```
/*
 *
 * Dahlander.ino
 *
 * This sketch aims at replacing a Dahlander switch with an Arduino Pro Mini
 * steering 3 relays.
 *
 * Intended purpose
 * ----------------
 * It is intended as a replacement for a 3-position switch doing 0-3000rpm-6000rpm-0
 * on a Lurem wood machine. This switch is natively equipped with a mechanical device
 * preventing  going backwards, which would damage the equipment if spinning brutally
 * down to 3000 rpm from 6000rpm.
 * Remember that the cinetic energy varies with the square of the rotating speed,
 * so the machine is designed to spin down freely to zero from 6000 rpm.
 *
 * Lurem is long gone as a company, some vendors still have spare parts, but the original
 * switch is nowhere to be found. Some of these vendors have switches doing
 * 0-3000rpm-6000rpm-3000rpm-0 without the mechanical device preventing from going from
 * 6000 to 3000 rpm. As already mentioned, this is not adequate. Additionally, these sell
 * for around 200 EUR, and it is cheaper to implement it this way.
 *
 * Dahlander motors
 * ----------------
 * Dahlander motors are asynchronous 3-phase motors, usually wired to allow a ratio of two
 * between the low speed and the high speed. Other ratios are possible, though uncommon.
 * See: http://educypedia.karadimov.info/library/173.pdf , page 3 for an explanation on
 * Dahlander motors.
 *
 * It is EXTREMELY important to ensure that the shorting of 1U/1V/1W is done before applying
 * power to the motor, or we risk a very quick destruction of the motor. This is why we apply
 * the relay timing to ensure proper relay switch order.
 *
 * Overall design
 * --------------
 * One pushbutton is used to cycle as per the original design : 0-3000rpm-6000rpm-0, with
 * a short push each time (aka 'Click'). Additionally, a long press (> 2s) allows spinning
 * down from 3000 or 6000 rpm to 0.
 *
 * Timeouts are inserted :
 * - a 30s spin up timeout preventing any change while the machine is spinning up to 3000 rpm
 * - a 1 mn spin down timeout preventing the machine restart while being spinned down
 *   from 3000 rpm
 * - a 2 mn spin down timeout preventing the machine restart while being spinned down
 *   from 6000 rpm
 * - 200 ms timeouts inserted between relay changes. This is done so that we are sure we do
 *   not have a risk of wrong temporary connection while relays are changing state
 *
 * The timeouts introduce a security that did not exist in the original switch, so that
 * it is not possible to spin the machine up again while still rotating.
 *
 * Electronic design
 * -----------------
 * The device uses 3 BJTs to drive the 3 relays, as each relay draws 120 mA, way too much
 * the Arduino outputs. Also the relays are 24V, while the Arduino is a 5V device.
 *
 * As for the push button, we insert a 100 nF in parallel to improve de-bouncing
 * De-bouncing is implemented inside the OneButton library, which simplifies the external
 * electronics.
 *
 * The push button is an industrial type, big enough to fit the machine buttons. A much
 * smaller one would do, but would not be as resistant.
 *
 * Setup the circuit:
 * - Connect a pushbutton to pin D2 (ButtonPin) and ground.
 * - Connect relays K1, K2, K3 to pins D3, D4 and D5 respectively
 *
 * Libraries used
 * --------------
 * The code makes use of the OneButton library to detect short and long button press,
 * as this library makes it simple to implement a finite state machine.
 * See: http://www.mathertel.de/Arduino/OneButtonLibrary.aspx
 *
 * It also makes use of the SimpleTimer library in order to implement non-blocking delays
 * which call a pre-defined routine when the timeout has expired.
 * See: http://playground.arduino.cc/Code/SimpleTimer
 *
 * Additionally, we also use the SyncLED library in order to display current state
 * by blinking the built-in LED on D13. This is for debug purposes.
 * See: https://code.google.com/p/arduino-library-syncled/
 *
 * Relative to blinking the on-board LED (D13), we blink the stable state, and this is done
 * through coding stable states with numbers in the 1..15 range, while the sub-state number
 * is coded into the upper nibble. We lose the unstable state number, but the indication
 * is sufficient for our purposes.
 *
 * Copyright (c) by Jean-Noel Simonnet - March 2014
 * This work is licensed under a GPL V3 license.
 *
 */


#include "OneButton.h"
#include "SimpleTimer.h"
#include "SyncLED.h"

/*
 * Push button SW1 on pin D2
 * Relay K1 on pin D3
 * Relay K2 on pin D4
 * Relay K3 on pin D5
 */
#define SW1 2
#define K1 3
#define K2 4
#define K3 5

/*
 * Timings in ms, respectively:
 * - delay between relay changes
 * - delay (ms) when spinning up at 3000 rpm
 * - spin down time (ms) from 3000 rpm
 * - spin down time (ms) from 6000 rpm
 *
 * Adjust the last 3 to match the actual machine timings
 */
#define DELAY_RELAY    200L
#define DELAY_SPINUP   30000L
#define DELAY_HALT3000 60000L
#define DELAY_HALT6000 120000L

/*
 * Click and press timings
 * - Click = button press followed by 600 ms of idle time
 * - double click = two button presses within 600 ms
 * - press = button pressed for more than 2 s
 */
#define CLICK_TICKS    600
#define PRESS_TICKS    2000

// Finite machine states
typedef enum {
  ACTION_START     =0x01, // state at power up
  ACTION_3000RPM   =0x02, // machine rotating @3000 rpm
  ACTION_6000RPM   =0x03, // machine rotating @6000 rpm
  ACTION_HALT3000  =0x04, // machine halting from 3000 rpm
  ACTION_HALT6000  =0x05, // machine halting from 6000 rpm
  ACTION_3000RPM_1 =0x12, // machine is being spinned up to 3000 rpm
  ACTION_6000RPM_1 =0x13, // Moving to 6000rpm, intermediate state 1
  ACTION_6000RPM_2 =0x23,
  ACTION_HALT6000_1=0x15,
}
MyActions;

// Setup a new OneButton on pin D2
OneButton button(SW1, true);

// We need only one timer, as the Finite State Machine only takes one state at a time here
SimpleTimer timer;

// On-board LED to display which state we are
SyncLED status (13);
```

```
MyActions action = ACTION_START; // no action when starting


/*
 * Setup code here, to run once
 */
void setup() {
  // pullup on button
  pinMode (SW1, INPUT_PULLUP);
  pinMode (K1 , OUTPUT);
  pinMode (K2 , OUTPUT);
  pinMode (K3 , OUTPUT);

  // This delay is required, as otherwise we have a push button event
  // due to the 10K pullup + 100nF capacitor on SW1 (time constant = 1ms)
  delay (10); // This delay is required, as otherwise we have a push button event

  // Make sure the initial state is displayed
  status.blinkPattern ((byte)(action &0x0F));

  // Configure click and press timings
  button.setClickTicks (CLICK_TICKS);
  button.setPressTicks (PRESS_TICKS);

  // link the myClickFunction function to be called on a click event.
  button.attachClick(myClickFunction);

  // link the myPressFunction function to be called on a press event.
  button.attachPress(myPressFunction);
} //setup


/*
 * main code here, to run repeatedly:
 */
void loop() {
  unsigned long now = millis();

  // keep watching the push button:
  button.tick();

  // keep updating the timer
  timer.run();

  // update state indicator
  status.update();

  // Set outputs corresponding to each state; it does not matter
  // that we reapply them at each loop, as they remain the same.
  if (action == ACTION_START) {
    // All relays off initially
    digitalWrite (K1, LOW);
    digitalWrite (K2, LOW);
    digitalWrite (K3, LOW);

  } else if (action == ACTION_3000RPM_1) {
    digitalWrite (K2, LOW);
    digitalWrite (K3, LOW);
    digitalWrite (K1, HIGH);

  } else if (action == ACTION_3000RPM) {
    digitalWrite (K2, LOW);
    digitalWrite (K3, LOW);
    digitalWrite (K1, HIGH);

  } else if (action == ACTION_6000RPM_1) {
    digitalWrite (K1, LOW);
    digitalWrite (K3, LOW);
    digitalWrite (K2, LOW);

  } else if (action == ACTION_6000RPM_2) {
    digitalWrite (K1, LOW);
    digitalWrite (K3, HIGH);
    digitalWrite (K2, LOW);
  } else if (action == ACTION_6000RPM) {
    digitalWrite (K1, LOW);
    digitalWrite (K3, HIGH);
    digitalWrite (K2, HIGH);

  } else if (action == ACTION_HALT3000) {
    // All relays off
    digitalWrite (K1, LOW);
    digitalWrite (K2, LOW);
    digitalWrite (K3, LOW);

  } else if (action == ACTION_HALT6000_1) {
    digitalWrite (K2, LOW);
    digitalWrite (K3, HIGH);
    digitalWrite (K1, LOW);

  } else if (action == ACTION_HALT6000) {
    // All relays off
    digitalWrite (K2, LOW);
    digitalWrite (K3, LOW);
    digitalWrite (K1, LOW);

  }

} // loop


/*
 * The following functions are call-backs which perform
 * the state transitions.
 */

// Called when the button was pressed 1 time and them some time has passed.
void myClickFunction() {
  if (action == ACTION_START) {
    action = ACTION_3000RPM_1;
    timer.setTimeout (DELAY_SPINUP, myTimerEvent);

  } else if (action == ACTION_3000RPM) {
    action = ACTION_6000RPM_1;
    timer.setTimeout (DELAY_RELAY, myTimerEvent);

  } else if (action == ACTION_6000RPM) {
    action = ACTION_HALT6000_1;
    timer.setTimeout (DELAY_RELAY, myTimerEvent);
  }

  status.blinkPattern ((byte)(action &0x0F));
} // myClickFunction

// Called when the button was pressed for a long time (> 2s)
void myPressFunction() {
  if (action == ACTION_3000RPM) {
    action = ACTION_HALT3000;
    timer.setTimeout (DELAY_HALT3000, myTimerEvent);

  } else if (action == ACTION_6000RPM) {
    action = ACTION_HALT6000_1;
    timer.setTimeout (DELAY_RELAY, myTimerEvent);

  } else if (action == ACTION_3000RPM_1) {
    action = ACTION_HALT3000;

  }
  status.blinkPattern ((byte)(action &0x0F));
} // myPressFunction

// Called on double click events
void myDoubleClickFunction() {
  // Unused in this example
}

// Called on timer events
void myTimerEvent() {
  if (action == ACTION_HALT3000) {
    action = ACTION_START;
  } else if (action == ACTION_HALT6000) {
    action = ACTION_START;

  } else if (action == ACTION_3000RPM_1) {
    action = ACTION_3000RPM;

  } else if (action == ACTION_6000RPM_1) {
    action = ACTION_6000RPM_2;
    timer.setTimeout (DELAY_RELAY, myTimerEvent);

  } else if (action == ACTION_6000RPM_2) {
    action = ACTION_6000RPM;

  } else if (action == ACTION_HALT6000_1) {
    action = ACTION_HALT6000;
    timer.setTimeout (DELAY_HALT6000, myTimerEvent);

  }
  status.blinkPattern ((byte)(action &0x0F));
} // myTimerEvent
```