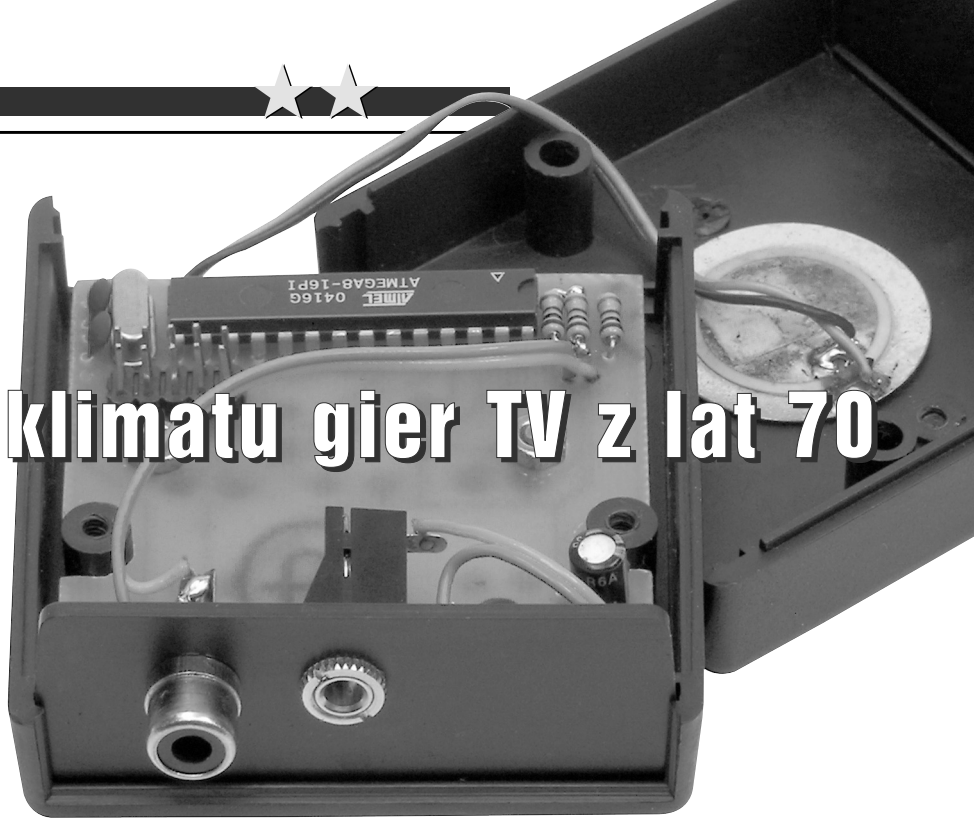
Snake,

czyli powrót do klimatu gier TV z lat 70



kit 2806 AVT

Na pewno każdy z Was, drodzy Czytelnicy, miał kiedyś w ręku telefon firmy Nokia i grał w legendarną już grę Wąż (z ang. Snake). Wiele osób zapewne przesiedziało długie godziny nad telefonem komórkowym, zjadając kolejne kropki powyginanym i wypełniającym prawie cały ekran wężem. Ale czy wiecie, że pomysł na tę grę nie jest wcale autorstwa firmy Nokia? Jej pierwsze odmiany pojawiły się już w latach 70. na automatach do gier i następnie, w latach 80. na domowych konsolach. Prawdziwą popularność jednak zdobyła na początku lat 90., gdy Microsoft wydał grę Nibbles napisaną w QBasicu i dołączoną do tego kompilatora jako przykład. Był to właśnie nasz Wąż. Pomysł reaktywowała ponownie Nokia w okolicy 1998 roku. Od tego czasu w telefonach pojawiają się coraz to nowe odmiany tej gry. Musimy manewrować ciałem węża w labiryncie, w zbieranych kropkach mogą się znajdować bonusy, możemy też przechodzić przez ściany itp. Ja jednak „wychowałem" się na klasycznej wersji tej gry, bez zbędnych dodatków, i takiej jestem zwolennikiem.

Artykuł prezentuje grę w węża, jednak polem gry nie jest malutki ekran telefonu komórkowego, a ekran telewizora. Pomysł powstał przypadkiem podczas prób i eksperymentów z generowaniem obrazu telewizyjnego przez mikrokontroler AVR. Nie chciałem jednak tworzyć kolejnej wersji Tetrisa lub Ponga - w Internecie można znaleźć co najmniej kilka takich projektów. W pewnym momencie oświeciło mnie – nigdy nie spotkałem się z Wężem, w którego można by grać na telewizorze! Efektem prac jest kompletna gra oparta o procesor ATMega8, zamknięta w niewielkiej obudowie. Do sterowania służy 5 przycisków, a do pracy niezbędny jest tylko zasilacz 12V z wtyczką jack oraz kabel cinch do połączenia z telewizorem. W obudowie zainstalowana jest membrana piezo, dzięki której całość będzie radośnie pikać podczas gry. Niestety obraz generowany na telewizorze jest czarno-biały (a dokładniej, występują kolory: czarny, szary i biały), ponieważ przy zachowaniu niskich kosztów całości i możliwej prostoty, nie da się procesorem AVR stworzyć kolorowego obrazu. Wymagałoby to zastosowania szybszego i droższego procesora lub równie drogich specjalizowanych układów, przy których cała zabawa traci sens. Czarno-biały obraz ma za to taką zaletę, że przywołuje niesamowity klimat pierwszych gier telewizyjnych z lat 70.

Układ jest na tyle nieskomplikowany, że może go złożyć nawet bardzo początkujący Czytelnik. Natomiast dla bardziej dociekliwych będzie źródłem informacji oraz inspiracji do własnych opracowań tego rodzaju.

Opis układu

Opis układu proponuję zacząć od opisania tajników obrazu telewizyjnego w standardzie PAL. Układ generuje obraz typu Composite, czyli złożony. Jest to sygnał małej częstotliwości, w którym za pomocą jednego przewodu przesyłana jest informacja o obrazie, kolorach oraz synchronizacji. Sygnał taki jest przesyłany zazwyczaj przez popularne cinche (nie należy tego mylić z sygnałem antenowym wysokiej częstotliwości). Większość Czytelników zapewne spotkała się z rysunkami poglądowymi opisującymi zasadę działania telewizora. Wyświetlany obraz składa się z 625 poziomych linii (z czego tylko około 576 jest widocznych) „rysowanych" na wewnętrznej powierzchni kineskopu pokrytej luminoforem. Strumień elektronów powoduje świecenie luminoforu, a powtarzanie cyklu rysowania wszystkich linii 50 razy na sekundę daje złudzenie stałego obrazu. Jednak TV w jakiś sposób musi się dowiedzieć, kiedy należy zacząć rysować każdą linie oraz kiedy zaczyna się kolejna klatka obrazu. Do tego celu służą impulsy synchronizacji. Na **rysunku 1** można zobaczyć, jak wyglądają dwie następujące po sobie linie czarno-białego obrazu. Czas trwania jednej linii to ok. 64µs. Można w nim wyróżnić cztery etapy:

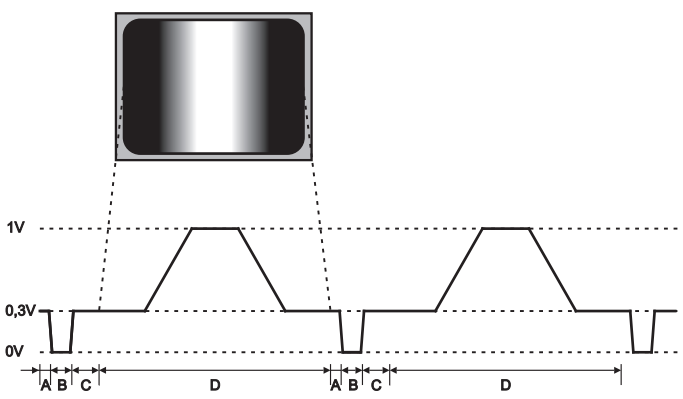
A – przedni przedsionek – odstęp pomiędzy poprzednią linią a impulsem synchronizacji, trwa ok. 2µs.

B – impuls synchronizacji poziomej, trwa ok. 4µs.

C – tylny przedsionek – odstęp pomiędzy impulsem synchronizacji a obrazem, w tej części sygnału są zakodowane informacje dla dekodera koloru, trwa ok. 6µs.

D – obraz, trwa ok. 52µs, z czego na ekranie

Rys. 1



jest widoczne od 40 do 48µs, zależnie od odbiornika (reszta pozostaje „schowana" za krawędziami ekranu).

Jak widać na przykładzie, maksymalna amplituda sygnału wynosi 1V. W części sygnału odzwierciedlającej obraz, wartości napięcia odpowiadają skali szarości, przy czym ok. 0,3V reprezentuje kolor czarny, a 1V – kolor biały. Impuls synchronizacji musi, mieć napięcie równe 0V.

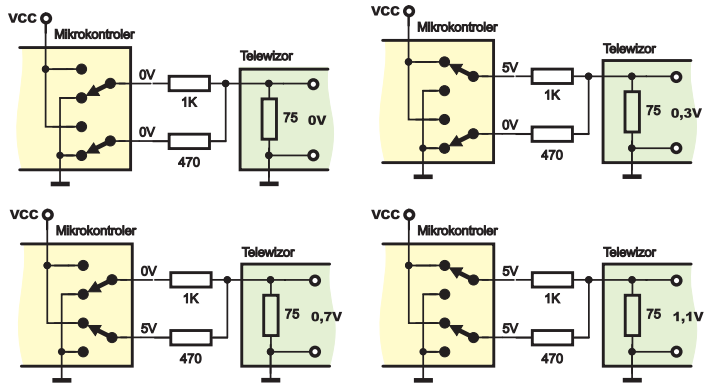
Synchronizacja pionowa informująca TV o nadejściu nowej klatki obrazu jest procesem trochę bardziej złożonym niż pozioma. W standardowym obrazie występuje tzw. przeplot – rysowany jest on w dwóch fazach. W fazie pierwszej kreślone są linie o numerach nieparzystych (1 3 5 7…), w drugiej – parzyste (2 4 6 8 …). W ten sposób uzyskujemy pełną klatkę obrazu zawierającą 625 linii. Jak łatwo się domyślić, za jednym razem ekran jest zapełniany maksymalnie 312,5 liniami (połowa z 625). Informacja o tym, czy aktualnie przesyłane są linie parzyste, czy nieparzyste, jest kodowana za pomocą odpowiedniej serii impulsów synchronizacji pionowej. Nasz generator nie będzie wykorzystywał jednak tej metody. Praktyka pokazuje, że do poprawnej pracy odbiornika telewizyjnego wystarczy impuls synchronizacji pionowej przypominający poziomą (czyli o poziomie 0V), ale trwający ok. 128µs (czas trwania dwóch linii). Następnie należy przesłać 310 linii obrazu, z czego widocznych jest 250 – 280. Pozostałe powinny być całe czarne, czyli zawierać tylko impuls synchronizacji poziomej. Dobierając odpowiednio liczbę linii przed i po właściwym obrazie, można ustawić go idealnie w środku ekranu.

Ten sposób generowania obrazu jest mało elegancki i pozwala na uzyskanie rozdzielczości pionowej maksymalnie 256 pikseli, lecz jest prosty w implementacji i działa z praktycznie wszystkimi odbiornikami TV. Podobna metoda była stosowana z powodzeniem w starych komputerach podłączanych do telewizora (np. ZX80).

Przy próbach wygenerowania obrazu za pomocą procesora AVR pojawiają się dwa podstawowe problemy. Pierwszy to poziomy sygnałów – potrzebne są 0-1V, a procesor dostarcza na swoich portach co najwyżej 0 lub 5V. Rozwiązaniem jest stworzenie bardzo prostego przetwornika cyfrowo-analogowego, wykorzystującego dwa porty wyjściowe procesora, który można zobaczyć na **rysunku 2**. Dwa rezystory w połączeniu z rezystorem 75Ω znajdującym się na wejściu telewizora tworzą dzielnik napięcia. Wartości są tak dobrane, aby dało się uzyskać napięcia 0V, 0,3V, 0,7V i 1,1V (odpowiada to poziomowi synchronizacji, poziomowi czerni, kolorowi szaremu i bieli). Rysunek prezentuje wszystkie możliwe stany wyjściowe portów i odpowiadające im napięcia.

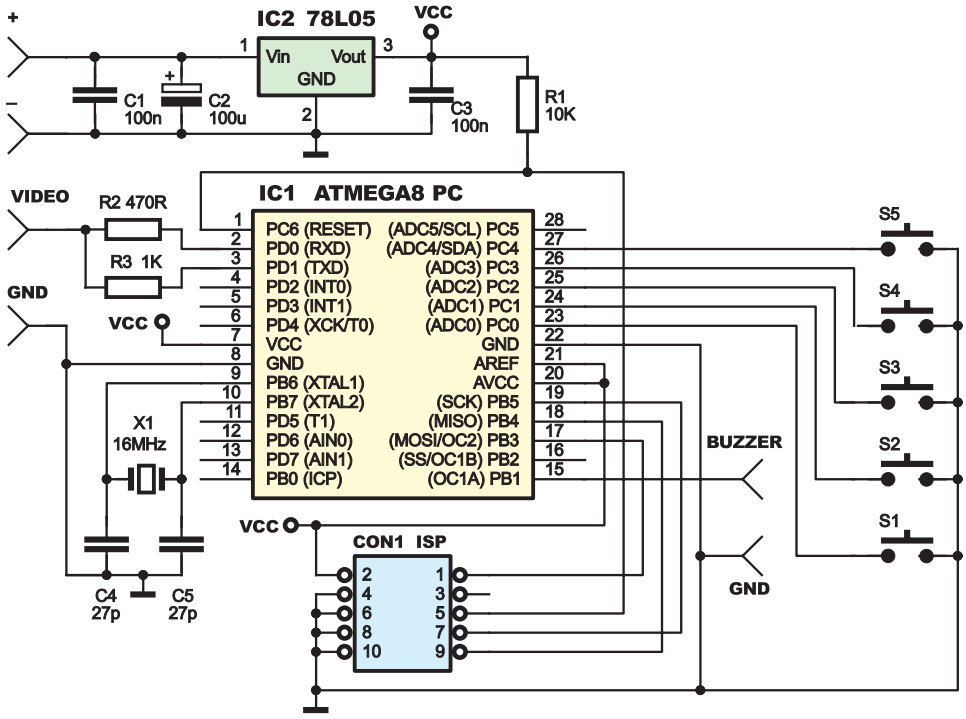
Drugim problemem jest prędkość procesora. Jak wyżej wspomniano, część widzialna obrazu trwa ok. 52µs. Aby mieć pewność, że obraz nie będzie obcięty po bokach, należy skrócić ten czas do np. 40µs. Chcąc uzyskać rozdzielczość w poziomie, np. 120 pikseli, procesor musi być w stanie zmieniać napięcia na wyjściu co 300ns. Przy zegarze taktującym procesor a równym 16MHz daje to czas na wykonanie zaledwie ok. 5 instrukcji! W tym projekcie zdecydowałem się zastosować dużo niższą rozdzielczość obrazu – 40 pikseli w poziomie i 32 piksele w pionie. Jest zupełnie wystarczająca do rysowania napisów, a w trybie gry – jeden punkt odpowiada jednemu elementowi ciała węża. Wizualną atrakcyjność podnosi istnienie dodatkowo koloru szarego. Więcej szczegółów na temat generowania obrazu przez program będzie omówione w dalszej części artykułu.

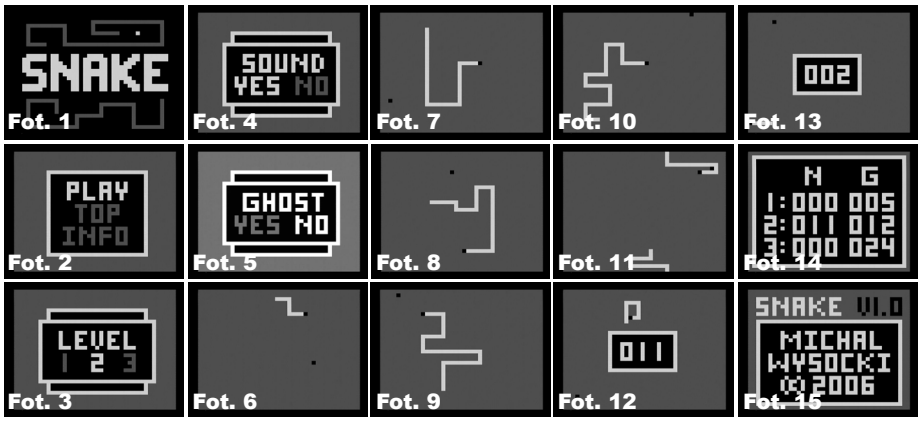
Schemat układu można zobaczyć na **rysunku 3**. Jego podstawowym elementem jest procesor ATMega8. Wybrałem go ze względu na dużą (1KB) ilość pamięci RAM, która jest potrzebna do przechowywania wyświetlanego obrazu oraz opisu kształtu ciała węża. Procesor napędzany jest kwarcem 16MHz, co daje duży zapas czasu procedurze generującej obraz. Do pinów portu C podłączonych jest pięć mikroprzełączników z 6mm klawiszami – dzięki temu można je wyprowadzić bezpośrednio przez otwory w obudowie na zewnątrz, bez żadnych dodatkowych nasadek. Przyciski są na tyle duże, że można je wygodnie naciskać palcami. Klawisze zostały ułożone tak, aby utworzyły strzałki góra, dół, prawo, lewo. Środkowy przycisk pełni funkcje potwierdzenia wyboru „OK". Ze względu na wbudowane rezystory podciągające w procesorze, nie trzeba było umieszczać ich na zewnątrz. Piny PD0 i PD1 podłączone są do rezystorów przetwornika cyfrowo-analogowego, a pin PB2 steruje blaszką piezo, generującą dźwięki. Do programowania procesora na płytce zostało umieszczone 10-pinowe złącze w najpopularniejszym standardzie STK200. Dzięki temu każdy posiadacz tego programatora będzie mógł szybko i wygodnie wgrać oprogramo-



Rys. 2

Rys. 3





wanie. Całość zasilana jest ze stabilizatora 78L05 w jego klasycznej aplikacji.

Jak widać, konstrukcja układu jest stosunkowo prosta – cały ciężar wykonywanego zadania przejmuje oprogramowanie. Na razie chciałbym pokazać to, co zapewne najbardziej Cię ciekawi, Czytelniku, od samego początku, czyli „jak właściwie wygląda ta gra, i co potrafi?". W **tabelce 1** znajdują się opisy zrzutów ekranu wykonanie za pomocą karty telewizyjnej w komputerze, z prawdziwej gry. Będę się nimi posiłkował przy opisywaniu naszej zabawki. Na obrazku 1 (**Fot. 1**) można zobaczyć ekran początkowy – pojawia się on zaraz po włączeniu zasilania oraz po zakończeniu gry, towarzyszy mu dźwięk przywitalny. Aby przejść do menu głównego (**Fot. 2**), wystarczy nacisnąć dowolny klawisz. Poruszamy się w nim za pomocą klawiszy góra / dół i zatwierdzamy wybór środkowym przyciskiem. Pierwsza pozycja (PLAY) rozpoczyna grę. Zostaniemy poproszeni o wybór poziomu gry (**Fot. 3**). Wszystkie levele (poziomy) różnią się miedzy sobą tylko prędkością poruszania się węża, podobnie jak w grach będących pierwowzorem. Zatwierdzamy wybór naciśnięciem OK. Kolejne dwie pozycje (**Fot. 4** i **5**) to włączenie lub wyłączenie dźwięków podczas gry oraz gra w trybie GHOST. Co to takiego? Jak sugeruje angielska nazwa (ghost – duch), pozwala nam to po prostu na przechodzenie przez ściany – w tym przypadku przez krawędzie ekranu. Wąż, który wpadł na brzeg ekranu, wychodzi po przeciwnej stronie (**Fot. 11**). Innowacja ta pojawiła się w kolejnych wersjach gry na telefonach. Pierwotnie mój wąż miał być pozbawiony takich „kwiatków", lecz zostałem zmuszony przez znajomych testujących grę do dodania tego trybu. Gra jest w nim o wiele prostsza – węża jesteśmy w stanie zabić tylko i wyłącznie zderzając się z własnym ciałem, a odpowiednio manewrując, można zapełnić jego ciałem cały ekran (choć jest to strasznie pracochłonne). Na fotografiach 6-9 możemy zobaczyć przebieg normalnej gry. Zaczyna się zawsze od węża, który składa się z 4 elementów. W przypadku śmierci węża (**Fot. 12** i **13**) usłyszymy smutny dźwięk i zobaczymy liczbę zebranych punktów. Podczas gry możemy włączyć pauzę, naciskając OK. Warto wspomnieć, że wszystkie wybrane ustawienia (poziom, wyłączenie dźwięków, ghost) oraz najlepsze wyniki są zapamiętywane w nieulotnej pamięci EEPROM procesora i domyślnie wybierane przy ponownym włączeniu.

Dwie pozostałe pozycje z menu głównego to TOP lista (**Fot. 14**) oraz informacje o autorze (**Fot. 15**). Lista najlepszych wyników posiada dwie kolumny, odpowiednio dla normalnej gry „N" i dla trybu ghost „G", oraz trzy wiersze dla trzech poziomów trudności.

Tab. 1

Fot. 1	Ekran startowy, przywitanie
Fot. 2	Główne menu
Fot. 3	Wybór poziomu gry
Fot. 4	Wybór włączenia lub wyłączenia dźwięków
Fot. 5	Wybór gry z przechodzeniem przez ściany lub bez
Fot. 6-10	Przebieg gry
Fot. 11	Gra w trybie GHOST z przechodzeniem przez ściany
Fot. 12-13	Koniec gry
Fot. 13	
Fot. 14	Tabela najlepszych wyników
Fot. 15	Informacja o grze i autorze

Tylko dla dociekliwych - oprogramowanie

Przejdę teraz do opisu serca projektu, czyli oprogramowania. Mniej zaawansowani Czytelnicy mogą opuścić ten fragment i przejść od razu do części poświęconej montażowi.

Oprogramowanie zostało napisane w C i skompilowane przy użyciu AVR-GCC (pakiet WinAvr). Można je ściągnąć z Elportalu EdW. Od razu powinienem zaznaczyć, że projekt nie jest stworzony zgodnie ze sztuką pisania programów, dlatego początkujący nie powinni się na nim wzorować. Wynika to z tego, że przechodził różne stadia rozwoju. Na samym początku składał się tylko z pliku 'snake.c', w którym de facto nie było nic, poza konfiguracją timerów oraz 'snake.s' zawierającego zalążek procedury wyświetlania obrazu. Procedura ta nie jest mocno skomplikowana, lecz jak to bywa z kodem napisanym w asemblerze, nawet ja, jako autor, po jakimś czasie mam pewne problemy z rozszyfrowaniem jej :-). Timer0 procesora jest skonfigurowany tak, aby wywoływać ją co 64µs, czyli raz na linię obrazu. Na początku kodu znajdują się typowe operacje odkładania rejestrów na stos, następnie seria pętli oraz warunków. Pierwsza pętla generuje 4µs opóźnienie, generując impuls synchronizacji. Następnie sprawdzane są numery aktualnie generowanej linii (0 do 311). Linie 0 do 255 są liniami wyświetlanymi na ekranie, kolejne kilkanaście linii tworzy dolną czarną ramkę, następnie 2 linie synchronizacji pionowej (0V na wyjściu Video przez 128µS) oraz aż do końca linie wygaszania pionowego i górna ramka obrazu. Po części zliczającej następuje pętla, której długością możemy regulować miejsce, w którym na ekranie zaczyna się obraz w każdej linii. Sama procedura wyświetlania kolejnych pikseli jest bardzo prosta. Jak łatwo policzyć, rozdzielczość 40x32 piksele daje nam 1280 komórek pamięci po 2 bity koloru. Kompresując całość po 4 komórki w bajcie, otrzymujemy 320 bajtów pamięci. Program wczytuje z pamięci bajt i kopiuje go bezpośrednio do portu D procesora. Następnie odczekuje około 800ns i przesuwa zawartość bajtu o dwa bity, i tak trzykrotnie. Całość powtarzana jest przez pętlę 10 razy, co daje nam w rezultacie 40 pikseli. Odpowiednie warunki na początku powodują, że ta sama linia jest wyświetlana 8 razy, dzięki czemu każdy piksel ma kształt kwadratu. Po zakończeniu procedury mamy około 10µs czasu aż do następnego wywołania, w którym to czasie procesor może wykonywać główny program. Niestety, pojawił się tutaj pewien mankament. Procesory AVR posiadają instrukcje, które wykonywane są w dwóch lub nawet trzech cyklach maszynowych. Skutkuje to tym, że jeśli przerwanie timera „wstrzeli" się w środek takiej długiej instrukcji, to musi odczekać, aż zostanie ona zakończona i czas pomiędzy kolejnymi wywołaniami nie jest równy idealnie 64µs. W rezultacie na ekranie obraz ma swoiste ząbki na swoich krawędziach, których kształt i rozkład zmienia się w zależności od kodu, jaki procesor wykonuje w głównym programie. Problem wydawał się trudny do obejścia, lecz w pewnej chwili uświadomiłem sobie, że wcale nie potrzebuję tego wolnego czasu procesora w każdej linii. Urządzenie komunikuje się tylko i wyłącznie z człowiekiem, którego czas reakcji jest na poziomie setek milisekund, dlatego wszystkie procedury wykonywane są podczas wyświetlania 56 czarnych linii tworzących ramkę, co 20ms (pełna klatka obrazu).

W głównym pliku projektu początkowo

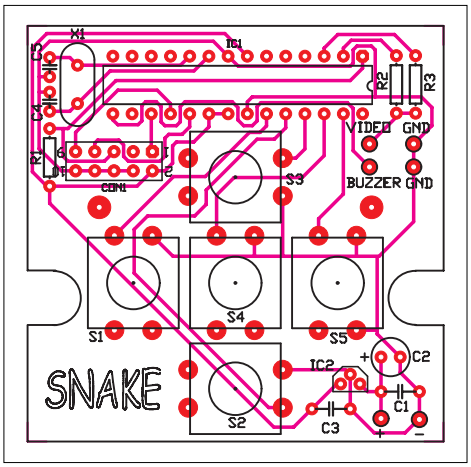
eksperymentowałem z wyświetlaniem różnych cieszących oko kształtów i prostych animacji. Jak się okazało, stworzenie procedury obsługującej i pamiętającej kształt węża nie było takie proste. Podstawowy problem, jaki się pojawił, to ilość zajmowanej pamięci – załóżmy, że gracz wypełni cały ekran ciałem węża w trybie GHOST. Daje nam to 1280 cząstek ciała, których pozycje trzeba zapamiętać. W praktyce ograniczyłem się do maksymalnie 1000 cząstek – wątpię, aby komuś udało się uzyskać taki wynik:-). Jeśli chcielibyśmy zapamiętać współrzędne każdego punktu, potrzebowalibyśmy prawie 1,5KB pamięci RAM, a to zdecydowanie przekracza zasoby naszego małego procesora. Zostało zastosowane inne rozwiązanie – każdy punkt jest opisany za pomocą 4 bitów (2 punkty na bajt), 2 bity informują o tym, czy kolejny element w kolejce znajduje się na prawo / lewo od obecnego, a 2 wskazują górę / dół. To rozwiązanie zmniejszyło nam tablice do 500 bajtów. Samo ciało węża tworzy niekończącą się kolejkę, która przy dojściu do 1000 elementu „zawijana" jest do elementu 0. Program posiada współrzędne i numer w kolejce pierwszego elementu (głowy węża) oraz ostatniego (ogona). Jeśli wąż ma się przesunąć o jedno pole, wtedy usuwamy ostatni element (zwiększamy numer ogona o 1) i dodajemy nowy element na jej początku, zgodnie z kierunkiem, w którym porusza się wąż. Jeśli akurat wąż trafił na kropkę – nie usuwamy ostatniego elementu, co skutkuje wydłużeniem ciała.

Program szybko się rozrastał, aż zaszła potrzeba, aby posegregować procedury w kilku plikach zewnętrznych. Z czystego lenistwa, nie powstały pliki nagłówkowe *.h – pliki *.c z procedurami są po prostu dołączane za pomocą #include do głównego pliku projektu. Nie będę tutaj szczegółowo omawiał wszystkich plików. Na uwagę zasługują tylko dwa rozwiązania – generowanie dźwięków oraz losowanie pozycji kropek do zjedzenia. Początkowo układ nie posiadał możliwości „pikania". Rozwiązaniem było podłączenie membranki piezo do pinu procesora, będącego wyjściem układu porównawczego timera1. Timer został skonfigurowany jako generator częstotliwości, której wartość zależy od rejestru OCR1A. Wydawałoby się, że dzięki temu generowanie dźwięków jest proste i bezbolesne. Niestety – musimy pamiętać o tym, że kod programu jest wykonywany co 20ms, co spowodowało, że czas krótkiego piknięcia nigdy nie był stały – czasami było to 20ms, a czasami 40. Poza tym generowanie dźwięków zauważalnie zatrzymywało na chwilę węża, co było dość irytujące. Rozwiązanie polega na tym, że program posiada globalną zmienną, której wartość informuje o tym, czy ma zostać wygenerowane piknięcie. Ze zmiennej tej korzysta… procedura generowania obrazu w asemblerze, która może łatwo włączyć dźwięk podczas jednej synchronizacji pionowej, a wyłączyć podczas następnej, co daje nam zawsze równe 20ms i nie zatrzymuje pracy głównego programu.

Generatory liczb losowych zawsze stanowiły problem w przypadku mikrokontrolerów. Ja użyłem do losowania współrzędnych punktów gotowej funkcji rand() znajdującej się w bibliotece 'stdlib.h'. Haczyk polega na tym, że jest to generator liczb pseudolosowych – liczby zawsze tworzą pewien matematyczny ciąg, który po jakimś czasie powtarza się i zaczyna zawsze od tej samej wartości. Z pomocą przychodzi tutaj funkcja srand(int), którą możemy zdefiniować od jakiej liczby generator ma zacząć pracę. W przypadku komputera PC wykorzystuje się po prostu aktualny czas systemowy, co zapewnia całkiem dobrą losowość wyniku. Co jednak zrobić w tym przypadku? Z pomocą przychodzi Timer2 procesora, oraz… użytkownik. Timer zostaje skonfigurowany tak, aby pracował non stop, lecz nie wywoływał żadnych procedur. Jego wartość jest przekazywana do funkcji srand w momencie, gdy zostanie naciśnięty dowolny przycisk, po wyświetleniu ekranu startowego. Czas, który upłynął od wyświetlenia do naciśnięcia jest zupełnie przypadkowy – nigdy nie uda się żadnemu człowiekowi uzyskać dwa razy takiego samego czasu. Dzięki temu inicjujemy generator wartością totalnie losową i mamy pewność, że nigdy nie powtórzy się taka sama sekwencja pozycji kropek na ekranie.

Montaż i uruchomienie

Gra została wykonana na bardzo prostej, jednostronnej płytce drukowanej (**rysunek 4**). Jej kształt został dopasowany do obudowy Z-67 i jej użycie zalecam. Montaż rozpoczynamy od wlutowania rezystorów oraz kondensatorów. Następnie montujemy podstawkę pod procesor, złącze programowania oraz kwarc i stabilizator. Kondensator C2 może mieć dowolną wartość większą lub równą 47µF, podobnie jak rezystor R1 1-100kΩ. Za to bardzo istotne są wartości rezystorów R2 i R3, które tworzą przetwornik D/A. Wstrzymujemy się na razie z wlutowaniem przycisków i przystępujemy do przygotowania obudowy.



Rys. 4

Z praktyki wiem, jak trudne może być precyzyjne wykonanie otworów pod przyciski, dlatego proponuję sprawdzoną metodę. Schemat montażowy płytki pokazany na rysunku 4 wycinamy lub kserujemy, a następnie układamy nadrukiem do góry, na dnie wewnętrznej strony górnej połówki obudowy (dolna posiada otwory na śrubki mocujące). W ten sposób uzyskaliśmy szablon, przez który zaznaczamy w plastiku ostrym szpikulcem środek każdego przycisku (zaznaczony kropką) oraz dwa otwory mocujące. Po usunięciu szablonu wiercimy otwory przycisków wiertłem 6,5mm oraz dwa otwory 3mm dla śrub mocujących. W prototypie wykonałem wgłębienia i zastosowałem śruby z łbem stożkowym, które nie wystają z obudowy. Płytkę od obudowy oddzielają plastikowe tulejki dystansowe. Można pominąć otwory mocujące, jeśli zdecydujemy się na wklejenie metalowych dystansów z gwintem, do których przykręcimy płytkę. W jednej ze ścianek bocznych montujemy gniazdko zasilające jack 3,5mm, oraz pojedyncze gniazdko cinch (należy je zamontować dość nisko, tak, aby nie zawadzały o płytkę) i łączymy je przewo-

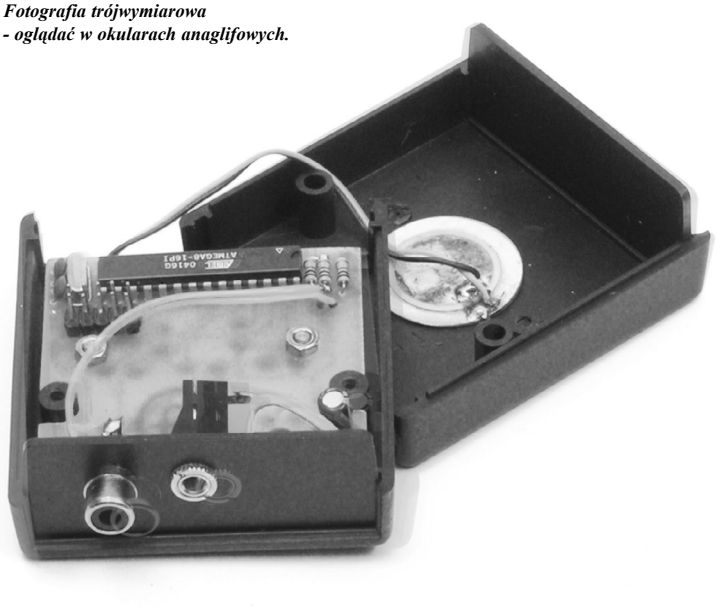
dami z płytką zgodnie z oznaczeniami (przypominam, że w gniazdku jack końcówka bolca to „+”, a dolna część to „-”, natomiast w gniazdku cinch środkowy bolec to sygnał, a zewnętrzny korpus to masa). Do dolnej części obudowy przyklejamy np. klejem na gorąco membrankę piezo i łączymy ją przewodami z polami na płytce oznaczonymi BUZZER oraz GND. Przyszedł czas na zamontowanie przycisków. Montujemy je **OD STRONY ŚCIEŻEK**, zaczynając od S1, S4, S5. Lutujemy tylko dwie nóżki, po przekątnej, co pozwoli nam na dogięcie i wyrównanie wszystkich przycisków, tak aby nie klinowały się w otworach obudowy. Po wyrównaniu, możemy dolutować pozostałe dwie nogi. Podobnie postępujemy z S2 i S3.

Ostatnim etapem jest zaprogramowanie procesora. W katalogu AVR poza plikami programu i plikiem snake.hex, znajduje się skrypt program.bat oraz katalog avreal. Program avreal jest to bezpłatny programator procesorów AVR obsługiwany z linii poleceń, a skrypt ‘program.bat’ wykona za nas wszystkie niezbędne operacje (ustawienie fuse’ów, kasowanie, zapis, weryfikacja). Podłączmy posiadany programator STK200 (dostępny jako kit w AVT) do portu LPT1 i naszego układu (należy pamiętać o podłączeniu także zasilania!). Uruchamiamy skrypt – jeśli wszystko przebiegło dobrze, program poinformuje nas o poprawnej weryfikacji. Mniej zaawansowanych Czytelników zachęcam do zakupu kitu z gotowym zaprogramowanym procesorem. Układ nie wymaga uruchomienia, powinien działać natychmiast po podłączeniu zasilacza 9 lub 12V.

Najlepszy wynik, jaki udało się uzyskać moim znajomym testującym grę, to 367 pkt., na poziomie średnim w trybie GHOST. Do maksymalnej wartości 999 pkt. jest więc daleko i mam nadzieje, że Czytelnicy poprawią znacznie ten wynik.

Michał Wysocki
mwsoft@o2.pl

Fotografia trójwymiarowa - oglądać w okularach anaglifowych.



Wykaz elementów

R1 . . . . . 10kΩ	CON1. . . . . jumper 5x2
R2. . . . . 470Ω	S1-S5 . . . . . microswith 6mm
R3 . . . . . 1kΩ	X1 . . . . . 16MHz
C1,C3 . . . . . 100nF	Buzzer
C2 . . . . . 100µF	Gniazdo cinch przykręcane do obudowy
C4,C5 . . . . . 27pF	Złącze minijack (do zasilacza)
IC1 . . . . . ATMega8-16PC	Podstawka DIP-28
IC2 . . . . . 78L05	Obudowa Z-67

Płytka drukowana jest dostępna w sieci handlowej AVT jako kit szkolny AVT-2806