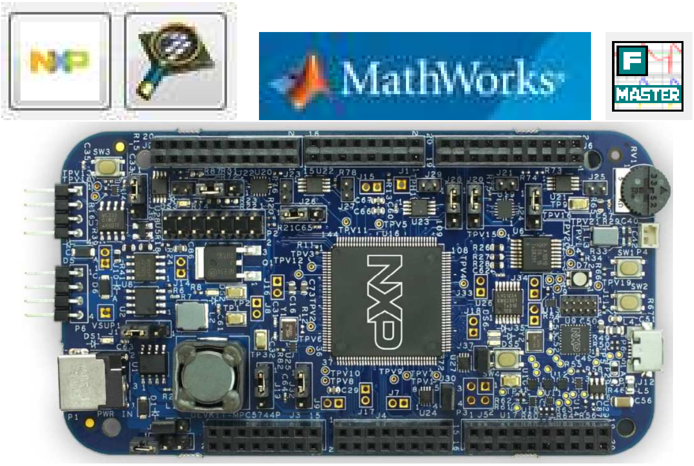
Introduction to software tools for Automotive Electronics lab



http://www.alexandre-boyer.fr

Alexandre Boyer
Patrick Tounsi

5^{e} année ESPE
October 2018

This document aims at providing basic support for the different software tools used to achieve the motor application project in Automotive Electronics lab:

- S32 Design Studio (S32DS) IDE: the integrated development environment provided by NXP for Power Architecture MCU
- Matlab / Simulink: As NXP provides Simulink toolbox for modeling, simulation and development of motor control application embedded on Power Architecture MCU, Matlab/Simulink can be used as both modeling and cross compilation tool for embedded application.
- FREEMASTER: NXP proposes this tool to communicate with embedded applications and monitor/visualize in real-time internal variables of embedded applications

In order to facilitate the development, simulation and debugging of motor control applications, NXP also provides:

- Model Based Design Toolbox (MBDT) for MPC574xP processor family: this Matlab / Simulink-compatible toolbox aims at configuring rapidly and modeling the internal blocks of MPC574xP MCU, and generating/downloading automatically the executable code.
- Automotive Math and Motor Control Library (AMMCLIB) for NXP MPC574xP: this library is compatible with both S32DS and Matlab/Simulink as a toolbox. It contains basic and complex mathematical functions dedicated to motor control applications (basic mathematical operation, digital filtering, Park transform, SVM, etc…)

The use of these different tools is not mandatory for the project. Actually, the design of motor control application could be done entirely in S32DS IDE or in Matlab Simulink, with or without using the MBDT and AMMCLIB. The purpose of this document is not to provide the design flow that you will use. It helps you to start development with these tools rapidly and underlines the functionality offered by these tools in order to help you to choose the more appropriate design flow.

This document is intended for motor control application development on MPC5744P microcontroller, mounted on the DEVKIT-MPC5744P development kit.

This document is not exhaustive. More information about the different tools, toolbox and library can be found in the references provided in the part Links.

I - Getting started S32 Design Studio for Power Architecture IDE

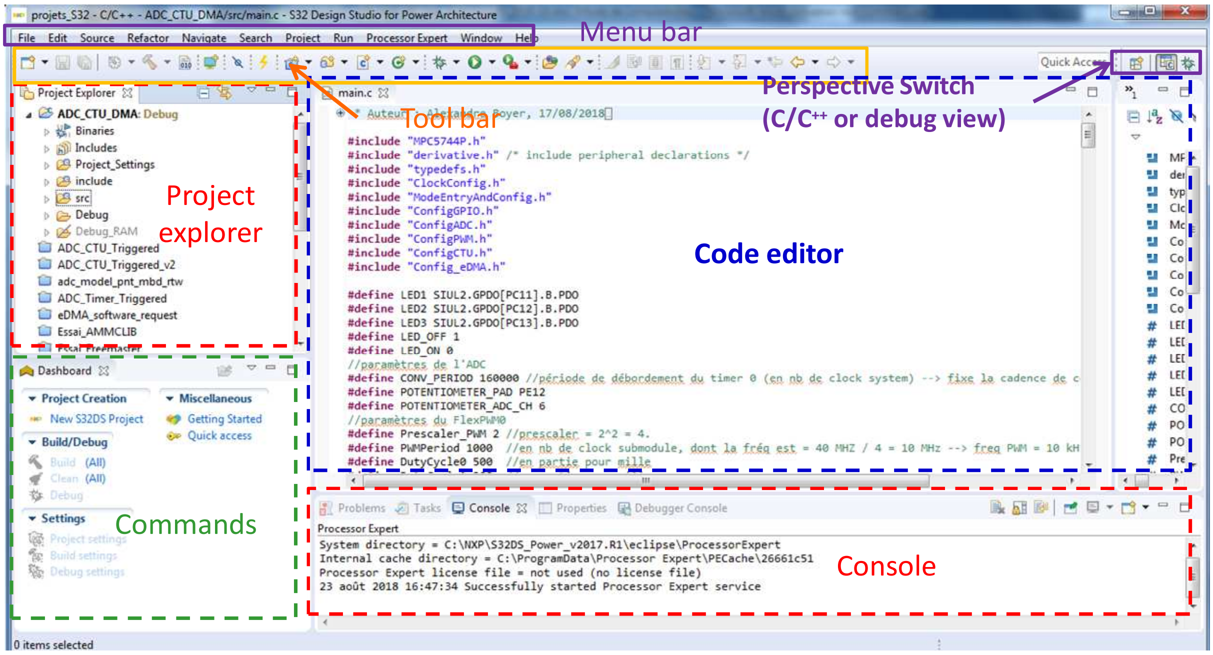
1. Overview of S32DS for Power architecture

S32DS for Power architecture is the integrated development environment provided by NXP for Power Architecture microcontroller (MCU) and automotive applications. The S32 Design Studio is based on the Eclipse open development platform and integrates the Eclipse IDE, GNU Compiler Collection (GCC), GNU Debugger (GDB). It also provides in-situ debugger through several interfaces: P&E Multilink/Cyclone/OpenSDA and supports two software design kits (SDK) that will be used in this lab: FREEMASTER serial communication drivers and Automotive Math and Motor Control Libraries (AMMCLIB).

In this part, the main steps to launch S32DS, create a new project, compile, build, debug and flash your application in the MCU will be described. Here, the MCU MPC5744P is considered.

2. Create a project from scratch

Launch the S32 Design Studio for Power Architecture. A dialog window opens in order to select your workspace. All the S32 project saved in this workspace will be imported.
The window shown in Figure 1 opens. If some existing projects are in the workspace, they will appear in the Project Explorer. The organization of the window is configurable with the menu Window.



```
* Auteur : Alexandre Boyer, 17/08/2018

#include "MPC5744P.h"
#include "derivative.h" /* include peripheral declarations */
#include "typedefs.h"
#include "ClockConfig.h"
#include "ModeEntryAndConfig.h"
#include "ConfigGPIO.h"
#include "ConfigADC.h"
#include "ConfigPWM.h"
#include "ConfigCTU.h"
#include "Config_eDMA.h"

#define LED1 SIUL2.GPDO[PC11].B.PDO
#define LED2 SIUL2.GPDO[PC12].B.PDO
#define LED3 SIUL2.GPDO[PC13].B.PDO
#define LED_OFF 1
#define LED_ON 0
//paramètres de l'ADC
#define CONV_PERIOD 160000 //période de débordement du timer 0 (en nb de clock system) --> fixe la cadence de c
#define POTENTIOMETER_PAD PE12
#define POTENTIOMETER_ADC_CH 6
//paramètres du FlexPWM0
#define Prescaler_PWM 2 //prescaler = 2^2 = 4.
#define PWMPeriod 1000  //en nb de clock submodule, dont la fréq est = 40 MHZ / 4 = 10 MHz --> freq PWM = 10 kH
#define DutyCycle0 500  //en partie pour mille
```

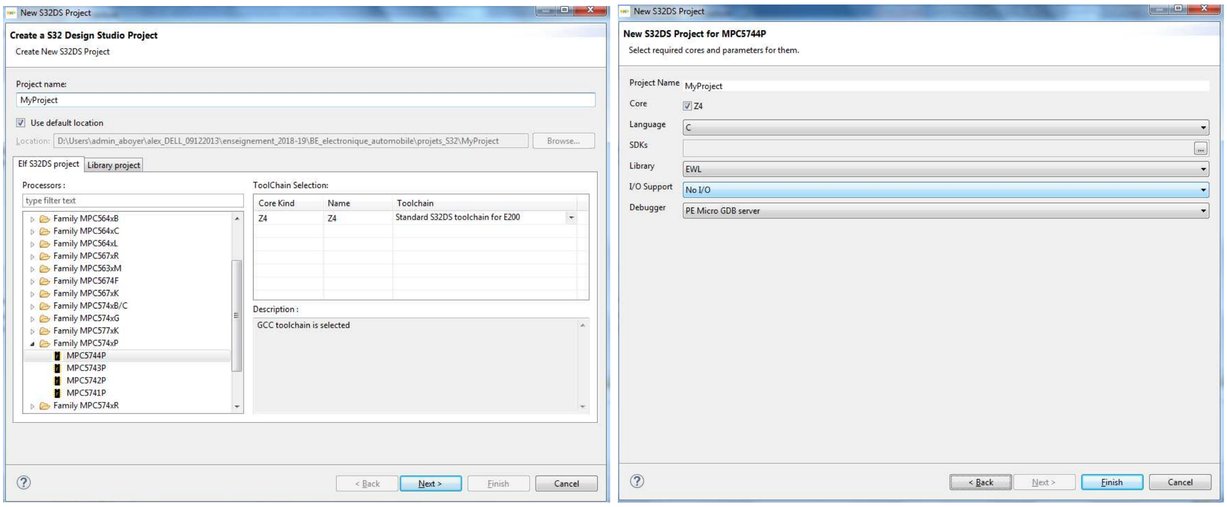
```
System directory = C:\NXP\S32DS_Power_v2017.R1\eclipse\ProcessorExpert
Internal cache directory = C:\ProgramData\Processor Expert\PECache\26661c51
Processor Expert license file = not used (no license file)
23 août 2018 16:47:34 Successfully started Processor Expert service
```

Figure 1 - Main window of S32DS IDE

To open an existing project, write click on its name in the project explorer and select Open Project. You can open source files (.c or .h) and modify them in the Code editor part. Several commands available either in the menu bar, tool bar or Commands window launch the compilation, building, debugging and flashing process. They will be presented later.

In order to create a project from scratch for MPC5744P, follow the procedure described below:

- In the menu bar, click on **File > New > New S32DS project**. The window below opens to setting the target MCU and the project options
- Enter the name of the project in Project Name and its location (by default, in your workspace). In the Elf S32DS project window, select the target microcontroller: **Family MPC574xP** > **MPC5744P**. Click on Next button.



New S32DS Project

Create a S32 Design Studio Project

Create New S32DS Project

Project name: MyProject

☑ Use default location

Location: D:\Users\admin_aboyer\alex_DELL_09122013\enseignement_2018-19\BE_electronique_automobile\projets_S32\MyProject

ToolChain Selection:

Core Kind	Name	Toolchain
Z4	Z4	Standard S32DS toolchain for E200

Description : GCC toolchain is selected

New S32DS Project

New S32DS Project for MPC5744P

Select required cores and parameters for them.

Project Name: MyProject

Core: ☑ Z4

Language: C

SDKs:

Library: EWL

I/O Support: No I/O

Debugger: PE Micro GDB server

Figure 2 - Creating a new S32DS project from scratch

- Select the project options (language, import Software Design Kits (SDK), type of debugger, …). The default configurations are sufficient for this project, except if you need to import SDK (AMMCLIB or FREEMASTER). This point will be addressed in 5).
- Click on Finish to generate the project

S32DS generates the project with all the necessary libraries, start-up, debugger and linker codes. The project folder is visible in the Project Explorer. By default, the Perspective Switch is in C/C++ mode for code development. The target memory is written just after the name of the project:

- debug: the executable code will be downloaded in the Flash memory of the MCU
- debug RAM: the executable code will be downloaded in the SRAM of the MCU

Tips: select Flash to store your program in non volatile memory. Programming Flash is a little longer than programming RAM. During debug stage, it can be more convenient to download your code in RAM.

The project structure is organized as follows:

- Project_Settings: this folder contains all the required files to compile the project, link the files and the start-up code.
- Include: it contains all the header files .h of the project.
- src: it contains all the C/C++ source file of the project. By default, the following files are added after the creation of a new project:
  - main.c: your main code
  - intc_SW_mode_isr_vectors_MPC5744P.c: this file defines the interrupt vector table
  - MPC57xx__Interrupt_Init.c, vector.c and intc_sw_handlers.s: these files define all the function requires for interrupt management
- Debug: this folder contains all the executable source files that will be downloaded into Flash memory. The .elf file is the executable file and the .map file provides the memory location of the code.
- Debug_RAM : this folder contains all the executable source files that will be downloaded into SRAM. The .elf file is the executable file and the .map file provides the memory location of the code.

A default main.c file is opened in the code editor. You can write your own code in this file. Existing files can be copied and pasted from one project to another directly by right clicking on them and selecting Copy or Paste. You can create new source C file by clicking on src folder in your own project and clicking on **New** > **Source File** or on the icon . Type the name of the new source file (give .c as extension). Do not forget to create also the associated header file (.h) that must be included in the folder include. To do so, click on include folder in your own project and clicking on **New** > **Header File** or on the icon .

3. Compile and build your project

Once your source code files (.h and .c files) are written, they must be compiled and the project has to be built before downloading it to the MCU for debugging purpose.
Click on the button **Build** in the Command part, or click in the menu **Project > Build**. Prior to this step, it is necessary to define the target memory (Flash or RAM), as shown in the figure below. It can also be defined by clicking on the small arrow in the right of the button Build .

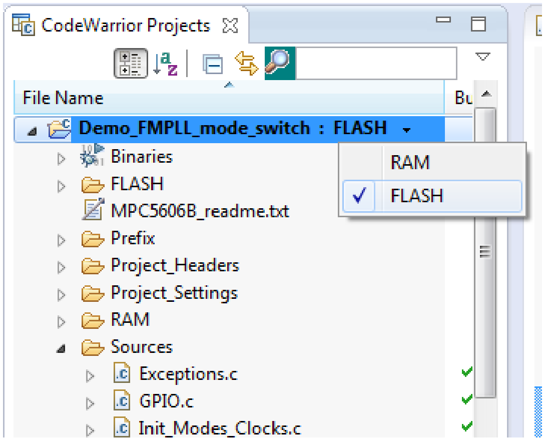
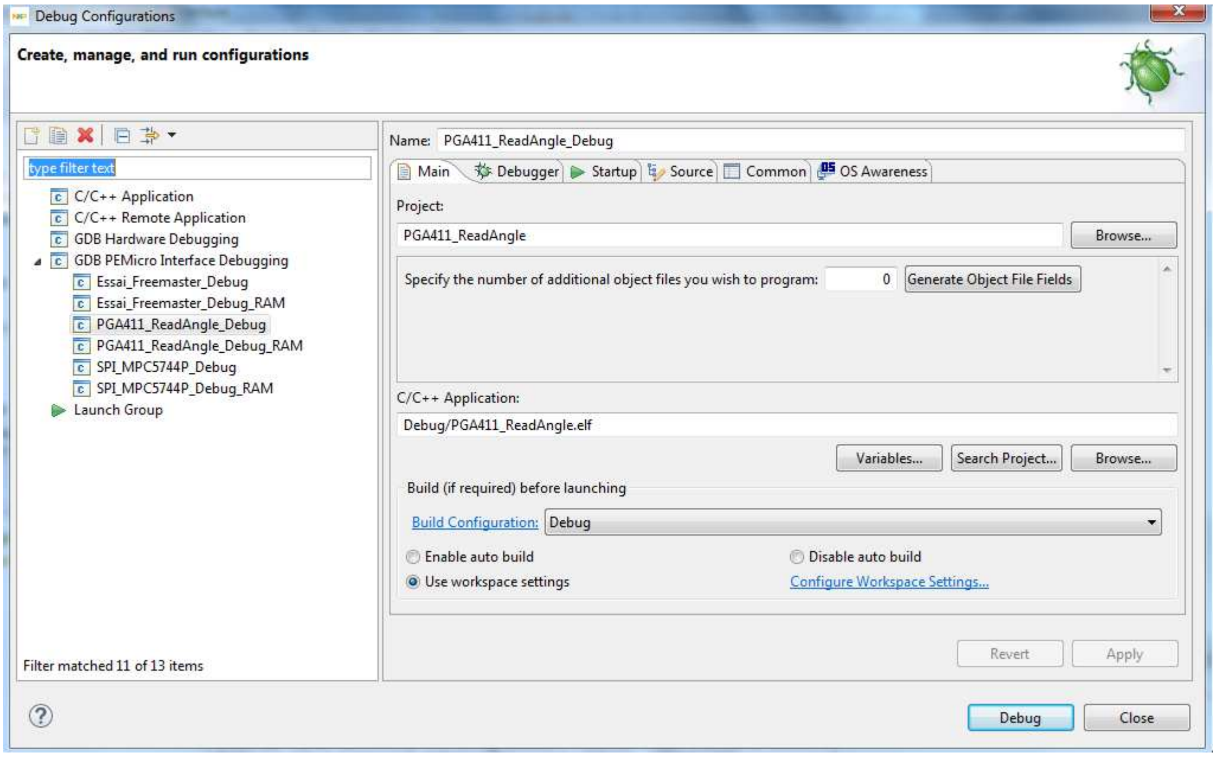


Figure 3 - Select the target memory (Flash or RAM)

Tips: in case of problems during build and link steps, it is recommended to click on the button **Clean** .

4. Programming the MCU

The first step consists in configuring the debug settings. Here, only the selection of the executable files (either those for Flash or those for SRAM) and the programming of the MCU is configured. For the other parameters, the default values can be kept.
Click on the menu **Run > Debug configurations** or on the small arrow in the right of the button Debug to open the debug configuration panel. The window shown in Figure 4 opens. On the left part of the window, the different opened and built source code projects are shown. Select the project the project that you want to download to the MCU. If the executable files have to be downloaded into Flash, select the project with '_Debug' suffix. Otherwise, select the project with '_Debug_RAM' suffix. In right part of the window, in the page **Main**, verify that the correct executable file .elf is selected.



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Remote Application
- GDB Hardware Debugging
- GDB PEMicro Interface Debugging
  - Essai_Freemaster_Debug
  - Essai_Freemaster_Debug_RAM
  - PGA411_ReadAngle_Debug
  - PGA411_ReadAngle_Debug_RAM
  - SPI_MPC5744P_Debug
  - SPI_MPC5744P_Debug_RAM
- Launch Group

Filter matched 11 of 13 items

Name: PGA411_ReadAngle_Debug

Main | Debugger | Startup | Source | Common | OS Awareness

Project:
PGA411_ReadAngle [Browse...]

Specify the number of additional object files you wish to program: 0 [Generate Object File Fields]

C/C++ Application:
Debug/PGA411_ReadAngle.elf

[Variables...] [Search Project...] [Browse...]

Build (if required) before launching

Build Configuration: Debug

○ Enable auto build ○ Disable auto build

● Use workspace settings Configure Workspace Settings...

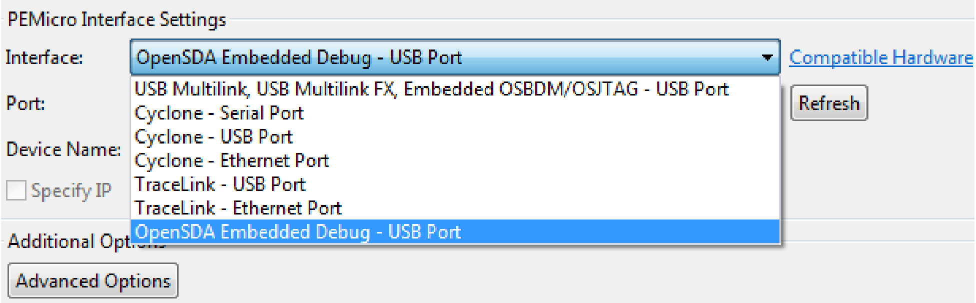
[Revert] [Apply]

[Debug] [Close]

Figure 4 - Debug configuration panel (Run > Debug configurations)

Then, go to the page **Debugger**. The list **Interface** contains all the supported programming interfaces, as shown below. In this lab, you will use development board DEVKIT_MPC5744P. An on-board programming interface, called Open-Standard Serial and Debug Adapter (OpenSDA) is mounted on the board, offering an economical programming interface for the user. You will use this interface primarily. Thus select **OpenSDA Embedded Debug- USB Port** in the list. If the board is connected on a USB port of your computer, information about the port number and the device mounted on the development kit should appear in the fields **Port**, **Device Name** and **Core**.

A programming interface alternative is the **USB Multilink**. This external programming interface is available in this lab.



PEMicro Interface Settings

Interface: OpenSDA Embedded Debug - USB Port

Compatible Hardware

Port:

Device Name:

□ Specify IP

[Refresh]

- USB Multilink, USB Multilink FX, Embedded OSBDM/OSJTAG - USB Port
- Cyclone - Serial Port
- Cyclone - USB Port
- Cyclone - Ethernet Port
- TraceLink - USB Port
- TraceLink - Ethernet Port
- OpenSDA Embedded Debug - USB Port

Additional Opt...

[Advanced Options]

Figure 5 - Selection of the programming interface (Run > Debug configurations)

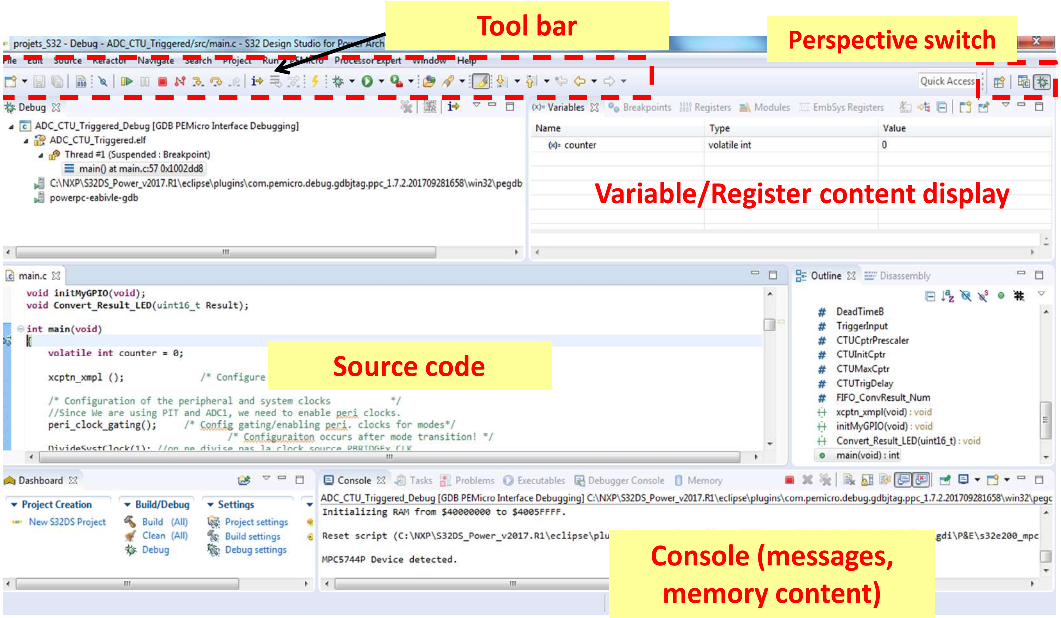
Finally, you can click on the button Debug to start the downloading of the code into MCU memory.

You are not forced to return to the Debug configurations to start the programming of the MCU. Once it has been configuring, you can click on the menu **Run > Run** (Ctrl + F11) or on the button .

5. Debugging your application

Once you click on the button Debug, the downloading of the executable files into the MCU starts. Ensure that the MCU board is connected to your computer through a programming interface and correctly powered. The process can last several tens of seconds. As explained before, programming the Flash memory is longer than the programming of the RAM.

During the downloading process, the Perspective Switch changes from C/C++ to Debug mode and window shown in Figure 6 appears.


Tool bar
Perspective switch
Variable/Register content display
Source code
Console (messages, memory content)
projets_S32 - Debug - ADC_CTU_Triggered/src/main.c - S32 Design Studio for Power Arch
File Edit Source Refactor Navigate Search Project Run PEMicro Processor Expert Window Help
Quick Access
Debug
Variables Breakpoints Registers Modules EmbSys Registers
main.c
Outline Disassembly
Dashboard
Project Creation
New S32DS Project
Build/Debug
Build (All)
Clean (All)
Debug
Settings
Project settings
Build settings
Debug settings
Console Tasks Problems Executables Debugger Console Memory
DeadTimeB
TriggerInput
CTUCptrPrescaler
CTUInitCptr
CTUMaxCptr
CTUTrigDelay
FIFO_ConvResult_Num
xcptn_xmpl(void) : void
initMyGPIO(void) : void
Convert_Result_LED(uint16_t) : void
main(void) : int

ADC_CTU_Triggered_Debug [GDB PEMicro Interface Debugging]
ADC_CTU_Triggered.elf
Thread #1 (Suspended : Breakpoint)
main() at main.c:57 0x1002dd8
C:\NXP\S32DS_Power_v2017.R1\eclipse\plugins\com.pemicro.debug.gdbjtag.ppc_1.7.2.201709281658\win32\pegdb
powerpc-eabivle-gdb

Name	Type	Value
counter	volatile int	0

```
void initMyGPIO(void);
void Convert_Result_LED(uint16_t Result);

int main(void)
{
    volatile int counter = 0;

    xcptn_xmpl ();              /* Configure

    /* Configuration of the peripheral and system clocks          */
    //Since We are using PIT and ADC1, we need to enable peri clocks.
    peri_clock_gating();      /* Config gating/enabling peri. clocks for modes*/
                                /* Configuraiton occurs after mode transition! */
    DivideSystClock(1); //on ne divise pas la clock source PBRIDGEx_CLK
```

```
ADC_CTU_Triggered_Debug [GDB PEMicro Interface Debugging] C:\NXP\S32DS_Power_v2017.R1\eclipse\plugins\com.pemicro.debug.gdbjtag.ppc_1.7.2.201709281658\win32\pegc
Initializing RAM from $40000000 to $4005FFFF.

Reset script (C:\NXP\S32DS_Power_v2017.R1\eclipse\plu                gdi\P&E\s32e200_mpc

MPC5744P Device detected.
```

Figure 6 - In-situ debugging interface

The debugging process is controlled by the commands provided in the Tool bar (Figure 7). The Run button starts the execution of the embedded program. The execution can be paused by clicking on the button Pause. The execution can be performed step-by-step by clicking on the step buttons.

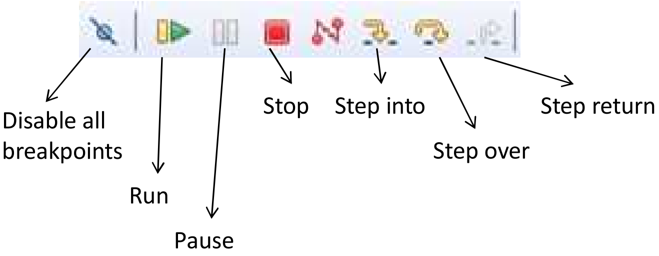


Figure 7 - Debug toolbar

Breakpoints can be inserted in the source code by double clicking on the source code line where you want to insert the breakpoint. You can also right click and select **Add Breakpoint**. The breakpoint is removed by double clicking on it or right clicking and select sur **Toggle Breakpoint**. It can be deactivated by clicking on **Disable breakpoint** and reactivated with **Enable Breakpoint**. Each time the execution pauses (due to a breakpoint or a click on button Pause), the memory content is refreshed.

At the end of in-situ debug operation, you have to stop the debugger by clicking on the button Stop. Then, click on the perspective switch to return in C/C++ mode.

Tips : do not forget to stop the debugger before returning in C/C++ Perspective. Otherwise, the debugger will continue to run. The next time you will try to reprogram the MCU, an error message will be displayed to warn you that a in-situ debug is still on-going.

6. Closing and importing project

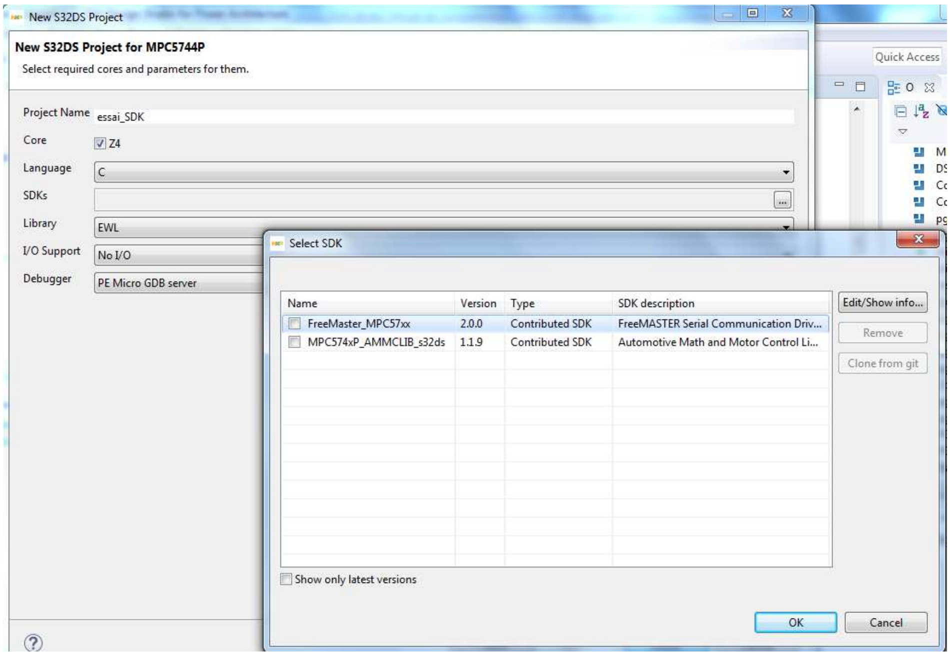
In order to close a project, select the project in the Project Explorer. Click on the menu **Project > Close Project** or right click on it and select **Close Project**.
If you click on **Edit > Delete**, the project is removed from the Workspace and thus from the Project Explorer.
If you want to import an existing project into your workspace, click in the menu **File > Import** or on the button **Import project**.

7. Installing and using SDK

In this lab, you will certainly use two software design kits (SDK) provided by NXP:

- FREEMASTER communication drivers
- Automotive Math and Motor Control Libraries (AMMCLIB)

Both SDK are free but they are not installed in S32DS by default. Here, we explain how to install SDK and use the provided source codes and drivers. The downloading links and the contents of these SDK will be detailed in part III and V of this document.
SDK have to be selected during the project creation, in order to import all the library files. When you create a new project with S32DS, as explained in part I.2, click on the button ... in the field SDK of the window New S32DS Project. The window shown in Figure 8 opens. All the SDK installed in your PC are listed (in this example, Freemaster and AMMCLIB are installed). Select the SDK that you want to use and click OK. In the window New S32DS Project, click on Finish. A new project is automatically generated. In the Project Explorer, you can verity that source files associated to the SDK have been added.



New S32DS Project

New S32DS Project for MPC5744P
Select required cores and parameters for them.

Project Name: essai_SDK
Core: ☑ Z4
Language: C
SDKs:
Library: EWL
I/O Support: No I/O
Debugger: PE Micro GDB server

Select SDK

Name	Version	Type	SDK description
□ FreeMaster_MPC57xx	2.0.0	Contributed SDK	FreeMASTER Serial Communication Driv...
□ MPC574xP_AMMCLIB_s32ds	1.1.9	Contributed SDK	Automotive Math and Motor Control Li...

□ Show only latest versions

Figure 8 - Selecting SDK in a S32DS project

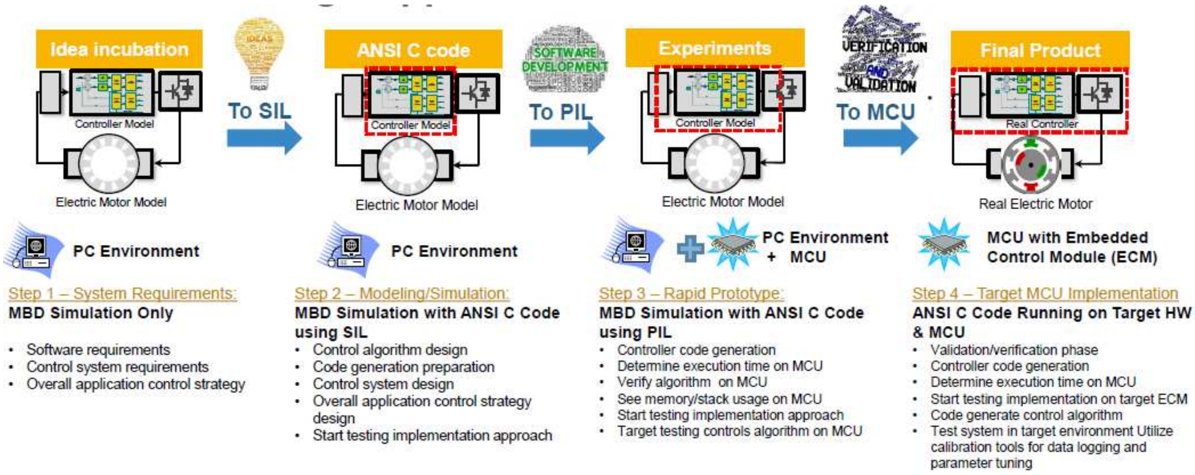
II - Presentation of Matlab/Simulink for motor control application development on MPC5744P

Simulink is dedicated to the modeling and simulation of dynamic multidomain systems, operating in continuous or discrete time, or mixed. For this reason, it is widely used in motor control design. Moreover, Simulink is able to generate code in C/C++. That's why Matlab/Simulink is also a more and more popular tool in embedded code design, especially in automotive industry. For example, in this project, from MBDT toolbox and coupled with S32DS compiler, embedded code for MPC5744P can be generated from Simulink diagram and flashed into the MCU.

Why using Simulink instead of a classical integrated development environment (IDE) ? Within a typical IDE, you develop your code and then you download it and you can do in-situ debug. With Simulink, if the toolbox associated to a hardware target (e.g. a MCU) is provided (Model-Based Design (MBD) approach) and a link with its compiler exist, then system modeling and code generation can be mixed. The MCU and its environment can be modeled in Simulink and the command concept or algorithm validated by simulation. The MCU, the external devices are emulated by Simulink behavioral or physical models. For example, for MPC5744P, MCU model is based on the MBDT toolbox for MPC574xP presented in part IV.

But there is no guarantee that it will work in a real implementation. For example, if you power driver and a motor must be used, any undetected algorithm error may lead to a serious failure. In order to prevent this situation, Software In the Loop (SIL) and Processor In the Loop (PIL) are widely used in industry before the real implementation on MCU, as illustrated in Figure 9:

- SIL: the model of the MCU is replaced by the C code of the future embedded application, which can be interpreted by Simulink. All the external devices (power devices, motor, sensors) are modeled with Simulink model.
- PIL: in a co-simulation environment, the C code is executed on the MCU in order to validate the algorithm and its performances on a real hardware platform. The emulating models of external components are executed by Simulink models. It requires specific hardware devices to interface with Simulink.



Step 1 – System Requirements:
MBD Simulation Only

- Software requirements
- Control system requirements
- Overall application control strategy

Step 2 – Modeling/Simulation:
MBD Simulation with ANSI C Code using SIL

- Control algorithm design
- Code generation preparation
- Control system design
- Overall application control strategy design
- Start testing implementation approach

Step 3 – Rapid Prototype:
MBD Simulation with ANSI C Code using PIL

- Controller code generation
- Determine execution time on MCU
- Verify algorithm on MCU
- See memory/stack usage on MCU
- Start testing implementation approach
- Target testing controls algorithm on MCU

Step 4 – Target MCU Implementation
ANSI C Code Running on Target HW & MCU

- Validation/verification phase
- Controller code generation
- Determine execution time on MCU
- Start testing implementation on target ECM
- Code generate control algorithm
- Test system in target environment Utilize calibration tools for data logging and parameter tuning

Figure 9 - Illustration of SIL and PIL in motor control application development (from NXP)

Simulink is able to produce embedded code which respond to quality standard required by automotive industry (such as MISRA-C, AUTOSAR) or safety requirements (such as ISO 26262 for integrity levels ASIL A to D). However, the equipment manufacturer has certainly to include additional requirements which were not provided by Simulink, or reduce the size of the code. The last stage consists in optimizing the code before the final implementation of the code in the equipment.

In the Automotive Lab, Simulink will be used to model the motor and its command, but also to generate C-code for the MPC5744P with the MBDT toolbox and S32DS compiler. SIL approach will be possible. PIL approach will not be done during the lab because the required equipments are not available. Thus, after a validation during SIL, tests will be done directly on the real implementation.

In this chapter, a rapid description of the steps to create and simulate Simulink models is provided. The most important Simulink libraries for the project are presented briefly. A particularly important Simulink library for algorithm development is Stateflow. It provides blocks to create state diagrams which is a powerful and suitable approach to construct algorithm. Finally, the steps to generate C-code for MPC5744P and dowload it to the MCU target are presented.

1. Creating model in Simulink

In Matlab, the first step before the creation of any model files is the definition of a workspare, where all your models and associated data will be saved.

From Matlab, you can launch Simulink by clicking on Simulink icon (in Home) or in the menu **Home > New > Simulink model**. You can also click on the icon New and select Simulink model. If existing Simulink models exist in your workspace, you can directly click on them to launch Simulink.
Simulink opens after several seconds. In order to create a new model, click on the menu **File > New > Blank model** or on the icon . A Simulink model consists in a set of interconnected blocks which models a continuous, discrete or mixed systems. Existing blocks can be found in libraries and toolboxes installed on your working environment. Click on the menu **View > Library browser** or on the icon to open the library browser, which lists

all the installed libraries, as shown in Figure 10. The main common libraries for the Automotive electronics lab will be briefly presented in part II.2.

To place a block in the model diagram, click on the library to open it. Then, select the correct category and the desired block. Finally, drag and drop it on the model schematic. Place as many blocks on the diagram.

Each block has a name, which appears on the bottom side of the block. You can change it by clicking on the name. The properties of the model associated to the block can be modified by double clicking on the block. A window opens to modify the model properties (model parameters, data type, min/max value…). Graphical properties can be modified by right clicking on the block and select **Properties** in the pop-up window. The different blocks can be interconnected by clicking on one terminal of a block. A wire starts to be plotted. It finishes at the position where you release the mouse cursor. If you double click on a wire, you can give a label to the wire.

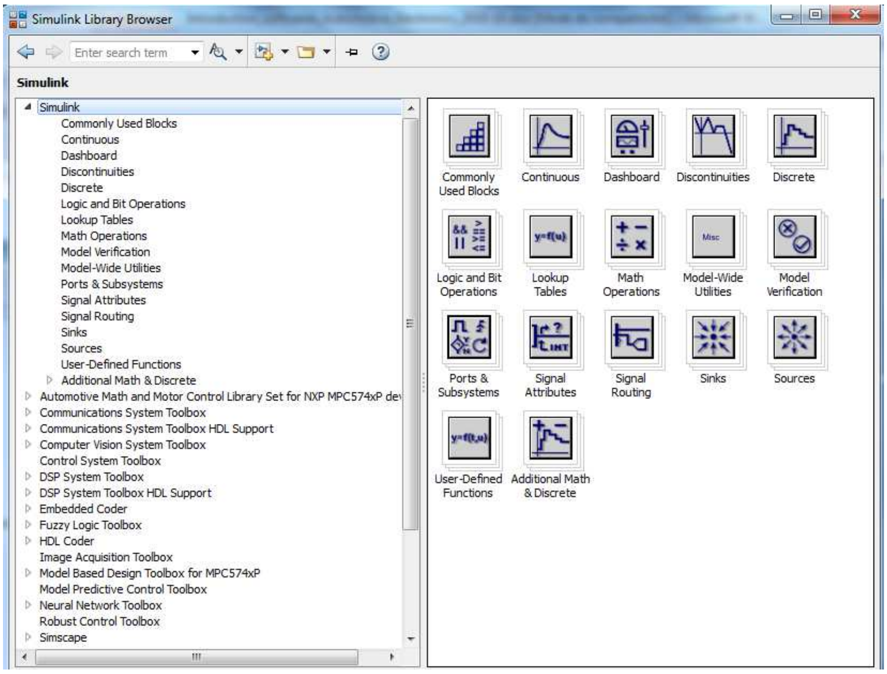


Figure 10 - Simulink Library browser

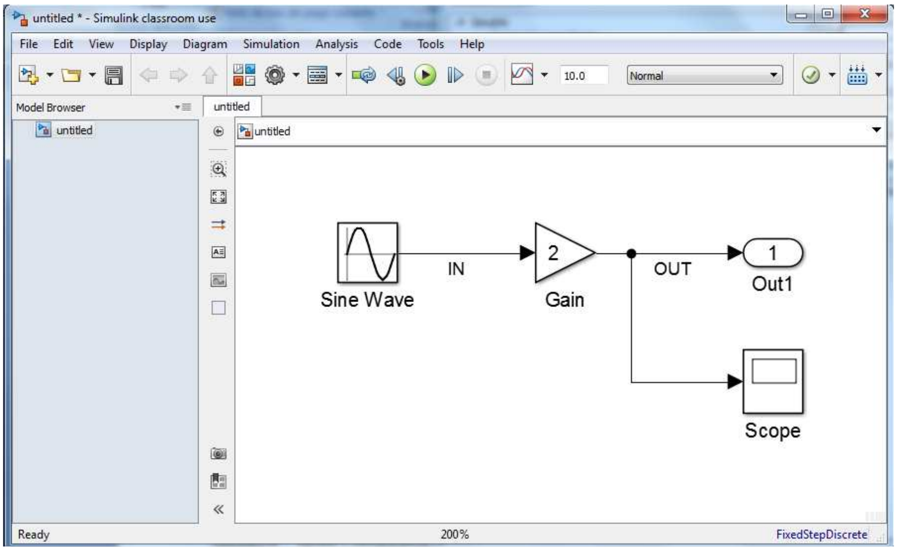
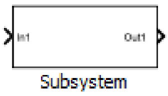


Figure 11 - Creation of a simple Simulink diagram

<u>Tips : data type.</u> Most of the blocks has an output data type properties (in Signal Attributes). By default, it is set to Inherit, meaning that Simulink will set the type after a global analysis of the schematic. It is recommended to set the data type to prevent errors due to data type mismatch.

When a large number of blocks are added on the diagram, its reading becomes quite difficult. Moreover, the complete model of a system is usually divided in several subparts. That's why it is recommended to organize blocks. Simulink proposes to build hierarchical view of models through the use of subsystems. Several types of subsystem blocks are proposed in the native Simulink library / Ports & Subsystems. More details about the different subsystems will be given in part II.3. The most basic subsystem block is:



Place it on the diagram. Select the blocks that you want to insert in the subsystem, cut them, double click on the subsystem to open it and paste the blocks. If input and output ports are required, add (1)> and >(1) on the terminals.

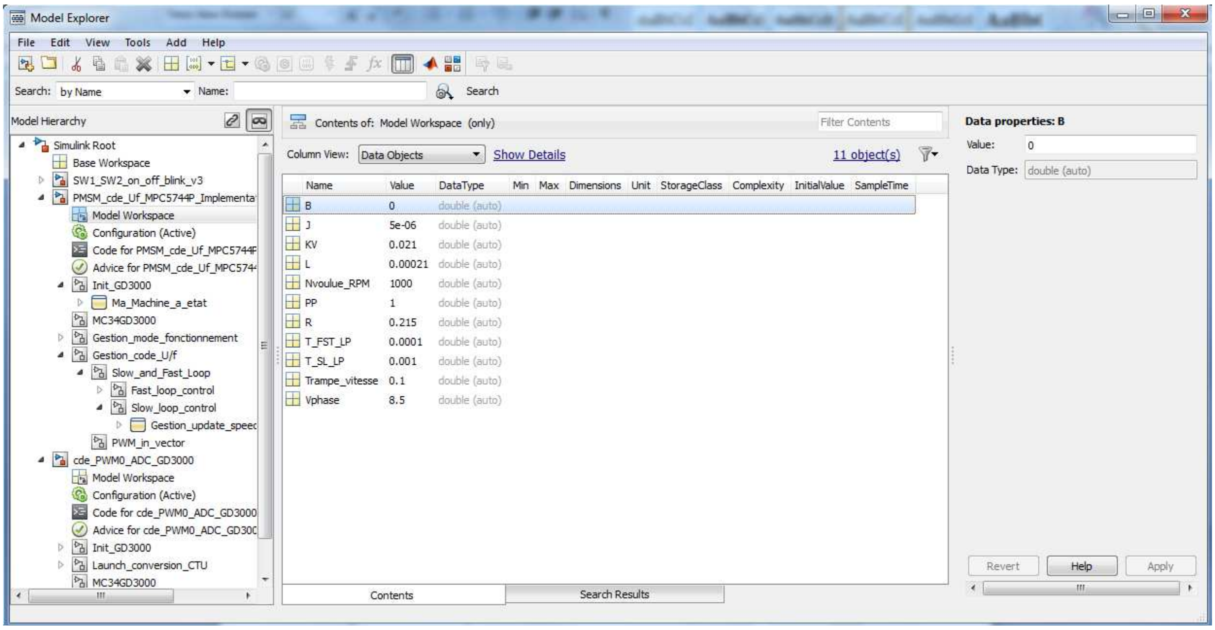
The main icons to interact with diagram are summarized in the following table:

⛶	View fil all (space bar)	▨	Add image
🔍	View in (CTRL + '+'). View out = CTRL + '-'.	⇧	To return to the higher level in the diagram hierarchy
A≡	Add text annotation	⇦ ⇨	Back/forward = return to the previous/next view

2. *Creation of variables*

Variables can be added in Simulink's diagrams. These variables are stored in the workspace shared with Matlab. To create or modify variable properties, you have to open the **Model Explorer**, by clicking on icon ▤, or on menu **View > Model Explorer > Model Explorer** or CTRL+ H. The window shown below appears. On the panel on the left, all the variables, subsystems, blocks, charts added in the opened model diagrams are visible. For each Simulink model, a Model Workspace is defined by default. It stores all the variables of this model. The values of the variables, types and properties are visible in the central panel. Double click on them to change their properties. After each change, click to button **Apply** to validate the changes.

To create new variables, click on the icon Add MATLAB variable ⊞. To remove one variable, click on it and press the key 'Delete'.



Search: by Name Name: Search

Contents of: Model Workspace (only)

Column View: Data Objects

Name	Value	DataType	Min	Max	Dimensions	Unit	StorageClass	Complexity	InitialValue	SampleTime
B	0	double (auto)								
J	5e-06	double (auto)								
KV	0.021	double (auto)								
L	0.00021	double (auto)								
Nvoulue_RPM	1000	double (auto)								
PP	1	double (auto)								
R	0.215	double (auto)								
T_FST_LP	0.0001	double (auto)								
T_SL_LP	0.001	double (auto)								
Trampe_vitesse	0.1	double (auto)								
Vphase	8.5	double (auto)								

Data properties: B

Value: 0

Data Type: double (auto)

Figure 12 - Model Explorer

3. Placing scopes

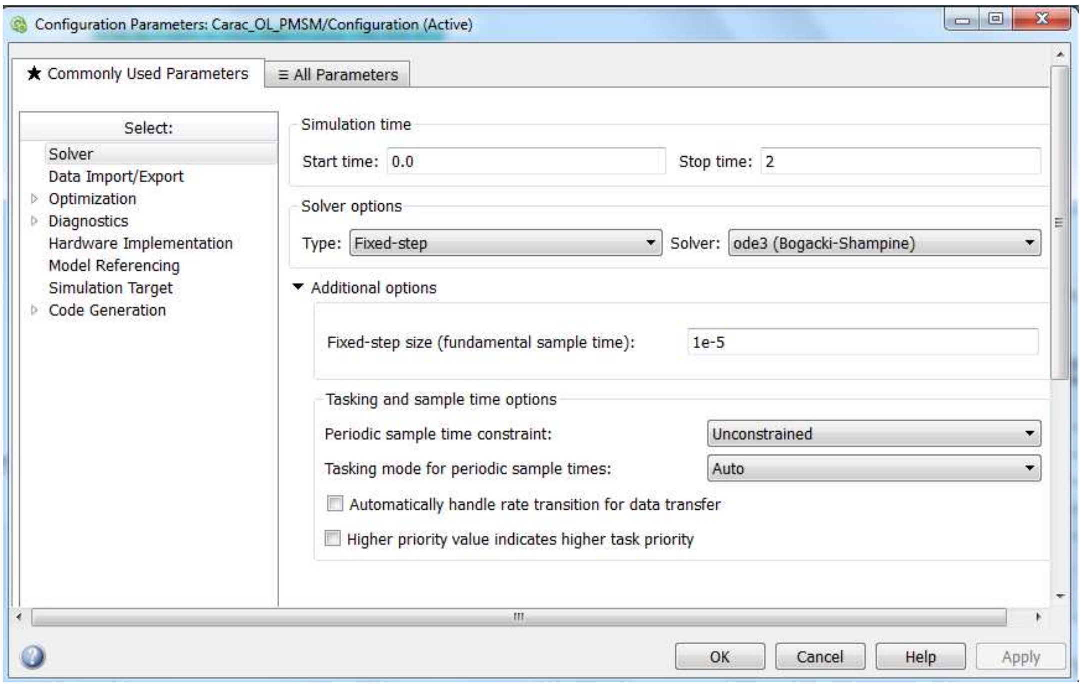
Several methods can be used to display the values of one net of the diagram during simulation of a Simulink model.

The first method consists in placing a scope connected to the net. Only the net connected to the block can be observed with this method. Scope block is available in the native Simulink library / Sinks Scope . Click on this block to configure the scope settings.

In complex diagrams, it is usually necessary to observe numerous signals simultaneously and plot them either on the same graphs or one different subwindows. It is possible with Floating scope, which is also available in Sinks library Floating Scope . You can place it anywhere on the diagram. Double click on this block to edit its properties. Click on **View > Layout** to select the number of subwindows. Click on **Simulation > Signal selector** to select the signals to be plotted and the destination subwindows (or axes).

4. Configuration of simulation

The simulation must be configured before the first launching. Click on **Simulation > Model Configuration Parameters** or on the icon to open the simulation configuration window, as shown in Figure 13. Select the pane Solver. By default, the solver is automatically chosen by Simulink and only the start and stop times can be configured. The solver can be chosen and also the step type (fixed or variable). In the pane Diagnostics, various sources of errors can be configured to trigger simulation errors or be ignored.



Configuration Parameters: Carac_OL_PMSM/Configuration (Active)

Simulation time

Start time: 0.0 Stop time: 2

Solver options

Type: Fixed-step Solver: ode3 (Bogacki-Shampine)

Additional options

Fixed-step size (fundamental sample time): 1e-5

Tasking and sample time options

Periodic sample time constraint: Unconstrained

Tasking mode for periodic sample times: Auto

□ Automatically handle rate transition for data transfer

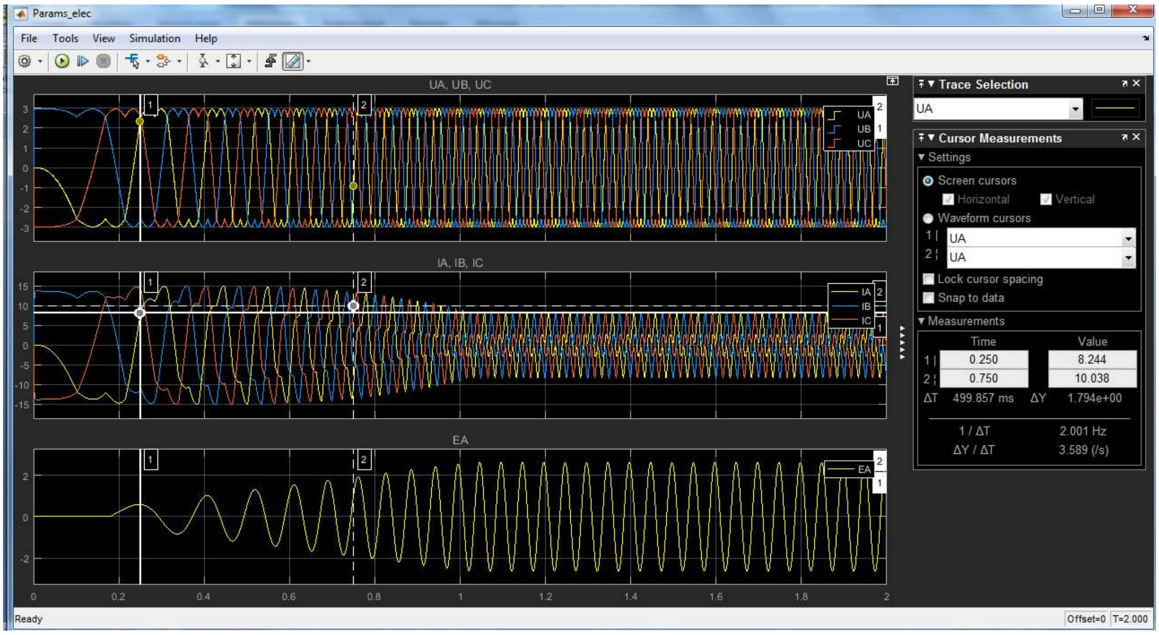
□ Higher priority value indicates higher task priority

Figure 13 - Configuration Parameters window

5. Launching simulation

The simulation can be launched by clicking on **Simulation > Run** or on the icon ▶. Simulink starts the simulation process by a compilation of the model and, if no errors are detected, launches the simulation. It ends when the stop time is reached. The simulation can be also done step-by-step by clicking on the icon ▷. It can be stopped by clicking on the icon ■.

Results are displayed in real-time on all the opened oscilloscopes. For example, Figure 14 shows simulation results plotted on a floating oscilloscope, configured with three subwindows. The menu and toolbar aim at controlling simulation, selecting the curves to be plotted, modifying graphical parameters and placing cursors.



Trace Selection: UA

Cursor Measurements

Settings

● Screen cursors

☑ Horizontal ☑ Vertical

○ Waveform cursors

1: UA

2: UA

□ Lock cursor spacing

□ Snap to data

Measurements

	Time	Value
1	0.250	8.244
2	0.750	10.038
ΔT	499.857 ms	ΔY 1.794e+00
1 / ΔT		2.001 Hz
ΔY / ΔT		3.589 (/s)

Figure 14 - Example of simulation results plotted on a floating oscilloscope

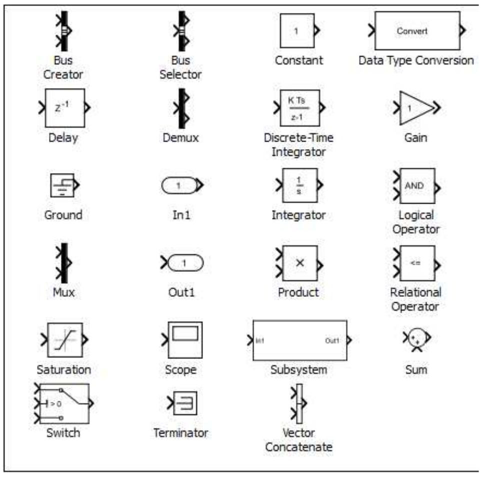
In View > Configuration Properties …, the main properties of the graph window can be modified. For example, click on the panel Display and check the box Show Legend to display the legend of the plotted curves. Click on Tools > Measurements > Curve measurements to display cursors on the graph. The cursor panel is displayed on the right of the graph, as shown in Figure 14.

6. The main common libraries

In this part, the most important library for the Automotive Lab are presented briefly. More details can be found in the on-line help of Simulink. These libraries are present by default in Simulink. These are those shown in Figure 10.

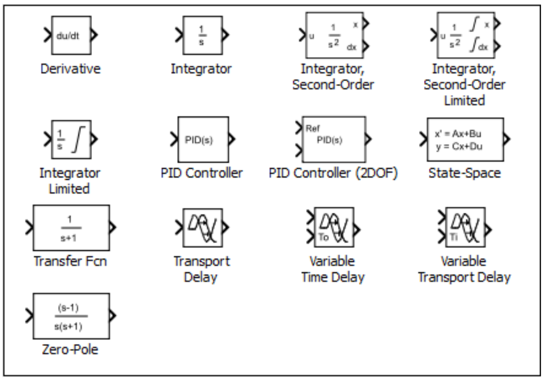
a. Commonly used blocks

The sublibrary lists the mostly used blocks. These blocks are available in other sublibraries.



b. Continuous

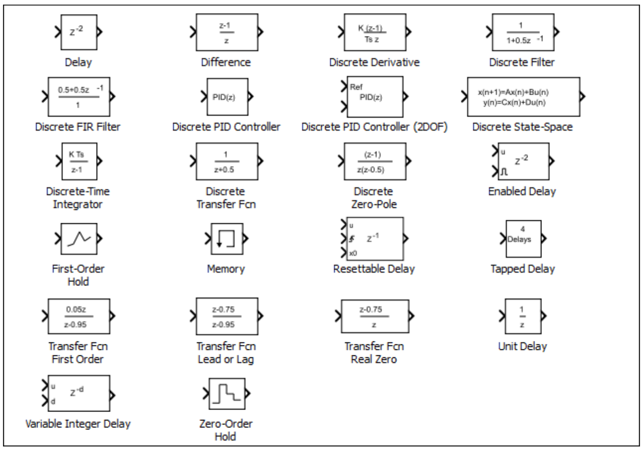
This sublibrary proposes all the basic blocks to define linear system behavior in continuous time: derivator, integrator, transfer function in Laplace domain, PID controller, state-space model…

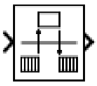


c. Discrete

This sublibrary proposes all the basic blocks to define linear system behavior in discrete time: discrete-time derivator, discrete-time integrator, delay, transfer function in Z-domain, discrete PID controller, …

Discrete-time blocks can coexist with continuous-time blocks. Use of discrete-time function requires sampling time, which must be in accordance with simulation step time.



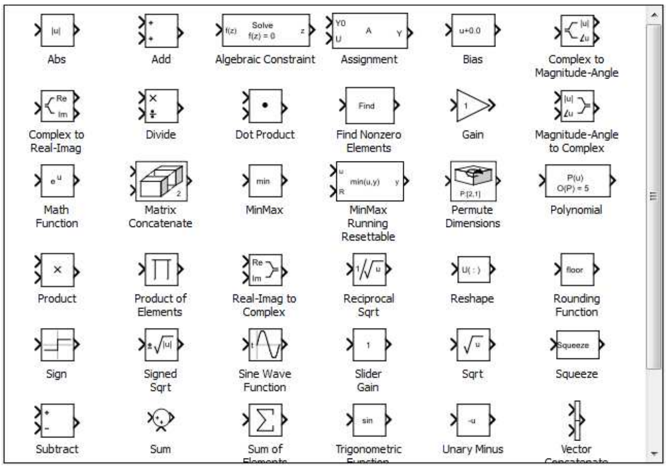
Two discrete-time blocks which operate at two different sampling times can be connected, if Rate transition block  (available in Signal Attributes sublibrary) is inserted within the interconnection.

d. Math operation

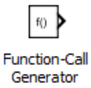
:

This sublibrary proposes all the basic mathematical operations: add, substract, multiply, gain, sign, sqrt…

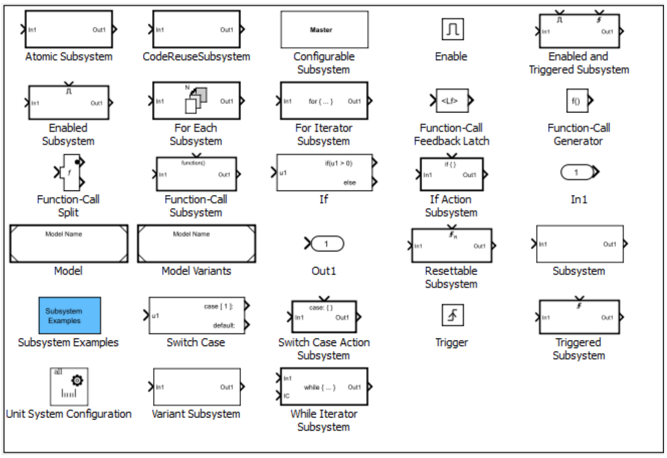
.



e. Ports and subsystem

This sublibrary proposes all the elements to build subsystems (refer to part 1), to define their ports and call them.
Submodels inserted in a block Subsystem are executed all the time. Submodels inserted in Function-Call Subsystem are executed only when a function-call trigger them. Function-call generator 

can be inserted in a subsystem to trigger a function-call when the subsystem is executed.

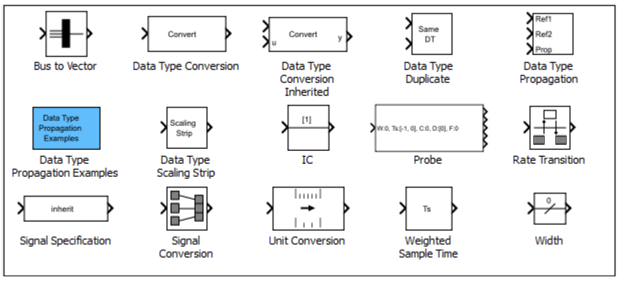


Function-call signals is a specific signal in Simulink which cannot be processed as the other signals (whatever their types). Blocks as Function-call Split

are useful to deliver the same function-call signal to several subsystems.

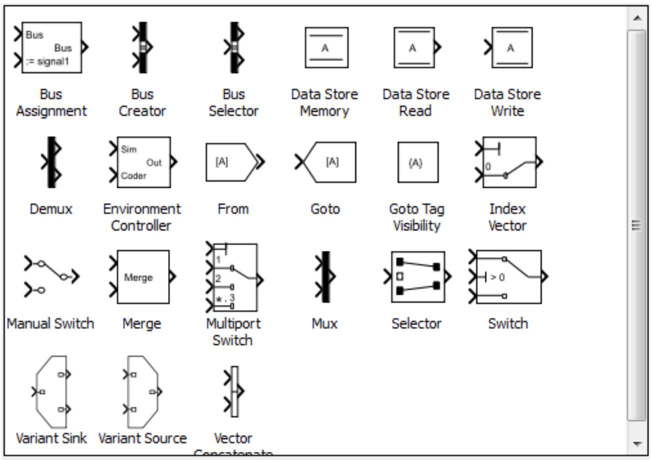
f. Signal attributes

This sublibrary provides blocks to manipulate signal properties: data type conversion, rate transition (when two interconnected blocks operate at two different sampling times)…



g. Signal routing

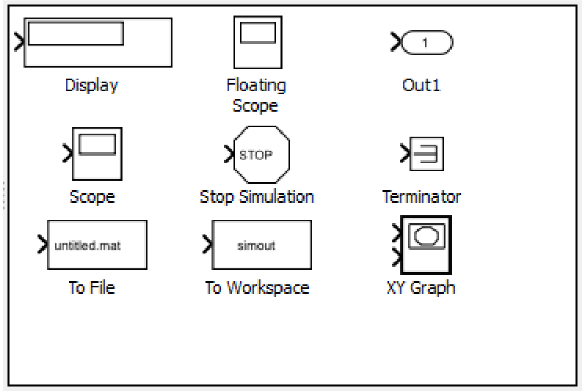
This sublibrary provides blocks to route signals and manipulate variables stored in memory.
Blocks Mux and Demux are useful in a complex diagram where a large number of signals are exchanged. Signals with similar functions can be gathered by multiplexing.
Variables can be created with the block Data Store Memory. the variable name is defined in this block When C code is generated with Simulink, a variable is creates when this block is inserted.



Writing this variable is done with the block Data Store Write . This variable is read with the block Data Store Read .

h. Sinks

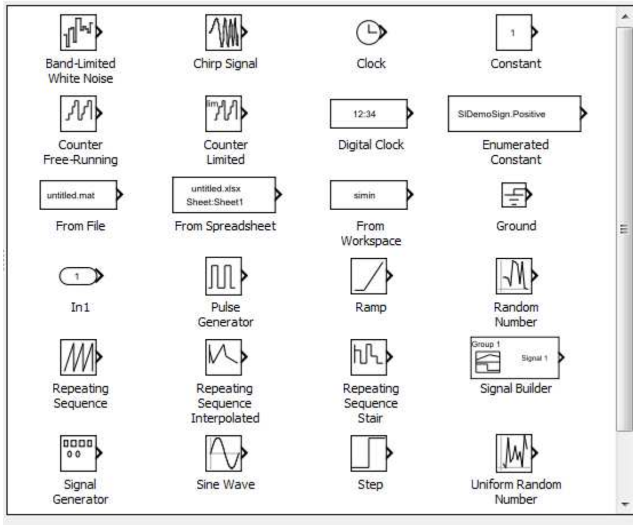
This sublibrary contains elements that terminate a chain of elements. An output cannot be let floating.
Signals can be terminated by graphical visualization tools, such as oscilloscopes (Floating scope, scope, XY Graph). signals waveforms can also be exported to result file or workspace.
Placing an output port is also acceptable for a subsystem. If the previous solutions are not possible, place Terminator block.



i. Sources

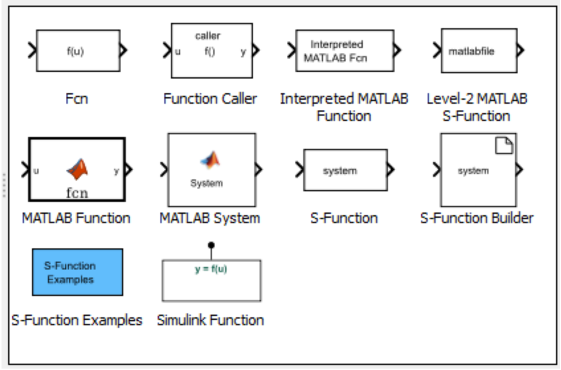
This sublibrary provides various excitation sources: constant signal, sine waveform, step, pulse, ramp, random signal, noise, arbitrary signal …

If the simulation time has to be stored, the Clock symbol can be placed on the model diagram and connected to Workspace.
Signal Generator block provides a graphical interface to set the time-domain evolution of a signal.



j. User-defined function

This sublibrary provides blocks to include functions developed by the user: Matlab function, Simulink model, S-function.



7. Stateflow library

Stateflow is the appropriate Simulink toolbox to model and generate systems with control and supervision logic. It proposes a graphical environment to design state diagrams, flow charts and truth tables. While Simulink is dedicated to model continuous change in dynamical systems, Statflow aims at modeling instantaneous changes, such as it exists in systems with a control logic. Usual continuous and discrete-time Simulink blocks can be combined with Stateflow blocks to model such systems.

a. Content of the library

Stateflow offers various types of blocks, as shown in Figure 15. In this part, only the creation of Chart will be described. Two types of charts exist in Stateflow: state diagrams and flow charts.

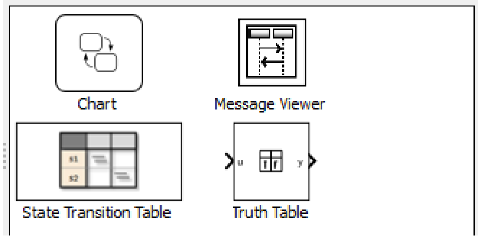


Figure 15 - Blocks provided by Stateflow library

b. Creating state diagram

A state diagram is a modeling approach of reactive systems. It defines all the finite states and behaviors of a system, and the conditions for transition from one state to another. State diagrams can be used to model complex logic in dynamic systems.

Stateflow state diagrams or charts contains five fundamental elements:

- the states
- the transitions
- the actions
- sub-diagrams
- graphical functions

Click on the symbol Chart to create a new state diagram (or flow chart). Chart is a particular subsystem with signal and trigger inputs and outputs. Once created, the chart has no inputs and outputs. Double click on the block to edit it. The chart edition window opens. The chart is blank.
The building elements are available in the toolbar on the left part of the screen Figure 16).
For state diagrams, only State, Transition, Simulin/Graphical/Matlab functions, Truth table and History blocks are used. Junction block defines a pseudo-state used in flow charts.

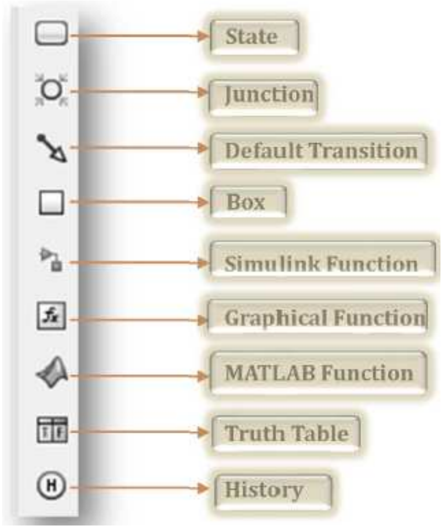


Figure 16 - Building elements of a chart

Creating the states:

Before starting the edition of the flowchart in Simulink, draw it and validate its structure. In order to create a new state, click on the button State and give it a name. By default, when the first state of a diagram is created, the Default transition is added. This transition is done at initial execution of the diagram. The default transition can be added on any states by clicking on the button . Actions can be associated to each state. Click on a state and type 'entry:'+command to specify actions that must be executed when the system enters in a state. Similarly, 'exit:'+command to specify actions that must be executed when the system leaves a state. Variables with various types (boolean, integer, double, enum…) can be defined in the command.

Creating the transitions between the states:

The states are connected by transitions. Click on one side of a state: an arrow appears. Drag it to one side of another state to create a transition. Click on the transition to add information to this transition:

- transition conditions, defined by expression written between [ ]. Expressions contain variables in logical equations, where ! or ~ define logical not, && defines and, || defines or, and == defines equality. Functions can also be used to test transition conditions. Variables can also be array with index defined between [] and starting at 0.
- transition actions, which defines actions triggered by a transition between two states. The syntax is a list of actions separated by ';' between {}: {action1;action2; …}. Actions consists in modifying output signals and variables.
- Time conditions for transitions, such as after(value). The time value can be expressed in sec, msec, usec, or expressed in tick (i.e. sampling time)
- Event on transition with the nature of the trigger (rising/falling edge or both, function call.

Creating input/output ports:

The variables defined in state commands or transitions conditions/actions may be associated to input or output ports of the diagram. Thus, they must appear on the Simulink diagram to connect the chart to the rest of the Simulink diagram. Moreover, all the other variables and parameters used in the chart must be defined in the Workspace. For this purpose, the tool Symbol Wizard is used. A simple method to open it consists in launch Simulink simulation

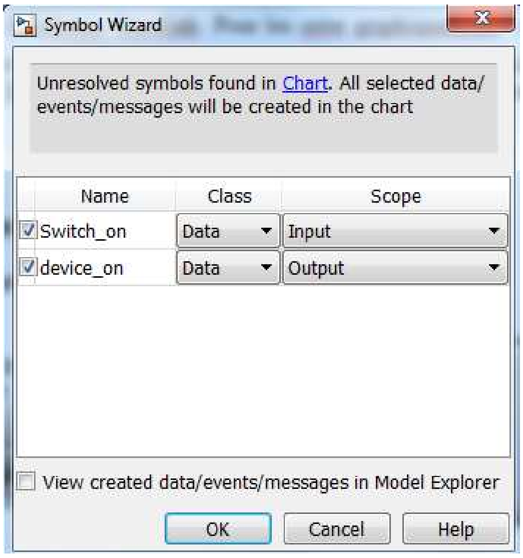
▶. If a variable has not been declared, error arises and Symbol Wizard opens automatically, to suggest you to declare the variables (Figure 17). They can be defined in the column Class as:

- Data
- Event

They can be defined in Scope as:

- Input
- Output
- Local (the variable is local to the chart)

Click on OK. Normally, all the variables defines as input or output appears as input or output terminal of the chart symbol. All these properties can be modified later in the Model Explorer (Ctrl+H).



Unresolved symbols found in Chart. All selected data/ events/messages will be created in the chart

Name	Class	Scope
☑ Switch_on	Data	Input
☑ device_on	Data	Output

□ View created data/events/messages in Model Explorer

Figure 17 - Symbol Wizard - definition of the nature of variables of a chart

The other method to declare the nature of variables consists in clicking in the menu **Chart > Add Inputs & Outputs > Data Input from Simulink** or **Data Output to Simulink** in the chart edition window.

Example of state flow diagram:

Figure 18 presents an example of simple Stateflow diagram. It models the behavior of a system, which operates in three states (Reset, Init, Normal), according to the logical commands sent by two switches (InitOK_switch and Reset_switch). On the left part of the figure, the complete Simulink diagram is presented. The switches are connected to the chart inputs. The chart has three outputs: Out0, Out1 and TestInit, which change according to the system states. On the right part of the figure, the state diagram is presented. At the initial execution, the system enters in the state Reset, since the default transition is connected to this state. Within the state Reset, three actions are launched when the system enters in this states. The system exits Reset state when a logic '0' is applied on the input Reset, and enters in thue state Init. The system will go the state Normal when a logical '1' is applied on the input Init_OK and will stay in this state until a logic '1' is applied on the input Reset.

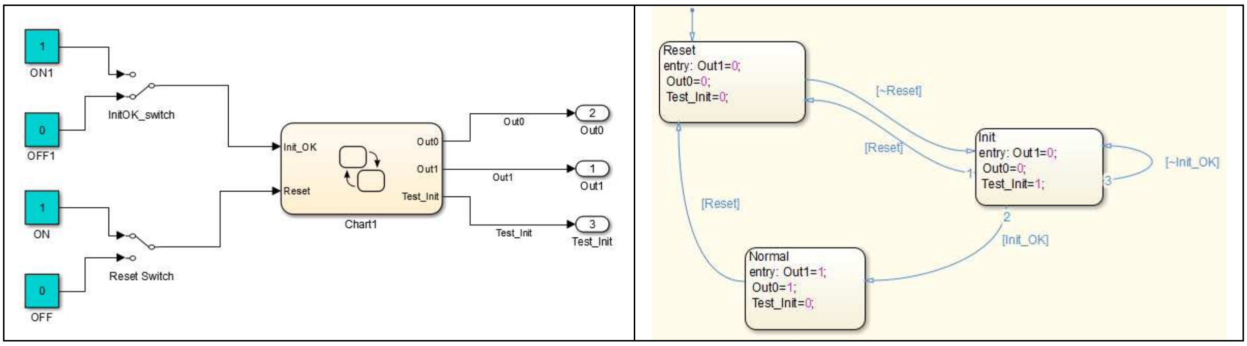


Figure 18 - Example of Stateflow diagram

Insert chart in a box:

When the diagram becomes complex, it is necessary to organize it by adding some hierarchy.

Box is the equivalent of subsystem in Simulink. Click on the button Box □ to add a box in the chart diagram. Modify its dimensions such that it surrounds the part of the chart that you want to include in the box. Give a name to the box. Right-click on it and select **Group & Subchart > Subchart** in the pop-up window. Automatically, the selected part of the chart is hidden within the box.

State diagram with predefined logical structure:

When algorithms are modeled with Simulink, they often follow a decision structure (if-else), a loop structure (for) or a switch structure. This structure can be built automatically. Click on the menu **Chart > Add Pattern in Chart** in the chart edition window. Select the used logic (decision, loop, switch, custom...), enter the conditions and the associated actions.

Exclusive or parallel execution:

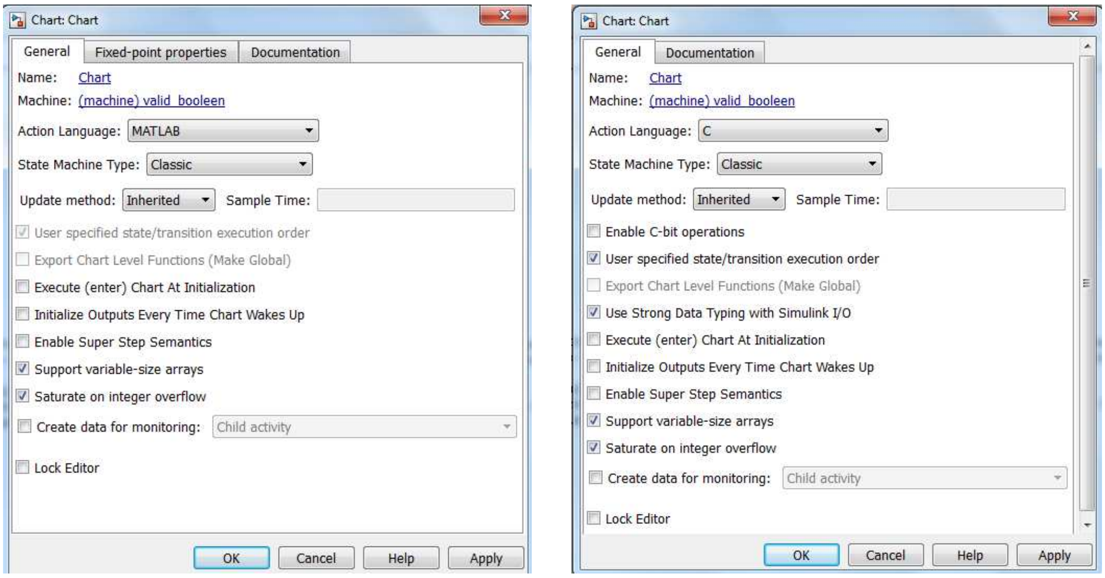
A chart can contain several subcharts. Depending on the modeled system, these subcharts can operate in simultaneously (in parallel) or exclusively (one or the other). To define such a behavior, click on the menu **Chart > Decomposition : OR (Exclusive)** or **AND (Parallel)**.

Use of boolean variables in a chart

Boolean variables can be used in Charts, but compilation errors may arise depending on the Chart configuration. Even if a chart input/output is declared as a boolean in Model Explorer, the compilation leads to the following error:

'Type mismatch in initial value. Data 'Out' is set to be of type 'boolean' and its initial value '0' evaluated to type 'double'.'

To correct this problem, the chart properties must be changed. In the Simulink diagram, right click on the chart and click on the menu **File > Model Properties > Chart Properties** (also available in Model Explorer). The window shown in Figure 19-left opens. In the General tab, in Action Language, MATLAB is selected. All the input and output are considered as double. Select 'C' in Action Language and set the box 'Use Strong Data Typing with Simulink I/O' as shown in Figure 19-right. Select also 'Enable C-bit Operations'. The compilation error will disappear.



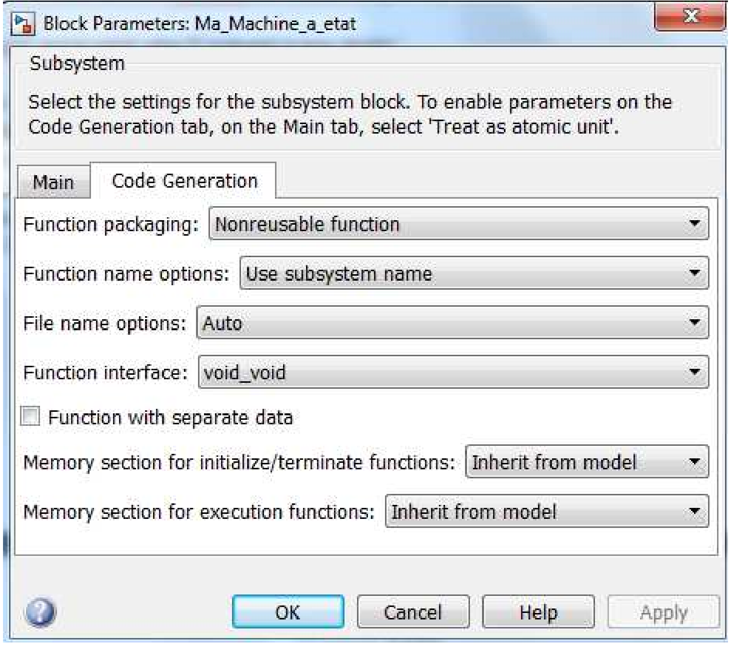
Name: Chart
Machine: (machine) valid_booleen
Action Language: MATLAB
State Machine Type: Classic
Update method: Inherited Sample Time:
☑ User specified state/transition execution order
☐ Export Chart Level Functions (Make Global)
☐ Execute (enter) Chart At Initialization
☐ Initialize Outputs Every Time Chart Wakes Up
☐ Enable Super Step Semantics
☑ Support variable-size arrays
☑ Saturate on integer overflow
☐ Create data for monitoring: Child activity
☐ Lock Editor

Name: Chart
Machine: (machine) valid_booleen
Action Language: C
State Machine Type: Classic
Update method: Inherited Sample Time:
☐ Enable C-bit operations
☑ User specified state/transition execution order
☐ Export Chart Level Functions (Make Global)
☑ Use Strong Data Typing with Simulink I/O
☐ Execute (enter) Chart At Initialization
☐ Initialize Outputs Every Time Chart Wakes Up
☐ Enable Super Step Semantics
☑ Support variable-size arrays
☑ Saturate on integer overflow
☐ Create data for monitoring: Child activity
☐ Lock Editor

Figure 19 - Modify Chart properties to prevent boolean error

Tips : Stateflow objects and code generation

In order to ensure a correct C-code generation and compilation on real hardware target, a modification of chart parameters must be done. Right-click on the chart and select **Block parameters (subsystem)** in the pop-up menu. In the part Code Generation, in the Function Packaging list, the parameter is set to Auto by default. Modify the parameters as shown below to ensure a correct compilation.



Subsystem

Select the settings for the subsystem block. To enable parameters on the Code Generation tab, on the Main tab, select 'Treat as atomic unit'.

Function packaging: Nonreusable function
Function name options: Use subsystem name
File name options: Auto
Function interface: void_void
☐ Function with separate data
Memory section for initialize/terminate functions: Inherit from model
Memory section for execution functions: Inherit from model

Figure 20 - Chart parameters to ensure a correct compilation

c. Create Flow chart

« Flow Charts » or flux diagrams are an alternative to state diagrams. They are convenient to describe basic algorithm structures as alternative choice (if then else, switch…) and iteration (for, while). Contrary to state diagrams, there are no state, but only transitions and junctions between them. One junction constitutes a pseudo-state. Figure 21 presents a simple example of flow chart, with five junctions. The default transition is visible on the top left part of the diagram. Depending on the sign of the input variable u, the system runs the '1' or '2' transition and its output takes the value 0 or 1000. The system goes into the junction in the bottom left junction.

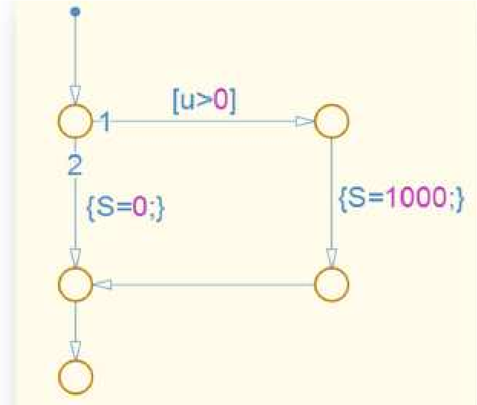


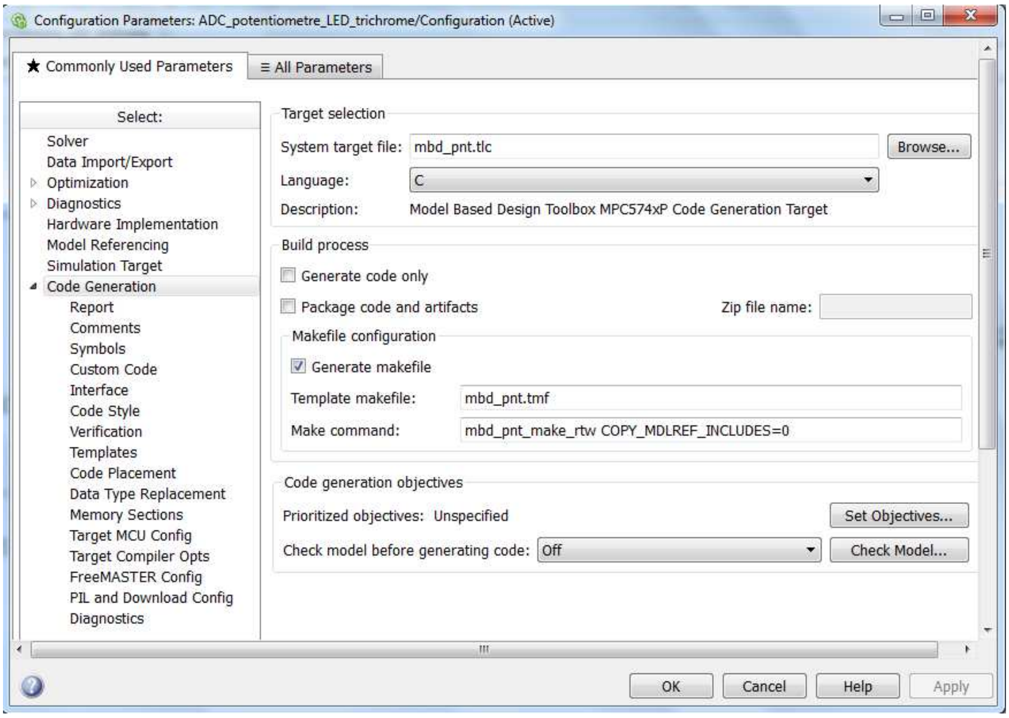
Figure 21 - Example of flow chart

Flow charts are built exactly the same manner than state diagram, except that junctions are used instead of states. No actions can be associated to a junction

8. Generating C code and flashing MPC5744P

Simulink can generate C-code source for hardware target if toolbox associated to the target exists and if a link with the target compiler has been created. Thus, Simulink becomes a co-simulation tool. Moreover, if link is done with a flashing tool, the target can be flashed directly from Simulink.

With MPC5744P, the library Model Based Design Toolbox (MBDT) offers all the libraries and configuration tool of this microcontroller. Refer to part IV for more details about MBDT. A target configuration block IV-2-a must be placed on the Simulink diagram. The parameters for the compilation, code building and connections with the MPC5744P are available from this block, or from **Simulation > Model Configuration Parameters** or the icon . Figure 22 shows the configuration window. All the parameters that you have to configure are in the part Code Generation.



Configuration Parameters: ADC_potentiometre_LED_trichrome/Configuration (Active)

Target selection

System target file: mbd_pnt.tlc

Language: C

Description: Model Based Design Toolbox MPC574xP Code Generation Target

Build process

□ Generate code only

□ Package code and artifacts   Zip file name:

Makefile configuration

☑ Generate makefile

Template makefile: mbd_pnt.tmf

Make command: mbd_pnt_make_rtw COPY_MDLREF_INCLUDES=0

Code generation objectives

Prioritized objectives: Unspecified

Check model before generating code: Off

Figure 22 - Configuration Parameters window - Code generation properties

The most basic configurations are:

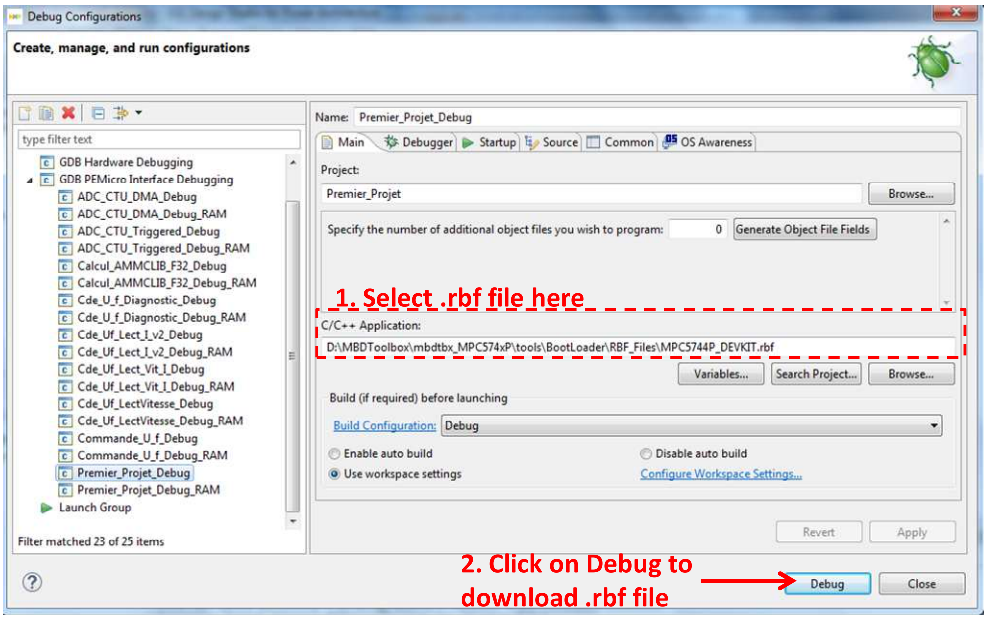
- in Code generation part, verify that C is selected in Language list
- in Report, if you check "Create code generation report", a report listing all the generated C-code files will open at the end of the building process.
- in Target MCU, select the target microcontroller (MPC5744P), its package (144LQFP), the core frequency (200 MHz), the crystal frequency (40 MHz). Interrupt priority parameters can be let by default.
- Target Compiler Opts: verify that S32 Design Studio is selected in the list "Compiler Selection". In the list Target Memory, select FLASH or SRAM depending on the target memory.
- FreeMASTER Config: enable FreeMASTER and configure UART physical I/O pads and baud rate (refer to part IV.2.a for more information about FreeMASTER configuration with MBDT toolbox).
- PIL and Download Config: Check "Enable Download Code after Build" if you want to launch the code downloading after the code building process. Define the COM port and the baud rate (115200 Bds by default).
- Diagnostics: select which MCU peripherals are verified during compilation. In particular case where errors is related to Simulink, it can be useful to deactivate the diagnostics of MCU peripherals.

Before building and flashing the microcontroller, you have to ensure that a bootloader application for MPC5744P is installed. The required tool is **RAppID_BL.exe**, which is installed with MBDT toolbox. RAppID application is available in MBDT toolbox directory (\MBDToolbox\mbdtbx_MPC574xP\tools\BootLoader).

<u>Tips : installation of .rbf file</u>

A simple operation must be performed to download code from this tool: a bootloader file .rbf must be downloaded in the microcontroller first. For the development kit DEVKIT-MPC5744P used in this project, select the file MPC5744P_DEVKIT.rbf, available in the

MBDT toolbox directory (\MBDToolbox\mbdtbx_MPC574xP\tools\BootLoader\RBF_Files\). The easiest way to do that is to use S32DS. Create a new S32DS project as explained in I.2. The project can be blank, it does not matter. Connect the MPC5744P target on the host computer and click on the menu **Run > Debug configurations**. In the window shown in Figure 23, modify the C/C++ Application and select the file MPC5744P_DEVKIT.rbf. Click on the button Debug to download the bootloader file in the microcontroller. Wait the end of the downloading process. It is finished, you can close S32DS and use RAppID to flash the microcontroller from Simulink.



Debug Configurations

Create, manage, and run configurations

type filter text

- GDB Hardware Debugging
- GDB PEMicro Interface Debugging
  - ADC_CTU_DMA_Debug
  - ADC_CTU_DMA_Debug_RAM
  - ADC_CTU_Triggered_Debug
  - ADC_CTU_Triggered_Debug_RAM
  - Calcul_AMMCLIB_F32_Debug
  - Calcul_AMMCLIB_F32_Debug_RAM
  - Cde_U_f_Diagnostic_Debug
  - Cde_U_f_Diagnostic_Debug_RAM
  - Cde_Uf_Lect_I_v2_Debug
  - Cde_Uf_Lect_I_v2_Debug_RAM
  - Cde_Uf_Lect_Vit_I_Debug
  - Cde_Uf_Lect_Vit_I_Debug_RAM
  - Cde_Uf_LectVitesse_Debug
  - Cde_Uf_LectVitesse_Debug_RAM
  - Commande_U_f_Debug
  - Commande_U_f_Debug_RAM
  - Premier_Projet_Debug
  - Premier_Projet_Debug_RAM
- Launch Group

Filter matched 23 of 25 items

Name: Premier_Projet_Debug

Main | Debugger | Startup | Source | Common | OS Awareness

Project:
Premier_Projet

Specify the number of additional object files you wish to program: 0

1. Select .rbf file here

C/C++ Application:
D:\MBDToolbox\mbdtbx_MPC574xP\tools\BootLoader\RBF_Files\MPC5744P_DEVKIT.rbf

Build (if required) before launching

Build Configuration: Debug

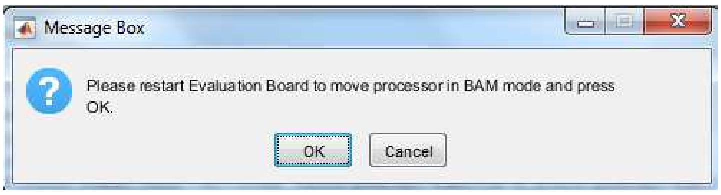
○ Enable auto build ○ Disable auto build

● Use workspace settings

2. Click on Debug to download .rbf file

Figure 23 - Downloading of the .rbf file (S32DS - Run > Debug configurations)

In Simulink, click on the menu **Code > C/C++ code > Build Model** or on the icon to build the application C-code and flashing the microcontroller. The compilation and building process can take 30 seconds/1 minute depending on the size of the project. At the end of the building, RAppID is automatically launched. A message is displayed, as shown in Figure 24, which asks you to reset the microcontroller to force it in bootloading mode. Press the button Reset of the MPC5744 development kit (button SW3) for 1 or 2 seconds, and, rapidly, click on the button OK. The downloading process starts (RAppID BL progress bar). At the end, the microcontroller is flashed correctly. Tools as FREEMASTER (see part III) can be used to monitor in real time the execution of the embedded code.



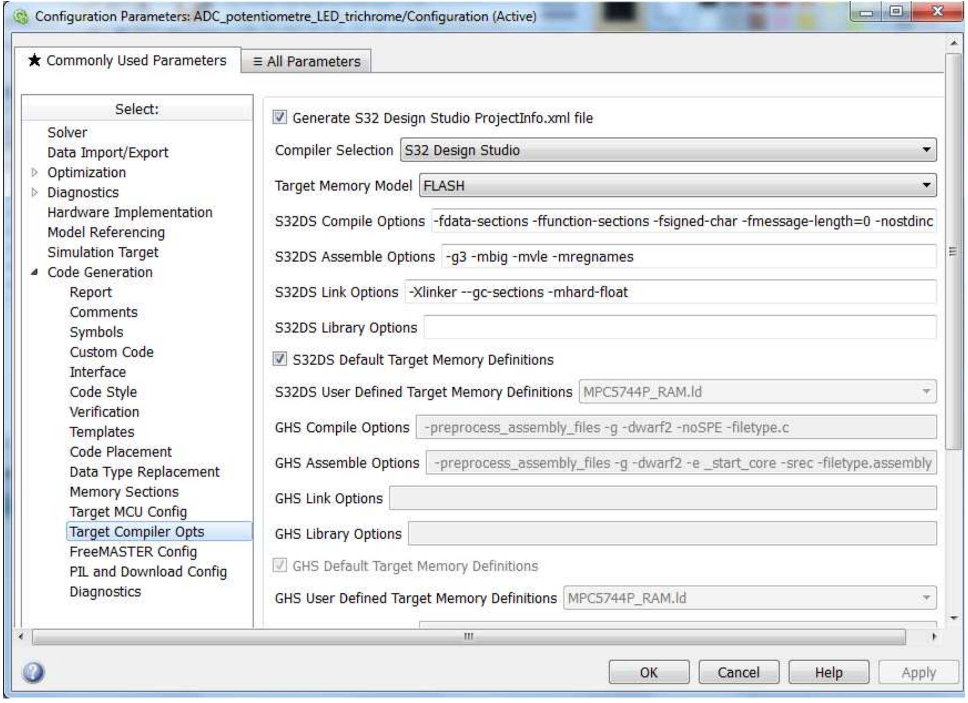
Message Box

Please restart Evaluation Board to move processor in BAM mode and press OK.

Figure 24 - Download the C-code application with RAppID

9. Importing C code into S32DS

If you want to create a S32DS project from the source code files generated by Simulink, the creation of the project can be quite tedious because Simulink may generate tens of .c and .h files. There is a simple method to create a new S32DS project from the C-code files generated by Simulink. In **Configuration Parameters**, in the part "Target compiler Opts", select the option "Generate S32 Design studio ProjectInfo.xml file" (Figure 25). At the end of the building process, a XML file is generated which lists all the required the .c/.h files to generate a new project with another IDE.



☑ Generate S32 Design Studio ProjectInfo.xml file

Compiler Selection: S32 Design Studio

Target Memory Model: FLASH

S32DS Compile Options: -fdata-sections -ffunction-sections -fsigned-char -fmessage-length=0 -nostdinc

S32DS Assemble Options: -g3 -mbig -mvle -mregnames

S32DS Link Options: -Xlinker --gc-sections -mhard-float

S32DS Library Options:

☑ S32DS Default Target Memory Definitions

S32DS User Defined Target Memory Definitions: MPC5744P_RAM.ld

GHS Compile Options: -preprocess_assembly_files -g -dwarf2 -noSPE -filetype.c

GHS Assemble Options: -preprocess_assembly_files -g -dwarf2 -e _start_core -srec -filetype.assembly

GHS Link Options:

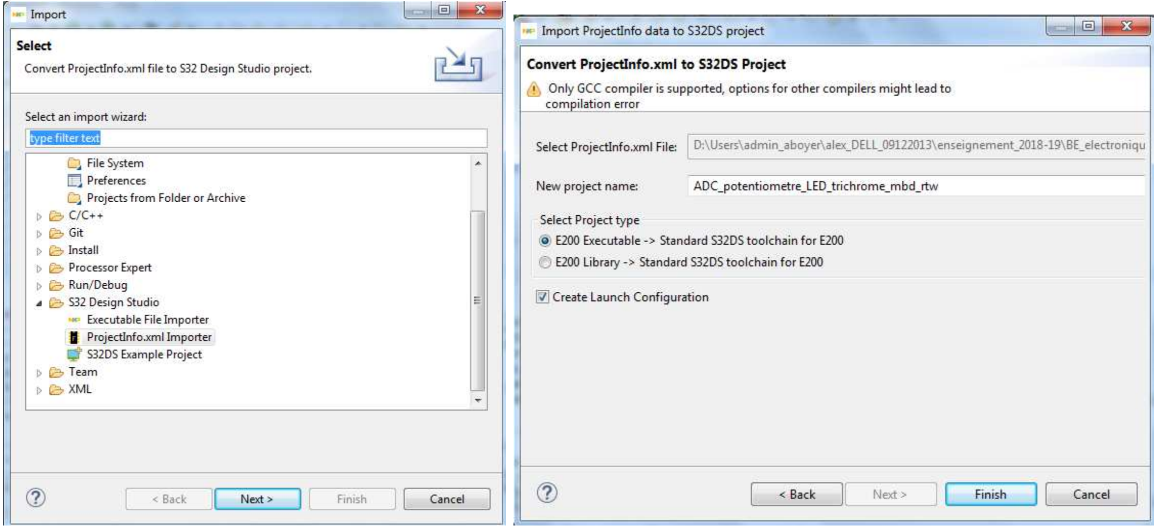
GHS Library Options:

☑ GHS Default Target Memory Definitions

GHS User Defined Target Memory Definitions: MPC5744P_RAM.ld

Figure 25 - Exporting C-code source file generated by Simulink to S32DS based on .xml file

In S32DS, click on the menu **File > Import**. The window shown in Figure 26-left appears. Select **ProjectInfo.xml Importer** and click on **Next**. In the window shown in Figure 26-right, select the .xml file, give a name to the new project and select E200 Executable. Click on Finish. After several seconds, a new S32DS project is created with all the executable files generated by Simulink.



Convert ProjectInfo.xml file to S32 Design Studio project.

Select an import wizard: type filter text

Convert ProjectInfo.xml to S32DS Project

Only GCC compiler is supported, options for other compilers might lead to compilation error

Select ProjectInfo.xml File: D:\Users\admin_aboyer\alex_DELL_09122013\enseignement_2018-19\BE_electroniqu

New project name: ADC_potentiometre_LED_trichrome_mbd_rtw

Select Project type

● E200 Executable -> Standard S32DS toolchain for E200

○ E200 Library -> Standard S32DS toolchain for E200

☑ Create Launch Configuration

Figure 26 - Importing C-code source file generated by Simulink in S32DS

III - Presentation of FREEMASTER

1. Overview

FREEMASTER is a PC-based development tool serving as a real-time monitor, visualization tool, and graphical control panel of embedded applications implemented on NXP microcontroller, as described in Figure 27. The FREEMASTER application repetitively sends a request to obtain the current values of chosen variables used in the embedded application and display them on a graphical interface. Communication between FREEMASTER application is supported by serial communication interface (SCI) such as UART, CAN bus or JTAG. In this document, we will only consider communication through UART (based on LIN interface). In the MPC5744P_DEVKIT, communication will pass transit through the USB port of the OpenSDA interface.

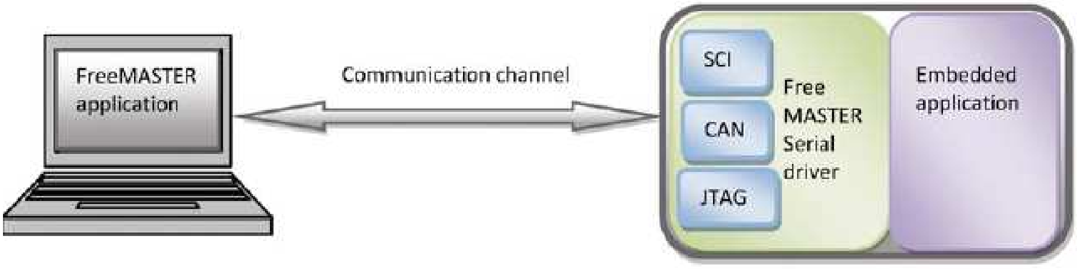


Figure 27 - Freemaster principle

However, the communication requires FREEMASTER serial driver, that must be used by the embedded application to ensure protocol functions and handle peripherals. This part aims at:

- describing the main functions of FREEMASTER serial driver to enable FREEMASTER communication either in S32DS or Simulink project
- presenting the interface of FREEMASTER application

FREEMASTER application and serial driver can be downloaded from NXP website (www.nxp.com) without any charge. Refer to part VI for installation of FREEMASTER application and serial drivers.

2. Adding FREEMASTER communication driver to S32DS project

a. The main macros and functions of FREEMASTER API

All the details about the functions and macros of FREEMASTER driver API can be found in [FRUG].
The configuration of FREEMASTER is done by freemaster_cfg.h file, through several macros. Here, only the most important are described here:

- Interrupt mode selection: assert to '1' only one the three macros below. If long interrupt mode is used, the function FMSTR_Isr() handles FreeMASTER protocol decoding and execution. As it can be a long process, give it a low interrupt priority level. In short interrupt mode (the most versatile mode), the raw serial communication is handled by the FMSTR_Isr() interrupt service routine, while the protocol decoding and execution is handled in the FMSTR_Poll() routine. In polling mode, Both the serial (SCI/CAN) communication and the FreeMASTER protocol execution are done in the FMSTR_Poll() routine.

```
#define FMSTR_LONG_INTR 0 /* complete message processing in interrupt */
#define FMSTR_SHORT_INTR 1 /* only SCI FIFO - queuing done in interrupt */
#define FMSTR_POLL_DRIVEN 0 /* no interrupt needed, polling only */
```

- Selection of communication interface: FMSTR_DISABLE must be set to '0' (default value) to enable FREEMASTER functionalities. FREEMASTER communication is based either on SCI (LINFlexD_0 or 1), CAN or BDM communication interface. To select one of these interfaces, set the corresponding macros to '1'.

```
#define FMSTR_DISABLE          0    /* To disable all FreeMASTER functionalities */
#define FMSTR_USE_SCI          1    /* To select SCI communication interface */
#define FMSTR_USE_FLEXCAN      0    /* To select FlexCAN communication interface */
#define FMSTR_USE_PDBDM        0    /* To select Packet Driven BDM communication
interface (optional) */
```

- Definition of communication interface memory address for SCI and CAN interface: refer to memory address map of the microcontroller and write the starting memory address corresponding to communication interface. Any errors in memory address will result in unpredictable application error.

```
#define FMSTR_SCI_BASE         0xFFE90000UL /* LINFlex1 base on MPC574xP */
#define FMSTR_CAN_BASE         0xFFEC0000UL /* FlexCAN0 base on MPC574xP */
```

For the other macros, the default values are sufficient for the application developed in the Automotive Electronics lab.

FREEMASTER API contains numerous functions. However, in order to initialize and launch communication with FREEMASTER application, only three functions are required, which have to be called by the embedded C code:

- FMSTR_init(): it initializes internal variables of the FreeMASTER driver and enables the communication interface (SCI, JTAG or CAN). It does not change the configuration of the selected communication module. The user must initialize the communication module (LINFlex as UART, JTAG or CAN) before the FMSTR_Init() function is called.
- FMSTR_Poll(): in poll-driven or short interrupt modes, this function handles the protocol decoding and execution. In the poll-driven mode, this function also handles the interface communication with the PC. Typically, FMSTR_Poll() is called during the 'idle' time in the main application loop.
- FMSTR_Isr(): it is the interface to the interrupt service routine of the FreeMASTER serial driver. In long or short interrupt modes, this function must be set as the interrupt vector calling address when a transmission or reception is performed by the communication module (LIN, CAN or JTAG). On platforms where interface processing is split into multiple interrupts, this function should be set as a vector for each such interrupt.

Besides, two additional functions can be used if the recorder functionality is used:

- FMSTR_Recorder(): it takes one sample of the variables being recorded using the FreeMASTER recorder. If the recorder is not active at the moment when FMSTR_Recorder is called, the function returns immediately. When the recorder is initialized and active, the values of the variables being recorded are copied to the recorder buffer and the trigger condition is evaluated.
- FMSTR_TriggerRec(): it forces the recorder trigger condition to happen, which causes the recorder to be automatically de-activated after post-trigger samples are sampled.

This function can be used in the application when it needs to have the trigger occurrence under its control. This function is optional. The recorder can also be triggered by the PC tool or when the selected variable exceeds a threshold value.

It is not necessary to indicate which variables will be transferred from the MCU to the FREEMASTER PC-application. All the global variables can be transferred to FREEMASTER application, if these variables have been selected to be watched.

b. Configuration of FREEMASTER driver in C code application

FREEMASTER serial drivers are provided as a SDK. In order to use FREEMASTER communication in a new S32DS project, FREEMASTER SDK has to be imported first. Refer to part I.7 for this action.
The header file freemaster_cfg.h is automatically added in the folder include of S32DS project. The macros should be updated following the explanation given in part III.2.a. Include the header file freemaster.h in all the source code file where a FREEMASTER API function is used.

The four main steps are:

1. Configure all the necessary peripherals required for the FREEMASTER communication (clock gating, interrupt controller, initialization of the UART (SCI or CAN) and the used external pins). Ensure that the UART, its pins and its timing parameters are correctly set.
2. Initialize FREEMASTER by calling FMSTR_Init() just once at the code start, typically after the start-up code, at the beginning of the main function.
3. Call FMSTR_Poll(void) periodically in your code. A typical place is in the main loop.
4. FMSTR_Isr() must be assigned to the used UART interrupt vectors (e.g. interrupt vectors associated to transmission and reception of the used UART). Configure also the interrupt priority level associated to these interrupt requests. Use a low priority level to ensure that FREEMASTER will not affect your application. In S32DS project, the interrupt vectors are defined in the file intc_SW_mode_isr_vectors_MPC5744P.c.

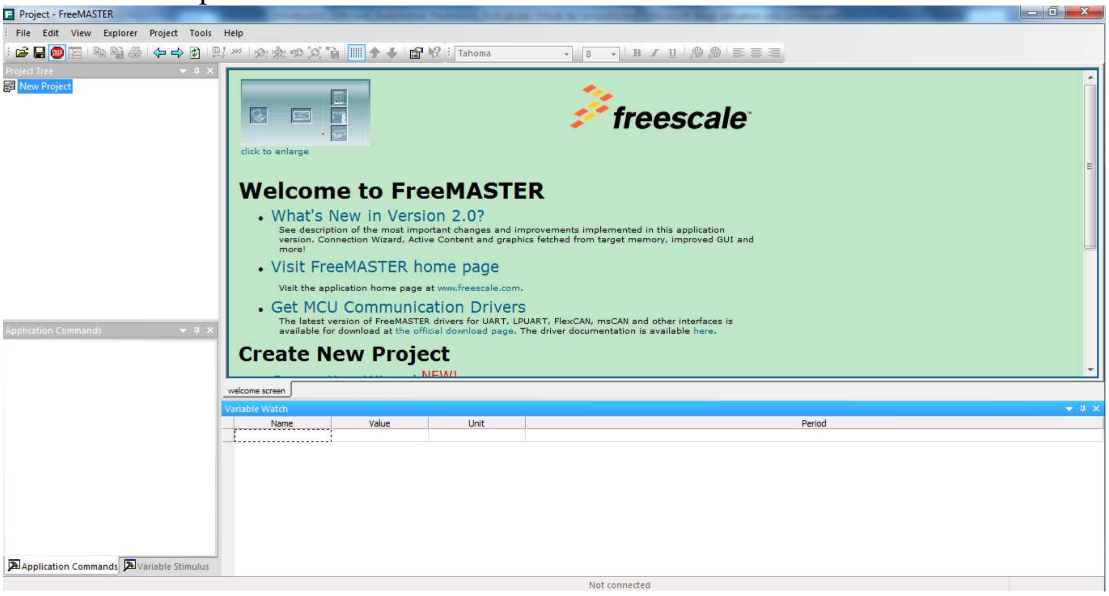
3. Configuring FREEMASTER communication driver in Matlab/Simulink

Although the code compiled and the project was built, it is not a guarantee that your application is functional. With in-situ debug, provided by S32DS IDE, it can be verified by step-by-step running. Contrary to S32DS IDE, Simulink does not offer in-situ debugging options. FREEMASTER is the only tool to perform this debug by reading in real time internal variables.
With Model Based Design Toolbox for MPC574xP, the configuration of FREEMASTER is straightforward since it consists in enabling it in the Target Configuration Block and adding a block in Simulink diagram for data recorder. More details will be given in part IV.

4. Configuring FREEMASTER application

Connect the microcontroller to a USB port of your PC through. Ensure that the microcontroller has been flashed and is powered correctly.

Click on FREEMASTER 2.0 icon to launch the FREEMASTER application. The following window opens. By default, no project is loaded. Here, the different steps to start communication with an embedded program in a microcontroller and to visualize internal variables will be explained.



Welcome to FreeMASTER

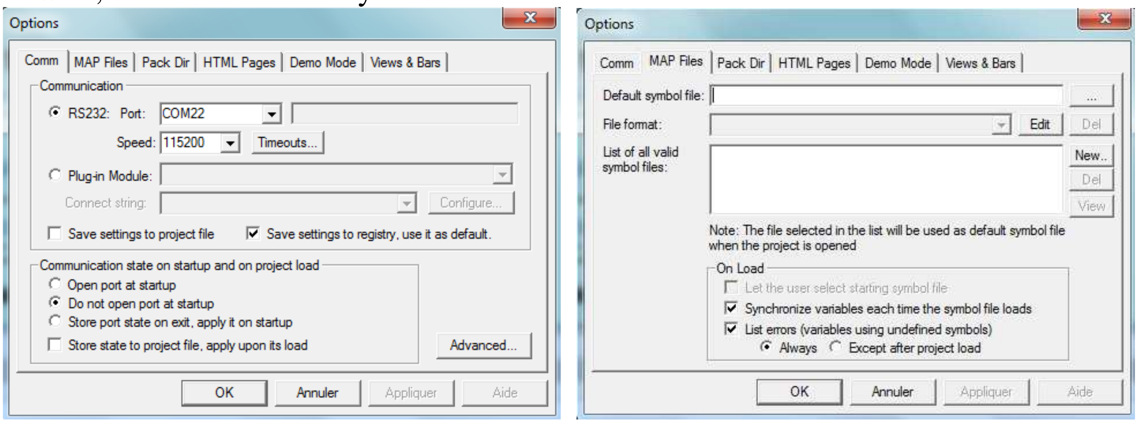
- What's New in Version 2.0?
  See description of the most important changes and improvements implemented in this application version. Connection Wizard, Active Content and graphics fetched from target memory, improved GUI and more!
- Visit FreeMASTER home page
  Visit the application home page at www.freescale.com.
- Get MCU Communication Drivers
  The latest version of FreeMASTER drivers for UART, LPUART, FlexCAN, msCAN and other interfaces is available for download at the official download page. The driver documentation is available here.

Create New Project

Figure 28 - FREEMASTER application - main window

In the menu bar, click on **Project > Options**. The window shown below appears. Only two operations are required to configure the communication. First, you have to set the parameters of the communication port. In the tab **Comm.**, select the communication port on which the microcontroller board is connected. If you do not know it, in Window start menu, go to **Control Panel > Device manager > Ports (COM & LPT)** and find the number of the Com port. Then, set the correct baud rate. Secondly, the executable file (.elf) embedded in the microcontroller must be provided to FREEMASTER to make the link with variables read continuously. In the tab MAP Files, select the .elf file in the field **Default symbol file**.

Tips: ensure that the .elf file corresponds to the actual code embedded in the microcontroller. Otherwise, communication may fail.



Communication
● RS232: Port: COM22
Speed: 115200
○ Plug-in Module:
Connect string:
☐ Save settings to project file ☑ Save settings to registry, use it as default.

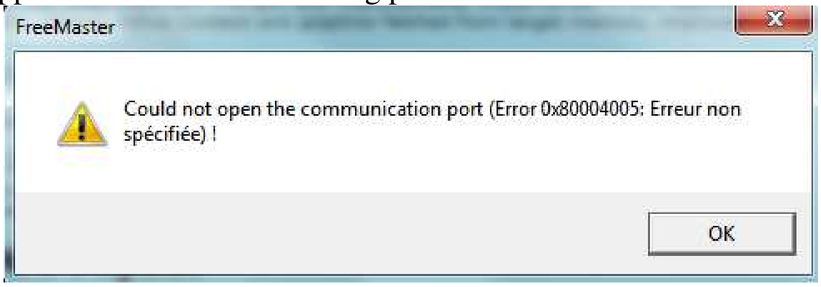
Communication state on startup and on project load
○ Open port at startup
● Do not open port at startup
○ Store port state on exit, apply it on startup
☐ Store state to project file, apply upon its load

Default symbol file:
File format:
List of all valid symbol files:
Note: The file selected in the list will be used as default symbol file when the project is opened
On Load
☐ Let the user select starting symbol file
☑ Synchronize variables each time the symbol file loads
☑ List errors (variables using undefined symbols)
● Always ○ Except after project load

Figure 29 - FREEMASTER application - configuration of the communication port (on the left) and selection of the executable file (on the right)

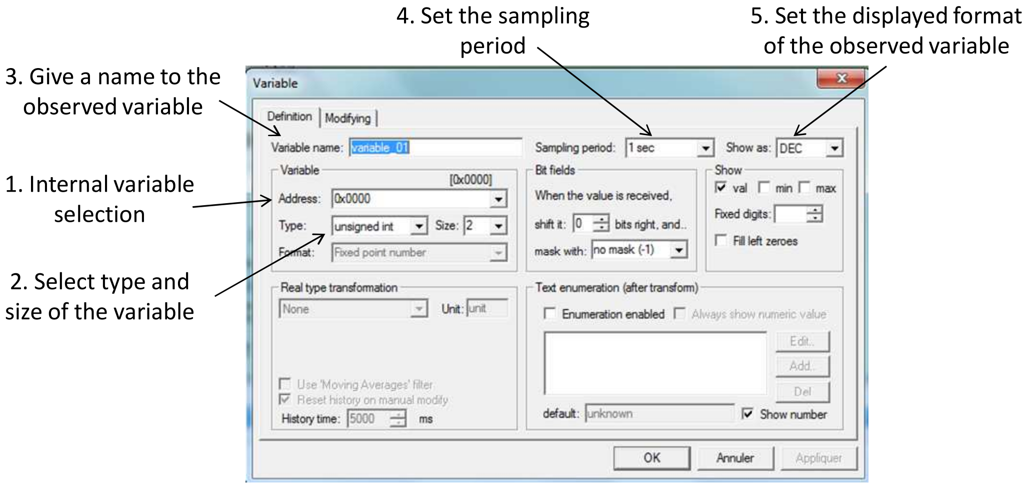
Click on the button OK. A new project was created, visible in the Project tree. Click in the menu **File > Save Project** or on icon 💾 to save it. The extension of the file is .pmp. The communication port configuration and visualized variables are saved. The next time you will connect to the microcontroller, you will import the .pmp file directly.

In order to launch the communication with the microcontroller, click on the button **Start/stop communication** 🛑. The status of the communication is displayed in the message bar, in the bottom part of the main window. If the communication is active, the COM port, and the baud rate must be written. Otherwise, the message "Not connected" is written and an error message shown below appears. Refer to the following part to solve this issue.



Could not open the communication port (Error 0x80004005: Erreur non spécifiée) !

Once the .elf file has been loaded, the variables to be visualized can be selected. In the table **Variable Watch**, right click and select **Create New Watched Var…** in the pop-up menu. The following window appears. In the list **Type**, select the type of the variable. In the list **Address**, select the variable to be visualized. Give it an arbitrary name in **Variable name**. Set the **Sampling period**. If the visualization must refreshed as fast as possible, select Fastest. Select also the format of the visualized variable (decimal, hexadecimal, binary…) in the list **Show as**.



4. Set the sampling period

5. Set the displayed format of the observed variable

3. Give a name to the observed variable

1. Internal variable selection

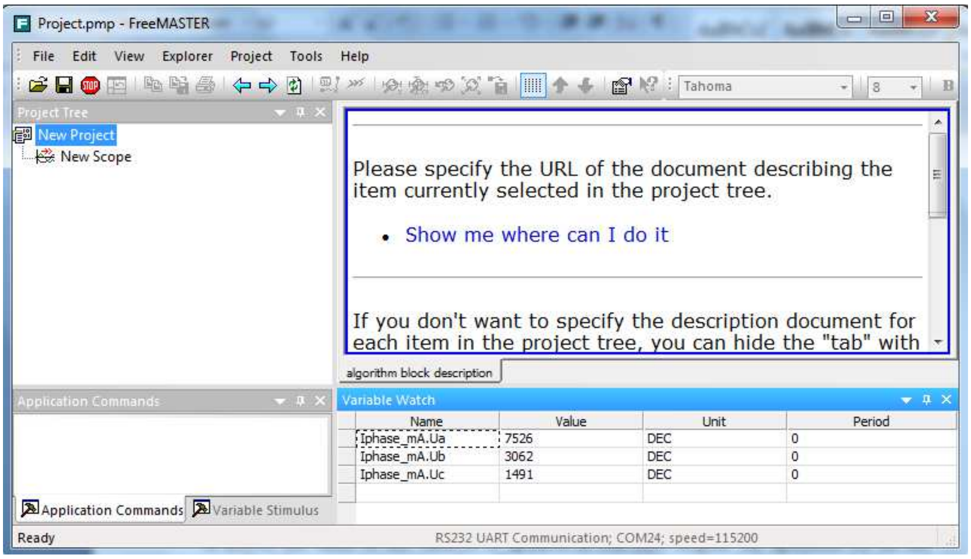
2. Select type and size of the variable

Variable name: variable_01
Sampling period: 1 sec
Show as: DEC
[0x0000]
Address: 0x0000
Type: unsigned int
Size: 2
Format: Fixed point number
When the value is received, shift it: 0 bits right, and.. mask with: no mask (-1)
☑ val ☐ min ☐ max
Fixed digits:
☐ Fill left zeroes
None
Unit: unit
☐ Use 'Moving Averages' filter
☑ Reset history on manual modify
History time: 5000 ms
☐ Enumeration enabled ☐ Always show numeric value
default: unknown
☑ Show number

Figure 30 - Selection of a new variable to be visualized with FREEMASTER

Click on OK. The new variable is added in the table **Variable Watch**, as shown in Figure 31 where three variables have been added in the Variable Watch panels. When the communication is active, they are updated in real-time.

Repeat the operation for all the variables that you want to visualize. To delete one variable, right click on the variables and select **Remove from watch** in the pop-up menu. To modify it, select **Edit variable**.



Please specify the URL of the document describing the item currently selected in the project tree.

- Show me where can I do it

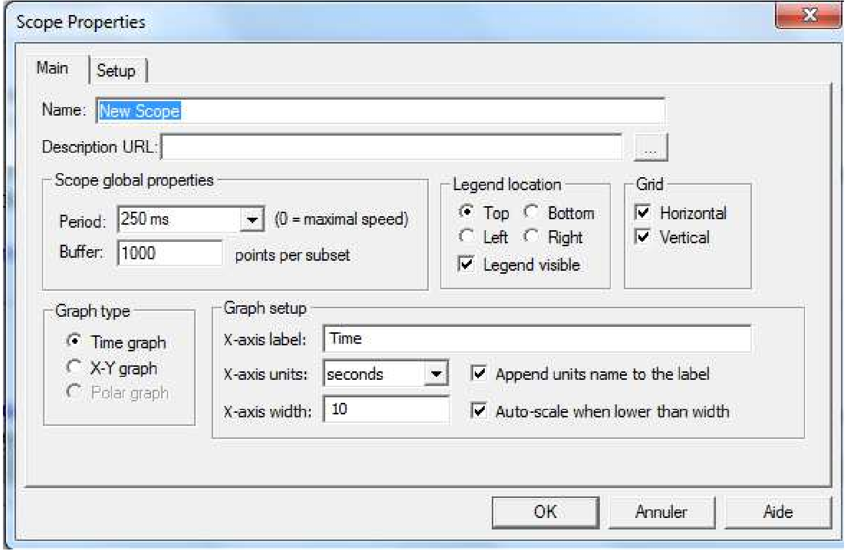
If you don't want to specify the description document for each item in the project tree, you can hide the "tab" with

Name	Value	Unit	Period
Iphase_mA.Ua	7526	DEC	0
Iphase_mA.Ub	3062	DEC	0
Iphase_mA.Uc	1491	DEC	0

Ready RS232 UART Communication; COM24; speed=115200

Figure 31 - Example of real-time observation of three variables with Freemaster

FREEMASTER application proposes also to visualize time domain evolution of variables in a 2D graph (called Scope). In Project tree, right click on the project name and select **Create scope** in the pop-up menu. In the tab **Main**, define the name of the Scope in the field **Name**. Specify the sampling period in the field **Period**. Define the number of points visualized in a scope in **Buffer**. You can also modify general graphical properties of the Scope.



Name: New Scope

Description URL:

Period: 250 ms (0 = maximal speed)

Buffer: 1000 points per subset

● Top ○ Bottom
○ Left ○ Right
☑ Legend visible

☑ Horizontal
☑ Vertical

● Time graph
○ X-Y graph
○ Polar graph

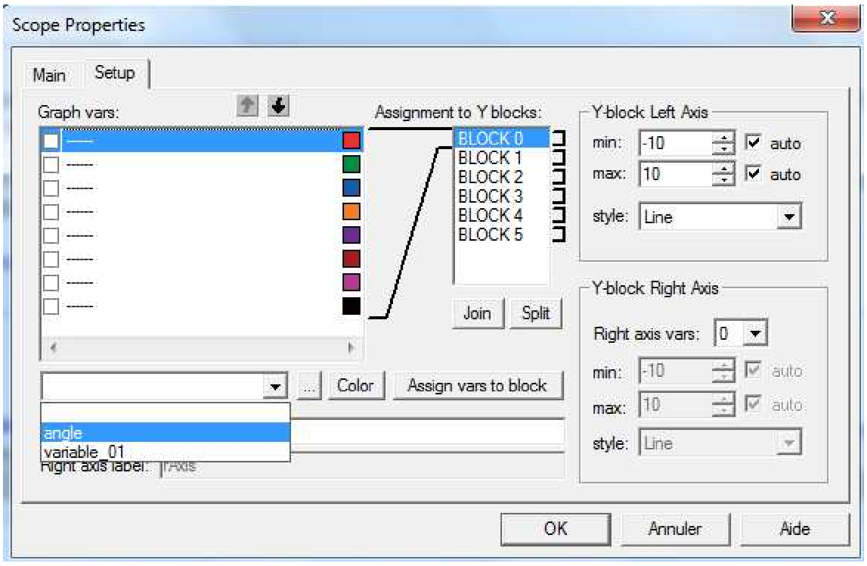
X-axis label: Time

X-axis units: seconds ☑ Append units name to the label

X-axis width: 10 ☑ Auto-scale when lower than width

Figure 32 - Properties of a scope

Then, select the variables that you want to visualize. Only watched variables can be displayed in a scope. Click on the tab Setup. The window shown below is displayed. **Graph vars** lists all the variables to be plotted In the list below, select the watched variables to be plotted. The variable appears in **Graph Vars** list. On the right, the list **Assignment to Y blocks** is visible. A block is a subgraph. You can plot as many variables in a block. If you want to plot two variables in two different blocks, select the variables, the block and click on the button **Assign vars to block**. You can modify general properties of curves and blocks in this screen.



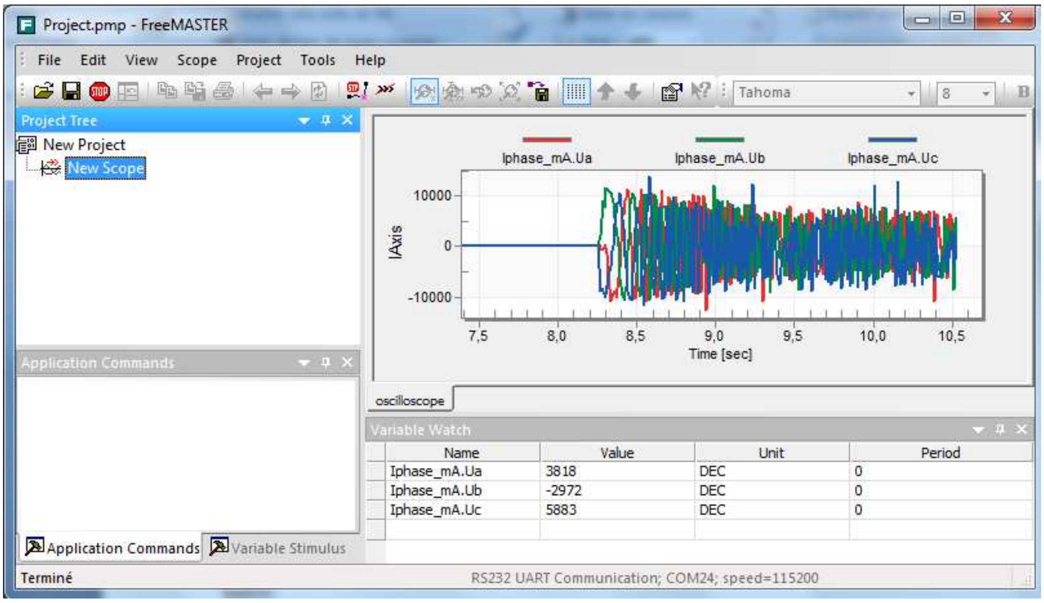
Y-block Left Axis
min: -10 ☑ auto
max: 10 ☑ auto
style: Line

Y-block Right Axis
Right axis vars: 0
min: -10 ☑ auto
max: 10 ☑ auto
style: Line

angle
variable_01
Right axis label: lAxis

Figure 33 - Selection of plotted variables in a Scope

Click on OK. The graph appears. If the communication is active, the selected watched variables are plotted directly. Their evolution is plotted in real-time, as shown in the figure below.



Name	Value	Unit	Period
Iphase_mA.Ua	3818	DEC	0
Iphase_mA.Ub	-2972	DEC	0
Iphase_mA.Uc	5883	DEC	0

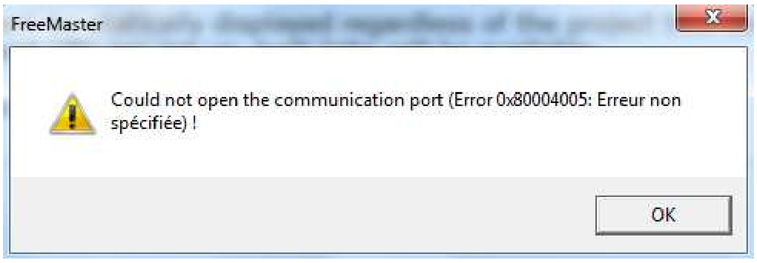
Terminé
RS232 UART Communication; COM24; speed=115200

Figure 34 - Observation of the time-domain evolution of three variables in a Scope

Do not forget to save the project before closing FREEMASTER application.

5. Debug FREEMASTER

Wrong configurations will result in loss of communications between the MCU board and FREEMASTER PC application, shown by this error message :



FreeMaster

Could not open the communication port (Error 0x80004005: Erreur non spécifiée) !

Freemaster is usually not able to communicate with the board in the five following situations:

- FREEMASTER drivers have not been called in the embedded application (or incorrectly called)
- the associated UART has not been correctly configured (baud rate, I/O pads not configured, …)
- FMSTR_Isr() function has not been linked to interrupts vectors associated to the UART (for TX and RX)
- port and baud rate specified in FREEMASTER application are wrong
- the communication port between your PC and the microcontroller board is already used by another application (e.g. S32DS in-situ debugger).
- the embedded application was compiled and built, but it results in wrong operation (e.g. crash due to a critical interrupt request).

If the first four reasons have been verified, then you can conclude that FREEMASTER is correctly configured, but your embedded application is not operational.

IV - Presentation of Model Based Design Toolbox (MBDT) for MPC574xP

This Matlab/Simulink toolbox is compatible only from R2015.a release. In this document, the version 2.0.0 for MPC574xP MCU is considered. Equivalent toolboxes exist for other NXP's MCU such as S32K, MPC564xL, MPC7xK, Kinetis KV1x and KV3x, S12ZVM…
Model-Based Design Toolbox can be downloaded here: http://www.nxp.com/support/developer-resources/run-time-software/automotive-software-and-tools/model-based-design-toolbox:MC_TOOLBOX

1. Overview

The NXP's Model-Based Design Toolbox (MBDT) provides an integrated development environment and toolchain for configuring and generating all of the necessary software automatically (including initialization routines and device drivers) to execute complex applications (e.g. motor control algorithms, communication protocols CAN, SPI, I2C, UART, and sensor-based applications) on NXP MCUs. Its main application domain is motor control application.
The toolbox includes integrated Simulink embedded target for NXP MCUs, peripheral device blocks and drivers and provides built-in support for Software and Processor-in-the-Loop (SIL and PIL) simulations. The toolbox contains peripheral driver interface blocks, a code generation target for MPC574xP family of processors, optimized target code blocks for the

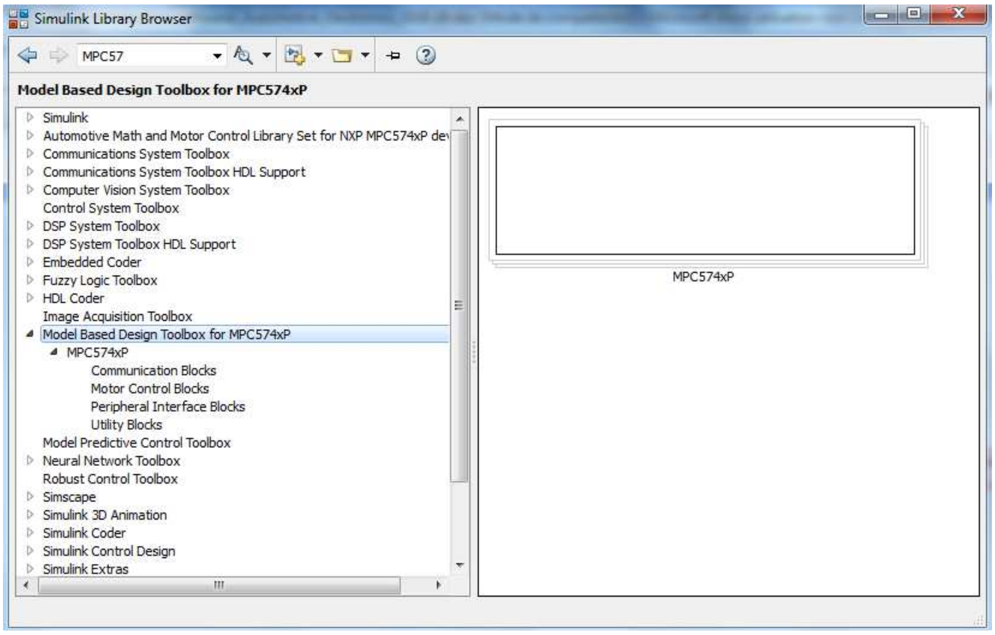
MPC574xP processor, and support for model reference Software-In-the-Loop (SIL) / Processor-In-the-Loop (PIL) code generation and co-simulation testing.
Its main features are:

- Generation of code for standalone application with direct download to target support
- Drivers for I/O blocks including CAN, SPI, PIT timer, Sine Wave Generation, eTimer, PWM and A/D
- Data acquisition and calibration using FreeMASTER tool
- Bootloader utility for programming application in flash (RAppID application)

2. *Rapid presentation of the functions*

MBDT contains configuration-dependent blocks for controlling the low level drivers and I/O of the target. These blocks are considered configuration-dependent because the MBD_MPC574xP_Config_Information block must be used when any of the other blocks in the library is used within the model. In some cases, there are additional configuration blocks that must be used as well, for example, FlexCAN Configuration, DSPI Configuration, ADC Configuration, CTU Configuration, and eTimer Configuration. The blocks within the Model Based Design Toolbox library are not supported in all modes.

In Simulink, MBDT blocks can be found in Library Browser, as shown in Figure 35. The different blocks are organized in four categories, which are briefly described in the following part. More details can be found in [MBDT]. The configuration of these blocks requires a knowledge of the MPC5744P peripherals and associated registers. It is recommended to refer to the MPC5744P reference manual to have details about peripheral configurations.



Simulink Library Browser

MPC57

Model Based Design Toolbox for MPC574xP

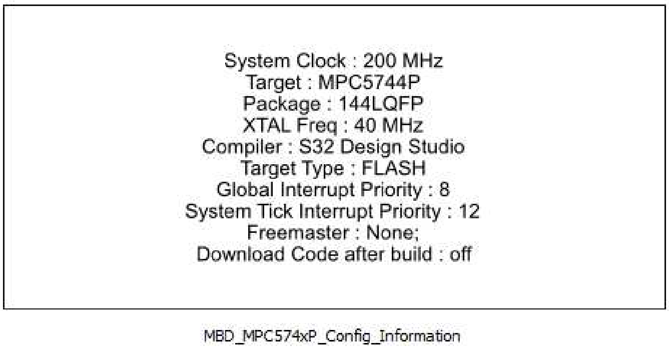
- Simulink
- Automotive Math and Motor Control Library Set for NXP MPC574xP dev
- Communications System Toolbox
- Communications System Toolbox HDL Support
- Computer Vision System Toolbox
- Control System Toolbox
- DSP System Toolbox
- DSP System Toolbox HDL Support
- Embedded Coder
- Fuzzy Logic Toolbox
- HDL Coder
- Image Acquisition Toolbox
- Model Based Design Toolbox for MPC574xP
  - MPC574xP
    - Communication Blocks
    - Motor Control Blocks
    - Peripheral Interface Blocks
    - Utility Blocks
- Model Predictive Control Toolbox
- Neural Network Toolbox
- Robust Control Toolbox
- Simscape
- Simulink 3D Animation
- Simulink Coder
- Simulink Control Design
- Simulink Extras

Figure 35 - MBDT in Simulink's Library Brower

a. Target configuration block

This block is available in the root of the MBDT library (MPC574xP). It is dedicated to the configuration of MPC574xP MCU. This block actually opens the **Configuration Parameters Dialog** ⚙, which gives access to simulation, code generation, verification and downloading configuration. The Configuration Parameters dialog contains the following specific panes for MPC5744P target configuration:

- Target MCU
- Target Compiler Opts
- FreeMASTER Config
- PIL and Download Config

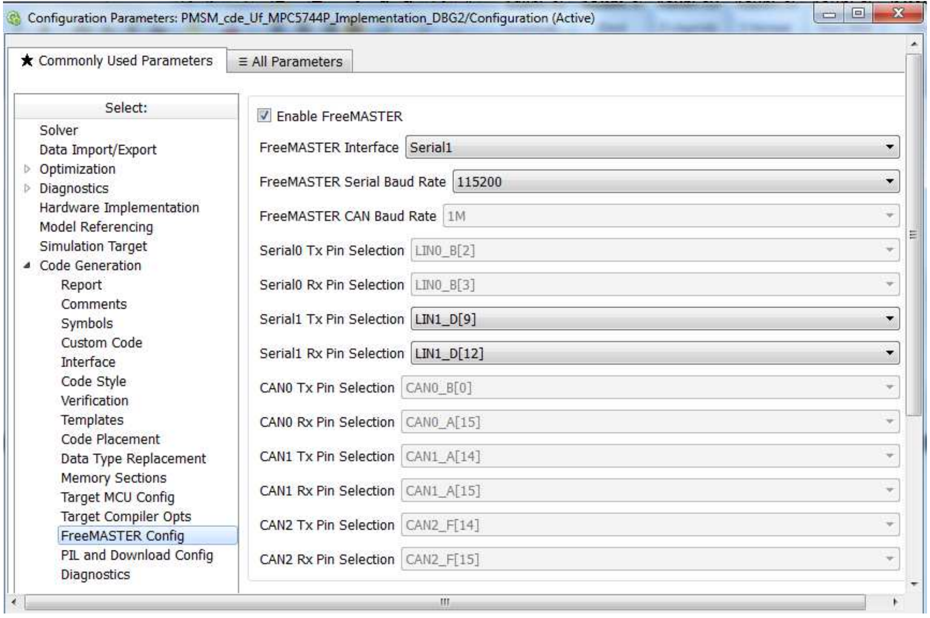

System Clock : 200 MHz
Target : MPC5744P
Package : 144LQFP
XTAL Freq : 40 MHz
Compiler : S32 Design Studio
Target Type : FLASH
Global Interrupt Priority : 8
System Tick Interrupt Priority : 12
Freemaster : None;
Download Code after build : off

MBD_MPC574xP_Config_Information

Figure 36 - MPC5744P Configuration block

For most use of MPC5744P, three panes provided in Code Generation are important:

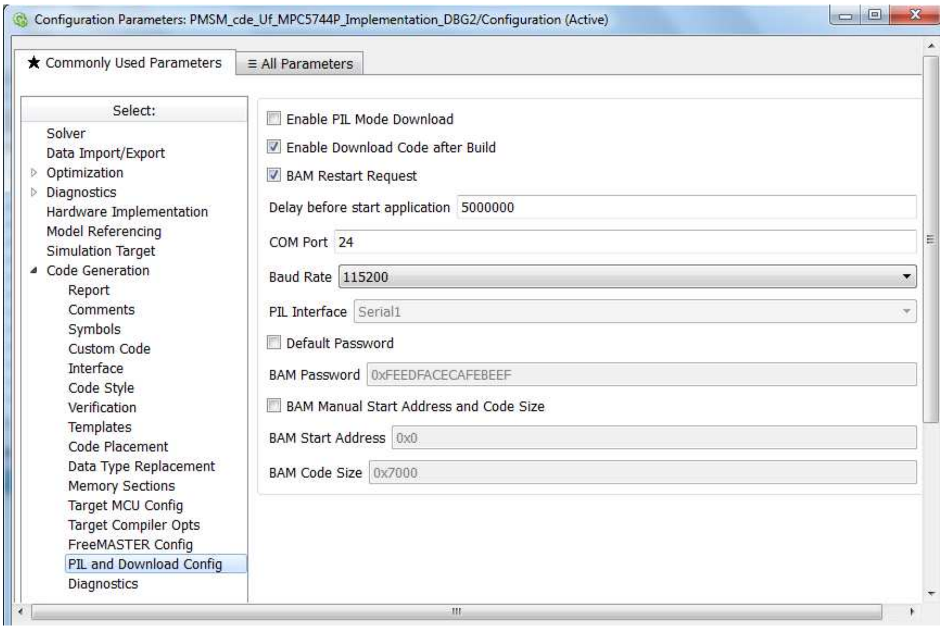
- Target MCU Config: the version of the MCU and its system clock frequency can be configured. Default values are acceptable for the needs of Automotive electronics lab.
- Freemaster Config: enable Freemaster drivers, configure the communication port, its baud rate and the used Tx/Rx pads. Figure 37 shows configuration to activate Freemaster through LINFlexD on the MPC5744P_DEVKIT. No other blocks are required to use Freemaster, except Freemaster Recorder.
- PIL and Download Config: configuration of the code downloading and Processor-In-The-Loop options. In Figure 38, the configuration required for Automotive electronics lab is shown. Select Enable Download after Build to launch the downloading of the executable file into the MCU target at the end of the code building. Select also BAM Restart Request. Set the actual COM port and the baud rate.


Configuration Parameters: PMSM_cde_Uf_MPC5744P_Implementation_DBG2/Configuration (Active)

☑ Enable FreeMASTER

FreeMASTER Interface: Serial1
FreeMASTER Serial Baud Rate: 115200
FreeMASTER CAN Baud Rate: 1M
Serial0 Tx Pin Selection: LIN0_B[2]
Serial0 Rx Pin Selection: LIN0_B[3]
Serial1 Tx Pin Selection: LIN1_D[9]
Serial1 Rx Pin Selection: LIN1_D[12]
CAN0 Tx Pin Selection: CAN0_B[0]
CAN0 Rx Pin Selection: CAN0_A[15]
CAN1 Tx Pin Selection: CAN1_A[14]
CAN1 Rx Pin Selection: CAN1_A[15]
CAN2 Tx Pin Selection: CAN2_F[14]
CAN2 Rx Pin Selection: CAN2_F[15]

Figure 37 - Configuration of Freemaster



Configuration Parameters: PMSM_cde_Uf_MPC5744P_Implementation_DBG2/Configuration (Active)

□ Enable PIL Mode Download

☑ Enable Download Code after Build

☑ BAM Restart Request

Delay before start application 5000000

COM Port 24

Baud Rate 115200

PIL Interface Serial1

□ Default Password

BAM Password 0xFEEDFACECAFEBEEF

□ BAM Manual Start Address and Code Size

BAM Start Address 0x0

BAM Code Size 0x7000

Figure 38 - Configuration of code downloading on the MCU target

b. Communication blocks

This category contains all the blocks dedicated to the configuration of communication peripherals, transmission/reception functions and associated ISR. They are provided only for CAN, SPI and LIN interface. The list of blocks are:

Name of the blocks	**Description**
LINFlexD Config	Configuration of LINFlex module (LIN unit, LIN/UART mode, TX/RX pins, baud rate…)
LINFlexD Receive Data	This block provides received data through LINFlex interface in unsigned 8 bit integer format
LINFlexD Receive Data Trigger	This block generates a call user function when LINFlexD message is coming
LINFlexD Transmit Data	This block transmits data through LINFlex interface. The data is in unsigned 8 bit integer format.
FlexCAN Config	Configuration of FlexCAN module (CAN module, protocol version, timing characteristics, clock source…)
FlexCAN Receive Data	This block provides received data through FlexCAN interface: ID of message and data in unsigned 8 bit integer format are provided
FlexCAN Receive Data Trigger	This block generates a call user function when FlexCAN message is coming
FlexCAN Transmit Data	This block transmits data through FlexCAN interface. The data and the length of the message (up to 8 bytes) must be precised.
FlexCAN ISR	This block generates a call user function when an ISR related to FlexCAN is triggered. It also defines the priority level of the ISR.
FlexCAN Interrupt Enable/Disable	This block enables or disables ISR related to FlexCAN.
DSPI Config	Configuration of DSPI module (module number, master/slave mode, frame size, timing characteristics, CS signals…)

DSPI Receive	This block provides received data through DSPI interface, in unsinged 16 bit integer format
DSPI Receive Trigger	This block generates a call user function when DSPI message is coming and provides the received data
DSPI Transmit	This block transmits data through DSPI interface, in unsigned 16 bit integer format. This block also defines the CS, SCLK, MISO and MOSI I/O pads.
DSPI ISR	This block generates a call user function when an ISR related to DSPI is triggered. It also defines the priority level of the ISR.
DSPI_ISR_Enable_Disable	This block enables or disables the different interrupt sources related to DSPI

The configuration of the different communication blocks requires a good knowledge of the characteristics of MPC5744P and the internal registers.

c. Motor control blocks

This category contains all the blocks for the configuration and activation of the peripherals dedicated to motor control : ADC, FlexPWM, CTU, eTimer, DMA and sine wave generator. Refer to reference manual of MPC5744P for more information about these peripherals and their dependencies.
The list of blocks are:

Name of the blocks	**Description**
ADC Channel Config	Configure an ADC channel (provide the ADC number, the channel number, the operating mode, and activate interrupt)
ADC Channel Input	This block provides the data converted by an ADC channel when they are updated
ADC Configuration	Configure an entire ADC block (clock selection, alignment, external trigger, CTU mode, timing characteristics…)
ADC ISR	This block generates a call user function on ADC conversion. It also defines the priority level of the ISR.
ADC Interrupt Enable	Enable/disable ADC interrupts
ADC Trigger	This block generates a function-call to trigger an ADC Conversion
ADC Watchdog Threshold	Set the lower/upper threshold of the analog watchdog of a particular ADC channel and generates a function-call when converted value exceeds the threshold
ADC Watchdog Threshold ISR	Enable/disable ADC watchdog interrupt
CTU Config	Configure a CTU block (input trigger(s), triggering mode, output trigger(s), external trigger pin, ADC triggering…)
CTU ISR	This block generates a call user function on trigger input arrival, error or reload of CTU
CTU ISR Enable	Enable/disable CTU interrupts
Complementary FlexPWM Output	Generate a complementary PWM signal on output A or B of a selected FlexPWM module
FlexPWM ISR	This block generates a call user function on compare, error or reload of FlexPWM
FlexPWM ISR Enable	Enable/disable FlexPWM interrupts
Simple FlexPWM Output	Generate a simple phase-shifted PWM signals on A and B outputs of the selected FlexPWM module

Sine Wave Generator	Generate sinusoidal voltage signal
Sine Wave Generator with Input Trigger	Generate sinusoidal voltage signal triggered by an input signal
Three-phase FlexPWM Output	Generate a three-phase complementary center-aligned PWM signals on A and B outputs of the three selected FlexPWM modules. Deadtime is configurable.
eTimer ISR	This block generates a call user function on eTimer
eTimer ISR Enable	eTimer Interrupt Enable/Disable
eTimer Capture	Read eTimer Capture FIFOs
eTimer Configuration	Configuration of eTimer block (submodule, channel, count mode, counter source …)
eTimer Pre-load	eTimer counter and comparators pre-loading
DMA Configuration	Configure the parameters of the DMA. CTU FIFOs are the only supported data sources.
DMA Channel ISR	This block generates a call user function when a major iteration count is completed
Memory Read	Read data at memory location specified by the base address and the offset value
Memory Write	Write data at memory location specified by the base address and the offset value

Tips : in the ADC_Config block, there is an error in the field WLSIDE, which sets the right/left alignment of converted data. Contrary to the comments in the block, when WLSIDE is set to 0, the conversion result is right-aligned. when WLSIDE is set to 1, the conversion result is left-aligned.

Whatever the considered peripherals, the configuration process is nearly the same. For example, let consider ADC. ADC configuration block must be placed in the diagram (obviously not in subsystem called by other functions which use ADC channels). ADC Channel configuration blocks must also be placed to set the parameter of the input channel. If interrupts will be used, associated ISR must be enabled by placing ADC_Interrupt_Enable block. Each time a conversion is done, a function can be called by ADC_ISR block. The ADC conversion result is available at the output of ADC_Channel_Input block.

d. Peripheral interface blocks

Configuration of GPIO in general purpose mode.

e. Utility blocks

This category contains some blocks for Freemaster and for Periodic Interrupt Timer (PIT).

Name of the blocks	**Description**
FreeMaster Data Recorder	Activate FreeMaster DataRecorder. Each time it is called, recorded variables are stored
Periodic Interrupt Timer	Configure PIT to trigger an interrupt periodically
PIT Interrupt Enable-Disable	Enable/disable PIT interrupts

Freemaster is configured in the Target Configuration block (port and baud rate of the UART).

3. Use in Matlab/Simulink

The functions provided by the library can be used in Simulink directly (once the library has been installed).

V - Presentation of Automotive Math and Motor Control Library for MPC574xP

This Matlab/Simulink toolbox is compatible only from R2014.a release. In this document, the version 2.2 for MPC574xP MCU is considered. More information about the mathematical functions provided by this toolbox can be found in [AMMC]. Model-Based Design Toolbox can be downloaded here: http://www.nxp.com/support/developer-resources/run-time-software/automotive-software-and-tools/model-based-design-toolbox:MC_TOOLBOX

1. Overview

The NXP's Automotive Math and Motor Control Library (AMMCLIB) provides a list of mathematical functions dedicated to motor control, which supports different number representations (fixed or floating point). This library is supported by S32DS compiler so it appears as a SDK for S32DS. Moreover, models compatible with Simulink are also available. Thus it can be used as a Matlab/Simulink's toolbox.

The AMMCLIB for NXP MPC574xP devices is organized in several sub-libraries, as depicted in Figure 39:

- Mathematical Function Library (MLIB) - it comprises basic mathematical operations such as addition, multiplication, etc.
- General Function Library (GFLIB) - it comprises basic trigonometric and general math functions such as sine, cosine, tan, hysteresis, limit, etc.
- General Digital Filters Library (GDFLIB) - it includes digital IIR and FIR filters designed to be used in a motor control application
- General Motor Control Library (GMCLIB) - it includes standard algorithms used for motor control such as Clarke/Park transformations, Space Vector Modulation, etc.
- Advanced Motor Control Function Library (AMCLIB) - it comprises advanced algorithms used for motor control purposes

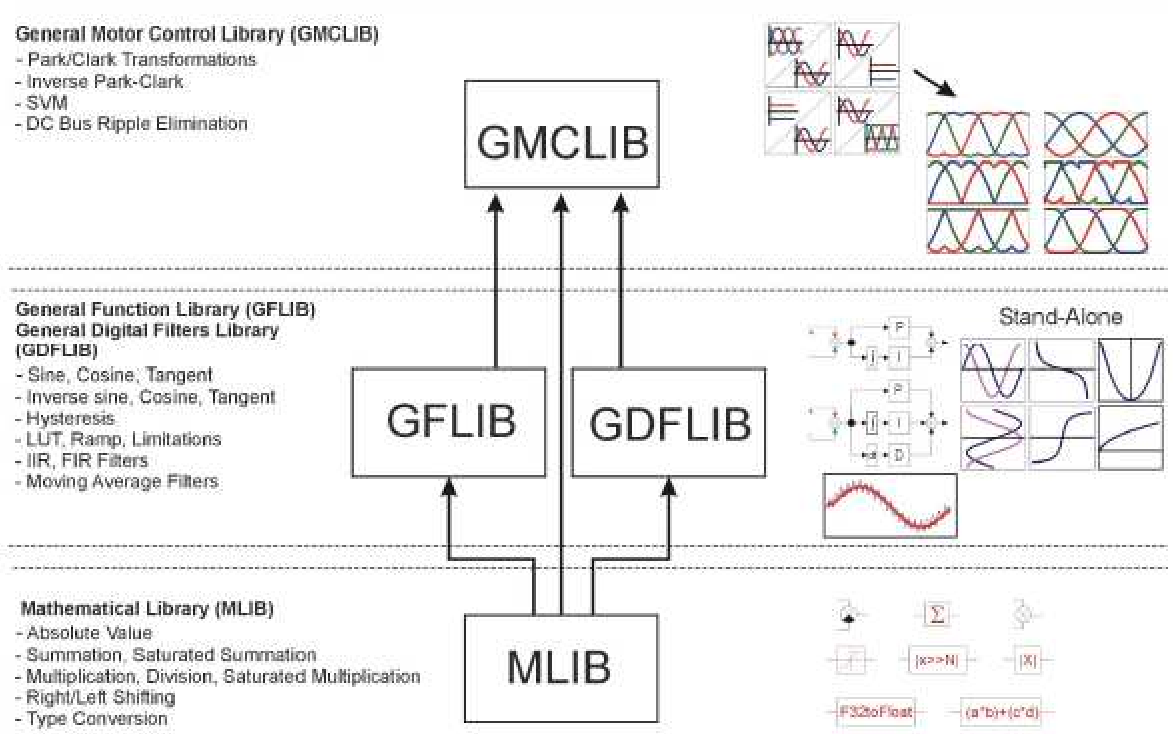

General Motor Control Library (GMCLIB)
- Park/Clark Transformations
- Inverse Park-Clark
- SVM
- DC Bus Ripple Elimination
GMCLIB
General Function Library (GFLIB)
General Digital Filters Library (GDFLIB)
- Sine, Cosine, Tangent
- Inverse sine, Cosine, Tangent
- Hysteresis
- LUT, Ramp, Limitations
- IIR, FIR Filters
- Moving Average Filters
GFLIB
GDFLIB
Stand-Alone
P
I
D
Mathematical Library (MLIB)
- Absolute Value
- Summation, Saturated Summation
- Multiplication, Division, Saturated Multiplication
- Right/Left Shifting
- Type Conversion
MLIB
Σ
|x>>N|
|X|
F32toFloat
(a*b)+(c*d)

Figure 39 - Organization of sublibraries of AMMCLIB [AMMC]

The AMMCLIB for NXP MPC574xP devices was developed to support three major implementations:

- Fixed-point 32-bit fractional (suffix F32) or Q1.31 format: in this format, the number are comprised between -1 and $1-2^{-31}$. The minimum positive value is normalized to 2^{-31}. Using the format requires a scaling of number to the interval [-1;+1] beforehand.
- Fixed-point 16-bit fractional (suffix F16) or Q1.15 format : in this format, the number are comprised between -1 and $1-2^{-15.}$ The minimum positive value is normalized to 2^{-15}. Using the format requires a scaling of number to the interval [-1;+1] beforehand.
- Single precision (32 bits) floating point (suffix FLT): in this format, the number are comprised between -2^{128} and 2^{128}, with a minimum positive value normalized to 2^{-128}. The MSB is the sign, the next 24 bits are the mantissa and the 7 last bits form the exponent.

The fixed-point 32-bit fractional and fixed-point 16-bit rational functions are implemented based on the unity model. It means that, before using blocks based on these formats, numbers must be normalized so that their values remain in the range [-1 ; +1]. Most of the functions do not integrate saturation function: any output that exceed this range induce an overflow condition.

<u>Tips: converting binary code to decimal number in Q1.31 format:</u> evaluation by the hand the number coded by a binary or hexadecimal in Q1.31 representation can be quite tedious. In Matlab, the function dec2hex() can be used to convert decimal number to hexadecimal representation, while the function hex2dec() aims at converting hexadecimal to decimal representation. Based on this two functions and on scaling transform, the conversion between decimal and Q1.31 is possible:

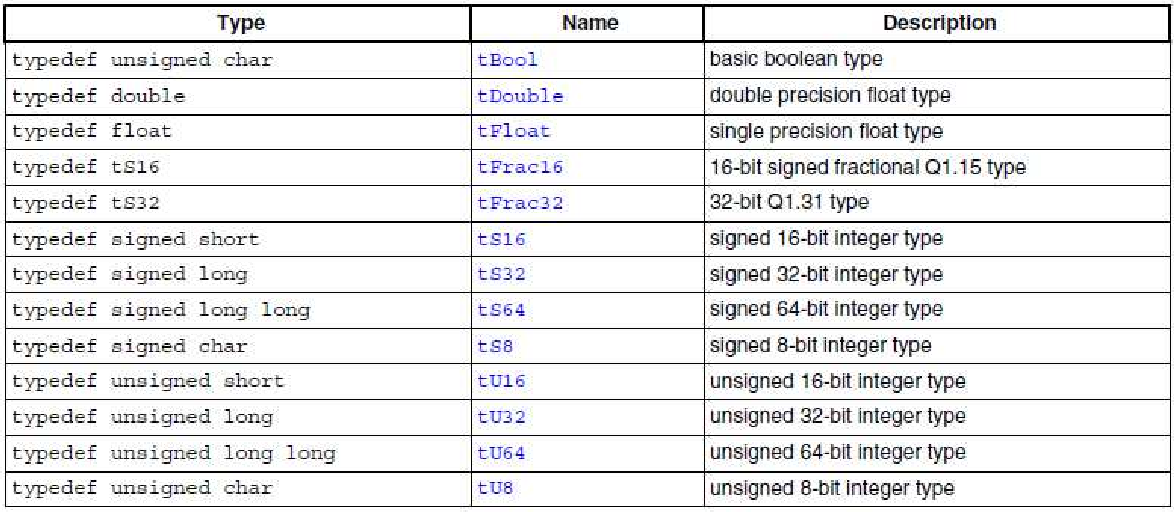
- Conversion from decimal to Q1.31 format:

- if the number is positive, use the following command: dec2hex(floor(number*2^{31})).
- if the number is negative, the sign is indicated by the MSB while the other bits code the shift from -1. Use the following command: dec2hex(floor((1-number)*2^{31})).

- Conversion from Q1.31 format to decimal:
  - if the number is positive (MSB = '0'), use the following command: hex2dec('hexa_code')/2^{31}.
  - if the number is negative (MSB = '1'), remove the MSB and use the following command: hex2dec('hexa_code')/2^{31}-1

2. Types provided by AMMCLIB

Numerous types are defined in AMMCLIB files. The basic types are summarized in the following table. Boolean, unsigned/signed integer or floating number formats are provided. Fixed-point 32-bit fractional type is called tFrac32, while tFrac16 is the type for fixed-pont 16 bit fractional number.



Type	Name	Description
typedef unsigned char	tBool	basic boolean type
typedef double	tDouble	double precision float type
typedef float	tFloat	single precision float type
typedef tS16	tFrac16	16-bit signed fractional Q1.15 type
typedef tS32	tFrac32	32-bit Q1.31 type
typedef signed short	tS16	signed 16-bit integer type
typedef signed long	tS32	signed 32-bit integer type
typedef signed long long	tS64	signed 64-bit integer type
typedef signed char	tS8	signed 8-bit integer type
typedef unsigned short	tU16	unsigned 16-bit integer type
typedef unsigned long	tU32	unsigned 32-bit integer type
typedef unsigned long long	tU64	unsigned 64-bit integer type
typedef unsigned char	tU8	unsigned 8-bit integer type

Numerous compound types also exist. They are detailed in part 7 of the AMMCLIB reference document [AMMC].

3. Brief presentation of the functions

Functions of Advanced Motor Control sublibrary are not presented in this document as they will not be used in the Automotive Electronics lab. The full list of functions is given in chapter 4 (p 151) of [AMMC].
Read carefully the MMCLIB User's guide [AMMC] to verify the performed mathematical operations, the input and output types and the required conditions. Any violation of these conditions may result in bug of Simulink simulation or code compilation (best case), or in unpredictable behavior of the embedded application (worst case).

In the following paragraphs, the list of functions in each sublibrary is given. Each function is terminated by a suffix F16, F32 or FLT to indicate the supported format. As all the functions support these three formats, the suffix is omitted in the next parts.

a. Math Function library (MLIB)

All these functions start with the prefix MLIB_. Details can be found between pages 531 and 650 of [AMMC].

Name	Description
Abs	Absolute value of input parameter
AbsSat	Absolute value of input parameter with saturation on output
Add	Addition of the two input parameters
AddSat	Addition of the two input parameters with saturation on output
Convert_FaFb	Conversion between type Fa and type Fb. The conversion functions exist for the three supported types of the library
Div	Division of the two input parameters
DivSat	Division of the two input parameters with saturation on output
Mac	Multiply - accumulate function
MacSat	Multiply - accumulate function with saturation on output
Mnac	Multiply - substract function
Msu	Multiply - substract function
Mul	Multiplication of the two input parameters
MulSat	Multiplication of the two input parameters with saturation on output
Neg	Negative value of the input parameter
NegSat	Negative value of the input parameter with saturation on output
RndSat	Round the input parameter
Round	Round the input parameter with saturation on output
ShBi	Shift to the left or right
ShBiSat	Shift to the left or right with saturation on output
ShL	Shift to the left
ShLSat	Shift to the left with saturation on output
ShR	Shift to the right
Sub	Substrate the two input parameters
SubSat	Substrate the two input parameters with saturation on output
VMac	Vector multiply accumulate function

b. General Functions library (GFLIB)

All these functions start with the prefix GFLIB_. Details can be found between pages 255 and 469 of [AMMC].

Name	Description
Acos	Arccosine function
Asin	Arcsine function
Atan	Arctangent function
AtanYX	Arctangent function applied on two input arguments
AtanYXShifted	Calculate the angle of two sinusoidal signals, one shifted in phase to the other.
ControllerPip	Parallel form of the Proportional-Integral controller, without integral anti-windup
ControllerPipAW	Parallel form of the Proportional-Integral controller, with integral anti-windup

ControllerPir	Standard recurrent form of the Proportional-Integral controller, without integral anti-windup
ControllerPirAW	Standard recurrent form of the Proportional-Integral controller, with integral anti-windup
Cos	Cosine function
Hyst	Calculation of a hysteresis function
IntegratorTR	Discrete implementation of the integrator (sum)
Limit	Test whether the input value is within the upper and lower limits
LowerLimit	Test whether the input value is above the lower limit
Lut1D	Implementation of a one-dimensional look-up table
Lut2D	Implementation of a two-dimensional look-up table
Ramp	Up/down ramp with a step increment/decrement
Sign	Sign of the input argument
Sin	Sine function
SinCos	Return Sine and Cosine functions
Sqrt	Square-root function
Tan	Tangent function
UpperLimit	Test whether the input value is below the upper limit
VectorLimit	Limit the magnitude of the input vector

c. General Digital Filters library (GDFLIB)

All these functions start with the prefix GDFLIB_. Details can be found between pages 208 and 254 of [AMMC].

Name	Description
FilterFIRInit	Initialization of FIR filter buffer
FilterFIR	Performs a single iteration of an FIR filter
FilterIIR1Init	Initialization of first order IIR filter buffer
FilterIIR1	Implements the first order IIR filter
FilterIIR2Init	Initialization of second order IIR filter buffer
FilterIIR2	Implements the second order IIR filter
FilterMAInit	Clears the internal filter accumulator
FilterMA	Implements an exponential moving average filter

d. General Motor Control library (GMCLIB)

All these functions start with the prefix GMCLIB_. Not all the functions are listed below. More details can be found between pages 474 and 526 of [AMMC].

Name	Description
ClarkInv	Compute inverse Clark transform
Clark	Compute Clark transform
ParkInv	Compute inverse Park transform
Park	Compute Park transform
SvmStd	Duty-cycle ratios using the Standard Space Vector Modulation technique

4. *Using in S32DS environment*

The first step is the import of AMMCLIB SDK during the creation of S32DS project. Refer to part I.7 of this document for SDK import.

a. Setting the implementation

By default the support of all implementations is turned off, thus the error message *"Define at least one supported implementation in SWLIBS_Config.h file."* is displayed during the compilation if no implementation is selected, preventing the user application building. Following are the macro definitions enabling or disabling the implementation support:

- SWLIBS_SUPPORT_F32 for 32-bit fixed-point implementation support selection
- SWLIBS_SUPPORT_F16 for 16-bit fixed-point implementation support selection
- SWLIBS_SUPPORT_FLT for single precision floating-point implementation support selection

These macros are defined in the SWLIBS_Config.h file located in Common directory of the AMMCLIB for NXP MPC574xP devices installation destination. To enable the support of each individual implementation the relevant macro definition has to be set to *SWLIBS_STD_ON.*

Moreover, the SWLIBS_DEFAULT_IMPLEMENTATION macro definition has to be setup properly. This macro definition is not defined by default thus the error message "Define default implementation in SWLIBS_Config.h file." is displayed during the compilation, preventing the user application building. The SWLIBS_DEFAULT_IMPLEMENTATION macro is defined in the SWLIBS_Config.h file located in Common directory of the AMMCLIB for NXP MPC574xP devices installation destination. The SWLIBS_DEFAULT_IMPLEMENTATION can be defined as the one of the following supported implementations:

- SWLIBS_DEFAULT_IMPLEMENTATION_F32 for 32-bit fixed-point implementation
- SWLIBS_DEFAULT_IMPLEMENTATION_F16 for 16-bit fixed-point implementation
- SWLIBS_DEFAULT_IMPLEMENTATION_FLT for single precision floating point implementation

b. Calling mathematical function

After proper definition of *SWLIBS_DEFAULT_IMPLEMENTATION* macro, the AMMCLIB for NXP MPC574xP devices functions can be called using standard legacy API convention: 'Sublibrary_name'_'Function_name'_'Format_suffix'. For example if the *SWLIBS_DEFAULT_IMPLEMENTATION* macro definition is set to *SWLIBS_DEFAULT_IMPLEMENTATION_F32*, the 32-bit fixed-point implementation of sine function is invoked after the *GFLIB_Sin(x)* API call. The command GFLIB_Sin_F32(x) has to be added in the C code. Moreover, the header file where the used mathematical function is declared must be included in the C code file which uses the function. For example, if the GFLIB_Sin_F32(x) is used, the directive '#include gflib.h' must be added in the C code.

5. Using in Matlab/Simulink environment

The functions provided by the library can be used in Simulink directly (once the library has been installed).

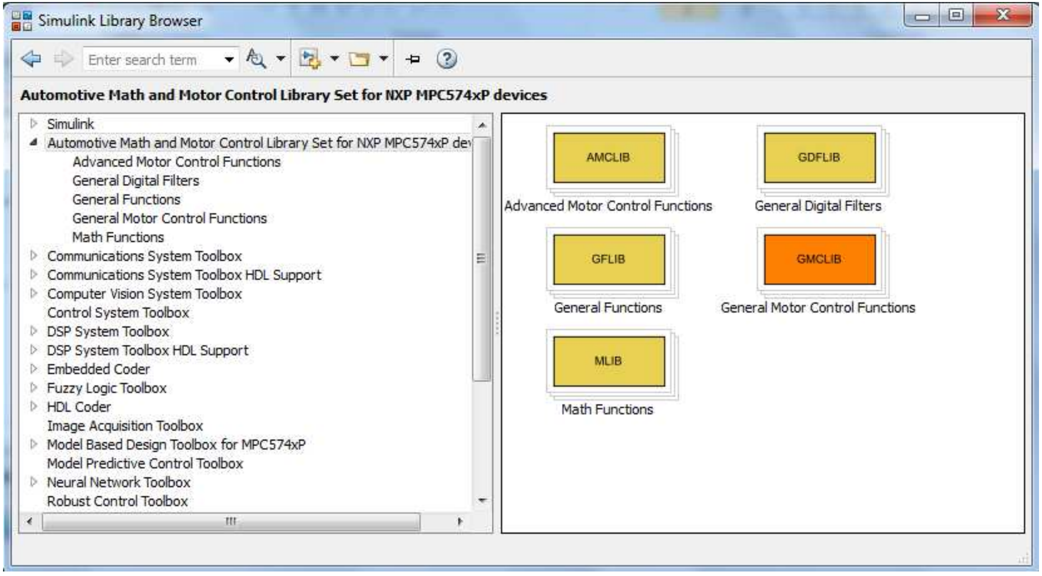

Simulink Library Browser
Enter search term
Automotive Math and Motor Control Library Set for NXP MPC574xP devices
Simulink
Automotive Math and Motor Control Library Set for NXP MPC574xP dev
Advanced Motor Control Functions
General Digital Filters
General Functions
General Motor Control Functions
Math Functions
Communications System Toolbox
Communications System Toolbox HDL Support
Computer Vision System Toolbox
Control System Toolbox
DSP System Toolbox
DSP System Toolbox HDL Support
Embedded Coder
Fuzzy Logic Toolbox
HDL Coder
Image Acquisition Toolbox
Model Based Design Toolbox for MPC574xP
Model Predictive Control Toolbox
Neural Network Toolbox
Robust Control Toolbox
AMCLIB
Advanced Motor Control Functions
GDFLIB
General Digital Filters
GFLIB
General Functions
GMCLIB
General Motor Control Functions
MLIB
Math Functions

Figure 40 - AMMCLIB library in Simulink

VI - References

[MBDT]	Model-Based Design Toolbox, User Manual, An Embedded Target for the MPC574xP Family of Processors, Version 2.0.0, 2017, www.nxp.com
[FRUG]	FreeMASTER Serial Communication Driver, User's Guide, Rev. 3.0, August 2016, NXP Semiconductors, www.nxp.com/docs/en/user-guide/FMSTRSCIDRVUG.pdf
[AMMC]	Automotive Math and Motor Control Library Set for NXP MPC574xP devices, User's Guide, Rev. 12, MPC574XPMCLUG, www.nxp.com