Autorem projektu jest Stuart Johnson, dyrektor firmy Logic Ethos Ltd., z Southampton z Wielkiej Brytanii. Całość kodu projektu można znaleźć pod adresem **https://goo.gl/riA2DH**, a jego krótki, angielski opis na **https://goo.gl/hjj0CQ**. Zachęcamy przede wszystkim do obejrzenia filmiku prezentującego działanie aparatu **https://goo.gl/wMFtE4**.

PiTelephone – retro telefon z Raspberry Pi

W „Elektronice Praktycznej" opisywaliśmy już projekt telefonu, w którym zastosowano Raspberry Pi, ale prezentowany poniżej PiTelephone to zupełnie inna konstrukcja. Widać to już z zewnątrz, bo aparat telefoniczny jest stacjonarny i tarczowy i wygląda, jakby wyprodukowano go w latach 70. XX wieku. W środku znajduje się natomiast Raspberry Pi i szereg dodatkowych komponentów, które zmieniają wybieranie impulsowe na tonowe oraz sterują pracą dzwonka. Ponadto, projekt został napisany w języku C#, a nie Pythonie.

Tworząc telefon stacjonarny w stylu retro, oparty na Raspberry Pi, trzeba pokonać wiele problemów, a właściwie zmierzyć się z licznymi, bardzo różnorodnymi zadaniami. Wynika to przede wszystkim z różnic, jakie zaszły w telekomunikacji na przestrzeni dziesięcioleci. Niemniej, uzyskany efekt może być bardzo atrakcyjny. Gotowy telefon w stylu retro, podłączony do Internetu, robi nie lada wrażenie, a korzystanie z niego może naprawdę sprawiać przyjemność.

Jeden telefon – wiele aspektów

Po pierwsze, problematyczne jest wybieranie impulsowe, od którego centrale telefoniczne już dawno odeszły i zastąpiły je wybieraniem tonowym. Ta druga metoda jest po prostu szybsza, gdyż zamiast zliczać następujące po sobie impulsy, wystarczy rozpoznać dwie składowe częstotliwości, użyte do wygenerowania wybranego tonu. Trwa to ułamek czasu potrzebnego na rozpoznanie zera wybieranego impulsowo. Dlatego autor musiał wykonać układ, który będzie samodzielnie rozpoznawał wybrany na tarczy numer.

Po drugie, problematyczne jest zasilanie. Klasyczne telefony tarczowe korzystają z napięcia w sieci telekomunikacyjnej i nie potrzebują dodatkowego podłączenia do prądu. Tu jednak trzeba zasilić cały komputer, wymagający 5 V, a dodatkowo napędzić dzwonek, który jest niemałym elementem elektromechanicznym i 5 V do tego nie wystarczy.

Po trzecie, stosowane w starych telefonach głośniki i mikrofony mają się nijak do obecnych standardów odnośnie do interfejsów audio w komputerach. Dlatego konieczne staje się

Listing 1. Fragmenty pliku program.cs, zawiadującego sprzętem wewnątrz telefonu

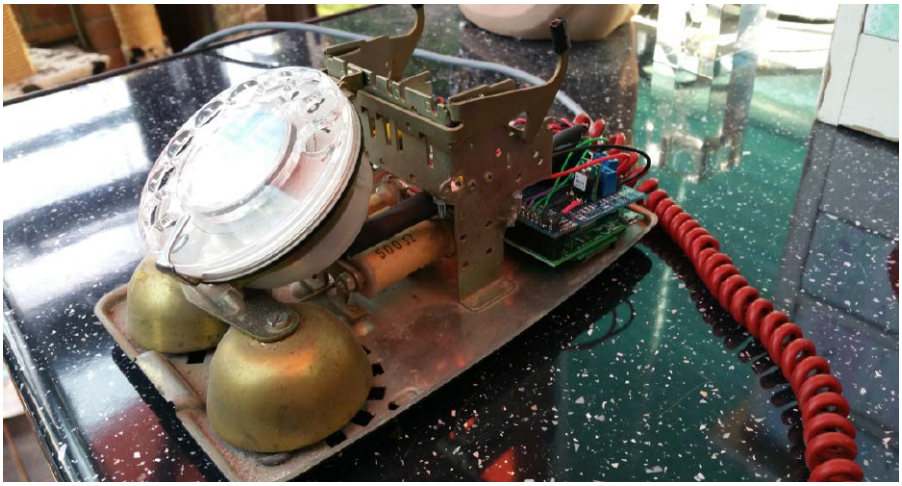
```
//załadowanie bibliotek, w tym Mono
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using Mono.Posix;
using Mono.Unix;

namespace PiTelephone
{
    class Program
    {
        static void Main (string[] args)
        {
//ustawianie wejść i wyjść w oparciu o Raspberry# IO
            var RingerPower = new Raspberry.IO.GeneralPurpose.OutputPinConfiguration (Raspberry.IO.GeneralPurpose.ProcessorPin.Pin17);
            var RingerOscillator = new Raspberry.IO.GeneralPurpose.OutputPinConfiguration (Raspberry.IO.GeneralPurpose.ProcessorPin.Pin18);
            var HookSwitch = new Raspberry.IO.GeneralPurpose.InputPinConfiguration (Raspberry.IO.GeneralPurpose.ProcessorPin.Pin22)
            {
                Reversed = true
            };
            var DialPulseSwitch = new Raspberry.IO.GeneralPurpose.InputPinConfiguration (Raspberry.IO.GeneralPurpose.ProcessorPin.Pin27);

//Przygotowanie klas do obsługi GPIO dla dzwonka i tarczy
            using (var ringer = new clsRinger (RingerPower, RingerOscillator))
            {
                using (var dialListener = new clsDialHookListener(HookSwitch,DialPulseSwitch))
                {
//Wprowadzenie dodatkowej funkcji testu dzwonka w dwóch stylach, za pomocą cyfr 0 lub 9
                    dialListener.NumberDialed += (uint NumberDialed) =>
                    {
                        Console.WriteLine(„Number Dialed:{0}”,NumberDialed);
                        if (NumberDialed == 0)
                        {
                            ringer.SetRingPattern (clsRinger.ringPattern_UK);
                            ringer.StartRing ();
                        } else if (NumberDialed == 9)
                        {
                            ringer.SetRingPattern (clsRinger.ringPattern_USA);
                            ringer.StartRing ();
                        }
                    };
//Dodatkowa funkcja wyłączenia dzwonka
                    dialListener.HookSwitchChange += (bool OnHook, uint Pulse) =>
                    {
                        if (!OnHook)
                        ringer.StopRing ();
                    };
                    UnixSignal[] signals = new UnixSignal []
                    {
                        new UnixSignal (Mono.Unix.Native.Signum.SIGINT),
                        new UnixSignal (Mono.Unix.Native.Signum.SIGUSR1),
                    };
//Główna pętla programu
                    while (true)
                    {
                        int index = UnixSignal.WaitAny (signals, -1);
                        Mono.Unix.Native.Signum signal = signals [index].Signum;
                        Console.Write(„SIGNAL:{0}”,signal.ToString());
                        break;
                    };
                }
            }
            Console.WriteLine („**end**”);
        }
    }
}
```

zastąpienie dawnych podzespołów audio nowymi.

Po czwarte, korzystając z Raspberry Pi, nie otrzymujemy złącza telefonicznego, które można byłoby wykorzystać do podłączenia się do takiej sieci. W tym przypadku autor poszedł nieco na łatwiznę i zrezygnował z podłączenia do kabla telefonicznego, zastępując go siecią ethernetową i technologią VOIP (czy np. choćby Skype). W efekcie niniejszy projekt jest także świetnym przykładem programowania elektroniki w praktyce, tj. z wykorzystaniem komercyjnie dostępnych rozwiązań i usług. Wiedza zaczerpnięta z niego może posłużyć do przygotowania innego rozwiązania, bazującego na minikomputerze, komercyjnej technologii VOIP i języku C.



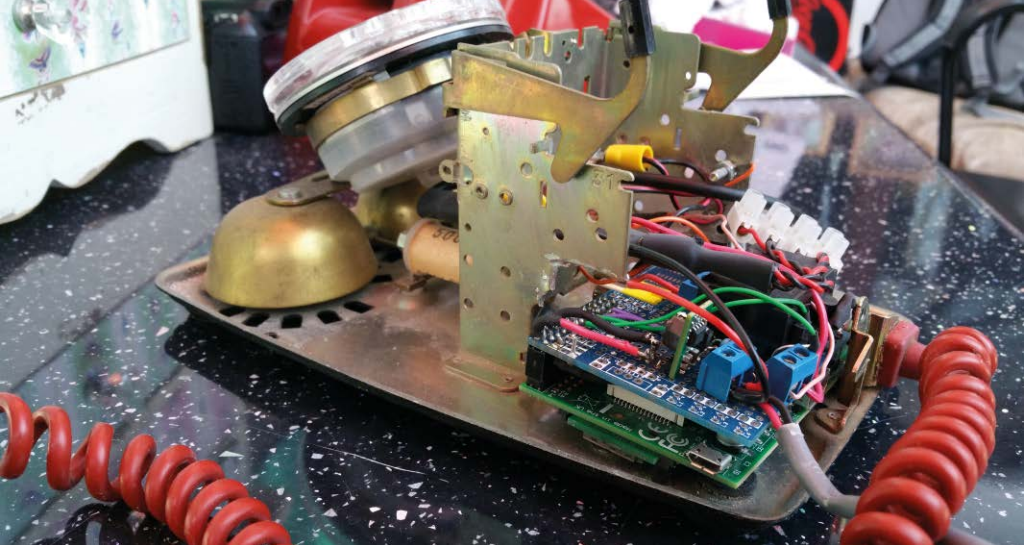
Fotografia 1. Tarcza i dzwonek

Podstawa, czyli stary telefon

Autor postanowił, że jego urządzenie będzie miało wygląd klasycznego telefonu tarczowego z Wielkiej Brytanii, z lat 70. ubiegłego wieku. Wybrał model GPO 746, zakupiony na eBayu. W Polsce z łatwością można zdobyć podobne aparaty na Allegro czy innym serwisie z ogłoszeniami, a pewnie część czytelników ma jeszcze takie telefony w szafie lub piwnicy. Koszt zakupu to obecnie od kilkudziesięciu złotych za modele z lat 80. i 90., przez ok. 100 zł za aparaty ebonitowe z lat 50.–70., aż do kilkuset za prawdziwe, stylowe zabytki. Większość z nich będzie działała podobnie, choć pewne różnice mogą obejmować cewkę dzwonka, która w praktyce może pracować z innymi napięciami.

Zasilanie

Gdyby chcieć zastosować się do napięć znamionowych komponentów telefonu, trzeba by było zapewnić dostępność prądu przemiennego o napięciu 50 V, jednocześnie zasilając Raspberry Pi napięciem stałym 5 V. W praktyce jednak w wielu telefonach 50 V to znacznie więcej niż



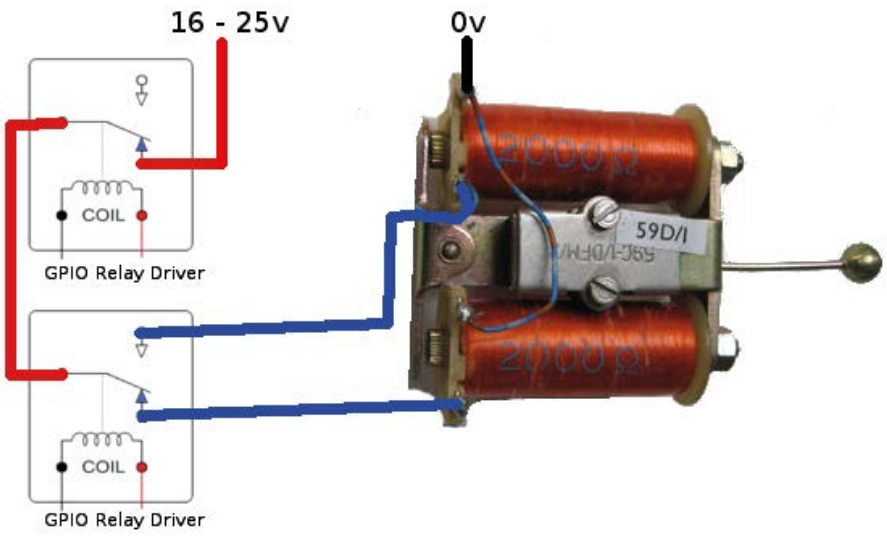
Fotografia 2. Upakowane komponenty elektroniczne

faktycznie potrzeba do poruszenia cewkami dzwonka. W przypadku modelu GPO 746 okazało się, że wystarczy 16 V, by gong przesunął się, uderzając w kopułę dzwonka. Ponadto nie ma potrzeby, by podawane napięcie zmieniało się w jakiś dokładnie określony sposób. Wystarczy, że odpowiednio szybko cewki będą naprzemiennie polaryzowane, co spowoduje uderzenia gongu i da efekt dzwonka.

Napięcie stałe 16 V lub nieco wyższe można łatwo uzyskać z większości zasilaczy do laptopów. Tak też zrobił autor projektu, stosujc swój stary, 19-woltowy zasilacz. Mając już względnie stabilne 19 V, za pomocą małej, scalonej przetwornicy można uzyskać napięcie 5 V, potrzebne do Raspberry Pi. Autor użył modelu Murata OKI-78SR-5/1.5-W36-C, czyli przetwornicy impulsowej DC-DC. Jej zaletą są niewielkie wymiary, brak konieczności stosowania radiatora czy dodatkowych komponentów zewnętrznych, zabezpieczenie przed zwarciem oraz szeroki zakres napięć wejściowych – od 7 VDC do 36 VDC (a więc będzie pasować także do innych zasilaczy laptopowych). Wydajność prądowa tego układu to 1,5 A, co w zupełności wystarcza dla projektowanej aplikacji. Układ ma kształt i wyprowadzenia zaplanowane tak, by mógł z łatwością zastąpić stabilizatory liniowe serii 78xx w obudowach TO-220. W przypadku Raspberry Pi nie trzeba się też martwić o podłączanie zasilania do gniazda micro USB – wystarczy podłączyć je do odpowiednich wyprowadzeń GPIO i system będzie działać.

Dzwonek

Typowo dzwonek w telefonie bije z częstotliwością 25 Hz, tj. 50 razy na sekundę słychać uderzenie – raz w jedną kopułę, a raz w drugą. Dokładna wartość nie ma jednak większego znaczenia – ważne, by zastosowana częstotliwość pozwalała uzyskać właściwe wrażenie dzwonka.



Rysunek 3. Schemat połączeń przekaźników z cewkami dzwonka

Dostarczone napięcie stałe 19 V z zasilacza trzeba więc podawać na dwie cewki naprzemiennie, z częstością ok. 25 Hz. W tym celu wystarczy zastosować dwa przekaźniki. Teoretycznie wystarczyłby jeden, przełączający napięcie z zasilacza pomiędzy cewkami, ale rzecz w tym, by prąd nie płynął przez żadną cewkę w czasie, gdy telefon nie dzwoni. Przekaźniki warto podłączyć tak, jak przedstawiono na rysunku 3., czyli szeregowo. Pierwszy włącza mechanizm dzwonka, czyli dopływ prądu do drugiego przekaźnika, a drugi przełącza prąd pomiędzy cewkami ok. 50 razy na sekundę.

Wybieranie tarczą

Na tym etapie mogą pojawić się pewne trudności, a ponadto można się spodziewać większych rozbieżności pomiędzy poszczególnymi aparatami telefonicznymi. By poznać cyfrę wybieraną przez użytkownika, należy zliczyć liczbę następujących po sobie impulsów elektrycznych, pochodzących z tarczy. Użytkownik wybiera cyfrę, przekręcając kółko tarczy do odpowiedniej pozycji, po czym puszcza je i w określonym, stałym tempie, wraca ono do pozycji wyjściowej, emitując przy tym impulsy w regularnych odstępach czasowych. Odstępy te będą różne dla poszczególnych aparatów, a dokładniej – będą zależeć od szybkości powrotu tarczy do pozycji wyjściowej. W przypadku aparatu GPO 746 impuls pojawia się co ok. 91,5 milisekundy. Ważne jest też, by rozpoznawać początek i koniec impulsów dla wprowadzanej cyfry. Telefony tarczowe były konstruowane tak, by fizycznie nie dało się zbyt szybko wybierać jednej cyfry po drugiej – początkowy ruch tarczy nie powoduje wysyłania impulsów, toteż pomiędzy każdymi wybieranymi cyframi należy spodziewać się przerwy. Autor w swoim projekcie na podstawie doświadczeń ustalił, że przyjmie, że musi ona trwać nie mniej niż ok. 400 ms. Wystarczy więc podłączyć wyprowadzenia tarczy do GPIO Raspberry Pi i nawet bez dodatkowych elementów elektronicznych czekać na pojawienie się impulsu, po czym zliczać je, aż do momentu, gdy przestaną się pojawiać przez przynajmniej 400 ms. Raspberry Pi, dzięki dosyć szybkiemu procesorowi, jest w stanie liczyć czas pomiędzy impulsami nawet z dokładnością do ok. 1 ms, co może mieć znaczenie w przypadku pojawiania się ewentualnych, niepożądanych drań i fałszywych impulsów. Należy to zbadać doświadczalnie i w razie czego ignorować te, które występują znacznie częściej niż regularne (a więc dużo częściej niż co ok. 90 ms).

Listing 2. Fragmenty pliku odpowiadającego za obsługę dzwonka

```
namespace PiTelephone
{
    public class clsRinger : IDisposable
    {
//Definicje długości sygnałów dźwiękowych
        public static readonly float[] ringPattern_UK = { 0.4f, 0.2f, 0.4f, 2f };     public
static readonly float[] ringPattern_USA = { 2f, 4f };
        public int ringHz = 25;
//////////////////////////////////////////////////////
//funkcja uruchamiająca dzwonek
        public void StartRing ()
        {
            if ((RingerThread.ThreadState & (ThreadState.Unstarted | ThreadState.WaitSleep-
Join | ThreadState.Stopped)) != ThreadState.Running)
            {
                try
                {
                    RingerThread.Start();
                }
                catch (Exception ex)
                {
                    Console.WriteLine („Error starting Ringer Thread: „ + ex.Message);
                }
            }
        }
//funkcja zatrzymująca dzwonek poprzez wyłączenie wątku dzwoniącego
        public void StopRing()
        {
            RingerThread.Abort();
        }
//funkcja dzwoniąca
            private void Ring()
        {
            Console.WriteLine(„Ringer Start”);
//włączenie zasilania do cewek dzwonka
            if (!GPIO[RingerPowerPin.Pin])  GPIO.Toggle(RingerPowerPin.Pin);
            try
            {
                while (true)
               {
                    int ms = 1000 / ringHz;
                    for (int f = 0; f < ringPattern.Length; f++)
                    {
                         for (int i = 0; i < ringHz * ringPattern[f]; i++)
                         {
//zmiana cewki dzwonka, przez którą płynie prąd
                             GPIO.Toggle(RingerOscillatorPin);
                                  Thread.Sleep(ms);
                         }
                         Thread.Sleep((int)(ringPattern[++f] * 1000));
                    }
                }
            }
            catch (System.Threading.ThreadAbortException)
            {
                Console.WriteLine(„Ringer Stop”);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
//resetowanie wyjść
            finally
            {
                if (GPIO[RingerOscillatorPin.Pin])  GPIO.Toggle(RingerOscillatorPin.Pin);
                if (GPIO[RingerPowerPin.Pin])  GPIO.Toggle(RingerPowerPin.Pin);
            }
        }
```

Audio

Problem starego mikrofonu i głośnika autor rozwiązał w dosyć prosty sposób. Po prostu kupił tani zestaw słuchawkowy i wyjął z niego komponenty, po czym wmontował je w słuchawkę, zastępując stare. Większość klasycznych aparatów telefonicznych została zbudowana tak, że wystarczy odkręcić nakładki na obudowie słuchawki i można łatwo dostać się do komponentów audio.

Dopasowanie obudowy i pozostałe komponenty

Teoretycznie wszystkie potrzebne komponenty zostały już opisane, ale może się okazać, że w przypadku konkretnego modelu telefonu warto zastosować dodatkowe przejściówki lub płytki. Autor, by zmieścić wszystko wewnątrz obudowy i wygodnie podłączyć podzespoły, skorzystał z dodatkowego PCB do wyprowadzenia sygnałów GPIO oraz z przejściówki z karty micro SD na SD. Potrzeba dodatkowych komponentów będzie też zależeć od użytej wersji

Raspberry Pi. Przydatne okażą się też zapewne śrubki, którymi elektronikę będzie można przytwierdzić wewnątrz obudowy. Ewentualnie można użyć także kleju termicznego, który zredukuje wpływ wibracji dzwonka na resztę komponentów oraz odizoluje je elektrycznie w razie, gdyby zaczęły drgać i stykać się.

Niestety, umieszczenie całości w starym telefonie będzie prawdopodobnie wymagało wycięcia elementów ze środka czy nawet z zewnętrznej części obudowy. Być może nawet będą to elementy konstrukcji metalowej, dlatego warto mieć pod ręką miniwiertarkę, która posłuży także jako szlifierka kątowa. W przypadku zabytkowych aparatów warto zminimalizować liczbę nacięć, gdyż zmniejszą one wartość telefonu. Oczywiście – bardzo wiele będzie zależeć od wielkości użytego aparatu.

Schemat połączeń wewnątrz telefonu wynika z opisanych już elementów. Warto tylko dodać, że oprócz rozpoznawania impulsów, które polegają na zwieraniu jednego z wejść GPIO do masy, należy też rozpoznawać stan słuchawki – tj. tego, czy leży ona na „widełkach", czy też jest podniesiona. To również można zrealizować za pomocą zwierania kolejnej linii GPIO do masy.

Oprogramowanie

Choć wydawać się może, że o atrakcyjności projektu decydują elementy mechaniczne, obudowa i sposób pokonania opisanych problemów, tak naprawdę klucz do sukcesu leży raczej w oprogramowaniu. To dzięki niemu możliwe jest np. zliczanie impulsów bez użycia dodatkowych komponentów (a więc tanio) czy podłączenie się do niedrogich serwisów oferujących połączenia VOIP.

Unikalne wśród opisywanych dotąd projektów w EP jest zastosowanie języka C#, który przecież kojarzy się bardzo silnie ze środowiskiem Microsoft Windows, a działającego raczej na komputerach z procesorami x86, a nie ARM. Autor posłużył się biblioteką Mono – sponsorowaną przez Microsoft platformą programową, która

Fotografia 4. Gotowy, niepozorny aparat telefoniczny

umożliwia tworzenie aplikacji .NET-owych na wiele różnych systemów operacyjnych. Mono jest dostępne na systemy Windows (zarówno 32-, jak i 64-bitowe), Linux (oficjalnie dla Debiana, Ubuntu i pochodnych, CentOS-a, Fedory i pochodnych oraz openSUSE i SLES, a za pośrednictwem niezależnie tworzonych paczek także na Arch Linux, CentOS EPEL i Gentoo) oraz na Mac OS X. Raspbian wykorzystywany powszechnie w Raspberry Pi jest systemem pochodnym od Debiana, więc na nim Mono też będzie działać. Aktualnie dostępna wersja Mono to 4.6.

Naturalnie, samo Mono nie będzie miało zaimplementowanego specjalnego wsparcia dla każdego komputerka jednopłytkowego, gdyż jest to biblioteka programowa, przygotowywana ogólnie pod kątem systemów operacyjnych, a nie platform sprzętowych. Dlatego konieczne jest użycie odpowiedniej biblioteki, która pozwoli na wygodną komunikację z GPIO Raspberry PI. Dotąd w opisywanych projektach wykorzystywane były głównie biblioteki dla Pythona, a tymczasem tu potrzebne są pliki w C# dla .NET. Na szczęście społeczność twórców Raspberry Pi przygotowała już odpowiednie narzędzia pod nazwą Raspberry# IO. Aktualna wersja tej biblioteki jest określana mianem wczesnej i nie wszystkie jej funkcje zostały w pełni przetestowane, ale umożliwia podawanie i wczytywanie stanów niskich i wysokich poprzez GPIO, a także komunikację przez SPI i I²C. Jej twórcy udostępnili także przykłady implementacji komunikacji przez I²C, SPI z różnymi podzespołami oraz wykorzystania alfanumerycznego wyświetlacza LCD z kontrolerem HD44780. Pewnym ograniczeniem w niektórych projektach może być natomiast fakt, że programy uruchamiane z użyciem Raspberry# IO muszą mieć uprawnienia administratora.

Zaletą użycia Raspberry# IO jest wbudowana w tę bibliotekę obsługa zdarzeń odnośnie do GPIO. Oznacza to, że nie ma potrzeby ciągłego, „ręcznego" monitorowania stanu wejść. Można z łatwością określić zdarzenia – jak np. pojawienie się zwarcia na określonej linii GPIO, odpowiadającej za odbiór impulsów z tarczy i na tej podstawie wywołać zliczanie okresów pomiędzy impulsami, a więc określanie wybranej cyfry. Co więcej, mając wszystko pod kontrolą programową, można zaimplementować w telefonie dodatkowe funkcje, których oryginalnie nie miał. Przykładowo – szybkie naciśnięcie widełek może powodować wstrzymanie rozmowy, zamiast jej rozłączenia. Wystarczy tylko zliczać czas zwarcia na linii GPIO podłączonej do widełek.

Autor niniejszego projektu napisał całość w Visual Studio 2010, niecałe 4 lata temu, a więc gdy platforma Mono była dostępna w wersji 3.0, a Raspberry# IO dopiero powstawało. Mimo to udało się stworzyć w pełni funkcjonalny telefon, ale trzeba mieć na uwadze, że gdy odtwarza się projekt z nowymi bibliotekami, może wystąpić konieczność wprowadzenia pewnych zmian.

Listing 3. Funkcja odpowiadająca za zliczanie impulsów z tarczy. Została umieszczona w pliku wraz z kodem obsługującym widełki, ale tę część pominięto

```
private void ListenDial ()
{
    int MillisecondsToDialedNumber = (int)DialPulseMaxMs + 100;
    Console.WriteLine („ListenDial Thread Started");
    while (KeepRunning)
    {
        DialWaitEvent.WaitOne(); //Czekaj na pierwszy impuls
        DialWaitEvent.Reset();
        LastDialPulse = DateTime.UtcNow;
        Console.Write(„|+");
        while (KeepRunning)  //Czekanie na kolejną cyfrę
        {
            DialWaitEvent.WaitOne(MillisecondsToDialedNumber); //Czekanie na kolejny im-
puls lub na upływ czasu oczekiwania
            var milliseconds = (DateTime.UtcNow-LastDialPulse).TotalMilliseconds;
            Console.Write(„+");
            Console.WriteLine („+ „+milliseconds.ToString());
            LastDialPulse = DateTime.UtcNow;
            if (milliseconds>DialPulseMaxMs)  //Zakończenie zliczania impulsów
            {
                if (Debug) Console.WriteLine („Dial: {0} {1}",DialPulseCount,GPIO[Dial-
IO]?"On":"Off");
                if (NumberDialed != null) NumberDialed(DialPulseCount > 9 ? 0 : DialPulse-
Count);
                DialPulseCount = 0;
                DialWaitEvent.Reset();
                break; //powrót do głównej pętli
            }
            DialWaitEvent.Reset();
        }
    }
    Console.WriteLine („ListenDial Thread STOPPED");
}
```

Na główną część programu składają się trzy pliki z kodem, uzupełnione przez szereg plików konfiguracyjnych, głównie zawierających ustawienia wymagane do obsługi środowiska .NET. Kod głównego pliku program.cs został pokazany wraz z komentarzem na listingu 1. Uzupełniają go kody dzwonka i licznika impulsów tarczy, umieszczone w oddzielnych plikach, których fragmenty zostały pokazane na **listingach 2** i **3**. Obejmują one całą obsługę zdarzeń związanych ze sprzętem telefonicznym, ale nie zawierają kodu potrzebnego do wykorzystania zewnętrznych usług VOIP.

Obsługa VOIP

Pozostaje realizacja faktycznej komunikacji telefonicznej za pośrednictwem Internetu. Autor opracował swój projekt dosyć dawno i zastosowane przez niego rozwiązania nie są już całkiem aktualne. Niemniej próbował zaimplementować obsługę trzech alternatywnych względem siebie technologii: Skype, PJSIP i FreeSWITCH. Spośród nich, obecnie wydaje się, że najlepiej udokumentowana i utrzymywana jest biblioteka dla Skype. Nie będziemy jej opisywać, gdyż jest dosyć duża, ale pozwala na odbieranie i nawiązywanie połączeń po cenach Skype. Opis API Skype dla rozmów głosowych znaleźć można pod adresem ***https://goo.gl/LkG9qe***.

Podsumowanie i ocena projektu

Omawiany projekt można zaliczyć do bardzo udanych. Jest na tyle atrakcyjny i wzbudza zainteresowanie, że można byłoby sobie wyobrazić oferowanie tego typu adaptacji starych telefonów na VoIP lub po prostu sprzedawanie takich zmodernizowanych aparatów (pomijamy kwestie potrzebnych certyfikatów). Można też sobie wyobrazić przygotowanie wersji, w której za pomocą odpowiedniej przejściówki sygnały tonowe przekazywane są do rzeczywistej linii telefonicznej.

Użycie Raspberry Pi 3 pozwoliłoby natomiast na zrealizowanie wersji bezprzewodowo podłączonej do Internetu (przez Wi-Fi), a ewentualne wykorzystanie modemu sieci GSM lub 3G sprawiłoby, że aparat byłby faktycznie telefonem komórkowym, tyle że wymagającym stałego podłączenia do zasilania.

MARCIN KARBOWNICZEK, EP