PROGRAMMING THE A17802 OR A17803 WITH MATLAB AND A17802-3 PROGRAMMING BOARD

By Loïc Chretien
Allegro MicroSystems

INTRODUCTION

This application note describes how to use the A17802 or A17803 (A17802/3) MATLAB scripts provided on the Allegro software portal together with the A17802-3 programming board (PB) to configure and read out the A17802/3 device.

For Python programming, additional documentation is available in the Allegro software portal [1] (registration is required).

For a graphical user interface (GUI), a user manual is available in the software portal.

HARDWARE AND CONFIGURATION

Use of the A17802/3 with MATLAB via the Allegro Advanced Sensor Programming (AASP) DLL requires the following equipment:

- A17802-3 programming board (PB) plus STM Nucleo-L432KC microcontroller board (TED-0003955).
- Inductive sensor board with the A17802/3 device (such as TED-0004559 daughter board).
- PC with USB connection to PB.

To enable programming access, the PB must be connected to the sensor board using the provided ribbon cable, or according to the following scheme:

- For SPI programming for the A17802 or A17803-S:
  - Connect PB Pin 1 to the sensor board pin corresponding to A17802/3 Pin 1 (SINN – MISO).
  - Connect PB Pin 2 to the sensor board pin corresponding to A17802/3 Pin 2 (SINP – MOSI).
  - Connect PB Pin 3 to the sensor board pin corresponding to A17802/3 Pin 3 (COSP – SCLK).
  - Connect PB Pin 4 to the sensor board pin corresponding to A17802/3 Pin 4 (COSN – CSN).
  - Connect PB VCC Pin 6 to the sensor board pin corresponding to A17802/3 Pin 6 (VCC).
  - Connect PB GND Pin 5 to the sensor board pin corresponding to A17802/3 Pin 5 (GND).
- For Manchester programming for the A17802 or A17803-M:
  - Connect PB Pin 1 to the sensor board pin corresponding to A17802/3 Pin 1 (SINP/MHT – MHT / SENT/ PWM).
  - Connect PB VCC Pin 6 to the sensor board pin corresponding to A17802/3 Pin 6 (VCC).
  - Connect PB GND Pin 5 to the sensor board pin corresponding to A17802/3 Pin 5 (GND).

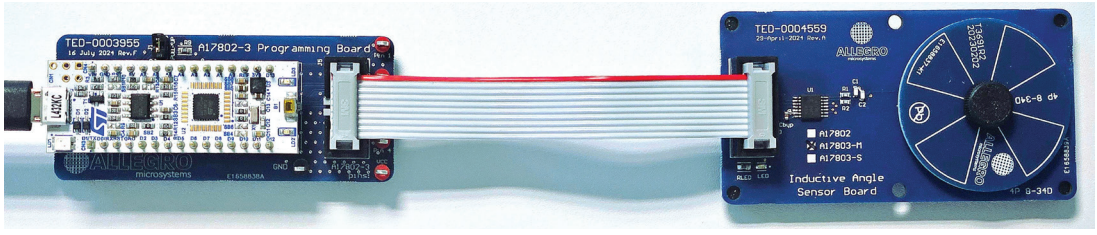


Figure 1: A17802 Evaluation Kit

[1] https://registration.allegromicro.com/

SOFTWARE

Preparation

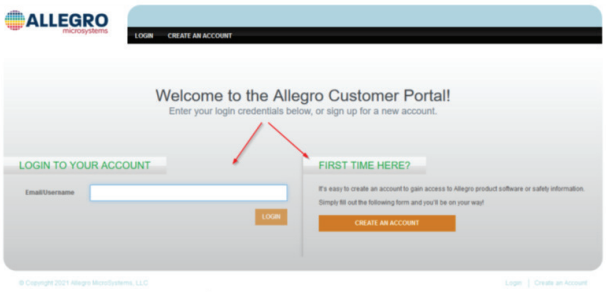
MATLAB requirements:

- MATLAB R2022a or higher

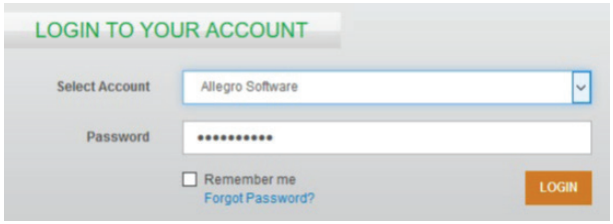
Allegro Software Requirements

Acquire access to the Allegro software:

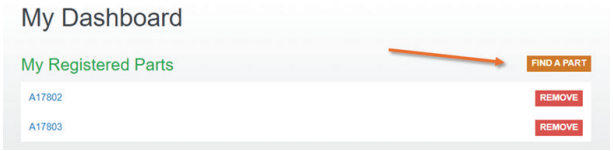
1. Log in or create an account on the Allegro software portal. [1]



2. Choose the "Allegro Software" option when logging in.



3. Click on "Find A Part", then search for the "A17802" or "A17803" software packages.



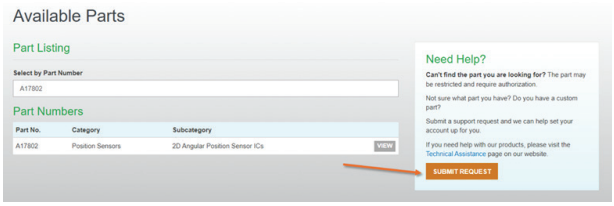
PB Firmware

The default firmware on the PB is not guaranteed to be the correct software version 1.3.4 or greater. Therefore, the PB firmware version must be checked. Use the PB GUI to check and update the PB firmware version as described in the A17802 Evaluation Kit User Guide, available for download from the Allegro software portal. [1]

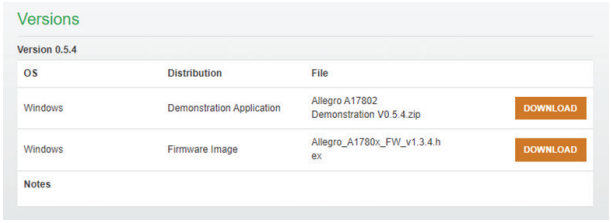
A17802 DLL Software

The required A17802 DLL files are also available on the software portal.

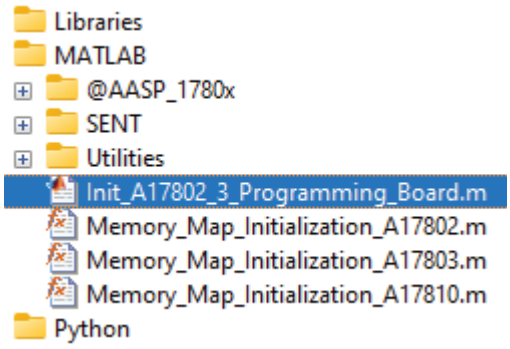
1. If the user is not registered for the A17802 software package, a request must be made. The request is checked by an Allegro employee, then approved to ensure access to the DLL.



2. Download the A17802-CUSTOMERSCRIPTS folder. The GUI programming application can also be downloaded here.



3. Unzip the files from the downloaded .zip file (code is provided in the MATLAB Programming section). MATLAB scripts are in the MATLAB folder. Open the INIT_A17802_3_PROGRAMMING_BOARD.M file.



MATLAB PROGRAMMING

First, the hardware needs to be set up (see the Hardware and Configuration section). At this point, at least two LEDs must be red on the programming board. Then, the user can initialize the setup.

⚠ WARNING: The red LED next to the USB connector might turn green unexpectedly: This means the UART connection between the PC and the PB is lost. In this case, the PB needs to be disconnected and reconnected (unplug and replug the USB cable) and the PB must be reinitialized (see Initialization Script section).

Initialization Script

The INIT_A17802_3_PROGRAMMING_BOARD.M file is made to automatically set up MATLAB to establish a connection with the PB and enable communication with the A17802/3 on the wired sensor board.

IMPORTANT: Because this script handles the initial MATLAB setup, this script must always be run first.

NOTE: The first command line of the script ensures that all downloaded folders are added to the MATLAB search path (as used DLL files are located in the Libraries folder); the required folders can also be added to the MATLAB search path manually.

The FTE object contains all A17802/3 methods, memory map, and serial communication data. To enable initiation of serial communication with the PB, any existing FTE variable must be cleared; this has the effect of closing any previous serial communication opened before using this variable name.

```
% Closes COM port connection if already open
if exist('fte','var')
    clear fte
end
```

At this moment, the FTE object can be created calling the class constructor. The desired communication port (COM port) can be optionally passed as an argument. Once the USB cable is plugged into the computer, the COM port can be found in the Window's Device Manager (Ports section). This function is detailed in the AASP Script and DLL Loading section.

```
% Creates FTE object from class constructor.
fte = AASP_1780x;
```

The A17802/3 memory map can be initialized, using either:

```
fte = Memory_Map_Initialization_A17802(fte);
```

or:

```
fte = Memory_Map_Initialization_A17803(fte);
```

Finally, the script finds the A17802/3 communication protocol and output type, then powers it on and accesses the A17802/3.

```
% Automatically detect configuration, power-on and send access codes:
getComProtocol(fte);
```

At this point, the green LED on the A17802/3 sensor board should be turned on. This means the A17802/3 is powered on. It should now be possible to send commands to the IC. To verify proper functionality, perform the following read operation:

```
angle = fte.ReadPartial(fte.direct.angle)
```

AASP Script and DLL Loading

The PB command set is a subset of the Allegro Advanced Sensor Programming (AASP) command set. Allegro provides a DLL that manages serial communication using the command set. The DLL methods provided enable all actions necessary to program the IC.

The script that follows provides all steps needed to load the DLL and start communication. Located in the AASP_1780X.M file, the AASP DLL constructor is called via the AASP_1780x(com) function. This function is called in the initialization script.

When called, it starts by loading the AASP_A1780X.DLL DLL library.

```
NET.addAssembly(directory);                % load DLL library
```

Then, it creates an object called AASP, which handles AASP operations.

```
aasp = Allegro.ASEK.AASP;                  % create AASP object
```

A log file is created to keep track of each command and communication event. This file is hosted in the current folder opened in MATLAB.

```
aasp.StartCommunicationLogging("log.txt");  % create a LOG file
```

The serial communication is opened with the PB using the COM port, passed as the parameter in the COMPORT variable.

```
aasp.OpenCommunicationPort(comport);        % opens the serial communication
```

Finally, the PORT, DEVICE, and DEVICE_ID objects are created for appropriate methods handling.

```
port0 = Allegro.ASEK.AASP_Port(aasp, 0);    % creates PORT object
device = Allegro.ASEK.AASP_A1780x(port0);  % creates DEVICE object
deviceID = device.GetDeviceID;             % gets DeviceID for device reference within
AASP function calls
aasp.CreateDevice(deviceID);
```

Accessing Communication with IC

To communicate with the A17802/3 device, the communication protocol, output type, and access code settings need to be configured in the PB. Configuration of these settings is automated in the initialization script via the "getComProtocol(fte)" function. This function automatically detects the IC configuration and power-on settings, then sets V_{CC} to 5 V and sends the access codes.

Communication Protocol—SPI

To configure the PB for SPI communication (for A17802 or A17803-S), use the following script lines:

```
% sets the device communication protocol
fte.device.SetParameter("DeviceProtocol", 'SPI');
% sets the device output type
fte.device.SetParameter("DeviceOutputType", 'Analog');
% configures the output MUX for SPI communication
fte.device.SetOutputMUX(Allegro.ASEK.tMuxOutput.SPI);
% SPI clock frequency can go from 100000 Hz (0.1 MHz) to 10000000 Hz (10 MHz)
fte.device.SPIClockFrequency=100000;
```

Alternatively, the fte.setSPI wrap-up function can be used to execute this full set of commands.

Communication Protocol—Manchester

For Manchester communication (for A17802 or A17803-M), use the following script lines:

```
% sets the device communication protocol
fte.device.SetParameter("DeviceProtocol", "Manchester");
% configures the output MUX for Manchester communication
fte.device.SetOutputMUX(Allegro.ASEK.tMuxOutput.Manchester);
% Manchester frequency can go from 2000 Hz (2 kHz) to 40000 Hz (40 kHz)
fte.device.SetParameter("ManchesterBitRate", 2000);
```

Alternatively, the fte.setMan('SENT') wrap-up function can be used to execute this full set of commands, where 'SENT' can be replaced by the A17803-M output type.

Access to communication with the A17803-M device requires a specific configuration of timings for auxiliary (aux) interrupt pulses that are dependent on the output protocol configuration of the IC.

The different settings are provided in the table that follows according to the output protocol of the IC. The table indicates how parameters should be set in order to access communication. For more information, refer to the A17803 datasheet.

Parameter	Output on A17803-M Pin 1					
	PWM	SENT Free Running	TSENT	ASENT	SSENT Short	SSENT Long
Device Output Type	PWM	SENT	SENT	SENT	SENT	SENT
Manchester Command Enable Type	PWM Function Pulse	SENT Function Pulse	Trigger SENT	AUX Function Pulse	AUX Function Pulse	AUX Function Pulse
Manchester Aux Pulse Width	–	50 × TICK_TIME	40 × TICK_TIME	55 × TICK_TIME	55 × TICK_TIME	250 × TICK_TIME
Manchester Trigger Width	4/(PWM_FREQ)	–	–	–	–	–
Manchester High Delay Width	0	0	0	0	0	0
Manchester Low Delay Width	0	0	0	0	0	0

```
SENT_tick_time = 1.0e-6; % in seconds (1 us by default)
PWM_freq = 125; % in Hz (125 Hz by default)

switch(output_type)
    case ‘ANALOG’
        fte.device.SetParameter(“DeviceOutputType”, “Analog”);
        fte.device.SetParameter(“ManchesterCommandEnableType”, “Overdrive”);
        fte.device.SetFloatParameter(“ManchesterTriggerWidth”, 0.024);
     case ‘PWM’
        fte.device.SetParameter(“DeviceOutputType”, 2);
        fte.device.SetParameter(“ManchesterCommandEnableType”, “PWMFunctionPulse”);
        fte.device.SetFloatParameter(“ManchesterTriggerWidth”, 4 / PWM_freq); % use 4 * (1
/ PWM Carrier Frequency)

    case ‘SENT’
        fte.device.SetParameter(“DeviceOutputType”, 2);
        fte.device.SetParameter(“ManchesterCommandEnableType”, “SENTFunctionPulse”);
        fte.device.SetFloatParameter(“ManchesterAuxPulseWidth”, 50 * SENT_tick_time); %
use 50 * SENT Tick Time

    case ‘TSENT’
        fte.device.SetParameter(“DeviceOutputType”, 2);
        fte.device.SetParameter(“ManchesterCommandEnableType”, “TriggerSENT”);
        fte.device.SetFloatParameter(“ManchesterTriggerWidth”, 40 * SENT_tick_time); % use
40 * SENT Tick Time except if it falls below the minimum trigger time defined by trig-
ger_cfg
        fte.device.SetFloatParameter(“ManchesterDelayWidth”, 40 * SENT_tick_time); % use
40 * SENT Tick Time
        fte.device.SetFloatParameter(“ManchesterAuxPulseWidth”, 40 * SENT_tick_time); %
use 40 * SENT Tick Time

    case ‘ASENT’
        fte.device.SetParameter(“DeviceOutputType”, 2);
        fte.device.SetParameter(“ManchesterCommandEnableType”, “AUXFunctionPulse”);
        fte.device.SetFloatParameter(“ManchesterAuxPulseWidth”, 55 * SENT_tick_time); %
use 55 * SENT Tick Time
```

```
    case 'SSENT SHORT'
        fte.device.SetParameter("DeviceOutputType", 2);
        fte.device.SetParameter("ManchesterCommandEnableType", "AUXFunctionPulse");
        fte.device.SetFloatParameter("ManchesterAuxPulseWidth", 55 * SENT_tick_time); %
use 55 * SENT Tick Time

    case 'SSENT LONG'
        fte.device.SetParameter("DeviceOutputType", 2);
        fte.device.SetParameter("ManchesterCommandEnableType", "AUXFunctionPulse");
        fte.device.SetFloatParameter("ManchesterAuxPulseWidth", 250 * SENT_tick_time); %
use 250 * SENT Tick Time
end
```

Access Codes

Access codes need to be written to a specific memory address in order to access indirect memory for the SPI interface and both direct and indirect memory for the Manchester interface.

To communicate access codes and respective register addresses to the PB, use the following lines:

```
% sets the unlock code addresses
fte.device.SetArrayParameter("DeviceUnlockAddresses", [30 30]);
% sets the unlock code
fte.device.SetArrayParameter("DeviceUnlockCodes", [50200 3713]);
```

Power-On and Access

Accessing programming for the A17802/3 requires sending access codes after a power-on. Use the SetVccOn command to power-on the IC and write access codes to the specified address within the access time:

```
fte.port0.SetVccOn(5.0,true); % supplies 5V and unlocks the device with access codes.
% First argument is Vcc voltage supplied to the IC (in Volts).
% Second argument, if true, sends the unlock codes after power-on.
```

Alternatively, the fte.unlock wrap-up function can be used to power-cycle the device and to send access codes.

Reading and Writing Methods

Read and write methods within the device object can be used to read and write whole registers in the memory map.

```
angle = fte.device.Read(0x9); % read angle output at register 9
fte.device.Write(0x0D,1); % writes 1 in register 1D triggering a soft reset
```

Indirect memory locations can be directly addressed with a single command line by shifting the indirect memory address by 0x1000000. The multiple read and write operations to realize a single indirect memory read and write are managed by the PB and are transparent to the user.

```
fte.device.Read(0x1000012) % reads indirect memory register 0x12
```

The AASP command set offers the possibility of reading and writing to the memory field instead of whole registers. The write-field and read-field methods are used for this purpose as in the example that follows.

```
% Write bits 16 to 1 in indirect memory register 0x1C “del_zero_angle” to 32768.
% This shifts angle output of 180°
fte.device.WriteField(0x1000000+0x0000001C, 16,1, 32768)
Data = fte.device.ReadField(0x1000000+0x0000001C,16,1); % read “del_zero_angle” field
del_zero_angle = Data(1) % field value is contained in first cell
```

Alternatively, the following wrap-up functions can be used to read and write fields by name:

```
% These functions use memory map fields defined in fte object
% The argument syntax is ‘fte.memory_type.field_name’.
fte.WritePartial(fte.eeprom.del_zero_angle, 32768);
del_zero_angle = fte.ReadPartial(fte.eeprom.del_zero_angle)
```

PERFORMING END-OF-LINE CALIBRATION

A17802/3 calibration for evaluation purposes can be performed by using the PB and MATLAB. Details about how to perform end-of-line calibration are provided in the Allegro application note “Configuring A17802 AND A17803 (AN296299).”

The A1780X_SIMPLE_CALIBRATION.M and A1780X_ADVANCED_CALIBRATION_QUASI_STATIC.M MATLAB scripts provide examples. A placeholder is included for the final user to add commands to move the angle of the target by using a rotary stage.

The Allegro_A17802_trim_x_y function implements all equations needed to calculate compensations and returns a dictionary containing the trimming codes. Codes can then be written in the EEPROM memory, following the calibration script.

CONCLUSION

This application note demonstrates how to set up and use MATLAB scripts to streamline A17802/3 communication and configuration and introduces the end-of-line calibration process. By leveraging the automation and analysis capabilities of these scripts, a higher level of calibration accuracy and repeatability can be achieved, leading to enhanced A17802/3 performance and reliability.

The examples and techniques presented in this guide provide a practical framework for engineers to develop and implement efficient calibration routines. Wider adoption of these methods is encouraged to promote further optimization of the calibration procedures and a commitment to delivering high-quality products.

Because contingencies may arise, these considerations are recommended in the development phase. For any questions or assistance with implementation, contact Allegro support resources. [2]

[2] https://www.allegromicro.com/en/about-allegro/contact-us

Revision History

Number	Date	Description	Responsibility
–	October 1, 2025	Initial release	Loïc Chretien

955 PERIMETER ROAD • MANCHESTER, NH 03103 • USA
+1-603-626-2300 • FAX: +1-603-641-5336 • ALLEGROMICRO.COM