Training Unit 8: Arduino – Matlab/Simulink

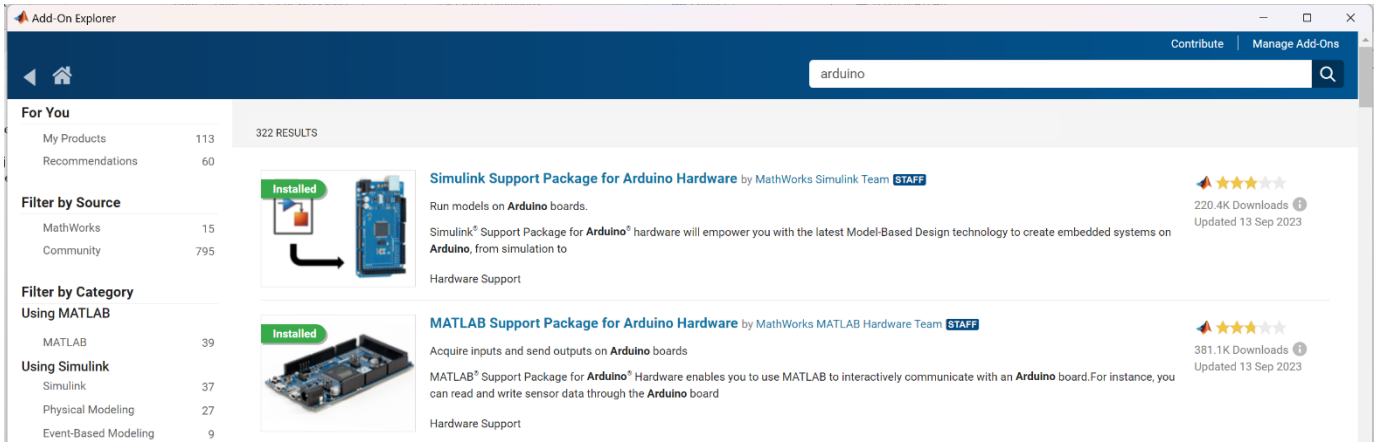
1 INTRODUCTION TO TRAINING UNIT

In the last lecture unit, we covered the topics of *testing*, *verification* and *validation* with the help of Matlab / Simulink when using the V-model as development process. We were looking into different possibilities to use Simulink in combination with real hardware or powerful real-time computers for verifying that the software works as expected. We also discussed a fast development possibility, were developing and testing on the real hardware is used. This possibility is also called Rapid Control Prototyping (RCP) and will a major topic for this training unit.

2 PREPARATION

2.1 Get all necessary Add-on packages

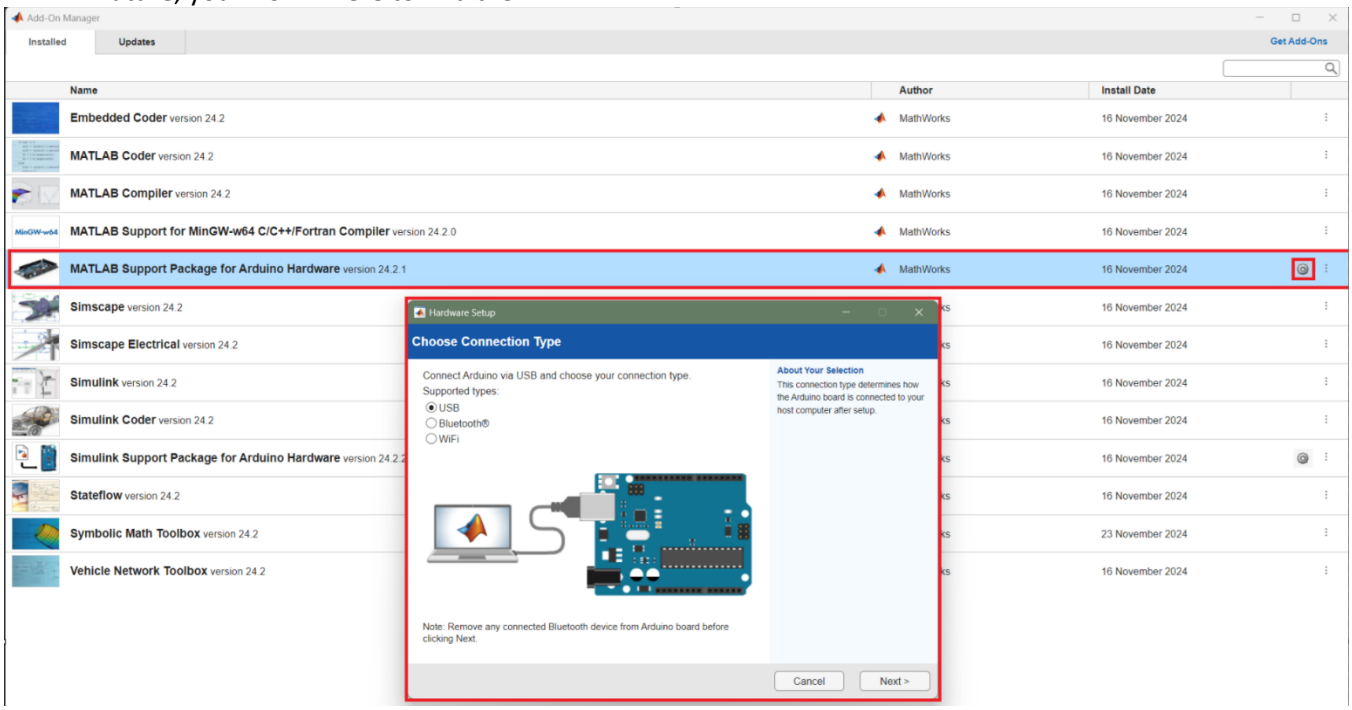
- For this unit you will need following Mathworks Add-ons toolboxes (included in students license):
  - Matlab Coder
  - Matlab Compiler
  - Simulink Coder
  - Embedded Coder
  - Simulink Support Package for Arduino Hardware (free Add-On)
  - Matlab Support Package for Arduino Hardware (free Add-On)
- As you used the first 4 Add-Ons from the bullet point list in training units 6 and 7, the installation should already be on your computer. For detailed information regarding the installation process, refer to training unit 6, section 2 documentation.
- To install the Matlab and Simulink Support Package for your Arduino Uno, you need to go to your home tab and click the Add-ons button.
- This opens the Add-on window where you can search for both packages (see picture below)
- As soon as you installed them, there will be a green label above both packages, indicating that everything was successful during the installation process.
- If the installation process does not work, use the available FH Joanneum computers. Here all necessary Add-on packages should already be set-up correctly.



2.2 Warm-up using the Arduino Explorer

- After installing the “Matlab support package for Arduino Hardware” you will be asked to configure the “Add-on” package (if you did not do it, you can find the configure option within your “Add-On manager”)
- With this option you can upload several server options to your Arduino Uno board, that Matlab can use

to communicate with it. We will not need any of those options for this task, but if you need them in the future, you know where to find them.



- Choose **Uno** as board and select the correct "Port" (should only be one available, once you connected your Arduino to your PC) click on "Program".
- Once you programmed the board (should only take some couple of seconds) press next which loads the "Test Arduino Section" screen where you can verify the connection
- After verifying your connection, you will be asked to open the **Arduino Explorer app**. Make sure to tick this option and check out this app (if you missed it, you will be able to access it via the Apps tab under the "Test and Measurement" section any time)
- Once you opened it, you should already see your Arduino board with the COM-Port that you specified in the previous step
- Click on it and explore this app. You will find a Pinout description and some possibilities to operate with your board.
- Make a first test by connection a digital output to one Arduino Analog input. Then control the output pin using the "Pin Explorer" within the application. Configure the Analog input port and the digital output port with the help of this app.
- Write a "High" to the pin and try to measure it using the "oscilloscope" like chart at the bottom.
- You have now successfully finished the configuration of your board and verified the connection with your first "Hello World" by controlling some pins of your board. This concludes the warm-up section for this training unit

3 REPORT

- The report for this training unit is based on a **Matlab live script** and **all your files** that are created during the training unit. (zip your working directory and upload it to Moodle)
- Make sure to structure your work in the live script using the available editor tools (structure your code in sections, add headers to your sections, table of contents, use **bold** or *italic* formats, bullet points, etc.)
- As you will consistently add new features to your live script, make sure to add new sections to each task, to provide a well-documented and structured live script. This will require you to copy-paste some of your code throughout the document most likely. Use the **subtask number** of this training unit description and **header** for

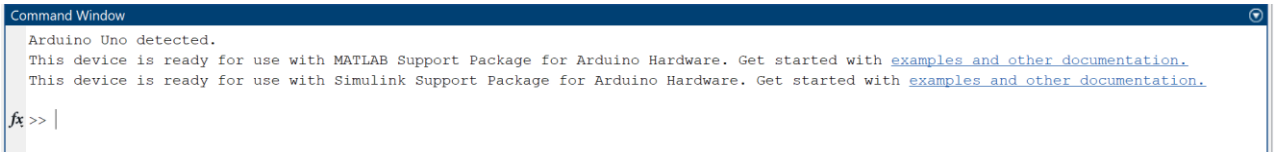
your live script as **section header** for a more **consistent documentation**

- Try to make use of UI-controls throughout the script, to generate an interactive script
- Be sure to check the correct function of your scripts before uploading it. If your script **does not work**, you **will not receive any points** for this training unit.
- Make sure, that the script shows **your work** on this training unit task. Using solutions from other colleagues will result in zero points for you and your colleague.
- **Caution (!)** be sure to check the **due date** that is set for uploading the files on Moodle!

4 LABORATORY WORK

4.1 Establish a connection between your Arduino Uno Board and Matlab

- Even though we already verified the correct connection to your board, we will test it once again, just to make sure, everything is correctly installed
- Therefore, disconnect your Arduino Uno and restart Matlab
- After starting Matlab successfully, connect your Arduino Uno Board and check whether you receive following message automatically in your Matlab Command window:



- If you installed both Support Packages successfully, Matlab will be able to recognise your Arduino Uno Board and give you two links that lead to MathWorks documentation sides, were you can get more detailed information and examples. (If you have troubles or are interested in some advanced features, those links are a good starting point)
- Your first test is to connect to your Board using following command:

```
a = arduino('COM3','Uno', 'BaudRate', 115200)
```

  - Where 'a' is the name of your Arduino class within your Matlab environment, that stores your connection setup to your board. You can choose any name you want for that variable. This variable will be used throughout the training unit to operate with your board
  - You can omit everything within the parentheses if you want, as Matlab should automatically recognise your board. You can check the COM port within your windows device manager for example
- The first connection process can take up to some few minutes, so make sure your do not abort the connection process
- Be sure that you closed all other running connections to your Arduino board (e.g. the Arduino Explorer App), as only one connection can be active at a time
- After successfully connecting to your board, check out your workspace variable, where the connection to your board is stored ('a')
- Make sure you do not overwrite or delete it, as this is necessary for communicating with your board. Without the variable your connection will be lost to your board
- What can you see in this class variable ('a')? Examine it closer.
- A class usually has also some methods defined, that act on the classes' objects. The class 'a' does have some predefined methods as well. To check out this classes' methods, use following command:

```
methods(a)
```

- Some of those methods will be required for the next tasks, so make sure to check out the methods using Matlab's "help" or check out this page (link) for an overview of available methods
- (**Hint for following subtasks!**) If it is possible, leave the breadboard hardware setup, that you build in every subtask, remaining on your breadboard (if you have enough space and enough components), as in later tasks you may can reuse some of the hardware set-up again, therefore saving time.

4.2 Using Matlab to control inputs and outputs of your board

- The first task will be the control of outputs of your board and the reading back the binary feedback of those outputs
- Therefore, you need to configure the pins correctly to be an input / output
- For this configuration you need the method configurePin():

```
ret = configurePin(a, “pin_as_input”, “Type”);
```

- Look at the schematic of your Arduino board, to check which pins are free to be used as inputs or output respectively
- Before you connect your input to your output, try to read the input pin by using following command:

```
ret = readDigitalPin(a, pin_as_input)
```

- If you call this command several times, can you see a change in the read value? Try to touch one end of a connection wire attached to the input pin with your finger. Why is the value changing arbitrarily? Can you explain this behaviour?
    - Tip: Use following small function and adapt it to your workspace variables, to continuously update a Matlab plot and to be able to see live changes while reading the input. (This function resembles an oscilloscope within a Matlab plot)

```
figure
plot(1,1); % create plot with dummy value
ax = gca; % get handle of current axes;
line = get(ax, 'Children') % get handle to line object

for run_var = 1 : 100
    read_value = readDigitalPin(a, pin_as_input);
    pause(0.1);
    line.XData = [line.XData run_var];
    line.YData = [line.YData read_value];
end
```

- With this function, you should be able to observe live data from your Arduino Uno board like you would with an oscilloscope but with very slow sampling frequency. You can export the video to an *.mp4 or *.gif, if you like as well. Export your video and then embed it to your report.
- To resolve the problem of random value reading, check the “configurePin” method for an alternative configuration type for digital inputs, to avoid those phantom read values
- Have you spotted and tested the new input type configuration? Make sure to explain the initial problem and the solution to the problem properly in your report
- Now connect input and output together and try to write and read the output values

4.3 Find out the minimum cycle time possible

- You may have already experienced some delay from sending the command to your Arduino Board until your command is computed by the Arduino Board
- Now lets try to find out the lowest cycle time, that is possible with your Arduino Board and Matlab in this rapid control prototyping setup
- Create a small test bench, that sends a command to your Arduino and measure the time it takes to evaluate the command:
    - Use a for-loop that runs for a minimum of 100 cycles
    - In your for-loop, use the command to toggle an output pin
    - Use Pin “D13” for this test, as the yellow SMD-LED is already connected to this pin on your board
    - Following tasks shall be done within your for-loop testbench
        - Set to output to digital “High”
        - Wait for 10 millisecond
        - Set to output to digital “Low”
    - This should toggle the onboard yellow SMD-LED
    - Now add a tic command before setting the output to “High” and a toc command directly after this command
    - As there will be no huge differences between setting the pin to “High” and setting the pin to “Low” only one tic-toc command is necessary (you can measure the time for setting the output to “Low” as well, to check this assumption)
    - Now check / plot your results. What can you examine from your testbench? What is the minimum cycle time that can be expected from this set-up?
- Now let’s try to change the baud rate of your Matlab <-> Arduino connection to see the effects of the serial communication speed on the cycle time
- Therefore your need to close your current Matlab <-> Arduino connection by deleting the Arduino class variable (‘a’) and restarting the connection with a different baud rate speed.
    - First try a very low speed like 110 baud/s
    - Then try a faster speed like 256 kbaud/s
- Re-run the performance testbenches and compare the results from your standard baud rate of 115.2 kbaud/s then your low speed connection with 110 baud/s and then with your increased speed of 256 kbaud/s

4.4 Create a visual binary counter

- To get some practise with setting the output of your Arduino board, lets create a binary counter, that counts from 0 to 15 and visualises the counter value with the help of 4 external LED's
- For this small project you will need the following hardware:
  - 4 Outputs from your Arduino Board
  - 4 LED's
  - A breadboard
  - Some cables to connect the Arduino to the breadboard
  - 4 resistors
- Implement the binary counter within a for-loop that increases the counter value every second until reaching its maximum value
- After the visual counter works as desired, add an additional feature to your visual counter to display the number that a user can input in the Matlab command window
  - Therefore use a while-loop instead of a for-loop that runs until a non-numeric value is provided by the user
  - The while-loop should update the LEDs after every user input
  - The while-loop should stop and wait during every iteration, until a user input is given
  - Is the user input non-numeric, the while-loop should be exited
  - Test your function if it works as intended

4.5 Use the PWM outputs and the ADC inputs for LED intensity control

- Now let's test some more complex functions of the Arduino Uno board which are the analogue read and PWM write functions
- Let's Start with the PWM output test
    - First check out the schematics again to find a suitable output pin
    - Configure the pin as PWM mode
    - Use a LED with resistor and connect it to the correct output pin
    - Write a desired voltage value to the PWM output pin and check if the LED gets brighter / darker
    - Write a small test bench, that increases and decreases the LED intensity from 0 V to 5 V and back to 0 V again
- Now that you are familiar with the PWM output functionality, let's use the ADC input to measure voltage levels
- For a warm-up try following test setup:
    - First check out the schematics again to find a suitable ADC input pin
    - Configure the pin as Analog Input
    - Connect the ADC input to the GND reference and check if you read 0 V
    - Now connect the ADC input to the 5 V reference and check if you read 5 V
    - Now connect the ADC input to your PWM output pin, set it to 2.5 V and read the voltage back
        - What do you receive? Do you read the voltage value that you specified in your PWM output (2.5 V)? Explain your results
    - Now use a potentiometer and connect it to your bread board and use the 5 V and 0 V for the two end terminals of your potentiometer
    - Use the potentiometer grinder contact for your ADC input
        - Are your now able to read the voltage correctly? Explain why it works now and before it did not work with the PWM output
- After working independently with PWM output and ADC input, now it is time to connect both functions to be able to control the LED light intensity by using the potentiometer
- Consider following steps:
    - Connect your LED and resistor to your PWM output
    - Connect your potentiometer to your ADC input
    - Continuously read the ADC voltage in your while and update the PWM output value accordingly
    - To exit the endless while-loop, consider a function that counts the time a certain input value occurs and leaves the loop as soon as a defined counter value is reached
        - E.g. the ADC voltage reads 0 V for 3 seconds à exit main loop
        - Or implement another function, that allows the infinite-loop to be exited
- This task concludes the Matlab rapid control prototyping session. The following task will be implemented in Simulink. As mentioned in subtask 4.1, leave your created hardware set-up remaining on your breadboard, as you can reuse it in later subtasks

4.6 Connect your Arduino hardware to your Simulink

- As you have already installed the Simulink hardware support package during the training unit preparation, you can now open a Simulink model and connect your Arduino board to it.
- Got to Apps à Open drop-down menu à Setup to run on hardware à Run on hardware board and select your Arduino Uno board.
- This opens a new toolstrip menu named “Hardware” where specific settings can be done for your device
- You can specify 2 modes which are the “External mode” and the “Inputs/Outputs mode”
- First let’s try to use the “external mode”
- Now we still need to check the solver settings
    - Open the Configuration parameters à Solver
    - Make sure you select “Fixed-Step” and “Discrete solver”
    - Change the step size to 10 ms for a first try
    - Change to the Hardware Implementation section and check if the Arduino Uno board is selected here as well
    - If you are interested in some specific device settings you can explore this section in more detail
- Now we should be able to connect to the board. Therefore add some blocks that are related to the Arduino Uno board. Go to the Library Browser and under Simulink Support Package for Arduino go to the Common section and add a “Digital Output” block
- It is rather straight forward to configure this block. Double-click on the block and specify the desired pin and you are ready for a first test.
- Now add a constant block and set the output to 1 or 0 and connect it to your “Digital Ouput” block
- Click on run, which should start the code generation
- After some seconds/minutes the code generation should be finished
    - Did you successfully connect to the Arduino board?
    - Did you receive some errors regarding communication errors, as the board was not found?
    - Try to specify the COM port manually in your configuration settings à hardware implementation à Hardware board settings and target hardware resources and “host-board connection”
        - Set Host port COM port to “Manually specify”
        - Check your windows device manager for the correct COM-port
        - Use 115200 as standard baud rate
    - Retry to connect to your Arduino Uno board
    - Did the code generation finish now? If yes, were you able to run the target in external mode? If not, what was the reason for it?
- Let’s try a different approach to connect to your board then. Use the “Inputs/Outputs mode” instead of the “external mode” and try again to connect to your board
- Does it work now? Try to explain the difference between those modes in your report
- Test if you can control the output now with your constant block, changing the value from 0 to 1

4.7 Control an output with an input from your Arduino Uno

- As the connection to your board in “Inputs/Outputs mode” is established, try to control an output pin where an LED is attached with an input pin.
- You need one LED and a resistor for this test
- Connect one free output pin to your LED and resistor
- Use a different input and a connection cable where you can establish a connection to 5 V and 0 V
- Setup your Simulink model to forward the information that you read from the input pin directly to your output pin
- Add a scope as well to be able to track the input signal also in a Simulink scope
- Run the simulation and test your application
    - Leave your connection open (not connected to 5 V or 0 V)
        - What happens? Does your LED start blinking? If so, explain why, if not explain why. (Hint: Check the pin mode in your input block parameters, or check 4.2 for a similar problem when using Matlab to read input values)
    - Connect it to 5 V and check if the LED goes on
    - Connect it to 0 V and check if the LED goes off
- Add a picture of your scope to your Matlab live script or directly export the data to your Matlab workspace for easier plotting

4.8 Control the LED intensity with a potentiometer

- Similar as in section 4.5 we want to control the LED intensity by using a potentiometer
- If you have your hardware setup still built on your breadboard you should be able to reuse it
- Add an “analogue input” block and a “PWM output” block to your Simulink model
- First test your ADC input alone to check if it works correctly and see how what data format is exiting the ADC block (ADC range) (Volts?, Millivolts, precent of maximum range? Etc.)
- Now check if your PWM is working correctly by manually specifying a PWM value using a constant block
- If both blocks working as expected, connect the output from the ADC block to the input of the PWM block
    - Is it possible to directly connect them or is some data editing necessary?
- Try to add some dashboard elements like gauges for visualising the ADC value e.g. or a slider for controlling the PWM value, etc.

4.9 Check the minimal cycle time in your Simulink model

- Remember there was a minimum cycle time when using Matlab that you measured in subtask 4.3 of this training unit
- Let's try to evaluate the same here in your Simulink model
- Reduce the step size in your Solver configuration (= cycle time) to check if lower cycle times are possible, compared to the test results within your Matlab environment
- If so, what is the lowest limit that you could reach with your setup? Is there something like a limit in this mode?
- What is the limiting factor of the cycle time in Simulink "Inputs/Outputs mode"? (baud rate, the Arduino board, Simulink itself, your laptop/PC, etc.?)