

TUTORIAL 2 – COMMUNICATIONS

EQUIPMENT NEEDED

Your Laptop
Arduino Nano 33 IoT

NAU7802 + loadcell
Breadboard

DIGITAL BUSES

In this section we are going to work with digital buses. Specifically, we are going to use **only** the I2C bus. The reason is that Simulink uses the Serial bus to communicate with the Nano 33 IoT. Other Arduinos do not have this issue and Serial communications are possible.

I2C COMMUNICATIONS – THE BUILD-IN ACCELEROMETER

First we will use the build-in LSM6DS3 Accelerometer/Gyroscope. Please have the datasheet pdf also open for this activity (datasheet in Moodle). In the Moodle Tutorial Section for this Chapter there is a Template which you should use to complete the next steps.

The **I2C Slave Address** of the sensor is 0b01101010 (p.35 of the datasheet.)

For the Accelerometer to work you need to setup four **Registers** before you read the Acceleration and Gyroscope information. The Settings for each Register are as follows:

Register Name (Short Name, Hex Address)	Setting Short Name - Number of bits - (Function)	Value, (explanation)
Gyroscope Configuration (CTRL2_G, 0x11)	FS_125 – 1 – Quick Setup to 125dps ¹	0b0 (disabled)
	FS_G – 2 – Full-scale of Gyroscope	0b11 (2000dps)
	ODR_G – 4 – Data Rate for Gyroscope	0b0100 (104 Hz)
Accelerometer Configuration (CTRL1_XL, 0x10)	BW_XL – 2 – Anti-aliasing filter bandwidth	0b10 (100 Hz)
	FS_XL1 – 2 – Full-scale of Accelerometer	0b00 (±2g)
	ODR_XL – 4 – (Data Rate for Gyroscope)	0b0100 (104 Hz)
Gyroscope high-performance and high-pass filter (CTRL7_G, 0x16)	ROUNDING_STATUS – 1 – Rounding of Registers	0b0 (disabled)
	HP_G_RST – 1 – Gyro High Pass Filter reset	0b0 (disabled)
	HPCF_G – 2 – Gyro High Pass Filter cut-off frequency	0b0 (disabled)
	HP_G_EN – 1 – Gyro High Pass Filter Enable	0b0 (disabled)
	G_HM_MODE – 1 – High-performance operation Enable	0b0 (disabled)
Accelerometer Filters Configurations (CTRL8_XL, 0x17)	LOW_PASS_ON_6D – 1 – Low Pass filter Accelerometer	0b1 (enabled)
	HP_SLOPE_XL_EN – 1 – HP or Slope Filter Selection	0b1 (Slope)
	HPCF_XL – 2 – Slope Filter Configuration	0b00 (ODP/4)
	LPF2_XL_E – 1 – Low Pass Filter Enable	0b0 (disabled)

NOTE: Some registers have always-0 bits (e.g. bit 0 of CTRL2_G). Read the datasheet carefully!

Note: There is a mistake in the original datasheet for register CTRL8_XL. The Moodle pdf version is corrected.

¹ Dsp: degrees per second

- Based on the table above, the datasheet, and using the Simulink template provided complete the configuration of the two missing Registers (CTRL1_XL and CTRL8_XL). This should be done in the **Sensor Setup** subsystem using **I2C Master Write**, **Shift Arithmetic**, **Constant** and **Bitwise OR** blocks.

Things to notice:

How the blocks **Shift Arithmetic** and **Bitwise OR** are used to combine binary values.

All **Constants** are **uint8** datatypes (Block Parameters->Signal Attributes->Output data type).

- After the configuration use **I2C Master Read** blocks to read the values from the respective registers (datasheet) of the Accelerometer and the Gyroscope for X, Y, Z. The template has the X-axis.
- In the subsystem **Main Sensor Read** include 4 more **Outports** to export the values.
- Include 4 more **Gain** blocks (with correct values) before connecting 4 new **Display** blocks

Things to notice:

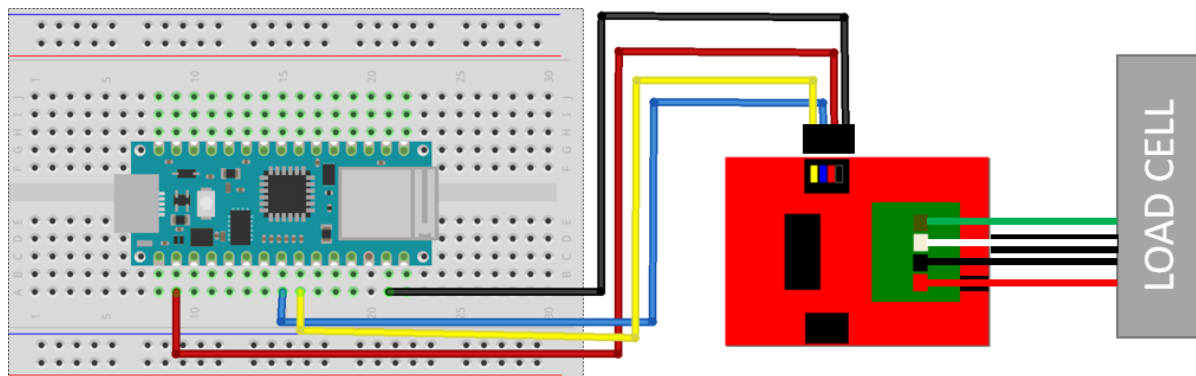
The **I2C Master Read** block is LittleEndian. Consider the reason why (hint: the answer is in the name of the registers you read)

The **Gain** blocks have values 2000/32768 and 2/32768. Why 2000 and 2? Why division by 32768? (hint: see the configuration option above. How many registers you read?)

- Run the model in **Normal** mode (for faster run times and adjustments) with **Pacing**. Observe the results when you shake/move/tap the Arduino. (If the displays change the configuration and read steps are working fine).

I2C COMMUNICATIONS – NAU7802 LOAD-CELL AMPLIFIER

The next sensor we will use is the NQU7802 load cell amplifier. The connection diagram can be seen below.



For the connection between the NAU7802 and the Arduino you should use the cable with the JST header, and the jumper ends on the other side. For the load-cell you should use the spring-loaded terminal block. You must push the orange leavers on the block, insert the load-cell wire (exposed copper) in the hole and release the lever. If the wire cannot be removed, you are done! Otherwise you need to repeat the push/insert.

The board is powered and communicates with the Arduino via these cables which are part of a I2C implementation called [Qwiic](#) by Sparkfun Electronics. Many sensors can be connected in series using this protocol and then accessed based on their I2C Slave Address.

For the NAU7802 the **I2C Slave Address** is **0x2A** (p.16 of the datasheet.)

Now using the template from Moodle for the NAU7802, the datasheet, and the configuration register settings below create a Simulink model to read the raw value of the load-cell.

Register Name (Short Name, Hex Address)	Setting Short Name - Number of bits - (Function)	Value, (explanation)
Power Up Register (PU_CTRL, 0x00)	RR – 1 – Register Reset (this happens before the others) ²	0b0 (disabled)
	PUD – 1 – Power up digital circuit	0b1 (power up)
	PUA – 1 – Power up analog circuit	0b1 (power up)
	PUR – 1 – Power Ready	READ ONLY ³
	CS – 1 – Cycle start	0b0 (no sync)
	CR – 1 – Cycle ready	READ ONLY ³
	OSCS – 1 – System Clock source select	0b0 (internal clock)
	AVDDS – 1 – Analog Voltage Reference	0b1 (internal LDO)
Amplifier Configuration (CTRL1, 0x01)	GAINS – 3 – Amplifier gain	0b111 (x128)
	VLDO – 3 – Internal LDO voltage (must be <=Vcc)	0b100 (3.3V)
	DRDY_SEL – 1 – Conversion Ready Pin behaviour	0b0(flip when ready)
	CRP – 1 – Conversion Ready Pin Polarity	0b0 (0:ctive High)
Calibration, Rate, and Channel Register (CTRL2, 0x02)	CALMOD – 2 – Calibration Mode Selection	0b00 (Internal Calib.)
	CALS – 1 – Start Calibration based on Mode	0b0 (No trigger)
	CAL_ERR – 1 – Calibration Error bit	READ ONLY ³
	CRS – 3 – Conversion rate in SPS (Samples per second)	0b011 (80SPS)
	CHS – 1 – Analog input channel	0b0 (Ch1)
ADC Configuration Register (OPT_B1, 0x15)	REG_CHP – 2 – Delay between ADC and chopper	0b00 (no delay)
	ADC_VCM – 2 – Unipolar Configuration	0b00 (disabled)
	REG_CHPS – 2 – Chopper clock frequency	0b11 (turned off)
Power Control Register (0x1C)	PGA_CURR – 2 – Programmable Gain Amplifier Current	0b00 (100% master)
	ADC_CURR – 2 – ADC Current	0b00 (100% master)
	MASTER_BIAS_CURR – 3 – Master Current Bias	0b000 (100%)
	PGA_CAP_EN – 1 – External bypass Capacitor Enable	0b1 (enabled)

- Complete the Subsystem **Sensor Setup** for all five registers.
- In the subsystem **Main Sensor Read** use an **I2C Master Read** block to read the ADC values and connect the result to the **Output**. Read all three bytes with a single block (What you need to consider?)
- The output from the previous step needs to be MSB first (Big Endian) for the subsystem **24bit (3 byte) signed value to 32bit signed value** to work. (How is the subsystem performing the conversion?)
- Run the model in **Normal** mode (for faster run times and adjustments) with **Pacing**. Observe the results when you hold one end of the load-cell and tap/push the other. One direction should give you positive and one negative values.

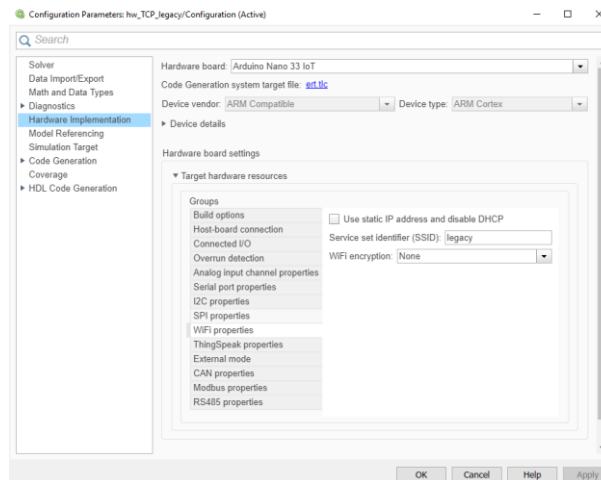
² In the template you will see that there are setup subsystems

³ This means we leave it as 0 when we write to the register

SETTING UP YOUR SYSTEM FOR WIFI COMMUNICATIONS

For TCP communication you will need to connect your laptop to a WiFi Network.

While in the university you can connect to the network **legacy** with no WiFi encryption and unticked the *Use static IP...* The image below shows this configuration via the model parameters:



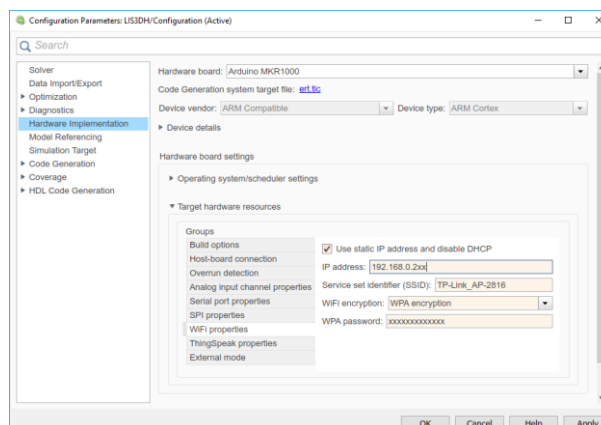
Warning: To be able to connect to **legacy** you need to have the MAC address of the Arduino registered ahead of time. The process is given in [this](#) Moodle page.

At home you will need to use your home WiFi, so you need to know the name (SSID).

Warning: Your home WiFi needs to be 2.4GHz, most house routers are 5GHz but can also enable 2.4GHz.

Most likely your encryption is WPA.

It is preferred to use a static IP address. To find what form your network IP addresses are use the command **ipconfig** in Windows *command prompt* or the command **ifconfig** in the Mac OSX *terminal* to get your laptop/desktop IP. The IP will be in the form 192.168.<sthA>.<sthB>. Set the Arudino's IP to 192.168.<sthA>.200 or another high (<255) value. An example network setting can be seen below:



These two steps should be enough for you to get ready for the next activities.

TCP/UDP COMMUNICATIONS

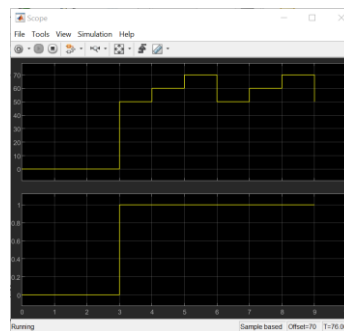
SEND DATA TO THE ARDUINO

First we will see how to send data to the Arduino using a TCP/IP connection.

1. Connect both outputs of the block **WiFi TCP/IP Receive** to a **Scope**.
2. Setup the block as a *Server* and ensure that the *Sample time* is 1. Note the port (default is 25000)
3. In the *Hardware* Tab set the *Stop Time* of the model to *inf*. You need to do this so you can test the Python code multiple times.
4. Run the model in *Monitor & Tune* Mode.
5. Download from Moodle the python script *TCP_Client_RepeatSend.py*, and open it in your Python IDE.

This script connects to the Arduino and afterwards sends 10 times the numbers 50, 60, 70 with an interval of 1 second.

6. Check that line 15 *TCP_IP* is the IP of the Arduino and that line 18 *TCP_PORT* is the above noted port.
7. Run the script and observe in the Scope the series of ladders (similar to the image below). Also notice that the status should be 1 when there is data to read.



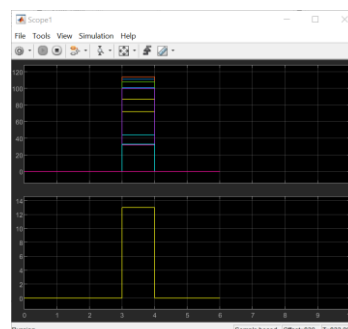
8. You can change line 30 to other numbers and the Scope should reflect this.

Then we will see how to send data to the Arduino using a UDP Datagram.

9. Connect both outputs of the block **WiFi UDP Receive** to a **Scope**.
10. Set the *Data size (N)* to 14 and ensure that the *Sample time* is 1. Note the Local IP port (default 50001)
11. In the *Hardware* Tab set the *Stop Time* of the model to *inf*. You need to do this so you can test the Python code multiple times.
12. Run the model in *Monitor & Tune* Mode.
13. Download from Moodle the python script *UDP_Send.py*, and open it in your Python IDE.

The script sends to the Arduino the message *Hello, World!* as a single burst of 13 bytes.

14. Check that line 13 *UDP_IP* is the IP of the Arduino and that line 15 *UDP_PORT* is the above noted port.
15. Run the script and observe in the Scope how the values stack up, as per the figure below. Also notice that the Size was 13 indicating how many bytes have arrived.



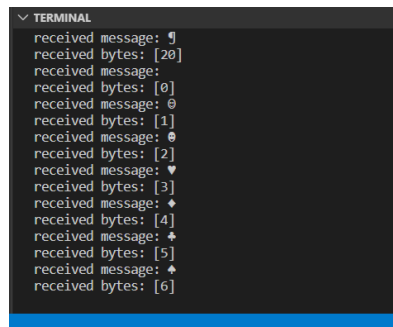
RECEIVE DATA FROM THE ARDUINO

Now we are going to receive data from the Arduino. We are going to use only the UDP method because the TCP/IP method tends to be more tricky to realise on the PC side.

1. Connect a **Counter Limited** source block via a **Data Type Conversion** block to a **WiFi UDP Send**.
2. Configure the **Counter Limited Upper Limit** to 20.
3. Select in the **Data Type Conversion** the **Output data type** to *uint8*.
4. In the **WiFi UDP Send** in the IP address enter your laptop's /desktop's. Note the Remote IP port (default 50002)
5. In the **Hardware** Tab set the **Stop Time** of the model to *inf*. You need to do this so you can test the Python code multiple times.
6. Run the model in *Monitor & Tune* Mode.
7. Download from Moodle the python script *UDP_Receive.py*, and open it in your Python IDE.

The script prints the value received from the Arduino as an ASCII character and as a decimal value (>255)

8. Check that line 18 *UDP_PORT* is the above noted one.
9. Run the script and you should see in the terminal the numbers coming in one at a time. The result should be similar to the one below.



```
▼ TERMINAL
received message: 0
received bytes: [20]
received message: 0
received bytes: [0]
received message: 0
received bytes: [1]
received message: 0
received bytes: [2]
received message: 0
received bytes: [3]
received message: 0
received bytes: [4]
received message: 0
received bytes: [5]
received message: 0
received bytes: [6]
```

10. Try to combine the send and receive blocks above with other Arduino blocks, e.g. turn LEDs on and off from Python etc.