Programowanie eZ430 Chronos

Projekt zegarka sportowego

Celem artykułu jest przybliżenie Czytelnikom problematyki programowania zestawu eZ430 Chronos, aby ułatwić im uruchamianie w nim własnych aplikacji. Opisany przykładowy program nie będzie tylko demonstracyjnym, lecz jest to rozbudowana aplikacja zegarka sportowego, którego można z powodzeniem używać podczas treningów lub zawodów sportowych. Jego koszt jest kilkukrotnie niższy niż cena podobnych produktów markowych.

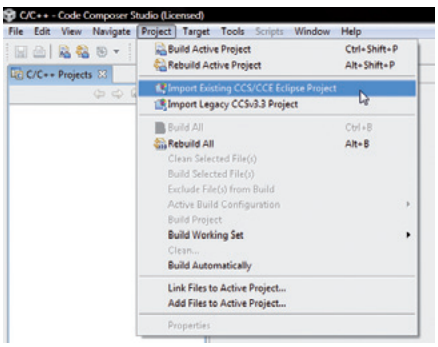
Programu dla Chronosa opracowano w środowisku *Code Composer Studio* (CCS) dostarczanym na płycie CD wraz z zestawem. Jest to środowisko uruchomieniowe przygotowane przez Texas Instruments na bazie Eclipse. Oprócz niego, na płycie zamieszczono także pliki instalacyjne środowiska *IAR KickStart*, które jest alternatywnym dla CCS.

Po uruchomieniu, jeśli komputer jest połączony z Internetem, CCS sam sprawdzi, czy są dostępne aktualizacje i zaproponuje ich zainstalowanie.

Po wgraniu i aktualizacji CCS należy zainstalować oprogramowanie do obsługi Chronosa oraz sterowniki dołączonego programatora i Access Pointa. Po dołączeniu Access Pointa do komputera PC jest on automatycznie wykrywany i system Windows instaluje odpowiednie sterowniki. Podobnie jest z programatorem USB. Można także sprawdzić czy wszystko działa prawidłowo uruchamiając program *Chronos Control Center*.

Po uruchomieniu CCS jest wyświetlane okno wyboru katalogu roboczego. Do pracy z Chronosem zalecam utworzenie oddzielnego folderu na dysku, ponieważ po utworzeniu katalogu projektów w folderze instalacyjnym Chronosa (*C:\Program Files\Texas Instruments...*) może podczas kompilacji wystąpić błąd spowodowany zbyt długą ścieżką dostępu do plików.

Po pierwszym uruchomieniu, w oknie po lewej stronie ekranu nie ma w folderze roboczym żadnych projektów. Dla potrzeb wykonania opisywanej aplikacji posłużymy się gotowym przykładem projektu przygotowanym przez firmę Texas Instruments, który należy zaimportować. Z menu wybieramy *Project–>Import Existing CCS/CCE Eclipse Project* (**rysunek 1**). W oknie dialogowym wybieramy projekt zegarka sportowego. Domyślnie jest on instalowany w *C:\Program Files\Texas Instruments\eZ430-Chronos\Software Projects\CCS\Sports Watch*. Zaznaczamy również opcję *Copy projects into workspace* i klikamy przycisk *Finish*. W oknie po lewej stronie pojawi się struktura projektu.



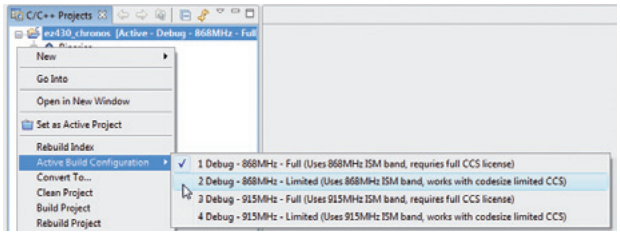
Rysunek 1. Wygląd okna importu projektu

Użytkownicy darmowej wersji CCS powinni wybrać do kompilacji konfigurację *Limited*. Dokonujemy tego poprzez kliknięcie prawym przyciskiem w oknie *C/C++ Projects* i wybieramy opcję z *Active Build Configurations* (**rysunek 2**). Dwukrotne kliknięcie na nazwie pliku powoduje otwarcie go w środkowym oknie edycji. Na początku proponuję skompilować program na próbę, aby upewnić się, że wszystko działa prawidłowo. Polecenie *Build Active Project* uruchomimy klikając na trzeciej ikonce od lewej na pasku zadań lub poleceniem z menu *Project*.

Przed podłączeniem układu zegarka do programatora należy wyjąć go z obudowy i odłączyć baterię. Jest to opisane w dostarczonej dokumentacji oraz dobrze pokazane w filmiku na płycie CD, w folderze *Documentations\Videos*. Po podłączeniu przesyłamy program do zegarka komendą *Debug Active Project* (szósta ikona z menu *Target*), po czym otworzy się okno debuggera. Po wgraniu oprogramowania należy je uruchomić ikonką *Run* z górnego paska okna *Debug*. Zmianę widoku przeprowadzamy po prawej stronie okna CCS.

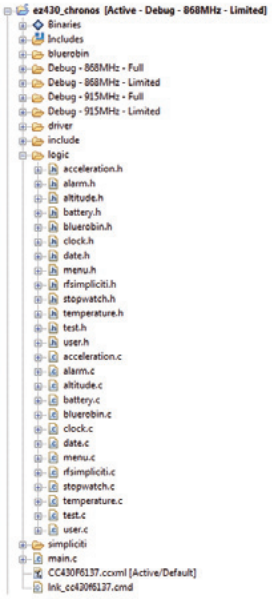
Struktura przykładowego programu

Przed rozpoczęciem zmian przyjrzyjmy się strukturze oprogramowania. Wyjściowym jest plik *main.c*. Na **rysunku 3** pokazano drze-



Rysunek 2. Wybór opcji aktywnej konfiguracji

wo projektu. Jego pliki znajdują się w katalogach: *simpliciti*, *logic*, *include*, *driver* i *bluerobin*. W katalogu *include* znajduje się plik nagłówkowy *project.h* zawierający definicje typów danych. Katalogi *simpliciti* i *bluerobin* zawierają skompilowane biblioteki interfejsów radiowych. Katalog *driver* zawiera pliki z procedurami obsługi peryferiów mikrokontrolera, zarówno wbudowanych (porty IO, timery, przetwornik ADC czy układ radiowy), jak i urządzeń zewnętrznych (buzzer, akcelerometr, czujnik ciśnienia, wyświetlacz). Najwiecej zmian dokonamy w plikach z katalogu *logic*.



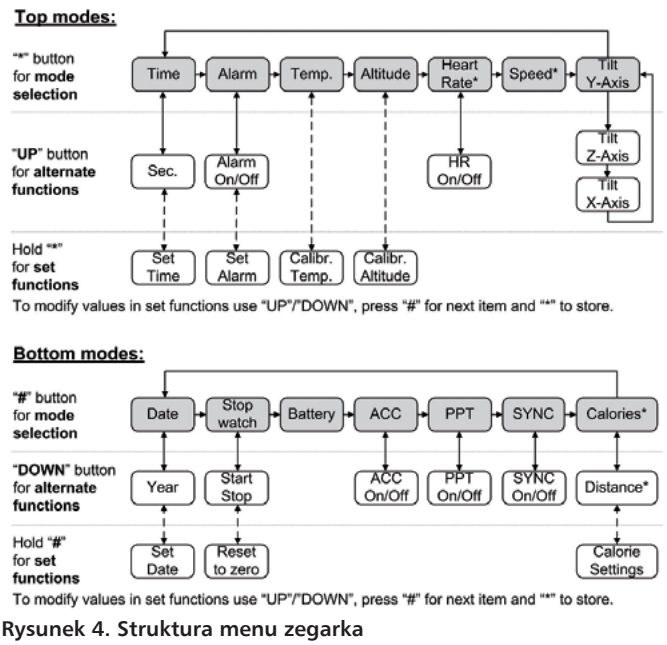
Rysunek 3. Wygląd drzewa projektu

Z menu zegarka (**rysunek 4**) wynika, że pliki te odpowiadają znajdującym się w nim funkcjom. Dla każdej funkcji jest plik nagłówkowy oraz pliki źródłowe napisane w języku C. Pliki *rfsimplici* zawierają wszystkie funkcje komunikacyjne z protokołem *SimpliciTI*, a więc tryby ACC, PPT oraz SYNC. W module *bluerobin* są funkcje obsługi opaski pomiarowej, a więc pomiaru pulsu (*Heart rate*), prędkości i pomiaru wydatkowanych kalorii. Moduł *user* zawiera funkcję *set_value*, która powoduje ustawianie wartości z inkrementacją, dekrementacją, przyspieszoną zmianę przy trzymaniu przycisku oraz automatyczną aktualizację wyświetlacza.

Typy zmiennych są zdefiniowane w pliku *bm.h*. Podstawowe, używane w języku C zostały przedefiniowane, aby były zgodne z danymi używanymi przez mikrokontroler. Zgodnie z nomenklaturą przyjętą przez firmę TI, składają się z litery odpowiadającej typowi liczby oraz cyfr określających liczbę bitów przyjętych do jej przechowywania. Przykładowo u8 oznacza ośmiobitową liczbę bez znaku (*unsigned*), s16 to szesnastobitowa liczba ze znakiem (*signed*), a f32 to liczba zmiennoprzecinkowa (*float*) zajmująca cztery bajty.

W oprogramowaniu jest używane też specjalne nazewnictwo dla przycisków użytkownika. Lewe przyciski oznaczone są jako M1 i M2 natomiast prawe S1, S2. Przycisk podświetlenia oznaczony jest jako BL (**rysunek 5**).

Moduł menu zawiera definicje poszczególnych pozycji wyświetlanych w menu. Struktura menu jest podzielona na linię dolną i górną. Przyjrzyjmy się bliżej definicji struktury menu z pliku *menu.h* (**listing 1a**). Po wciśnięciu danego przycisku, program uruchomia żądaną funkcję bez sprawdzania, która pozycja menu jest aktywna. Przy danych pozycjach menu podane są funkcje obsługujące zdarzenia. Ostatnią pozycją jest wskaźnik na następną pozycję menu. Podczas modyfikacji należy pamiętać, że ostatnia deklaracja menu musi wskazywać na pierwszą funkcję. Przykładową deklarację pozycji menu zamieszczono na **listingu 1b**.



Rysunek 4. Struktura menu zegarka

Rysunek 5. Rozmieszczenie przycisków na zegarku

Wszystkim operacjom w pliku *main.c* przyporządkowano odpowiednie funkcje. Najpierw jest uruchamiana funkcja *init_application()*, która przygotowuje mikrokontroler do pracy. Następnie w funkcji *init_global_variables()* ustawiane są wartości początkowe zmiennych. Zmienne lokalne dla modułów inicjalizowane są w funkcjach *reset_xxx* np. *reset_stopwatch()*. Ustawiane są także domyślnie pierwsze aktywne pozycje menu dla górnego i dolnego wiersza. W funkcji

Radiowe protokoły transmisji wykorzystywane w ez430-Chronos

SimpliciTI

Protokół transmisji opracowany przez Texas Instruments. Może być stosowany we wszystkich zakresach częstotliwości ISM zarówno <1 GHz jak i 2,4 GHz. Przeznaczony do urządzeń bateryjnych o niewielkim zapotrzebowaniu na moc. Obsługuje połączenia typu peer-to-peer jak również strukturę gwiazdową. Proste API obejmujące 5 instrukcji. Główne przeznaczenie to bezprzewodowe sieci czujników. Darmowa implementacja bez potrzeby licencjonowania. Przykłady implementacji dla wszystkich układów radiowych TI (dawniej Chipcon) oraz CC430. W Chronosie odpowiada za komunikację z komputerem PC.

BlueRobin

Protokół opracowany i opatentowany przez BM wireless Ltd.&Co.KG. Dostępne implementacje dla częstotliwości: 315/433/868/915 MHz oraz 2,4GHz. W tej chwili dostępny dla układów radiowych TI, Nordic Semiconductors oraz procesorów TI MSP430 i Epson S1C17. Przeznaczony głównie do komunikacji jednostronnej, lecz możliwa także dwustronna komunikacja. Wbudowana obsługa kolizji pakietów. Stacja odbiorcza może obsługiwać do 200 urządzeń pomiarowych. Opracowany do zastosowań w sporcie i medycynie. Używanie wymaga licencjonowania. W Chronosie odpowiada za komunikację z opcjonalną opaską do pomiaru pulsu.

Listing 1a

```
struct menu
{
   // Wskaźnik na główną funkcję trybu (po wciśnięciu
prawego przycisku)
   void (*sx_function)(u8 line);
   // Wskaźnik na drugą funkcję (po przytrzymaniu lewego
przycisku)
   void (*mx_function)(u8 line);
   // Wskaźnik na funkcję wyświetlającą
   void (*display_function)(u8 line, u8 mode);

   // Zmienna wyzwalająca odświeżanie wyświetlacza
   u8 (*display_update)(void);
   // Wskaźnik na następną pozycję menu (zmiana po
wciśnięciu lewego przycisku)
   const struct menu *next;
};
```

Listing 1b

```
// Line1 - Time
const struct menu menu_L1_Time =
{
   FUNCTION(sx_time),              // direct function
   FUNCTION(mx_time),              // sub menu function
   FUNCTION(display_time),         // display function
   FUNCTION(update_time),          // new display data
   &menu_L1_Alarm,
};
```

Listing 2a
```
// Global Variable section
u8 BL_timer;

...

void BL_on(u8 seconds)
{
   //setting up backlight duration
   BL_timer = seconds;

   BUTTONS_DIR |= BUTTON_BL_PIN; //setup P2.3 to output
   BUTTONS_OUT |= BUTTON_BL_PIN; //set 1 to output
}

void BL_off(void)
{
   // Set button ports to input
   BUTTONS_DIR &= ~ALL_BUTTONS;

   // Enable internal pull-downs
   BUTTONS_OUT &= ~ALL_BUTTONS;
   BUTTONS_REN |= ALL_BUTTONS;
}
```

Listing 2b
```
//Backlight countdown and off
if (BL_timer != 0)
{
   BL_timer--;

   if (BL_timer == 0)
   BL_off();
}
```

Listing 2c
```
// BL button event---------------------------------------
// Activate the backlingt for some time (3 second)
else if(button.flag.bl)
{
   //start backlight (user.c)
   BL_on(3);

   button.flag.bl = 0;
}
```

test_mode() jest wyświetlany ekran powitalny – włączone wszystkie ikony oraz napisy 430 i CC430.

Po inicjalizacji jest wykonywana pętla programowa. Procesor wprowadzany jest w stan uśpienia LPM3 (wyłączone taktowanie procesora, działa tylko zegar ACLK taktujący Timer0) i oczekuje na przerwanie wybudzające.

Procedury obsługi przerwań umieszczone są w poszczególnych modułach urządzeń peryferyjnych w katalogu *driver*. W procedurach przerwań ustawiane są odpowiednie flagi programowe, na przykład flagi od naciśnięcia przycisków. Zdarzenia te obsługiwane są przez funkcję *wakeup_event()*. Tam znajdziemy obsługę wszystkich przycisków. Opisane wyżej funkcje mogą zażądać wykonania pomiarów lub odświeżenia wyświetlacza. Odbywa się to poprzez ustawienie odpowiednich znaczników programowych, których obsługą zajmują się funkcje *process_requests()* oraz *display_update()*. Umieszczenie tych funkcji „na zewnątrz" pozwala na szybkie odblokowanie systemu przerwań.

Listing 3a
```
//counting down Menu L1 in Use timer
if (MenuL1_in_use != 0)
{
        MenuL1_in_use--;
}

//counting down Menu L2 in Use timer
if (MenuL2_in_use != 0)
{
        MenuL2_in_use--;
}
```

Listing 3b
```
//Check if menu is not in use
if ((MenuL1_in_use != 0 ) || (ptrMenu_L1->next == menu_
L1_Time.next))
{
        // Go to next menu entry
        ptrMenu_L1 = ptrMenu_L1->next;
}
else
        // Go to first menu entry
        ptrMenu_L1 = &menu_L1_Time;

// Reset menuL1_in_use timer
MenuL1_in_use = 10;
```

Zmiany

Głównym celem wykonywanych modyfikacji jest dodanie zegarkowi funkcji rejestracji czasów okrążeń. Usuniemy również funkcje nieprzydatne oraz takie, które nie działają zbyt dobrze (pomiar spalonych kalorii, pomiar wysokości). Poprawimy również działanie podświetlenia, pomiaru pulsu oraz usprawnimy obsługę menu.

Podświetlenie

W oryginalnym Chronosie podświetlenie jest załączane przez przycisk. Dla użytkownika jest to niekorzystne, ponieważ należy trzymać go wciśniętym podczas odczytu danych z wyświetlacza. W czasie biegania czy uprawiania innych sportów, przy których pracują ręce, jest to kłopotliwe, a w przypadku jazdy na rowerze wręcz niebezpieczne. Zmienimy sposób działania przycisku tak, aby po jego przyciśnięciu podświetlenie włączało się na określony czas, np. 3 sekundy.

Jeśli przeanalizujemy schemat układu Chronosa to stwierdzimy, że brak jest odrębnego wyprowadzenia portu I/O mikrokontrolera sterującego podświetleniem. Dostępne jest tylko wyprowadzenie, do którego oprócz podświetlenia jest dołączony przycisk. Zatem po wciśnięciu przycisku musimy programowo zmienic tryb portu na wyjsciowy, zasilić podświetlenie i po upływie określonego czasu je wyłączyć, a następnie przywrócić pierwotną funkcję wyprowadzenia portu.

W pliku *ports.c* jest przygotowana funkcja odbioru przerwania z przycisku podświetlenia (*BL*) i ustawienia flagi *button.flag.bl*. Wystarczy dopisać obsługę tej flagi w *main.c*. Wcześniej musimy zmodyfikować dwa inne pliki źródłowe. Przede wszystkim odliczanie czasu zaimplementujemy w procedurze obsługi przerwania od Timera 0 zliczającego sekundy. Dodatkowo, w pliku *user.c* dodamy globalną zmienną do zapamiętania odliczanego czasu oraz funkcje: włączającą (*BL_on*) i wyłączającą (BL_off) podświetlenie. Definicje tych funkcji pokazano na **listingu 2a**. Ich deklaracje należy również dodać do pliku nagłówkowego *user.h*, który trzeba dołączyć dyrektywą *#include* w plikach *timer.c* i *main.c*. Plik *timer.c* należy włączyć do kompilacji. W tym celu klikamy na drzewie projektu i odznaczamy opcję *Exclude File(s) from Build*. Jak widać na listingu, funkcja *BL_on()* zamienia kierunek portu I/O na wyjściowy ustawia na nim „1", natomiast funkcja *BL_off()* przywraca poprzedni stan wyprowadzenia portu.

Następnie musimy zmodyfikować obsługę przerwania od Timera0. W pliku *timer.c*, po dołączeniu pliku *user.h* do funkcji *__interrupt void TIMER0_A0_ISR(void)*, dopisujemy fragment kodu pokazany na **listingu 2b**. Ja dopisałem swoje instrukcje począwszy od linii 271 po instrukcji *display.flag.update_time=1*. Zainicjowana przez funkcję *BL_on()* zmienna *BL_timer* jest co sekundę zmniejszana o 1 a podświetlenie jest wyłączane po zakończeniu odliczania.

Pozostaje dopisanie obsługi przycisku podświetlenia w pliku main.c. W procedurze *wakeup_event* odnajdujemy obsługę przycisku *s2* i pod nim dopisujemy kod z **listingu 2c**. W moim przypadku znalazł się on w linii 494.

Szybciej przez menu

Kolejna modyfikacja pozwoli nam szybciej korzystać z menu. W rozbudowanym menu Chronosa, przejście od aktualnie używanej funkcji do początku wymaga kilkukrotnego naciśnięcia przycisku, jednak najczęściej chcemy szybko powrócić do ekranu początkowego. Wprowadzimy więc modyfikację, która (jeżeli menu było aktywne dłużej niż przez określony czas) po naciśnięciu przycisku zmiany przeniesie nas do ekranu początkowego. Funkcja ta przydaje się też, jeśli podczas ćwiczeń z włączonym pomiarem pulsu chcemy np. szybko sprawdzić godzinę. Podobnie jak w poprzedniej modyfikacji, zmiany wprowadzamy w modułach: *user*, *timer* i *main*. Pliki *user.c* oraz *user.h* użyjemy do zadeklarowania dwóch globalnych zmiennych: *u8 MenuL1_in_use* i *u8 MenuL2_in_use*. W procedurze obsługującej jednosekundowe przerwanie z Timera0 dodamy fragment odliczający zadany czas (**listing 3a**).

Listing 4a
```
void display_heartrate(u8 line, u8 update)
{
   u8 * str;
   u8 LastZone;


   if (update != DISPLAY_LINE_CLEAR)
   {
      if (is_bluerobin())
      {
         str = itoa(sBlueRobin.heartrate, 3, 2);
         display_chars(LCD_SEG_L1_2_0, str, SEG_ON);

         LastZone = HR_Zone;

         if (sBlueRobin.heartrate>=HR_MAX)
         {
            HR_Zone = 3;
            display_symbol(LCD_SYMB_ARROW_UP, SEG_ON);
            display_symbol(LCD_SYMB_ARROW_DOWN, SEG_OFF);
         }
         else if (sBlueRobin.heartrate<=HR_MIN)
         {
            HR_Zone = 1;
            display_symbol(LCD_SYMB_ARROW_UP, SEG_OFF);
            display_symbol(LCD_SYMB_ARROW_DOWN, SEG_ON);
         }
         else
         {
            HR_Zone = 2;
            display_symbol(LCD_SYMB_ARROW_UP, SEG_OFF);
            display_symbol(LCD_SYMB_ARROW_DOWN, SEG_OFF);
         }

         if (LastZone != HR_Zone) //Zone change
            start_buzzer(2, CONV_MS_TO_TICKS(20), CONV_MS_
TO_TICKS(150));
      }
      else
      {
         display_chars(LCD_SEG_L1_2_0, (u8 *)"---", SEG_
ON);
         display_symbol(LCD_SYMB_ARROW_UP, SEG_OFF);
         display_symbol(LCD_SYMB_ARROW_DOWN, SEG_OFF);
      }
   }

   // Redraw whole screen
   if (!is_bluerobin())
   {
      if (update == DISPLAY_LINE_UPDATE_FULL)
      {
          display_symbol(LCD_ICON_HEART, SEG_ON);
      }
      else if (update == DISPLAY_LINE_CLEAR)
      {
         // Clear heart when not connected
         display_symbol(LCD_ICON_HEART, SEG_OFF);
         display_symbol(LCD_SYMB_ARROW_UP, SEG_OFF);
         display_symbol(LCD_SYMB_ARROW_DOWN, SEG_OFF);

      }
   }
}
```

Ostatniej modyfikacji dokonamy w funkcjach obsługi przycisków M1 i M2 w pliku main.c. Znajduje się tam linijka, która zmienia aktywny tryb poprzez przepisanie wskaźnika do następnej struktury menu (*ptrMenu_L1 = ptrMenu_L1->next;*). Jeżeli upłynął odliczany czas i zmienna *MenuLx_in_use* wyzeruje się, to przepisany zostanie wskaźnik do pierwszej pozycji menu. Wspomnianą linię zastępujemy kodem z **listingu 3b**. Przedstawiony fragment kodu przeznaczony jest dla obsługi przycisku M1. Analogiczną zmianę wprowadzamy w obsłudze przycisku M2.

Ćwiczymy w strefach

Do treningów wytrzymałościowych czy odchudzających bardzo przydatne jest kontrolowanie zakresu tętna, przy którym możemy ćwiczyć. Dlatego dodamy Chronosowi, w trybie *Heart rate*, możliwość ustalenia zakresu pulsu, a zegarek – oprócz wyświetlenia aktualnej strefy – powiadomi dźwiękiem o jej zmianie. Do sygnalizacji stosowane są strzałki, które znajdują się po lewej stronie górnego wyświetlacza. Strzałki zgaszone oznaczają pracę w wybranym zakresie. Świecąca strzałka oznacza przekroczenie zakresu w jednym z kierunków.

Niezbędne modyfikacje wprowadzimy w pliku *bluerobin.c*. Dla nowych funkcji zadeklarujmy trzy zmienne globalne typu u8: *HR_MAX*, *HR_MIN*, *HR_Zone*. W dwóch pierwszych jest zapamiętany zakres wyznaczonej strefy. Ich początkowe wartości należy ustalić w funkcji *reset_bluerobin()*. Obsługa wyświetlania strefy i sygnalizacji zmian jest dosyć prosta – wpisujemy ją do ciała funkcji *display_hartrate()* w pliku *bluerobin.c*. Zmienioną funkcję zamieszczono na **listingu 4a**. Z pozostałych funkcji warta omówienia jest *LCD display_symbol()*. Jako jej pierwszy parametr podajemy symbol, który chcemy obsłużyć, a następnie podajemy jedną z wartości *SEG_ON* lub *SEG_OFF*, w zależności od tego czy chcemy włączyć, czy wyłączyć segment. Symbole na wyświetlaczu są zdefiniowane w pliku *display.h* umieszczonym w folderze *driver*. Druga funkcja pozwala łatwo skorzystać z buzzera.

Listing 4b
```
void mx_bluerobin(u8 line)
{
#if REMEMBER_TX_ID == TRUE
  u8 i;

  // Reset chest strap ID
  sBlueRobin.cs_id = 0;

  display_chars(LCD_SEG_L1_2_0, (u8*)"CLR", SEG_ON);
  for (i=0; i<4; i++) Timer0_A4_Delay(CONV_MS_TO_
TICKS(500));
#endif

  //Function that allow to set the Training Zone
  u8 select;
  s32 hr_up;
  s32 hr_down;
  u8 * str;
  u8 * str1;

  // Clear display
  clear_display_all();

  // Convert global to local variables
  hr_up  = (s32)HR_MAX;
  hr_down= (s32)HR_MIN;

  // Init value index
  select = 0;

  // Init display
  // LINE1: HR_MAX
  // LINE2: HR_MIN

  str = itoa(hr_up, 3, 2);
  display_chars(LCD_SEG_L1_2_0, str, SEG_ON);

  str1 = itoa(hr_down, 3, 2);
  display_chars(LCD_SEG_L2_2_0, str1, SEG_ON);

  display_symbol(LCD_ICON_HEART, SEG_ON);

  // Loop values until all are set or user breaks  set
  while(1)
  {
    // Idle timeout: exit without saving
    if (sys.flag.idle_timeout) break;

    // M1 (short): save, then exit
    if (button.flag.m1)
    {

     // Check if hr_up is bigger than hr_down if not then
update hr_down
     if (hr_down >= hr_up) hr_down = hr_up - 1;
     // Copy local variables to global variables
     HR_MAX = (u8)hr_up;
     HR_MIN = (u8)hr_down;

    // Full display update is done when returning from
function
     break;
    }

    switch (select)
    {
     case 0: // Set HR_MAX
         set_value(&hr_up, 3, 1, 60, 250, SETVALUE_
DISPLAY_VALUE + SETVALUE_NEXT_VALUE, LCD_SEG_L1_2_0,
display_value1);
         select = 1;
         break;

     case 1: // Set HR_MIN
         set_value(&hr_down, 3, 1, 60, 250, SETVALUE_
DISPLAY_VALUE + SETVALUE_NEXT_VALUE, LCD_SEG_L2_2_0,
display_value1);
         select = 0;
         break;
    }


  }

  // Clear simulated button event
  button.all_flags = 0;
}
```

Listing 5a
```
u8 update_training(void)
{
  return (display.flag.update_time || display.flag.update_
stopwatch || display_training_triger);
}

…

// Line1 - Training
const struct menu menu_L1_Training =
{
  FUNCTION(s1_training),                              //
direct function
  FUNCTION(m1_training),                              // sub
menu function
  FUNCTION(display_training),                         //
display function
  FUNCTION(update_training),                          // new
display data
  &menu_L1_Alarm,
};
```

Listing 5b
```
// S2 button eve
nt-------------------------------------------------
// Activate user function for Line2 menu item
else if(button.flag.s2)
{
  if (ptrMenu_L1 == &menu_L1_Training)
  {
    s2_training(LINE2);
  }
  else
  {
    // Call direct function
    ptrMenu_L2->sx_function(LINE2);

    // Set Line2 display update flag
    display.flag.line2_full_update = 1;
  }

  // Clear button flag
  button.flag.s2 = 0;
}
```

Listing 5c
```
//if go out from training than Line2 is going back to
position1
if (ptrMenu_L1 == &menu_L1_Training)
{
  // Go to first menu entry
  ptrMenu_L2 = &menu_L2_Date;

  // Assign new display function
  fptr_lcd_function_line2 = ptrMenu_L2->display_function;

  // Set Line2 display update flag
  display.flag.line2_full_update = 1;

}

…

//if go into Training assign also LINE2 display functions
if (ptrMenu_L1 == &menu_L1_Training)
{
  fptr_lcd_function_line2 = &display_trainingL2;
  display.flag.line2_full_update = 1;

}
```

Listing 5d
```
str = itoa(sBlueRobin.heartrate, 4, 3);
display_chars(LCD_SEG_L1_3_0, str, SEG_ON);
..
display_chars(LCD_SEG_L1_3_0, (u8 *)"----", SEG_ON);
```

Listing 5e
```
//When training is Active counts 0..4 seconds
if (is_training())
{
  trainingDisplayCycle++;
  if (trainingDisplayCycle == 5)
    trainingDisplayCycle = 0;
}

//in training display last lap for some seconds
if (displayLastLap)
{
  displayLastLap--;
  if (displayLastLap == 0)
    updateTrainingL2 = 1;
}
```

Listing 5f
```
struct slap_time
{
  u8    hours;
  u8  minutes;
  u8  seconds;
  u8  hundrets;
};

...

// *****************************************
// Prototypes section
struct slap_time code_laptime(struct stopwatch sw_struct);
void display_lap_time(struct slap_time lapTime);
struct slap_time add_lap_time(struct slap_time lapTime1,
struct slap_time lapTime2);
void display_training(u8 line, u8 update);
void display_trainingL1(u8 line, u8 update);
void display_trainingL2(u8 line, u8 update);
u8 is_training(void);
void m1_training(u8 line);
void s1_training(u8 line);
void m2_training(u8 line);
void m2_long_training(u8 line);
void s2_training(u8 line);
void reset_training(void);
void new_lap(void);
```

Listing 5g
```
void display_training(u8 line, u8 update)
{
  u8 * str;
  if (trainingRecall && (recallLap != 0))
  {
    display_chars(LCD_SEG_L1_3_0, (u8 *)"LP", SEG_ON);
    str = itoa(recallLap, 2, 1);
    display_chars(LCD_SEG_L1_1_0, str, SEG_ON);

    display_lap_time(laps[recallLap-1]);
  }
  else
  if (displayLastLap)
  {
    display_chars(LCD_SEG_L1_3_0, (u8 *)"LP", SEG_ON);
    str = itoa(currentLap-1, 2, 1);
    display_chars(LCD_SEG_L1_1_0, str, SEG_ON);

    display_lap_time(laps[currentLap-2]);
  }
  else
  {
    display_trainingL1(LINE1, update);
    display_trainingL2(LINE2, update);
  }

  display_training_triger = 0;
}
```

Jako parametry funkcji *start_buzzer()* podajemy trzy wartości: liczbę sygnałów dźwiękowych, długość trwania sygnału oraz czas trwania przerwy. Pomocne jest tu makro *CONV_MS_TO_TICKS*. Aby go użyć należy dołączyć plik *buzzer.h* do kompilacji instrukcją *#include*.

Aby umożliwić użytkownikowi zmianę ustalonych wartości *HR_MAX* i *HR_MIN* zastosujemy ten sam mechanizm, który służy do ustawiania zegarka, daty czy godziny alarmu. Programiści TI przygotowali w tym celu funkcję *set_value()*. Jako jej argumenty podajemy kolejno: nazwę zmiennej, liczbę cyfr, liczbę wiodących zer do wygaszenia, dolne ograniczenie, górne ograniczenie, tryb zmiany, pozycję na wyświetlaczu, funkcję wyświetlającą. Zmodyfikowaną funkcję *mx_bluerobin* pokazano na **listingu 4b**. Do zmiany używamy zmiennych lokalnych. Dodatkowo, wszystkie liczbowe parametry *set_value()* powinny być typu s32. Przy przepisywaniu zmiennych wykonujemy więc rzutowanie typów. Po naciśnięciu przycisku M1, który zatwierdza zmiany aktualizujemy zmienne globalne po uprzednim sprawdzeniu czy *HR_MAX* jest większe od *HR_MIN*.

Porządki

Oprogramowanie demonstracyjne Chronosa przygotowano tak, aby zilustrować ogrom możliwości zestawu oraz pracę wszystkich peryferiów wbudowanych i dołączonych do mikrokontrolera. W zegarku większość z tych funkcji nie jest potrzebna. Dodatkowo, oprogramowanie zajmuje sporą część pamięci Flash i dodanie kolejnych funkcji wymaga usunięcia niektórych z istniejących.

Na **rysunku 6** pokazano nową strukturę menu, które będzie miał Chronos. Na listingu 1b umieszczono przykład struktury menu w pliku *menu.c*. Zmiana polega jedynie na modyfikacji ostatniej wartości struktury, czyli wskaźnika na następną pozycję. Dotychczasową kolejkę dla linii górnej *[Time]* –> *Alarm* –> *Temperature* –> *Altitude* –>

Listing 5h
```
void sx_stopwatch(u8 line)
{
  // S2: RUN, STOP
  if(button.flag.s2)
  {
    if (sStopwatch.state == STOPWATCH_STOP)
    {
      // (Re)start stopwatch
      start_stopwatch();
    }
    else
    {
      // Change lap
      // done in ports.c
    }

  }
}
```

Listing 5i
```
u8 is_stopwatch(void)
{
  return ((sStopwatch.state == STOPWATCH_RUN) && (ptrMenu_
L1 == &menu_L1_Training));
}
```

Listing 5j
```
// -------------------------------------------------------
// S2 button IRQ
else if (IRQ_TRIGGERED(int_flag, BUTTON_S2_PIN))
{
  // Filter bouncing noise
  if (BUTTON_S2_IS_PRESSED)
  {
    button.flag.s2 = 1;

    // Generate button click
    buzzer = 1;

    // Faster reaction for stopwatch stop button press
    if ((is_stopwatch()) && (trainingRecall == 0))
    {
      new_lap();
      button.flag.s2 = 0;
    }
  }
}
```

Heart rate –> *Speed* –> *Acceleration* zmieniamy na *[Time]* –> *Training* –> *Heart rate* –> *Alarm*. Pamiętajmy, że *Alarm* musi wskazywać z powrotem *Time*.

Kolejka dolna również będzie zawierać cztery pozycje. Istniejące *[Date]* –> *Stopwatch* –> *Battery* –> *ACC* –> *PPT* –> *SYNC* –> *Calories/Distance* zmieniamy na *[Date]* –> *PPT* –> *SYNC* –> *Battery*. Funkcję pokazywania napięcia baterii przesunięto na koniec jako najrzadziej używaną.

W celu zaoszczędzenia pamięci postanowiono usunąć funkcje obsługi czujnika ciśnienia. Należy więc dołączyć do kompilacji plik *vti_ps.c* z katalogu *driver*. Można usunąć ten plik, jednak wtedy należałoby usunąć także wszystkie odwołania do niego. Łatwiej jest kod wszystkich funkcji ująć w komentarz, a dla funkcji zwracających wartości dopisać w ostatniej linii *return (1)*. Dodatkowo, należy zablokować w funkcji *init_global_variables* znajdującej się *main.c*, wywołanie funkcji *reset_altitude_measurement*, aby nie wywoływać funkcji pomiaru ciśnienia.

Menu trening

Poszerzając możliwości funkcjonalne zegarka dodano opcję treningu z możliwością rejestracji czasów poszczególnych odcinków. W tym celu utworzono nową pozycję menu – *Trening*. Utrudnieniem w dodaniu nowej pozycji menu są ograniczenia wyświetlacza. Należy podać użytkownikowi kilka wielkości: aktualny puls, aktualny numer okrążenia, czas aktualnego okrążenia, czas całkowity. W tym celu potrzebujemy większej liczby przycisków, niż przewidziano do obsługi modułu. *Trening* umieścimy w górnym łańcuchu menu, a więc przełączany będzie klawiszem M1. W górnej linii będzie wyświetlany aktualny puls oraz raz na 5 sekund numer aktualnego okrążenia jako LPXX. W dolnej linii będzie wyświetlany stoper włączany przyciskiem S2. Ponowne przyciśnięcie przycisku spowoduje przełączenie

na nowe okrążenie. Pomiar czasu będzie zatrzymywany przyciskiem M2. Dłuższe jego przytrzymanie spowoduje wyzerowanie stopera. Po zmianie okrążenia na chwilę zamrożony zostanie na wyświetlaczu czas okrążenia.

W czasie pomiaru wciśnięcie przycisku S1 spowoduje wyświetlenie w dolnej linii całkowitego czasu treningu oraz napisu *TOTAL* pod dolną linią. Jeśli stoper nie jest aktywny, to przytrzymanie przycisku M1 spowoduje przejście w tryb odczytu zapisanych okrążeń (o ile oczywiście je zapamiętano) sygnalizowane wyświetleniem się znaczka „®". Okrążenia w pamięci zmieniamy prawymi przyciskami, tak jak w przypadku ustawiania wartości. Ograniczona liczba przycisków powoduje, że pomiar pulsu należy włączyć osobno w menu *Heart rate*.

Zmiany można prześledzić w plikach zamieszczonych na płycie CD_EP11/2010 w katalogu z dodatkowymi materiałami do artykułu. Aby umożliwić działanie nowego trybu, to zmiany przeprowadzono także w pliku *main.c* oraz modułach: *menu*, *timer*, *stopwatch* i *ports*. We wszystkich tych plikach musimy pamiętać o dołączeniu pliku *timer.h*.

Rodzaj przeprowadzanych zmian powoduje, że aby rozpocząć testowanie musimy wprowadzić ich dość dużo. Dodajemy nową strukturę menu *menu_L1_Training* oraz funkcję *update_training*, jak na **listingu 5a**. Pozostałe funkcje odświeżające są wywoływane pojedynczą zmienną. W tym przykładzie będą to trzy zmienne. Pierwsza jest używana przez moduł *bluerobin* do wyświetlania pulsu. Drugiej używa moduł stopera, a dzięki trzeciej będzie można uruchomić odświeżanie wyświetlacza.

W pliku *training.c* jest funkcja realizująca inicjalizację zmiennych o standardowej nazwie *reset_training()*. Tę funkcję należy wywołać w *init_global_variables()* w *main.c*. Górne przyciski są przeznaczane do obsługi treningu poprzez strukturę menu. Dolne będą używane do jednego z elementów dolnego menu. Na **listingu 5b** zamieszczono przerobiony fragment funkcji *wakeup_event()* do obsługi przycisku S2. Analogiczne przeróbki należy wprowadzić przy obsłudze przycisku M2 przy krótkim albo długim przyciśnięciu.

Krótkie przyciśnięcie przycisku M1 powoduje zmianę menu. Jeśli dojdziemy do menu treningu, należy również odświeżyć dolną linię. Natomiast jeśli wychodzimy z treningu, to linii drugiej trzeba przywrócić pierwotne działanie. Dla uproszczenia: L2 wróci do swojej pierwszej pozycji. Dwa fragmenty programu umieszczono na **listingu 5c**. Należy zwrócić uwagę na miejsce wstawiania kodu, gdyż te dwie części oddziela fragment przypisania nowej wartości do wskaźnika.

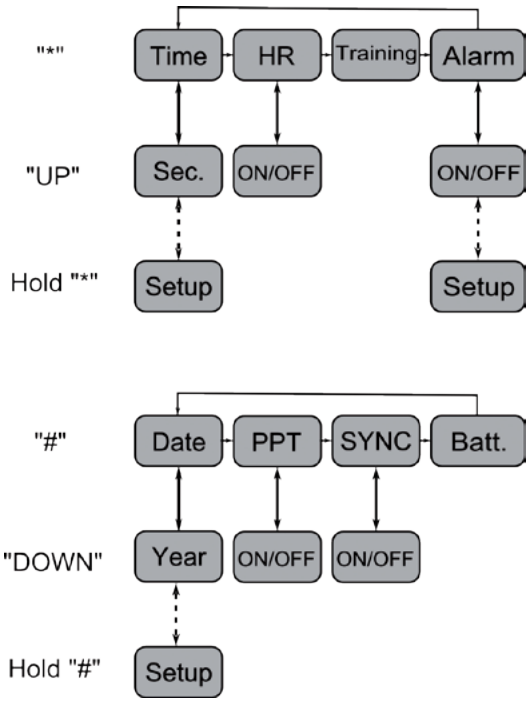
Niewielkiej modyfikacji wymaga również funkcja *display_heartrate()* modułu *bluerobin*. Modyfikacje umieszczono na **listingu 5d**. Wyświetlane zamiennie puls i aktualne okrążenie zajmują 4 znaki wyświetlacza, więc modyfikacja pozwala na wygaszenie pierwszego znaku („L").

W pliku *timer.c* w przerwaniu co 1 sekundę dodamno dwa liczniki. Jeden będzie zliczać sekwencje zmiany wyświetlania numeru okrążenia i pulsu, a drugi zatrzymanie wyświetlania po zmianie okrążenia (**listing 5e**).

Training.c

Na **listingu 5f** przedstawiono definicję struktury *slap_time* zawartą w *training.h* oraz funkcje modułu *training*. Moduł *stopwatch* przechowuje aktualną wartość czasu w zmiennej typu *string*. Jest to nieefektywne rozwiązanie, gdy trzeba zapisać czas oraz dokonać na nim operacji matematycznych. Pierwsze trzy funkcje dokonują przekształceń między tymi formami zapisu oraz pozwalają na dodanie dwóch struktur *slap_time*. Funkcje *mX_training* oraz *sX_training* umożliwiają obsługę przycisków wywoływaną w pętli głównej programu w funkcji *main()*.

Do obsługi wyświetlania zastosowano funkcję *display_training()* pokazaną na **listingu 5f**. Dostępne są trzy podstawowe tryby wyświetlania: tryb wyświetlania danych z pamięci, wyświetlanie czasu ostatniego okrążenia oraz tryb zwykły, w którym używano odrębnych funkcji dla linii L1 i L2. Te ostatnie również nie wyświetlają tylko jednej informacji. L1 wyświetla naprzemiennie puls oraz aktualne okrążenie, natomiast L2 czas stopera oraz czas całkowity po wciśnięciu S1. Funkcje te oczywiście odwołują się do oryginalnych procedur wyświetlających z modułów *bluerobin* oraz *stopwatch*.



Rysunek 6. Budowa menu zegarka po modyfikacjach

Ostatnią z funkcji jest *new_lap*. W niej zerujemy stoper zapamiętując wcześniej czas w tablicy *laps* oraz zwiększamy wartość *totalTime*.

stopwatch.c

Dużym zmianom uległ moduł *stopwatch*. Do funkcji *display_stopwatch* dodałem możliwość ręcznego wyzwolenia odświeżenia wyświetlacza za pomocą zmiennej *updateTrainingL2*. Zmiany te oznaczyłem komentarzem „EP update". Jedną ze zmienionych funkcji jest wywoływana przyciśnięciem przycisku S2 *sx_stopwatch* (**listing 5g**). W tej wersji przycisk ten nie będzie odpowiedzialny za zatrzymywanie stopera, więc funkcję należy usunąć i przenieść ją do *m2_training*. W funkcji *reset_stopwatch()* dodajemy zerowanie zmiennej *currentLap*, a w *is_stopwatch()* zmienił się aktywny tryb menu, więc modyfikujemy ją w sposób pokazany na **listingu 5i**.

ports.c

Zatrzymywanie stopera było wywoływane w oryginalnym oprogramowaniu w procedurze obsługi przerwania z przycisku S2, aby maksymalnie skrócić czas reakcji. Ponieważ przenieśliśmy tę funkcję na przycisk M2, a S2 ma teraz inną funkcję (jednak również wymagającą szybkiej reakcji), musimy zmodyfikować obsługę przerwań. Funkcję stopera z obsługi przycisku S2 przenosimy do M2, a dla S2 wstawiamy kod z **listingu 5j**.

Podsumowanie

Wprowadzenie ostatniej zmiany i obejście pewnych ograniczeń nie jest proste. Pozwala jednak na zrealizowanie założeń i zegarek podczas testów nie przysparzał żadnych problemów. Zachęcam Czytelników do rozbudowy tego oprogramowania oraz przygotowywania innych projektów w oparciu o ez430-Chronos.

Grzegorz Latocha
glatocha@gmail.com