Universitatea Politehnica din Bucuresti

Facultatea de Automatica si Calculatoare

Proiectarea cu microprocesoare

Snake

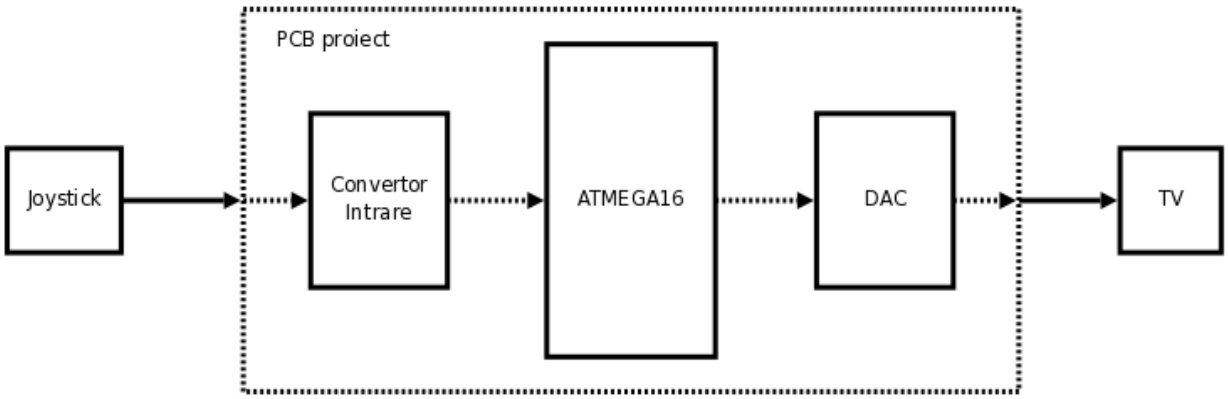
Andrei Homescu, 344 CA

1. Introducere

Snake este un joc foarte popular instalat pe foarte multe modele de telefoane produse de Nokia. Am ales sa implementez o versiune a jocului pe un microcontroller Atmega16, memoria si viteza acestuia fiind potrivite unei astfel de aplicatii. Comenzile sunt date de jucator printr-un joystick conectat la placa, iar pentru afisare se foloseste un televizor. Jocul se conecteaza la intrarea *Composite* a televizorului.

2. Descriere generala

Pentru interactiunea utilizatorului cu jocul am avut de ales intre mai multe variante: butoane, joystick sau accelerometre pentru comenzi si televizor, calculator sau display LCD pentru afisare. Am ales joystick-ul si afisarea pe televizor deoarece nu am reusit sa gasesc accelerometre si display LCD cu performante si pret care sa se incadreze in bugetul prevazut. De asemenea, display-urile LCD folosesc protocoale complexe de afisare; in schimb, comunicarea intre ATMEGA16 si TV foloseste doar 2 biti din iesirea controllerului.



Semnalul TV este generat in intregime in software si convertit dintr-o valoare pe 2 biti in semnal analogic, cu ajutorul unui DAC. Standardul folosit in tara noastra este PAL. Generarea software a semnalului a fost cea mai delicata parte a proiectului, necesitand multe incercari si ajustari pentru a genera semnalele potrivite si a centra imaginea pe ecran.

Programul este format dintr-o bucla principala, care la un interval fix avanseaza starea jocului. Semnalele software sunt generate de catre o rutina de tratare a intreruperilor de *timer*. Aceasta rutina trebuie sa poata intrerupe orice alta functie din program, dar nu poate fi intrerupta de nici o alta functie, deoarece semnalul video trebuie sa aiba un format precis; fiecare instructiune din functie trebuie executata la un moment precis, altfel apar distorsiuni mari in imaginea afisata. Functia de afisare consuma o cantitate destul de mare de timp de procesor; programul afiseaza de 25 de ori pe secunda cate 625 de linii. Dintre acestea, 121 sunt folosite pentru sincronizare sau nu sunt afisate pe ecran, deci timpul de executie corespunzator poate fi folosit in alte scopuri. Functia de afisare ruleaza doar pentru cateva cicluri de ceas in aceste cazuri. Astfel, aproximativ 80% din ciclurile de ceas ale microcontrollerului sunt petrecute in functia de generare a semnalului PAL.

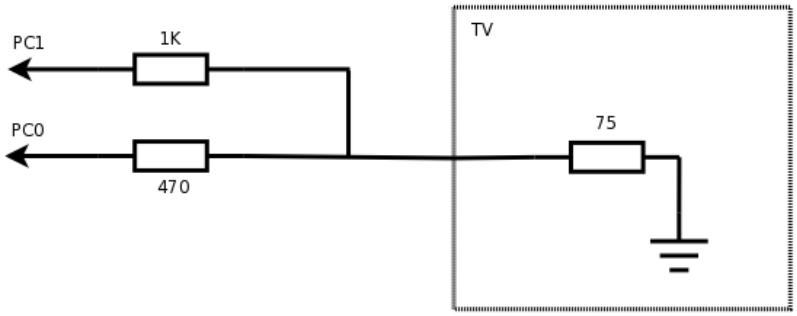
Joystickul este alcatuit din cate un potentiometru pentru fiecare axa. In pozitia de repaus, fiecare

potentiometru are o rezistenta de aproximativ 50kΩ; rezistentele corespunzatoare celor 2 capete de axe sunt 0 si 100kΩ. Rezistenta unui potentiometru are o dependenta liniara fara de pozitia sa. Rezistentele sunt convertite in tensiuni, care sunt citite apoi de catre microcontroller si convertite cu ajutorul ADC-ului incorporat.

3. Hardware design

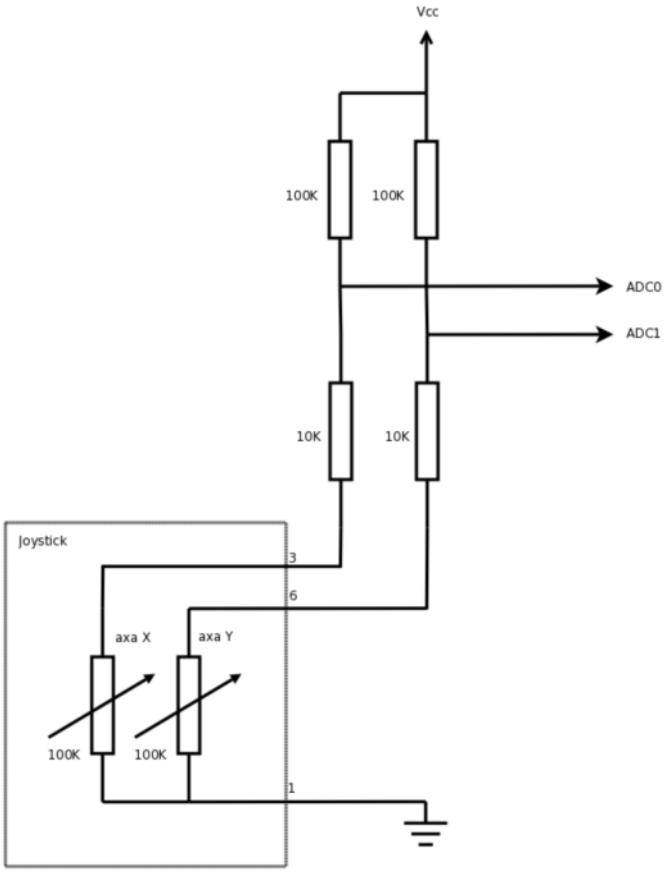
Am folosit doar 4 dintre porturile ATMEGA16 pentru comunicare. PA0 (ADC0) si PA1 (ADC1) sunt intrarile folosite pentru a citi tensiunile generate de convertorul de la intrare. PC0 si PC1 sunt folosite ca iesiri care comanda DAC-ul.

Semnalul TV este reprezentat prin tensiuni intre 0 si 1V. Tensiunea de 0V reprezinta un semnal de sincronizare; nuantele de gri sunt reprezentate prin tensiuni intre 0.33V si 1V, prima valoare corespunzand unui punct negru, iar a doua unui punct alb. Deoarece pe 2 biti se pot reprezenta 4 valori, am mai ales si tensiunea de 0.67V prin care se reprezinta puncte gri. Tensiunile generate de schema de mai jos sunt reprezentate in tabel.



PC1	PC0	V_{out}	Constanta C	Culoare
0	0	0	TV_SYNC	Semnal de sincronizare
0	1	0.67	TV_GREY	Gri
1	0	0.33	TV_BLACK	Negru
1	1	1	TV_WHITE	Alb

ATMEGA16 nu detine intrari cu care sa se poata masura rezistente; in schimb, intrarile sale pot fi folosite ca intrari analogice ale caror valori pot fi convertite cu un ADC intern. Acesta poate fi configurat o tensiune de referinta V_{ref}=2.56V, aproximativ egala cu jumatate din tensiunea de alimentare V_{cc}. Folosind divizoare de tensiune, putem genera tensiuni direct proportionale cu pozitiile potentiometrelor din joystick.



4. Software design

Am scris aproape tot programul in limbajul C, folosind compilatorul *avr-gcc*, care in Windows face parte din pachetul *WinAVR*. In anumite locuri am folosit cod in asamblare introdus in sursele C cu directiva *__asm__*. Un astfel de loc este functia de generare a semnalului video, unde am avut nevoie de pauze de durate precise. Pentru a simplifica procesul de compilare, am scris un *Makefile* care compileaza fisierul sursa. Folosind comanda *make program* poate fi pornita programarea microcontrollerului.

Semnalul TV este generat digital de catre program si apoi convertit cu ajutorul DAC-ului prezentat mai sus. Imaginea de pe ecran este reprezentata printr-o matrice de 19x23 celule, fiecare celula avand una dintre culorile de mai sus. Fiecare linie din matrice reprezinta 14 linii impare si 14 linii impare pe ecran. Matricea este reprezentata in RAM-ul microcontrollerului prin variabila *cell*.

Imaginea de pe televizor este actualizata de 25 de ori pe secunda. Fiecare cadru PAL este format din 625 de linii; fiecare linie dureaza astfel 64µs. Imaginea este transmisa intretesuta; se transmit intai cateva semnale de sincronizare, apoi liniile impare de pe ecran, apoi cateva linii de sincronizare, apoi liniile pare. Deoarece nu as fi reusit sa scriu fara un efort major cod in asamblare care sa dureze exact 64µs, am folosit timerul din ATMEGA16 pentru a executa o functie la un interval de 64µs. Fiecare apel al functiei genereaza semnalul pentru o linie de pe ecran. Codul pentru iniile de sincronizare si cele din afara ecranului este foarte simplu si rapid, activand semnalul corespunzator pe portul C si iesind din functie. Pentru liniile care contin informatie utila, functia trebuie sa ruleze pe toata perioada data, modificand valoarea portului in functie de celula curenta. Dupa generarea semnalului pentru linia

curenta, este incrementata variabila globala care retine numarul liniei. Primele 4µs dintr-o linie reprezinta semnalul de sincronizare orizontala; apoi urmeaza un interval de 8µs care nu este afisat pe ecran. Afisarea trebuie facuta intr-un interval de 52µs, dar nu am reusit sa acopar in intregime liniile din cauza timpului prea mic ramas dupa executarea instructiunilor din functie.

Desi standardul PAL specifica un format foarte strict si destul de complex al liniilor de sincronizare, programul meu a functionat cu urmatorul format: 2µs de TV_BLACK apoi 62µs de TV_SYNC. Pentru introducerea pauzelor de durate fixe, am scris functia *usecDelay* care primeste ca parametru o anumita durata si ruleaza un program care sa se ruleze intr-un numar de cicluri de ceas egal cu de 16 ori durata data, valoarea 16 fiind data de frecventa microcontrollerului.

Bucla principala a programului asteapta trecerea unui anumit interval (64ms) si face actualizarea starii jocului. Primul pas este pornirea ADC-ului pe intrari (alternativ) pentru a calcula pozitiile joystick-ului pe cele 2 axe. In functie de valorile din registrele ADC-ului, se recalculeaza orientarea “sarpelui”. Al doilea pas, care se efectueaza la intervale mai mari de timp (512 ms) este generarea urmatoarei celule din sarpe si “taierea cozii” acestuia. Al treilea pas este generarea “punctelor” pe care le poate manca sarpele pentru a isi mari dimensiunea. Pentru a da un caracter aleator pozitiilor generate, am folosit generatorul de numere pseudoaleatoare din *avr-libc,* adica functiile *rand()* si *srand()*. Punctele de “mancare” vor aparea in pozitii aparent aleatoare, dar in realitate bine definite de valoarea *seed-ului* din codul sursa; punctele vor aparea de fiecare data in aceleasi pozitii. O imbunatatire care poate fi adusa proiectului pe care nu am reusit s-o realizez din cauza timpului este folosirea unui generator de numere pseudoaleatoare cu entropie care sa foloseasca evenimentele generate de utilizator (in acest caz, miscarea joystickului) pentru a da un caracter cu adevarat aleator pozitiilor. Pot fi folosite valorile din registrul ADC si intervalele de timp dintre 2 masuratori la care directia se modifica; este foarte greu pentru un jucator sa repete aceeasi secventa de miscari la 2 jocuri diferite.

Sarpele este reprezentat in memorie prin coordonatele “capului” si ale “cozii” in matrice, orientarea capului sarpelui (in ce directie se va « lungi » sarpele la urmatorul pas) si printr-o lista simplu-inlantuita a pozitiilor sarpelui in care in fiecare celula sunt rezervati 2 biti pentru stocarea directiei catre urmatoarea celula din sarpe (cate o valoare pentru fiecare din cele 4 puncte cardinale). Miscarea sarpelui este echivalenta cu taierea unui segment din “coada” si adaugarea unui segment in “cap”, dupa orientarea capului. Aceste operatii se fac doar asupra variabilelor corespunzatoare si a maxim 2 pozitii din *cell*, deci necesita un timp foarte scurt. Daca sarpele trebuie sa creasca in lungime, la pasul urmator sarim peste pasul de “taiere din coada” si doar adaugam un nou bloc in “cap”. Structura unei valori din *cell* este :

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
X	**X**	**X**	**X**	directie		culoare	

X = pozitii nefolosite

directie = directia catre urmatoarea celula din sarpe ; valorile posibile sunt date in fisierul **snake.c** in tipul **Directions**

culoare = culoarea pe display a celulei ; ia una dintre valorile TV_XXX

O variabila globala *dead* ia valoarea 1 daca sarpele a murit, ciocnindu-se de sine sau de unul dintre pereti. Din momentul mortii sarpelui, starea jocului nu se mai actualizeaza si se aprinde LED-ul rosu de pe placa, LED conectat la pinul PD7 conform schemei de baza. Pe placa exista un buton pentru resetarea jocului, conectat printr-un jumper la pinul RST al ATMEGA16. La apasarea acestui buton, microcontrollerul se reseteaza.

5. Functionare

Proiectul a functionat aproape conform asteptarilor. Controlul cu joystickul a functionat perfect, dar la afisare apar artefacte vizuale (distorsiuni in imagine) din cauza intreruperii rutinei de generare a semnalului PAL de catre celeilalte intrerupere de *timer* si de catre evenimentele generate de ADC. La miscarea joystickului, unii pixeli de pe ecran se deplaseaza fata de cei de pe liniile adiacente.

In rest, functionarea proiectului a fost perfecta. Folosirea unui interval mic intre citirile ADC-ului a asigurat un raspuns bun al programului la comenzile date de utilizator si nu au aparut intarzieri sau erori.

6. Concluzii

Am reusit sa realizez aproape toate scopurile initiale. O imbunatatire pe care n-am mai apucat sa o fac si care poate fi realizata in viitor este introducerea generatorului aleator de pozitii unde apar punctele de mancare, dar proiectul este perfect functional si fara aceasta componenta.

Cea mai complexa parte a proiectului a fost generarea semnalului video. Desi pentru acest proiect nu a fost necesar, pot fi aduse imbunatatiri la codul scris de mine pentru a genera o rezolutie mai mare (se pot obtine 120-140 de pixeli pe fiecare rand folosind cod optimizat scris in limbaj de asamblare). Cu toate acestea, generarea semnalului video folosind un microcontroller este o varianta proasta. O varianta mult mai buna ar fi fost folosirea unui FPGA pentru generarea semnalului, microcontrollerul fiind conectat la acesta prin TWI, SPI sau printr-o interfata proprie ; in acest caz, ATMEGA16 ar transmite doar comenzi de scriere si citire in/din memoria video aflata in FPGA sau externa, pe care FPGA-ul ar citi-o separat. De asemenea, nu este posibila generarea unui semnal color cu un microcontroller.

Alte variante studiate pentru interfata cu utilizatorului au fost : butoane si accelerometre pentru comanda si afisarea pe un LCD, monitor sau transmiterea prin interfata seriala de comenzi catre calculator. Din toate variantele, cele folosite in final in proiect au fost cele mai simple de implementat si au fost potrivite aplicatiei alese.

Comanda prin accelerometre ar fi presupus miscarea sarpelui in functie de inclinarea placii ; problema intalnita a fost lipsa unor accelerometre potrivite in magazinele din Romania.

Afisarea pe calculator ar fi presupus scrierea unei aplicatii Windows sau Linux care sa afiseze intr-o fereastra un bitmap care sa poata fi modificat prin mesaje primite pe portul serial ; microcontrollerul ar fi folosit interfata seriala (USART) pentru a trimite aceste mesaje. Principiul de utilizare este acelasi ca la solutia cu FPGA sau cu LCD. O varianta studiata a fost implementarea unui subset minimal al protocolului X11 ; mesajele protocolului ar fi fost transmise prin interfata seriala catre o aplicatie-wrapper de pe un calculator pe care sa ruleze Linux, aplicatia trimitand mai departe mesajele catre serverul X local.

7. Surse de informatie folosite

- *Characteristics of B,G_PAL and M_NTSC television systems*
  http://www.kolumbus.fi/pami1/video/pal_ntsc.html
- *Retroleum - UK PAL TV timing and voltage specifications.mht*

http://www.retroleum.co.uk/PALTVtimingandvoltages.html

- *Rickard's electronic projects page – Howto* http://www.rickard.gunee.com/projects/video/sx/howto.php
- *ATMEGA16 datasheet* http://www.atmel.com/dyn/resources/prod_documents/doc2466.pdf
- *AVR 8-bit instruction set* http://www.atmel.com/dyn/resources/prod_documents/doc0856.pdf
- *AVR Libc User Manual* http://www.nongnu.org/avr-libc/user-manual/index.html