To już 12-ty odcinek kursu Raspberry Pi. Numery archiwalne MT z poprzednimi odcinkami można kupić na www.ulubionykiosk.pl

Raspberry Pi (12) łączenie logiki

Wykorzystanie Raspberry Pi do kontrolowania innych urządzeń elektronicznych otwiera przed konstruktorami całą kategorię nowych wyzwań. Nawiązanie komunikacji między RPi a np. czujnikiem odległości oznacza konieczność pokonania barier komunikacyjnych – wzajemnego rozumienia „języka", którym łączone elementy się posługują. Do tego dochodzą jeszcze zagadnienia typowo fizyczne – takie, jak kwestie zasilania (napięcie, prąd) i różnych poziomów napięcia, jakimi mogą operować układy. Są to kwestie na tyle specyficzne, że często sprawiają problemy Młodym Technikom. Co więcej – pozostawiając sprawy samym sobie, ryzykujecie uszkodzenie tych układów.

Wyposażenie Raspberry Pi w złącze GPIO umożliwia łączenie go z wieloma różnymi układami elektronicznymi. Mogą to być urządzenia wejściowe, jak czujniki zbierające dane o otoczeniu, np. odległości, koloru, dymu – a nawet kamery termowizyjne. Urządzenia wyjściowe informują (lub ostrzegają) otoczenie o wynikach przetwarzania zebranych danych. Na pewno słyszeliście o różnego rodzaju wyświetlaczach (diody LED, LCD) oraz układach generujących dźwięki (buzzery, syreny itp.). Rynek jest pełen nie tylko poszczególnych elementów, ale i całych modułów gotowych do podłączenia do jednostki sterującej. Dzięki temu projekty zaczynają przypominać budowanie z klocków.

Niestety, czasami bywa to złudne. Mimo wielu ułatwień, każdy układ elektroniczny ma swoją specyficzną naturę. Wymaga odpowiedniego napięcia zasilania, pobiera moc, komunikuje się w charakterystyczny dla siebie sposób. Nie wszystkie klocki pasują do innych – a niektóre nawet się nie lubią. Weźmy za przykład kondensatory elektrolityczne. Znajdują się w wielu modułach. Zasilenie ich niezgodnie z polaryzacją sprawi, że kondensatory zaczną puchnąć, żeby po chwili eksplodować! Oczywiście część układów będzie na taką ewentualność zabezpieczona. Ale nie wszystkie! Ostra konkurencja cenowa sprawia, że produkty są „odchudzane" o co mniej efektowne funkcje. Możecie powiedzieć, że dioda kosztuje grosze. Przemnóżcie jednak te grosze przez miliony sprzedanych sztuk, a szybko zbierze się sumka nie do pogardzenia.

Wnioski z powyższego sprowadzają się do konieczności uprzedniego dokładnego przyjrzenia się właściwościom danego modułu. Jakiego napięcia wymaga, jaki prąd pobiera, jak komunikuje się z układem sterującym, na co jest wrażliwy.

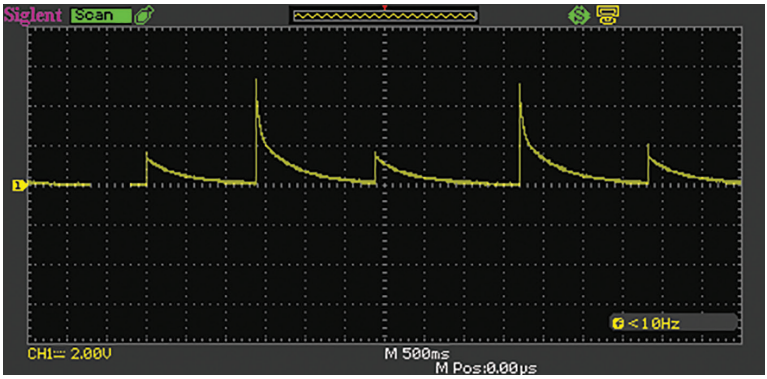
Rozumu nigdy za wiele

Wśród adeptów budowania własnych urządzeń elektronicznych (tzw. DYI – ang. *do it yourself*, czyli rodzime zrób-to-sam) obserwuję niebezpieczną tendencję do kompletnego zawierzenia wszelkim poradnikom, których całe mnóstwo można znaleźć w Internecie. Chciałbym Was szczególnie uczulić na to, że nie zawsze są one prawdziwe, aktualne i uniwersalne. Generalnie powinniście zachować dużą ostrożność i nigdy nie robić niczego na ślepo czy tylko na podstawie jednego tekstu. Często ich autorzy idą na różne skróty (specjalnie lub z niewiedzy), a przedstawione przez nich rozwiązania mogą mieć charakter bardzo prowizoryczny. Zazwyczaj zadziałają – ale czy po pół godzinie nie doprowadzą do przegrzania jakiegoś układu, znacznie skracając czas jego życia (czasami do tej właśnie pół godziny)? Czy po dłuższym okresie używania nie staną się źródłem zagrożenia? Jest to szczególnie istotne w dziedzinie budowania automatyki domowej. Tam, gdzie w grę wchodzą duże napięcia i prądy, naprawdę warto zachować rozsądek i na chłodno podejść do tematu. A zanim zaczniecie grzebać w skrzynce z bezpiecznikami, może lepiej poradzić się zawodowego elektryka?

Wiele z rozwiązań prezentowanych w Sieci jest często dokumentowanych po fakcie. Łatwo w takim przypadku zapomnieć o pewnych istotnych

szczegółach. W rezultacie zainwestujecie w projekt pieniądze (kupno komponentów), a na końcu może się okazać, że nie działa, jak powinien, lub tak, jak to wygląda na filmiku zamieszczonym przez autora.

Kolejną kwestią są same komponenty. Nigdy nie ma nic za darmo. Kupując „odpowiednik" lub „klon" kosztujący dziesięć razy mniej niż firmowy układ, musicie wziąć pod uwagę, że niższa cena nie bierze się z niczego. Oczywiście, część ceny każdego produktu stanowią koszty związane z jego projektowaniem, dystrybucją i utrzymaniem pracowników. Płacicie także za „markę". Ale część z tych kosztów to również elementy o większych tolerancjach błędów, które potrafią dłużej wytrzymać przeciążenia, są bardziej odporne na zakłócenia i spełniają odpowiednie normy środowiskowe. Pamiętajcie, że do tych norm w niektórych krajach przykłada się większą wagę niż w innych.



1. Przetwornica DC-DC, generująca piki napięcia na wyjściu USB, przy podłączaniu i odłączaniu jej od źródła zasilania

W efekcie możecie dostać wyrób modułopodobny, który w zasadzie działa, trochę działa, lub czasem działa... Może się okazać, że wymagane są np. dodatkowe kondensatory filtrujące na zasilaniu (np. klony nrf24l01), albo układ jest dużo mniej stabilny niż jego firmowy odpowiednik. Często – idąc tropem obniżania kosztów – okrojona jest też sama funkcjonalność.

Problem z niefirmowymi elementami polega na tym, że nie zawsze są one odpowiednio udokumentowane. Brakuje odpowiednich schematów. Na płytkach umieszcza się tajemnicze przyciski i zworki, których funkcjonalność można odkryć dopiero po zakupie. Dodatkową kwestią jest jakość wykonania. Zdarzają się elementy po prostu wadliwe. Jedno z moich Arduino padło ofiarą rozszerzenia (ang. *shield*), które spinało zasilanie z masą. Efekty takiego problemu są nietrudne do przewidzenia. Kiedyś natrafiłem na egzemplarz przetwornicy DC-DC z wyjściem USB. Wszystko byłoby w porządku, gdyby nie piki zasilania generowane przy jej włączaniu. Powodowały one krótkie, trwające kilkadziesiąt milisekund pojawienia się napięcia. Podłączone do USB urządzenie włączało się na część sekundy, a potem wyłączało. Nie trzeba chyba Wam tłumaczyć, co mogłoby to oznaczać dla np. twardego dysku czy Raspberry otwierającego system plików. Problem polega na tym, że są to zachowania, które bez porządnego sprzętu laboratoryjnego raczej ciężko przebadać.

W przypadku towarów będących przedmiotem prywatnego importu z Dalekiego Wschodu praktycznie nie ma możliwości wyegzekwowania gwarancji (rękojmi) ani uzyskania pomocy od producenta.

Nie jestem jednak przeciwnikiem tanich komponentów (oczywiście pod warunkiem, że nie naruszają niczyich praw autorskich, majątkowych itp.). Co więcej – uważam, że to głównie dzięki nim i ich masowej dostępności, jesteśmy świadkami boomu na DYI. Proponuję jednak, żebyście zawsze podchodzili do nich z pewną rezerwą. Oscyloskop to raczej droga „zabawka". Nawet podstawowy model kosztuje co najmniej tysiąc zł. Pozwala jednak na dokładną analizę tego, co dzieje się z układami. Nie raz uratował mi projekt. Jeżeli już macie dostateczne przekonanie, że z elektroniką związecie się na dłużej – warto w niego zainwestować.

Wiele można się również dowiedzieć za pomocą zwykłego multimetru. Pozwoli Wam stwierdzić, czy w badanym układzie nie ma przypadkiem żadnych zwarć. Warto sprawdzić pod tym kątem każdy nowy moduł, zanim podłączycie go do Waszego Raspberry.

Żeby badać zachowania już uruchomionych układów zaopatrzcie się w analizator stanów logicznych. Ceny najtańszych narzędzi tego typu zaczynają się zazwyczaj od 30 zł – i takie dla amatorów są zazwyczaj wystarczające.

Poza tym gorąco namawiam Was do czytania, czytania i jeszcze raz czytania. Nie poprzestawajcie na jednej stronie www. Spójrzcie na kilka podobnych projektów, korzystajcie z zaufanych źródeł – stron producentów lub firm specjalizujących się w aplikacji modułów i przede wszystkim oficjalnych not katalogowych. A gdy już będziecie wiedzieli, czego nie rozumiecie – poszukajcie odpowiedzi na specjalistycznych forach. Pod warunkiem zadania pytania dotyczącego unikalnego problemu, na pewno znajdzie się ktoś, kto udzieli fachowej rady.

I najważniejsze: nie odpuszczajcie. Czasami zrozumienie samego problemu zajmuje godziny, dni i tygodnie. Ale dzięki poświęconemu czasowi będziecie mogli odkryć prawdziwą naturę zagadnienia i poznać wiele szczegółów, które pozwolą Wam uniknąć podobnych sytuacji w przyszłości. Dotyczy to nie tylko samego sprzętu, ale również oprogramowania.

Biblioteki dostępne są najczęściej na zasadach wolnych i otwartych licencji. Taki model pozwala

na analizę kodu źródłowego a to z kolei – na zrozumienie reguł rządzących daną implementacją. To wspaniale podnosi poziom wiedzy, umiejętności i – moim zdaniem – jest jednym z kluczowych czynników sprzyjających rozwojowi kompetencji technicznych.

Kod udostępniany na zasadach Open Source może mieć jednak swoje wady. Czasami jest tworzony na potrzeby bardzo konkretnego projektu – wcale nie z zamiarem bycia uniwersalnym (ta uwaga dotyczy również rozwiązań sprzętowych). Może się więc zdarzyć, że nie będzie dokładnie pasował do Waszych potrzeb. Wtedy pozostaje odpowiednio go zmodyfikować (najlepiej publikując zmiany dalej, na potrzeby kolejnych „pokoleń") lub... poszukać innej biblioteki. I tak się często dzieje. Powstaje wiele „klonów" mniej lub bardziej różniących się od oryginalnej pracy – nie zawsze z nią do końca zgodnych (tzw. kompatybilność wstecz). Miksowanie takich bibliotek z oryginalnym rozwiązaniem może nie przynieść spodziewanych efektów.

(Nie)Zgodność charakterów

Udało się Wam już upewnić, że zebrane moduły nie przedstawiają bezpośredniego zagrożenia dla życia i portfela. Teraz czas na analizę, czy będą ze sobą współpracować. W tym tekście zajmę się tylko komunikacyjną częścią dopasowania elementów elektronicznych. Sprawdzimy, czy istnieją logiczne przeszkody, żeby elementy ze sobą „rozmawiały". W szczególności przeanalizujemy tu zagadnienia związane z dopasowaniem poziomów logiki.

Poziomy logiki

Binarne układy cyfrowe posługują się sygnałami logicznymi o dwóch poziomach. Określamy je jako „zero" i „jedynkę" lub sygnał „niski" i „wysoki". Sygnały te realizowane są przez impulsy o odpowiednim napięciu i minimalnym natężeniu (rzędu miliamperów). Większość układów posługuje się systemami opartymi na elementach 3,3 V (CMOS) lub 5 V (TTL). Popularnie uważa się, że np. sygnał 0 V to logiczne „0", a sygnał 3,3 V lub 5 V (w zależności od typu układu) to logiczna „1". W rzeczywistości posługiwanie się napięciami o dokładnie takich wartościach byłoby bardzo trudne do realizacji i jednocześnie mało praktyczne. Zamiast konkretnych wartości, mamy do czynienia z pewnymi przedziałami napięć uważanymi za „0" lub „1", dodatkowo różnymi na ich wejściu i wyjściu. I tak dla 5 V układów TTL (zob. [1]):

- logiczne „0" na wyjściu = 0-0,4 V;
- logiczne „0" na wejściu = 0-0,8 V;
- logiczne „1" na wyjściu = 2,7-5 V;
- logiczne „1" na wejściu = 2-5 V.

Oczywiście są to wartości teoretyczne, wynikające z budowy samych tranzystorów. Praktyczne realizacje mogą się różnić o tolerancje lub specyficzne warunki – jak w przypadku granicznego 5 V. I tak np. dla Arduino opartego na mikrokontrolerze Atmel ATMega328p dopiero napięcie 3 V na wejściu będzie oznaczało „1"; w szczególności:

- logiczne „0" na wyjściu = 0-0,9 V;
- logiczne „0" na wejściu = 0-1,5 V;
- logiczne „1" na wyjściu = 4,2-5 V;
- logiczne „1" na wejściu = 3-5 V.

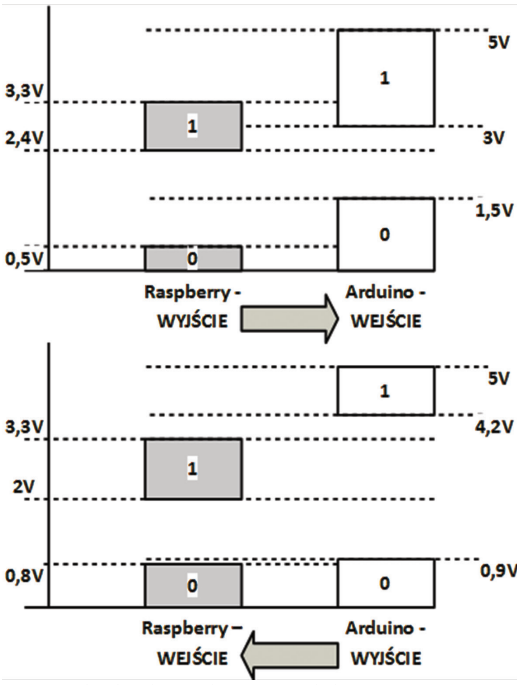
Dla CMOS'owych urządzeń 3,3 V, jakim jest CPU Raspberry (zob. [2]), poziomy te będą całkiem inne:

- logiczne „0" na wyjściu = 0-0,5 V;
- logiczne „0" na wejściu = 0-0,8 V;
- logiczne „1" na wyjściu = 2,4-3,3 V;
- logiczne „1" na wejściu = 2-3,3 V.

Zauważcie, że w obydwu przypadkach – dla TTL i CMOS – istnieje pewna „szara strefa", gdzie napięcie już nie jest „zerem" ale jeszcze nie jest „jedynką". I jak to w szarej strefie – nigdy nie wiadomo, jak zostanie to zinterpretowane.

Załóżmy teraz (czysto teoretycznie!), że podłączamy piny GPIO Raspberry bezpośrednio do portów Arduino. Stan niski na wyjściu pinu Raspberry to do 0,5 V. Arduino na wejściu za stan niski uważa napięcie do 1,5 V, więc tu się urządzenia dogadają. Stan wysoki na wyjściu Raspberry to minimum 2,4 V – na wejściu Arduino to 3 V. Arduino może więc uznać taki sygnał za szum. Praktycznie Raspberry daje na stanie wysokim sygnał pod 3 V – więc tu najprawdopodobniej nie powinno być problemów.

W drugą stronę Raspberry zrozumie „0" z Arduino. Jednak „1" z Atmela może już uszkodzić Raspberry! Przyłożenie na piny GPIO napięcia większego niż



2. Zazębianie się poziomów logiki

3,3 V (tu: 4,2-5 V) jest niebezpieczne dla Raspberry. Konieczna jest konwersja napięcia wyjściowego logiki z Arduino na odpowiedni poziom napięcia wejściowego dla Raspberry. Przeanalizujmy to na podstawie ultradźwiękowego miernika odległości HC-SR04 oraz cyfrowego czujnika temperatury DS18B20.

Ultradźwiękowy czujnik odległości HC-SR04

Ultradźwiękowy miernik odległości HC-SR04 ma cztery piny: dwa opisane jako *Trig* i *Echo* odpowiedzialne za wymianę danych, masę oraz zasilanie Vcc (zob. [3]). Czujnik wymaga zasilania 5 V i posługuje się logiką TTL. Zgodnie ze specyfikacją pobiera ok. 15 mA prądu, co mieści się w zakresie możliwości pinu 2 (5 V) Raspberry. Jego obsługa polega na podaniu impulsu na pin *Trig* (od ang. *trigger* – wyzwalacz) i obserwacji odpowiedzi na pinie *Echo*. Sekwencja wygląda tak:

- wystaw krótki sygnał wysoki na pin *Trig* (Python: time.sleep(0.0001));
- zaczekaj aż pin *Echo* zmieni swój stan na wysoki;
- zmierz czas, przez który pin *Echo* utrzymuje się w stanie wysokim;
- pomnóż zmierzony czas przez 17 150, a otrzymasz odległość do przeszkody w cm.

Napiszmy teraz w Pythonie program, który pozwoli nam przeprowadzić doświadczenie. Wymagane jest zainstalowanie biblioteki Rpi.GPIO (preinstalowana w większości nowszych dystrybucji):

```
$ sudo apt-get install python-rpi.gpio
```

Otwórzcie nowy skrypt w edytorze *nano* (na podstawie [4]; uwaga na wcięcia charakterystyczne dla Pythona):

```
$ nano dist.py

#importujemy biblioteki
import RPi.GPIO as GPIO
import time

#Numeracja portow wedlug BCM i wylaczenie ostrzezen
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

#Piny odpowiedzialne za wysylanie pobudzenia, fizyczny 7
TRIG_PIN=4
#Piny odpowiedzialne za odebranie pobudzenia, fizyczny 11
ECHO_PIN=27
#ustawiamy ktory pin jest wyjsciem
a ktory wejsciem
GPIO.setup(TRIG_PIN, GPIO.OUT)
GPIO.setup(ECHO_PIN, GPIO.IN)

#Poczekaj na ustalenie czujnika
GPIO.output(TRIG_PIN, False)
time.sleep(1)

print „Rozpoczynam pomiar”
GPIO.output(TRIG_PIN, True)
time.sleep(0.00001)
GPIO.output(TRIG_PIN, False)

#Czekamy na stan wysoki; uwaga
na wciecie
while GPIO.input(ECHO_PIN) == 0:
        impuls_start = time.time()

#to jest odpowiedz czujnika; uwaga
na wciecie
while GPIO.input(ECHO_PIN) == 1:
        impuls_koniec = time.time()

#obliczmy ilosc milisekund i odległosc
impuls_czas = impuls_koniec
- impuls_start
odległosc = impuls_czas * 17150

print „Odleglosc: „, odleglosc, „ cm”
```

Uwaga: funkcja time.time() działa z dokładnością co do dziesiątek milisekund (2 miejsca po przecinku sekund) – zależnie od systemu.

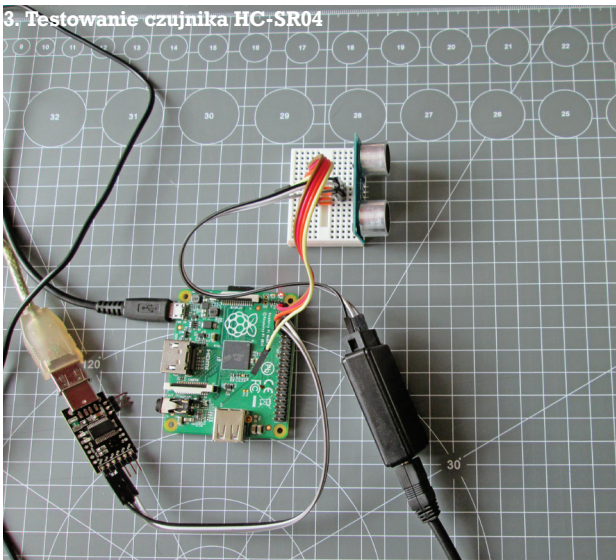
Teraz zajmiemy się podłączeniem kabelków – ale na razie bez pinu *Echo*. Połączcie więc:

- pin *GND* czujnika do pinu 9 Raspberry (ja mam na 6 podłączony UART);
- pin *VCC* czujnika do pinu 2 Raspberry (zasilanie 5 V);
- pin *Trig* czujnika do pinu 7 Raspberry (GPIO4);
- pin *Echo* czujnika – na razie niepodłączony.

Dodatkowo do czujnika podłączyłem analizator logiczny. Przejściówka UART-USB pozwoliła mi na łatwą komunikację (podłączona GND – pin 6, Rx/Tx do pinów 8 i 10). Układ testowy wyglądał jak na **ilustracji 3**.

Wydruk z analizatora logicznego znajdziecie na **ilustracji 4**. Długi impuls na *Echo* miał czas

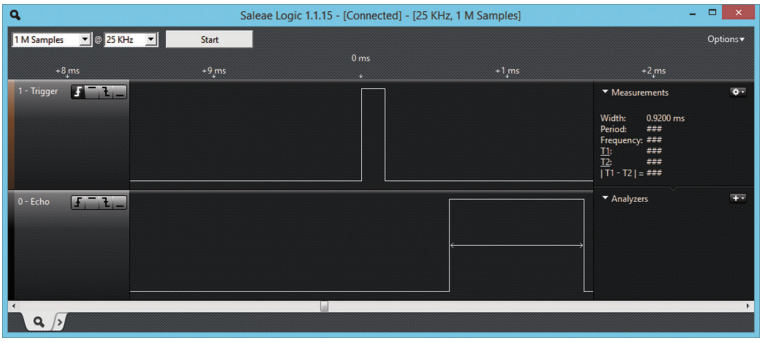
3. Testowanie czujnika HC-SR04



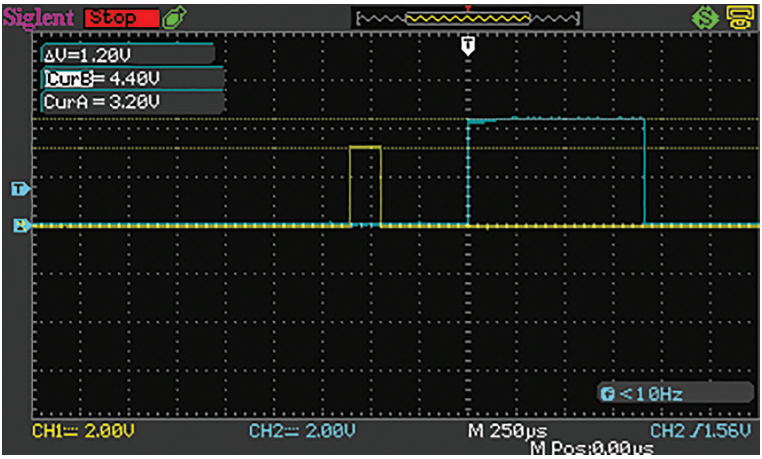
trwania ok. 0,9 ms, daje to ok. 15,5 cm – co zgadzało się z odległością czujnika od testowej przeszkody.

Pobudzenie działa, kod gotowy – dlaczego jeszcze nie podłączyłem *Echo* do Raspberry? Żeby Wam to dokładnie pokazać, zamiast analizatora logicznego użyję oscyloskopu (**ilustracja 5**).

Widać tu, że o ile pobudzenie jest na poziomie 3,3 V – odpowiedź ma 5 V! To zdecydowanie może zaszkodzić Raspberry. Wyniki zgadzają się z teoretyczną dyskusją z początku tego tekstu. Czujnik, będąc układem TTL, na wejściu jako logiczną „1” oczekuje napięcia co najmniej 2 V. CMOS'owy Raspberry na wyjściu podaje od 2,4 V. To sprawia, że czujnik „rozumie” pobudzenie z Raspberry i poprawnie na nie odpowiada. Zgodnie z teorią – odpowiada sygnałem 5 V na pinie *Echo*. Taka wartość napięcia jest jednak niebezpieczna dla Raspberry!



4. HC-SR04 pobudzony z Raspberry odpowiada na wyjściu *Echo* sygnałem o długości odpowiadającym odległości od wykrytej przeszkody

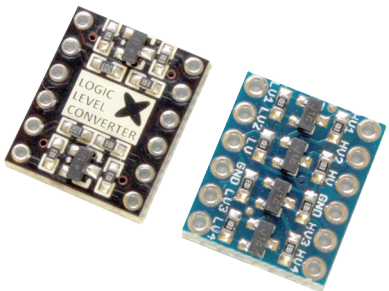


5. Pobudzenie i odpowiedź czujnika HC-SR04 na wydruku z oscyloskopu

Konwersja poziomów logiki

Do „tłumaczenia” różnych poziomów logiki stosuje się gotowe układy, tzw. konwertery poziomów logicznych. Zamieniają one sygnały 5 V na 3,3 V – w jedną lub obydwie strony. Każdy z takich modułów wymaga podłączenia zasilania 5 V i 3,3 V oraz masy. Pozostałe piny służą do łączenia linii logiki.

Poszczególne realizacje takich konwerterów mogą się od siebie znacząco różnić. Niektóre z nich oferują linie jednokierunkowe – gdzie napięcia konwertowane są jedynie z wyższych na niższe. Praca dwukierunkowa pozwala natomiast na łączenie układów porozumiewających się np. za pomocą protokołu i2c. Stosując konwertery w Waszych realizacjach, musicie być świadomi takich właściwości i odpowiednio dobierać ich piny do potrzeb (**ilustracja 6**).

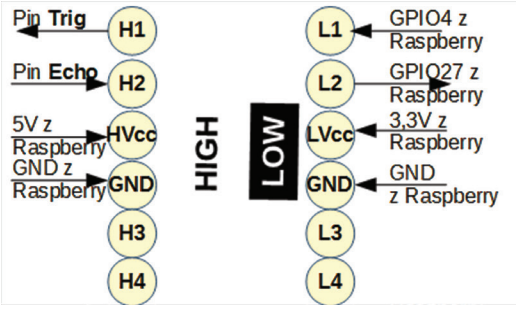


6. Konwertery logiki: niektóre linie są jednokierunkowe (z lewej produkowany przez msx-elektronika.pl)

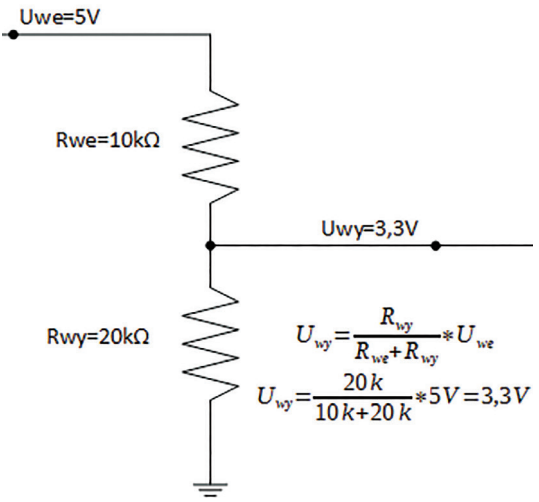
Schemat podłączenia Raspberry do czujnika odległości HC-SR04 poprzez konwerter z liniami jednokierunkowymi (lewy na ilustracji 6) przedstawiłem na **ilustracji 7**. Zauważcie, że:

- po stronie „HIGH” podłączacie napięcie 5 V z Raspberry i wejścia/wyjścia czujnika;
- po stronie „LOW” podłączacie napięcie 3,3 V i GPIO z Raspberry;
- musicie zwrócić uwagę, które z linii są jednokierunkowe – tutaj H2 do L2 i H3 do L3.

Wniosek z powyższego: konwertery są proste w używaniu, nie trzeba się ich obawiać. Ich ceny kształtują się na poziomie kilku zł, nie obciążą więc znacząco budżetu. Niestety, wymagają dodatkowych podłączeń, co w ciasnych obudowach może być kłopotliwe. Jeżeli planujecie używać wielu układów operujących różnymi poziomami logiki (np. czujniki), musicie w swoim projekcie uwzględnić kilka dodatkowych elementów i kabli.



7. Podłączenie czujnika HC-SR04 do Raspberry przez konwerter logiki z msx-elektronika.pl

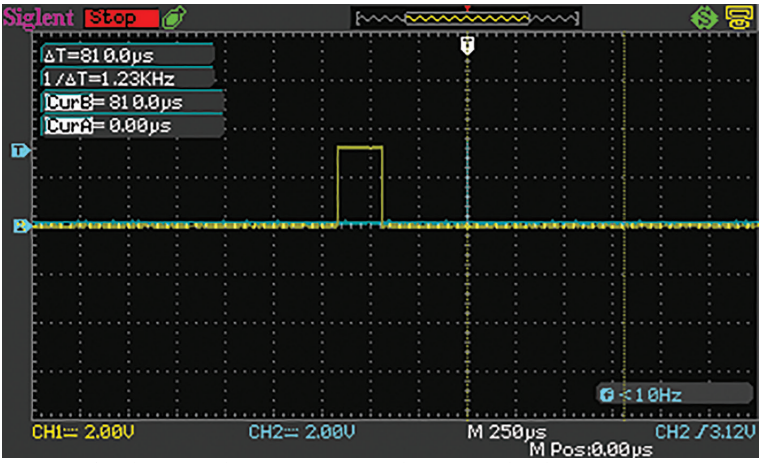


8. Dzielnik napięcia na potrzeby czujnika HC-SR04

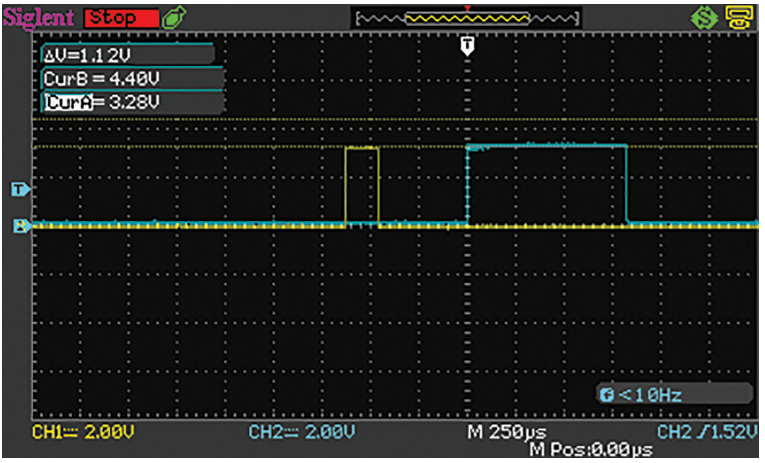
A dzielnik napięcia?

Gdy dokładniej przyjrzycie się płytkom konwerterów logiki, oferującym ruch jednokierunkowy, okaże się, że ich podstawą są dzielniki napięcia. Dzielnik napięcia opiera się na podstawowych związkach rządzących napięciem i prądem. Dzięki niemu możemy zredukować pobudzenie 5 V do 3,3 V za pomocą dwóch rezystorów (**ilustracja 8**).

Taki układ wystarczy zastosować dla pinu *Echo* i Wasz Raspberry powinien być bezpieczny. Zauważcie jednak, że nie każda kombinacja rezystorów się nadaje. Jeżeli wybierzecie zbyt małe – wydzieli się na nich duża moc i mogą ulec spaleniu. Dobierajcie duże wartości, tak jak pokazałem – np. 10 k i 20 kΩ. Pamiętajcie również, że takich dzielników napięcia można używać jedynie na potrzeby dostosowywania poziomów logiki a nie konwersji napięcia zasilania. Do tego służą przetwornice napięcia.



9. HC-SR04 nie odpowiada na pobudzenie przy zasilaniu 3,3 V

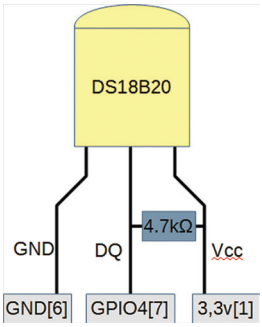


10. Inny HC-SR04, który bez problemu odpowiada na pobudzenie przy zasilaniu 3,3 V

Trochę inny HC-SR04

Okazuje się, że jeden HC-SR04 drugiemu nierówny. Realizacja konkretnego modułu może się różnić w zależności od producenta. Moduł czujnika z poprzedniego paragrafu nie odpowiadał prawidłowo po zasileniu przez 3,3 V (**ilustracja 9**).

Ale w szufladzie znalazłem taki HC-SR04, który z niskim napięciem poradził sobie doskonale! Porównajcie



11. Wyprowadzenia termometru DS18B20

ilustracje 9 i **10**. Widać stąd, że w tym drugim przypadku można używać czujnika z napięciem 3,3 V bez konieczności buforowania przez konwerter. Niestety, nie jest to zasadą, a raczej zaletą konkretnej realizacji. Weryfikacja tego faktu wymaga posiadania bardziej dokładnego sprzętu – multimetry są zbyt wolne, żeby wykryć taki impuls. Bezpieczniej założyć gorszy scenariusz i postępować zgodnie z dokumentacją.

Termometr DS18B20

Przedstawiony powyżej czujnik HC-SR04 posługuje się bardzo prostym protokołem komunikacyjnym. Pobudzony impulsem odpowiada proporcjonalnie do odległości od przeszkody. Logika komunikacji może być nawet jeszcze prostsza. Detektory podczerwone PIR zmieniają poziom sygnału wyjściowego z wysokiego na niski, gdy tylko jakiś obiekt znajduje się w ich polu widzenia.

Sytuacja komplikuje się nieco, gdy układy mają znacznie więcej do powiedzenia lub sterowanie nimi wymaga dużo więcej zabiegów. Mowa tutaj o protokołach typu 1-Wire, SPI czy wreszcie i2c. Wszystkie są wspierane przez Raspberry. Dostępny jest bardzo szeroki wybór narzędzi i bibliotek, które znacznie ułatwiają korzystanie z nich oraz urządzeń do nich podłączonych. Ich obsługa mogłaby wydawać się całkowicie „przezroczysta". Niestety, nie zawsze tak jest w rzeczywistości.

1-Wire (inaczej TWI) został opracowany przez firmę Dallas Semiconductors. Jego nazwa wywodzi się od tego, że komunikacja odbywa się po pojedynczej linii danych. Używają go m.in. bardzo popularne termometry cyfrowe DS18B20.

Wsparcie dla protokołu 1-Wire jest domyślnie wbudowane w Raspbiana. Wystarczy aktywować odpowiednie moduły (zob. [5]). Najpierw do pliku */boot/config.txt* dodajcie wpis (dotyczy najnowszych wersji z kernelem 3.18+; możecie sprawdzić wersję kernela poleceniem *uname -a*):

```
$ sudo nano /etc/config.txt
dtoverlay=w1-gpio
```

Wciśnijcie [CTRL]+[X], [Y] i [Enter], żeby zapisać zmiany. Zrestartujcie system komendą *sudo reboot*. Następnie w konsoli wpiszcie:

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
```

Możecie sprawdzić, czy moduły się załadowały:

```
$ lsmod | grep w1
w1_therm                3325  0
w1_gpio                 4502  0
wire                   31280  2
w1_gpio,w1_therm
```

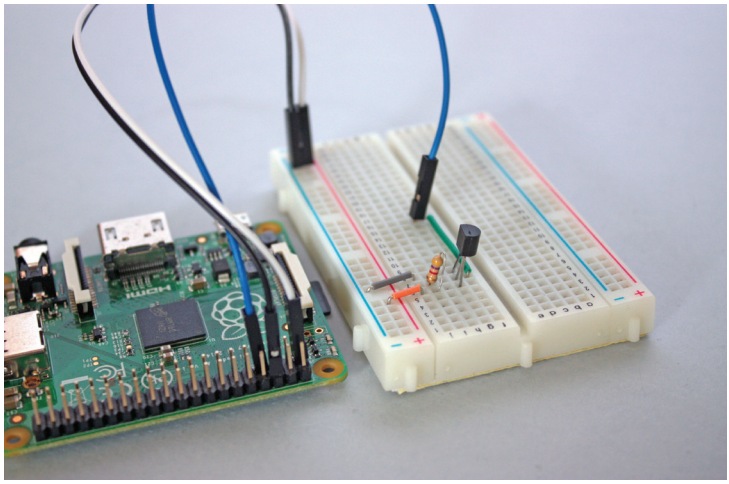
Jeżeli ich nie widzicie, warto sprawdzić plik */etc/modprobe.d/raspi-blacklist.conf*, który zawiera listę blokowanych modułów. W ostatnich wersjach Raspbiana jest pusty, ale upewnijcie się. Jeżeli znajdziecie tam wpisy zaczynające się od „w1" – usuńcie je.

Zastopujcie Raspberry poleceniem *sudo* halt i wyłączcie z prądu. Termometr wymaga podłączenia zasilania (3,3 V, 1 pin GPIO), masy (GND, 6 pin GPIO), linii komunikacyjnej i rezystora podciągającego 4,7 kΩ między linią danych i zasilaniem. Domyślnie sterowniki szukają urządzeń 1-Wire na GPIO4 (fizyczny pin 7) – i tam właśnie podłączcie linię danych (środkowy pin termometru). Upewnijcie się, że nie pomyliliście pinów zasilania i masy – inaczej czujnik nagle zacznie się szybko grzać i po chwili przestanie odpowiadać (już na zawsze).

Po podłączeniu czujnika, w katalogu /sys/bus/w1/devices znajdziecie poszczególne czujniki przedstawione jako kolejne katalogi, np.:

```
$ cd /sys/bus/w1/devices
$ ls
10-000800edbd8f
w1_bus_master1
```

Katalogi nazwane są zgodnie z unikalnymi identyfikatorami podłączonych czujników, nadawanymi im przez producenta w fabryce; tutaj akurat



12. Termometr cyfrowy DS18B20 podłączony do Raspberry A+

DC ELECTRICAL CHARACTERISTICS (-55°C to +125°C; V_{DD}=3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
Supply Voltage	V_{DD}	Local Power	+3.0		+5.5	V	1
Pullup Supply Voltage	V_{PU}	Parasite Power	+3.0		+5.5	V	1,2
		Local Power	+3.0		V_{DD}		
Thermometer Error	t_{ERR}	-10°C to +85°C			±0.5	°C	3
		-55°C to +125°C			±2		
Input Logic-Low	V_{IL}		-0.3		+0.8	V	1,4,5
Input Logic-High	V_{IH}	Local Power	+2.2		The lower of 5.5 or V_{DD} + 0.3	V	1, 6
		Parasite Power	+3.0				
Sink Current	I_L	$V_{I/O}$ = 0.4V	4.0			mA	1
Standby Current	I_{DDS}			750	1000	nA	7,8
Active Current	I_{DD}	V_{DD} = 5V		1	1.5	mA	9
DQ Input Current	I_{DQ}			5		µA	10
Drift				±0.2		°C	11

13. Wypis z instrukcji do DS18B20 (zob. [6])

10-000800edbd8f (Wasz czujnik będzie miał oczywiście inny symbol). *w1-bus_master* to sterownik magistrali. Żeby odczytać temperaturę, wystarczy wyświetlić zawartość pliku *./10-000800edbd8f/w1-slave*:

```
$ cd 10-000800edbd8f
$ cat w1-slave
36 00 52 4d ff ff 0c 10 c3 : crc=c3 YES
36 00 52 4d ff ff 0c 10 c3 t=27000
```

Podzielcie wartość parametru *t = 27000*, aby uzyskać poprawną temperaturę.

W tym przypadku system operacyjny i sterowniki załatwiają za nas całą kwestię komunikacji. Dialog między Raspberry i czujnikiem zachodzi niejako „pod spodem". Użytkownik widzi jedynie plik z efektami końcowymi. Plik *w1-slave* ma na tyle łatwy format, że można go szybko zanalizować i odczytać temperaturę, np. w skrypcie Pythonowym.

Powróćmy jednak do dyskusji na temat poziomów logicznych. Czy można bezpiecznie podłączyć DS18B20 do Raspberry? Jeżeli spojrzycie na dokumentację modułu (zob. [6]), na stronie 19 znajdziecie informację na temat zasilania i poziomów logiki (**ilustracja 13**).

Widać stamtąd, że termometr może być zasilany napięciem od 3,3 V do 5 V. Instrukcja specyfikuje jedynie „wejściowe" poziomy logiki. Ponieważ jednak komponent postępuje zgodnie z zasadami 1-Wire, można założyć, że górne ograniczenie dla wyjścia logiki będzie takie samo jak dla wejścia. Stąd, przy zasilaniu zewnętrznym (nie w trybie „pasożytniczym"), stan wysoki to co najmniej 2,2 V do mniejszej z wartości 5,5 V lub Vdd+0,3; gdzie Vdd to napięcie zasilania (oczywiście jest to związane z rezystorem podciągającym). Widać, jakim błędem byłoby zasilenie termometru (Vdd) poprzez pin 2 GPIO, który dostarczają 5 V. Jego odpowiedź na linii danych byłaby na poziomie do 5,3 V, co zdecydowanie mogłoby zaszkodzić Raspberry. Miejcie się więc na baczności.

Podsumowanie

W powyższym tekście pokazałem kilka przykładowych zagadnień z dziedziny łączenia układów elektronicznych na poziomie logiki. Jak sami widzicie, kryje się tu kilka pułapek, które mogą doprowadzić do zniszczenia obydwu układów. W tym temacie należy więc zachować dodatkową ostrożność i konsultować wszelkie instrukcje, które znajdziecie w Internecie. Ślepe zawierzanie im, może skończyć się nieprzyjemnie dla Waszego portfela. Co gorsze – rezultaty popełnionego błędu nie muszą się wcale objawić natychmiastowo w postaci efektów dymno-zapachowych. Nowoczesne układy wykazują pewną tolerancję (projektanci czasami wpisują w nie zapas tolerancji nawet większy niż w instrukcji). Ale granica jest niezwykle płynna i może objawić się dopiero po pewnym czasie. Krótkie działanie elementu nie zawsze jest więc wynikiem jego wad czy kiepskiej jakości. Degeneracja może rozciągać się i postępować, nie zawsze zauważalnie.

Postępujcie rozważnie i ostrożnie, czytajcie nie tylko sieciowe poradniki – ale i oficjalne noty aplikacyjne dostarczane przez producentów. Konsultujcie swoje i znalezione projekty. Nie polegajcie na przypadku ani szczęściu, a na pewno zmontowane urządzenia odpłacą się Wam długą i bezawaryjną eksploatacją. ■

Arkadiusz Merta

Źródła

[1] https://goo.gl/57AAlr
[2] http://goo.gl/JxK1pk
[3] http://goo.gl/nA8jW6
[4] Magazyn The MagPi, numer 27, http://goo.gl/yi7oIU
[5] https://goo.gl/VZdwSj
[6] http://goo.gl/YP4ajF