AKADEMIA GÓRNICZO - HUTNICZA
im. STANISŁAWA STASZICA w KRAKOWIE

PRACA DYPLOMOWA

Temat:

„Program obsługujący elektrokardiograf w trybie polikardiograficznym”

Wykonał:
Piotr Zieliński

Promotor:
dr inż. Piotr Augustyniak

Kraków 2001

Spis treści:

1 Cel i zawartość pracy.

Elektrokardiograf jest urządzeniem samodzielnym, którego głównym przeznaczeniem jest rejestracja sygnałów elektrycznych powstających w trakcie pracy serca. Dodatkowo dzięki specjalnym przystawkom można rejestrować sygnały innego rodzaju jak np.: sygnał fonokardiograficzny oraz puls serca. Rejestracja różnego rodzaju sygnałów jednocześnie nazywa się badaniem polikardiograficznym. Rejestracja sygnałów odbywa się na papierze przy pomocy drukarki wbudowanej w urządzenie. Urządzenie wykonuje automatyczny pomiar i analizę zarejestrowanych sygnałów elektrokardiograficznych. Sterowanie kardiografem odbywa się za pomocą wbudowanej klawiatury. Aparat posiada również wbudowany wyświetlacz ciekłokrystaliczny, który zawiera informacje o aktualnym stanie urządzenia.

Kardiograf posiada złącze szeregowe przeznaczone do współpracy z komputerem. Celowe jest więc napisanie programu, który będzie komunikował się z kardiografem. Okazuje się, że komunikacja z komputerem znacznie rozszerza możliwości urządzenia.

Celem tej pracy było opracowanie i implementacja programu sterującego elektrokardiografem AsCARD_3, pracującego w trybie polikardiograficznym (EKG + puls + fonokardiogram). Program miał umożliwić ponadto graficzną prezentację sygnałów oraz ich zapis na dysk.

Program współpracujący z elektrokardiografem został już wcześniej napisany i znajduje się w pracowni elektrokardiografii, nie posiada on jednak możliwości rejestracji próbek w trybie polikardiograficznym i został napisany dla starej wersji systemu operacyjnego: *Microsoft Windows 3.11*

Program opracowany w ramach niniejszej pracy daje następujące korzyści wynikające ze współpracy z komputerem:

- Możliwość ciągłej obserwacji pracy serca. Program w czasie rzeczywistym wyświetla wykres z trzech kanałów jednocześnie. Mogą to być sygnały elektryczne odbierane w trzech punktach ciała pacjenta jak i trzy sygnały różnego rodzaju : EKG, puls i fonokardiogram odbierane w trybie polikardiograficznym. Sygnały przekazywane są z częstotliwością 300 próbek na sekundę i posiadają 9-bitową rozdzielczość.

- Ciągła archiwizacja sygnałów przekazywanych z rejestratora na dysku komputera. Daje to możliwości komputerowej obróbki próbek przy pomocy innego programu jak i późniejszego wyświetlenia ich na monitorze. Co sekundę zapisywanych jest 1800 bajtów danych, co daje w ciągu godziny plik wielkości ok. 6,5 MB. Całkowity czas rejestracji jest ograniczony jedynie dostępnymi zasobami komputera.

- Zdalne sterowanie pracą urządzenia. Operator kardiografu ma możliwość pełnej kontroli nad urządzeniem bez odchodzenia od komputera. Zawartość wyświetlacza LCD, która pokazuje stan urządzenia jest przekazywana na bieżąco do komputera, natomiast z komputera do kardiografu przekazywane są kody klawiatury sterujące urządzeniem. Sterowanie aparatem odbywa się przy pomocy myszki oraz klawiatury.

- Istnieje ponadto możliwość wykorzystania funkcji opracowanego programu jako biblioteki procedur obsługi elektrokardiografu. Możliwe jest zatem ich użycie w zewnętrznym programie nadzorującym automatyczne badanie EKG (np. próbę wysiłkową)

Zawartość pracy

Najważniejszym elementem tej pracy jest program *EKG* przeznaczony do pracy na komputerze PC w systemie operacyjnym *Microsoft Windows 95/98/NT/2000*. Program napisany został w języku C++ przy pomocy programu *C++ Builder 4.0* firmy *Borland.* Program współpracuje z elektrokardiografem poprzez port szeregowy i wykonuje wymienione powyżej zadania.

Poniżej znajduje się opis głównych rozdziałów niniejszego dokumentu.

Do napisania programu potrzebne były wiadomości na temat komunikacji szeregowej, które znajdują się **w rozdziale 2.** Opisane są tutaj takie zagadnienia jak właściwości i możliwości interfejsu RS-232, opis przepływu danych, opis linii używanych w transmisji, sposoby łączenia urządzeń oraz inne kwestie wraz zaletami i wadami tego sposobu komunikacji.

W rozdziale 3 znajdują się niezbędne informacje o kardiografie. Na początku rozdziału zamieszczony został skrócony opis możliwości oraz obsługi kardiografu. Informacje te zostały zaczerpnięte z instrukcji obsługi aparatu. Na końcu rozdziału znajduje się szczegółowy opis transmisji między kardiografem i komputerem. Przedstawione są tu parametry transmisji szeregowej, rodzaje i schemat przesyłanych danych oraz opis zachowania kardiografu w określonych sytuacjach. Dokładne zapoznanie się z tym opisem było niezbędne do napisania programu.

Szczegółowy opis programu zamieszczony został **w rozdziale 4** niniejszej pracy i stanowi jej najważniejszą część. Na początku rozdziału przedstawione są przyczyny wyboru języka programowania oraz kompilatora. Następnie znajduje się opis kodu źródłowego programu z uwzględnieniem podziału na moduły. Przy opisie poszczególnych modułów zamieszczony został kod źródłowy ważniejszych funkcji wraz ze szczegółowym jego opisem. Na końcu rozdziału znajduje się opis obsługi programu.

Rozdział 5 zamyka pracę i zawiera podsumowanie jej osiągnięć. Omówiono w nim także problemy, które pojawiły się w trakcie konstrukcji i uruchamiania oprogramowania oraz sposoby ich rozwiązania.

2 Opis interfejsu RS-232

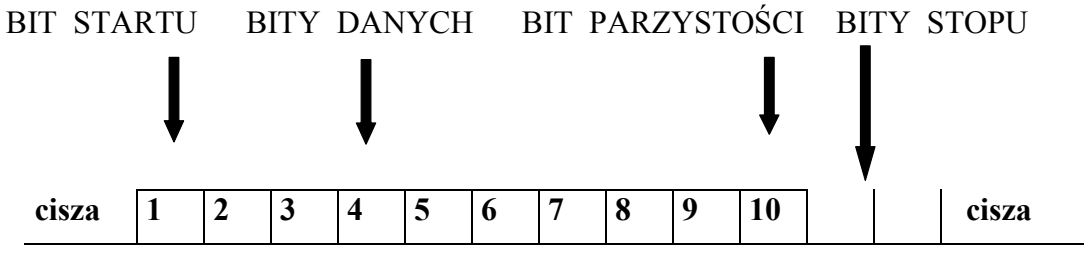
2.1 Podstawowe informacje

Standard **RS-232** (Recommended Standard) został wprowadzony w 1962 roku. Został on opracowany w celu normalizacji interfejsu pomiędzy *urządzeniem końcowym dla danych* (**DTE**- Data Terminal Equipment) a *urządzeniem komunikacyjnym dla danych* (**DCE**- Data Communication Equipment) przez amerykańskie stowarzyszenie **EIA** (Electronic Industries Assocation). W sierpniu 1969 roku wprowadzono zrewidowaną normę oznaczoną RS-232C, która reprezentuje powszechnie akceptowany sposób transmisji danych na nieduże odległości (do 15m) , z szybkością do 115 kbitów/s. Standard RS-232C został później przyjęty w Europie przez komitet normalizacyjny **CCITT** i otrzymał oznaczenie **V24**. W komputerach osobistych stał się on standardem łącza szeregowego. W systemach operacyjnych portom takim przyznano nazwy logiczne COMn (gdzie n oznacza numer portu). Oprócz obsługi modemów , interfejs umożliwia podłączenie takich urządzeń jak mysz czy drukarka. Może posłużyć również do połączenia bezpośrednio dwóch komputerów (przez tzw. Kabel Null-Modem)

Transmisja odbywa się szeregowo asynchronicznie, tzn. informacja jest przesyłana w "paczkach' o z góry określonym formacie, wyposażonych w sygnały początku i końca paczki.

Czas trwania pojedynczego bitu nazywany jest odstępem jednostkowym. Jego odwrotność określa szybkość transmisji w bodach (bitach na sekundę). Typowe prędkości transmisji wynoszą : 300,1200,2400,9600,14400,28800,56600,115000. Długość pola danych może wynosić od 5 do 8 bitów. Opcjonalny bit kontroli transmisji może pracować według jednej z dwóch zasad: kontrola parzystości (even parity) lub kontrola nieparzystości (odd parity). Transmisje kończą jeden lub dwa bity stopu.

Schemat ramki danych



Kontrola transmisji
Bit kontrolny jest najczęściej *bitem parzystości,* którego stan określa się według jednej z dwóch zasad:
- kontrola parzystości (even parity)
- kontrola nieparzystości (odd parity)

Kontrola parzystości polega na sprawdzeniu ilości jedynek na polu danych i ustawieniu bitu kontrolnego na "1", w przypadku nieparzystej ilości jedynek, lub na "0" w przypadku parzystej ilości jedynek.
Przy **kontroli nieparzystości** bit zabezpieczający ustawia się na jeden, w przypadku parzystej ilości jedynek w polu danych, lub na zero, jeżeli ilość jedynek jest nieparzysta. Bit kontroli parzystości pozwala wykryć fakt przekłamania znaku w polu danych, pod warunkiem, że liczba przekłamań jest nieparzysta.

Parametry elektryczne
Poziomy logiczne interfejsu różnią się od poziomów zastosowanych wewnątrz komputera. Zastosowano napięcia wyższe i o różnym znaku. Logiczna '1' to napięcie z zakresu -3 do –15[V] dla wejść i –5 do –15[V] dla wyjść. Logiczne '0' to napięcie od +3 do +15[V] dla wejść i +5 do +15[V] dla wyjść.
Zwiększenie wartości napięć i zastosowanie napięć o obydwu znakach pozwoliło zwiększyć odporność sygnału na zakłócenia i przesyłać informacje na większe odległości. Należy zauważyć, że w przypadku użycia zbyt długich przewodów poziomy napięć spadają poniżej dozwolonych wartości. Dodatkowo pojemności przewodów wpływają na jakość sygnału wygładzając przejścia napięć dodatnich w ujemne. Standard RS-232-C nie przewiduje jednak połączeń na duże odległości, za maksymalną odległość dzielącą połączone urządzenia uważa się 50 stóp przy korzystaniu ze standardowych przewodów.

Typy urządzeń
Ponieważ interfejs RS232 wywodzi się od modemów, przy jego opisie stosowana jest terminologia związana z transmisją informacji pomiędzy komputerami przy użyciu modemów. Stąd też wyróżnia się dwa rodzaje urządzeń:
urządzenia **DTE** (Data Terminal Equipment) będące końcowymi urządzeniami dla przesyłanej informacji (np. komputer)
urządzenia **DCE** (Data Communication Equipment) pośredniczące między urządzeniem DTE a siecią telekomunikacyjną (modem)

2.2 Magistrala interfejsu.

W interfejsie RS-232C najczęściej stosowane jest 25-stykowe złącze szufladkowe typu CANNON DB-25P lub DB-25S. Często spotyka się również 9-stykowe złącze szufladkowe typu DB-9, na które wyprowadzono tylko najważniejsze sygnały przeznaczone do asynchronicznej transmisji metodą start-stop.

Topografia wyprowadzeń sygnałów dla łącza RS-232C

Wtyk DB-25	Wtyk DB-9	Nazwa sygnału	Kierunek sygnału
1	-	Masa ochronna	-
2	3	Dane wysyłane (TxD)	wy
3	2	Dane otrzymywane (RxD)	we
4	7	Gotowość wysłania danych (RTS)	wy
5	8	Gotowość przyjęcia danych (CTS)	we
6	6	Gotowy zestaw danych (DSR)	we
7	5	Masa sygnałowa (SG)	-
8	1	Wybrany (RLSD)	we
20	4	Urządzenie gotowe (DTR)	wy
22	9	Sygnał dzwonienia (RI)	we

dodatkowo poza sygnałami wyszczególnionymi wyżej:
9 - napięcie testowe dodatnie
10 - napięcie testowe ujemne
11 STF – Select Transmit Channel
12 S.CD – Secondary Carrier Detect
13 S.CTS – Secondary Clear to Send
14 S.TxD – Secondary Transmit Data
15 TC – Transmission Signal Element Timing
16 S.RxD – Secondary Receive Data
17 RC – Receiver Signal Element Timing
18 LL – Local Loop Control
19 S.RTS – Secondary Request to Send
21 RL – Remote Loop Control (jakość sygnału odbieranego)
23 CH/CI - wybór szybkości transmisji
24 XTC (w innych źródłach TSTE)
25 BUSY (w innych źródłach Test Mode)

W magistrali interfejsu RS-232C można wyróżnić kilka grup linii:
- linie danych,
- linie sterujące,
- linie synchronizacji,
- linie masy.

Linie danych

Do dwukierunkowego przesyłania danych przeznaczone są 4 linie. Dwie z nich (TxD, RxD) tworzą kanał podstawowy, dwie pozostałe (STxD, SRxD) - kanał powrotny, nie wykorzystywany w przypadkach bezpośredniej współpracy komputer-terminal. Funkcje linii kanału podstawowego są następujące:

TxD (2) - *dane nadawane*. Linia wykorzystywana jest do przesyłania danych przez DTE. Standard wymaga, aby w odstępach między przesyłanymi danymi linia była w stanie logicznym "1 ". Transmisja może odbywać się tylko wtedy, gdy aktywne są sygnały CTS. DSR, DTR i RTS .
RxD (3) - *dane odbierane*:- Linia wykorzystywana jest do przesyłania danych przez DCE. Linia ta powinna być w stanie logicznym "1", gdy linia DCD jest w stanie pasywnym. W przypadku jednej linii transmisyjnej między urządzeniami, po której oba urządzenia mogą przesyłać dane (oczywiście nie równocześnie), linia RxD powinna być w stanie "1", gdy aktywny jest RTS.

Linie sterujące

W tej grupie linii najistotniejsze ze względu na połączenie komputer-terminal są linie przekazujące sygnały gotowości urządzeń do pracy (DSR. DTR) oraz sygnały gotowości do transmisji (RTS, CTS)
DSR (6) - *gotowość DCE*. Stan aktywny oznacza gotowość DCE do współpracy, tzn. gotowość do wymiany dalszych sygnałów sterujących w celu dokonania transmisji danych. Nie oznacza to jednak, że istnieje gotowy zbiór danych, które DCE chce przesłać do DTE. Jest to jedynie informacja o braku przeszkody do transmisji (nie zaistniał żaden defekt).
DTR (20) - *gotowość DTE*. Stan aktywny oznacza gotowość DTE do współpracy z DCE, rozumianej podobnie jak dla linii DSR.
RTS (4) - *żądanie nadawania*. Stan aktywny tej linii oznacza, że DTE zgłasza do DCE żądanie wysyłania danych. Powoduje to załączenie przez DCE sygnału CTS. Dane nie mogą być przesyłane, jeśli nie została aktywnie ustawiona linia CTS. Po przejściu sygnału RTS w stan pasywny nie powinien on zostać powtórnie załączony, dopóki DCE nie wycofa aktywnego sygnału CTS.
CTS (5) - *Gotowość do nadawania*. Za pomocą tej linii DCE może zgłaszać do DTE swoją gotowość do odbioru danych z DTE. Przy bezpośredniej współpracy terminal-komputer stan aktywny tej linii oznacza gotowość komputera do przyjmowania danych, a przy pracy z modemem - gotowość do nadawania przez DCE informacji do odległego urządzenia DCE. Linie RTS i CTS mają swoje odpowiedniki wśród linii kanału powrotnego (SRTS i SCTS), nie wykorzystywanego przy bezpośredniej współpracy komputer-terminal.
Na uwagę zasługują również dwie linie związane z poprawnością sygnałów odbieranych: DCD i CG (ozn. zgodnie z RS-232C).
DCD (8) - *poziom sygnału odbieranego*. Linia ta jest wykorzystywana zasadniczo tylko przy współpracy z modemem. Stan aktywny jest generowany przez modem (DCE) i przekazywany do DTE w przypadku odebrania przez modem poprawnego sygnału częstotliwości nośnej z kanału transmisyjnego, co oznacza, że sygnał odbierany z tego kanału przez DCE znajduje się w zakresie wartości prawidłowych. Przy współpracy komputera z terminalem (urządzeniem) może jednak zachodzić konieczność załączenia stanu aktywnego na tej linii.
Linia DCD (ozn. również RLSD) ma swój odpowiednik w kanale powrotnym: SRLSD (12).
CG (21) - *jakość sygnału odbieranego*. Linią tą są przekazywane dla DTE informacje o jakości sygnału odbieranego z linii transmisyjnej przez DCE. Stan aktywny linii CG informuje, że transmisja danych prawdopodobnie odbywa się bez błędów. Natomiast stan nieaktywny oznacza, że istnieje duże prawdopodobieństwo przekłamania danych.
Pozostałe linie sterujące nie są w zasadzie wykorzystywane w systemach pomiarowych. Dotyczą one wyboru szybkości transmisji przez DTE lub DCE (ozn. CH/CI zgodnie z RS-232C) oraz wskaźnika wywołania RI.

CH/CI (23) - *wybór szybkości transmisji*. Linia CH/CI służy do wyboru prędkości transmisji między dwiema szybkościami, które mogą być dostępne w przypadku transmisji synchronicznej lub dwoma zestawami prędkości, które mogą być dostępne w przypadku transmisji asynchronicznej. Linia ta może być sterowana albo przez DTE albo przez DCE. DCE może określać szybkość transmisji DTE, natomiast DTE - szybkość transmisji (nadawania i odbioru) realizowanej przez DCE. Jeśli źródłem sygnału jest DTE, a odbiorcą DCE, to linia nosi nazwę CH; w przeciwnym przypadku - CI. Stan aktywny sygnału oznacza wybój większej z dwu dostępnych prędkości lub większego z dwu dostępnych zestawów.
RI (22) - *wskaźnik wywołania*. Za pomocą tej linii DCE może informować DTE o tym, czy odbiera sygnał z oddalonego DCE.

Linie synchronizacji

Przesyłanie informacji po liniach danych może być realizowane asynchronicznie lub synchronicznie. Przy transmisji synchronicznej wykorzystuje się linie podstawy czasu, którymi przesyłane są tzw. sygnały zegarowe.
Na złączu 25-stykowym występują trzy linie tego typu, oznaczone zgodnie z RS-232C jako:
DA (24) - podstawa czasu z DTE dla elementów nadawanych,
DB (25) - podstawa czasu z DCE dla elementów nadawanych,
DD (17) - elementowa podstawa czasu wytwarzana w DCE.
Pierwsza z nich sterowana jest przez DTE, pozostałe - przez DCE. Linie te umożliwiają:
- nadawanie przez DTE danych linią TxD w rytmie własnego zegara (z wykorzystaniem linii DA),
- nadawanie przez DTE danych linią TxD w rytmie zegara pochodzącego z DCE (z wykorzystaniem linii DB),
- odbieranie przez DTE danych z linii RxD w rytmie zegara DCE (z wykorzystaniem linii DD)

Przy nadawaniu danych przez DTE nie jest oczywiście możliwe wykorzystanie jednoczesne linii DA i DB.
Na liniach DA i DD przejście ze stanu "1", do stanu "0" powinno wskazywać środek bitu nadawanego odpowiednio przez DTE i DCE, i w ten sposób określać optymalny moment próbkowania bitu przez urządzenie odbierające. Na linii DB przejście ze stanu "0" do "1" powinno wskazywać chwilę, w której urządzenie nadające informację ma zacząć przekazywać nowy bit, czyli określać dla urządzenia nadającego rytm wysyłania bitów.
Wykorzystując linię DA do synchronizacji transmisji należy generować na niej sygnały zegarowe (na przemian "0" i "I" przez równe odcinki czasu) co najmniej wtedy, gdy aktywna jest linia DSR (6) informująca o gotowości DCE do współpracy z DTE. Korzystanie z linii DB i DD wymaga zapewnienia istnienia na nich sygnałów zegarowych w przypadku aktywnego stanu linii DCD, sygnalizującego poprawność sygnału odbieranego.
Linie synchronizacji zwykle nie są konieczne przy transmisji szeregowej między komputerem a przyrządem pomiarowym.

Linie masy

W interfejsie RS-232C rozróżnia się dwie masy:
PG (1) - masa ochronna,
SG (7) - masa sygnałowa.
Pierwsza z nich jest masą zabezpieczającą (ang. Protective Ground), łączoną z obudową urządzenia; druga - masą sygnałową (ang. Signal Ground), stanowiącą odniesienie dla wszystkich pozostałych sygnałów interfejsu.

2.3 Łączenie urządzeń

Występują dwa podstawowe problemy związane z transmisją RS-232.

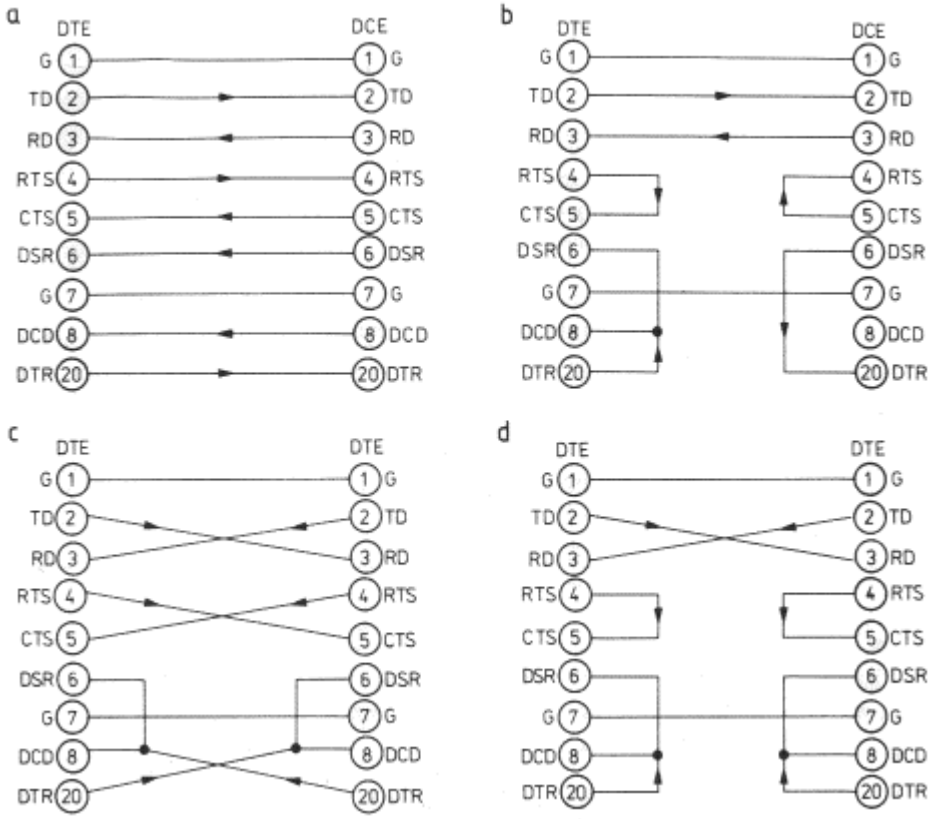
a) Z punktu widzenia złącza zdefiniowano dwa rodzaje urządzeń w ten sposób, że wejścia urządzenia jednego rodzaju odpowiadają wyjściom urządzenia drugiego rodzaju i na odwrót. praktyce można więc chcieć połączyć ze sobą urządzenia jednakowe lub urządzenia komplementarne.
b) Zdefiniowano pięć sygnałów potwierdzenia (ang. handshaking signals). Niektóre urządzenia wysyłają je i oczekują odpowiedzi na te sygnały, natomiast inne ignorują sygnały handshakingu prowadzane do ich wejść i same ich nie wysyłają. Aby uporać się z tymi problemami, trzeba zrozumieć wszelkie ich niuanse.

RS-232 wprowadzono w celu znormalizowania połączeń między urządzeniami typu DTE (data terminal equipment") i urządzeniami typu DCE ("data communication equipment") Terminal jest zawsze urządzeniem DTE, natomiast modem jest zawsze urządzeniem DCE.
Niestety, inne urządzenia, miedzy innymi mikrokomputery, mogą być jednym i drugim. IBM PC jest urządzeniem typu DTE wyposażonym w męską część złącza, natomiast większość dużych komputerów zachowuje się jak urządzenie typu DCE. Połączenie DTE z DCE jest rozumiane jako złożenie ze sobą obu połówek złącza DB-25 (które mogą być zarówno męskie, jak i żeńskie, i to po obu stronach!). Przy odrobinie szczęścia może się zdarzyć, że łącze będzie działać. Ten łut szczęścia jest potrzebny ze względu na to, że nie wiemy, które linie handshakingu i w jaki sposób wykorzystuje jedno i drugie urządzenie.

Oprócz problemów sprzętowych mogą tu jeszcze odegrać swoją rolę problemy z oprogramowaniem, tzn. nawet po pomyślnym rozwiązaniu problemu z kablem trzeba uzgodnić szybkość transmisji, sposób reakcji na bit parzystości itp. Przy łączeniu ze sobą urządzeń tego samego typu nie można zwyczajnie złożyć obu połówek złącza razem, gdyż wtedy zostaną zwarte ze sobą wyjścia obu urządzeń: urządzenie typu DTE ma wyjście na wyprowadzeniu 2, a wejście na wyprowadzeniu 3 złącza, natomiast urządzenie typu DCE - odwrotnie. Dlatego trzeba zastosować specjalny kabel (nazywany "**modemem zerowym**"), w którym skrzyżowano przewody prowadzące do wyprowadzeń 2 i 3. Niestety, nie wyczerpuje to koniecznych w tym przypadku działań.

Gdyby wszystkie urządzenia pracujące według standardu RS-232 wysyłały i odbierały cały komplet sygnałów, do poprawnej pracy łącza wystarczałoby proste połączenie urządzenia DTE z urządzeniem DCE lub skrzyżowane połączenie urządzeń DTE z DTE lub DCE z DCE. Jeśli jednak połączy się urządzenie całkowicie ignorujące handshaking z urządzeniem oczekującym na sygnały potwierdzeń, transmisja danych nie jest możliwa. Tak więc trzeba przyjąć strategię postępowania odpowiadającą rzeczywistości, co czasami wymaga użycia podstępu. Na rysunku pokazano, w jaki sposób należy wykonać kable, które umożliwią transmisję w każdej sytuacji (ściślej mówiąc, prawie w każdej sytuacji). Połączenia przedstawione na **rys. 1a** (na następnej stronie) dotyczą urządzeń DTE i DCE, wymagających pełnego handshakingu. RTS i CTS stanowią jedną parę linii potwierdzeń, a DTR i DSR drugą. Taki sam przypadek, dotyczący dwóch urządzeń z pełnym handshakingiem, lecz obu typu DTE, przedstawiono na **rys. 1c**. Tym razem trzeba użyć "**modemu zerowego**" w celu skrzyżowania odpowiednich linii sygnałowych dwu urządzeń tego samego typu. Kabel wykonany według tego rysunku będzie również odpowiedni do łączenia dwóch urządzeń typu DCE, przy czym dla zachowania porządku należałoby na rysunku odwrócić kierunki strzałek i usunąć połączenia z wyprowadzeniami 8. Kable te nie będą pracować w przypadku, gdy

jedno urządzenie wymaga doprowadzenia sygnałów potwierdzających, a drugie ich nie wysyła. Najprostszym sposobem rozwiązania problemu sygnałów potwierdzających jest takie wykonanie kabla, aby każde urządzenie wysyłało i odbierało własne sygnały handshakingu, czyli aby samo sobie zezwalało na transmisję. Jak to należy zrobić, pokazano na **rys. 1b** dla pary urządzeń DTE i DCE oraz na **rys. 1d** dla . pary urządzeń DTE, DTE (taki kabel jest również dobry dla pary DCE, DCE, lecz należy usunąć połączenia z wyprowadzeniami 8).



Rys 1a, 1b, 1c, 1d : Kable połączeniowe RS-232.

2.4 Tryby pracy

Wyróżnić możemy 3 tryby pracy:

- Simpleks - transmisja odbywa się tylko w jednym kierunku
- Półdupleks - transmisja odbywa się w obu kierunkach na przemian
- Dupleks (Pełny dupleks) - transmisja odbywa się w obu kierunkach jednocześnie

2.5 Kontrola przepływu danych

Kontrolę przepływu możemy podzielić na :

- Sprzętową - za pomocą sygnałów RTS/CTS - urządzenie, które nie może chwilowo odebrać danych (np. ma pełny bufor odbiorczy) dezaktywuje sygnał CTS, co powoduje zatrzymanie transmisji kolejnych znaków, do czasu ponownego przejścia sygnału CTS w stan aktywny. Wykorzystywane są wszystkie opisane sygnały.
- Programową - protokół XON/XOFF - urządzenie aby wstrzymać lub wznowić transmisję wysyła do urządzenia transmitującego specjalny znak sterujący. Przy takim rozwiązaniu do transmisji wystarczą tylko linie TxD i RxD , lecz transmisja musi się odbywać w trybie pełnego dupleksu.

2.6 Podsumowanie

Interfejs RS-232c powstał dość dawno, lecz dzięki dobrym parametrom i sporej popularności przetrwał do dziś. Popularność tą zawdzięcza przede wszystkim mikrokomputerom klasy PC, ich niskiej cenie oraz dostępności. Jest on wykorzystywany do komunikacji mikrokomputera z urządzeniami zewnętrznymi takimi jak modem, terminal, mysz, drukarka. Dzięki temu, że do poprawnej komunikacji wystarcza trójżyłowy przewód, prostota i niska cena układów sprawiły, że interfejs RS-232 znalazł zastosowanie w prostych systemach pomiarowych składających się z kontrolera i przyrządu, jak również w rozproszonych systemach pomiarowych (szczególnie z czujnikami inteligentnymi), w których istnieje konieczność przesyłania danych na większe odległości.

Interfejs RS-232 ma jednak dużo wad. Najważniejsze to mała prędkość (do 115 kb/s), mały zasięg (do 15 m), niezgodność z technologią TTL, niejednolite złącza. Obecnie widać tendencję odejścia od budowy urządzeń wykorzystujących ten interfejs, na rzecz takich standardów jak USB, które zapewniają większą prędkość transmisji, ujednolicenie złącza dla różnych urządzeń oraz możliwość ich łączenia w szereg.

3 Elektrokardiograf AsCARD-3

W rozdziale tym znajdują się niezbędne informacje o kardiografie. Na początku rozdziału zamieszczony został skrócony opis możliwości oraz opis obsługi kardiografu. Informacje te zostały zaczerpnięte z instrukcji obsługi aparatu. Na końcu rozdziału znajduje się szczegółowy opis transmisji między kardiografem i komputerem. Przedstawione są tu parametry transmisji szeregowej, rodzaje i schemat przesyłanych danych oraz opis zachowania kardiografu w określonych sytuacjach. Dokładne zapoznanie się z tym opisem było niezbędne do napisania programu.

3.1 Przeznaczenie i możliwości aparatu

Aparat przeznaczony jest do wykonywania badań elektrokardiograficznych oraz polikardiograficznych. Umożliwia wykonanie elektrokardiogramu w zakresie 12 odprowadzeń (12 elektrod podłączonych do ciała pacjenta), z automatycznym pomiarem parametrów sygnału EKG oraz jego analizą. Oprócz rejestracji sygnału EKG możliwa jest także rejestracja pulsu oraz sygnału fonokardiograficznego (badanie polikardiograficzne).

Kardiograf posiada drukarkę termiczną drukującą raport zawierający przebiegi EKG, dane pacjenta i wyniki pomiarów. Kardiograf jest urządzeniem samodzielnym, obsługiwanym za pomocą własnej klawiatury. Aktualny stan aparatu wyświetlany jest na wyświetlaczu LCD.

Dodatkową możliwością aparatu jest współpraca z komputerem poprzez port RS-232. Daje to nowe możliwości takie jak obserwacja sygnału EKG na monitorze i jego zapis na dysk, przetwarzanie sygnału EKG, sterowanie aparatem za pomocą komputera, odczyt stanu aparatu na monitorze (wyświetlacz LCD). Komputerowe przetwarzanie sygnału EKG jest zagadnieniem obszernym, które przekracza ramy tej pracy, dlatego program będący treścią tej pracy nie wykonuje żadnej analizy sygnału, a jedynie wyświetla oraz zapisuje na dysk pobrane dane.

3.2 Obsługa aparatu

Funkcje klawiatury są następujące:

AUTO: - start zapisu automatycznego
MANUAL: - start zapisu ręcznego
RHYTM: - start zapisu rytmu
STOP: - zatrzymanie aktualnie trwającej rejestracji
1 mV: - wydruk cechy 1 mV
mm/mV: - zmiana czułości rejestracji
mm/s: - zmiana prędkości rejestracji
MENU: - obsługa pamięci aparatu, przełączenie trybu pracy EKG, POLI
FILTER 35: - włączenie/wyłączenie filtru zakłóceń "mięśniowych"
FILTER 50: - włączenie/wyłączenie filtru zakłóceń "sieciowych"
Klawisze alfanumeryczne :
- wprowadzanie danych pacjenta.
SPACE: - spacja
BS (BackSpace): - skasowanie poprzednio wprowadzonego znaku.
ESC: - przejście do wyższego poziomu menu, anulowanie polecenia
ENTER: - akceptacja, przejście do następnego wiersza
Strzałki (góra, dół, lewo, prawo):
- zmiana wierszy i opcji menu.
ALT: - włączenie alternatywnej funkcji innych klawiszy. Klawisz działa dopiero w połączeniu z innym wciśniętym jednocześnie klawiszem.
ALT cyfra: - wybór grupy odprowadzeń EKG
ALT litera: - generuje polskie litery
ALT AUTO, ALT MANUAL, ALT RHYTM :
- wyświetlenie w dolnym wierszu wyświetlacza parametrów stanu aparatu, odpowiadających wybranemu trybowi bez startu zapisu.
ALT STOP: - wyświetlenie aktualnego czasu, daty i częstości akcji serca.
ALT MENU: - konfiguracja aparatu
ALT mm/mV: - odwrotny kierunek zmian czułości
ALT mm/s: - odwrotny kierunek zmian prędkości zapisu
ALT SPACE: - wyświetlenie informacji o jakości sygnału EKG na poszczególnych elektrodach.
ALT 1mV: - wydrukowanie danych pacjenta.

Informacje o obsłudze kardiografu za pomocą programu EKG będącego treścią tej pracy znajdują się w rozdziale: 4.3

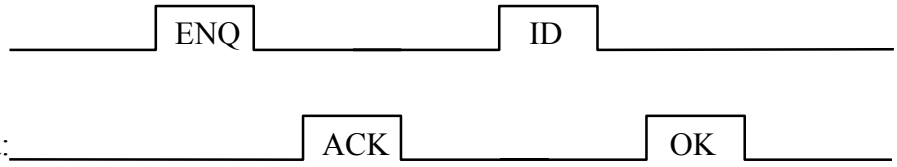
Aby uzyskać dokładniejsze informację o obsłudze kardiografu, należy zapoznać się z dołączoną do niego instrukcją obsługi.

3.3 Opis transmisji łączem RS-232

W bieżącym rozdziale omówiona została komunikacja komputer - EKG w zakresie wykorzystywanym w programie. Transmisja danych między EKG i komputerem odbywa się poprzez łącze RS-232 z następującymi parametrami:

- prędkość transmisji: 19200 bodów (bitów na sekundę)
- 8 bitów danych
- 2 bity stopu
- używany jest bit parzystości dla weryfikacji poprawności danych
- linia RTS (request to send) jest włączona

Transmisja odbywa się według ściśle określonego schematu. Informacje przekazywane są w ten sposób, że najpierw następuje nagłówek, po którym mogą wystąpić dane. Nagłówek składa się z 4 bajtów i przeprowadzany jest według następującego diagramu czasowego:



Stroną inicjującą transmisję może być zarówno komputer jak i kardiograf. Urządzenie inicjujące wysyła znak żądania transmisji ENQ (05h) i otrzymuje odpowiedź ACK (06h) oznaczającą gotowość rozpoczęcia transmisji. Strona inicjująca wysyła wtedy ID - kod identyfikacyjny trybu transmisji, która ma być rozpoczęta. Po potwierdzeniu znakiem OK (0FFh) rozpoczyna się właściwa transmisja danych.

W razie błędu transmisji, odczytu innego znaku niż oczekiwany lub po przekroczeniu limitu czasu oczekiwania na odpowiedź wysyłany jest znak BREAK (0FEh), który przerywa transmisję.

W programie używane są 4 tryby transmisji. Tryb transmisji wybierany jest przez urządzenie inicjujące poprzez wysłanie odpowiedniego znaku ID w trakcie transmisji nagłówka.

Tryby transmisji używane w programie:

1 Tryb klawiatury:

Kardiograf posiada możliwość odbioru kodów klawiszy, które traktuje na równi z kodami własnej klawiatury. Umożliwia to zdalne sterowanie aparatem. Transmisja przebiega w ten sposób, że komputer po zakończeniu nagłówka wysyła kod znaku -
(1 bajt), na który kardiograf odpowiada znakiem OK.

3 Tryb przesyłania zawartości LCD

Każda modyfikacja zawartości wyświetlacza powoduje inicjację przez kardiograf wysłania nagłówka w trybie 3, o ile łącze jest wolne i jest ustawiony znacznik gotowości (patrz tryb 5). Po wysłaniu nagłówka kardiograf wysyła 82 bajty z których 80 stanowi zawartość wyświetlacza, natomiast 2 ostatnie to pozycja kursora na wyświetlaczu. Tryb 3 nie jest inicjowany po zainicjowaniu trybu 8 (monitora)

5 Tryb testu

W tym trybie wysyłany jest tylko nagłówek. Urządzeniem inicjującym jest komputer. Przesłanie nagłówka w tym trybie powoduje ustawienie znacznika gotowości w kardiografie.

8 Tryb monitora

Tryb ten inicjowany jest przez komputer. Przesłanie nagłówka w tym trybie powoduje rozpoczęcie wysyłania przez kardiograf próbek z 3 kanałów wraz z innymi informacjami. Próbki są 9 bitowe, przesyłane po 2 w paczce z szybkością 150 razy na sekundę. Daje to 300 próbek na sekundę. Paczka składa się z 8 bajtów i posiada następującą strukturę:

Bajt	Znaczenie
0	sześć starszych bitów sześciu próbek EKG o następującej postaci : [1, 1, 1k, 2k, 3k, 1k, 2k, 3k]
1	pierwsza próbka kanału 1
2	pierwsza próbka kanału 2
3	pierwsza próbka kanału 3
4	druga próbka kanału 1
5	druga próbka kanału 2
6	druga próbka kanału 3
7	jeden z kolejnych znaków ciągu o długości 88 bajtów: [LCD – 80 bajtów...], [kolumna], [wiersz], [prędkość], [HR], [odprowadzenie 1], [odprowadzenie 2], [odprowadzenie 3], [0FFh]

Po każdej paczce komputer przesyła potwierdzenie ACK lub kod przerwania BREAK. Brak potwierdzeń przez kilka cykli powoduje zaprzestanie wysyłania próbek przez kardiograf.
Dwa najstarsze bity pierwszego bajtu paczki są zawsze równe 1. Te jedynki umożliwiają wykrycie błędu lub transmisji. Ostatni bajt paczki służy do przesyłania zawartości wyświetlacza i innych dodatkowych informacji w kolejności podanej powyżej. Końcowy znak 0FFh służy do weryfikacji synchronizacji.

4 Opis programu *EKG*.

W rozdziale tym opisany jest program *EKG* będący treścią niniejszej pracy. Na początku rozdziału przedstawione są przyczyny wyboru języka programowania oraz kompilatora. Następnie znajduje się opis kodu źródłowego programu z uwzględnieniem podziału na moduły. Przy opisie poszczególnych modułów zamieszczony został kod źródłowy ważniejszych funkcji wraz ze szczegółowym jego opisem. Na końcu rozdziału znajduje się opis obsługi programu.

4.1 Wybór języka i kompilatora.

Program napisany został w dla systemu operacyjnego *Microsoft Windows 95/98/NT/2000* w języku *C++*. O wyborze języka zadecydowały takie czynniki jak:

- duże możliwości języka
- jego olbrzymia popularność
- dostępność narzędzi do tworzenia aplikacji w tym języku.
- dobra znajomość, i doświadczenie autora w pisaniu programów w C++.

Spośród wielu dostępnych kompilatorów autor wybrał *C++ Builder* w wersji 4.0 firmy *Borland*, który został uznany za najwygodniejsze i najszybsze narzędzie do tworzenia aplikacji w C++ dla *Windows*. *Builder* należy do systemów błyskawicznego projektowania aplikacji (ang. *RAD – Rapid Application Developement*). Oznacza to w praktyce, że cały proces projektowania interfejsu użytkownika (czyli systemu okienek, menu, przycisków, dialogów) realizowany jest graficznie, poprzez rozmieszczanie odpowiednich komponentów na ekranie. Tworzenie nawet wyrafinowanych aplikacji jak na przykład przeglądarek WWW staje się kwestią minut. Dzięki takim rozwiązaniom programista nie traci wiele czasu na tworzenie interfejsu, ale zajmuje się tym co jest głównym zadaniem programu, który jednocześnie posiada elegancki wygląd.

Rozważany był również wybór środowiska *Microsoft Visual C++,* jednak środowisko to, zdaniem autora, ustępuje systemowi *Builder* jeśli chodzi o szybkość i łatwość budowy interfejsu użytkownika. *Builder* posiada łatwiejszy dostęp do właściwości poszczególnych komponentów.

4.1.1 C++ Builder i biblioteka *VCL.*

Komponenty w systemie *Builder* bazują na bibliotece klas *VCL*. Biblioteka *VCL* (*Visual Component Library*) pochodzi z systemu *Delphi* i napisana jest w języku *Object Pascal.* Do napisania jakiegokolwiek programu w systemie *Builder* wykorzystującego komponenty używana jest biblioteka *VCL*. Nie oznacza to, konieczności poznania języka *Object Pascal*, gdyż *Builder* ukrywa przed programistą jej pascalowe pochodzenie. Program w systemie *Builder* pisany jest oczywiście w języku C++.

Każdy komponent np.: formularz, przycisk, etykieta, pole edycyjne, zegar – jest obiektem odpowiedniej klasy biblioteki VCL.

Komponent posiada odpowiednie atrybuty *(properities)* definiujące jego wygląd oraz wiele innych właściwości. Atrybutem może być: pozycja na ekranie, wielkość, kolor, tekst etykiety, obraz, czas cyklu dla zegara i inne. Atrybuty te mogą być odczytywane i modyfikowane zarówno na etapie projektowania aplikacji jak i jej działania. W trakcie działania programu atrybuty widoczne są jako pola (zmienne należące do klasy) obiektu reprezentującego dany komponent.

Komponent posiada skojarzone z nim zdarzenia, którym można przypisać definiowane przez użytkownika funkcje. Zdarzenia mogą pochodzić zarówno od użytkownika jak i systemu operacyjnego *Windows*. Zdarzeniami mogą być: utworzenie obiektu, wyświetlenie komponentu, kliknięcie myszką na komponencie, zmiana tekstu w polu edycyjnym, zdarzenie cykliczne od zegara i inne.

Komponent wyposażony jest również w odpowiednie metody. Metody te często operują na atrybutach komponentu i wywoływane są w celu uzyskanie pożądanego zachowania komponentu. Na przykład dla każdego komponentu widzialnego zdefiniowane są metody *show* i *hide* powodujące odpowiednio: pojawienie się na ekranie oraz ukrycie komponentu.

4.2 Główne elementy programu.

Program składa się z następujących modułów:

- *ekg*
- *serialComm*
- *transmission*
- *formMain*
- *formHelp*
- *formPort*
- *formSampleName*

Moduł składa się pliku źródłowego **.cpp* wraz z plikiem nagłówkowym **.h*. W pliku źródłowym zawarte są definicje funkcji i zmiennych, które zadeklarowane zostały w pliku nagłówkowym. Plik nagłówkowy zawiera także deklaracje klas. Moduł kompilowany jest przez kompilator do pliku binarnego **.obj*. Skompilowane moduły łączone są za pomocą programu łączącego (*linker*) do pliku wykonywalnego **.exe*.
Moduł może być skojarzony z formularzem. Formularz definiuje okno programu, zarówno okno główne jak i wszystkie inne okienka. W opisywanym programie używane są 4 formularze: okna głównego, okna pomocy, okna dialogu do wyboru portu, oraz okna do zmiany nazwy pliku. W trakcie projektowania na formularzu umieszczane są przyciski, etykiety, pola edycyjne i inne komponenty typowe dla okienka. Informację o tych elementach przechowywane są w pliku formularza: **.dfm*. Równocześnie formularz posiada skojarzoną z nim klasę w zadeklarowaną w pliku nagłówkowym modułu. Klasa ta zawiera wskaźniki do wszystkich elementów zawartych na formularzu, oraz deklaracje do funkcji związanych z konkretnymi zdarzeniami dotyczącymi elementów formularza, np.: deklaracja funkcji obsługującej kliknięcie przycisku znajdującego się na formularzu.

4.2.1 Moduł *ekg.*

Jest to główny moduł aplikacji, automatycznie generowany przez *Builder*. Tutaj znajduje się definicja funkcji *WinMain* (głównej funkcji każdego programu w systemie operacyjnym *Windows*), która wykonuje czynności przygotowujące takie jak np.: stworzenie formularzy (okienek) używanych w programie. W module tym programista nie dokonuje żadnych zmian dlatego nie będzie on dalej omawiany.

Oto treść modułu ekg:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
USERES("ekg.res");
USEFORM("formMain.cpp", Form_Main);
USEUNIT("serialComm.cpp");
USEUNIT("transmission.cpp");
USEFORM("formPort.cpp", Form_Port);
USEFORM("formHelp.cpp", Form_Help);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
                Application->Initialize();
                Application->CreateForm(__classid(TForm_Main), &Form_Main);
                Application->CreateForm(__classid(TForm_Port), &Form_Port);
                Application->CreateForm(__classid(TForm_Help), &Form_Help);
                Application->Run();
        }
        catch (Exception &exception)
        {
                 Application->ShowException(&exception);
        }
        return 0;
}
//---------------------------------------------------------------------------
```

4.2.2 Moduł *serialComm.*

Ten moduł odpowiada za obsługę transmisji danych na poziomie portu szeregowego, i jest używany przez funkcje synchronizujące komunikację. Poniżej znajduje się deklaracja klasy *serialComm* oraz zaprzyjaźnionego z nią wątka *ReadThread*. Klasa zawiera metody służące do otwarcia i zamknięcia portu szeregowego oraz umożliwiające odczyt i zapis danych.

```
class SerialComm // klasa odpowiadająca za obsługę portu szeregowego
     {

     private:
           HANDLE hComm;          // handler portu – uchwyt pliku
           DCB dcb;               // struktura zawierająca ustawienia portu
           char portName[10];     // nazwa portu
           char comSettings[20]; // podstawowe ustawienia portu
           int thread_id;         // identyfikator wątka

     public:
           int port;           // numer portu
           String readBuffer; // bufor wejściowy portu


           bool InitializeSerialCommunication(int _port);
       // otwarcie portu i dodatkowe czynności przygotowujące do transmisji

           void CloseSerialCommunication();
           friend void ReadThread( LPVOID pParam );
              // zaprzyjaźniony wątek czytający

           BOOL SerialWrite(char * lpBuf, DWORD dwToWrite);
                                                  //zapis ciągu znaków
           BOOL send(unsigned char character);          //zapis znaku
     };


void ReadThread( LPVOID pParam );
```

opis ważniejszych zmiennych (typ zmiennej, nazwa):

HANDLE hComm – ta zmienna jest uchwytem do pliku reprezentującego port szeregowy. W systemie operacyjnym *Windows* port szeregowy jest traktowany jak zwykły plik dyskowy. Ma on nazwę COM1 lub COM2. Wykonuje się na nim takie operacje, jak na pliku z prawem odczytu i zapisu. Najpierw należy go otworzyć, a na końcu zamknąć. Wysyłanie danych odbywa poprzez zapisywanie danych do pliku, natomiast odbieranie realizowane jest poprzez odczyt z pliku.

DCB dcb – Jest to zmienna złożona (struktura). Port od strony fizycznej posiada wiele parametrów, które za pomocą funkcji *SetCommState* i z użyciem tej zmiennej można ustawić. Najważniejsze z nich to: szybkość pracy, ilość bitów danych w bajcie, ilość bitów stopu (odstęp między danymi), parzystość - czyli sposób na wykrywanie błędów transmisji.

Poniżej znajduje się opis funkcji modułu *serialComm*:

****

InitializeSerialCommunication – funkcja wywoływana przed użyciem portu. Otwiera port, ustawia jego parametry oraz uruchamia wątek odpowiedzialny za odczyt danych. W przypadku wystąpienia błędu wyświetla odpowiedni komunikat (funkcja *MessageBox*).
Dla zapewnienia współpracy z elektrokardiografem ustawiane są następujące parametry:
- prędkość transmisji: 19200 bodów (bitów na sekundę)
- 8 bitów danych
- 2 bity stopu
- używany jest bit parzystości dla weryfikacji poprawności danych
- linia RTS (request to send) jest włączona

Powyższe parametry opisane są w poprzednim rozdziale dotyczącym transmisji szeregowej. Poniżej znajduje się treść opisywanej funkcji:

```
////////////////////////////////////////////////////////////////////////////
//////////////////////InitializeSerialCommunication/////////////////////////
////////////////////////////////////////////////////////////////////////////
// parametry wejsciowe:
//     int _port : numer portu który zostanie otwarty
// zwracana wartosc:
//     true : w przypadku sukcesu
//     false: blad

bool SerialComm::InitializeSerialCommunication(int _port)
{
port=_port;
sprintf(portName,"COM%d",port);                // ustalanie nazwy
sprintf(comSettings,"COM%d:19200,e,8,2",port); // i parametrów portu


//******************* Otwarcie portu **************
hComm = CreateFile( portName,
                    GENERIC_READ | GENERIC_WRITE,
                    0,
                    0,
                    OPEN_EXISTING,
                    0,//FILE_FLAG_OVERLAPPED,
                    0);
if (hComm == INVALID_HANDLE_VALUE)
{
    String errMsg="Cannot open port : "+(String)portName;
    MessageBox(0,errMsg.c_str(),"error",0);
    return false;
} else

{
Form_Main->Caption = Form_Main->Caption + " na " + (String)portName;
}

//****** Budowa struktury DCB
FillMemory(&dcb, sizeof(dcb), 0);
dcb.DCBlength = sizeof(dcb);

if (!BuildCommDCB(comSettings, &dcb))
{
```

```
    MessageBox(0,"Couldn't build the DCB.\n Usually a problem "
                "with the communications specification string","error",0);
    return 0;
}
dcb.fRtsControl=RTS_CONTROL_ENABLE;  // RTS line ON

//****** ustawienie parametrów portu
if (!SetCommState(hComm, &dcb))
{
    MessageBox(0,"Error in SetCommState.\n"
    "Possibly a problem with the communications,\n"
    "port handle or a problem with the DCB structure itself.","error",0);
    return 0;
}

//************** uruchomienie wątka czytającego
if ((thread_id = _beginthread(ReadThread,4096,NULL)) == (unsigned long)-1)
{
    MessageBox(0,"Unable to create thread","error",0);
    return false;
}

return true;
};
```

****

CloseSerialCommunication – jedyną czynnością którą wykonuje ta funkcja jest zamkniecie pliku umożliwiającego dostęp do portu szeregowego.

```
void SerialComm::CloseSerialCommunication()
{
   //_endthread();
   CloseHandle(hComm);   // zamknięcie portu
};
```

****

send – ta funkcja wysyła do portu pojedynczy znak, wywołując funkcję *SerialWrite*

```
// parametry wejściowe:
//     unsigned char character : znak do wysłania
// zwracana wartość:
//     true : w przypadku sukcesu
//     false: błąd

BOOL SerialComm::send(unsigned char character)
{
    return SerialWrite((char *)&character,1); // wysłanie znaku
};
```

****

SerialWrite – wysyła ciąg znaków do portu szeregowego. Do napisania tej funkcji oraz wątka odczytującego wykorzystany został przykładowy program znaleziony w systemie pomocy pakietu Microsoft Visual C++. Operacje zapisu i odczytu wykonywane są w trybie *overlapped.* Nie wdając się w szczegóły tego sposobu komunikacji warto wspomnieć, że umożliwia to równoczesny odczyt i zapis, co jest wymagane w tym programie.

W przypadku użycia prostego odczytu i zapisu występuje niekorzystna sytuacja zakleszczenia odczytu i zapisu. Załóżmy, że wątek odczytujący oczekuje na znak z portu, i w tym czasie chcemy wysłać znak. Wtedy wywołanie funkcji zapisującej zostaje zatrzymane przez nie zakończoną jeszcze operację odczytu. Efekt jest taki, że nie można wysłać przez port dopóki nie zostanie z niego odczytany jakiś znak, i do tego czasu zatrzymany jest wątek, który dodatkowo blokuje funkcję zapisującą. Sposobem na rozwiązanie tego problemu jest stosowanie operacji w trybie *overlapped*, dzięki któremu następuję natychmiastowy powrót z funkcji odczytu i zapisu, wraz z zapisaniem stanu operacji do specjalnej struktury. Dzięki tej strukturze można dowiedzieć się np.: że odczyt jeszcze trwa, już się zakończył, albo że wystąpił błąd.

```
//////////////////////////////////////////////////////////////////////
/////////////////////////// SerialWrite //////////////////////////////
//////////////////////////////////////////////////////////////////////
// parametry wejściowe:
//     char * lpBuf : ciąg znaków do wysłania
//     DWORD dwToWrite : liczba znaków do wysłania
// zwracana wartość:
//     true : w przypadku sukcesu
//     false: błąd

BOOL SerialComm::SerialWrite(char * lpBuf, DWORD dwToWrite)
{
   OVERLAPPED osWrite = {0};
   DWORD dwWritten;
   DWORD dwRes;
   BOOL fRes;

   // Create this write operation's OVERLAPPED structure's hEvent.
   osWrite.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
   if (osWrite.hEvent == NULL)
      // error creating overlapped event handle
      return FALSE;


   // Issue write.
   if (!WriteFile(hComm, lpBuf, dwToWrite, &dwWritten, &osWrite))
   {
      if (GetLastError() != ERROR_IO_PENDING)
        {
          // WriteFile failed, but isn't delayed. Report error and abort.
          fRes = FALSE;
        }
      else
          // Write is pending.
          dwRes = WaitForSingleObject(osWrite.hEvent, INFINITE);
          switch(dwRes)
          {
            // OVERLAPPED structure's event has been signaled.
            case WAIT_OBJECT_0:
```

```
                if (!GetOverlappedResult(hComm, &osWrite, &dwWritten,
                                                                FALSE))
                     fRes = FALSE;
                else
                // Write operation completed successfully.
                fRes = TRUE;
                break;

            default:
                // An error has occurred in WaitForSingleObject.
                // This usually indicates a problem with the
                // OVERLAPPED structure's event handle.
                fRes = FALSE;
                break;
          }

   }
   else
      // WriteFile completed immediately.
      fRes = TRUE;

   CloseHandle(osWrite.hEvent);

return fRes;

};
```

****

ReadThread – wątek odczytujący dane z portu szeregowego do bufora *readBuffer*. Operacja odczytu wykonywana jest w trybie *overlapped*. Ten tryb odczytu został opisany przy opisie poprzedniej funkcji (*SerialWrite*).

Funkcja ta wykonywana jest jako wątek co oznacza, że wykonuje się pseudo-równolegle (czyli jednocześnie z punktu widzenia użytkownika) wraz z wykonaniem głównego programu. Oczywiście wątki mogą wykonywać się równocześnie tylko w systemie wieloprocesorowym. Na komputerze wyposażonym w 1 procesor, odbywa się w ten sposób, że wątki czekają w cyklicznej kolejce do wykonania. Na wykonanie wątka przydzielany jest od procesora krótki czas, po przekroczeniu którego jego działanie jest zawieszone i przekazywane następnemu wątkowi, do czasu kiedy znów przyjdzie kolej na ten wątek. Użycie wątka w tym programie było potrzebne ze względu na konieczność równoległego wykonywania operacji odczytu i zapisu.

Wątek pierwotnie miał należeć do klasy, jednak ze względu na trudności z kompilacją został z niej usunięty. Jest funkcją zaprzyjaźnioną klasy, co oznacza, że ma dostęp do jej prywatnych pól.

```
////////////////////////////////////////////////////////////////////////////
/////////////////////////// ReadThread /////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

void ReadThread( LPVOID pParam ) // wątek czytający
{
OVERLAPPED osReader = {0};
char c;
BOOL fWaitingOnRead = FALSE;
unsigned long bytes_read=3;
DWORD dwRes;

//*********************ReadFile************************************//
// Create the overlapped event. Must be closed before exiting
// to avoid a handle leak.

osReader.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

while (1)
    {
    if (osReader.hEvent == NULL)
        {
        MessageBox(0,"Error creating overlapped event","error",0);
        return;
        }

    if (!fWaitingOnRead)
        {
        // Issue read operation.
        if (!ReadFile(Form_Main->COM->hComm, &c, 1, &bytes_read,
                                                        &osReader))
            {
            if (GetLastError() != ERROR_IO_PENDING)  // read not delayed?
                MessageBox(0,"Error in communications\n","error",0);
            else
                fWaitingOnRead = TRUE;
            }
        else
```

```
            {
            transmissionLock->Enter();
            Form_Main->COM->readBuffer+=c; //dodaje przeczytany znak do
                                           //bufora
            transmission();
            transmissionLock->Leave();
            }
        };


    #define READ_TIMEOUT 1      // milliseconds

    if (fWaitingOnRead)
        {
        dwRes = WaitForSingleObject(osReader.hEvent, READ_TIMEOUT);

        switch(dwRes)
            {
            // Read completed.
            case WAIT_OBJECT_0:

                if (!GetOverlappedResult(Form_Main->COM->hComm,
                  &osReader, &bytes_read, FALSE))
                   MessageBox(0,"Error in communications; report
                   it\n","error",0);
                else
                   // Read completed successfully.
                   {
                   transmissionLock->Enter();
                   Form_Main->COM->readBuffer+=c;// dodaje przeczytany
                                                 // znak do bufora
                   transmission();
                   transmissionLock->Leave();
                   }

                // Reset flag so that another opertion can be issued.
                fWaitingOnRead = FALSE;
                break;

            case WAIT_TIMEOUT:
                //fWaitingOnRead flag isn't
                // changed since I'll loop back around, and I don't want
                // to issue another read until the first one finishes.
                //
                // This is a good time to do some background work.
                break;
            default:
                MessageBox(0,"Error in the WaitForSingleObject; abort \
                This indicates a problem with the OVERLAPPED structure's\
                event handle.\n","error",0);
                break;
            }; // switch

        }; //if (fWaitingOnRead)

    }; //while (1)

};
```

4.2.3 Moduł *transmission.*

Moduł nie zawiera żadnej klasy, ani nie definiuje formularza. Zawiera natomiast 2 najważniejsze dla działania programu funkcje: *transmission* realizującą transmisję z elektrokardiografem, oraz *monitor*, która przedstawia na wykresie, przekazywane na bieżąco próbki trzech kanałów EKG oraz wykonuje ich zapis na dysk.

Poniżej wypisane są stałe używane w module:

```
//...... stałe dla zmiennej transMode ..................................

#define TRANS_MODE_KEY     1      // tryb klawiatury
#define TRANS_MODE_LCD     3      // tryb przesyłania zawartości LCD
#define TRANS_MODE_TEST    5      // test
#define TRANS_MODE_MONITOR 8      // włączenie trybu monitora

//...... stałe dla zmiennej headerState ................................

#define HEADER_STATE_SEND_ENQ    0
#define HEADER_STATE_GET_ACK     1
#define HEADER_STATE_GET_OK      2
#define HEADER_STATE_DONE        3

#define HEADER_STATE_EKG_REQUEST 4
#define HEADER_STATE_EKG_LCD     5

//.... specjalne znaki używane w transmisji ..............................

#define ENQ    0x05   // żądanie transmisji
#define ACK    0x06   // akceptacja żądania transmisji
#define OK     0xFF   // akceptacja trybu transmisji
#define BREAK  0xFE   // przerwanie transmisji
```

Spośród zmiennych szczególne znaczenie przebiegu transmisji mają dwie: *transMode,* która pamięta aktualny tryb transmisji, oraz *headerState*, która odpowiada za poszczególne etapy transmisji nagłówka i danych po nim następujących. W zależności od tych zmiennych sterowanie transmisją przekazywane jest do odpowiedniego fragmentu kodu funkcji *transmission.*

Oto lista zmiennych globalnych używanych w module:

```
////////////////////////////////////////////////////////////////////////

unsigned char transMode;   // zmienna trybu transmisji
int headerState;           // zmienna stanu nagłówka

String KeyboardRequest=""; // bufor klawiatury
bool KeyAccepted;          // potwierdzenie przyjęcia klawisza
bool EnableGrid;           // przełącza wyświetlanie siatki

int idleCounter=0;        // czas nie używania łącza liczony taktami zegara
bool idleTimeOut=false;    // łącze zbyt długo nie używane, -> tryb testu
bool transmissionFlag=false;
```

```
TCriticalSection *transmissionLock; // sekcja krytyczna obejmująca
                                    // wywołanie funkcji transmission

bool requestMonitor=false;          // żądanie trybu monitora
int x=1;                            // aktualna współrzędna pozioma na
                                    // wykresie
int xMax;                           // maksymalny rozmiar wykresu

String *sampleBuffer;               // bufor na próbki
int    sampleCounter=0;             // próbki wczytane

WORD sampleSaveBuffer[300][3];      // bufor na próbki do zapisu na dysk
int  sampleSaveCounter=0;           // licznik bufora do zapisu

int sampleBufferTail=0;             // próbki przeanalizowane
int old_s1b=0,old_s2b=0,old_s3b=0;  // wartości poprzednich próbek

HANDLE hSampleFile;                 // uchwyt do pliku z próbkami
```

Poniżej znajduje się opis funkcji używanych w module:

****

transmission – Funkcja ta odpowiada za komunikację z elektrokardiografem. Wywoływana jest przez wątek czytający w przypadku odczytania znaku z portu, a także przez funkcję obsługi zegara (*timera*) w przypadku naciśnięcia klawisza lub przy długim okresie bezczynności na łączu szeregowym.

Całe ciało funkcji wypełnione jest instrukcją *switch,* która w zależności od wartości zmiennej *headerState* przekazuje sterowanie do odpowiedniego fragmentu rozpoczynającego się słowem kluczowym *case*. Następnie wykonywane są instrukcje, które przetwarzają odebrane dane i wysyłają odpowiedź. Dzieje się tak, aż do napotkania instrukcji *break,* po której następuje wyjście z funkcji. Funkcja może zmienić wartość zmiennej *headerState,* i wtedy przy następnym jej wywołaniu sterowanie zostanie przekazane do innego fragmentu funkcji.

Poniżej znajduje się opis czynności, które wykonywane są w poszczególnych sekcjach *case* funkcji *transmission*.

HEADER_STATE_SEND_ENQ – komputer rozpoczyna transmisję wysyłając znak *ENQ*, który jest pierwszym znakiem nagłówka, następnie ustawia zmienną *headerState* na wartość *HEADER_STATE_GET_ACK*.

HEADER_STATE_GET_ACK – komputer oczekuje na znak *ACK,* który jest zgodą na rozpoczęcie transmisji. Jeżeli taka odpowiedź przyszła wysyłana jest zawartość zmiennej *transMode*. Powoduje to przejście kardiografu do jednego z trzech trybów oznaczonych stałymi: *TRANS_MODE_TEST, TRANS_MODE_KEY* lub *TRANS_MODE_MONITOR*. Następnie ustawiana jest zmienna *headerState* na wartość *HEADER_STATE_GET_OK*.
Jeżeli komputer otrzyma inną odpowiedź niż *ACK* wysyła *BREAK* a następnie *ENQ* co powoduje restart transmisji.

HEADER_STATE_GET_OK – oczekiwany jest znak *OK*, który oznacza akceptację trybu transmisji przez kardiograf. W przypadku akceptacji, jeżeli bieżącym trybem jest tryb klawiatury wysyłany jest do kardiografu kod klawisza i ustawiana jest flagę informującą o oczekiwaniu na jego potwierdzenie. Zmienna *headerState* ustawiana jest na wartość *HEADER_STATE_DONE.*
W przypadku otrzymania innej odpowiedzi niż *OK* wysyłana jest sekwencja *BREAK, ENQ* oraz ustawiana jest zmienna *headerState* na wartość *HEADER_STATE_GET_ACK* co powoduje restart transmisji.

HEADER_STATE_DONE – zakończono już transmisję nagłówka i dalsze czynności zależą od trybu transmisji oraz czy odebrano jakieś znaki z kardiografu.

Jeżeli odebrano jakieś znaki, wtedy rozpatrywany jest w kolejności jeden z następujących przypadków:

- Jeżeli program znajduje się w trybie monitora, oczekiwane jest przyjście 8-bajtowej próbki, która po przyjściu jest poddawana weryfikacji (najstarsze bity pierwszego bajtu muszą być równe 1). Próbka jest dodawana do cyklicznego bufora *sampleBuffer*. Jeżeli próbka potwierdzona zostanie znakiem ACK, spowoduje to kontynuację wysyłania próbek przez kardiograf.
Potwierdzenia nie zostanie wysłane jeżeli w buforze klawiatury znajduje się klawisz oczekujący na przesłanie, co powoduje chwilową przerwę transmisji próbek, a następnie przejście do trybu klawiatury, po czym następuje powrót do trybu monitora.
Próbka przesłana do bufora *sampleBuffer* będzie przetwarzana na przez funkcję *monitor*, która dekoduje próbkę i rysuje na ekranie wykres EKG. Funkcja *monitor* jest wywoływana przez funkcję obsługi zegara *(timera)*.
Jeżeli próbka nie przejdzie weryfikacji, transmisja jest przerywana a następnie ponawiana, podobnie reaguje program w przypadku, gdy przez długi czas nie uda się skompletować ośmiu bajtów próbki.

- Jeżeli odebrano znak *ENQ ,* oznacza to, że kardiograf prześle zawartość swojego wyświetlacza LCD, którego zawartość właśnie się zmieniła. Sytuacja ta nie występuje w trybie monitora, gdzie zawartość wyświetlacza jest przesyłana w ostatnim bajcie 8-bajtowej próbki.
Odebranie znaku *ENQ* jest początkiem nagłówka, gdzie stroną inicjującą jest kardiograf. Dlatego w odpowiedzi wysyłany jest znak *ACK*, a następnie ustawiona zostaje zmienna *headerState* na wartość *HEADER_STATE_EKG_REQUEST*.

- Jeżeli odebrano znak *BREAK,* kardiograf przerwał transmisję. Program wykonuję wtedy restart transmisji z przejściem do trybu testu.

- Jeżeli odebrano znak *OK,* oznacza to, że kardiograf zaakceptował przesłany klawisz. Zostaje wtedy ustawiona flaga *KeyAccepted*, która oznacza zezwolenie na transmisję następnego klawisza.

- Jeżeli nie wystąpi, żaden z powyższych przypadków, czyli odebrano jakiś inny znak, a program nie znajduje się w trybie monitora, to taka sytuacja jest błędna i wykonywany jest restart transmisji.

Jeżeli nie odebrano żadnych znaków, rozpatrywany jest w kolejności jeden z następujących trzech przypadków:

- Jeżeli w buforze klawiatury jest znak do wysłania i ustawiona jest flaga *KeyAccepted,* wtedy program rozpoczyna transmisję nagłówka w trybie klawiatury. Polega to na ustawieniu zmiennej *transMode* na *TRANS_MODE_KEY,* zmiennej *headerState* na *HEADER_STATE_GET_ACK,* oraz wysyłaniu znaku *ENQ.*

- Jeżeli ustawiona jest flaga *idleTimeOut,* wtedy program rozpoczyna transmisję nagłówka w trybie testu i kasuje flagę *idleTimeOut*. Gdyby zaniedbać podtrzymywanie transmisji w trybie testu, kardiograf przestałby wysyłać zawartość wyświetlacza w przypadku jego zmian. Ustawiona przez funkcję obsługi zegara, flaga *idleTimeOut* oznacza, że przez długi czas (około jednej sekundy) łącze nie było używane. Po dłuższym czasie bezczynności kardiograf przerywa transmisję.

- Jeżeli ustawiona jest flaga *requestMonitor,* wtedy program rozpoczyna transmisję nagłówka w trybie monitora i kasuje flagę *requestMonitor.*

HEADER_STATE_EKG_REQUEST – komputer oczekuje na żądany tryb transmisji, po zainicjowaniu nagłówka przez kardiograf. Jedynym właściwym trybem jest w tym przypadku *TRANS_MODE_LCD*. Po odebraniu tego znaku potwierdzany jest tryb transmisji poprzez wysłanie znaku *OK* oraz ustawienie zmiennej *headerState* na *HEADER_STATE_EKG_LCD*. W przypadku otrzymania innego znaku, program wykonuje restart transmisji.

HEADER_STATE_EKG_LCD – komputer oczekuje na 82 bajty z zawartością wyświetlacza LCD. Jeżeli zostaną przysłane każdy z nich przetwarzany jest przez funkcję: *polskie*, która zamienia kody polskich liter z używanych w kardiografie na kody używane w komputerze. Następnie zawartość wyświetlacza przesyłana jest na monitor komputera.

Poniżej znajduje się cała treść funkcji *transmission.*

```
//---------------------------------------------------------------------------
void transmission(void)  // funkcja realizująca
                         // protokół transmisji EKG<->PC
{
idleCounter=0;

// Podejmowana jest odpowiednia czynność w zależności od dotychczasowego
// przebiegu transmisji oraz zdarzeń pochodzących od użytkownika
// i programu oraz odbieranych znaków z EKG.

   switch (headerState)
      {
      //-----------------------------------------------------------------

      case HEADER_STATE_SEND_ENQ:  // komputer rozpoczyna transmisję
           Form_Main->COM->send(ENQ);
```

```
                    // wysłanie ENQ - rozpoczęcie transmisji
     headerState=HEADER_STATE_GET_ACK;
                          // przejście do stanu oczekiwania
     break;                              // na potwierdzenie
//---------------------------------------------------------------------

case HEADER_STATE_GET_ACK:   // oczekiwanie na potwierdzenie przez
                             // kardiograf rozpoczęcia transmisji

     if (Form_Main->COM->readBuffer.Length())
        {
        if ((Form_Main->COM->readBuffer.Length()==1)&&
                                  (Form_Main->COM-
                           >readBuffer[1]==ACK))
            {                      // otrzymano potwierdzenie
            Form_Main->COM->readBuffer.Delete(1,1);
            if ((transMode==TRANS_MODE_TEST))
                                  // przejście do odpowiedniego
                                  // trybu
                {                 // transmisji zgodnie z transMode
                Form_Main->COM->send(transMode);
                headerState=HEADER_STATE_GET_OK;
                }
            else if ((transMode==TRANS_MODE_KEY))
                {
                Form_Main->COM->send(transMode);
                headerState=HEADER_STATE_GET_OK;
                }
            else if ((transMode==TRANS_MODE_MONITOR))
                {
                Form_Main->COM->send(transMode);
                headerState=HEADER_STATE_GET_OK;
                };

            }
         else  // otrzymano złą odpowiedź
            {
            Form_Main->COM->readBuffer="";
            Form_Main->COM->send(BREAK);
            Form_Main->COM->send(ENQ);    // ponowienie transmisji
            }
        }
     break;
//---------------------------------------------------------------------

case HEADER_STATE_GET_OK:
                 // oczekiwanie na potwierdzenie trybu transmisji
                // przez EKG
     if (Form_Main->COM->readBuffer.Length())
        {
        if ((unsigned char)(Form_Main->COM->readBuffer[1])==OK)
           {              // odebrano potwierdzenie
           Form_Main->COM->readBuffer.Delete(1,1);
           headerState=HEADER_STATE_DONE;

           if (transMode==TRANS_MODE_KEY)
                         // w trybie klawiatury wysyłany
               {         // jest znak do EKG
               Form_Main->COM->send(KeyboardRequest[1]);
               transMode=TRANS_MODE_TEST;
               KeyboardRequest.Delete(1,1);
```

```
                KeyAccepted=false;  // program będzie czekał na
                }                   // potwierdzenie przyjęcia
                                    // klawisza przez kardiograf
            }
         else  // niewłaściwa odpowiedź, odrzucono tryb transmisji
            {
            Form_Main->COM->readBuffer="";
            Form_Main->COM->send(BREAK);
            Form_Main->COM->send(ENQ);     // ponowienie transmisji
            headerState=HEADER_STATE_GET_ACK;
            }
         };
      break;
//---------------------------------------------------------------------

case HEADER_STATE_DONE:      // zakończono transmisję nagłówka


      if (Form_Main->COM->readBuffer.Length())
                                    // odczytano jakieś znaki
         {
         if (transMode==TRANS_MODE_MONITOR)
                                   // w trybie monitora oczekiwanie
            {                      // na 8-bajtową próbkę
            if (Form_Main->COM->readBuffer.Length()>=8)
                {
                if ((Form_Main->COM->readBuffer[1]& 0xC0 ) == 0xC0 )
                    // pierwsze 2 bity próbki muszą być 11

                    {
                    sampleBuffer[sampleCounter]=
                         Form_Main->COM->readBuffer.SubString(1,8);
                              // dodanie próbki do cyklicznego bufora
                    sampleCounter++;
                    if (sampleCounter>=88) sampleCounter=0;

                    Form_Main->COM->readBuffer.Delete(1,8);

                    if (KeyboardRequest.Length() && KeyAccepted)
                        { // jeśli naciśnięto klawisz
                        }
                    else // w przeciwnym razie wysłanie potwierdzenia
                        Form_Main->COM->send(ACK);
                    }
                else Form_Main->COM->send(BREAK);
                                   // brak synchronizacji
                }
             else if (idleTimeOut)
                          // jeżeli przez długi czas nie otrzymano
                {         // próbek ponowienie transmisji
                transMode=TRANS_MODE_TEST;
                Form_Main->COM->send(BREAK);
                Form_Main->COM->send(ENQ);
                idleTimeOut=false;
                Form_Main->COM->readBuffer="";
                headerState=HEADER_STATE_GET_ACK;
                }

            }
         else if ( ((unsigned char)
                           (Form_Main->COM->readBuffer[1])==ENQ) &&
```

```
                        (Form_Main->COM->readBuffer.Length()==1)  )
            {           // żądanie transmisji ze strony EKG
            Form_Main->COM->readBuffer.Delete(1,1);
            Form_Main->COM->send(ACK);
            headerState=HEADER_STATE_EKG_REQUEST;
            }
         else if ( ((unsigned char)
                        (Form_Main->COM->readBuffer[1])==BREAK) &&
                        (Form_Main->COM->readBuffer.Length()==1)  )
            {           // kardiograf przerwał transmisję
            Form_Main->COM->readBuffer.Delete(1,1);
            Form_Main->COM->send(BREAK);
            Form_Main->COM->send(ENQ);
            headerState=HEADER_STATE_GET_ACK;
            transMode=TRANS_MODE_TEST;
            }
         else if ( ((unsigned char)
                       (Form_Main->COM->readBuffer[1])==OK))
            {            // kardiograf zaakceptował przesłany klawisz
            Form_Main->COM->readBuffer.Delete(1,1);
            KeyAccepted=true;
            } //nie można rozpoczynać nowej
              //transmisji bez tego potwierdzenia
         else
            { // błąd
            Form_Main->COM->readBuffer="";
            Form_Main->COM->send(BREAK);
            }
         }
      else     // nic nie przeczytano
         {

          if (KeyboardRequest.Length() && KeyAccepted)
                          // naciśnięto klawisz
              {           // oraz poprzedni został zaakceptowany
              transMode=TRANS_MODE_KEY;
                    //rozpoczęcie transmisji w trybie klawiatury
              headerState=HEADER_STATE_GET_ACK;
              Form_Main->COM->send(ENQ);
              }
          else if (idleTimeOut) // po dłuższym czasie zastoju
              {         // rozpoczęcie transmisji w trybie testu
              transMode=TRANS_MODE_TEST;
              Form_Main->COM->send(ENQ);
              idleTimeOut=false;
              headerState=HEADER_STATE_GET_ACK;
              }
          else if (requestMonitor)
               //rozpoczęcie transmisji w trybie monitora
              {
              transMode=TRANS_MODE_MONITOR;
              headerState=HEADER_STATE_GET_ACK;
              Form_Main->COM->send(ENQ);
              requestMonitor=false;
              sampleCounter=0;
              sampleBufferTail=0;
              }
          }
      break;
//---------------------------------------------------------------------
```

```
      case HEADER_STATE_EKG_REQUEST: // kardiograf inicjuje transmisję
                                     // sprawdzenie żądanego trybu transmisji
           if (Form_Main->COM->readBuffer.Length())
              {
              if ( ((unsigned char)
                        (Form_Main->COM->readBuffer[1])==TRANS_MODE_LCD)
                             &&(Form_Main->COM->readBuffer.Length()==1)  )
              {                      // będą przesyłane dane z LCD
              Form_Main->COM->readBuffer.Delete(1,1);
              Form_Main->COM->send(OK);
                                     // potwierdzenie trybu transmisji
              headerState=HEADER_STATE_EKG_LCD;
              }
              else
              { // nieznany tryb transmisji.
              Form_Main->COM->readBuffer="";
              Form_Main->COM->send(BREAK);
              Form_Main->COM->send(ENQ);     // ponowienie transmisji
              headerState=HEADER_STATE_GET_ACK;

              }
              };

           break;

      //-----------------------------------------------------------------

      case HEADER_STATE_EKG_LCD:
                           // oczekiwanie na 82 bajty z zawartością LCD

           if (Form_Main->COM->readBuffer.Length()>=82)
              {
                 for (int i=1; i<=80; i++)
                     Form_Main->COM->readBuffer[i]
                       =polskie(Form_Main->COM->readBuffer[i]);

              Form_Main->EKG_LCD1->Caption=
                     Form_Main->COM->readBuffer.SubString(1,40);
              Form_Main->EKG_LCD2->Caption=
                     Form_Main->COM->readBuffer.SubString(41,40);
              Form_Main->COM->readBuffer.Delete(1,82);
              headerState=HEADER_STATE_DONE;
              // indeks 81,82 zawiera pozycję kursora
              };

           break;

      //-----------------------------------------------------------------

      }

}
```

****

monitor – Funkcja ta przetwarza otrzymywane w trybie monitora próbki i na ich podstawie rysuje na bieżąco wykres EKG oraz odświeża zawartość wyświetlacza LCD i podaje aktualny puls. Wywoływana jest przez funkcję obsługi zegara, o ile w buforze *sampleBuffer* umieszczone zostały nowe próbki. Zmienna s*ampleBuffer* jest buforem cyklicznym o długości 88 próbek. Ta liczba związana jest z cyklem przesyłania zawartości wyświetlacza LCD oraz innych informacji w ostatnim – ósmym bajcie każdej próbki.

Główna część funkcji zamknięta jest wewnątrz instrukcji *while,* która przetwarza kolejne otrzymane próbki w ośmiobajtowych porcjach.

Wewnątrz instrukcji *while* wykonywane są następujące czynności:

- W ostatnim bajcie próbki wykonywana jest konwersja kodu polskich liter, o ile próbka ma numer od 1 do 80 (próbki o tych numerach zawierają po 1 znaku z wyświetlacza LCD)

- Modyfikowany jest odpowiedni znak na wyświetlaczu, albo etykieta HR (heart rate, czyli puls) dla próbki o numerze 84

- Wartości sygnału dla poszczególnych kanałów są dekodowane. 9-cio bitowe wartości zostają złożone z najstarszego bitu z pierwszego bajtu próbki i z ośmiu młodszych bitów umieszczonych w kolejnych bajtach próbki. Razem przesyłanych jest sześć 9-bitowych wartości, z trzech kanałów, po dwie kolejne wartości w czasie na kanał. Wartości przechowywane są w zmiennych typu *int*

- Wartości kanałów przepisywane są do bufora *sampleSaveBuffer.* Bufor ten jest tablicą 16-bitowych zmiennych *(word)* i przeznaczony jest do późniejszego zapisania na dysk.

- Jeżeli dla każdego kanału w buforze *sampleSaveBuffer* zbierze się 300 wartości, próbki zapisywane są na dysk. Liczba 300 wartości odpowiada czasowi 1 sekundy. Daje strumień wielkości 1800 bajtów na sekundę (300 wartości * 3 kanały * 2 bajty na wartość). Drugą czynnością która jest przy tej okazji wykonywana, jest modyfikacja zmiennej *mmMV* na podstawie zawartości wyświetlacza LCD. Zmienna ta określa czułość rejestracji próbek EKG i jest używana do ustawienia właściwego odstępu miedzy poziomymi liniami siatki.

- Wartości próbek dla trzech kanałów rysowane są na ekranie, w głównym oknie aplikacji. Co 30 kolejnych próbek rysowany jest czarny prostokąt który zamazuje stary wykres, następnie rysowany jest wykres dla kanałów 1,2 i 3 kolejno w kolorach : czerwony, żółty i zielony.

- Jeżeli zmienna *EnableGrid* ustawiona jest na wartość *true* wtedy rysowana jest siatka razem z czarnym prostokątem zamazującym stary wykres. Siatka składa się z linii pionowych i poziomych. Rysowane siatki odbywa się co 30 pikseli co odpowiada odstępowi czasowemu równemu 1/10 sekundy. Linie poziome rysowane są w zmiennych odstępach w zależności od zmiennej *mmMV,* która na podstawie zawartości wyświetlacza LCD określa czułość rejestracji sygnału EKG. Odstęp między liniami odpowiada wielkości sygnału równej 1 mV (miliwolt).

Poniżej znajduje się treść funkcji *monitor*.

```
//////////////////////////////////////////////////////////////

void monitor() // analiza i wyświetlanie wyników 8-bajtowej próbki
{              // otrzymanej z EKG e trybie monitora
TPoint tmpPenPos;
int s1a,s2a,s3a,s1b,s2b,s3b;
unsigned long writeResult;

String l1=Form_Main->EKG_LCD1->Caption;
String l2=Form_Main->EKG_LCD2->Caption;
                              // pobranie aktualnego stanu wyświetlacza

while (sampleBufferTail!=sampleCounter)
                              // 8-bajtowe próbki zapisane są w cyklicznym
    {                         // buforze długości 88 próbek
    if (sampleBufferTail)
        {

        if (sampleBufferTail<=80)  // w próbce od 1 do 80 ósmy bajt zawiera
                                   // znak z LCD
            sampleBuffer[sampleBufferTail][8]=
                polskie(sampleBuffer[sampleBufferTail][8]);

        if (sampleBufferTail<=40)       // pierwsza linia LCD
            l1[sampleBufferTail]=sampleBuffer[sampleBufferTail][8];
        else if (sampleBufferTail<=80)  // druga linia LCD
            l2[sampleBufferTail-40]=sampleBuffer[sampleBufferTail][8];
        else if (sampleBufferTail==84)  // HR - puls
            Form_Main->LabelHR->Caption="HR : "+
                        String((int)sampleBuffer[sampleBufferTail][8]);
       }

            //dekodowanie 9-bitowych próbek
    s1a=(unsigned char)(sampleBuffer[sampleBufferTail][2]);
    s2a=(unsigned char)(sampleBuffer[sampleBufferTail][3]);
    s3a=(unsigned char)(sampleBuffer[sampleBufferTail][4]);
    s1b=(unsigned char)(sampleBuffer[sampleBufferTail][5]);
    s2b=(unsigned char)(sampleBuffer[sampleBufferTail][6]);
    s3b=(unsigned char)(sampleBuffer[sampleBufferTail][7]);

    if ((sampleBuffer[sampleBufferTail][1]) & 0x20) s1a+=256;
    if ((sampleBuffer[sampleBufferTail][1]) & 0x10) s2a+=256;
    if ((sampleBuffer[sampleBufferTail][1]) & 0x08) s3a+=256;
    if ((sampleBuffer[sampleBufferTail][1]) & 0x04) s1b+=256;
    if ((sampleBuffer[sampleBufferTail][1]) & 0x02) s2b+=256;
    if ((sampleBuffer[sampleBufferTail][1]) & 0x01) s3b+=256;

            // wypełnianie bufora do zapisu

    sampleSaveBuffer[sampleSaveCounter][0]=(WORD)s1a;
    sampleSaveBuffer[sampleSaveCounter][1]=(WORD)s2a;
    sampleSaveBuffer[sampleSaveCounter][2]=(WORD)s3a;
    sampleSaveBuffer[sampleSaveCounter+1][0]=(WORD)s1b;
    sampleSaveBuffer[sampleSaveCounter+1][1]=(WORD)s2b;
    sampleSaveBuffer[sampleSaveCounter+1][2]=(WORD)s3b;

    sampleSaveCounter+=2;
```

```
    if (sampleSaveCounter==300) // po umieszczeniu 300 próbek = 1 sekunda
                                //zapis na dysk
       {
       sampleSaveCounter=0;
       WriteFile(hSampleFile,sampleSaveBuffer,
           sizeof(sampleSaveBuffer),&writeResult,NULL);

       String tmp=Form_Main->EKG_LCD2->Caption.SubString(8,5);
       if      (tmp=="2.5mm") mmMV=5;  // raz na sekundę sprawdzamy czułość
       else if (tmp==" 5 mm") mmMV=10; // rejestracji na podstawie
       else if (tmp==" 10mm") mmMV=20; // zawartości wyświetlacza LCD
       else if (tmp==" 20mm") mmMV=40;
       else if (tmp==" 40mm") mmMV=80;
       else mmMV=0;

       }

if ((x%30)==1)  // co 30 pikseli = 10x/sec
        {
        x2=x+30;                    // x2 z wyprzedza x o 30 pikseli
        if (x2>xMax) x2=1;
                                    // zamazywanie starego wykresu
        Form_Main->Image1->Canvas->Pen->Color=clBlack;
        Form_Main->Image1->Canvas->Brush->Color=clBlack;
        Form_Main->Image1->Canvas->Rectangle(x2,0,x2+30,386);

        if (EnableGrid)
            {                       // rysowanie siatki
            Form_Main->Image1->Canvas->Pen->Color=clGray;
            Form_Main->Image1->Canvas->PenPos=TPoint(x2,0);
            Form_Main->Image1->Canvas->LineTo(x2,386);
            if (mmMV)
                for (int i=0; i<386; i+=mmMV)
                   {
                   Form_Main->Image1->Canvas->PenPos=TPoint(x2,i);
                   Form_Main->Image1->Canvas->LineTo(x2+30,i);
                   }
            }
        }

                                                // rysowanie nowego wykresu
    Form_Main->Image1->Canvas->Pen->Color=clRed;      // kanał 1
    Form_Main->Image1->Canvas->PenPos=TPoint(x,128-old_s1b/4 );
    Form_Main->Image1->Canvas->LineTo(x+1,128-s1a/4 );
    Form_Main->Image1->Canvas->LineTo(x+2,128-s1b/4 );

    Form_Main->Image1->Canvas->Pen->Color=clYellow;   // kanał 2
    Form_Main->Image1->Canvas->PenPos=TPoint(x,256-old_s2b/4 );
    Form_Main->Image1->Canvas->LineTo(x+1,256-s2a/4 );
    Form_Main->Image1->Canvas->LineTo(x+2,256-s2b/4 );

    Form_Main->Image1->Canvas->Pen->Color=clLime;     // kanał 3
    Form_Main->Image1->Canvas->PenPos=TPoint(x,384-old_s3b/4 );
    Form_Main->Image1->Canvas->LineTo(x+1,384-s3a/4 );
    Form_Main->Image1->Canvas->LineTo(x+2,384-s3b/4 );
    x+=2;
    if (x>xMax) x=1;

    sampleBufferTail=rot88(sampleBufferTail,1);
```

```
    old_s1b=s1b;
    old_s2b=s2b;
    old_s3b=s3b;
    } // while

Form_Main->EKG_LCD1->Caption=l1;
Form_Main->EKG_LCD2->Caption=l2;

}
```

****

polskie - W module *transmission* znajdują się jeszcze dwie mniej istotne funkcje. Pierwsza z nich nazywa się *polskie* i przetwarza kody polskich liter z wartości używanych w kardiografie na wartości używane w komputerze. Używana przy przesyłaniu zawartości wyświetlacza z kardiografu do komputera.

```
char polskie(char ch) // konwersja znaków polskich EKG->PC
{
char c;
switch((unsigned char)ch)
    {
    case 128: c=165;break;
    case 129: c=198;break;
    case 130: c=202;break;
    case 131: c=163;break;
    case 132: c=209;break;
    case 133: c=211;break;
    case 134: c=140;break;
    case 135: c=175;break;
    case 136: c=143;break;
    default : c=ch ;break;
    };
return c;
}
```

****

rot88 - jest funkcją , która inkrementuje modulo 88 zmienną *source* o wartość *offset.* Używana jest do zwiększania wartości zmiennych obsługujących cykliczny bufor *sampleBuffer.*

```
int rot88(int source,int offset) //dodawanie modulo 88
{
return (source+offset+88)%88;
};
```

4.2.4 Moduł *formMain*

Ten moduł związany jest z głównym formularzem (okienkiem) aplikacji. W pliku nagłówkowym modułu zawarta jest deklaracja klasy *TForm_Main,* która dziedziczy po klasie *TForm*. Klasa *TForm* należy do hierarchii klas biblioteki VCL i jest klasą macierzystą wszystkich formularzy w systemie *Builder*.

Przyglądając się deklaracji klasy *TForm* warto zwrócić uwagę na sekcję *__published*. Sekcja ta zawiera elementy publiczne i jest automatycznie generowana i modyfikowana przez *Builder*, w trakcie budowy formularza. W sekcji tej zawarte są pola, które są komponentami umieszczonymi na formularzu. Wśród nich znajdują się: etykiety, przyciski, panele, obraz i zegar. Metodami sekcji *__published* są funkcje obsługi zdarzeń związanych z formularzem oraz z komponentami umieszczonymi na formularzu. Metodami są np.: funkcja obsługująca kliknięcie przycisku, naciśniecie przycisku klawiatury komputera, obsługa cyklicznego zdarzenia zegara.

Oto deklaracja klasy *TForm*:

```
//---------------------------------------------------------------------------
--
class TForm_Main : public TForm
{
__published:      // IDE-managed Components
        TTimer *Timer;
        TImage *Image1;
        TPanel *Panel1;
        TPanel *Panel2;
        TPanel *Panel3;
        TPanel *Panel4;
        TLabel *EKG_LCD1;
        TLabel *EKG_LCD2;
        TLabel *LabelHR;
        TSpeedButton *ButtonAuto;
        TSpeedButton *ButtonManual;
        TSpeedButton *ButtonFilter35;
        TSpeedButton *ButtonmmmV;
        TSpeedButton *ButtonMenu;
        TSpeedButton *ButtonStop;
        TSpeedButton *ButtonRhytm;
        TSpeedButton *Filter50;
        TSpeedButton *Buttonmms;
        TSpeedButton *Button1mV;
        TSpeedButton *ButtonZamknij;
        TSpeedButton *ButtonPort;
        TSpeedButton *ButtonMonitor;
        TSpeedButton *ButtonHelp;
        TSpeedButton *ButtonChangeSampleFile;
        TSpeedButton *ButtonGridOnOff;

        void __fastcall FormActivate(TObject *Sender);
        void __fastcall TimerTimer(TObject *Sender);
        void __fastcall ButtonMonitorClick(TObject *Sender);
        void __fastcall ButtonFilter35Click(TObject *Sender);
        void __fastcall Filter50Click(TObject *Sender);
        void __fastcall ButtonAutoClick(TObject *Sender);
        void __fastcall ButtonManualClick(TObject *Sender);
        void __fastcall ButtonStopClick(TObject *Sender);
```

```
        void __fastcall ButtonRhytmClick(TObject *Sender);
        void __fastcall ButtonmmmVClick(TObject *Sender);
        void __fastcall ButtonmmsClick(TObject *Sender);
        void __fastcall ButtonMenuClick(TObject *Sender);
        void __fastcall Button1mVClick(TObject *Sender);
        void __fastcall KeybDown(TObject *Sender, WORD &Key,
                                                  TShiftState Shift);
        void __fastcall FormKeyUp(TObject *Sender, WORD &Key,
                                                  TShiftState Shift);
        void __fastcall ButtonZamknijClick(TObject *Sender);
        void __fastcall ButtonPortClick(TObject *Sender);
        void __fastcall ButtonHelpClick(TObject *Sender);

private:    // User declarations

public:           // User declarations
        SerialComm *COM; // obiekt obługi portu szeregowego

        __fastcall TForm_Main(TComponent* Owner);
};
//---------------------------------------------------------------------------
--
```

Poniżej znajduje się opis funkcji używanych w module:

****

FormActivate – Funkcja obsługuje zdarzenie *OnActivate* formularza. Jest to pierwsza wykonywana funkcja w programie, spośród funkcji zdefiniowanych przez użytkownika. Wykonuje czynności przygotowawcze do pracy aplikacji. Są to:

- Załadowanie z pliku numeru portu szeregowego używanego do transmisji. Jeśli plik nie istnieje ustawiana jest wartość domyślna *COM1*.
- Stworzenie obiektu klasy *serialComm* zdefiniowanej w module *serialComm*. Obiekt ten obsługuje operacje wejścia/wyjścia portu szeregowego.
- Inicjuje / zeruje wiele zmiennych globalnych takich jak: bufor klawiatury, bufor próbek i inne.
- Ustawia tryb transmisji na *TRANS_MODE_TEST*.
- Otwiera do zapisu plik przeznaczony na próbki EKG.
- Inicjuje sekcję krytyczną blokującą możliwość wielokrotnego równoczesnego wywołanie funkcji *transmission*
- Inicjuje komunikację szeregową (otwarcie portu, ustawienie parametrów i uruchomienie wątka czytającego).
- Uruchamia zegar.
- Pierwszy raz wywołuje funkcję *transmission.*

Od tego momentu rozpoczyna się wymiana danych między komputerem i kardiografem.

Poniżej znajduje się treść funkcji *FormActivate.*

```
//---------------------------------------------------------------------------
void __fastcall TForm_Main::FormActivate(TObject *Sender)
{
TStringList *tmp;
tmp = new TStringList;

if (access(CONFIG_FILE,6)==0)          // jeżeli plik istnieje
    {
    tmp->LoadFromFile(CONFIG_FILE);    // załadowanie z pliku numeru portu.
    Form_Port->RadioGroup1->ItemIndex=tmp->Strings[0].ToInt()-1;
    };


COM = new SerialComm; // tworzenie obiektu obsługującego port szeregowy

transMode=TRANS_MODE_TEST;             // zmienna stanu transmisji
headerState=HEADER_STATE_SEND_ENQ;     // zmienna stanu transmisji
KeyboardRequest="";                    // bufor klawiatury
KeyAccepted=true;                      // zmienna stanu akceptacji klawisza

sampleBuffer = new String[88];         // bufor 9-bitowych próbek EKG
xMax=Image1->Width-4;

hSampleFile= CreateFile("ekg_sample.dat",GENERIC_WRITE,0,NULL,
                        CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);

COM->InitializeSerialCommunication(Form_Port->RadioGroup1->ItemIndex+1);
// otwarcie portu, uruchomienie wątka

transmissionLock = new TCriticalSection;
Timer->Enabled=true;

transmissionLock->Enter();
transmission();                         // rozpoczęcie transmisji
transmissionLock->Leave();

requestMonitor=true;
}
```

****

TimerTimer – Funkcja obsługuje zdarzenie *OnTimer* obsługujące komponent zegar o nazwie *Timer*. Zdarzenie generowane jest około 20 razy na sekundę w Windows 95/98. Dla Windows NT/2000 dzięki lepszemu zarządzaniu czasem Można ustalić generację zdarzenia z większą częstotliwością.

Głównym zadaniem tej funkcji jest wywoływanie funkcji *transmission* oraz *monitor* w określonych sytuacjach, innych jednak niż odczytanie znaku z portu. W przypadku odczytania znaku z portu funkcja *transmission* jest wywoływana przez wątek czytający *readThread.*

Wywołanie funkcji *transmission* zawsze otoczone jest dwoma instrukcjami jak poniżej:

```
transmissionLock->Enter();
transmission();
transmissionLock->Leave();
```

Zmienna *transmissionLock* jest obiektem klasy *TCriticalSection*. Obiekty te używane są do chronienia fragmentów kodu przed równoczesnym dostępem przez różne wątki. W tym programie chroniona jest funkcja *transmission*, która jest wywoływana zarówno przez opisywaną tutaj funkcję *TimerTimer* jak i wątek czytający *readThread*. Niedopuszczalna byłaby sytuacja, jednoczesnego wywołania funkcji *transmission,* gdyż mogłoby to doprowadzić do nieoczekiwanego zachowania programu. Wywołanie przez pierwszy wątek instrukcji *transmissionLock->Enter()* powoduje, że drugi wątek który wywoła taką samą instrukcję zostanie wstrzymany, do czasu kiedy pierwszy wywoła instrukcję *transmissionLock->Leave()*.

Funkcja *TimerTimer* wykonuje następujące czynności:

- Inkrementuje licznik *timerCounter.*

- Jeżeli naciśnięto klawisz a poprzedni został obsłużony i nie jesteśmy w trakcie transmisji nagłówka, wywoływana jest funkcja *transmission.* Spowoduje to wejście w tryb klawiatury.

- Jeżeli ustawiana jest flaga *requestMonitor* i łącze jest wolne, wywoływana jest funkcja *transmission.* Spowoduje to wejście w tryb monitora.

- Funkcja *transmission* wywoływana jest także gdy łącze szeregowe jest wolne przez ponad 40 wywołań zegara.

- Jeżeli jesteśmy w trybie monitora to jeżeli w buforze *sampleBuffer* znajdują się próbki wywoływana jest funkcja *monitor*, jeżeli natomiast nie otrzymaliśmy próbek w trybie monitora przerywamy transmisję i ustawiamy flagę *requestMonitor* co spowoduje ponowne uruchomienie trybu monitora.

Poniżej znajduje się treść funkcji *TimerTimer*

```
//---------------------------------------------------------------------------
void __fastcall TForm_Main::TimerTimer(TObject *Sender)
{

timerCounter++;

if ( KeyboardRequest.Length() && KeyAccepted &&
(headerState==HEADER_STATE_DONE)
     &&(idleCounter)
   )
        // jeżeli naciśnięto klawisz, poprzedni został obsłużony i łącze
        // jest wolne następuje wysłanie go do EKG
    {
    transmissionLock->Enter();
    transmission();
    transmissionLock->Leave();
    }
```

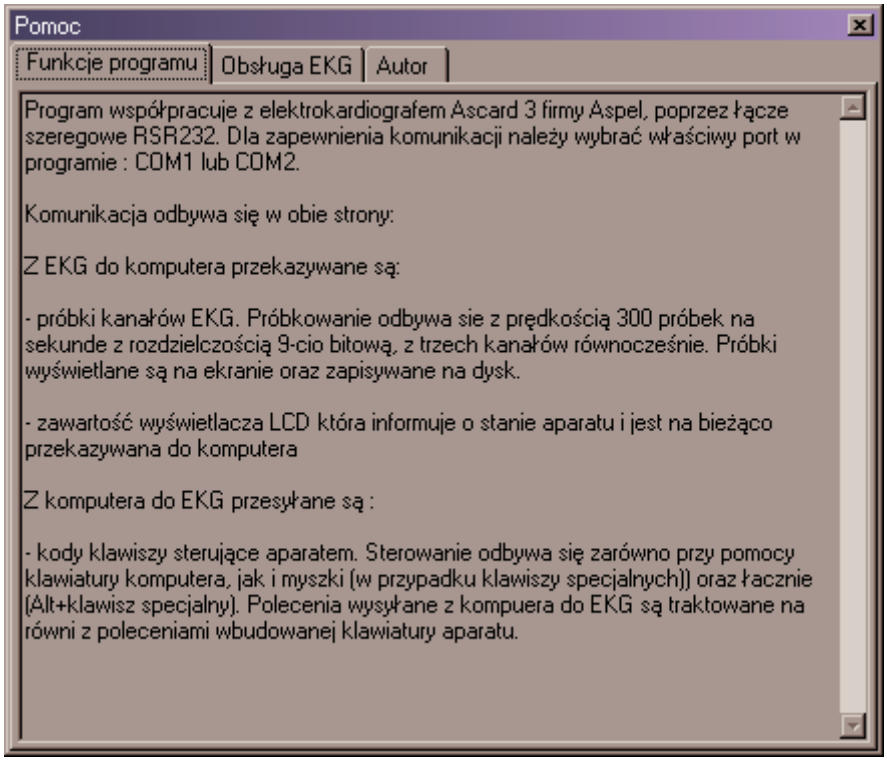
```
else if (  requestMonitor
           && KeyAccepted && (headerState==HEADER_STATE_DONE)
           &&(idleCounter)
        ) // w przeciwny razie jeżeli jest
          // żądanie trybu monitora, obsłużono
          // klawisz i łącze jest wolne następuje inicjacja trybu monitora
    {
    transmissionLock->Enter();
    transmission();
    transmissionLock->Leave();
    }
else // w przeciwny razie
    {
    idleCounter++;
    if ((idleCounter>40) && (headerState==HEADER_STATE_DONE) )
       // jeżeli łącze wolne ponad 40 okresów
       // timera -> podtrzymanie transmisji
        {
        idleTimeOut=true;
        transmissionLock->Enter();
        transmission();
        transmissionLock->Leave();
        }
    }

if (transMode==TRANS_MODE_MONITOR)
    { // w trybie monitora -> analiza i rysowanie próbek z EKG
    if (sampleBufferTail!=sampleCounter)
        monitor();
    else if (headerState==HEADER_STATE_DONE)
       {  // jeżeli nie ma próbek -> prawdopodobne zerwanie transmisji
          // -> następuje przywrócenie transmisje w trybie monitora
       requestMonitor=true;
       transMode=TRANS_MODE_TEST;
       COM->readBuffer="";
       }
    }
}
```

****

Inne funkcje modułu *formMain* obsługują zdarzenia naciśnięcia odpowiednich przycisków. Te przyciski, które są klawiszami z klawiatury kardiografu powodują dodanie do bufora klawiatury *KeyboardRequest* odpowiedniego kodu znaku.
Podobnie działa funkcja obsługi klawiatury komputera *KeybDown*. Uwzględnia ona dodatkowo naciśnięcie klawisza *ALT* który modyfikuje działanie innych klawiszy.
Funkcje *ButtonPortClick, ButtonHelpClick i ButtonChangeSampleFileClick* powodują wyświetlenie odpowiedniego okienka poleceniem *ShowModal*, które wyświetla okienko modalne, czyli takie które blokuje działanie okienka macierzystego, aż do zamknięcia okienka potomnego.

4.2.5 Moduł *formHelp*

Moduł *formHelp* zawiera definicję okienka pomocy. Na formularzu zawarte są 3 zakładki z krótkim tekstem pomocy dotyczącym funkcji programu, obsługi EKG i ogólnymi informacjami o programie. Nie ma w tym module żadnych zdefiniowanych przez programistę funkcji, ani obsługi zdarzeń.



Rys 2. Okienko pomocy

4.2.6 Moduł *formPort.*

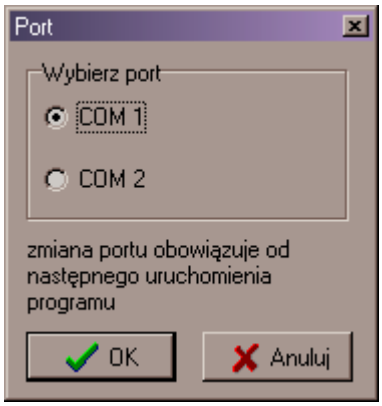
Moduł *formPort* zawiera definicję okienka dialogowego służącego do wyboru portu szeregowego COM1 lub COM2 służącego do komunikacji z elektrokardiografem. Na formularzu zawarta jest dwuelementowa grupa przycisków wyboru (klasa TRadioGroup), która charakteryzuje się tym, że tylko jeden z przycisków jest wybrany.
Pod przyciskami wyboru znajdują się klawisze *OK* i *Anuluj,* których naciśniecie powoduje zamknięcie okienka.
W klasie definiującej opisywany formularz zawarte są dwie metody związane ze zdarzeniami dotyczącymi formularza:

FormShow – wywoływana w czasie rysowania formularza powoduje ustawienie aktywnego przycisku wyboru zgodnie z używanym portem. Treść funkcji jest następująca:

```
//---------------------------------------------------------------------------
void __fastcall TForm_Port::FormShow(TObject *Sender)
{
RadioGroup1->ItemIndex=Form_Main->COM->port-1;
}
```

BitBtnOKClick – wywoływana w momencie kliknięcia w przycisk OK, powoduje zapisanie do pliku zdefiniowanego przez stałą CONFIG_FILE wybranego numeru portu. Treść funkcji jest następująca:

```
//---------------------------------------------------------------------------
void __fastcall TForm_Port::BitBtnOKClick(TObject *Sender)
{
TStringList *tmp;
tmp = new TStringList;

tmp->Add(RadioGroup1->ItemIndex+1);
tmp->SaveToFile(CONFIG_FILE);
}
```



Rys 3. Okienko wyboru portu.

4.2.7 Moduł *formSampleName*

Moduł ten definiuje okienko służące do zmiany nazwy pliku, do którego zapisywane są próbki EKG. Na formularzu znajduje się pole edycyjne do wpisania nazwy pliku, etykieta oraz dwa przyciski *OK* i *Anuluj*. Naciśnięcie któregoś z przycisków powoduje zamknięcie okienka.
W klasie definiującej opisywany formularz zawarte są dwie metody związane ze zdarzeniami dotyczącymi formularza:

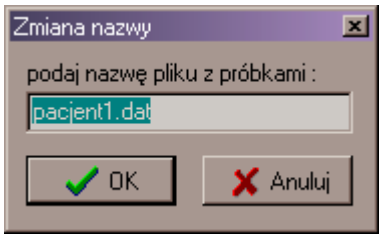
FormShow – wywoływana w czasie pojawienia się formularza na ekranie powoduje skopiowanie nazwy pliku ze zmiennej reprezentującej etykietę w okienku głównym aplikacji do pola edycyjnego. Treść funkcji jest następująca:

```
//---------------------------------------------------------------------------
void __fastcall TForm_SampleName::FormShow(TObject *Sender)
{
EditSampleFile->Text=Form_Main->LabelSampleFileName->Caption;
}
```

BitBtnOKClick – wywoływana w momencie kliknięcia w przycisk OK, powoduje skopiowanie nazwy w odwrotną stronę niż poprzednia funkcja, co jest równoznaczne z akceptacją wpisanej nazwy pliku. Treść funkcji jest następująca:

```
//---------------------------------------------------------------------------
void __fastcall TForm_SampleName::BitBtnOKClick(TObject *Sender)
{
Form_Main->LabelSampleFileName->Caption=EditSampleFile->Text;
}
```

Zmiana nazwy pliku realizowana jest dopiero przy zakończeniu programu przez funkcję *FormDestroy*. W trakcie działania programu próbki zapisywane są do pliku tymczasowego.

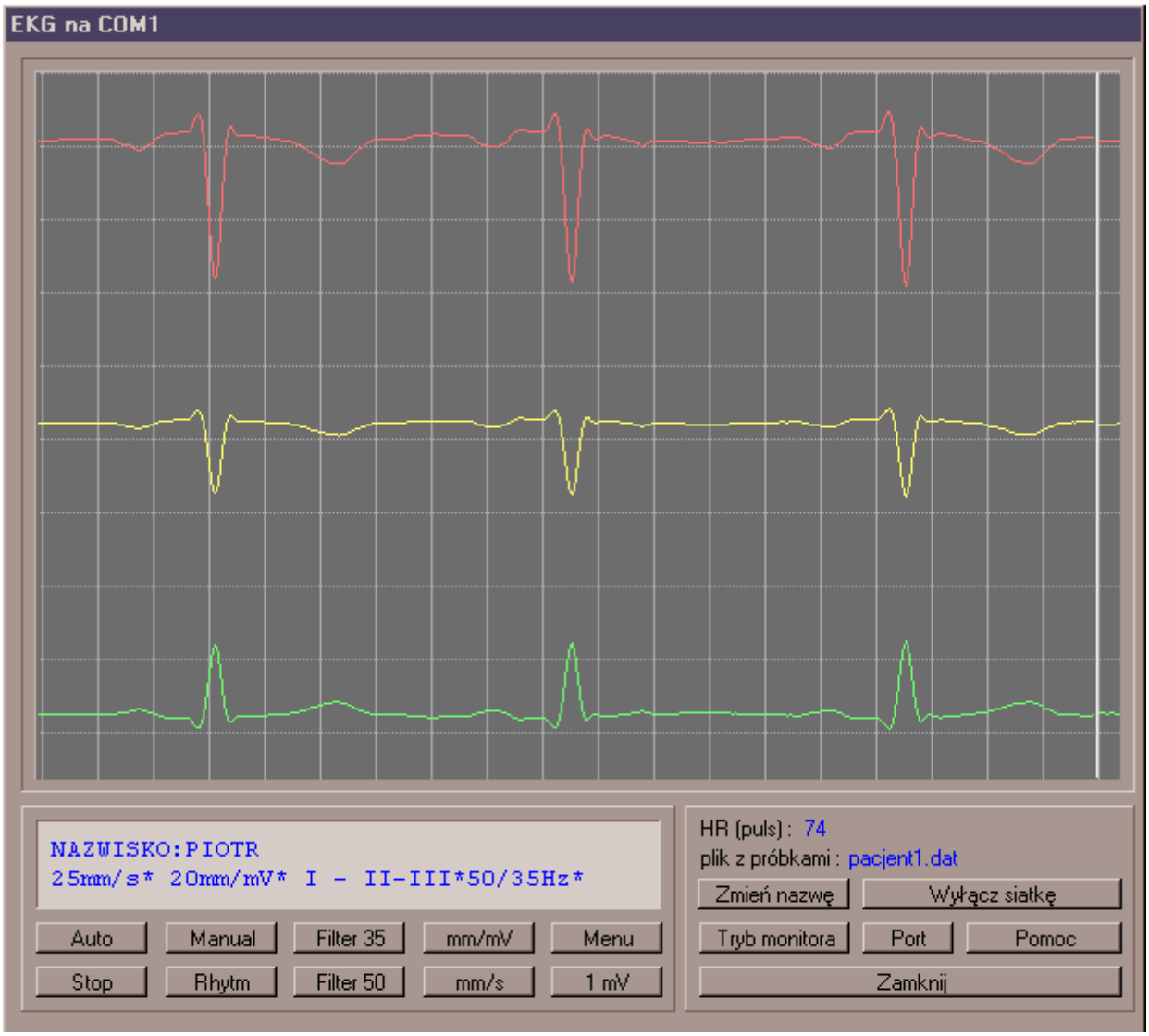


Rys 4. Okienko zmiany nazwy pliku z próbkami

4.3 Obsługa programu.

Program współpracuje z elektrokardiografem AsCARD-3 firmy Aspel, poprzez łącze szeregowe RS-232. Dla zapewnienia komunikacji należy połączyć komputer i kardiograf odpowiednim przewodem.

Oto wygląd głównego okna programu:



Rys 5. Główne okno programu w trakcie pracy

Po uruchomieniu program od razu łączy się kardiografem i rozpoczyna rysowanie wykresu pracy serca, zapisuje próbki na dysk, podaje puls, przesyła stan wyświetlacza i jest gotowy do sterowania kardiografem przez użytkownika programu. Jeżeli wykres nie jest rysowany ani nie jest widoczny stan wyświetlacza należy sprawdzić czy kardiograf jest włączony i prawidłowo podłączony do komputera oraz czy w programie wybrany jest właściwy port.

Na górze okienka, na pasku tytułu podawany jest aktualnie używany port.

Największą część okna zajmuje wykres EKG. Rysowane jednocześnie są 3 przebiegi z trzech kanałów EKG (w trybie EKG) lub też innych sygnałów jak np. fonokardiogram (w trybie polikardiograficznym). Na wykresie rysowana jest siatka, która umożliwia oszacowanie amplitudy sygnału w miliwoltach, a także jego zmian w czasie. Linie poziome siatki rysowane są co 1 mV natomiast pionowe co 1/10 sekundy. Rysowanie siatki można wyłączyć lub włączyć przyciskiem *Wyłącz siatkę / Włącz siatkę*.

Pod wykresem EKG znajduje się pole z dwoma liniami tekstu. Jego zawartość jest identyczna z zawartością wyświetlacza LCD aparatu. W pierwszej linii wyświetlane są informacje o pacjencie.

Druga linia wyświetlacza zawiera po kolei następujące informacje:

- Szybkość rejestracji : jest to prędkość przesuwu papieru w drukarce aparatu. Dla programu nie ma ona znaczenia.

- Czułość rejestracji : podaje jaką wielkość w milimetrach ma sygnał 1 mV. W zależności od czułości skalowana jest siatka na wykresie.

- Nazwy trzech kolejnych transmitowanych i rysowanych wyprowadzeń EKG lub też innych sygnałów w trybie polikardiograficznym. Kanał 1 rysowany jest kolorem czerwonym, kanał 2 kolorem żółtym, a kanał 3 kolorem zielonym.

- Stan filtrów. Sygnał EKG może był filtrowany dla usunięcia zakłóceń dwoma filtrami 35 i 50 Hz. Ich użycie jest sygnalizowane przez wyświetlenie na wyświetlaczu liczby 35 i (lub) 50.

- W przypadku niewłaściwego podłączenia elektrod wyświetlany jest symbol „INOP" oznaczający duże zakłócenia na elektrodach.

Pod polem z zawartością wyświetlacza znajduje się grupa 10 specjalnych przycisków z klawiatury kardiografu. Przyciski te wybierane są za pomocą myszy.

- *Auto*, *Manual* i *Rhytm* służą do rozpoczęcia rejestracji przebiegu EKG na drukarce aparatu w trybie: automatycznym ręcznym oraz w trybie z zapisem rytmu serca. Przycisk *stop* wyłącza rejestrację.

- Przyciski *Filter 35* i *Filter 50* włączają odpowiednie filtry.

- Przyciski *mm/s* i *mm/mV* ustawiają odpowiednio prędkość i czułość rejestracji.

- Przycisk *Menu* umożliwia zarządzanie pamięcią EKG, konfiguracje aparatu oraz przełączenie się między trybem elektrokardiograficznym i polikardiograficznym.

- Przycisk 1mV drukuje na drukarce aparatu sygnał 1mV w aktualnej skali.

Pozostałe klawisze jak *Enter*, *Escape*, klawisze kursora, litery, cyfry oraz ich kombinacje z *Altem* wybierane są za pomocą klawiatury komputera. Skrócony przegląd funkcji klawiatury znajduje się w rozdziale 3.2. Po dokładniejszy opis należy sięgnąć do instrukcji obsługi aparatu.

W prawym dolnym rogu okienka znajduje się pole zawierające 2 etykiety oraz 6 przycisków:

- Etykieta *HR (puls)* : podaje ilość uderzeń serca na minutę. Wartość jest aktualizowana jeżeli włączone jest przesyłanie próbek do komputera

- Etykieta *plik z próbkami* : podaje nazwę pliku do którego zapisywane są próbki EKG. Plik tworzony jest po zakończeniu pracy programu. W trakcie pracy programu próbki zapisywane są do pliku tymczasowego. Jeżeli plik z próbkami istniał już wcześniej zostanie bez ostrzeżenia nadpisany.

- Przycisk *Zmień nazwę* : używany jest do zmiany nazwy pliku do którego zostaną zapisane próbki EKG. Aby uniknąć nadpisywania próbek należy podawać unikatowe nazwy plików.

- Przycisk W*yłącz siatkę (włącz siatkę) :* służy do przełączania wyświetlania siatki. Siatka przydaje się do określenia parametrów wykresu pracy serca.

- Przycisk *Tryb monitora* przełącza przesyłanie do komputera próbek EKG. Przy włączonym trybie monitora sygnał z trzech kanałów EKG jest rysowany i zapisywany na dysk. Jest także podawana częstość pracy serca. Wyłączenie powoduje zaprzestanie rysowania wykresu, oraz przerwanie zapisywania próbek na dysk. Po ponownym naciśnięciu przycisku rysowanie i zapisywanie próbek jest wznawiane. Niezależnie od trybu monitora zawartość wyświetlacza zawsze jest uaktualniana, i zawsze można wysyłać znaki sterujące kardiografem.

- Przycisk *Port* powoduje wybranie numeru portu używanego do komunikacji z kardiografem. Może być to *COM1* lub *COM2*. Numer portu zapamiętywany jest w odpowiednim pliku. Zmiana portu obowiązuje od następnego uruchomienia programu.

- Przycisk *Pomoc* służy do wyświetlenia przydatnych informacji o programie.

- Przycisk *Zamknij* kończy pracę programu.

5 Podsumowanie i wnioski.

W podsumowaniu należy zaznaczyć, że cel pracy został osiągnięty, czyli został napisany w pełni działający program spełniający wymagania tematu pracy.

Oczywiście w trakcie pisania programu pojawiały się liczne problemy. Część problemów związana była z używanym systemem operacyjnym *Microsoft Windows 95* który już przy niewielkim obciążeniu procesora nierówno odmierza takty zegara i przydziela niedostateczną ilość czasu dla programu. Efektem tego jest częste zrywanie transmisji z kardiografem, szczególnie w trybie monitora, co związane jest z rysowaniem wykresu, które zwiększa obciążenie procesora. Program wyposażony jest w mechanizm szybkiego wykrywania takich sytuacji i wtedy natychmiastowo ponawia zerwaną transmisję. Taka sytuacja może być widoczna dla użytkownika jako chwilowe zatrzymanie rysowania wykresu. Powoduje to również utratę tej części próbek w czasie kiedy nastąpiło przerwanie transmisji.

Dla zwiększenia stabilności pracy warto byłoby zmienić używany system operacyjny na *Windows NT* ewentualnie *Windows 2000*, które to systemy pracują znacznie pewniej niż używany w pracowni *Windows 95*. W trakcie pracy programu sporadycznie zdarzają się takie sytuacje, że transmisja nagle się zatrzymuje i jedynym rozwiązaniem jest wyłączenie i ponowne uruchomienie programu. Jest rzeczą oczywistą, że w dużych programach, a zwłaszcza wielowątkowych występują błędy ujawniające się w szczególnych sytuacjach. Wykrycie i usunięcie takich błędów jest bardzo trudne. Najlepszym przykładem jest system *Windows* w którym błędów jest bardzo wiele. Również w programie będącym treścią tej pracy mogą występować błędy, których nie udało się dotąd zauważyć. Jednakże w trakcie pisania programu autor dołożył wszelkich starań, aby błędów było jak najmniej.

Dla zwiększenia użyteczności programu, można rozbudować go o funkcje wykonujące pomiary i analizę zarejestrowanych przebiegów sygnałów, co jednak wykracza poza zakres tej pracy. Program badający pracę serca może nie tylko mierzyć sygnał, ale też próbować wystawiać diagnozę. Można wyobrazić sobie sytuację, w której bardziej rozbudowany program tego typu, mógłby w szpitalu na bieżąco kontrolować stan zdrowia pacjentów i wykrywać nieprawidłowości pracy serca, pomagając w ten sposób personelowi szpitala.

Program ten będzie stanowił pomoc dydaktyczną przy stanowisku elektrokardiografii.

6 Literatura.

(1) Kent Reisdorph „C++ Builder 3”, Gliwice: Helion 1998

(2) Andrzej Zalewski „Programowanie w językach C i C++ z wykorzystaniem pakietu Borland C++”, Poznań: Wydaw. Nakom, 1994

(3) Jan Bielawski „ANSI C++”, Warszawa : Intersoftland, 1997

(4) Pomoc elektroniczna systemu *Builder*

(5) „Opis transmisji szeregowej elektrokardiograf – PC”, Zabierzów: Aspel S.A.

(6) „Instrukcja obsługi elektrokardiografu”, Zabierzów: Aspel S.A.

(7) Wojciech Mielczarek: „Szeregowe interfejsy cyfrowe”, Gliwice: Helion, 1993

(8) Wiesław Winiecki: „Organizacja komputerowych systemów pomiarowych”, Warszawa: Oficyna Wydaw. Politechniki Warszawskiej, 1997

(9) http://wwwnt.if.pwr.wroc.pl

(10) http://interface.tme.szczecin.pl

(11) http://ise.pl/informacje/komputery