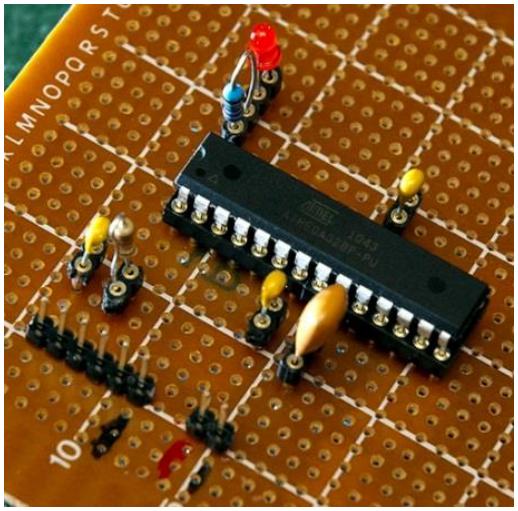
Aergia Arduino

O Arduino preguiçoso que aguenta 12 meses a pilhas



Apresentação de uma solução “power saving” utilizando o chip que está presente no Arduino.

O Arduino Uno, que inclui um regulador de tensão, USB interface e alguns LEDs consome demasiada energia para poder ser utilizado como uma solução portátil alimentado apenas a pilhas. O Arduino Uno apresenta um consumo elevadíssimo! Mesmo sem estar a fazer nada. O sketch mais simples consome cerca de 50mA se alimentado pelo “jack power in” (igual a alimentar pelo pino Vin) e mesmo alimentado pelo pino +5V consome cerca de 0.49mA (a alimentação no pino +5V faz bypass ao regulador de tensão do Arduino uno, deverá ser neste caso ser regulado antes ou corre o risco de danificar o chip).

Uma pilha alcalina de 9V com capacidade de aproximadamente 450/550 mAh durará assim apenas 9 horas (450mAh/50mA). Uma das soluções para o problema passa necessariamente por duas etapas.

Baptizei esta solução como “Aergia Arduino” em honra da deusa grega da preguiça e do ócio.

O Aergia Arduino é uma solução que eu adaptei e tentei documentar e verificar na melhor da minha competência. Não estará certamente isenta de incorrecções e foi simplificada em diversos pontos. Após verificar as muitas e diversas técnicas e soluções encontradas pela web (onde nem todas são racionais!), decidi publicar este tutorial. Pressupõe algum conhecimento de electrónica, microprocessadores e familiarização com a programação de um Arduino.

1ª Ponto: Barebone Arduino

O primeiro passo para resolução deste problema será ver-nos livres dos periféricos desnecessários. Uma opção é construir o denominado “Barebone Arduino” utilizando para isso o microcontrolador Atmega328P (datasheet) que é actualmente utilizado no Arduino Uno Rev3. O “Barebone Arduino” consume cerca de 6 a 15mA, o que representa já uma poupança significativa e é imprescindível a sua implementação, e será metade da caminhada.

Lista de material:

Breadbroad de 170 ou 400 pontos (0.84€ a 1.53€)
ou Prototype PCB (5 a 2.00€) e 24 pin DIP socket (10 a 0.55€)
ATmega328P com arduino Bootloader (1.00 € a 1.80€)
Cristal de 16mHz (20 oscilador 1.12€ ou um por 0.60€)
Resistência de 10kohm (0.05€)
Um condensador de 10uF (0.07€)
Dois condensadores de 22pF (0.20€)

É necessário saber a topologia dos portos no Atmega328P:

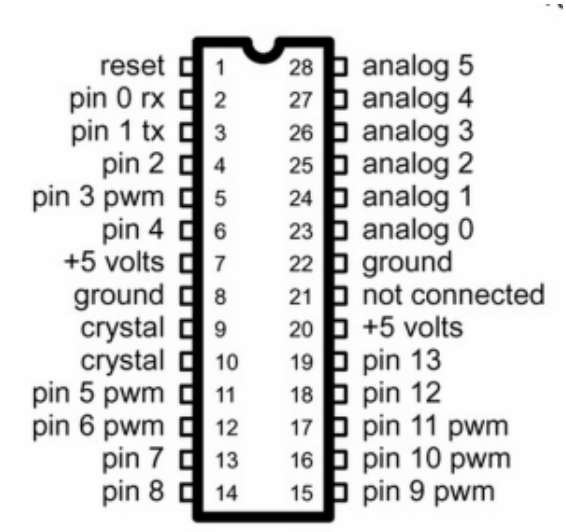


Ilustração 1 - Pin layout do microcontrolador da Atmega328P

E depois montar o seguinte circuito:

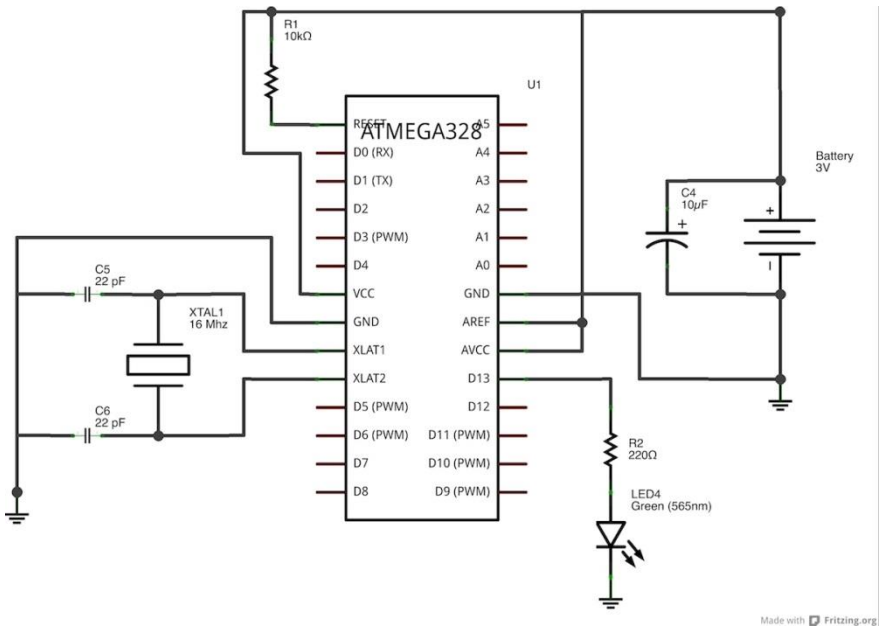
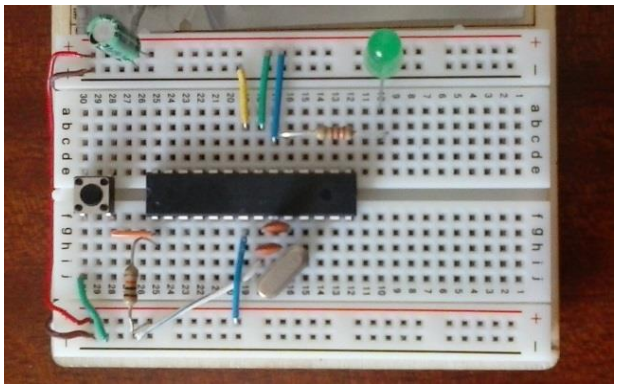


Ilustração 2 - Esquema do circuito "Barebone Arduino"

Sendo que pode igualmente colocar um switch on/off para forçar um Reset ao microcontrolador e o conhecido "LED13". Basta para isso colocar o botão de pressão que force a entrada no pino "reset" a 0V e um LED em série com uma resistência de 220/440 Ohm ao pino 19 do Atmega328P.

Dois exemplos do produto final:



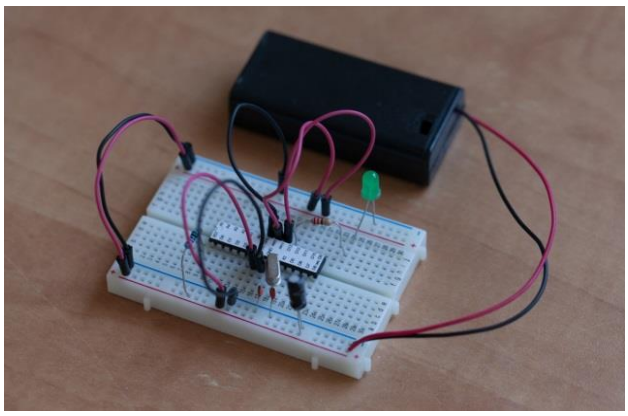


Ilustração 3 - Arduino Barebone com reset switch e LED13

Atenção! Esta montagem só poderá ser alimentada no máximo com 5V, sugiro a utilização de 3 pilhas alcalinas de 1.5V (sendo que existe outras soluções, poderá alimentar com >5V (e.g. pilha 9V) mas deve utilizar um regulador de tensão (e.g LM317 ou equivalente). Caso alimente com tensões superior a 5.5V o chip irá fritar.

2ª Ponto – SLEEP MODE e INT

O passo seguinte é fazer uso do modo "SLEEP" e utilização de "Interrupts" para acordar ("Wake-up") o controlador. A correcta utilização dessas funcionalidades faz baixar o consumo para 360uA!!! (sim, micro, uma poupança de XXX%!!).
De notar contundo, que a poupança efectuada com recurso a "SLEEP", sendo o mais modo mais "profundo", o "Power-Down Mode" só faz sentido se utilizarmos o "Barebone Arduino". O "Barebone" representa uma poupança na ordem dos 35mA enquanto as soluções de SLEEP contribuem com poupanças de cerca 15mA.

Pode agora saltar para a 3ª Etapa para um resumo pragmático e respectiva implementação desta solução. Irá contudo falhar a leitura interessante e enfadonha de gráficos e tabelas.

Para mais informação sugiro a leitura do capitulo "Power Management and Sleep Modes – 9 " do datasheet para mais informação dos diferentes modos de "SLEEP" e a sua correspondência com o desabilitar de funcionalidades,

assim como os “Wake-up” possíveis para cada modo (pagina 39-43). A leitura dos gráficos do capitulo “Typical Characteristics -29” (pag 399 a 406) que exploram a poupança possível com recurso a relógio de menor frequência, alimentação <5V, ect.. é também valiosa caso este assunto lhe cause curiosidade.

Neste tutorial faço apenas especial referência aos gráficos 29-140 e 29-149 e tabela 9-1.

Table 9-1. 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_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other I/O	Software BOD Disable
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		
Power-down								X(3)	X				X		X
Power-save					X		X(2)	X(3)	X	X			X		X
Standby(1)						X		X(3)	X				X		X
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

Figure 29-140. ATmega328P: Active Supply Current vs. Frequency (1-20 MHz)

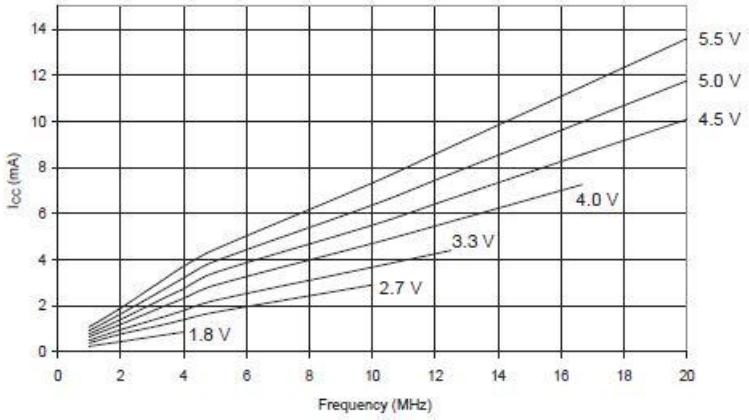


Figure 29-149. ATmega328P: Power-Down Supply Current vs. V_{CC} (Watchdog Timer Disabled)

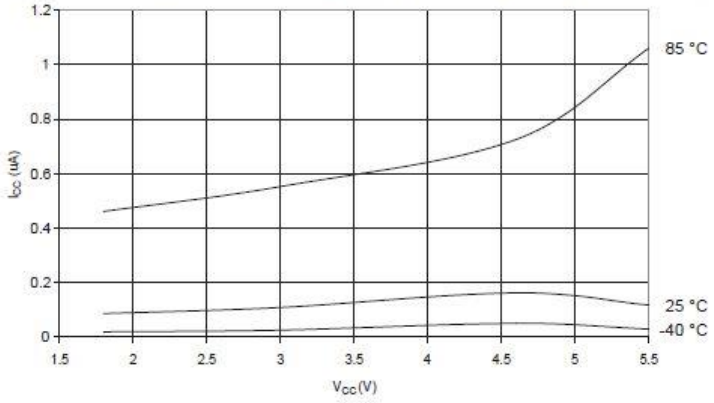
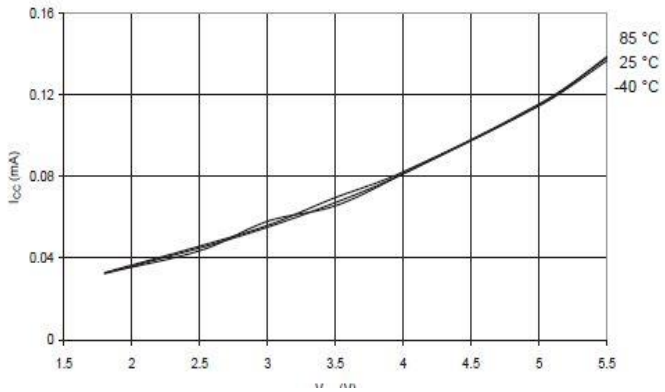


Figure 29-141. ATmega328P: Active Supply Current vs. V_{CC} (Internal RC Oscillator, 128 kHz)



De notar que é possível o recurso ao oscilador interno do Atmega328P (128KHz) para baixar o consumo ainda mais, contundo é necessário ter em conta que isso faz aumentar o tempo de resposta do controlador. Uma solução razoável e com poupanças adicionais significativas que não irá ser explorada neste tutorial serão a alimentar a 3.3V com oscilador de 8MHz (3mA de "active supply current").

É sempre necessário escolher o modo de SLEEP adequado, tendo presente que isso tem como vantagem baixar o consumo mas restringe a escolha de "Wake-up" (interrupções) possíveis de serem utilizadas.

Existem dois tipos de interrupções externas no atmega328P, as conhecidas INT0 e INT1 e o "pin change interrupts".

O INT0 e INT1 correspondem aos pinos 2 e 3 respectivamente e é possível configurar a interrupção para responder apenas a um dos seguintes "triggers": uma mudança de nível, um flanco ascendente, um flanco descendente ou a um "low-level input" (link). Uma função possível de ser utilizada para configuração das INT é a "attachInterrupt()" que permite habilitar a interrupção, escolher a função a ser chamada na ISR, o pino e o modo de "trigger". Sugiro a seguinte leitura (link) para mais informação (ao ver o código em exemplo mais à frente).

Para acordar o microprocessador do modo "power-down" só é possível com uma rotina desencadeada por um "low-level input".

O "pin change interrupt" responde a qualquer alteração do nível logico do sinal nos portos, sendo importante referir que no atmega328P existem 3 portos (PORT A/B/C) com 8 pinos cada, sendo que a rotina de interrupção (ISR) indicará em que banco foi registado a alteração mas não descrimina qual o pino. Com o recurso a bibliotecas (<PinChangeInt.h> e <PinChangeIntConfig.h>) é possível estender algumas características de INT0 e INT1 ao restantes pinos do controlador. Informação sobre biblioteca "PinChangeInt" pode ser consultada na página do Arduino (link) com um exemplo esclarecedor (link).

Para projectos menos complexos e com a finalidade de "power saving" recomendo sempre a escolha de INT 0/1 e a escolha de SLEEP "power-down" e/ou "idle".

De notar que o atmega328P também permite interrupções internas, fazendo uso de timers, comparadores, usart, adc, ect.. O uso de timers e "watchdog" (WDT) permite, por exemplo, colocar o microcontrolador em sleep e acordar após um certo tempo, tornando desnecessário a utilização da função delay() e permitindo assim poupanças de energia (próximo tutorial).

3ª Ponto: Implementar o código

A nossa escolha recai na utilização do modo sleep “Power-Down” para uma maior poupança, e pretendemos acordar o atmega328P com uma mudança de sinal num porto 2 (INT0). De notar que existem varias formas de implementar e estruturar o código, tanto na chamada do sleep como sobretudo na gestão do código a efectuar na Interrupção. Aqui a intenção é apenas acordar o microcontrolador. Esta é uma solução que considero versátil e de fácil aprendizagem.

/* Implementação de codigo standart para sleep e ISR – Fabrício Agosto de 2014 */
Link para download do ficheiro ino

/* Implementação de codigo standart para sleep e ISR – Fabrício Agosto de 2014 */

```
/* Incluir a biblioteca de sleep*/
#include <avr/sleep.h>

/* Pino usado no Wake-up */
int wakePin= 2;

/* Função ISR que é chamada depois do Wake-up */
void wakeUpNow()
{
/* Código executado depois do wake-up e antes de voltar a função loop()
 * de notar que timers e serial.print (e muito mais) não funciona em funções ISR.
 * Esta função é necessária mesmo que não seja feito nada da função wakeUpNow.
 * Pode ser, por exemplo, adequado a marcação de flags
 * Variáveis utilizadas aqui deverão ser globais e declaradas como volatile (importante!)
 *
 */
}

void setup()
{
/* Configuração do porto para entrada com pull-up interna habilitada */
pinMode (wakePin, INPUT);
digitalWrite (wakePin, HIGH);

/* Iniciar a comunicação “série” para permitir um debug mais fácil do código
 * Deverá ser colocado em comentário na versão final
 */
  Serial.begin(9600);

/* Habilitar “interrupt” e escolha da interrupção a utilizar:
 * attachInterrupt (A,B,C)
 * [A] pode ser 0 (INT0 que corresponde ao pino 2) ou 1 (INT1 que corresponde ao pino 3).
 * [B] é o nome da função que queremos utilizar na interrupção.
 * [C] é tipo de trigger da interrupção de pino que pode ser:
 *  LOW - quando o sinal é levado a 0V (vref)
 *  CHANGE – mudança de flanco no pino
 *  RISING – flanco ascendente do sinal no pino
 *  FALLING – flanco descendente do sinal no pino
 * nota: só no sleep mode idle é possível a escolha de [C], nos restantes apenas LOW funciona!
 *
 */
```

```
 /* Uso de INT0 (pino 2) corre função wakeUpNow quando tivermos 0V na entrada
 * o attachInterruput aqui pode ser retirado caso não queiramos INT logo no inicio
 */
  attachInterrupt(0, wakeUpNow, LOW);
}


/* Funcão para colocar o Atmega328P em modo SLEEP */
void sleepNow()
{
   /* Escolha do modo de sleep:
    *
    *
    * Existe muitas escolhas sendo que estas são as de maior interesse
    *   SLEEP_MODE_IDLE         // - poupança, + versátil no wakeUpNow, permite ADC, SPI, USART, TIMER/COUNTER
    *   SLEEP_MODE_ADC              // apenas desliga o clk i/o, até o ADC poderá acordar o atmega
    *   SLEEP_MODE_PWR_SAVE    //  igual a power_down mas permite TIMER/COUNTER 2
    *   SLEEP_MODE_PWR_DOWN // + poupança, apenas op. sem clk são permitidas (INT0/1, SERIAL, ect)
    *
    */

   /* Escolha do modo Sleep */
   set_sleep_mode(SLEEP_MODE_PWR_DOWN);

   /* Permissão do modo sleep*/
   sleep_enable();

   /*
    * Habilitar a interrupção dentro da função sleepNow permitindo
    * Assim um maior controlo da INT no resto do código.
    * Neste exemplo só é pretendido que a INT ocorra caso o Atmega esteja em Sleep
    *
    */
   attachInterrupt(0,wakeUpNow, LOW);

   /* Aqui o Atmega é efectivamente colocado a dormir! */
   sleep_mode();     // nota que o atmega328P pode demorar 65ms a acordar depois de um INT

   /* O CODIGO RETOMA DAQUI DEPOIS DE ACORDAR */


   /* Primeira coisa a fazer é desabilitar a opção de sleep*/
   sleep_disable();

   /* Segunda coisa a fazer é desabilitar a interrupção para que não volte a acontecer
    * Não queremos que o wakeUpNow ocorra novamente sem a nossa instrução explícita
    * Permite assim proteger que uma INT não aconteça enquanto o código de trabalho esteja a correr
    *
   */
   detachInterrupt(0);
}


void loop()
{
  /* Codigo apenas para debug que deve ser comentado na versao final */
  Serial.println("Inicio do loop\n");
  delay(1000);
```

```
 /* Codigo apenas para debug que deve ser comentado na versao final*/
 Serial.println("Timer: Entering Sleep mode");
 delay(100);

 /* Função para colocar o atmega em SLEEP é aqui chamada*/
 sleepNow();
}
```

Esta configuração permite facilmente ter um microcontrolador a durar 6 meses com recurso a apenas 3 pilhas alcalinas de 1,5V. A escolha de onde colocar o código de trabalho deverá ser decidida causticamente consoante o projecto que se quer implementar.

Soluções semelhantes são possíveis de implementar para permitir acordar o controlador por comunicação série (podendo até o exemplo supra ser utilizado e colocar uma resistência entre o pino RX e o pino 2). Contundo, projectos onde o wake up é efectuado por um timer poderão/deverão ser implementados de outra forma (fazendo uso por exemplo do watchdog interrupt. Existem igualmente bibliotecas especificas para lidar com INT causadas por comunicação I2C, ect..

Maiores poupanças são ainda possíveis alimentando o chip com tensões inferiores e utilização de osciladores de menor frequência ou recurso a pre-scales. Com soluções ainda mais profundas, existe literatura a fazer referencia a soluções com valores na ordem dos 0.35uA no modo sleep durante 99% do tempo e com 1% do tempo no modo activo a consumir 5mA. O que pode significa utilizações superiores a 33 meses com pilhas AAA.

É de notar, que uma pilha descarrega mesmo que não esteja a ser utilizada e esse valor pode até ser superior ao que é consumido pelo projecto em causa.

4ª Ponto: Outros cuidados

Caso seja necessário um regulador de tensão, sugiro a escolha de um com baixa corrente de repouso (“quiescent corrent”) como por exemplo o LM7805 (que mesmo assim consome 5.2mA). Uma outra montagem bastante conhecida é fazendo uso do LM317 e mesmo este, a corrente de repouso é cerca de 1 a 3.5 mA. (ver tutorial sobre reguladores de tensão).

LED irá consumir pelo menos 10mA, evitar! Mesmo que use uma solução de acender durante 10ms por segundo para fazer uso da “persistência da visão” irá consumir uma corrente considerável.

Use PWM quando possível, entradas analógicos apenas quando imprescindível, utilizar MOSFET para on/off da alimentação dos periféricos é uma óptima e apetecível ideia (essencial para evitar correntes de repouso). Quando estamos na escala dos micros Amperes, a simples configuração dos pinos não utilizados permite poupanças. Uma configuração com todos os pinos Low consume 0.35uA enquanto o estado High 1.86uA. Recomendo sempre a utilização em Output (alta impedância) para segurança do chip.

5ª Ponto: Literatura e recursos

Datasheet do Atmega328P
Atmega48/88/168/328

Ebook – Arduino Internals by Dale Wheat
(sem link, e não deverão comprar este livro via piratebay.org ou pedir que eu empreste)

O fórum oficial do Arduino e o respectivo playground
forum.arduino.cc/
playground.arduino.cc/

Arduino Stand Alone (post oficial)
arduino.cc/en/Main/Standalone

Imprescindível para qualquer aprendiz de EE
electronics.stackexchange.com/

Index de wiki, forum, e blogs
freeduino.org/

Universidade de South Florida tem diversos vídeos online sobre o Arduino (aprendi muito com estes vídeos!)
Makecourse.com

Site que utilizo para comprar material:
Ebay.com (porque encontramos o bom (e o horrível) barato e com free shipping)
Sonigate.com (é relativamente perto e gostamos de apoiar o comercio local)
Banggood.com (Free shipping mas tem um armazém no Reino Unido, possui reviews esclarecidas de utilizadores)
Dx.com (mais dispendioso mas com reviews de utilizadores)

Fabricio Ferreira Antunes – Setembro de 2014
(http://aergiaee.wordpress.com/

FIM