Documentation of Library KK-Switch

License and Copyright

Introduction

A library for handling any kinds of digital inputs from switches, push buttons, rotary encoders and a lot of other devices. Several options are offered for an easy and performant handling.

The main target was to get an easy understandable code and the chance to concentrate on the real functionality of the project. All functionality for the digital inputs is encapsulated in a separate class and only changes of the input need attention.

Different digital inputs can be handled completely independent in parallel with Switch objects and the raw digital inputs and the resulting outputs are named „states“ of a switch.

Advantages

- no active waits
- high performance
- low memory usage
- handling current and previous state
- state caching enables stable state analysis
- easy handling in loops with little code
- state mapping (optional)
- negative input pin logic (optional)
- software debouncing (optional)
- reduction of raw state reads (optional)
- sequence analysis (optional)
- raw input state change via interrupt possible (with HW debouncing)
- easy alternative to callback functions available
- sequence analysis available for push buttons and rotary encoders (optional)
- push button events like single, double, long single and long repeat available (optional)

Installation

The library can be installed via Arduino Library Manager or by downloading the archive from directory „library“ and unpacking the archive in IDEs libraries directory.

Getting started

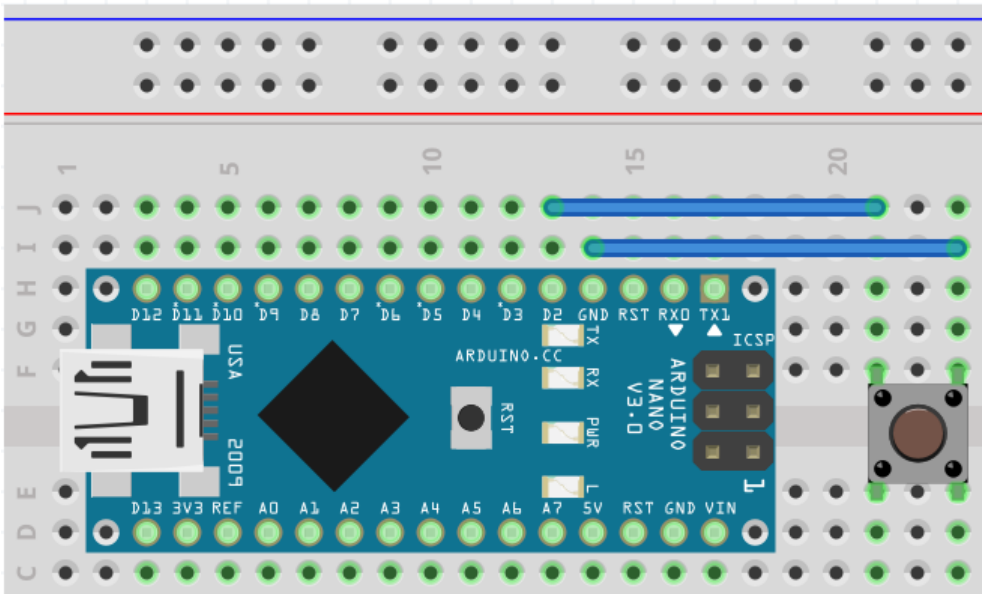
Only a few lines of code are necessary to understand the logic behind:

```
#include <Switch.h>
#define INPUT_PIN 2

// use 2 states, no mapping, pin 2, no read wait, debouncing, negative logic
Switch sw = Switch(2, false, INPUT_PIN, 0, 20, true);

void setup() {

  sw.configurePin();

}

void loop() {
  if(sw.hasChanged()){
    // do what you want, whenever the digital signal changes
  }
}
```

See the following examples and more details below.

Examples

Based on a typical easy electrical design (like shown in the following picture with Arduino Nano and a push button)



several examples are included to show the usage and possibilities.

Simple

```
/*
```

```
  Copyright (c) 2025 Kay Kasper
  under the MIT License (MIT)
*/

#include <Switch.h>

/*
  Simple example with a push button that is connected
  on one side to GND and on the other side to INPUT_PIN.
  (inverse or negative logic)
*/

#define INPUT_PIN 2

Switch sw = Switch(2, false, INPUT_PIN, 0, 20, true);

void setup() {
  Serial.begin(9600);
  // set pinMode
  sw.configurePin();
}

void loop() {
  // identify changes when button is pushed
  if(sw.hasChanged()){
    // print the current state (1=was pushed, 0=was released)
    Serial.println(sw.getState());
  }
}
```

SimpleLED

Simple example, where button push is directly contolling the build in LED (button pressed = LED on).

Prerequisite is a push button that is connected on one side to GND and on the other side to INPUT_PIN (inverse or negative logic). LED is assumed to be always available at pin LED_BUILTIN.

Mapping

Mapping-Example, that shows how to use mapping values and different variants of instantiation.

Prerequisite is a push button that is connected on one side to GND and on the other side to INPUT_PIN (inverse or negative logic).

CallbackAlternative

Example, where button push state change sequence is contolling the build in LED by using a PushButtonRepeatAnalyzer.

Single push: switch on/off permanent light (toggle)

Long push: blinking while button is pushed. after releasing long push, LED returns to state on or off as was before long push.

It is shown, how easy the implementation of function calls in case of changes with Switch library is. This is as easy as call back implementations of other libraries.

Prerequisite is a push button that is connected on one side to GND and on the other side to INPUT_PIN (inverse or negative logic). LED is assumed to be always available at pin LED_BUILTIN.

Interrupt

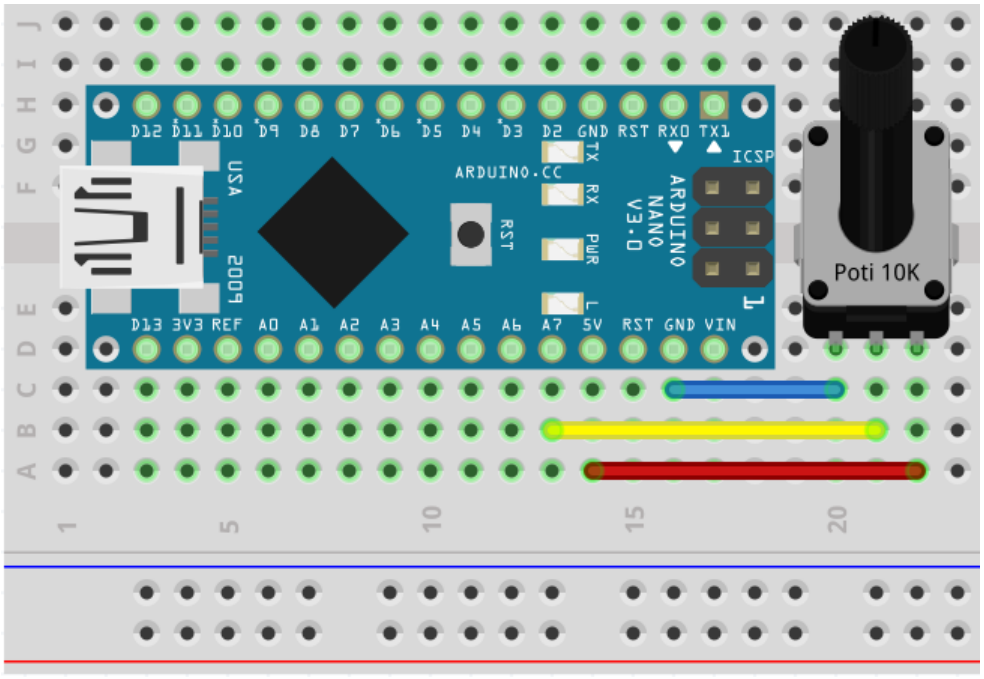
Example for using an interrupt as trigger for changes. A button push is directly contolling the build in LED (button pressed = LED on).

Prerequisites:

A push button that is connected on one side to GND and on the other side to INPUT_PIN (inverse or negative logic). LED is assumed to be always available at pin LED_BUILTIN. Hardware debouncing instead of software debouncing is strongly recommended for the input signale and interrupt. Software debouncing and readCycleMillis waits will fail because of missing continously calls in the loop.

AnalogSwitch

Here we need another electrical setup with a potentiometer instead of a push button.



Example with a new subclass of Switch to generate raw input states based on an analog input pin. The variable value of the analog pin will change the speed of blinking of the build in LED.

Prerequisite is an potentiometer/attenuator connected with the middle pin (variable voltage) to an analog input pin. LED is assumed to be always available at pin LED_BUILTIN.

PushButtonSingleDoubleLong

Example, where button push state change sequence is contolling the build in LED:

Single push: switch on permanent light

Double push: switch off permanent light

Long push: switch on/off blinking light (toggeling)

Prerequisite is a push button that is connected on one side to GND and on the other side to INPUT_PIN (inverse or negative logic). LED is assumed to be always available at pin LED_BUILTIN.

PushButtonSingleRepeat

Example, where button push state change sequence is contolling the build in LED:

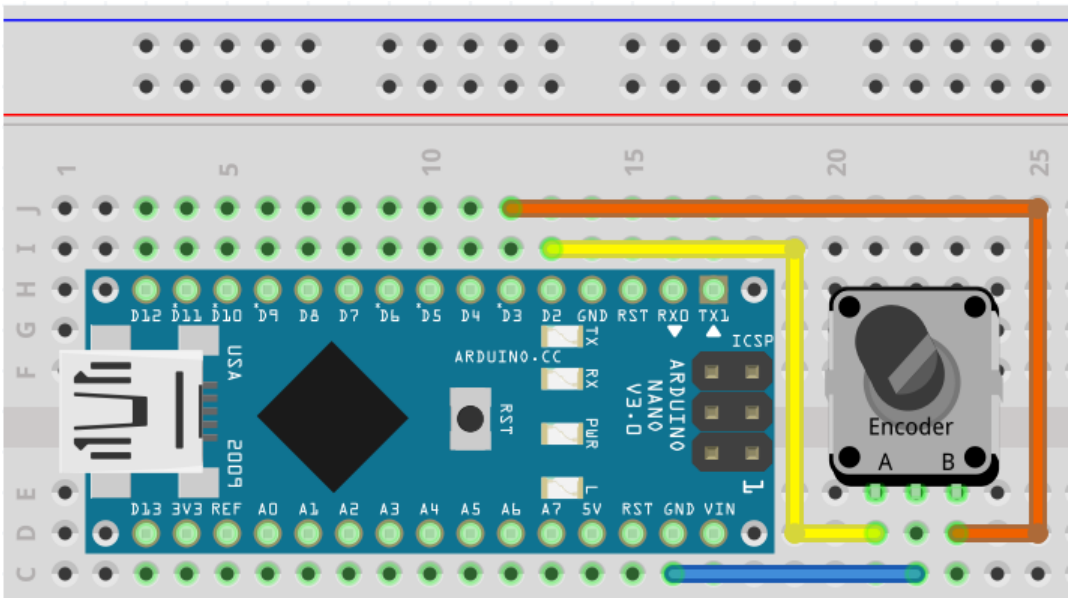
Single push: switch on/off permanent light (toggle)

Long push: blinking while button is pushed. after releasing long push, LED returns to state on or off as was before long push.

Prerequisite is a push button that is connected on one side to GND and on the other side to INPUT_PIN (inverse or negative logic). LED is assumed to be always available at pin LED_BUILTIN.

RotaryEncoder

Here we need another electrical setup with a rotary encoder instead of a push button.



Example with a subclass of Switch for rotary encoders.

The turning of the rotary encoder will change the speed (+/-) of blinking of the build in LED.

Prerequisite is an rotary encoder connected with the pins A and B to two digital input pins, that pull the level to GND (negative logic) if connected. LED is assumed to be always available at pin LED_BUILTIN.

Hardware debouncing is recommended due to fast raw input state changes. Software debouncing is to slow or will be not precise enough. But SW debouncing is used here to simplify the electrical design.

Switch

Class for handling digital input signals typically generated by switches, push buttons, rotary encoders and other binary signals generating objects.

Input and output information is named here as states of the switch.

The raw inputs are named raw states and are read from one or more digital input pins, where pins are HIGH or LOW, but the states need to be numbers >= 0 in a sequence.

The default implementation is based on one input pin with two raw states

- 0 (SW_STATE_DEFAULT_OFF)
- 1 (SW_STATE_DEFAULT_ON).

The class can handle up to 6 input pins with up to 64 raw states by overwriting the function getRawState() and setting parameter numStates accordingly.

The evaluation of output state changes is done only in the function hasChanged() and it is the only function, that calls getRawState(), to read the input states. Therefore the function has to be called in the programm loop at least once, to identify changes as fast as necessary.

Only when hasChanged() deliveres true, it is normally interresting to react on that. Between calls of hasChanged() the output states are stable and will not change. The functions getState(), getMappedState(), getPrevState() and getPrevMappedState() will always return the output value, that was identified and internally set by the last call of hasChanged().

If raw input states are based on slow operations (e.g. analog pins and analog values that are used as calculation base for the defined input stated) the number of getRawState() calls can be reduced to a minimum.

The class can behave in two different ways, depending on the instantiation with or without a subclass of SwitchStateAnalyzer.

Without a subclass of SwitchStateAnalyzer it handles any kind of states directly with equal number of raw input states and output states (= numState).

Without a subclass of SwitchStateAnalyzer input states (function getRawstate()) and output states (functions getState() and getPrevState()) have the same values from 0 up to numStates-1. But with defining an optional mapping value for each output state, input and output state (getMappedState() and getPrevMappedState()) values can be different. The mapping must be defined previously before using it and costs some extra memory (numState bytes). The mapping values may make a further processing of state changes easier.

With a subclass of SwitchStateAnalyzer it handles any kind of digital raw input states and hands them for further processing over to the state analyzer with different number and meaning of raw input states and output states. The value of numState is here equal to the number of output states. E.g. a sequence of 2 push button raw input states can be transformed to 4 push button events like no push, single push, double push or long pushs or 4 input raw states (values 0 to 3 of pin A and B) of an rotary encoder can be transformed to 3 events like no rotation, one rotation step right and one rotation step left.

Parameters

Name	Description
ssa	Mandatory pointer to an SwitchStateAnalyzer (or subclass) variable
numState	Number of output states. Without given state analyzer number of output and raw input states are the same. Values from 2 to 64 allowed.
enableMapping	Defines, if output states shall be mapped to other values and be prepared for later mapping definition.
bufferMapping	Pointer to the memory area for mapping values. At least numState bytes must be available there, if pointer is given. Value 0 means no memory available and no mapping enabled.
inputPin	The pin to read raw input states from in standard implementation. The pin must be configured (INPUT or INPUT_PULLUP) suitable before first usage depending on invertRaw.
readCycleMillis	Minimum number of millis to wait betweens repeating raw input state reads if <> 0. Value 0 means each hasChanged() reads raw input state
debounceMillis	Minimum number of millis to wait during debouncing. It is the time between first input state change detection and final input read. Value 0 means that the internal software debouncing is not used.
invertRaw	Defines, if raw input states shall be inverted internally (before further processing) due to inverse electrical configuration. E.g. if true and numStates=4 we invert input 0,1,2,3 to 3,2,1,0
state	The output state (values from 0 to numState-1) that shall be mapped. If out of range or mapping not enabled, no mapping will be set.
mappingValue	The mappping value for a state

Constants

SW_STATE_UNDEFINED	state unclear / not defined yet
SW_STATE_DEFAULT_OFF	state for input and output, LOW / not pushed
SW_STATE_DEFAULT_ON	state for input and output, HIGH / pushed

SwitchStateAnalyzer

Class for analyzing sequences of raw state changes of digital input like push buttons within a Switch class.

The Switch class handles the raw states of a switch, push buttons or digital input. The SwitchStateAnalyzer or sub classes can handle a sequence of raw states over time to identify and deliver more complex and completly different information (states). It gives an easy extension for transformations of simple raw inputs to complex output information with good structuring of functionality.

Apart from a default implementation of the class SwitchStateAnalyzer 2 other implementations are available for analyzing sequences for single pushes, double pushes, single long pushes and repeating long pushes.

PushButtonDoubleLongAnalyzer

Implementation of an analyzer, that can analyzing sequences for single pushes, double pushes and single long pushes. Supports up to 65 seconds push duration. Based on 1 digital input pin with 2 raw input states.

Parameter names	Description
maxDoubleMillis	Maximun time in milliseconds for completing a double push. Value 0 means, that double push analysis is switched off
minLongMillis	Minimum time in milliseconds that a push must last to be identified as long push. Value 0 means, that long push analysis is switched off
endLongByTime	Only relevant for long push analysis. Defines if long push should end based on minLongMillis time (tue) or SW_STATE_DEFAULT_OFF input (false)

Constants for output states	Description
PBDL_STATE_E_OFF	no push
PBDL_STATE_E_SINGLE	single push completed
PBDL_STATE_E_DOUBLE	double push completed
PBDL_STATE_E_LONG	long push completed

PushButtonRepeatAnalyzer

Implementation of an analyzer, that can analyzing sequences for single pushes and repeating long pushes. Continous push delivers an output of two toggeling states, as long as the button is pushed. Supports up to 65 seconds push duration. Based on 1 digital input pin with 2 raw input states.

Parameter names	Description
longStartMillis	Time in milliseconds that is used to differentiate between single and continous push. Push durations < longStartMillis mean singe push, otherwise continous push. Value = 0 means that no continous pushes are recognized Range from 0 to 2000 allowed.
repeatMillis	Duration in milliseconds of each phase A and B during continous pushes Value = 0 means that no continous pushes are recognized Range from 0 to 2000 allowed.

Constants for output states	Description
PBR_STATE_E_OFF	no push
PBR_STATE_E_SINGLE	single push completed
PBR_STATE_E_CONT_A	continous push phase A completed
PBR_STATE_E_CONT_B	continous push phase B completed

RotaryEncoderAnalyzer

Implementation of an analyzer, that can analyze sequences for rotations of rotary encoders. Based on 2 digital input pins (A and B) with 4 raw input states.

Constants for output states	Description
REA_STATE_E_OFF	no rotation
REA_STATE_E_DIRECTION_R	right turn one step completed
REA_STATE_E_DIRECTION_L	left turn one step completed

Rotary Encoder

Implementation of a specific switch for rotary encoders. Subclass of class Switch that needs to be instantiated with an RotaryEncoderAnalyzer.

Analysis of rotary encoder input signals from 2 digital pins (4 raw states) to deliver the information (3 states) which movements were done:

- no movement (REA_STATE_E_OFF)
- right turn one step (REA_STATE_E_DIRECTION_R)
- left turn on step (REA_STATE_E_DIRECTION_L)

@seealso Switch.h

@seealso RotaryEncoderAnalyzer