BIBLIOTEKA
POLSKIEGO KRÓTKOFALOWCA 17

Krzysztof Dąbrowski
OE1KDA

Radiolatarnie małej mocy

Wiedeń 2013

Radiolatarnie małej mocy

Krzysztof Dąbrowski OE1KDA

Wydanie 1
Wiedeń, lipiec 2013

Spis treści

Wstęp

Krótkofalowcy od dawna uruchamiali radiolatarnie (ang. *beacon*) – automatycznie pracujące stacje nadawcze – pozwalające na obserwację i badanie warunków propagacji fal w różnych zakresach częstotliwości a także ułatwiające zauważenie otwarć pasm spowodowanych zjawiskami występującymi sporadycznie i trudnymi do przewidzenia j.np. występowanie warstwy Es, duktów troposferycznych albo zorzy polarnej.
Radiolatarnie automatyczne o możliwie dużym zasięgu są instalowane przez kluby i związki krótkofalowców. Ich uruchomienie wymaga koordynacji częstotliwości pracy w skali krajowej lub międzynarodowej i uzyskania oddzielnego zezwolenia radiowego (licencji).
Natomiast radiolatarnie bardzo małej mocy pracujące pod nadzorem operatora nie wymagają ani koordynacji częstotliwości ani własnych licencji. Przeważnie pracują one z QTH operatora w przewidzianych do tego celu i ogólnie znanych wycinkach pasm, ale ponieważ prawdopodobieństwo spowodowania przez nie zakłóceń w pracy innych użytkowników pasm jest znikome mogą one pracować na dowolnych częstotliwościach amatorskich w podzakresach przewidzianych dla używanej przez nie emisji.
Prywatne radiolatarnie tego rodzaju służą przeważnie również do badania warunków propagacji – stąd też wzięła się ich angielska nazwa ***M**anned **E**xperimental **P**ropagation **T**ransmitter*, w skrócie MEPT – ale można je wykorzystywać też do transmisji różnego rodzaju danych telemetrycznych lub meteorologicznych, ostrzeżeń itp. albo badania przydatności różnych emiesji w określonych warunkach propagacyjnych.
Obecnie najbardziej znanym i rozpowszechnionym rozwiązaniem są radiolatarnie pracujące emisją WSPR ale stacje pracujące wolną telegrafią QRSS, systemem Slowhell, JT65, JT4 lub innymi emisjami o małej szybkości transmisji i szerokości pasma były uruchamiane już przedtem na falach długich, krótkich, w paśmie 50 MHz i powyżej. Do innych często stosowanych emisji należą zwykła telegrafia (z szybkością pozwalającą na odbiór słuchowy), PSK31, RTTY i dalekopis Hella. Stacje eksperymentalne pracowały także innymi mniej rozpowszechnionymi rodzajami emisji. W niektórych rozwiązaniach transmisja odbywała się naprzemian kilkoma emisjami przy czym do najczęstszych kombinacji należała transmisja naprzemian zwykłą telegrafią i jednym z wymienionych powyżej rodzajów emisji o małej przepływności.
Systemy o małej przepływności umożliwiają odbiór nawet przy bardzo niekorzystnych stosunkach poziomu sygnału do szumów i zakłóceń ale ilość przekazywanej informacji jest bardzo ograniczona ze względu na czas konieczny do jej nadania. Przeważnie nadawany bywa tylko znak wywoławczy a czasem tylko jego część z dodatkiem co najwyżej kwadratu lokatora lub mocy nadajnika (o ile jest ona tak niska, że warto zwrócić na nią uwagę odbiorców).
Radiolatarnie pracujące emisjami o przeciętnej przepływności pozwalają natomiast na transmisję różnego rodzaju danych telemetrycznych lub meteorologicznych oprócz znaku i QTH.
Przeważnie moce nadajników nie przekraczają kilku W ale wiele stacji pracuje mocami kilkuset lub tylko kilkudziesięciu mW a niektóre nawet 1 mW i poniżej.
Szczegółowe informacje na temat stosowanych rodzajów emisji i ich właściwości znajdą czytelnicy w tomach poświęconych technice słabych sygnałów i łącznościom cyfrowym na falach krótkich wchodzących w skład niniejszej serii.
W najprostszym przypadku na wyposażenie radiolatarni składa się dowolna radiostacja KF lub UKF mogąca pracować telegrafią i SSB i pozwalająca na odpowiednie obniżenie mocy nadawania oraz komputer wyposażnony w program terminalowy do pracy emisjami cyfrowymi. Nie jest to jednak rozwiązanie optymalne z punku widzenia bilansu energetycznego stacji. Najczęściej spotykane radiostacje SSB o mocach 100 – 200 W pozwalają wprawdzie na obniżenie mocy wyjściowej do kilku W ale przy znacznym pogorszeniu sprawności. Moc strat przewyższa więc wielokrotnie pożądaną moc użyteczną. Dalsze obniżenie mocy wyjściowej można osiągnąć jedynie dzięki dodatkowym tłumikom co oznacza, że wypadkowa sprawność będzie leżała poniżej wszelkiej krytyki. Po doliczeniu zużycia energii przez komputer widać od razu, że nie tędy droga. Poza tym szkoda do takiego celu blokować drogą radiostację i komputer.
Radiolatarnia QRP lub QRPP (w dalszym ciągu dla uproszczenia zwana najczęściej radiolatarnią QRP) powinna się więc składać ze specjalnie do tego celu skonstruowanego nadajnika małej mocy i elektronicznego układu kluczującego zastępującego komputer. Przy obecnym stanie rozwoju techniki są to praktycznie układy mikrokomputerowe a spośród nich jednym z wygodniejszych w użyciu jest

mikrokomputer „Arduino” w dowolnym wydaniu. Autor nie ogranicza się jednak wyłącznie do prezentacji rozwiązań opartych o „Arduino” i w dalszej części opracowania będą reprezentowane również konstrukcje oparte o inne typy mikroprocesorów – zwłaszcza z rodziny PIC. Znaczną część rozwiązań stanowią układy kluczujące nadajniki ale w części z nich generowana jest podnośna akustyczna kluczowana lub zmodulowana bardziej skomplikowanymi sygnałami cyfrowymi. Część z nich stanowią także układy sterujące syntezery cyfrowe dostarczające od razu sygnału w.cz. dla pożądanej emisji lub sygnału m.cz. do wysterowania nadajnika SSB.
Ze względu na prostotę układu największym powodzeniem cieszą się nadajniki telegraficzne kluczowane amplitudowo lub częstotliwościowo co rzutuje także na wybór emisji ale zasadniczo konstrukcja prostego nadajnika SSB małej mocy nie przekracza możliwości średnio zaawansowanego krótkofalowca. Wąskie pasmo zajmowane przez stosowane w radiolatarniach sygnały cyfrowe oznacza, że nie muszą być to nawet nadajniki wyposażone w filtry kwarcowe a wystarczą układy pracujące metodą fazową i wyposażone w stosunkowo proste (bo wąskopasmowe) przesuwniki fazy m.cz.
Prywatne radiolatarnie QRP nie są czynne naogół przez dłuższy czas bez przerwy a jedynie w wybranych odcinkach czasu: wieczorami przez kilka godzin, w dni wolne od pracy (operatora), pod koniec tygodnia czasem przez cały dzień lub kilka dni pod rząd. Do wyjątków należały jak dotąd radiolatarnie pracujące przez kilka miesięcy albo dłużej ale jest to oczywiście sprawa decyzji ich operatorów.
Przed uruchomieniem radiolatarni nawet tylko na dzień czy dwa warto rozpowszechnić informacje o tym fakcie np. w grupie dyskusyjnej *cnts.be/mailman/listinfo/knightsqrss_cnts.be* albo na witrynie *www.wsprnet.org*. Zwiększa to prawdobodobieństwo zauważenia aktywności przez innych zainteresowanych a zatem i otrzymania potwierdzeń odbioru – elektronicznych lub kart QSL.
Można także obserwować własne sygnały na publicznych odbiornikach internetowych (ang *grabber*) dostrojonych do podzakresów QRSS i zbliżonych. Jeden ze spisów takich odbiorników znajduje się w witrynie *http://digilander.libero.it/i2ndt/grabber/grabber-compendium.htm*.

Krzysztof Dąbrowski OE1KDA
Wiedeń
Lipiec 2013

Wprowadzenie

Budowa i uruchamianie radiolatarni małej i bardzo małej mocy (QRP i QRPP) służących do badania warunków propagacji stały się w ostatnich latach dość popularne w niektórych krajach zachodnich i to jeszcze przed rozpowszechnieniem się emisji WSPR. Największą jednak popularność zyskały sobie takie radiolatarnie we Włoszech. Tamtejsi krótkofalowcy przeprowadzali i przeprowadzają szereg eksperymentow z nadajnikami o bardzo prostej konstrukcji kluczowanymi amplitudowo lub czestotliwościowo wolną telegrafią QRSS o długości kropki wynoszącej przeważnie 3, 6 lub 10 sekund, czasami także 1 sekundę. Najczęściej pracują one w paśmie 28322 kHz lub jego podwielokrotnych. Ze względu na małą szybkość transmisji zamiast pełnego znaku wywoławczego nadawany jest jedynie sufiks albo inna dowolna, wybrana przez operatora kombinacja liter.
Pomysł ten może być interesujący także dla krótkofalowców polskich i oczywiście możliwe jest rozszerzenie go na inne rodzaje emisji i zakresy fal. Zaprezentowane dalej pomysły i konstrukcje nie są pomyślane jako konkurencja dla WSPR a jako możliwość wzbogacenia naszego hobby. Niektóre z rozwiązań pozwalają także na rozsyłanie danych telemetrycznych co nie jest możliwe w systemie WSPR.

Rodzaje emisji

Do najprostszych rozwiązań nadajników zaliczają się nadajniki telegraficzne kluczowane w amplitudzie lub częstotliwości. Fakt ten przesądza w znacznym stopniu o popularności emisji stosowanych w radiolatarniach QRP. Do najczęściej stosowanych w nich emisji należą więc telegrafia ze standardową szybkością i telegrafia wolna QRSS, dalekopisy (RTTY), różne warianty systemu dalekopisowego Hella i emisje pseudograficzne.
Dzięki rozpowszechnieniu syntezerów cyfrowych (ang. *DDS*) stosunkowo łatwo można przy użyciu mikroprocesorów generować także bardziej skomplikowane wielostanowe sygnały kluczowane częstotliwościowo j.np. dla emisji WSPR, JT65, JT2, JT4 itp. Oznacza to, że również i w takich rozwiązaniach można zrezygnować ze stosowania komputerów PC.
Miniaturowe PC w rodzaju „Raspberry Pi” i podobnych mogą jednak w najbliższym czasie stać się ponownie popularnym źródłem sygnałów m.cz. różnych emisji zastępując „Arduino” i spółkę.

Telegrafia

Zastosowanie telegrafii Morse’a z przeciętną, pozwalająca na odbiór słuchowy szybkością jest rozwiązaniem oczywistym i nasuwającym się od samego początku wielu konstruktorom radiolatarni nie tylko małej mocy. W porównaniu z fonią uzyskiwane są większe zasięgi nawet w trudnych warunkach a sygnał telegraficzny może być odbierany na słuch. Jednak obniżenie szybkości transmisji dające zawężenie pasma zajmowanego przez nadawany sygnał oznacza, pod warunkiem dostosowania szerokości pasma przenoszenia filtru po stronie odbiorczej, obniżenie stosunku sygnału do szumów i zakłóceń niezbędnego dla prawidłowego zdekodowania sygnału odbieranego.

Tabela 1.1. Optymalna szerokość pasma i różnica stosunku sygnału do zakłóceń w odniesieniu do telegrafii z szybkością transmisji 12 słów/min.

Szybkość transmisji [sł./min] lub długość elementu	Optymalna szerokość pasma [Hz]	Zysk w stosunku do szybkości 12 sł./min [dB]	Porównanie mocy
12	10	0	5 W
8	6,67	+1,8	3,3 W
4	3,33	+4,8	1,3 W
1 sek. długość kropki	1	+10	500 mW
3 sek. długość kropki	0,33	+14,8	133 mW
10 sek. długość kropki	0,1	+20	50 mW

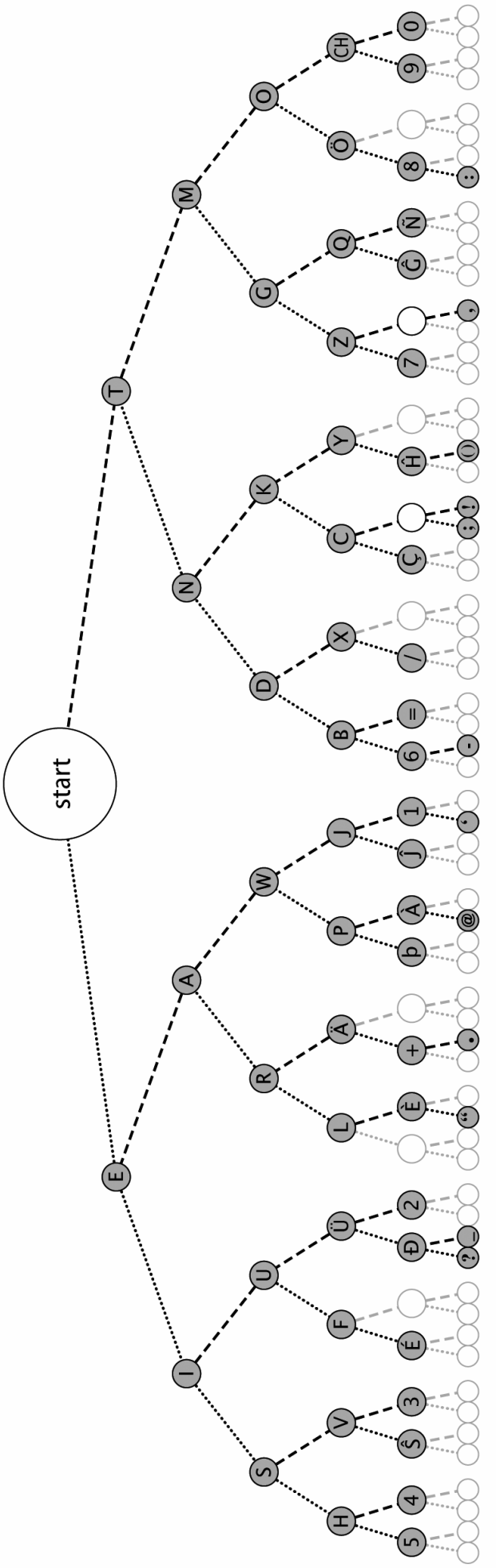
Jedną z częściej więc stosowanych przez radiolatarnie emisji jest powolna telegrafia QRSS, w której czas trwania kropki wynosi od jednej do kilkudziesięciu sekund. Odpowiednio do czasu trwania kropki

w oznaczeniu emisji na końcu podawana jest w postaci liczbowej jej długość w sekundach – i tak np. QRSS3 oznacza telegrafię z długością kropki wynoszącą 3 sekundy (długość kreski wynosi więc 9 sekund, również długości odstępów odpowiadają ogólnie znanej normie), QRSS10 – 10 sek. itd. Oznaczenie pochodzi od znanego skrótu QRS oznaczającego zwolnienie szybkości telegrafowania a dodatkowa litera „S" sygnalizuje znaczne obniżenie szybkości.
Jak wiadomo poprawę stosunku sygnału użytecznego do szumów i zakłóceń można uzyskać zawężając pasmo przenoszenia odbiornika co wymaga oczywiście odpowiedniego ograniczenia pasma transmitowanego sygnału. Konsekwencją tego wymogu jest obniżenie szybkości transmisji co nie tylko w warunkach amatorskich nie stanowi istotnego mankamentu.
Jako przybliżoną regułę obliczania pasma zajmowanego przez sygnał telegraficzny można przyjąć:
Szer [Hz] = 3 / długość kropki [sek].
Przy szybkości telegrafowania wynoszącej 12 słów/min (ang. *wpm*) długość kropki wynosi w przybliżeniu 1/10 sek. co oznacza, że optymalną szerokością pasma jest 30 Hz. Filtr SSB o szerokości pasma 2400 Hz jest więc 80-krotnie szerszy niż byłoby to niezbędne i jak łatwo zauważyć ponad 98% pasma zajmują sygnały niepożądane w danym momencie – czyli stosunek sygnału użytecznego do niepożądanych jest gorszy o ok. 19 dB. Dla telegrafii QRSS3 niezbędna szerokość pasma wynosi 0,33 Hz co daje zysk 14,8 dB w stosunku do telegrafii z szybkością 12 słów/min., a dla QRSS10 zysk ten wynosi już 20 dB. Radiolatarnie QRP stosują najczęściej wariant QRSS3, ale spotykane są też QRSS1 (oznaczane też tylko jako QRSS), QRSS6 a nawet QRSS10 i więcej. W niektórych przypadkach nadawany jest naprzemian sygnał telegraficzny ze zwykłą szybkością (pozwalającą na odbiór słuchowy, np. 6 albo 12 sł./min.) i QRSS albo też w kombinacji i z innymi rodzajami emisji.

W telegrafii dwuczęstotliwościowej (z kluczowaniem częstotliwości, tak że częstotliwość wyższa odpowiada kreskom a niższa kropkom) długości kropek i kresek są sobie równe i stosowane są odpowiednio oznaczenia DFCW dla elementów o czasie trwania 1 sekundy, DFCW3 – dla elementów o czasie trwania 3 sekund itd. Jednakowa długość obu elementów kodu powoduje efektywne skrócenie czasu trwania znaku a więc i całkowitego czasu transmisji.

Oprócz tych dwóch wariantów spotykany w eterze jest trzeci – będący zwykłą telegrafią z kluczowaniem częstotliwości zamiast amplitudy i noszący oznaczenie FSKCW lub FSCW. Również i tutaj na końcu podaje się cyfrę określającą długość kropki. W wariancie tym niższa częstotliwość odpowiada przerwom a wyższa elementom znaku. Dewiacja dla zasięgów kontynentalnych przy długościach kropek od jednej do kilku sekund wynosi przeważnie około 4 – 5 Hz (nie przekracza 10 Hz) podobnie jak dla DFCW. Dla zasięgów międzykontynentalnych gdzie długości kropek mogą dochodzić nawet 120 sekund stosowana jest dewiacja poniżej 1 Hz.

Szybkość transmisji w systemie QRSS jest tak niska, że niemożliwy jest odbiór telegrafii na słuch. Jedynym praktycznym sposobem odbioru jest zastosowanie komputera i wyświetlanie odbieranych znaków na jego ekranie. Rozwiązanie to ma jeszcze dodatkową zaletę – operatorzy nie znający telegrafii mają dosyć czasu na znalezienie znaku w tabeli i jego prawidłowe rozpoznanie. Bardzo wygodnym rozwiązaniem jest przedstawione na rys. 1.1 drzewko telegraficzne pozwalające na dekodowanie znaków alfabetu Morse'a przez poruszanie się po jego gałęziach w kierunku kropek lub kresek w miarę odbierania kolejnych elementów znaku.



Wzrokowe rozpoznawanie odbieranej informacji daje dodatkowe korzyści wynikające z wykorzystania inteligencji człowieka, który może prawidłowo rozpoznać obraz znaku nawet gdy jest on poważnie na zniekształcony przez zaniki i zakłócenia. W przypadku odbioru pisma w postaci graficznej (j.np. w emisji Hella) z pomocą przychodzi także naturalna redundancja liter, dzięki czemu można je rozpoznać nawet wówczas gdy widoczna jest tylko część. Niezależnie od tego dzięki redundancji mowy można często domyślić się brakującego znaku w oparciu o kontekst.

Do odbioru emisji QRSS i wyświetlania odbieranych sygnałów na ekranie komputera można zastosować bezpłatne programy „Argo" lub „Spectran" autorstwa Alberta di Bene I2PHD, trochę bardziej skomplikowany w obsłudze ale i dający większe możliwości „Spectrum Lab" albo jedną z nowszych wersji programu MultiPSK. Dla Linuksa dostępny jest nadawczo-odbiorczy program „glfer" obsługujący emisje QRSS CW i DFCW.

Programy odbiorcze analizują odbierane sygnały przy użyciu Szybkiej Transformaty Fouriera (STF; ang. *Fast Fourier Transformation - FFT*) i wyświetlają ich przebieg w skali czasu i częstotliwości w postaci wskaźnika wodospadowego na ekranie komputera (w przypadku gdy oś czasu jest położona poziomo j.np. w programie „Argo" wskaźnik ten bywa nazywany także wskaźnikiem kurtynowym).

Podstawowym sposobem transmisji telegraficznej QRSS CW jest kluczowanie amplitudy identycznie jak w przypadku klasycznej telegrafii. Oprócz tego stosowane bywa kluczowanie częstotliwości sygnału w takt znaków alfabetu Morse'a z dewiacją od kilku do 10 Hz – FSCW (ang. *frequency shift CW*).

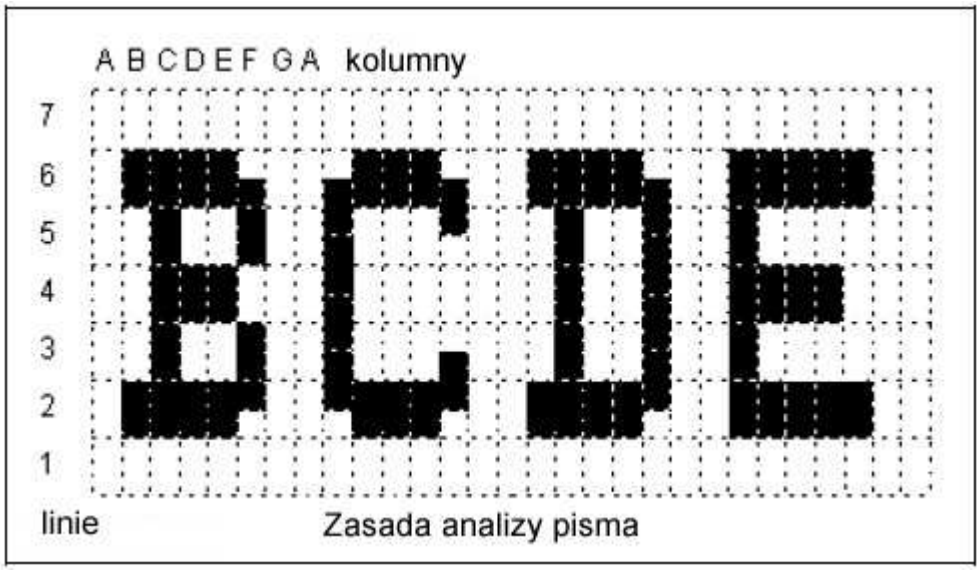
W obu przypadkach obowiązują standardowe zależności długości elementów znaku i odstępów, co oznacza, że długość kreski jest równa trzykrotnej długości kropki itd. Wyraźne skrócenie czasu transmisji – około 2,5 do 3 razy – daje natomiast system kluczowania częstotliwości DFCW (ang. *dual frequency CW*). W emisji tej kropkom jest przypisana niższa częstotliwość nadawanego syganału a kreskom – wyższa rys. 1.5). Dzięki temu możliwe jest przyjęcie jednakowego czasu trwania kropek i kresek i skrócenie odstępów pomiędzy nimi. Podobnie jak w przypadku FSCW dewiacja jest mała i nie przekracza 10 Hz. Analogicznie jak dla zwykłej telegrafii w oznaczeniu podawana jest długość elementu, a więc przykładowo dla DFCW3 czas trwania kropek i kresek wynosi po 3 sekundy. Krótkofalowcy brytyjscy stosują w niektórych radiolatarniach mikrofalowych system DFCWi. Występuje w nim trzecia częstotliwość – spoczynkowa – odpowiadająca przerwom międzyelementowym. Jest ona niższa od częstotliwości kropek.

Rys. 1.2. Telegrafia FSCW. Częstotliwość wyższa odpowiada elementom znaku a niższa – przerwom

Rys. 1.3. Telegrafia DFCW. Częstotliwość wyższa odpowiada kreskom a niższa kropkom. Brak sygnału w przerwach między elementami

Dalekopis Hella

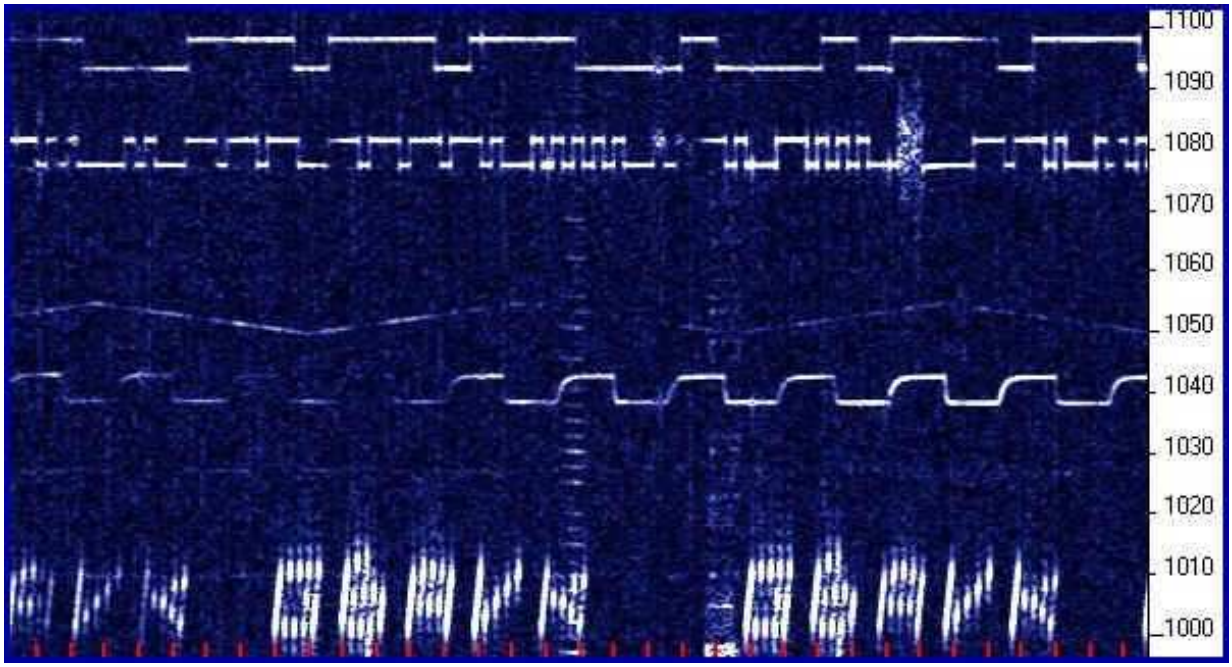
Oprócz telegrafii radiolatarnie QRP posługują się emisją Hella polegającą na graficznej transmisji znaków zamiast kodów komputerowych jak w PSK31 czy packet radio – jest to więc rodzaj transmisji faksymile. Do najczęściej stosowanych wariantów należą Feldhell, MT-Hell i Slowfeld.
Litery alfabetu są podzielone na elementy nadawane kolejno lub grupami. Zasadę podziału ilustruje rys. 1.4. W wersji podstawowej noszącej nazwę *Feldhell* pole (matryca) znaku jest podzielone na 7 x 7 elementów z tego po odjęciu marginesów dla litery pozostaje 5 x 5 elementów. Elementy są nadawane po kolei w kierunku od lewego dolnego rogu w górę i w prawo – czyli kolumny są transmitowane kolejno od dołu do góry. Szybkość transmisji wynosi 122,5 boda co odpowiada 17,5 kolumnom/sek. czyli 2,5 znaku/sek co się równa 150 zn./min (transmisja pojedyńczego elementu trwa ok. 8,163 msek.). Średni współczynnik wypełnienia wynosi tutaj ok. 21% co pozwala na pracę z pełną mocą nadajnika (dla porównania dla telegrafii Morse'a wynosi on ok. 46%). Cechą charakterystyczną systemu jest pochylenie odbieranych liter na wydruku w lewo spowodowane następowaniem po sobie kolejnych elementów znaku na tle przesuwającej się taśmy papierowej – obecnie ruch ten jest symulowany na ekranie komputera i wobec tego zachowano pochylenie wyświetlanych liter.



Rys. 1.4. Analiza pisma w systemie Feldhell z uwzględnieniem podziału na połówki elementów

Sygnał Hella może bezpośrednio kluczować amplitudę nośnej w.cz. identycznie jak w przypadku telegrafii. W radiolatarniach pracujących emisją Hella można więc stosować proste rozwiązania nadajników telegraficznych ze stopniami mocy w klasie C.
Po dokładnym przyjrzeniu się alfabetowi łatwo zauważyć, że niektóre elementy znaków rozpoczynają się w połowie pola lub też trwają o połowę pola dłużej. W zasadzie można więc także do analizy znaku przyjąć podział na 14 x 7 elementów (w sumie 98 elementów), przy czym dla ograniczenia szerokości pasma sygnału nadawane (czarne) elementy znaku nie mogą być krótsze niż dwa elementy matrycy (licząc w kierunku transmisji czyli pionowym – jest to tzw. reguła dwóch elementów lub dwóch punktów). Taki sposób podziału pozwala na uzyskanie ładniejszego wyglądu znaków. Stosowana w tej normie pseudo-szybkość transmisji 122,5 boda jest osiągana dzięki regule dwóch elementów. Formalnie rzecz biorąc szybkość transmisji wynosi 245 bodów (długość połówki elementu wynosi ok. 4,08 ms) ale nie jest całkowicie wykorzystana co w efekcie daje ograniczenie o połowę szerokości pasma sygnału.
Transmisja nie wymaga dokładnej synchronizacji a jedynie zgrubnego wyrównania prędkości po stronach nadawczej i odbiorczej z dokładnością do poniżej 1 %. W przypadku pełnej zgodności prędkości znaki są wyświetlane na ekranie poziomo natomiast powstanie odchyłki powoduje wyświetlanie pisma ukosem w górę (gdy podstawa czasu odbiornika jest szybsza od podstawy nadajnika) lub w dół (gdy podstawa czasu odbornika jest wolniejsza).
Dzięki zastosowaniu kluczowania telegraficznego i wykorzystaniu ludzkiej inteligencji w procesie dekodowania system ten ma w trudnych warunkach odbioru (lub przy małych mocach nadawania) zalety z grubsza zbliżone do zalet telegrafii Morse'a.
Oprócz normy podstawowej w pracy QRP stosuje się albo transmisję *Feldhell* z 1/8 standardowej szybkości (*Slowhell*) albo opracowaną przez G3PPT odmianę pod nazwą *Slowfeld* (skr. od *Slow Feldhell*) różniącą się od podstawowej znacznie niższą – około 100-krotnie – szybkością transmisji (0,5 – 3 znaków/min, często używana – 1 element/s co daje ok. 1,5 zn./min.). W wariancie „Slowfeld" i w pozostałych wariantach wielotonowych MT-Hell (*Multi-tone Hell*) zamiast wspólnej częstotliwości podnośnej dla wszystkich elementów znaku stosuje się oddzielne podnośne dla każdej jego linii (nadajnik jest kluczowany – modulowany – częstotliwościowo). Rozróżniane są dwie odmiany wariantu MT-Hell: z kolejną transmisją linii – *S/MT-Hell* i z równoległą transmisją – *C/MT-Hell*. Stosowane są alfabety z podziałem znaku na 7 – 16 linii.
W normie *Slowfeld* różnice częstotliwości między kolejnymi liniami wynoszą 1 – 1,5 Hz przy czym dokładna wartość nie jest krytyczna ale nie może być mniejsza liczbowo od szybkości transmisji wyrażonej w bodach. Przy transmisji jednego elementu na sekundę różnica częstotliwości między kolejnymi liniami nie może więc być mniejsza od 1 Hz – w przeciwnym przypadku elementy stają się rozmyte a litery trudno rozpoznawalne. Stosowany alfabet jest zasadniczo identyczny lub podobny do używanego w normie *Feldhell* ale odczyt w dziedzinie częstotliwości pozwala na stosowanie dowolnych innych krojów czcionek i sposobu ich podziału na elementy a nawet większej liczby linii przykładowo 14 lub 16 zamiast 7. Czas trwania każdego z elementów znaku leży najczęściej w granicach od 0,1 do 0,5 sekundy chociaż stosowane bywają czasy dłuższe od 1 do 5 sekund. Również i w przypadku tej emisji dobór parametrów transmisji jest dowolny i zależny od decyzji operatora. Jednym z typowych czasów trwania elementu jest 0,5 sekundy co odpowiada ok. 3 znakom (literom) na minutę.
Dzięki rozmieszczeniu elementów znaku w dziedzinie częstotliwości a nie czasu odbierany tekst jest bezpośrednio czytelny na wskaźnikach wodospadowych programów stosowanych w łącznościach QRSS takich jak „Argo", „Spectran", „Spectrum Lab" i podobnych (rys. 1.5). Ponieważ nie występują tutaj problemy z synchronizacją tekst odbierany nie musi być wyświetlany dwukrotnie.
Również sygnały dwóch innych wariantów Hella: FM-Hell i PSK-Hell mogą być stosunkowo łatwo generowane przez mikroprocesory nawet bez pomocy syntezerów częstotliwości – a zwłaszcza sygnały pierwszej z nich. Oba systemy pracują ze standardową szybkością jak w normie Feldhell (245 bodów) i stosowane czcionki mogą być identyczne (podział na 98 elementów) ale można także używać czcionek, w których nie stosuje się zasady dwóch punktów czyli dowolnie wykorzytujących wszystkie elementy. W normie PSK-Hell występuje różnicowe kluczowanie fazy podobnie jak w PSK31 przy czym elementom białym odpowiada zmiana fazy sygnału na początku elementu a czarnym – jej brak.
Norma FM-Hell korzysta z kluczowania częstotliwości z minimalną dewiacją (o indeksie 0,5) – MSK – Minimum Shift Keying – i z zachowaniem ciągłości fazy i z tego powodu powinna nosić właściwie oznaczenie MSK-Hell. Sygnał FM-Hell zajmuje nieco węższe pasmo niż dla PSK-Hella a częstotliwość

spoczynkowa odpowiadająca bieli leży powyżej częstotliwości czerni. Oprócz tego istnieją warianty o zmniejszonej szybkości transmisji równej 105 bodów posługujące się w związku z tym uproszczonym alfabetem – o znakach dzielonych na 42 (6 x 7) elementy.
Zarówno w normie PSK-Hell jak i FM-Hell – i to dla 245 i 105 bodów – wypadkowa szybkość transmisji jest identyczna jak dla Feldhella i wynosi 2,5 zn./s czyli 25 sł./min. (w przybliżeniu taką samą szybkość osiąga się w normie S/MT-Hell).
Do odbioru obu tych norm można korzystać z „MultiPSK" a do odbioru normy FM-Hell także z „Fldigi" (występuje w nim pod nazwą FSK-Hell).
Typowe szerokości pasm sygnałów wynoszą dla normy Feldhell 350 do 400 Hz, dla PSK-Hella przy 245 bodach – 245 Hz, dla FM-Hella przy 245 bodach – 125 Hz a dla 105 bodów odpowiednio 105 i 55 Hz. Dla S/MT-Hella typowe szerokości pasma leżą w okolicach 300 Hz.
Do transmisji w normach FM-Hell i S/MT-Hell wystarczy nadajnik telegraficzny kluczowany częstotliwościowo (ze stopniem mocy w klasie C) natomiast dla norm PSK-Hell i C/MT-Hell konieczny jest tor liniowy czyli nadajnik SSB modulowany podnośną akustyczną.
Graniczny stosunek poziomu sygnału do szumów i zakłóceń wynosi dla normy Feldhell ok. 6 – 8 dB, dla norm PSK-Hell i FM-Hell z przepływnościa 105 bodów ok. -12 dB a z przepływnością 245 bodów jest o ok. 3 dB gorszy. Pod względem przydatności do pracy radiolatarni warianty Hella można więc uprządkować następująco: Slowfeld, FM-Hell i Feldhell. W omówieniu pominięto warianty mało rozpowszechnione lub nieprzydatne do wymienionego celu co jednak nie wyklucza ich z dalszych eksperymentów.



Rys. 1.5. Odbiór sygnałów „Slowfeld". Powyżej widoczne sygnały FSCW oraz przebiegi: trójkątny i prostokątny

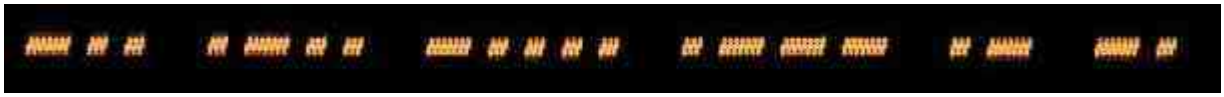
Inne systemy graficzne

Oprócz emisji Hella stosowana jest również transmisja graficznej reprezentacji znaków telegraficznych np. w postaci ukośnych kresek lub trójkątów. W przypadku pierwszym ukośne kreski opadające reprezentują kreski alfabetu Morse'a natomiast wznoszące – kropki (rys. 1.6). W reprezentacji przy użyciu trójkątów kreskom odpowiadają trójkąty zwrócone bokiem do góry, natomiast kropkom – z wierzchołkiem w górze (rys. 1.7). Spotyka się także reprezentację znaków telegraficznych za pomocą wężyków „generalskich" o różnej długości zamiast zwykłych kresek i kropek (FATCW).
Czasami zamiast informacji użytecznej transmitowane są sygnały prostokątne, piłokształtne lub inne (rys. 1.9). Źródłem modulacji w takich przypadkach są proste generatory odpowiedniego przebiegu na

obwodach LM555 lub innych. Transmisje takie są przeważnie zapowiadane w internecie ponieważ w przeciwnym przypadku niemożliwe byłoby rozpoznanie źródła pochodzenia sygnału.

Rys. 1.6. Reprezentacja znaków telegraficznych za pomocą ukośnych kresek

Rys. 1.7. Reprezentacja sygnałów telegraficznych za pomocą trójkątów



Rys. 1.8. Sygnał telegraficzny w postaci wężyków „generalskich”

Rys. 1.9. Przykład innego rodzaju sygnałów

Pozostałe emisje

Niektóre z amatorskich radiolatarni małej mocy posługują się także emisją PSK31 albo nawet jej wolniejszymi wariantami jak PSK10 czy PSK08 (tymi ostatnimi zwłaszcza na falach długich) lub RTTY z odstępem 170 Hz. Radiolatarnie posługujące się emisjami SSTV, MFSK8/16, MT63 itp. można zaliczyć do wyjątków.
Emisje o bardziej skomplikowanej strukturze sygnałów nie są praktycznie używane przez radiolatarnie QRP, pomimo, że są od dawna dostępne w popularnych programach terminalowych w rodzaju „Multi-PSK”, „Fldigi”, „Chip”, „ROS” itp. Jak dotąd brak jest odpowiednich programów do tego celu co wymagałoby samodzielnego zaprogramowania odpowiedniego rozwiązania a poza tym w większości przypadków konieczne byłoby użycie nadajnika SSB podczas gdy w konstrukcjach radiolatarni QRP obserwuje się dążenie do maksymalnego uproszczenia układów. W związku z tym przeważają w nich nadajniki telegraficzne kluczowane amplitudowo lub częstotliwościowo.
Zaprogramowanie generatora sygnałów różnych złożonych emisji może jednak stanowić interesujące wyzwanie dla programistów, pod warunkiem posiadania odpowiedniego doświadczenia. W internecie dostępnych jest wiele przykładów programów i podprogramów dla różnych typów procesorów, które można wykorzystać (po ewentualnej modyfikacji) we własnych opracowaniach. Przykłady te mają również dużą wartość dydaktyczną.
Od niedawna pojawiły się programy radiolatarni dla Arduino generujące sygnały WSPR i innych emisji (m.cz. bezpośrednio lub w.cz. za pośrednictwem syntezerów cyfrowych). Spotykane są też eksperymentalne rozwiązania oparte o emisje JT65, JT2 i JT4. Na falach krótkich i w paśmie 50 MHz stosowany jest wariant JT65A natomiast w pasmach 144 i 430 MHz – wariant JT65B a powyżej JT65C. Interesujące byłoby także uruchomienie w pasmach 50 i 70 MHz radiolatarni QRP pracujących emisją JT6M lub ISCAT.

Radiolatarnie pracujące emisjami WSPR lub należącymi do grupy WSJT wymagają dokładnej synchronizacji czasu. Konieczne jest więc użycie dodatkowego odbiornika GPS albo DCF-77 wraz z odpowiednim dekoderem. Prawidłowe zdekodowanie tych sygnałów przez odbiorców wymaga aby transmisja odbywała się zgodnie z przyjętym rastrem czasowym (z dokładnością do 1 – 2 sekund).
Na falach długich, średnich i 160 m od pewnego czasu spotykana jest eksperymentalna emisja WSPR-15. Odstęp częstotliwości zmniejszono w nim ośmiokrotnie do około 0,18 Hz a czas transmisji komunikatu wydłużono ośmiokrotnie do ok. 15 minut. Transmisja komunikatu rozpoczyna się w minutach 0, 15, 30 i 45 po pełnej godzinie. Treść komunikatów i sposób kodowania pozostają identyczne jak dla standardowej normy WSPR (oznaczanej w związku z tym czasami jako WSPR-2).

Częstotliwości pracy

Podane poniżej częstotliwości i podzakresy pracy stacji QRSS należy potraktować jako orientacyjne a w przypadku podania pojedyńczej częstotliwości należy przeszukiwać jej okolice – na początek najlepiej w trybie wyświetlania całego pasma. Głównym podzakresem, w którym można spotkać najczęściej i względnie największą liczbę stacji jest pasmo 10 MHz a konkretnie jego wycinek o szerokości 100 Hz położony tuż poniżej podzakresu WSPR. Zasadniczo zasada taka, chociaż jeszcze nie stosowana mogłaby zostać przyjęta i dla wszystkich innych pasm amatorskich. Ze względu na rozpowszechnienie systemu WSPR byłoby to znaczne ułatwienie dla operatorów stacji pracujących innymi emisjami zapewniającymi łączności pomiędzy stacjami bardzo małej mocy. W niektórych pasmach radiolatarnie spotyka się także w podzakresach leżących powyżej górnej granicy pasm WSPR.
Następnym interesującym ciągiem częstotliwości są podwielokrotne częstotliwości 28322 kHz. Jest on często stosowany przez prywatne radiolatarnie QRSS uruchamiane przez krótkofalowców włoskich. Okolice częstotliwości należących do wymienionego ciągu mogłyby stać się drugim standardem dla QRSS i pokrewnych emisji i to nie tylko dla radiolatarni ale także i dla prowadzenia łączności. W poniższym spisie ciąg ten oraz ciąg podzakresów leżących poniżej pasm WSPR podano wytłuszczoną czcionką. Pozostałe podzakresy bywały używane rzadziej lub częściej – wiązało się to także z dostępnością kwarców i ich kosztami – ale zdaniem autora powinny ustąpić miejsca wyróżnionym ciągom co ułatwiłoby orientację zarówno nadawcom (operatorom radiolatarni) jak i odbiorcom – zwłaszcza obecnie kiedy syntezery stały się łatwo dostępne i to po przystępnych cenach. Częstotliwości ciągu podwielokrotnych 28322 kHz należy traktować jako częstotliwości środkowe danych podzakresów.
Radiolatarnie QRSS są także czasami uruchamiane poza pasmami amatorskimi – w podzakresach przemysłowych takich jak 13560 kHz. Dzięki niskim mocom nadajników mogą one łatwo spełnić wymagania ustawowe dotyczące pasm przemysłowych i pracy w nich stacji nielicencjonowanych.

Najczęściej używane częstotliwości QRSS, FSCW, DFCW, Slowfeld i emisji graficznych (dane orientacyjne, najważniejsze ciągi jak WSPR i podwielokrotnych 28322 podano w postaci wytłuszczonej):
Długie fale 137600Hz – 137800Hz, najczęściej 137700Hz do 137750Hz,
440 lub 500 – 507 kHz w zależności od sytuacji prawnej w danym kraju.
476,175 kHz – w krajach gdzie dopuszczone jest pasmo 472 – 479 kHz
1837,9 – 1838,0 kHz (wycinek poniżej pasma WSPR)
1843,0 – 1843,1 kHz albo
1843,2 – 1843,3 kHz
1843,157 kHz (~1/16 częstotliwości 28322 kHz)
1919 kHz
3500,8 – 3500,9 kHz
3540,25 kHz (~1/8 częstotliwości 28322 kHz)
3575 kHz
3579 kHz (częstotliwość popularnych kwarców)
3585,0 – 3585,1 kHz
3588,0 – 3588,1 kHz
3593,9 – 3594,0 kHz (wycinek poniżej pasma WSPR)
3599,9 – 3600,0 kHz
7000,8 – 7000,9 kHz

7037,0 – 7037,1 kHz
7039,9 – 7040,0 kHz(wycinek poniżej pasma WSPR)
7040,2 – 7040,3 kHz (wycinek powyżej pasma WSPR)
7059,9 – 7060,0 kHz
7080,5 kHz (~1/4 częstotliwości 28322 kHz)
10139,5 kHz
10140,000 – 10140,100 kHz – główny i najczęściej używany podzakres (wycinek poniżej pasma WSPR)
14000,8 – 14000,9 kHz
14096,9 – 14097,0 kHz (wycinek poniżej pasma WSPR)
14098,9 – 14099,0 kHz
14161 kHz (~1/2 częstotliwości 28322 kHz)
14318,18 kHz
18068,8 – 18068,9 kHz
18096 kHz (częstotliwość kwarców QRP)
18105,0 – 18105,1 kHz,
18105,9 – 18106,0 kHz (wycinek poniżej pasma WSPR)
18108,9 – 18109,0 kHz
21000,8 – 21000,9 kHz
21095,9 – 21096,0 kHz (wycinek poniżej pasma WSPR)
21241,5 kHz (~3/4 częstotliwości 28322 kHz)
24890,8 – 24890,9 kHz
24912 kHz
24925,9 – 24926,0 kHz (wycinek poniżej pasma WSPR)
24928,9 – 24929,0 kHz
28000,8 – 28000,9 kHz
28125,9 – 28126,0 kHz (wycinek poniżej pasma WSPR)
28188 kHz
28322 kHz – w rzeczywistości pasmo 28321 – 28323 kHz – używane przez sieć prywatnych włoskich radiolatarni QRSS3 i QRSS10 o mocach nadawania od 10 mW do 1 W
50294,4 – 50294,6 kHz (pasmo WSPR)
13560 +/- 7 kHz – ogólnodostępne pasmo przemysłowe, należy zwrócić uwagę na aktualnie obowiązujące ograniczenia mocy dla stacji nie wymagających licencji – 10 mW. W paśmie nieamatorskim nie należy posługiwać się amatorskimi znakami wywoławczymi.

Częstotliwości radiolatarni Hella pracujących ze standardowymi szybkościami powinny leżeć w podzakresach używanych najczęściej do tego typu łączności, na ich skraju lub w bezpośrednim pobliżu, tak aby potencjalni odbiorcy mogli łatwiej zauważyć ich sygnały. Ta sama uwaga dotyczy także innych emisji. Nie należy pracować na mniej lub bardziej oficjalnych częstotliwościach wywoławczych ale można umieścić się w ich pobliżu.
Częstotliwości przewidziane dla dalekopisów Hella (używane także przez wiele innych emisji w tym też dla PSK31 i pokrewnych):
1840 kHz,
3574 – 3584 kHz,
7040 – 7044 kHz,
10140 – 10150 kHz (podawana jest też górna granica 10144 lub 10145 kHz),
14063 – 14080 kHz, wywoławcza 14074 kHz
18101 – 18107 kHz, wywolawcza 18104 kHz
21063 – 21080 kHz, wywoławcza 21074 kHz
24924 kHz,
28063 – 28080 kHz, wywoławcza 28074 kHz
28100 – 28110 kHz,
50,286 MHz.

Pasma WSPR (częstotliwości ustawione na skali lub wskaźniku radiostacji leżą 1400 Hz poniżej dolnych granic podzakresów – używana jest górna wstęga boczna USB):
137,400 – 137,600 kHz
137,600 – 137,625 kHz (WSPR-15)
475,600 – 475,800 kHz
475,800 – 475,825 kHz (WSPR-15)
1838,000 – 1838,200 kHz
1838,200 – 1838,225 kHz (WSPR-15)
3594,000 – 3594,200 kHz
5288,600 – 5288,800 kHz (tam gdzie dostępne jest pasmo 60 m)
7040,000 – 7040,200 kHz
10140,100 – 10140,300 kHz
14097,000 – 14097,200 kHz
18106,000 – 18106,200 kHz
21096,000 – 21096,200 kHz
24926,000 – 24926,200 kHz
28126,000 – 28126,200 kHz
50294,400 – 50294,600 kHz
70092,400 – 70092,600 kHz (tam gdzie dostępne jest pasmo 4 m)
144,490400 – 144,490600 MHz.

Układy nadajników

Małe moce wyjściowe nadajników radiolatarni pozwalają na znaczne uproszczenie ich konstrukcji co często stanowi dodatkowy bodziec do wykonania ich we własnym zakresie a samodzielna budowa nadajnika, nawet małej mocy, może być pouczająca i dać dużo radości i satysfakcji. Jest to tym bardziej słuszne, że sprzęt nadawczy większej mocy jest powszechnie dostępny na rynku i większość krótkofalowców nie ma powodów aby konstruować go samemu. Niskie moce wyjściowe, małe napięcia i prądy zasilania sprzętu QRP wyraźnie ułatwiają takie przedsięwzięcie. Nadajniki małej mocy wraz z mikroprocesorowym układem kluczującym mogą być zasilane z akumulatorów lub baterii słonecznych co ułatwia ich instalację w dowolnych innych dogodnych miejscach a nie tylko w stałym QTH.
Wymogi stawiane takim konstrukcjom nie są naogół wysokie. W większości przypadków wystarcza kwarcowa stabilność częstotliwości chociaż dla niektórych wąskopasmowych emisji korzystna albo nawet niezbędna okazuje się stabilizacja temperatury kwarcu. W niektórych przypadkach wystarcza jednak umieszczenie nadajnika albo tylko generatora sterującego w izolowanej termicznie obudowie (np. wyłożonej wewnątrz styropianem). Pasma przewidziane dla radiolatarni małej mocy są na tyle wąskie (przeważnie o szerokości 100 – 200 Hz), że konieczne jest staranne dostrojenie nadajnika do częstotliwości pracy przy użyciu dokładnego częstościomierza lub odbiornika komunikacyjnego.
Moce nadajników leżą przeważnie w zakresie od 100 mW do 1 W, rzadziej do kilku W. Część eksperymentatorów uruchamiała nawet nadajniki o mocach rzędu 10 mW, 1 mW i poniżej uzyskując, w zależności od pasma, interesujące i nie tak łatwo przewidywalne wyniki. W stopniach mocy pracują więc przeważnie tanie tranzystory złączowe lub polowe powszechnego użytku, rzadziej typowe tranzystory mocy w.cz. i tylko czasami w układach równoległych lub przeciwsobnych – co oznacza, że ewentualne niepowodzenia w trakcie ich uruchamiania nie pociągają za sobą dużych kosztów.
Dążenie do jak największego uproszczenia konstrukcji powoduje, że najczęściej stosowane są emisje polegające na zwykłym kluczowaniu amplitudy lub częstotliwości nadawanego sygnału, przy czym dewiacje częstotliwości wynoszą przeważnie tylko kilka Hz. Znacznie rzadziej spotyka się kluczowanie fazy lub różnego rodzaju modulacje złożone stanowiące kombinację kilku rodzajów kluczowania i ewentualnie także większej liczby podnośnych. Wymagają one liniowego toru nadawczego a więc w praktyce nadajnika SSB.
Do popularnych konstrukcji nadajników należą kilkutranzystorowe nadajniki telegraficzne sterowane kwarcem i kluczowane amplitudowo albo także częstotliwościowo za pomocą diody pojemnościowej. Spotyka się też układy nadajników pracujących na obwodach logicznych z serii 74HC. Jednym z często stosowanych rozwiązań jest układ oparty na obwodzie scalonym 74HC240, w którym jedna z bramek pracuje jako generator kwarcowy lub separator a wzmacniacz mocy składa się z czterech bramek połączonych równolegle. Nadajnik taki może dostarczyć do anteny od kilkudziesięciu do ponad 100 mW mocy ponieważ w praktyce bramki te mogą pracować bez uszkodzenia przy napięciach zasilania 7 – 8 V. Górna częstotliwość graniczna (szybkość przełączania) i dopuszczalne napięcie zasilania różnią się dla obwodów poszczególnych producentów.
Wadą takiego rozwiązania jest jednak twarde kluczowanie amplitudy sygnału powodujące stuki na początku i końcu każdego z elementów znaku. Wady tej nie mają nadajniki tranzystorowe w klasycznych układach, ponieważ łatwiejdobrać w nich takie czasy narastania i opadania sygnału aby uniknąć wytwarzania niepożądanych zakłóceń.
Do kluczowania częstotliwości nadajników kwarcowych stosowane są często zamiast warikapów spolaryzowane w kierunku przewodzenia a czasami też wstecznie czerwone i zielone diody elektroluminescencyjne (świecące).
Pomimo, że w dalszej części rozdziału przedstawiono układy nadajników z pojedyńczym generatorem kwarcowym najczęściej spotykane kwarce dają się zastosować jedynie w niektórych pasmach. W pozostałych konieczne może być mieszanie częstotliwości pochodzących z dwóch generatorów, o ile zamówienie specjalnego kwarcu na wybraną częstotliwość jest trudne albo nieopłacalne.
W tabeli 2.1 podano niektóre praktyczne kombinacje częstotliwości popularnych kwarców do różnych zastosowań (kwarców komputerowych, zegarowych, nadawczych QRP itp.). Jeden z dwóch generatorów musi być oczywiście przestrajany (VXO).

Tabela 2.1. Schematy przemiany dla niektórych pasm amatorskich

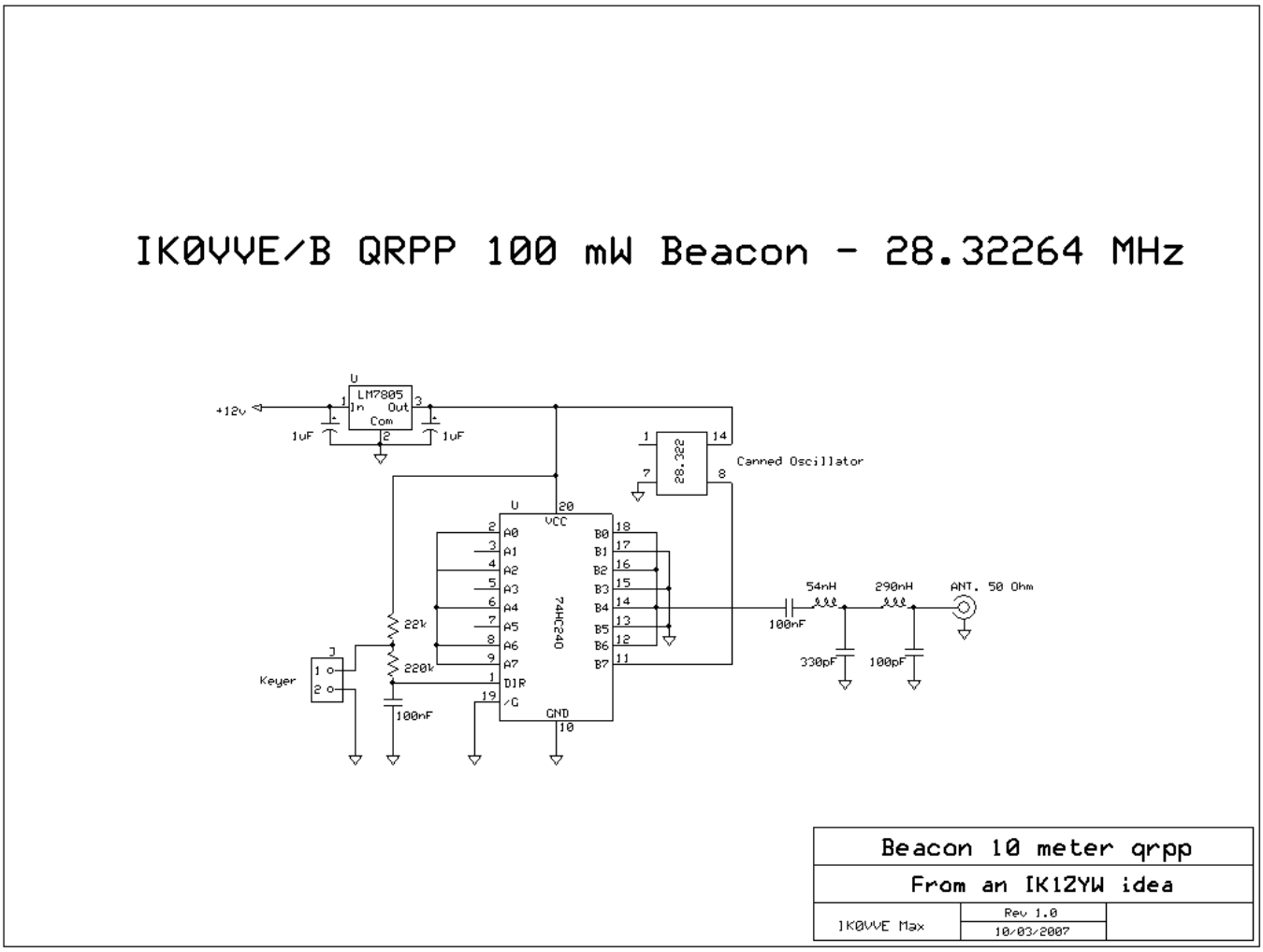
Częstotliwość wyjściowa [MHz]	**Częstotliwości kwarców [MHz] i sposób przemiany**
1,8372	8,8672 – 7,030
14,096	10,000 + 4,096
14,096	18,096 – 4,000
14,097	12,000 – 2,09715
18,106	8,000 + 10,106
21,096	4,096 + 2 x 8,500
21,097	2,09715 + 38,000 / 2
24,926	14,7456 + 10,180
28,125	18,000 + 10,125

Rozpowszechnienie się syntezerów częstotliwości z bezpośrednią syntezą cyfrową (*DDS*) lub opartych na cyfrowej pętli synchronizacji fazy (np. Si570) spowodowało pojawienie się licznych rozwiązań na nich opartych. Najczęściej posiadają one jednostopniowy wzmacniacz mocy na pojedyńczych lub połączonych równolegle tranzystorach małej mocy w.cz. Dzięki syntezie znika rówież problem zaopatrzenia w kwarce lub ich doboru i przemiany częstotliwości w nadajnikach.
Pomimo tak niskich mocy nadawania należy zwrócić uwagę na to, aby poziom harmonicznych i innych sygnałów niepożądanych na wyjściu nadajnika nie przekraczał dopuszczalnych granic. W każdym przypadku konieczne jest umieszczenie na ich wyjściu kilkusekcyjnych filtrów dolnoprzepustowych.
Źródłem sygnałów kluczujących są najczęściej układy mikroprocesorowe a w przypadku prostych sygnałów graficznych przerzutniki lub generatory napięć o innych kształtach (przykładowo napięć trójkątnych lub piłokształtnych).

Nadajniki na bramkach logicznych

Nadajnik o mocy 100 mW na pasmo 10 m

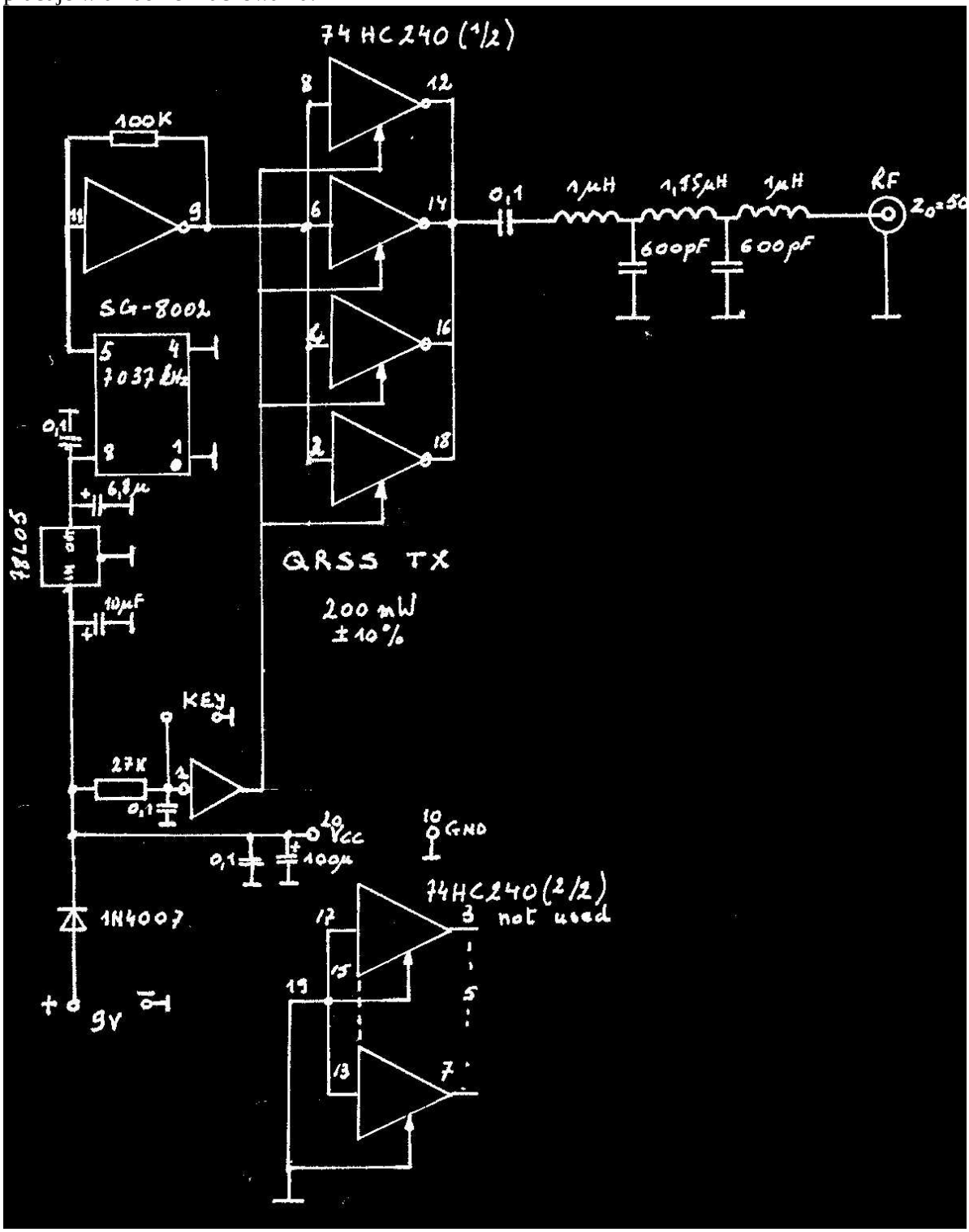
W układzie nadajnika z rys. 2.1 pracuje układ scalony 74HC240 zawierający 8 inwerterów (z czego 4 pracują równolegle jako wzmacniacz mocy, jeden jako separator a pozostałe trzy są niewykorzystane) i kwarcowy generator typu komputerowego o częstotliwości 28322 kHz. Całość jest zasilana stabilizowanym przez LM7805 napięciem 5 V. Dla zwiększenia mocy wyjściowej można obwód 74HC240 zasilać wyższym napięciem 6 – 8 V. Układ kluczujący jest podłączony do dzielnika napięcia 22 kΩ / 220 kΩ. Znajdujący się w obwodzie kondensator 100 nF eliminuje wpływ ewentualnego iskrzenia styków klucza na pozim sygnału na wejściu obwodu scalonego.
Na wyjściu nadajnika znajduje się filtr dolnoprzepustowy złożony z indukcyjności L1 54 nH (3 zw. przew. o śr. 1 mm na rdzeniu T50-2) i L2 290 nH (8 zwojów przewodu o średnicy 1 mm na rdzeniu T50-2) oraz pojemności 330 pF i 100 pF.
Po odpowiednim przeliczeniu wartości elementów filtru dolnoprzepustowego i wymianie oscylatora kwarcowego albo dodaniu dzielnika częstotliwości np. na przerzutnikach 74HC74 można układ ten łatwo przystosować do pracy w innych pasmach amatorskich. Sygnał wyjściowy nadajnika jest falą prostokątną dlatego też należy zwrócić szczególną uwagę na skuteczność tłumienia harmonicznych przez filtr wyjściowy.



Rys. 2.1. Nadajnik na 74HC240

Nadajnik na pasmo 40 m

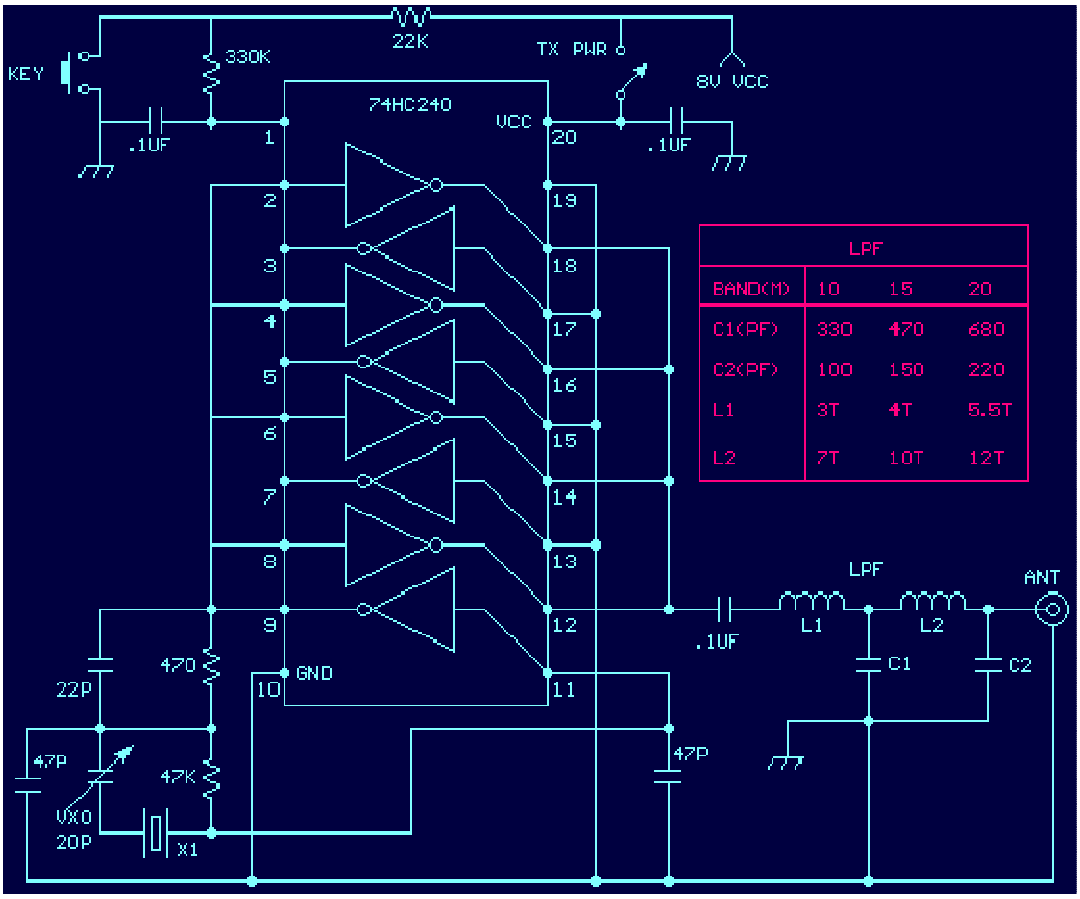
Przedstawiony na schemacie 2.2 nadajnik na pasmo 40 m ma układ zbliżony do poprzedniego. Również i tutaj zastosowano komputerowy oscylator kwarcowy w obudowie metalowej (na częstotliwość 7037 kHz), którego sygnał poprzez bramkę pracującą jako liniowy wzmacniacz (dzięki ujemnemu sprzężeniu zwrotnemu przez opornik 100 kΩ) steruje wzmacniaczem mocy złożonym z czterech bramek połączonych równolegle. Niższa częstotliwość pracy i wyższe napięcie zasilania (ok. 8,3 V) pozwalają na uzyskanie (wg. autora) mocy wyjściowej ok. 200 mW. Szósta z zawartych w obwodzie scalonym bramek pracuje w układzie kluczowania.



Rys. 2.2. Schemat nadajnika na pasmo 40 m

Kwarcowy oscylator TTL jest zasilany napięciem 5 V pochodzącym ze stabilizatora 78L05.
Wejścia dwóch pozostałych nieużywanych inwerterów są zwarte do masy.
Na wyjściu nadajnika znajduje się filtr dolnoprzepustowy złożony z trzech indukcyjności: 1 µH, 1,95 µH i 1 µH oraz pojemności 2 x 600 pF.

Nadajnik na pasma 10 – 20 m



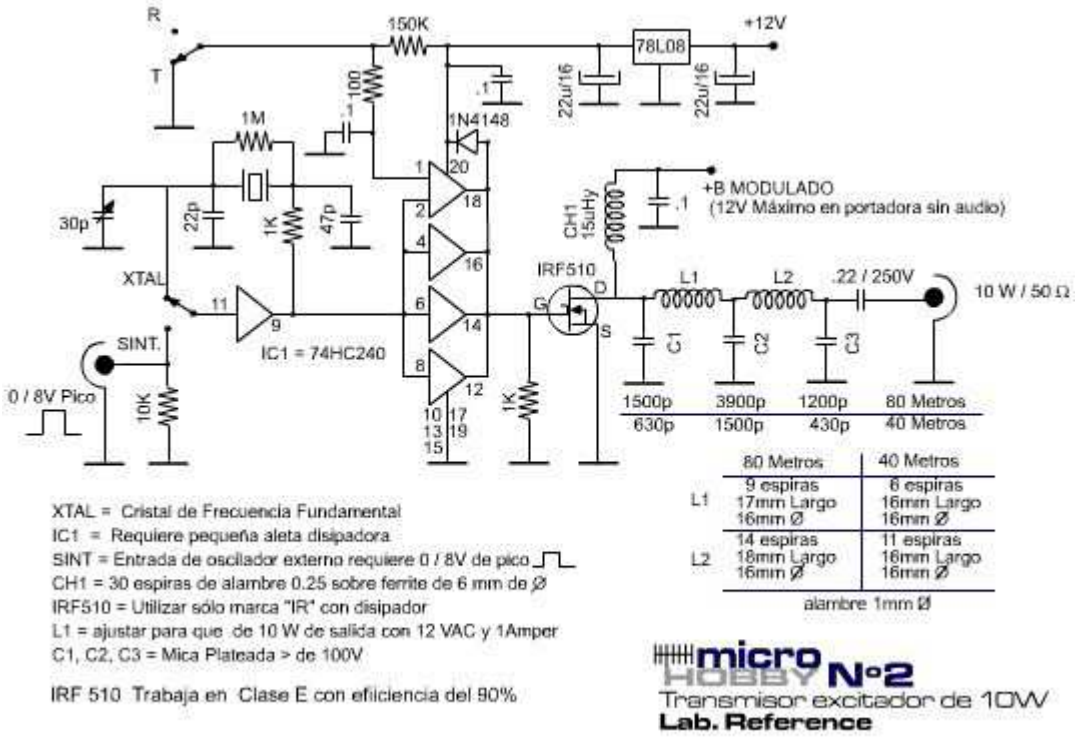
Rys. 2.3. Wielopasmowy nadajnik na 74HC240

VXO w układzie Pierce'a pracuje na inwerterze połączonym z nóżkami 9 i 11. Jego sygnał wyjściowy jest doprowadzony do czterech połączonych równolegle bramek (inwerterów) służących za stopień mocy. Dla uzyskania większej mocy wyjściowej nadajnik jest zasilany napięciem 8 V. W ramce podane są wartości elementów filtru dolnoprzepustowego dla pasm 10, 15 i 20 m. Cewki powietrzne nawinięte są na karkasach o średnicy 9–10 mm lub bez nich. Układ można łatwo przystosować do pracy w paśmie 30 m przez wymanę kwarcu i elementów filtru dolnoprzepustowego.

Nadajnik o mocy 10 W na pasma 80 i 40 m

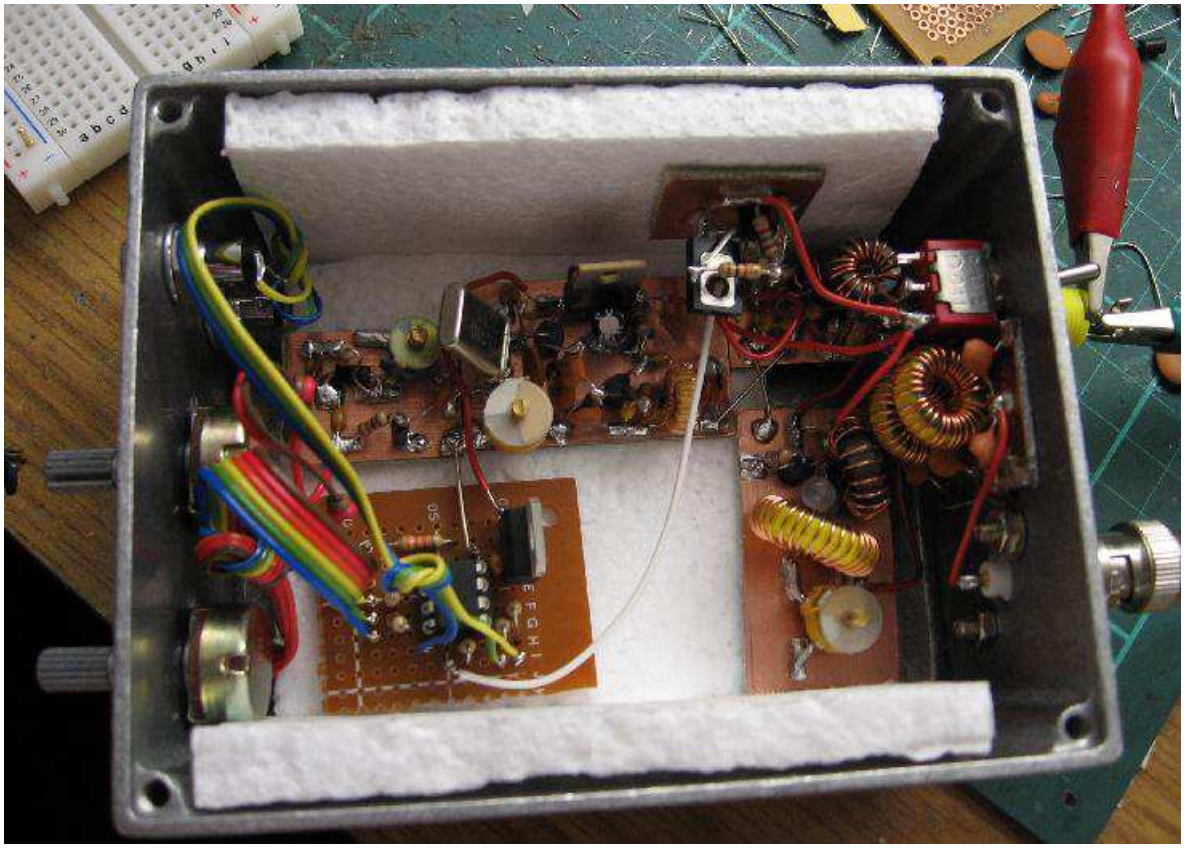
Układ nadajnika na 74HC240 można rozbudować dodając stopień mocy na tranzystorze IRF 510 lub podobnym. W konstrukcji z rys. 2.4 uzyskano moc wyjściową 10 W. Jest to wprawdzie znacznie więcej niż potrzeba dla radiolatarni QRP ale czasami może to być potrzebne w szczególnych sytuacjach. Stosując inny typ tranzystora lub obniżając napięcie zasilania można ograniczyć moc wyjściową do pożądanej wartości.
Do punktu B podłączony jest układ kluczujący lub modularor amplitudy zależnie od rodzaju emisji. Maksymalne napięcie zasilające IRF510 wynosi 12 V.
Obwód IC1 wymaga niewielkiego radiatora. Kwarc pracuje na częstotliwości podstawowej.
Cewka L1 składa się z 9 zwojów nawiniętych na średnicy 16 mm i długości 17 mm dla pasma 80 m lub 6 zwojów nawiniętych na tej samej średnicy i długości 16 mm dla pasma 40 m.

Cewka L2 ma ta samą średnicę i dla pasma 80 m składa się z 14 zwojów na długości 18 mm a dla pasma 40 m – z 11 zwojów na długości 16 mm. Cewke L1 nalezy dostroić tak aby przy napieciu zasilania 12 V uzyskać na wyjściu moc 10 W.
Tranzystor IRF510 pracuje w klasie E ze sprawnością 90 % ale wymaga mimo to radiatora. Dławik CH1 ma indukcyjność 15 µH i jest nawinięty przewodem o średnicy 0,25 mm na rdzeniu ferrytowym o średnicy 6 mm.
Przełącznik na wejściu bramki generatora pozwala na doprowadzenie sygnału logicznego z zewnętrznego źródła.



Rys. 2.4. Schemat radiostacji z dodatkowym wzmacniaczem mocy

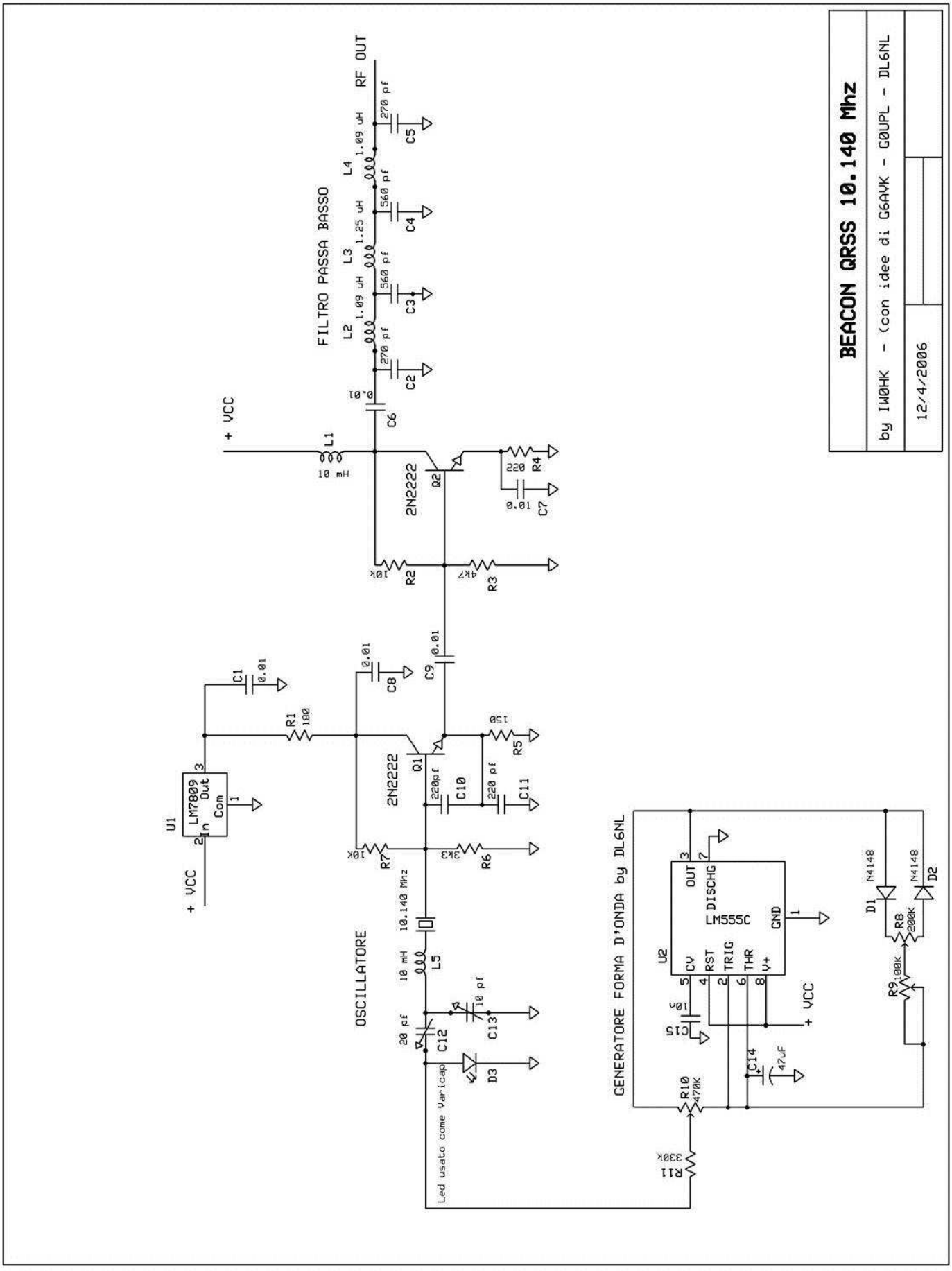
Proste nadajniki kwarcowe



Fot. 2.5. Przykładowa konstrukcja mechaniczna radiolatarni

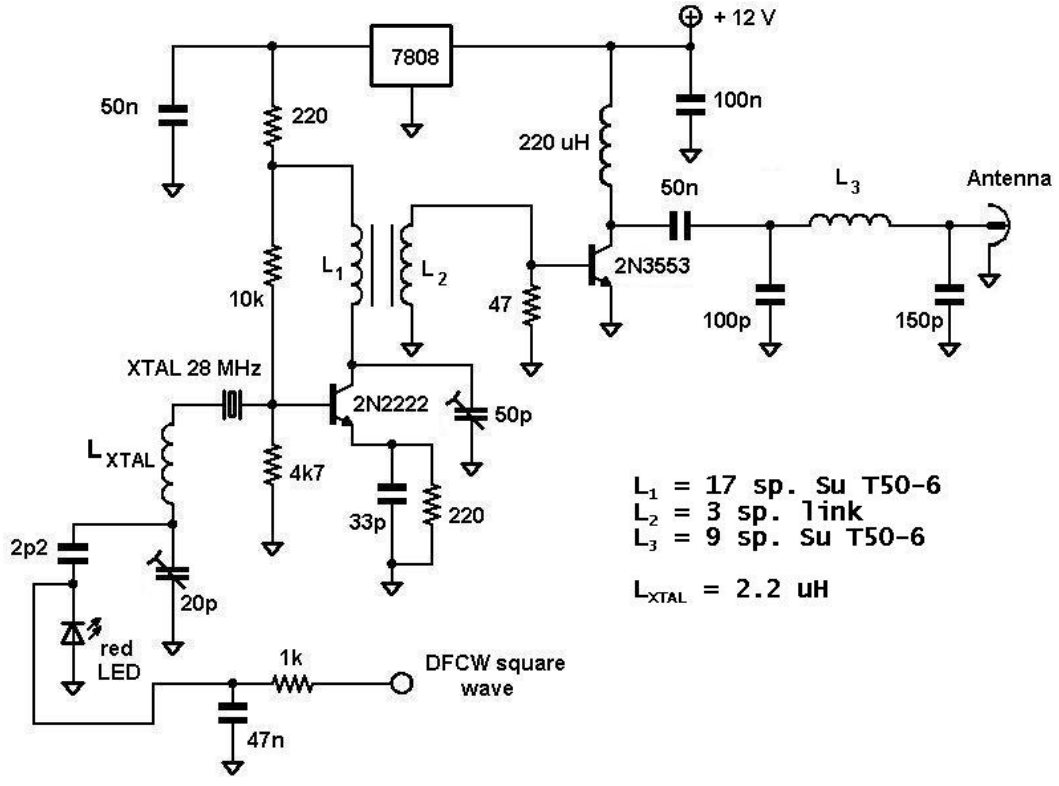
Zdjęcie 2.5 przedstawia przykład konstrukcji tranzystorowego nadajnika radiolatarni QRP. Dla poprawy stabilności częstotliwości jest on umieszczony w obudowie metalowej izolowanej termicznie przez warstwę styropianu. Temperatura kwarcu może być dodatkowo stabilizowana np. przy użyciu przedstawionych dalej układów grzejników. Łatwiejsze do wykonania i zapewniające jeszcze lepszą izolację termiczną jest umieszczenie samego generatora sterującego w małym pudełeczku wykonanym ze styropianu.
Układy nadawcze i kluczujące są na tyle nieskomplikowane, że mogą być wykonane w dowolny sposób: na dowolnego rodzaju drukowanych płytkach uniwersalnych dziurkowanych lub nie, metodą wysepek frezowanych na laminacie, w powietrzu metodą pająków albo na specjalnie przygotowanych obwodach drukowanych. Wykonanie poszczególnych modułów na oddzielnych płytkach ułatwia ich wymianę i modyfikację stacji ale prostota układów pozwala także na skonstruowanie całości na jednej wspólnej płytce.

Dwustopniowy nadajnik FSK na pasmo 30 m



Dwustopniowy nadajnik (rys. 2.6) składa się z przestrajanego generatora kwarcowego VXO na tranzystorze Q1 pracującego w pobliżu częstotliwości 10140 kHz i stopnia mocy na tranzystorze Q2. Do jego strojenia w wąskim zakresie służy kondensator C13 (10 pF) a do kluczowania lub modulacji częstotliwości dioda D3. Zamiast zwykle stosowanej do tego celu diody pojemnościowej autor konstrukcji użył diody świecącej (elektroluminescencyjnej) spolaryzowanej w kierunku przewodzenia. Dewiacja częstotliwości jest regulowana za pomocą trymera C12 (20 pF).
W rozwiązaniu przedstawionym na schemacie jako źródło sygnału kluczującego służy LM555C w układzie przerzutnika astabilnego. Częstotliwość przebiegu jest regulowana za pomocą potencjometru R9, a amplituda za pomocą R10. Potencjometr R9 służy do zmiany kształtu (stopnia wypełnienia) przebiegu. Zamiast tego można oczywiście podłączyć generator dowolnego sygnału innego typu np. telegrafii QRSS albo dalekopisów wg normy *Slowfeld*. VXO jest zasilany napięciem 9 V ze stabilizatora U1 (LM7809).
W stopniu mocy (Q2) pracuje tranzystor tego samego typu – 2N2222 – który można łatwo zastąpić przez odpowiednik europejski lub nawet tylko tranzystor o zbliżonych parametrach. Tranzystor ten jest zasilany napięciem 12 V przez dławik o indukcyjności 10 mH (?) a na jego wyjściu znajduje się filtr dolnoprzepustowy. Kluczowanie amplitudy nadajnika polega na włączaniu i wyłączaniu napięcia zasilającego wzmacniacz mocy w takt sygnału. W zależności od potrzeb układ kluczowania częstotliwości można odłączyć lub podać na jego wejście stabilizowane napięcie stałe.
Indukcyjności cewek L2 – L4 filtru wyjściowego wynoszą odpowiednio 1,09, 1,25 i 1,09 µH a pojemności C2 – C5 – 270 pF, 560 pF, 560 pF i 270 pF.
Podane na schemacie wartości indukcyjności szeregowej w obwodzie kwarcu i dławika zasilającego stopień mocy wydają się autorowi nierealistycznie duże – zamiast 10 mH należałoby raczej zastosować w obu miejscach 10 lub kilkanaście µH. Być może jest to poprostu błąd w podpisach na rysunku. Dla obliczenia indukcyjności dlawika zasilającego stopień mocy stowowany jest często wzór L[µH] = 1000 / f [MHz] ale nie trzeba się tego trzymać niewolniczo. Wartości podane na wielu kolejnych schematach odbiegają od obliczonych w ten sposób.

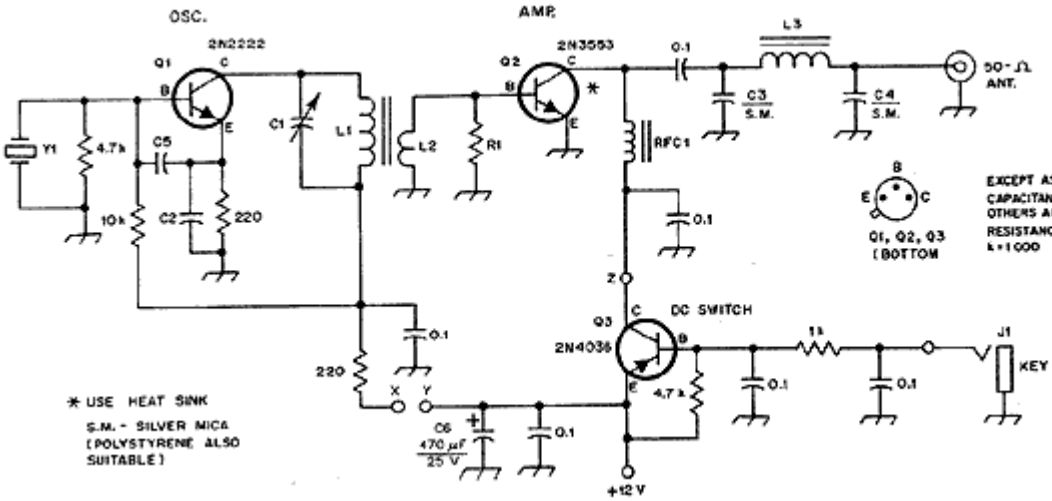
Dwustopniowy nadajnik FSK na pasmo 10 m



Rys. 2.7. Kwarcowy nadajnik tranzystorowy

Jest to układ zasadniczo podobny do poprzedniego z tą różnicą, że w stopniu mocy zastosowano tranzystor 2N3553 i dzięki temu moc wyjściowa może wynosić ok. 1 W. Tranzystor pracuje w klasie C i jest zasilany napięciem 12 V przez dławik 220 µH.
Obwód kluczowania częstotliwości jest zbliżony do poprzedniego ale dioda jest spolaryzowana zaporowo i wyraźnie słabiej sprzężona z obwodem kwarcu, co powinno w zupełności wystarczyć dla uzyskania dewiacji kilku Hz – dla FSCW lub DFCW. Napięcie kluczujące jest filtrowane przez filtr dolnoprzepustowy RC – 1 kΩ, 47 nF. Jego stałą czasu można dobrać w zależności od szybkości zmian sygnału kluczującego.
Kluczowanie amplitudy nadajnika polega na włączaniu i wyłączaniu napięcia zasilającego wzmacniacz mocy w takt sygnału. W zależności od potrzeb układ kluczowania częstotliwości można odłączyć lub podać na jego wejście stabilizowane napięcie stałe.
Cewki L1, L2 i L3 są nawinięte na rdzeniach pierścieniowych T50-6 (żółtych) i mają odpowiednio 17, 3 i 9 zwojów.
Generator VXO jest zasilany napięciem stabilizowanym 8 V. Jego zakres przestrajania można rozszerzyć łącząc ze sobą równolegle dwa kwarce o tej samej częstotliwości rezonasowej.

Dwustopniowy nadajnik telegraficzny na pasma 160 – 10 m



	C1 (pF)	C2 (pF)	C3 (pF)	C4 (pF)	C5 (pF)	L1	L2	L3	R1	RFC1
160 m	400	1800	1800	1800	360	73 t No. 28 T50-2	8 t	30 t No. 26 T50-2	18 Ω	30 t No. 28 FT-37-61 (50 µH)
80 m	400	100	750	750	200	43 t No. 26 T50-2	5 t	21 t No. 22 T50-2	39 Ω	21 t No. 28 FT-37-61 (25 µH)
40 m	180	100	470	470	—	35 t No. 26 T50-2	4 t	14 t No. 22 T50-2	39 Ω	30 t No. 28 FT-37-63 (15 µH)
20 m	60	33	210	210	—	27 t No. 24 T50-6	3 t	12 t No. 22 T50-6	47 Ω	30 t No. 28 FT-37-63 (15 µH)
15/10 m	60	33	105	130	—	17 t No. 24 T50-6	3 t	9 t No. 22 T50-6	47 Ω	30 t No. 28 FT-37-63 (15 µH)

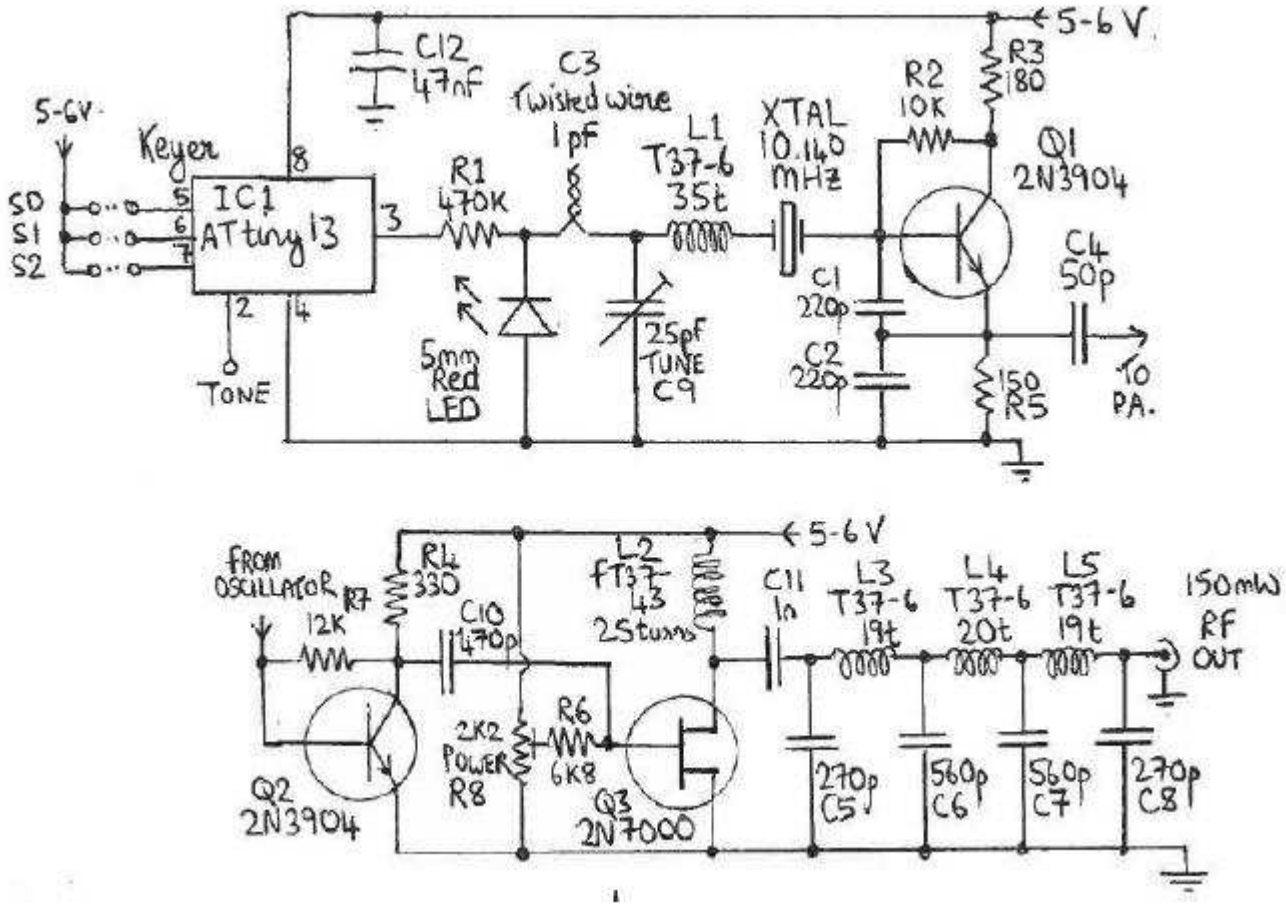
Toroid cores are used in L1, L2 and L3. These are powdered-iron cores available from Amidon Associates and Palomar Engineers (T50-2, etc.). RFC1 is wound on a small ferrite core (FT-37-67), and so on), available from same suppliers. The letter "t" signifies the number of wire turns in the winding.

Rys. 2.8. Wielopasmowy nadajnik tranzystorowy K0KP wraz z wartościami elementów

Jest to rozwiązanie zasadniczo podobne do poprzedniego zawierające dodatkowo układ kluczowania amplitudy na tranzystorze Q3 (PNP). Nadajnik ten może więc służyć do transmisji telegraficznej, QRSS i dalekopisowej w normie *Feldhell*. Po zmianie wartości elementów obwodu rezonansowego i filtru

dolnoprzepustowego, zgodnie z podanymi przykładami można go dostosować do pracy w różnych pasmach amatorskich.
Cewki L1, L2 i L3 nawinięte są na pierścieniowych rdzeniach proszkowych T50-2 (czerwonych) a dławik w kolektorze stopnia końcowego (RFC1) na rdzeniu ferrytowym FT37-61 lub FT37-63. Liczby poprzedzające literę „t" oznaczają liczbę zwoi. Podana na schemacie pojemność 0,1 oznacza 0,1 µF. Tranzystor Q2 wymaga radiatora. Układ został opracowany i opublikowany przez K0KP.

Trzystopniowy nadajnik na pasma 30, 40 i 80 m
ze wzmacniaczem mocy na tranzystorze polowym



Rys. 2.9 Układ nadajnika G0UPL

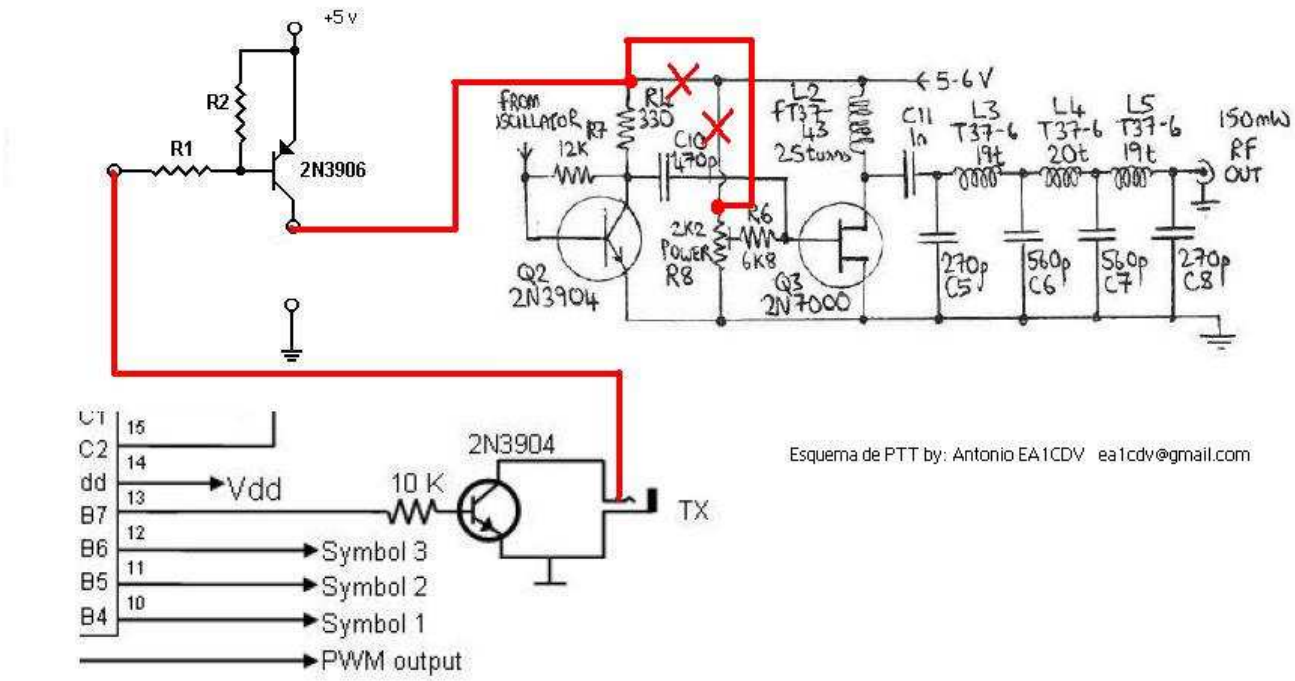
Schemat przedstawia układ trzystopniowego nadajnika na pasmo 30 m różniącego się od poprzednich tym, że w stopniu wyjściowym pracuje polowy tranzystor przełącznikowy 2N7000 (można zastąpić go przez BS170 mający w przybliżeniu podobne parametry, VN10 itp.). Dla uzyskania większej mocy możliwe jest równoległe połączenie dwóch lub trzech tranzystorów tego typu. Moc wyjściowa nadajnika z pojedyńczym tranzystorem wynosi 100 – 150 mW. Przy zasilaniu napięciem 12 V i użyciu dwóch równolegle połączonych tranzystorów można bez trudu uzyskać moce przekraczające 700 mW.
Generator VXO na tranzystorze Q1 pracuje w układzie Colpittsa a wzmacniacz na tranzystorze Q2 służy jako separator. Na doprowadzenie bramki tranzystora mocy można założyć perełkę ferrytową dla stłumienia ewentualnych oscylacji pasożytniczych w zakresach UKF.
Nadajnik jest kluczowany częstotliwościowo – z dewiacją ok. 5 Hz – za pomocą mikroprocesora ATtiny13 (zwory S0 – S2 służą do wyboru szybkości nadawania QRSS1 – QRSS20 lub 6 albo 12 sł./min.). Podobnie jak w innych przestawionych tu konstrukcjach zamiast diody pojemnościowej pracuje zwykła czerwona dioda świecąca o średnicy 5 mm. Kondensator 1 pF sprzęgający ją z obwodem kwarcu uzyskano przez skręcenie ze sobą dwóch przewodów na długości ok. 2,5 cm (długość przewodów przed skręceniem wynosi w przybliżeniu 5 cm). Trymer C9 służy do dokładnego dostrojenia generatora do wybranej częstotliwości pracy. Cewka L1 składa się z 35 zwojów przewodu emaliowane-

go nawiniętych na proszkowym rdzeniu pierścieniowym T37-6 (żółtym). Na takich samych rdzeniach nawinięte są cewki filtru dolnoprzepustowego L3 – L5 mające odpowiednio po 19, 20 i 19 zwojów. Dławik w obwodzie drenu stopnia mocy zawiera 25 zwojów nawiniętych na rdzeniu ferrytowym FT37-43 (czarnym).
Przed uruchomieniem układu należy ustawić suwak potencjometru R8 na potencjale masy i następnie stopniowo zwiększać napięcie polaryzacji bramki aż do osiągnięcia mocy wyjściowej rzędu 100 mW, zwracając uwagę na to, żeby nie przegrzać tranzystora. Mocy wyjściowej 100 mW odpowiada napięcie o wartości międzyszczytowej ok. 6,3 V (można je obserwować na oscyloskopie) na obciążeniu 50 Ω. Za pomocą trymera C9 należy dostroić nadajnik do dowolnej częstotliwości w zakresie 10140,0 – 10140,1 kHz. W razie trudności w dostrojeniu należy zmienić indukcyjność L1 zmniejszając lub powiększając ilość zwojów. Pojemność kondensatora skrętkowego należy dobrać tak, aby dewiacja FSK wynosiła ok. 5 Hz. Najłatwiej dokonać tego obserwując własny sygnał w oknie programu „Argo" lub podobnego.
Autorem konstrukcji jest Hans Summers G0UPL (*www.hanssummers.com*). Zestaw konstrukcyjny lub zaprogramowany mikroprocesor są osiągalne w sklepie internetowym konstruktora. W dalszej części skryptu podano kod źródłowy programu dla mikroprocesora ATtiny13 oraz przykład kodu dla procesora 12F629. Układ ten spotykany jest w wielu wariantach sterowanych przez różne typy procesorów lub mikrokomputer „Arduino".

Tabela 2.2. Wartości elementów dla pasm 80 – 30 m.

	Pasmo 80 m	**Pasmo 40 m**	**Pasmo 30 m**
L1	27 zw. T37-6 (żółty)	27 zw. T37-6 (żółty)	27 zw. T37-6 (żółty)
L2	25 zw. FT37-43 (czarny)	25 zw. FT37-43 (czarny)	25 zw. FT37-43 (czarny)
L3	25 zw. T37-2 (czerwony)	19 zw. T37-6 (żółty)	19 zw. T37-6 (żółty)
L4	27 zw. T37-2 (czerwony)	21 zw. T37-6 (żółty)	20 zw. T37-6 (żółty)
L5	25 zw. T37-2 (czerwony)	19 zw. T37-6 (żółty)	19 zw. T37-6 (żółty)
C1, 2	680 pF	470 pF	220 pF
C3	1 pF skrętka	1 pF skrętka	1 pF skrętka
C4	47 pF	47 pF	47 pF
C5, 8	470 pF	270 pF	270 pF
C6, 7	1200 pF (1,2 nF)	680 pF	560 pF
C9	25 pF, trymer	25 pF, trymer	25 pF, trymer
C10	470 pF	470 pF	470 pF
C11	1 nF	1 nF	1 nF
C12	47 nF	47 nF	47 nF
Kwarc	3500 kHz	7000 kHz	10140 kHz

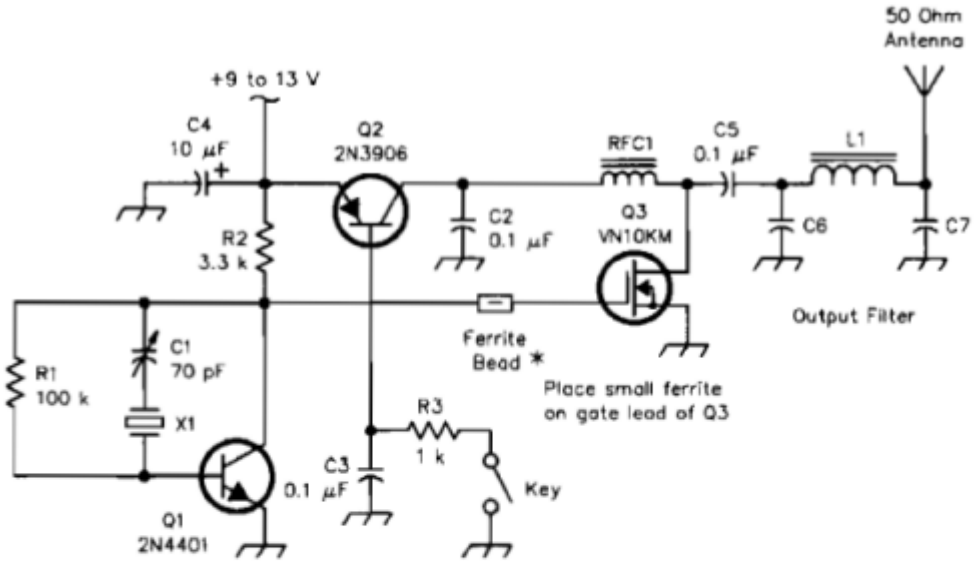
Na schemacie 2.10 przedstawiono sposób kluczowania amplitudy nadajnika przez mikroprocesor (za pośrednictwem tranzystora wykonawczego dowolnego typu) lub podłączony klucz telegraficzny. Modyfikację opracował Antonio EA1CDV. Tranzystor PNP 2N3906 można zastapić przez tranzystor europejski dowolnego typu np. BC177 a NPN 2N3904 – przez BC107 albo BC547. Opornik R2 ma wartość 100 kΩ a R1 ok. 10 kΩ.



Rys. 2.10. Modyfikacja dla kluczowania amplitudy

Telegraficzny nadajnik o mocy 1 W na pasma 160 – 17 m

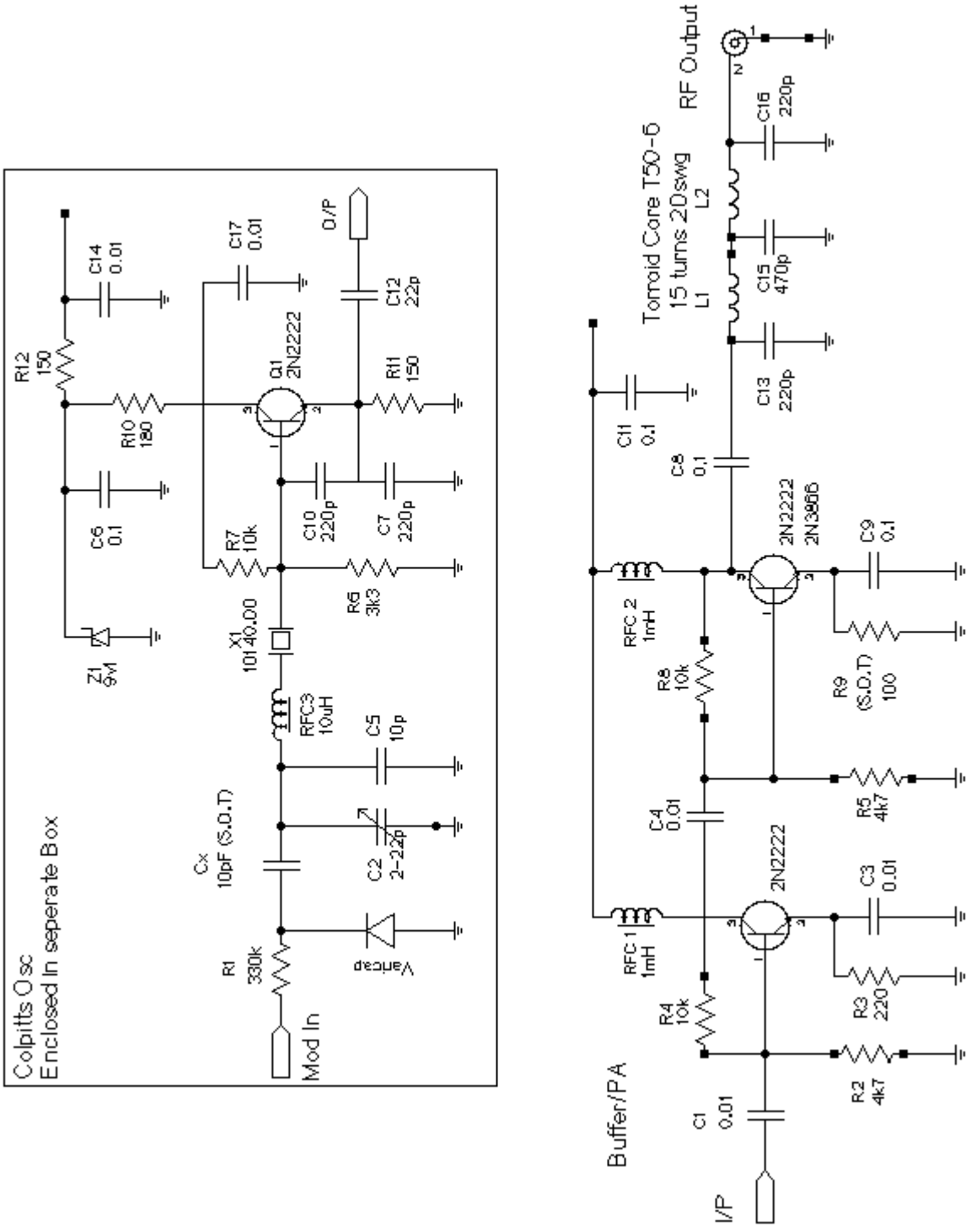
Dwustopniowy nadajnik ze stopniem mocy na tranzystorze polowym VN10KM może w zależności od napięcia zasilania i pasma dostarczyć mocy w.cz. dochodzącej do 1 W. Na doprowadzeniu jego bramki znajduje się perełka ferrytowa zapobiegająca pasożytniczym oscylacjom w zakresie UKF. Perełka taka może przydać się również i w innych rozwiązaniach wzmacniaczy mocy z tranzystorami polowymi.
Generator VXO w układzie Pierce'a jest przestrajany w wąskim zakresie czętotliwości za pomocą kondensatora C1.
Tranzystor Q2 (PNP) służy do kluczowania nadajnika poprzez włączanie napięcia zasilania stopnia końcowego po zwarciu jego bazy do masy. W zależności od pasma dławik w obwodzie zasilania Q3 powinien mieć indukcyjność od 15 µH (pasma 40 – 17 m) do 25 – 50 µH (odpowiednio dla pasm 80 i 160 m).
Filtr dolnoprzepustowy R3C3 decyduje o czasach narastania i opadania sygnału telegraficznego.
W miejsce klucza telegraficznego można oczywiście podłączyć dowolny –na przykład mikroprocesorowy – układ kluczujący. Przykłady takich rozwiązań podano w rozdziale 4.
Wartości elementów filtru dolnoprzepustowego (C6, C7, L1) dla poszczególnych pasm podano na schemacie. Kondensatory C6 i C7 są kondensatoramu ceramicznymi. Cewka L1 jest nawinięta na rdzeniu pierścieniowym T37-2 (czerwonym).



C6,C7	L1
820 pF disc ceramic (160 meters)	33 turns, #30, T37-2 (160 meters)
470 pF disc ceramic (80 meters)	23 turns, #30, T37-2 (80 meters)
220 pF disc ceramic (40 meters)	17 turns, #26, T37-2 (40 meters)
150 pF disc ceramic (30 meters)	14 turns, #26, T37-2 (30 meters)
100 pF disc ceramic (20 meters)	12 turns, #26, T37-2 (20 meters)
82 pF disc ceramic (17 meters)	10 turns, #26, T37-2 (17 meters)

Rys. 2.11. Wielopasmowy nadajnik telegraficzny

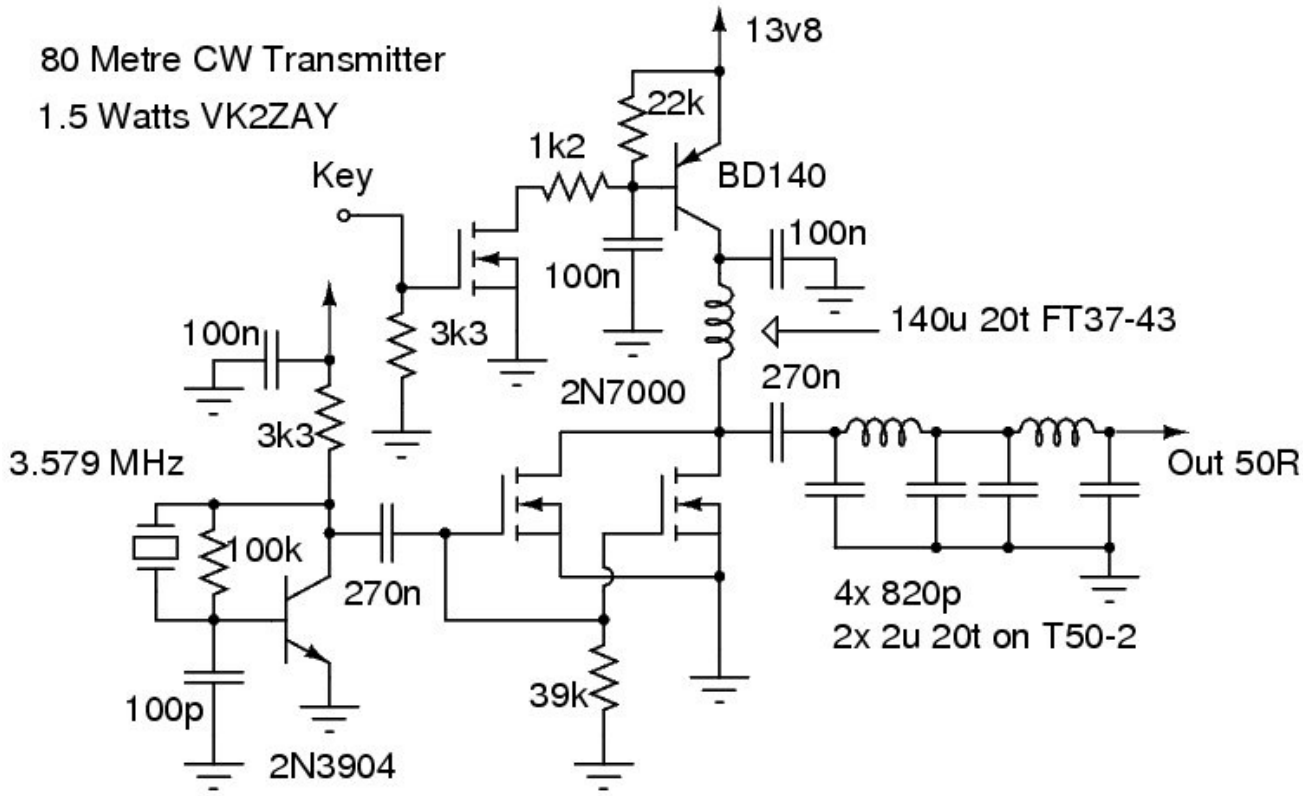
Trzystopniowy nadajnik na pasmo 30 m na tranzystorach złączowych



Rys. 2.12. Nadajnik na pasmo 10 MHz

Rozwiązanie to nie wymaga zasadniczo szczegółowego opisu. Generator sterujący w układzie Colpittsa jest zasilany napięciem stabilizowanym i umieszczony w oddzielnej obudowie dla uzyskania lepszej izolacji cieplnej. Jest on kluczowany częstotliwościowo za pomocą diody pojemnościowej, ale dodanie układu kluczowania amplitudy nadajnika nie stanowi większego problemu.
Wartość pojemności Cx należy dobrać tak aby uzyskać dewiację kilku Hz.

Nadajnik telegraficzny o mocy 1,5 W na pasmo 80 m

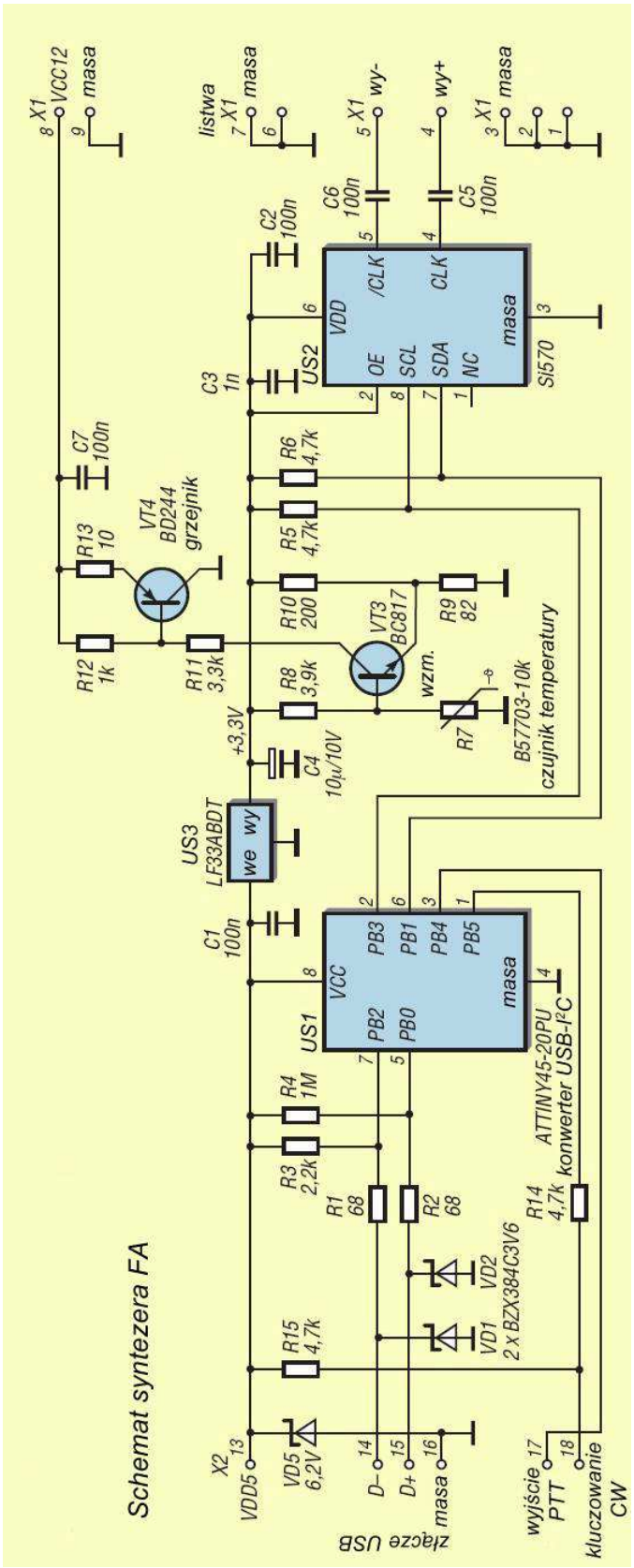


Rys. 2.13. Prosty nadajnik telegraficzny

Układ nadajnika zawiera stopnie zasadniczo zbliżone do występujących w poprzednich rozwiązaniach i nie wymaga dokładniejszego omówienia. Generator wzbudzający w układzie Pierce'a steruje stopień mocy na dwóch równoległych tranzystorach polowych 2N7000. Przy napięciu zasilania 13,8 V uzyskuje się moc wyjściową rzędu 1,5 W. W ramach modyfikacji można dodać potencjometr do regulacji napięcia polaryzacji bramek tranzystorów i ewentualnie także trzeci tranzystor 2N7000.
Jedynie układ kluczujący amplitudę jest bardziej rozbudowany i zawiera oprócz tranzystora kluczującego PNP typu BD140 także stopień odwracający polaryzację na tranzystorze 2N7000 (nadajnik jest kluczowany napięciem dodatnim a nie przez zwarcie wejścia do masy). Po wymianie kwarcu i elementów filtru dolnoprzepustowego można go łatwo przystosować do pracy w innych pasmach amatorskich. Indukcyjność dławika dla środkowych częstotliwości KF (40 – 20 m) można zmniejszyć o połowę lub nawet więcej a dla pasm wyższych nawet do ¼ podanej wartości (140 µH). W doborze indukcyjności dławika można kierować się innymi podanymi w skrypcie rozwiązaniami.
VK2ZAY wypróbował różne rozwiązania stopnia mocy z tranzystorami IRF510 i VN10 KM ale najlepsze wyniki dawały połączone równolegle tranzystory 2N7000. Dławik w obwodzie ich zasilania składa się z 20 zwojów nawiniętych na rdzeniu ferrytowym FT37-43 a cewki filtru dolnoprzepustowego zwierają po 20 zwojów nawiniętych na rdzeniu proszkowym T50-2 (czerwonym).

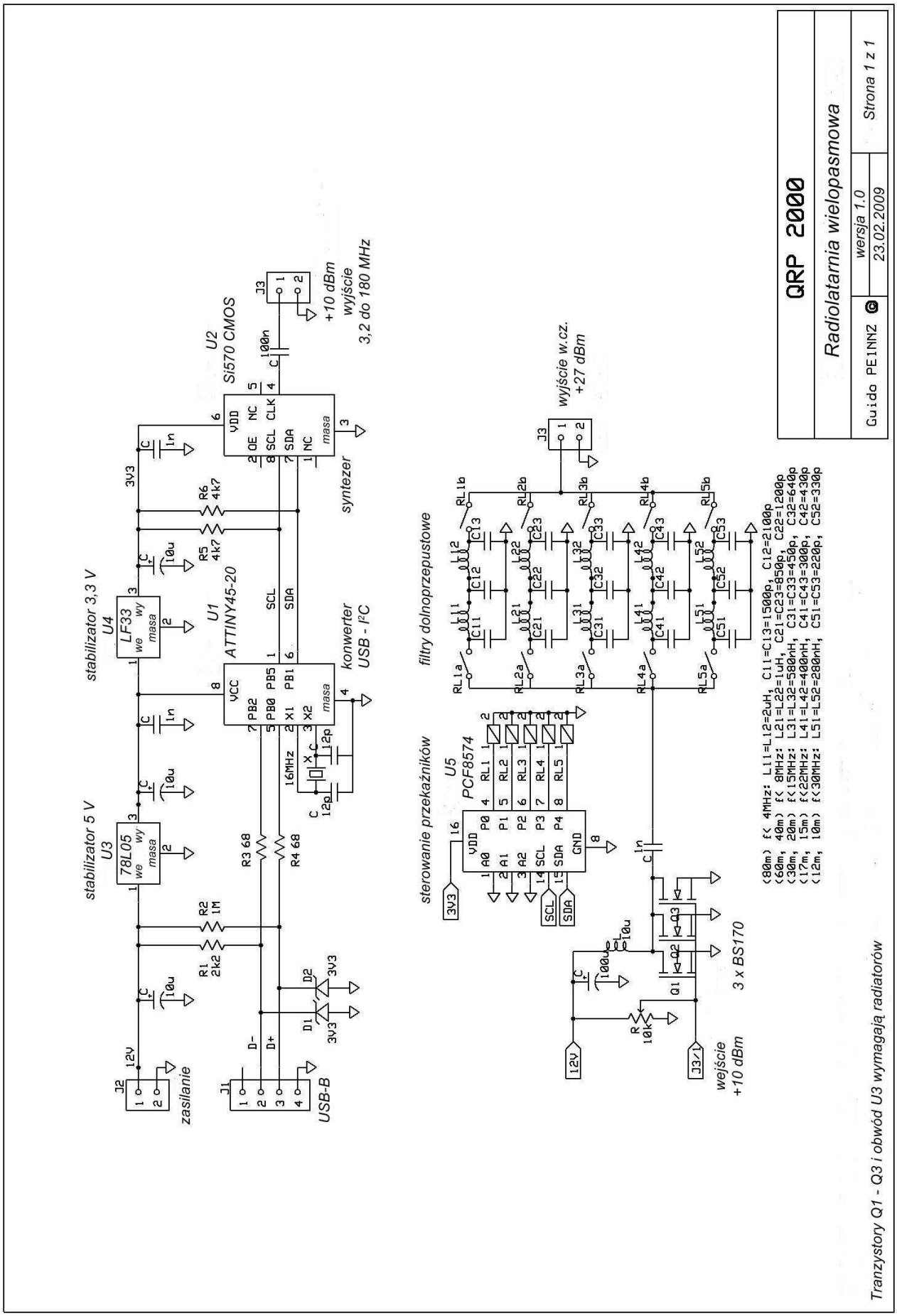
Nadajniki z syntezą częstotliwości

Nadajniki z syntezerem Si570



Do najważniejszych zalet hybrydowego syntezera Si570 firmy SiliconLaboratories należy widmowa czystość sygnału. Jest ona znacznie lepsza aniżeli w wielu syntezerach cyfrowych DDS.
Moduł syntezera zawiera sterowany cyfrowo generator odniesienia, synchronizowany generator mikrofalowy pracujący na częstotliwości około 5 GHz oraz układ cyfrowej pętli synchronizacji fazy DSPLL. Po podzieleniu częstotliwości pracy generatora mikrofalowego otrzymuje się pożądaną częstotliwość wyjściową syntezera. Metoda ta zapewnia jednocześnie obniżenie poziomu szumów fazowych na wyjściu w stosunku równym stosunkowi częstotliwości.
Zakres częstotliwości wyjściowych (w zależności od typu układu) dochodzi do 945 MHz przy czym dolna częstotliwość graniczna dla wszystkich typów wynosi 10 MHz. Moduły wyposażone w układ wyjściowy CMOS pracują w zakresie do 160 MHz natomiast zawierające wyjścia LVDS albo LVPECL – do 215, 810 lub 945 MHz. Górna czestotliwość graniczna ostatniego z nich wynosi nawet 1400 MHz ale producent nie gwarantuje możliwości ustawienia dowolnych częstotliwości pracy w podzakresie 945 – 1400 MHz, a jedynie w jego pewnych fragmentach.
Dolna częstotliwość pracy podana w danych technicznych wynosi wprawdzie 10 MHz ale próby przeprowadzone przez krótkofalowców wykazały, że możliwe jest uzyskanie częstotliwości niższych dochodzących nawet do 3,5 MHz – jednak i w tym podzakresie nie da się zagwarantować uzyskania wszystkich częstotliwości wyjściowych. Obwód w wykonaniu CMOS pozwala natomiast na połączenie z dzielnikiem częstotliwości i uzyskanie w ten sposób sygnałów o dowolnie niskich częstotliwościach.
Stabilność generowanej częstotliwości wynosi dla układu CMOS +/-50 x 10^{-6} a dla LVDS - +/-20 x 10^{-6}. Można ją jednak znacznie poprawić zapewniając stabilizację temperatury pracy modułu co daje się uzyskać już za pomocą stosunkowo prostych układów (patrz schemat syntezera opracowanego przez miesięcznik „Funkamateur” – FA – rys. 2.14).

Dokładność częstotliwości syntezera FA wynosi +/-1,5 x 10^{-6} bez przeprowadzenia kalibracji ale kalibracja polegająca na porównaniu częstotliwości sygnału syntezera z sygnałem stacji częstotliwości wzorcowej jest zabiegiem niekomplikowanym i zasadniczo wystarczy przeprowadzenie jej po uruchomieniu układu lub też dodatkowo w dłuższych odstępach czasu dla wyeliminowania skutków starzenia się układu – jeżeli zmiany te są istotne w danym przypadku.
Moduły zasilane są napięciem 3,3 V (oferowane są także wykonania dla 1,8 lub 2,5 V) i charakteryzują się poborem prądu od 90 (CMOS) do 130 mA.
Moduł w wersji CMOS dostarcza na wyjściu fali prostokątnej o amplitudzie 2,6 V przy pojemności obciążenia 15 pF – w zakresie do ponad 100 MHz uzyskuje się więc poziom sygnału przekraczający 10 dBm. Dla modułu LVDS wartość międzyszczytowa napięcia przy obciążeniu symetrycznym 100 Ω (dopasowanie do obciążenia 50 Ω jest uzyskiwane za pomocą transformatora szerokopasmowego) wynosi 0,7 V – poziom sygnału wyjściowego w zakresie do 144 MHz wynosi 1 – 2 dBm, a w zakresie do 200 MHz – 0 dBm. Dla obciążenia niesymetrycznego 50 Ω napięcie wynosi połowę podanej wartości.
Syntezer jest sterowany za pomocą magistrali I2C dlatego też w rozwiązaniach praktycznych do jego sterowania lub pośrednictwa w komunikacji z komputerem PC konieczne jest użycie mikrokontrolera. Si570 może być wprawdzie przestrajany z rozdzielczością ułamka Herca ale wielu rozwiązaniach amatorskich przyjęto krok 1 Hz co i tak jest przeważnie wartością wystarczającą.
Grupa krótkofalowców QRP2000 w składzie DG8SAQ, G8BTR, M0PUB, PE1NNZ, G8XAR i G0BBL opracowała jakiś czas temu zestaw konstrukcyjny syntezera opartego na module Si570 przewidziany dla programowalnych odbiorników i radiostacji Softrock, SDR1000, Flex5000 itp, ale oczywiście znajduje on zastosowanie także i w innych dowolnych konstrukcjach amatorskich nadajników, odbiorników i radiolatarni QRSS, WSPR albo innych emisji.
W oparciu o rozwiązanie QRP2000 powstała udoskonalona wersja syntezera (rys. 2.14) dostępna w sklepie internetowym miesięcznika „Funkamateur” (*www.funkamateur.de*). Analogicznie jak w rozwiązaniu poprzednim pracuje tutaj Si570 (IC2) sterowany przez mikrokontroler ATTiny45 (IC1). Syntezer może być zasilany ze złącza USB komputera, z dodatkowego zasilacza albo z układu, do którego został wbudowany.
Do jego sterowania i konfiguracji służą te same programy dla PC co i dla syntezera QRP2000, z tym że stosując go do innych celów aniżeli w odbiornikach programowalnych najlepiej skorzystać z programu USB_synth.exe.
Wejście CW służy do kluczowania sygnału wyjściowego – w zależności od ustawień w programie sterującym – ale wymaga to połączenia syntezera z komputerem. W programie tym wybierana jest również dewiacja dla kluczowania częstotliwości jak i oczywiście częstotliwość pracy. Wyjście PTT służy do ewentualnego kluczowania dalszych stopni nadajnika. Zarówno przewód zasilający napięciem 5 V jak i przewody sygnałowe USB są zabezpieczone przed przepięciami za pomocą diod Zenera. W zależności od ustawień w progamie sterującym mikroprocesor zapamiętuje ostatnio wybraną częstotliwość dzięki czemu syntezer nie musi być połączony stale z komputerem a jedynie w przypadku zmiany ustawień lub kluczowania.
Wejście kluczujące Si570 (OE, nóżka 2) nie jest używane i jest połączone na stałe z napięciem zasilania. W układzie QRP2000 jest ono stosowane do kluczowania sygnału wyjściowego syntezera.
Zasadniczą różnicę w stosunku do rozwiązania QRP2000 stanowi układ regulacji temperatury modułu syntezera utrzymujący jego temperaturę w pobliżu 40 °C. Składa się on z tranzystora BD244 (VT4) służącego jako element grzejny, wzmacniacza na tranzystorze BC817 (VT3), oporników R8 – R13 i termistorowego czujnika temperatury (R7). Zarówno tranzystor VT4 jak i termistor muszą mieć zapewniony dobry kontakt cieplny z obudową Si570. Jako tranzystor VT4 wybrano tranzystor pnp aby jego kolektor mógł znajdować się na potencjale masy i dzięki temu przylegać bezpośrednio do obudowy Si570. W zależności od temperatury obudowy i otoczenia układ grzejnika pobiera dodatkowo do 50 mA ze źródła 12 V a w początkowym okresie po włączeniu nawet do 125 mA.
Pomimo prostoty układ zapewnia wyraźną poprawę stabilności częstotliwości wyjściowej syntezera (osiąga ona swoją wartość nominalną już po kilku minutach). Może on być oczywiście stosowany do stabilizacji temperatury kwarców w dowolnych innych rozwiązaniach generatorów i nadajników.


QRP 2000
Radiolatarnia wielopasmowa
Guido PE1NNZ
wersja 1.0
23.02.2009
Strona 1 z 1
stabilizator 5 V
stabilizator 3,3 V
zasilanie
USB-B
konwerter USB - I²C
syntezer
+10 dBm wyjście 3,2 do 180 MHz
filtry dolnoprzepustowe
sterowanie przekaźników
wyjście w.cz. +27 dBm
wejście +10 dBm
3 x BS170
Tranzystory Q1 - Q3 i obwód U3 wymagają radiatorów

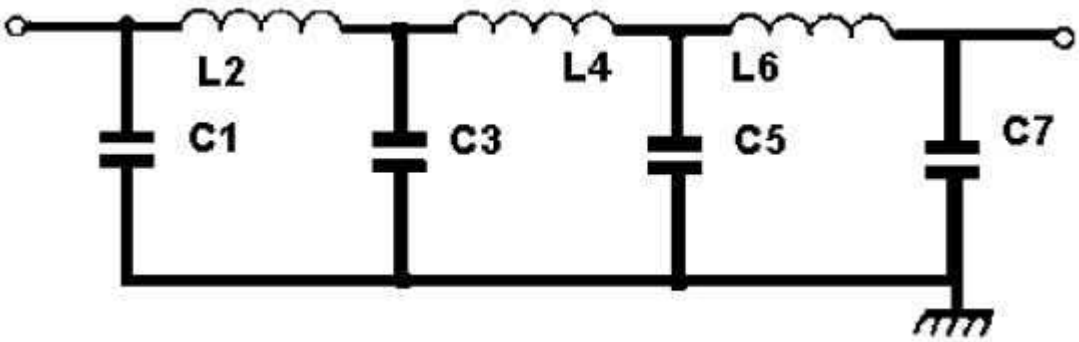
Na schemacie 2.15 przedstawiono przykład zastosowania syntezera w nadajniku radiolatarni QRSS lub WSPR małej mocy (ok. 0,5 W). Wielopasmowa radiolatarnia z automatycznym przełączaniem wyjściowych filtrów za pomocą mikrokontrolera została opracowana przez PE1NNZ i opublikowana w internecie. Po dołączeniu do komputera PC może ona służyć do transmisji WSPR ale nadaje sie też do transmisji telegrafii QRSS, dalekopisowej Hella itp. za pomocą układu mikroprocesorowego bez użycia komputera (połączenie z PC jest konieczne wówczas jedynie w przypadku zmiany częstotliwości pracy). W razie rezygnacji z automatycznego przełączania pasm lub przejścia na przełączane ręczne zbędne stają się obwód scalony U5 (PCF8574) i przekaźniki w obwodzie filtrów dolnoprzepustowych a sama liczba filtrów zależy od rzeczywiście używanych pasm. We wzmacniaczu mocy pracują trzy połączone równolegle tranzystory polowe małej mocy typu BS170, 2N7000 lub podobne. Już dla dwóch równoległych tranzystorów 2N7000 OE1KDA uzyskał w paśmie 30 m moc wyjściową ok. 700 mW. W układzie radiolatarni opracowanym przez G0UPL w stopniu mocy pracował tranzystor IRF740 dostarczając również mocy około 0,5 W. Typ tranzystora nie jest krytyczny a więc można go zastąpić przez jakikolwiek inny w przybliżeniu podobny i dostarczający zbliżonej mocy wyjściowej. Radiolatarnie QRSS pracują z tak niskimi mocami, ze nie trzeba wykorzystywać tranzystorów do granic ich możliwości i łatwo jest je zastąpić przez inne nawet tylko trochę zbliżone typy.

Nadajniki z bezpośrednią syntezą cyfrową

Nadajniki z bezpośrednią syntezą cyfrową (ang. *DDS*) umożliwiają łatwą generację sygnałów wielu emisji cyfrowych – zwłaszcza polegających na kluczowaniu fazy lub częstotliwości – jak PSK31, JT65, WSPR, FSK441, JT6M, MT-Hell, Slowfeld i RTTY. Kluczowanie lub modulacja amplitudy wymaga użycia dodatkowego modulatora, co zresztą nie jest bardzo skomplikowane. Amatorskie konstrukcje są obecnie przeważnie oparte na scalonych syntezerach z serii AD98xx i AD995x. Generowane mogą być zarówno sygnały w.cz. o częstotliwości pracy radiolatarni jak i sygnały m.cz. do wysterowania nadajników SSB.
Wadą rozwiązań tego typu jest znaczny poziom składowych pasożytniczych w sygnałach wyjściowych (zwłaszcza na wyższych częstotliwościach pracy – w stosunku go częstotliwości granicznej obwodu). Wymaga to użycia bardziej rozbudowanych flitrów dolnoprzepustowych. Teoretycznie zakres częstotliwości wyjściowych syntezera może sięgać do połowy częstotliwości zegarowej ale w praktyce nie przekracza on jej 40%.
Scalone syntezery AD985x są sterowane z generatorów wzorcowych – przeważnie – 10 MHz, które mogą być z kolei stabilizowane za pomocą dokładnych wzorców częstotliwości takich jak GPS, lokalne wzorce rubidowe, Warszawa 1(225 kHz) albo DCF-77. Odbiornik GPS może służyć także do synchronizacji czasu dla emisji tego wymagających (JT65, WSPR itp.).
Na rynku dostępny jest szereg modułów, a w internecie opisów konstrukcji syntezerów na AD9834, AD9835, AD9850 – 52, AD9854, AD9950 itp. o częstotliwościach granicznych od około 30 do ponad 100 MHz. Mogą być one łatwo sterowane przez programy pracujące na Arduino lub innych mikroprocesorach popularnych w środowisku krótkofalarskim – wystarczą nawet 16F84 czy 16F628. Układy sterująco-kluczujące mogą służyć nie tylko do transmisji stałych komunikatów ale także komunikatów zawierających dane telemetryczne.
W dalszej części skryptu przytoczono kilka przykładów rozwiązań takich nadajników dla różnych emisji wraz z prostymi programami.

Układy pomocnicze

Filtry dolnoprzepustowe



Rys. 3.1. Filtr dolnoprzepustowy 7 rzędu

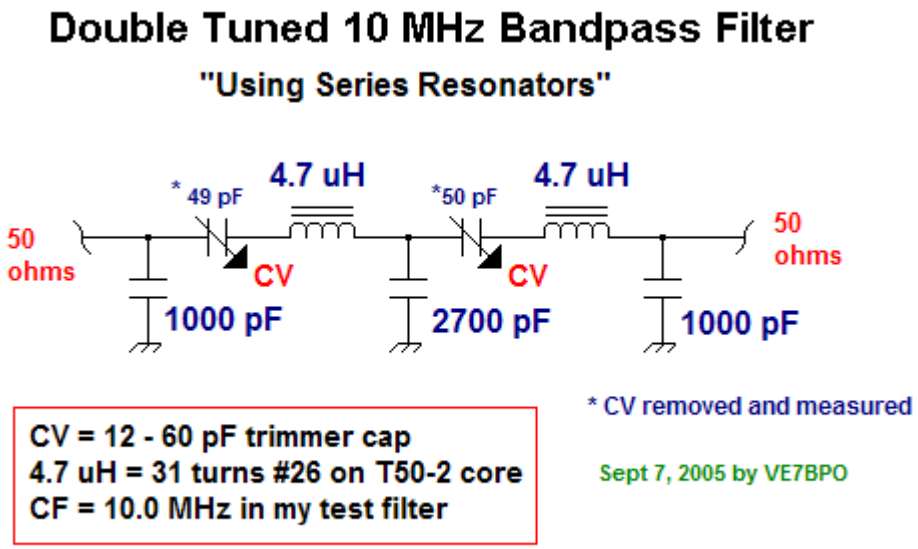
Tabela 3.1. Elementy filtru dolnoprzepustowego 7 rzędu o opornościach wejściowej i wyjściowej 50 Ω dla różnych pasm amatorskich

Pasmo [MHz[	F gr. [MHz]	F -3 dB [MHz]	F -30 dB [MHz]	C1,7 [pF]	C3,5 [pF]	L2,6 [µH]	L4 [µH]
0,137				2,2 n//10 n	4,7 n//10 n		
0,500				2,2 n//2,2 n	10 n		
1,8	2,16	2,76	4,0	820	2200	4,442	5,608
3,5	4,125	5,11	7,3	470	1200	2,434	3,012
5,2				680	1200		
7,0	7,36	9,04	12,9	270	680	1,380	1,698
10,1	10,37	11,62	15,8	270	560	1,090	1,257
14,0	14,40	16,41	22,5	180	390	0,773	0,904
18,068	18,93	22,89	32,3	110	270	0,548	0,668
21,0	21,55	27,62	39,9	82	220	0,444	0,561
24,98	25,24	28,94	39,8	100	220	0,438	0,515
28–30	31,66	40,52	58,5	56	150	0,303	0,382

Pasmo [MHz]	L2,6 [zw.]	L4 [zw.]	Rdzeń	Przewód nr
0,137	105	105	T50-2	
0,500	64	70	T50-2	
1,8	30	34	T50-2	30
3,5	25	27	T37-2	28
5,2	23	24	T37-2	
7,0	19	21	T37-6	26
10,1	19	20	T37-6	26
14,0	16	17	T37-6	24
18,068	13	15	T37-6	24
21,0	12	14	T37-6	24
24,98	12	13	T37-6	22
28–30	10	11	T37-6	22

Oznaczenie przewodu wg normy amerykańskiej. Średnica przewodu nie jest krytyczna, uzwojenie musi się tylko zmieścić na rdzeniu. Typy rdzeni wybrane dla mocy nie przekraczających 10 W. Dla fal długich i średnich pojemności podane są w nF i uzyskiwane przez równoległe łączenie kondensatorów.

Filtr pasmowy na pasmo 10 MHz

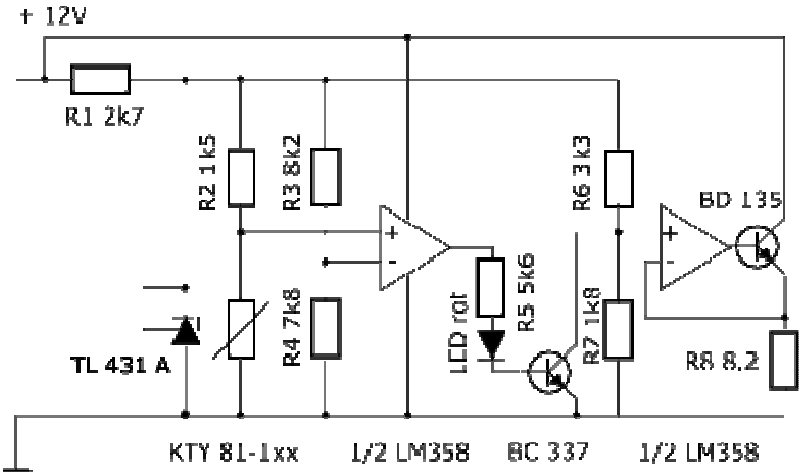


Rys.3.2.Schemat strojonego filtru pasmowego

Alternatywą dla filtrów dolnprzepustowych na wyjściu nadajników mogą być filtry pasmowe. Na schemacie 3.2 przedstawiono przykład rozwiązania takiego filtru dla pasma 10 MHz.
Kondensatory CV są trymerami o pojemności ok. 12 – 60 pF a cewki składają się 31 zwojów przewodu nawiniętych na rdzeniach pierścieniowych T50-2 (czerwonych).

Stabilizacja temperatury rezonatorów

Układ stabilizatora temperatury na LM358



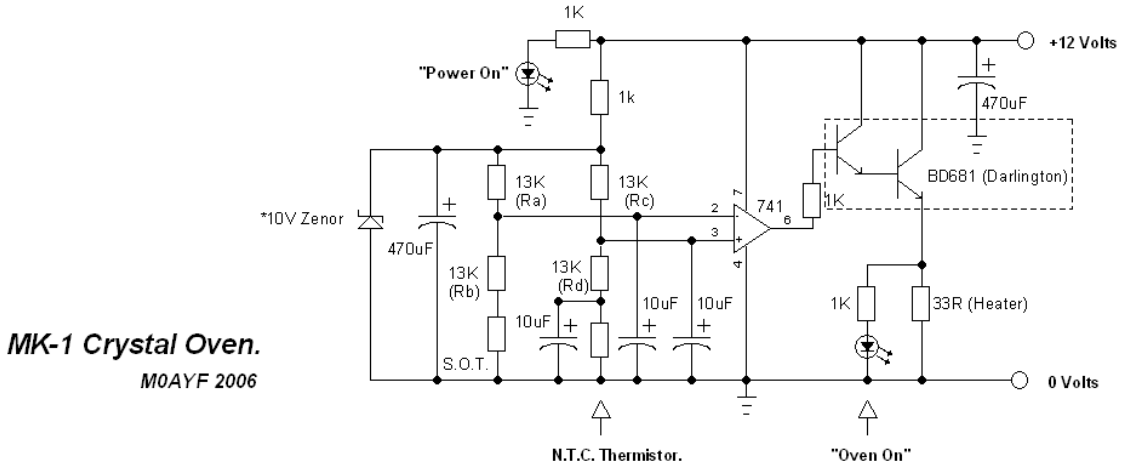
Rys. 3.3. Stabilizator temperatury ze wzmacniaczem operacyjnym LM358

Programowany stabilizator TL431 utrzymuje napięcie zasilania dzielników oporowych na poziomie 2,5 V.
Termistor typu KTY81-1xx (1 kΩ przy 25 °) – dowolnie 101, 120, 121, ...
Dla termistora KTY81-2xx należy zwiększyć opornik szeregowy R2 do 3,3 kΩ.
Dzielnik R3R4 ustala próg odniesienia dla komparatora. Elementy należy dobrać tak aby próg przełączania grzejnika leżał w okolicach 60 ° (jest to zależne od charakterystyki temperaturowej kwarcu).
W tym celu należy zmierzyć najpierw oporność termistora w tej temperaturze o dzielnik dobrać tak aby

na wejściu komparatora panowało napięcie równe w przybliżeniu napięciu na termistorze w temperaturze 60 °. Drugi wzmacniacz operacyjny wraz z tranzystorem BD135 tworzą źródło prądowe o natężeniu ustalonym przez dzielnik R6R7. Jako element grzejny służy opornik R8. Dla wartości elementów podanych na schemacie natężenie prądu płynącego przez opornik R8 jest ograniczone do około 100 mA. Dioda świecąca sygnalizuje wyłączenie grzejnika. Zamiast wzmacniacza operacyjnego LM358 można zastosować LM324 lub inny, w którym napięcie wyjściowe może schodzić do poziomu masy..

Układ stabilizacji MK-1 autorstwa M0AYF

Do pomiaru temperatury kwarcu służy umieszczony na jego obudowie termistor . Napięcie otrzymane z dzielnika, w którego obwód jest on włączony jest porównywane w komparatorze (LM741, µA741) z napięciem wzorcowym pochodzącym z dzielnika Ra, Rb, SOT a napięcie wyjściowe z komparatora steruje tranzystor BD681 kluczujący prąd płynący przez opornik 33 Ω/25 W służący jako grzejnik. Jest on również umieszczony na obudowie kwarcu lub oscylatora albo w izolowanej termicznie obudowie mieszczącej generator. Włączona równolegle dioda świecąca sygnalizuje fazy grzania ale nie jest ona niezbędna. Opornik *SOT* jest dobierany w zależności od pożądanej temperatury kwarcu. W konstrukcji M0AYF dla temperatury 40 ° jego wartość wynosiła 452 Ω. Jego dokładna wartość zależy nie tylko od pożądanej temperatury kwarcu ale także od tolerancji oporników Ra, Rb, Rc i Rd i typu termistora. Zmniejszenie oporności *SOT* powoduje wzrost wartości stabilizowanej temperatury. Tranzystor Darlingtona BD681 można zastąpić przez dwa pojedyńcze. Musi on być umieszczony na niewielkim radiatorze.
Wartości oporników Ra, Rb, Rc i Rd nie są krytyczne i mogą wynosić 10, 12, 13 lub 15 kΩ ale powinny być dobrane tak, aby miały identyczne wartości (wystarczy zmierzyć je zwykłym omomierzem). Dioda Zenera jest konieczna tylko jeśli układ jest zasilany napięciem niestabilizowanym.



NOTES:

1) *10V Zenor only required if the supply is un-regulated.

2) The S.O.T. resistor is chosen to give the desired regulated temperature. In my oven this turned out to be 452 Ohms to give a temperature of 40 degrees-C. This is only a guide and the exact value depends on other component tollerances, the values of Ra, Rb, Rc, Rd, and on the type of thermistor used.
Reducing the value of S.O.T. (Select on test) will increase the regulated temperature. Increasing the value of S.O.T. will reduce the regulated temperature.

3) The 33 Ohm "Heater" resistor is a 25 Watt component which is bolted to the oscillator's enclosure. (See images below)

4) The BD681 Darlington can be replaced with two descrete components if desired. A small heatsink is required on the BD681 device (or fitted to the second device in the case of descrete components being used)

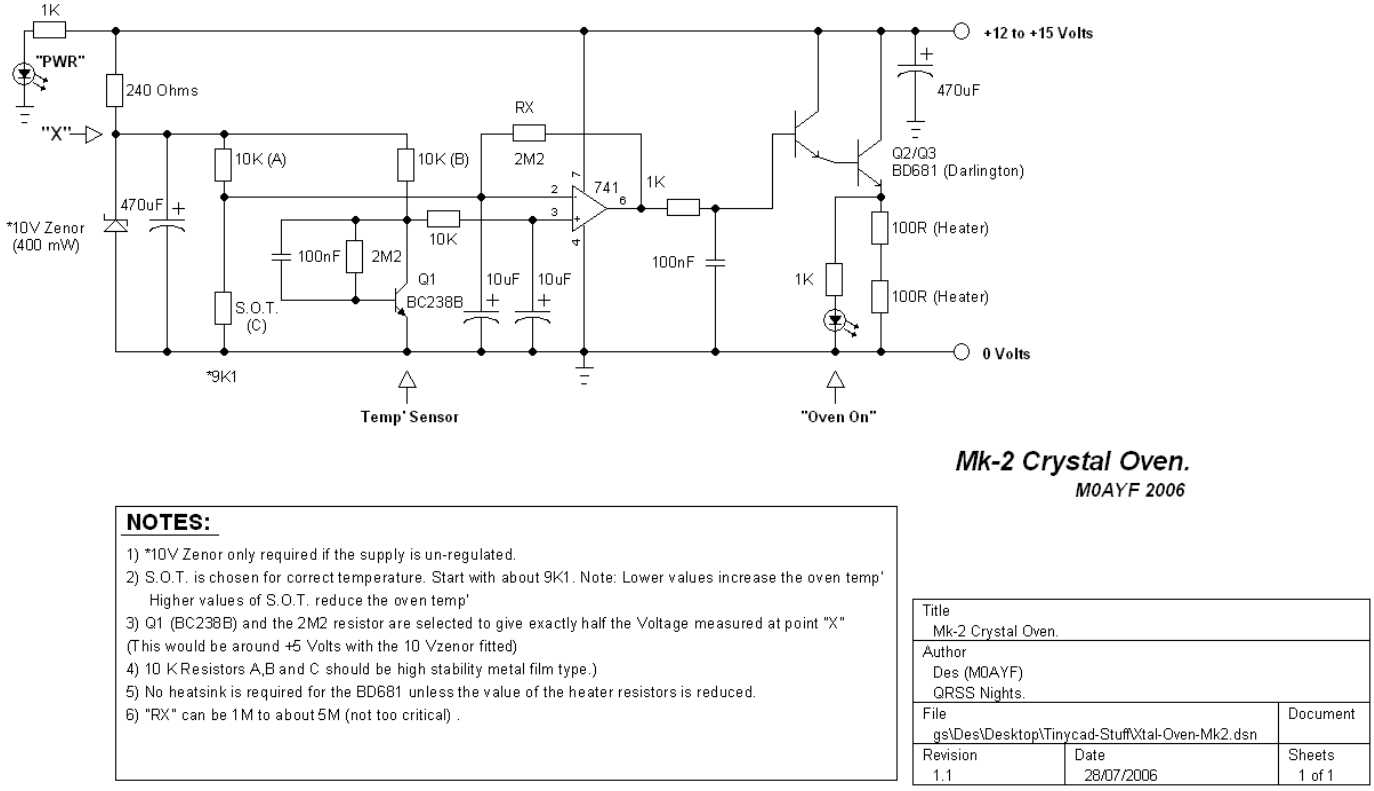
5) Ra, Rb, Rc and Rd can be 10K, 12K, 13K, 15K (not critical) but they MUST be closely matched to each other. e.g. Take a "bunch" of 10K resistors and match the resistance of four of them with a multimeter.

Title: MK-1 Oven Circuit		
Author: Des (M0AYF) QRSS Nights.		
File: gs\Des\Desktop\Tinycad-Stuff\Mk1-Xtal-Oven.dsn		Document
Revision: 1.1	Date: 19/06/2006	Sheets: 1 of 1

Rys. 3.4. Układ stabilizacji temperatury kwarcu z termistorem

Układ stabilizacji MK-2

W opracowanym przez tego samego autora układzie MK-2 jako czujnika temperatury użyto tranzystora BC238B (zamiast termistora jak w rozwiązaniu poprzednim). Jak wiadomo wartość spadku napiecia na złączu baza-emiter jest liniowo zależna od temperatury.
Układ jest poza tym zasadniczo podobny do poprzedniego i prawie wszystkie uwagi podane w poprzednim opisie odnoszą się także i do tego rozwiązania. Jako grzejnik służą dwa połączone szeregowo oporniki 100 Ω. Zmniejszenie prądu grzania oznacza, że tranzystor BD681 nie musi być umieszczony na radiatorze.
W układzie skonstruowanym przez M0AYF wartość dobieranego opornika *SOT* wynosiła 9,1 kΩ. Tranzystor BC238B i opornik jego polaryzacji bazy (2,2 MΩ) są dobierane tak, aby napięcie na jego kolektorze było równe połowie napięcia mierzonego w punkcie „X”.



NOTES:

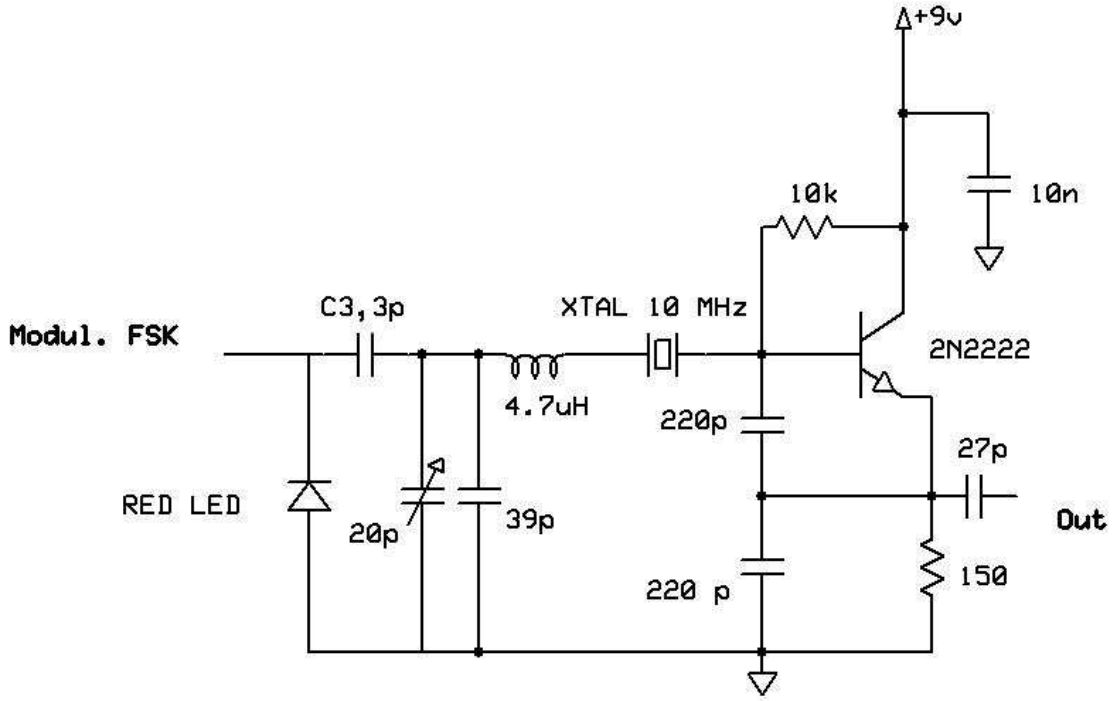
1) *10V Zenor only required if the supply is un-regulated.
2) S.O.T. is chosen for correct temperature. Start with about 9K1. Note: Lower values increase the oven temp' Higher values of S.O.T. reduce the oven temp'
3) Q1 (BC238B) and the 2M2 resistor are selected to give exactly half the Voltage measured at point "X" (This would be around +5 Volts with the 10 Vzenor fitted)
4) 10 K Resistors A,B and C should be high stability metal film type.)
5) No heatsink is required for the BD681 unless the value of the heater resistors is reduced.
6) "RX" can be 1M to about 5M (not too critical) .

Title: Mk-2 Crystal Oven.		
Author: Des (M0AYF) QRSS Nights.		
File: gs\Des\Desktop\Tinycad-Stuff\Xtal-Oven-Mk2.dsn		Document
Revision: 1.1	Date: 28/07/2006	Sheets: 1 of 1

Rys. 3.5. Układ stabilizacji temperatury kwarcu z czujnikiem tranzystorowym

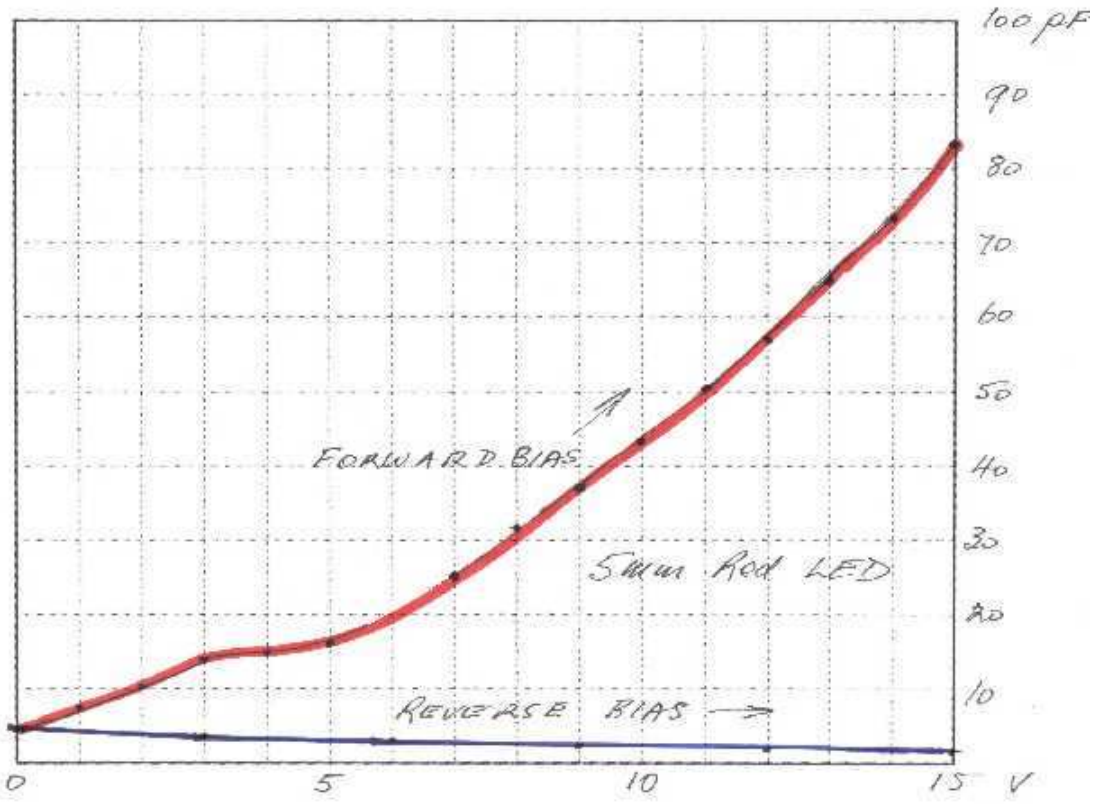
Kluczowanie częstotliwości

Kwarcowy generator VXO

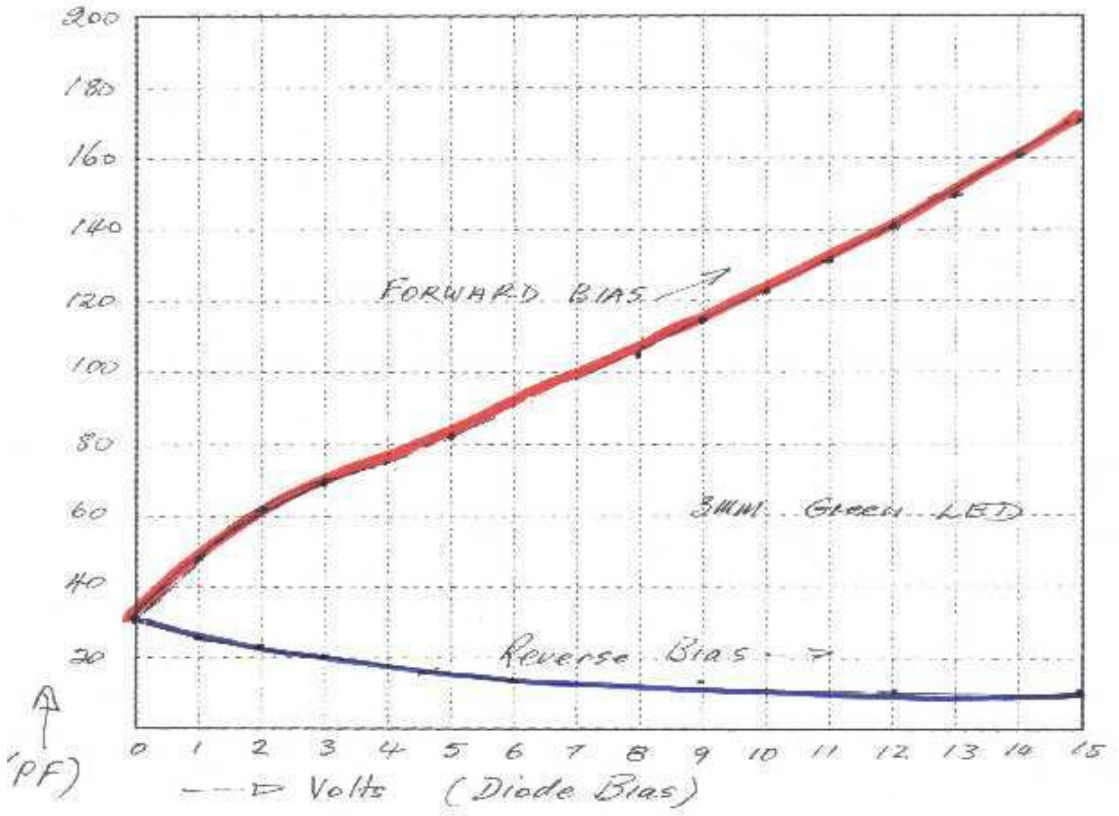


Rys. 3.6. Układ VXO kluczowanego częstotliwościowo

W układzie VXO na pasmo 30 m do modulacji lub kluczowania częstotliwości zastosowano zamiast diody pojemnościowej czerwoną diodę elektroluminescencyjną (świecącą). Jest ona słabo sprzężona z obwodem kwarcu ponieważ wymagane dewiacje są niewielkie i wynoszą przeważnie od kilku do 10 Hz. W niektórych rozwiązaniach spotykane są także diody o kolorze zielonym.
Zakres przestrajania oscylatora można rozszerzyć łącząc ze sobą równolegle dwa kwarce o tej samej częstotliwości rezonasowej, czasami spotyka sie nawet rozwiązania z trzema połączonymi tak kwarcami. W jednym z nich przy użyciu trzech kwarców o częstotliwości 10,24 MHz uzyskano pokrycie pasma 30 m. W niektórych przypadkach konieczna może okazać się zmiana indukcyjności cewki włączonej w szereg z kwarcem. Dodanie równoległego kwarcu powoduje rozszserzenie zakresu przestrajania dzięki zmniejszeniu wypadkowej dobroci obwodu ale odbywa się to kosztem pewnego pogorszenia stabilności częstotliwości. Dlatego też w praktyce rzadko spotyka się pracę więcej niż dwóch kwarców równoległych a w miarę zwiększania ich liczby stabilność częstotliwości zbliża się coraz bardziej do stabilności generatora samowzbudnego i może okazać się niedostateczna jak na potrzeby transmisji wąskopasmowych np. QRSS.
Na wykresie 3.7 przedstawione są przykładowe charakterystyki zależności pojemności czerwonej diody elektroluminescencyjnej przy polaryzacji w kierunku przewodzenia (linia czerwona) i zaporowym (linia granatowa u dołu).
Drugi z wykresów (3.8) przedstawia te same zależności dla diody zielonej o średnicy 3 mm. Powodem do za–stąpienia przez niektórych konstruktorów diod waraktorowych przez elektroluminescencyjne jest łatwiejsza dostępność tych ostatnich – przynajmniej w niektórych krajach czy rejonach.



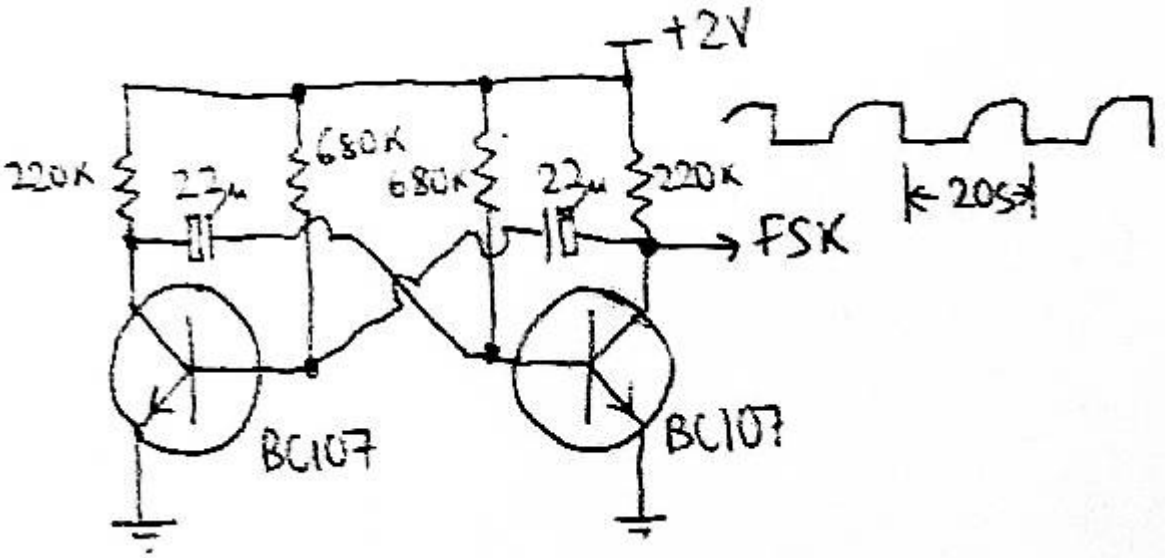
Rys. 3.7. Charakterystyki pojemnościowe diod czerwonych



Rys. 3.8. Charakterystyki pojemnościowe diod zielonych

Przerzutnik kluczujący falą prostokątną

Prosty układ przerzutnika astabilnego dostarczającego fali w przybliżeniu prostokątnej do kluczowania częstotliwości nadajników radiolatarni zawiera jedynie dwa tranzystory NPN dowolnego typu, np. BC107,BC547. Jest on zasilany stosunkowo niskim napięciem 2 V. O okresie (częstotliwości) sygnału wyjściowego decydują stałe czasu w obwodach baz tranzystorów (oporniki 680 kΩ i kondensatory 22 µF).

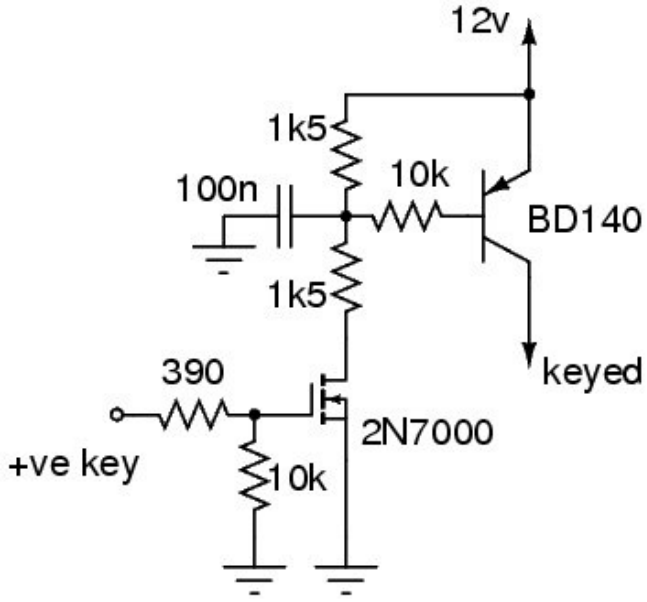


Rys. 3.9. Schemat przerzutnika

Kluczowanie amplitudy

Układ kluczujący napięciem dodatnim

W wielu przedstawionych powyżej układach nadajników do kluczowania napięcia zasilania stopnia mocy stosowany był włączony w szereg tranzystor PNP o odpowiednim dopuszczalnym prądzie kolektora. Kluczowanie odbywało się tam poprzez zwarcie wejścia (obwodu bazy) do masy. Obwody RC w bazie służyły do ukształtowania zboczy sygnału telegraficznego tak, aby zapobiec stukom na początku i końcu elementu i powodowanym przez nie zakłóceniom pracy innych użytkowników pasma.

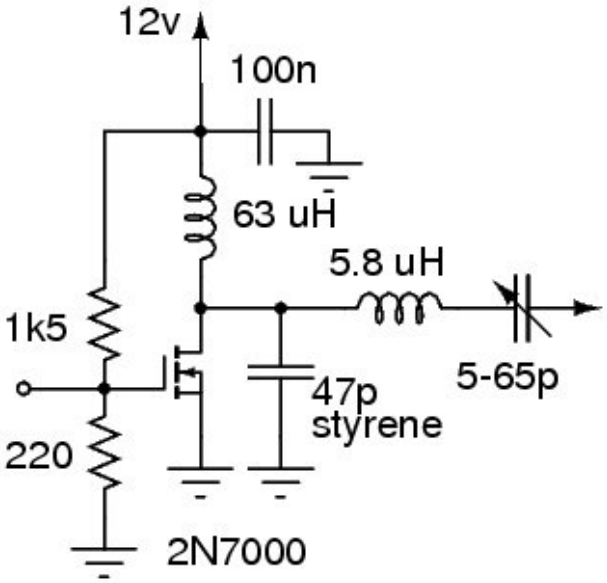


W odróżnieniu od nich układ z rys. 3.10 zawiera dodatkowy stopień odwracający polaryzację sygnału kluczującego – nadajnik jest więc kluczowany za pomocą napięcia dodatniego zamiast poprzez zwarcie wejścia do masy. Rozwiązanie to może być korzystne w przypadku korzystania z mikroprocesorowych układów kluczujących.

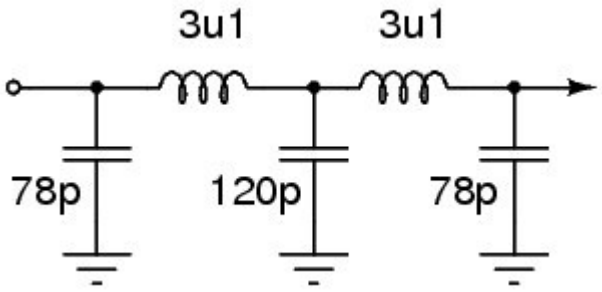
Rys. 3.10.Układ kluczowania amplitudy

Sygnał kluczujący jest podawany na bramkę tranzystora 2N7000 a wzmacniacz mocy jest podłączony do obwodu kolektora BD140.

Wzmacniacz w klasie E na pasmo 30 m



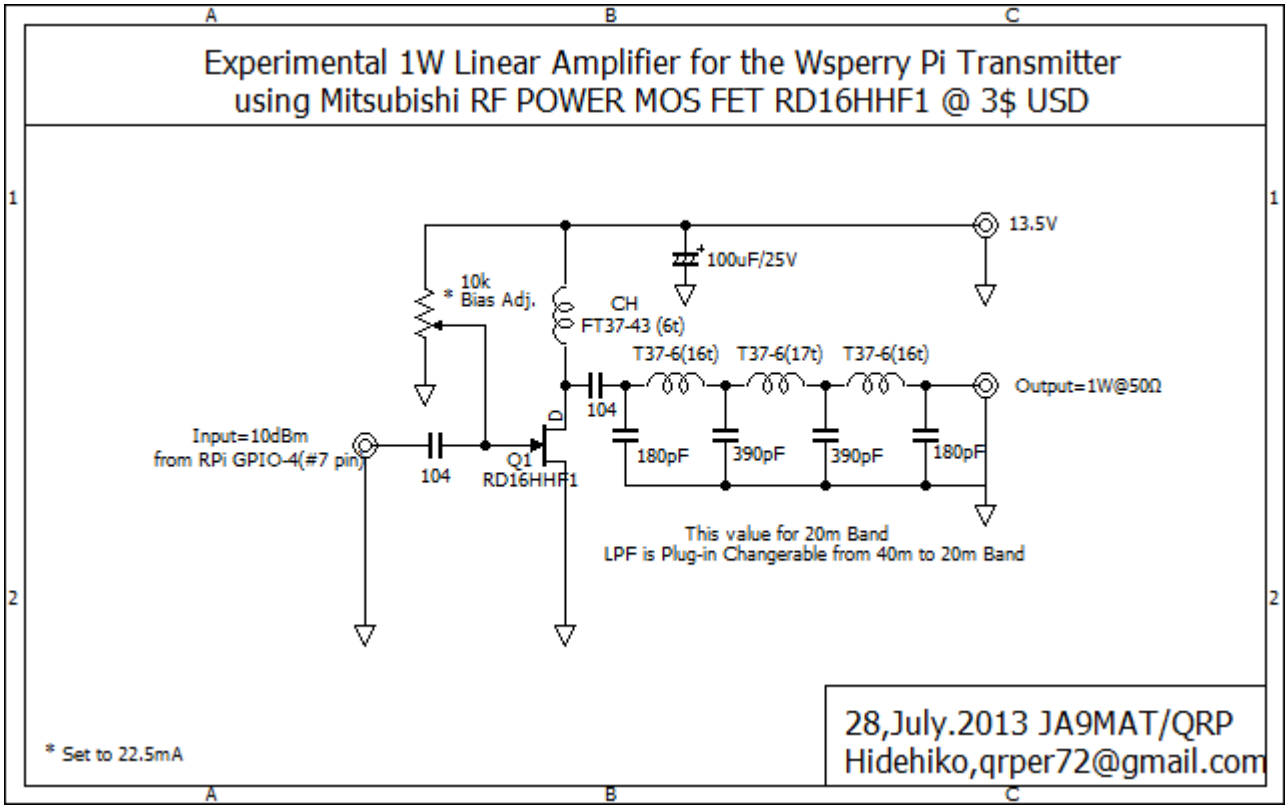
Wzmacniacz klasy E charakteryzuje się większą sprawnością aniżeli wzmacniacze klasy C – łatwe do uzyskania są wartości rzędu 80–90%. Uzyskuje się ją dzięki temu, że tranzystor jest przełączany w momentach (prawie) zerowego napięcia drenu i w związku z tym w chwili przełączania płynie przez niego znikomy prąd (obciążenie tranzystora musi mieć charakter reaktancyjny). Zmniejsza to w znacznym stopniu straty mocy w momentach przełączania. Wymagają one także mniejszych mocy sterowania aniżeli wzmacniacze w klasie C. W nadajnikach małej i bardzo małej mocy nie jest to wprawdzie sprawa bardzo istotna (chyba, że chodzi o uzyskanie możliwie długiego czasu pracy przy zasilaniu bateryjnym) ale niemniej warto zapoznać się i z układami tego typu. Rozwiązanie przedstawione na schemacie 3.11 dostarcza 2 W mocy wyjściowej na obciążeniu 50 Ω przy wysterowaniu 25 mW i napięciu zasilania 13,8 V (1,5 W przy napięciu 12 V). Tranzystor tylko nieznacznie się nagrzewa a napięcie polaryzacji bramki też jest stosunkowo niskie.
Dławik zasilający składa się z 12 zwojów przewodu nawiniętych na rdzeniu ferrytowym FT50-43. Dokładna wartość indukcyjności nie jest krytyczna i zasadniczo może być ona nawet mniejsza niż podano na schemacie. Zasada pracy wzmacniacza w klasie E polega na tym, że tranzystor jest włączany w momencie gdy w obwodzie rezonasowym złożonym z dławika i kondensatora obciążającego (w układzie 47 pF) panuje napięcie zerowe. Wzmacniacz powinien być obciążony niską impedancją tylko dla częstotliwości pracy dlatego też pomiędzy filterm dolnoprzepustowym a wzmacniaczem włączony jest szeregowy obwód rezonansowy. Dla prawidłowej pracy wzmacniacza w klasie E istotny jest natomiast właściwy współczynnik wypełnienia przebiegu sterującego. W podanym przykładzie wzmacniacz musi być zasilany przebiegiem symetrycznym czyli mającym współczynnik wypełnienia 50 %.
Cewka szeregowa w obwodzie wyjściowym składa się z 25 zwojów przewodu nawiniętych na rdzeniu T68-6 (żółtym) lub nawet na T50-6.
Do najpoważniejszych wad wzmacniaczy w klasie E należą wrażliwość na przeciążenia (zbyt niską impedancję obciążenia) i niedopasowanie anteny. W obu tych przypadkach może stosunkowo łatwo dojść do zniszczenia tranzystora. Obwody zabezpieczające muszą być bardziej rozbudowane aniżeli w przypadku wzmacniaczy w klasie C.
Skuteczną regulację mocy wyjściowej uzyskuje się jedynie przez zmianę napięcia zasilania. Natomiast regulacja poprzez zmianę polaryzacji bramki powoduje niestety obniżenie sprawności wzmacniacza. Tranzystory o małej pojemności bramki j.np. 2N7000 (Cwe = 25 pF) lepiej nadają się do pracy w klasie C aniżeli np. IRF510 (Cwe = 190 pF) itp. ponieważ czasy przełączania są wyraźnie krótsze. Powyżej 10 MHz sprawność wzmacniaczy na tranzystorach 2N7000 jednak szybko maleje i wynosi przykładowo w paśmie 14 MHz tylko około 70 % ponieważ czasy przełączania zaczynają odgrywać coraz większą rolę w stosunku do okresu nadawanej fali. Dla tranzystorów IRF510 sprawność zaczyna maleć już powyżej 2 MHz. Dodatkowo wymagają one także wyraźnie większej mocy sterującej aniżeli 2N7000.
Na wyjściu wzmacniacza należy włączyć filtr dolnoprzepustowy. Przykład rozwiązania filtru przedstawiono na schemacie 3.12. Indukcyjności 3,1 µH uzyskano przez nawinięcie 28 zwojów przewodu na rdzeniach pierścieniowych T50-2. W filtrze zastosowano kondensatory ceramiczne.



Rys. 3.12. Dwusekcyjny filtr dolnoprzepustowy na pasmo 30 m

Wzmacniacz 1 W na pasmo 20 m

Układ opracowany przez JA9MAT zawiera nowoczesny tranzystor mocy typu RD16HHF1 i dostarcza przy napięciu zasilania 13,5 V mocy 1 W w paśmie 20 m. Po zmianie wartości elementów wyjściowego filtru dolnoprzepustowego można go jednak łatwo dostosować do pracy w innych amatorskich pasmach krótkofalowych.
Dławik w obwodzie zasilania tranzystora składa się z 6 zwojów nawiniętych na rdzeniu ferrytowym FT37-43 a dławiki filtru dolnoprzepustowego są nawinięte na rdzeniach pierścieniowych T37-6 (żółtych) i mają odpowiednio po 16, 17 i 16 zwojów.
Prąd spoczynkowy tranzystora wynosi ok. 22,5 mA.
Moc wyjściową można zmniejszyć dobierając polaryzację bramki tranzystora za pomocą potencjometru montażowego 10 kΩ. Wzmacniacz został wprawdzie przewidziany przez konstruktora do współpracy z radiolatarnią na mikrokomputerze „Raspberry Pi" ale może on być oczywiście użyty i w innych układach nadajników.



Rys. 3.13. Wzmacniacz mocy na RD16HHF1

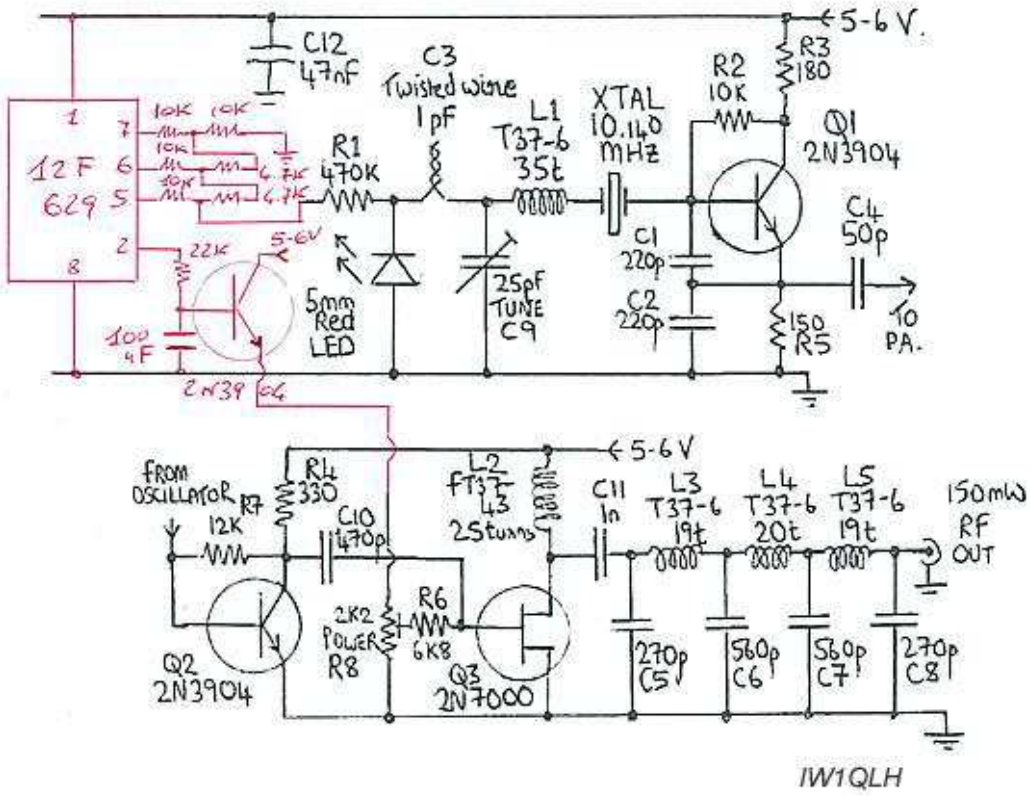
Sterowniki mikroprocesorowe

Rozdział 4 zawiera wybrane przykłady programów radiolatarni dla różrnych emisji pracujących na niektórych najbardziej rozpowszechnionych typach mikroprocesorów jak seria PIC, AVR i moduły Arduino. Dla Arduino podano też przykłady sterowania syntezera cyfrowego (DDS) i generacji sygnału akustycznego – z jednej strony jako rozszerzenie w odniesieniu do programów radiolatarni a z drugiej jako przykład realizacji funkcji, które mogą się przydać do własnych celów. Wszystkie z przytoczonych przykładów wymagają co najmniej wprowadzenia do kodu własnego znaku wywoławczego i ewentualnych innych własnych danych jak QTH, lokator itp. Są one jednak w pierwszym rzędzie pomyślane jako materiał dydaktyczny. Bardziej doświadczeni programiści mogą je stosunkowo łatwo dostosować do innych podobnych typów mikroprocesorów. W programach wykorzystywanych lub modyfikowanych przez OE1KDA zmiany są komentowane po polsku a wśród danych znajduje się jego znak i dalsze informacje o stacji. W pozostałych programach pozostawiono znaki wywoławcze autora i komentarze w języku oryginału z zachowaniem pisowni i czasami omyłek autora. Nazwiska i ewentualne znaki wywoławcze autorów podane są w komentarzach i opisach programów a wstawienie w publikacji znaku OE1KDA w niektórych miejscach jako dane robocze nie oznacza przywłaszczenia sobie autorstwa ani współautorstwa danego programu.
Dla ułatwienia analizy przy wielu programach zamieszczono także schematy ideowe albo blokowe obrazujące sposób podłączenia mikroprocesora i wykorzystania jego wejść i wyjść. Podane w niektórych przykładach dodatkowe pliki nagłówkowe i podobne są przeważnie zawarte w odpowiednim środowisku programistycznym i dlatego zrezygnowano z ich przytaczania. Wykorzystanie programów wymaga zainstalowania środowiska programistycznego i ewentualnych dodatkowych kompilatorów. Środowiska dla mikroprocesorów PIC (MPLAB) i Arduino oraz kompilator C dla PIC są dostępne bezpłatnie w Internecie.
Wiele gotowych programów w postaci szesnastkowej – do bezpośredniego zaprogramowania mikrokomputerów – można jednak znaleźć bez trudu w internecie m.in. pod podanymi na końcu skryptu adresami lub podając ich nazwy w wyszukiwarce. Dane osobiste i podstawowe komunikaty umieszczane są często w pamięci EEPROM procesora a ich zmiany można dokonać łatwo w programatorze bez dekompilacji programu.

Procesory PIC

Radiolatarnia CW, QRSS i Hella na procesorze 12F629

W przedstawionym na rys. 4.1 układzie procesor ATtiny13 został zastąpiony przez 12F629. Od układu radiolatarni Hansa Summersa G0UPL omawianej poprzednio różni się on tylko procesorem i sposobem jego podłączenia (część narysowana na czerwono). Cała reszta układu jest identyczna i nie wymaga ponownego omówienia.
Program radiolatarni nadającej kolejno komunikaty telegrafią zwykłą (CW, 20 sł./min.), telegrafią wolną (QRSS6) i w systemie Hella opracowany przez IW1QLH (*www.iw1qlh.net*) pracuje na popularnym mikroprocesorze PIC 12F629. Przed jego skompilowaniem należy do kodu wprowadzić własny znak. Transmisja w systemie Hella (Slowfeld) wymaga użycia przetwornika cyfrowo-analogowego. W tym układzie jest to przetwornik drabinkowy typu R-2R złożony z oprników podłączonych do nóżek 5, 6 i 7 mikroprocesora. Nóżka 2 jest wyjściem sygnału kluczującego amplitudę za pośrednictwem tranzystora NPN 2N3904. Można go łatwo zastąpić przez tranzystor europejski dowolnego typu np. BC107.



Rys. 4.1. Radiolatarnia IW1QLH

```
#include <pic.h>
#include "delay.c"

__CONFIG(0x3FFF & BOREN & MCLRDIS & PWRTDIS & WDTEN & INTIO & WDTEN);

// REVISIONE 3 - WATCHDOG
// REVISIONE 2 - CW STANDARD (dot=150mS) + CW LENTO (dot=6S) + HELL

#define CW 0
#define QRSS 2
#define HELL 1

eeprom char callsign[32] = {
        'I', 'W', '1', 'Q', 'L', 'H', '/', 'B',  0xFF };

eeprom char hell[64] =
{
        0,127,127,0,3,28,96,24,96,
        28,3,0,2,127,0,62,65,
        65,81,97,190,0,0,127,64,
        64,64,0,127,8,8,8,8,
        127,0,64,32,16,8,
        4,2,0,127,73,73,73,73,
        54,
        0xFF
};

const char cwtable[128] = {
        ' ', 0, 0xFF,
        'I', 0b10100000, 0xFF,
        'W', 0b10111011, 0b10000000, 0xFF,
```

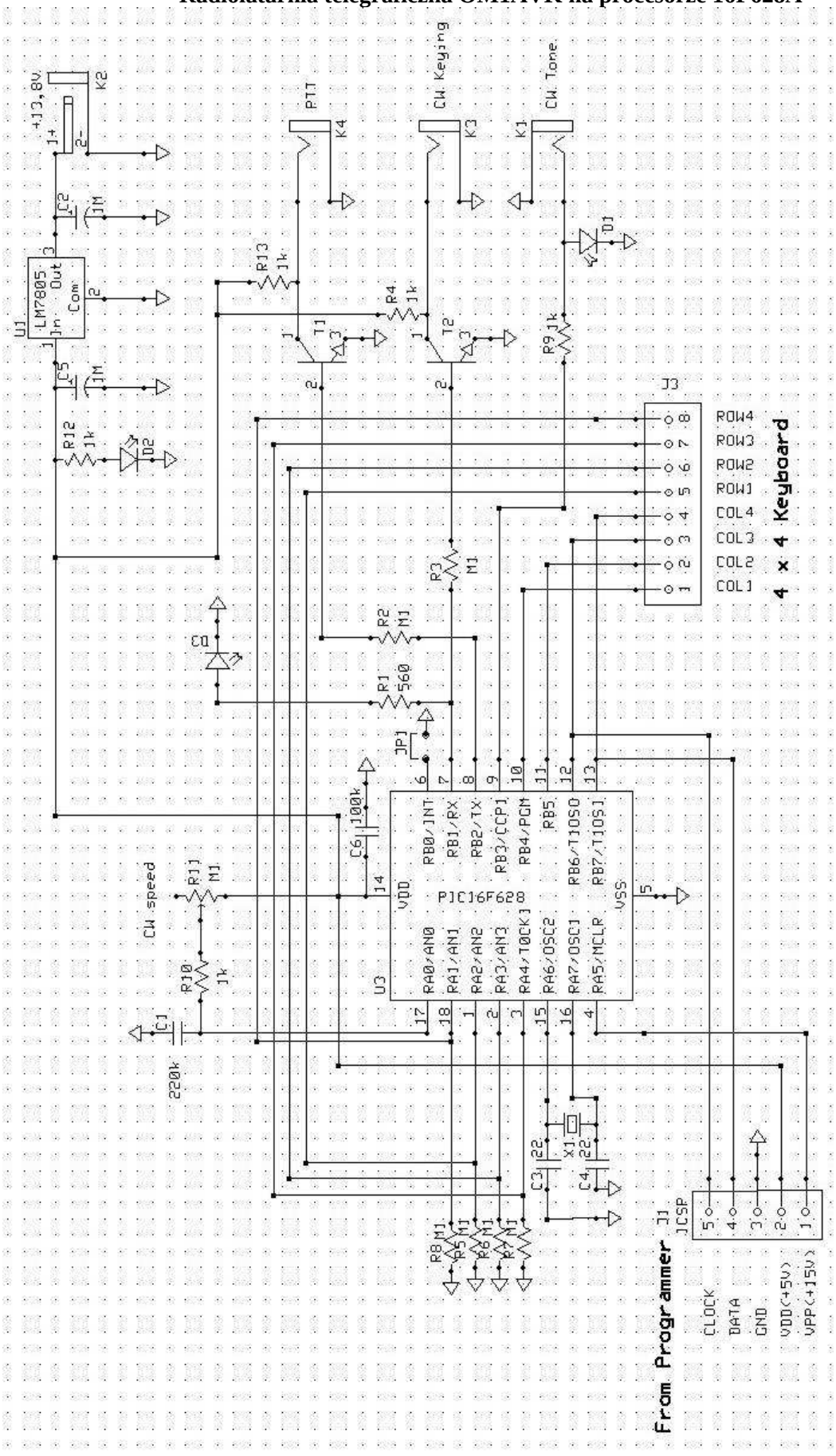
```
        '1', 0b10111011, 0b10111011, 0b10000000, 0xFF,
        '3', 0b10101011, 0b10111000, 0xFF,
        '4', 0b10101010, 0b11100000, 0xFF,
        'Q', 0b11101110, 0b10111000, 0xFF,
        'L', 0b10111010, 0b10000000, 0xFF,
        'H', 0b10101010, 0xFF,
        '/', 0b11101010, 0b11101000, 0xFF,
        'B', 0b11101010, 0b10000000, 0xFF,
        'R', 0b10111010, 0xFF,
        'P', 0b10111011, 0b10100000, 0xFF,
        'J', 0b10111011, 0b10111000, 0xFF,
        'N', 0b11101000, 0xFF,
        'A', 0b10111000, 0xFF,
        'V', 0b10101011, 0b10000000, 0xFF,
        0xFF, 0xFF
        };

char c, b;
char i, icw;
unsigned char mode, dac;
unsigned int count;

void txOff(void)
{
        GPIO = 0;
}

#define TXON 0b00100000

void txOn(void)
{
        GPIO5 = 1;
}

void cwOff(void)
{
        switch (mode)
        {
                case CW:
                        txOff();
                        break;
                case QRSS:
                        GPIO = 0b0000000 | TXON;
                        txOn();
                        break;
        }
}

void cwOn(void)
{
        switch (mode)
        {
                case CW:
                        txOn();
                        break;
                case QRSS:
                        GPIO = 0b00000100 | TXON;
                        txOn();
                        break;
        }
}


void DelayDot(void)
{
        unsigned char n;
        switch (mode)
        {
```

```
                case CW:
                case HELL:
                        n = 0;
                        break;
                case QRSS:
                        n = 39;
                        break;
        }
        do
        {
                DelayMs(150);
                CLRWDT();
        }
        while (n--);
}

void incDac(void)
{
        dac++;
        icw = icw >> 1;
        if (dac > 7)
        {
                dac = 0;
                icw = 0b10000000;
        }
}

void main(void)
{
        CLRWDT();
        //OPTION  = 0b11111111; // (default - prescaler 1:128)

        INTCON = 0;
        CMCON   = 0b00000111;    // Comparator OFF
        //ANSEL    = 0b00000000;
        TRISIO     = 0b00011000;
        GPIO= 0;

        mode = CW;
        goto main_loop;

next_mode:
        mode++;
        txOff();
        SLEEP();
        SLEEP();

main_loop:

        // LOOP DIFFERENTI MODI DI TRASMISSIONE
        while(1)
        {
                switch (mode)
                {
                        case CW:
                                txOff();
                                i = (char)callsign;
                                break;
                        case QRSS:
                                txOn();
                                i = (char)callsign;
                                break;
                        case HELL:
                                txOn();
                                i = (char)hell;
                                break;
                        default:
                                txOff();
```

```
                mode = 0;
                count = 130;
                do {
                        SLEEP();
                } while (count--);
                goto main_loop;
}

// LOOP TRASMISSIONE CARATTERI
while(1)
{

        switch (mode)
        {
                case CW:
                case QRSS:

                        // SPAZIO TRA DUE CARATTERI
                        cwOff();
                        DelayDot();
                        DelayDot();
                        DelayDot();

                        c = callsign[i];
                        if (c == 0xFF)
                                goto next_mode;
                        if (c == 0)
                        {
                                for(b = 0; b < 8; b++)
                                        DelayDot();
                                i++;
                                continue;
                        }
                        icw = 0;
                        while (cwtable[icw] != c)
                        {
                                do
                                        icw++;
                                while(cwtable[icw] != 0xFF);
                                icw++;
                        }

                        icw++;
                        while (1)
                        {
                                c = cwtable[icw];
                                b = (cwtable[icw + 1] == 0xFF) ? ((c == 0) ? 0b11100000 : c) : 0xFF;
                                if (c == 0xFF)
                                        break;
                                while (b != 0)
                                {
                                        if ((c & 0b10000000) != 0)
                                                cwOn();
                                        else
                                                cwOff();
                                        c = c << 1;
                                        b = b << 1;
                                        DelayDot();
                                }
                                icw++;
                        }
                        i++;
                        break;

                case HELL:
                        c = callsign[i];
                        if (c == 0xFF)
                                goto next_mode;
```

```
                                if (c == 0)
                                {
                                        txOff();
                                        for(b = 0; b < 8; b++)
                                                DelayDot();
                                        i++;
                                        continue;
                                }

                                dac = 0;
                                icw = 0b10000000;
                                for(b = 0; b < 8; b++)
                                {
                                        while ((c & icw) == 0)
                                                incDac();
                                        GPIO = dac | TXON;
                                        DelayDot();
                                        incDac();
                                }
                                i++;
                                break;
                        }
                }
        }
}
```

Radiolatarnia telegraficzna OM1AVK na procesorze 16F628A



Rys. 4.2

```
;*********************************************************************
;*           OM1AVK -  CW MEMORY KEYER                               *
;*********************************************************************
;*          AVKEY : FIX MEMORY KEYER with PIC16F628A                 *
;*********************************************************************
;*           Freeware for radio amateur usage                        *
;*           STATUS : DEVELOPMENT FOR FINAL SOLUTION ON THE BOARD    *
;*********************************************************************
;*           OM1AVK / avk@kanich.net        *                        *
;*           special thanks to : DL4YHF                              *
;*********************************************************************
#include <p16F628.inc>
        errorlevel -302

        __CONFIG        _BODEN_ON & _CP_OFF & _DATA_CP_OFF & _PWRTE_ON &
_WDT_OFF & _LVP_OFF & _MCLRE_OFF & _XT_OSC
;=====================================================================
;       Variable Definition
;=====================================================================
;PORTA bit assignment
SW4             EQU     H'02'           ;SW1 is triggering RA2
SW3             EQU     H'03'           ;SW2 is triggering RA3
SW2             EQU     H'04'           ;SW3 is triggering RA4
SW1             EQU     H'01'           ;SW4 is triggering RA1
;General registers
        cblock  0x20
                TIMER1          ;Used in delay routine
                TIMER2          ; "     "       "
                PATERN          ;Pattern data for effect's
                TIMER_DOT       ;Basic timer for dot length
                KY_char         ;Register for image of CW  character
                KY_count        ;Counter for dashes+dots in a character
                CW_decode       ;(ASCII->internal CW code) temp
                POTI            ;poti reading counter
                POTI2           ;poti data = 3 * POTI
                TIMER_RX        ;Timer for RX wait in repeat loop
        endc
#define IOP_POTI1       PORTA, 0 ; only if poti is POLLED (NO IRQ!!)
CWCHR   macro   char,pause
        MOVLW   char
        CALL    KYT_DEC
        if pause==0
                CALL    DASH_SP
        else
                CALL    WORD_SP
        endif
        endm
;=====================================================================
;       Program Definition
;=====================================================================
                ORG     0               ;Reset vector address
                GOTO    RESET           ;goto RESET routine when boot.
; Storage format of "spaces", "control characters" and "transmittable CW
letters"
; ---------------------------------------------------------------------
;  The most significant bits define the code type.
;     Bit7: 1 = "this is SPACE (=pause) or CONTROL character", in this case:
;        Bit6,Bit5 = control character type:
;        0x = SPACE (=pause,    bits5..0 contain the length in DOTS (max.63)
;        10 = extra long "dash", bits4..0 contain the length in DOTS
(max.31)
```

```
;        11 = future reserve,    bits4..0 contain 31 possible codes
;     Bit7: 0 = "this is a transmittable character (not SPACE or CONTROL)".
;        Bits6..0 contain a maximum of 6 dashes or dots,
;             with leading 1="Startbit" before the "dash/dot-matrix",
;             dash/dot-matrix is in Bit0..max.Bit5:
;                  0=dot 1=dash
;             Bit0 always contains the LAST TRANSMITTED dash/dot.
; See definitions below for some examples how CW letters are stored in
memory.
#define CW_0     b'00111111'  ; "0"
#define CW_1     b'00101111'  ; "1"
#define CW_2     b'00100111'  ; "2"
#define CW_3     b'00100011'  ; "3"
#define CW_4     b'00100001'  ; "4"
#define CW_5     b'00100000'  ; "5"
#define CW_6     b'00110000'  ; "6"
#define CW_7     b'00111000'  ; "7"
#define CW_8     b'00111100'  ; "8"
#define CW_9     b'00111110'  ; "9"
#define CW_A     b'00000101'  ; 'A'
#define CW_B     b'00011000'  ; 'B'
#define CW_C     b'00011010'  ; 'C'
#define CW_D     b'00001100'  ; 'D'
#define CW_E     b'00000010'  ; 'E'
#define CW_F     b'00010010'  ; 'F'
#define CW_G     b'00001110'  ; 'G'
#define CW_H     b'00010000'  ; 'H'
#define CW_I     b'00000100'  ; 'I'
#define CW_J     b'00010111'  ; 'J'
#define CW_K     b'00001101'  ; 'K'
#define CW_L     b'00010100'  ; 'L'
#define CW_M     b'00000111'  ; 'M'
#define CW_N     b'00000110'  ; 'N'
#define CW_O     b'00001111'  ; 'O'
#define CW_P     b'00010110'  ; 'P'
#define CW_Q     b'00011101'  ; 'Q'
#define CW_R     b'00001010'  ; 'R'
#define CW_S     b'00001000'  ; 'S'
#define CW_T     b'00000011'  ; 'T'
#define CW_U     b'00001001'  ; 'U'
#define CW_V     b'00010001'  ; 'V'
#define CW_W     b'00001011'  ; 'W'
#define CW_X     b'00011001'  ; 'X'
#define CW_Y     b'00011011'  ; 'Y'
#define CW_Z     b'00011100'  ; 'Z'

#define CW_SEPAR b'00110001'  ; '='  (-...-)  this is the "official"
separator
#define CW_SEPA2 b'01100001'  ; '-'  (-....-) but this is also frequenty
used
#define CW_POINT b'01010101'  ; '.'  (-.-.-.)
#define CW_SLASH b'00110010'  ; '/'  (-..-.)
#define CW_?     b'01001100'  ; '?'  (..--..)
#define CW_AR    b'00101010'  ; '+[AR]' (.-.-.)
#define CW_SK    b'01000101'  ; '*[SK]' (...-.-)
#define CW_KA    b'00110101'  ; '$[KA}' (-.-.-)
#define CW_KN    b'00110110'  ; '#[KN]' (-.--.)
#define CW_EOM   b'01011111'  ; 'EOM' (.-----) used for "partitions" of msg
#define CW_NNN   b'01101010'  ; 'NNN' (-.-.-.) replaced by serial number
#define CW_ANN   b'01011010'  ; 'ANN' (.--.-.) advance to next number
#define CW_SPACE b'10000000'  ; ' ' (pause length in "dots" will be added)
;*****************************************
```

```
;*  Main delay routine                      *
;*******************************************
DELAY_ROUTINE   MOVLW   D'3'      ;54 Generate approx 10mS delay at 4Mhz CLK
                MOVWF   TIMER2
DEL_LOOP1       MOVLW   D'160'     ;60
                MOVWF   TIMER1
DEL_LOOP2       DECFSZ  TIMER1,F
                GOTO    DEL_LOOP2
                DECFSZ  TIMER2,F
                GOTO    DEL_LOOP1
               RETLW   0
        ;
;*******************************************
;*  RX delay routine (repeat function)       *
;*  with exit to MENU when ESC pressed       *
;*******************************************
DELAY_RX        MOVLW   D'3'      ;54 Generate approx 10mS delay at 4Mhz CLK
                 MOVWF   TIMER2
DEL_RX1         MOVLW   D'160'    ;60
                 MOVWF   TIMER1
DEL_RX2         BSF     PORTB,7           ;Activate (RB7)
                BCF     OPTION_REG, NOT_RBPU
                BTFSC   PORTA,SW2         ;Check ESC
                GOTO    MENU
                DECFSZ  TIMER1,F
                 GOTO    DEL_RX2
                 DECFSZ  TIMER2,F
                 GOTO    DEL_RX1
                RETLW   0
        ;
;*******************************************
;*  CW character processing                 *
;*******************************************
KYT_DEC         BSF     PORTB,2
                MOVWF   KY_char
                MOVLW   6                  ; load number of dots+dashes
                MOVWF   KY_count          ; ..into "element counter"
                BTFSC   KY_char,6          ; is it 6-element-char ?
                GOTO    kyt_play6          ; else..
                DECF    KY_count,F
                BTFSC   KY_char,5          ; is it 5-element-char ?
                GOTO    kyt_play5          ; else..
                DECF    KY_count,F
                BTFSC   KY_char,4          ; is it 4-element-char ?
                GOTO    kyt_play4          ; else..
                DECF    KY_count,F
                BTFSC   KY_char,3          ; is it 3-element-char ?
                GOTO    kyt_play3          ; else..
                DECF    KY_count,F
                BTFSC   KY_char,2          ; is it 2-element-char ?
                GOTO    kyt_play2          ; else must be 1-element-char
                DECF    KY_count,F        ; is it 1-element-char
                RLF     KY_char,F
kyt_play2       RLF     KY_char,F
kyt_play3       RLF     KY_char,F
kyt_play4       RLF     KY_char,F
kyt_play5       RLF     KY_char,F
kyt_play6       BTFSC    KY_char,5
                GOTO    KYT_DASH
                BSF     PORTB,1
                CALL    DOT
                GOTO    KYT_NEXT
```

```
KYT_DASH        BSF     PORTB,1
                CALL    DASH
KYT_NEXT        BCF     PORTB,1
                DECFSZ  KY_count,F
                GOTO    kyt_playb
                RETLW   0
kyt_playb       CALL    DOT_SP
                RLF     KY_char,F
                GOTO    kyt_play6
;
;*********************************
;**  RESET :  main boot routine  **
;*********************************
RESET           MOVLW   B'00000111'     ;Disable Comparator module's
                MOVWF   CMCON
                ;
                BSF     STATUS,RP0      ;Switch to register bank 1
                                        ;Disable pull-ups
                                        ;INT on rising edge
                                        ;TMR0 to CLKOUT
                                        ;TMR0 Incr low2high trans.
                                        ;Prescaler assign to Timer0
                                        ;Prescaler rate is 1:256
                MOVLW   B'11010111'     ;Set PIC options (See datasheet).
                MOVWF   OPTION_REG      ;Write the OPTION register.
                ;
                CLRF    INTCON          ;Disable interrupts
                MOVLW   B'00000000'
                MOVWF   TRISB           ;RB7...RB0 are outputs.
                MOVLW   B'11111110'     ;RA(1-4) ports are inputs,RA(0) are
outputs
                MOVWF   TRISA
                BCF     STATUS,RP0
                MOVLW   D'30'
                MOVWF   POTI
                MOVLW   D'90'
                MOVWF   POTI2
                GOTO    MENU
;*****************************************
;  4 x 4 keypad read & switch
;1              CQ CQ TEST DE OM1AVK OM1AVK BK
;2              ?QRZ OM1AVK BK
;3              CFM 73 TU K
;4              DE OM1AVK OM1AVK BK
;5              DE OM1AVK OM1AVK OM1AVK BK
;6              DE OM1AVK OM1AVK OM1AVK OM1AVK BK
;7              ?AGN AGN K
;8              NR? NR? BK
;9              LOC? LOC? BK
;0              OM1AVK
;A              WKD B4 TNX
;B              73 TU K
;C              ? RPT ALL BK
;D              JN88OD
;*              ESC (stop)
;#              (start)
;*****************************************
MENU            CLRF    PORTB
                CALL    SETPOTI
                MOVLW   B'00010000'     ;Activate (RB4)
                MOVWF   PORTB
                BCF     OPTION_REG, NOT_RBPU
```

```
                BTFSC   PORTA,SW1
                GOTO    MA
                BTFSC   PORTA,SW2
                GOTO    M1
                BTFSC   PORTA,SW3
                GOTO    M2
                BTFSC   PORTA,SW4
                GOTO    M3
                MOVLW   B'00100000'     ;Activate (RB5)
                MOVWF   PORTB
                BCF     OPTION_REG, NOT_RBPU
                BTFSC   PORTA,SW1
                GOTO    MB
                BTFSC   PORTA,SW2
                GOTO    M4
                BTFSC   PORTA,SW3
                GOTO    M5
                BTFSC   PORTA,SW4
                GOTO    M6
                MOVLW   B'01000000'     ;Activate (RB6)
                MOVWF   PORTB
                BCF     OPTION_REG, NOT_RBPU
                BTFSC   PORTA,SW1
                GOTO    MC
                BTFSC   PORTA,SW2
                GOTO    M7
                BTFSC   PORTA,SW3
                GOTO    M8
                BTFSC   PORTA,SW4
                GOTO    M9
                MOVLW   B'10000000'     ;Activate (RB7)
                MOVWF   PORTB
                BCF     OPTION_REG, NOT_RBPU
                BTFSC   PORTA,SW1
                GOTO    MD
                BTFSC   PORTA,SW2
                GOTO    ESC
                BTFSC   PORTA,SW3
                GOTO    M0
                BTFSC   PORTA,SW4
                GOTO    NUM
                GOTO    MENU
                ;
ReadPoti        BCF     INTCON, GIE       ;01 disable IRQs while we are in
register bank 1
                                  ; (and to prevent bad poti readings !)
                 CLRF   POTI
                 CLRF   POTI2
                BCF    IOP_POTI1 ;02 clear Poti pin output latch to
discharge
                BSF     STATUS, RP0    ;! ;03 set RP0 for TRIS access (;!)
                BCF     IOP_POTI1      ;! ;04 define PortB.0 as output ->
begin discharge
                  ; (typical discharge of 220nF from 2V to 0V takes about
100usec)
                NOP                    ;! ;05 ensure capacitor gets
completely discarged !
                BSF     IOP_POTI1      ;! ;06 define PortB.0(!!) as input ->
start e-function
                BCF     STATUS, RP0    ;! ;07 clear RP0 for "normal" access
                  ; after a certain time the state of the poti input
                  ; will toggle from '0' (which it should be now) to '1'.
```

```
LoopPoti        BTFSC   IOP_POTI1         ;08 check if already charged
                RETURN
                INCF    POTI
                BTFSC   STATUS,Z
                RETURN
                GOTO    LoopPoti
                ;
SETPOTI         CALL    ReadPoti
;               RRF     POTI
                MOVF    POTI,W
                ANDLW   H'F0'
                MOVWF   POTI
                SWAPF   POTI
                MOVLW   D'14'
                ADDWF   POTI
;               MOVF    POTI,W
;               CALL    KYT_DEC
                MOVF    POTI,W
                MOVWF   POTI2
                ADDWF   POTI2
                ADDWF   POTI2
                RETURN
;*****************************************
;* Wait routine for memory #1 repeat call    *
;*****************************************
RX_WAIT         BCF     PORTB,2         ;Stop PTT -> OFF
                MOVLW   D'200'
                MOVWF   TIMER_RX
RX_LOOP         CALL    DELAY_RX
                CALL    DELAY_RX
                DECFSZ  TIMER_RX,F
                GOTO    RX_LOOP
                RETURN
;*****************************************
;* Message generation                     *
;*****************************************
M1              CALL    CQTEST          ;CQ CQ TEST DE OM1AVK OM1AVK BK
                CALL    OM1AVK
                CALL    OM1AVK
                CALL    BK
                CALL    RX_WAIT
                GOTO    M1
                ;
M2              CALL    CQTEST          ;CQ CQ TEST DE OM1AVK OM1AVK OM1AVK BK
                CALL    OM1AVK
                CALL    OM1AVK
                CALL    OM1AVK
                CALL    BK
                GOTO    MENU
                ;
M3              CWCHR   CW_?,0          ;?QRZ OM1AVK BK
                CWCHR   CW_Q,0
                CWCHR   CW_R,0
                CWCHR   CW_Z,1
                CALL    OM1AVK
                CALL    BK
                GOTO    MENU
                ;
M4              CALL    DE1             ;DE OM1AVK OM1AVK BK
                CALL    OM1AVK
                CALL    OM1AVK
                CALL    BK
```

```
                GOTO    MENU
M5              CALL    DE1             ;DE OM1AVK OM1AVK OM1AVK BK
                CALL    OM1AVK
                CALL    OM1AVK
                CALL    OM1AVK
                CALL    BK
                GOTO    MENU
M6              CALL    DE1             ;DE OM1AVK OM1AVK OM1AVK OM1AVK BK
                CALL    OM1AVK
                CALL    OM1AVK
                CALL    OM1AVK
                CALL    OM1AVK
                CALL    BK
                GOTO    MENU
M7              MOVLW   CW_?            ;?AGN AGN K
                CALL    KYT_DEC
                CALL    DASH_SP
                CALL    AGN
                CALL    AGN
                MOVLW   CW_K
                CALL    KYT_DEC
                GOTO    MENU
M8              CALL    NR              ;NR? NR? BK
                CALL    NR
                CALL    BK
                GOTO    MENU
M9              CALL    LOC             ;LOC? LOC? BK
                CALL    LOC
                CALL    BK
                GOTO    MENU
M0              NOP
                GOTO    MENU
ESC             NOP
                GOTO    MENU
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
NUM             BSF     PORTB,2         ;Switch PTT ON without CW keying
NUM1            BSF     PORTB,7         ;Activate (RB7)
                BCF     OPTION_REG, NOT_RBPU
                BTFSC   PORTA,SW4
                GOTO    MENU
                GOTO    NUM1
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
MA              CALL    ReadPoti
                RLF     POTI
                MOVF    POTI,W
                ANDLW   H'F0'
                MOVWF   POTI
                SWAPF   POTI
                INCF    POTI
                MOVLW   D'13'
                ADDWF   POTI
                MOVF    POTI,W
                CALL    KYT_DEC
                MOVF    POTI,W
                MOVWF   POTI2
                ADDWF   POTI2
                ADDWF   POTI2
                GOTO    MENU
MB              NOP
                GOTO    MENU
MC              NOP
                GOTO    MENU
```

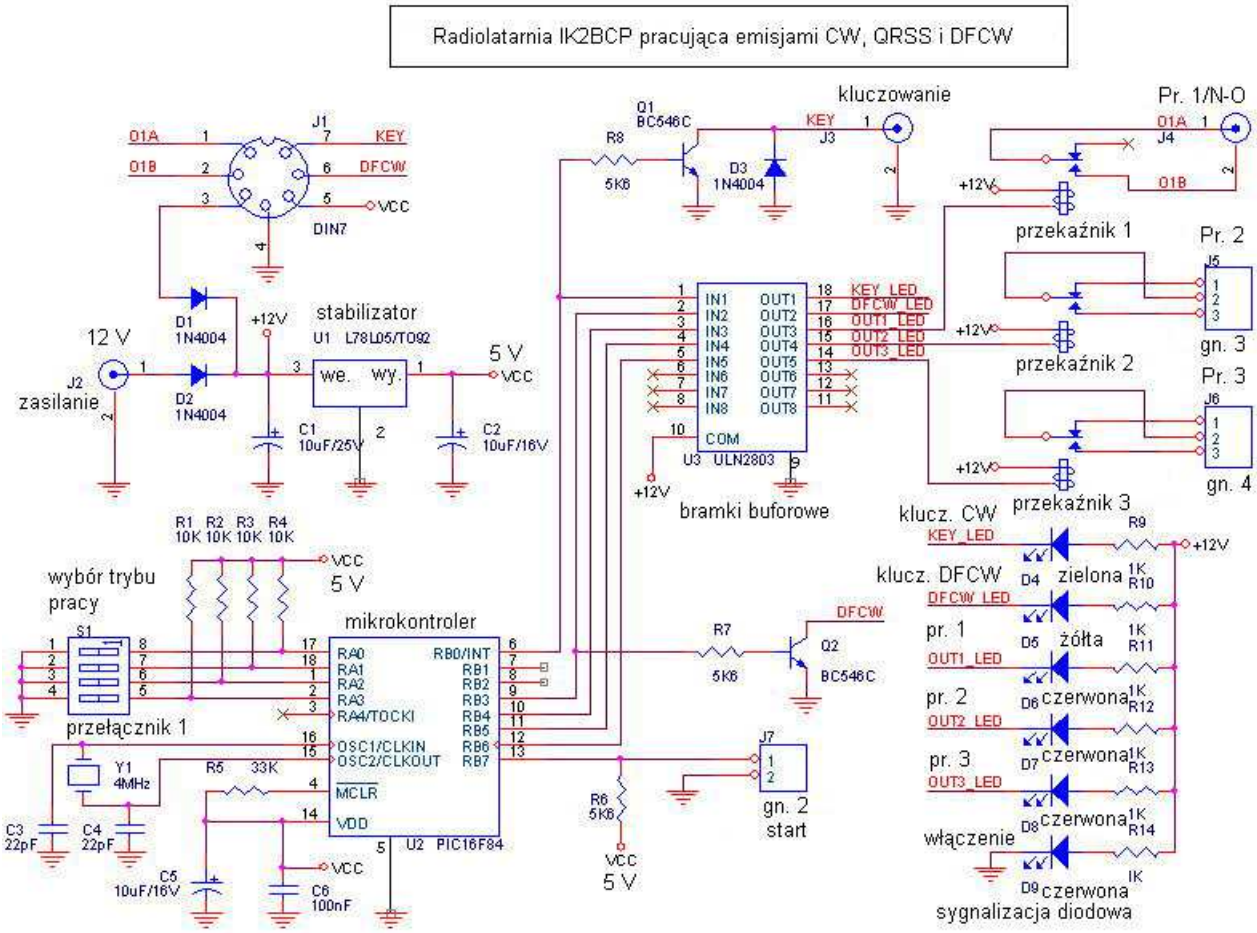
```
MD              NOP
                GOTO    MENU
OM1AVK          MOVLW   CW_O
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_M
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_1
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_A
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_V
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_K
                CALL    KYT_DEC
                CALL    WORD_SP
                RETURN
                ;
CQTEST          MOVLW   CW_C            ;CQ CQ TEST DE
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_Q
                CALL    KYT_DEC
                CALL    WORD_SP
                MOVLW   CW_C
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_Q
                CALL    KYT_DEC
                CALL    WORD_SP
                ;
                MOVLW   CW_T
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_E
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_S
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_T
                CALL    KYT_DEC
                CALL    WORD_SP
                ;
                CALL    DE1
                RETURN
                ;
DE1             MOVLW   CW_D
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_E
                CALL    KYT_DEC
                CALL    WORD_SP
                RETURN
                ;
BK              MOVLW   CW_B
                CALL    KYT_DEC
                CALL    DOT_SP
```

```
                MOVLW   CW_K
                CALL    KYT_DEC
                RETURN
                ;
AGN             MOVLW   CW_A
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_G
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_N
                CALL    KYT_DEC
                CALL    WORD_SP
                RETURN
                ;
NR              MOVLW   CW_N
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_R
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_?
                CALL    KYT_DEC
                CALL    WORD_SP
                RETURN
                ;
LOC             MOVLW   CW_L
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_O
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_C
                CALL    KYT_DEC
                CALL    DASH_SP
                MOVLW   CW_?
                CALL    KYT_DEC
                CALL    WORD_SP
                RETURN
                ;
DOT             MOVF    POTI,W          ;D'30'
                ;MOVLW  D'30'
                MOVWF   TIMER_DOT
D_LOOP          BSF     PORTB,0         ;Activate LD1 (RB0)
                CALL    DELAY_ROUTINE
                BCF     PORTB,0         ;DeActivate LD1 (RB0)
                CALL    DELAY_ROUTINE
                DECFSZ  TIMER_DOT,F
                GOTO    D_LOOP
                RETLW   0
                ;
DOT_SP          MOVF    POTI,W          ;D'30'
                ;MOVLW  D'30'
                MOVWF   TIMER_DOT
S_LOOP          NOP
                CALL    DELAY_ROUTINE
                NOP
                CALL    DELAY_ROUTINE
                DECFSZ  TIMER_DOT,F
                GOTO    S_LOOP
                RETLW   0
                ;
```

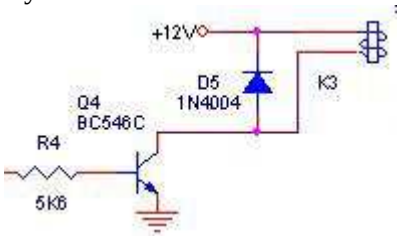
```
DASH            MOVF    POTI2,W                 ;D'90'
                ;MOVLW  D'90'
                MOVWF   TIMER_DOT
D_LOOP2         BSF     PORTB,0         ;Activate LD1 (RB0)
                CALL    DELAY_ROUTINE
                BCF     PORTB,0         ;DeActivate LD1 (RB0)
                CALL    DELAY_ROUTINE
                DECFSZ  TIMER_DOT,F
                GOTO    D_LOOP2
                RETLW   0
                ;
DASH_SP         MOVF    POTI2,W                 ;D'90'
                ;MOVLW  D'90'
                MOVWF   TIMER_DOT
S_LOOP2         ;NOPxxxxxxxxxxxxxxxxxxxxxxxxx ESC Trap
                BSF     PORTB,7         ;Activate (RB7)
                BCF     OPTION_REG, NOT_RBPU
                BTFSC   PORTA,SW2
                GOTO    MENU
                ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                CALL    DELAY_ROUTINE
                CALL    DELAY_ROUTINE
                DECFSZ  TIMER_DOT,F
                GOTO    S_LOOP2
                RETLW   0
                ;
WORD_SP         MOVF    POTI2,W                 ;D'90'
                ;MOVLW  D'90'
                MOVWF   TIMER_DOT
S_LOOP3         ;NOPxxxxxxxxxxxxxxxxxxxxxxxxx ESC Trap
                BSF     PORTB,7         ;Activate (RB7)
                BCF     OPTION_REG, NOT_RBPU
                BTFSC   PORTA,SW2
                GOTO    MENU
                ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                CALL    DELAY_ROUTINE
                NOP
                CALL    DELAY_ROUTINE
                DECFSZ  TIMER_DOT,F
                GOTO    S_LOOP3
                RETLW   0
                END
```

Radiolatarnia CW, QRSS i DFCW na 16F84

Układ i oprogramowanie radiolatarni służącej do pracy emisjami CW, QRSS i DFCW opracował włoski krótkofalowiec IK2BCP. Obwód ULN2803 służy do sterowania przekaźników ale można zastąpić go przez pojedyńcze tranzystory (rys. 4.4).
Program dla mikroprocesora 16F84 i program konfigurujący na PC są do pobrania w internecie pod adresem *http://www.hamlan.org/tech/picbeacon2/picbeacon2eng.htm.*

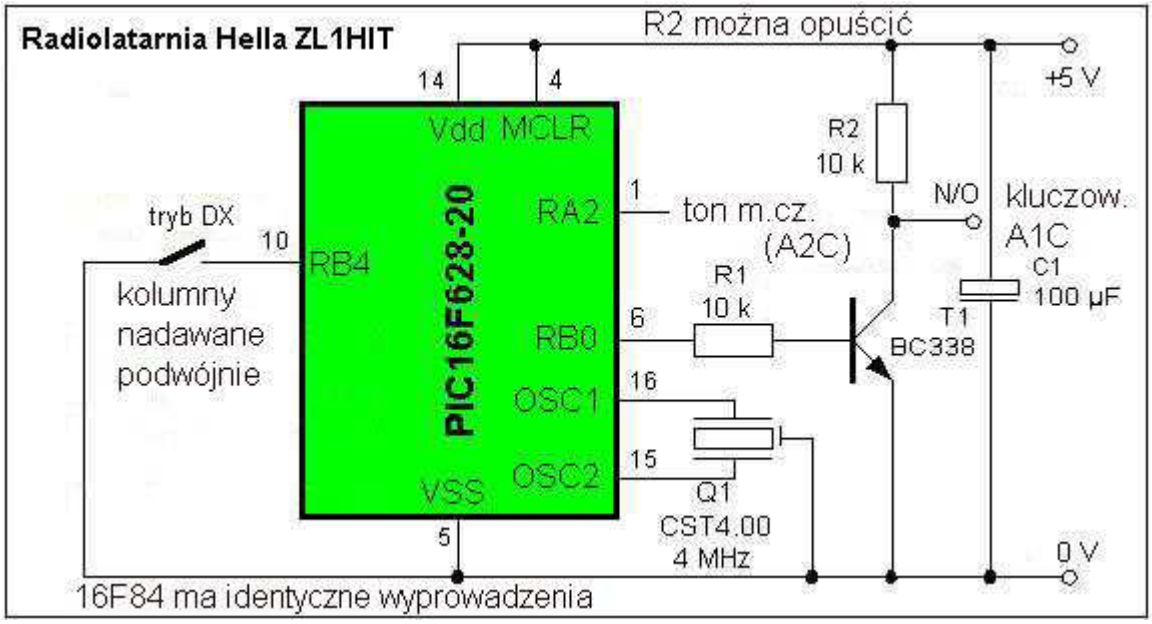


Rys. 4.3. Schemat radiolatarni



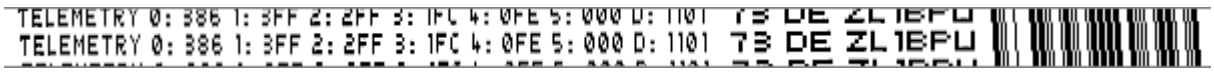
Rys. 4.4. Alternatywny sposób sterowania przekaźników

Radiolatarnia Hella na 16F627/16F628



Rys. 4.5. Schemat radiolatarni

Schemat układu jest identyczny jak dla rozwiązania wzorcowego ZL1HIT (poza zmienionym typem mikroprocesora) a samo opracowanie stanowi modyfikację dokonaną przez OE1KDA w celu dostosowania programu wzorcowego do pracy na nowocześniejszych mikroprocesorach typu 16F627(A) i 16F628(A). Oba mikroprocesory różnią się tylko pojemnością pamięci programu, ale jest on na tyle krótki, że mieści się w pamięci 16F627(A).
Program oryginalny, napisany w asemblerze, pracował na procesorze 16C84 z kwarcem o czestotliwości 7328 kHz. Dostarczał on sygnału kluczującego nadajnik telegraficzny na wyjściu A0, i kluczowanego sygnału akustycznego (fali prostokątnej) na wyjściu A2. Zwarcie wejścia B4 do masy powoduje przejście do transmisji w trybie DX-owym, w którym kolumny dla zwiększenia czytelności są nadawane podwójnie (metoda ta jest stosowana w wielu programach). W pamięci zawarte są dwa niezależne teksty przeznaczone odpowiednio do transmisji w systemie *Feldhella* i telegrafią. Uproszczona czcionka opracowana przez ZL1BPU ma rozdzielczość 5 x 5 elementów co po dodaniu obrzeża daje znaki o rozdzielczości 7 x 7 elementów.
Program został dostosowany do częstotliwości kwarcu 4 MHz a wszystkie zmiany i uzupełnienia dokonane przez OE1KDA są komentowane po polsku.
W układzie można zastosować rezonator ceramiczny tak jak to pokazano na schemacie lub kwarcowy. Pomiędzy nóżkami 15 i 16 a masą należy w tym drugim przypadku włączyć kondensatory o pojemności 15 – 33 pF. Sygnał wyjściowy z kolektora tranzystora jest doprowadzony do gniazda klucza telegraficznego w nadajniku.
W zakresie długofalowym stosowana jest często o połowę mniejsza szybkość transmisji co wymaga w najprostszym przypadku jedynie zmiany kwarcu. Dla jeszcze mniejszych szybkości transmisji (1/4, 1/8 lub Slowfeld – 1/100) konieczna jest – stosunkowo nieskomplikowana – modyfikacja programu.



Rys. 4.6. Komunikat w systemie Hella i telegraficzny (pionowe kreski po prawej stronie) na ekranie komputera.

```
;*****************************************************************
; Feld Hellschreiber / CW Beacon Base Band Beacon
; Author: Bryan J. Rentoul
```

```
; Date: 10-April-1999
;
; All rights reserved
;
; Description: This project was inspired, yeh "specced" even, by
Murray,
; ZL1BPU. It purpose is to send a mixture of CW and Hellschreiber via
; a simple CW transmitter based around a 3.58 MHz Xtal oscillator in
the
; classic style. This is an early version and lacks any "user
friendly"
; interface for loading new messages etc.
;
; This version uses the 5x5 pixel all upper case font set provided
; by ZL1BPU. (Letters 'C', 'V' an 'O' slighly modified for clarity.
;
;
;*****************************************************************
; Program dopasowany do mikrokontrolerow 16F627(A), 16F628(A) i
czestotliwosci
; zegarowej 4 MHz przez Krzysztofa Dabrowskiego OE1KDA
; Zmiany komentowane w jezyku polskim
;*****************************************************************
; PA0 - kluczowanie A1 nadajnika
; PA2 - ton m.cz. (do transmisji A2)
; PB4 - wejscie wyboru trybu (masa - tryb DX, podwojna transmisja
kolumn)
;*****************************************************************
        include "p16f627.inc"
    errorlevel-302
;*** SET CONFIG FUSES ***

;   __CONFIG _CP_OFF & _WDT_OFF & _HS_OSC
    __CONFIG _PWRTE_ON & _WDT_OFF & _XT_OSC & _BODEN_OFF & _LVP_OFF

; czestotliwosc zegarowa (wybor alternatywny)
#define KWARC4
;#define KWARC7

; poczatkowy stan licznika TMR0 dla 7,3728 MHz (zachowany z
oryginalu)
#ifdef KWARC7
#define POCZ_LICZ 18
#endif

; poczatkowy stan licznika TMR0 dla 4 MHz
; po wstepnym obliczeniu (proporcja)
; wartosc dobrana eksperymentalnie dla uzyskania
; wlasciwej szybkosci transmisji (w miare poziomego polozenia tekstu
; po stronie odbiorczej)
#ifdef KWARC4
#define POCZ_LICZ 133
#endif

;For Assembler PORTAbility
    radix dec
```

```
W       EQU     0               ;For file,W
w       EQU     0               ;For file,W
F       EQU     1               ;For file,F
f       EQU     1               ;For file,F

;*****************************************************************
;* REGISTER DECLARATIONS
;*****************************************************************
ind     equ     0               ;0=pseudo-reg 0 for indirect (FSR)
RTCC    equ     1               ;1=Real Time Clock/Counter
TMR0     equ      1             ;1=Timer0 (same as above)
PC      equ     2               ;2=PC
STATUS  equ     3               ;3=Status Reg
INTCON  equ       0Bh
PCLATH  equ       0Ah

;* Definicje bitow w Status reg
#define B_C STATUS,0    ;Carry
#define B_DC STATUS,1       ;Half carry
#define B_Z STATUS,2    ;Zero
#define B_PD STATUS,3       ;Power down
#define B_TO STATUS,4       ;Timeout
#define B_PA0 STATUS,5      ;Page select (56/57 only)
#define B_PA1 STATUS,6      ;Page select (56/57 only)
#define B_PA2 STATUS,7      ;GP flag
;----
;* Definicje bitow w OPTION reg
#define B_RBPU OPTION_REG,7
;----
;* Definicje bitow w INTCON reg (podane w p16f627.inc)
#define I_GIE INTCON,7
#define I_PEIE INTCON,6
#define I_T0IE INTCON,5
#define I_INTE INTCON,4
#define I_RBIE INTCON,3
#define I_T0IF INTCON,2
#define I_INTF INTCON,1
#define I_RBIF INTCON,0

;-----------------------------------------------------------------

FSR     equ     4             ;4=file select reg 0-4=indirect address
fsr     equ     4             ;4=file select reg 0-4=indirect address
PORTA   equ     5             ;5=port A I/O register (4 bits)
PORTB   equ     6             ;6=port B I/O register
porta   equ     5             ;5=port A I/O register (4 bits)
portb   equ     6             ;6=port B I/O register

;-----------------------------------------------------------------
#define RA0 PORTA,0
#define RA1 PORTA,1
#define RA2 PORTA,2
#define RA3 PORTA,3
#define RA4 PORTA,4
#define RB0 PORTB,0
#define RB1 PORTB,1
```

```
#define RB2 PORTB,2
#define RB3 PORTB,3
#define RB4 PORTB,4
#define RB5 PORTB,5
#define RB6 PORTB,6
#define RB7 PORTB,7
#define PTT PORTA,0     ; PTT Output
#define SPAREA1 PORTA,1
#define SIDETONE PORTA,2
#define TIMECAP PORTA,3     ; Timer capacitor output
#define TIMEIN PORTA,4      ; Timer cap. input
#define TXOUT PORTB,0
#define FSK0 PORTB,1
#define FSK1 PORTB,2
#define DXMODE PORTB,4      ; DX Mode input. Jumper to GND for DX mode
enable

;-----------------------------------------------------------------
SAVEW  equ 2Ch ; Used in INT routine to save W and STATUS register
SAVES  equ 2Dh
FLAGS  equ 2Eh ; Inputs status register
DL1 equ 2Fh ; Delay routine counters
DL2 equ 30h ;    "    "      "
DL3 equ 31h ;    "    "      "
INTCNT equ 32h ; Interrupt call counter to count 64 calls then reset
TONECNT equ   33h ; Tone counter using to toggle output every 8 int
calls
COLCNT equ 34h ; Column counter
ROWCNT equ 35h ; Row counter
CURCOL equ 36h ; Current column data for rotating each bit into carry
flag
CHAR   equ 37h ; Character code to send (ASCII values from 32 to 95
OK)
TEMP   equ 38h
TEMP1  equ 39h
TIMER  equ 40h ; We use this as a free running timer incremented
          ; by the Int routine. It is for CW dot / dash timing etc
SAVE1  equ 41h ; Just another TEMP variable
SAVE2  equ 42h ;
OUTBUF equ 43h ; CW Output buffer

; 20h to 2B  free

    ; Define some status bits (flags)
#define BUSY FLAGS,0       ; 0 = Buffer awaits new character to send
(1 = busy)
#define F_SIDETONE FLAGS,1
#define F_TABLE FLAGS,2     ; 0 = Font Table 1, 1 = Font Table 2
#define F_CW FLAGS,3    ; In CW mode (1) the Int does not toggle
TXOUT
#define CLB bcf
#define SEB bsf
#define SKBC btfsc
#define SKBS btfss
;*** Reset Vector
*****************************************************
```

```
   ORG     0000
reset
   GOTO    INIT

; ========================
; == INTERRUPT PROCEDURE ==
; ========================
   ORG 0004
TIMERINT   ; Interrupt routine called (17.5 * 5 * 60) times per
second providing
      ; pixel timing (5 pixels/rows  per column)

   ; Save W and Status
     CLB      I_T0IE    ; Disable this interrupt
     CLB      I_T0IF    ; clear the interrupt
        movwf   SAVEW       ; save w reg in Buffer
        swapf   SAVEW, f; swap it
        swapf   STATUS,w; get status (swapped)
        movwf   SAVES       ; and save it

   movlw  POCZ_LICZ       ; Reset Counter to POCZ_LICZ (instead of
around zero or so)
   movwf  TMR0            ;   /

   ; Interrupt procedure propper starts here

   decfsz TONECNT, f ; Time for side tone click?
   goto   INEXT1

   movlw  4            ; Toggle the Sidetone output every 6 calls
   movwf  TONECNT       ;   /
   btfss  F_SIDETONE    ; ...but only if the F_SIDETONE flag is set
   goto   INEXT1
   btfsc SIDETONE
   goto   $+3
   bsf SIDETONE
   goto   $+2
   bcf SIDETONE


INEXT1
   decfsz INTCNT, f  ; Time to send a dot pixel?
   goto   IEND            ; Nope...

   ;-------
   decf   TIMER, f        ; decrement out free runing timer used in
DELAY function
   ;-------

   movlw  60           ; Yes...
   movwf  INTCNT    ; Reset INTCNT to 60

   ; Check the BUSY FLAG. If zero do nothing
   btfss  BUSY
   goto   IEND
```

```
    movfw  ROWCNT
    btfsc  B_Z       ; If ROWCNT is zero then we are done
    goto   IENDCOL

    ; Other wise send next bit

    movfw  CURCOL
    rrf CURCOL, f       ; Rotate bit into carry flag for testing
    btfsc  B_C
    goto   ITONEON

    btfsc  F_CW      ; Dont touch TXOUT or F_SIDETONE if in CW mode
    goto   INEXT2
    CLB TXOUT           ; Turn TX carrier OFF
    CLB F_SIDETONE      ; flag side tone to OFF
    goto   INEXT2

ITONEON
    btfsc  F_CW      ; Dont touch TXOUT or F_SIDETONE if in CW mode
    goto   INEXT2
    SEB TXOUT           ; Turn TX carrier ON
    SEB F_SIDETONE      ; flag side tone to ON

INEXT2
    decf   ROWCNT, f       ; Decrement row counter
    goto   IEND            ; Exit the int routine

IENDCOL
    CLB BUSY            ; Clear the BUSY flag to indicate column sent

IEND
        swapf   SAVES,w     ; get status (swapped back)
        movwf   STATUS      ; and restore it
        swapf   SAVEW,w     ; restore W reg (swapped back to restore
Z,C,DC flags)
    SEB I_T0IE    ; Re-enable the interrupt
    retfie

;---- END OF INTERRUPT ROUTINE ------------------------------------
;------------------------------------------------------------------

;*******************************************************************
;** INIT
;** Hardware reset entry point
;**
;*******************************************************************

INIT    ;Power-on entry

;*******************************************************************
;** RESET
;** software reset entry point
;**
;*******************************************************************
RESET   ;Soft reset
```

```
        ;Init ports
    movlw   10h             ; All Port A bits to outputs except RA4
(Timer Cap input)
    banksel   TRISA
    movwf   TRISA

    movlw  0F0h             ; Initialize Port B I/O pins
    banksel   PORTB
    movwf   PORTB
    movlw  0F0h             ; Port B<0-3> Output, Port B<4-7> Input
    banksel   TRISB
    movwf    TRISB

    movlw  b'00011001'       ; 0 Turn ON Port B pullups
                        ; 0 External Int triggers on falling edge
                        ; 0 TOCS low (timer run from internal clock)
                        ; 1 TOSE - Counter timer/counter triggers on
rising edge
                        ; 1 Prescaler switched OUT (to WDT) ...
                        ; 110  ...with div. 64 (WDT) or 128 (TMR0)
    banksel   OPTION_REG

    movwf  OPTION_REG        ; Store W in the Option register.
    banksel CMCON    ; wylaczenie komparatorow
    bsf CMCON, 0
    bsf CMCON, 1
    bsf CMCON, 2
    banksel FLAGS
    CLB BUSY             ; Clear busy flag
    CLB F_SIDETONE
    CLB F_CW             ; Start in Hell' mode

    clrf   COLCNT
    clrf   ROWCNT

    ; Set up the timer interrupt to give us the required 122.5 dots
per second
    ; (17.5 columns per second with 7 pixel rows)
    ; We reset the counter to 18 each time to yield 240 counts per
    ; interrupt = (7.3728 / 4 / 240) = 7680 ints. per second.
    ; 7680 / (17.5 x 7 row pixels) = 62.69. So every 63rd call to the
int. should
    ; send one pixel.
    ;
    ; NOTE: In practice it was found tat every 60th call produced the
right timing!

    movlw  6
    movwf  TONECNT         ; Init tone counter reg. Toggle TXOUT
every 6 calls to int.
    movlw  60
    movwf  INTCNT       ; Initialize the INTCNT counter reg.
    SEB I_T0IE       ; enable timer overflow interrupt
    SEB I_GIE              ; global interrupt enable
                        ; The TIMERINT interrupt routine is now running
```

```
   goto   MAIN_LOOP

;=====================================================================

DELAY  ; Arbitary delay routine for CW dash/dot timing or whatever

   ; Make a delay send one blank column
   clrf   CURCOL
   call   TXCOL
   return
;---------------------------------------------------------------------

;=====================================================================
; Subroutines for various things shall go here..
;=====================================================================

TXCOL  ; Sets the Int routine up to send a column - 5 rows
   ; (bits stretched over usual 14 rows)
   ; The CURCOL variable hold the column dat to send

   movfw  CURCOL     ; Save column data (the Int routine destroys it)
   movwf  SAVE1

   movlw  6
   movwf  ROWCNT     ; ROWCNT will process 5 rows (0 not processed)
   SEB BUSY            ; Setting this bit signals Int routine to send
a column
   btfsc  BUSY          ; sit in loop waiting for BUSY to clear
   goto   $-1

   movfw  SAVE1           ; Restore column data (in case we want to
send it again)
   movwf  CURCOL

   return
;---------------------------------------------------------------------

TXCHAR ; Send character in "W" using Hellscheiber
   movwf  TEMP
   movlw  32           ; make the char code zero based
   subwf  TEMP,f     ; and store in TEMP

   ; The FONT table is split into two parts. Characters 0 to 31 are
in part 1
   ; The rest are in part 2 (FONTTAB2)

   movlw  32           ; Is the character to send less than 32?
   subwf  TEMP, w       ; Subtract W (32) from TEMP
   btfsc  B_C        ; Carry Flag is SET for a positive result (TEMP
> 32)
   goto   USETABLE2
USETABLE1
   movlw  HIGH FONTTAB1 ; Get the jump page right and stick
   movwf  PCLATH     ; it in PCLATH as per the book :)
```

```
   CLB F_TABLE       ; Clear the F_TABLE flag to indicate FONTTAB1
below
   goto   SENDCHAR        ; --> go lookup and send this character

USETABLE2
   movlw  32           ; Subtract addition 32 to correct char offset
for table 2
   subwf  TEMP, f       ;   /
   movlw  HIGH  FONTTAB2   ; Load PCLATH to correct memory page is
used
   movwf  PCLATH     ;   /
   SEB F_TABLE       ; Set the F_TABLE flag to indicate FONTTAB2
below

SENDCHAR
   movfw  TEMP            ; Retrive char offset
   addwf  TEMP, f       ; multiply by 5 to get to start of
   addwf  TEMP, f       ; char in font table
   addwf  TEMP, f       ;   /
   addwf  TEMP, f       ;   /

   clrf   COLCNT     ; Reset COLCNT (column counter)
TAB1LOOP
   movfw  TEMP            ; Retrieve char font table offset
   addwf  COLCNT, w       ; Add the column number
   btfss  F_TABLE       ; Call the right table based on F_TABLE flag
...
   call   FONTTAB1
   btfsc  F_TABLE       ; ...set above to get the column data
   call   FONTTAB2
   movwf  CURCOL

   call   TXCOL       ; Flag the Interrupt routine to send this
column
   SKBS   DXMODE     ; If no jumper on for DX mode then exit
   call   TXCOL           ; otherwise send the column a 2nd time

   incf   COLCNT, f       ; Increment COLCNT
   movfw  COLCNT     ; Last column (5) ?
   sublw  5
   btfss  B_Z        ; If not then send next column
   goto   TAB1LOOP

   clrf   CURCOL     ; Send one blank column
   call   TXCOL           ;    /

   return

;-------------------------------------------------------------------
;--- CW ROUTINES START HERE ----------------------------------------
;-------------------------------------------------------------------
SENDDASH
   SEB F_SIDETONE
   SEB TXOUT           ; PTT on
   call   DELAY           ; Dash delay
   call   DELAY
```

```
    call   DELAY
    call   DELAY
    call   DELAY
    CLB TXOUT           ; PTT off
    CLB F_SIDETONE
    call   DELAY
    return
;-------------------------------------------------------------------
SENDDOT
    SEB F_SIDETONE
    SEB TXOUT           ; PTT on
    call   DELAY           ; Dot delay
    call   DELAY
    CLB TXOUT           ; PTT off
    CLB F_SIDETONE
    call   DELAY           ; "space between" delay
    return
;-------------------------------------------------------------------

SENDCW ; Sends CW character held in W. RRF's each bit out until W = 1
    ; 00h in W means send space between words

    SEB F_CW            ; Tell the Int routine we are in CW mode

    movwf  OUTBUF
    xorlw  0            ; Trigger Z flag (maybe)
    btfsc  B_Z          ; Its zero so just send a space
    goto   SSPACE
SILOOP
    movfw  OUTBUF
    sublw  1
    btfsc  B_Z          ; If just one then we are done
    goto   SIEND

    CLB B_C
    rrf OUTBUF, f       ; Get next bit to send

    btfss  B_C          ; Branch to DOT if Carry clear. Otherwise its
a DASH
    goto   SDOT
    call   SENDDASH
    goto   SILOOP    ; loop back
SDOT
    call   SENDDOT
    goto   SILOOP    ; loop back

SSPACE
    call   DELAY             ; Send word space (just wait)
    call   DELAY
    call   DELAY
SIEND
    call   DELAY             ; Inter char delay
    call   DELAY
    call   DELAY

    CLB F_CW            ; Tat's enough CW for now thanks :)
```

```
    return
;-----------------------------------------------------------------

;*****************************************************************
;****** M A I N   L O O P  ***************************************
;*****************************************************************

MAIN_LOOP

    ; Send the beacon string from the "T_BEACON" table
    clrf   TEMP1
BEACONL1
    movlw  HIGH  T_BEACON   ; Stick the MSByte of T_BEACON in PCLATH
    movwf  PCLATH
    movfw  TEMP1            ; Retrieve the beacon character offset
    call   T_BEACON         ; Go get the character
    xorlw  0            ; To update the Z flag
    btfsc  B_Z          ; Zero? If so we are done so jump out to
MLEND1
    goto   MLEND1
    call   TXCHAR     ; Otherwise send this character
    incf   TEMP1, f         ; Increment to next character
    goto   BEACONL1         ; Loopy de loop

MLEND1
    call   DELAY
    call   DELAY

;-----------------------------------------------------------------

    ; Same again but this time CW
    clrf   TEMP1
BEACONL2
    movlw  HIGH  T_CALLSIGN ; Stick the MSByte of T_BEACON in PCLATH
    movwf  PCLATH
    movfw  TEMP1            ; Retrieve the beacon character offset
    call   T_CALLSIGN       ; Go get the character
    xorlw  0            ; To update the Z flag
    btfsc  B_Z          ; Zero? If so we are done so jump out to
MLEND1
    goto   MLEND2
    movwf  TEMP             ; Save W
    movlw  HIGH CWT         ; Get the dot and dashes from the CWT
    movwf  PCLATH     ; /
    movfw  TEMP             ; Restore W
    call   CWT          ; /
    call   SENDCW     ; Send this character
    incf   TEMP1, f         ; Increment to next character
    goto   BEACONL2         ; Loopy de loop

MLEND2
    call   DELAY
    call   DELAY

    goto   MAIN_LOOP        ; Luckily PIC MCU's do not get bored!
;-----------------------------------------------------------------
```

```
    ORG 0200h

FONTTAB1
    addwf  PC, f

    retlw   0  ;  32 -
    retlw   0
    retlw   0
    retlw   0
    retlw   0

    retlw   0  ;  33 - !
    retlw   0
    retlw   29
    retlw   0
    retlw   0

    retlw   0  ;  34 - "
    retlw   24
    retlw   0
    retlw   24
    retlw   0

    retlw   10 ;  35 - #
    retlw   31
    retlw   10
    retlw   31
    retlw   10

    retlw   8  ;  36 - $
    retlw   21
    retlw   31
    retlw   21
    retlw   2

    retlw   25 ;  37 - %
    retlw   26
    retlw   4
    retlw   11
    retlw   19

    retlw   10 ;  38 - &
    retlw   21
    retlw   13
    retlw   2
    retlw   5

    retlw   0  ;  39 - '
    retlw   16
    retlw   24
    retlw   0
    retlw   0

    retlw   0  ;  40 - (
    retlw   14
```

```
    retlw   17
    retlw   0
    retlw   0

    retlw   0 ;  41 - )
    retlw   0
    retlw   17
    retlw   14
    retlw   0

    retlw   21;  42 - *
    retlw   14
    retlw   31
    retlw   14
    retlw   21

    retlw   4 ;  43 - +
    retlw   4
    retlw   31
    retlw   4
    retlw   4

    retlw   0 ;  44 - ,
    retlw   1
    retlw   6
    retlw   0
    retlw   0

    retlw   4 ;  45 - -
    retlw   4
    retlw   4
    retlw   4
    retlw   4

    retlw   0 ;  46 - .
    retlw   3
    retlw   3
    retlw   0
    retlw   0

    retlw   1 ;  47 - /
    retlw   2
    retlw   4
    retlw   8
    retlw   16

    retlw   14;  48 - 0
    retlw   19
    retlw   21
    retlw   25
    retlw   14

    retlw   0 ;  49 - 1
    retlw   16
    retlw   31
    retlw   0
```

```
    retlw   0

    retlw   3  ;  50 - 2
    retlw   21
    retlw   21
    retlw   21
    retlw   9

    retlw   0  ;  51 - 3
    retlw   21
    retlw   21
    retlw   21
    retlw   10

    retlw   0  ;  52 - 4
    retlw   30
    retlw   2
    retlw   7
    retlw   2

    retlw   28 ;  53 - 5
    retlw   21
    retlw   21
    retlw   21
    retlw   2

    retlw   14 ;  54 - 6
    retlw   21
    retlw   21
    retlw   21
    retlw   2

    retlw   16 ;  55 - 7
    retlw   16
    retlw   23
    retlw   24
    retlw   0

    retlw   10 ;  56 - 8
    retlw   21
    retlw   21
    retlw   21
    retlw   10

    retlw   8  ;  57 - 9
    retlw   21
    retlw   21
    retlw   21
    retlw   14

    retlw   0  ;  58 - :
    retlw   5
    retlw   5
    retlw   0
    retlw   0
```

```
    retlw   0 ;  59 - ;
    retlw   10
    retlw   11
    retlw   0
    retlw   0

    retlw   4 ;  60 - <
    retlw   10
    retlw   17
    retlw   0
    retlw   0

    retlw   0 ;  61 - =
    retlw   10
    retlw   10
    retlw   10
    retlw   0

    retlw   0 ;  62 - >
    retlw   0
    retlw   17
    retlw   10
    retlw   4

    retlw   8 ;  63 - ?
    retlw   16
    retlw   21
    retlw   20
    retlw   8
;-----------------------------------------------------------------

CWT ; CW Table Call with W=ASC of char to send (EG. 'A' (65) for
"di'dah" )
    addlw  0
    SKBC   B_Z       ; Return a zero if zero received
    retlw  0

    movwf  SAVE1      ; Calculate offset in table from ASCI char given
    movlw  48     ; [ ASCII code for "0" ]
    subwf  SAVE1, w
    addwf  PC, f

    retlw  b'00111111' ; 0
    retlw  b'00111110' ; 1
    retlw  b'00111100' ; 2
    retlw  b'00111000' ; 3
    retlw  b'00110000' ; 4
    retlw  b'00100000' ; 5
    retlw  b'00100001' ; 6
    retlw  b'00100011' ; 7
    retlw  b'00100111' ; 8
    retlw  b'00101111' ; 9
    retlw  b'00000000' ; :
    retlw  b'00000000' ; ;
    retlw  b'00000000' ; <
    retlw  b'00000000' ; =
```

```
   retlw  b'00000000' ; >
   retlw  b'00000000' ; ?
   retlw  b'00000000' ; @
   retlw  b'00000110' ; A
   retlw  b'00010001' ; B
   retlw  b'00010101' ; C
   retlw  b'00001001' ; D
   retlw  b'00000010' ; E
   retlw  b'00010100' ; F
   retlw  b'00001011' ; G
   retlw  b'00010000' ; H
   retlw  b'00000100' ; I
   retlw  b'00011110' ; J
   retlw  b'00001101' ; K
   retlw  b'00010010' ; L
   retlw  b'00000111' ; M
   retlw  b'00000101' ; N
   retlw  b'00001111' ; O
   retlw  b'00010110' ; P
   retlw  b'00011011' ; Q
   retlw  b'00001010' ; R
   retlw  b'00001000' ; S
   retlw  b'00000011' ; T
   retlw  b'00001100' ; U
   retlw  b'00011000' ; V
   retlw  b'00001110' ; W
   retlw  b'00011001' ; X
   retlw  b'00011101' ; Y
   retlw  b'00010011' ; Z

;********************************************************************

   ; FONT TABLE 2 (second half)

   ORG 0300h        ; start table at even multiple of 256 address

FONTTAB2
   addwf  PC, f

   retlw   14 ;  64 - @
   retlw   17
   retlw   21
   retlw   21
   retlw   8

   retlw   15 ;  65 - A
   retlw   20
   retlw   20
   retlw   20
   retlw   15

   retlw   31 ;  66 - B
   retlw   21
   retlw   21
   retlw   21
   retlw   10
```

```
    retlw   14 ;  67 - C
    retlw   17
    retlw   17
    retlw   17
    retlw   0  ;was 10

    retlw   31 ;  68 - D
    retlw   17
    retlw   17
    retlw   17
    retlw   14

    retlw   31 ;  69 - E
    retlw   21
    retlw   21
    retlw   21
    retlw   17

    retlw   31 ;  70 - F
    retlw   20
    retlw   20
    retlw   20
    retlw   16

    retlw   14 ;  71 - G
    retlw   17
    retlw   17
    retlw   19
    retlw   10

    retlw   31 ;  72 - H
    retlw   4
    retlw   4
    retlw   4
    retlw   31

    retlw   0  ;  73 - I
    retlw   17
    retlw   31
    retlw   17
    retlw   0

    retlw   2  ;  74 - J
    retlw   17
    retlw   31
    retlw   16
    retlw   0

    retlw   31 ;  75 - K
    retlw   4
    retlw   10
    retlw   17
    retlw   0

    retlw   31 ;  76 - L
```

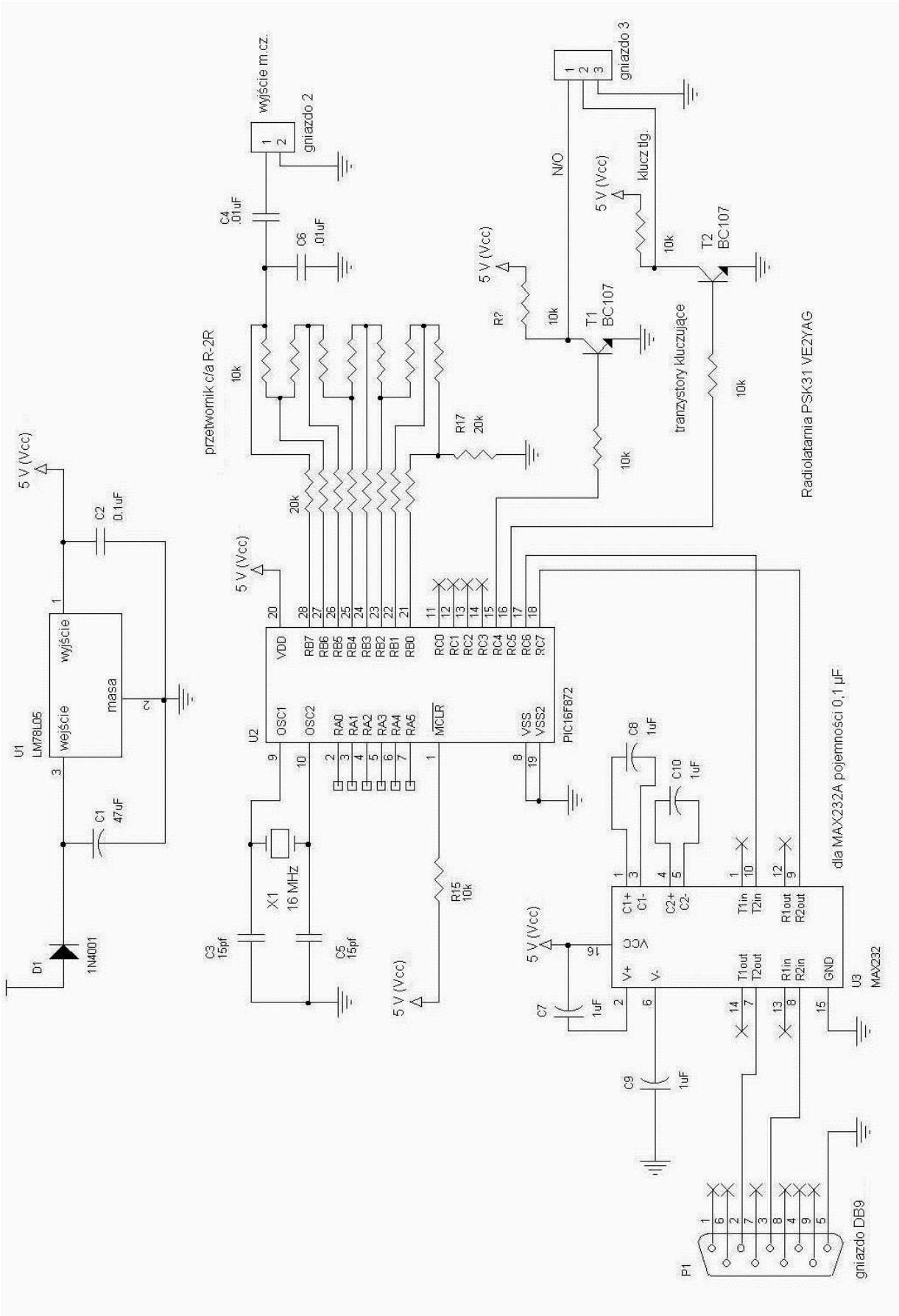
```
    retlw   1
    retlw   1
    retlw   1
    retlw   0

    retlw   31 ;  77 - M
    retlw   8
    retlw   4
    retlw   8
    retlw   31

    retlw   31 ;  78 - N
    retlw   8
    retlw   4
    retlw   2
    retlw   31

;   retlw   31 ;  79 - O
;   retlw   17
;   retlw   17
;   retlw   17
;   retlw   31

    retlw   14 ;  79 - O
    retlw   17
    retlw   17
    retlw   17
    retlw   14

    retlw   31 ;  80 - P
    retlw   20
    retlw   20
    retlw   20
    retlw   8

    retlw   31 ;  81 - Q
    retlw   17
    retlw   21
    retlw   19
    retlw   15

    retlw   31 ;  82 - R
    retlw   20
    retlw   22
    retlw   21
    retlw   9

    retlw   8  ;  83 - S
    retlw   21
    retlw   21
    retlw   21
    retlw   2

    retlw   16 ;  84 - T
    retlw   16
    retlw   31
```

```
    retlw   16
    retlw   16

    retlw   31 ;  85 - U
    retlw   1
    retlw   1
    retlw   1
    retlw   31

;   retlw   16 ;  86 - V
;   retlw   8
;   retlw   4
;   retlw   2
;   retlw   31

    retlw   24 ;  86 - V
    retlw   6
    retlw   1
    retlw   6
    retlw   24

    retlw   30 ;  87 - W
    retlw   1
    retlw   6
    retlw   1
    retlw   30

    retlw   17 ;  88 - X
    retlw   10
    retlw   4
    retlw   10
    retlw   17

    retlw   16 ;  89 - Y
    retlw   8
    retlw   7
    retlw   8
    retlw   16

    retlw   17 ;  90 - Z
    retlw   19
    retlw   21
    retlw   25
    retlw   17

    retlw   31 ;  91 - [
    retlw   17
    retlw   17
    retlw   0
    retlw   0

    retlw   16 ;  92 - \
    retlw   8
    retlw   4
    retlw   2
    retlw   1
```

```
    retlw   17 ;  93 - ]
    retlw   17
    retlw   31
    retlw   0
    retlw   0

    retlw   0  ;  94 - ^
    retlw   8
    retlw   16
    retlw   8
    retlw   0

    retlw   31 ;  95 - _
    retlw   31
    retlw   31
    retlw   31
    retlw   31

T_BEACON
    addwf  PC, f
    retlw  ' '
    retlw  '>'
    retlw  '>'
    retlw  ' '
    retlw  'O'
    retlw  'E'
    retlw  '1'
    retlw  'K'
    retlw  'D'
    retlw  'A'
    retlw  ' '
    retlw  'J'
    retlw  'N'
    retlw  '8'
    retlw  '8'
    retlw  'E'
    retlw  'D'
    retlw  ' '
    retlw  '3'
    retlw  '0'
    retlw  ' '
    retlw  'D'
    retlw  'B'
    retlw  'M'
    retlw  ' '
    retlw  'P'
    retlw  'S'
    retlw  'E'
    retlw  ' '
    retlw  'Q'
    retlw  'S'
    retlw  'L'
    retlw  ':'
    retlw  ' '
    retlw  'O'
```

```
    retlw  'E'
    retlw  '1'
    retlw  'K'
    retlw  'D'
    retlw  'A'
    retlw  '@'
    retlw  'O'
    retlw  'E'
    retlw  'V'
    retlw  'S'
    retlw  'V'
    retlw  '.'
    retlw  'A'
    retlw  'T'
    retlw  ' '
    retlw  ' '
    retlw  ' '
    retlw  ' '
    retlw  '<'
    retlw  '<'
    retlw  ' '
    retlw  ' '
    retlw  0

T_CALLSIGN
    addwf  PC, f
    retlw  'O'
    retlw  'E'
    retlw  '1'
    retlw  'K'
    retlw  'D'
    retlw  'A'
    retlw  0

    END
```

Radiolatarnia PSK31 na 16F872



Rys. 4.7. Schemat ideowy radiolatarni PSK31

Opracowane przez VE2YAG (*members.tripod.com/ve2yag/index.htm*) rozwiązanie (rys. 4.7) jest oparte na mikrokontrolerze PIC 16F872 – zamiast niego można zastosować 16F873, 16F876 lub inny z serii 16F8xx; różnią się one jedynie wielkością pamięci przy identycznych wyprowadzeniach – i pozwala na wprowadzanie nadawanego tekstu oraz odstępu czasu między transmisjami (0–255 sek.) za pomocą złącza szeregowego RS-232. Przetwornik cyfrowo-analogowy składający się z drabinkowo połączonych oporników R-2R dostarcza sygnału m.cz. o częstotliwości 1 kHz modulującego nadajnik SSB. Dla poprawienia tłumienia drugiej harmonicznej można na jego wyjściu dodać (nie pokazany na schemacie) aktywny filtr dolnoprzepustowy lub zaporowy.
Radiolatarnia może także pracować w trybie telegraficznym (naprzemian z PSK31 lub wyłącznie) dlatego też oprócz wyjścia służącego do włączania nadajnika posiada też wyjście kluczujące CW. Kluczowanie odbywa się za pomocą tranzystorów wykonawczych npn małej mocy dowolnego typu np. BC107.
Obwód scalony MAX232 służy do dopasowania poziomów sygnałów TTL do standardu RS-232 (+/- 12 V). Transmisja w złączu szeregowym odbywa się z szybkością 9600 bitów/sek. z parametrami 8N1. Do wprowadzania tekstów może służyć program terminalowy na komputerze PC (np. *Hyperterminal* dla Windows albo opracowany przez VE2YAG *psk31config.exe* dostępny w pliku *winpsksrc.zip*) lub dowolne automatyczne urządzenie pomiarowe po odpowiednim dopasowaniu komunikatów.
Program jest od dłuższego czasu w użyciu u OE1KDA.

```
//-------------------------------------------------------------------
// PSK31.C
//
// PSK31 beacon (PIC16F872 @ 16Mhz)
// 16Mhz give me exact timing, I have tried 20Mhz crystal but timer
resolution
// is not enough to have precise baudrate. (31.25baud is very
critical)
//
// Created by : Remi Bilodeau VE2YAG 31 mars 2002
// Last Modification : 11 avril 2002 Version 1.1 built 01
//
// CPU power is not enough to generate PSK31 modulation. To correct
that I have
// digitalized a 1khz carrier, with amplitude modulation at 31.25hz
sinus. The
// modulation is digitalized at 16khz. I have tried 64khz frequency
but CPU is
// too slow and 16khz is more than enough to have good waveform.
//
// 1- For 1khz carrier full wave cycle I have 16 sample to output to
R-R2 network.
// 2- The modulation cycle is 32ms (31.25baud), 32 time the carrier:
//    16 samples x 32 cycle = 512 total sample.
//
// 512 sample(bytes) is too much for small F872 memory, I have to cut
a little bit.
//
// 1-Carrier cycle, negative side is a copy of positive, I have
erased
//   all negative side of carrier (180 to 360 degre) cut memory to
256 bytes.
// 2-Modulation, rising side (0% to 100% modulation) is a copy of
falling
//   side, falling side erased.
//
```

```
// Sample table is down to 128 bytes and just a small CPU power is
necessary
// to restore full modulation waveform. (in interrupt function)
//
// 16 samples = 1khz sinus, sythetised at 16khz
// 1000us     = 1khz period,
// CPU Tcycle = 0.25us@16Mhz (Tosc*4)
//
// (1000us / 16samples) / Tcycle = ?
//     (62.5us)         / 0.25us  = 250 cycles
//
// 250 cycle between interrupt: .25us * 250 cycles * 16 samples  =
1000.0Hz frequency tone
//
//
// Usage:
// Type any key to enter in config mode. The beacon is shutdown in
this mode. Type
// R to restore beacon activity.
//
// B<text>    PSK31 beacon text. Type ^ to switch to CW. Type ~ to
output
//            1 sec of 1khz tone. (for CW beacon or end of PSK31
modulation)
//
//              Example: BT VE2YAG Beacon, FN19ES k k k~^  DE VE2YAG
//                       BT ^VE2YAG FN19 ~~ (CW only beacon)
//              note: Use space in CW to insert small delay betwen
PSK and CW.
// E<00-FF>     Wait x second before transmission, 00=countinous
sending, in hexadecimal.
// R            Reset beacon, and start sending.
// D            Send in hex: CPUTYPE + SOFTVERSION + EEPROMSIZE + 00
+ (dump en eeprom)
//
//
// History:
// 03/31/2002  V1.0b01  Initial version for PIC16F872.
// 04/11/2002  V1.1     bug: PTT not release when enter config mode
//
// Pinout:
// TMR0 Alan and AX25 timing
//
// RA0
// RA1
// RA2
// RA3
// RA4
// RA5
// RB0 R-R2 DAC output 1
// RB1 R-R2 DAC output 2
// RB2 R-R2 DAC output 3
// RB3 R-R2 DAC output 4
// RB4 R-R2 DAC output 5
// RB5 R-R2 DAC output 6
// RB6 R-R2 DAC output 7
```

```
// RB7 R-R2 DAC output 8
// RC0
// RC1
// RC2
// RC3
// RC4
// RC5
// RC6 Serial port Output
// RC7 Serial port Input
//
//-------------------------------------------------
#include <pic.h>
#include <ctype.h>

#define CFG_CPUTYPE     0x72  // Type of CPU 16F872
#define CFG_SOFTVERSION 0x10  // Version 1.0
#define CFG_EEPROMSIZE  64    // Size of eeprom

/*{{{  Define*/


#define FALSE    0           // Value of False
#define TRUE  !FALSE      // Value of True

#define   PORTBIT(adr, bit)((unsigned)(&adr)*8+(bit))
#define CLEARWATCHDOG() asm("clrwdt")

#define byte unsigned char
#define word unsigned int

#define WaitCycleEnd() while(ModCount!=0) if(RC7==0) break
#define WaitCycleStart() while(ModCount!=1) if(RC7==0) break

/*}}}  */

/*{{{  variable*/

const unsigned char hex[] = {"0123456789ABCDEF"};

#define MODE_PSK 0
#define MODE_CW 1

#define PTT RC4     // Radio PTT, active +5v
#define KEY RC5     // CW Key, active +5v
#define SERIN RC7   // Serial in (modify in sbit function also)

//-------------------------------------------------------
// EEPROM map
//-------------------------------------------------------
#define MAXCARACTER   49    // Maximum caracter in line buffer
(Beacon size-1)

#define EE_BEACONDELAY    0
#define EE_BEACONTEXTSIZE 1
#define EE_BEACONTEXT     16
```

```
//-----------------------------------------------------
// Modulator variable
//-----------------------------------------------------
#define FLAT     0
#define RISING   1
#define FALLING  2
#define NONE     3
unsigned char ModType;    // type of modulation for half-cycle
unsigned char ModCount;   // 0-255 count for half-cycle
unsigned char ModPhase;   // 0 or 1 for audio signal phase reversal


//-----------------------------------------------------
// Config mode variable
//-----------------------------------------------------
#define BK              8       /* Backspace key */
#define CR              0x0D    /* Carriage return key */
#define DEL             127     /* Delete key */
#define CTRL_C          0x03    /* Ctrl-C key */
unsigned char LineCount;                  // Number of caracter in
buffer
unsigned char LineBuffer[MAXCARACTER];    // Line buffer
const    byte bk_str[] = { BK,' ',BK,0 }; // Backspace string


//-----------------------------------------------------------
// Digitalized PSK31 modulation.
// sinus at 1 khz, only rising side, positive carrier only
// 8 samples(positive side, carrier)
// 16 cycles from 100% to 0% modulated signal
// 8 * 16 = 128 total samples
//-----------------------------------------------------------
const unsigned char Waveform[] = { \
        128, 175, 217, 245, 254, 245, 217, 175,  \
        128, 175, 216, 244, 253, 244, 216, 175,  \
        128, 174, 214, 240, 248, 240, 213, 173,  \
        128, 171, 209, 234, 242, 233, 208, 170,  \
        128, 168, 204, 227, 234, 226, 202, 167,  \
        128, 165, 197, 217, 224, 216, 195, 163,  \
        128, 160, 189, 207, 212, 206, 187, 159,  \
        128, 156, 180, 196, 200, 194, 178, 154,  \
        128, 151, 171, 184, 188, 183, 169, 149,  \
        128, 147, 163, 173, 176, 171, 160, 145,  \
        128, 142, 154, 162, 164, 161, 152, 140,  \
        128, 138, 147, 152, 153, 151, 145, 137,  \
        128, 134, 140, 143, 144, 142, 138, 133,  \
        128, 131, 134, 136, 137, 136, 133, 130,  \
        128, 129, 131, 131, 131, 131, 130, 129,  \
        128, 128, 128, 128, 128, 128, 128, 128 };

//-----------------------------------------------------------
// Varicode: The alphabet of PSK31, only 0-127 value.
//-----------------------------------------------------------
const unsigned int Varicode[128] = {
   0xAAC0,    // ASCII =   0   1010101011
   0xB6C0,    // ASCII =   1   1011011011
```

```
    0xBB40,    // ASCII =   2   1011101101
    0xDDC0,    // ASCII =   3   1101110111
    0xBAC0,    // ASCII =   4   1011101011
    0xD7C0,    // ASCII =   5   1101011111
    0xBBC0,    // ASCII =   6   1011101111
    0xBF40,    // ASCII =   7   1011111101
    0xBFC0,    // ASCII =   8   1011111111
    0xEF00,    // ASCII =   9   11101111
    0xE800,    // ASCII =  10   11101
    0xDBC0,    // ASCII =  11   1101101111
    0xB740,    // ASCII =  12   1011011101
    0xF800,    // ASCII =  13   11111
    0xDD40,    // ASCII =  14   1101110101
    0xEAC0,    // ASCII =  15   1110101011
    0xBDC0,    // ASCII =  16   1011110111
    0xBD40,    // ASCII =  17   1011110101
    0xEB40,    // ASCII =  18   1110101101
    0xEBC0,    // ASCII =  19   1110101111
    0xD6C0,    // ASCII =  20   1101011011
    0xDAC0,    // ASCII =  21   1101101011
    0xDB40,    // ASCII =  22   1101101101
    0xD5C0,    // ASCII =  23   1101010111
    0xDEC0,    // ASCII =  24   1101111011
    0xDF40,    // ASCII =  25   1101111101
    0xEDC0,    // ASCII =  26   1110110111
    0xD540,    // ASCII =  27   1101010101
    0xD740,    // ASCII =  28   1101011101
    0xEEC0,    // ASCII =  29   1110111011
    0xBEC0,    // ASCII =  30   1011111011
    0xDFC0,    // ASCII =  31   1101111111
    0x8000,    // ASCII = ' '   1
    0xFF80,    // ASCII = '!'   111111111
    0xAF80,    // ASCII = '"'   101011111
    0xFA80,    // ASCII = '#'   111110101
    0xED80,    // ASCII = '$'   111011011
    0xB540,    // ASCII = '%'   1011010101
    0xAEC0,    // ASCII = '&'   1010111011
    0xBF80,    // ASCII = '''   101111111
    0xFB00,    // ASCII = '('   11111011
    0xF700,    // ASCII = ')'   11110111
    0xB780,    // ASCII = '*'   101101111
    0xEF80,    // ASCII = '+'   111011111
    0xEA00,    // ASCII = ','   1110101
    0xD400,    // ASCII = '-'   110101
    0xAE00,    // ASCII = '.'   1010111
    0xD780,    // ASCII = '/'   110101111
    0xB700,    // ASCII = '0'   10110111
    0xBD00,    // ASCII = '1'   10111101
    0xED00,    // ASCII = '2'   11101101
    0xFF00,    // ASCII = '3'   11111111
    0xBB80,    // ASCII = '4'   101110111
    0xAD80,    // ASCII = '5'   101011011
    0xB580,    // ASCII = '6'   101101011
    0xD680,    // ASCII = '7'   110101101
    0xD580,    // ASCII = '8'   110101011
    0xDB80,    // ASCII = '9'   110110111
```

```
    0xF500,    // ASCII = ':'   11110101
    0xDE80,    // ASCII = ';'   110111101
    0xF680,    // ASCII = '<'   111101101
    0xAA00,    // ASCII = '='   1010101
    0xEB80,    // ASCII = '>'   111010111
    0xABC0, // ASCII = '?'  1010101111
    0xAF40,    // ASCII = '@'   1010111101
    0xFA00,    // ASCII = 'A'   1111101
    0xEB00,    // ASCII = 'B'   11101011
    0xAD00,    // ASCII = 'C'   10101101
    0xB500,    // ASCII = 'D'   10110101
    0xEE00,    // ASCII = 'E'   1110111
    0xDB00,    // ASCII = 'F'   11011011
    0xFD00,    // ASCII = 'G'   11111101
    0xAA80,    // ASCII = 'H'   101010101
    0xFE00,    // ASCII = 'I'   1111111
    0xFE80,    // ASCII = 'J'   111111101
    0xBE80,    // ASCII = 'K'   101111101
    0xD700,    // ASCII = 'L'   11010111
    0xBB00,    // ASCII = 'M'   10111011
    0xDD00,    // ASCII = 'N'   11011101
    0xAB00,    // ASCII = 'O'   10101011
    0xD500,    // ASCII = 'P'   11010101
    0xEE80,    // ASCII = 'Q'   111011101
    0xAF00,    // ASCII = 'R'   10101111
    0xDE00,    // ASCII = 'S'   1101111
    0xDA00,    // ASCII = 'T'   1101101
    0xAB80,    // ASCII = 'U'   101010111
    0xDA80,    // ASCII = 'V'   110110101
    0xAE80,    // ASCII = 'W'   101011101
    0xBA80,    // ASCII = 'X'   101110101
    0xBD80,    // ASCII = 'Y'   101111011
    0xAB40,    // ASCII = 'Z'   1010101101
    0xFB80,    // ASCII = '['   111110111
    0xF780,    // ASCII = '\'   111101111
    0xFD80,    // ASCII = ']'   111111011
    0xAFC0,    // ASCII = '^'   1010111111
    0xB680,    // ASCII = '_'   101101101
    0xB7C0,    // ASCII = '`'   1011011111
    0xB000,    // ASCII = 'a'   1011
    0xBE00,    // ASCII = 'b'   1011111
    0xBC00,    // ASCII = 'c'   101111
    0xB400,    // ASCII = 'd'   101101
    0xC000,    // ASCII = 'e'   11
    0xF400,    // ASCII = 'f'   111101
    0xB600,    // ASCII = 'g'   1011011
    0xAC00,    // ASCII = 'h'   101011
    0xD000,    // ASCII = 'i'   1101
    0xF580,    // ASCII = 'j'   111101011
    0xBF00,    // ASCII = 'k'   10111111
    0xD800,    // ASCII = 'l'   11011
    0xEC00,    // ASCII = 'm'   111011
    0xF000,    // ASCII = 'n'   1111
    0xE000,    // ASCII = 'o'   111
    0xFC00,    // ASCII = 'p'   111111
    0xDF80,    // ASCII = 'q'   110111111
```

```
    0xA800,    // ASCII = 'r'   10101
    0xB800,    // ASCII = 's'   10111
    0xA000,    // ASCII = 't'   101
    0xDC00,    // ASCII = 'u'   110111
    0xF600,    // ASCII = 'v'   1111011
    0xD600,    // ASCII = 'w'   1101011
    0xDF00,    // ASCII = 'x'   11011111
    0xBA00,    // ASCII = 'y'   1011101
    0xEA80,    // ASCII = 'z'   111010101
    0xADC0,    // ASCII = '{'   1010110111
    0xDD80,    // ASCII = '|'   110111011
    0xAD40,    // ASCII = '}'   1010110101
    0xB5C0,    // ASCII = '~'   1011010111
    0xED40 // ASCII = 127   1110110101
     };

//--------------------------------------------------------------------
// The word in the CW table is divided into 8 groups of two bits
starting
// at the msb side.  The two bits represent one of four possible
states.
//  00 - end of character
//  01 - DOT
//  10 - DASH
//  11 - SPACE of two dot times
//--------------------------------------------------------------------
const unsigned int morse[59] = {
    0xF000,       // 1111 1100 0000 0000b  ( 32)       WORD SPACE
    0x0000,       // 0000 0000 0000 0000b  ( 33)  !
    0x0000,       // 0000 0000 0000 0000b  ( 34)  "
    0x0000,       // 0000 0000 0000 0000b  ( 35)  #
    0x0000,       // 0000 0000 0000 0000b  ( 36)  $
    0x0000,       // 0000 0000 0000 0000b  ( 37)  %
    0x0000,       // 0000 0000 0000 0000b  ( 38)  &
    0x0000,       // 0000 0000 0000 0000b  ( 39)  '
    0x0000,       // 0000 0000 0000 0000b  ( 40)  (
    0x0000,       // 0000 0000 0000 0000b  ( 41)  )
    0x566C,       // 0101 0110 0110 1100b  ( 42)  *    ...-.-  SK
    0x6670,       // 0110 0110 0111 0000b  ( 43)  +    .-.-.   AR
    0xA5AC,       // 1010 0101 1010 1100b  ( 44)  ,    --..--
    0x0000,       // 0000 0000 0000 0000b  ( 45)  -
    0x666C,       // 0110 0110 0110 1100b  ( 46)  .    .-.-.-
    0x9670,       // 1001 0110 0111 0000b  ( 47)  /    -..-.
    0xAAB0,       // 1010 1010 1011 0000b  ( 48)  0    -----
    0x6AB0,       // 0110 1010 1011 0000b  ( 49)  1    .----
    0x5AB0,       // 0101 1010 1011 0000b  ( 50)  2    ..---
    0x56B0,       // 0101 0110 1011 0000b  ( 51)  3    ...--
    0x55B0,       // 0101 0101 1011 0000b  ( 52)  4    ....-
    0x5570,       // 0101 0101 0111 0000b  ( 53)  5    .....
    0x9570,       // 1001 0101 0111 0000b  ( 54)  6    -....
    0xA570,       // 1010 0101 0111 0000b  ( 55)  7    --...
    0xA970,       // 1010 1001 0111 0000b  ( 56)  8    ---..
    0xAA70,       // 1010 1010 0111 0000b  ( 57)  9    ----.
    0x0000,       // 0000 0000 0000 0000b  ( 58)  :
    0x0000,       // 0000 0000 0000 0000b  ( 59)  ;
    0x0000,       // 0000 0000 0000 0000b  ( 60)  <
```

```
    0x95B0,       // 1001 0101 1011 0000b  ( 61)  =    -...-   BT
    0x0000,       // 0000 0000 0000 0000b  ( 62)  >
    0x5A5C,       // 0101 1010 0101 1100b  ( 63)  ?    ..--..
    0x0000,       // 0000 0000 0000 0000b  ( 64)  @
    0x6C00,       // 0110 1100 0000 0000b  ( 65)  A    .-
    0x95C0,       // 1001 0101 1100 0000b  ( 66)  B    -...
    0x99C0,       // 1001 1001 1100 0000b  ( 67)  C    -.-.
    0x9700,       // 1001 0111 0000 0000b  ( 68)  D    -..
    0x7000,       // 0111 0000 0000 0000b  ( 69)  E    .
    0x59C0,       // 0101 1001 1100 0000b  ( 70)  F    ..-.
    0xA700,       // 1010 0111 0000 0000b  ( 71)  G    --.
    0x55C0,       // 0101 0101 1100 0000b  ( 72)  H    ....
    0x5C00,       // 0101 1100 0000 0000b  ( 73)  I    ..
    0x6AC0,       // 0110 1010 1100 0000b  ( 74)  J    .---
    0x9B00,       // 1001 1011 0000 0000b  ( 75)  K    -.-
    0x65C0,       // 0110 0101 1100 0000b  ( 76)  L    .-..
    0xAC00,       // 1010 1100 0000 0000b  ( 77)  M    --
    0x9C00,       // 1001 1100 0000 0000b  ( 78)  N    -.
    0xAB00,       // 1010 1011 0000 0000b  ( 79)  O    ---
    0x69C0,       // 0110 1001 1100 0000b  ( 80)  P    .--.
    0xA6C0,       // 1010 0110 1100 0000b  ( 81)  Q    --.-
    0x6700,       // 0110 0111 0000 0000b  ( 82)  R    .-.
    0x5700,       // 0101 0111 0000 0000b  ( 83)  S    ...
    0xB000,       // 1011 0000 0000 0000b  ( 84)  T    -
    0x5B00,       // 0101 1011 0000 0000b  ( 85)  U    ..-
    0x56C0,       // 0101 0110 1100 0000b  ( 86)  V    ...-
    0x6B00,       // 0110 1011 0000 0000b  ( 87)  W    .--
    0x96C0,       // 1001 0110 1100 0000b  ( 88)  X    -..-
    0x9AC0,       // 1001 1010 1100 0000b  ( 89)  Y    -.--
    0xA5C0    // 1010 0101 1100 0000b  ( 90)  Z    --..
};
/*}}}  */


/*{{{  Interrupt Routine*/
/* ---------------------------------------------------------------
 * Interrupt Routine
 * 250 cycle interrupt interval.
 * .25us * 250 cycles * 16 samples = 1000.0Hz frequency tone at 31.25
baud
 * ---------------------------------------------------------------
 */

unsigned char DACvalue;

void interrupt tc_int(void) {
    TMR0 = 6+17;
    switch(ModType) {
        case RISING:  DACvalue =
Waveform[(ModCount&0x07)+(((~ModCount)>>1)&0x78)];  break;
        case FALLING: DACvalue =
Waveform[(ModCount&0x07)+((ModCount>>1)&0x78)];     break;
        case FLAT:    DACvalue = Waveform[ModCount&7];  break;
        case NONE:    DACvalue = 128; break;
    };
```

```
    if(ModCount&8) DACvalue=~DACvalue; // Create negative if needed
    if(ModPhase) DACvalue=~DACvalue;   // Invert sample if phase is
180 degre.
    PORTB = DACvalue;
    ModCount+=1;

    T0IF = 0;
}
/*}}}  */


/*{{{  eeprom function*/

void EEPromWrite(addr, value) {
    while(WR)continue;
    EEADR=(addr);
    EEDATA=(value);
    GIE=0;
    WREN=1;
    EECON2=0x55;
    EECON2=0xAA;
    WR=1;
    WREN=0;
    GIE=1;
}

unsigned char EEPromRead(unsigned char addr) {
    return ((EEADR=(addr)),(RD=1),EEDATA);
}
/*}}}  */

/*{{{  Serial port function*/

#define baud 31            // Bit delay for 2400 baud (4-MHz PIC
clock).
#define ser_o       7,6    // Serial Output Pin RC6
#define ser_i       7,7    // Serial Input Pin RC7

byte temp;                // Temporary Register (used by serial,)
byte bits;                // Bit counter used with Serial Routines
byte xbyte;                // Transmit/Receive Register for Serial
Routines

/*{{{  sbit   - basic delay function*/

#asm
;---------------------------------------------------------------------
; Low-Level Serial Routine  (FOR DEVICE WITHOUT USART !)
;
; Use: temp, xbyte, bits, (baud, sio_p)
;---------------------------------------------------------------------

; --- Subroutine used by Serout to send a bit and generate time
delays.
sBit_
        BTFSS 3,0            ; Output Carry flag on serial pin
```

```
        BCF ser_o
        BTFSC 3,0
        BSF ser_o

        MOVLW baud          ; Baud-rate delay in temp variable
        MOVWF _temp

                           ; Delay routine used by Serin & Serout.
Load the length of
                           ; delay required into temp before calling.

sloop_
;        GOTO    sl1        ; Two-cycle nops for time delay.
;sl1
;        GOTO    sl2        ; "
;sl2
;        GOTO    sl3        ; "
;sl3
;        GOTO    sl4        ; "
;sl4
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

        NOP                 ; Two-cycle nops = ( NOP+ CLRWDT )
       CLRWDT             ; Clear Watchdog !
        DECFSZ  _temp       ; Number of trips through loop
        GOTO    sloop_
                           ; Set by baud rate in temp.
       RETURN             ; Return to calling routine.
#endasm
/*}}}  */

/*{{{  Serout - Serial output*/

void Serout(char car) {
    xbyte = car;
#asm
; ---------------------------------------------------------------------
; --- Transmits serial a single byte with a fixed output pin, baud
rate,
; and polarity. Serout uses the common data format of no parity, 8
data
; bits, 1 stop bit (N81).
; - Upon entry, the desired pin must already be set up as an output.
; - Upon return, the data in the buffer will be unchanged

        BCF     11,7        ; Disable interrupt

        BCF     3,0   ; Clear carry (Set up start bit, send Raw Data)
       CALL   sBit_      ; Send start bit.
```

```
        MOVLW   8           ; Send 8 data bits.
        MOVWF   _bits

xbit_
        RRF       _xbyte    ; Rotate bit into carry.
       CALL   sBit_      ; Send the data bit.
        DECFSZ  _bits       ; Number of trips through loop
        GOTO    xbit_
       RRF    _xbyte     ; Rotate data back into original position.
       BSF     3,0           ; Set up stop bit.
       CALL   sBit_      ; Send the stop bit.

        BSF     11,7        ; Enable interrupt
#endasm
}
/*}}}  */

/*{{{  Serin  - Serial input (temp=255 if caracter received else
temp=0) */

unsigned char Serin() {
#asm
;---------------------------------------------------------------------
;--- Receive serially a single byte with a fixed input pin, baud
rate,
; and polarity. Serin uses the common data format of no parity, 8
data
; bits, 1 stop bit (N81).
; - Upon entry, the desired pin must already be set up as an input.
; - Upon return, the data in the buffer wil be the received byte.
;
;   IF TEMP=0 NO CARACTER RECEIVED -> ELSE TEMP=255.
;

        CLRF    _xbyte      ; Clear Receive Register

rpoll_
        CLRF    _temp
        CLRWDT            ; Clear Watchdog
        BTFSC   ser_i
        RETLW   0

        BCF     11,7        ; Disable interrupt

        MOVLW   baud/2    ; Wait 1/2 bit time to middle of start bit.
        MOVWF   _temp
        CALL  sloop_      ; Call Bit Delay Sub-Routine
        BTFSC   ser_i
        GOTO    rpoll_

        MOVLW   8            ; Load 8 bit into xbyte
        MOVWF   _bits
rcv_
        MOVLW   baud         ; Wait bit time.
        MOVWF   _temp
        CALL  sloop_      ; Call Bit Delay Sub-Routine
```

```
        BTFSS   ser_i        ; Put serial pin into carry
        BCF     3,0
        BTFSC   ser_i
        BSF     3,0
        RRF     _xbyte      ; Rotate carry into msb of data byte.
        DECFSZ  _bits        ; Bits=bits-1. If bits<>0 then rcv_.
        GOTO    rcv_

        MOVLW   baud         ; Delay 1 bit time for stop bit.
        MOVWF   _temp
        CALL  sloop_     ; Call Bit Delay Sub-Routine
        DECF    _temp

        BSF     11,7         ; Enable interrupt
#endasm
    return xbyte;
}
/*}}}  */

/*{{{  SerStr - send string to serial port*/

void SerStr(const char *p) {
    unsigned char i;

    i=0;
    while(1) {
       if(p[i]==0) return;
       Serout(p[i++]);
    }
}
/*}}}  */

/*{{{  SerHex - Send 8-bit hex number to serial port*/

void SerHex(unsigned char c) {
    Serout(hex[c>>4]);
    Serout(hex[c&15]);
}
/*}}}  */

/*}}}  */

/*{{{  PSK function*/

/*{{{  PSKStartSync - Send 60 zero*/
//-----------------------------------------------
// Send 60 zero (phase reversal)
//-----------------------------------------------
void PSKStartSync() {
    unsigned char i;

    for(i=0; i<60; i++) {
        WaitCycleEnd();
        ModType=FALLING;
        WaitCycleStart();
```

```
        WaitCycleEnd();
        if(ModPhase)          // Invert phase in middle of cycle
            ModPhase=0;
         else
            ModPhase=1;
        ModType=RISING;
        WaitCycleStart();
    }
};
/*}}}  */

/*{{{  PSKPutc - Send caracter to PSK31*/

void PSKPutc(unsigned char c) {
    unsigned int code;
    unsigned char zero;

    code = Varicode[c&127];
    zero = 0;

    while(zero!=2) {
        if(code&0x8000) {
            /*{{{  Constant phase*/

    // Set modulation FLAT for all cycle
    WaitCycleEnd();
    ModType=FLAT;
    WaitCycleStart();
    WaitCycleEnd();
    WaitCycleStart();
/*}}}  */

            zero=0;
        } else {
            /*{{{  Reverse phase*/

    WaitCycleEnd();
    ModType=FALLING;
    WaitCycleStart();

    WaitCycleEnd();
    if(ModPhase)          // Invert phase in middle cycle
        ModPhase=0;
     else
        ModPhase=1;
    ModType=RISING;
    WaitCycleStart();
/*}}}  */

            zero++;
        }
        code = code<<1;
    }
}
/*}}}  */
```

```
/*{{{  CWPutc - Send caracter in CW*/
////////////////////  Various constant tabels
//////////////////////////
// The word in the CW table is divided into 8 groups of two bits
starting
// at the msb side.  The two bits represent one of four possible
states.
//  00 - end of character
//  01 - DOT
//  10 - DASH
//  11 - SPACE of two dot times

void Dot() {
    unsigned char i;

    for(i=0; i<5; i++) {
        WaitCycleEnd();
        WaitCycleStart();
    }
};

void CWPutc(unsigned char c) {
    unsigned int code;

    if(c<32 || c>90) return;
    code = morse[c-32];

    while(1) {
        if((code&0xC000)==0xC000) Dot();
        if((code&0xC000)==0x8000) { ModType=FLAT; KEY=1; Dot();
Dot(); Dot(); KEY=0; }
        if((code&0xC000)==0x4000) { ModType=FLAT; KEY=1; Dot();
KEY=0; }
        if((code&0xC000)==0x0000) break;
        ModType=NONE; Dot();
        code=code<<2;
    }
}
/*}}}  */

/*}}}  */

/*{{{  Misc function*/

/*{{{  Hex2Dec - Conversion function*/

//-----------------------------------------
// Convert binary to hexadecimal caracter
//-----------------------------------------
unsigned char Hex2Dec(char c) {
    if(c>='0' && c<='9') return c-48;
    if(c>='A' && c<='F') return c-55;
    if(c>='a' && c<='f') return c-87;
    return 255;
}
/*}}}  */
```

```
/*{{{  WaitSec - Wait x second*/
//-----------------------------------------
// Wait aprox. 1 sec (0.992sec)
//-----------------------------------------
void WaitSec(unsigned char c) {
    unsigned int i;

    for(i=0; i<(unsigned int)c*62; i++) {      // wait 0.992sec
        WaitCycleEnd();
        WaitCycleStart();
        if(SERIN==0) return;
    }
};
/*}}}  */


/*}}}  */


/*{{{  Main*/


void main() {
    byte c,i,mode;

    TRISA   = 0b11111111;       // All input
    TRISC   = 0b10001111;       // serial port in/out RC6+RC7 PTT and
KEY output
    TRISB   = 0b00000000;       // R-R2 DAC output port

    OPTION  = 0b00001111;       // Pull-up + WDT prescale at 1:128
    INTCON  = 0b00100000;       // TMR0 interrupt
    PIE1    = 0b00000000;       // No peripheral interrupt
    PIE2    = 0b00000000;
    T1CON   = 0b00000000;       // TMR1 is off
    T2CON   = 0b00000000;       // TMR2 is off
    CCP1CON = 0b00000000;       // Capture/compare/pwm #1 is off
    SSPCON  = 0b00000000;       // Sync serial port is off
    ADCON0  = 0b00000000;       // A/D is off
    ADCON1  = 0b00000110;       // A/D is off

    TMR0 = 6+14;                // For 16 sample at 1khz@16Mhz
                                // 1000us = 1khz period,
Tcycle=0.25us@20Mhz
                                // (1000us/16)/Tcycle
                                // 62.5us / 0.25us = 250 cycles
between interrupt
                                // 256 - 250 = 6 + 14(delay to reload
timer)
                                // 250 cycles * 0.25us * 16 samples =
1000.0hz at 31.25baud

    ModType = FLAT;
    PTT = 0;
    KEY = 0;
```

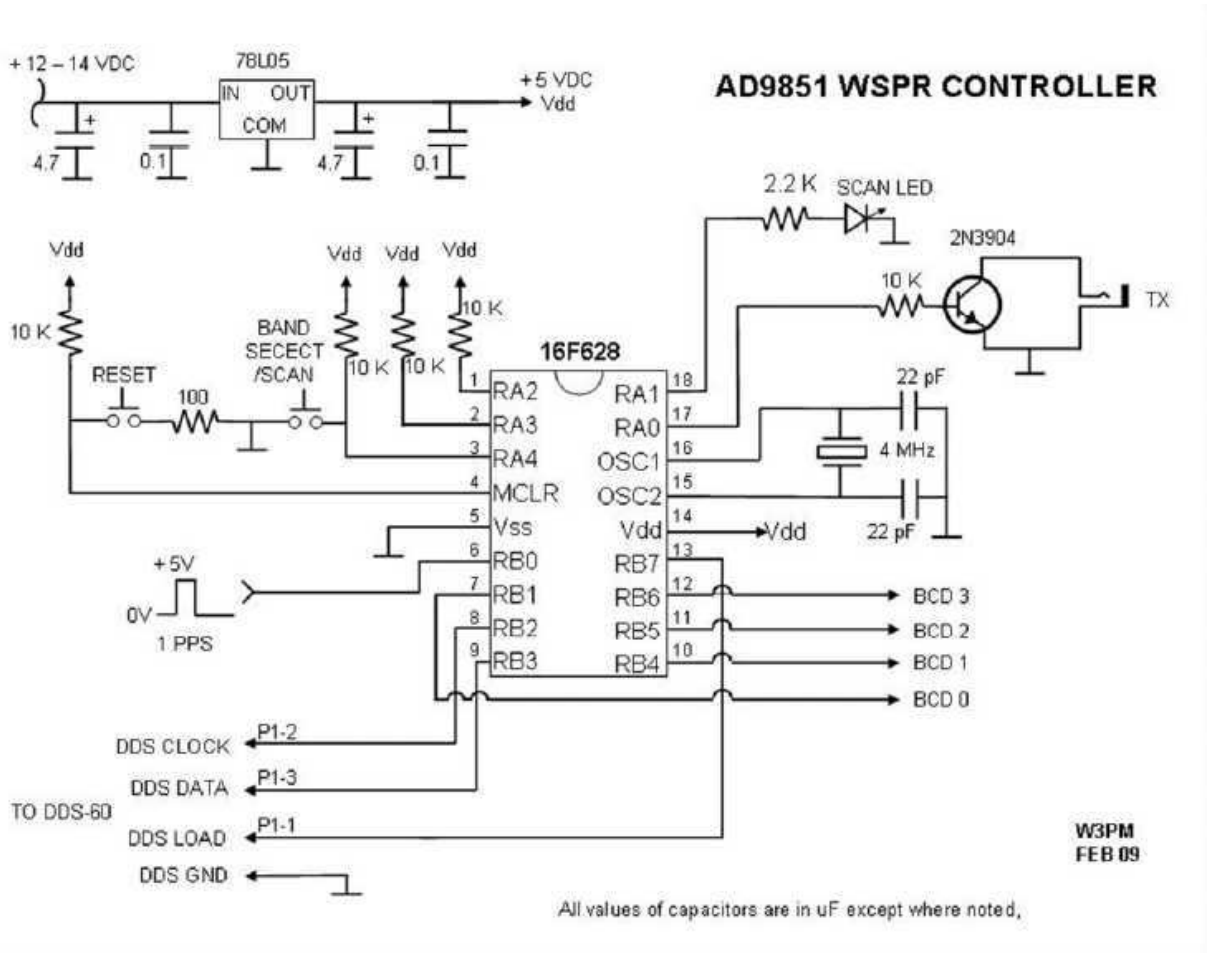
```
    SerStr("PSK31 beacon for PIC, by VE2YAG\r\r");

    T0IE = 1;           // Enable TMR0 interrupt
    while(1) {
        /*{{{  Sending mode*/

    i = 0;
    mode = MODE_PSK;

    while(1) {

        if(SERIN==0) break;       // Check if key pressed

        //-----------------------------------------
        //  Verify for end of message, back to PSK,
        //  stop carrier and wait delay
        //-----------------------------------------
        if(i==EEPromRead(EE_BEACONTEXTSIZE)) {
            ModType = NONE;
            mode = MODE_PSK;
            PTT=0;
            i=0;
            WaitSec(EEPromRead(EE_BEACONDELAY));
            continue;
        }

        c = EEPromRead(EE_BEACONTEXT+i);    // Read caracter to send

        //--------------------------------
        //  Switch to CW, stop carrier
        //--------------------------------
        if(c=='^') { ModType = NONE;
                     mode = MODE_CW;
                     i++;
                     continue;
                   };

        PTT=1;
        i++;

        //--------------------------------
        //  TONE sending
        //--------------------------------
        if(c=='~') {
            ModType = FLAT;
            WaitSec(1);
            continue;
        }

        //--------------------------------
        //  PSK sending
        //--------------------------------
        if(mode == MODE_PSK) {
            if(i==1) PSKStartSync();
            PSKPutc(c);
            continue;
```

```
        }

        //-------------------------------------------
        //  CW sending
        //-------------------------------------------
        CWPutc(toupper(c));

    }
/*}}}  */

        /*{{{  Config mode*/

//-----------------------------
// Enter in config mode
//-----------------------------
    ModType = NONE;
    PTT = 0; KEY=0;
    LineCount = 0;
    T0IE = 0;   // Disable TMR0 interrupt

    SerStr("Config mode: ? for help\r>");

    while(1) {
        c = Serin();
        if(temp) {
            if( (c==DEL || c==BK) && LineCount) { SerStr(bk_str);
LineCount--; continue; }
            if(c==CR) { /*{{{  Process command*/

    if(LineCount==0) { SerStr("\r>"); continue; }

    switch(LineBuffer[0]) {
        case 'R': /*{{{  R command*/

 SerStr("\rBeacon Activated\r"); LineCount=0; /*}}}  */
 break;
        case 'B': /*{{{  B command*/

    for(c=0; c<LineCount-1; c++) EEPromWrite(EE_BEACONTEXT+c,
LineBuffer[c+1]);
    EEPromWrite(EE_BEACONTEXTSIZE, LineCount-1);
/*}}}  */
 break;
        case 'E': /*{{{  E command*/

    c = Hex2Dec(LineBuffer[1]);
    temp = Hex2Dec(LineBuffer[2]);

    if((c==255) || (temp==255)) { SerStr("\r?error"); break; }

    EEPromWrite(EE_BEACONDELAY, (c*16)+temp);
/*}}}  */
 break;
        case 'D': /*{{{  D command*/

    SerStr("\r!");
```

```
    SerHex(CFG_CPUTYPE);
    SerHex(CFG_SOFTVERSION);
    SerHex(CFG_EEPROMSIZE);
    SerHex(0x00);

    for(i=0; i<CFG_EEPROMSIZE; i++) SerHex(EEPromRead(i));
/*}}}  */
 break;
        case '?': /*{{{  ? command*/

    SerStr("\rB<text> ~ for 1sec tone ^ to switch CW\rExx     (Second
to wait in hex)\rR       (Restart)\rD       (eeprom hex dump)");
/*}}}  */
 break;
        default:  SerStr("\r?error");
    }

    if(LineCount==0) break; // Reset

    SerStr("\r>");
    LineCount=0;
/*}}}  */
 }
            if(LineCount<MAXCARACTER && c>31 && c<127) {
                LineBuffer[LineCount++]=c;
                Serout(c);
            }
         }
    }

    T0IE = 1;   // Enable TMR0 interrupt
/*}}}  */

    }
}

/*}}}  */
```

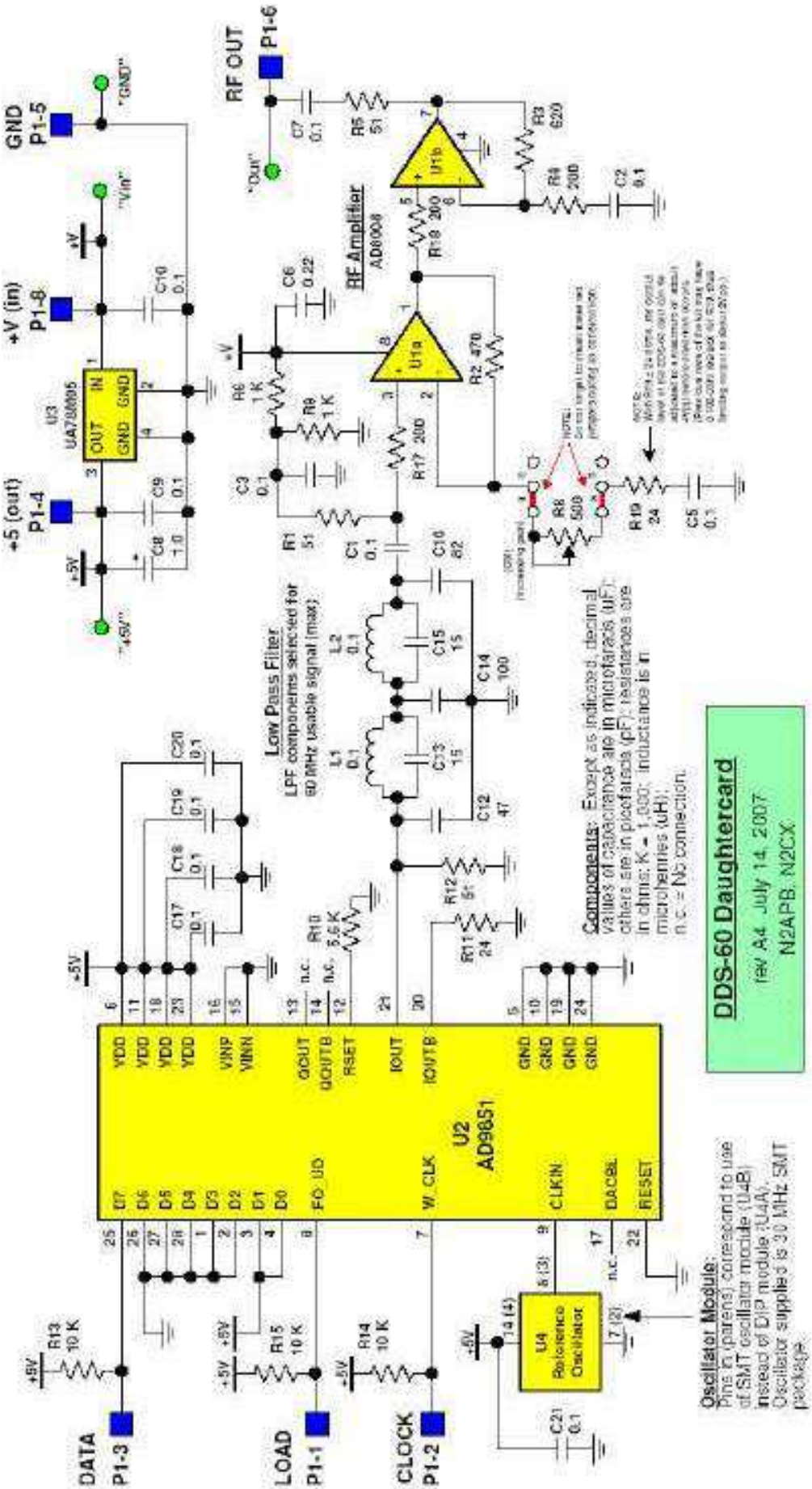
Radiolatarnia WSPR na 16F628 i AD9851

Program opracowany przez W3PM (GM4YRE) pozwala na uruchomienie radiolatarni pracującej na dowolnym paśmie amatorskim od 160 do 6 m. Może ona pracować na jednym wybranym paśmie albo przełączać się automatycznie na kolejne pasma. Domyślnie po włączeniu wybierane jest pasmo 30 m. Do ręcznej zmiany pasma albo wyboru trybu automatycznego przełączania służy przycisk „BAND SELECT/SCAN". Jego naciśnięcie – w czasie pomiędzy transmisjami – powoduje przejście na kolejne wyższe pasmo. W celu włączenia trybu automatycznego należy go nacisnąć i przytrzymać naciskając przycisk zerowania („RESET") na początku parzystej minuty. W trybie automatycznym transmisja rozpoczyna się na pierwszym paśmie powyżej domyślnego i po zakończeniu nadawania następuje przejście na kolejne pasmo. W celu zakończenia trybu automatycznego należy nacisnąć przycisk zerowania. Mikroprocesor 16F628A steruje cyfrowy syntezer AD9851 pracujący na częstotliwości nadawania radiolatarni. Jako VFO zastosowano moduł DDS-60 opracowany przez amerykański klub QRP. Komunikacja z syntezerem odbywa się za pomocą magistrali SPI złożonej z trzech przewodów odpowiadających sygnałom DDS_DATA, DDS_CLOCK i DDS_LOAD. Czteroprzewodowe wyjście BCD służy do przełączania wzmacniaczy i filtrów dolnoprzepustowych w zależności od pasma nadawania. Dioda świecąca sygnalizuje tryb automatycznego przełączania pasm (na wyjściu podawane jest napięcie +5 V). Również na wyjście kluczowania nadajnika podawane jest napięcie +5 V na czas nadawania. Wszystkie istotne parametry transmisji można łatwo zidentyfikować w kodzie źródłowym i dostosować do własnych potrzeb albo do innych typów syntezerów i modułów VFO.



Rys. 4.8. Sterownik na 16F628

Moduł DDS-60 zawierający syntezer AD9851, generator sterujący i dolnoprzepustowy filtr eliptyczny piątego rzędu może pracować w zakresie 1 – 60 MHz. Moduł ten lub rozwiązania na nim wzorowane są często spotykane w konstrukcjach amatorskich.



Rys. 4.9. Schemat ideowy modułu DDS-60

O ile na schemacie nie zaznaczono inaczej oporności podane są w kΩ, pojemności w pF a indukcyjności w µH. Filtr dolnoprzepustowy zaprojektowany dla maksymalnej częstotliwości wyjściowej 60 MHz.

```
;
; DDS_WSPR - Uses AD9851 to tranmit WSPR code on 20% transmit cycle
;           - Multi-band 160 thru 6 meters
;          - Transmit band scanning
;
;
;  Gene Marcus W3PM GM4YRE
;  16 February, 2009
;
;  Clock Frequency: 4 MHz
;
;
;
;                             PIC16F628A
;                            __________
;     Not used    ----RA2 |1       18| RA1---------Scan LED
;     Not used    ----RA3 |2       17| RA0---------Transmit
;     PB_1-Bandswitch-RA4 |3       16| OSC1--------XTAL
;     +5V-----------!MCLR |4       15| OSC2--------XTAL
;     Ground----------Vss |5       14| VDD---------+5 V
;     1 PPS-----------RB0 |6       13| RB7---------DDS_LOAD
;     Band BCD0-------RB1 |7       12| RB6---------Band BCD3
;     DDS_CLK---------RB2 |8       11| RB5---------Band BCD2
;     DDS_DATA--------RB3 |9       10| RB4---------Band BCD1
;                          ----------
;

;=====================================================================

        processor    pic16f628a
        include      "p16f628a.inc"
        __config     _CP_OFF & _LVP_OFF & _BODEN_OFF & _MCLRE_ON & _INTRC_OSC_NOCLKOUT
& _WDT_OFF & _PWRTE_ON

        movlw        0x07
        movwf        CMCON        ; Turn off comparator

        list     b=4,n=70

;=====================================================================
;   Manifest Constants
;=====================================================================

DDS_clk  equ     0x02        ; AD9850/AD9851 write clock
DDS_dat  equ     0x03        ; AD9850/AD9851 serial data input
DDS_load equ     0x07        ; Update pin on AD9850/AD9851
pb_1        equ     0x04        ; Change band/band scan
TX          equ       0x00     ; Transmit on/off
BCD_0        equ      0x01     ; BCD outputs used to control
BCD_1        equ      0x04     ; CD4028 BCD to decimal switch
BCD_2        equ      0x05     ; or directly for amplifier/filter
BCD_3        equ      0x06     ; selection and/or LED band indicator
scan_LED equ      0x01     ; Used to indicate scan enabled

;=====================================================================
;   File register use
;=====================================================================
        cblock       H'20'
```

```
            basefreq_0          ; Base frequency MSB
            basefreq_1          ;
            basefreq_2          ;
            basefreq_3          ; Base frequency LSB
            offset              ; Offset to add to base frequency
            offset_0        ; WSPR offset for symbol o
            offset_1        ; WSPR offset for symbol 1
            offset_2        ; WSPR offset for symbol 2
            offset_3        ; WSPR offset for symbol 3
            DDSword_0           ; (base freq + offset) MSB
            DDSword_1           ;
            DDSword_2           ;
            DDSword_3           ; (base freq + offset) LSB
            DDSword_4           ; x6 multiply for AD9851
            temp
            temp1
            temp2
            temp3
        bit_count
        byte2send
            timer1
            timer2
            Inputs
            counter
            w_temp
            status_temp
            sec_count
            min_count
            symbolcount
            symboltimer
            sendcode
            pad
            two_sec
            band
            led_stat
            scan_flag
        endc

        goto    start


;=====================================================================
;  Subroutines
;=====================================================================

    ; Interrupt service routine - triggered by 1 PPS GPS on pin 6 (RB0)

IRQSVC
        org         H'04'           ; Interrupt 1PPS starts here
        movwf       w_temp          ; Save off the W register
        swapf       STATUS,W    ; And the STATUS (use swapf
        movwf       status_temp     ; so as not to change STATUS)

        decfsz      sec_count       ; End of even 2 minute iterval?
        goto    main3           ; No, end interrupt service routine
        movlw       D'120'          ; Yes, reset 2 minute counter
        movwf       sec_count       ;
        clrf    scan_flag       ; start scan transmit

        decfsz      min_count       ; End of 10 minute interval?
        goto    main3           ; No, end interrupt service routine
        movlw       D'5'        ; Yes, reset 10 minute counter
```

```
        movwf       min_count       ;
        clrf    sendcode    ; Flag to transmit

main3
        bcf         INTCON,T0IF     ; Clear the old interrupt
        bcf         INTCON,INTF

        swapf       status_temp,W; Restore the status
        movwf       STATUS          ; register
        swapf       w_temp,F    ; Restore W without disturbing
        swapf       w_temp,W    ; the STATUS register

        retfie

;
;
;_____________________________________________________________________________
;
;  STATION CALLSIGN, POWER LEVEL, AND GRIDSQURE IS CONTAINED IN THE FOLLOWING TABLE
;
;  Symbol table contains the station's callsign, powerlevel, and gridsquare
;
;  The follwing data is for W3PM, 10 dBm, EM64
;
;  Note: End of table flag is D'4'
;
;_____________________________________________________________________________
;

;               org         H'A0'
Table                               ; WSPR symbol data starts here
                addwf   PCL,F
                retlw       D'3'
                retlw       D'1'
                retlw       D'0'
                retlw       D'0'
                retlw       D'2'
                retlw       D'0'
                retlw       D'0'
                retlw       D'2'
                retlw       D'1'
                retlw       D'0'
                retlw       D'0'
                retlw       D'0'
                retlw       D'3'
                retlw       D'1'
                retlw       D'1'
                retlw       D'0'
                retlw       D'2'
                retlw       D'2'
                retlw       D'3'
                retlw       D'0'
                retlw       D'0'
                retlw       D'3'
                retlw       D'0'
                retlw       D'3'
                retlw       D'3'
                retlw       D'3'
                retlw       D'1'
                retlw       D'2'
                retlw       D'0'
                retlw       D'0'
                retlw       D'0'
```

```
        retlw   D'2'
        retlw   D'0'
        retlw   D'2'
        retlw   D'3'
        retlw   D'0'
        retlw   D'0'
        retlw   D'1'
        retlw   D'2'
        retlw   D'1'
        retlw   D'2'
        retlw   D'0'
        retlw   D'2'
        retlw   D'2'
        retlw   D'2'
        retlw   D'0'
        retlw   D'3'
        retlw   D'0'
        retlw   D'3'
        retlw   D'3'
        retlw   D'2'
        retlw   D'2'
        retlw   D'3'
        retlw   D'1'
        retlw   D'0'
        retlw   D'3'
        retlw   D'2'
        retlw   D'2'
        retlw   D'0'
        retlw   D'3'
        retlw   D'1'
        retlw   D'2'
        retlw   D'1'
        retlw   D'0'
        retlw   D'2'
        retlw   D'2'
        retlw   D'2'
        retlw   D'1'
        retlw   D'1'
        retlw   D'0'
        retlw   D'1'
        retlw   D'2'
        retlw   D'1'
        retlw   D'2'
        retlw   D'1'
        retlw   D'0'
        retlw   D'1'
        retlw   D'2'
        retlw   D'2'
        retlw   D'1'
        retlw   D'0'
        retlw   D'0'
        retlw   D'3'
        retlw   D'2'
        retlw   D'1'
        retlw   D'1'
        retlw   D'2'
        retlw   D'0'
        retlw   D'0'
        retlw   D'1'
        retlw   D'3'
        retlw   D'0'
        retlw   D'3'
        retlw   D'0'
```

```
        retlw   D'1'
        retlw   D'2'
        retlw   D'2'
        retlw   D'2'
        retlw   D'1'
        retlw   D'2'
        retlw   D'2'
        retlw   D'2'
        retlw   D'2'
        retlw   D'0'
        retlw   D'1'
        retlw   D'0'
        retlw   D'0'
        retlw   D'3'
        retlw   D'2'
        retlw   D'0'
        retlw   D'1'
        retlw   D'3'
        retlw   D'3'
        retlw   D'0'
        retlw   D'1'
        retlw   D'3'
        retlw   D'0'
        retlw   D'2'
        retlw   D'1'
        retlw   D'3'
        retlw   D'2'
        retlw   D'1'
        retlw   D'2'
        retlw   D'2'
        retlw   D'2'
        retlw   D'3'
        retlw   D'3'
        retlw   D'1'
        retlw   D'2'
        retlw   D'0'
        retlw   D'0'
        retlw   D'0'
        retlw   D'0'
        retlw   D'3'
        retlw   D'2'
        retlw   D'1'
        retlw   D'0'
        retlw   D'0'
        retlw   D'1'
        retlw   D'1'
        retlw   D'0'
        retlw   D'2'
        retlw   D'0'
        retlw   D'2'
        retlw   D'2'
        retlw   D'2'
        retlw   D'2'
        retlw   D'3'
        retlw   D'3'
        retlw   D'0'
        retlw   D'3'
        retlw   D'0'
        retlw   D'1'
        retlw   D'3'
        retlw   D'2'
        retlw   D'2'
        retlw   D'0'
```

```
                retlw         D'3'
                retlw         D'3'
                retlw         D'2'
                retlw         D'2'
                retlw         D'2'
                retlw         D'4' ; end_of_table flag

;
; ********************************************************************************
; * band table.                                                                  *
; *                                                                                *
; * Example: Fout   = 14.0971 MHz                                                *
; *          Fclock = 180 MHz                                                    *
; *          basefreq = (14.0971*10^6) * (2^32) / (180*10^6) = 336369908   *
; *           336369908 = 14 0c 98 f4 hex                                        *
; *                                                                              *
; * Each entry is four instructions long, with each group of four literals   *
; * representing the frequency as a 32 bit integer.                          *
; *                                                                              *
; ********************************************************************************
;
band_table
        addwf   PCL,f;__________________
        retlw   0x47 ; 6 meters  MSB 0
        retlw   0x87 ;
        retlw   0xab ; band 10
        retlw   0x2a ;____________LSB__
        retlw   0x28 ; 10 meters MSB 4
        retlw   0x00 ;
        retlw   0x66 ; band 9
        retlw   0x87 ;__________________
        retlw   0x23 ; 12 meters MSB 8
        retlw   0x73 ;
        retlw   0x50 ; band 8
        retlw   0xe8 ;__________________
        retlw   0x1e ; 15 meters MSB 12
        retlw   0x00 ;
        retlw   0xdb ; band 7
        retlw   0x09 ;__________________
        retlw   0x19 ; 17 meters MSB 16
        retlw   0xc0 ;
        retlw   0x3a ; band 6
        retlw   0xd6 ;__________________
        retlw   0x14 ; 20 meters MSB 20
        retlw   0x0c ;
        retlw   0x98 ; band 5
        retlw   0xf4 ;__________________
        retlw   0x0e ; 30 meters MSB 24
        retlw   0x6b ;
        retlw   0xef ; band 4
        retlw   0x24 ;__________________
        retlw   0x0a ; 40 meters MSB 28
        retlw   0x03 ;
        retlw   0x38 ; band 3
        retlw   0xe1 ;__________________
        retlw   0x05 ; 80 meters MSB 32
        retlw   0x1c ;
        retlw   0x92 ; band 2
        retlw   0x66 ;__________________
        retlw   0x02 ; 160 meters MSB 40
        retlw   0x9d ;
        retlw   0x3b ; band 1
        retlw   0x56 ;__________________
```

```
;
; ***************************************************************************
; * main                                                                    *
; *                                                                         *
; * Purpose:  This routine retrives WSPR symbols, determines symbol frequency *
; *            offset and symbol transmit delay timing.                      *
; *                                                                         *
; *   Input:  Symbol data from symbol table                                 *
; *                                                                         *
; *  Output:  Subroutine calls to calc_DDSword and send_DDS-word             *
; *                                                                         *
; ***************************************************************************
;
main                                ; Symbol send code starts here
        movlw       D'0'
        movwf       symbolcount
        bsf         PORTA,TX     ; Turn on transmitter
        call    get_band
        movlw       D'30'           ; 2 sec delay before data start
        movwf       two_sec         ;

tdelay                              ;
        btfss       INTCON,T0IF  ; Did timer overflow?
        goto        tdelay          ; No, hang around some more
        bcf     INTCON,T0IF  ; reset overflow flag
        decfsz      two_sec,F       ; Count down
        goto        tdelay          ; Not time yet

symbol_loop
        movlw       D'21'
        movwf       symboltimer
        movfw       symbolcount
        call    Table
        movwf       temp
        incf    symbolcount
        sublw       D'4'         ; Test for end_of_table flag
        bz          stop            ; Yes, stop
        movfw       temp         ; No, reload w from temp and continue
        sublw       D'0'         ; Test for symbol 0
        bz          zero         ; Yes, goto zero
        movfw       temp         ; No, reload w from temp and continue
        sublw       D'1'         ; Test for symbol 1
        bz          one             ; Yes, goto one
        movfw       temp         ; No, reload w from temp and continue
        sublw       D'2'         ; Test for symbol 2
        bz          two             ; Yes, goto two
                                    ; No, it must be symbol 3
        movfw       offset_3     ; Load the offset for symbol 3
        movwf       offset          ; into offset
        call    calc_DDSword ; Calculate the word to send to the DDS
        call    send_dds_word; Transmit symbol
        goto    delay           ; Wait for 680 mSec

zero
        movfw       offset_0     ; Load the offset for symbol 0
        movwf       offset          ; into offset
        call    calc_DDSword ; Calculate the word to send to the DDS
        call    send_dds_word; Transmit symbol
        goto    delay           ; Wait for 680 mSec
one
        movfw       offset_1     ; Load the offset for symbol 1
```

```
        movwf       offset          ; into offset
        call    calc_DDSword ; Calculate the word to send to the DDS
        call    send_dds_word; Transmit symbol
        goto    delay           ; Wait for 680 mSec
two
        movfw       offset_2    ; Load the offset for symbol 2
        movwf       offset          ; into offset
        call    calc_DDSword ; Calculate the word to send to the DDS
        call    send_dds_word; Transmit symbol

delay
        btfss       INTCON,T0IF  ; Did timer overflow?
        goto        delay           ; No, hang around some more
        movlw       D'39'           ;
        movwf       pad             ; Calibrate symbol timer
timepad decfsz      pad,F           ; to 680 mSec
        goto    timepad         ;
        bcf     INTCON,T0IF  ; Reset overflow flag
        movlw       D'130'          ; Timer will count
        movwf       TMR0        ; 127 (256-129) counts
        decfsz      symboltimer,F   ; Count down
        goto        delay           ; Not time yet

        goto    symbol_loop
stop
        movlw       D'1'
        movwf       sendcode    ; End of transmit
        movwf       scan_flag       ; End of transmit for scan function
        bcf         PORTA,TX    ; Turn off tranmitter

        return


;
; ********************************************************************************
; * calc_DDSword                                                                 *
; *                                                                              *
; * Purpose:  This routine calculates the DDSword control word to be sent     *
; *            to the DDS.                                                           *
; *                                                                              *
; *   Input:  basefreq_3 ... basefreq_0, offset                                  *
; *                                                                              *
; *  Output:  DDSword_3 ... DDSword_0                                           *
; *                                                                              *
; ********************************************************************************
;

calc_DDSword
        movf        basefreq_0,w ; LSD freq byte operand
        addwf       offset,w        ; Add offset to LSDfreq byte operand
        movwf       DDSword_0       ; Store result
        movf        basefreq_1,w ; Pick up next operand
        movwf       temp3           ; Temporarily store
        btfss       STATUS,C        ; Was there a carry?
        goto    a01             ; No, skip to a01
        movlw       0x01            ; Yes, add in carry
        addwf       temp3,w     ; Add to temp

a01     movwf       DDSword_1       ; Store result
        movf        basefreq_2,w ; Pick up next operand
        movwf       temp3           ; Temporarily store
        btfss       STATUS,C        ; Was there a carry?
        goto    a02             ; No, skip to a02
```

```
        movlw       0x01            ; Yes, add in carry
        addwf       temp3,W     ; Add to temp

a02     movwf       DDSword_2       ; Store result
        movf        basefreq_3,w ; Pick MSB freq byte operand
        movwf       temp3           ; Temporarily store
        btfss       STATUS,C        ; Was there a carry?
        goto    a03             ; No, skip to a03
        movlw       0x01            ; Yes, add in carry
        addwf       temp3,w     ; Add to temp

a03     movwf       DDSword_3       ; Store result

        return

;
; ********************************************************************************
; * send_dds-word                                                                *
; *                                                                              *
; * Purpose:  This routine sends the DDSword control word to the DDS             *
; *           using a serial data transfer.                                      *
; *                                                                              *
; *   Input:  DDSword_4 ... DDSword_0                                            *
; *                                                                              *
; *  Output:  The DDS chip register is updated.                                  *
; *                                                                              *
; ********************************************************************************
;

send_dds_word
        movlw   DDSword_0       ; Point FSR at Least Significant Byte
        movwf   FSR             ;
next_byte
        movf    INDF,w          ;
        movwf   byte2send       ;
        movlw   0x08            ; Set counter to 8
        movwf   bit_count       ;
next_bit
        rrf     byte2send,f     ; Test if next bit is 1 or 0
        btfss   STATUS,C        ; Was it zero?
        goto    send0           ; Yes, send zero
        bsf     PORTB,DDS_dat   ; No, send one
        bsf     PORTB,DDS_clk   ; Toggle write clock
        bcf     PORTB,DDS_clk   ;
        goto    break           ;
send0
        bcf     PORTB,DDS_dat   ; Send zero
        bsf     PORTB,DDS_clk   ; Toggle write clock
        bcf     PORTB,DDS_clk   ;
break
        decfsz  bit_count,f     ; Has the whole byte been sent?
        goto    next_bit        ; No, keep going.
        incf    FSR,f           ; Start the next byte unless finished
        movlw   DDSword_4+1     ; Next byte (past the end)
        subwf   FSR,w           ;
        btfss   STATUS,C        ;
        goto    next_byte       ;
        bsf     PORTB,DDS_load  ; Send load signal to the AD9850/DDSword
        bcf     PORTB,DDS_load  ;
        return
                ;

;
```

```
; ***************************************************************************
; * get_band                                                                *
; *                                                                         *
; * Purpose:  This routine reads the frequency value of a band table entry  *
; *           pointed to by band and returns it in freq_3...freq_0.         *
; *                                                                         *
; *   Input:  band must contain the index of the desired band entry * 4     *
; *           (with the entries numbered from zero).                        *
; *                                                                         *
; *  Output:  The band frequency in freq.                                   *
; *                                                                         *
; ***************************************************************************
;

get_band
        movf    band,w          ; Get the index of the high byte
        call    band_table      ; Get the value into W
        movwf   basefreq_3      ; Save it in basefreq_3
        incf    band,f          ; Increment index to next byte
        movf    band,w          ; Get the index of the next byte
        call    band_table      ; Get the value into W
        movwf   basefreq_2      ; Save it in basefreq_2
        incf    band,f          ; Increment index to the next byte
        movf    band,w          ; Get the index to the next byte
        call    band_table      ; Get the value into W
        movwf   basefreq_1      ; Save it in basefreq_1
        incf    band,f          ; Increment index to the low byte
        movf    band,w          ; Get the index to the low byte
        call    band_table      ; Get the value into W
        movwf   basefreq_0      ; Save it in basefreq_0
        movlw   0x03            ; Get a constant three
        subwf   band,f          ; Restore original value of band
        return                  ; Return to the caller

;
; ***************************************************************************
; * LED_status                                                              *
; *                                                                         *
; * Purpose:  Reads the variable led_stat and outputs BCD word to ports.    *
; *           BCD ports used to indicate band number selected.               *
; *                                                                         *
; *   Input:  led_stat                                                          *
; *                                                                       *
; *                                                                         *
; *  Output:  Ports RB1,RB4,RB5, and RB6.                                   *
; *                                                                         *
; ***************************************************************************
;

LED_status
        btfss   led_stat,0
        goto $+3
        bsf     PORTB,BCD_0
        goto $+2
        bcf     PORTB,BCD_0
        btfss   led_stat,1
        goto $+3
        bsf     PORTB,BCD_1
        goto $+2
        bcf     PORTB,BCD_1
        btfss   led_stat,2
        goto $+3
        bsf     PORTB,BCD_2
```

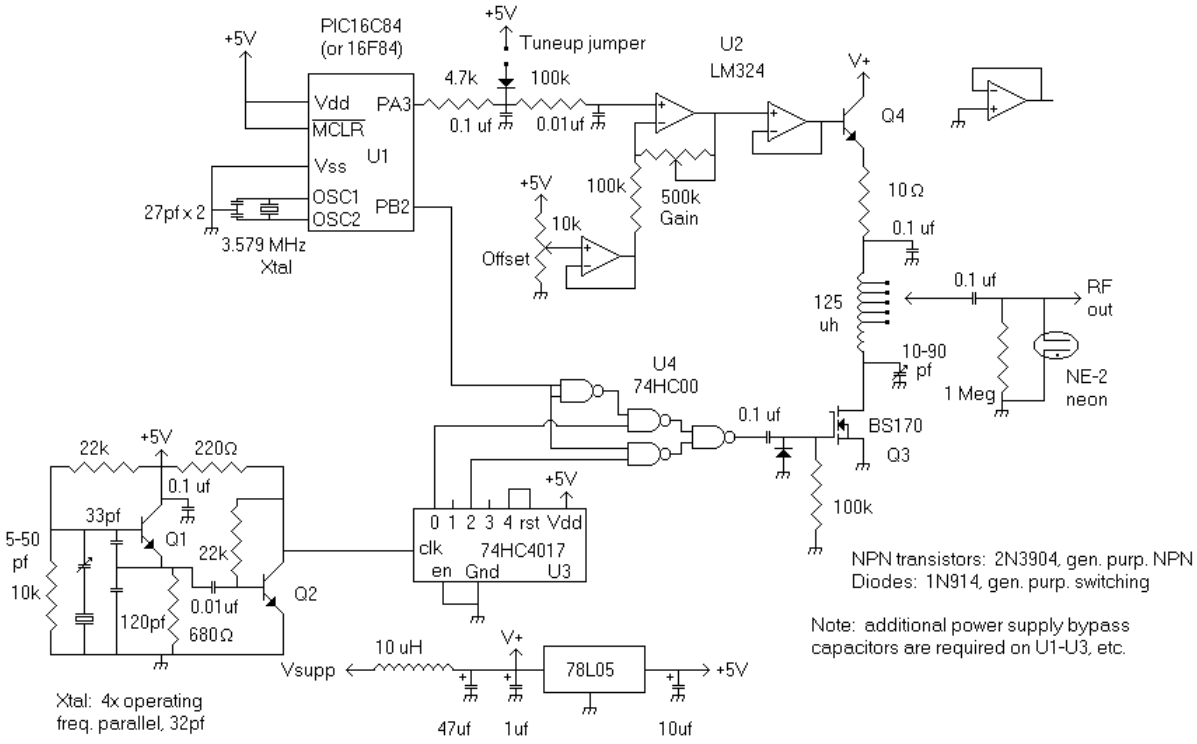
```
        goto $+2
        bcf     PORTB,BCD_2
        btfss   led_stat,3
        goto $+3
        bsf     PORTB,BCD_3
        goto $+2
        bcf     PORTB,BCD_3
        return

;_____________________________________________________________________


start

;-----------------------------------------------------------------
;   Set up timer
;-----------------------------------------------------------------
        errorlevel  -302
        banksel     INTCON
        bsf         INTCON,GIE
        bcf         INTCON,T0IE     ; Mask timer interrupt
        bsf         INTCON,INTE

        banksel     OPTION_REG
        bsf         OPTION_REG,INTEDG
        bcf         OPTION_REG,T0CS  ; Select timer
        bcf         OPTION_REG,PSA   ; Prescaler to timer
        bsf         OPTION_REG,PS2   ; \
        bsf         OPTION_REG,PS1   ;  >- 1:256 prescale
        bsf         OPTION_REG,PS0   ; /

;-----------------------------------------------------------------
;   Set up I/O
;-----------------------------------------------------------------

        banksel  TRISA
       movlw   B'11011100'       ; Tristate PORTA (all Inputs except RA0,RA1)
       movwf   TRISA             ;
        movlw    B'00000001'
       movwf   TRISB             ; Set port B to all outputs except RB0
        banksel  PORTA
        clrf PORTA
        clrf PORTB

;-----------------------------------------------------------------
;   Initialize memory
;-----------------------------------------------------------------


;       Set default basefreq to 10.1042 MHz
       movlw    D'24'             ; MSB for band (refer to band table)
        movwf       band
        movlw       D'4'          ; band number from band table
        movwf       led_stat      ; set up LED (BCD) status
        call        LED_status

;       Set DDSword_4 to turn on AD9851 6x clock multiplier
       movlw    0x01             ; Turn on 6x clock multiplier (AD9851)
       movwf    DDSword_4        ; Last byte to be sent
                                  ; Mult answer is in bytes _3 .. _0
;_____________________________________________________________________________
;
```

```
;  offset - N*12000*2^32 / 8192*Fclock
;  Example: (for symbol 2)
;           Fclock = 180 MHz
;           offset = 2*12000*2^32 / 8192*(180*10^6) = 69.905
;            69.905 ~ 70 = 46 Hex
;_____________________________________________________________________________

;       Load WSPR offsets
        movlw       0x00        ; 0.00 Hz
        movwf       offset_0
        movlw       0x23        ; 1.46 Hz
        movwf       offset_1
        movlw       0x46        ; 2.93 Hz
        movwf       offset_2
        movlw       0x69        ; 4.39 Hz
        movwf       offset_3

        movlw       D'120'          ; for 2 minute intervals
        movwf       sec_count

        movlw       D'5'
        movwf       min_count       ; for 10 minute intervals

        clrf    sendcode
        clrf    scan_flag

        btfsc       PORTA,pb_1      ; PB1 down?
        goto        WaitForInt      ; No, skip to WaitForInt

TestPB1up
        btfss       PORTA,pb_1      ; PB1 up?
        goto        TestPB1up       ; No, wait for release
        goto    start_scan


WaitForInt
        movfw       sendcode    ; Will go to transmit on the
        addlw       D'0'        ; default frequency upon reset then
        bz          transmit    ; transmit on a 10 min. interval.


; Note: pushbutton is only active when not transmitting

        btfsc       PORTA,pb_1      ; PB1 down?
        goto        WaitForInt      ; No, skip to WaitForInt

TestPB1u
        btfss       PORTA,pb_1      ; PB1 up?
        goto        TestPB1u        ; No, wait for release

PB_yes1
        bcf         STATUS,C
       movlw    0x04            ; get 4 bytes to subtract
       subwf    band,f      ; Move down to MSB in band list
        incf    led_stat    ; Increment band LED
       btfss    STATUS,C        ; Off the bottom?
       goto     reset_var      ; Yes, reset variables
        call    LED_status      ; Set up BCD outputs
        movlw       D'1'
        movwf       sendcode    ; Ensure timing integity
       goto     WaitForInt

start_scan
```

```
        movlw       D'1'
        movwf       scan_flag       ; Set scan flag
        bsf         PORTA,scan_LED  ; Turn on scan LED

loop
        movfw       scan_flag       ; Interrupt routine will
        addlw       D'0'        ; clear scan_flag at 120 sec interval
        bz          transmit2       ; Time to transmit?
        goto    loop        ; No, wait for even minute

transmit
        call    main        ; Start transmit routine
        goto        WaitForInt      ; Start over and wait for interrupts


reset_var
        movlw       D'40'           ; Set counter to end of band table
        movwf       band
        movlw       D'0'        ; Initialize LED (BCD) status
        movwf       led_stat
        goto    PB_yes1

transmit2
        bcf         STATUS,C
        movlw   0x04            ; Get 4 bytes to subtract
        subwf   band,f      ; Move down to MSB in band list
        incf    led_stat    ; Increment band LED
        btfss   STATUS,C        ; Off the bottom?
        goto    reset_var2     ; Yes, reset variables
        call    LED_status      ; Set up BCD outputs
        call    main        ; Start transmit routine
        goto        loop        ; Start over and wait for interrupt


reset_var2
        movlw   D'40'               ; Set counter to end of band table
        movwf   band
        movlw   D'0'            ; Initialize LED (BCD) status
        movwf   led_stat
        goto transmit2

        end
```

Radiolatarnia PSK31 z kombinowaną modulacją fazy i amplitudy

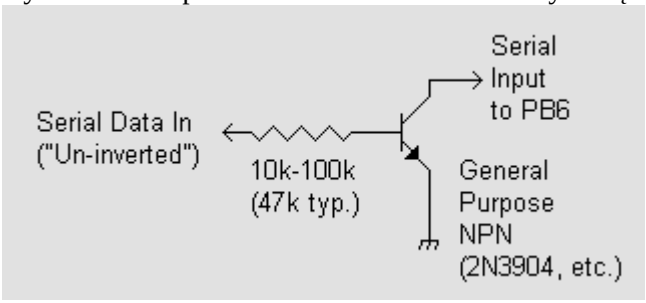
W odróżnienieniu od najczęściej spotykanych rozwiązań w układzie radiolatarni na amerykańskie pasmo eksperymentalne 1700 kHz (można go łatwo dostosować do pracy w amatorskim paśmie 160 m lub wyższych – 40, 80 m) zastosowano nieliniowy wzmacniacz mocy w klasie E a sygnał PSK31 uzyskiwany jest przez przełączanie fazy za pomocą bramki logicznej XOR w połączeniu z modulacją amplitudy sygnału wyjściowego. Modulacja amplitudy odbywa się za pomocą tranzystora szeregowego (wtórnika emiterowego) w obwodzie zasilania stopnia mocy. Analogowy sygnał modulujący amplitudę jest generowany przez mikroprocesor 16F84 poprzez modulację szerokości impulsów.



Rys. 4.10. Nadajnik z kombinowaną metodą otrzymywania sygnału PSK31

Generator sterujący pracuje na 4-krotnej częstotliwości wyjściowej nadajnika a jego sygnał podawany jest na dzielnik częstotliwości 74HC107. W zależności od poziomu logicznego na wyjściu PB3 procesora sterującego układ 74HC00 tranzystor mocy sterowany jest sygnałem w.cz. o fazie 0 lub 180°. Na czas przełączania fazy amplituda sygnału jest obniżana do 0 tak jak tego wymaga specyfikacja PSK31. Sygnał modulujący z wyjścia PA3 jest podawany przez filtr dolnoprzepustowy i wzmacniacz U2 (LM324) na bazę tranzystora Q2 włączonego w szereg z tranzystorem mocy w.cz. Q3 (BS170). Stopień mocy jest zabezpieczony przed szkodliwym wpływem wyładowań atmosferycznych i elektryczności statycznej za pomocą neonówki i włączonego równolegle do niej opornika 1 MΩ. Tranzystory Q1 i Q4 są typu 2N3904 ale można je zastąpić przez dowolne krzemowe tranzystory europejskie NPN ogólnego zastosowania.
Po zwarciu wejścia B5 do masy radiolatarnia może być sterowana z komputera PC przez złącze szeregowe procesora PB6 (szybkość transmisji wynosi 1200 bodów, 8N1). Wyjście PB0 (PTT_OUT) może być wykorzystane do włączania dodatkowego liniowego wzmacniacza mocy. Wyjście PB1 (KEY_OSC) jest zasadniczo przewidziane do włączania zasilania generatora sterującego ale lepiej aby

był on stale zasilany – uzyskuje się wówczas większą stabilność częstotliwości. Wyjście PB3 przewidziane było zasadniczo do celów diagnostycznych ale można je także wykorzystać do kluczowania częstotliwości nadajnika FSK31. Emisja ta jest jednak rzadko stosowana a do jej odbioru nadaje się jedynie MixW. Wejście PB4 należy pozostawić otwarte – służy ono do wyboru wariantu współpracującego ze starszym (nie pokazanym tutaj) rozwiązaniem nadajnika.
PB7 jest również wyjściem diagnostycznym – służy do podłączenia poprzez opornik szeregowy diody elektroluminiescencyjnej migającej w takt pulsu programu: włączenie na 32 ms następuje w momencie nadawania pierwszego znaku komunikatu, na 4 ms w trakcie oczekiwania na dane na złączu szeregowym i na 8 ms po odebraniu znaku ASCII na tym złączu.



Rys. 4.11. Sposób podłączenia złącza szeregowego

Rozkazy sterujące na złączu szeregowym:
STX (kod ASCII 02h, control-B) Kasowanie bufora w pamięci RAM i ustawienie wskaźnika adresowego na jego początek
ETX (kod ASCII 03h, control-C) Nadanie zawartości bufora a po niej komunikatu z pamięci EEPROM
SO (kod ASCII 0Eh, control-N) Powtórzenie transmisji komunikatu z bufora RAM bez nadania po nim tekstu z pamięci EEPROM
SI (kod ASCII 0Fh, control-O) Włączenie zasilania generatora(PB1)
DC1 (kod ASCII 11h, control-Q) Skopiowanie zawartości bufora RAM do pamięci EEPROM
DC2 (kod ASCII 12h, control-R) Dodanie zarawtości bufora RAM do komunikatu w pamieci EEPROM
DC3 (kod ASCII 13h, control-S) Włączenie nadawania z pamięci EEPROM w trybie pracy nadawania z pamięci RAM – właściwy tryb radiolatarni
DC4 (kod ASCII 14h, control-T) Wyłączenie transmisji z pamięci EEROM w trybie nadawania z pamięci RAM
ETB (kod ASCII 17h, control-W) Włączenie sygnalizacji pulsu
CAN (kod ASCII 18h, control-X) Wyłączenia sygnalizacji pulsu
ESC (kod ASCII 1Bh, "escape") Transmisja wyłącznie komunikatu z pamięci RAM bez dodatku z pamięci EEPROM.
Długość komunikatu w pamięci RAM procesora 16F84 może wynosić

```
/*
The following program is a pic-based PSK31 sender, Version 2.0x.

*** NOTE:  Look below for a conditional flag defining the use of a
PIC16C84 or PIC16F84.
The former has enough RAM for a 16 byte RAM buffer, whereas the
PIC16F84 can provide for
48 bytes.

The 0.05 base version had PWM analog output.  This is accomplished by
sampling the output at
32 times the bitrate (i.e. 1 KHz) with a PWM signal.  The PWM
generator in the ISR has a
total of 100 "counts."  The input variable "PWM" varies from 1 to 99
- one portion of
```

routine outputs a 1 for "PWM" counts while the other portion outputs a zero for "100-PWM"
counts. For the rest of the time this output is in hi-Z state. The R/C filter on the output
smooths the output and allows for a linear 0-5 volt output for a 1-99 input (0 and 100 are
not used.) The sine table (SINTBL) goes from 28 to 78 to keep away from the supply rails
and within the working range of the output buffer/amplifier. The "zero" output (for the
balanced modulator) occurs at approximately mid-supply.

The 0.09 base version has an input select (PB4) that, when low (grounded) selects the mode
whereby PA3 outputs a PWM version as described above. When PB4 is open (high) PA3
outputs a modulation envelope for an outboard PA. In this case, when a 0 is to be transmitted
(i.e. a phase shift) the PA3 output goes low, PB3 changes state (i.e. phase) and then PA3
goes high again. This allows the phase shift to occur when the output power is at (or very
near) zero. The phase information is taken from PB2.

It should be noted that pin PB3 carries the varicode data output. This is simply a logic level
that correlates with the current phase state. Originally a debug tool, this is also useful for
FSK31 modulation. This pin outputs the data whether in the "AM" *or* the "balanced modulator" mode.
It is worth noting that in the "AM" mode, the same data is output on pin PB2 (but slightly delayed
to account for delay in the R/C lowpass filter) for phase modulation.

Note that ALL input pins on port-B (i.e. RB4, RB5 and RB6) have weak pull-up resistors that are
internal to the PIC. These input pins are also diode-protected, so voltages above Vdd and below
Vss (ground) are clamped, as long as the input currents are limited to under 20 milliamps with a
series resistance. For "high" level states, the appropriate pin may be left floating.

THE SERIAL INPUT MODE:

Version 2.00 introduces a serial port. This is selected by grounding pin RB5. This operates
at 1200 baud, 8 bits, no parity, 1 stop bit. The serial data expected at this pin is to be
"inverted" - that is, the output of an RS-232 port may be put directly into this pin through
a series resistor. Recommended values are 2.7k for a unipolar output (i.e. voltage that goes
down to zero, but not negative) or up to 10k if the output is bipolar (i.e. like most RS-232
ports.)

NOTICE: WHEN TRANSMITTING, THE SERIAL INPUT IS DISABLED! If this is used "interactively", the
host device should monitor the "PTT" line (RB0) to determine when data is being transmitted.

NOTICE: The MSB (the eighth bit) of all incoming data is ignored and internally set to zero!

This version allows beacon text to be sent. To maintain compatibility with previous versions,
the PB5 pin is used as a "mode select" pin: When this pin is left open, it defaults to the "PLAY
from EEPROM" mode. When this pin is grounded, the "serial input mode" is active.

It is important to note that in the SERIAL INPUT mode, the MSB of the first byte in the EEPROM
(byte address 0) is used to indicate if, in RAM mode, EEPROM data is to be sent repeatedly.

There are several "commands" available in the serial input mode. These are as follows:

STX (ASCII code 02h, control-B) - Clear buffer
ETX (ASCII code 03h, control-C) - Transmit contents of buffer, followed immediately by EEPROM contents
SO (ASCII code 0Eh, control-N) - "Replay" RAM (*NOT* followed by EEPROM data)
SI (ASCII code 0Fh, control-O) - Turn on oscillator (it gets turned off when PTT is unkeyed)
DC1 (ASCII code 11h, control-Q) - Copy contents of RAM buffer into EEPROM. No other action is taken.
DC2 (ASCII code 12h, control-R) - Append contents of RAM buffer into EEPROM. No other actiion is taken.
DC3 (ASCII code 13h, control-S) - Enable EEPROM-send mode when in RAM mode
DC4 (ASCII code 14h, control-T) - Disable EEPROM-send mode when in RAM mode
ETB (ASCII code 17h, control-W) - Enable "heartbeat" flash
CAN (ASCII code 18h, control-X) - Disable "heartbeat" flash
ESC (ASCII code 1Bh, "escape") - Transmit contents of buffer only. (No EEPROM data)

It is recommended that, prior to ANY RAM buffer loading, the "STX" (clear buffer) command is sent to
assure that the RAM buffer doesn't have any "garbage" in it.

Sending a message:

First, it should be noted that this software supports ONLY ASCII codes 0-127: The 8th bit is NOT
supported. Also, the above control characters may NOT be sent as they will be intercepted and
interpreted as their control character equivalents BEFORE they are sent.

There are TWO modes whereby a message in RAM may be sent:

The first of these is uses the ETX (control-C, 03h) character. This transmits the contents of the
RAM buffer (up to 47 bytes) and follows it IMMEDIATELY with the EEPROM contents. This is useful if,
for example, you have a beacon that is sending telemetry and you are always ending the message with
unchanging information, such as a callsign.

The second mode uses the ESCape (1Bh) character. This sends the current RAM test WITHOUT following it
with the EEPROM contents.

There is one more character that will cause a transmission: This is the SO (control-N, 0Eh) character.
This will cause whatever is in RAM to be played back and is NOT followed by the EEPROM contents. This
may be useful if you want to replay the RAM contents but do NOT want re-load them. Note that this command
will play back ANYTHING that is in RAM - even garbage! It should be noted that this command works
EVEN if you sent the <STX> character to clear the RAM! (The <STX> command only resets the RAM
pointer, and doesn't erase anything.)

A note: To force a playback of EEPROM contents ONLY, the following sequence is recommended:

<STX> <ETX>

The <STX> character clears the RAM buffer and the <ETX> character plays back the RAM buffer (which contains
NOTHING at this point) followed by the EEPROM contents.

Finally, it should be noted that the <ETX> and <ESC> characters are saved in RAM - this is necessary both
to mark the end of the text and to indicate the desired sending mode. For this reason one RAM location
is required for storing this information. The "Transfer to EEPROM" command does not need this, so 48 bytes
are available for *THAT* commands... (In case you were wondering...)

Transferring data into EEPROM:

The "DC1" character (11h, control-Q) takes the current RAM buffer data and copies it into
the EEPROM, terminating the EEPROM data automatically with the FFh character needed to
signify the end of the text. Note that only as many bytes may be entered as there is room in
the RAM buffer (48 bytes) so the EEPROM may NEVER be filled to capacity with just this command.
Once this command is given, wait at least 250 milliseconds before issuing ANY other command.

A "companion" command uses the "DC2" character (12h, control-R) appends the current RAM buffer data
to what is currently in the EEPROM. The use of this command in conjunction with the "DC1" character
command allows all 64 bytes of the EEPROM to be filled.

The DC3 and DC4 control characters simply set and clear (respectively) the MSB in EEPROM location 0.
This indicates whether or not the "beacon" mode is to be enabled. When in "beacon" mode, the EEPROM
text will be repeatedly sent, pausing for a second or so between cycles to allow a "DC4" character to
be received to disable the "beacon" mode.

OPERATION OF THE SERIALLY-INITIATED "BEACON MODE" (NOT to be confused with the mode that is operated
when pin RB5 is left ungrounded...):

Sending a control-S (13h) will put the controller in the beacon mode. This is done by setting the MSB
of the first EEPROM character (byte 0). In this mode, a preamble will be sent (about 2.5 seconds of
"zeros") to allow syncronization of the receiver followed by the contents of the EEPROM, followed by
about 250 milliseconds of "zeroes". Following this is a pause of approximately 1 second where only the
PTT_OUT (RB0) line is are dropped. It is ONLY during this "window" that the "stop
EEPROM Beacon Mode" command (control-T, 14h) will be recognized. If you are using a host computer, it
is recommended that the PTT_OUT line be monitored to determine when you have "seized" control of the
processor.

It should be further noted that upon going into and out of beacon mode, the processor will experience
a forced watchdog reset, causing all I/O lines to momentarily enter a Hi-Z state.

A final note: While in the serially-initiated EEPROM "BEACON" mode the SERIAL INPUT line may be used
to control transmission. If this line is held LOW (as is the case on an RS-232 output) then the
beacon will operate normally. If this line is HIGH (or disconnected) then the beacon mode will halt
after the current message cycle is completed. This may be used to control a timed beacon

PTT and OSCILLATOR control lines:

There are two lines used for controlling power to various pieces of an attached transmitter. RB0 is
the PTT_OUT line and it goes high when a message is being sent in any mode. (I.E. in the strappable
EEPROM-Beacon mode, it is ALWAYS high.)

```
The other line is the KEY_OSC line (RB1) and it is ALWAYS high when
PTT_OUT is high, and it is ALWAYS
set to 0 when the PTT_OUT line is "unkeyed" - EXCEPT in the serially-
controlled EEPROM PLAYBACK mode.
It does, however, allow for a special command, the SI control
character (control-O, 0Fh) to set this
to a HIGH state.  This allows a host processor to turn on the
oscillator BEFORE the beginning of a
transmission so that its frequency may stabilize, but not pull full
current of the transmitter's
output stages.

The HEARTBEAT indicator.

An LED may be connected between pin RB7 and ground (with a series
resistor) for an indication of PIC
activity.  It is recommended that when used as an unattended beacon,
this LED be disconnected (by
using a push-on jumper, for example) to reduce power consumption.
This pin will be pulsed high
under the following conditions:

- During the first bit of a varicode character being transmitted (for
32 milliseconds)
- While the PIC is awaiting an input character.  It will flash
approximately once per second for about 4
milliseconds.
- While an ASCII character is being received on the serial line
(approximately 8 milliseconds.)

These flashes may be used to determine what the PIC is doing and may
be a useful troubleshooting tool.

In order to save power, the "heartbeat" LED may be disabled via
software.  This is done by sending a the
CAN control character (control-X, 18h.)  It may be re-enabled with
the ETB character (control-W, 17h.)
The HEARTBEAT is enabled by default upon powerup and it is enabled in
both the pin-strapped EEPROM-SEND
mode and the serially-initiated BEACON mode.

Copyright 1999, 2000 by Clint Turner, KA7OEI

Revision History:

0.01  - 19990905  - First working version
0.05  - 19990906  - Version with PWM analog output for sine-shaped
output
0.07  - 19990910  - Version with another PWM output mode for
modulation of an output power amp
                    (see above)
0.09  - 19991025  - PB4 (mode select) inverted (leaving PB4 open for
power-saving AM mode).  Also
                    transmits EEPROM contents
1.00  - 19991103  - First "released" version - still using R/C LPF.
```

```
2.00  - 20001008  - Work begun on version with "serial port" - semi
operational.
2.01  - 20001013  - First fully operational, mostly debugged version
2 code.
2.02  - 20001022  - Minor bug fixed which prevented some non "beacon-
control" control characters from being transmitted
2.03  - 20001025  - Added 0.75 seconds of "dead carrier" tail to
transmissions to facilitate squelch operation
2.04  - 20001107  - Changed "sine" lookup tables to greatly reduce
IMD - especially in "AM" mode
*/

// ***********   THE FOLLOWING STATEMENT DEFININES WHETHER TO USE
PIC16C84 or PIC16F84 ************
// Operational differences between the 16C84 and 16F84 are that the
former has 16 bytes of RAM and the latter has
// 48 bytes of RAM available:  This correlates with 15 and 47 bytes
of message text available, respectively.

//#define  16C84    TRUE     // this is DEFINEd if a 16C84 is to be
used, or comment it out if 16F84 is to be used

#OPT 9

#ifdef 16C84            // are we using a 16C84 or 16F84?
   #include <16c84.h>   // we use a 16C84
#else                 // we do not use 16C84
   #include   <16f84.h> // define use of 16F84
#endif

#include "varicode.h"   // include varicode table

#define       PORT_A_ADDR   0x05   // port A address
#define       PORT_A_TRIS   0x00   // port A I/O mask - all output
#define     PORT_A_TRIS_A3I 0x08 // port A I/O mask with A3 as input
(for PWM output)
#define       PORT_B_ADDR   0x06   // port B address
#define       PORT_B_TRIS   0x70   // port B I/O mask - Outputs except
B4, B5, B6
#define       CALL_LEN  16     // length of "callsign" in bytes
#define     RTCC_CNT   39     // timer setting of RTCC loop for 1 KHz
interrupt (32x bitrate)
                              // - do not adjust this value!  (error:
approx +0.3 Hz)
//
#ifdef   16C84       // set buffer size appropriate for 16C84
   #define     BUFSIZE  16       // size of RAM buffer in bytes
#else
   #define     BUFSIZE  48       // size of RAM buffer in bytes
#endif
//
#define     STX         0x02  // (^B) character indicating that
buffer should be cleared.
#define     ETX         0x03  // (^C) character indicating end of RAM
text - to be followed by EEPROM text
```

```
#define     SO          0x0E  // (^N) character indicating that
contents of RAM should be "played"
#define     SI          0x0F  // (^O) character indicating that
oscillator control pin (RB1) should be active (=1)
#define     DC1         0x11  // (^Q) character indicating that RAM
data be copied to EEPROM
#define     DC2         0x12  // (^R) character indicating that RAM
data is to be appended to EEPROM data
#define     DC3         0x13  // (^S) character indicating that
EEPROM Beacon mode should be initiated
#define     DC4         0x14  // (^T) character indicating that
EEPROM Beacon mode should be stopped
#define     ESC         0x1b  // (ESC) character indicating end of
RAM text - NOT followed by EEPROM text
#define     ETB         0x17  // (^W) character to enable "heartbeat"
(default mode)
#define     CAN         0x18  // (^X) character to disable
"heartbeat"
//
#byte     PORT_A = 5
#byte     PORT_B = 6
#fuses    XT, WDT, NOPROTECT, PUT
// standard xtal, watchdog, no code protect, Power Up Timer, no
brownout protect

#use delay(clock=3579545, RESTART_WDT) // calibrate delays for
apparent clock speed
#use  rs232(baud=1200, rcv=PIN_B6, RESTART_WDT, PARITY=N, INVERT)
//
#ROM  0x2100   =  { 10, 13,'Y','o','u','r','
','m','e','s','s','a','g','e',' ','g','o','e','s',
' ','h','e','r','e','.',' ','I','t',' ','m','a','y','
','c','o','n','t','a','i','n',' ','u','p',
' ','t','o',' ','6','4','
','c','h','a','r','a','c','t','e','r','s','.',10,13,0xff }
//

byte const  sintbl[32] =   {     // for cosine modulation of balanced
modulator
24, 24, 25, 27, 29, 30, 33, 35, 37, 40, 42, 44, 47, 49, 52, 54, 56,
59, 61, 64,
66, 68, 70, 72, 73, 75, 76, 77, 78, 79, 79, 79 };

// original waveform 28, 28, 29, 30, 31, 32, 34, 35, 37, 39, 41, 43,
46, 48, 51, 53, 55, 58, 60, 63,
//65, 67, 69, 71, 72, 74, 75, 76, 77, 78, 78, 78 };

byte const pw_sine[32] =   {     // for power modulation cosinewave
(when RB4 is left open (high))
92, 92, 89, 88, 86, 82, 78, 73, 67, 61, 53, 46, 38, 32, 22, 13, 5,
23, 32,
40, 49, 57, 62, 68, 73, 77, 82, 84, 88, 90, 91, 92};

// original waveform 98, 96, 91, 86, 79, 71, 63, 54, 45, 37, 29, 22,
17, 12, 10,  9, 10, 12, 17, 22,
//29, 37, 45, 54, 63, 71, 79, 86, 91, 96, 98, 99  };
```

```
#use   fast_io(a)    // set port A for fixed-mode of I/O direction
#use   fast_io(b)    // set port B for fixed-mode of I/O direction

#byte PORT_A = PORT_A_ADDR    // define Port A as a memory location
#byte PORT_B = PORT_B_ADDR    // define Port B as a memory location
//
#bit  TOGGLE = PORT_A_ADDR.1  // debug/test bit
#bit  PWM_OUT = PORT_A_ADDR.3 // PWM output
//
#bit  PTT_OUT = PORT_B_ADDR.0    // push-to-talk output.  1=keyed
#bit  KEY_OSC = PORT_B_ADDR.1    // oscillator keying
#bit  OUT_PHASE = PORT_B_ADDR.2  // phase shift output - used when
P.A. is modulated
#bit  OUT_BIT = PORT_B_ADDR.3    // data bit output.  May be used for
modulating a frequency-shift network
                                 // if FSK31 operation is desired.
#bit  OUT_MODE = PORT_B_ADDR.4   // mode-select input.  When high
(open) it is in the
                                 // amplitude modulator mode
#bit  SERIAL_MODE = PORT_B_ADDR.5   // serial port input enable.
When high (open) the
                                 // EEPROM data is repeatedly sent.
When low (connected to
                                 // ground) the serial input mode is
enabled.
#bit  SER_IN = PORT_B_ADDR.6     // Serial data input
#bit  HEARTBEAT = PORT_B_ADDR.7  // this may be used to flash an LED
(connected to ground) to
                                // indicate that the PIC is operating.
//
byte  c;             // general-purpose character holder and counter
byte  cnt;           // another general-purpose byte counter
byte  v_byte;        // varicode word bit holder
byte  idx;           // index of varicode
byte  v_len;         // varicode word length
byte  pwm;           // pwm value
byte  pwm_n;         // pwm work variable
byte  pwm_i;         // inverse of pwm work variable
byte  sinptr;        // pointer within sine table
byte  baud_cnt;      // baud rate counter
byte  buffer[BUFSIZE];  // RAM buffer for storing serial inputted
data
//
short send_ok;       // set to TRUE by the ISR if it is time to send
another bit
short doing_char;    // flag to indicate that a character is being
sent
short bit_wait;      // flag to indicate that a bit is waiting to be
sent
short bit_send;      // bit that contains bit that is currently being
sent (1 or 0)
short mbit_send;     // bit that contains a copy of bit_send for the
PA modulation routine
short mbit_flag;     // bit that flags when mbit_send is used
```

```
short last_bit;      // bit that contains the previous bit that was
sent
short msg_complete;  // this indicates that the RAM message being
input is complete.
short ram_complete;  // this indicates that the RAM message being
*sent* is complete.
short no_eepromdat;  // this indicates that no EEPROM data is to be
sent following RAM text
short heartbeat_enb; // this is true if heartbeat is to be enabled.
short eeprom_playback;  // this is true if in "eeprom" playback mode

// Code begins

#int_rtcc      // interrupt function for the timeout of the RTCC
timer_isr()
{
   #asm
      nop                  // just a leeetle bit more delay to make
it as close to 1KHz as possible
      nop
   #endasm
   set_rtcc(RTCC_CNT);     // refresh RTCC counter

   pwm_n = pwm;            // load current PWM value
   pwm_i = 100 - pwm_n;    // calculate inverse of pwm variable

    set_tris_a(PORT_A_TRIS);  // take out of HI-Z mode to output mode
   PWM_OUT = 1;            // set PWM output to high state charge cap
   #asm                    // do the PWM part in assembly
   norloop:
      decfsz   pwm_n,f     // spin wheels here until pwm count is
zero and charge cap
      goto     norloop
   #endasm
   PWM_OUT = 0;            // set PWM output low to discharge cap
   #asm
   invloop:
      decfsz   pwm_i,f     // spin wheels here, too - to discharge
cap
      goto     invloop
   #endasm
    set_tris_a(PORT_A_TRIS_A3I);    // go to hi-z state between ISR
cycles

   baud_cnt++;             // increment baud counter
   if(baud_cnt > 31)  {     // every 32 interrupts, a new data bit is
ready
      send_ok = 1;         // it is now ok to send a character
     baud_cnt = 0;            // reset bit counter
   }

   if(!OUT_MODE)   {  // if in the "feed the balanced modulator"
mode, do this...
      if((OUT_BIT) && (sinptr < 31)) {    // is output bit 1 AND
pointer in table NOT at top?
         sinptr++;               // increment position in sine table
```

```
         pwm = sintbl[sinptr];   // look up sine value (only if index
was changed...)
      }
      else if(sinptr > 0)  {  // if output bit is 0 AND pointer in
table is NOT at bottom
         sinptr--;            // decrement position in sine table
         pwm = sintbl[sinptr];   // look up sine value
      }
   }
   else  {     // We are in the "AM" mode
      if(!mbit_send) {        // is there supposed to be a phase
shift?
         mbit_send = 1;       // yes - clear it...
         sinptr = 0;          // reset sine pointer
      }
      pwm = pw_sine[sinptr]; // get current data point in sine table
      if(sinptr < 31)   {
         sinptr++;
         if(sinptr == 19)  {     // are we in the middle of the phase
shift?
            OUT_PHASE = !OUT_PHASE;    // yes - flip the phase when
the sine is at zero
         }

      }
   }
}

// the following function interfaces with the ISR and the
send_varichar() function and
// actually diddles the bit sending data

void bitsend(void)
{
   if(send_ok) {     // is there a bit ready to be sent?
      if(!bit_send)  {     // yes - is it a zero?
         OUT_BIT = !OUT_BIT;  // toggle send bit (This represents a
zero)
      }
      send_ok = 0;   // clear the send-bit flag
      bit_wait = 1;  // let it be known that the current bit has been
sent...
   }
}


// This function sends a "preamble" of "c+1" "zeros" (bit
transitions) to allow synchronization of the receiver.

void send_preamble(void)
{
   restart_wdt();
   for(cnt = 0; cnt < c; cnt++) {     // send some zeroes before each
EEPROM playback cycle
      bit_send = 0;
      mbit_send = 0;
```

```
      bit_wait = 0;
      while(!bit_wait)  {
         restart_wdt();
         bitsend();
      }
   }
}

// the following function sends the the character in the global
variable "c" as the varicode bitstream at
// 31.25 baud.  It uses interrupt-based timers for pacing.  Note:
This is a consolidation of several
// discrete functions from the previous version of the code.

void send_character(void)
{
byte b_count;

   c &= 0x7f;     // make sure the MSB is clear
   idx = c;
   idx += c;   // double the index - each varicode entry takes two
bytes
   v_byte = varicode[idx];          // get bits of varicode
   doing_char = 1;                  // indicate that a character is
ready to be sent
   b_count = 0;         // init bit counter
   bit_send = 1;        // init holder for current bit
   last_bit = 1;        // init holder for the last bit sent
   if(heartbeat_enb) {  // is the "heartbeat" enabled?
      HEARTBEAT = 1;          // yes - turn on "heartbeat" LED at
beginning of each character
   }
   while(doing_char) {     // hang around here until the character is
done
      restart_wdt();
      bit_wait = 0;              // reset bit indicator
      last_bit = bit_send;       // get previous bit
      bit_send = bit_test(v_byte, 7);     // get the current bit to
send
      if(!mbit_flag) {           // copy current bit if it hasn't
been done before...
         mbit_flag = 1;
         mbit_send = bit_send;   // actually copy the bit...
      }
      b_count++;        // bump bit count
      v_byte <<= 1;     // shift the current varicode byte left 1 bit
      if((!last_bit) && (!bit_send))   {  // are the past two recent
bits both zero?
         doing_char = 0;      // signal that this is the last bit of
this character
      }
      if(b_count > 7)   {     // if this is the last bit of this
word, get the next one...
         idx++;               // look at the next bit of the
         v_byte = varicode[idx];    // get the remaining bits of the
varicode
```

```
         b_count = 0;      // reset the bit counter
      }
      mbit_flag = 0;       // clear flag in preparation for next bit
      while(!bit_wait)  {
         restart_wdt();
         bitsend();        // hang around until the current bit is
sent
      }
      HEARTBEAT = 0;       // turn off "heartbeat" LED after first
bit is sent...
   }
}

// the following function copies the contents of the RAM buffer to
EEPROM starting at address zero.  Note
// that since the size of the available RAM is less than the EEPROM
size, one may NEVER fill the EEPROM
// using this method.
// the global variable "c" should indicate where copy should *start*
and the global variable "cnt" should
// indicate where copy should *end*.
// since 'pwm_n' and 'pwm_i' are used ONLY in the interrupt routine -
and since the interrupt is disabled, we can
// use them here.

void copy_to_eeprom(void)
{
   c = 0;
   while(c < cnt) {
      restart_wdt();
      pwm_n = c + pwm_i;      // calculated offset address (if any,
offset will be in 'pwm_i')
      write_eeprom(pwm_n, buffer[c]);    // write EEPROM data *plus*
offset address
      c++;
   }
   restart_wdt();
   pwm_n++;          // a 0xff goes one character past where we ended
   if(pwm_n < 64) {     // unless we go past end of EEPROM
      write_eeprom(pwm_n, 0xff);
      restart_wdt();
   }
}

// the following function finds the end of the current EEPROM data
and appends to it what is in the RAM.

void append_to_eeprom(void)
{
   pwm_i = 0;         // find the ending of the EEPROM data
   while((pwm_i < 64) && (read_eeprom(pwm_i) != 0xff))  {     // look
for a 0xff, marking end of EEPROM
      restart_wdt();
      pwm_i++;        // set pointer to it
   }
   if(pwm_i + cnt > 63) {
```

```
      cnt = 64;      // set to indicate "end" of EEPROM (plus one...)
   }
   // pwm_i contains position where current EEPROM data ends and
where appending is to begin
   copy_to_eeprom();  // copy to EEPROM beginning at this location.
}

// the following function sends the contents of the EEPROM - until it
hits the last character in the
// EEPROM *OR* the end-of-EEPROM character (i.e. 0xff)

void do_eeprom_send(void)
{
   cnt = 0;                // initialize EEPROM send count
   c = read_eeprom(cnt);   // get first character to send...
   while((c != 0xff) && (cnt < 64))  {        // was this the last
character?
      c = read_eeprom(cnt);
      send_character();
      cnt++;
      c = read_eeprom(cnt);   // get new character...
   }
}

// the following function reads the EEPROM data at address zero and
puts it in c

void read_eeprom_zero(void)
{
   c = read_eeprom(0);
}

// the following function puts what is in 'c' into EEPROM address
zero.
// A few bytes of ROM are saved if we define a function to do this
task...

void write_eeprom_zero(void)
{
   write_eeprom(0, c);
}

// this function simply puts the character in 'c' into the RAM buffer
// at location [cnt] and increments the [cnt] pointer by one.

void put_in_buffer(void)
{
   buffer[cnt] = c;
   cnt++;
}

// the following function keys transmit lines

void key_tx(void)
{
   PTT_OUT = 1;
```

```
   KEY_OSC = 1;
}

// the following function turns off the interrupts, PTT lines as
appropriate for the mode, and appends
// a "dead carrier" tail.

void tx_end(void)
{
   disable_interrupts(GLOBAL);   // turn off interrupts again
   delay_ms(250); // give 3/4 second of dead carrier after
transmission
   delay_ms(250);
   delay_ms(250);
   PTT_OUT = 0;   // turn off transmitter...
   if(!eeprom_playback) {
      KEY_OSC = 0;      // we never shut off oscillator when in
serial EEPROM Beacon mode
   }
}

void main(void)
{
    set_rtcc(RTCC_CNT);  // initialize the rtcc
    setup_counters(RTCC_INTERNAL, RTCC_DIV_4);   // set up to use
internal clock
    enable_interrupts(RTCC_ZERO);  // setup for interrupt on an RTCC
count hitting zero
   //  enable_interrupts(GLOBAL);  // we don't turn the interrupt on
during serial mode operations
    PORT_B_PULLUPS(TRUE);
//  PORT_A = 0;   // init output bits
//  PORT_B = 0;
    SET_TRIS_A(PORT_A_TRIS);// set I/O direction for ports
    SET_TRIS_B(PORT_B_TRIS);
   //
   send_ok = 0;         // initialize various flags and counters
   msg_complete = 0;
   ram_complete = 0;
   heartbeat_enb = 1;   // "heartbeat" is enabled by default
   doing_char = 0;
//   bit_wait = 0;
//   bit_send = 0;
   sinptr = 15;       // place the sine table pointer in the middle
   last_bit = 0;
   pwm = 50;         // init the PWM algorithm

//
   while(TRUE) {

      if(!SERIAL_MODE) {   // is "mode select" input high or low?
         read_eeprom_zero();
         if(c & 0x80)  {     // is MSB of first byte in EEPROM set?
            eeprom_playback = 1;    // yes - it is in the "eeprom"
playback mode
         }
```

```
         else  {
            eeprom_playback = 0;   // no, it is "normal" serial mode.
         }
         restart_wdt(); // it is LOW, therefore it is in serial mode.
         tx_end();
         cnt = 0;       // reset receive character count
         msg_complete = 0;    // reset "message complete" flag
         ram_complete = 0;
         no_eepromdat = 0;
         while(!msg_complete)   {  // we wait here for a message to
be received via the serial port
            pwm_i = 1;        // use pwm_i and pwm_n as timers, as
interupt isn't using them now...
            pwm_n = 0;
            while(!kbhit())  {
               restart_wdt();
               pwm_i++;
               if(pwm_i == 0) {
                  pwm_n++;
                  pwm_i++;
               }
               if(pwm_n == 254) {
                  if(heartbeat_enb) {
                     HEARTBEAT = 1;    // turn on Heartbeat LED, and
then...
                  }
                  if(eeprom_playback)  {     // if this times out AND
we are in EEPROM playback mode, play EEPROM message
                     HEARTBEAT = 0;    // make sure heartbeat LED is
off...
                     key_tx();      // key transmitter
                     enable_interrupts(GLOBAL); // now, we actually
turn on the interrupts...
                     c = 80;
                     send_preamble();       // warble a preamble
                     cnt = 0;             // start at beginning of
message
                     do_eeprom_send();
                     tx_end();         // end transmission - note
that KEY_OSC isn't affected in EEPROM_BEACON mode
                     pwm_i = 1;      // reinitialize timer variables
                     pwm_n = 0;
                  }
               }
               else if(pwm_n == 255) {
                  HEARTBEAT = 0;    // turn off after 1/255th of loop
delay duration
               }
            }
            if(heartbeat_enb) {
               HEARTBEAT = 1;    // if there *is* a character, flash
the LED
            }
            c = getch();      // get current character on serial port
            HEARTBEAT = 0;    // (turn off LED after receiving
character)
```

```
            c &= 0x7f;        // mask off the MSB
            if((c < ' ') && (c != ETX) && (c != ESC)) {     // is
what follows a control character?
               if(c == STX)   {  // is this the "Clear RAM buffer"
character?
                  cnt = 0;       // yes - reset the counter
               }
               else if(c == DC1)  { // is this the "copy to EEPROM"
command?
                  pwm_i = 0;        // preset "offset" for zero
                  copy_to_eeprom();    // yes - do the copy...
                  cnt = 0;
               }
               else if(c == DC2) {  // is this the "append to EEPROM"
command?
                  append_to_eeprom();  // yes - do the append...
                  cnt = 0;
               }
               else if(c == DC3)   {   // is this the "turn on beacon
mode" character?
                  read_eeprom_zero();  // yes - get the first EEPROM
byte
                  c |= 0x80;           // set the MSB of this byte
                  write_eeprom_zero(); // put it back into the EEPROM
                  while(TRUE);  // stall here, forcing a WDT reset...
               }
               else if(c == DC4)    {  // is this the "turn off
beacon mode" character?
                  read_eeprom_zero();  // yes - get the first EEPROM
byte
                  c &= 0x7f;           // clear the MSB
                  write_eeprom_zero(); // put it back...
                  while(TRUE);  // stall here, forcing a WDT reset...
               }
               else if(c == SO)     {  // is this the "play RAM
again" character?
                  msg_complete = 1;    // yes - make it play the RAM
again - w/out EEPROM
                  no_eepromdat = 1;
                  cnt = 0;
               }
               else if(c == SI)  {     // turn on oscillator?
                  KEY_OSC = 1;         // yes - turn on the
oscillator
               }
               else if(c == ETB)    {  // is this the "enable
heartbeat" command?
                  heartbeat_enb = 1;   // yes - enable heartbeat
               }
               else if(c == CAN)    {  // is this the "disable
heartbeat" command?
                  heartbeat_enb = 0;   // yes - disable heartbeat
               }
               else  {           // if it was ***NOT*** one of the
above control characters, we simply pass it through
```

```
                  put_in_buffer();     // put this in the RAM buffer
at location [cnt]
               }
            }
            else  {           // not a control character...
                  put_in_buffer();     // put this in the RAM buffer
at location [cnt]
            }
            if(cnt > BUFSIZE) {  // are we at the end of the buffer
space?
               cnt--;         // yes - move the buffer back.  Allow
overwriting of the last character
            }
            if(c == ETX)  {     // Control character to send w/EEPROM
text appended?
               msg_complete = 1;    // yes - indicate that we are now
ready to send a message
            }
            else if(c == ESC)  {  // Control character to send w/out
EEPROM text?
               msg_complete = 1;
               no_eepromdat = 1;
            }
         }
         if(msg_complete)  {
            key_tx();      // key transmitter
            enable_interrupts(GLOBAL);   // now, we actually turn on
the interrupts...
            c = 160;                 // set for 5 seconds of "zeroes"
            send_preamble();        // send preamble to allow
receiver sync
            cnt = 0;                // initialize buffer index
            c = 32;                 // load *something* (like a
space...) in the send register to begin with...
            while(!ram_complete)  {    // was this the last character
or end of RAM buffer?
               c = buffer[cnt];
               cnt++;
               if((c == ETX) || (c == ESC) || (cnt > BUFSIZE)) {  //
are we at the end of the RAM buffer?
                  ram_complete = 1;    // yes - set the flag
               }
               else  {  // no, we aren't at the end.  Continue...
                  send_character();
               }
            }
            if(ram_complete && !no_eepromdat)  {      // send at end
of message *if* eeprom data is to be sent.
               do_eeprom_send();    // send contents of EEPROM
            }
            if(ram_complete)  {    // once the message(s) are done...
               msg_complete = 0;    // we are done - clear flag
            }
         }
      }
```

```
      else {    // do the EEPROM send if we are in the hardware
"EEPROM Send" mode
         enable_interrupts(GLOBAL);   // actually turn on interrupts
         key_tx();   // make sure we are keying tx
         c = 80;                 // set for 2.5 seconds of preamble
(zeroes)
         send_preamble();
         do_eeprom_send();       // send contents of EEPROM
      }
   }
}
 !no_eepromdat)  {      // send at end of message *if* eeprom data is
to be sent.
               do_eeprom_send();    // send contents of EEPROM
            }
            if(ram_complete)  {    // once the message(s) are done...
               msg_complete = 0;    // we are done - clear flag
            }
         }
      }
      else {    // do the EEPROM send if we are in the hardware
"EEPROM Send" mode
         enable_interrupts(GLOBAL);   // actually turn on interrupts
         key_tx();   // make sure we are keying tx
         c = 80;          // set for 2.5 seconds of preamble (zeroes)
         send_preamble();
         do_eeprom_send();       // send contents of EEPROM
      }
   }
}
```

Do skompilowania programu konieczny jest oprócz jego głównego pliku przytoczony poniżej plik pomocniczy varicode.h.

```
/* The following table defines the PKS31 varicode.  There are 128
entries,
corresponding to ASCII characters 0-127 with two bytes for each
entry.  The bits
for the varicode are to be shifted out MSB-first for both bytes, with
the first byte
in the table being the first one to be sent.

The leading zeroes in the first byte should be ignored (i.e. shift
past them to the
first 1).  More than one zero in sequence signifies the end of the
character (i.e.
two zeroes are the intercharacter sequence, so at least two zeroes
should always be
sent before the next character is sent.

For modulation, a 0 represents a phase reversal while a 1 represents
a steady-state
carrier.
```

```
This file is constructed with information from the article "PSK31
Fundamentals"
by Peter Martinez, G3PLX by Clint Turner, KA7OEI
*/
byte const varicode[256] =  {
0b10101010,
0b11000000, // 0 NUL
0b10110110,
0b11000000, // 1 SOH
0b10111011,
0b01000000, // 2 STX
0b11011101,
0b11000000, // 3 ETX
0b10111010,
0b11000000, // 4 EOT
0b11010111,
0b11000000, // 5 ENQ
0b10111011,
0b11000000, // 6 ACK
0b10111111,
0b01000000, // 7 BEL
0b10111111,
0b11000000, // 8 BS
0b11101111,
0b00000000, // 9 HT
0b11101000,
0b00000000, // 10 LF
0b11011011,
0b11000000, // 11 VT
0b10110111,
0b01000000, // 12 FF
0b11111000,
0b00000000, // 13 CR
0b11011101,
0b01000000, // 14 SO
0b11101010,
0b11000000, // 15 SI
0b10111101,
0b11000000, // 16 DLE
0b10111101,
0b01000000, // 17 DC1
0b11101011,
0b01000000, // 18 DC2
0b11101011,
0b11000000, // 19 DC3
0b11010110,
0b11000000, // 20 DC4
0b11011010,
0b11000000, // 21 NAK
0b11011011,
0b01000000, // 22 SYN
0b11010101,
0b11000000, // 23 ETB
0b11011110,
0b11000000, // 24 CAN
0b11011111,
```

```
0b01000000, // 25 EM
0b11101101,
0b11000000, // 26 SUB
0b11010101,
0b01000000, // 27 ESC
0b11010111,
0b01000000, // 28 FS
0b11101110,
0b11000000, // 29 GS
0b10111110,
0b11000000, // 30 RS
0b11011111,
0b11000000, // 31 US
0b10000000,
0b00000000, // 32 SP
0b11111111,
0b10000000, // 33 !
0b10101111,
0b10000000, // 34 "
0b11111010,
0b10000000, // 35 #
0b11101101,
0b10000000, // 36 $
0b10110101,
0b01000000, // 37 %
0b10101110,
0b11000000, // 38 &amp;amp;
0b10111111,
0b10000000, // 39 '
0b11111011,
0b00000000, // 40 (
0b11110111,
0b00000000, // 41 )
0b10110111,
0b10000000, // 42 *
0b11101111,
0b10000000, // 43 +
0b11101010,
0b00000000, // 44 ,
0b11010100,
0b00000000, // 45 -
0b10101110,
0b00000000, // 46 .
0b11010111,
0b10000000, // 47 /
0b10110111,
0b00000000, // 48 0
0b10111101,
0b00000000, // 49 1
0b11101101,
0b00000000, // 50 2
0b11111111,
0b00000000, // 51 3
0b10111011,
0b10000000, // 52 4
0b10101101,
```

```
0b10000000, // 53 5
0b10110101,
0b10000000, // 54 6
0b11010110,
0b10000000, // 55 7
0b11010101,
0b10000000, // 56 8
0b11011011,
0b10000000, // 57 9
0b11110101,
0b00000000, // 58 :
0b11011110,
0b10000000, // 59 ;
0b11110110,
0b10000000, // 60 &lt;
0b10101010,
0b00000000, // 61 =
0b11101011,
0b10000000, // 62 &gt;
0b10101011,
0b11000000, // 63 ?
0b10101111,
0b01000000, // 64 @
0b11111010,
0b00000000, // 65 A
0b11101011,
0b00000000, // 66 B
0b10101101,
0b00000000, // 67 C
0b10110101,
0b00000000, // 68 D
0b11101110,
0b00000000, // 69 E
0b11011011,
0b00000000, // 70 F
0b11111101,
0b00000000, // 71 G
0b10101010,
0b10000000, // 72 H
0b11111110,
0b00000000, // 73 I
0b11111110,
0b10000000, // 74 J
0b10111110,
0b10000000, // 75 K
0b11010111,
0b00000000, // 76 L
0b10111011,
0b00000000, // 77 M
0b11011101,
0b00000000, // 78 N
0b10101011,
0b00000000, // 79 O
0b11010101,
0b00000000, // 80 P
0b11101110,
```

```
0b10000000, // 81 Q
0b10101111,
0b00000000, // 82 R
0b11011110,
0b00000000, // 83 S
0b11011010,
0b00000000, // 84 T
0b10101011,
0b10000000, // 85 U
0b11011010,
0b10000000, // 86 V
0b10101110,
0b10000000, // 87 W
0b10111010,
0b10000000, // 88 X
0b10111101,
0b10000000, // 89 Y
0b10101011,
0b01000000, // 90 Z
0b11111011,
0b10000000, // 91 [
0b11110111,
0b10000000, // 92 \
0b11111101,
0b10000000, // 93 ]
0b10101111,
0b11000000, // 94 ^
0b10110110,
0b10000000, // 95 _ (underline)
0b10110111,
0b11000000, // 96 `
0b10110000,
0b00000000, // 97 a
0b10111110,
0b00000000, // 98 b
0b10111100,
0b00000000, // 99 c
0b10110100,
0b00000000, // 100 d
0b11000000,
0b00000000, // 101 e
0b11110100,
0b00000000, // 102 f
0b10110110,
0b00000000, // 103 g
0b10101100,
0b00000000, // 104 h
0b11010000,
0b00000000, // 105 i
0b11110101,
0b10000000, // 106 j
0b10111111,
0b00000000, // 107 k
0b11011000,
0b00000000, // 108 l
0b11101100,
```

```
0b00000000, // 109 m
0b11110000,
0b00000000, // 110 n
0b11100000,
0b00000000, // 111 o
0b11111100,
0b00000000, // 112 p
0b11011111,
0b10000000, // 113 q
0b10101000,
0b00000000, // 114 r
0b10111000,
0b00000000, // 115 s
0b10100000,
0b00000000, // 116 t
0b11011100,
0b00000000, // 117 u
0b11110110,
0b00000000, // 118 v
0b11010110,
0b00000000, // 119 w
0b11011111,
0b00000000, // 120 x
0b10111010,
0b00000000, // 121 y
0b11101010,
0b10000000, // 122 z
0b10101101,
0b11000000, // 123 {
0b11011101,
0b10000000, // 124 |
0b10101101,
0b01000000, // 125 }
0b10110101,
0b11000000, // 126 ~
0b11101101,
0b01000000, // 127 (del)
};
```

Arduino

Radiolatarnia Hella

Program (dla Arduino Duemillanove lub Arduino Uno) służy do nadawania komunikatów radiolatarni w standardzie *Feldhell* i został opracowany przez amerykańskiego krótkofalowca K6HX. Jest on wprawdzie stosunkowo prosty ale można go też łatwo uzupełnić o transmisję znaku wywoławczego telegrafią lub o wybór tekstów czy szybkości transmisji w zależności od poziomu logicznego na dowolnie wybranych wejściach lub napięcia na jednym z wejść analogowych. W odróżnieniu od często spotykanych rozwiązań amatorskich korzysta on z podziału znaku na 7 x 14 elementów a nie z uproszczonego 7 x 7.
Czas trwania połowicznego elementu wynosi 4,08 ms dlatego też autor programu po odliczeniu czasu aktywnej pracy programu dobrał opóźnienie równe 4,05 ms. Nominalna szybkość transmisji wynosi 245 bodów ale po uwzględnieniu w definicji alfabetu reguły dwóch punktów jej rzeczywista wartość równa się standardowym 122,5 bodom.
Sygnał kluczujący nadajnik jest dostępny na nóżce 13 (1 linia programu) i ma dodatnią polaryzację (napięcie wysokie, jedynka logiczna odpowiada ciemnym elementom znaku). Jego polaryzację można łatwo odwrócić w programie bądź zastosować w nadajniku podany powyżej układ z tranzystorem odwracającym polaryzację. W pętli głównej należy wprowadzić własny znak wywoławczy i dostosować tekst do bieżących potrzeb.

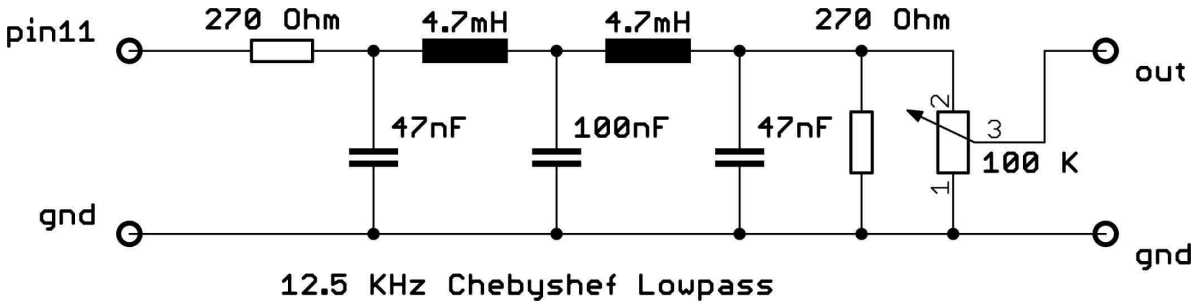
```
int radioPin = 13 ;
typedef struct glyph {
char ch ;
word col[7] ;
} Glyph ;
Glyph glyphtab[] PROGMEM = {
{' ', {0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'A', {0x07fc, 0x0e60, 0x0c60, 0x0e60, 0x07fc, 0x0000, 0x0000}},
{'B', {0x0c0c, 0x0ffc, 0x0ccc, 0x0ccc, 0x0738, 0x0000, 0x0000}},
{'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}},
} ;
#define NGLYPHS (sizeof(glyphtab)/sizeof(glyphtab[0]))
void
encodechar(int ch)
{
int i, x, y, fch ;
word fbits ;
/* It looks sloppy to continue searching even after you've
* found the letter you are looking for, but it makes the
* timing more deterministic, which will make tuning the
* exact timing a bit simpler.
*/
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(9600) ;
pinMode(radioPin, OUTPUT) ;
}
void
loop()
{
encode("K6HX QTH CM87UX TMP 72F PWR 500 MICROWATTS ") ;
}
```

Radiolatarnia WSPR

Program autorstwa Martina Nawratha DH3JO dla Arduino Diecimila albo Duemilanove generuje sygnał m.cz. służący do wysterowania nadajnika SSB. Synchronizację czasu zapewnia odbiornik DCF-77 (np. firmy Conrad) podłączony do wejścia 7. Na podstawie impulsów otrzymanych z odbiornika program dekoduje czas i rozpoczyna transmisje w zadanych momentach.
Wyjście 3 jest wyjściem sygna łu m.cz. a 10 służy do kluczowania nadajnika.
Sygnał wyjściowy m.cz. Arduino jest generowany przy użyciu modulacji szerokości impulsów i dla odfiltrowania składowej taktu wymagane jest użycie filtru dolnoprzepustowego (schemat został wzięty z innego podobnego rozwiązania dla Arduino dlatego też podana zostało na nim wyjście 11 a nie 3 jak w programie WSPR).



Rys. 4.12. Filtr dolnoprzepustowy m.cz.

```
/*
 * WSPR beacon for weak signal  radio transmission
 * encodes a WSPR message from call locator and power to a symbol/tone
 * and generates sinewaves with a PWM and software DDS
 * Thanks to Andy Talbot for his article The WSPR Coding Process
 * generates from a Callsign, Locator, and Power Level a symbol Table
 * due to the WSPR protocol
 * rund on a Arduino Diecimila or Duemilanove board / Arduino 17
 *
 * ptt connected to pin10
 * vfo for tone modulation connected to pin 11
 * tone output PWM connected to pin 3
 * DCF-time receiver connected to pin 7
```

```
 * soft DDS with ATMEGS 168
 * Timer2 set to 31373 KHz for interrupt and DDS timebase
 *
 *
 *
 * korekta OE1KDA - tryb 1 a nie 3 PWM

 * KHM 2009 /  Martin Nawrath
 * Kunsthochschule fuer Medien Koeln
 * Academy of Media Arts Cologne

 */

#include <avr/pgmspace.h>

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


//! Macro that clears all Timer/Counter1 interrupt flags.
#define CLEAR_ALL_TIMER1_INT_FLAGS    (TIFR1 = TIFR1)


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
};

// table of 256 sine values / one sine period / stored in flash memory
PROGMEM  prog_uchar sine256[]  = {

127,130,133,136,139,143,146,149,152,155,158,161,164,167,170,173,176,178,181,184,187,190,192,19
5,198,200,203,205,208,210,212,215,217,219,221,223,225,227,229,231,233,234,236,238,239,240,

242,243,244,245,247,248,249,249,250,251,252,252,253,253,253,254,254,254,254,254,254,254,253,25
3,253,252,252,251,250,249,249,248,247,245,244,243,242,240,239,238,236,234,233,231,229,227,225,2
23,

221,219,217,215,212,210,208,205,203,200,198,195,192,190,187,184,181,178,176,173,170,167,164,16
1,158,155,152,149,146,143,139,136,133,130,127,124,121,118,115,111,108,105,102,99,96,93,90,87,84,
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 }; // DDS freq table for 1497,8 1499,3 1500,7 1502,2 WSPR tones
```

```
int ledPin = 13;                // LED pin 13
int led2Pin = 8;                // LED pin 13
int t1Pin = 4;            // test pin interrupt
int t2Pin = 5;            // test pin main
int pttPin=10;
int dcfPin=7;
int state=0;


// byte sine[258];           // sinewave Memory Array 8-Bit
byte c[11];               // encoded message
byte sym[170];            // symbol table 162
byte symt[170];           // symbol table temp
char call[] = "OE1KDA";    // default values
char locator[] = "JN88";
byte power = 37;
byte dcfPin_a;
int dcfcnt;
int dcfmin;
int dcfhour;
byte n;
byte f_led;


unsigned long n1;    // encoded callsign
unsigned long m1;    // encodes locator

char cnt1;
unsigned char cc1;
int ii,bb;

// interrupt service variables
volatile boolean f_tone;      // set 1,46 Hz tonespacing time flag
volatile unsigned int tcnt;   // tonespacing timer
volatile unsigned long cnt16u;   // 16us timer
volatile byte icnt;
volatile byte icnt2;
volatile unsigned long phaccu;  // soft DDS phase accu
volatile unsigned long mm;      // soft DDS frequency word
volatile unsigned int sycnt;
volatile unsigned int cpin;


void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
  pinMode(led2Pin, OUTPUT);
  pinMode(t1Pin, OUTPUT);
  pinMode(t2Pin, OUTPUT);

  pinMode(pttPin, OUTPUT);
  pinMode(dcfPin, INPUT);
```

```
 pinMode(11, OUTPUT);   // PWM VFO Control
 pinMode(3, OUTPUT);    // PWM WSPR tone output
 Serial.begin(115200);       // connect to the serial port
 Serial.println("WSPR beacon");


 Setup_timer2();


 //cli();                      // disable interrupts to avoid distortion
 //  cbi (TIMSK0,TOIE0);             // disable Timer0 !!! arduino delay function is off now
 sbi (TIMSK2,TOIE2);             // enable Timer2 Interrupt

 OCR2B=64;
 OCR2A=64;

 mm=402063;


 encode_call();
 Serial.print("call: ");
 Serial.print(call);
 Serial.print(" ");
 Serial.print(n1,HEX);
 Serial.println(" ");


 encode_locator();
 Serial.print("locator: ");
 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);
   Serial.print(".");
   if ( (bb+1) %32 == 0) Serial.println("");
 }
 Serial.println("");

 interleave_sync();

 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 ){ // every 10 minutes for 4 minutes send
       Serial.println("Wspr Start");
       digitalWrite(pttPin,1);
       tcnt=0;
       sycnt =0;
       state=5;
     }

     if ((ii >= 0) &&  (ii-4) % 10 ==0 ){ // every 10 minutes 6 for 1 minute without ptt
       Serial.println("Wspr Start without ptt");
       tcnt=0;
       sycnt =0;
       state=5;
     }

     if (f_tone==1) {
       f_tone=0;
       switch (state){
       case 0:
         break;

       case 5:  // Wspr Start with some delay
         if (sycnt==6) {
           sycnt=0;
           state=10;

         }

         break;

       case 10: // Wspr Run
```

```
      bb=sym[sycnt];
      mm=mt[bb];
      OCR2A=240-(40*bb);  // output symbol as a control voltage to vco

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

      if (sycnt >= 162) state = 11;
      break;
    case 11: // Wspr Stop

      digitalWrite(pttPin,0);
      break;
    }
  }


 }
} // loop
//**************************************************************************
// timer2 setup
// set prscaler to 1, PWM mode to phase correct PWM,  16000000/510 = 31372.55 Hz clock
void Setup_timer2() {

// Timer2 Clock Prescaler to : 1
 sbi (TCCR2B, CS20);
 cbi (TCCR2B, CS21);
 cbi (TCCR2B, CS22);

 // Timer2 PWM Mode set to Phase Correct PWM
 cbi (TCCR2A, COM2A0);  // clear Compare Match
 sbi (TCCR2A, COM2A1);

 cbi (TCCR2A, COM2B0);  // clear Compare Match
 sbi (TCCR2A, COM2B1);

 sbi (TCCR2A, WGM20);  // Mode 1  / phase coherent PWM
 cbi (TCCR2A, WGM21);
 cbi (TCCR2B, WGM22);


}

//**************************************************************************
int dcf_minute() {
 int bb;
 int bb2,bb3;
 int rr=-1;
 bb= digitalRead(dcfPin);
 if (bb == 1 && dcfPin_a==0) {
```

```
   cpin=0;
   digitalWrite(ledPin,f_led);
   digitalWrite(led2Pin,f_led);
   f_led= !f_led;
  }
  if (bb == 0 && dcfPin_a==1) {
   bb2=0;
   dcfcnt++;

   if (cpin > 30) bb2=1;

   if ( dcfcnt >  15 && dcfcnt < 29) {
     /*
    Serial.print(" S:");
     Serial.print(dcfcnt);
     Serial.print(" ");
     Serial.print(bb2);
     */
    Serial.print(".");

   }

   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;

   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:");
   Serial.print(dcfhour);
   Serial.print(":");
   Serial.println(dcfmin);
   dcfmin=0;
   dcfhour=0;

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

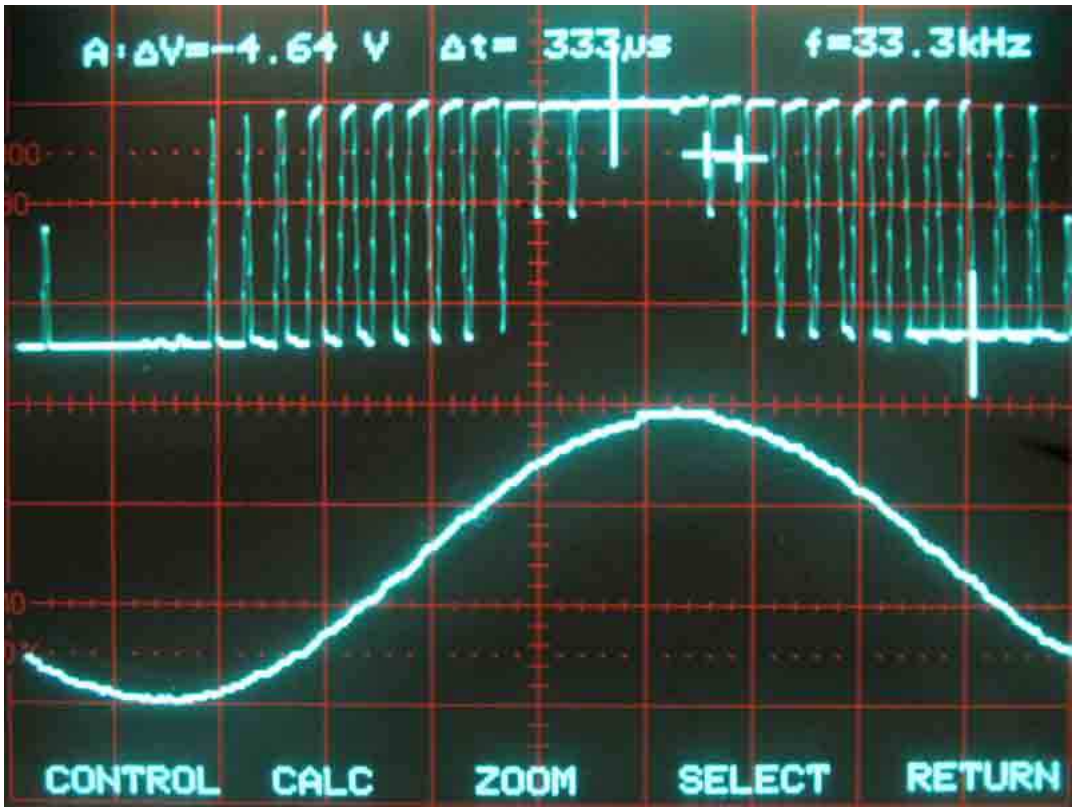
```
//*********************************************************************
void encode() {
  encode_call();
  encode_locator();
  encode_conv();
  interleave_sync();
};
//*********************************************************************
// normalize characters 0..9 A..Z Space in order 0..36
char 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(){
  unsigned long t1;

  // coding of callsign
  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;

  // merge coded callsign into message array 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(){

  unsigned long t1;
  // coding of locator
  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;

  // merge coded locator and power into message array 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;

}
//************************************************************************
// convolutional encoding of message array c[] into a 162 bit stream
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)
{
  byte po = 0;
  while(li != 0)
  {
    po++;
    li&= (li-1);
  }
  return (po & 1);
}
//************************************************************************
// interleave reorder the 162 data bits and and merge table with the sync vector
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++;
    }
  }
}


//*********************************************************************
// Timer2 Interrupt Service at 31372,550 KHz = 32uSec
// this is the timebase REFCLOCK for the DDS generator
// FOUT = (M (REFCLK)) / (2 exp 32)
// runtime : 8 microseconds ( inclusive push and pop)
ISR(TIMER2_OVF_vect) {

  sbi(PORTD,4);       // set PORTD,4 high to observe timing with a oscope
  //  cnt16u++;
  if (tcnt++ >= 21417) {
    tcnt=0;          // set tone spacing flag at 1,4648 Hz
    f_tone=1;
    sycnt++;
  }

  phaccu=phaccu+mm;   // soft DDS phase accu with 24 bits
  icnt=phaccu >> 16;  // upper 8 bits for pwm modulator
  OCR2B=sine256[icnt];     // send phase state to PWM
  if(icnt2++ == 0) cpin++;

  cbi(PORTD,4);      // set PORTD,4


}
```

Generator m.cz.

Program tego samego autora generuje w oparciu o bezpośrednią syntezę cyfrową (ang. *DDS*) sygnał m.cz. przy użyciu modulacji szerokości impulsów. Może on być po uzupełnieniu użyty jako część własnego rozwiązania dla innych rodzajów emisji albo jako generator sygnałów wzorcowych. Pracuje on na Arduino Diecimilla lub Duemillanove ale przystosowanie go do innych wariantów Arduino nie powinno przysporzyć zasadniczych trudności. Przykładem jego zastosowania jest przytoczony powyżej program radiolatarni WSPR.
Oscylogramy sygnału przed i po odfiltrowaniu za pomocą podanego powyżej filtru dolnoprzepustowego są widoczne na zdjęciu 4.13.



Fot. 4.13. Przebiegi sygnałów wyjściowych

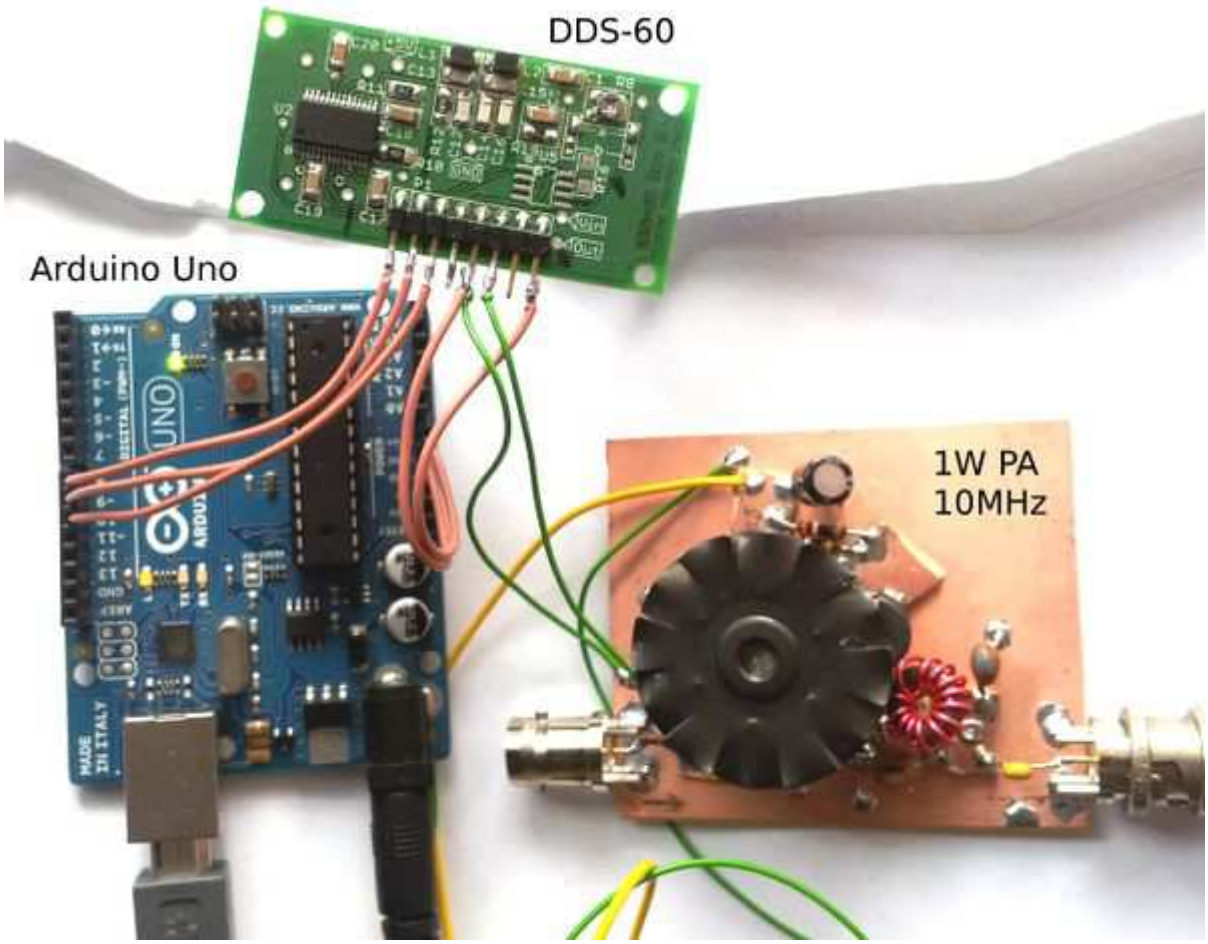
```
/*
 *
 * DDS Sine Generator mit ATMEGS 168
 * Timer2 generates the  31250 KHz Clock Interrupt
 *
 * KHM 2009 /  Martin Nawrath
 * Kunsthochschule fuer Medien Koeln
 * Academy of Media Arts Cologne

 */

#include "avr/pgmspace.h"

// table of 256 sine values / one sine period / stored in flash memory
PROGMEM  prog_uchar sine256[]  = {

   127,130,133,136,139,143,146,149,152,155,158,161,164,167,170,173,176,178,1
   81,184,187,190,192,195,198,200,203,205,208,210,212,215,217,219,221,223,22
   5,227,229,231,233,234,236,238,239,240,
```

```
   242,243,244,245,247,248,249,249,250,251,252,252,253,253,253,254,254,254,2
   54,254,254,254,253,253,253,252,252,251,250,249,249,248,247,245,244,243,24
   2,240,239,238,236,234,233,231,229,227,225,223,

   221,219,217,215,212,210,208,205,203,200,198,195,192,190,187,184,181,178,1
   76,173,170,167,164,161,158,155,152,149,146,143,139,136,133,130,127,124,12
   1,118,115,111,108,105,102,99,96,93,90,87,84,81,78,

   76,73,70,67,64,62,59,56,54,51,49,46,44,42,39,37,35,33,31,29,27,25,23,21,2
   0,18,16,15,14,12,11,10,9,7,6,5,5,4,3,2,2,1,1,1,0,0,0,0,0,0,0,1,1,1,2,2,3,
   4,5,5,6,7,9,10,11,12,14,15,16,18,20,21,23,25,27,29,31,

   33,35,37,39,42,44,46,49,51,54,56,59,62,64,67,70,73,76,78,81,84,87,90,93,9
   6,99,102,105,108,111,115,118,121,124

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

int ledPin = 13;                // LED pin 7
int testPin = 7;
int t2Pin = 6;
byte bb;

double dfreq;
// const double refclk=31372.549;  // =16MHz / 510
const double refclk=31376.6;      // measured

// variables used inside interrupt service declared as voilatile
volatile byte icnt;              // var inside interrupt
volatile byte icnt1;             // var inside interrupt
volatile byte c4ms;              // counter incremented all 4ms
volatile unsigned long phaccu;   // pahse accumulator
volatile unsigned long tword_m;  // dds tuning word m

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
  Serial.begin(115200);        // connect to the serial port
  Serial.println("DDS Test");

  pinMode(6, OUTPUT);      // sets the digital pin as output
  pinMode(7, OUTPUT);      // sets the digital pin as output
  pinMode(11, OUTPUT);     // pin11= PWM  output / frequency output

  Setup_timer2();

  // disable interrupts to avoid timing distortion
  cbi (TIMSK0,TOIE0);              // disable Timer0 !!! delay() is now not
   available
  sbi (TIMSK2,TOIE2);              // enable Timer2 Interrupt

  dfreq=1000.0;                    // initial output frequency = 1000.o Hz
  tword_m=pow(2,32)*dfreq/refclk;  // calulate DDS new tuning word

}
void loop()
{
  while(1) {
     if (c4ms > 250) {                // timer / wait fou a full second
      c4ms=0;
```

```
      dfreq=analogRead(0);             // read Poti on analog pin 0 to
   adjust output frequency from 0..1023 Hz

      cbi (TIMSK2,TOIE2);              // disble Timer2 Interrupt
      tword_m=pow(2,32)*dfreq/refclk;  // calulate DDS new tuning word
      sbi (TIMSK2,TOIE2);              // enable Timer2 Interrupt

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

   sbi(PORTD,6); // Test / set PORTD,7 high to observe timing with a scope
   cbi(PORTD,6); // Test /reset PORTD,7 high to observe timing with a scope
  }
 }
//*************************************************************
// timer2 setup
// set prscaler to 1, PWM mode to phase correct PWM,  16000000/510 =
   31372.55 Hz clock
void Setup_timer2() {

// Timer2 Clock Prescaler to : 1
  sbi (TCCR2B, CS20);
  cbi (TCCR2B, CS21);
  cbi (TCCR2B, CS22);

  // Timer2 PWM Mode set to Phase Correct PWM
  cbi (TCCR2A, COM2A0);  // clear Compare Match
  sbi (TCCR2A, COM2A1);

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

//*************************************************************
// Timer2 Interrupt Service at 31372,550 KHz = 32uSec
// this is the timebase REFCLOCK for the DDS generator
// FOUT = (M (REFCLK)) / (2 exp 32)
// runtime : 8 microseconds ( inclusive push and pop)
ISR(TIMER2_OVF_vect) {

  sbi(PORTD,7);          // Test / set PORTD,7 high to observe timing with a
   oscope

  phaccu=phaccu+tword_m; // soft DDS, phase accu with 32 bits
  icnt=phaccu >> 24;     // use upper 8 bits for phase accu as frequency
   information
                         // read value fron ROM sine table and send to PWM
   DAC
  OCR2A=pgm_read_byte_near(sine256 + icnt);

  if(icnt1++ == 125) {  // increment variable c4ms all 4 milliseconds
    c4ms++;
    icnt1=0;
   }

 cbi(PORTD,7);            // reset PORTD,7
}
```

Radiolatarnia QRSS na Arduino i AD9851

Program (opracowany przez M1GEO) daje przykład współpracy Arduino z syntezerem cyfrowym AD9851 (zawartym m.in. w module DDS-60). Komunikacja między Arduino i AD9851 (podobnie jak i z innymi pokrewnymi obwodami syntezerów odbywa się za pomocą magistrali SPI). Nadajnik pracuje na częstotliwości 10,140040 MHz i jest kluczowany w standardzie QRSS6. Wszystkie parametry pracy dają się łatwo zlokalizować w kodzie źródłowym i dopasować do potrzeb.



Fot. 4.14. Arduino z modułem DDS-60 i wzmacniaczem mocy

```
// QRSS Transmitter from AD9851
// George Smart, M1GEO.
//  here: http://www,george-smart.co.uk/wiki/Arduino_QRSS
//
// AD9851 Code
// Originally by Peter Marks
//  here: http://blog.marxy.org/2008/05/controlling-ad9851-dds-with-
arduino.html
//

// DDS Reference Oscilliator Frequency, in Hz. (Remember, the PLL).
#define DDS_REF   180000000

// DDS Offset in Hz
#define DDS_OSET  404
```

```
// QRSS Output Frequency
#define QRSS_TX  10.140040e6

// QRSS Offset Frequency
#define QRSS_OSET 5

// QRSS Timings in milliseconds
#define QRSS_DIT  6000   //5000
#define QRSS_DAH  (QRSS_DIT*3)  //18000

// Triange Ramp time in milliseconds
#define TRI_TIME  (2*QRSS_DIT)

// DDS/Arduino Connections
#define DDS_LOAD  8
#define DDS_CLOCK 9
#define DDS_DATA  10
#define LED       13
#define QRSS_TXF  (QRSS_TX+DDS_OSET)
#include <stdint.h>

void setup()
{
  // Set all pins to output states
  pinMode (DDS_DATA,  OUTPUT);
  pinMode (DDS_CLOCK, OUTPUT);
  pinMode (DDS_LOAD,  OUTPUT);
  pinMode (LED,       OUTPUT);

  // Setup RS232
  Serial.begin(9600);

  // let the radio settle a bit before we start sending data
  delay(100);
  frequency(QRSS_TXF);
  waitDit();
}

void loop()
{
  Serial.println("*** Starting Cycle");

  waitDah();

  Serial.print("Sending Morse: ");
  sendPattern("--/.----/--././---"); //M1GEO in morse.
  Serial.println(" OK");

  Serial.print("Sending 3 Triangles: ");
  // Send 3 up ramps
  for(int i=0;i<3;i++) {
    Serial.print(i+1);
    digitalWrite(LED, HIGH);
    sendUpRamp();
    digitalWrite(LED, LOW);
    sendDownRamp();
```

```
    Serial.print(" ");
  }
  Serial.println("OK");

  waitDah();
  waitDah();
  waitDah();
}

void frequency(unsigned long frequency) {
  unsigned long tuning_word = (frequency * pow(2, 32)) / DDS_REF;
  digitalWrite (DDS_LOAD, LOW); // take load pin low

  for(int i = 0; i < 32; i++) {
    if ((tuning_word & 1) == 1)
      outOne();
    else
      outZero();
    tuning_word = tuning_word >> 1;
  }
  byte_out(0x09);
  digitalWrite (DDS_LOAD, HIGH); // Take load pin high again
}

void byte_out(unsigned char byte) {
  int i;

  for (i = 0; i < 8; i++) {
    if ((byte & 1) == 1)
      outOne();
    else
      outZero();
    byte = byte >> 1;
  }
}

void outOne() {
  digitalWrite(DDS_CLOCK, LOW);
  digitalWrite(DDS_DATA, HIGH);
  digitalWrite(DDS_CLOCK, HIGH);
  digitalWrite(DDS_DATA, LOW);
}

void outZero() {
  digitalWrite(DDS_CLOCK, LOW);
  digitalWrite(DDS_DATA, LOW);
  digitalWrite(DDS_CLOCK, HIGH);
}

void sendDit() {
  frequency(QRSS_TXF + QRSS_OSET);
  waitDit();
  frequency(QRSS_TXF);
  waitDit();  // always end on a dit duration
}
```

```
void sendDah() {
  frequency(QRSS_TXF + QRSS_OSET);
  waitDah();
  frequency(QRSS_TXF);
  waitDit();  // always end on a dit duration
}

void waitDit() {
  int a = 0;
  for (a=0;a<QRSS_DIT;a+=100) {
    digitalWrite(LED, HIGH);
    delay(25);
    digitalWrite(LED, LOW);
    delay(75);
  }
}

void waitDah() {
  int a = 0;
  for (a=0;a<QRSS_DAH;a+=100) {
    digitalWrite(LED, HIGH);
    delay(75);
    digitalWrite(LED, LOW);
    delay(25);
  }
}

void sendPattern(String str) {
  int strLen = str.length();
  Serial.print(strLen);
  Serial.print(" elements: ");

  int i=0;
  for(i=0;i<strLen;i++) {
    switch (str.charAt(i)) {
      case '-':
        Serial.print("-");
        sendDah();
        break;
      case '.':
        Serial.print(".");
        sendDit();
        break;
      case '/':
      case ' ':
        Serial.print("/");
        waitDah();
        break;
      default:
        Serial.print("(I didn't recognise '");
        Serial.print(str.charAt(i));
        Serial.print("'.  Permitted Chars are -  .  or  /)");
    }
  }
}
```

```
void sendDownRamp() {
  for (int i=QRSS_OSET; i>=0;i--) {
    frequency(QRSS_TXF + i);
    delay(TRI_TIME/QRSS_OSET);
  }
}

void sendUpRamp() {
  for (int i=0; i<=QRSS_OSET;i++) {
    frequency(QRSS_TXF + i);
    delay(TRI_TIME/QRSS_OSET);
  }
}
```

Program sterujący AD9851 dla Arduino

Poniższy prosty program stanowi przykład sterowania syntezerem AD9851 przez mikrokomputer Arduino. Obrazuje on zasadę sterowania syntezerem i dzięki temu, że nie zawiera żadnych dodatkowych funkcji może być łatwo używy we własnych bardziej skomplikowanych programach. Dla innych typów syntezerów konieczne może być dopasowanie kodów sterujących w oparciu o dane katalogowe.

```
// Control a AD9851 DDS based on the good work of others including:
// Mike Bowthorpe, http://www.ladyada.net/rant/2007/02/cotw-ltc6903/ and
// http://www.geocities.com/leon_heller/dds.html
// This code by Peter Marks http://marxy.org

#define DDS_CLOCK 180000000

byte LOAD = 8;
byte CLOCK = 9;
byte DATA = 10;
byte LED = 13;

void setup()
{
  pinMode (DATA, OUTPUT); // sets pin 10 as OUPUT
  pinMode (CLOCK, OUTPUT); // sets pin 9 as OUTPUT
  pinMode (LOAD, OUTPUT); // sets pin 8 as OUTPUT
  pinMode (LED, OUTPUT);
}

void loop()
{
  // Do a frequency sweep in Hz
  for(unsigned long freq = 10000000; freq < 10001000; freq++)
  {
    sendFrequency(freq);
    delay(2);
  }
}

void sendFrequency(unsigned long frequency)
{
  unsigned long tuning_word = (frequency * pow(2, 32)) / DDS_CLOCK;
  digitalWrite (LOAD, LOW); // take load pin low

  for(int i = 0; i < 32; i++)
  {
    if ((tuning_word & 1) == 1)
      outOne();
```

```
    else
      outZero();
    tuning_word = tuning_word >> 1;
  }
  byte_out(0x09);

  digitalWrite (LOAD, HIGH); // Take load pin high again
}

void byte_out(unsigned char byte)
{
  int i;

  for (i = 0; i < 8; i++)
  {
    if ((byte & 1) == 1)
      outOne();
    else
      outZero();
    byte = byte >> 1;
  }
}

void outOne()
{
  digitalWrite(CLOCK, LOW);
  digitalWrite(DATA, HIGH);
  digitalWrite(CLOCK, HIGH);
  digitalWrite(DATA, LOW);
}

void outZero()
{
  digitalWrite(CLOCK, LOW);
  digitalWrite(DATA, LOW);
  digitalWrite(CLOCK, HIGH);
}
```

Radiolatarnia RTTY na Arduino i AD9851

Program jest dostosowany do transmisji RTTY z szybkością 45,45 boda i odstępem częstotliwości 170 Hz. Częstotliwość w.cz. wynosi 50,100 MHz. Wszystkie parametry można łatwo zidentyfikować i zmienić w kodzie źródłowym.

```
// RTTY Transmitter from AD9851
// George Smart, M1GEO.
//  here: http://www,george-smart.co.uk/wiki/Arduino_RTTY
//
// AD9851 Code
// Originally by Peter Marks
//  here: http://blog.marxy.org/2008/05/controlling-ad9851-dds-with-
arduino.html
//
// RTTY Baudot Code
// Tim Zaman
//  here: http://www.timzaman.nl/?p=138&lang=en

// DDS Reference Oscilliator Frequency, in Hz. (Remember, the PLL).
#define DDS_REF 180000000
```

```
// RTTY Output Frequency
#define RTTY_TXF  50.100e6

// RTTY Offset Frequency
#define RTTY_OSET 170

// DDS/Arduino Connections
#define DDS_LOAD  8
#define DDS_CLOCK 9
#define DDS_DATA  10
#define LED       13

#include <stdint.h>

#define ARRAY_LEN 32
#define LETTERS_SHIFT 31
#define FIGURES_SHIFT 27
#define LINEFEED 2
#define CARRRTN  8

#define is_lowercase(ch)    ((ch) >= 'a' && (ch) <= 'z')
#define is_uppercase(ch)    ((ch) >= 'A' && (ch) <= 'Z')

unsigned long time;

char letters_arr[33] = "\000E\nA SIU\rDRJNFCKTZLWHYPQOBG\000MXV\000";
char figures_arr[33] = "\0003\n-
\a87\r$4',!:(5\")2#6019?&\000./;\000";

void setup()
{
  // Set all pins to output states
  pinMode (DDS_DATA,  OUTPUT);
  pinMode (DDS_CLOCK, OUTPUT);
  pinMode (DDS_LOAD,  OUTPUT);
  pinMode (LED,       OUTPUT);

  // Setup RS232
  Serial.begin(9600);
}

void loop()
{

  // let the radio settle a bit before we start sending data
  delay(100);
  frequency(RTTY_TXF);
  delay(1000);

  time=millis();

  rtty_txstring("\r\n      \r\n");
  rtty_txstring("George Smart, M1GEO.\r\n");
  rtty_txstring("http://www.george-smart.co.uk/\r\n");
  rtty_txstring("Arduino RTTY Test!\r\n\r\n");
```

```
  rtty_txstring("\"An expert is a person who has made all the
mistakes that can be made in a very narrow field\"  Nils
Bohr\r\n\r\n");
  rtty_txstring("\"Most of the important things in the world have
been accomplished by people who have kept on trying when there seemed
to be no hope at all\"  Dale Carnegie\r\n\r\n");
  rtty_txstring("\"If others would think as hard as I did, then they
would get similar results\"  Newton\r\n\r\n");
  rtty_txstring("\"If you canâ€™t explain what you are doing to a
nine-year-old, then either you still donâ€™t understand it very well,
or itâ€™s not all that worthwhile in the first place.\"  Albert
Einstein\r\n\r\n");
  rtty_txstring("-----\r\n");


  time=millis()-time;
  Serial.println(time);

  digitalWrite (LED, HIGH);
  delay(100);
  digitalWrite (LED, LOW);
  delay(1000);

  frequency(0);
  delay(1000);

}

void frequency(unsigned long frequency) {
  unsigned long tuning_word = (frequency * pow(2, 32)) / DDS_REF;
  digitalWrite (DDS_LOAD, LOW); // take load pin low

  for(int i = 0; i < 32; i++) {
    if ((tuning_word & 1) == 1)
      outOne();
    else
      outZero();
    tuning_word = tuning_word >> 1;
  }
  byte_out(0x09);
  digitalWrite (DDS_LOAD, HIGH); // Take load pin high again
}

void byte_out(unsigned char byte) {
  int i;

  for (i = 0; i < 8; i++) {
    if ((byte & 1) == 1)
      outOne();
    else
      outZero();
    byte = byte >> 1;
  }
}

void outOne() {
```

```
  digitalWrite(DDS_CLOCK, LOW);
  digitalWrite(DDS_DATA, HIGH);
  digitalWrite(DDS_CLOCK, HIGH);
  digitalWrite(DDS_DATA, LOW);
}

void outZero() {
  digitalWrite(DDS_CLOCK, LOW);
  digitalWrite(DDS_DATA, LOW);
  digitalWrite(DDS_CLOCK, HIGH);
}

uint8_t char_to_baudot(char c, char *array)
{
  int i;
  for (i = 0; i < ARRAY_LEN; i++)
  {
    if (array[i] == c)
      return i;
  }

  return 0;
}

void rtty_txbyte(uint8_t b)
{
  int8_t i;

  rtty_txbit(0);

  /* TODO: I don't know if baudot is MSB first or LSB first */
  /* for (i = 4; i >= 0; i--) */
  for (i = 0; i < 5; i++)
  {
    if (b & (1 << i))
      rtty_txbit(1);
    else
      rtty_txbit(0);
  }

  rtty_txbit(1);
}

enum baudot_mode
{
  NONE,
  LETTERS,
  FIGURES
};

void rtty_txstring(char *str)
{
  enum baudot_mode current_mode = NONE;
  char c;
  uint8_t b;
```

```
  while (*str != '\0')
  {
    c = *str;
    /* some characters are available in both sets */
    if (c == '\n')
    {
      rtty_txbyte(LINEFEED);
    }
    else if (c == '\r')
    {
      rtty_txbyte(CARRRTN);
    }
    else if (is_lowercase(*str) || is_uppercase(*str))
    {
      if (is_lowercase(*str))
      {
        c -= 32;
      }

      if (current_mode != LETTERS)
      {
        rtty_txbyte(LETTERS_SHIFT);
        current_mode = LETTERS;
      }

      rtty_txbyte(char_to_baudot(c, letters_arr));
    }
    else
    {
      b = char_to_baudot(c, figures_arr);

      if (b != 0 && current_mode != FIGURES)
      {
        rtty_txbyte(FIGURES_SHIFT);
        current_mode = FIGURES;
      }

      rtty_txbyte(b);
    }

    str++;
  }
}



// Transmit a bit as a mark or space
void rtty_txbit (int bit) {
  if (bit) {
    // High - mark
    //digitalWrite(2, HIGH);
    //digitalWrite(3, LOW);

    frequency(RTTY_TXF + RTTY_OSET);

  } else {
```

```
    // Low - space
    //digitalWrite(3, HIGH);
    //digitalWrite(2, LOW);

    frequency(RTTY_TXF);

  }

  // Delay appropriately - tuned to 45.45 baud.
  delay(22); //sets the baud rate
  //delayMicroseconds(250);
}
```

Procesory AVR

Radiolatarnia CW i QRSS G0UPL

Kod źródłowy radiolatarni CW i QRSS autorstwa G0UPL dla procesora ATtiny13.
W tabeli 4.1 podano połączenia zworek dla możliwych szybkości transmisji.

Tabela 4.1

	12 sł./min	6 sł./min.	QRSS1	QRSS3	QRSS6	QRSS10	QRSS15	QRSS20
S2 (n. 7)					X	X	X	X
S1 (n. 6)			X	X			X	X
S0(n. 5)		X		X		X		X

```
#include <avr/io.h>
#include <avr/interrupt.h>

volatile uint8_t msgIndex;
volatile uint8_t timerCounter;
volatile uint8_t counter2;
volatile uint8_t audio;
volatile uint8_t key;
volatile uint8_t bit;
volatile uint8_t pause;
volatile uint8_t character;
volatile uint8_t speed;
volatile uint16_t callsign;
volatile uint8_t keyDelay;

#define PERIOD 6

#define A     0b11111001,
#define B     0b11101000,
#define C     0b11101010,
#define D     0b11110100,
#define E     0b11111100,
#define F     0b11100010,
#define G     0b11110110,
#define H     0b11100000,
#define I     0b11111000,
#define J     0b11100111,
#define K     0b11110101,
#define L     0b11100100,
#define M     0b11111011,
#define N     0b11111010,
#define O     0b11110111,
#define P     0b11100110,
#define Q     0b11101101,
#define R     0b11110010,
#define S     0b11110000,
#define T     0b11111101,
#define U     0b11110001,
#define V     0b11100001,
#define W     0b11110011,
#define X     0b11101001,
```

```
#define Y     0b11101011,
#define Z     0b11101100,
#define _SPC  0b11101111
#define _0    0b11011111,
#define _1    0b11001111,
#define _2    0b11000111,
#define _3    0b11000011,
#define _4    0b11000001,
#define _5    0b11000000,
#define _6    0b11010000,
#define _7    0b11011000,
#define _8    0b11011100,
#define _9    0b11011110,
#define _BRK  0b11010010
#define _KEYUP   0b10000000
#define _KEYDN   0b10100000

#define MSGMAX 9
#define SHORTSTART 0;
int8_t msg[MSGMAX] = { K B _1 K I X _BRK, _2 _SPC };

uint8_t speeds[8] = {1, 2, 10, 30, 60, 100, 150, 200};
uint8_t dit[8] = {150, 150, 150, 150, 150, 150, 150, 150};
//uint8_t speeds[8] = {1, 1, 1, 10, 30, 60, 100, 200};
//uint8_t dit[8] = {150, 36, 30, 150, 150, 150, 150, 150};
// DIT        SPEED  WPM
// 150        1      12wpm
// 150     2      6wpm
// 150     10     QRSS1
// 150     30     QRSS3
// 150     60     QRSS6
// 150     100    QRSS10
// 150     150    QRSS15
// 150     200    QRSS20
// 36      1      50wpm
// 30      1      60wpm

int main(void)
{
    DDRB = 24;

    TCCR0B |= (1<<CS01) | (1<<CS00);  // Prescale by 8
    TIMSK0 |= (1<<TOIE0);
    msgIndex = 0xff;

    sei();

    return 0;
}

SIGNAL(SIG_OVERFLOW0)
{
    audio++;

    if (audio == 1)
    {
```

```
        if (key) PORTB |= 0x08;
    }
    else
    {
        PORTB &= ~(0x08);
        audio = 0;
    }

              // 1500Hz here
    timerCounter++;

    if (timerCounter == dit[speed])
    {
              // 10Hz here
        timerCounter = 0;
        callsign++;

        if (keyDelay)
            keyDelay--;
        else
        {
            counter2++;
            if (counter2 >= speeds[speed])
            {
                counter2 = 0;

                if ((character == _KEYDN) || (character == _KEYUP))
                {
                    key = 0xff;
                    bit = 0;
                }
                else
                {
                    if (!pause)
                    {
                        key--;
                        if ((!key) && (!bit)) pause = 2;
                    }
                    else
                        pause--;
                }

                if (key == 0xff)
                {
                    if (!bit)
                    {
                        msgIndex++;
                        if (msgIndex == MSGMAX)
                        {
                            msgIndex = SHORTSTART;
                            if (callsign > 6000)
                            {
                                msgIndex = 0;
                                callsign = 0;
                                speed = 0;
                            }
```

```
                        else
                        {
                            msgIndex = SHORTSTART;
                            speed = (PINB & 0x07);
                        }
                    }

                    bit = 7;
                              // Get character from message
                    character = msg[msgIndex];
                              // Look for 0 signifying start of
coding bits
                    while (character & (1<<bit))
                    {
                        bit--;
                    }
                }

                bit--;

                if (character == _SPC)
                    key = 0;
                else if (character == _KEYDN)
                    key = 1;
                else if (character == _KEYUP)
                    key = 0;
                else
                {
                    key = character & (1<<bit);

                    if (key)
                        key = 3;
                    else
                        key = 1;
                }

                if ((character == _KEYDN) || (character == _KEYUP))
keyDelay = 100;
            }

            if (key)
                PORTB |= (0x10);
            else
                PORTB &= ~(0x10);
        }
    }
}

    TCNT0 = 156;
}
```

Literatura i adresy internetowe

Spis do rozdziału 1

[1] f6cte.free.fr – program “Multipsk”
[2] www.qsl.net/dl4yhf/spectra1.html – „Spectrum Lab” autorstwa DL4YHF
[3] www.qsl.net/in3otd/glfer.html – „glfer” autorstwa Claudia Girardi IN3OTD
[4] www.w1hkj.com/Fldigi.html – program „Fldigi“
[5] www.weaksignals.com – witryna Albertio di Bene I2PHD (programy „Argo”, „Spectran”, „Jason” i in.)

Spis do rozdziału 2

[1] “FA-Syntesizer-Bausatz mit beheizten Si570”, Norbert Graubner, DL1SNG, Funkamateur 57 (2008), nr. 9, str. 953 – 956.
[2] “Minimalistischer hochwertiger Synthesizer mit USB Steuerung”, Thomas Baier, DG8SAQ, Funkamateur 57 (2008), nr. 6, str. 622 – 624.
[3] pe1nnz.nl.eu.org – radiolatarnia z Si570
[4] www.funkamateur.de – moduł syntezera na Si570
[5] www.hanssummers.com – konstrukcje radiolatarni QRSS
[6] www.hanssummers.com/qrss570.html
[7] www.mydarc.de/dg8saq/SI570/index.shtml – oprogramowanie mikrokontrolera ATTiny
[8] www.mydarc.de/dg8saq/hidden/USB_synth.zip – program sterujący na PC

Spis do rozdziału 4

[1] „A DDS based QRSS (and CW) beacon”, Mateo Campanella, IZ2EEQ, “QEX” wrz./paźdź. 2007, str. 24 – 28.
[2] hamradio.lakki.iki.fi/new/Beacons/controllers/hellschreiber_beacon/hellbeacon.shtml – radiolatarnia ZL1HIT
[3] home.swipnet.se/~w-41522/lfbeacon/longwave.html – radiolatarnia PSK31/CW dla pasma 136 kHz
[4] ka7oei.com/ – radiolaratnie PSK31 i FSK31 na fale długie i średnie
[5] members.tripod.com/ve2yag/index.htm – witryna VE2YAG
[6] www.arduino.cc
[7] www.bknd.com/cc5x/index.shtml – kompilator C dla mikroprocesorów PIC
[8] www.conquestsys.com/hamradio – radiolatarnia ZL1HIT
[9] www.g6avk.demon.co.uk/beaconpics.html
[10] www.hamlan.org/tech/picbeacon/picbeacon.htm
[11] www.hamlan.org/tech/picbeacon2/picbeacon2eng.htm – oprogramowanie radiolatarni IK2BCP
[12] www.hamlan.org/tech/picbeacon/picbeacon.htm – drugi wariant radiolatarni
[13] www.microchip.com – asembler dla mikroprocesorów PIC
[14] www.sprut.de/electronic/pic/index.htm – przykłady różnorodnych zastosowań mikrokontrolerów PIC ze schematami i programami w asemblerze
[15] www.ussc.com/~turner/psk_medfer.html – radiolatarnie PSK31 i FSK31 dla pasm długo- i średniofalowych

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”