

Politechnika Świętokrzyska
Wydział Mechatroniki i Budowy Maszyn
Kierunek: Mechanika i Budowa Maszyn
Specjalność: Zastosowanie Informatyki
Ścieżka dyplomowania: Automatyka

PRACA DYPLOMOWA

MONITOROWANIE WYBRANYCH SYGNAŁÓW WYKORZYSTUJĄCE KONTROLER JEDNOUKŁADOWY.

Sławomir Koczubiej

*Opiekun pracy:
dr inż. Zbigniew Lis*

Kielce 2002

Składam serdeczne podziękowania opiekunowi naukowemu Panu dr inż. Zbigniewowi Lisowi za cenne uwagi i wskazówki pomocne w napisaniu niniejszej pracy.

Mamie.

1. Cel i zakres pracy	5
2. Wstęp	
2.1. Znaczenie techniki mikroprocesorowej	7
2.2. Znaczenie przetwarzania A/C w technice.....	7
2.3. Sprzętowa realizacja przetwarzania A/C	8
3. Przetwarzanie analogowo-cyfrowe sygnałów ciągłych	
3.1. Wprowadzenie	9
3.2. Przetwarzanie w dziedzinie czasu	9
3.3. Przetwarzanie w dziedzinie amplitudy	11
3.4. Budowa przetworników A/C.....	12
4. Technika mikrokontrolerowa	
4.1. Wprowadzenie	18
4.2. Podstawowe elementy systemów mikroprocesorowych	20
4.3. Budowa i architektura mikrokontrolerów	22
4.4. Urządzenie peryferyjne i wyposażenie mikrokontrolerów	29
5. Środowisko programowe kontrolerów przemysłowych	
5.1. Wprowadzenie	33
5.2. Cechy i cykl programowania mikrokontrolera	34
5.3. Asembler	35
5.4. Inne środowiska	36
6. Budowa układu pomiarowego JKSPv2	
6.1. Wprowadzenie	40
6.2. Wybór podzespołów elektronicznych	40
6.3. Montaż i uruchomienie.....	44
7. Oprogramowanie współpracujące z układem JKSPv2	
7.1. Wprowadzenie	52
7.2. Środowisko programowania i system	52
7.3. Oprogramowanie komunikacji	57
7.4. Oprogramowanie analizy danych.....	66
8. Zastosowanie systemu pomiarowego JKSPv2	
8.1. Wprowadzenie	79
8.2. Wykonanie pomiarów.....	79
8.3. Analiza	80
9. Podsumowanie	89
Literatura	91

1. Cel i zakres pracy.

Techniki mikroprocesorowa i cyfrowa są jednymi z bardziej dynamicznie rozwijających się gałęzi nowoczesnego przemysłu. Ich szerokie możliwości sprawiają, iż można je odnaleźć w wielu zastosowaniach.

Celem pracy było zbudowanie systemu pomiarowego, składającego się z mikrokontrolerowego układu pomiarowego i stworzenie programu komputerowego do jego obsługi.

Pod pojęciem układu pomiarowego kryje się układ elektroniczny z własną pamięcią, sterowany mikrokontrolerem do przetwarzania analogowo-cyfrowego sygnałów i akwizycji pozyskanych w ten sposób danych. Wymagało to zapoznania się z budową i sposobem działania układów mikroprocesorowych jak również z zasadami budowy układów elektroniki cyfrowej. Mikrokontroler musi być jeszcze wyposażony w oprogramowanie sterujące jego pracą i oprogramowanie dla komputera klasy PC pozwalające na przesłanie do niego danych i ewentualnie dalszą ich analizę. Opracowanie oprogramowania obsługującego wymagało wykorzystania zaawansowanych technik programowania w systemach rodziny Microsoft Windows, za pomocą interfejsu programowania Win32 API.

W niniejszej pracy przedstawiono kolejno następujące zagadnienia:

- znaczenie techniki mikroprocesorowej i cyfrowej,
- omówienie podstawowych zagadnień z dziedziny przetwarzania analogowo-cyfrowego sygnałów ciągłych i jej sprzętowej realizacji,
- podstawowe wiadomości z zakresu budowy i funkcjonowania układów mikrokontrolerowych,
- wprowadzenie do środowisk programowania mikrokontrolerów przemysłowych, sposoby, realizacja i cykl programowania mikrokontrolerów,
- opis budowy i sposobu działania mikrokontrolerowego układu pomiarowego,
- podstawowe wiadomości z zakresu konstrukcji i zasad działania oprogramowania pracującego pod systemem Windows,
- opis i zasady funkcjonowania oprogramowania przeznaczonego do komunikacji z mikrokontrolerowym układem pomiarowym,
- opis i zasady działania oprogramowania do analizy danych,
- zastosowanie omówionego układu i oprogramowania.

Zbudowanie systemu pomiarowego opartego o mikrokontroler wraz z jego oprogramowaniem stanowiło duże wyzwanie. Należało rozwiązać problemy natury elektronicznej (fizyczna budowa układu pomiarowego oraz jego testowanie), problemy natury komunikacji szeregowej w środowisku Windows (opracowanie programu, wykorzystanie interfejsu programowania Win32 API i Borland

Delphi 5), problemy związane z oprogramowaniem mikrokontrolera i jego układów peryferyjnych (opracowanie programu, obsługa: pamięci, przetwornika analogowo-cyfrowego, wyświetlacza LCD, układu UART przy pomocy MCS Electronics Bascom) jak i natury programowej (wizualizacja i podstawowa analiza pozyskanych danych, Borland Delphi 5).

Zbudowanie systemu pomiarowego, było od początku podzielone na dwa dopełniające się zadania dyplomowe: *Monitorowanie wybranych sygnałów wykorzystujące kontroler jednoukładowy* – zadanie na pracę dyplomową Sławomira Koczubieja i *Komputerowa obsługa układu monitorowania wybranych sygnałów* – zadanie na pracę dyplomową Dariusza Juszczyka. Obejmują one wspólną budowę, uruchomienie i testowanie układu pomiarowego. Z tego powodu w tej pracy rozdział 6 *Budowa układu pomiarowego* i rozdział 8 *Zastosowanie systemu pomiarowego* powstały w wyniku wspólnej pracy Dariusz Juszczyka i Sławomira Koczubieja. Ponadto praca Dariusza Juszczyka zawiera rozważania nt. zagadnień transportu, analizy i przechowywania danych oraz programowania mikrokontrolerów za pomocą pakietu MCS Electronics Bascom. Natomiast praca Sławomira Koczubieja porusza zagadnienia sprzętowej budowy układów mikrokontrolerowych, sposobów ich programowania oraz zasady funkcjonowania i budowy programów pracujących w systemach operacyjnych rodziny Windows, budowanych przy pomocy pakietu Borland Delphi.

2.1. Znaczenie techniki mikroprocesorowej.

Historia techniki mikroprocesorowej nie jest długa. Za jej początek przyjmuje się rok 1972, kiedy to amerykańska firma Intel wprowadziła na rynek pierwszy mikroprocesor – układ scalony o oznaczeniu 4004 [12]. W czasie krótszym od 30 lat technika mikroprocesorowa rozwinęła się do tego stopnia, że bez przesady można ją uznać za jedną z głównych motorów napędowych rozwoju naszej cywilizacji technicznej. Układy techniki mikroprocesorowej są obecnie stosowane powszechnie: od komputerów osobistych, poprzez telefony komórkowe, sprzęt AGD, do robotów i skomplikowanego sprzętu medycznego oraz wojskowego.

2.2. Znaczenie przetwarzania A/C w praktyce.

Większość sygnałów w otaczającym nas świecie ma charakter analogowy (ciągły). Oznacza to, że są to sygnały ciągłe ze względu na czas i amplitudę. W technice zajmujemy się sygnałami charakteryzującymi obserwowane obiekty i zjawiska, które również mają charakter analogowy. Sygnały cyfrowe uzyskuje się z sygnałów analogowych poprzez przetwarzanie analogowo-cyfrowe (*przetwarzania A/C, analog-digital conversion*). Przetwarzanie analogowo-cyfrowe polega na kwantowaniu amplitudy sygnału za pomocą przetworników analogowo-cyfrowych. Kwantowanie amplitudy sygnału odbywa się w określonych dyskretnych przedziałach czasu. Proces ten (kwantowania amplitudy sygnału) nazywa się próbkowaniem.

Parametrami różnego rodzaju zjawisk fizycznych jak i procesów technologicznych są ciągłe wielkości nieelektryczne. Wielkości takie, jak temperatura czy ciśnienie są zamieniane na sygnały elektryczne, a następnie przetwarzane na wartości liczbowe, co pozwala na uzależnienie od nich procesów sterowania i zastosowania dużo tańszych niż analogowe, regulatorów cyfrowych.

Wynika stąd obszar zastosowanie układów analogowo-cyfrowych. Przede wszystkim, są one stosowane w układach sterowania i pomiarowych. Przy zastosowaniu przetworników A/C sterowaniu podlegają sygnały analogowe. Aczkolwiek przetwarzanie analogowo-cyfrowe nie jest wykorzystywane tylko w sterowaniu. Jest ono stosowane wszędzie, gdzie mamy do czynienia z cyfrowym przetwarzaniem i obróbką sygnałów, telekomunikacji, inżynierii dźwięku, metrologii.

2.3. Sprzętowa realizacja przetwarzania A/C.

Rozwój technologii półprzewodnikowych układów scalonych wywiera decydujący wpływ na budowę przyrządów i systemów przetwarzania (A/C i pomiarowych). Układy takie coraz częściej są wykonywane z pojedynczych płytek krzemowych, zawierających zarówno systemy przetwarzania jak i pozyskiwania informacji [7, 13].

We współczesnych systemach pomiarowych przetworniki analogowo-cyfrowe występują albo w postaci specjalizowanych kart, przystosowanych do umieszczania w komputerach osobistych, albo w postaci oddzielnych bloków, jakimi są uniwersalne multimetry cyfrowe [7]. Można spotkać również urządzenia specjalizowane takie, jak dyktafony cyfrowe czy holtery cyfrowe.

Stosowanie komputerów oraz specjalnego oprogramowania wspomagającego jest najmniej kosztownym sposobem realizacji cyfrowych systemów pomiarowych i przetwarzania danych. Takie układy można stosować do realizacji mniej dokładnych systemów. Podstawową przeszkodą jest mała rozdzielczość przetwarzania kart A/C oraz wpływ zakłóceń elektromagnetycznych, generowanych przez komputer. Należy zaznaczyć, że komputery podczas pracy wytwarzają szerokie spektrum zakłóceń elektromagnetycznych, zarówno przewodzonych jak i wypromieniowanych. Zakłócenia te nie pozostają bez wpływu na procesy przetwarzania stosunkowo słabych sygnałów analogowych.

Pewniejszą metodą jest stosowanie uniwersalnych multimetrów cyfrowych. Mają one galwanicznie rozdzielone obwody pomiarowe układu od wyjść łączących z cyfrową częścią systemu pomiarowego. Posiadają też dużo większą prędkość działania i większą rozdzielczość.

Podstawowymi parametrami charakteryzującymi przetworniki analogowo-cyfrowe są [7, 13]:

- długość słowa – liczba bitów na których podany jest wynik przetwarzania A/C, typowo 8, 10, 12 do 24,
- czas przetwarzania – czas od rozpoczęcia przetwarzania do momentu, gdy wynik może być odczytany, od ułamków μs do setek ms,
- napięcie wejściowe – a w zasadzie zakres (unipolarny lub bipolarny), który zostanie przetworzony na odpowiadające mu liczby.

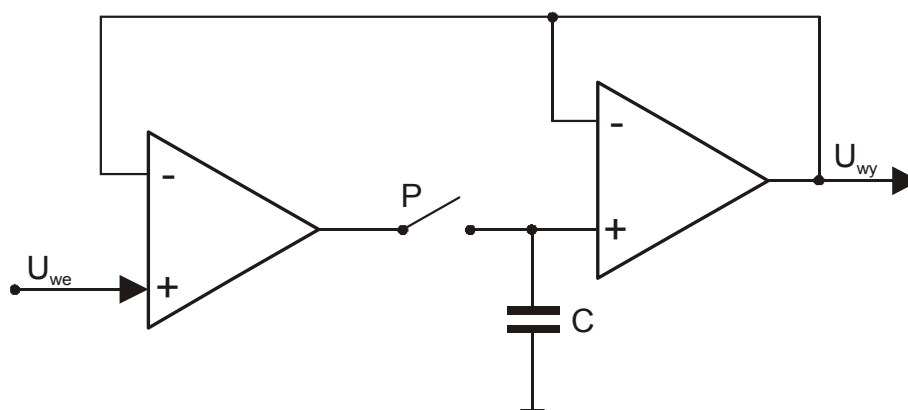
3. Przetwarzanie analogowo-cyfrowe sygnałów ciągłych.

3.1. Wprowadzenie.

Sygnały analogowe są ciągłe pod względem czasu jak i amplitudy. Sygnały cyfrowe uzyskuje się na drodze dyskretyzacji w dziedzinie czasu i amplitudy. Dyskretyzacja w dziedzinie czasu nosi nazwę próbkowania. Dyskretyzacja w dziedzinie amplitudy nazywa się kwantowaniem. Układy dyskretyzacji w dziedzinie czasu i amplitudy są niezbędne w systemach przetwarzania i pozyskiwania danych cyfrowych z sygnałów analogowych [13].

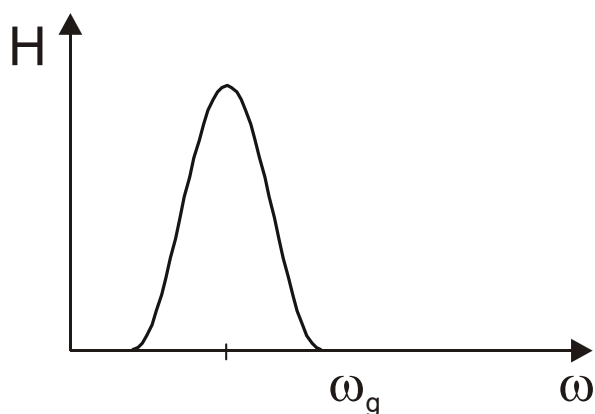
3.2. Przetwarzanie w dziedzinie czasu.

Dyskretyzacji sygnałów w dziedzinie czasu dokonuje się przez próbkowanie. Stosuje się do tego układy próbkująco-pamiętające (*układy P/P, sample and hold circuits*), jak na rysunku 3.1. W zakresie małych i średnich częstotliwości układy próbkująco-pamiętające są realizowane za pomocą wzmacniaczy operacyjnych ze sprzężeniem zwrotnym, wyposażonym w kondensator pamiętający [7, 13]. W stanie śledzenia (przełącznik P układu jest zamknięty) napięcie na kondensatorze C powtarza sygnał wejściowy U_{we} . Otwarcie przełącznika P powoduje przełączenie układu do stanu pamiętania. W stanie tym na wyjściu układu utrzymywana jest wartość chwilowa sygnału, występująca na jego wejściu w chwili otwierania przełącznika, tj. w chwili próbkowania.



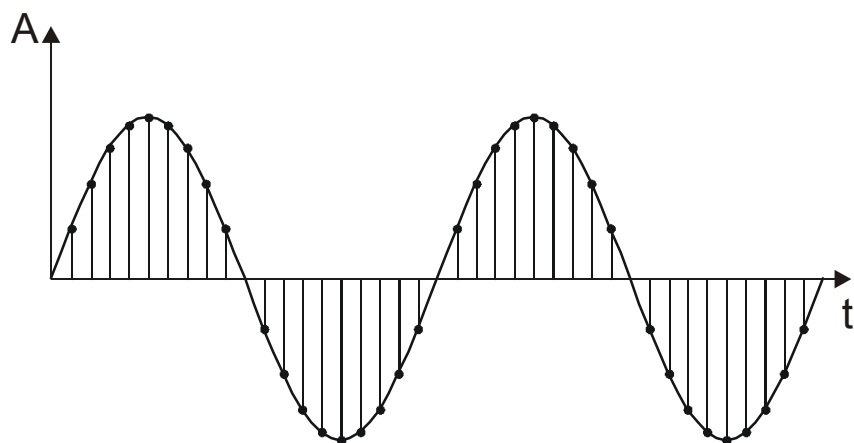
Rysunek 3.1. Budowa typowego układu próbkująco-pamiętającego.

Niezależnie od właściwości zastosowanych układów, próbkowanie może powodować błędy, powodowane niedostateczną częstotliwością pobierania próbek. Zgodnie z twierdzeniem Nyquista przyjmuje się, że częstotliwość próbkowania powinna być co najmniej równa dwukrotnej częstotliwości granicznej próbkowanego sygnału [7, 14].



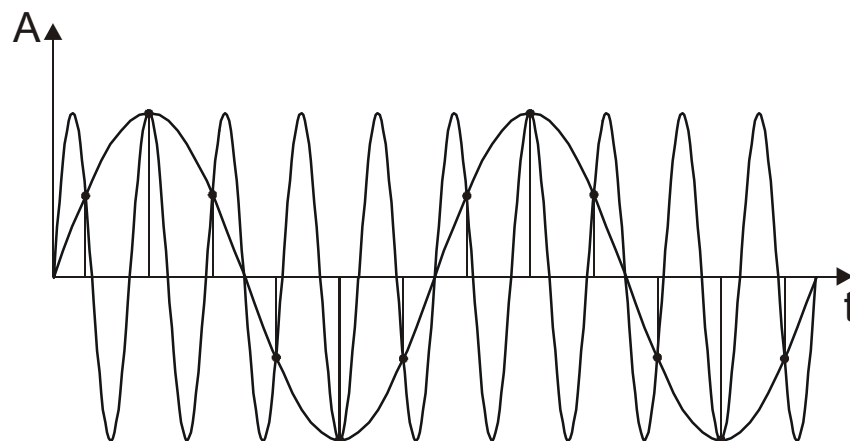
Rysunek 3.2. Rozkład sygnału w dziedzinie częstotliwości.

Częstotliwość graniczna (rysunek 3.2) jest równa najwyższej składowej próbkowanego sygnału. Poprawnie skwantowany sygnał jest pokazany na rysunku 3.3. Niedotrzymanie tego warunku powoduje brak możliwości jednoznacznej interpretacji wyników próbkowania (rysunek 3.4). Należy dodać, że spełnienie kryterium Nyquista jest wystarczającym warunkiem odtworzenia próbkowanego przebiegu tylko przy nieskończonym czasie próbkowania. W warunkach rzeczywistych, przy ograniczonym czasie próbkowania, odtworzenie parametrów i kształtu sygnału wymaga zastosowania większej częstotliwości próbkowania, niż to wynika z kryterium Nyquista [9, 14].

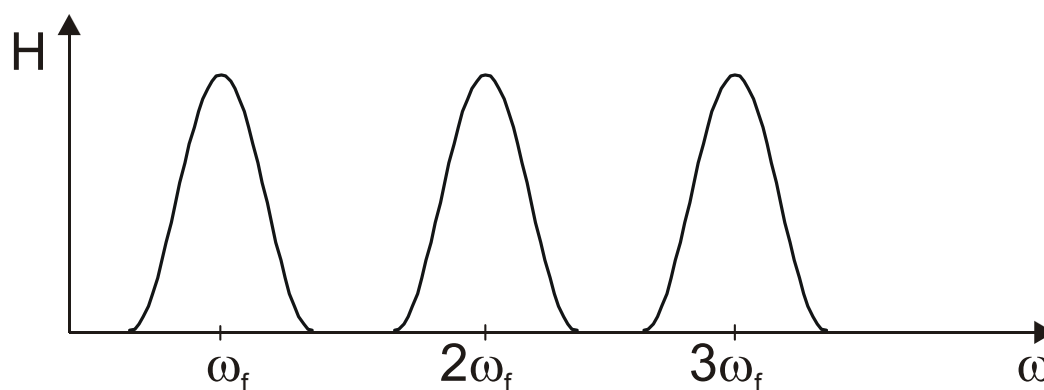


Rysunek 3.3. Próbkowanie przebiegu z dostateczną częstotliwością.

Operacja próbkowania wprowadza powielanie widma sygnału próbkowanego (rysunek 3.5). Powielanie następuje wokół częstotliwości próbkowania sygnału i jego wielokrotności. W celu uniknięcia nakładania się tych widm (*aliasing*) należy stosować filtry dolnoprzepustowe mające na celu ograniczenie pasma przenoszenia sygnału wejściowego [9, 14].



Rysunek 3.4. Próbkowanie przebiegu z niedostateczną częstotliwością.



Rysunek 3.5. Powielanie widma przy próbkowaniu sygnałów analogowych.

3.3. Przetwarzanie w dziedzinie amplitudy.

Dyskretyzacji sygnałów w dziedzinie amplitudy dokonuje się przez kwantyzację. Stosuje się do tego przetworniki analogowo-cyfrowe (*układy A/C, digital analog converters*). Proces przetwarzania A/C w dziedzinie amplitudy polega na zamianie wartości wejściowego sygnału analogowego na odpowiadającą mu wartość wyrażoną cyfrowo. Wartość odpowiednika cyfrowego sygnału analogowego zależy od rozpiętości przetwornika i jego zakresu przetwarzania. Zakres przetwarzania n -bitowego przetwornika jest proporcjonalnie podzielony na $2^n - 1$ równych części. Wartości sygnałów analogowych zawierające się w granicach jednej części, otrzyma taką samą wartość cyfrową. Zatem dokładność przetwarzania zwiększa się wraz ze zwiększaniem rozpiętości przetwornika i ze zmniejszaniem zakresu jego przetwarzania.

3.4. Budowa przetworników A/C.

Rozwój technologii scalonych układów cyfrowych, dokonany na przestrzeni ostatnich lat, spowodował istotną zmianę asortymentu wytwarzanych przetworników analogowo-cyfrowych. Dostępne przetworniki można zaliczyć do trzech grup [7, 13]:

- z równoczesną komparacją (*flash converters*),
- z szeregową komparacją (*successive approximation converters*),
- nadpróbkujące z modulatorem sigma-delta (*oversampling sigma-delta converters*).

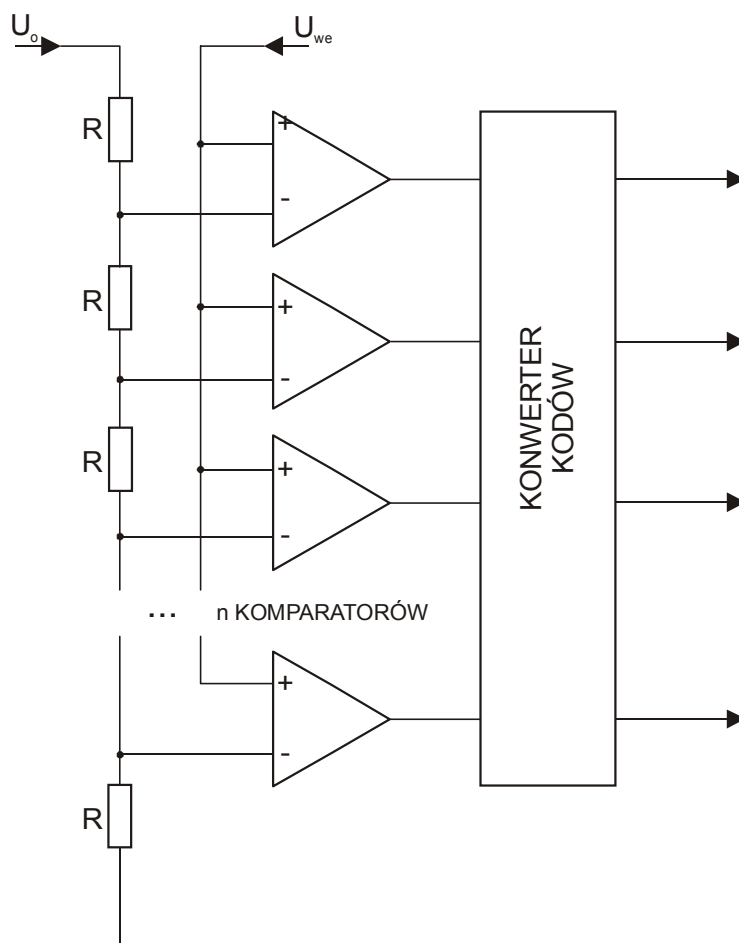
Pozostałe, do niedawna popularne sposoby przetwarzania analogowo-cyfrowego, takie, jak np. dwukrotne całkowanie i ich kolejne modyfikacje, nie są już obecnie stosowane [7, 13].

PRZETWORNIKI A/C Z RÓWNOCZESNĄ KOMPARACJĄ.

Przetworniki o długości słowa wyjściowego n bitów zawiera $2^n - 1$ komparatorów, porównujących amplitudę sygnału wejściowego U_{we} z odpowiednimi częściami napięcia odniesienia U_o , uzyskiwanymi z kolejnych wyjść dzielnika złożonego z jednakowych rezystorów. Kod termometryczny sygnałów wyjściowych poszczególnych komparacji jest przeliczany na kod dwójkowy za pomocą cyfrowego konwertera kodów (rysunek 3.6).

Zaletą przetwornika z równoczesną komparacją jest duża szybkość działania, wynikająca z braku elementów sekwencyjnych. Podstawową wadą znaczny pobór mocy oraz dosyć duże wymiary struktury. Budowa przetwornika o długości słowa 8 bitów wymaga zastosowania 255 komparatorów. Zwiększenie rozdzielczości o dodatkowy bit pociąga za sobą dwukrotny wzrost liczby niezbędnych elementów. Z tych względów zasada równoczesnej komparacji jest wykorzystywana do budowy najszybszych przetworników o rozdzielczości nie przekraczających 10 bitów.

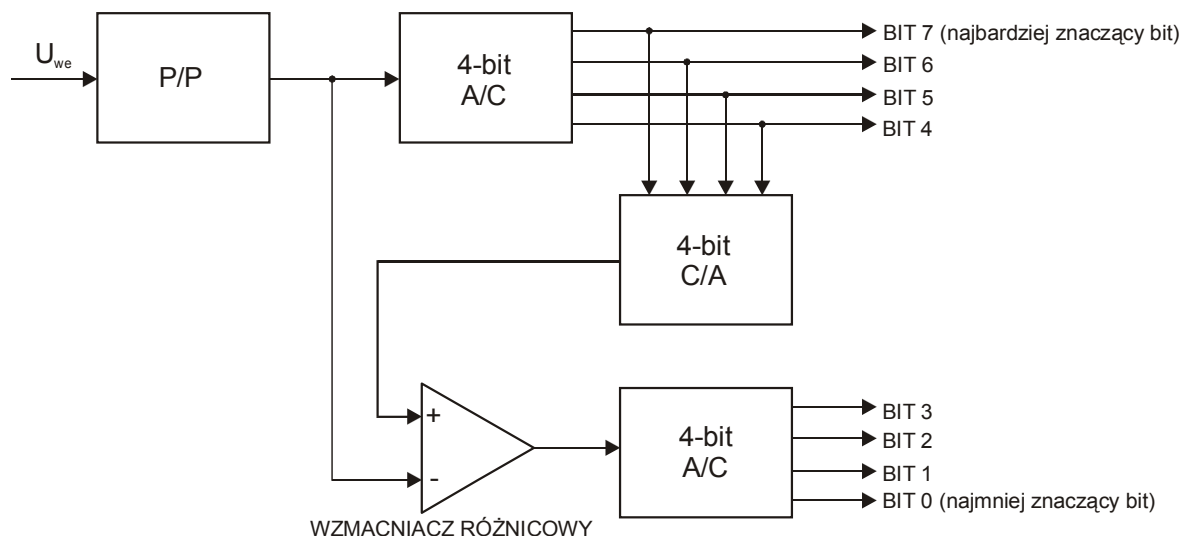
Szybkie przetworniki o rozdzielczości przekraczającej 8 bitów wymagają stosowania komparacji dwustopniowej (rysunek 3.7). Cykl przetwarzania takich przetworników składa się z dwóch faz. W pierwszej fazie za pomocą równoczesnego przetwarzania analogowo cyfrowego ustalana jest wartość kilku bardziej znaczących bitów wyniku. Wynik pierwszego przetwarzania jest za pomocą przetwornika cyfrowo-analogowego (C/A) zamieniany na odpowiadającą mu wartość sygnału analogowego i odejmowany od sygnału wejściowego. Sygnał różnicowy jest przetwarzany za pomocą drugiego równoczesnego przetwornika analogowo-cyfrowego, tworząc mniej znaczącą część bitów wyniku.



Rysunek 3.6. Jednostopniowy przetwornik analogowo-cyfrowy z równoczesną komparacją.

W przypadku wykorzystania podobnie szybkich elementów, cykl przetwarzania przetwornika dwustopniowego trwa dwa razy dłużej niż jednostopniowego. W celu uzyskania poprawnych wyników, należy zapewnić stałą wartość przetwarzanego sygnału podczas obu faz pracy przetwornika. Dlatego niezbędne jest poprzedzenie przetwornika dwustopniowego układem próbkująco-pamiętającym (*P/P*). Układ próbkująco-pamiętający znajduje się podczas całego cyklu przetwarzania w stanie pamiętania.

Budowa przetwornika dwustopniowego wymaga znacznie mniejszej liczby komparatorów i elementów dekodujących niż, jednostopniowego układu z bezpośrednią komparacją. Realizacja układu przedstawionego na rysunku wymaga zaledwie 30 komparatorów, co stanowi bardzo istotną zaletę w porównaniu z liczbą 255 komparatorów, niezbędnych do realizacji przetwornika 8 bitowego z bezpośrednią komparacją.

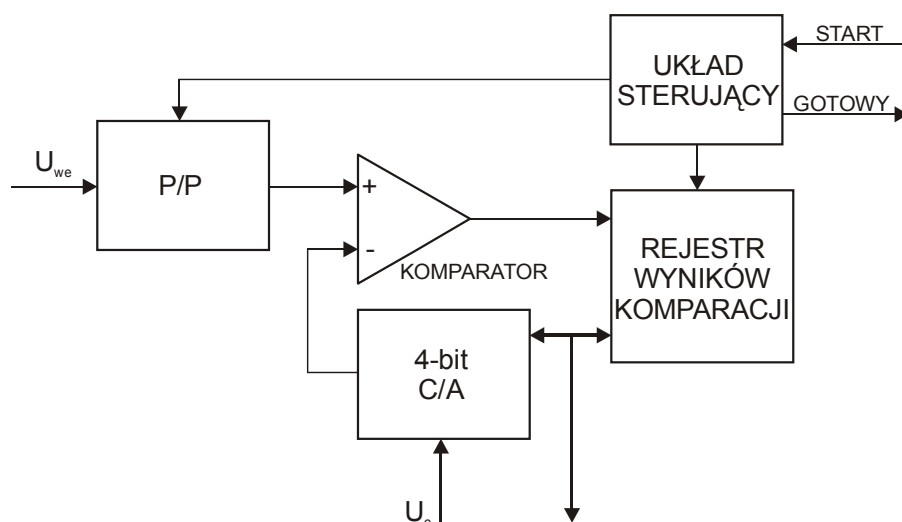


Rysunek 3.7. Dwustopniowy przetwornik z równoczesną komparacją.

W przemyśle są dostępne 12-bitowe, trzystopniowe przetworniki przystosowane do przetwarzania z szybkością 20 Mb/s (milionów próbek na sekundę) [7].

PRZETWORNIKI A/C Z SZEREGOWĄ KOMPARACJĄ.

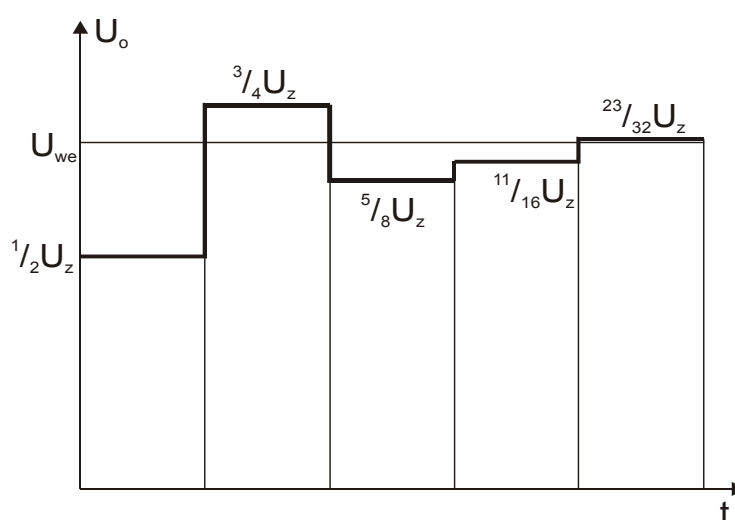
W przetwornikach analogowo-cyfrowych o średniej szybkości działania (rzędu 10^5 - 10^7 próbek na sekundę [7, 15]) niemal wyłącznie stosuje się zasadę szeregową komparacji sygnału wejściowego z kolejno malejącymi przyrostami napięcia odniesienia (rysunek 3.8).



Rysunek 3.8. Przetwornik analogowo-cyfrowy z komparacją szeregową.

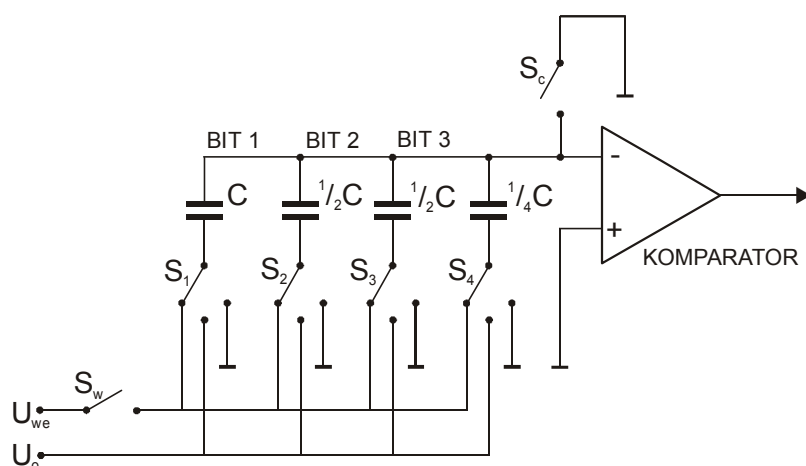
W pierwszym cyklu sygnał wejściowy porównywany jest z napięciem wzorcowym, generowanym przez pomocniczy przetwornik cyfrowo-analogowy.

Następuje to po włączeniu pierwszego bitu wyniku, napięcie wzorcowe w tej fazie odpowiada połowie całego zakresu przetwarzania przetwornika. W zależności od wyniku komparacji ustalonym po wykonaniu pierwszego kroku, najbardziej znaczący bit wyniku i odpowiadający mu przyrost napięcia odniesienia pozostaje w stanie włączonym lub jest zerowany. Następnie przetwornik cyfrowo-analogowy włącza kolejny przyrost napięcia odniesienia, w tym przypadku odpowiadający $1/4$ całego zakresu przetwarzania. W następnym kroku, w zależności od wyniku komparacji, ostatnio włączony przyrost napięcia odniesienia pozostaje w stanie włączonym lub jest zerowany, dalej włączany jest kolejny przyrost tj. $1/8$ napięcia odniesienia. Procedura ta jest powtarzana aż do wyczerpania wszystkich bitów przetwornika analogowo-cyfrowego. Pełen cykl przetwarzania n bitowego przetwornika składa się z n cykli zegarowych i po jego ukończeniu stan przetwornika cyfrowo-analogowego powinien z dokładnością do najmniej znaczącego bitu odwzorowywać wartość sygnału wejściowego (rysunek 3.9). Podobnie jak w przypadku wielostopniowego przetwornika z równoczesną komparacją, przetwornik z komparacją szeregową powinien być poprzedzony układem próbkująco-pamiętającym. W związku z tym pełen cykl przetwarzania tego przetwornika należy uzupełnić o okres pobierania próbki mierzonego sygnału.



Rysunek 3.9. Zasada komparacji sygnału wejściowego w przetworniku analogowo-cyfrowym z szeregową komparacją.

Niemal wszystkie przetworniki analogowo-cyfrowe są obecnie wykonywane w postaci układów scalonych [7, 13]. Powoduje to tendencję do zastępowania drabinek rezystorowych zespołami kondensatorów o niewielkich pojemnościach (rysunek 3.10), które mogą być wykorzystywane do generacji regulowanych napięć odniesienia.

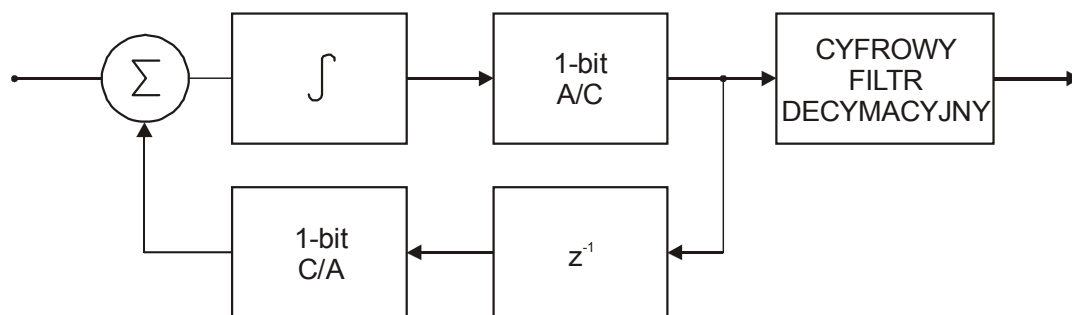


Rysunek 3.10. Zasada działania 3-bitowego przetwornika z dzielnikiem pojemnościowym.

Jeżeli S_1 , S_2 , S_3 i S_4 są dołączone do potencjału zerowego, na wejściu A komparatora powstaje potencjał równy $-U_{we}$. Przełączenie przełącznika S_1 do napięcia referencyjnego U_o spowoduje dodanie połowy wartości U_o do potencjału $-U_{we}$. Stan wyjścia komparatora może być wykorzystany do podjęcia decyzji o pozostawieniu lub wyłączeniu najbardziej znaczącego bitu tj. o pozostawieniu przełącznika S_1 dołączonego do U_o lub przełączeniu go do potencjału zerowego. W następnych cyklach podobna procedura jest kontynuowana w odniesieniu do przełączników S_2 , S_3 i S_4 . Po zakończeniu, przełączniki S_1 , S_2 , S_3 , S_4 i S_w są dołączane do wejścia U_{we} , przełącznik S_c do potencjału zerowego.

NADPRÓBKUJĄCE PRZETWORNIKI A/C.

W ostatnich latach szerokie zastosowanie znalazły przetworniki z modulatorami sigma-delta (rysunek 3.11). W przetwornikach tego typu wykorzystanie nadpróbkowania zapewnia możliwość osiągnięcia wysokiej rozdzielczości bez konieczności stosowania precyzyjnych lub wymagających

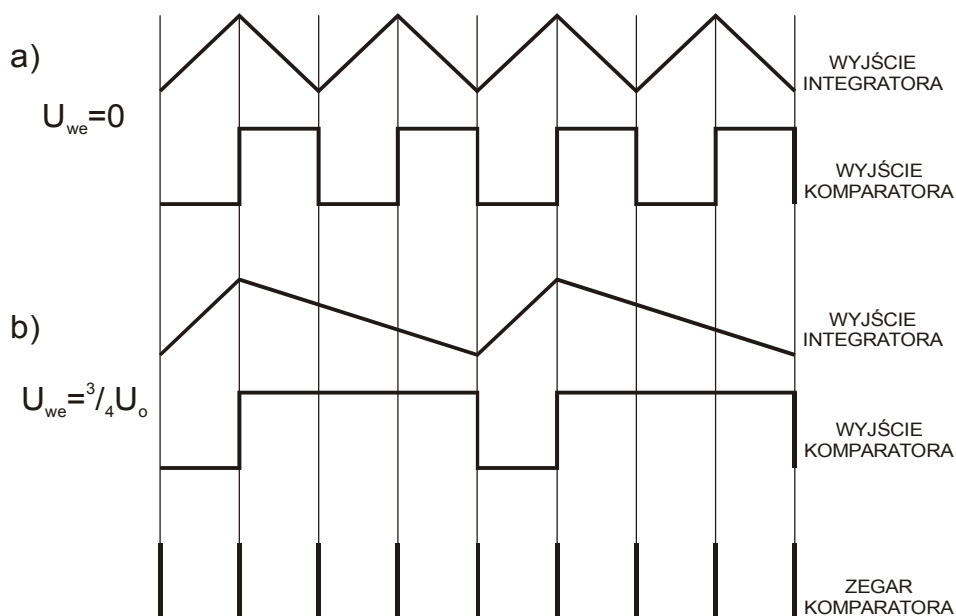


Rysunek 3.11. Struktura przetwornika analogowo-cyfrowego z modulacją sigma-delta.

dobierania elementów. Wysoką rozdzielczość uzyskuje się dzięki cyfrowej obróbce sygnałów, dokonywanej w filtrach cyfrowych wchodzących w skład przetworników.

Modulator sigma-delta, złożony z integratora, komparatora i jednobitowego przetwornika cyfrowo-analogowego zamienia sygnał wejściowy przetwornika w ciąg impulsów zero-jedynkowych o częstotliwości zgodnej z częstotliwością zegara komparatora. Sygnały wejściowe jednobitowego przetwornika cyfrowo-analogowego są odejmowane od sygnału wejściowego, a działanie pętli sprzężenia zwrotnego o odpowiednio dużym wzmocnieniu sprawia, że w dłuższym okresie czasu wartość średnia ciągu sygnałów wyjściowych komparatora odpowiada wartości średniej sygnału wejściowego.

Filtr cyfrowy, stanowiący istotną część przetwornika, pełni również funkcję decymatora, zamieniającego szeregowy strumień sygnałów binarnych na równoległe słowa, generowane z odpowiednio mniejszą częstotliwością, stanowiącą podwielokrotność zegara próbkującego komparator. Stosunek częstotliwości próbkowania komparatora do podwojonej szerokości pasma przenoszenia filtru decymacyjnego nazywamy współczynnikiem nadpróbkowania (*oversampling ratio*). Zazwyczaj może on być programowany przez użytkownika i jest jednym z ważniejszych parametrów, określających właściwości przetwornika. Na rysunku 3.12 przedstawiono sygnały w przetworniku z modulacją sigma-delta.



Rysunek 3.12. Sygnały w przetworniku z modulacją sigma-delta przy różnym sygnale wejściowym, a) zerowy sygnał wejściowy, b) sygnał wejściowy o wartości $\frac{3}{4}$ napięcia odniesienia.

Przetworniki nadpróbkujące przeznaczone do celów pomiarowych z reguły zawierają programowane cyfrowo układy kondycjonowania sygnałów wejściowych. Wykorzystanie tych układów do wzmacniania sygnałów pomiarowych znacznie upraszcza dołączanie czujników i budowę systemów pomiarowych. Inne cechy charakterystyczne dla tej grupy przetworników [7]:

- modulatory sigma-delta z podwójnymi i potrójnymi integratorami,
- wysoka rozdzielczość nominalna (zazwyczaj 24 bity),
- szeregowo wejścia i wyjścia sygnałów cyfrowych,
- możliwość synchronizacji cykli formowania wyników za pomocą zewnętrznego sygnału,
- rozbudowane systemy automatycznej, cyfrowej kalibracji i korekcji wyników.

Dodatkowymi cechami, występującymi tylko w niektórych typach przetworników są [7]:

- wbudowane źródła napięcia odniesienia, przeznaczone do zasilania mostków tensometrycznych,
- wbudowane, cyfrowo sterowane multipleksery wejściowe,
- dodatkowe źródła prądowe, przeznaczone do badania ciągłości dołączonych do przetwornika obwodów pomiarowych.

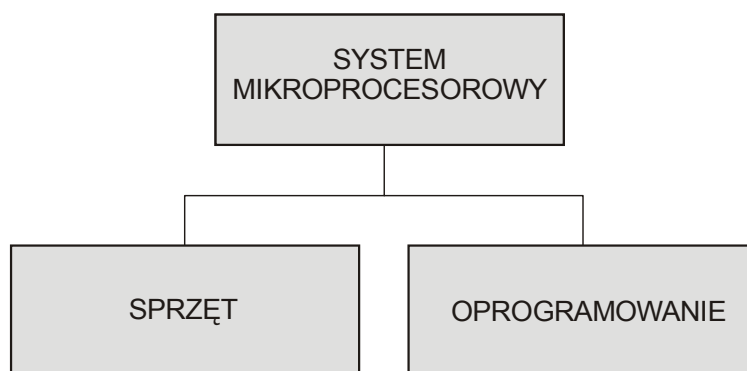
4. Technika mikrokontrolerowa.

4.1. Wprowadzenie.

Technika mikroprocesorowa zajmuje się przetwarzaniem danych w oparciu o cyfrowe programowalne układy scalone. Systemy przetwarzające dane w oparciu o takie układy nazywane są systemami mikroprocesorowymi (*microprocessor systems*). Technika mikroprocesorowa łączy dwie oddzielne dziedziny [3]:

- elektronikę w zakresie cyfrowych układów scalonych,
- informatykę w zakresie programowania tych układów.

Zatem systemy mikroprocesorowe to nie tylko sprzęt (*hardware*) ale i oprogramowanie (*software*), jak na rysunku 4.1.



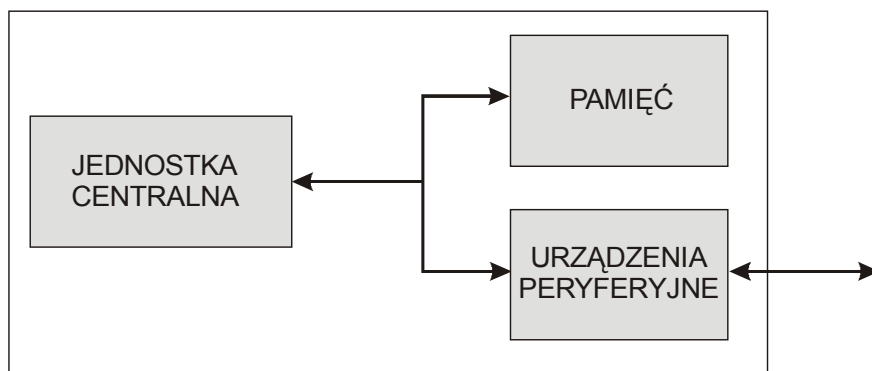
Rysunek 4.1. Elementy systemu mikroprocesorowego.

Określenie sprzęt oznacza płytę drukowaną (*printed board*) z umieszczonymi na niej układami scalonymi techniki mikroprocesorowej, a także innymi elementami i układami elektronicznymi.

Określenie oprogramowanie oznacza opracowany przez twórców systemu algorytm jego działania wynikający z założonej funkcji użytkowej systemu. Fizycznie algorytm ten znajduje się w jednym z elementów sprzętowych systemu, tj. w pamięci.

Podstawową dziedziną techniki mikroprocesorowej jest budowa systemów przetwarzających dane, tj. komputerów (rysunek 4.2).

Przedstawiona budowa komputera według koncepcji zaproponowanej jeszcze w 1944 roku przez Johanna von Neumanna jest aktualna do chwili obecnej [3]. Najważniejszą częścią komputera jest jednostka centralna (*CPU*, *Central Processor Unit*). Zadaniem jednostki centralnej jest szeregowe przetwarzanie danych poprzez kolejne wykonywanie rozkazów



Rysunek 4.2. Budowa komputera według koncepcji Johanna von Neumanna.

zapisanych przez użytkownika w postaci programu (*program code*) umieszczonego w pamięci komputera (*memory*). Program operuje na danych (*program data*), które również znajdują się w pamięci komputera. Z punktu widzenia użytkownika najbardziej istotną cechą jednostki centralnej jest jej lista rozkazów (*instruction set*). Lista rozkazów to zbiór rozkazów elementarnych operacji typu dodaj dwie liczby czy przesun zawartość komórki pamięci z jednego obszaru pamięci do innego. Program użytkownika zapisany w pamięci może składać się tylko z rozkazów zawartych w liście rozkazów danej jednostki centralnej. Komunikacja komputera z otoczeniem odbywa się za pomocą urządzeń peryferyjnych (*peripheral units*), zwanych także układami wejścia wyjścia (*input-output units*).

Podstawową właściwością części sprzętowej komputera – jak każdego urządzenia wykonanego w technice cyfrowej – jest to, że wszystkie występujące w nim sygnały mogą przyjmować w sposób stabilny tylko jedną z dwóch wartości umownie oznaczonych jako 0 i 1. W porównaniu z układami analogowymi układy cyfrowe mają dwie ważne zalety:

- ich realizacja elektroniczna jest łatwiejsza,
- są bardziej odporne na zakłócenia.

4.2. Podstawowe elementy systemów mikroprocesorowych.

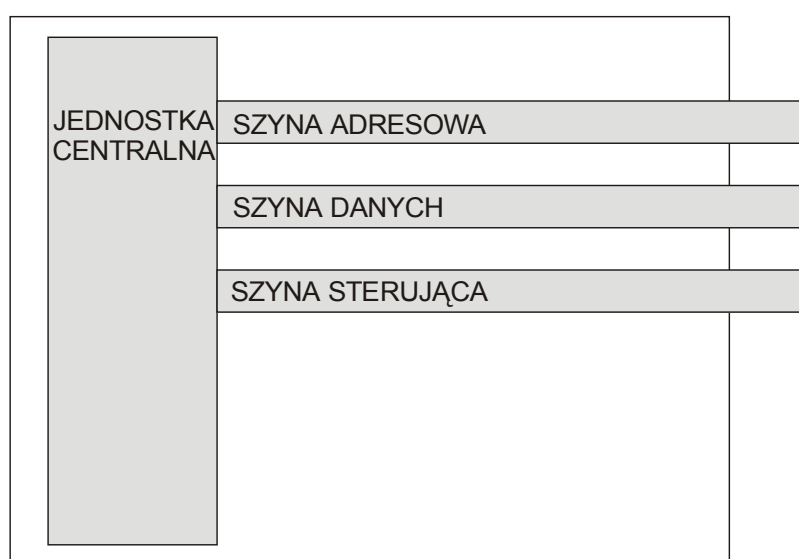
Najważniejszymi układami scalonymi techniki mikroprocesorowej są te elementy, które mają możliwość przetwarzania danych. Takim elementem jest jednostka centralna. Produkowane są dwie podstawowe grupy układów techniki mikroprocesorowej, zawierające w swojej strukturze jednostki centralne. Są to [3]:

- mikroprocesory (*microprocessor*),
- mikrokontrolery (*microcontroller*).

Mikroprocesor, to układ scalony w strukturze którego zintegrowana jest kompletna jednostka centralna komputera (rysunek 4.3). Mikroprocesor komunikuje się z otoczeniem za pomocą trzech zespołów sygnałów cyfrowych, tzw. szyn (*bus*). Są to:

- szyna danych (*data bus*),
- szyna adresowa (*adress bus*),
- szyna sterująca (*control bus*).

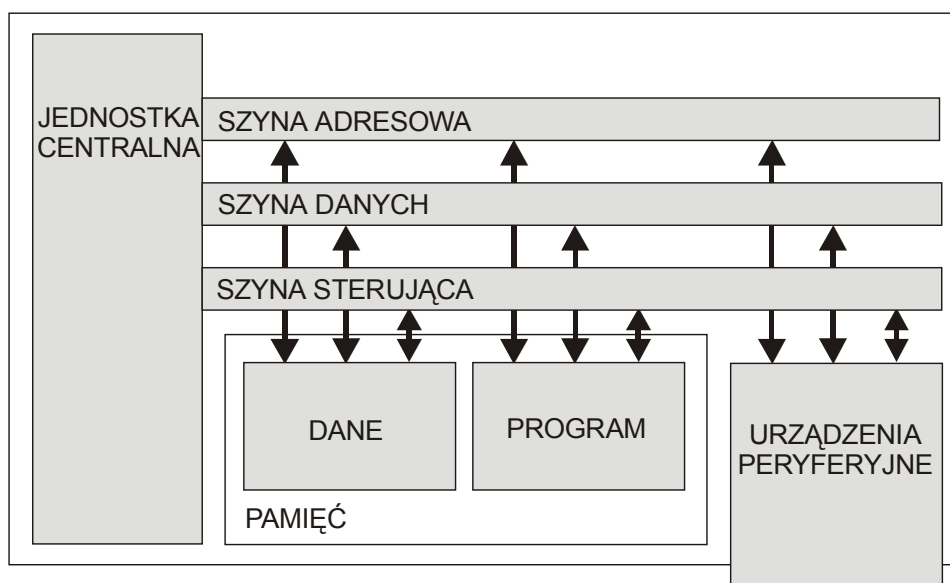
Zespół trzech szyn określany jest jako szyny systemowe. Za ich pomocą przyłączane są do mikroprocesora dwa pozostałe elementy niezbędne do budowy komputera: pamięć ora urządzenia peryferyjne.



Rysunek 4.3. Budowa mikroprocesora.

Mikrokontroler to układ scalony, w którego strukturze zintegrowane są wszystkie elementy kompletnego komputera: jednostka centralna, pamięć oraz urządzenia peryferyjne (rysunek 4.4). Jest to zatem komputer w jednym układzie scalonym. Z tego powodu uzasadnione jest określenie tego układu nazwą mikrokomputer jednoukładowy. Mikrokontroler komunikuje się z otoczeniem za pośrednictwem wewnętrznych urządzeń peryferyjnych.

Jest jeszcze jedna istotna różnica między mikroprocesorem a mikrokontrolerem. Zadania, które wykonuje komputer stosowany w układach sterowania są zazwyczaj niezbyt złożone pod względem obliczeniowym, natomiast często wymagają szybkiej reakcji na sygnały zewnętrzne. Funkcję tę realizuje przyporządkowany jednostce centralnej tzw. układ przerwań (*interrupt unit*). W mikrokontrolerach jest on zazwyczaj szybszy i bardziej rozbudowany niż w mikroprocesorach.



Rysunek 4.4. Budowa mikrokontrolera.

4.3. Budowa i architektura mikrokontrolerów.

W strukturze mikrokontrolera zawarte są zarówno układy peryferyjne, jak i układy pamięci. Jednakże w praktyce systemy mikroprocesorowe oparte na mikrokontrolerach bardzo często wykorzystują zarówno pamięć zewnętrzną, jak i zewnętrzne układy peryferyjne. Jest to spowodowane przyczynami [3]:

- zapotrzebowanie na pamięć nowoczesnych mikrokontrolerów jest bardzo duże, nawet rzędu megabajtów, wynika to z dużych mocy obliczeniowych i dużych przestrzeni adresowych oferowanych szczególnie przez mikrokontrolery 16- i 32-bitowe,
- współczesne mikrokontrolery oferują olbrzymią różnorodność urządzeń peryferyjnych zawartych w strukturach tych układów, mimo to w przeważającej części systemów mikroprocesorowych opartych na mikrokontrolerach wykorzystuje się układy scalone zawierające dodatkowe urządzenia peryferyjne.

Opisana sytuacja powoduje, że znaczna część mikrokontrolerów musi umożliwiać przyłączanie zewnętrznych układów scalonych zawierających pamięć lub układy peryferyjne. Przyłączanie zewnętrznych przez porty nie zawsze jest możliwe. Układy pamięci, a także niektóre układy peryferyjne przystosowane są do przyłączenia poprzez szyny systemowe, a nie przez porty. Dlatego mikrokontrolery dzielimy ze względu na swoją strukturę na [3, 12]:

- mikrokontrolery nie udostępniające szyn systemowych,

- mikrokontrolery udostępniające szyny systemowych poprzez wyprowadzenia portów,
- mikrokontrolery udostępniające bezpośrednio szyny systemowe.

Ważniejsze elementy, z których zbudowane są współczesne systemy mikroprocesorowe to [3, 12]:

- jednostki centralne wraz z układem przerwań,
- pamięci,
- urządzenia peryferyjne.

Elementy mogą być realizowane zarówno w postaci osobnych układów scalonych, jak i znajdować się w strukturze jednego układu scalonego wraz z innymi elementami. Przykładem jest pamięć systemów mikroprocesorowych, która umieszczana jest zarówno w strukturze mikrokontrolerów, jak i tworzona jako osobne układy scalone.

JEDNOSTKI CENTRALNE.

Z przedstawionych opisów mikrokontrolera i mikroprocesora jasno wynika, że najważniejszą częścią zarówno mikroprocesora, jak i mikrokontrolera jest jednostka centralna. Znaczenie jednostki centralnej polega na tym, że w systemie mikroprocesorowym jest to jedyny układ w sposób aktywny przetwarzający dane. Podstawowe własności jednostki centralnej:

- jednostka centralna należy do grupy układów cyfrowych synchronicznych i sekwencyjnych,
- synchroniczność oznacza, że wszystkie operacje wykonywane przez jednostkę centralną odbywają się w rytmie narzuconym przez zewnętrzny układ generujący sygnał periodyczny o stałej częstotliwości, sygnał ten nosi nazwę sygnału zegarowego (*clock*),
- sekwencyjność oznacza, że stan wyjść jednostki centralnej nie zależy tylko od stanu wejść, ale i od poprzednich stanów tego układu; jest to cech układów cyfrowych z pamięcią; jednostka centralna ma własną pamięć potrzebną np. do przechowywania argumentów rozkazów niezbędnych do wykonania na nich określone operacji; własne komórki pamięci jednostki centralnej noszą nazwę rejestrów (*register*).

Działanie jednostki centralnej polega na cyklicznym wykonywaniu rozkazów zawartych w programie użytkownika przechowywanym w pamięci systemu mikroprocesorowego. Cykl wykonywania rozkazu rozpoczyna się zawsze od wczytania do wewnętrznych rejestrów jednostki centralnej komórki pamięci zawierającej kod kolejnego rozkazu. Może się zdarzyć, że do wykonania rozkazu niezbędne są argumenty. W takim przypadku w następnych komórkach zawarte są informacje, w jakich miejscach pamięci argumenty są przechowywane. Kolejną

fazą wykonywania rozkazu jest pobranie tych argumentów i umieszczenie ich w odpowiednich wewnętrznych rejestrach jednostki centralnej. Po tym następuje wykonanie kolejnego rozkazu.

Lista operacji wykonywanych przez jednostkę centralną jest ograniczona i wynika z jej budowy sprzętowej. Zbiór wszystkich możliwych operacji nosi nazwę listy rozkazów jednostki centralnej. Rozkazy są prostymi działaniami elementarnymi np. przesunięcie zawartości komórki pamięci, operacje arytmetyczne.

Budowa funkcjonalna jednostki centralnej określana jest także nazwą architektury tej jednostki. Często spotykane pojęcia odnoszące się do jednostek centralnych to architektura typu CISC i architektura typu RISC.

W dużym uproszczeniu jednostki centralne o architekturze CISC (*Complex Instruction Set Computer*) charakteryzują się rozbudowaną listą rozkazów (często powyżej 100 rozkazów [3]) oraz większym stopniem złożoności operacji rozkazów. Wykonywanie rozkazów w jednostce o architekturze CISC trwa względnie długo.

Obecnie w nowych jednostkach centralnych stosuje się architekturę RISC (*Reduced Instruction Set Computer*). Lista rozkazów w jednostce centralnej o tej architekturze obejmuje zazwyczaj tylko kilkadziesiąt rozkazów. Wykonanie rozkazu w jednostce o architekturze RISC trwa jednak znacznie krócej niż w jednostce o architekturze CISC.

Jednostki centralne zasadniczo przetwarzają dane w sposób szeregowy. Oznacza to, że dopiero po skończeniu jednego rozkazu rozpoczyna się wykonywanie następnego. W nowszych jednostkach centralnych w celu przyspieszenia przetwarzania danych odchodzi się częściowo od tej zasady. Niektóre etapy wykonywania dwóch rozkazów następują równolegle w czasie. Jednostki centralne o takich własnościach określane są mianem jednostek o architekturze potokowej (*pipeline*) [3, 12].

Najczęściej stosowaną miarą prędkości przetwarzania danych przez jednostkę centralną jest liczba milionów rozkazów wykonywanych w czasie 1 sekundy. Jest to moc obliczeniowa wyrażona w jednostkach MIPS (*Milion Instruction Per Second*). Moc obliczeniowa jednostek o architekturze RISC jest z reguły większa od jednostek CISC.

UKŁADY PRZERWAŃ.

Innym ważnym pojęciem związanym z jednostką centralną jest układ przerwań (*interrupt unit*). Układ przerwań umożliwia czasowe wstrzymanie aktualnie wykonywanych sekwencji programu użytkownika i przejście do innej sekwencji programu związanej z obsługą określonego zdarzenia, zaistniałego

najczęściej w urządzeniach peryferyjnych. Celowość istnienia układu przerwań można zilustrować następującym przykładem: zadaniem danego systemu mikroprocesorowego jest szybka reakcja na określone wydarzenie, np. na przekroczenie dozwolonej temperatury kontrolowanej przez system. Przekroczenie tej temperatury uwidacznia się np. poprzez zmianę stanu określonego wejścia cyfrowego w układach peryferyjnych. Ponieważ nie można z góry określić, kiedy nastąpi przekroczenie temperatury, jedyną metodą pozwalającą na szybkie wykrycie takiej sytuacji jest możliwe częste odczytywanie przez jednostkę centralną stanu takiego wejścia. Oznacza to jednak dylemat dla programisty: częste odczytywanie gwarantuje wprowadzenie wykrycia przekroczenia temperatury z małym opóźnieniem, ale znaczna część czasu jednostki centralnej poświęcona jest na operacje odczytywania danego wejścia. O ile przekroczenie w danym okresie czasu nie wystąpi, odczytywanie będzie właściwie zbędne.

Znacznie lepszym rozwiązaniem byłoby stworzenie możliwości samoczynnego zgłaszania jednostce centralnej przez urządzenie peryferyjne wystąpienia określonego zdarzenia. Taką możliwość daje układ przerwań. Problem polega tylko na tym, że takie zgłoszenie następuje zawsze w trakcie wykonywania jakiejś sekwencji programu użytkownika. Schemat obsługi przerwania jest następujący [12]:

- urządzenie peryferyjne wykrywa wystąpienie określonego zdarzenia wymagającego reakcji systemu mikroprocesorowego,
- urządzenie peryferyjne zgłasza jednostce centralnej, za pośrednictwem układu przerwań, żądanie przerwania,
- jednostka centralna przerywa aktualnie wykonywaną sekwencję programu i zachowuje w specjalnym obszarze pamięci, tzw. stosie (*stack*) wszystkie dane związane z aktualnie wykonywanym programem,
- jednostka centralna przechodzi do wykonywania programu napisanego specjalnie przez użytkownika do obsługi danego wydarzenia,
- po wykonaniu tego programu jednostka centralna odtwarza ze stosu dane związane z przerwany programem i kontynuuje wykonywanie przerwano programu.

Dosyć często występuje w praktyce sytuacja, w której program obsługi przerwania jest przerywany przez inne przerwanie o wyższym priorytecie (*priority level*).

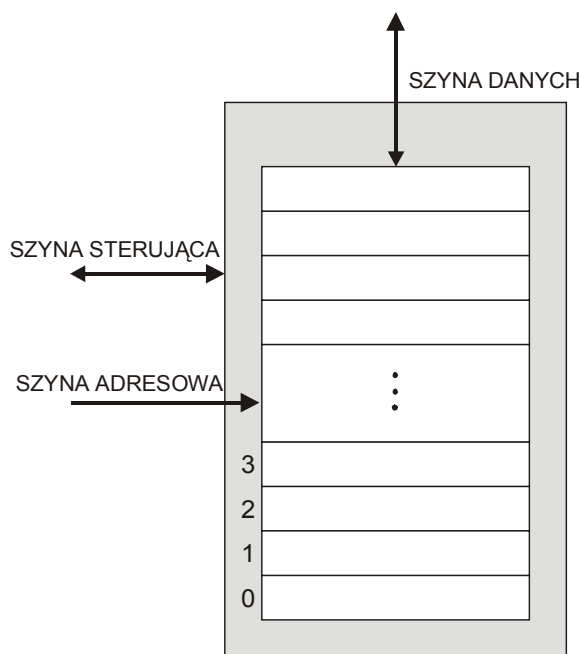
PAMIĘCI.

Podstawowe elementy techniki to te, które są niezbędne do budowy komputera. Oprócz jednostki centralnej i urządzeń peryferyjnych są to pamięci.

Układy pamięci (rysunek 4.5) kontaktują się z otoczeniem poprzez szynę danych, szynę adresową i szynę sterującą. Szerokość szyny danych określa liczbę bitów zapisywanych w pamięci lub czytanych z pamięci w trakcie pojedynczej operacji komunikacji z pamięcią. Szyny danych obecnie produkowanych układów pamięci mają szerokość 8 lub 16 bitów. Szerokość szyny adresowej określa maksymalną liczbę komórek, które mogą być zapamiętane w danym układzie pamięci. Tę maksymalną liczbę, czyli tzw. pojemność pamięci można łatwo obliczyć przez podniesienie liczby 2 do potęgi równej szerokości szyny adresowej. Sygnały wchodzące w skład szyny sterującej są uzależnione od typu pamięci. Często są to trzy sygnały, oznaczone jako [3]:

- CE (*chip enable*),
- OE (*output enable*),
- WE (*write enable*).

Bez sygnału CE nie jest możliwe wykonanie jakiejkolwiek operacji na układzie pamięci. Stosowanie tego sygnału jest konieczne, ponieważ w danym systemie mikroprocesorowym używanych jest przeważnie kilka układów pamięci. Dzięki odpowiednim operacjom logicznym na poszczególnych liniach szyny adresowej uzyskuje się układ, w którym w danym zakresie przestrzeni adresowej aktywny jest tylko pojedynczy wybrany układ pamięci. Jest to niezbędne do poprawnego działania systemu mikroprocesorowego.



Rysunek 4.5. Struktura układów pamięci.

Sygnał OE używany jest podczas operacji czytania z pamięci. Po jego doprowadzeniu, na szynie danych układu pamięci pojawia się zawartość tej

komórki pamięci, na którą wskazywał stan szyny adresowej bezpośrednio przed

Własności pamięci	Stosowane typy pamięci
<p>Zawartość pamięci nie znika wraz z wyłączeniem napięcia zasilania.</p> <p>Z pamięci można czytać, lecz nie można do niej wpisywać danych. Umieszczenie danych wymaga specjalnego procesu, zwanego programowaniem.</p>	<p>ROM (<i>Read Only Memory</i>)</p> <ul style="list-style-type: none"> - programowanie zawartości pamięci następuje w procesie produkcyjnym i nie może być przeprowadzone przez użytkownika <p>EPROM (<i>Erasable Programmable ROM</i>)</p> <ul style="list-style-type: none"> - pamięci z możliwością kasowania dotychczasowej zawartości promieniami ultrafioletowymi i wprowadzania nowej zawartości za pomocą zewnętrznego programatora, umieszczane są w budowach z okienkiem kwarcowym w celu umożliwienia kasowania <p>OPT (<i>One Time Programmable</i>)</p> <ul style="list-style-type: none"> - pamięci typu EPROM umieszczanie w obudowach bez okienka kwarcowego; dlatego możliwe jest tylko jednokrotne zaprogramowanie pamięci bez możliwości skasowania jej zawartości <p>FLASH (<i>Bulk Erasable Non-Volatile Memory</i>)</p> <ul style="list-style-type: none"> - pamięci z możliwością kasowania i programowania bezpośrednio w systemie mikroprocesorowym
<p>Zawartość pamięci znika wraz z wyłączeniem napięcia zasilania.</p> <p>Z pamięci można zarówno czytać, jak i wpisywać do niej dane.</p>	<p>SRAM (<i>Static Random Access Memory</i>)</p> <ul style="list-style-type: none"> - pamięci RAM statyczne; są to pamięci o krótkich czasach dostępu, prostsze w obsłudze przez jednostkę centralną <p>DRAM (<i>Dynamic Random Access Memory</i>)</p> <ul style="list-style-type: none"> - pamięci RAM dynamiczne; są to pamięci, których obsługa przez jednostkę centralną jest bardziej skomplikowana; polega to na konieczności wykonywania w krótkich odstępach czasu określonych operacji na pamięci, tzw. odświeżania;
	<p>w przeciwnym razie dane zawarte w pamięci dynamicznej zanikają</p>

Tablica 4.1. Przegląd podstawowych układów pamięci.

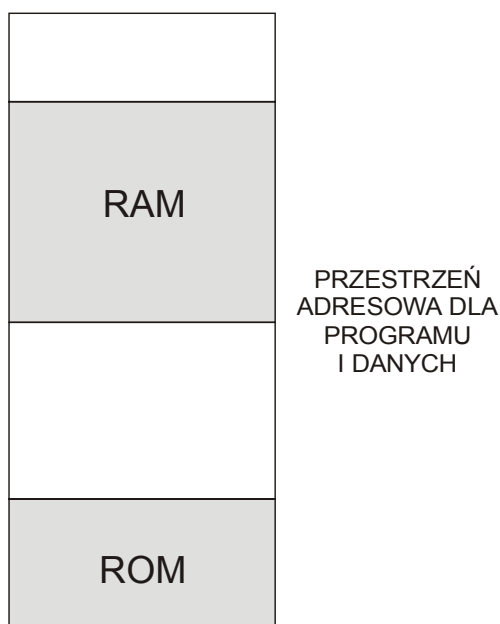
doprowadzeniem sygnału OE. Z czytaniem wiąże się ważny parametr układów pamięci, tzw. czas dostępu (*read acces time*). Parametr ten – podawany najczęściej w nanosekundach – określa czas opóźnienia między doprowadzeniem sygnału OE a pojawieniem się na szynie danych zawartości komórki pamięci. W najszybszych pamięciach produkowanych obecnie czas ten jest mniejszy od 10 ns.

Sygnał WR używany jest podczas operacji wpisywania do pamięci. Po jego doprowadzeniu do tej komórki pamięci, na którą wskazywał stan szyny adresowej bezpośrednio przed doprowadzeniem sygnału WR, wpisywana jest wartość zgodna z aktualnym stanem szyny danych.

Bardzo istotną kwestią związaną ze współpracą jednostek centralnych z pamięciami jest tzw. model pamięci.

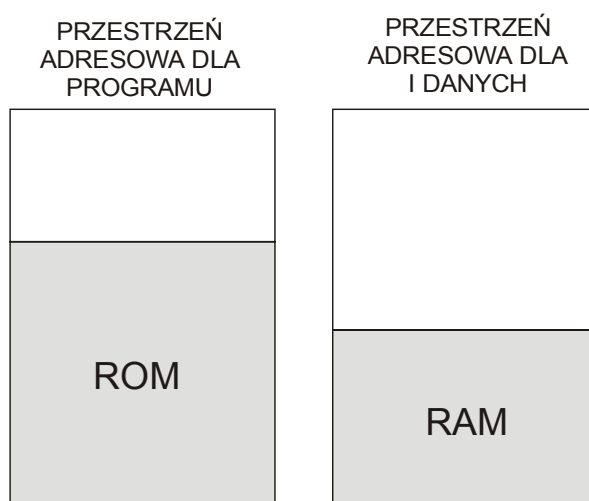
Jeżeli w systemie mikroprocesorowym jest kilka układów pamięci, to szyna danych dochodzi do każdego z nich. Jednakże operacja zapisu lub czytania do określonej komórki pamięci dotyczy tylko jednego układu. Zatem w danym momencie powinien być aktywny tylko jeden układ pamięci. Problem modelu pamięci sprowadza się do odpowiedzi na pytanie, w jaki sposób to osiągnąć.

Najbardziej oczywistą metodą rozwiązania tego problemu jest przypisanie każdej komórce pamięci niepowtarzalnego adresu. Warunkiem jest tu dostateczna szerokość szyny adresowej, tak aby była w stanie wygenerować liczbę adresów wystarczającą dla wszystkich komórek pamięci. Taki model pamięci nosi nazwę modelu liniowego lub modelu von Neumanna [3, 12] (rysunek 4.6).



Rysunek 4.6. Przykładowa przestrzeń adresowa w modelu pamięci typu von Neumanna.

W modelu typu Harvard [3, 12] umieszcza się układy zawierające pamięć programu i układy zawierające pamięć danych w osobnych przestrzeniach adresowych (rysunek 4.7). W modelu tym dwie komórki pamięci mogą mieć jednakowe adresy pod warunkiem, że leżą w innych przestrzeniach adresowych. Takie rozwiązanie tylko pozornie zmniejsza wymaganą wielkość przestrzeni adresowej, a zatem liczbę wyprowadzeń szyny adresowej. Zmniejszenie liczby wyprowadzeń szyny adresowej odbywa się kosztem zwiększenia wyprowadzeń szyny sterującej. Dodatkowe sygnały sterujące muszą być generowane przez jednostkę centralną w celu wskazania przestrzeni adresowej, w której dokonuje się operacji odczytu lub zapisu komórki pamięci o danym adresie.



Rysunek 4.7. Przykładowa przestrzeń adresowa w modelu pamięci typu Harvard.

Oba modele są wykorzystywane we współczesnych systemach mikroprocesorowych.

4.4. Urządzenie peryferyjne i wyposażenie mikrokontrolerów.

Podobnie jak w przypadku układów pamięci, urządzenia peryferyjne stosowane są zarówno w systemach zawierających mikroprocesory, jak i mikrokontrolery.

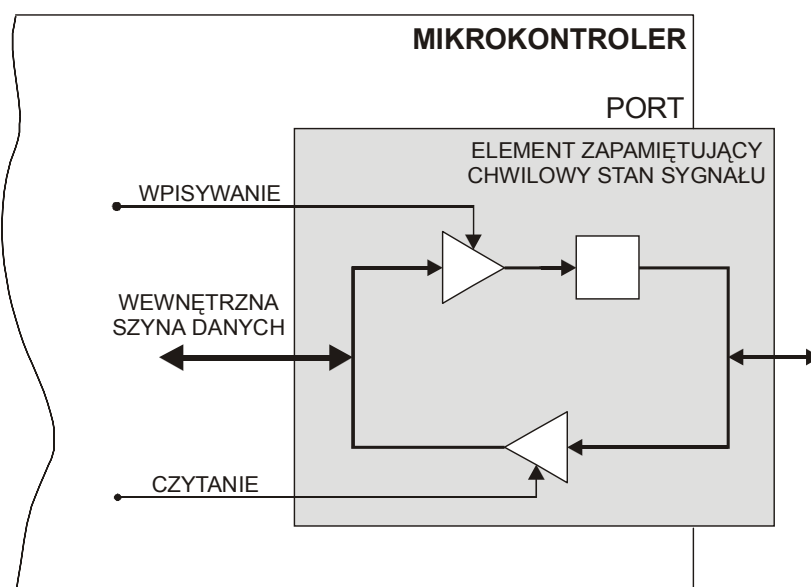
W przypadku mikroprocesorów jest to oczywiste, ponieważ nie ma tych urządzeń w strukturze układu scalonego mikroprocesora. Natomiast stosowanie zewnętrznych układów peryferyjnych w systemach zawierających mikrokontrolery jest niezbędne wówczas, gdy projektant systemu nie znalazł odpowiadającego mu układu zawierającego w strukturze urządzenia peryferyjne potrzebne w danym zastosowaniu.

Różnorodność urządzeń peryferyjnych jest tak duża, że trudno nawet dokonać ich klasyfikacji. Za podstawowe urządzenia peryferyjne można uznać te,

które zostały umieszczone w strukturze pierwszych mikrokontrolerów i umieszczane są w strukturach mikrokontrolerów produkowanych obecnie. Są to [3]:

- porty (*port*),
- układy czasowe (*timer/counter*),
- sterowniki komunikacji szeregowej (*serial communication controller*).

Porty są podstawowymi i najważniejszymi urządzeniami peryferyjnymi (rysunek 4.8). Wszystkie pozostałe urządzenia peryferyjne wykorzystują funkcję portów. Zadaniem portów jest doprowadzenie sygnałów binarnych z otoczenia mikrokontrolera do jego wnętrza oraz odwrotnie – wyprowadzenie sygnałów z wnętrza mikrokontrolera do jego otoczenia. Za transport informacji wewnątrz mikrokontrolera odpowiedzialna jest wewnętrzna szyna danych. Zatem czytanie danych przez port polega na doprowadzeniu chwilowych stanów wyprowadzeń portu (tzn. istniejących w momencie operacji czytania) do wewnętrznej szyny danych układu. Natomiast operacja wpisania do portu powoduje, że chwilowy stan wewnętrznej szyny danych zostaje zapamiętany na wyprowadzeniach portu. Stan wyprowadzeń portu pozostaje niezmienny, póki nie nastąpi kolejna operacja wpisywania do portu.



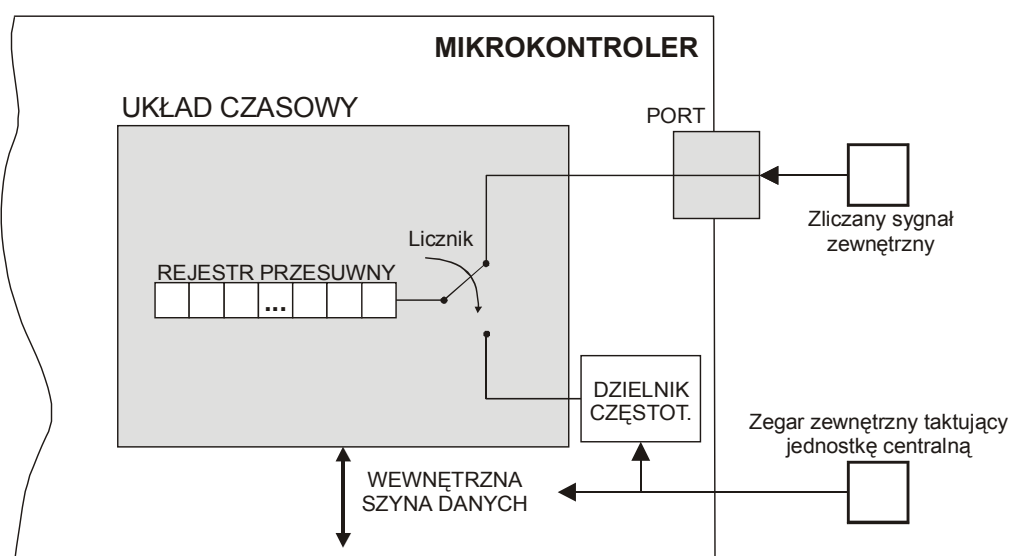
Rysunek 4.8. Schematyczna budowa portu.

Podstawowe układy czasowe to rejestr przesuwny o określonej długości. Układy czasowe mogą pracować w dwóch podstawowych funkcjach:

- jako właściwe układy czasowe (*timer*),
- jako liczniki (*counter*).

Jako układy czasowe są taktowane wewnętrznym sygnałem zegarowym przeznaczonym do taktowania jednostki centralnej. Właściwe układy czasowe są wykorzystywane w programie użytkownika jako wzorce czasu. W celu generowania wzorców czasu różnej długości wewnętrzny sygnał zegarowy, przed doprowadzeniem do układu czasowego, przechodzi przez programowalny dzielnik częstotliwości (rysunek 4.9).

W przypadku wykorzystywania jako liczniki, układy czasowe są taktowane zewnętrznymi sygnałami, doprowadzanymi przez wyprowadzenia portów. Służą wtedy do zliczania zmian poziomów sygnałów zewnętrznych.



Rysunek 4.9. Schematyczna budowa układu czasowego.

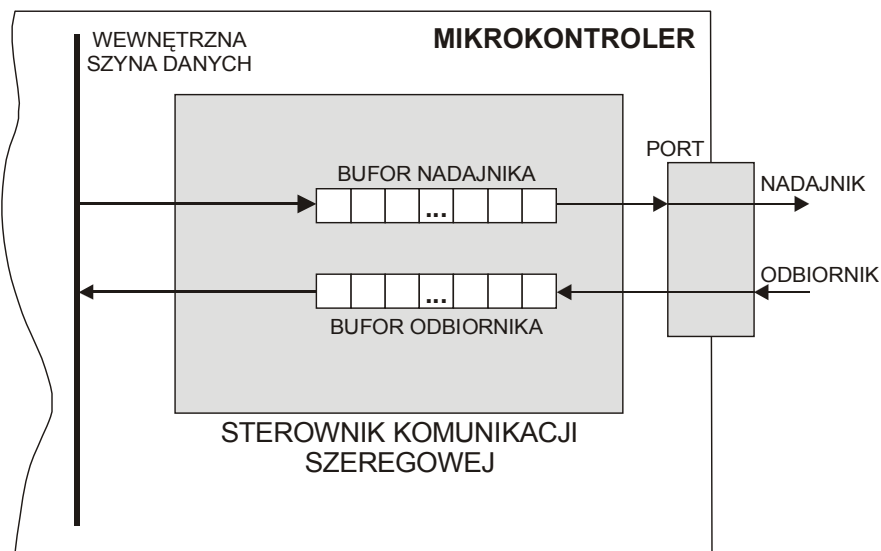
Rejestr przesuwany zliczający liczbę zmian poziomów doprowadzonych do niego sygnałów jest dostępny z poziomu programu użytkownika.

Ostatnim urządzeniem peryferyjnym należącym do grupy urządzeń podstawowych jest sterownik komunikacji szeregowej (rysunek 4.10). Urządzenie to umożliwia wysyłanie zawartości określonego rejestru, tzw. bufora nadajnika w postaci szeregowej poprzez określone wyprowadzenie portu. Oznacza to, że na wyjściu portu pojawia się ciąg binarny odpowiadający zawartości wysyłanego rejestru. Jest to funkcja nadajnika (*transmitter*). W funkcji odbiornika (*receiver*) sterownika komunikacji szeregowej potrafi przetworzyć ciąg binarny doprowadzony do wejścia określonej linii portu na zawartość rejestru, zwanego buforem odbiornika. Bufory nadajnika i odbiornika są dostępne z poziomu programu użytkownika.

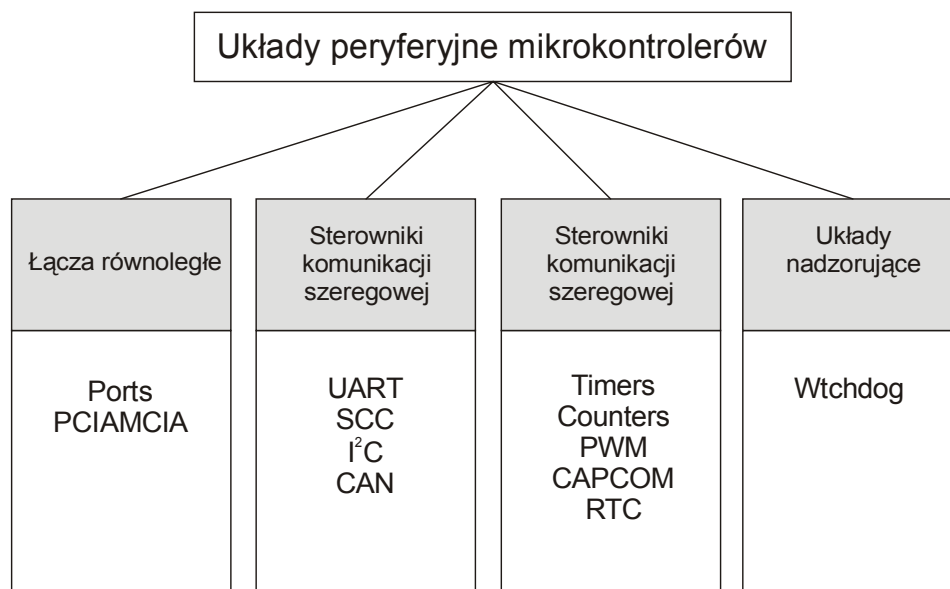
Rozwój wewnętrznych układów peryferyjnych mikrokontrolerów jest jednym z podstawowych kierunków ich rozwoju (rysunek 4.11). W strukturach pierwszych mikrokontrolerów umieszczane były tylko podstawowe urządzenia peryferyjne: porty, łącze komunikacji szeregowej oraz proste układy czasowe [3]. Od tego

czasu do chwili obecnej nastąpił znaczny postęp rozwoju wewnętrznych urządzeń peryferyjnych mikrokontrolerów:

- w strukturze mikrokontrolera integrowano więcej urządzeń peryferyjnych,
- są to urządzenia bardziej „inteligentne”, tzn. są w stanie samoczynnie realizować złożone funkcje bez ingerencji jednostki centralnej,
- są bardziej wyspecjalizowane.



Rysunek 4.10. Schematyczna budowa łącza komunikacji szeregowej.



Rysunek 4.11. Klasyfikacja wewnętrznych urządzeń peryferyjnych mikrokontrolerów oraz przykłady urządzeń należących do poszczególnych grup.

5. Środowiska programowe mikrokontrolerów przemysłowych.

5.1. Wprowadzenie.

W ostatnich latach w technice mikrokontrolerów na pierwszy plan wysunęły się sprawy programowania. Nowe możliwości w zakresie tworzenia oprogramowania pojawiły się z chwilą wprowadzenia mikrokontrolerów 32-bitowych i związanym z tym skokowym wzrostem oferowanej mocy obliczeniowej oraz dostępnej wielkości pamięci. Tworzenie programów dla tych mikrokontrolerów przez jednego programistę oraz stosowanie asemblera okazało się niemożliwe. Naturalną możliwością było zatem sięgnięcie do metod i narzędzi programowania opracowanych już wcześniej systemów obliczeniowych opartych na procesorach 32-bitowych, głównie dla komputerów osobistych.

Takimi metodami i narzędziami programowania były systemy operacyjne umożliwiające jednoczesną pracę wielu programistów oraz stosowanie języków programowania wyższego poziomu (głównie języka C). Bezpośrednie zastosowanie tych narzędzi programowych do programowania mikrokontrolerów nie jest jednak możliwe. Systemy operacyjne stosowane w systemach obliczeniowych nie gwarantują rozpoczęcia wykonywania określonego zadania w narzuconym, nieprzekraczalnym czasie [3]. Z kolei zastosowanie niektórych języków programowania wyższego poziomu, z punktu widzenia programisty jest wprawdzie bardzo wygodne, ale generowany w taki sposób kod programu jest bardzo duży [5, 12].

Aktualną sytuację w zakresie programowania mikrokontrolerów można określić jako proces adaptacji metod i narzędzi stosowanych w programowaniu systemów obliczeniowych do wymagań systemów pracujących w zastosowaniach pomiarowo-kontrolnych i komunikacyjnych.

Programy narzędziowe służące do programowania mikrokontrolerów są łączone w jeden pakiet programów. Takie rozwiązanie było stosowane już wcześniej do programowania komputerów osobistych i rozwinęło się wraz z wprowadzeniem systemu Windows. Idea takiego pakietu polega na umożliwieniu realizacji wszystkich etapów cyklu rozwojowego oprogramowania dla mikrokontrolera w ramach jednego programu. Podobnie jak dla komputerów osobistych, środowisko takie określane jest skrótem IDE (*Integrated Development Enviroment*). Dla podkreślenia ukierunkowania takiego środowiska na programowanie mikrokontrolerów czasami określane jest ono także jako EDE (*Embedded Development Enviroment*). Różnice między IDE dla mikrokomputerów osobistych a mikrokontrolerów nie są duże. Sprowadzają się do integracji w IDE specyficznych narzędzi dla uruchamiania programów np. symulatorów programowych mikrokontrolerów.

Środowiska uruchomieniowe rezydują w systemie obliczeniowym. Z reguły jest to komputer osobisty pracujący w systemie operacyjnym Windows lub stacja robocza pracująca w jednej z wersji systemu operacyjnego UNIX [3, 12].

Przykładowe korzyści wynikające ze zastosowania środowisk uruchomieniowych to:

- łatwe zarządzanie procesem rozwoju oprogramowania za pomocą tzw. projektów (*projects*),
- elementem środowiska jest z reguły specjalizowany edytor tekstu, jego zaletą w porównaniu ze zwykłymi edytorami jest m.in. to, że wspiera tworzenie programu w jakimś konkretnym języku programowania; jeżeli program tworzony jest w C, edytor zaznacza różnymi kolorami słowa kluczowe języka C oraz komentarze, zwiększa to czytelność programu i zmniejsza prawdopodobieństwo popełnienia przypadkowych błędów.

5.2. Cechy i cykl programowania mikrokontrolera.

Tworzenie oprogramowania dla systemów mikroprocesorowych opartych na mikrokontrolerach i ukierunkowanych na zadania pomiarowo-sterujące oraz komunikacyjne różni się znacznie od tworzenia oprogramowania dla systemów obliczeniowych (tablica 5.1). Główne cechy programów dla mikrokontrolerów [3, 5]:

- program jednoznacznie ustala funkcję systemu mikroprocesorowego, tzn. użytkownik ma możliwość zmiany funkcji systemu tylko w niewielkim zakresie przewidziany przez program użytkowy, ta właśnie cecha określana jest jako zagnieżdżenie programu (*embedded program*),
- działanie programu musi spełniać określone wymagania czasowe dotyczące nie przekraczania maksymalnego czasu reakcji na określone zdarzenia zewnętrzne oraz realizacji określonych zadań programowych w nieprzekraczalnym czasie, ta cecha określana jest jako praca programu w czasie rzeczywistym,
- są to programy działające na specyficznych zasobach sprzętowych warunkowanych ukierunkowaniem budowy sprzętowej systemu mikroprocesorowego na konkretne zadanie.

Cykl tworzenia oprogramowania dla mikrokontrolerów jest zbliżony do cyklu tworzenia oprogramowania dla komputerów osobistych. Cykl ten obejmuje trzy fazy [4, 5]:

- napisanie kodu źródłowego programu,
- przetłumaczenie kodu źródłowego na kod maszynowy,
- uruchomienie programu na systemie docelowym.

Cykl tworzenia oprogramowania dla mikrokomputerów różni się jednak od cyklu programowania komputerów osobistych w ważnym punkcie. W komputerach osobistych wszystkie fazy cyklu rozwoju oprogramowania wykonywane są w jednym systemie. Komputer osobisty jest jednocześnie narzędziem za pomocą którego tworzone jest oprogramowanie, jak i systemem, który wykonuje ten program. W przypadku mikrokontrolerów faza pierwsza i druga wykonywana jest najczęściej przy zastosowaniu komputera osobistego. Komputer w takiej funkcji określany jest jako system rozwojowy (*development system*). Faza trzecia realizowana jest na rzeczywistym systemie zawierającym mikrokontroler. System ten określany jest jako system docelowy (*target system*) [3].

Etap	Czynności wykonywane w ramach etapu	Stosowane programy narzędziowe	Format kodu wyjściowego etapu
Napisanie kodu źródłowego programu	Podział programu na moduły i napisanie kodu źródłowego poszczególnych modułów w języku assemblera lub wyższego poziomu	Edytor tekstu	Kod ASCII
Przetłumaczenie kodu źródłowego na kod maszynowy	Przetłumaczenie kodów źródłowych poszczególnych modułów	Asembler i kompilator skrośny	Kod wynikowy relokowalny
	Łączenie kodów poszczególnych modułów w kod programu	Linker	Kod wynikowy absolutny
Uruchomienie programu w systemie docelowym	Przeniesienie kodu programu do pamięci systemu docelowego	Programator pamięci	Kod maszynowy
	Kontrola poprawności działania programu i korekcja błędów	Specjalistyczny sprzęt i oprogramowanie	

Tablica 5.1. Etapy tworzenia oprogramowania dla mikrokontrolerów.

5.3. Asembler.

Język ten to tzw. język niższego poziomu. Operuje bezpośrednio na liście rozkazów danej jednostki centralnej. Są to operacje typu: przenieś zawartość komórki o określonym adresie w pamięci zewnętrznej do określonego rejestru wewnętrznego jednostki centralnej. Programista nie pisze jednak programu w kodzie maszynowym. Bezpośrednie posługiwanie się ciągami zer i jedynek byłoby zbyt uciążliwe. Dlatego w języku assemblera każdy rozkaz z listy rozkazów jednostki centralnej ma przyporządkowaną krótką nazwę, tzw. mnemonik. Nazwa

ta jest tak dobrana, aby kojarzyła się z funkcją danego rozkazu. Na przykład rozkazy związane z przesunięciem komórek z jednego obszaru pamięci do drugiego mają najczęściej mnemoniki o nazwie LD (*ang. load – załadować*) lub MOV (*ang. move – przesunąć*). Zadaniem programu tłumaczącego jest proste przetłumaczenie mnemoników na odpowiadający im kod maszynowy. Stąd określenie język niższego poziomu. Program tłumaczący z języka asemblera określany jest jako asembler (*assembler*) [3, 5].

5.4. Inne środowiska.

Możliwość druga polega na stosowaniu języka programowania wyższego poziomu. Języki programowania wyższego poziomu są w zasadzie niezależne od listy rozkazów konkretnej jednostki centralnej. Programista operuje rozkazami wykonującymi bardziej złożone operacje niż język asemblera. Na przykład dla porównania dwóch liczb programista w języku wyższego poziomu używa bezpośrednio znaku relacji < lub >. W języku asemblera ta prosta operacja musi być realizowana za pomocą kilku lub nawet kilkunastu rozkazów w zależności od wielkości porównywanych liczb. Program tłumaczący z języka programowania nosi nazwę kompilatora (*compiler*).

W języku asemblera każda linia programu tłumaczona jest na odpowiedni rozkaz jednostki centralnej mikrokontrolera. W języku wyższego poziomu każda linia programu tłumaczona jest na kilka, kilkanaście a nawet kilkadziesiąt rozkazów asemblera [5]. Ten ostatni przypadek ma miejsce np. wówczas gdy w programie zastosowane są działania na liczbach zmiennoprzecinkowych, zaś jednostka centralna mikrokontrolera nie wspiera operacji zmiennoprzecinkowych.

Możliwość przenoszenia kodu źródłowego między różnymi mikrokontrolerami staje się coraz ważniejsza. Różne jednostki centralne mają różne listy rozkazów, zatem program napisany w języku asemblera jest związany na stałe z daną jednostką centralną. Program napisany w języku programowania wyższego poziomu jest w niewielkim stopniu związany z daną jednostką centralną. W celu uruchomienia tego programu w systemie zawierającym inny mikrokontroler wystarczą na ogół niewielkie modyfikacje kodu źródłowego, a następnie przetłumaczenie tego kodu na kod maszynowy przy zastosowaniu narzędzi programowych właściwych dla danego kompilatora. Zmiany w kodzie muszą być znacznie większe, jeżeli obydwa mikrokontrolery wykorzystują inne układy peryferyjne.

Szerszego komentarza wymaga kwestia optymalności programu pisanego w języku asemblera i języku wyższego poziomu. Ogólna zasada brzmi: program może być pisany tym bardziej optymalnie, im bardziej pracuje na rozkazach zbliżonych do list rozkazów danego procesora. Zasada ta sprawdza się w przypadku programów krótkich. Programista znający dobrze architekturę i listę

rozkazów jednostki centralnej, pisząc program w języku assemblera potrafi optymalnie wykorzystać wszystkie zasoby jednostki centralnej. Problem jednak polega na tym, że w miarę rozwoju techniki mikroprocesorowej programy dla bardziej zaawansowanych mikrokontrolerów stają się coraz dłuższe. Nawet doświadczony programista nie jest w stanie napisać optymalnie programu liczącego np. kilkanaście lub kilkadziesiąt linii kodu źródłowego [5]. W takim programie byłoby również bardzo trudne wykrywanie i usuwanie błędów. Program realizujący tę samą funkcję w języku wyższego poziomu byłby prawdopodobnie krótszy. Optymalnym wykorzystaniem zasobów jednostki centralnej (np. wykorzystaniem rejestrów wewnętrznych) zajmuje się w większym stopniu kompilator niż sam programista. Dlatego w ujęciu całościowym złożony program napisany w języku wyższego poziomu jest bardziej optymalny, jeżeli nawet niektóre pojedyncze fragmenty tego programu – gdyby były napisane w języku assemblera – byłyby bardziej optymalne niż odpowiadające im fragmenty w języku wyższego poziomu [3, 12].

Dominującym językiem wyższego poziomu używanym do programowania mikrokontrolerów jest język C. Proces jego standaryzacji w zasadzie został zakończony, a obecnie obowiązujący standard tego języka określa się mianem ANSI C. Jest to jednak standard przeznaczony głównie do systemów obliczeniowych pracujących pod kontrolą systemów operacyjnych.

Większość kompilatorów języka C przeznaczonych do programowania konkretnych mikrokontrolerów nie odpowiada dokładnie ANSI C, ale zawiera dodatkowe rozszerzenia tego języka ze specyficznymi wymaganiami przy programowaniu mikrokontrolerów. Najczęściej są to następujące elementy [3, 12]:

- wprowadzane są nowe typy zmiennych, np. zmienne bitowe lub zmienne umożliwiające bezpośredni dostęp do specjalnych rejestrów procesora, z drugiej strony w przypadku mikrokontrolerów 8 bitowych niektóre złożone typy zmiennych nie są przez kompilator w ogóle wspierane,
- przy deklaracji zmiennych wprowadzone są mechanizmy umożliwiające alokację (umiejscowienie) zmiennej w określonym miejscu przestrzeni adresowej,
- wprowadzane są nowe słowa kluczowe języka C.

Ogólnie różnice pomiędzy ANSI C a odmianami tego języka stosowanymi do programowania mikrokontrolera nie są duże. Ponadto w wielu przypadkach stosowanie rozszerzeń języka C nie jest konieczne i ten sam efekt osiąga się innymi metodami. Dotyczy to w szczególności mikrokontrolerów o liniowej przestrzeni adresowej. Na przykład alokacja zmiennych może zostać przeprowadzona nie bezpośrednio w kodzie źródłowym, ale poprzez odpowiednie ustawienia (dyrektywy) linkera.

Elementem programowania w języku C jest możliwość wykorzystywania gotowych funkcji wykonujących pewne standardowe zadania. Funkcje te są zawarte w tzw. bibliotekach i są dołączane do kodu wynikowego na etapie łączenia programu.

Pozycja języka C jako podstawowego narzędzia do pisania oprogramowania do mikrokontrolerów w chwili obecnej nie jest zagrożona [3]. Największe firmy specjalizujące się w pisaniu programów narzędziowych dla systemów zagnieżdżonych, niezależnie od udoskonalenia oferowanych programów narzędziowych dla języka C, badają możliwości zastosowania do programowania mikrokontrolerów innych języków programowania. Aktualnie w centrum zainteresowania są języki C++ oraz Java.

Język C++ jest rozszerzeniem języka C o elementy umożliwiające tzw. programowanie obiektowe. Jest to obecnie bardzo rozpowszechniony język programowania systemów obliczeniowych pracujących z wykorzystaniem systemu operacyjnego Windows. W przypadku złożonych programów zastosowanie języka C++ jest bardziej efektywne niż języka C. Bezpośrednie zastosowanie tego języka do programowania mikrokontrolerów jest trudniejsze niż w przypadku języka C. Problem nie dotyczy tylko – podobnie jak to ma miejsce w przypadku języka C – konieczności rozbudowy języka C++ o specyficzne elementy związane z programowaniem mikrokontrolera. Wygoda programowania w języku C++ okupiona jest dużą wielkością programów generowanych przez narzędzia programowe dla tego języka. Systemy mikroprocesorowe oparte na mikrokontrolerach nie dysponują zazwyczaj tak dużymi zasobami pamięci jak systemy obliczeniowe. Dlatego oferowane obecnie kompilatory języka C++ dla konkretnych mikrokontrolerów zawierają niepełną wersję tego języka. Wersje te pozbawione są tych elementów języka C++, których użycie najbardziej wpływa na zwiększenie kodu wynikowego programu. Ponadto kompilatory języka C++ oferowane są tylko dla mikrokontrolerów 16- i 32-bitowych, ponieważ mikrokontrolery 8-bitowe nie mają wystarczającej przestrzeni adresowej [12].

Java jest językiem programowania opracowanym przez firmę Sun Microsystems. Ideą tego języka programowania jest zapewnienie całkowitej niezależności programu napisanego w Javie od płaszczyzny sprzętowej, na której kod jest wykonywany. Dlatego Java nie jest tylko językiem programowania, zawiera następujące elementy [3]:

- język programowania (*programming language*),
- wielozadaniowy system operacyjny (*multithreading operating system*),
- wbudowany interpreter języka Java (*Java virtual machine*).

Dzięki tym właściwościom program napisany w języku Java może być pobrany bezpośrednio z dowolnego miejsca sieci rozległej i uruchomiony w dowolnym systemie zawierającym wymienione elementy tego języka.

Prace nad zastosowaniem języka Java do programowania mikrokontrolerów są jeszcze w początkowym etapie. Prace te wskazują, że zakres koniecznych zmian w obrębie Javy musi być znacznie większy niż w przypadku C++. W szczególności:

- podobnie jak w C++ niektóre elementy języka Java, np. dynamiczne usuwanie nieużywanych bloków pamięci (*garbage collection*), nie mogą być stosowane w programowaniu systemów zagnieżdżonych,
- zawarty w Javie system operacyjny nie zapewnia spełnienia wymagań pracy programu w narzuconym reżimie czasowym i musi być całkowicie przebudowany,
- wykonywanie programu metodą interpretacji ze względów czasowych jest niedopuszczalne i musi zostać zastąpione wykonywaniem programu po kompilacji.

Wymienione problemy powodują, że większość firm ogranicza się do opracowania kompilatora skrótnego ograniczonej wersji języka Java oraz umożliwia wykonywanie powstałych w ten sposób programów pod kontrolą własnych systemów operacyjnych [3].

Tablica 5.2 przedstawia porównanie własności programowania w języku assemblera i języku wyższego poziomu.

Cecha programu	Język assemblera	Język wyższego poziomu
Długość kodu źródłowego programu realizującego określoną funkcję	Większa	Mniejsza
Możliwości przenoszenia kodu źródłowego programu między różnymi mikrokontrolerami	Praktycznie niemożliwe	W większości przypadków możliwe po niewielkich modyfikacjach kodu źródłowego
Koszty programu tłumaczącego	Małe	Duże
Optymalność programu (mała pojemność pamięci, duża szybkość działania)	Duża dla krótkich programów, mniejsza dla programów dłuższych	Mała dla programów krótkich, większa dla programów dłuższych

Tablica 5.2. Porównanie własności programowania w języku assemblera i języku wyższego poziomu.

6. Budowa układu pomiarowego JKSPv2.

6.1. Wprowadzenie.

Budowany układ pomiarowy ma być przede wszystkim urządzeniem uniwersalnym i o małych gabarytach. Jego stosowanie nie może wymagać obecności komputera podczas pomiaru. Będzie go można zastosować do pomiarów sygnałów o średniej i niskiej zmienności. Przewidywana rozdzielczość przetworników analogowo-cyfrowych układu to 8 bitów. Układ powinien pozwalać na pomiar kilkukanalowy. Zmierzony sygnał nie może zniknąć wraz z odłączeniem układu od zasilania i będzie go można przesłać do komputera klasy PC celem dalszej analizy.

Omawiany układ pomiarowy JKSPv2 (**J**uczszyk **K**oczubiej **S**ystem **P**omiarowy wersja 2) będzie opierał się o mikrokontroler z rodziny 8051. Są to mikrokontrolery bardzo popularne, dość proste w montażu, nawet dla osób, nie mających doświadczenia w budowaniu układów elektronicznych. Ich niekwestionowaną zaletą, jest też dosyć niska cena i bardzo szeroki wachlarz modeli kontrolerów z dostępnymi różnymi układami peryferyjnymi. Są to układy fabrycznie wyposażane w pamięć ROM, EPROM, czy też sterowniki komunikacji (RS 232, I²C, SPI) [3, 12]. Mikrokontrolery te mogą pracować z oscylatorami o częstotliwości rezonansowej z przedziału od 4 MHz do 24 MHz. Mają również szeroki zakres napięć zasilania, co umożliwia stosowanie zasilania bateryjnego.

Mikrokontrolery te można programować w języku niskiego poziomu – assemblerze (w przeciwieństwie do układów bardziej zaawansowanych: 16- i 32-bitowych) jak i w językach wysokiego poziomu. Z pomocą przychodzą kompilatory C jak i znakomite środowisko programowania Bascom firmy MCS Electronics.

Zastosowanie innego (16-bitowego lub 32-bitowego) kontrolera pociągało by za sobą konieczność stosowania specjalistycznych i drogich narzędzi do programowania kontrolerów oraz znacznie komplikowałoby jego budowę i zwiększało gabaryty.

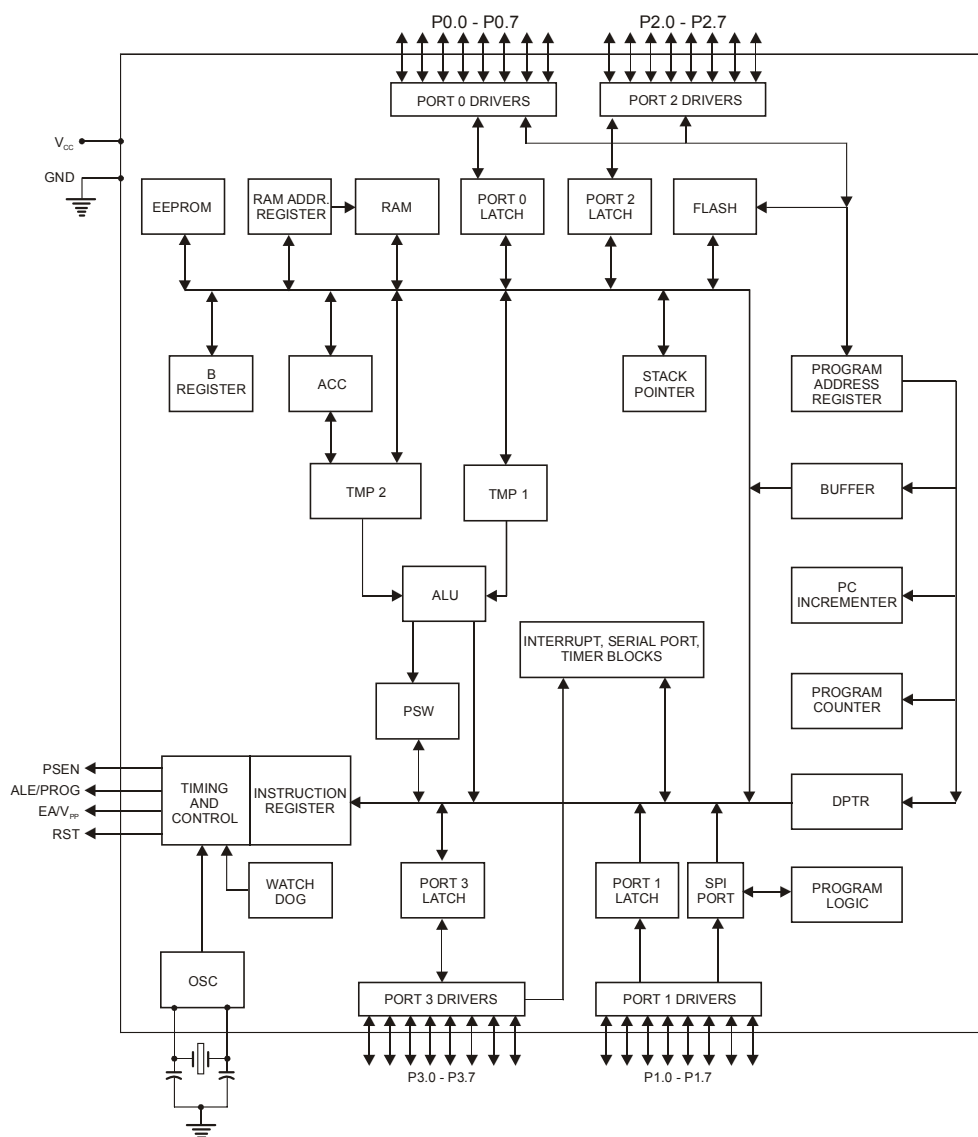
6.2. Wybór podzespołów elektronicznych.

Układu JKSPv2 będzie pracował z kontrolerem AT89S8252 firmy Atmel. Zostanie zastosowany konwerter napięć portu szeregowego (TTL/RS 232) HIN232 firmy Harris. Układ pomiarowy będzie wyposażony w czterokanałowy przetwornik analogowo-cyfrowy PCF8591 firmy Philips. Jest to urządzenie z portem komunikacji I²C. Do przechowywania danych posłużą trzy kości pamięci Atmel EEPROM AT24C512 również ze sterownikami komunikacji I²C. Do komunikacji z użytkownikiem zastosowany zostanie wyświetlacz alfanumeryczny LCD 2x16, z kontrolerem szeregowym HD144870 i klawiatura złożona z pięciu przycisków.

Poza tym, użyte zostaną diody elektroluminescencyjne, oscylator kwarcowy, kondensatory, diody Zenera, diody prostownicze, rezystory, wyłączniki, gniazda i wtyki.

MIKROKONTROLER AT98S8252.

Układ AT98S8252 (rysunek 6.1) jest wysokowydajnym, energooszczędnym 8-bitowym mikrokontrolerem [2]. Posiada jednostkę centralną zbudowaną w standardzie CMOS. Jego zestaw instrukcji (CISC) i układ wyprowadzeń jest zgodny z kontrolerami rodziny 80C51. Posiada 8 kB pamięci FlashROM i 2 kB pamięci EEPROM. Mikrokontroler może być programowany konwencjonalnie, w równoległym programatorze pamięci lub w systemie docelowym za pomocą programatora szeregowego typu SPI. Mikrokontroler ten posiada 256 B pamięci



Rysunek 6.1. Budowa mikrokontrolera AT98S8252 [2].

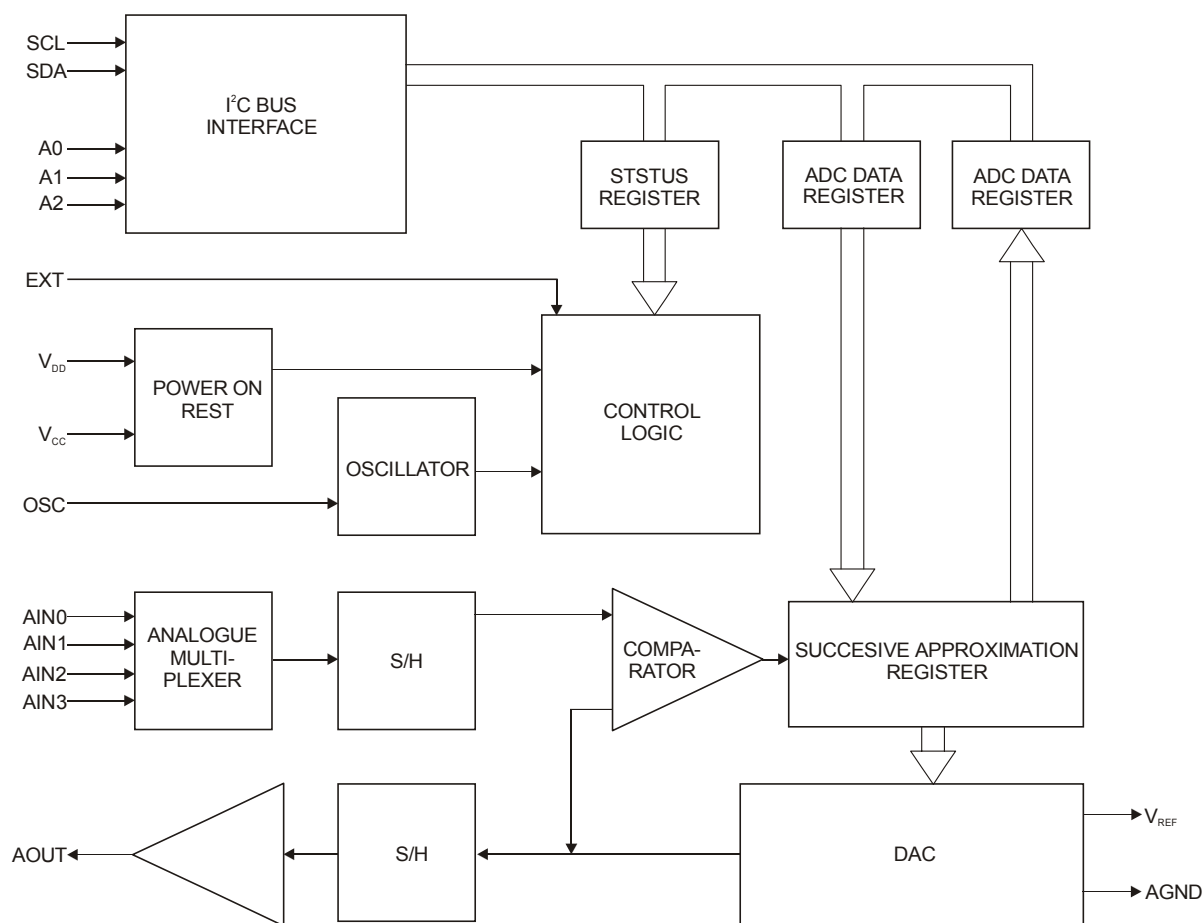
RAM. Komunikować się z układem można za pomocą 32 linii wejścia-wyjścia i sterownika komunikacji szeregowej UART. Mikrokontroler jest ponadto wyposażony w programowalny układ Watch-Dog. Dwa 16-bitowe liczniki/timery mogące pracować w trzech trybach (jako 8-, 13- i 16-bitowe). Można je uruchamiać programowo lub sprzętowo. Układ posiada dziewięć źródeł przerwań o różnym priorytecie, również przerwanie z kontrolera zużycia energii. Może pracować w dwóch trybach energooszczędnych. Według producenta, pamięć FlashROM może być programowana i kasowana 1.000 razy, natomiast pamięć EEPROM 100.000 razy. Procesory są produkowane w obudowach TQFP, PLCC, PDIP, PQFP.

PAMIĘĆ AT24C512.

Układ AT24C512 jest elektrycznie zapisywalną i kasowalną pamięcią typu EEPROM [1]. Jej pojemność wynosi 64 kB (512 stron x 128 B). Zapisywanie do pamięci może odbywać się w dwojaki sposób. Przede wszystkim możemy zapisywać pojedyncze bajty danych. Możemy również zapisywać blokami po 128 bajtów (1 strona). Pamięć może pracować w sposób samoinkrementujący i samodekrementujący się. Oznacza to, że zapisywanie i odczytywanie kolejnych bajtów danych (za wyjątkiem pierwszego), nie wymaga podawania adresów ich przeznaczenia. Bajty te są wpisywane lub odczytywane kolejno po sobie. Pamięć pozwala również na odczyt adresu znajdującego się w rejestrze adresowym. Sterowanie odbywa się za pomocą interfejsu szeregowego w standardzie I²C. Posiada możliwość sprzętowej blokady zapisu. Sposób adresowania układu (w magistrali I²C) pozwala na równoległe stosowanie czterech kości pamięci dla jednego sterownika I²C (jedna kość posiada dwie nóżki do konfiguracji adresu). Według producenta, pamięć może być zapisywana 100.000 razy i nie traci swojej zawartości przez 40 lat. Produkowana jest w obudowach SOIC, PDIP, dBGa.

PRZETWORNIK A/C PCF8591.

Układ PCF8591 (rysunek 6.2) jest 8-bitowym przetwornikiem analogowo-cyfrowym i cyfrowo-analogowym [11]. Jest to układ z szeregową komparacją. Posiada cztery wejścia analogowe (A/C) i jedno wyjście analogowe (C/A). Ma wbudowany układ próbkującą-pamiętający (P/P). Komunikację z nim zapewnia sterownik komunikacji szeregowej w standardzie I²C. Przetwornik jest z klasy układów I²C o trzech nóżkach adresowych, zatem można podłączyć aż osiem takich układów do jednej magistrali. Przetwornik PCF8591 umożliwia wybór kanału próbkowania lub też może pracować w trybie samodzielnej zmiany kanału po każdorazowym pomiarze. Układy produkowane są w obudowach DIP i SO.



Rysunek 6.2. Budowa przetwornika PCF8591 [11].

KONWERTER HIN232.

Układ HIN232 jest konwerterem napięć w komunikacji szeregowej standardu RS 232C [6]. Zasilany napięciem 5 V, jest sterowany napięciem z przedziału 0 V do +5 V (poziom TTL) generuje sygnały o potencjale w granicach -10 V do +10 V. Jest układem energooszczędnym, posiadającym galwaniczną separację wejścia i wyjścia. Spełnia specyfikację RS 232C i V.28.

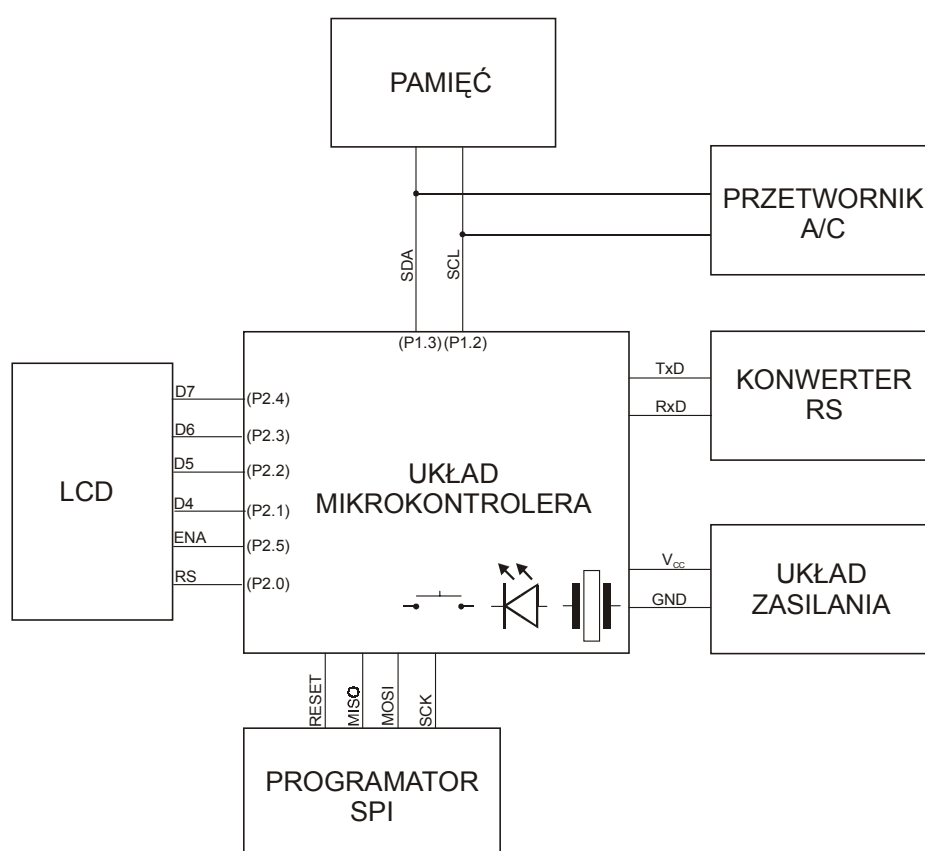
WYŚWIETLACZ LCD.

Zastosowany został podświetlany, alfanumeryczny wyświetlacz ciekłokrystaliczny 2 x 16. Pozwala on na wyświetlenie dwóch linii po szesnaście znaków. Jego sterowanie odbywa się za pomocą sterownika szeregowego HD144870 (Hitachi).

6.3. Montaż i uruchomienie.

W budowie układu pomiarowego JKSPv2 (rysunek 6.3), można wyróżnić następujące bloki:

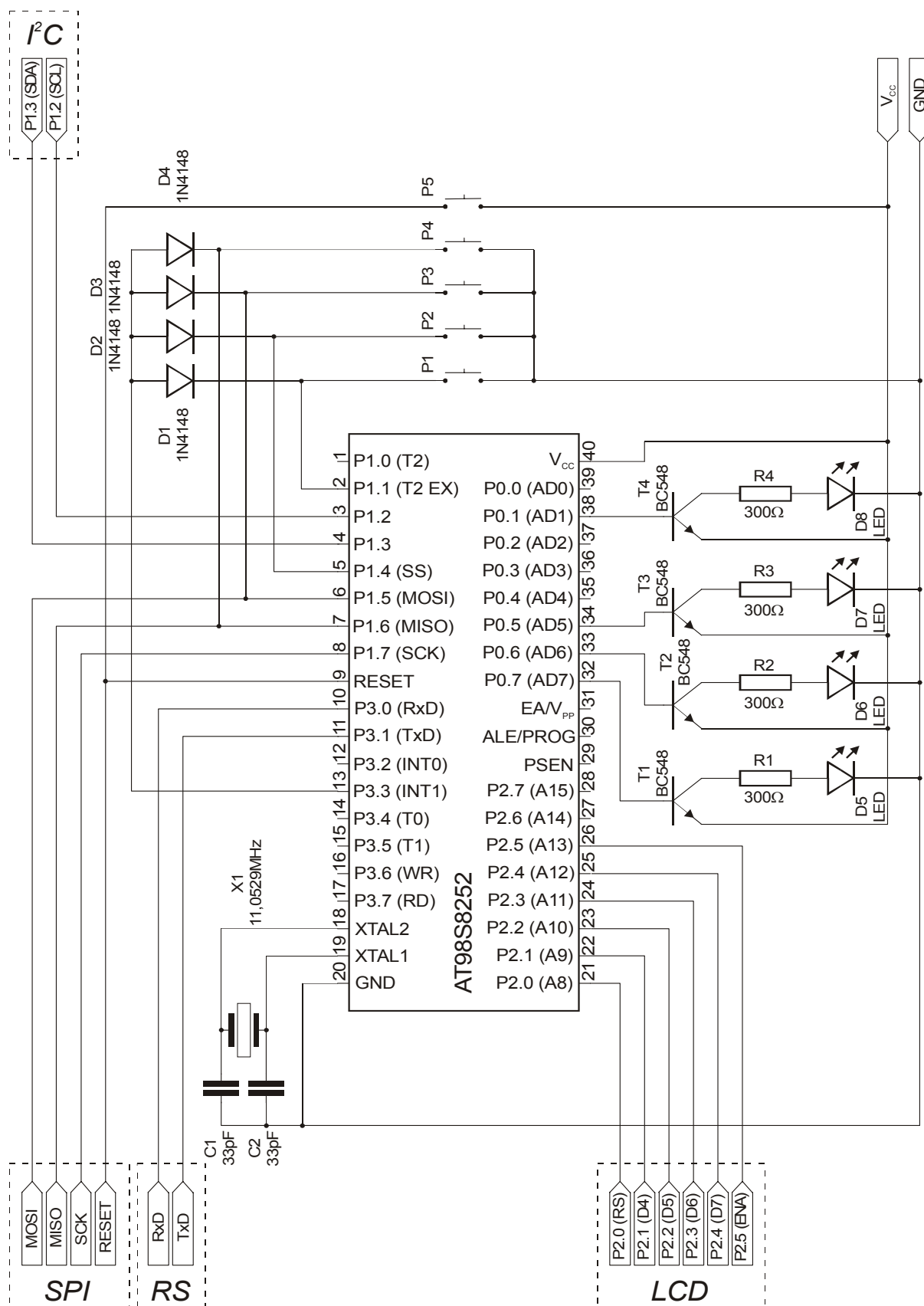
- układ mikrokontrolera (oscylator, klawiatura, sygnalizacja LED),
- pamięć,
- układ przetwornika A/C,
- układ programatora SPI,
- układ wyświetlacza LCD,
- konwerter napięć RS 232,
- układ zasilania.



Rysunek 6.3. Schemat budowy układu pomiarowego.

UKŁAD MIKROKONTROLERA.

Mikrokontroler jest taktowany sygnałem zewnętrznym oscylatora, którego częstotliwość jest stabilizowana kwarem X1 o częstotliwości rezonansowej wynoszącej 11,0592 MHz. Układ mikrokontrolera wyposażony jest w pięć przycisków (4 przyciski klawiatury: P1-P4 i reset mikrokontrolera: P5). Są one podłączone do portów I/O ogólnego zastosowania oraz do kontrolera przerwań (INT1). Podłączenie do kontrolera przerwań przez diody (D1-D4) umożliwia

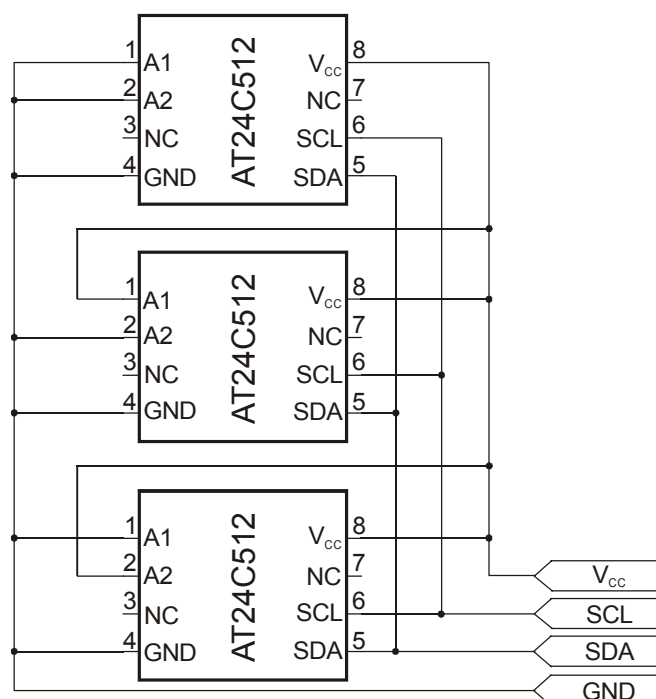


Rysunek 6.4. Schemat układu mikrokontrolera.

uzyskanie tego samego przerwania ze wszystkich przycisków klawiatury. Ponadto układ jest wyposażony w diody elektroluminescencyjne sygnalizujące stan jego pracy. Są one podłączone poprzez rezystory (R1-R4) i sterowane za pomocą tranzystorów T1-T4. Podłączenie takie umożliwia uzyskanie mniejszego napięcia i większego prądu zasilania diod (rysunek 6.4). Wśród wewnętrznych zasobów mikrokontrolera nie ma sterownika komunikacji I²C, jest on symulowany programowo z użyciem dwóch linii I/O ogólnego zastosowania. Obsługa magistrali I²C jest możliwa dzięki implementacji procedur i funkcji do jej obsługi w pakiecie Bascom.

PAMIĘĆ.

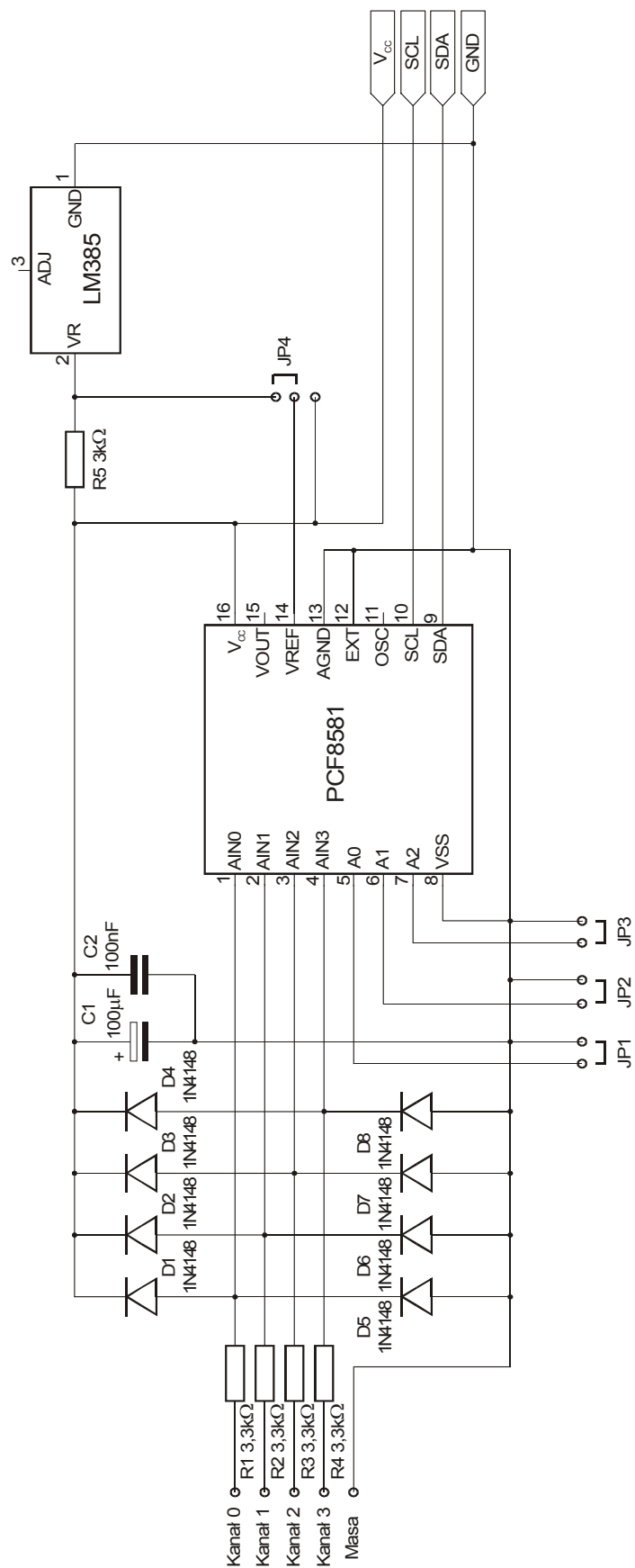
Pamięć jest podłączona poprzez dwuprzewodowy interfejs I²C (rysunek 6.5). Poprzez niego przesyłane i odbierane są dane. Podłączone są trzy kości pamięci i pomiary z każdego kanału będą zapisywane do osobnego układu. Pamięci te posiadają dwie nóżki adresowe (A1 i A2), zatem można podłączyć maksymalnie cztery takie układy do jednej magistrali I²C.



Rysunek 6.5. Sposób podłączenia pamięci.

UKŁAD PRZETWORNIKA A/C.

Przetwornik pracuje z interfejsem I²C (rysunek 6.6). Poprzez niego są przesyłane i odbierane dane. Interfejs służy również do wysyłania rozkazów do przetwornika. Konfigurowanie polega na ustawieniu adresu sprzętowego (JP1 do JP3). Przetwornik posiada trzy nóżki adresowe, zatem można podłączyć aż osiem

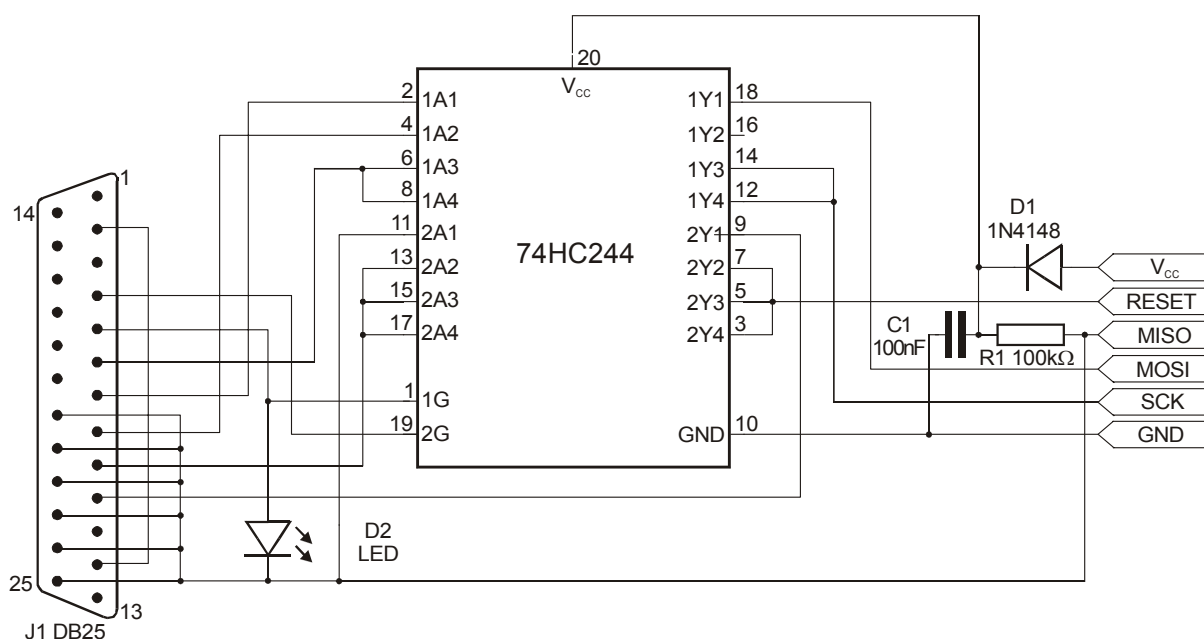


Rysunek 6.6. Sposób podłączenia przetwornika A/C [16].

takich układów do jednej magistrali I²C. Napięciem odniesienia może być napięcie zasilania (5 V) lub napięcie z wysokostabilnego źródła napięcia (LM385) o wartości 1,25 V. Diody D1-D8. Kondensatory C1-C2 i rezystory R1-R4 zabezpieczają przetwornik PCF8051 przed nadmiernym napięciem wejściowym na kanałach pomiarowych. Polecenia obsługi wpisywania, czytania interfejsu I²C, wykorzystywanego w komunikacji z przetwornikiem są wbudowane w pakiet Bascom.

UKŁAD PROGRAMATORA SPI.

Programowanie odbywa się przy pomocy komputera PC, poprzez cztery linie portu Centronics (rysunek 7.7). Jest to możliwe, ponieważ interfejs szeregowy SPI wykorzystuje poziomy TTL. Zastosowanie interfejsu SPI pozwala na programowanie mikrokontrolera w układzie docelowym, bez potrzeby przekładania go układu programatora. Układ 74HC244 separuje obwody układu mikrokontrolera. Jego zastosowanie wymaga dodatkowego źródła zasilania. Wykorzystany został zasilacz układu docelowego. Programator wyposażony jest w diodę elektroluminescencyjną sygnalizującą poprawność podłączenia, uruchomienie pakietu Bascom i gotowość do pracy. Program obsługujący programator SPI jest wbudowany w pakiet Bascom.

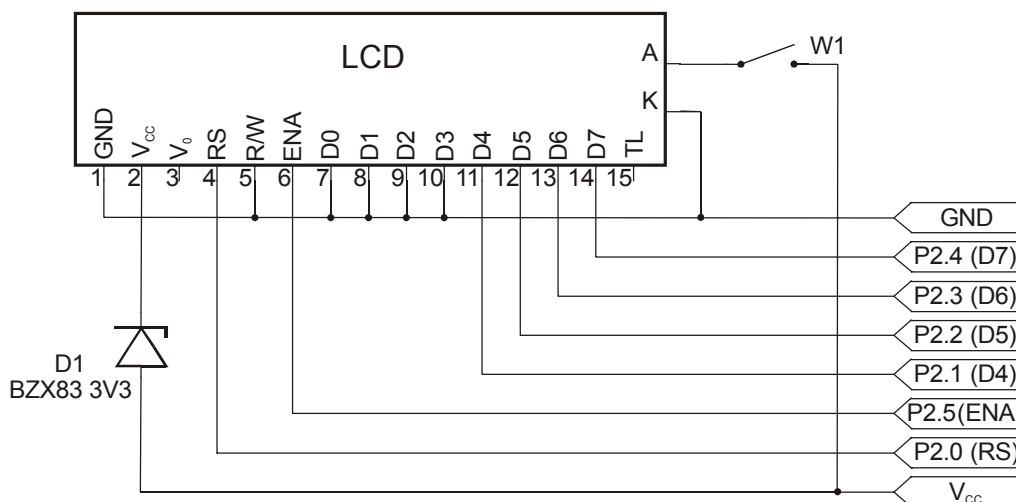


Rysunek 6.7. Sposób podłączenia programatora SPI.

UKŁAD WYŚWIETLACZA LCD.

W układzie znalazł zastosowanie alfanumeryczny wyświetlacz ciekłokrystaliczny. Pozwala on na wyświetlenie dwóch linii po szesnaście znaków. Wyświetlacz jest wyposażony w sterownik szeregowy standardu HDI44870.

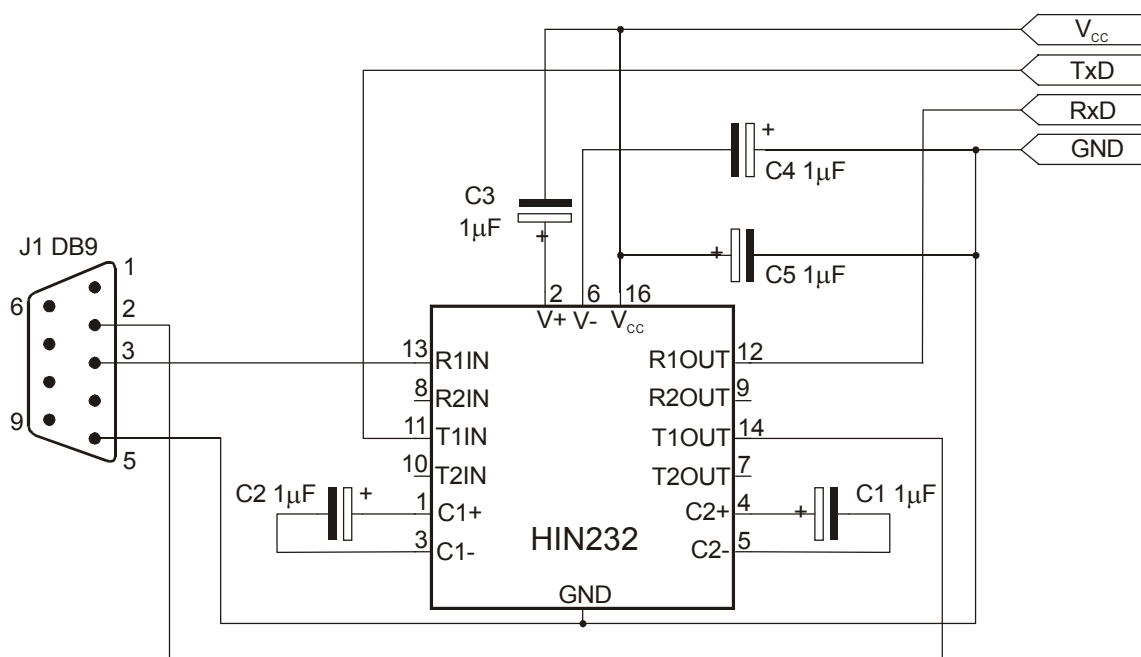
Sterowany jest poprzez porty ogólnego przeznaczenia. Zastosowany wyświetlacz może być podświetlany. Ze względów oszczędnościowych, podświetlenie może być wyłączone. Służy do tego wyłącznik W1. Procedury i funkcje obsługujące wyświetlacz są wbudowane w pakiet Bascom.



Rysunek 6.8. Sposób podłączenia wyświetlacza LCD [15].

KONWERTER NAPIĘĆ.

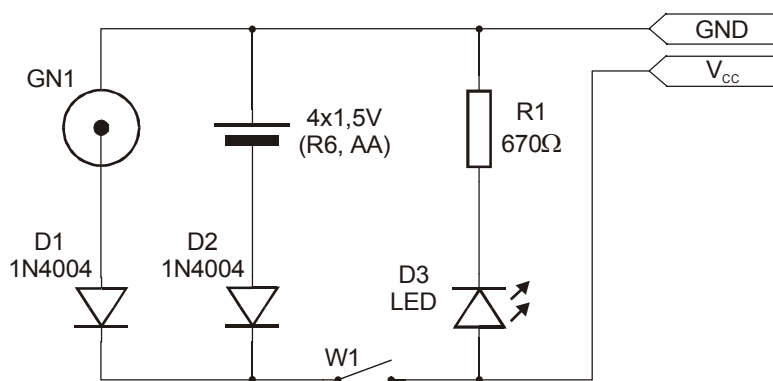
Sygnały elektryczne TxD i RxD mikrokontrolera nie odpowiadają standardowi RS 232, zatem należy użyć układu do konwersji napięć (rysunek 6.9). Do zadania spełnia układ HIN232. Jest to odpowiednik popularnego układu MAX232. Kondensatory C1-C5 służą do podbicia wartości napięcia.



Rysunek 6.9. Sposób podłączenia konwertera napięć [6].

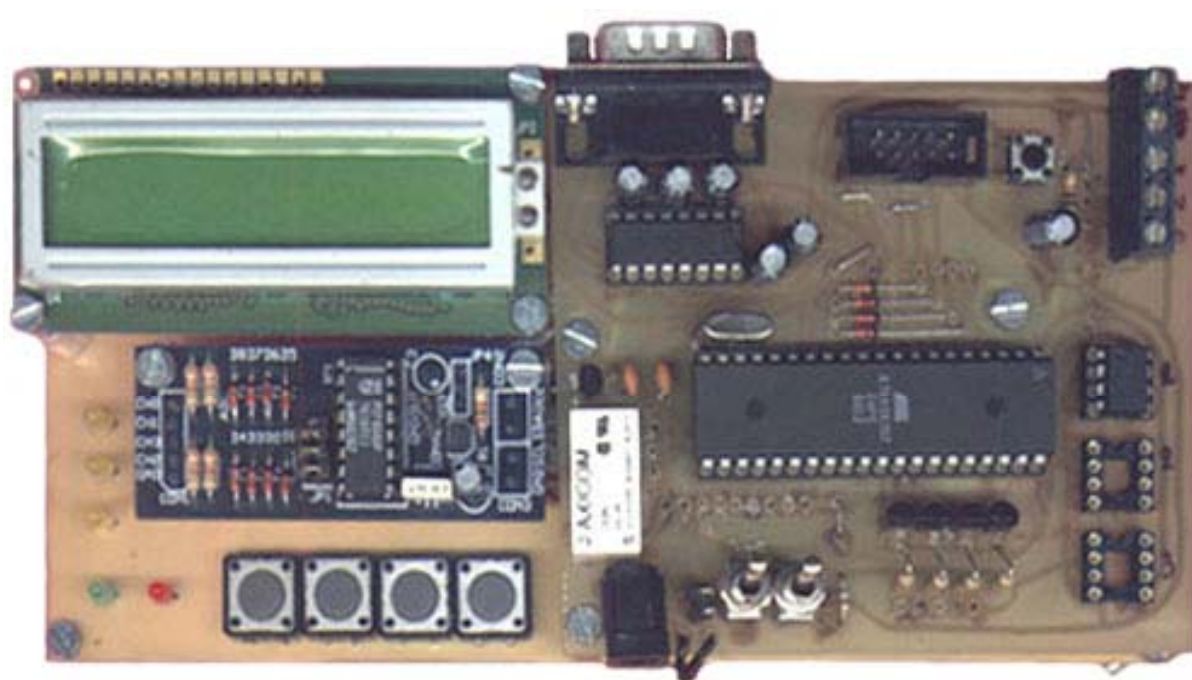
UKŁAD ZASILANIA.

Układ jest zasilany z baterii bądź z zewnętrznego zasilacza (napięciem o wartości 5 V) jak na rysunku 6.10. W przypadku podłączenia obu źródeł zasilania, układ jest zabezpieczony diodami D1-D2. Dioda LED, sygnalizuje włączone zasilanie układu. Układ włącza się za pomocą wyłącznika W1.

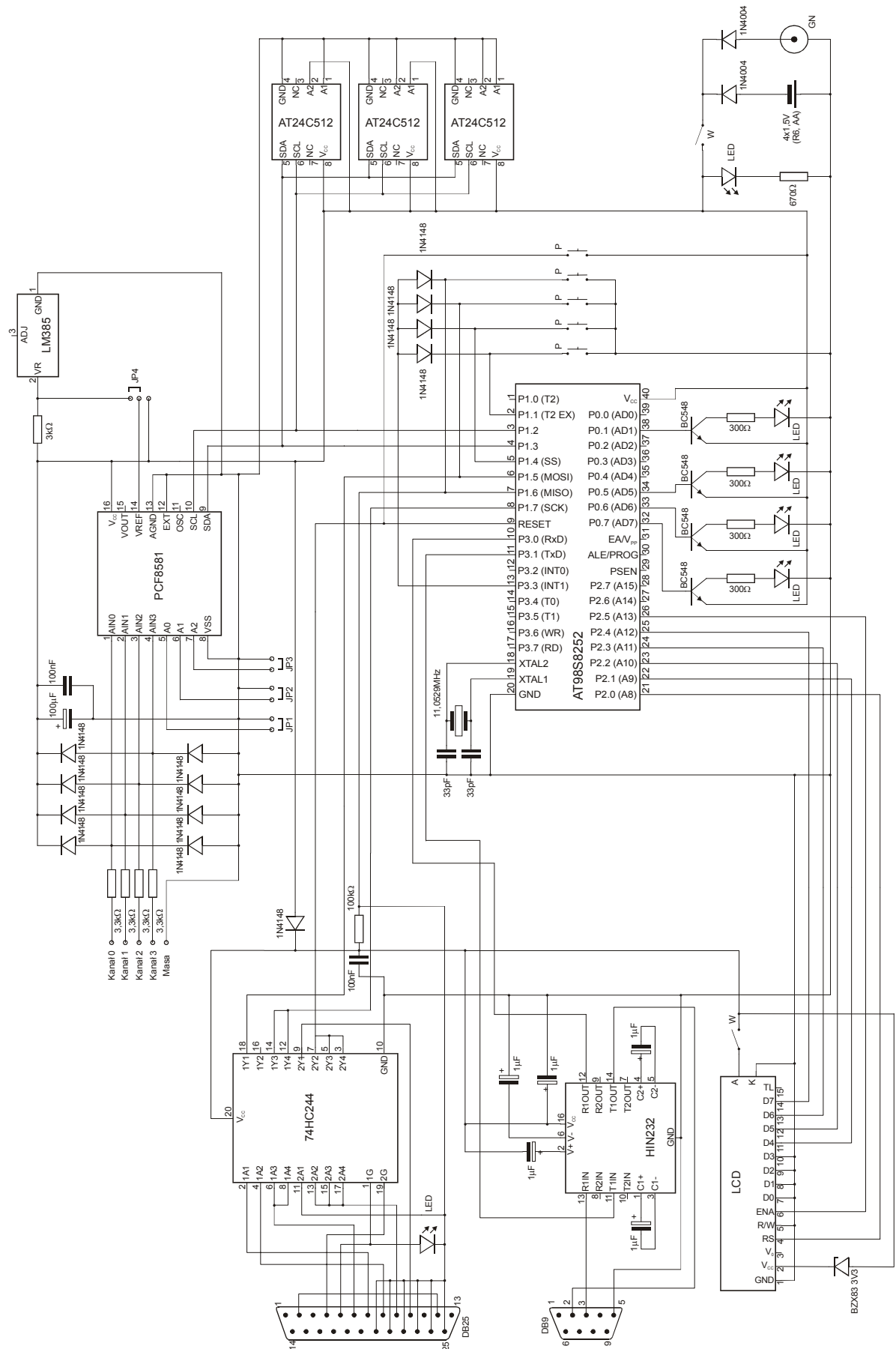


Rysunek 6.10. Sposób zasilania układu pomiarowego.

Na rysunku 6.11. jest przedstawiony jest układ pomiarowy JKSPv2.



Rysunek 6.11. Układ pomiarowy JKSPv2.



Rysunek 6.12. Schemat elektryczny JKSPv2.

7. Oprogramowanie współpracujące z układem JKSPv2.

7.1. Wprowadzenie.

Oprogramowane powinno zapewniać komunikację z **JKSPv2**, mającą na celu transmisję danych. Dane te powinno się dać zapisać do pliku tekstowego, który daje dużą swobodę importu do innych systemów obliczeniowych celem ich dalszej analizy. W miarę możliwości, należy opracować program, pozwalający na wizualizację i wstępną analizę odczytanych danych pomiarowych.

Do tego celu, ze względu na popularność, łatwość obsługi i możliwości graficznej reprezentacji danych najlepiej nadają się systemy operacyjne z rodziny Microsoft Windows. Opracowane programy powinny dać się uruchomić pod wszystkimi dostępnymi systemami Windows. Obsługa komunikacji poprzez port RS 232 wymaga pewnych restrykcji czasowych i niezawodności. Powyższe cele sprawiły, iż oprogramowanie składa się z dwóch części. Podstawowy program do komunikacji **JKSP Communicator** jest napisany z wykorzystaniem wątku pobocznego Windows. Obsługa portu jest dokonana za pomocą interfejsu programowania Win32 API. Zapewnia to niezawodność i uniwersalność. Druga część projektu, program do analizy **JKSP Analyser** powstał, jako niezależne narzędzie dedykowane do analizy danych, pobranych z kontrolera (czyta pliki w takim formacie, w jakim zapisuje program do komunikacji). Pozwala na wybór zakresu analizy, obliczenie dyskretnej transformaty Fouriera (DFT), wygładzanie danych za pomocą średniej ruchomej. Ponadto oblicza podstawowe dane statystyczne. Pozwala na zapis do pliku tekstowego bądź graficznego wyników DFT, jak również zapis wyników obliczeń statystycznych do pliku tekstowego.

7.2. Środowisko programowania i system.

Powstałe programy zostały napisane i skompilowane w środowisku Borland Delphi 5 Enterprise. Jest to rozbudowany pakiet RAD służący do budowy aplikacji dla Windows. Delphi posługuje się zaawansowanym językiem programowania obiektowego Object Pascal 7. Dostarcza szerokiego zestawu narzędzi wspomagających proces tworzenia oprogramowania.

ŚRODOWISKO DELPHI.

Termin RAD (*Rapid Application Development*) oznacza bardzo szybki proces tworzenia aplikacji. Takim terminem określa się narzędzia, które, obok programowania tradycyjnego pozwalają na zastosowanie znacznie efektywnego programowania „wizualnego”. Przykładowo zamiast pisać rozbudowany fragment kodu tworzącego okno wraz z jego składnikami: menu, przyciskami, polami edycyjnymi, itp., można je zbudować z gotowych graficznych elementów, które

wystarczy przeciągnąć i upuścić na projekcie formularza za pomocą myszy. W wyniku tych manipulacji potrzebny kod zostanie wygenerowany automatycznie. Oczywiście, nowy sposób tworzenia aplikacji nie zastąpił całkowicie tradycyjnego programowania. Nadal musimy ręcznie napisać kod funkcji, która ma zostać wykonana w wyniku np. naciśnięcia przycisku okna aplikacji. Niemniej jednak programowanie wizualne stanowi bardzo duże ułatwienie i znakomicie przyspiesza pracę projektantów.

Delphi posługuje się językiem programowania Object Pascal 7. Wywodzi się on od języka Pascal. Pozwala tworzyć przejrzyste i eleganckie programy. Jest znacznie potężniejszy od swojego poprzednika.

Kilka ważniejszych cech języka [10, 17]:

- język w pełni zorientowany obiektowo (enkapsulacja, dziedziczenie, hermetyzacja),
- konstrukcje amorficzne, tablice otwarte, tablice stałych,
- przeciążanie procedur i funkcji,
- obsługa wyjątków,
- inne dodatkowe elementy języka: technologia OLE, ActiveX, obsługa komunikatów Windows.

Kompilator Delphi jest narzędziem bardzo wydajnym. Bardzo krótki jest czas kompilacji i konsolidacji kodu źródłowego. Nie jest to kod pośredni, ale rzeczywisty kod maszynowy, który nie wymaga działania żadnych dodatkowych środowisk czy bibliotek uruchomieniowych. Poza tym dzięki optymalizacji kodu programy napisane w Delphi są bardzo szybkie.

SYSTEM WINDOWS.

Działanie środowiska graficznego, takiego jak Windows jest oparte na koncepcji programowania sterowanego zdarzeniami. W programach dla systemów operacyjnych DOS lub UNIX sterowanie przebiega w sposób sekwencyjny i dobrze ustalony, być może różnymi drogami i z pewnymi przerwami. Natomiast aplikacje Windows zwykle „nie robią niczego”. Jest to jednak mylne stwierdzenie. Program pracujący pod Windows jest w ciągłej gotowości, aby w razie potrzeby, zareagować na różne potencjalne zdarzenia, które mogą mieć miejsce w systemie. Zdarzeniem takim może być, naciśnięcie klawisza klawiatury, ruch wskaźnika myszy, przesunięcie okna bądź naciśnięcie kontrolki. Łatwo sobie wyobrazić, iż w takim programie nie ma jakiegoś zdeteterminowanego przebiegu instrukcji. Natomiast, w zależności od zaistniałego zdarzenia, wykona się właściwy kod obsługi.

Wszystkie programy systemu Windows zbudowane są tak, że na niskim poziomie wywołują funkcje interfejsu programowania Win32 API. Tzn. każdorazowe odwołanie się do sprzętu, obiektu lub zdarzenia w Windows odbywa się za pomocą

Win32 API. Aplikacje napisane w Delphi również korzystają z tych mechanizmów, jednak nie jest to dla programisty bezpośrednio widoczne.

Prześledźmy, co po kolei dzieje się w najprostszym – otwierającym okno, programie napisanym dla Windows¹. W pierwszej kolejności, wypełniany jest rekord zmiennej klasy okna. Uwaga: nie chodzi tu o klasę w znaczeniu znanym dla języków zorientowanych obiektowo, ale raczej w znaczeniu „kategoria” lub „rodzaj”. Żeby otworzyć dowolne okno, należy najpierw zarejestrować (za pomocą Win32 API: `RegisterClass()` [28, 29]) jego klasę. Po przypisaniu odpowiednich wartości (nazwa identyfikująca klasę, wskaźnik procedury okna, uchwyt okna, styl klasy, uchwyt ikony, uchwyt kursora, itp.) do pól rekordu klasy okna, rejestrujemy klasę. Jeśli się to powiedzie, możemy w oparciu o klasę utworzyć nowe okno (funkcją Win32 API: `CreateWindow()` [28, 29]). W oparciu o tę samą klasę możemy utworzyć wiele dosyć różnych obiektów. Jeśli uda nam się utworzyć okno, wówczas funkcja tworząca zwróci uchwyt do nowo utworzonego obiektu. Mając uchwyt, możemy już wyświetlić okno (za pomocą funkcji Win32 API: `ShowWindow()` i `UpdateWindow()` [28, 29]).

Następnie program zawiera pętlę, która ma zasadnicze znaczenie dla jego działania. Jej zadaniem jest określone przetwarzanie komunikatów kierowanych do aplikacji. Kiedy zostanie uruchomiony program w środowisku Windows, system utworzy dla niego tzw. kolejkę komunikatów (mówiąc ściśle kolejka zostanie utworzona dla każdego wątku). Każde zdarzenie (w rozumieniu systemu operacyjnego), np. naciśnięcie przycisku myszy lub klawiatury, ale także utworzenie okna, przesunięcie wskaźnika nad kontrolką, itd., jest przez system przekształcone do postaci komunikatu, będącego rekordem², zawierającego istotne dane o zaistniałym zdarzeniu. Taki rekord może zostać umieszczony w kolejce komunikatów, którego najważniejszym składnikiem jest unikalny identyfikator komunikatu.

Działanie pętli polega na tym, iż pobiera ona z kolejki poszczególne rekordy komunikatów (funkcja `GetMessage()` [28, 29]), następnie pozwala systemowi na przetworzenie niektórych komunikatów (np. związanych z klawiaturą: `TranslateMessage()` [28, 29]), po czym pobrany komunikat wraca do systemu, aby ten mógł go skierować do właściwego okna (funkcja: `DispatchMessage()` [28, 29]).

Podsumowując, najpierw program rejestruje klasę okna. Później, w oparciu o nią, tworzy oraz wyświetla główne okno. Następnie zaś, w pętli komunikatów, zajmuje się ich ciągłym pobieraniem z kolejki i rozsyłaniem do właściwych okien.

¹ Oczywiście chodzi o „zwykłe” aplikacje, wykorzystujące graficzny interfejs użytkownika; jest to najczęściej spotykany, ale nie jedyny rodzaj aplikacji Windows.

² w Delphi typu `TMsg`

Dzieje się tak, dopóki pętla nie natrafi na komunikat `WM_QUIT`. Kiedy to się stanie, pętla zostanie przerwana, a aplikacja zakończy działanie [10].

Jeszcze jednym istotnym zagadnieniem jest tzw. procedura okna (*window procedure*). Procedura okna związana jest z klasą okna. Jej zadaniem jest przetwarzanie komunikatów kierowanych do dowolnego okna utworzonego w oparciu o tę właśnie klasę (jest więc dla nich wspólna). Od jej reakcji na poszczególne komunikaty zależy zachowanie i wygląd okna. Trafiają bowiem komunikaty zawiadamiające o poruszaniu się myszy w obszarze okna, o naciskaniu przycisków myszy i klawiatury, o potrzebie przerysowania obszaru klienta. Procedura okna wywoływana jest przez system z poziomu Win32 API (*callback function*) [10], a jako jej parametry przekazywane są informacje dotyczące komunikatu, który ma zostać obsłużony.

Treść każdej procedury okna jest zwykle podobna. Zawiera instrukcję `case of`, w której bada, jaki komunikat otrzymała i w oparciu o to podejmuje działania. Procedura nie musi jednak obsługiwać wszystkich komunikatów. Wystarczy, iż rozpozna tylko te komunikaty, które są dla niej interesujące, a pozostałe odeśle do standardowej procedury obsługi dostarczonej przez system. Jest to funkcja API o nazwie `DefWindowProc()` [28, 29]. Dzięki niej możemy być pewni, że każdy pominięty komunikat zostanie odpowiednio obsłużony.

Nasz najprostszy przykład, musi zawierać procedurę okna, rozpoznającą jedynie komunikat `WM_DESTROY` [10]. W odpowiedzi na niego powinna wywołać funkcję o nazwie: `PostQuitMessage()` [28, 29]. Ta z kolei umieszcza w kolejce komunikat o nazwie `WM_QUIT` [10]. Jeśli zostanie on pobrany przez funkcję `GetMessage()` w pętli obsługi komunikatów, to nastąpi jej przerwanie. W konsekwencji aplikacja się zakończy.

Pakiet Borland Delphi wszystkie opisane powyżej funkcje generuje automatycznie. Tak więc programista, może się skupić na zaprojektowaniu interfejsu graficznego nowopowstającej aplikacji i oprogramować niezbędne zdarzenia (w sensie Delphi, np. procedury uruchamiające się po wciśnięciu przycisku w oknie aplikacji) będące używane w programie. Zaletą takiego programowania jest elastyczność, przejrzystość i wygoda.

WIELOWĄTKOWOŚĆ.

Systemy operacyjne Windows są systemami wielozadaniowymi z wyłączeniem [23, 24]. Oznacza to, iż w jednym czasie, można na nich uruchamiać wiele programów, pomiędzy którymi będzie dzielony czas procesora. Każdy uruchomiony program, nazywany procesem, posiada swoją własną przestrzeń adresową oraz przynajmniej jeden wątek. Jak z powyższego wynika, to właśnie wątki, a nie procesy są w danej chwili realizowane, bądź też czekają na przydzielenie czasu procesora. W szczególności, wątek główny procesu może

zdefiniować tzw. wątki poboczne, co pozwala na równoczesne wykonywanie kilku czynności w ramach jednego procesu, a więc również w tej samej przestrzeni adresowej. Słowo „równoczesne” znaczy tutaj cykliczne. Chodzi o to, że poszczególne wątki mają cyklicznie przydzielane pewne niewielkie kwanty czasu procesora, w ramach których wykonywany jest ich kod. Z tego powodu odnosimy wrażenie, iż wiele zadań przebiega równolegle. Z kolei wyłączenie, to cecha, dzięki której żaden wątek nie może zagarnąć procesora dla siebie – po upływie przydzielonego mu kwantu czasu, wykonywanie wątku zostanie zawieszone, a procesor zajmie się obsługą kolejnego.

Każdy proces posiada co najmniej jeden wątek, nazywany wątkiem głównym. Dodatkowo, można dla niego tworzyć wątki poboczne, a służy do tego funkcja Win32 API, o nazwie `CreateThread()` [28]. W Delphi mamy do dyspozycji funkcję o nazwie `BeginThread()` [23] z modułu `SysUtils` (jak można było przypuszczać, jest w niej wywoływana wyżej wspomniana funkcja Win32 API). Normalnie, zakończenie wątku następuje po zakończeniu wykonywanego w nim kodu. Zanim to nastąpi można również wymusić, żeby wątek się skończył. Funkcją Win32 API, która do tego służy, jest `ExitThread()` [28]. Oczywiście istnieje odpowiednia funkcja w Delphi obudowująca wspomnianą funkcję Win32 API, a nosi nazwę `EndThread()` [23].

Do tworzenia i manipulowania wątkami można wykorzystać klasę Delphi o nazwie `TThread` [10], implementującą całą ich funkcjonalność. Pewne dodatkowe, oferowane przez nią właściwości umożliwiają korzystanie z obiektów biblioteki VCL przez wątki poboczne. Utworzenie wątku następuje w momencie wywołania konstruktora klasy, którego parametr decyduje o tym, czy wątek ma być automatycznie gotowy do wykonania, czy też zawieszony. Czynności potrzebne do usunięcia wątku i zwolnienia wszystkich zasobów zachodzą na skutek wywołania destruktora klasy.

Aby określić czynności, które ma wykonywać wątek, należy odpowiednio zdefiniować metodę `Execute()` jego klasy (w klasie bazowej `TThread` jest to metoda abstrakcyjna). W metodzie `Execute()` należy sprawdzać dodatkowy warunek, pozwalający na zakończenie wątku w dowolnej chwili. Do osiągnięcia tego stosujemy metodę `Terminate()` i flagę `Terminated` klasy.

We właściwościach klasy `TThread` można znaleźć:

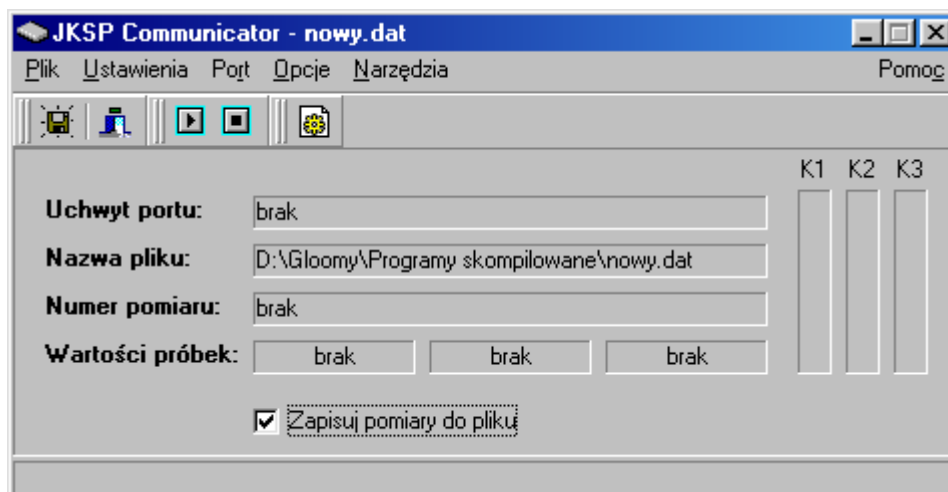
- priorytet względny wątku,
- uchwyt wątku,
- informacje o tym, czy wątek aktualnie jest zawieszony,
- kod powrotu procedury wątku.

Klasa wątku zawiera jeszcze metodę `Synchronize()`. Umożliwia ona korzystanie z komponentów VCL w wątku drugorzędnym. Aby móc ją użyć, należy

najpierw zdefiniować metodę typu `TThreadMethod`, w której trzeba umieścić żądany kod odwołujący się do VCL. Teraz w kodzie wątku można wywoływać metod `Synchronize()`, podając jako jej parametr nazwę (wskaźnik) utworzonej metody operującej na elementach biblioteki VCL. W wyniku tego, metoda operująca na komponentach VCL, zostanie bezpiecznie wykonana w kontekście wątku głównego³.

7.3. Oprogramowanie komunikacji.

Program obsługujący port komunikacyjny RS 232 **JKSP Communicator**, (rysunek 7.1), wpisuje wszystkie odebrane dane do pliku tekstowego. Obsługa portu jest zorganizowana w wątku pobocznym, ze zwiększonym priorytetem. Za obsługę wątku odpowiada klasa `TPortThread`, dziedzicząca po klasie bazowej `TThread`. Główny moduł programu zawiera jeszcze klasę `TRsMain_Form` dziedziczącą po `TForm` i zawierającą metody i obiekty głównego okna programu.



Rysunek 7.1. Główne okno programu JKSP Communicator.

Po uruchomieniu programu, użytkownik może (wybór najważniejszych procedur, pełne źródła ze względu na ograniczoną objętość pracy, dołączone są na płycie CD):

- zmienić nazwę pliku docelowego,
- zmienić konfigurację portu szeregowego,
- zdecydować, czy odbierane dane będą zapisywane do pliku,
- rozpocząć czytanie portu,
- zakończyć czytanie portu,

³ Nastąpi przesłanie komunikatu `CM_EXECPROC` do okna obsługiwanego w kontekście wątku głównego, a obiekt `TThread` zostanie przesłany w parametrze `lParam` komunikatu [10].

- uruchomić program do analizy danych.

Ponadto, program można uruchomić tylko w jednym egzemplarzu (wpis do tablicy `GlobalAtom`, obsługa komunikatu sprawdzającego) i zmienia menu systemowe.

ZMIANA NAZWY PLIKU DOCELOWEGO.

Plik/Zmień nazwę... lub przycisk 

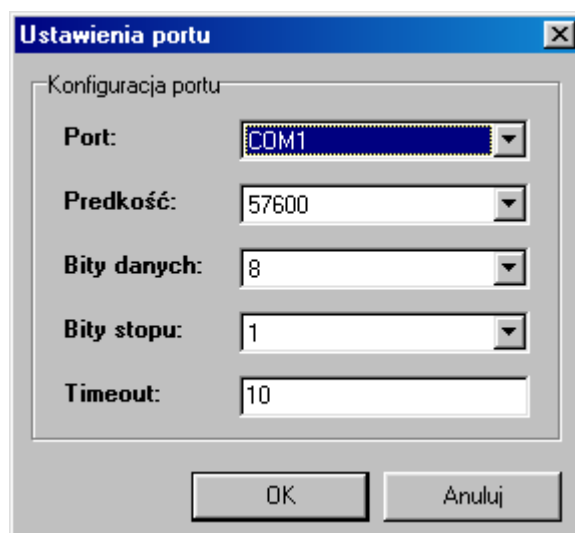
Kod procedury:

```
procedure TRsMain_Form.SaveAs(Sender: TObject);
begin
  if Ds_SaveAs.Execute then
  begin
    FileName:=Ds_SaveAs.FileName;
    RsMain_Form.St_FileName.Caption:=FileName;
    Caption:='Gloomy DyplomRs - '+ExtractFileName(FileName);
  end
  else
  begin
    GetDir(0, FileName);
    FileName:=FileName+'\nowy.dat';
    RsMain_Form.St_FileName.Caption:=FileName;
    Caption:='Gloomy DyplomRs - '+ExtractFileName(FileName);
  end;
end;
```

Działanie procedury polega na uruchomieniu systemowego okna dialogowego wyboru nazwy pliku (typu `TSaveDialog`) i zapamiętaniu nowej nazwy. Ponadto, procedura zmienia nazwę głównego okna.

ZMIANA KONFIGURACJI PORTU SZEREGOWEGO.

Ustawienia/Konfiguracja portu... lub przycisk  (rysunek 7.2)



Rysunek 7.2. Konfiguracja portu RS 232.

Kod procedury:

```
procedure TRsConfig_Form.Ok(Sender: TObject);  
begin  
  case Cb_Port.ItemIndex of  
    0: PortName:=PChar('COM1');  
    1: PortName:=PChar('COM2');  
  end;  
  with PortConfigRecord.dcb do  
    begin  
      case Cb_BaudRate.ItemIndex of  
        0: BaudRate:=CBR_110;  
        1: BaudRate:=CBR_300;  
        2: BaudRate:=CBR_600;  
        3: BaudRate:=CBR_1200;  
        4: BaudRate:=CBR_2400;  
        5: BaudRate:=CBR_4800;  
        6: BaudRate:=CBR_9600;  
        7: BaudRate:=CBR_14400;  
        8: BaudRate:=CBR_19200;  
        9: BaudRate:=CBR_38400;  
        10: BaudRate:=CBR_56000;  
        11: BaudRate:=CBR_57600;  
        12: BaudRate:=CBR_115200;  
        13: BaudRate:=CBR_128000;  
        14: BaudRate:=CBR_256000;  
      end;  
      case Cb_ByteSize.ItemIndex of  
        0: ByteSize:=5;  
        1: ByteSize:=6;  
        2: ByteSize:=7;  
        3: ByteSize:=8;  
      end;  
      case Cb_StopBits.ItemIndex of  
        0: StopBits:=ONESTOPBIT;  
        1: StopBits:=TWOSTOPBITS;  
      end;  
    end;  
    PortTimeoutsRecord.ReadIntervalTimeout:=StrToInt  
      (Ed_Timeout.Text);  
end;
```

Działanie procedury polega na otwarciu okna konfiguracji portu i wpisaniu danych konfiguracji portu do rekordu konfiguracyjnego PortConfigRecord typu TCommConfig oraz PortTimeoutsRecord typu TCommTimeouts. Podczas otwierania portu, jego konfiguracja jest zmieniana według parametrów zawartych w tych rekordach konfiguracyjnych, za pomocą funkcji SetCommConfig() i SetCommTimeouts().

CZYTANIE PORTU.

Port/Odbieraj lub przycisk 

Kod procedury:

```
procedure TPortThread.Execute;
var
  Channel, NumberOfChannels, ByteValue: Byte;
  i: Shortint;
  Step, Delta: Double;
  TxtFile: TextFile;
begin
  ReadEvent:=TEvent.Create(nil, True, False, '');
  EventEvent:=TEvent.Create(nil, True, False, '');
  StopEvent:=TEvent.Create(nil, True, False, '');
  EventsArray[0]:=ReadEvent.Handle;
  EventsArray[1]:=EventEvent.Handle;
  EventsArray[2]:=StopEvent.Handle;
  ReadOverlapped.hEvent:=ReadEvent.Handle;
  EventOverlapped.hEvent:=EventEvent.Handle;
  NumberOfChannels:=3;
  SampleNumber:=0;
  Channel:=0;
  ByteValue:=0;
  Ch1:=0;
  Ch2:=0;
  Ch3:=0;
  Step:=0;
  Delta:=0;
  try
    try
      AssignFile(TxtFile, FileName);
      if SaveToFile then
        Rewrite(TxtFile);
      TransmitCommChar(PortHandle, #0);
      WaitCommEvent(PortHandle, EventMask, @EventOverlapped);
      while True do
        case WaitForMultipleObjects(EventsCount+1,
          @EventsArray, False, INFINITE) of
          WAIT_OBJECT_0:
            begin
              GetOverlappedResult(PortHandle, ReadOverlapped,
                ReadedBytes, False);
              for i:=0 to ReadedBytes-1 do
                begin
                  SampleValue:=Ord(RcvBuffer[i]);
                  Inc(Channel);
                  case Channel of
                    1: Ch1:=SampleValue;
                    2: Ch2:=SampleValue;
                    3: Ch3:=SampleValue;
                  end;
                  if Channel=3 then
                    begin
                      case NumberOfChannels of
                        2: begin
                          Ch1:=Ch2;
                          Ch2:=Ch3;
                          Ch3:=0;

```

```
        end;
    1: begin
        Ch1:=Ch3;
        Ch2:=0;
        Ch3:=0;
        end;
    end;
    Inc(SampleNumber);
    Ch1Vol:=Ch1*Step;
    Ch2Vol:=Ch2*step;
    Ch3Vol:=Ch3*Step;
    Synchronize(WriteVCL);
    case SampleNumber of
        1: begin
            ByteValue:=Ch1;
            if (Ch2=0) and (Ch3=0) then
                Delta:=0.02*ByteValue
            else
                Delta:=(Ch3+Ch2/100);
            end;
        2: begin
            NumberOfChannels:=ByteValue;
            Voltage:=Ch1+Ch2/10+Ch3/100;
            Step:=Voltage/Power(2, 8);
            if SaveToFile then
                begin
                    WriteLn(TxtFile,'dt[s]'+#9+
                        'z[V]'+#9+'lk');
                    WriteLn(TxtFile,
                        FloatToStrF(Delta, ffFixed,
                        3,3)+#9+FloatToStrF
                        (Voltage, ffFixed, 3, 2)+
                        #9+IntToStr(
                        NumberOfChannels));
                    WriteLn(TxtFile);
                    WriteLn(TxtFile, 't[s]'+
                        #9+#9+'K1[V]'+#9+#9+
                        'K2[V]'+#9+#9+'K3[V]');
                end;
            end
        else
            if SaveToFile then
                WriteLn(TxtFile, FloatToStrF(
                    (SampleNumber-3)*Delta,
                    ffExponent, 9,2)+#9+FloatToStrF(
                    Ch1Vol, ffExponent,
                    9, 2)+#9+FloatToStrF(Ch2Vol,
                    ffExponent, 9, 2)+#9+FloatToStrF(
                    Ch3Vol, ffExponent, 9, 2));
            end;
            Channel:=3-NumberOfChannels;
        end;
    end;
    ReadFile(PortHandle, RcvBuffer, BytesToRead,
        ReadBytesNumber, @ReadOverlapped);
end;
```

```
        WAIT_OBJECT_0+1:
        begin
            Application.MessageBox('Zdarzenie błędu: EV_ERR
            lub EV_BREAK.', 'JKSP Communicator',
            MB_ICONERROR or MB_OK);
            WaitCommEvent(PortHandle, EventMask,
            @EventOverlapped);
        end;
        WAIT_OBJECT_0+2:
        begin
            StopEvent.ResetEvent;
            Break;
        end;
    end;
except
    on EInOutError do
        Application.MessageBox('Błąd zapisu do pliku. Wyjątek:
        EInOutError.', 'JKSP Communicator', MB_ICONERROR or
        MB_OK);
    end;
finally
    ReadEvent.Free;
    EventEvent.Free;
    StopEvent.Free;
    if SaveToFile then
        CloseFile(TxtFile);
    end;
end;
```

Jest to metoda `Execute()` klasy `TPortThread`, a więc uruchamiana w wątku pobocznym. W pierwszej kolejności, tworzy ona zdarzenia wskazujące (`TEvent`) na zakończenie działania funkcji. Zdarzenie przechodzi w stan wysoki gdy czynność rozpoczęta wywołaniem funkcji zostaje zakończona – czy to z powodu skompletowania operacji odczytu czy też timeoutu. Inaczej mówiąc, zdarzenie jest powiązane z odpowiednią funkcją za pomocą struktury `EventOverlapped` typu `TOverlapped`, skojarzonej z tą funkcją. Powyżej możemy znaleźć trzy zdarzenia:

- `ReadEvent` – związane z funkcją czytania portu `ReadFile()`,
- `EventEvent` – związane ze stanem portu.
- `StopEvent` – służy do zatrzymania wątku.

Po utworzeniu zdarzeń i wypełnieniu struktury `TOverlapped`, możemy zacząć czytać port, służy do tego funkcja `ReadFile()`, wcześniej utworzony został plik tekstowy, do zapisu danych odbieranych z portu (`AssignFile()` i `RewriteFile()`). Czytanie portu znajduje się w pętli bez warunku wyjścia **with True do**. W konstrukcji pętli użyta jest funkcja oczekująca, `WaitForMultipleObjects()`. Ma ona za zadanie wykrycie przejścia któregoś ze zdarzeń w stan wysoki (*signaled*). Pozwala ona na obsługę więcej niż jednego zdarzenia. Jednym z jej parametrów jest wskaźnik na tablicę zdarzeń. W razie

zajścia jakiegos zdarzenia, pozwala określić je i obsłużyć. W przypadku zgłoszenia zdarzenia zakończenia czytania portu, funkcją `GetOverlappedResult()` sprawdzana jest liczba przeczytanych bajtów. Następnie dane przeczytane są zapisywane do pliku tekstowego. W przypadku zajścia sytuacji uwzględnionej przy konfiguracji maski zdarzeń portu (w tym wypadku `EV_ERR` i `EV_BREAK`) jest to sygnalizowane w postaci monitu systemowego. Zakończenie wykonywania metody wątku, jest wykonywane po zajściu zdarzenia `StopEvent`⁴. Jego celem jest tylko przekazanie do wątku pomocniczego polecenia zakończenia pracy. Powyższe zadania są wykonywane w sekcji chronionej (`try except` i `try finally`). Potrafi ona przechwycić i obsłużyć błędy związane z zapisem do pliku. W razie błędu, przerywana jest pętla, zwalniane są zdarzenia i zamykany jest plik.

Czytanie portu jest poprzedzone jego otwarciem i utworzeniem wątku pobocznego.

Kod procedury:

```
procedure TRsMain_Form.StartPortThread;  
begin  
  if PortHandle=0 then PortHandle:=OpenPort(PortName);  
  if PortHandle<>0 then  
    St_Handle.Caption:=IntToStr(PortHandle);  
  if PortThread=nil then  
    begin  
      PortThread:=TPortThread.Create(True);  
      PortThread.Priority:=tpHigher;  
      PortThread.Resume;  
      Running:=True;  
    end;  
end;
```

Powyższa procedura sprawdza, czy port już nie jest otwarty, jeśli nie to otwiera go za pomocą funkcji `OpenPort()`. Jeżeli nie istnieje też wątek obsługi portu, to tworzy go i uruchamia z podwyższonym priorytetem.

Funkcja otwierająca port ma następującą treść:

```
function TRsMain_Form.OpenPort(PortName: PChar): THandle;  
var  
  PortOpeningHandle: THandle;  
begin  
  Result:=0;  
  PortOpeningHandle:=CreateFile(PortName, GENERIC_READ or  
    GENERIC_WRITE, 0, nil, OPEN_EXISTING,  
    FILE_FLAG_OVERLAPPED, 0);  
  if (PortOpeningHandle=INVALID_HANDLE_VALUE) then  
    begin  
      Application.MessageBox('Błąd otwarcia portu:  
        INVALID_HANDLE_VALUE.', 'Gloomy DyplomRs',  

```

⁴ Nie można skorzystać z metody `Terminate()`, gdyż po wywołaniu funkcji oczekującej wątek znajduje się w stanie zawieszenia i nie ma jak wykryć ewentualnego ustawienia `Terminated`.

```
        MB_ICONERROR or MB_OK);  
    Exit;  
end;  
if (GetFileType(PortOpeningHandle) <> FILE_TYPE_CHAR) then  
begin  
    Application.MessageBox('Błąd typu pliku:  
        FILE_TYPE_CHAR.', 'Gloomy DyplomRs',  
        MB_ICONERROR or MB_OK);  
    Exit;  
end;  
SetCommConfig(PortOpeningHandle, PortConfigRecord,  
    PortConfigSize);  
SetCommTimeouts(PortOpeningHandle, PortTimeoutsRecord);  
SetCommMask(PortOpeningHandle, EV_BREAK or EV_ERR);  
SetupComm(PortOpeningHandle, QueueLength, QueueLength);  
Result := PortOpeningHandle;  
end;
```

Jej parametrem jest nazwa portu, a wynikiem uchwyt do nowo otwartego portu. Do otwarcia portu służy wielozadaniowa funkcja Win32 API `CreateFile()`. W jej wywołaniu, należy podać min. [23, 24]:

- wskaźnik na null-terminated string, opisujący nazwę portu,
- rodzaj dostępu (zapis czy/lub odczyt),
- określenie współdzielenia zasobu,
- rodzaj akcji na istniejącym lub nie istniejącym pliku,
- atrybuty pliku i sposób realizacji dostępu (synchroniczny czy asynchroniczny).

Funkcja ta zwraca uchwyt do otworzonego w ten sposób pliku. Port jest otwierany w trybie asynchronicznym. Oznacza to, że funkcja czytania portu (i zapisu również) od razu po jej wywołaniu (nie czekając na zakończenie) zwraca sterowanie. Do zgłoszenia wykonania swojej operacji używa odrębnych mechanizmów (nie komunikatów).

Po otwarciu portu, następuje jego konfiguracja poprzez struktury `PortConfigRecord` i `PortTimeoutsRecord`.

ZAKOŃCZENIE CZYTANIA PORTU.

Port/Wstrzymaj lub przycisk 

Zakończenie czytania portu polega na uruchomieniu procedury `ClosePort()` o treści:

```
procedure TRsMain_Form.ClosePort;  
begin  
    if PortThread <> nil then  
        begin  
            PortThread.StopEvent.SetEvent;  
            PortThread.Suspend;  
        end;  
    if PortHandle <> 0 then
```

```
begin
    CloseHandle(PortHandle);
    PortHandle:=0;
end;
if PortThread<>nil then
begin
    PortThread.Free;
    PortThread:=nil;
end;
St_Handle.Caption:='brak';
end;
```

Procedura kończy działanie wątku pobocznego poprzez ustawienie w stan wysoki zdarzenia `StopEvent` (następuje przerwanie działania pętli w metodzie `Execute()`). Następnie zawiesza działanie wątku i zamyka port uniwersalną funkcją Win32 API `CloseHandle()`. Potem już tylko są zwalniane zasoby wątku pobocznego metodą `Free`.

URUCHOMIENIE PROGRAMU DO ANALIZY.

Narzędzia/JKSP Analyser

Uruchomienie programu do analizy danych następuje w procedurze:

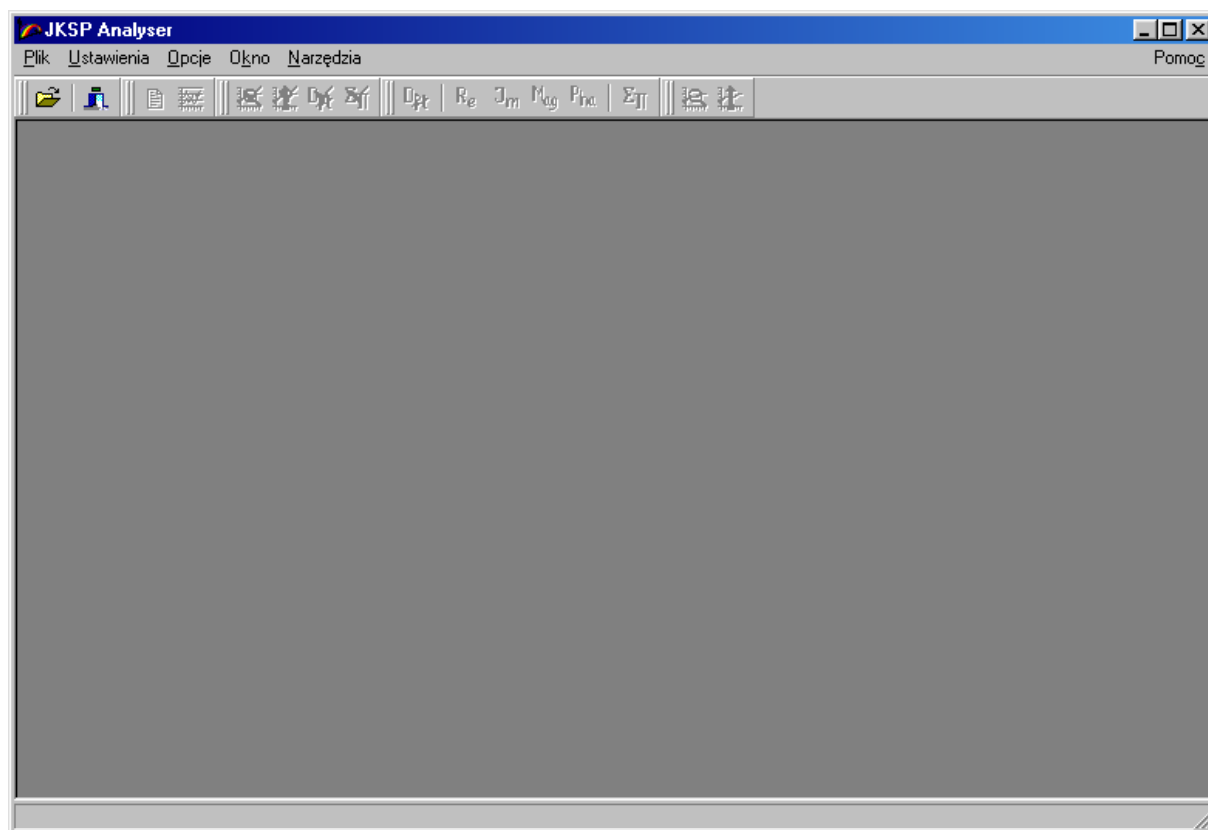
```
procedure TRsMain_Form.RunDyplomAn(Sender: TObject);
var
    ShellExecuteReturn: LongWord;
begin
    ShellExecuteReturn:=ShellExecute(Handle, 'open',
        'JKSP Analyser.exe', nil, nil,
        SW_SHOWNORMAL);
    case ShellExecuteReturn of
        ERROR_FILE_NOT_FOUND:
            Application.MessageBox('Błąd uruchomienia programu:
                ERROR_FILE_NOT_FOUND.', 'Gloomy DyplomRs',
                MB_ICONERROR or MB_OK);
        ERROR_PATH_NOT_FOUND:
            Application.MessageBox('Błąd uruchomienia programu:
                ERROR_PATH_NOT_FOUND.', 'Gloomy DyplomRs',
                MB_ICONERROR or MB_OK);
    end;
end;
```

Zastosowana jest tu funkcja `ShellExecute()` z modułu `ShellApi`. Uruchamia ona program do analizy danych **JKSP Analyser**, będący w tym samym katalogu co program do komunikacji **JKSP Communicator**. W razie nie odnalezienia pliku **JKSP Analyser.exe**, wyświetlany jest monit systemowy o błędzie.

7.4. Oprogramowanie analizy danych.

Program do analizy danych **JKSP Analyser**, (rysunek 7.3) jest aplikacją MDI. Aplikacje MDI pozwalają na pracę z wieloma dokumentami równocześnie. Ich cechą charakterystyczną jest to, że posiadają jedno okno główne, pełniące rolę pojemnika dla okien potomnych [10, 22]. Tych zaś okien jest zazwyczaj kilka. W obrębie okna głównego można sterować ich rozmieszczeniem i rozmiarami, a także dynamicznie je tworzyć bądź usuwać.

Program **JKSP Analyser** jest dedykowany do współpracy z programem do obsługi komunikacji **JKSP Communicator**. Odczytuje pliki tworzone przez tamten i pozwala go uruchomić ze swojego poziomu. Jest to aplikacja jednowątkowa. Pozwala na wybór danych poddawanych późniejszej analizie (kanał pomiarowy i zakres). Można edytować plik danych (plik tekstowy), i przedstawić graficznie, w postaci wykresu dane analizowane. Program potrafi obliczyć dyskretną transformatę Fouriera (DFT) i podstawowe parametry statystyczne z danych analizowanych. W ustawieniach DFT, można zmieniać okienkowanie próbek sygnału i wybrać jednostkę osi odciętych (numery próbek, częstotliwość, częstość, częstotliwość znormalizowana). Wyniki analizy można zapisać do plików tekstowych (wyniki DFT i statystyki) lub w postaci obrazów (wykresy z analizy DFT). Dane analizowane i wyniki można wygładzić średnią ruchomą o zadanych parametrach.



Rysunek 7.3. Główne okno programu JKSP Analyser.

Po uruchomieniu programu, użytkownik może (wybór najważniejszych procedur, pełne źródła ze względu na ograniczoną objętość pracy, dołączone są na płycie CD):

- wczytać plik danych,
- ustalić zakres analizy,
- edytować plik tekstowy danych,
- wykreślić i zapisać (w postaci pliku tekstowego lub obrazu wykresu) dane analizowane,
- kalibrować sygnał analizowany,
- wyznaczyć DFT sygnału (po lub bez kalibracji i wygładzania),
- wykreślić i zapisać (w postaci pliku tekstowego lub obrazu wykresu) wyniki DFT (amplituda DFT, faza DFT, część rzeczywista i urojona DFT),
- wygładzić wyniki i sygnał analizowany średnią ruchomą,
- wyliczyć podstawowe dane statystyczne sygnału,
- uruchomić program do komunikacji.

Ponadto, program można uruchomić tylko w jednym egzemplarzu (wpis do tablicy GlobalAtom, obsługa komunikatu sprawdzającego) i zmienia menu systemowe.

WCZYTYWANIE DANYCH.

Plik/Otwórz... lub przycisk 

Kod procedury:

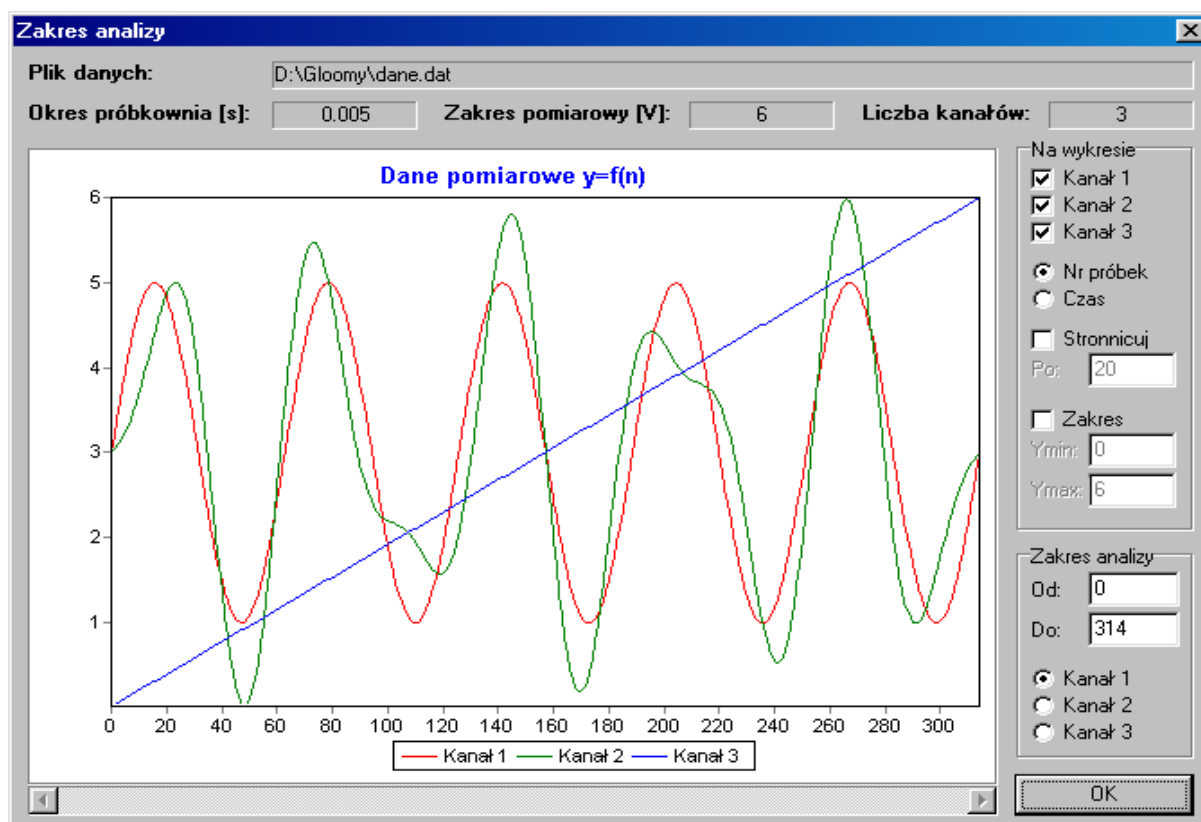
```
procedure TAnMain_Form.OpenData(Sender: TObject);  
var  
    TxtFile: TextFile;  
begin  
    if Do_OpenData.Execute then  
        begin  
            Caption:='JKSP Analyser - '+ExtractFileName(  
                Do_OpenData.Filename);  
            Application.Title:='JKSP Analyser - '+  
                ExtractFileName(Do_OpenData.Filename);  
            n:=0;  
            try  
                try  
                    AssignFile(TxtFile, Do_OpenData.FileName);  
                    Reset(TxtFile);  
                    ReadLn(TxtFile);  
                    Read(TxtFile, Delta);  
                    Read(TxtFile, Voltage);  
                    ReadLn(TxtFile, ChannelsNumber);  
                    ReadLn(TxtFile);  
                    ReadLn(TxtFile);  
                    while not Eof(TxtFile) do  
                        begin  
                            Read(TxtFile, InputTime[n]);
```

```
        Read(TxtFile, InputChannel1[n]);
        Read(TxtFile, InputChannel2[n]);
        ReadLn(TxtFile, InputChannel3[n]);
        Inc(n);
    end;
    MaxSampleNumber:=n;
    Application.CreateForm(TAnSelectData_Form,
        AnSelectData_Form);
    AnSelectData_Form.ShowModal;
    if ViewForm then ViewInputData(Sender);
    if GraphForm then InputDataGraph(Sender);
    if GraphOutForm then AnGraphOut_Form.Close;
    TbBt_ViewInputData.Enabled:=True;
    TbBt_InputDataGraph.Enabled:=True;
    TbBt_Dft.Enabled:=True;
    TbBt_ScaledDataGraph.Enabled:=True;
    TbBt_FilAvrGraph.Enabled:=True;
    Mi_ScalConf.Enabled:=True;
    Mi_FilAvrConf.Enabled:=True;
    Mi_DftConf.Enabled:=True;
    Mi_StatConf.Enabled:=True;
    TbBt_Stat.Enabled:=True;
    TbBt_ReDft.Enabled:=False;
    TbBt_ImDft.Enabled:=False;
    TbBt_MagDft.Enabled:=False;
    TbBt_PhasDft.Enabled:=False;
    TbBt_ScalConf.Enabled:=True;
    TbBt_FilAvrConf.Enabled:=True;
    TbBt_DftConf.Enabled:=True;
    TbBt_StatConf.Enabled:=True;
    Vol2:=Voltage;
    NewValue2:=Voltage;
except
    on EInOutError do
        Application.MessageBox('Błąd odczytu pliku.
            Wyjątek: EInOutError.', 'JKSP Analyser',
            MB_ICONERROR or MB_OK);
    end;
finally
    CloseFile(TxtFile);
end;
end;
end;
```

Procedura wczytuje pierwsze linijki pliku, w których zawarte są: okres próbkowania, liczba kanałów pomiarowych i zakres pomiarowy. Dalej następuje wczytywanie i konwersja danych z tekstu na liczby. Plik ma formę trzech kolumn (trzy kanały pomiarowe) rozdzielonych znakami tabulacji. Próba odczytania błędnego pliku jest sygnalizowana monitem. W przypadku powodzenia operacji, stają się dostępne przyciski analizy, edycji i wizualizacji.

USTALANIE ZAKRESU ANALIZY.

Po odczytaniu pliku, wyświetla się okno pozwalające dokonać wyboru kanału pomiarowego i próbek do analizy (rysunek 7.4).



Rysunek 7.4. Okno wyboru zakresu analizy.

W oknie znajduje się nazwa pliku i jego sygnatura. Na wykresie możemy oglądać poszczególne sygnały z kanałów pomiarowych. Przy dłuższych pomiarach, można włączyć stronicowanie i określić liczbę próbek, znajdujących się naraz na wykresie. Można też ręcznie zamienić zakres osi rzędnych. Wyboru danych do analizy dokonujemy poprzez podanie zakresu (numery próbek) i określeniu kanału pomiarowego.

EDYCJA PLIKU TEKSTOWEGO DANYCH I ZAPISYWANIE ZMIAN.

Przycisk

Program do komunikacji **JKSP Communicator** tworzy pliki tekstowe. Po otwarciu ich w programie do analizy, można je edytować (rysunek 7.5). W menu podręcznym znajduje się opcja zapisu zmian. Jeżeli plik jest zbyt duży, wtedy do edycji jest używany systemowy **WordPad**.

Edycja danych analizowanych				
dt[s]	z[V]	lk		
0.005	6.00	3		
t[s]	K1[V]	K2[V]	K3[V]	
3.1746032e-003	3.0000000e+000	3.0000000e+000	1.9047619e-002	
6.3492063e-003	3.1996668e+000	3.0403486e+000	3.8095238e-002	
9.5238095e-003	3.3973387e+000	3.0827721e+000	5.7142857e-002	
1.2698413e-002	3.5910404e+000	3.1292612e+000	7.6190476e-002	
1.5873016e-002	3.7788367e+000	3.1816412e+000	9.5238095e-002	
1.9047619e-002	3.9588511e+000	3.2414950e+000	1.1428571e-001	
2.2222222e-002	4.1292849e+000	3.3100934e+000	1.3333333e-001	
2.5396825e-002	4.2884354e+000	3.3883349e+000	1.5238095e-001	
2.8571429e-002	4.4347122e+000	3.4766963e+000	1.7142857e-001	
3.1746032e-002	4.5666538e+000	3.5751955e+000	1.9047619e-001	
3.4920635e-002	4.6829420e+000	3.6833684e+000	2.0952381e-001	
3.8095238e-002	4.7824147e+000	3.8002604e+000	2.2857143e-001	
4.1269841e-002	4.8640782e+000	3.9244327e+000	2.4761905e-001	
4.4444444e-002	4.9271164e+000	4.0539834e+000	2.6666667e-001	
4.7619048e-002	4.9708995e+000	4.1865835e+000	2.8571429e-001	
5.0793651e-002	4.9949900e+000	4.3195268e+000	3.0476190e-001	
5.3968254e-002	4.9991472e+000	4.4497918e+000	3.2380952e-001	
5.7142857e-002	4.9833296e+000	4.5741155e+000	3.4285714e-001	
6.0317460e-002	4.9476953e+000	4.6890759e+000	3.6190476e-001	
6.3492063e-002	4.8926002e+000	4.7911822e+000	3.8095238e-001	
6.6666667e-002	4.8185949e+000	4.8769690e+000	4.0000000e-001	
6.9841270e-002	4.7264187e+000	4.9430938e+000	4.1904762e-001	

Rysunek 7.5. Okno edycji danych tekstowych.

USTAWIENIA KALIBRACJI I KALIBRACJA.

Ustawienia/Parametry kalibracji...[Kalibruj] lub przycisk 

Kalibracja polega, na podaniu wartości dwóch punktów charakteryzujących liniową zależność pomiędzy sygnałem zmierzonym w voltach a jego fizyczną wartością amplitudy, w odpowiednich jednostkach (rysunek 7.6). Jeśli chcemy aby sygnał został skalibrowany przed obliczaniem DFT, należy włączyć kalibrację konfiguracji DFT. Przycisk **[OK]** powoduje akceptację parametrów kalibrowania, a przycisk **[Skaluj]** ponadto wyświetla podgląd sygnału po kalibracji.

Ustawienia kalibracji

Parametry kalibracji, punkt 1

Wartość napięcia [V]:

0

Fizyczna wartość amplitudy:

0

Parametry kalibracji, punkt 2

Wartość napięcia [V]:

6

Fizyczna wartość amplitudy:

6

Kalibruj

OK

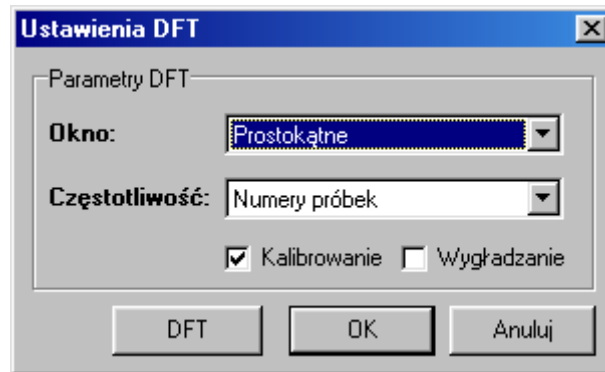
Anuluj

Rysunek 7.6. Ustawienia parametrów skalowania.

USTAWIENIA I OBLICZANIE DFT.

Ustawienia/Parametry DFT...[DFT] lub przycisk 

W ustawieniach dyskretnego przekształcenia Fouriera można zmienić funkcję okienkującą i jednostki osi dociętych (rysunek 7.7). Ponadto można włączyć lub wyłączyć skalowanie i wygładzanie analizowanego sygnału, według parametrów dla nich określonych. Przycisk **[OK]** powoduje akceptację parametrów DFT, natomiast przycisk **[DFT]** ponadto uruchamia obliczenia.



Rysunek 7.7. Ustawienia parametrów DFT.

Obliczenia prowadzone są według zależności [9, 14]:

$$X(\varpi) = \int_{-\infty}^{+\infty} x(t) \cdot e^{-i\varpi t} dt \quad (7.1)$$

Dla sygnału dyskretnego (analogicznie do wzoru 7.1) [19, 20]:

$$\text{Re } X(k) = \sum_{n=0}^{N-1} x_n \cdot \cos\left(\frac{2\pi \cdot k \cdot n}{N}\right) \quad (7.2)$$

$$\text{Im } X(k) = -\sum_{n=0}^{N-1} x_n \cdot \sin\left(\frac{2\pi \cdot k \cdot n}{N}\right) \quad (7.3)$$

Procedura obliczająca DFT (według wzoru 7.2):

```
procedure TAnMain_Form.Dft(Sender: TObject);
var
  i, j, l: Longint;
  WinValue, mValue, bValue: Double;
begin
  Screen.Cursor:=crHourGlass;
  Aborted:=False;
  SetLength(ScaledInputData, n);
  SetLength(FilAvrData, n);
  SetLength(ReDft, n);
  SetLength(ImDft, n);
  SetLength(MagnitudeDft, n);
  SetLength(PhaseDft, n);
  SetLength(FreqDft, n);
  Application.CreateForm(TAnProgress_Form, AnProgress_Form);
```

```

AnProgress_Form.Show;
AnProgress_Form.Update;
WinValue:=0;
mValue:=(NewValue2-NewValue1)/(Vol2-Vol1);
bValue:=((Vol2-Vol1)*NewValue1-(NewValue2-
  NewValue1)*Vol1)/(Vol2-Vol1);
for i:=0 to n-1 do
  begin
    case Window of
      0: WinValue:=1;
      1: begin
          if i<=(n div 2) then WinValue:=i/(n div 2)
          else WinValue:=2-i/(n div 2);
        end;
      2: WinValue:=0.50-0.50*Cos(2*Pi*i/n);
      3: WinValue:=0.54-0.46*Cos(2*Pi*i/n);
      4: WinValue:=0.42-0.50*Cos(2*Pi*i/n)+
          0.08*Cos(4*Pi*i/n);
      5: WinValue:=0.36-0.49*Cos(2*Pi*i/n)+
          0.14*Cos(4*Pi*i/n)-0.01*Cos(6*Pi*i/n);
    end;
    case Scale of
      True: ScaledInputData[i]:=(InputData[i]*
        mValue+bValue)*WinValue;
      False: ScaledInputData[i]:=InputData[i]*WinValue;
    end;
  end;
if FilAvr then
  for i:=0 to n-1 do
    begin
      l:=0;
      FilAvrData[i]:=0;
      for j:=i-SampleBackward to i+SampleForward do
        if (j>=0) and (j<=n) then
          begin
            FilAvrData[i]:=FilAvrData[i]+ScaledInputData[j];
            Inc(l);
          end;
        ScaledInputData[i]:=FilAvrData[i]/l;
      end;
    for i:=0 to n-1 do
      begin
        ReDft[i]:=0;
        ImDft[i]:=0;
        for j:=0 to n-1 do
          begin
            ReDft[i]:=ReDft[i]+
              ScaledInputData[j]*Cos(2*Pi*i*j/n);
            ImDft[i]:=ImDft[i]+
              ScaledInputData[j]*Sin(2*Pi*i*j/n);
          end;
        ImDft[i]:=-ImDft[i];
        MagnitudeDft[i]:=Sqrt(Sqr(ReDft[i])+Sqr(ImDft[i]));
        PhaseDft[i]:=ArcTan2(ImDft[i], ReDft[i]);
        case Freq of
          0: FreqDft[i]:=i;

```

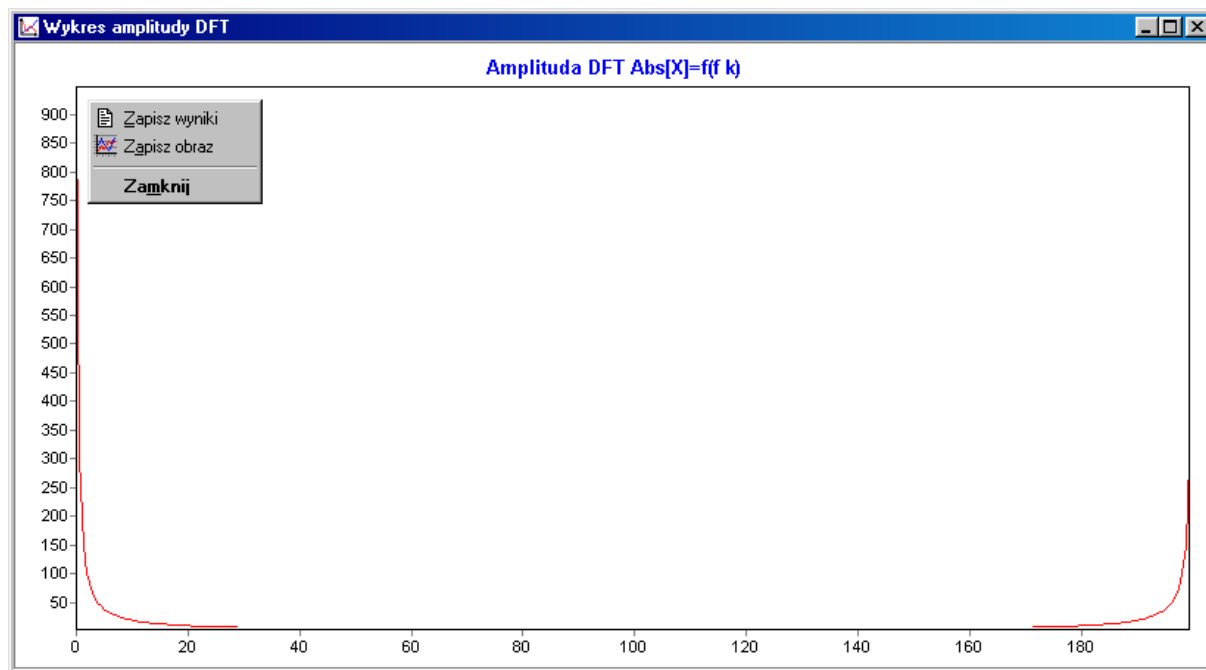
```
1: FreqDft[i]:=i/(Delta*n);
2: FreqDft[i]:=2*Pi*i/n;
3: FreqDft[i]:=i/n;
end;
AnProgress_Form.Ga_Progress.Progress:=Round(100*(i+
1)/n);
Application.ProcessMessages;
if Aborted then Break;
end;
AnProgress_Form.Close;
if GraphOutForm then AnGraphOut_Form.Close;
if Aborted then
begin
    TbBt_ReDft.Enabled:=False;
    TbBt_ImDft.Enabled:=False;
    TbBt_MagDft.Enabled:=False;
    TbBt_PhasDft.Enabled:=False;
    Mi_ScalConf.Enabled:=True;
    Mi_FilAvrConf.Enabled:=True;
    Mi_DftConf.Enabled:=True;
end
else
begin
    TbBt_ReDft.Enabled:=True;
    TbBt_ImDft.Enabled:=True;
    TbBt_MagDft.Enabled:=True;
    TbBt_PhasDft.Enabled:=True;
    if GraphForm then InputDataGraph(Sender);
end;
Screen.Cursor:=crDefault;
end;
```

Procedura ustala wielkość dynamicznych tablic (`SetLength()`), potrzebnych do zapamiętania wyników. Następnie w pętli `for to` skaluje i okienkuje dane poddawane analizie. Wybór okna następuje w instrukcji `case of`. Następnie dany sygnał jest wygładzany (pod warunkiem wybrania tej opcji). W kolejnym kroku jest obliczane przekształcenie Fouriera (części: rzeczywista i urojona a na ich podstawie amplituda i faza). W przypadku nie przerywania obliczeń, stają się aktywne przyciski wyświetlania wyników i analizy statystycznej. Wewnątrz pętli obliczeń znajduje się polecenie `Application.ProcessMessages`, pozwala ono na przerywanie obliczeń (obsługa kolejki komunikatów, podczas wykonywania obliczeń w pętli; umożliwia reakcję na przyciśnięcie klawisza Esc).

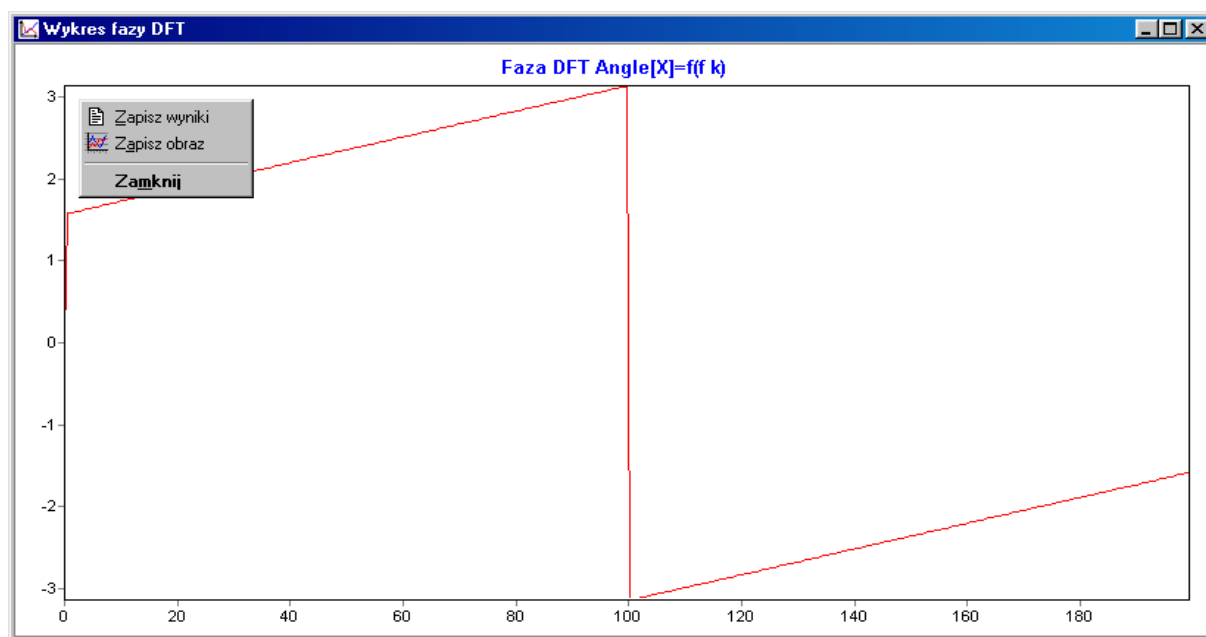
WYKREŚLANIE I ZAPISYWANIE WYNIKÓW.

Przyciski:  - część rzeczywista DFT,  - część urojona DFT,
 - amplituda DFT,  - faza DFT.

Po wciśnięciu któregoś z powyższych przycisków, zostanie wyświetlony wykres (rysunki 7.8 i 7.9). Po przyciśnięciu prawego klawisz myszki, będzie można wybrać format zapisu wyników. Do wyboru jest: obraz (bitmapa) wykresu lub plik tekstowy ze współzrędnymi punktów (odcięte i rzędne), rozdzielone znakami tabulacji. Proponowana nazwa pliku do zapisu wyników, obejmuje nazwę pliku z pomiarami i sugeruje jaka część wyników analizy jest zapamiętana w pliku.



Rysunek 7.8. Okno wykresu wyniku obliczeń (amplituda DFT).

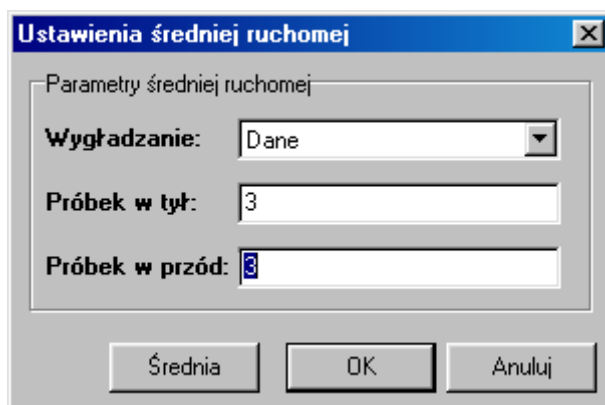


Rysunek 7.9. Okno wykresu wyniku obliczeń (faza DFT).

PARAMETRY WYGŁADZANIA I WYGŁADZANIE.

Ustawienia/Parametry średniej ruchomej...[Średnia] lub przycisk 

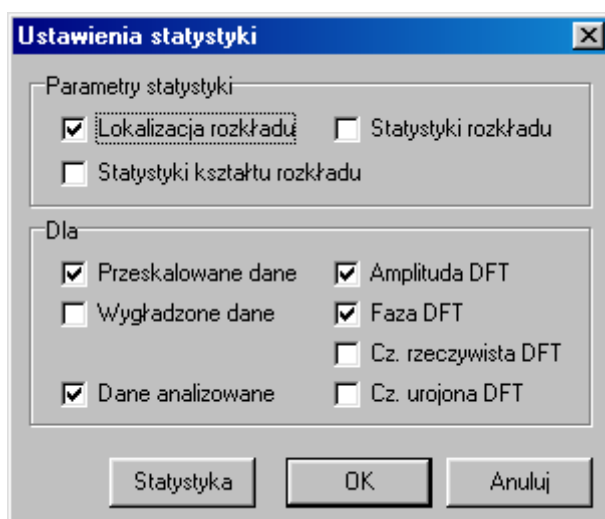
Wygładzanie polega na wyliczeniu średniej ruchomej z danego sygnału. Możliwe jest określenie liczby próbek, z których będzie liczona średnia (rysunek 7.10). Przed obliczeniem DFT, możliwe jest tylko wygładzenie sygnału analizowanego. Po analizie Fouriera, wygładzać można już części: rzeczywistą i urojoną oraz amplitudę i fazę DFT. Przycisk **[OK]** powoduje akceptację parametrów wygładzania, a przycisk **[Średnia]** ponadto wyświetla podgląd wygładzonego sygnału. Po wygładzeniu można nagrać wykres w postaci bitmapy lub pliku tekstowego (menu podręczne). Jeżeli sygnał analizowany ma być wygładzony przed obliczeniem DFT, opcję tę należy jeszcze włączyć w konfiguracji DFT.



Rysunek 7.10. Ustawienia parametrów średniej ruchomej.

PARAMETRY I OBLICZANIE STATYSTYKI.

Ustawienia/Parametry statystyki...[Statystyka] lub przycisk 



Rysunek 7.11. Ustawienia parametrów statystyki.

Ustawienia statystyki polegają na wyborze z jakich źródeł będą prowadzone obliczenia i jakie parametry statystyczne będą obliczone (rysunek 7.11).

Parametry statystyczne podzielone są na kategorie [4]:

- statystyki lokalizacji rozkładu,
- statystyki rozproszenia rozkładu,
- statystyki kształtu rozkładu.

Statystyki lokalizacji rozkładu zawierają [4]:

- minimalną wartość $x_{\min}$,
- maksymalną wartość $x_{\max}$,
- średnią arytmetyczną, określaną z zależności:

$$\bar{x} = \frac{1}{N} \sum_{i=0}^{N-1} x_i, \quad (7.4)$$

- rozstęp, określany z zależności:

$$R = x_{\max} - x_{\min}, \quad (7.5)$$

- medianę (wartość środkową) m_e .

Statystyki rozproszenia rozkładu zawierają [4]:

- odchylenie przeciętne od wartości średniej, określane z zależności:

$$d_1 = \frac{1}{N} \sum_{i=0}^{N-1} |x_i - \bar{x}|, \quad (7.6)$$

- odchylenie przeciętne od wartości mediany, określane z zależności:

$$d_2 = \frac{1}{N} \sum_{i=0}^{N-1} |x_i - m_e|, \quad (7.7)$$

- odchylenie standardowe, określane z zależności:

$$s = \sqrt{s^2}, \quad (7.8)$$

- wariancję, określaną z zależności:

$$s^2 = \frac{1}{N} \sum_{i=0}^{N-1} (x_i - \bar{x})^2, \quad (7.9)$$

Statystyki kształtu rozkładu zawierają [40]:

- współczynnik koncentracji, określany z zależności:

$$K = \frac{M_4}{s^4}, \quad (7.10)$$

- współczynnik asymetrii, określany z zależności:

$$g = \frac{M_3}{s^3}, \quad (7.11)$$

- moment zwykły rzędu 2, określany z zależności:

$$m_2 = \frac{1}{N} \sum_{i=0}^{N-1} x_i^2, \quad (7.12)$$

- moment zwykły rzędu 3, określany z zależności:

$$m_3 = \frac{1}{N} \sum_{i=0}^{N-1} x_i^3, \quad (7.13)$$

- moment zwykły rzędu 4, określany z zależności:

$$m_4 = \frac{1}{N} \sum_{i=0}^{N-1} x_i^4, \quad (7.14)$$

- moment centralny rzędu 2, określany z zależności:

$$M_2 = \frac{1}{N} \sum_{i=0}^{N-1} (x_i - \bar{x})^2, \quad (7.15)$$

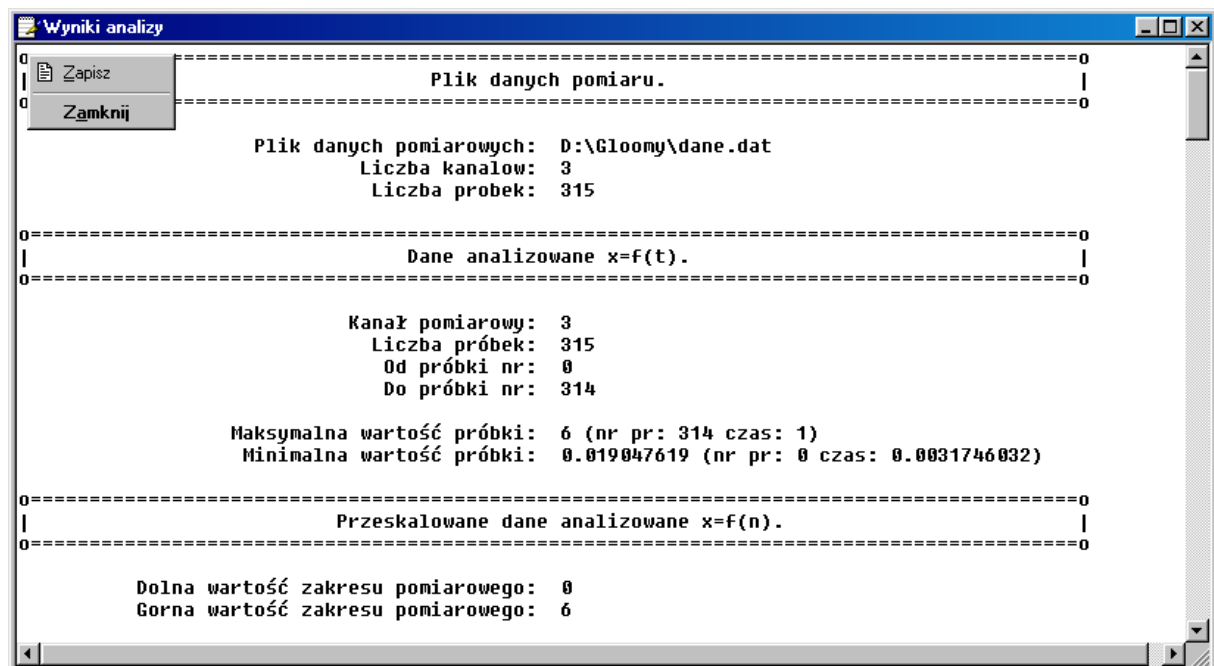
- moment centralny rzędu 3, określany z zależności:

$$M_3 = \frac{1}{N} \sum_{i=0}^{N-1} (x_i - \bar{x})^3, \quad (7.16)$$

- moment centralny rzędu 4, określany z zależności:

$$M_4 = \frac{1}{N} \sum_{i=0}^{N-1} (x_i - \bar{x})^4. \quad (7.17)$$

Przed obliczeniem DFT, niedostępne będą źródła z nią związane (faza, amplituda, część rzeczywista i urojona przekształcenia). Przycisk **[OK]** powoduje akceptację parametrów statystyki, a przycisk **[Statystyka]** ponadto wyświetla



Rysunek 7.12. Okno z wynikami obliczeń statystycznych.

obliczenia. Można je zapisać do pliku tekstowego. Na rysunku 7.12. znajdują się przykładowe okno z wynikami obliczeń.

URUCHAMIANIE PROGRAMU DO KOMUNIKACJI.

Narzędzia/JKSP Communicator

Uruchomienie programu komunikacji następuje w procedurze:

```
procedure TAnMain_Form.RunDyplomRs(Sender: TObject);  
var  
    ShellExecuteReturn: LongWord;  
begin  
    ShellExecuteReturn:=ShellExecute(Handle, 'open',  
        'JKSP Communicator.exe', nil, nil, SW_SHOWNORMAL);  
    case ShellExecuteReturn of  
        ERROR_FILE_NOT_FOUND:  
            Application.MessageBox('Błąd uruchomienia programu:  
                ERROR_FILE_NOT_FOUND.', 'Gloomy DyplomAn',  
                MB_ICONERROR or MB_OK);  
        ERROR_PATH_NOT_FOUND:  
            Application.MessageBox('Błąd uruchomienia programu:  
                ERROR_PATH_NOT_FOUND.', 'Gloomy DyplomAn',  
                MB_ICONERROR or MB_OK);  
    end;  
end;
```

Zastosowana jest tu funkcja `ShellExecute()` z modułu `ShellApi`. Uruchamia ona program do obsługi komunikacji **JKSP Communicator**, będący w tym samym katalogu co program do analizy **JKSP Analyser**. W razie nie odnalezienia pliku **JKSP Communicator.exe**, wyświetlany jest monit systemowy o błędzie.

8. Zastosowanie układu i oprogramowania.

8.1. Wprowadzenie.

Przy pomocy **JKSPv2** zostały przeprowadzone pomiary:

- sygnałów okresowo zmiennych pochodzących z generatora laboratoryjnego, z pomocą komputera PC,
- całodobowy monitoring temperatury otoczenia z zapisem do pamięci.

Do generacji sygnałów testowych posłużył generator G 432 Function Generator. Użyliśmy przebiegów:

- piłokształtnego o częstotliwościach: 1 Hz i 10 Hz,
- prostokątnego o częstotliwościach: 1 Hz i 10 Hz,
- sinusoidalnego o częstotliwościach: 1 Hz, 10 Hz, 100 Hz i 1000 Hz.

Sygnały te były próbkowane z szybkością 500 próbek na sekundę i wysyłane bezpośrednio do komputera PC.

Temperatura została zmierzona z wykorzystaniem czujnika LM 35 firmy National Semiconductor. Pomiary temperatury były wykonywane przez 24 godziny z zapisem do pamięci co 60 s.

8.2. Wykonanie pomiarów.

POMIAR Z WYKORZYSTANIEM KOMPUTERA.

Do zbierania i zapamiętywania pomiarów posłużył komputer klasy PC z systemem operacyjnym Microsoft Windows 98 SE. Aby dokonać pomiaru należało podłączyć układ **JKSPv2** do komputera przez port RS 232C oraz podłączyć generator do pierwszego kanału pomiarowego układu. Następnie po włączeniu **JKSPv2** i wybraniu jednego kanału pomiarowego (**Liczba kanałów**), z jego menu należy wybrać **Pomiar (dane na RS)** i zaakceptować wybór. Na wyświetlaczu LCD układu pojawia się napis **Czekam...** co oznacza gotowość do pracy. Po uruchomieniu programu **JKSP Communicator** i skonfigurowaniu protokołu transmisji (**Ustawienia/Konfiguracja portu...** lub ) pomiar rozpoczyna się po wybraniu opcji **Port/Odbieraj** lub wciśnięciu przycisku . Jeśli zaznaczona jest opcja **Zapisuj pomiary do pliku**, transmitowane dane będą zapamiętane w pliku o wybranej nazwie. Układ pomiarowy sygnalizuje pracę napisem: **Pomiar + transmisja...** Pomiar kończy za pomocą opcji **Port/Wstrzymaj** lub przyciskiem , co powoduje zakończenie transmisji i reset mikrokontrolera.

POMIAR Z ZAPISEM DO PAMIĘCI.

W takim wypadku do zapamiętywania pomiarów służy wewnętrzna pamięć EEPROM **JKSPv2**. Aby wykonać taki pomiar, należy najpierw ustawić czas próbkowania (opcja **Czas próbkowania**) i liczbę kanałów pomiarowych (opcja **Liczba kanałów**). Po podłączeniu układu pomiar rozpoczyna się za pomocą opcji **Pomiar z zapisem**. Praca układu jest sygnalizowana pulsowaniem diody LED (koloru zielonego) i napisem **Pomiar** na wyświetlaczu LCD. Praca układu kończy się automatycznie po wypełnieniu pamięci (komunikat **Pełna pamięć**) lub po wciśnięciu przycisku **C** mikrokontrolera. Po wykonaniu pomiaru należy przetransmitować zebrane dane do komputera PC za pomocą programu **JKSP Communicator** (opcja **Port/Odbieraj** lub przycisk ).

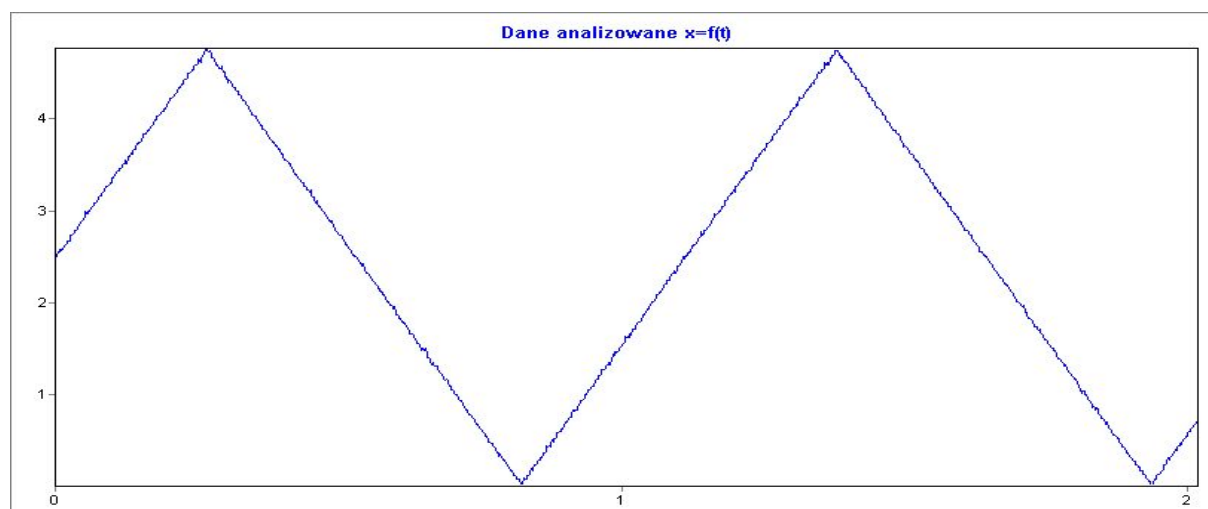
8.3. Analiza.

Po uruchomieniu programu **JKSP Analyser**, należy wczytać plik danych za pomocą opcji **Plik/Otwórz...** lub przycisku . W okienku zakresu analizy wybieramy kanał pomiarowy i zakres próbek. Po wciśnięciu przycisku  możemy obejrzeć analizowany sygnał. Po ustawieniach parametrów kalibracji (**Ustawienia/Parametry kalibracji...** lub ), wygładzania (**Ustawienia/Parametry Średniej ruchomej...** lub ), statystyki (**Ustawienia/Parametry Statystyki...** lub ) i przekształcenia DFT (**Ustawienia/Parametry DFT...** lub ), możemy poddać analizie sygnał (przyciski:  i ).

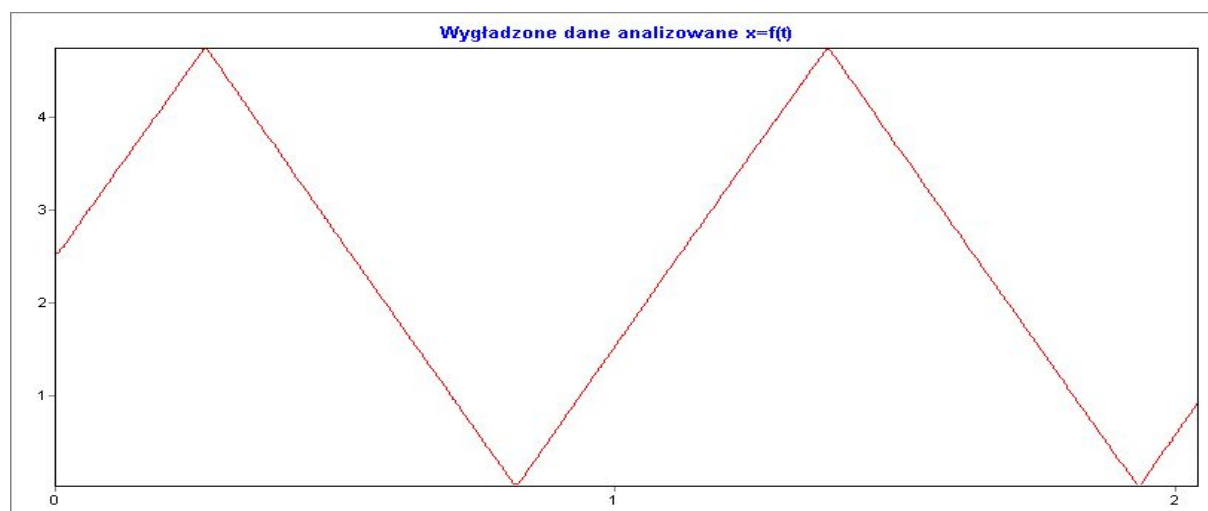
Na rysunkach: 8.1 do 8.16 zamieszczone są przykładowe wyniki analizy uzyskane programem **JKSP Analyser** (wszystkie wykresy ze względu na ograniczoną objętość pracy, dołączone są na płycie CD).

Rysunek 8.1 przedstawia prawidłowo próbkowany przebieg piłokształtny o częstotliwości 1 Hz. Na rysunku 8.2 znajduje się wygładzony przebieg piłokształtny z rysunku 8.1. Widoczne zmniejszenie zaszumienia sygnału.

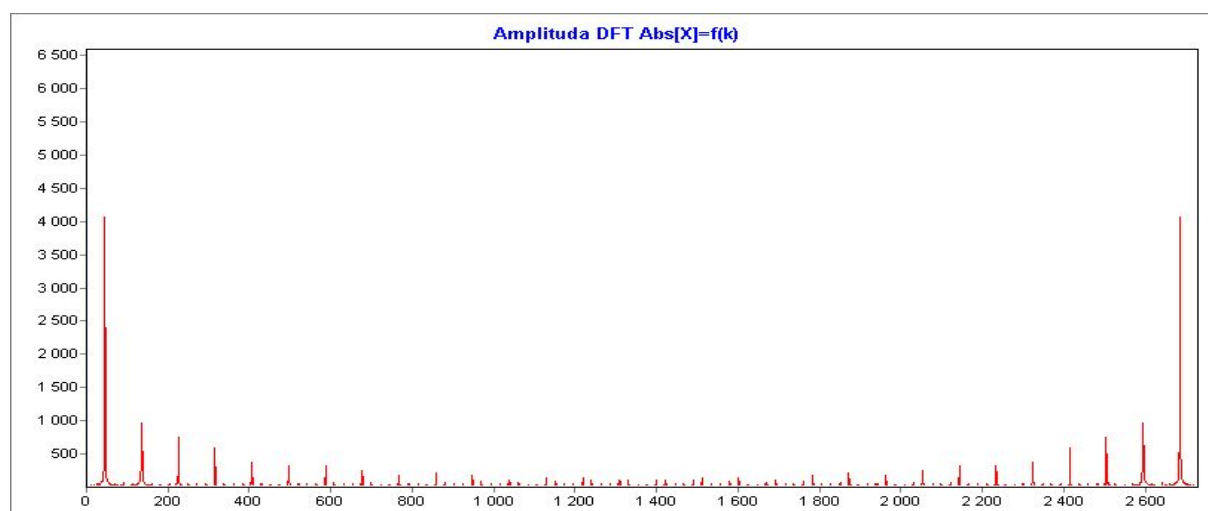
Na rysunku 8.3 znajduje się gęstość rozkładu w dziedzinie częstotliwości przebiegu prostokątnego o częstotliwości 10 Hz. Charakterystyczne piki oznaczają kolejne składowe harmoniczne przebiegu. Rysunek 8.4 pokazuje gęstość rozkładu tego samego przebiegu prostokątnego ale z zastosowaniem okienkowania według funkcji Harrisa. Widoczne jest zmniejszenie tzw. przecieków widma DFT (prążki składowych harmonicznnych są bardziej ostre i smukłe).



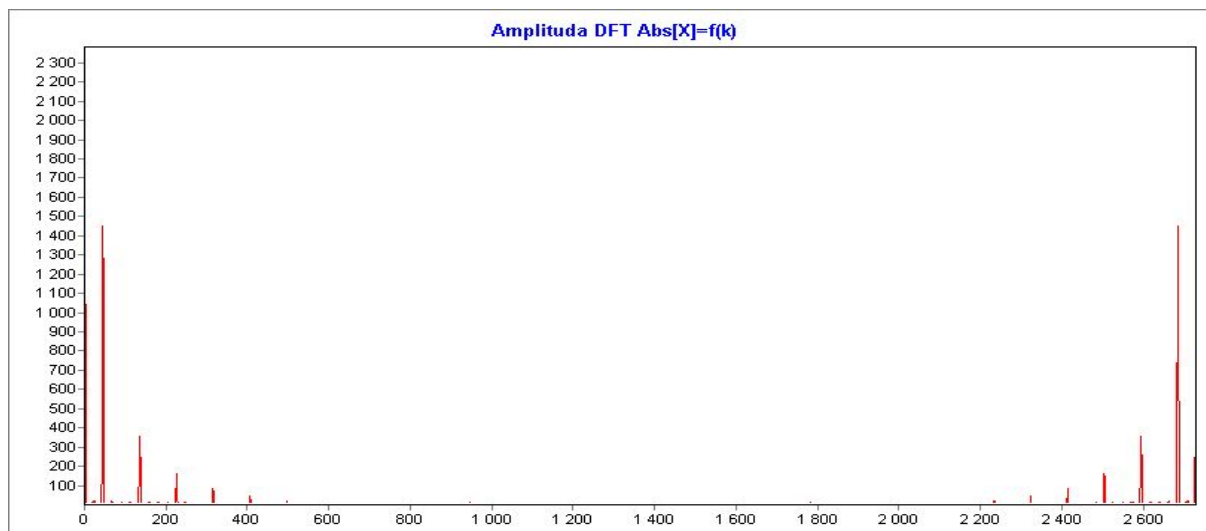
Rysunek 8.1. Przykład przebiegu piłokształtnego.



Rysunek 8.2. Przebieg z rysunku 8.1 po wygładzaniu.

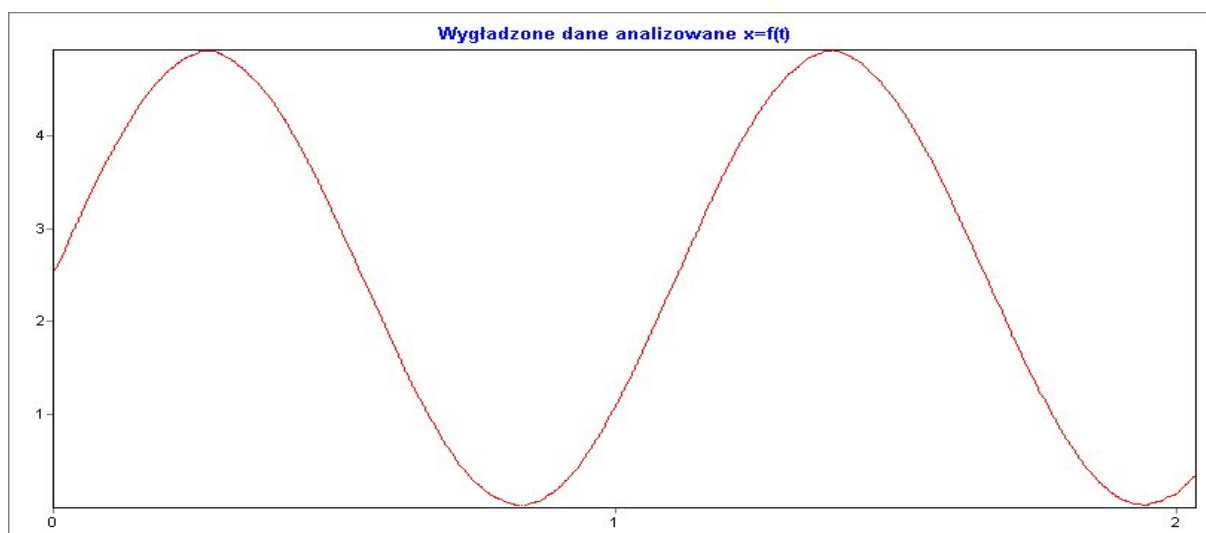


Rysunek 8.3. Widmo amplitudowe DFT przebiegu prostokątnego.

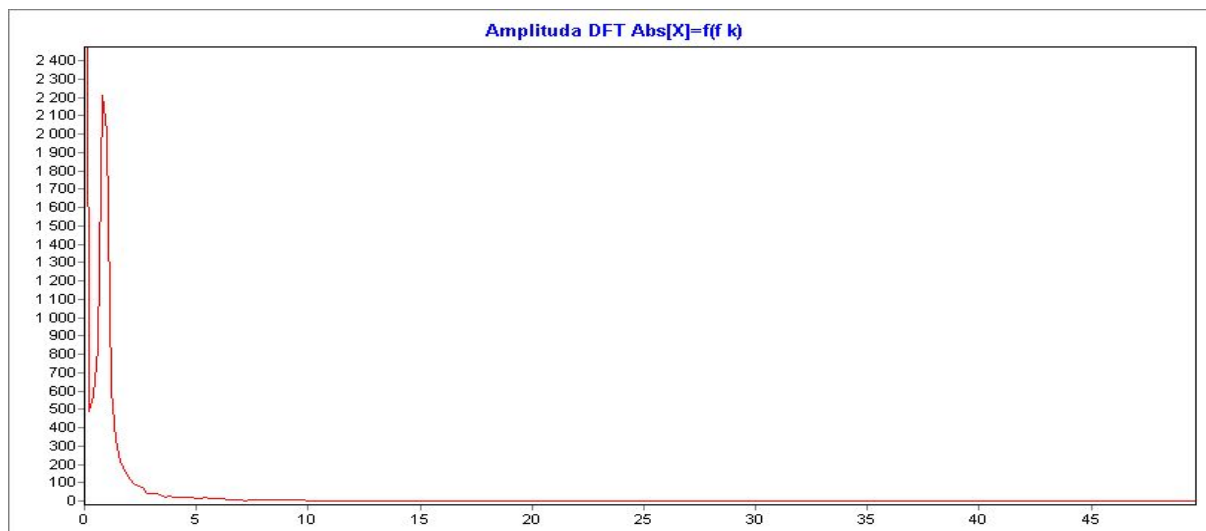


Rysunek 8.4. Widmo amplitudowe DFT przebiegu prostokątnego z zastosowaniem okienkowania i wygładzania.

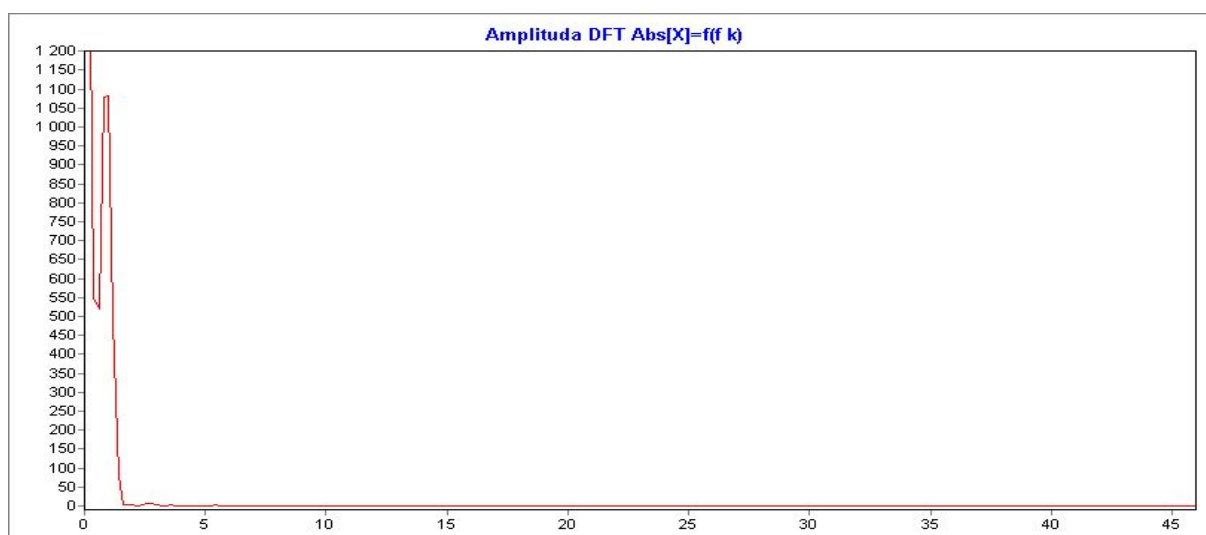
Rysunek 8.5 pokazuje prawidłowo próbkowany sygnał sinusoidalny o częstotliwości 1 Hz (wartość ustawiana na generatorze G 432) po wygładzeniu. Natomiast na rysunku 8.6 widzimy prawidłowo odczytaną częstotliwość tego przebiegu. Widać, że jest ona mniejsza od 1 Hz. Wynika to ze słabej jakości generowanego sygnału - rysunek 8.5 pokazuje, że pełen okres trwa nieco powyżej 1 s. Rysunek 8.6 przedstawia rozkład częstotliwości tego samego przebiegu sinusoidalnego z zastosowaniem okienka Harrisa.



Rysunek 8.5. Przykład sygnału sinusoidalnego wygładzonego.

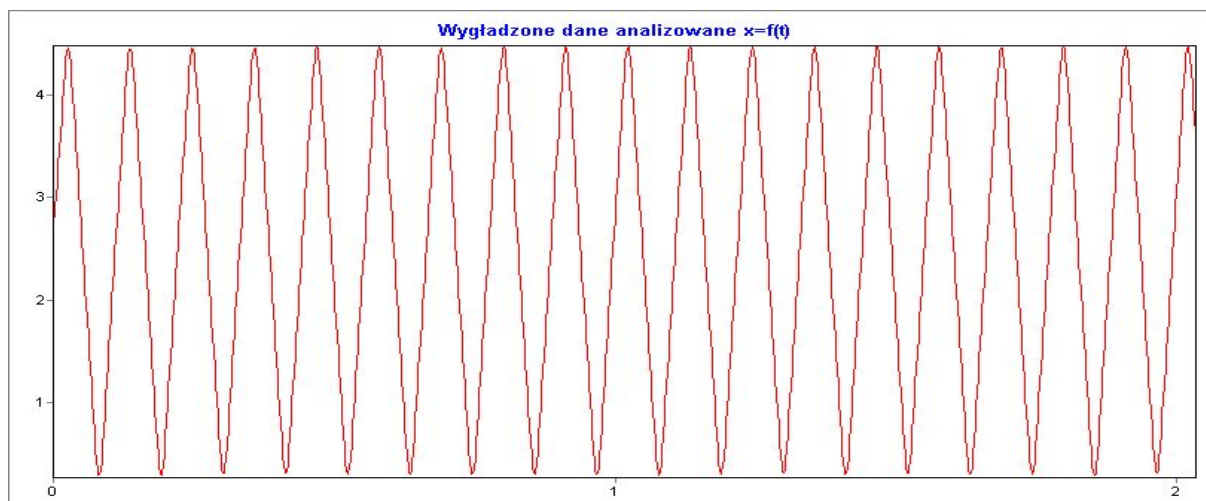


Rysunek 8.6. Widmo amplitudowe DFT przebiegu sinusoidalnego z rysunku 8.5.

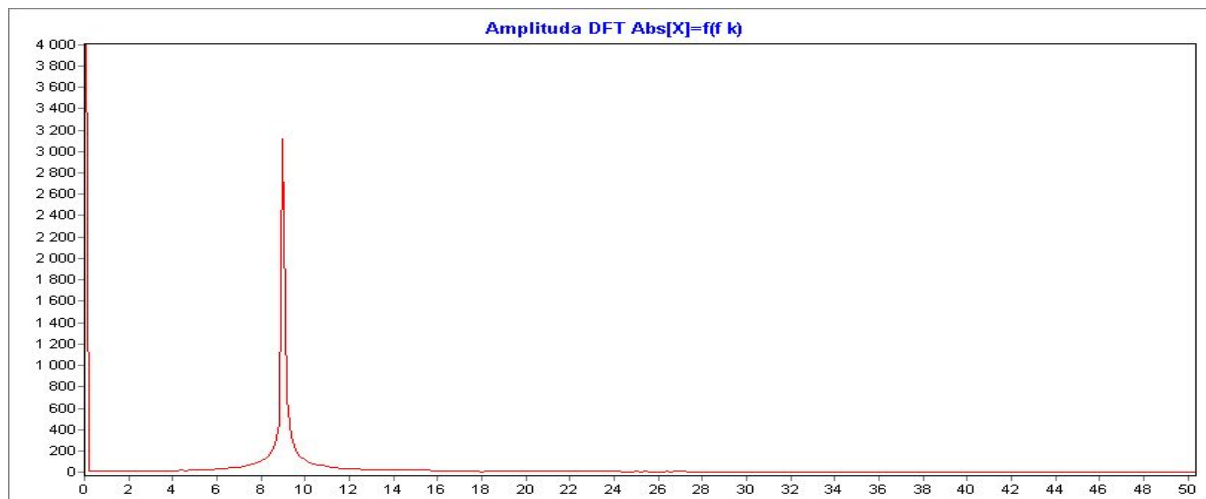


Rysunek 8.7. Widmo amplitudowe DFT przebiegu sinusoidalnego z rysunku 8.5 z zastosowaniem okienkowania.

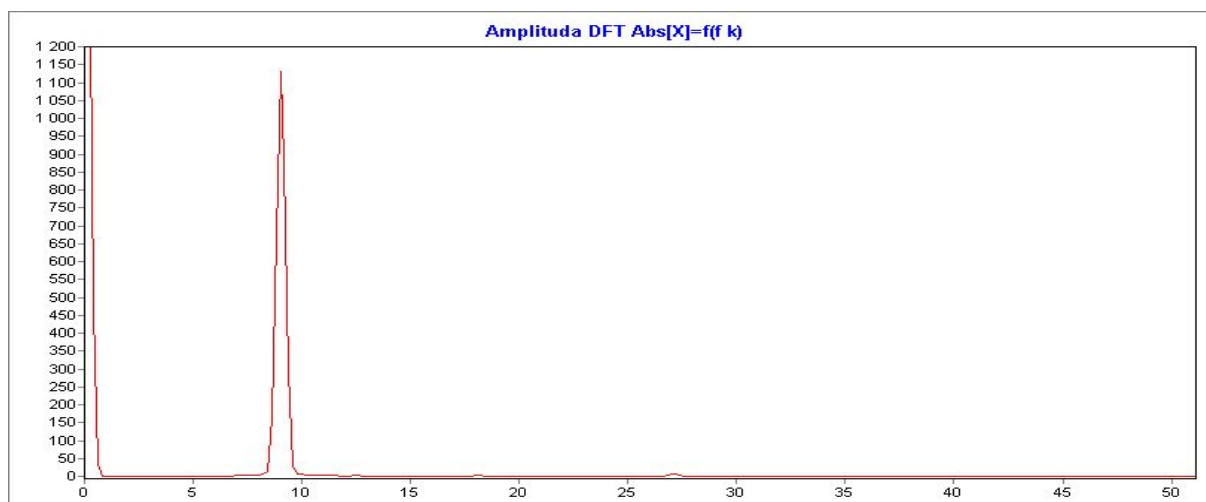
Na rysunku 8.8 znajduje się przykład przebiegu sinusoidalnego (10 Hz) próbkowanego z szybkością 500 próbek na sekundę. Na rysunkach 8.9 i 8.10 pokazane jest widmo amplitudowe przebiegu sinusoidalnego z rysunku 8.8. Na rysunkach 8.11 i 8.13 przedstawiono nieprawidłowo próbkowane sygnał sinusoidalne o częstotliwości 100 Hz i 1000 Hz. Rysunki 8.12 i 8.14 pokazują dodatkowe błędy wprowadzone przez nieprawidłowe wygładzanie (zmniejszenie amplitudy przebiegów).



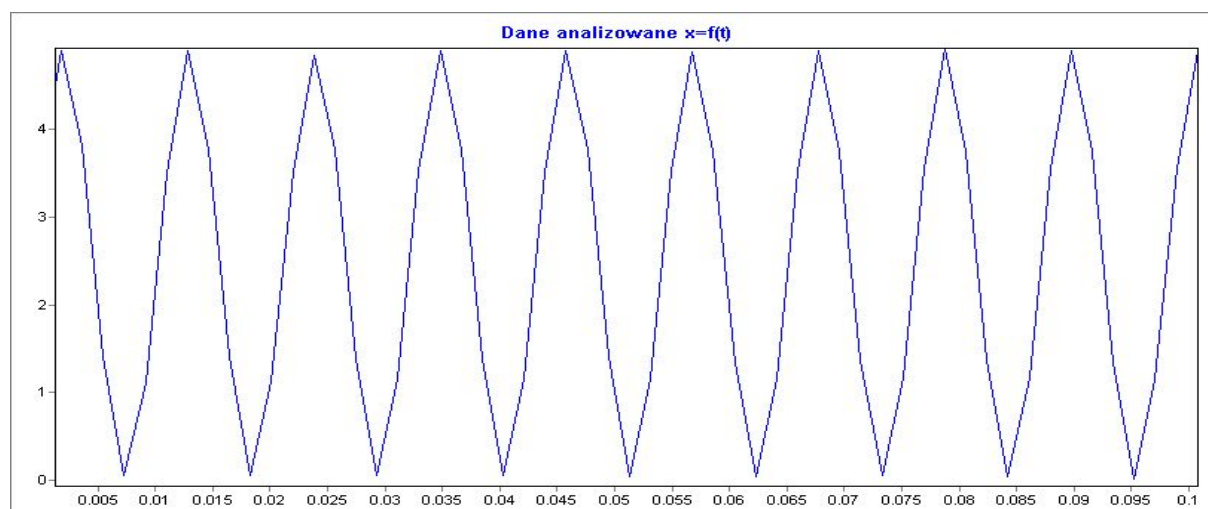
Rysunek 8.8. Przykład sygnału sinusoidalnego o częstotliwości 10 Hz.



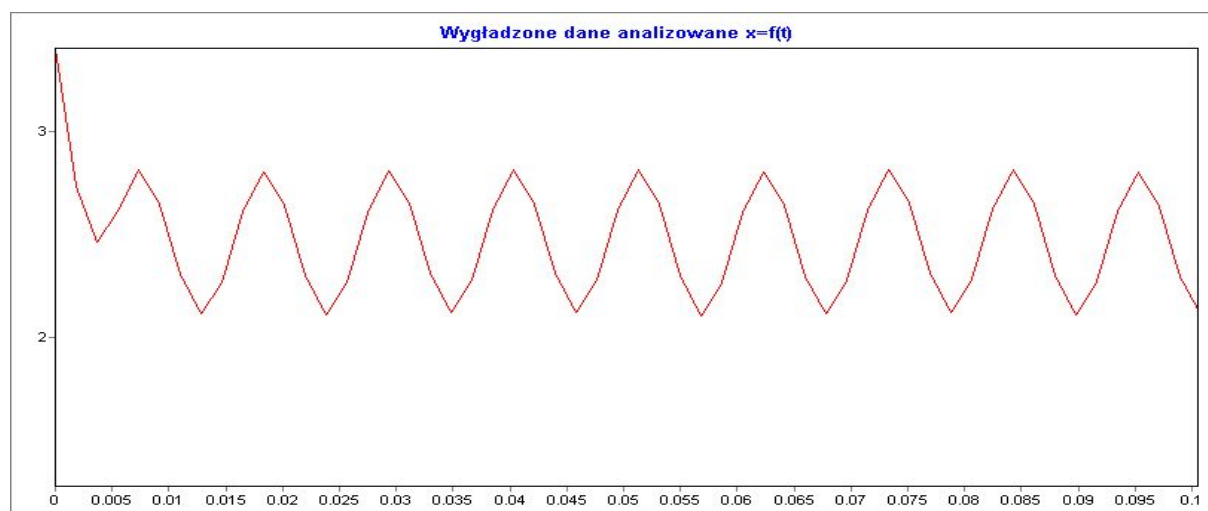
Rysunek 8.9. Widmo amplitudowe DFT przebiegu sinusoidalnego z rysunku 8.8.



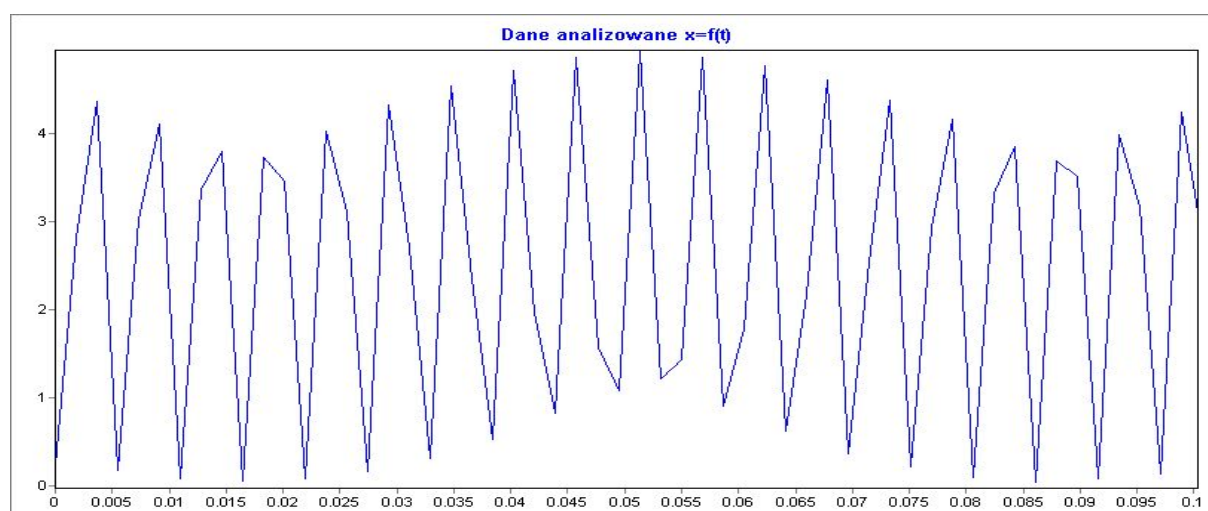
Rysunek 8.10. Widmo amplitudowe DFT przebiegu sinusoidalnego z rysunku 8.9 z zastosowaniem okienkowania.



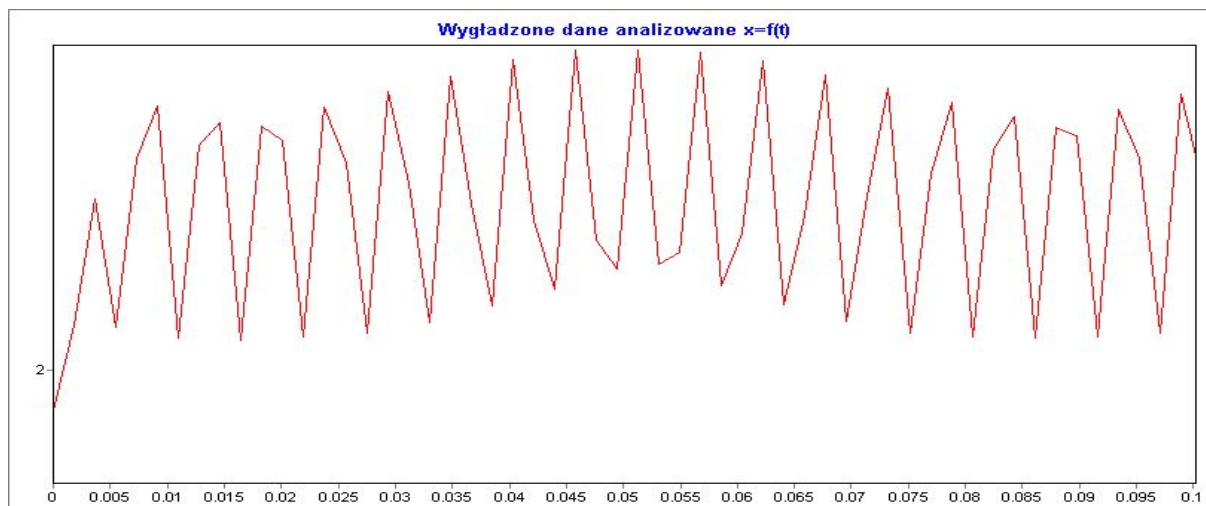
Rysunek 8.11. Nieprawidłowo próbkowany przebieg sinusoidalny.



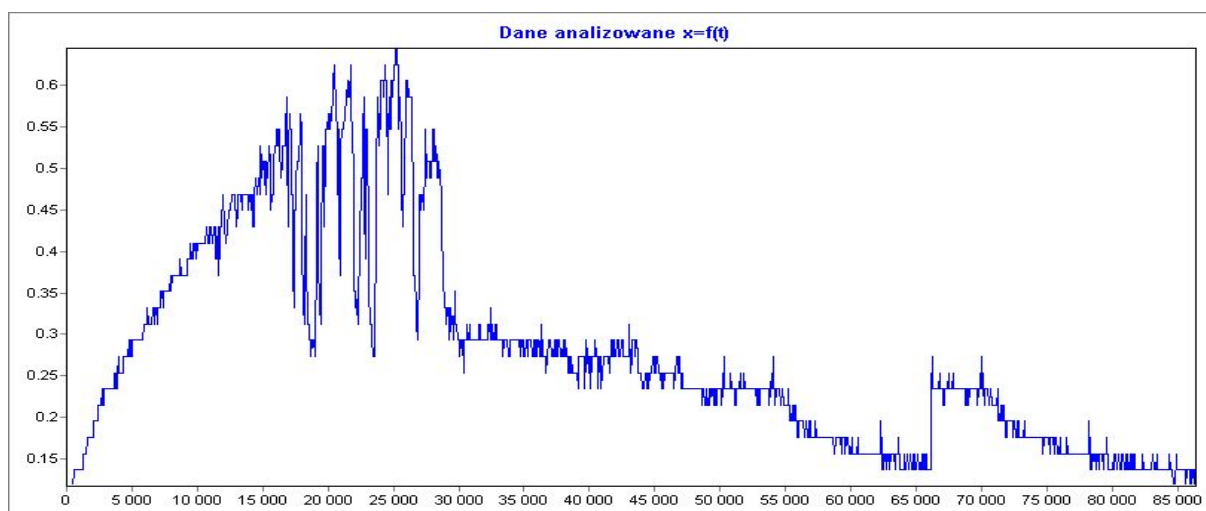
Rysunek 8.12. Próba wygładzenia przebiegu z rysunku 8.11.



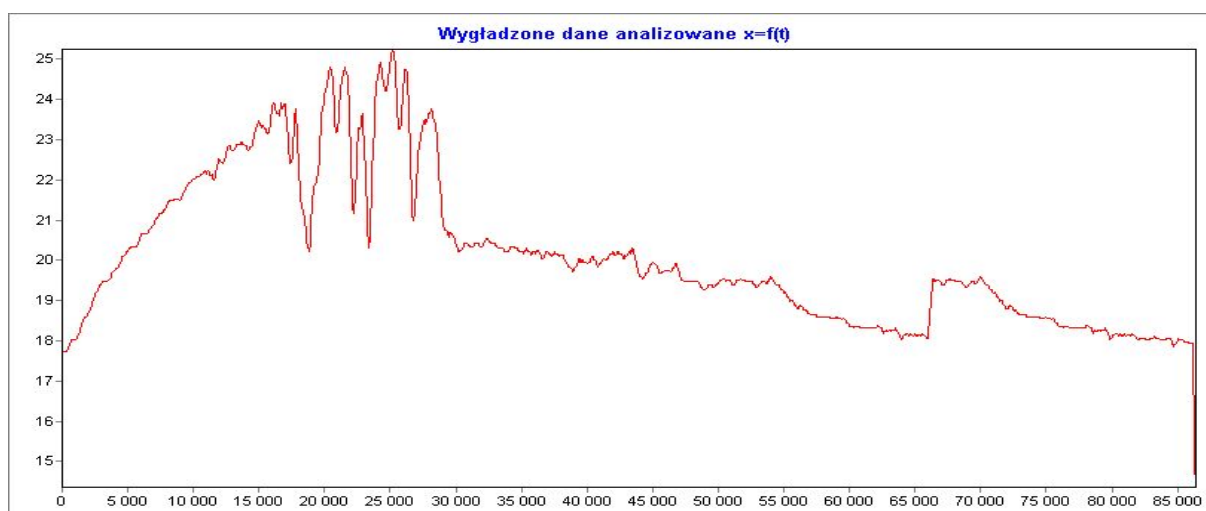
Rysunek 8.13. Nieprawidłowo próbkowany przebieg sinusoidalny.



Rysunek 8.14. Próba wygładzenia przebiegu z rysunku 8.13.



Rysunek 8.15. Wyniki pomiaru temperatury.



Rysunek 8.16. Wyniki pomiaru temperatury po wygładzeniu i kalibracji.

Na rysunku 8.15 przedstawiony jest wynik całodobowego pomiaru temperatury, natomiast na rysunku 8.16 jest widoczny pomiar temperatury po wygładzeniu i kalibracji (na osi rzędnych znajduje się fizyczna wartość amplitudy sygnału, w tym wypadku są to stopnie Celsjusza).

Poniżej widoczny jest fragment logu, tworzonego podczas obliczeń statystycznych. Zawiera on dane pliku pomiarowego (nazwa, liczba kanałów i pomiarów). Możemy z niego odczytać min.: maksymalną temperaturę otoczenia wynoszącą 25,54 °C, minimalną temperaturę otoczenia równą 17,73 °C i średnią temperaturę dobową wynoszącą 20.11 °C. Ponadto mamy rozstęp (różnicę pomiędzy maksymalną i minimalną temperaturą) równy 7,80 °C i wartość środkową: 19,47 °C. Dalej mamy statystyki rozproszenia i kształtu rozkładu.

```

0=====0
|                               Plik danych pomiaru.                               |
0=====0

```

```

Plik danych pomiarowych: D:\Gloomy\Pomiary\Temperatura24h.dat
Liczba kanałów: 1
Liczba pomiarów: 1441

```

```

0=====0
|                               Dane analizowane x=f(t).                               |
0=====0

```

```

Kanał pomiarowy: 1
Liczba próbek: 1441
Od próbki nr: 0
Do próbki nr: 1440

```

```

Maksymalna wartość próbki: 0.64453125 (nr pr: 419 czas:25140)
Minimalna wartość próbki: 0.1171875 (nr pr: 0 czas: 0)

```

```

0=====0
|                               Kalibrowane dane analizowane x=f(n).                               |
0=====0

```

```

Dolna wartość zakresu pomiarowego: 0
Górna wartość zakresu pomiarowego: 5
Okres próbkowania: 60

```

```

Maksymalna wartość próbki: 25.5390625 (nr pr: 419 czas:25140)
Minimalna wartość próbki: 17.734375 (nr pr: 0 czas: 0)

```

```

Średnia arytmetyczna: 20.1146773074254
Rozstęp: 7.8046875
Mediana: 19.46875
Odchylenie przeciętne od wartości średniej: 1.4523947004571
Odchylenie przeciętne od wartości mediany: 1.4057945870923
Odchylenie standardowe: 1.86441450464048
Wariancja: 3.47604144511379

```

```

Współczynnik koncentracji: 0.259719132352294
Współczynnik asymetrii: 0.155002300641514

```

```

Moment centralny rzędu 2: 0.00228003939438534
Moment centralny rzędu 3: 6.51017978337206
Moment centralny rzędu 4: 37.9178519979159

```

Moment zwykły rzędu 2: 408.076284626968
Moment zwykły rzędu 3: 8354.67186581931
Moment zwykły rzędu 4: 172701.518581055

9. Podsumowanie.

Celem pracy było opracowanie, wykonanie i oprogramowanie cyfrowego systemu do pomiaru i akwizycji danych, sterowanego mikrokontrolerem. Wymagało to zapoznania się z budową i zasadą działania mikrokontrolerów i układów cyfrowych jak również z metodami programowania pod Windows za pomocą interfejsu programowania Win32 API.

Opracowany system **JKSPv2** (układ pomiarowy i oprogramowanie) jest w pełni funkcjonalnym urządzeniem pomiarowym. Wyposażony jest w trójkanałowy 8-bitowy przetwornik analogowo-cyfrowy. Pozwala na pracę jako samodzielny układ pomiarowy zdolny do przechowywania zebranych danych, bądź podobnie jak karta A/C może tylko dokonywać przetwarzania analogowo-cyfrowego i przysyłać dane do stanowiska komputerowego. Pomiary można prowadzić z maksymalną prędkością 20 próbek/s – bez względu na ilość kanałów pomiarowych i zapisie do pamięci – lub 500 próbek/s podczas jednokanałowego pomiaru z bezpośrednią transmisją do komputera. Układ jest zdolny zapamiętać po 65535 próbek z każdego kanału pomiarowego. Ponadto posiada dwa zakresy pomiarowe. System pomiarowy **JKSPv2** może być zasilany z zewnętrznego źródła zasilania lub bateryjnie. Zaletą takiego rozwiązania jest zwiększona uniwersalność układu pomiarowego. Daje on możliwość dokonywania pomiarów w miejscach trudnodostępnych, niezależnie od obecności źródła zasilania i komputera. Po dokonaniu pomiaru, nawet po odłączeniu od zasilania, dane nie znikną dzięki zastosowaniu pamięci EEPROM, jako nośnika danych. Układ charakteryzuje się małymi gabarytami. Program do komunikacji **JKSP Communicator** w prosty sposób pozwala na przetransmitowanie zebranych danych do komputera PC. Dane te, mogą być zapisane do pliku tekstowego (w którym zawarte są też dane o prędkości próbkowania i zakresie pomiarowym) co daje dużą uniwersalność wczytania ich do profesjonalnych systemów analizy danych. **JKSP Communicator** może również tylko wyświetlać wyniki pomiarów przeprowadzonych w czasie rzeczywistym, bez ich zapisywania do pliku. Całość systemu **JKSPv2** dopełnia program **JKSP Analyser**, który służy do podstawowej analizy danych pomiarowych. Pozwala on na wybór zakresu analizy, edycję i wizualizację danych. Ponadto można dokonać kalibracji sygnału, wygładzenia sygnału, obliczenia dyskretnej transformaty Fouriera i podstawowej charakterystyki statystycznej.

Dalszy rozwój zaproponowanego rozwiązania powinien przebiegać w kierunku rozbudowy oprogramowania do analizy oraz zwiększenia szybkości próbkowania i rozdzielczości przetwornika A/C. Zastosowanie pamięci EEPROM do zapisu pomiarów powodowało samodzielność układu (brak komputera PC podczas pomiaru), ale kosztem zmniejszenia prędkości próbkowania. Zwiększenie częstotliwości próbkowania wymaga zastąpienia pamięci typu EEPROM, pamięcią

typu RAM. Ta jednak traci swą zawartość po odłączeniu zasilania. Optymalnym rozwiązaniem byłoby zastosowanie pamięci RAM do czasowego przechowywania pomiarów i przegrywanie ich do innych nośników po zakończeniu pomiaru. Do tego celu można wykorzystać karty pamięci typu Flash lub napędy Microdrive. Są to urządzenia dysponujące pojemnościami rzędu dziesiątek megabajtów (a nawet 1 GB - IBM Microdrive).

Wykonane pomiary i przeprowadzona analiza uzyskanych wyników potwierdzają przydatność opracowanego systemu pomiarowego.

- [1] *2-wire Serial EEPROM 64K. AT24C512.*
Atmel Data Sheet.
- [2] *8-bit Microcontroller with 8K Bytes Flash. AT89S8252.*
Atmel Data Sheet.
- [3] Dańca W., *Mikrokontrolery. Od układów 8-bitowych do 32-bitowych.*
Mikom, Warszawa 2000.
- [4] Dreszer J., Koch E., *Matematyka. Poradnik inżyniera.*
WNT, Warszawa 1971.
- [5] Gałka P., Gałka P., *Podstawy programowania mikrokontrolera 8051.*
Mikom, Warszawa 1995.
- [6] *HIN230 thru HIN241.*
Harris Semiconductor, 1997.
- [7] Jackiewicz B., *Przetwarzanie analogowo-cyfrowe w nowoczesnych systemach pomiarowych.* Metrologia w systemach jakości – 3. VI Sympozjum Klubu PF ISO 9000.
PŚk, Kielce 2000.
- [8] Lachand-Robert T., *Programowanie obiektowe w Turbo Pascalu.*
Helion, Gliwice 1993.
- [9] Lyons R. G., *Wprowadzenie do cyfrowego przetwarzania sygnałów.*
WKiŁ, Warszawa 1999.
- [10] Paślawski A., *Programowanie w Delphi 5.0.*
Edition 2000, Kraków 2000.
- [11] *PCF8591 8-bit A/D and D/A converter.*
Philips Data Sheet, 1997.
- [12] Pełka R. *Mikrokontrolery – architektura, programowanie, zastosowania.*
WKiŁ, Warszawa 1999.
- [13] Plassche R., *Scalone przetworniki analogowo-cyfrowe i cyfrowo-analogowe.*
WKiŁ, Warszawa 1997.
- [14] Oppenheim A. V., Schafer R. W., *Cyfrowe przetwarzanie sygnałów.*
WKiŁ, Warszawa 1979.
- [15] *Programowany zegar z wyświetlaczem LCD.*
Elektronika Praktyczna 7/2000.
- [16] *Przetwornik AC/CA z interfejsem I²C.*
Elektronika Praktyczna 10/2000.

- [17] Strzałkowski K., *Podstawy Delphi*.
Stachurski, Kielce 2000.
- [18] Teixeira S., Pacheco X., *Delphi 4. Vademecum profesjonalisty*.
Helion, Gliwice 1999.
- [19] Walkowiak T., *DSP. W-V-3*.
- [20] <http://www.phys.uni.torun.pl/~peplow/sygnaly>
Analiza sygnałów.
- [21] <http://www.ep.com.pl>
Elektronika Praktyczna.
- [22] <http://delphi5.topcities.com>
Delphi Page.
- [23] <http://delphi.cartall.com.pl>
Delphi FAQ.
- [24] <http://www.delphiqa.org.pl>
Delphi w Pytaniach i Odpowiedziach.
- [25] <http://www.wpk.p.lodz.pl/~dsakson/fourier>
Szybkie Przekształcenie Fouriera.
- [26] <http://wms.mat.agh.edu.pl/~wojda/zbior/node17>
Szeregi Fouriera.
- [27] <http://helios.et.put.poznan.pl/~mhalbs/download/TeoriaSygnaLOW-Wzory>
Teoria Sygnałów.
- [28] <http://www.webwareindex.com/tutorials/API>
Using Win32 API.
- [29] <http://www.iseran.com/Win32/FAQ>
Win32 Programming FAQ.