Lessons: 00.Arduino-examples

02. Digital

Version: 0.0.7, Date: 2016-10-26

Contents

- Blink Without Delay
  - Time To Blink Function
- Button
- State Change Detect
- Debounce
  - Debounce Class
- Digital Input Pullup

Blink Without Delay

```
#include "Cosa/OutputPin.hh"
#include "Cosa/RTT.hh"

OutputPin led(Board::LED);

const uint32_t BLINK_INTERVAL_MILLIS = 1000L;
uint32_t previousBlinkMillis = 0;

void setup()
{
  RTT::begin();
}

void loop()
{
  // Check if the blink interval has expired
  uint32_t currentMillis = RTT::millis();
  if ((currentMillis - previousBlinkMillis) >= BLINK_INTERVAL_MILLIS) {
    previousBlinkMillis = currentMillis;
    led.toggle();
  }
}
```

Blink Without Delay Notes

- Function(s) can be used to abstract the code and make it more readable.
- The `loop()` should become easier to understand and extend.
- Easier to understand how to copy this pattern to create addition periodic actions (e.g. Debounce).

Time To Blink Function

```
#include "Cosa/OutputPin.hh"
#include "Cosa/RTT.hh"

OutputPin led(Board::LED);

const uint32_t BLINK_INTERVAL_MILLIS = 1000L;
uint32_t previousBlinkMillis = 0;

bool timeToBlink();

void setup()
{
  RTT::begin();
}

void loop()
{
  if (timeToBlink()) led.toggle();
}
```

Time To Blink Function

```
bool timeToBlink()
{
  // Get current milli-seconds
  uint32_t currentMillis = RTT::millis();

  // Return false if the blink interval has not expired
  if ((currentMillis - previousBlinkMillis) < BLINK_INTERVAL_MILLIS)
    return (false);

  // Return true if expired, set the new latest time stamp
  previousBlinkMillis = currentMillis;
  return (true);
}
```

Button

```
#include "Cosa/InputPin.hh"
#include "Cosa/OutputPin.hh"

InputPin button(Board::D2);
OutputPin led(Board::LED);

void loop()
{
  led = button;                          // Read button and write led
}
```

State Change Detect

```
#include "Cosa/InputPin.hh"
#include "Cosa/OutputPin.hh"
#include "Cosa/UART.hh"
#include "Cosa/IOStream.hh"

// Button pin, push count and state
InputPin button(Board::D2);
uint16_t buttonPushCounter = 0;
bool previousButtonReading = 0;

// LED pin
OutputPin led(Board::LED);

IOStream ios(&uart);

void setup()
{
  uart.begin(9600);
}
```

Cont. State Change Detect

```
void loop()
{
  // Read the button pin and check for change
  bool buttonReading = button;
  if (buttonReading != previousButtonReading) {

    // Check if the transition is from off to on
    if (buttonReading) {
      ios << ++buttonPushCounter << PSTR(": on ");
    }
    // Otherwise from on to off
    else {
      ios << PSTR("off") << endl;
    }
    delay(50);
  }
  // Save reading
  previousButtonReading = buttonReading;

  // Blink every fourth button push
  led = (buttonPushCounter % 4 == 0);
}
```

Debounce

```
#include "Cosa/InputPin.hh"
#include "Cosa/OutputPin.hh"
#include "Cosa/RTT.hh"

InputPin button(Board::D2);
OutputPin led(Board::LED, 1);

// Debounced button state
bool buttonState = false;
bool previousButtonReading = false;

// Latest debounce time in milli-seconds
uint32_t previousDebounceMillis = 0;

// Debounce delay in in milli-seconds
const uint32_t DEBOUNCE_DELAY_MILLIS = 50;

void setup()
{
  RTT::begin();
}
```

Cont. Debounce

```
void loop()
{
  // Get current clock and button pin
  uint32_t currentMillis = RTT::millis();
  bool reading = button;

  // Restart the debounce timer if button pin changed
  if (reading != previousButtonReading) {
    previousDebounceMillis = currentMillis;
  }

  // Check if the debounce delay has expired
  else if ((currentMillis - previousDebounceMillis) > DEBOUNCE_DELAY_MILLIS) {

    // Check the button state did actually change
    if (reading != buttonState) {
      buttonState = reading;

      // Toggle LED on rising transition of button
      if (buttonState) {
        led.toggle();
      }
    }
  }

  // Save the button pin reading
  previousButtonReading = reading;
}
```

Debounce Notes

- Has much the same structure as Blink Without Delay.
- Must wait a debounce delay period to determine if the button state has actually changed.
- Refactor to a Debounce Class to make the pattern easier to reuse.

Debounce Class

```
#include "Debounce.h"
#include "Cosa/OutputPin.hh"
#include "Cosa/RTT.hh"

Debounce button(Board::D2);
OutputPin led(Board::LED);

void setup()
{
  RTT::begin();
}

void loop()
{
  if (button.released()) led.toggle();
}
```

Cont. Debounce Class

```
#ifndef DEBOUNCE_H
#define DEBOUNCE_H

#include "Cosa/RTT.hh"
#include "Cosa/InputPin.hh"

class Debounce {
public:
  Debounce(Board::DigitalPin pin);
  operator bool();
  bool changed();
  bool pushed();
  bool released();
protected:
  const uint32_t DEBOUNCE_DELAY_MILLIS = 50L;
  InputPin m_pin;
  bool m_state;
  bool m_previousReading;
  uint32_t m_previousDebounceMillis;
};

#endif
```

Cont. Debounce Class

```
/**
 * Construct debounced input pin with given board pin.
 * @param[in] pin digital pin.
 */
Debounce(Board::DigitalPin pin) :
  m_pin(pin),
  m_state(false),
  m_previousReading(false),
  m_previousDebounceMillis(0)
{}

/**
 * Return the current debounced input pin state.
 * @return bool.
 */
operator bool()
{
  return (m_state);
}
```

Cont. Debounce Class

```
/**
 * Check if the debounce pin state has changed. Should be frequently
 * called to allow debounce logic to work. Returns true(1) if the
 * state was changed otherwise false(0).
 * @return bool.
 */
bool changed()
{
  // Get current clock and read input pin
  uint32_t currentMillis = RTT::millis();
  bool reading = m_pin;
  bool res = false;

  // Restart the debounce timer if input pin changed
  if (reading != m_previousReading) {
    m_previousDebounceMillis = currentMillis;
  }

  // Check if the debounce delay has expired
  else if ((currentMillis - m_previousDebounceMillis) > DEBOUNCE_DELAY_MILLIS) {
    if (reading != m_state) {
      m_state = reading;
      res = true;
    }
  }

  // Save the input pin reading
  m_previousReading = reading;
  return (res);
}
```

Cont. Debounce Class

```
/**
 * Check if the input pin was pushed (falling). Should be frequently
 * called to allow debounce logic to work. Returns true(1) if the
 * button was pushed otherwise false(0).
 * @return bool.
 */
bool pushed()
{
  return (changed() && !m_state);
}

/**
 * Check if the input pin was released (rising). Should be frequently
 * called to allow debounce logic to work. Returns true(1) if the
 * button was released otherwise false(0).
 * @return bool.
 */
bool released()
{
  return (changed() && m_state);
}
```

Digital Input Pullup

```
#include "Cosa/InputPin.hh"
#include "Cosa/OutputPin.hh"
#include "Cosa/UART.hh"
#include "Cosa/IOStream.hh"

OutputPin led(Board::LED, 0);
InputPin button(Board::D2, InputPin::PULLUP_MODE);
IOStream ios(&uart);

void setup()
{
  uart.begin(9600);
}

void loop()
{
  bool reading = button;
  ios << reading << endl;
  led = !reading;
  delay(100);
}
```

Digital Input Pullup Notes

- The `InputPin` contructor has an optional mode parameter:
  - `InputPin::NORMAL_MODE` (default)
  - `InputPin::PULLUP_MODE`
- The `OutputPin` constructor has also an optional initial value parameter. The default value is `0`.

License: LGPL-2.1