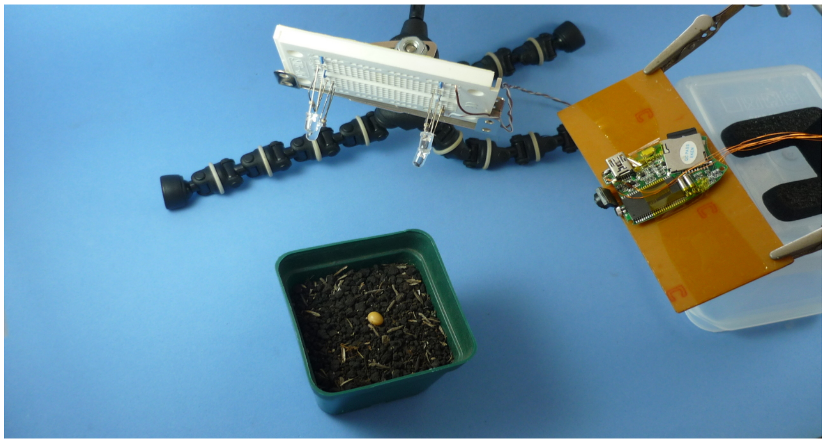
LOW BUDGET
TIME LAPSE
CAMERA



KEYCHAIN CAMERA

This diminute mini cameras were built into multiple forms like keychains, gum pack, pen, cigarette lighter, etc. All basically have a CMOS picture sensor, image processor and a rechargeable battery to record video (with audio!) and to take still pictures, usually using a removable micro SD card.

The build quality of those cameras is very low (their price is usually under 5 USD/EUR/GBP) so they fails often. The rechargeable battery like in other portable electronic devices is their Aquiles Tallon. If there's a battery failed camera somewhere lying in a drawer, it's time to give a second chance as a time lapse camera with very low budget additional components.

The project's features are:

* Programmable time intervals
* Very low power
* Automatic flash
* External battery
* Low budget (could be 0)

EXTERNAL CONNECTIONS

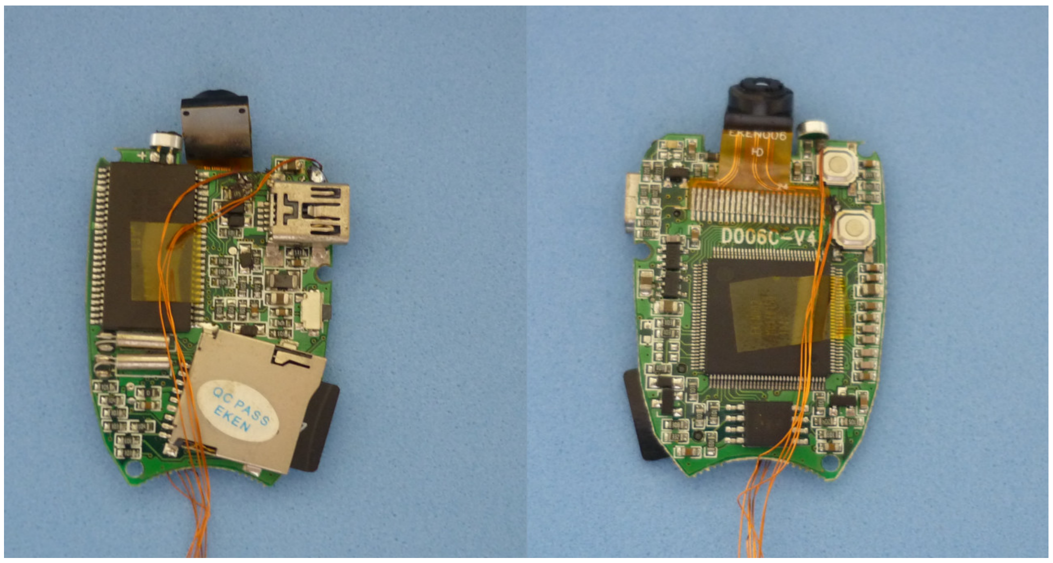
The camera has two push-buttons to set up the different operating modes, like video recording or still pictures taking. Usually one of the two buttons, when pressed for some amount of time powers on the camera , and if pressed again for the same amount of time, the camera powers off .

The on/off process is all in software because the power of the camera is never cut (to preserve the time and date).

The other push-button is used as camera shutter to take still pictures if pressed and released quickly. If kept pressed for some amount of time, the camera enters into a continuous video recording mode, and it will be in that mode until the button is pressed again or battery is depleted. If the camera isn't recording video it will shut down automatically after some time, if there is no other action like taking still pictures.

Push-button quantity, time intervals to turn on/off, take pictures, start/stop video recording may vary depending on camera model and manufacturer, but generally speaking, tends to work in the way mentioned earlier.

To control and power the camera externally (in case of defective battery or to give more autonomy), wires must be soldered into the circuit board to the push-button contacts and battery PCB pads. Use enameled wire as thin as possible.



The SX1276 chip embedded in the module is able to operate from 100 MHz to 1050 MHz. However, this chip requires some external components between RF input/output pins and antenna. To minimize component burden and design complexity, the integrator put some passive and active components (capacitors, inductors, RF switch) to form a matched network, that will operate efficiently only in a small frequency range. There are different versions of modules manufactured to work in different ISM bands (a,b,c,d) to match local radio spectrum regulations, because not all countries allow to use the same ISM bands. In this example the 915Mhz version is used.

AUTOMATE BUTTON PRESS

To automate the picture taking process, a simple 555 or transistor based circuit is enough. However, if more complex things are required like taking pictures with varying time intervals, or the time interval is longer than the automatic shut down camera timer, or if a flash is needed in low light conditions, a "smart device" is needed.

To complete this task, whatever embedded programmable platform could be used, in this case a Digispark will be used, thanks to its small size, very low cost (clones available for around 1USD/EUR/GBP worldwide shipment included),plugs directly into USB port so no cable is required, and also with a bit of effort supported in the Arduino suite.

The cyclic steps executed by the microcontroller are the following:

- Press power button, keep pressing and then release (camera power on)
- Wait while the camera initializes
- Observe light conditions (using toy solar panel) to turn on "flash" if needed.
- Press shutter button, keep pressing and release quickly (to avoid entering into video recording mode)
- Turn off flash
- Wait until the picture is written into the micro SD card
- Press power button, keep pressed and then release (camera power off)
- Enable watchdog and put microcontroller in sleep mode.

The different waiting times, like button presses, initialization, turn on, turn off, should be found experimentally because they tend to vary according to the camera used. Usually one of the push-button works in "pull-up" mode and the other in "pull-down" mode. Verify this with a multimeter to know the proper way to connect each optocoupler.

The system was programmed in C using AVR GCC, Micronucleus drivers must be installed to program the Digispark module.

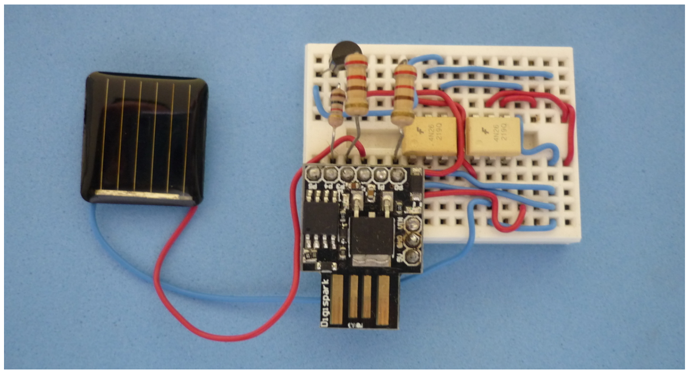


IMAGE PROCESSING

The camera stores the images on a removable Micro SD card. Every image is stamped with the programmed date time. This could be a potential problem if the images would be used to make a video, because there is no official way to remove the date time in every frame. There are some alternatives in the 808 forum, like to modify the firmware (only for very specific camera models), or to apply virtual dub filters, etc.

Una alternativa simple para eliminar la estampa de fecha/hora en muchas imagenes, es utilizar Imagemagick, la idea es recortar de la foto la porcion donde aparece lo que se quiere eliminar, ademas puede ser necesario redimensionar para que la imagen cuadre con el tamaño del video de la siguiente forma:

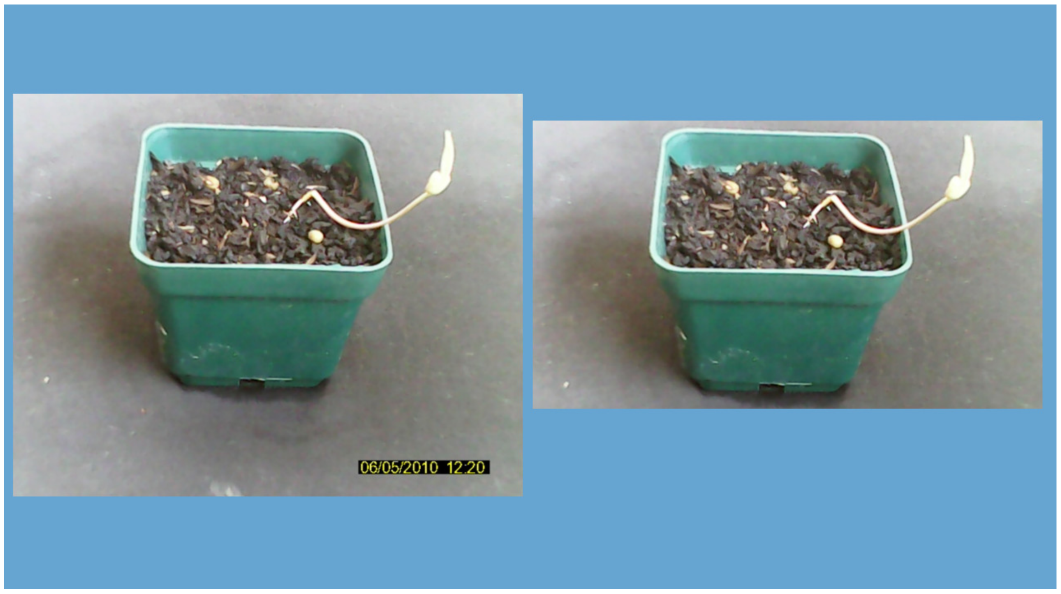
A much simpler approach to remove the timestamp from the image is to use imagemagick, cropping the area of the photo where the timestamp lies, also could be necessary to resize the image to match the aspect ratio of standard video formats in the following way:

```
mogrify -crop 1280x720+0+64 *.*
```

The bottom 64 pixel lines from the image were cropped, also the resulting image is 1280 x 720 pixels matching 720p video resolution.

Finally to generate a 30 fps video from multiple images, with their filenames in sequence starting from zero (i.e. EKEN0000.jpg, EKEN0001.jpg, EKEN0002.jpg...) avconv could be used:

```
avconv -r 30000/1001 -i EKEN%04d.jpg -r 30000/1001 video.mp4
```



CONCLUSIONS

- Image quality isn't very good, play with the object distance and different kinds of illumination to get better results.

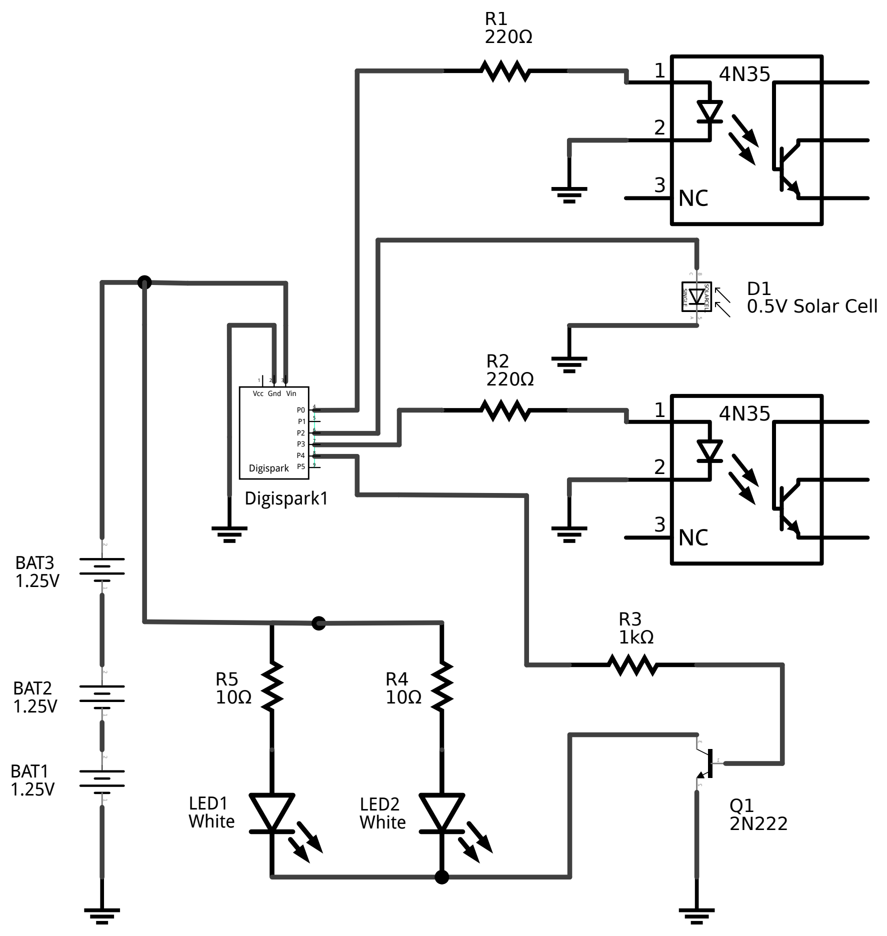
- Putting the microcontroller in sleep power down mode, and turning off the camera between takes, saves a lot of energy, so external batteries could be used for days making a relatively "portable" system.

- There are a lot of alternatives for time lapse photography, usually more expensive or power hungry. This simple exercise can be made in a couple of hours with components already available in the parts bin with, very low power consumption.

LINKS

Video showing camera modifications and time lapse sample: **http://youtu.be/MoLjWz_UUXY**
Digispark clone: **http://s.click.aliexpress.com/e/FQRfuvzvz**
Low cost mini camera: **http://s.click.aliexpress.com/e/J66eIiU7M**
Toy solar panel: **http://s.click.aliexpress.com/e/QfUfU3zvv**

SCHEMATIC



R1
220Ω
4N35
1
2
3
NC
D1
0.5V Solar Cell
R2
220Ω
4N35
1
2
3
NC
Vcc Gnd Vin
P0
P1
P2
P3
P4
P5
Digispark
Digispark1
BAT3
1.25V
BAT2
1.25V
BAT1
1.25V
R5
10Ω
R4
10Ω
R3
1kΩ
LED1
White
LED2
White
Q1
2N222

MAIN.C

main.c

```
// ********************************************
// Absolutelyautomation.com
// ********************************************
//  Includes
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
// ********************************************

uint8_t watchdog_count = 0;

// ********************************************
// ISRs
ISR(WDT_vect){
		++watchdog_count;
}
// ********************************************

// Digispark Internal LED

#define CAMERA_POWER		PB0
#define ONBOARD_LED		PB1
#define LIGHT_SENSOR			PB2
#define CAMERA_SHUTTER	PB3
#define EXT_FLASH_LIGHT	PB4



#define DELAY_SHUTTER			1000
#define DELAY_CAM_RDY		5000
#define DELAY_CAM_PWR		2500
#define DELAY_CAM_PIC			3000

int main(void) {

		// Init ONBOARD_LED pin as output
		DDRB |= (1 << ONBOARD_LED);
		// Init CAMERA_POWER pin as output
		DDRB |= (1 << CAMERA_POWER);
		// Init CAMERA_SHUTTER pin as output
		DDRB |= (1 << CAMERA_SHUTTER);
		// Init EXT_FLASH_LIGHT pin as output
		DDRB |= (1 << EXT_FLASH_LIGHT);

		// reset sequence!! (to know visually if reset)

		PORTB |= (1 << ONBOARD_LED);
		_delay_ms(100);
		PORTB ^= (1 << ONBOARD_LED);
		_delay_ms(500);
		PORTB |= (1 << ONBOARD_LED);
		_delay_ms(100);
		PORTB ^= (1 << ONBOARD_LED);
		_delay_ms(1000);


		for (;;) {

			cli(); //Disable interrupts
			MCUSR &= ~(1<<WDRF); // clear the watchdog reset
			// disable watchdog timer
			WDTCR |= (1 << WDCE ) ;
			//WDTCR &= ( 0 << WDE );
```

```
WDTCR &= ~( 1 << WDE );

if(watchdog_count > 2){
        watchdog_count = 0;

        // ************************************************************************
        //                      put your code here
        // ************************************************************************

        // Light up ONBOARD_LED
        PORTB |= (1 << ONBOARD_LED);

        _delay_ms(100);

        // pressing, waiting and releasing turn on camera button
        PORTB |= (1 << CAMERA_POWER);
        _delay_ms(DELAY_CAM_PWR);
        PORTB &= ~ (1 << CAMERA_POWER);

        // wait for the camera ready
        _delay_ms(DELAY_CAM_RDY);

        // Light conditions
        init_ADC();

        // start ADC measurement
        ADCSRA |= (1 << ADSC);
        // wait till conversion complete
        while (ADCSRA & (1 << ADSC) );

        if (ADCH > 25)
                {
                        // turn off external flash light
                        PORTB &= ~(1 << EXT_FLASH_LIGHT );
                } else {
                        // turn on external flash light
                        PORTB |= (1 << EXT_FLASH_LIGHT );
                }

        // pressing, waiting and releasing shutter camera button
        PORTB |= (1 << CAMERA_SHUTTER);
        _delay_ms(DELAY_SHUTTER);
        PORTB &= ~ (1 << CAMERA_SHUTTER);

        // wait for the picture taken
        _delay_ms(DELAY_CAM_PIC);

        // turn off external flash light
        PORTB &= ~(1 << EXT_FLASH_LIGHT );

        _delay_ms(100);

        // pressing, waiting and releasing turn off camera button
        PORTB |= (1 << CAMERA_POWER);
        _delay_ms(DELAY_CAM_PWR);
        PORTB &= ~(1 << CAMERA_POWER);

        _delay_ms(1000);

        // turn off ONBOARD_LED
        PORTB &= ~ (1 << ONBOARD_LED);

        _delay_ms(100);

        // ************************************************************************
}

TCCR0B = (1<<CS01); // Set Prescaler to 8

MCUCR = (1<<SM1)|(1<<SE); //Setup Power-down mode and enable sleep
```

```
                MCUSR &= ~(1<<WDRF); // clear the watchdog reset
                // set up watchdog timer
                WDTCR |= (1 << WDCE ) | ( 1 << WDE );
                WDTCR |= (1 << WDIE );
                WDTCR |= (1 << WDP3) | (1 << WDP0); // timer goes off every 8 seconds

                sei(); //Enable interrupts

                asm volatile ("sleep");

        }

  return 0;
}

void init_ADC()
{


        ADMUX =
                                (1 << ADLAR) |   // left shift result
                                (0 << REFS2) |     // Sets ref. voltage to 1.1V, bit 2
                                (1 << REFS1) |     // Sets ref. voltage to 1.1V, bit 1
                                (0 << REFS0) |     // Sets ref. voltage to 1.1V, bit 0

                                (0 << MUX3)  |     // use ADC1 for input (PB2), MUX bit 3
                                (0 << MUX2)  |     // use ADC1 for input (PB2), MUX bit 2
                                (0 << MUX1)  |     // use ADC1 for input (PB2), MUX bit 1
                                (1 << MUX0);       // use ADC1 for input (PB2), MUX bit 0

        ADCSRA =
                                (1 << ADEN)  |     // Enable ADC
                                (1 << ADPS2) |     // set prescaler to 128, bit 2
                                (1 << ADPS1) |     // set prescaler to 128, bit 1
                                (1 << ADPS0);      // set prescaler to 128, bit 0

}
```

MAKEFILE

MAKEFILE

```
# Absolutelyautomation.com

DEVICE     = attiny85           # See avr-help for all possible devices
CLOCK      = 16000000           # 16Mhz
OBJECTS    = main.o             # Add more objects for each .c file here

UPLOAD = micronucleus --run
COMPILE = avr-gcc -Wall -Os -DF_CPU=$(CLOCK) -mmcu=$(DEVICE)

# symbolic targets:
all:	main.hex

.c.o:
	$(COMPILE) -c $< -o $@

flash:	all
	$(UPLOAD) main.hex

clean:
	rm -f main.hex main.elf $(OBJECTS)

main.elf: $(OBJECTS)
	$(COMPILE) -o main.elf $(OBJECTS)

main.hex: main.elf
	rm -f main.hex
	avr-objcopy -j .text -j .data -O ihex main.elf main.hex
	avr-size --format=avr --mcu=$(DEVICE) main.elf
```