BIBLIOTEKA
POLSKIEGO KRÓTKOFALOWCA 20

Krzysztof Dąbrowski
OE1KDA

Arduino w krótkofalarstwie
Tom 1

Wiedeń 2013

© Krzysztof Dąbrowski OE1KDA
Wiedeń 2013

Opracowanie niniejsze może być rozpowszechniane i kopiowane na zasadach niekomercyjnych w dowolnej postaci (elektronicznej, drukowanej itp.) i na dowolnych nośnikach lub w sieciach komputerowych pod warunkiem nie dokonywania w nim żadnych zmian i nie usuwania nazwiska autora. Na tych samych warunkach dozwolone jest tłumaczenie na języki obce i rozpowszechnianie tych tłumaczeń.
Na rozpowszechnianie na innych zasadach konieczne jest uzyskanie pisemnej zgody autora.

Arduino w krótkofalarstwie

Tom 1

Krzysztof Dąbrowski OE1KDA

Wydanie 1
Wiedeń, listopad 2013

Spis treści

Wstęp

Mikroprocesory i mikrokomputery znalazły już od dawna zastosowanie w krótkofalarstwie w układach pomiarowch, sterujących lub służących do przetwarzania i dalszej transmisji sygnałów. Możliwości sprzętowych jest dużo podobnie jak dużo jest rozmaitych typów procesorów i przeznaczonych dla nich zestawów uruchomieniowych i ewaluacyjnych. W ostatnich latach pojawiło się też szereg mikro- i minikomputerów o małych wymiarach, niskim poborze energii i co najważniejsze oszczędzających użytkownikom konieczności konstruowania i uruchamiania własnych rozwiązań tego typu. Mogą więc oni skoncentrować się na doborze układów peryferyjnych lub ich konstrukcjach oraz na pisaniu programów spełniających ich wymagania.
Do stosunkowo nieskomplikowanych w użyciu i wykorzystaniu rozwiązań należy Arduino. Zostało ono zresztą opracowane z myślą o „laikach" komputerowych czyli osobach nie zajmujących się zawodowo programowaniem i informatyką. Oprócz całego szeregu płytek Arduino o różnych możliwościach technicznych i stopniach rozbudowy dostępne są różnorodne moduły rozszerzeń do nich a zresztą konstrukcja nowych do własnych potrzeb też nie przedstawia większej trudności. Przyczyniło się to w znacznym stopniu do rozpowszechnienia się Arduino także wśród krótkofalowców.
Autor korzystał poprzednio z prostych układów mikroprocesorowych (opartych na procesorach PIC z serii 16Fxxx) własnej konstrukcji ale ostatnio coraz częściej wybiera do tych celów Arduino. Nie znaczy to oczywiście, że w takich czy innych przypadkach własna konstrukcja nie okaże się bardziej celowa ale przeważnie gotowa płytka z działającym procesorem i urządzeniami peryferyjnymi oszczędza mnóstwo czasu i pozwala skoncentrować się na sprawach istotnych dla danego projektu.
Możliwości Arduino i podobnych układów wystarczają wprawdzie do wielu zastosowań krótkofalarskich ale dla bardziej skomplikowanych konieczne jest użycie minikomputerów PC w rodzaju Raspberry Pi, BeagleBoard miniaturowych komputerów PC z Androidem i podobnych. Są to płytki o wielkości zbliżonej do kart kredytowych wyposażone w procesory dostatecznie szybkie aby mogły na nich pracować specjanie przystosowane wersje Linuksa, posiadające szereg standardowych złączy takich jak HDMI, Ethernet czy USB i pozwalające na podłączenie standardowych urządzeń peryferyjnych: klawiatur, podsystemów dźwiękowych itd. W zastosowaniach krótkofalarskich można dzięki temu korzystać z linuksowych wersji programów komunikacyjnych dla różnych emisji cyfrowych, pracują one także w radiowo-internetowych bramkach echolinkowych i APRS.
Konstrukcje te ze względu na oferowane możliwości i stopień skomlikowania wymagają oddzielnego omówienia dlatego też w obecnym skrypcie będziemy o nich wspominać tylko na marginesie. W odróżnieniu od Arduino są one zresztą przeznaczone dla osób mających więcej doświadczenia informatycznego i potrafiących zainstalować od zera system operacyjny, skompilować do niego potrzebne programy, skonfigurować je i uruchomić korzystając z wiersza poleceń a nie z wygodnej graficznej powierzchni obsługi jak w przypadku Windowsów – przynajmniej na początku.
Skrypt niniejszy jest przeznaczony dla osób mających pewne doświadczenie w programowaniu (niekoniecznie na Arduino) i nie jest pomyślany jako metodyczny kurs dla zupełnie początkujących. Zawarte w nim wiadomości dotyczące języka programowania Arduino i dodatkowych bibliotek mają charakter encyklopedyczny. Wiadomości zawarte w rozdziale 1 mają jedynie ułatwić zrozumienie omawianych dalej programów i nie mogą zastąpić systematycznej nauki.
Czytelników pragnących nauczyć się programowania od podstaw zachęcamy do skorzystania z innej literatury. Kurs programowania Arduino na CD jest dostępny w internetowym sklepie AVT. Dużym ułatwieniem w opanowaniu programowania Arduino jest choćby pobieżna znajomość języka C (C++) ale nie jest to warunkiem koniecznym. Oprócz najwygodniejszego, zwłaszcza dla mniej zaawansowanych, rozwiązania wykorzystującego opisane dalej środowisko programistyczne i nieskomplikowany język wzorowany na C i C++ możliwe jest także programowanie Arduino w specjalnie dla niego przygotowanej odmianie Basicu – Bascom lub w klasycznym języku C. Ich omówienie przekraczałoby jednak ramy obecnego opracowania.
Programy zamieszczone w skrypcie stanowią zamknięte całości i mogą być z niego bezpośrednio skopiowane do edytora Arduino a następnie po ewentualnym dopasowaniu niektórych danych (znaki wywoławcze, imiona, lokalizacja stacji, moc nadajnika, hasła dostępu, nazwa użytkownika, adresy IP i MAC, nazwy sieci lokalnych itp.) skompilowane i załadowane do pamięci procesora. Z tego też powodu w komentarzach zrezygnowano z polskich liter i znaków specjalnych. W żadnym z przykładów nie usunięto też części powtarzających się w innych aby oszczędzić czytelnikom ich poszukiwania. Ich

wykorzystanie może jednak wymagać uprzedniego zainstalowania dodatkowych potrzebnych im bibliotek. Ich pliki nagłówkowe są wymieniane najczęściej na początku programów w poleceniach *#include*. Kolejność umieszczenia programów w skrypcie uzależniona jest od (subiektywnej) oceny ich stopnia trudności względnie pewnych powiązań tematycznych – dotyczy to przykładowo grupy rozwiązań korzystających z programowej syntezy sygnałów m.cz. albo grupy programów sterujących scalonymi syntezerami cyfrowymi. Chęć przytoczenia pełnych kodów źródłowych i znaczna ilość materiału spowodowały konieczność podziału opracowania na kilka tomów.
Do najważniejszych w praktyce krótkofalarskiej grup tematycznych należy zaliczyć generację sygnałów (wzorcowych) różnych emisji w zakresie m.cz. i w.cz., transmisję różnorodnych komunikatów w tym APRS, DPRS i telemetrycznych, sprawy związane z dostępem do internetu lub sieci lokalnych, odbiór wzorcowych sygnałów czasu i ich dekodowanie, usprawnienie pomiarów różnych wielkości, zdalne sterowanie sprzętem krótkofalarskim i wiele innych.
Część z przytoczonych programów stanowią proste przykłady ilustrujące sposób rozwiązania danego problemu lub też nadające się do włączenia do większej całości po ewentualnym rozszerzeniu funkcjonalności. Są też jednak wśród nich kompletne rozwiązania gotowe do praktycznego zastosowania jedynie po dopasowaniu najważniejszych parametrów. Również i one mogą być jednak stosunkowo łatwo rozbudowywane tak aby odpowiadały potrzebom użytkownika.

Krzysztof Dąbrowski OE1KDA
Wiedeń
Listopad 2013

Arduino

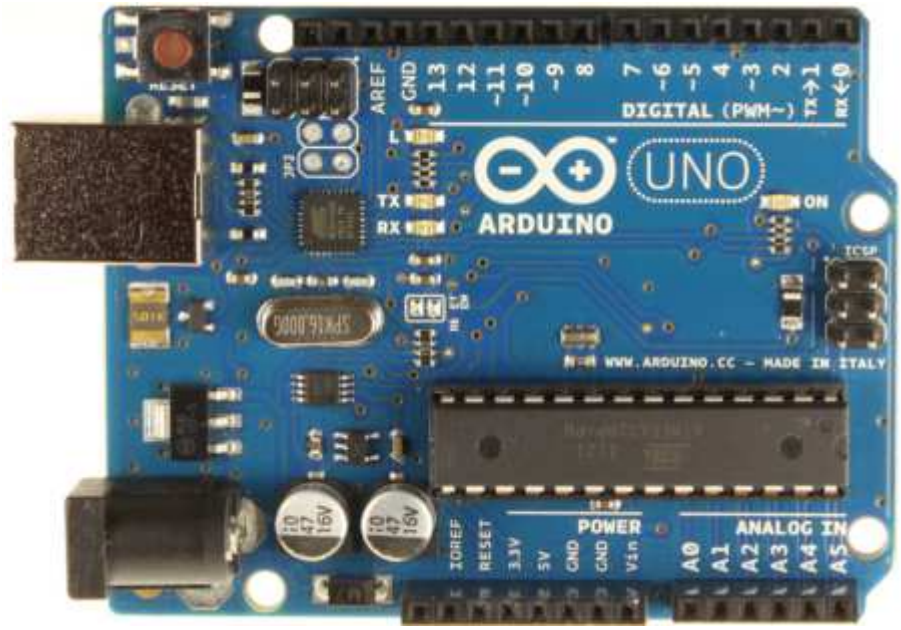
Arduino jest nieskomplikowanym i niedrogim mikrokomputerem jednopłytkowym opartym na mikroprocesorach z rodziny Atmega. Najczęściej używane są obecnie mikroprocesory Atmega168, Atmega328 i Atmega2560. W zależności od modelu mikrokomputera i użytego w nim procesora pamięci programu mają pojemności 16 – 512 kB, pamięci robocze 1–96 kB a pamięci nieulotne EEPROM 0,512 – 4 kB. Z zapisu i odczytu pamięci EEPROM mogą korzystać pracujące na mikrokomputerze programy dlatego też obszar ten jest przeznaczony w pierwszym rzędzie dla rzadko zmieniających się danych j.np. parametrów konfiguracyjnych czy kalibracyjnych w układach pomiarowych. Dane te nie ulegają skasowaniu po wyłączeniu zasilania ale ponieważ liczba cykli zapisu jest stosunkowo ograniczona nie należy wykorzytywać tej pamięci jako pamięci roboczej.
Większość modeli posiada kilka wejść analogowych, kilka wyjść sygnałów z modulacją szerokości impulsów (ang. *PWM*) i kilkanaście wejść/wyjść logicznych (patrz tabela 1.1). Są one przeważnie doprowadzone do listew kontaktowych umieszczonych na krawędziach płytki. Konstrukcja ta pozwala na wtykanie do nich modułów rozszerzeń zwanych w literaturze angielskiej *shield* czyli w dosłownym tłumaczeniu – tarcza. W gwarze Arduino programy noszą angielską nazwę *sketch* czyli szkic, my jednak pozostaniemy przy polskiej terminologii.
Większość modeli jest wyposażona w gniazdo USB wykorzystywane do połączenia z komputerem PC na czas ładowania i uruchamiania programów. Gniazdo to może służyć także do zasilania mikrokomputera napięciem 5 V chociaż dodatkowo możliwe jest zasilanie przez oddzielne gniazdo napięciami 7–9 lub 7–12 V. W modelach Leonardo i DUE do gniazd USB można podłączyć komputerowe klawiatury i myszy. W modelach nieposiadających gniazda USB do ładowania programu służy przeważnie złącze ICSP. Złącze to wykorzystywane jest także do ładowania programów skompilowanych przez kompilatory innych języków.
Mikrokomputery Arduino zostały opracowane z myślą o użytkownikach mających niewielkie doświadczenie w programowaniu (przykładowo osobach związanych ze sztuką i pragnących osiągnąć różne efekty dźwiękowe lub optyczne) i dlatego zarówno język programowania jest stosunkowo prosty jak i środowisko programistyczne jest łatwe w instalacji i obsłudze. W modelach wyposażonych w gniazdo USB programy są ładowane do pamięci procesora za jego pośrednictwem bez konieczności korzystania z oddzielnego programatora. W tym celu procesory zamontowane na płytkach Arduino posiadają wprowadzony program ładujący (ang. *bootloader*) zajmujący ok. 0,5 – 2 kB pamięci programu. Możliwe jest także ładowanie programu bezpośrednio przez znajdujące się na płytce złącze ICSP ale bywa to w praktyce rzadziej stosowane. Może ono służyć przykładowo do ładowania programów skompilowanych przez inny dowolny kompilator np. języka C dla procesorów AVR. Konieczne jest wówczas użycie dodatkowego programatora co oznacza kolejne inwestycje.
Generator częstotliwości zegarowej jest stabilizowany kwarcem dzięki czemu jest ona wystarczająco stabilna i dokładna dla większości zastosowań.
Zestawienie najważniejszych parametrów technicznych najczęściej używanych modeli zawiera tabela 1.1. Bieżące informacje o dostępnych modelach mikrokomputerów, bibliotekach programów, aktualnych wersjach środowiska programistycznego, wiele praktycznych przykładów programów, porad itp. zawiera witryna *www.arduino.cc*. Dla ułatwienia zrozumienia praczy programów krótkofalarskich i ich modyfikacji przytaczamy niektóre z tych przykładów w dodatkach.
Zarówno schematy Arduino jak i opracowane dla nich programy dostępne są bezpłatnie w internecie i mogą być dalej rozpowszechniane na tych samych zasadach jednak ze względu na zastrzeżenia konstruktorów dotyczące nazwy na rynku pojawiły się modele o nazwach zbliżonych ale jednak różniących się od oryginału j.np. Freeduino, AVTDuino.
Zalety Arduino zostały stosunkowo szybko po jego pojawieniu się odkryte przez krótkofalowców i dzięki temu istnieje wiele rozwiązań i programów radiomatorskich, poczynając od źródeł sygnałów różnych emisji (nazywanych w dalszej części skryptu radiolatarniami chociaż ich zastosowanie nie ogranicza się tylko do tego celu – mogą one przykładowo służyć jako generatory laboratoryjne do różnego rodzaju pomiarów sprzętu i kanałów transmisyjnych albo jako nadajniki danych telementrycznych nawet tylko o bardzo ograniczonym zasiegu) poprzez układy sterujące i pomiarowe aż do bramek radiowo-internetowych APRS lub serwerów dostępnych przez internet albo tylko w sieciach lokalnych.
Mikrokomputery Arduino (AVTDuino) wraz z modułami rozszerzeń a także dyski CD z oprogramowaniem i kursem programowania „Elektroniki Praktycznej” są dostępne w Polsce m.in. w sklepie AVT.

Popularność konceptu Arduino stała się bodźcem do opracowania rozwiązań podobnych ale opartych o mikroprocesory innych typów: PIC, ARM itd. W ich nazwach występuje najczęściej sylaba „duino”. Ważnymi zaletami rozwiązań opartych na tych i innych podobnych mikro- i minikomputerach są niski pobór energii i małe wymiary. W wielu przypadkach opłaca się więc zastąpić nimi rozwiązania oparte o komputery PC.
Do zastosowań wymagających większej mocy obliczeniowej i bardziej rozbudowanego wyposażenia (np. dla bramek radiowo-internetowych Echolinku) świetnie nadają się natomiast linuksowe minikomputery w rodzaju Raspberry Pi. Są one również coraz częściej używane przez krótkofalowców.

Fot. 1.1. Kusrs programowania Arduino Elektroniki Praktycznej

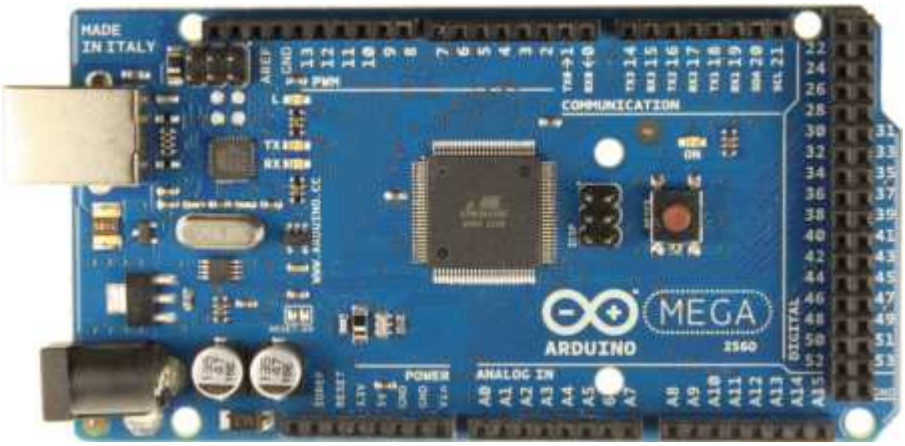
1.1 Rozwiązania układowe

Spośród wielu dostępnych i przystosowanych do różnych celów modeli Arduino autor wybrał kilka stosunkowo najpopularniejszych i najprzydatniejszych w zastosowaniach krótkofalarskich. Nie oznacza to wcale, że w pewnych szczególnych przypadkach nie okaże się praktyczniejszy któryś z pozostałych nieomówionych tutaj modeli. W swojej dotychczasowej praktyce autor korzystał z modeli Duemilanove i UNO.



Fot. 1.2. Arduino UNO

Fot. 1.3. AVTDuino (fot. ze sklepu AVT)



Fot. 1.4. Arduino MEGA 2560

Tabela 1.1. Parametry techniczne wybranych modeli Arduino

	Arduino UNO	**Arduino Duemila-nove, AVTDuino**	**Arduino Leonardo**	**Arduino DUE**	**Arduino MEGA 2560, MEGA ADK**	**Arduino Nano**
Procesor	Atmega328	Atmega168	Atmega32µ4	AT91SAM 3X8E	Atmega2560	Atmega168
		Atmega328				Atmega328
Takt	16 MHz	16 MHz	16 MHz	84 MHz	16 MHz	16 MHz
Wejścia analogowe	6	6	12	12	16	6
Wyjścia analogowe	0	0	0	2	0	0
We./wy. logiczne	14	14	20	54	54	14
Wyjścia PWM	6	6	7	12	15	6
Pamięć programu	32 kB	16 kB	32 kB	512 kB	256 kB	16 kB
		32 kB				32 kB
Pamięć robocza (RAM)	2 kB	1 kB	2,5 kB	96 kB	8 kB	1 kB
		2 kB				2 kB
Pamięć EEPROM	1 kB	0,512 kB	1 kB	-	4 kB	0,512 kB
		1 kB				1 kB
UART	1	1	1	4	4	1
Gniazdo USB	standard B	standard B	mikro	2 mikro	standard B	mini B
Zasilanie	5 / 7–12 V	5 / 7-12 V	5 / 7–12 V	3,3 / 7–12 V	5 / 7–12 V	5 / 7–9 V

Uwagi:

- Arduino Micro – parametry identyczne jak Leonardo, ale napięcia zasilania 5 V / 7–12 V.
- Arduino Mini – jak UNO ale bez gniazda USB i UART-u, 8 wejść i wyjść logicznych, zasilanie 5 V / 7–9 V.
- Arduino Explora – jak Leonardo ale bez wyprowadzonych wejść i wyjść logicznych, analogowych i PWM, za to wyposażony we własne czujniki.
- Arduino UNO jest następcą bardzo popularnego do niedawna modelu Arduino Duemilanove.
- Dla Arduino Duemilanove i Arduino Nano podano w górnych kratkach dane dla wersji z procesorem Atmega168 a w dolnej – z Atmega328 tam gdzie występują różnice.

1.1.1 Wyprowadzenia na płytkach Arduino Duemilanove i Arduino UNO

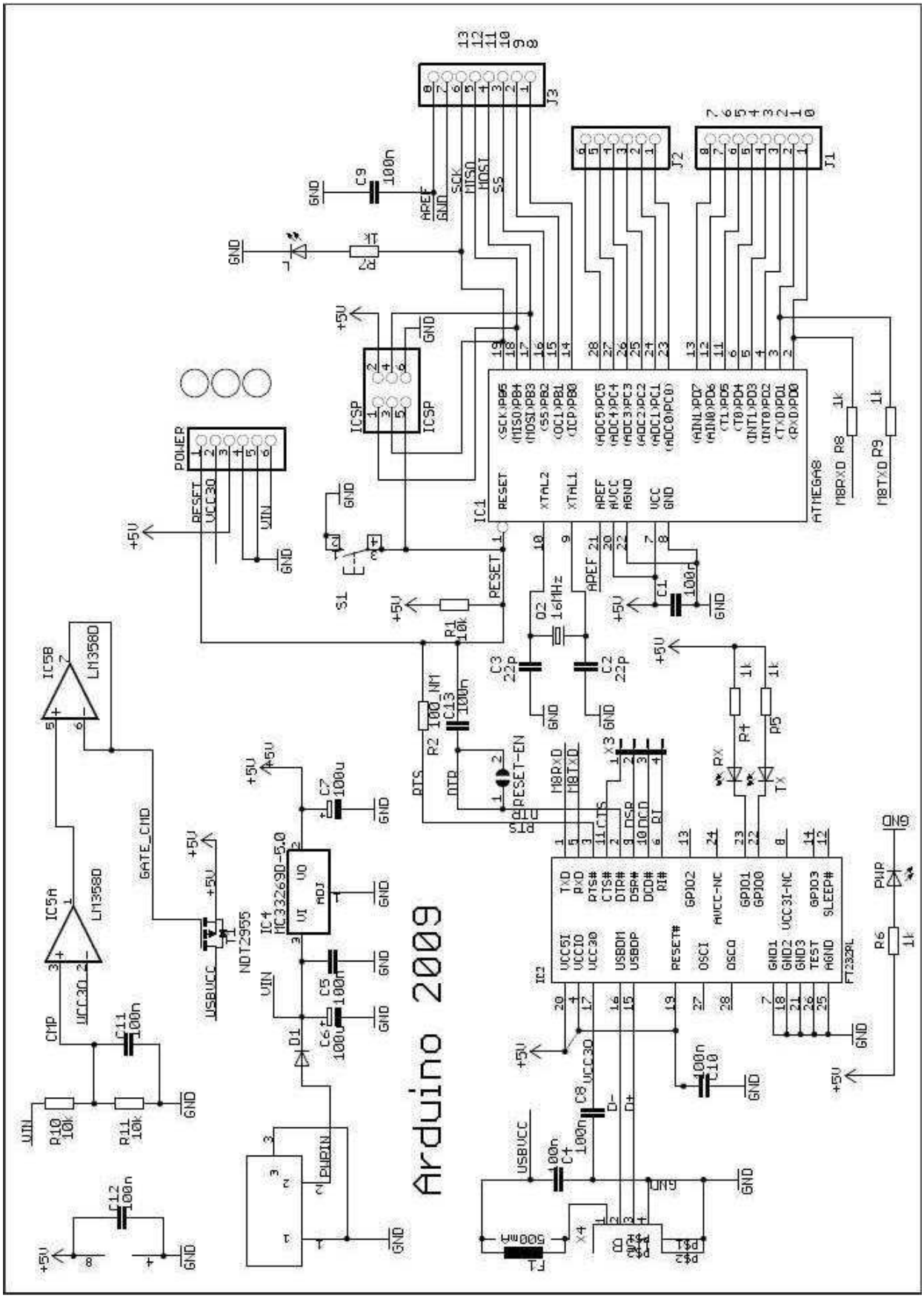
Tabela 1.2 Sygnały na złączach płytki Arduino

Kontakt	Sygnały
Logiczny 0	Złącze szeregowe RX, połączony też z kontrolerem FTDI.
Logiczny 1	Złącze szeregowe TX, połączony też z kontrolerem FTDI.
Logiczne 2, 3	Zewnętrzne przerwania jeśli zostały włączone za pomocą funkcji attachInterrupt(); reagują na poziom 0, zbocza sygnału lub zmianę wartości.
Logiczne 3, 5, 6, 9, 10, 11 (zaznaczone ~)	Wyjścia impulsów o modulowanej szerokości (PWM) – wyjścia pseudo-analogowe.
Logiczny 4	Sterowanie dostępem do modułów pamięci SD. Wymagana dodatkowa biblioteka „SD”.
Logiczne 10, 11, 12, 13	Sygnały dla złącza SPI – 10: SS, 11: MOSI, 12: MISO, 13: SCK; wymagana dodatkowa biblioteka funkcji SPI; połączone ze złączem ICSP.
Analogowe A4, A5	Sygnały dla złącza I2C – A4: SDA, A5: SCL; wymagana dodatkowa biblioteka „Wire”.
Logiczny 13	Dioda świecąca na płytce.
Analogowe A0 – A5	Wejścia analogowe, rozdzielczość 10 bitów (zakres wartości 0–1023), maksymalne napięcie wejściowe 5 V; w razie potrzeby stosować dzielniki napięć i zabezpieczenia przed przepięciami mogącymi uszkodzić procesor.
AREF	Napięcie odniesienia dla przetworników analogowo-cyfrowych; do przełączania służy funkcja analogReference().
GND	masa
5 V	Stabilizowane napięcie +5 V
3,3 V	Stabilizowane napięcie 3,3 V
Vin	Niestabilizowane napięcie zasilania z gniazda koncentrycznego.
Reset	Służy do zerowania procesora.

Uwagi:

- Zależnie od konfiguracji i użycia pomocniczych bibliotek niektóre z kontaktów mogą pełnić różne funkcje.
- Numery kontaktów używane w programach odpowiadają numerom wydrukowanym na płytce a nie numerom wyprowadzeń procesora. Wyprowadzenia procesorów Atmega8, Atmega 168 i Atmega328 są identyczne. Główną różnicę między nimi są pojemności pamięci.

Poziomy napięć na wejściach i wyjściach logicznych odpowiadają standardowi TTL. Dopuszczalna obciążalność prądowa wyjść wynosi 40 mA. Należy jednak zwrócić uwagę aby przy maksymalnym obciążeniu większej liczby wyjść nie przekroczyć dopuszczalnej mocy strat procesora. Dla uniknięcia tej sytuacji korzystne jest dla większych obciążeń zastosowanie tranzystorów wykonawczych na wyjściach logicznych a wtórników napięciowych np. na wzmacniaczach operacyjnych na wyjściach analogowych. Arduino MEGA dysponuje znacznie większą liczbą wejść i wyjść każdego rodzaju.



Rys. 1.5. Schemat ideowy Arduino Duemilanove i AVTDuino

1.2 Język programowania

Język programowania Arduino jest zbliżony do języka C++ i zawiera pewne gotowe obiekty jak np. *Serial* czy *String*. W początkowej fazie można nie korzystać z nich i wówczas staje się on praktycznie pewnym podzbiorem języka C dostosowanym do możliwości sprzętowych i uzupełnionym o funkcje i polecenia służące do wykorzystania dostępnych urządzeń peryferyjnych – w pierwszym rzędzie korzystania z różnego rodzaju wejść i wyjść: logicznych, analogowych i impulsowych (PWM) oraz ich konfiguracji. Istotnym rozszerzeniem jego możliwości są dodatkowe biblioteki funkcji obsługujące przykładowo moduły rozszerzeń albo protokóły komunikacyjne.
Wejścia analogowe Arduino mają rozdzielczość 10 bitów (odpowiada to zakresowi wartości 0 – 1023) a wyjścia z modulacją szerokości impulsu – rozdzielczość 8 bitów (zakres wartości 0 – 255).

Tabela 1.3 Elementy języka

Sterujące przebiegiem programu	if (warunek)	if (warunek)...else	for
	switch case	while (warunek)	do... while (warunek)
	break	continue	return
	goto		
Struktura programu	setup()	loop()	
Dalsze elementy składni	; (średnik – zakończenie linii)	{} (nawiasy wygięte – początek i koniec bloku lub funkcji)	// – początek komentarza jednoliniowego
	/* */ komentarz wieloliniowy	#define – definiowanie stałych lub makrorozkazów	#include – włączenie pliku
	#ifdef ... #else ... #endif – kompilacja warunkowa	#ifndef... #else... #endif – kompilacja warunkowa, warunek odwrotny	#undef – wyłączenie definicji z #define
	() – obejmują spis parametrów funkcji lub warunki w pętlach i rozgałęzieniach	[] – zawierają indeksy w dostępie do tablic	
Operatory arytmetyczne	= – przyporządkowanie wartości	+ – dodawanie	– – odejmowanie
	* – mnożenie	/ – dzielenie	% – modulo
Operatory porównania	== – równość w warunkach	!= – nierówność w warunkach	< – mniejsze
	> – większe	≤ – mniejsze lub równe	≥ – większe lub równe
Operatory logiczne	&& – i	|| – lub	! – negacja
Wskazania	* – zawartość wskazywanego adresu	& – adres zmiennej	
Operatory bitowe	& – bitowe i	| – bitowe lub	^ – bitowa alternatywa
	~ – bitowa negacja	<< – przesunięcie w lewo	>> – przesunięcie w prawo
Operatory złożone	++ – dodanie 1	-- – odjęcie 1	+= – dodawanie
	-= – odejmowanie	*= –mnożenie	/= – dzielenie
	&= – bitowe i	|= – bitowe lub	
Zdefiniowane stałe	HIGH	LOW	INPUT
	OUTPUT	true	false
Typy danych	void	boolean	char
	unsigned char	byte	int
	unsigned int	word	long
	unsigned long	float	double
	string – tabela znaków	string – obiekt	array

Zamiana typów zmiennych	char()	byte()	int()
	word()	long()	float()
Zakres ważności zmiennych – modyfikatory	static – zmienne nie dynamiczne ale lokalne w funkcji	volatile – zmienne ulotne modyfikowane w przerwaniach	const – stałe niezmienialne w programie
Funkcje pomocnicze	sizeof()		
Funkcje			
Wejścia/wyjścia logiczne	pinMode() – kierunek pracy danego we./wy.	digitalWrite() – zmiana stanu wyjścia	digitalRead() – odczyt stanu wejścia
Wejscia/wyjścia analogowe	analogReference() – napięcie odniesienia dla przetworników a/c	analogRead() – odczyt wejścia analogowego w zakresie 0 – 1023	analogWrite() – wyjście z modulowaną szerokością impulsów – PWM w zakresie 0 – 255
Dodatkowe	tone()	noTone()	shiftOut()
	pulseIn()		
Czas	millis() – liczba milisekund od momentu uruchomienia Arduino	micros() – liczba mikrosekund od momentu uruchomienia Arduino	delay() – opóźnienie w ms, przerwanie wykonywania programu
	delayMicroseconds() – opóźnienie w µs		
Matematyczne	min() – minimum	max() – maksimum	abs() – wartość bezwzględna
	constrain() – przynależność do zakresu	map() – zmiana zakresu liczbowego np. z 0 – 1023 na 0 – 255	pow() – potęgowanie
	sqrt() – pierwiastek kwadratowy		
Trygonometryczne	sin()	cos()	tan()
Liczby losowe	randomSeed()	random()	
Operacje bitowe i bajtowe	lowByte()	highByte()	bitRead()
	bitWrite()	bitSet()	bitClear()
	bit()		
Przerwania zewnętrzne	attachInterrupt()	detachInterrupt()	
Przerwania	interrupts() – włączenie przerwań	noInterrupts() – wyłączenie przerwań	ISR (adres) – podprogram przerwania
Komunikacja	serial – obiekt		

1.3. Struktura programu

Podobnie jak w języku C program składa się z funkcji, po których nazwie występują zawsze nawiasy zwykłe – () – puste lub obejmujące spis parametrów z ich typami. Funkcje dostarczające jakiejś wartości mają przed nazwą podany typ zmiennej np. char, int itd. natomiast funkcje nie dostarczające żadnej wartości mają typ void.
Kod funkcji jest zawarty w nawiasach wygiętych. Te same nawiasy służą do ograniczenia bloków poleceń w strukturach if(...)...else, pętlach for(...), while(...), do... while(...) lub w innych dowolnych miejscach.
Przykład funkcji typu integer z parametrem „a”:

```
int funkcja1(int a)
{ int b;   // zmienna lokalna – dynamiczna
  static int g; // zmienna lokalna – statyczna
  // poniżej kod funkcji
....
  return b;
```

```
}
```
Przykład funkcji nie obliczającej żadnej wartości (odpowiada procedurze w innych jezykach) i bez parametru wywołania:
```
void funkcja2()
{ int b;
  // poniżej kod funkcji
....
  b = a * c;
....
}
```
Liczba funkcji i ich nazwy są dowolne jednak program dla Arduino musi zawierać obowiązkowo dwie funkcje: setup() i loop() – mogą one być też jedynymi częściami programu. Pierwsza z nich jest wykonywana tylko raz po uruchomieniu programu i zawiera przeważnie polecenia konfigurujące i inicjalizujące wartości zmiennych. Kod zawarty w funkcji loop() jest wykonywany w pętli bez końca aż do zatrzymania programu lub wyłączenia Arduino. Funkcja ta jest wywoływana po zakończeniu funkcji setup() i pełni praktycznie rolę następującej konstrukcji w języku C:
```
main()
{while(1)
  {.....
   .....
   }
}.
```
Minimalny program wygląda więc następująco (w nawiasach wygiętych zawarty jest oczywiście kod programu):
```
unsigned int a; // Definicje zmiennych globalnych
void setup()
{boolean b; // Definicja zmiennej lokalnej
  ....
}
void loop()
{char b; // Definicja zmiennej lokalnej, nazwy zmiennych lokalnych mogą się powtarzać
          // w innych funkcjach
  ....
}
```
Każda z tych funkcji może być pusta (nie zawierać żadnego kodu) ale nie można jej opuścić.
Po nich następuje dowolna liczba funkcji dowolnego typu i dowolnymi parametrami lub bez. Możliwe jest też umieszczenie funkcji setup() i loop() na końcu programu po wszystkich innych funkcjach. Nie zmienia to kolejności jego wykonywania. W przypadku gdy funkcje napisane przez użytkownika mają typy inne niż int lub ich argumenty są innego typu konieczne może być umieszczenie na początku programu (w każdym razie przed kodem pierwszej funkcji) ich deklaracji (prototypów), przykładowo:
void funkcja3(char arg1, char arg2) – dla funkcji typu void z dwoma parametrami typu char,
char funkcja4(void) – dla funkcji typu char bez parametrów itd.
Ogólnie rzecz biorąc prototypy, o ile są zgodne z kodem funkcji nie szkodzą w żadnym wypadku i lepiej umieścić w programie więcej z nich niż za mało.
Zmienne o charakterze globalnym definiowane są przeważnie od razu na początku programu przed kodem funkcji. Zasadniczo mogą one być definiowane w dowolnych miejscach kodu na zewnątrz funkcji ale są wówczas trudniejsze do znalezienia. Ich nazwy nie mogą się powtarzać na poziomie globalnym ale dozwolone jest użycie tych samych nazw dla zmiennych lokalnych. Zamaskowane przez nie zmienne globalne są niedostępne w ramach tej funkcji.
Zmienne definiowane wewnątrz funkcji mają charakter lokalny – są widoczne tylko w ramach swojej funkcji i są zmiennymi dynamicznymi tzn. po opuszczeniu funkcji ich zawartość jest tracona a zajmowane przez nie miejsce wolne i niedostępne pod tą nazwą. Zmienne statyczne są zmiennymi lokalnymi ale po opuszczeniu funkcji zachowują swoją wartość i mogą być wykorzystane po jej ponownym wywołaniu. Nazwy zmiennych lokalnych mogą się powtarzać w innych funkcjach. Jeżeli zmienne lokalne noszą nazwy zmiennych globalnych odpowiednie zmienne globalne nie są widoczne w ramach tych

funkcji (są zamaskowane). Dla uniknięcia niejasności i spowodowanych tym omyłek lepiej jest jednak unikać takich sytuacji.
Identycznie jak w C stałe programowe lub makrorozkazy są definiowane za pomocą polecenia *#define* a polecenie *#include* służy do włączenia do kodu programu zawartości dodatkowych plików nagłówkowych .h albo dalszych modułów kodu. Pliki nagłówkowe są konieczne m.in. do korzystania z dodatkowych bibliotek dla Arduino.
Polecenia *#ifdef… #else… #endif* i *#ifndef...#else...#endif* sterują przebiegiem kompilacji w zależności od wartości odpowiedniej stałej zdefiniowanej za pomocą *#define* lub usuniętej za pomocą *#undef*. Ułatwia to kompilowanie różnych wersji programów w zależności od potrzeb. Przykładowo zdefiniowanie stałej TEST za pomocą
#define TEST
pozwala na warunkowe umieszczenie w programie poleceń diagnostycznych, które po zmianie definicji i ponownym skompilowaniu programu zostaną z niego usunięte bez konieczności dokonywania jakich kolwiek zmian w kodzie źródłowym (i równie łatwo je ponownie przywrócić w innym momencie gdy okażą się znowu potrzebne):

```
#ifdef TEST
  Serial.print(.....)
#endif
```

Warunkowa kompilacja ułatwia również przygotowywanie wersji programu przeznaczonych do współpracy z różnym sprzętem i dopasowania do niego niezbędnych parametrów (np. szybkości transmisji dla różnych TNC albo radiostacji D-STAR) albo różnych wersji językowych w oparciu o ten sam kod źródłowy. Ułatwia to późniejsze dokonywanie w nim zmian i modyfikacji, które w przeciwnym przypadku należałoby wprowadzać w wielu plikach co zwiększałoby prawdopodobieństwo omyłek i kosztowałoby dużo więcej pracy.
Podprogramy przerwań zaczynają się od zarezerwowanego słowa ISR (ang. *Interrupt Service Routine*) a jako argument wywołania podawany jest tzw. wektor przerwania czyli adres początku podprogramu zależny od wydarzenia, które go spowodowało np. *TIMER2_OVF_vect* – dla przerwania spowodowanego przepełnieniem licznika TIMER2 lub *TIMER1_COMPA_vect* dla przerwania wywołanego przez komparator A licznika TIMER1:

```
ISR(TIMER2_OVF_vect)
{
  ….       //Kod podprogramu
}
```

Konfiguracja przerwań związanych z licznikami wymaga wprowadzenia pasujących danych do rejestrów TCCRxA, TCCRxB, TCNTx, OCRxA, OCRxB, ICRx, TIMSKx, TIFRx – gdzie x oznacza numer licznika 0 (licznik 8-bitowy), 1 (licznik 16-bitowy), lub 2 (licznik 8-bitowy) dla Arduino Duemilanove UNO, AVTDuino itp. a dodatkowo 3 – 5 (liczniki 16-bitowe) dla Arduino MEGA. Źródłem przerwań mogą być także sygnały z zewnątrz doprowadzone do wejść logicznych 2 lub 3 o ile przerwania te zostały włączone za pomocą funkcji *attachInterrupt()*. Do ogólnego włączenia przerwań służy funkcja *interrupts()* (patrz tabela 1.3).
Niestety niemożliwe jest wykonywanie programu krok po kroku jak w innych systemach i środowiskach programistycznych i dlatego identycznie jak w przypadku dawniejszej praktyki programowania w BASIC-u konieczne jest umieszczanie w strategicznych lub badanych miejscach programu rozkazów „wydruku“ a praktycznie wyświetlania na ekrenie odpowiednich informacji. Są właśnie pokazane powyżej polecenia *Serial.print()* a właściwie wywołania metody *print* obiektu *serial*. Jako argument w nawiasach podawane są nazwy obserwowanych zmiennych, teksty itp. Spokrewniona z nią metoda *Serial.println()* dodaje na końcu wyświetlanych danych znak nowej linii i często okazuje się praktyczniejsza od poprzedniej.
Korzystanie z metody *Serial.print()* lub *Serial.println()* wymaga uprzedniego zainicjalizowania złącza w funkcji setup() za pomocą metody *Serial.begin(9600)* – w nawiasie podana jest szybkość transmisji, w tym przykładzie 9600 bit/s. Może być ona dowolna w ramach możliwości komputera PC i musi być identyczna z wybraną w środowisku programistycznym w oknie monitora szeregowego. Dane transmitowane są przez złącze USB dlatego też konieczne jest zainstalowanie na PC odpowiedniego sterownika USB/FTDI dla Arduino, symulującego złącze szeregowe COMx dodatkowo do środowiska programistycznego. Standardowe sterowniki Windows przeważnie nie współpracują prawidłowo z Arduino.

W internecie pojawił się wprawdzie symulator Arduino (*http://www.arduino.com.au/Simulator-for-Arduino.html; http://www.buildinginternetofthings.com/2012/03/25/arduino-simulators/*) pozwalający na wykonywanie krokowo programów Arduino na PC ale autor nie wypróbował jeszcze jego pracy i nie może wypowiedzieć się na temat jego jakości.
W wielu zastosowaniach programy pracują zresztą w czasie rzeczywistym i ich krokowe wykonywanie a nawet tylko wyświetlanie informacji diagnostycznych spowalniają ich pracę i wobec tego pozwalają jedynie na wstępne sprawdzenie przebiegu wykonywania programu. Jedną z możliwości diagnozy w czasie rzeczywistym jest zaprogramowanie zmian stanu któregoś z nieużywanych wyjść logicznych i jego obserwacja na osyloskopie dodatkowo do obserwacji sygnałów użytkowych. Również i te kroki diagnostyczne mogą być skompilowane warunkowo i usunięte z ostatecznej wersji programu jak to pokazano powyżej.

1.4 Dodatkowe biblioteki

Tabela 1.4 zawiera zestawienie najważniejszych bibliotek mogących znaleźć zastosowanie w programach krótkofalarskich. Liczba dotąd opracowanych bibliotek jest znaczna a poza tym bardziej doświadczeni programiści mogą bez trudności opracowywać dowolne biblioteki przydatne dla ich celów (szczegóły podano w dodatkach) – dlatego też trudno byłoby wymieniać tutaj choćby tylko najważniejsze biblioteki z różnych odległych od krótkofalarstwa dziedzin. Tabela 1.5 zawiera natomiast zestawienie klas i ich metod dla niektórych z nich. I znowu ze względu na obszerność materiału autor wybrał bardziej złożone i przez to trudniejsze w użyciu biblioteki występujące w przedstawionych dalej programach lub takie najczęściej używane jak *Serial*.
Menu „*Sketch*" | „*Impor*t *Library*" („Programy" | „Importuj bibliotekę") pozwala na dodanie do środowiska programistycznego dowolnych bibliotek i korzystanie z nich jak z bibliotek standardowych. Dla każdej z nich konieczne jest włączenie do programu związanego z nią pliku nagłówkowego .h za pomocą polecenia #include. W przypadkach szczególnych korzystanie z jednej z bibliotek wiąże się z użyciem innych i wtedy niezbędne jest włączenie większej liczby nagłówków. Przykładowo moduły dostępu do internetu komunikują się z Arduino przez złącze (magistralę) SPI i wymagają użycia biblioteki SPI oprócz odpowiednich bibliotek dostępowych. Przykładowe programy z wykorzystaniem bibliotek sieciowych przytoczono w dodatkach.
Część z dostępnych bibliotek ma zastosowanie uniwersalne ale niektóre z nich są przeznaczone wyłącznie dla wybranych typów płytek Arduino lub też istnieją oddzielne wersje dla różnych typów płytek. Jest to związane z ich wyposażeniem układowym. Niktóre biblioteki są też związane z konkretnymi modułami rozszerzeń.

Tabela 1.4 Niektóre dodatkowe biblioteki dla Arduino

Biblioteka	Zastosowanie
SoftwareSerial	Arduino UNO, Duemilanove, Diecimila. Komunikacja szeregowa przez dowolne wyprowadzenia dodatkowo do standardowego złącza szeregowego.
NewSoftSerial	Unowocześniona wersja biblioteki SoftwareSerial.
Serial	Standardowa biblioteka dla złącza szeregowego zawarta w środowisku programistycznym. W Arduino MEGA obsługuje 4 złącza, w pozostałych jedno.
FFT	Służy do analizy sygnałów m.cz. przy użyciu szybkiej transformaty Fouriera.
Tone	Generacja fali prostokątnej małej czestotliwości na dowolnym wyjściu Arduino.
Ethernet	Arduino UNO, Duemilanove, MEGA. Kablowy dostęp do internetu.
GSM	Zapewnia dostęp do sieci telefonii komórkowej GSM przy użyciu modułu GSM.
WiFi	Arduino UNO, Duemilanove, MEGA. Radiowy dostęp do internetu.
Webduino	Serwer internetowy HTTP przy użyciu modułu ethernetowego.
SPI	Arduino UNO, Duemilanove, MEGA. Komunikacja przez złącze SPI

SD	Dostęp do modułów pamięci SD, zarządzanie pamięcią i plikami.
Wire	Komunikacja przez magistralę I2C. Zawarta w środowisku programistycznym.
EEPROM	Umożliwia korzystanie z pamięci EEPROM
LiquidCrystal	Do obsługi wyświetlaczy ciekłokrystalicznych. Zawarta w środowisku programistycznym.
OneWire	Dla jednoprzewodowej magistrali „1-Wire”.
Xbee	Do komunikacji z urządzeniami Xbee w trybie API.
Servo	Do obsługi silników serwo.
Stepper	Do obsługi silników krokowych.

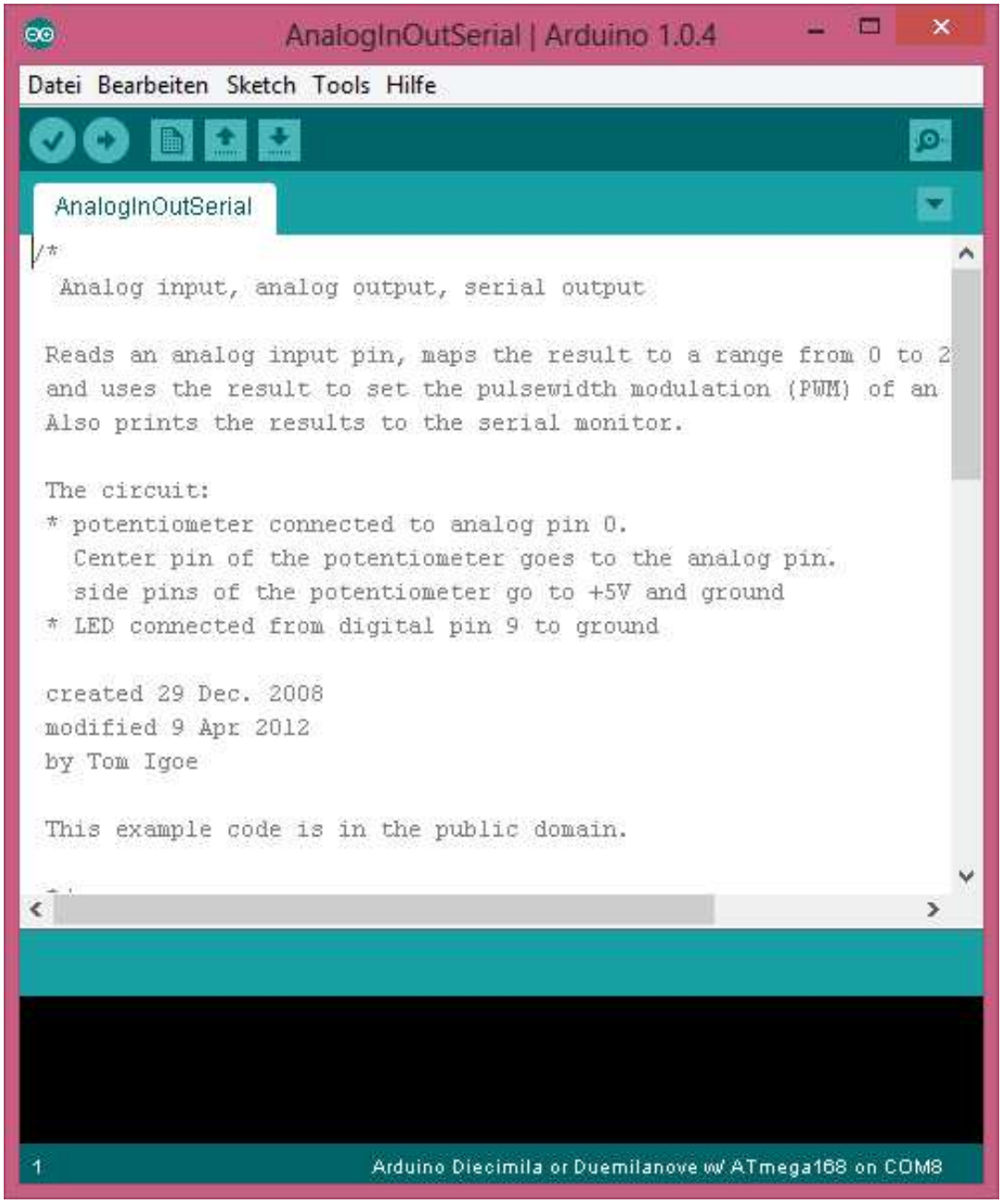
Tabela 1.5 Klasy i metody wybranych bibliotek Arduino

Biblioteka Ethernet (#include <Ethernet.h>, #include <SPI.h>)		
Klasy	**Funkcje (metody klas)**	
Ethernet –inicjalizacja biblioteki i sieci	begin()	localIP()
	maintain()	
IPAddress	IPAddress()	
Server – serwer dla klientów sieci	EthernetServer()	begin()
	available()	write()
	print()	println()
Client – klienci w sieci	EthernetClient()	if (EthernetClient)
	connected()	connect()
	write()	print()
	println()	available()
	read()	flush()
	stop()	
EthernetUDP – komunikacja w protokóle UDP	begin()	read()
	write()	beginPacket()
	endPacket()	parsePacket()
	available()	remoteIP()
	remotePort()	
Biblioteka WiFi (#include <WiFi.h>, #include <SPI.h>)		
Klasy	**Funkcje (metody klas)**	
WiFi –inicjalizacja biblioteki i sieci	begin()	disconnect()
	config()	setDNS()
	SSID()	BSSID()
	RSSI()	encryptionType()
	scanNetworks()	getSocket()
	macAddress()	
IPAddress	localIP()	subnetMask()
	gatewayIP()	
Server – serwer dla klientów sieci	Server()	WiFiServer()
	begin()	available()
	write()	print()
	println()	
Client – klienci w sieci	Client()	WiFiClient()
	connected()	connect()
	write()	print()
	println()	available()
	read()	flush()
	stop()	

UDP – komunikacja w protokóle UDP	WiFiUDP()	begin()
	available()	beginPacket()
	endPacket()	write()
	parsePacket()	peek()
	read()	flush()
	stop()	remoteIP()
	remotePort()	
Biblioteka SPI (#include <SPI.h>)		
Klasy	**Funkcje (metody klas)**	
SPI	begin()	end()
	setBitOrder()	setClockDivider()
	setdataMode()	transfer()
Biblioteka Wire (#include <Wire.h>)		
Klasy	**Funkcje (metody klas)**	
Wire	begin()	requestFrom()
	beginTransmission()	endTransmission()
	write()	available()
	read()	onReceive()
	onRequest()	
Biblioteka Serial		
Klasy	**Funkcje (metody klas)**	
Serial	available()	begin()
	end()	find()
	findUntil()	flush()
	parseFload()	parseInt()
	peek(0	print()
	println()	read()
	readBytes()	readBytesUntil()
	setTimeaout()	write()
	serialEvent()	
Biblioteka SD (#include <SD.h>)		
Klasy	**Funkcje (metody klas)**	
SD – zarządzanie pamięcią	begin()	exists()
	mkdir()	open()
	remove()	rmdir()
File – zarządzanie plikami	available()	close()
	flush()	peek()
	position()	print()
	println()	seek()
	size()	read()
	write()	isDirectory()
	openNextFile()	rewindDirectory()
Biblioteka EEPROM (#include <EEPROM.h>)		
Klasy	**Funkcje (metody klas)**	
EEPROM	clear()	read()
	write()	
Biblioteka LiquidCrystal (#include <LiquidCrystal.h>)		
Klasy	**Funkcje (metody klas)**	
lcd	begin()	blink()
	noBlink()	print()

1.5 Środowisko programistyczne

Środowisko programistyczne Arduino jest wieloplatformową aplikacją napisaną w języku Java wydzieloną z IDE przygotowanego dla języka Processing i projektu Wiring. Środowisko jest zaprojektowane w taki sposób, aby było przyjazne dla hobbystów i osób niezajmujących się tworzeniem oprogramowania. Środowisko zawiera edytor kodu z takimi funkcjami jak podświetlanie składni czy automatyczne wcięcia w kodzie, oraz pozwala na kompilację i ładowanie programu do procesora Arduino przez złącze USB bez konieczności użycia programatora. Zazwyczaj nie ma potrzeby dodatkowej edycji plików „Makefile" lub uruchamiania programów z linii poleceń.
Standardowo środowisko Arduino zawiera bibliotekę C/C++ o nazwie "Wiring" (z projektu o tej samej nazwie), dzięki czemu wykonywanie podstawowych operacji wejścia / wyjścia staje się znacznie łatwiejsze. Programy dla Arduino są napisane głównie w języku podobnym C/C++.
Obecnie pod adresem *www.arduino.cc* dostępne są wersje dla Windows (w postaci pliku instalacyjnego lub archiwum zip), Mac OS X i Lunuksa oraz kod źródłowy dla specjalistów.



Rys. 1.6. Okno główne środowiska Arduino

Okno główne zawiera u góry następujące menu (kolejno od lewej do prawej):

- „Plik" („*File*" lub „*Datei*" w zależności od języka). Środowisko Arduino pozwala na wybór jednego z ponad 30 języków obsługi. Domyślnie w trakcie instalacji wybierany jest język używany przez system operacyjny.
  - Nowy plik – programu
  - Otwórz plik – programu
  - Katalog programów (w kolejnym menu wyświetlany jest spis programów zawartych w katalogu)
  - Przykłady (w kolejnym menu zawarty jest tematyczny spis dostępnych przykładowych programów, a w dalszych menu – programów odpowiadających wybranemu tematowi).
  - Zamknij
  - Zapisz – program w pliku
  - Zapisz pod... – inną nazwą
  - Załaduj (do Arduino) – początek procesu ładowania jest sygnalizowany miganiem znajdującej się na płytce diody świecącej połączonej z wyjściem 13. Sam przebieg sygnalizują diody Rx i Tx połączone z przewodami logicznymi 0 i 1. Brak sygnalizacji oznacza wystąpienie błędów w komunikacji PC z Arduino np. wskutek nieprawidłowego wyboru typu płytki albo złącza COM albo też niewłaściwych sterowników USB (konieczne jest zainstalowanie sterowników Arduino zastępujących standardowe sterowniki Windows).
  - Załaduj za pomocą programatora – używany programator należy wybrać uprzednio w menu „Narzędzia" („*Tools*") i połączyć z komputerem i płytką Arduino.
  - Format papieru
  - Wydruk
  - Ustawienia – otwarcie okna ustawień programu.
  - Zakończ

- „Edycja" („*Edition*" lub „*Bearbeiten*")
  - Cofnij
  - Powtórz
  - Wytnij
  - Skopiuj
  - Skopiuj dla forum – kopiowanie programu do schowka Windows, sformatowanego i podkolorowanego tak, aby można go wygodnie zamieścić na forum dyskusyjnym.
  - Skopiuj jako HTML – skopiowanie kodu do schowka w formacie HTML aby można go wygodnie umieścić na witrynie internetowej.
  - Wklej
  - Wybierz wszystko
  - Komentuj (w kodzie)
  - Wetnij (kod)
  - Cofnij wcięcie
  - Szukaj
  - Znajdź następny
  - Znajdź poprzedni
  - Zastosuj wybór do poszukiwania

- „Programy" („*Sketch*")
  - Sprawdź / kompiluj
  - Pokaż katalog programów
  - Dodaj plik – dodanie do programu zawartości wybranego pliku, dalszych cześci kodu.
  - Importuj bibliotekę (w kolejnym menu wyświetlany jest spis bibliotek możliwych do zaimportowania). Biblioteki te są zawarte w katalogu „*libraries*" znajdującym się w katalogu instalacyjnym środowiska Arduino. Tam też należy dodawać nowe bilioteki pobrane z Internetu lub własnego autorstwa przed zaimportowaniem ich i udostępnieniem w ten sposób do użycia w programach. Import powoduje skompilowanie kodów źródłowych biblio-

tek. Począwszy od wersji 1.0.5 biblioteki mogą być importowane także z archiwów zip. Najczęściej potrzebne biblioteki są standardowo zawarte w środowisku Arduino i nie wymagają ponownego zaimportowania.

- „Narzędzia" („*Tools*")
  - Automatyczne formatowanie kodu
  - Archiwizacja programu – w postaci skomprymowanego archiwum zip.
  - Naprawa i ponowne załadowanie kodu
  - Okno monitora szeregowego (otwierane jest okno mogące służyć do celów diagnostycznych). Okno otwierane jest tylko w czasie wykonywania programu przez Arduino i zamykane automatycznie w czasie ładowania nowego programu. W oknie tym wyświetlane są informacje wydawane przez program za pomocą poleceń *Serial.print()* lub *Serial.println()*. W oknie należy z rozwijanej listy wybrać szybkość transmisji zgodną z podaną w programie za pomocą polecenia *Serial.begin()*. Ogólnie rzecz biorąc szybkość ta może być wybrana ze spisu dowolnie i dowolnie podana w zgodzie z tym w programie ale w niektórych przypadkach, jeśli np. złącze szeregowe lub USB ma być wykorzystywane docelowo do komunikacji z innymi urządzeniami np. radiostacjami, TNC, PTC itd. Korzystnie jest wybrać od razu szybkość docelową. Polecenia diagnostyczne mogą być kompilowane warunkowo jak to pokazano wcześniej. W wersjach dla Mac OS X i Linuksa otwarcie okna monitora powoduje wyzerowanie procesora i rozpoczęcie wykonywania programu od początku.
  - Płytka – służy do wyboru modelu Arduino zgodnie z używanym. W kolejnym menu wyświetlany jest spis obsługiwanych modeli. Dla niektórych z nich dostępne są warianty zawierające różne typy procesorów. Należy wtedy zwrócić uwagę również na typ mikroprocesora.
  - Złącze szeregowe. W kolejnym menu wyświetlany jest spis dostepnych złączy szeregowych COM zawierający złącza wirtualne udostępniane przez sterownik Arduino.
  - Programator. W kolejnym menu wyświetlany jest spis obsługiwanych programatorów dla złącza ICSP. Ich użycie może być konieczne w przypadku ładowania programów skompilowanych przez inny kompilator albo w przypadku usunięcia z Arduino programu ładującego (ang. *bootloader*). Programy skompilowane w środowisku Arduinio wygodniej jest ładować przez złącze USB – widziane przez środowisko jako wirtualne złącze COM. W przypadku nieprawidłowego wyboru płytki Arduino w menu, nieprawidłowego wyboru złącza albo niezainstalowania sterowników Arduino w dolnym polu okna głównego na zakończenie próby załadowania programu użytkowego lub programu ładującego wyświetlają się meldunki błędów. Tam też wyświetlane są meldunki o błędach w programie wykrytych przez kompilator.
  - Ostani punkt służy do załadowania do Arduino programu ładującego. Procesor znajdujący się na płytce Arduino zawiera go oczywiście, a więc punkt ten jest istotny tylko w przypadku wymiany procesora na nowy.

- „Pomoc" („*Hilfe*" lub „*Help*") – wywołanie stron HTML z tekstami pomocy. Strony znajdują się w katalogu pomocy zawartym w katalogu instalacyjnym środowiska Arduino.
  - Pierwsze kroki
  - Środowisko
  - Szukanie błędów
  - Encyklopedia języka programowania
  - Poszukiwanie w encyklopedii
  - Podstawowe pytania
  - Odwiedziny witryny *www.arduino.cc*
  - Informacja o programie.

W menu obok nazw funkcji podane są ewentualne kombinacje klawiszy służące do ich szybszego wywołania.

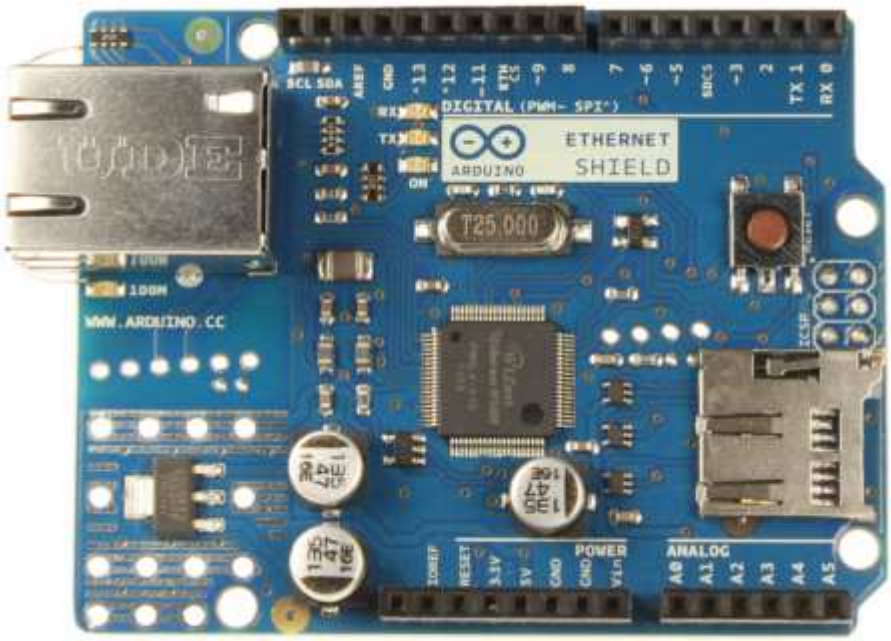
Pod linią menu znajduje się szereg symboli służących (licząc od lewej do prawej) do:

- Próbnej kompilacji programu (sprawdzenia jego formalnej poprawności) –
- Załadowania programu do Arduino przez złącze USB/COM –
- Otwarcia pustego okna edytora dla nowego programu –
- Otwarcia istniejącego pliku zawierającego program – . W dotakowym menu wyświetlany jest spis przykładów i już istniejących programów do wyboru.
- Zapisu programu zawartego w oknie edytora w pliku – . Otwierane jest okno wyboru katalogu do zapisu i nazwy pliku.
- Otwarcia okna diagnostycznego (monitora złącza szeregowego) – .
  Po najechaniu myszą na powyższe symbole wyświetlana jest obok nich informacja o ich funkcji.

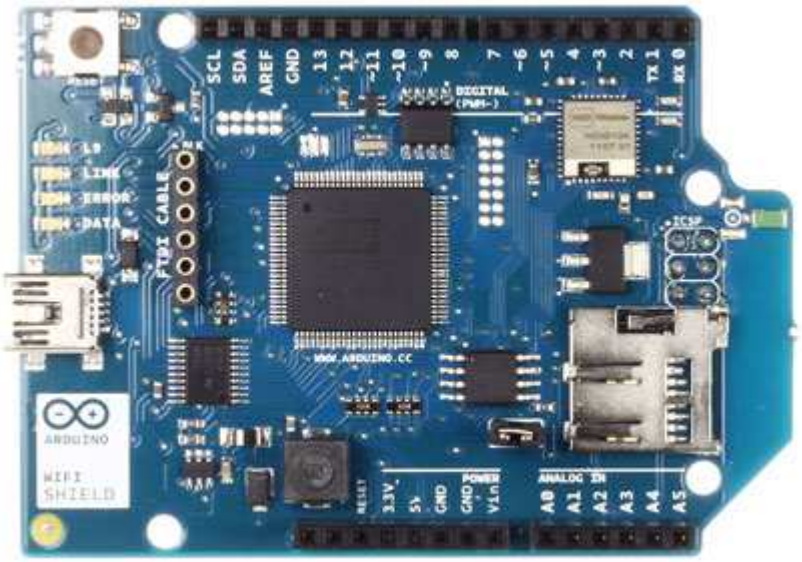
Pod nimi znajduje sie okno edytora z nazwą zawartego w nim programu a poniżej pole, w którym wyświetlane są informacje robocze i meldunki błędów. Edytor pozwala na pisanie programów w języku Arduino (pliki bez wyświetlonego rozszerzenia), C (pliki z rozszerzeniem .c), C++ (pliki z rozszerzeniem .cpp) oraz plików nagłówkowych (z rozszerzeniem .h).
W linii informacyjnej u samego dołu okna wyświetlany jest wybrany w menu model płytki i złącze COM.

1.6 Dodatkowe moduły

Moduły rozszerzeń są wtykane bezpośrednio na płytkę Arduino i wykorzystują do komunikacji z procesorem sygnały występujące na listwach kontaktowych na krawędziach płytek bądź złącze ICSP (np. dla magistrali SPI). Pozostałe przewody (kontakty na listwach bocznych) mogą być dowolnie wykorzystywane do innych celów. O ile nie dochodzi do konfliktów wykorzystania sygnałów przez różne moduły możliwe jest tworzenie konstrukcji zawierających więcej modułów rozszerzeń – np. moduł sieciowy i wyświetlacz ciekłokrystaliczny. Należy jednak zwrócić wówczas uwagę czy sumaryczny pobór prądu – łącznie z prądem dostarczanym przez wyjścia płytek do układów peryferyjnych, zasilanem modułów pamięci SD itp. – nie przekracza możliwości złącza USB i w razie potrzeby zasilać urządzenie z zewnętrznego zasilacza.



Rys. 1.7. Moduł Ethernetu W5100

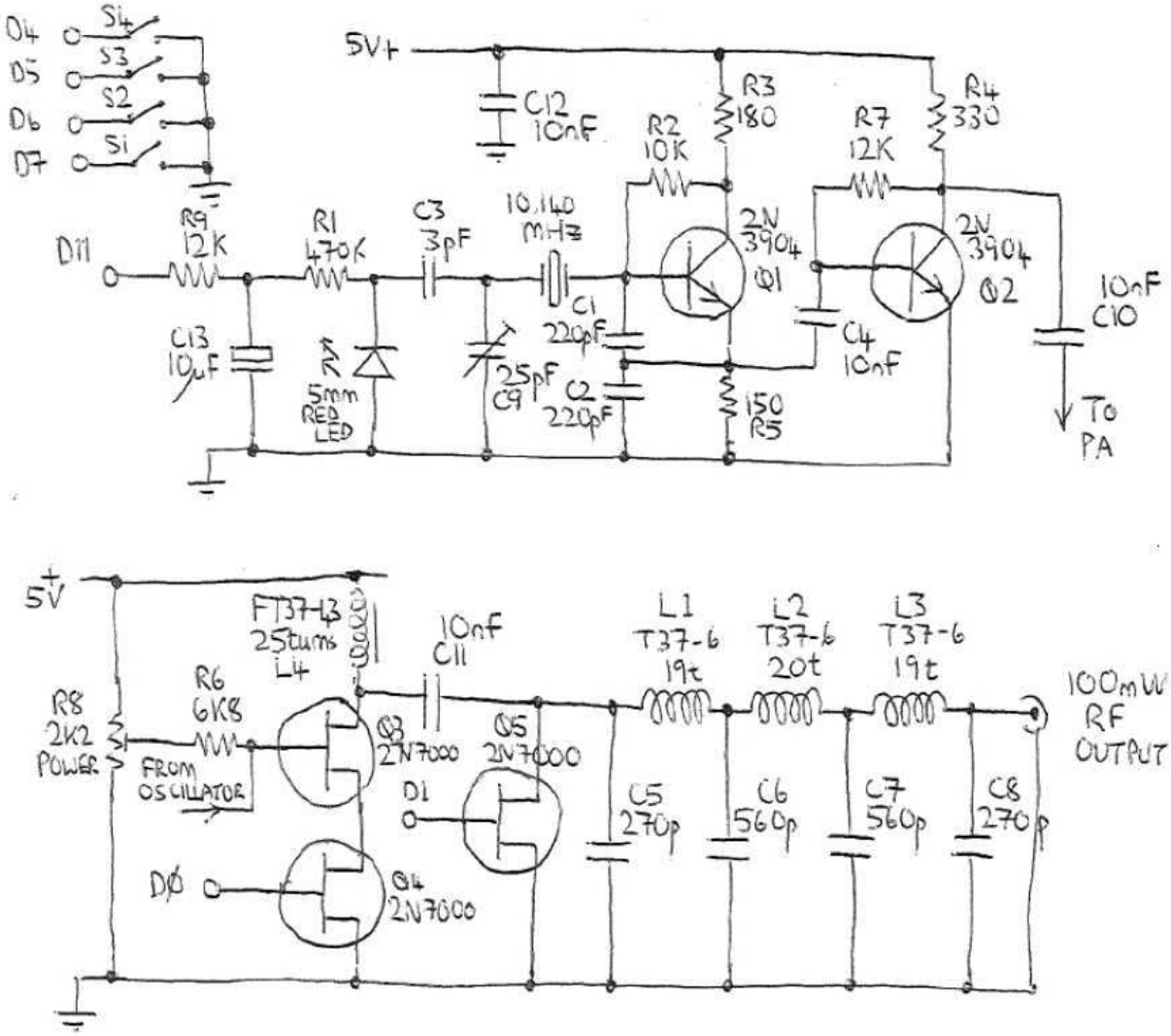


Rys. 1.8. Moduł WiFi

Moduły rozszerzeń stanowią praktyczne uzupełnienie systemu Arduino i pozwalają na wykorzystanie mikrokomputerów do różnych nawet stosunkowo skomplikowanych zastosowań. Do modułów przydatnych krótkofalowcom należą w pierwszym rzędzie moduł prototypowy zawierający płytkę dziurkowaną do konstrukcji dowolnych własnych układów, moduły Ethernetu i WiFi do połączenia z internetem bezprzewodowo lub za pośrednictwem kabla, moduł sterujący do silników, moduł telefonii GSM, moduły radiowe zgodne ze standardem Xbee, moduły Blutooth, wyświetlacze ciekłokrystaliczne oraz moduły konstruowane specjalnie dla, a częściowo również i przez krótkofalowców. Należą do nich moduł radiowy Argent Radio będący w rzeczywistości modemem TNC do packet-radio i APRS lub mininadajnik radiolatarni amatorskiej dla QRSS, FSCW i DFCW na pasma 80, 40 lub 30 m konstrukcji Hansa Summersa (*www.hanssummers.com*).
Moduły Ethernetu i WiFi są pod względem funkcjonalności bardzo podobne do siebie – z uwzględnieniem specyfiki dostępu do sieci – i również ich biblioteki zawierają prawie identyczne funkcje (o odpowiednio różnych nazwach), dlatego też przystosownie podanych w skrypcie przykładów do wybranego rodzaju dostępu do internetu nie powinno przysporzyć większych trudności.
Moduł Ethernetu W5100 zawiera własny procesor obsługujący rodzinę protokółów TCP/IP (ang. *stack*), pozwala na korzystanie przez Arduino z modułów pamięci microSD i komunikuje sie z Arduino za pośrednictwem złącza SPI. Korzystanie z niego wymaga więc użycia bibliotek SPI i „Ethernet". Umożliwia on nawiązywanie do 4 równoległych połączeń TCP i UDP.
Moduł WiFi obsługuje protokóły 802.11b/g z kodowaniem WEP, WPA albo „WPA2 Personal" lub niekodowane. Również i on posiada kieszeń dla modułów pamięciowych microSD i komunikuje się z Arduino za pośrednictwem złącza SPI. Dodatkowo posiada on własne złącze mikroUSB do aktualizacji oprogramowania a do celów diagnostycznych złącze FTDI. Moduł współpracuje z Arduino Duemilanove (a więc i AVTDuino), UNO i MEGA i posiada własną antenę nadawczo-odbiorczą.
Moduł Xbee, zgodny z normą Zigbee/802.15.4 pracuje w paśmie przemysłowym 2,4 GHz z mocą 1 mW (0 dBm). Czułość odbiornika wynosi -92 dBm a podawany przez producenta zasięg do 30 m w pomieszczeniach i do 100 m na zewnątrz. Szybkość transmisji w kanale radiowym wynosi 250 kb/s a na złączu szeregowym 1200 – 115200 b/s.

Radiolatarnia QRSS

Skonstruowany przez Hansa Summersa G0UPL (*www.hanssummers.com*) moduł nadawczy dla Arduino zawiera nadajnik małej mocy – 100 do 150 mW zależnie od pasma – na jedno z pasm amatorskich 80, 40 lub 30 m. Nadajnik jest kluczowany amplitudowo przez sygnał logiczny z wyjścia D0 i tranzystor Q4 lub częstotliwościowo przez sygnał analogowy z wyjścia D11. Zamiast diody waraktorowej użyto tutaj czerwonej diody świecącej o srednicy 5 mm spolaryzowanej w kierunku zaporowym (temat ten jest omawiany obszerniej w tomie poświęconym radiolatarniom małej mocy. Tranzystor Q5 (występujący w programie pod nazwą tłumika – ATT) sterowany sygnałem z wyjścia D1 Arduino zwiera wyjście stopnia mocy nadajnika zapobiegając przenikaniu sygnału w.cz. przez pojemności tranzystora nadawczego Q3 w przerwach nadawania. Sygnał sterujący tranzystor Q5 ma fazę (wartość logiczną) odwrotną do sygnału kluczującego Q4.
Dławik w obwodzie drenu wzmacniacza mocy zawiera 25 zwojów przewodu nawiniętych na rdzeniu pierścieniowym FT37-43. Cewki filtru dolnoprzepustowego są nawinięte na rdzeniach proszkowych T37-6 i mają odpowiednio 19, 20 i 19 zwojów. Widoczny na schemacie nadajnik pracuje na częstotliwości 10140 kHz. Przełączniki S1–S4 służą do wyboru rodzaju emisji i szybkości transmisji w programie. Rozwiązania nadajników na pozostałe pasma opisane są w witrynie konstruktora.
Zamieszczony poniżej program służy do nadawania za pomocą opisanego nadajnika sygnałów emisji QRSS (wolnej telegrafii z kluczowaniem amplitudy), FSCW (wolnej telegrafii z kluczowaniem częstotliwości) i DFCW (wolnej telegrafii, w której częstotliwość niższa odpowiada kropkom a wyższa – kreskom alfabetu Morse'a). Emisje QRSS są opisane szczegółowo w tomach poświęconych technice słabych sygnałów a podobne rozwiązania nadajników w tomie poświęconym radiolatarniom małej mocy.



Rys. 2.1. Schemat modułu nadawczego G0UPL

Dla otrzymania w programie wymaganych przez rytm nadawania odstępów czasu zastosowano standardowy licznik milisekund – funkcję millis(). Na początku okresu czasu jego stan jest zapisywany w odrębnej zmiennej a następnie jego stan bieżący jest porównywany z tą zmienną. Pomiar czasu dokonywany jest przez obliczenie różnicy ich obu Następnie wystarczy tylko porównać różnicę z zadaną wartością a po jej przekroczeniu przejść do następnej akcji programu. Dzięki pomiarowi czasu unika się korzystania z funkcji opóźnienia delay(), w czasie trwania której mikroprocesor jest zablokowany i nie może wykonywać żadnych innych zadań. W praktyce stosowanie funkcji delay() ogranicza się więc do najprostszych programów o charakterze ćwiczebnym lub pomocniczym.
Dla krótszych lub bardziej precyzyjnie określonych odstępów czasów Arduino dysponuje też licznikiem mikrosekund. Z jego użyciem zapoznany się w programach nadawczych dla emisji Hella.
Znaki alfabetu Morse'a są kodowane poprzez wartości odpowiednich bitów w zmiennej typu char (o długości bajtu). Zawartość każdego z bajtów definicji alfabetu jest odczytywana bitowo w kolejności od bitu najstarszego do najmłodszego. Bity najstarsze stanowiące wypełniacze mają wartość 1 i są ignorowane, pierwszy bit o wartości 0 sygnalizuje początek znaku – podobnie jak bit startowy w kodzie RTTY – a następne bity o wartościach 0 odpowiadają kropkom a o wartościach 1 – kreskom w alfabecie Morse'a. Kodowanie tego typu jest konieczne z powodu zmiennej długości znaku. Oczywiście bity wypełniajace mogą mieć wartość 0 a bit startowy – 1, ale nie zmienia to samej zasady kodowania. Również odwrotna kolejność odczytu – od bitu najmłodszego do najstarszego – pasuje do opisanej zasady i bywa stosowana w wielu programach.

Kod źródłowy radiolatarni QRSS/FSCW/DFCW

```
//
// Uklad kluczujacy QRSS/FSKCW/DFCW
// Hans Summers G0UPL, 2012
//

const char msg[] = "ZNAK ";    // Znak wywolawczy duzymi literami. Na koncu odstep!
const unsigned int speeds[] = {1, 30, 60, 100};   // Szybkosci: 12 slow/min, QRSS3, QRSS6, QRSS10

#define KEY 0                    // Kluczowanie tranzystora Q4 na nozce 0
#define ATT 1          // Tlumik, sterowanie tranzystora Q5, na nozce 1 (odwrocony sygnal kluczujacy)
#define LED 13                   // Standardowa sygnalizacja dioda swiecaca na nozce 13
#define FSK 11                   // FSK generowane przez PWM (analogowo) na nozce 11

#define MODE_NONE 0              // Tryb NONE – wylaczenie PA
#define MODE_QRSS 1              // Tryb QRSS (zwykla CW)
#define MODE_FSKCW 2             // Tryb FSK CW
#define MODE_DFCW 3              // Tryb DFCW

#define FSK_HIGH 160             // Wysoki poziom analogowy dla FSK
#define FSK_LOW 100              // Niski poziom analogowy dla FSK

#define SWITCH1 7                // Wejscia logiczne dla
#define SWITCH2 6                // przelacznika trybow pracy
#define SWITCH3 5
#define SWITCH4 4

//
// Standardowa funkcja inicjalizujaca setup()
void setup()
{
  pinMode(KEY, OUTPUT);          // Sygnaly KEY, ATT, LED, FSK - wyjscia
  pinMode(ATT, OUTPUT);
```

```
  pinMode(LED, OUTPUT);
  pinMode(FSK, OUTPUT);

  pinMode(SWITCH1, INPUT);        // Wejscia dla przelacznika trybow pracy
  pinMode(SWITCH2, INPUT);
  pinMode(SWITCH3, INPUT);
  pinMode(SWITCH4, INPUT);

  digitalWrite(SWITCH1, true);         // Wlaczenie opornikow podtrzymujacych na wejsciach
  digitalWrite(SWITCH2, true);         // przelacznika
  digitalWrite(SWITCH3, true);
  digitalWrite(SWITCH4, true);
}

//
// Standardowa funkcja loop()
void loop()
{
  static unsigned long milliPrev;       // zmienna statyczna zawiera poczatkowy stan licznika milisekund
  unsigned long milliNow;

  milliNow = millis();                  // odczyt biezacego stanu licznika milisekund

  if (milliNow != milliPrev)            // po uplywie milisekundy wywolanie funkcji beacon()
  {
    milliPrev = milliNow;
    beacon();
  }
}

//
// funkcja dostarcza kodu CW dla znaku podanego w argumencie
byte charCode(char c)
{
   switch (c)
   {
     case 'A':  return B11111001; break;    // A  .-
     case 'B':  return B11101000; break;    // B  -...
     case 'C':  return B11101010; break;    // C  -.-.
     case 'D':  return B11110100; break;    // D  -..
     case 'E':  return B11111100; break;    // E  .
     case 'F':  return B11100010; break;    // F  ..-.
     case 'G':  return B11110110; break;    // G  --.
     case 'H':  return B11100000; break;    // H  ....
     case 'I':  return B11111000; break;    // I  ..
     case 'J':  return B11100111; break;    // J  .---
     case 'K':  return B11110101; break;    // K  -.-
     case 'L':  return B11100100; break;    // L  .-..
     case 'M':  return B11111011; break;    // M  --
     case 'N':  return B11111010; break;    // N  -.
     case 'O':  return B11110111; break;    // O  ---
     case 'P':  return B11100110; break;    // P  .--.
     case 'Q':  return B11101101; break;    // Q  --.-
     case 'R':  return B11110010; break;    // R  .-.
     case 'S':  return B11110000; break;    // S  ...
```

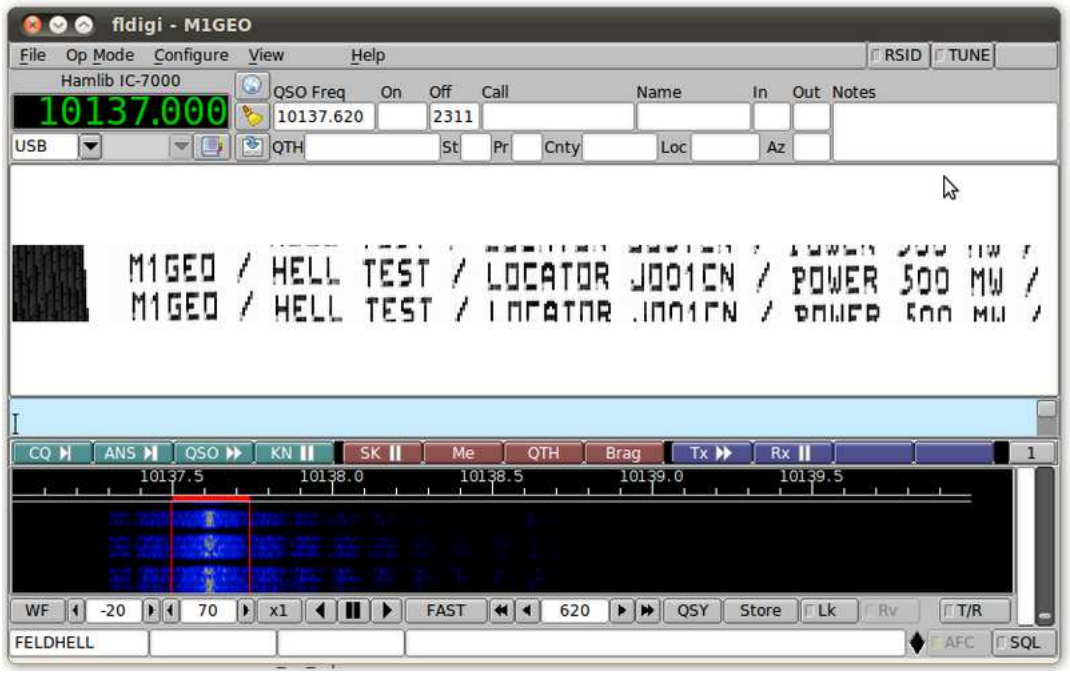
```
    case 'T':  return B11111101; break;    // T  -
    case 'U':  return B11110001; break;    // U  ..-
    case 'V':  return B11100001; break;    // V  ...-
    case 'W':  return B11110011; break;    // W  .--
    case 'X':  return B11101001; break;    // X  -..-
    case 'Y':  return B11101011; break;    // Y  -.--
    case 'Z':  return B11101100; break;    // Z  --..
    case '0':  return B11011111; break;    // 0  -----
    case '1':  return B11001111; break;    // 1  .----
    case '2':  return B11000111; break;    // 2  ..---
    case '3':  return B11000011; break;    // 3  ...--
    case '4':  return B11000001; break;    // 4  ....-
    case '5':  return B11000000; break;    // 5  .....
    case '6':  return B11010000; break;    // 6  -....
    case '7':  return B11011000; break;    // 7  --...
    case '8':  return B11011100; break;    // 8  ---..
    case '9':  return B11011110; break;    // 9  ----.
    case ' ':  return B11101111; break;    // Space
    case '/':  return B11010010; break;    // /  -..-.
    default: return charCode(' ');
   }
}

//
// Ustawienia wartosci na wyjsciu FSK (kluczowania nosnej w.cz.)
void setFSK(boolean high)
{
  if (high)
    analogWrite(FSK, FSK_HIGH);
  else
    analogWrite(FSK, FSK_LOW);
}

//
// Wlaczenie wzmacniacza mocy przez tranzystor Q4/D0 i wylaczenie tlumika Q5/D1
void setRF(boolean on)
{
  digitalWrite(KEY, on);
  digitalWrite(ATT, !on);
}

//
// Funkcja wywolywana 1000 razy na sekunde
void beacon()
{
  static byte timerCounter;          // Licznik dla podzialu przez 100 dla otrzymania 10 Hz
  static int ditCounter;             // Licznik do pomiaru dlugosci kropek
  static byte pause;                 // Przerwa miedzy znakami
  static byte msgIndex = 255;        // numeracja znakow komunikatu
  static byte character;             // Kombinacja bitow nadawanego znaku
  static byte key;                   // Stan wyjscia kluczujacego
  static byte charBit;               // Numer nadawanego bitu
  static byte ditSpeed;              // Wskaznik do tabeli szybkosci
  static byte mode;                  // Tryb pracy (None, QRSS, FSK/CW albo DFCW)
  static boolean dah;                // Wartosc 1 w czasie nadawania kreski
```

```
 byte divisor;                          // Podzial 1 kHz przez 100, dla DFCW przez 33
 // Odczyt przelacznikow wyboru emisji (trybu) i szybkosci
 mode = digitalRead(SWITCH1) + 2 * digitalRead(SWITCH2);
 ditSpeed = digitalRead(SWITCH3) + 2 * digitalRead(SWITCH4);

 if (mode == MODE_DFCW)       // Podzial przez 33 dla DFCW, dla uzyskania prawidlowego rytmu
  divisor = 33;                // Odstep miedzy symbolami ma dlugosc 1/3 kropki
 else
  divisor = 100;               // Dla pozostalych standardowe dlugosci

 timerCounter++;               // generacja 1000 Hz

 if (timerCounter == divisor)       // Podzial przez 100 lub 33 dla DFCW
 {
  timerCounter = 0;                 // generacja 10 Hz (30Hz dla DFCW)

  ditCounter++;                     // Licznik długosci kropki
  if (ditCounter >= speeds[ditSpeed])
  {
    ditCounter = 0;

    if (!pause)
    {                               // Przerwa ustawiana na 2 po ostatnim elemencie znaku
      key--;                        // Pomiar przerwy pomiedzy znakami (3 kropki)
      if ((!key) && (!charBit))
      {
       if (mode == MODE_DFCW)
         pause = 3;     // Dla DFCW konieczne dodatkowe opoznienie dla uzyskania dlugosci 4/3 kropki
       else
         pause = 2;
      }
    }
    else
      pause--;

// Zmienna key przyjmuje wartosc 255 po nadaniu biezacego elementu znaku (kropki lub kreski)
    if (key == 255)
    { // Po nadaniu ostatniego elementu biezacego znaku wczytanie nastepnego znaku
      if (!charBit)
      {               // Nastepna wartosc wskaznika znakow komunikatu
        msgIndex++;
              // Skok na poczatek komunikatu po jego nadaniu
        if (!msg[msgIndex]) msgIndex = 0;
         // Wczytanie zakodowanego bitowo znaku alfabetu Morse‘a
        character = charCode(msg[msgIndex]);
                // Poczatek od 7 (najstarszego) bitu w bajcie
        charBit = 7;
           // Poszukiwanie bitu 0 oznaczajacego poczatek kodu znaku
        while (character & (1<<charBit))
          charBit--;
      }

      charBit--;                    // Przejscie do kolejnego (mlodszego) bitu zakodowanego znaku
                                    // Przypadek specjalny dla znaku odstepu
      if (character == charCode(' '))
```

```
        {
          key = 0;
          dah = false;
        }
        else
        {                   // Wczytanie wartosci biezacego bitu kodu
          key = character & (1<<charBit);
          if (key)              // Jesli bit ma wartosc 1 oznacza kreske
          {
            key = 3;
            dah = true;
          }
          else                      // w przypadku przeciwnym – kropke
          {
            if (mode == MODE_DFCW)     // Przypadek specjalny dla DFCW – kreski i kropki
              key = 3;                 // o tej samej dlugosci.
            else
              key = 1;

            dah = false;
          }
        }
      }

      if (!key) dah = false;

      if (mode == MODE_FSKCW)
      {
        setRF(true);        // dla FSK/CW, sygnal w.cz. zawsze nadawany
        setFSK(key);        // kluczowanie FSK zalezne od stanu logicznego zmiennej key
      }
      else if (mode == MODE_QRSS)
      {
        setRF(key);                 // Dla QRSS kluczowanie amplitudy nosnej w.cz.
        setFSK(false);                // a FSK zawsze wylaczone
      }
      else if (mode == MODE_DFCW)
      {
        setRF(key); // Dla DFCW kluczowanie amplitudy nosnej w.cz. (sygnal w czasie kropek i kresek).
        setFSK(dah);              // a FSK zalezy od stanu zmiennej key
      }
      else
        setRF(false);
      digitalWrite(LED, key);          // Sygnalizacja kluczowania na diodzie swiecacej Arduino
    }
  }
}
```

Radiolatarnia dalekopisowa Hella

Prosty program nadawczy dla normy Feldhell (podstawowej normy spotykanej na pasmach amatorskich) został opracowany przez Marka Vandewetteringa i jest dostępny m.in. w jego w witrynie *www.brainwagon.org*.
Program korzysta z czcionek podzielonych na 98 elementów (14 x 7) opartych na oryginalnym alfabecie Rudolfa Hella z lat 1920-tych. Dzięki podziałowi elementów na półpola uzyskuje się ładniejszy kształt liter ale dla utrzymania standardowej szerokości sygnału transmitowane są zawsze minimum dwa półpola (ich liczba jest dowolna, może być też nieparzysta). Podział na półpola i zasada transmisji minimum dwóch z nich dotyczy także przerw między elementami znaku – czyli jego „białych" części. Każde z nich ma długość 4,081 ms co odpowiada długości pełnego elementu (przy podziale na 7 x 7 elementów) 8,163 ms. Znak „@" został dodany przez OE1KDA. Program transmituje stały komunikat podany w wywołaniu funkcji encode() w pętli loop(). Stosunkowo łatwo można go jednak przystosować do transmisji różnych komunikatów – telemetrycznych – w zależności od stanu wybranych wejść logicznych lub napięcia panującego na jednym z wejść analogowych lub na kilku z nich. Może on też – podobnie jak i wiele następnych – służyć jako laboratoryjne źródło sygnałów dla różnego rodzaju pomiarów, badań nadajników i odbiorników itp.
Szczegółowy opis systemu Hella i jego odmian znajduje się w tomie poświęconym pracy emisjami cyfrowymi na falach krótkich.



Rys. 3.1. Odbiór tekstu w normie Feldhell na ekranie programu Fldigi. Tekst nadawany jest raz ale wyświetlany podwójnie.

Program kluczuje amplitudę nadawanego sygnału w.cz. jak w przypadku zwykłej telegrafii i korzysta do tego celu z wyjścia logicznego 10 na płytce. Zmiany wyjścia kluczującego dokonuje się w pierwszej linii programu. W kodach zawartych w tabeli glyphtab[] bity o wartości 1 odpowiadają elementów znaku (polom „czarnym") a 0 – odstępom między nimi (polom „białym). Tabela glyphtab jest tabelą struktur typu Glyph zawierajacych zmienną ch oznaczającą znak alfabetu i tabelę zawartości 7 kolumn tej litery. Zmienne tabeli są typu integer a więc dwa z 16 bitów pozostają niewykorzystane. Dla nadania znaku konieczne jest najpierw znalezienie w tej złożonej tabeli odpowiedniej litery lub cyfry i następnie wykorzystanie 7-elementowej tabeli col zawierającej kolumny znaku.
Po odliczeniu czasu wykonywania programu czas oczekiwania na zakończenie elementu przyjęto na 4045 µs. W tej prostej wersji programu wykrzystano funkcję delayMicroseconds() mimo, że nie jest to najbardziej elegancki sposób wykonania zadania. W czasie wykonywania tej funkcji procesor jest zablokowany i nie może wykonywać żadnych innych zadań. Należy tu jednak zauważyć, że i tak nie

miałby nic do roboty. Bardziej eleganckim rozwiązaniem byłoby zastosowanie tutaj pomiaru czasu w sposób podobny jak w programie poprzednim ale z wykorzystaniem licznika mikrosekund a nie milisekund dla otrzymania niezbędnej dokładności. Argument funkcji opóźnienia może wymagać lekkiej korekty w zależności od dokładnej częstotliwości kwarcu.
Jeżeli odbierany tekst na ekranie wznosi się do góry oznacza to, że szybkość pracy (częstotliwość elementów) odbiornika jest większa od szybkości nadawania a jeżeli opada – jest ona mniejsza.

Kod źródłowy programu

```
int radioPin = 10;                          // kluczowanie nadajnika
typedef struct glyph {
char ch;
word col[7];
} Glyph;                                    // struktura dla znaku i jego elementow
Glyph glyphtab[] PROGMEM = {
{' ', {0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'A', {0x07fc, 0x0e60, 0x0c60, 0x0e60, 0x07fc, 0x0000, 0x0000}},
{'B', {0x0c0c, 0x0ffc, 0x0ccc, 0x0ccc, 0x0738, 0x0000, 0x0000}},
{'C', {0x0ffc, 0x0c0c, 0x0c0c, 0x0c0c, 0x0c0c, 0x0000, 0x0000}},
{'D', {0x0c0c, 0x0ffc, 0x0c0c, 0x0c0c, 0x07f8, 0x0000, 0x0000}},
{'E', {0x0ffc, 0x0ccc, 0x0ccc, 0x0c0c, 0x0c0c, 0x0000, 0x0000}},
{'F', {0x0ffc, 0x0cc0, 0x0cc0, 0x0c00, 0x0c00, 0x0000, 0x0000}},
{'G', {0x0ffc, 0x0c0c, 0x0c0c, 0x0ccc, 0x0cfc, 0x0000, 0x0000}},
{'H', {0x0ffc, 0x00c0, 0x00c0, 0x00c0, 0x0ffc, 0x0000, 0x0000}},
{'I', {0x0ffc, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'J', {0x003c, 0x000c, 0x000c, 0x000c, 0x0ffc, 0x0000, 0x0000}},
{'K', {0x0ffc, 0x00c0, 0x00e0, 0x0330, 0x0e1c, 0x0000, 0x0000}},
{'L', {0x0ffc, 0x000c, 0x000c, 0x000c, 0x000c, 0x0000, 0x0000}},
{'M', {0x0ffc, 0x0600, 0x0300, 0x0600, 0x0ffc, 0x0000, 0x0000}},
{'N', {0x0ffc, 0x0700, 0x01c0, 0x0070, 0x0ffc, 0x0000, 0x0000}},
{'O', {0x0ffc, 0x0c0c, 0x0c0c, 0x0c0c, 0x0ffc, 0x0000, 0x0000}},
{'P', {0x0c0c, 0x0ffc, 0x0ccc, 0x0cc0, 0x0780, 0x0000, 0x0000}},
{'Q', {0x0ffc, 0x0c0c, 0x0c3c, 0x0ffc, 0x000f, 0x0000, 0x0000}},
{'R', {0x0ffc, 0x0cc0, 0x0cc0, 0x0cf0, 0x079c, 0x0000, 0x0000}},
{'S', {0x078c, 0x0ccc, 0x0ccc, 0x0ccc, 0x0c78, 0x0000, 0x0000}},
{'T', {0x0c00, 0x0c00, 0x0ffc, 0x0c00, 0x0c00, 0x0000, 0x0000}},
{'U', {0x0ff8, 0x000c, 0x000c, 0x000c, 0x0ff8, 0x0000, 0x0000}},
{'V', {0x0ffc, 0x0038, 0x00e0, 0x0380, 0x0e00, 0x0000, 0x0000}},
{'W', {0x0ff8, 0x000c, 0x00f8, 0x000c, 0x0ff8, 0x0000, 0x0000}},
{'X', {0x0e1c, 0x0330, 0x01e0, 0x0330, 0x0e1c, 0x0000, 0x0000}},
{'Y', {0x0e00, 0x0380, 0x00fc, 0x0380, 0x0e00, 0x0000, 0x0000}},
{'Z', {0x0c1c, 0x0c7c, 0x0ccc, 0x0f8c, 0x0e0c, 0x0000, 0x0000}},
{'0', {0x07f8, 0x0c0c, 0x0c0c, 0x0c0c, 0x07f8, 0x0000, 0x0000}},
{'1', {0x0300, 0x0600, 0x0ffc, 0x0000, 0x0000, 0x0000, 0x0000}},
{'2', {0x061c, 0x0c3c, 0x0ccc, 0x078c, 0x000c, 0x0000, 0x0000}},
{'3', {0x0006, 0x1806, 0x198c, 0x1f98, 0x00f0, 0x0000, 0x0000}},
{'4', {0x1fe0, 0x0060, 0x0060, 0x0ffc, 0x0060, 0x0000, 0x0000}},
{'5', {0x000c, 0x000c, 0x1f8c, 0x1998, 0x18f0, 0x0000, 0x0000}},
{'6', {0x07fc, 0x0c66, 0x18c6, 0x00c6, 0x007c, 0x0000, 0x0000}},
{'7', {0x181c, 0x1870, 0x19c0, 0x1f00, 0x1c00, 0x0000, 0x0000}},
{'8', {0x0f3c, 0x19e6, 0x18c6, 0x19e6, 0x0f3c, 0x0000, 0x0000}},
{'9', {0x0f80, 0x18c6, 0x18cc, 0x1818, 0x0ff0, 0x0000, 0x0000}},
{'*', {0x018c, 0x0198, 0x0ff0, 0x0198, 0x018c, 0x0000, 0x0000}},
{'.', {0x001c, 0x001c, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'?', {0x1800, 0x1800, 0x19ce, 0x1f00, 0x0000, 0x0000, 0x0000}},
```

```
{'!', {0x1f9c, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'(', {0x01e0, 0x0738, 0x1c0e, 0x0000, 0x0000, 0x0000, 0x0000}},
{')', {0x1c0e, 0x0738, 0x01e0, 0x0000, 0x0000, 0x0000, 0x0000}},
{'#', {0x0330, 0x0ffc, 0x0330, 0x0ffc, 0x0330, 0x0000, 0x0000}},
{'$', {0x078c, 0x0ccc, 0x1ffe, 0x0ccc, 0x0c78, 0x0000, 0x0000}},
{'/', {0x001c, 0x0070, 0x01c0, 0x0700, 0x1c00, 0x0000, 0x0000}},
{'@', {0x1ff0, 0x3018, 0x670c, 0x678c, 0x3f38, 0x0000, 0x0000}},
};                                                  // Tabela struktur opisujacych znaki
#define NGLYPHS (sizeof(glyphtab)/sizeof(glyphtab[0]))
void encodechar(int ch)                             // kodowanie znaku
{
  int i, x, y, fch ;
  word fbits ;
/* Kontynuowanie poszukiwania wydaje sie zbedne po znalezieniu
* poszukiwanego znaku ale dzieki temu czas wykonywania tej czesci programu
* jest staly co ulatwia uzyskanie wymaganej stalosci szybkosci transmisji
*/
  //Serial.print(ch);
  for (i=0; i<NGLYPHS; i++) {
    fch = pgm_read_byte(&glyphtab[i].ch);
      if (fch == ch) {
        for (x=0; x<7; x++) {
          fbits = pgm_read_word(&(glyphtab[i].col[x])) ;
          for (y=0; y<14; y++) {
            if (fbits & (1<<y))
            digitalWrite(radioPin, HIGH);
          else
            digitalWrite(radioPin, LOW);
          delayMicroseconds(4045L);
                        }
                    }
                }
        }
}
void encode(char *ch)
{
  while (*ch != '\0')
    encodechar(*ch++);
}
void
setup()
{
  Serial.begin(38400) ;
  pinMode(radioPin, OUTPUT);
}
void
loop()
{
  encode("OE1KDA JN88ED 500 MILIWATTS QSL: OE1KDA@OEVSV.AT ");
}
```

Klucz elektroniczny

Program generuje sygnał m.cz. kluczowany znakami alfabetu Morse'a nadawanymi kluczem sztorcowym lub bocznym.
Stała USE_LCD pozwala na skompilowanie wersji wyposażonej we wskaźnik ciekłokrystaliczny lub bez jego użycia. Zastępuje go wówczas złącze szeregowe przez które dane są przekazywane do okna monitora Arduino.
Rytm pracy programu nadaje licznik TIMER1 a jego przepełnienie powoduje wywołanie podprogramu przerwania ISR(TIMER0_OVF_vect).
Próbki sygnału sinusoidalnego zawarte są w tabeli sine_table[] mającej długość 256 elementów. Tabela znajduje się w pamięci roboczej RAM ale rozwiązaniem korzystniejszym i pozostawiającym więcej miejsca na zmienne robocze programu w przypadku jego rozbudowy byłoby umieszczenie jej w pamięci programu jak to zobaczymy w następnych przykładach. Mamy więc tutaj (z omówioną dalej dokładniej) bezpośrednią syntezą cyfrową sygnału (ang. *DDS*). W tym przypadku ze względu na szybkość pracy procesora synteza możliwa jest jedynie w zakresie m.cz. co w pełni wystarcza do realizacji zadania. Sygnał m.cz. może być podawany na wejście mikrofonowe radiostacji SSB równolegle do jego odsłuchu przez głośnik. Nadajnik może być także kluczowany amplitudowo.
Wyjściowy sygnał akustyczny uzyskiwany jest na wyjściu pseudoanalogowym pracującym z modulacją szerokości impulsów (PWM). Wymaga on odfiltrowania za pomocą filtru dolnoprzepustowego.

Kod źródłowy programu

```
// TONE   D6        D6
// MODE   D7        D7

// Sygnal wyjsciowy PWM (nozka D6 w Arduino Nano) jest podawany przez
// prosty filtr dolnoprzepustowy RC o czestotliwosci granicznej
// ok. 16 kHz (R = 10 K, C = 0,001 uF)
// na wejscie scalonego wzmacniacza LM386.
// Standardowo czestotliwosc podnosnej akustycznej wynosi 600 Hz
// ale mozna ja zmienic za pomoca przyciskow

// Czestotliwosc modulowanych impulsow PWM wynosi 62,5 KHz.

#include <Arduino.h>

#include <stdint.h>
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#ifdef USE_LCD
#include <lcd.h>
#else
#ifdef USE_SERIAL
#define lcd_init() Serial.begin(19200)
#define lcd_line1(x) Serial.println(x)
#define lcd_line2(x) Serial.println(x)
#else
#define lcd_init()
#define lcd_line1(x)
```

```
#define lcd_line2(x)
#endif
#endif

#define UI_DELAY 80
#define STRAIGHT 0
#define PADDLE 1

void dash(uint8_t key_xmitter);

uint8_t mode;
uint8_t reverse_polarity;
uint8_t left_key_down, right_key_down, key_down;

volatile int wpm;                       // zmienna ulotna modyfikowana przez przerwanie
int dot_time;
char buf[64];

volatile uint8_t modulation;            // zmienna ulotna modyfikowana przez przerwanie
volatile uint8_t rl_modulation = 0;   // modulacja o ograniczonej szybkosci;

volatile uint8_t timer1_busy = 0;     // zmienna ulotna modyfikowana przez przerwanie

int frequency;
int8_t sine_table[256];                 // tabela probek sinusoidy
volatile uint16_t phase;                // zmienna ulotna modyfikowana przez przerwanie
volatile uint16_t delta_phase;

ISR(TIMER0_OVF_vect)      // Podprogram przerwania wywolywany po przejsciu licznika 0 przez 0
{
    int value;

    if (rl_modulation < modulation) {
        rl_modulation++;
    } else if (rl_modulation > modulation) {
        rl_modulation--;
    }

    value = sine_table[phase >> 8];
    value = value * rl_modulation;      // [-(127*256)..(127*256)]
    value = value + (127*255);
    value = value >> 8;
    phase += delta_phase;

    OCR0A = value;                  // wydanie wartosci na wyjscie PWM
    return;
}

ISR(TIMER1_OVF_vect)    // Podprogram przerwania wywolywany przy przejsciu licznika 1 przez 0
{
    modulation = 0;
    timer1_busy = 0;
}

void start_timer1(int msec)
```

```
{
    uint16_t count;
    TCCR1B = 0;                 // Wylaczenie licznika 1 (timer1)
    TIFR1 = TOV1;               // Wyzerowanie bitu przepelnienia
    timer1_busy = 1;

    // licznik zlicza do 0xFFFF
    count = trunc(65536.0 - (msec * 1000.0)/16.0);
    cli();
    TCNT1 = count;
    sei();
    timer1_busy = 1;
    TCCR1B = 0x04;              // wlaczenie licznika 1 (timer1)
    // 16 usec ticks
}

void check_key()
{
    // D4 = Ostrze wtyczki, D5 = pierscien wtyczki
    if (reverse_polarity) {
        right_key_down = ((PIND & _BV(PORTD4)) == 0);
        left_key_down = ((PIND & _BV(PORTD5)) == 0);
    } else {
        left_key_down = ((PIND & _BV(PORTD4)) == 0);
        right_key_down = ((PIND & _BV(PORTD5)) == 0);
    }
}

void space(uint8_t i)
{
    while (i > 0) {
         start_timer1(dot_time);
         while (timer1_busy != 0);
         i--;
    }
}

void dot(uint8_t key_xmitter)     // kluczowanie - kropka
{
    uint8_t got_dash = 0;
    modulation = 255;
    if (key_xmitter) PORTB |= _BV(PORTB4);
    PORTB |= _BV(PORTB5);
    start_timer1(dot_time);
    while (timer1_busy != 0) {
         // check for dash
         check_key();
         if (right_key_down) got_dash = 1;
    }
    modulation = 0;
    PORTB &= ~_BV(PORTB4);
    PORTB &= ~_BV(PORTB5);
    space(1);
    if (got_dash) dash(key_xmitter);
}
```

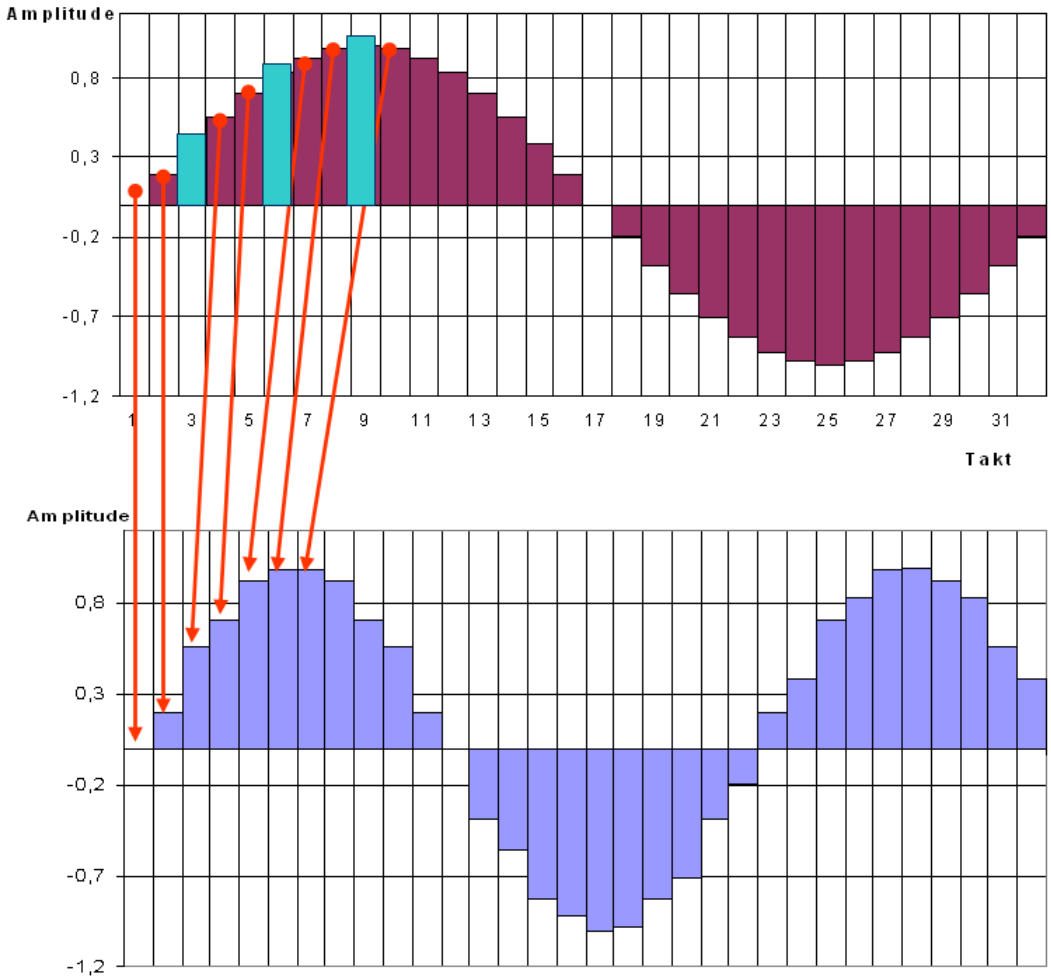
```
void dash(uint8_t key_xmitter)          // kluczowanie - kreska
{
  uint8_t got_dot = 0;
  modulation = 255;
  if (key_xmitter) PORTB |= _BV(PORTB4);
  PORTB |= _BV(PORTB5);
  start_timer1(dot_time * 3);
  while (timer1_busy != 0) {
    // sprawdzenie czy kropka
    check_key();
    if (left_key_down) got_dot = 1;
  }

  modulation = 0;
  PORTB &= ~_BV(PORTB4);
  PORTB &= ~_BV(PORTB5);
  space(1);
  if (got_dot) {
    dot(key_xmitter);
  }
}

void update_display()                   // wyswietlenie informacji
{
  if (mode == STRAIGHT) {
    lcd_line1("Klucz sztorcowy");
  } else {
    if (reverse_polarity)
      lcd_line1("Klucz boczny (odwr)");
    else lcd_line1("Klucz boczny");
  }
  sprintf(buf, "%d Hz %d WPM", frequency, wpm);
  lcd_line2(buf);
}

int main( void )
{
  int i;

  // Inicjalizacja tabeli probek - sine_table[] – w pamieci RAM
  for (i = 0; i < 256; i++) {
    sine_table[i] = 127 * sin((2*M_PI*i)/256.0);
  }

  DDRB |= _BV(PORTB5); // Wyjscie na diode swiecaca - LED
  DDRD |= _BV(PORTD6); // Wyjscie z modulacja szerokosci impulsow PWM
  DDRB |= _BV(PORTB4); // Wyjscie kluczujace nadajnik.

  // Wlaczenie opornikow podtrzymujacych na wejsciach
  PORTD |= _BV(PORTD4) | _BV(PORTD5) |_BV(PORTD7);
  PORTB |= _BV(PORTB0) | _BV(PORTB1) | _BV(PORTB2) | _BV(PORTB3) | _BV(PORTB4);

  TCCR0A = 0x83;
  TCCR0B = 0x01;
```

```
  OCR0A = 128;
  TIFR0 = 0x01;                 // Wyzerowanie bitu przepelnienia dla licznika 0

  TIMSK0 = 0x01;
  TCCR1A = 0x00;
  TIFR1 = 0x01;                 // Wyzerowanie bitu przepelnienia dla licznika 1

  TIMSK1 = 0x01;
  PORTB &= ~_BV(PORTB4);       // Wyzerowanie wyjscia kluczujacego

  lcd_init();
  if (!(PIND & _BV(PORTD7)))   // rozpoznanie rodzaju klucza – trybu pracy
      mode = PADDLE;
  else
      mode = STRAIGHT;

  frequency = 600;
  wpm = 15;
  dot_time = 1200/wpm;
  phase = 0;
  delta_phase = round((256.0*256.0*16.0*frequency)/1000000.0);  // probkowanie co 16 us
  modulation = 0;
  rl_modulation = 0;
  timer1_busy = 0;

  // Jezeli w momencie wlaczenia wejscie w stanie 0 zamiana przyporzadkowania kropek i kresek w
kluczu bocznym
  check_key();
  reverse_polarity = left_key_down || right_key_down;
  update_display();
  sei();

  dot(0); dot(0); dot(0); dash(0);

  while (1) {
    check_key();

    if (mode == STRAIGHT) {
      // Tryb pracy klucza storcowego - STRAIGHT
      if (!(PIND & _BV(PORTD7))) {
        mode = PADDLE;
        update_display();
      }

      key_down = left_key_down || right_key_down;

      if (key_down) {
        modulation = 255;
        PORTB |= _BV(PORTB4); // wlaczenie nadajnika
        PORTB |= _BV(PORTB5); // LED
      } else {
        modulation = 0;
        PORTB &= ~_BV(PORTB4); // wylaczenie nadajnika
        PORTB &= ~_BV(PORTB5); // LED
      }
```

```
    } else {
      // Tryb pracy klucza bocznego
      if (PIND & _BV(PORTD7)) {
        mode = STRAIGHT;
        update_display();
      }

      if (left_key_down) {
        dot(1);
      } else {
        modulation = 0;
      }

      if (right_key_down) {
        dash(1);
      } else {
        modulation = 0;
      }
    }

    if (((PINB & _BV(PORTB2)) == 0) || ((PINB & _BV(PORTB3)) == 0)) {
      // zmiana szybkosci transmisji - wpm
      if (PINB & _BV(PORTB2)) wpm--; else wpm++;
      if (wpm >= 30) wpm = 30;
      if (wpm <= 5) wpm = 5;
      dot_time = 1200/wpm;
      update_display();
      dot(0);
    }

    if (((PINB & _BV(PORTB0)) == 0) || ((PINB & _BV(PORTB1)) == 0)) {
      // zmiana czestotliwosci sygnalu akustycznego
      if (PINB & _BV(PORTB0)) frequency -= 10; else frequency += 10;
      if (frequency <= 300) frequency = 300;
      if (frequency >= 1200) frequency = 1200;
      delta_phase = round((256.0*256.0*16.0*frequency)/1000000.0); // szybkosc probkowania 1/16 us
      update_display();
      dot(0);
    }
  }

  return 0;
}
```

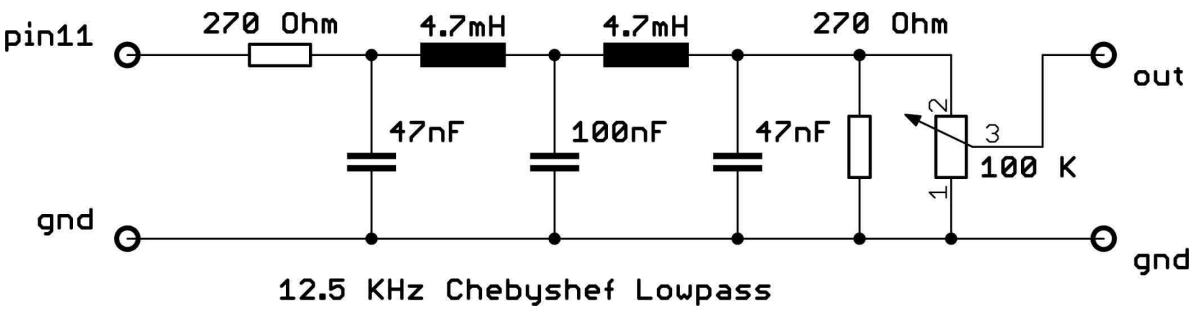
Cyfrowy generator sygnału m.cz.

Bezpośrednia cyfrowa synteza sygnałów m.cz. lub w.cz. polega na cyklicznym odczytywaniu próbek sygnału (najczęściej jest to sinusoida) z pamięci. Do jej adresowania służy specjalny licznik adresowy zwany również licznikiem fazy lub akumulatorem fazy.
Odczytane z pamięci wartości próbek (u góry na rys. 5.1) podawane są na przetwornik cyfrowo-analogowy, na którego wyjściu otrzymuje się schodkowe przybliżenie sygnału wyjściowego (na rys. 5.1 u dołu). Dla uzyskania dostatecznie czystego sinusoidalnego sygnału wyjściowego konieczne jest odfiltrowanie go za pomocą filtru dolnoprzepustowego. Próbki odczytywane są z ustaloną częstotliwością zegarową – taktu. W najprostszym przypadku gdy stan licznika fazy zmienia się w każdym cyklu zegarowym o 1 na wyjściu syntezera otrzymujemy sinusoidę złożoną z pełnej liczby próbek i częstotliwości równej częstotliwości zegarowej podzielonej przez liczbe próbek. W przykładzie z rys. 5.1 dla zawartych w pamięci 32 próbek przy częstotliwości zegarowej 32 kHz na wyjściu otrzymywany jest sygnał sinusoidalny o częstotliwości 1 kHz. Ten sygnał można nazwać sygnałem naturalnym lub sygnałem symetrii dla danego syntezera i danej częstotliwości zegarowej. Dla otrzymania sygnałów o wyższych częstotliwościach konieczne jest opuszczanie – pomijanie w odczycie – pewnych próbek a dla otrzymania sygnałów o częstotliwościach niższych powtarzanie w odczycie niektórych z nich. Sytuacja nie jest jednak taka prosta jak mogłoby się wydawać na pierwszy rzut oka. Odczyt co drugiej albo co trzeciej próbki da wprawdzie przebieg stosunkowo ładny i symetryczny (dla dużej liczby próbek wyjściowych) ale skok częstotliwości będzie duży. Na wyjściu otrzymano by częstotliwości dwu- lub trzykrotnie większe itd.

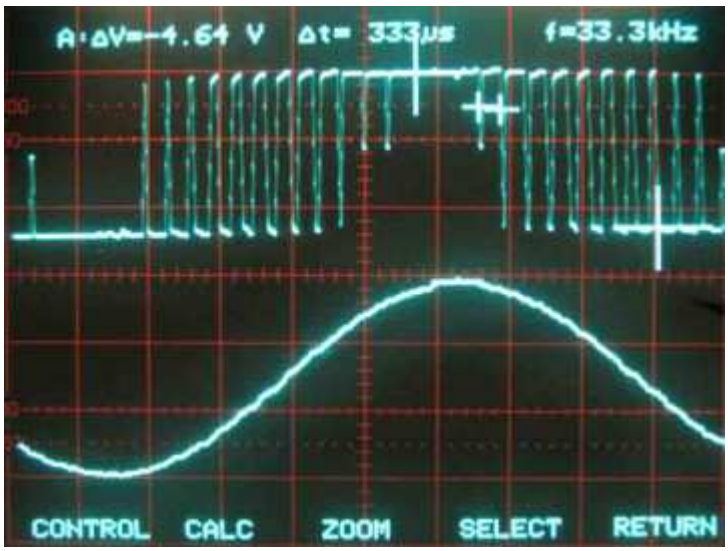


Rys. 5.1. Zasada bezpośredniej cyfrowej syntezy częstotliwości

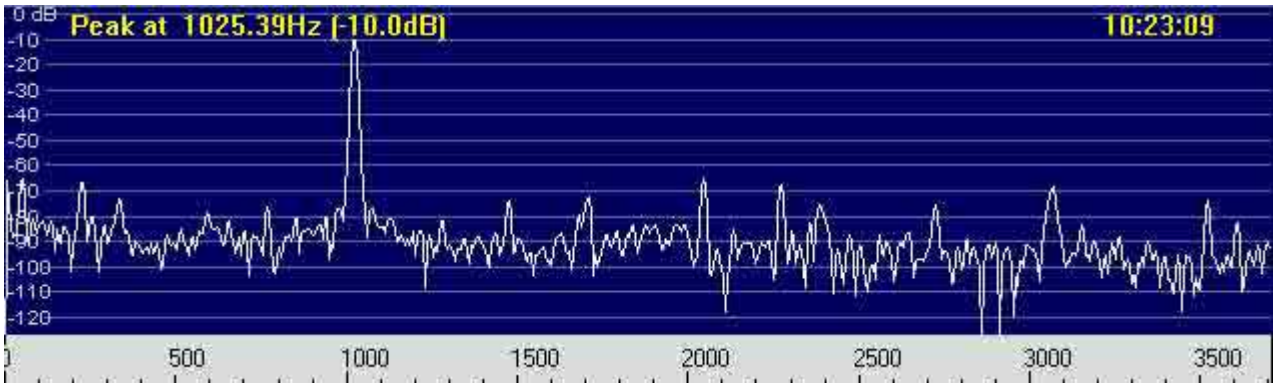
Przy opuszczaniu 1/3 próbek (jednej na trzy jak na rysunku) uzyskuje się częstotliwość półtora raza większą. Opuszczenie jednej próbki na 1000 dawałoby wzrost częstotliwości o 1/1000 i w ten sposób zbliżamy się już do zasady pracy rzeczywistego syntezera. Opuszczanie lub powtarzanie próbek w tak dużych lub jeszcze większych stosunkach najłatwiej zrealizować gdy licznik fazy zmienia swoją wartość o ułamki adresu pamięci próbek (skok fazy jest ułamkowy). Przykładowo gdy stan licznika (akumulatora) będzie się zmieniał o 1,001 to po tysiącu kroków jego stan przeskoczy o dwa adresy i w odczycie zostanie pominięta jedna próbka. Dla skoków fazy 1,002 próbka zostanie pominięta co 500 kroków, dla zmian o 0,999 co 1000 kroków jedna z próbek zostanie odczytana dwukrotnie (powtórzona) a dla zmian stanu licznika o 0,998 – dwukrotny odczyt wystąpi co 500 kroków. Licznik pracuje jak widzimy na ułamkach ale do adresowania pamięci używane są wyłącznie części całkowite a części ułamkowe są ignorowane. Ponieważ próbki są odczytywane z pamięci cyklicznie (po dojściu do ostatniej licznik wraca do stanu zerowego) ten cykl opuszczania lub powtarzania próbek może być znacznie dłuższy od długości tabeli, ale może też być od niej krótszy w zależności od generowanej częstotliwości. Nie musi on być też w żaden sposób zsynchronizowany z cyklem odczytu tabeli.
W omawianych przykładach dla tabeli o długości 256 próbek przy opuszczaniu co tysięcznej sytuacja taka wystapi dopiero w czwartym cyklu odczytu tabeli. W pierwszych trzech odczytywana byłaby cała jej zawartość. Oczywiście w rzeczywistych rozwiązaniach licznik pracuje dwójkowo i ma długość wyrażaną w bitach o wiele z nich dłuższą od części (górnej) używanej do adresowania. Przykładowo z 28 bitów do adresowania wykorzystywanych jest górnych 16 a pozostałych 12 niższych stanowi tą umowną część ułamkową.
Sygnał wyjściowy ma przebieg nierównomierny – w pewnych momentach występują skoki amplitudy powodujące zwiększenie jego odchyłki od poprawnej sinusoidy. To oraz jego schodkowy charakter spowodowany dyskretnym a nie ciągłym wydawaniem wartości powodują konieczność filtrowania go za pomocą mniej lub bardziej (a przeważnie bardziej) rozbudowanych filtrów dolnoprzepustowych.
Teoretycznie z zasady próbkowania Nyquista wynika,że najwyższa częstotliwość wyjściowa jest równa połowie częstotliwości zegarowej syntezera. W praktyce użyteczne wartości dochodzą do około jej 40%. Zasadniczo rozdzielczość amplitudy powinna być o 2 lub więcej bitów większa od rozdzielczości fazy i dla podanego dalej rozwiązania powinna wynosić co najmniej 10 bitów – jednak dla uproszczenienia w procesorze 8-bitowym przyjęto obie rozdzielczości wynoszące 8 bitów – po 256 wartości.
Dla uzyskania wysokich częstotliwości stosuje się specjalne scalone układy cyfrowych syntezerów pracujące z częstotliwościami zegarowymi od kilkudziesięciu do ponad 100 i więcej MHz. Zawierają one oczywiście pamięci próbek o większej pojemności i długości słowa aniżeli w rozwiązaniu programowym dla Arduino.
Jednym z takich popularnych w kręgach krótkofalarskich obwodów jest AD9850 ale gama produktów jest bardzo szeroka i można wybrać pasujący do potrzeb układ scalony. Obwody takie można sterować za pomocą Arduino ale tą sprawą zajmiemy się w dalszej części opracowania.
W przykładowym programie opracowanym przez Martina Nawratha DH3JO Arduino pracuje jako programowy syntezer cyfrowy. Jego szybkość pracy pozwala na generowanie jedynie sygnałów małej częstotliwości (akustycznych), ale mogą one służyć do sprawdzania i uruchamiania układów a przy odpowiednim ich kluczowaniu w programie (możliwe jest zarówno kluczowanie amplitudy jak i częstotliwości a w bardziej rozbudowanych rozwiązaniach także fazy) Arduino może być wykorzystane jako źródło sygnałów różnych emisji amatorskich. Sygnał ten po odfiltrowaniu w filtrze dolnoprzepustowym (przykład rozwiązania filtru użytego przez autora programu przedstawia rys. 5.2) może być podany na wejście mikrofonowe amatorskich nadajników SSB lub FM. Ten sam filtr jest konieczny również i dla następnych programów wykorzystujących syntezer cyfrowy m.cz. kluczowany emisjami cyfrowymi.
Częstotliwość generatora m.cz. w pierwszym z przykładów jest regulowana za pomocą potencjomeru podłączonego do wejścia analogowego A0. Sygnał impulsowy z modulacją szerokości impulsów (ang. *PWM*) jest pobierany z wyjścia 11.
Syntezer ten jest wykorzystywany w kilku następnych przykładach do generowania podnośnej m.cz. kluczowanej amplitudowo lub częstotliwościowo przez sygnały różnych emisji cyfrowych.
Programiści posiadający pewne doświadczenie mogą opierając się na tym module generatora i przytoczonych przykładach generowania sygnałów emisji cyfrowych opracować programy generatorów dla wielu dalszych emisji.



Rys. 5.2. Filtr dolnoprzepustowy o częstotliwości granicznej 12,5 kHz i charakterystyce Czebyszewa.



Fot. 5.3. Przebieg sygnału impulsowego na wyjściu PWM Arduino – nóżce 11 – (u góry) i po filtrze dolnoprzepustowym z rysunku poprzedniego (u dołu). Częstotliwość wyjściowa wynosi w przybliżeniu 1000 Hz. Ilustracja z witryny DH3JO.



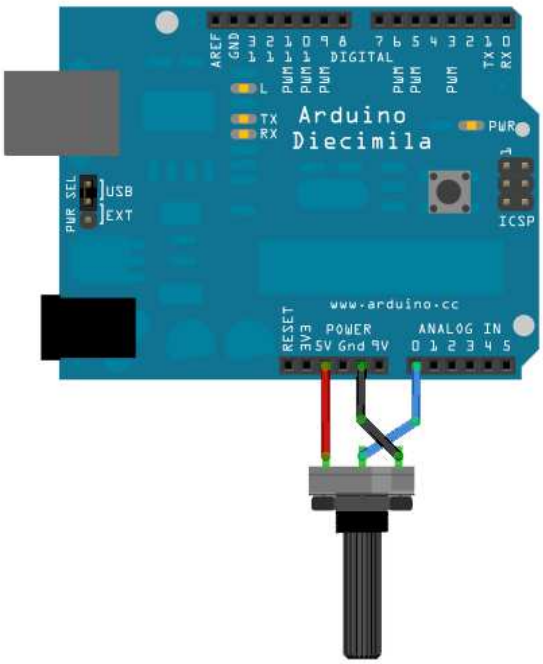
Fot. 5.4. Widmo sygnału wyjściowego syntezera na Arduino. Składowe pasożytnicze są stłumione o co najmniej 50 dB w stosunku do sygnału użytecznego. Ilustracja z witryny DH3JO.

Kod źródłowy programu

```
/*
 * Syntezer sygnalu sinusoidalnego na ATMEGA 168
 * Licznik 2 (Timer2) generuje sygnal podstawy czasu 31250 kHz dla przerwan
 *
 * KHM 2009 /  Martin Nawrath
 * Kunsthochschule fuer Medien Koeln
 * Academy of Media Arts Cologne
*/

#include "avr/pgmspace.h"

// Tabela 256 probek sinusoidy /pelny jej okres/ zawarta w pamieci programu. Sinusoida jest
przesunieta w gore o 128 a wiec jej wartosci zerowej odpowiada 128, szczytom dodatnim 255 a
szczytom ujemnym 0.
PROGMEM  prog_uchar sine256[]  = {
  127,130,133,136,139,143,146,149,152,155,158,161,164,167,170,173,176,178,181,184,187,190,192,
195,198,200,203,205,208,210,212,215,217,219,221,223,225,227,229,231,233,234,236,238,239,240,
 242,243,244,245,247,248,249,249,250,251,252,252,253,253,253,254,254,254,254,254,254,254,253,
253,253,252,252,251,250,249,249,248,247,245,244,243,242,240,239,238,236,234,233,231,229,227,
225,223,
  221,219,217,215,212,210,208,205,203,200,198,195,192,190,187,184,181,178,176,173,170,167,164,
161,158,155,152,149,146,143,139,136,133,130,127,124,121,118,115,111,108,105,102,99,96,93,90,87,
84,81,78,
  76,73,70,67,64,62,59,56,54,51,49,46,44,42,39,37,35,33,31,29,27,25,23,21,20,18,16,15,14,12,11,10,9,
7,6,5,5,4,3,2,2,1,1,1,0,0,0,0,0,0,0,0,1,1,1,2,2,3,4,5,5,6,7,9,10,11,12,14,15,16,18,20,21,23,25,27,29,31,
  33,35,37,39,42,44,46,49,51,54,56,59,62,64,67,70,73,76,78,81,84,87,90,93,96,99,102,105,108,111,
115,118,121,124

};
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))

int ledPin = 13;                // Dioda swiecaca na plytce
int testPin = 7;                // Sygnal diagnostyczny
int t2Pin = 6;
byte bb;

double dfreq;
// const double refclk=31372.549;  // =16 MHz / 510 – teoretyczna czestotliwosc odniesienia
const double refclk=31376.6;        // zmierzona czestotliwosc odniesienia

// zmienne uzywane rowniez w podprogramach przerwania zadeklarowane jako ulotne - volatile
volatile byte icnt;                     // uzywana przez podprogam przerwania
volatile byte icnt1;                    // uzywana przez podprogam przerwania
volatile byte c4ms;                     // licznik zmienia stan co  4 ms
volatile unsigned long phaccu;      // akumulator fazy
volatile unsigned long tword_m;  // slowo sterujace syntezer cyfrowy (skok fazy dla danej
czestotliwosci)

void setup()
{
  pinMode(ledPin, OUTPUT);      // Przelaczenie linii 13 do pracy jako wyjscie
  Serial.begin(115200);               // inicjalizacja zlacza szeregowego
```

```
  Serial.println("DDS Test");

  pinMode(6, OUTPUT);           // Ustawienie wyjsc na liniach 6, 7 i 11
  pinMode(7, OUTPUT);
  pinMode(11, OUTPUT);          // linia 11 – wyjscie PWM, analogowe

  Setup_timer2();

  // wylaczenie przerwan dla unikniecia zaklocen rytmu pracy
  cbi (TIMSK0,TOIE0);           // wylacznie licznika 0 (Timer0) !!! wylaczona tez funkcja delay()
  sbi (TIMSK2,TOIE2);           // wlacznie przerwania wywolywanego przez licznik 2 (Timer2)

  dfreq=1000.0;                 // poczatkowa czestotliwosc wyjsciowa = 1000,0 Hz
  tword_m=pow(2,32)*dfreq/refclk;  // obliczenie slowa sterujacego syntezer dla tej czestotliwosci
                                   // (skoku fazy)
}
void loop()
{
  while(1) {
    if (c4ms > 250) {           // licznik / oczekiwanie na uplyw sekundy
     c4ms=0;
     dfreq=analogRead(0);       // odczyt potencjometru na wejsciu analogowym 0 –
                                //strojenie czestotliwosci w zakresie 0..1023 Hz
     cbi (TIMSK2,TOIE2);        // wylacznie przerwania od licznika 2 (Timer2)
     tword_m=pow(2,32)*dfreq/refclk;  // obliczenie slowa sterujacego syntezer dla tej czestotliwosci
     sbi (TIMSK2,TOIE2);        // wlaczenie przerwan pochodzacych od licznika 2 (Timer2)

     Serial.print(dfreq);
     Serial.print("  ");
     Serial.println(tword_m);
    }

   sbi(PORTD,6); // Diagnoza / wysoki poziom na wyjsciu PORTD,7 dla obserwacji przebiegu pracy na
oscyloskopie
   cbi(PORTD,6); // Diagnoza / niski poziom na wyjsciu PORTD,7 dla obserwacji na oscyloskopie
  }
}
//*********************************************************************
// inicjalizacia licznika 2 (timer2)
// dzielnik wstepny przez 1, tryb PWM „phase correct PWM”, zegar 16000000/510 = 31372,55 Hz
void Setup_timer2() {

// Dzielnik wstepny = 1
  sbi (TCCR2B, CS20);
  cbi (TCCR2B, CS21);
  cbi (TCCR2B, CS22);

  // Tryb PWM „Phase Correct PWM“
  cbi (TCCR2A, COM2A0);  // wyzerowanie wskazania porownan
  sbi (TCCR2A, COM2A1);

  sbi (TCCR2A, WGM20);  // Tryb 1  / „Phase Correct PWM”
  cbi (TCCR2A, WGM21);
  cbi (TCCR2B, WGM22);
}
```

```
//*****************************************************************
// Podprogram przerwan licznika 2 (Timer2) wywolywany z czestot. 31372,550 kHz czyli co 32 us
// podstawa czasu REFCLOCK dla syntezera cyfrowego
// FOUT = (M (REFCLK)) / (2 exp 32)
// czas wykonywania: 8 mikrosekund (z uwzglednieniem rozkazow push i pop)
ISR(TIMER2_OVF_vect) {

  sbi(PORTD,7);          // Diagnoza / poziom wysoki na wyjsciu PORTD,7
                         //dla obserwacji na oscyloskopie
  phaccu=phaccu+tword_m; // zmiana stanu 32-bitowego akumulatora (licznika) fazy
  icnt=phaccu >> 24;     // do adresowania tabeli probek uzywane gorne 8 bitow
                    // odczyt wartosci probki (sinusoidy) z tabeli w pamieci programu i wydanie na wyjsciu
PWM – przetwornika c/a.
  OCR2A=pgm_read_byte_near(sine256 + icnt);

  if(icnt1++ == 125) {  // zmiana stanu licznika c4ms co 4 millisekundy
    c4ms++;
    icnt1=0;
   }

 cbi(PORTD,7);           // zerowanie wyjscia PORTD,7
}
```

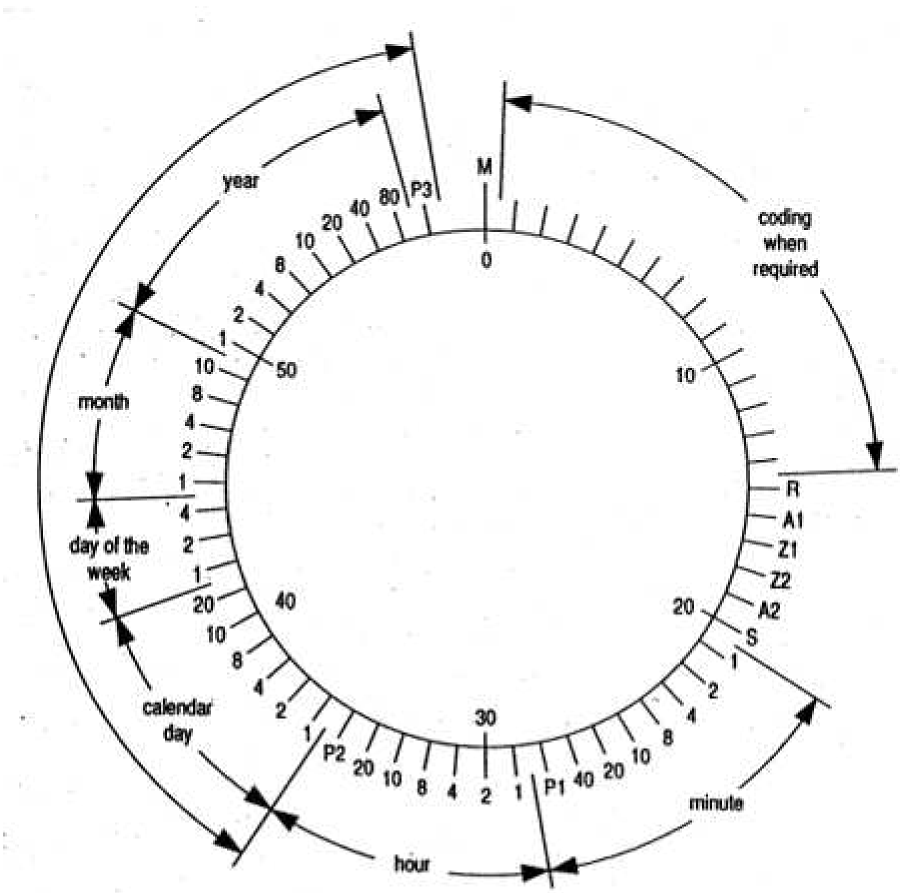


Rys. 5.5. Sposób podłączenia potencjometru strojeniowego do wejścia A0.

Radiolatarnia WSPR

Radiolatarnia WSPR bazuje na opisanym w poprzednim rozdziale programie generatora małej częstotliwości. Częstotliwość generatora nie jest tutaj oczywiście zmieniana za pomocą potencjometru a jest kluczowana w zależności od zakodowanej treści nadawanego komunikatu WSPR. Struktura komunikatu WSPR jest opisana w skrypcie poświęconymtechnice słabych sygnałów. Sygnał wyjściowy mikroprocesora jest doprowadzany do wejścia mikrofonowego nadajnika SSB. Nadawany jest standardowy komunikat zawierający znak, lokator i moc stacji w dBm. Dekodowany sygnał czasu pochodzi z odbiornika czasu wzorcowego dostrojonego do pracującej w Mainflingen w okolicach Frankfurtu n/Menem stacji DCF77, podłączonego do wejścia nr 7. Odbiornik ten dostarcza impulsów o polaryzacji dodatniej i długościach 100 lub 200 ms odpowiadających bitom o wartościach 0 i 1, za pomocą których zakodowana jest informacja o czasie, dacie, dniu tygodnia itp. Impulsy te nadawane są co sekundę. Pozostała niewykorzystana do tego celu część impulsów służy do transmisji prognoz pogody w systemie „Meteotime", ale nie ma to znaczenia dla pracy radiolatarni WSPR, która zasadniczo potrzebuje informacji o początkach 2-minutowych odcinków, w czasie których może podjąć nadawanie lub też nie w zależności od założonego prawdopodobieństwa transmisji.
Program pracuje na Arduino Duemilanove, Diecimila i UNO. Jego autorem jest Martin Nawrath DH3JO z Akademii Sztuk Medialnych w Kolonii. Konieczne jest włączenie filtru dolnoprzepustowego na wyjściu m.cz. Schemat filtru podano w opisie wyjściowego programu generatora m.cz.

Kodowanie danych DCF



Rys. 6.1. Schemat kodowania DCF77

Kodowanie wartości bitów:

- Jedynka – 0,2 sek
- Zero – 0,1 sek

- Sekundy 0 – 19 bez znaczenia
  1– 14 – „Meteotime”, informacja o zakłóceniach w pracy 15, sygnalizacja zmiany czasu 16, strefa czasowa 17–18, dodatkowe sekundy wyrównawcze 19 itp.),
- sekunda 20 – start kodu czasu, jedynka
- sekundy 21 – 27 – nadchodząca minuta w kodzie BCD,
- sekunda 28 – bit parzystości minut (P1),
- sekundy 29 – 34 – godzina w kodzie BCD
- sekunda 35 – bit parzystości godzin (P2)
- sekundy 36 – 41 – dzień,
- sekundy 42 – 44 – dzień tygodnia,
- sekundy 45 – 49 – miesiąc,
- sekundy 50 – 57 – rok,
- sekunda 58 – bit parzystości daty (P3),
- sekunda 59 – bez danych.

Czas jest kodowany dla nadchodzącej minuty.

Kod źródłowy programu

```
/*
 * Radiolatarnia WSPR do amatorskich badan propagacji za pomoca slabych sygnalow
 * kodowany i nadawany jest komunikat zawierajacy znak, lokator i moc
 * wyjsciowy sygnal sinusoidalny gnerowany jest przy uzyciu programowego syntezera cyfrowego
 * i przetwarzany na postac analogowa za pomoca modulacji szerokosci impulsow
 *  Autor dziekuje Andy Talbotowi za artykul opisujacy algorytm kodowania WSPR
 * program pracuje na Arduino Diecimila, Duemilanove, UNO / Arduino 17
 *
 * do kluczowania nadajnika sluzy wyjscie 10
 * wyjscie napiecia dla kluczowania czestotliwosci dodatkowego VCO na kontakcie 11
 * sygnal wyjsciowy z modulacja szerokosci impulsow jest dostepny na kontakcie 3
 * odbiornik sygnalow czasu DCF podlaczony do wejscia 7

 * wykorzystywany jest programowy syntezer cyfrowy dla ATMEGA168
 * Licznik 2 (Timer2) służy do generowania podstawy czasu dla syntezera i wywolywania przerwan.
 * Jej czestotliwosc wynosi 31,373 kHz
 * KHM 2009 /  Martin Nawrath
 * Kunsthochschule fuer Medien Koeln
 * Academy of Media Arts Cologne
 *
 * korekta OE1KDA - tryb 1 zamiast 3
*/
// naglowek niezbedny dla umieszczenia tabel danych w pamieci programu
#include <avr/pgmspace.h>

#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))

//! Makropolecenie kasujace bity wywolania przerwan licznika 1.
#define CLEAR_ALL_TIMER1_INT_FLAGS    (TIFR1 = TIFR1)
// Wektor synchronizacji dla danych WSPR
const char SyncVec[162] = {
  1,1,0,0,0,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,1,0,1,1,1,1,0,0,0,0,0,0,0,1,0,0,1,0,1,0,0,0,0,0,0,1,0,
  1,1,0,0,1,1,0,1,0,0,0,1,1,0,1,0,0,0,0,1,1,0,1,0,1,0,1,0,1,0,0,1,0,0,1,0,1,1,0,0,0,1,1,0,1,0,1,0,
  0,0,1,0,0,0,0,0,1,0,0,1,0,0,1,1,1,0,1,1,0,0,1,1,0,1,0,0,0,1,1,1,0,0,0,0,0,1,0,1,0,0,1,1,0,0,0,0,
  0,0,0,1,1,0,1,0,1,1,0,0,0,1,1,0,0,0
```

```
};

// Tabela 256 probek sinusoidy zawarta w pamieci programu. Sinusoida jest przesunieta w gore o 128
// a wiec jej wartosci zerowej odpowiada 128, szczytom dodatnim 255 a szczytom ujemnym 0.
PROGMEM  prog_uchar sine256[]  = {
  127,130,133,136,139,143,146,149,152,155,158,161,164,167,170,173,176,178,181,184,187,190,192,
195,198,200,203,205,208,210,212,215,217,219,221,223,225,227,229,231,233,234,236,238,239,240,
  242,243,244,245,247,248,249,249,250,251,252,252,253,253,253,254,254,254,254,254,254,254,253,
253,253,252,252,251,250,249,249,248,247,245,244,243,242,240,239,238,236,234,233,231,229,227,
225,223,
  221,219,217,215,212,210,208,205,203,200,198,195,192,190,187,184,181,178,176,173,170,167,164,
161,158,155,152,149,146,143,139,136,133,130,127,124,121,118,115,111,108,105,102,99,96,93,90,87,
84,81,78,
  76,73,70,67,64,62,59,56,54,51,49,46,44,42,39,37,35,33,31,29,27,25,23,21,20,18,16,15,14,12,11,10,9,
7,6,5,5,4,3,2,2,1,1,1,0,0,0,0,0,0,0,1,1,1,2,2,3,4,5,5,6,7,9,10,11,12,14,15,16,18,20,21,23,25,27,29,31,
  33,35,37,39,42,44,46,49,51,54,56,59,62,64,67,70,73,76,78,81,84,87,90,93,96,99,102,105,108,111,
115,118,121,124

};

const unsigned long mt[4] = {  800974,801757,802540,803323 }; // Tabela slow sterujacych syntezer
                          // (skoków fazy) dla czestotliwosci WSPR 1497,8 1499,3 1500,7 1502,2 Hz
int ledPin = 13;              // dioda swiecaca na plytce - kontakt 13
int led2Pin = 8;              // dioda swiecaca - kontakt 8
int t1Pin = 4;                // wyjscie diagnostyczne dla podprogramow przerwan
int t2Pin = 5;                // wyjscie diagnostyczne dla programu glownego
int pttPin=10;                // kluczowanie (wlaczanie) nadajnika
int dcfPin=7;                 // wejscie impulsow z odbiornika DCF
int state=0;

// byte sine[258];            // tabela 8-bitowych wartosci probek sinusoidy
byte c[11];                   // zakodowany komunikat
byte sym[170];                // tabela 162 symboli strumienia
byte symt[170];               // tymczasowa tabela symboli
char call[] = "DH3JO ";  // znak, lokator i moc. Zmienic na wlasne
char locator[] = "JO30";
byte power = 20;
byte dcfPin_a;                // zapamietany poprzedni stan wejscia impulsow z odbiornika DCF
int dcfcnt;
int dcfmin;                   // czas DCF - minuty
int dcfhour;                  // czas DCF - godziny
byte n;
byte f_led;

unsigned long n1;          // zakodowany znak wywolawczy
unsigned long m1;          // zakodowany lokator

char cnt1;
unsigned char cc1;
int ii,bb;                    // indeksy do tabel i petli

// zmienne ulotne dla podprogramow przerwan
volatile boolean f_tone;              // sygnalizator zmiany tonow - odstep 1,46 Hz
volatile unsigned int tcnt;           // licznik czasu zmiany tonow
volatile unsigned long cnt16u;   // licznik czasu 16 us
```

```
volatile byte icnt;
volatile byte icnt2;
volatile unsigned long phaccu;  // akumulator fazy syntezera cyfrowego
volatile unsigned long mm;      // slowo sterujace dla nadawanej czestotliwosci (skok fazy)
volatile unsigned int sycnt;
volatile unsigned int cpin;

void setup()
{
 pinMode(ledPin, OUTPUT);     // Programowanie wyjsc i wejsc
 pinMode(led2Pin, OUTPUT);
 pinMode(t1Pin, OUTPUT);
 pinMode(t2Pin, OUTPUT);

 pinMode(pttPin, OUTPUT);    // wyjscie do kluczowania nadajnika
 pinMode(dcfPin, INPUT);      // wejscie impulsow z odbiornika DCF

 pinMode(11, OUTPUT);        // wyjscie PWM dla modulacji VFO
 pinMode(3, OUTPUT);         // wyjscie PWM tonow WSPR
 Serial.begin(115200);       // inicjalizacja zlacza szeregowego
 Serial.println("WSPR beacon"); // wyswietlenie informacji w monitorze szeregowym Arduino

 Setup_timer2();

 //cli();                     // wylaczenie przerwan dla unikniecia zaklocen w pracy
 //  cbi (TIMSK0,TOIE0);      // wylaczenie licznika 0 (Timer0) !!! funkcja delay() wylaczona
 sbi (TIMSK2,TOIE2);         // wlaczenie przerwan pochodzacych od licznika 2 (Timer2)

 OCR2B=64;                   // wydanie wartosci poczatkowej na wyjscia PWM
 OCR2A=64;

 mm=402063;

 encode_call();              // kodowanie znaku wywolawczego
 Serial.print("call: ");     // wydanie go na komputer
 Serial.print(call);
 Serial.print(" ");
 Serial.print(n1,HEX);
 Serial.println(" ");

 encode_locator();           // kodowanie lokatora
 Serial.print("locator: ");  // wydanie go na komputer
 Serial.print(locator);
 Serial.print(" ");
 Serial.print(m1 << 2,HEX);
 Serial.println(" ");

 for (bb=0;bb<=10;bb++){
  Serial.print(c[bb],HEX);
  Serial.print(",");
 }
 Serial.println("");
 encode_conv();

 Serial.println("");
```

```
  for (bb=0;bb<162 ;bb++){
    Serial.print(symt[bb],DEC);                    // wydanie zakodowanych symboli na komputer
    Serial.print(".");
    if ( (bb+1) %32 == 0) Serial.println("");
  }
  Serial.println("");

  interleave_sync();                               // wplecenie danych synchronizacyjnych

  for (bb=0;bb<162 ;bb++){
    Serial.print(sym[bb],DEC);
    Serial.print(".");
    if ( (bb+1) %32 == 0) Serial.println("");
  }
  Serial.println("");

  tcnt=0;
  mm=0;
  dcfmin=-1;
}

void loop()
{
  sycnt =0;
  state=10;
  while(1) {

    ii=dcf_minute();

    if ((ii >= 0) && ( ii % 10 ==0) || (ii-2) % 10 ==0 ){ // 4 minuty nadawania w odcinku 10-minutowym
      Serial.println("Wspr Start");            // komunikat dla operatora
      digitalWrite(pttPin,1);                  // wlaczenie nadajnika
      tcnt=0;
      sycnt =0;
      state=5;                                  // poczatek odcinka 2-minutowego = poczatkowi transmisji
    }

    if ((ii >= 0) &&  (ii-4) % 10 ==0 ){ // 6 minut pseudotransmisji (bez wlaczenia nadajnika) w odcinku
10-minutowym
      Serial.println("Wspr Start without ptt");  // komunikat dla operatora
      tcnt=0;
      sycnt =0;
      state=5;
    }

    if (f_tone==1) {
      f_tone=0;
      switch (state){            // wybor stanow (etapow pracy) programu
      case 0:
        break;

      case 5:                    // poczatek transmisji WSPR z pewnym opoznieniem
        if (sycnt==6) {
          sycnt=0;
```

```
        state=10;               // w nastepnym przebiegu petli stan transmisji

       }
       break;

     case 10:                  // transmisja Wspr w toku
       bb=sym[sycnt];
       mm=mt[bb];
       OCR2A=240-(40*bb);  // wydanie napiecia sterujacego ew. zewnetrzne (dodatkowe) VCO

       /*
       Serial.print(sycnt);
        Serial.print(" ");
        Serial.print(bb);
        */
       Serial.print("*");

       if (sycnt >= 162) state = 11;       // po nadaniu 162 symboli w nastepnym przebiegu petli stan
zakonczenia transmisji
       break;
     case 11:                   // koniec transmisji Wspr

       digitalWrite(pttPin,0);         // wylaczenie nadajnika
       break;
     }
    }

  }
} // loop
//**********************************************************************
// inicjalizacja licznika 2 (timer2)
// dzielnik wstepny z podzialem przez 1, tryb PWM „phase correct PWM”, czestotliwosc zegarowa
16000000/510 = 31372,55 Hz
void Setup_timer2() {

// Stosunek podzialu dla dzielnika wstepnego: 1
  sbi (TCCR2B, CS20);
  cbi (TCCR2B, CS21);
  cbi (TCCR2B, CS22);

  // Tryb PWM licznika 2 „Phase Correct PWM”
  cbi (TCCR2A, COM2A0);  // wyzerowanie sygnalizacji komparatora A
  sbi (TCCR2A, COM2A1);

  cbi (TCCR2A, COM2B0);  // wyzerowanie sygnalizacji komparatora B
  sbi (TCCR2A, COM2B1);

  sbi (TCCR2A, WGM20);  // Tryb 1  / „phase correct PWM”
  cbi (TCCR2A, WGM21);
  cbi (TCCR2B, WGM22);

}

//**********************************************************************
// Dekodowanie sygnalow z odbiornika DCF
```

```
int dcf_minute() {
  int bb;
  int bb2,bb3;
  int rr=-1;
  bb= digitalRead(dcfPin);              // odczyt wejscia DCF
  if (bb == 1 && dcfPin_a==0) {    // rozpoznawanie zmiany stanu wejscia 0 -> 1 - poczatku impulsu
    cpin=0;                              // przez porownanie ze stanem poprzednim - dcfPin_a
    digitalWrite(ledPin,f_led);
    digitalWrite(led2Pin,f_led);
    f_led= !f_led;
  }
  if (bb == 0 && dcfPin_a==1) {   // rozpoznawanie zmiany stanu wejscia 1 -> 0 – konca impulsu
    bb2=0;                               // przez porownanie ze stanem poprednim - dcfPin_a
    dcfcnt++;

    if (cpin > 30) bb2=1; // bb2 – wartosc bitu w zaleznosci od dlugosci impulsu, 100 ms – 0, 200 ms – 1

    if ( dcfcnt >  15 && dcfcnt < 29) {   // uwzgledniane impulsy sekund 15…28 – zakodowane minuty
      /*
      Serial.print(" S:");
       Serial.print(dcfcnt);
       Serial.print(" ");
       Serial.print(bb2);
       */
      Serial.print(".");

    }
// dekodowanie minut
    if ((dcfcnt==22)  && (bb2 == 1))   dcfmin +=1;
    if ((dcfcnt==23)  && (bb2 == 1))   dcfmin +=2;
    if ((dcfcnt==24)  && (bb2 == 1))   dcfmin +=4;
    if ((dcfcnt==25)  && (bb2 == 1))   dcfmin +=8;
    if ((dcfcnt==26)  && (bb2 == 1))   dcfmin +=10;
    if ((dcfcnt==27)  && (bb2 == 1))   dcfmin +=20;
    if ((dcfcnt==28)  && (bb2 == 1))   dcfmin +=40;
// dekodowanie godzin
    if ((dcfcnt==30)  && (bb2 == 1))   dcfhour +=1;
    if ((dcfcnt==31)  && (bb2 == 1))   dcfhour +=2;
    if ((dcfcnt==32)  && (bb2 == 1))   dcfhour +=4;
    if ((dcfcnt==33)  && (bb2 == 1))   dcfhour +=8;
    if ((dcfcnt==34)  && (bb2 == 1))   dcfhour +=10;
    if ((dcfcnt==35)  && (bb2 == 1))   dcfhour +=20;
  }

  if (cpin > 300) {
    dcfcnt=0;
    cpin=0;
    rr=dcfmin;
    Serial.print("DCF Time:");    // wydanie na komputer zdekodowanego czasu
    Serial.print(dcfhour);
    Serial.print(":");
    Serial.println(dcfmin);
    dcfmin=0;
    dcfhour=0;
  }
```

```
  dcfPin_a=bb;
  return(rr);
}

//**********************************************************************
void encode() {            // kodowanie pelnego komunikatu
  encode_call();
  encode_locator();
  encode_conv();
  interleave_sync();
};
//**********************************************************************
// normalizowanie znakow 0..9 A..Z odstep w kolejnosci 0..36
char chr_normf(char bc ) {
  char cc=36;
  if (bc >= '0' && bc <= '9') cc=bc-'0';
  if (bc >= 'A' && bc <= 'Z') cc=bc-'A'+10;
  if (bc == ' ' ) cc=36;

  return(cc);
}

//**********************************************************************
void encode_call(){                // kodowanie znaku wywolawczego
  unsigned long t1;

  // kodowanie znaku
  n1=chr_normf(call[0]);
  n1=n1*36+chr_normf(call[1]);
  n1=n1*10+chr_normf(call[2]);
  n1=n1*27+chr_normf(call[3])-10;
  n1=n1*27+chr_normf(call[4])-10;
  n1=n1*27+chr_normf(call[5])-10;

  // dopisanie zakodowanego znaku wywolawczego do tabeli komunikatu c[]
  t1=n1;
  c[0]= t1 >> 20;
  t1=n1;
  c[1]= t1 >> 12;
  t1=n1;
  c[2]= t1 >> 4;
  t1=n1;
  c[3]= t1 << 4;
}

//**********************************************************************
void encode_locator(){                    // kodowanie lokatora

  unsigned long t1;
  // kodowanie lokatora
  m1=179-10*(chr_normf(locator[0])-10)-chr_normf(locator[2]);
  m1=m1*180+10*(chr_normf(locator[1])-10)+chr_normf(locator[3]);
  m1=m1*128+power+64;

  // dopisanie zakodowanego lokatora i mocy do tabeli komunikatu c[]
```

```
  t1=m1;
  c[3]= c[3] + ( 0x0f & t1 >> 18);
  t1=m1;
  c[4]= t1 >> 10;
  t1=m1;
  c[5]= t1 >> 2;
  t1=m1;
  c[6]= t1 << 6;
}
//*********************************************************************
// splotowe kodowanie komunikatu z tabeli c[] do postaci 162-bitowego strumienia
void encode_conv(){
  int bc=0;
  int cnt=0;
  int cc;
  unsigned long sh1=0;

  cc=c[0];

  for (int i=0; i < 81;i++) {
    if (i % 8 == 0 ) {
      cc=c[bc];
      bc++;
    }
    if (cc & 0x80) sh1=sh1 | 1;

    symt[cnt++]=parity(sh1 & 0xF2D05351);
    symt[cnt++]=parity(sh1 & 0xE4613C47);

    cc=cc << 1;
    sh1=sh1 << 1;
  }

}

//*********************************************************************
byte parity(unsigned long li)    // obliczanie bajtu parzystosci
{
  byte po = 0;
  while(li != 0)
  {
    po++;
    li&= (li-1);
  }
  return (po & 1);
}
//*********************************************************************
// przeplot 162 bitow danych i dodanie wektora synchronizacji z tabeli
void interleave_sync(){
  int ii,ij,b2,bis,ip;
  ip=0;

  for (ii=0;ii<=255;ii++) {
    bis=1;
    ij=0;
```

```
   for (b2=0;b2 < 8 ;b2++) {
     if (ii & bis) ij= ij | (0x80 >> b2);
     bis=bis << 1;
   }
   if (ij < 162 ) {
     sym[ij]= SyncVec[ij] +2*symt[ip];
     ip++;
   }
  }
}

//*****************************************************************
// Podprogram przerwania powodowanego przez licznik 2 (Timer2) z czestotliwoscia 31372,550 kHz
czyli co 32 us
// podstawa czasu REFCLOCK dla syntezera cyfrowego
// FOUT = (M (REFCLK)) / (2 exp 32)
// czas wykonania : 8 mikrosekund (z uwzglednieniem rozkazow push i pop)
ISR(TIMER2_OVF_vect) {

 sbi(PORTD,4);       // wyskoki poziom na wyjsciu PORTD,4 dla obserwacji pracy na oscyloskopie
 //  cnt16u++;
 if (tcnt++ >= 21417) {
   tcnt=0;                        // sygnalizator odstepu tonow na 1,4648 Hz
   f_tone=1;
   sycnt++;
 }

 phaccu=phaccu+mm;       // 24-bitowy akumulator fazy syntezera cyfrowego
 icnt=phaccu >> 16;          // gorne 8 bitow adresuje pamiec probek sinusoidy
 OCR2B=sine256[icnt];     // wydanie wartosci probki na wyjscie analogowe PWM
 if(icnt2++ == 0) cpin++;

 cbi(PORTD,4);               // zerowanie wyjscia PORTD,4
}
```

Radiolatarnia Hella z kluczowaną podnośną

W przykładzie tym wykorzystano programowy syntezer podnośnej m.cz. z programów poprzednich i znany już program dalekopisowej radiolatarni Hella. Najważniejszą różnicą w stosunku do poprzedniego rozwiązania radiolatarni jest to, że w zależności od nadawanego elementu znaku kluczowana jest amplituda generowanej przez Arduino podnośnej m.cz. Dla unikniecia szkodliwych i mogących zakłócać pracę sąsiednich stacji stuków na początku i końcu elementu (analogicznie jak w przypadku telegrafii) zastosowano tutaj kluczowanie podnośnej (jej włączanie i wyłączanie) w zerze sinusoidy co wymaga aby w podstawowym elemencie – półpolu – mieściła się pełna liczba okresów podnośnej m.cz. Dla 4 okresów na półpole otrzymuje się częstotliwość podnośnej 980 Hz. W zależności od dokładnej częstotliwości kwarcu mogą tutaj wystąpić drobne odchyłki częstotliwości podnośnej – różnice są jednak niewielkie, najwyżej rzędu kilku Hz.
Dla pięciu okresów sinusoidy na półpole częstotliwość podnośnej wynosiłaby 1225 Hz a dla sześciu – 1470 Hz. Częstotliwość m.cz. jest stała a wszelkie polecenia pozwalające na jej regulację za pomocą potencjometru podłączonego do wejścia analogowego nr 0 (pochodzące z programu generatora) pozostawiono dla ułatwienia przy ewentualnym wprowadzaniu rozszerzeń.
Program pozwala też na kluczowanie nadajnika telegraficznego jak w poprzednim przykładzie. Konieczne jest włączenie filtru dolnoprzepustowego na wyjściu m.cz. Schemat filtru podano w opisie wyjściowego programu generatora m.cz.

Kod źródłowy programu

```
/* Radiolatarnia Hellduino (Mark VandeWettering K6HX; www.brainwagon.com) polaczona
 * z generatorem (cyfrowym syntezerem) podnosnej
 * kluczowanej w takt elementow pisma
 *
 * Polaczenie i dopasowanie obu programow
 * Krzysztof Dabrowski OE1KDA
 * 29.08.2013
 *
 * Przyjety staly raster czasowy dla nadawania
 * 127 x 32 us = 4064 us dla polelementu
 * dlugosc pelnego elementu 8128 us
 * znormalizowane dlugosci 4,081 i 8,163 ms
 * odchylka ok. 0,4 %
 * Raster narzucany przez przerwanie syntezera (licznik 2)
 * W porownaniu z programem syntezera dodano synchronizacje
 * warunkow poczatkowych przed kazda transmisja tekstu
 * podnosna 984 Hz (standardowo 980 Hz) - dopasowana do
 * rzeczywistych dlugosci elementow
 * zapewnienie kluczowania podnosnej w zerze lub w poblizu
 * 3.09.2013
 * Dodany wybor wyjscia analogowego 3 lub 11
 */

/*
 * Cyfrowy syntezer czestotliwosci na ATMEGA 168
 * Licznik 2 (Timer2) generuje przerwania z czestotliwoscia 31,250 kHz
 *
 * KHM 2009 /  Martin Nawrath
 * Kunsthochschule fuer Medien Koeln
 * Academy of Media Arts Cologne
*/

#include "avr/pgmspace.h"
```

```
// tabela 256 probek sinusoidy /pelny okres/ zawarta w pamieci programu
PROGMEM  prog_uchar sine256[]  = {
  127,130,133,136,139,143,146,149,152,155,158,161,164,167,170,173,176,178,181,184,187,190,192,
195,198,200,203,205,208,210,212,215,217,219,221,223,225,227,229,231,233,234,236,238,239,240,
  242,243,244,245,247,248,249,249,250,251,252,252,253,253,253,254,254,254,254,254,254,254,253,
253,253,252,252,251,250,249,249,248,247,245,244,243,242,240,239,238,236,234,233,231,229,227,
225,223,
  221,219,217,215,212,210,208,205,203,200,198,195,192,190,187,184,181,178,176,173,170,167,164,
161,158,155,152,149,146,143,139,136,133,130,127,124,121,118,115,111,108,105,102,99,96,93,90,87,
84,81,78,
  76,73,70,67,64,62,59,56,54,51,49,46,44,42,39,37,35,33,31,29,27,25,23,21,20,18,16,15,14,12,11,10,9,
7,6,5,5,4,3,2,2,1,1,1,0,0,0,0,0,0,0,1,1,1,2,2,3,4,5,5,6,7,9,10,11,12,14,15,16,18,20,21,23,25,27,29,31,
  33,35,37,39,42,44,46,49,51,54,56,59,62,64,67,70,73,76,78,81,84,87,90,93,96,99,102,105,108,111,
115,118,121,124

};
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
// wybor wyjscia 3 lub 11
#define SYG3
#undef SYG11

int ledPin = 13;                // dioda swiecaca na plytce
int testPin = 7;                // dioda swiecaca na wyjsciu 7
int t2Pin = 6;
byte bb;
//---- deklaracje z Hellduino
int radioPin = 10 ;          // kluczowanie nadajnika
typedef struct glyph {     // struktura zawierajaca znaki alfabetu i 7 jego kolumn
char ch ;
word col[7] ;
} Glyph ;                       // struktura dla znaku i jego elementow

Glyph glyphtab[] PROGMEM = {
{' ', {0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'A', {0x07fc, 0x0e60, 0x0c60, 0x0e60, 0x07fc, 0x0000, 0x0000}},
{'B', {0x0c0c, 0x0ffc, 0x0ccc, 0x0ccc, 0x0738, 0x0000, 0x0000}},
{'C', {0x0ffc, 0x0c0c, 0x0c0c, 0x0c0c, 0x0c0c, 0x0000, 0x0000}},
{'D', {0x0c0c, 0x0ffc, 0x0c0c, 0x0c0c, 0x07f8, 0x0000, 0x0000}},
{'E', {0x0ffc, 0x0ccc, 0x0ccc, 0x0c0c, 0x0c0c, 0x0000, 0x0000}},
{'F', {0x0ffc, 0x0cc0, 0x0cc0, 0x0c00, 0x0c00, 0x0000, 0x0000}},
{'G', {0x0ffc, 0x0c0c, 0x0c0c, 0x0ccc, 0x0cfc, 0x0000, 0x0000}},
{'H', {0x0ffc, 0x00c0, 0x00c0, 0x00c0, 0x0ffc, 0x0000, 0x0000}},
{'I', {0x0ffc, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'J', {0x003c, 0x000c, 0x000c, 0x000c, 0x0ffc, 0x0000, 0x0000}},
{'K', {0x0ffc, 0x00c0, 0x00e0, 0x0330, 0x0e1c, 0x0000, 0x0000}},
{'L', {0x0ffc, 0x000c, 0x000c, 0x000c, 0x000c, 0x0000, 0x0000}},
{'M', {0x0ffc, 0x0600, 0x0300, 0x0600, 0x0ffc, 0x0000, 0x0000}},
{'N', {0x0ffc, 0x0700, 0x01c0, 0x0070, 0x0ffc, 0x0000, 0x0000}},
{'O', {0x0ffc, 0x0c0c, 0x0c0c, 0x0c0c, 0x0ffc, 0x0000, 0x0000}},
{'P', {0x0c0c, 0x0ffc, 0x0ccc, 0x0cc0, 0x0780, 0x0000, 0x0000}},
{'Q', {0x0ffc, 0x0c0c, 0x0c3c, 0x0ffc, 0x000f, 0x0000, 0x0000}},
{'R', {0x0ffc, 0x0cc0, 0x0cc0, 0x0cf0, 0x079c, 0x0000, 0x0000}},
{'S', {0x078c, 0x0ccc, 0x0ccc, 0x0ccc, 0x0c78, 0x0000, 0x0000}},
```

```
{'T', {0x0c00, 0x0c00, 0x0ffc, 0x0c00, 0x0c00, 0x0000, 0x0000}},
{'U', {0x0ff8, 0x000c, 0x000c, 0x000c, 0x0ff8, 0x0000, 0x0000}},
{'V', {0x0ffc, 0x0038, 0x00e0, 0x0380, 0x0e00, 0x0000, 0x0000}},
{'W', {0x0ff8, 0x000c, 0x00f8, 0x000c, 0x0ff8, 0x0000, 0x0000}},
{'X', {0x0e1c, 0x0330, 0x01e0, 0x0330, 0x0e1c, 0x0000, 0x0000}},
{'Y', {0x0e00, 0x0380, 0x00fc, 0x0380, 0x0e00, 0x0000, 0x0000}},
{'Z', {0x0c1c, 0x0c7c, 0x0ccc, 0x0f8c, 0x0e0c, 0x0000, 0x0000}},
{'0', {0x07f8, 0x0c0c, 0x0c0c, 0x0c0c, 0x07f8, 0x0000, 0x0000}},
{'1', {0x0300, 0x0600, 0x0ffc, 0x0000, 0x0000, 0x0000, 0x0000}},
{'2', {0x061c, 0x0c3c, 0x0ccc, 0x078c, 0x000c, 0x0000, 0x0000}},
{'3', {0x0006, 0x1806, 0x198c, 0x1f98, 0x00f0, 0x0000, 0x0000}},
{'4', {0x1fe0, 0x0060, 0x0060, 0x0ffc, 0x0060, 0x0000, 0x0000}},
{'5', {0x000c, 0x000c, 0x1f8c, 0x1998, 0x18f0, 0x0000, 0x0000}},
{'6', {0x07fc, 0x0c66, 0x18c6, 0x00c6, 0x007c, 0x0000, 0x0000}},
{'7', {0x181c, 0x1870, 0x19c0, 0x1f00, 0x1c00, 0x0000, 0x0000}},
{'8', {0x0f3c, 0x19e6, 0x18c6, 0x19e6, 0x0f3c, 0x0000, 0x0000}},
{'9', {0x0f80, 0x18c6, 0x18cc, 0x1818, 0x0ff0, 0x0000, 0x0000}},
{'*', {0x018c, 0x0198, 0x0ff0, 0x0198, 0x018c, 0x0000, 0x0000}},
{'.', {0x001c, 0x001c, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'?', {0x1800, 0x1800, 0x19ce, 0x1f00, 0x0000, 0x0000, 0x0000}},
{'!', {0x1f9c, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'(', {0x01e0, 0x0738, 0x1c0e, 0x0000, 0x0000, 0x0000, 0x0000}},
{')', {0x1c0e, 0x0738, 0x01e0, 0x0000, 0x0000, 0x0000, 0x0000}},
{'#', {0x0330, 0x0ffc, 0x0330, 0x0ffc, 0x0330, 0x0000, 0x0000}},
{'$', {0x078c, 0x0ccc, 0x1ffe, 0x0ccc, 0x0c78, 0x0000, 0x0000}},
{'/', {0x001c, 0x0070, 0x01c0, 0x0700, 0x1c00, 0x0000, 0x0000}},
{'@', {0x1ff0, 0x3018, 0x670c, 0x678c, 0x3f38, 0x0000, 0x0000}},
} ;                                                  // tabela struktur znakow
#define NGLYPHS (sizeof(glyphtab)/sizeof(glyphtab[0]))
//---- koniec deklaracji z Hellduino
char amplituda;                 // amplituda podnosnej, stan odpowiada wyjsciu kluczujacemu
//---- dalej deklaracje generatora
double dfreq;
// const double refclk=31372.549;  // =16MHz / 510
const double refclk=31376.6;         // zmierzona czestotliwosc podstawy czasu

// zmienne ulotne uzywane w podprogramie przerwania
volatile byte icnt;                  // uzywana w podprogamie przerwania
volatile byte icnt1;                 // uzywana w podprogramie przerwania
volatile byte c4ms;                  // licznik czasu co 4,064 ms; polowa dlugosci elementu
volatile unsigned long phaccu;     // akumulator fazy
volatile unsigned long tword_m;  // slowo sterujace syntezer
//---- funkcje z Hellduino
void encodechar(int ch)              // kodowanie znaku
{
  int i, x, y, fch ;
  word fbits ;
/* Poszukiwanie dalej pomimo znalezienia znaku
* dla zapewnienia stalego czasu wykonywania
* ulatwia otrzymanie stalej dlugosci elementu
*/
  //Serial.print(ch);
  for (i=0; i<NGLYPHS; i++) {
    fch = pgm_read_byte(&glyphtab[i].ch) ;
      if (fch == ch) {
```

```
      for (x=0; x<7; x++) {
        fbits = pgm_read_word(&(glyphtab[i].col[x])) ;
        for (y=0; y<14; y++) {
          if (fbits & (1<<y))
            {digitalWrite(radioPin, HIGH) ;  // wyjscie kluczujace – poziom wysoki
             amplituda = 1;                  // wewnetrzne kluczowanie podnosnej
            }
          else
            {digitalWrite(radioPin, LOW) ;   // wyjscie kluczujace – poziom niski
             amplituda = 0;                  // wewnetrzne kluczowanie podnosnej
            }
//          delayMicroseconds(4045L) ;
          while (!c4ms)                      // oczekiwanie na koniec polelementu
            ;
          c4ms = 0;                          // ponownie 0 dla nastepnego polelementu
                        }
                    }
                }
        }
}
void encode(char *ch)
{
  while (*ch != '\0')
    encodechar(*ch++) ;
}

//---- koniec funkcji z Hellduino
void setup()
{
  pinMode(ledPin, OUTPUT);          // wyjscie logiczne
  Serial.begin(115200);             // inicjalizacja zlacza szeregowego
  Serial.println("DDS Test");

  pinMode(6, OUTPUT);               // wyjscie logiczne
  pinMode(7, OUTPUT);               // wyjscie logiczne
#ifdef SYG11
  pinMode(11, OUTPUT);              // wyjscie11 = PWM / m.cz.
#endif
#ifdef SYG3
  pinMode(3, OUTPUT);               // wy. 3 = PWM / m.cz.
#endif

  Setup_timer2();

  // wylaczenie przerwan dla unikniecia zaklocen w pracy
  cbi (TIMSK0,TOIE0);           // wylaczenie licznika 0 (Timer0) !!! funkcja delay() wylaczona
  sbi (TIMSK2,TOIE2);           // wlaczenie przerwan powodowanych przez licznik 2 (Timer2)

//  dfreq=1000.0;                 // czestotliwosc poczatkowa = 1000,0 Hz
  dfreq = 984.25;               // czestotliwosc podnosnej 984,25 Hz
                                // zeby kluczowanie wypadalo w w zerze sinusoidy
  tword_m=pow(2,32)*dfreq/refclk;  // obliczenie slowa sterujacego syntezer (skoku fazy)
  phaccu = 0;                   // poczatek od fazy zerowej
  pinMode(radioPin, OUTPUT) ;   // wyjscie kluczujace nadajnik
  digitalWrite(radioPin, LOW) ;  // wylaczenie nadajnika
```

```
  amplituda = 0;
}
void loop()
{
//  while(1) {
//     if (c4ms > 250) {             // sprawdzanie stanu licznika / oczekiwanie na uplyniecie pelnej sekundy
//      c4ms=0;
//      dfreq=analogRead(0);           // odczyt potencjometru na wejsciu analogowym 0 - strojenie w
zakresie 0..1023 Hz

//      cbi (TIMSK2,TOIE2);             // wylaczenie przerwania licznika 2 (Timer2)
//      tword_m=pow(2,32)*dfreq/refclk;  // obliczenie slowa sterujacego syntezer
//      sbi (TIMSK2,TOIE2);             // wlaczenie przerwania licznika 2 (Timer2)

//      Serial.print(dfreq);
//      Serial.print("  ");
//      Serial.println(tword_m);
//    }

   digitalWrite(radioPin, LOW) ;       // wylaczenie nadajnika
   cbi (TIMSK2,TOIE2);                 // wylaczenie przerwan licznika 2 (Timer2)
   intcnt1 = 0;                        // punkt wyjsciowy dla rastru czasowego
   c4ms = 0;
   OCR2A = 128;                           // wylaczenie podnosnej
   amplituda = 0;                      // zerowa amplituda podnosnej
   phaccu = 0;                         // synchronizacja fazy
   sbi (TIMSK2,TOIE2);                 // wlaczenie przerwania licznika 2 (Timer2)

   encode("OE1KDA JN88ED 500 MILIWATTS QSL: OE1KDA@OEVSV.AT ") ;

   sbi(PORTD,6); // Diagnoza / poziom wysoki na wyjsciu PORTD,7 dla obserwacji na oscyloskopie
   cbi(PORTD,6); // Diagnoza /poziom niski na wyjsciu  PORTD,7 dla obserwacji na oscyloskopie
  }
}
//*****************************************************************
// konfiguracja licznika 2 (timer2)
// dzielnik wstepny – podzial przez 1, tryb PWM „phase correct PWM”, czestotliwosc zegarowa
16000000/510 = 31372,55 Hz
void Setup_timer2() {

// Dzielnik wstepny dla licznika 2 – podzial przez 1
  sbi (TCCR2B, CS20);
  cbi (TCCR2B, CS21);
  cbi (TCCR2B, CS22);

  // tryb pracy PWM „Phase Correct PWM“
#ifdef SYG11
  cbi (TCCR2A, COM2A0);  // wyzerowanie sygnalizacji komparatora A
  sbi (TCCR2A, COM2A1);  // dla 11 - OCR2A
#endif
#ifdef SYG3
  cbi (TCCR2A, COM2B0);  // wyzerowanie sygnalizacji komparatora B
  sbi (TCCR2A, COM2B1);  // dla 3 - OCR2B
#endif
```

```
  sbi (TCCR2A, WGM20);  // Tryb 1  / „Phase Correct PWM”
  cbi (TCCR2A, WGM21);
  cbi (TCCR2B, WGM22);
}

//*****************************************************************
// Podprogram przerwan generowanych przez licznik 2 (Timer2) z czestotliwoscia 31372,550 kHz co
odpowiada 32 us
// podstawa czasu REFCLOCK dla syntezera cyfrowego
// FOUT = (M (REFCLK)) / (2 exp 32)
// czas wykonywania: 8 mikrosekund (wlacznie z rozkazami push i pop)
ISR(TIMER2_OVF_vect) {
  sbi(PORTD,7);          // Diagnoza / poziom wysoki na wyjsciu PORTD,7 dla obserwacji na
oscyloskopie

  phaccu=phaccu+tword_m; // 32-bitowy akumulator fazy syntezera
  icnt=phaccu >> 24;     // 8 wyzszych bitow uzywanych do adresowania tabeli probek sinusoidy
                         // odczyt wartosci z tabeli w celu wydania na przetwornik cyfrowo-analogowy
#ifdef SYG11
  if (amplituda == 1) OCR2A=pgm_read_byte_near(sine256 + icnt); else OCR2A = 127; // podnosna z
kluczowaniem amplitudy
#endif
#ifdef SYG3
  if (amplituda == 1) OCR2B=pgm_read_byte_near(sine256 + icnt); else OCR2B = 127; // podnosna z
kluczowaniem amplitudy
#endif

  if(icnt1++ == 127) {   // zliczanie w liczniku c4ms co 4,064 milliseconds
    c4ms++;              // dla 1 sekundy bylo to 125
    icnt1=0;
   }

 cbi(PORTD,7);           // zerowanie wyjscia PORTD,7
}
```

Radiolatarnia systemu Slowfeld

W systemie Slowfeld (= Slow Feldhell) w ciągu sekundy transmitowany jest najczęściej jeden element znaku, albo dwa lub trzy. Każdemu z wierszy znaku odpowiada inna częstotliwość wyjściowa a różnica częstotliwości pomiędzy wierszami musi być równa co najmniej szybkości transmisji elementów. Zasada ta ob owiązuje także we wszystkich pozostałych normach S/MT dalekopisów Hella. System ten jest stosowany w radiolatarniach bardzo małej mocy i jest swego rodzaju obrazowym odpowiednikiem QRSS. Nadawane znaki są odczytywane na wskaźniku wodospadowym programu odbiorczego np. Argo. Szczegóły dotyczące tego rodzaju emisji są zawarte w skryptach poświęconych technice słabych sygnałów.
W przytoczonym programie zastosowano szybkość transmisji równą jednemu półelementowi/sek. i odstęp częstotliwości między wierszami wynoszący 2 Hz.
Do generacji podnośnej kluczowanej częstotliwościowo m.cz. wykorzystano znany już z poprzednich przykładów program cyfrowego syntezera m.cz. a do jej kluczowania odpowiednio zmodyfikowany program dla transmisji w normie Feldhell. Analogicznie jak poprzednio znaki alfabetu są podzielone na 14 x 7 elementów.
Podobnie jak w poprzednim programie wprowadzono tutaj możliwość wyboru wyjścia 3 lub 11 dla sygnału m.cz. Konieczne jest włączenie filtru dolnoprzepustowego na wyjściu m.cz. Schemat filtru podano w opisie wyjściowego programu generatora m.cz.
Oprócz normy Slowfeld istnieje także grupa norm Slowhell, w której transmisja danych odpowiada normie Feldhell ale odbywa się z szybkością kilkakrotnie (np. 8-krotnie) mniejszą. Programowa realizacja emiesji Slowhell na Arduino nie powinna więc przysporzyć większych trudności. Sprawę tą pozostawiamy czytelnikom skryptu jako ćwiczenie praktyczne.
Podobnie opierając się na poprzednich i na obecnym przykładzie czytelnicy mogą stosunkowo łatwo zrealizować systemy transmisji Hella z kluczowaniem częstotliwości – FSK-Hell lub FM-Hell (= MSK-Hell).
W pierwszym przypadku czarnym elementom obrazu odpowiada najczęściej częstotliwość 980 Hz a białym o 245 Hz wyższa czyli 1225 Hz a w drugim elementom białym odpowiada częstotliwość 980 Hz a czarnym o 122,5 Hz niższa. Norma FSK-Hell jest w praktyce bardzo rzadko używana natomast FM-Hell nadaje się bardzo dobrze do łączności DX-owych.

Kod źródłowy programu

```
/* Radiolatarnia Hellduino (Mark VandeWettering K6HX; www.brainwagon.com) polaczona
 * z generatorem (cyfrowym syntezerem) podnosnej
 * kluczowanej w takt elementow pisma
 *
 * Polaczenie i dopasowanie obu programow
 * Krzysztof Dabrowski OE1KDA
 * 29.08.2013
 *
 * Dla normy Feldhell przyjety staly raster czasowy dla nadawania
 * 127 x 32 us = 4064 us dla polelementu
 * dlugosc elementu 8128 us
 * znormalizowane dlugosci 4,081 i 8,163 ms
 * odchylka ok. 0,4 %
 * Raster narzucany przez przerwanie syntezera (licznik 2)
 * W porownaniu z programem syntezera dodano synchronizacje
 * warunkow poczatkowych przed kazda transmisja tekstu
 * podnosna 984 Hz (standardowo 980 Hz) - dopasowana do
 * rzeczywistych dlugosci elementow
 * zapewnienie kluczowania podnosnej w zerze lub w poblizu
 * 3.09.2013
 * Dodany wybor wyjscia 3 lub 11
 */
```

```
/* 2.09.2013
 * Dostosowanie do normy Slowfeld
 * Dla normy Slowfeld polelement - 1/2 sekundy
 * Czestotliwosc podstawowa 1000 Hz
 * roznica czestotliwosci miedzy wierszami 2 Hz
 */

/*
* Syntezer cyfrowy na ATMEGA 168
 * Licznik 2 (Timer2) generuje przerwania z częstotliwością 31250 kHz
 *
 * KHM 2009 /  Martin Nawrath
 * Kunsthochschule fuer Medien Koeln
 * Academy of Media Arts Cologne
*/

#include "avr/pgmspace.h"

// tabela 256 probek sinusoidy /pełny okres/ w pamięci programu
PROGMEM  prog_uchar sine256[]  = {
  127,130,133,136,139,143,146,149,152,155,158,161,164,167,170,173,176,178,181,184,187,190,192,
195,198,200,203,205,208,210,212,215,217,219,221,223,225,227,229,231,233,234,236,238,239,240,
  242,243,244,245,247,248,249,249,250,251,252,252,253,253,253,254,254,254,254,254,254,254,253,
253,253,252,252,251,250,249,249,248,247,245,244,243,242,240,239,238,236,234,233,231,229,227,
225,223,
  221,219,217,215,212,210,208,205,203,200,198,195,192,190,187,184,181,178,176,173,170,167,164,
161,158,155,152,149,146,143,139,136,133,130,127,124,121,118,115,111,108,105,102,99,96,93,90,87,
84,81,78,
  76,73,70,67,64,62,59,56,54,51,49,46,44,42,39,37,35,33,31,29,27,25,23,21,20,18,16,15,14,12,11,10,9,
7,6,5,5,4,3,2,2,1,1,1,0,0,0,0,0,0,0,0,1,1,1,2,2,3,4,5,5,6,7,9,10,11,12,14,15,16,18,20,21,23,25,27,29,31,
  33,35,37,39,42,44,46,49,51,54,56,59,62,64,67,70,73,76,78,81,84,87,90,93,96,99,102,105,108,111,
115,118,121,124

};
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
// wybor wyjscia 3 lub 11
#define SYG3
#undef SYG11

int ledPin = 13;                // dioda swiecaca na plytce
int testPin = 7;                // wyjscie diagnostyczne
int t2Pin = 6;
byte bb;
//---- deklaracje z Hellduino
int radioPin = 10 ;             // kluczowanie nadajnika
typedef struct glyph {       // struktura dla znakow alfabetu i ich rozlozenia na kolumny
char ch ;
word col[7] ;
} Glyph ;

Glyph glyphtab[] PROGMEM = {
{' ', {0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'A', {0x07fc, 0x0e60, 0x0c60, 0x0e60, 0x07fc, 0x0000, 0x0000}},
```

```
{'B', {0x0c0c, 0x0ffc, 0x0ccc, 0x0ccc, 0x0738, 0x0000, 0x0000}},
{'C', {0x0ffc, 0x0c0c, 0x0c0c, 0x0c0c, 0x0c0c, 0x0000, 0x0000}},
{'D', {0x0c0c, 0x0ffc, 0x0c0c, 0x0c0c, 0x07f8, 0x0000, 0x0000}},
{'E', {0x0ffc, 0x0ccc, 0x0ccc, 0x0c0c, 0x0c0c, 0x0000, 0x0000}},
{'F', {0x0ffc, 0x0cc0, 0x0cc0, 0x0c00, 0x0c00, 0x0000, 0x0000}},
{'G', {0x0ffc, 0x0c0c, 0x0c0c, 0x0ccc, 0x0cfc, 0x0000, 0x0000}},
{'H', {0x0ffc, 0x00c0, 0x00c0, 0x00c0, 0x0ffc, 0x0000, 0x0000}},
{'I', {0x0ffc, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'J', {0x003c, 0x000c, 0x000c, 0x000c, 0x0ffc, 0x0000, 0x0000}},
{'K', {0x0ffc, 0x00c0, 0x00e0, 0x0330, 0x0e1c, 0x0000, 0x0000}},
{'L', {0x0ffc, 0x000c, 0x000c, 0x000c, 0x000c, 0x0000, 0x0000}},
{'M', {0x0ffc, 0x0600, 0x0300, 0x0600, 0x0ffc, 0x0000, 0x0000}},
{'N', {0x0ffc, 0x0700, 0x01c0, 0x0070, 0x0ffc, 0x0000, 0x0000}},
{'O', {0x0ffc, 0x0c0c, 0x0c0c, 0x0c0c, 0x0ffc, 0x0000, 0x0000}},
{'P', {0x0c0c, 0x0ffc, 0x0ccc, 0x0cc0, 0x0780, 0x0000, 0x0000}},
{'Q', {0x0ffc, 0x0c0c, 0x0c3c, 0x0ffc, 0x000f, 0x0000, 0x0000}},
{'R', {0x0ffc, 0x0cc0, 0x0cc0, 0x0cf0, 0x079c, 0x0000, 0x0000}},
{'S', {0x078c, 0x0ccc, 0x0ccc, 0x0ccc, 0x0c78, 0x0000, 0x0000}},
{'T', {0x0c00, 0x0c00, 0x0ffc, 0x0c00, 0x0c00, 0x0000, 0x0000}},
{'U', {0x0ff8, 0x000c, 0x000c, 0x000c, 0x0ff8, 0x0000, 0x0000}},
{'V', {0x0ffc, 0x0038, 0x00e0, 0x0380, 0x0e00, 0x0000, 0x0000}},
{'W', {0x0ff8, 0x000c, 0x00f8, 0x000c, 0x0ff8, 0x0000, 0x0000}},
{'X', {0x0e1c, 0x0330, 0x01e0, 0x0330, 0x0e1c, 0x0000, 0x0000}},
{'Y', {0x0e00, 0x0380, 0x00fc, 0x0380, 0x0e00, 0x0000, 0x0000}},
{'Z', {0x0c1c, 0x0c7c, 0x0ccc, 0x0f8c, 0x0e0c, 0x0000, 0x0000}},
{'0', {0x07f8, 0x0c0c, 0x0c0c, 0x0c0c, 0x07f8, 0x0000, 0x0000}},
{'1', {0x0300, 0x0600, 0x0ffc, 0x0000, 0x0000, 0x0000, 0x0000}},
{'2', {0x061c, 0x0c3c, 0x0ccc, 0x078c, 0x000c, 0x0000, 0x0000}},
{'3', {0x0006, 0x1806, 0x198c, 0x1f98, 0x00f0, 0x0000, 0x0000}},
{'4', {0x1fe0, 0x0060, 0x0060, 0x0ffc, 0x0060, 0x0000, 0x0000}},
{'5', {0x000c, 0x000c, 0x1f8c, 0x1998, 0x18f0, 0x0000, 0x0000}},
{'6', {0x07fc, 0x0c66, 0x18c6, 0x00c6, 0x007c, 0x0000, 0x0000}},
{'7', {0x181c, 0x1870, 0x19c0, 0x1f00, 0x1c00, 0x0000, 0x0000}},
{'8', {0x0f3c, 0x19e6, 0x18c6, 0x19e6, 0x0f3c, 0x0000, 0x0000}},
{'9', {0x0f80, 0x18c6, 0x18cc, 0x1818, 0x0ff0, 0x0000, 0x0000}},
{'*', {0x018c, 0x0198, 0x0ff0, 0x0198, 0x018c, 0x0000, 0x0000}},
{'.', {0x001c, 0x001c, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'?', {0x1800, 0x1800, 0x19ce, 0x1f00, 0x0000, 0x0000, 0x0000}},
{'!', {0x1f9c, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'(', {0x01e0, 0x0738, 0x1c0e, 0x0000, 0x0000, 0x0000, 0x0000}},
{')', {0x1c0e, 0x0738, 0x01e0, 0x0000, 0x0000, 0x0000, 0x0000}},
{'#', {0x0330, 0x0ffc, 0x0330, 0x0ffc, 0x0330, 0x0000, 0x0000}},
{'$', {0x078c, 0x0ccc, 0x1ffe, 0x0ccc, 0x0c78, 0x0000, 0x0000}},
{'/', {0x001c, 0x0070, 0x01c0, 0x0700, 0x1c00, 0x0000, 0x0000}},
{'@', {0x1ff0, 0x3018, 0x670c, 0x678c, 0x3f38, 0x0000, 0x0000}},
} ;                                                    // tabela struktur znakow
#define NGLYPHS (sizeof(glyphtab)/sizeof(glyphtab[0]))
//---- koniec deklaracji z Hellduino
char amplituda;                    // amplituda podnosnej, stan odpowiada wyjsciu kluczujacemu
//---- dalej deklaracje generatora
double dfreq;
double deltafr;
// const double refclk=31372.549;  // =16MHz / 510
const double refclk=31376.6;       // zmierzona czestotliwosc podstawy czasu
```

```
// zmienne ulotne uzywane w podprogramie przerwania
volatile byte icnt;                    // uzywana w podprogramie przerwania
volatile byte icnt1;                   // uzywana w podprogramie przerwania
volatile byte c4ms;                    // licznik zliczajacy 4,064 ms; polowa dlugosci elementu (polpole)
volatile unsigned long phaccu;     // akumulator fazy
volatile unsigned long tword_m;   // slowo sterujace syntezer (skok fazy)
unsigned long tword[14];             // tabela dla kolejnych czestotliwosci (odpowiadajacych wierszom)
//---- funkcje z Hellduino
void encodechar(int ch)
{
  int i, x, y, fch ;
  word fbits ;
/* Poszukiwanie kontynuowane po znalezieniu znaku
* dla zapewnienie stalego czasu wykonywania tej czesci programu
* i dzieki temu dlugosci elementu znaku
*/
  //Serial.print(ch);
  for (i=0; i<NGLYPHS; i++) {
    fch = pgm_read_byte(&glyphtab[i].ch) ;
      if (fch == ch) {
        for (x=0; x<7; x++) {
         fbits = pgm_read_word(&(glyphtab[i].col[x])) ;
          for (y=0; y<14; y++) {
            tword_m = tword[y];                  //czestotliwosc wiersza

            if (fbits & (1<<y))
              {digitalWrite(radioPin, HIGH) ;  // wyjscie kluczujace – stan wysoki
               amplituda = 1;                        // wewnetrzne kluczowanie podnosnej
              }
            else
              {digitalWrite(radioPin, LOW) ;     // wyjscie kluczujace – stan niski
               amplituda = 0;                        // wewnetrzne kluczowanie podnosnej
              }
//          delayMicroseconds(4045L) ;
          while (c4ms < 125)                    // oczekiwanie na koniec polelementu (1/2 sekundy)
            ;
          c4ms = 0;                                 // ponownie 0 dla nastepnego polelementu
                        }
                    }
                }
        }
}
void encode(char *ch)
{
  while (*ch != '\0')
    encodechar(*ch++) ;
}

//---- koniec funkcji z Hellduino
void setup()
{
  pinMode(ledPin, OUTPUT);          // wyjscie logiczne
  Serial.begin(115200);                // inicjalizacja zlacza szeregowego
  Serial.println("DDS Test");          // komunikat dla uzytkownika
```

```
  pinMode(6, OUTPUT);                 // wyjscie logiczne
  pinMode(7, OUTPUT);                 // wyjscie logiczne
#ifdef SYG11
  pinMode(11, OUTPUT);                // wyjscie11= PWM / podnosna m.cz.
#endif
#ifdef SYG3
  pinMode(3, OUTPUT);                 // wy. 3 = PWM / m.cz.
#endif

  Setup_timer2();

  // wylaczenie przerwan dla unikniecia zaklocen w pracy programu
  cbi (TIMSK0,TOIE0);              // wylaczenie licznika 0 (Timer0) !!! funkcja delay() wylaczona
  sbi (TIMSK2,TOIE2);              // wlaczenie przerwan pochodzacych od licznika 2 (Timer2)

  deltafr = 2.0;                   // roznica czestotliwosci miedzy wierszami
  dfreq = 1000.0;                  // czestotliwosc poczatkowa = 1000,0 Hz
  for (i = 0; i < 14; i++)
   {
    tword[i] = pow(2,32)*dfreq/refclk;
    dfreq = dfreq + deltafr;          // czestotliwosci dla poszczegolnych wierszy
   }
  tword_m = tword[0];

  //dfreq = 984.25;                  // czestotliwosc podnosnej 984,25 Hz
                                     // zeby kluczowanie wypadalo w w zerze
  // tword_m=pow(2,32)*dfreq/refclk;  // obliczenie nowego slowa sterujacego syntezer

  phaccu = 0;                        // poczatek od fazy zerowej
  pinMode(radioPin, OUTPUT) ;      // wyjscie kluczujace nadajnik
  digitalWrite(radioPin, LOW) ;    // wylaczenie nadajnika
  amplituda = 0;

}
void loop()
{
//  while(1) {
//     if (c4ms > 250) {                  // licznik czasu / oczekiwanie na uplyniecie pelnej sekundy
//      c4ms=0;
//      dfreq=analogRead(0);            // odczyt potencjometru na wejsciu analogowym 0 w celu strojenia
czestotliwosci w zakresie 0..1023 Hz

//      cbi (TIMSK2,TOIE2);             // wylaczenie przerwania generowanego przez licznik 2 (Timer2)
//      tword_m=pow(2,32)*dfreq/refclk;  // obliczenie nowego slowa sterujacego dla syntezera (skoku
fazy)
//      sbi (TIMSK2,TOIE2);             // wlaczenie przerwan generowanych przez licznik 2 (Timer2)

//      Serial.print(dfreq);
//      Serial.print("  ");
//      Serial.println(tword_m);
//    }

   digitalWrite(radioPin, LOW) ;        // wylaczenie nadajnika
   cbi (TIMSK2,TOIE2);                  // wylaczenie przerwan licznika 2 (Timer2)
   intcnt1 = 0;                         // punkt wyjsciowy dla rastru czasowego
```

```
   c4ms = 0;
   OCR2A = 128;                    // wylaczenie podnosnej
   amplituda = 0;                  // zerowa amplituda podnosnej
   phaccu = 0;                     // synchronizacja fazy
   sbi (TIMSK2,TOIE2);             // wlaczenie przerwania licznika 2 (Timer2)

   encode("OE1KDA JN88ED 500 MILIWATTS QSL: OE1KDA@OEVSV.AT ") ;

   sbi(PORTD,6); // diagnoza / poziom wysoki na wyjsciu PORTD,7 dla obserwacji na oscyloskopie
   cbi(PORTD,6); // diagnoza / poziom niski na wyjsciu PORTD,7 dla obserwacji n aoscyloskopie
  }
}
//*********************************************************************
// inicjalizacja licznika timer2
// dzielnik wstepny – podzial przez 1, tryb PWM „phase correct PWM”, czestotliwosc zegarowa
16000000/510 = 31372,55 Hz
void Setup_timer2() {

// dzielnik wstepny dla licznika Timer2 – podzial przez 1
  sbi (TCCR2B, CS20);
  cbi (TCCR2B, CS21);
  cbi (TCCR2B, CS22);

  // tryb PWM „Phase Correct PWM“
#ifdef SYG11
  cbi (TCCR2A, COM2A0);  // zerowanie sygnalizacji komparatora A
  sbi (TCCR2A, COM2A1);  // dla 11 - OCR2A
#endif
#ifdef SYG3
  cbi (TCCR2A, COM2B0);  // zerowanie sygnalizacji komparatora B
  sbi (TCCR2A, COM2B1);  // dla 3 - OCR2B
#endif

  sbi (TCCR2A, WGM20);  // tryb 1 / „Phase Correct PWM”
  cbi (TCCR2A, WGM21);
  cbi (TCCR2B, WGM22);
}

//*********************************************************************
// przerwanie generowane przez licznik 2 (Timer2) z czestotliwoscia 31372,550 Hz  czyli co 32 us
// podstawa czasu REFCLOCK dka syntezera cyfrowego
// FOUT = (M (REFCLK)) / (2 exp 32)
// czas wykonywania: 8 mikrosekund (wlacznie z rozkazami push i pop)
ISR(TIMER2_OVF_vect) {

  sbi(PORTD,7);         // diagnoza / poziom wysoki na wyjsciu PORTD,7 dla obserwacji na
oscyloskopie

  phaccu=phaccu+tword_m; // 32-bitowy akumulator fazy
  icnt=phaccu >> 24;            // gornych 8 bitow do adresowania tabeli probek
                                // odczyt wartosci z tabeli i wydanie na przetwornik c-a
#ifdef SYG11
  if (amplituda == 1) OCR2A=pgm_read_byte_near(sine256 + icnt); else OCR2A = 127; // podnosna z
kluczowaniem amplitudy
#endif
```

```
#ifdef SYG3
  if (amplituda == 1) OCR2B=pgm_read_byte_near(sine256 + icnt); else OCR2B = 127; // podnosna z
kluczowaniem amplitudy
#endif

  if(icnt1++ == 125) {   // licznik c4ms zlicza co 4 millisekundy
    c4ms++;              // dla 1 sekundy bylo to 125
    icnt1=0;
   }

 cbi(PORTD,7);           // zerowanie wyjscia diagnostycznego PORTD,7
}
```

Odbiorcza bramka internetowa

Odbiorcza bramka radiowo-internetowa (*RxGate, Air Gate, iGate*) przekazuje odebrane przez radio (najczęściej na częstotliwości 144,800 MHz) pakiety APRS do serwera sieci APRS-IS gdzie zostają one wyświetlone na mapach.
Najczęściej używanymi do tego celu serwerami APRS-IS są:

- igates.aprs.fi,
- finland.aprs2.net,
- rotate.aprs.net.

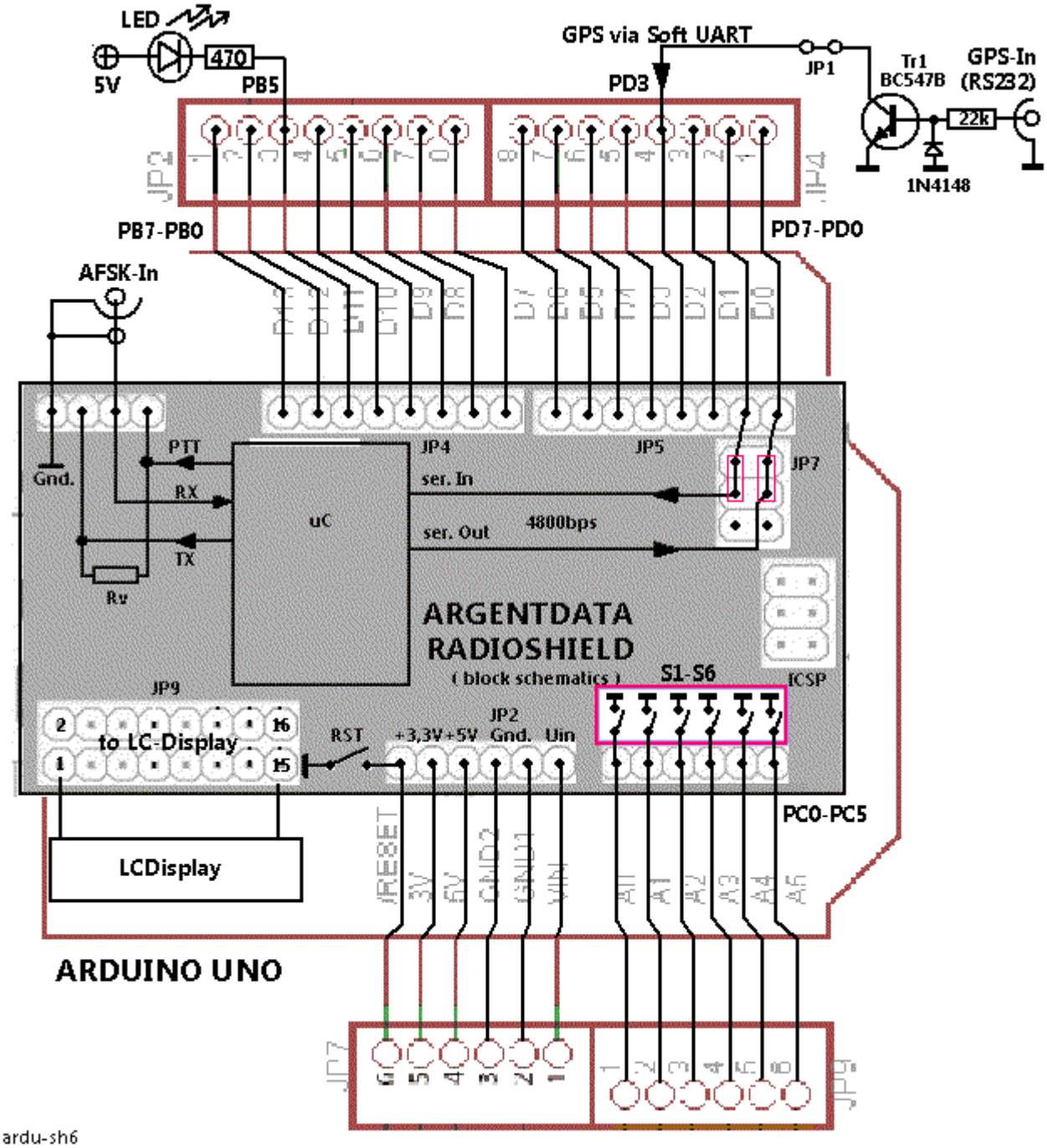
Dostęp do serwerów odbywa się najczęściej w kanale TCP 14580 ale możliwe jest też użycie protokółu UDP. W trakcie meldowania się na serwerze należy podać jako nazwę użytkownika własny znak wywoławczy i obliczone na jego podstawie hasło dostępu. Algorytm obliczania hasła nie jest oficjalnie publikowany i można go jedynie odczytać z niektórych (dostępnych także w internecie) programów korzystających z tych serwerów ale na szczęście w internecie istnieją co najmniej dwie strony obliczające wg tego algorytmu wymagane hasło po podaniu na nich znaku wywoławczego. Strony te dostępne są pod adresami:
http://www.george-smart.co.uk/wiki/APRS_Callpass
http://apps.magicbug.co.uk/passcode/

Sprawdzenie prawidłowości funkcjonowania połączenia można przeprowadzić nawiązując połączenie telnetowe z jednym z podanych serwerów korzystając z kanału logicznego 14580 zamiast ze standardowego kanału Telnetu – 23. Do tego celu nadaje sie dowolny telnetowy program terminalowy j.np. dostępny bezpłatnie Putty. Jako nazwę użytkownika należy i w tym przypadku podać własny znak wywoławczy a jako hasło dostępu hasło obliczone w jednej z podanych powyżej witryn. Bardziej doświadczeni użytkownicy mogą nawet w ramach diagnozy wpisać ręcznie w programie terminalowym komunikat APRS a następnie sprawdzić czy jest on widoczny w sieci (np. na mapach pod adresem *aprs.fi*).
W skład wyposażenia bramki oprócz Arduino wchodzi moduł rozszerzeń ethernetowy (jak w poniższym przykładzie) lub WiFi. Moduły te kontaktują się odpowiednio z domowym punktem dostępowym do internetu (ang. *router*) za pomocą kabla ethernetowego lub bezprzewodowo. Oprócz tego konieczny jest odbiornik FM na pasmo 2 m – może to być po prostu nie używana do innych celów radiostacja na to pasmo. Jeżeli radiostacja nie jest wyposażona we własny TNC konieczny jest też dodatkowy modem TNC lub moduł dodatkowy zawierający TNC (występujący pod nazwą *radio shield*) np. moduł firmy ArgentData. We wbudowany modem TNC do APRS wyposażone są m.in. modele TH-D7E, TH-D72E, TM-D700E, TM-D710E, VX-8DE, VX-8GE i FTM-350AE.
Podstawową częstotliwością pracy radiowej sieci APRS jest 144,800 MHz ale w rejonach o dużym natężeniu ruchu korzytne może być uruchamianie pomocniczych stacji (w tym także tego typu odbiorczych bramek internetowych) w paśmie 70 cm. Najczęściej stosowane są w nim simpleksowe częstotliwości 432,500 lub 433,800 MHz. Dawniej zamiast 432,500 MHz w użyciu była częstotliwość 430,800 MHz. W obu pasmach stosowana jest szybkość transmisji 1200 bodów. Transmisja APRS przez Międzynarodową Stację Kosmiczną odbywa się na częstotliwości 144,825 MHz.
Ponieważ biblioteki obsługujące oba moduły rozszerzeń są ze sobą blisko spokrewnione i nazwy odpowiadających sobie funkcji są również zbliżone do siebie dostosowanie poniższego przykładu do dostępu bezprzewodowego nie powinno nastręczyć zbytnich trudności bardziej doświadczonym programistom.
Komunikaty APRS transmitowane do serwera internetowego muszą mieć format tekstowy zwany również formatem TNC-2:
ŹRÓDŁO-1>CEL-2, TRASA*, TRASA2: dane użytkowe.

W retransmisji komunikatów do serwera internetowego obowiązują następujące zasady:

- Komunikaty zawierające w trasach specyfikacje TCPIP lub ogólnie TCPXX nie są retransmitowane,
- Komunikaty zawierające w trasach specyfikacje RFONLY lub NOGATE również nie są retransmitowane,
- Nie podlegają retransmisji komunikaty, w których dane użytkowe rozpoczynają się od znaku zapytania (zapytania sieciowe),

- W trasie retransmisji pakietów kierowanych do serwera APRS-IS dodawane są na końcu (przed dwukropkiem) informacje ”,qAR,ZNAK_IGATE-ROZSZERZENIE”,
- Elementy trasy zakończone gwiazdką nie podlegają żadnym modyfikacjom,
- Bardziej rozbudowane bramki mogą prowadzić krótkoterminową rejestrację retransmitowanych komunikatów aby nie retransmitować do sieci ich powtórzeń,
- Wszystkie pozostałe komunikaty mogą być retransmitowane do serwera,
- Zakończeniem komunikatu przekazywanego do serwera APRS-IS jest zawsze para znaków CR/LF; jeżeli odbierane przez radio komunikaty zawierają na końcu tylko jeden z tych znaków należy go usunąć i dodać obowiązkowo parę CR/LF.



Rys. 9.1. TNC dla Arduino – moduł „Radioshield” firmy „Argentdata”. Schemat połączeń z Arduino, radiostacją i odbiornikiem GPS.

Kod źródłowy bramki bez wyświetlacza i pamięci dodatkowej

```
// Airgate TCP – czysto odbiorcza bramka iGate
// retransmituje pakiety APRS z TNC „RadioShield” w protokole TCP do sieci APRS-IS
// Prawa autorskie 2012 Markus Heller, DL8RDS
// Patrz LICENSE.txt

#include <Arduino.h>
#include <SPI.h>
#include <Ethernet.h>

// Zmienic znak, parametry sieci i inne parametry konfiguracyjne w pliku config.h
#include "config.h"

#define BUFFERSIZE 260
char buf[BUFFERSIZE+1];
int buflen = 0;
boolean bad_packet = false;

EthernetClient client;                          // nowy obiekt typu klient ethernetowy

void setup() {
  Serial.begin(4800);                           // inicjalizacja zlacza szeregowego
  display("AIRGATE - DL8RDS");                  // wyswietlenie meldunku

#ifdef USE_DHCP
  Ethernet.begin(mac);                          // nawiazanie polaczenia ethernetowego
#else
  Ethernet.begin(mac, ip, gateway, subnet);
  delay(1000);
#endif
}

void loop() {
  // polacz lub wznow polaczenie z bramka APRS
  boolean connected = check_connection();

  // Odczyt i nadawanie danych z bufora nadawczego
  while (Serial.available() > 0) {
    char inbyte = Serial.read();
    if (inbyte == '\n') {
      // koniec linii: wyswietlenie i nadanie pakietu
      buf[buflen] = 0;
      if (! bad_packet && buflen != 0) {
        if (connected) send_packet();
        display_packet();
      }
      // zerowanie bufora pakietow
      buflen = 0;
      bad_packet = false;
    } else if ((inbyte < 31 && inbyte != 0x1c && inbyte != 0x1d && inbyte != 0x27)) {
      // ignorowanie blednie zdekodowanych pakietow ale dopuszczanie pakietow w formacie MIC-E
    } else if (! bad_packet && buflen != BUFFERSIZE) {
      // Jesli jest miejsce w buforze dopisanie danych.
```

```
      buf[buflen++] = inbyte;
    } else {
      display("Data Too Long");                    // meldunek o przepelnieniu bufora odbiorczego
      // Sytuacja trudna po zapelnieniu bufora przed odebraniem EOL
      // pakiet zaznaczony jako bledny i przejscie do odbioru nastepnego.
      bad_packet = true;
    }
  }

  // W przypadku polaczenia z serwerem APRS-IS odbior i wyswietlanie odpowiedzi serwera APRS-IS.
  // bufor 80-znakowy, wyswietlanie pojedynczych znakow powolne.
  if (connected) {
    receive_data();
  }
}

// Patrz http://www.aprs-is.net/Connecting.aspx
boolean check_connection() {
  if (!client.connected()) {
    display("Connecting...");
    if (client.connect(aprsgate, aprsport)) {                                    // po nawiazaniu polaczenia
      client.println("user " CALLSIGN " pass " PASSCODE " vers " VERSION);      // zameldowanie na
serwerze, znak uzytkownika, haslo obliczone na jego podstawie (patrz opis na poczatku rozdzialu),
wersja programu nie obowiazkowa, podawana dowolnie
      // nadanie raportu ze wspolrzednymi
      client.println(CALLSIGN ">" DESTINATION ":!" LATITUDE "I" LONGITUDE "&" PHG "/"
INFO);
      display("Connected");
    } else {                                                 // proba polaczenia zakonczona niepowodzeniem
      display("Failed");
      delay(1000);              // w przypadku braku polaczenia odczekanie sekundy przed nastepna proba.
    }
  }
  return client.connected();
}

void receive_data() {               // odbior danych
  if (client.available()) {
    char rbuf[81];
    int i = 0;
    while (i < 80 && client.available()) {
      char c = client.read();
      rbuf[i++] = c;
      if (c == '\n') break;
    }
    rbuf[i] = 0;
    display(rbuf);
  }
}

void send_packet() {           // nadanie pakietu
  client.println(buf);
  client.println();
}
```

```
void display_packet() {              // wyswietleni epakietu
  display(buf);
}

void display(char *msg) {
  // Wprowadzenie znaku odstepu przed kazda linia wydawana na zlacze szeregowe aby uniknac
nadawania polecen do TNC (modulu „RadioShield”).
  Serial.print(" ");
  Serial.println(msg);
}
```

Kod źródłowy pliku konfiguracyjnego config.h

```
// Przyklad pliku konfiguracyjnego bramki AirGate. Nalezy podac w nim wlasne dane.

// Wlasny znak wywolawczy z rozszerzeniem:
#define CALLSIGN "WA5ZNU-2"

// Haslo dostepu do serwera APRS:
// Sposob obliczenia hasla podano powyzej.
#define PASSCODE "99999"

// Szerokosc i dlugosc geograficzna lokalizacji bramki:
#define LATITUDE "3725.73N"
#define LONGITUDE "12206.90W"

// PHG – informacja o wyposazeniu stacji i jej mozliwosciach.
// PHG01000 odnosi sie do stacji czysto odiorczej z antena dyskowo-stozkowa o zysku 0 dB na
wysokosci ok. 7 m nad ziemia.
// PHG01600 to samo dla anteny o zysku 6dB (J)
// PGO02600 dla anteny J na wysokosci ok. 10 m.
// Obliczenia dla wlasnego wyposazenia pod adresem http://www.aprsfl.net/phgr.php
#define PHG "PHG01000"

// adres IP dostepu do krajowej bramki APRS-IS:
IPAddress aprsgate(198, 137, 202, 21);    //sjc.aprs2.net

// Adres MAC modulu ethernetowego Arduino:
// Podany na plytce lub na opakowaniu.
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x75, 0xCA };

// Definicja potrzebna w przypadku korzystania z DHCP:
#define USE_DHCP 1
 // W przeciwnym przypadku usunac definicje i podac adres IP, bramke i maske sieci:
// IPAddress ip(192, 168, 178, 177);
// byte gateway[] = { 192, 168, 178, 1 };
// byte subnet[] = { 255, 255, 255, 0 };

// Kanal TCP serwera APRS-IS:
int aprsport = 14580;

// Informacja o programie i wersji do zameldowania na serwerze.
#define VERSION "Arduino_AirGate_TCP (dl8rds,2012-05-30)"
#define INFO "Arduino AirGate IGATE"
```

```
// Adres docelowy pakietow APRS informujacy o uzywanym sprzecie:
// Przyklad dla produktow firmy Argent Data.
#define DESTINATION "APOTW1"
```

Kod źródłowy bramki pracującej w protokóle UDP

```
// Airgate UDP – odbiorcza bramka iGate
// Retransmituje pakiety APRS z modułu „RadioShield” do serwera UDP sieci APRS-IS
// z wykorzystaniem modułu ethernetowego
// Prawa autorskie 2012 Markus Heller, DL8RDS
// patrz LICENSE.txt

#include <Arduino.h>

#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>

EthernetUDP Udp;

// Adresy MAC i IP dla modulu.
// Adres IP zalezny od sieci lokalnej:
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x75, 0xCA };

#define DHCP

#ifndef DHCP
IPAddress ip(192,168,1,145);
byte gateway[] = {192,168,1,1};        // adres IP bramki
byte subnet[] = { 255, 255, 255, 0 };  // maska sieciowa
#endif

byte aprsgate[] = { 173,164,210,105 }; // Adres IP bramki APRS; w tym przykladzie adres lokalnego
komputera
unsigned int localPort = 8888;        // lokalny kanal logiczny nadawcy
int aprsport = 8080;                  // kanal logiczny bramki APRS

char inbyte = 0;
char buf[260];
int buflen = 0;

void setup() {
  // uruchomienie polaczenia ethernetowego i UDP:
#ifndef DHCP
  Ethernet.begin(mac,ip);
#else
  Ethernet.begin(mac);
#endif
  Udp.begin(localPort);
  Serial.begin(4800);                         // inicjalizacja zlacza szeregowego
}

void loop() {
  // odbior danych ze zlacza szeregowego i transmisja do serwera jesli polaczenie nawiazane,
  while (Serial.available() > 0) {
```

```
    inbyte = Serial.read();
    // Get the byte
    if (inbyte == '\n') {
      // sprawdzanie czy osiagniety koniec linii
      buf[buflen++] = inbyte;
      buf[buflen] = 0;
      Udp.beginPacket(aprsgate, aprsport);
      Udp.write(buf);
      Udp.endPacket();
      Serial.print(" ");
      Serial.println(buf);
      buflen = 0;
      buf[buflen] = 0;
    } else if ((inbyte > 31 || inbyte == 0x1c || inbyte == 0x1d || inbyte == 0x27) && buflen < 260) {
      // dopuszczalne tylko znaki czytelne ale rowniez i MIC-E
      buf[buflen++] = inbyte;
      buf[buflen] = 0;
    }
  }
}
```

Transmisja komunikatów APRS przez TNC

Poniższy prosty przykład ilustruje możliwość transmisji komunikatów APRS przy użyciu Arduino i w najprostszym przypadku jednego z modeli radiostacji wyposażonych w modem TNC dla packet-radio i APRS. Oczywiście możliwe jest też użycie dowolnego modelu TNC w połączeniu z dowolną radiostacją FM – ręczną lub domową.
Do połączenia Arduino (Duemilanove, UNO lub podobnego modelu) z radiostacją lub TNC służy jego standardowe złącze szeregowe ale konieczne jest dodanie do niego układu dopasowującego poziomy logiczne TTL do normy RS-232.
Po uruchomieniu program nadaje w funkcji setup() pojedyńczy komunikat tekstowy. Znak nadawcy i adresata podane są odpowiednio w zmiennych *myCall* i *zielCall*. Funkcja loop() jest pusta.
W podprogramie inicjalizacyjnym inicjalizowany jest TNC radiostacji TH-D7E a następnie jest on przełączany na tryb konwersacji (*convers*). Jako trasę transmisji podano w nim „WIDE1-1” ale można ja zmodyfikować w zależności od potrzeb, przykładowo:
Serial1.println("unproto aprs via wide1-1, wide2-2");
Do nadania komunikatu służy funkcja *aprsMsg()*. Następuje w niej złożenie komunikatu do znormalizowanego formatu i wydanie przez złącze szeregowe do TNC.
Program można uzupełnić o:

- Uniwersalny sposób inicjalizacji TNC dostosowany do różnych modeli,
- Dodanie potwierdzeń komunikatów APRS,
- Obsługę dodatkowej klawiatury służącej do wpisywania komunikatów,
- Obsługę dodatkowego wyświetlacza ciekłokrystalicznego dla odebranych komunikatów.

Kod źródłowy

```
/*****************************************************************************
*  Prosty przykład transmisji komunikatów APRS
*  24.09.2011  Jürgen Mayer, DL8MA
*/
char myCall[] = "DL8MA-15";                 // Nadawca
char zielCall[] = "DL8MA-9";                // Adresat

void setup() {
  Serial1.begin(9600);                      // Inicjalizacja TNC w TH-D7E
  Serial1.println( char( 0x03) );           // znak ESC
  delay( 500 );
  Serial1.print("mycall " );                // Wprowadzenie wlasnego znaku do parametru Mycall
  Serial1.println( myCall );
  delay( 500 );
  Serial1.println("unproto aprs via wide1-1" ); // Trasa UNPROTO – WIDE1-1
  delay( 500 );
  Serial1.println("converse" );             // Tryb konwersacji
  delay( 500 );

  aprsMsg( zielcall, "Test II" );           // Nadanie probnego komunikatu
}

void loop() {
}

/*****************************************************************************
*  Nadanie komunikatu APRS
*
*  24.09.2011  JM, DL8MA
*
```

```
* w:   zmiennej  zielCall       znak adresata
*       zmiennej  msgText       komunikat, o dlugosci do 67 znakow
*/
void aprsMsg( String zielCall, String msgText )
{
  int i = 0;
  int laenge = zielCall.length();          // Dlugosc znaku wywolawczego adresata

  Serial1.print(":");                      // wprowadzenie :
  Serial1.print( zielCall );
  if( laenge < 9 ) {                       // wypelnienie zmiennej adresowej (9 znakow) znakami odstepu
    for( i = 0; i < 9 - laenge; i++ )
      Serial1.print(" ");
  }
  Serial1.print(":");                      // rozgranicznie miedzy trasa transmisji i tekstem komunikatu
  Serial1.println( msgText);               // tekst komunikatu
}
```

Dodatek A
Prosty serwer konferencyjny

Serwer konferencyjny dla sieci lokalnej rozsyła otrzymane wiadomości (teksty) do wszystkich podłączonych użytkowników w oparciu o protokół Telnet. Do połączenia z siecią służy moduł Ethernet. Program ten i następne pochodzą z dokumentacji Arduino z witryny *www.arduino.cc* i mają służyć jako przykłady do zastosowań amatorskich. Przed ich wypróbowaniem należy wprowadzić rzeczywiste adresy IP i sprzętowe MAC.
Do nawiązania połączenia telnetowego z serwerem należy skorzystać z dowolnego telnetowego programu terminalowego j.np. Putty.

```
/*
Serwer konferencyjny

Prosty serwer rozsylajacy otrzymane komunikaty do wszystkich podlaczonych uzytkownikow
W celu przekazania komunikatu doserwera nelezy nawiazac z nim polaczenie telnetowe
Podajac jego adres IP. Otrzymane komunikaty wyswietlane sa w oknie monitora Arduino.
Program wykorzystuje ethernetowy modul rozszerzenia Wiznet.

Konstrukcja:
* Modul ethernetowy jest podlaczony do kontaktow 10, 11, 12, 13
* Wejscia analogowe podlaczone do kontaktow A0 do A5

Program z 18 grudnia 2009
autor David A. Mellis
zmodyfikowany 9 kwietnia 2012
przez Toma Igoe
*/

#include < SPI.h>
#include <Ethernet.h>

// W liniach ponizej podane sa adres MAC i IP modulu ethernetowego
// Adres IP zalezny od lokalnej sieci uzytkownika
// adres bramki (gateway) i maska sieci nie sa obowiazkowe:
byte mac[] = { 0xDE, 0xAD,0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192,168,1, 177);
IPAddress gateway(192,168,1, 1);
IPAddress subnet(255, 255, 0, 0);

// telnet korzysta domyslnie z kanalu logicznego 23
EthernetServer server(23);
boolean alreadyConnected =false; // sprawdzenie czy klient byl juz poprzednio polaczony

void setup() {
 // inicjalizacja modulu ethernetowego
 Ethernet.begin(mac, ip, gateway, subnet);
 // poczatek nasluchu klientow
 server.begin();
 // Inicjalizacja zlacza szeregowego i oczekiwanie na jego otwarcie:
 Serial.begin(9600);
  while (!Serial){
   ; // oczekiwaie konieczne tylko dla Arduino Leonardo
}
```

```
Serial.print("Chat server address:");
Serial.println(Ethernet.localIP());
}

void loop() {
  // oczekiwanie na polaczenie ze strony klienta:
  EthernetClient client = server.available();

  // powitanie klienta po otrzymaniu od niego pierwszego bajtu:
  if (client) {
    if(!alreadyConnected){
      // oproznienie bufora odbiorczego:
      client.flush();
      Serial.println("We have a new client");          // informacja w oknie monitora Arduino
      client.println("Hello, client!");
      alreadyConnected = true;
    }

    if (client.available() >0) {
      // odczyt danych nadchodzacych od klienta:
      char thisChar = client.read();
      // echo dla klienta:
      server.write(thisChar);
      // echo do monitora:
      Serial.write(thisChar);
    }
  }
}
```

Dodatek B
Klient HTTP

Prosty program klienta łączy się z serwerem HTTP w Internecie. Podobnie jak poprzedni pochodzi z dokumentacji Arduino i stanowi przykład wykorzystania biblioteki „Ethernet”.

```
/*
Klient HTTP

Program nawiazuje polaczenie z witryna internetowa (http://www.google.com)
Korzystajac z modulu rozszeren Arduino Wiznet Ethernet.

Konstrukcja:
* modul ethernetowy podlaczony do kontaktow 10, 11, 12, 13

Program z 18 grudnia 2009
autor David A. Mellis
zmodyfikowany 9 kwietnia 2012
przez Toma Igoe, oparty na opracowaniu Adriana McEwena
*/

#include <SPI.h>
#include < Ethernet.h>

// Adres MAC modulu ethernetowego
// Nowsze moduly maja adres podany na nalepce na plytce
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
// w przypadku nie korzystania z DNS (zapewnia zmniejszenie objetosci programu)
// nalezy podac adres IP zamiast nazwy serwera:
//IPAddress server(74,125,232,128); // adres IP witryny Google (bez DNS)
char server[] = "www.google.com"; // adres symboliczny witryny Google (korzystanie z DNS)

// Podanie stalego adresu IP modulu gdy nie wykorzystywany DHCP
IPAddress ip(192,168,0,177);

// Inicjalizacja biblioteki klienta ethernetowego
// podanie adresu i kanalu logicznego serwera
// dla HTTP standardowo kanal 80:
EthernetClient client;

void setup(){
  // Inicjalizacja zlacza szeregowego i oczekiwanie na jego otwarcie:
  Serial.begin(9600);
   while (!Serial){
    ; // oczekiwanie na otwarcie konieczne tylko dla Arduino Leonardo
    }

  // nawiazanie polaczenia ethernetowego:
  if (Ethernet.begin(mac)== 0){
    Serial.println("Failed to configure Ethernet using DHCP");
    // w przypadku niepowodzenia z wykorzystaniem DHCP
    // nalezy przyznac adres IP:
    Ethernet.begin(mac, ip);
  }
  // odczekanie sekundy na inicjalizacje modulu:
```

```
  delay(1000);
  Serial.println("connecting...");

  // informacja o nawiazaniu polaczenia wyswietlana w oknie monitora Arduino:
  if (client.connect(server, 80)) {
    Serial.println("connected");
    // Wyslanie zapytania HTTP:
    client.println("GET /search?q=arduino HTTP/1.1");
    client.println("Host: www.google.com");
    client.println("Connection: close");
    client.println();
  }
  else {
    // informacja w przypadku niemoznosci nawiazania polaczenia:
    Serial.println("connection failed");
  }
}

void loop()
{
  // odbior informacji od serwera i ich wyswietlenie w oknie monitora:
  if (client.available()) {
    char c = client.read();
    Serial.print(c);
  }

  // po zakonczeniu polaczenia z serwerem zatrzymanie programu klienta:
  if(!client.connected()) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();

    //petla bez zadnych dalszych czynnosci:
    while(true);
  }
}
```

Dodatek C
Klient Telnetu

Protokół Telnet jest wykorzystywany do komunikacji tekstowej (terminalowej) w internecie i do różnych celów diagnostycznych. Standardowo korzysta on z kanału logicznego 23 ale w dla celów diagnostycznych należy korzystać z kanału logicznego badanej usługi.

```
/*
Klient Telnetu

Program sluzy do nawiazania polaczenia telnetowego z serwerem internetowym
Przy uzyciu modulu ethernetowego Wiznet dla Arduino. Dla sprawdzenia programu konieczny jest
dostep do serwera Telnetu.
Moze byc to podany w przykladzie Processing's ChatServer dostepny w kanale logicznym 10002.
http://processing.org/

Konstrukcja:
* modul ethernetowy podlaczony do kontaktow 10, 11, 12, 13

program z 14 wrzesnia 2010
zmodyfikowany 9 kwietnia 2012
przez Toma Igoe
*/

#include <SPI.h>
#include <Ethernet.h>

// Wprowadzenie adresow MAC i IP dla modulu.
// Adres IP zalezny od lokalnej sieci:
byte mac[] = { 0xDE, 0xAD,0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192,168,1,177);

// Wprowadzenie adresu IP serwera docelowego:
IPAddress server(1,1,1,1);

// Inicjalizacja biblioteki klienta ethernetowego
// przez podanie adresu IP i kanalu logicznego serwera docelowego
// domyslnie telnet korzysta z kanalu 23;
// na serwerze Processing's ChatServer jest to kanal 10002:
EthernetClient client;

void setup() {
  // nawiazanie polaczenia ethernetowego:
  Ethernet.begin(mac, ip);
  // Inicjalizacja zlacza szeregowego i oczekiwanie na jego otwarcie:
  Serial.begin(9600);
  while (!Serial){
    ; // oczekiwanie na otwarcia konieczne tylko dla Arduino Leonardo
  }

  // odczekanie sekundy na inicjalizacje:
  delay(1000);
  Serial.println("connecting...");                    // meldunek w oknie monitora Arduino

  // po nawiazaniu polaczenia informacja dla uzytkownika:
```

```
  if (client.connect(server, 10002)){
    Serial.println("connected");
  }
else {
  // informacja o niemoznosci nawiazania polaczenia:
  Serial.println("connection failed");
  }
}

void loop()
{
  // odbior i wyswietlenie danych otrzymywanych z serwera
  if (client.available()) {
    char c = client.read();                 // odbior przez obiekt klienta
    Serial.print(c);                        // wydanie na komputer
  }

  // odczyt informacji z bufora nadawczego i nadanie ich przez telnet jesli polaczenie jest otwarte
  while (Serial.available() >0) {
    char inChar = Serial.read();
    if (client.connected()) {
      client.print(inChar);
    }
  }

  // po rozlaczeniu przez serwer zatrzymanie klienta:
  if(!client.connected()) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
    // petla bez zadnych dalszych czynnosci:
    while(true);
  }
}
```

Dodatek D
Klient Twittera z wykorzystaniem serwera DHCP

```
/*
Klient Twittera z wykorzystaniem biblioteki String

Program nawiazuje polaczenie z Twitterem przez modul ethernetowy i przeszukuje
Otrzymany dokument XML w poszukiwaniu konstrukcji  <text>tekst komunikatu</text>

Mozliwe jest uzycie modulu ethernetowego Arduino lub Adafruit
poniewaz oba zawieraja kontroler Wiznet

W tym przykladzie wykorzystano funkcje DHCP z biblioteki ethernetowej.
Naleza one do jadra Arduino poczawszy od wersji 1.0 beta 1

Przyklad korzysta takze z biblioteki String nalezacej do jadra Arduino poczawszy od wersji 0019.

Konstrukcja:
* modul ethernetowy podlaczony do kontaktow 10, 11, 12, 13

Program z 21 maja 2011
Zmodyfikowany 9 kwietnia 2012
przez Toma Igoe

Kod programu jest ogolnie dostepny
*/

#include <SPI.h>
#include < Ethernet.h>

// Podanie adresow MAC i IP dla modulu ethernetowego.
// Adres IP zalezny od lokalnej sieci:
byte mac[]= { 0x00, 0xAA,0xBB, 0xCC, 0xDE, 0x01 };
IPAddress ip(192,168,1,20);

// inicjalizacja obiektu klienta:
EthernetClient client;

const unsigned long requestInterval = 60000; // odstep czasu pomiedzy zapytaniami

char serverName[] = "api.twitter.com"; // adrers URL twittera

boolean requested; // sygnalizacja czy od nawiazania polaczenia zostalo nadane zapytanie
unsigned long lastAttemptTime = 0; // odstep czasu od ostatniego polaczenia w ms

String currentLine = "";                // zmienna tekstowa na odpowiedz od serwera
String tweet = "";                      // zmienna tekstowa dla komunikatu
boolean readingTweet = false;           // sygnalizacja biezacego odbioru komunikatu

void setup(){
  // rezerwacja miejsca na teksty:
  currentLine.reserve(256);
  tweet.reserve(150);

  // Inicjalizacja zlacza szeregowego i oczekiwanie najego otwarcie:
```

```
  Serial.begin(9600);
   while (!Serial){
     ; // oczekiwanie na otwarcie konieczne tylko dla Arduino Leonardo
  }

  // proba polaczenia DHCP:
  Serial.println("Attempting to get an IP address using DHCP:");
  if (!Ethernet.begin(mac)){
    // w przypadku niemoznosci skorzystania z DHCP uzycie wpisanego adresu:
    Serial.println("failed to get an IP address using DHCP, trying manually");
    Ethernet.begin(mac, ip);
  }
  Serial.print("My address:");
  Serial.println(Ethernet.localIP());
  // polaczenie z Twitterem:
  connectToServer();
}

void loop()
{
  if (client.connected()) {
    if (client.available()) {
      // odczyt odbieranych danych:
      char inChar = client.read();

      // dodanie odebranych danych na koncu linii:
      currentLine += inChar;

      // po odebraniu znaku nowej linii kasowanie bufora linii:
      if (inChar == '\n') {
        currentLine ="";
      }
      // jesli na koncu linii znajduje sie ciag <text> sygnalizuje to
      // poczatek komunikatu:
      if ( currentLine.endsWith("<text>")) {
        // poczatek komunikatu. Skasowanie bufora komunikatu:
        readingTweet =true;
        tweet = "";
      }
      // odczytywane dane komunikatu dodawane do bufora komunikatu
      if (readingTweet) {
        if (inChar != '<'){
          tweet += inChar;
        }
      else{
        // znak "<" sygnalizuje
        // koniec komunikatu:
        readingTweet = false;
        Serial.println(tweet);
        // zakonczenie polaczenia z serwerem:
        client.stop();
      }
    }
   }
  }
```

```
  else if (millis() - lastAttemptTime > requestInterval) {
    // po uplywie dwoch minut od przerwania polaczenia ponowna proba nawiazania
    connectToServer();
  }
}

void connectToServer(){
  // proba nawiazania polaczenia i oczekiwanie:
  Serial.println("connecting to server...");
  if (client.connect(serverName, 80)) {
    Serial.println("making HTTP request...");
    // nadanie polecenia HTTP GET do twittera:
    client.println("GET /1/statuses/user_timeline.xml?screen_name=arduino&count=1 HTTP/1.1");
    client.println("HOST: api.twitter.com");
    client.println("Connection: close");
    client.println();
  }
  // zapamietanie czasu proby:
  lastAttemptTime =millis();
}
```

Dodatek E
Serwer HTTP

Program serwera HTTP udostępnia klientom wartości odczytane z wejść analogowych Arduino.

```
/*
Serwer http

Prosty serwer http wyswielajacy wartosci napiec na wejsciach analogowych Arduino.
Korzysta z ethernetowego modulu rozszerzen.

Konstrukcja:
* Modul ethernetowy podlaczony do kontaktow 10, 11, 12, 13
* Wejscia analogowe na kontaktach A0 do A5

Program z 18 grudnia 2009
autor David A. Mellis
zmodyfikowany 9 kwietnia 2012
przez Toma Igoe
*/

#include < SPI.h>
#include <Ethernet.h>

// Wprowadzenie adresow MAC i IP dla modulu ethernetowego.
// Adres IP zalezny od lokalnej sieci:
byte mac[] = { 0xDE, 0xAD,0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192,168,1,177);

// Inicjalizacja etherenetowej biblioteki klienta
// podanie kanalu logicznego
// (domyslnie dla http uzywany kanal 80):
EthernetServer server(80);

void setup() {
  // Inicjalizacja zlacza szeregowego i oczekiwanie na otwarcie:
  Serial.begin(9600);
   while (!Serial){
    ; // oczekiwanie na otwarcie konieczne tylko dla Arduino Leonardo
   }

  // Nawiazanie polaczenia ethernetowego z serwerem:
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());
}

void loop() {
  // oczekiwanie na zgloszenia klientow
  EthernetClient client = server.available();
  if (client){
  Serial.println("new client");
  // Polecenie http zakonczone pusta linia
  boolean currentLineIsBlank = true;
```

```
  while(client.connected()) {
   if (client.available()) {
     char c = client.read();
     Serial.write(c);
     // po dojsciu do konca linii pustej (odbiorze pary znakow CR/LF)
     // czyli po odebraniu pelnego polecenia mozna udzielic odpowiedzi
     if (c == '\n' && currentLineIsBlank) {
       // nadanie standardowego naglowka odpowiedzi
       client.println("HTTP/1.1 200 OK");
       client.println("Content-Type: text/html");
       client.println("Connection: close"); // po zakonczeniu odpowiedzi przerwanie polaczenia
       client.println("Refresh: 5"); // automatyczne odswiezanie strony co 5 sekund
       client.println();
       client.println("<!DOCTYPE HTML>");
       client.println("<html>");
       // wydanie wartosci napiec na kazdym z wejsc analogowych
       for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
         int sensorReading = analogRead(analogChannel);
         client.print("analog input ");
         client.print(analogChannel);
         client.print(" is ");
         client.print(sensorReading);
         client.println("<br />");
       }
       client.println("</html>");
       break;
     }
     if (c =='\n'){
       // w przypadku nadania znaku nowej linii
       currentLineIsBlank = true;
     }
     else if (c !='\r'){
       // odbior znaku alfanumerycznego w bieyacej linii
       currentLineIsBlank = false;
     }
   }
  }
  // odstep czasu na odbior danych przez przegladarke internetowa
  delay(1);
  // przerwanie polaczenia:
  client.stop();
  Serial.println("client disconnected");
 }
}
```

Dodatek F
Serwer konferencyjny WiFi

Jest to analogiczny przykład jak dla biblioteki ethernetowej ale z wykorzystniem modułu i połączenia bezprzewodowego. Również następne przykłady są analogiczne do przykładów ethernetowych ale wykorzystują połączenie bezprzewodowe i pochodzą z oficjalnej dokumentacji Arduino. Dodatkowo do wprowadzenia adresów IP i MAC konieczne jest tutaj podanie nazwy sieci bezprzewodowej i kodu dostępu do niej.
Oprócz sieci otwartych dostępnych dla wszystkich – co może mieć sens w miejscach publicznych ale nie w prywatnych lub firmowych sieciach WiFi – najczęściej spotykane są sieci dostępne jedynie po podaniu hasła dostępu lub klucza. Obecnie w lokalnych sieciach bezprzewodowych stosowane są zabezpieczenia WEP, WPA i WPA2.

- WEP (ang. *Wired Equivalent Privacy* – zabezpieczenie na poziomie sieci kablowej, IEEE 802.11) – jest dawniejszym standardem szyfrowania obecnie rzadziej używanym ponieważ nie zapewnia on dostatecznego bezpieczeństwa. Dane szyfrowane są za pomocą operacji XOR między nimi a wygenerowanym w oparciu o algorytm RC4 pseudolosowym ciągiem bitów. Generator liczb (ciągów) pseudolosowych jest inicjalizowany za pomocą podanego klucza zwanego także wektorem początkowym. Do strumienia zaszyfrowanych danych dodawana jest suma kontrolna CRC. Klucz WEP służy także do uwierzytelnienia użytkowników sieci. Każdy z użytkowników dysponujący właściwym kluczem otrzymuje dostęp do sieci i może korzystać z jej funkcjonalności.
- WPA (ang. *WiFi Protected Access* – zabezpieczony dostęp do sieci WiFi, IEEE 802.11i) – jest standardem wywodzącym się z WEP i bazującym również na algorytmie RC4. Oprócz wektora początkowego o długości 48 bitów stosowane są dalsze mechanizmy. Jako dodatkowe zabezpieczenie służy klucz dynamiczny obliczany według algorytmu TKIP (ang. *Temporary Key Integrity Protocol*). Do uwierzytelnienia użytkowników służy albo klucz PSK (ang. *Pre-shared Key*) albo protokół EAP (ang. *Extensible Authentification Protocol*). W małych prywatnych sieciach najczęściej stosowany jest klucz PSK.
- Następcą WPA jest standard WPA2. Znalazł on użycie w sieciach opartych na standardach IEEE 802.a, b, g, i n. Podstawę WPA2 stanowi standard szyfrowania AES (ang. *Advanced Encryption Standard*). Do uwierzytelnienia użytkowników może służyć zarówno tajne hasło, klucz PSK jak i serwer RADIUS. W małych prywatnych sieciach bardzo często stosowany jest klucz PSK.
  O bezpieczeństwie sieci decyduje w znacznym stopniu długość hasła dostępu, dlatego też zalecane jest aby miało ono długość co najmniej 20 znaków i zawierało kombinację liter małych, dużych, cyfr i znaków specjalnych). Przejście z WPA na WPA2 jest w wielu przypadkach możliwe poprzez wymianę oprogramowania firmowego urządzeń takich jak punkty dostępowe (ang. *router*) itp.

W zależności od rodzaju zabezpieczeń sieci programiści Arduino mają do dyspozycji następujące warianty funkcji WiFi.begin():

- WiFi.begin();
- WiFi.begin(ssid);
- WiFi.begin(ssid, pass);
- WiFi.begin(ssid, keyIndex, key);

Gdzie:

- ssid – oznacza nazwę wykorzystywanej sieci bezprzewodowej,
- keyIndex – indeks do jednego z możliwych czterech kluczy w sieciach WEP,
- key – kod zabezpieczaj acy dla sieci WEP. Ma postać szesnastkową.
- pass – hasło dostępu do sieci WPA.

1) Przykład dostępu do sieci zabezpieczonej w standardach WPA/WPA2 Personal (moduł nie współpracuje z sieciami zabezpieczonymi w standardzie WPA2 Enterprise):

```
#include <WiFi.h>

//Nazwa (SSID) sieci
char ssid[] = "yourNetwork";
// hasło dostępu do sieci WPA
```

```
char pass[] = "secretPassword";

void setup()
{
  WiFi.begin(ssid, pass);
}

void loop () {}
```

2) Przykład dostępu do sieci zabezpieczonej w standardzie WEP:

```
#include <WiFi.h>

char ssid[] = "yourNetwork"; // nazwa (SSID) sieci
char key[] = "D0D0DEADF00DABBADEAFBEADED"; // kod dostępu
int keyIndex = 0; // indeks do kodu
int status;

void setup()
{
  Status = WiFi.begin(ssid, keyIndex, key);
}

void loop () {}
```

3) Przykład dostepu do sieci nie zabezpieczonej

```
#include <WiFi.h>

char ssid[] = "yourNetwork"; // nazwa (SSID) sieci
int status;

void setup()
{
  Status = WiFi.begin(ssid);
}

void loop () {}
```

Kod źródłowy

```
/*
Serwer konferencyjny

Prosty serwer rozsylajacy otrzymane wiadomosci do wszystkich polaczonych z nim uzytkownikow
Nalezy nawiazac z nim polaczenie telnetowe poslugujac sie jego adresem IP
i napisac wiadomosc. Wiadomosc ta jest wyswietlana takze w oknie monitora szeregowego Arduino.

Przyklad ten jest przeznaczony dla sieci korzystajacej z kodowania WPA. Dla pozostalych wariantow
nalezy odpowiednio dopasowac wywolanie Wifi.begin().

Konstrukcja:
* podlaczony modul WiFi

Program z 18 grudnia 2009
autor David A. Mellis
```

```
*zmodyfikowany 31 maja 2012
przez Toma Igoe
*/

#include <SPI.h>
#include <WiFi.h>

char ssid[] = "yourNetwork"; // podac nazwe (SSID) wlasnej sieci
char pass[] = "secretPassword"; // podac haslo dostepu dla WPA, lub klucz dla WEP

int keyIndex = 0; // indeks klucza w sieci (konieczny tylko dla WEP)

int status = WL_IDLE_STATUS;

WiFiServer server(23);

boolean alreadyConnected = false; // sygnalizacja czy klient jest juz polaczony

void setup() {
  // Inicjalizacja zlacza szeregowego i oczekiwanie na jego otwarcie:
  Serial.begin(9600);
  while (!Serial) {
    ; // oczekiwanie na otwarcie konieczne tylko dla Arduino Leonardo
  }

// Sprawdzenie czy modul WiFi jest podlaczony do Arduino:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // przerwanie dzialanosci:
    while(true);
  }

  // proba polaczenia z siecia WiFi:
  while ( status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    // Polaczenie z siecia WPA/WPA2. Dla WEP nalezy odpowiednio zmienic polecenie:
    status = WiFi.begin(ssid, pass);

    // oczekiwanie 10 sekund na uzyskanie polaczenia:
    delay(10000);
  }
  // uruchomienie serwera:
  server.begin();
  // wyswietlenie informacji gdy polaczenie nawiazane:
  printWifiStatus();
}

voidloop() {
  // oczekiwanie na polaczenie ze strony klienta:
  WiFiClient client = server.available();

  // powitanie klienta po otrzymaniu od niego pierwszego bajtu danych:
  if (client) {
    if (!alreadyConnected) {
```

```
      // oproznienie bufora wejsciowego:
      client.flush();
      Serial.println("We have a new client");
      client.println("Hello, client!");
      alreadyConnected = true;
     }

     if (client.available() >0) {
      // odczyt danych otrzymywanych od klienta:
      char thisChar = client.read();
      // wyslanie ich jako echo do klienta:
      server.write(thisChar);
     // wyslanie ich takze do zlacza szeregowego:
     Serial.write(thisChar);
     }
   }
}

void printWifiStatus(){
  // wyswietlenie nazwy sieci (SSID):
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // wyswietlenie adresu IP modulu WiFi:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  //wyswietlenie sily odbieranego sygnalu:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

Dodatek G
Bezprzewodowy klient HTTP

```
/*
Klient HTTP
Program nawiazuje polaczenie z podana witryna internetowa (http://www.google.com)
Korzystajac z modulu bezprzewodowego WiFi
Przyklad jest przeznaczony dla sieci korzystajacej z szyfrowania WPA. Dla pozostalych wariantow
nalezy odpowiednio zmodyfikowac wywolanie Wifi.begin().

Konstrukcja:
* podlaczony modul WiFi

Program z 13 lipca 2010
Autor dlf (Metodo2 srl)
zmodyfikowany 31 maja 2012
przez Toma Igoe
*/

#include <SPI.h>
#include <WiFi.h>

char ssid[] = "yourNetwork"; // nazwa (SSID)sieci
char pass[] = "secretPassword"; // haslo dostepu do sieci dla WPA, lub klucz dla WEP
int keyIndex =0; // indeks do hasla, potrzebny tylko dla WEP

int status = WL_IDLE_STATUS;
// w razie nie korzystania z DNS (co skraca  program)
// nalezy wpisac adres IP zamiast nazwy serwera:
// IPAddress server(74,125,232,128); // adres IP witryny Google (bez DNS)
char server[] = "www.google.com"; // nazwa witryny Google (z uzyciem DNS)

// Inicjalizacja biblioteki klienta z podaniem kanalu logicznego
// (domyslnie dla http jest to kanal 80):
WiFiClient client;

void setup(){
  // Inicjalizacja zlacza szeregowego i oczekiwanie na jego otwarcie:
  Serial.begin(9600);
  while (!Serial) {
    ; // oczekiwanie na otwarcie konieczne tylko dla Arduino Leonardo
  }

  // sprawdzenie czy modul podlaczony:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // jesli nie – przerwanie dzialalnosci:
    while(true);
  }

  // proba polaczenia z siecia bezprzewodowa (Wifi):
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
```

```
    // Polaczenie z siecia WPA/WPA2 network. Dla pozostalych wariantow nalezy odpowiednio
zmodyfikowac wywolanie WiFi.begin():
    status = WiFi.begin(ssid, pass);

    // oczekiwanie przez 10 sekund na nawiazanie polaczenia:
    delay(10000);
  }
  Serial.println("Connected to wifi");       // meldunek w oknie monitora Arduino
  printWifiStatus();

  Serial.println("\nStarting connection to server...");
  // po nawiazaniu polaczenia wyswietlenie informacji w oknie monitora:
  if (client.connect(server, 80)) {
    Serial.println("connected to server");
    // Nadanie polecenia HTTP:
    client.println("GET /search?q=arduino HTTP/1.1");
    client.println("Host: www.google.com");
    client.println("Connection: close");
    client.println();
  }
}

void loop() {
  // odczyt danych z serwera i ich wyswietlenie:
  while (client.available()) {
    char c = client.read();
    Serial.write(c);
  }

  // po rozlaczeniu przez serwer zatrzymanie programu klienta:
  if(!client.connected()) {
    Serial.println();
    Serial.println("disconnecting from server.");
    client.stop();

    // petla bez zadnej dalszej dzialalnosci:
    while(true);
  }
}

void printWifiStatus(){
  Serial.print("SSID: ");                    // wyswietlenie nazwy (SSID) sieci:
  Serial.println(WiFi.SSID());

  IPAddress ip = WiFi.localIP();         // wyswietlenie adresu modulu WiFi:
  Serial.print("IP Address: ");
  Serial.println(ip);

  long rssi = WiFi.RSSI();                    // wyswietlenie sily odbieranego sygnalu:
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

Dodatek H
Bezprzewodowy klient Twittera

```
/*
Bezprzewodowy klient Twittera z wykorzystaniem obiektu Strings

Program laczy sie z Twitterem poprzez bezprzewodowy modul WiFi Arduino.
Interpretuje otrzymany dokument XML w celu wylowienia z niego komunikatow < text>tekst
komunikatu</text>

Przyklad dla sieci WPA. Dla pozostalych wariantow nalezy odpowiednio zmodyfikowac wywolanie
Wifi.begin().

Przyklad korzysta z biblioteki String, nalezacej do jadra Arduino poczawszy od wersji  0019.

Konstrukcja:
* modul WiFi podlaczony do kontaktow 10, 11, 12, 13

Program z 23 kwietnia 2012
zmodyfikowany 31 maja 2012
przez Toma Igoe
Kod dostepny bezplatnie.
*/
#include < SPI.h>
#include <WiFi.h>

char ssid[] = "yourNetwork"; // nazwa (SSID) sieci
char pass[] = "password"; // haslo dostepu (dla WPA, lub jako klucz dla WEP)
int keyIndex = 0; // indeks, konieczny tylko dla WEP

int status = WL_IDLE_STATUS; // stan polaczenia wifi

// inicjalizacja obiektu klienta:
WiFiClient client;

const unsigned long requestInterval = 30*1000; //odstep czasu miedzy zapytaniami, 30 sekund

// w razie nie korzystania z DNC (co skraca program)
// nalezy podac adres IP serwera zamiast jego nazwy:
//IPAddress server(199,59,149,200); // adres IP witryny api.twitter.com
char server[] = "api.twitter.com"; // adres symboliczny Twittera

boolean requested; // sygnalizacja czy wyslane zapytanie od czasu nawiazania polaczenia
unsigned long lastAttemptTime = 0; // odstep czasu od ostatniego polaczenia w milisekundach

String currentLine =""; // bufor odbiorczy dla danych z serwera
String tweet = ""; // bufor dla komunikatu
boolean readingTweet =false; // sygnalizacja trwajacego odbioru komunikatu

void setup(){
  // rezerwacja miejsca dla buforow:
  currentLine.reserve(256);
  tweet.reserve(150);
  // Inicjalizacja zlacza szeregowego i oczekiwanie na jego otwarcie:
  Serial.begin(9600);
```

```
  while (!Serial) {
    ; // oczekiwanie na otwarcie, konieczne tylko dla Arduino Leonardo
  }

  // sprawdzenie czy modul podlaczony do Arduino:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // przerwanie pracy:
    while(true);
  }

  // proba polaczenia z siecia bezprzewodowa:
  while ( status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    //Polaczenie z siecia WPA/WPA2. Dla polaczenia z siecia otwarta lub WEP nalezy zmodyfikowac
wywolanie:
    status = WiFi.begin(ssid, pass);

    // odczekanie 10 sekund na nawiazanie polaczenia:
    delay(10000);
  }
  // po nawiazaniu polaczenia wyswietlenie informacji o tym:
  printWifiStatus();
  connectToServer();
}

void loop()
{
  if (client.connected()) {
    if (client.available()) {
      // odbior naplywajacych informacji:
      char inChar = client.read();

      // umieszczanie ich na koncu bufora linii:
      currentLine += inChar;

      // po odebraniu znaku nowej linii skasowanie bufora:
      if (inChar == '\n') {
        currentLine ="";
      }
      // odbior na koncu linii ciagu <text>
      // sygnalizuje poczatek komunikatu:
      if ( currentLine.endsWith("<text>")) {
        // poczatek komunikatu. Skasowanie dotychczasowej zawartosci bufora:
        readingTweet =true;
        tweet = "";
        // opuszczenie petli, tak ze biezacy znak nie jest dodawany do komunikatu:
        return;
      }
      // odbierane dane komunikatu sa dodawane do jego bufora:
      if (readingTweet) {
        if (inChar != '<'){
          tweet += inChar;
        }
```

```
        else{
          // odbior znaku"<" sygnalizuje koniec komunikatu
          readingTweet = false;
          Serial.println(tweet);
          // przerwanie polaczenia z serwerem:
          client.stop();
        }
      }
    }
  }
  else if (millis()- lastAttemptTime > requestInterval) {
    // po uplywie dwoch minut od ostatniego polaczenia proba ponownego jego nawiazania:
    connectToServer();
  }
}

void connectToServer(){
  // proba polaczenia i oczekiwaniue:
  Serial.println("connecting to server...");
  if (client.connect(server, 80)) {
    Serial.println("making HTTP request...");
    //nadanie zapytania GET do twittera:
    client.println("GET /1/statuses/user_timeline.xml?screen_name=arduino HTTP/1.1");
    client.println("Host: api.twitter.com");
    client.println("Connection: close");
    client.println();
  }
  // zapamietanie czasu proby nawiazania polaczenia:
  lastAttemptTime =millis();
}

void printWifiStatus() {
  // wyswietlenie nazwy (SSID) sieci:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // wyswietlenie adresu IP modulu WiFi:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // wyswietlenie sily odbieranego sygnalu:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

Dodatek I
Bezprzewodowy serwer HTTP

```
/*
Bezprzewodowy serwer HTTP

Prosty serwer HTTP udostepniajacy wartosci wczytane na wejsciach analogowych.
Korzysta z modulu WiFi.

Przyklad dla sieci zabezpieczonej przez WPA. Dla pozostalych wariantow nalezy odpowiednio
dopasowac wywolanie Wifi.begin().

Konstrukcja:
* Podlaczony modul WiFi
* Wejscia analogowe podlaczone do kontaktow A0 do A5

Program z 13 lipca 2010
autor dlf (Metodo2 srl)
zmodyfikowany 31 maja 2012
przez Toma Igoe
*/

#include <SPI.h>
#include <WiFi.h>

char ssid[] = "yourNetwork";       // nazwa (SSID) sieci
char pass[] = "secretPassword";  // haslo dostepu
int keyIndex = 0;                  // indeks do hasla (konieczny tylko dla WEP)

int status = WL_IDLE_STATUS;

WiFiServer server(80);

void setup() {
  // Inicjalizacja zlacza szeregowego i oczekiwanie na jego otwarcie:
  Serial.begin(9600);
  while (!Serial) {
    ; // oczekiwanie konieczne tylko dla Arduino Leonardo
  }

  // sprawdzenie czy modul podlaczony:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // przerwanie dzialalnosci:
    while(true);
  }

  // proba nawiazania polaczenia z siecia bezprzewodowa:
  while ( status != WL_CONNECTED) {
  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);
  // Polaczenie z siecia WPA/WPA2. Dla WEP lub sieci otwartej dopasowac wywolanie:
  status = WiFi.begin(ssid, pass);

  // oczekiwanie przez 10 sekund na nawiazanie polaczenia:
```

```
  delay(10000);
  }
  server.begin();
  // wyswietlenie informacji o nawiazaniu polaczenia:
  printWifiStatus();
}

void loop() {
  // oczekiwanie na polaczenia klientow
  WiFiClient client = server.available();
  if (client) {
    Serial.println("new client");
    // zapytanie HTTP zakonczone pusta linia
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
       // odbior znaku nowej linii na koncu linii pustej
       // oznacza zakonczenia zapytania i serwer moze zaczac nadawac odpowiedz:
        if (c == '\n' && currentLineIsBlank) {
          // nadanie standardowego naglowka HTTP
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close"); // na zakonczenie przerwanie polaczenia
          client.println("Refresh: 5"); // automatyczne odswiezanie strony co 5 sekund
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");
          // wydanie wartosci z kazdego z wejsc analogowych
          for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
            int sensorReading = analogRead(analogChannel);
            client.print("analog input ");
            client.print(analogChannel);
            client.print(" is ");
            client.print(sensorReading);
            client.println("<br />");
          }
          client.println("</html>");
          break;
        }
        if (c == '\n') {
          // poczatek nowej linii
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          // odbior znaku w biezacej linii
          currentLineIsBlank = false;
        }
      }
    }
    // czas dla przegladarki na odbior danych
    delay(1);

    // zakonczenie polaczenia:
```

```
    client.stop();
    Serial.println("client disonnected");
  }
}

void printWifiStatus() {
  // wyswietlenie nazwy sieci:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // wyswietlenie adresu IP modulu WiFi:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // wyswietlenie sily odbieranego sygnalu:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

Dodatek J
Poszukiwanie sieci bezprzewodowych

```
/*
Przykladowy program wyswietlajacy adres MAC modulu WiFi i poszukujacy
dostepnych sieci bezprzewodowych. Poszukiwanie jest powtarzane co 10 sekund.
Program korzysta z modulu WiFi.
Program nie nawiazuje polaczenia z zadna z nich dlatego nieistotny jest sposob zabezpieczenia dostepu
do nich.

Konstrukcja:
*podlaczony modul WiFi

Program z 13 lipca 2010
autor dlf (Metodo2 srl)
zmodyfikowany 21 czerwca 2012
przez Toma Igoe i Jaymesa Dec
*/

#include <SPI.h>
#include <WiFi.h>

void setup() {
  // Inicjalizacja zlacza szeregowego i oczekiwanie na jego otwarcie
  Serial.begin(9600);
  while (!Serial) {
    ; // oczekiwanie konieczne tylko dla Arduino Leonardo.
  }

  // sprawdzenie czy modul podlaczony do Arduino:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // przerwanie dzialanosci:
    while(true);
  }

  // Print WiFi MAC address:
  printMacAddress();

  // poszukiwanie dostepnych sieci:
  Serial.println("Scanning available networks...");
  listNetworks();
}

void loop() {
  delay(10000);                                    // odczekanie 10 sekund
  // poszukiwanie dostepnych sieci:
  Serial.println("Scanning available networks...");
  listNetworks();
}

void printMacAddress(){
  // odczyt adresu MAC modulu
  byte mac[6];
```

```
  // wyswietlenie adresuMAC modulu WiFi:
  WiFi.macAddress(mac);
  Serial.print("MAC: ");
  Serial.print(mac[5],HEX);
  Serial.print(":");
  Serial.print(mac[4],HEX);
  Serial.print(":");
  Serial.print(mac[3],HEX);
  Serial.print(":");
  Serial.print(mac[2],HEX);
  Serial.print(":");
  Serial.print(mac[1],HEX);
  Serial.print(":");
  Serial.println(mac[0],HEX);
}

void listNetworks() {
  // poszukiwanie dostepnych sieci:
  Serial.println("** Scan Networks **");
  int numSsid = WiFi.scanNetworks();
  if (numSsid == -1)
 {
    Serial.println("Couldn't get a wifi connection");
    while(true);
  }

  // wyswietlenie spisu znalezionych sieci:
  Serial.print("number of available networks:");
  Serial.println(numSsid);

  // wyswietlenie numerow i nazw sieci:
  for (int thisNet = 0; thisNet<numSsid; thisNet++) {
    Serial.print(thisNet);
    Serial.print(") ");
    Serial.print(WiFi.SSID(thisNet));
    Serial.print("\tSignal: ");
    Serial.print(WiFi.RSSI(thisNet));
    Serial.print(" dBm");
    Serial.print("\tEncryption: ");
    printEncryptionType(WiFi.encryptionType(thisNet));
  }
}

void printEncryptionType(int thisType) {
  // odczyt sposobu zabezpieczenia i wyswietlenie nazwy:
  switch (thisType) {
    case ENC_TYPE_WEP:
      Serial.println("WEP");
      break;
    case ENC_TYPE_TKIP:
      Serial.println("WPA");
      break;
    case ENC_TYPE_CCMP:
      Serial.println("WPA2");
      break;
```

```
    case ENC_TYPE_NONE:
      Serial.println("None");
      break;
    case ENC_TYPE_AUTO:
      Serial.println("Auto");
      break;
  }
}
```

Dodatek K
Opracowywanie bibliotek dla Arduino

Sposób przygotowywania bibliotek dla Arduino przedstawiono na na przykładzie biblioteki nadającej znaki alfabetem Morse'a za pomocą znajdującej się na płytce diody świecącej. Opracowanie biblioteki rozpoczyna się od przygotowania programu zapewniającego wymaganą funkcjonalność a następnie zostaje on przetworzony na bibliotekę o standardowym formacie. Tak przygotowana biblioteka może być wykorzystywana w wielu programach i w razie jej udoskonalania i rozbudowy w wygodny sposób udostępnia nową funkcjonalność. Przykład pochodzi z oficjalnej dokumentacji dostępnej w witrynie Arduino.
Pierwszym krokiem jest przygotowanie programu nadającego tekst alfabetem Morse'a:

```
int pin = 13;                          // linia, do ktorej podlaczona jest dioda swiecaca na plytce

void setup()
{
  pinMode(pin, OUTPUT);                // przelaczenie na wyjscie
}

void loop()
{                                      // nadawanie SOS za pomoca migania diody
  dot(); dot(); dot();
  dash(); dash(); dash();
  dot(); dot(); dot();
  delay(3000);
}

void dot()                             // nadawanie kropki
{
  digitalWrite(pin, HIGH);
  delay(250);
  digitalWrite(pin, LOW);
delay(250);
}

void dash()                            // nadawanie kreski
{
  digitalWrite(pin, HIGH);
  delay(1000);
  digitalWrite(pin, LOW);
  delay(250);
}
```

Ten prosty ćwiczebny program nadaje po uruchomieniu tekst SOS za pomocą diody świecącej podłączonej do kontaktu 13. Zawiera on kilka części, które mogą być przeniesione do biblioteki. Należą do nich w pierwszym rzędzie funkcje dot() i dash() – służące odpowiednio do nadawania kropek i kresek alfabetu Morse'a. Należy do nich także zmienna ledPin decydująca o tym, który kontakt jest używany jako wyjście oraz funkcja pinMode() służąca do definiowania wejść i wyjść logicznych – w tym przypadku wyjścia nadawczego.
Przetworzenie programu na bibliotekę wymaga przygotowania dwóch plików: nagłówkowego (z rozszerzeniem .h) i źródłowego (z rozszerzeniem .cpp). Plik nagłówkowy zawiera deklaracje wszystkich składników biblioteki a plik źródłowy ich kod.
Ponieważ biblioteka ma nosić nazwę "Morse" więc plik nagłówkowy musi mieć nazwę *Morse.h*. Plik w naszym przykładzie ma następującą zawartość:

```
class Morse
{
public:
  Morse(int pin);
  void dot();
  void dash();
private:
  int _pin;
};
```

Zadeklarowana w nim klasa stanowi zbiór funkcji (metod) i zmiennych (właściwości) złożonych w jedną całość i podzielonych na dwie kategorie: publicznych (*public*) czyli dostępnych do użytku z zewnątrz i prywatnych (*private*) nie widocznych na zewnątrz i dostępnych tylko dla funkcji należących do tej klasy. W innych językach występuje większa liczba kategorii a ich właściwości związane są m.in. z dziedzicznością elementów tej klasy przez klasy pochodne od niej.
Każda z klas zawiera specjalną funkcję zwaną konstruktorem, której zadaniem jest inicjalizacja obiektu stanowiącego wcielenie danej klasy. Konstruktor nosi ta samą nazwę co klasa i nie dostarcza żadnego wyniku.
Oprócz tego plik nagłówkowy zawiera polecenia *#include* włączające pliki definiujące standardowe typy i stałe stosowane w języku Arduino. Typy te i stałe są automatycznie dostępne w zwykłych programach ale nie w bibliotekach.
Przykład:
#include "Arduino.h"
Oprócz tego przyjęło się ujmować całość w ramy następującej konstrukcji:

```
#ifndef Morse_h
#define Morse_h

// #include i pozostała treść

#endif
```

Zapobiega to problemom mogącym wyniknąć z wielokorotnego włączenia tej samej biblioteki do programu. Na początku pliku nagłówkowego umieszczana jest informacja o autorze, wersji, dacie powstania i krótki opis funkcji.

Pełny plik nagłówkowy może więc wyglądać następująco:

```
/*
Morse.h – biblioteka dla świetlnej sygnalizacji alfabetem Morse‘a.
Autor David A. Mellis, 2 listopada 2007.
Program ogólnie dostępny.
*/
#ifndef Morse_h
#define Morse_h

#include "Arduino.h"

class Morse
{
public:
  Morse(int pin);
  void dot();
  void dash();
private:
  int _pin;
};
```

```
#endif
```
Drugi z plików zawiera właściwy kod źródłowy programu i nosi w tym przykładzie nazwę *Morse.cpp*. Na jego początku znajdują się ewentualne polecenia włączenia potrzebnych nagłówków i definicji:
```
#include "Arduino.h"
#include "Morse.h"
```
Po nich następuje konstruktor inicjalizujący obiekt danej klasy po jego utworzeniu w programie. W tym przykładzie inicjalizuje on pożądane wyjście i zapisuje jego numer w prywatnej zmiennej:
```
Morse::Morse(int pin)
{
pinMode(pin, OUTPUT);
_pin = pin;
}
```

Prefiks “Morse::” poprzedzający nazwę funkcji oznacza, że należy ona do klasy Morse. W ten sam sposób zapisywane są też wszystkie pozostałe funkcje klasy. Zmiena prywatna może mieć oczywiście dowolną nazwę – pod warunkiem, że będzie ona zgodna z podaną w nagłówku – ale tutaj dla ułatwienia nadano jej nazwę zmiennej publicznej – w tym przypadku argumentu funkcji – poprzedzoną podkreślnikiem (_pin). Sposób ten jest często stosowany w nazewnictwie zmiennych prywatnych.
W dalszym ciągu następuje kod pozostałych funkcji przejętych z programu wyjściowego:
```
void Morse::dot()
{
digitalWrite(_pin, HIGH);
delay(250);
digitalWrite(_pin, LOW);
delay(250);
}

void Morse::dash()
{
digitalWrite(_pin, HIGH);
delay(1000);
digitalWrite(_pin, LOW);
delay(250);
}
```
Od funkcji wyjściowych różnią się one prefiksem “Morse::” w nazwach i użyciem zmiennej prywatnej “_pin” zamiast “pin”.

Na początku pliku umieszcza się zwykle informacje o programie, wersji, dacie powstania, autorze itp.
```
/*
Morse.cpp - biblioteka dla świetlnej sygnalizacji alfabetem Morse‘a..
Autor David A. Mellis, 2 listopada 2007.
Program ogólnie dostepny.
*/

#include "Arduino.h"
#include "Morse.h"

Morse::Morse(int pin)
{
pinMode(pin, OUTPUT);
_pin = pin;
}

void Morse::dot()
```

```
{
digitalWrite(_pin, HIGH);
delay(250);
digitalWrite(_pin, LOW);
delay(250);
}

void Morse::dash()
{
digitalWrite(_pin, HIGH);
delay(1000);
digitalWrite(_pin, LOW);
delay(250);
}
```

W celu udostępnienia biblioteki w środowisku programistycznym należy utworzyć katalog (folder) *Morse* w katalogu *libraries* zawartym z kolei w katalogu *sketchbook*. Do katalogu Morse należy następnie skopiować oba pliki: *Morse.h* i *Morse.cpp*. Po uruchomieniu środowiska należy skompilować bibliotekę posługując się punktem *Sketch* | *Import Library* (“Program” | “Importuj bibliotekę”).
W razie gdyby proces kompilacji nie przebiegł prawidłowo lub nie został wogóle wykonany warto sprawdzić czy pliki mają prawidłowe rozszerzenia *.h* i *.cpp* bez jakichkolwiek dodatków i czy nazwy są zapisane prawidłowo.
Program sygnalizujący SOS z wykorzystaniem biblioteki może więc wyglądać następujaco:

```
#include <Morse.h>

Morse morse(13);

void setup()
{
}

void loop()
{
morse.dot(); morse.dot(); morse.dot();
morse.dash(); morse.dash(); morse.dash();
morse.dot(); morse.dot(); morse.dot();
delay(3000);
}
```

W porównaniu z programem podanym na początku rozdziału różni się on wprowadzeniem polecenia #include na początku co udostępnia bibliotekę Morse. Następnie tworzony jest obiekt morse typu klasy Morse:
Morse morse(13);
W linii tej następuje (jeszcze przed wywołaniem funkcji setup()) inicjalizacja obiektu za pomocą konstruktora klasy z argumentem określającym numer wyjścia.
Funkcja setup() jest w tym przykładzie pusta. Wywołania funkcji (metod) klasy są poprzedzone nazwą obiektu je typu a więc np. morse.dot().
W bardziej rozbudowanym programie może wystąpić kilka oiektów tego typu o różniacych się nazwach a więc np.
Morse morse(13);
Morse morse2(12);
W każdym z nich numer używanego wyjścia jest zapamiętany w ich prywatnej zmiennej _pin. Wywołanie funkcji morse2.dot() spowoduje wydanie sygnału na wyjściu 12 (zmienna _pin ma wartość 12).

W obecnej wersji środowiska funkcje tak utworzonej biblioteki nie są wyróżniane innym kolorem (co byłoby wygodne dla programisty).

Aby temu jakoś zaradzić można założyć (w katalogu *Morse*) dodatkowy plik tekstowy keywords.txt o nastepującej (dla tego przykładu) treści:
Morse NAZWA1
dash NAZWA2
dot NAZWA2
Każda z linii zawiera nazwę samej klasy lub elementu klasy i jakąś inną przypisaną im nazwę oddzieloną od pierwszej tabulatorem (nie może to być znak odstępu). NAZWA1 określa klasy, NAZWA2 – funkcje. Dzięki temu klasy wyświetlane są w tekście programu w kolorze pomarańczowym a funkcje – w kolorze brązowym. Zawartość pliku jest używana dopiero po ponownym uruchomieniu środowiska.
Zalecane jest też przygotowanie krótkiego przykładowego programu ilustrującego sposób korzystania z nowoutworzonej klasy. Korzystne jest umieszczenie w tym przykładzie dodatkowych możliwie wyczerpujących komentarzy. Przykład ten (lub przykłady) powinny znajdować się w katalogu *examples* leżącym w katalogu biblioteki (tutaj w katalogu *Morse*).
Do wywołania przykładu służy menu *File* | *Sketchbook* | *Examples* (“Plik” | “Katalog programów” | “Przykłady”).

W serii „Biblioteka polskiego krótkofalowca” dotychczas ukazały się:

Nr 1 – „Poradnik D-STAR”
Nr 2 – „Instrukcja do programu D-RATS”
Nr 3 – „Technika słabych sygnałów” Tom 1
Nr 4 – „Technika słabych sygnałów” Tom 2
Nr 5 – „Łączności cyfrowe na falach krótkich” Tom 1
Nr 6 – „Łączności cyfrowe na falach krótkich” Tom 2
Nr 7 – „Packet radio”
Nr 8 – „APRS i D-PRS”
Nr 9 – „Poczta elektroniczna na falach krótkich” Tom 1
Nr 10 – „Poczta elektroniczna na falach krótkich” Tom 2
Nr 11 – „Słownik niemiecko-polski i angielsko-polski” Tom 1
Nr 12 – „Radiostacje i odbiorniki z cyfrową obróbką sygnałów” Tom 1
Nr 13 – „Radiostacje i odbiorniki z cyfrową obróbką sygnałów” Tom 2
Nr 14 – „Amatorska radioastronomia”
Nr 15 – „Transmisja danych w systemie D-STAR”
Nr 16 – „Amatorska radiometeorologia”
Nr 17 – „Radiolatarnie małej mocy”
Nr 18 – „Łączności na falach długich”
Nr 19 – „Poradnik Echolinku”
Nr 20 – „Arduino w krótkofalarstwie” Tom 1
Nr 21 – „Arduino w krótkofalarstwie” Tom 2