
	Instytut Informatyki Politechniki Śląskiej Zespół Mikroinformatyki i Teorii Automatów Cyfrowych			
Rok akademicki:	Rodzaj studiów*: SSI/NSI/NSM	Przedmiot (Języki Asemblerowe/SMiW):	Grupa	Sekcja
2016/2017	SSI	SMiW		8
Imię:	Kamil		JP	
Nazwisko:	Ziętek			
<p style="text-align: center;"><i>Raport końcowy</i></p>				
<p>Temat projektu:</p> <p style="text-align: center; margin-top: 100px;">Gra zręcznościowa w ustalanie trasy przejazdu pojazdów</p>				
Data oddania: dd/mm/rrrr			07/02/2017	

1. Temat projektu i opis założeń

Głównym założeniem projektu jest stworzenie urządzenia, będącego grą zręcznościową w ustalanie trasy przejazdu pojazdów. Celem gry jest takie ustalenie trasy, aby doprowadzić każdy pojazd do odpowiedniego celu. Pionowo ustawiona plansza składa się z 3 części: punktów startu na górze, dróg łączących każdy punkt z każdym w środku, oraz punktów mety na dole. Po uruchomieniu gry w losowym punkcie startu pojawia się pojazd, sygnalizowany przez włączenie wyświetlacza znajdującego się w tym punkcie, z ustalonym punktem mety (wskazywany przez wyświetlacz). Gracz ma do dyspozycji przełącznik przy każdym skrzyżowaniu ścieżek prowadzących do mety, zmieniający miejsce w które skręci pojazd przejeżdżając przez rozjazd. Zadaniem jest takie ułożenie drogi, aby pojazd dotarł do odpowiedniego punktu końcowego. Za doprowadzenie do dobrego celu naliczane są punkty. Wraz z upływem czasu skraca się odstęp między kolejnymi generowanymi pojazdami, aż do osiągnięcia minimalnej wartości. Gra kończy się w momencie gdy któryś z pojazdów trafi do nieodpowiedniego punktu końcowego.

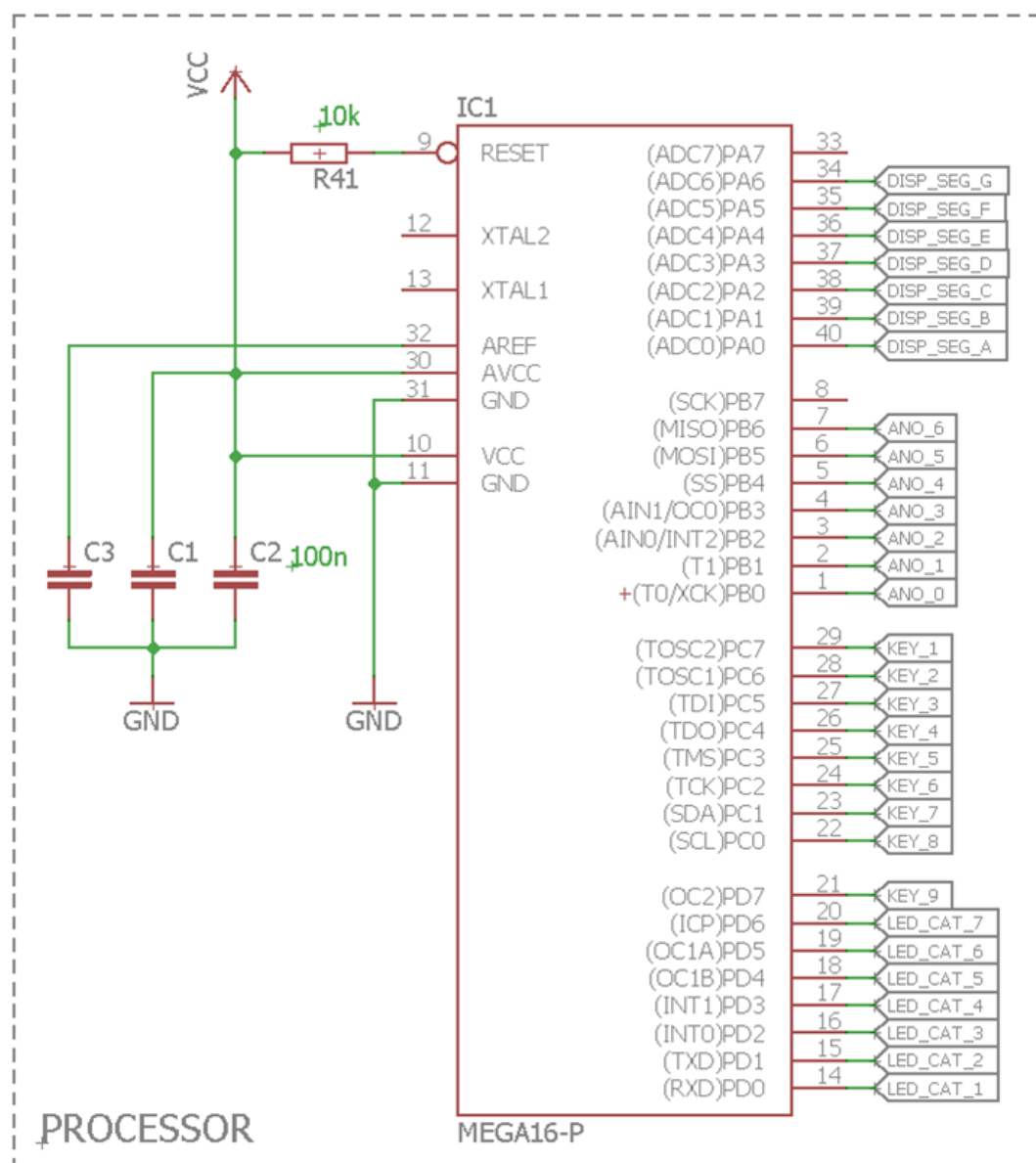
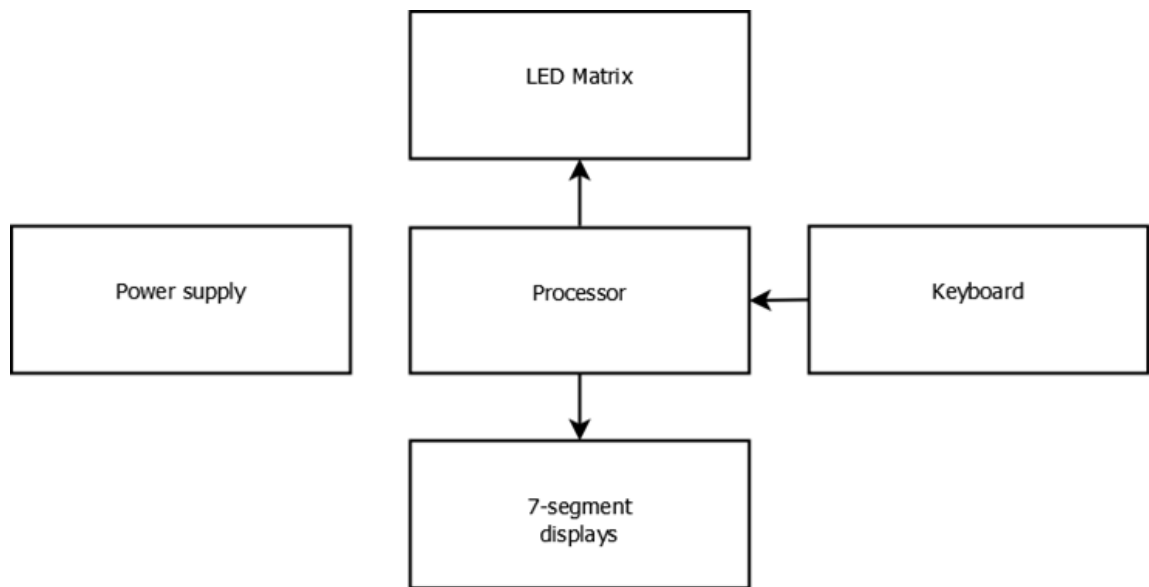
2. Analiza zadania

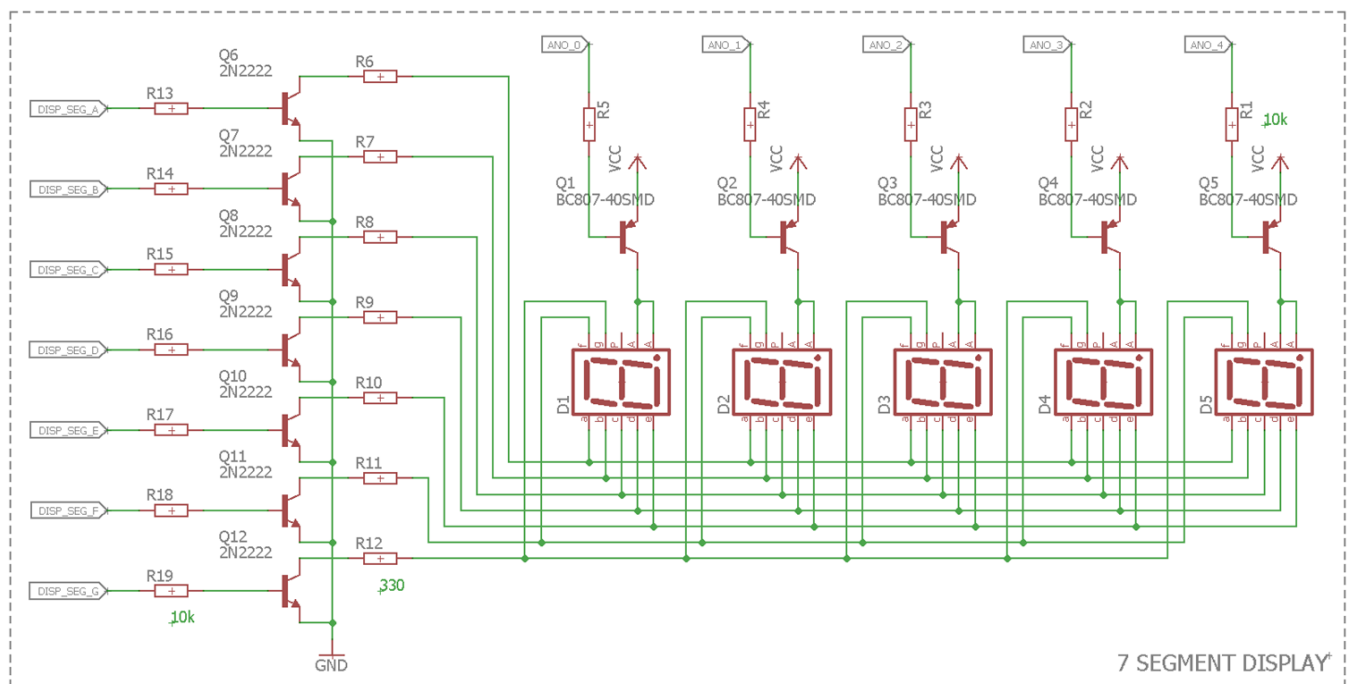
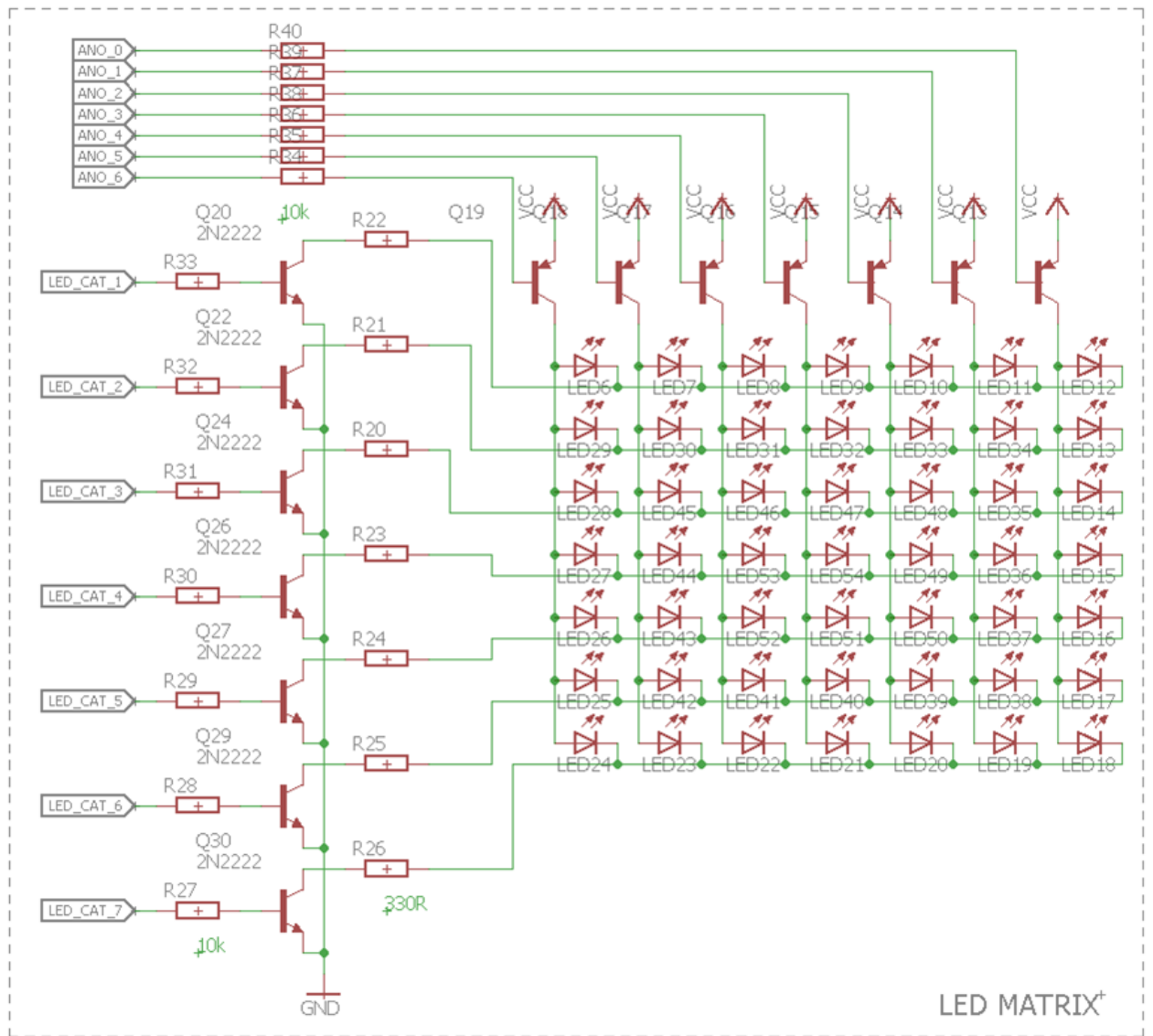
Część urządzenia odpowiedzialna za interakcję z użytkownikiem składa się z przełączników i przycisku (sterowanie stanem gry - start/pauza/reset - oraz ustawienie kierunku skrętu za pomocą przełącznika) jako wejście, oraz wyświetlaczy siedmiosegmentowych (do wskazywania celu w punkcie startu oraz aktualnego wyniku punktowego) i diod LED (do wskazywania położenia pojazdów, diody są połączone w ścieżki odpowiadające trasom) jako wyjście. Za zarządzanie tymi elementami odpowiada procesor Atmega16 – wybór rodziny AVR uzasadniony jest ich niską ceną, łatwą dostępnością i niewielką ceną programatora, a także stosunkowo dużą ilością poradników w sieci (rzecz ważna o tyle, że była to moja pierwsza styczność z tworzeniem układu opartego na mikrokontrolerze zupełnie od zera). Kryterium wyboru konkretnego procesora była ilość portów wejścia/wyjścia (wykorzystane są wszystkie dostępne w procesorze) oraz dostępność wersji w obudowie przewlekanej (nie mając doświadczenia w lutowaniu uznałem, że elementy SMD będą zbyt małe abym dał radę je zlutować). Jako że nie potrzebowałem żadnych nietypowych funkcji ani dużej ilości pamięci, wybrałem podstawowy model spełniający te wymagania, czyli Atmegę 16.

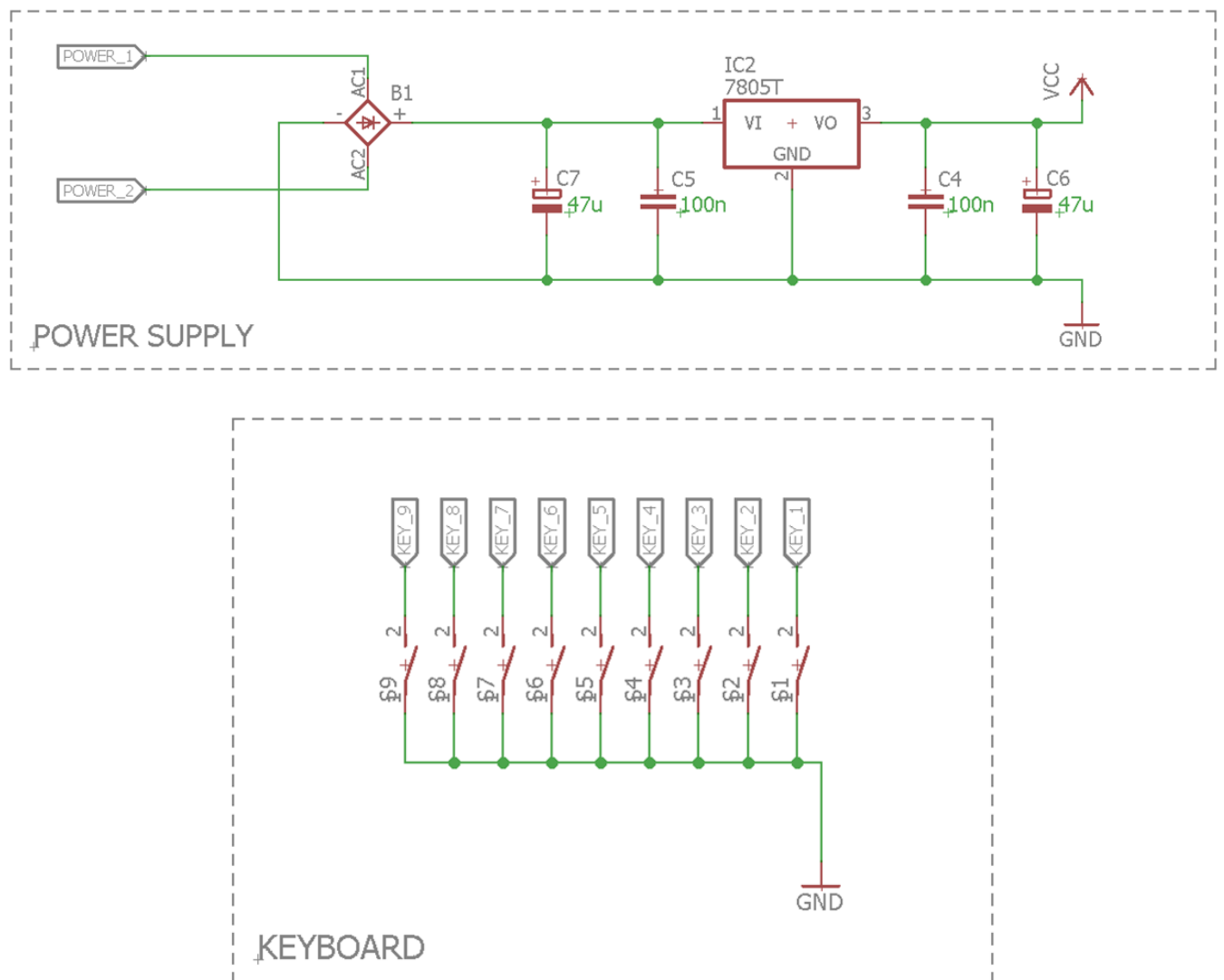
Samo urządzenie podzieliłem na dwie części: część użytkową (zawierającą wyświetlacze, diody i przyciski), oraz część „techniczną” (procesor wraz z pozostałymi elementami, na których użytkownik nie operuje bezpośrednio: oporniki, tranzystory itd.). Pierwotnym planem było zbudowanie części użytkowej na płytce uniwersalnej (diody muszą być ułożone regularnie, a z powodu ich dużej ilości brak konieczności wiercenia dziur był sporym ułatwieniem), a części technicznej na wytrawionym laminacie. W ostateczności jednak, część techniczną również zbudowałem na płytce uniwersalnej – zaważyła duża ilość dziur, które byłyby konieczne do wywiercenia, oraz brak doświadczenia w przenoszeniu schematu na laminat i wytrawianiu płytki, szczególnie biorąc pod uwagę jej potencjalnie duży rozmiar. Wobec tego, wszystkie elementy zostały przylutowane do płytek uniwersalnych i połączone za pomocą ścieżek zrobionych częściowo z cyny, a częściowo z wykorzystaniem przewodów typu kynar. Dodatkowo, obie płytki połączone są taśmą ATA 40-pinową, dzięki czemu możliwe jest umieszczenie części technicznej w oddaleniu od panelu gry (względy estetyczne).

3. Specyfikacja wewnętrzna urządzenia

3.1 Schemat blokowy i ideowy urządzenia





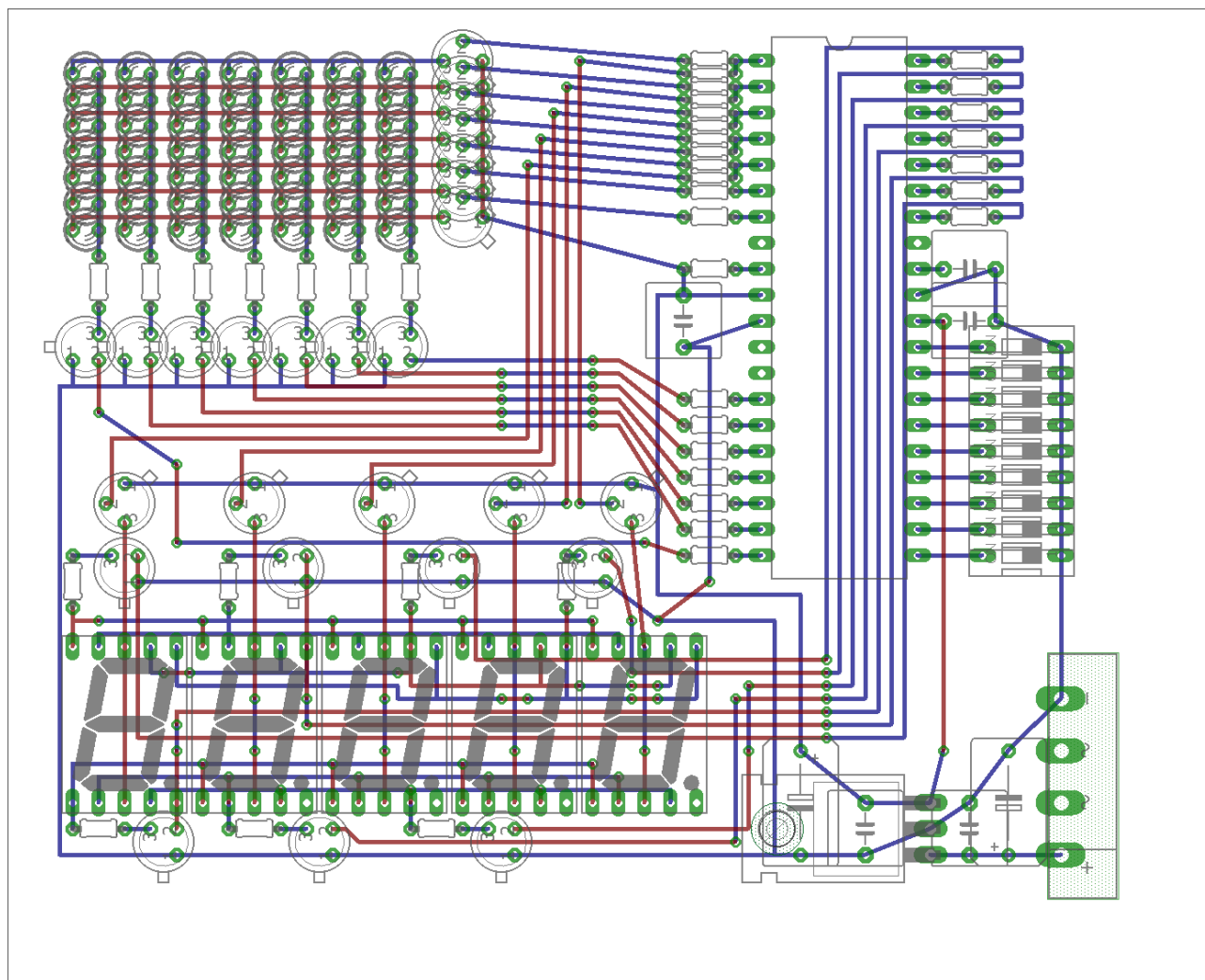


3.2 Opis funkcji poszczególnych bloków układu, szczegółowy opis działania ważniejszych elementów układu

- Procesor** – Atmega 16, wraz z połączeniem zasilania wymaganych do prawidłowego działania części cyfrowej oraz połączeniami z pozostałymi elementami układu. Oba piny GND zwarte do masy, oba piny zasilania (VCC i AVCC) podłączone do zasilania, wraz z kondensatorami filtrującymi 100nF umieszczonymi możliwie jak najbliżej nóżek procesora. Pin AREF jest podłączony przez kondensator 100nF do masy nieco na wyrost – filtracja na tym pinie umożliwia korzystanie z przetwornika analogowo cyfrowego, który co prawda nie jest wykorzystywany w projekcie, ale kondensatory i tak były kupione w pakiecie, więc nie widziałem przeciwwskazań przed jego zamontowaniem. Dalszy wzrost dokładności ADC można było uzyskać dokładając dławik między AVCC a zasilanie. Poza wymienionymi już elementami, podłączony jest również pin RESET przez rezystor 10k do zasilania: wymusza on jedynkę na tym pinie zabezpieczając przed nieoczekiwanym resetem urządzenia, który mógłby w przeciwnym wypadku wystąpić (Atmega ma wewnętrzny rezystor podciągający na tym pinie, jednak producent nie zapewnia jego działania w każdych warunkach i zaleca użycie zewnętrznego opornika). Piny XTAL1 i XTAL2 zostały niepodłączone, z powodu braku konieczności dokładnego pomiaru czasu za pomocą rezonatora kwarcowego.

- **Matryca LED** – 49 diod połączonych w matrycę tak, aby umożliwić ich multipleksowane sterowanie. Każda kolumna anod połączona jest z kolektorem tranzystora PNP (którego baza jest połączona z mikrokontrolerem, a emiter z zasilaniem), a każdy wiersz katod z kolektorem tranzystora NPN (którego baza jest połączona z mikrokontrolerem, a emiter z masą), aby diody nie pobierały zasilania z mikrokontrolera, a bezpośrednio z zasilania układu. Dodatkowo, aby zabezpieczyć diody przed spalaniem, między kolektorem a katodą umieszczone zostały rezystory 330R (w każdej chwili do jednej świecącej diody przydzielony jest jeden rezystor, równie dobrze mogłyby się one znajdować między kolektorem a anodą). Poza tym, między bazami tranzystorów a pinami mikrokontrolera znajdują się rezystory 10k, aby zabezpieczyć procesor przed przyjęciem zbyt dużego prądu.
- **Wyświetlacze siedmiosegmentowe** – schemat ich budowy jest identyczny jak w przypadku matrycy LED, z tą różnicą, że jako iż wykorzystuję 5 wyświetlaczy ze wspólną anodą, matryca składa się z 7 katod (ósma, odpowiedzialna za kropkę dziesiętną jest niewykorzystywana, a więc niepodłączona) i 5 anod. Wszystkie pozostałe elementy są takie same, jak w przypadku diod.
- **Zasilanie** – konwerter napięcia zasilania podawanego przez zasilacz sieciowy na napięcie 5V, zbudowany w oparciu o stabilizator napięcia 7805. Wejściem układu jest mostek prostowniczy, umożliwiający podłączenie w zasadzie dowolnego źródła zasilania (pamiętając o ograniczeniach stabilizatora – maksimum to 35V prądu stałego, ale wówczas stabilizator może się silnie grzać) i zabezpieczający przed odwrotnym połączeniem zasilania i masy. Poza tymi elementami, układ zasilania posiada umieszczone obok stabilizatora cztery kondensatory (dwa ceramiczne 100nF oraz dwa elektrolityczne 47uF), umieszczone parami (jeden elektrolityczny i jeden ceramiczny) przy wejściu oraz wyjściu układu.
- **Klawiatura** – zbudowana z 9 przełączników, które z jednej strony są połączone z procesorem, z drugiej z masą. Jako że Atmega posiada wbudowane rezystory podciągające (aktywowane ustawieniem odpowiednich wartości w rejestrze PORTX jeśli w rejestrze DDRX port jest ustawiony jako wejściowy), nie ma konieczności stosowania zewnętrznych oporników: przełącznik w pozycji zwartej łączy pin wejściowy procesora z masą zapewniając zero w rejestrze PINX, w pozycji rozłączonej natomiast wbudowane rezystory podciągające zapewniają jedynekę w rejestrze PINX.

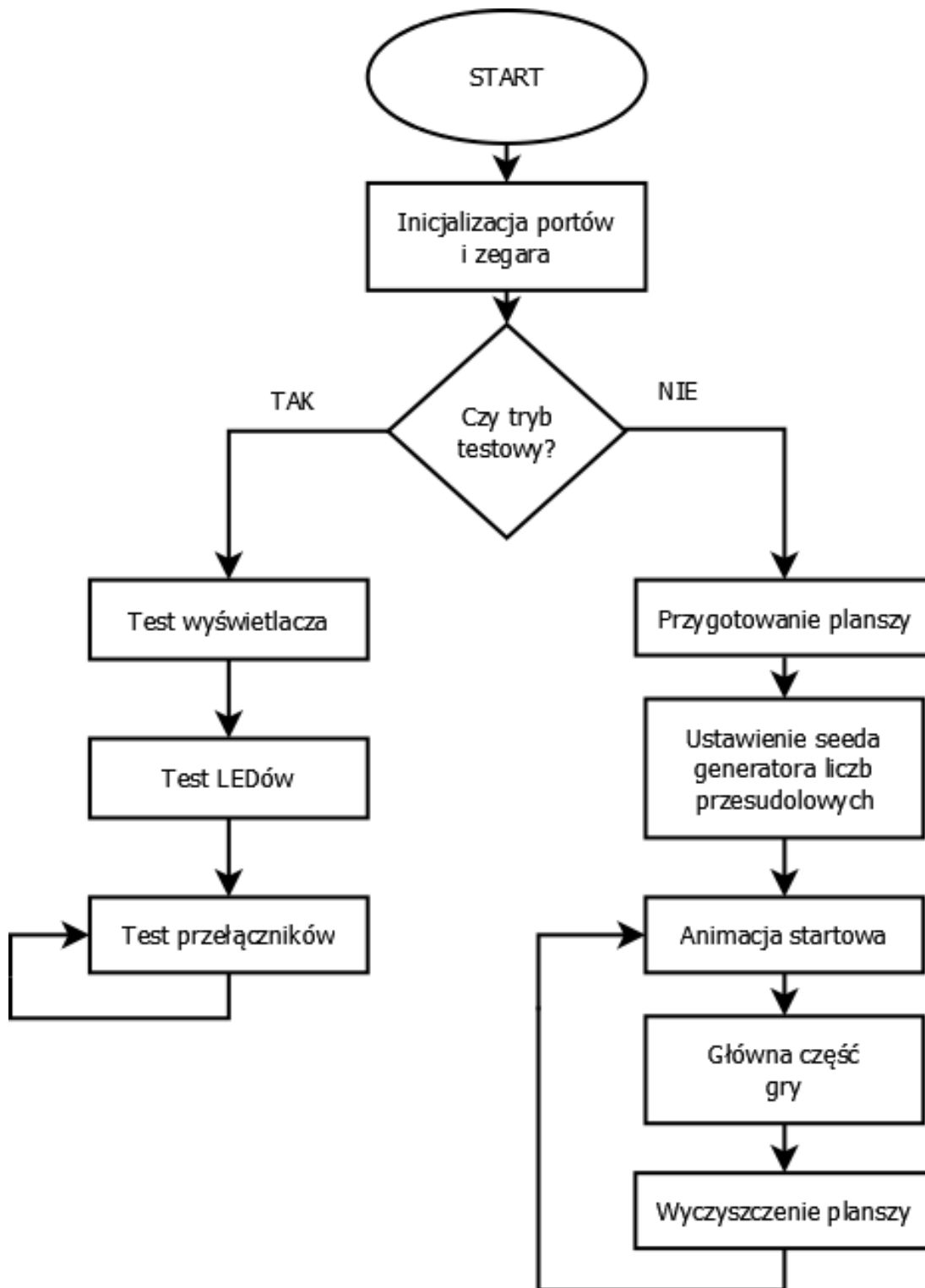
3.3 Schemat montażowy



3.4 Lista elementów

L.P.	Oznaczenie	Nazwa elementu	Wartość	Ilość	Uwagi	Cena (szt)	Cena (całk)
1	ATMEGA16A	Procesor	-	1	obudowa DIP	12,90 zł	12,90 zł
2	LEDxx	Dioda LED	-	49	zielona, 5mm	0,15 zł	7,35 zł
3	Dx	Wyświetlacz 7-segmentowy	-	5	wspólna anoda	0,90 zł	4,50 zł
4	7805	Stabilizator napięcia	-	1	L7805CV	0,90 zł	0,90 zł
5	KBU6M	Mostek prostowniczy	-	1		1,25 zł	1,25 zł
6	C6, C7	Kondensator	47uF	2	elektrolityczny	0,30 zł	0,60 zł
7	C1-C5	Kondensator	100nF	5	ceramiczny	0,10 zł	0,50 zł
8	Rxx	Rezystor	330Ω	14		0,06 zł	0,89 zł
9	Rxx	Rezystor	10kΩ	27		0,06 zł	1,71 zł
10	2N3906	Tranzystor PNP	-	12		0,20 zł	2,38 zł
11	BC547	Tranzystor NPN	-	14		0,20 zł	2,77 zł
12	Sx	Przełącznik dźwigniowy	-	9		1,50 zł	13,50 zł
13	-	Płytki uniwersalne	-	2		6,50 zł	13,00 zł
14	-	Złącze goldpin	-	1		1,20 zł	1,20 zł
Suma							63,44 zł

3.5 Algorytm oprogramowania urządzenia



3.6 Opis wszystkich zmiennych,

```
typedef struct
{
    uint8_t active; - zmienna wskazująca, czy samochód jest aktywny, czy też został już
usunięty
    uint8_t destination; - cel, do którego dany pojazd ma dotrzeć
    uint8_t position; - aktualne położenie pojazdu
} car; - struktura, w której zapisane są informacje o pojedynczym samochodzie
```



```

car cars[10]; - tablica przechowująca wszystkie samochody aktywne w grze
uint8_t failed_car = 0; - numer samochodu, który spowodował zakończenie gry
(wykorzystywane przy przegranej, do wskazania z jakiego powodu nastąpiła przegrana)
uint16_t counter = 0; - aktualna wartość licznika, zwiększana przy każdym wywołaniu
przerwania przez układ czasowy
uint8_t interrupt_flag = 0; - flaga ustawiana przy wywołaniu przerwania przez układ
czasowy
uint8_t points = 0; - aktualny wynik (lub wynik ostatnio rozegranej gry przy przegranej)
uint8_t switches = 0x00; - stan 8 przełączników na planszy, każdy bit odpowiada za jeden
przełącznik
uint8_t LED_TAB[7]; - stan każdego LEDa, jeden bit odpowiada jednej diodzie, najstarszy bit
nieużywany
uint8_t DISPLAY_TAB[5]; - stan każdego wyświetlacza
const uint8_t DISPLAY_VALUE[] = {
    0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x27, 0x7F, 0x6F, // 0 - 9
    0x77, // 10 A
    0x7C, // 11 b
    0x39, // 12 C
    0x5E, // 13 d
    0x79, // 14 E
    0x71, // 15 F
    0x00, // 16 OFF_DISPLAY
}; - tablica z wartościami wyświetlacza siedmiosegmentowego

```

3.7 Opis funkcji wszystkich procedur

```

void set_led(uint8_t id) - zapalenie LEDa o podanym ID, wykonywane przez ustawienie
jedyńki na odpowiadającym podanemu parametrowi bicie tablicy LED_TAB
void clear_led(uint8_t id) - zgaszenie LEDa o podanym ID, wykonywane przez ustawienie
zera na odpowiadającym podanemu parametrowi bicie tablicy LED_TAB
uint8_t check_led(uint8_t id) - sprawdzenie LEDa o podanym ID, 1 true jeśli jest
włączony oraz 0 jeśli jest wyłączony, wykonywane przez sprawdzenie wartości odpowiadającego
podanemu parametrowi bitu tablicy LED_TAB
void refresh_switches() - odświeżenie przełączników (zmiennej switches), poprzez
sprawdzenie ich stanu i ustawienie wszystkich bitów zmiennej switches: wartość 0 na bicie danego
przełącznika oznacza ustawienie przełącznika w pozycję do skrótu, 1 – skierowanie prosto
void refresh_output() - ustawienie na diodach i wyświetlaczu wartości z tablic oraz
odświeżenie, musi być wykonywana w pętli bez przerwy aby wyświetlacze działały (multipleksowanie),
funkcja wyświetla wartości umieszczone w tablicach LED_TAB oraz DISPLAY_VALUE, wysyłając je
bezpośrednio na piny mikrokontrolera
void int_to_display(uint16_t val, uint8_t start_seg, uint8_t end_seg)
- funkcja wyświetlająca liczbę naturalną na siedmiosegmentowcu, parametr value to wartość do
wyświetlenia, natomiast start_seg i end_seg to zakres, wskazujący które wyświetlacze mają wyświetlić
tę liczbę. Przypisuje odpowiadające podanym parametrom wartości do tablicy DISPLAY_VALUE
void error(uint8_t error_code) - funkcja wywoływana przy błędach - wyświetla ERR +
kod błędu, po czym przerywa działanie programu (wyłączając przerwania układu czasowego)
void set_interrupt_delay(uint16_t delay) - funkcja ustawia co ile czasu ma wywołać
się przerwanie TIMER1_COMPA, wartość musi być podzielna przez 4 z powodu użycia preskalera
ISR(TIMER1_COMPA_vect) - funkcja wywoływana przy nadejściu przerwania z układu
zegarowego, inkrementująca zmienną counter oraz ustawiająca flagę wystąpienia przerwania na 1
void test_board() - funkcja testująca działanie płytki, po kolei pozwalając sprawdzić
poprawność działania diod, wyświetlaczy oraz przełączników
void prepare_rand() - przygotowanie generatora liczb pseudolosowych przez ustalenie seeda do
funkcji rand() - seedem jest aktualna wartość zmiennej counter, zsumowana z wartością ustawioną na
przełącznikach

```

`void prepare_game()` - przygotowanie gry przed startem
`void startup_animation()` - wyświetlenie animacji startowej
`void proceed_cars()` - przesunięcie pojazdów jedno pole do przodu
`void create_new_car()` – utworzenie nowego pojazdu
`void play_game()` - główna pętla gry
`int main(void)` - funkcja `main` wywoływana przy starcie, mająca na celu inicjalizację portów, a następnie wejście w pętlę `while(true)` wywołującą po kolei funkcje `prepare_game()`, `startup_animation()` i `play_game()`.

3.8 Szczegółowy opis działania ważniejszych procedur

Najważniejszą procedurą w programie jest `play_game()`, będąca główną logiką gry. Jej zadaniem jest tworzenie nowych samochodów oraz przesuwanie już wcześniej utworzonych. Działanie uzależnione jest od flagi przerwania, wywoływanej co określony czas, malejący wraz ze zdobywaniem punktów (w celu ciągłego wzrostu poziomu trudności). Gdy nadejdzie przerwanie (na starcie programu wywoływane jest co 2 sekundy), procedura przesuwa wszystkie pojazdy o jedno pole w dół (wywołując procedurę `proceed_cars()`), a co cztery przerwania tworzy dodatkowy samochód (wywołując `create_new_car()`).

Za każdym razem, bezpośrednio przed wywołaniem procedury `play_game()` wyświetlana jest animacja startowa za pomocą wywołania `startup_animation()`. Ma ona na celu poinformować gracza o rozpoczęciu rozgrywki, polega na kolejnym zapaleniu i zgaszeniu wszystkich diod. Po zgaszeniu ostatniego rzędu diod gra się rozpoczyna.

Procedura `proceed_cars()` przegląda tablicę wszystkich samochodów, i dla każdego aktywnego (takiego który nie dotarł jeszcze do celu) wylicza jego nową pozycję. Domyślnie jest to przesunięcie pojazdu jedno pole w dół (w tym „spadek” pojazdu z wyświetlacza będącego polem startowym na pierwszy rząd), z ewentualnym uwzględnieniem skrętu (stan przełączników jest sprawdzany na starcie tej procedury). Dodatkowo, jeśli pojazd znajduje się w drugim rzędzie, wygaszany jest wyświetlacz wskazujący jego cel, a jeśli w ostatnim – sprawdzana jest poprawność celu (czy samochód trafił w miejsce wskazywane przez wyświetlacz). Jeśli tak, to naliczane są punkty, jeśli nie – gra kończy się.

Procedura `create_new_car()` tworzy nowy samochód. Uruchamiana jest co 4 przerwania nadchodzące z układu zegarowego. Funkcja umieszcza utworzony samochód w pierwszym wolnym miejscu tablicy (takim, którego stan jest oznaczony jako nieaktywny), losuje dla niego pozycję startową i cel, oraz uruchamia odpowiedni wyświetlacz siedmiosegmentowy. Przy kolejnym przerwaniu i wywołaniu funkcji `proceed_cars()`, samochód będzie już widziany jako aktywny i normalnie przesunięty do przodu.

Procedura `prepare_game()` odpowiada za przygotowanie gry do startu, zarówno przy uruchomieniu urządzenia, jak i po przegranej lub reset. Kontroluje ona stan przełącznika i przycisku na górze urządzenia, a jeśli zajdą odpowiednie warunki (dźwignia zostanie przesunięta z pozycji pause na pozycję start, a ilość punktów jest równa zero) uruchamia grę. W przypadku przegranej, dodatkowo do czasu zresetowania wyświetla ona pojazd który spowodował koniec gry oraz jego prawidłowy cel.

4. Specyfikacja zewnętrzna urządzenia

4.1 Opis funkcji elementów sterujących urządzeniem

Elementy sterujące to przełącznik start/pause, przycisk reset oraz 8 przełączników przy diodach LED służących do wyboru drogi. Przełączniki przy diodach służą w trakcie gry do wyboru miejsca, w które trafić ma pojazd po przejechaniu przez skrzyżowanie. Ustawienie dźwigni bezpośrednio wskazuje kierunek, w jaki przemieści się samochód.

Przełącznik start/pause oraz reset wykorzystywane są w trakcie gry oraz przed jej startem. W trakcie gry, gdy dźwignia znajduje się w pozycji START, przycisk reset jest nieaktywny. Przesunięcie

dźwigni w stan PAUSE powoduje wstrzymanie gry – można wówczas albo ją wznowić (przesuwając dźwignię z powrotem w górę), albo zresetować (wciskając przycisk reset) – w tym drugim przypadku nastąpi wyjście z pętli gry i przejście do stanu „przed startem”.

W stanie przygotowania do startu, gdy dźwignia jest w położeniu górnym, przycisk resetu jest nieaktywny, a na planszy wyświetlany jest albo powód przegranej, albo pusta plansza (zależnie od tego, czy dźwignia jest w pozycji startu z powodu przegranej, czy tak była położona w momencie podłączenia urządzenia do prądu). Przesunięcie dźwigni w stan PAUSE odblokowanie przycisku RESET, którego wciśnięcie wyczyści wynik punktowy i przywróci planszę do stanu początkowego. Jeśli wynik punktowy jest równy zero, przesunięcie dźwigni w górę spowoduje uruchomienie gry i wywołanie animacji startowej. Jeśli różny od zera, przesunięcie dźwigni w górę jest bezsensowne, gdyż spowoduje ponowne zablokowanie niebieskiego przycisku resetu i konieczność przesunięcia dźwigni w dół.

4.2 Opis funkcji elementów wykonawczych (wyświetlacz, diody LED, przekaźniki)

Dwa siedmiosegmentowe wyświetlacze położone obok niebieskiego przycisku to wskaźnik punktów – pokazywany jest na nich aktualny wynik punktowy gracza.

Trzy wyświetlacze poniżej, to pole startowe – na nich pojawiają się pojazdy, co sygnalizowane jest przez zapalenie wyświetlacza i wskazanie na nim za pomocą litery od A do E celu danego pojazdu.

49 LEDów położonych pod wyświetlaczami to plansza gry, pokazująca aktualne położenie pojazdów na mapie. Dioda zapalona oznacza samochód w danym punkcie, zgaszona – jego brak.

4.3 Skrócona instrukcja obsługi urządzenia

Po włożeniu wtyczki do gniazdka urządzenie jest w stanie przygotowania do gry. Warunkiem startu jest przesunięcie dźwigni start/pause z pozycji dolnej do góry, jeśli wynik na wyświetlaczu punktowym jest równy zero (aby go wyzerować, należy wcisnąć przycisk reset). Po wyświetleniu animacji startowej na jednym z wyświetlaczy pojawi się litera oznaczająca cel. Należy ułożyć dźwignie w taki sposób, aby pojazd dojechał do przypisanego mu celu. Gdy pojazd będzie w drugim rzędzie, wyświetlacz zostanie zgaszony, a na polu startu pojawi się nowy samochodzik. Gra kończy się w momencie, w którym jeden z pojazdów trafi do złego punktu końcowego. Wówczas przejdzie ona ponownie w stan przygotowania do gry.

4.4 Opis złączy

Na każdej z dwóch płytek znajduje się 40-pinowe złącze, które należy połączyć taśmą ATA, aby część „techniczna” była połączona z częścią zawierającą elementy sterujące i wykonawcze. Należy się upewnić, że wtyczka włożona jest w dobrą stronę (najprościej sprawdzić to różowym oznaczeniem na taśmie – w obu złączach musi być ono w tym samym miejscu). Schemat złącza:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	LC7(20)	LC6(19)	LC5(18)	LC4(17)	LC3(16)	LC2(15)	LC1(14)	KEY9(21)	GND	GND	GND	GND	KEY8(22)	LED ANODA 0..6 (1-7)						
2	DA0(1)	DA1(2)	DA2(3)	DA3(4)	DA4(5)	KEY10(33)	KEY2(28)	KEY1(29)	KEY7(23)	KEY5(25)	KEY4(26)	KEY6(24)	KEY3(27)	DSA(40)	DSB(39)	DSC(38)	DSD(37)	DSE(36)	DEF(35)	DSG(34)

DA - DISPLAY ANODA
DS - DISPLAY SEGMENT
KEY - KEYBOARD
LC - LED KATODA
LA - LED ANODA
GND - MASA

Nazwy pinów odpowiadają nazwom ze schematu ideowego, natomiast wartość w nawiasie to konkretny pin procesora, z którym połączona jest dana nazwa.

Dodatkowo, na płytce technicznej znajduje się sześciopinowy port, służący do podłączenia programatora – konkretne linie są opisane na porcie, od lewej strony jest to MOSI, MISO, SCK, RESET, VCC, GND. Dodatkowo, port ten może uruchomić funkcję testowania płytki: aby przejść w ten

tryb, należy zewrzeć pin SCK do masy, a następnie zresetować urządzenie (czy to przez wyjęcie i włożenie wtyczki do gniazdka, czy to przez zwarcie pinu RESET do masy).

5. Opis montażu układu, sposobu uruchamiania oraz testowania

Pierwsza wersja układu została zmontowana na płytce stykowej – umieściłem tam wszystkie wyświetlacze, przełączniki, oraz matrycę 4x4 diody (z racji ograniczonej ilości miejsca na płytce nie mogłem tam umieścić wszystkich diod). Prototyp układu działał prawidłowo, wobec czego przeszedłem do montażu elementów płytki „użytkowej” na płytce uniwersalnej (wybór uzasadniony brakiem konieczności wiercenia otworów pod każdą diodę). Po przylutowaniu poszczególnych elementów przeszedłem do ich łączenia w matryce odpowiadające schematowi ideowemu. Początkowo było to dość proste, wszystkie elementy łączyłem za pomocą cynowych ścieżek, jednak szybko przyszła konieczność przycinania się ścieżek. Rozwiązałem to najpierw używając miedzianego kabla ze skrętki sieciowej, a wraz ze wzrostem ich ilości na płytce - przewodu typu kynar. Pajęczyna przewodów w niektórych miejscach jest przerażająca, jednak miałem możliwości ich innego ułożenia, z powodu ściśle określonego ułożenia elementów na płytce (położenie każdej diody i przełącznika musiało być takie jak na początku założyłem wymyślając wygląd planszy, mimo że utrudniało to łączenie). Niemożliwe było również przetestowanie działania tych połączeń (w zakresie większym niż sprawdzanie zwarcia poszczególnych linii) do momentu zmontowania płytki z procesorem i pozostałymi elementami, a z powodu opóźnień w dostawie drugiej płytki nie miałem możliwości ich równoczesnego wykonywania.

Wykonanie drugiej płytki („technicznej”) było już zdecydowanie łatwiejsze. Po pierwsze, miałem już większe doświadczenie w lutowaniu, nabyte przy pierwszej płytce, po drugie – mogłem ułożyć elementy w zasadzie w dowolny sposób, a ponieważ ta część nie ma żadnego znaczenia dla użytkownika końcowego, mogłem przełożyć wygodę montażu nad wygląd końcowy. Skutki tego są widoczne porównując spodnie części płytek – techniczna jest ułożona zdecydowanie bardziej symetrycznie, ma też więcej cynowych ścieżek niż zewnętrznych przewodów.

Do połączenia obu płytek zdecydowałem się wykorzystać 40-pinową taśmę ATA, gdyż właśnie taka ilość wystarczała do przekazania wszystkich potrzebnych sygnałów między płytkami. Uznałem, że zastosowanie po prostu drutów w takiej dużej ilości będzie nieco niewygodne, większy bałagan, a ponadto uniemożliwi późniejsze oddzielenie dwóch części urządzenia od siebie. Po połączeniu mogłem wreszcie przystąpić do uruchomienia układu i przetestowania jego działania. Testowanie polegało na uruchomieniu tego samego kodu, który był wcześniej napisany na potrzeby płytki stykowej (jego zmodyfikowana wersja nadal jest zachowana, i występuje pod nazwą trybu testowego) i sprawdzeniu poprawności działania poszczególnych układów. Gdy wszystkie błędy zostały naprawione (kilka linii było zwartych, kilka nie łączyło) przystąpiłem do pisania głównej części aplikacji (logiki gry), a następnie wykonywanego w pętli: sprawdzania czy wszystko działa zgodnie z założeniami, modyfikowania kodu, programowania płytki, sprawdzania czy wszystko działa zgodnie z założeniami, dodawania nowej funkcjonalności itd. Testowanie polegało więc na wielokrotnym graniu, i sprawdzaniu czy gra działa poprawnie. Błędy wykryte i poprawione to chociażby nieprawidłowe gaszenie wyświetlacza (jeśli samochód został dwa razy pod rząd wygenerowany w tym samym miejscu, cel dla drugiego auta był wyświetlany bardzo krótko – rozwiązałem to zmieniając czas wyświetlania celu) czy problem z przechodzeniem samochodu na kolejną pozycję (złe policzenie diod, każda z nich jest reprezentowana przez kolejny numer od 1 do 49, przez pomyłkę w jednym miejscu wpisałem zły numer kolejnej diody). O wiele trudniejsze było napisanie funkcji wejścia/wyjścia, interpretujących przełączniki i obsługujących wyświetlacze i diody. Zajęło to więcej czasu, dlatego że diody i przełączniki są tak podłączone, aby uprościć ich montaż, więc niektóre z przełączników dają „0” a inne „1” na wejściu, mimo położenia ich w tej samej pozycji. Podobnie sprawa ma się z diodami – współrzędne matrycy nie odpowiadają w żaden sposób ułożeniu diod na płytce, więc każdej z nich musiałem niezależnie sprawdzić współrzędne i zapisać je w kodzie. W momencie gdy uznałem, że program działa w sposób poprawny, a funkcjonalność jest satysfakcjonująca i spełnia założenia projektu, do procesora wgrałem finalną wersję kodu, oznaczoną jako 1.0

6. Wnioski i uwagi z przebiegu pracy

6.1 Wnioski - część sprzętowa

W przypadku obu płytek, zdecydowanie wykonanie urządzenia ułatwiłoby skorzystanie z gotowej, zamówionej płytki. Musząc wybierać między wierceniem ogromnej ilości otworów a tworzeniem dużej ilości ścieżek z cyny i drutu, biorąc dodatkowo pod uwagę rozmiary płytki i brak doświadczenia w wytrawianiu, zdecydowałem się na tę drugą opcję, czyli skorzystanie z płytki uniwersalnej. O ile zamontowanie w niej elementów było bardzo proste, o tyle ich odpowiednie połączenie sprawiało większe trudności. Tworzenie cynowych ścieżek obok siebie tak żeby się ze sobą nie stykały, odpowiednie przylutowanie kabla do elementu tak aby łączył bez żadnych zakłóceń jest dość czasochłonne i często powoduje błędy, wymagało też dość dużego nakładu pracy. Efekt końcowy jest daleki od ideału, aczkolwiek wszystkie połączenia działają i płytka jest w pełni funkcjonalna, a to jest moim zdaniem najważniejsze. Testowanie połączeń również nie było bardzo uciążliwe – wystarczyło uruchomić urządzenie w trybie testowym (napisanym już wcześniej przy tworzeniu prototypu na płytce stykowej) i sprawdzić z którymi połączeniami jest problem, a następnie je poprawić.

Występujące w trakcie montażu błędy spowodowane były brakiem doświadczenia w precyzyjnym lutowaniu małych elementów, polegały głównie na niestykających połączeniach lub zwarcu dwóch sąsiadujących ze sobą kabli. Wszystkie tego typu błędy dało się łatwo wykryć dzięki użyciu multimetru z brzęczykiem, sygnalizującym zwarcie między dwiema sondami. Testowanie układu bez dostępnego miernika byłoby mocno utrudnione.

6.2 Wnioski - część programu

Wbrew wcześniejszym obawom, nie doskwierał mi wcale brak debuggera – w kodzie nie zarządzam bezpośrednio pamięcią (malloc / free) a korzystam w większości ze zmiennych globalnych, co zmniejsza ryzyko powstania błędów. Dużym ułatwieniem pracy okazało się napisanie kodu odpowiedzialnego za wejście i wyjście układu już wcześniej. Przy wykorzystaniu już wcześniej stworzonych funkcji, sprawdzających i ustalających stan każdej diody i przełącznika, samo napisanie gry okazało się bardzo proste – każda dioda, wyświetlacz i przycisk ma odpowiadającą za niego zmienną przechowującą stan, i to na nich operuje logika gry, a przekazywaniem tych zmiennych na odpowiednie piny procesora zajmuje się już funkcja odświeżająca. Dzięki temu ewentualne modyfikacje programu gry będą stosunkowo proste – nie trzeba pamiętać, która dioda jest podłączona do którego portu i w którym wierszu matrycy się znajduje, zamiast tego wystarczy użyć funkcji zapalającej diodę o określonym identyfikatorze. Podobnie sprawa ma się z przełącznikami – nie trzeba wiedzieć który przełącznik jest jak podłączony, bo stan wszystkich jest przechowywany w jednej zmiennej, pod postacią zinterpretowaną do wiadomości czy wskazuje na wprost, czy nakazuje skrócić.

Dodatkowym ułatwieniem było odpowiednie skonfigurowanie środowiska Atmel Studio do współpracy z programatorem USB Asp. Dzięki dodaniu do paska narzędziowego trzech przycisków uruchamiających w tle program AVR Dude, odpowiedzialnych za wgranie programu do procesora, wyczyszczenie wgranego programu oraz zresetowanie układu, testowanie poprawności działania aplikacji było wygodne i intuicyjne (nie wymagało ręcznego wpisywania komendy do zewnętrznego programu).

6.3 Wnioski końcowe

Wbrew początkowym obawom, wykonanie projektu okazało się być prostsze niż początkowo zakładałem. Główną trudnością do pokonania w trakcie tworzenia projektu był fakt, że to była moja pierwsza styczność z projektowaniem i wykonywaniem tego typu układów. Niepewność od czego zacząć i co zrobić żeby nie uszkodzić jakiegoś elementu była głównym czynnikiem spowalniającym pracę. Gdy jednak udało już mi się zacząć jakiś konkretny etap i pokonać początkowe trudności, dalsza praca szła już zdecydowanie prościej, a zobaczenie końcowego efektu pod postacią w pełni działającego układu dało ogromną satysfakcję.