

TUTORIAL 1 – SIMULINK AND ARDUINO

EQUIPMENT NEEDED

Your Laptop
 Arduino Nano 33 IoT
 Breadboard
 One or more LEDs
 One or more Switches

SN74HC74N D flip flop
 SN74HC00N NAND Gate
 PC-based oscilloscope
 Wires

SIMULINK WORK

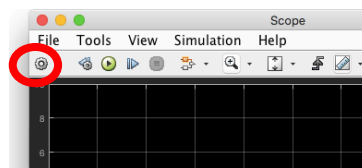
WORKING WITH SOURCES, SINKS, AND THE DASHBOARD

1. Test different sources from the **Library->Source**, e.g. **Constant**, **Pulse Generator**, **Repeating Sequence**, **Sine Wave** with **Library->Sink->Scope** or a **Library->Sink->Display** or the **Data Inspector**.
2. Change the different parameters in the sources and see the effect in the sinks/inspector.
3. Connect a **Library->Source->Signal Builder** to a **Library->Sink->Scope**. Double-click on the **Signal Builder** and adapt the signal created. Create one signal that has, in this order:
 - a. After 1 – a pulse of magnitude 1, duration 2;
 - b. After 2 – a triangular pulse of magnitude 4, duration 3;
 - c. After 1 – a pulse of magnitude 2, duration 1;

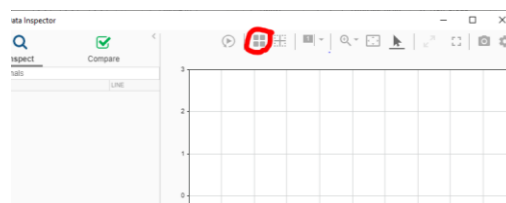
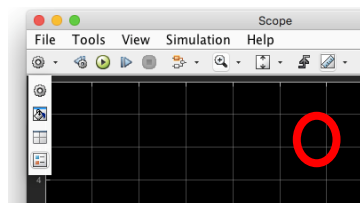
Note: To add points to a signal you need to Shift+click.

4. In the scope, experiment with parameters

Note: You can do the following using the Data Inspector instead.



- a. Connect **more than one sources** to the scope. (Drag sources into the scope and new input ports will appear).
- b. Create **stacked scope views** for each signal OR in inspector use stacked views



5. Place a **Knob** and a **Dashboard Scope** in a model with at least a constant. You can use the Dashboard to create a User Interface to interact with the model.

Note: To connect Dashboard controls you need to double-click on the control and then select a tuneable parameter (e.g. constant value) or one or more signals.

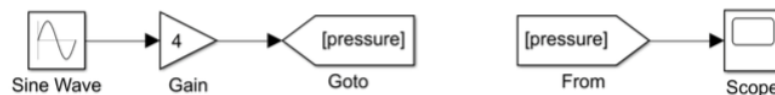
Note: The Dashboard Scope will not behave “sensibly” in *Normal* simulation mode. It will behave better in simulation mode.

USING THE DIFFERENT RUNNING AND PACING OPTIONS

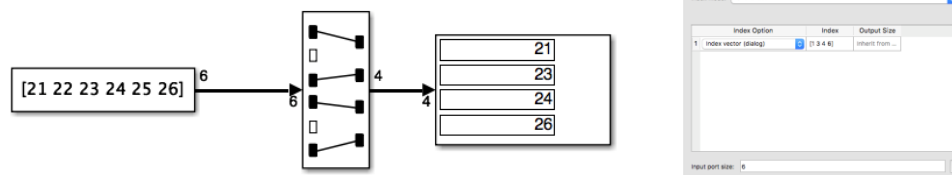
For the above (as well the below) tests do not forget to change the *Duration of Run* as well as the *Pacing* of Simulation Options.

WORKING WITH MATH OPERATIONS, LOGIC AND BIT OPERATIONS, AND SIGNAL ROUTING

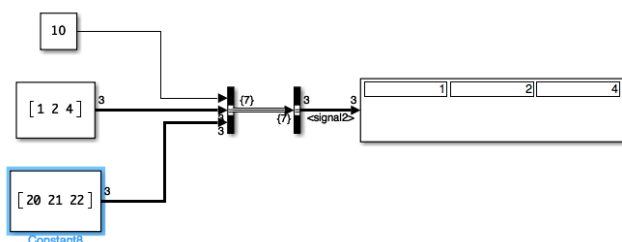
- Experiment with different **Math Operations** blocks and **Sources** the most common ones are:
 - **Abs**: Absolute value of incoming signal
 - **Add** and **Divide**: You can increase the number of input ports by adding more +, -, *, / in the parameters
 - **Gain**: The simplest way to multiple a signal
 - **Sum**: Another way to do additions. *Very common in control models (will see in future tutorials)*
- Experiment with **Logic and Operations** most common/useful are:
 - **Compare to Constant**: This will return a 0 or 1 depending on the comparison
 - **Relational Operator**: This will perform a comparison between the two values and return 0 or 1.
- Experiment with **Signal Routing** blocks. The most common/useful are:
 - **From/Goto**: Those can be used to connect two signals that are far away in the canvas. Use this to declutter your model. For example :



- **Selector**: It enables you to select specific elements from a matrix. For example (and parameters):



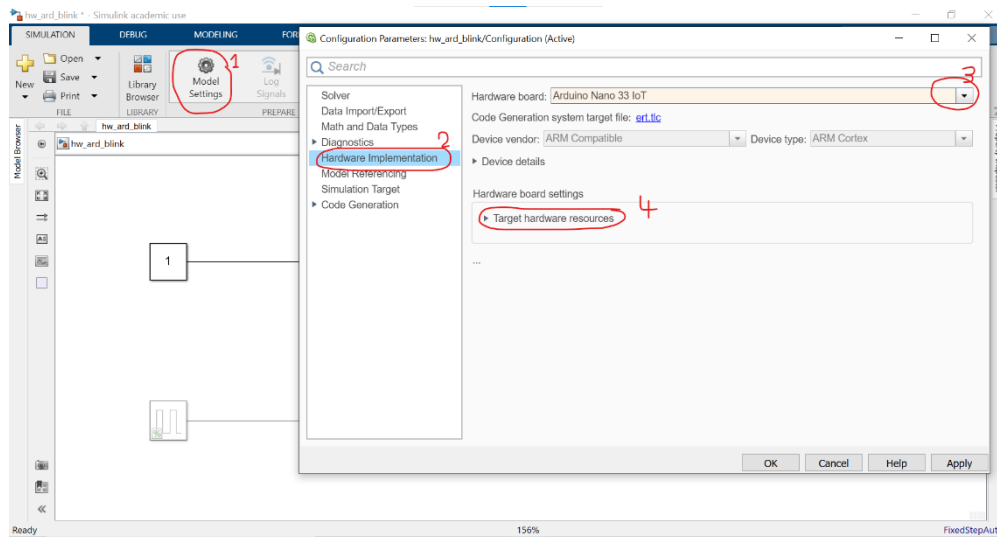
- **Bus Creator/Selector** and **Mux/Demux**: This allows you to group signals together. The Mux/Demux is for same type and dimensions will the Bus is for any type and dimensions. Try the following to test this:
 - Create three constants, one with a value of 10 one with a matrix [1 2 4] and one with a matrix [20 21 22]
 - Group them with a Mux
 - Connect the output to a display. What is the result?
 - Change the type of 10 to *int8*. This can be done in Parameters>Signal Attributes> Output data type. Does this work?
 - Group them with a Bus instead. Notice the more parameters.
 - Ungroup them with a Bus selector. Notice that you need to select which signal you want. This means that will be good to have them labelled.



SIMULINK IN ARDUINO

You can run a model on an Arduino as we have seen in three modes: **Normal**, **Monitor & Tune**, **Deploy**. Not all blocks will work in the first, but all should work in the other two. **Normal** operation is like the one we have seen above. It is recommended to run all your models first in **Normal** (with Pacing) before trying **Monitor & Tune** or **Deploy**.


In order to enable Running in your Arduino you must select the appropriate hardware in the **Model Settings (1)**, under the **Hardware Implementation (2)** and select the **Hardware Board (3) Arduino Nano 33 IoT**:




Note: In case you have issues with connecting to the Arduino check what is selected under **Target hardware resources**→**Host-board connection (4)**. *Automatically* should work most of the time.

WARNING: THE Nano 33 IoT IS 3.3V compatible, DO NOT USE THE 5V PIN PROVIDED FOR ANY OF THE FOLLOWING EXPERIMENTS

Important note – **Compiling for Hardware**: Running a model in **Monitor & Tune** mode can take up to 1.5 minutes (depending your operating system) before it starts, this is normal since the model needs to be compiled into C code and transferred to the Arduino.

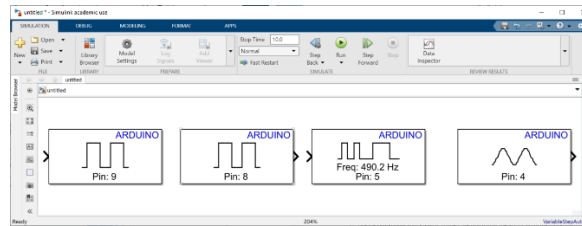
Alternatively, and this is also for final use of your model, you can use the option to **Deploy** to Hardware . This is faster **BUT** any Scopes and Displays will not update. We will see ways to provide feedback via communication channels in the respective sessions.

Important note – **Sample Time**: When creating a model for physical hardware you need to think carefully about the *Sample time* of blocks. Blocks like the **Digital Input** and **Analog Input** have the *Sample time* parameter (double-click). This value is usually in seconds, i.e. 0.1 means every 100ms or a refresh rate of 10Hz. For Outputs, e.g. **Digital Output**, **PWM**, **Analog Output** this block are using the sample time of their source. Most sources, like the **Constant** will have a *Sample time* of **inf** which is impossible in a physical system.

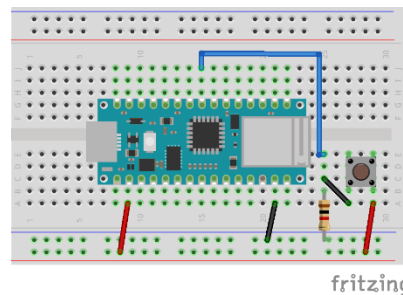
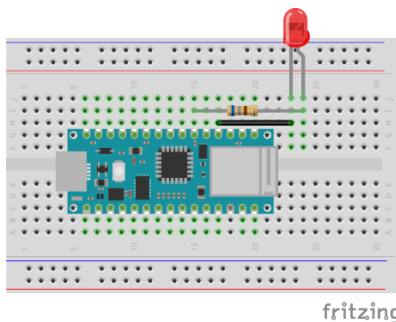
Do not forget to enable the sample time annotations to help you observe this. Either via the  or by <Right-click> -> **Sample Time Display**

PRACTICE WITH THE BASIC ARDUINO BLOCKS

Using some of the common Arduino blocks, below, combine them with **Sources** and **Sinks** as well as other blocks.



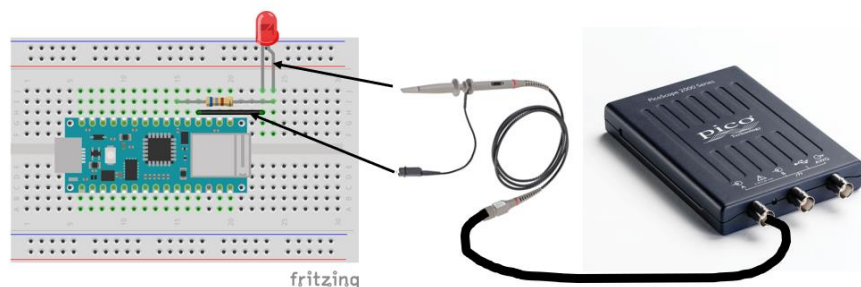
You can connect a Switch and an LED as can be seen below. Notice the resistor in line with the LED (680Ohm) and the pull-down resistor (1kOhm) for the switch.



Warning: The Nano 33 IoT can handle up to **7mA** per pin, you will need **at least 470 Ohm** in line to ensure that the LED is not frying the Arduino. The image above uses 680 Ohm.

1. Use the **Digital Input** block with the Switch. Connect the switch from the VCC (3.3V) to the Digital Input pin. (Select the correct pin by *Double Clicking* the **Digital Input** block, the image above uses D5)
2. Use a **Source**, e.g. the **Signal Builder** or **Repeating Sequence Stair** with the **Digital Output** and the LED to create an SOS in Morse code. Connect the LED's anode (long leg) to the pin (image above uses D3) with the in-line resistor. (Alternatively, you can use the on-board LED in Pin 13)
3. Connect the LED and control it with the **PWM** block.

You can also observe the PWM signal using the oscilloscope. A proposed connection below.



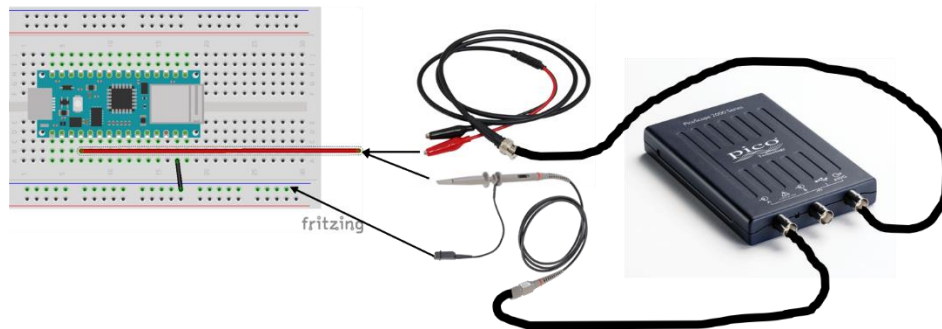
4. Using your oscilloscope connect Channel A to the anode of the LED and monitor the **PWM** signal. (Do not forget to connect the ground crocodile clip to the ground of the Arduino)

Notes about the **PWM** block:

The block accepts values between 0 and 255

You can alter the *Frequency* option to *Specify*. This does not have an observable effect for LEDs (unless too low) but will play a role when controlling Motors. This should be visible with the oscilloscope only.

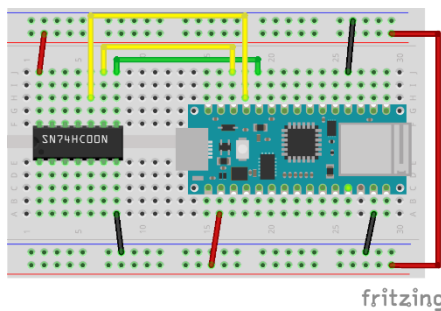
For using the Analog Input you can use Wave Generator of the oscilloscope. The connection can be found below.



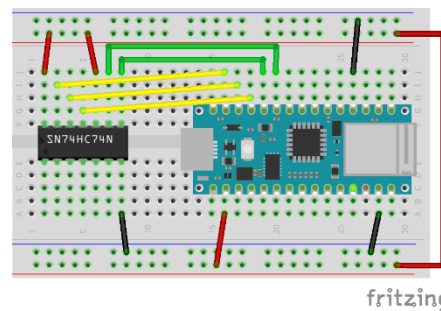
5. Connect the **Analog Input** block to a scope or display. You can use the Data Inspector if you prefer
6. Set the Duration of Run to **Inf**.
7. Run the model in **Normal** mode (remember the Pacing) or in **Monitor and Tune** (slower start)
8. From the Picoscope software alter the behaviour of the Signal Generator and observe the differences.

COMPARE A MODEL WITH REAL COMPONENTS

In this section we will make some simple examples on how we can compare models with “real” components and systems. Specifically, we will compare the behaviour of NAND gates and D flip flops with their physical counterparts. **There should not be any difference** given the simple nature. You will need to wire the Arduino to the NAND and D Flip Flop ICs (possible one at a time) as seen below.



Connections: A->D10, B->D9, Y->D8



Connections: CLR->D11, D->D10, CLK->D9, Q->D8, !Q->D7

For the NAND Gate, first the “virtual”

9. Find the block **NAND** under Simulink->Quick Insert->Logic and Bit Operations.
10. Connect two **Sources** to the inputs. It can be **Constants** (not much to see), **Pulse Generators** of different *Period/Pulse Width*. Any other source.
11. Connect a **Sink** (maybe a **Display** or **Scope** or the **Data Inspector**) to the output of NAND.
12. Run the model in Normal and observe that the truth table for a NAND gate (below) is achieved.

NAND Gate Truth Table

Inputs		Output
A	B	Y
H	H	L
L	x	H
x	L	H

x: the state of the pin does not matter

Now the “physical”

13. Connect the same above **Sources** to two **Digital Write** blocks (pins D10, D9 in the above connections)
14. Connect the same above **Sink** to a **Digital Read** block (D8).
15. Remember to setup the model for the Arduino Nano 33 IoT
16. Run the model (**Normal** will be the fastest) and observe that the truth table for a NAND gate is achieved.

If you have not done so you can have the “virtual” and the “physical” running on the Arduino and directly compare the output from the **NAND** block with the output from the **Digital Read** block. This will be best done in the **Data Inspector** or a **Scope**

Note: This is a more complicated experiment because the changes happen on transitions.

For the D Flip-Flop Gate, first the “virtual”

17. Find the block **D Flip-Flop** under Simulink Extras->Flip Flops.
18. Connect three **Sources** to the inputs. It has to be a **Pulse Generator** for the signal CLK. The other two (D and !CLR) can be anything but **Pulse Generator** with different *Period/Pulse Width* will work the best
19. Connect **Sinks** (maybe a **Display** or **Scope** or the **Data Inspector**) to the outputs.
20. Run the model in **Normal** and observe that the Function Table for a D Flip Flop (below) is achieved.

D Flip Flop Function Table

Inputs			Outputs	
!CLR	CLK	D	Q	!Q
L	x	x	L	H
H	↑	H	H	L
H	↑	L	L	H
H	L	x	Q ₀	!Q ₀
x: the state of the pin does not matter ↑: the change happens on the rise edge Q ₀ : the previous state will remain.				

21. Connect the same **Sources** to three **Digital Write** blocks (pins D11, D10, D9 in the above connections)
22. Connect the same **Sinks** to two **Digital Read** blocks (D8, D7).
23. Remember to setup the model for the Arduino Nano 33 IoT.
24. Run the model (**Normal** will be the fastest) and observe that the Function table above.

As with the NAND gate compare if there is any significant difference.

Note about D Flip Flops: Flip Flops are the simplest elements of memory in electronic systems. In effect they are bit memory blocks that can store/recall the previous state. 8 Flip Flops can be used for an 8-bit register.