PATIENT MONITORING DEVICE

Dokumentacja do projektu.

Bartłomiej Cerek
Arkadiusz Chudy

Spis treści

Abstrakt

Przedstawiony dokument jest wstępna dokumentacją do projektu „Patient Monitoring Device”. Projekt jest tworzony w ramach konkursu o tematyce IoT organizowanego przez firmę Nordic Semiconductor.

1. Warstwa sprzętowa

Akcelerometr

Wykorzystywany jest moduł LSM303D, komunikacja z nim zachodzi przez interfejs I2C (TWI). Do prawidłowego działania konieczne są 4 połączenia: Vin (3,3V), GND oraz SCL i SDA. Dwa pierwsze należy podpiąć do wyjść Vdd i GND płytki deweloperskiej nrf52. Dwa ostatnie należy wpiąć tak jak jest to ustawione w pliku konfiguracyjnym config.h. Konfiguracja wykorzystywana przez prototyp (piny podpięte z użyciem przewodów, bez lutowania) załączona jest na poniższym rysunku, numery pinów odpowiadają numerom na płytce.

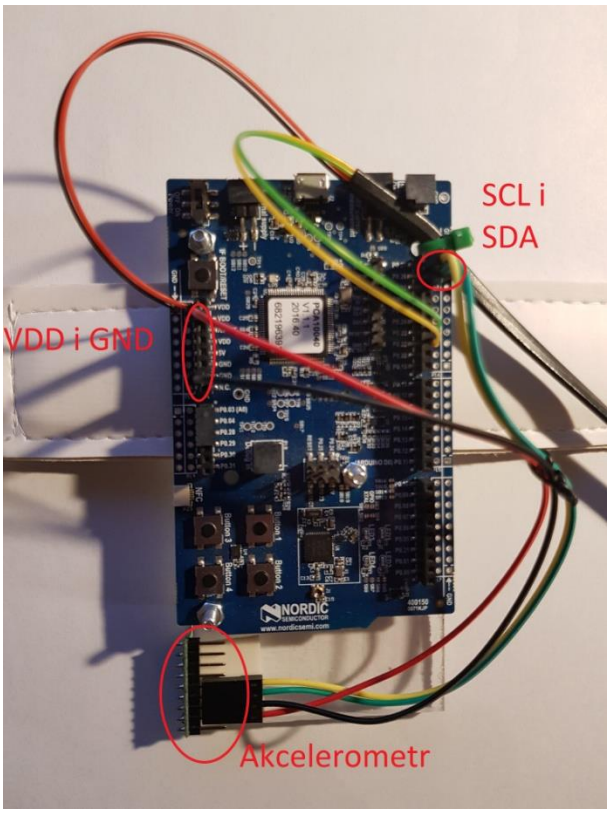
```
//Akcelerometr konfiguracja pinow

#define LSM303D_SCL 26
#define LSM303D_SDA 2

//Heart Rate Sensor konfiguracja pinow

#define MAX30105_SCL 22
#define MAX30105_SDA 23
```

Rysunek 1 Fragment pliku konfiguracyjnego określający piny SCL i SDA



Rysunek 2 Schemat podpięć w prototypie

Moduł musi być zainicjowany przez ustawienie 'jedynek' w rejestrach odpowiadających za odblokowanie akcelerometrów w kierunkach X,Y i Z.
Dane pozyskane z konwertera ADC należy złożyć (wartość w każdym kierunku podawana jest jako dwa bajty high i low), a następnie przeliczyć na procentową część przyspieszenia ziemskiego wg wzoru x = a_x * 2.0 / 32678.0.
Na ich podstawie możemy wnioskować o pozycji pacjenta oraz o ewentualnym wystąpieniu upadku.

Czujnik przepływu krwi

Wykorzystywany jest moduł MAX30105 - przemysłowy czujnik cząsteczek na breakeout. komunikacja z nim zachodzi przez interfejs I2C (TWI). Do prawidłowego działania konieczne są 4 połączenia: Vin (3,3V), GND oraz SCL i SDA. Dwa pierwsze należy podpiąć do wyjść Vdd i GND płytki deweloperskiej nrf52. Dwa ostatnie należy wpiąć tak jak jest to ustawione w funkcji konfiguracyjnej w pliku config.h. Domyślna konfiguracja wykorzystywana przez prototyp (piny przylutowane) załączona jest na poniższym rysunku, numery pinów odpowiadają numerom na płytce.

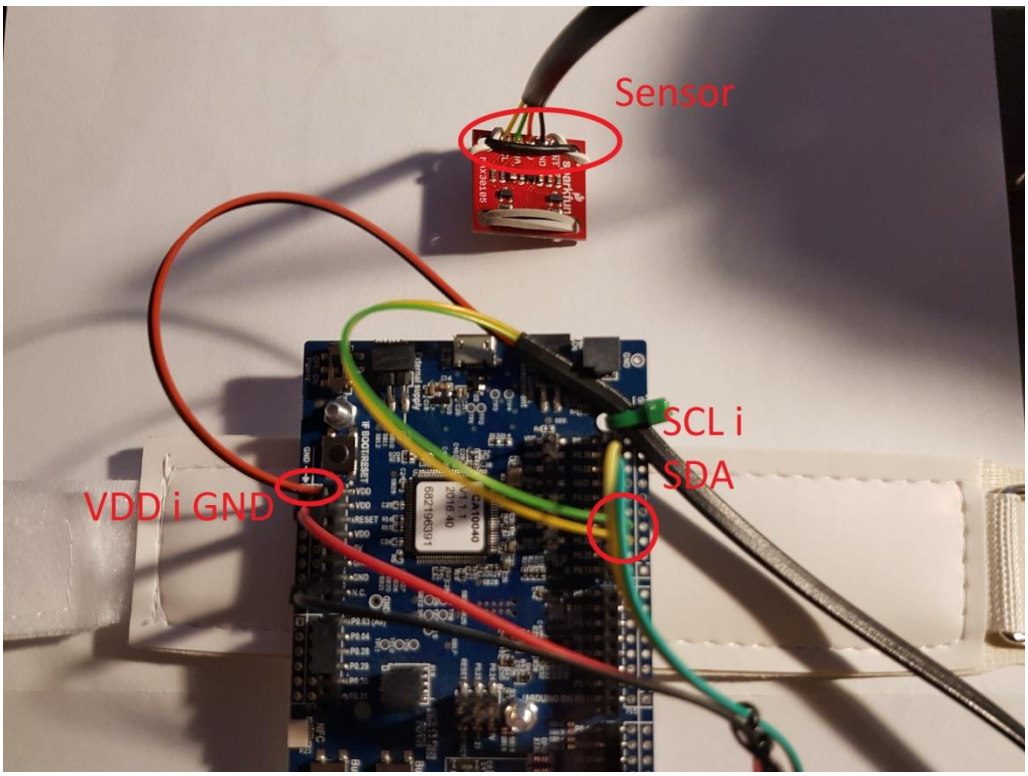
```
//Akcelerometr konfiguracja pinow

#define LSM303D_SCL 26
#define LSM303D_SDA 2

//Heart Rate Sensor konfiguracja pinow

#define MAX30105_SCL 22
#define MAX30105_SDA 23
```

Rysunek 3 Fragment kodu określający piny SCL i SDA



Rysunek 4 Schemat podpięć w prototypie

Czujnik wyposażony jest w diodę emitującą promieniowanie czerwone widzialne i IR, oraz możliwość pomiaru natężenia odbitej części promieniowania. Ponieważ krew natlenowana pochłania większą ilość promieniowania podczerwonego, a serce pompuje krew pulsacyjnymi falami, na podstawie zmian odczytu ADC modułu wnioskować można o pulsie i przepływie krwi pacjenta. Ważną kwestią jest, fakt że moduł musi być przytwierdzony bezpośrednio do skóry (niekoniecznie przy dużym naczyniu krwionośnym), a jego nacisk musi być STAŁY. Najlepiej uzyskać ten efekt stosując jakiegoś rodzaju elastyczny materiał (w prototypie jest to gumka elastyczna).
Jeżeli przebieg zmian wartości uzyskiwanej z ADC pokażemy na wykresie to ilość pików w ciągu 60 sekund (lub proporcja z krótszego czasu) będzie stanowiła puls pacjenta. Dane odbierane są w paczkach 4 bajtowych (unsigned int 16), z KOLEJKI w urządzeniu. Aby ograniczyć błędy i zmniejszyć ilość

danych ustawiono urządzenie na uśrednianie 8 wartości, oznacza to, że do kolejki przesyłamy średnią z 8 ostatnich wartości pobranych przez czujnik.
Układ scalony kolejkuje wartości które odczytuje się przez wielokrotne odczytanie jednego rejestru. W przeciwieństwie do klasycznego odczytu (wykorzystywanego na przykład w module akcelerometru) ostatnia wartość nie jest stale odświeżana w odpowiednim rejestrze. Przy wywołaniu funkcji odczytującej wartość HRS tak naprawdę odczytywana jest cała kolejka w celu zwolnienia miejsca i wybierana ostatnia próbka. Ilość nagromadzonych próbek zmienia się nieco w czasie, aby uniknąć dynamicznej alokacji pamięci zastosowano bufor na 99 próbek.

Montaż prototypu

W celu uproszczenia budowy prototypu nie wykorzystano żadnej zewnętrznej obudowany na płytkę deweloperską, zamiast tego przykręcono ją trzema 2,5 mm śrubkami do odpowiednio wyciętego wcześniej kawałka plexi. Aby płytki nie stykały sie bezpośrednio zastosowano plastikowe tulejki o wysokości 5 mm. Przewody potrzebne do czujnika pulsu przylutowano do odpowiadających im pinów, schodzą się one w 60 cm kabel telekomunikacyjny (4 przewodowy) który prowadzi do czujnika. Akcelerometr został połączony zwyczajnymi przewodami ze względu na fakt że nie będzie poddawany dużym obciążeniom i znajduje się blisko mikrokontrolera. Jego breakout board przyklejono do podstawkowego kawałka plexi.

2. BLE

Serwisy i charakterystyki

Wszystkie wykorzystywane serwisy, charakterystyki i UUID są zdefiniowane i obsłużone w plikach our_service.h i our_service.c . Wszystkie charakterystyki są typu notify aby ułatwić ich odczyt.
Serwis o UUID ACC0 przechowuje dane odbierane z akcelerometru. Posiada on 3 charakterystyki 1ACC, 2ACC i 3ACC odpowiadające za kolejno osie X, Y i Z.
Serwis o UUID EE00 przechowuje dane odbierane z czujnika cząsteczek MAX30105. Posiada on charakterystykę EE10 w której przechowywane są ostatnie wartości wyciągnięte z kolejki modułu.

3. Warstwa aplikacji

Baza danych

W projekcie wykorzystujemy zaprojektowaną pod nasze wymagania bazę danych. Baza (chwilowo testowa – w końcowej fazie projektu przejdziemy na wersję finalną) znajduje się na własnym serwerze w prywatnej sieci. Serwer bazodanowy opiera się na MSSQL 2014, połączenie z bazą możliwe jest z wykorzystaniem VPNa. Baza zawiera 6 tabeli : jedna główna (Patients) oraz 5 tabeli pomiarowych do których zbieramy dane z pomiarów (HeartRate, Temperatura, 3 x Akcelerometr – na każdą z osi).

Każda z tabeli pomiarowych posiada 7 kolumn – jedna to PrimaryKey dla danej paczki pomiarowej, druga to ForeignKey mówiący o ID badanego pacjenta, pozostałe 5 kolumn przeznaczone jest na zachowanie wartości paczki danych. Typy kolumn pod paczkę danych to liczby zmiennoprzecinkowe o

pojedynczej precyzji - „float” – wartości otrzymywane z sensorów przerabiamy na dane zmiennoprzecinkowe o pojedynczej precyzji używając do tego programu napisanego w LabVIEW.

W projekcie wysyłamy dane co 1 sekundę w paczkach 5 pomiarów. Taki efekt spowodowany jest tym, aby ograniczyć ruch sieciowy.

Do połączenia się z bazą oraz wysyłania odpowiednich kwerend używamy wcześniej napisanej przez nas biblioteki. Biblioteka „NordicDatabaseDLL” została napisana w VisualStudio w technologii .NET przy użyciu języka C#. Dzięki zaimplementowanym w niej funkcjom możemy otwierać i zamykać połaczenie z bazą oraz wykonywać SELECTy i INSERTy na odpowiednich tabelach, podając tylko dane wejściowe w postaci tabeli liczb zmiennoprzecinkowych o pojedynczej precyzji - „float” o długości 5.

```
#region Insert/Select

private void Insert(float[] data, DBTableEnum type)
{
    string tableName = type.ToString();
    string columnName = SwitchToColumnName(type);

    float data1 = data[0];
    float data2 = data[1];
    float data3 = data[2];
    float data4 = data[3];
    float data5 = data[4];

    string sql = string.Format("INSERT INTO {0} (patient_ID, {1}1, {1}2, {1}3, {1}4, {1}5) VALUES ({2}, {3}, {4}, {5}, {6}, {7});",
        tableName, columnName, patientNumber, data1, data2, data3, data4, data5);

    using (SqlTransaction transaction = connection.BeginTransaction())
    {
        try
        {
            SqlCommand cmd = new SqlCommand(sql, connection, transaction);
            cmd.ExecuteNonQuery();
            transaction.Commit();
        }
        catch(Exception)
        {
            transaction.Rollback();
        }
    }

}
```

Rysunek 5 Metoda dla INSERTa z biblioteki

```
private DataTable Select(DBTableEnum type, int patientID)
{
    string tableName = type.ToString();
    string columnName = SwitchToColumnName(type);
    string idColumn = RefactorToIdString(columnName);

    string sql = string.Format("SELECT TOP 1 {0}1, {0}2, {0}3, {0}4, {0}5 FROM {1} WHERE patient_ID = {2} ORDER BY {3} DESC",
        columnName, tableName, patientID, idColumn);

    SqlCommand cmd = new SqlCommand(sql, connection);

    DataTable dt = new DataTable();
    dt.Load(cmd.ExecuteReader());

    return dt;
}

#endregion
```

Rysunek 6 Metoda dla SELECTa z biblioteki

INSERT do bazy wykonywany jest w momencie zebrania 5 kolejnych pomiarów. Taka paczka, wraz z ID pacjenta dla którego wykonywany jest pomiar, wysyłana jest do bazy po wcześniejszym przygotowaniu kwerendy (SQL).

Metoda do wykrywania upadków

W bibliotece została zaimplementowana specjalistyczna klasa **FallChecker** odpowiedzialna za wykrywanie upadku pacjenta. Klasa posiada metodę, która pobiera podane argumenty :

- tabele X, Y i Z akcelerometru
- różnicę wartości przy której wykrywany jest upadek
- ilość próbek z których liczona ma być różnica do weryfikacji

Algorytm do rejestracji składu się z kilku etapów :

1. Jeżeli podane długości tabeli X, Y, Z są mniejsze od zadanej wartości próbek, to zwracana jest wartość **FALSE – upadek nie nastąpił**.
2. Jeżeli podane długości są odpowiednie (większe od zadanej wartości próbek) następują kolejne obliczenia.
3. Algorytm analizuje wartości znajdujące się w każdej z tabel zaczynając od indeksu 0. Analizowane jest tyle próbek ile zostało podane w zadanej wartości próbek w funkcji.
4. Liczona jest delta między każdą z wartości bezwzględnych z próbek. Jeżeli któraś z obliczonych delt jest większe od tej zadanej – algorytm zwraca wartość **TRUE – upadek nastąpił.**
5. Po analizie danej ilości próbek (od indeksu 0) algorytm przeskakuje do kolejnego indeksu (jeżeli jest to możliwe) i wykonuje dalsze obliczenia. Akcja ta powtarzana jest dla całej tablicy.

Metoda do wykrywania upadków zaimplementowana jest tak w aplikacji na PC jak i aplikacji mobilnej. Obydwu aplikacjom przy wykryciu upadku pacjenta towarzyszy zmiana koloru interfejsu na odcień czerwieni. Gdy upadek nie jest już wykrywany – kolor powraca do wyjściowego.

Odbieranie danych z akcelerometru – UART (debugging)

Do odbierania danych po magistrali UART wykorzystaliśmy środowisko LabVIEW oraz bibliotekę NI-VISA. Program odbiera dane wysyłane przez nRF52 jako kolejne łańcuchy znaków – stringi. Następnie program dokonuje obróbki – dostosowuje łańcuch znaków – string do późniejszej zamiany na 32-bitową liczbę zmiennoprzecinkową o pojedynczej precyzji (w zamyśle – możliwa implementacja z podwójna precyzją).

W pętli głównej programu znajduje się klaster tabel (typedef), do których dane z akcelerometru (już po zamianie na liczby zmiennoprzecinkowe o pojedynczej precyzji - „float") są wrzucane. Wszelkie błędy (błędny łańcuch znaków – string, obcięte dane, błąd konwersji) są eliminowane. Gdy taki błąd się pojawia – brana jest poprzednia dana z tabeli, tak by zachować ciągłość akwizycji.

Program jest także przystosowany do wyświetlania danych z 3 osi akcelerometru na jednym, wspólnym wykresie. Wartości z tabel przy każdym pomiarze są wyciągane i umieszczane na plocie. Pozwala nam to nie tylko zbierać i przetwarzać dane, ale również je wizualizować.

W projekcie znajduję się także subVI nazwany FGV. Jest to nasza funckjonalna zmienna globalna, która (już przygotowana do odbierana wszystkich danych, łącznie z heartRatem) wykorzystując predefiniowany klaster danych oraz niezainicjowany rejestr przesuwny, wkłada dane do tabel klastra i daje output w zależności od rodzaju przychodzącej do niej danej (output -> tabela XAccelero jeżeli przychodząca dana to wartość z osi X akcelerometru).

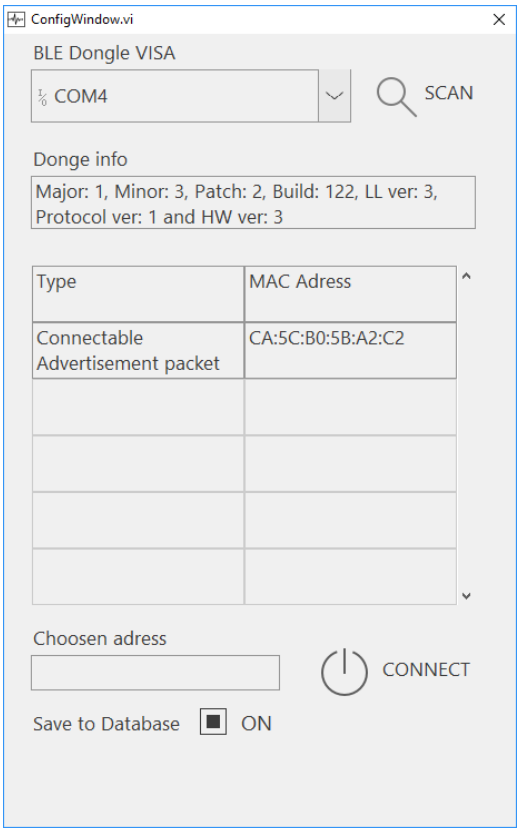
Na wyjściu wcześniej opisanej FGV powstanie kolejna funkcja sprawdzająca, czy długość tabeli jest równa 5. Jeżeli tak -> rozpoczynamy komunikacje bazodanową. Jej implementacja (z wykorzystaniem biblioteki NordicDatabaseDLL) powstanie z wykorzystaniem metod .NET w środowisku LabVIEW.

Przedstawiony sposób odbierania danych poprzez UART jest tylko sposobem testowania oprogramowania. Domyślnie wszystko zostanie przeniesione na BLE. Do komunikacji po bluetoothie wykorzystanie zostanie bluetooth dongle (zbieranie danych po COM porcie) oraz BLE Toolkit w środowisku LabVIEW.

Główny program na PC

Program został zrealizowany przy użyciu środowiska LabVIEW.

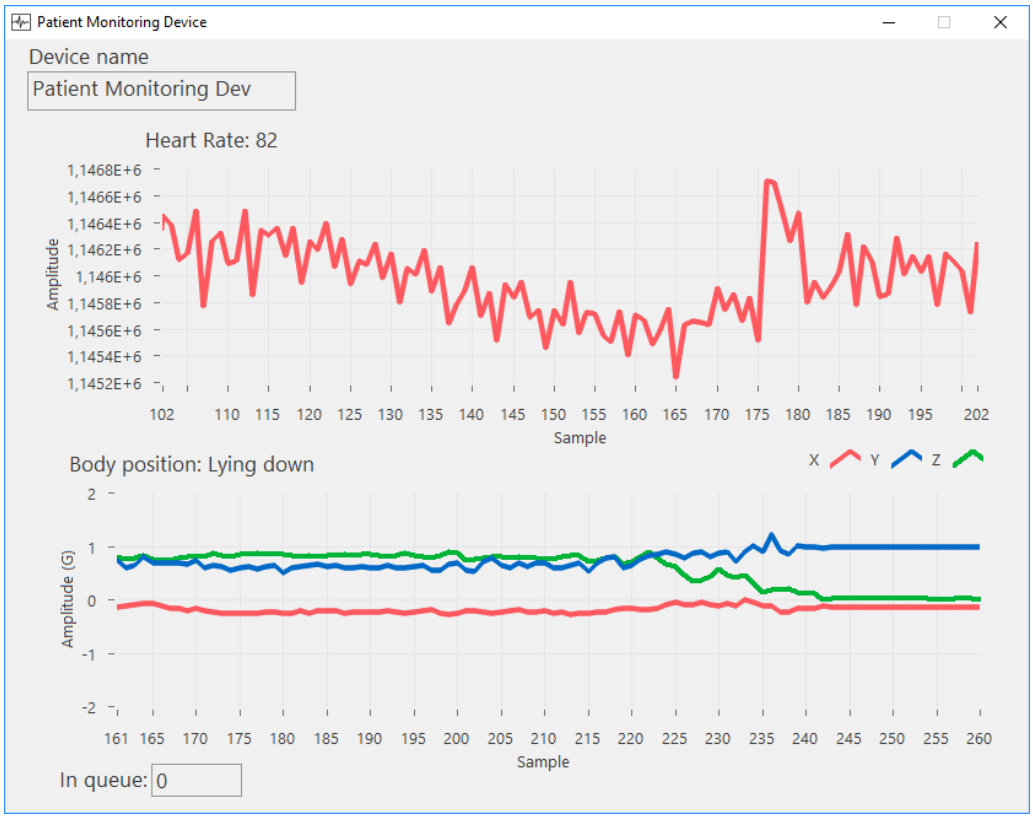
Przy włączeniu aplikacji pojawia się pierwsze okno konfiguracyjne służące za ekran połączenia z zewnętrznymi urządzeniami BLE. Użytkownik wybiera COM port na którym znajduje się BLE Dongle, po którym odbierane są wszelkie dane. Po wyborze urządzenia i kliknięciu **Scan** program po chwili przedstawia listę wszystkich urządzeń bluetooth znalezionych w niedalekiej okolicy. Użytkownik wybiera nRF52 o adresie : **CA:5C:B0:5B:A2:C2**, ustawia opcje czy program ma wykonywać wpisy do zewnętrznej bazy danych (czy być tylko wersją do wizualizacji przebiegu parametrów pacjenta) i przechodzi do dalszej części programu.



Rysunek 7 Front okna konfiguracyjnego

Na ekranie głównym aplikacji znajdują się dwa wykresy : jeden odpowiada za wyświetlania 3 osi akcelerometru, drugi wyświetla tętno. Wykres HR (heart rate) wyświetla dane w momencie jeżeli ich wielkości są zgodne z parametrami ludzkimi (zabezpieczenie przed wyświetlaniem szumów i nieprawidłowych danych). Po pewnym czasie HR jest kalkulowany i wyświetlana jest jego dokładna wartość w indykatorze nad wykresem. Użytkownik może kliknąć w wykres i odświeżyć jego przebieg.

Wykres akcelerometru wyświetla odebrane dane na 3 osiach. Pokazywana jest także pozycja ciała człowieka. Jeżeli pacjent zmieni swoją pozycje gwałtownie (np. upadnie), tło zmienia kolor na czerwony – jest to informacja dla lekarza, iż badanemu mogło się coś stać.



Rysunek 8 Front głównego okna programu

Na dole głównego opcja jest wyświetlana ilość elementów, które są zakolejkowane i czekają na wysłanie do zewnętrznej bazy danych.

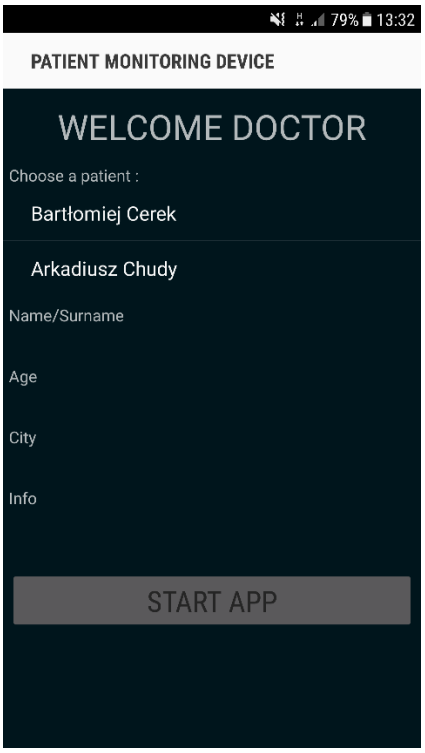
4. Aplikacja mobilna

Architektura

W ramach projektu stworzona została aplikacja mobilna osadzona na systemie Android. Aplikacja została stworzona w programie Visual Studio wykorzystując framework Xamarin oraz język C#.

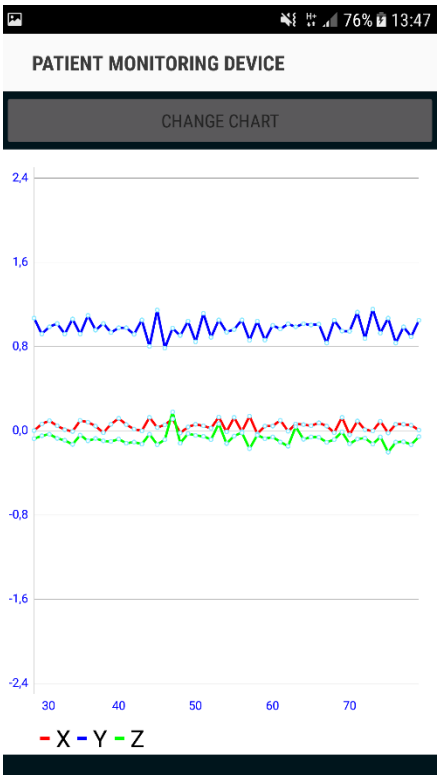
Aplikacja składa się z 3 activities :

- **WebConnection** – activity będą formą ekranu przejściowego, jeżeli urządzenie nie jest połączone z internetem, to ekran przełącza się na główny View tego activity
- **InfoActivity** – ekran startowy aplikacji, użytkownik może wybierać badanego pacjenta z ich listy, po wyborze na ekranie pojawiają się informacje o danej osobie, po kliknięciu określonego guzika użytkownik przechodzi do ekranu głównego aplikacji



Rysunek 9 Front dla InfoActivity

- **MainActivity** – activity odpowiedzialne za wykreślanie wykresów akcelerometru o hearRate'a na ekranie urządzenia, użytkownik ma możliwość przełączania się pomiędzy wykresami w czasie rzeczywistym



Rysunek 10 Front dla MainActivity

Główne activity w programie korzysta z dwóch klas – **HeartPlot** oraz **AcceleroPlot**. Te klasy są odpowiedzialne za akwizycję danych z bazy dla danego typu wykresu oraz dodawanie ich do swojego pola danych wykresu. W głównym programie do wykresu przypisywane są dane z jednej z tych klas i odświeżane w czasie rzeczywistym.

Użytkownik chcąc zmienić wyświetlanie danego wykresu może uczynić to klikając w guzik **Change Chart** znajdujący się w górnej części ekranu. Nie występuje tutaj żadne przekierowanie do innego acitivity, gdyż tylko dane wykresu oraz granice jego osi Y są podmienianie na wartości z danej klasy Plot.

Sposób pobierania danych z bazy

W aplikacji zostały zaimplementowane struktury, które na bieżąco badają ilość rekordów znajdujących się w bazie danych. Gdy w bazie wystąpi INSERT, ostatnia wartość indeksu pomiaru weryfikowana jest z zapisana ówcześnie wartością w programie. Jeżeli nowo otrzymana wartość jest większa – następuje operacja SELECT i dane są pobierane a następnie obrabiane i dodawane do wykresu.

Jeżeli aplikacja zostanie włączona, lecz urządzenie (nRF 52) nie jest włączone, to ekran główny aplikacji zostanie niezainicjowanym wykresem. Dopiero w momencie zmian ilości rekordów w bazie dochodzi do dalszych akcji.

Monitorowanie dostępu do internetu

W aplikacji zostały także zaimplementowane struktury, które na bieżąco monitorują stan sieciowy urządzenia.

Jeżeli przy włączeniu aplikacji urządzenie nie miało dostępu do internetu, to użytkownik zostanie poproszony o jego włączenie, gdyż jest on niezbędny do działania kolejnych komponentów.

W momencie gdy użytkownik znajduje się na głównym ekranie aplikacji i dostęp do sieci zostanie utracony – występuje stosowny komunikat i powrót do ekranu głównego WebConnection activity. Gdy dostęp zostanie przywrócony następuje przeskok – omijane jest InfoActivity, a użytkownik ponownie może śledzić wielkości parametrów fizjologicznych danego pacjenta.

5. Kompilacja

Embedded

Aby skompilować pliki sourcowe najlepiej posłużyć się programem Keil uVision w którym kod został przygotowany. Otworzyć należy projekt nrf52-ble-tutorial-characteristic.uvprojx (nazwa szablonu nie została zmieniona aby uniknąć ewentualnych problemów z deppendencies).
Skompilowana wersja tego projektu znajduje się już w folderze i można z niej od razu korzystać.

LabVIEW

Kompilacja aplikacji napisanej w LabVIEW wymaga środowiska deweloperskiego tego programu w wersji 2015 lub późniejszej. W projekcie NORDIC Patient Monitoring Device.lvproj przygotowana jest już specyfikacja Buildu. Wystarczy po otwarciu kliknąć na nią prawym przyciskiem myszy i wybrać 'build'. Co ważne zbudowany exec będzie wymgał LabVIEW RunTime Engine 2015 lub późniejszego do działania! Są to pliki które można ściągnąć ze stron National Instruments w pojedynczej paczce, bez rejstracji i opłat. Przykładowy engine dla Windowsa 64-bitowego: http://www.ni.com/download/labview-run-time-engine-2016/6067/en/
Skompilowana wersja tego projektu znajduje się już w folderze i można z niej od razu korzystać.

Biblioteka do obsługi bazy

Kompilacja aplikacji napisanej w języku C# i środowisku Visual Studio 15 wymaga tego środowiska lub wersji późniejszej. Aby dokonać kompilacji potrzebny jest zainstalowany na komputerze .NET Framework w wersji 4.5.2. Biblioteka została stworzona jako template **Class library** w Visual Studio, co oznacza, że kompilowana jest od razu do pliku z rozszerzeniem *.dll.

Skompilowana wersja biblioteki, gotowa do użycia znajduje się pod ścieżką :
NORDIC Database\NordicDatabaseDLL\NordicDatabaseDLL\bin\Debug\NordicDatabaseDLL.dll

Aplikacja mobilna (Xamarin)

Kompilacja aplikacji mobilnej napisanej w języku C# i środowisku Visual Studio przy użyciu Xamarin framework wymaga zainstalowanego Visual Studio w wersji 15 lub wyższej oraz Xamarina. Poza tym, niezbędne jest aby zainstalowane zostało odpowiednie SDK Androida – omawiana aplikacja będzie działać od SDK 16 (API 16 – Android Jelly Bean), aczkolwiek kompilowana jest przy użyciu SDK 23 (API 23 – Android Marshmallow).

Aby uruchomić aplikację na tym etapie na telefonie należy posiadać urządzenie z systemem operacyjnym powyżej wersji Android 4.1, uruchomić opcje debugowania przez USB na komórce oraz uruchomić aplikację w trybie Debug w Visual Studio. Nie powstał gotowy build do instalacji aplikacji bezpośrednio na telefonie.

6. Podsumowanie

Dokumentacja zawiera wszelkie idee, które zostały zawarte w projekcie Patient Moitoring Device. Oprogramowanie oraz wszelkie inne dokumenty znajudją się na naszym repozytorium na githubie. Projekt został zrealizowany na licencji OpenSource.

https://github.com/ace333/Nordic_PMD/tree/PMD