

# Hands-on Workshop: Introduction to Baremetal Embedded Applications

Larisa Radu – Software Engineer, Software Development

Monica Ignat – Applications Engineering Manager

# Who are we?

Founded **1965**

Headquarters **Norwood, MA**

Employees **~15,000**

Countries **20+**

Products **~45,000 SKUs**

Customers **125,000**

Publicly Listed **NASDAQ:ADI**  
Part of S&P 500 and NASDAQ 100

Design Centers **~45**

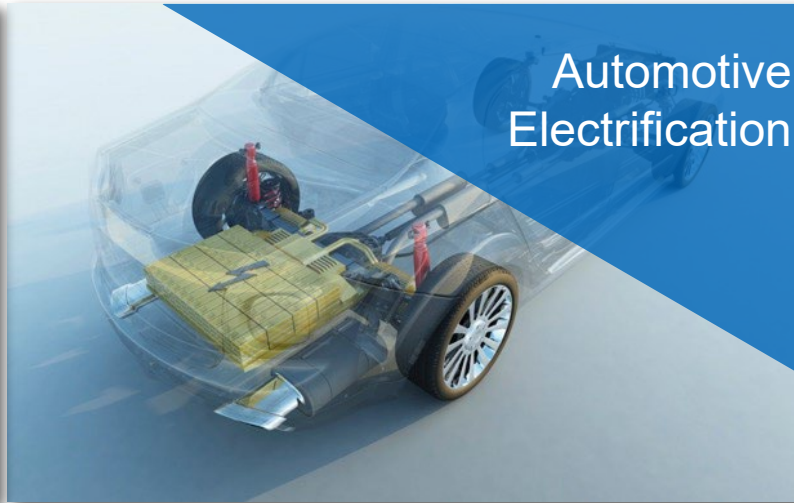
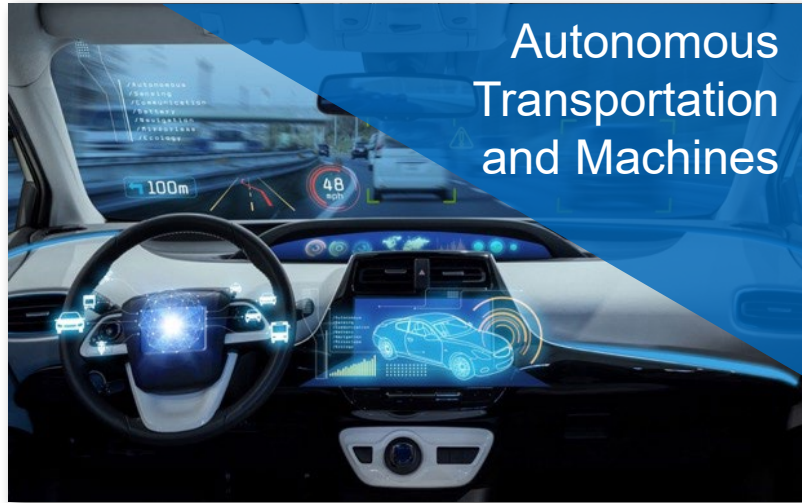
Global Manufacturing **U.S. (Massachusetts, California,  
Washington), Ireland, Philippines,  
and Malaysia**



**Over 4700 patents  
and \$4 billion R&D  
investment in the  
past 10 years.**



# Technologies for Today's and Tomorrow's Innovations





# ADI Romania Design Center

- ▶ Founded in 2011
- ▶ Office 1 - UBC Riviera
  - 1,000 square meters, 100 people capacity
- ▶ Office 2 - UBC Tower
  - 1,000 square meters, 120 people capacity
- ▶ Multidisciplinary team
  - Hardware design
  - FPGA development (VHDL, Verilog)
  - Embedded software (C/C++, Linux)
  - Applications software (Python, MATLAB, C++)
  - Devops (Jenkins, Microsoft Azure, CI/CD)
  - System architecture
  - UX design
  - Program/Project management
- ▶ Project fields
  - RF Communications
  - Precision & High-Speed Instrumentation
  - Depth, Perception and Ranging (ToF, LIDAR)
  - Industrial Automation



## ► Part 1

- Background – History and evolution of no-OS
- What is Baremetal (no-OS)
- What is a baremetal device driver
- No-OS API and platforms
- No-OS projects and examples
- IIO Concepts
- Takeaways

## ► Part 2

- Instructor-led demo
- Q&A session

# Hands-on Workshop: Introduction to Embedded Software Applications

## Part 1

- Background – History and evolution of baremetal
- What is No-OS
- No-OS Device Drivers
- No-OS API
- No-OS Platforms
- IIO Concepts



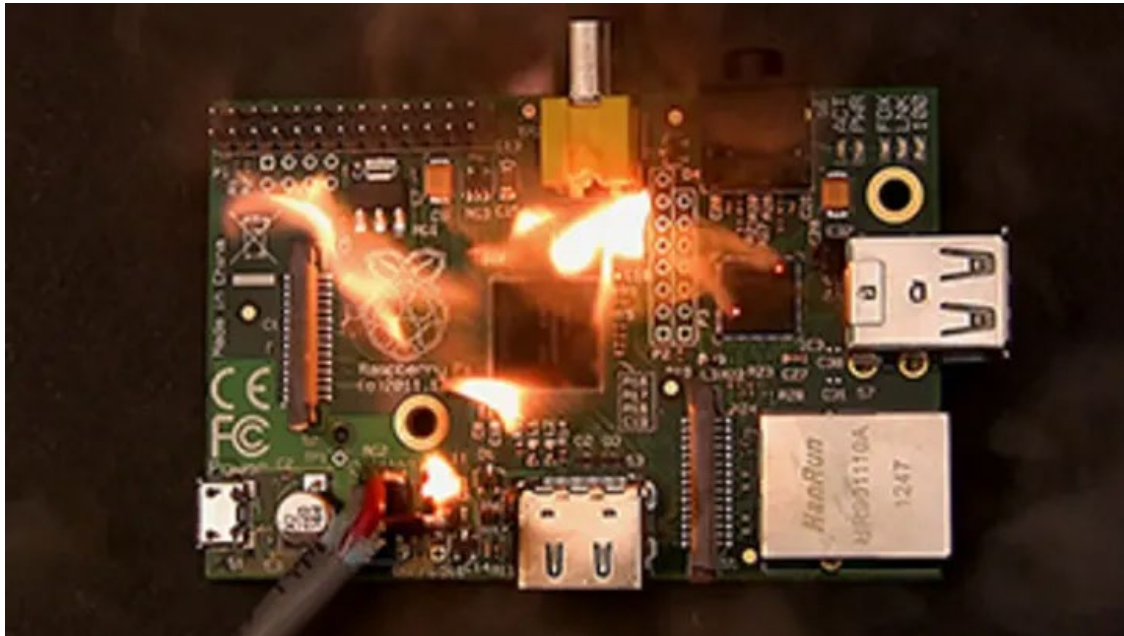
# Baremetal background

Traditionally, industry chip manufacturers would sell ICs without accompanying software:

- Chip
- Datasheet
- Pseudocode with an initialization sequence and/or sequence of data acquisition

Complex parts require complex software

ADI addressed this issue and started providing software for its parts as well.



<https://res.cloudinary.com/>



<https://www.pcbtrain.co.uk/>

# Baremetal background

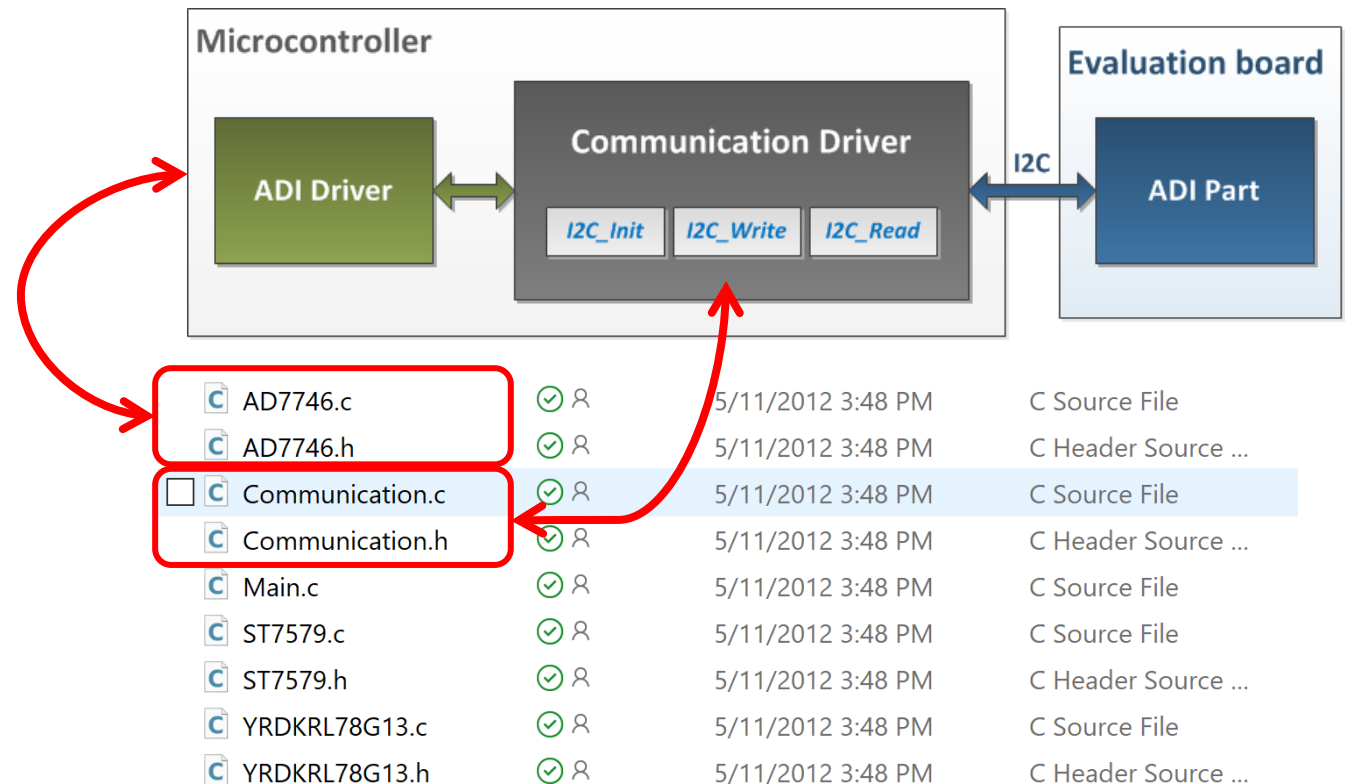
Baremetal 10 years ago looked like this:

- A .zip file containing the driver files
- A .zip file containing a project

<https://wiki.analog.com/resources/tools-software/uc-drivers/renesas/ad7746>

## Downloads

-  AD7746 Generic Driver
-  AD7746 RL78G13 Driver





## ADI part driver:

```
void AD7746_Write(unsigned char subAddr,
                 unsigned char* dataBuffer,
                 unsigned char bytesNumber)
{
    unsigned char sendBuffer[10] = {0, };
    unsigned char byte = 0;

    sendBuffer[0] = subAddr;
    for(byte = 1; byte <= bytesNumber; byte++)
    {
        sendBuffer[byte] = dataBuffer[byte - 1];
    }
    I2C_Write(AD7746_ADDRESS,
             sendBuffer,
             bytesNumber + 1,
             1);
}
```

## Communication template:

```
unsigned char I2C_Write(unsigned char slaveAddress,
                       unsigned char* dataBuffer,
                       unsigned char bytesNumber,
                       unsigned char stopBit)
{
    // Add your code here.
}
```

## Communication implementation for Renesas MCU:

```
unsigned char I2C_Write(unsigned char slaveAddress,
                       unsigned char* dataBuffer,
                       unsigned char bytesNumber,
                       unsigned char stopBit)
{
    unsigned char wait = 0;
    unsigned char byte = 0;

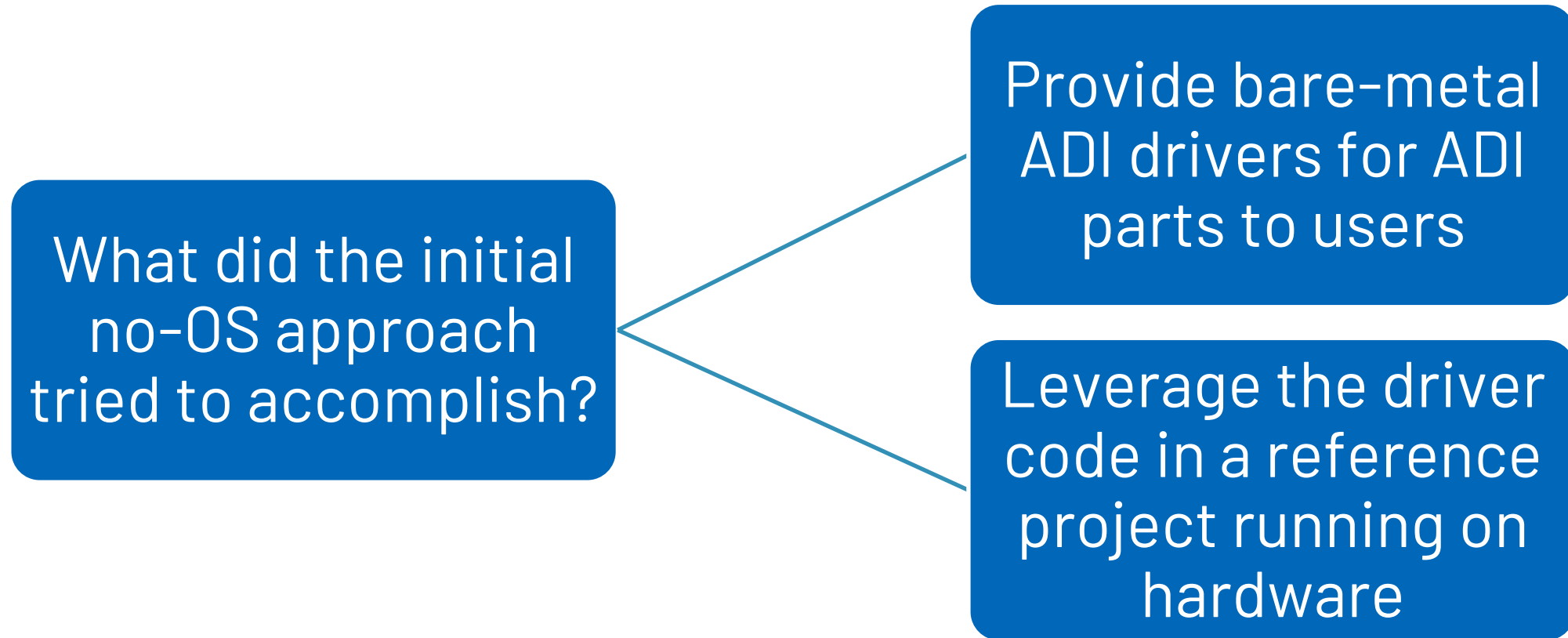
    ST0 |= 0x0002; // Stop the communication operation for manipulate the CKOm.
    SOE0 &= ~0x0002; // Disable serial output.
    SO0 |= 0x0200; // Set SCLK.
    SO0 |= 0x0002; // Set SDA.
    for(wait = 0; wait < 10; wait++)
    {
        NOP();
    }
    SO0 &= ~0x0002; // Clear SDA.
    ST0 &= ~0x0002; // Start the communication operation.
    SS0 |= 0x0002; // Enable the operation.

    SOE0 |= 0x0002; // Enable serial output.
    SCR01 &= ~0x0000;
    SCR01 |= 0x8000; // Set the operation mode to transmission only.
    IICIF01 = 0; // Clear the interrupt flag.
    IICMK01 = 0; // Interrupt servicing enabled.

    nineClocksFlag = 0;
    SIO01 = (slaveAddress << 1);
    while(nineClocksFlag == 0);
    for(byte = 0; byte < bytesNumber; byte++)
    {
        nineClocksFlag = 0;
        SIO01 = *dataBuffer;
        while(nineClocksFlag == 0);
        dataBuffer++;
    }

    if(stopBit)
    {
        ST0 |= 0x0002; // Stop the communication operation for manipulate the CKOm.
        SOE0 &= ~0x0002; // Disable serial output.
        SO0 &= ~0x0002; // Clear SDA.
        SO0 |= 0x0200; // Set SCLK.
        for(wait = 0; wait < 10; wait++)
        {
            NOP();
        }
        SO0 |= 0x0002; // Set SDA.
    }

    return bytesNumber;
}
```



## Advantages

Driver code was MCU independent

## Disadvantages

Customer responsibility to port reference project on a different MCU

.zip file distribution led to no version control and code duplication

## Evolution provided:

Provide a way for reference projects to run on multiple hardware combinations

Provide a build system that generates binaries and run them on hardware

Expose parts as IIO devices to PC applications

Improve code quality



# What is no-OS?

Baremetal(No-OS) = system that does not run on a specific operating system

## No-OS

- Software framework for embedded bare-metal development
- Open-source
- ADI-BSD license
- Free
- Large collection of platform agnostic device drivers for ADI parts
- Significant collection of reference projects leveraging ADI evaluation boards
- Reference projects can run on a wide range of hardware
- Provides IIO enabled devices, making them accessible to PC applications that use libiio

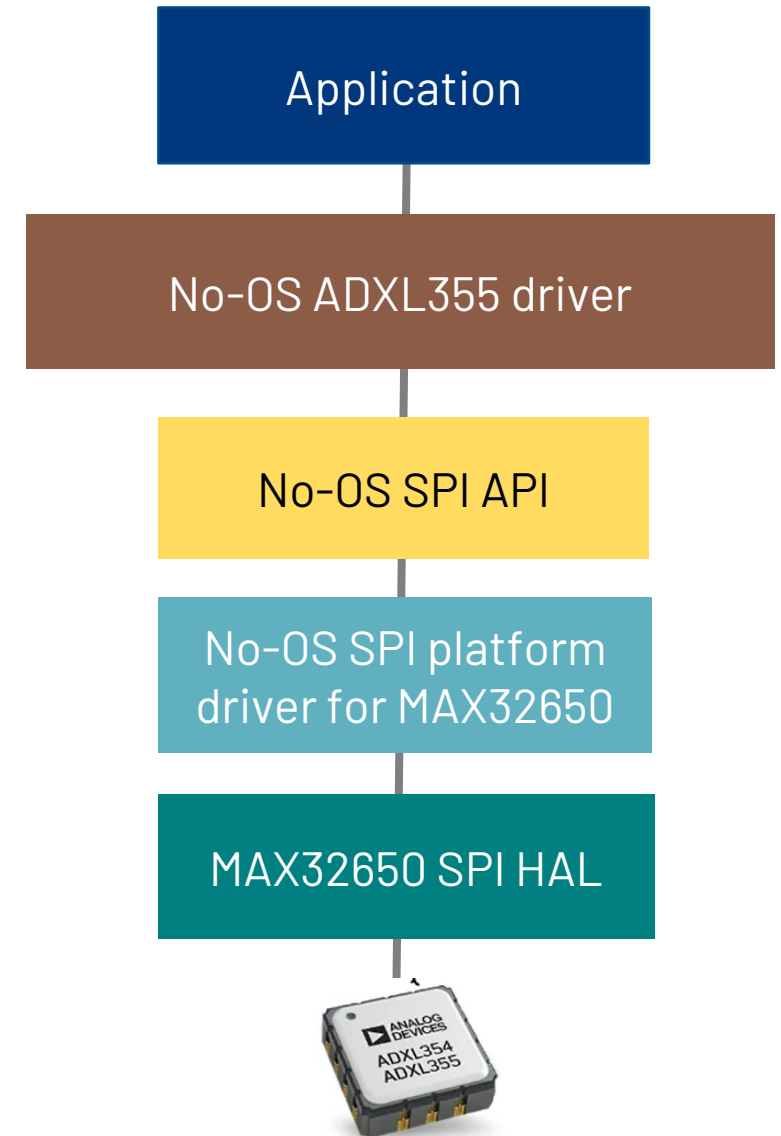


### Resources

<https://wiki.analog.com/resources/no-os>  
<https://github.com/analogdevicesinc/no-OS>

## What is a No-OS device driver?


- Implemented in C, in a .c and .h pair, stored under /drivers
- Its programming interface is directly called by the application code
- Defines its own descriptor structures and init\_param
- Contains minimum init() and remove() functions
- Performs no-OS API calls, does not perform platform specific function calls, it's platform agnostic
- Software application can access hardware functionality without knowing in detail how the device operates



- *Implemented in C as a .c and .h pair, stored under drivers/*

🔑 master 🔑 152 branches 🔖 5 tags

Go to file Add file <> Code

 danmois and CiprianRegus projects:adrv9009:src:app\_transceiver.c ADX... 8f20091 12 days ago 4,384 commits

ci	doc: move files under doxygen subfolder	3 months ago
doc/doxygen	doc: move files under doxygen subfolder	3 months ago
drivers	platform: maxim: max32665: Change UART pins	20 days ago
iio	Fixed bug in storing context attrs into xml string	26 days ago
include	Added basic APIs for EEPROM interface	last month

- *Consists of init() and remove() functions at minimum*

```
322  /*! Init. the comm. peripheral and checks if the ADXL355 part is present. */
323  int adxl355_init(struct adxl355_dev **device,
324                  struct adxl355_init_param init_param);
325
326  /*! Free the resources allocated by adxl355_init(). */
327  int adxl355_remove(struct adxl355_dev *dev);
```



- Its programming interface is directly called by the application code*

```
58  int dummy_example_main()
59  {
60      struct adxl355_dev *adxl355_desc;
61      int ret;
62
63      ret = adxl355_init(&adxl355_desc, adxl355_ip);
64      if (ret)
65          goto error;
66      ret = adxl355_soft_reset(adxl355_desc);
67      if (ret)
68          goto error;
69      ret = adxl355_set_odr_lpf(adxl355_desc, ADXL355_ODR_3_906HZ);
70      if (ret)
71          goto error;
72      ret = adxl355_set_op_mode(adxl355_desc, ADXL355_MEAS_TEMP_ON_DRDY_OFF);
73      if (ret)
74          goto error;
```

- *Defines its own init\_param and descriptor structures*

```
231 struct adxl355_init_param {
232     /** Device Communication initialization structure: either SPI or I2C */
233     union adxl355_comm_init_param comm_init;
234     /** Device Communication type: ADXL355_SPI_COMM, ADXL355_I2C_COMM */
235     enum adxl355_comm_type comm_type;
236     /** Device type: ADXL355 or 359 */
237     enum adxl355_type dev_type;
238 };

298 struct adxl355_dev {
299     /** Device type */
300     enum adxl355_type dev_type;
301     /** Device communication descriptor */
302     union adxl355_comm_desc com_desc;
303     /** Device Communication type: ADXL355_SPI_COMM, ADXL355_I2C_COMM */
304     enum adxl355_comm_type comm_type;
305     enum adxl355_op_mode op_mode;
306     enum adxl355_odr_lpf odr_lpf;
307     enum adxl355_hpf_corner hpf_corner;
308     enum adxl355_range range;
309     uint16_t x_offset;
310     uint16_t y_offset;
311     uint16_t z_offset;
312     uint8_t fifo_samples;
313     union adxl355_act_en_flags act_en;
314     uint8_t act_cnt;
315     uint16_t act_thr;
316     uint8_t comm_buff[289];
317 };
```

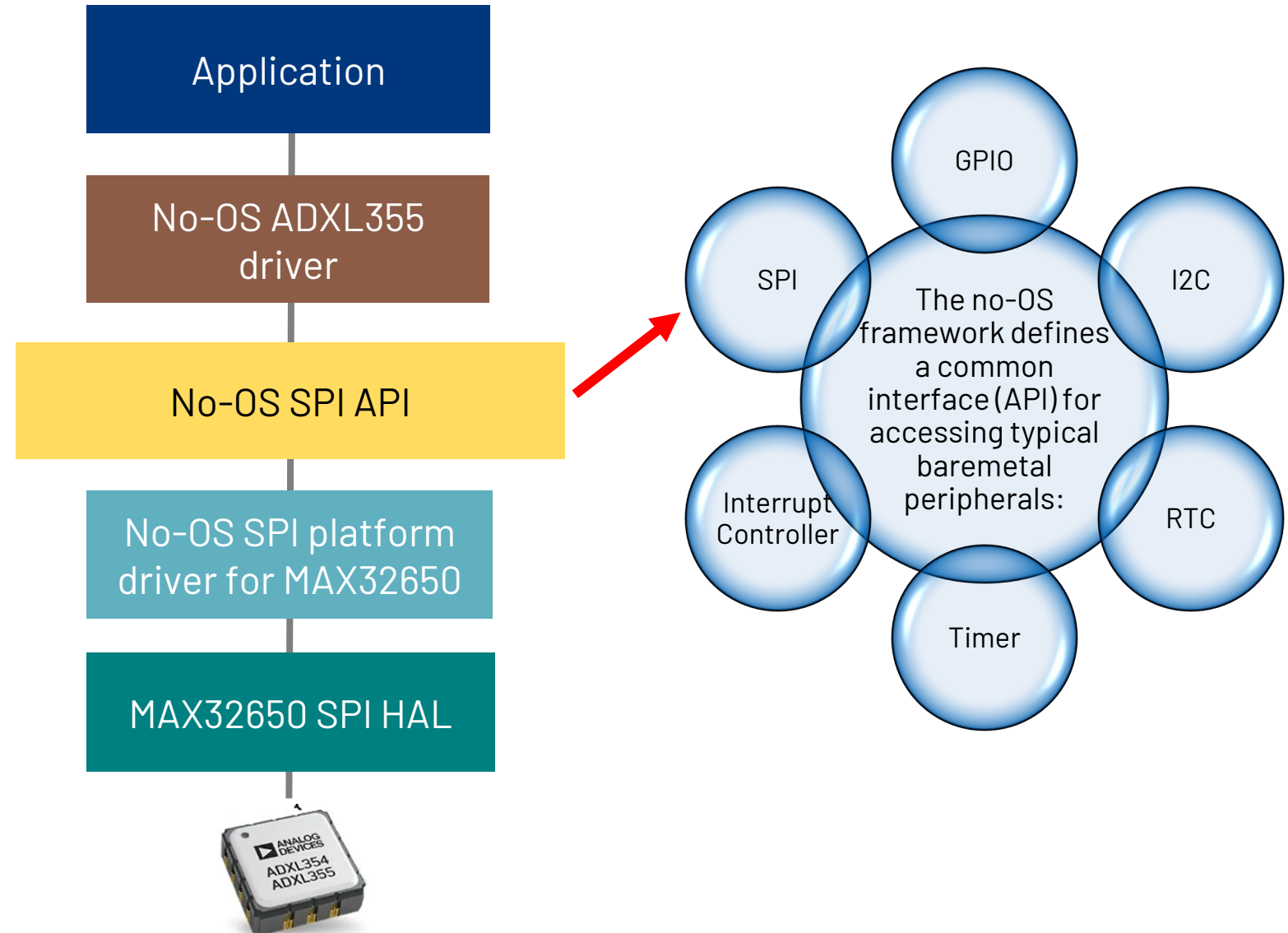
- *Performs no-OS API calls, does not perform platform specific function calls, therefore it is platform agnostic*

```
110  int adxl355_write_device_data(struct adxl355_dev *dev, uint8_t base_address,
111                                uint16_t size, uint8_t *write_data)
112  {
113      int ret;
114
115      for (uint16_t idx = 0; idx < size; idx++)
116          dev->comm_buff[1+idx] = write_data[idx];
117
118      if (dev->comm_type == ADXL355_SPI_COMM) {
119          dev->comm_buff[0] = ADXL355_SPI_WRITE | (base_address << 1);
120          ret = no_os_spi_write_and_read(dev->com_desc.spi_desc, dev->comm_buff,
121                                          size + 1);
122      } else {
123          dev->comm_buff[0] = base_address;
124          ret = no_os_i2c_write(dev->com_desc.i2c_desc, dev->comm_buff, size + 1, 1);
125      }
```



## API – Application Programming Interface

- No-OS API allows any code that uses it (driver or app) to remain platform agnostic



# No-OS Platforms

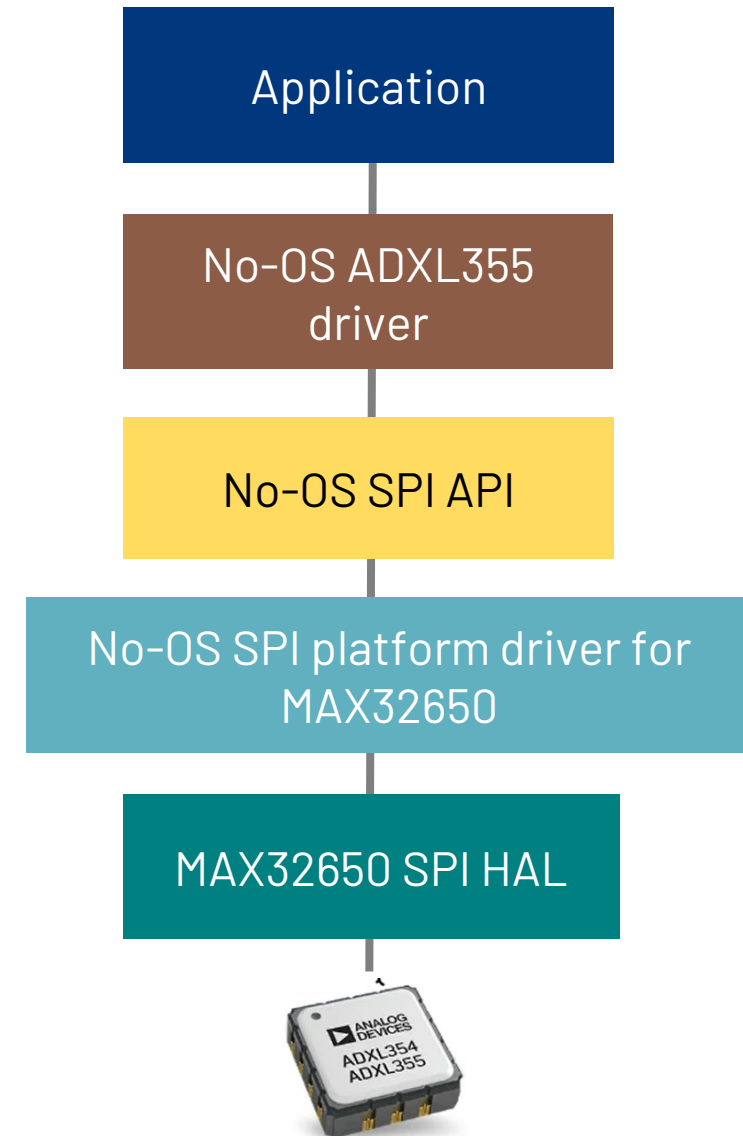
Platform drivers – implementations of peripheral related no-OS API on a specific platform

Platform drivers use vendor HAL - Hardware Abstraction Layer

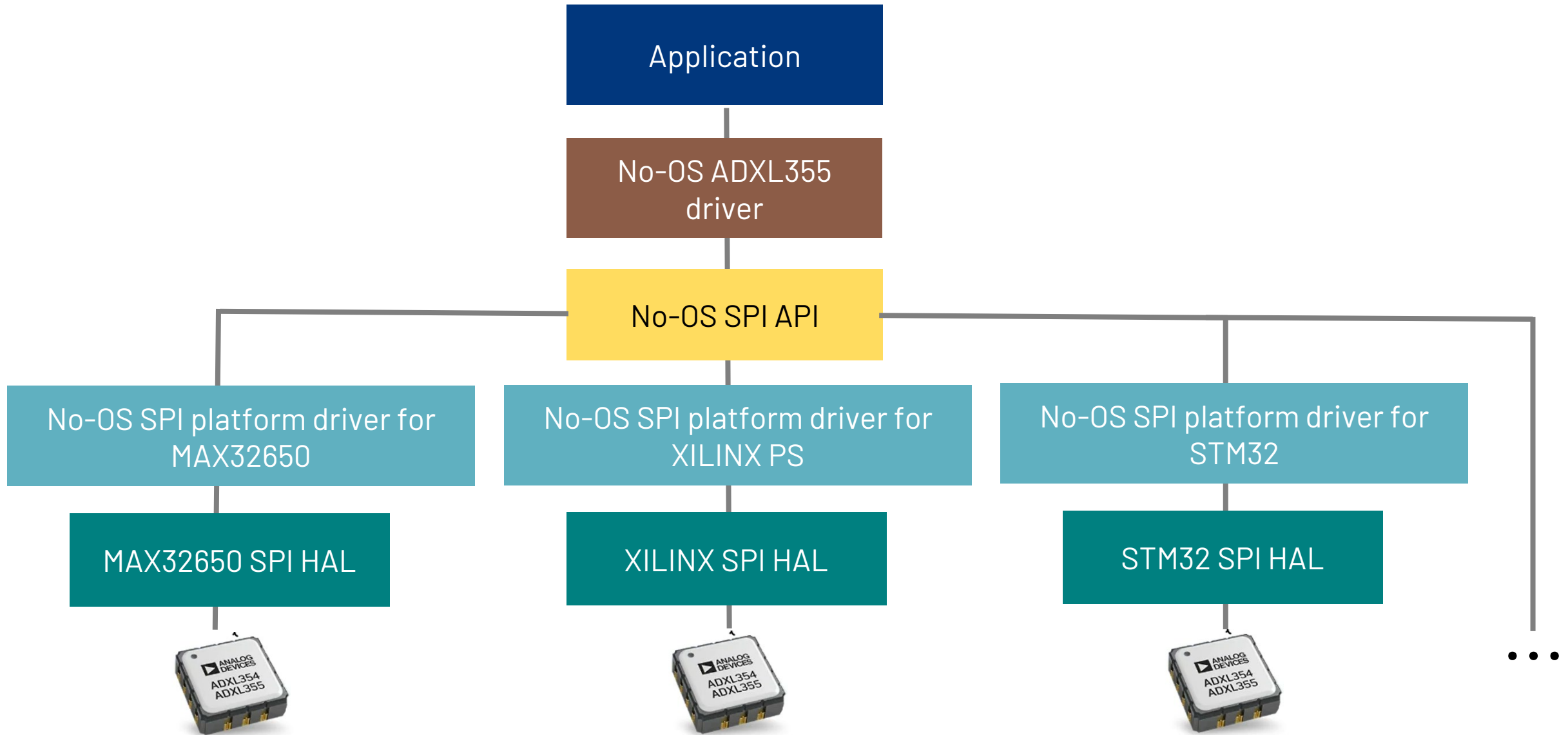
No-OS platform drivers are implementations of peripheral no-OS API on a particular platform.

No-OS modularity allow it to run a lot of its code on different platforms like:

- Xilinx (Zynq7000, ZynqMP, Microblaze)
- Maxim (32650, 32655, 32660, 32670, 78000) ADuCM (3029)
- STM32 (almost any)
- RaspberryPi Pico
- Mbed



# No-OS Platforms





## An application can be built, run and debugged on hardware

- Located under projects/
- It has a main() function
- Uses drivers/ and drivers/platforms directories
- Uses no-OS API
- Uses various libraries
- User interaction – serial, iio-oscilloscope
- makefiles

## Project hardware typically is made of

- An evaluation board
- A carrier board

## No-OS projects are used for

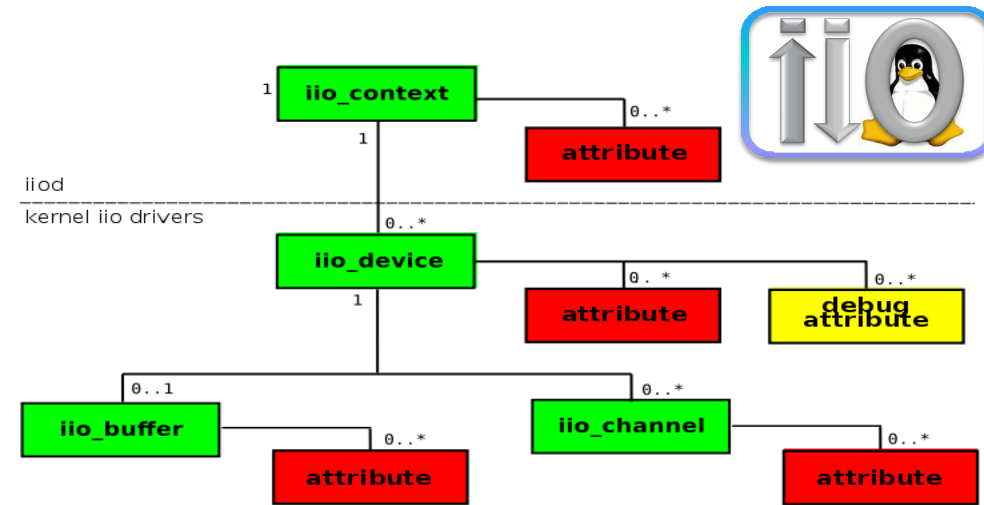
- ADI parts evaluation
- Starting development based on a no-OS project

- ▶ The Linux **Industrial I/O (IIO)** subsystem is intended to provide support for devices that, in some sense, are analog-to-digital or digital-to-analog converters
  - Devices that fall into this category are:
    - ADCs
    - DACs
    - Accelerometers, gyros, IMUs
    - Capacitance-to-Digital converters (CDCs)
    - Pressure, temperature, and light sensors, etc.
    - RF Transceivers (like the AD9361 / AD9364 / AD9371 / ADRV9009)
  - Can be used on ADCs ranging from a 1MSPS SoC ADC to >5 GSPS ADCs

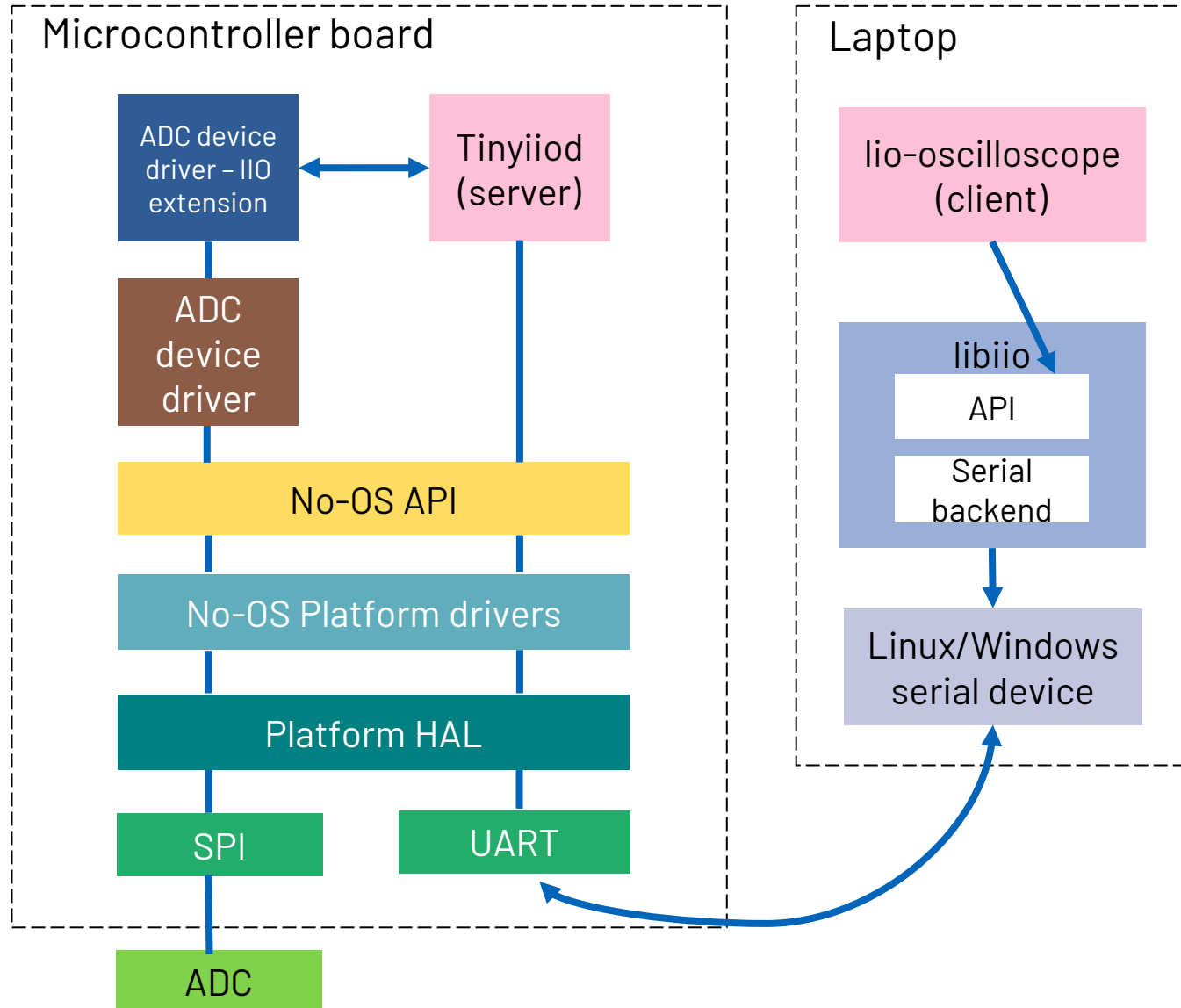
```
root:/> cd /sys/bus/iio/devices/
root:/sys/bus/iio/devices> ls
iio:device0

root:/sys/bus/iio/devices> cd iio:device0

root:/sys/devices/platform/i2c-bfin-twi.0/i2c-0/0-002a/iio:device0> ls -l
-r--r--r-- 1 root root 4096 Jan 1 00:38 dev
drwxr-xr-x 2 root root 0 Jan 1 00:38 events
-rw-r--r-- 1 root root 4096 Jan 1 00:38 in_temp0_mean_raw
-r--r--r-- 1 root root 4096 Jan 1 00:38 in_temp0_raw
-rw-r--r-- 1 root root 4096 Jan 1 00:38 in_temp0_scale
-r--r--r-- 1 root root 4096 Jan 1 00:38 in_voltage0_raw
-r--r--r-- 1 root root 4096 Jan 1 00:38 in_voltage1_raw
-rw-r--r-- 1 root root 4096 Jan 1 00:38 in_voltage2_raw
-r--r--r-- 1 root root 4096 Jan 1 00:38 in_voltage3_raw
-rw-r--r-- 1 root root 4096 Jan 1 00:38 in_voltage4_raw
-r--r--r-- 1 root root 4096 Jan 1 00:38 in_voltage5_raw
-rw-r--r-- 1 root root 4096 Jan 1 00:38 in_voltage6_raw
-r--r--r-- 1 root root 4096 Jan 1 00:38 in_voltage7_raw
-rw-r--r-- 1 root root 4096 Jan 1 00:38 in_voltage_scale
-r--r--r-- 1 root root 4096 Jan 1 00:38 name
drwxr-xr-x 2 root root 0 Jan 1 00:38 power
--w----- 1 root root 4096 Jan 1 00:38 reset
lrwxrwxrwx 1 root root 0 Jan 1 00:38 subsystem -> ../../../../bus/iio
-rw-r--r-- 1 root root 4096 Jan 1 00:38 uevent
```

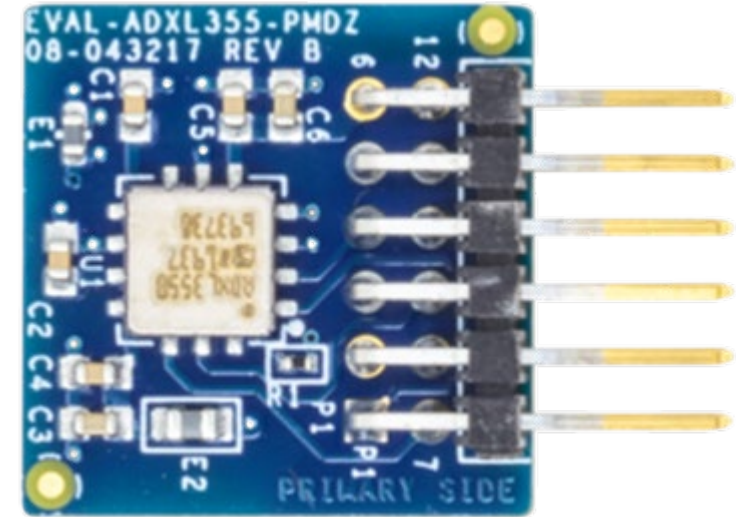


# libIIO and applications



## Low Noise, Low Drift, Low Power, 3-Axis MEMS Accelerometers

- ▶ ADXL355 digital output features
  - Digital SPI and I<sup>2</sup>C interfaces supported
  - 20-bit ADC
  - Data interpolation routine for synchronous sampling
  - Programmable high- and low-pass digital filters
- ▶ 0 g offset vs. temperature (all axes): 0.15 mg/°C maximum
- ▶  $V_{\text{SUPPLY}}$  with internal regulators: 2.25 V to 3.6





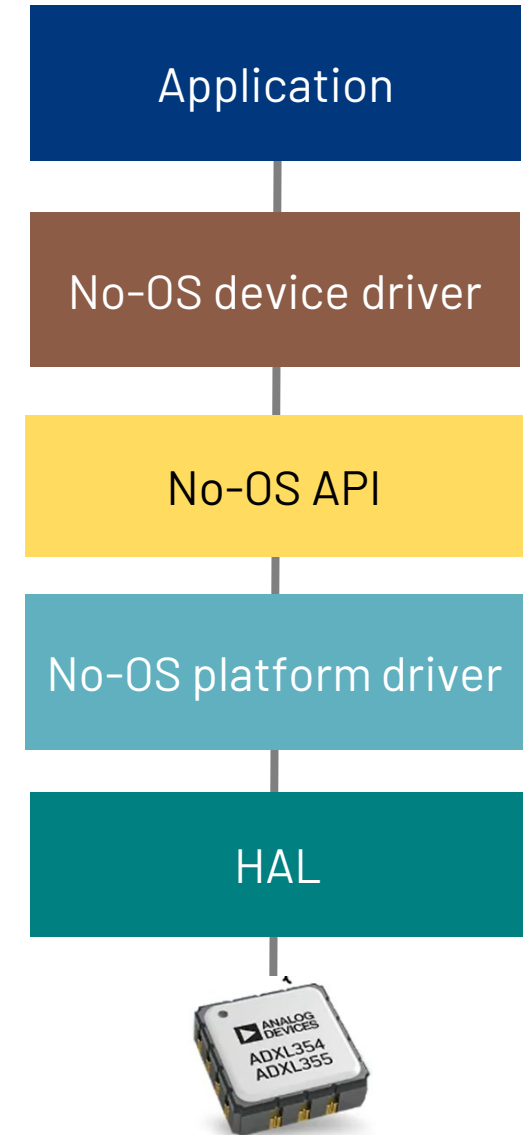
# No-OS Device Drivers – takeaways

No-OS device drivers are bits of platform agnostic code

Device drivers can be used to control ADI parts purely from software

No-OS platform drivers are implementations of peripheral no-OS API on a particular platform

No-OS has a build system for projects to be built and run on hardware



# Hands-on Workshop: Introduction to no-OS Embedded Applications

## Part 2

Hands on Lab

## By the end of this lab, you will learn:

- How to build a no-OS project
- How to run a no-OS application on a specific hardware
- How to use a no-OS demo running on a specific hardware
- How to modify written code to do a certain task



<http://www.iconarchive.com/show/noto-emoji-objects-icons-by-google/62807-radio-icon.html>

<http://www.streamlineicons.com>

<http://pixelkit.com>

# Hands On Lab – Getting started steps

Insert the provided USB stick into an USB port of your computer.

Turn off your computer, then turn it on and at the same time keep pressing on the F12 key 2-3 times a second to enter BIOS.

Select the USB option when asked what device to boot from.

Select the first option (Debian GNU/Linux Live) and wait for Linux to boot.

Click *Activities* (upper-left corner), search for *Terminal* and open it.

Before building a project, set the MAXIM\_LIBRARIES, PLATFORM and TARGET environment variables with the following command:

```
export MAXIM_LIBRARIES=~/.workshop/tools/MAX78000_SDK/Libraries; export PLATFORM=maxim;  
export TARGET=max78000
```



## Equipment used:

- ▶ EVAL-ADXL355-PMDZ
- ▶ MAX78000FTHR
- ▶ USB cable

### Demo 1

Prints a “Hello World” message over the serial UART.

### Demo 2

Reads temperature from the internal temperature sensor of the ADXL355 device.

### Demo 3

Reads temperature and the acceleration values from the ADXL355 and converts the data from raw values into user readable values.

### Demo 4

Accelerometer-enabled game that lets you place components on a circuit by physically tilting the accelerometer.



<http://www.iconarchive.com/show/noto-emoji-objects-icons-by-google/62807-radio-icon.html>

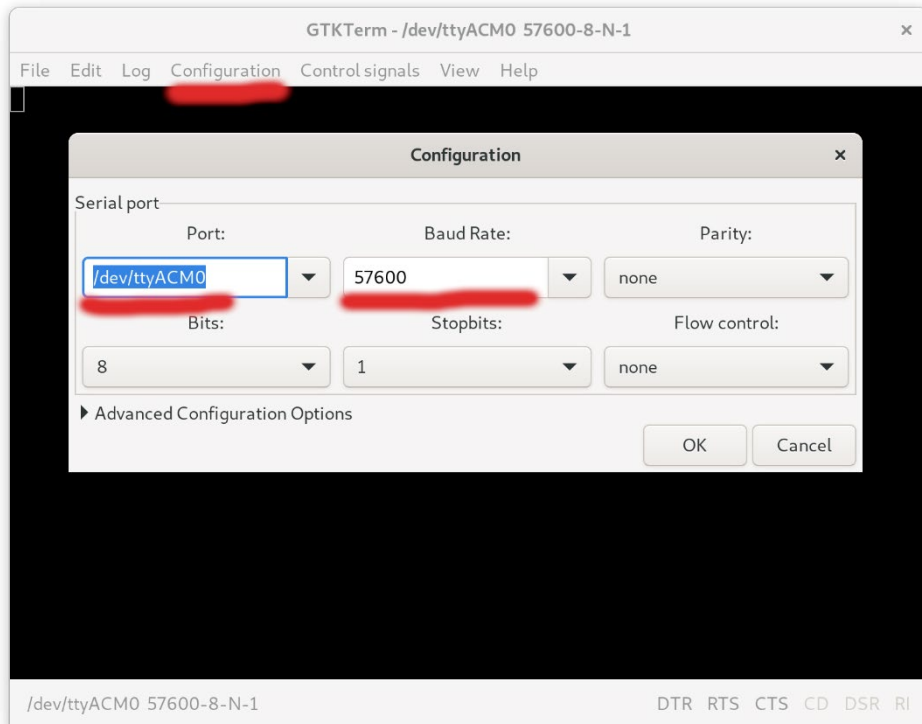
<http://www.streamlineicons.com>

<http://pixelkit.com>

## Demo 1

Prints a “Hello World” message over the serial UART.

Setup



## Procedure:

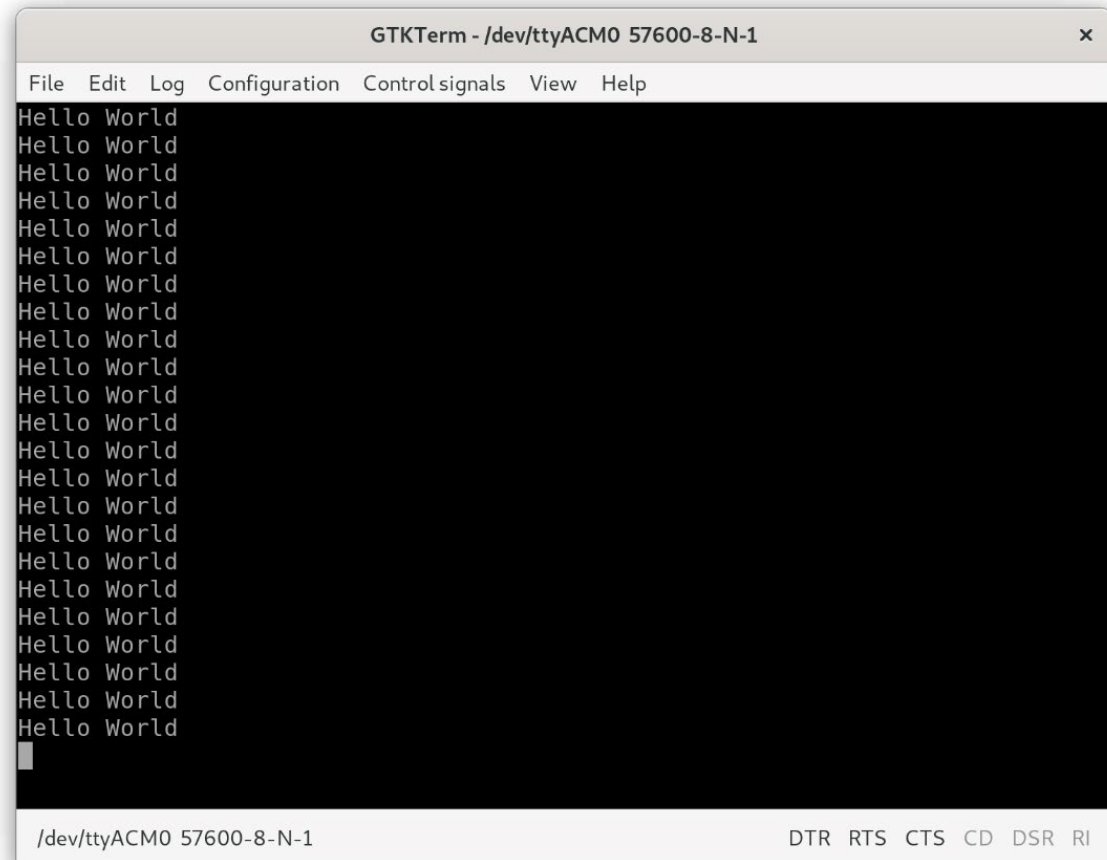
- **Move to the no-OS workshop project location:**
  - `$ cd ~/workshop/no-OS/projects/workshop`
- **Reset the workspace:**
  - `$ make reset`
- **Build the first project example:**
  - `$ make EXAMPLE_NUMBER=1`
- **Connect the MAX78000FTHR to the PC using a USB cable.**
- Configure GTKTerm application as shown in the left
- **From the terminal, write the following command to program MAX78000:**
  - `$ make EXAMPLE_NUMBER=1 run`

## Demo 1

Prints a “Hello World” message over the serial UART.

### Procedure:

- You will see the “Hello World” message running on the GTK terminal every second



The screenshot shows a GTKTerm window titled "GTKTerm - /dev/ttyACM0 57600-8-N-1". The window has a menu bar with "File", "Edit", "Log", "Configuration", "Controlsignals", "View", and "Help". The main area is a black terminal with white text displaying "Hello World" repeated 20 times. At the bottom, the status bar shows "/dev/ttyACM0 57600-8-N-1" on the left and "DTR RTS CTS CD DSR RI" on the right.

## Demo 2

Reads temperature from the internal temperature sensor of the ADXL355 device.

EVAL-ADXL355-PMDZ pinout:

Pin Number	Pin Function	Mnemonic
Pin 1	Chip Select	CS
Pin 2	Master Out Slave In	MOSI
Pin 3	Master In Slave Out	MISO
Pin 4	Serial Clock	SCLK
Pin 5	Digital Ground	DGND
Pin 6	Digital Power	VDD
Pin 7	Interrupt 1	INT1
Pin 8	Not Connected	NC
Pin 9	Interrupt 2	INT2
Pin 10	Data Ready	DRDY
Pin 11	Digital Ground	DGND
Pin 12	Digital Power	VDD

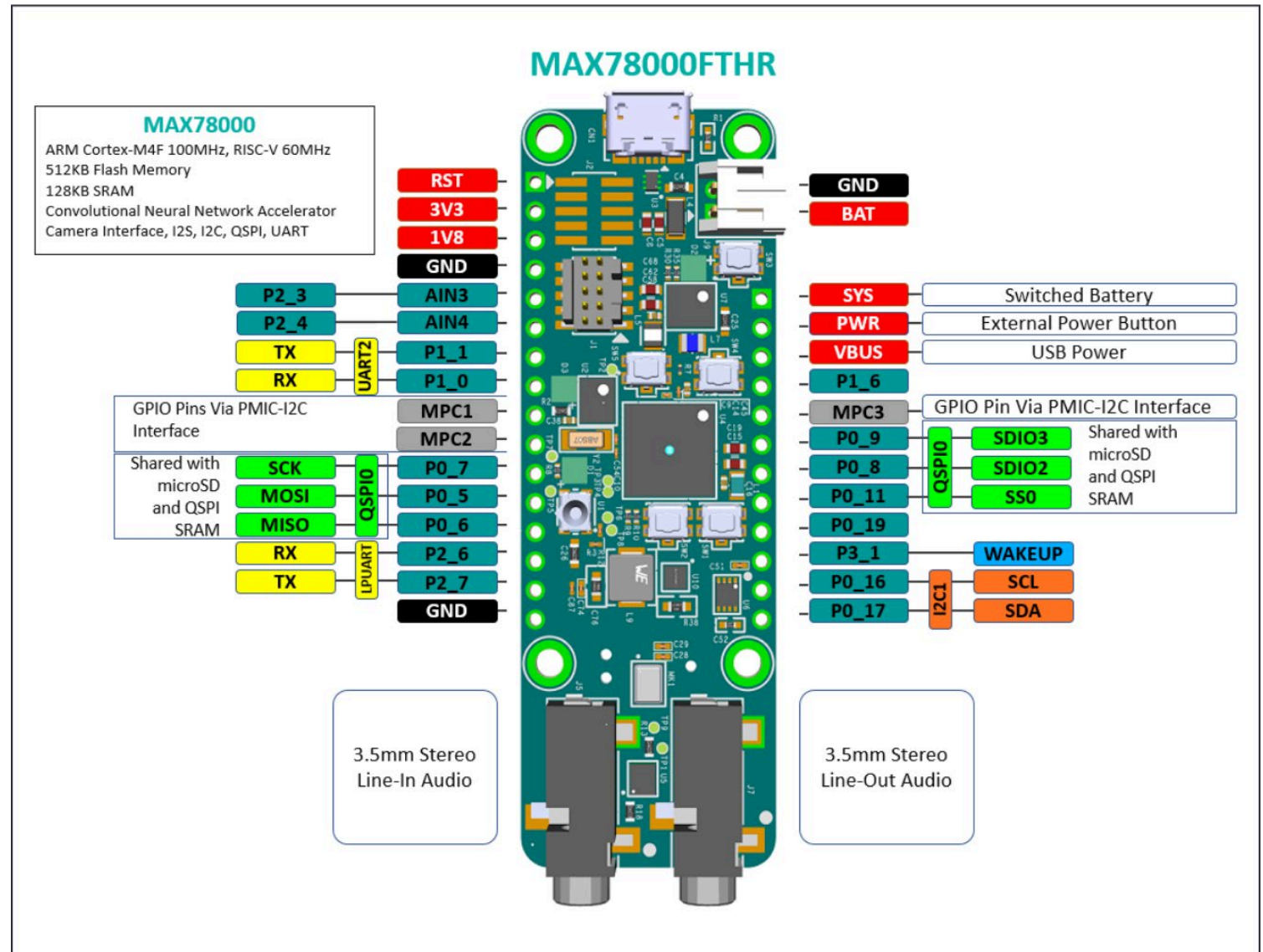
## Procedure:

- **Disconnect the MAX78000FTHR USB cable from the PC.**
- **Connect the EVAL-ADXL355-PMDZ to the MAX78000FTHR by using the information below:**
  - EVAL-ADXL355-PMDZ Pinout
  - MAX78000FTHR Pinout
  - Pin correspondence table

## Demo 2

Reads temperature from the internal temperature sensor of the ADXL355 device.

MAX78000FTHR Pinout:





## Demo 2

Reads temperature from the internal temperature sensor of the ADXL355 device.

Pin correspondence table:

MAX78000FTHR	Signal	EVAL-ADXL355-PMDZ
3V3	Digital power	6 or 12
GND	Digital ground	5 or 11
P0_11/SS0	SPI Chip Select	1
P0_5/MOSI	SPI Master Out Slave In	2
P0_6/MISO	SPI Master In Slave Out	3
P0_7/SCK	SPI Serial Clock	4

## Procedure:

- **Disconnect the MAX78000FTHR USB cable from the PC.**
- **Connect the EVAL-ADXL355-PMDZ to the MAX78000FTHR by using the information below:**
  - EVAL-ADXL355-PMDZ Pinout
  - MAX78000FTHR Pinout
  - Pin correspondence table

## Demo 2

Reads temperature from the internal temperature sensor of the ADXL355 device

### Procedure:

- Plug in the MAX78000FTHR into the PC using the USB cable.
- **Go to the no-OS workshop project location and reset the workspace:**
  - `$ cd ~/workshop/no-OS/projects/workshop`
  - `$ make reset`
- **Build the example:**
  - `$ make EXAMPLE_NUMBER=2`
- **Open GTKTerm and correctly configure it and then load the example onto the board:**
  - `$ make EXAMPLE_NUMBER=2 run`

Reads temperature from the internal temperature sensor of the ADXL355 device

```
GTKTerm - /dev/ttyACM0 57600-8-N-1 x
File Edit Log Configuration Controlsignals View Help
Current temperature is 24447.051390750 millidegrees Celsius
Current temperature is 24337.001666950 millidegrees Celsius
Current temperature is 24447.051390750 millidegrees Celsius
Current temperature is 24337.001666950 millidegrees Celsius
Current temperature is 24447.051390750 millidegrees Celsius
Current temperature is 24668.050838350 millidegrees Celsius
Current temperature is 24447.051390750 millidegrees Celsius
Current temperature is 24337.001666950 millidegrees Celsius
Current temperature is 24337.001666950 millidegrees Celsius
Current temperature is 24447.051390750 millidegrees Celsius
Current temperature is 24447.051390750 millidegrees Celsius
Current temperature is 24337.001666950 millidegrees Celsius
Current temperature is 24447.051390750 millidegrees Celsius
Current temperature is 24226.051943150 millidegrees Celsius
Current temperature is 24558.001114550 millidegrees Celsius
Current temperature is 24447.051390750 millidegrees Celsius
Current temperature is 24226.051943150 millidegrees Celsius
Current temperature is 24447.051390750 millidegrees Celsius
Current temperature is 24337.001666950 millidegrees Celsius
Current temperature is 24447.051390750 millidegrees Celsius
Current temperature is 24447.051390750 millidegrees Celsius
Current temperature is 24447.051390750 millidegrees Celsius
Current temperature is 24337.001666950 millidegrees Celsius
Current temperature is 24447.051390750 millidegrees Celsius
/dev/ttyACM0 57600-8-N-1 DTR RTS CTS CD DSR RI
```

## Demo 2

Reads temperature from the internal temperature sensor of the ADXL355 device

## Challenge:

- Change the current format of the printed temperature from millidegrees to degrees.
- **Example:**
  - the current format: *27545.032056750 millidegrees*
  - the new format: *27.54 degrees*

## Demo 3

Read temperature from the ADXL355 internal temperature sensor and convert it from raw values into user readable values.

### Procedure:

- **Change the working directory and reset the workspace:**
  - `$ cd ~/workshop/no-OS/projects/eval-adxl355-pmdz`
  - `$ make reset`
- **Build the WORKSHOP\_EXAMPLE of this project:**
  - `$ make IIO_EXAMPLE=n WORKSHOP_EXAMPLE=y`
- **Open GTKTerm and load the example onto the board:**
  - `$ make IIO_EXAMPLE=n WORKSHOP_EXAMPLE=y run`



## Demo 3

Read temperature from the ADXL355 internal temperature sensor and convert it from raw values into user readable values.

### Challenge:

- **Challenge:** compute the temperature and the accelerations from the raw values.
- **Hint:** For temperature you need to compute the *temp\_dividend* and *temp\_divisor*.
- For accelerometer values you need to compute the *x\_dividend*, *y\_dividend*, *z\_dividend* and *accel\_divisor*.

The formula for the temperature:

$$TEMPERATURE = (RAW + OFFSET) \cdot SCALE$$

$$TEMPERATURE = \left( RAW + \frac{OFFSET}{OFFSET\_DIV} \right) \cdot \frac{SCALE\_FACTOR}{SCALE\_FACTOR\_DIV}$$

$$TEMPERATURE = \frac{(RAW \cdot OFFSET\_DIV + OFFSET) \cdot SCALE\_FACTOR}{OFFSET\_DIV \cdot SCALE\_FACTOR\_DIV}$$

The formula for the acceleration:

$$ACCELERATION = RAW \cdot SCALE$$

$$ACCELERATION = \frac{RAW \cdot SCALE\_FACTOR\_MUL}{SCALE\_FACTOR\_DIV}$$

## Demo 3

Read temperature from the ADXL355 internal temperature sensor and convert it from raw values into user readable values.

Parameter correspondence table:

PARAMETER	VALUE
TEMPERATURE OFFSET	- 2111.25
TEMPERATURE SCALE	- 110.497238
ACCELERATION SCALE	0.00003824593

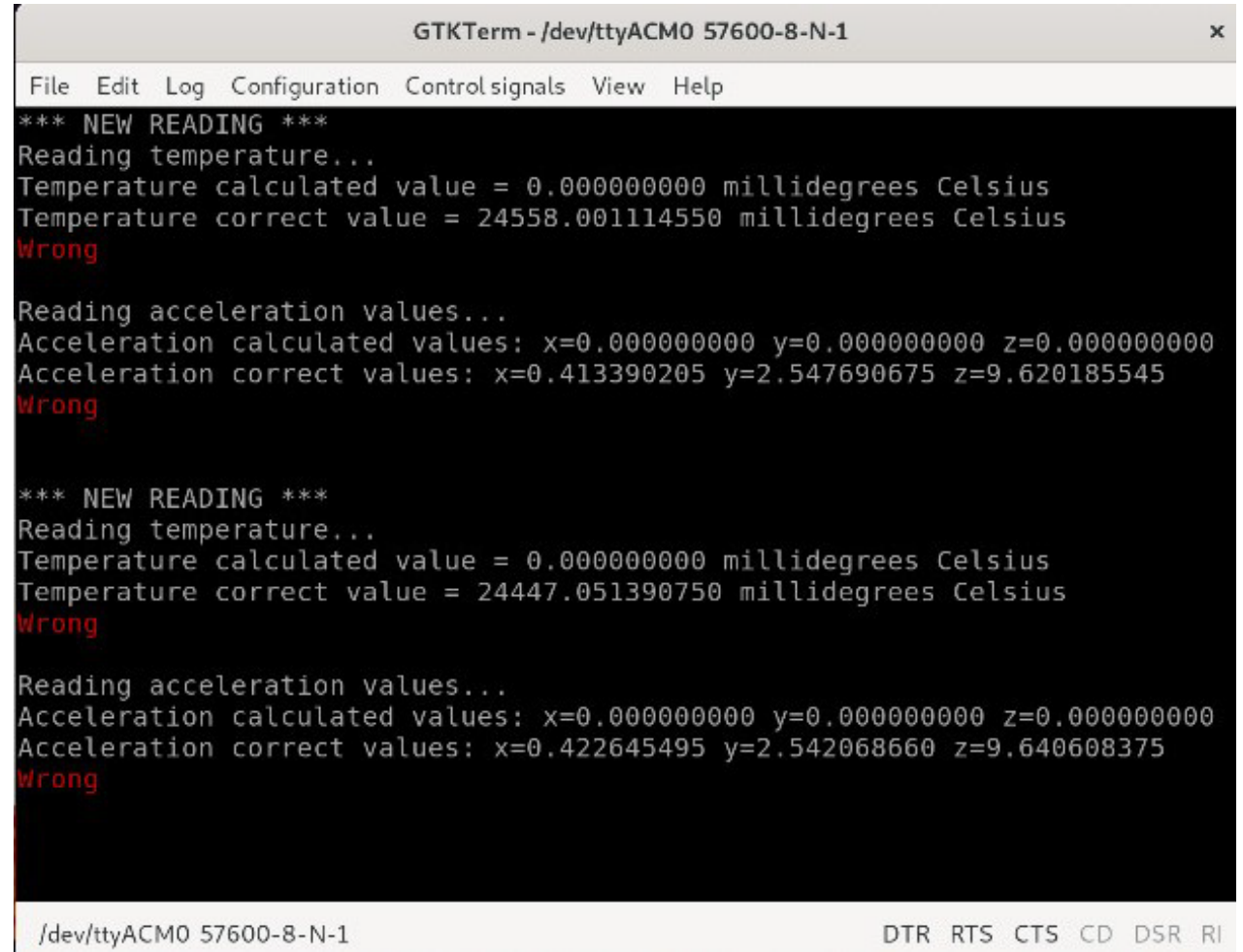
## Challenge:

Macro correspondence table:

MACRO	VALUE
ADXL355_TEMP_OFFSET	- 211125
ADXL355_TEMP_OFFSET_DIV	100
ADXL355_TEMP_SCALE_FACTOR	-110497238
ADXL355_TEMP_SCALE_FACTOR_DIV	1000000
ADXL355_ACC_SCALE_FACTOR_MUL	38245
ADXL355_ACC_SCALE_FACTOR_DIV	1000000000

## Demo 3

Read temperature and acceleration from the ADXL355 internal temperature sensor and convert it from raw values into user readable values.



The screenshot shows a terminal window titled "GTKTerm - /dev/ttyACM0 57600-8-N-1". The window contains the following text:

```
File Edit Log Configuration Controls signals View Help
*** NEW READING ***
Reading temperature...
Temperature calculated value = 0.000000000 millidegrees Celsius
Temperature correct value = 24558.001114550 millidegrees Celsius
Wrong

Reading acceleration values...
Acceleration calculated values: x=0.000000000 y=0.000000000 z=0.000000000
Acceleration correct values: x=0.413390205 y=2.547690675 z=9.620185545
Wrong

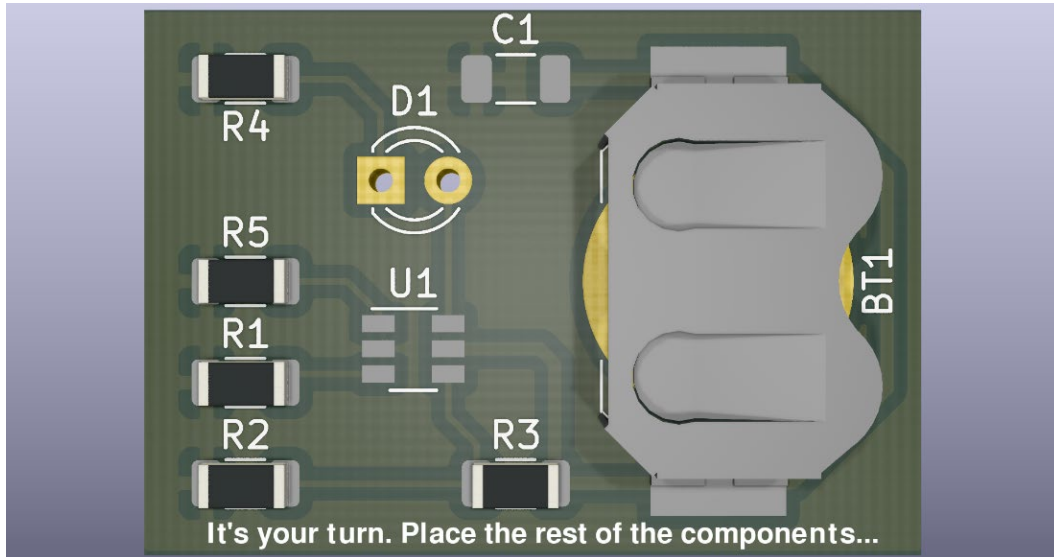
*** NEW READING ***
Reading temperature...
Temperature calculated value = 0.000000000 millidegrees Celsius
Temperature correct value = 24447.051390750 millidegrees Celsius
Wrong

Reading acceleration values...
Acceleration calculated values: x=0.000000000 y=0.000000000 z=0.000000000
Acceleration correct values: x=0.422645495 y=2.542068660 z=9.640608375
Wrong

/dev/ttyACM0 57600-8-N-1 DTR RTS CTS CD DSR RI
```

## Demo 4

An accelerometer-enabled game that lets you place components on a circuit by physically tilting the accelerometer.



## Procedure:

- Make sure you are in the `~/workshop/no-OS/projects/eval-adx1355-pmdz` directory.
- **Build the IIO\_EXAMPLE of this project:**
  - `$ make IIO_EXAMPLE=y WORKSHOP_EXAMPLE=n`
- **Program the board:**
  - `$ make IIO_EXAMPLE=y WORKSHOP_EXAMPLE=n`  
`run`
- **Change the directory and run the game:**
  - `$ cd ~/workshop/play`
  - `$ python3 play.py`



AHEAD OF WHAT'S POSSIBLE™

# Resources

## No-OS Wiki:

<https://wiki.analog.com/resources/no-os/api>

[https://wiki.analog.com/resources/no-os?s\[\]=no&s\[\]=os](https://wiki.analog.com/resources/no-os?s[]=no&s[]=os)


<https://www.analog.com/en/analog-dialogue/articles/understanding-and-using-the-no-os-and-platform-drivers.html>

<https://github.com/analogdevicesinc/no-OS/tree/master/projects>

## *Specific hardware resources:*

[https://wiki.analog.com/resources/eval/user-guides/eval-adxl355-pmdz/no-os-setup?s\[\]=no&s\[\]=os#adxl355\\_driver](https://wiki.analog.com/resources/eval/user-guides/eval-adxl355-pmdz/no-os-setup?s[]=no&s[]=os#adxl355_driver)

# Opportunities at ADI

- 
- Internships
  - Jobs
  - Summer practice
  - Get hardware and support from ADI to develop your own projects



## Our departments:

- Hardware Design
- FPGA Digital Design
- Embedded Software
- Applications Software
- Applications Engineering

Send us your CV!

To: [office.romania@analog.com](mailto:office.romania@analog.com)

Subject: Internship/Practica

Check our available  
positions with the QR  
code:



# Thank You! Questions?