Pirometr tęczowy

Urządzenie mierzy temperaturę pirometrem. Dodatkowo odczytuje temperaturę z maksymalnie siedmiu układów DS18B20. Wynik jest wyświetlany na trzycyfrowym wyświetlaczu 7-segmentowym. Ponadto obrazowany jest na 15 diodach WS2812. Wynik może być odczytywany przez USB. Wyniki pomiaru są dostępne przez stronę www, ponadto są zapisywane na PenDrive. Pirometr może odczytywać temperaturę ze zdalnych termometrów przez W-Fi.

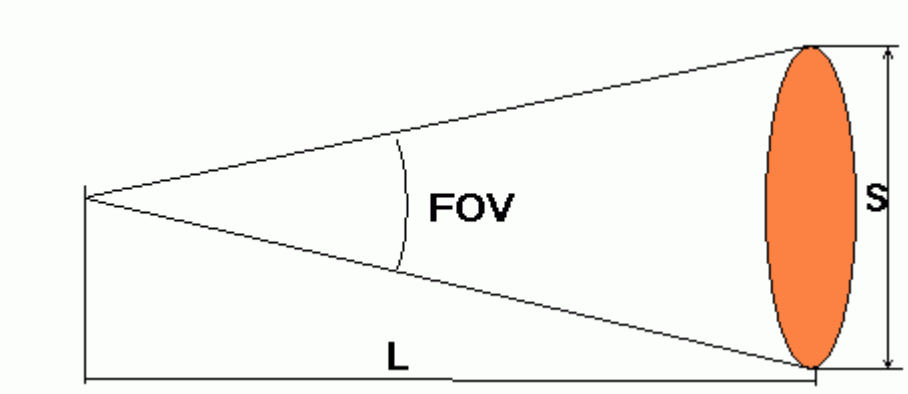
Parametry termometru:

- zasilanie z USB lub zasilacza 7..15V
- odczyt z pirometru w zakresie -70(-40)^{O}C ...380(125)^{O}C
- odczyt z DS18B20 w zakresie -50^{O}C...+125^{O}C
- odczyt danych z termometrów zdalnych
- prezentacja wyników na 3 wyświetlaczach LED 7-segmentów i 15 diodach WS2812 (zastosowano korektę Gamma)
- obsługa termometru pilotem IR (standardy od popularnych RC5, RC6, Samsung, Samsung32, NEC po egzotyczne Merlin, Pentax)
- regulacja jasności LED, ustawienia zapisywane w EEPROM
- zapis pomiarów na PenDrive
- port drukarki, klawiatury, mostków USB i układów muzycznych firmy FTDI *
- komunikacja przez Wi-Fi
- synchronizacja czasu z NTP
- zmiana obsługiwanego termometru przyciskiem
- nadajnik IR *

* opcja nieprogramowana

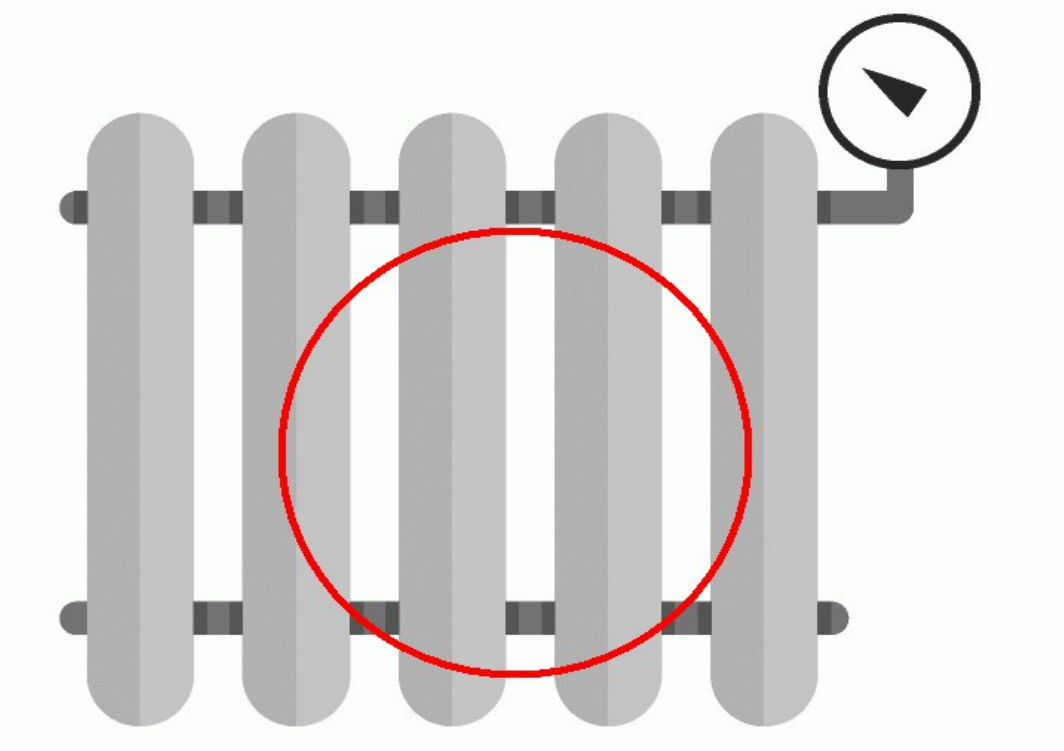
Zasada działania pirometru:

Na wstępie napisze parę słów o zastosowanym pirometrze. MLX90614 występuje w kilku wersjach. Najistotniejsze to zakres mierzonych temperatur -70..125˚C lub -40...80 ˚C, napięcie zasilania 3 lub 5V, kąt pola widzenia (FOV) 90°, 35°, 12°, 10° lub 5°. Pirometr komunikuje się z mikrokontrolerem magistralą SMBus w tym przypadku kompatybilną z I^{2}C dzięki czemu, do komunikacji, można wykorzystać TWI. Zmieniając konfigurację pamięci EEPROM pirometru, można ustawić generowanie sygnału PWM zależnie od zmierzonej temperatury.
FOV definiuje nam z jakiej maksymalnej odległości możemy zmierzyć temperaturę obiektu o określonym rozmiarze. (Plik „Pirometr FOV.gif”)



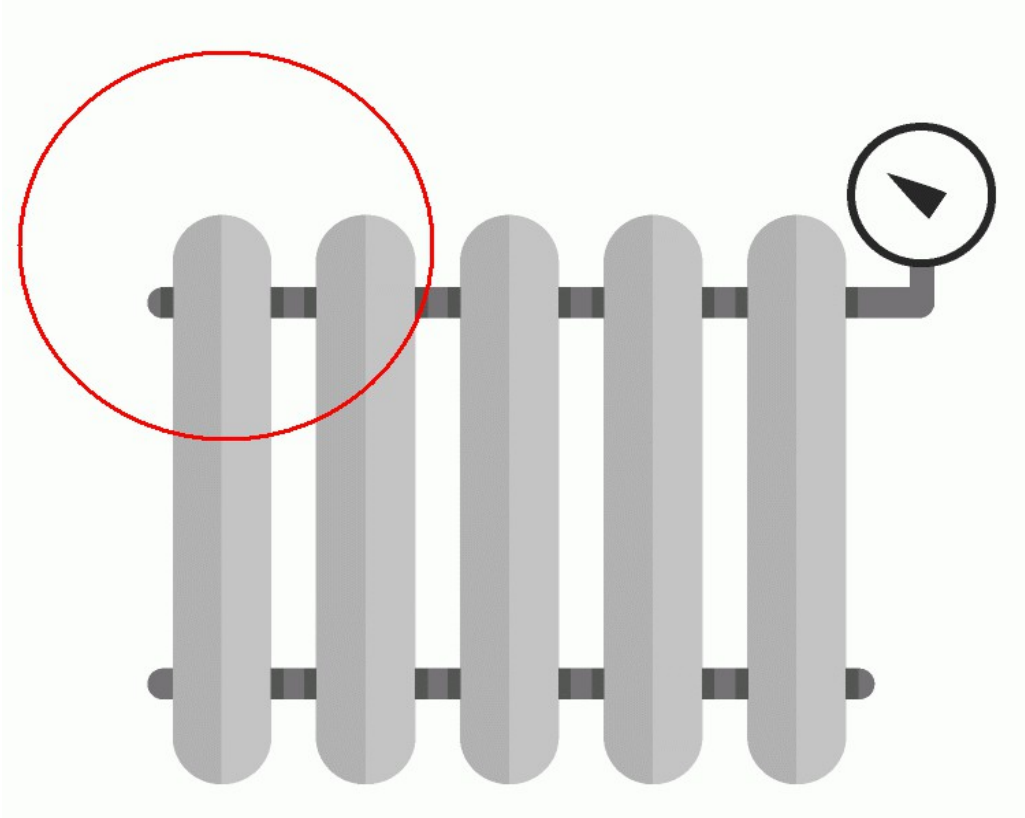
Czujnik przetwarza średnią wartość odbieranego promieniowania obiektu wyznaczoną przez okrąg

o średnicy S. Ważne jest więc aby okrąg znajdował się wewnątrz obiektu, którego temperaturę chcemy mierzyć. (Plik „Kaloryfer_Dobrze.GIF”)



(obrazki ze strony https://pl.freepik.com/darmowe-wektory/ogrzewanie-i-chłodzenie)

Jeśli pirometr widzi także inne obiekty, np. tło (Plik „Kaloryfer_Zle.GIF”)

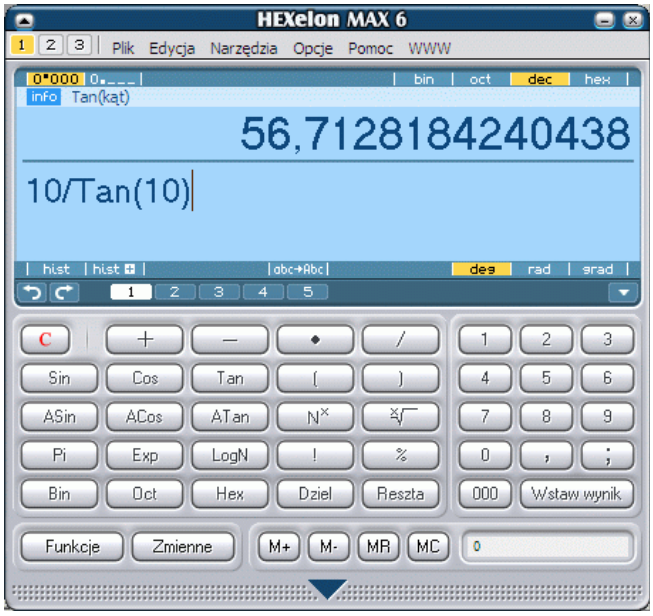


wynik pomiaru będzie średnią temperatur obiektu i tła.

W zależności od wersji czujnika MLX90614, kąty pola widzenia FOV wynoszą odpowiednio: 90°, 35°, 12°, 10° i 5°. Im mniejszy jest ten kąt, tym dalej może znajdować się czujnik od badanego obiektu. Dla kątów mniejszych niż 45° występują zależności:
S = tan(FOV) * L
i
L = S / tan(FOV)

Przykładowo, aby zbadać temperaturę obiektu oddalonego od czujnika o 1metr dla czujnika o wartości FOV=10°:
S = tan(FOV) * L (cm)
S = tan(FOV) * 100 cm
S = 0.1763269807 * 100 cm
S = 17.63269807 cm
(Plik „Pirometr S=.gif")

Obliczmy, z jakiej maksymalnej odległości możemy zbadać temperaturę obiektu o rozmiarze 10 cm
L = S / tan(FOV)
L = 10 cm / tan(10°)
L = 10 cm / 0.1763269807
L = 56,7128184240438 cm
(Plik „Pirometr L=.gif")



W pirometrze MLX90614ESF-AAA użytym w prototypie kąt widzenia wynosi 90°. Dla kątów większych niż 45° do obliczeń przyjmujemy jego połowę, a otrzymany wynik mnożymy przez 2.
S / 2 = tan(FOV / 2) * L
Dla 90° obliczenia można uprościć, ponieważ tan(90°/2) = 1, dlatego:
S=L*2

Z odległości 10cm można mierzyć temperaturę obiektu o średnicy co najmniej 20cm. O pomiarze temperatury odległych obiektów, można więc zapomnieć ale do szybkiego pomiaru temperatury radiatora taki pirometr będzie wystarczający.
Na dokładność pomiaru temperatury wpływ ma również współczynnik emisyjności. Materiały mają różną zdolność emitowania promieniowania podczerwonego. Zależy to głównie od barwy powierzchni i jej gładkości. Ciemne szorstkie obiekty mają większą zdolność emisyjną niż materiały o barwie jasnej i gładkiej. Ludzka skóra posiada współczynnik emisyjności około 0.9, wypolerowane aluminium 0.05. W MLX90614 współczynnik emisyjności jest ustawiony na 1. Można go zmienić modyfikując zawartość pamięci EEPROM. W EEPROM zapisany jest także adres czujnika, można go więc zmienić. Teoretycznie można podłączyć 126 czujników(adres 0 i 255 jest zarezerwowany), ale 2 adresy zajmuje mostek USB.

Budowa:

Budowa sprzętowa nie jest zbyt skomplikowana. Zasilacz składa się z mostka i stabilizatora 5V. Te elementy można pominąć, jeśli termometr będzie zasilany z USB. Stabilizator 3V3 jest konieczny, jeśli będziemy używać zapisu na PenDrive, jeśli nie potrzebujemy tej opcji, elementy związane z stabilizatorem można pominąć. Katody wyświetlacza 3x7-segmentów są sterowane multipleksowo bezpośrednio z wyprowadzeń mikrokontrolera, anody za pośrednictwem tranzystorów T1..T3. Pirometr przyłączamy do magistrali IIC używając złącza J7. Do J8 można podłączyć drugi pirometr w trybie PWM. Konieczne jest jednak wcześniejsze ustawienie pirometru w ten tryb. Wydaj się, że skoro i tak trzeba zmienić zawartość eeprom, to prościej kolejnym termometrom nadać inne adresy. Tej opcji nie przewidziano w oprogramowaniu. W razie zainteresowania taką opcją proszę o e-maile. Magistrala 1-Wire jest zabezpieczona układem U4. Obsługiwanych będzie maksymalnie 7 termometrów DS18B20. U6 odbiera sygnał z pilota podczerwieni. Dioda nadawcza D6 nie jest aktualnie używana. Włączono ja, na wyjście timera2 OC2A. Aktualnie Timer2 używany do odbioru IR ale można łatwo zmienić go na Timer3 jeśli nadawanie IR byłoby potrzebne. Buzzer BZ1 nie jest używany. Można go wykorzystać do sygnalizowania przekroczenia zadanego zakresu temperatur. Układ U2 neguje sygnał z USART aby

sterować diodami WS2812. Dzięki wykorzystaniu USART-a, aktywne mogą być przerwania o „niższym priorytecie”. W przeciwieństwie do ARM, AVR nie ma priorytetowego systemu przerwań. Do obsługi WS2812 wymagane jest, aby USART miał najwyższy priorytet. Uzyskano to sztucznie, podobnie jak w Z-80 z systemem wielopoziomowym, przez wykonanie rozkazu SEI zaraz po wejściu w przerwanie dla przerwań o „niskim poziomie”. Uzyskuje się to przez zadeklarowanie przerwania jako INTERRUPT lub ISR z atrybutem ISR_NO_BLOCK. Przerwanie od USART dla WS2812 musi być przerwaniem SIGNAL lub ISR bez atrybutów. Więcej szczegółów w kodach źródłowych. Za komunikacje z PenDrive odpowiada U7. Trzeba do niego wgrać program używając złącza J11. Aktualnie oprogramowanie mikrokontrolera nie obsługuje PenDrive. Kartę Wi-Fi ESP8266 łączy się z termometrem używając dodatkowej płytki z konwerterem IIC/SPI-USART przyłączonej do J14. Na płytce tej znajduje się układ SC16IS760IPW. Od strony rejestrów jest kompatybilny z popularnym USART-em 16C550. Jego zaletą jest 64 bajtowe FIFO. Ponadto, w przeciwieństwie do 16C550, można odczytać zajętość FIFO, zarówno nadawczego jak i odbiorczego co umożliwia transfer blokowy (adres rejestru wysyłamy tylko raz). Posiada też inne, nieprzydatne w tym projekcie opcje, jak sprzętowe sterowanie przepływem czy linią DIR dla RS485 oraz 4..12 linii GPIO, które mogą zgłaszać przerwania. Aktualnie program zajmuje ponad 42kB, zużycie RAM ponad 2,7kB. Wymagany jest więc procesor Mega664, ale zalecałbym Mega1284. Na co zużyto tak dużo ram? Ze względu na pseudomultitasking, bufory nadawcze i odbiorcze dla NVC-2 i ESP2866 są od siebie niezależne. Ramka CSV to data/godzina 18 bajtów, każdy termometr 9, +CRLF, razem max 101 bajtów. Ramka dla strony WWW: nagłówek-70bajtów, 1 termometr (z nr seryjnym Dallasa)-25, razem 263 bajty. Do tego bufory odbiorcze po 512 dla ESP i 128 bajtów dla VNC-2, bufor dla WS2812 120 bajtów (8 na diodę). Bufor na ID termometrów to kolejne 56bajtów. Nazwa sieci Wifi i hasło to 50 bajtów (32-nazwa, 16-hasło + 2 razy kod końca teksu).Tak ziarnko, do ziarnka i uzbierało się spora miarka. Kompilując program, można by zaoszczędzić pamięć ram (np. hasło/nazwę sieci Wi-Fi odczytywać bezpośredni z eeprom), ale skoro program zajmuje ponad 32k, to dysponujemy 4kB RAM. Nie widziałem więc sensu oszczędzać. W programie kluczowe bufory są kontrolowane i w przypadku ich przepełnienia sygnalizowany jest błąd.
Kilka uwag dotyczących programu. Wielokrotnie spotkałem się z programami, gdzie odczyt temperatury z termometrów 1-Wire polegał na analizie pierwszych dwóch bajtów, bez sprawdzania CRC. W konsekwencji, zwarcie magistrali do masy, dawało odczyt 0 stopni. W termometrze, ze względu na możliwość obsługi kilku układów, używam komendy MATCH ROM, w której zawarte jest CRC. Dzięki temu można wykryć nieprawidłowości. Ponadto czytane jest 9 bajtów danych z układu, i analizowane CRC. Dzięki temu błędy są wykrywane. Można to zaobserwować odłączając układ w czasie pracy termometru (pojawi się błąd). Tak jak w przypadku 1-Wire, CRC tak w przypadku IIC, sprawdzany jest bajt PEC pirometru. Dzięki temu można wykryć błędy transmisji. Jest to konieczne, ponieważ czujnik jest połączony z płytka przewodami. Brak masy może powodować czasem "poprane" odczyty IIC ale nigdy nie da poprawnego PEC (CRC). Poniżej obliczanie PEC dla MLX90614:

```
crc = _crc8_ccitt_update( 0, MLX90614_ADR );    // Adres WR
crc = _crc8_ccitt_update( crc, adr );           // CMD
crc = _crc8_ccitt_update( crc, MLX90614_ADR+1 );// Adres RD
crc = _crc8_ccitt_update( crc, temp & 0Xff );   // Low
crc = _crc8_ccitt_update( crc, temp >> 8 );     // High
```

Jak sterować USARTem AVT-a diodami WS2812? Użyłem znanego rozwiązania używającego transmisji 7-bit. Wymagane jest zanegowanie sygnału z USART. Jak pojawia się trudność? Aby sterować diodami, USART musi pracować z prędkością 2..2,5Mb/s. Wydawało by się, że jest ona nieosiągalna dla AVR. Jest to możliwe, trzeba tylko wpisać do UBRR wartość 0. Minimalna częstotliwość taktowania CPU, wymagana do sterowania diod to 18MHz, przy 16 pojawiają się

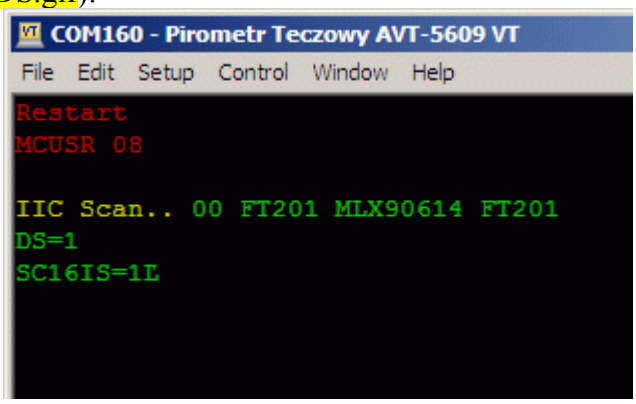
zakłócenia przy większej ilości diod. Problemu tego nie będzie z diodami WS2813, których czasy są bardziej restrykcyjne. W termometrze użyłem kwarcu 18,4321MHz, zapewnia on poprawne sterowanie diod, a jednocześnie jest to częstotliwość „UART'owa". W projekcie wystarcza procesor AtMega664, jednak w razie rozbudowy aplikacji potrzebny będzie AtMega1284. Nie wynika to z powodu braku pamięci flash, problemem staje się pamięć ram. Aplikacje internetowe wymagają jej dość dużo. W tym projekcie 64 bajtowe FIFO w UART nie zawsze rozwiązuje problem i trzeba stosować różne sztuczki i ograniczenia, o których można przeczytać w kodach źródłowych. Wspomniałem o pseudomultuitaskingu. Oprogramowanie oparte jest o zdarzenia. Jedyne „delay'e" to 50us gwarantujące sygnał reset dla diod WS2812 i 800ms potrzebne do wygenerowania softwarowego resetu przy u życiu WatchDog'a. Wszystko więc dzieje się równolegle. W czasie zapisu na dysk, mogą być odbierane/nadawane dane przez Wi-Fi, czy odbierane rozkazy z pilota IR. Odstępstwem od multitaskingu jest czekanie na zakończenie transmisji nadawczych (przez USB, Wi-Fi, VNC-2). Można by te operacje przenieść na przerwania, ale zwiększy to zapotrzebowanie na pamięć RAM co postawiło by pod wątpliwość użycie AtMega664 i konieczne byłoby montowanie Mega1284.
Oprogramowanie jest przygotowane do odbioru danych z innych termometrów przez sieć Wi-Fi. Dane te są wysyłane po porcie 80, tak jak zapytania HTTP, z tym, ze używa się metody PUT a nie GET. Przykładowe ramki danych:
PUT / TXT Ta 127.34
PUT / TXT Tb -33.22
Jeśli ciąg znaków będzie zakończony kodem CR to ramka zostanie zwrócona z pominięciem ciągu „PUT / TXT „. Więcej o tej funkcjonalności będzie napisane w dalszej części artykułu.

Uruchomienie:

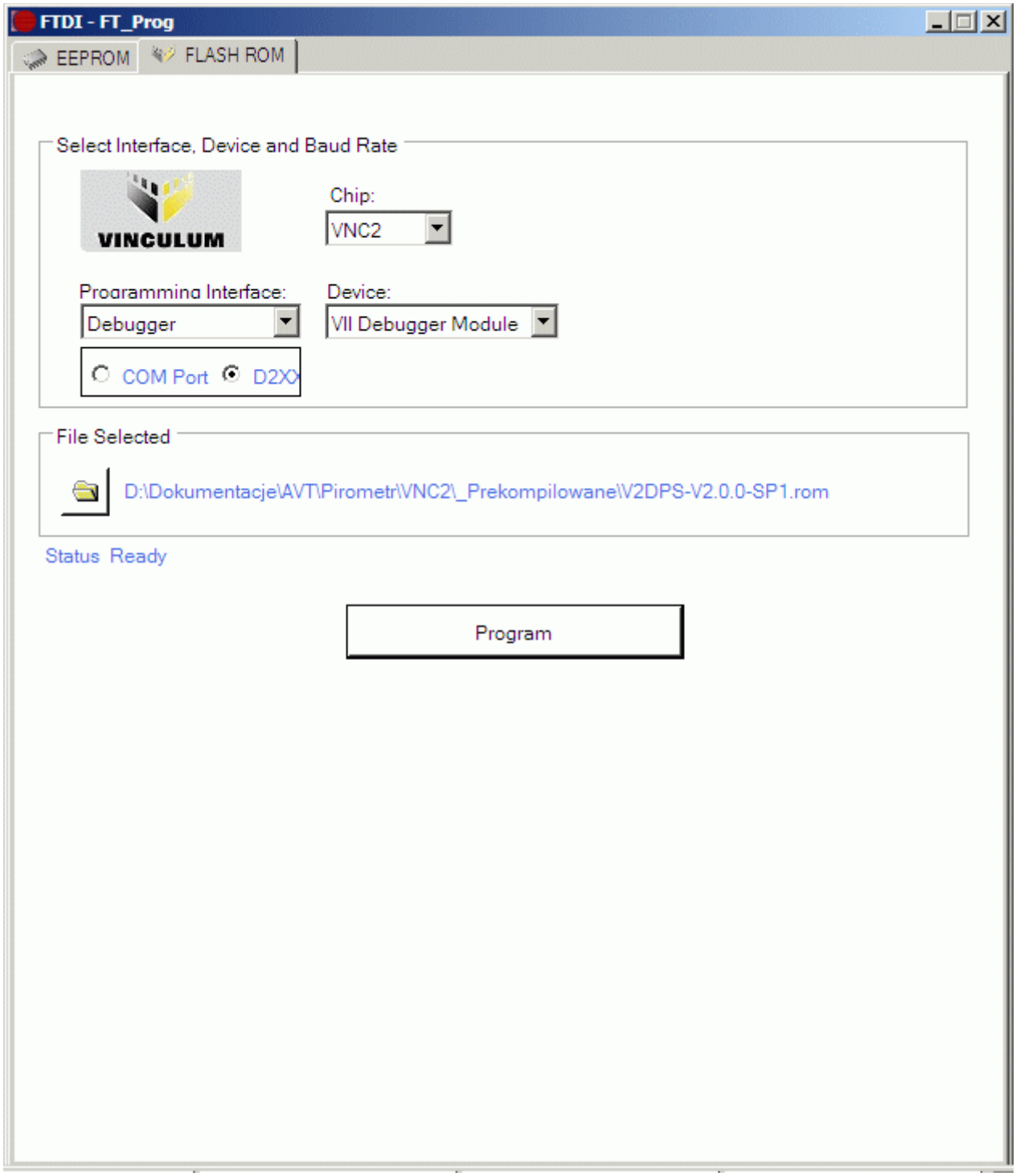
Tradycyjnie rozpoczynamy od zasilacza. Termometr można zasilić z USB i wtedy problem zasilacza nie istnieje. J17 ma być rozwarta. Pirometr podłączamy do złącza J7.

Uruchomienie urządzenia może być kłopotliwe gdy będzie się chciało wykorzystać zapis na PenDrive (konieczność posiadania DebugModule). W przypadku Wi-Fi, może trzeba będzie zmienić prędkość transmisji.

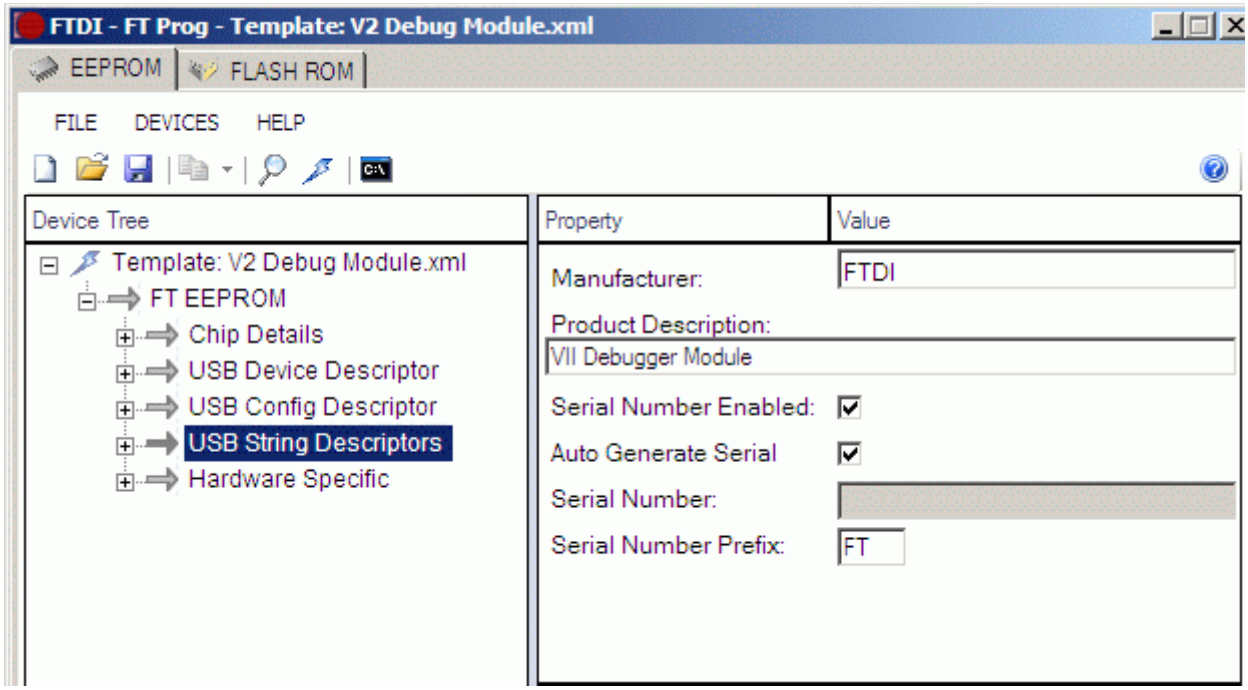
Jeśli termometr podłączony jest do komputera, w terminalu zobaczymy informację o resecie, jego źródle, wyniku skanowania magistrali IIC oraz liczbie znalezionych termometrów DS18B20 (Ekran Restart, scan IIC DS.gif):



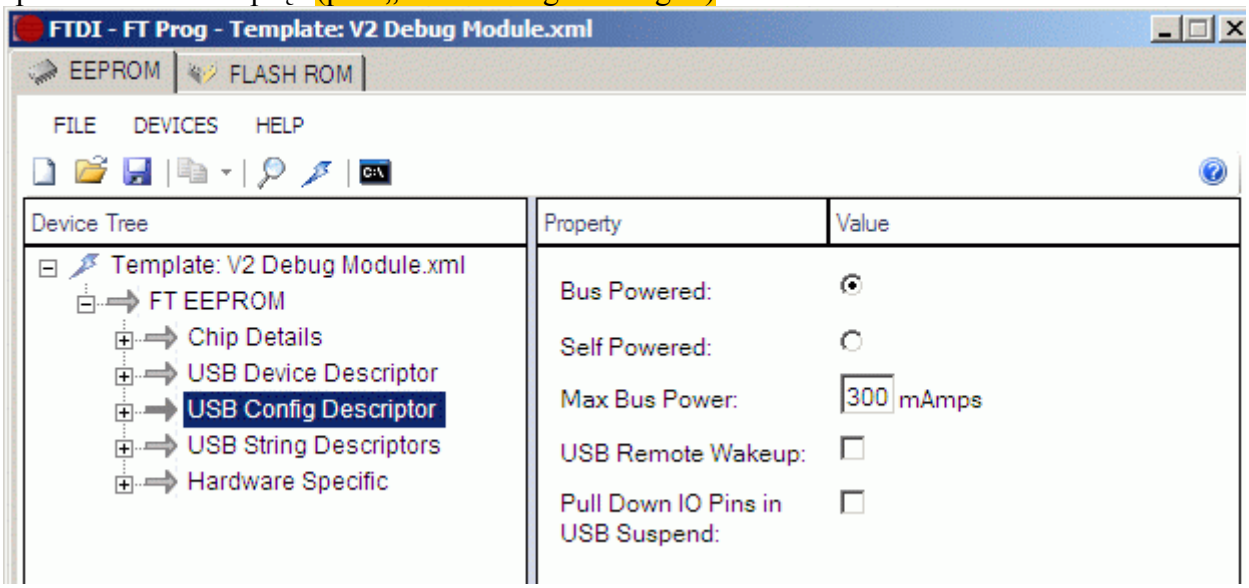
Jeśli używamy karty Wi-Fi należy ja ustawić na prędkość 115200 komendą „AT+CIOBAUD". Starsze wersje kart ESP2866 komunikowały się standardowo na 9600, nowsze na 115200.
Jeśli wykorzystamy zapis na PenDrive, to VNC trzeba wgrać program „V2DAP-115200 bez FLOW CTRL.rom". Najprościej zrobić to FT_PROG'em. (plik „VNC Wgranie softu.gif")



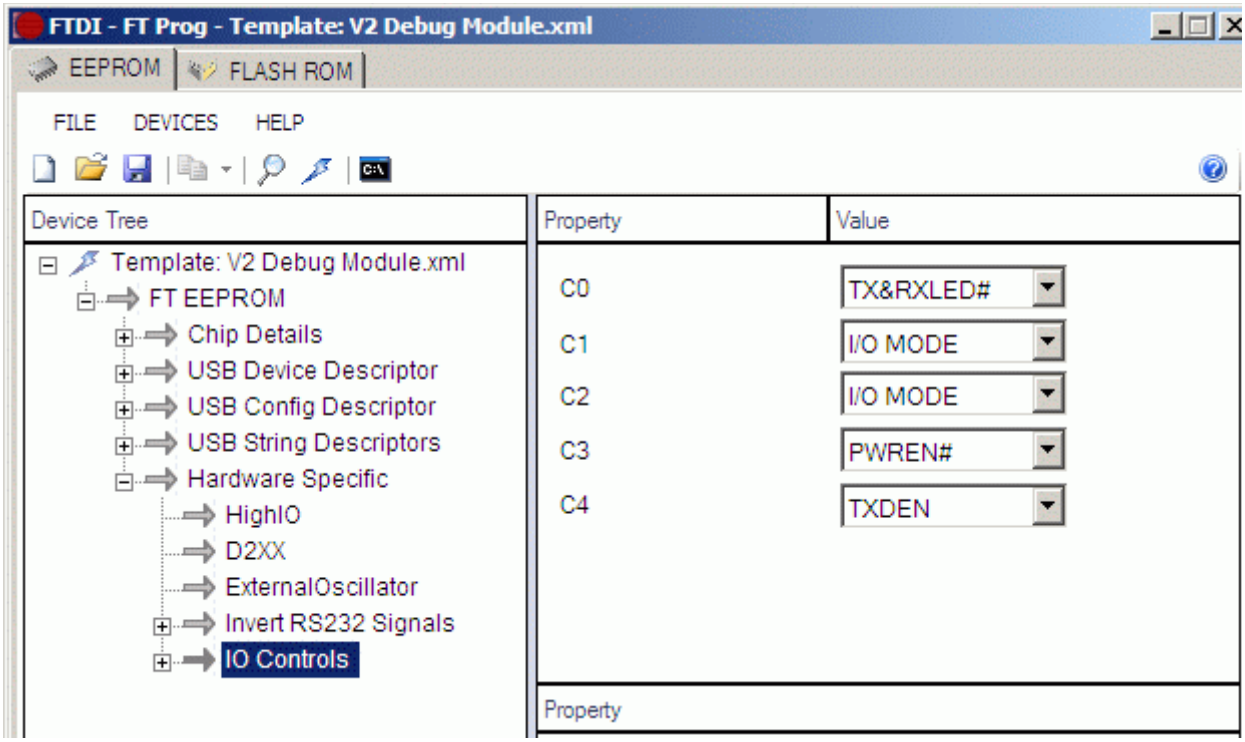
Programator DebugModule jest dość drogi. Można go jednak zbudować samemu z układu FT232RL i bufora 3-stanowego. Poniżej schemat i rysunki płytek. (schemat w pliku projektu Protela99) Układ FT232RL zależy skonfigurować plikiem „V2 Debug Module.xml". Z istotnych ustawień ważne są: nazwa modułu (plik „VNC Debug Name.gif")



Zapotrzebowanie na prąd (plik „VNC Debug Power.gif”)

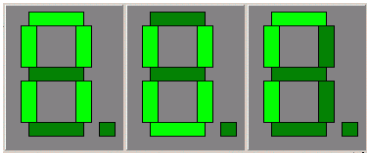


i najważniejsze, funkcje pinów GPIO (plik „VNC Debug IO.gif”)

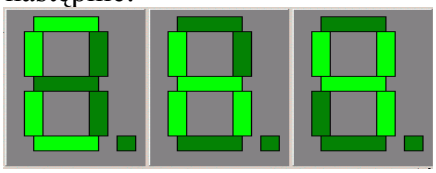


Obsługa:

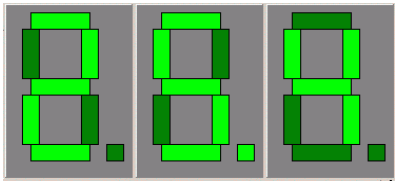
Po resecie przeprowadzony jest test LED, na wyświetlaczy 7-seg zobaczymy:
(pliki rysunków wyświetlaczy maja nazwy zaczynające się od „7_”)



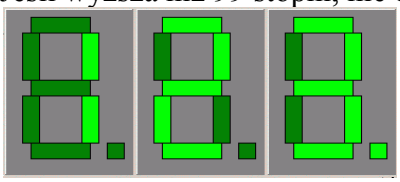
następnie:



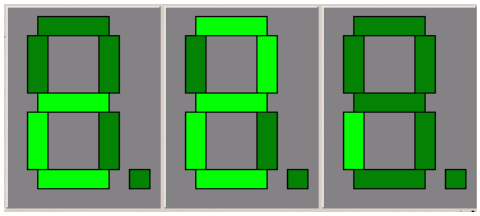
gdzie 4 – liczba znalezionych termometrów. po chwili pokaże się temperatura, np.



Jeśli wyższa niż 99 stopni, nie będą wyświetlane części dziesiętne:



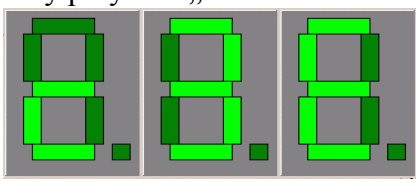
Przyciskiem można zmienić nr termometru. Po jego naciśnięciu na wyświetlaczu pokaz się np:



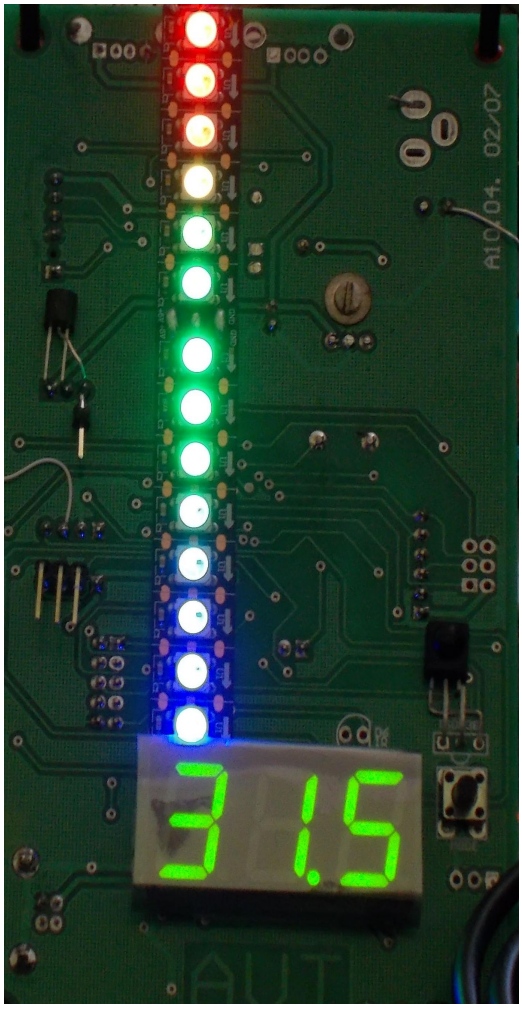
„c2i” gdzie 2 – nr termometru, i – zakres wyświetlanych temperatur:

E	external:	-15..0
i	internal:	8..30
C	CO (kaloryfer):	40..60
L	low:	25..150
h	high:	150..250
H	very high:	200..500

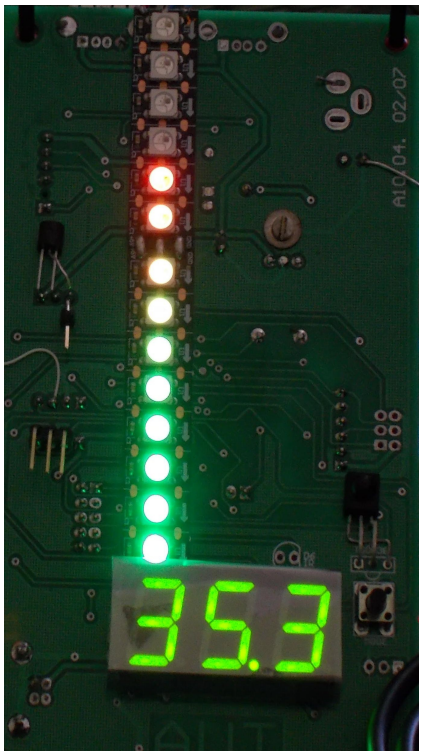
Inny przykład „c3E”:



Zakres wyświetlanych temperatur, określa minimalną i maksymalną temperaturę sygnalizowaną przez diody od niebieskiej, przez zieloną, żółtą, pomarańczową, do czerwonej. Wygląd skali Interal przy 31 stopniach Celcjusza (plik „Led 31 i.jpg”)



Exteral przy 35 stopniach Celcjusza (plik „Led 35 E.jpg”)



Temperatury ujemne sygnalizują barny od niebieskiej (0 stopni) do białej (-15).
(plik „Led -15.jpg”)

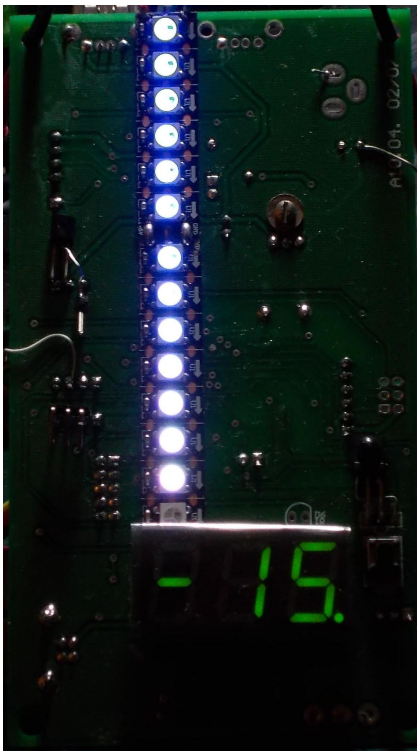


Foto proponuje zrobic w redakcji, ja nie bardzo mam warunki Wydając komendę „:Txx” można wy świetlić dowolna temperaturę. Można także wysłać ramkę „PUT / TXT” po procie 80 (przykłady dalej)

Temperatura poniżej -90 stopni oznacza błąd termometru, wynik na wyświetlaczu miga, wykaz błędów:

-91..-99	Błąd IIC

-90 błąd startu
-91 błąd adresowania
-92 błąd zapisu komendy/adresu
-93 błąd adresowania do odczytu
-94 błąd odczytu bajtu LOW
-94 błąd odczytu bajtu HIGH
-94 błąd odczytu bajtu PEC

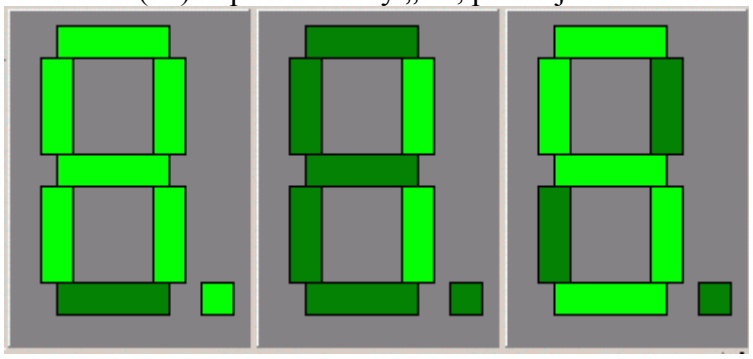
E-1 temperatura DS18B20 niższa niż -99 stopni (uszkodzony czujnik temperatury)
E-2 temperatura DS18B20 wyższa niż 999 stopni (uszkodzony czujnik temperatury)
E-3 przepełniony bufor „str[]" lub „csv[]" (błąd krytyczny, program może się zawiesić)
E-4 przepełniony bufor „www_in[]" lub „www_out[]" (błąd krytyczny, program może się zawiesić)

Obsługa pilotem IR:

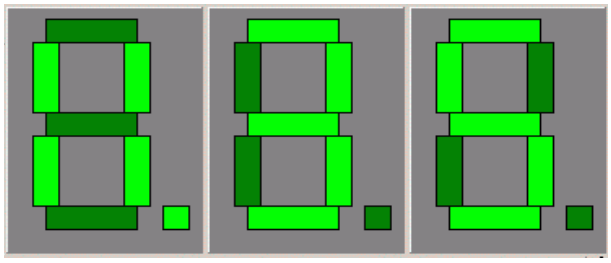
Domyślnie, w eeprom ustawiony jest standard RC5 (TV1), przypisanie kodów jest następujące:

1..9	wybór termometru: 1-Pirometr obiekt, 2-pirometr tło, 3.. - kolejne DS18B20
Ch+/Ch-	wybór termometru (następny, poprzedni)
Vol+/Vol-	regulacja jasności LED
TXT (czerwony)	Zmiana zakresu wyświetlanych temperatur
ON/Off	reset urządzenia
0	ostatni bajt adresu IP
zielony	wyświetlenie czasu.

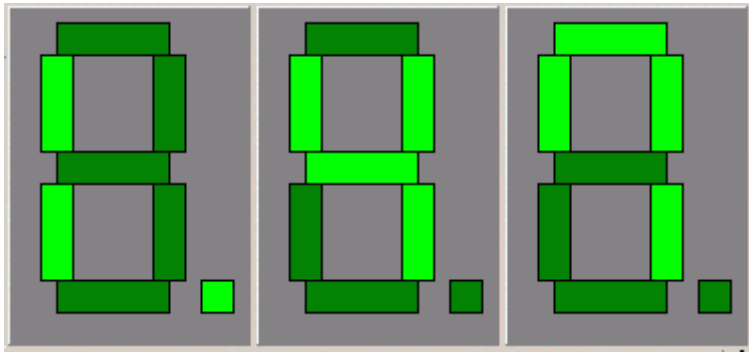
Czas jest wyświetlany w formacie 12-godzinnym. Godzinny 1..9 wyświetlane są standardowo, godz 12 i północ jako 0. Godzina 10 (22) w postaci litery „A", poniżej 10:15.



11 i 23 jako dwie pionowe kreski. Na rysunku poniżej godzina 11:35



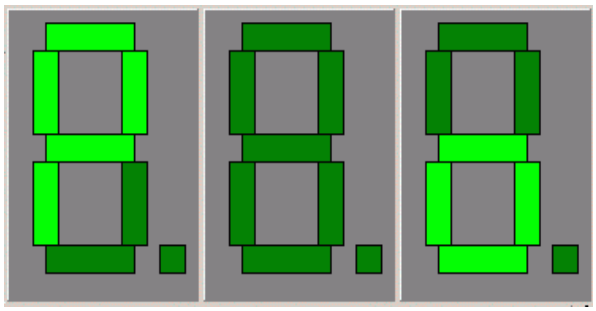
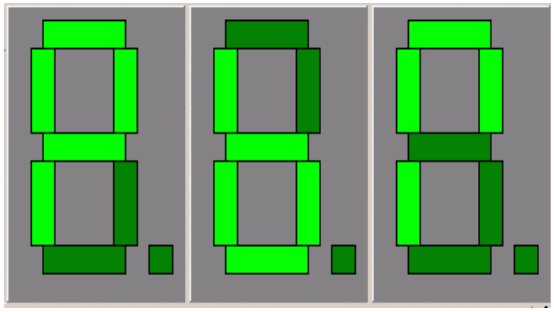
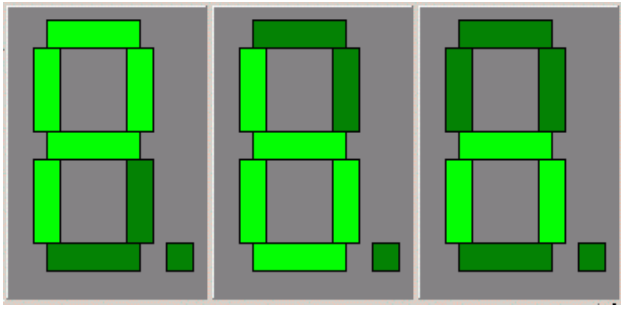
Dla lepszej czytelności, godzina 1 i 13 przedstawiana jest nietypowo:

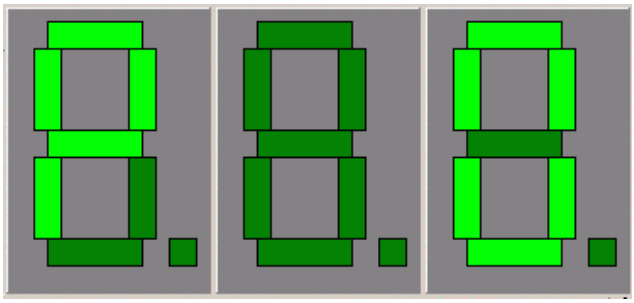
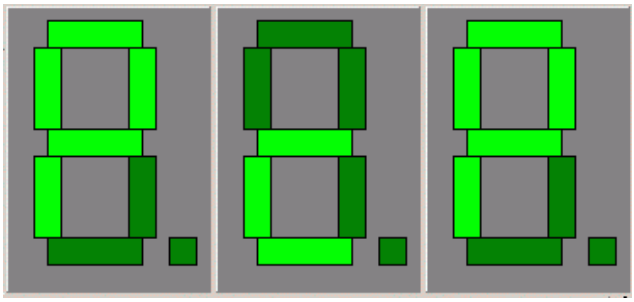
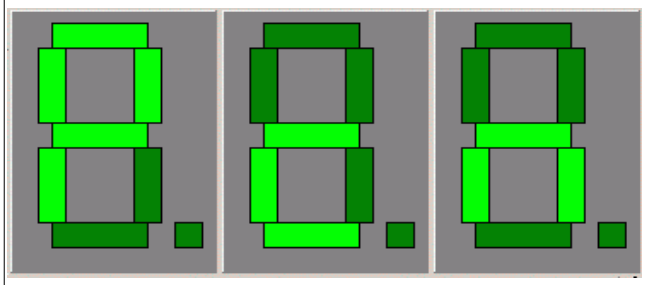
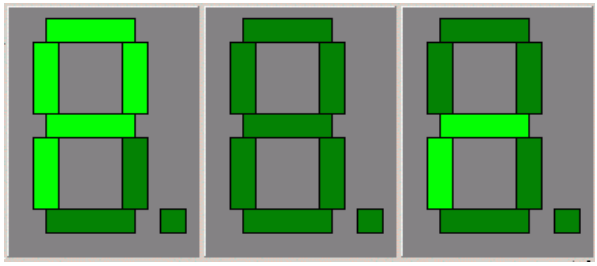
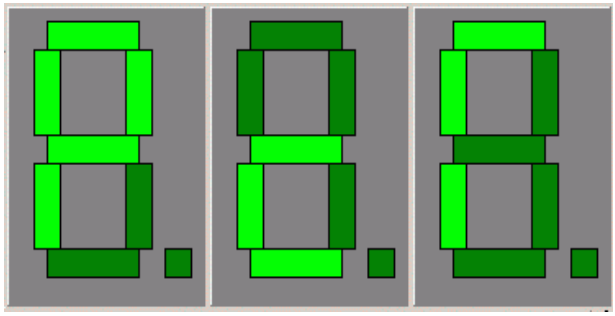


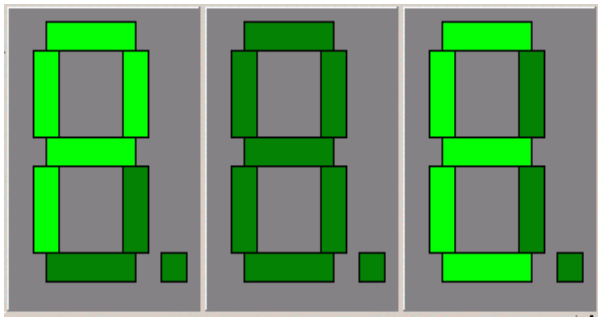
Jeśli czas nie będzie zsynchronizowany z NTP (brak synchronizacji przez ponad godzinę) to wyświetlacza będzie migał.

Podczas odbioru kodów z pilota, kropka na wyświetlaczu 7-seg miga. Kody pilota i format transmisji można poznać używając terminala komputerowego.

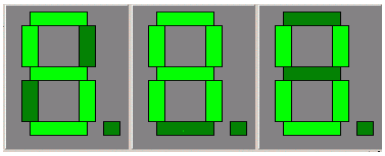
Do termometru można przypisać dowolne kody, praktycznie dowolnego pilota. Aby wejść w tryb uczenia, należy w terminalu wydać komendę „:I” lub przytrzymać przez 5 sekund przycisk. Na wyświetlaczu pojawi się:

	oczekując na naciśnięcie na pilocie przycisku wywołującego reset termometru, następnie:
	oczekując na naciśnięcie przycisku zwiększania jasności LED
	oczekując na naciśnięcie przycisku zmniejszania jasności

	oczekując na naciśnięcie przycisku „0”
	oczekując na naciśnięcie przycisku zwiększania nr termometru
	oczekując na naciśnięcie przycisku zmniejszania nr termometru
	oczekując na naciśnięcie przycisku zmiany typu termometru (zewnętrzny, wewnętrzny, temperatury niskie, wysokie)
	Oczekiwanie na przycisk wyświetlania czasu

	informując o końcu trybu uczenia się

Po zakończeniu nauki, na chwilę pojawi się napis

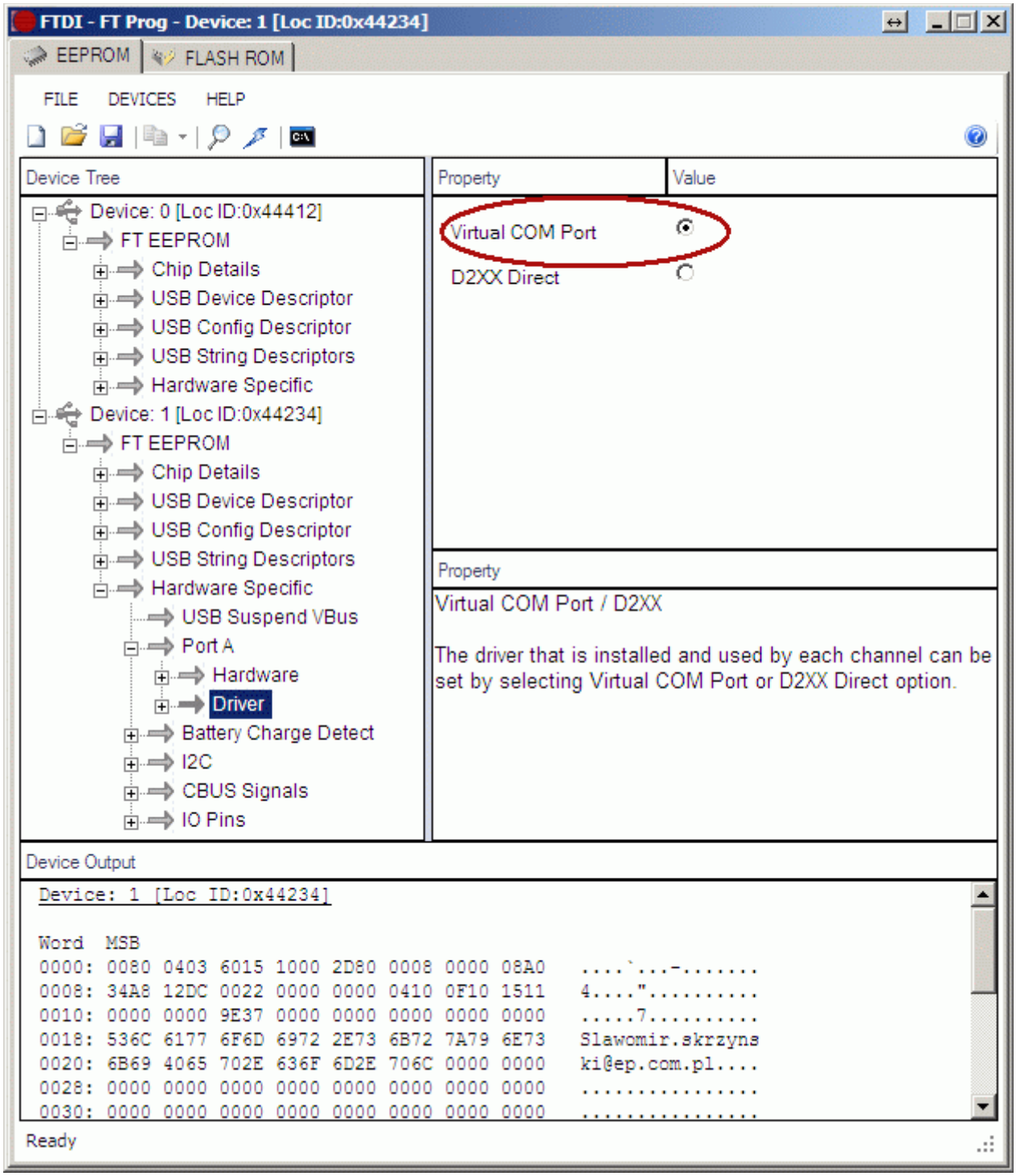


informujący o zapisie do eeprom. Wyjście z trybu uczenia kodów IR nastąpi także po 60 sekundach bezczynności.

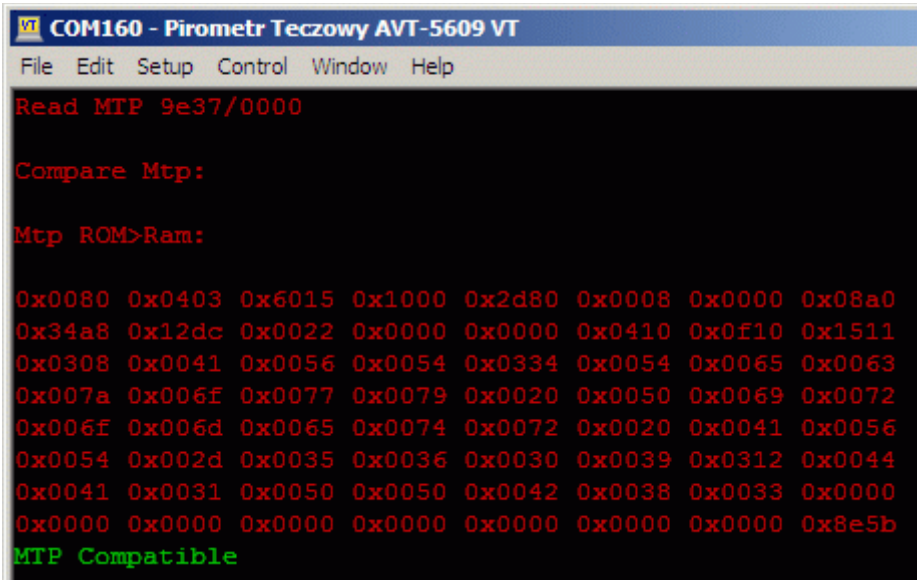
W czasie pracy pirometru, może na chwilę pojawić się napis „SAV”. Ma to miejsce po 10 sekundach od naciśnięcia przycisku S2 „mode” lub odebrania rozkazu z pilota IR.

Obsługa terminalem:

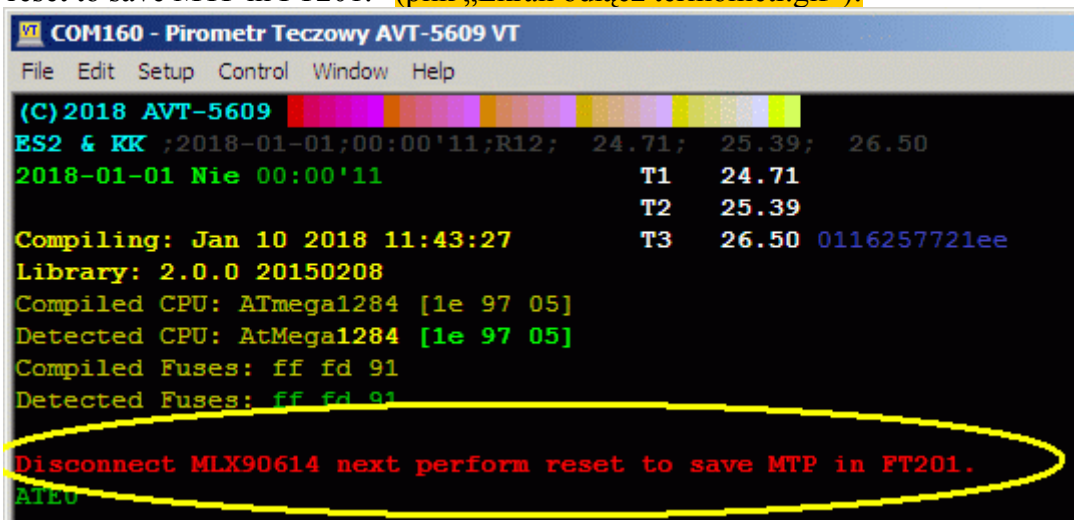
Terminal musi obsługiwać kody ANSI. Zalecam program TeraTerm. Parametry transmisji nie są istotne (ważny jest nr COM). Jeśli pamięć MTP nie jest skonfigurowana, można to zrobić przy użyciu FT_PROG. Wystarczy ustawić VCP (plik „Ekran VCP.gif”):



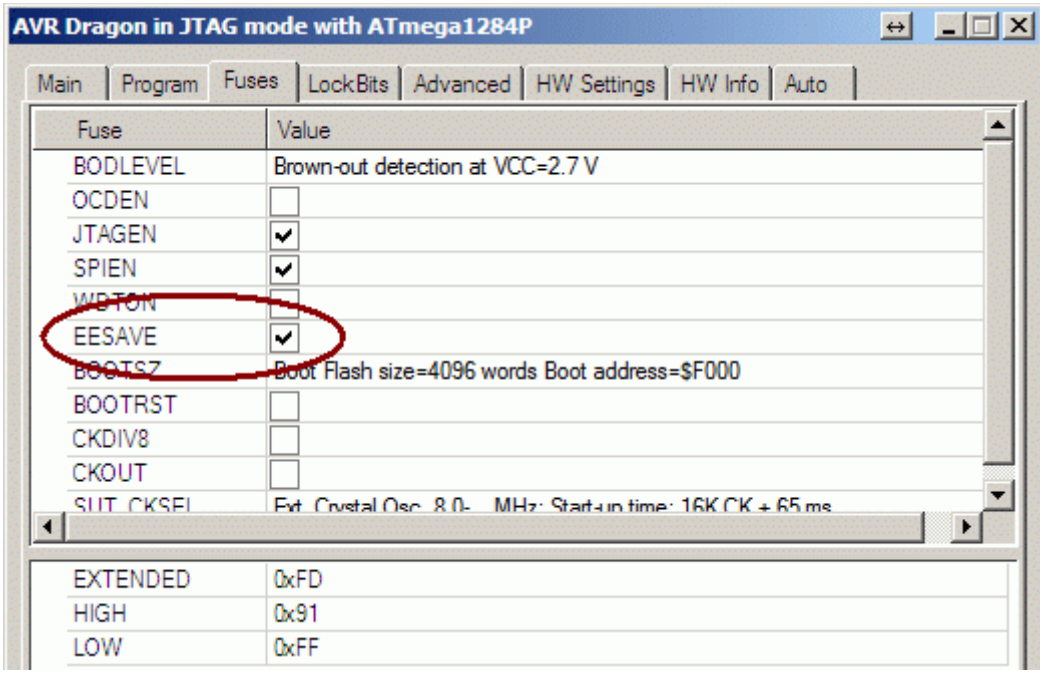
aby uzyskać komunikację. Prościej jednak aby mikrokontroler zapisał MPT. Aby to zrobić, należy odłączyć zewnętrzny zasilacz. Termometr będzie zasilany z USB. Odłączamy pirometr IIC, po czym podłączamy do komputera (nie może to być zasilacz USB, musi to być port pracującego systemu operacyjnego). Pamięć MTP układu FTDI zostanie skonfigurowana ale w komputerze będą dane przed konfiguracją, należy na chwilę odłączyć i przyłączyć wtyk USB. W terminalu pokaże się informacja o konfiguracji (plik „Ekran MTP.gif”):



Jeśli pamięć jest skonfigurowana (komunikat MTP Campatible") można podłączyć pirometr IIC. Podczas funkcjonowania termometru, może pojawić się komunikat „Disconnect MLX90614 next perform reset to save MTP in FT201." (plik „Ekran odłącz termometr.gif"):

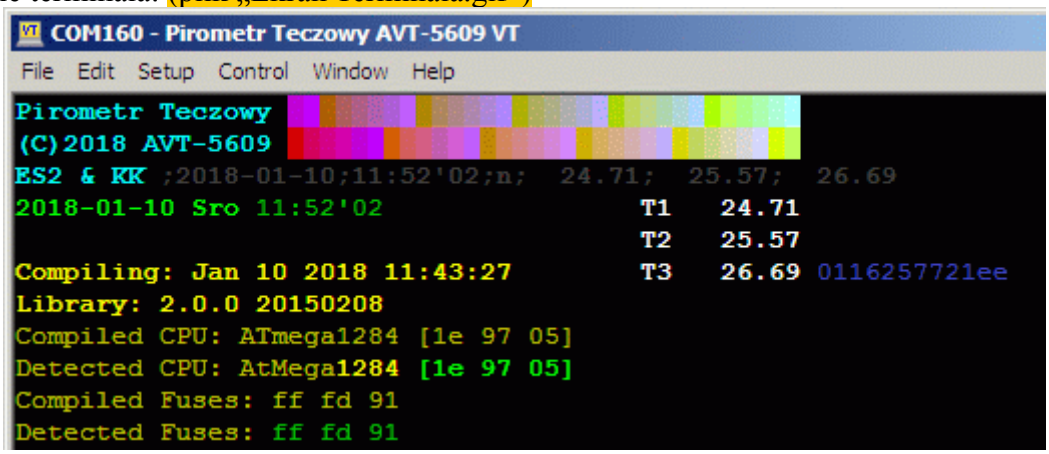


Taka informacja pojawi się, gdy podczas wgrywania programu zostanie zmodyfikowana zawartość pamięci eeprom mikrokontrolera. W Fuses (plik „Ekran Fuses EESAVE.gif"):

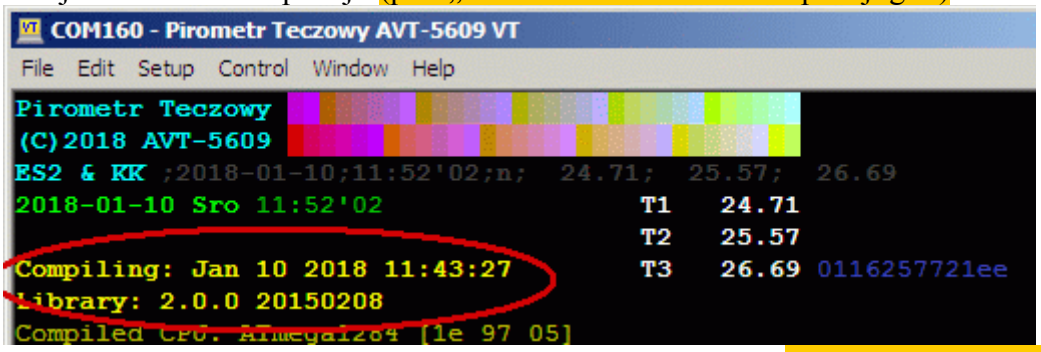


ustawiony jest bit EESAVE i w czasie wgrywania nowego programu do mikrokontrolera eeprom nie jest modyfikowany ale gdy wgra się program używając debugera eeprom może zostać zmodyfikowany.

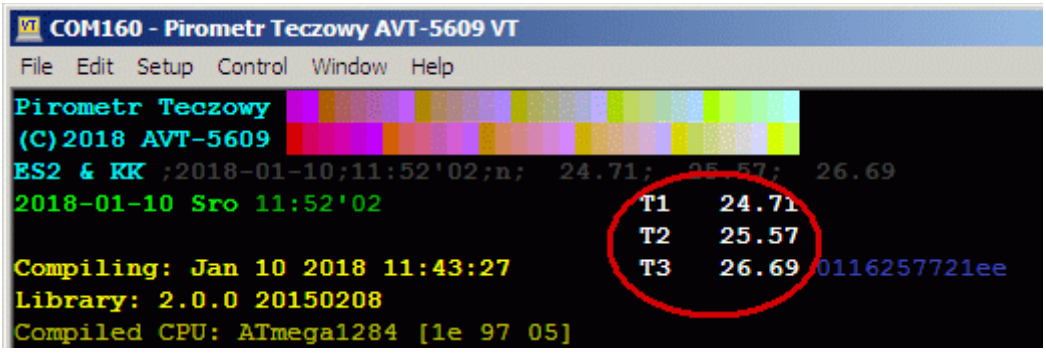
Na ekranie terminala: (plik „Ekran Terminala.gif")



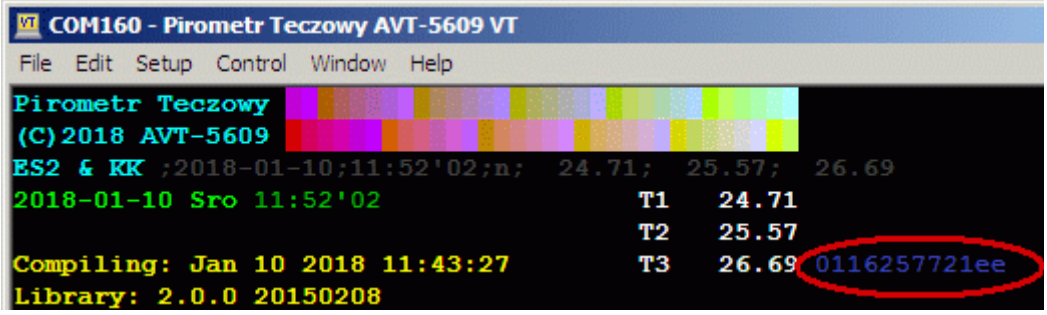
poza informacjami o dacie kompilacji: (plik „Ekran Terminala data kompilacji.gif")



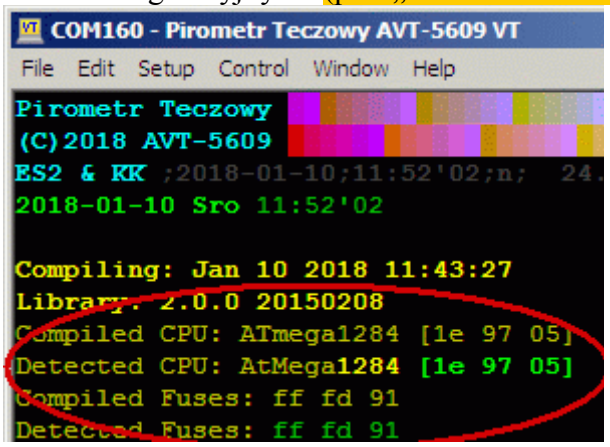
wyświetlana jest temperatura odczytana z podłączonych termometrów:(plik „Ekran Terminala temperatury.gif")



T1 i T2 to temperatury pirometru (T1 – obiektu, T2 – tła), pozostałe to odczyty z DS18B20 (max 7 termometrów tego typu). Po prawej stronie T4..T9): (plik „Ekran Terminala adr DS18B20.gif")



znajdują się adresy poszczególnych układów. Wyświetlane są także informacje o wykrytym procesorze i ustawianiach bitów konfiguracyjnych: (plik „Ekran Terminala cpu-fuses.gif")



Podczas odbioru kodów z pilota IR będą pojawić się informacje o standardzie, adresie, komendzie: (plik „Ekran Terminala Kody IR.gif")

```
RC Protocol:0x2 [NEC] , Adr:0xFB04, Cmd:0x2, Flags:0x0
RC Protocol:0x2 [NEC] , Adr:0xFB04, Cmd:0x2, Flags:0x1
RC Protocol:0x2 [NEC] , Adr:0xFB04, Cmd:0x3, Flags:0x0
RC Protocol:0x2 [NEC] , Adr:0xFB04, Cmd:0x0, Flags:0x0
RC Protocol:0x2 [NEC] , Adr:0xFB04, Cmd:0x0, Flags:0x1
RC Protocol:0x2 [NEC] , Adr:0xFB04, Cmd:0x1, Flags:0x0
RC Protocol:0x2 [NEC] , Adr:0xFB04, Cmd:0x1, Flags:0x1
RC Protocol:0x2 [NEC] , Adr:0xFB04, Cmd:0x11, Flags:0x0
RC Protocol:0x2 [NEC] , Adr:0xFB04, Cmd:0x11, Flags:0x1
RC Protocol:0x2 [NEC] , Adr:0xFB04, Cmd:0x12, Flags:0x0
RC Protocol:0x2 [NEC] , Adr:0xFB04, Cmd:0x12, Flags:0x1
RC Protocol:0x7 [RC5] , Adr:0x0, Cmd:0x10, Flags:0x0
RC Protocol:0x7 [RC5] , Adr:0x0, Cmd:0x10, Flags:0x1
RC Protocol:0x7 [RC5] , Adr:0x0, Cmd:0x11, Flags:0x0
RC Protocol:0x7 [RC5] , Adr:0x0, Cmd:0x11, Flags:0x1
RC Protocol:0x7 [RC5] , Adr:0x0, Cmd:0x1, Flags:0x0
RC Protocol:0x7 [RC5] , Adr:0x0, Cmd:0x2, Flags:0x0
```

Można więc użyć termometru jako testera pilotów.

Wróćmy do obsługi termometru z programu terminala. W ustawieniach należy wyłączyć echo, ponieważ urządzenie odsyła odebrane znaki. Wszystkie komendy rozpoczynają się znakiem „:” (dwukropek), kończą kodem CRLF, spis komend (wielkość znaków ma znaczenie):

:?	wyświetla pomoc
:R	restart urządzenia
:r	odświeża ekran terminala (wykonywane automatycznie co 60 sekund od ostatniego znaku odebranego przez terminal)
:c1	włącza wyświetlanie danych CSV na ekranie terminala
:c0	wyłącza wyświetlanie danych CSV na ekranie terminala (znaki w kolorze tła)
:m1	włącza monitorowanie komunikatów z ESP(Wi-Fi), VNC2 (PenDrive) i pilota IR
:m0	wyłącza monitorowanie komunikatów ESP i VNC-2 oraz pilota IR
:I	tryb nauki kodów IR
:S	nazwa sieci Wi-Fi.
:P	hasło do sieci Wi-Fi.
:d	komendy dla VNC2 (PenDrive)
:i	odczyt adresu IP

Szczegółowego omówienia wymaga komenda „:I”. Po jej wydaniu zostaniemy poproszeni o naciśnięcie na pilocie klawiszy odpowiedzialnych za zmiane nr termometru, jasności i innych. Przykładowy ekran wygląda tak: (plik „Ekran nauka IR.gif”)

```
:I
Press On/Off...  RC5 Adr=0 Cmd=C
Press Bright+ ... Cmd=10
Press Bright- ... Cmd=11
Press 0 ... Cmd=0
Press Ch+ ... Save File 14:15'44
Cmd=20
Press Ch- ... Cmd=21
Press Time ... Cmd=A
Press Range ... Cmd=3CSave...
Science END
```

Wywoływana jest to ta sama procedura co w przypadku przytrzymania S1 „Mode” ponad 5 sekund. Takie same są tez reakcje wyświetlacza 7-segmentowego.

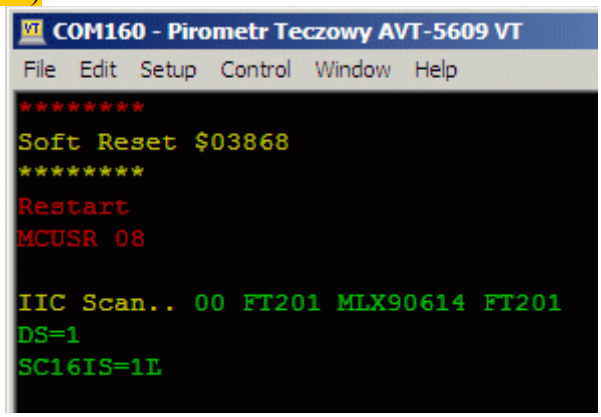
Kody pilota zostaną zapisane w EEPROM. Termometr rozpoznaje następujące standardy: RC5, RC6, NEC, APPLE, Samsung, Samsg32, Kaseikyo, JVC, NEC16, NEC42, Matsushita, DENON,

Sharp, RC5, RC6 & RC6A, IR60 (SDA2008), Grundig, Siemens Gigaset, Nokia, BOSE, Kathrein, NUBERT, FAN (ventilator), SPEAKER (~NUBERT), Bang & Olufsen, RECS80 (SAA3004), RECS80EXT (SAA3008), Thomson, NIKON camera, Netbox keyboard, ORTEK (Hama), Telefunken 1560, FDC3402 keyboard, RC Car, iRobot Roomba, RUWIDO, T-Home, A1 TV BOX, LEGO Power RC, RCMM 12,24, or 32, LG Air Condition, Samsung48, Merlin, Pentax, S100, ACP24, TECHNICS, PANASONIC Beamer, Mitsubishi Aircond, VINCENT, SAMSUNG AH, RADIO, e.g. TEVION.

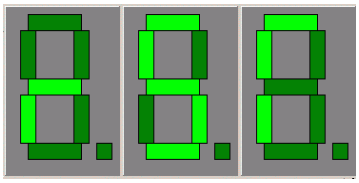
Wyjście w trybu uczenia następuje po naciśnięcie „ESC” w terminalu, naciśnięciu S2 „Mode” lub po 60 sekundach bezczynności.
Do dekodowania kodów pilotów użyto biblioteki Franka Meyera

Uwaga!
W większości pilotów, kody 0..9 ułożone są kolejno , niestety zdarza się, ze jest inaczej. W tej sytuacji, wybór termometru będzie następował przy użyciu klawiszy zapamiętanych jako „Ch+” i „Ch-”.

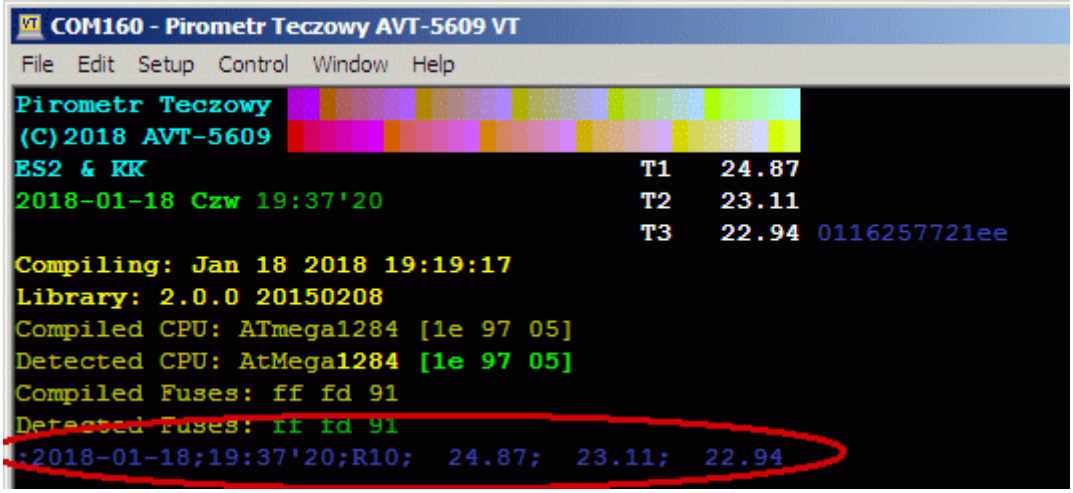
Komenda „R” wywołuje restart termometru:
(plik „Ekran Restart soft.gif”)



Po odebraniu komendy resetu, na wyświetlaczu 7-segmentowym, wyświetli się:

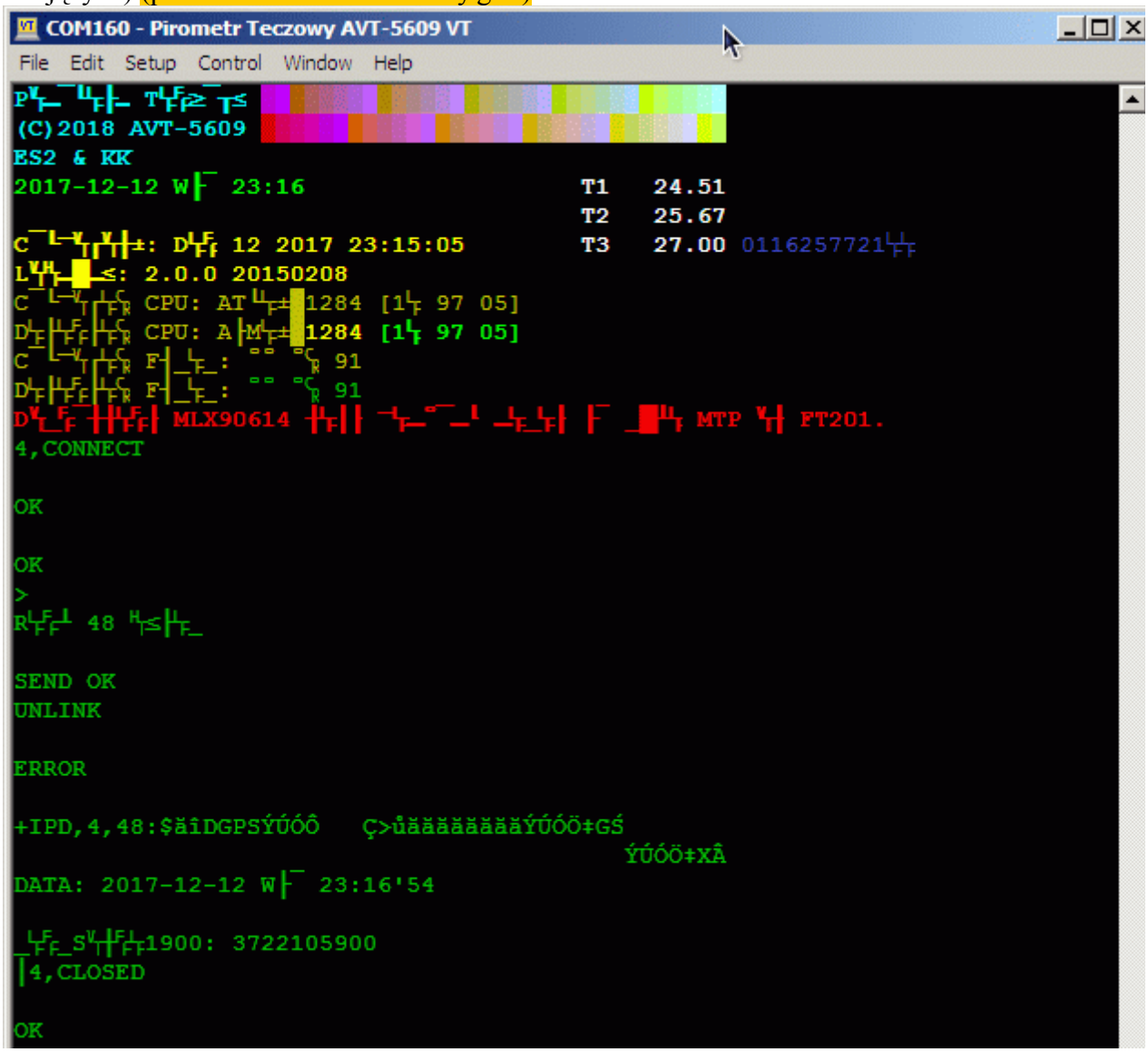


Komenda „:c1” uwidacznia dane CSV w oknie terminala: (plik „Ekran CSV widoczne.gif)

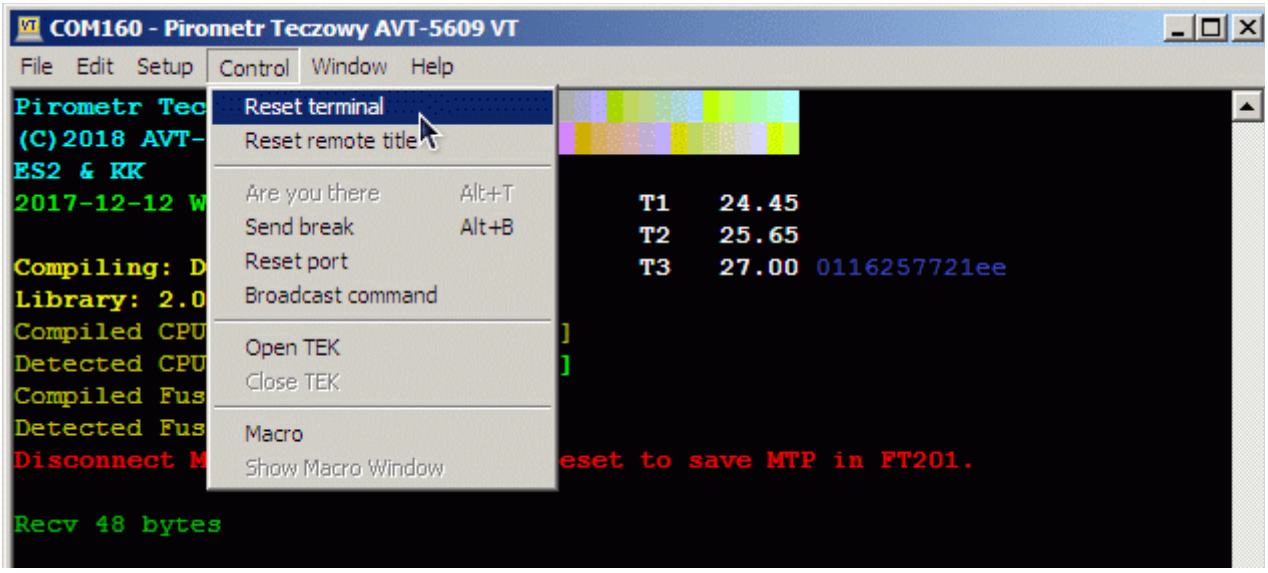


Budowa ramki CSV, zapisywanej co minute na PenDrive zostanie opisana w dalszej części artykułu.

Komenda „”:m1” włącza monitorowanie (wyświetlanie odpowiedzi) z modułu Wi-Fi (ESP8266) i PenDrive (VNC-2). Komenda jest zapamiętywana w eeprom. Wydanie komendy dotyczącej wyżej wymienionych urządzeń („AT” i „:d”) także włącza monitorowanie aby można zobaczyć odpowiedź, ale nie jest to zapisywane w eeprom. Włączenie komendy monitorowania, może powodować błędne wyświetlanie danych w terminalu (interpretowanie danych jako kodów sterujących) (plik”EkranTerminala Bledy.gif”)



W takiej sytuacji trzeba zresetować terminal (plik”Ekran Terminala Reset.gif”)



Wi-Fi:

Termometr może być wyposażony w moduł Wi-Fi. Staje się wtedy serwerem. Adres moduł uzyskuje z DHCP. Nazwę sieci i hasło wprowadzamy komendami „:S" i „:P". Hasło/nazwę sieci wprowadzamy zaraz po komendzie (bez spacji), Nazwa sieci, hasło, zostaną zapamiętane w eeprom. Przykład: (plik „WiFi_Siec_Haslo.gif")

```
:Smoja siec
Save... Save End
:Phaslo
Save... Save End
```

Powyższe rozkazy zapamiętały nazwę sieci „moja siec" i hasło „haslo". Wprowadzone dane zostaną użyte przy kolejnym logowaniu (resecie), który można wywołać komenda „:R".
Jeśli włączone jest monitorowanie („:m1") wszystkie dane przesyłane z modułu Wi-Fi są wyświetlane w terminalu: (plik „WiFi_logowanie.gif")

```
WIFI DISCONNECT
WIFI CONNECTED
WIFI GOT IP

OK
+CIPSTA:ip:"192.168.1.27"
+CIPSTA:gateway:"192.168.1.1"
+CIPSTA:netmask:"255.255.255.0"
```

Terminal jest transparentny dla modułu Wi-Fi i przesyła do niego wszystkie komendy AT: (plik „WiFi Transparent.gif")

```
at+GMR
AT version:1.2.0.0(Jul  1 2016 20:04:45)
SDK version:1.5.4.1(39cb9a32)
Ai-Thinker Technology Co. Ltd.
Dec  2 2016 14:21:16
OK
```

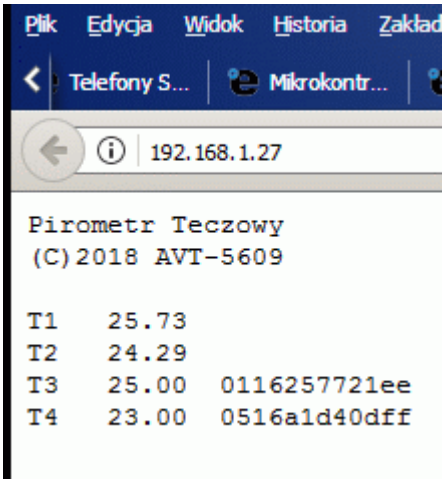
Powyżej przykład komendy „AT+GMR" wyświetlającej wersje modułu WiFi. Dzięki temu można poznać adres który otrzymał moduł z DHCP (komenda AT+CIPSTA?). Ze względu na mały bufor FIFO (64-bajty) dane przychodzące z modułu Wi-Fi mogą być obcięte. Problem można rozwiązać zwiększając rozmiar bufora oraz realizując wysyłanie danych do terminala na przerwaniach, jednak z Wi-Fi do mikrokontrolera nie jest przesyłana duża ilość danych i zrezygnowano z tego. Co ok 60

sekund wysyłane jest zapytanie do serwera NTP. Odczytany czas jest używany podczas zapisu pliku na PenDrive (data/czas utworzenia pliku,modyfikacji oraz w ramce CSV). Sam termometr jest wyposażony w programowy RTC ale nie jest on zbyt dokładny. Pierwszy problem to dokładność kwarcu, na poziomie 30..50ppm, drugi to to, że częstotliwość zastosowanego kwarcu, nie dzieli się przy użyciu preskalera i timera 16-bit tak, aby uzyskać dokładnie 1 sekundę. Nie przewidziano też opcji ustawiania czasu, dlatego brak połączenia z Interntem zaowocuje zapisywaniem błędnej daty/czasu. Można by wyposażyć termometr w zewnętrzny RTC czy na etapie konstruowania wykorzystać asynchroniczny timer T2 na cele RTC. Biorą jednak pod uwagę cenę ESP-01 i możliwości jakie daje oraz fakt, że czas w termometrze nie jest sprawa kluczową, zrezygnowano z takiego rozwiązania. 25 marca 2018 roku program automatycznie przejdzie na czas letni, zmiany na czas zimowy nie przewidziano. Wynika to z tego, że w sejmie znajduje się projekt ustawy, likwidującej zmiany czasu. Po jej wejściu w 2018 roku zmiana czasu z zimowego na letni byłaby ostatnia. Niestety na przeszkodzie może stanąć unijna dyrektywa z 19 stycznia 2001 roku . W razie konieczności zmodyfikuję oprogramowanie uwzględniając zmianę czasu. W programie uwzględniono możliwość przewinięcia licznika sekund 7 lutego 2036 o godzinie 06:28:16 (czas UTC). Wynika to z tego, że czas jest reprezentowany, jako liczba sekund od 1 stycznia 1900 roku (w systemach UNIX od 1 stycznia 1970 roku). Adres serwera jest zapisany w pamięci FLASH mikrokontrolera. Można go znaleźć w funkcji:

```
void wifiASK()
{
…....
…....
        PrintStringSC16IS_P( 0, (char*)PSTR("AT+CIPSTART=4,\"UDP\",\"193.67.79.202\",123,1112,0"CRLF) );    // Otwarcie NTP
```

W aktualnym oprogramowaniu adres ten można zmienić tylko przez ponowną kompilację i wgranie jej do mikrokontrolera. Jeśli będzie potrzeba umożliwienia zmiany adresu serwera z terminala proszę pisać e-maile.

Strona termometru nie jest zbyt okazała:
(plik „Ekran WiFi strona.gif”)



Wyświetlane są informacje podobnie jak w terminalu. Mikrokontroler ma zapas pamięci, więc nie ma problemu aby dodać opcję np. wysyłania danych na serwer. W połączeniu z zapisem na PenDrive, można wysyłać co jakiś czas lub na żądanie, wyniki pomiarów na e-mail czy zewnętrzny serwer w celu zobrazowania zmian temperatury w czasie. W tej sprawie proszę o e-mail'e.

PenDrive:

Co 60 sekund, wyniki pomiaru są zapisywane na PenDrive. Widać to w oknie terminala: (plik „Pen_Zapis.gif”)

Jeśli monitorowanie jest wyłączone („m0”) pojawi się tylko napis Save File.

Dane są zapisane w formacie CSV w pliku „data.csv”. Jeśli plik istnieje, dane są do niego dopisywane, jeśli nie, zostanie utworzony”. PenDrive musi być sformatowany w FAT16 lub FAT32. Ramka CSV zbudowana jest następująco:

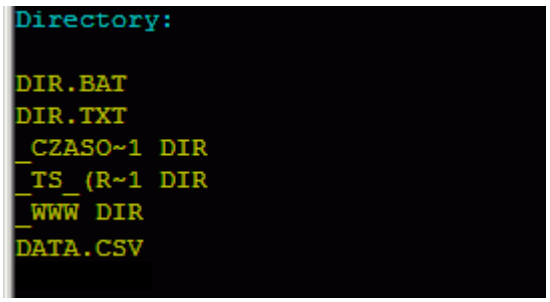
;YYYY-MM-DD;gg:mm'ss;S; T0; T1; T2[CRLF]

YYYY-MM-DD	rok, miesiąc, dzień		
gg:mm'ss	godzina, minuta, sekunda		
S	status	Rxx	przyczyna resetu gdzie xx=MCUSR
		?	nieznany
		N	synchronizacja NTP
		n	brak synchronizacji z NTP przez co najmniej godzinę
T0..Tn	temperatury poszczególnych termometrów		
[CRLF]	znak końca linii		

np.

;2018-01-08;17:22'00;n; 26.43; 25.75; 27.38

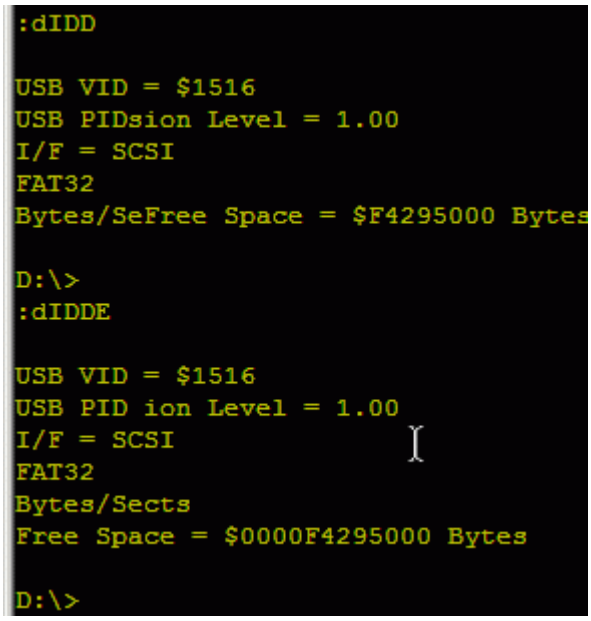
Zawartość PenDrive (główny katalog) można wyświetlić komendą „d”: (plik „Pen_Dir.gif”).



„:d” z parametrem – wydanie komendy dla układu VNC2. Każda komenda dla VNC, ustawia timeout zapisu pliku CSV na 60 sekund. Dzięki czemu, automatyczny zapis, nie przeszkadza podczas operacji na dysku (zakładanie, kasowanie pliku itp.), Spis najważniejszych komend:

CD file	otworzenie katalogu
CD ..	cofnięcie się o jeden katalog w górę
RD file	wyświetlenie zawartości pliku
DLD ile	skasowanie pliku
MKD file	utworzenie katalogu
REN file file	zmiana nazwy pliku lub katalogu
IDDE	informacje o dysku
IDD	informacje o dysku

Wszystkie komendy są opisane w dokumentacji układu VNC-2. Przykłady użycia komend (plik „PenDrive Cmd 1.gif”)



Tu także pojawia się „wąskie gardło” podczas przesyłania danych z VNC2 do terminala przez mostek IIC-USB. Problem można rozwiązać przenosząc wysyłanie po IIC na przerwania i zwiększając wielkość bufora.
Obsługiwane są podstawowe błędy dyskowe, takie jak „No Disk” czy „Disk Full” aby nie doprowadzić do zawieszenia się oprogramowania. Po wydaniu komendy „:d” może pojawić się błąd „DriveLock” jeśli wykonywane są jakieś operacje na dysku podczas próby wydania komendy dyskowej.
VNC-2 obsługuje pamięci masowe na porcie 2 (J12). Na porcie 1 (J10) obsługiwane mogą być drukarki, klawiatury, myszki oraz układy produkowane przez firmę FTDI takie jak np. FT232, FT245. Aktualnie oprogramowanie termometru nie obsługuje tych układów.

Dla programistów:

Z pirometrem mogą komunikować się termometry zdalne. Kolejne termometry oznaczane są jako Ta, Tb, Tc itd. Termometr zdalny wysyła do pirometru ramkę tekstową w formacie:
PUT / TXT Tx jjj.dd[CRLF]

gdzie:	PUT / TXT T	nagłówek informujący o ramce z termometru zdalnego
	x	nr termometru oznaczony kolejnymi literami a, b, c...g
	jjj	jedności temperatury, jeśli ujemna poprzedzone znakiem „-”
	.	separator
	dd	dziesiąte części temperatury

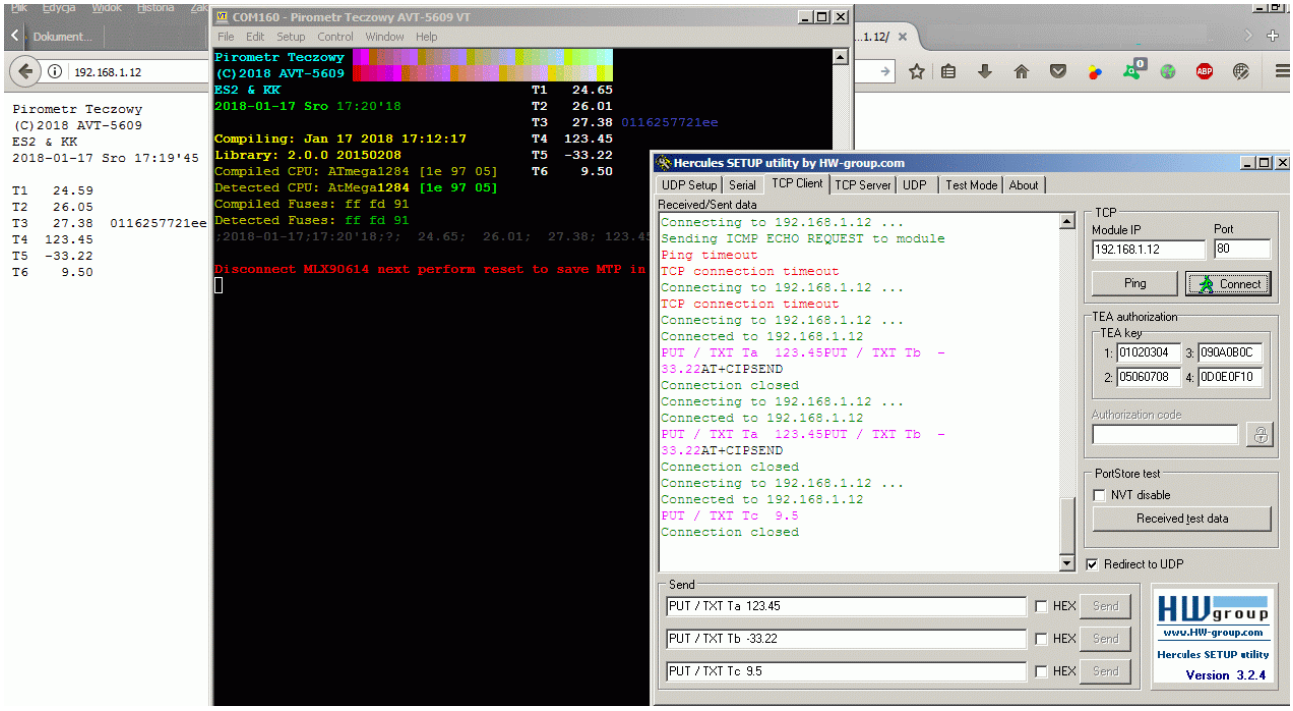
Kilka przykładów:

PUT / TXT Ta 123.45	termometr „a” temperatura 123,45
PUT / TXT Tb -12.34	termometr „b” temperatura -12,34

Ramka nie musi być zakończona znakami CR+LF ale w takim przypadku nie odeśle potwierdzenia komendy w postaci „echa” składającego się z ciągu znaków znajdującego się za „PUT / TXT T”.
Przykładowo:

PUT / TXT Ta 123.45[CRLF]	echo Ta 123.45[CRLF]
PUT / TXT Tb -12.34[CRLF]	echo Tb -12.34[CRLF]

Na poniższym zrzucie ekranowym, widać efekt wysłania z programu „Hercules” danych z trzech termometrów zdalnych.
(plik „Ekran ZdalneTermometry.gif)



Zarówno w programie terminala jak i w przeglądarce pojawiły się dane z termometrów.

WS2812 na przerwaniach od USART:

Diody adresowalne, nazywane także diodami inteligentnymi, mają dość restrykcyjne zależności czasowe. Sterowanie diodami przez manipulowanie GPIO ma tą wadę, że angażuje procesor w 100% podczas transmisji danych do diody. Użycie do tego celu interfejsu SPI, niewiele poprawia sytuację. Problemem jest to, że aby wysłać jeden bit do diody, trzeba zaangażować 3 bity SPI. Ze względu na to, potrzeba 9 bajtów na jedną diodę. Nie to jest największym problemem. Główny kłopot to to, że bajt może kończyć się wartością jeden. SPI zgłosi przerwanie, które musi być natychmiastowo obsłużone, ponieważ czas trwania stanu wysokiego dla WS2812 jest dość restrykcyjny. W praktyce oznacza to, że wszystkie inne przerwania muszą być nieblokowane czyli zadeklarowane jako INTERRUPT lub ISR z atrybutem ISR_NO_BLOCK. Niestety, przerwania odbiorcze USART, muszą być SIGNAL lub ISR bez atrybutu ISR_NO_BLOCK. W praktyce, powoduje to nieprzewidywalne zachowanie się diod podczas wysyłania danych do nich i równoczesnego odbioru danych przez USART.
Problem został rozwiązany przez wysyłanie danych do diod przy użyciu USART. Podobnie jak w przypadku SPI, jeden bit dla diody, to 3 bity USART ale wykorzystano także bit startu i stopu. Z tego tytułu są dwie korzyści. Pierwsza to 8 a nie 9 bajtów danych na diodę. Drugą, ważniejszą korzyścią, jest to, że transmisja zawsze kończy się bitem stopu, czyli stanem niskim (dane z USART muszą być zanegowane). Dzięki temu kolejna transmisja może nastąpić później niż w przypadku transmisji po SPI. To natomiast umożliwia bezproblemowe wykorzystanie innego USART'a do odbioru danych. Poniżej szczegółowy opis jak zrealizowano transmisję.
Ramka transmisyjna UART 7n1 składa się z dziewięciu odcinków czasowych (bit startu, 7 danych, bit stopu) więc w jednym bajcie można przesłać 3 bity do diody. Bit startu i stopu ma z góry ustaloną wartość. Dioda led a właściwie jej sterownik, przyjmuje impulsy, w których w pierwszym odcinku zawsze jest jeden, drugi decyduje o wartości przesyłanej informacji, trzeci zawsze jest zerem. Niestety bit startu to zero a stopu jeden. Po zanegowaniu sygnału UART;a otrzymujemy sygnał wymagany przez LED-y. Aby wysłać trzy bity o wartości 0 do UART-a należy wpisać wartość $26 (binarnie 100110). Tabela wyjaśni dlaczego:

	Bit startu	7	6	5	4	3	2	1	Bit stopu

Wartość bajtu w UART	0	A	1	0	B	1	0	C	1
Zanegowane bity na wyjściu UART	1	/A	0	1	/B	0	1	/C	0

Na pomarańczowo zaznaczono bity startu i stopu, na żółto stałą wartość jaka musi być wysyłana UART-em, litery A, B, C ustalają wartości bitów 1, 2, 3 transmitowanych do LED-a.
Jaką maksymalną przepływność może uzyskać UART w AVR? Według noty katalogowej:
Fosc / 8 * UBRR + 1
Dla 20MHz i UBRR=1 otrzymamy 1,25Mb/s czyli dwa razy za wolno. Okazało się, że można tam wpisać zero otrzymując przepływność 2,5Mb/s:
20MHz bez optymalizacji.gif

Czy da się transmisję zrealizować na przerwaniach? Jak najbardziej jest to możliwe nawet bez wstawki assemblerowej. Wynik działania programu obsługi przerwania:

```
SIGNAL(USART1_TX_vect)
{
        if ( pozScr )
                {
                UDR1 = *(scr+pozScr++);
                pozScr &=SCRLEN-1;
                }
        if ( pozScr >= SCRLEN ) FL_TransEnd=true;
}
```

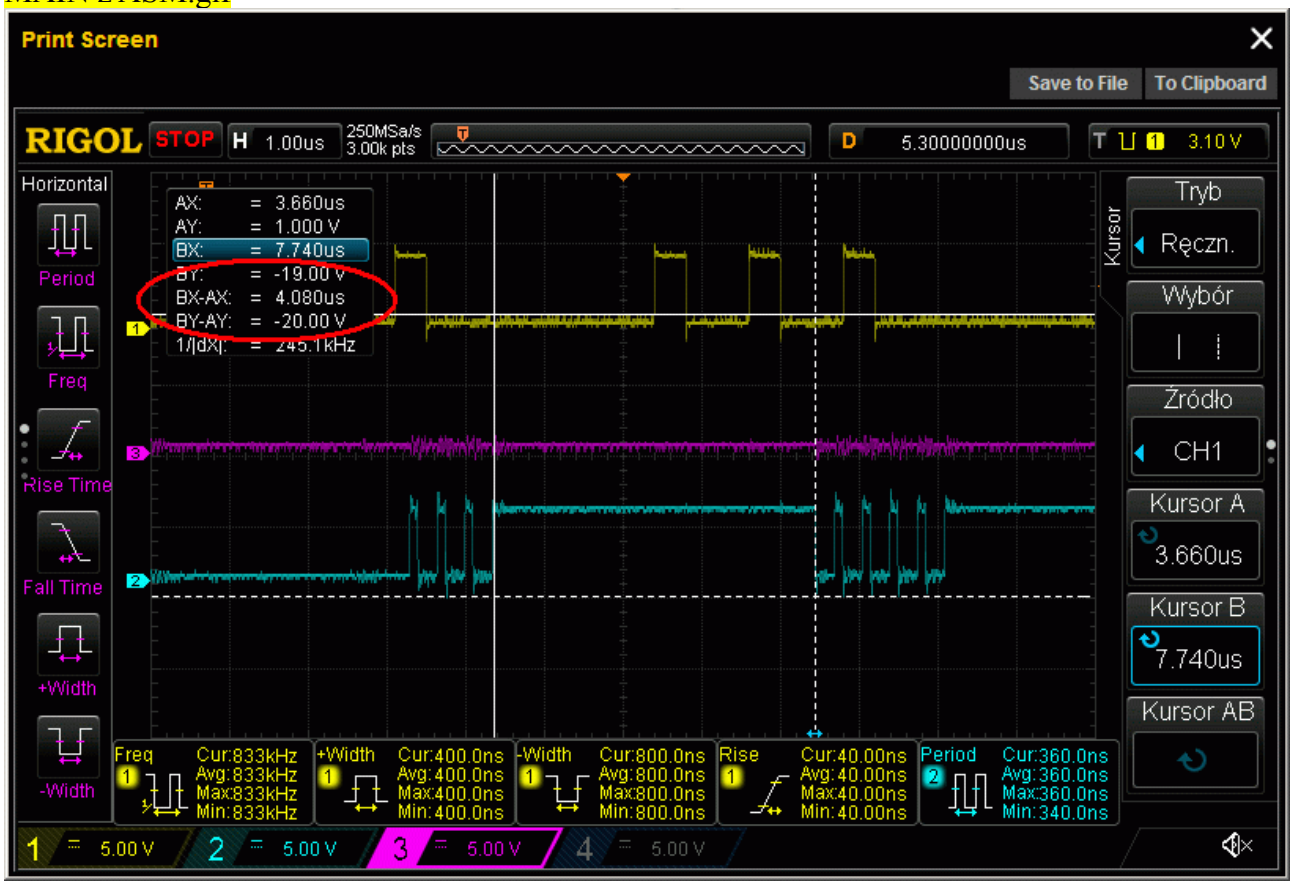
pokazano na ponizszym ekranie:
MAIN bez ASM.gsm

Żółty przebieg to dane wysyłane do WS2812, niebieski zmiana GPIO w pętli main. Jak widać obciążenie procesora jest dość duże. Czas obsługi IRQ wynosi 4,44us więc na pewno nie będzie zinterpretowany przez diody jako reset. Po optymalizacji (niepotrzebne operacje wyróżniono kolorem):

```
ISR( USART1_TX_vect, ISR_NAKED )
{
asm volatile (
// usunięta operacja na nieużywanym  R1 i R0
//          "push   r1                  \n\t"
//          "push   r0                  \n\t"
//          "in     r0, 0x3f  \n\t"     // SREG
//          "push   r0                  \n\t"
//          "eor    r1, r1    \n\t"
            "push   r18                 \n\t"
//SREG na stos przy użyciu używanego później R18 (+3)
            "in     r18, 0x3f           \n\t"     // SREG
            "push   r18                 \n\t"
//Dalej tak jak zrobił kopilator
            "push   r24                 \n\t"
            "push   r25                 \n\t"
            "push   r30                 \n\t"
            "push   r31                 \n\t"
           );
//I dotychczasowy kod:
            if ( pozScr )
                    {
                    UDR1 = *(scr+pozScr++);
                    pozScr &=SCRLEN-1;
                    }
            if ( pozScr >= SCRLEN ) FL_TransEnd=true;
asm volatile (
            "pop    r31                 \n\t"
            "pop    r30                 \n\t"
            "pop    r25                 \n\t"
            "pop    r24                 \n\t"
//Dodane SREG przy użyciu R18
            "pop    r18                 \n\t"
            "out    0x3f, r18\n\t"
//Dalej normalnie R18
            "pop    r18                 \n\t"
//Usunięte operacje na R0 i R1
```

```
//          "pop    r0                  \n\t"
//          "out    0x3f, r0\n\t"
//          "pop    r0                  \n\t"
//          "pop    r1                  \n\t"
            );
            reti();;
}
```

czas przerwania zmniejszył się do 4,08us. Łączna oszczędność 7 rozkazów/cykli ma dość duże znaczenie z racji bardzo częstego wywoływania przerwań. Efekt działania zoptymalizowanej procedury:
MAIN z ASM.gif



Niebieski przebieg pokazuje pętlę główna programu:

```
loop:
 wdg();
 PORTA |= _BV(PA3);
 PORTA &= ~_BV(PA3);
```

która po skompilowaniu wykonuje się w czterech cyklach maszynowych. Jak łatwo zauważyć pomiędzy przerwaniami program główny wykonuje około 16 rozkazów. Dość mało i niewiele da się zrobić w takim czasie. Jeśli jednak weźmiemy pod uwagę to, że transmisja nie musi trwać bez przerwy i pomiędzy paczkami wstawimy pauzę o długości 20ms, co da odświeżanie LED 50Hz to przy 60 LED-ach otrzymamy:
transmisja do LED: 60 * 1,2us * 24 bity ~1,7ms czyli zajętość CPU na poziomie 8,5%.
Przykładowe obliczenia obciążenia CPU:

Liczba LED	Odświeżanie	Częstotliwość odświeżania	Czas transmisji	Zajętość CPU

400	20 ms	50 Hz	11,5 ms	57%
400	50 ms	20 Hz	11,5 ms	23,00%
60	50 ms	20 Hz	1,7 ms	3,40%

Jak pokazałem wcześniej, rozwinięcie Assemblerowe wykazało niepotrzebne operacje na R0 i R1 (to jest niestety cecha AVR-GCC), które nie są później używane (żółte tło) i przechowywanie na stosie SREG z użyciem R0 (pomarańczowe tło), którą to operację można wykonać używając któregoś z rejestrów używanych w IRQ, które z konieczności muszą być odłożone na stosie. Wydawać by się mogło, że oszczędność 7 cykli zegarowych w przerwaniu to niewiele ale należy pamiętać, że przerwania są wywoływane z częstotliwością ok230kHz przy taktowaniu 18MHz.
Mała wskazówka dotycząca optymalizacji. Aby całego programu nie pisać od początku w assemblerze wystarczy po skompilowaniu skopiować rozwinięcie asseblerowe. Operację ułatwi „TextPad”, który posiada opcję zaznaczania kolumnowego. Taki kod wystarczy tylko poprawić i umieścić w swoim programie.

Większość WS2812 akceptuje poziom niski o czasie 15..25us ale trafiają się takie, którym 5us wystarczy aby potraktować to jako reset. Producent zauważył, że jest to problem i WS2813 są mniej restrykcyjne jeśli chodzi o czasy impulsów, ponadto mają przyjemną cechę pomijania uszkodzonych diod.

Podczas pisania oprogramowania, przez nieuwagę przerwanie obsługujące pilota IR (10kHz) było zadeklarowane jako SIGNAL a nie INTERRUPT. Powodowało to czasami rozbłyski led'ów. Znalezienie błędu było kłopotliwe, bo rozbłyski zdarzały się co najwyżej kilka razy na godzinę. Po zadeklarowania przerwania jako INTERRUPT problemy zniknęły.

Odbiór danych z VNC przez USART0:

W programie mogą być bez ograniczeń używane przerwania nieblokowane (INTERRUPT czy ISR z atrybutem NOBLOCK). Sytuacja się komplikuje gdy istnieje konieczność użycia przerwań blokowanych np. od USART-a. Jak wiadomo przerwanie to musi być typu SIGNAL lub ISR (z domyślnym atrybutem BLOCK) w przeciwnym wypadku nastąpi przepełnienie stosu. Jeśli przerwanie wykonywało będzie się w czasie krótszym niż około 20us nie będzie problemu. Dlaczego 20us? Jakkolwiek czas resetu WS2812 to 50us ale jest to czas, który gwarantuje reset i w wielu przypadkach następuje on gdy poziom niski trwa 15..25us. Co więc zrobić jeśli przerwanie trwa dłużej albo tego typu przerwań jest więcej? Istnieją dwa rozwiązania:

1) Najszybciej jak to możliwe w przerwaniu wykonać komendę sei(). W przypadku USART-a można to zrobić po odczytaniu rejestru UDR (w praktyce bezpośrednio przed bo po wykonaniu sei() gwarantowane jest wykonanie co najmniej jednego rozkazu przez mikrokntroler).
2) Jeśli przerwanie dotyczy UART-a można do sprawdzania jego stanu użyć przerwania od timera. Tu należy wspomnieć, że w przypadku przerwań nadawczych nie ma problemu bo mogą one być nieblokowane (flaga przerwania jest kasowana po wejściu w IRQ a nie po operacji na UDR).

Najprostsza implementacja rozwiązania nr jeden pokazana jest na listingu:

```
SIGNAL(USART0_RX_vect)
{
  2782:  1f 92        push    r1
  2784:  0f 92        push    r0
  2786:  0f b6        in      r0, 0x3f    ; 63
  2788:  0f 92        push    r0
  278a:  0b b6        in      r0, 0x3b    ; 59
  278c:  0f 92        push    r0
  278e:  11 24        eor     r1, r1
  2790:  8f 93        push    r24
  2792:  9f 93        push    r25
  2794:  ef 93        push    r30
  2796:  ff 93        push    r31
  2798:  80 91 c6 00  lds     r24, 0x00C6
  279c:  80 93 44 06  sts     0x0644, r24
  27a0:  78 94        sei
  27a2:  90 91 44 06  lds     r25, 0x0644
Usart0Loop:
            Usart0_RxBuf[Usart0_PtrRxW++] = DanaUdr0;                       // wpisz znak do bufora i zwiększ długość
  27a6:  80 91 8f 01  lds     r24, 0x018F
  27aa:  e8 2f        mov     r30, r24
  27ac:  f0 e0        ldi     r31, 0x00   ; 0
  27ae:  e2 50        subi    r30, 0x02   ; 2
  27b0:  f7 4f        sbci    r31, 0xF7   ; 247
  27b2:  90 83        st      Z, r25
  27b4:  8f 5f        subi    r24, 0xFF   ; 255
  27b6:  80 93 8f 01  sts     0x018F, r24
            if( Usart0_PtrRxW >= USART0_RXBUF) Usart0_PtrRxW=0;           // Ograniczenie wielości bufora (max 255 bajtów)
  27ba:  80 91 8f 01  lds     r24, 0x018F
  27be:  87 ff        sbrs    r24, 7
  27c0:  02 c0        rjmp    .+4         ; 0x27c6 <__vector_20+0x44>
  27c2:  10 92 8f 01  sts     0x018F, r1
            #ifdef USART0_FAST_FIFO
             if( (UCSR0A & (1<<RXC0)) )                                     // Odczytuj dopóki sa znaki w FIFO
  27c6:  80 91 c0 00  lds     r24, 0x00C0
  27ca:  87 ff        sbrs    r24, 7
  27cc:  03 c0        rjmp    .+6         ; 0x27d4 <__vector_20+0x52>
                      {
                      DanaUdr0 = UDR0;
  27ce:  90 91 c6 00  lds     r25, 0x00C6
  27d2:  e9 cf        rjmp    .-46        ; 0x27a6 <__vector_20+0x24>
                      goto Usart0Loop;
  27d4:  90 93 44 06  sts     0x0644, r25
                      }
             UCSR0B |= (1<<RXCIE0);                                         // Właczenie przerwań odborczych
  27d8:  80 91 c1 00  lds     r24, 0x00C1
  27dc:  80 68        ori     r24, 0x80   ; 128
  27de:  80 93 c1 00  sts     0x00C1, r24
}
  27e2:  ff 91        pop     r31
  27e4:  ef 91        pop     r30
  27e6:  9f 91        pop     r25
  27e8:  8f 91        pop     r24
  27ea:  0f 90        pop     r0
  27ec:  0b be        out     0x3b, r0    ; 59
  27ee:  0f 90        pop     r0
  27f0:  0f be        out     0x3f, r0    ; 63
  27f2:  0f 90        pop     r0
  27f4:  1f 90        pop     r1
  27f6:  18 95        reti
```

Na żółto oznaczono niepotrzebne operacje (SREG można zapamiętać używając np. rejestru R25). Rozkaz SEI wykonany jest po 23[*1] cyklach maszynowych (rozkazy PUSH i LDS wykonują się w 2 cykle).

Po optymalizacji:

```
ISR( USART0_RX_vect, ISR_NAKED )
{
__asm__ volatile (
   27b8:   9f 93        push    r25
   27ba:   9f b7        in      r25, 0x3f   ; 63
   27bc:   9f 93        push    r25
   27be:   90 91 c6 00          lds     r25, 0x00C6
   27c2:   78 94        sei
   27c4:   90 93 ad 09          sts     0x09AD, r25
   27c8:   8f 93        push    r24
   27ca:   1f 92        push    r1
   27cc:   11 24        eor     r1, r1
   27ce:   ef 93        push    r30
   27d0:   ff 93        push    r31
           Usart0_RxBuf[Usart0_PtrRxW++] = bUDR0;                  // wpisz znak do bufora i zwiększ długość
   27d2:   90 91 91 01          lds     r25, 0x0191
   27d6:   e9 2f        mov     r30, r25
   27d8:   f0 e0        ldi     r31, 0x00   ; 0
   27da:   ef 50        subi    r30, 0x0F   ; 15
   27dc:   f7 4f        sbci    r31, 0xF7   ; 247
   27de:   80 91 ad 09          lds     r24, 0x09AD
   27e2:   80 83        st      Z, r24
   27e4:   9f 5f        subi    r25, 0xFF   ; 255
   27e6:   90 93 91 01          sts     0x0191, r25
           if( Usart0_PtrRxW >= USART0_RXBUF) Usart0_PtrRxW=0;     // Ograniczenie wielości bufora
   27ea:   80 91 91 01          lds     r24, 0x0191
   27ee:   87 ff        sbrs    r24, 7
   27f0:   02 c0        rjmp    .+4         ; 0x27f6 <__vector_20+0x3e>
   27f2:   10 92 91 01          sts     0x0191, r1

__asm__ volatile (
   27f6:   ff 91        pop     r31
   27f8:   ef 91        pop     r30
   27fa:   1f 90        pop     r1
   27fc:   8f 91        pop     r24
   27fe:   9f 91        pop     r25
   2800:   9f bf        out     0x3f, r25   ; 63
   2802:   9f 91        pop     r25
   2804:   18 95        reti
```

SEI wykonuje się po 7[*1] cyklach maszynowych. Niestety jest problem z wbudowanym w AVR FIFO. Pojawienie się 2 znaków spowoduje wywołanie irq usart zaraz po SEI. Nawet gdy odczytany bajt, nie byłby przechowywany w zmiennej co powoduje „zamazanie znaku” tylko przechowany w rejestrze, to zamieni się kolejności bajtów. Trzeba by więc SEI wykonać po zapisie danej w buforze w RAM. Znalazłem rozwiązanie. Blokuje przerwania odbiorce usart, po czym wykonuję SEI

```
ISR( USART0_RX_vect, ISR_NAKED )
{
__asm__ volatile (
   2782:   9f 93        push    r25
   2784:   9f b7        in      r25, 0x3f   ; 63
   2786:   9f 93        push    r25
   2788:   90 91 c1 00          lds     r25, 0x00C1
   278c:   9f 77        andi    r25, 0x7F   ; 127
   278e:   90 93 c1 00          sts     0x00C1, r25
   2792:   78 94        sei
   2794:   8f 93        push    r24
   2796:   ef 93        push    r30
   2798:   ff 93        push    r31


   279a:   90 91 c6 00          lds     r25, 0x00C6                     // odczyt UDR
Usart0Loop:
            Usart0_RxBuf[Usart0_PtrRxW++] = DanaUdr0;                   // wpisz znak do bufora i zwiększ długość
   279e:   80 91 8f 01          lds     r24, 0x018F
   27a2:   e8 2f        mov     r30, r24
   27a4:   f0 e0        ldi     r31, 0x00   ; 0
   27a6:   e2 50        subi    r30, 0x02   ; 2
   27a8:   f7 4f        sbci    r31, 0xF7   ; 247
   27aa:   90 83        st      Z, r25
   27ac:   8f 5f        subi    r24, 0xFF   ; 255
   27ae:   80 93 8f 01          sts     0x018F, r24
            if( Usart0_PtrRxW >= USART0_RXBUF) Usart0_PtrRxW=0;         // Ograniczenie wielości bufora (max 255 bajtów)
   27b2:   80 91 8f 01          lds     r24, 0x018F
   27b6:   87 ff        sbrs    r24, 7
   27b8:   02 c0        rjmp    .+4         ; 0x27be <__vector_20+0x3c>
   27ba:   10 92 8f 01          sts     0x018F, r1
             if( (UCSR0A & (1<<RXC0)) )                                 // Odczytuj dopóki są znaki w FIFO
   27be:   80 91 c0 00          lds     r24, 0x00C0
   27c2:   87 fd        sbrc    r24, 7
   27c4:   ea cf        rjmp    .-44        ; 0x279a <__vector_20+0x18>
              UCSR0B |= (1<<RXCIE0);                                    // Właczenie przerwań odborczych
   27c6:   80 91 c1 00          lds     r24, 0x00C1
   27ca:   80 68        ori     r24, 0x80   ; 128
   27cc:   80 93 c1 00          sts     0x00C1, r24
```

Dzięki temu zabiegowi, rozkaz SEI wykonuje się po 10[*1] cyklach maszynowych a bufor FIFO działa poprawnie. Ponadto odczytany znak z UDR jest przechowywany w rejestrze a nie zmiennej globalnej.

* - Rozkazy wykonują się w większości w 2 cyklach (pomijając EOR, ANDI, SEI) więc zysk czasowy jest duży choć nawet używając standardowego przerwania SIGNAL przerwanie od USART nie powinno przekroczyć 5us.

Trochę się rozpisałem o USART AVR'a, ale jest to jedna z jego „ułomności” brak priorytetowego systemu przerwań, z którego znany był Z-80, Motorolli 68k czy choćby popularnego 8051. W AVR można to uzyskać w sztuczny sposób, ale czasem jest to zadanie skomplikowane. Niestety dotyczy to często używanego USART'a. Światełko w tunelu pojawiło się nowych AtTiny (Tiny1614, 1616, 1617), które są wzorowane na Xmega. Nie wiadomo jednak czy zostanie to wdrożone w linii Mega.

Kody źródłowe dostępne są w materiałach dodatkowych i na http://avt.4ra.pl/

EP, ES2 & KK

Spis elementów:

Pirometr:

R1 R4 R6 R7 R8 R10 R11	1k	1206	Rezystor
R9 R14 R15 R24 R25	4k7	1206	Rezystor
R12	10R	1206	Rezystor
R5 R19 R20	10k	1206	Rezystor
R2 R3 R16 R17 R18 R21	27R	1206	Rezystor
RP1	510R * 8	1206	
R13 R22	510R	1206	Rezystor
C1	1000uF/25V	ce8/35	Electrolytic Capacitor
C4 C5 C15 C19 C23	10uF	1206	Electrolytic Capacitor
C12	10uF	ce6.3/2.5	Kondensator Elektrolityczny
C16 C17 C18 C20	22pF	1206	Kondensator Ceramiczny
C7 C8 C24 C25 C26 C27	47pF	1206	Kondensator Ceramiczny
C2 C3 C6 C9 C10 C11 C13 C14 C21 C22 C28 C29	100nF	1206	Capacitor
U1	7805	78xxR	Positive regulator
U2	74HC00D	SO-14	Quadruple Positive-NAND Gate with Schmitt-Trigger Input
U3	FT201XS	SSOP-16	USB-IIC Slave 3,4MB/a
U4	DS9503	SO-6	
U5	ATMEGA664/1284-TQFP44	TQFP-44	
U6	TSOP38	TSOP	Scalony odbiornik podczerwienie
U7	VNC2-32L1B	LQFP-32	Vinculum2
U8	SPX1117	SOT-223	
M1	DF06S	DF06S	Mostek Prostowniczy
D1	5V	1206	Dioda LED Zielona
D2	USB	1206	Dioda LED Niebieska
D3	RX	1206	Dioda LED Zielona
D4	TX	1206	Dioda LED Zolta
D5	OPD-T5620LB-BW	LED_3x7-seg	3 wyświetlacze 7-seg
D6	IR	Dioda nadawcza IR	
D7	3V3	1206	Dioda LED Zielona
T1 T3	MMBTA56LT1G	SOT-23	PNP Transistor
T2	MMBTA56LT1G / BC607	SOT-23	PNP Transistor
BZ1	BUZZER 5V	Buzzer	
Q2	12MHz	KWARC_SMD	
Q1	20MHz	KWARC_SMD	
F2 F3	PTC 0.5	1210	Fuse
F1	PTC 100mA	1210	Fuse
J12	BOMS Flash Disk	USBA-G-A	Gniazdo USB kątowe
J10	Slave Pheripheral	USBA-G-A	Gniazdo USB kątowe
J4	USBB-BV	USB-B	Gniazdo USB kątowe
J5	DS18B20	NS25-W3	Connector
J11	Debug	Goldpin 1x6	
J9	ISP Atmel	IDC6MLP	
J6	JTAG	IDC10MLP	
J3		ARK3 + KAmodWS2812-8	
J7 J8	MLX906 (modMLX906 Arduino - KAMAMI)	SIP4	Connector
S2	MODE	SW5.7	Mswitch
J1	PC-GK2.1	Gniado Jack	

Modul Wi-Fi:

R1 R2 R3	560R	1206	Rezystor
R4 R5	10k	1206	Rezystor
C1 C2	22pF	1206	Kondensator Ceramiczny
C3 C4	10uF	1206	

U1	SC16IS760IPW	SSOP-24	USART (16C550) via SPI/IIC 64B FIFO
U2	SPX1117MP-3.3	SOT-223	
D2	OK	1206	Dioda LED Zielona
D3	Data	1206	Dioda LED Zolta
D4	Err	1206	Dioda led czerwona

Q1	11,0592MHz	KWARC
J1	ESP-01 - moduł WiFi z ESP8266 i anteną	

VNC2 Debug module:

R1	100R	1206	Rezystor
R4	1k	1206	Rezystor
R2 R3 R5	10k	1206	Rezystor

C1 C3	100nF	1206	Kondensator Ceramiczny
C2	10uF	1206	Kondensator Elektrolityczny

U1	74LVC125AD	SO-14	QUAD BUS BUF 3SO
U2	FT232RL	SSOP-28	
D1	Tx/Rx	1206	Dioda LED Zielona

J11	Debug	Gniazdo goldpin 1x6
J4	USBB-BV	Gniazdo USB kątowe