Tutorial: Autogen

- **Goal:** Generate Autogen code based on Simulink model.
- **Complexity:** 3 / 5
- **Estimated Time:** 40 min

This tutorial goes over:

- Organization of your repository to store the Simulink Autogen codebase
- Creation a Simulink simulation model to generate Autogen code
- Control implementation using the AMDC hardware

Tutorial Requirements

1. Working AMDC hardware
2. Completion of the "Voltage Source Inverter" tutorial
3. Read "Control with AMDC Using Simulink Autogen" article to understand an overview of Autogen

File Organization

The first step is to organize your repository. Create a new `modeling/simulink` folder and an `autogen` folder in your repository, as shown below:

```
my-AMDC-workspace/               <= master repo
|-- modeling/
|    |-- simulink/               <= Now create this folder
|-- control/
     |-- AMDC-Firmware/          <= AMDC-Firmware as submodule
     |-- my-AMDC-private-C-code/ <= Your private user C code
         |-- usr/
             | -- controller/    <= Your private user app
                 | -- autogen/   <= Now create this folder
```

Install Required MATLAB/Simulink Toolbox

To develop control code using Simulink Autogen, the following software components are required:

- Simulink Coder
- Embedded Coder

Additional toolboxes may be required depending on the specific control design.

Create a Simulink Model

Now that you create a Simulink model that will be used to generate Autogen code. In this example, you will replace the C code in the VSI app developed in the "Voltage Source Inverter" tutorial with Autogen code to calculate the duty ratios.

1. In `simulink` folder, create a new MATLAB file named `setup.m`.
2. In `setup.m`, copy-paste the following MATLAB code:

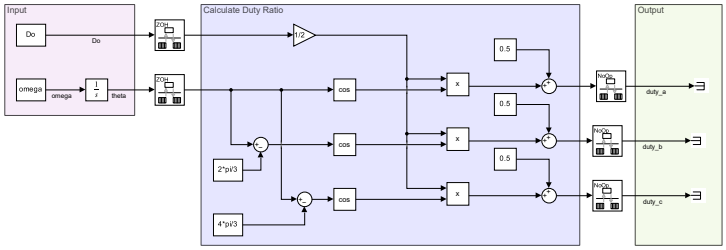
```
fs = 10e3;      % sampling frequency (Hz)
Ts = 1/fs;      % sampling time (sec)
Tsim = Ts/10;   % simulation time (s)

omega = 377.0;  % (rad/s)
Do    = 0.8;    % (--)
```

3. Open a blank model of Simulink, and save it as `setupModel.slx` in the `simulink` folder.
4. Add a `Constant` block and set its value to `omega`.
5. Add an `Integrator` block and connect it to the `Constant` block. Rename the output signal of this integrator to be `theta`.
6. Add a `Constant` block and set its value to `Do`.
7. Add two `Rate Transition` blocks and connect them to the `Integrator` and `Constant` blocks created above. In each `Rate Transition` block, set the sampling time `Ts`.
8. Create Simulink blocks to implement the following duty ratio calculation developed here:

```
// Calculate desired duty ratios
double duty_a = 0.5 + Do/2.0 * cos(theta);
double duty_b = 0.5 + Do/2.0 * cos(theta - 2.0*M_PI/3.0);
double duty_c = 0.5 + Do/2.0 * cos(theta - 4.0*M_PI/3.0);
```

9. Add a `Rate Transition` block after the duty ratio calculations. In these blocks, set the sampling time to `-1`.
10. The expected block diagram is shown below:

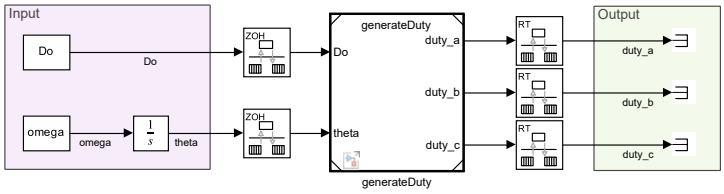


Model Setting

1. In `Modeling` tab, press `Model Settings` in `TOP MODEL` section.
    1. Under the `Solver` tree, in the `Solver Selection`, press `Fixed-step`.
    2. Set `Fixed-step-size` as `Tsim`.
2. Go to `Code Generation`.
    1. Click `Browse` for the `System target file`.
    2. Select `ert.tlc Embedded coder`.
    3. In the `Build process` section, check `Generate code only`.
3. Go to `Optimization` under `Code Generation`.
    1. Choose `None` for the `Leverage target hardware instruction set extensions` in the `Target specific optimizations`.
4. Go to `Templates` under `Code Generation`.
    1. Uncheck `Generate an example main program` in the `Custom templates` section.
5. Click `Apply` and `OK`.

Create a Referenced Model

1. Select the duty ratio calculation blocks, and right-click.
2. Select `Create Subsystem from Selection`.
3. Right-click on the subsystem created. Select `Block parameters (Subsystem)`, check `Treat as atomic unit`, and click `OK`.
4. Right-click on the subsystem and select `Subsystem & Model Reference`. Select `Convert` and click `Referenced Model ...`.
5. In the `Input Parameters` section, define the `New model name` as `generateDuty`.
6. Click `Apply` and `Convert`.
7. Rename the referenced model block to be `generateDuty`. The expected Simulink model is shown below:



Referenced Model Setting

1. Double-click the `generateDuty` referenced model and click `Model Settings` under `Modeling` tab.
2. Click `Model Settings` in the `REFERENCED MODEL` section.
    1. Under the `Solver` tree, set `Fixed-step-size` as `Ts`.
3. Save the Simulink file.

The example of Simulink file along with the referenced model is stored here. Note that this Simulink model was created using MATLAB R2024b.

Generate C-code

1. Open the `setup.m`.
2. Copy and paste the following code.

```
%% Autogen code for the controller
model='generateDuty';  % name of the controller to be built
slbuild(model);      % generates the Autogen code
oldFolder = cd('C:generateDuty_ert_rtw\');
% Copy only .c and .h files in autogen folder
command = 'for /r %i in (*.c, *.h) do copy /y %i ..\..\..\control\my-AMDC-private-C-code\usr\controll
[status, cmdout] = system(command);
cd(oldFolder);
```

3. Run the `setup.m` to generate C code. All autogenerated `.c` and `.h` files should be placed within `autogen` folder. Among these generated files, the most relevant are:

- `generateDuty.c` — implementation of the control logic
- `generateDuty.h` — interface definitions (inputs, outputs, function declarations)

These files define the controller as a callable function with input and output structures, consistent with the execution model described earlier.

4. Open the SDK, and make sure that the `autogen` folder appears under the `Project Explorer`. If it does not, refresh the project.

Integration with AMDC

Now, the user needs to update the user C code developed in the "Voltage Source Inverter" tutorial to incorporate the Autogen code generated from Simulink. Specifically, this requires modifying `task_controller_clear`, `task_controller_init`, and `task_controller_callback` functions. Within the callback function, the control task executes the developed code at a fixed sampling interval, and the following items need to be included:

1. Populate inputs, such as constant values or sampled sensor data. The input variables are defined with `_U` suffix (e.g., `generateDuty_U`).
2. Call the controller step function
3. Route outputs to actuators, such as PWM duty cycles. The output variables are defined with `_Y` suffix (e.g., `generateDuty_Y`).

The functions that need to be updated for this example are shown below:

`task_controller.c`:

```
// ...

int task_controller_clear(void)
{
  // ...

  // Clear state struct for Simulink controller
  memset(((void *) &generateDuty_DW_DW), 0, sizeof(DW_generateDuty_T));

  // ...
}

int task_controller_init(void)
{
  // ...

  // Initialize Autogen step
  generateDuty_initialize();

  // ...
}

double Ts    = 1.0 / (double) TASK_CONTROLLER_UPDATES_PER_SEC;
double theta = 0.0;    // [rad]
double omega = 377.0;  // [rad/s]
double Do    = 0.8;    // [--]

void task_controller_callback(void *arg)
{
  // ...

  // Update theta
  theta += (Ts * omega);

  // Wrap to 2*pi
  theta = fmod(theta, 2.0 * M_PI);

  // Populate inputs
  generateDuty_U.Do = Do;  // Inputs to controller as a constant value
  generateDuty_U.theta = theta;  // Inputs to controller as feedback

  // Call Autogen code
  generateDuty_step();

  // Calculate desired duty ratios
  double duty_a = generateDuty_Y.duty_a;  // Outputs from controller
  double duty_b = generateDuty_Y.duty_b;  // Outputs from controller
  double duty_c = generateDuty_Y.duty_c;  // Outputs from controller

  // Update PWM peripheral in FPGA
  pwm_set_duty(0, duty_a); // Set HB1 duty ratio (INV1, PWM1 and PWM2)
  pwm_set_duty(1, duty_b); // Set HB2 duty ratio (INV1, PWM3 and PWM4)
  pwm_set_duty(2, duty_c); // Set HB3 duty ratio (INV1, PWM5 and PWM6)

  // ...
}
```

Note

In this example, the input `theta` is generated within the code. However, this can be obtained from sensor measurements, such as encoder readings.

Running the AMDC

We are now ready to run the new control code with Autogen! Try running the control task by typing `ctrl init`. You should obtain the same results as in the "Voltage Source Inverter" tutorial.

Conclusion

Congratulations!

You have now built an updated user app with Autogen code for the voltage source inverter. These techniques can be extended for many other control problems, e.g., current regulation, motion control, or even more advanced controller. By modifying the Simulink model and continuously generating C code, you can effectively develop new control algorithms.