Custom Element Language Reference

The language reference for custom elements on iCircuit

Krueger Systems, Inc.

2025-07-25

Contents

Introduction

The Custom Elements programming language is designed to expand iCircuit Application functionality, this allows users to build and simulate own created custom circuits along with their interactions with other components. With these tools, developers can define component interfaces, manage digital and analog inputs and outputs, and establish parameters to enhance the performance and behavior of their circuits.

Also, the Custom Elements programming language is designed to be simple and easy to use, with a syntax based on the Arduino Framework. This makes it easy for developers to create custom circuits and components that can be easily integrated into their projects.

Main Functions

setup()

[Main Functions]

```
void setup();
```

Description

The `setup()` function is called when a C program starts. As the Arduino equivalent use it to initialize variables, pin modes, initialize modules, draw symbols, etc. This function will only run once, after each program change or reset of the custom element.

Parameters

None.

Returns

Nothing.

Example Code

```
void setup() {
    // Set the chip name "74LS04" (Hex inverting gates)
    chipName("74LS04");
    logicFamily(TTL); // Specify TTL logic family

    // Define left side input and output pins
    pin(1, "1A");
    pin(2, "Y1", OUTPUT);
    pin(3, "2A");
    pin(4, "Y2", OUTPUT);
    pin(5, "3A");
    pin(6, "Y3", OUTPUT);

    // Define the ground pin
    pin(7, "GND", GROUND);

    // Define the right side input and output pins
    pin(8, "Y4", OUTPUT);
    pin(9, "4A");
    pin(10, "Y5", OUTPUT);
    pin(11, "5A");
    pin(12, "Y6", OUTPUT);
```

```
    pin(13, "6A");

    // Define the power pin
    pin(14, "VCC", POWER);
}

void loop() {}
```

The drawing defined here will be immutable, it will not change during the simulation. If you need to change the drawing during the simulation, use the `loop()` function.

loop()

[Main Functions]

```
void loop();
```

Description

The `loop()` function is called repeatedly by the simulation engine. It loops continuously, allowing your program to change the state of the pins, respond to external signals, and draw on the chip. Use it to implement the logic of your custom element.

Parameters

None.

Returns

Nothing.

Example Code

```
void setup() {
    // Set the chip name "74LS04" (Hex inverting gates)
    chipName("74LS04");
    logicFamily(TTL); // Specify TTL logic family

    // Define left side input and output pins
    pin(1, "1A");
    pin(2, "Y1", OUTPUT);
    pin(3, "2A");
    pin(4, "Y2", OUTPUT);
    pin(5, "3A");
    pin(6, "Y3", OUTPUT);

    // Define the ground pin
    pin(7, "GND", GROUND);

    // Define the right side input and output pins
    pin(8, "Y4", OUTPUT);
    pin(9, "4A");
    pin(10, "Y5", OUTPUT);
    pin(11, "5A");
    pin(12, "Y6", OUTPUT);
    pin(13, "6A");

    // Define the power pin
```

```
    pin(14, "VCC", POWER);
}

void loop() {
    // Write the negation of the first input pin to the first output pin
    bool a1 = digitalRead( 1);
    digitalWrite(2, !a1);

    // Write the negation of the second input pin to the second output pin
    bool a2 = digitalRead( 3);
    digitalWrite(4, !a2);

    // Write the negation of the third input pin to the third output pin
    bool a3 = digitalRead( 5);
    digitalWrite(6, !a3);

    // Write the negation of the fourth input pin to the fourth output pin
    bool a4 = digitalRead( 9);
    digitalWrite(8, !a4);

    // Write the negation of the fifth input pin to the fifth output pin
    bool a5 = digitalRead( 11);
    digitalWrite(10, !a5);

    // Write the negation of the sixth input pin to the sixth output pin
    bool a6 = digitalRead( 13);
    digitalWrite(12, !a6);
}
```

Adding drawing code inside the loop function will lead to a flickering effect.

draw()

[Main Functions]

```
void draw();
```

Description

The `draw()` function does precisely what its name suggests: it draws on the chip. This function is called repeatedly after power-up the chip, allowing your program to change the appearance of the chip, show the internal state of the chip, draw symbols, or even animate the chip.

Parameters

None.

Returns

Nothing.

Example Code

The code updates the color of a graphic element based on the state of the logical input `a1`. If `a1` is false, the `fillTriangle()` function will display the color green, and if it is true, the color red will be displayed.

```
// Define the static variable a1
static bool a1;

void setup() {

    chipName("Inverter");

    // Define the input pin
    pin(1, "1A");
    // Define the power pin
    pin(2, "VCC", POWER);
    // Define ground pin
    pin(3, "GND", GROUND);
    // Define the output pin
    pin(4, "1Y", OUTPUT);

}

// Define the loop function
void loop() {
    // Write the negation of the input pin to the output pin
```

```
    a1 = digitalRead(1);
    digitalWrite(4, !a1);
}

// Define the color constants
#define RED 255, 0, 0
#define GREEN 0, 255, 0

// Define the draw function
void draw(){

    // Draw the not gate symbol
    int w = chipWidth();   // gets the width of the chip
    int h = chipHeight(); // gets the height of the chip
    int middle_w = w / 2;
    int middle_h = h / 2;

    drawTriangle(middle_h - 10 ,middle_w - 10,middle_h - 10, middle_w + 10,
↪   middle_h + 10, middle_w);
    drawLine(middle_h - 20, middle_w , middle_h -10, middle_w);
    drawLine(middle_h + 18, middle_w, middle_h +24, middle_w);
    drawCircle(middle_h + 14, middle_w, 4);

    // Set color based on a1 value
    if (!a1)
        setColor(GREEN);
    else
        setColor(RED);

    // Draw Fill
    fillTriangle(middle_h - 9 ,middle_w - 9,middle_h - 9, middle_w + 9,
↪   middle_h + 8, middle_w);

    // Reset the color
    setDefaultColor();
}
```

reset()

[Main Functions]

```
void reset();
```

Description

The `reset()` function will be called all the time when the chip is powered down, use it to restore the initial state of the chip, since CLanguage won't erase the memory when the chip is powered down.

Parameters

None.

Returns

Nothing.

Example Code

```
void setup() {
    chipName("74HC163"); // Set the chip name
    logicFamily(TTL); // Define the logic family as TTL

    pin(1, "CLR", INPUT, LINE_OVER | CLOCK); // Clear pin (CLR), input with
↪  line over and clock

    // Parallel data input pins
    pin(3, "A");
    pin(4, "B");
    pin(5, "C");
    pin(6, "D");

    pin(7, "ENP"); // Enable Parallel (count enable)
    pin(8, "GND", GROUND); // Ground pin

    pin(9, "LOAD", INPUT, LINE_OVER); // Parallel load pin, input with line
↪  over
    pin(10, "ENT"); // Enable T (serial count enable)

    // Output pins for counter bits
    pin(11, "QD", OUTPUT);
    pin(12, "QC", OUTPUT);
    pin(13, "QB", OUTPUT);
    pin(14, "QA", OUTPUT);
```

```
    pin(15, "RC0", OUTPUT); // Ripple Carry output
    pin(16, "VCC", POWER); // Power supply pin
}

// Control variables
bool lastCp = false; // Previous state of the clock signal
int count = 0; // Internal counter

void loop() {
    bool cp = digitalRead(2); // Read the clock signal from pin 2

    // Detects a rising edge in the clock signal
    if (!lastCp && cp) {
        if (!digitalRead(1)) // If CLR pin is low, reset the counter
            count = 0;
        else if (!digitalRead(9)) // If LOAD pin is low, load values from
        ↪  A-D into the counter
            count = digitalReadNibble(3, 4, 5, 6);
        else if (digitalRead(7) && digitalRead(10)) // If ENP and ENT are
        ↪  high, increment the counter
            count = (count + 1) & 0x1F; // Ensure the counter does not
↪   exceed the limit
    }

    // Write the counter value to outputs QA-QD
    digitalWriteNibble(14, 13, 12, 11, count);

    // Compute the ripple carry output (RC0)
    bool val = !(!digitalRead(11) && !digitalRead(12) && !digitalRead(13)
    ↪  && !digitalRead(14) && !digitalRead(10));
    digitalWrite(15, val); // Write the carry output to RC0 pin

    lastCp = cp; // Store the current clock state for the next iteration
}

void reset() {
    // Reset the chip state
    lastCp = false;
    count = 0;
}
```

Circuit Interface

chipName()

[Circuit Interface]

```
void chipName(const char *name);
```

Description

Defines the name for the custom element.

Parameters

name: the string you want to assign as name.

Returns

Nothing.

Example Code

Sets chip name to “IC7447”.

```
void setup() {
   chipName("IC7447");
}

void loop() {}
```

This function can work properly only in the setup context, calling it in the loop context won’t result in a chip name change.

pin()

[Circuit Interface]

```
void pin (unsigned short pin, const char* label, unsigned short mode,
↪  unsigned int flags);
```

Description

Creates a pin and allows you to configure its label, mode, position, and appearance.

It is possible to enable pull-up resistors with the mode `INPUT_PULLUP`. Additionally, alternative output modes are available for all pins, such as `THREE_STATE`, `OPEN_DRAIN`, and `OPEN_COLLECTOR`.

Also, it is possible to set the position of the pin on the chip as a flag, which can be `LEFT`, `RIGHT`, `TOP`, or `BOTTOM`.

It is required to specify a mode pin before set a flag.

Another flag you can use to change the appearance of the pin are `LINE_OVER`, which will add a line over the pin label, `BUBBLE`, which will add a dot at the pin post, and `CLOCK`, which will add a > symbol at the pin label position.

Parameters

`pin`: the Custom Element pin to set the label of.

`label`: the string you want to assign as pin label.

`mode`: `POWER`, `GROUND` ,`INPUT`, `OUTPUT`, `INPUT_PULLUP`, `THREE_STATE`, `OPEN_DRAIN`, or `OPEN_COLLECTOR`. Default value as INPUT.

`flags`: `LINE_OVER`, `BUBBLE`, `CLOCK`, `LEFT`, `RIGHT`, `TOP`, or `BOTTOM`. To set multiple flags you can use the bit or operator `|`.

Returns

Nothing.

Example Code

The codes assigns the specific labels, mode, flags and position for each pin in the chip.

```
void setup() {
    // Set pin 1 as input on the left side of the chip
    pin(1, "A", INPUT, LEFT);
    // Set pin 2 as output on the right side of the chip
    pin(2, "B", OUTPUT, RIGHT);
```

```
    // Set pin 3 as open collector on the left side of the chip
    pin(3, "C", OPEN_COLLECTOR, LINE_OVER | LEFT);
    // Set pin 4 as power on the right side of the chip
    pin(4, "VCC", POWER, RIGHT);
    // Set pin 5 as ground on the left side of the chip
    pin(5, "GROUND", GROUND, LEFT);
    // Sets the digital pin 7 an `OPEN_COLLECTOR` output
    // and toggles it by alternating between HIGH and LOW at one second
    ↪  pace.
    pin(6, "OUT", OPEN_COLLECTOR, RIGHT);
}

void loop() {
    // Remember that the pin 6 is an OPEN COLLECTOR output,
    // so it will only pull the voltage down

    digitalWrite(6, HIGH);   // sets the digital pin 7 to HIGH
    delay(1000);             // waits for a second
    digitalWrite(6, LOW);    // sets the digital pin 7 to LOW
    delay(1000);             // waits for a second
}
```

- This function can work properly only in the setup context, calling it in the loop context won't result in a pin label change.
- All the pins can be used as digital pins, except for the power and ground pins.

pinLabel() (Deprecated)

[Circuit Interface]

```
void pinLabel(unsigned short pin, const char *label, unsigned short flags =
↪  0);
```

Description

Defines the label to show besides the pin.

It is possible to change the appearance of the pin and the label using flags.

Parameters

`pin`: the Custom Element pin to set the label of.

`name`: the string you want to assign as pin label

`flags`: `LINE_OVER`, `BUBBLE`, or `CLOCK`. To set multiple flags you can use the bit or operator `|`. This argument is optional.

Returns

Nothing.

Example Code

The codes assign the label “CLK” to the pin 13, and sets the flags `BUBBLE` and `CLOCK`

```
void setup() {
   pinLabel(13, "CLK", BUBBLE | CLOCK);
}

void loop() {}
```

This function can work properly only in the setup context, calling it in the loop context won’t result in a pin label change.

logicFamily()

[Circuit Interface]

```
void logicFamily(unsigned short family);
```

Description

Set the generic family electronic standard for the Custom Element.

Parameters

`family`: `TTL`, or `CMOS`.

Returns

Nothing.

Example Code

The code sets the logic family for the chip to `TTL`.

```
void setup() {
   logicFamily(TTL);
}

void loop() {}
```

By default, the family is set to `CMOS`.

Digital IO

digitalRead()

[Digital I/O]

```
unsigned char digitalRead(unsigned short pin);
```

Description

Reads the value from a specified digital pin.

Parameters

`pin`: the Custom Element pin you want to read.

Returns

`HIGH` or `LOW`.

Example Code

Sets pin 7 to the same value as pin 3.

```
void setup() {
    pin(7, "OUT", OUTPUT);   // sets the digital pin 7 as output
    pin(3, "IN");    // sets the pin 3 label to "IN" (by default will be an
↪   input)
}

void loop() {
    int val = digitalRead(3);   // read the input pin
    digitalWrite(7, val);       // sets the output pin value
}
```

If the pin is defined as three-state and the ground pin isn't connected to anything, `digitalRead()` will always return `HIGH`.

digitalReadNibble()

[Digital I/O]

```
unsigned char digitalReadNibble(unsigned short a, unsigned char b, unsigned
↪  char c, unsigned char d);
```

Description

Reads the value from the specified nibble (4 bits) of the Custom Element pins.

Parameters

a: the pin of the most significant bit of the nibble (MSB).

b: the pin of the second most significant bit of the nibble.

c: the pin of the second least significant bit of the nibble.

d: the pin of the least significant bit of the nibble (LSB).

Returns

The value of the nibble (0-15) as an 8-bit value.

Example Code

The code reads the value of a nibble from the Custom Element pins 3, 4, 5, and 6.

```
void setup() {
    pin(3, "IN");   // sets the pin 3 label to "IN" (by default will be an
↪  input)
    pin(4, "IN");   // sets the pin 4 label to "IN" (by default will be an
↪  input)
    pin(5, "IN");   // sets the pin 5 label to "IN" (by default will be an
↪  input)
    pin(6, "IN");   // sets the pin 6 label to "IN" (by default will be an
↪  input)
}

void loop() {
    unsigned char nibble = digitalReadNibble(3, 4, 5, 6);   // reads the
    ↪  value of the nibble from the pins 3, 4, 5, and 6
}
```

If the pin is defined as three-state and the ground pin isn't connected to anything, `digitalReadNibble()` will always return 15.

digitalWrite()

[Digital I/O]

```
void digitalWrite(unsigned short pin, unsigned char value);
```

Description

Writes a `HIGH` or a `LOW` value to a digital pin.

If the pin has been configured as an output with `pinMode()`, its voltage will be set to the corresponding value, following the family electronic standards configured with `logicFamily()`.

If the pin is configured as an input, the function will have no effect.

Parameters

`pin`: the Custom Element pin.

`value`: `HIGH` or `LOW`

Returns

Nothing.

Example Code

The code makes the digital pin 7 an OUTPUT and toggles it by alternating between HIGH and LOW at one second pace.

```
void setup() {
    pin(7, "OUT", OUTPUT);   // sets the digital pin 7 as output
}

void loop() {
    digitalWrite(7, HIGH);   // sets the digital pin 7 to HIGH
    delay(1000);             // waits for a second
    digitalWrite(7, LOW);    // sets the digital pin 7 to LOW
    delay(1000);             // waits for a second
}
```

All the pins are set as inputs by default, so they must be configured as outputs with `pinMode()` before using `digitalWrite()`.

digitalWriteNibble()

[Digital I/O]

```
void digitalWriteNibble(unsigned short a, unsigned short b, unsigned short
↪  c, unsigned short d, unsigned char value);
```

Description

Writes a 4-bit value (nibble) to the specified Custom Element pins.

If the pin has been configured as an output with `pinMode()`, its voltage will be set to the corresponding value, following the family electronic standards configured with `logicFamily()`.

If the pins are configured as an input, the function will have no effect.

Parameters

a: the pin of the most significant bit of the nibble (MSB).

b: the pin of the second most significant bit of the nibble.

c: the pin of the second least significant bit of the nibble.

d: the pin of the least significant bit of the nibble (LSB).

`value`: `HIGH` or `LOW`

Returns

Nothing.

Example Code

The code writes the value of a nibble to the Custom Element pins 3, 4, 5, and 6.

```
void setup() {
    pin(3, "OUT", OUTPUT);   // sets the digital pin 3 as output

    pin(4, "OUT", OUTPUT);   // sets the digital pin 4 as output

    pin(5, "OUT", OUTPUT);   // sets the digital pin 5 as output

    pin(6, "OUT", OUTPUT);   // sets the digital pin 6 as output
}

void loop() {
    digitalWriteNibble(3, 4, 5, 6, 10);   // writes the value 0b1010 to the
↪  pins 3, 4, 5, and 6
}
```

All the pins are set as inputs by default, so they must be configured as outputs with `pinMode()` before using `digitalWriteNibble()`.

pinThreeStateMode()

[Digital I/O]

```
void pinThreeStateMode(unsigned short pin, unsigned short mode);
```

Description

Configures the impedance mode of the specified three-state pin.

It only works with pins that have been defined as three-state pins with `pinMode()`.

Parameters

`pin`: the Custom Element pin.

`mode`: `HIGH_IMPEDANCE`, `HIGH_Z` or `LOW_IMPEDANCE`, `LOW_Z`

Returns

Nothing.

Example Code

The code sets the digital pin 7 as a three-state pin toggle the impedance state between `HIGH_Z` and `LOW_Z`.

```
void setup() {
    pin(7, "OUT", THREE_STATE); // sets the digital pin 7 as THREE_STATE
↪   output
}

void loop() {
    // Remember that the pin is a THREE_STATE output,
    // so the High Z state will disconnect the pin from the circuit
    digitalWrite(7, HIGH);                // sets the digital pin 7 to HIGH
    pinThreeStateMode(7, HIGH_IMPEDANCE);   // sets the digital pin 7 to
↪   HIGH_IMPEDANCE
    delay(1000);                          // waits for a second
    pinThreeStateMode(7, LOW_IMPEDANCE);    // sets the digital pin 7 to
↪   LOW_IMPEDANCE
    delay(1000);                          // waits for a second
}
```

All the pins can be used as three-state pins, except for the power and ground pins. By default, all pins are set as input pins (High Z Mode).

maxVoltage()

[Digital I/O]

```
void maxVoltage(unsigned short pin, float voltage);
```

Description

Sets the maximum input voltage for the specified pin.

Parameters

`pin`: the Custom Element pin.

`max_voltage`: the maximum input voltage allowed for the pin.

Returns

Nothing.

Example Code

The code sets the maximum input voltage for the open-collector pin 7 to 15V.

```
void setup() {
    pin(7, "OUT", OPEN_COLLECTOR);   // sets the digital pin 7 as OPEN
↪   COLLECTOR output
    maxVoltage(7, 15);    // sets the maximum input voltage for the pin 7
↪   to 15V
}

void loop() {}
```

The maximum input voltage can be set for all pins, except for the power and ground pins.
The default value is defined by the family electronic standards.
This voltage will be used to raise warnings when the input voltage exceeds the maximum value.

Analog IO

analogRead()

[Analog I/O]

```
short analogRead(unsigned short pin);
```

Description

Reads the value from the specified pin.

This function allows to read the voltage value from every single pin from the Custom Element. The value is returned as an integer between 0 and 1023, where 0 corresponds to 0V and 1023 corresponds to the power voltage.

Parameters

`pin`: the Custom Element pin.

Returns

An integer value between 0 and 1023.

Example Code

The code reads the analog value from the pin 3 and sets the value obtained to the pin 2.

```
void setup() {
    pin(3, "IN");   // sets the pin 3 label to "IN"
    pin(2, "OUT", OUTPUT);  // sets the pin 2 label to "OUT"
}

void loop() {
    int value = analogRead(3);  // reads the analog value from the pin 3
    analogWrite(2, value);      // sets the value obtained to the pin 2
}
```

The analogRead() function can be used with all the pins, even for the power and ground pins.

analogWrite()

[Analog I/O]

```
void analogWrite(unsigned short pin, short value);
```

Description

Writes an analog value to a pin.

The `analogWrite()` function writes an analog value to a pin. The value is specified as an integer between 0 and 1023, where 0 corresponds to 0V and 1023 corresponds to the output voltage defined by the family electronic specifications.

Parameters

`pin`: the Custom Element pin.

`value`: the value to write to the pin.

Returns

Nothing.

Example Code

The code reads the analog value from the pin 3 and sets the value obtained to the pin 2.

```
void setup() {
    pin(3, "IN");   // sets the pin 3 label as input
    pin(2, "OUT", OUTPUT);  //  sets the digital pin 2 as output
}

void loop() {
    int value = analogRead(3);  // reads the analog value from the pin 3
    analogWrite(2, value);      // sets the value obtained to the pin 2
}
```

The `analogWrite()` function can be used only the pins defined as outputs.

outputCurrent()

[Analog I/O]

```
float outputCurrent(unsigned short pin);
```

Description

Reads the current value from the specified output pin.

Parameters

`pin`: the Custom Element pin.

Returns

A float value representing the current value in A.

Example Code

The code reads the current value from the pin 3 and converts the value to a voltage for the `analogWrite()` function.

```
void setup() {
    pin(3, "OUT", OUTPUT);   // sets the digital pin 3 as output
    pin(2, "IN");    // sets the digital pin 2 as input
}

void loop() {
    float current = outputCurrent(3);  // reads the current value from the
    ↪  pin 3
    float voltage = current * 10;      // converts the current value to a
    ↪  voltage
    analogWrite(2, voltage);           // sets the voltage obtained to the
↪  pin 2
}
```

The `outputCurrent()` function can be used only with the pins defined as outputs.

Timing

delay()

[Timing]

```
void delay(unsigned long ms);
```

Description

Pauses the program for the specified amount of time (in milliseconds) as parameter.

Parameters

`ms`: time in milliseconds.

Returns

Nothing.

Example Code

The code creates a clock with the pin 2, toggling the output every 500 milliseconds (1Hz).

```
void setup() {
    pin(2, "OUT", OUTPUT);  // sets the digital pin 2 as output
}

void loop() {
    delay(500);  // pauses the program for 500 milliseconds
    digitalWrite(2, !digitalRead(2));  // toggles the output of the pin 2
}
```

There are no ways to avoid the delay function, it is a blocking function that stops the program execution for the specified time.

micros()

[Timing]

```
unsigned long micros();
```

Description

Returns the number of microseconds since the Custom Element was powered up.

Parameters

None.

Returns

The number of microseconds.

Example Code

The code creates a counter that increments the value of `counter` every 1000 microseconds (1kHz), and shows the output using the output pins 1, 2, 3, and 4 (4 bits).

```
void setup() {
    pin(1, "O1", OUTPUT);  // sets the digital pin 1 as output
    pin(2, "O2", OUTPUT);  // sets the digital pin 2 as output
    pin(3, "O3", OUTPUT);  // sets the digital pin 3 as output
    pin(4, "O4", OUTPUT);  // sets the digital pin 4 as output
}

void loop() {
    static unsigned long lastTime = 0;
    static unsigned long counter = 0;
    if (micros() - lastTime >= 1000) {  // checks if the number of
    ↪  microseconds is greater than or equal to 1000
        lastTime = micros();
        counter++;
        digitalWrite(1, counter & 0x01);  // sets the output pin 1 with the
↪   first bit of the counter
        digitalWrite(2, counter & 0x02);  // sets the output pin 2 with the
↪   second bit of the counter
        digitalWrite(3, counter & 0x04);  // sets the output pin 3 with the
↪   third bit of the counter
        digitalWrite(4, counter & 0x08);  // sets the output pin 4 with the
↪   fourth bit of the counter
    }
}
```

There are 1,000 microseconds in a millisecond and 1,000,000 microseconds in a second.

millis()

[Timing]

```
unsigned long millis();
```

Description

Returns the number of milliseconds since the Custom Element was powered up.

Parameters

None.

Returns

The number of milliseconds.

Example Code

The code creates a clock with the pin 2, toggling the output every 500 milliseconds (1Hz).

```
void setup() {
    pin(2, "OUT", OUTPUT);  // sets the digital pin 2 as output
}

void loop() {
    if (millis() % 500 == 0) {  // checks if the number of milliseconds is
    ↪  divisible by 500
        digitalWrite(2, !digitalRead(2));  // toggles the output on pin 2
    }
}
```

The return value of the `millis()` function is of type `unsigned long`, some logic operations may occur if the programmer to do arithmetic operations with smaller data types `int`.

Simulation

capacitance()

[Simulation]

```
void capacitance(unsigned short pin_from, unsigned short pin_to, float
↪  capacitance);
```

Description

Creates a capacitance between the specified pins.

Parameters

`pin_from`: the Custom Element pin where the capacitance starts.

`pin_to`: the Custom Element pin where the capacitance ends.

`capacitance`: the capacitance value in farads.

Returns

Nothing.

Example Code

The code creates a capacitance of 1uF between the pins 3 and 4.

```
void setup() {
    pin(3, "IN");   // sets the pin 3 label to "IN" (input by default)
    pin(4, "OUT", OUTPUT);  // sets the pin 4 label to "OUT" (input by
↪  default)

    capacitance(3, 4, 0.000001);  // creates a capacitance of 1uF between
↪  the pins 3 and 4
}

void loop() {
}
```

- The `capacitance()` function can be used with all the pins, even for the power and ground pins.
- The capacitance value must be greater than 0 F.
- The `capacitance()` function can only be set during the setup, and won't change the pin mode, so be careful when using it with the pins defined as outputs.
- Any call to `capacitance()` during the loop will be ignored.

currentSource()

[Simulation]

```
void currentSource(unsigned short pin_from, unsigned short pin_to, float
↪  current, unsigned short flags = LINEAR);
```

Description

Creates a current source between the specified pins.

Parameters

`pin_from`: the Custom Element pin where the capacitance starts.

`pin_to`: the Custom Element pin where the capacitance ends.

`current`: the current value in amperes.

`flags`: `NON_LINEAR` or `LINEAR` to set the current source as non-linear or linear.

Returns

Nothing.

Example Code

The code creates a current source of 1A between the pins 3 and 4, and creates a non-linear current source between the pins 5 and 6 and increases the current value by 1mA every 100ms (resets at 100mA).

```
static unsigned long lastTime = 0;
static float current = 0.001;

void setup() {
    pinLabel(3, "IN");   // sets the pin 3 label to "IN" (input by default)
    pinLabel(4, "OUT");  // sets the pin 4 label to "OUT" (input by
↪  default)

    pinLabel(5, "IN");   // sets the pin 5 label to "IN" (input by default)
    pinLabel(6, "OUT");  // sets the pin 6 label to "OUT" (input by
↪  default)

    currentSource(3, 4, 1);  // creates a current source of 1A between the
↪  pins 3 and 4
    currentSource(5, 6, current, NON_LINEAR);  // creates a non-linear
↪  current source between the pins 5 and 6
}
void loop() {
```

```
    if (millis() - lastTime >= 100) {      // checks if the number of
    ↪  miliseconds is greater than 100
        lastTime = millis();
        current += 0.001;                  // increases the current value
↪   by 1mA
        if (current >= 0.1) {              // checks if the current value
        ↪  is greater than or equal to 100mA
            current = 0.001;
        }
        currentSource(5, 6, current);  // updates the current value between
↪   the pins 5 and 6 (the flag is not needed)
    }
}
```

- The `currentSource()` function can be used with all the pins, even for the power and ground pins.
- The current value must be greater than 0 A.
- The `flags` parameter is optional, if not specified, the current source will be set as linear.
- If the current source is set as linear, it can only be set during the setup.
- If the current source is set as non-linear, it can be changed during the setup or the loop, but needs to be initialized during the setup.
- The `currentSource()` function can be used to create a current source between any two pins, and won't change the pin mode, so be careful when using it with the pins defined as outputs.

inductance()

[Simulation]

```
void inductance(unsigned short pin_from, unsigned short pin_to, float
↪  inductance);
```

Description

Creates an inductance between the specified pins.

Parameters

`pin_from`: the Custom Element pin where the inductance starts.

`pin_to`: the Custom Element pin where the inductance ends.

`inductance`: the inductance value in henries.

Returns

Nothing.

Example Code

The code creates an inductance of 1mH between the pins 3 and 4.

```
void setup() {
    pin(3, "IN");   // sets the pin 3 label to "IN" (input by default)
    pin(4, "OUT", OUTPUT);  // sets the pin 4 label to "OUT" (input by
↪  default)

    inductance(3, 4, 0.001);  // creates an inductance of 1mH between the
↪  pins 3 and 4
}

void loop() {
}}
```

- The `inductance()` function can be used with all the pins, even for the power and ground pins.
- The inductance value must be greater than 0 F.
- The `inductance()` function can only be set during the setup, and won't change the pin mode, so be careful when using it with the pins defined as outputs.
- Any call to `inductance()` during the loop will be ignored.

resistance()

[Simulation]

```
void resistance(unsigned short pin_from, unsigned short pin_to, float
↪  resistance, unsigned short flags = LINEAR);
```

Description

Creates a resistance between the specified pins.

Parameters

`pin_from`: the Custom Element pin where the resistance starts.

`pin_to`: the Custom Element pin where the resistance ends.

`resistance`: the resistance value in ohms.

`flags`: `NON_LINEAR` or `LINEAR` to set the resistance as non-linear or linear, respectively, this parameter is optional.

Returns

Nothing.

Example Code

The code creates a resistance of 1k ohm between the pins 3 and 4, and creates a non-linear resistance between the pins 5 and 6 and increases the resistance value by 100 ohms every 100ms (resets at 1M ohm).

```
static unsigned long lastTime = 0;
static float r = 100;

void setup() {
    pin(3, "IN");   // sets the pin 3 label to "IN" (input by default)
    pin(4, "OUT", OUTPUT);  // sets the pin 4 label to "OUT" (input by
↪  default)

    pin(5, "IN");   // sets the pin 5 label to "IN" (input by default)
    pin(6, "OUT", OUTPUT);  // sets the pin 6 label to "OUT" (input by
↪  default)

    resistance(3, 4, 1000);  // creates a resistance of 1k ohm between the
↪  pins 3 and 4
    resistance(5, 6, r, NON_LINEAR);  // creates a non-linear resistance
↪  between the pins 5 and 6
```

```
}

void loop() {
    if (millis() - lastTime >= 100) {      // checks if the number of
     ↪  miliseconds is greater than 100
        lastTime = millis();
        r += 100;                 // increases the resistance value by 100
↪   ohms
        if (r >= 1000000) {       // checks if the resistance value is
         ↪  greater than or equal to 1M ohm
            r = 100;
        }
        resistance(5, 6, r);  // updates the resistance value between the
↪   pins 5 and 6 (the flag is not needed)
    }
}
```

- The `resistance()` function can be used with all the pins, even for the power and ground pins.
- The resistance value must be greater than 0 ohms.
- The `flags` parameter is optional, if not specified, the resistance will be set as linear.
- If the resistance is set as linear, it can only be set during the setup.
- If the resistance is set as non-linear, it can be changed during the setup or the loop, but needs to be initialized during the setup.
- The `resistance()` function can be used to create a resistance between any two pins, and won't change the pin mode, so be careful when using it with the pins defined as outputs.

EEPROM

beginEEPROM()

[EEPROM]

```
void beginEEPROM (unsigned short size, unsigned char byte_width = 1);
```

Description

Initializes the EEPROM memory, allowing the programmer to read and write data to the EEPROM.

Parameters

`size`: the size of the EEPROM memory.

`byte_width`: the width of the data to be written in bytes.

Returns

Nothing.

Example Code

The code initializes the EEPROM memory with a size of 1024 bytes and a data width of 1 byte.

```
void setup() {
    beginEEPROM(1024);  // initializes the EEPROM memory with a size of
↪   1024 bytes and a data width of 1 byte
}

void loop() {
}
```

- The `beginEEPROM()` function must be called only once in the setup.
- The `byte_width` parameter is optional, and if not specified, the default value is 1 byte.
- This memory is non-volatile, so the data will be preserved even if the Custom Element is powered off, or iCircuit is closed.
- To get the real size of the EEPROM in bytes, multiply the `size` parameter by the `byte_width` parameter.

writeEEPROM()

[EEPROM]

```
void writeEEPROM(unsigned short address, unsigned char value);
```

Description

Writes a byte value (8 bits) to the EEPROM memory at the specified address.

Parameters

`address`: the address where the data will be written.

`value`: the data to be written.

Returns

Nothing.

Example Code

The code writes the value 0x55 to the EEPROM memory at the address 0x00.

```
void setup() {
    beginEEPROM(1024);        // initializes the EEPROM memory with a size
↪   of 1024 bytes and a data width of 1 byte
    writeEEPROM(0x00, 0x55);  // writes the value 0x55 to the EEPROM memory
↪   at the address 0x00
}

void loop() {
}
```

- The `writeEEPROM()` function can only be used after the `beginEEPROM()` function.
- The `value` parameter must be an 8-bit value (0-255) or an `unsigned char`.

writeEEPROM16()

[EEPROM]

```
void writeEEPROM16(unsigned short address, unsigned short value);
```

Description

Writes a short value (16 bits) to the EEPROM memory at the specified address.

Parameters

`address`: the address where the data will be written.

`value`: the data to be written.

Returns

Nothing.

Example Code

The code initializes the EEPROM with a size of 1024 and a width of 2 bytes and writes the value 0x55FF to the EEPROM memory at the address 0x00.

```
void setup() {
    beginEEPROM(1024, 2);          // initializes the EEPROM memory with a
↪   size of 1024 * 2 bytes
    writeEEPROM16(0x00, 0x55FF);   // writes the value 0x55FF to the EEPROM
↪   memory at the address 0x00
}

void loop() {
}
```

- The `writeEEPROM16()` function can only be used after the `beginEEPROM()` function.
- The `value` parameter must be a 16-bit value (0-65535) or an `unsigned short`.
- If the programmer initializes the EEPROM with a width lower than 2 bytes, the function will write only 8 bits of the data.

readEEPROM()

[EEPROM]

```
unsigned char readEEPROM(unsigned short address);
```

Description

Reads a byte value (8 bits) from the EEPROM memory at the specified address.

Parameters

`address`: the address where the data will be read.

Returns

The data read from the EEPROM memory at the specified address. The data is an 8-bit value (0-255).

Example Code

The code initializes the EEPROM with a size of 1024 bytes and a width of 1 byte and reads the value from the EEPROM memory at the address 0x00 to place it in the pins 1, 2, 3, and 4.

```
void setup() {
    beginEEPROM(1024);        // initializes the EEPROM memory with a size
↪   of 1024 bytes and a data width of 1 byte
    writeEEPROM(0x00, 0x55);  // writes the value 0x55 to the EEPROM memory
↪   at the address 0x00
}

void loop() {
    unsigned char value = readEEPROM(0x00);  // reads the value from the
     ↪  EEPROM memory at the address 0x00
    digitalWriteNibble(1, 2, 3, 4, value); // takes the 4 least significant
↪   bits
                                           // and writes them to the
                                            ↪  specified pins
}
```

- The `readEEPROM()` function can only be used after the `beginEEPROM()` function.
- The returned value is an 8-bit value (0-255) or an `unsigned char`.

readEEPROM16()

[EEPROM]

```
unsigned short readEEPROM16(unsigned short address);
```

Description

Reads a short value (16 bits) from the EEPROM memory at the specified address.

Parameters

`address`: the address where the data will be read.

Returns

The data read from the EEPROM memory at the specified address. The data is a 16-bit value (0-65535).

Example Code

The code initializes the EEPROM with a size of 1024 bytes and a width of 2 byte and reads the value from the EEPROM memory at the address 0x00 to place it in the pins 1, 2, 3, and 4.

```
void setup() {
    beginEEPROM(1024, 2);       // initializes the EEPROM memory with a
↪   size of 1024 * 1 bytes
    writeEEPROM16(0x00, 0x55FF);  // writes the value 0x55 to the EEPROM
↪   memory at the address 0x00
}

void loop() {
    unsigned short value = readEEPROM(0x00);  // reads the value from the
     ↪  EEPROM memory at the address 0x00
    digitalWriteNibble(1, 2, 3, 4, value); // takes the 4 least significant
↪   bits
                                           // and writes them to the
                                            ↪  specified pins
}
```

- The `readEEPROM16()` function can only be used after the `beginEEPROM()` function.
- The returned value is a 16-bit value (0-65535) or an `unsigned short`.
- If the programmer initializes the EEPROM with a width lower than 2 bytes, the function will return only 8 bits of the data.

updateEEPROM()

[EEPROM]

```
void updateEEPROM(unsigned short address, unsigned char value);
```

Description

Updates the EEPROM memory with the specified byte value at the specified address.

Parameters

`address`: the address where the data will be written.

`value`: the data to be written if the data is different from the data already stored in the EEPROM memory.

Returns

Nothing.

Example Code

The code initializes the EEPROM with a size of 1024 bytes and a width of 1 byte and updates the value in the EEPROM memory at the address 0x00 with a counter value, incrementing it by one each time the loop is executed.

```
void setup() {
    beginEEPROM(1024);        // initializes the EEPROM memory with a size
↪   of 1024 bytes and a data width of 1 byte
    writeEEPROM(0x00, 0x00);  // writes the value 0x00 to the EEPROM memory
↪   at the address 0x00
}

void loop() {
    unsigned char value = readEEPROM(0x00);  // reads the value from the
     ↪  EEPROM memory at the address 0x00
    value++;                                 // increments the value by one
    updateEEPROM(0x00, value);               // updates the value in the
↪   EEPROM memory at the address 0x00
}
```

- The `updateEEPROM()` function can only be used after the `beginEEPROM()` function.
- The `value` parameter must be an 8-bit value (0-255) or an `unsigned char`.

updateEEPROM16()

[EEPROM]

```
void updateEEPROM16(unsigned short address, unsigned short value);
```

Description

Updates the EEPROM memory with the specified short value at the specified address.

Parameters

`address`: the address where the data will be written.

`value`: the data to be written if the data is different from the data already stored in the EEPROM memory.

Returns

Nothing.

Example Code

The code initializes the EEPROM with a size of 1024 and a width of 2 bytes and updates the value in the EEPROM memory at the address 0x00 with a counter value, incrementing it by one each time the loop is executed.

```
void setup() {
    beginEEPROM(1024, 2);          // initializes the EEPROM memory with a
↪   size of 1024 * 2 bytes
    writeEEPROM16(0x00, 0x0000);   // writes the value 0x0000 to the EEPROM
↪   memory at the address 0x00
}

void loop() {
    unsigned short value = readEEPROM16(0x00);  // reads the value from the
     ↪  EEPROM memory at the address 0x00
    value++;                                    // increments the value by
↪   one
    updateEEPROM16(0x00, value);                // updates the value in the
↪   EEPROM memory at the address 0x00
}
```

- The `updateEEPROM16()` function can only be used after the `beginEEPROM()` function.
- The `value` parameter must be a 16-bit value (0-65535) or an `unsigned short`.
- If the programmer initializes the EEPROM with a width lower than 2 bytes, the function will write only 8 bits of the data.

clearEEPROM()

[EEPROM]

```
void clearEEPROM ();
```

Description

Clears the EEPROM memory by writing zeros to all addresses.

Parameters

None.

Returns

Nothing.

Example Code

The code initializes the EEPROM with a size of 1024 bytes and a width of 1 byte and clears the EEPROM memory.

```
void setup() {
    beginEEPROM(1024);  // initializes the EEPROM memory with a size of
↪   1024 bytes and a data width of 1 byte
    clearEEPROM();      // clears the EEPROM memory
}

void loop() {
}
```

- The `clearEEPROM()` function can only be used after the `beginEEPROM()` function.
- The `clearEEPROM()` function can be called multiple times to clear the EEPROM memory, even inside the loop.

RAM

beginRAM()

[RAM]

```
void beginRAM (unsigned short size, unsigned char byte_width = 1);
```

Description

Initializes the RAM memory, allowing the programmer to read and write data to the RAM.

Parameters

`size`: the size of the RAM memory.

`byte_width`: the width of the data to be written in bytes.

Returns

Nothing.

Example Code

The code initializes the RAM memory with a size of 1024 bytes and a data width of 1 byte.

```
void setup() {
    beginRAM(1024);  // initializes the RAM memory with a size of 1024
↪   bytes and a data width of 1 byte
}

void loop() {
}
```

- The `beginRAM()` function must be called only once in the setup.
- The `byte_width` parameter is optional, and if not specified, the default value is 1 byte.
- This memory is volatile, so the data will be lost when the power is turned off, reset, or iCircuit is closed.
- To get the real size of the RAM in bytes, multiply the `size` parameter by the `byte_width` parameter.

writeRAM()

[RAM]

```
void writeRAM(unsigned short address, unsigned char value);
```

Description

Writes a byte value (8 bits) to the RAM memory at the specified address.

Parameters

`address`: the address where the data will be written.

`value`: the data to be written.

Returns

Nothing.

Example Code

The code writes the value of a counter to the RAM memory at the address 0x00.

```
static unsigned char counter = 0;

void setup() {
    beginRAM(1024);         // initializes the RAM memory with a size of
↪   1024 bytes and a data width of 1 byte
}

void loop() {
    writeRAM(0x00, counter);  // writes the value of the counter to the RAM
↪   memory at the address 0x00
    counter++;                // increments the counter by one
}
```

- The `writeRAM()` function can only be used after the `beginRAM()` function.
- The `value` parameter must be an 8-bit value (0-255) or an `unsigned char`.

writeRAM16()

[RAM]

```
void writeRAM16(unsigned short address, unsigned short value);
```

Description

Writes a short value (16 bits) to the RAM memory at the specified address.

Parameters

`address`: the address where the data will be written.

`value`: the data to be written.

Returns

Nothing.

Example Code

The code initializes the RAM with a size of 1024 and a width of 2 bytes and writes the value of a counter to the RAM memory at the address 0x00.

```
static unsigned short counter = 0;

void setup() {
    beginRAM(1024, 2);          // initializes the RAM memory with a size
↪   of 1024 * 2 bytes
}

void loop() {
    writeRAM16(0x00, counter);  // writes the value of the counter to the
↪   RAM memory at the address 0x00
    counter++;                  // increments the counter by one
}
```

- The `writeRAM16()` function can only be used after the `beginRAM()` function.
- The `value` parameter must be a 16-bit value (0-65535) or an `unsigned short`.
- If the programmer initializes the RAM with a width lower than 2 bytes, the function will write only 8 bits of the data.

readRAM()

[RAM]

```
unsigned char readRAM (unsigned short address);
```

Description

Reads a byte value (8 bits) from the RAM memory at the specified address.

Parameters

`address`: the address where the data will be read.

Returns

The data read from the RAM memory at the specified address. The data is an 8-bit value (0-255).

Example Code

The code initializes the RAM with a size of 1024 bytes and a width of 1 byte and reads the value from the RAM memory at the address 0x00 to place it in the pins 1, 2, 3, and 4.

```
void setup() {
    beginRAM(1024);        // initializes the RAM memory with a size of
↪   1024 bytes and a data width of 1 byte
    writeRAM(0x00, 0x55);  // writes the value 0x55 to the RAM memory at
↪   the address 0x00
}

void loop() {
    unsigned char value = readRAM(0x00);  // reads the value from the RAM
     ↪  memory at the address 0x00
    digitalWriteNibble(1, 2, 3, 4, value); // takes the 4 least significant
↪   bits
                                           // and writes them to the
                                            ↪  specified pins
}
```

- The `readRAM()` function can only be used after the `beginRAM()` function.
- The returned value is an 8-bit value (0-255) or an `unsigned char`.

readRAM16()

[RAM]

```
unsigned short readRAM16 (unsigned short address);
```

Description

Reads a short value (16 bits) from the RAM memory at the specified address.

Parameters

`address`: the address where the data will be read.

Returns

The data read from the RAM memory at the specified address. The data is a 16-bit value (0-65535).

Example Code

The code initializes the RAM with a size of 1024 bytes and a width of 2 bytes and reads the value from the RAM memory at the address 0x00 to place it in the pins 1, 2, 3, and 4.

```
void setup() {
    beginRAM(1024, 2);       // initializes the RAM memory with a size of
↪  1024 * 1 bytes
    writeRAM16(0x00, 0x55FF);  // writes the value 0x55 to the RAM memory
↪  at the address 0x00
}

void loop() {
    unsigned short value = readRAM16(0x00);  // reads the value from the
    ↪  RAM memory at the address 0x00
    digitalWriteNibble(1, 2, 3, 4, value); // takes the 4 least significant
↪  bits
                                           // and writes them to the
                                            ↪  specified pins
}
```

- The `readRAM16()` function can only be used after the `beginRAM()` function.
- The returned value is a 16-bit value (0-65535) or an `unsigned short`.
- If the programmer initializes the RAM with a width lower than 2 bytes, the function will return only 8 bits of the data.

updateRAM()

[RAM]

```
void updateRAM(unsigned short address, unsigned char value);
```

Description

Updates the RAM memory with the specified byte value at the specified address.

Parameters

`address`: the address where the data will be written.

`value`: the data to be written if the data is different from the data already stored in the RAM memory.

Returns

Nothing.

Example Code

The code initializes the RAM with a size of 1024 bytes and a width of 1 byte and updates the value in the RAM memory at the address 0x00 with a counter value, incrementing it by one each time the loop is executed.

```
void setup() {
    beginRAM(1024);        // initializes the RAM memory with a size of
↪   1024 bytes and a data width of 1 byte
    writeRAM(0x00, 0x00);  // writes the value 0x00 to the RAM memory at
↪   the address 0x00
}

void loop() {
    unsigned char value = readRAM(0x00);  // reads the value from the RAM
     ↪  memory at the address 0x00
    value++;                                // increments the value by one
    updateRAM(0x00, value);              // updates the value in the RAM
↪   memory at the address 0x00
}
```

- The `updateRAM()` function can only be used after the `beginRAM()` function.
- The `value` parameter must be an 8-bit value (0-255) or an `unsigned char`.

updateRAM16()

[RAM]

```
void updateRAM16(unsigned short address, unsigned short value);
```

Description

Updates the RAM memory with the specified short value at the specified address.

Parameters

`address`: the address where the data will be written.

`value`: the data to be written if the data is different from the data already stored in the RAM memory.

Returns

Nothing.

Example Code

The code initializes the RAM with a size of 1024 and a width of 2 bytes and updates the value in the RAM memory at the address 0x00 with a counter value, incrementing it by one each time the loop is executed.

```
void setup() {
    beginRAM(1024, 2);          // initializes the RAM memory with a size
↪   of 1024 * 2 bytes
    writeRAM16(0x00, 0x0000);   // writes the value 0x0000 to the RAM
↪   memory at the address 0x00
}

void loop() {
    unsigned short value = readRAM16(0x00);  // reads the value from the
     ↪  RAM memory at the address 0x00
    value++;                                    // increments the value by
↪   one
    updateRAM16(0x00, value);                // updates the value in the RAM
↪   memory at the address 0x00
}
```

- The `updateRAM16()` function can only be used after the `beginRAM()` function.
- The `value` parameter must be a 16-bit value (0-65535) or an `unsigned short`.
- If the programmer initializes the RAM with a width lower than 2 bytes, the function will write only 8 bits of the data.

clearRAM()

[RAM]

```
void clearRAM ();
```

Description

Clears the RAM memory by writing zeros to all addresses.

Parameters

None.

Returns

Nothing.

Example Code

The code initializes the RAM with a size of 1024 bytes and a width of 1 byte and clears the RAM memory.

```
void setup() {
    beginRAM(1024);  // initializes the RAM memory with a size of 1024
↪   bytes and a data width of 1 byte
}

void loop() {
    clearRAM();      // clears the RAM memory
}
```

- The `clearRAM()` function can only be used after the `beginRAM()` function.
- The `clearRAM()` function can be called multiple times to clear the RAM memory, even inside the loop.

SPI

beginSPI()

[Serial Peripheral Interface]

```
void beginSPI(unsigned short clock,
              unsigned short dataIn,
              unsigned short dataOut,
              unsigned short chip_select = 32768,
              bool polarity = false);
```

Description

Initializes the SPI communication.

To use the SPI peripheral as follower, the `chip_select` and `polarity` parameters must be specified.

Parameters

`clock`: the pin for the clock.

`data_in`: the pin for the data in.

`data_out`: the pin for the data out.

`chip_select`: the pin for the chip select. *This parameter is optional and only necessary if the chip is acting as a follower*.

`polarity`: the chip select polarity. *This parameter is optional and only necessary if the chip is acting as a follower*.

Returns

Nothing.

Example Code

For a Leader Device The code initializes the SPI communication with the clock pin connected to pin 13, the data in pin connected to pin 12, and the data out pin connected to pin 11.

```
void setup() {
    beginSPI(13, 12, 11);  // initializes the SPI communication with the
↪   clock pin connected to pin 13, the data in pin connected to pin 12, and
↪   the data out pin connected to pin 11
}
```

```
void loop() {
}
```

For a Follower Device The code initializes the SPI communication with the clock pin connected to pin 13, the data in pin connected to pin 12, the data out pin connected to pin 11, the chip select pin connected to pin 10, and the polarity of the chip select line set to LOW.

```
void setup() {
    beginSPI(13, 12, 11, 10, LOW);  // initializes the SPI communication
↪   with the clock pin connected to pin 13, the data in pin connected to
↪   pin 12, the data out pin connected to pin 11, the chip select pin
↪   connected to pin 10, and the polarity of the chip select line set to
↪   LOW
}

void loop() {
}
```

- Any pin can be used for the clock, data in, data out, and chip select lines, except for the ground and power pins.
- This function will take of all the pins specified, and configure them as needed, please, do not change the configuration of the pins after calling this function.

beginSPITransmission()

[Serial Peripheral Interface]

```
void beginSPITransmission(unsigned short chip_select, bool polarity =
↪  false);
```

Description

Begins an SPI transmission from a Leader Device.

Parameters

`chip_select`: the pin for the chip select line.

`polarity`: the polarity of the chip select line.

Returns

Nothing.

Example Code

The code initializes the SPI communication with the clock pin connected to pin 13, the data in pin connected to pin 12, and the data out pin connected to pin 11.

The code then begins an SPI transmission to the follower chip with the chip select pin connected to pin 10, and the polarity of the chip select line set to LOW.

```
void setup() {
    beginSPI(13, 12, 11);  // initializes the SPI communication with the
↪  clock pin connected to pin
                           // 13, the data in pin connected to pin 12, and
                            ↪  the data out pin connected
                           // to pin 11
}

void loop() {
    beginSPITransmission(10, LOW);  // begins an SPI transmission to the
↪  follower chip with the chip
                                    // select pin connected to pin 10, and
                                     ↪  the polarity of the chip
                                    // select line set to LOW

    writeSPI(42);                   // sends the value 42 to the follower
↪  chip
    endSPITransmission(10, LOW);    // ends the SPI transmission
}
```

The `beginSPITransmission()` function should be called before sending data to the follower chip.

endSPITransmission()

[Serial Peripheral Interface]

```
int endSPITransmission(unsigned short chip_select, bool polarity = false);
```

Description

Ends an SPI transmission from a Leader Device.

Parameters

`chip_select`: the pin for the chip select line.

`polarity`: the polarity of the chip select line.

Returns

Nothing.

Example Code

The code initializes the SPI communication with the clock pin connected to pin 13, the data in pin connected to pin 12, and the data out pin connected to pin 11.

The code then begins an SPI transmission to the follower chip with the chip select pin connected to pin 10, and the polarity of the chip select line set to LOW.

```
void setup() {
    beginSPI(13, 12, 11);  // initializes the SPI communication with the
↪  clock pin connected to pin
                           // 13, the data in pin connected to pin 12, and
                            ↪  the data out pin connected
                           // to pin 11
}

void loop() {
    beginSPITransmission(10, LOW);  // begins an SPI transmission to the
↪  follower chip with the chip
                                    // select pin connected to pin 10, and
                                     ↪  the polarity of the chip
                                    // select line set to LOW

    writeSPI(42);                   // sends the value 42 to the follower
↪  chip
    endSPITransmission(10, LOW);    // ends the SPI transmission
}
```

The `beginSPITransmission()` function should be called after sending all the data to the follower chip, to release the SPI bus.

selectedSPI()

[Serial Peripheral Interface]

```
bool selectedSPI();
```

Description

Checks if the chip is selected.

Parameters

None.

Returns

A boolean value that indicates if the chip is selected.

Example Code

The code initializes the SPI communication with the clock pin connected to pin 13, the data in pin connected to pin 12, the data out pin connected to pin 11, the chip select pin connected to pin 10, and the polarity of the chip select line set to LOW.

Continuously checks if the chip is selected, and if it is, it sends data to the leader chip.

```
void setup() {
    beginSPI(13, 12, 11, 10, LOW);  // initializes the SPI communication
↪   with the clock pin connected to pin 13, the data in pin connected to
↪   pin 12, the data out pin connected to pin 11, the chip select pin
↪   connected to pin 10, and the polarity of the chip select line set to
↪   LOW
}

void loop() {
    if (selectedSPI()) {  // checks if the chip is selected
        writeSPI(0x01);   // sends data to the leader chip
    }
}
```

This function is only useful if the chip is acting as a follower, and it is needed to continue with the communication.

writeSPI()

[Serial Peripheral Interface]

```
bool writeSPI(unsigned char data);
```

Description

Writes data to the SPI bus and sends it through the data out line.

Parameters

`data`: the data to be sent through the data out line.

Returns

A boolean value that indicates if the data was successfully sent.

Example Code

The code initializes the SPI communication with the clock pin connected to pin 13, the data in pin connected to pin 12, the data out pin connected to pin 11, the chip select pin connected to pin 10, and the polarity of the chip select line set to LOW.

Continuously checks if the chip is selected, and if it is, it sends data to the leader chip.

```
void setup() {
    beginSPI(13, 12, 11, 10, LOW);  // initializes the SPI communication
↪   with the clock pin connected to pin 13, the data in pin connected to
↪   pin 12, the data out pin connected to pin 11, the chip select pin
↪   connected to pin 10, and the polarity of the chip select line set to
↪   LOW
}

void loop() {
    if (selectedSPI()) {  // checks if the chip is selected
        writeSPI(0x01);   // sends data to the leader chip
    }
}
```

The data parameter must be an 8-bit value or an `unsigned char` type.

availableSPI()

[Serial Peripheral Interface]

```
bool availableSPI();
```

Description

Checks if the SPI bus buffer contains any data.

Parameters

None.

Returns

A boolean value that indicates if the SPI bus buffer contains any data.

Example Code

The code initializes the SPI communication with the clock pin connected to pin 13, the data in pin connected to pin 12, the data out pin connected to pin 11, the chip select pin connected to pin 10, and the polarity of the chip select line set to LOW.

Continuously checks if the chip is selected, and if it is, checks if the SPI bus buffer contains any data, and if it does, reads the data from the SPI bus buffer and save it into the EEPROM.

```
void setup() {
    beginEEPROM(32);                // initializes the EEPROM
    beginSPI(13, 12, 11, 10, LOW);  // initializes the SPI communication
↪   with the clock pin connected to pin 13, the data in pin connected to
↪   pin 12, the data out pin connected to pin 11, the chip select pin
↪   connected to pin 10, and the polarity of the chip select line set to
↪   LOW
}

void loop() {
    if (selectedSPI()) {            // checks if the chip is selected
        if (availableSPI()) {       // checks if the SPI bus buffer
        ↪  contains any data
            writeEEPROM(1, readSPI()); // reads the data from the SPI bus
↪   buffer and save it into the EEPROM
        }
    }
}
```

readSPI()

[Serial Peripheral Interface]

```
unsigned char readSPI();
```

Description

Reads data from the follower chip contained in the buffer.

Parameters

None.

Returns

The next byte from the buffer.

Example Code

The code initializes the SPI communication with the clock pin connected to pin 13, the data in pin connected to pin 12, the data out pin connected to pin 11, the chip select pin connected to pin 10, and the polarity of the chip select line set to LOW.

Continuously checks if the chip is selected, and if it is, checks if the SPI bus buffer contains any data, and if it does, reads the data from the SPI bus buffer and save it into the EEPROM.

```
void setup() {
    beginEEPROM(32);                // initializes the EEPROM
    beginSPI(13, 12, 11, 10, LOW);  // initializes the SPI communication
↪   with the clock pin connected to pin 13, the data in pin connected to
↪   pin 12, the data out pin connected to pin 11, the chip select pin
↪   connected to pin 10, and the polarity of the chip select line set to
↪   LOW
}

void loop() {
    if (selectedSPI()) {            // checks if the chip is selected
        if (availableSPI()) {       // checks if the SPI bus buffer
        ↪  contains any data
            writeEEPROM(1, readSPI()); // reads the data from the SPI bus
↪   buffer and save it into the EEPROM
        }
    }
}
```

- If the function `readSPI()` is called when no data is available, the function will return 0.
- The function `readSPI()` will delete the data from the buffer after reading it, if you want to keep the data, you should store it in a variable, or call the function `peekSPI()` instead.
- The return value of the function `readSPI()` is an integer between 0 and 255 or an `unsigned char`.

peekSPI()

[Serial Peripheral Interface]

```
unsigned char peekSPI();
```

Description

Peeks at the data from the follower chip contained in the buffer.

Parameters

None.

Returns

The next byte received from the follower chip without removing it from the buffer.

Example Code

The code initializes the SPI communication with the clock pin connected to pin 13, the data in pin connected to pin 12, the data out pin connected to pin 11, the chip select pin connected to pin 10, and the polarity of the chip select line set to LOW.

Continuously checks if the chip is selected, and if it is, checks if the SPI bus buffer contains any data, and if it does, peek the data and check if it is different from 0, finally reads the data from the SPI bus buffer and save it into the EEPROM.

```
void setup() {
    beginEEPROM(32);                // initializes the EEPROM
    beginSPI(13, 12, 11, 10, LOW);  // initializes the SPI communication
↪   with the clock pin connected to pin 13, the data in pin connected to
↪   pin 12, the data out pin connected to pin 11, the chip select pin
↪   connected to pin 10, and the polarity of the chip select line set to
↪   LOW
}

void loop() {
    if (selectedSPI()) {                    // checks if the chip is
     ↪  selected
        if (availableSPI() && peekSPI()) {  // checks if the SPI bus buffer
         ↪  contains any data
            writeEEPROM(1, readSPI());          // reads the data from the
↪   SPI bus buffer and save it into the EEPROM
        }
    }
}
```

- If the function `peekSPI()` is called when no data is available, the function will return 0.
- The return value of the function `peekSPI()` is an integer between 0 and 255 or an `unsigned char`.

onSPITransaction()

[Serial Peripheral Interface]

```
void onSPITransaction(void (*function)());
```

Description

Sets the function that will be called when the follower SPI bus is selected by first time.

Parameters

`function`: the function that will be called when the follower SPI bus is selected.

Returns

Nothing.

Example Code

The code initializes the SPI communication with the clock pin connected to pin 13, the data in pin connected to pin 12, the data out pin connected to pin 11, the chip select pin connected to pin 10, and the polarity of the chip select line set to LOW.

Sets the function `transactionEvent` as callback when the follower SPI bus is selected.

```
void transactionEvent() {
    // code to be executed when the follower SPI bus is selected
}

void setup() {
    beginSPI(13, 12, 11, 10, LOW);  // initializes the SPI communication
↪   with the clock pin connected to pin 13, the data in pin connected to
↪   pin 12, the data out pin connected to pin 11, the chip select pin
↪   connected to pin 10, and the polarity of the chip select line set to
↪   LOW
    onSPITransaction(transactionEvent);  // sets the function
↪   transactionEvent as callback when the follower SPI bus is selected
}

void loop() {
}
```

The parameter `function` must be a function that does not receive any parameters.

Graphics

chipWidth()

[Graphics]

```
int chipWidth ();
```

Description

Gets the width of the chip.

Parameters

None.

Returns

An integer value that represents the width of the chip in pixels.

Example Code

The code gets the width of the chip and saves it into a variable.

```
void setup() {
    int width = chipWidth(); // gets the width of the chip
}

void loop() {
}
```

This function can be used to calculate relative position of graphics inside the Custom Element canvas

chipHeight()

[Graphics]

```
int chipHeight ();
```

Description

Gets the height of the chip.

Parameters

None.

Returns

An integer value that represents the height of the chip in pixels.

Example Code

The code gets the height of the chip and saves it into a variable.

```
void setup() {
    int width = chipWidth();   // gets the width of the chip
    int height = chipHeight(); // gets the height of the chip
}

void loop() {
}
```

This function can be used to calculate relative position of graphics inside the Custom Element canvas

drawLine()

[Graphics]

```
void drawLine (int x1, int y1, int x2, int y2);
```

Description

Draws a line from the point (x1, y1) to the point (x2, y2).

Parameters

`x1`: the x-coordinate of the starting point of the line.

`y1`: the y-coordinate of the starting point of the line.

`x2`: the x-coordinate of the ending point of the line.

`y2`: the y-coordinate of the ending point of the line.

Returns

Nothing.

Example Code

The code draws a diagonal line from the top-left corner to the bottom-right corner of the chip.

```
void setup() {
    int width = chipWidth();   // gets the width of the chip
    int height = chipHeight(); // gets the height of the chip

    drawLine(0, 0, width, height); // draws a diagonal line from the
↪   top-left corner to the bottom-right corner of the chip
}

void loop() {
}
```

All coordinates must be inside the chip boundaries, use the functions `chipWidth()` and `chipHeight()` to get the chip dimensions.

drawRect()

[Graphics]

```
void drawRect (int x, int y, int width, int height);
```

Description

Draws a rectangle on the chip.

Parameters

`x`: the x-coordinate of the center of the rectangle.

`y`: the y-coordinate of the center of the rectangle.

`width`: the width of the rectangle.

`height`: the height of the rectangle.

Returns

Nothing.

Example Code

The code draws a rectangle in the center of the chip, with a width of 10 pixels and a height of 20 pixels.

```
void setup() {
    int width = chipWidth();   // gets the width of the chip
    int height = chipHeight(); // gets the height of the chip

    // draws a rectangle in the center of the chip, with a
    // width of 10 pixels and a height of 20 pixels
    drawRect(width / 2, height / 2, 10, 20);
}

void loop() {
}
```

All coordinates must be inside the chip boundaries, use the functions `chipWidth()` and `chipHeight()` to get the chip dimensions.

fillRect()

[Graphics]

```
void fillRect (int x, int y, int width, int height);
```

Description

Draws a filled rectangle on the chip.

Parameters

`x`: the x-coordinate of the center of the rectangle.

`y`: the y-coordinate of the center of the rectangle.

`width`: the width of the rectangle.

`height`: the height of the rectangle.

Returns

Nothing.

Example Code

The code draws a filled rectangle in the center of the chip, with a width of 10 pixels and a height of 20 pixels.

```
void setup() {
    int width = chipWidth();   // gets the width of the chip
    int height = chipHeight(); // gets the height of the chip

    // draws a filled rectangle in the center of the chip, with a
    // width of 10 pixels and a height of 20 pixels
    fillRect(width / 2, height / 2, 10, 20);
}

void loop() {
}
```

All coordinates must be inside the chip boundaries, use the functions `chipWidth()` and `chipHeight()` to get the chip dimensions.

drawCircle()

[Graphics]

```
void drawCircle (int x, int y, int radius);
```

Description

Draws a circle on the chip.

Parameters

`x`: the x-coordinate of the center of the rectangle.

`y`: the y-coordinate of the center of the rectangle.

`radius`: the radius of the circle.

Returns

Nothing.

Example Code

The code draws circle in the center of the chip, with a radius of 10 pixels.

```
void setup() {
    int width = chipWidth();   // gets the width of the chip
    int height = chipHeight(); // gets the height of the chip

    // draws a circle in the center of the chip, with a
    // width of 10 pixels and a height of 20 pixels
    drawCircle(width / 2, height / 2, 10);
}

void loop() {
}
```

All coordinates must be inside the chip boundaries, use the functions `chipWidth()` and `chipHeight()` to get the chip dimensions.

fillCircle()

[Graphics]

```
void fillCircle (int x, int y, int radius);
```

Description

Draws a filled circle on the chip.

Parameters

`x`: the x-coordinate of the center of the circle.

`y`: the y-coordinate of the center of the circle.

`radius`: the radius of the circle.

Returns

Nothing.

Example Code

The code draws a filled circle in the center of the chip, with a width of 10 pixels and a height of 20 pixels.

```
void setup() {
    int width = chipWidth();   // gets the width of the chip
    int height = chipHeight(); // gets the height of the chip

    // draws a filled circle in the center of the chip, with a
    // width of 10 pixels and a height of 20 pixels
    fillCircle(width / 2, height / 2, 10);
}

void loop() {
}
```

All coordinates must be inside the chip boundaries, use the functions `chipWidth()` and `chipHeight()` to get the chip dimensions.

drawText()

[Graphics]

```
void drawText (int x, int y, const char* text);
```

Description

Draws text on the chip.

Parameters

x: the x-coordinate where the text will begin.

y: the y-coordinate where the text will begin.

`text`: the text to be drawn.

Returns

Nothing.

Example Code

The code draws the string “Hello” in the center of the chip.

```
void setup() {
    int width = chipWidth();   // gets the width of the chip
    int height = chipHeight(); // gets the height of the chip

    // draws the string "Hello" in the center of the chip
    drawText(width / 2, height / 2, "Hello");
}

void loop() {
}
```

All coordinates must be inside the chip boundaries, use the functions `chipWidth()` and `chipHeight()` to get the chip dimensions.

drawArc()

[Graphics]

```
void drawArc (int x, int y, int radius, float startAngle, float endAngle);
```

Description

Draws an arc on the chip.

Parameters

x: the x-coordinate of the center of the arc.

y: the y-coordinate of the center of the arc.

`radius`: the radius of the arc.

`startAngle`: the start angle of the arc.

`endAngle`: the end angle of the arc.

Returns

Nothing.

Example Code

The code draws an arc in the center of the chip, with a radius of 50 pixels, a start angle of 0 degrees, and an end angle of 90 degrees.

```
void setup() {
    int width = chipWidth();   // gets the width of the chip
    int height = chipHeight(); // gets the height of the chip

    // draws an arc in the center of the chip, with a radius of 50
    // pixels, a start angle of 0 degrees, and an end angle of 90 degrees
    drawArc(width / 2, height / 2, 50, 0, 90 * (PI / 180));
}

void loop() {
}
```

- All coordinates must be inside the chip boundaries, use the functions `chipWidth()` and `chipHeight()` to get the chip dimensions.
- The angles are in radians, use the `PI` constant to convert from degrees to radians.

fillArc()

[Graphics]

```
void fillArc (int x, int y, int width, int height);
```

Description

Draws a filled arc on the chip.

Parameters

x: the x-coordinate of the center of the arc.

y: the y-coordinate of the center of the arc.

`radius`: the radius of the arc.

`startAngle`: the start angle of the arc.

`endAngle`: the end angle of the arc.

Returns

Nothing.

Example Code

The code draws a filled arc in the center of the chip, with a radius of 50 pixels, a start angle of 0 degrees, and an end angle of 90 degrees.

```
void setup() {
    int width = chipWidth();   // gets the width of the chip
    int height = chipHeight(); // gets the height of the chip

    // draws a filled arc in the center of the chip, with a radius of 50
    // pixels, a start angle of 0 degrees, and an end angle of 90 degrees
    fillArc(width / 2, height / 2, 50, 0, 90 * (PI / 180));
}

void loop() {
}
```

- All coordinates must be inside the chip boundaries, use the functions `chipWidth()` and `chipHeight()` to get the chip dimensions.
- The angles are in radians, use the `PI` constant to convert from degrees to radians.

drawOval()

[Graphics]

```
void drawOval (int x, int y, int width, int height);
```

Description

Draws an oval on the chip.

Parameters

x: the x-coordinate of the center of the oval.

y: the y-coordinate of the center of the oval.

`width`: the width of the oval.

`height`: the height of the oval.

Returns

Nothing.

Example Code

The code draws an oval in the center of the chip, with a width of 10 pixels and a height of 20 pixels.

```
void setup() {
    int width = chipWidth();   // gets the width of the chip
    int height = chipHeight(); // gets the height of the chip

    // draws an oval in the center of the chip, with a
    // width of 10 pixels and a height of 20 pixels
    drawOval(width / 2, height / 2, 10, 20);
}

void loop() {
}
```

All coordinates must be inside the chip boundaries, use the functions `chipWidth()` and `chipHeight()` to get the chip dimensions.

fillOval()

[Graphics]

```
void fillOval (int x, int y, int width, int height);
```

Description

Draws a filled oval on the chip.

Parameters

x: the x-coordinate of the center of the oval.

y: the y-coordinate of the center of the oval.

`width`: the width of the oval.

`height`: the height of the oval.

Returns

Nothing.

Example Code

The code draws a filled oval in the center of the chip, with a width of 10 pixels and a height of 20 pixels.

```
void setup() {
    int width = chipWidth();   // gets the width of the chip
    int height = chipHeight(); // gets the height of the chip

    // draws a filled oval in the center of the chip, with a
    // width of 10 pixels and a height of 20 pixels
    fillOval(width / 2, height / 2, 10, 20);
}

void loop() {
}
```

All coordinates must be inside the chip boundaries, use the functions `chipWidth()` and `chipHeight()` to get the chip dimensions.

drawTriangle()

[Graphics]

```
void drawTriangle (int x1, int y1, int x2, int y2, int x3, int y3);
```

Description

Draws a triangle on the chip.

Parameters

`x1`: the x-coordinate of the first point of the triangle.

`y1`: the y-coordinate of the first point of the triangle.

`x2`: the x-coordinate of the second point of the triangle.

`y2`: the y-coordinate of the second point of the triangle.

`x3`: the x-coordinate of the third point of the triangle.

`y3`: the y-coordinate of the third point of the triangle.

Returns

Nothing.

Example Code

The code draws a triangle on the chip, with the three points at (10, 10), (20, 30), and (30, 10).

```
void setup() {
    int width = chipWidth();   // gets the width of the chip
    int height = chipHeight(); // gets the height of the chip

    // draws a triangle on the chip, with the three points
    // at (10, 10), (20, 30), and (30, 10)
    drawTriangle(10, 10, 20, 30, 30, 10);
}

void loop() {
}
```

All coordinates must be inside the chip boundaries, use the functions `chipWidth()` and `chipHeight()` to get the chip dimensions.

fillTriangle()

[Graphics]

```
void fillTriangle (int x1, int y1, int x2, int y2, int x3, int y3);
```

Description

Draws a filled triangle on the chip.

Parameters

`x1`: the x-coordinate of the first point of the triangle.

`y1`: the y-coordinate of the first point of the triangle.

`x2`: the x-coordinate of the second point of the triangle.

`y2`: the y-coordinate of the second point of the triangle.

`x3`: the x-coordinate of the third point of the triangle.

`y3`: the y-coordinate of the third point of the triangle.

Returns

Nothing

Example Code

The code draws a filled triangle on the chip, with the three points at (10, 10), (20, 30), and (30, 10).

```
void setup() {
    int width = chipWidth();   // gets the width of the chip
    int height = chipHeight(); // gets the height of the chip

    // draws a filled triangle on the chip, with the three points
    // at (10, 10), (20, 30), and (30, 10)
    fillTriangle(10, 10, 20, 30, 30, 10);
}

void loop() {
}
```

All coordinates must be inside the chip boundaries, use the functions `chipWidth()` and `chipHeight()` to get the chip dimensions.

Math

cos()

[Math]

```
double cos(double x);
```

Description

Returns the cosine of *x* in radians.

Parameters

x: The angle in radians.

Returns

The cosine of *x* in radians.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the cosine to the pin OUT.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output

    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND); // set pin 4 as ground pin
}

void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double angle = map(val, 0, 1023, 0, 360); // map the value (0 to 360)
    double cosVal = cos(angle * PI / 180); // calculate the cosine
    analogWrite(2, (int) cosVal * 1023); // map the value (0 to 1023)
    delay(100); // delay for 100ms
}
```

The angle should be in radians. To convert the angle from degrees to radians, multiply the angle by `PI / 180`.

sin()

[Math]

```
double sin(double x);
```

Description

Returns the sine of *x* in radians.

Parameters

x: The angle in radians.

Returns

The sine of *x* in radians.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the sine to the pin OUT.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double angle = map(val, 0, 1023, 0, 360); // map the value to the range
    ↪  of 0 to 360
    double sinVal = sin(angle * PI / 180); // calculate the sine of the
    ↪  angle in radians
    analogWrite(2, (int) sinVal * 1023); // map the value to the range of 0
↪  to 1023
    delay(100); // delay for 100ms
}
```

The angle should be in radians. To convert the angle from degrees to radians, multiply the angle by `PI / 180`.

tan()

[Math]

```
double tan(double x);
```

Description

Returns the tangent of *x* in radians.

Parameters

x: The angle in radians.

Returns

The tangent of *x* in radians.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the tangent to the pin `OUT`.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "VCC", GROUND);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double angle = map(val, 0, 1023, 0, 360); // map the value to the range
    ↪  of 0 to 360
    double tanVal = tan(angle * PI / 180); // calculate the tangent of the
    ↪  angle in radians
    analogWrite(2, (int) tanVal * 1023); // map the value to the range of 0
↪  to 1023
    delay(100); // delay for 100ms
}
```

The angle should be in radians. To convert the angle from degrees to radians, multiply the angle by `PI / 180`.

acos()

[Math]

```
double acos(double x);
```

Description

Returns the principal value of the arc cosine of *x* in radians.

Parameters

x: The angle in radians.

Returns

The principal value of the arc cosine of *x* in radians.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the cosine to the pin `OUT`.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set the label of pin 2 to "OUT"
    pin(3, "VCC", POWER);  // set the label of pin 3 to "VCC"
    pin(4, "GND", GROUND);  // set the label of pin 4 to "GND"
}
void loop() {
    int val = analogRead(0); // read the value of the potentiometer
    double angle = map(val, 0, 1023, 0, 180); // map the value
    double acosVal = acos(angle * PI / 180); // calculate the arc cosine
    analogWrite(2, (int)map(val, -PI/2, PI/2, 0, 1024)); // map the value
    delay(100); // delay for 100ms
}
```

- The angle should be in radians. To convert the angle from degrees to radians, multiply the angle by `PI / 180`.
- A domain error occurs if the argument is outside the range [-1, 1]. In this case, the function returns NAN.

asin()

[Math]

```
double asin(double x);
```

Description

Returns the principal value of the arc sine of *x* in radians.

Parameters

x: The angle in radians.

Returns

The principal value of the arc sine of *x* in radians.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the arc sine to the pin `OUT`.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set the label of pin 2 to "OUT"
    pin(3, "VCC", POWER);  // set the label of pin 3 to "VCC"
    pin(4, "GND", GROUND);  // set the label of pin 4 to "GND"
}
void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double angle = map(val, 0, 1023, 0, 180); // map the value
    double asinVal = asin(angle * PI / 180); // calculate the arc sine
    analogWrite(2, (int)map(val, -PI/2, PI/2, 0, 1024)); // map the value
    delay(100); // delay for 100ms
}
```

- The angle should be in radians. To convert the angle from degrees to radians, multiply the angle by `PI / 180`.
- A domain error occurs if the argument is outside the range [-1, 1]. In this case, the function returns NAN.

atan()

[Math]

```
double atan(double x);
```

Description

Returns the principal value of the arc tangent of *x* in radians.

Parameters

x: The angle in radians.

Returns

The principal value of the arc tangent of *x* in radians.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the arc tangent to the pin `OUT`

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}
void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double angle = map(val, 0, 1023, 0, 180); // map the value (0 to 180)
    double atanVal = atan(angle * PI / 180); // calculate the arc tangent
    analogWrite(2, (int)map(val, 0, PI, 0, 1024)); // map the value
    delay(100); // delay for 100ms
}
```

- The angle should be in radians. To convert the angle from degrees to radians, multiply the angle by `PI / 180`.
- A domain error occurs if the argument is outside the range [-1, 1]. In this case, the function returns NAN.

atan2()

[Math]

```
double atan2(double y, double x);
```

Description

Returns the principal value of the arc tangent of $\frac{y}{x}$ in radians, using the signs of both arguments to determine the quadrant of the return value.

Parameters

x: The x-coordinate of the point.

y: The y-coordinate of the point.

Returns

The principal value of the arc tangent of $\frac{y}{x}$ in radians.

Example Code

The code reads the values of two potentiometer connected to pins `INY` and `INX` and place the tangent to the pin `OUT`.

```
void setup() {
    pin(1, "INY");  // set the label of pin 1 to "INY"
    pin(2, "INX");  // set the label of pin 2 to "INX"
    pin(3, "OUT", OUTPUT);  // set pin 3 as output

    pin(4, "VCC", POWER);  // set pin 4 as power pin
    pin(5, "GND", GROUND); // set pin 5 as ground pin
}

void loop() {
    // read the value of the potentiometers
    int val_x = analogRead(1);
    int val_y = analogRead(2);
    // map the values to the range of 0 to 180
    double angle_y = map(val_x, 0, 1023, 0, 180);
    double angle_x = map(val_y, 0, 1023, 0, 180);
    // calculate the arc tangent of the angle in radians
    double atan2Val = atan2(angle_x * PI / 180, angle_y * PI / 180);
    // map the value to the range of 0 to 1023
    analogWrite(3, (int) atan2Val * 1023);
    // delay for 100ms
    delay(100);
```

```
}
```

- The angle should be in radians. To convert the angle from degrees to radians, multiply the angle by `PI / 180`.
- The output value is in the range of $-\pi$ to π.

cosh()

[Math]

```
double cosh(double x);
```

Description

Returns the hyperbolic cosine of *x* in radians.

Parameters

x: The angle in radians.

Returns

The hyperbolic cosine of *x* in radians.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the hyperbolic cosine to the pin `OUT`.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output

    pin(2, "VCC", POWER);  // set pin 3 as power pin
    pin(3, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double angle = map(val, 0, 1023, 0, 360); // map the value to the range
    ↪  of 0 to 360
    double coshVal = cosh(angle * PI / 180); // calculate the hyperbolic
    ↪  cosine of the angle in radians
    analogWrite(2, (int) coshVal * 1023); // map the value to the range of
↪   0 to 1023
    delay(100); // delay for 100ms
}
```

The angle should be in radians. To convert the angle from degrees to radians, multiply the angle by `PI / 180`.

sinh()

[Math]

```
double sinh(double x);
```

Description

Returns the hyperbolic sine of *x* in radians.

Parameters

x: The angle in radians.

Returns

The hyperbolic sine of *x* in radians.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the hyperbolic sine to the pin OUT.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output

    pin(2, "VCC", POWER);  // set pin 3 as power pin
    pin(3, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double angle = map(val, 0, 1023, 0, 360); // map the value to the range
    ↪  of 0 to 360
    double sinhVal = sinh(angle * PI / 180); // calculate the hyperbolic
    ↪  sine of the angle in radians
    analogWrite(2, (int) sinhVal * 1023);  // map the value to the range of
↪  0 to 1023
    delay(100); // delay for 100ms
}
```

The angle should be in radians. To convert the angle from degrees to radians, multiply the angle by `PI / 180`.

tanh()

[Math]

```
double tanh(double x);
```

Description

Returns the hyperbolic tangent of *x* in radians.

Parameters

x: The angle in radians.

Returns

The hyperbolic tangent of *x* in radians.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the hyperbolic tangent to the pin `OUT`.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double angle = map(val, 0, 1023, 0, 360); // map the value to the range
    ↪  of 0 to 360
    double tanhVal = tanh(angle * PI / 180); // calculate the hyperbolic
    ↪  tangent of the angle in radians
    analogWrite(2, (int) tanhVal * 1023); // map the value to the range of
↪  0 to 1023
    delay(100); // delay for 100ms
}
```

The angle should be in radians. To convert the angle from degrees to radians, multiply the angle by `PI / 180`.

acosh()

[Math]

```
double acosh(double x);
```

Description

Returns the principal value of the arc hyperbolic cosine of *x* in radians.

Parameters

x: The angle in radians.

Returns

The principal value of the arc hyperbolic cosine of *x* in radians.

Example Code

The code reads the value of a potentiometer connected to pin IN and place the arc hyperbolic cosine to the pin OUT.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set the label of pin 2 to "OUT"
    pin(3, "VCC", POWER);  // set the label of pin 3 to "VCC"
    pin(4, "GND", GROUND);  // set the label of pin 4 to "GND"
}
void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double angle = map(val, 0, 1023, 0, 180); // map the value
    double acoshVal = acosh(angle * PI / 180); // calculate the value
    analogWrite(2, (int) acoshVal * 1023); // multiply the value by 1023
    delay(100); // delay for 100ms
}
```

- The angle should be in radians. To convert the angle from degrees to radians, multiply the angle by PI / 180.
- A domain error occurs if the argument is lower than 1. In this case, the function returns NAN.

asinh()

[Math]

```
double asinh(double x);
```

Description

Returns the principal value of the arc hyperbolic sine of *x* in radians.

Parameters

x: The angle in radians.

Returns

The principal value of the arc hyperbolic sine of *x* in radians.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the arc hyperbolic sine to the pin `OUT`.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}
void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double angle = map(val, 0, 1023, 0, 180); // map the value
    double asinhVal = asinh(angle * PI / 180); // calculate the value
    analogWrite(2, (int) asinhVal * 1023); // multiply the value by 1023
    delay(100); // delay for 100ms
}
```

- The angle should be in radians. To convert the angle from degrees to radians, multiply the angle by `PI / 180`.

atanh()

[Math]

```
double atanh(double x);
```

Description

Returns the principal value of the arc hyperbolic tangent of *x* in radians.

Parameters

x: The angle in radians.

Returns

The principal value of the arc hyperbolic tangent of *x* in radians.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the arc hyperbolic tangent to the pin `OUT`.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}
void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double angle = map(val, 0, 1023, 0, 180); // map the value
    double atanhVal = atanh(angle * PI / 180); // calculate the value
    analogWrite(2, (int) atanhVal * 1023); // multiply the value by 1023
    delay(100); // delay for 100ms
}
```

- The angle should be in radians. To convert the angle from degrees to radians, multiply the angle by `PI / 180`.
- A domain error occurs if the argument is outside the range [-1, 1]. In this case, the function returns NAN.

exp()

[Math]

```
double exp(double x);
```

Description

Returns the exponential value of *x*.

Parameters

x: The value to calculate the exponential value.

Returns

The exponential value of *x*.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the exponential value of x on pin OUT.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);   // set pin 2 as output

    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double x = map(val, 0, 1023, 0, 5); // map the value of the
    ↪  potentiometer from 0 to 1023 to the range of 0 to 5
    double y = exp(x); // calculate the exponential value of x
    int pwm = (int) map(y, 0, 148.41, 0, 1023); // map the value of y from
    ↪  0 to 148.41 to 0 to 1023
    analogWrite(2, pwm); // write the value of pwm to pin 1
}
```

The angle should be in radians. To convert the angle from degrees to radians, multiply the angle by `PI / 180`.

log()

[Math]

```
double log(double x);
```

Description

Returns the natural logarithm of *x*.

Parameters

x: The value to calculate the natural logarithm.

Returns

The natural logarithm of *x*.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the natural logarithm of x on pin OUT.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output

    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    // read the value of the potentiometer
    int val = analogRead(1);
    // map the value of the potentiometer from 0 to 1023
    // to the range of 0 to 5
    double x = map(val, 0, 1023, 0, 5);
    // calculate the natural logarithm of x
    double y = log(x);
    // map the value of y from -1.60944 to 1.60944 to 0 to 1023
    int pwm = (int) map(y, -1.60944, 1.60944, 0, 1023);
    // write the value of pwm to pin 2
    analogWrite(2, pwm);
}
```

log10()

[Math]

```
double log10(double x);
```

Description

Returns the base 10 logarithm of *x*.

Parameters

x: The value to calculate the base 10 logarithm.

Returns

The base 10 logarithm of *x*.

Example Code

The code reads the value of a potentiometer connected to pin IN and place the base 10 logarithm of x on pin OUT.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output

    pin(3, "VCC");  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    // read the value of the potentiometer
    int val = analogRead(1);
    // map the value of the potentiometer from 0 to 1023
    // to the range of 0 to 5
    double x = map(val, 0, 1023, 0, 5);
    // calculate the base 10 logarithm of x
    double y = log10(x);
    // map the value of y from -0.69897 to 0.69897 to 0 to 1023
    int pwm = (int) map(y, -0.69897, 0.69897, 0, 1023);
    // write the value of pwm to pin 2
    analogWrite(2, pwm);
}
```

pow()

[Math]

```
double pow(double x, double y);
```

Description

Returns the value of *x* to the exponent of *y*.

Parameters

x: The base value.

y: The exponent value.

Returns

The value of *x* to the exponent of *y*.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the value of x to the exponent of y on pin `OUT`.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output

    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double x = map(val, 0, 1023, 0, 5); // map the value of the
    ↪  potentiometer from 0 to 1023 to the range of 0 to 5
    double y = pow(x, 3); // calculate the value of x to the exponent of 3
    int pwm = (int) map(y, 0, 125, 0, 1023); // map the value of y from 0
    ↪  to 125 to 0 to 1023
    analogWrite(2, pwm); // write the value of pwm to pin 2
}
```

square()

[Math]

```
double square(double x);
```

Description

Returns the square value of *x*.

Parameters

x: The value to calculate the square value.

Returns

The square value of *x*.

Example Code

The code reads the value of a potentiometer connected to pin IN and place the square value of x on pin OUT.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}
void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double x = map(val, 0, 1023, 0, 5); // map the value
    double y = square(x); // calculate the square value of x
    int pwm = (int) map(y, 0, 25, 0, 1023); // map the value
    analogWrite(2, pwm); // write the value of pwm to pin 2
}
```

- This function does not belong to the standard C library.

sqrt()

[Math]

```
double sqrt(double x);
```

Description

Returns the square root of *x*.

Parameters

x: The value to calculate the square root.

Returns

The square root of *x*.

Example Code

The code reads the value of a potentiometer connected to pin IN and place the square root of x on pin OUT.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    // read the value of the potentiometer
    int val = analogRead(1);
    // map the value of the potentiometer from 0 to 1023
    // to the range of 0 to 5
    double x = map(val, 0, 1023, 0, 5);
    // calculate the square root of x
    double y = sqrt(x);
    // map the value of y from 0 to 2.23607 to 0 to 1023
    int pwm = (int) map(y, 0, 2.23607, 0, 1023);
    // write the value of pwm to pin 2
    analogWrite(2, pwm);
}
```

ceil()

[Math]

```
double ceil(double x);
```

Description

Returns the smallest integer value greater than or equal to *x*, expressed as a floating-point number.

Parameters

x: The value to calculate the ceiling.

Returns

The smallest integer value greater than or equal to *x*.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the ceiling of x on pin `OUT`.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    // read the value of the potentiometer
    int val = analogRead(1);
    // map the value of the potentiometer from 0 to 1023
    // to the range of 0 to 5
    double x = map(val, 0, 1023, 0, 5);
    // calculate the ceiling of x
    double y = ceil(x);
    // map the value of y from 0 to 5 to 0 to 1023
    int pwm = (int) map(y, 0, 5, 0, 1023);
    // write the value of pwm to pin 1
    analogWrite(2, pwm);
}
```

floor()

[Math]

```
double floor(double x);
```

Description

Returns the largest integer value less than or equal to *x*, expressed as a floating-point number.

Parameters

x: The value to calculate the floor.

Returns

The largest integer value less than or equal to *x*.

Example Code

The code reads the value of a potentiometer connected to pin IN and place the floor of x on pin OUT.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output

    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    // read the value of the potentiometer
    int val = analogRead(1);
    // map the value of the potentiometer from 0 to 1023
    // to the range of 0 to 5
    double x = map(val, 0, 1023, 0, 5);
    // calculate the floor of x
    double y = floor(x);
    // map the value of y from 0 to 5 to 0 to 1023
    int pwm = (int) map(y, 0, 5, 0, 1023);
    // write the value of pwm to pin 2
    analogWrite(2, pwm);
}
```

fabs()

[Math]

```
double fabs(double x);
```

Description

Returns the absolute value of a floating-point number *x*.

Parameters

x: The value to calculate the absolute value.

Returns

The absolute value of *x*.

Example Code

The code reads the value of a potentiometer connected to pin IN and places the absolute value of x on pin OUT.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output

    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    // read the value of the potentiometer
    int val = analogRead(1);
    // map the value of the potentiometer from 0 to 1023
    // to the range of -5 to 5
    double x = map(val, 0, 1023, -5, 5);
    // calculate the absolute value of x
    double y = fabs(x);
    // map the value of y from 0 to 5 to 0 to 1023
    int pwm = (int) map(y, 0, 5, 0, 1023);
    // write the value of pwm to pin 2
    analogWrite(2, pwm);
}
```

fmod()

[Math]

```
double fmod(double x, double y);
```

Description

Returns the remainder of the division of $\frac{x}{y}$.

Parameters

x: The dividend value. y: The divisor value.

Returns

The remainder of the division of $\frac{x}{y}$.

Example Code

The code reads the value of two potentiometers connected to pins INX and INY and place the remainder of the division of $\frac{x}{y}$ on pin OUT.

```
void setup() {
    pin(1, "INY");  // set the label of pin 1 to "INY"
    pin(2, "INX");  // set the label of pin 2 to "INX"
    pin(3, "OUT", OUTPUT);  // set pin 3 as output
    pin(4, "VCC", POWER);  // set pin 4 as power pin
    pin(5, "GND", GROUND);  // set pin 5 as ground pin
}
void loop() {
    int val_x = analogRead(1); // read the value of the potentiometers
    int val_y = analogRead(2);
    // map the values of the potentiometers from 0 to 1023
    // to the range of 0 to 10
    double x = map(val_x, 0, 1023, 0, 10);
    double y = map(val_y, 0, 1023, 0, 10);
    double z = fmod(x, y); // calculate the remainder of the division
    int pwm = (int) map(z, 0, 10, 0, 1023); // map the value
    analogWrite(3, pwm); // write the value of pwm to pin 3
}
```

modf()

[Math]

```
double modf(double x, double *pointer);
```

Description

Returns the signed fractional part of the floating-point number *x*.

This function breaks the argument *x* into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a floating-point number in the location pointed to by *pointer* and returns the fractional part.

Parameters

`x`: The value to calculate the fractional part.

`pointer`: The address of a variable to store the integral part.

Returns

The fractional part of *x*.

Example Code

The code reads the value of two potentiometers connected to pins `INX` and `INY` and place the fractional part of x on pin `OUT` and the integral part on pin `OUT2`.

```
void setup() {
    pin(1, "INY");  // set the label of pin 1 to "INY"
    pin(2, "INX");  // set the label of pin 2 to "INX"
    pin(3, "OUT", OUTPUT);  // set pin 3 as output
    pin(4, "OUT2", OUTPUT);  // set pin 4 as output

    pin(5, "VCC", POWER);  // set pin 5 as power pin
    pin(6, "GND", GROUND);  // set pin 6 as ground pin
}

void loop() {
    // read the value of the potentiometers
    int val_x = analogRead(1);
    int val_y = analogRead(2);
    // map the value of the potentiometers from 0 to 1023
    // to the range of 0 to 10
    double x = map(val_x, 0, 1023, 0, 10);
    double y = map(val_y, 0, 1023, 0, 10);
    // calculate the fractional part of x
```

```
    double z;
    double integral;
    z = modf(x, &integral);
    // map the value of z from 0 to 1 to 0 to 1023
    int pwm = (int) map(z, 0, 1, 0, 1023);
    // write the value of pwm to pin 3
    analogWrite(3, pwm);
    // map the value of integral from 0 to 10 to 0 to 1023
    pwm = (int) map(integral, 0, 10, 0, 1023);
    // write the value of pwm to pin 4
    analogWrite(4, pwm);
}
```

frexp()

[Math]

```
double frexp(double x, int *pointer);
```

Description

This function breaks a floating-point number *x* into a normalized fraction and an integral power of 2. It stores the integer in the location pointed to by *pointer*.

If *x* is a normal floating-point number, this function returns the value *v*, such that *v* has a magnitude in the interval [0.5, 1) or zero, and *x* equals *v* times 2 raised to the power of *pointer*. If *x* is zero, both parts of the result are zero. If *x* is not a finite number, this function returns *x* as is and stores zero by *pointer*.

Parameters

`x`: The value to calculate the normalized fraction and the integral power of 2. `pointer`: The address of a variable to store the integral power of 2.

Returns

The fractional part of *x*.

Example Code

The code reads the value of two potentiometers connected to pins `INX` and `INY` and place the normalized fraction of x on pin `OUT` and the integral power of 2 on pin `OUT2`.

```
void setup() {
    pin(0, "INY");  // set the label of pin 0 to "INY"
    pin(1, "INX");  // set the label of pin 1 to "INX"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "OUT2", OUTPUT); // set pin 3 as output

    pin(4, "VCC", POWER);  // set pin 4 as power pin
    pin(5, "GND", GROUND); // set pin 5 as ground pin
}

void loop() {
    // read the value of the potentiometers
    int val_x = analogRead(1);
    int val_y = analogRead(2);
    // map the value of the potentiometers from 0 to 1023
    // to the range of 0 to 10
    double x = map(val_x, 0, 1023, 0, 10);
```

```
    double y = map(val_y, 0, 1023, 0, 10);
    // calculate the normalized fraction of x
    double z;
    int exponent;
    z = frexp(x, &exponent);
    // map the value of z from 0.5 to 1 to 0 to 1023
    int pwm = (int) map(z, 0.5, 1, 0, 1023);
    // write the value of pwm to pin 3
    analogWrite(3, pwm);
    // map the value of exponent from -10 to 10 to 0 to 1023
    pwm = (int) map(exponent, -10, 10, 0, 1023);
    // write the value of pwm to pin 4
    analogWrite(4, pwm);
}
```

This function permits a zero pointer as a directive to skip a storing the exponent.

ldexp()

[Math]

```
double ldexp(double x, int exponent);
```

Description

Returns the value of *x* times 2 raised to the power of *exponent*.

Parameters

`x`: The value to calculate the fractional part.

`exponent`: The power of 2 to multiply *x*.

Returns

The value of *x* times 2 raised to the power of *exponent*.

Example Code

The code reads the value of two potentiometers connected to pins `INX` and `INY` and place the value of *x* times 2 raised to the power of *exponent* on pin OUT.

```
void setup() {
    pin(1, "INY");  // set the label of pin 1 to "INY"
    pin(2, "INX");  // set the label of pin 2 to "INX"
    pin(3, "OUT", OUTPUT);  // set pin 3 as output
    pin(4, "OUT2", OUTPUT); // set pin 4 as output

    pin(5, "VCC", POWER);  // set pin 5 as power pin
    pin(6, "GND", GROUND);  // set pin 6 as ground pin
}

void loop() {
    // read the value of the potentiometers
    int val_x = analogRead(1);
    int val_y = analogRead(2);
    // map the value of the potentiometers from 0 to 1023
    // to the range of 0 to 10
    double x = map(val_x, 0, 1023, 0, 10);
    double y = map(val_y, 0, 1023, 0, 10);
    // calculate the value of x times 2 raised to the power of y
    double z = ldexp(x, (int) y);
    // map the value of z from 0 to 10230 to 0 to 1023
    int pwm = (int) map(z, 0, 10230, 0, 1023);
    // write the value of pwm to pin 3
```

```
    analogWrite(3, pwm);
}
```

This function permits a zero pointer as a directive to skip storing the exponent.

hypot()

[Math]

```
double hypot(double x, double y);
```

Description

Returns $\sqrt{x^2 + y^2}$ without undue overflow or underflow.

This is the length of the hypotenuse of a right-angled triangle with sides of length *x* and *y*, or the distance of the point *(x, y)* from the origin..

Parameters

x: The value of the first side of the triangle.

y: The value of the second side of the triangle.

Returns

The length of the hypotenuse of a right-angled triangle with sides of length *x* and *y*.

Example Code

The code reads the value of two potentiometers connected to pins `INX` and `INY` and place the length of the hypotenuse of a right-angled triangle with sides of length *x* and *y* on pin `OUT`.

```
void setup() {
    pin(1, "INY");  // set the label of pin 1 to "INY"
    pin(2, "INX");  // set the label of pin 2 to "INX"
    pin(3, "OUT", OUTPUT);  // set pin 3 as output
    pin(4, "OUT2", OUTPUT); // set pin 4 as output

    pin(5, "VCC", POWER);  // set pin 5 as power pin
    pin(6, "GND", GROUND);  // set pin 6 as ground pin
}

void loop() {
    // read the value of the potentiometers
    int val_x = analogRead(1);
    int val_y = analogRead(2);
    // map the value of the potentiometers from 0 to 1023
    // to the range of 0 to 10
    double x = map(val_x, 0, 1023, 0, 10);
    double y = map(val_y, 0, 1023, 0, 10);
    // calculate the length of the hypotenuse of a right-angled triangle
    // with sides of length x and y
```

```
    double z = hypot(x, y);
    // map the value of z from 0 to 14.14 to 0 to 1023
    int pwm = (int) map(z, 0, 14.14, 0, 1023);
    // write the value of pwm to pin 2
    analogWrite(3, pwm);
}
```

This function permits a zero pointer as a directive to skip storing the exponent.

round()

[Math]

```
double round(double x);
```

Description

Returns the value of *x* rounded to the nearest integer value.

Parameters

x: The value to calculate the rounded value.

Returns

The value of *x* rounded to the nearest integer value as a floating-point number.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the rounded value of x on pin OUT.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "VCC", POWER); // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}
void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double x = map(val, 0, 1023, -5, 5); // map the value
    double y = round(x); // calculate the rounded value of x
    int pwm = (int) map(y, -5, 5, 0, 1023); // map the value
    analogWrite(2, pwm); // write the value of pwm to pin 2
}
```

- If x is an integral or infinite, x itself is returned.
- If x is NaN, NaN is returned.

lround()

[Math]

```
long lround(double x);
```

Description

Returns the value of *x* rounded to the nearest integer value, but rounds halfway cases away from zero.

Parameters

x: The value to calculate the rounded value.

Returns

The value of *x* rounded to the nearest integer value as a long integer.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the rounded value of x on pin `OUT`.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}
void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double x = map(val, 0, 1023, -5, 5); // map the value
    long y = lround(x); // calculate the rounded value of x
    int pwm = (int) map(y, -5, 5, 0, 1023); // map the value
    analogWrite(2, pwm); // write the value of pwm to pin 2
}
```

- If `x` is not a finite number or an overflow occurs, this function returns the `LONG_MIN` value (`0x80000000`).

lrint()

[Math]

```
long lrint(double x);
```

Description

Returns the value of *x* rounded to the nearest integer value, rounding halfway cases to the even integer value.

Parameters

x: The value to calculate the rounded value.

Returns

The value of *x* rounded to the nearest integer value as a long number.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and places the rounded value of x on pin `OUT`.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}
void loop() {
    int val = analogRead(1); // read the value of the potentiometer
    double x = map(val, 0, 1023, -5, 5); // map the value
    long y = lrint(x); // calculate the rounded value of x
    int pwm = (int) map(y, -5, 5, 0, 1023); // map the value
    analogWrite(2, pwm); // write the value of pwm to pin 2
}
```

- If `x` is not a finite number or an overflow occurs, this function returns the `LONG_MIN` value (`0x80000000`).

trunc()

[Math]

```
double trunc(double x);
```

Description

Returns the value of *x* truncated to the nearest integer not larger in absolute value.

Parameters

x: The value to calculate the truncated value.

Returns

The value of *x* truncated to the nearest integer not larger in absolute value as a floating-point number.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and places the truncated value of x on pin OUT.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    // read the value of the potentiometer
    int val = analogRead(1);
    // map the value of the potentiometer from 0 to 1023
    // to the range of -5 to 5
    double x = map(val, 0, 1023, -5, 5);
    // calculate the truncated value of x
    double y = trunc(x);
    // map the value of y from -5 to 5 to 0 to 1023
    int pwm = (int) map(y, -5, 5, 0, 1023);
    // write the value of pwm to pin 2
    analogWrite(2, pwm);
}
```

fdim()

[Math]

```
double fdim(double x, double y);
```

Description

Returns `max(x - y, 0)`.

Parameters

`x`: The value to calculate the difference. `y`: The value to subtract from `x`.

Returns

The value of `max(x - y, 0)`.

Example Code

The code reads the value of two potentiometers connected to pins `INX` and `INY` and places the value of `max(x - y, 0)` on pin `OUT`.

```
void setup() {
    pin(1, "INY");  // set the label of pin 1 to "INY"
    pin(2, "INX");  // set the label of pin 2 to "INX"
    pin(3, "OUT", OUTPUT);  // set pin 3 as output
    pin(4, "OUT2", OUTPUT); // set pin 4 as output

    pin(5, "VCC", POWER);  // set pin 5 as power pin
    pin(6, "GND", GROUND);  // set pin 6 as ground pin
}

void loop() {
    // read the value of the potentiometers
    int val_x = analogRead(1);
    int val_y = analogRead(2);
    // map the value of the potentiometers from 0 to 1023
    // to the range of 0 to 10
    double x = map(val_x, 0, 1023, 0, 10);
    double y = map(val_y, 0, 1023, 0, 10);
    // calculate the value of max(x - y, 0)
    double z = fdim(x, y);
    // map the value of z from 0 to 10 to 0 to 1023
    int pwm = (int) map(z, 0, 10, 0, 1023);
    // write the value of pwm to pin 3
    analogWrite(3, pwm);
}
```

When both x and y are NaN, NaN is returned.

fmax()

[Math]

```
double fmax(double x, double y);
```

Description

Returns the greater of the two values between x and y.

Parameters

x: The first value to compare.

y: The second value to compare.

Returns

The greater of the two values x and y.

Example Code

The code reads the value of two potentiometers connected to pins INX and INY and places the greater of the two values on pin OUT.

```
void setup() {
    pin(1, "INY");  // set the label of pin 1 to "INY"
    pin(2, "INX");  // set the label of pin 2 to "INX"
    pin(3, "OUT", OUTPUT);  // set pin 3 as output
    pin(4, "OUT2", OUTPUT); // set pin 4 as output
    pin(5, "VCC", POWER);  // set pin 5 as power pin
    pin(6, "GND", GROUND);  // set pin 6 as ground pin
}
void loop() {
    int val_x = analogRead(1); // read the value of the potentiometers
    int val_y = analogRead(2);
    analogWrite(3, (int) fmax(val_x, val_y)); // write fmax to pin 2
}
```

- If an argument is NaN, NaN is returned.
- If both arguments are NaN, NaN is returned.

fmin()

[Math]

```
double fmin(double x, double y);
```

Description

Returns the smaller of the two values between x and y.

Parameters

x: The first value to compare.

y: The second value to compare.

Returns

The smaller of the two values x and y.

Example Code

The code reads the value of two potentiometers connected to pins INX and INY and place the smaller of the two values on pin OUT.

```
void setup() {
    pin(1, "INY");  // set the label of pin 1 to "INY"
    pin(2, "INX");  // set the label of pin 2 to "INX"
    pin(3, "OUT", OUTPUT);  // set pin 3 as output
    pin(4, "OUT2",OUTPUT); // set pin 4 as output
    pin(5, "VCC", POWER);  // set pin 5 as power pin
    pin(6, "GND", GROUND);  // set pin 6 as ground pin
}
void loop() {
    int val_x = analogRead(1);  // read the value of the potentiometers
    int val_y = analogRead(2);
    double z = fmin(val_x, val_y); // get the smaller value
    analogWrite(3, (int) z); // write the value of z to pin 3
}
```

- If an argument is NaN, NaN is returned.
- If both arguments are NaN, NaN is returned.

fma()

[Math]

```
double fma(double x, double y, double z);
```

Description

Returns the value of `x * y + z` as a floating-point number.

Parameters

x: The first value to multiply.

y: The second value to multiply.

z: The value to add.

Returns

The value of `x * y + z`.

Example Code

The code reads the value of three potentiometers connected to pins `INX`, `INY` and `INZ` and places the greater of the two values on pin `OUT`.

```
void setup() {
    pin(1, "INY");  // set the label of pin 1 to "INY"
    pin(2, "INX");  // set the label of pin 2 to "INX"
    pin(3, "OUT", OUTPUT);  // set pin 3 as output
    pin(4, "OUT2", OUTPUT); // set pin 4 as output
    pin(5, "INZ");  // set the label of pin 5 to "INZ"
    pin(6, "VCC", POWER);  // set pin 6 as power pin
    pin(7, "GND", GROUND);  // set pin 7 as ground pin
}
void loop() {
    int val_x = analogRead(1); // read the value of the potentiometers
    int val_y = analogRead(2);
    int val_z = analogRead(5);
    double z = fma(val_x, val_y, val_z); // calculate the multiply-add
    ↪  operation
    analogWrite(3, (int) z); // write the value of z to pin 3
}
```

isinf()

[Math]

```
bool isinf(double x);
```

Description

Returns `true` if *x* is infinite, `false` otherwise.

Parameters

x: The value to check if it is infinite.

Returns

`true` if *x* is infinite, `false` otherwise.

Example Code

The code reads the value of two potentiometers connected to pins INX and INY and places `true` on pin OUT if the $\frac{x}{y}$ is infinite, `false` otherwise.

```
void setup() {
    pin(0, "INY");  // set the label of pin 1 to "INY"
    pin(1, "INX");  // set the label of pin 2 to "INX"
    pin(2, "OUT", OUTPUT);  // set pin 3 as output
    pin(3, "OUT2", OUTPUT); // set pin 4 as output

    pin(4, "VCC", POWER);  // set pin 5 as power pin
    pin(5, "GND", GROUND);  // set pin 6 as ground pin
}

void loop() {
    int val_x = analogRead(1);  // read the value of the potentiometers
    int val_y = analogRead(2);
    // map the value of the potentiometers from 0 to 1023
    // to the range of 0 to 10
    double x = map(val_x, 0, 1023, 0, 10);
    double y = map(val_y, 0, 1023, 0, 10);
    bool z = isinf(x / y); // check if x/y is infinite
    digitalWrite(3, z); // write the value of z to pin 3
}
```

isnan()

[Math]

```
bool isnan(double x);
```

Description

Returns `true` if *x* is NaN, `false` otherwise.

Parameters

x: The value to check if it is NaN.

Returns

`true` if *x* is NaN, `false` otherwise.

Example Code

The code reads the value of two potentiometers connected to pins `INX` and `INY` and place `true` on pin OUT if the $\frac{x}{y}$ is NaN, `false` otherwise.

```
void setup() {
    pin(1, "INY");  // set the label of pin 1 to "INY"
    pin(2, "INX");  // set the label of pin 2 to "INX"
    pin(3, "OUT", OUTPUT);  // set pin 3 as output
    pin(4, "OUT2", OUTPUT); // set pin 4 as output

    pin(4, "VCC", POWER);  // set pin 5 as power pin
    pin(5, "GND", GROUND);  // set pin 6 as ground pin
}

void loop() {
    int val_x = analogRead(1);  // read the value of the potentiometers
    int val_y = analogRead(2);
    // map the value of the potentiometers from 0 to 1023
    // to the range of 0 to 10
    double x = map(val_x, 0, 1023, 0, 10);
    double y = map(val_y, 0, 1023, 0, 10);
    bool z = isnan(x / y); // check if x/y is NaN
    digitalWrite(3, z); // write the value of z to pin 3
}
```

signbit()

[Math]

```
double signbit(double x);
```

Description

Returns the sign bit of the value x.

Parameters

x: The value to check the sign bit.

Returns

The sign bit of the value x.

Example Code

The code reads the value of a potentiometer connected to pin IN and place the sign bit of x on pin OUT.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output
    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    // read the value of the potentiometer
    int val = analogRead(1);
    // map the value of the potentiometer from 0 to 1023
    // to the range of -5 to 5
    double x = map(val, 0, 1023, -5, 5);
    // calculate the sign bit of x
    double y = signbit(x);
    // write the value of y to pin 2
    digitalWrite(2, (int) y);
}
```

copysign()

[Math]

```
double copysign(double x, double y);
```

Description

Returns the value of *x* with the sign of *y*.

Parameters

x: The value to adjust the sign of.

y: The value to copy the sign from.

Returns

The value of *x* with the sign of *y*.

Example Code

The code reads the value of two potentiometers connected to pins INX and INY and places the value of *x* with the sign of *y* on pin OUT.

```
void setup() {
    pin(1, "INY");  // set the label of pin 1 to "INY"
    pin(2, "INX");  // set the label of pin 2 to "INX"
    pin(3, "OUT", OUTPUT);  // set pin 3 as output
    pin(4, "OUT2", OUTPUT); // set pin 4 as output

    pin(5, "VCC", POWER);  // set pin 5 as power pin
    pin(6, "GND", GROUND);  // set pin 6 as ground pin
}

void loop() {
    // read the value of the potentiometers
    int val_x = analogRead(1);
    int val_y = analogRead(2);
    // map the value of the potentiometers from 0 to 1023
    // to the range of -10 to 10
    double x = map(val_x, 0, 1023, -10, 10);
    double y = map(val_y, 0, 1023, -10, 10);
    // calculate the value of x with the sign of y
    double z = copysign(x, y);
    // map the value of z from -10 to 10 to 0 to 1023
    int pwm = (int) map(z, -10, 10, 0, 1023);
    // write the PWM value to pin 2
```

```
    analogWrite(3, pwm);
}
```

map()

[Math]

```
double map(double x, double in_min, double in_max, double out_min, double
↪  out_max);
```

Description

Maps the value of *x* from the range of *in_min* to *in_max* to the range of *out_min* to *out_max*.

Parameters

`x`: the value to be mapped.

`in_min`: the minimum value of the input range.

`in_max`: the maximum value of the input range.

`out_min`: the minimum value of the output range.

`out_max`: the maximum value of the output range.

Returns

The mapped value of *x*.

Example Code

The code reads the value of a potentiometer connected to pin `IN` and place the cosine value mapped to the pin `OUT`.

```
void setup() {
    pin(1, "IN");   // set the label of pin 1 to "IN"
    pin(2, "OUT", OUTPUT);  // set pin 2 as output

    pin(3, "VCC", POWER);  // set pin 3 as power pin
    pin(4, "GND", GROUND);  // set pin 4 as ground pin
}

void loop() {
    // read the value of the potentiometer
    int val = analogRead(1);
    // map the value to the range of 0 to 360
    double angle = map(val, 0, 1023, 0, 360);
    // calculate the cosine of the angle in radians
    double cosVal = cos(angle * PI / 180);
    // map the value to the range of 0 to 1023
    analogWrite(2, cosVal * 1023);
```

```
    // delay for 100ms
    delay(100);
}

double map(double x, double in_min, double in_max, double out_min, double
↪  out_max) {
    // maps the value of x from the range of in_min
    // to in_max to the range of out_min to out_max
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) +
    ↪  out_min;
}
```