TRANSDUCERS & INSTRUMENTATION LAB (AUGUST 2026)

Lab 01
Getting Started with Microcontroller Interfacing

AIM

The aim of this lab is to learn to program a Teensy 4.1 microcontroller and interface it with external circuits, through a series of lab exercises.

LEARNING OUTCOMES

By the end of this lab, you should be able to:

1. Describe the pin types and peripherals available on the Teensy 4.1 microcontroller.
2. Write programs to configure and use digital pins.
3. Evaluate the timing accuracy of signals generated by the microcontroller.
4. Acquire an analog signal using the on-board analog-to-digital converter.
5. Demonstrate the continuous control of an LED's intensity using pulse width modulation.

OUTLINE OF THE SESSION

The experiments are to be performed in the order listed, as each depends upon the circuit and the concepts established in the preceding one. Warnings relating to the electrical limits of the board are given within the relevant experiment and must be observed before the corresponding circuit is connected.

1 Background

A microcontroller is a single chip containing all the essential parts of a computer. It has a processor core that executes instructions, non-volatile program memory (flash), which retains the program when power is removed, random-access memory (RAM) for values while the program

runs, and peripherals that connect it to the outside world — digital pins, an analog-to-digital converter, and serial ports — along with internal peripherals such as timers. Rather than serving as a general-purpose computer, a microcontroller is designed to be embedded within a larger device, where it operates without a display or keyboard of its own. A microprocessor, by contrast, contains only the processor core; memory and I/O must be provided by separate chips.

The program written by the user is stored in flash memory and is retained when the board is powered off. On power-up, the processor reads these instructions in sequence and executes them, using RAM to hold the variables created while the program runs. The microcontroller interacts with the outside world solely through its pins: it senses by reading the voltage applied to a pin, and it acts by setting the voltage that a pin produces.

1.1 The Teensy 4.1 board and its pins

The Teensy 4.1 is a development board built around the NXP i.MX RT1062 microcontroller, which uses an ARM Cortex-M7 processor running at 600 MHz with hardware support for floating-point arithmetic. It provides 8 MB of flash for program storage, 1 MB of RAM, and 4 KB of emulated EEPROM. The board exposes 55 digital input/output pins, of which 42 are accessible along the two rows of breadboard-compatible headers, 35 can generate PWM, and 18 can be used as analog inputs. It also provides 8 serial ports, 3 SPI and 3 I2C interfaces, 3 CAN bus interfaces, a native SD card socket, a USB host port, and a 10/100 Mbit Ethernet interface (which requires an external magjack module). All logic operates at 3.3 V, and the board is powered either over USB or through the VIN pin.

Table 1 summarises the categories.

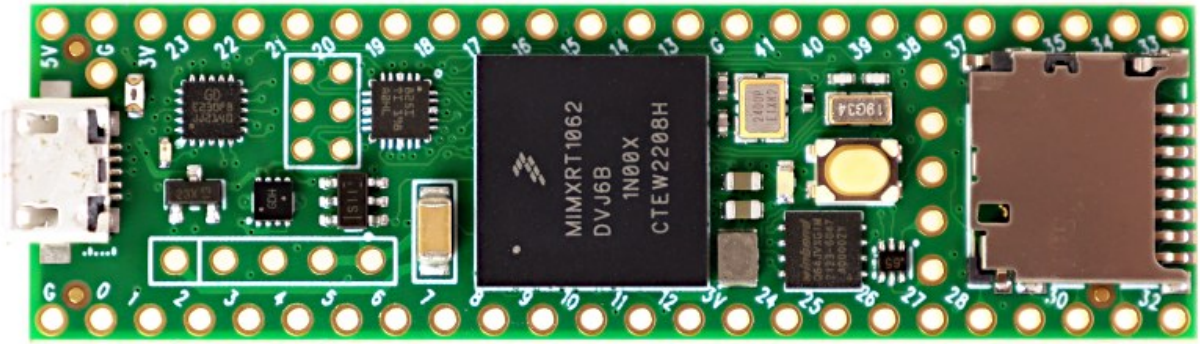

Figure 1: Pin assignment of the Teensy 4.1.

> **TIP**
>
> Keep the PJRC pinout card open while you work (`https://www.pjrc.com/teensy/pinout.html`). The pin numbers printed on the board are the digital pin numbers used by `digitalWrite` and `digitalRead`; the analog designations `A0`–`A17` are a second labelling of a subset of those same pins.

1.2 Programming the Teensy with the Arduino IDE

A program written in the Arduino IDE, known as a *sketch*, is source code and cannot be executed by the microcontroller directly. When the sketch is uploaded, the IDE first compiles it into machine code for the ARM Cortex-M7 processor and produces a binary image of the program.

Table 1: Categories of pin available on the Teensy 4.1.

Category	Designation	Remarks
Power	`VIN`, `GND`, `3V3`, `VBAT`, `VUSB`	`VIN` accepts 3.6 to 5.5 V. `3V3` is an *output*, not an input, and can supply about 250 mA in total. `VUSB` is connected to the USB 5 V supply; it must not be driven from an external source while USB is connected.
Digital I/O	0 to 54	Each may be configured as `OUTPUT`, `INPUT`, `INPUT_PULLUP`, `INPUT_PULLDOWN`, or `OUTPUT_OPENDRAIN`. Every one of them can raise an interrupt.
PWM	35 of the digital pins	Driven by hardware timers through `analogWrite`, so the waveform continues without processor involvement.
Analog input	18 pins, `A0`–`A17`	12-bit ADC over 0 to 3.3 V. `analogRead` returns 0 to 1023 by default; call `analogReadResolution(12)` for 0 to 4095. `A0`–`A9` correspond to digital pins 14 to 23.
Analog output	none	The Teensy 4.1 has **no** digital-to-analog converter. `analogWrite` produces a PWM square wave, not a voltage.
Serial (UART)	8 ports, `Serial1`–`Serial8`	Separate from `Serial`, which is the USB connection to the workstation.
I^2C	3 buses: `Wire`, `Wire1`, `Wire2`	Each is an `SDA`/`SCL` pair. Used by most digital MEMS sensors.
SPI	3 buses	`MOSI`, `MISO`, `SCK`, and a chip-select (CS) line per device.
CAN	3 controllers	Two CAN 2.0B and one CAN FD.
Dedicated	SD socket, USB host, Ethernet	Fixed hardware; the associated pins are not available for general use. Ethernet additionally requires an external magjack module.

This image is then transferred over the USB connection to a second, smaller microcontroller on the Teensy board, which serves as its *bootloader*. The bootloader erases the existing contents of the flash memory, writes the new image into it, and restarts the main processor, which then begins executing the program from the start. Because the program resides in non-volatile flash memory, it is retained when the board is disconnected and runs again automatically whenever power is applied. Every sketch contains two functions: `setup()`, which is executed once when the board is powered or reset and is used to configure the pins and peripherals, and `loop()`, which is executed repeatedly thereafter for as long as the board remains powered.

1.3 Setup

1. Install the Arduino IDE, version 2.0.4 or later.
2. Open **File** → **Preferences** and enter the following address under *Additional boards manager URLs*: `https://www.pjrc.com/teensy/package_teensy_index.json`
3. Open **Boards Manager**, search for *Teensy*, and install the package.
4. Select **Tools** → **Board** → **Teensy** → **Teensy 4.1**, and set **Tools** → **USB Type** to *Serial*.
5. Connect the board to the workstation using a USB cable. Uploading a sketch opens the Teensy Loader window, which transfers the program automatically. If the board does not respond, press the white **Program** button on the board and upload again.

2 Apparatus and components

APPARATUS AND COMPONENTS

Qty	Item	Specification
1	Teensy 4.1 development board	with micro-USB cable
1	Solderless breadboard	—
2	LED	—
2	Resistor	330 Ω, 5 %
1	Push-button switch	tactile, SPST, breadboard mount
1	Function generator	sine output, adjustable DC offset
1	Digital storage oscilloscope	≥ 50 MHz, 2 channels
1	Digital multimeter	—
–	Jumper wires	assorted

3 Experiment 1 — Lighting an LED from a digital pin

3.1 Objective

To light an LED from a digital output pin and verify that the current drawn remains within the pin's rating.

3.2 Circuit

Connect the anode of the LED (the longer lead) to digital pin 2 through the 330 Ω series resistor, and the cathode to GND. With a forward voltage of about 2 V for a red LED, verify whether the current

$$I_{\mathrm{F}} = \frac{V_{\mathrm{OH}} - V_{\mathrm{F}}}{R} = \frac{3.3\,\mathrm{V} - 2.0\,\mathrm{V}}{330\,\Omega} \approx 3.9\,\mathrm{mA}, \tag{1}$$

is within the pin's recommended maximum of 4 mA.

3.3 Observations

Write a sketch that configures pin 2 as an output and sets it `HIGH`, and a second version that sets it `LOW`. For each, measure the voltage across the LED and across the resistor, and record them in Table 2.

Table 2: Voltages measured in the LED branch.

Pin state	V across LED (V)	V across R (V)	I_{F} (mA)
`HIGH`			
`LOW`			

Calculate I_{F} from the measured voltage across the resistor and compare it with Equation (1). Forward voltage varies between individual LEDs, so a measured current between 3 and 5 mA is expected. Identify the sources of the difference between the measured and calculated current.

> **Warning**
>
> The recommended maximum output current from a pin on the Teensy 4.1 is 4 mA. An LED connected directly between a pin and GND, without a series resistor, draws roughly ten times this value and will damage the pin over time, even though it appears to work.

4 Experiment 2 — Blinking an LED using `delay()`

4.1 Objective

To generate a periodic signal on a digital pin using `delay()`, and to compare the resulting period with the value set in firmware.

4.2 Observations

Statements placed in `loop()` are executed repeatedly for as long as the board is powered. The function `delay(n)` halts execution for `n` milliseconds before the next statement is reached.

Write a sketch that sets pin 2 `HIGH`, waits, sets it `LOW`, waits again, and repeats. Using a delay of 500 ms, confirm that the LED blinks once per second.

Reduce the delay in stages, and for each value measure the period of the waveform on the oscilloscope. Record the results in Table 3.

Table 3: Commanded and measured blink periods.

Delay (ms)	Expected period (ms)	Measured period (ms)	Error (%)
500			
50			
5			
1			

The expected period is twice the delay, since each cycle contains two waits. State whether the error is constant in absolute terms or as a percentage, and use this to explain where the extra time comes from.

5 Experiment 3 — Controlling brightness using PWM

5.1 Objective

To vary the brightness of an LED by pulse-width modulation, and to relate the duty cycle of the waveform to the average voltage delivered to the load.

5.2 Experiment 3A — Duty cycle and average voltage

The Teensy 4.1 has no digital-to-analog converter, and a digital pin can produce only 0 V or 3.3 V. An intermediate average is obtained instead by switching the pin rapidly between the two states and varying the fraction of each cycle for which it is high. This fraction is the duty cycle D, and the average voltage is

$$V_{\text{avg}} = D\,V_{\text{OH}}. \tag{2}$$

The function `analogWrite(pin, value)` generates such a waveform, where `value` ranges from 0 to 255 by default, corresponding to duty cycles from 0 % to 100 %.

Write a sketch that applies to pin 2 each of the duty cycles listed in Table 4, calculating in each case the value to be passed to `analogWrite`. For each, observe the waveform on the oscilloscope, measure its duty cycle, and measure the average voltage at the pin using the multimeter in DC mode. Record the frequency of the waveform, which is unchanged throughout.

Compare the measured average voltage with Equation (2). Note that the voltage at the pin at any instant is never equal to this average, and state why the LED nevertheless appears to change brightness rather than flicker. Note also that not every duty cycle can be commanded exactly, since only whole values may be passed to `analogWrite`; state the resulting error for each entry in the table.

Table 4: Commanded and measured duty cycle and average voltage.

Required D (%)	`analogWrite` value	Measured D (%)	Measured V_{avg} (V)	$D\,V_{OH}$ (V)
0				
25				
50				
75				
100				

5.3 Experiment 3B — Generating the waveform in software

The same waveform can be produced without `analogWrite`, by setting the pin high and low directly and controlling the time spent in each state. Write a sketch that uses `digitalWrite()` together with `delayMicroseconds()` to generate a duty cycle of 50 % at the frequency measured above, and confirm on the oscilloscope that the waveform matches. Milliseconds are too coarse for this purpose and produce visible flicker.

5.4 Experiment 3C — Hardware versus software generation

Now add a `Serial.println()` statement inside the loop and observe the LED and the waveform again. Repeat the same addition to the sketch that uses `analogWrite`. Explain the difference in behaviour with reference to where the waveform is generated in each case.

5.5 Experiment 3D — Carrier frequency and flicker

The frequency of the waveform may be changed with `analogWriteFrequency(pin, f)`, called before `analogWrite`. Set the duty cycle to 50 % and vary the frequency, confirming each value on the oscilloscope. Reduce it in stages and record the frequency below which the LED is seen to flicker rather than to appear continuously lit. Comment on why a higher frequency is required when the same technique is used to drive a motor or a loudspeaker.

6 Experiment 4 — Reading a switch with a digital input

6.1 Objective

To read the state of a push-button using a digital input pin, to observe the behaviour of a floating input, and to use the internal pull-up resistor to define its state.

6.2 Circuit

Connect one terminal of the push-button to digital pin `3` and the other to `GND`. The LED circuit of Experiment 1 is retained.

6.3 Observations

A digital input does not measure voltage. It compares the voltage at the pin against an internal threshold and reports one of two states, `HIGH` or `LOW`, which is returned by `digitalRead(pin)`.

With the circuit above and the pin configured using `pinMode(3, INPUT)`, the pin is connected to `GND` while the button is pressed, but is connected to nothing at all while it is released. Such a pin is said to be *floating*: its voltage is determined by stray charge and interference rather than by the circuit.

Write a sketch that configures pin 3 as `INPUT` and prints `digitalRead(3)` continuously to the serial monitor. Observe the voltage at the pin on the oscilloscope while the button is released, and again while a finger is brought near the connecting wire.

Change the configuration to `pinMode(3, INPUT_PULLUP)`, which connects an internal resistor between the pin and 3.3 V, and repeat the observations. Record the results in Table 5.

Table 5: Input state and pin voltage with and without the internal pull-up.

Configuration	Button released	V at pin (V)	Button pressed	V at pin (V)
`INPUT`				
`INPUT_PULLUP`				

Finally, extend the sketch so that the LED on pin 2 lights only while the button is pressed. Note that with the pull-up enabled, a pressed button returns `LOW` rather than `HIGH`, and explain why.

7 Experiment 5 — Blocking and non-blocking timing

7.1 Objective

To demonstrate that `delay()` prevents a program from responding to an input, and to implement the same timing without blocking, using `millis()`.

7.2 Circuit

The button of Experiment 4 on pin 3 is retained. Add the second LED and 330 Ω resistor between pin 4 and `GND`, wired as in Experiment 1.

7.3 Observations

Write a sketch that blinks the LED on pin 2 once per second using `delay(500)`, and which, in the same loop, reads the button and lights the LED on pin 4 whenever the button is pressed.

Press the button briefly at random intervals and record how many presses out of twenty are registered.

The function `millis()` returns the number of milliseconds elapsed since the board was powered, obtained from a hardware timer that runs independently of the processor. Rather than waiting, the program may record the time at which the LED was last changed and test on each pass

through the loop whether the required interval has elapsed:

$$\texttt{millis()} - t_{\text{last}} \geq 500\,\text{ms}.$$

Rewrite the sketch so that the blink is produced in this way and no `delay()` remains. Repeat the twenty presses and record the result in Table 6.

Table 6: Response to button presses with blocking and non-blocking timing.

Implementation	Presses registered (of 20)	Hold time required (ms)
`delay()`		
`millis()`		

8 Experiment 6 — Detecting a button press using an interrupt

8.1 Objective

To detect a change on a digital pin using a hardware interrupt rather than by repeated reading, and to observe the contact bounce of a mechanical switch.

8.2 Circuit

The circuit of Experiment 4 is retained, with the push-button on pin 3 configured as `INPUT_PULLUP`. Connect the oscilloscope probe to pin 3.

8.3 Experiment 6A — Counting presses and observing contact bounce

In the previous experiments the state of the button was found by reading the pin repeatedly. The processor may instead be interrupted by the pin itself: the statement

```
attachInterrupt(digitalPinToInterrupt(3), countPress, FALLING);
```

causes the function `countPress` to be executed whenever the voltage at pin 3 falls from `HIGH` to `LOW`, whatever the program is doing at that moment. Such a function is called an interrupt service routine. It must complete quickly, and any variable it shares with the main program must be declared `volatile`.

Write a sketch in which the interrupt service routine increments a counter, and `loop()` prints the counter once per second. Press the button exactly twenty times and record the value reported.

Table 7: Presses recorded for twenty actuations of the switch.

Method	Count recorded
Interrupt, no additional measures	
Interrupt, with presses ignored for 20 ms after each	

The count obtained is generally greater than twenty. Set the oscilloscope to trigger once on a falling edge at pin 3, with a timebase of a few milliseconds per division, and capture a single

press. Sketch or record the waveform obtained, and measure the interval over which the contacts continue to make and break.

Modify the interrupt service routine so that a press occurring within 20 ms of the previous one is ignored, using `millis()`, and repeat the count.

8.4 Experiment 6B — Interrupts during a blocking delay

Add a blinking LED to `loop()` using `delay(500)`, as in Experiment 5, and confirm that presses are still counted correctly. Explain why this is so, with reference to where the counting now takes place.

9 Experiment 7 — Acquiring an analog signal

9.1 Objective

To acquire a time-varying voltage using the on-board analog-to-digital converter, and to convert the resulting counts into a voltage.

9.2 Circuit

Connect the output of the function generator to analog pin `A0` and its ground to `GND`. Connect the oscilloscope across the same two points so that the applied signal can be monitored throughout.

Warning

The analog pins accept voltages between 0 V and 3.3 V only. A negative voltage, or one above 3.3 V, will damage the input immediately.
Before connecting the generator to the board, set its amplitude and DC offset and verify on the oscilloscope that the entire waveform lies within 0 V and 3.3 V. Set the generator's output load to High Z. If it is set to 50 Ω, the actual output is twice the displayed amplitude, and the signal will exceed 3.3 V.

9.3 Experiment 7A — Reading the waveform and converting counts

An analog input does not return a voltage. The analog-to-digital converter compares the applied voltage against its reference of 3.3 V and returns a count, which `analogRead(pin)` reports in the range 0 to 1023 by default. The corresponding voltage is

$$V = \frac{n}{1023} \times 3.3\,\mathrm{V}, \tag{3}$$

so that each count corresponds to approximately 3.2 mV.

Set the generator to produce a sine wave of 1 Hz with an amplitude of 1 V peak-to-peak and a DC offset of 1.65 V, and verify the waveform on the oscilloscope before connecting it to the board.

Write a sketch that reads `A0` and prints the value to the serial monitor continuously. Record the maximum and minimum counts observed, convert them to voltages using Equation (3), and compare them with the values measured on the oscilloscope. Enter the results in Table 8.

Table 8: Signal amplitude measured by the ADC and by the oscilloscope.

Quantity	Count	Voltage from Eq. (3)	Oscilloscope
Maximum			
Minimum			
Peak-to-peak			

The resolution of the converter may be raised to 12 bits by calling `analogReadResolution(12)` in `setup()`, after which `analogRead` returns values in the range 0 to 4095 and Equation (3) must be modified accordingly. Repeat the measurement of the maximum and minimum values at this resolution.

9.4 Experiment 7B — Sampling rate and apparent frequency

Increase the frequency of the generator in stages and observe the values printed. State the frequency above which the apparent frequency of the printed data no longer matches that set on the generator, and explain the observation in terms of the rate at which the sketch takes and transmits samples.

References

[1] PJRC, *Teensy 4.1 Development Board*. https://www.pjrc.com/store/teensy41.html

[2] PJRC, *Teensyduino Reference: Digital I/O and Timing*. https://www.pjrc.com/teensy/td_digital.html

[3] NXP Semiconductors, *i.MX RT1060 Processor Reference Manual*.