SZKOŁA

Poziom tekstu: średnio trudny

Raspberry Pi: Arduino Nano jako alternatywne rozwiązanie

W ostatnich odcinkach tej serii zajmowaliśmy się wyłącznie Raspberry Pi. Warto teraz przyjrzeć się innym rozwiązaniom. Wybrałem Arduino Nano: jeden z mniejszych zestawów z całej rodziny Arduino. Sprawdźmy, czy można go zastosować do zadań, które dotychczas powierzaliśmy Raspberry Pi.

Środki i cele

Stare porzekadło mówi: „mierz siły na zamiary". W dziedzinie inżynierii można je przetłumaczyć na: „dobieraj środki do celów". Jest to bardzo ważne zagadnienie. Młody inżynier często patrzy na postawione przed nim zadanie najpierw przez pryzmat znanych mu technologii. Oznacza to, że nie próbuje rozwiązać problemu, tylko dopasowuje go do już znanych mu rozwiązań technicznych. W skrajnych przypadkach robi to na siłę. I zazwyczaj kończy się to źle, zarówno dla samego procesu tworzenia, jak i produktu końcowego.

Doświadczony inżynier najpierw patrzy na to, co ma być zrobione. Dopiero później rozważa, jak to wykonać. Różnica jest fundamentalna. W tym podejściu **priorytetem jest cel,** efekt końcowy. To on steruje całym projektem. Zawsze powinniście patrzeć przed siebie i zadawać sobie pytanie, jak podejmowane przez Was decyzje wpłyną na końcowe rozwiązanie. Jasny i konkretny cel pozwoli Wam szybko weryfikować pomysły i odrzucać te, które do niego nie prowadzą.

Zdefiniowanie celu nie jest wcale takie łatwe. W jednym z poprzednich tekstów („Młody Technik" 10/2014) podałem **metodę SMART**. Nazwa SMART to akronim pochodzący od angielskich słów ***s**pecific* (specyficzne, konkretne), ***m**easurable* (mierzalne, dające się określić), ***a**chievable* (możliwe do osiągnięcia), ***r**easonable* (sensowne, skutkujące postępem, innowacją) i ***t**imeable* (ograniczone czasowo, mające swój koniec w określonym czasie). W praktyce oznacza to wybieranie celów konkretnych, wymiernych, możliwych do realizacji w określonym czasie i wnoszących coś nowego – czy to do Waszego doświadczenia, czy – a jakże! – do historii świata.

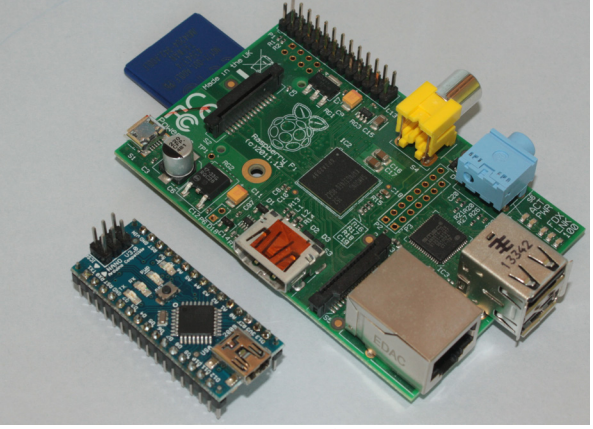
Gdy cel jest priorytetem, droga do niego wcale nie staje się mniej ważna. Jest wiele dróg do rozwiązania każdego problemu. Jedne są prostsze, drugie bardziej kręte – co wcale nie musi przesądzać o ich poprawności. Jeżeli za cel postawimy sobie zbudowanie rozwiązania uniwersalnego – droga do niego będzie bardziej skomplikowana. Rozważenie wielu przypadków, również tych skrajnych (ang. *corner cases*), zapewne wydłuży czas wykonania. Ale na końcu otrzymamy rozwiązanie, które zastosujemy jeszcze wielokrotnie. I możemy być pewni, że nie zawiedzie podczas prezentacji. Czasami jednak celem jest coś, co inżynierowie nazywają POC: ang. *proof of concept*. W tym przypadku nie chodzi o pełne rozwiązanie problemu czy dostarczenie stabilnego rozwiązania, które będzie używane przez pokolenia. Chodzi o sprawdzenie pewnych możliwości, ogólne udowodnienie, że cele są w ogóle realizowalne (ang. *achievable*).

W związku z powyższym, **nie bójcie się poszukiwać nowych rozwiązań**. Zanim zaczniecie pracę, poświęćcie trochę czasu na dokładne określenie celu. Zmieniajcie swoje podejście do problemu, oceniajcie nowe pomysły pod kątem ich wpływu na efekt końcowy. Szacujcie czas, koszty, ryzyko kolejnych pomysłów. Postarajcie się oceniać je w kategoriach liczbowych. W takim rachunku uwzględnijcie również inne potencjalne zyski – zdobyte doświadczenie, możliwość jego ponownego wykorzystania. I nie bójcie się wyzwań!

David i Goliat

Raspberry Pi (RPi) oraz Arduino Nano to zawodnicy dwóch skrajnych wag. Są na tyle różne, że porównywanie ich może wydać się niecelowe.

RPi to minikomputer. Można do niego podłączyć klawiaturę, mysz, monitor HDMI i pracować jak na zwykłym PC z Linuksem jako systemem operacyjnym. Ma wystarczająco dużo mocy obliczeniowej, aby służyć jako domowe centrum multimediów, włącznie z odtwarzaniem filmów w formacie HD. Linuks, wspierany przez całą gamę aplikacji i bibliotek, ułatwia tworzenie złożonych rozwiązań w językach programowania, takich jak Java czy Python. Umożliwia łączenie z multimediami, stawianie



1. Raspberry Pi oraz Arduino Nano

serwerów WWW i tym podobne. To taki **komputer w skali karty kredytowej.**

Nano to całkiem inny świat. Napędza go typowy mikrokontroler. Nie działa na nim Linuks, nie da rady zrobić z niego biurkowego komputera. Miniaturowe rozmiary (ok. 4,3x1,7 cm; podczas gdy RPi: 8,5x5,6 cm), minimalny pobór prądu w czasie pracy (~20 mA) i łatwe programowanie sprawiają, że **jego domeną są projekty typowo elektroniczne**.

RPi i Nano łączą wspólne cechy: kompaktowa budowa oparta na jednej płytce oraz porty, których można użyć do kontrolowania innych elementów elektronicznych (np. diód, czujników). Czy pamiętacie ciężarówkę wykonaną z klocków lego, sterowaną przez Rpi, przedstawioną w artykule z listopadowego numeru „Młodego Technika"? Wnioski z tego projektu sugerowały, że RPi nie był dla niego najlepszym wyborem. Długi start systemu, cały bagaż Linuksa z mnogością wymaganych bibliotek, problemy z kartą SD – wszystko to podpowiada, że platforma RPi była zbyt „ciężka". Spróbujmy więc przyjrzeć się Nano jako alternatywnej metodzie rozwiązania naszego „problemu ciężarówkowego".

Open Source

Zanim przejdziemy do szczegółów, warto zwrócić uwagę na to, że Raspberry Pi i Arduino są projektami bardzo mocno wspieranymi przez międzynarodową społeczność. Ich twórcy zdecydowali się na maksymalne „otwarcie" swoich produktów. Na **bazie wolnych licencji udostępnili dosłownie wszystko** – od schematów elektrycznych, projektów płytek PCB, środowisk do programowania i bibliotek. Większość z tych elementów, które zazwyczaj chroni się patentami i licencjami, jest dostępnych za darmo do wykorzystania i modyfikowania. Efekt takiego postępowania to **przechodząca wszelkie oczekiwania popularność**. Miliony ludzi na świecie budują urządzenia oparte o Raspberry i Arduino, dzielą się wynikami swoich doświadczeń, pomagają innym entuzjastom w tworzeniu ich własnych projektów (zob. [1]). Okazję zauważyły też firmy komercyjne. Dostarczają wiele komponentów, z których można łatwo budować swoje projekty. *Shield'y* dla Arduino, rozszerzenia GPIO czy nadchodzący standard *HAT* dla Raspberry – tworzenie z nich przypomina powoli budowanie z klocków. W tym sposobie liczą się inwencja, innowacja, kreatywność. Wszystko leży na stole – i od Was zależy, jak to połączycie! To wyjątkowy i niespotykany wcześniej ruch, przez niektórych określany wręcz mianem III rewolucji przemysłowej. Każdy, nawet dysponujący minimalnymi umiejętnościami, może stworzyć własny projekt nie tylko do zastosowań domowych. Dzięki serwisom typu KickStarter komercjalizacja pomysłu nie jest już żadnym problemem. Droga do bycia przedsiębiorcą jest otwarta – od Was zależy, czy zechcecie nią podążyć!

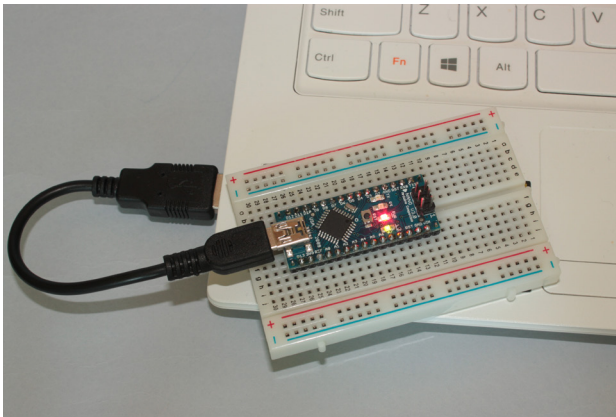
Plug&… pray?

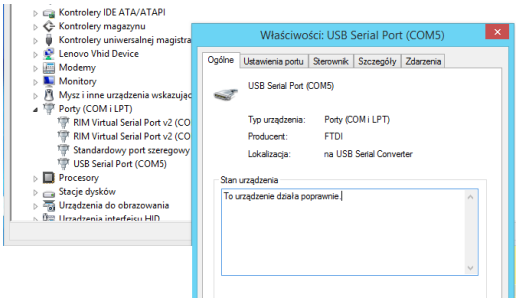
Z poprzednich tekstów tej serii wiecie już, że tworzenie środowiska dla Raspberry Pi nie jest specjalnie skomplikowane – ale też nie takie znowu trywialne. Praca z RPi wymaga kilku peryferiów (np. zasilacza, karty SD, klawiatury, monitora, przejściówki UART-do-USB), być może zmian w konfiguracji domowego rutera (np. ustawianie RPi stałego adresu IP). Wymaga też pewnego zasobu wiedzy w temacie Linuksa (ang. *learning curve*). Oczywiście w zamian otrzymujemy bardzo uniwersalny zestaw umożliwiający wiele różnych eksperymentów – nie tylko amatorskich. Ale czy można prościej?

W przypadku Nano całe peryferia sprowadzają się do… kabla miniUSB. Za jego pomocą podłącza się Nano bezpośrednio do komputera. Odpowiedzialny za komunikację szeregową układ FTDI na większości systemów operacyjnych nie wymaga instalacji żadnych dodatkowych sterowników. Programy piszemy z użyciem aplikacji „Arduino IDE", dostępnej nieodpłatnie na stronie *http://goo.gl/yOz5J7*. Jej instalacja nie powinna nastręczać żadnych problemów. Jedyną trudność może sprawić zidentyfikowanie numeru wirtualnego portu szeregowego (ang. *Virtual COM Port*, VCP), tworzonego przez system operacyjny po podłączeniu Nano do komputera. W tym celu:

- podłączcie Nano do komputera (**2**);
- otwórzcie Menedżera Urządzeń;
- w wyświetlonym drzewku rozwińcie element „Porty (COM & LPT)";
- odszukajcie „USB Serial Port (COMx)" gdzie ‚x' będzie numerem portu, np. 5

2. Arduino Nano podłączone do portu USB komputera





3. Szeregowy port VCP utworzony po podłączeniu

– zapamiętajcie go, będzie potrzebny za chwilę (**3**).

Następnie w aplikacji Arduino:

- w menu „Tools/Serial Port" ustawcie COMx;
- w menu „Tools/Board" wybierzcie model Nano, np. „Arduino Nano w/ ATMega 328".

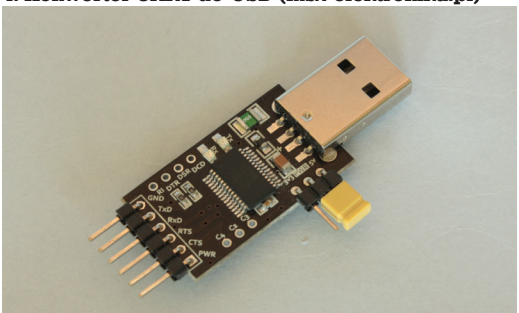
Arduino IDE wykorzysta port szeregowy VCP, żeby przesłać tworzony przez Was program do Nano. Żeby przetestować połączenie, otwórzcie przykładowy projekt: „File/Examples/01. Basics/Blink" i wciśnijcie ikonkę „Upload" (z poziomą strzałką). Jeżeli wszystko dobrze podłączyliście i ustawiliście, przykład zostanie skompilowany (tzn. przetłumaczony na instrukcje zrozumiałe dla procesora Atmel napędzającego Nano) i wysłany przez port USB na Wasz mikrokontroler. Po chwili Wasze oczy ucieszy regularnie mrugająca diodka. Brawo, Wasz pierwszy program na Nano zaliczony!

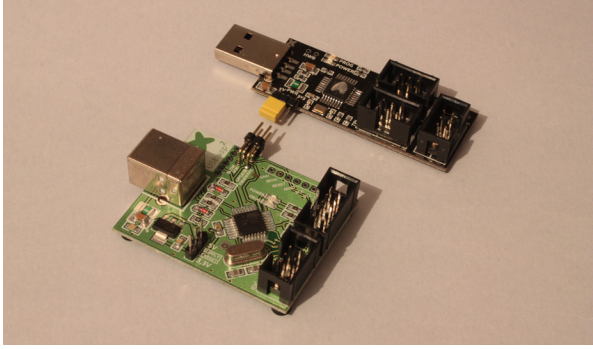
Zauważyłem jednak, że port miniUSB jest stosunkowo delikatny. Podczas któregoś z eksperymentów udało mi się go uszkodzić na tyle skutecznie, że programy muszę teraz ładować przez UART (z użyciem konwertera UART-do-USB, omawianego w listopadowym „Młodym Techniku") lub poprzez wyspecjalizowany programator ICSP (**5**).

Konwerter UART-do-USB (**4**) należy podłączyć w następujący sposób (niektóre konwertery mogą mieć trochę inne oznaczenia):

- odłączcie kabel z gniazda USB Nano; układ zasilimy teraz przez konwerter UART-do-USB;
- zworkę konwertera UART-do-USB przełączcie tak, aby zewrzeć pin opisany 5 V i środkowy; jeżeli na konwerterze nie ma takiej zworki, upewnijcie się, że działa on na logice 5 V;
- podłączcie pin RxD Nano do pinu TxD konwertera;
- podłączcie pin TxD Nano do pinu RxD konwertera;

4. Konwerter UART-do-USB (msx-elektronika.pl)





5. Programatory ICSP (msx-elektronika.pl)

- podłączcie pin 5V Nano do pinu PWR konwertera;
- podłączcie pin GND Nano do pinu GND konwertera;
- odnajdźcie port szeregowy przyporządkowany konwerterowi (jak dla połączenia Nano kablem USB powyżej);
- w programie „Arduino" ustawcie odpowiedni port szeregowy;
- załadujcie kod na Nano jak poprzednio.

Potrzebujecie razem cztery kable żeńsko-żeńskie (RTS i CTS konwertera nie muszą być podłączone). Zauważcie, że **piny danych są łączone na krzyż** – RxD z TxD.

Kolejna uwaga dotyczy klonów Arduino. Jak wspomniałem powyżej, Arduino to otwarty projekt. Na podstawie dostarczonych przez twórców materiałów, każdy producent może stworzyć własną jego wersję. Ba – możecie to nawet zrobić sami! Problem polega na tym, że niektórzy producenci nie przykładają się do pracy dostatecznie lub tworzą własne wersje, wykorzystując trochę inne komponenty np. do komunikacji. Zdarzają się klony, gdzie FTDI zastępowane są np. przez tańszy CH340G. Układy takie wymagają samodzielnej instalacji sterowników. Często nie obsługują wszystkich systemów operacyjnych (np. Win8) i nie gwarantują bezproblemowego użytkowania. Zwróćcie na to uwagę, zanim skusi Was znacznie niższa cena.

Karta SD a wewnętrzny Flash

Pamiętacie, ile problemów przysporzyła karta SD? Wszystko skończyło się dobrze, wadliwy adapter został wymieniony, ale ile to kosztowało nerwów? A wszystko przez to, że **RPi nie ma wbudowanej pamięci nieulotnej**. Pamięć nieulotna (ang. *non-volatile*) to taka, której zawartość „przeżyje" wyłączenie zasilania. RPi używa do tego celu karty SD. Tam właśnie zapisywany jest system operacyjny i system plików. Niestety, karta bywa kapryśna. Z drugiej strony, takie rozwiązanie ma wiele zalet – jest tanie, proste a przede wszystkim elastyczne. Zmiana RPi z komputerka do nauki Scratch'a w multimedialny kombajn (np. XBMC/Kodi) sprowadza się do wymiany jednej karty na drugą. W razie kłopotów można odtworzyć cały system z obrazu (kopia zapasowa). Pamiętajmy również o Raspberry Pi model B+, gdzie pełnowymiarowa karta SD została zamieniona na micro-SD i już prawie nie wystaje spoza płytki.

Wymieniono także sam czytnik na metalowy, dużo solidniejszy.

Nano nie ma czytnika kart SD. Wyposażono go za to w **pamięć typu flash i EEPROM**. Flash jest przeznaczony do przechowywania kodu programu. To bardzo popularna technologia, stosowana np. w pendrive'ach. Taka pamięć jest wbudowana w Nano – nie ma możliwości jej uszkodzenia przez wkładanie/wyjmowanie. Moja wersja ma 32 kB. Wersje z procesorem ATMega168 o połowę mniej. 2 KB flasha zajmuje tzw. bootloader – specjalny kod startujący układ i ładujący Wasze programy. Przy 16 GB karty SD w moim RPi, 32 kB to niewiele (jakieś 524 288 razy mniej), ale wbrew pozorom... najczęściej w zupełności wystarcza. Taka ilość pamięci pozwala na zapisanie całkiem pokaźnego kawałka programu wraz z bibliotekami. Przykładowy kod „Blink", który wysłaliście na Nano w poprzedniej sekcji, zajmuje ok. 1 KB.

EEPROM jest kolejnym rodzajem wbudowanej pamięci nieulotnej obecnej na pokładzie Nano. Jak wspominałem, pamięć flash służy jedynie do przechowywania programów. Z poziomu kodu nie można jej zapisać (np. stworzyć w niej pliku). Wyobraźmy sobie jednak, że chcielibyśmy zapamiętać wyniki pomiarów temperatury i mieć do nich dostęp nawet w przypadku restartu lub wyłączenia prądu. Dla RPi możemy to osiągnąć, tworząc plik na karcie. W przypadku Nano mamy do dyspozycji 1 KB EEPROM. Jak na obecne czasy nie jest to ilość oszałamiająca (1024 bajtów), ale przy odrobinie wysiłku wystarczy na wiele (zob. [2]).

Kwestia priorytetów

Wyniki pomiarów zapisane w pamięci EEPROM wyobrażam sobie jako binarne rekordy, gdzie początkowa liczba (1 bajt) to pierwsza zmierzona temperatura dnia, a następne to tzw. delty, różnice w temperaturze w stosunku do poprzedniego rozmiaru. Temperatura może rosnąć lub maleć – zarezerwuję na to jeden bit (‚1' to zmiana na plus, ‚0' na minus). Na 3 bitach mogę zapisać 8 liczb (binarnie 000, 001, 010, 011, 100, 110 i 111, dziesiątkowo: 0, 1, 2, 3, 4, 5, 6, 7), co odpowiada zmianie temperatury w ciągu godziny o 7 stopni (gdybym użył 2 bity: 00, 01, 10, 11 – o 3 stopnie). Nie jestem meteorologiem, ale wydaje mi się że to wystarczająca rozdzielczość. W pierwszej godzinie zapisuję 1 bajt, następne zmiany połówkami bajtów – po 24 godzinach mam rekord o rozmiarze 13 bajtów (temperatura początkowa i 23 pomiary pół bajtu). Ostatnie 4 bity mogą służyć jako warunek końca rekordu, np. „1000" („0000" to brak zmiany). W ten sposób na 1024 bajtach EEPROM zmieszczę: 1024/13 = 78 dni. Jestem pewien, że wymyślicie jeszcze inne, znacznie sprytniejsze sposoby upakowania danych. Czy tak samo rozwiązałbym ten problem w przypadku RPi? Ponieważ RPi nie ma EEPROMu, raczej zdecydowałbym się na zapisywanie danych w pliku, najlepiej w formacie XML. Mógłby on wyglądać tak:

```
<?xml version="1.0" encoding="UTF-8"?>
<temperatura czujnik="ds1820" id="51">
<dzien>
<dzisiaj>22.08.2002</dzisiaj>
<pomiary>
<odczyt>
<godzina>21:00</godzina>
<stan>23</stan>
</odczyt>
<odczyt>
<godzina>22:00</godzina>
<stan>20</stan>
</odczyt>
//...
</pomiary>
</dzien>
</temperatura>
```

Na cały dzień pewnie ze 2 KB zejdzie... Ale za to łatwo napisać transformację XSLT, zamieniając „surowe" dane w ładną stronę WWW, z komputera wpisać adres IP mojego RPi i obserwować wykresy przebiegów zmian.

Termometr nie był kluczowym elementem naszej budowanej z lego ciężarówki. Proszę jednak zwrócić uwagę na różnice w podejściu do zagadnienia. **W przypadku Nano liczymy każdy bit zasobów**. Dla RPi nie jest to wcale krytyczne. 10 kB czy 100 kB – w skali karty SD różnica jest niewielka. Dla Nano będziemy nawet optymalizowali ilość zapisów/odczytów z EEPROMu. Na RPi zrobi to za nas system operacyjny (zarządzanie plikami), odpowiednie biblioteki (dostęp do elementów drzewa XML) czy całe aplikacje (Apache serwujący stronę html). W przypadku układów jak Nano jesteśmy blisko sprzętu, musimy liczyć się z bardzo ograniczonymi zasobami. Za to mamy pełną władzę nad wykonaniem programu. W przypadku maszynek jak RPi – wiele ułatwia system i biblioteki, ale należy pożegnać się z pełną kontrolą. Raspbian (najbardziej popularna dystrybucja Linuksa dla RPi) domyślnie nie jest systemem czasu rzeczywistego. Może się zdarzyć, że jakiś ważniejszy proces przejmie na chwilę kontrolę i nasz pomiar nie odbędzie się dokładnie w przewidzianym momencie. Oczywiście w przypadku temperatury 50 milisekund różnicy nie zrobi. Co jednak, gdy sterujemy wypełnieniem sygnału PWM do kontroli serwomechanizmu (zobacz poniżej)? Albo kontrolujemy silnik krokowy w drukarce 3D? Oczywiście można obejść takie problemy, ale znowu – zakres wymaganej wiedzy rośnie eksponencjalnie.

Pisanie oprogramowania w Pythonie RPi jest stosunkowo łatwe, wręcz intuicyjne. Dostępne biblioteki oferują potężną funkcjonalność. Co więcej – kod piszemy i wykonujemy od razu na samej Raspberry. Najpierw jednak trzeba wszystko zainstalować i skonfigurować, upewnić się, że spełniono zależności itp. W przypadku niektórych bibliotek może to być nielichym wyzwaniem. Za przykład niech posłuży moduł „lirc", który użyliśmy do interpretacji

Tabela 1. RPi i Arduino Nano – podstawowe dane

	Raspberry Pi B	Arduino Nano	Uwagi
CPU (ang. Central Procesor Unit)	Broadcom, ARM11, 700 MHz	Atmel 328P; 16 MHz	Procesor centralny
GPU (ang. Graphical Processing Unit)	Tak	Nie	Osobny układ graficzny, np. dekodujący filmy HD
Rozmiar	85×56×21 mm	18×43×8 mm (bez pinów)	
Pamięć operacyjna	512 MB	1 kB	Pamięć, w której wykonują się programy
Wbudowana pamięć programu flash	Karta SD	32 kB	Pamięć, w której zapisuje się programy
Wbudowana pamięć danych EEPROM	Karta SD	1 kB	Pamięć dla danych
Karta SD	Tak	Nie	
Uniwersalne wyjścia/wyjścia cyfrowe	17 (+9 dla B+)	22	Dodatkowe funkcje, jak UART, I²C, SPI
Wejścia analogowe	0	8	Np. konwerter analog do cyfrowy, do pomiaru naładowania LiPo
PWM sprzętowy	1	6	Generowanie sygnału do sterowania serwami
Napięcie zasilania	5 V przez microUSB, 5 V przez pin 2 (back-power, ostrożnie!)	5 V przez USB 5 V przez pin PWR 6-20 V (przez Vin), wewnętrznie stabilizowane	
Prąd zasilania	co najmniej 400 mA	ok. 20 mA	Bez obciążeń
Prąd dostarczany na pin	20 mA	40 mA	
HDMI	Tak	Nie	
Wewnętrzny zegar RTC	Nie, usługi 'fake-hwclock' lub NTP gdy podłączony do Internetu	Nie	Jako dodatkowe akcesorium
System operacyjny	Linuks	Brak	
Orientacyjna cena	150 zł (+wymagane akcesoria)	35 zł	

sygnałów z odbiornika podczerwieni. Przyznam się, że poprawne skonfigurowanie pilota do ciężarówki zajęło mi kilka dobrych wieczorów (łącznie z doczytaniem dokumentacji). W przypadku Arduino podłączenie, dodanie odpowiedniej biblioteki i odebranie pierwszych kodów z pilota trwało… nie więcej niż 15 minut. Linuks może być zbawieniem, ale i przeszkodą.

Kwestią otwartą pozostaje, czy nastolatki, które zbudowały ciężarówkę, lepiej poradziłyby sobie z językiem C Arduino (właściwie językiem stworzonym na bazie C) niż np. Pythonem RPi (lub innym, który można użyć na tej platformie). Python ma niewielki narzut składniowy, daje natychmiastowe efekty (jest interpretowany), łatwo znaleźć błędy. I przede wszystkim jest o wiele bardziej wyrozumiały... A jak coś nie idzie w Pythonie, zawsze można przerzucić się na Scratcha, Javę czy coś równie efekciarskiego. W przypadku C Arduino kompilacja jednak trochę trwa, potem następuje ładowanie kodu na kontroler i dopiero można patrzeć na logi z portu szeregowego. Oczywiście jeżeli wcześniej zaopatrzyliśmy nasz kod w odpowiednie linijki typu Serial.println(...) drukujące wartości odpowiednich zmiennych czy kroków programu. Dodatkowo trzeba pamiętać o zamknięciu aplikacji monitora portu (wbudowany lub np. Putty), zanim zaczniemy ładować kod. Coś za coś. Oczywiście możemy zaopatrzyć się w bardziej zaawansowane narzędzia i oprogramowanie (np. Atmel Studio, dostępne za darmo), ale skala wyzwania wtedy rośnie.

Linuks czy nie Linuks?

W naszym projekcie jednym z większych **problemów RPi był stosunkowo długi czas startu**. Zanim system się załadował i uruchomił właściwy skrypt obsługi ciężarówki, mijały kolejne minuty. Podczas prezentacji w klasie, dzieci nerwowo naciskały przyciski pilota, a ciężarówka uparcie stała w miejscu. Zaczynała odpowiadać dopiero po dłuższej chwili. W tym czasie my z napięciem wpatrywaliśmy się w diodę ACT (oznacza pracę urządzenia), czy przypadkiem nie zgaśnie lub nie zacznie dziwnie migać (czytaj: regularnie, co oznacza awarię). **W przypadku Arduino, minuty potrzebne na start zamieniają się w części sekundy.** Program startuje bez zauważalnej zwłoki, co jest niewątpliwą zaletą w wielu zastosowaniach.

Oczywiście nie warto porównywać pracy, jaką wykonują RPi i Nano przy starcie. Nano nie ma nawet systemu operacyjnego! Program w języku C

jest tłumaczony bezpośrednio na instrukcje kontrolera i na nim wykonywany. Właściwie nie mówimy o starcie systemu – jest inicjacja układu i uruchomienie kodu zapisanego w pamięci flash (zadanie bootloadera). W przypadku Nano mamy tzw. ang. *bare metal* – czysty sprzęt.

Zasilanie

RPi wymaga 5 V źródła zasilania. W normalnej konfiguracji używa się do tego zasilacza podłączonego do wejścia microUSB. Model B (z portem Ethernet) wymaga źródła o wydajności co najmniej 400 mA (bez peryferiów, np. klawiatury). Zapotrzebowanie modelu A (256 Mb RAM i brak portu Ethernet) jest mniejsze, ok. 300 mA – podobnie jak nowego modelu B+. W Internecie natknąłem się na możliwość wyłączenia wyjść HDMI/PAL, co zaoszczędzi ok. 20 mA (komenda /opt/vc/bin/tvservice -off). Nadal nie są to prądy, które może dostarczyć np. bateria 9 V (zob. [3]).

Inną opcją jest pin numer 2. Normalnie dostarcza on napięcie 5 V dla podłączanych do GPIO układów. Istnieje jednak możliwość wykorzystania go do zasilenia RPi (ang. *back-power*). Problem polega na tym, że pin nie ma żadnych zabezpieczeń przeciwko skutkom spięcia, podania zbyt wysokiego napięcia czy przeciążenia. Podobne zdarzenia mogą doprowadzić do nieodwracalnego uszkodzenia RPi. Do niezależnego zasilania RPi potrzebujecie więc **stabilnego źródła 5 V, o wydajności co najmniej 400 mA**.

W projekcie ciężarówki zasilanie dostarczyła „awaryjna" bateria dla telefonów komórkowych. Jej cena była dość rozsądna (ok. 60 zł), a duża pojemność (5000 mAh) pozwala na kilkugodzinną nawet pracę (zależnie od ilości podłączonych urządzeń, sam RPi, bez silników). Model baterii, który użyliśmy (PNY Fancy Power Bank) ma dwa wyjścia, każde może dostarczyć do 1 A prądu z separacją wyjść.

Było to wygodne rozwiązanie. Jedno z wyjść użyliśmy do zasilenia RPi, drugie – do zasilenia silnika napędzającego ciężarówkę. Minusem podobnych baterii jest to, że są stosunkowo duże i ciężkie. Nasza waży prawie 150 g i ma wymiary 8x7x2,5 cm. Dodatkowo trudno ładnie ukryć wystające z nich wtyczki USB.

6. Bateria „alarmowa" użyta do zasilania ciężarówki

Nano jest znacznie bardziej elastyczne niż RPi. Można go zasilać na kilka sposobów:

- poprzez wbudowany port USB (wtyczka typu miniUSB, 5 V);
- poprzez pin 27 (oznaczony 5 V), stabilizowanym napięciem 5 V;
- poprzez pin 30 (oznaczony jako Vin lub RAW dla Mini Pro), napięciem w granicach 6-20 V.

Zwłaszcza ta ostatnia opcja wydaje się atrakcyjna. Nano ma bowiem regulator na płytce, który dostosuje napięcie z pinu Vin do wymaganego poziomu. Konsumpcja prądu podczas pracy jest o ponad rząd niższa – **w granicach 20 mA**.

Silniki

RPi nie ma wystarczająco mocy, aby napędzić co bardziej „prądożerne" peryferia. RPi może wystawić maksymalnie 20 mA na pinie. Taki prąd starcza na obsługę małych diód LED, wyświetlacza LCD z komórki, czujnika temperatury (DS18b20), czujnika odległości (HC-SR04) czy nawet niedużych serwo-mechanizmów (np. Redox s90). Na większe serwa (np. TowerPro SG-5010) czy silniki prądu zdecydowanie zabraknie. Sama RPi nie będzie potrafiła przełączać tak dużych mocy.

Konieczne jest więc stosowanie dodatkowych układów, które pozwolą na sterowanie silnikami. W naszej ciężarówce użyliśmy DRV8833. Inne, np. oparte na L293D, też spełnią swoją rolę.

Na rynku można również znaleźć zestawy wyposażone w przetwornicę 5 V (np. PiMotor DC, msx-elektronika.pl). Umożliwia to zainstalowanie pojedynczego źródła zasilania, które będzie obsługiwało zarówno RPi, jak i silniki. Tak postąpiłem w przypadku budowy innego robota samojezdnego, gdzie mogłem użyć dużo mniejszy i lżejszy pakiet LiPo 2S 7,4 V (**7**).

Użycie Nano nie wykluczyłoby potrzeby zewnętrznego zasilania dla silników. Mimo że dostarcza nawet 40 mA na nóżkę, to wciąż za mało, aby polegać jedynie na prądzie z mikrokontrolera. Oferta rozszerzeń do Arduino (tzw. shield'ów) jest nawet szersza niż dla Raspberry, nie ma więc problemu ze znalezieniem odpowiedniego do takiego projektu.

GPIO

W RPi elementy elektroniczne można podpinać do złącza GPIO. Składa się na nie 26 szpilek (dwa rzędy po 13). Piny mają różne przeznaczenie:

- masa: fizyczne piny 6, 9, 14, 20, 25;
- zasilanie 3,3 V: 1, 17;
- zasilanie 5 V: 2, 4.

Pozostałych 17 pinów traktuje się jako uniwersalne, cyfrowe porty wejścia/wyjścia. Niektóre z nich pełnią funkcje specjalne, np.:

- komunikacja po I^2C (SDA/SCL, fizyczne piny 3 i 5);
- SPI;
- UART (TxD/RxD – 10/12);
- PWM (14).

Wersję B+ wyposażono w rozszerzone, 40-pinowe złącze (dwa rzędy po 20 pinów), oferujące dodatkowych 9 portów uniwersalnych (reszta to GND i szyna I²C dla IDEEPROM).

Oprócz zasilania i masy, małe Nano oferuje aż 22 uniwersalne piny. 14 z nich to piny typowo cyfrowe (D0-D14) ze specjalnymi funkcjami, jak UART czy SPI. Kolejne 8 szpilek to piny analogowe A0-A7. Oprócz pinów A4 i A5 (przeznaczone dla I²C) mogą by używane jako piny cyfrowe.

Widać więc, że na polu liczby dostępnych portów małe Nano wcale nie ustępuje RPi. Jego piny analogowe można wykorzystać np. do wykonania miernika napięcia LiPo. Należy pamiętać o tym, że Arduino używa logiki 5 V. Nie są potrzebne żadne konwertery poziomów napięcia (ang. *level shifter*) dla czujników działających zgodnie z tym napięciem (np. czujnik odległości HC-S04). Z drugiej strony układy działające na 3,3 V będą już ich wymagać (odwrotnie niż dla Raspberry).

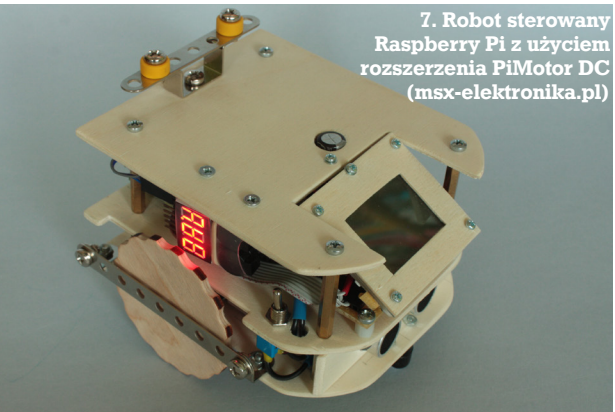
Serwomechanizmy i PWM

Serwomechanizmy (serwa) to elementy wykonawcze, które są podstawą wielu robotów. Świetnie sprawdzają się np. do przemieszczania ramion robotów, obsługi chwytaków. Po modyfikacji można ich również używać jako alternatywę dla silników. W projekcie ciężarówki taki serwomechanizm miał być odpowiedzialny za skręcanie przednich kół. Ostatecznie nie zrealizowano tej funkcji.

Zanim zdecydujemy się na użycie Raspberry w projektach opartych na serwach, należy upewnić się, czy będzie je potrafił odpowiednio kontrolować. Wiąże się to bowiem z koniecznością generowania stabilnego sygnału PWM (ang. *Pulse Width Modulation)*. W normalnej sytuacji piny przyjmują stan wysoki lub niski. PWM polega na regulacji długości trwania impulsu wysokiego, generując w ten sposób sygnał o różnym wypełnieniu. Serwomechanizmy zazwyczaj oczekują impulsu co 20 ms (zależnie od producenta wartość ta może być różna). Daje to 50 impulsów na sekundę, czyli częstotliwość 50 Hz. Sygnał trwający 1 ms ustawi serwo w pozycji zerowej, 1,5 ms – w centralnej, a 2 ms – na skrajnej.

Wyzwanie dotyczy stabilności tego sygnału. Jeżeli taktowanie nie będzie dokładne, nasze serwo zacznie zachowywać się „nerwowo". Niestety, Raspberry ma do dyspozycji tylko jeden pin do generowania sprzętowego PWM (fizyczny 12). Jeżeli macie więcej serw, sygnał PWM trzeba generować za pomocą dodatkowego rozszerzenia GPIO lub programowo. Ta ostatnia metoda wymaga wyspecjalizowanych bibliotek, które działają w czasie rzeczywistym. Jest to jednak pewne przybliżenie, a rezultaty mogą być różne.

Nano oferuje aż **6 pinów, które generują sprzętowy sygnał PWM** (3, 5, 6, 9, 10 i 11). Stanowi to dużą zaletę. Wiele projektów opiera się na Nano generującym PWM i Raspberry nim sterującym. Taki tandem bywa bardzo skuteczny.



7. Robot sterowany Raspberry Pi z użyciem rozszerzenia PiMotor DC (msx-elektronika.pl)

Co wybrać?

Jak już się pewnie zorientowaliście, nie ma jednej odpowiedzi na pytanie „co wybrać – RPi czy Nano?". Nano pochodzi z konkretnego świata: jego dziedziną są projekty elektroniczne. Jeżeli Waszym celem jest ściśle elektroniczny projekt – pewnie wybierzecie Arduino. Jeżeli jednak chcecie poeksperymentować, łączyć różne techniki – będziecie potrzebować RPi. Kiedy liczy się szybkość – wybierzecie Nano. Ale nie wyobrażam sobie, żebyście np. tworzyli serwer WWW na Nano (czy nawet na jego starszym bracie – UNO). Po co, skoro na RPi się go po prostu instaluje.

Inne zalety Arduino to:

- mniejszy rozmiar;
- mniej kłopotliwe zasilanie (regulator na płytce);
- brak konieczności konwersji poziomów logiki (mimo wszystko większość czujników działa na 5 V);
- większa liczba dostępnych wejść/wyjść, w tym analogowe;
- prostsze, bardziej zwarte środowisko – można skoncentrować się na samym programowaniu, nie na zawiłościach systemu operacyjnego;
- niższa cena.

Oprócz wspomnianych zalet, RPi oferuje:

- większy wachlarz potencjalnych zastosowań;
- większą elastyczność;
- większy zakres możliwości edukacyjnych;
- możliwość używania w bardzo różnych projektach.

Zwłaszcza ten ostatni punkt wydaje się wielką zaletą RPi. Zaraz po prezentacji mój syn wymontował RPi z ciężarówki i zajął się instalacją Minecrafta. Inne RPi działa u nas w domu jako serwer multimediów pod kontrolą XBMC/Kodi.

Ale to już całkiem inna historia. ■

Arkadiusz Merta

Źródła
[1] http://goo.gl/sbqwc0
[2] http://goo.gl/nvWxtk
[3] http://goo.gl/CwbGpw

Szanowni Czytelnicy. Ewentualne pytania do autora można kierować bezpośrednio, na adres: arkadiusz.merta@mt.com.pl