Ethernetowy sterownik I/O

Moduł EM500 umożliwia łatwe podłączenie urządzenia do sieci Ethernet. O jego atrakcyjności decydują spore możliwości przy miniaturowych wymiarach wynoszących zaledwie 18,5×16×6,5 mm. Niżej prezentujemy praktyczną realizację miniaturowego sterownika z możliwością komunikacja via protokół TCP/IP w oparciu o Tibbo EM500.

Dodatkowe informacje:
Soyter Sp. z o.o., 05-082 Stare Babice, Blizne Łaszczyńskiego, ul. Warszawska 3,
tel.: 22 722-06-85 wewn. 116,
fax: 22 722-05-50, *www.soyter.pl*

Rysunek 1. Wygląd modułu Tibbo EM500

Wygląd modułu EM500 pokazano na **rysunku 1**. Wyprowadzenia są wykonane jako 22-pinowe złącze dwurzędowe. Opis funkcji poszczególnych wyprowadzeń zamieszczono w **tabeli 1**.

Moduł ma 512 kB wewnętrznej pamięci Flash, która jest dzielona pomiędzy aplikację użytkownika (320 kB) a system operacyjny TiOS (192 kB). Oprócz Flash, moduł ma pamięć EEPROM o pojemności 208 bajtów. Służy ona głównie do przechowywania nastaw, a w EM500 nabiera szczególnego znaczenia, ponieważ moduł nie ma wewnętrznego dysku Flash. System operacyjny modułu EM500 udostępnia wprawdzie obiekt o nazwie *fd.*, będący sterownikiem dysku Flash, jednak może on współpracować jedynie z pamięciami zewnętrznymi, dołączanymi za pomocą interfejsu SPI.

Pojemność pamięci RAM nieznacznie przekracza 16 kB. Pamięć ta jest współdzielona pomiędzy zmienne a bufory interfejsów komunikacyjnych. W praktyce oznacza to, że im więcej pamięci przeznaczymy na obsługę połączeń TCP, tym mniej pozostanie jej na zmienne.

Pewną trudnością w aplikacjach EM500 jest to, że wymaga on zewnętrznego układu zerowania. Producent zaleca użycie specjalizowanych układów np. MCP130-300, co nie wpływa korzystnie na budżet projektu.

Tabela 1. Wyprowadzenia modułu Tibbo EM500

Pin	Funkcja	Opis
1 (1,2,3)	GPIO0/P0.0/INT0	I/O (P0.0); linia przerwania 0
2 (1,2,3)	GPIO1/P0.1/INT1	I/O (P0.1); linia przerwania 1.
3 (1,2)	GPIO2/P0.2	I/O (P0.2)
4 (1,2)	GPIO3/P0.3	I/O (P0.3)
5 (1,2)	GPIO4/P0.4	I/O (P0.4)
6 (1,2)	GPIO5/P0.5	I/O (P0.5)
7 (1,2)	GPIO6/P0.6	I/O (P0.6)
8 (1,2)	GPIO7/P0.7	I/O (P0.7)
9 (1)	RX	Linia wejścia portu szeregowego
10 (1)	TX	Linia wyjścia portu szeregowego
11	GND	GND
12	MD	Zewnętrzny przycisk MD (w praktyce jest to kolejne linia przerwania)
13	RST	Wejście zerujące
14	SE	Linia diody stanu linku Ethernet
15	SR	Czerwona dioda statusu LED
16	SG	Zielona dioda statusu LED
17	RX-	Port Ethernet: linia RX-
18	RX+	Port Ethernet: linia RX+
19	TX-	Port Ethernet: linia TX-
20	TX+	Port Ethernet: linia TX+
21	AVCC	Wyjście napięcia zasilającego obwód zewnętrznego transformatora Ethernet
22	VCC	Zasilanie układu: 3,3 V (min. 260 mA).

Uwagi:
Linia akceptuje poziomy napięć TTL (5 V)
Może pełnić funkcję linii RTS/Wout/cout portu szeregowego
Może pełnić funkcję linii CTS/W0&1in/cin portu szeregowego

Ścieżki łączące transformator oraz gniazdo RJ45 z modułem powinny być jak najkrótsze, co wymusza umieszczenie modułu bardzo blisko gniazda. Interfejs Ethernet jest zgodny z 10/100 BaseT i auto-MDIX. Przykładowy schemat połączeń pokazano na **rysunku 2**.

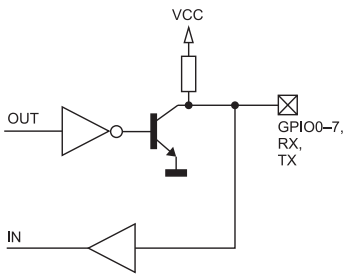
Linie wejścia-wyjścia

Poza interfejsem Ethernet moduł ma 8-bitowy port I/O oraz interfejs szeregowy SPI o prędkości transmisji do 460,8 kbps mogący pracować w jednym z kilku trybów.

Wewnętrzna budowa linii I/O wymaga krótkiego komentarza. Typowo z portem I/O powiązane są dwa rejestry. Od jednego z nich zależy kierunek transmisji a od drugiego poziom napięcia na wyjściu. W takie rejestry wyposażone są starsze moduły: EM1202, EM1206, EM1000. W module EM500 zastosowano inną koncepcję, której schemat pokazano na **rysunku 2**. Bufor wyjściowy zawiera tranzystor pracujący w układzie OE z rezystorem *pull-up* o dużej rezystancji. Tak wykonana linia I/O może przewodzić w stanie niskim prąd 10 mA, natomiast jej obciążalność w stanie wysokim jest niewielka. Ograniczenie to pozwala jednak na pracę linii w charakterze wejścia, bez potrzeby zmiany kierunku transmisji danych poprzez port. Z podobnymi rozwiązaniami portów quasi dwukierunkowych spotkamy się w mikrokontrolerach (np. z rdzeniem Intel 8051). Należy jednak pamiętać, że aby odczytać poziom logiczny napięcia wejściowego, najpierw trzeba wprowadzić tranzystor w stan zatkania.

Sieciowy sterownik linii I/O

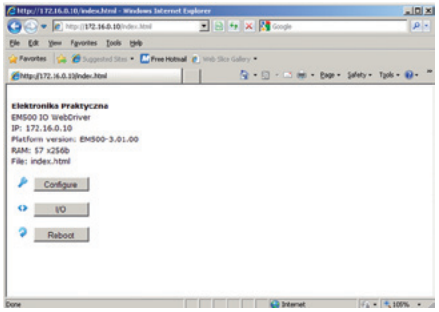
Mając wiedzę na temat działania pojedynczej linii portu I/O możemy przystąpić do utworzenia przykładowego programu.



Rysunek 2. Schemat quasi dwukierunkowej linii portu I/O

Addr: 044	Gateway
Addr: 040	Gateway Length
Addr: 024	Netmask
Addr: 020	Netmask Length
Addr: 004	IP
Addr: 002	IP Length
Addr: 001	Stor Valid Char

Rysunek 3. Mapa pamięci obszaru konfiguracyjnego



Rysunek 4. Wygląd strony index.html

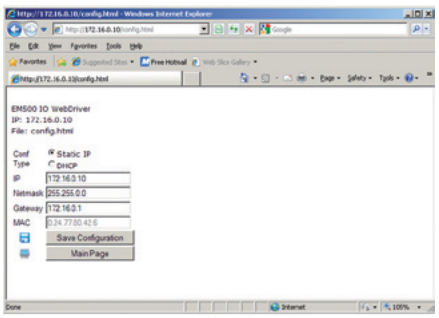
W tym celu należy pobrać instalator środowiska programistycznego TIDE w wersji 2.20.25 lub nowszej (dopiero od tej wersji TIDE jest wspierany moduł EM500) oraz obraz systemu operacyjnego TiOS w wersji 3.01.00. Instalacja systemu TiOS w module jest wykonywana za pomocą programu *Device Explorer*, instalowanego razem z TIDE.

Przykład programu jest oparty na szablonie *DHCP+setup Project*, który można wybrać przy rozpoczynaniu nowego projektu.

W pierwszej kolejności zajmiemy się konfiguracją modułu po starcie systemu. Przyjrzyjmy się **listingowi 1**, na którym zamieszczono obsługę zdarzenia systemowego *on_sys_init*. Pierwsze linie odpowiedzialne są za konfigurowanie IP modułu, aby umożliwić dostęp z poziomu przeglądarki WWW. W pamięci EEPROM jest tworzony specjalny obszar, który zawiera niezbędne nastawy. Obejmują one:

– informację czy konfiguracja IP jest statyczna czy ma zostać pobrana z serwera DHCP (dynamiczna),
– adres IP (dla konfiguracji statycznej),
– maskę podsieci,
– adres IP bramy.

Mapę pamięci tego obszaru pokazano na **rysunku 3**. Pod adresem początkowym (Addr: 01) znajduje się znak poprawności



Rysunek 5. Wygląd strony config.html

Listing 1. Obsługa zdarzenia systemowego on_sys_init

```
sub on_sys_init()
        dim dhcp_ip, dhcp_gateway, dhcp_netmask as string(16)
        dim dhcp_result as ok_ng
        dim ipcfg as ip_config
,perform IP configuration. First chceck if there is a valid configuration in
stor
        if stor_valid() = NG then
stor_ip_set(STOR_DEFAULT_CONF, STOR_DEFAULT_IP, STOR_DEFAULT_NM, STOR_DEFAULT_
GW) ,if not, create the default configuration
        end if
,get stor configuration
        ipcfg = stor_ip_get()
,examine if we go for dhcp configuration
        if ipcfg.conf = DHCP then
                dhcp_lease_time=0 ,we clear this first
        dhcp_result=dhcp_obtain(dhcp_ip,dhcp_gateway,dhcp_netmask,dhcp_lease_
time,"")
                if      dhcp_result = OK then set_ip(dhcp_ip, dhcp_gateway,
dhcp_netmask)  ,dhcp ok
                else
                        set_ip(ipcfg.ip, ipcfg.gateway, ipcfg.netmask)
,dhcp ng - use static conf
                end if
        else
                set_ip(ipcfg.ip, ipcfg.gateway, ipcfg.netmask)  ,use static
conf
        end if
,end: perform IP configuration. First chceck if there is a valid configuration
in stor

,perform basic sock configuration
        init_sock()
        io_init()
end sub
```

Listing 2. Fragment pliku nagłówkowego *config.tbh* zawierającego domyślną konfigurację IP

```
#define STOR_DEFAULT_CONF  (STATIC_IP)
#define STOR_DEFAULT_IP    („172.16.0.10")
#define STOR_DEFAULT_NM    („255.255.0.0")
#define STOR_DEFAULT_GW    („172.16.0.1")
```

List. 3. Obsługa zdarzenia przycisku MD

```
sub on_button_pressed()
,reset the ip setings to the default and reboot:
        stor_ip_set(STOR_DEFAULT_CONF,STOR_DEFAULT_IP,STOR_DEFAULT_NM,STOR_
DEFAULT_GW)
        sys.reboot
end sub
```

List. 4. Fragment pliku *io.html*

```
...
<br>
        EM500 IO WebDriver
<br>
IP:
        <?
                sock.setdata(net.ip)
                sock.send
        ?>
<br>
        File: io.html
<br>
...
<form action="io_get_set.html" method="get">
        <input type="checkbox" name="P0.0" value="chcked"
        <?
                io.num = PL_IO_NUM_0_INT0
                if io.state = HIGH  then
                        sock_setsend(„checked")
                end if
                if io.enabled =NO then
                        sock_setsend(„ disabled")
                end if
        ?>
        >P0.0
...
```

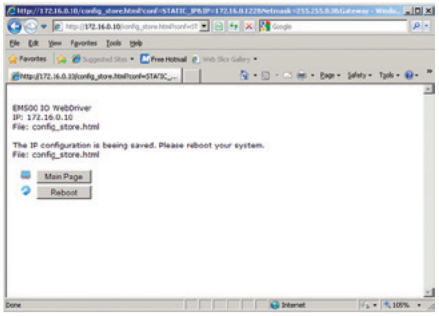
List. 5. Fragment pliku *io_get_set.html*

```
...
                File: io_get_set.html

        <?
                dim ios as io_array_type

                ios = http_req_2_io(sock.httprqstring)
                io_set(ios)
        ?>

        <meta HTTP-EQUIV="REFRESH" content="0; url=http://<? sock_
setsend(net.ip) ?>/io.html">
...
```



Rysunek 6. Wygląd strony reboot.html

konfiguracji. Jest on sprawdzany tuż po starcie za pomocą funkcji *stor_valid()*. Po pierwszym uruchomieniu programu można spodziewać się, iż poprawna konfiguracja nie zostanie odnaleziona. W takim przypadku jest wykonywany podprogram *stor_ip_set()*, zadaniem którego jest zapamiętanie nastaw w pamięci nieulotnej. Korzysta on przy tym z nastaw domyślnych, które podane są w pliku *config.tbh* (**listing 2**). Następnie zostaje wywołana procedura odczytująca nastawy z pamięci EEPROM *stor_ip_get()*, a rezultat jej pracy jest zapamiętywany w zmiennej *ipcfg* typu *ip_config*. Ten typ jest zdefiniowany w pliku *config.tbh*:

```
type ip_config
     conf as ip_config_type
     ip as string
     netmask as string
     gateway as string
end type
```

W dalszej kolejności jest sprawdzane pole *ipcfg.conf* określające czy moduł ma być skonfigurowany statycznie, czy pobrać nastawy z serwera DHCP. Do ustawiania adresów jest używana procedura *set_ip()* dostarczana z szablonem.

Jako kolejna przeprowadzana jest podstawowa konfiguracja gniazd TCP i HTTP. Zajmuje się tym podprogram *init_sock()*. Ponownie zostały użyte procedury dostępne w szablonie. Następnie dzięki wywołaniu procedury *serial0_setup()* jest wykonywane wstępne konfigurowanie portu szeregowego, ale nie jest on uaktywniany.

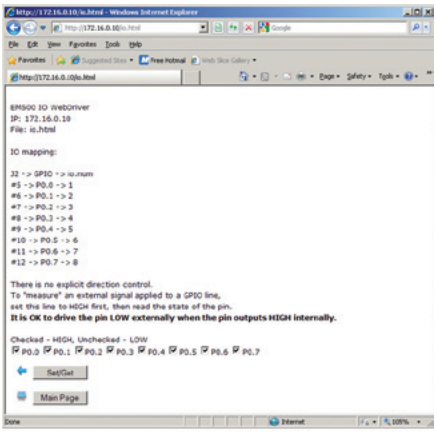
Po skompilowaniu programu źródłowego i załadowaniu go do modułu, można obejrzeć wygląd strony WWW (*index.html*), która powinna być taka, jak na **rysunku 4**. Do dyspozycji użytkownika są trzy przyciski: *Configure, I/O* oraz *Reboot*. Pierwszy z nich służy do modyfikacji ustawień sieciowych. Po jego kliknięciu moduł prześle podstronę *config.html* pokazaną na **rysunku 5**. Umożliwia ona zmianę ustawień sieciowych oraz podaje adres MAC modułu. W razie potrzeby ustawienia te możemy modyfikować, a wykonane zmiany zapisać w pamięci EEPROM po kliknięcie przycisku *Save Configuration*. Nowe ustawienia modułu zostaną wprowadzone po ponownym włączeniu zasilania lub kliknięciu przycisku *Reboot* (**rysunek 6**).

W przypadku wykonania błędnej konfiguracji można zewrzeć linię MD do masy, co powoduje wywołanie zdarzenia *on_button_pressed*. W następstwie zostaje wykonany podprogram przywracający domyślne nastawy IP (zamieszczone na list. 2). Podprogram obsługi zdarzenia od przycisku MD zamieszczono na **listingu 3**.

Linie I/O

Podstronę odpowiedzialną za sterowanie liniami portu I/O, wywoływaną po naciśnięciu przycisku *I/O* na stronie głównej, pokazano na **rysunku 7**.

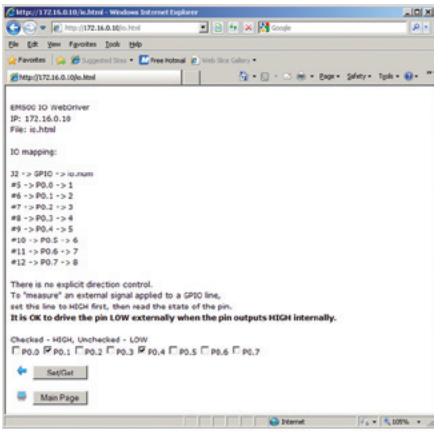
Składa się ona z formularza z ośmioma polami typu „checkbox". Każde z nich kontroluje poziom wyjściowy odpowiedniej linii I/O oraz informuje o panującym na niej poziomie logicznym. Przyjrzymy się implementacji strony *io.html*, której fragment zamieszczono na **listingu 4**. Jest on odpowiedzialny jest za „checkbox" ustalający poziom bitu 0 portu 0. Wyświetlanie pól jest zależne od aktualnego stanu linii. Odczytujemy go za pomocą *io.state*, po uprzednim wyborze numeru linii (*io.num*). W przypadku odczytu poziomu wysokiego zapisujemy do bufora nadawczego gniazda TCP łańcuch „chcecked", który parametryzuje wyświetlane pole, aby zaznaczenie było widoczne.



Rysunek 7. Wygląd strony io.html

Po załadowaniu strona pokazuje poziom I/O w momencie, w którym był on odczytany przez podprogram generujący stronę. Prześledźmy teraz, co wydarzy się, jeśli użytkownik zmieni poziom którejś z linii (**rysunek 8**).

Po kliknięciu przycisku *Set/Get*, do serwera zostaje wysłane następujące zapytanie http: *http://10.0.0.10/io_get_set.html?P0.1=chcked&P0.4=chcked*. W jego wyniku serwer wygeneruje stronę *io_get_set.html* i w rezultacie przetworzy kod tej stro-



Rysunek 8. Przykładowa zmiana stanu linii P0.0, P0.2, P0.3, P0.5, P0.6 i P0.7

ny. Najistotniejszy jej fragment pokazano na **listingu 5**. W celu odczytania informacji o nowym poziomie portu wywołujemy funkcję *http_req_2_io()*, która jako argument przyjmuje wartość własności *sock.httprqstring*. W omawianym przypadku jest to *P0.1=chcked&P0.4=chcked HTTP/1.1*. Można zauważyć, że jako parametr wywołania strony zostały dołączone jedynie zaznaczone pola formularza.

Definicję podprogramu *http_req_2_io()* pokazano na **listingu 6**. Funkcja zwraca wartość typu *io_array_type* zdefiniowaną w pliku nagłówkowym *io.tbh* w postaci struktury:

```
type io_array_type
     io(8) as off_on
end type
```

Struktura zawiera jeden element – 8-elementową tablicę zmiennych typu wyliczeniowego (*PL_OFF, PL_ON*). W pierwszej kolejności funkcja nadaje (w pętli) wszystkim elementom zwracanej tablicy wartość *PL_OFF*. Następnie jest sprawdzane czy parametr wywołania strony przekazany do funkcji poprzez własność *sock.httprqstring* (i skopiowany do zmiennej lokalnej *httpreqstr*) nie jest pustym ciągiem znaków. Jeśli by tak było, funkcja zakończy działanie i zwróci tablicę wartości *PL_OFF*. Jeśli jednak długość łańcucha znakowego jest różna od zera, wówczas jest przeprowadzana jego analiza. W pierwszej kolejności poszukiwany jest łańcuch składający się z dwóch znaków – „P0”. Wyszukiwanie jest realizowane za pomocą funkcji *instr(1, httpreqstr, „P0”, 1)*. Rozpocznie ona przeszukiwanie zawartości zmiennej *httpreqstr* (drugi argument wywołania), począwszy od jej początku (pierwszy argument wywołania). Poszukiwanym elementem jest pierwsze (czwarty argument) wystąpienie łańcucha „P0” (trzeci argument). Rezultatem jest numer pozycji poszukiwanego ciągu. W przypadku niepowodzenie funkcja zwraca 0. Całe wywołanie jest warunkiem wykonania się pętli *while*. Wewnątrz pętli, za pomocą funkcji *mid*, z ciągu wybieramy cztery pierwsze znaki zmiennej *httpreqstr*. Następnie funkcja (list. 6) przekształca ciąg znaków z postaci „*P0.x*” na liczbową wartość *x*, która tu jest numerem linii. Wartość ta przypisywana jest do zmiennej idx. Ponieważ mówimy o 8-bitowym porcie I/O, zatem zakres wartości wynosi od 0...7. W przypadku błędu funkcja zwraca wartość 0.

Kolejnym krokiem jest usuniecie z ciągu *httpreqstr* początkowych 12 znaków: „P0.x=chcked&” związanych z przetwarzaną informacją na temat aktualnej linii. Tak przygotowany łańcuch jest gotowy na następną iterację pętli. Dla obliczonego numeru linii (*idx*) wstawiamy wartość *PL_ON* w tablicy poziomów. Na samym końcu zapisujemy wartość zmiennej lokalnej *result* jako wartość funkcji.

List. 6. Definicja funkcji *http_req_2_io()*

```
,translate http request string to io action
function http_req_2_io(httpreqstr as string) as io_array_type
        dim sp as byte
        dim ep as byte
        dim l as byte
        dim i as byte
        dim result as io_array_type
        dim idx as byte
        dim ios as byte
                for i = 0 to (IO_NUM - 1)
                        result.io(i) = PL_OFF
        next i
        ,maybe there is nothing to do?
        if len(httpreqstr) = 0 then
                http_req_2_io = result
        else  ,more than one IO was checked
                while instr(1,httpreqstr,"P0",1) <> 0
                        idx = io_pxx_2_idx(mid(httpreqstr,1,4))
                        httpreqstr = mid(httpreqstr,13,len(httpreqstr)-13)
                        result.io(idx) = PL_ON
                wend
                ,check last IO
                http_req_2_io = result
        end if
end function
```

List. 6. Definicja funkcji *io_pxx_2_idx()*

```
'convert Px.x to array index
function io_pxx_2_idx(pinName as string) as byte

        select case pinName
                case "P0.0": io_pxx_2_idx = 0
                case "P0.1": io_pxx_2_idx = 1
                case "P0.2": io_pxx_2_idx = 2
                case "P0.3": io_pxx_2_idx = 3
                case "P0.4": io_pxx_2_idx = 4
                case "P0.5": io_pxx_2_idx = 5
                case "P0.6": io_pxx_2_idx = 6
                case "P0.7": io_pxx_2_idx = 7
                case else: io_pxx_2_idx = 0
        end select
end function
```

List. 7. Definicja funkcji *io_set()*

```
'sets io's state
sub io_set(io_set as io_array_type)
        dim i as byte
        for i = IO_FIRST_IO to IO_LAST_IO
                io.num = i
                if io.enabled = YES then
                        if io_set.io(i-IO_FIRST_IO) = PL_ON then
                                io.state = HIGH
                        else
                                io.state = LOW
                        end if
                end if
        next i
end sub
```

Powróćmy teraz do strony http (listing 5). Wartość funkcji zostaje przepisana do lokalnej zmiennej *ios*, która następnie jest przekazywana do funkcji *io_set(ios)* (**listing 7**). Funkcja ta w pętli nadaje poziomy odpowiednim liniom portu 0. Zakres działania pętli wyznaczają stałe *IO_FIRST_IO* oraz *IO_LAST_IO*. Symbole te zdefiniowane są w pliku nagłówkowym *io.tbh*:

```
#define IO_FIRST_IO  (0)
#define IO_LAST_IO   (7)
```

Z powyższego wynika, iż faktyczna zmiana stanu bądź konfiguracji linii odbywa się dopiero podczas przetwarzania stron związanych z przyciskiem użytkownika (io_get_set.html). Taką metodą nie możemy przeprowadzać akwizycji sygnałów z dużymi prędkościami, jednak takie podejście może z powodzeniem być używane w niewymagających dużych prędkości aplikacjach kontrolnych (np. sterowanie oświetleniem za pomocą przekaźników).

Podsumowanie

W opisywanym projekcie znikomą wartość ma rozdzielenia modułu, transformatora i złącza RJ45 w EM500. Komponenty te i tak muszą znaleźć się w pobliżu modułu, zwiększając gabaryty całego urządzenia. Potencjalnie ciekawym zastosowaniem EM500 jest jego połączenie z GA1000 i utworzenie węzła WiFi o niewielkich wymiarach. Na taką możliwość musimy jednak jeszcze poczekać, gdyż w chwili obecnej system operacyjny modułu EM500 nie wspiera GA1000 (brak implementacji obiektu *wln*.). Opisana aplikacja może służyć jako baza do bardziej zaawansowanych projektów niż tylko proste przełączanie linii I/O. Jednak w praktyce często jest potrzeba budowy jak najprostszego i najtańszego sterownika przekaźników, który dałoby się sterować poprzez Ethernet.

Marcin Chruściel, EP
marcin.chrusciel@ep.com.pl