_OUTPUT(taille);"
- You also have to change the "saveData.cpp" and the "saveData.h" file. The only modifications you need to do is to add the line "#include <vector>" on both files and replace in the function definition "long double tableau[TAILLEMAX]" by "vector<long double> tableau".