W ofercie AVT*
AVT6054

Najważniejsze parametry:
- napięcie zasilania: 5 V (DC),
- maksymalny prąd obciążenia (bez podłączonych diod LED): 70 mA,
- zakres mierzonych temperatur: 0...125°C,
- rozdzielczość pomiaru temperatury: 1°C,
- dokładność pomiaru temperatury: ±0,5°C,
- zakres ustawień temperatur: 0...180°C,
- rozdzielczość ustawień temperatur: 10°C,
- zgodność ze standardami (nazw własnych użyto wyłącznie w celu identyfikacji produktu): Asus Aura Sync, Gigabyte RGB Fusion Ready, MSI Mystic Light Sync, ASRock Polychrome Sync.

Dodatkowe materiały do pobrania ze strony www.ulubionykiosk.pl/media

AVT5784	Wolnozmienny sterownik taśmy RGB (EP 8/2020)
AVT5778	Sterownik taśm LED RGB+W zgodny z HomeKit (EP 7/2020)
AVT5596	Mieszacz kolorów RGB (EP 7/2017)

* **Uwaga!** Elektroniczne zestawy do samodzielnego montażu. Wymagana umiejętność lutowania! Podstawową wersją zestawu jest wersja **[B]** nazywana potocznie KIT-em (z ang. zestaw). Zestaw w wersji **[B]** zawiera elementy elektroniczne (w tym **[UK]** – jeśli występuje w projekcie), które należy samodzielnie wlutować w dołączoną płytkę drukowaną (PCB). Wykaz elementów znajduje się w dokumentacji, która jest podlinkowana w opisie kitu. Mając na uwadze różne potrzeby naszych klientów, oferujemy dodatkowe wersje:
- **wersja [C]** – zmontowany, uruchomiony i przetestowany zestaw **[B]** (elementy wlutowane w płytkę PCB),
- **wersja [A]** – płytka drukowana bez elementów i dokumentacji.

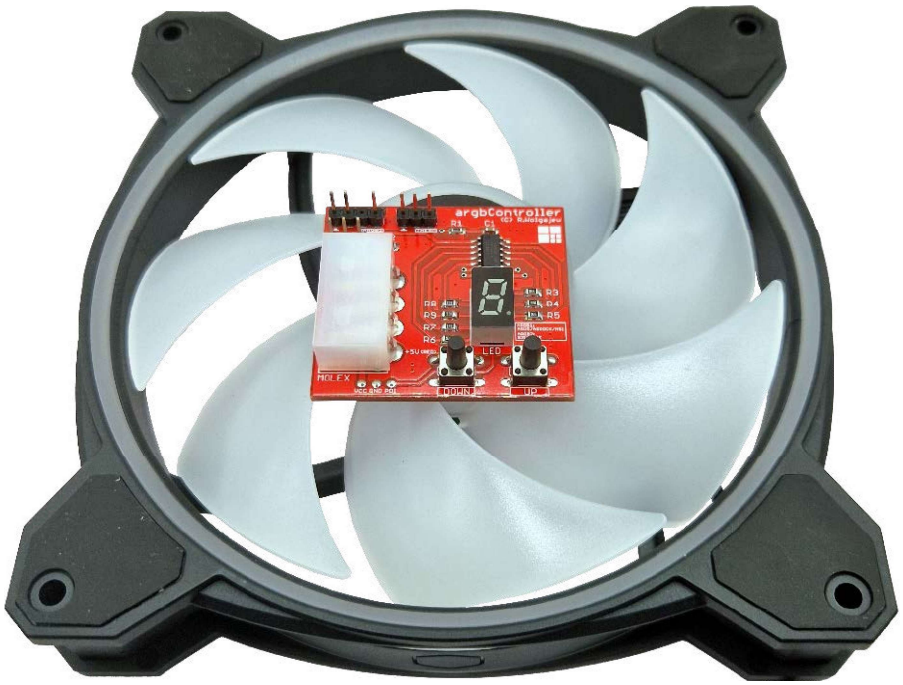
Kity, w których występuje układ scalony wymagający zaprogramowania, mają następujące dodatkowe wersje:
- **wersja [A+]** – płytka drukowana **[A]** + zaprogramowany układ **[UK]** i dokumentacja,
- **wersja [UK]** – zaprogramowany układ.

Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik PDF! Podczas składania zamówienia upewnij się, którą wersję zamawiasz! **http://sklep.avt.pl**

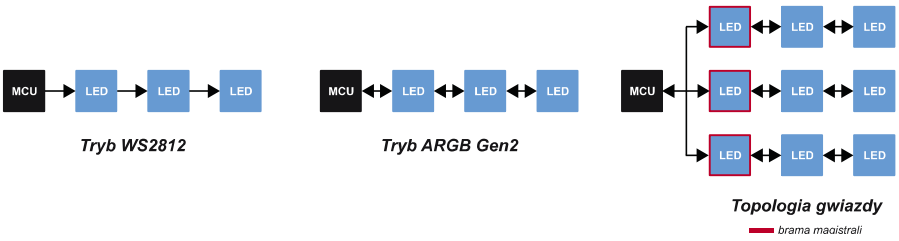
W przypadku braku dostępności na stronie sklepu osoby zainteresowane zakupem płytek drukowanych (PCB) prosimy o kontakt via e-mail: **kity@avt.pl**.

argbController (1)

Dzisiejsze komputery klasy PC już dawno przestały być li tylko bezdusznymi maszynami liczącymi, używanymi do nauki czy rozrywki, a stały się elementami wyposażenia wnętrz. W tej branży funkcjonalność musi iść w parze z estetyką (wszak chcemy otaczać się przedmiotami pięknymi). Tę naturalną kolej rzeczy zauważyli już dawno producenci sprzętu komputerowego – właśnie z tego względu coraz nowsze jednostki wnoszą wzornictwo przemysłowe na jeszcze wyższy poziom. Jednym z istotnych elementów stylistyki współczesnych urządzeń okazują się wszelkiego rodzaju elementy świetlne, które w dobie oświetlenia LED mogą przybierać bardzo wysmakowane formy.



Dokładnie w ten trend wpisuje się oświetlenie LED stosowane w komputerach PC, w przypadku którego funkcjonalność podporządkowana jest niejednokrotnie niebanalnej formie. Producenci urządzeń stosują do realizacji założonych celów projektowych nowoczesne elementy LED w postaci adresowalnych diod (obecnych na rynku od dobrych paru lat), dzięki czemu mogą uzyskać skomplikowane sceny świetlne. Trafny przykład stanowią tu wszelkiego rodzaju wentylatory przeznaczone do różnych podzespołów PC: jeszcze nie tak dawno służyły wyłącznie chłodzeniu – natomiast dzisiaj stanowią element co najmniej estetyczny, którego jednym (i nie mniej istotnym) z zadań jest generowanie sceny świetlnej. Ale nie tylko wentylatory wyposaża się w tego rodzaju funkcjonalność. Takie samo zjawisko stało się udziałem klawiatur, myszy, monitorów czy nawet obudów PC lub podkładek(!) pod mysz. I tutaj pojawia się problem. Mamy wiele urządzeń w jednym zestawie, z których każde może generować podświetlenie w sobie tylko znany sposób, przez co efekt finalny nie musi być zgodny z oczekiwaniami. Jak nad tym zapanować? Jak to wszystko zsynchronizować? Jak tym globalnie sterować? Miłośnikom nowoczesnego sprzętu przyszli z pomocą producenci płyt głównych, którzy zaczęli stosować zarówno specjalne złącza do sterowania oświetleniem LED, jak i niezbędne oprogramowanie. Początkowo były to złącza RGB, nieoferujące wprawdzie możliwości indywidualnego sterowania poszczególnymi peryferiami wyposażonymi w podświetlenie LED, lecz przynajmniej pozwalające na zastosowanie tego samego ustawienia dla wszystkich podzespołów zamontowanych w obudowie. W końcu pojawił się „jakiś" standard cyfrowy (używam słowa „jakiś", gdyż – po pierwsze – nie jest to standard otwarty, a po drugie – nie jest on wspierany globalnie przez wszystkich producentów z opisywanej branży). Mowa o podświetleniu LED standardu ARGB Gen2, którego złącza znajdziemy na płytach głównych takich producentów, jak ASUS, ASROCK, MSI czy GIGABYTE; co ciekawe: złącza różnego typu. Z lektury dostępnej w Internecie można wywnioskować, że największym promotorem wspomnianego standardu jest firma Cooler Master, produkująca systemy chłodzenia komputerów klasy PC oraz niezbędne sterowniki (w tym oświetlenia). Ale tą drogą poszły również inne firmy, więc uznajmy, że ARGB Gen2 stał się w świecie oświetlenia PC standardem – jednak, o czym wspomniałem na wstępie, standardem zamkniętym (i to zamkniętym na tyle skutecznie, że w Internecie znajdziecie niewiele informacji z tego zakresu).



Rysunek 1. Graficzna prezentacja dostępnych trybów pracy diod ARGB Gen2

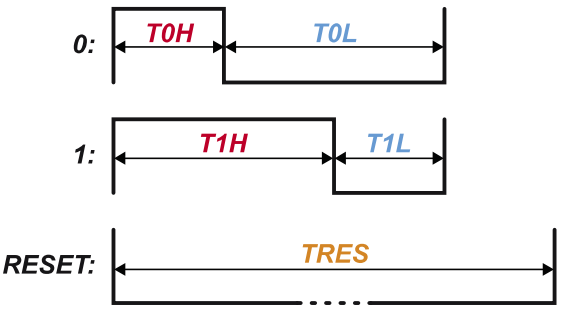
Jedną z bardziej użytecznych dokumentacji, na które możemy się natknąć, jest ta dotycząca adresowalnej diody LED typu SK6112-RG-002 produkcji dalekowschodniej firmy pod (nic niemówiącą Europejczykowi) nazwą DONGGUAN OPSCO OPTOELECTRONICS CO., LTD, dzięki której możemy zorientować się, z jakim rozwiązaniem mamy do czynienia. Na szczęście – jak zwykle bywa w podobnych sytuacjach – znalazł się pasjonat, który dzięki inżynierii wstecznej sterownika Coolermaster A1, jak i skąpej dokumentacji diody LED, opracował oraz opublikował na GitHubie (pod nickiem cpldcpu) szczegóły protokołu komunikacyjnego, zwalniając mnie z tej żmudnej pracy. Trzeba mieć jednak na uwadze, że wspomniane opracowanie nie musi być zgodne ze szczegółową dokumentacją producenta (i – na dobrą sprawę – prawdziwe), gdyż powstało jedynie na drodze eksperymentalnej. Tym niemniej jego autor (de facto bazujący na chińskich opracowaniach) sprawdził swoją pracę pod kątem użyteczności z elementami tego standardu, więc możemy przyjąć uzyskane przez niego wyniki jako bazę do dalszego działania. Warto zauważyć, że część spośród kluczowych mechanizmów obsługi komunikacji opisywała (na szczęście) dokumentacja diody SK6112-RG-002.

Zacznijmy od początku. Elementy LED standardu ARGB Gen2 są adresowalnymi diodami LED, zgodnymi z dobrze znanym standardem diod WS2812 i podobnych. Znaczy to ni mniej, ni więcej tyle, że sterowane protokołem diody WS2812 będą zachowywały się zgodnie ze swoim pierwowzorem. Jednak diody ARGB Gen2 udostępniają bardzo ciekawe rozszerzenia protokołu WS2812, dzięki czemu możemy nimi sterować w jeszcze bardziej elastyczny sposób. Wystarczy wspomnieć, że każdy element ARGB Gen2 możemy zarówno indywidualnie konfigurować, gdyż ma on specjalny rejestr modyfikujący pewne ustawienia sprzętowe – jak i odczytywać niektóre dane w nim zapisane (tak naprawdę w zasadzie tylko 2). Tak, tak, nie przesłyszeliście się, interfejs komunikacyjny diod ARGB Gen2 pozwala na dwukierunkową komunikację! Co więcej, protokół komunikacyjny tychże elementów LED wyposażono w możliwość pracy wielu łańcuchów diod LED połączonych w topologii gwiazdy, sterowanych – UWAGA – **niezależnie**, przez ten sam mikrokontroler. Graficzną prezentację dostępnych trybów pracy diod ARGB Gen2 pokazano na **rysunku 1**.

Podstawowy tryb pracy to **tryb zgodny ze standardem WS2812**. W tym przypadku mamy do czynienia z asynchronicznym interfejsem komunikacyjnym, w którym nie ma wyprowadzenia sygnału zegarowego, w związku z czym dane przesyłane przy jego użyciu muszą być w pewien sposób zakodowane, by możliwe stało się ich proste zdekodowanie i by zyskały one odporność na zakłócenia czy artefakty. Zastosowano tutaj mechanizmy dobrze znane z interfejsów bezprzewodowej transmisji danych stosowanych w torach podczerwieni, gdzie stany logiczne „1” i „0” zakodowane zostały długością impulsu. Dodatkowo wprowadzono tak zwany sygnał RESET (również zakodowany długością impulsu), który powoduje zresetowanie interfejsów komunikacyjnych sterowników diod LED i ich oczekiwanie na nowe dane. Na **rysunku 2** pokazano przebiegi sygnałów interfejsu komunikacyjnego diody SK6112-RG-002 w trakcie transmisji bitu logicznej „1”, logicznego „0” i sygnału RESET.

Co ważne, pojedyncze bity danych zgrupowane w bajty przesyłane są w kolejności od bitu najstarszego (MSB) do najmłodszego (LSB), a każda dioda LED w łańcuchu oczekuje na 3 bajty danych odpowiedzialnych za składowe jej koloru – przesyłane w kolejności G, R, B. Ale skąd każda z diod w łańcuchu „wie”, które z przesyłanych danych użytecznych przeznaczone są właśnie dla niej, a nie dla innej? Zrealizowano to w bardzo prosty, acz skuteczny sposób. Każda z diod LED w łańcuchu po włączeniu zasilania (jak również po odebraniu sygnału RESET) oczekuje na 3 bajty danych przeznaczonych wyłącznie dla niej. Do tego czasu jej wyjściowy interfejs komunikacyjny (wyjście DOUT) jest „nieprzezroczysty” dla nadchodzących danych (a ściślej rzecz biorąc, jest „przezroczysty” wyłącznie dla sygnału RESET oraz innych trybów pracy). Po odebraniu wspomnianych 3 bajtów danych dioda ta staje się „przezroczysta” dla kolejnych nadchodzących danych, co znaczy, że retransmituje do kolejnych diod w łańcuchu. Biorąc pod uwagę, że dokładnie tak samo zachowuje się każda dioda w łańcuchu, dość szybko zdamy sobie sprawę, że kolejne dane użyteczne przesyłane przez tak skonstruowany interfejs komunikacyjny trafiają kolejno do następujących po sobie (w sensie elektrycznym) diod w łańcuchu. Tyle w kwestii trybu zgodnego z WS2812. Żadnego novum, ale najciekawsze nadejdzie za chwilę. Zastanówmy się: skoro diody ARGB Gen2 są podstawowo zgodne z WS2812 i realizują ich protokół komunikacyjny, to w jaki sposób „wymusić” na nich przejście do innych trybów pracy? Ano w podobny sposób, w jaki zakodowane zostały stany logiczne, czyli długością impulsu, a w zasadzie sekwencją impulsów o predefiniowanej i odpowiednio dobranej długości – tak by nie zostały zidentyfikowane przez diodę (diody) jako bity informacji. A zatem, by wejść w **tryb konfiguracji** diod ARGB Gen2, należy na magistralę danych przesłać sekwencję impulsów pokazaną na **rysunku 3**.

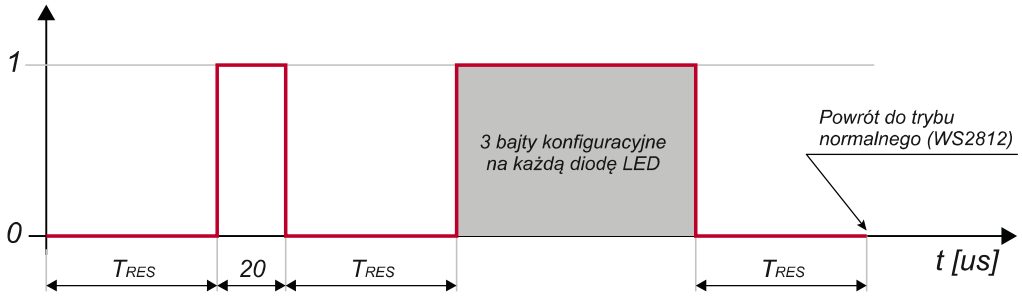
Co ważne, powyższa sekwencja retransmitowana jest do kolejnych diod w łańcuchu (są dla niej niejako „przezroczyste”), przez co – po jej wysłaniu przez sterownik – wszystkie diody w tymże łańcuchu przechodzą w tryb konfiguracji. Następnie, po odczekaniu czasu sygnału RESET, sterownik powinien wysłać 3 bajty danych na każdą diodę LED, tak jak to miało miejsce w przypadku standardu WS2812, przy czym wspomniane 3 bajty danych nie zostaną przez każdą z nich zinterpretowane jako wartości RGB, tylko jako bajty konfiguracyjne. Ich znaczenie dla konfiguracji elementu LED, określone empirycznie, zebrano w **tabeli 1**.



[us]	min.	typ.	max.
T0H	0.2	0.3	0.4
T0L	0.8	0.9	
T1H	0.65	0.9	1.0
T1L	0.2	0.3	
TRES	250		

Rysunek 2. Przebiegi sygnałów interfejsu komunikacyjnego diody SK6112-RG-002 w trakcie transmisji bitu logicznej „1”, logicznego „0” i sygnału RESET

Przejdźmy zatem do szczegółów. Pole pwmFreq odpowiada za częstotliwość sygnału PWM kontrolera diody ARGB Gen2 sterującego poszczególnymi strukturami LED RGB według poniższej specyfikacji: 0 → 1,178 kHz, 1 → 2,351 kHz, 2 → 9,431 kHz, 3 → 18,87 kHz. Ustawienie to wpływa na kompromis pomiędzy zjawiskiem migotania elementu LED a zakłóceniami EMI, jakie może generować.



Rysunek 3. Sekwencja impulsów interfejsu komunikacyjnego diod ARGB Gen2 odpowiedzialna za wprowadzenie tych diod w tryb konfiguracji

Pole respType wpływa z kolei na rodzaj odpowiedzi, jaka jest wysyłana przez diodę ARGB Gen2 w trybie odczytu, o którym później. Jeśli ten bit ustawimy na wartość „1”, to dioda RGB Gen2 w trybie odczytu wyśle swój bit identyfikacyjny ID, przy czym wartość tego bitu zakodowana jest długością impulsu odpowiedzi według poniższej specyfikacji: impuls LO (10 µs) → bit „0”, impuls HI (40 µs) → bit „1”. Wartość bitu ID ustalana jest losowo na etapie produkcji, zaś jego obecność umożliwia fizyczną identyfikację łańcuchów LED w topologii gwiazdy. Z kolei, jeśli bit respType ustawimy na wartość „0”, to dioda ARGB Gen2 w trybie odczytu wyśle nam ustawienie sumarycznego prądu elementu LED, przy czym wartość tego prądu zakodowana jest, podobnie jak poprzednio, długością impulsu odpowiedzi według poniższej specyfikacji: impuls LO (10 µs) → 5 mA, impuls MID (20 µs) → 12 mA, impuls HI (40 µs) → 20 mA.

Pola **fineG**, **fineR** i **fineB** odpowiadają za dokładne dostrojenie prądu poszczególnych struktur R, G, B elementu LED (a więc także wzajemnych relacji pomiędzy nimi), dając możliwość korekty ustawień kolorów przezeń wyświetlanych. Maksymalne ustawienie, równe w tym przypadku 31, powoduje podwojenie prądu danej struktury LED.

Pola **ovCurr** odpowiadają za sumaryczny prąd diody ARGB Gen2. Ustawienie wszystkich pól ovCurr na wartość „1” powoduje zmniejszenie o połowę sumarycznego prądu tejże diody. Sprawia również, że w trybie odczytu ustawienia prądu takiej diody zmieniają się z wartości domyślnej MID (12 mA) na wartość LO (5 mA). Wszystkie inne kombinacje ustawień pól ovCurr nie przynoszą żadnego efektu, przy czym możliwe jest, że były to ustawienia charakterystyczne dla konkretnego elementu LED, gdyż trudno sensownie wytłumaczyć obecność 3 bitów konfiguracyjnych (a więc 8 możliwych ustawień) dla wprowadzenia 2 możliwych stanów pracy, niewyjaśniona pozostaje ponadto kwestia sposobu na przestawienie tego prądu na wartość HI (20 mA).

Kolejnym, ciekawym rozszerzeniem funkcjonalności diod WS2812 w przypadku standardu ARGB Gen2 jest **tryb odczytu**, który umożliwia odczyt dwóch ustawień każdego z elementów LED w łańcuchu. Jest to bit identyfikacyjny ID lub wartość prądu diody. Tak jak wspomniano wcześniej: fakt, którą z wartości dioda LED wystawiać będzie na port komunikacyjny (jak zawsze kodując ją długością impulsu), zależeć będzie od wcześniejszej konfiguracji. Niemniej jednak, aby wejść w tryb odczytu, podobnie jak to miało miejsce w trybie konfiguracji, musimy zainicjować go odpowiednią sekwencją impulsów sterujących – pokazaną na **rysunku 4**.

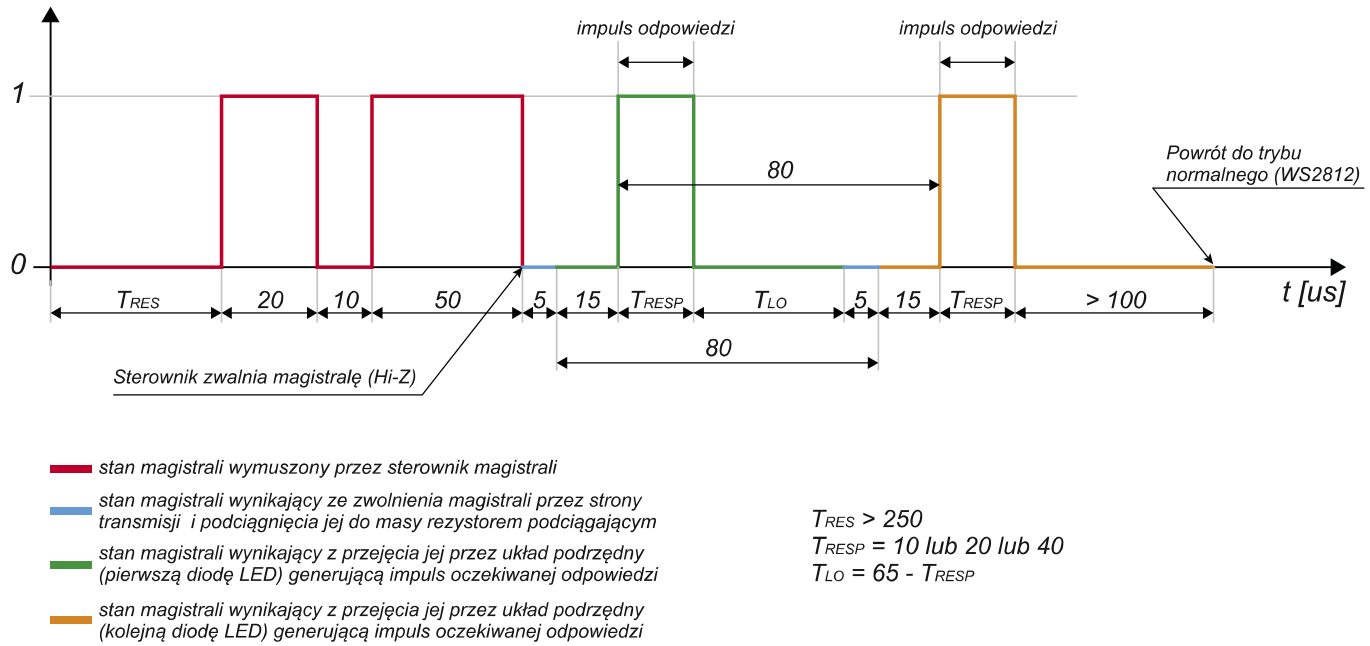
Tabela 1. Opis i znaczenie bajtów konfiguracyjnych diod ARGB Gen2

Bity	7	6	5	4	3	2	1	0
Bajt 1	pwmFreq			fineG				
Bajt 2	ovCurr	ovCurr	respType	fineR				
Bajt 3	ovCurr			fineB				

Po wyemitowaniu powyższej sekwencji sygnałów sterownik powinien zwolnić magistralę danych, ustawiając swój port komunikacyjny w tryb wejścia (Hi-Z) i – po odczekaniu 5 µs – rozpocząć odczytywanie impulsów sterujących, które z kolei powinny być wysyłane przez kolejne diody LED w łańcuchu. Dokładnie w tym samym czasie, gdy sterownik zwalnia magistralę danych (czyli po ostatnim zboczu opadającym sekwencji sygnałów z rysunku 4), pierwsza dioda LED w szeregu wykonuje jednocześnie dwie czynności:

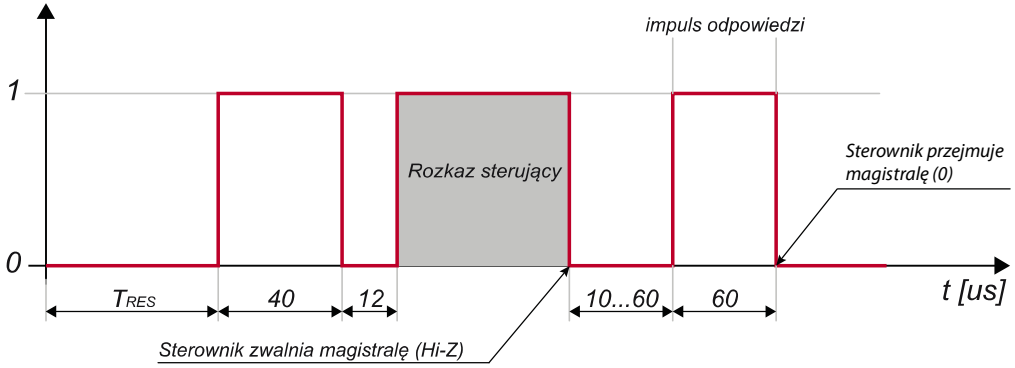
- retransmituje otrzymaną sekwencję impulsów sterujących odpowiedzialną za przejście do trybu odczytu na swoje wyjście (DOUT), przekazując ją tym samym kolejnej diodzie w łańcuchu, po czym, po upływie 5 µs, ustawia wspomniane wyjście (DOUT) w tryb wejścia (Hi-Z), by być gotową na retransmisję odpowiedzi kolejnej diody w łańcuchu na swoje wejście (DIN) ustawione w tryb wyjścia,
- po upływie 5 µs ustawia wejście własnego interfejsu komunikacyjnego (DIN) w tryb wyjścia, by po kolejnych 10 µs przesłać impuls dodatni, którego długość zależy od typu oczekiwanej odpowiedzi (o czym pisałem wcześniej, omawiając znaczenie pola respType).

Po przesłaniu impulsu – jak wyżej – i odczekaniu pewnego czasu (wynikającego z okresu retransmisji), pierwsza z diod w łańcuchu przesyła na swoje wejście (DIN, nadal znajdujące się w trybie wyjścia) impuls dodatni, będący odpowiedzią drugiej diody w łańcuchu. Jako że powyższy scenariusz powtarza każdorazowo każda z diod w łańcuchu, to na wejściu pierwszej z diod (DIN, ustawionym nadal jako wyjście) otrzymujemy szereg impulsów stanowiących odpowiedzi kolejnych diod w łańcuchu, przy czym czas pomiędzy kolejnymi odpowiedziami wynosi 80 µs. Po retransmisji wszystkich odpowiedzi i braku aktywności na magistrali danych (>100 µs) każda z diod wychodzi z trybu odczytu, przyjmując domyślne ustawienia kierunków swojego interfejsu komunikacyjnego (pinów DIN i DOUT). Łatwo zauważyć, że dzięki tak wymyślnemu mechanizmowi – oprócz możliwości odczytania interesujących nas wartości odpowiedzi każdej z diod LED w łańcuchu – dostajemy również możliwość **określenia liczby diod znajdujących się na magistrali** danych. A teraz wisienka na torcie: **tryb pracy w topologii gwiazdy**, nazywany również **multi-layer**. W tym trybie możliwe jest podłączenie wielu łańcuchów diod ARGB Gen2 do jednego sterownika (w topologii gwiazdy), ich niezależna konfiguracja, adresowanie oraz odczyt. Zrealizowano to w bardzo ciekawy i nietuzinkowy sposób. W przypadku pracy wielu łańcuchów diod LED – podłączonych w tym samym punkcie do wyjść sterownika magistrali (np. mikrokontrolera) – każda z pierwszych diod w łańcuchu przejmuje rolę bramki (gate), decydującej o ważności danych przesyłanych do łańcucha, na którym się znajduje – lub z niego odczytywanych (wszak mamy przecież do dyspozycji także tryb odczytu). Aby jednak możliwa stała się taka selektywna wymiana danych pomiędzy sterownikiem a wybranym



Rysunek 4. Sekwencja impulsów interfejsu komunikacyjnego diod ARGB Gen2 odpowiedzialna za wprowadzenie tych diod w tryb odczytu

łańcuchem LED, niezbędne są mechanizmy adresujące, a dotychczasowe tryby pracy takowych nie udostępniają. Jak więc rozwiązano ten problem? Wprowadzono dodatkowy tryb pracy, nazywany **trybem rozkazów sterujących**. Ale jak wejść w ten tryb? Podobnie jak to miało miejsce w trybie konfiguracji, musimy zainicjować go odpowiednią sekwencją impulsów sterujących, którą pokazano na **rysunku 5**.



Rysunek 5. Sekwencja impulsów interfejsu komunikacyjnego diod ARGB Gen2 odpowiedzialna za wprowadzenie tychże diod w tryb rozkazów sterujących

Co ważne i oczywiste, powyższa sekwencja sygnałów odbierana jest jednocześnie przez pierwsze diody w poszczególnych łańcuchach, przez co – po jej wysłaniu przez sterownik – każda ze wspomnianych diod przejmuje rolę bramki. Wynika z tego, że tym razem nie jest ona również dostępna dla pozostałych diod w danym łańcuchu – z wyjątkiem wspomnianej bramki. W dalszym kroku, po wyemitowaniu powyższej sekwencji, należy wysłać jeden bajt danych, stanowiący rozkaz sterujący (z opcjonalnym adresem umieszczonym na 4 najmłodszych bitach) – przy czym sposób jego transmisji jest identyczny, jak to miało miejsce w przypadku standardu WS2812. Dalej, w przypadku wybranych rozkazów sterujących, oczekiwana może być odpowiedź bramki, która polega na wygenerowaniu przez nią dodatniego impulsu (60 µs) na magistrali danych, w czasie od 10 µs do 60 µs, licząc od końca transmisji rozkazu. Takie działanie sprawia, że – podobnie jak to miało miejsce w trybie odczytu – po wyemitowaniu powyższej sekwencji sygnałów sterownik powinien zwolnić magistralę danych, ustawiając swój port komunikacyjny w tryb wejścia (Hi-Z). Następnie musi rozpocząć odczytywanie impulsu odpowiedzi, który z kolei powinien być wysyłany przez bramkę (pamiętajmy: tylko dla wybranych rozkazów). Listę dostępnych rozkazów wraz z opisem ich znaczenia dla protokołu ARGB Gen2 pokazano w **tabeli 2**.

Znamy już listę rozkazów, w związku z czym zastanówmy się, jak powinna wyglądać procedura inicjalizująca opisany system w celu jego dalszej obsługi. Jak łatwo się domyślić, należy rozpocząć od odszukania wszystkich łańcuchów LED i nadania im unikalnych adresów. Procedura taka powinna składać się z następujących kroków:

- wysłania rozkazu RESET_ADDRESS (0x20), by zresetować potencjalne, wcześniejsze przydzielenie adresów dla łańcuchów LED,
- wysłania rozkazu ASSIGN_ADDRESS (0x10) z adresem 0x01 w celu przydzielenia go pierwszemu, przypadkowo wybranemu łańcuchowi LED,
- odczytania odpowiedzi łańcucha LED (w zasadzie jego bramki) w celu potwierdzenia skutecznego przydzielenia adresu 0x01.

Operację powyższą powtarzamy dla wszystkich obecnych na magistrali łańcuchów LED połączonych w topologii gwiazdy, przez co – po jej wykonaniu – każdy z tychże łańcuchów (maksymalnie 15) otrzyma unikalny adres. Ale jak działa ten mechanizm z punktu widzenia każdego z łańcuchów LED? Otóż: każdy z nich (a w zasadzie jego bramka) po otrzymaniu

Tabela 2. Lista dostępnych rozkazów protokołu ARGB Gen2 wraz z opisem ich znaczenia

Numer rozkazu	Nazwa	Opis
0x10	ASSIGN_ADDRESS	Przydziela adres przypadkowo wybranemu łańcuchowi LED (zgodnie z procedurą arbitrażu). Dopuszczalny zakres adresów (wartość adresu zastępuje 4 najmłodsze bity rozkazu) to 0x01 do 0x0F. Oczekiwana odpowiedź potwierdza przydzielenie adresu wybranemu łańcuchowi LED. Wartość adresu równa 0x00 również powoduje wysłanie odpowiedzi, lecz żaden adres nie jest przydzielany.
0x20	RESET_ADDRESS	Kasuje poprzednie przydzielenie adresu wybranemu (0x01 do 0x0F) lub wszystkim (0x00) łańcuchom LED. Funkcja nie zgłasza odpowiedzi.
0x30	PING_ADDRESS	Potwierdza obecność wybranego i wcześniej zaadresowanego (0x01 do 0x0F) łańcucha LED. Oczekiwana odpowiedź potwierdza obecność zaadresowanego wcześniej łańcucha LED na magistrali. Jeśli przesłana wartość adresu równa jest 0x00, to oczekiwana odpowiedź odebrana zostanie wyłącznie wtedy, gdy wcześniejsze przydzielenie adresu wykonano z użyciem adresu 0x00.
0x40	ACTIVATE_ADDRESS	Aktywuje wybrany (0x01 do 0x0F) i wcześniej zaadresowany łańcuch LED. Po wykonaniu tego rozkazu wszelkie transmisje (w tym dla trybu odczytu) na magistrali danych, niezależnie od ich charakteru, będą odbywać się wyłącznie między sterownikiem a wybranym łańcuchem LED – aż do zmiany ustawień poprzez wysłanie innego rozkazu sterującego. Co ważne, aktywacja łańcucha LED dezaktywuje każdy inny, który był dotychczas aktywny. Funkcja nie zgłasza odpowiedzi.

takiego rozkazu odczekuje przypadkowy czas z zakresu pomiędzy 10 µs a 60 µs, testując nieustannie stan magistrali. Jeśli magistrala nadal pozostaje w stanie niskim, oznacza to, że żaden inny łańcuch nie przejął jeszcze tego adresu, w związku z czym bieżący łańcuch (a w zasadzie jego bramka) wysyła impuls odpowiedzi (stan wysoki o czasie trwania 60 µs), przyjmując wysłany adres jako swój. Taka procedura arbitrażu umożliwia bezkolizyjne przydzielenie adresu wszystkim łańcuchom LED na magistrali. Co oczywiste, żaden już zaadresowany łańcuch nie bierze udziału w tym mechanizmie, przez co możliwość potencjalnej kolizji znacznie się zmniejsza. Dodatkowo, jeśli mamy taką potrzebę, obecność zaadresowanego łańcucha LED na magistrali danych możemy sprawdzić poniższą sekwencją rozkazów:

- wysyłamy rozkaz PING_ADDRESS (0x30) z adresem 0x01 w celu sprawdzenia obecności łańcucha o adresie 0x01 na magistrali danych,
- odczytujemy impuls odpowiedzi potwierdzający wspomnianą obecność.

Ponadto możemy aktywować wybrany łańcuch LED rozkazem ACTIVATE_ADDRESS (0x40) z przykładowym adresem 0x01, a następnie przesłać do niego chociażby zestaw danych RGB w celu organoleptycznej weryfikacji udanego przydzielenia adresu. Co ważne, w jednym czasie aktywny może być wyłącznie jeden łańcuch LED, w związku z czym aktywacja kolejnego łańcucha LED dezaktywuje każdy inny, który był dotychczas aktywny. Jak widzicie, zastosowany mechanizm adresacji i następnej aktywacji wybranego łańcucha LED pozwala na selektywne oraz niezależne sterowanie każdym z nich, pracującym w topologii gwiazdy. Przyznacie, że jest to bardzo nowatorskie i ciekawe rozwiązanie, znacznie poszerzające funkcjonalność typowego interfejsu WS2812. Uważny Czytelnik dostrzeże prawdopodobnie pewną niedoskonałość takiego rozwiązania, a mianowicie brak możliwości fizycznej identyfikacji wybranego łańcucha LED bez organoleptycznej weryfikacji. Możemy wszak zaadresować je indywidualnie, lecz tak naprawdę nie jesteśmy w stanie stwierdzić z wyprzedzeniem, który z adresów zostanie przypisany konkretnemu (w sensie fizycznym) łańcuchowi. Na szczęście możemy się ratować jedną z możliwości interfejsu – a mianowicie możliwością odczytu pola ID każdej z diod w łańcuchu, które to pole konfigurowane jest wartością losową w procesie produkcji. Po przeprowadzeniu odczytu wszystkich pól ID diod LED danego łańcucha LED dysponujemy wtedy n-bitowym numerem ID tegoż łańcucha. Co prawda istnieje pewne prawdopodobieństwo, że dwa łańcuchy LED (zwłaszcza złożone z niewielkiej liczby diod) dysponować będą identycznym n-bitowym numerem ID, lecz wydaje się ono akceptowalne. Opisana powyżej procedura powinna składać się z następujących kroków:

- wysłania słowa konfiguracyjnego 0x002000 do wszystkich diod LED w wybranym łańcuchu LED (wcześniej aktywowanym rozkazem 0x40) – w celu konfiguracji rodzaju odpowiedzi wysyłanej przez diody z tegoż łańcucha (respType = 1),
- uruchomienia trybu odczytu i następującego po nim odczytania pól ID wszystkich diod LED łańcucha,
- połączenia otrzymanych wartości pól ID w celu określenia n-bitowego numeru ID.

W powyższy sposób możemy powiązać przydzielony (w procesie arbitrażu) logiczny adres łańcucha z jego adresem fizycznym, otrzymanym poprzez połączenie pól ID, ale i tutaj mamy pewien problem. Wynika on z faktu, że element LED w żaden sposób nie jest fizycznie identyfikowany (np. w postaci nadrukowanego opisu) przydzieloną na etapie produkcji wartością pola ID, przez co – w moim przekonaniu – identyfikacja, jak powyżej, niewiele zmienia. Tak czy inaczej, zarówno autor wspomnianego na początku opracowania, jak i ja na etapie prób nie znaleźliśmy innej, stuprocentowo pewnej drogi do fizycznej identyfikacji łańcuchów LED (a więc i urządzeń w nie wyposażonych). Zapewne wynika to z braków w dostępie do dokumentacji producenta lub, co też możliwe, nie było niezbędne w sensie wymagań protokołu. Na tym zakończę (mam nadzieję – interesujący) wywód dotyczący bardzo ciekawych komponentów, jakimi niewątpliwie są diody ARGB Gen2.

Za miesiąc przyjrzymy się szczegółom praktycznej implementacji opisanego powyżej protokołu na bazie bardzo prostego sterownika mikroprocesorowego. Co ważne, jako że diody ARGB Gen2 wymagają dość restrykcyjnych zależności czasowych (podobnie zresztą, jak diody WS2812), prezentowane w kolejnej części artykułu rozwiązanie zakłada dość wysoką częstotliwość taktowania mikrokontrolera, równą 20 MHz. Należy uwzględnić to ograniczenie przy przenoszeniu opisanych rozwiązań na inne platformy.

Robert Wołgajew, EP

W ofercie AVT*
AVT6054

Najważniejsze parametry:
- napięcie zasilania: 5 V (DC),
- maksymalny prąd obciążenia (bez podłączonych diod LED): 70 mA,
- zakres mierzonych temperatur: 0...125°C,
- rozdzielczość pomiaru temperatury: 1°C,
- dokładność pomiaru temperatury: ±0,5°C,
- zakres ustawień temperatur: 0...180°C,
- rozdzielczość ustawień temperatur: 10°C,
- zgodność ze standardami (nazw własnych użyto wyłącznie w celu identyfikacji produktu): Asus Aura Sync, Gigabyte RGB Fusion Ready, MSI Mystic Light Sync, ASRock Polychrome Sync.

Dodatkowe materiały do pobrania ze strony www.ulubionykiosk.pl/media

AVT6054	argbController (1) (EP 9/2024)
AVT5784	Wolnozmienny sterownik taśmy RGB (EP 8/2020)
AVT5778	Sterownik taśm LED RGB+W zgodny z HomeKit (EP 7/2020)
AVT5596	Mieszacz kolorów RGB (EP 7/2017)

* **Uwaga!** Elektroniczne zestawy do samodzielnego montażu. Wymagana umiejętność lutowania! Podstawową wersją zestawu jest wersja **[B]** nazywana potocznie KIT-em (z ang. zestaw). Zestaw w wersji **[B]** zawiera elementy elektroniczne (w tym **[UK]** – jeśli występuje w projekcie), które należy samodzielnie wlutować w dołączoną płytkę drukowaną (PCB). Wykaz elementów znajduje się w dokumentacji, która jest podlinkowana w opisie kitu. Mając na uwadze różne potrzeby naszych klientów, oferujemy dodatkowe wersje:
- **wersja [C]** – zmontowany, uruchomiony i przetestowany zestaw **[B]** (elementy wlutowane w płytkę PCB),
- **wersja [A]** – płytka drukowana bez elementów i dokumentacji.

Kity, w których występuje układ scalony wymagający zaprogramowania, mają następujące dodatkowe wersje:
- **wersja [A+]** – płytka drukowana **[A]** + zaprogramowany układ **[UK]** i dokumentacja,
- **wersja [UK]** – zaprogramowany układ.

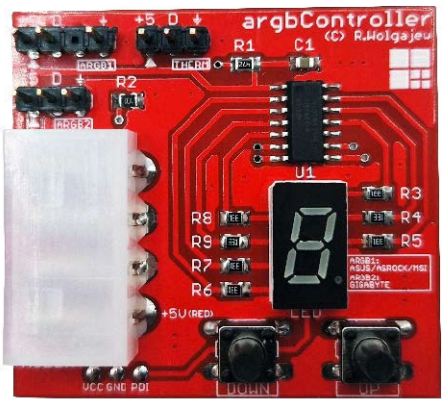
Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik PDF! Podczas składania zamówienia upewnij się, którą wersję zamawiasz! **http://sklep.avt.pl**

W przypadku braku dostępności na stronie sklepu osoby zainteresowane zakupem płytek drukowanych (PCB) prosimy o kontakt via e-mail: **kity@avt.pl**.

argbController (2)

Pierwsza część artykułu znajduje się pod adresem: **https://ulubionykiosk.pl/media**

W poprzedniej części artykułu opisaliśmy szczegóły techniczne nieco tajemniczego protokołu ARGB Gen2. Tym razem przechodzimy do kwestii praktycznych – prezentujemy zarówno oprogramowanie sterujące łańcuchami LED zgodnymi z omawianym standardem transmisji, jak i konstrukcję prostego urządzenia mikroprocesorowego implementującego ów protokół.



Oprogramowanie mikrokontrolera

Plik nagłówkowy, pokazany na **listingu 1**, definiuje główne ustawienia sprzętowe, a także wprowadza niezbędne typy danych – w tym configType, który upraszcza późniejszą konfigurację diod LED. Dalej, na **listingu 2**, pokazano kod funkcji inicjalizacyjnej interfejsu komunikacyjnego, którego wyłącznym zadaniem jest ustawienie kierunku portu danych i jego stanu spoczynkowego (0).

Z kolei na **listingu 3** pokazano ciała funkcji odpowiedzialnych za przesłanie jednego bitu danych (0 lub 1), przy udziale wspomnianego wcześniej interfejsu komunikacyjnego. Warto zauważyć, że stosowne funkcje zdefiniowano, używając atrybutu `__always_inline__`, który instruuje kompilator, by ZAWSZE wstawiał w miejscu wywołania rozwinięcie funkcji, nie zaś skok do jej ciała. Jest to o tyle istotne, że operujemy na dość rygorystycznych timingach (rzędu setek nanosekund). W takiej sytuacji – przy częstotliwości taktowania mikrokontrolera równej 20 MHz (okres 50 ns) – jakiekolwiek niepotrzebne (z punktu widzenia programisty, nie kompilatora) wykonanie rozkazu asemblera może rozłożyć całą transmisję danych, całkowicie uniemożliwiając sterowanie wspomnianymi elementami.

Skoro mamy już funkcje umożliwiające przesłanie jednego bitu informacji, pora na funkcję (także odpowiednio zadeklarowaną), umożliwiającą przesłanie kompletnego bajtu danych za pomocą interfejsu ARGB Gen2 – patrz **listing 4**.

W dalszej kolejności, by uprościć zapis danych do adresowalnych diod LED RGB, przewidziano funkcję pozwalającą na przesłanie kompletnej informacji o kolorze tejże diody LED. Ciało tej funkcji pokazano na **listingu 5**.

```
//Definicja typu strukturalego przechowującego konfigurację diod LED
typedef struct
{
    uint8_t pwmFreq;
    uint8_t ovCurr;
    uint8_t respType;
    uint8_t fineR;
    uint8_t fineG;
    uint8_t fineB;
}configType;


//Definicje portu sterującego diodami LED
#define ARGB_PORT_NAME PORTB
#define ARGB_PORT_MASK PIN2_bm  //PB2

#define ARGB_PORT_AS_OUTPUT ARGB_PORT_NAME.DIRSET = ARGB_PORT_MASK
#define ARGB_PORT_AS_INPUT ARGB_PORT_NAME.DIRCLR = ARGB_PORT_MASK
#define ARGB_PORT_SET ARGB_PORT_NAME.OUTSET = ARGB_PORT_MASK
#define ARGB_PORT_RESET ARGB_PORT_NAME.OUTCLR = ARGB_PORT_MASK
#define ARGB_PORT_IS_SET (ARGB_PORT_NAME.IN & ARGB_PORT_MASK)
#define ARGB_PORT_IS_RESET (!(ARGB_PORT_NAME.IN & ARGB_PORT_MASK))

//Definicje timingów
#define NOP __asm__ __volatile__ ("nop")  //50ns
#define T0H NOP; NOP; NOP; NOP  //Minimum: 200 ns
#define T0L T0H; T0H; T0H; T0H  //Minimum: 800 ns
#define T1H T0H; T0H; T0H; NOP  //Minimum: 650 ns
#define T1L NOP; NOP; NOP; NOP  //Minimum: 200 ns
#define RESET_TIME 250  //us
#define READ_TIMEOUT 160  //us
#define COMMAND_TIMEOUT 60  //us

//Definicje stałych konfiguracji diod LED
#define PWM_FREQ_1k 0
#define PWM_FREQ_2k 1
#define PWM_FREQ_9k 2
#define PWM_FREQ_18k 3
#define OVERALL_CURR_NORMAL 0
#define OVERALL_CURR_HALVED 1
#define RESPONSE_IS_ID 1
#define RESPONSE_IS_CURRENT 0

//Definicje stałych rozkazów oraz statusów ich wykonania
#define CMD_ASSIGN_ADDRESS 0x10
#define CMD_RESET_ADDRESS 0x20
#define CMD_PING_ADDRESS 0x30
#define CMD_ACTIVATE_ADDRESS 0x40
#define CMD_EXECUTED 0
#define CMD_DISCARDED 1
```

Listing 1. Plik nagłówkowy modułu obsługi diod ARGB Gen2

Ustawienia Fuse-bitów:
FREQSEL[1:0]: 10*
RSTPINCFG[1:0]: 01*
SUT[2:0]: 111*
EESAVE: 1
*ustawienie domyślne producenta

Na razie wszystko wydaje się proste, gdyż mamy do czynienia z funkcjami niczym nieróżniącymi się od specyfikacji dobrze znanych diod WS2812. To prawda – wspomniałem przecież na wstępie, że diody ARGB Gen2 są wstecznie zgodne ze specyfikacją elementów WS2812, dlatego mogą być obsługiwane przez sterowniki starego typu. Przejdźmy zatem do zagadnień nieco bardziej skomplikowanych. Na **listingu 6** pokazano ciało funkcji pozwalającej na policzenie elementów LED znajdujących się na magistrali i korzystającej z trybu odczytu protokołu ARGB Gen2. Dalej, na **listingu 7**, widnieje z kolei ciało funkcji pozwalającej na konfigurację diod LED typu ARGB Gen2.

Co ważne, aby stosowną konfigurację przeprowadzić w sposób szybki i łatwy, najlepiej użyć zmiennej typu `configType` (będącej argumentem wywołania funkcji) i predefiniowanych w pliku nagłówkowym stałych. Należy również podkreślić, że prezentowana funkcja zakłada wysłanie takiej samej konfiguracji do wszystkich diod LED na magistrali, co zwykle okaże się najbardziej pożądane. Natomiast na **listingu 8** pokazano kluczową funkcję interfejsu ARGB Gen2, stosowaną w przypadku pracy magistrali danych w topologii gwiazdy – jej zadaniem jest wysłanie rozkazu sterującego (oraz opcjonalny odbiór odpowiedzi).

Przejdźmy teraz do założeń aplikacyjnych sterownika **argbController**. Przyznam szczerze, że pomysł (choć bardziej pasuje mi tu słowo „zapotrzebowanie") na ten projekt podsunął mi mój kolega Andrzej. Używając różnego rodzaju podzespołów PC (w tym wyposażonych w podświetlenie ARGB Gen2), zauważył on brak systemu, który w zależności od wyników pomiaru temperatury sterowałby kolorem tego rodzaju podświetlenia. Istnieją co prawda systemy

```
void argbInit(void)
{
    //Port sterujący LED, jako wyjściowy ze stanem 0
    ARGB_PORT_RESET;
    ARGB_PORT_AS_OUTPUT;
}
```

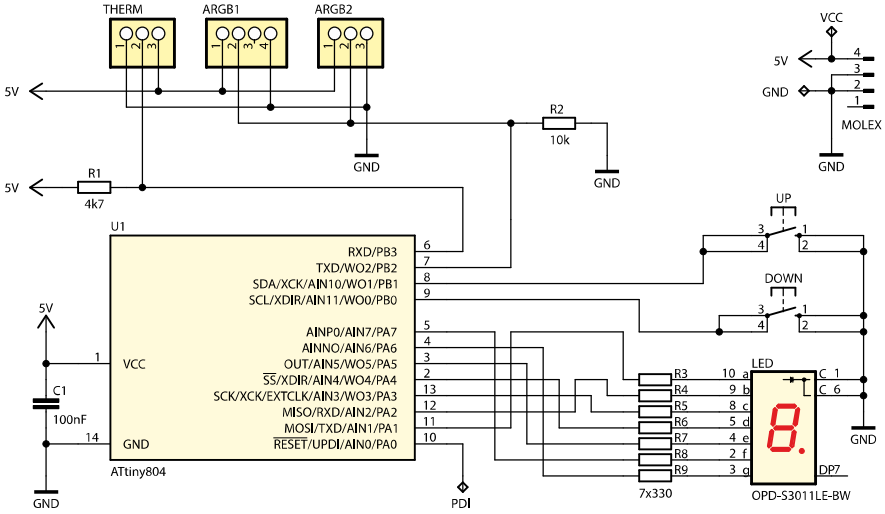
Listing 2. Kod funkcji inicjalizacyjnej interfejsu ARGB Gen2

```
//Funkcje zakładają częstotliwość taktowania mikrokontrolera równą
//F_CPU = 20 MHz (NOP = 50 ns)

__inline__ void __attribute__((__always_inline__)) argbSendBit0(void)
{
    ARGB_PORT_SET;
    T0H;
    ARGB_PORT_RESET;
    T0L;
}

__inline__ void __attribute__((__always_inline__)) argbSendBit1(void)
{
    ARGB_PORT_SET;
    T1H;
    ARGB_PORT_RESET;
    T1L;
}
```

Listing 3. Funkcje odpowiedzialne za przesłanie jednego bitu danych interfejsu ARGB Gen2

```
__inline__ void __attribute__((__always_inline__)) argbSendByte(uint8_t Byte)
{
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
    {
        if(Byte & 0x80) argbSendBit1(); else argbSendBit0();
        if(Byte & 0x40) argbSendBit1(); else argbSendBit0();
        if(Byte & 0x20) argbSendBit1(); else argbSendBit0();
        if(Byte & 0x10) argbSendBit1(); else argbSendBit0();
        if(Byte & 0x08) argbSendBit1(); else argbSendBit0();
        if(Byte & 0x04) argbSendBit1(); else argbSendBit0();
        if(Byte & 0x02) argbSendBit1(); else argbSendBit0();
        if(Byte & 0x01) argbSendBit1(); else argbSendBit0();
    }
}
```

Listing 4. Funkcja odpowiedzialna za przesłanie jednego bajtu danych interfejsu ARGB Gen2



Rysunek 6. Schemat ideowy urządzenia argbController

```
__inline__ void __attribute__((__always_inline__)) argbSendColor(uint8_t R, uint8_t G, uint8_t B)
{
    argbSendByte(G);
    argbSendByte(R);
    argbSendByte(B);
}
```

Listing 5. Funkcja pozwalającą na przesłanie kompletnej informacji o kolorze diody LED typu ARGB Gen2

Wykaz elementów:

Rezystory: (SMD 0805)
R1: 4,7 kΩ
R2: 10 kΩ
R3...R9: 330 Ω

Kondensatory: (SMD 0805)
C1: 100 nF (ceramiczny X7R)

Półprzewodniki:
U1: ATtiny804 (SO14)
LED: wyświetlacz 7-segmentowy 0,3" typu OPD-S3011LE-BW (wspólna katoda)

Pozostałe:
UP, DOWN: microswitch TACT, h=9 mm
ARGB2, THERM: listwa kołkowa goldpin, prosta, 1×3 pin
ARGB1: listwa kołkowa goldpin, prosta, 1×4 pin
MOLEX: gniazdo MOLEX męskie typu C5080WR-F-04P (JOINT TECH)
SONDA: termometr scalony typu DS18B20 (lub sonda w niego wyposażona)

```
uint8_t argbCountLeds(void)
{
    uint8_t Timeout = 0;
    uint8_t Leds = 0;

    ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
    {
        //Inicjujemy tryb odczytu
        ARGB_PORT_SET;
        _delay_us(20);
        ARGB_PORT_RESET;
        _delay_us(10);
        ARGB_PORT_SET;
        _delay_us(50);
        ARGB_PORT_RESET;
        //Konfigurujemy interfejs danych, jako wejściowy, by odczytywać impulsy wysyłane przez diody LED
        ARGB_PORT_AS_INPUT;

        //Odczytujemy kolejne impulsy odpowiedzi wysyłane przez diody LED. Magistrala podciągnięta jest do masy rezystorem 10k
        while(1)
        {
            //Sprawdzamy czy nie wystąpił time-out oznaczający koniec transmisji lub zupełny brak diod na magistrali
            if(ARGB_PORT_IS_RESET)
            {
                if(++Timeout > READ_TIMEOUT) break;
                _delay_us(1);
            }
            //Liczymy impulsy wysyłane przez diody LED by określić liczbę tychże diod na magistrali danych
            if(ARGB_PORT_IS_SET)
            {
                ++Leds;
                //Czekamy na zakończenie bieżącego impulsu (stanu wysokiego)
                while(ARGB_PORT_IS_SET);
                _delay_us(1);
                Timeout = 0;
            }
        }
        //Konfigurujemy interfejs danych, jako wyjściowy (domyślnie)
        ARGB_PORT_AS_OUTPUT;
    }

    return Leds;
}
```

Listing 6. Funkcja pozwalającą na policzenie elementów LED znajdujących się na magistrali ARGB Gen2

```
void argbSendConfig(configType *Config, uint8_t Leds)
{
    uint8_t Byte1, Byte2, Byte3;

    //Przygotowujemy dane do wysłania według specyfikacji słów konfiguracyjnych
    Byte1 = (Config->pwmFreq<<6)|Config->fineG;
    Byte2 = (Config->ovCurr<<7)|(Config->ovCurr<<6)|(Config->respType<<5)|Config->fineR;
    Byte3 = (Config->ovCurr<<7)|Config->fineB;

    ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
    {
        //Inicjujemy tryb konfiguracji
        ARGB_PORT_SET;
        _delay_us(20);
        ARGB_PORT_RESET;
        _delay_us(300);

        //Wysyłamy taką samą konfigurację do wszystkich diod LED
        for(uint8_t i=0; i<Leds; i++) argbSendColor(Byte2, Byte1, Byte3);
        _delay_us(300);
    }
}
```

Listing 7. Funkcja pozwalającą na konfigurację diod LED typu ARGB Gen2

pozwalające zarządzać bogactwem różnych efektów RGB (np. Asus Aura Sync, Gigabyte RGB-Fusion Ready, MSI Mystic Light Sync czy ASRock Polychrome Sync), lecz nie oferują one tak pożądanej, a jednocześnie prostej funkcjonalności, jak zmiana koloru podświetlenia na skutek zmian temperatury elementu monitorowanego. Łatwo wyobrazić sobie zastosowanie takiego systemu do monitorowania stanu radiatora procesora PC lub temperatury wewnątrz obudowy typu desktop: w ramach pełnionej funkcji zmieniałby on (w sposób sugestywny) kolor podświetlenia wentylatora, dając bezpośrednią informację na temat kondycji urządzenia. To oczywiście tylko jedno z wielu możliwych zastosowań, gdyż nic nie każe nam ograniczać się do sprzętu z branży PC.

Schemat ideowy układu pokazano na **rysunku 6**. Sercem urządzenia jest mikrokontroler ATtiny804, taktowany wewnętrznym oscylatorem o częstotliwości równej 20 MHz i odpowiedzialny za realizację całej założonej funkcjonalności urządzenia. Jako element pomiarowy (termometr) zastosowano scalony, cyfrowy i bardzo dobrze znany element typu DS18B20 produkcji Maxim Integrated, którego obsługę zrealizowano przy użyciu programowej implementacji magistrali 1-Wire. Wybór tego elementu z szerokiej palety dostępnych termometrów scalonych podyktowany był jego dużą dostępnością i niską ceną. Co więcej: w handlu znaleźć możemy gotowe moduły w formie wodoszczelnych sond pomiarowych, odpornych na warunki zewnętrzne, z zatopionym wewnątrz układem DS18B20, przez co wybór ten stał się jeszcze bardziej oczywisty. Ponadto mikrokontroler nasz obsługuje jednocyfrowy,

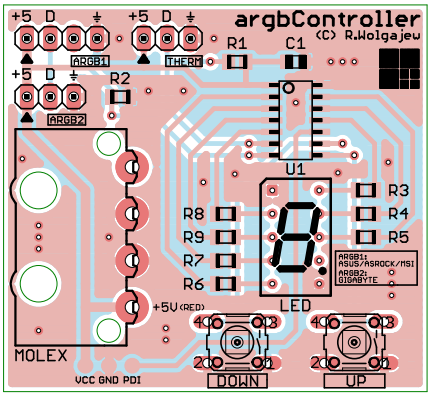
```
uint8_t argbSendCommand(uint8_t Command, uint8_t Address)
{
    uint8_t cmdStatus = CMD_DISCARDED, Timeout = 0;

    ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
    {
        //Inicjujemy tryb rozkazów
        ARGB_PORT_SET;
        _delay_us(40);
        ARGB_PORT_RESET;
        _delay_us(12);

        //Wysyłamy rozkaz (korzystając z predefiniowanych wartośći) wraz z pożądanym adresem łańcucha LED
        argbSendByte(Command|Address);

        //Sprawdzamy odpowiedź łańcucha, która to powinna być wysłana prze pierwszą z diod LED w tymże łańcuchu LED (tak zwaną brakmę)
        //Odpowiedź ta to impuls stanu wysokiego o długości 60uS wysłany w losowym czasie od 10us do 60uS od zbocza opadającego ostatniego
        //bitu rozkazu. Konfigurujemy interfejs danych, jako wejściowy, by odczytać odpowiedź łańcucha biorąc pod uwagę ewentualny time-out
        ARGB_PORT_AS_INPUT;

        while(1)
        {
            //Sprawdzamy czy nie wystąpił time-out oznaczający niewykonanie rozkazu
            if(ARGB_PORT_IS_RESET)
            {
                if(++Timeout > COMMAND_TIMEOUT) break;
                _delay_us(1);
            }
            //Sprawdzamy obecność impulsu stanu wysokiego wysłanego przez bramkę łańcucha LED potwierdzającego wykonanie rozkazu
            if(ARGB_PORT_IS_SET)
            {
                //Czekamy na zakończenie bieżącego impulsu (stanu wysokiego)
                while(ARGB_PORT_IS_SET);
                _delay_us(1);
                cmdStatus = CMD_EXECUTED;
                break;
            }
        }
        //Konfigurujemy interfejs danych, jako wyjściowy (domyślnie)
        ARGB_PORT_AS_OUTPUT;
    }

    return cmdStatus;
}
```

Listing 8. Funkcja pozwalająca na przesłanie rozkazu i opcjonalny odbiór odpowiedzi na magistrali typu ARGB Gen2

7-segmentowy wyświetlacz LED oraz dwa przyciski funkcyjne, oznaczone jako UP i DOWN, które stanowią elementarną wersję interfejsu użytkownika. W konstrukcji układu przewidziano również złącza goldpin przeznaczone do podłączenia wentylatorów (lub innych urządzeń), wyposażonych w interfejs ARGB Gen2 zasilany napięciem 5 V. Zastosowano 2 typy złączy w różnej konfiguracji sygnałów wyjściowych, typowych dla płyt głównych ASUS/ASROCK/MSI lub GIGABYTE. Do zasilania sterownika napięciem 5 V przewidziano z kolei typowe gniazdo MOLEX, stosowane w komputerach klasy PC, gdyż kable tego rodzaju – wyposażone we wtyki żeńskie – znajdują się w każdym komputerze typu desktop (i służą m.in. do zasilania napędów). Do obsługi przycisków funkcyjnych oraz programowej eliminacji drgania styków zastosowano przerwanie od przepełnienia timera TCB0 wbudowanego w strukturę mikrokontrolera, więc możliwa stała się również detekcja krótkiego i długiego naciśnięcia przycisków – bez blokowania programu obsługi aplikacji.

Montaż i uruchomienie

Schemat montażowy naszego urządzenia pokazano na **rysunku 7**. Niewielki, dwustronny obwód drukowany przeznaczony jest w zdecydowanej większości do montażu powierzchniowego, który rozpoczynamy od przylutowania mikrokontrolera. Element ten – mimo obudowy SMD – charakteryzuje się dość szerokim rozstawem wyprowadzeń, przez co nie powinien nastręczać problemów podczas lutowania. Następnie montujemy elementy bierne, wyświetlacz LED, by na końcu wlutować złącza goldpin ARGB1, ARGB2 i THERM, przyciski UP i DOWN oraz złącze zasilające MOLEX. W przypadku złącza goldpin ARGB1 należy usunąć nieużywany pin drugi od prawej (w celu zabezpieczenia przed odwrotnym podłączeniem), gdyż gniazda tego rodzaju nie przewidują jego obecności. Poprawnie zmontowany układ nie wymaga jakichkolwiek regulacji i powinien działać tuż po włączeniu zasilania i zaprogramowaniu procesora. Na **fotografii tytułowej** zaprezentowano wygląd zmontowanej płytki drukowanej urządzenia, widzianej od strony TOP.

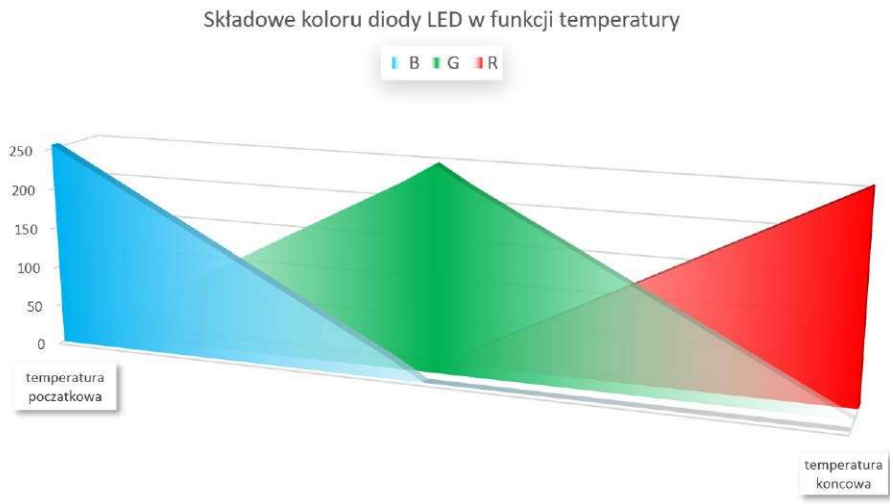


Rysunek 7. Schemat montażowy urządzenia argbController

Obsługa urządzenia

Obsługa sterownika argbController jest niezmiernie prosta. Do ustawienia mamy dwie wartości: temperaturę początkową i przedział temperatur, przy czym obie wartości regulowane są ze skokiem 10°C, w zakresie od 0°C do 90°C dla temperatury początkowej i od 10°C do 90°C – dla przedziału temperatur. Co ważne, na wyświetlaczu urządzenia temperatury pokazywane są w dziesiątkach °C. Na podstawie wprowadzonych ustawień urządzenie wylicza składowe kolorów RGB i wysyła je (takie same) do wszystkich diod LED we wszystkich podłączonych łańcuchach LED, w cyklach 1-sekundowych (przy czym liczbę podłączonych łańcuchów diod LED oraz liczbę samych diod LED w każdym z tych łańcuchów urządzenie sprawdza wyłącznie podczas włączania zasilania). Jak łatwo się domyślić, temperatura początkowa określa minimalną temperaturę, dla której urządzenie prześle do diod LED składowe kolorów RGB odpowiedzialne za wyświetlenie koloru niebieskiego, zaś temperatura końcowa, obliczona jako suma temperatury początkowej i przedziału temperatur, określa maksymalną temperaturę, dla której urządzenie prześle do diod LED składowe kolorów RGB odpowiedzialne za wyświetlenie koloru czerwonego. Wartości spoza zakresu obcięte zostaną do wartości skrajnych, zaś wszystkie pośrednie powodują przesłanie składowych koloru RGB obliczanych zgodnie z **rysunkiem 8**.

Nasuwa się pytanie, jak więc wprowadzamy niezbędne ustawienia? Urządzenie



Rysunek 8. Sposób wyznaczania składowych koloru diod LED na podstawie wartości mierzonej temperatury

po włączeniu zasilania przechodzi do trybu bezczynności, z wyłączonym wyświetlaczem LED (w celu oszczędzania energii). Jakiekolwiek naciśnięcie przycisków funkcyjnych przełącza je w tryb regulacji temperatury początkowej (włączając jednocześnie wyświetlacz LED) – w trybie tym krótkie naciśnięcie przycisków UP lub DOWN skutkuje zmianą wartości temperatury (w pętli). Długie naciśnięcie przycisku UP powoduje z kolei przejście do trybu regulacji przedziału temperatur, w którym – jak poprzednio – krótkie naciśnięcie przycisków UP lub DOWN wywołuje zmianę bieżącej wartości (również w pętli). Długie naciśnięcie przycisku UP powoduje tym razem powrót urządzenia do trybu bezczynności i wyłączenie wyświetlacza z jednoczesnym zapisaniem ustawień w nieulotnej pamięci EEPROM mikrokontrolera, by mogły być wczytane jako startowe podczas włączania urządzenia. Co ważne, program wyposażono dodatkowo w zegar bezczynności, który po upływie 20 sekund braku reakcji ze strony użytkownika powoduje automatyczne przełączenie w tryb bezczynności i wyłączenie wyświetlacza, bez zapisywania dokonanych ustawień w pamięci EEPROM. Dodatkowo każdorazowo podczas dokonywania pomiaru temperatury (czyli co 1 sekundę) sprawdzana jest obecność czujnika temperatury i – w przypadku jego braku (lub problemów na magistrali 1-wire) – na wyświetlaczu widoczna jest informacja o błędzie w postaci litery „**E**” (dopóki dany problem nie ustąpi).

Robert Wołgajew, EP