Co potrafią ARM–y:

Wolfenstein 3D na STR911

AVT–5122

PROJEKT Z OKŁADKI

Niemal każdy ambitny elektronik–cyfrowiec zbudował lub przynajmniej chciał zbudować w swojej karierze grę komputerową działającą na jakimś mikrokontrolerze. W Internecie można znaleźć wiele opisów rozwiązań Pongów, Tetrisów czy Snake'ów. Takie gry są tworzone często w celu zademonstrowania możliwości mikrokontrolera, na którym działają.

My prezentujemy grę, która pokazuje co potrafią nowoczesne mikrokontrolery z rdzeniami ARM. Grę, która 15 lat temu dała początek gatunkowi FPS (First Person Shooter). Znów padną strzały i poleje się nazistowska krew. Oto przed Państwem: Wolfenstein 3D.

Dodatkowe materiały do artykułu publikujemy na CD-EP oraz www.ep.com.pl

PODSTAWOWE PARAMETRY

- Wymiary PCB: 127x58 mm
- zasilanie: 2x1,2 V (akumulatory AA)
- możliwość zasilania zewnętrznego
- CPU z rdzeniem z rodziny ARM9 (STR911)
- współpracuje z modułem dipARM (ZL21ARM)
- wbudowany wzmacniacz audio
- kolorowy wyświetlacz LCD z Nokii 6100 (132x132 punkty)
- wbudowana przetwornica podświetlacza
- wbudowany interfejs komunikacyjny RS232

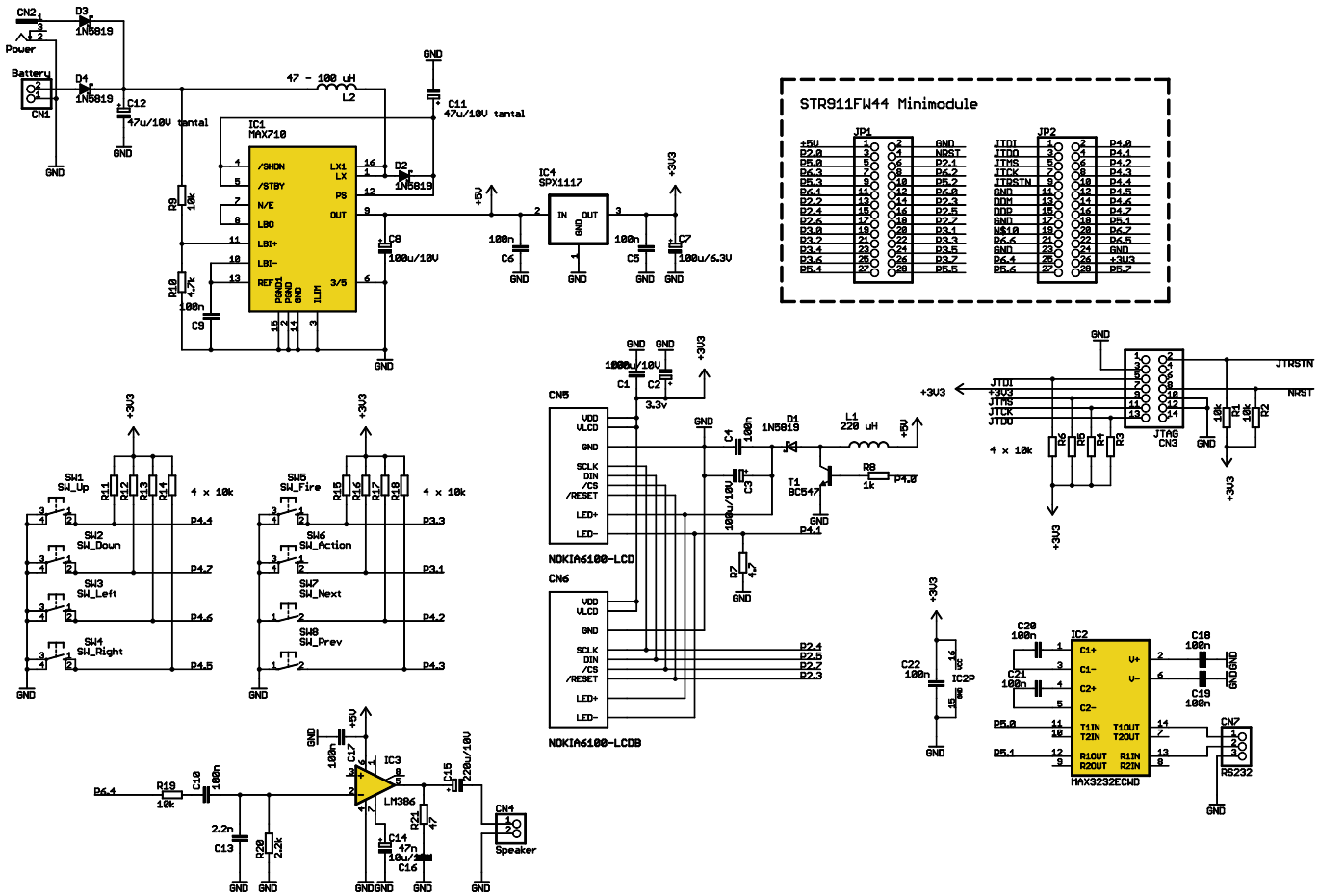
Trochę historii

Pierwsze gry z namiastką trójwymiarowości pojawiły się około roku 1973, ale nie zdobyły popularności – w tamtych czasach komputer był rzadkością. W latach 80. powstało kilka nowych tytułów, głównie na automaty i domowe komputery 8–bitowe (ZX Spectrum, Atari, itp.). Ze względu na ich mizerną moc obliczeniową, gry te nie zachwycały jakością grafiki.

Przełom nastąpił na początku lat 90. wraz z upowszechnieniem PC–tów. W 1991 roku nikomu jeszcze wówczas nie znana firma Id Software opublikowała grę Hovertank 3D, w której zadaniem gracza było eksplorowanie podziemnych labiryntów obserwowanych oczami bohatera i likwidowanie napotkanych po drodze przeciwników. Kilka miesięcy później pojawiła się gra Catacomb 3D, bazująca na zmodyfikowanym silniku graficznym Hovertank, umożliwiającym nakładanie tekstur na ściany labiryntów. Wprowadzono też widok dłoni bohatera trzymającej broń, co pogłębiało wrażenie obserwowania świata gry z perspektywy pierwszej osoby.

W roku 1992 ukazał się Wolfenstein 3D. Gra została wydana na zasadach *shareware* – wersja z mniejszą liczbą poziomów była dostępna za darmo. Dzięki takiemu sposobowi dystrybucji oraz niewielkim wymaganiom sprzętowym Wolf3D odniósł spektakularny sukces i do dziś jest uważany za prekursora gatunku *First Person Shooter* (FPS).

Po Wolfensteinie firma Id Software tworzyła kolejne, coraz bardziej zaawansowane technologicznie gry, z których większość stała się hitami – chyba każdy słyszał o Doomie, Hereticu czy Quake'u (jeden z pierwszych FPS–ów z prawdziwą grafiką trójwymiarową).



Rys. 1. Schemat elektryczny płyty bazowej konsoli

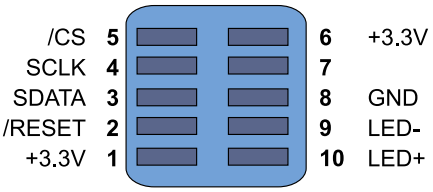
Sprzęt

Przy projektowaniu „konsoli", na której będzie działał Wolf3D, przyjąłem następujące założenia:

- forma – kieszonkowa konsola do gier sterowana za pomocą kilku przycisków,
- urządzenie ma być w miarę proste i tanie w wykonaniu (najbardziej kosztowne elementy mogłyby być wykorzystane do innych projektów bez niszczenia konsoli),
- mikrokontroler – ARM z jak największą ilością wbudowanej pamięci, zarówno Flash, jak i RAM. Dane gry muszą się gdzieś zmieścić, a stosowanie zewnętrznych pamięci odpada ze względu na wymaganą prostotę sprzętu. Moje wymagania spełnia układ STR911FW44 firmy STMicroelectronics, wyposażony w rdzeń ARM966E–S pracujący z maksymalnym zegarem 96 MHz, wyposażony w 512 kB pamięci Flash oraz 96 kB RAM–u. Ponadto, są produkowane DIP–moduły oparte na tym mikrokontrolerze, dzięki czemu można go wykorzystać w innych projektach, gdy gra nam się znudzi. Jedynym minusem jest dość kiepska dokumentacja mikrokonrolerów STR91x (mam nadzieję, że STM szybko to poprawi),
- wyświetlacz – kolorowy, łatwy do zdobycia, tani – wymagania te wyśmienicie spełniają wyświetlacze od telefonów Nokia 6100, 6610 i kilku innych modeli,
- dźwięk – jak wyżej, czyli bez drogich lub specjalizowanych układów,
- zasilanie – 2 akumulatorki R6, albo zasilacz sieciowy.

Schemat elektryczny urządzenia pokazano na **rys. 1.** Jak już wspominałem „sercem" konsoli jest mikrokontroler STR911FW44 w formie DIP–modułu z dwoma dwurzędowymi złączami typu IDC, oznaczonymi na schemacie jako JP1 i JP2. Pracuje on z zegarem 96 MHz (maksymalna dopuszczalna częstotliwość taktowania), wytwarzanym przez wbudowany w układ powielacz częstotliwości z PLL. Mikrokontroler może być programowany w systemie przez złącze JTAG (CN3) pasujące do programatora ST FlashLink. Rezystory R1...R6 podciągają wszystkie sygnały interfejsu JTAG do napięcia zasilania.

Ekranem konsoli jest wyświetlacz od telefonu Nokia 6100 o rozdzielczości 132x132 z kontrolerem Epson S1D15G10. Ponieważ w sprzedaży są dwa warianty różniące się złączami (dwurzędowe mikrozłącze SMD z rastrem 0,5 mm lub jednorzędowe z rastrem 1,27 mm), urządzenie zostało wyposażone w obydwa typy złącz (CN5, CN6). Opis wyprowadzeń wyświetlacza pokazano na **rys. 2.** Wyświetlacz komunikuje się z mikrokontrolerem za pomocą inter-



Rys. 2. Złącze wyświetlacza z telefonu Nokia 6100

SPIS ELEMENTÓW

Rezystory

R7: 4,7 Ω przewlekany
R8: 1 kΩ SMD 0805
R10: 4,7 kΩ SMD 0805
R20:2,2 kΩ SMD 0805
pozostałe – 10 kΩ SMD 0805

Półprzewodniki

D1...D4: 1N5819
IC1: MAX710 SOIC
IC2: MAX3232C SOIC
IC3: LM386 DIP8
IC4: 1117–3.3 DPAK
T1: BC547

Kondensatory

C1, C4...C6, C9, C10, C17...C22: 100 nF SMD 0805
C2, C3, C7, C8: 100 µF/10 V przewlekany
C11, C12: 47 µF/10 V tantalowy low ESR SMD 6032
C13: 2,2 nF SMD 0805
C14: 10 µF/10 V przewlekany
C15: 220 µF/10 V przewlekany
C16: 47 nF SMD 0805

Pozostałe

CN2: miniaturowe złącze zasilacza DC
CN3 pinhead 2x7
CN5: mikrozłącze wyświetlacza LCD
CN7: pinhead 1x3
JP1, JP2: pinheady żeńskie 2x14
L1: 220 µH
L2: 47...100 µH
SW1...SW6: mikroprzełączniki (duże)
SW7, SW8" mikroprzełączniki kątowe
DIP–moduł z mikrokontrolerem STR911FW44

fejsu szeregowego zgodnego z SPI. Linie DIN i SCLK są obsługiwane przez kontroler SSP (*Synchronous Serial Port*), zaś linie /CS i /RESET są obsługiwane przez linie I/O.

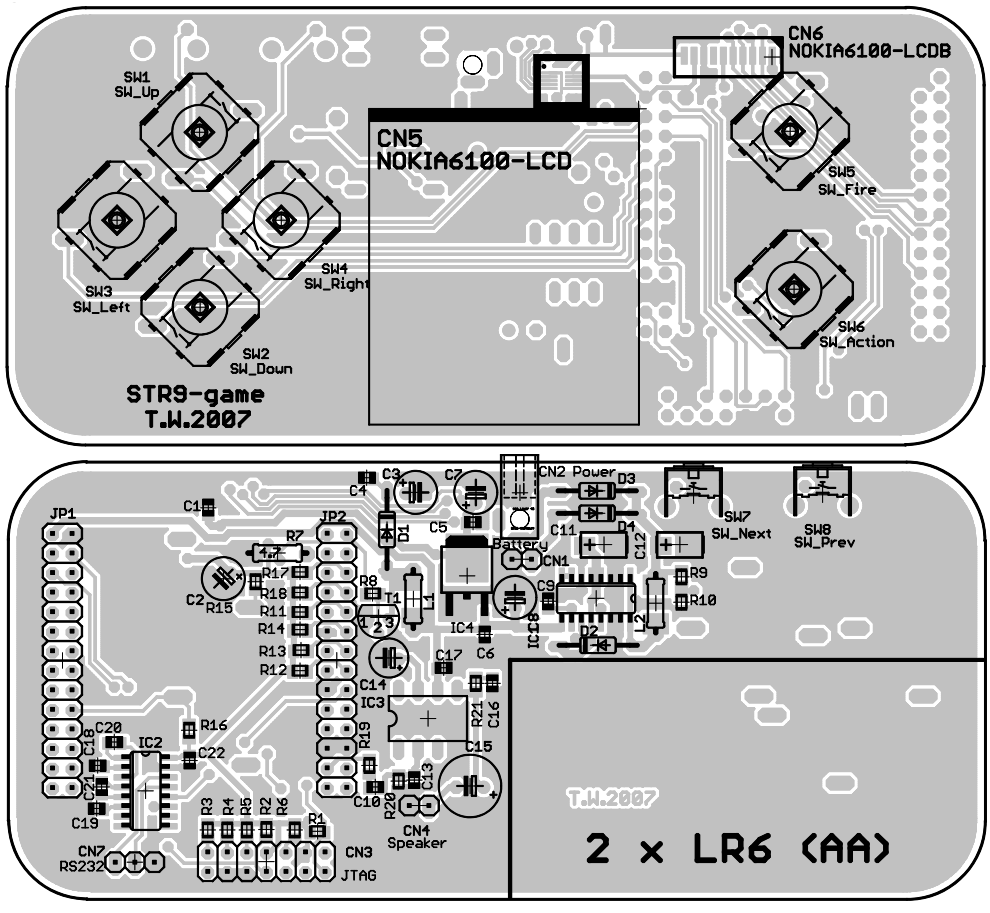
Zastosowany wyświetlacz ma wbudowane podświetlenie (dwie białe diody LED). Zasila je prosta przetwornica podwyższająca napięcie (elementy L1, T1, D1, C3, C4). Tranzystor T3 jest kluczowany przez kontroler PWM mikrokontrolera. Napięcie na rezystorze R7, proporcjonalne do prądu płynącego przez LED–y, jest doprowadzone do przetwornika A/C w mikrokontrolerze. Takie rozwiązanie umożliwia programową regulację jasności wyświetlacza – dobieramy wypełnienie przebiegu sterującego T3 tak, aby uzyskać żądany prąd podświetlenia.

Urządzenie obsługuje się za pomocą sześciu dużych mikroprzełączników oznaczonych SW1...SW6 (kursory oraz Fire i Action) oraz dwóch mikroprzełączników kątowych (SW7, SW8). Przyciski są podciągnięte do napięcia +3,3 V przez rezystory R11...R18 i podłączone bezpośrednio do pinów GPIO mikrokontrolera.

Jak na porządną konsolę do gier przystało, mamy również dźwięk, wytwarzany przez timer pracujący w trybie PWM. System dźwiękowy nie spełnia co prawda surowych audiofilskich wymagań, ale za to jest prosty i tani. Przebieg z wyjścia PWM trafia do filtru dolnoprzepustowego złożonego z elementów R19, C13, tłumiącego częstowości powyżej 5 kHz. Rezystor R20 zmniejsza amplitudę sygnału tak, aby nie przesterować wzmacniacza mocy (IC3) typu LM386, zaś kondensator C10 eliminuje składową stałą. Wyjście wzmacniacza IC3 należy podłączyć do niewielkiego głośniczka przez złącze CN4.

Na uwagę zasługuje zastosowany w konsoli układ zasilania. Dostarcza on napięć +5 V (do zasilania minimodułu z STR911 i wzmacniacza audio) oraz +3,3 V (zasilanie LCD, JTAG, RS232). Napięcie +5 V wytwarza stabilizator MAX710 (IC1) firmy Maxim, będący połączeniem przetwornicy impulsowej *step–up* ze stabilizatorem LDO. Dzięki takiemu rozwiązaniu układ ma bardzo szeroki zakres napięć wejściowych (od 1,8 V do 11 V) przy prądzie obciążenia 250 mA (przy Uwe > 3 V – 500 mA). Umożliwiło to zasilanie konsoli zarówno z dwóch „paluszków" R6, jak i z zasilacza sieciowego o napięciu do 11 V DC. Uklad MAX710 potrzebuje kilku elementów zewnętrznych, najważniejsze z nich to:

– cewka (L2) o indukcjności z zakresu 18...100 µH,



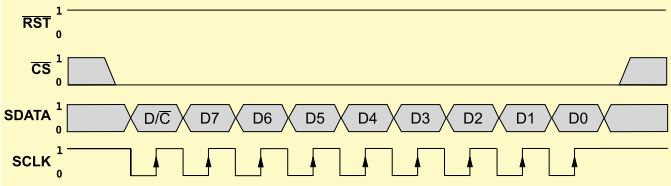
Rys. 3. Rozmieszczenie elementów na płytce konsoli

Sterowanie wyświetlaczem z telefonu Nokia 6100

Wyświetlacze te mają rozdzielczość 132x132 piksele i 12–bitowy kolor (po 4 bity na składowe R, G, B). Występują w dwóch wersjach – ze sterownikiem Philips PCF8833 lub Epson S1D15G10. Różnią się one formatem komend sterujących, interfejs szeregowy obsługuje się w obu wersjach tak samo. Poniżej znajduje się skrótowy opis obsługi modułu z kontrolerem Epsona, wykorzystanego w testowym egzemplarzu konsoli.
Wyświetlacz łączymy z mikrokontrolerem przez 4–przewodowy interfejs szeregowy zbliżony do SPI składający się z linii:

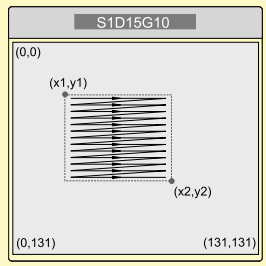
- SCLK – Serial Clock – zegar SPI,
- SDATA – Serial Data – dane SPI,
- /CS – Chip Select – stan niski zezwala na odczyt/zapis danych do wyświetlacza. Jeśli do magistrali jest podłączony tylko wyświetlacz, możemy zewrzeć ją do masy,
- /RESET – stan niski zeruje sterownik wyświetlacza.

Zalecam użycie mikrokontrolera ze sprzętowym portem SPI. Można oczywiście emulować go programowo, ale możemy wtedy zapomnieć o wyświetlaniu jakiejkolwiek bardziej skomplikowanej animacji – praktycznie cały czas procesora sterującego wyświetlaczem będzie marnowany na obsługę SPI.
Transmisję pojedynczego bajtu do wyświetlacza zilustrowano na rysunku poniżej. Ramka danych ma długość 9 bitów, z czego 8 młodszych bitów to przesyłany bajt. Od stanu najstarszego bitu zależy, czy zostanie on zinterpretowany przez wyświetlacz jako polecenie (D/C=0), czy jako dana (D/C=1).



Transmisja jednego bajtu do wyświetlacza

Inicjalizację wyświetlacza zaczynamy od wyzerowania sterownika, wymuszając przez kilkadziesiąt milisekund stan niski na linii /RESET przy SCLK=1 i SDATA=0. Ponieważ układ Epsona może pracować z różnymi matrycami LCD, niezbędne jest ustawienie przy starcie kilku „niskopoziomowych" parametrów sterownika. Przykładową sekwencję danych inicjalizujących wyświetlacz z telefonu Nokia przedstawiono poniżej (komendy zaznaczono pogrubioną czcionką, dane – zwykłą):
Display Control, Common Scan Direction: ustawiamy timing sygnałów sterujących matrycą
DISPCTL 0xca, 0x130, 0x120, 0x1c0, 0x100
COMSCN 0xbb, 0x101
Oscillator On, Sleep Out: włączamy generator przebiegów sterujących LCD i wyłączamy tryb uśpienia
OSCON 0xd1
SLPOUT 0x94
Volume Control: kolejne parametry „niskopoziomowe"
VOLCTR 0x81, 0x105, 0x101
Power Control: włączamy zasilanie matrycy LCD
PWRCTR 0x20, 0x10f
— czekamy kilkadziesiąt milisekund —
Inverse Display: odwracamy jasność pikseli (aby jasność pikseli była proporcjonalna do ich wartości, czyli 0 = piksel zgaszony, maksimum = piksel całkowicie zapalony)
DISINV 0xa7
Data Control: ustawiamy format danych obrazu: 12 bitów (2 bajty) na piksel, standardowa organizacja pamięci (lewo–prawo, góra–dół, początek pamięci odpowiada lewemu górnemu rogowi ekranu)
DATCTL 0xbc, 0x100, 0x100, 0x104, 0x100
Display On: po ustawieniu wszystkiego możemy włączyć odświeżanie wyświetlacza
DISON 0xaf
Aktualizacja zawartości ekranu polega na wybraniu prostokątnego obszaru, którego wnętrze ma być odświeżone, i przesłaniu wartości kolejnych pikseli znajdujących się wewnątrz wybranego prostokąta. Lewy górny róg obszaru oznaczmy jako (x1, y1), zaś prawy dolny jako (x2, y2). Położenie aktualizowanego fragmentu przekazujemy do wyświetlacza za pomocą następujących poleceń:
No operation: komenda przerywa wykonanie bieżącego polecenia, np. zapisu do pamięci
NOP 0x25
Page address set: ustawiamy współrzędne pierwszej i ostatniej kolumny aktualizowanego fragmentu ekranu
PASET 0x75, 0x100 | (x1+2), 0x100 | (x2+2)
Column address set: ustawiamy współrzędne pierwszego i ostatniego wiersza aktualizowanego obszaru
CASET 0x15, 0x100 | y1, 0x100 | y2
RAM write: włączamy zapis do pamięci RAM obrazu – od tej pory każda wysłana do sterownika dana zostanie zapisana do RAM-u. Tryb zapisu można opuścić wysyłając komendę NOP.
RAMWR 0x5c
Następnie przesyłamy kolory pikseli znajdujących się wewnątrz ustalonego obszaru. Sterownik S1D15G10 ma wbudowaną pamięć obrazu, dlatego nie ma potrzeby aktywnego odświeżania matrycy – wystarczy jednorazowe zapisanie danych. Kolejność transmisji (z góry do dołu, od lewej do prawej) oraz format pikseli są pokazane na poniższym rysunku.



Format pojedynczego piksela

Bajt 1

MSB							LSB
X	X	X	X	R3	R2	R1	R0

Bajt 2

MSB							LSB
G3	G2	G1	G0	B3	B2	B1	B0

Kolejność i format pikseli przy aktualizacji obrazu

Przykładowy kod w C wypełniający kolorem czerwonym prostokąt o zadanych współrzędnych przedstawiono na **list. 1.**

List. 1.

```
(....)
  spi_tx(0x25);          // NOP
  spi_tx(0x75);       // Page Address Set
  spi_tx(0x100 | (x1 + 2));
  spi_tx(0x100 | (x2 + 2));
  spi_tx(0x15);          // Column Address Set
  spi_tx(0x100 | (y1));
  spi_tx(0x100 | (y2));
  spi_tx(0x5c);          // Write to RAM

  for (y=y1; y<=y2; y++)
     for (x=x1; x<=x2; x++)
     {
        spi_tx(0x100 | 0xf); // bajt 1: R = 15
        spi_tx(0x100 | 0x0); // bajt 2: G = 0, B = 0
     }
```

Kompletny kod obsługujący wyświetlacz z wykorzystaniem kontrolera DMA dla mikrokontrolera STR911 znajduje się w pliku *epslcd.c* wchodzącym w skład archiwum ze źródłami gry. Powyższy opis nie przedstawia wszystkich możliwości sterownika S1D15G10. Szczegóły (np. praca z obniżonym poborem energii, scrollowanie obrazu, pozostałe formaty kolorów, itp.) można znaleźć w karcie katalogowej układu.

- dwa kondensatory elektrolityczne (C11, C12) przeznaczone do pracy przy dużych częstotliwościach (np. tantalowe) o niskiej rezystancji szeregowej (*low ESR*),
- 1–amperowa dioda Schottky'ego (D2).

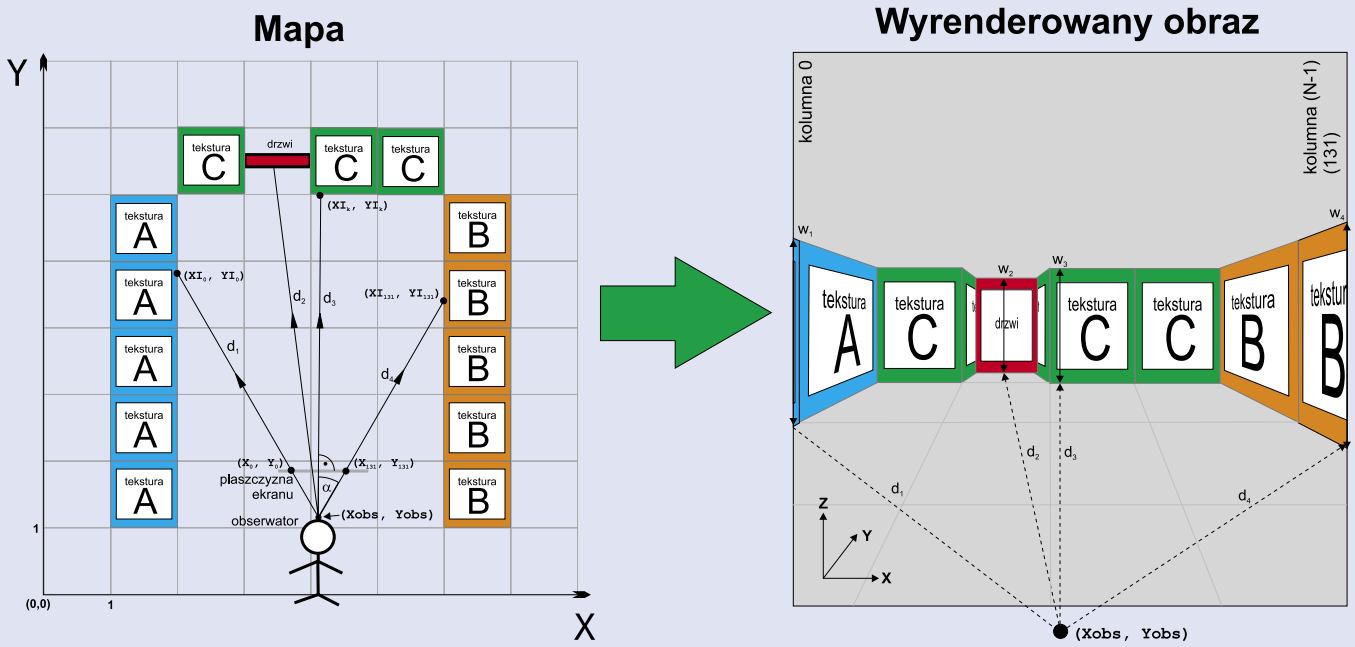
Diody D3 i D4 zabezpieczają urządzenie przed odwrotnym podłączeniem zasilania. Ponadto D3 odłącza baterie, jeśli korzystamy z zasilacza sieciowego. Napięcie +3,3 V jest wytwarzane z +5 V przez standardowy stabilizator LDO typu 1117 (IC4).

Konsolę wyposażono w interfejs RS232 (IC2, CN7). Był on pomocny przy testowaniu oprogramowania, może także posłużyć do zaprogramowania mikrokontrolera za pomocą bootloadera, jeśli nie mamy odpowiedniego interfejsu JTAG.

Montaż i uruchomienie

Układ został wykonany na dwustronnej płytce o wymiarach 128x58 mm. Kształt płytki, jak również rozmieszczenie przycisków przypomina typowy *gamepad*. Stosowałem zarówno elementy SMD jak i przewlekane. Schemat montażowy przedstawiono na **rys. 3**. Wszystkie części oprócz przycisków SW1...SW6 i złącza wyświetlacza montujemy od

Jak działa silnik graficzny Wolfensteina?



Zasada działania silnika 3D w grze Wolfenstein

Labirynt, po którym porusza się bohater gry jest zapisany w formie bitmapy o wymiarach 64x64. Wartości poszczególnych pikseli decydują czy w danym miejscu znajduje się ściana, drzwi, obiekt (np. nieprzyjaciel) lub pusta przestrzeń.
Silnik graficzny Wolfensteina wykorzystuje do rysowania ścian technikę nazywaną *raycastingiem.* Ideę *raycastingu* pokazano na powyższym rysunku.

Załóżmy że obserwator stoi w punkcie (Xobs, Yobs). Tuż przed nim znajduje się odcinek prostopadły do kierunku patrzenia będący płaszczyzną ekranu. Rozmieszczamy na nim w równych odległościach N punktów, gdzie N jest szerokością (w pikselach) renderowanego obrazu, w naszym przypadku N=132. Punkty te oznaczmy jako (Xk, Yk). Obraz widziany przez obserwatora składa się z N pionowych linii (kolumn) pokrytych odpowiednimi teksturami. Tekstury mają rodzielczość 64x64 piksele. Rendering zaczynamy od narysowania ścian. Realizuje się to w następujący sposób:

1. Wypuszczamy z punktu (Xobs, Yobs) 132 „promienie" – półproste przechodzące przez kolejne punkty (Xk, Yk) i znajdujemy miejsca, w których przecinają się ze ścianami na mapie (Xlk, Ylk).
2. Wyznaczamy długości poszczególnych promieni:

$$d_k = \sqrt{(X_k - XI_k)^2 + (Y_k - YI_k)^2}$$

3. Obliczamy wysokości kolejnych kolumn obrazu:

$$w_k = \left(\frac{w_0}{d_k} \cdot \cos\alpha \right)$$

gdzie α jest kątem zawartym między kierunkiem, w którym patrzy obserwator, a w_0 – współczynnikiem skalujacym wysokosc scian.

4. Wyznaczamy indeks kolumny tekstury (Uk), która zostanie nałożona na bieżącą kolumnę obrazu. Jest to przeskalowana część ułamkowa jednej ze współrzędnych punktu przecięcia promienia ze ścianą. Jeżeli promień trafił w ścianę „pionową" (przypadki d1, d4), wówczas Uk=frac(Ylk)*64, jeśli w ścianę „poziomą" (d2, d3), to Uk=frac(Xlk)*64.
5. Mając powyższe dane (wysokości i indeksy tekstur kolejnych kolumn), możemy narysować ściany. Poniżej znajduje się uproszczony kod w C realizujący to zadanie:

```
// szerokość i wysokośc ekranu
#define SZER_EKR 132

#define WYS_EKR 132

// tablica wysokosci poszczególnych kolumn i indeksów
kolumn tekstur, które mają być nałożone
int Wk[SZER_EKR], Uk[SZER_EKR];
unsigned char bufor_ekranu[SZER_EKR*WYS_EKR];
unsigned char jakas_tekstura[64*64];

void rysuj_kolumne(int k)
{
  int y1, v, dv;
  unsigned char *w, *t;

  y1 = (SZER_EKR - Wk[k]) / 2;
  v = 0;
  dv = (64 << 16) / Wk[k];
  w = (bufor_ekranu + SZER_EKR * y1 + k);
  t = jakas_tekstura + 64 * Uk[k];

// rysujemy 64-pikselowa kolumne tekstury rozciagnieta
tak, aby miała wysokosc Wk.
  for(i=0; i<Wk[k]; i++)
  {
     *w = *(t + (v >> 16));
     w+=SZER_EKR;
     v+=dv;
  }
}

void rysuj_sciany()
{
  int x;
// czyscimy ekran
  memset(bufor_ekranu, 0, SZER_EKR * WYS_EKR);

  for(x=0; x<SZER_EKR; x++)
     rysuj_kolumne(x);
}
```

Drugim etapem jest wyrenderowanie obiektów (wystroju pomieszczeń, przeciwników, itp.). Wyliczone wcześniej wysokości kolumn (Wk) pozwalają na wyeliminowanie tych, które są niewidoczne (np. znajdujących się w pomieszczeniu za ścianą). Następnie obiekty są sortowane, zmniejszane proporcjonalnie do odległości od obserwatora i rysowane w kolejności od najdalszego do najbliższego.
Ostatnią czynnością jest dorysowanie interfejsu użytkownika (liczby punktów, życia, itd.) i przesłanie całości do wyświetlacza.

spodu płytki. Kolejność montażu – standardowa, czyli od najmniejszych (rezystorów, kondensatorów SMD) do największych (elektrolity, złącza JP1, JP2). Na końcu mocujemy (np. klejem na gorąco) koszyk na baterie i podłączamy krótkimi przewodami do CN1.

Ostatnim montowanym elementem powinien być wyświetlacz. Jeśli posiadany przez nas ekran ma złącze przypominające wyglądem CN6, wystarczy połączyć go z płytką za pomocą kawałka przewodu taśmowego. Gorzej, gdy mamy wersję z mikrozłączem SMD – wówczas mamy do wyboru dwie możliwości:

– zdobycie odpowiedniego gniazda i przylutowanie go w miejscu CN5 (można wydobyć je z uszkodzonego telefonu),
– połączenie mikrozłącza w wyświetlaczu z CN6 na płytce za pomocą cienkich przewodów (np. kynaru). Wymaga to trochę cierpliwości i pewnego doświadczenia w lutowaniu. Po sprawdzeniu poprawności połączenia warto je zabezpieczyć przed uszkodzeniem np. zalewając żywicą epoksydową.

Zanim zainstalujemy wyświetlacz i DIP–moduł z mikrokontrolerem, ra-

Oprogramowanie i dodatkowe materiały dotyczące projektu można znaleźć na stronie http://wlostowski.ep.com.pl/wolfarm oraz na płycie CD dołączonej do EPo/oL.

dzę sprawdzić poprawność napięć zasilających (pin 1 JP1 – +5 V, pin 1 CN6 – +3,3 V). Jeśli wszystko jest w porządku, wkładamy je na miejsce, podłączamy do złącza CN6 programator FlashLink i włączamy zasilanie. Firmware konsoli znajduje się w pliku *wolf3d_str911_jtag.bin*. Po zaprogramowaniu urządzenie jest gotowe do pracy.

Jeśli nie posiadamy FlashLinka, możemy zaprogramować mikrokontroler przez RS232 – wówczas łączymy złącze CN7 z portem szeregowym komputera i wysyłamy oprogramowanie za pomocą dowolnego programu terminalowego (HyperTerminal, minicom, itp.). Plik z *firmware* nosi nazwę *wolf3d_str911_bootldr.bin*. Mikrokontroler musi mieć zainstalowany odpowiedni bootloader (plik *str911_gameconsole_bootloader.bin*). Instalacja jest jednorazowa, ale można jej dokonać tylko przez JTAG. Aby uruchomić aplikację bootloadera należy przy włączaniu zasilania konsoli trzymać wciśnięty przycisk Fire (SW6). Więcej informacji na temat bootloadera dla mikrokontrolerów STR9 można znaleźć w EP5/2007.

Oprogramowanie

Powstanie ARM–owej wersji „Wolfa" było możliwe dzięki udostępnieniu przez Id Software kodu źródłowego gry. To jedna z niewielu firm, które publikują źródła swoich starych gier – oprócz Wolfensteina są dostępne kody np. Dooma, Hexena, Heretica i Quake–ów 1, 2, 3. Chwała im za to!

Oryginalny Wolf3D był napisany w języku przypominającym trochę C (DOS–owy kompilator C Borlanda), pracował w 16–bitowym trybie *real*, korzystał z pamięci EMS i innych „dziwnych" rzeczy. Ponadto zawierał sporo wstawek asemblerowych. Utrudniało to stworzenie portu dla innych platform. Na szczęście znaleźli się ludzie, którzy opracowali wersję gry dla systemu Linux, pozbawioną 16–bitowego kodu i napisaną w całości w C. Właśnie na niej jest oparty port Wolfa dla STR9.

Poważnym problemem przy opracowywaniu wersji dla konsoli były wymagania pamięciowe Wolfensteina. Gra w wersji na PC potrzebowała około 1 MB RAM–u i 1,5 MB miejsca na dysku. Mikrokontroler STR911FW44 posiada 512 kB Flasha i 96 kB RAM. O ile pamięć RAM nie była poważnym problemem (wystarczyło zoptymalizowanie typów składowych struktur danych i przerzucenie części danych z RAM-u do Flasha), o tyle „upchanie" 1,5 MB kontentu gry do niecałego 0,5 MB pamięci Flash wymagało prawdziwych ewolucji programistycznych. Po usunięciu części grafiki (gra jest pozbawiona menu) rozmiar danych gry skurczył się do 1 MB, na których pomieszczenie dysponowałem mniej więcej 400 kilobajtami Flasha. Jedynym rozwiązaniem była kompresja, a ponieważ nasza platforma sprzętowa ma bardzo niewielką pamięć RAM, algorytmy musiały być na tyle szybkie, aby dane można było zdekompresować „w locie":

- tekstury zostały potraktowane zmodyfikowanym algorytmem LZ77, w którym słownik jest jednocześnie buforem docelowym (zaoszczędzono w ten sposób 4 kB pamięci RAM). 256 kB tekstur udało się „ścisnąć" do 107 kB,
- *sprite'y* (postacie, broń, obiekty statyczne) skompresowałem algorytmem RLE. 500 kB grafiki zmieściło się w 174 kB Flash,
- dźwięk (8–bit PCM, 8 kHz) został spakowany za pomoca algorytmu kodującego metodą Huffmana różnice między sąsiednimi próbkami. 148 kB próbek zajęło 97 kB pamięci Flash.

Dekompresja danych jest najbardziej czasochłonnym procesem wykonywanym przez oprogramowanie (zużywa ok 70% czasu procesora). Rdzeń ARM966E–S, na którym oparte są mikrokontrolery STR9 poradziłby sobie bez większych problemów także z nowszymi grami (Doom, Hexen). Potrzeba tylko trochę więcej RAM-u – 96 kB to stanowczo za mało.

Kod źródłowy gry jest dość obszerny (zajmuje około 400 kB), dlatego został podzielony na dwie części:

- właściwy kod gry, niezależny od sprzętu – znajduje się w podkatalogu *game*. Najważniejsze pliki to: *wl_draw.c* (silnik graficzny), *wl_act[1–3].c* (sztuczna inteligencja przeciwników), *wl_play.c* (główna pętla gry)
- pliki specyficzne dla mikrokontrolera STR911 – sterownik wyświetlacza (*epslcd.c*), dźwięku (*sd_str911.c*, *sound_FIQ.S*), klawiatury (*vi_str911.c*) itd – znajdują się w katalogu głównym.

Oprogramowanie powstało w Zintegrowanym Środowisku Uruchomieniowym pracującym pod kontrolą systemu Linux, składającym się z kompilatora GCC dla architektury ARM, edytora z podświetlaniem składni *mcedit*, programu *make* i flashera *str91isp*. Źródła gry bez problemu kompilują się również w WinARM–ie.

Słowo końcowe

Projekt nie powstał po to, by pokazać, jak zrobić grę 3D na ARM-a. Zadaniem opisanej w artykule konsoli jest efektowna demonstracja możliwości 32–bitowych mikrokontrolerów i zachęcenie Czytelników do bliższego zapoznania się z nimi. Programowanie ARM–ów lub innych 32–bitowców wcale nie jest trudniejsze niż procesorów 8–bitowych – wystarczy znajomość języka C. Często producenci udostępniają kompletne biblioteki do obsługi peryferiów wbudowanych w dany układ, dzięki czemu można tworzyć całkiem rozbudowane projekty nie znając nawet rejestrów SFR. Cały czas powiększa się dostępna w Internecie baza wiedzy w postaci kursów, not aplikacyjnych i przykładowych projektów.

ARM-y nie wymagają drogich programatorów ani kosztownego oprogramowania. Na początek wystarczy zwykły Wiggler (koszt wykonania około 10 zł), pakiet WinARM (za darmo) i trochę chęci. Rozbudowane debuggery i sprzętowe emulatory bywają przydatne, choć w 99 procentach przypadków da się je zastąpić wywołaniem funkcji *printf()* – w taki sposób zostało odpluskwione oprogramowanie konsoli. Zachęcam zatem do tworzenia własnych aplikacji dla 32–bitowych mikrokontrolerów i pochwalenia się nimi na łamach EP.

Tomasz Włostowski, EP
tomasz.wlostowski@ep.com.pl

W projekcie wykorzystano gotowy moduł dipARM z zamontowanym mikrokontrolerem z rodziny STR911. Dzięki takiemu rozwiązaniu kłopoty z montażem elementów zostały zminimalizowane, a sam moduł można wykorzystać w wielu projektach bez konieczności używania narzędzi. Dodatkowym ułatwieniem dla użytkowników są zintegrowane na płytce modułu dipARM: rezonator kwarcowy i stabilizatory napięć zasilających.
Moduł ZL21ARM znajduje się w ofercie handlowej AVT (*sklep.avt.pl*).