AutoMATiC – Tutorial

Patryk Chaber

May 17, 2019

Contents

1 Introduction

2 Requirements

This software was tested using following setup:

- Microsoft Windows 10 Pro
- MATLAB 2019a
- MinGW64 Compiler (C++)

and for the programming of the exemplary microcontroller:

- STM32F746IGT6 microcontroller (part of the Core7XXI evaluation board with Open7XXI-C board – both produced by WaveShare)
- J-Link v9.3
- Keil uVision v5.17.0.0
- STM32CubeMX

Despite aforementioned configuration, this software was tested with MATLAB 2017b, thus should work with any other MATLAB version in between of those two. This software should work also on Linux based operating systems, but it may require minor changes in paths notation.

3 Installation

To build AutoMATiC system compile its sources using `mex` compiler in MATLAB with following steps:

1. download or clone the repository of AutoMATiC:
   `https://github.com/pjchaber/automatic-mpc.git`
2. make sure that MATLAB is callable from the shell by executing:
   ```
   matlab -batch "disp(version);"
   ```
   which should print output similar to the following:
   ```
   9.6.0.1072779 (R2019a)
   ```
   If the output differs, make sure that the following MATLAB directory:
   ```
   <MATLAB root directory>\bin\win64
   ```
   is listed in `PATH` variable of your system. For example:
   ```
   E:\Program Files\MATLAB\R2019a\bin\win64
   ```
3. register MATLAB for further use of its engine executing of the following instruction in shell:
   ```
   matlab -regserver
   ```
4. change current working directory to a directory with the source of the project
5. execute in MATLAB following instruction
   ```
   mex -v -client engine main.cpp XGetOpt.cpp;
   ```
   If the compilation fails, make sure that you have set a C++ compiler by executing in MATLAB:
   ```
   mex -setup C++
   ```
   Also make sure, to use the same MATLAB version which was used in the shell at step 2 and 3.

4 Single Algorithm in a Controller

Firstly lets create STM32CubeMX project which will be used as a base for further definition of a target platform's configuration (it is expected from the reader to know how to use STM32CubeMX and how to create projects for STM32F746IGT6 or similar, and how to, based on such a project, generate a Keil uVision 5 project). Details of this project are as follows:

- Target Platform: STM32F746IGTx
- Pinout and Configuration:
  - RCC (HSE as Crystal/Ceramic Resonator)
  - ADC3 (IN0 and IN1 enabled):
    - Clock Prescaler: PLCK divided by 4
    - Scan Conversion: Enabled
    - Continuous Conversion: Enabled
    - DMA Continuous Requests: Enabled
    - Number of Regular Conversions: 2
    - Rank 1:
      - Channel: Channel 0
      - Sampling Time: 144 Cycles
    - Rank 2:
      - Channel: Channel 1
      - Sampling Time: 144 Cycles
  - DAC (OUT1 and OUT2 enabled), for both outputs:
    - Output Buffer: Enable
    - Trigger: None
  - TIM2 (Clock Source: Internal Clock):
    - Prescaler: 1000-1
    - Counter Period: 10800
  - TIM5 (Clock Source: Internal Clock):
    - Prescaler: 0
    - Counter Period: 0xffffffff
  - USART1 (Asynchronous):
    - Data Direction: Transmit Only
  - DMA2
    - ADC3:
      - Stream: DMA2 Stream 0
      - Direction: Peripheral to Memory
      - Priority: Low
      - Mode: Circular
      - Peripheral Data Width: Word
      - Memory Data Width: Word
    - ADC3:
      - Stream: DMA2 Stream 0
      - Direction: Peripheral to Memory

        - Priority: Low
        - Mode: Circular
        - Peripheral Data Width: Word
        - Memory Data Width: Word
    - NVIC:
        * Time base of System tick Timer (Preemption Priority: 5)
        * ADC1, ADC2 and ADC3 global interrupts (Enable, Preemption Priority: 3)
        * TIM2 global interrupts (Enabled, Preemption Priority: 1)
        * USART1 global interrupts (Enabled)
        * DMA2 stream0 global interrupts (Preemption Priority: 4)

- Clock Configuration:
    - Input Frequency for HSE: 8Hz
    - PLL Source Mux: HSE
    - HCLK (MHz): 216 (confirm by pressing ENTER)

- Project Manager:
    - Toolchain / IDE: MDK-ARM V5
    - Minimum Heap Size: 0x8000
    - Minimum Stack Size: 0x40000

After generating code to a Keil uVision 5 project communication interface with the PC and interface to connect process of control has to be implemented. It is worth making sure, that all the code, that is added to the project after its generation is placed between comments `/* USER CODE BEGIN <name> */` and `/* USER CODE END <name> */`, which will further on be denoted as "¡name¿" block. It will further ease the process of regeneration of the microcontroller's configuration.

4.1 Communication with PC using USART

Firstly, in the `main.c` file, `string.h` library header has to be included in the "Includes" block. This will allow to send text messages to a PC in a simple manner. Next a simple wrapper for transmitting of a string should be defined in block "0":

```
void write_string(char * txt){
  while(HAL_UART_GetState(&huart1) == HAL_UART_STATE_BUSY_TX);
  if(HAL_UART_Transmit_IT(&huart1, (uint8_t*)txt, strlen(txt))!= HAL_OK)
    Error_Handler();
  while(HAL_UART_GetState(&huart1) == HAL_UART_STATE_BUSY_TX);
}
```

At this point, it is possible to send text messages to a PC, it is worth noting, that the new line character and carriage return is not included by default in the sent string.

4.2 Measuring and Creating Voltage Signal

Voltage signals after conversion to a digital values will be averaged over 100 consecutive samples. Therefore a table for those raw measurements have to be defined in block "PV" as:

```
uint32_t adc_val_raw[ADC_SIZE*2] = {0};
```

where `ADC_SIZE` is defined in block "PD" as:

```
#define ADC_SIZE 100
```

Due to having two signals, the size of table with raw measurements is `ADC_SIZE*2`. To increase the resolution of averaged measurements, those will be stored as `float` variables (block "PV"):

```
float adc_val_f[2] = {0,0};
```

Having those defined, it is important to start a ADC unit. It can be implemented by inserting following code into a block "2":

```
HAL_ADC_Start(&hadc3);
if(HAL_ADC_Start_DMA(&hadc3, (uint32_t*)adc_val_raw, ADC_SIZE*2) != HAL_OK)
    Error_Handler();
```

Thanks to a DMA mechanism, after obtaining 100 consecutive samples of each signal a specific callback function will be executed. It can be redefined to perform measurements' averaging as soon as they are available, by inserting following code in block "0":

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle)
{
  static int i=0;
  static uint32_t tmpval[2] = {0,0};
  for(i=0,tmpval[0]=0,tmpval[1]=0;i<ADC_SIZE; ++i){
    tmpval[0] += adc_val_raw[2*i];
    tmpval[1] += adc_val_raw[2*i+1];
  }
  adc_val_f[0] = ((float)tmpval[0]/ADC_SIZE-2047.5f)/2047.5f;
  adc_val_f[1] = ((float)tmpval[1]/ADC_SIZE-2047.5f)/2047.5f;
}
```

This will allow to have constant supply of averaged results from ADC without consuming much of a computational power of a microcontroller core (thanks to DMA).

4.3 Dynamic Matrix Control – Analytic version

4.4 Dynamic Matrix Control – Numeric version

4.5 Generalized Predictive Control – Analytic version

4.6 Generalized Predictive Control – Numeric version

5 Multiple Algorithms in a Controller

5.1 Analytic and Numeric versions of DMC

6 Extending of the Library

6.1 Proportional-Integral-Derivative Controller

A Test

Two