Terminal Szeregowy – BIOS

Wojciech Kaniewski, gr. O
VI sem. EiT, 2003/2004

1. Założenia projektu

Celem projektu jest stworzenie oprogramowania i sprzętu pozwalającego zmianę zwykłego komputera klasy PC w terminal szeregowy bez użycia stacji dysków, dysku twardego czy jakiegokolwiek innego nośnika danych. Początkowo program miał zastępować BIOS komputera, jednak po przeanalizowaniu budowy kilku komputerów okazało się, że należałoby stworzyć osobne wersje dla większości dostępnych na rynku płyt głównych. Oryginalny BIOS zajmuje się inicjalizacją wszystkich podzespołów komputera – od prostej inicjalizacji standardowych układów typu 8259 czy 8254 w starszych płytach, do inicjalizacji mostków północnych i południowych w nowych komputerach. Niestety ze względu na różnorodność rozwiązań nie da się napisać programu, który za pomocą tego samego kodu inicjalizowałby wszystkie dostępne płyty główne. Dlatego zdecydowano się na pozostawienie oryginalnego BIOS-u i stworzenie karty rozszerzeń ISA, która będzie zawierać podstawkę pamięci EPROM, w której można umieścić dowolny program, który zostanie wykonany po zainicjowaniu systemu i przeprowadzeniu POST (Power On Self Test). Uniezależnia to program od sprzętu, ponieważ ten będzie dostępny przez ustandaryzowane funkcje BIOS. Z powyższych powodów projekt podzielono na część sprzętową (karta rozszerzeń) i programową (program do umieszczenia w pamięci EPROM).

2. Etapy realizacji

Wykonanie części sprzętowej projektu podzielono na następujące etapy:

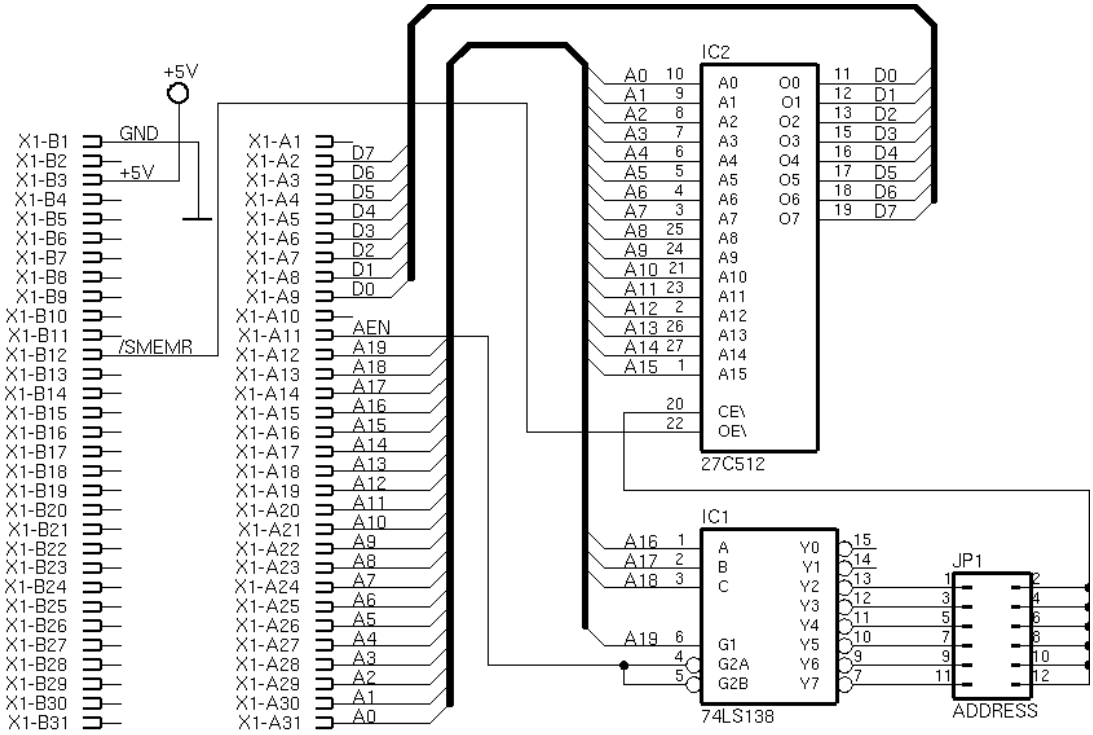
1. Przeanalizowanie zasady działania złącza ISA.
2. Zaprojektowanie i zbudowanie dekodera adresów.
3. Podłączenie wejść adresowych, wyjść danych oraz wejścia OE\ pamięci EPROM do złącza ISA, podłączenie wejścia CE\ do dekodera adresów.
4. Przetestowanie karty w komputerze za pomocą prostego programu zapisanego w pamięci EPROM.

Wykonanie części programowej podzielono na:

1. Znalezienie odpowiedniego kompilatora C, który pozwala łączyć kod z kodem w assemblerze i umieszczać go w dowolnym miejscu pamięci.
2. Analiza kodów źródłowych istniejących BIOS-ów (FreeBIOS, OpenBIOS, LinuxBIOS, BIOS dołączony do emulatora Bochs) w celu poznania zasady działania.
3. Napisanie kodu inicjującego pamięć (segmenty danych i stosu) oraz szkieletu programu w C, którego funkcja `main()` zostanie wywołana z kodu asemblerowego zaraz po inicjalizacji.
4. Napisanie funkcji obsługi ekranu w języku C – wyświetlanie znaków, przesuwanie kursora, przesuwanie ekranu, gdy kursor dojdzie do końca ekranu.
5. Inicjalizacja i obsługa systemu przerwań.
6. Obsługa klawiatury – początkowo wyświetlanie danych z klawiatury na ekranie.
7. Inicjalizacja portu szeregowego – najpierw dla standardowych ustawień 9600 bps, brak parzystości, 1 bit stopu.
8. Obsługa portu szeregowego na przerwaniach – obsługa bufora FIFO, wyświetlanie danych z portu szeregowego na ekranie, wysyłanie danych z klawiatury na port szeregowy.
9. Obsługa sekwencji sterujących VT100.
10. Kod konfiguracji portu szeregowego (wybór portu, parametrów transmisji).
11. Testowanie BIOS-u na komputerze w laboratorium i ewentualne poprawki.

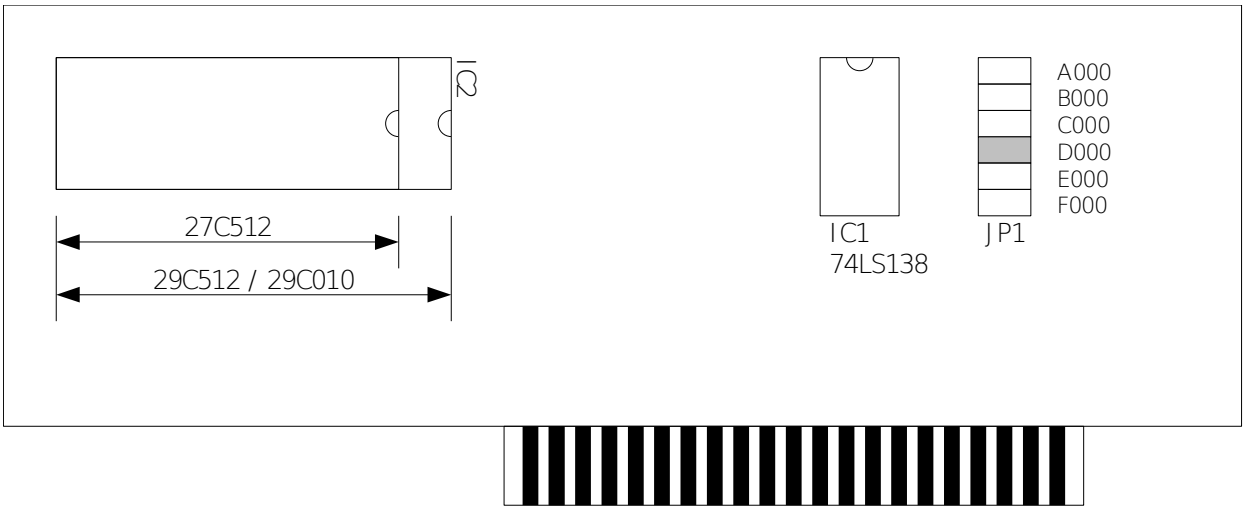
3. Część sprzętowa projektu

3.1. Schemat układu



Karta rozszerzeń zawiera złącze pamięci EPROM z liniami A0-A15 oraz D0-D7 podłączonymi bezpośrednio do magistrali ISA. Dekoder adresów zbudowano na układzie demultipleksera z 3 bitów wejściowych na 8 bitów wyjściowych. Jest to rozwiązanie znacznie łatwiejsze niż budowa dekodera adresów z bramek NAND, a poza tym pozwala w łatwy sposób konfigurować adres bazowy pamięci. Demultiplekser zawiera dodatkowo 3 wejścia sygnałów (jedno proste, dwa zanegowane), do których podłączono linię adresową A19 (wszystkie konfigurowalne przedziały będą miały ją w stanie wysokim) oraz AEN (stan wysoki jest wykorzystywany przy transferach DMA, dla normalnej pracy jest w stanie niskim).

3.2. Rozmieszczenie elementów na płytce



Zastosowane większego gniazda pamięci EPROM (32 piny zamiast 28 pinów) pozwala umieścić na karcie również pamięci Flash 29C512 i 29C010, które mają większą ilość wyprowadzeń, ale starają się być maksymalnie zgodne z pamięciami EPROM.

4. Część programowa projektu

4.1. Wykorzystane narzędzia

Do kompilacji wykorzystano *bcc* (Bruce's C Compiler) i asembler *as86* dostępne na stronie internetowej http://www.cix.co.uk/~mayday/. Dla ułatwienia kompilacji wykorzystano *GNU Make*. Program testowano za pomocą emulatora komputera PC *Bochs* dostępnego na stronie http://www.bochs.org/.

4.2. Budowa programu

Program składa się kilku modułów pełniących różne funkcje:

- `io.c` – niskopoziomowe funkcje wejścia-wyjścia:

  `portWrite(port, value)` – zapis do portu wejścia-wyjścia

  `portRead(port)` – odczyt z portu wejścia-wyjścia

  `memoryReadByte(segment, offset)` – odczyt bajtu z pamięci

  `memoryReadWord(segment, offset)` – odczyt słowa z pamięci

  `memoryWriteByte(segment, offset, x)` – zapis bajtu do pamięci

  `memoryWriteWord(segment, offset, x)` – zapis słowa do pamięci

  `interruptSet(number, offset)` – ustawia obsługę przerwania na podaną funkcję

- `keyboard.c` – funkcje obsługi klawiatury:

  `keyboardGetChar()` - czeka na naciśnięcie klawisza i zwraca jego kod

  `keyboardPressed()` - informuje, czy wciśnięto jakiś klawisz

  Funkcje klawiatury korzystają ze standardowych funkcji BIOS przerwania 16h.

- `screen.c` – funkcje obsługi ekranu:

  `screenCursorUpdate()` - przesuwa kursor ekranowy w odpowiednie miejsce

  `screenSave()` - zapisuje zawartość ekranu i atrybuty do bufora

  `screenRestore()` - przywraca ekran i atrybuty z bufora

  `screenClearLine(line)` – czyści podaną linię

  `screenClear()` - czyści cały ekran

  `screenScroll(from, to, lines)` – przesuwa zawartość ekranu liniami

  `screenCursorX(scroll)` - przesuwa kursor w podanym kierunku X

  `screenPutChar(ch)` – wyświetla podany znak ASCII

  `screenPutHex(x)` – wyświetla liczbę heksadecymalną

  `screenPutString(s)` – wyświetla ciąg znaków

  `screenInit()` - inicjalizuje ekran

  Funkcje obsługi ekranu korzystają z przerwania 10h tylko do włączenia trybu 80x25 i przesunięcia kursora. Wszystkie operacje związane z wyświetlaniem znaków operują bezpośrednio na buforze karty graficznej w segmencie B800h.

- `serial.c` – funkcje obsługi portu szeregowego:

  `serialDataReady()` - makro informujące czy odebrano jakiś znak

  `serialGetChar()` - pobiera z bufora wejściowego znak

  `serialPutChar()` - wysyła do bufora wyjściowego znak

  `serialPutString()` - wysyła ciąg znaków do bufora wyjściowego

  `serialSetup()` - ustawia port szeregowy do podanych parametrów

  `serialConfig()` - uruchamia dialog konfiguracji portu szeregowego

  Funkcje obsługi portu szeregowego operują bezpośrednio na portach portu szeregowego. Wysyłanie i odbieranie jest zrealizowane na przerwaniach i wykorzystuje bufory 8192 bajtów dla danych obieranych i 256 bajtów dla danych wysyłanych. Pozwala to uniknąć problemów z gubieniem znaków przy większych szybkościach. Kontrola przepływu nie jest obsługiwana, jednak włączenie na stałe odpowiednich linii sygnałowych (RTS i DTR) pozwala wykorzystać zwykłe przewody RS-232, nie tylko null-modem. Wciśnięcie klawisza Scroll Lock podczas uruchamiania programu włączy pętlę lokalną portu szeregowego – wszystkie wysyłane dane będą wracały do portu (pozwala to testować obsługę terminalu bez podłączania niczego do portu). Konfiguracja portu szeregowego jest dostępna po wciśnięciu klawisza F12.

- `terminal.c` – główny moduł programu:

  Zdefiniowanie w tym pliku pewnych dyrektyw preprocesora pozwala śledzić działanie programu i analizować potencjalne problemy:

  - `DEBUG_KEYBOARD` – wyświetlane będą kody wciśniętych klawiszy w prawym dolnym rogu ekranu.
  - `DEBUG_SERIAL` – w ostatniej linii ekranu będzie wyświetlana zawartość rejestrów portu szeregowego.
  - `DEBUG_TRANSFER` – oprócz zwykłych kodów ASCII będą wyświetlane wartości heksadecymalne wszystkich wysłanych i odebranych bajtów.

- `vt100.c` – funkcje obsługi sekwencji sterujących VT100:

  `vtLEDUpdate()` - uaktualnia stan diod LED

  `vtGetChar()` - pobiera sekwencję VT100 dla wciśniętego klawisza

  `vtPutChar(ch)` – wyświetla bajt analizując sekwencje VT100

  Lista obsługiwanych sekwencji VT100 znajduje się w dalszych rozdziałach dokumentacji.

Program jest kompilowany jako zawartość pamięci ROM dla karty rozszerzeń oraz jako plik COM dla systemu MS-DOS (w tej wersji wciśnięcie klawisza F11 powoduje powrót do systemu operacyjnego). Wersja do pamięci ROM na początku kodu zawiera sygnaturę pamięci rozszerzeń (`0x55`, `0xAA`) po której znajduje się bajt określający rozmiar programu w 512-bajtowych jednostkach. Kod programu rozpoczyna się od bajtu o offsecie 3. Suma kontrolna takiego programu musi się równać 0, więc po kompilacji do bajtu o offsecie 6 jest wpisywany bajt uzupełnienia sumy do 256. Program generujący sumę kontrolą jest dołączony do źródeł programu.

4.3. Obsługa portu szeregowego

Standardowy port szeregowy oparty na układach 8250 lub 16x50 zawiera następujące rejestry:

COM1	*COM2*	*Offset*	*DLAB*	*Rejestr*
3F8h	2F8h	+0	0	RBR Receive Buffer Register (odczyt) THR Transmitter Holding Register (zapis)
3F9h	2F9h	+1	0	IER Interrupt Enable Register
3F8h	2F8h	+0	1	DL Divisor Latch (LSB)
3F9h	2F9h	+1	1	DL Divisor Latch (MSB)
3FAh	2FAh	+2	x	IIR Interrupt Identification Register (odczyt) FCR FIFO Control Register (zapis, 16550+)
3FBh	2FBh	+3	x	LCR Line Control Register
3FCh	2FCh	+4	x	MCR Modem Control Register
3FDh	2FDh	+5	x	LSR Line Status Register
3FEh	2FEh	+6	x	MSR Modem Status Register
3FFh	2FFh	+7	x	SCR Scratch Register (16450+)

IER Interrupt Enable Register (+1)								
Bit	7	6	5	4	3	2	1	0
Nazwa	-	-	-	-	EDSSI	ELSI	ETBEI	ERBFI
Domyślnie	0	0	0	0	0	0	0	0

Rejestr wybiera, które zdarzenia powodują wywołanie przerwania:

- ERBFI (bit 0) – generuje przerwanie, gdy dane oczekują na odczytanie (DR)
- ETBEI (bit 1) – generuje przerwanie jeśli rejestr wyjściowy jest pusty (THRE)
- ELSI (bit 2) – generuje przerwanie w przypadku błędów transmisji
- EDSSI (bit 3) – generuje przerwanie gdy zmieni się stan którejś z linii kontrolnych MSR

IIR Interrupt Identification Register (+2)								
Bit	7	6	5	4	3	2	1	0
Nazwa	FIFO enable	FIFO enable	-	-	IID			Pending
Domyślnie	0	0	0	0	0	0	0	1

Rejestr pozwala poznać przyczynę przerwania. Jedno przerwanie jest zgłaszane za jednym razem. Bit 0 określa czy przerwanie zostało wywołane przez ten port szeregowy. Po przeanalizowaniu rejestru należy przeczytać go ponownie i sprawdzić, czy bit 0 jest nadal w stanie niskim. Jeśli tak, należy obsłużyć kolejne przerwanie. Procedurę obsługi przerwania można obsłużyć, gdy bit 0 jest w stanie wysokim.

Bit 3	*Bit 2*	*Bit 1*	*Bit 0*	*Priorytet*	*Znaczenie*
x	x	x	1	x	Brak oczekujących przerwań.
0	1	1	0	najwyższy	Jeden z bitów rejestrów LSR został ustawiony: OE, PE, FE lub BI. Przerwanie czyszczone czytaniem rejestru LSR.
0	1	0	0	...	Odebrano znak lub przekroczono poziom kolejki FIFO. Przerwanie czyszczone czytaniem RBR dopóki poziom kolejki nie zejdzie poniżej określonego maksimum.

Bit 3	*Bit 2*	*Bit 1*	*Bit 0*	*Priorytet*	*Znaczenie*
1	1	0	0	...	Nie podjęto żadnej akcji od okresu 4 słów, mimo że dane są w kolejce. Przerwanie czyszczone czytaniem rejestru RBR.
0	0	1	0	...	Rejestr nadajnika pusty. Przerwanie czyszczone czytaniem tego rejestru lub zapisem do THR.
0	0	0	0	najniższy	Zmienił się stan jednej z linii sterującej. Przerwanie czyszczone czytaniem rejestru MSR.

FCR FIFO Control Register (+2)								
Bit	7	6	5	4	3	2	1	0
Nazwa	Trigger Level		-	-	DMA Enable	TX Clear	RX Clear	FIFO Enable
Domyślnie	0	0	0	0	0	0	0	0

Rejestr służy do sterowania kolejki FIFO w układach 16550 i lepszych:

- FIFO Enable (bit 0) – włączenie kolejki FIFO
- RX Clear (bit 1) – wyczyszczenie kolejki odbiorczej
- TX Clear (bit 2) – wyczyszczenie kolejki nadawczej
- DMA Enable (bit 3) – włączenie trybu DMA (nie ma znaczenia w PC)
- Trigger Level (bity 6 i 7) – poziom kolejki odbiorczej, przy której zostanie wygenerowane przerwanie DR

Bit 7	*Bit 6*	*Poziom kolejki*
0	0	1
0	1	4
1	0	8
1	1	14

LCR Line Control Register (+3)								
Bit	7	6	5	4	3	2	1	0
Nazwa	DLAB	SBR	Stick parity	Even parity	Parity enable	Stop bits	Word length	
Domyślnie	0	0	0	0	0	0	0	0

Rejestr określa parametry transmisji i pozwala przez ustawienie bitu DLAB określenie prędkości transmisji.

Bit 5	*Bit 4*	*Bit 3*	*Parzystość*
x	x	0	None
0	0	1	Odd
0	1	1	Even
1	0	1	Mark
1	1	1	Space

Bit 1	*Bit 0*	*Długość słowa*
0	0	5 bitów
0	1	6 bitów
1	0	7 bitów
1	1	8 bitów

Bit 2	*Długość słowa*	*Bity stopu*
0	x	1
1	5	1,5
1	6, 7, 8	2

Bit 7	*Rejestry +0,+1*
0	Normalne funkcje

Bit 6	*Wyjście TXD*
0	Normalna funkcja

Bit 7	*Rejestry +0,+1*
1	Divisor Latch (DL)

Bit 6	*Wyjście TXD*
1	+12V (break)

MCR Modem Control Register (+4)								
Bit	7	6	5	4	3	2	1	0
Nazwa	-	-	-	LOOP	OUT2	OUT1	RTS	DTR
Domyślnie	0	0	0	0	0	0	0	0

Rejestr pozwala kontrolować część linii sterujących:

- DTR (bit 0) – kontroluje sygnał DTR
- RTS (bit 1) – kontroluje sygnał RTS
- OUT1 (bit 2) – w niektórych sterownikach włącza port szeregowy, dlatego najlepiej wpisywać tu zawsze 1
- OUT2 (bit 3) – wpisanie 1 włącza generowanie przerwań
- LOOP (bit 4) – włącza lokalną pętlą, wyłącza wszystkie wyjścia, wszystkie wysłane dane zostaną zwrócone na wejście

LSR Line Status Register (+5)								
Bit	7	6	5	4	3	2	1	0
Nazwa	FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR
Domyślnie	0	1	1	0	0	0	0	0

Rejestr informuje o stanie odbiornika, nadajnika i błędach transmisji:

- Data Ready (bit 0) – układ odebrał dane, zerowane gdy rejestr i kolejka odbiorcza są już puste
- Overrun Error (bit 1) – utracono dane
- Parity Error (bit 2) – błąd tranismisji
- Framing Error (bit 3) – błąd ramkowania, brakujący bit stopu
- Break Indicator (bit 4) – RXD jest w stanie niskim dłużej niż okres jednego słowa
- Transmitter Holding Register Empty (bit 5) – nowe dane mogą być zapisane do rejestru wyjściowego
- Transmitter Empty (bit 6) – nie przebiega żadna transmisja
- FIFO Error (bit 7) – co najmniej jeden znak z kolejki wejściowej został odczytany z błędem

MSR Modem Status Register (+6)								
Bit	7	6	5	4	3	2	1	0
Nazwa	DCD	RI	DSR	CTS	DDCD	TERI	DDSR	DCTS
Domyślnie	x	x	x	x	0	0	0	0

Rejestr określa stan wejść kontrolnych:

- Delta CTS (bit 0) – stan CTS zmienił się od ostatniego czytania portu
- Delta DSR (bit 1) – stan DSR zmienił się od ostatniego czytania portu
- TERI (bit 2) – stan RI zmienił się z niskiego na wysoki (+12V na -12V)
- Delta DCD (bit 3) – stan DCD zmienił się od ostatniego czytania portu
- CTS (bit 4) – aktualny stan CTS
- DSR (bit 5) – aktualny stan DSR
- RI (bit 6) – aktualny stan RI
- DCD (bit 7) – aktualny stan DCD

Przy włączonej pętli lokalnej, CTS określa stan wyjścia RTS, DSR określa stan wyjścia DTS, RI określa stan wyjścia OUT1, DCD określa stan wyjścia OUT2.

4.4. Sekwencje VT100

4.4.1. Kody sterujące ASCII

Nazwa	*Mnemonik*	*Kod*	*Znaczenie*
Null	NUL	0	Ignorowany
Enquire	ENQ	5	Wysyła odpowiedź (nieobsługiwany)
Bell	BEL	7	Generuje dźwięk (nieobsługiwany)
Backspace	BS	8	Przesuwa kursor w lewo, zatrzymuje się na lewej stronie ekranu
Horizontal Tab	HT	9	Przesuwa kursor do następnej tabulacji, zatrzymuje się na końcu linii
Line Feed	LF	10	Przesuwa kursor linię w dół
Vertical Tab	VT	11	Traktowany jak LF
Form Feed	FF	12	Traktowany jak LF
Carriage Return	CR	13	Przesuwa kursor do lewej strony ekranu
Shift Out	SO	14	Wybiera zestaw znaków G0
Shift In	SI	15	Wybiera zestaw znaków G1
Device Control 1	DC1	17	Traktowany jak XON, rozpoczyna nadawanie (nieobsługiwany)
Device Control 3	DC3	18	Traktowany jak XOFF, zatrzymuje nadawanie oprócz XOFF i XON (j.w.)
Cancel	CAN	24	Jeśli otrzymany w trakcie sekwencji kontrolnej, zostaje ona anulowana.
Substitute	SUB	26	Traktywany jak CAN.
Escape	ESC	27	Początek sekwencji sterującej.
Delete	DEL	127	Ignorowane

4.4.2. Obsługiwane sekwencje VT100

Przewijanie tekstu:

- ESC [*pt* ; *pb* r ustaw region przewijania (*pt* górny wiersz, *pb* dolny)

Kursor:

- ESC [*pn* A kursor w górę *pn* razy (bez przewijania ekranu)
- ESC [*pn* B kursor w dół *pn* razy (bez przewijania ekranu)
- ESC [*pn* C kursor w prawo *pn* razy
- ESC [*pn* D kursor w lewo *pn* razy
- ESC [*pl* ; *pc* H ustaw pozycję kursora (*pl* linia, *pc* kolumna)
- ESC [*pl* ; *pc* f j.w.
- ESC [H ustaw kursor na początku ekranu
- ESC [f j.w.
- ESC D kursor w dół (z przewijaniem ekranu)
- ESC M kursor w górę (z przewijaniem ekranu)
- ESC E następna linia (to samo co CR LF)

Zestawy znaków:

- ESC (A zestaw znaków UK jako G0
- ESC (B zestaw znaków US jako G0
- ESC (0 zestaw semigrafiki jako G0

- ESC) A zestaw znaków UK jako G1
- ESC) B zestaw znaków US jako G1
- ESC) 0 zestaw semigrafiki jako G1

Atrybuty znaków:

- ESC [0 m wyłączenie wszystkich atrybutów
- ESC [1 m podświetlenie tekstu
- ESC [4 m podkreślenie tekstu (symulowane kolorem niebieskim)
- ESC [5 m mruganie
- ESC [7 m negatyw

Usuwanie tekstu:

- ESC [0 K usuwanie do końca linii (włącznie)
- ESC [1 K usuwanie do początku linii (włącznie)
- ESC [2 K usuwanie całej linii (bez przesuwania kursora)
- ESC [0 J czyszczenie do końca ekranu (włącznie)
- ESC [1 J czyszczenie do początku ekranu (włącznie)
- ESC [2 J czyszczenie całego ekranu (bez przesuwania kursora)

Żądania (odpowiedzi):

- ESC [5 n żądanie informacji o stanie terminala (ESC [0 n)
- ESC [6 n żądanie położenia kursora (ESC [*pl* ; *pc* R)
- ESC [0 c inicjalizacja i identyfikacja terminala VT100 (ESC [? 1 ; 0 c)
- ESC Z j.w.

Inicjalizacja:

- ESC c powrót do stanu początkowego

Diody LED:

- ESC [0 q wyłącz wszystkie diody LED
- ESC [1 q włącz diodę nr 1
- ESC [2 q włącz diodę nr 2
- ESC [3 q włącz diodę nr 3
- ESC [4 q włącz diodę nr 4

Tryb aplikacji/normalny:

- ESC [? 1 h kursory w trybie aplikacji (ESC O P, ...)
- ESC [? 1 l kursory w trybie normalnym (ESC [A, ...)

4.4.3. Obsługiwane sekwencje terminala Linux

Przewijanie tekstu:

- ESC [*pn* L przesuń ekran o *pn* linii w dół
- ESC [*pn* M przesuń ekran o *pn* linii w górę

Usuwanie tekstu:

- ESC [*pn* P usuwanie *pn* znaków na prawo od kursora (włącznie)

4.4.4. Nieobsługiwane sekwencje VT100

Część sekwencji VT100 nie jest obsługiwanych. Powody tego są różne:

- niejasność opisu specyfikacji VT100,
- brak drukarki,
- brak możliwości uzyskania rozdzielczości 132x24, znaków podwójnej wysokości na standardowej karcie graficznej VGA,
- programowa realizacja terminalu – brak możliwości testów sprzętu,
- znikoma przydatność w aktualnych aplikacjach.

Przewijanie tekstu:

- ESC [? 6 h włącz region
- ESC [? 6 l wyłącz region (przewijanie całego ekranu)

Tryb aplikacji/normalny:

- ESC = klawiatura numeryczna w trybie aplikacji
- ESC > klawiatura numeryczna w trybie normalnym

Zestawy znaków:

- ESC N wybiera tryb G2 dla następnego znaku
- ESC O wybiera tryb G3 dla następnego znaku

Atrybuty linii:

- ESC # 3 podwójna wysokość (górna połowa), podwójna szerokość
- ESC # 4 podwójna wysokość (dolna połowa), podwójna szerokość
- ESC # 5 pojedyncza wysokość, pojedyncza szerokość
- ESC # 6 pojedyncza wysokość, podwójna szerokość

Tabulacja:

- ESC H ustaw tabulację w aktualnej pozycji
- ESC [0 g usuń tabulację z aktualnej pozycji
- ESC [3 g usuń wszystkie tabulacje

Drukowanie:

- ESC [0 i drukuj stronę
- ESC [1 i drukuj linię
- ESC [? 4 i wyłącz automatyczne drukowanie
- ESC [? 5 i włącz automatyczne drukowanie
- ESC [4 i wyłącz kontroler drukowania
- ESC [5 i włącz kontroler drukowania

Żądania (odpowiedzi):

- ESC [? 1 5 n żądanie informacji o stanie drukarki (ESC [? 1 0 n)

Testy:

- ESC [2 ; 1 y test po włączeniu zasilania
- ESC [2 ; 2 y test pętli
- ESC [2 ; 9 y test po włączeniu, aż trafi na błąd lub wyłączenie zasilania
- ESC [2 ; 10 y test pętli, aż trafi na błąd lub wyłączenie zasilania
- ESC # 8 test obrazu (wypełnienie ekranu znakami „E”)

Funkcje konfiguracji:

- ESC [? 2 l włącz tryb emulacji VT52
- ESC < wyjdź z trybu emulacji VT52
- ESC [? 3 h tryb 132-kolumnowy
- ESC [? 3 l tryb 80-kolumnowy
- ESC [? 4 h płynne przewijanie
- ESC [? 4 l skokowe przewijanie
- ESC [? 5 h czarne znaki na białym tle
- ESC [? 5 l białe znaki na czarnym tle
- ESC [? 7 h zawijanie do nowej linii
- ESC [? 7 l zawijanie wyłączone
- ESC [? 8 h włącz automatyczne powtarzanie klawiatury
- ESC [? 8 l wyłącz automatyczne powtarzanie klawiatury
- ESC [? 9 h 480 linii ekranowych
- ESC [? 9 l 240 linii ekranowych
- ESC [? 1 8 h włącz wysuwanie strony przy drukowaniu
- ESC [? 1 8 l wyłącz wysuwanie strony przy drukowaniu
- ESC [? 1 9 h drukuj cały ekran
- ESC [? 1 9 l drukuj tylko region przewijania
- ESC [2 0 h LF, FF, VT, CR = CR/LF
- ESC [2 0 l LF, FF, VT = LF, CR = CR

4.4.5. Sekwencje klawiszowe terminali VT100 i Linux

	tryb normalny	tryb aplikacji	terminal
F1	ESC O P		VT100
F2	ESC O Q		VT100
F3	ESC O R		VT100
F4	ESC O S		VT100
F5	ESC [1 5 ~		Linux
F6	ESC [1 7 ~		Linux
F7	ESC [1 8 ~		Linux
F8	ESC [1 9 ~		Linux
F9	ESC [2 0 ~		Linux
F10	ESC [2 1 ~		Linux
↑	ESC [A	ESC O A	VT100
↓	ESC [B	ESC O B	VT100
→	ESC [C	ESC O C	VT100
←	ESC [D	ESC O D	VT100
PgUp	ESC [5 ~		Linux
PgDn	ESC [6 ~		Linux
Ins	ESC [2 ~		Linux
End	ESC [3 ~		Linux

- Home ESC [H Linux
- End ESC [F Linux

Klawiatura numeryczna w trybie aplikacji bez włączonego NumLock:

- 0 ESC O p
- 1 ESC O q
- 2 ESC O r
- 3 ESC O s
- 4 ESC O t
- 5 ESC O u
- 6 ESC O v
- 7 ESC O w
- 8 ESC O x
- 9 ESC O y
- 8 ESC O z
- – ESC O m
- , ESC O l
- . ESC O n
- Return ESC O M

Klawiatura numeryczna w trybie aplikacji nie jest obsługiwana przez terminal. Opis sekwencji dołączono jedynie dla kompletności.

5. Możliwości rozwoju

Ze względu na ograniczone ramy czasowe w projekcie nie udało się zrealizować pewnych kwestii, które choć nie są niezbędne do pracy, mogłyby ułatwić pracę lub uatrakcyjnić sam program:

- zestaw znaków ISO-8859-2 i obsługa polskiej klawiatury,
- obsługa innych terminali jak VT220, xterm itp,
- sprzętowa (RTS-CTS, DTR-DSR) i programowa (XON-XOFF) kontrola przepływu,
- obsługa bufora FIFO portu szeregowego,
- konfiguracja opcji terminalu takich jak echo lokalne, konwersja CR → CRLF,

6. Literatura

- Dokumentacja terminalu VT100
  http://www.vt100.net/

- The Serial Port – dokumentacja portu szeregowego komputera PC
  http://colargol.edb.tih.no/~bardj/