To już 15-ty odcinek kursu Raspberry Pi. Numery archiwalne MT z poprzednimi odcinkami można kupić na www.ulubionykiosk.pl

Raspberry Pi (15)

GPS jako zegar czasu rzeczywistego

W poprzednim odcinku rozważaliśmy konfigurację zegara dla Raspberry Pi odłączonego od Sieci. Pokazałem, jak działa domyślny pakiet *fake-hwclock*, jak konstruować własne układy zawierające zegar sprzętowy i używać wersji gotowych. Tym razem przedyskutujemy jedno z najdokładniejszych źródeł czasu rzeczywistego: GPS.

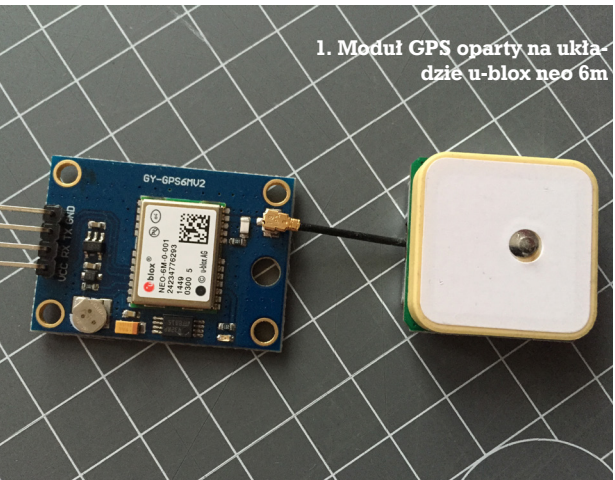
W obliczu braku innego źródła czasu, odłączony od Sieci RPi ratuje się, zapisując czas systemowy na karcie SD. Robi to w określonych odstępach czasu lub przy wystąpieniu pewnych wydarzeń, np. zatrzymaniu systemu za pomocą komendy *halt*. Potem, przy ponownym starcie, czas ten jest odczytywany z pliku i ustawiany jako systemowy. Oczywiście, przy takiej operacji, Raspberry traci okresy, kiedy był wyłączony. Tak działa pakiet *fake-hwclock*. W poprzednim odcinku dokładnie prześledziliśmy ten mechanizm.

Umiecie już skonstruować własny zegar czasu rzeczywistego RTC (ang. *real-time clock*), podłączyć go do Raspberry i odpowiednio skonfigurować. Przy wyłączeniu RPi czas systemowy zapisuje się w pamięci RTC (polecenie *hwclock --systohc*). Przy włączeniu – jest odczytywany z niego i ustawiany jako obowiązujący (polecenie *hwclock --hctosys*). Układy z RTC mają własne, niezależne od Raspberry zasilanie. Może to być bateria (jak w pokazanym rozszerzeniu PiClock) lub kondensator, jak w układzie, który sami skonstruowaliśmy. Nawet gdy wyłączycie Raspberry, RTC dalej działa i odmierza czas. Robi to całkiem dokładnie – zwłaszcza, gdy dołoży się obwody kompensujące dryft oscylatorów oraz uwzględni warunki środowiskowe (np. wpływ temperatury).

Tego typu zegary wymagają jednak pewnych zabiegów. Działają tak długo, jak ich bateria. Nie jest to zazwyczaj problemem, gdyż jedna najczęściej wystarcza nawet na kilka lat. Największym mankamentem jest jednak to, że najpierw... musimy im ustawić aktualny czas. Bez tego będą tylko tykać, odmierzając sekundy od pewnego momentu ustawionego im w fabryce. To Wy musicie zrobić ten pierwszy krok i zainicjować RTC odpowiednim momentem startowym. Później dadzą sobie już radę same (jak długo działa zasilanie).

Dobrze jednak wiecie, że istnieją źródła czasu, które nie wymagają „ręcznej ingerencji". Jednym z nich jest znany wszystkim GPS.

Uwaga! Podłączcie Raspberry do Internetu. Będziemy musieli doinstalować kilka pakietów. Bez dostępu do Sieci byłoby to zbyt skomplikowane. Gdy już wszystko skonfigurujemy, przetestujemy działanie na konfiguracji *off-line*. Do doświadczeń użyjemy dystrybucji Raspbiana z maja 2015 r. na Raspberry B+.



1. Moduł GPS oparty na układzie u-blox neo 6m

GPS (ang. *Global Positioning System*)

GPS to system, który pozwala na określenie położenia oraz czasu. Jego głównym elementem są satelity krążące wokół Ziemi (kontrolowane przez stacje naziemne). Znając położenie tych satelitów, na podstawie pomiaru czasu dotarcia sygnału od nich, odbiorniki GPS są w stanie obliczyć swoją pozycję geograficzną. Odpowiednio dokładny czas jest tu więc wartością krytyczną. Dlatego też, obok zegarów atomowych, GPS jest systemem określanym mianem wzorcowego (lub pierwotnego) źródła czasu.

Jeszcze kilka lat temu moduły GPS były zbyt drogie do zastosowań hobbystycznych. Obecnie ceny ich spadły na tyle, że nie jest problemem

zakup takiego rozszerzenia grubo poniżej 100 zł (**1**). Rozszerzenia takie komunikują się z jednostką główną poprzez USB lub UART. Tutaj opiszę ten drugi przypadek, chociaż (poza instalacją) większość operacji jest identyczna również dla GPS podłączanych przez USB.

Podłączenie do Raspberry

Żeby podłączyć GPS do UART Raspberry, wystarczą cztery kabelki:

1. zasilanie: najczęściej 3,3 V (fizyczny pin 1 Raspberry);
2. masa (np. pin 6);
3. TX GPS do pinu RX Raspberry (pin 8);
4. RX GPS do pinu TX Raspberry (pin 6).

Zwróćcie uwagę na krzyżowe podłączenie TX GPS do RX Raspberry. Upewnijcie się, jakiego napięcia zasilania wymaga Wasz moduł GPS. To raczej delikatne urządzenia. Jeżeli Wasz GPS działa z logiką 5 V, będziecie potrzebowali przekonwertować sygnał na 3,3 V, akceptowalny przez GPIO Raspberry (więcej na ten temat znajdziecie w [1]).

I tutaj pojawia się wytłumaczenie, dlaczego połączyliśmy Raspberry do komputera za pomocą kabla ethernetowego, a nie przejściówki UART-USB. Po prostu: to GPS będzie wykorzystywał UART, blokując w ten sposób dostęp do Raspberry przez konsolę szeregową.

Żeby otworzyć dla GPS komunikację po porcie UART, uniemożliwimy jego wykorzystywanie przez system. W tym celu odetniemy konsolę systemową. W pliku */boot/cmdline.txt* usuńcie wpisy podobne do „console=” (dwa wpisy) oraz „kgdboc” (edycja po wywołaniu: *$sudo nano /boot/cmdline.txt*). Linia komend powinna wyglądać wtedy podobnie do:

```
dwc_otg.lpm_enable=0 root=/dev/mmcblk0p2
rootfstype=ext4 elevator=deadline rootwait
```

Będąc w edytorze *nano*, wciśnijcie [CTRL]+[X], [Y], [Enter], żeby zapisać plik i wyjść. Następnie w pliku */etc/inittab* zakomentujcie linię (znajduje się na końcu pliku):

```
#Spawn a getty on Raspberry Pi serial line
T0:23:respawn:/sbin/getty -L ttyAMA0 115200
vt100
```

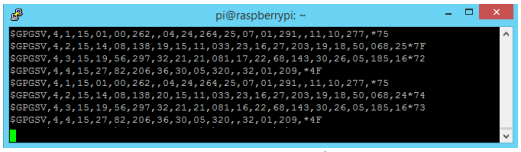
Ma wyglądać następująco:

```
#Spawn a getty on Raspberry Pi serial line
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200
vt100
```

Teraz zastopujcie Raspbiana (*$ sudo halt*) i odłączcie Raspberry z zasilania.

Uwaga! Nigdy nie podłączajcie niczego do GPIO przy włączonym zasilaniu! Może to doprowadzić do uszkodzenia Raspberry. Podłączcie GPS jak powyżej. Uruchomcie Raspberry, zalogujcie się i podejrzyjcie, co się dzieje na urządzeniu */dev/ttyAMA0*:

```
$ cat /dev/ttyAMA0
$GPRMC,175527.00,V,,,,,,,030715,,,N*7E
$GPVTG,,,,,,,,,N*30
$GPGGA,175527.00,,,,,0,00,99.99,,,,,,*65
$GPGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*30
$GPGSV,1,1,03,18,,,29,19,,,26,23,,,25*72
```



2. Wiadomości GPS na konsoli ($ cat /dev/ttyAMA0)

Wasz GPS działa i już szuka satelitów. Wiadomości, które widzicie, sformatowano zgodnie ze standardem NMEA.

NMEA

Protokołu komunikacyjnego NMEA używa wiele różnych urządzeń, głównie związanych z zastosowaniami morskimi i nawigacyjnymi. Standard określa m.in. format danych zwracanych przez urządzenia GPS. Możecie spotkać jego dwie wersje: NMEA 0183 (w skrócie NMEA) oraz NMEA 2000. Różnice polegają na szybkości transmisji danych, NMEA 2000 jest też protokołem binarnym, a NMEA 0183 – tekstowym. Mój moduł porozumiewa się za pomocą NMEA 0183.

Dane w tym standardzie transmitowane są jako sekwencje. Każda zaczyna się symbolem „$”, a kończy znakiem końca linii „<cr><lf>” (0x10, 0x13). Kolejne rekordy w sekwencji oddzielane są przecinkami. Przykładowa ramka wygląda następująco (na podstawie [2]):

```
$GPRMC,205622.00,A,4916.45,N,012311.12, W,0.
336,,140715,,,A*71
```

gdzie:
205622.00: czas pobrania namiaru, w czasie uniwersalnym (UTC) – 20:56.22;
A: ostrzeżenia odbiornika (tu OK, V: ostrzeżenie);
4916.45: szerokość geograficzna 49 stopni i 16,45 minut;
N: szerokość północna;
012311.12: długość geograficzna 123 stopnie, 11,12 minuty;
W: długość zachodnia;
0.336: szybkość, w węzłach;
140715: data namiaru: 14 lipca 2015 r.;
***71:** suma kontrolna

Standard definiuje wiele różnych typów wiadomości. Mogą one nie tylko nieść dane, ale i służyć do konfiguracji samego odbiornika.

GPSD

W następnym kroku zainstalujemy w systemie usługę GPS: *gpsd*. Zaczniemy od dodania kilku narzędzi do obsługi GPS (jak zwykle najpierw uzupełnimy informacje o dostępnych pakietach):

```
$ sudo apt-get update
$ sudo apt-get -y install gpsd gpsd-clients
```

Następnie skonfigurujemy usługę (demona) GPS (*gpsd*). W tym celu uruchomcie:

```
$ sudo dpkg-reconfigure gpsd
```

Podajcie kolejno:

- „Start gpsd automatically?”: automatyczne uruchomienie usługi GPS przy starcie systemu: wybierzcie „Yes”;
- „Should gpsd handle attached USB GPS receivers automatically?”– jeżeli macie GPS na USB: „Yes”;
- „Device the GPS receiver is attached to:” – wpiszcie: */dev/ttyAMA0*;
- „Options to gpsd:”, wpiszcie: ‘-n’ (więcej o opcjach *gpsd* znajdziecie w [3]);
- „gpsd control socket path:” **–** gniazdo danych GPS dla aplikacji; powinno być: „/var/run/gpsd.sock” (domyślne).

Teraz zrestartujcie usługę *gpsd*:

```
$ sudo service gpsd restart
```

Zauważcie, że *gpsd* zainstalowała się w systemie i będzie uruchomiana przy każdym starcie:

```
$ ls /etc/rc*/*gps*
/etc/rc0.d/K01gpsd  /etc/rc2.d/S03gpsd  /etc/rc4.d/S03gpsd  /etc/rc6.d/K01gpsd
/etc/rc1.d/K01gpsd  /etc/rc3.d/S03gpsd  /etc/rc5.d/S03gpsd
```

Wywołajcie program czytający pozycję z usługi *gpsd* ([CTRL]+[C] żeby wyjść):

```
$ cgps -s
```

Jeżeli aplikacja pokaże „3D FIX” – Wasz GPS działa i potrafi zwrócić poprawny namiar (**ilustracja 3**).

Pamiętajcie, że złapanie pełnego namiaru GPS nigdy nie jest natychmiastowe. Zależy to od samego modułu (układu, obecności podtrzymywania bateryjnego, rodzaju anteny itp.), warunków pogodowych, miejsca, w którym urządzenie jest zlokalizowane (w pomieszczeniu, na zewnątrz) i czasu, który upłynął od uzyskania ostatniego namiaru. Dla przykładu, zimny start (więcej niż sześć godzin nieaktywności) może zająć nawet sześć minut. Gorący start (do kilkunastu minut od ostatniego namiaru): pojedyncze sekundy. Do monitorowania zachowania GPS możecie też użyć programu *gpsmon*.

NTP

Jeżeli podłączycie Raspberry do Internetu, zauważycie, że czas systemu operacyjnego zostanie wkrótce zsynchronizowany z czasem rzeczywistym. RPi sam się do niego dostroi. Powodem takiego zachowania jest domyślnie uruchamiana usługa NTP (ang. *Network Time Protocol*).

NTP to protokół, który pozwala urządzeniom na odczytanie aktualnego czasu z serwerów czasu dostępnych w Internecie. Wzorcowy czas pochodzi np. z zegarów atomowych. NTP określa takie źródło jako STRATUM 0. Czas STRATUM 0 rozpowszechnia kolejna warstwa STARTUM 1. Ta z kolei stanowi odniesienie dla STRATUM 2. Każda warstwa STRATUM jest źródłem (serwerem) czasu dla następnej. Warstwa STARTUM 16 oznacza urządzenia, których czas nie jest synchronizowany. Im większy numer STARTUM, tym urządzenie jest „dalej” od wzorca czasu. Ma to jednak ograniczone znaczenie, gdyż protokół NTP opiera się na obliczaniu opóźnień między serwerem a klientem. Dzięki takim informacjom klienci czasu mogą skalibrować swój zegar. Zależnie od różnicy, system płynnie nadrabia różnice czasowe – przyspieszając lub opóźniając swój czas. Dla niewielkich odchyleń po prostu ustawia się na nowy NTP. Jeżeli jednak różnica między czasem klienta i serwera ntp (ang. *offset*) będzie zbyt duża (>1000 s), usługa NTP uzna, że dane źródło czasu nie jest wiarygodne i takiej korekty nie wykona. Niedogodność rozwiązuje się poprzez dodanie parametru „-g” przy starcie demona ntp (*ntpd*, plik */etc/default/ntp*), który wymusza synchronizację, nawet jeżeli *offset* jest znaczny.

Dodatkowo NTP potrafi korzystać z wielu źródeł czasu jednocześnie. Dzięki temu można uzyskiwać jeszcze większe dokładności. Synchronizacje (ang. *poll*) mają miejsce co cztery sekundy do siedemnastu minut. Żeby dowiedzieć się, skąd pobieracie czas, możecie wydać komendę:

```
$ ntpq -p
```

Przykładowy wydruk znajdziecie na **ilustracji 4**. Oczywiście poszczególne źródła mogą być różne – dodatkowo zmieniają się dynamicznie, w zależności od dostępności.

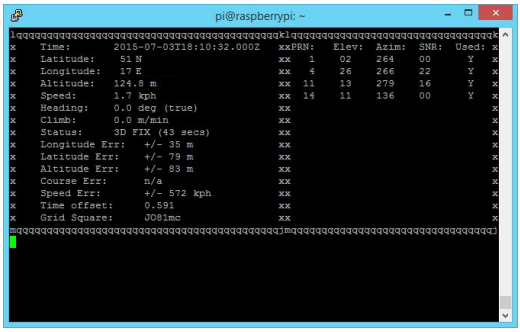
Zwróćcie uwagę na oznaczenia na ilustracji 4 (na podstawie [4]): „*” to aktualne wykorzystywane źródło czasu (tu: 194.29.130.252), a „+” – źródło wykorzystywane w obliczeniach. Kolumna „st” oznacza numer stratum. Pozostałe kolumny zawierają (zob. [5]):

t: typ źródła – *l*: lokalny, *u*: *unicast* (jeden-do-jeden), *m*: *multicast* (jeden-do-wielu), *b*: rozgłoszeniowy (*broadcast*);

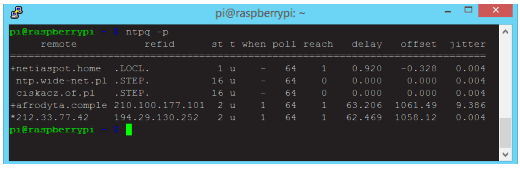
when: czas, który upłynął od ostatniego kontaktu z tym źródłem;

poll: częstotliwości synchronizacji (w sekundach);

delay: czas od zapytania do odpowiedzi z serwera;



3. Wyjście programu cgps



4. Rezultat polecenia ntpq -p

offset: różnica czasu między klientem i serwerem (milisekundy);

jitter: rozrzut wartości między dwoma próbami (milisekundy).

Możecie zignorować kolumnę „remote". Zazwyczaj nie podaje rzeczywistych nazw domen.

PPS

Wracając do GPS: zapewne zauważyliście, że dane wysyłane są przez port szeregowy UART mniej więcej co sekundę. Użyjemy ich, pamiętając, że sam UART nie jest w stanie dostarczać namiarów w dostatecznie regularnych odstępach czasu. Potrzebujemy więc dodatkowego mechanizmu, który będzie precyzyjnie podawał sekundy. System dostosuje do nich dane z GPS i na tej podstawie uzyska bardzo dokładne namiary czasu.

Funkcja generowania impulsów w ścisłych, sekundowych odstępach nazywa się *(One) Pulse Per Second* – PPS. Takie impulsy dostarcza sam odbiornik GPS, jako osobne wyjście. Niestety, w wielu modelach nie wyprowadzono go z czipa. Tak właśnie jest w przypadku mojego modułu.

Modyfikacja okazała się niespecjalnie trudna – wystarczyło w odpowiednie miejsce przylutować dodatkowy kabelek (**ilustracja 5**). Zamiast bezpośrednio do nóżki układu GPS wybrałem styk przed rezystorem ograniczającym prąd dla diody sygnalizującej impuls PPS.

Dzięki tej modyfikacji uzyskałem sygnał PPS (**ilustracja 6**).

Pozostało podłączyć sygnał PPS do GPIO18 Raspberry (fizyczny pin 12), pobrać odpowiednie narzędzia i skonfigurować system w plikach */boot/config.txt* i */etc/modules*:

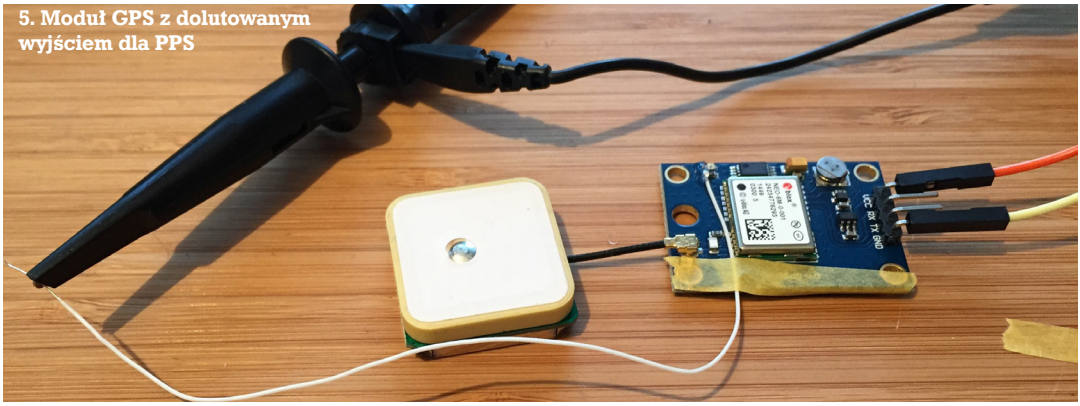
```
$ sudo apt-get install pps-tools
```

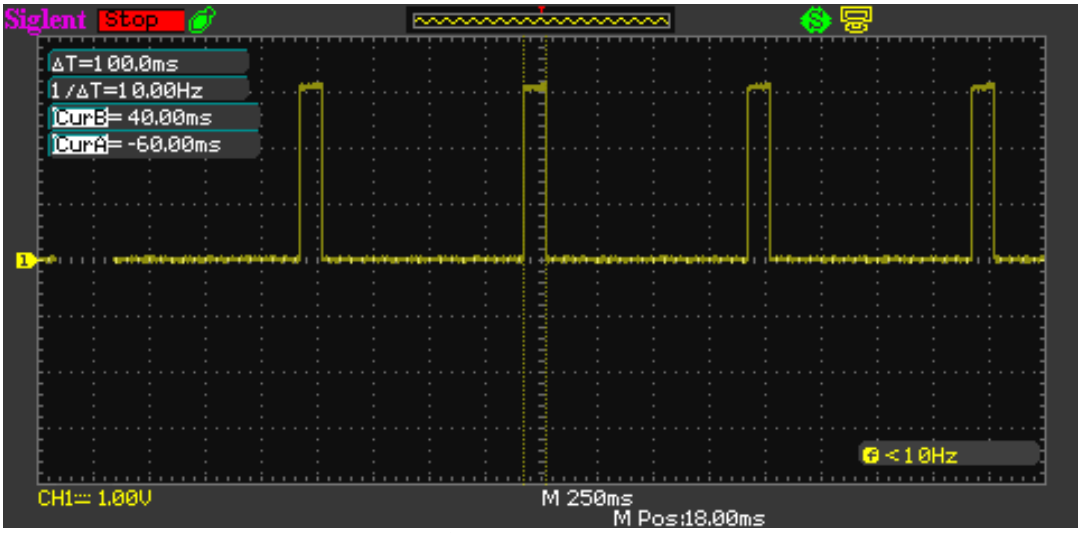
W */boot/config.txt* dodajcie linię *dtoverlay*:

```
$ sudo nano /boot/config.txt
dtoverlay=pps-gpio,gpiopin=18
```

W */etc/modules* dodajcie moduł *pps-gpio*:

```
$ sudo nano /etc/modules
pps-gpio
```





6. Sygnał PPS z GPS śledzony na oscyloskopie

Pomimo że system automatycznie zainstaluje urządzenie */dev/pps0*, zmiany w */boot/config.txt* i */etc/modules* są konieczne. Bez nich, przy próbie wywołania aplikacji *ppstest* zobaczycie:

```
$ sudo ppstest /dev/pps0
trying PPS source "/dev/pps0"
found PPS source "/dev/pps0"
ok, found 1 source(s), now start fetching
data...
time_pps_fetch() error -1 (Connection timed
out)
time_pps_fetch() error -1 (Connection timed
out)
```

Po wykonaniu zmian i restarcie (*$sudo reboot*) możecie sprawdzić, czy PPS jest prawidłowo zainstalowane w systemie:

```
$ dmesg | grep pps
[    4.709208] pps_core: LinuxPPS API ver. 1
registered
…[    4.829103] pps pps0: Registered IRQ 412
as PPS source
[   28.424317] pps_ldisc: PPS line discipline
registered
[   28.428538] pps pps1: new PPS source
ttyAMA0
[   28.428844] pps pps1: source "/dev/tty-
AMA0" added
```

System powinien załadować odpowiednie moduły:

```
$ lsmod | grep pps
pps_ldisc              2285  2
pps_gpio               2897  0
pps_core               8752  3
pps_ldisc,pps_gpio
```

Teraz sprawdzimy, czy działa już *ppstest*:

```
$ sudo ppstest /dev/pps0
trying PPS source "/dev/pps0"
found PPS source "/dev/pps0"
ok, found 1 source(s), now start fetching
data...
source 0 - assert 400.922965675, sequence:
381 - clear  0.000000000, sequence: 0
source 0 - assert 401.922978675, sequence:
382 - clear  0.000000000, sequence: 0
```

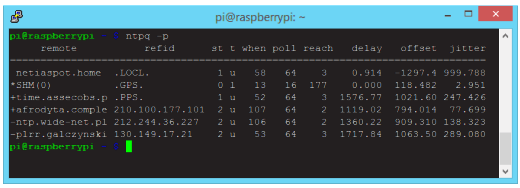
GPS i PPS dla NTP

Niestety, wersja pakietu *ntp* dostarczana z Raspbianem (stan na lipiec 2015 r.) nie wspiera PPS. Trzeba do systemu dodać kilka rzeczy i odświeżyć samo *ntp*. Szczegółowe instrukcje znajdziecie w doskonałym opracowaniu [6] – tutaj zamieszczam je w skrócie.

Najpierw uzupełnimy system:

```
$ sudo apt-get update
$ sudo apt-get -y dist-upgrade
```

Polecenie *apt-get update* jest Wam na pewno znane. Uzupełnia ono listę pakietów, sprawdza zależności między nimi – ale samo nic nie instaluje. Przygotowuje „grunt" pod *apt-get upgrade*. Zamiast *upgrade* użyjemy jednak *dist-upgrade*. Samo *upgrade*



7. Wydruk ntpq -p: jest już GPS, brakuje PPS (porównajcie z ilustracją 4)

uzupełnia wszystkie zainstalowane na Raspberry pakiety do ich najnowszych wersji. Żadne pakiety nie są usuwane, ani żadne nowe dodawane. Jeżeli nowsza wersja jakiegoś pakietu wymaga dodania elementów, które nie były wcześniej zainstalowane w systemie – pakiet ten nie zostanie uwzględniony w zmianach.

Inaczej jest z *dist-upgrade*. Spełnia ona te same funkcje co *upgrade*, ale nie zawaha się dodać/usunąć pakietów tak, aby system miał wszystko co najnowsze w repozytorium. Trzecia z instrukcji tego typu (tu nie użyta), *rpi-update*, koncentruje się na uzupełnieniu jądra systemu i firmware. Firmware i kernel zapisują się na pierwszej partycji karty SD (widocznej pod Windows).

Wykonanie powyższych komend może trochę potrwać. Proces jest automatyczny. Po zakończeniu nie restartujcie jeszcze systemu.

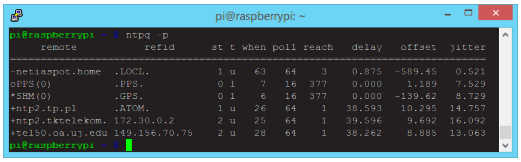
Teraz skonfigurujemy usługę *ntp* i *pps*. Otwórzcie do edycji plik */etc/ntp.conf* (*$ sudo nano /etc/ntp.conf*) i dopiszcie na jego końcu nowe źródła czasu oraz ich parametry (na podstawie [7]):

```
# PPS
server 127.127.22.0 minpoll 4 maxpoll 4
fudge 127.127.22.0 refid PPS
fudge 127.127.22.0 flag3 1
# GPS
server 127.127.28.0 minpoll 4 maxpoll 4 mode
1 prefer
fudge 127.127.28.0 flag1 1 refid GPS
```

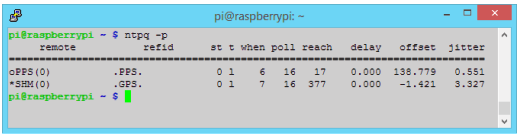
Zauważcie, że (szczegóły [8], [9]):

- usługi („server") GPS/PPS dostępne są pod lokalnym adresem 127.127.x;
- adres 127.127.22.0 odnosi się do usługi PPS a 127.127.28.0 do generycznego GPS;
- „refid" określa nazwę, pod jaką źródła będą identyfikowane w systemie (widoczne na wydruku komendy *ntpq -p*);
- „flag1 1" i „flag3 1" pozwalają jądru i GPS na używanie PPS;
- „mode 1" przetwarza komunikaty NMEA GPRMC;
- „prefer" zaznacza źródło jako preferowane;
- „minpoll/maxpoll" oznacza minimalny/maksymalny odstęp czasu między kolejnymi zapytaniami serwera, w sekundach jako potęgi 2 (tu: 4 – > 2^4 = 16 sekund)

Teraz przerestartujcie Raspberry komendą *$ sudo reboot*. Odczekajcie chwilę po starcie i sprawdźcie:



8. NTP: dodane źródła GPS i PPS (porównajcie z ilustr. 4 i 7)

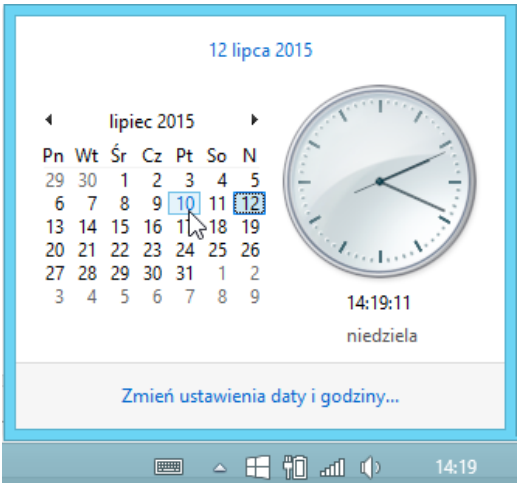


9. NTP: dodane źródła GPS i PPS – konfiguracja bez Sieci

```
$ ntpq -p
```

Powinniście uzyskać efekt podobny jak na **ilustracji 7**. Jeżeli porównacie go z ilustracją 4, zauważycie, że pojawiło się nowe źródło czasu – zadeklarowany przez nas GPS (lokalny startum 0!). Co więcej, został on wybrany jako główne odniesienie (gwiazdka przy wpisie). Niestety, nadal brakuje tam dodanego przez nas źródła PPS (kolumna *refid*). Na przeszkodzie w korzystaniu z PPS stoi jeszcze usługa *ntpd*. Niestety, w wersji rozpowszechnianej z Raspbianem nie zawiera ona wsparcia dla tego źródła. Nie stanowi to jednak problemu: możemy pobrać z Sieci jej inną wersję, przekompilować i zainstalować samodzielnie (uwaga: kroki *./configure* i *make* mogą potrwać nawet pół godziny; na podstawie [6]):

```
$ sudo apt-get install libcap-dev
$ wget http://archive.ntp.org/ntp4/ntp-4.2.8p3.tar.gz
```



10. Zmiana daty w Windows 8

Tabela 1. Zestawienie ważniejszych komend użytych w tekście

Polecenie	Znaczenie
cat /dev/ttyAMA0	Wyświetl zawartość pliku, w tym przypadku pliku urządzenia /dev/ttyAMA0
sudo apt-get install ...	Zainstaluj pakiet, np. *sudo apt-get install gpsd*
sudo service gpsd restart	Zatrzymaj i uruchom ponownie usługę *gpsd*
sudo service gpsd stop	Zatrzymaj usługę *gpsd*
sudo service gpsd start	Uruchom usługę *gpsd*
ntpq -p	Wyświetl źródła czasu wykorzystywane przez usługę *ntp*
cgps -s	Wyświetl aktualne namiary GPS (błędy, satelity)
gpsmon	Wyświetl stan GPS
ls /etc/rc*/*gps*	Wyświetl wszystkie pliki z „gps" w nazwie we wszystkich podkatalogach /etc o nazwie zaczynającej się od „rc"
sudo ppstest /dev/pps0	Uruchamia test PPS dla urządzenia /dev/pps0
dmesg | grep pps	Wyświetl wpisy z logu systemowego (dmesg), które zawierają frazę „pps" (część grep pps)
lsmod | grep pps	Wyświetl załadowane moduły (lsmod), których nazwa zawiera frazę „pps" (część grep pps)
sudo cp /usr/local/bin/ntp* /usr/bin/	Skopiuj wszystkie pliki o nazwie zaczynającej się od „ntp", z katalogu /usr/local/bin/ do katalogu /usr/bin
sudo apt-get update	Uzupełnienie informacji o dostępnych pakietach
sudo apt-get upgrade	Odświeżenie zainstalowanych pakietów (bez dodawania czy usuwania)
sudo apt-get dist-upgrade	Jak *upgrade*, ale dodaje/usuwa pakiety tak, aby system miał wszystko, co najnowsze w repozytorium
sudo rpi-update	Uzupełnienie jądra i *firmware*
sudo date --set='22 August 2002 21:00'	Ustaw czas systemowy na 22 sierpnia 2002 r., godzinę 21:00
sudo halt	Zatrzymaj system
sudo reboot	Zrestartuj system

```
$ tar zxvf ntp-4.2.8p3.tar.gz
$ cd ntp-4.2.8p3
$ ./configure -enable-linuxcaps
$ make
$ sudo service ntp stop
$ sudo make install
$ sudo cp /usr/local/bin/ntp* /usr/bin/ &&
sudo cp /usr/local/sbin/ntp* /usr/sbin/
$ sudo service ntp start
```

Po (dłuższej) chwili (czasami potrzebny jest restart) polecenie *ntpq -p* powinno podać wyniki podobne do tych na **ilustracji 8**. Zauważcie, że obydwa źródła – GPS i PPS zostały dodane, z czego lokalny GPS stał się głównym, a PPS – źródłem uwzględnianym w obliczeniach czasu (oznaczenie „o").

Czas z kosmosu

Wróćmy na chwilę do konfiguracji *off-line* (bez Sieci, jak w poprzednim tekście). Podłączcie Raspberry bezpośrednio do komputera, za pomocą kabla Ethernet. Na karcie SD zmieńcie plik *cmdline.txt* (możecie to zrobić, np. wkładając kartę SD do komputera z Windows). Linia komend powinna wyglądać tak:

```
dwc_otg.lpm_enable=0
ip=169.254.1.1:::255.255.0.0 root=/dev/mm-
cblk0p2 rootfstype=ext4 elevator=deadline
rootwait
```

Przełóżcie kartę z powrotem do Raspberry i uruchomcie go. Po zalogowaniu ustawcie czas na jakiś moment w przeszłości i wyłączcie Raspberry:

```
$ sudo date --set='22 August 2002 21:00'
$ sudo halt
```

Po kilku minutach włączcie go z powrotem. Zauważycie, że czas na Raspberry Pi wkrótce wróci do aktualnego.

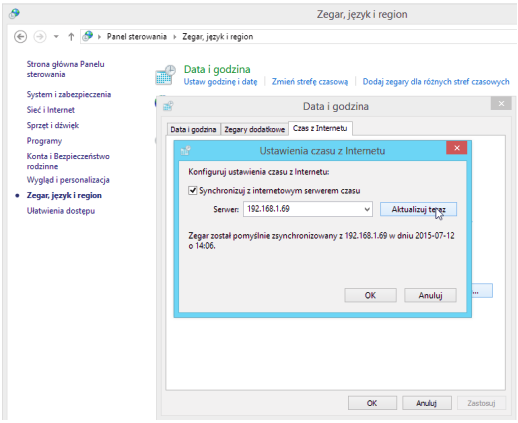
Uwaga! Synchronizacja może zająć nawet kilka minut (zależnie od stanu GPS)! Na **ilustracji 9** widać, że:

- system widzi dwa skonfigurowane przez nas źródła czasu: GPS i PPS;
- GPS wybrano jako główne źródło (oznaczony „*"), a PPS jako uwzględniane („o");
- są to jedyne dostępne źródła – z braku Sieci inne zadeklarowane jako „server" w pliku */etc/ntp.config*, nie są uwzględniane.

Gratulacje, Wasz Raspberry bierze teraz czas bezpośrednio z kosmosu!

Własne Stratum 1?

Wasz Raspberry stał się w tej chwili źródłem czasu STRATUM 1 (0 to sam GPS). Jeżeli podłączycie go z powrotem do sieci domowej, może stać się źródłem czasu dla innych znajdujących się w niej urządzeń. Dla przykładu, zmieńcie czas systemowy na swoim komputerze z systemem operacyjnym Windows 8 – np. o dwa dni w tył. Możecie po prostu kliknąć na datę na pasku startu i wybrać jeden



11. Zmiana ustawień synchronizacji w Windows 8

z poprzednich dni tygodnia (musicie być w grupie administratorów, **ilustracja 10**).

Teraz otwórzcie panel sterowania i wybierzcie element „Zegar, język i region", a następnie: „Data i godzina". W otwartym oknie, na zakładce „Czas z Internetu" kliknijcie „Zmień ustawienia". W polu „Synchronizuj z internetowym serwerem czasu" podajcie adres IP Waszej Raspberry. Żeby się dowiedzieć, jaki jest adres RPi, wydajcie w jego konsoli polecenie *$ ifconfig*. Następnie na Windows kliknijcie przycisk „Aktualizuj teraz". Oczekujcie komunikatu „Zegar pomyślnie zsynchronizowany..." – czasami klikanie trzeba powtórzyć kilka razy (**ilustracja 11**). Teraz sprawdźcie czas systemowy. Powinien być z powrotem aktualny, a przynajmniej identyczny z tym na Raspberry.

Podsumowanie

W tym odcinku kontynuowaliśmy omawianie zagadnień związanych z odmierzaniem czasu. Tym razem wykorzystaliśmy do tego moduł GPS. Wiązało się to z koniecznością doinstalowania kilku narzędzi i przekonfigurowania systemu. Nie jest to więc funkcjonalność, która działa od ręki (ang. *out of the box*). Ale przy okazji nauczyliśmy się kilku nowych sztuczek, które na pewno przydadzą się Wam w kolejnych projektach.

Źródła

[1] Raspberry Pi: łączenie logiki, MT 7/2015, http://goo.gl/Wj7nIS
[2] http://aprs.gids.nl/nmea/
[3] http://www.catb.org/gpsd/gpsd.html
[4] http://goo.gl/7MCtmg
[5] http://goo.gl/G6X7qa
[6] http://goo.gl/aO6p9J
[7] http://ntpi.openchaos.org/pps_pi/
[8] https://goo.gl/TQ73OY
[9] http://doc.ntp.org/4.1.1/confopt.htm