UM2116

User manual

STEVAL-MKI109V3 Professional MEMS Tool motherboard for MEMS adapter boards

Introduction

The STEVAL-MKI109V3 motherboard provides users with a complete, ready-to-use platform for the evaluation of STMicroelectronics MEMS products.

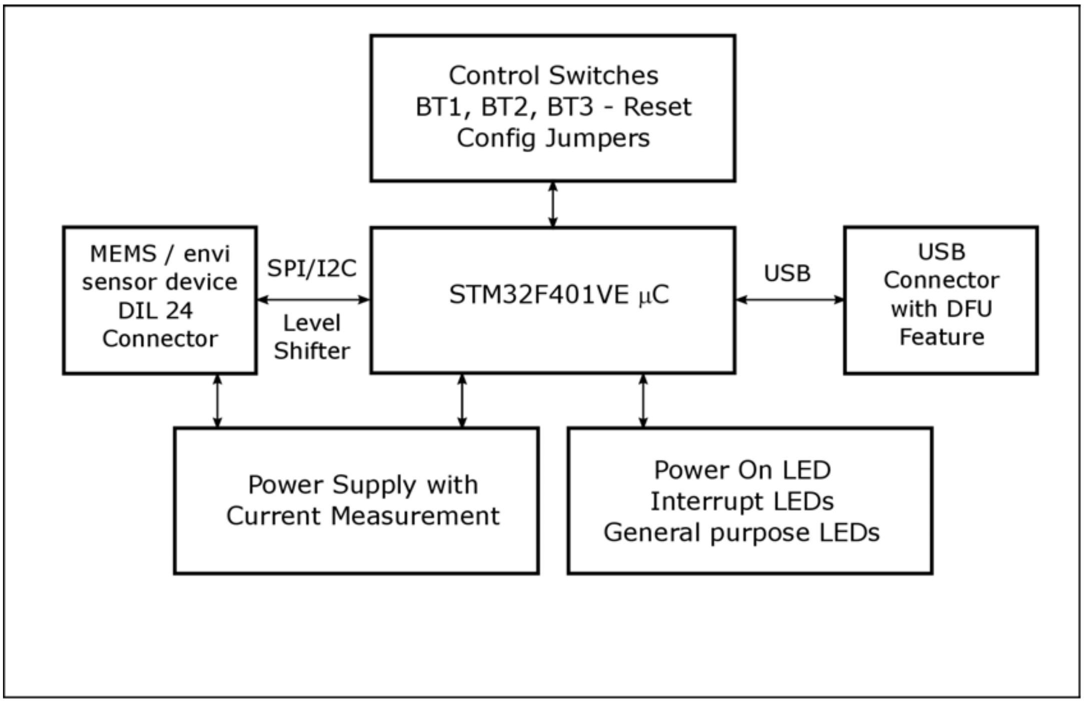
It includes a high-performance 32-bit microcontroller which functions as a bridge between the sensors and a PC, on which you can download and run the graphical user interface (GUI) or dedicated software routines for customized applications.

The board features a DIL24 socket to mount all available adapters for both digital and analog output MEMS devices.

1 Demonstration kit description

The Professional MEMS Tool is a complete demonstration kit for digital and analog MEMS sensors. Thanks to its DIL24 connector, a wide range of MEMS adapter boards can be used.

Figure 1. Demonstration board block diagram



The Professional MEMS Tool demonstration kit is based on the STM32F401VE microcontroller and can be connected to a PC via USB. Data from MEMS sensors connected to the board can be read through the PC GUI provided with the kit.

The Professional MEMS Tool also implements the DFU (device firmware upgrade) feature, so it can be reprogrammed with a new firmware release without the need to use a programmer (see www.st.com/mems).

The Professional MEMS Tool integrates:

- Six LEDs:
  - two LEDs connected via FET buffers to the interrupt pins of digital adapters
  - a power/USB LED
  - three general-purpose LEDs for firmware state indication
- Three buttons:
  - two user buttons on a dedicated GPIO of the microcontroller
  - a microcontroller reset button

All the MEMS adapter pins are available on board connectors J1 and J3.

Figure 2. Top silkscreen of the Professional MEMS Tool kit

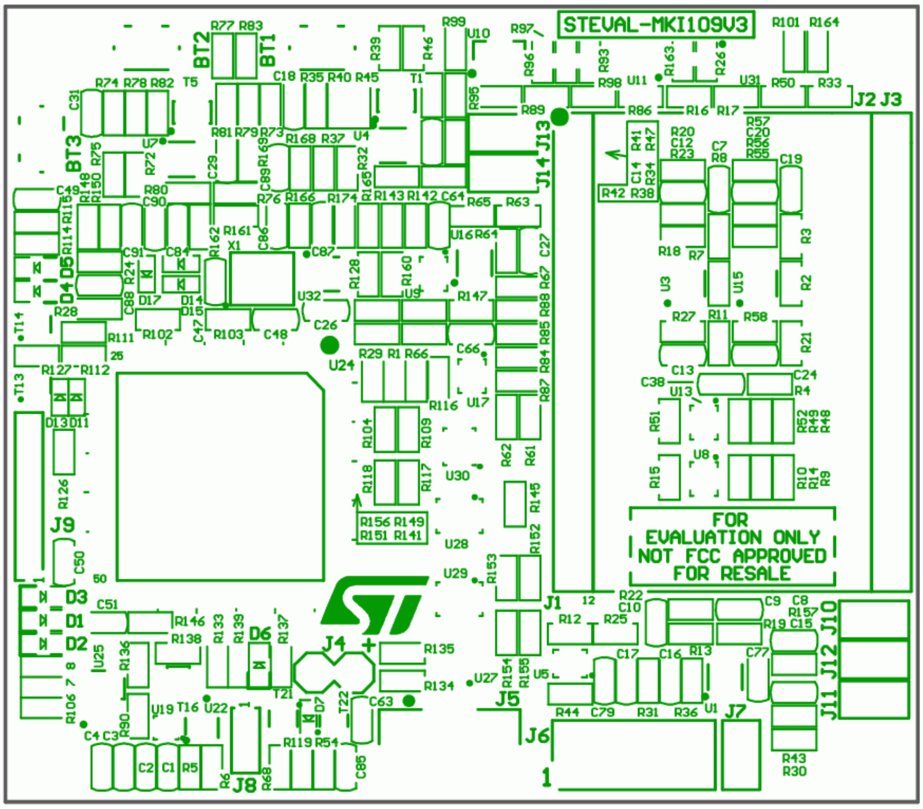


Figure 3. Top view of Professional MEMS Tool kit

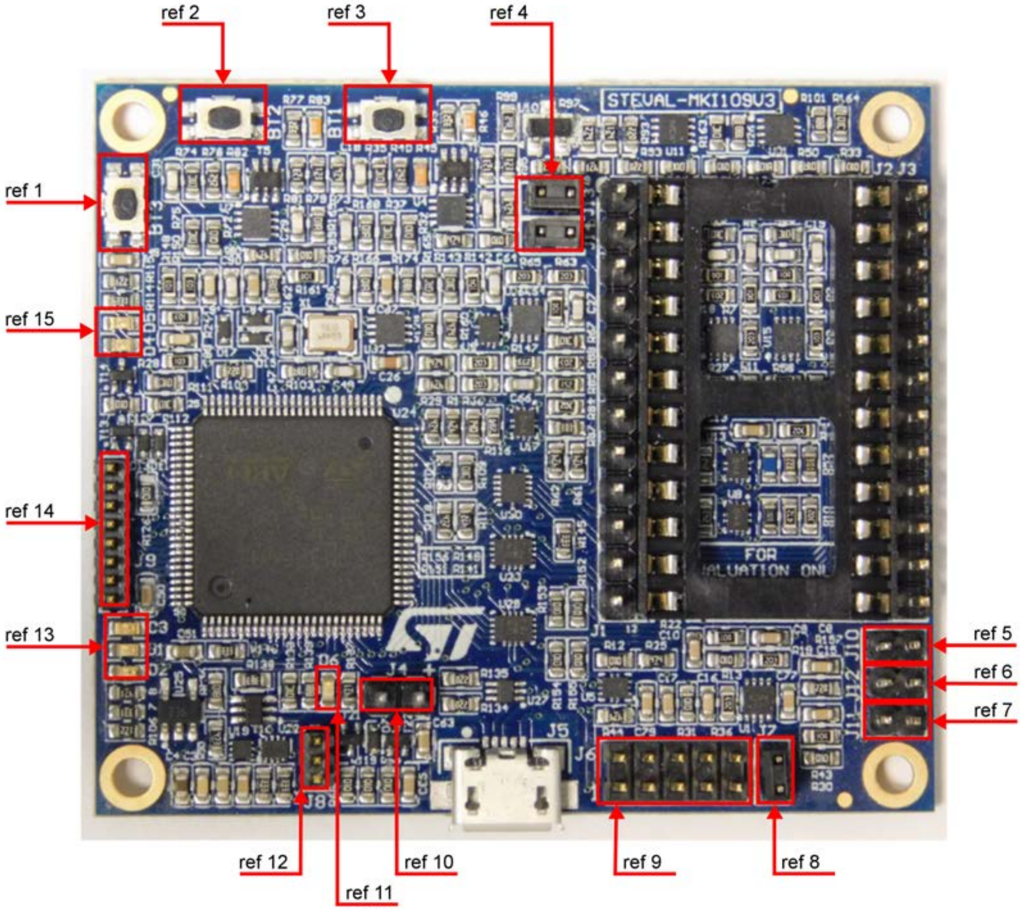


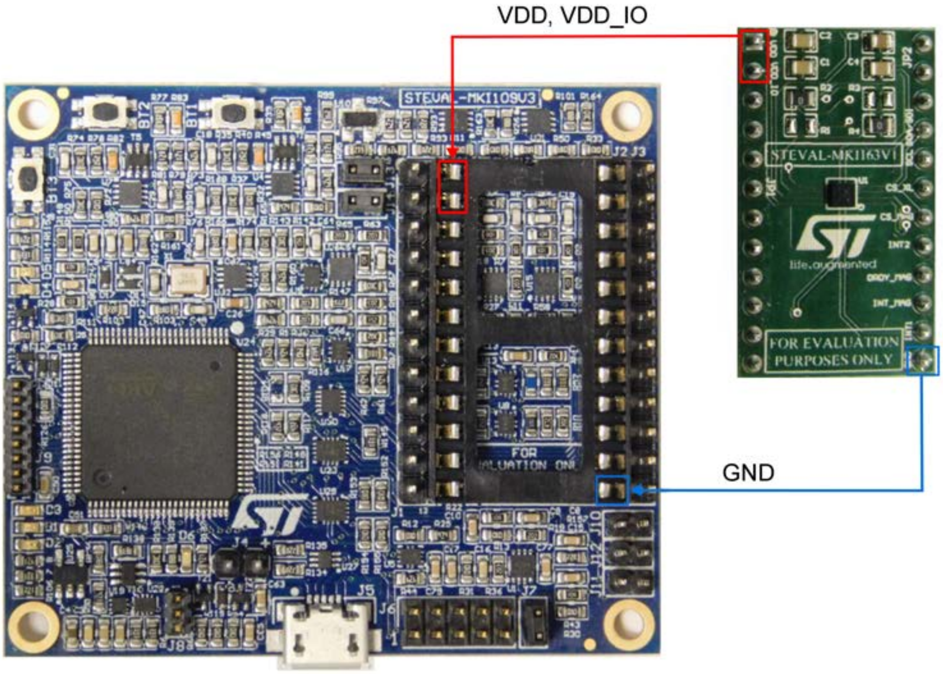
Figure 3 highlights some of the main components on the top layer of the Professional MEMS Tool kit.

1. Button BT3 is used to reset the STM32.
2. Button BT2 connected to STM32 GPIOs and available to the user. To enter DFU mode:
   a. press buttons BT3 (Reset) and BT2 together
   b. first release BT3 and then release BT2
3. BT1 connected to STM32 GPIOs and available to the user.
4. Jumpers J13 (VDD) and J14 (VDDIO) allow the user to measure the sensor current consumption by connecting a multimeter in series with their terminals.
5. Jumper J10 is used as a general purpose input to manually set certain features for several MEMS adapters.
6. Jumper J12 is used as a general purpose input to manually set certain features for several MEMS adapters.
7. Jumper J11 is used to set the self-test feature during testing of Professional MEMS Tool PCB.
8. Jumper J7 is used to select either JTAG (JP7 open – NRST control not allowed from programming connector J6) or SWD mode (JP7 shorted – NRST control allowed from connector J6).
9. J6 connector can be used to reprogram the STM32 and debug the code through the JTAG or SWD protocols.
10. Jumper J4 can be used to directly supply the board (from 4.5 V to 5.5 V) instead of through the USB connector.
11. LED D6 lights up when the board is powered.
12. J8 connector can be used for UART RX/TX communication.
13. LEDs D1, D2, and D3 are general-purpose LEDs used to indicate firmware states; e.g.:

   a. LED D3 YELOW light up when specific firmware is selected from those available
   b. LED D2 RED on indicates that the microcontroller is properly configured for communication with the sensor
   c. LED D1 GREEN blinks according to the sensor data rate selected
14. J9 connector can be used for general purpose SPI bus.
15. LEDs D4 and D5 are directly connected to the interrupt pins of the MEMS digital adapters (if available on the sensor mounted on the adapter board).

Figure 4. How to plug the DIL24 adapter on STEVAL-MKI109V3 shows how to plug the DIL 24 adapter MEMS module on the Professional MEMS Tool. VDD and VDDIO are in the top left corner (pins 1 and 2) and GND is in the bottom right corner (pin 13).

Figure 4. **How to plug the DIL24 adapter on STEVAL-MKI109V3**



2 Professional MEMS Tool board installation

The software packages can be downloaded from the st.com website; it is arranged in the following directory structure:

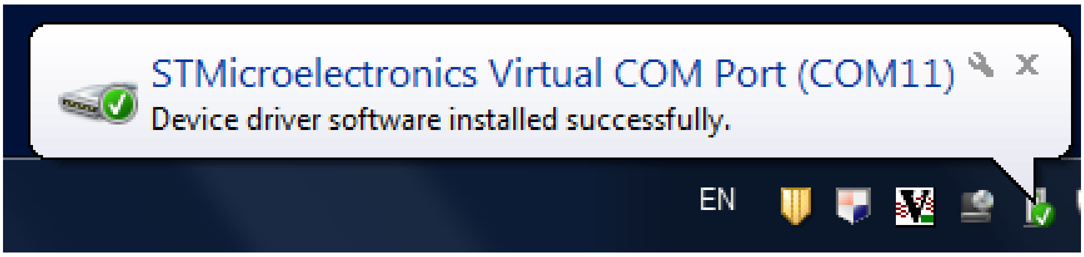
- **DRIVER**: it contains the installation package for the USB drivers needed to connect the Professional MEMS Tool board to the PC. No driver is needed on Linux and Mac OS platforms, so this directory is included in the Windows installation package only.
- **DFU**: it contains the .dfu files and the installation package for the software needed to upgrade the firmware of the Professional MEMS Tool board.
- **FIRMWARE**: it contains the source code of the firmware of the Professional MEMS Tool board together with the corresponding binary file that can be flashed to the board using the DFU software.

2.1 Hardware installation (Windows® platforms)

- For Linux® and Mac OS® platforms, no driver installation is required.
- For Windows platforms, install the STM32 virtual COM port driver by running VCP_V1.4.0_Setup.exe in the DRIVER folder of the Windows installation package and follow the instructions.

Once the driver is installed, connect the demonstration kit board to a free USB port. A confirmation message should appear.

Figure 5. **Successful device driver installation**

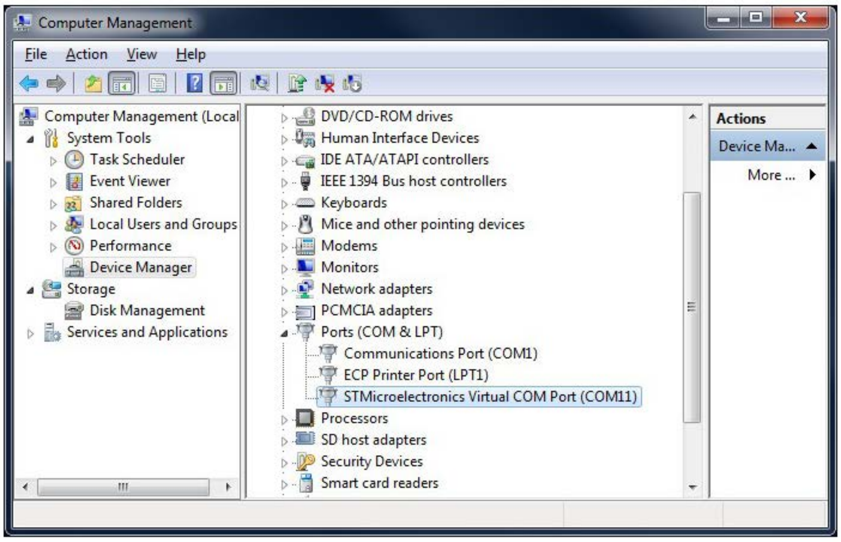


STMicroelectronics Virtual COM Port (COM11)
Device driver software installed successfully.

Confirm which COM port has been assigned to the board: right click on My Computer and select Manage, then select Device Manager and scroll through the list to Ports (COM & LPT).

Note: The STM32 virtual COM port driver for Windows platforms and related documents are packaged with the STSW-STM32102 software download at www.st.com

Figure 6. **Virtual COM port assignment**



2.2 DFU

The MEMS STEVAL-MKI109V3 demonstration board can reprogram an application via USB, in accordance with the DFU class specification defined by the USB Implementers Forum. This direct reprogramming of the microcontroller is particularly suited to USB applications where the same USB connector can be used both for the standard operating mode and the reprogramming process.

To configure the Professional MEMS Tool board in DFU mode:

- press button BT2 before supplying the board and release it when LED D6 lights up
- or
  1. press BT3 (Reset) and BT2 together
  2. release BT3 followed by BT2.

Led D6 will light up and the device should appear in Windows Device manager as “STM device in DFU mode”.

2.2.1 DFU on Windows®

To install the DFU software, run DfuSe_Demo_V3.0.5_Setup.exe included in the software package and follow the instructions.

Step 1. Following correct installation launch the software from Start > STMicroelectronics > DfuSe > DfuSeDemo.

The typical location of the executable file is C:\Program Files (x86)\STMicroelectronics\Software\DfuSe v3.0.5\Bin\DfuSeDemo.exe.

Step 2. In the Upgrade or Verify Action section of the DfuseDemo tool click on the Choose... button and select the target .dfu file; then click the ‘Upgrade’ button to start the firmware upgrade.

For more details regarding DFU and the microcontroller ST GUI, see the user manual located typically in

– C:\Program Files (x86)\STMicroelectronics\Software\DfuSe v3.0.5\Bin\Doc\UM0412.pdf

– Start > STMicroelectronics > DfuSe > Docs > UM0412.pdf.

– The DFU utility tool and relative documentation are available on ww.st.com by searching STSW-STM32080.

2.2.2 DFU on Linux®

The DFU program for Linux operating systems is dfu-util. The procedure for Ubuntu Linux operating systems is described below.

Step 1. Open a terminal and run (with sudo to ensure the correct permissions):

```
sudo apt-get install dfu-util
```

Step 2. Create a udev rules file:

```
sudo gedit /etc/udev/45-Professional MEMS Tool.rules
```

Step 3. Fill it with the following content:

```
# 0483:5740 - STM32F4 in USB Serial Mode (CN5)
ATTRS{idVendor}=="0483", ATTRS{idProduct}=="5740",
ENV{ID_MM_DEVICE_IGNORE}="1"
ATTRS{idVendor}=="0483", ATTRS{idProduct}=="5740",
ENV{MTP_NO_PROBE}="1" SUBSYSTEMS=="usb",
ATTRS{idVendor}=="0483", ATTRS{idProduct}=="5740",
MODE:="0666"
KERNEL=="ttyACM*", ATTRS{idVendor}=="0483",
ATTRS{idProduct}=="5740", MODE:="0666"
# 0483:df11 - STM32F4 in DFU mode (CN5) SUBSYSTEMS=="usb",
ATTRS{idVendor}=="0483", ATTRS{idProduct}=="df11", MODE:="0666"
```

Step 4. Instruct udev to reload its rules:

```
sudo udevadm control --reload-rules
```

You should now be able to program the board.

Step 5. Connect the Professional MEMS Tool board in DFU mode and run:

```
sudo dfu-util -a 0 -D dfu_path/file.dfu -d 0483:df11
```

where:

- dfu_path is the path to the dfu file
- file.dfu is the dfu file name

example: `sudo dfu-util -a 0 -D Desktop/Professional MEMS ToolV2_REL_4_0.dfu -d 0483:df11`.

Step 6. Disconnect and reconnect the board to exit DFU mode and start using the board with the new firmware.

2.2.3 DFU on Mac OS®

The DFU program used for Mac operating systems is dfu-util.

Step 1. Before installing DFU on your Mac OS, you need to install Homebrew. Open a terminal and run:

```
ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/go/install)"
```

Step 2. Install dfu-utils:

```
brew install dfu-util
```

You should now be able to program the board

Step 3. Connect the Professional MEMS Tool board in DFU mode, and run:

```
dfu-util -a 0 -D dfu_path/file.dfu -d 0483:df11
```

where:

- dfu_path is the path to the dfu file
- file.dfu is the dfu file name

example: `dfu-util -a 0 -D Desktop/Professional MEMS ToolV2_REL_4_0.dfu -d 0483:df11`.

Step 4. Disconnect and reconnect the board to exit DFU mode and start using the board with the new firmware.

3 Supported MEMS adapter boards

Table 1. List of supported MEMS adapter boards

Adapter board	Device
STEVAL-MET001V1	LPS22HB
STEVAL-MKI087V1	LIS331DL
STEVAL-MKI089V1	LIS331DLH
STEVAL-MKI092V1	LIS331HH
STEVAL-MKI105V1	LIS3DH
STEVAL-MKI106V1	LSM303DLHC
STEVAL-MKI107V1	L3G4200D
STEVAL-MKI107V2	L3GD20
STEVAL-MKI108V2	9AXISMODULE v2 [LSM303DLHC + L3GD20]
STEVAL-MKI110V1	AIS328DQ
STEVAL-MKI122V1	LSM330DLC
STEVAL-MKI125V1	A3G4250D
STEVAL-MKI134V1	LIS3DSH
STEVAL-MKI135V1	LIS2DH
STEVAL-MKI136V1	L3GD20H
STEVAL-MKI137V1	LIS3MDL
STEVAL-MKI141V1	HTS221
STEVAL-MKI142V1	LPS25H
STEVAL-MKI151V1	LIS2DH12
STEVAL-MKI154V1	LSM9DS0
STEVAL-MKI158V1	AIS3624DQ
STEVAL-MKI159V1	LSM9DS1
STEVAL-MKI160V1	LSM6DS3
STEVAL-MKI161V1	LSM6DS0
STEVAL-MKI163V1	LSM303C
STEVAL-MKI164V1	LIS2HH12
STEVAL-MKI165V1	LPS25HB
STEVAL-MKI166V1	H3LIS100DL
STEVAL-MKI167V1	H3LIS200DL
STEVAL-MKI168V1	IIS2DH
STEVAL-MKI169V1	I3G4250D
STEVAL-MKI170V1	IIS328DQ
STEVAL-MKI172V1	LSM303AGR
STEVAL-MKI173V1	LSM303AH
STEVAL-MKI174V1	LIS2DS12
STEVAL-MKI175V1	LIS2DE12
STEVAL-MKI176V1	LSM6DS3H

Adapter board	Device
STEVAL-MKI177V1	LPS35HW
STEVAL-MKI178V1	LSM6DSL
STEVAL-MKI178V2	LSM6DSL
STEVAL-MKI179V1	LIS2DW12
STEVAL-MKI180V1	LIS3DHH
STEVAL-MKI181V1	LIS2MDL
STEVAL-MKI182V1	ISM330DLC
STEVAL-MKI183V1	LPS33HW
STEVAL-MKI184V1	ISM303DAC
STEVAL-MKI185V1	IIS2MDC
STEVAL-MKI186V1	IIS3DHHC
STEVAL-MKI188V1	L20G20IS
STEVAL-MKI189V1	LSM6DSM
STEVAL-MKI190V1	LIS2DTW12
STEVAL-MKI191V1	IIS2DLPC
STEVAL-MKI192V1	LPS22HH
STEVAL-MKI193V1	ASM330LHH
STEVAL-MKI194V1	LSM6DSR
STEVAL-MKI195V1	LSM6DSRX
STEVAL-MKI196V1	LSM6DSO
STEVAL-MKI197V1	LSM6DSOX
STEVAL-MKI198V1K	STTS751
STEVAL-MKI199V1K	STLM20
STEVAL-MKI201V1K	STTS75
STEVAL-MKI202V1K	STDS75
STEVAL-MKI203V1K	STCN75
STEVAL-MKI204V1K	STLM75
STEVAL-MKI205V1	LPS33W

4 Supported commands

The microcontroller mounted on the Professional MEMS Tool board is equipped with dedicated firmware that allows control of the digital output MEMS sensor, and acquisition of the measured data.

The firmware also handles the communication between the board and the PC through the USB bus.

4.1 Getting started

Before using the commands supported by the firmware, the following procedure must be performed:

Step 1. Connect the Professional MEMS Tool to the USB port

Step 2. Launch an application that allows sending commands through the virtual serial port. The remainder of this document assumes the use of Microsoft® HyperTerminal program available with the Windows XP operating system, but you can use any similar tool.

Step 3. Create a new connection, enter a name (e.g. STEVAL-MKI109V3), and click `OK`.

Step 4. In the `Connect Using` field, select the virtual COM port to which the USB port has been mapped, and click `OK`.

Step 5. In port settings, set `bits per second` to 115200, `data bits` to 8, `parity` to none, `stop bits` to 1, and `flow control` to none; click `OK`

Step 6. In the HyperTerminal application window, select `files > properties > settings`, then click `ASCII Setup`.

Step 7. Select `Send line ends with line feeds` and `Echo typed characters locally`

Step 8. Click `OK` to close the ASCII Setup window

Step 9. Click `OK` button to close the Properties window.

Once this procedure has been completed you can use the commands described in the following sections by typing them in the "HyperTerminal" window.

4.1.1 Quick start

The basic sequence of commands (based on the LIS3DH accelerometer) to start a data communication session and to retrieve X, Y, and Z acceleration data from the demonstration kit is:

Step 1. Connect the Professional MEMS Tool to the USB port

Step 2. Start "Microsoft© HyperTerminal" (or another similar application) and configure it as described in Section 4.1 Getting started

Step 3. Enter the `*setdb105v1` command in the HyperTerminal" window, (supposing the LIS3DH adapter board is used – for other adapters see the relevant datasheets to check the register configuration), enter the command `* Zoff` to enable the control of the device by the STM32F401VE microcontroller, and `*w2047` to switch on the LIS3DH and to set the data rate to 50 Hz

Step 4. Send the `*debug` command to get the X, Y, and Z data measured by the sensor

Step 5. Send `*stop` to end the continuous acquisition and visualization.

4.2 Supported commands

The firmware supports a wide range of MEMS adapters; the complete list of supported commands and their descriptions are given below. Commands are not case sensitive.

Table 2. **List of supported commands**

Command	Description	Returned value[1]
*setdbXXXVY	Selects firmware according to the adapter connected	Device name e.g.: LIS3DH
*start	Starts continuous data acquisition	(see Table 3. Returned values for *start command)
*debug	Returns the output data in readable text format	
*stop	Stops data acquisition	
*Zon	Forces High impedance state	
*Zoff	Exits from High impedance state	
*dev	Device name	e.g.: LIS3DH
*ver	Firmware version	e.g.: V1.5.2
*board	Returns board name	
*rAA	Accelerometer register read	e.g.: RAAhDDh
*wAADD	Accelerometer register write	
*grAA	Gyroscope register read	e.g.: GRAAhDDh
*gwAADD	Gyroscope register write	
*mrAA	Magnetometer register read	e.g.: MRAAhDDh
*mwAADD	Magnetometer register write	
*prAAx	Pressure sensor register read	e.g.: PRAAhDDh
*pwAADD	Pressure sensor register write	
*hrAA	Humidity sensor register read	e.g.: HRAAhDDh
*hwAADD	Humidity sensor register write	
*trAA	Temperature sensorregister read	e.g.: TRAAhDDh
*twAADD	Temperaturesensor register write	
*single	It gets a single X, Y, and Z data acquisition	(see Table 3. Returned values for *start command)
*list	Prints the list of MKIs supported	e.g.: 087V1 089V1 092V1 105V1...
*listdev	Prints the list of devices supported	e.g.: LIS331DL LIS331DLH LIS331HH…
*echoon	Activates the write verbose mode	e.g.: RAAhDDh
*echooff	Deactivates the write verbose mode	
*fiforst	Accelerometer “Reset mode” enable	st XH XL YH YL ZH ZL IR FC FS
*fifomde	Accelerometer “FIFO mode” enable	st XH XL YH YL ZH ZL IR FC FS
*fifostr	Accelerometer “FIFO stream” enable	st XH XL YH YL ZH ZL IR FC FS
*fifostf	Accelerometer “Stream to FIFO” enable	st XH XL YH YL ZH ZL IR FC FS
*fifobtf	Accelerometer “Bypass to FIFO” enable	st XH XL YH YL ZH ZL IR FC FS
*fifobts	Accelerometer “Bypass to stream” enable	st XH XL YH YL ZH ZL IR FC FS
*fifodstr	Accelerometer “Dynamic stream” enable	st XH XL YH YL ZH ZL IR FC FS
*gfiforst	Gyroscope “Reset mode” enable	st XH XL YH YL ZH ZL IR FC FS
*gfifomde	Gyroscope “FIFO mode” enable	st XH XL YH YL ZH ZL IR FC FS
*gfifostr	Gyroscope “FIFO stream” enable	st XH XL YH YL ZH ZL IR FC FS
*gfifostf	Gyroscope “Stream to FIFO” enable	st XH XL YH YL ZH ZL IR FC FS
*gfifobtf	Gyroscope “Bypass to FIFO” enable	st XH XL YH YL ZH ZL IR FC FS
*gfifobts	Gyroscope “Bypass to stream” enable	st XH XL YH YL ZH ZL IR FC FS

Command	Description	Returned value[1]
*gfifodstr	Gyroscope “Dynamic stream” enable	st XH XL YH YL ZH ZL IR FC FS
*pfiforst	Pressure sensor “Reset mode” enable	st XH XL YH YL ZH ZL IR FC FS
*pfifomde	Pressure sensor “FIFO mode” enable	st XH XL YH YL ZH ZL IR FC FS
*pfifostr	Pressure sensor “FIFO stream” enable	st XH XL YH YL ZH ZL IR FC FS
*pfifostf	Pressure sensor “Stream to FIFO” enable	st XH XL YH YL ZH ZL IR FC FS
*pfifobtf	Pressure sensor “Bypass to FIFO” enable	st XH XL YH YL ZH ZL IR FC FS
*pfifobts	Pressure sensor “Bypass to stream” enable	st XH XL YH YL ZH ZL IR FC FS
*pfifodstr	Pressure sensor “Dynamic stream” enable	st XH XL YH YL ZH ZL IR FC FS
*POWER_ON	Turns on VDD and VDDIO power supply	
*POWER_OFF	Turns off VDD and VDDIO power supply	
*setvddaX.Y	Sets VDD voltage value “X.Y” Volts e.g.: 3.6	
*setvddioX.Y	Sets VDDIO voltage value “X.Y” Volts e.g.: 3.6	
*adc_single	Measures VDD, VDDIO, IDD, IDDIO and another values sent in binary form	adc:D1D2D3…D20
*rmAA$_1$NN	Multiple read of NN Accelerometer successive registers	RMAA$_1$hNNhDD$_1$hDD$_2$...DD$_{NN}$h
*mutli-rAA$_1$AA$_2$ AA$_3$…	Accelerometer registers multiple read	MULTI-RAA$_1$hDD$_1$h AA$_2$hDD$_2$h…. AAnDDnh
*grmAA$_1$NN	Multiple read of NN Gyroscope successive registers	GRMAA$_1$hNNhDD$_1$hDD$_2$...DD$_{NN}$h
*multi-grAA$_1$AA$_2$ AA$_3$…	Gyroscope registers multiple read	MULTI-GRAA$_1$hDD$_1$h AA$_2$hDD$_2$h…. AAnDDnh
*mrmAA$_1$NN	Multiple read of NN Magnetometer successive registers	MRMAA$_1$hNNhDD$_1$hDD$_2$...DD$_{NN}$h
*multi-mrAA$_1$AA$_2$ AA$_3$…	Magnetometer registers multiple read	MULTI-MRAA$_1$hDD$_1$h AA$_2$hDD$_2$h…. AAnDDnh
*prmAA$_1$NN	Multiple read of NN Pressure sensor successive registers	PRMAA$_1$hNNhDD$_1$hDD$_2$...DD$_{NN}$h
*multi-prAA$_1$AA$_2$ AA$_3$…	Pressure sensor registers multiple read	MULTI-PRAA$_1$hDD$_1$h AA$_2$hDD$_2$h…. AAnDDnh
*hrmAA$_1$NN	Multiple read of NN Humidity sensor successive registers	HRMAA$_1$hNNhDD$_1$hDD$_2$...DD$_{NN}$h
*multi-hrAA$_1$AA$_2$ AA$_3$…	Humidity sensor registers multiple read	MULTI-HRAA$_1$hDD$_1$h AA$_2$hDD$_2$h…. AAnDDnh
*trmAA$_1$NN	Multiple read of NN Temperature sensor successive registers	TRMAA$_1$hNNhDD$_1$hDD$_2$...DD$_{NN}$h
*multi-trAA$_1$AA$_2$ AA$_3$…	Temperature sensor registers multiple read	MULTI-TRAA$_1$hDD$_1$h AA$_2$hDD$_2$h…. AA$_n$DD$_n$h

[1] *RP: Reference pressure XLSB.MSB, IR: interrupt byte; FC: FIFO control register; FS: FIFO source register*

4.2.1 *setdbXXXVY

This command selects the part of the firmware able to handle the adapter board sensor connected to the board. For example, *setdb105V1 selects the firmware for the LIS3DH.

The D3 LED (yellow) switches on automatically.

4.2.2 *start

This command initiates continuous data acquisition. When sent, the device returns a string of bytes (plus carriage return and line feed) like `st OUT1 OUT2 OUT3 IR STP BT`.

The first two bytes are always the ASCII char `s` and `t` which correspond to the hexadecimal values {73h 74h}.

`OUT1`, `OUT2`, and `OUT3` contain the values measured at device outputs; if the output data is represented in more than 8 bits, `OUT1`, `OUT2`, and `OUT3` are split into high byte (e.g., `XH`) and low byte (e.g., `XL`). In case of 24-bit resolution for some sensors, there is also an extra-low byte (e.g., pressure data: `PXL PL PH`).

`IR` (INT1 INT2) contains the interrupt bytes and `BT` SW1|SW2 contains the bytes that describe the state of the buttons integrated on the board.

Specifically, bit#0 of the SW1|SW2 data corresponds to the status of the SW1 button on the demonstration kit board: it is set to 1 when the SW1 is pressed (otherwise 0). Bit#1 has the same behavior but is dedicated to the SW2.

`STP` (STPL STPH) contains the step counter bytes for the internal device step counter value.

The string is ended with the carriage return (`\r`) and line feed (`\n`) bytes.

Before sending the `*start` command, the device must be out of 3-state (high impedance) and some registers must be configured according to user needs. Therefore, `*start` must be preceded by a `* zoff` and some `Register Write` commands.

As data is continuously acquired, LED D1 (green) blinks according to sensor data rate selected.
Table 3. Returned values for *start command shows the format of the string returned for each device when a `*start` command is sent. Similar byte strings are returned for groups of commands related to FIFO, as is shown in Table 4. Digital output accelerometers: supported commands list, Table 5. Digital output gyroscopes: supported commands list, Table 6. Digital output magnetometer: supported commands list, Table 7. Digital output pressure sensor: supported commands list, Table 8. Digital output humidity sensor: supported commands list and Table 9. Digital output temperature sensor: supported commands list.

Table 3. Returned values for *start command

STEVAL # (Device)	Returned value[1]
STEVAL-MKI089V1 (LIS331DLH) STEVAL-MKI092V1 (LIS331HH) STEVAL-MKI105V1 (LIS3DH) STEVAL-MKI110V1 (AIS328DQ) STEVAL-MKI125V1 (A3G4250D) STEVAL-MKI134V1 (LIS3DSH) STEVAL-MKI135V1 (LIS2DH) STEVAL-MKI136V1 (L3GD20H) STEVAL-MKI151V1 (LIS2DH12) STEVAL-MKI158V1 (AIS3624DQ) STEVAL-MKI164V1 (LIS2HH12) STEVAL-MKI168V1 (IIS2DH) STEVAL-MKI170V1 (IIS328DQ) STEVAL-MKI179V1 (LIS2DW12) STEVAL-MKI180V1 (LIS3DHH) STEVAL-MKI186V1 (IIS3DHHC) STEVAL-MKI191V1 (IIS2DLPC)	s t XH XL YH YL ZH ZL int1 int2 sw1|sw2 \r \n
STEVAL-MKI087V1 (LIS331DL) STEVAL-MKI175V1 (LIS2DE12)	s t X Y Z int1 int2 sw1|sw2 \r \n
STEVAL-MKI166V1 (H3LIS100DL) STEVAL-MKI167V1 (H3LIS200DL)	s t X 0 Y 0 Z 0 int1 int2 sw1|sw2 \r \n

STEVAL # (Device)	Returned value[1]
STEVAL-MKI174V1 (LIS2DS12) STEVAL-MKI176V1 (LSM6DS3H)	s t XH XL YH YL ZH ZL int1 int2 stpL stpH 0 sw1|sw2 \r \n
STEVAL-MKI137V1 (LIS3MDL) STEVAL-MKI181V1 (LIS2MDL) STEVAL-MKI185V1 (IIS2MDC)	s t XH XL YH YL ZH ZL int1 sw1|sw2 \r \n
STEVAL-MKI107V1 (L3G4200D) STEVAL-MKI107V2 (L3GD20) STEVAL-MKI169V2 (I3G4250D) STEVAL-MKI188V1 (L20G20IS)	s t G_XH G_XL G_YH G_YL G_ZH G_ZL G_int1 G_int2 sw1|sw2 \r \n
STEVAL-MKI122V1 (LSM330DLC)	s t A_XH A_XL A_YH A_YL A_ZH A_ZL G_XH G_XL G_YH G_YL G_ZH G_ZL A_int1 A_int2 G_int1 G_int2 sw1|sw2 \r \n
STEVAL-MKI106V1(LSM303DLHC)	s t A_XH A_XL A_YH A_YL A_ZH A_ZL M_XH M_XL M_YH M_YL M_ZH M_ZL A_int1 A_int2 sw1|sw2 \r \n
STEVAL-MKI108V2 (9AXIS MODULE) STEVAL-MKI159V1 (LSM9DS1)	s t A_XH A_XL A_YH A_YL A_ZH A_ZL G_XH G_XL G_YH G_YL G_ZH G_ZL M_XH M_XL M_YH M_YL M_ZH M_ZL A_int1 G_int2 G_int3 0 sw1|sw2 \r \n
STEVAL-MKI154V1 (LSM9DS0)	s t A_XH A_XL A_YH A_YL A_ZH A_ZL G_XH G_XL G_YH G_YL G_ZH G_ZL M_XH M_XL M_YH M_YL M_ZH M_ZL A_int1 A_int2 sw1|sw2 \r \n
STEVAL-MKI159V1 (LSM9DS1)	s t A_XH A_XL A_YH A_YL A_ZH A_ZL G_XH G_XL G_YH G_YL G_ZH G_ZL M_XH M_XL M_YH M_YL M_ZH M_ZL A_int1 A_int2 G_int3 0 sw1|sw2 \r \n
STEVAL-MKI161V1 (LSM6DS0) STEVAL-MKI160V1 (LSM6DS3)	s t A_XH A_XL A_YH A_YL A_ZH A_ZL G_XH G_XL G_YH G_YL G_ZH G_ZL Int1 Int2 sw1|sw2 \r \n
STEVAL-MKI178V1 (LSM6DSL) STEVAL-MKI178V2 (LSM6DSL) STEVAL-MKI182V1 (ISM330DLC) STEVAL-MKI189V1 (LSM6DSM) STEVAL-MKI194V1 (LSM6DSR) STEVAL-MKI195V1 (LSM6DSRX) STEVAL-MKI196V1 (LSM6DSO) STEVAL-MKI197V1 (LSM6DSOX)	s t A_XH A_XL A_YH A_YL A_ZH A_ZL G_XH G_XL G_YH G_YL G_ZH G_ZL Int1 Int2 StpL StpH 0 sw1|sw2 \r \n
STEVAL-MKI163V1 (LSM303C)	s t A_XH A_XL A_YH A_YL A_ZH A_ZL M_XH M_XL M_YH M_YL M_ZH M_ZL A_int G_int sw1|sw2 \r \n
STEVAL-MKI172V1 (LSM303AGR)	s t A_XH A_XL A_YH A_YL A_ZH A_ZL M_XH M_XL M_YH M_YL M_ZH M_ZL A_int1 Aint2 G_int sw1|sw2 \r \n
STEVAL-MKI173V1 (LSM303AH) STEVAL-MKI184V1 (ISM303DAC)	s t A_XH A_XL A_YH A_YL A_ZH A_ZL M_XH M_XL M_YH M_YL M_ZH M_ZL A_int1 A_int2 M_int StpL StpH sw1|sw2 \r \n
STEVAL-MKI142V1 (LPS25H) STEVAL-MKI165V1 (LPS25HB)	s t PXL PL PH TL TH REF_PXL REF_PL REF_PH P_int1 sw1|sw2 \r \n

STEVAL # (Device)	Returned value[1]
STEVAL-MET001V1 (LPS22HB) STEVAL-MKI177V1 (LPS35HW) STEVAL-MKI183V1 (LPS33HW) STEVAL-MKI192V1 (LPS22HH) STEVAL-MKI205V1 (LPS33W)	s t PXL PL PH TL TH REF_PXL REF_PL REF_PH P_int1 sw1|sw2 \r \n
STEVAL-MKI141V1 (HTS221)	s t HL HH TL TH H_int1 sw1|sw2 \r \n
STEVAL-MKI190 (LIS2DTW12) STEVAL-MKI193 (ASM300LHH)	s t A_XH A_XL A_YH A_YL A_ZH A_ZL Int1 Int2 TL TH sw1|sw2 \r \n
STEVAL-MKI198V1K (STTS751)	s t TH TL Int1 sw1|sw2 \r \n
STEVAL-MKI199V1K (STLM20)	s t TH TL sw1|sw2 \r \n
STEVAL-MKI201V1K (STTS75) STEVAL-MKI202V1K (STDS75) STEVAL-MKI203V1K (STCN75) STEVAL-MKI204V1K (STLM75)	s t TH TL Int1 sw1|sw2 \r \n

1. *XH: X-axis output high byte (same for Y axis, Z axis, P pressure, H humidity, and T temperature). XL: X-axis output low byte (same for Y axis, Z axis, P pressure, H humidity, and T temperature)*

4.2.3 *debug

This command starts continuous data acquisition in debug mode. When sent to the board, it returns the output values measured by the device formatted in a readable text format.

4.2.4 *stop

This command interrupts any acquisition session that has been started with either the `*start` or `*debug` commands.

4.2.5 *Zon and *Zoff

These commands put the STM32F401VE microcontroller on the demonstration kit in 3-state (high impedance). They allow the isolation of the sensor from the microprocessor and let the user interact with the sensor in a purely analog fashion.

When the kit is first turned on, the lines are in 3-state (high impedance) mode and you must send the `* Zoff` command to allow communication between the sensor and the microcontroller.

After this command has been executed, LED D2 (Red) is turned on. If the `* Zoff` command has not been launched, the firmware ignores any other command sent to sensor.

4.2.6 *dev

This command retrieves the name of the adapter connected to the demonstration kit; e.g., `LIS3DH`.

4.2.7 *ver

This command returns the version of the firmware loaded in the microprocessor; e.g., `V1.5.2`.

4.2.8 *rAA

This command reads the contents of the accelerometer registers in the demonstration kit board. The hexadecimal value `AA` written in upper case represents the address of the register to be read.

Once the read command is issued, the board returns `RAAhDDh`, where `AA` is the address sent by the user and `DD` is the data present in the register.

For example, to read the register at address 0x20, the user issues the command `*r20`, which would return a result like `R20hC7h`.

*4.2.8.1 *rmAA1NN*

This command allows the contents of multiple accelerometer registers in the demonstration kit board to be read in single data block. Once this command is issued, the board returns a set of NN values starting with

RMAA$_1$hNNhDD$_1$hDD$_2$h… DD$_{NN}$h where AA$_1$ is the starting address set by user and DD$_1$ is the data present in this register and so on for next registers.

For example, *rm2006 reads six registers starting from 0x20, which would return a result like RM20h06h27h00h00h00hA0h0Bh.

4.2.8.2 *multi-rAA1AA2AA3...AAN

This command reads multiple accelerometer registers in the demonstration kit board in a single data block. Once this command is issued, the board returns set of N values starting RAA$_1$hDD$_1$h… AA$_N$hDDNh where AA$_1$ is the starting address set by user and DD$_1$ is the data present in this register.

For example, *multi-r202425292B2D reads six register starting from 0x20, which would return a result like MULTI-R20h27h24hA0h25h0Bh29hE0h2Bh3Fh2Dh90h.

4.2.9 *wAADD

This command writes the contents of the accelerometer registers in the demonstration kit board. The hexadecimal upper case values AA and DD represent the address of the register and the data to be written, respectively.

For example, *w20C7 writes 0xC7 to the register at address 0x20

4.2.10 *grAA

This command allows the contents of the gyroscope registers in the demonstration kit board to be read. The hexadecimal, upper case AA represents the address of the register to be read.

Once the read command is issued, the board returns GRAAhDDh, where AA is the address sent by the user and DD is the data present in the register.

For example, *gr20 reads the register at address 0x20, which would return a result like GR20hC7h.

4.2.10.1 *grmAA1NN

This command allows the contents of multiple gyroscope registers in the demonstration kit board to be read in single data block. Once this command is issued, the board returns set of NN values starting with GRMAA$_1$hNNhDD$_1$hDD$_2$h… DD$_{NN}$h where AA$_1$ is the starting address set by user and DD$_1$ is the data present in this register and so on.

For example, *grm2006 reads six registers starting from 0x20, which would return a result like GRM20h06h27h00h00h00hA0h0Bh.

4.2.10.2 *multi-grAA1AA2AA3...AAN

This command reads multiple gyroscope registers in the demonstration kit board in a single data block. Once this command is issued, the board returns set of N values starting with MULTI-GRAA$_1$hDD$_1$h… AA$_N$hDDNh where AA$_1$ is the starting address set by user and DD$_1$ is the data present in this register and so on.

For example, *multi-gr202425292B2D reads six registers starting from 0x20, which would return a result like MULTI-GR20h27h24hA0h25h0Bh29hE0h2Bh3Fh2Dh90h.

4.2.11 *gwAADD

This command writes the contents of the gyroscope registers in the demonstration kit board. The hexadecimal, upper case. AA and DD represent the address of the register and the data to be written, respectively.

For example, *gw20C7 writes 0xC7 to the register at address 0x20.

4.2.12 *mrAA

This command allows the contents of the magnetometer registers in the demonstration kit board to be read. The hexadecimal, upper case AA represents the address of the register to be read.

Once the read command is issued, the board returns MRAAhDDh, where AA is the address sent by the user and DD is the data present in the register.

For example, *mr00 reads the register at address 0x00, which would return a result like MR00h10h.

4.2.12.1 *mrmAA1NN

This command readss the contents of multiple magnetometer registers in the demonstration kit board in a single data block. Once this command is issued, the board returns set of NN values starting with RMAA$_1$hNNhDD$_1$hDD$_2$h… DD$_{NN}$h where AA$_1$ is the starting address set by user and DD$_1$ is the data present in this register.

For example, `*mrm2006` reads six register starting from 0x20, which would return a result like `MRM20h06h27h00h00h00hA0h0Bh`.

4.2.12.2 *multi-mrAA1AA2AA3...AAN

This command reads multiple magnetometer registers in the demonstration kit board in a single data block. Once this command is issued, the board returns set of N values starting with MULTI-MRAA$_1$hDD$_1$h... AA$_N$hDDNh, where AA$_1$ is the starting address set by user and DD$_1$ is the data present in this register, and so on…

For example, `*multi-mr202425292B2D` reads six registers starting from 0x20, which would return a result like `MULTI-MR20h27h24hA0h25h0Bh29hE0h2Bh3Fh2Dh90h`.

4.2.13 *mwAADD

This command writes the contents of the magnetometer registers in the demonstration kit board. Hexadecimal, upper case `AA` and `DD` represent the address of the register and the data to be written, respectively.

For example, `*mw0120` writes 0x20 to the register at address 0x01.

4.2.14 *prAA

This command reads the contents of the pressure sensor registers in the demonstration kit board. The hexadecimal, upper case `AA` represents the address of the register to be read.

Once the read command is issued, the board returns `PRAAhDDh`, where `AA` is the address sent by the user and `DD` is the data present in the register.

For example, `*pr20` reads the register at address 0x20, which would return a value like `PR20h10h`.

4.2.14.1 *prmAA1NN

This command reads the contents of multiple pressure sensor registers in the demonstration kit in a single data block. Once this command is issued, the board returns set of NN values starting with PRMAA$_1$hNNhDD$_1$hDD$_2$h... DD$_{NN}$h where AA$_1$ is the starting address set by user and DD$_1$ is the data present in this register, and so on…

For example, `*prm2006` reads six registers starting from 0x20, which would return a value like `PRM20h06h27h00h00h00hA0h0Bh`.

4.2.14.2 *multi-prAA1AA2AA3...AAN

This command reads multiple pressure sensor registers in the demonstration kit board in a single data block. Once this command is issued, the board returns set of N values starting with MULTI-PRAA$_1$hDD$_1$h... AA$_N$hDDNh where AA$_1$ is the starting address set by user and DD$_1$ is the data present in this register, and so on.

For example, `*multi-pr202425292B2D` reads six registers starting from 0x20, which would return a result like `MULTI-PR20h27h24hA0h25h0Bh29hE0h2Bh3Fh2Dh90h`.

4.2.15 *pwAADD

This command writes the contents of the pressure sensor registers in the demonstration kit board. The hexadecimal, upper case `AA` and `DD` represent the address of the register and the data to be written, respectively.

For example, `*pw20C7` writes 0xC7 to the register at address 0x20.

4.2.16 *hrAA

This command reads the contents of the humidity sensor registers in the demonstration kit board. The hexadecimal, upper case `AA` represents the address of the register to be read.

Once the read command is issued, the board returns `HRAAhDDh`, where `AA` is the address sent by the user and `DD` is the data present in the register.

For example, `*hr20` reads the register at address 0x20, which would return a result like `HR20h10h`.

4.2.16.1 *hrmAA1NN

This command reads the contents of multiple humidity sensor registers in the demonstration kit board in a single data block. Once this command is issued, the board returns set of NN values starting with HRMAA$_1$hNNhDD$_1$hDD$_2$h... DD$_{NN}$h where AA$_1$ is the starting address set by user and DD$_1$ is the data present in this register, and so on.

For example, `*hrm2006` reads six registers starting from 0x20, which would return a result like `HRM20h06h27h00h00h00hA0h0Bh`.

*4.2.16.2 *multi-hrAA1AA2AA3...AAN*

This command reads multiple humidity sensor registers in the demonstration kit board in a single data block. Once this command is issued, the board returns set of N values starting with MULTI-HRAA$_1$hDD$_1$h… AA$_N$hDDNh where AA$_1$ is the starting address set by user and DD$_1$ is the data present in this register, and so on.

For example, `*multi-hr202425292B2D` reads six registers starting from 0x20, which would return a result like `MULTI-HR20h27h24hA0h25h0Bh29hE0h2Bh3Fh2Dh90h`.

4.2.17 *hwAADD

This command writes the contents of the humidity sensor registers in the demonstration kit board. The hexadecimal, upper case `AA` and `DD` represent the address of the register and the data to be written, respectively.

For example `*hw20C7` writes 0xC7 to the register at address 0x20.

4.2.18 *trAA

This command reads the contents of the temperature sensor registers in the demonstration kit board. The hexadecimal, upper case `AA` represents the address of the register to be read. Once the read command is issued, the board returns `TRAAhDDh`, where `AA` is the address sent by the user and `DD` is the data present in the register. For example, `*tr20` reads the register at address 0x20, which would return a result like `TR20h10h`.

*4.2.18.1 *trmAA1NN*

This command reads the contents of multiple temperature sensor registers in the demonstration kit board in a single data block. Once this command is issued, the board returns set of `NN` values starting with TRMAA$_1$hNNhDD$_1$hDD$_2$h… DD$_{NN}$h where AA$_1$ is the starting address set by user and DD$_1$ is the data present in this register, and so on. For example, `*trm2006` reads six registers starting from 0x20, which would return a result like `TRM20h06h27h00h00h00hA0h0Bh`.

*4.2.18.2 *multi-trAA1AA2AA3...AAN*

This command reads multiple temperature sensor registers in the demonstration kit board in a single data block. Once this command is issued, the board returns set of `N` values starting with MULTI-TRAA$_1$hDD$_1$h… AANhDDNh where AA$_1$ is the starting address set by user and DD$_1$ is the data present in this register, and so on. For example, `*multi-tr202425292B2D` reads six registers starting from 0x20, which would return a result like `MULTI-TR20h27h24hA0h25h0Bh29hE0h2Bh3Fh2Dh90h`.

4.2.19 *twAADD

This command writes the contents of the temperature sensor registers in the demonstration kit board. The hexadecimal, upper case `AA` and `DD` represent the address of the register and the data to be written, respectively. For example `*tw20C7` writes 0xC7 to the register at address 0x20.

4.2.20 *single

This command may be used to read just one set of data. It returns the read values of one data sample if the sensor is configured properly.

4.2.21 *list

The command returns the list of MKI adapters supported by the firmware in ASCII format.

4.2.22 *listdev

This command returns the list of devices supported by the firmware in ASCII format.

4.2.23 *echoon

This command is used to activate the write command verbose mode so that the firmware automatically reads the contents of a register that has just been written to check if the write was successful.

For example, `* echoon` launched after `*w2027` returns `R2027`.

4.2.24 *echooff

This command stops the write command verbose mode.

4.2.25 *fiforst

This command enables the accelerometer FIFO reset mode. For more details see application note AN3308 on www.st.com.

4.2.26 *fifomde

This command enables the accelerometer FIFO mode. For more details see application note AN3308 on www.st.com.

4.2.27 *fifostr

This command enables the accelerometer FIFO stream mode. For more details see application note AN3308 on www.st.com.

4.2.28 *fifostf

This command enables the accelerometer Stream-to-FIFO mode. For more details see application note AN3308 on www.st.com.

4.2.29 *fifobtf

This command enables the accelerometer Bypass-to-FIFO mode.

4.2.30 *fifobts

This command enables the accelerometer Bypass-to-Stream mode.

4.2.31 *fifodstr

This command enables the accelerometer Dynamic Stream mode.

4.2.32 *gfiforst

This command enables the gyroscope FIFO reset mode.

4.2.33 *gfifomde

This command enables the gyroscope FIFO mode.

4.2.34 *gfifostr

This command enables the gyroscope FIFO stream mode.

4.2.35 *gfifostf

This command enables the gyroscope Stream-to-FIFO mode.

4.2.36 *gfifobtf

This command enables the gyroscope Bypass-to-FIFO mode.

4.2.37 *gfifobts

This command enables the gyroscope Bypass-to-Stream mode.

4.2.38 *gfifodstr

This command enables the gyroscope Dynamic Stream mode.

4.2.39 *pfiforst

This command enables the pressure sensor FIFO reset mode.

4.2.40 *pfifomde

This command enables the pressure sensor FIFO mode.

4.2.41 *pfifostr

This command enables the pressure sensor FIFO stream mode.

4.2.42 *pfifostf

This command enables the pressure sensor Stream-to-FIFO mode.

4.2.43 *pfifobtf

This command is used to enable the pressure sensor Bypass-to-FIFO mode.

4.2.44 *pfifobts

This command is used to enable the pressure sensor Bypass-to-Stream mode.

4.2.45 *pfifodstr

This command enables the pressure sensor Dynamic Stream mode.

4.2.46 *setvddaX.Y and *setvddioX.Y

These commands set the power supply VDD and VDDIO voltage values of device adapter.

For example, `*setvdda3.6` sets 3.6V on VDDA. Note that both voltages VDD and VDDIO for modules are independent. If `* setvdda` and `* setvddio` have not been sent before the `* power_on` command, the setting is the default voltage defined in the device datasheet selected during the `* setdb` initialization. Please refer to device datasheet regarding specified VDD and VDDIO values and their relationship (i.e., which of these can be higher, etc.).

As the maximum voltages are defined also by the `* setdb` command, you cannot set values above these limits.

4.2.47 *power_on and *power_off

These commands are used to switch on and to switch off the VDD and VDDIO power supplies of the device adapter together. The proper power-on sequence of both voltages is handled internally by firmware.

4.2.48 *adc_single

This command is used to get a one-shot set of measured values of VDD and VDDIO voltages; IDD and IDDIO currents, etc.

4.3 Digital output accelerometers: supported commands

Table 4. Digital output accelerometers: supported commands list

Command	Description	Returned value[1]
*setdbXXXVY	Selects firmware according to the adapter connected	
*start	Starts continuous data acquisition	(see Table 3. Returned values for *start command)
*debug	Returns the output data in readable text format	
*stop	Stops data acquisition	
*Zon	Forces High impedance state	
*Zoff	Exits from High impedance state	
*dev	Device name	e.g.: LIS3DH
*ver	Firmware version	e.g.: V1.5.2
*rAA	Accelerometer register read	e.g.: RAAhDDh
*rmAA$_1$NN	Multiple accelerometer registers read	e.g.: RMAA$_1$hNNhDD$_1$h…DD$_{NN}$h
*multi-rAA$_1$ .. AA$_N$	Multiple accelerometer registers read	e.g.:MULTI-RAA$_1$hDD$_1$h…AA$_N$hDDNh
*wAADD	Accelerometer register write	
*single	It gets a single X, Y, and Z data acquisition	
*list	Prints the list of MKIs supported	e.g.: 087V1 089V1 092V1 105V1...
*listdev	Prints the list of devices supported	e.g.: LIS331DL LIS331DLH LIS331HH…
*echoon	Activates the write verbose mode	e.g.: RAAhDDh
*echooff	Deactivates the write verbose mode	
*fiforst [2]	Accelerometer “Reset mode” enable	st XH XL YH YL ZH ZL IR FC FS \r \n

Command	Description	Returned value[1]
*fifomde[2]	Accelerometer “FIFO mode” enable	st XH XL YH YL ZH ZL IR FC FS \r \n
*fifostr[2]	Accelerometer “FIFO stream” enable	st XH XL YH YL ZH ZL IR FC FS \r \n
*fifostf[2]	Accelerometer “Stream-to-FIFO” enable	st XH XL YH YL ZH ZL IR FC FS \r \n
*fifobtf[2]	Accelerometer “Bypass-to-FIFO” enable	st XH XL YH YL ZH ZL IR FC FS \r \n
*fifobts[2]	Accelerometer “Bypass-to-Stream” enable	st XH XL YH YL ZH ZL IR FC FS \r \n
*fifodstr[2]	Accelerometer “Dynamic Stream” enable	st XH XL YH YL ZH ZL IR FC FS \r \n

1. *IR: interrupt bytes; FC: FIFO control register; FS: FIFO source register*
2. *Available only for devices with embedded FIFO*

4.4 Digital output gyroscopes: supported commands

Table 5. Digital output gyroscopes: supported commands list below lists the commands supported by the devices/ demonstration boards including a digital output gyroscope:

Table 5. Digital output gyroscopes: supported commands list

Command	Description	Returned value[1]
*setdbXXXVY	Selects FW according to the adapter connected	
*start	Starts continuous data acquisition	(see Table 3. Returned values for *start command)
*debug	Returns the output data in readable text format	
*stop	Stops data acquisition	
*Zon	Forces High impedance state	
*Zoff	Exits from High impedance state	
*dev	Device name	e.g.: LIS3DH
*ver	Firmware version	e.g.: V1.5.2
*grAA	Gyroscope register read	e.g.: GRAAhDDh
*grmAA$_1$NN	Multiple gyroscope registers read	e.g.: GRMAA$_1$hNNhDD$_1$h…DD$_{NN}$h
*multi-grAA$_1$ .. AA$_N$	Multiple gyroscope registers read	e.g.:MULTI-GRAA$_1$hDD$_1$h…AA$_N$hDDNh
*gwAADD	Gyroscope register write	
*single	It gets a single X, Y, and Z data acquisition	
*list	Prints the list of MKIs supported	e.g.: 087V1 089V1 092V1 105V1...
*listdev	Prints the list of devices supported	e.g.: LIS331DL LIS331DLH LIS331HH…
*echoon	Activates the write verbose mode	e.g.: GRAAhDDh
*echooff	Deactivates the write verbose mode	
*gfiforst [2]	Gyroscope “Reset mode” enable	st XH XL YH YL ZH ZL IR FC FS \r \n
*gfifomde[2]	Gyroscope “FIFO mode” enable	st XH XL YH YL ZH ZL IR FC FS \r \n
*gfifostr[2]	Gyroscope “FIFO stream” enable	st XH XL YH YL ZH ZL IR FC FS \r \n
*gfifostf[2]	Gyroscope “Stream to FIFO” enable	st XH XL YH YL ZH ZL IR FC FS \r \n
*gfifobtf[2]	Gyroscope “Bypass to FIFO” enable	st XH XL YH YL ZH ZL IR FC FS \r \n
*gfifobts[2]	Gyroscope “Bypass to stream” enable	st XH XL YH YL ZH ZL IR FC FS \r \n
*gfifodstr[2]	Gyroscope “Dynamic stream” enable	st XH XL YH YL ZH ZL IR FC FS \r \n

1. *IR: interrupt bytes; FC: FIFO control register; FS: FIFO source register*

2. *Available only for devices with embedded FIFO*

4.5 Digital output magnetometers: supported commands

Table 6. Digital output magnetometer: supported commands list

Command	Description	Returned value[1]
*setdbXXXVY	Selects firmware according to the adapter connected	
*start	Starts continuous data acquisition	(see Table 3. Returned values for *start command)
*debug	Returns the output data in readable text format	
*stop	Stops data acquisition	
*Zon	Forces High impedance state	
*Zoff	Exits from High impedance state	
*dev	Device name	e.g.: LIS3DH
*ver	Firmware version	e.g.: V1.5.2
*mrAA	Magnetometer register read	e.g.: MRAAhDDh
*mrmAA$_1$NN	Multiple magnetometer registers read	e.g.: MRMAA$_1$hNNhDD$_1$h…DD$_{NN}$h
*multi-mrAA$_1$...AA$_N$	Multiple magnetometer registers read	e.g.:MULTI-MRAA$_1$hDD$_1$h…AA$_N$hDDNh
*mwAADD	Magnetometer register write	
*single	It gets a single X, Y, and Z data acquisition	
*list	Prints the list of MKIs supported	e.g.: 087V1 089V1 092V1 105V1...
*listdev	Prints the list of devices supported	e.g.: LIS331DL LIS331DLH LIS331HH…
*echoon	Activates the write verbose mode	e.g.: MRAAhDDh
*echooff	Deactivates the write verbose mode	

1. *IR: interrupt bytes; FC: FIFO control register; FS: FIFO source register*

4.6 Digital output pressure sensor: supported commands

Table 7. Digital output pressure sensor: supported commands list

Command	Description	Returned value[1]
*setdbXXXVY	Selects firmware according to the adapter connected	
*start	Starts continuous data acquisition	(see Table 3. Returned values for *start command)
*debug	Returns the output data in readable text format	
*stop	Stops data acquisition	
*Zon	Forces High impedance state	
*Zoff	Exits from High impedance state	
*dev	Device name	e.g.: LIS3DH
*ver	Firmware version	e.g.: V1.5.2
*prAA	Pressure sensor register read	e.g.: PRAAhDDh
*prmAA$_1$NN	Multiple pressure sensor registers read	e.g.: PRMAA$_1$hNNhDD$_1$h…DD$_{NN}$h
*multi-prAA$_1$ .. AA$_N$	Multiple pressure sensor registers read	e.g.:MULTI-PRAA$_1$hDD$_1$h…AA$_N$hDDNh
*pwAADD	Pressure sensor register write	
*single	It gets a single point data acquisition	

Command	Description	Returned value[1]
*list	Prints the list of MKIs supported	e.g.: 087V1 089V1 092V1 105V1...
*listdev	Prints the list of devices supported	e.g.: LIS331DL LIS331DLH LIS331HH…
*echoon	Activates the write verbose mode	e.g.: PRAAhDDh
*echooff	Deactivates the write verbose mode	
*pfiforst [2]	Pressure sensor “Reset mode” enable	st PXL PL PH TL TH IR FC FS \r \n
*pfifomde[2]	Pressure sensor “FIFO mode” enable	st PXL PL PH TL TH IR FC FS \r \n
*pfifostr[2]	Pressure sensor “FIFO stream” enable	st PXL PL PH TL TH IR FC FS \r \n
*pfifostf[2]	Pressure sensor “Stream to FIFO” enable	st PXL PL PH TL TH IR FC FS \r \n
*pfifobtf[2]	Pressure sensor “Bypass to FIFO” enable	st PXL PL PH TL TH IR FC FS \r \n
*pfifobts[2]	Pressure sensor “Bypass to stream” enable	st PXL PL PH TL TH IR FC FS \r \n
*pfifodstr[2]	Pressure sensor “Dynamic stream” enable	st PXL PL PH TL TH IR FC FS \r \n

[1] *IR: interrupt bytes; FC: FIFO control register; FS: FIFO source register*

[2] *Available only for devices with embedded FIFO*

4.7 Digital output humidity sensor: supported commands

Table 8. Digital output humidity sensor: supported commands list below lists the commands supported by the devices/demonstration boards including a digital output humidity sensor:

Table 8. Digital output humidity sensor: supported commands list

Command	Description	Returned value
*setdbXXXVY	Selects firmware according to the adapter connected	
*start	Starts continuous data acquisition	(see Table 3. Returned values for *start command)
*debug	Returns the output data in readable text format	
*stop	Stops data acquisition	
*Zon	Forces High impedance state	
*Zoff	Exits from High impedance state	
*dev	Device name	e.g.: LIS3DH
*ver	Firmware version	e.g.: V1.5.2
*hrAA	Humidity sensor register read	e.g.: HRAAhDDh
*hrmAA$_1$NN	Multiple humidity sensor registers read	e.g.: HRMAA$_1$hNNhDD$_1$h…DD$_{NN}$h
*multi-hrAA$_1$ .. AA$_N$	Multiple humidity sensor registers read	e.g.:MULTI-HRAA$_1$hDD$_1$h…AA$_N$hDDNh
*hwAADD	Humidity sensor register write	
*single	It gets a single point data acquisition	
*list	Prints the list of MKIs supported	e.g.: 087V1 089V1 092V1 105V1...
*listdev	Prints the list of devices supported	e.g.: LIS331DL LIS331DLH LIS331HH…
*echoon	Activates the write verbose mode	e.g.: HRAAhDDh
*echooff	Deactivates the write verbose mode	

4.8 Digital output temperature sensor: supported commands

Table below lists the commands supported by the devices/evaluation boards including a digital output temperature sensor:

Table 9. Digital output temperature sensor: supported commands list

Command	Description	Returned value
*setdbXXXVY	Selects firmware according to the adapter connected	
*start	Starts continuous data acquisition	(See: Table 3. Returned values for *start command)
*debug	Returns the output data in readable textformat	
*stop	Stops data acquisition	
*Zon	Forces 3-state	
*Zoff	Exits from 3-state	
*dev	Device name	e.g.:LIS3DH
*ver	Firmware version	e.g.: V1.5.2
*trAA	Temperature sensor register read	e.g.:TRAAhDDh
*trmAA$_1$NN	Multiple temperature sensor registers read	e.g.: TRMAA$_1$hNNhDD$_1$h…DD$_{NN}$h
*multi-trAA$_1$ .. AAN	Multiple temperature sensor registers read	e.g.:MULTI-TRAA$_1$hDD$_1$h…AA$_N$hDD$_N$h
*twAADD	Temperature sensor register write	
*single	It gets a single point data acquisition	
*list	Prints the list of MKIs supported	e.g.: 087V1 089V1 092V1 105V1...
*listdev	Prints the list of devices supported	e.g.: LIS331DL LIS331DLH LIS331HH…
*echoon	Activates the write verbose mode	e.g.: TRAAhDDh
*echooff	Deactivates the write verbose mode	

5 Schematic diagrams

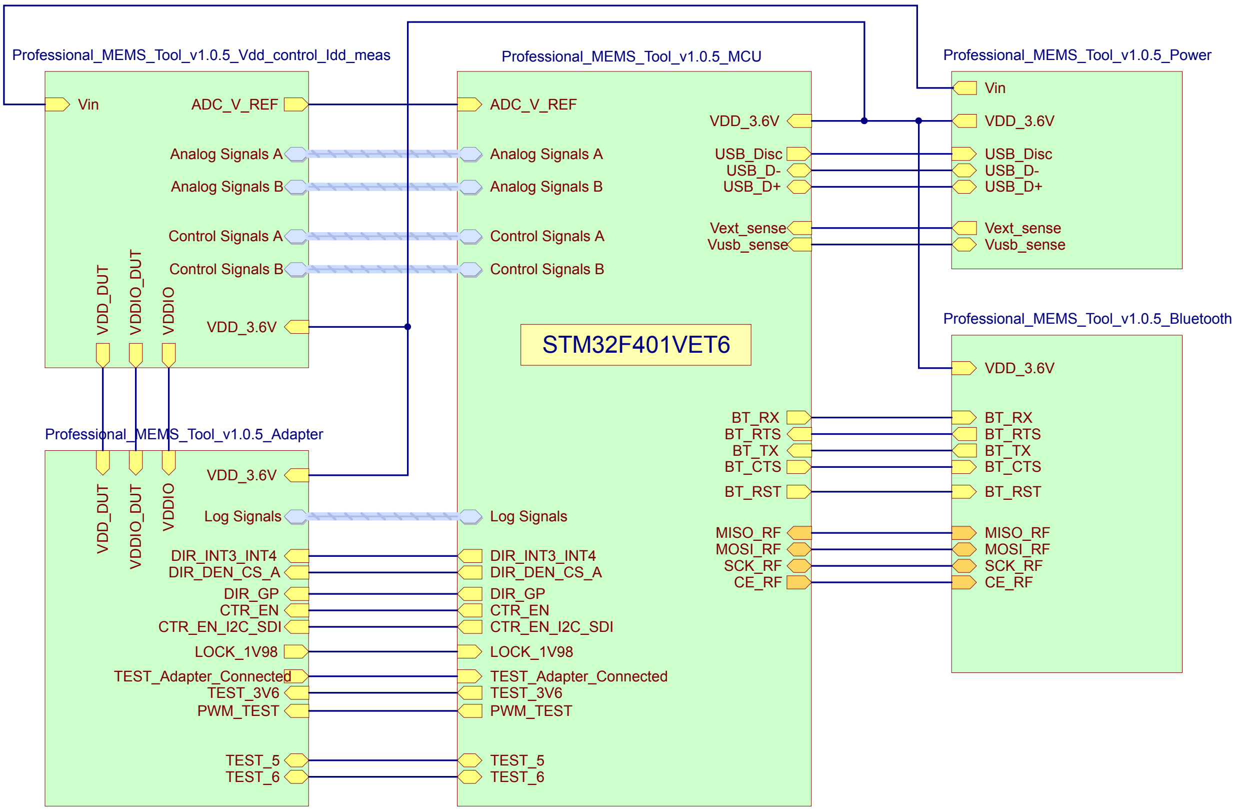


Figure 7. STEVAL-MKI109V3 circuit schematic (1 of 8)

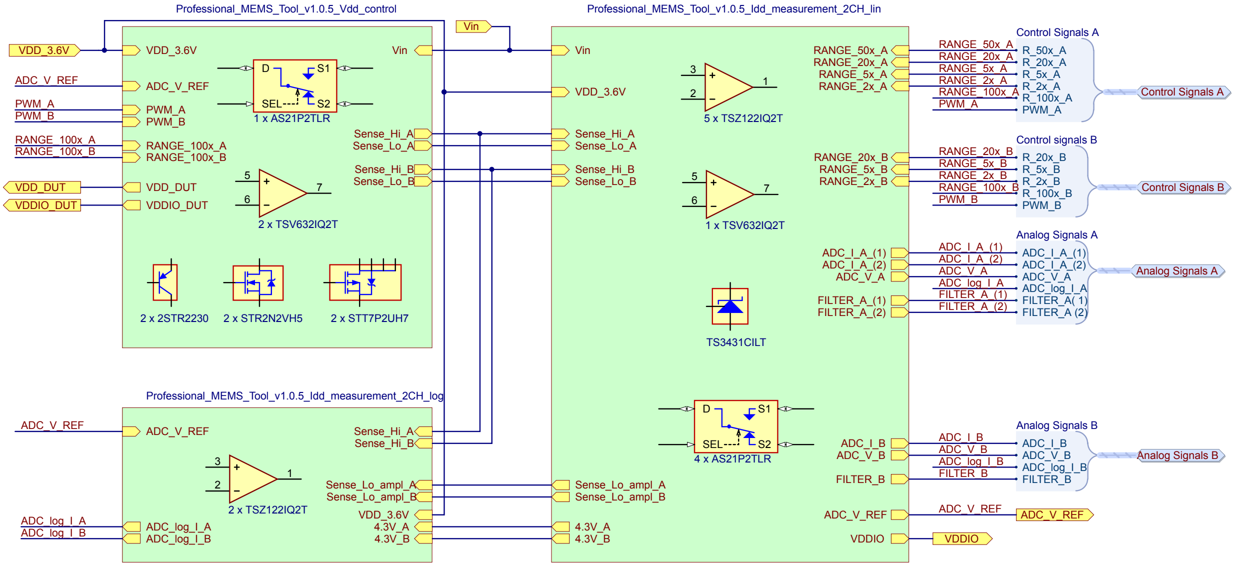


Figure 8. STEVAL-MKI109V3 circuit schematic (2 of 8)

Figure 9. **STEVAL-MKI109V3 circuit schematic (3 of 8)**

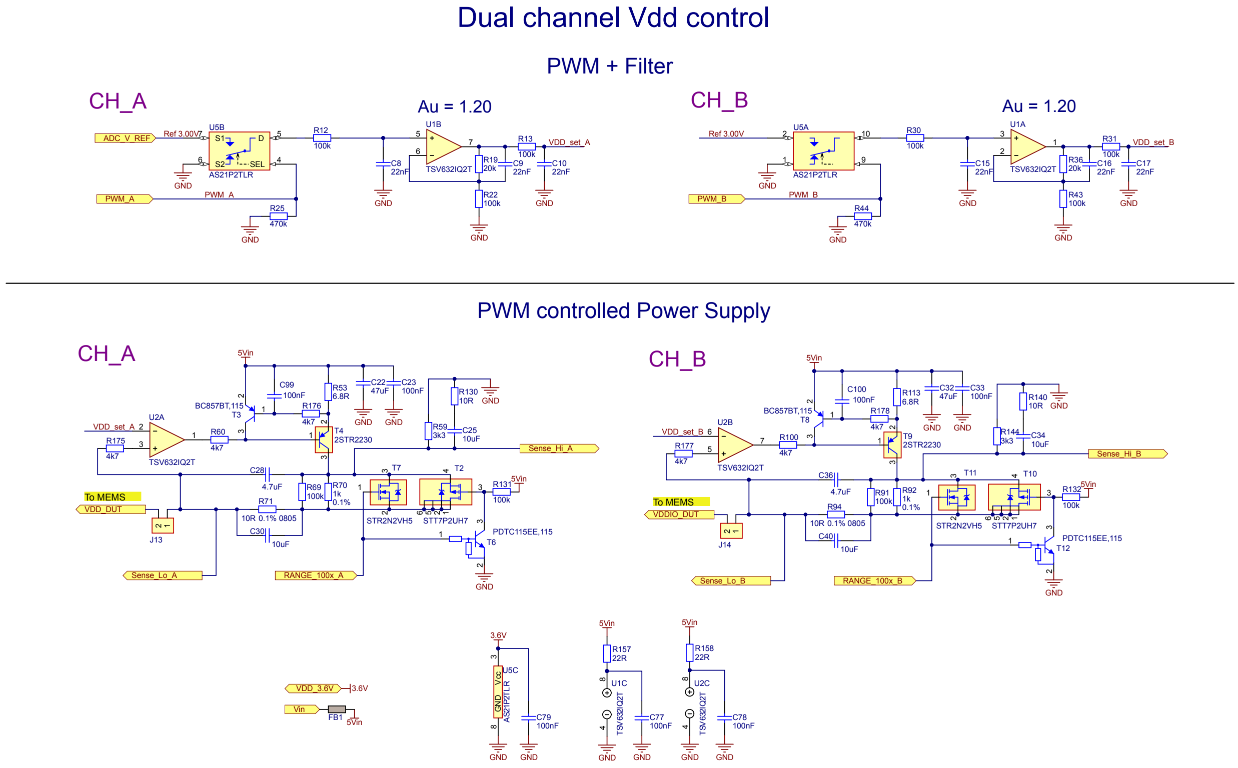


Figure 10. **STEVAL-MKI109V3 circuit schematic (4 of 8)**

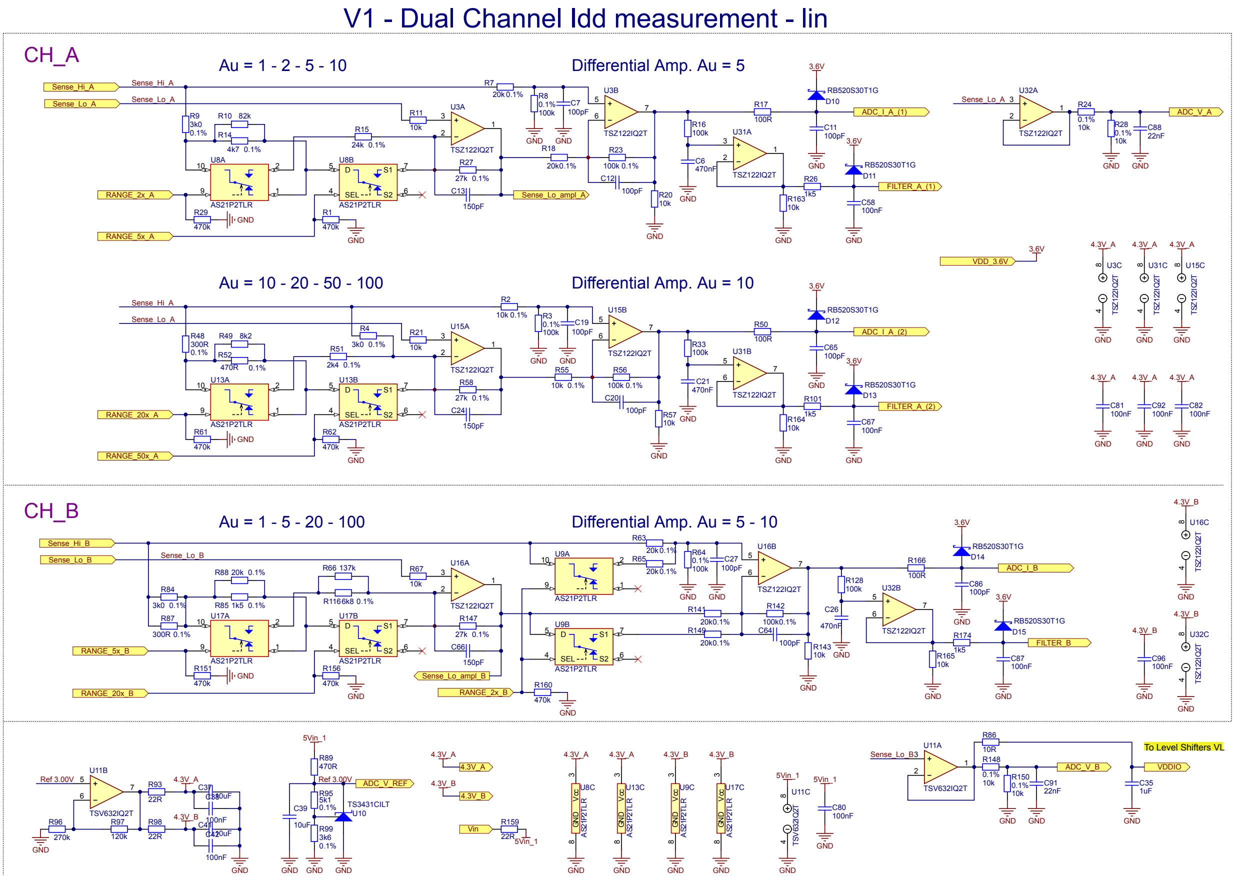
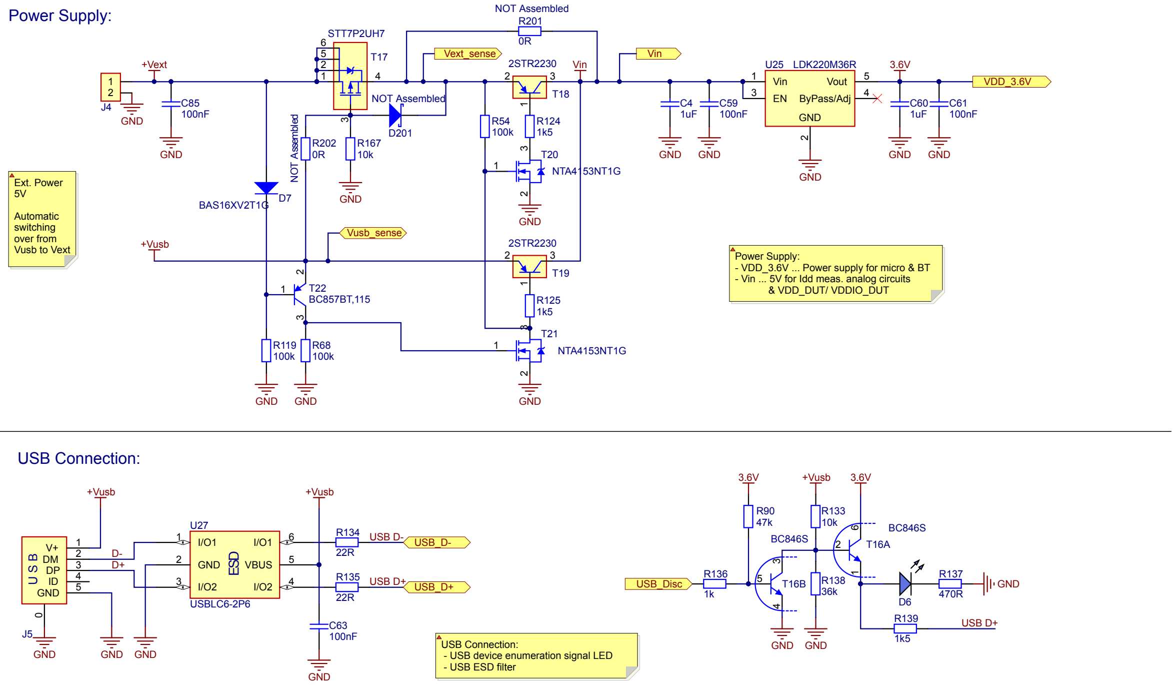


Figure 11. STEVAL-MKI109V3 circuit schematic (5 of 8)



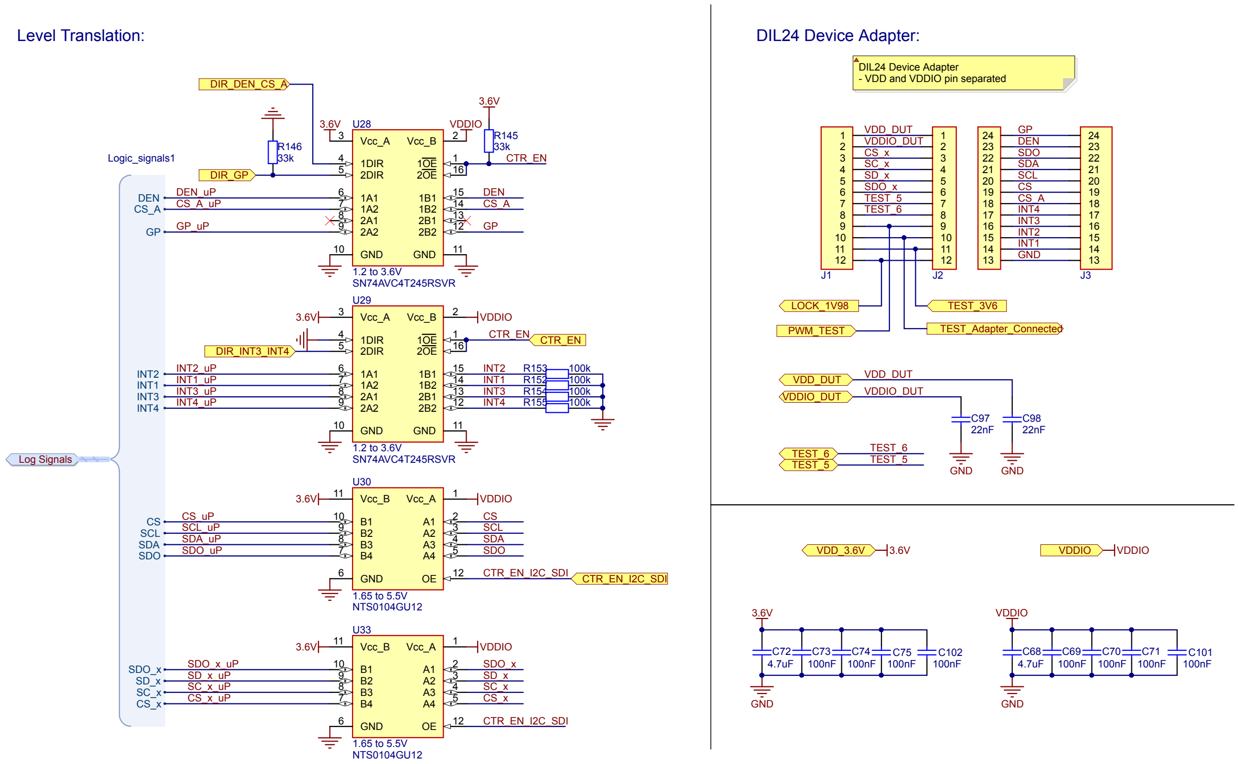
Ext. Power
5V

Automatic
switching
over from
Vusb to Vext

Power Supply:
- VDD_3.6V ... Power supply for micro & BT
- Vin ... 5V for Idd meas. analog circuits
& VDD_DUT/ VDDIO_DUT

USB Connection:
- USB device enumeration signal LED
- USB ESD filter

Figure 12. STEVAL-MKI109V3 circuit schematic (6 of 8)



DIL24 Device Adapter
- VDD and VDDIO pin separated

Figure 13. STEVAL-MKI109V3 circuit schematic (7 of 8)

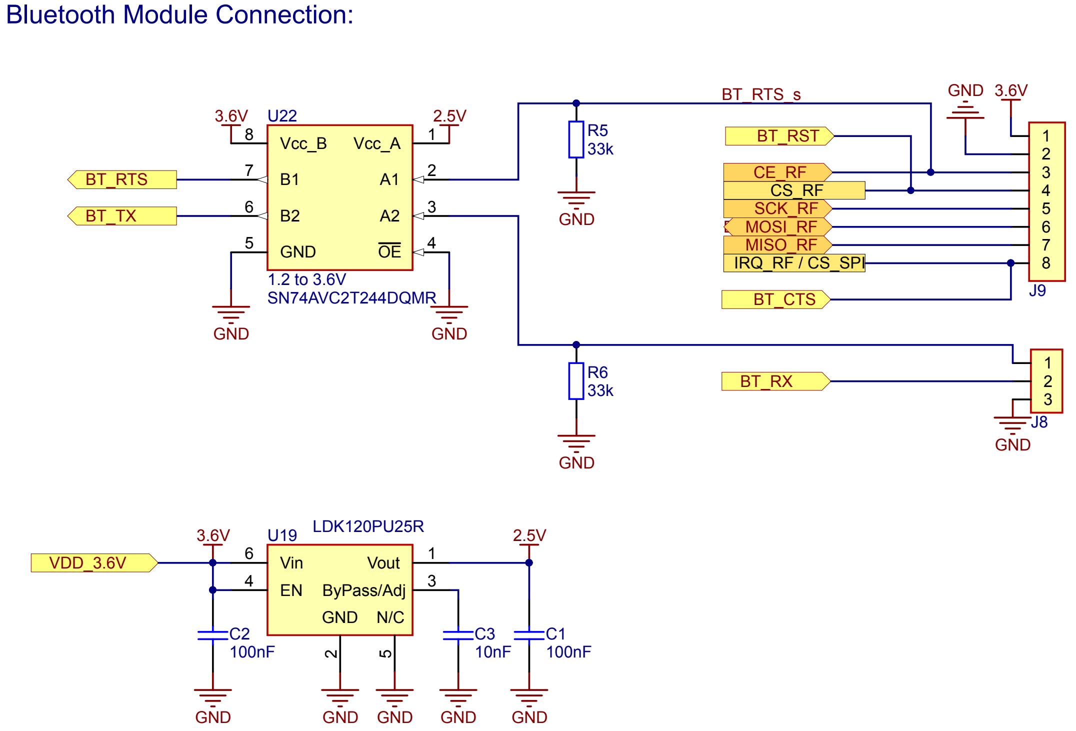
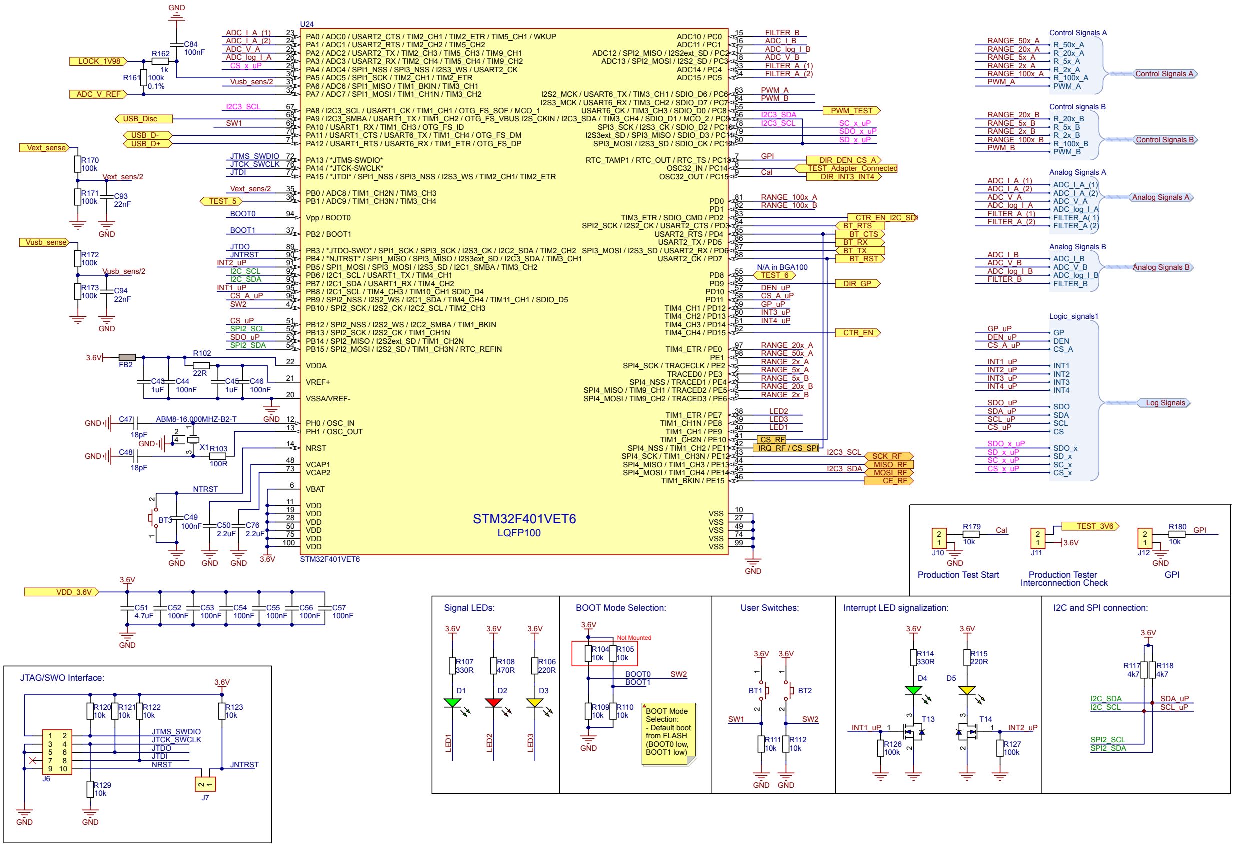


Figure 14. STEVAL-MKI109V3 circuit schematic (8 of 8)



Revision history

Table 10. **Document revision history**

Date	Version	Changes
05-Oct-2016	1	Initial release.
26-Feb-2018	2	Updated List of supported MEMS adapter boards, List of supported commands and Returned values for *start command
27-Jul-2018	3	Updated List of supported MEMS adapter boards, List of supported commands and Returned values for *start command Changed all descriptions for *Zon to "Forces High impedance state" (was Forces 3-state) and *Zoff to "Exits from High impedance state" (was Exits from 3-state) Minor text changes
23-Jan-2019	4	Updated Table 1. List of supported MEMS adapter boards and Table 3. Returned values for *start command. Added Section 4.8 Digital output temperature sensor: supported commands.
02-Jul-2019	5	Updated Table 1. List of supported MEMS adapter boards and Table 3. Returned values for *start command.

Contents

List of tables

List of figures