Aplikácia viacnásobných minútiek, bežiace na taskmanažéri cocoOS [ATmega8]

Známy ma poprosil, či by som mu nedal dohromady nejaký "countdown timer" - minútky, ale s možnosťou viacerých prednastavených časov.
Aplikácia jednoduchá, tak prečo ju neskomplikovať.
Aj tak som chcel vyskúšať OS (taskmanažér) CocoOS, tak mi to prišlo vhod.

Účel zariadenia
Požadované boli klasické kuchyňské minútky (odpočítavanie z prednastaveného času), ale doplnené o viac časov (konkrétne 2).
To znamená, že po odpočítaní prvého času, zaznie akustický signál, a automaticky sa po ňom spustí odpočet ďalšieho prednastaveného času, opäť signál, nasleduje odpočet prvého zadaného času, a tak ďalej.
Malo sa jednať o jednorázové zariadenie (alebo skôr vzorové), takže výroba DPS odpadla. Výsledná cena mala byť minimálna.

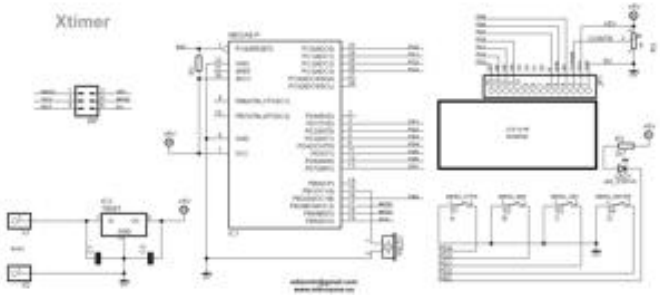
Hardware
Prvý problém bol vyriešiť celkový vzhľad zariadenia tak, aby to nestálo veľa peňazí. Tento problém riešim zakaždým keď chcem niekomu ponúknuť lacný domáci "zlepenec". Vyriešiť elektroniku a program nie je problém, ale ten vzhľad....
No, tak som sa poobzeral v zásobách nepoužitých vecí, a našiel som.
Našiel som nevyužitý (a možno aj pokazený) digitálny telefón Panasonic KX-T7433, ktorý ma zaujal pre jeho zaujímavé riešenie.

Nie že by som ho chcel použiť ako skrinku pre tieto minútky, ale iba jednu jeho časť - a to displej.
Displej je v samostatnej krabičke, so 4 tlačítkami a jednou signalizačnou LED diódou. Táto krabička/modul je cez plastový pánt prichytený k telefónu tak, aby sa dal tento displej nakláňať v určitom rozsahu.
Okamžite som modul oddelil a delaboroval.

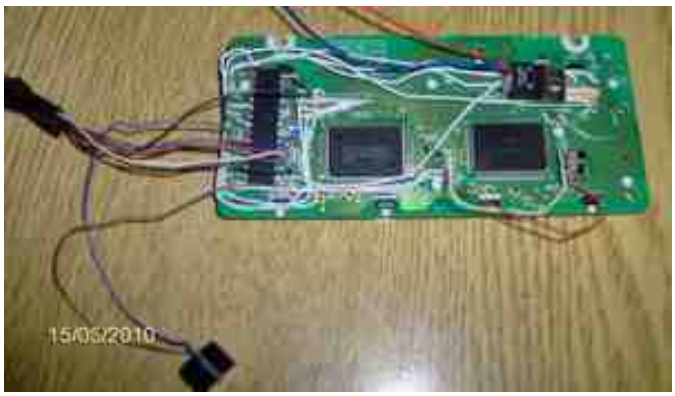
Vnútri bola úplná lahôdka:
3 riadkový displej (vlastne 4-riadkový) s radičom HD44780, štyri tlačítka a signalizačná LED dióda a to všetko samostatne prepojené na konektor ZIF. Okrem toho dosť miesta na doplnenie riadiacim procesorom....

Bolo rozhodnuté.

V šuplíku som rozmerovo vhodný procesor našiel len ATmega8 v DIL prevedení, tak som ho upravil na SMD verziu (vyrovnal vývody).
Doplnil som prepotrebný 5V stabilizátor (napájanie som plánoval riešiť externým AC/DC adaptérom), piezo menič (pípak), trimer na riadenie jasu LCD a vytiahol som ISP konektivitu (mám "len" Presto od ASIXu).



K tomu za hrsť drátových prepojok a elektrickú časť hardwéru a obal som mal elegantne poriešenú.



Software
ATmega8 má 8 kilovú FLASH pamäť, čo na minútky bohate postačuje, tak som ju musel nejako vyplniť.
Program sa vo finále skladá s troch častí:
task manažér - cocoOS
riadenie LCD - LCD library for HD44870 based LCD's
samotná aplikácia - do ktorej som ešte dorobil:
automatické prednastavenie časovania podľa poistiek (F_CPU)
filtrácia tlačidiel s funkciou zrýchľovania pri držaní
nastavenie cez MENU
generovanie zadanej frekvencie - pre pípak, ktorý môže zapískať melódiu

Kultúra programu nie je nijak čítanková, predsa len obsahuje kód od troch autorov, a ešte aj ten môj výmysel avr_compat (samozrejme nedotestovaný), a celé je to umocnené nedostatkom času a nočným kódovaním....
Teraz sa pozriem na jednotlivé časti.

cocoOS
je jednoduchý portovateľný task manažér, od autora Petra Eckstranda.
Je šírený zdarma pod GNU General Public licenciou.

Po stiahnutí a vytvorení test projektu v prostredí AVR Studia, mi kompilér bez optimalizácie zahlásil výslednú veľkosť 3266 bytov, len čisto tento task manažér bez akýchkoľvek inicializácii.
Výsledky s optimalizáciou:

- O1 - veľkosť 1840 bytov

- O2 - veľkosť 1830 bytov

- O3 - veľkosť 2228 bytov

- Os - veľkosť 1728 bytov

Z RAM pamäte si ukrojil 26 bytov.
Použil som verziu cocoOS 1.2.0 (2010-04-12).

Príručka je dostačujúca, a ďalšie študovanie skrátil aj vzorový príklad.
V procesore ATmega8 zaberie podstatnú časť pamäte (bez optimalizácie skoro 40%), ale zvyšný kód nemal byť nejak zložitý, tak je tam.

Chod task manažéra, je riadený prerušením od časovača (v mojom prípade Timer 0 - overflow), ktorý ho krokuje volaním funkcie *os_tick()*.
Timer som nastavil na 1ms. Tento krok som zvolil hlavne preto, že funkciu (*clock_init*) ktorá mi prepočíta nastavenie timera som mal už hotovú, ale jej najmenší vstup bola práve 1 ms.

Takže pri prerušení od časovača T0, je riadenie predané systémovej funkcii *os_task_tick*, ktorá pohľadá vytvorený čajkajúci task s najvyššou priopritu, a predá mu riadenie.
Pokiaľ sa tento task do 1ms neukončí, bude prerušený a zavolaný bude opäť ďalší task, atď....

CocoOS umožňuje aj prácu so semafórmi, a udalosťami (events), ale k tomu som sa už nestihol nedostať.

Inicializácia task manažéra je jednoduchá.

```
1. os_init();                                   // inicializacia OS
2. os_task_create( nazov_tasku, 1 );            // vytvorim task s prioritou 1 -  najvyšou
3. os_task_create( iny_task, 2 );               // taktiež vytvorím iný task  s prioritou 2
4. clock_init(1);                               // nastavím a spustím 1ms prerušenie
5. os_start();                                  // a predám riadenie manažérovi.
```

Samotný task potom vypadá takto:

```
1. static int nazov_tasku(void)
2. {
3.  OS_BEGIN;
4.    .
5.    .  // task code
6.    .
7.  OS_WAIT_TICKS( 200 );  // tento task uspím na 200ms. Medzitým sa môže vykonať iný  task.
8.  OS_END;
9.  return 0;
```

```
10.
    }
```

LCD library for HD44870 based LCD's
Táto céčkovská knižnica (písaná priamo pre AVR procesory a GNU C kompilér) obsahuje základné rutiny pre ovládanie a riadenie LCD modulov s integrovaným radičom HD44780 a KS0073.
Napísal ju Peter Fleury, a vychádzal zo zdroja: Volker Roth - LCD library (originál som na internete nenašiel), pričom ju doplnil o niektoré funkcie, o 4-bitové pripojenie a celkovo ju optimalizoval.
Podporuje komunikáciu s LCD modulom buď v IO móde - ľubovolné priradenie pinov MCU k LCD modulu, alebo v pamäťovom móde - LCD modul je pripojený na adresno-dátovú zbernicu a pristupuje s k nemu ako k externej pamäti.

Práve túto knižnicu som vybral spomedzi dostupných na internete, pretože sa mi zdala maximálne konfigurovateľná a ponúkajúca mnohé (síce momentálne nevyužité) funkcie.
Jej rozchodenie nebol problém. Po stiahnutí a vytvorení test projektu s minimom potrebných funkcií:

```
1.
    lcd_init(LCD_DISP_ON);

2.
    lcd_clrscr();

3.
    lcd_puts("LCD Test Line 1\n");
```

mi výsledné veľkosti vyšli takto:

- O0 - veľkosť 1110 bytov
- O1 - veľkosť 660 bytov
- O2 - veľkosť 510 bytov
- O3 - veľkosť 2214 bytov (naozaj 2214!)
- Os - veľkosť 490 bytov

Z RAM pamäte si rutiny vzali 18 bytov.
Pracoval som s verziou knižnice 1.14.2.1 (2006-01-29)
Autor ponúka aj online manuál.

Aplikácia - rutiny.

Automatické prednastavenie časovania podľa poistiek (F_CPU)
Túto rutinu (*Get_CoreFreq*) som napísal hlavne z dôvodu častého výskytu zlého nastavenia poistiek pri programovaní. Osvedčilo sa mi to hlavne pri testovacích programoch, kedy zariadenie "zaručene" ožije aj pri inom nastavení poistiek CKSEL0 - CKSEL3 ako má byť. Resp. Minimálne dokáže o tejto skutočnosti informovať obsluhu, či už po sériovej linke alebo inak (časovo závislým

kanálom).
V tejto aplikácii sa pri štarte prečíta stav poistiek, na záíklade čoho sa určí F_CPU – hodnota je uložená v premennej Real_Core_FREQ. Táto informácia sa aj vypisuje pri štarte na displej spolu s verziou SW, názvom aplikácie a samozrejme aj s textom „www.mikrozone.eu“ .

Ďalšia výhoda premennej *Real_Core_FREQ* (tyto) je v programovaní 8051 (čiže časovačov), kde je vytvorená pod prerobenie sa nejak nikdy nenašiel čas, ale je funkčná.

Filtrácia tlačidiel s funkciou zrýchlenia pri držaní
Toto je tiež jedna z funkcií, ktorú som mal pri ňom v mojom prípade. Nie je nič horšieho ako keď je potrebné tlačítko niekoľkokrát stlačiť, kým sa to „chytí“. Tento vyšší komfort ovládania zákazníkovi prístroja nie je nič horšieho ako keď je potrebné tlačítko niekoľkokrát stlačiť, kým sa to „chytí“. Výsledok je ten, že pri jednom stlačení sa vráti zvyšok kódu stavu aktívneho tlačítka, alebo nedostatočne stlačené tlačítko. Pre vyriešenie tohoto problému je nutné, aby sa pri jednom stlačení tlačítka vykonal iba jeden krát. Ak je vhodná programovo vyriešiť zmenu stavu aktívneho tlačítka s tým, že sa spustí funkcia repeat (samozrejme stále sa testuje či je fyzicky tlačítko aktivované), teda rutina simuluje pravidelné stláčanie tlačítka.
Vo vizuálnej podobe sa to javí tak, že po stlačení je k hodnote pripočítaná jednotka (napríklad), potom sa chvíľku nič nedeje, a potom sa začne pripočítavať ďalšia jednotka, ďalšia.... atď. Po malom čase sa pripočítavaná hodnota zvýši na dva, neskôr na tri a atď.... (tzv. Additívna hodnota, dá sa využiť napríklad aj na zrýchlenie pripočítavania).

Nastavenie cez MENU
No ono to celkom menu nie je, teda lepšie povedané, už som spravil aj prepracovanejšie MENU, hlavne bolo prehľadnejšie a pridávanie ďalších položiek bolo jednoduché. V tejto aplikácii som MENU vytvoril iba z 2 dôvodov:

- čas od ktorého sa bude odpočítavať treba nejako nastaviť

- predpokladám že o chvíľu budem dorábať nejaké ďalšie nastavovanie do aplikácie, tak nech je to „pod jednou strechou“

Do tejto funckie sa program dostáva vždy pri zmene odpočítavaného času (teda každú sekundu) v režime odpočítavania, a vždy po ukončení podprogramu na filtráciu tlačítok.
Ďalej sa vo funkcii *void App_Menu(void)* spracovávanie rozdeľuje podľa prevádzkového stavu aplikácie, rteda buď sa odpočítava čas, alebo sa nastavujú parametre v MENU (funkcia tlačidiel je iná a aj text na displeji je iný).
V režime MENU sa nastavuje čas Tx pre odpočet. Čas sa vyberá stlačením tlačítka „D“, minúty zvoleného času sa iba inkrementujú tlačítkom „B“, sekundy tlačítkom „C“. Súčasným stlačením tlačidiel „B“ a „C“ sa zvolený čas nuluje, a stlačením tlačidla „A“ sa všetky časy uložia do EEPROM pamäte.
V režime odpočtu času (a po spustení) sú aktívne iba dve tlačidlá - „A“ spúšta a zastavuje odpočet (indikovaná aj blikaním LED diódy, a „D“ - zabezpečuje prechod do režimu MENU.)

Generovanie zadanej frekvencie - pre pípak, ktorý môže zapískať melódiu
Toto je posledná vec, ktorou som zapratával miesto vo FLASH.
Nevedel som sa rozhodnúť, aé pípanie sa má ozývať pri dosiahnuití odpočtu času na nulu, a tak som vytvoril jednoduché funkcie

- *int Beep(unsigned int Freq)* a

- *void Pause(unsigned int ms)*

Beep vygeneruje na pine SNDOUT (app_conf.h) signál o danej frekvencii, ktorý je „zhudobnený“ pripojený piezomeničom. Pri Beep(0) sa prestane generovať akýkoľvek signál – generuje sa ticho.
Pause pozastaví (úplne!) chod programu na daný čas (v milisekundách).
Pomocou volania týchto funkcií (parametre treba vedieť zvoliť, však muzikanti !) je možné generovať jednoduchú melódiu alebo zvuk podobný digitálnym hodinkám.....
Generovanie je pomocou 16-bitového časovača T1, ktorý je nastavený (*void T1_init(uint32_t tick_ms)*) v režime CTC (mode 4). Zmena frekvencie je realizovaná zmenou TCNT a aj jeho reload premennej TCNT1_ReloadVal.
Výstup signálu na pin je povolený/zakázaný nastavením bitu COM1A0 v registri TCCR1A.

Zozipovaný zdrojový kód je tu: xtimer-src

Výsledok prác

Zdrojový kód v C je v prílohe. Výsledná veľkosť je 7842 bytov (ostalo miesto na melódiu...) pri optimalizácii Os (inak tam nevojde), RAM nároky sú 280 bytov a v eeprom sú obsadene 4 bajty. Veľkosť obsadenia pamäte je závislé na počte časov (u mňa 2 -T0 a T1). Tieto sa dajú zmeniť cez konštantu MAX_STOPWATCH (app_conf.h).
Súborov .c a .h je trochu viac než by sa na stopky dalo čakať, ale organizácia je jednoduchá. Všetky exportované c funkcie a premenné sú zapísané v hlavičkových súboroch, ktoré sú includované v jednom – global.h
Tento je potom includovaný vo všetkých zdrojových .c súboroch.
Celkové nastavenie aj hardwéru je v app_conf.h (aspoň dúfam), LCD piny a nastavenie sú však v lcd.h súbore.
Takéto rozloženie používam ja, možno nie je dobré, ale nechám sa poučiť.

Epilóg

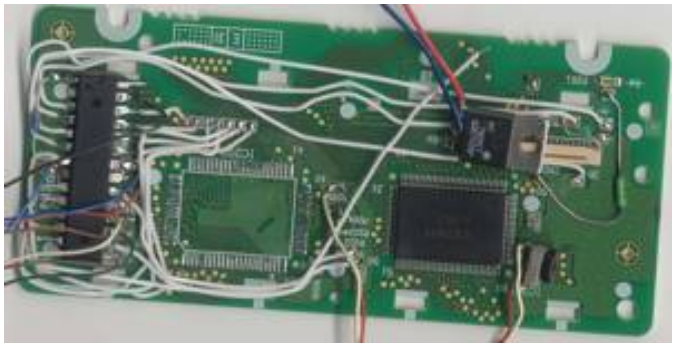
Murphyho zákone poznáte. Konkrétne čo sa môže pokaziť sa aj pokazí.
Doplnené o zákon posledného šróbu - po zatiahnutí posledného šróbu na obale, zariadenie skape.

Tak to sa mi presne stalo.

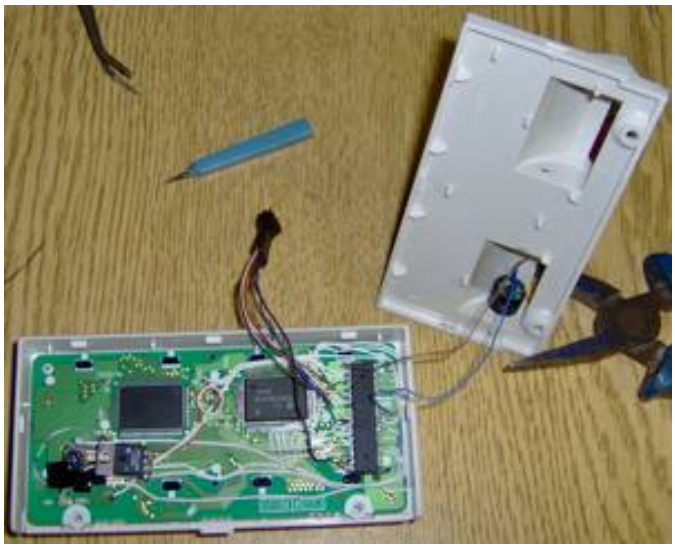
Minútky prestali pracovať. Prejavovalo sa to tak, že program vôbec nebežal, ale programátor s MCU veselo komunikoval.
Po hľadaní príčiny, som dospel k záveru, že program nedokáže inicializovať displej, pretože HD44780 má stále nastavený BUSY flag.
Po štúdiu jeho nijak detailného datasheetu (vlastne iba blok. zapojenia) a nájdení ďalšieho problému (popísané tu), som dospel k záveru, že radič HD44780 je vadný.

No a teraz mi moja geniálna myšlienka - využiť LCD modul ako nosnú dosku pre MCU, dala po hube.

LCD modul nevymeniteľný, takže trebalo vymeniť len radič.
Hurá na to (no hurá, bolo okolo polnoci).
Lámacím nožíkom som odrezal vývody zlého radiča, následne opatrne mikropájkou odspájkoval zvyšky vývodov, a očistil liehom.



V mojom šrote starých DPS som našiel jednu s osadeným radičom HD44780 - nik ale nevedel či bude dobrý. No iný nebol....
Teplovzdušná pištoľ sa postarala o bezpečné odcínovanie (bezpečné pre radič, zvyšok dosku to neprežil práve najlepšie), a moja ER50 sa postarala o opätovné naspájkovanie na môj "geniálny" LCD modul.



Po spustený nastala radosť.
Chvíľku.
Tak od 02:00 do 02:02.
3/4 displeja svietili na čierno, a v jednej štvrtine bolo vidno korektný text.

Znova rozoberanie, už aj nejaké drátové prepojky odišli a hľadanie chyby.
Logicky (ako to na nočnú hodinu šlo) som si domyslel, že druhý čip na displeji zrejme nefunguje. No meniť nebolo za čo.
Heuréka, po prepískaní komunikačných spojov medzi druhým čipom a HD44780, sa našla malá cínová guľôčka, posadená medzi jeho vývodmi.

Radosť.

Všetko funguje, hurá na kute.....