POLITECHNIKA LUBELSKA

Wydział Elektrotechniki i Informatyki

Kierunek Elektrotechnika

PRACA INŻYNIERSKA

Projekt instalacji elektrycznej w ogrodzie sterowanej za pomocą platformy Arduino

Project of electrical installation in the garden controlled by Arduino platform

Dyplomant:
Emil Rudnicki
nr albumu: 090813

Promotor:
dr. inż. Sebastian Franciszek Styła

Lublin 2022

Streszczenie

W pracy przedstawiono projekt sterownika ogrodowego opartego na płytce ewaluacyjnej Arduino wraz z modelem dydaktycznym odzwierciedlającym jego prace.

Przedstawiony został opis platformy Arduino oraz protokołów komunikacyjnych wykorzystywanych przez mikrokontrolery, na podstawie przykładów w literaturze. Omówiony został również schemat ideowy oraz wszystkie podzespoły sterownika wraz z magistralami danych, które są wykorzystywane do komunikacji.

Ponadto przeanalizowany został interfejs graficzny użytkownika. Opisane zostały także wszystkie parametry sterownika, które użytkownik może modyfikować podczas pracy urządzenia.

Omówiony został kod źródłowy mikroprocesora wraz z opisem najważniejszych funkcji, procedur i stanów pracy, w których może się znaleźć sterownik podczas normalnego trybu pracy.

Słowa kluczowe: programowanie, Arduino, projektowanie urządzeń, ogród, protokoły komunikacyjne.

Abstract

The purpose of thesis is to design and create didactic model reflecting principle of operating of a garden controller based on Arduino evaluation board

A description of the Arduino platform and communication protocols used by microcontrollers was presented, based on examples in literature. The schematic diagram and all controller components were discussed, along with the data buses they use for communication.

The graphical user interface was analyzed. All the parameters of the controller that can be modified by the user during the operation of the device have been described.

Last described issue is microprocessor source code, with contains description of all functions, procedures and operating states, in with the controller can be found during normal operations.

Key words: programming, Arduino, device design, garden, communication protocols.

Spis treści:

1. Wstęp

Postęp elektroniki w dzisiejszych czasach sprzyja rozwojowi różnych dziedzin życia. Jedną z tych dziedzin są różnego rodzaju sterowniki montowane w domach jednorodzinnych bądź mieszkaniach, mające za zadanie ułatwić wykonywanie codziennych czynności przez domowników.

W ostatnich czasach można zaobserwować zdecydowany wzrost zainteresowania ogrodnictwem. Spowodowało to rozwój kolejnej gałęzi elektroniki, do której należą urządzenia wspomagające pielęgnowanie upraw w ogrodzie. Ich celem jest zapewnienie uprawom odpowiedniego nawodnienia oraz oświetlenia, w taki sposób aby uzyskały optymalne warunki do wzrostu i wegetacji.

Sterowniki ogrodowe najczęściej oparte są na różnego rodzaju układach programowalnych, które zarządzają pracą wszystkich podzespołów. Ze względu na rodzaj układów programowalnych znajdujących się wewnątrz sterowników ogrodowych, można zauważyć podział na: systemy inteligentnej instalacji elektrycznej oraz sterowniki oparte na mikroprocesorach. Zakres działań sterowników ogrodowych odpowiada inteligentnej instalacji elektrycznej (najczęściej KNX/EIB). Wewnątrz niej wszystkie elementy instalacji komunikują się miedzy sobą nie posiadając przy tym centralnej jednostki sterującej. Wbrew powszechnej opinii, którą spotyka się w literaturze [3], urządzenia działające w sieci KNX/EIB nie są jedynymi, które mogą być użyte w projekcie inteligentnego domu. Sterowniki, w których wykorzystane są mikroprocesory mogą spełniać te same funkcje co KNX/EIB, a także mogą być rozszerzone o dowolny zakres działań (czyli np. dodatkowy czujnik, inna oprawa graficzna czy rozmieszczenie danych na wyświetlaczu).

2. Cel i zakres pracy

Celem pracy jest zaprojektowanie, połączenie i zaprogramowanie sterownika ogrodowego, który zasymuluje podstawowe elementy instalacji ogrodowej. Elementami wykonawczymi tej instalacji będą pompka wody do podlewania upraw oraz oświetlenie. W skład funkcji sterownika wchodzić będzie pokazywanie na wyświetlaczu: aktualnych wskazań czasu i daty, dzięki dołączonemu zegarowi czasu rzeczywistego oraz wskazań wilgotności i temperatury. Dodatkowo sterownik, dzięki zastosowaniu wyświetlacza dotykowego będzie posiadał możliwość ustawienia:

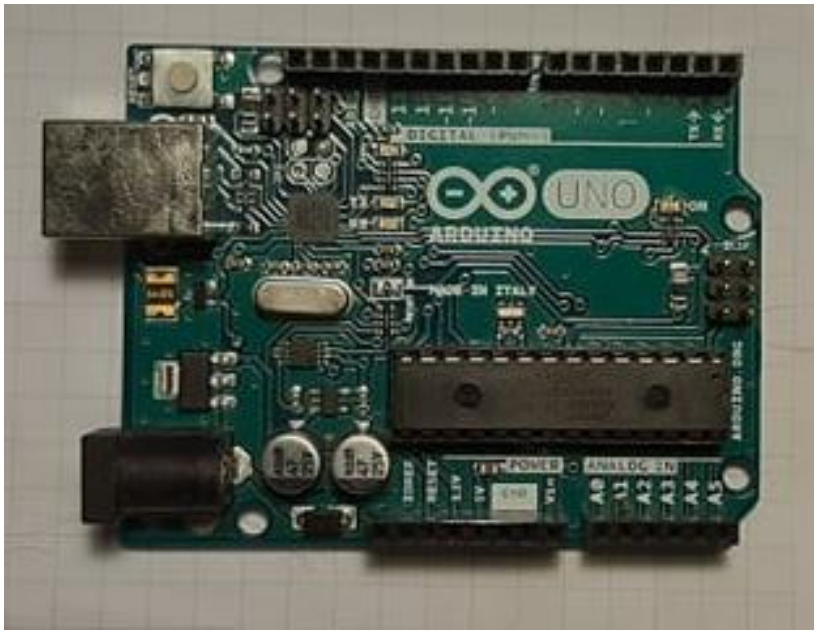
- aktualnej godziny,
- aktualnej daty,
- godziny załączenia i wyłączenia pompki wody,
- poziomu jasności światła na zewnątrz.

Praca swoim zakresem obejmować będzie: zaprojektowanie interfejsu graficznego użytkownika, a także menu, w którym ustawić będzie można wyżej wymienione opcje. Ponadto zakres pracy obejmie wykonanie działającego prototypu sterownika ogrodowego, który zaprezentuje możliwości gotowego urządzenia. Dodatkowo w pracy znajdzie się objaśnienie zarówno wskazań wyświetlanych na ekranie głównym, jak i dostępnych opcji znajdujących się w menu i podmenu. Na podstawie wiedzy zaczerpniętej z literatury [2], opisane będą wszystkie niezbędne protokoły komunikacyjne użyte przez mikrokontroler. Zaprezentowane zostaną czujniki wraz z pozostałymi podzespołami znajdującymi się w projekcie. W dalszej części znajdzie się analiza kodu źródłowego programu i bibliotek wraz z komentarzem do poszczególnych funkcji czy podprogramów. Głównym założeniem tej części pracy będzie pomoc w zrozumieniu zasady działania urządzenia. Na końcu sformułowane zostaną wnioski, które pozwolą podsumować zakres prac związanych z budową sterownika oraz jego ogólnym zastosowaniem.

3. Platforma Arduino

Najważniejszą częścią całego sterownika jest Arduino Uno, czyli płytka ewaluacyjna zawierająca mikrokontroler Atmega328p, będący produktem flagowym firmy Atmel dobrze znanej programistom mikroprocesorów. Samo Arduino używane jest między innymi w celach edukacyjnych. Istnieje wiele książek oraz poradników dla osób bez doświadczenia w programowaniu, z którymi osoby takie będą w stanie nauczyć się programowania w języku C# oraz C++. Kolejnym zastosowaniem dla Arduino jest tworzenie prototypów urządzeń ze względu na obszerny zbiór bibliotek do podzespołów, jakich się używa przy ich tworzeniu. Te biblioteki pozwalają sprawdzić jak dany podzespół sprawdzi się w projekcie oraz jak będzie działała całość.

Wymieniony mikroprocesor Atmega328p, czyli programowalny układ scalony wyposażony jest w zbiór wejść / wyjść cyfrowych oraz w wejścia analogowe określane jako ADC (ang. Analog to Digital Converter) przekształcających poziom napięcia danego wejścia na wartość binarną. Ponadto znajdują się tutaj wyjścia PWM (ang. Pulse Width Modulation) generujące sygnał o określonej częstotliwości i wypełnieniu, a także klika wyjść dedykowanych do użycia magistrali danych I^2C, SPI oraz RS232 posiadających specjalne zbiory rejestrów mikroprocesora (dokładne opisy są w nocie katalogowej produktu [9]).



Rys. 3.1. Arduino Uno

Na płytce Arduino Uno (rys. 3.1) poza mikrokontrolerem znajdują się również inne elementy, takie jak: złącze programatora po lewej stronie na górze, złącze zasilacza w lewym dolnym rogu, jak również 4 złącza wyprowadzeń z samego mikroprocesora (2 na górze i 2 na dole). W skład płytki ewaluacyjnej wchodzi również:

- programator,
- stabilizator napięcia zmieniający napięcie z 12 V na 5 V do zasilenia procesora i podzespołów,
- stabilizator napięcia zmieniający napięcie z 5 V na 3,3 V do zasilania podzespołów pracujących przy tym napięciu.

Są to najważniejsze elementy, których nie można pominąć przy przedstawieniu Arduino, ponieważ są one niezbędne do poprawnego działania sterownika. Dalszy opis pozostałych elementów elektronicznych oraz złącz znajdujących się na płytce Arduino Uno, można znaleźć w literaturze [5].

Warto także wspomnieć, że o ile do programowania samej Atmegi328p istnieje wiele środowisk programistycznych, takich jak AtmelStudio (dawniej AVR Studio), czy Eclipse, w których używa się języka C# o tyle do zaprogramowania Arduino przeznaczone jest specjalne środowisko o tej samej nazwie, czyli Arduino. Używa się w nim języka C++. Ponadto rozbudowane jest ono o biblioteki ułatwiające sterowanie portami wejścia / wyjścia w literaturze określanymi jako GPIO (ang. General Port Input / Output) [4]. Biblioteki Arduino sprawiają że jest to środowisko bardziej wysoko poziomowe, w porównaniu do pozostałych środowisk programistycznych poświęconych mikrokontrolerom AVR. W Arduino udział programisty wyklucza błędny zapis rejestrów. Jedyną wadą staje się wydłużenie czasu wykonywania procedury, ze względu na warunki przez które musi przejść funkcja Arduino zanim nadpisze rejestr.

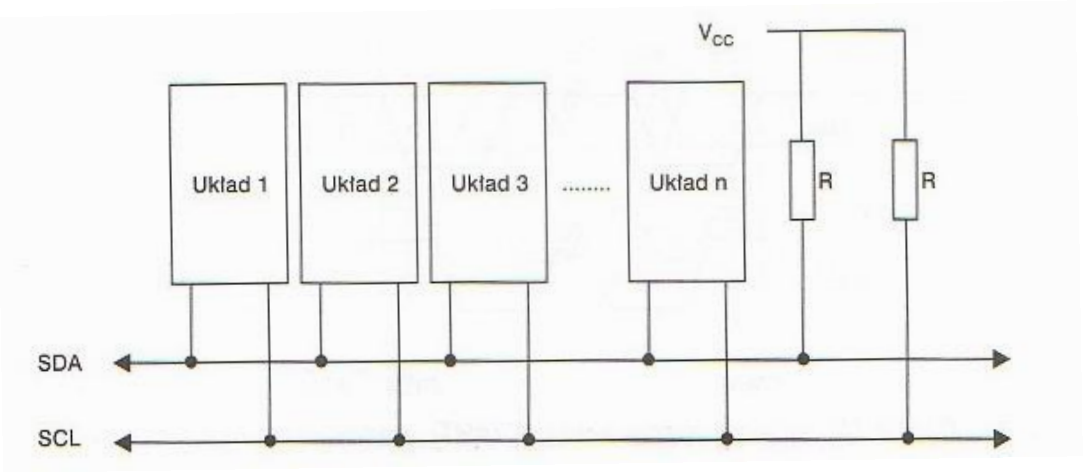
Kolejnym ważnym aspektem tej platformy jest duża ilość wspieranych magistrali danych, będących podstawą do komunikacji z podzespołami nazywanymi jako układy podrzędne. Jak zostało wcześniej wspomniane rejestry Atmegi wspierają tylko 3 protokoły. Natomiast Arduino posiada ich już dużo więcej. Zawarte są one we wspomnianym już wcześniej zbiorze bibliotek. Magistrale danych stanowią podstawę komunikacji między podzespołami, czyli układami podrzędnymi a procesorem będącym układem nadrzędnym. Z tego też względu kolejny rozdział poświęcony zostanie na opis tych magistrali, które zostały użyte w projekcie.

4. Magistrale danych

Jednym z podstawowych zagadnień, potrzebnych do zrozumienia dalszej części pracy, jest znajomość magistrali danych. Głównym celem stosowania magistrali jest wymiana informacji. Występuje ona między układem nadrzędnym a podrzędnym (lub grupą układów podrzędnych). W przypadku sterownika ogrodowego układem nadrzędnym stanie się Arduino Uno, natomiast do układów podrzędnych należeć będą różnego rodzaju czujniki omówione dalej. W tym rozdziale opisane zostaną wszystkie magistrale danych oraz protokoły komunikacyjne, z których korzystać będzie Arduino.

4.1. I^2C

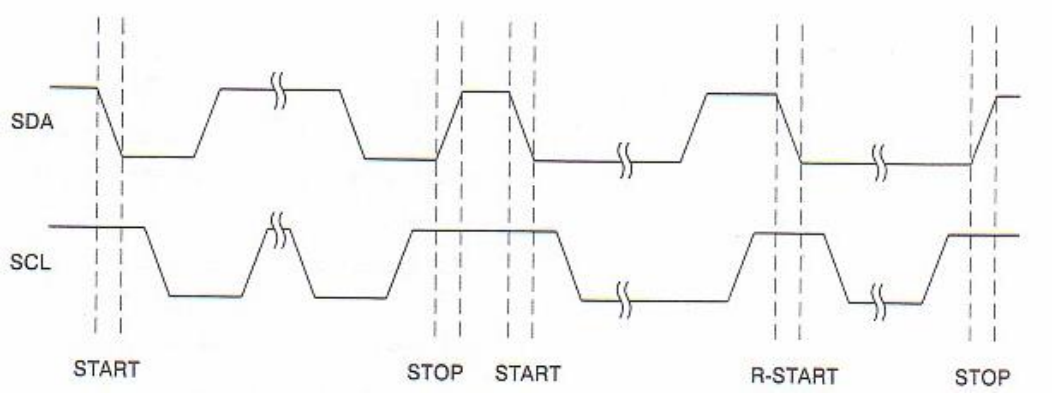
Nazwa magistrali danych I^2C pochodzi od rozwinięcia skrótu Inter-Integrated Circuit co w tłumaczeniu oznacza obwód wewnętrznie zintegrowany. Do jej określenia używa się również skrótu TWI (ang. Two Wire Interface) czyli interfejsu dwuprzewodowego. I^2C jest w istocie synchronicznym interfejsem szeregowym małego zasięgu, który pozwala na dołączanie wielu nadajników i odbiorników do wspólnej magistrali (rys. 4.1) poprzez linie (SDA i SCL). Linie te podłączone są poprzez rezystory podciągające do napięcia zasilania, co zapewnia stabilność transmisji danych. Dalsza analiza bazować będzie na ilustracjach zaczerpniętych z literatury [1], w której znajduje się również opis zagadnień związanych z magistralami danych.



Rys. 4.1. Układ połączeń magistrali I^2C [1]

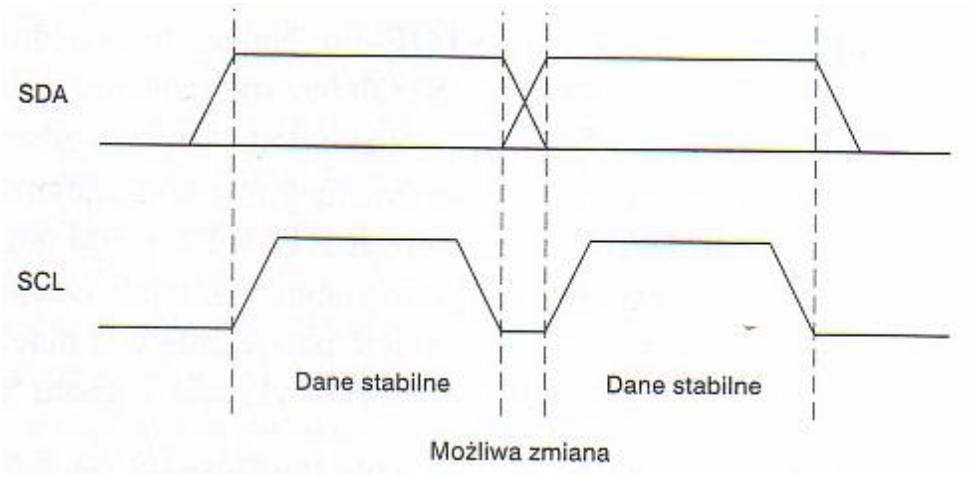
Linia SDA (ang. Serial DatA) jest dwukierunkową linią danych, której używają do wysyłania i odbierania danych zarówno układy nadrzędne jak i podrzędne. Linia SCL (ang. Serial CLock), czyli linia zegarowa, służy do przesyłania sygnału taktującego (czyli sygnału

prostokątnego o stałym wypełnieniu 50 %). W przypadku tej linii sygnał generuje tylko układ nadrzędny, który sprawuje nadzór nad magistralą. Co za tym idzie, tylko on jest w stanie rozpocząć komunikacje z dowolnym układem podrzędnym.



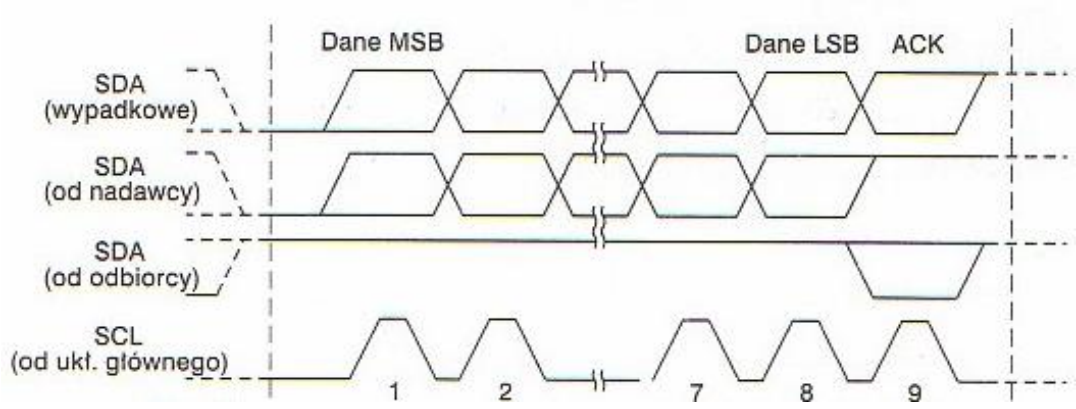
Rys. 4.2. Sygnały rozpoczęcia, ponownego rozpoczęcia i zakończenia transmisji [1]

Podstawową informacją związaną z interfejsem I^2C jest sposób rozpoczęcia, ponownego rozpoczęcia i zakończenia transmisji przy ustawieniu przez układ nadrzędny odpowiedniego stanu logicznego na obu liniach danych (rys. 4.2). Dla sygnału rozpoczęcia (START) jest to zmiana z logicznego stanu wysokiego na niski na linii danych SDA, przy jednoczesnym zachowaniu stanu wysokiego na linii zegarowej SCL. Od tego momentu rozpoczyna się komunikacja między układem nadrzędnym, a podrzędnym. Do zakończenia przesyłania danych potrzebne jest podanie sygnału STOP, czyli przejście ze stanu niskiego do stanu wysokiego na linii danych, przy jednoczesnym zachowaniu stanu wysokiego na linii zegarowej. Po przesłaniu tego sygnału układy podrzędne nie będą odbierać danych dopóki nie pojawi się ponownie sygnał START. W trakcie komunikacji możliwe jest nadanie sygnału ponownego rozpoczęcia komunikacji (R-START), wyglądającego tak samo jak wcześniej omówiony sygnał START. Używa się go w momencie, gdy układ nadrzędny chce ponownie zainicjować transmisję danych. Dzieje się tak kiedy w magistrali znajdują się co najmniej dwa układy nadrzędne, a układ aktualnie prowadzący transmisję danych nie chce stracić panowania nad magistralą (co mogłoby się zdarzyć, gdyby inny układ nadrzędny wygenerowałby sygnał STOP).



Rys. 4.3. Przebiegi czasowe przesyłanych bitów na liniach SDA i SCL [1]

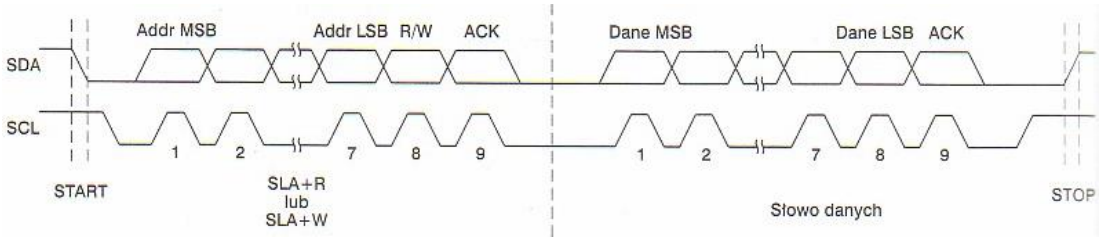
Znając powyższe sygnały oraz ich znaczenie można przystąpić do objaśnienia zasady odczytu pojedynczych bitów oraz tego jak są one interpretowany przez układy posiadające interfejs I^2C. Po rozpoczęciu komunikacji (po podaniu sygnału START) i rozpoczęciu generowania przebiegu prostokątnego przez układ nadrzędny na linii zegarowej, możliwa zmiana stanu logicznego na linii danych następuje tylko wtedy, gdy na linii SCL znajduje się stan niski. Od momentu pojawienia się na niej zbocza narastającego, do momentu zakończenia zbocza opadającego, stan logiczny na linii SDA nie może się zmieniać (rys. 4.3). Określa się to jako stan danych stabilnych. Wysyłany lub odbierany jest wtedy pojedynczy bit mogący przyjmować wartość „0” lub „1” (zależnie od stanu logicznego na linii SDA).



Rys. 4.4. Przebiegi czasowe ramki danych na liniach SDA i SCL [1]

Pojedyncza ramka danych składa się z 9 bitów (rys. 4.4), gdzie pierwsze 8 bitów są to dane przesyłane od bitu najstarszego (MSB) do bitu najmłodszego (LSB) przez układ nadawczy. Ostatni bit nazywany jest jako ACK (ang. ACKnowledbe bit) służy do

potwierdzenia odebrania ramki danych przesłany przez układ nadawczy. Po przesłaniu 8 bitów układ nadawczy może odebrać bit „0”, który potwierdzi przyjęcie ramki danych przez odbiorcę i oznaczać będzie gotowość na odebranie kolejnej ramki danych, lub gdy odbiorca przekaże bit „1” oznaczać to będzie zakończenie transmisji danych i wymagane będzie podanie sygnału STOP przez układ nadrzędny. Przesyłanie bitu potwierdzenia dotyczy zarówno układów nadrzędnych jak i podrzędnych znajdujących się w magistrali w momencie kiedy są one odbiorcami ramki danych.



Rys. 4.5. Transmisja jednego bajta [1]

Przy rozpoczęciu komunikacji między układem nadrzędnym a podrzędnym, poza podaniem sygnału START przez układ główny, wymagane jest przesłanie w pierwszej ramce 7-bitowego adresu układu podrzędnego. Określanego w literaturze [1] jako SLA (ang. SLave Address) w kolejności od bitu najbardziej znaczącego do bitu najmniej znaczącego (rys. 4.5). Taki adres musi posiadać każdy układ podrzędny. Bit nr 8 oznaczony jako R/W (ang. Read / Write), czyli zapis / odczyt zawiera informację o tym czy układ nadrzędny zamierza wysyłać dane w kolejnych ramkach. Wtedy ten bit przyjmuje logiczną wartość „0”, bądź jeżeli będzie to „1” oznaczać to będzie, że układ główny w dalszej części transmisji będzie odczytywać dany rejestr układu podrzędnego. Na rysunku 4.5 przedstawiono przykładową transmisję jednego bajta. Analizując rysunek od lewej strony widać, że podany zostaje sygnał START, następnie przesłano 7-bitowy adres, dalej bit odczytu / zapisu oraz bit potwierdzenia, który generuje układ odbiorczy. W kolejnej ramce układ nadrzędny przesyła 8 bitów danych, a następnie odbiera od układu podrzędnego po raz kolejny bit potwierdzenia, po czym generuje sygnał kończący transmisję.

W praktyce schemat wysyłania danych polega na wygenerowaniu sygnału START, następnie przesłaniu w pierwszej ramce adresu układu odbiorczego wraz z bitem R/W przyjmującym logiczną wartość „0”. W kolejnej ramce przesyłany jest adres rejestru, który procesor będący układem nadrzędnym ma zamiar zapisać bądź nadpisać, a w trzecim bajcie następuje faktyczna zmiana wartości rejestru układu podrzędnego będącym również odbiornikiem. Po tym procesor może wygenerować sygnał STOP, który zakończyłby aktualną

transmisję, lub przesyłać kolejne bajty, co skutkowałoby zapisem kolejnych rejestrów odbiornika. W takim przypadku drugi bajt przesłany przez procesor oznacza rejestr od którego rozpoczyna się zapis. Przykładowo, jeżeli wartość tego bajta wynosiłaby 10, a następnie przesłane byłyby 3 ramki, wówczas nastąpiłoby zapisanie/nadpisanie rejestrów o numerach 10, 11 i 12. Rejestry te przyjęły by kolejno wartości ramek danych przesłanych przez procesor. Po wysłaniu przez procesor ostatniej ramki danych wygenerowałby on sygnał STOP kończący transmisję.

Schemat odczytu danych dla większości podzespołów wykorzystujących interfejs I^2C wygląda podobnie, jak miało to miejsce w przypadku wysyłania danych. Wyjątkiem jest to, że po wygenerowaniu sygnału START, w pierwszym bajcie po adresie odbiorcy musi znaleźć się bit „1". Jest on w istocie sygnałem dla układu podrzędnego, że procesor zamierza odczytać dany rejestr lub zbiór rejestrów. Drugi bajt jest również wysyłany przez procesor i oznacza on tym razem numer rejestru od którego procesor zamierza odczytywać dane. Następnie linia SDA w procesorze zmienia się z wyjścia cyfrowego na wejście cyfrowe. Wtedy odczytywany jest dany bajt, po którym układ nadrzędny przesyła bit potwierdzenia ACK. Jeżeli nie jest to ostatnia ramka którą procesor zamierza odebrać, wtedy przyjmuje on wartość „0". Układ odbiorczy przesyła wtedy ramkę o wartości jaką przyjmuje rejestr będący o 1 większy od rejestru uprzednio odczytanego. W momencie kiedy procesor po odczytaniu danego bajta wyśle bit ACK posiadający wartość „1" następuje zakończenie odczytu danych. Następnie układ nadrzędny wysyła sygnał STOP kończący transmisję.

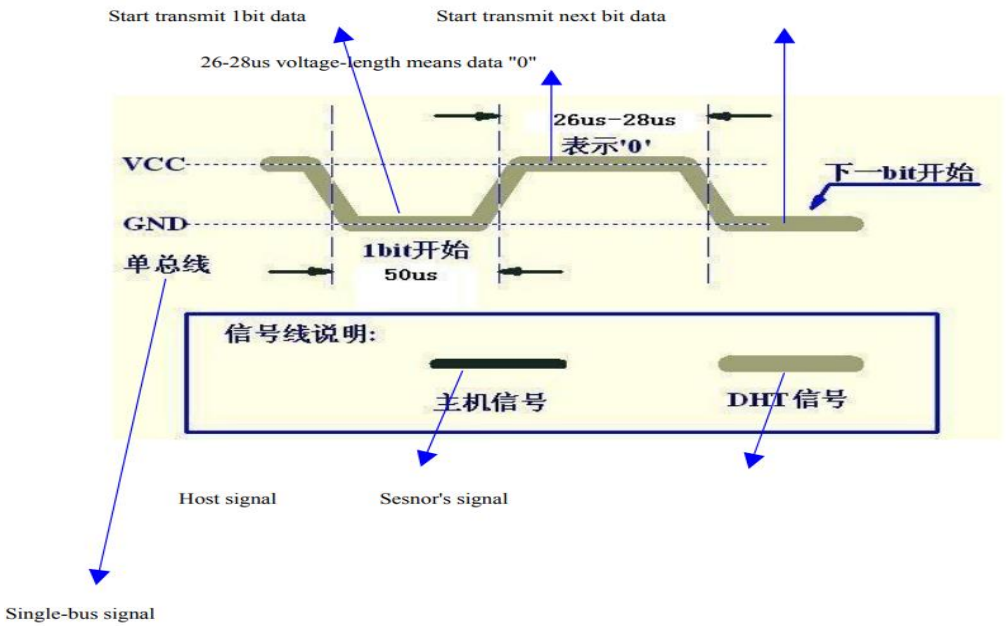
4.2. One Wire

One Wire (ang. Jeden Przewód) jak sama nazwa wskazuje jest protokołem komunikacyjnym używającym pojedynczego przewodu do transmisji danych. Sam sposób wymiany informacji zbliżony jest do I^2C z tym wyjątkiem że nie ma linii zegarowej którą generowałby układ nadrzędny. Protokół ten posiada zarówno zalety jak i wady w porównaniu z I^2C.

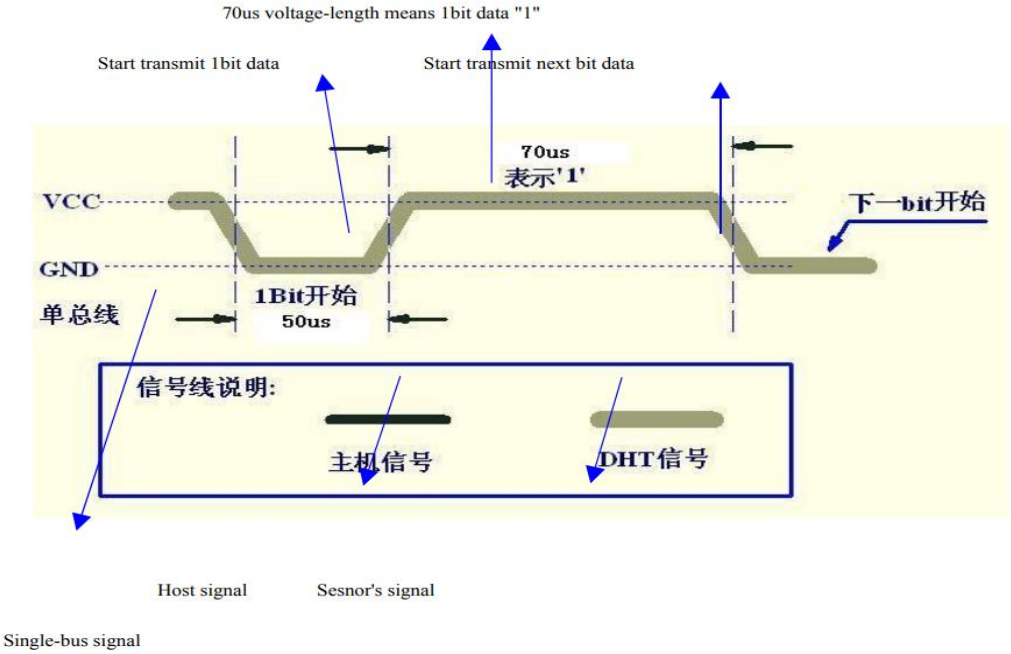
Najważniejszą zaletą One Wire jest fakt, że mikroprocesor do nawiązania komunikacji z danym podzespołem używającym tej magistrali, może użyć dowolnego portu wejścia / wyjścia cyfrowego (czyli GPIO). Co jest ważne w momencie, kiedy ograniczona jest ilość portów mikrokontrolera do zagospodarowania przez użycie w projekcie większej ilości podzespołów. Kolejnym aspektem pozytywnie wpływającym na decyzję o zastosowaniu urządzeń wykorzystujących 1-Wire jest niski poziom skomplikowania budowy samej magistrali. Przesłanie lub odebranie jednego bitu polega na podaniu na magistralę w pierwszej kolejności

logicznego stanu niskiego o odpowiedniej długości, a następnie stanu wysokiego również o określonej długości. W zależności od czasu trwania zarówno stanu niskiego jak i wysokiego przesyłany zostaje bit o wartości „0” lub „1”, jest to równoznaczne z tym że do przesłania bitu należy wygenerować impuls o ściśle określonym okresie jak i wypełnieniu.

Niestety jak każdy protokół i w tym przypadku znajdą się wady, których nie można pominąć. Przede wszystkim zastosowanie tylko jednego przewodu do komunikacji nie może zapewnić stabilności przesyłanych danych, a sam błąd (jeżeli wystąpi) jest bardziej problematyczny do wykrycia. Brak stabilności bezpośrednio wiąże się z brakiem możliwości przesyłania danych na większe odległości. Większa odległość powodowałaby większe ryzyko zniekształcenia przesyłanego sygnału, który posiada ściśle określone przedziały czasowe. Zakłócenie ich spowodowałoby błędny odczyt wszystkich pozostałych danych w przesyłanej ramce.

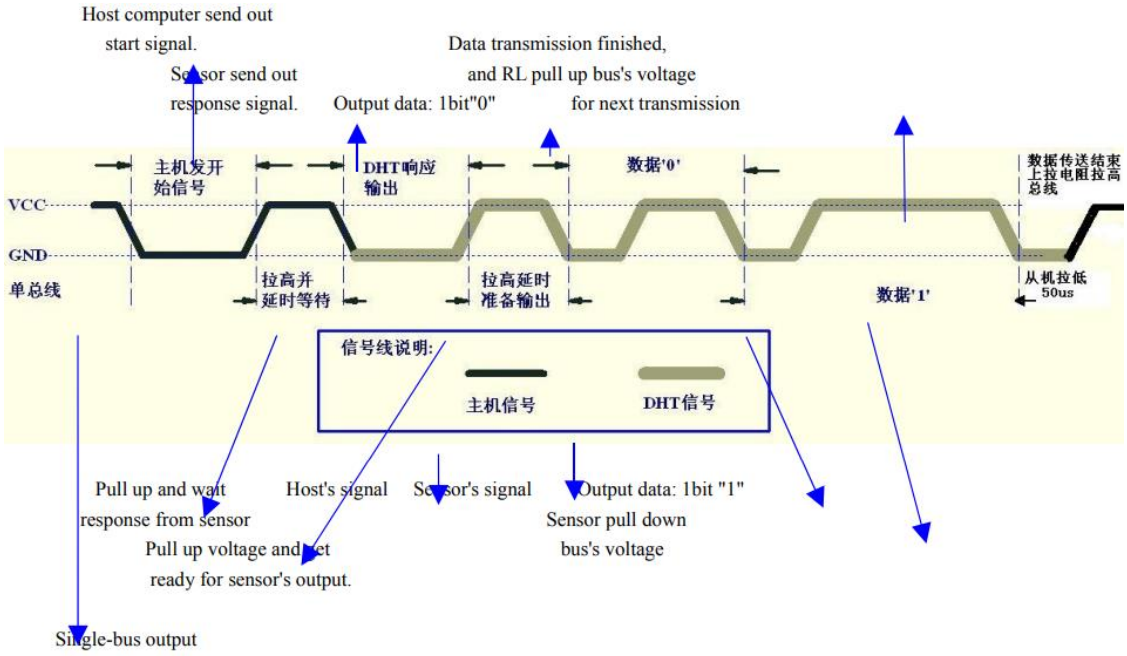


Rys. 4.6. Przebieg czasowy bitu „0” w protokole 1-Wire [6]



Rys. 4.7. Przebieg czasowy bitu „1” w protokole 1-Wire [6]

Rysunek 4.6 przedstawia sposób przesłania bitu o wartości „0” dla czujnika temperatury i wilgotności DHT22, przedstawiony w nocie katalogowej [6]. W pierwszej kolejności podawany jest na magistralę niski stan logiczny trwający 50 µs, a następnie stan wysoki trwający od 26 do 28 µs. Na rysunku 4.7 przedstawiony jest sposób przesłania bitu „1”, wyglądającego podobnie jak bit „0”, z tym wyjątkiem, że czas trwania wysokiego stanu logicznego trwa 70 µs. Warto zaznaczyć, że opisane przebiegi czasowe odnoszą się tylko i wyłącznie do czujnika DHT22, lecz doskonale obrazują zasadę działania protokołu 1-Wire. W innych urządzeniach czasy trwania poszczególnych stanów logicznych mogą się różnić, lecz zasada działania przesyłania poszczególnych bitów pozostaje ta sama.



Rys. 4.8. Schemat ramki danych protokołu 1-Wire [6]

Na rysunku 4.8 przedstawiony jest sygnał ramki danych składającej się z 2 bitów dla czujnika temperatury i wilgotności DHT22 [6], który użyty został w projekcie. Do rozpoczęcia komunikacji układ nadrzędny musi podać sygnał START, czyli podać na magistralę niski stan logiczny przez określony czas. W przypadku tego czujnika jest to co najmniej 1 ms. Następnie zmienia stan logiczny linii na wysoki i czeka od 20 do 40 µs na odpowiedź układu podrzędnego. Wtedy czujnik wysyła potwierdzenie rozpoczęcia transmisji podając stan „0” na magistralę o określonym czasie (tutaj jest to 80 µs), a następnie stan „1” trwający tyle samo. Po czym następuje faktyczna transmisja danych składająca się z przesłania bitu „0” oraz „1”. Kiedy układ nadrzędny uzna, że przesłane zostały wszystkie dane wówczas podaje na linię wysoki stan logiczny będący sygnałem STOP, który zakończy transmisję. Jak wspomina sam autor noty katalogowej [6] powyższy rysunek symbolizuje standardowy wygląd ramki danych w protokole 1-Wire. Co za tym idzie czasy trwania poszczególnych stanów logicznych na rysunku 4.8 w niektórych miejscach różnią się od tych, które zostały tutaj opisane. Jest tak ponieważ dotyczą one jedynie czujnika DHT22, będącego przykładem opisywanego protokołu.

4.3. Protokół równoległy

Protokół równoległy stosowany jest przede wszystkim do komunikacji wewnątrz-urządzeniowej. W przypadku mikrokontrolerów używa się go najczęściej do komunikacji między mikroprocesorem a wyświetlaczem ciekłokrystalicznym LCD (ang. Liquid Crystal Display). Jedną z cech podstawowych tego protokołu jest szerokość. Odnosi się ona do ilości linii danych używanych do wymiany informacji miedzy układem nadrzędnym a podrzędnym. Często spotykane są układy posiadające 4, 8 lub 16 linii danych. Magistrala protokołu równoległego poza liniami danych składa się również z linii specjalnych, generowanych przez układ nadrzędny do określenia trybu pracy podzespołu, jak również do potwierdzania przesłania bądź odebrania ramki danych. Ilość linii specjalny oraz to jakie pełnią one funkcje nie jest ściśle określona normą i w przypadku różnych producentów podzespołów używających protokół równoległy zachodzą pewne różnice zarówno w ilości, zastosowaniu, jak i terminologii poszczególnych linii. Dlatego dalszy opis skupi się na przedstawieniu protokołu równoległego używanego przez ILI9341, czyli sterownik wyświetlacza TFT LCD zastosowanego w projekcie.

Sterownik ILI9341 posiada 8 linii danych oraz 5 linii specjalnych, które przedstawione zostały w tabeli 4.1. Ze względu na różnice w terminologii przyjętej przez producenta układu ILI9341, a producenta płytki z wyświetlaczem dotykowym użytej w projekcie, w tabeli znalazły się dwa opisy wyprowadzeń. Dalsza część rozdziału zawierać będzie terminologię podjętą przez producenta ILI9341, która znajduje się w nocie katalogowej produktu [7] ze względu na opis stanów logicznych oraz przebiegów czasowych.

Tabela. 4.1. Opis wyprowadzeń wyświetlacza TFT LCD

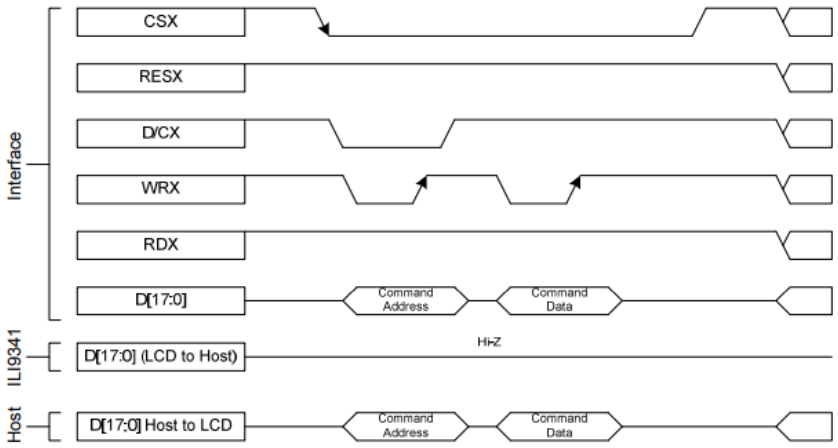
Nazwa linii ILI9341	Nazwa linii w projekcie	Zastosowanie
CSX	LCD_CS	Pin załączający komunikację między procesorem a wyświetlaczem, stan niski rozpoczyna transmisję danych
RES	LCD_RST	Odpowiada za reset sterownika i powrót do ustawień fabrycznych, podanie stanu niskiego powoduje reset
D/CX	LCD_RS	Wybór między rodzajem przesyłanych informacji są nimi dane („1”), lub komendy („0”)
WRX	LCD_WR	Pin sterujący zapisem danych do ILI9341
RDX	LCD_RD	Pin sterujący odczytem danych z ILI9341
D0	LCD_D0	Bit 0 w magistrali protokołu równoległego
D1	LCD_D1	Bit 1 w magistrali protokołu równoległego
D2	LCD_D2	Bit 2 w magistrali protokołu równoległego
D3	LCD_D3	Bit 3 w magistrali protokołu równoległego
D4	LCD_D4	Bit 4 w magistrali protokołu równoległego
D5	LCD_D5	Bit 5 w magistrali protokołu równoległego
D6	LCD_D6	Bit 6 w magistrali protokołu równoległego
D7	LCD_D7	Bit 7 w magistrali protokołu równoległego

Do rozpoczęcia komunikacji ze sterownikiem ILI9341 należy w pierwszej kolejności podać niski stan logiczny na linię CSX, co spowoduje rozpoczęcie komunikacji z układem podrzędnym. Następnie należy ustawić stan niski na RES i po czasie 1 ms zmienić na wysoki, co wywoła reset wszystkich ustawień sterownika. Po wykonaniu tych operacji rozpocznie się część właściwa gdzie zapisywane i odczytywane będą poszczególne rejestry układu podrzędnego.

Samo przesyłanie informacji zaczyna się od podania odpowiednich stanów logicznych na linie danych (D0 – D7), w tym procesie ramka danych, czyli jeden bajt, rozbijana jest na poszczególne bity. Każdy bit odpowiada stanowi logicznemu danej linii, gdzie bit najmłodszy odnosi się do D0, a bit najstarszy do D7. Dla przykładu, jeżeli procesor chce wysłać ramkę o wartości binarnej B01000001, wtedy na liniach D6 i D0 pojawi się stan wysoki, a na wszystkich pozostałych niski.

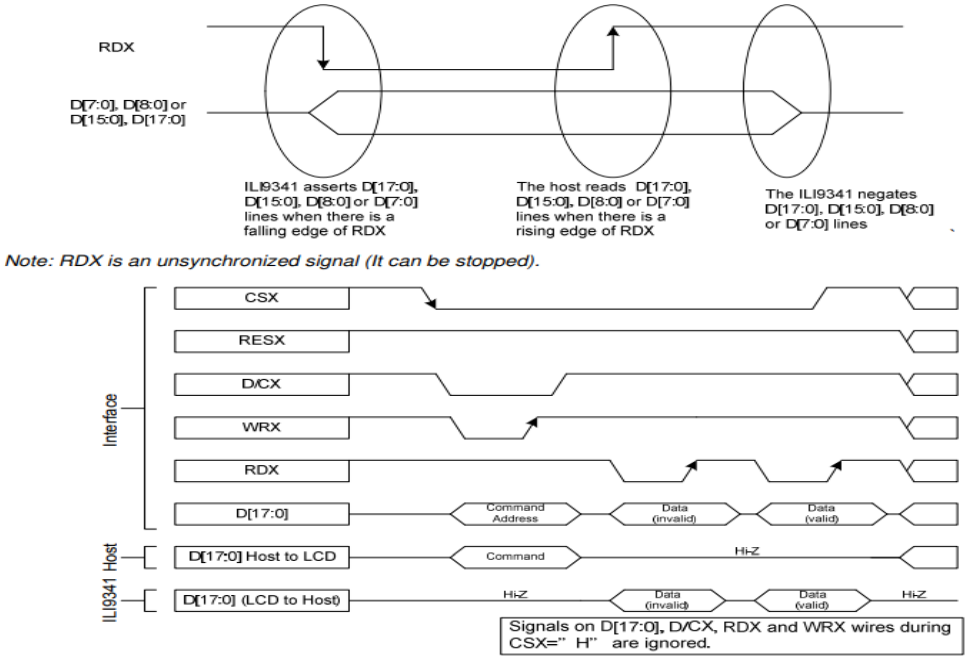
Całość komunikacji poza ustawieniem linii danych składa się także z odpowiedniego ustawienia linii specjalnych poza tymi które zostały uprzednio wspomniane, czyli D/CX, WRX i RDX. Informacje w protokole równoległym zarówno w przypadku ILI9341 jak i innych sterowników, można podzielić na dwa typy. Pierwszym są komendy czyli np. adresy rejestrów

czy adresy komórek wyświetlacza, a drugim dane czyli wartości jakie przyjmują rejestry bądź wartości określające kolor danej komórki wyświetlacza. Zależnie od rodzaju informacji jaką ma się zamiar przesłać należy podać odpowiedni stan logiczny na linię D/CX, gdzie „1” oznacza dane a „0” oznacza komendy. Kolejna linia to WRX i używana jest w momencie gdy układ nadrzędny chce przesłać informacje do podrzędnego, wtedy po odpowiednim ustawieniu linii danych należ podać na WRX zbocze narastające, które spowoduje przyjęcie danych przez układ podrzędny. Podobną zasadą działania charakteryzuje się linia RDX, z tą różnicą, że służy ona do odczytu danych z ILI9341. Gdy procesor wyślę prośbę o odczytaniu wybranego rejestru, wówczas w momencie podania zbocza narastającego odczytane zostaną przez niego linie danych, a następnie zdekodowane zostaną stany logiczne do zmiennej 8-bitowej.



Rys. 4.9. Przebiegi czasowe zapisu danych do układu ILI9341 [7]

Znając wszystkie wyprowadzenia aktualnie używane w protokole równoległym można omówić proces wysyłania informacji przedstawiony na rysunku 4.9. W pierwszej kolejności układ nadrzędny musi ustawić adres rejestru, który chce zapisać. Czyli podaje stan niski na linię D/CX (ponieważ jest to komenda) oraz podaje stan niski na WRX. Następnie ustawiony zostaje adres rejestru na liniach danych zależnie od jego wartości, po czym zmieniany jest stan linii WRX na wysoki co powoduje przyjęcie danych przez układ podrzędny. W dalszej kolejności procesor zmienia stan D/CX na wysoki ponieważ dalej przesyłane będą dane do rejestru i ponownie stan niski na WRX by przygotować się do wysłania ramki danych. Wtedy zmieniane zostają stany logiczne na liniach danych zależnie od bajta jaki ma być wysłany do układu podrzędnego, po czym na WRX ustawiony zostaje stan wysoki i sterownik ILI9341 nadpisuje wskazany wcześniej rejestr aktualnie przesłanym bajtem.



Rys. 4.10. Przebiegi czasowe odczytu danych z układu ILI9341 [7]

Schemat odczytu informacji (rys. 4.10.) zaczyna się podobnie jak w przypadku zapisu. Ponieważ wybrany musi zostać rejestr który procesor zamierza odczytać, więc należy przesłać jego numer w pierwszym bajcie (jako komendę). Po ustawieniu adresu rejestru, co zostało opisane w poprzednim akapicie, procesor rozpoczyna odczyt wartości danego rejestru. Wszystkie linie specjalne z wyjątkiem CSX przyjmują wysoki stan logiczny, po czym na RDX podaje niski stan logiczny. Wtedy linie danych stają się wejściami układu nadrzędnego, na które układ podrzędny poda wartość odczytywanego rejestru. W momencie kiedy procesor zmieni RDX na „1” powinien nastąpić w nim odczyt linii danych oraz zdekodowanie ich stanów logicznych do jednobajtowej zmiennej. Samą czynność odczytu procesor może wykonać wielokrotnie, wtedy odczytane zostaną wartości kolejnych rejestrów.

5. Elementy użyte w projekcie

Magistrale danych, które zostały opisane w poprzednim rozdziale, będą odnosić się do podzespołów, jakie zastosowano w projekcie. Rozdział ten poświęcony będzie omówieniu wszystkich elementów wchodzących w skład sterownika ogrodowego. Przedstawione zostaną ich najważniejsze parametry techniczne oraz funkcje jakie pełnią w odniesieniu do całości projektu.

5.1. Wyświetlacz TFT LCD z panelem dotykowym

Wyświetlacz ciekłokrystaliczny użyty w projekcie (rys. 5.1 i rys. 5.2) charakteryzuję się matrycą o przekątnej 2,8”, długością 320 pixeli i szerokością 240 pixeli. Posiada on dodatkowo takie samo rozmieszczenie wyprowadzeń jak Arduino Uno. Na rysunku 5.2 przedstawiony jest opis wyprowadzeń świadczący o tym, że w przypadku tego podzespołu użyty został protokół równoległy przeanalizowany w rozdziale 4.3 wraz z opisem wyprowadzeń. Linie których nie ma w tabeli 4.1, a znajdują się na rysunku 5.2, czyli te z przedrostkiem SD_, odnoszą się do magistrali komunikacyjnej SPI. Służy ona do zapisu / odczytu danych z karty SD, która nie jest używana w projekcie. Do obsługi protokołu równoległego użyty został układ scalony ILI9341. Jego parametry znajdują się w nocie katalogowej [7]. Użyty wyświetlacz dotykowy pozwala na odczyt w którym miejscu użytkownik dotknął matrycę, co pozwala na dodatkową interakcję. Interpretacja aktualnej pozycji przez układ nadrzędny polega na odczycie poziomu napięcia na liniach LCD_WR (pozycja x) i LCD_RS (pozycja y). Ze względu na fakt że użyty został wyświetlacz dotykowy rezystancyjny niezbędna jest transformacja poziomu napięcia na wartość binarną a następnie jej interpretacja na wartości od 0 do 240 w przypadku osi x oraz od 0 do 320 dla osi y. Zarówno za czynność interpretacji, jak i za obsługę protokołu równoległego odpowiedzialna jest gotowa biblioteka dedykowana dla tego wyświetlacza. Jest ona jedną ze wspomnianego zbioru bibliotek Arduino, o których mowa w rozdziale 3. Jej funkcje omówione zostaną w rozdziale 8 podczas omawiania kodu programu.

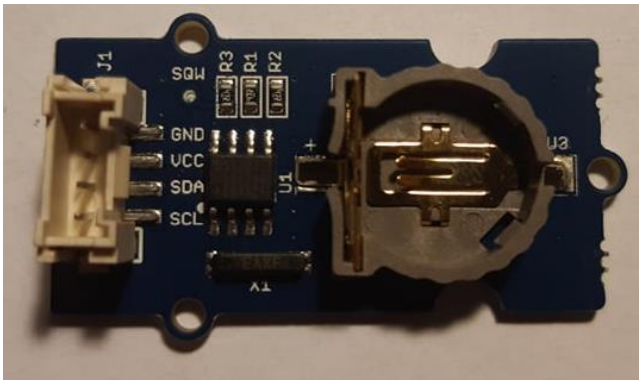
Rys. 5.1. Wyświetlacz TFT LCD (przód)

Rys. 5.2. Wyświetlacz TFT LCD (tył)

5.2. Zegar czasu rzeczywistego

Moduł RTC (ang. Real Time Clock) czyli moduł zegara czasu rzeczywistego użytego w projekcie przedstawiony został na rysunku powyżej (rys. 5.3). Jest to znany układ scalony DS1307. Do komunikacji wykorzystuje magistralę I2C i posiada standardowy układ rejestrów do zapisy/odczytu: daty, dnia tygodnia oraz godziny. DS1307 stanowi układ podrzędny z którym komunikować się będzie Arduino, a na podstawie odczytanych wartości wyświetlać

będzie aktualny czas na ekranie głównym. Dodatkowo użytkownik sterownika ogrodowego będzie miał możliwość zmiany daty i godziny przy użyciu wyświetlacza dotykowego.



Rys. 5.3. Zegar czasu rzeczywistego DS1307

5.3. Czujnik temperatury i wilgotności

Za odczyt aktualnych wskazać temperatury i wilgotności w projekcie odpowiedzialny będzie gotowy układ DHT22, który do procesu komunikacji wykorzystywać będzie magistralę One Wire. Na rysunku 5.4 widoczne są trzy wyprowadzenia, czyli zasilanie układu i linia danych, co również wskazuje na użyty protokół komunikacyjny. Dane które zostaną odczytane z tego układu podrzędnego przez Arduino będą wyświetlone na ekranie głównym, odnosząc się do zewnętrznych warunków atmosferycznych.

Rys. 5.4. Czujnik wilgotności i temperatury DHT22

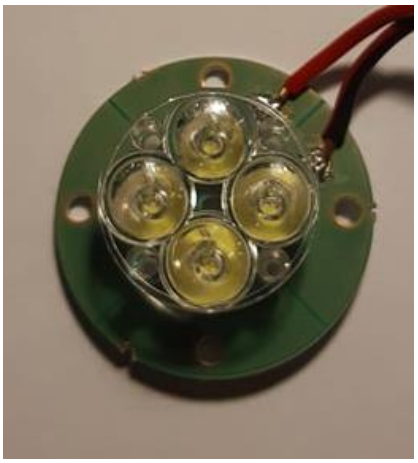
5.4. Pompa wody do podlewania

Do zasymulowania pracy pompki podlewającej uprawy w ogrodzie użyty zostanie silnik prądu stałego (rys. 5.5), który podczas pracy zasilany jest napięciem 12 V. Użytkownik dzięki wyświetlaczowi dotykowemu może wybrać godzinę rozpoczęcia i zakończenia podlewania, wtedy Arduino podaje na bramkę tranzystora MOSFET wysoki stan logiczny. Powoduje to wejście tranzystora w stan aktywny. Kiedy Arduino odczyta z RTC godzinę i minutę, którą użytkownik ustawił jako czas zakończenia podlewania, wtedy na bramkę podany zostanie niski stan logiczny. Tranzystor wchodzi w stan zatkania i pompka przestaje pracować.

Rys. 5.5. Silnik prądu stałego

5.5. Oświetlenie LED

Jako oświetlenie w projekcie sterownika ogrodowego użyta zostanie pojedyncza oprawa, przedstawiona na rysunku 5.5. Składa się ona z 4 diod LED połączonych szeregowo. Napięcie zasilania to także 12 V jak w przypadku silnika prądu stałego, z tym wyjątkiem że użytkownik może ustawić w odpowiednim podmenu jasność od 0% do 100%. Oprawa zasilana będzie przez tranzystor MOSFET, gdzie na bramkę Arduino podawać będzie sygnał PWM o określonym wypełnieniu zależnym od procentowego stopnia jasności wybranego przez użytkownika.



Rys. 5.6. Oprawa LED

5.6. Zasilacz

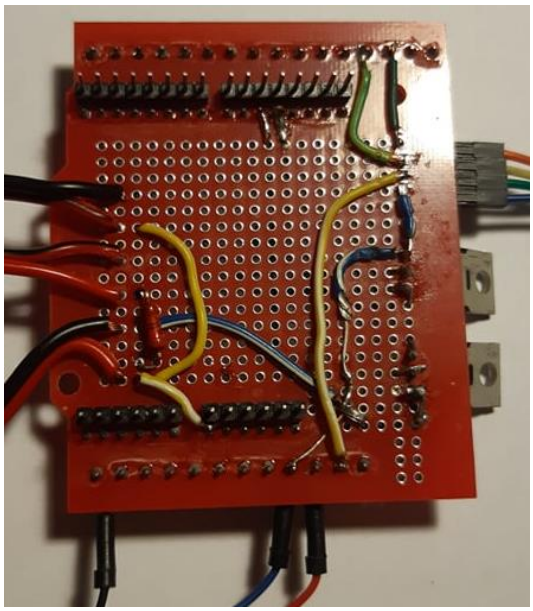
Do zasilenia całego układu zastosowany zostanie zasilacz impulsowy (rys. 5.7), którego napięcie wejściowe może wynosić od 100 V AC do 240 V AC. Na wyjściu zasilacza jest napięcie stałe o potencjale 12 V, z którego zasilane jest Arduino, oświetlenie LED, silnik prądu stałego oraz pozostałe podzespoły. Znamionowy prąd wyjściowy zasilacza to 1,5 A. Stanowi on spory zapas względem mocy pobieranej przez całe urządzenie.

Rys. 5.7. Zasilacz

5.7. Nakładka na wyświetlacz dotykowy i Arduino Uno

Nakładka przedstawiona na rysunku 5.8 podobnie jak wyświetlacz TFT LCD posiada wyprowadzenia kompatybilne z Arduino Uno, stanowiąc warstwę pośrednią między wyświetlaczem a Arduino. Od strony dolnej dołączona zostanie płytka Arduino Uno, a od strony

górnej dołączony będzie wyświetlacz dotykowy omówiony w rozdziale 5.1. Użycie nakładki pozwala na zachowanie ciągłości połączeń wszystkich wyprowadzeń mikrokontrolera z liniami komunikacyjnymi i liniami zasilającymi wyświetlacza. Ponadto nakładka posiada dodatkowe złącza do których można podłączyć dowolne podzespoły. Zostanie to wykorzystane do dołączenia DHT22 i DS1307. Na samej nakładce stworzony zostanie układ połączeń, w skład którego wchodzą 2 tranzystory MOSFET, na rysunku 5.8 po prawej stronie. Dodatkowo na nakładce umieszczone zostanie złącze 4 – pinowe nad tranzystorami, do którego przyłączone zostaną linie danych i zasilania do modułu zegara czasu rzeczywistego (I^2C). Na rysunku 5.8 po lewej stronie widoczne są 3 kable zasilające. Górny kabel zasilać będzie pompkę wody (silnik prądu stałego), środkowy kabel służyć będzie do zasilania oświetlenia LED, a dolny stanowić będzie napięcie wejściowe (z zasilacza) i zasili cały sterownik. Sam układ połączeń omówiony zostanie w rozdziale 6 na podstawie schematu ideowego sterownika.



Rys. 5.8. Zmodyfikowana nakładka na Arduino

5.8. Arduino

Przedstawiony w rozdziale 3 moduł Arduino Uno (rys. 3.1), jest układem nadrzędnym w procesie komunikacji z pozostałymi elementami sterownika ogrodowego. Jego głównym zadaniem są:

- odczyt wskazań temperatury i wilgotności z czujnika DHT22,
- odczyt i możliwość zmiany aktualnej daty i godziny z układu DS1307,
- generacja sygnału PWM sterującego jasnością oświetlenia,

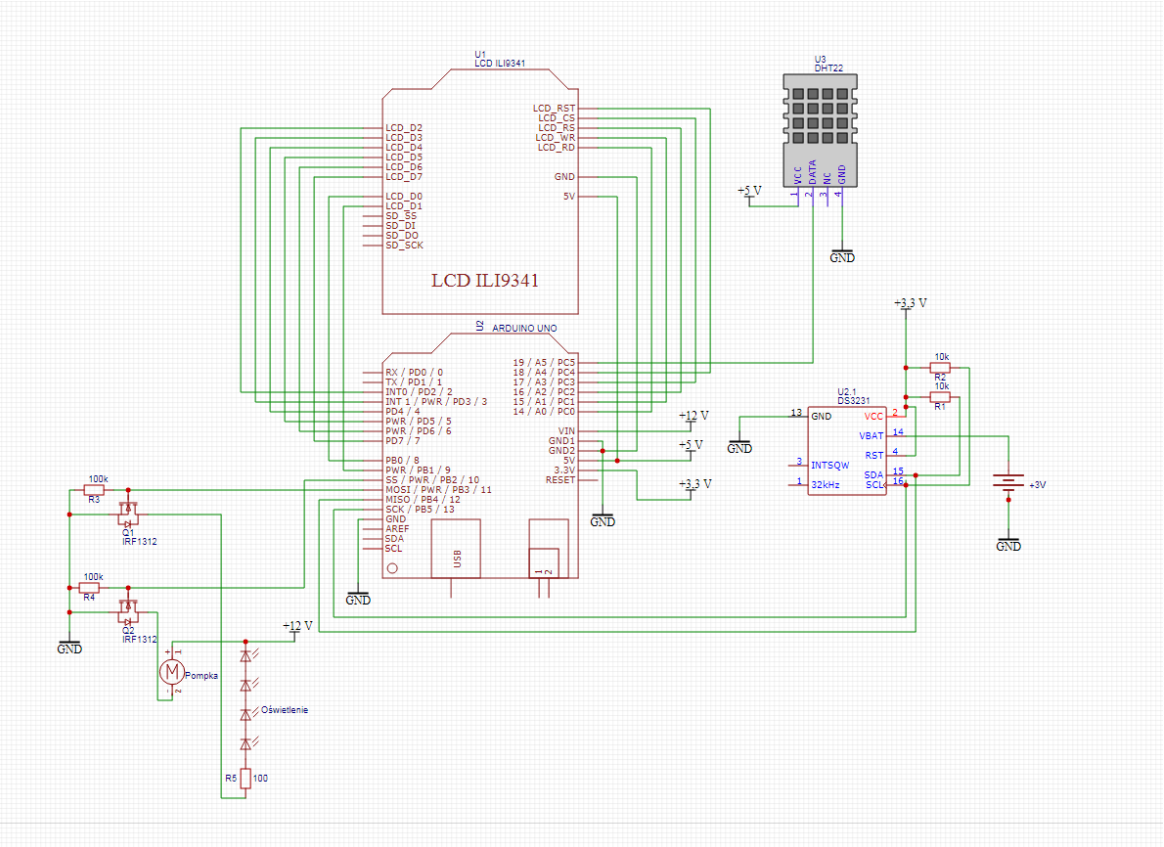
- załączanie / wyłączanie silnika prądu stałego imitującego pompkę do podlewania ogrodu,
- wyświetlanie odpowiednich komunikatów na wyświetlaczu dotykowym zależnie od: aktualnego stanu pracy, wskazań układów podrzędnych i odczytów wyświetlacza dotykowego.

Arduino zaprogramowane jest do wykonania powyższych zadań w odpowiedniej rutynie, eliminującej powstawanie konfliktów między wykonywanymi procesami. Dodatkowo przewidziane jest w programie dwupoziomowe menu, które w sposób czytelny pozwoli użytkownikowi na:

- określenie godziny rozpoczęcia i zakończenia podlewania,
- ustawienie jasności oświetlenia na zewnątrz budynku,
- zmianę aktualnej daty lub godziny.

6. Schemat ideowy instalacji ogrodowej

Po omówieniu użytych podzespołów można szerzej opisać ich sposób połączenia. Do zasilania układu użyty został zasilacz o napięciu wyjściowym 12 V DC. Na schemacie ideowym sterownika (rys. 6.1) podłączony jest on do wejścia Arduino VIN. Analizując dalej schemat można zauważyć, że w całym układzie znajdują się trzy napięcia o wartościach:12 V, 5 V oraz 3,3 V. Dwa ostatnie napięcia generowane są przez stabilizatory liniowe znajdujące się na płytce Arduino Uno. Ich zadaniem jest zasilenie podzespołów napięciem o odpowiedniej dla nich wartości.



Rys. 6.1. Schemat ideowy instalacji ogrodowej

Przechodząc do omówienia układów podrzędnych warto zacząć od wyświetlacza TFT z ekranem dotykowym, opartego na sterowniku ILI9341 (widoczny w górnej części schematu). Jest on połączony bezpośrednio do kolejnych wyjść Arduino Uno i związane z jest to zastosowanym w projekcie modelem płytki wyświetlacza zachowującym kompatybilność względem rozmieszczenia wyprowadzeń Arduino. Linie danych oznaczone od LCD_D0 do LCD_D7 widoczne po lewej stronie układu połączone są w mikrokontrolerze do wyjść

cyfrowych oznaczonych od 2 do 9. Linie specjalne widoczne po prawej stronie układu ILI9341 podłączone są do wyjść mikroprocesora z oznaczeniami od A0 do A4. Są to piny mogące także pełnić funkcję wejść ADC, natomiast w tym projekcie użyte zostały jako wyjścia cyfrowe. Zarówno wyświetlacz jak i jego sterownik zasilane są napięciem o potencjale 5 V.

Kolejnym układem podrzędnym przedstawionym w poprzednim rozdziale jest zegar czasu rzeczywistego zrealizowany na układzie scalonym DS1307. Układ ten widoczny po prawej stronie schematu, zasilony jest napięciem o potencjale 3,3 V, zgodnie z zaleceniami producenta [8]. Przechodząc do dalszych wyprowadzeń tego układu, widoczne na schemacie oznaczenie VBAT, połączone jest do baterii litowo-jonowej o napięciu 3 V. Zastosowanie baterii pozwala na zachowanie i ciągłą aktualizację rejestrów przechowujących datę oraz czas, mimo braku zasilania ze strony zasilacza lub Arduino. Widoczna dalej linia RST odnosi się do resetu rejestrów do wartości fabrycznych. Ze względu na brak potrzeby resetowania w trakcie działania układu jest połączona do +3,3 V. Linie na samym dole układu DS1307 to SDA i SCL czyli linie danych magistrali I^2C. Połączone są one z portami cyfrowych wejść / wyjść mikrokontrolera o numerach 12 i 13. Dodatkowo są one połączone przez rezystory podciągające do napięcia 3,3 V. Wartość ich rezystancji zgodnie z zaleceniami producenta to 10 kΩ, a samo zastosowanie tych rezystorów ma na celu poprawienie stabilności komunikacji.

Ostatnim układem komunikującym się z mikroprocesorem jest czujnik DHT22 widoczny w prawym górnym rogu. Jak zostało wspomniane w poprzednim rozdziale układ ten używa do komunikacji protokół One Wire. Posiada on dzięki temu widoczne na schemacie tylko 3 wyprowadzenia, czyli zasilanie oraz linię danych. Jak widać na rysunku 6.1 jego napięcie zasilania wynosi 5 V, a linia danych połączona jest do wejścia Arduino z oznaczeniem A5.

Poza układami komunikującymi się z Arduino użyte zostały elementy wykonawcze odpowiedzialne za pracę pompki (czyli w tym projekcie silnika prądu stałego) oraz jasność świecenia oprawy LED. Mowa tutaj o elementach w lewym dolnym rogu schematu. Warto dodać że napięcie przy jakim pracuje zarówno pompka jak i oświetlenie to 12 V i jest ono podłączone bezpośrednio do wyprowadzenia dodatniego obu odbiorników. Przechodząc do dokładniejszej analizy omówiony zostanie przypadek oświetlenia. Mikrokontroler steruje jasnością oprawy LED podając sygnał PWM na wyjściu cyfrowym Arduino o numerze 11. Sygnał doprowadzony zostaje do bramki tranzystora MOSFET o symbolu IRF1312. W momencie kiedy wypełnienie sygnału jest większe niż 0 % na linii 11 występuje wysoki stan logiczny czyli 5 V. Kiedy na bramce pojawia się zbocze narastające tranzystor przechodzi w stan przewodzenia. Wtedy na drenie pojawia się potencjał masy układu. Obwód oświetlenia zostaje zamknięty i zaczyna płynąć przez niego prąd. W celu ograniczenia prądu płynącego

przez oprawę LED między drenem, a anodą pierwszej diody zastosowany jest rezystor o wartości 100 Ω. Kiedy na linii numer 11 pojawia się stan logiczny niski wtedy tranzystor wchodzi w stan zatkania. Aby zapewnić stabilną zmianę stanu pracy tranzystora do bramki dołączony jest rezystor podciągający do masy o rezystancji 100 kΩ. Kiedy przez mikrokontroler generowany jest sygnał PWM, na przewodach zasilających LED pojawia się sygnał PWM o amplitudzie 12 V oraz częstotliwości i wypełnieniu tym samym jakie jest generowane przez mikroprocesor. Jako że częstotliwość wynosi 500 Hz to zależnie od wypełnienia ustawiana jest jasność na diody podawane zostają tylko 2 potencjały czyli 0 V i 12 V. Wysoka częstotliwość powoduje że użytkownik jest w stanie stwierdzić zmianę jasności ze względu na zjawisko bezwładności oka ludzkiego którego granica częstotliwości przy której jest w stanie wychwycić zmianę obrazu to 200 Hz.

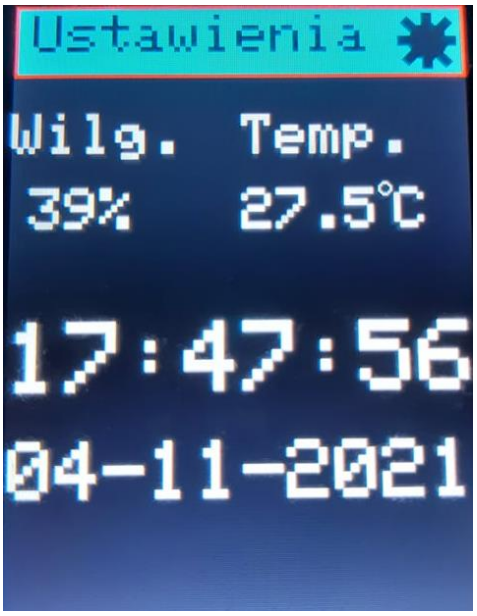
Sterowanie pracą pompki opiera się na podobnej zasadzie działania co w przypadku oświetlenia, z tą różnicą że nie ma tutaj generowanego sygnały PWM, a sam tranzystor MOSFET sterowany jest z wyjścia cyfrowego Arduino z numerem 10. W tym przypadku również wykorzystano IRF1312 oraz rezystor podciągający do masy (którego oporność to 100 kΩ), aby zachować stabilne przejście tranzystora, ze stanu przewodzenia, do stanu zatkania. Kiedy na bramkę podany zostanie wysoki stan logiczny wtedy między przewodem plusowym pompki, a przewodem minusowym (na drenie tranzystora) pojawi się różnica potencjałów 12 V. Wtedy pompka, czyli w projekcie użyty silnik prądu stałego, zacznie pracować.

7. Interfejs użytkownika

Ta część pracy poświęcona będzie opisowi interfejsu użytkownika. Poniżej przedstawione zostaną wszystkie zrzuty z ekranu sterownika ogrodowego. Dodatkowo znajdzie się również analiza poszczególnych komunikatów oraz opis wszystkich możliwych interakcji jakie użytkownik może podjąć. W tym rozdziale nastąpi opis przycisków wraz z funkcjami jakie pełnią, a także dokładne omówienie menu oraz opcji wewnątrz niego.

7.1. Panel główny

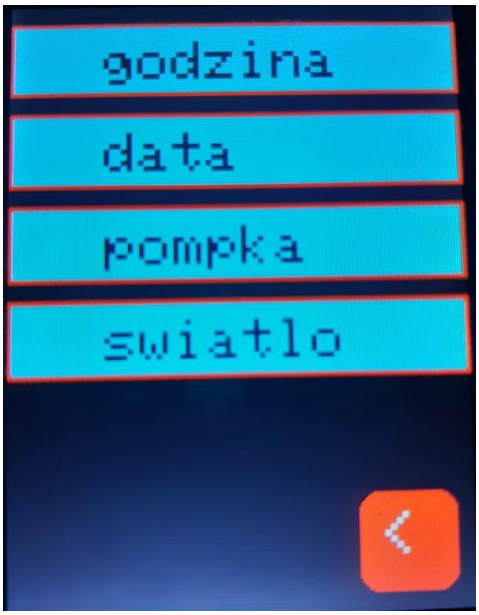
Ekran panelu głównego (rys. 7.1) wyświetla się od raz po zasileniu sterownika, znajdują się na nim aktualne wskazania temperatury i wilgotności z czujnika DHT22 oraz aktualny czas i data odczytywane z układu zegara czasu rzeczywistego DS1307. Wskazania na ekranie głównym aktualizowane są w trakcie każdego cyklu mikroprocesora. A w momencie kiedy jedno ze wskazań różni się od tego jakie było w poprzednim cyklu, wtedy część wyświetlacza na której zapisana jest liczba, zostaje wypełniona kolorem tła wyświetlacza. Następnie przesunięty zostaje znacznik na docelowe pole w którym ma nastąpić nadpisanie, po czym procesor zapisuje aktualne wskazanie z danego czujnika. Takie działanie programu ma na celu zwiększenie częstotliwość pracy mikrokontrolera. W momencie kiedy przy aktualizacji wartości nadpisywana byłaby cała matryca (zamiast przykładowo 30 pikseli na których nastąpiła zmiana wartości), wtedy przy przeliczeniu ilości wszystkich pikseli, czyli 76800 px, wychodzi że czas trwania zapisu byłby dłuższy o 2560 razy. Na ekranie głównym znajduje się przycisk ustawień, gdy użytkownik go wciśnie nastąpi przejście do menu głównego.



Rys. 7.1. Ekran panelu głównego

7.2. Panel menu głównego

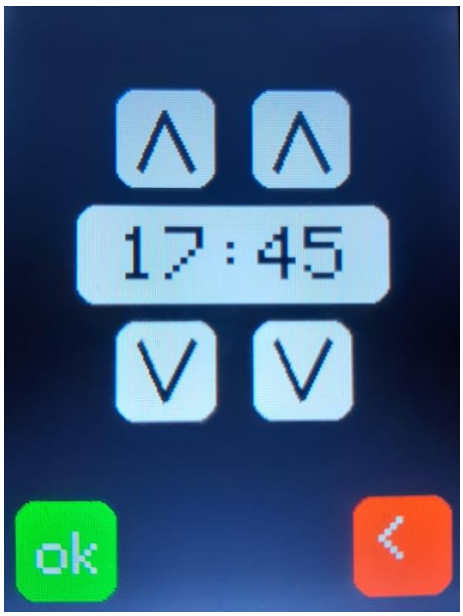
Po wciśnięciu przycisku ustawień ekran zostaje wyczyszczony i wyświetlone zostaje 5 przycisków (rys. 7.2). Wciśnięcie przycisków w górnej części wyświetlacza czyli: „godzina”, „data”, „pompka” i „światło” powoduje przejście do odpowiedniego podmenu. Wciśnięcie czerwonego przycisku w prawym dolnym rogu wyświetlacza powoduje powrót do ekranu głównego.



Rys. 7.2. Ekran panelu menu głównego

7.3. Panel zmiany ustawień godziny

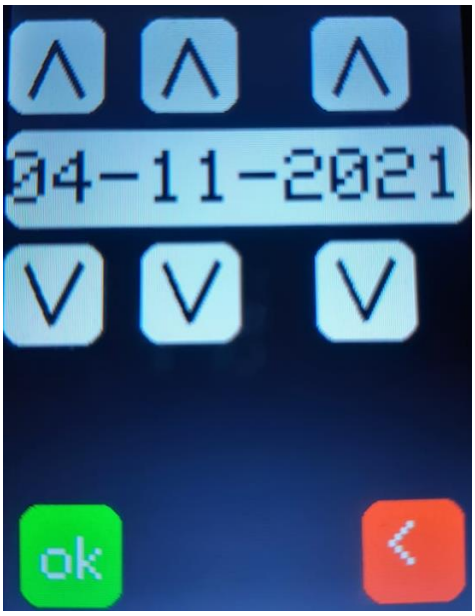
Na rysunku 7.3 przedstawione zostało podmenu do zmiany godziny i minuty. Widoczne na nim jest aktualne wskazanie godziny, czyli wartość w dużym prostokącie w centralnej części wyświetlacza po lewej stronie. Wskazanie minuty umiejscowione jest po prawej stronie tego prostokąta. Za zmianę godziny odpowiedzialne są 2 przyciski umiejscowione nad i pod wyświetlaną wartością godziny. Przycisk górny powoduje zwiększenie licznika godziny. W momencie kiedy licznik osiągnie wartość 23 dalsze zwiększanie nie będzie możliwe. Przycisk dolny wywołuje zmniejszanie licznika w którym przechowywana jest wartość ustawianej godziny, do momentu gdy jego wartość osiągnie 0. Na podobnej zasadzie działają przyciski nad i pod wskazaniem minuty - z tą różnicą, że zmieniana wartość licznika minut nie może być większa niż 59. Wciśnięcie zielonego przycisku w lewym dolnym rogu powoduje że mikrokontroler nadpisuje rejestry DS1307. Nastąpi wtedy faktyczna zmiana godziny i minuty, po czym wyświetlone zostanie menu główne. Jeżeli użytkownik nie zamierza zmieniać godziny i minuty wtedy powinien wcisnąć czerwony przycisk w prawym dolnym rogu. Spowoduje to powrót do menu głównego bez nadpisywania rejestrów RTC.



Rys. 7.3. Ekran panelu menu ustawień godziny

7.4. Panel zmiany ustawień daty

W tym podmenu użytkownik ma możliwość zmiany daty. Na rysunku 7.4 widoczny jest duży biały prostokąt w którym wyświetlone są aktualne wartości licznika dnia miesiąca (po lewej stronie), miesiąca (w środku) oraz roku (po prawej stronie). Nad i pod każdym wskazaniem danego licznika wyświetlone są przyciski (strzałki), służące do zwiększania lub zmniejszania danego licznika. Zasada działania jest podobna jak w przypadku podmenu do wyboru godziny, z tą różnicą, że licznik dnia miesiąca nie może być większy niż 31, licznik miesiąca większy niż 12, a licznik roku nie może być mniejszy niż 21. Rejestr przechowujący rok jest 8-bitowy. Wynika to z założenia, iż rok nie może być mniejszy niż 2000. Stąd w rejestrze przechowywane są 3 ostatnie cyfry roku. Jako że rok w którym został napisany kod programu to 2021, niemożliwym jest ustawienie niższej wartości niż 21. Na dole wyświetlacza znajdują się 2 przyciski pełniące tą samą funkcję co te opisane w podmenu ustawień godziny. Jedyną różnicą jest to że przycisk zielony powoduje nadpisanie rejestrów daty do układu DS1307.



Rys. 7.4. Ekran panelu menu ustawień daty

7.5. Panel zmiany ustawień podlewania

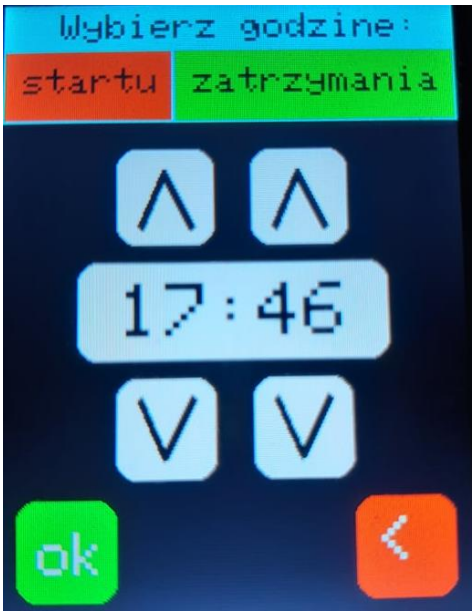
Po wciśnięciu przez użytkownika przycisku „pompka” w menu głównym (rys. 7.2) następuje przejście do podmenu przedstawionego na rysunku 7.5. Użytkownik może ustawić w

nim czas rozpoczęcia podlewania. Zasada działania wyboru godziny jest taka sama jak dla podmenu z rozdziału 7.3. Różnica w działaniu polega na tym, że po ustawieniu danej godziny rozpoczęcia i nie ustawieniu godziny zakończenia, wciśnięcie zielonego przycisk „ok" nie wywoła żadnej reakcji. Takie rozwiązanie, co jest logicznie uzasadnione, blokuje możliwość zatwierdzenia samej godziny rozpoczęcia, ponieważ od momentu startu pompki nie nastąpiło by jej wyłączenie. Na ekranie omawianego podmenu znajdą się jeszcze dwa przyciski: „startu" i „zatrzymania". Tłem tekstu jednego z przycisków jest kolor zielony, a drugiego czerwony. Zielony kolor tła oznacza że dana godzina (startu lub zatrzymania pracy pompki) jest aktualnie wybierana przez użytkownika. W momencie kiedy wciśnie się przycisk którego tło ma czerwony kolor wtedy odpowiednio nastąpi zmiana wyboru godziny startu / zatrzymania.

Rys. 7.5. Ekran panelu menu ustawień czasu podlewania cz. 1

Dla przykładu (rys. 7.5) kiedy użytkownik wciśnie przycisk „zatrzymania" wtedy nastąpi zmiana kolorów tła obu przycisków na przeciwne co zostało przedstawione na rysunku 7.6. Użytkownik będzie mógł wybrać godzinę zatrzymania pracy pompki przy użyciu 4 przycisków na których znajdują się symbole strzałek. Dopiero po wyborze dogodnej godziny i minuty zarówno początku i końca pracy, możliwe będzie zatwierdzenie tych wartości zielonym przyciskiem w lewym dolnym rogu matrycy. Po zatwierdzeniu nastąpi przejście do menu głównego. Jeżeli użytkownik by się rozmyślił, to w każdym momencie możliwe jest wyjście do menu głównego (rys. 7.2). Należy wtedy wcisnąć czerwony przycisk w prawym dolnym rogu. Po zatwierdzeniu czasu pracy pompki, mikroprocesor załączy odpowiednie wyjście sterujące,

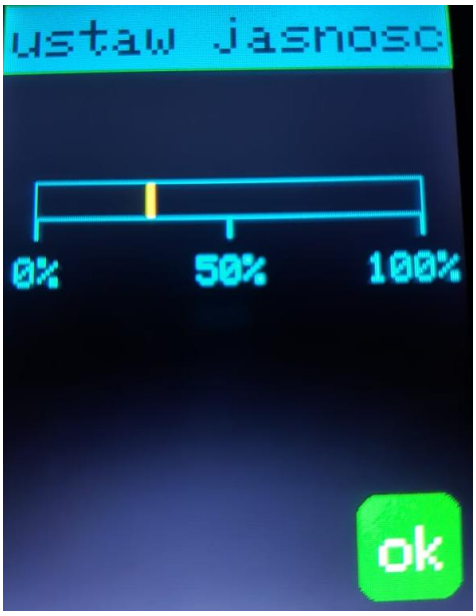
kiedy godzina odczytana z RTC, będzie taka sama jak ta którą użytkownik wskazał jako godzinę rozpoczęcia. Kiedy mikrokontroler odczyta z DS1307 godzinę w której ma nastąpić koniec pracy, wtedy na wyjściu sterującym pojawi się niski stan logiczny a pompka zostanie wyłączona.



Rys. 7.6. Ekran panelu menu ustawień czasu podlewania cz. 2

7.6. Panel zmiany ustawień oświetlenia

Aby przejść do podmenu ustawienia jasności (rys. 7.7), należy w menu głównym (rys. 7.2) wybrać opcję „swiatlo”. Możliwe jest tutaj ustawienie jasności w zakresie od 0 % do 100 %. Na środku wyświetlacza widoczny jest pasek, którego dotknięcie w dowolnym miejscu, powoduje ustawienie procentowego wypełnienia sygnału PWM sterującego jasnością LED. Po dotknięciu paska w dowolnym miejscu, sygnał PWM podany zostaje na odpowiednie wyjście Arduino. O aktualnej jasności oprawy LED informuje żółty prostokąt wewnątrz paska wyboru jasności. Jego położenie zmienia się przy każdym przyciśnięciu paska. Jeżeli użytkownik dotknie miejsce na lewo od paska (mniej niż 0 %), wtedy Arduino ustawi logiczny stan niski na wyjściu sterującym oprawą LED. Wskaźnik pozycji, czyli żółty prostokąt ustawiony zostanie na pozycję 0% . Dotknięcie miejsca na prawo od paska jasności (więcej niż 100 %), spowoduje podanie stanu wysokiego na wyjście sterujące. Oprawa będzie świeciła ze swoją pełną jasnością i wskaźnik pojawi się na pozycji 100%. Aby wyjść z tego podmenu należy kliknąć zielony przycisk „ok” w prawym dolnym rogu. Nastąpi wtedy przejście do menu głównego.



Rys. 7.7. Ekran panelu menu ustawień jasności

8. Kod programu oraz opis funkcji

Omówiony interfejs użytkownika, przedstawiony w poprzednim rozdziale jest efektem działającego programu, który został wgrany na Arduino Uno. Całość tego rozdziału będzie poświęcona omówieniu kodu źródłowego programu. Dodatkowo przedstawiony zostanie dokładny opis: zasady działania menu, zmiennych przechowujących informacje o aktualnym stanie pracy sterownika oraz sposobu w jaki mikroprocesor komunikuje się z układami podrzędnymi.

8.1. Wykorzystane biblioteki Arduino

Zanim zostanie przedstawiony opis kodu źródłowego programu, niezbędne będzie przedstawienie funkcji oraz rodzajów zmiennych, które pochodzą z bibliotek dedykowanych danym podzespołom. Znajomość funkcji opisanych w tym podrozdziale pozwoli na zrozumienie instrukcji jakie będą wykonywane zarówno wewnątrz bibliotek, jak i w części głównej programu.

8.1.1. Biblioteki wyświetlacza dotykowego

Jak zostało wcześniej wspomniane platforma Arduino pozwala na skorzystanie z licznego zbioru bibliotek wspomagających komunikację z podzespołami. Niżej znajdzie się opis poszczególnych funkcji i struktur bibliotek: Adafruit_TFTLCD, Adafruit_GFX oraz TouchScreen. Poświęcone są one obsłudze protokołu równoległego opisanego w rozdziale 4.3 obsługującego zastosowany wyświetlacz z ekranem dotykowym przedstawiony w rozdziale 5.1. Dodatkowo przeanalizowanie poniższych instrukcji niezbędne jest do zrozumienia działania napisanych bibliotek oraz głównej części programu.

Klasy i struktury:

Adafruit_TFTLCD – jest to nazwa klasy przechowującej informacje o aktualnym stanie w jakim znajduje się wyświetlacz. Dodatkowo posiada zbiór funkcji do obsługi matrycy wyświetlacza. Jej zainicjowanie jest niezbędne do rozpoczęcia komunikacji ze sterownikiem ILI9341;

TouchScreen – klasa służąca do odczytu współrzędnych punktu na panelu dotykowym. Po jej zainicjowaniu możliwe jest wywoływanie funkcji odczytujących współrzędne osi x i osi y;

TSPoint – struktura przechowująca współrzędne osi x oraz osi y.

Funkcje:

reset() – jest to funkcja resetująca układ sterujący wyświetlaczem dotykowym do ustawień fabrycznych;

uint16_t readID() – odczytuje z układu rejestr w którym znajduje się cyfrowy identyfikator układu podrzędnego;

begin(uint16_t id) – pobiera zmienną 16-bitową id jako dane wejściowe. Jej zastosowanie polega na inicjacji sterownika ILI9341, w taki sposób by możliwe było nawiązanie dalszej komunikacji z wyświetlaczem;

fillRect(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2,uint32_t colour) – funkcja pobiera jako dane wejściowe 4 zmienne 16 bitowe i jedną zmienną 32 bitową. Rysuje prostokąt o początku w punkcie x1,y1 oraz końcu w punkcji x2,y2 wewnątrz którego wszystkie piksele mają kolor określony przez zmienną wejściową colour;

drawRect(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2,uint32_t colour) – działa tak samo jak fillRect z tym wyjątkiem że piksele wewnątrz prostokąta nie ulegają zmianie;

setCursour(uint16_t x, uint16_t y) – ustawia wskaźnik na pozycje x, y i z tej pozycji nastąpi dalsze nadpisywanie pikseli lub tekstu na wyświetlaczu;

print(char* str) – zmienna wejściowa funkcji to ciąg znaków alfanumerycznych, funkcja wypisuje ten ciąg znaków na wyświetlaczu, w miejscu w którym aktualnie znajduje się wskaźnik;

setTextSize(uint8_t size) – funkcja zmieniająca rozmiar czcionki;

fillRect(uint32_t colour) – funkcja ustawiająca kolor tekstu;

getPoint() – funkcja odczytujące wartości ADC informujące o współrzędnych osi x i osi y.

8.1.2. Biblioteka czujnika DHT22

Biblioteka DHTStable obsługująca komunikacje z układem DHT22 podobnie jak w poprzednim podrozdziale również wchodzi w skład gotowych bibliotek Arduino. Jej zadaniem jest odczyt odpowiednich rejestrów czujnika przy użyciu magistrali OneWire. Poniżej zostaną przedstawione wszystkie komendy struktury i funkcje mające na celu odczyt wskazań wilgotności i temperatury.

Klasy i struktury:

DHTStable – jest nazwą klasy w której przechowywane będą odczyty z czujnika DHT22.

Funkcje:

int Read22(uint8_t pin) – funkcja ta posiada jedną 8-bitową zmienną wejściową. Jest nią numer wyprowadzenia z płytki Arduino do którego przyłączona jest linia danych czujnika. Wewnątrz funkcji znajdują się instrukcje odczytujące linię danych w celu pozyskania numeru identyfikacyjnego czujnika. Numer ten jest zwracany przez funkcję w formie zmiennej int czyli liczby rzeczywistej;

float getHumidity() – odczytuje oraz dekoduje wartość wilgotności z czujnika, podając ją jako zmienną wyjściową, będącą zmienną zmiennoprzecinkową;

float getTemperature() – odczytuje wartość temperatury w otoczeniu czujnika, a następnie ją dekoduje i również przedstawia w postaci zmiennej wyjściowej zmiennoprzecinkowej.

8.2. Autorskie biblioteki

W tym podrozdziale przedstawione będą biblioteki, które zostały napisane samodzielnie i posiadają funkcje niezbędne do działania całego oprogramowania. Główną ideą tworzenia bibliotek jest zachowanie przejrzystości kodu. Przykładem tego będą wszystkie funkcje globalne, omówione tutaj i zastosowane w części głównej programu.

8.2.1. i2c.h

Funkcje omawiane w tym podrozdziale skupiają się na komunikacji I^2C. Wszystkie gotowe biblioteki Arduino do komunikacji I^2C, wykorzystują tylko 2 ściśle określone wyprowadzenia z mikrokontrolera, czyli linie analogowe A4 i A5. Zaistniała więc potrzeba napisania biblioteki która umożliwi komunikację przy pomocy 2 dowolnych wyprowadzeń. Jak widać na schemacie ideowym (rys. 6.1) linie komunikacyjne SDA i SCL dołączone są do wyprowadzeń nr. 12 i 13 na płytce Arduino. Dokładny opis magistrali I^2C został przedstawiony w rozdziale 4.1 i posłuży on do analizy kodu znajdującego się w bibliotekach i2c.h i i2c.c. Warto także wspomnieć że obie te biblioteki zostały napisane w sposób uniwersalny dla wszystkich mikroprocesorów AVR. Wszystkie operacje na portach GPIO zostały przeprowadzone poprzez bezpośredni zapis do rejestrów mikrokontrolera.

```
4   *
5   *  Created on: 24 paź 2017
6   *      Author: Emil Rudnicki
7   */
8
9  //#ifndef I2C_H_
11
12  #include "Arduino.h"
13 //#include <avr/io.h>
15
17 #if defined(_AVR_ATtiny44A_H_)
54  #define i2c_ddr DDRB
55  #define i2c_pin PINB
56  #define i2c_port PORTB
57  #define scl_port PB5
58  #define sda_port PB4
59
60
61  #define scl_out i2c_ddr |= (1<<scl_port)    // ustawienie linii scl jako wyjście
62  #define scl_one i2c_port |= (1<<scl_port)   // utawienie logicznej "1" na linii scl
63  #define scl_zero i2c_port &= ~(1<<scl_port) // utawienie logicznego "0" na linii scl
64  #define scl_in i2c_ddr &= ~(1<<scl_port)    // ustawienie linii scl jako wejście
65  #define get_scl (i2c_pin & (1<<scl_port))   // odczyt linii scl
66  #define sda_out i2c_ddr |= (1<<sda_port)    // ustawienie linii sda jako wyjście
67  #define sda_one i2c_port |= (1<<sda_port)   // utawienie logicznej "1" na linii sda
68  #define sda_zero i2c_port &= ~(1<<sda_port) // utawienie logicznego "0" na linii sda
69  #define sda_in i2c_ddr &= ~(1<<sda_port)    // ustawienie linii sda jako wejście
70  #define get_sda !(i2c_pin & (1<<sda_port))  // odczyt linii sda
71
72  #define delay1 _delay_us(2)
73  #define delay2 _delay_us(4)
74
75
76  class i2c_class
77  {
78
79  public:
80      void init(void);
81      void start(void);
82      void stop_trans(void);
83      void requestFrom(uint8_t addr);
84      void sendTo(uint8_t addr);
85      uint8_t send_byte(uint8_t data);
86      void send(uint8_t slaveAddr,uint8_t data,uint8_t stop);
87      uint8_t read(uint8_t slaveAddr,uint8_t stop);
88      uint8_t read_byte(uint8_t stop);
89  };
90
```

Rys. 8.1. Biblioteka i2c.h

Na rysunku 8.1 przedstawiony jest kod źródłowy biblioteki zawierający definicję wszystkich makr i funkcji. W linii nr. 12 załączona zostaje biblioteka Arduino obsługująca rejestry Atmegi. W liniach 54 do 58 znajdują się definicje portów i pinów na których znajdują się wyprowadzenia TWI czyli SCL na porcie PB5 i SDA na PB4. Nazewnictwo rejestrów i rodzaj w jaki są ustawiane opisuje nota katalogowa mikroprocesora Atmega328p [9]. W dalszej części biblioteki linijki 61 do 70 znajdują się makra odpowiedzialne za ustawiania kierunku linii (wejście / wyjście) i ustawianie stanów logicznych „0” i „1” na obu liniach. Linijki 72 i 73 zawierają funkcje opóźnienia kolejno 2 µs i 4 µs. W dalszej części zainicjowana zostaje klasa i2c_class w której znajdują się wszystkie niezbędne funkcje do: rozpoczęcia / zakończenia komunikacji, wysłania jednego bajta, zainicjowania wysłania / odbierania danych. Dokładny opis instrukcji znajdzie się w kolejnym podrozdziale poświęconym bibliotece i2c.c

8.2.2. i2c.c

W bibliotece i2c.c zawarte zostały wszystkie instrukcje wewnątrz funkcji uprzednio zadeklarowanych. Zaczynając od funkcji przedstawionych na rysunku 8.2 na początku załączona zostaje poprzednio opisana biblioteka, czyli i2c.h. Dalej funkcja init (linijki 2 do 7) ustawia porty SDA i SCL jako wyjścia cyfrowe oraz podaje na obie linie wysoki stan logiczny. Dalej funkcje start(void) i stop_trans(void) (linijki 8 do 27) odpowiadają za wygenerowanie sygnału start i stop, według schematu który opisany został w rozdziale 4.1.

```
1  #include "i2c.h"
2  void i2c_class::init(void){
3      scl_out;
4      sda_out;
5      scl_one;
6      sda_one;
7  }
8  void i2c_class::start(void){
9      scl_out;
10     sda_out;
11     delay2;
12     scl_one;
13     sda_one;
14     delay1;
15     sda_zero;
16     delay1;
17     scl_zero;
18 }
19 void i2c_class::stop_trans(void){
20     delay1;
21     sda_out;
22     sda_zero;
23     delay1;
24     scl_one;
25     delay1;
26     sda_one;
27 }
28 uint8_t i2c_class::send_byte(uint8_t data){
29     uint8_t i;
30     sda_out;
31     delay1;
32     for(i=0;i<8;i++){
33             if((data>>(7-i)) & 1)sda_one;
34             else sda_zero;
35             delay1;
36             scl_one;
37             delay2;
38             scl_zero;
39             delay1;
40     }
41     sda_in;
42     delay1;
43     scl_one;
44     if(get_sda & 1)i=1;
45     else i=0;
46     delay2;
47     scl_zero;
48     if(i)return 1;
49     else return 0;
50 }
```

Rys. 8.2. Biblioteka i2c.c linijki 1 do 50

Kolejna funkcja przedstawiona na rysunku 8.2 to send_byte(uint8_t data), odpowiada za przesłanie pojedynczej 8-bitowej ramki danych określonej jako data. Utworzona zostaje wewnątrz funkcji zmienna lokalna, która posłuży jako licznik w pętli. Dalej linia SDA zostaje ustawiona jako wyjście i po czasie 2 µs procesor wchodzi w pętlę składającą się z 8 kroków. W

każdym kroku instrukcja warunkowa z linijek 33 i 34 wystawia odpowiedni stan logiczny na SDA w oparciu o dane wejściowe czyli zmienną data. Bity jak zostało to wspomniane w rozdziale poświęconym I^2C wysyłane są od najstarszego do najmłodszego o czym świadczy nawias (7-i) wewnątrz instrukcji warunkowej. W dalszej części przez Arduino generowany jest sygnał zegarowy (linijki 35 do 39) tworząc przebieg prostokątny o wypełnieniu 50%. Po wysłaniu 8 bitów linia SDA zostaje zmieniona na wejście cyfrowe po czym generowane jest kolejne zbocze narastające na SCL i odczytywany zostaje bit potwierdzenia. Bit potwierdzenia generuje układ podrzędny i po odczytaniu funkcja zwraca jego wartość jako zmienną wyjściową.

```
51 void i2c_class::requestFrom(uint8_t addr){
52     send_byte((addr<<1)|1);
53 }
54 void i2c_class::sendTo(uint8_t addr){
55     send_byte((addr<<1)|0);
56 }
57 void i2c_class::send(uint8_t slaveAddr,uint8_t data,uint8_t stop){
58     //  cli();
59     //uint8_t i;
60     start();
61     send_byte((slaveAddr<<1)|0);
62     send_byte(data);
63     //if(!i)led2_off;
64     if(stop)stop_trans();
65 //  sei();
66 }
67 uint8_t i2c_class::read_byte(uint8_t stop){
68     uint8_t i,c,val=0;
69     sda_in;
70     sda_one;
71     for(i=0;i<8;i++){
72         delay2;
73         scl_one;
74         delay1;
75         c = get_sda;
76         if(c==1){
77             //led2_off;
78             val |= (1<<(7-i));
79         }
80         delay1;
81         scl_zero;
82     }
83     delay1;
84     sda_out;
85     if(stop)sda_zero;
86     else sda_one;
87     delay1;
88     scl_one;
89     delay2;
90     scl_zero;
91     delay1;
92     if(stop)sda_zero;
93     delay1;
94     scl_one;
95     delay2;
96     scl_zero;
97     if(stop)stop_trans();
98     val = ~val;
99     return val;
100 }
```

Rys. 8.3. Biblioteka i2c.c linijki 51 do 100

Na rysunku 8.3 przedstawione są dalsze linijki kodu, gdzie kolejne 2 funkcje czyli requestFrom(uint8_t addr) i sendTo(uint8_t addr) służą kolejno do wysłania żądania odebrania

danych i wysłania danych do układu podrzędnego. Obie te operacje polegają na wysłaniu 7-bitowego adresu odbiornika oraz ustawienia ostatniego bitu jako „1” jeżeli ma nastąpić odbieranie danych lub „0” jeżeli będzie miało miejsce wysyłanie danych. Następną funkcją na rysunku 8.3 jest uint8_t read_byte(uint8_t stop), służy ona do odebrania pełnej ramki danych którą wysyła układ podrzędny. Na początku inicjowane są zmienne lokalne funkcji czyli i (pełni rolę licznika), c (przechowuje wartość odczytanego bitu) oraz val (do którego zapisana zostaje wartość bajta). Dalej linia danych ustawiana jest jako wejście i generowana jest na niej logiczna „1”. Po tym program wchodzi w 8 krokową pętle podobnie jak w przypadku send_byte(uint8_t data). Różnica polega na tym że w każdym kroku po wygenerowaniu zbocza narastającego na SCL odczytywany zostaje stan logiczny na SDA a jego pojedyncza wartość przekazywana jest do zmiennej c. W trakcie każdego kroku aktualizowana jest zmienna val o każdy kolejny odczytany bit. Po wyjściu z pętli zależnie od zmiennej we. stop na linii SDA generowany jest sygnał „0” lub „1”. Uzależnione jest to od tego czy w dalszej części program ma zamiar odczytywać kolejne rejestry lub nie. Po wygenerowaniu odpowiedniego bitu potwierdzenia zwracana zostaje przez funkcje zmienna odczytanego bajta.

```
97      if(stop)stop_trans();
98      val = ~val;
99      return val;
100 }
101 uint8_t i2c_class::read(uint8_t slaveAddr,uint8_t stop){
102     start();
103     uint8_t i,c,val=0;
104     i = send_byte((slaveAddr<<1)|1);
105     sda_in;
106     sda_one;
107     for(i=0;i<8;i++){
108         delay2;
109         scl_one;
110         delay1;
111         c = get_sda;
112         if(c==1){
113             val |= (1<<(7-i));
114         }
115         delay1;
116         scl_zero;
117     }
118     delay1;
119     sda_out;
120     sda_zero;
121     delay1;
122     scl_one;
123     delay2;
124     scl_zero;
125     if(stop)stop_trans();
126     val = ~val;
127     return val;
128 }
129
130
```

Rys. 8.4. Biblioteka i2c.c linijki 98 do 129

Ostatnia funkcja która została zadeklarowana to uint8_t read(uint8_t slaveAddr, uint8_t stop) przedstawiona jest na rysunku 8.4 (linijki 101 do 128). Stanowi ona rozbudowanie uint8_t read_byte(uint8_t stop), co widać po tym że w tej funkcji dodana zostaje jedna zmienna

wejściowa, która ma zawierać adres układu odbiorczego. Różnica wewnątrz funkcji polega na rozpoczęciu komunikacji przez nadanie sygnału start. Kolejna instrukcja wewnątrz tej funkcji dotyczy wysłania żądania odebrania danych wraz z adresem układu podrzędnego (linijka 104). Pozostała części funkcji nie zmienia się względem tej w której odbierany był pojedynczy bajt.

8.2.3. ds1307.h

Posiadając uniwersalną bibliotekę (na 2 dowolne porty GPIO) do obsługi magistrali I^2C, można przejść do omówienia kolejnej biblioteki która z niej bezpośrednio korzysta. W tym oraz w następnym podrozdziale przedstawiona zostanie biblioteka, która odczytuje wskazania zegara czasu rzeczywistego. Definicje rejestrów oraz ich wartości, które się tutaj znajdują zostały szczegółowo opisane w nocie katalogowej DS1307 [8]. Warto także wspomnieć że wszystkie układy RTC z przedrostkiem DS w nazwie posiadają uniwersalną numerację rejestrów, więc tą bibliotekę również można określić jako uniwersalną.

```
1  #pragma once
2
3  #include "Arduino.h"
4  #include "../i2c/i2c.h"
5
6  #define ds_addr 0b1101000
7  #ifdef SQW
8  #define ds_setSQW PORTC |= (1<<PC7);
9  #define ds_resetSQW PORTC &= ~(1<<PC7);
10 #endif
11
12 #define secReg 0
13 #define minReg 1
14 #define hourReg 2
15 #define dTReg 3
16 #define dayReg 4
17 #define mounthReg 5
18 #define yearReg 6
19
20 struct ds_val{
21     uint8_t sec;
22     uint8_t mins;
23     uint8_t hour;
24     uint8_t dayofWeek;
25     uint8_t day;
26     uint8_t mounth;
27     uint8_t year;
28 };
29
30 class ds_class
31 {
32 public:
33     i2c_class i2c;
34     void init(void);
35 //  void setTime(uint8_t sec,uint8_t  min,uint8_t  ho
36     void setTime(ds_val in_val);
37     void setSingleReg(uint8_t regNr,uint8_t data);
38     uint8_t readSingleReg(uint8_t  regNr);
39     uint16_t readTime(void);
40     uint16_t readDate(void);
41     ds_val readAll(void);
42     void enableSQW(void);
43     void checkSQW(void);
44     void SQWblink(void);
45 };
46
```

Rys. 8.5. Biblioteka ds1307.h

Przechodząc do analizy kodu zawartego w bibliotece ds1307.h przedstawionej na rysunku 8.5, widać że zaczyna się ona od #pragma once. Jest to dyrektywa niezbędna do działania całego programu gdy jedna biblioteka jest załączana w co najmniej 2 innych bibliotekach lub pod-programach. Taka sytuacja ma miejsce w przypadku tego programu. Aby kompilator nie kompilował wielokrotnie tej biblioteki, linijka z dyrektywą #pragma once zapobiega temu procesowi. W kolejnych linijkach (3 i 4) załączone zostają biblioteki Arduino oraz i2c. Niżej widać definicje adresu układu podrzędnego w postaci binarnej znanego z noty katalogowej [8], a pod nim znajduje się szare pole wykluczone z kompilacji. SQW jest to jedno z wyjść DS1307 na którym w razie potrzeby generowany jest sygnał prostokątny o wypełnieniu 50% oraz częstotliwości 1 Hz. Jako że nie znajduje zastosowania w przypadku sterownika ogrodowego nie jest on potrzebny. Wszystkie linijki kodu odnoszące się do niego zostały wyłączone z kompilacje. W linijkach 12 do 18 znajdują się definicje makr posiadających liczbową wartość rejestrów układu podrzędnego [8]. Dalej w bibliotece zainicjowana zostaje struktura ds_val (linijki 20 do 28). Przeznaczona jest ona do przechowywania wartości wszystkich możliwych wskazań czasu z DS1307. Następnie zainicjowana zostaje klasa ds_class wewnątrz której znajdują się deklaracje funkcji oraz klasy lokalnej, czyli i2c w linijce 33. Funkcje wewnątrz klasy ds_class szerzej zostaną opisane w kolejnym podrozdziale.

8.2.4. ds1307.c

Aby przejść do opisu funkcji wraz z instrukcjami wewnątrz nich, należy rozpocząć od analizy rysunku 8.6. Na początku biblioteki znajduje się standardowe załączenie biblioteki ds1307.h. Pierwszą widoczną funkcją jest init. Wywoływana jest ona jednokrotnie w programie do zainicjowania pracy układu RTC. Proces inicjacji polega na odpowiednim ustawieniu rejestrów odpowiedzialnych za pracę GPIO co zostało opisane w rozdziale 8.2.2. Widoczne jest więc odwołanie do biblioteki i2c w linijce 4. Kolejną funkcją jest setTime. Jej ogólne zastosowanie to pobranie struktury danych ds_val z czasem i nadpisanie odpowiednich rejestrów na wartości czasu jakie zostały podane. W linijkach 31 do 37 dane liczbowe ze struktury podlegają transformacji na postać jaką akceptuje układ podrzędny. Sam opis jaki jest format wartości czasu dla poszczególnych rejestrów znajduje się w nocie katalogowej [8]. Dla większości rejestrów format danych polega na przesłaniu w bajcie cyfry dziesiątek umieszczonej w starszych 4 bitach i cyfry jedności w młodszych 4 bitach. Przykładowo jeżeli mikroprocesor ma zapisać do RTC godzinę 12 to jej zapis binarny będzie miał postać 0b00010010.

```
1  #include "ds1307.h"
2
3  void ds_class::init(void){
4      i2c.init();
5  //  ds_enableSQW();
6  }
7  //void ds_class::setTime(uint8_t sec,uint8_t  min,uint8_t  hour,uint8_t dayOfWeek,uint8_t
29
30 void ds_class::setTime(ds_val in_val){ //rok to liczba 2 cyfrowa np. 17 oznacza rok 2017
31     in_val.sec = ((in_val.sec/10)<<4)|(in_val.sec%10);
32     in_val.mins = ((in_val.mins/10)<<4)|(in_val.mins%10);
33     in_val.hour = ((in_val.hour/10)<<4)|(in_val.hour%10);
34     in_val.hour &= ~(0b11<<6);
35     in_val.day = ((in_val.day/10)<<4)|(in_val.day%10);
36     in_val.mounth = ((in_val.mounth/10)<<4)|(in_val.mounth%10);
37     in_val.year = ((in_val.year/10)<<4)|(in_val.year%10);
38     i2c.start();
39     i2c.sendTo(ds_addr);
40     i2c.send_byte(0);   // adres rozpoczęcia zapisu danych
41     i2c.send_byte(in_val.sec);
42     i2c.send_byte(in_val.mins);
43     i2c.send_byte(in_val.hour);
44     i2c.send_byte(in_val.dayofWeek);
45     i2c.send_byte(in_val.day);
46     i2c.send_byte(in_val.mounth);
47     i2c.send_byte(in_val.year);
48     i2c.stop_trans();
49
50 }
51 void ds_class::setSingleReg(uint8_t regNr,uint8_t data){
52     i2c.start();
53     i2c.sendTo(ds_addr);
54     i2c.send_byte(regNr);
55     i2c.send_byte(data);
56     i2c.stop_trans();
57 }
58 uint8_t ds_class::readSingleReg(uint8_t  regNr){
59     i2c.start();
60     i2c.sendTo(ds_addr);
61     i2c.send_byte(regNr);
62     i2c.start();
63     i2c.requestFrom(ds_addr);
64     uint8_t ret=i2c.read_byte(0);
65
66     return ret;
67 }
68 uint16_t ds_class::readTime(void){  //starsze 8 bitów godzina, młodsze 8 bitów minuty
69     uint8_t min = readSingleReg(minReg);
70     min = (((min>>4)&0x0F)*10)+(min&0x0F);
71     uint8_t hour = readSingleReg(hourReg);
```

Rys. 8.6. Biblioteka ds1307.c linijki od 1 do 71

Po odpowiednim nadpisaniu danych w strukturze in_val rozpoczęta zostaje komunikacja z DS1307. Na magistralę I^2C podany zostaje sygnał START. Pierwszy przesłany bajt zawiera adres układu podrzędnego wraz z bitem informującym że w dalszej części nastąpi zapis danych. Kolejny bajt określa numer rejestru który ma zostać nadpisany i jest nim 0 co wynika z noty katalogowej [8]. Następnie przesłane zostają kolejno bajty w których znajduje się wartość: sekundy, minuty, godziny, dnia tygodnia, dnia miesiąca, miesiąca oraz roku. Po przesłaniu danych do wszystkich pożądanych rejestrów generowany jest sygnał STOP.

Następna funkcja widoczna na rysunku 8.6 to setSingleReg(uint8_t regNr, uint8_t data). Służy ona do zapisania pojedynczego rejestru (o numerze regNR) układu DS1307 pojedynczym bajtem (data). Instrukcje wykonujące to zadanie znajdują się w linijkach od 52 do 26 a ich zapis pokrywa się z teorią przedstawioną w rozdziale 4.1. Ostania funkcja widoczna na rysunku 8.6 to setSingleReg(uint8_t regNr). Jej zadaniem jest odczyt pojedynczego rejestru (regNr) z układu

podrzędnego DS1307. Tutaj również linijki wewnątrz (59 do 66) napisane zostały w oparciu o schemat odczytu z rozdziału 4.1. Funkcja co oczywiste zwraca wartość 8-bitową.

```
63      i2c.requestFrom(ds_addr);
64      uint8_t ret=i2c.read_byte(0);
65
66      return ret;
67  }
68  uint16_t ds_class::readTime(void){  //starsze 8 bitów godzina, młodsze 8 bitów minuty
69      uint8_t min = readSingleReg(minReg);
70      min = (((min>>4)&0x0F)*10)+(min&0x0F);
71      uint8_t hour = readSingleReg(hourReg);
72      hour = (((hour>>4)&0x0F)*10)+(hour&0x0F);
73      uint16_t ret = (hour<<8)|min;
74      return ret;
75  }
76  uint16_t ds_class::readDate(void){  //starsze 8 bitów miesiąc, młodsze 8 bitów dzień
77      uint8_t day = readSingleReg(dayReg);
78      day = (((day>>4)&0x0F)*10)+(day&0x0F);
79      uint8_t mon = readSingleReg(mounthReg);
80      mon = (((mon>>4)&0x0F)*10)+(mon&0x0F);
81      uint16_t ret = (mon<<8)|day;
82      return ret;
83  }
84  ds_val ds_class::readAll(void){
85      ds_val ret;
86      uint16_t Time,Date;
87      uint8_t second,year,doW;
88      Time = readTime();
89      Date = readDate();
90      second = readSingleReg(secReg);
91      second = (((second>>4)&0x0F)*10)+(second&0x0F);
92      doW = readSingleReg(dTReg);
93      doW = (((doW>>4)&0x0F)*10)+(doW&0x0F);
94      year = readSingleReg(yearReg);
95      year = (((year>>4)&0x0F)*10)+(year&0x0F);
96      ret.sec = second;
97      ret.mins = Time & 0xFF;
98      ret.hour = (Time >> 8) & 0xFF;
99      ret.dayofWeek = doW;
100     ret.day = Date & 0xFF;
101     ret.mounth = (Date >> 8) & 0xFF;
102     ret.year = year;
103     return ret;
104 }
105
106
107 #ifdef SQW
108 void ds_enableSQW(void){
118 void ds_checkSQW(void){
119     if((PINB>>PB2)&1)PORTC |= (1<<PC7); //sprawdza pors sqw z ds1307 i ustawia diode na w
120     else PORTC &= ~(1<<PC7);
121 }
```

Rys. 8.7. Biblioteka ds1307.c linijki od 63 do 121

Rysunek 8.7 zawiera instrukcje wewnątrz 3 ostatnich funkcji jakie zostały zadeklarowane. Funkcje readTime i readDate nie będą używane w żadnej innej bibliotece ani programie ponieważ ich zastosowanie w przypadku pracy sterownika ogrodowego nie jest wymagane. Służą one do odczytu 2 rejestrów, readTime odczytuje i transformuje wartości godziny i minuty, a readDate wartości dnia i miesiąca. Po odczycie i transformacji następuje zwrócenie obu wartości w zmiennej 16-bitowej. Starsze 8 bitów zawiera miesiąc / godzinę, a młodsze 8-bitów zawiera dzień miesiąca / minutę. Zastosowania struktury przechowującej wszystkie wskazania czasu jest najlepszym rozwiązaniem. Ponieważ jak zostało to wcześniej wspomniane na ekranie wyświetlacza mają się one znajdować na ekranie głównym. Funkcją odpowiedzialną za to jest

readAll (linijki 84 do 104) a jej danymi wyjściowymi jest struktura ds_val. Na początku tej funkcji inicjowana jest struktura lokalna czyli ret, a także pozostałe zmienne lokalne (8-bitowe i 16-bitowe) przechowujące wskazania czasu. W dalszej części następuje odczyt wszystkich rejestrów a następnie jego transformacja i zapis do struktury ret, która pod koniec funkcji zwraca aktualne wskazania czasu z DS1307.

8.2.5. menu.h

Ostatnią biblioteką autorską użytą w projekcie jest menu. Jak nazwa na to wskazuje do jej głównych zadań należy zarządzanie wyświetlanym obrazem, w zależności od interakcji jaką podejmie użytkownik. Dodatkowo w bibliotece znajdują się odpowiednie struktury danych w których przechowywane są informacje na temat w jakim stanie wyświetlacz się aktualnie znajduje.

Początek biblioteki menu.h widoczny jest na rysunku 8.8. W linijkach 1 do 6 załączone zostały wszystkie biblioteki które opisane były w tym rozdziale. Związane jest to ze złożonością wykonywanych operacji, które szerzej zostaną opisane w kolejnym podrozdziale. W dalszej części tj. linijki 8 do 15 zawarte są definicje domyślnych kolorów z których korzystać będzie wyświetlacz. Następnie makra w linijkach 17 do 23 odnoszą się do stanu pracy pompki a także odnoszą się do instrukcji jej załączania i wyłączania. Kolejne widoczne makra informować będą o aktualnym stanie wyświetlanego ekranu zależnie od stanu pracy mikrokontrolera. Ostatnia definicja czyli pwm(p) jest czytelnym skrótem do instrukcji ustalającej wypełnienie sygnału PWM używanego do sterowania jasnością oświetlenia zewnętrznego.

```
1  #include "Arduino.h"
2  #include <Adafruit_GFX.h>    // Core graphics library
3  #include <Adafruit_TFTLCD.h> // Hardware-specific library
4  #include <TouchScreen.h>
5  #include "../i2c/i2c.h"
6  #include "../ds1307/ds1307.h"
7
8  #define BLACK   0x0000
9  #define BLUE    0x001F
10 #define RED     0xF800
11 #define GREEN   0x07E0
12 #define CYAN    0x07FF
13 #define MAGENTA 0xF81F
14 #define YELLOW  0xFFE0
15 #define WHITE   0xFFFF
16
17 #define SET_PUMP_ON digitalWrite(10,1)
18 #define SET_PUMP_OFF digitalWrite(10,0)
19 #define PUMP_TURN_OFF 0
20 #define PUMP_WAITING_ST 1
21 #define PUMP_TURN_ON 2
22 #define START_TIME_SETTINGS 3
23 #define END_TIME_SETTINGS 4
24
25 #define MAIN_SCREEN 0
26 #define MAIN_MENU_SCREEN 1
27 #define SET_TIME_SCREEN 2
28 #define SET_DATE_SCREEN 3
29 #define PUMP_START_TIME_SCREEN 4
30 #define PUMP_END_TIME_SCREEN 5
31 #define LIGHT_SCREEN 6
32 #define MISSED_TOUCH 0xFF
33
34 #define pwm(p) analogWrite(11,p)
35
36 struct screen_var{
37     float Hum;
38     float Temp;
39     ds_val dateVar;
40 };
41
42 struct pump_settings{
43     ds_val start_time;
44     ds_val end_time;
45     uint8_t in_use;
46 };
47
48 struct menu_state{
49     uint8_t screen_state;   // poziom menu w którym jest obecnie wyświetlacz
50     ds_val time;            // zapisywane dane czasu w strukturze
```

Rys. 8.8. Biblioteka menu.h linijki od 1 do 50

Kolejna część biblioteki menu.h poświęcona jest zainicjowaniu struktur. W pierwszej kolejności widać strukturę screen_var. Wewnątrz niej znajdują się wskazania: wilgotności (Hum), temperatury (Temp) oraz czasu (dateVar). Następna struktura czyli pump_settings posiada informację o godzinie rozpoczęcia podlewania (start_time), zakończenia podlewania (end_time) i zmienną dateVar. Zmienna ta odnosić się będzie do aktualnego trybu pracy pompki, a zapisywane do niej będą wartości liczbowe makr znajdujących się w linijkach 19 do 23.

Rysunek 8.9 zawiera ostatnią i najważniejszą strukturę czyli menu_state znajdującą się w linijkach 48 do 56. Sam opis zmiennych wewnątrz struktury opisany jest komentarzami widocznymi na rysunku. Zasada działania całego programu opierać się będzie o wymianę danych między biblioteką menu a częścią główną programu. Dlatego właśnie większość funkcji które się tutaj znajdują, będą nadpisywać strukturę menu_state. Nadpisywanie nastąpi zarówno

w bibliotece jak i poza nią. Dane wewnątrz tej struktury niezbędne będą do użycia w odpowiednich instrukcjach warunkowych, odpowiedzialnych za interakcje użytkownika z wyświetlaczem, a także do sterowania pracą pompki oraz oświetlenia.

```
36  struct screen_var{
37      float Hum;
38      float Temp;
39      ds_val dateVar;
40  };
41
42  struct pump_settings{
43      ds_val start_time;
44      ds_val end_time;
45      uint8_t in_use;
46  };
47
48  struct menu_state{
49      uint8_t screen_state;   // poziom menu w którym jest obecnie wyświetlacz
50      ds_val time;            // zapisywane dane czasu w strukturze
51      pump_settings pump_val; // struktura z godziną rozpoczęcia zakończenia podlewania
52      uint8_t last_var;       // wartośc bierzącej zmiennej (o ile jest w użytku)
53      uint16_t px;            // współrzędne punktu x wyświetlacza dotykowego
54      uint16_t py;            // współrzędne punktu y wyświetlacza dotykowego
55      uint8_t light_px = 21;  // położenie wskaźnika jasnosci
56  };
57
58  class screen
59  {
60  public:
61      Adafruit_TFTLCD initScreen(Adafruit_TFTLCD tft_in);
62      void print_main_screen();
63      void print_main_menu_screen();
64      void updateMainScreen(screen_var curr_var);
65      menu_state check_btn_routine(menu_state screen_state);
66      void print_light_screen(menu_state scr);
67      menu_state print_screen(menu_state scr);
68  private:
69      Adafruit_TFTLCD tft_lib;
70      screen_var prev_var;
71      menu_state check_light_screen_routine(menu_state scr);
72      menu_state check_screen_routine(menu_state scr);
73      menu_state set_date_check_routine(menu_state set_date_st);
74      menu_state set_time_check_routine(menu_state set_time_st);
75      menu_state set_pump_check_routine(menu_state set_pump_st);
76      void setting_sign(int x0, int y0,uint16_t colour);
77      void up_sign(int x0, int y0,uint16_t bg_colour ,uint16_t colour);
78      void down_sign(int x0, int y0,uint16_t bg_colour ,uint16_t colour);
79      void exit_sign(int x0, int y0);
80      void ok_sign(int x0, int y0);
81  };
82
```

Rys. 8.9. Biblioteka menu.h linijki od 36 do 82

Przechodząc do dalszej części biblioteki widoczna jest definicja klasy screen która znajdzie zastosowanie w programie głównym czyli main.c. Widoczny jest tutaj podział na funkcje globalne z których korzystać będzie main.c oraz na funkcje i zmienne lokalne które używane są wyłącznie wewnątrz samej biblioteki. Wśród deklaracji funkcji lokalnych znalazły się również 2 struktury a są nimi tft_lib i prev_var. Struktura tft_lib odpowiadać będzie za nadpisywanie stanu wyświetlacza. Jej zastosowanie uprości ilość zmiennych wejściowych wszystkich funkcji biblioteki, ponieważ każda funkcja zawierać będzie instrukcje związane z nadpisywaniem wyświetlacza. Druga struktura czyli prev_var przechowywać będzie wartości wszystkich zmiennych z poprzedniego cyklu procesora. Takie rozwiązanie pozwala nadpisywać na wyświetlaczu tylko te wartości które uległy zmianie co znacząco skraca czas trwania

pojedynczego cyklu procesora. Szczegółowy opis wszystkich funkcji znajdzie się w kolejnym podrozdziale gdzie omówiona będzie biblioteka main.c.

8.2.6. menu.c

Ostatnią omawianą biblioteką jest main.c, jej pierwsza część listingu przedstawiona jest na rysunku 8.10. W pierwszej linijce następuje standardowe załączenie biblioteki deklaracji z poprzedniego podrozdziału. Pierwszą funkcją natomiast jest initScreen, służy ona do zainicjowania procesu wyświetlania danych na wyświetlaczu TFT LCD, oraz nadpisania struktury Adafruit_TFTLCD. Wewnątrz funkcji znajdują się instrukcje resetujące sterownik wyświetlacza, odczytujące jego sygnaturę (numer id) oraz zapisujące rejestry, w sposób umożliwiający dalszą komunikację. Następnie wysyłana jest instrukcja wypełniająca cały ekran kolorem czarnym. Na końcu funkcji, struktura pobrana po zainicjowaniu pracy wyświetlacza, zostaje przekazana do struktury lokalnej tft_lib. Natomiast struktura prev_var zostaje wyzerowana co umożliwi nadpisanie wszystkich zmiennych na wyświetlaczu przy pierwszym zapisaniu ekranu głównego.

```
#include "menu_lib.h"

Adafruit_TFTLCD screen::initScreen(Adafruit_TFTLCD tft_in){
    tft_in.reset();

    uint16_t identifier = tft_in.readID();

    if(identifier == 0x9341) {
        Serial.println(F("Found ILI9341 LCD driver"));
    }

    tft_in.begin(identifier);
    tft_in.fillScreen(BLACK);
    tft_lib = tft_in;
    prev_var = {0};
    return tft_in;
}
void screen::exit_sign(int x0, int y0){
    tft_lib.setTextSize(3);
    tft_lib.fillRoundRect(x0,y0,50,50,10,RED);
    tft_lib.setCursor(x0+10,y0+10);
    tft_lib.setTextColor(WHITE);
    tft_lib.print("<");
}
void screen::ok_sign(int x0, int y0){
    tft_lib.setTextSize(3);
    tft_lib.fillRoundRect(x0,y0,50,50,10,GREEN);
    tft_lib.setCursor(x0+10,y0+15);
    tft_lib.setTextColor(WHITE);
    tft_lib.print("ok");
}
void screen::up_sign(int x0, int y0,uint16_t bg_colour ,uint16_t colour){
    uint8_t w = 50;
    tft_lib.fillRoundRect(x0,y0,50,50,10,bg_colour);
    uint8_t i;
    for(i=0;i<3;i++){
        tft_lib.drawLine(x0+10 + i, y0+w-10 + i, x0+23 + i, y0+10 + i,colour);
        tft_lib.drawLine(x0+10 + 1 + i, y0+w-10 + i, x0+23 + i, y0+10 + 1 + i,colour);

        tft_lib.drawLine(x0+w-10 - i ,y0+w-10 + i ,x0+25 - i ,y0+10 + i, colour);
        tft_lib.drawLine(x0+w-10 - 1 - i ,y0+w-10 + i ,x0+25 - 1 - i ,y0+10 + i, colour);
    }
}
void screen::down_sign(int x0, int y0,uint16_t bg_colour ,uint16_t colour){
    uint8_t w = 50;
    tft_lib.fillRoundRect(x0,y0,50,50,10,bg_colour);
    uint8_t i;
    for(i=0;i<3;i++){
        tft_lib.drawLine(x0+10 +i, y0+10 -i, x0+23+i, y0+w-10-i,colour);
        tft_lib.drawLine(x0+10 +i, y0+10+1 -i, x0+23+i, y0+w-10+1-i,colour);
```

Rys. 8.10. Biblioteka menu.c linijki od 1 do 50

Funkcje exit_sign i ok_sign posiadają po 2 zmienne wejściowe a ich zadaniem jest nadpisanie wyświetlacza odpowiednio znakiem wyjścia lub zatwierdzenia. Miejsce w którym ma nastąpić nadpisanie ustalone jest przez zmienne wejściowe x0 i y0. Sam proces nadpisywania polega na użyciu komend rysujących odpowiednie kształty lub symbole. Opisane są one w rozdziale 8.1.1 będąc komendami gotowej biblioteki odpowiedzialnej za komunikacje ze sterownikiem wyświetlacza. Kolejną funkcją na rysunku 8.10 jest up_sign i służy do narysowania przycisku zwiększania, czyli strzałki w górę. Zawiera ona 4 zmienne wejściowe: x0 i y0 (określające położenie rysowanego znaku), bg_colour (kolor tła przycisku) oraz colour (kolor rysowanej strzałki). Również w przypadku tej funkcji instrukcje opierają się na narysowaniu kształtu przy pomocy odpowiednich komend. Ustalenie koloru tła i strzałki potrzebne będzie gdy mikrokontroler wykryje wciśnięcie przycisku. Wtedy przycisk zmieni swój kolor.

```
44 void screen::down_sign(int x0, int y0,uint16_t bg_colour ,uint16_t colour){
45     uint8_t w = 50;
46     tft_lib.fillRoundRect(x0,y0,50,50,10,bg_colour);
47     uint8_t i;
48     for(i=0;i<3;i++){
49         tft_lib.drawLine(x0+10 +i, y0+10 -i, x0+23+i, y0+w-10-i,colour);
50         tft_lib.drawLine(x0+10 +i, y0+10+1 -i, x0+23+i, y0+w-10+1-i,colour);
51         //
52         tft_lib.drawLine(x0+w-10 - i ,y0+10 - i ,x0+25-i ,y0+w-10-i , colour);
53         tft_lib.drawLine(x0+w-10-1 - i ,y0+10 - i ,x0+25-1-i ,y0+w-10-i , colour);
54     }
55 }
56
57 void screen::setting_sign(int x0, int y0,uint16_t colour){
58     uint8_t w = 32;
59         tft_lib.fillCircle(x0+16, y0+16, 10, colour);
60         tft_lib.fillRect(x0+13, y0, 6, w, colour);
61         tft_lib.fillRect(x0, y0+13, w, 6, colour);
62         uint8_t i;
63         for (i = 0; i < 4; i++) {
64         tft_lib.drawLine(x0 + 2 + i, (y0+w) - 2 - 4 + i, (x0+w) - 2 - 4 + i, y0 + 2 + i, colour);
65         tft_lib.drawLine(x0 + 2 + 1 + i, (y0+w) - 2 - 4 + i, (x0+w) - 2 - 4 + i, y0 + 2 + 1 + i, colour);
66         tft_lib.drawLine(x0 + 2 + i, y0 + 2 + 4 - i, (x0+w) - 2 - 4 + i, (y0+w) - 2 - i, colour);
67         tft_lib.drawLine(x0 + 2 + i, y0 + 2 + 4 - 1 - i, (x0+w) - 2 - 4 - 1 + i, (y0+w) - 2 - i, colour);
68       }
69 }
70 void screen::print_main_screen(){
71
72     tft_lib.fillScreen(BLACK);
73     tft_lib.fillRect(0,0,240,40,CYAN);
74     tft_lib.drawRect(0,0,240,40,RED);
75     tft_lib.drawRect(1,1,238,38,RED);
76     setting_sign(204,4,BLACK);
77     tft_lib.setCursor(10, 5);
78     tft_lib.setTextColor(BLACK);
79     tft_lib.setTextSize(3);
80     tft_lib.print("Ustawienia");
81
82     tft_lib.setCursor(0, 60);
83     tft_lib.setTextColor(WHITE);
84     tft_lib.print("Wilg.");
85
86     tft_lib.setCursor(120, 60);
87     tft_lib.println("Temp.");
88     tft_lib.drawCircle(195, 100, 3, WHITE); // znak stopnia Celciusza
89     tft_lib.setCursor(200, 100);
90     tft_lib.println("C");
91 }
92
93 void screen::print_main_menu_screen(){
```

Rys. 8.11. Biblioteka menu.c linijki od 44 do 93

Na rysunku 8.11 widoczne są kolejne funkcje, czyli setting_sign i print_main_screen. Pierwsza funkcja nadpisuje wyświetlacz w punkcie o współrzędnych x0, y0 o znak ustawień, czyli powszechnie znany kształt zębatki. Do tego zastosowane zostały instrukcje rysujące okrąg i 4 prostokąty (składające się z linii). Następna funkcja czyli print_main_screen służy do wyświetlenia ekranu głównego widocznego na rysunku 7.1. W pierwszej kolejności wypełnia całą matrycę kolorem czarnym, a następnie wyświetla wszystkie stałe elementy na ekranie głównym. Są nimi przycisk ustawień, tekst Wilg., tekst Temp. oraz znak stopnia Celsjusza.

Kolejną funkcją biblioteki menu.c jest print_main_menu_screen (rys. 8.12) i tak jak nazwa wskazuje odpowiada ona za wyświetlenie menu głównego przedstawionego na rysunku 7.2. Wewnątrz funkcji znajduje się tablica ciągu znaków która przechowuje tekst znajdujący się wewnątrz przycisków. Przed procesem wyświetlenia menu głównego ekran jest wypełniany kolorem tła czyli czarnym. Następnie funkcja przechodzi do pętli for która zależnie od licznika

i, rysuje kolejne przyciski. Przyciski składają się z czerwonej obwódki, cyjanowego tła przycisku i czarnego tekstu z zainicjowanej wcześniej tablicy

```
93  void screen::print_main_menu_screen(){
94      uint8_t btn_w = 240, btn_h = 40, i;
95      char* btn_txt[4] ={
96              "godzina",
97              "data",
98              "pompka",
99              "swiatlo"
100     };
101     tft_lib.fillScreen(BLACK);
102     for(i=0;i<4;i++){
103         tft_lib.fillRect(0,(i * btn_h)+(10 * i) + 10,btn_w,btn_h,CYAN);
104         tft_lib.drawRect(0,(i * btn_h)+(10 * i) + 10,btn_w,btn_h,RED);
105         tft_lib.drawRect(1,(i * btn_h)+(10 * i) + 11,btn_w - 2,btn_h - 2,RED);
106         tft_lib.setCursor(50 ,(i * btn_h)+(10 * i) + 20);
107         tft_lib.setTextColor(BLACK);
108         tft_lib.setTextSize(3);
109         tft_lib.print(btn_txt[i]);
110     }
111     exit_sign(180,260);
112 }
113
114 void screen::updateMainScreen(screen_var curr_var){
115     float recHum = curr_var.Hum;
116     float recTemp = curr_var.Temp;
117     ds_val DateNtime = curr_var.dateVar;
118     ds_val last = prev_var.dateVar;
119     float lastHum = prev_var.Hum;
120     float lastTemp = prev_var.Temp;
121     tft_lib.setTextSize(3);
122     if ((int)recHum != (int)lastHum) {   // wilgotność
123         tft_lib.fillRect(10, 100, 34, 30, BLACK);
124         tft_lib.setCursor(10, 100);
125         tft_lib.print((int)recHum);
126         tft_lib.print("%");
127       }
128       if (recTemp != lastTemp) {  // temperatura
129         tft_lib.fillRect(120, 100, 70, 22, BLACK);
130         tft_lib.setCursor(120, 100);
131         tft_lib.print((int)recTemp);
132         tft_lib.print(".");
133         recTemp *= 10;
134         int a = (int)recTemp;
135         a %= 10;
136         tft_lib.print(a);
137       }
138       tft_lib.setTextSize(5);
139       if (last.hour != DateNtime.hour) {       // godzina
140         tft_lib.fillRect(0, 170, 90, 38, BLACK);
141         tft_lib.setCursor(0, 170);
142         if (DateNtime.hour < 10)tft lib.print("0");
```

Rys. 8.12. Biblioteka menu.c linijki od 93 do 142

Na rysunku 8.12 widoczny jest również początek funkcji updateMainScreen która pobiera strukturę curr_val z aktualnymi odczytami czasu, wilgotności i temperatury. Zadaniem tej funkcji jest nadpisywanie wyświetlacza o wartości wskazań czujników które uległy zmianie. Tutaj zastosowanie znajduje struktura lokalna prev_var ponieważ wewnątrz funkcji porównane zostają wartości z poprzedniego i aktualnego cyklu. W linijkach od 115 do 121 zainicjowane zostają zmienne lokalne przyjmujące wartości zmiennych obu struktur. Dalsza część poświęcona jest instrukcjom warunkowym gdzie następuje porównanie wartości poprzednich i aktualnych. Jeżeli wykryta zostanie zmiana, wtedy Arduino wypełnia na wyświetlaczu prostokąt danej zmiennej i wpisuje w niego aktualną wartość. Linijki 122 do 136 poświęcone są

zmiennym przechowującym wilgotność i temperaturę na co wskazują komentarze przy funkcjach.

Pozostałe instrukcje warunkowe porównujące zmienne z aktualnego i poprzedniego cyklu mikrokontrolera znajdują się na rysunku 8.13. Porównane są na nim odczyty czasu, czyli: godzina, minuta, sekunda, dzień miesiąca, miesiąc i rok. Wszystkie początki warunków posiadają komentarz, a ich zasada działania jest taka sama jak w przypadku wilgotności i temperatury. Warto zauważyć że ostatnia linijka funkcji (186) zawiera instrukcje która nadpisuje strukturę lokalną biblioteki prev_var o wartości które w danym cyklu były aktualne. Dzięki tej instrukcji możliwy jest proces porównywania wskazań który nastąpi w kolejnym cyklu mikrokontrolera.

```
138       tft_lib.setTextSize(5);
139       if (last.hour != DateNtime.hour) {       // godzina
140         tft_lib.fillRect(0, 170, 90, 38, BLACK);
141         tft_lib.setCursor(0, 170);
142         if (DateNtime.hour < 10)tft_lib.print("0");
143         tft_lib.print(DateNtime.hour);
144         tft_lib.println(":");
145       }
146       if (last.mins != DateNtime.mins) {        // minuta
147         tft_lib.fillRect(90, 170, 90, 38, BLACK);
148         tft_lib.setTextColor(WHITE);
149         tft_lib.setCursor(90, 170);
150         if (DateNtime.mins < 10)tft_lib.print("0");
151         tft_lib.print(DateNtime.mins);
152         tft_lib.println(":");
153       }
154       if (last.sec != DateNtime.sec) {        // sekunda
155         tft_lib.fillRect(180, 170, 90, 38, BLACK);
156         tft_lib.setTextColor(WHITE);
157         tft_lib.setCursor(180, 170);
158         if (DateNtime.sec < 10)tft_lib.print("0");
159         tft_lib.print(DateNtime.sec);
160       }
161       tft_lib.setTextSize(4);
162       if (last.day != DateNtime.day) {        // dzień miesiąca
163         tft_lib.fillRect(2, 230, 70, 38, BLACK);
164         tft_lib.setTextColor(WHITE);
165         tft_lib.setCursor(2, 230);
166         if (DateNtime.day < 10)tft_lib.print("0");
167         tft_lib.print(DateNtime.day);
168         tft_lib.println("-");
169       }
170       if (last.mounth != DateNtime.mounth) {        // miesiąc
171         tft_lib.fillRect(72, 230, 70, 38, BLACK);
172         tft_lib.setTextColor(WHITE);
173         tft_lib.setCursor(72, 230);
174         if (DateNtime.mounth < 10)tft_lib.print("0");
175         tft_lib.print(DateNtime.mounth);
176         tft_lib.println("-");
177       }
178       if (last.year != DateNtime.year) {        // rok
179         tft_lib.fillRect(142, 230, 98, 38, BLACK);
180         tft_lib.setTextColor(WHITE);
181         tft_lib.setCursor(142, 230);
182         tft_lib.print("20");
183         tft_lib.print(DateNtime.year);
184       }
185
186       prev_var = curr_var;
187   }
```

Rys. 8.13. Biblioteka menu.c linijki od 138 do 187

Kolejną funkcją omawianej biblioteki jest print_screen, która posiada strukturę wejściową menu_state. Przedstawiona jest na rysunkach 8.14 i 8.15. Odpowiada ona za wyświetlenie odpowiedniego ekranu zależnie od podmenu ustawień godziny, daty i czasu pracy pompki. Warto zauważyć że, wszystkie te ekrany (widoczne na rysunkach 7.3, 7.4 i 7.5) posiadają część wspólną czyli przyciski i wskazanie czasu, dodatkowo struktura menu_state przechowuje zmienną, która określa odpowiedni tryb pracy urządzenia. Pozwala to na napisanie wspólnego zbioru instrukcji do wyświetlenia odpowiedniego ekranu. Zależnie od wartości zmiennej struktury screen_state funkcja odpowiednio umiejscowi i wyświetli wszystkie pola informacyjne. Pierwsza część funkcji (linijki 189 do 201) odczytuje i zapisuje aktualną wartość czasu z RTC, a następnie w instrukcji warunkowej sprawdzana poprawność odczytu. Jeżeli DS1307 jest nieustawiony lub zresetowany, wtedy wyświetlane są wartości dnia, miesiąca i roku odpowiadające dacie 13.10.2021 (wtedy napisana została ta część biblioteki). Następna część funkcji zawiera inicjację zmiennych w których ustawiane będą pozycje przycisków i pól wyświetlacza i są ta zmienne px i py. Tablica scr_val przechowyje wartości czasu które mają zostać wyświetlone.

```
188 menu_state screen::print_screen(menu_state scr){
189     ds_class ds_set;
190     ds_val tim_set = ds_set.readAll();
191     tim_set.sec = 0;
192     scr.time = tim_set;
193 
194     if(scr.time.hour > 23 || scr.time.mins > 59 || scr.time.day > 31
195             || scr.time.mounth > 12 || scr.time.day == 0 || scr.time.mounth == 0){
196         scr.time.hour = 0;
197         scr.time.mins = 0;
198         scr.time.day = 13;
199         scr.time.mounth = 10;
200         scr.time.year = 21;
201     }
202 
203     uint16_t tim_px = 40,
204         tim_py = 80,
205         w = 160;
206     uint8_t scr_vals[3];
207 
208     tft_lib.fillScreen(BLACK);
209 
210     switch(scr.screen_state){
211     case LIGHT_SCREEN:
212         break;
213 
214     case SET_TIME_SCREEN:
215         tim_px = 60;
216         tim_py = 50;
217         scr_vals[0] = scr.time.hour;
218         scr_vals[1] = scr.time.mins;
219         break;
220 
221     case SET_DATE_SCREEN:
222         tim_px = 0;
223         tim_py = 10;
224         scr_vals[0] = scr.time.day;
225         scr_vals[1] = scr.time.mounth;
226         scr_vals[2] = scr.time.year;
227         w = 240;
228         break;
229 
230     case PUMP_START_TIME_SCREEN:
231         tim_px = 60;
232         tim_py = 80;
233 
234         scr_vals[0] = scr.time.hour;
235         scr_vals[1] = scr.time.mins;
236 
237         tft_lib.fillRect(0,0,240,25,CYAN);
```

Rys. 8.14. Biblioteka menu.c linijki od 188 do 237

```
238         tft_lib.drawRect(0,25,90,40,CYAN);
239         tft_lib.fillRect(90,25,150,40,RED);
240         tft_lib.drawRect(90,25,150,40,CYAN);
241         tft_lib.setTextSize(2);
242         tft_lib.setTextColor(BLACK);
243         tft_lib.setCursor(30,5);
244         tft_lib.print("Wybierz godzine:");
245         tft_lib.setCursor(10,35);
246         tft_lib.print("startu");
247         tft_lib.setCursor(100,35);
248         tft_lib.print("zatrzymania");
249         break;
250     }
251 
252     tft_lib.setTextSize(4);
253     tft_lib.setTextColor(BLACK);
254 
255 
256     if(scr.screen_state != SET_DATE_SCREEN){
257         tft_lib.fillRoundRect(tim_px - 20,tim_py + 60,w,50,10,WHITE);
258     }else{
259         tft_lib.fillRoundRect(tim_px,tim_py + 60,w,50,10,WHITE);
260     }
261     tft_lib.setCursor(tim_px,tim_py + 70);
262     if(scr_vals[0] < 10)tft_lib.print("0");
263     tft_lib.print(scr_vals[0]);
264     if(scr.screen_state == SET_DATE_SCREEN)tft_lib.print("-");
265     else tft_lib.print(":");
266     if(scr_vals[1] < 10)tft_lib.print("0");
267     tft_lib.print(scr_vals[1]);
268     if(scr.screen_state == SET_DATE_SCREEN){
269         tft_lib.print("-20");
270         tft_lib.print(scr_vals[2]);
271     }
272 
273     up_sign(tim_px,tim_py,WHITE,BLACK);
274     up_sign(tim_px + 70,tim_py,WHITE,BLACK);
275     down_sign(tim_px,tim_py + 120,WHITE,BLACK);
276     down_sign(tim_px + 70,tim_py + 120,WHITE,BLACK);
277 
278     if(scr.screen_state == SET_DATE_SCREEN){
279         up_sign(tim_px + 160,tim_py,WHITE,BLACK);
280         down_sign(tim_px + 160,tim_py + 120,WHITE,BLACK);
281     }
282 
283     exit_sign(180,260);
284     ok_sign(10,260);
285 
286     return scr;
287 }
```

Rys. 8.15. Biblioteka menu.c linijki od 238 do 287

Część funkcji gdzie następuje faktyczny zapis wyświetlacza rozpoczyna się od linijki 208, gdzie cały ekran wypełniony zostaje kolorem czarnym. Dalej widoczna jest instrukcja warunkowa switch, która zależnie od wybranego podmenu ustawiać będzie w odpowiedni sposób wcześniej zainicjowane zmienne lokalne. W przypadku ekranu pompki zapis wyświetlacza następuje w linijkach 237 do 248. Efekt końcowy widoczny jest na rysunku 7.5. W linijkach 256 do 271 znajdują się warunki wraz z instrukcjami wewnątrz których znajdują się instrukcje w odpowiedni sposób nadpisujące wyświetlacz zależnie od tego czy wyświetlane jest menu daty czy menu godziny. W ostatnich linijkach rysowane są przyciski wyjścia z menu oraz zatwierdzenia ustawień (exit_sign i ok_sign) oraz zwracana jest struktura wejściowa.

```
290 menu_state screen::check_screen_routine(menu_state scr){
291     uint16_t px = scr.px,
292             py = scr.py;
293 
294     uint16_t tim_px = 50,
295             tim_py = 0;
296 
297     uint16_t btn_pos_x[6];
298     uint16_t btn_pos_y[6];
299 
300     switch(scr.screen_state){
301     case SET_TIME_SCREEN:
302         tim_px = 60;
303         tim_py = 50;
304         break;
305 
306     case SET_DATE_SCREEN:
307         tim_px = 0;
308         tim_py = 10;
309         break;
310 
311     case PUMP_START_TIME_SCREEN:
312         tim_px = 60;
313         tim_py = 80;
314         break;
315 
316     }
317 
318     btn_pos_x[0] = tim_px;
319     btn_pos_y[0] = tim_py;
320 
321     btn_pos_x[1] = tim_px + 70;
322     btn_pos_y[1] = tim_py;
323 
324     btn_pos_x[2] = tim_px;
325     btn_pos_y[2] = tim_py + 120;
326 
327     btn_pos_x[3] = tim_px + 70;
328     btn_pos_y[3] = tim_py + 120;
329 
330     btn_pos_x[4] = tim_px + 160;
331     btn_pos_y[4] = tim_py;
332 
333     btn_pos_x[5] = tim_px + 160;
334     btn_pos_y[5] = tim_py + 120;
335 
336     uint8_t i,btn_quantity = 4 , nr_of_clicked_button = 0xFF;
337     if(scr.screen_state == SET_DATE_SCREEN)btn_quantity = 6;
338     for(i=0;i<btn_quantity;i++){
339 
```

Rys. 8.16. Biblioteka menu.c linijki od 290 do 339

Następną funkcją jest check_screen_routine (rys. 8.16, 8.17 i 8.16) nadpisuje ona strukturę menu_state, zależnie od interakcji jaką podejmie użytkownik z wyświetlaczem dotykowym. Początek funkcji jest podobny jak w print_screen. Inicjowane są zmienne lokalne określające pozycje przycisków znajdujących się w podmenu godziny, daty i czasu podlewania. Zainicjowane są również zmienne które określają współrzędne dotkniętego punktu na wyświetlaczu.

```
335
336     uint8_t i,btn_quantity = 4 , nr_of_clicked_button = 0xFF;
337     if(scr.screen_state == SET_DATE_SCREEN)btn_quantity = 6;
338     for(i=0;i<btn_quantity;i++){
339
340         if(px > btn_pos_x[i] && px < btn_pos_x[i] + 50 && py > btn_pos_y[i] && py < btn_pos_y[i] + 50){
341             nr_of_clicked_button = i;
342             if(scr.screen_state != SET_DATE_SCREEN){
343                 switch(i){
344                 case 0:
345                     if(scr.time.hour < 23)scr.time.hour++;
346                     break;
347                 case 1:
348                     if(scr.time.mins < 59)scr.time.mins++;
349                     break;
350                 case 2:
351                     if(scr.time.hour > 0)scr.time.hour--;
352                     break;
353                 case 3:
354                     if(scr.time.mins > 0)scr.time.mins--;
355                     break;
356                 }
357             }else{
358                 switch(i){
359                 case 0:
360                     if(scr.time.day < 31)scr.time.day++;
361                     break;
362                 case 1:
363                     if(scr.time.mounth < 12)scr.time.mounth++;
364                     break;
365                 case 2:
366                     if(scr.time.day > 1)scr.time.day--;
367                     break;
368                 case 3:
369                     if(scr.time.mounth > 1)scr.time.mounth--;
370                     break;
371                 case 4:
372                     scr.time.year++;
373                     break;
374                 case 5:
375                     if(scr.time.year > 21)scr.time.year--;
376                     break;
377                 }
378             }
379         }
380     }
381
382     if( nr_of_clicked_button == 0 || nr_of_clicked_button == 1 || nr_of_clicked_button == 4){
383         up_sign(btn_pos_x[nr_of_clicked_button],btn_pos_y[nr_of_clicked_button],WHITE,GREEN);
384         delay(30);
```

Rys. 8.17. Biblioteka menu.c linijki od 336 do 384

```
380     }
381
382     if( nr_of_clicked_button == 0 || nr_of_clicked_button == 1 || nr_of_clicked_button == 4){
383         up_sign(btn_pos_x[nr_of_clicked_button],btn_pos_y[nr_of_clicked_button],WHITE,GREEN);
384         delay(30);
385         up_sign(btn_pos_x[nr_of_clicked_button],btn_pos_y[nr_of_clicked_button],WHITE,BLACK);
386     }
387     if( nr_of_clicked_button == 2 ||nr_of_clicked_button == 3 || nr_of_clicked_button == 5){
388         down_sign(btn_pos_x[nr_of_clicked_button],btn_pos_y[nr_of_clicked_button],WHITE,GREEN);
389         delay(30);
390         down_sign(btn_pos_x[nr_of_clicked_button],btn_pos_y[nr_of_clicked_button],WHITE,BLACK);
391     }
392
393     tft_lib.setTextColor(BLACK);
394     tft_lib.setTextSize(4);
395
396     if(scr.screen_state != SET_DATE_SCREEN){
397     if( nr_of_clicked_button == 0 ||nr_of_clicked_button == 2){
398         tft_lib.fillRoundRect(tim_px,tim_py + 60,50,50,10,WHITE);
399         tft_lib.setCursor(tim_px,tim_py + 70);
400         if(scr.time.hour < 10)tft_lib.print("0");
401         tft_lib.print(scr.time.hour);
402     }
403     if( nr_of_clicked_button == 1 ||nr_of_clicked_button == 3){
404         tft_lib.fillRoundRect(tim_px + 70,tim_py + 60,70,50,10,WHITE);
405         tft_lib.setCursor(tim_px + 70,tim_py + 70);
406         if(scr.time.mins < 10)tft_lib.print("0");
407         tft_lib.print(scr.time.mins);
408     }
409     }else{
410         if( nr_of_clicked_button == 0 ||nr_of_clicked_button == 2){
411             tft_lib.fillRoundRect(tim_px,tim_py + 60,50,50,10,WHITE);
412             tft_lib.setCursor(tim_px,tim_py + 70);
413             if(scr.time.day < 10)tft_lib.print("0");
414             tft_lib.print(scr.time.day);
415         }
416         if( nr_of_clicked_button == 1 ||nr_of_clicked_button == 3){
417             tft_lib.fillRoundRect(tim_px + 70,tim_py + 60,50,50,10,WHITE);
418             tft_lib.setCursor(tim_px + 70,tim_py + 70);
419             if(scr.time.mounth < 10)tft_lib.print("0");
420             tft_lib.print(scr.time.mounth);
421         }
422         if( nr_of_clicked_button == 4 ||nr_of_clicked_button == 5){
423             tft_lib.fillRoundRect(tim_px + 190,tim_py + 60,50,50,10,WHITE);
424             tft_lib.setCursor(tim_px + 192,tim_py + 70);
425             tft_lib.print(scr.time.year);
426         }
427     }
428     return scr;
429 }
```

Rys. 8.18. Biblioteka menu.c linijki od 380 do 429

Dalej na rysunku 8.17 widoczne są instrukcje warunkowe (linijki 340 do 380), które zwiększają lub zmniejszają licznik zmienianej wartości czasu, zależnie od danego podmenu. Nadpisane zostają wtedy zmienne czasu struktury menu_state. Warunki działają wewnątrz pętli for (linijka 338) i opierają się na zainicjowanych współrzędnych przycisków. Kolejne instrukcje warunkowe znajdują się w linijkach od 382 do 391. Po wykryciu kliknięgo przycisku zmieniają kolor strzałki na zielony a po czasie 30 ms z powrotem na czarny. Ma to na celu potwierdzenie przez Arduino że poprawnie zostały odebrane dane o kliknięciu przycisku. Następny zbiór warunków znajdujący się w linijkach 398 do 427, nadpisuje znajdujące się aktualnie na wyświetlaczu zmieniane wartości czasu. Działania wewnątrz warunków charakteryzują się tą samą zasadą działania co nadpisywanie ekranu głównego. Prostokąty wewnątrz których wartość uległa modyfikacji wypełniane są kolorem tła. Następnie wpisywana

jest w nie aktualna wartość danej zmiennej. Po przejściu przez wszystkie warunki funkcja zwraca nadpisaną strukturę scr jako dane wyjściowe.

```
431 void screen::print_light_screen(menu_state scr){
432     tft_lib.fillScreen(BLACK);
433     tft_lib.fillRect(0,0,240,40,CYAN);
434     tft_lib.drawRect(0,0,240,40,GREEN);
435     tft_lib.setCursor(5,10);
436     tft_lib.setTextSize(3);
437     tft_lib.setTextColor(BLACK);
438     tft_lib.print("ustaw jasnosc");
439     tft_lib.setTextSize(2);
440     tft_lib.drawRect(20,100,200,20,CYAN);
441     tft_lib.fillRect(20,120,2,10,CYAN);
442     tft_lib.fillRect(218,120,2,10,CYAN);
443     tft_lib.fillRect(119,120,2,10,CYAN);
444     tft_lib.setCursor(10,140);
445     tft_lib.setTextColor(CYAN);
446     tft_lib.print("0%");
447     tft_lib.setCursor(103,140);
448     tft_lib.print("50%");
449     tft_lib.setCursor(190,140);
450     tft_lib.print("100%");
451     tft_lib.fillRect(scr.light_px,101,4,18,YELLOW);
452     ok_sign(180,260);
453 }
454 menu_state screen::check_light_screen_routine(menu_state scr){
455
456     uint16_t px = scr.px,
457              py = scr.py;
458     if(px >= 0xFFF)return scr;
459     uint8_t light_val;
460     if(py > 80 && py < 140){
461         tft_lib.fillRect(21,101,198,18,BLACK);
462         if(px > 20 && px < 216){
463             light_val = px - 20;
464             light_val /= 2;
465             light_val *= 2.55;
466         }
467         if(px<=20){
468             light_val=0;
469             px = 21;
470         }
471         else if(px>=216){
472             light_val=255;
473             px = 215;
474         }
475         tft_lib.fillRect(px,101,4,18,YELLOW);
476         pwm(light_val);
477         scr.light_px = px;
478     }
479     return scr;
480 }
```

Rys. 8.19. Biblioteka menu.c linijki od 431 do 480

Na rysunku 8.19 przedstawione są funkcje odpowiedzialne za wyświetlanie podmenu jasności i są to print_light_screen oraz check_light_screen_routine. Pierwsza funkcja używa komend nadpisujących wyświetlacz, tak by pokazywał on ekran zmiany jasności. Efekt działania tej funkcji widoczny jest na rysunku 7.7. Jedyną zmienną którą ta funkcja pobiera ze struktury menu_state jest wskaźnik pozycji procentowej jasności czyli light_px. Zasada działania funkcji jest oczywista i nie wymaga głębszej analizy.

Druga funkcja na rysunku powyżej (linijki 454 do 480) odpowiada za zmianę położenia wskaźnika pozycji na wyświetlaczu oraz ustawienie sygnału PWM zależnie od tej pozycji. Funkcja inicjuje zmienne współrzędnych w których wykryte zostało wciśnięcie. Następnie

jeżeli mieszczą się one w odpowiednim zakresie przeliczane są na wartość rejestru ustalającego wypełnienie sygnału PWM. Po czym przeliczona wartość zostaje przekazana do makra pwm w linijce 476. Dodatkowo jeżeli użytkownik kliknie w miejsce na lewo bądź na prawo od pola na którym może się znaleźć wskaźnik. Wtedy odpowiednio wskaźnik zostanie ustawiony na 0% lub na 100%. Warunki które to określają znajdują się w linijkach 426 do 474.

```
482 menu_state screen::check_btn_routine(menu_state screenState){
483     uint16_t px = screenState.px;
484     uint16_t py = screenState.py;
485 
486     switch(screenState.screen_state){
487     case MAIN_SCREEN:
488         if(py < 40){  // zostanie wciśnięty przycisk 'ustawienia'
489             print_main_menu_screen();
490             screenState.screen_state = MAIN_MENU_SCREEN;
491         }
492         break;
493     case MAIN_MENU_SCREEN:
494         if(px > 180 && px < 230 && py > 260 && py < 310){  // został wciśnięty przycisk powrotu do ekranu głównego
495             prev_var = {0};   // aby odświerzyc ekran startowy
496             print_main_screen();
497             screenState.screen_state = MAIN_SCREEN;
498         }
499         if(py > 10 && py < 50){  // został wciśnięty przycisk ustawienia godziny
500             screenState.screen_state = SET_TIME_SCREEN;
501             screenState = print_screen(screenState);
502         }
503         if(py > 60 && py < 100){  // został wciśnięty przycisk ustawienia godziny
504             screenState.screen_state = SET_DATE_SCREEN;
505             screenState = print_screen(screenState);
506         }
507         if(py > 110 && py < 150){  // został wciśnięty przycisk ustawień czasu pracy pompki
508             screenState.screen_state = PUMP_START_TIME_SCREEN;
509             screenState = print_screen(screenState);
510             screenState.pump_val.end_time = {0};
511         }
512         if(py > 160 && py < 200){  // został wciśnięty przycisk ustawień jasnosci
513             screenState.screen_state = LIGHT_SCREEN;
514             print_light_screen(screenState);
515         }
516         break;
517     case SET_TIME_SCREEN:
518         if(px > 180 && px < 230 && py > 260 && py < 310){  // został wciśnięty przycisk powrotu do menu głównego
519             print_main_menu_screen();
520             screenState.screen_state = MAIN_MENU_SCREEN;
521         }
522         if(px > 10 && px < 60 && py > 260 && py < 310){  // został wciśnięty przycisk zatwierdzenia godziny
523             ds_class ds_set;
524             screenState.time.sec = 0;
525             ds_set.setTime(screenState.time);
526             print_main_menu_screen();
527             screenState.screen_state = MAIN_MENU_SCREEN;
528 
529         }
530         break;
531     case SET_DATE_SCREEN:
```

Rys. 8.20. Biblioteka menu.c linijki od 482 do 531

```
531     case SET_DATE_SCREEN:
532         if(px > 180 && px < 230 && py > 260 && py < 310){  // został wciśnięty przycisk powrotu do menu głównego
533             prev_var = {0};   // aby odświerzyc ekran startowy
534             print_main_menu_screen();
535             screenState.screen_state = MAIN_MENU_SCREEN;
536         }
537         if(px > 10 && px < 60 && py > 260 && py < 310){  // został wciśnięty przycisk zatwierdzenia godziny
538             ds_class ds_set;
539             ds_set.setTime(screenState.time);
540             prev_var = {0};   // aby odświerzyc ekran startowy
541             print_main_menu_screen();
542             screenState.screen_state = MAIN_MENU_SCREEN;
543 
544         }
545         break;
546     case PUMP_START_TIME_SCREEN:
547         if(px > 180 && px < 230 && py > 260 && py < 310){  // został wciśnięty przycisk powrotu do menu głównego
548             print_main_menu_screen();
549             screenState.last_var = 0;
550             screenState.screen_state = MAIN_MENU_SCREEN;
551         }
552         if(px > 10 && px < 60 && py > 260 && py < 310 && screenState.pump_val.in_use == END_TIME_SETTINGS){
553             // został wciśnięty przycisk zatwierdzenia godziny
554             if(screenState.last_var == END_TIME_SETTINGS)screenState.pump_val.end_time = screenState.time;
555             switch(screenState.last_var){
556             case END_TIME_SETTINGS:
557                 screenState.pump_val.end_time = screenState.time;
558                 break;
559             case START_TIME_SETTINGS:
560                 screenState.pump_val.start_time = screenState.time;
561                 break;
562             }
563 
564             screenState.pump_val.in_use = PUMP_WAITING_ST;
565             print_main_menu_screen();
566             screenState.screen_state = MAIN_MENU_SCREEN;
567             screenState.last_var = 0;
568 
569         }
570         if(px < 90 && py > 25 && py < 65 && screenState.last_var != START_TIME_SETTINGS){
571             // został wciśnięty przycisk wyboru godziny rozpoczęcia podlewania
572             screenState.last_var = START_TIME_SETTINGS;
573             tft_lib.fillRect(0,25,90,40,GREEN);
574             tft_lib.drawRect(0,25,90,40,CYAN);
575             tft_lib.fillRect(90,25,150,40,RED);
576             tft_lib.drawRect(90,25,150,40,CYAN);
577             tft_lib.setCursor(10,35);
578             tft_lib.setTextSize(2);
579             tft_lib.print("startu");
580             tft_lib.setCursor(100,35);
```

Rys. 8.21. Biblioteka menu.c linijki od 531 do 580

```
566             screenState.screen_state = MAIN_MENU_SCREEN;
567             screenState.last_var = 0;
568 
569         }
570         if(px < 90 && py > 25 && py < 65 && screenState.last_var != START_TIME_SETTINGS){
571             // został wciśnięty przycisk wyboru godziny rozpoczęcia podlewania
572             screenState.last_var = START_TIME_SETTINGS;
573             tft_lib.fillRect(0,25,90,40,GREEN);
574             tft_lib.drawRect(0,25,90,40,CYAN);
575             tft_lib.fillRect(90,25,150,40,RED);
576             tft_lib.drawRect(90,25,150,40,CYAN);
577             tft_lib.setCursor(10,35);
578             tft_lib.setTextSize(2);
579             tft_lib.print("startu");
580             tft_lib.setCursor(100,35);
581             tft_lib.print("zatrzymania");
582             screenState.pump_val.end_time = screenState.time;
583         }
584         if(px > 90 && py > 25 && py < 65 && screenState.last_var != END_TIME_SETTINGS){
585             // został wciśnięty przycisk wyboru godziny zakończenia podlewania
586             screenState.last_var = END_TIME_SETTINGS;
587             tft_lib.fillRect(0,25,90,40,RED);
588             tft_lib.drawRect(0,25,90,40,CYAN);
589             tft_lib.fillRect(90,25,150,40,GREEN);
590             tft_lib.drawRect(90,25,150,40,CYAN);
591             tft_lib.setCursor(10,35);
592             tft_lib.setTextSize(2);
593             tft_lib.print("startu");
594             tft_lib.setCursor(100,35);
595             tft_lib.print("zatrzymania");
596             screenState.pump_val.start_time = screenState.time;
597             screenState.pump_val.in_use = END_TIME_SETTINGS;
598         }
599         break;
600     case LIGHT_SCREEN:
601         screenState = check_light_screen_routine(screenState);
602         if(px > 180 && px < 230 && py > 260 && py < 310){  // został wciśnięty przycisk powrotu do menu głównego
603             print_main_menu_screen();
604             screenState.last_var = 0;
605             screenState.screen_state = MAIN_MENU_SCREEN;
606         }
607         break;
608     }
609     if(screenState.screen_state == SET_DATE_SCREEN || screenState.screen_state == SET_TIME_SCREEN
610             || screenState.screen_state == PUMP_START_TIME_SCREEN)
611         screenState = check_screen_routine(screenState);
612     return screenState;
613 }
```

Rys. 8.22. Biblioteka menu.c linijki od 566 do 613

Ostatnią funkcją znajdującą się w bibliotece menu.c jest check_btn_routine która poza nadpisywaniem struktury menu_state korzysta ze wszystkich opisanych w tym podrozdziale funkcji. Można określić ją dodatkowo jako łącznik między bibliotekami a częścią główną programu. Zapewni ona całkowitą wymianę danych o wszystkich odczytywanych wartościach w pętli głównej programu poprzez strukturę menu_state. Ta funkcja także będzie zarządzać zmianami dokonywanymi w menu jak również aktualizacją danych podlegających zmianom we wszystkich podmenu. Funkcja check_btn_routine prezentuje ideę działania całego kodu. Polega ona na wywoływaniu jednej linijki w pętli głównej programu, która w każdym cyklu zapewni odpowiednią reakcję na kliknięcie ekranu niezależnie od menu lub podmenu w którym się znajduje.

Wewnątrz funkcji zainicjowane zostają zmienne współrzędnych dotkniętego punktu px i py na podstawie których interpretowane będą dalej wciśnięte przyciski. Dalej w funkcji znajduje

się instrukcja warunkowa switch, która rozpatruje zmienną screen_state informującą o stanie menu. Zmienna ta jest pobrana ze struktury screenState i aktualizowana wewnątrz funkcji. Makra informujące o rodzaju menu rozpatrywane są wewnątrz wspomnianego warunku. Na podstawie ich wartości podejmowane są dalsze instrukcje w oparciu o współrzędne px i py. Pierwszym rozpatrywanym warunkiem jest ekran główny widoczny na rysunku 8.20 (linijki 487 do 491). Jeżeli został wciśnięty przycisk ustawień, czyli py jest większe niż 40, wtedy wywoływana jest odpowiednia funkcja czyli print_main_menu_screen. Wyświetla ona menu główne. Następnie zmienna struktury stanu menu (screen_state) nadpisywana jest wartością makra menu głównego. Kolejne rozpatrywane menu i podmenu w warunku switch (linijka 486) charakteryzują się tą samą zasadą działania. Wszystkie możliwości wciskanych przycisków opisane są w kodzie odpowiednimi komentarzami. Dodatkowo jeżeli w danym podmenu nastąpi zatwierdzenie to wewnątrz warunku użyta zostanie funkcja zapisująca daną zmienną, strukturę lub rejestr. Jedyną różnicą od wykonywania instrukcji wewnątrz warunków stanowi przypadek menu ustawień czasu podlewania przedstawiony na rysunkach 8.21 i 8.22. W linijkach od 570 do 598 rozpatrywane są dodatkowo przypadki kliknięcia przycisku „startu" lub „zakończenia". Wtedy zmieniony zostaje kolor tła obu przycisków co można zaobserwować na rysunkach 7.5 i 7.6. W ostatnich linijkach (609 - 611) omawianej funkcji znajduje się zbiorczy warunek dla podmenu ustawień godziny, daty i czasu pracy pompki. Wywołuje on funkcję check_screen_routine zapewniającą dalszą interakcję która została już wcześniej opisana. Na końcu funkcji zwracana jest wartość nadpisanej struktury zależnie od miejsca kliknięcia matrycy.

8.3. Część główna programu

Ostatnią omawianą częścią programu sterownika ogrodowego jest część główna, czyli plik main.c. Zainicjowane tutaj zostały wszystkie zaprezentowane w tym rozdziale klasy oraz struktury. Dodatkowo zapisane są tutaj instrukcje wykonywane jednokrotnie przy uruchomieniu urządzenia (void setup(void)), a także instrukcje wewnątrz pętli głównej programu czyli void loop().

Na rysunku 8.23 widoczny jest początek programu. W linijkach od 1 do 7 dodane zostały wszystkie biblioteki omówione w rozdziale 8. Dalsza cześć programu czyli linijki 9 do 25 zawierają definicje:

- wyprowadzenia czujnika DHT22 (linijka 9),

- wyprowadzeń do odczytu współrzędnych dotkniętego punktu na wyświetlaczu dotykowym (linijki 11 do 14),
- wartości krańcowe jakie mogą zostać odczytane z wyświetlacza dotykowego (linijki 16 do 19),
- wyprowadzeń linii specjalnych protokołu równoległego wyświetlacza (linijki 21 do 25).

```
1  #include <Adafruit_GFX.h>    // Core graphics library
2  #include <Adafruit_TFTLCD.h> // Hardware-specific library
3  #include <TouchScreen.h>
4  #include "DHTStable.h"
5  #include "i2c.h"
6  #include "ds1307.h"
7  #include "menu_lib.h"
8
9  #define DHT22_PIN A5
10
11 #define YP A3
12 #define XM A2
13 #define YM 9
14 #define XP 8
15
16 #define TS_MINX 150
17 #define TS_MINY 120
18 #define TS_MAXX 920
19 #define TS_MAXY 940
20
21 #define LCD_CS A3
22 #define LCD_CD A2
23 #define LCD_WR A1
24 #define LCD_RD A0
25 #define LCD_RESET A4
26
27 TouchScreen ts = TouchScreen(XP, YP, XM, YM, 300);
28 i2c_class i2c;
29 ds_class ds;
30 DHTStable DHT;
31 Adafruit_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);
32 screen ili9341;
33 ds_val DateNtime = {0},last = {0};
34 screen_var recent_var;
35 menu_state btn_state;
36 ds_class ds1307;
37 float lastHum = 0, recHum, lastTemp = 0, recTemp;
38
39 void setup(void) {
40
41   tft = ili9341.initScreen(tft);
42   ili9341.print_main_screen();
43   btn_state.screen_state = MAIN_SCREEN;
44   pinMode(10,OUTPUT);
45   DateNtime = ds.readAll();
46   if(DateNtime.year < 21){
47     ds1307.setSingleReg(yearReg,(2<<4)|1);
48     btn_state.time.year = 21;
49   }
50 }
```

Rys. 8.23. main.c linijki 1 do 50

Deklaracje klas i struktur zawarte zostały w linijkach od 27 do 37. Poza opisanymi już wcześniej deklaracjami widoczna jest inicjacja zmiennych zmiennoprzecinkowych w linijce 37. Przechowywać one będą odczyty wskazań temperatury i wilgotności z aktualnego i poprzedniego cyklu programu. W kolejnej części programu widoczne są instrukcje wykonywane jeden raz (po zasileniu mikrokontrolera). Mowa tutaj o linijkach 39 do 50 w

funkcji setup. Zostaje tutaj zainicjowany sterownik ILI9341 oraz nadpisana struktura wyświetlacza. Następnie wyświetlony zostaje ekran główny, a struktura przechowująca dane o stanie menu zostaje zapisana przez odpowiednie makro ekranu głównego (linijka 43). Dalej pin nr. 10 Arduino ustawiony zostaje jako wyjście cyfrowe (linijka 44) ponieważ znajduje się na nim wyjście sterujące pracą pompki. W ostatniej części funkcji setup odczytywany zostaje czas i jeżeli odczytany rok jest mniejszy niż 21 (2021), wtedy rejestr roku zostaje zapisany tą właśnie wartością.

```
43   btn_state.screen_state = MAIN_SCREEN;
44   pinMode(10,OUTPUT);
45   DateNtime = ds.readAll();
46   if(DateNtime.year < 21){
47     ds1307.setSingleReg(yearReg,(2<<4)|1);
48     btn_state.time.year = 21;
49   }
50 }
51
52 void loop()
53 {
54   int chk = DHT.read22(DHT22_PIN);
55   float hum = DHT.getHumidity();
56   float temp = DHT.getTemperature();
57
58   DateNtime = ds.readAll();
59
60   if ( hum > 0 ){
61     recHum = hum;
62     recTemp = temp;
63   }
64
65   if(btn_state.screen_state == MAIN_SCREEN){
66   recent_var.Hum = recHum;
67   recent_var.Temp = recTemp;
68   recent_var.dateVar = DateNtime;
69   ili9341.updateMainScreen(recent_var);
70   }
71
72   if(btn_state.pump_val.in_use ==  PUMP_WAITING_ST && DateNtime.mins >= btn_state.pump_val.start_time.mins && DateNtime.hour >= btn_state.pump_val.start_time.hour){
73     btn_state.pump_val.in_use = PUMP_TURN_ON;
74     SET_PUMP_ON;
75   }
76   if(btn_state.pump_val.in_use ==  PUMP_TURN_ON && DateNtime.mins >= btn_state.pump_val.end_time.mins && DateNtime.hour >= btn_state.pump_val.end_time.hour){
77     btn_state.pump_val.in_use = PUMP_TURN_OFF;
78     SET_PUMP_OFF;
79   }
80
81   TSPoint p = ts.getPoint();
82   pinMode(XM, OUTPUT);
83   pinMode(YP, OUTPUT);
84
85   if (p.z > 10 && p.z < 1000) {
86     p.x = map(p.x, TS_MINX, TS_MAXX, tft.width(), 0);
87     p.y = map(p.y, TS_MINY, TS_MAXY, tft.height(), 0);
88     btn_state.px = p.x;
89     btn_state.py = p.y;
90     btn_state = ili9341.check_btn_routine(btn_state);
91   }
92 }
```

Rys. 8.24. main.c linijki 44 do 93

Na rysunku 8.24 przedstawiona została pętla główna programu znajdująca się w funkcji loop. Wszystkie instrukcje znajdujące się wewnątrz wykonywane będą podczas każdego cyklu mikrokontrolera. Na początku widoczne są deklaracje zmiennych lokalnych do których zapisywane są odczyty temperatury i wilgotności z czujnika DHT22 oraz następuje odczyt

aktualnych wartości czasu. Następnie sprawdzana jest poprawność odczytanych pomiarów (linijka 60). Jeżeli wyniki są poprawne wtedy następuje ich zapis do zmiennych z przedrostkiem rec. W dalszej części funkcji widoczny jest kolejny warunek gdzie sprawdzany zostaje rodzaj wyświetlanego ekranu. Jeżeli zmienna przyjmuje wartość makra ekranu głównego wtedy aktualne wartości czasu i odczytów DHT22 zapisane zostają do struktury recent_var. Wywołana zostaje funkcja aktualizująca wskazania na wyświetlaczu, a recent_var są jej danymi wejściowymi. Kolejne instrukcje warunkowe widoczne w linijkach od 72 do 79 zarządzają włączanie i wyłączaniem pompki wody. Jeżeli użytkownik zapisał w menu „pompka" godzinę rozpoczęcia i zakończenia, wtedy procesor po odczytaniu godziny rozpoczęcia z układu RTC, ustawi stan wysoki na wyjście sterujące pompką. Jeżeli natomiast wykryta zostanie godzina zakończenia podlewania, a Arduino uprzednio wystawi stan wysoki na wyjście pompki, wtedy nastąpi przełączenie na stan niski. Makra i zmienne struktur użyte w tych warunkach przedstawione zostały w bibliotece menu.h. Przechodząc dalej, w linijce 81 następuje odczyt wyprowadzeń wyświetlacza dotykowego a dane odczytane z wejść ADC przekazywane są do struktury p. Po wykonaniu odczytu linie XM i YP ustawiane zostają ponownie jako wyjścia cyfrowe (używane przez ILI9341) a następnie wykonywana jest instrukcja warunkowa w linijce 85. Sprawdzana jest tutaj zmienna która informuje o sile z jaką użytkownik dotkną matrycę wyświetlacza dotykowego. Jeżeli siła dotyku jest większa niż 10 i mniejsza niż 1000 (są to wartości umowne) wtedy wykonywane zostają instrukcje wewnątrz warunku. Przeliczane są wtedy zmienne x i y na wartości szerokości i długości matrycy wyświetlacza, gdzie x mieści się w zakresie od 0 do 240 (tft.width()), a y w zakresie od 0 do 320 (tft.height()). Następnie przeliczone współrzędne przekazywane są do struktury btn_state, po czym wywoływana jest funkcja check_btn_routine nadpisująca ową strukturę. Jak zostało to opisane w podrozdziale 8.2.6 współrzędne x oraz y poddawane są analizie. Na jej podstawie dokonywany jest kolejny zbiór warunków i instrukcji takich jak wejście do menu czy reakcja na wciśnięcie danego przycisku. Pętla po wykonaniu wszystkich instrukcji wraca do linijki 53 gdzie rozpoczyna się kolejny cykl mikrokontrolera.

9. Podsumowanie

Po wykonaniu schematu sterownika ogrodowego, oraz połączenia wszystkich podzespołów na jego podstawie, możliwe było przystąpienie do napisania kodu przy użyciu platformy Arduino. Sam kod, jak zostało to opisane w rozdziale 8, podzielony został na 3 części, czyli:

- gotowe biblioteki ze zbioru Arduino,
- biblioteki autorskie do sterowania dedykowanymi funkcjami sterownika ogrodowego,
- część główna programu wykorzystująca funkcje bibliotek a także sterująca pracą wszystkich elementów podrzędnych użytych w projekcie.

Taki podział zapewnia przejrzystość kodu, a także eliminuje powstawanie niepożądanych stanów pracy urządzenia. Dodatkowo dzięki znajomości protokołu I^2C opisanego w rozdziale 4.1 możliwe było napisanie biblioteki „i2c" wykorzystującej dowolne dwa porty GPIO. Rozwiązało to problem zastosowania standardowej biblioteki I^2C ze zbioru bibliotek Arduino, ponieważ opiera się ona na rejestrach dedykowanych dla wyprowadzeń których używa w projekcie wyświetlacz dotykowy.

Po napisaniu bibliotek oraz części głównej programu całość została skompilowana i wgrana na płytkę Arduino Uno. Sam program wymagał niewielkich poprawek a jego działanie zostało omówione w rozdziale 7. Widoczny tam interfejs użytkownika w sposób czytelny informuje o aktualnych wskazaniach układów podrzędnych. Dodatkowo spełnione zostały wszystkie założenia o zasadzie działania sterownika ogrodowego a są nimi ustawianie:

- aktualnej godziny i daty,
- czasu pracy pompki, która podlewa uprawy w ogrodzie,
- jasności oświetlenia zewnętrznego.

Wszystkie te operacje są możliwe dzięki zastosowaniu dwupoziomowego menu opartego na wskazaniach pozycji odczytanych z wyświetlacza dotykowego. Zbiór działań które mogą być wykonywane przy użyciu menu został kilkukrotnie przetestowany na działającym urządzeniu. Po analizie wszystkich stanów pracy sterownika możliwe jest stwierdzenie że działa on poprawnie i może być zastosowany w domu jednorodzinnym. Dzięki swoim funkcjonalnościom sterownik mógłby w realny sposób ułatwić wykonywanie codziennych czynności jakimi są podlewanie ogrodu czy sterowanie oświetleniem zewnętrznym. Wykonanie projektu w oparciu o platformę Arduino tym samym potwierdza początkowe założenie. Projekty inteligentnych instalacji można wykonywać nie tylko w oparciu o system KNX/EIB.

Mikrokontrolery spełniają te same funkcje i pozwalają na elastyczne dopasowanie do potrzeb potencjalnego nabywcy.

10. Literatura

Książki:

[1] Baranowski R.: Mikrokontrolery AVR ATmega w praktyce. Wydawnictwo BTC. Warszawa 2005.

[2] Doliński J.: Mikrokontrolery AVR w praktyce. Wydawnictwo BTC. Warszawa 2003.

[3] Duszczyk K., Dubrawski A., Dubrawski A., Pawlik M., Szafrański M.: Inteligentny budynek. Wydawnictwo Naukowe PWN. Warszawa 2019.

[4] Francuz T.: Język C dla mikrokontrolerów AVR: od podstaw do zaawansowanych aplikacji. Wydawnictwo Helion. Katowice 2011.

[5] Margolis M.,Jepson B.,Weldin N.: Arduino. Przepisy na rozpoczęcie, rozszerzanie i udoskonalanie projektów. Wydawnictwo Helion. Londyn 2021.

Noty katalogowe:

[6] https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf, (dostęp: 10.01.2022)

[7] https://cdn-shop.adafruit.com/datasheets/ILI9341.pdf, (dostęp: 10.01.2022)

[8] https://datasheets.maximintegrated.com/en/ds/DS1307.pdf, (dostęp: 10.01.2022)

[9] https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf, (dostęp: 10.01.2022)

11.Spis ilustracji