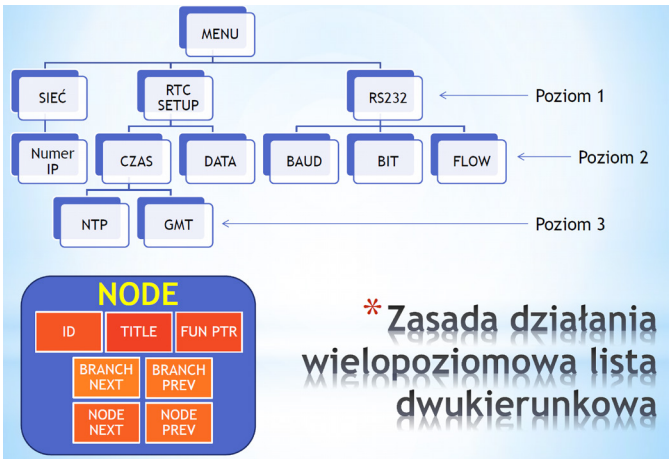
MkMenu Library – oprogramowanie do tworzenia menu

W artykule opisano uniwersalną biblioteką z generatorem do tworzenia własnych systemów menu dla dowolnej rodziny mikrokontrolerów. Biblioteki można używać nie tylko z dowolnym mikrokontrolerem, lecz również z dowolnym rodzajem wyświetlacza, a nawet na terminalu RS232. Do sterowania można użyć nieograniczonej gamy urządzeń wejściowych: od zwykłych przycisków, poprzez enkoder, terminal, podczerwień aż po mechanizmy typu „CapSense”.

Myślę, że każdy kto zajmuje czy zajmował się mikrokontrolerami wcześniej czy później stanie lub stanął przed zagadnieniem związanym z utworzeniem menu, które miałoby posłużyć do wykonywania nastaw własnego urządzenia. Przez wiele lat borykałem się z tym, aż wstyd przyznać, na zasadzie tworzenie całkowicie odrębnych mechanizmów obsługi menu dla każdego z moich urządzeń. Efektem był nadmiar pracy i za każdym razem pisanie kodu źródłowego całkowicie od nowa. Powiem więcej, nawet nie wyobrażałem sobie tego, aby można było wykonać jakiś uniwersalny mechanizm do tego typu projektów z uwagi właśnie na fakt, że przecież każdy projekt różni się diametralnie od innego, jeśli chodzi o sprzęt. Zarówno ten, na którym ma wyświetlać się menu, a także sprzęt odpowiedzialny za obsługę urządzeń wejściowych. Owszem, język C pozwala na tworzenie na pewnym etapie tzw. list jedno- i dwukierunkowych, a za pomocą wskaźników można ułatwiać sobie ten proces, jednak największą udręką – i każdy chyba to przyzna – jest próba dostosowywania raz napisanych procedur obsługi menu do innego urządzenia. Nadszedł w końcu czas, gdy postanowiłem się z tym raz na zawsze rozprawić, czyli przygotować nie tylko uniwersalną bibliotekę w języku C, która stanowi całkowicie abstrakcyjną warstwę obsługi menu (zupełnie oderwaną od sprzętu), ale do tego przygotowałem narzędzie w postaci programu na komputer PC pod Windows, który pełni rolę generatora menu. Wykonuje za nas najcięższą pracę, zaś programista może skupić się jedynie na prostym mechanizmie utworzenia szkieletu MENU w przyjaznym interfejsie użytkownika.



Rysunek 1. Poglądowa struktura menu

Gdy rozpoczynałem pracę koncepcyjną nad stworzeniem uniwersalnego menu, postawiłem przed sobą pewne założenia, które na początkowym etapie wydawały mi się wręcz nie do zrealizowania.

1. Zamknięcie całości w jedną bibliotekę w języku C.
2. Prostota obsługi menu we własnym kodzie każdego projektu.
3. Możliwość zastosowania absolutnie dowolnego wyświetlacza począwszy od alfanumerycznych poprzez graficzne monochromatyczne i kolorowe aż po możliwość zastosowania terminalu RS232 uruchomionym na dowolnym urządzeniu. Terminal z obsługą VT100.
4. Możliwość zastosowania dowolnych urządzeń wejściowych, jak zwykłe przyciski, piloty podczerwieni, enkoder, terminal czy też obsługa przycisków dotykowych „CapSense”.
5. Wielopoziomowa struktura menu.
6. Proste dostosowanie dowolnej struktury MENU do własnego projektu.
7. Stworzenie dodatkowego narzędzia do wspomagania konstrukcji struktury menu i generator kodu źródłowego w C.

Pierwszym krokiem była decyzja oparcia działania biblioteki MkMenu o mechanizm wielopoziomowej listy dwukierunkowej w języku C, gdzie każdy węzeł menu będzie umożliwiał poruszanie się po niej w każdym z dwóch wymiarów. Alternatywnie można by było pomyśleć o zwykłej tablicy dwuwymiarowej, lecz szybko stanęlibyśmy przed problemem mało optymalnego zagospodarowania wszystkich jej komórek. Tymczasem lista dwukierunkowa i wielopoziomowa oparta dodatkowo tylko o kilka wskaźników pozwala na zagospodarowanie tylko tyle pamięci w mikrokontrolerze, ile węzłów ma liczyć sobie menu. Na **rysunku 1** pokazano poglądowy plan przykładowego menu wraz z widokiem pojedynczego węzła. W przykładzie tym widoczne są tylko trzy poziomy menu. Każdy węzeł (node) składa się zaledwie z 7 wskaźników:

1. **ID** – unikalny numer ID dla każdego węzła w ramach całego menu.
2. **TITLE** – wskaźnik do tekstu, który ma być wyświetlany dla węzła.
3. **FUN PTR** – wskaźnik do funkcji, która ma zostać wykonana po wybraniu pozycji menu, o ile wcześniej zdefiniujemy funkcję dla węzła. Nie każdy węzeł musi z niego korzystać.
4. **BRANCH NEXT** – wskaźnik do poprzedniej gałęzi w menu.
5. **BRANCH PREV** – wskaźnik do następnej gałęzi w menu.
6. **NODE NEXT** – wskaźnik do następnego węzła.
7. **NODE PREV** – wskaźnik do poprzedniego węzła.

Cztery ostatnie wskaźniki zapewniają poruszanie się w dowolnym kierunku po wszystkich węzłach (opcjach) całego menu, nawet przy dużo bardziej rozbudowanej strukturze niż na rys. 1. Jeśli komuś wydaje się, że wykonanie takiej struktury w języku C będzie skomplikowane, proszę spojrzeć na **rysunek 2** przedstawiający przykładowy kod źródłowy. Struktura ma nazwę TNODE i zawiera w sobie dokładnie te pola, które przed chwilą wymieniłem w punktach od 1 do 7. W związku z tym, że całe menu ma opierać się na identycznie skonstruowanych węzłach, cztery ostatnie wskaźniki muszą być dokładnie takiego samego typu, stąd bierze się ich typ o nazwie „ANODE”.

Proszę zwrócić uwagę, że na tym etapie nie mamy nawet w najmniejszym stopniu jakichkolwiek związków ze sprzętem. Struktura jest czysto abstrakcyjna i może zostać umieszczona w dowolnym mikrokontrolerze

Rysunek 2. Przykład struktury reprezentującej węzeł w języku C

Rysunek 3. Przykładowe kody sterowania menu

w dowolnej pamięci. Z uwagi na fakt, że cały przykład jest przeznaczony dla mikrokontrolerów AVR i przygotowany dla kompilatora AVR GCC to stąd wynikają takie a nie inne nazwy typów dla pól wewnątrz struktury.

Zanim przejdę do dalszego opisu chciałbym przedstawić na **rysunku 3** przykład z wczesnych prac nad mechanizmem biblioteki MkMenu, który w najlepszy sposób odda łatwość sterowania menu za pomocą trzech zupełnie różnych urządzeń wejściowych: zwykłych przycisków, enkodera i pilota podczerwieni.

Tutaj już mamy pierwszy styk warstwy abstrakcyjnej ze sprzętem tyle, że jak się za chwilę przekonamy, mechanizmy biblioteki MkMenu, nie mają absolutnie nic wspólnego z obsługą przycisków, pilota podczerwieni czy też enkodera. Można zauważyć, że biblioteka udostępnia jedynie kilka krótkich metod (funkcji), które programista może wywoływać w reakcji na wciśnięcie przycisku, przycisku pilota, enkodera albo w reakcji na obracanie enkodera w dowolnym kierunku. W docelowym kodzie biblioteki MkMenu występuje dosłownie kilka funkcji odpowiedzialnych za poruszaniem się po menu (sterowanie kursorem):

- mk_menu_dec(),
- mk_menu_inc(),
- mk_menu_click(),
- mk_menu_hide(),
- mk_menu_toggle().

Mam nadzieję, że już same nazwy zaprezentowanych funkcji, pozwalają domyśleć się, w jakich momentach powinny być zastosowane. Dla przykładu, jeśli mielibyśmy używać dwóch przycisków odpowiedzialnych za poruszanie się po menu w górę i w dół to należy w ramach procedury obsługi odpowiedniego przycisku wywołać albo **mk_menu_inc()** albo **mk_menu_dec()**, co spowoduje jak się później przekonamy przesunięcie kursora menu. Jeśli np. zechcemy wejść w głębszy poziom menu, albo wywołać funkcję związaną z danym węzłem, lub też dokonać zmiany nastaw, łatwo się domyśleć, że będzie trzeba wywołać funkcję **mk_menu_click()**, która zadziała odpowiednio w zależności od kontekstu, czyli w zależności od tego czy np. dany węzeł jest gałęzią, która ma kolejny poziom węzłów czy też zechcemy kliknąć w konkretny węzeł, aby dokonać zmiany nastaw z nim związanych. Pozostałe dwie funkcje służą do zmiany widoczności menu na ekranie.

Na tym etapie nadmienię i jest to bardzo istotne dla zrozumienia działania całego mechanizmu biblioteki MkMenu, że to programista sam we własnym zakresie musi bądź to przygotować własne biblioteki do obsługi dowolnych urządzeń wejściowych bądź posłużyć się wcześniej zdobytymi ew zakupionymi bibliotekami. Dzięki temu łatwo sobie wyobrazić, że rozbudowa możliwości przedstawionych na rysunku 3 staje się banalnie prosta. Jeśli przykładowo zastosujemy obsługę przycisków dotykowych, to po prostu użyjemy dokładnie wciąż tych samych metod (funkcji) biblioteki MkMenu odpowiadających za zmianę pozycji kursora (węzła) w menu.

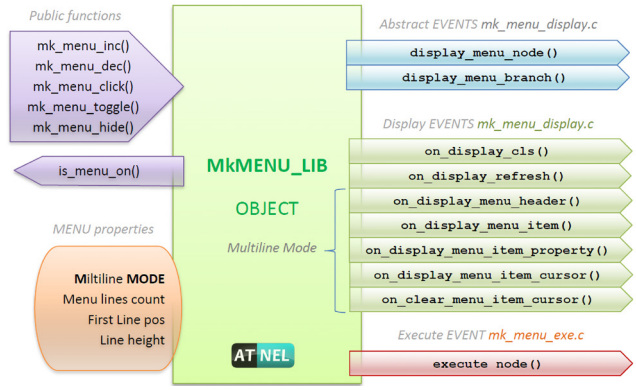
Widać zatem, jaka jest idea łączenia warstwy abstrakcyjnej biblioteki MkMenu z warstwami odpowiedzialnymi za niskopoziomową obsługę, związaną ściśle już ze sprzętem, odpowiedzialną za poruszanie się po menu. Łatwo się domyśleć, że na podobnej zasadzie będzie odbywało się połączenie warstwy abstrakcyjnej ze sprzętową, jeśli chodzi o sam proces wyświetlania.

Można śmiało powiedzieć, że sama biblioteka MkMenu nie ma również pojęcia o tym, co zdecyduje programista, który się nią posługuje jak chodzi o wybór rodzaju wyświetlacza. To programista podobnie jak w przypadku np. doboru biblioteki do obsługi enkodera, bo na taki sposób sterowania się zdecydował, będzie musiał wybrać sobie rodzaj wyświetlacza i we własnym zakresie wybrać stosowną bibliotekę do jego obsługi.

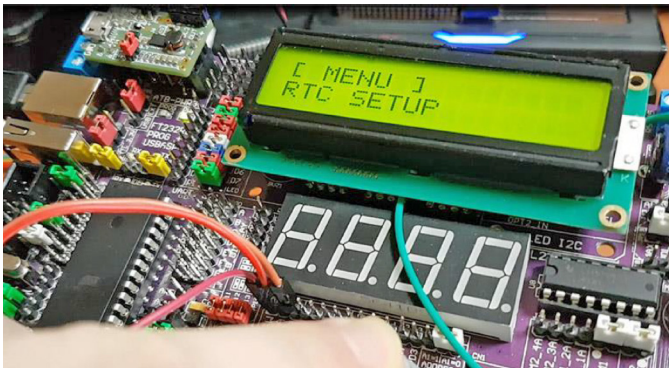
Zdaję sobie sprawę, że o ile łatwo sobie wyobrazić sposób połączenia abstrakcyjnej części biblioteki ze sprzętową obsługą urządzenia wejściowego do poruszania się po menu, w końcu to tylko kilka funkcji, to zapewne dużo trudniej jest wyobrazić sobie (szczególnie początkującemu programiście) jak to będzie wyglądało w przypadku połączenia z warstwą sprzętową odpowiedzialną za obsługę wyświetlaczy. Nie wiem wprawdzie czy czytając ten tekst miałeś do czynienia być może wcześniej z jakimś językiem obiektowym na komputerze PC, czy znajome są tobie te zagadnienia, ale nawet jeśli nie, to podpowiem, że moim głównym celem było stworzenie i to w czystym języku C czegoś na kształt obiektu. W językach obiektowych jak np. C++, C# i innych programowanie charakteryzuje się m.in. tym, że programista może tworzyć na bazie klas obiekty a później powoływać do życia ich instancje w programie. Taki obiekt ma właściwości (properties) i metody (methods). Dzięki temu jeden obiekt bazowy dobrze skonstruowany można używać do obsługi pewnych logicznych jednostek.

Biblioteka MkMENU będzie stanowiła taki właśnie pseudo obiekt. Używam słówka „pseudo" z uwagi, na to, że język C nie wspiera w ogóle obiektowości, nie mniej jednak myślenie obiektowe bardzo się przydaje nawet w języku C. Na **rysunku 4** pokazano poglądowy widok obiektu, jaki stanowi MkMenu. Po lewej stronie na dole w pomarańczowej ramce pokazano właściwości (properties) naszego obiektu. Jest to tak naprawdę kilka zmiennych, które należy odpowiednio ustawić przy inicjalizacji obiektu, aby działał on zgodnie założeniami, które zależne są z kolei od pewnych konkretnych wartości tychże właściwości. Również po lewej stronie lecz w ramach fioletowych ramek/figur przedstawiłem kilka publicznych funkcji jakie udostępnia obiekt dla użytkownika i już na podstawie wcześniejszego opisu rozpoznajemy, że są one odpowiedzialne za sterowanie kursorem MENU, odpowiadają za poruszanie się po całym menu, za wyświetlanie czy ukrywanie menu na ekranie.

Po prawej stronie rys. 4 pokazano cały szereg (nazwałem to zdarzeniami EVENTS), ale gdyby to był prawdziwy obiekt, to należałoby zastosować tutaj nazwę METODY. (Metody obiektu to w rzeczywistości również pewne funkcje, tyle, że są one wywoływane jako zdarzenia przez sam obiekt w zależności od pewnych sytuacji, które zachodzą w kodzie). Łatwo się domyśleć, że w wypadku menu ta interakcja będzie ściśle związana



Rysunek 4. MkMenu jako obiekt



Rysunek 5. Przykład MkMenu na wyświetlaczu alfanumerycznym 2×16

z działaniem funkcji publicznych zawiadujących kursorem menu czy też klikaniem w odpowiednie pozycje, a co za tym idzie obiekt będzie musiał podejmować szereg czynności w reakcji na te zmiany i wywoływać swoje metody odpowiedzialne w przypadku MkMenu za dwie grupy zdarzeń:

1. Zdarzenia związane z wyświetlaniem danych menu na ekranie.
2. Zdarzenie **execute_node()** odpowiedzialne za wywoływanie funkcji przydzielonych do niektórych węzłów przez programistę.

Zdarzenia (Metody) obiektu podzielone są na kilka grup i podgrup. Oznaczone są one odpowiednimi deseniami kolorów. Patrząc od góry mamy do czynienia z dwoma bazowymi zdarzeniami o nazwach:

- display_menu_node(),
- display_menu_branch().

W zasadzie mogłyby nie istnieć pozostałe w zielonych ramkach, ponieważ wszystkie oznaczone na zielono poniżej wywodzą się, z tych dwóch. Innymi słowy wszystkie zdarzenia oznaczone zielonym kolorem ramek można napisać we własnym zakresie poza biblioteką. Dlaczego zatem obiekt został zaopatrzony również w te dodatkowe zdarzenia wewnątrz zielonych ramek? Odpowiedź jest prosta. Zdarzenia o nazwach:

- on_display_cls(),
- on_display_refresh(),
- on_display_menu_header(),
- on_display_menu_item(),
- on_display_menu_item_property(),
- on_display_menu_item_cursor(),
- on_clear_menu_item_cursor(),

powstały w drugim etapie budowy całej biblioteki, ponieważ na początku założyłem, że programista mając do dyspozycji tylko te dwa bazowe zdarzenia (niebieskie ramki) będzie musiał sam we własnym zakresie oprogramować i wyprowadzić sobie wg własnego życzenia wszystkie pozostałe zdarzenia związane z wyświetlaniem danych na własnym wyświetlaczu. Sam jednak szybko się przekonałem, że warto wprowadzić pewien ustandaryzowany sposób wyświetlania menu, który oczywiście jest tylko pewnego rodzaju propozycją z mojej strony, lecz „zawsze pod ręką", co drastycznie znowu upraszcza posługiwanie się biblioteką. Zdarzenia w zielonych ramkach, zawierają pełną logikę związaną z obliczaniem pozycji wyświetlanych danych menu, właściwości menu, a także samego kursora menu. Oczywiście, jest to możliwe po ustawieniu właściwości o nazwie „**MILTILINE MODE**" (=1). Jest to również ustawienie domyślne. Korzystanie z tych mechanizmów wiąże się z pewnym narzutem, jeśli chodzi o zajętość pamięci Flash, ale jeśli ktoś zechciałby zrealizować to wszystko własnym sumptem i z zastosowaniem własnej optymalizacji kodu, to jest to jak najbardziej możliwe. Wystarczy wyłączyć „**Multiline MODE**" (wyzerować), co spowoduje, że z kodu biblioteki automatycznie zniknie i nie skompiluje się cały fragment kodu odpowiedzialny za logikę wyświetlania zaproponowaną przeze mnie, nie skompilują się procedury zdarzeń z zielonych ramek zaś programista sam to wszystko oprogramuje (jak wspominałem wcześniej) na podstawie dwóch bazowych zdarzeń umieszczonych w niebieskich ramkach.

Spoglądając na nazwy zdarzeń opisanych w kolorze zielonym, również można się domyśleć, za co każde z nich odpowiada i kiedy jest ono wywoływane. W ramach plików bibliotecznych umieściłem nawet jeden plik źródłowy „mk_menu_display.c" zawierający prototypy tych funkcji wraz z przykładowymi, ale opatrzonymi komentarzem kodami źródłowymi, na których podstawie każdy może sobie obsłużyć dla pierwszego testu wyświetlanie menu na popularnym wyświetlaczu alfanumerycznym LCD 2×16 ze sterownikiem HD44780 (**rysunek 5**). Wystarczy zainkludować swoją własną bibliotekę do obsługi tego rodzaju wyświetlacza. Gdyby ktoś chciał, to istnieje również możliwość skorzystania z przykładów testowych dla wyświetlacza graficznego OLED 128×64 lub podobnego (**rysunek 6**).

Pozostało do omówienia jeszcze tylko jedno – na rys. 4, w jego prawym dolnym rogu, treść umieszczona w czerwonej ramce o nazwie **execute_node()**. Jest ono bardzo istotne z uwagi na uniwersalność mechanizmu menu, ponieważ odpowiada ono za wywoływanie funkcji programisty odpowiedzialnych za dokonywanie nastaw urządzenia związanych z konkretną pozycją (węzłem) menu, albo wykonywaniem jakiejś innej specyficznej akcji, np. testu urządzenia itp. W tym celu w ramach biblioteki utworzony został plik źródłowy o nazwie „mk_menu_exe.c", w którym znajduje się funkcja zdarzenia wywoływana za każdym razem przez bibliotekę, gdy użytkownik urządzenia kliknie przycisk na pozycji menu mającej przydzieloną funkcję. Zasada działania jest następująca: do wywoływanej funkcji przekazane są argumenty zawierające odpowiedni numer funkcji. Przy okazji numer przekazywany jest w postaci nazwy za pomocą typu wyliczeniowego enum po to, aby programiście łatwo było skojarzyć, w którym miejscu należy wywołać już własną specyficzną funkcję obsługi. Jest to zatem trzecie miejsce styku warstwy abstrakcyjnej biblioteki z częścią sprzętową lub logiczną całego projektu.

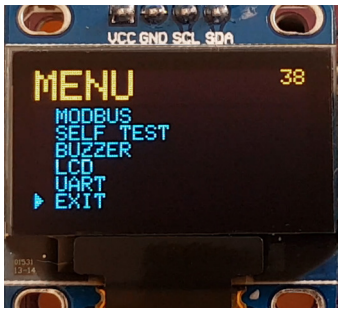
Zapewne każdy zadaje sobie na tym etapie dwa pytania.

1. Jak wygląda kod źródłowy biblioteki?
2. W jaki sposób należy przygotować samą strukturę menu, wszystkie węzły?

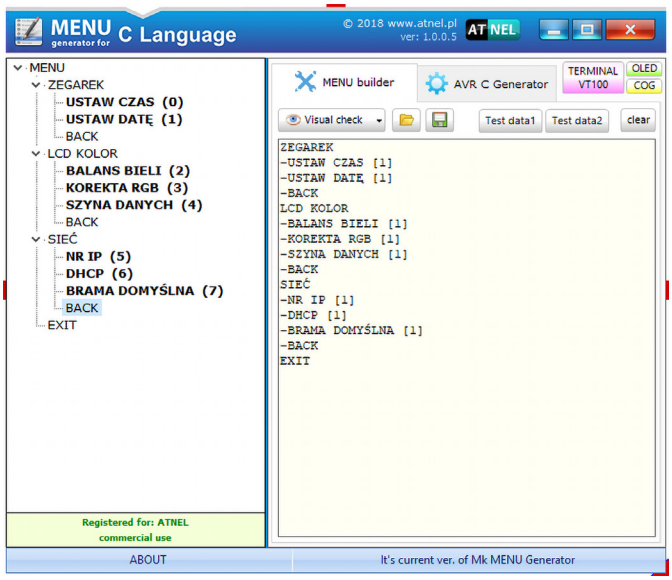
Zwykle najwięcej problemów i najwięcej pracy jest na etapie tworzenia w kodzie programu struktury całego menu niezależnie od sposobu, w jaki realizujemy menu. Zapewne każdy, kto chociaż raz zmierzył się z tym zagadnieniem we własnym zakresie przyzna mi rację. Tymczasem odpowiedzi na te dwa pytania związane są z pewnym narzędziem programistycznym, które przygotowałem na potrzeby MkMenu. Powstał bowiem specjalistyczny program pod Windows o nazwie „Mk Menu Gen", czyli generator MkMenu. Pełni on dwie zasadnicze funkcje.

1. Umożliwia projektowanie dowolnej struktury MENU w bardzo prosty i intuicyjny sposób.
2. Potrafi wygenerować całą strukturę menu w postaci kodu źródłowego w języku C.

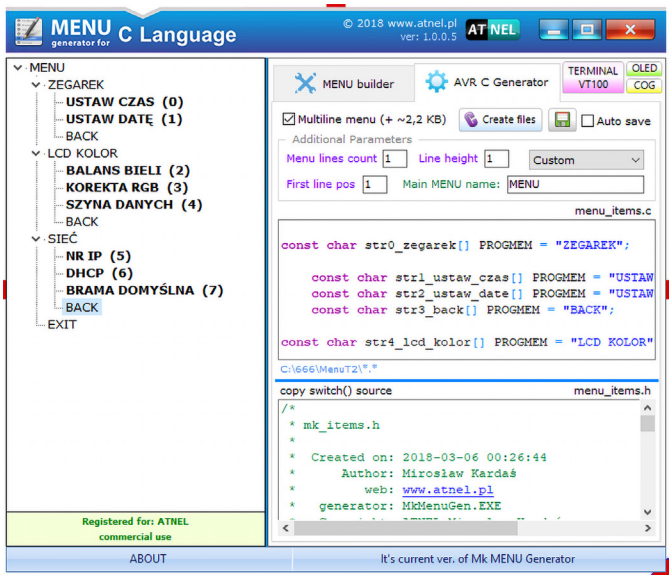
Dodatkowo, generuje pełny kod źródłowy biblioteki a nawet



Rysunek 6. Przykład MkMenu na wyświetlaczu OLED



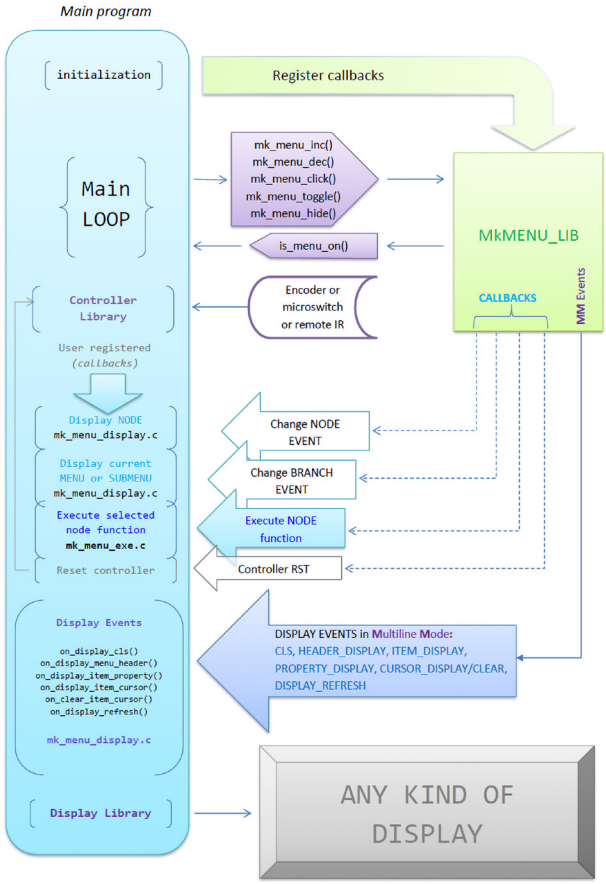
Rysunek 7. Widok programu „Mk Menu Gen" – zakładka do projektowania menu



Rysunek 8. Program „Mk Menu Gen" zakładka generatora kodu źródłowego C

przykładowe kody źródłowe do obsługi niektórych popularnych na rynku rodzajów wyświetlaczy. Na **rysunku 7** widać główne okno programu wraz z pierwszą zakładką, w której możemy wygodnie zaprojektować własne menu w zwykłym trybie tekstowym a następnie podejrzeć je w postaci drzewka po lewej stronie, aby zweryfikować czy wszystko poszło po naszej myśli.

W żółtej ramce po prawej stronie można wpisywać poszczególne pozycje menu z zagłębieniami, które symbolizujemy za pomocą znaku myślnika(-ów) przed nazwą węzła. Następnie wystarczy kliknąć przycisk o nazwie „Visual check", aby podejrzeć w ramce po lewej stronie widok projektowanego menu w postaci drzewka. Naturalnie projekt można zapisać lub wczytać z dysku za pomocą przycisków z odpowiednimi ikonkami. Gdy już jesteśmy pewni, że jest to ostateczna wersja menu wystarczy przejść do drugiej zakładki (AVR GCC Generator) jak na **rysunku 8** i za pomocą przycisku „Create Files" wygenerować kody źródłowe w języku C. Można je przenieść przez „copy & paste" do własnego projektu albo zapisać wręcz bezpośrednio do plików projektu, uprzednio ustawiając właściwą ścieżkę do projektu, w którym będziemy korzystać z projektowanej wersji menu. Dzięki temu programowi dokonywanie późniejszych zmian w strukturze menu, gdy zajdzie taka konieczność (np. dodania nowych pozycji w trakcie



Rysunek 9. MkMenu – Diagram

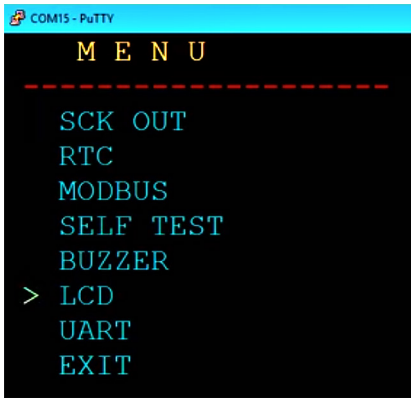
rozwoju urządzenia), jest bajecznie proste i wygodne. Program generuje pliki w taki sposób aby nie zakłócić i nie nadpisać tych miejsc gdzie wcześniej sami uzupełnialiśmy nasz kod źródłowy.

Program na komputer PC można pobrać zupełnie za darmo ze strony ***http://atnel.pl/mkmenugen.html***. Wersja demonstracyjna programu potrafi wygenerować pełną wersję biblioteki w języku C dla mikrokontrolera AVR, a także w 100 procentach przetestować tworzenie własnego menu. Ciekawym uzupełnieniem są poradniki wideo dostępne za darmo na kanale youtube w ramach **playlisty** „**MkMENU**" ***www.youtube.com/mirekk36***. Pomagają one znakomicie szybko wdrożyć się i zapoznać z podstawami nawet przez zupełnie początkujących programistów.

W ramach podsumowania na **rysunku 9** pokazano diagram prezentujący ogólną zasadę działania biblioteki MkMenu jako obiektu w ramach całego projektu, z uwzględnieniem wszystkich zagadnień o których była wcześniej mowa.

Na zakończenie, jako ciekawostkę na **rysunku 10** pokazano widok menu zorganizowanego tym razem nie na wyświetlaczu, lecz w zwykłym programie terminala obsługującego standard VT100. Może to być znany i popularny programu o nazwie „Putty.exe". Natomiast samo menu obsługiwane jest za pomocą enkodera. Dzięki obsłudze kodów VT100 w terminalu można wręcz powiedzieć, że mamy do dyspozycji nowy rodzaj i to kolorowego wyświetlacza alfanumerycznego.

Mirosław Kardaś
biuro@atnel.pl



Rysunek 10. Widok MkMenu w terminalu – Putty