Autocoding test of a two order model

We have seen in the Code Generation Tutorial that the Autocoding works quiet well. But we may not truly understand all the code generated.

Knowing that for the project we will autocode a lot of blocks and program a main in order to simulate the complete model behaviour, we have to learn how to combine all the generated codes.

That is why, we decide to simulate a two order model presented below by re-using the codes generated for the first order model.

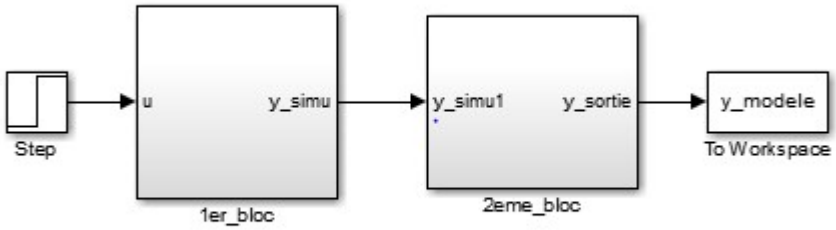

Figure 1: Two order model

The two blocks are the same except the gain Ka (block 1) and Kb (block 2).

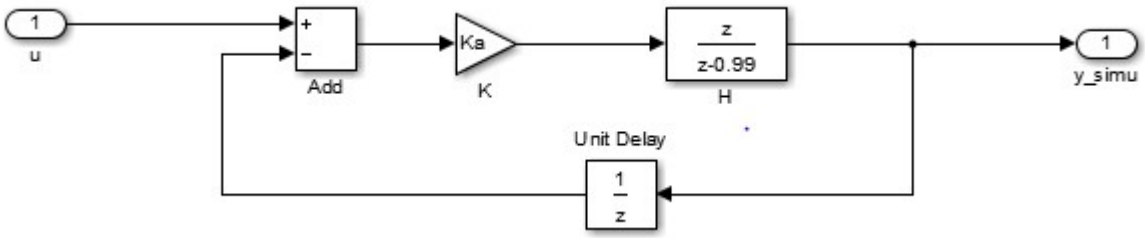

Figure 2: First order model _ Block 1

In the Tutorial, we autocoded the block 1. And as presented on the Simulink model, the two order model will take as an input “u” which is actually the block 1 input and as an output “y_sortie” which corresponds to the block 2 output.

How do we succeed to combine the blocks?

I. Creation of the project

Firstly, we create a C++ project on Code Blocks as shown in the Tutorial. Then we add the files of the autocoded block as you can see in the following figure:

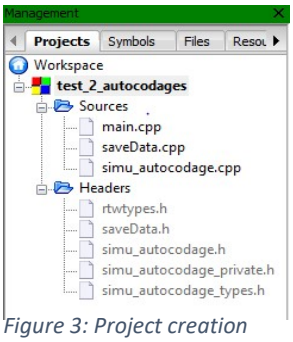

Figure 3: Project creation

II. The Main

Autocode a model is to create a class which contains all the functions and all the variables of this model.

So in our case, when we autocoded the First order model, we actually create a class of first order models in which you can change the gain. The name “simu_autocodage” is because we named the Simulink file like this.

The “simu_autocodage.h” file is a header which contains the declaration of all the functions and all the variables as you can see on the figure below:

```
// Class declaration for model simu_autocodage
class simu_autocodageModelClass {
  // public data and function members
 public:
  // External inputs
  ExtU_simu_autocodage_T simu_autocodage_U;

  // External outputs
  ExtY_simu_autocodage_T simu_autocodage_Y;

  // Model entry point functions

  // model initialize function
  void initialize();

  // model step function
  void step();

  // model terminate function
  void terminate();

  // Constructor
  simu_autocodageModelClass();

  // Destructor
  ~simu_autocodageModelClass();

  // Real-Time Model get method
  RT_MODEL_simu_autocodage_T * getRTM();
```

```
// Block states (auto storage) for syst
typedef struct {
  real_T Delay_DSTATE;
  real_T DiscreteZeroPole_DSTATE;
} DW_simu_autocodage_T;

// External inputs (root inport signals
typedef struct {
  real_T In1;
} ExtU_simu_autocodage_T;

// External outputs (root outports fed
typedef struct {
  real_T Out1;
} ExtY_simu_autocodage_T;

// Parameters (auto storage)
struct P_simu_autocodage_T_ {
  real_T Ka;


};
```

Figure 4: Simu_autocodage header

According to this figure, it appears that the input and the output are in a structure. Which means that if you have to autocode a block with many inputs (or outputs), you have to use buses in order to have only one structure with all the inputs (or outputs).

The “simu_autocodage.cpp” contains the Initialization function, the Class basic functions (constructor, destructor) and the function realized in the block which presented in the following

```
// Model step function
void simu_autocodageModelClass::step()
{
  // local block i/o variables
  real_T rtb_Gain;

  // Gain: '<Root>/Gain' incorporates:
  //   Delay: '<Root>/Delay'
  //   Inport: '<Root>/In1'
  //   Sum: '<Root>/Sum'

  rtb_Gain = (simu_autocodage_U.In1 - simu_autocodage_DW.Delay_DSTATE) *
    simu_autocodage_P.Ka;

  // DiscreteZeroPole: '<Root>/Discrete Zero-Pole'
  {
    simu_autocodage_Y.Out1 = 1.0*rtb_Gain;
    simu_autocodage_Y.Out1 += 0.99*simu_autocodage_DW.DiscreteZeroPole_DSTATE;
  }

  // Update for Delay: '<Root>/Delay'
  simu_autocodage_DW.Delay_DSTATE = simu_autocodage_Y.Out1;

  // Update for DiscreteZeroPole: '<Root>/Discrete Zero-Pole'
  {
    simu_autocodage_DW.DiscreteZeroPole_DSTATE = rtb_Gain + 0.99*
      simu_autocodage_DW.DiscreteZeroPole_DSTATE;
  }
}
```

Figure 5: The block function

figure:

With all this, a block is just an object of the Class that we autocoded. In our case, to realize the two

```
static simu_autocodageModelClass bloc1;
static simu_autocodageModelClass bloc2;
```

order model we started by creating two objects of the simu_autocodage Model Class:

We also duplicated for each block, the function which allows to advance in the block:

As we said previously, the input “u” is the block 1 input and the output “y_sortie” is the block 2

```
void rt_OneStep1();
void rt_OneStep2();
```

output. So we did this code in the main:

```
// Initialize model
bloc1.initialize();
bloc2.initialize();
float entree=1.0;
float sortie1;
const int taille = 150; //length for the output and input vectors you can choose wh
bloc1.simu_autocodage_U.In1=entree; //The input is the block1 input
double out_tab[taille];
double inp_tab[taille];
printf("Simulation du modèle pour une entree = %f\n",bloc1.simu_autocodage_U.In1);
//creation of output and input vectors

//filling of vectors
for(int i=0 ; i<taille ;i++)
{
rt_OneStep1();//Advancing in the block1
inp_tab[i]=bloc1.simu_autocodage_U.In1;//Fulling the Input tab
sortie1=bloc1.simu_autocodage_Y.Out1;//Getting the block1 output

bloc2.simu_autocodage_U.In1=sortie1;//which is the block2 input
rt_OneStep2();//Advancing in the block2
out_tab[i]=bloc2.simu_autocodage_Y.Out1;//Getting the global output
}
printf("simu_autocodage_Y.Output = %f\n",bloc2.simu_autocodage_Y.Out1);
saveData(taille,inp_tab,"Entree.txt");//Looading the input in the text file
saveData(taille,out_tab,"Sortie.txt");//Loading the output in the text file
```

Figure 6: Main of the second order model

Now that we know how to use properly the code generated by the autocoding we can now proceed for the block listed in the Architecture Document.

The main representing the behaviour of our system will be coding according to the sequence diagram.