kit 2853 AVT

AVRTV

czyli jak zmienić telewizor w wielki wyświetlacz

Kilka lat temu w Internecie zaczęły się pojawiać projekty gier typu Tetris czy Pong, zrealizowane na procesorach PIC. Zainspirowało mnie to do wykorzystania małych mikroprocesorów jednoukładowych do generowania obrazu telewizyjnego. Efektem zainteresowania tym tematem był projekt „Snake" opublikowany w EdW 12/2006 – gra w popularnego węża, wykorzystująca telewizor jako wyświetlacz. Czytając tamten artykuł, myślałem, że jest to maksimum, jakie da się uzyskać pod względem rozdzielczości (obraz miał rozmiar 40x25 dużych pixeli), jednak byłem w błędzie – procesor ATMEGA8 (użyty także w tym projekcie) posiada jeszcze dużo niezagospodarowanych możliwości!

AVRTV jest generatorem obrazu wideo, sterowanym za pomocą interfejsu RS232, który potrafi wyświetlić 25 linii po 40… **ZNAKÓW**! To daje rozdzielczość aż 320x200. Niestety, niewielka pamięć RAM procesora (1024 bajty) pozwala tylko na zapisanie kodów znaków do wyświetlenia (40 wierszy x 25 kolumn = 1000 znaków na cały ekran), a nie pełnej grafiki rastrowej. W pamięci programu zapisana jest czcionka 8x8 pikseli, mająca polskie znaki kodowane w systemie CP1250 (kodowanie Windows). Wszystkie ważniejsze parametry układu zebrane zostały w **tabeli 1**.

Układ został zrealizowany w postaci małego modułu, na jednostronnej płytce drukowanej i idealnie nadaje się do wbudowania do większego urządzenia. Interfejs RS232 jest wyprowadzony bezpośrednio, bez konwertera napięć, co pozwala na proste połączenie z głównym procesorem. Wpisywanie znaków odbywa się bardzo podobnie jak w zwykłym terminalu RS232 – znak odebrany zostaje wyświetlony na aktualnej pozycji, a kursor (który jest niewidoczny) przesuwa się w prawo. Po dojściu do końca wiersza przechodzi na początek kolejnego, a jeśli jest to ostatni wiersz ekranu, to cała zawartość zostaje przewinięta w górę i tworzy się nowa pusta linijka. Jednak to nie wszystko – moduł posiada interpreter specjalnych kodów kontrolnych (tworzonych poprzez znak '[' oraz literę), dzięki którym możemy przesuwać kursor, wyczyścić ekran, lub wyświetlić tekst wbudowanej pomocy. Więcej na ten temat będzie w dalszej części artykułu.

Na **rysunku 1** możesz zobaczyć efekt pracy układu – ekran pomocy, który prezentuje wszystkie znaki wbudowanej czcionki, informacje o autorze oraz ściągę ze spisem wszystkim dostępnych komend. Na **rysunku 2** pokazany został pomysł, jak można wykorzystać AVRTV np. do konstrukcji rejestratora temperatury. Jest to tylko statyczny obraz narysowany dla przykładu, lecz wierzę, że wielu Czytelników będzie potrafiło wykorzystać AVRTV do tworzenia wielu ciekawych projektów. Na **rysunku 3** możemy zobaczyć kolejny przykład, który nie wymaga chyba komentarza. Powyższe obrazy zostały wygenerowane na komputerze, natomiast **fotogra-**

Tabela 1

Rozdzielczość	40 kolumn x 25 wierszy znaków 8x8 pikseli
Czcionka	256 znaków, w tym podstawowy zestaw znaków ASCII, znaki specjalne (do tworzenia prostej grafiki) oraz polskie znaki w kodowaniu CP1250 (Windows)
Sterowanie	Interfejs RS232, z poziomami napięć TTL5V 2400 – 56700 bps ustawiane zworkami na płytce:

JP3	JP2	JP1	Prędkość
zwarta	zwarta	zwarta	2400
zwarta	zwarta	-	4800
zwarta	-	zwarta	9600
zwarta	-	-	14400
-	zwarta	zwarta	19200
-	zwarta	-	28800
-	-	zwarta	38400
-	-	-	57600

Wyjście video	Composite 1Vpp/75Ohm
Wymiary modułu	57x40mm
Zasilanie	5V / ~ 35mA

fie 1 i **2** przedstawiają efekt współpracy modułu z prawdziwym telewizorem. Jak widać, pomiędzy znakami w poziomie są odstępy o szerokości jednego piksela. Dlaczego tak jest? To zostanie wyjaśnione poniżej. W celu zachowania proporcji, w pionie także dodano puste linie pomiędzy wierszami, dzięki czemu czarna ramka wokół całego obrazu ma ze wszystkich stron podobną szerokość.

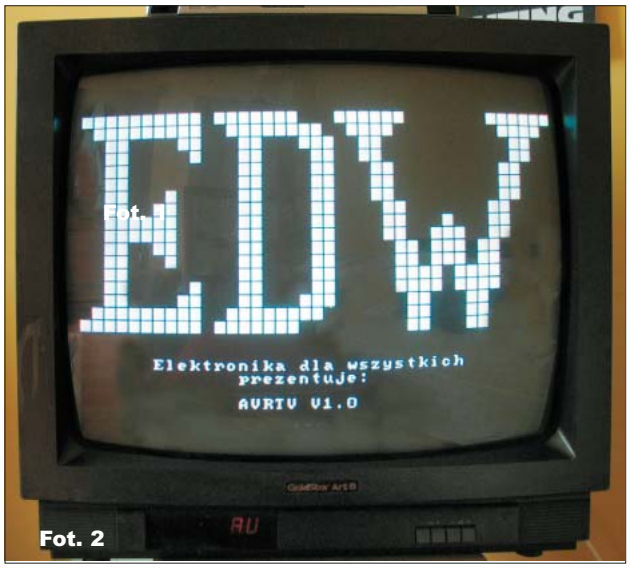
Opis układu

Rysunek 4 przedstawia schemat ideowy. Jak widać, układ zrealizowano na mikrokontrolerze AVR ATMEGA8. Procesorowi towarzyszy standardowo kwarc 16MHz (maksymalna częstotliwość dla MEGA8) oraz złącze programowania w systemie (ISP) w popularnym, 10-pinowym standardzie Atmela STK200. Rezystor R7 profilaktycznie „podciąga' pin resetu procesora do logicznej jedynki, aby uniknąć zakłóceń, gdy programator nie jest podłączony do układu. Zasilanie +5V pochodzi ze złącza CON1 — powinno być ono stabilizowane (układ nie będzie poprawnie pracował z zasilaniem 3,3V). Napięcie jest filtrowane przez kondensatory C1 i C2. Linie RXD oraz TXD interfejsu RS232 procesora są wyprowadzone także na złącze CON1. Możemy je połączyć bezpośrednio np. z innym procesorem AVR lub poprzez konwerter napięć z portem COM komputera. Zworki podłączone do PORTC pozwalają na wybór prędkości, z jaką ma się odbywać komunikacja RS232 (konfiguracje zworek i odpowiadających im prędkości są spisane w tabeli 1). Ponieważ tylko trzy z pięciu zworek są wykorzystane, pozostałe dwie użytkownik może spożytkować w dowolny sposób, jeśli zdecyduje się na modyfikację oprogramowania.

To właściwie wszystko na temat typowej części układu. Szczegółowego wyjaśnienia natomiast wymaga układ wyjściowy wideo. Przed dalszym opisem wypada przekazać kilka słów na temat standardu przesyłania obrazu w systemie PAL. Pełniejszy opis znajduje się w poprzednim artykule „Snake" z EdW 12/06. Skrótowo: obraz składa się z 312 linii (w praktyce 2 x 312, dla klatki parzystej i nieparzystej, ale nas w tym rozwiązaniu to nie interesuje). Jedna klatka obrazu trwa 20ms, a jedna linia – 64µs. Każda linia rozpoczyna się impulsem synchronizacji trwającym 4-5µs, podobnie jak każda klatka, lecz tutaj długość impulsu musi wynosić 128µs. Część widoczna obrazu zaczyna się ok. w ok. 12µs linii, a kończy w ok. 62µs. Poziom napięcia sygnału w tym czasie oddaje jasność danego punktu. Wartość napięcia dla impulsów synchronizacji (zarówno poziomej, jak i pionowej) wynosi 0V, a skali szarości odpowiadają napięcia od 0,3V (czarny) do 1V (biały). Pomijam tutaj kwestie kodowania koloru (niemożliwa do zrealizowania w naszym układzie) i inne niuanse.

Generowanie grafiki sprowadza się do wytworzenia impulsów synchronizacyjnych i wysyłania w kolejnych liniach odpowiedniej liczby pikseli tworzących obraz. Na **rysunku 5** można zobaczyć fragment jednej linii oraz towarzyszące jej przebiegi w układzie. W tym momencie można już zdradzić, na czym polega niezwykłość rozwiązania – pin PB3 (MOSI) nieprzypadkowo jest połączony nie tylko do złącza programowania. Jest to wyjście danych interfejsu SPI, który poza wykorzystaniem do programowania układu, pozwala na synchroniczne (w takt sygnału zegarowego) przesyłanie bajtów z bardzo dużą prędkością (do 8MHz, czyli 8 megabitów na sekundę). Używając go, możemy generować serie po 8 impulsów (bitów) o dowolnej wartości. Jeśli za pomocą SPI wyślemy bajt 10101010, otrzymamy na wyjściu MOSI serie naprzemiennych zer i jedynek, które, jeśli zostaną skonwertowane do poziomu napięć sygnału wideo (0,3V – 1V), utworzą nam na ekranie 8 pixeli, 4 zaświecone i 4 zgaszone. Dzięki temu, że jeden bit przy częstotliwości 8MHz trwa 125ns, możemy w jednej mikrosekundzie zmieścić szerokość jednego znaku, a w całej linii obrazu – do 40 i więcej. Przy generowaniu pikseli programowo byłoby to całkowicie niewykonywalne, ponieważ 125ns to czas trwania zaledwie dwóch instrukcji w asemblerze. Gdy interfejs sprzętowo generuje nam 8 kolejnych punktów, procesor ma czas na przygotowanie bajtu dla kolejnego znaku. Dokładniej zostanie to opisane w dalszej części artykułu. Wracając do rysunku, pin PB1 wykorzystany jest do wymuszania 0V na wyjściu, co można zobaczyć po lewej stronie. Dalej widać fragment obrazu składa-

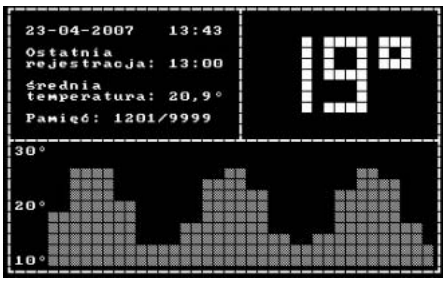
Fot. 1



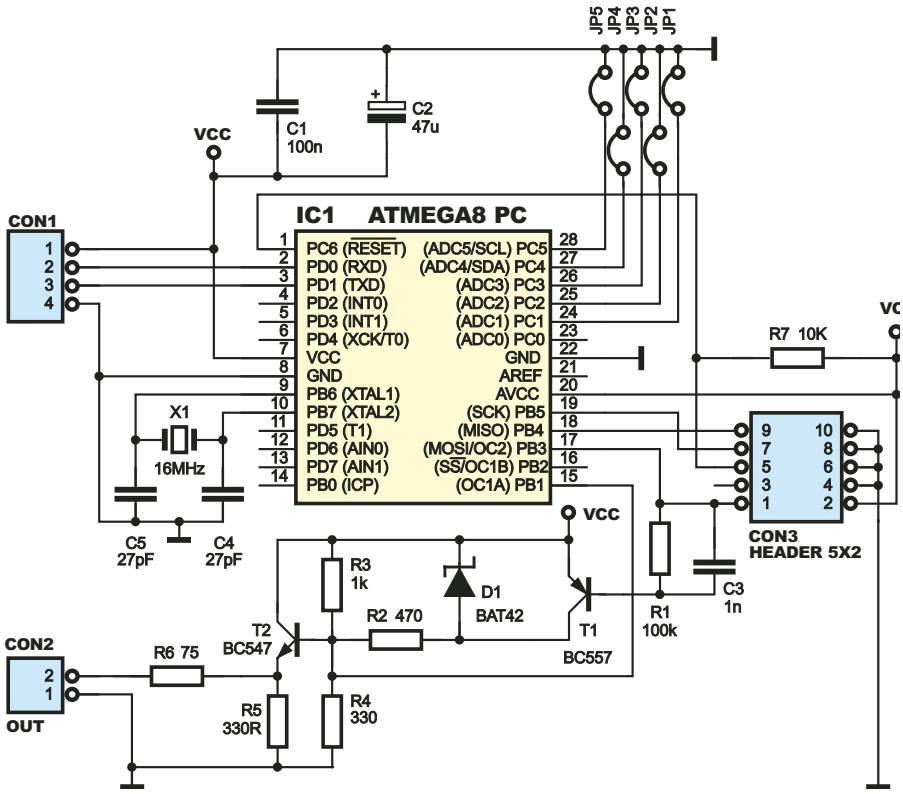
Fot. 2

Rys. 1

Rys. 2

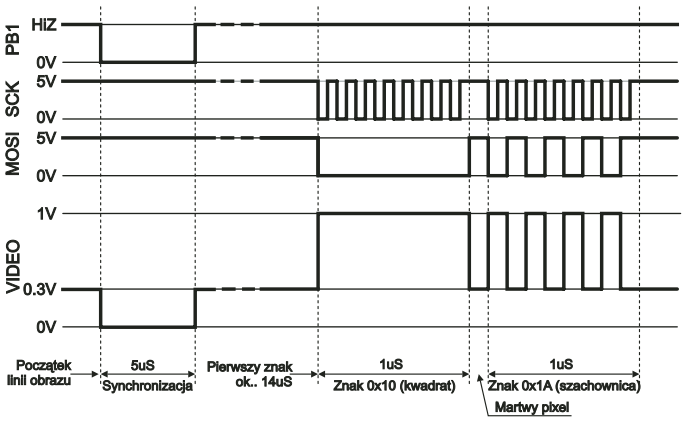


Rys. 3



Rys. 4 Schemat ideowy

Rys. 5 Przebiegi



jący się z dwóch znaków – kwadracika oraz szachownicy. Niestety, mimo swoich zalet w tym rozwiązaniu, SPI ma dwie wady: gdy nie są przesyłane żadne dane, wyjście domyślnie przechodzi w stan wysoki (odpowiednik koloru białego), co zmusza nas do zanegowania tego sygnału oraz danych, które wysyłamy. Kolejny problem: bajtów nie można wysyłać ciurkiem – pomiędzy kolejnymi transmisjami musi być odstęp o szerokości jednego bitu. Druga wada powoduje, że nie da się w ten sposób wytworzyć ciągłego potoku pixeli, a więc pełnej grafiki rastrowej. Znaki zostają rozstrzelone, na co zwróciłem już uwagę we wstępie. Wyjście zegarowe SCK interfejsu SPI nie jest w żaden sposób wykorzystywane do tworzenia obrazu – zostało pokazane dla dodania przejrzystości.

Mając już sygnały sterujące wyprowadzone z procesora, musimy je skonwertować do poziomów napięć wideo. Zajmują się tym tranzystory T1 i T2 oraz towarzyszące im elementy. Tranzystor NPN T2 tworzy wyjściowy wtórnik napięciowy, który zapewnia dużą wydajność prądową niezależną od źródła sygnału. Rezystor R6 o wartości 75Ω zapewnia dopasowanie impedancyjne z kablem wideo oraz odbiornikiem. Gdy obraz jest czarny i nie jest wymuszana synchronizacja, rezystory R3 i R4 (tworzące dzielnik) podają na bazę T2 napięcie ok. 1,24V. Po odjęciu od tego 0,65V (napięcie baza-emiter w tranzystorze bipolarnym) uzyskamy na rezystorze R6 ok. 0,6V. Rezystor ten, wraz z rezystorem na końcu linii (w telewizorze), tworzą dzielnik 1: 2, a więc ostatecznie uzyskujemy pożądaną wartość 0,3V. Jak już wspomniałem, sygnał z pinu MOSI musi być zanegowany, co uzyskujemy za pomocą tranzystora PNP T1. Gdy na MOSI panuje stan wysoki, tranzystor jest zatkany, gdy pojawi się stan niski – tranzystor otwiera się, dołączając do R3 rezystor R2. Wartość napięcia podawanego na bazę wtórnika T2 zmienia się na ok. 2,5V, co daje nam w rezultacie na wyjściu ok. 0,9V, czyli kolor biały. Pin PB1, generujący synchronizacje, normalnie jest w stanie wysokiej impedancji, dlatego dołączenie go do bazy T2 nic nie zmienia. Gdy na pinie pojawi się stan niski, wymusza zatkanie tranzystora, a więc 0V na wyjściu. Dodatkowego wyjaśnienia wymaga obecność elementów C3 oraz D1. Jak się okazało, tranzystor T1 nie potrafił dostatecznie szybko się przełączać, co skutkowało zlewaniem się pikseli, a nawet całych znaków. Kondensator C3 zapewnia dostarczenie do bazy tranzystora dużego impulsu prądowego podczas zmiany stanu na MOSI z 0 na 1 i odwrotnie. Dzięki temu czas przełączania spadł do ok. 30ns. Niestety obecność kondensatora powoduje powstanie przepięć (szpilek o dużym napięciu) na wyjściu tranzystora T1 – częściowo zapobiega temu dioda D1, lecz nie jest ona niezbędna do poprawnej pracy układu – może jedynie odrobinę poprawić jakość obrazu.

Komendy sterujące

Moduł jest swego rodzaju terminalem RS232, który wyświetla odebrane znaki na kolejnych pozycjach ekranu. Będziemy się tutaj posługiwać pojęciem kursora, lecz nie jest on w żaden sposób widoczny na ekranie – procesor pamięta aktualne współrzędne (nr wiersza i kolumny), do której musi zostać zapisany kolejny znak. Po dojściu do końca wiersza, kursor przechodzi do następnego, a gdy znajdzie się w dolnym prawym rogu ekranu, wszystkie wiersze zostają przesunięte w górę, tworząc nową pustą linię, na której początek przechodzi kursor.

Komendy sterujące składają się ze znaku **[** oraz dużej litery i opcjonalnie dwóch bajtów.

[N
NEW LINE
Przenosi kursor na początek kolejnej linii (odpowiednik znaków CR/LF). Jeśli zostanie użyte w ostatniej linii, obraz przesuwa się w górę, tworząc pusty wiersz.

[B
BACKSPACE
Cofa kursor, usuwając znak znajdujący się po lewej stronie aktualnej pozycji.

[A
CLEAR CURRENT LINE
Usuwa całą zawartość wiersza, w którym znajduje się kursor (pozycja kursora nie zostaje zmieniona)

[C
CLEAR SCREEN
Kasuje całą zawartość ekranu. Jest to procedura czasochłonna (do kilku milisekund), w której procesor nie może odebrać kolejnych znaków, dlatego **po wysłaniu komendy należy**

poczekać, aż moduł odeśle znak [, komunikując w ten sposób gotowość. Jeśli chcemy używać tylko jednostronnej komunikacji, program sterujący musi zapewnić odpowiednie opóźnienie.

[Pxy
SET POSITION
Ustawia pozycje kursora na ekranie, x oraz y są bajtami, których wartości reprezentują: numer kolumny (0-39) oraz numer wiersza (0-24). Przykładowo bajty o wartościach 10 i 1 przenoszą kursor do 11 kolumny i 2. wiersza.

[H
HOME
Przenosi kursor do lewego górnego rogu (pozycja 0,0)

[U
UP
Przesuwa kursor o jeden znak w górę. Po dojściu do brzegu ekranu komenda nie daje żadnego efektu.

[D
DOWN
Przesuwa kursor o jeden znak w dół. Po dojściu do brzegu ekranu komenda nie daje żadnego efektu.

[L
LEFT
Przesuwa kursor o jeden znak w lewo. Po dojściu do brzegu ekranu kursor przechodzi na koniec poprzedniego wiersza.

[R
RIGHT
Przesuwa kursor o jeden znak w prawo. Po dojściu do brzegu ekranu kursor przechodzi na początek kolejnego wiersza.

[T
TERMINAL MODE
Domyślny tryb wprowadzania tekstu. Po zapisaniu znaku na aktualnej pozycji, kursor przesuwa się w prawo.

[I
INSERT MODE
Tryb wstawiania. Odebrane znaki zapisywane są na aktualnej pozycji, lecz kursor pozostaje w miejscu. Do poruszania się można użyć komend [L [R [U [D.

[S
SCREENSHOT
Odsyła całą zawartość bufora ekranu (1000 bajtów z kodami znaków).

[[
Pozwala na wpisanie znaku [na ekranie.

[?
HELP
Wyświetla ekran pomocy, informacje o autorze oraz tablice znaków czcionki.

Jak widać, wbudowane komendy pozwalają na dowolne manipulowanie zawartością ekranu, a kilka z nich, mimo iż nie są niezbędne, znacznie ułatwią pracę. Format zapisu komendy [P został stworzony pod kątem sterowania przez inny mikroprocesor – wysłanie dwóch bajtów zawierających współrzędne, jest prostsze niż dziesiętny zapis liczb za pomocą kodów ASCII.

Przykład użycia komend (zakładamy, że kursor znajduje się w pozycji 0,0):
Za pomocą ciągu znaków:

```
PIERWSZA[NDRUGAA[B[NTRZECIA[A[N[R[R[RCZWARTA[P(6)(1):)[Hp
```

otrzymamy na ekranie:

```
PIERWSZA
DRUGA:)

  CZWARTA
```

Kod [B spowodował usunięcie podwójnej litery A, kod [A usunął zawartość trzeciej linii, kody [R[R[R przesunęły kursor w czwartej linii o trzy pozycje w prawo, kod [P ustawił kursor na końcu słowa w drugiej linii, a kod [H, przeniósł go do pozycji 0,0.

Tryb wstawiania jest przydatny, jeśli za pomocą znaków specjalnych (o czym za chwilę), chcemy narysować np. pionową linię:

```
[HLINIA[N[I (1) [D (1) [D (1) [D (1) [D[TKONIEC LINII
```

W rezultacie otrzymujemy:

```
LINIA
|
|
|
|
KONIEC LINII
```

Kod [H przeniósł kursor do pozycji (0,0). Po wpisaniu wyrazu LINIA i przejściu do kolejnego wiersza włączony został tryb wstawiania poprzez [I. Zapis (1) oznacza znak o numerze 1, który, jak widać na rysunku 1, jest pionową linią. Po jego wstawieniu wysyłana jest komenda [D, która przesuwa kursor o jedną pozycję w dół. Dzięki temu rozwiązaniu, nie musimy przed każdym elementem linii ustawiać pozycji kodem [P. Po zakończeniu wracamy do trybu terminalowego komenda [T i możemy wpisywać z powrotem normalne ciągi znaków.

Czcionka wykorzystywana przez AVRTV została zapożyczona z kolekcji czcionek konsolowych, które można znaleźć w praktycznie każdej dystrybucji Linuxa. Pierwotnie miała być użyta bez większych zmian, jednak zdecydowałem się na jej gruntowny lifting. Po pierwsze, polskie znaki diakrytyczne były bardzo niekształtne i wymagały narysowania od nowa. Po drugie, układ znaków był w Linuxowym standardzie ISO8859-2, co mogłoby przysporzyć problemów zdecydowanie przeważającej grupie osób piszących programy w systemem Windows. Należy pamiętać, że jeśli przykładowo piszemy program w C, do procesora AVR, który będzie wysyłał dane do AVRTV, wszystkie ciągi znaków zawierające polskie litery muszą być poprawnie kodowane. Edytory pracujące w Windowsie domyślnie wprowadzają znaki z kodowaniem CP1250 i tak je zapisują do pliku, a następnie kompilator umieszcza je w kodzie wynikowym. Dlatego zdecydowałem się zmodyfikować układ liter w czcionce, tak by odpowiadał powyższemu standardowi. Kolejną modyfikacją było dorysowanie całego kompletu znaków graficznych, które zajmują wszystkie pozycje od 0 do 31. Umożliwiają one tworzenie ramek, takich jak na rysunku 2, czy też np. pasków postępu (kwadraty i prostokąty). Wielu Czytelników zapewne zastanawia się od dłuższej chwili, dlaczego kody 10 i 13 (LF i CR używane do przejścia do nowej linii w większości terminali i systemów) są znaczkami graficznymi, a za „enter" służy komenda [N. Były ku temu dwa powody: duża dowolność i brak standardu w stosowaniu kombinacji CR i LF w transmisjach szeregowych, chęć nie zostawiania „dziury" w zestawie znaków graficznych oraz standaryzacja z pozostałymi kodami sterującymi. Wśród znaków powyżej 127. pozycji oprócz polskich liter znalazło się kilka symboli (takich jak stopień) przewidzianych przez standard CP1250. Zawiera on także dużo znaków z innych języków środkowoeuropejskich, jednak ja ograniczyłem się do Polski. Czcionkę można dowolnie modyfikować, choć wymaga to przekompilowania programu, o czym w następnej części artykułu.

Rysunek 1, prezentujący wbudowaną pomoc modułu, może być bardzo użyteczną ściągawką, pomagającą zlokalizować kod poszukiwanego przez nas znaku. Wystarczy sprawdzić, w którym wierszu i kolumnie on się znajduje, a następnie odczytać wartości z lewego i górnego brzegu ramki. W efekcie otrzymamy szesnastkowy kod znaku. Przykładowo: szukamy „°" – znajduje się on w 1. kolumnie i 12. wierszu, co daje wartość w systemie szesnastkowym 0xB0.

Oprogramowanie modułu

Jeśli nie interesują Cię, Czytelniku, szczegóły tego, jak moduł został oprogramowany, a jedynie jego walory użytkowe, możesz spokojnie pominąć ten podrozdział, jednak mimo wszystko zachęcam do lektury. Program został napisany w języku C oraz asem-

blerze przy użyciu środowiska AvrStudio 4.12 oraz kompilatora WINAVR. Całe źródło oraz skompilowane pliki wynikowe można ściągnąć z Elportalu. Głównym elementem programu jest plik *avrtv. c*, zawierający procedury inicjalizujące oraz główną pętlę odbierającą i interpretującą znaki z RS232. Cała procedura wyświetlania obrazu telewizyjnego jest zawarta w pliku *chargen. s* oraz pliku pomocniczym *singlechar. s*. Pliki *font. h* i *help. h* zawierają tablice z czcionką oraz tekst pomocy.

Procedura generatora wyzwalana jest przez przerwanie Timera 1, pracującego w trybie CTC. Tryb ten pozwala na wyzwolenie przerwania, gdy wartość licznika będzie równa zawartości rejestru OCR1B, a resetuje licznik, gdy osiągnie wartość z rejestru OCR1A. Licznik taktowany jest z podziałem wstępnym zegara przez 8, co daje jeden takt co 0,5µs. Do OCR1A zapisana zostaje wartość 127 i ostatecznie przerwanie wyzwalane jest co 64µs (długość linii obrazu). Pojawia się tutaj problem, o którym wspominałem w poprzednim artykule „Snake" – procesory AVR posiadają instrukcje wykonujące się w 1, 2, a nawet 3 taktach zegara. Skutkuje to tym, że jeśli przerwanie licznika trafi akurat w środek takiej długiej instrukcji, zostanie ona wykonana do końca, a dopiero wtedy procesor obsłuży przerwanie. Z tego powodu odstępy między przerwaniami (a więc i odstępy między impulsami synchronizacji) wahały się o +/-125ns, co było na ekranie widoczne w postaci postrzępionej krawędzi obrazu. W Snake'u problem został rozwiązany dość brutalnie – w czasie generowanie obrazu, program nie wracał do głównej pętli (pozostawał cały czas w procedurze przerwania), a cały kod gry wykonywał się w czasie synchronizacji pionowej, co nie wprowadzało zakłóceń. Problem wydawał się nie do rozwiązania, jednak z pomocą przyszły tutaj… tryby oszczędzania energii procesorów AVR. Komenda asemblera sleep powoduje przejście do stanu niskiego zużycia energii i zatrzymania wykonywania programu. Z tego stanu wybudzić procesor może np. przerwanie Timera. Procedura generatora, po jej wywołaniu, ustawia rejestr OCR1B na 2 (1us) i wchodzi w tryb sleep. Po 1us przerwanie OCR1B wybudza procesor, program skacze do ślepej procedury przerwania, która natychmiast wydaje komendę powrotu reti i program może spokojnie rozpocząć generowanie synchronizacji. Dzięki temu rozwiązaniu, **zawsze** początek i koniec impulsu występuje po takim samym czasie, niezależnie od tego, czy cała procedura miała kilkutaktowe opóźnienie, czy nie. Identyczny mechanizm wyzwala początek obrazu w każdej linii, przez co jest on zawsze w tym samym miejscu, niezależnie od tego, jakie operacje były wykonywane przed nim (m.in. zwiększanie numeru linii w zmiennych globalnych, czy też sprawdzenie, czy nie należy wygenerować synchronizacji pionowej). Odległość obrazu od lewej krawędzi ustala wartość zapisywana do OCR1B – domyślnie 14ms.

W procedurze zdeklarowane są trzy zmienne globalne: Buffer, Line, oraz Offset. Pierwsza z nich, jak nazwa wskazuje, jest buforem znaków o rozmiarze 1000 bajtów. Druga, typu unsigned int zawiera numer aktualnie wyświetlanej linii, a trzecia – określa „zawinięcie" pamięci znaków. Trzecia zmienna jest bardzo istotna – pozwala określić, który wiersz z bufora będzie widoczny na ekranie jako pierwszy. Gdy komendą [N tworzymy nowy pusty wiersz, na samym dole ekranu, musielibyśmy przepisać wszystkie wiersze o jeden w górę, zastępując ich poprzedniki. Byłoby to ogromnie czasochłonne. Jednak dzięki offsetowi wystarczy przesunąć cały obraz, tak aby pierwszy wiersz od góry stał się ostatnim, a następnie usunąć tylko jego zawartość (co wykonuje procedura w głównym pliku programu).

Pojedynczy znak generowany jest procedurą z pomocniczego pliku *singlechar. s*. Odczytuje ona kolejny kod znaku z bufora, nakłada go na adres czcionki w pamięci programu (domyślnie 0x1800, ostatnie 2KB flasha), a następnie dodaje numer wiersza w znaku (od 0 do 7). W efekcie otrzymujemy adres bajtu, który jest odczytywany z czcionki i wysyłany przez SPI. Niestety zabrakło w procedurze czasu na wykonanie pętli, więc plik *singlechar. s* jest po prostu 40 razy wstawiany poprzez dyrektywę #include do pliku *chargen. s.* Nie będę się zagłębiał tutaj w szczegóły – zainteresowani mogą sami przeanalizować kod. Po wyświetlaniu 40 znaków procedura kończy swoje działanie i wraca do głównego programu. W każdej linii mamy ok. 3–4µs czasu wolnego, co w zupełności wystarcza na obsłużenie przychodzących znaków przez RS232.

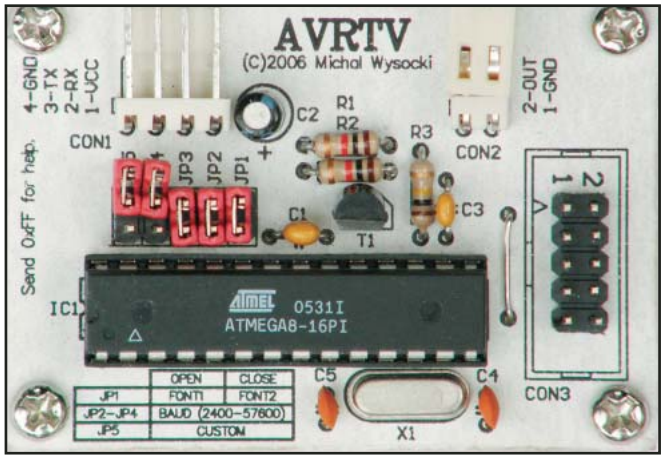
Główny plik programu zawiera podstawowe inicjalizacje procesora (konfiguracja portów, odczyt ze zworek i ustawienie prędkości UART-a, konfiguracja timera oraz deklaracja zmiennych globalnych wspólnych z *chargen. s*), proste makra GET_CHAR, PUT_CHAR, służące do odbierania i wysyłania znaków przez RS232, oraz główną pętlę interpretującą komendy. **Jest ona całkowicie niezależna od generatora obrazu. Dzięki temu możemy ją usunąć i wykonywać dowolne operacje** (np. komunikację przez I^2C z jakimś urządzeniem, wykorzystując nieużywane zworki JP4 i JP5 itp.). Musimy jednak pamiętać o kilku ograniczeniach. Głównym jest czas – jak już wspomniałem – główny program jest wykonywany tylko przez 3µs na każde 64µs, więc nie możemy wykonywać czynności krytycznych czasowo (takich jak np. komunikacja 1Wire**).** Dodatkowo mamy ok. 120µs wolnego czasu podczas synchronizacji pionowej (można to wykryć sprawdzając wartość zmiennej Line – synchronizacja rozpoczyna się, gdy zmienna przybierze wartość 0). Trzecim ograniczeniem jest rozmiar dostępnej pamięci RAM. Bufor znaków zajmuje 1000 bajtów, zostają 24 bajty, z czego znowu ponad połowę trzeba zarezerwować na stos i zmienne globalne. Zostaje ok. 7 bajtów, które możemy wykorzystać na zmienne (deklarowane na samym początku main ()). Bezwzględnie należy unikać tworzenia dodatkowych procedur – powoduje to odkładanie kolejnych dwójek bajtów na stos i prawie na pewno skończy się katastrofą dla naszego programu. Buforem znaków operujemy tak jak zwykłą tablicą, przykładowo: Buffer[82] = 'A' spowoduje zapisanie litery A do drugiego wiersza i drugiej kolumny ekranu. Numer komórki w tablicy można łatwo wyliczyć ze wzoru (y * 40) + x, gdzie x i y to numery kolumny i wiersza. Pisząc własny program należy także wystrzegać się dzielenia arytmetycznego oraz mnożenia/dodawania liczb typu long int (32 bity). Operacje te są wykonywane przez dodatkowe procedury dołączane przez kompilator do programu, a ich wywołanie może spowodować wspomniane już problemy z przepełnieniem stosu.

Plik font. h zawiera tablicę 2048 bajtów z zapisaną czcionką. Tablica deklaruje własną sekcję pamięci, której początek jest ustawiony na adres 0x1800 w opcjach kompilatora. Dzięki temu czcionka zawsze zajmuje końcowy obszar pamięci flash, co upraszcza nieco procedurę generowania obrazu. Jak już było wcześniej wspomniane, czcionkę można w łatwy sposób modyfikować do własnych potrzeb. Specjalnie do tego celu został utworzony program *fontconv,* który można pobrać wraz z resztą materiałów z Elportalu. Poza programem w katalogu umieszczone są pliki *cp1250. txt*, *help. bin*, *example1. bin* oraz *example2. bin*. Pierwszy plik zawiera naszą czcionkę w postaci łatwej do edycji. Wystarczy otworzyć go np. w Notatniku, a zobaczymy, w jaki sposób zapisywane są znaki (przykład dla liter A i B):

```
->65: 41
—XX—
-XXXX—-
XX—XX—
XX—XX—
XXXXXX—
XX—XX—
XX—XX—
————

->66: 42
XXXXXX—
-XX—XX-
-XX—XX-
-XXXXX—
-XX—XX-
-XX—XX-
XXXXXX—
————
```

W ten sposób zapisanych znaków jest w pliku 256 i nie można zmieniać tej wartości. Znak – oznacza piksel zgaszony, a X zaświecony. Używając dowolnego edytora tekstowego, możemy modyfikować istniejące znaki, dorysowywać własne lub zmieniać kolejność przez proste kopiuj-wklej. Nad każdym znakiem znajduje się jego numer zapisany dziesiętnie oraz szesnastkowo. Powinniśmy zwrócić uwagę, czy po edycji nie ma podwójnych odstępów między znakami lub ich całkowity brak



oraz czy wszystkie wiersze maja szerokość równą 8. Po przygotowaniu nowej czcionki zapisujemy ją pod inną nazwą, a następnie uruchamiamy linię poleceń systemu Windows i przechodzimy do katalogu z programem *fontconv*. Wydanie komendy:

```
fontconv nazwa_pliku. txt
```

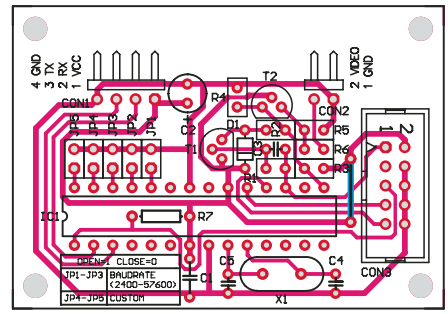
spowoduje wygenerowanie w tym samym katalogu pliku *output. h* z naszą czcionką skonwertowaną do tablicy. Teraz wystarczy zastąpić oryginalny plik *font. h* w źródle oprogramowania wygenerowanym plikiem, otworzyć projekt AvrStudio i skompilować (klawisz F7). W rezultacie otrzymamy plik do zaprogramowania procesora, zawierający naszą modyfikację. Program fontconv ma jeszcze jedną funkcję. Jeśli podamy jako drugi parametr plik *. bin zawierający 1000-bajtową zawartość ekranu (którą np. odczytaliśmy z AVRTV komendą [S]):

```
> fontconv cp1250.fnt help.bin
```

program wygeneruje oprócz czcionki obrazek o nazwie *output. bmp*. Zapewne domyślasz się już, Czytelniku, jak powstały obrazy z **rysunków 1-3** – wygenerował je właśnie *fontconv* na podstawie oryginalnej czcionki i plików *help. bin*, *example1. bin* oraz *example2. bin*. Funkcja ta może być przydatna do sprawdzenia, jak wyglądają nasze modyfikacje, bez potrzeby kompilowania programu i programowania procesora. Osoby, które zmodyfikowały czcionki (np. stworzyły układ znaków dla innych języków lub uzupełniły obecny) prosiłbym o nadsyłanie ich na mój adres e-mail, podany na końcu artykułu. Zostaną dołączone do obecnych materiałów i możliwe, że oszczędzą innym pracy.

Montaż i uruchomienie

AVRTV został zbudowany na jednostronnej płytce drukowanej, której wzór można zobaczyć na **rysunku 6**. Prototyp przeszedł kilka modyfikacji, dlatego płytka na fotografii tytułowej różni się nieco od ostatecznego wzoru. Montaż jest prosty i rutynowy, zaczynamy od wlutowania zworek, następnie lutujemy pozostałe elementy. Wszystkie rezystory należy zamontować pionowo, poza R7, który znajduje się pod podstawką mikrokontrolera. Kondensator C2 może mieć dowolną wartość od 10 do 100µF. Jako złączki CON1 i CON2 możemy zastosować zwykłe kątowe listy goldpin lub białe złącza kątowe, używane m.in. przy wentylatorach komputerowych (patrz: fotografia tytułowa). Niektórych może dziwić, że na płytce nie ma miejsca na złącze chinch, jako wyjście sygnału. Jest to posunięcie celowe – po pierwsze, wybór gniazd jest bardzo duży i z reguły jedne nie pasują na miejsce drugich, po drugie – AVRTV jest modułem przeznaczonym do wbudowania w większe urządzenie, dlatego wygodniejsze z punktu widzenia konstruktora jest przykręcenie do obudowy gniazda w dowolnym miejscu, a następnie połączenie go z płytką ekranowanym kabelkiem. Bardzo dobrze w tej roli sprawdza się pojedynczy przewód audio, jednak nie powinien być zbyt długi. Jeszcze raz muszę przypomnieć, że sygnały RXD i TXD są tutaj w standardzie TTL, dlatego próba podłączenia ich bezpośrednio do portu COM komputera skończy się prawdopodobnie uszkodzeniem mikrokontrolera. Po zaprogramowaniu przy użyciu typowego programatora zgodnego z STK200 i ustawieniu zworkami wymaganej prędkości, układ powinien od razu pracować.



Rys. 6

Zachęcam Czytelników do informowania o pomysłach i realizacjach wykorzystujących AVRTV. Jeśli układ zdobędzie popularność, zostanie prawdopodobnie utworzona osobna strona zawierająca materiały i przykłady projektów. Uprzedzając pytanie, jakie może się pojawić od początkujących Czytelników – układ nie potrafi działać jako tzw. OSD, czyli nakładać tekstu na istniejący sygnał wideo np. z kamery. Umożliwiająca to modyfikacja została tymczasowo pozostawiona w fazie testów – jeśli będzie odpowiednio duże zainteresowanie, wznowię prace nad nią i możliwe, że zostanie opublikowana.

Michał Wysocki
mwsoft@o2.pl

Wykaz elementów

R1	100kΩ	D1	BAT42
R2	470Ω	T1	BC557
R3	1kΩ	T2	BC547
R4,R5	330Ω	IC1	ATMEGA8 PC
R6	75Ω	CON1,CON2	listwy goldpin, kątowe
R7	10kΩ	CON3	jumper 5x2
C1	100nF	JP1-JP5	jumper
C2	47µF	X1	16MHz
C3	1nF		
C4,C5	27pF		

Komplet podzespołów z płytką jest dostępny w sieci handlowej AVT jako kit szkolny AVT-2853.

R E K L A M A