Programming the Magnet Switch yourself

Altimeter Cloud

The Magnet Switch is an open product. The code, schematic and parts list are all published, so if you fancy changing how it behaves you're welcome to. Here's how to program it, and the code to put back if you want to return to ours.

What you'll need

- **An Adafruit UPDI Friend.** There's a standard one and a High Voltage version, and either will do the job here. In the UK The Pi Hut and Pimoroni stock the High Voltage one. Adafruit have a full guide if you want to know more about it.
- **The Arduino IDE.** It's free from arduino.cc.
- **Three thin wires** to reach the board.

Our online programmer is for our ESP32 devices such as the Mercury, so it can't program the Magnet Switch. The ATtiny816 has to go through Arduino and the UPDI Friend.

Connecting it up

UPDI Friend	Magnet Switch
UPDI	PR
GND	Either – pad
PWR	3V

Set the voltage switch on the UPDI Friend to 3V before you connect anything, and disconnect your battery so the Friend is the only thing powering the board.

DO NOT USE 5V

The 3V hole goes straight onto the board's 3.3 V rail, so the UPDI Friend's voltage switch must be set to 3V.

Setting up Arduino

1. Open File > Preferences and add this to Additional boards manager URLs: `http://drazzy.com/package_drazzy.com_index.json`
2. Open Tools > Board > Boards Manager, search for megaTinyCore and click Install.
3. Under Tools > Board > megaTinyCore, pick the entry that covers the ATtiny816, the one without Optiboot.
4. Set the Tools menu options as shown in the table below.
5. Under Tools > Programmer pick `SerialUPDI - SLOW: 57600 baud` .
6. Plug in the UPDI Friend and pick its port under Tools > Port.
7. Upload with Sketch > Upload Using Programmer.

Tools option	Setting
Chip	ATtiny816
Clock	1 MHz internal
millis() timer	Disabled
Startup time	4 ms
WDT timeout	Disabled

Leave everything else at its default. In particular don't change the UPDI/reset pin option, as anything other than UPDI locks out the standard UPDI Friend and you'd need the High Voltage one to get back in.

The pins

Pin	What it does
PA1	Output. HIGH turns the output on.
PB0	LED. HIGH lights it.
PA4	Buzzer.

PA2	Power button and the PB hole. HIGH when pressed.
PA3	Magnet sensor. LOW when a magnet is present.
PA0	UPDI, the PR hole.

All of the other pins are unused and are driven LOW to save power.

How the code works

It's a small program. Each time round the loop it checks the inputs, acts on them and goes back to standby sleep, and the RTC's periodic interrupt wakes it again a second later. The on or off state is saved at EEPROM address 1 so that it survives losing power.

If you change it, keep the sleep in there. The chip draws around 900 µA awake against a few µA asleep, so the time it spends awake is what decides how long a battery lasts.

The code

This is our current code. Copy it into a new Arduino sketch, and upload it again any time you want to go back to it.

```
/*
  Rocketry Ltd Magnetic switch V1 code
  Alice Bowen & Neil Bowen, Rocketry Ltd 2025

  Contact: https://modelrockets.co.uk/contacts/
  Cloud data page: https://modelrockets.co.uk/cloud/

  Product support page:
  https://modelrockets.co.uk/cloud/support/manuals/mag_switch_v1/

  The Magnetic use a ATtiny816 microcontroller at it's core.
  There is a programming pin labelled "PR" on the board.
  You can program this device with the Adafruit HV UDPI Friend boardmand arduino IDE.

  ATTINY SUGGESTED SETUP
  CLOCK:          1MHz Internal
  millis() TIMER:   DISABLED
  STARTUP TIME:    4ms
  WDT Timeout:     DISABLED

  License:  You are free to modify and use our code for your own personal uses as required.

  Licences for includes:
        Please review the licence terms for each library and their creator before
        any redistrubution or commercial use. It seems that these are all OK
        to use regardless, but please don't take our word for it!
*/
#include <EEPROM.h>
#include <avr/io.h>
#include <avr/sleep.h>
#include <avr/interrupt.h>
#include <avr/power.h>

// Variables required
byte state = 0;
int action = 0;
unsigned long last_beep = 0;
uint32_t loops = 0;   // 32 bit unsigned so the count cannot overflow in the life of the device

void setup() {
  ADC0.CTRLA &= ~ADC_ENABLE_bm;  // Ensure ADC is shut down as not used to save power

  //  Read the last state of the switch from EEPROM address 0x01
  state = EEPROM.read(0x01);

  // If the state EEPROM entry is corrupt (not 0 or 1) then resave default state of 0
  if((state != 0) && (state != 1)){
    state = 0;
    EEPROM.write(0x01, state);
```

```
  }

  setup_pins();

  // Re-apply the last state when the device is powered up
  if(state == 0){
    digitalWrite(PIN_PA1, LOW);
    digitalWrite(PIN_PB0, LOW);
  }
  else {
    digitalWrite(PIN_PA1, HIGH);
    digitalWrite(PIN_PB0, LOW);   // LED stays off between flashes
  }

  while (RTC.STATUS > 0) {
    ;
  }
  RTC.CLKSEL = RTC_CLKSEL_INT32K_gc;
  RTC.PITCTRLA = RTC_PERIOD_CYC32768_gc
               | RTC_PITEN_bm;

  RTC.CTRLA = RTC_RUNSTDBY_bm; // Enable the RTC to run in Standby sleep mode
  RTC.PITINTCTRL = RTC_PI_bm; // Enable Periodic Interrupt
  sei(); // Enable global interupts.
}
ISR(RTC_PIT_vect) {
  RTC.PITINTFLAGS = RTC_PI_bm;
}

void loop() {
  loops++;

  // Detect if the Magnet is present, or button pressed, or external button pressed.
  if((digitalRead(PIN_PA2) == 1) || (digitalRead(PIN_PA3) == 0)){
    action++;   // Count number of cycles without break that the action is ON.
  }
  else {
    action = 0;
  }

  // If action trigger has been active for 3 cycles then perform the required action. (Requires the user to hold the button or magnet for 3 cycles to avoid accidental detection).
  if(action >= 2){
    action = 0;

    if(state == 0){
      // Turn the switch ON as current state is 0
      state = 1;
      EEPROM.write(0x01, state);    // Write new state to EEPROM
      digitalWrite(PIN_PA1, HIGH);  // OUTPUT
      digitalWrite(PIN_PB0, HIGH);  // LED
      tone(PIN_PA4, 4000, 200);     // 4 short buzzer tones to indicate power on
      delay(203);
      tone(PIN_PA4, 8000, 200);
      delay(203);
      tone(PIN_PA4, 4000, 200);
      delay(203);
      tone(PIN_PA4, 8000, 200);
      delay(203);
      tone(PIN_PA4, 3000, 200);
      delay(203);
      digitalWrite(PIN_PB0, LOW);   // LED off after the tune, it only flashes from here
    }
    else {
      // Turn the switch OFF as the current state is 1
      state = 0;
      EEPROM.write(0x01, state);    // Write new state to EEPROM
      digitalWrite(PIN_PA1, LOW);   // OUTPUT
      digitalWrite(PIN_PB0, LOW);   // LED
```

```
    tone(PIN_PA4, 4000, 2000);    // One long buzzer tone to indicate power off
    delay(2020);
   }

   // Prevent the action from flip-flopping by required a 5 second delay between actions. Pause here.
   // The pause sleeps between wakes so the capacitors hold through it, and the LED keeps flashing if the switch is ON.
   for(byte i = 1; i <= 6; i++){
    sleep_until_wake();
    if((state == 1) && ((i & 1) == 1)){
     flash_led();
    }
   }
  }

  // Short LED flash every other wake if the switch is ON
  if((state == 1) && ((loops & 1) == 0)){
   flash_led();
  }

  // Short beep every 6 seconds if the switch is ON
  if((state == 1) && (loops >= 10)){
   loops = 0;
   tone(PIN_PA4, 4000, 40);
   delay(40);
   tone(PIN_PA4, 5000, 30);
   delay(30);
  }
  // sleep for ~ 1 second
  delay(5);
  sleep_until_wake();
}
void sleep_until_wake() {
  digitalWrite(PIN_PA4, LOW);  // Make sure the buzzer is not left driven if a tone is still running when the timer stops in standby
  set_sleep_mode(SLEEP_MODE_STANDBY);
  sleep_enable();
  sleep_cpu();
  sleep_disable();
}
void flash_led() {
  digitalWrite(PIN_PB0, HIGH);
  delay(15);
  digitalWrite(PIN_PB0, LOW);
}
void setup_pins() {
  pinMode(PIN_PB0, OUTPUT);    // LED
  pinMode(PIN_PA1, OUTPUT);    // OUTPUT DRIVER
  pinMode(PIN_PA4, OUTPUT);    // BUZZER
  pinMode(PIN_PA2, INPUT);     // POWER BUTTON INPUT
  pinMode(PIN_PA3, INPUT);     // MAGNET SWITCH SENSOR INPUT
  pinMode(PIN_PC0, OUTPUT);
  pinMode(PIN_PC1, OUTPUT);
  pinMode(PIN_PC2, OUTPUT);
  pinMode(PIN_PC3, OUTPUT);
  pinMode(PIN_PA5, OUTPUT);
  pinMode(PIN_PA6, OUTPUT);
  pinMode(PIN_PA7, OUTPUT);
  pinMode(PIN_PB1, OUTPUT);
  pinMode(PIN_PB2, OUTPUT);
  pinMode(PIN_PB3, OUTPUT);
  pinMode(PIN_PB4, OUTPUT);
  pinMode(PIN_PB5, OUTPUT);
  digitalWrite(PIN_PC0, LOW);
  digitalWrite(PIN_PC1, LOW);
  digitalWrite(PIN_PC2, LOW);
  digitalWrite(PIN_PC3, LOW);
  digitalWrite(PIN_PA5, LOW);
  digitalWrite(PIN_PA6, LOW);
  digitalWrite(PIN_PA7, LOW);
```

```
  digitalWrite(PIN_PB1, LOW);
  digitalWrite(PIN_PB2, LOW);
  digitalWrite(PIN_PB3, LOW);
  digitalWrite(PIN_PB4, LOW);
  digitalWrite(PIN_PB5, LOW);
}
```