
# VGARM, czyli ARM + monitor komputerowy część 2

## Montaż i uruchomienie

Cały moduł mieści się na niewielkiej dwustronnej płytce z **rysunku 3**. Poza złączami, kwarcem i potencjometrem, wszystkie elementy montowane są powierzchniowo. Niewielkie rozmiary podyktowały konieczność dość mocnego upakowania komponentów, w niektórych obszarach zatem montaż musi być wykonany uważnie i starannie. Szczególnie tyczy się to procesora, którego 100 nóżek rozstawionych jest w rastrze 0,5mm.

Po montażu doprowadzamy napięcie zasilania 5V, a następnie należy zaprogramować mikrokontroler przez ISP, wykorzystując w tym celu wyprowadzenia RX, TX, ISP i RST. Można do tego celu użyć opisanego niedawno na łamach EdW uniwersalnego programatora UniProgUSB bądź dowolnego innego. Ponieważ procesor łatwo można wprowadzić w tryb programowania ręcznie, tak naprawdę wystarczy nam odpowiedni konwerter RS232 lub USB do poziomu 3,3...5V. Informacje na ten temat są szeroko dostępne w Internecie i literaturze. Plik programu nazwany jest *terminal.hex* i odnajdziemy go w folderze Debug wewnątrz katalogu z projektem.

Po poprawnym zaprogramowaniu i zresetowaniu układ powinien od razu poprawnie pracować. Możemy to sprawdzić, podłączając do złącza VGA dowolny monitor. Na ekranie powinien pojawić się ekran pomocy jak na **fotografii tytułowej**. Jeżeli mamy możliwość połączenia poprzez konwerter z portem szeregowym lub USB komputera, możemy uruchomić terminal i wypróbować działanie różnych komend. Na początek polecam wysłać dowolny ciąg znaków, który powinien pojawić się od lewego górnego rogu ekranu w białym kolorze czcionki na czarnym tle.

Jeżeli w slocie umieścimy kartę SD lub MMC, to po resecie na terminalu zobaczymy jej status, tzn. informację, czy została poprawnie zainicjalizowana. Jeżeli tak, to po podłączeniu do złącza audio np. słuchawek usłyszymy zapisaną na karcie muzykę bądź też zupełny bełkot w wypadku nieprzygotowanej karty ;-)

## Obsługa

Komendy sterujące pracą modułu są w większości identyczne w działaniu, jak te dla AVRTV. Ich implementację można zobaczyć w pliku *uart.c* w obsłudze przerwania od tego interfejsu. Działanie komend pokrótce wyjaśnione jest na ekranie pomocy, ale rozwińmy je nieco. Część opisu zamieszczonego w **tabeli 2** pozwoliłem sobie zaczerpnąć z artykułu kolegi Michała.

Myślę, że zrozumienie działania tego zestawu komend nie stwarza większych trudności, ale na wszelki wypadek podam jeden przykład zastosowania. Wysyłamy do modułu następujący ciąg znaków (zakładamy, że kursor znajduje się w pozycji 0,0, ustawiony jest tryb terminalowy i nie wysyłamy kolorów):

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

Na ekranie otrzymamy:

PIERWSZA
DRUGA:)
CZWARTA

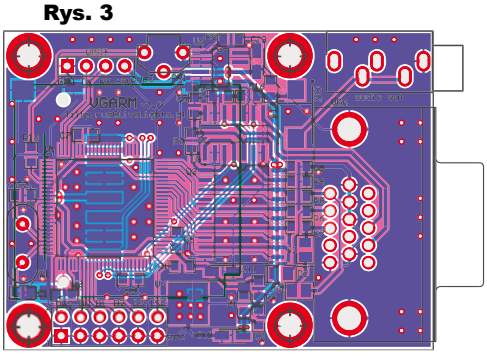
Podwójna litera „A” została usunięta komendą „[B”, następnie komenda „[A” usunęła zawartość trzeciej linii , trzykrotne wywołanie „[R” spowodowało przesunięcie początku czwartej linii tekstu, komenda „[P” przeniosła kursor na koniec drugiej linii (cyfry podane w nawiasach należy wprowadzać wprost „binarnie”, a nie dziesiętnie), dzięki czemu znalazła się tam emotikonka :). Identyczny przykład i kilka innych można znaleźć w opisie projektu AVRTV. Dodam jeszcze, że w odróżnieniu od tamtego projektu, znak CR i LF są w VGARM interpretowane odpowiednio jako przejście do początku linii i przejście do następnej linii.

Nieco szerzej wyjaśnijmy jeszcze sprawę wprowadzania koloru. Otóż w trybie domyślnym do modułu wysyłamy kolejne znaki i zostaną one wyświetlone w kolorze ustawionym jako domyślny. Ten domyślny kolor, jak widać w tabeli 2, ustalany jest przez drugi parametr komendy „[M”. Przykładowo dla wartości 0x24, czcionka koloru czerwonego będzie zapisywana na zielonym tle. Stanie się tak, gdyż starsze 4 bity kodujące kolor tła przyjmują wartość 2, która standardowo dla palety 16 kolorów oznacza kolor zielony, natomiast młodsze bity przyjmą wartość 4, co oznacza tekst w kolorze czerwonym. Kilka przykładów w **tabeli 3**.

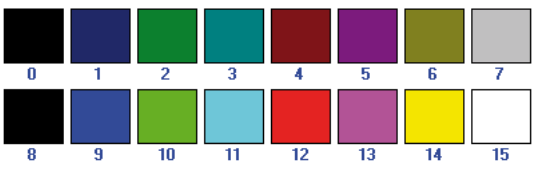
Na **rysunku 4** pokazana jest paleta 15 kolorów. Jak widać, kolory od 0 do 7 to barwy ciemne, a powyżej jasne. Oczywiście wyjątkiem jest tu kolor czarny, który występuje zarówno pod wartością 0, jak i 8. Wartość 7 odpowiada kolorowi szaremu.

W tym samym formacie podawany jest kolor indywidualnych znaków. Kiedy w parametrze „a” ustawimy bit0 na 1, a bit1 jest standardowo ustawiony na wartość 1, to wchodzimy w tryb, w którym naprzemiennie wysyłany jest znak i jego kolor. Innymi słowy, na każdy znak przypadają teraz dwa bajty, gdzie pierwszy to oczy-

**Rys. 3**



**Rys. 4**



wiście numer znaku w tablicy czcionki, a drugi to właśnie kolor. Ustawienie bitu 1 na 0 wprowadza nas w ciekawy tryb, gdzie wszystkie wysyłane do układu bajty (poza komendami) są interpretowane jako kolor. Można w ten sposób tworzyć na przykład prostą grafikę o rozdzielczości 80x40, gdzie „pikselami" są puste tła znaków.

Jeżeli chodzi o obsługę audio, to w obecnej wersji oprogramowania do tego celu nie są przewidziane żadne komendy i pozostawiam to inwencji użytkownika. Samo przygotowanie dźwięku dzieli się na następujące etapy:

Oryginalny plik dźwiękowy należy przekonwertować do formatu 8bit, mono, 31457Hz. Do tego celu można wykorzystać program do obróbki audio np. darmowy *Audacity*.

Teraz uruchamiamy niewielką aplikację *wav converter* i po podaniu nazwy zresamplowanego pliku czekamy na jego dalszą obróbkę. W tym momencie usuwane są niektóre próbki i wynikowy plik *out.txt* ma format odpowiedni do zapisania na karcie SD/MMC.

Trzecim krokiem jest zapisanie danych na karcie. Ponieważ nie korzystamy z żadnego systemu plików, należy posłużyć się programem *dd*, który pozwala na bezpośredni dostęp do sektorów wybranego dysku. W katalogu *audio* znajduje się przykładowy plik wsadowy *zapisz.bat*, który po uruchomieniu kopiuje zawartość pliku *out.txt* bezpośrednio na kartę, zaczynając od strony o adresie 0.

UWAGA! Programu *dd* należy używać ostrożnie, gdyż przy źle podanych parametrach, w skrajnym przypadku może dojść do uszkodzenia danych na dysku naszego komputera.

Przykładowy obraz z fotografii 1 (wygenerowany przez moduł), został stworzony za pomocą programu *VGARM terminal* autorstwa mojego kolegi **Gawła Kowalika** – **rysunek 5**. Jest to bardzo przydatna aplikacja, zarówno podczas testów, jak i przy normalnym użytkowaniu modułu. Pozwala ona między innymi na wprowadzenie wyglądu całego ekranu, a następnie przesłanie go poprzez port szeregowy do modułu VGARM.

| komenda | opis |
|---|---|
| [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. Dla zachowania kompatybilności z AVRTV po zakończeniu kasowania moduł odeśle znak „[". |
| [Pxy | SET POSITION – ustawia pozycję kursora na ekranie, x oraz y są bajtami, których wartości reprezentują odpowiednio numer kolumny (0–84) oraz numer wiersza (0–39). 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" i „[D". |
| [S | SCREENSHOT – odsyła całą zawartość bufora tekstu i koloru (3200 bajtów z kodami znaków, a następnie 3200 bajtów kodujących kolor). |
| [[ | Pozwala na wyświetlenie znaku „[". |
| [? | HELP – wyświetla ekran pomocy, informacje o autorze oraz tablicę znaków czcionki i paletę kolorów. |
| [Mab | MODE – funkcja niewystępująca w AVRTV, a służąca do ustalenia parametrów związanych z kolorami. Parametr „a" odpowiada za tryb komunikacji. Jeżeli bit0=1, to w komunikacji z modułem przekazywane są kolory każdego znaku z osobna. Jeżeli bit1=1, to w komunikacji wysyłane są kody znaków (tryb domyślny). Przykładowo kiedy bity mają wartości 11, to do modułu należy przesyłać naprzemiennie znak i jego kolor. Kiedy wartość wynosi 10, wysyłane są jedynie znaki, a 01 oznacza chęć przesyłania jedynie kolorów, co także może mieć sens. Pozostałe bity nie mają zastosowania. Parametr „b" ustala globalny schemat kolorów, przy czym na wyższych 4 bitach kodowany jest kolor tła, a na niższych kolor czcionki. Ten schemat kolorów używany jest przy domyślnym trybie wprowadzania tekstu, czyli bez podawania koloru każdego znaku z osobna oraz podczas wszystkich komend kasujących zawartość ekranu (następuje wypełnienie tłem koloru ustalonego komendą „[M"). |

**Tabela 2**

**Tabela 3**

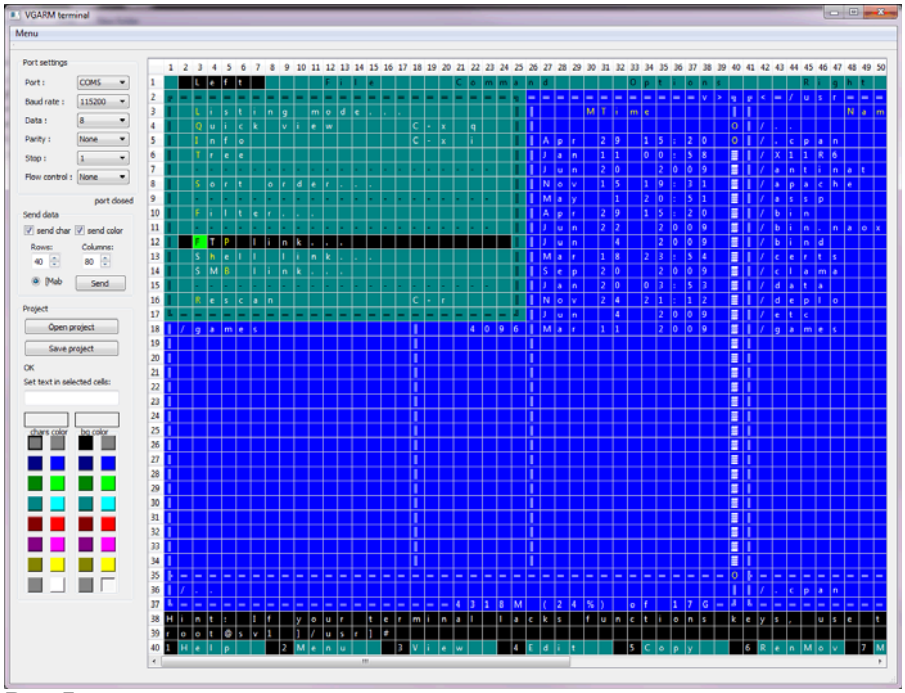
| wartość | | tło | znaki | przykład |
|---|---|---|---|---|
| 0x24 | 0010 0100 | ♦ | ♦ | Kolorowy tekst |
| 0xB5 | 1011 0101 | ♦ | ♦ | Kolorowy tekst |
| 0x0E | 0000 1110 | ♦ | ♦ | Kolorowy tekst |
| 0xC1 | 1100 0001 | ♦ | ♦ | Kolorowy tekst |

## Możliwości zmian

Pierwszą i podstawową zmianą, do której zachęcam, jest wykorzystanie procesora LPC1766 zamiast LPC1768. Różni on się jedynie wielkością pamięci flash (256KB zamiast 512KB), a kosztuje dobrych kilka złotych mniej. Choć pierwsze próby prowadzone były na makiecie z LPC1768, to w prezentowanym module pracuje właśnie LPC1766 i spisuje się świetnie.

Układ przy niewielkich modyfikacjach stwarza możliwość eksperymentowania z głębią koloru. Przykładowo można na kolory czcionki przeznaczyć 6 bitów (64 kolory), a na tło tylko 2 (4 kolory). Ktoś może uznać, że tło w ogóle nie jest istotne i pozostawić je czarne, a wszystkie 8 bitów przeznaczyć na barwę znaku, tym samym uzyskując 256 kolorów. Jeszcze ciekawsze byłoby zdublowanie obwodów buforujących i wykorzystanie 16 bitów np. po równo na tło i znaki, lub ponownie zmieniając te proporcje. W procesorze na pewno wystarczy pamięci na dwukrotne zwiększenie głębi koloru, a czas wysyłania 16 bitów na port powinien być taki sam lub niewiele dłuższy niż obecnie. Jest więc to jak najbardziej realne zadanie. Przy odpowiednim dostosowaniu timingów, można poważnie pomyśleć o stworzeniu wersji projektu przeznaczonej do współpracy z telewizorami za pośrednictwem sygnałów RGB w złączu SCART (Euro).

Jedną z bardziej oczywistych zmian, jakich można dokonać w programie, jest zmiana czcionki, a być może dodanie możliwości wyboru kilku różnych czcionek. Można także pomyśleć nad inną wysokością znaków. Ciekawym rozwiązaniem byłaby zmiana orientacji obrazu z poziomej na pionową, czyli przekręcenie wszystkich znaków o 90°. Dzięki temu można by uniknąć problemu poziomego odstępu między znakami w trybie SPI i uzyskać dużą dowolność w ich rozmiarze, szczególnie szerokości oraz ciągłość obrazu w poziomie. Wymaga to oczywiście wysyłania innej długości danych przez interfejs SSP, a zatem także dużych zmian i dobrego rozumienia zasady działania programu. Interfejs I2S jest mniej elastyczny pod względem ustalania wielkości wysyłanych danych (liczby bitów w słowie).



Rys. 5

Jeżeli chodzi o różnice pomiędzy trybami SPI i I2S, to wraz z użyciem tego drugiego pojawił się pewien problem z transmisją danych przez UART. Otóż dla rozpoczęcia transmisji po I2S z wykorzystaniem DMA potrzeba nieco więcej czasu czasu niż w przypadku SSP. Spowodowało to skrócenie wolnego czasu na końcu każdej linii, co w konsekwencji prowadziło do gubienia odbieranych przez procesor bajtów przy wyższych prędkościach (>9600). Obecnie rozwiązaniem tego problemu jest zmiana priorytetu przerwań – przerwanie od UART-u ma najwyższy priorytet, dzięki czemu dane nigdy nie zostaną utracone, jednak odbija się to na ciągłości wyświetlania obrazu. Innym rozwiązaniem byłoby zapewnienie stosownych opóźnień w transmisji np. co 10 wysłanych bajtów. Kosztem kompatybilności z AVRTV można zupełnie przerobić protokół transmisji tak, żeby wysyłać wiele znaków jeden za drugim i renderować je dopiero po odebraniu wszystkich, a nie tak jak ma to miejsce teraz, po każdym z osobna.

Do programu można próbować dodać pewną funkcjonalność, związaną z generowaniem grafiki niewielkiej rozdzielczości. Przy zastosowaniu różnych sprytnych rozwiązań można by zapewne uzyskać obrazy o rozdzielczości zbliżonej do 160x120, a może więcej.

Spore pole do popisu daje odtwarzanie audio. Można śmiało dodać nowe komendy z tym związane, a także eksperymentować z samym sposobem synchronizacji audio z obrazem. Zamiast powtarzania próbek, można spróbować odczytać 2 pełne strony i „marnotrawiąc" nieco miejsca na karcie, zapisać na niej wszystkie próbki.

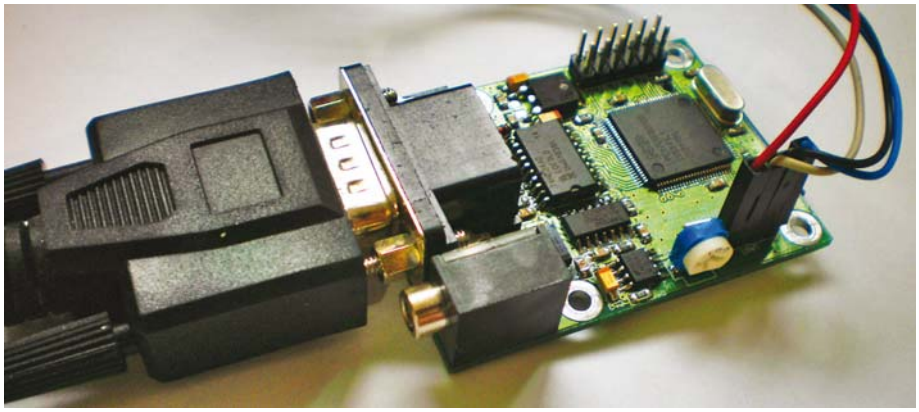
Dźwięk nie musi zawsze pochodzić z karty SD; można przecież dodać funkcje generujące prostsze dźwięki. Byłoby to szczególnie przydatne przy wykorzystaniu modułu np. jako interfejs sterujący i monitorujący pracę jakiegoś urządzenia. Piny wykorzystywane domyślnie dla zworek ustalających prędkość UARTa można śmiało wykorzystać do innych celów.

Jedną z ciekawszych możliwości byłoby stworzenie na samym module lub przy współpracy z dodatkowym mikrokontrolerem prostej gry „telewizyjnej". Można zacząć od nieśmiertelnego Tetrisa czy Snake'a, którym kolorowy obraz na pewno doda animuszu. Jednak drzemiące w układzie możliwości mogą się na tym nie kończyć. Otóż sporo starych gier konsolowych wykorzystywało tak zwaną grafikę kafelkową. Innymi słowy na ekranie widzieliśmy stosunkowo rozbudowane wzory w „wysokiej" rozdzielczości, ale były one składane z „kafelków", czyli niewielkich kawałków obrazu np. 8x8 pikseli zapisanych w pamięci. Czyż nie jest to tak naprawdę to samo, co wyświetlanie znaków czcionki? No może nie do końca, bo w naszym układzie kafelek dysponuje jedynie dwoma barwami, niemniej jednak propozycja też jest warta przemyślenia.

W tej chwili karta pamięci wykorzystywana jest tylko na potrzeby audio, jednak daje ona możliwość zapisania bardzo wielu tekstów, danych czy np. całych plansz z wyglądem ekranu. Sam moduł natomiast można wykorzystać na bardzo wiele sposobów, np. jako część „podglądacza" linii interfejsów RS232, 485, 422 i podobnych, terminal zgodny z VT100 do komunikacji np. z systemem Linux na minikomputerkach i wiele, wiele innych.

Na koniec polecam jeszcze wypróbowanie „ukrytej" komendy „[X" ;-) Efekt jej działania można zobaczyć pod adresem: *http:www.youtube.com/watch?v=tHymB8kCZ0&feature=youtu.be*.

**Filip Rus**
filip.rus@livelights.pl



**Wykaz elementów**

| | |
|---|---|
| R1, R2, R5 | 10kΩ |
| R3, R4 | 1kΩ |
| R6-R8 | 470Ω |
| R9 | 10Ω |
| R10 | 180kΩ |
| PR1 | pot 10kΩ |
| C1, C2 | 39pF |
| C3-C9 | 100nF |
| C11-C13 | 10uF |
| C14 | 33nF |
| C15 | 100-220uF |
| C16 | 47nF |
| U1 | LPC1766/68 |
| U2 | LF33 |
| U3 | HC244 |
| U4 | HC125 |
| U6 | LM386 |
| D1-D3 | BAS83 |
| VGA | DB15 kątowe |
| UART | goldpin 1x4 |
| JPS | goldpin 2x6 |
| audio out | gniazdo jack stereo pionowe |
| card | gniazdo karty SD/MMC |

**Płytka drukowana jest dostępna w sieci handlowej AVT jako kit szkolny AVT-3041.**