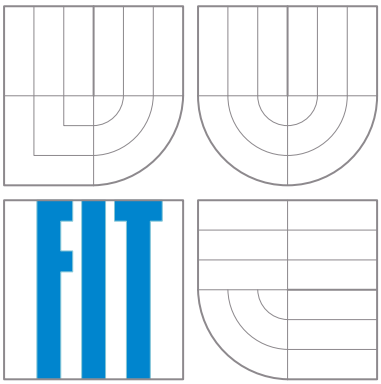

Dokumentácia k projektu pre predmet IVH

Hra Breakout

22. apríla 2013

Autor: Ivan Ševčík, xsevci50@stud.fit.vutbr.cz
Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

Obsah

1 Úvod

Ako projekt pre predmet IVH som si zvolil implementáciu hry Breakout na `FPGA` v zariadení `FitKit`.

1.1 Popis hry

Breakout alebo Arkanoid sú hry z roku 1976, resp. 1986. Tú prvú vyrobila firma Atari a stala sa predlohou pre všetky ostatné hry tohto štýlu, ktoré iba pridávali nový obsah, ale princíp zostal rovnaký. Hra bola inšpirovaná už existujúcou hrou Pong z roku 1972, taktiež od Atari.

Základými prvkami hry sú tzv. pádlo, loptička a bloky. Úlohou hráča je zničiť všetky bloky pomocou loptičky tým, že do nich loptičkou narazí a zabrániť jej pádu na spodný okraj hracej plochy. To sa dá dosiahnuť odrazom od pádla, ktoré je možné ovládať. Na tomto jednoduchom princípe sa dá ďalej stavať, napríklad pridáním rôznych druhov blokov alebo bonusov, ktoré hru rôzne ovplyvňujú.

1.2 Zadanie problému

Naprogramovať hru na `FPGA` za použitia vnútornej pamäte, `VGA` výstupu a zvolenej periférie pre ovládanie hry. Hra bude spočiatku zjednodušená – loptička sa bude pohybovať iba po diagonále, jej rýchlosť bude konštantná a v hre sa nebudú vyskytovať bonusy. Pri náraze loptičky do bloku, na stenu alebo pádlo sa loptička odrazí o 90 stupňov. Pri náraze do bloku sa navyše blok zničí (to môže byť neskôr rozšírené o rôzne druhy blokov). Hra končí víťazstvom pri zničení všetkých blokov alebo prehrou pri dopade na spodný okraj obrazovky.

2 Návrh riešenia

Pre popis logiky hry budem používať väčšinou behaviorálny popis. Hra bude bežať v rozlíšení 640 na 480 a maximálny počet blokov bude 30 krát 30. Jednotlivé bloky, ktoré treba zničiť, uložím do pamäte `RAM`, pre polohu loptičky a pádla mi budú stačiť dva signály – hodnota na osi x a y. Hra bude ovládaná myšou cez `PS2` port, výstupom bude obraz na monitore prenášaný cez `VGA` port. Pre samotnú hru vytvorím stavový stroj (`FSM`) so stavmi: *PAUSE, MOVE, UPDATE, LOAD, RAM_READ, BOUNCE, RAM_WRITE, VICTORY DEFEAT*. Následuje popis jednotlivých stavov:

PAUSE	Cyklický stav, ktorý je možné prerušiť iba stlačením klávesy pre pauzu.
MOVE	Aktualizuje sa poloha pádla a loptičky.
UPDATE	Prepočítajú sa odrazy od pádla, stien a prípadný dopad na spodný okraj obrazovky, pri ktorom sa hra končí – stroj prejde do stavu *DEFEAT*.
LOAD	Vystavenie adresy požadovaného bloku na vstup RAM.
RAM_READ	Prečíta sa hodnota vystavená na výstupe RAM.
BOUNCE	Podľa prečítanej hodnoty sa loptička odrazí a na vstup RAM sa vystaví aktualizovaná hodnota.
RAM_WRITE	Nová hodnota bloku bola zapísaná do RAM, bit pre povolenie zápisu sa nastaví na false (0).
VICTORY	Koncový stav, ktorý nastáva pri zničení všetkých blokov.
DEFEAT	Koncový stav, ktorý nastáva pri dopade loptičky na spodný okraj obrazovky.

Tabuľka 1: FSM

Medzi stavmi sa bude prechádzať v uvedenom poradí s výnimkou *VICTORY* a *DEFEAT*, teda za *RAM_WRITE* bude znovu nasledovať *PAUSE*. Navyše stavy *LOAD, RAM_READ, BOUNCE*

RAM_WRITE budú zreťazené do cyklu a pre každý blok, do ktorého by loptička mohla v danom momente vrážať, sa týmto cyklom raz prejde. Z hlavného hodinového signálu na `FitKit`-e vytvorím pomocou deliča hodinový signál hry, ktorý bude nepriamo vyjadrovať rýchlosť. Následne tento nový signál využijem pre zmenu stavu `FSM`.

`FPGA` bude obsahovať aj radič a kontrolér pre `PS2` myš, ktorý bude aktualizovať hodnotu na signále posunu pádla. Poslednou komponentou bude `VGA` kontrolér, ktorý bude vystavovať hodnotu farby na základe pozície vyžiadanej monitorom.

3 Realizácia riešenia

Prvým krokom je vytvoriť hraciu plochu. Obrazovku som rozdelil na stĺpce šírky 32 pixelov a riadky výšky 16 pixelov, čo sa dá vo `VHDL` spraviť jednoduchým pripojením signálu od piateho, resp. štvrtého bitu zo signálu s pozíciou vykreslovaného pixelu. Na adresovanie 30`x`30 blokov budem potom potrebovať 5 bitov v oboch rozmeroch. Keďže 5 bitov umožňuje 32 rôznych hodnôt, využijem prvé 2 stĺpce a riadky ako okraj hracej plochy. Bloky sú v `RAM` uložené tak, že po umiestnení piatich bitov stĺpca v obrátenom poradí za päť bitov riadku získam adresu bloku v `RAM`, s pozíciou stĺpec, riadok.

Každá bunka v `RAM` má veľkosť 1 byte. Ten som rozdelil na polovicu – vrchné 4 bity určujú typ bloku a spodné 4 jeho farbu. To umožňuje jednoduché definovanie blokov pomocou hexadecimálnej sústavy. Ich hodnoty majú význam podľa tabulky:

`0`	Prázdny blok.
`1`	Bežný blok.
`2`	Nezničiteľný blok.
`3`	Blok vyžadujúci 2 údery.
`4`	Blok vyžadujúci 3 údery.

Tabuľka 2: vrchné 4 bity – typ

`0`	Čierna.
`1`	Červená.
`2`	Žltá.
`3`	Oranžová.
`4`	Zelená.
`5`	Bledomodrá.
`6`	Modrá.
`7`	Hnedá.
`8`	Zašednutá biela.
`9`	Oceľovo sivá.
`A`	Fialová.
`B`	Purpurová.
`C`	Bledoružová.

Tabuľka 3: spodné 4 bity – farba

Platí, že blok typu 4 sa po náraze loptičky zmení na blok typu 3 a ten na blok typu 1. Pritom sa zmení aj jeho farba, prechod z typu 4 na 3 zmení farbu na purpurovú a prechod z typu 3 na 1 zmení farbu na bledoružovú. Je vhodné preto pre blok typu 4 zvoliť fialovú farbu a pre blok typu 3 purpurovú. O zobrazenie farebného pixelu na obrazovke sa stará radič `VGA`. Ten využíva trojbitové

rozslíšenie pre každú farebnú zložku RGB. Následne sú D-A prevodníkmi hodnoty skonvertované na elektrický signál, ktorý je privedený na VGA port. Bitovú reprezentáciu jednotlivých farieb je možné nájsť v zdrojovom kóde FPGA. Ďalšie farby je samozrejme možné pridať, no vzhľadom na obmedzenú paletu sa pravdepodobne nepodarí vytvoriť dostatočne odlišnú farbu.

Následne bolo potrebné vytvoriť loptičku a udeliť jej pohyb. Loptička je pre zjednodušenie iba štvorec a má konštantú veľkosť. Jej pozícia je uložená v dvoch signáloch popisujúcich súradnice pixelu v ľavom hornom rohu loptičky. Pre pohyb sú potrebné ďalšie dva signály – pohyb na x a y. Tu už je nutné implementovať FSM popísaný v tabulke 1, ktorý určuje, aká operácia sa má s loptičkou vykonať. Za zmienku stojí hlavne algoritmus pre analýzu nárazu loptičky. Keďže bloky sú obdĺžniky, môže loptička naraziť ôsmimi spôsobmi – do ich hrany alebo do rohu. Podľa polohy loptičky v mriežke blokov sa volí, do ktorých blokov by loptička mohla naraziť, tie sa následne načítajú z RAM a ak sa nejedná o prázdne bloky, loptička sa odrazí. Špeciálny je prípad, keď loptička naráža napríklad ľavým horným rohom, no existuje blok vlavo aj hore. V tom prípade sa loptička odrazí ako pri náraze na roh a úder získajú oba bloky. Do rohu je možné naraziť, iba ak v danom smere nie je možné najskôr naraziť do hrany iného bloku.

Nakoniec som vytvoril pádlo pre odraz loptičky. To opäť využíva dva signály pre súradnice pixelu ľavého horného horného rohu. Signál popisujúci súradnicu y však je konštantný, rovnako aj jeho šírka a výška. Jeden signál je potrebný pre pohyb pádla. Ten bude aktualizovaný kontrolérom myše. V stave *MOVE* je potom tento pohybový signál pripočítaný k jeho polohe, nikdy sa však pádlo nemôže dostať za okraj hracej plochy.

4 Záver

Hru sa podarilo úspešne implementovať a až na občasnú chybu v logike odrazu loptičky sa jej správanie zhoduje so zadaním. Napriek návrhu bolo ovládanie realizované pomocou klávesnice, keďže vytvoriť kontrolér pre myš bolo oveľa náročnejšie, než sa na prvý pohľad zdalo. Popis ovládania sa nachádza v nápovede programu. Keďže reinicializácia dát v RAM by bola bez použitia MCU náročná, pri požiadavke na novú hru sa reinicializuje celý FitKit.

Hru by bolo možné ďalej vylepšiť napríklad implementovaním pohybu loptičky vo všetkých smeroch, rôznym uhlom odrazu pri dopade na pádlo alebo bonusmi na zrýchlenie a spomalenie hry alebo rozšírenie a zúženie pádla. Rovnako zaujímavé by bolo rozšírenie o načítanie levelu z MCU, čo by umožnilo aj prirodzené spustenie novej hry, priadnie počítadla skóre alebo viacerých životov.

A Metriky kódu

Počet súborov: 3 súbory
Počet riadkov zdrojového textu: 623 riadkov

B Fotodokumentácia

Výstupy jednotlivých fáz vývoja boli zaznamenávané mobilným telefónom, výslednu fotodokumentáciu je možné nájsť na `https://www.dropbox.com/sh/dv7yxa021wfqru8/wBQxnxWywy`.