المدرسة الوطنية العليا في علم النانو و تكنولوجيا النانو
NATIONAL SCHOOL OF NANOSCIENCE AND NANOTECHNOLOGY

Project Report

BJT Transistor Characterization With An Arduino-Based Device

Mohamed Abdelhadi Bouziani, Ayoub Mazighi

Abstract

We present a simple Arduino-based device capable of characterizing the $I_C(V_{CE})$ curves of a BJT transistor, and an accompanying Julia system for data handling and real-time plotting.

This project was prepared by members of Lithium Club

1 Overview

1 System Composition

Hardware-wise, the system is composed of the Arduino, the circuitry, and of course the user interface.

We used an Arduino UNO R3 in the device to power the circuits with a DC voltage, to collect measurements on the circuit, and to send those measurements to a laptop via serial communication.

The transistor to be characterized is placed in its designated place in the circuit, where it will be driven and measured.

The user interface consists of a potentiometer whose knob can be rotated to adjust the base current, and a button which when pressed starts smoothly varying the collector voltage from the maximum value down to zero.

Regarding software, a simple script is uploaded to the Arduino which makes regular measurements and attempts to send them to the connected laptop over serial communication. We devised an interactive Julia program that receives, treats and plots the data. A secondary Julia program is provided to interpolate and plot the data.

2 Usage

Once launched, the program displays an empty figure on which incoming data is plotted in real-time. Other messages and alerts are displayed through the standard output (in the terminal).

The user would twist the knob to adjust the base current, the program would detect that and begin showing the user the current base current value, until the user is no longer adjusting the base current. It would then set this as the nominal curve value and switch to plotting incoming points. The user could then press the button which would begin a scan of V_{CC} across the possible range of values, realizing the characteristics curve in the process (provided the circuit isn't saturated).

This approach can then be repeated for multiple values of I_B to obtain a more complete characteristic plot. Closing the program saves the data to disk.

After data acquisition, this data can be passed to a secondary program that interpolates it to draw cleaner connected curves, as opposed to the scatter plots generated in real-time by the primary program. The data can also processing manually or a script before passing to the secondary program. One such script is provided in the project's repository.

3 Purpose

The purpose of this project is purely educational. While design choices are well-studied, and the system works fine, the design is rather empirical and manually-driven. This is not supposed to be used at scale, but it could certainly work for pedagogical laboratories at universities. The goal was to obtain clean curves of a transistor that fairly match its datasheet, a task the device achieves adequately.

4 Choices

4.1 Arduino

We chose to build on the Arduino platform due to its accessibility, simplicity and flexibility. We also simply did not have access to a different microcontroller or automatable measurement device at the time. A PIC microcontroller or an SMU unit could work just as well with minimal modifications to the project's software.

4.2 Julia

We chose to write the software in Julia because it is an easy, expressive and performant language, with readily available wrappers for C libraries. This is particularly useful when it comes to functionality such as serial communication, where Julia wrappers around existing battle-tested C libraries prove helpful. Julia also has a great ecosystem and we wish to adopt it for future projects, and this serves as a good starting point.

2 Hardware

1 Circuit Design

The input to the circuit (Figure 2.1) consists of a 5 kΩ potentiometer (bottom of figure), controlling the base voltage V_{BB}, as well as a button (top of figure) which can trigger a scan for the collector voltage V_{CC}.

The potentiometer functions as a voltage divider. Twisting it adjusts the resistance between the V_{DD} and the middle pins, which allows choosing any voltage value between 0 V and V_{DD}.

The button can be clicked to charge the capacitor through the 220 Ω resistor. Once the button is released, the capacitor discharges through the 1 MΩ resistor, creating a slowly varying collector voltage V_{CC}. Furthermore, due to the exponential nature of the discharge, V_{CC} will spend more time varying in the saturation region, providing sufficient time to record data points for the rapidly changing collector current.

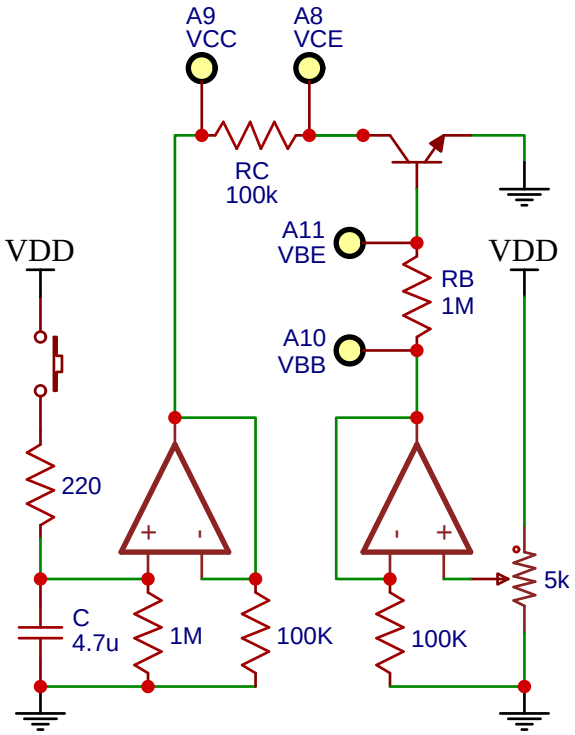

Figure 2.1: The circuit of the device

For the provided values, the time it takes for V_{CC} to go below 0.1 V is $t = RC \log \frac{V_{DD}}{V_{TH}} \approx 18.39$ s. Long enough to record sufficient data points.

Op-amp voltage buffers are used to decouple the input from the transistor, as the transistor currents could load both input circuits and affect their behavior, making the circuit less reliable. This decoupling also allows for separate control of I_C and I_B, without a change in either voltage affecting the other.

The necessary voltages are measured at four points indicated in the circuit alongside the corresponding Arduino analog pins used to measure.

2 Breadboards

We assembled a prototype of the device on breadboards. Figure 2.2 contains a picture of the assembled circuit.

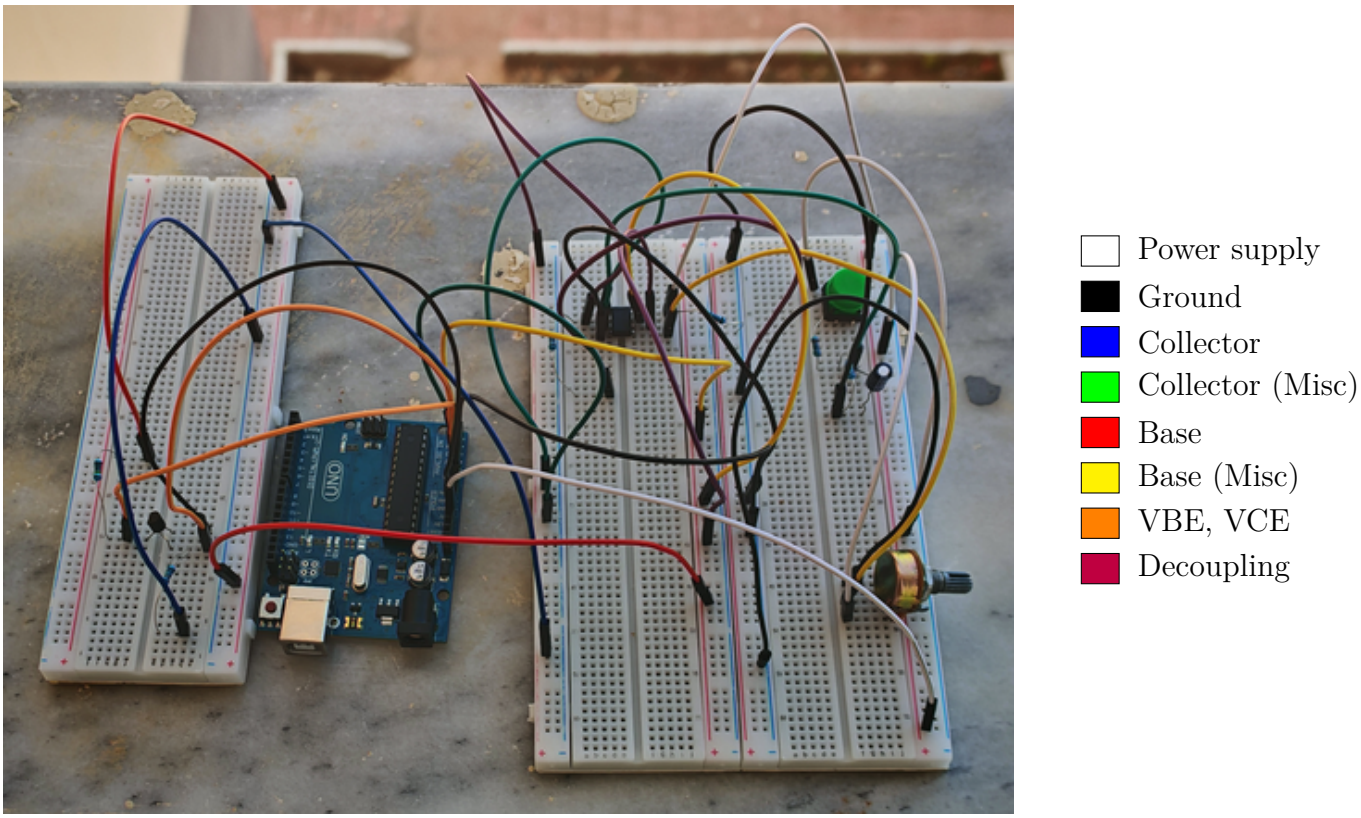

Figure 2.2: The circuit assembled on breadboards alongside the color-coding key

Wires are color-coded for readability, the key is provided next to the figure. The potentiometer and the button are present on the right of the picture.

3 Measurements

We use Arduino UNO's built-in 10-bit ADC to make measurements. For the software to function, we need to measure and stream four voltages: V_{CC}, V_{BB}, V_{BE}, and V_{CE}. The first two are used to determine the current state of the circuit (whether the user is varying base current, whether the capacitor is currently discharging), while the others are used to determine currents:

$$I_C = \frac{V_{CC} - V_{CE}}{R_C} \qquad I_B = \frac{V_{BB} - V_{BE}}{R_B}$$

Cables connect the four required points to the Arduino board's analog pins (A0, A1, A2, A3) to make the measurements. This is further discussed in the software part of the report.

One important consideration when operating the device is measurement noise. We found Arduino measurements to be quite susceptible to noise. We identified the following noise-contributing factors:

- **Power noise**: Powering the Arduino via an AC-to-DC power supply (even the common lab-grade ones used at pedagogical laboratories) introduces some sinusoidal noise into the

measurements. Even when powering the Arduino via a USB cable connected to a laptop, there is significant measurement noise when the laptop is connected to the charger versus when it is not.

- **Electromagnetic noise**: Operating the circuit near devices that generate significant electric or magnetic fields seems to influence the measurements. This includes devices such as fridges, power cables (chargers, wall plugs, ...etc), and AC units. We found significant noise reduction when moving away from such devices.
- **Cable and breadboard quality**: Loose cables, hot resistors, and low-quality breadboards should be avoided.

Overall, we were able to operate with insignificant noise when powering the circuit via an unplugged laptop (running on battery), with good breadboards and cable connections, well away from any electronic apparatus (in the middle of nowhere XD).

3 Software

The full code for the project is available on GitHub.

1 Arduino

A simple script in Arduino C is uploaded to the Arduino board. It simply makes measurements at regular time intervals and sends them over a serial communication channel established over USB with a 9600 baud rate. The script is provided below (and can be found on GitHub):

```
const int PIN_VCE = A0;
const int PIN_VBE = A1;
const int PIN_VBB = A2;
const int PIN_VCC = A3;

int VCE, IC, IB, VBB, VCC;

void setup() {
        delay(5000); Serial.begin(9600);
        while (!Serial);;
}

void loop() {
        VCC = analogRead(PIN_VCC);
        VCE = analogRead(PIN_VCE);
        VBB = analogRead(PIN_VBB);

        IC = VCC - VCE;
        IB = VBB - analogRead(PIN_VBE);

        Serial.print(VCE); Serial.print(",");
        Serial.print(IC); Serial.print(",");
        Serial.print(IB); Serial.print(",");
        Serial.print(VBB); Serial.print(",");
        Serial.println(VCC);

        delay(100);
}
```

2 Primary Program

2.1 Plotting (plot.jl)

We use the Makie.jl library and its Observable API to plot data in real-time. Data points are saved in the form of CSV files in the following format:

- A row for the nominal I_B value for the curve
- A row for V_{CE} values
- A row for I_C values

This structure with matching columns for data points is repeated interleaved with empty rows. Saving the data to disk occurs on auto-saves and on exiting.

2.2 Communication (serial.jl)

We use the LibSerialPort.jl (a Julia wrapper for libserialport) to establish serial communication and receive data from the Arduino board. The software automatically chooses the only device connected or prompts the user select one if multiple are connected.

2.3 Updating logic (update.jl)

This is where the bulk of the program resides. We use an update-based state machine with two states: Receiving points, and varying base current.

When receiving points, the program adds new points to the graph while observing changes in the other measurements. If I_B is detected as changing more than is tolerated by noise, and V_{CC} is not currently changing, the program transitions to the other state: Waiting for I_B to stabilize.

While in the state of waiting for I_B, the program assumes the user is adjusting it. Once the most recent I_B measurements have a standard deviation less than the maximum tolerated, I_B is considered stable and the program transitions back to the main state.

The behavior of this state machine is dependent on careful configuration of various constants and thresholds, such as: Tolerated noise levels, tolerated standard deviation, recent points to consider, minimum deviation from equilibrium considered changing, ...etc.

Some additional niceties are backed into the program to make for a nicer UX. For example, the program will not switch to a new nominal I_B value and start in a new curve if it detects the base current has only changed within a certain tolerated region of the previous value.

An auxiliary starting state is used to kickstart the program with preliminary measurements.

3 Secondary program

A small script can read the output data and interpolate it to obtain smoother curves, as opposed to the scatter plots obtained in real-time by the primary program.

4 Conclusion

4 Results Sample

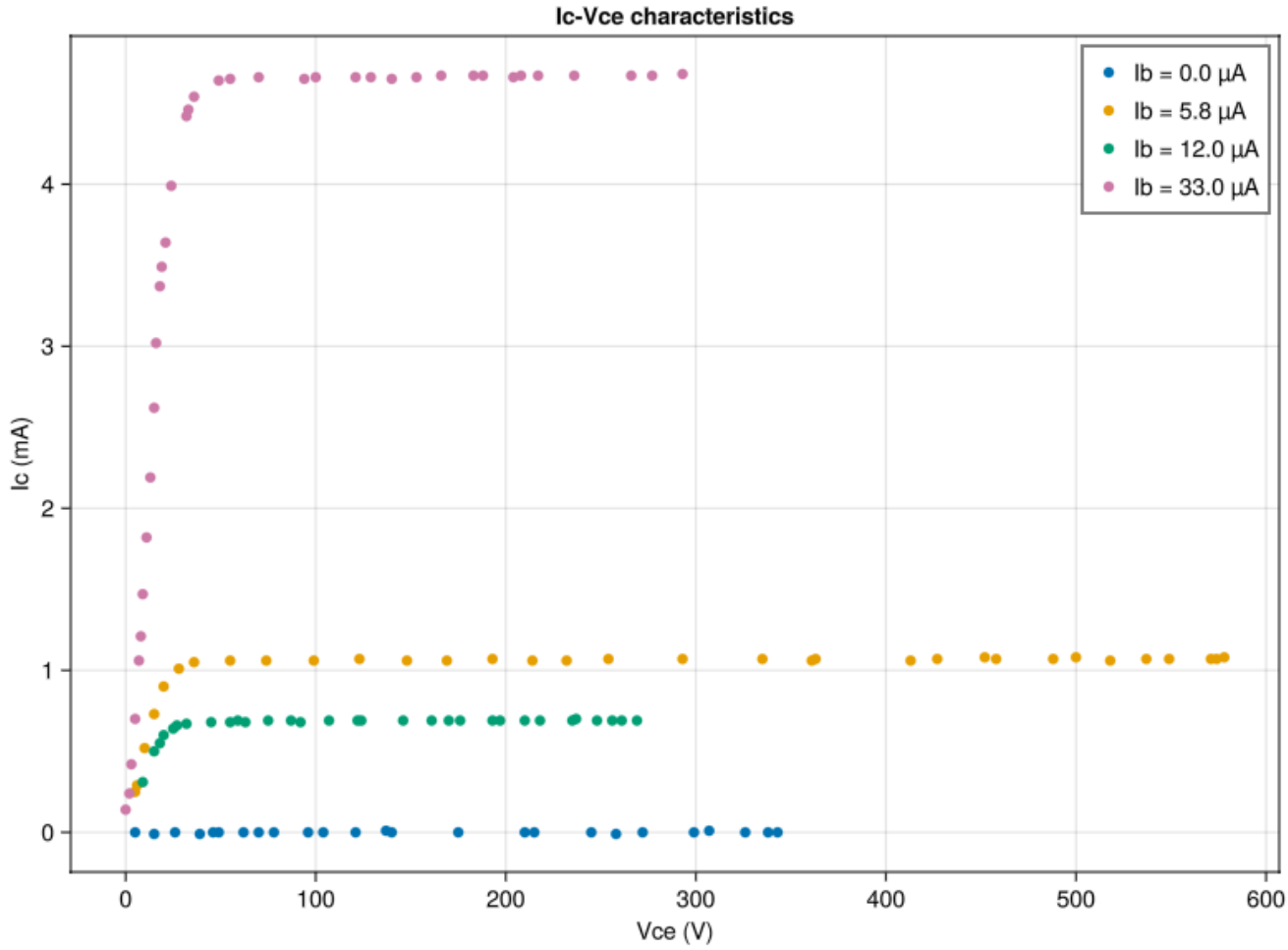

Figure 4.1: A scatter plot of the data

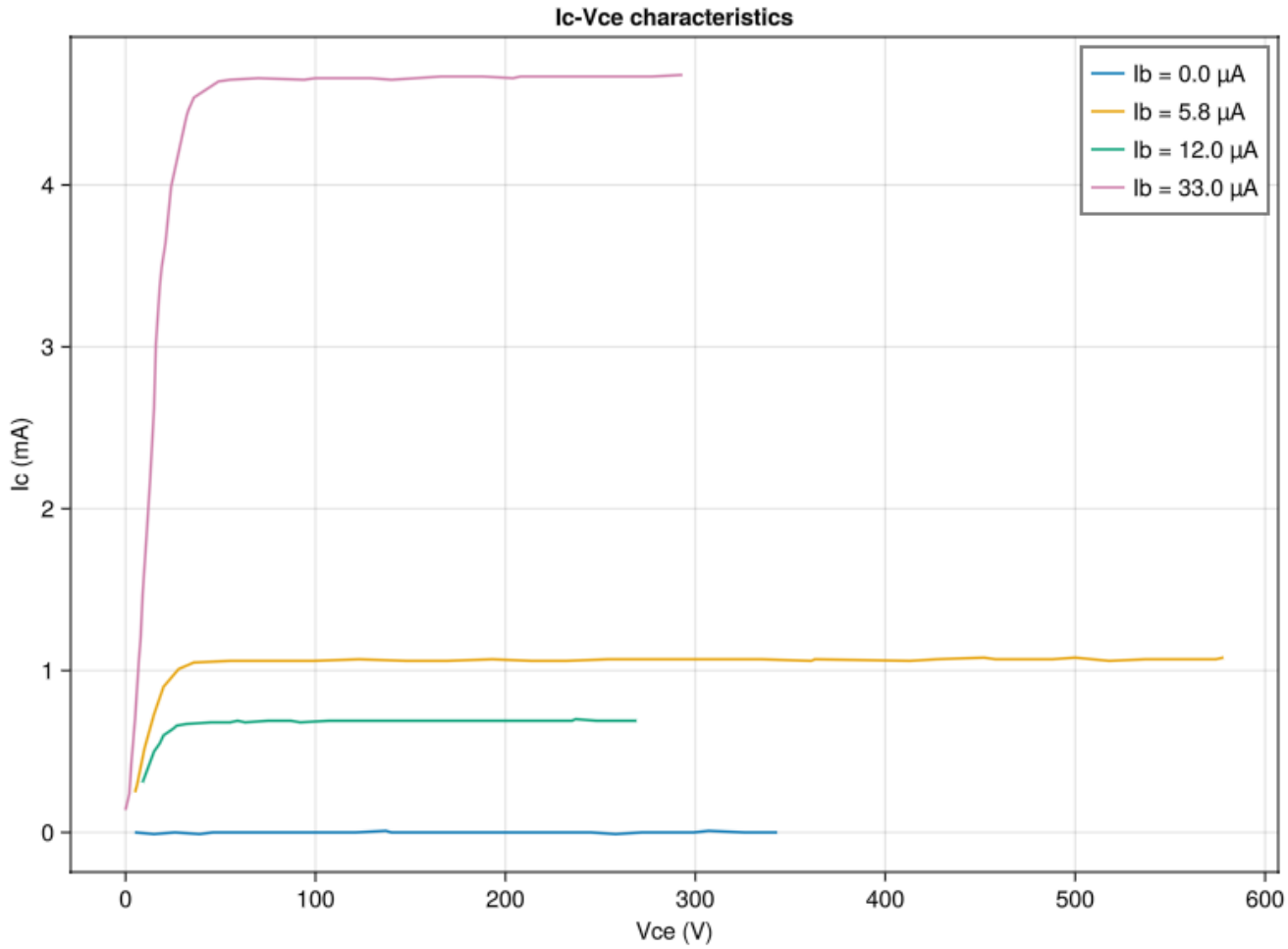

Figure 4.2: An interpolated plot of the data

5 Closing Thoughts

The project was quite fun to work on for us. There are many ways to go about improving this device: fully automatic sweeping, better noise handling (shielding/insulation), a more graphical UI, a more compact build, ...etc. Perhaps this will be revisited in the future.

Ultimately, this was quite exploratory and fully driven by intuition.