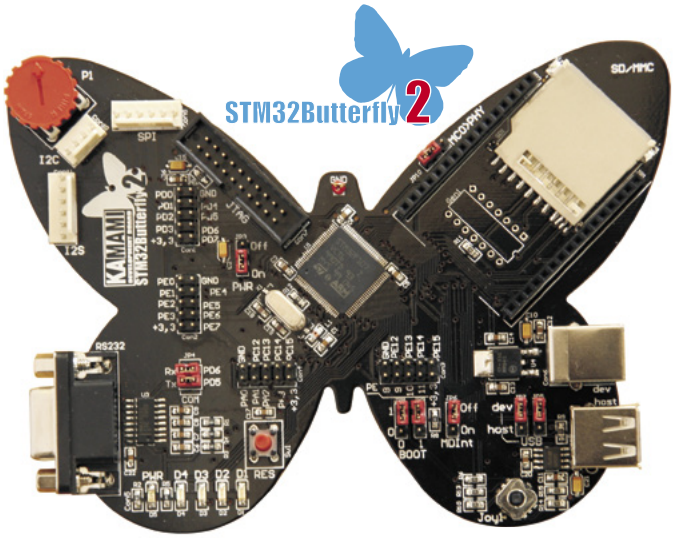
Cyfrowy analizator 32-kanałowy (2)

Implementacja aplikacji

Płytka ewaluacyjna zakupiona w celu zapoznania się z nowym układem scalonym po zakończeniu nauki nie musi być bezużyteczna. Można na jej bazie zbudować sterownik mikroprocesorowy lub użyteczny przyrząd laboratoryjny. W artykule opisano sposób wykonania analizatora stanów logicznych z wyświetlaniem zarejestrowanych poziomów sygnałów na ekranie PC, z którym komunikacja odbywa się przez port USB. Zaprezentowane rozwiązanie może posłużyć jako baza do budowy wielu własnych projektów.



Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 19623, pass: 6c5r20n3
- pierwsza część artykułu

W głównym oknie aplikacji dla komputera PC utworzono dwie zakładki. W pierwszej umieszczono składniki do konfigurowania analizatora, natomiast druga służy do obserwacji wyników komunikacji z analizatorem oraz analizy otrzymanych danych.

Konfigurowanie analizatora

Pierwszym krokiem, jaki należy wykonać przy używaniu aplikacji, jest skonfigurowanie *trigger word*, który w aplikacji jest reprezentowany jako 32 pola typu *combo box*. Do wyboru jest jedna z trzech możliwości:

0 – oczekiwane wystąpienie poziomu niskiego,

1 – oczekiwane wystąpienie poziomu wysokiego,

X – poziom logiczny nie ma znaczenia.

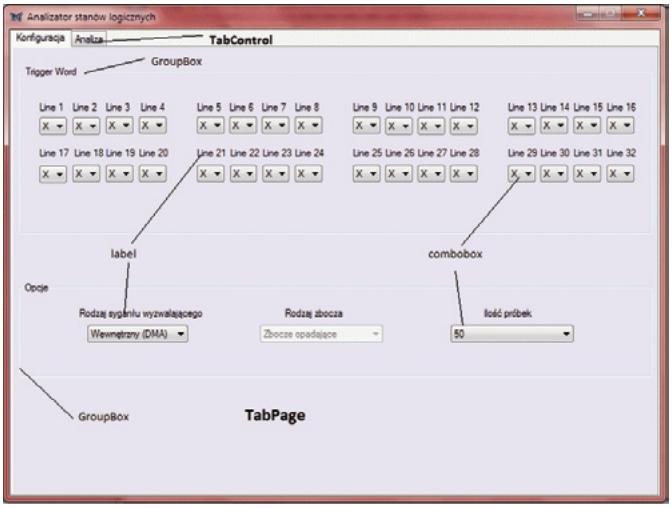
Odpowiedni fragment programu zamieszczono na **listingu 14**. Standardowo (przy uruchamianiu aplikacji) wszystkim polom *combo* jest nadawana wartość ‚X' (list. 1). Każdemu z pól *combo* przypisano również etykietę (*label*), która jest umieszczona bezpośrednio nad nim i zawiera opis, której linii I/O dotyczy dany element. Wygląd okna umożliwiającego konfigurowanie sposobu wyzwalania pokazano na **rysunku 1**.

Przed wysłaniem *trigger word* do urządzenia należy na podstawie wybranych oczekiwanych poziomów wyliczyć, jakie poziomy logiczne powinny pojawić się na liniach I/O odpowiednich rejestrów urządzenia (**listing 15**). Podobne funkcje pomocnicze utworzono dla rejestrów C, D, E oraz odpowiednio młodszej i starszej części tych rejestrów.

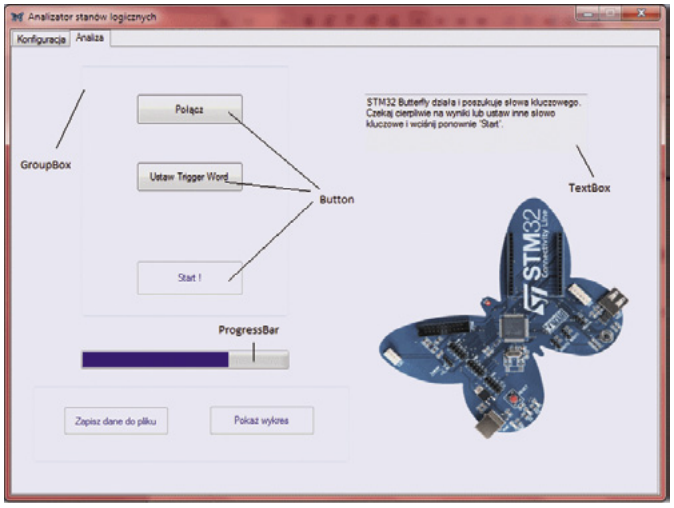
Listing 14. Inicjalizacja pola *combo* służącego do wyboru trybu pracy wyzwalania, nadanie mu wartości domyślnej
```
\\StmAnalyzer.cs - InitializeComponent
this.comboBox2.Items.AddRange(new object[] {
"0",
"1",
"X"});
...
\\StmAnalyzer.cs - public StmAnalyzer()
  for (int i = 0; i < comboBoxes.Length; i++)
  {
    comboBoxes[i].SelectedItem = "X";
  }
```

Listing 15. Wyliczenie warunków wyzwolenia analizatora
```
\\StmAnalyzer.cs

/* Funkcja ta na podstawie wartości ustawionych w combo boxach wyliczy jak przy takich danych powinna wyglądać młodsza
część rejestru wejściowego E, na którym monitorujemy linie od 24 do 32. */
private byte calculateLowE()
{
  byte lowE = 0;
/* Wartość mnożnika, zwiększana dwukrotnie po przejściu do następnej linii. */
  int mul = 1;
  int val;
  for (int i = 24; i < 32; i++)
  {
/* Jeśli podano 1 to szukamy jedynki, w przeciwnym wypadku do rejestru wpisujemy 0, co pozwala nam poprawnie porównywać
zwróconą wartość z wartością prawdziwego rejestru gdy na danej linii jest 0 lub ona nas nie interesuje. */
    val = comboBoxes[i].SelectedItem.Equals("1") == true ? 1 : 0;
    lowE += (byte)(val * mul);
    mul *= 2; (2)
  }
  return lowE;
}
```



Rysunek 1. Okno służące do wprowadzania nastaw



Rysunek 2. Główne okno interfejsu użytkownika

Do urządzenia jest wysyłana również maska, która jest nakładana na wartość prawdziwego rejestru, po czym porównywana z wprowadzoną przez użytkownika wartością słowa kluczowego. Takie postępowanie umożliwia nierejestrowanie wszystkich linii I/O (liniom nierejestrowanym jest nadawana wartość „X"). Przykładowo, dla młodszej części rejestru E maskę obliczamy w sposób pokazany na **listingu 16**.

Analizator może rejestrować poziomy występujące na doprowadzeniach I/O za pomocą sygnału wewnętrznego (taktowanie z DMA) lub zewnętrznego. W przypadku taktowania zewnętrznego należy również ustalić rodzaj zbocza.

Podobnie jak w wypadku *trigger word*, do wprowadzania opcji rejestracji użyto pól *combo*. Przy czym, jeśli mamy wybrany sygnał wewnętrzny, to pole *combo* do ustawiania rodzaju zbocza jest nieaktywne (**listing 17**). Liczbę próbek, które układ będzie przesyłał do komputera, ustalono na: 10, 50, 100, 200, 500 lub 1000. Dla uproszczenia, wyboru liczby próbek również dokonuje się za pomocą pola *combo*. Przy czym te wartości to liczba próbek pobranych przed wystąpieniem słowa kluczowego i po jego wystąpieniu. Czyli np. przy liczbie próbek wynoszącej 1000 układ odeśle 2001 próbek, po 10 bajtów każda, czyli w sumie około 20 kB.

Graficzny interfejs użytkownika

Kolejna zakładka o nazwie „Analiza" (**rysunek 2**) pełni funkcję interfejsu do komunikowania się komputera PC z użytkownikiem, po wcześniejszym wprowadzeniu nastaw. W celu ułatwienia pracy utwo-

Listing 16. Wyliczenie maski dla młodszej części rejestru E

```
\\StmAnalyzer.cs
private byte calculateLowEMask()
{
  byte lowE = 0;
  int mul = 1;
  int val;
  for (int i = 24; i < 32; i++)
  {
/* Jeśli mamy ‚X' to oznacza, że dana linia nie interesuje użytkownika więc możemy na odpowiednim bicie maski ustawić
0. */
    val = comboBoxes[i].SelectedItem.Equals("X") == true ? 0 : 1;
    lowE += (byte)(val * mul);
    mul *= 2;
  }
  return lowE;
}
```

Listing 17. Wybór źródła sygnału taktującego

```
//StmAnalyzer.cs
private void sygnalBox_SelectedIndexChanged(object sender, EventArgs e)
{
  if (((ComboBox)sender).SelectedItem.Equals("Zewnętrzny"))
  zboczeBox.Enabled = true; else zboczeBox.Enabled = false;
}
```

Listing 18. Obsługa zdarzenia po kliknięciu na przycisk „Połącz"

```
\\StmAnalyzer.cs – kod obsługi zdarzenia polegającego na kliknięciu przycisku „Połącz".
private void connectButton_Click(object sender, EventArgs e)
{
/* Proces enumeracji wybiera listę zainstalowanych urządzeń o podanym numerze VID i PID. Firma STElectronics otrzymała
do własnych potrzeb VendorId = 0x0483 oraz ProducttId = 0x5750. */
  HidDevice[] hids = HidDevices.Enumerate(0x0483, 0x5750);
  if (hids.Length > 0)
  {
/* Z listy znalezionych urządzeń wybierane jest pierwsze, które będzie utożsamione z używanym w programie sprzętem. */
    stm = hids[0]; (2)
/* Po otwarciu urządzenia można już wymieniać z nim dane. */
    stm.Open();     (3)
    Thread.Sleep(50);
    sendCaptureButton.Enabled = true;
    textBox.Text ="Sprawdź czy Trigger Word został poprawnie wprowadzony.\n Następnie wyślij słowo do urządzenia.";
  } else {
/* W wypadku braku połączenie z płytką użytkownik powiadomiony zostaje o tym poinformowany stosownym komunikatem. */
    MessageBox.Show(„Nie znaleziono urządzenia.\nSprawdź czy sprzęt jest poprawnie podłączony, następnie spróbuj
jeszcze raz.", „Błąd");  (4)
  }
}
```

rzono pole tekstowe (*TextBox*), w którym są wyświetlane instrukcje dla użytkownika zmieniające się wraz z użyciem przycisków umieszczonych na tej zakładce.

Pierwszą czynnością, którą należy wykonać jest nawiązanie połączenia z urządzeniem. Do tego służy przycisk „Połącz" oraz funkcja obsługi wywoływana po kliknięciu na ten przycisk (**listing 18**).

Listing 19. Funkcja obsługi wywoływana po kliknięciu przycisku „Ustaw Trigger Word"

```
//StmAnalyzer.cs
private void sendCaptureButton_Click(object sender, EventArgs e)
{
  sendCapture();
}

//StmAnalyzer.cs
private void sendCapture()
{
/* Aby zacząć wymianę z sprzętem dobrą praktyką jest sprawdzenie czy nadal sprzęt jest podłączony. */
  if (stm != null && stm.IsConnected)
  {
/* Dane wysyłane są w specjalnej strukturze zwanej raportem. */
    HidReport outputReport =
    new HidReport(stm.Capabilities.OutputReportByteLength);
/* Klasa HID używa numerów raportów, w raporcie należy  ustawić jeden z wykorzystywanych numerów raportów wyjściowych.
*/
    outputReport.ReportId = 0x01;
/* Następnym punktem jest wprowadzenie danych do przysłania, w naszym przypadku wysyłamy jeden bajt (wartość rejestru,
więc w tablicy danych pod elementem zerowym ustawiamy to, co chcemy wysłać. */
    outputReport.Data[0] = calculateLowC();
/* Utworzony raport wysyłamy do urządzenia. */
    stm.WriteReport(outputReport)
/* Po wysłaniu raportu odczekujemy pewien czas, w tym wypadku 30 ms, aby nie przepełnić bufora urządzenia i nie utracić
cennych danych. */
    Thread.Sleep(30);
     .
/* Wysyłamy wszystkie rejestry po kolei. */
     .
     .
/* Następnie wysyłamy wartość maski dla odpowiedniego rejestru. */
    outputReport.ReportId = 0x01;
    outputReport.Data[0] = calculateLowCMask();
    stm.WriteReport(outputReport);
    Thread.Sleep(30);
     .
     .
/* Powtarzamy dla reszty rejestrów. */
    startCaptureButton.Enabled = true;
    textBox.Text="W celu rozpoczęcia pomiarów wciśnij przycisk ‚Start'";
    progressBar.Value = 0;
    showChartButton.Enabled = false;
    saveButton.Enabled = false;
  }
  else
  {
    MessageBox.Show(„Sprzęt niepodłączony.\nPodłącz urządzenie, a następnie spróbuj ponownie.", „Błąd");
  }
}
```

Listing 20. Funkcja uruchamiająca analizator po kliknięciu na przycisk „Start"

```
\\StmAnalyzer.cs
private void startCaptureButton_Click(object sender, EventArgs e)
{
  startCapturing();
}

\\StmAnalyzer.cs
private void startCapturing()
{
  if (stm != null && stm.IsConnected)
  {
    HidReport outputReport=
    new HidReport(stm.Capabilities.OutputReportByteLength);
//Dla sygnału wewnętrznego (DMA) ustalono typ 0.
    byte type = 0;
/* Pobieramy numer wybranej opcji związanej z liczbą wysyłanych próbek, np. dla 100 próbek jest to 2. */
    int zakres = zakresBox.SelectedIndex; (2)
/* Mamy 5 możliwości wyboru liczby próbek, więc potrzebujemy 3 bity, aby tę informację zakodować. Kodujemy ją na
bitach: 2, 3, 4. */
    zakres =zakres << 2;
    if (sygnalBox.SelectedItem.Equals(„Zewnętrzny"))
      if (zboczeBox.SelectedItem.Equals(„Zbocze opadające"))
//Dla zbocza opadającego ustalamy typ 1.
      type = 1; else
        if (zboczeBox.SelectedItem.Equals("Zbocze rosnące"))
//Dla rosnącego typ 2.
        type = 2; else
//Dla opadającego lub rosnącego typ 3.
          type = 3;
/* Typ zegara zakodowany został na bitach 0 i 1. Dodając do tego zakres, otrzymujemy pełne opcje. */
    type =(byte) (type + ((byte) zakres));
    outputReport.ReportId = 0x04;
//Opcja zostają wpisane na pierwszy element tablicy danych do wysłania.
    outputReport.Data[0] = type;
    stm.WriteReport(outputReport);
    zakresDanych = zakresy[zakresBox.SelectedIndex];
    progressBar.Value = 0;
// Następnie tworzony jest wątek, który będzie odpowiadał za odbieranie danych.
    receiveData = new Thread(readData);
//Uruchamianie wątku.
    receiveData.Start();
  } else MessageBox.Show(„Nie nawiązano połączenia z STM32 !", „Błąd");
}
```

Po połączeniu się z urządzeniem należy wysłać wprowadzony *trigger word*. W tym celu utworzono przycisk „Ustaw Trigger Word" oraz zamieszczoną na **listingu 19** funkcję jego obsługi po kliknięciu. Na listingu podano również, jak należy poprawnie wysyłać dane do urządzenia, korzystając z klasy HID.

Aby rozpocząć działanie analizatora, należy wcisnąć przycisk „Start", który wyśle do urządzenia informacje na temat wybranego sygnału, zbocza oraz liczby rejestrowanych próbek. Stworzy przy tym wątek, który będzie czekał na wynik oraz odbierze zgromadzone dane od urządzenia (**listing 20**).

Na **listingu 21** pokazano funkcję wątku uruchamianego po wciśnięciu przycisku „Start". Demonstruje ona również sposób, w jaki należy odbierać dane z analizatora.

Zapisywanie danych do pliku

Do zapisywania danych do pliku posłużył *SaveFileDialog*. Przed jego użyciem należy ustalić, które pliki chcemy zapisywać. W tym celu wpisujemy następującą linijkę kodu (w module StmAnalyzer.cs, w funkcji public StmAnalyzer): *SaveData.Filter = „txt (*.txt)|*.txt|Wszystkie pliki (*.*)|*.*";*

Po tym jak uaktywni się przycisk „Zapisz dane do pliku", co oznacza odebranie kompletnych danych od analizatora, możemy go wcisnąć, co spowoduje pojawienie się okna zapisu do pliku (**rysunek 3**). Należy obsłużyć wybór pliku do zapisu poprzez implementację funkcji poprzez dodanie akcji oraz jej obsługę, co zeprezentowano na **listingu 22**.

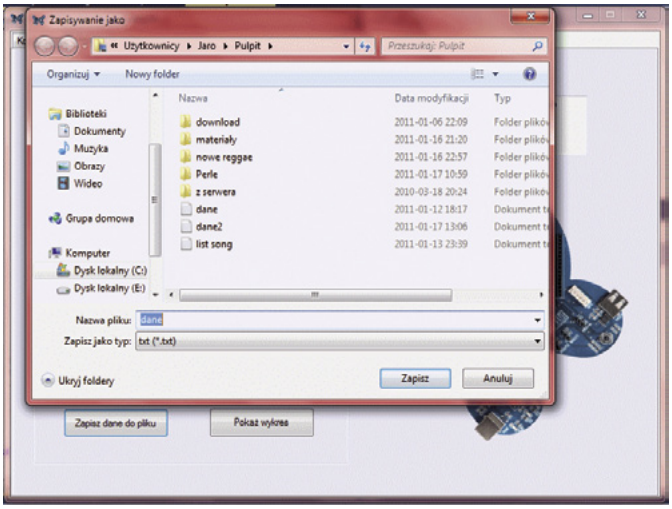
Generowanie i rysowanie wykresu

Do generowania i rysowania wykresów użyto darmowej biblioteki *ZedGraph*, która jest dostępna na licencji **LGPL.** Udostępniona biblioteka, dokumentacja oraz przykłady znajdują się na stronie *http://www.zedgraph.org*.

Formatkę wyświetlającą wykres stworzono na podstawie przykładu ze strony ZedGraph (*http://sourceforge.net/projects/zedgraph/files/*).

Listing 21. Wątek uruchamiany po wciśnięciu „Start"

```
\\StmAnalyzer.cs
private void readData()
{
  stop = false;
  lowCCount=0;
  highCCount=0;
  lowDCount=0;
  highDCount=0;
  lowECount=0;
  highECount=0;
  int received =0;
  textBox.Text="STM32 Butterfly działa i poszukuje słowa kluczowego. Czekaj na wyniki lub ustaw inne słowo kluczowe
i wciśnij ponownie 'Start'.";
  int toReceive = zakresDanych*2+1;
/* Podobnie jak przy wysyłaniu tak również przy odbieraniu dobrą praktyką jest sprawdzenie, czy nadal aplikacja ma
łączność z sprzętem. */
  if (stm != null && stm.IsOpen)
  {
/* Tworzymy raport wejściowy, do którego odebrane zostaną dane. */
    HidReport input =
    new HidReport(stm.Capabilities.InputReportByteLength);
    startCaptureButton.Enabled = false;
    while (!stop)
    {
/* Instrukcja ta powoduje wczytanie z bufora wejściowego raportu wysłanego przez sprzęt. Jeśli w danej chwili bufor
jest pusty, to wątek zawiesza się na wywołaniu ReadReport, aż do momentu pojawienia się danych. */
      input = stm.ReadReport();
/* W zależności od numeru odebranego raportu możemy wykonać odpowiednią akcję związaną z odebranymi danymi. */
      switch (input.ReportId)
      {
        case 0x01:
/* Odebrane dane, w naszym przypadku jeden bajt, znajdują się w pierwszym elemencie tablicy danych odebranego raportu.
*/
          lowCs[lowCCount] = input.Data[0];
          lowCCount++;
          break;
        case 0x02:
          highCs[highCCount] = input.Data[0];
          highCCount++;
          break;
        case 0x03:
          lowDs[lowDCount] = input.Data[0];
          lowDCount++;
          break;
        case 0x04:
          highDs[highDCount] = input.Data[0];
          highDCount++;
          break;
        case 0x05:
          lowEs[lowECount] = input.Data[0];
          lowECount++;
          received++;
/* Przy odebraniu ostatniego rejestru, co jest równoważne odebraniu stanu wszystkich 32 linii w pewnej jednostce czasu,
zmieniana jest wartość progressbar. */
          progressBar.Value = received*100/toReceive;
          break;
        case 0x06:
/* Odebranie wiadomości o numerze raportu 6 jest jednoznaczne ze skończeniem wysyłania przez sprzęt danych, w tym
momencie mamy dane potrzebne do dalszej obróbki. */
          stop = true;
          startCaptureButton.Enabled = true;
          break;
      }
    }
    progressBar.Value = 100;
/* Funkcja ta jest funkcją odwrotną do funkcji calculate(Low|High)(C|D|E), na podstawie rejestrów wylicza ona
dwuwymiarową tablicę reprezentującą stany 32 linii względem czasu. */
    makeLinesTable();
    textBox.Text = „Otrzymane dane można obejrzeć na wykresie lub zapisać do pliku";
    saveButton.Enabled = true;
    showChartButton.Enabled = true;
  } else {
    MessageBox.Show(„Nie nawiązano połączenia z STM32 !", „Błąd");
  }
}
```



Rysunek 3. Okno zapisu do pliku

Przykład ten dostosowano odpowiednio do potrzeb analizatora stanów logicznych.

Aplikacja główna wyposażona została w przycisk „Pokaż wykres", który uaktywniony zostaje tylko wtedy, gdy aplikacja poprawnie odebrała wszystkie dane (**listing 23**).

Dodawanie danych do wykresu i zmiana parametrów wykresu znajduje się jako obsługa akcji załadowania okna private void Form1_Load(object sender, EventArgs e). Dane otrzymane z głównej aplikacji należało odpowiednio wprowadzić, aby mogły być w przejrzysty sposób przedstawione na wykresie. Funkcję formatującą wykres zamieszczono na **listingu 24**, natomiast na **listingu 25** pokazano listę ważniejszych parametrów. Wykres zostanie wyświetlony po wywołaniu funkcji ładującej:

zg1.AxisChange();

Tabela 1. Sposób dołączenia sygnałów do płytki analizatora

GPIOC		GPIOD		GPIOE	
L4	GND	L12	GND	L25	GND
L5	L8	L13	L16	L26	L29
L6	L9	L14	L17	L27	L30
L7	L10	L15	L18	L28	L31
+	L11	+	L19	+	L32

JP1	-	-	L1	-	-	-	-	L3	L20	-	-	-	L24	L23	L22	L21
JP2	-	-	L2	-	-	-	-	-	-	-	-	-	-	-	-	-

zg1.Invalidate();

Po wciśnięciu prawego przycisku myszy mamy do wyboru operacje związane z narysowanym wykresem, możemy go zapisać do pliku w jednym z wybranych formatów (emf, png, gif, jpg, tif, bmp) lub wydrukować.

Uwagi odnośnie do użytkowania

Płytka ZL27 jest w stanie wygenerować sygnał zegarowy o maksymalnej częstotliwości około 1 MHz. W przeprowadzonych testach, urządzenie działające w trybie wyzwalania sygnałem zewnętrznym „poradziło sobie" z sygnałem taktującym o częstotliwości 1 MHz, natomiast przy zastosowaniu DMA uzyskano maksymalną częstotliwość sygnału wynoszącą około 100 kHz.

Sondy dołącza się do oznaczonych na rysunku pinów GPIOC, GPIOD, GPIOE oraz JP2 zgodnie z wykazem zamieszczonym w **tabeli 1**.

Jeśli częstotliwość próbkowania ma być równa częstotliwości sygnału zegarowego, to należy go doprowadzić do pinu PB6 portu I²C.

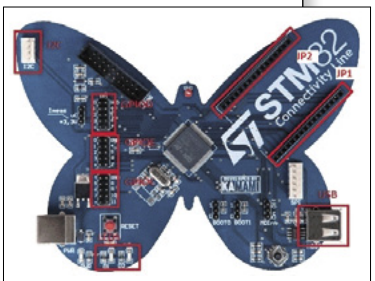
Stany pracy urządzenia są sygnalizowane za pomocą diod LED1 i LED2:

– LED1 świeci – analizator jest skonfigurowany,

Listing 22. Dodanie akcji oraz implementacja funkcji jej obsługi

```
1) Dodanie akcji:
\\StmAnalyzer.cs -  public StmAnalyzer()
this.SaveData.FileOk += new System.ComponentModel.CancelEventHandler(this.SaveData_FileOk);
...

2) Implementacji funkcji obsługi
\\StmAnalyzer.cs
private void SaveData_FileOk(object sender, CancelEventArgs e)
{
  String name;
/* Pobieramy ścieżkę do pliku pod jaką użytkownik chce zapisać dane. */
  if ((name = SaveData.FileName) != null)           (1)
  {
/* Tworzymy TextWriter, który pozwoli nam zapisać plik w postaci tekstowej. */
    TextWriter wText = new StreamWriter(name);
    for (int i = 0; i < lowCCount; i++)
    {
      for (int j = 0; j < lines.LongLength; j++)
      {
/* Zapisujemy kolejne wartości oddzielając je znakiem ';'. */
        wText.Write(lines[j][i] + ";");    (3)
      }
      if (i == lowCCount / 2)
      {
/* Dokładnie na środku znajduje się trigger word, zaznaczamy to w pliku, wstawiając na końcu linii, w której znajduje
się trigger word „*T*". */
        wText.Write(„*T*");
      }
      wText.Write(„\n");
    }
/* Po zapisaniu danych koniecznie należy zamknąć otwarty strumień. */
    wText.Close();
  }
}
```



Rysunek 4. Płytka ZL27 z zaznaczonymi złączami

Listing 23. Wyświetlanie wykresu

```
//StmAnalyzer.cs
private void showChartButton_Click(object sender, EventArgs e)
{
/* Utworzenie nowego okna, jako argument podajemy tablice dwuwymiarową lines = new byte[32][]; , która reprezentuje
stany logiczne na wszystkich liniach względem numeru pomiaru. */
  Wykres g = new Wykres(lines);
//Wykres uruchamiamy w zmaksymalizowanym oknie.
  g.WindowState = FormWindowState.Maximized;
//Wyświetlenie wykresu.
  g.Show();
}
```

Listing 24. Formatowanie wykresu

```
//Graph.cs - private void Form1_Load
/* Utworzenie linii, które zostaną dodane do wykresu. */
(PointPairList[] lists = new PointPairList[32];
/* Linia ta będzie linią pionową oznaczającą miejsce, gdzie wystąpił trigger word. */
PointPairList triggerWord = new PointPairList();
for (int i = 0; i < 32; i++)
{
  lists[i] = new PointPairList();
  for (int j = 0; j < _lines[i].Length; j++) {
/* Pobranie wartości (punktu na wykresie). */
    k =  _lines[i][j];
/* Jeśli wartość wynosi zero, to punkt znajduje się nieznacznie poniżej wartości odpowiadającej numerowi linii
w przeciwnym wypadku nieznacznie powyżej. */
    lists[i].Add(j, k==1 ? i+0.33+1 : i-0.33+1 );
/* Jeśli nastąpiła zmiana (np. z 0 na 1), to w punkcie gdzie była poprzednia wartość, wprowadzam fikcyjną wartość, aby
przejścia między stanami reprezentowane były przez pionowe linie, a nie ukośne. */
    if (j+1< _lines[i].Length &&_lines[i][j] != _lines[i][j + 1]){
        k = _lines[i][j + 1];
        lists[i].Add(j, k==1 ? i+0.33+1:i-0.33+1);
      }
    }
    LineItem li;
/* Jeśli linia jest o numerze parzystym, to zaznacz ją kolorem zielonym. */
    if (i % 2 == 0) {
      li = myPane.AddCurve("T" + i, lists[i], Color.Green, SymbolType.None);
    } else {
/* W przeciwnym razie zaznacz linię w kolorze niebieskim. */
      li = myPane.AddCurve("T" + i, lists[i], Color.Blue, SymbolType.None);
    }
    li.Line.Width = 2;
    li.Label.IsVisible = false;
  }
  k=0;
/* Dodanie linii wskazującej wystąpienie trigger word. */
  for (int i = 0; i <= 32; i++)
  {
    triggerWord.Add(_lines[0].Length/2-0.5,k);
    k += 1.5;
  }
  LineItem lin =  myPane.AddCurve("TriggerWord", triggerWord, Color.Red, SymbolType.None);
}
```

Listing 25. Lista ważniejszych parametrów wykresu

```
//Ustawienie miejsca, od którego wykres będzie widoczny w oknie.
myPane.XAxis.Scale.Min = _lines[0].Length / 2 - 0.5 - (_lines[0].Length >21 ? 19.5:9.5);
//Ustawienie miejsca, do którego wykres będzie widoczny w oknie.
myPane.XAxis.Scale.Max = _lines[0].Length / 2-0.5+(_lines[0].Length > 21 ? 20.5 : 10.5);
...
//Wykres zaczyna się na 0, przy czym pierwsza linia będzie na wysokości 1.
myPane.Y2Axis.Scale.Min = 0;
(4)Kończyć na 33, przy czym ostatnia linia będzie na wysokości 32.
myPane.Y2Axis.Scale.Max = 33;
...
//Możemy zmieniać rozmiar i przewijać w poziomie.
zg1.IsShowHScrollBar = true;
//Nie zezwalamy na zmianę rozmiaru i przewijanie w pionie.
zg1.IsShowVScrollBar = false;
//Ustawiamy, że wykres ma się automatycznie zmieniać dla naszych potrzeb.
zg1.IsAutoScrollRange = true;
```

– LED1 i LED2 świecą – analizator pobrał dane,
– LED2 świeci – analizator wysłał dane.

Po dołączeniu sond, ewentualnie doprowadzeniu zewnętrznego sygnału zegarowego, urządzenie należy połączyć z komputerem przez port USB kablem USB typu *męski-męski*. Analizator jest zasilany za pomocą napięcia występującego na złączu USB (**rysunek 4**). Po dołączeniu do komputera (system Windows XP lub nowszy) urządzenie powinno zostać wykryte jako „urządzenie interfejsu HID" a właściwy sterownik zostanie zainstalowany automatycznie.

Podsumowanie

Projekt służył zbadaniu możliwości (wydajności) omówionego sprzętu jako analizatora stanów logicznych. Po skończonej realizacji projektu i przy pełnej jego zakładanej funkcjonalności należy stwierdzić, że nie ma problemów ze zbudowaniem średniej klasy analizatora za pomocą płytki ewaluacyjnej ZL27. Należy jednak wziąć pod uwagę, że oprócz funkcjonalności oferowanej przez drogie urządzenia wyspecjalizowane do analizy sygnałów, urządzenie za pomocą popularnego mikrokontrolera monitoruje 32 linie I/O, co wiąże się z potrzebą kopiowania dużych obszarów pamięci. Prawdopodobne po ograniczeniu funkcjonalności urządzenia i przy monitorowaniu mniejszej liczby linii parametry techniczne analizatora uległyby znacznej poprawie.

Jarosław Dąbrowski
Sebastian Feduniak