Sleep Modes on ATTiny85

When using microcontroller in battery-based applications we really need to look at current consumption. This will affect the power consumption and hence the lifetime of the device.

This post shows some tests I performed to try and reduce the current consumption of an ATtiny85 AVR microcontroller using sleep modes and switching off various aspects of the IC when they are not required.

The test results and code for th Arudino IDE is shown here.

Test circuit

I needed a test circuit to check both the correct functioning of the different inputs and outputs and also for comparison of the current consumed when using different power modes.

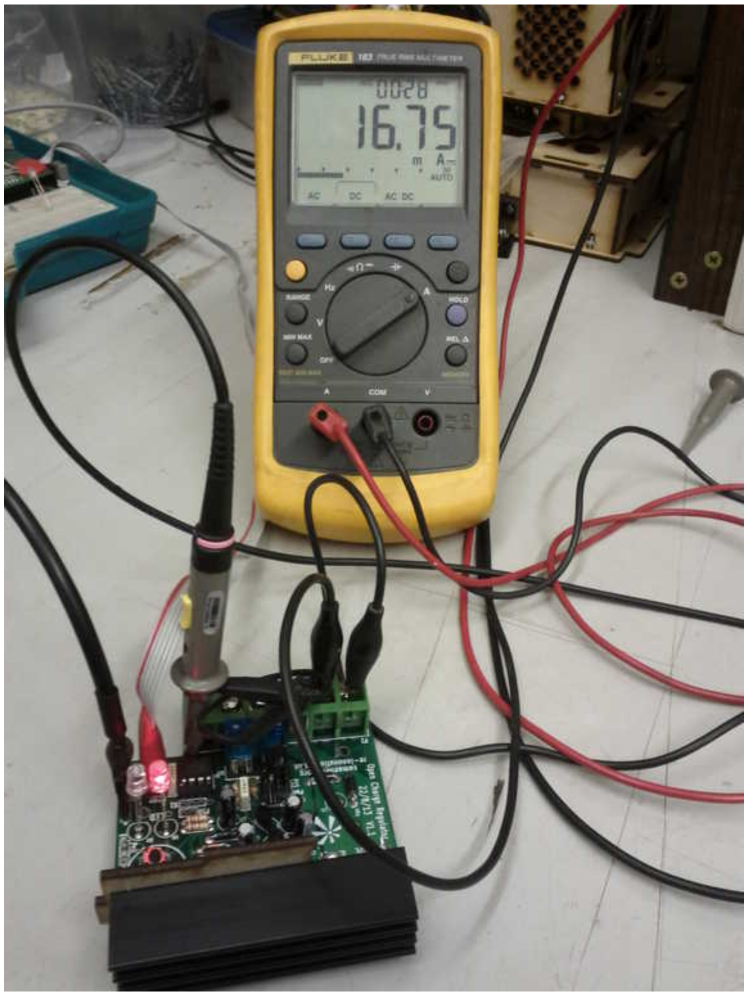
My test circuit comprised of:

- Software serial output every 1 second on pin 2 (Arudino 3)
- Fast PWM 32kHz output at 50% duty on pin 5 (Arduino 0)
- On/Off 0.5s output on pin 6 (Arduino 1)
- Analog read every 1 second on pin 7 (Arduino A1)
- FET output being driven at 32kHz (Fast PWM) at 50% duty (but NO load) on pin 3 (Arduino 4)

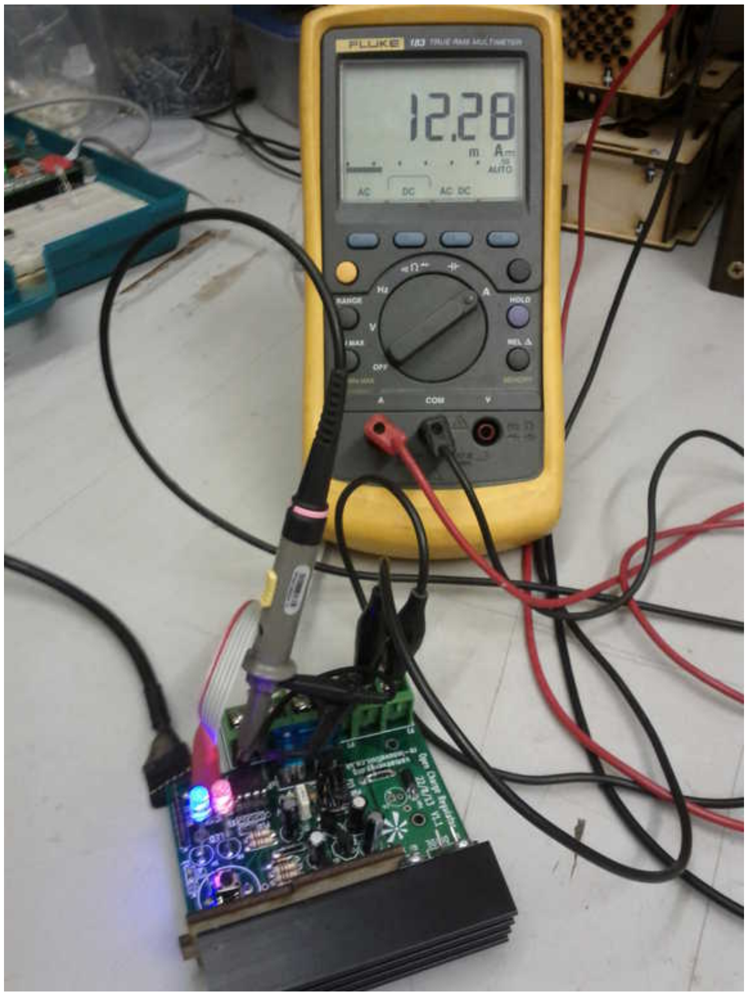
I wanted all of this to function at all times. I built this circuit and rigged up a Fluke multimeter to accurately measure the current.

I used a test input voltage of 16.0V DC from a bench power supply. This is similar to the maximum voltage from a 12V lead acid battery.

Using this test circuit with no power saving I found that the current consumption was on average 16-17mA:



Interestingly, when I did not have a serial output lead attached, the current consumption dropped to 10-13mA. This means that driving a serial output requires an extra 5-7mA. I am unsure why, apart from current required on the output Tx pin? Any ideas anyone?



While not a huge current draw this would still be a high current draw on a small lead-acid battery. This is designed for a renewable energy powered system and hence every uA matters.

Power reduction

There are a few ways we can save power. These include reducing the operating frequency, reducing the operating voltage, put the unit into sleep mode when not required, switch off any internal microcontroller modules when not in use.

I did not want to reduce the operating frequency as this would affect the PWM frequency output and the serial output. I did not want to reduce the operating voltage as this would require a re-design of the FET driver. So I have the option of putting the IC into sleep mode and switching off any modules which are not used.

ATTiny Sleep Modes

From the ATtiny25/45/85 data sheet (http://www.atmel.com/images/atmel-2586-avr-8-bit-microcontroller-attiny25-attiny45-attiny85_datasheet.pdf) we can look at section 7 which refers to Power Management and Sleep Modes.

There are also a couple of interesting videos here (http://www.youtube.com/watch?v=Ob5fHhPDqvU%20%20) and here (http://www.youtube.com/watch?v=Zwo_xuC5A48) from InsideGadgets. And its always worth reading up from the Arudino website about putting the 'normal' arduino (ATmega328) to sleep here (http://playground.arduino.cc/Learning/arduinoSleepCode).

Another article on testing the Arduino for power modes here (http://www.gammon.com.au/forum/?id=11497).

There are three sleep modes on the ATTiny25/45/85 as shown in the data and copied here:

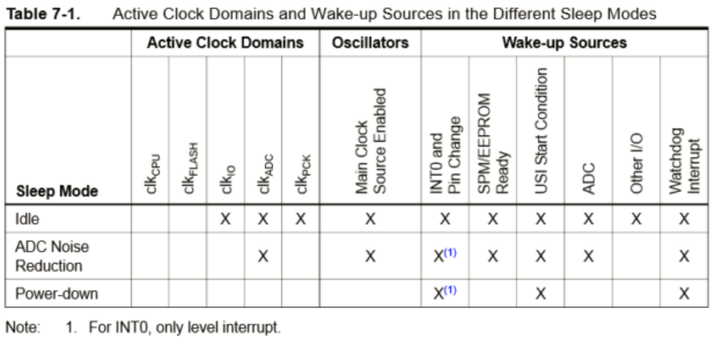


Table 7-1. Active Clock Domains and Wake-up Sources in the Different Sleep Modes

	Active Clock Domains					Oscillators	Wake-up Sources					
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{PCK}	Main Clock Source Enabled	INT0 and Pin Change	SPM/EEPROM Ready	USI Start Condition	ADC	Other I/O	Watchdog Interrupt
Idle			X	X	X	X	X	X	X	X	X	X
ADC Noise Reduction				X		X	X[1]	X	X	X		X
Power-down							X[1]		X			X

Note: 1. For INT0, only level interrupt.

ATTiny power management

There are also a number of 'modules' which can also be switched off to save even more power. These are also mentioned in the data sheet and include:

- Analog to digital converter – I switch this off before going to sleep.
- Analog comparator – In Power_Down sleep mode this is disabled automatically.
- Brown-out detector – I use this to check if voltage has dropped. This is set with fuses internal to the microcontroller
- Internal voltage reference – This is required for the Brown-out detection.
- Watchdog timer- This is used to wake from sleep so cannot switch off.
- The port pins – These are all set to input before sleeping to help save power

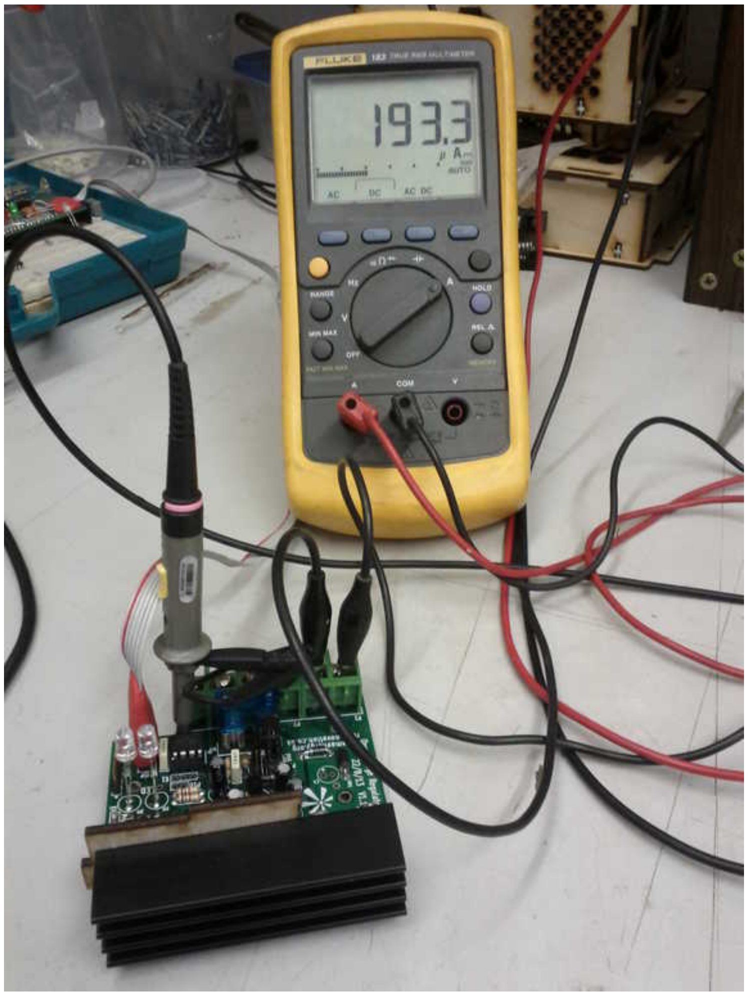
Arduino and ATTiny in sleep

In order to put the ATTiny to sleep we need to include two libraries: avr/sleep.h and avr/interrupt.h. One is required to give us access to the sleep functions and the the other is required to control the interrupts to awake from sleep.

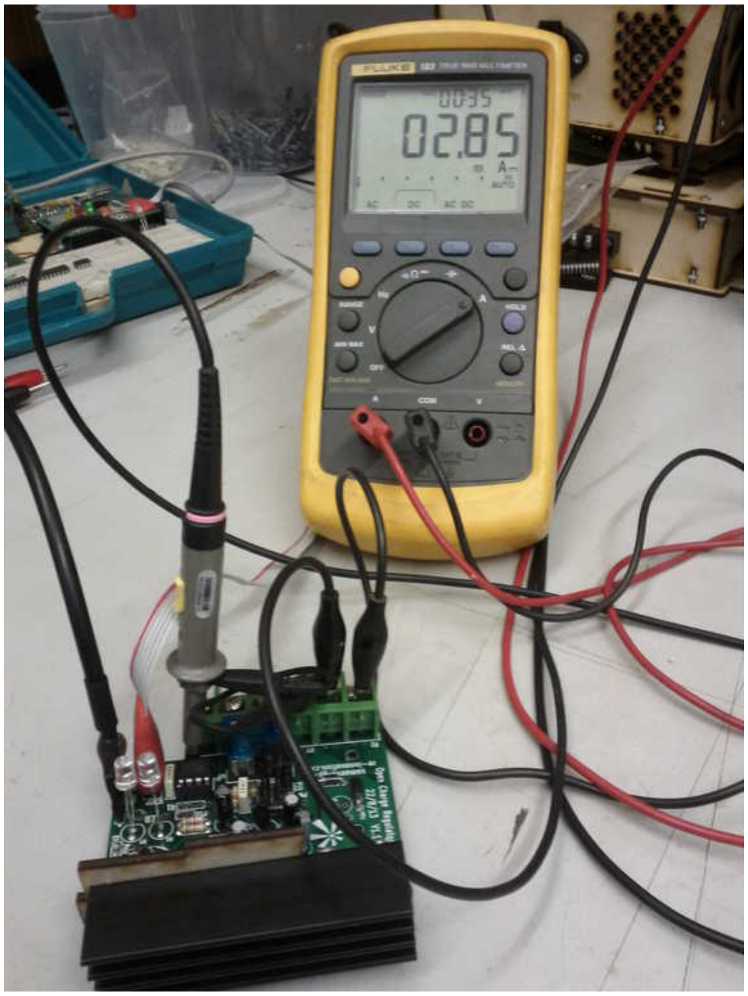
To start with I just want to do a power down and see what happens and the current reduction.

Using the following code I set the ATtiny to sleep for 4 seconds and then do all the normal functions (ADC read, Fast PWM output, write to the serial port, light the LED for 0.5 seconds), then go back to sleep.

Putting the IC to sleep massively changed the power consumption. It now consumed around 200uA (0.2mA) when in sleep, with an average of around 2mA, including the LED ON time.



Yey! In sleep much less current consumption (the reading is in uA…)



Averaging over 30ish seconds we gat an average of 2.85mA. This depends upon what is switched on when the unit is not asleep, but is much better than our previous value of 12mA.

Arduino code for ATTiny85

Here is the sample code including the sleep function. There are a few things to note here: We use the watchdog timer to wake up from sleep, so no external interrupt is required. When we do wake up due to the watchdog timer the watchdog flag is set, which allos our main code to run and then go back to sleep.

Putting the outpins into input before sleeping mode saves LOTs of current. When I did not do this the current consumption was still around 4mA when in sleep mode.

```
/*
 ATtiny85 sleep mode test

 Overview:

 This code reads in analog voltage, outputs a Fast PWM and switches ON/OFF an LED.
 I want to investigate sleep modes to see power consumption and savings.
 This is to reduce the energy consumed so it can be attached to a lead acid battery.

 This code is now fully powered down and only wakes up with the watchdog timer

 This code is designed to run on the ATTiny 25/45/85
 The serial output only works with the larger ATTiny85 IC

 The connections to the ATTiny are as follows:
 ATTiny    Arduino    Info
 Pin  1  - 5          RESET / Rx (Not receiving any data)
 Pin  2  - 3          Tx for serial conenction
 Pin  3  - 4          FET driver (PWM)
 Pin  4  -            GND
 Pin  5  - 0          RED LED (PWM)
 Pin  6  - 1          GREEN LED
 Pin  7  - 2 / A1     Vsensor (Analog)
 Pin  8  -   +Vcc
 See www.samaenergy.org/www.re-innovation.co.uk for more details including flow code

 14/11/13 by Matt Little (matt@re-innovation.co.uk/www.re-innovation.co.uk)

 Added code from (http://www.insidegadgets.com/2011/02/05/reduce-attiny-power-consumptio
 Thanks to:
 * KHM 2008 / Lab3/  Martin Nawrath nawrath@khm.de
 * Kunsthochschule fuer Medien Koeln
 * Academy of Media Arts Cologne
 *
 * Modified on 5 Feb 2011 by InsideGadgets (www.insidegadgets.com)
 * to suit the ATtiny85 and removed the cbi( MCUCR,SE ) section
 * in setup() to match the Atmel datasheet recommendations

 Updated:
 14/11/13   - Added Power_Down Sleep mode - Matt Little


 This example code is in the public domain.
 */

#define F_CPU 8000000  // This is used by delay.h library

#include <stdlib.h>
#include <EEPROM.h>

#include <avr/io.h>        // Adds useful constants
#include <util/delay.h>    // Adds delay_ms and delay_us functions

#include <avr/sleep.h>
#include <avr/interrupt.h>

// Routines to set and claer bits (used in the sleep code)
#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif
#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

// *********** Define I/O Pins *********************
// LED output pins:
const int redled = 0;         // Red LED attached to here (0, IC pin 5)
const int buzzLedSw = 1;       // Green LED/buzzer/Switch attached to here (1, IC pin 6)
// MOSFET Driver output
const int FETdriver = 4;
// Analog sensing pin
const int VsensePin = A1;    // Reads in the analogue number of voltage
// Only use Serial if using ATTiny85
// Serial output connections:
#include <SoftwareSerial.h>
#define rxPin 5    // We use a non-existant pin as we are not interested in receiving da
#define txPin 3
SoftwareSerial serial(rxPin, txPin);

#define INTERNAL2V56NC (6)  // We use the internal voltage reference

//************ USER PARAMETERS***********************
//**********MODE*************************************
const int deviceType = 85;  // 45 = ATTiny45, NO serial output, 85 = AtTiny85, with seri

// Variables
unsigned long int averageSensor = 0;  // This holds the average value of the sensor

// Varibales for EEPROM
int hiByte;      // These are used to store longer variables into EEPROM
int loByte;

// Varibles for the calibration factor
int calibrationFactor = 0;    // This holds the Vref value in millivolts

// Variables for the Sleep/power down modes:
volatile boolean f_wdt = 1;

// the setup routine runs once when you press reset:
void setup()  {

  // Set up FAST PWM
  TCCR0A = 2<<COM0A0 | 2<<COM0B0 | 3<<WGM00;  // Set control register A for Timer 0
  TCCR0B = 0<<WGM02 | 1<<CS00;  // Set control register B for Timer 0
  TCCR1 = 0<<PWM1A | 0<<COM1A0 | 1<<CS10;  // Set control register for Timer 1
  GTCCR = 1<<PWM1B | 2<<COM1B0;  // General control register for Timer 1

  // Set up IO pins
  pinMode(rxPin, INPUT);
  pinMode(txPin, OUTPUT);
  pinMode(redled, OUTPUT);
  pinMode(buzzLedSw, OUTPUT);        // First want to read the switch
  pinMode(FETdriver, OUTPUT);

  digitalWrite(FETdriver, LOW);   //  Switch the FET OFF

  if(deviceType==85)
  {
    // Start the serial output string - Only for ATTiny85 Version
    serial.begin(4800);
    serial.println("SLEEP ATTiny85");
    serial.println("14/11/13 Matt Little");
  }

  analogReference(INTERNAL2V56NC);  // This sets the internal ref to be 2.56V (or close
  delay(100);  // Wait a while for this to stabilise.

  // Read in the Calibration Factor
  hiByte = EEPROM.read(124);
  loByte = EEPROM.read(125);

  calibrationFactor = (hiByte << 8)+loByte;  // Get the sensor calibrate value

  serial.print("Calibration Factor: ");   // TESTING
  serial.println(calibrationFactor);   // TESTING

  // Set the Fast PWM output for the Red LED
  analogWrite(redled, 127);    // Set it to 50%  running at 31.2kHz
  analogWrite(FETdriver, 127);    // Set it to 50%  running at 31.2kHz

  setup_watchdog(8); // approximately 0.5 seconds sleep
}

// the loop routine runs over and over again forever:
void loop()  {

  if (f_wdt==1) {  // wait for timed out watchdog / flag is set when a watchdog timeout
    f_wdt=0;       // reset flag

    digitalWrite(buzzLedSw, HIGH);      // GREEN LED ON
    _delay_ms(500);  // Switch the LED on for 0.5 Seconds

    averageSensor = analogRead(VsensePin);
    averageSensor = (averageSensor*calibrationFactor)/1024;   // This is the int of the
    serial.print("Analog Voltage reading: ");
    serial.println(averageSensor);

    digitalWrite(buzzLedSw, LOW);  // Green LED OFF

    // Set the ports to be inputs - saves more power
    pinMode(txPin, INPUT);
    pinMode(redled, INPUT);
    pinMode(buzzLedSw, INPUT);        // First want to read the switch
    pinMode(FETdriver, INPUT);

    system_sleep();  // Send the unit to sleep

    // Set the ports to be output again
    pinMode(rxPin, INPUT);
    pinMode(txPin, OUTPUT);
    pinMode(redled, OUTPUT);
    pinMode(buzzLedSw, OUTPUT);        // First want to read the switch
    pinMode(FETdriver, OUTPUT);

  }
}


// set system into the sleep state
// system wakes up when wtchdog is timed out
void system_sleep() {

  cbi(ADCSRA,ADEN);                    // switch Analog to Digitalconverter OFF

  set_sleep_mode(SLEEP_MODE_PWR_DOWN); // sleep mode is set here
  sleep_enable();

  sleep_mode();                        // System actually sleeps here

  sleep_disable();                     // System continues execution here when watchdog

  sbi(ADCSRA,ADEN);                    // switch Analog to Digitalconverter ON

}

// 0=16ms, 1=32ms,2=64ms,3=128ms,4=250ms,5=500ms
// 6=1 sec,7=2 sec, 8=4 sec, 9= 8sec
void setup_watchdog(int ii) {

  byte bb;
  int ww;
  if (ii > 9 ) ii=9;
  bb=ii & 7;
  if (ii > 7) bb|= (1<<5);
  bb|= (1<<WDCE);
  ww=bb;

  MCUSR &= ~(1<<WDRF);
  // start timed sequence
  WDTCR |= (1<<WDCE) | (1<<WDE);
  // set new watchdog timeout value
  WDTCR = bb;
  WDTCR |= _BV(WDIE);
}

// Watchdog Interrupt Service / is executed when watchdog timed out
ISR(WDT_vect) {
  f_wdt=1;  // set global flag
}
```

This allows me to reduce the power consumption of my project (the open source charge controller project (https://www.re-innovation.co.uk/web12/index.php/en/projects/open-charge-regulator)).

Now I need to write a program flow which will put the unit into sleep mode as often as possible.

Edit 28/1/14:

Here is a very good tutorial on sleep modes for the Arduino:

http://donalmorrissey.blogspot.co.uk/2010/04/putting-arduino-diecimila-to-sleep-part.html (http://donalmorrissey.blogspot.co.uk/2010/04/putting-arduino-diecimila-to-sleep-part.html)

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

POST COMMENT

(https://www.re-innovation.co.uk/)
(https://www.re-innovation.co.uk/)

Re-Innovation
(https://www.re-innovation.co.uk/)

(https://www.re-innovation.co.uk/)

Renewable Energy Innovation specialise in electrical and electronic systems for renewable energy projects, mainly solar, wind and micro-hydro. We focus on renewable energy based stand-alone power supply systems (off-grid systems). This includes power and energy monitoring, battery charge control and wiring systems. Please contact us to talk about your project. If you require consultancy, design and implementation services for your renewable energy project please contact us.

(https://www.curiouselectric.co.uk/)
(https://www.curiouselectric.co.uk/)

Curious Electric Company

(https://www.curiouselectric.co.uk/)
(https://www.curiouselectric.co.uk/)

The Curious Electric Company specialise in creating electronic kits that help you monitor and measure the world around you, with clever data loggers and sensors you can take the pulse of your planet, collect and share data and knowledge with the world. Be informed - stay curious!

(https://www.re-innovation.co.uk/bespoke)
(https://www.re-innovation.co.uk/bespoke)

Bespoke Gear

(https://www.re-innovation.co.uk/bespoke)
(https://www.re-innovation.co.uk/bespoke)

Our sister business, Bespoke Gear have been building interactive pedal-power systems for over 15 years. Doing events and selling equipment to a wide variety of customers in that time we have learnt what works. We design and build high-quality, robust pedal-powered equipment and bespoke, interesting and interactive displays. If you are trying to promote environmental awareness, energy efficiency, getting people fit or just want something thats a bit different for your event then we might have something for you.