AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA
W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

KATEDRA AUTOMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa inżynierska

Testowanie kamery zaawansowanego systemu wspomagania kierowcy za pomocą metodologii Hardware-in-the-Loop z zamkniętym sprzężeniem zwrotnym

Testing ADAS camera with closed-loop HiL methodology

Autor: Paweł Jemioło

Kierunek studiów: Automatyka i Robotyka

Opiekun pracy: prof. dr hab. Marek Gorgoń

Kraków, 2017

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję profesorowi Markowi Gorgoniowi za możliwość napisania tej pracy, zaangażowanie i wyrozumiałość, doktorowi Mateuszowi Komorkiewiczowi za przekazaną wiedzę, motywację i wszelką inną pomoc oraz mojej rodzinie za wsparcie w każdym momencie.

Streszczenie

Niniejsza praca ma na celu stworzenie środowiska testowego dla nowoczesnych systemów wspomagania kierowcy opartych o dane z kamery. W dokumencie objaśniono podstawowe pojęcia charakterystyczne dla dziedziny oraz przedstawia ogólny sposób prowadzenia eksperymentów.

Istotnym elementem pracy jest opis metody zamknięcia pętli Hardware-in-the-Loop, tak aby można było uzyskać właściwe sprzężenie zwrotne i móc stwierdzić responsywność testowanego systemu wspomagania w czasie rzeczywistym w warunkach laboratoryjnych. Do uzyskania tego stanu rzeczy stosuje się metody pozwalające dostarczyć do kamery dane w taki sposób, aby stworzyć wrażenie, że porusza się wraz z samochodem po drodze.

Wyrażenia kluczowe: systemy bezpieczeństwa samochodowego, czynne bezpieczeństwo samochodowe, ADAS, kamera, symulacja, testowanie, Hardware-in-the-Loop, HiL, dSPACE, CarMaker, FPGA.

Abstract

The point of this work is to create testing environment for advanced driving assistance systems based on data from a camera. In the document, it has been explained basic terms characteristic for automotive systems and presented the way of conducting experiments.

An important element of this work is a description of the method how to close Hardware-in-the-Loop system, so that one is able to gain right feedback and to conclude if tested system responds in real time. Specified information is applied to camera to influence it.

Keyword phrases: automobile safety systems, active safety, ADAS, camera, simulation, testing, Hardware-in-the-Loop, HiL, dSPACE, CarMaker, FPGA.

Spis treści

Wstęp

W poniższym rozdziale przedstawiono krótkie wprowadzenie do zagadnienia, następnie opisano najważniejsze cele i założenia przyświecające powstawaniu projektu oraz omówiono zawartość i układ pozostałych części pracy.

Wprowadzenie

Współczesne samochody stają się coraz bardziej zaawansowane technologicznie. Systemy, które stosuje się w pojazdach można podzielić na dwie kategorie: bierne i czynne. Pierwsze z nich mają za zadanie ograniczanie skutków już zaistniałej kolizji, przy czym skupiają się przede wszystkim na ochronie osób wewnątrz samochodu. Zadaniem tych drugich jest zmniejszenie do minimum prawdopodobieństwa wystąpienia wypadku oraz zapewnienie bezpieczeństwa wszystkim uczestnikom ruchu drogowego.

Nie dziwi zatem fakt, że to właśnie systemy aktywne są jednymi z najintensywniej rozwijanych komponentów pojazdów. Dzięki danym zbieranym przez przyrządy pomiarowe możliwe jest informowanie kierowcy o potencjalnych zagrożeniach lub przejęcie kontroli nad samochodem w sytuacjach ekstremalnych (np. w celu wykonania hamowania awaryjnego).

Istotny udział w gromadzeniu informacji o otaczającym świecie mają urządzenia takie jak LIDAR (ang. *Light Detection and Ranging*), radar oraz kamery. Wizja odgrywa ogromną rolę we współczesnym przemyśle motoryzacyjnym. Ze względu na możliwość rozpoznawania przeszkód – pieszych, rowerów, czy ciężarówek i odczytywanie limitów prędkości, kamera jest praktycznie niezbędnym elementem nowoczesnych samochodów. Implementacja algorytmów wizyjnych pociąga za sobą stosowanie nowoczesnych układów takich jak układy reprogramowalne, czy szybkie procesory.

ADAS (ang. *Advanced Driver Assistance Systems* – zaawansowane systemy wspomagania kierowcy), czyli komponenty czynne, w dużym stopniu przyczyniają się do poprawy komfortu jazdy. Doskonałymi przykładami może być automatyczne parkowanie, jazda po zakorkowanej

ulicy w mieście, czy rozpoznawanie znaków i ograniczenie maksymalnej prędkości na autostradach i drogach szybkiego ruchu.

Systemy bezpieczeństwa czynnego w prosty sposób mogą być implementowane w samochodach pół-autonomicznych i autonomicznych, co stanowi kolejną korzyść z ich rozwijania. Poziom zaawansowania implementowanych układów wspomagania kierowcy jest bardzo wysoki i stale rośnie. Można więc odnieść wrażenie, że już niedługo pojazdy będą poruszały się po drogach samodzielnie, nie wymagając żadnej dodatkowej interwencji.

Każda innowacja niesie ze sobą jednak spore ryzyko. Tym bardziej taka, w której drobny błąd może kosztować ludzkie życie. Producenci poświęcają więc ogromne nakłady na testowanie opracowanych przez siebie samochodów. Obrane metody sprawdzania aut są bardzo zróżnicowane, często wykraczające poza niezbędne minimum określone przez normy międzynarodowe. Doskonałym przykładem są tutaj Ford i Toyota, które swoje pojazdy testują w specjalnie spreparowanych miastach, symulujących prawdziwe życie. Nieco inne podejście prezentuje Google, które sprawdza samochody firmy Lexus, jeżdżąc po drogach stanu Kalifornia. Samochody poruszają się samodzielnie po drogach użytkowanych publicznie, są jednak pod stałym nadzorem specjalnie przeszkolonych inżynierów. Tesla poszła o jeszcze jeden krok dalej – pojazdy tej marki są w trakcie nieustannych testów. Dane pochodzące od zwykłych użytkowników są zbierane przez cały czas i następnie analizowane. Na ich podstawie opracowywane są kolejne aktualizacje oprogramowania.

Przed testami całego samochodu, odbywa się sprawdzanie poszczególnych jego komponentów – między innymi tych, które tworzą układ bezpieczeństwa czynnego. Zagadnienie to jest niezwykle istotne, gdyż w sytuacjach wyjątkowych ADAS bardzo często dysponują możliwością przejęcia kontroli nad całym pojazdem.

W niniejszej pracy poruszony został temat testowania technologii wizyjnych używanych w samochodach za pomocą metody HiL (ang. *Hardware-in-the-Loop*) z zamkniętym sprzężeniem zwrotnym. Metoda ta wydaje się być bardzo ciekawym podejściem do sposobu ewaluacji systemów aktywnego bezpieczeństwa ze względu na możliwość przeprowadzenia procesu kontrolnego układów bezpośrednio w laboratorium. W perspektywie przynosi to ogromne zyski czasowe i pieniężne. Stworzenie takiego systemu jest jednak zadaniem złożonym, a implementacja nie może być przeprowadzona w jednej chwili. Biorąc pod uwagę, że kamery coraz częściej stanowią podstawę pewnych układów samochodowych, korzyści finansowe wynikające z wdrożenia metodologii HiL są dla firm bardzo istotne. Coraz więcej producentów samochodów próbuje stworzyć odpowiednie systemy testowe dla układów wizyjnych – i nie tylko – oparte o wspomnianą metodę.

Cele pracy

Celem pracy jest stworzenie systemu, który umożliwi przetestowanie kamery zaawansowanego systemu wspomagania kierowcy zgodnie z metodologią *Hardware-in-the-Loop* z zamkniętym sprzężeniem zwrotnym.

Założenia projektu

1. Przegląd aktualnej literatury oraz rozwiązań związanych z tematyką testowania systemów wizyjnych z wykorzystaniem metodologii *Hardware-in-the-Loop*.
2. Stworzenie modelu w środowisku MATLAB oraz Simulink, generacja kodu i wgranie na platformę dSPACE 1006.
3. Stworzenie aplikacji do komunikacji z kamerą, wykorzystując magistralę CAN oraz protokół sieciowy UDP.
4. Stworzenie wizualizacji drogi z wykorzystaniem programu CarMaker, która będzie przesyłana do kamery.
5. Stworzenie pełnego środowiska testowego umożliwiającego przetestowanie w kamerze funkcjonalności automatycznego hamowania (AEB), zgodnie z normami NCAP i wykonanie testów.
6. Wykorzystanie układu FPGA w celu zastąpienia sensora wizyjnego w kamerze, tak aby możliwe było wstrzykiwanie generowanego obrazu.
7. Praca pisana jest na podstawie umowy o współpracę z firmą Delphi Poland S.A.

Struktura pracy

Praca składa się z czterech kolejno ponumerowanych rozdziałów, podzielonych na odpowiednie mniejsze sekcje. Każda z sekcji opisuje inny problem teoretyczny, bądź praktyczny.

W rozdziale pierwszym przedstawiono aktualny stan wiedzy na temat sposobów testowania samochodowych systemów wizyjnych. Szczególny nacisk położono na zaprezentowanie publikacji dotyczących metodologii *Hardware-in-the-Loop*. Zebrano problemy związane z zagadnieniem. Zestawiono ze sobą różne modele HiL oraz przeanalizowano inne publikacje powiązane z zagadnieniami poruszanymi w niniejszej pracy.

Drugi rozdział zawiera sposób implementacji docelowego systemu testowego. Na początku szczegółowo omówiono wykorzystany sprzęt. Opis użytych elementów został opracowany na podstawie źródeł danych takich jak karty katalogowe produktów. Zamieszczono też informacje o tym, jakie role w projekcie pełniły poszczególne platformy. W kolejnych częściach tej sekcji omówiono etapy powstawania projektu inżynierskiego oraz zaprezentowano schematy stworzonego środowiska.

W rozdziale trzecim zwrócono uwagę na elementy potrzebne do przeprowadzenia ewaluacji stworzonego środowiska. W dalszej części przeanalizowano zebrane dane testowe, porównano je z dostępnymi danymi rzeczywistymi oraz sporządzono tabele zestawiające informacje o testowanym systemie.

Ostatnia sekcja to podsumowanie całej pracy. Wypisane zostały zrealizowane elementy projektu, zwrócono uwagę na mocne i słabe strony przedstawionego rozwiązania oraz wskazano możliwy kierunek dalszych prac.

Na końcu pracy znajduje się spis użytej literatury. Za nim umieszczono dodatek A, zawierający spis zawartości dołączonej płyty CD oraz dodatek B, czyli wykaz elementów stworzonego systemu.

1. Sposoby testowania samochodowych systemów wizyjnych

Rozpoznawanie obrazu odgrywa współcześnie jedną z najważniejszych ról w czynnych systemach bezpieczeństwa samochodowego. Jest jednym z głównych źródeł informacji na temat otaczającego świata (rys. 1.1) obok urządzeń takich jak radar, czy LIDAR (ang. *Light Detection and Ranging*).

Rysunek 1.1. *Rozpoznawanie samochodów i pozostałych elemntów ruchu drogowego przy pomocy kamery ADAS*, źródło: [7]

Typowy proces testowania układów wspomagania kierowcy można podzielić na kilka etapów, przy czym każdy kolejny etap ma większe znaczenie od etapu poprzedzającego [3]:

1. Wykonanie symulacji MiL (ang. *Model-in-the-Loop*). Na tym etapie tworzony jest model docelowego systemu, najczęściej przy pomocy języków programowania takich jak C/C++/Python/Matlab. Następnie napisany kod, uruchamiany jest na komputerze i poddawany specyficznym scenariuszom testowym. Dane wyjściowe z każdej z prób są porównywane z danymi referencyjnymi. W ten sposób określana jest ogólna dokładność stworzonego modelu oraz wszelkie inne potrzebne wskaźniki.

2. Wykonanie symulacji SiL (ang. *Software-in-the-Loop*). W drugim etapie bazowy model jest przepisywany do docelowego języka programowania jakim jest najczęściej C/C++ . Następnie ponownie poddaje się go ewaluacji przy pomocy wybranych scenariuszy testowych. Celem próby jest sprawdzenie, czy kod programu został poprawnie przepisany.

3. Wykonanie symulacji HiL (ang. *Hardware-in-the-Loop*). Następuje implementacja stworzonego wcześniej modelu systemu do docelowej platformy, którą może być np. mikrokontroler, bądź układ reprogramowalny FPGA (ang. *Field-Programmable Gate Array*). Ponownie przeprowadzane są testy – dostarczane są dane wejściowe, a dane wyjściowe są pobierane z testowanego urządzenia, zapisywane na dysku i porównywane z danymi referencyjnymi.

4. Przeprowadzenie ewaluacji z udziałem samochodu. Po zamontowaniu docelowego systemu w pojeździe, przeprowadzane są jazdy testowe. Zazwyczaj mają miejsce na zamkniętych drogach, w sztucznie utworzonych miastach, czy wreszcie na ogólnodostępnych terenach. W przypadku dwóch pierwszych lokalizacji symulowane są pewne warunki ściśle określone przez niezależne organizacje takie jak Euro NCAP (ang. *European New Car Assessment Programme*) lub IIHS (ang. *Insurance Institute for Highway Safety*). W przypadku dróg ogólnodostępnych testowane systemy poddawane są rzeczywistym, docelowym warunkom.

Co należy podkreślić, symulacje MiL, SiL, HiL pierwotnie wymagają zamkniętego sprzężenia zwrotnego. Można o nim mówić wtedy, kiedy stany wyjściowe, bądź sygnały wyjściowe danego systemu oddziałują na jego stany lub sygnały wejściowe. Ujmując rzecz prościej, układ ze sprzężeniem zwrotnym dostaje informacje o tym, jaki jest wynik jego działania. Ze względu na ten element, tworzenie wszystkich trzech wymienionych wyżej rodzajów symulacji bardzo się komplikuje.

Z pewnością można stwierdzić, że żaden człowiek nie chciałby, aby urządzenie nigdy wcześniej nietestowane w warunkach rzeczywistych, nagle znalazło się w pojeździe, który swobodnie porusza się po drodze. Z drugiej strony, przeprowadzanie samych testów w ściśle określonych, spreparowanych warunkach i dopuszczenie pojazdów na ogólnodostępne drogi może nieść za sobą zagrożenie dla wszystkich uczestników ruchu drogowego, włączając kierowcę tegoż samochodu. Zatem każdy z wyżej wypunktowanych sposobów ewaluacyjnych musi być wykonany, aby nowy system mógł zostać wdrożony do auta produkowanego seryjnie.

Skomplikowane testy wykonane w przygotowanych warunkach i wysyłanie samochodów w różne części świata wiąże się ze znacznymi obciążeniami czasowymi oraz z wysokimi kosztami całego procesu. Producenci samochodów coraz częściej sięgają po różnorodne systemy

testowe oparte o metodologię HiL. Umożliwiają one sprawdzenie nowych rozwiązań w warunkach laboratoryjnych dla zróżnicowanych scenariuszy.

Realizacja środowiska testowego z wykorzystaniem metodologii *Hardware-in-the-Loop* jest zagadnieniem nieco bardziej złożonym niż w przypadku innych rodzajów symulacji. Dzieje się tak, gdyż ewaluowany jest gotowy sprzęt, który w przyszłości ma być umieszczony w samochodzie. Należy wziąć zatem pod uwagę wymagania takiego urządzenia.

Komponenty auta do poprawnego działania potrzebują informacji o stanie pojazdu, w którym są zamotowane [3]. Takie dane muszą przybierać z góry określoną postać, aby mogły być odpowiednio zinterpretowane. Podobnie jest w przypadku danych odbieranych od testowanych układów – tutaj również istnieje opisana zależność. Symulacja *Hardware-in-the-Loop* niesie więc za sobą konieczność dopasowania odpowiednich informacji wejściowych i wyjściowych. Wymaga to dobrej znajomości sposobów komunikacji akceptowanych przez urządzenie. Ich implementacja musi przebiegać w taki sposób, jak to ma miejsce w samochodzie.

Realizacja środowiska testowego z wykorzystaniem *Hardware-in-the-Loop* dla układów samochodowych stwarza jeszcze jeden problem. Aby poprawnie zweryfikować dane wyjściowe z systemu wspomagania kierowcy, należy mieć możliwość przeprowadzenia symulacji w takich warunkach czasowych jak ma to miejsce w pojazdach. Pomocą są tutaj systemy czasu rzeczywistego. Platformy, które umożliwiają pracę w czasie rzeczywistym są zazwyczaj skomplikowane w obsłudze, ale przede wszystkim bardzo drogie.

W przypadku samochodowych układów wizyjnych jednym z najistotniejszych zagadnień dla systemów testowych jest sposób dostarczania informacji do kamery. Najczęściej w tym celu stosuje się wcześniej nagraną jazdę testową, komputerowo wygenerowaną symulację na podstawie rzeczywistych danych wejściowych, bądź całkowicie sztucznie wygenerowany obraz wideo. Sygnał wizyjny może być wyświetlany na specjalnym ekranie i obserwowany przez kamerę, bądź „wstrzykiwany" bezpośrednio do kamery. Wstrzykiwanie oznacza wprowadzenie sygnału z pominięciem sensora wizyjnego, bezpośrednio do modułów kamery odpowiedzialnych za analizę obrazu.

W dalszej części rozdziału omówione zostały sposoby testowania systemów samochodowych w oparciu o symulację HiL z wykorzystaniem sygnału wizyjnego.

1.1. Dostarczanie informacji do kamery

1.1.1. Wyświetlanie obrazu przy pomocy monitora

MiL (ang. *Monitor-in-the-Loop*) to jeden z podstawowych sposobów przeprowadzania testów wizyjnych samochodowych systemów bezpieczeństwa. Sztucznie generowany bądź wcześniej nagrany obraz jest wyświetlany za pomocą monitora bądź rzutnika. Taki film jest następnie bezpośrednio rejestrowany przez sensor wizyjny kamery ADAS, która jest ustawiona na wprost wyświetlanego obrazu (rys. 1.2).

Rysunek 1.2. *Kamera ADAS ustawiona na przeciwko monitora podczas symulacji Hardware-in-the-Loop*, źródło: opracowanie własne

Symulację z wykorzystaniem najpierw monitora, a później rzutnika, stworzyli autorzy publikacji [2]. Opracowany przez nich system VeHiL (ang. *Vehicle-Hardware-in-the-Loop*) jest swego rodzaju rozszerzeniem metodologii HiL. Autorzy zauważają, że klasyczny model testowania systemów wspomagania kierowcy nie bierze pod uwagę tego, że w samochodzie systemy te nie działają niezależnie od siebie, ale wszystkie ze sobą współpracują. Zatem testowanie poszczególnych części aktywnego systemu wspomagania kierowcy w ramach symulacji HiL również powinno się odbywać wspólnie. Jest to swego rodzaju wypełnienie luki, która znajduje się między sprawdzaniem ADAS w ramach symulacji HiL oraz jazdami.

Jak podkreślają twórcy publikacji [2] wykorzystanie monitora do celów wizualizacyjnych niesie za sobą pewne problemy, które należy wziąć pod uwagę przy projektowaniu systemu testowego. Przede wszystkim istotne jest pole widzenia dla sensora wizyjnego. Powinno być

ono ustawione w taki sposób, żeby pożądany obraz był jak najwierniej dostarczany do kamery. Ważne są także zniekształcenia obrazu takie jak dystorsja. Jest to wada optyczna polegająca na różnym powiększeniu elementów na obrazie w zależności od ich odległości od środka obrazu. Oznacza to, że tak wyświetlana scena może nie odwzorowywać rzeczywistości w sposób dokładny. Ponadto w pomieszczeniu, gdzie wyświetlany jest materiał wideo, należy unikać zewnętrznego źródła światła, które może wpływać na jasność wyświetlanego obrazu i powodować problemy z jego analizą. Do zalet tego sposobu testowania należą z całą pewnością prostota, gdyż jedyny potrzebny element to urządzenie służące do wizualizacji obrazu.

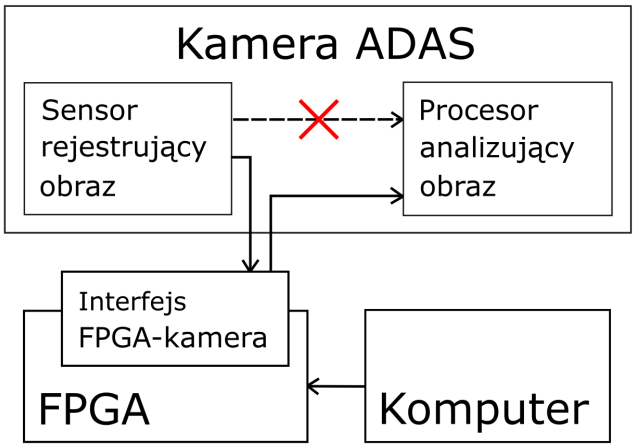
1.1.2. Wstrzykiwanie ramki obrazu

Wstrzykiwanie ramki obrazu bezpośrednio do kamery jest zagadnieniem znacznie bardziej złożonym od wcześniej omówionego wyświetlania sekwencji wideo na monitorze. Główną zaletą powyższego rozwiązania jest możliwość dostarczenia do układu obrazu, który jest w rzeczywistości dokładnie tym samym obrazem, który byłby do niego dostarczany podczas jazdy.

Autorzy pracy [1] jako sposób dostarczenia informacji do wykorzystanej przez siebie platformy Zedboard, wskazują protokół UDP (ang. *User Datagram Protocol* – protokół pakietów użytkownika). Rozwiązanie to, o ile ciekawe i niezbyt skomplikowane, jest jednak sporym uproszczeniem i ma wartość przede wszystkim akademicką. W rzeczywistych systemach wspomagania kierowcy nie ma możliwości wysyłania informacji wideo z wykorzystaniem *Ethernetu*.

Inne podejście wskazują autorzy [3]. Zaznaczają oni, że typowa kamera ADAS składa się z dwóch części: sensora wizyjnego oraz jednostki służącej do analizy obrazu. Zazwyczaj obie te części są ze sobą ściśle zintegrowane. Aby umożliwić wstrzykiwanie ramki obrazu bezpośrednio do kamery należy to połączenie przerwać (rys. 1.3). Autorzy [3] umieścili jednostkę reprogramowalną FPGA w miejsce przerwanego połączenia. FPGA otrzymuje standardowy, nagrany wcześniej materiał filmowy z komputera. Następnie wykonywana jest konwersja zgodna z formatem obrazu dostarczanego przez sensor kamery ADAS. Taki obraz jest dostarczany do procesora odpowiedzialnego za analizę obrazu.

Podejście przedstawione w [2, 3] ma znaczenie praktyczne. Wykorzystywany jest bowiem równoległy interfejs wizyjny, który jest wymagany przez kamery ADAS. Jak pokazują wyniki projektu z publikacji [3], rozwiązanie to już niedługo może skutecznie zastąpić dotychczasowe metody testowania wizyjnych systemów wspomagania kierowcy.



Rysunek 1.3. *Schemat systemu do wstrzykiwania obrazu do kamery ADAS przy pomocy układu FPGA*, źródło: opracowanie własne na podstawie [3]

1.2. Rodzaj dostarczanej informacji

1.2.1. Komputerowo wygenerowana scena

Współczesne możliwości symulacyjne i wizualizacyjne są ogromne. Istnieje bardzo wiele programów umożliwiających generowanie scen testowych, które w dużym stopniu naśladują rzeczywistość. Dla przykładu można wymienić aplikacje MotionDesk, Prescan, Pro-Sivi, czy CarMaker (rys. 1.4).

Rysunek 1.4. *Możliwości wizualizacyjne programu CarMaker na przykładzie wygenerowanej w nim drogi podmiejskiej*, źródło: opracowanie własne

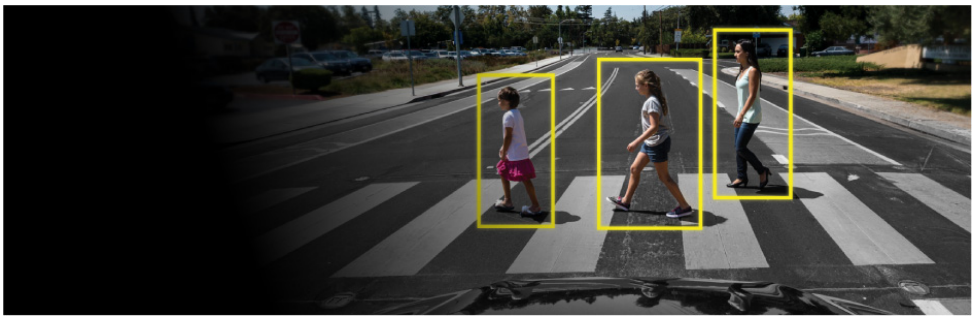
W publikacji [6] skorzystano z możliwości programu CarMaker. Rozwiązanie jest bardzo proste – danymi wejściowymi dla rozważanego systemu asystenta pasa ruchu były kolejne, wygenerowane w programie symulacje wideo. Wyniki symulacji zostały zestawione z modelem liniowym, co pozwoliło stwierdzić skuteczność implementacji. Okazała się ona zadowalająca. Autorzy nie pokusili się jednak o porównanie otrzymanych rezultatów z testami w prawdziwych warunkach.

Publikacje [4, 5] pokazują odmienne podejście. Z nagranej wcześniej, rzeczywistej sekwencji wideo wydobywane są wszelkie istotne informacje takie jak krzywizna drogi, informacje o testowanym pojeździe oraz konieczne informacje na temat środowiska. W dalszej kolejności generowana jest scena 3D. O ile jednak wyniki symulacji z takimi danymi wejściowymi są dla niektórych algorytmów zadowalające, dla innych znacznie odbiegają od rzeczywistości. Podane rozwiązanie wymaga zatem dalszych usprawnień i w obecnej formie nie nadaje się do testowania każdego rodzaju systemów wizyjnych.

Oba podejścia łączy ze sobą prostota, a komputerowa generacja scen stoi obecnie na bardzo wysokim poziomie. Nie sposób jednak ukryć, że nie jest to poziom fotorealistyczny. Otrzymane dzięki takim scenariuszom testowym wyniki, powinny zostać więc zestawione z danymi rzeczywistymi.

1.2.2. Rzeczywiste wideo

O ile komputerowa generacja scen może dostarczać wideo niepokrywające się z rzeczywistością, o tyle nie da się tego powiedzieć o nagraniach wideo powstałych podczas jazd po drogach. Jest to ogromna przewaga takich obrazów nad wizualizacjami wygenerowanymi przy pomocy sprzętu. Autorzy [1, 2, 3] do testów swoich systemów używają rzeczywistych nagrań (rys. 1.5), podkreślając w ten sposób możliwość ponownego używania takich obrazów do ewaluacji kolejnych wersji systemów wizyjnych ADAS. Porównanie skuteczności systemów wykorzystujących rzeczywiste nagrania z bezpośrednimi testami na drogach, pozwala dojść do wniosku, że jest to bardzo dobry sposób ewaluacji układów wspomagania kierowcy.



Rysunek 1.5. *Detekcja pieszych na wcześniej nagranej scenie*, źródło: [8]

Oczywiste zalety tego rozwiązania nie mogą jednak przysłonić bardzo istotnych wad. Zbieranie danych z jazd po ulicach wiąże się z koniecznością zaangażowania w badania dodatkowych ludzi, a to z kolei pociąga za sobą kwestie finansowe. Dodatkowo raz zebrane dane, aby móc wykorzystać je ponownie, należy przechowywać. Często są to ogromne ilości danych, sięgające petabajtów. Infrastruktura potrzebna dla takiego rozwiązania jest bardzo droga w utrzymaniu.

2. System testowy dla kamery ADAS

Niniejszy rozdział traktuje o sposobie implementacji docelowego systemu testowego dla kamery ADAS (ang. *Advanced Driver Assistance Systems* – zaawansowane systemy wspomagania kierowcy). Na początku szczegółowo omówiono wykorzystany sprzęt. Opis użytych elementów został opracowany na podstawie źródeł danych takich jak karty katalogowe produktów. Zamieszczono też informacje o tym, jakie role w projekcie pełniły poszczególne platformy. W kolejnych sekcjach omówiono etapy powstawania projektu inżynierskiego oraz zaprezentowano schematy stworzonego środowiska.

Jak zauważono we wcześniejszej części pracy, stworzenie środowiska opartego o metodologię *Hardware-in-the-Loop* wiąże się przede wszystkim z realizacją komunikacji pomiędzy odpowiednimi platformami sprzętowymi systemu. Zadanie nie jest trywialne i w związku z tym poświęcono mu dużą część tego rozdziału.

2.1. Wykorzystany sprzęt

2.1.1. Kamera ADAS

Kamera zaawansowanego systemu wspomagania kierowcy stanowiła centralny punkt projektu inżynierskiego. Urządzenie było złożone z dwóch części: cyfrowego sensora wizyjnego wykonanego w technologii CMOS (ang. *Complementary Metal Oxide Semiconductor*) oraz jednostki dokonującej analizy obrazu. Dzięki obecności interfejsu USB JTAG (ang. *Universal Serial Bus Joint Test Action Group*) i przy użyciu specjalnych, dedykowanych aplikacji, możliwe jest podglądanie całej ramki obrazu, przed i po przejściu przez procesor graficzny w kamerze.

Głównym zadaniem sensora wizyjnego jest rejestrowanie informacji wideo ze świata rzeczywistego i przekazywanie tych informacji w odpowiednim formacie do kolejnych elementów kamery. Wiodącymi producentami sensorów wizyjnych wykorzystywanych w systemach ADAS są: ON Semiconductor (dawniej Aptina), OmniVision oraz Melexis. Dla celów poniższej pracy omówiono przykładowy sensor o oznaczeniu AR0132AT firmy ON Semiconductor [9].

W inteligentnych kamerach stosowanych w przemyśle samochodowym [9], wspomniany sensor dołącza odpowiednie dane do ramki obrazu (rys. 2.1). Wyjściem z urządzenia jest obraz o specyficznej rozdzielczości: 1284 kolumn na 968 wierszy. Pierwsze dwa wiersze zawierają informacje na temat wszystkich ustawień kamery dla danej ramki obrazu. Dane pobierane są z odpowiednich rejestrów, a komunikacja z nimi odbywa się z wykorzystaniem szeregowej, dwukierunkowej magistrali I^2C (ang. *Inter-Integrated Circuit* – pośredniczący pomiędzy układami scalonymi). Kolejne wiersze zawierają piksele obrazu, zaś pod nimi znajdują się dwie linie statystyk, generowanych na podstawie aktualnej ramki obrazu. Wśród tych ostatnich danych najważniejsze miejsce ma histogram.

Rysunek 2.1. *Ramka obrazu wyjściowego z sensora AR0132AT*,
źródło: opracowanie własne

Parametr opisujący każdy piksel w obrazie to jasność. W zależności od specyfiki obrazu jasność może mieć różny zakres. W przypadku powyższego sensora każdy piksel jest 12-, 14- lub 20-bitowy w trybie szeregowym lub 12-bitowy w trybie równoległym. Oznacza to, że jasność piksela waha się od 0 do 2^n, gdzie n jest liczbą bitów na piksel.

Histogram obrazu to rozkład jasności pikseli. Jest najczęściej prezentowany graficznie w postaci wykresu słupkowego. Histogram obliczany w sensorze wizyjnym składa się z 244 poziomów jasności, przy czym w zależności od wartości luminancji piksela, liczba progów jest zróżnicowana:

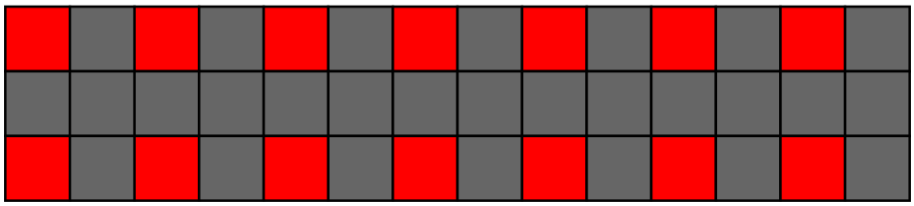
– 64 poziomy histogramu dla pikseli o wartości $0 - 2^{12}$,

– 120 poziomów histogramu dla pikseli o wartości $2^{12} - 2^{16}$,

– 60 poziomów histogramu dla pikseli o wartości $2^{16} - 2^{20}$.

Celem dostarczenia histogramu przez sensor jest przyporządkowanie dostarczanych wartości 12-bitowych do wartości docelowych 8-bitowych, które ma miejsce w procesorze. Takie przyporządkowanie nazywane jest mapowaniem.

Sensor wizyjny może dostarczać obraz, w różnych systemach barwnych, między innymi RGB (ang. *Red, Green, Blue* – czerwony, zielony, niebieski). W przemyśle samochodowym częściej jednak stosuje się przestrzeń RCCC (ang. *Red, Clear, Clear, Clear* – czerwony, czysty, czysty, czysty), gdzie „czysty" oznacza obraz w skalach jasności (rys. 2.2).



Rysunek 2.2. *Filtr RCCC stosowany w przemyśle samochodowym, każdy kwadrat oznacza jedn piksel*, źródło: opracowanie własne na podstawie [9]

2.1.2. Platforma dSPACE

Platforma dSPACE (ang. *Digital Signal Processing and Control Engineering*) to przykład sprzętowego systemu czasu rzeczywistego [11].

dSPACE jest wysoce zintegrowany z programem MATLAB oraz pakietem Simulink firmy Mathworks. Simulink to środowisko graficzne, w którym za pomocą bloczków w łatwy sposób można opisać pożądany algorytm lub system. Wraz ze specjalnym rozszerzeniem – Simulink Coder – oferuje możliwość konwersji przygotowanego modelu do kodu w języku C. Taki kod za pomocą programu graficznego Control Desk jest wczytywany bezpośrednio na platformę dSPACE. Dzięki bogatej bibliotece dla aplikacji Simulink, możliwa staje się obsługa magistrali takich jak CAN (ang. *Controller Area Network*) oraz protokołów typu UDP (ang. *User Datagram Protocol* – protokół pakietów użytkownika). Urządzenie komunikuje się z komputerem przy pomocy światłowodu. Zmienne użyte przy tworzeniu modelu mogą być podglądane, bądź zmieniane podczas pracy systemu dSPACE.

dSPACE (rys. 2.3) ma budowę modułową. Oprócz procesora, na którym uruchamiana jest symulacja, można do niego dołączać karty według własnych potrzeb. Instalacja modułu DS4302 umożliwia obsługę magistrali CAN, zaś DS4121 dodaje możliwość komunikacji z wykorzystaniem protokołu UDP.

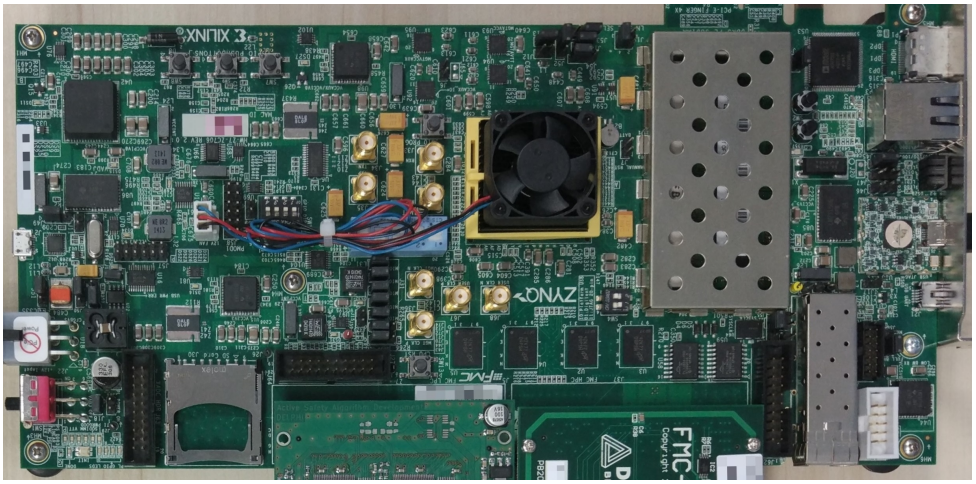
Rysunek 2.3. *Jednostka dSPACE z zainstalowanym procesorem DS1006 i kartami rozszerzeń*, źródło: opracowanie własne

Ponadto dSPACE oferuje obsługę aplikacji CarMaker. Jest to środowisko umożliwiające tworzenie zaawansowanych symulacji z wykorzystaniem samochodów osobowych. Po skonfigurowaniu połączenia pomiędzy programem a platformą, możliwe jest podglądanie aktualizowanej w czasie rzeczywistym wizualizacji. Kompleksowość tego rozwiązania zadecydowała o użyciu go do tworzenia docelowego systemu opisanego w poniższej pracy.

2.1.3. Karta Xilinx ZC706

Zastosowano kartę Xilinx ZC706 (rys. 2.4), na której znajdował się układ należący do grupy układów zwanych SOC (ang. *System On Chip*) [15]. Jest to układ, który zawiera kompletne, heterogeniczne środowisko elektroniczne. Układ SOC użyty w ZC706 składa się z zasobów reprogramowalnych w technologii FPGA (ang. *Field-Programmable Gate Array*) oraz procesora ARM (ang. *Advanced RISC Machine*) Cortex-A9. Obecność układu reprogramowalnego sprawia, że urządzenie świetnie nadaje się do zastosowań wizyjnych, wymagających szybkości oraz równoległego przetwarzania.

Istotną kwestią, która przyczyniła się do wyboru tego produktu, była możliwość obsługi interfejsu FMC (ang. *FPGA Mezzanine Card*). Jest to standard używany do podłączenia płytek, które rozszerzają działanie podstawowej karty z platformą FPGA, między innymi o obsługę transmisji danych z prędkością do 10Gbit/s czy też dużą liczbę portów wejściowych i wyjściowych. Na omawianej karcie Xilinx ZC706 znajdowały się dwa złącza FMC, co było bardzo istotne ze względu na możliwość dołączenia rozszerzeń niezbędnych w projekcie.



Rysunek 2.4. *Karta z układem SOC, zawierającym jednostkę reprogramowalną FPGA*, źródło: opracowanie własne

2.1.4. Karta Digilent FMC-HDMI

Jest to karta (rys. 2.5) rozszerzająca możliwości płytek z układami rekonfiguralnymi. Między innymi można ją dołączyć do wspomnianej karty Xilinx ZC706. Głównymi portami urządzenia są konektor FMC, który umożliwia współpracę z FPGA oraz dwa porty HDMI (ang. *High Definition Multimedia Interface* – multimedialny interfejs wysokiej rozdzielczości) [16].

Pierwszy z portów wykorzystuje bufor AD8195 i podaje na wyjście FMC zakodowany sygnał HDMI, zostawiając jego rozkodowanie innemu układowi. Drugi port zawiera odbiornik ADV7611 wraz z procesorem, który umożliwia dekodowanie dostarczonego sygnału i przekazanie go dalej przez FMC. Odbiornik i bufor są produkowane przez firmę Analog Devices.

Rysunek 2.5. *Karta z interfejsem FMC, rozszerzająca interfejsy płyt z FPGA o wejście HDMI*, źródło: opracowanie własne

2.1.5. Moduł FMC do połączenia z kamerą ADAS

W związku ze specyficznym połączeniem pomiędzy jednostką analizującą obraz a sensorem wideo w kamerze, skorzystano z zaprojektowanej wcześniej karty [3] z interfejsem FMC rozszerzającej interfejsy z płyt FPGA o odpowiednie wyjście. Dzięki temu rozwiązaniu możliwa była komunikacja między kartą Xilinx ZC706 a procesorem wideo w kamerze.

2.2. Symulacja w środowisku CarMaker

CarMaker jest programem, który umożliwia tworzenie zaawansowanych symulacji dla samochodów osobowych. Odwzorowywane pojazdy są matematycznymi modelami pojazdów rzeczywistych, a ich zachowanie odpowiada prawdziwym samochodom. Powyższe modele są opisane równaniami ruchu i kinematyki oraz formułami o wysokiej złożoności, definiującymi wieloskładnikowość każdego auta [19].

Co istotne, w programie CarMaker każdy „wirtualny pojazd" ma swojego „wirtualnego kierowcę". Taki „kierowca" reaguje podobnie jak jego rzeczywisty odpowiednik – zmiana biegów wraz ze wzrostem prędkości samochodu, hamowanie, wraz ze wszystkimi opóźnieniami dla tego procesu oraz przyśpieszanie rozłożone w czasie. Ostatnim elementem każdej symulacji tworzonej w aplikacji CarMaker jest „wirtualna droga". Może być ona wygenerowana na podstawie rzeczywistej trasy, z odwzorowaniem takich jej elementów, jak: krzywizna, nachylenie, a nawet otoczenie. Całość odbywa się automatycznie, przy czym można skorzystać z takich środowisk jak *Google Earth*. Oprócz tego „wirtualna droga" może być od początku stworzona w omawianym programie. Wszystkie trzy „wirtualne elementy" tworzą VVE (ang. *Virtual Vehicle Environment* – wirtualne środowisko pojazdów) [19].

Dzięki temu środowisku, możliwe jest odtwarzanie nieskończonej liczby scenariuszy z wykorzystaniem zróżnicowanego otoczenia przy obecności różnorakich przeszkód. Program CarMaker daje sposobność do podglądu wszystkich zmiennych, które opisują parametry samochodu oraz do zmiany dużej części z nich. Między innymi przyspieszenia, siły nacisku na pedał hamulca i podobnych.

Ta część realizacji stworzonego środowiska testowego sprowadzała się do zdefiniowania wszystkich trzech wyżej wspomnianych elementów programu CarMaker – „wirtualnej drogi", „wirtualnego samochodu" i „wirtualnego kierowcy". Następnie nastąpiło dodanie otoczenia oraz generacja innych uczestników ruchu drogowego (rys. 2.6). W tym celu skorzystano z biblioteki, która była dołączona wraz z programem.

Rysunek 2.6. *Symulacja terenów podmiejskich stworzona w programie CarMaker*, źródło: opracowanie własne

Po zapisaniu projektu, automatycznie wygenerowano model w aplikacji Siumlink. Składał się on z trzech bloczków (rys. 2.7), które odpowiadały za komunikację programu MATLAB ze środowiskiem CarMaker w dalszej implementacji.

Rysunek 2.7. *Fragment okna programu Simulink z wygenerowanymi bloczkami łączącymi projekt z aplikacją CarMaker*,
źródło: opracowanie własne

2.3. Aplikacja w środowisku Simulink

W tej części rozdziału poruszono zagadnienia związane z aplikacją, która została stworzona w środowisku MATLAB przy pomocy pakietu Simulink. Oba narzędzia stanowią własność firmy MathWorks i służą do wykonywania obliczeń inżynierskich, projektowych oraz tworzenia zaawansowanych symulacji komputerowych z użyciem bloczków.

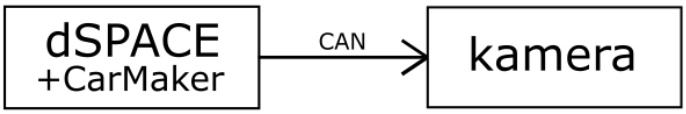
2.3.1. Komunikacja z wykorzystaniem CAN

CAN (ang. *Controller Area Network*) to szeregowa magistrala danych służąca wymianie danych pomiędzy systemami samochodowymi. Odgrywa szczególnie istotną rolę w przypadku układu bezpieczeństwa czynnego. Technologia została zaprojektowana w latach 80. ubiegłego wieku przez firmę Bosch. Współcześnie znajduje zastosowanie nie tylko w pojazdach, ale też w automatyce domowej i medycynie.

Warstwa fizyczna magistrali CAN składa się z dwu-przewodowej skrętki ze wspólną masą. Dane nadawane są za pomocą sygnału różnicowego, dzięki czemu CAN jest bardzo odporny na zakłócenia elektromagnetyczne. Najnowsza wersja standardu, opisującego tę magistralę – ISO (ang. *International Organization for Standardization* – Międzynarodowa Organizacja Normalizacyjna) 11898-2:2016 – określa prędkości przesyłu danych na sięgające $5Mbit/s$ [17].

W przypadku testowanej kamery ADAS, CAN był wykorzystywany zarówno w celu przesyłu jak i odbioru danych. Nadawanie informacji rozpoczynało się w momencie otrzymania przez urządzenie informacji o uruchomieniu pojazdu. Aby wysyłane przez kamerę pakiety były poprawne, musiała ona otrzymać informacje na temat prędkości samochodu, w którym jest zamontowana.

Z punktu widzenia pracy inżynierskiej, istotne było dostarczanie do kamery powyższych danych. Urządzenie nie było umieszczone w samochodzie, nie otrzymywało zatem żadnych informacji o stanie pojazdu. Aby móc zrealizować system testowy z użyciem metodologii HiL, zdecydowano się na użycie platformy dSPACE, umożliwiającej symulacje w czasie rzeczywistym. Całość sprowadzała się do pobrania odpowiednich danych z wcześniej stworzonej symulacji w środowisku CarMaker, a następnie do wysłania ich do testowanej kamery (rys. 2.8).

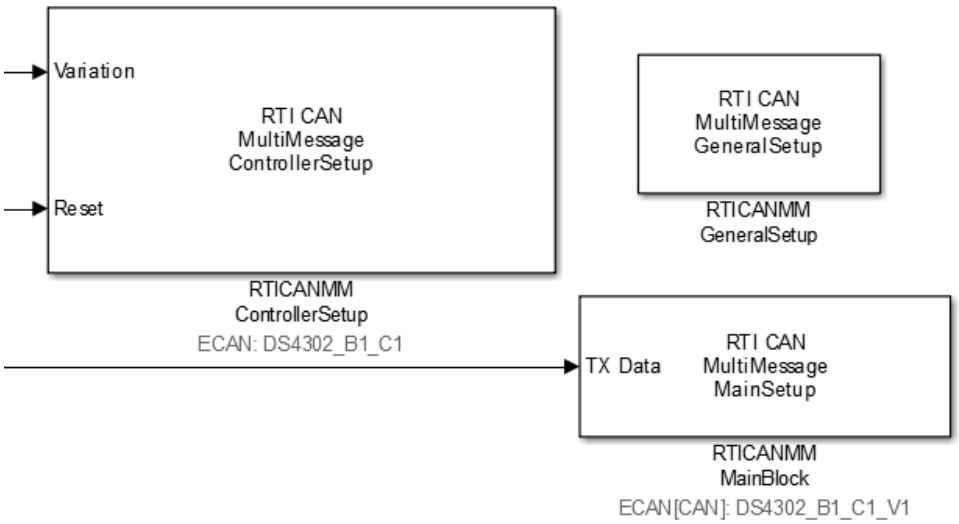


Rysunek 2.8. *Schemat podsystemu do obsługi magistrali CAN*,
źródło: opracowanie własne

Aby sprawdzić, czy dane są wysyłane poprawnie, wykonano prosty eksperyment. Połączono ze sobą dwa kanały CAN. Pierwszy z nich służył do wysyłania informacji, drugi miał je odbierać. Przy pomocy bloczków z biblioteki dSPACE, dołączonej do pakietu Simulink, wykonano odpowiedni model.

Model składał się z bloczków RTI (ang. *Real-Time Interface*): jednego *RTI CAN MultiMessage GeneralSetup*, dwóch *RTI CAN MultiMessage ControllerSetup* oraz dwóch *RTI CAN*

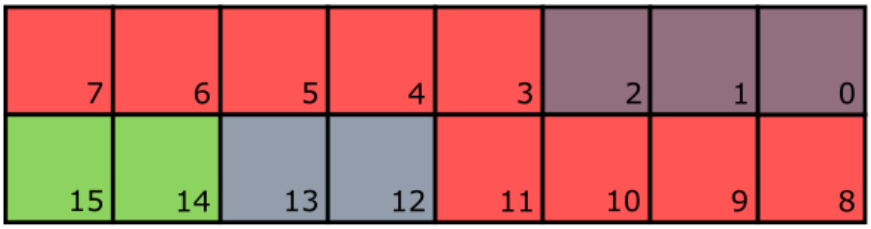
MultiMessage MainSetup (rys. 2.9). Zgodnie z [11] pierwszy bloczek umożliwia włączenie obsługi CAN dla dSPACE, drugi zaś odpowiada za obsługę każdego z kanałów magistrali CAN. Ustawienia tych bloczków pozostawiono domyślne. Dotyczyły one jedynie folderu, do którego generowany miał być później kod, wyboru karty obsługującej CAN (DS4302) oraz prędkości przesyłu danych (standardowe dla przemysłu motoryzacyjnego $500Kb/s$ [17]).



Rysunek 2.9. *Bloczki do obsługi magistrali CAN z biblioteki dSPACE*,
źródło: opracowanie własne

Następnie zajęto się konfiguracją bloczka *RTI CAN MultiMessage MainSetup*. To właśnie w nim definiowane są typy wiadomości oraz rodzaje sygnałów wysyłanych lub odbieranych w każdym z kanałów [12]. Konfiguracja sprowadzała się przede wszystkim do wskazania odpowiedniego pliku opisującego strukturę wysyłanej wiadomości poprzez magistralę CAN. Taki plik jest zapisywany w formacie DBC i najczęściej tworzony przy pomocy programu CANdb++ Editor firmy Vector Informatik. Pliki zawierają informacje o tym, jak kodowane są wszelkie wiadomości oraz wchodzące w ich skład pojedyncze sygnały o różnej długości bitów (rys. 2.10). Program CANdb++ Editor umożliwia sprawdzenie możliwych wartości każdego z sygnałów.

Po wskazaniu odpowiedniego pliku .dbc, wybrano sygnały, które miały być wysyłane z wykorzystaniem jednego kanału i odbierane przy pomocy drugiego. Po zaakceptowaniu zmian, wygenerowany został bloczek *Subsystem* zawierający bloczki uziemienia (wartość 0) połączone z odpowiednimi, wysyłanymi sygnałami oraz osobno bloczek *Demux*, rozdzielający wiadomość odbieraną na części. Wyjścia z tego ostatniego połączono z bloczkami *Display*, umożliwiającymi wyświetlanie wartości sygnałów. Symbole uziemienia zastąpiono bloczkami *Constant*



Rysunek 2.10. *Schemat przykładowej wiadomości wysyłanej po magistrali CAN, liczby odpowiadają kolejnym bitom wiadomości, zaś kolory mogą oznaczać różne sygnały takie jak przyspieszenie, prędkość, stan samochodu, ilość paliwa lub wiele innych, zależnie od zastosowanego sprzętu*, źródło: opracowanie własne

i wprowadzono ustawienia domyśle. Umożliwiło to operowanie wartościami sygnałów w programie Control Desk.

Tak stworzony układ poddano automatycznej generacji kodu, stworzono odpowiedni panel operatorski do podglądu wartości przekazywanych i otrzymywanych przez CAN (rys. 2.11). Ze względu na powtarzalność powyższych czynności w trakcie wykonywania docelowego środowiska testowego, zostały one omówione tylko raz w dalszych sekcjach tego rozdziału.

Variable	Value
Labels/DrivMan B	0
Labels/DrivMan B	0
Labels/DrivMan C	10
Labels/DrivMan G	35000
Labels/DrivMan G	0
Labels/DrivMan K	10
Labels/DrivMan S	99

(a) *Wartości wysyłane*

Variable	Value
TTCconf1/In1	0
TTCconf2/In1	0
TTCconf3/In1	10
TTCconf4/In1	35000
TTCconf5/In1	0
TTCconf6/In1	10
TTCconf7/In1	99

(b) *Wartości odbierane*

Rysunek 2.11. *Panel operatorski, umożliwiający przetestowanie połączenia za pomocą magistrali CAN*, źródło: opracowanie własne

Porównano wartości wysyłane i odbierane za pomocą magistrali CAN. Identyczność tych informacji pozwoliła stwierdzić poprawność stworzonego układu.

W ostatnim etapie usunięto kanał CAN odpowiadający za odbieranie danych oraz dołączono wcześniej wygenerowane bloczki umożliwiające komunikację z programem CarMaker. Do sygnału definiującego prędkość, z jaką porusza się symulowany pojazd, podpięto bloczek Read CM Dict (rys. 2.12). Zgodnie z [18], aktualna wartość zmiennej o nazwie wpisanej do

tego bloczka jest odczytywana z symulacji, a następnie podawana na jego wyjście. Do wejścia odpowiadającego za status symulowanego samochodu podpięto bloczek *Constant*, w który wpisano wartość stałą odpowiadającą stanowi – „uruchomiony”.

Rysunek 2.12. *Fragment modelu stworzonego w aplikacji Simulink wraz z bloczkiem Read CM Dict, umożliwiającym komunikację z programem CarMaker*, źródło: opracowanie własne

2.3.2. Komunikacja z wykorzystaniem UDP

UDP (ang. *User Datagram Protocol* – protokół pakietów użytkownika) jest jednym z podstawowych protokołów komunikacji internetowej [20]. Przy pomocy tego protokołu możliwe jest wysyłanie krótkich pakietów danych, czyli datagramów. UDP w przeciwieństwie do innego, popularniejszego protokołu przesyłu danych – TCP (ang. *Transmission Control Protocol*) nie kontroluje poprawności przesłanych pakietów. Umożliwia to większą prędkość transmisji danych oraz mniejsze opóźnienia w ich dostarczaniu. Wadą tego rozwiązania jest możliwość utracenia niektórych pakietów podczas komunikacji.

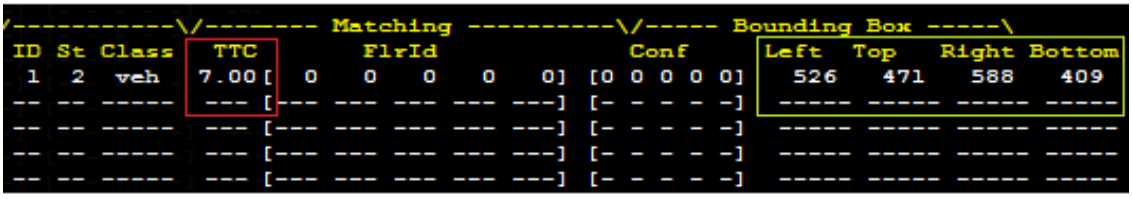
Protokół UDP znajduje zastosowanie przede wszystkim tam, gdzie krytyczna jest znajomość czasów opóźnienia, a pewna strata danych nie jest istotna ze względu na ich dużą ilość. UDP jest czasami wykorzystywany w grach internetowych oraz przy komunikacji wideo.

Omawiany protokół należy do warstwy czwartej – transportowej – ISO OSI RM (ang. *International Organization for Standardization Open Systems Interconnection Reference Model* – model odniesienia łączenia systemów otwartych Międzynarodowej Organizacji Normalizacyjnej). Model ten zapewnia podział systemów sieciowych na siedem współpracujących ze sobą

warstw. W warstwie transportowej dochodzi do odpowiedniego podziału danych w taki sposób, aby umożliwić komunikację pomiędzy dwoma różnymi urządzeniami w sieci [21].

Testowania kamera umożliwiała przesyłanie informacji o wykrytych obiektach przy pomocy UDP. Wśród dostarczanych danych znajdowały się między innymi: numery wierszy i kolumn wierzchołków prostokąta, którymi można otoczyć rozpoznany obiekt, czas do kolizji symulowanego samochodu z rozpoznanym obiektem oraz klasa rozpoznanego pojazdu. Kamera rozpoczynała nadawanie powyższych danych zaraz po dostarczeniu do niej odpowiednich informacji na temat symulowanego samochodu z wykorzystaniem magistrali CAN (poprzedni podrozdział).

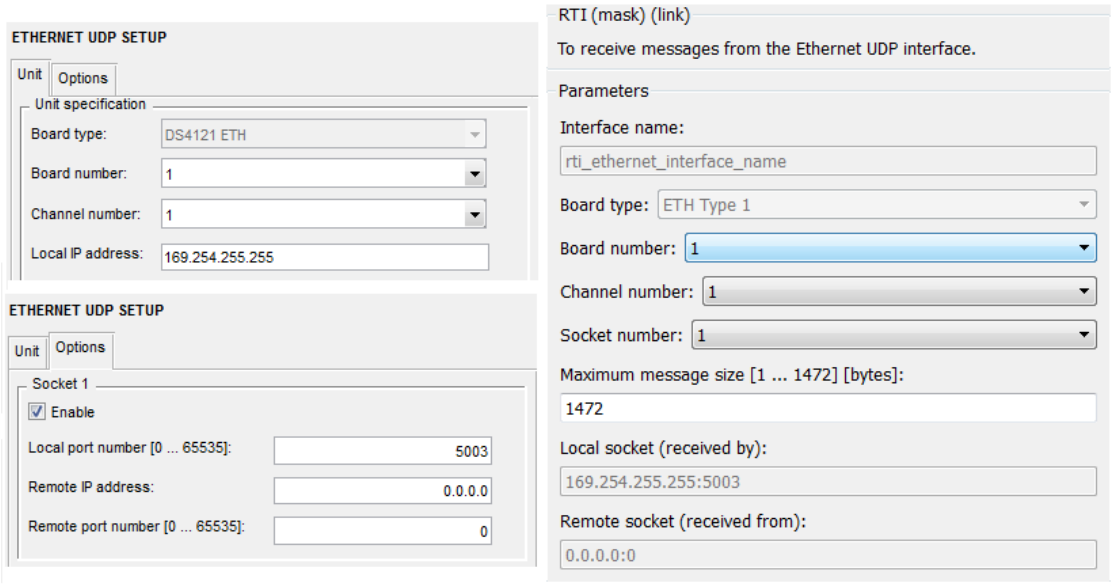
Odbiór danych był możliwy po podłączeniu kamery z komputerem przy pomocy *Ethernetu*, opisującego standardy budowy lokalnych sieci komputerowych, i ustawieniu odpowiedniego adresu IP zamontowanej w komputerze karty sieciowej. Kamera wysyłała dane tylko pod z góry określony adres IP. Urządzenie zostało ustawione naprzeciwko monitora i za pomocą magistrali CAN wysłano informacje o rzekomej prędkości symulowanego pojazdu (niezmiennej w czasie) oraz o tym, że samochód jest uruchomiony, a kamera rozpoczęła nadawanie danych. Podgląd odbieranych za pomocą UDP danych był możliwy przy pomocy dedykowanej aplikacji o nazwie DVTool, powstałej w firmie Delphi do celów wewnętrznych (rys. 2.13).



Rysunek 2.13. *Fragment programu DVTool z podglądem danych otrzymywanych z kamery ADAS, na czerwono – czas do kolizji, na zielono informacje o prostokącie, którym można otoczyć wykryty obiekt*,
źródło: opracowanie własne

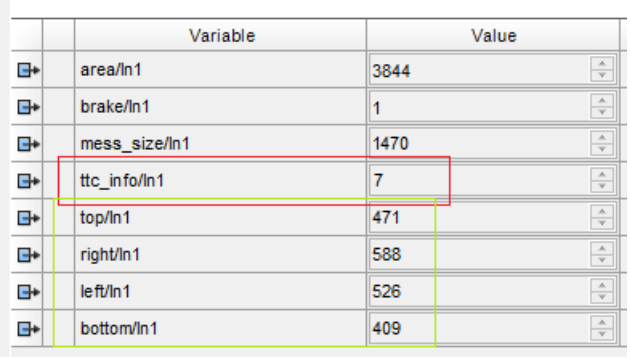
Otrzymywane w ten sposób dane nie mogły być wysyłane do tworzonej aplikacji na platformę dSPACE. Komunikacja za pomocą *Ethernetu* musiała odbywać się bezpośrednio między kamerą a wykorzystanym urządzeniem czasu rzeczywistego. W celu osiągnięcia tego założenia skorzystano ze specjalnej karty DS4121 rozszerzającej możliwości podstawowej platformy dSPACE o obsługę protokołu UDP. Analogicznie do obsługi magistrali CAN, skorzystano ze specjalnych bloczków z biblioteki pakietu Simulink. Tym razem były to *Ethernet UDP Setup*

oraz *Ethernet UDP Receive*. Zadaniem pierwszego z nich było zapoczątkowanie komunikacji z wykorzystaniem protokołu UDP, drugi odpowiadał za odbiór danych. Konfiguracja obu była bardzo intuicyjna (rys. 2.14). W przypadku pierwszego z nich sprowadziła się do ustawienia adresów IP i numeru portu na karcie DS4121, analogicznie jak w przypadku karty sieciowej w komputerze oraz wprowadzeniu adresu IP oraz portu, z którego nadchodziły dane. Jako dwie ostatnie wartości wybrano 0.0.0.0 oraz 0, co umożliwiało odbieranie danych od wszystkich adresów i z wszelkich możliwych portów [13]. Dane nie napływały w zbyt dużej ilości, gdyż do platformy dSPACE za pośrednictwem *Ethenertu* zostało podpięte tylko jedno urządzenie – testowana kamera. Przy konfiguracji drugiego bloczka wybrano jedynie rozmiar otrzymywanej wiadomości – ustawiono go na maksymalny, resztę wartości pozostawiono z ustawionymi wartościami domyślnymi [13].



Rysunek 2.14. *Konfiguracja bloczków pakietu Simulink z biblioteki dSPACE, umożliwiających komunikację z wykorzystaniem protokołu CAN*,
źródło: opracowanie własne

Tak stworzony model umożliwiał odbiór wszystkich potrzebnych danych, co przetestowano generując kod oraz zapisując go na platformę dSPACE. Odbierane dane wyświetlono przy pomocy stworzonego panelu operatorskiego (rys. 2.15). Pokrywały się one z danymi podglądanymi przy pomocy wcześniej wspomnianego programu DVTool (rys. 2.13), co pozwoliło stwierdzić, że opracowano poprawne rozwiązanie problemu.



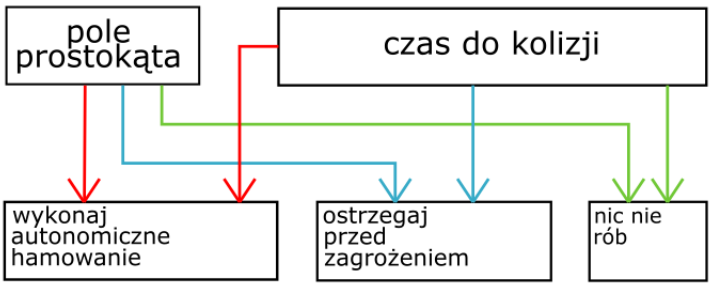
Rysunek 2.15. *Panel operatorski z opcją podglądu danych otrzymywanych przez UDP, na czerwono – czas do kolizji, na zielono informacje o prostokącie, którym można otoczyć wykryty obiekt*, źródło: opracowanie własne

2.3.3. Logika autonomicznego hamowania

Po rozwiązaniu dwóch najistotniejszych problemów, skupiono się na stworzeniu systemu automatycznego hamowania, który miałby następnie być sprawdzony przy pomocy tworzonego środowiska testowego w oparciu o metodologię *Hardware-in-the-Loop*.

Współczesne systemy AEB (ang. *Autonomous Emergency Braking* – autonomiczne hamowanie awaryjne) bazują na podstawie wielu danych, pochodzących nie tylko z kamer, ale także radarów oraz innych czujników. Jako że celem pracy było stworzenie środowiska wyłącznie dla systemu wizyjnego, postanowiono ograniczyć się jedynie do danych pochodzących z procesora urządzenia wideo.

W celu realizacji logiki AEB (rys. 2.16) wykorzystano informacje o prostokącie, którym można otoczyć obiekt wykryty przez kamerę ADAS oraz dane na temat czasu do kolizji symulowanego pojazdu z rozpoznaną przeszkodą.

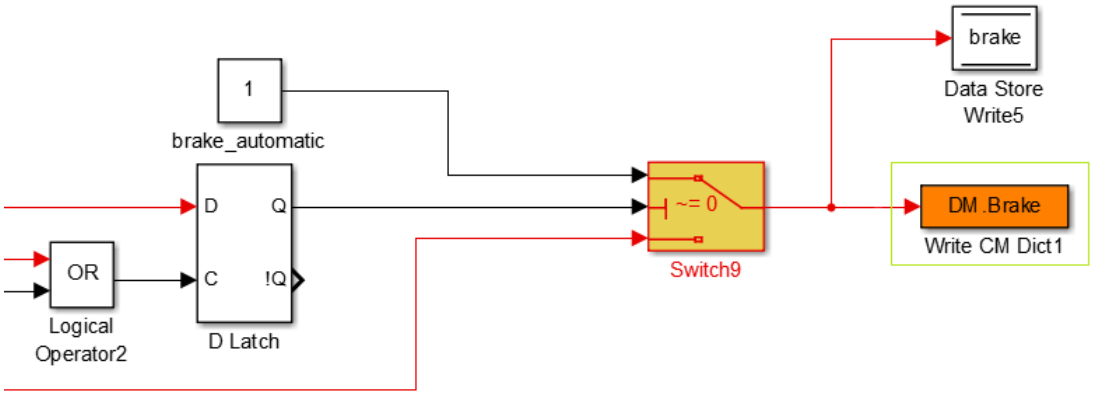


Rysunek 2.16. *Schemat systemu automatycznego hamowania awaryjnego, stworzonego w celu przetestowania kamery ADAS; kolor czerwony – próg został przekroczony, kolor niebieski – wartość zbliża się do progu, kolor zielony – wartość nie zbliża się do progu*, źródło: opracowanie własne

Jeśli pole powyższego prostokąta było większe niż ustanowiony próg lub czas do kolizji z przeszkodą spadł poniżej pewnej, określonej wartości rozpoczynano procedurę hamowania.

Oprócz powyższego systemu bezwarunkowego hamowania, zaimplementowano również możliwość sygnalizowania nadchodzącego niebezpieczeństwa. W momencie kiedy pole prostokąta lub czas do kolizji były blisko swoich wartości progowych, pojawiała się informacja o zbliżającym się niebezpieczeństwie. Wartości progów dobrano eksperymentalnie, testując różne sytuacje. Całość zaimplementowano wykorzystując środowisko Simulink.

Informacje na temat wyżej wspomnianego prostokąta oraz czasu do kolizji pobierano przy pomocy UDP, co osiągnięto poprzez wykorzystanie wcześniej stworzonego modelu. Podobnie, aby stwierdzić responsywność systemu, połączono go z modelem do obsługi magistrali CAN oraz z modelem umożliwiającym połączenie z programem CarMaker. Wygenerowany sygnał hamowania podawano do środowiska CarMaker za pomocą bloczka *Write CM Dict* (rys. 2.17). Analogicznie jak w przypadku *Read CM Dict* konfiguracja odbywała się poprzez wpisanie nazwy odpowiedniej zmiennej ze środowiska CarMaker. W ten sposób pętla sprzężenia zwrotnego została zamknięta, co zostanie szerzej opisane w dalszej części rozdziału.



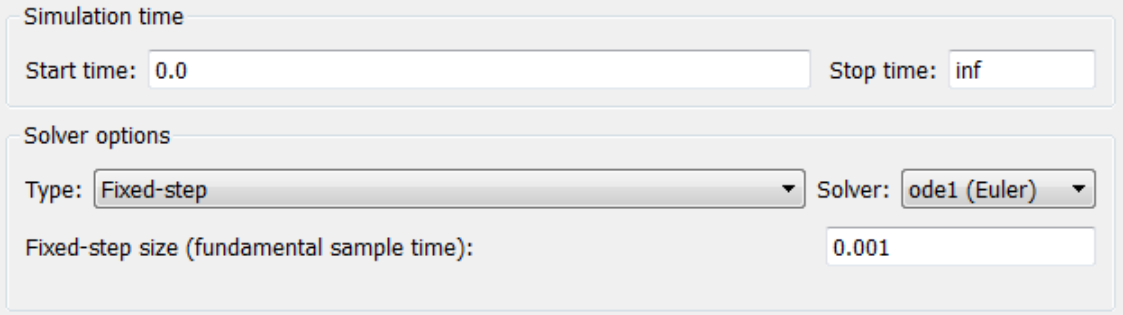
Rysunek 2.17. *Fragment logiki AEB stworzonej w aplikacji Simulink wraz z bloczkiem Write CM Dict, umożliwiającym wysyłanie danych do symulacji programu CarMaker*, źródło: opracowanie własne

2.3.4. Generacja kodu

Ważnym krokiem każdego z wyżej wymienionych etapów była generacja kodu do języka C na podstawie stworzonego modelu. Kod w takiej postaci mógł być, przy użyciu programu Control Desk, wczytany na platformę dSPACE, a tam uruchomiony.

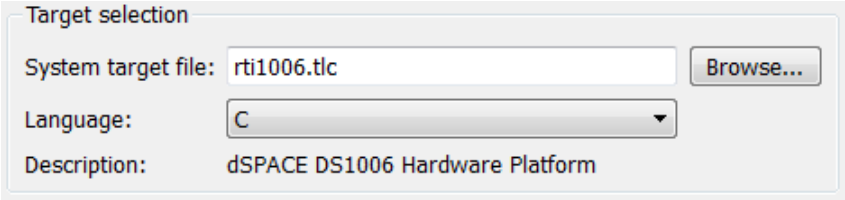
Generacja kodu odbywała się przy pomocy rozszerzenia pakietu Simulink, które nazywa się Simulink Coder. W menu *Configuration Parameters* należało użyć odpowiednich ustawień

w celu wygenerowania poprawnego kodu, współpracującego z odpowiednią platformą. W zakładce *Solver* (rys. 2.18) zdefiniowano czas startowy symulacji jako „0", co jest ważne w systemach czasu rzeczywistego. W przypadku czasu końcowego wybrano nieskończoność – „*inf*", aby umożliwić użytkowanikowi zatrzymanie symulacji w dowolnym momencie. Kolejno, na podstawie [22] ustawiono krok *solvera*, czyli sposobu obliczania kolejnych stanów układu, jako stały (*Fixed-step*) oraz wybrano metodę Eulera *ode1*, gdyż charakteryzuje się największą dokładnością spośród dostępnych. W celu zwiększenia dokładności wykonywanych obliczeń krok, z jakimi dokonywane było wyliczanie następnego stanu układu, ustawiono na 0,001. Skutkowało to zwiększeniem dokładności obliczeń, kosztem czasu uruchamiania symulacji [14].



Rysunek 2.18. *Ustawienia parametrów symulacji w programie Simulink*, źródło: opracowanie własne

W zakładce *Code Generation* (rys. 2.19) wybrano docelową platformę sprzętową jako DS1006 oraz docelowy język C, jako ten do którego ma być generowany model stworzonego systemu.



Rysunek 2.19. *Simulink, zakładka Code Genereation, wybór języka i platformy, dla której był generowany kod*, źródło: opracowanie własne

Po kliknięciu przycisku *Build* następował proces tworzenia pożądanego kodu. Tak otrzymany kod mógł zostać wczytany, a następnie uruchomiony na platformie dSPACE, co zostało szerzej opisane w dalszej części rozdziału.

2.4. Implementacja z udziałem platformy dSPACE

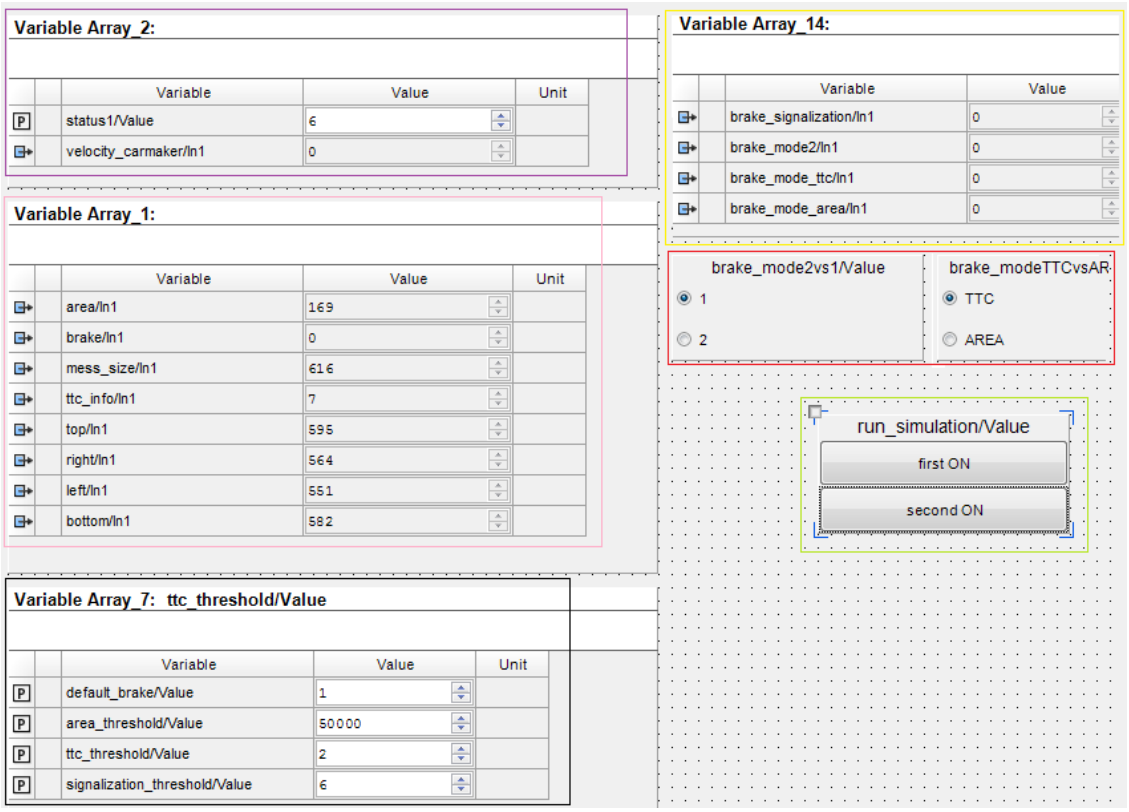
W tej części rozdziału poruszono zagadnienia związane z platformą dSPACE oraz programem Control Desk. Środowisko to oprócz wczytywania kodu do urządzeń czasu rzeczywistego, umożliwia również tworzenie paneli operatorskich do kontroli aplikacji uruchomionych w tych urządzeniach.

Wykonanie projektu w programie Control Desk można było podzielić na kilka etapów. W pierwszym z nich stworzono nowy projekt, wybrano docelową platformę sprzętową oraz wybrano miejsce na dysku, gdzie znajdował się wygenerowany wcześniej kod. Po zakończeniu czynności inicjujących wykonano kolejny krok. Stworzono panel operatorski, zarządzający symulacją.

Wykonanie panelu operatorskiego w środowisku Control Desk polegało na wybraniu sposobu wizualizacji oraz ustawiania danych. Spośród dostępnych opcji można było wybrać przełączniki, przyciski, wykresy, tabele i wiele innych. Kolejno z pliku opisującego zmienne użyte w modelu, który automatycznie załadował się do projektu, wybrano interesujące elementy i połączono je z układami graficznymi, tworząc ostateczny panel operatorski (rys. 2.20). Na wspomnianej grafice różnymi kolorami wyróżniono poszczególne sekcje:

- na czerwono – wybór czynnika decydującego o hamowaniu – pole prostokąta, którym można otoczyć wykryty obiekt, czy czas do kolizji z tym obiektem,
- na żółto – informacje o decyzji o hamowaniu, w zależności od wybranego czynnika decydującego,
- na zielono – przyciski uruchamiające symulację,
- na różowo – dane otrzymywane z wykorzystaniem UDP,
- na fioletowo – dane otrzymywane z wykorzystaniem CAN,
- na czarno – ustawienia progów hamowania i powiadomień o niebezpieczeństwie oraz informacja o zbliżającym się zagrożeniu.

Po uruchomieniu aplikacji na platformie dSPACE, otworzono program CarMaker. Wybrano odpowiedni projekt – ten, na podstawie którego była stworzona aplikacja w pakiecie Simulink. Następnie wskazano odpowiedni plik zawierający opis zmiennych z aplikacji wcześniej załadowanej na platformę dSPACE. Wszystkie powyższe czynności umożliwiły połączenie powyższego sprzętu z programem CarMaker.

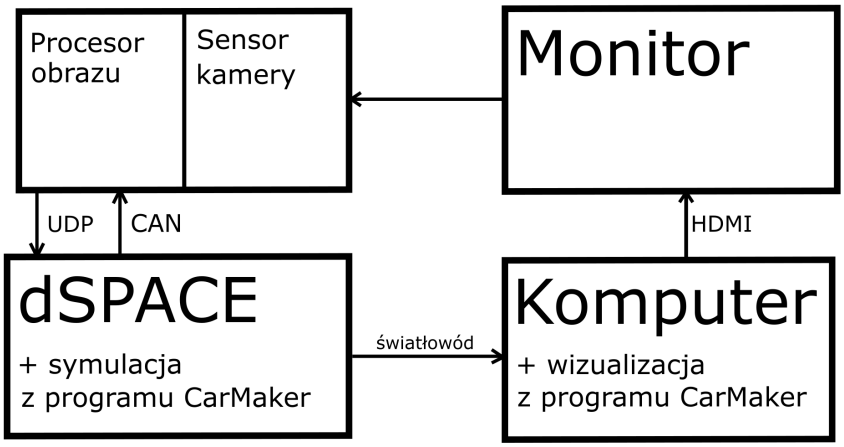


Rysunek 2.20. *Panel operatorski w programie Control Desk, umożliwiający zarządzanie stworzonym systemem testowym*, źródło: opracowanie własne

2.5. Zamknięcie pętli sprzężenia zwrotnego

W przypadku stworzonego systemu testowego (rys. 2.21) kamera otrzymywała trzy istotne informacje. Dane o obrazie (np. wizualizacja wygenerowana w programie CarMaker odbierana poprzez sensor wizyjny) oraz sztucznie wygenerowane dane o prędkości, z jaką poruszał się samochód, w którym kamera ta była „zamontowana" oraz wiadomość o tym, że „samochód" jest uruchomiony.

Dzięki tym informacjom, kamera – dokładniej procesor obrazu – był w stanie wysłać dane na temat odległości, w jakiej się „znajdowały" się inne obiekty na drodze od symulowanego samochodu. Jeśli obiekty znajdowały się w dalszej odległości, system nie reagował, gdy jednak symulowany samochód przybliżał się do przeszkody, wyświetlona była informacja o zbliżającej się kolizji. Jeśli nie została podjęta żadna akcja zapobiegająca wypadkowi, włączał się system automatycznego hamowania. Generowany sygnał hamowania wpływał na wizualizację programu CarMaker, sprawiając że testowany samochód unikał zderzenia z przeszkodą. Sprzężenie zwrotne zostało zamknięte.



Rysunek 2.21. *Schemat stworzonego środowiska testowego bez uwzględnienia możliwości zastąpienia sensora wizyjnego*, źródło: opracowanie własne

2.6. Wstrzykiwanie obrazu do kamery ADAS

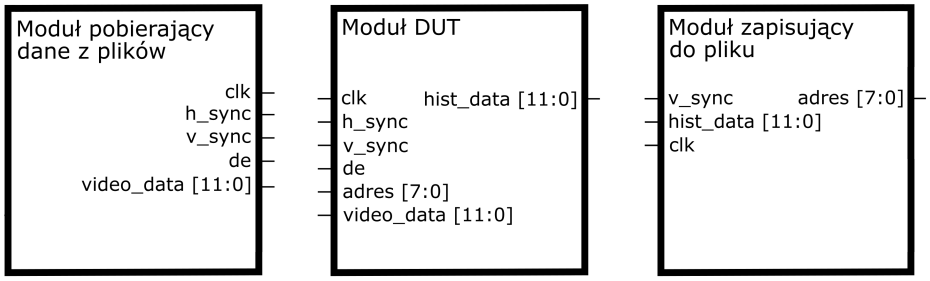
Kolejnym elementem pracy inżynierskiej było wykorzystanie układu FPGA (ang. *Field-Programmable Gate Array*) w celu zastąpienia sensora wizyjnego kamery.

2.6.1. Obliczanie histogramu

W początkowej fazie, przy pomocy środowiska MATLAB, napisano program do generacji 12-bitowych obrazów, a zatem takich, jakie dostarczał sensor wizyjny do kamery. Ramki były tworzone z wykorzystaniem funkcji *rand* z biblioteki pakietu MATLAB. Otrzymane wartości pomnożono następnie przez 2^{12}, gdyż wspomniana funkcja zwraca wartości z zakresu 0 – 1. Kolejno stworzone ramki zapisywano do formatu RAW (ang. *raw* – surowy) – bez nagłówków.

W kolejnej części skupiono się na przygotowaniu środowiska testowego dla modułu do obliczania histogramu (rys. 2.22). Implementacja została wykonana w programie Vivado firmy Xilinx, przy użyciu języku opisu sprzętu Verilog.

Środowisko składało się z modułu wczytującego kolejne obrazy z pliku RAW (rys. 2.22a). Wyjściami modułu były zegar (clk), sygnały synchronizacji (h_sync, v_sync), flaga, oznaczająca ważność piksela (de) oraz dane obrazu (video_data). Drugim elementem środowiska testowania był moduł (rys. 2.22c) do odczytywania danych z pamięci BRAM (ang. *Block Random Access Memory*) – docelowo wartości histogramu – i zapisywania ich do pliku, co zostało zrealizowane przy pomocy FSM (ang. *Finite State Machine* – skończona maszyna stanów). FSM

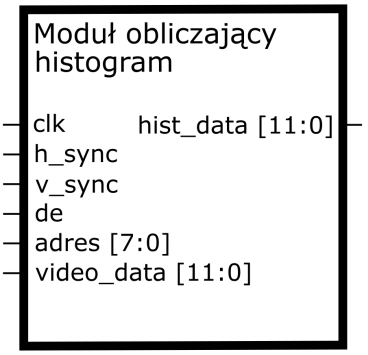


(a) *Odczyt z pliku* (b) *DUT* (c) *Zapis do pliku*

Rysunek 2.22. *Schematy modułów wraz z zaznaczonymi sygnałami wejściowymi i wyjściowymi, użyte do stworzenia środowiska testowego dla obliczania histogramu, źródło: opracowanie własne*

może być przedstawione przy pomocy grafu, w którym polami są odpowiednie stany. Od każdego stanu do kolejnego biegną strzałki wraz z umieszczonymi nad nimi opisami warunków. Strzałki obrazują przejścia z jednego stanu do drugiego. W implementacji sprzętowej FSM, bardzo dobrze nadają się one do obsługi pamięci oraz protokołów komunikacyjnych. Ostatnim stworzonym modułem w tej części, był moduł DUT (ang. *Design Under Test*), do którego wchodziły i wychodziły jedynie sygnały (rys. 2.22b).

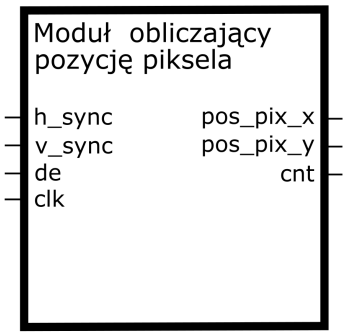
W kolejnej części zaprojektowano właściwy moduł do obliczania histogramu (rys. 2.23). Składał się on z dwóch bloków IP Core (ang. *Intellectual Property Core*) do generacji i zarządzania pamięcią BRAM. Bloki IP Core stanowią swego rodzaju „czarną skrzynkę” i po wstępnej konfiguracji umożliwiają przeprowadzanie wielu prostych bądź skomplikowanych operacji. Na ich wejście zazwyczaj podaje się sygnały wejściowe określone w specyfikacji takiego modułu, a otrzymuje pożądane, przetworzone sygnały wyjściowe.



Rysunek 2.23. *Schemat modułu do obliczania histogramu wraz z zaznaczonymi sygnałami wejściowymi i wyjściowymi*, źródło: opracowanie własne

Wygenerowane moduły IP Core nie były identyczne – pierwszy z nich umożliwiał obsługę pamięci dwu-portowej, Port o numerze jeden był w trybie *Write first* – najpierw zapisz podawaną do modułu wartość, później odczytaj. Drugi zaś w trybie, będącym odbiciem lustrzanym poprzedniego – *Read first*. Przy obliczaniu histogramu skorzystano również z modułu dodającego IP Core o latencji równej 0. Latencja określa, po ilu taktach zegara wartość z wejścia modułu zostanie wystawiona na jego wyjściu.

Dodatkowo napisano moduł (rys. 2.24), badający pozycję dla pojedynczego piksela. Zawierał on licznik (cnt) jedno-bitowy (odliczający do 1), resetujący się po uzyskaniu tej wartości. Licznik był wykorzystany do obsługi pamięci dwu-portowej. W pamięci tej przechowywany był histogram. W stworzonym układzie możliwe było odczytywanie wartości z pierwszych 244 komórek pamięci oraz zapisywanie do kolejnych 244, potem cyklicznie następowała zmiana. Jako że przestrzenią barw opisującą obraz wyjściowy sensora było RCCC, do obliczania histogramu należało wybrać tylko jeden piksel (z czterech, które przypadały na filtr) – ten który zawierał informacje o kolorze czerwonym. Zdefiniowano zatem kolejne dwa wyjścia z powyższego modułu określające pozycję piksela (pix_pos_x, pix_pos_y) w ramce obrazu. Umożliwiło to wybór piksela „R” z filtra.



Rysunek 2.24. *Schemat modułu do obliczania pozycji danego piksela z zaznaczonymi sygnałami wejściowymi i wyjściowymi*,
źródło: opracowanie własne

Moduł do przechowywania histogramu był pierwotnie zainicjowany zerami. Całkiem inaczej było w przypadku pamięci jedno-portowej. Jako że jej wejściem były 12-bitowe piksele, każdy z nich musiał być odpowiednio przyporządkowany do jednego z 244 komórek histogramu. Każda wartość 12-bitowa musiała zostać przepisana na wartość 0 – 244. Dokonano tego przy pomocy LUT (ang. *Look-Up Table*), narzędzia pozwalającego na mapowanie wartości. Pierwotne 12-bitowe piksele wchodziły do modułu LUT, aby w następnym takcie opuścić go jako wartość 0 – 244. Moduł IP Core został zainicjowany specjalnym plikiem o rozszerzeniu

COE, zawierającym rosnąco kolejno ułożone 2^{12} liczb, każda z zakresu 0 – 244. Plik ten wygenerowano w programie MATLAB. Wartości wyjściowe z bloku LUT były podawane na adres drugiego z portów pamięci BRAM. Z kolei zaś jego sygnał wyjściowy trafiał na moduł dodający, po czym trafiał na wejście pierwszego z portów tej samej pamięci. Zapis na tym porcie był aktywowany tylko wtedy, gdy odpowiednio opóźnione sygnały pozycji piksela oraz ważności obrazu wskazywały na filtr R z przestrzeni RCCC.

Odczytywanie wartości histogramu odbywało się na pierwszym porcie, w momencie kiedy sygnał synchronizacji pionowej (v_sync) opadał do 0, a zatem kiedy histogram obliczono dla całego obrazu.

W celu zweryfikowania poprawności działania stworzonego modułu, napisano program do obliczania histogramu w środowisku MATLAB. Po otrzymaniu danych wyjściowych implementacji sprzętowej (Vivado) oraz programowej (MATLAB), porównano oba wyniki. Okazały się identyczne. Pozwoliło to stwierdzić, że algorytm został zaimplementowany w sposób poprawny.

2.6.2. Dołączanie danych do ramki obrazu

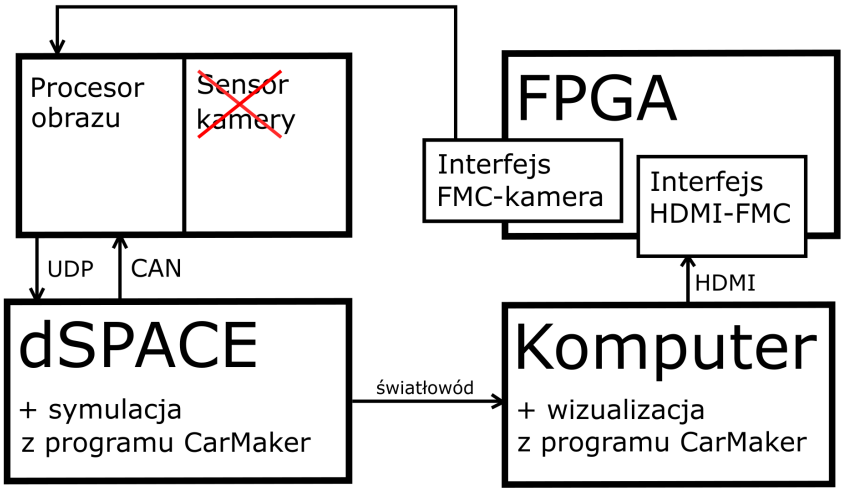
W celu dołączenia koniecznych danych do ramki obrazu, skorzystano z gotowego modułu zaprojektowanego w firmie Delphi do analogicznego celu. Działał on identycznie z napisanym wcześniej modułem, umożliwiającym odczyt histogramu w momencie wykrycia opadającego zbocza sygnału synchronizacji pionowej. Składał się zatem ze skończonej maszyny stanów.

Dane na temat rozkładu pikseli były opóźnione o jedną ramkę. Oznacza to, że zamieszczany na obrazie wyjściowym histogram był obliczony dla poprzedniej ramki, a nie dla tej, do której został dołączony. Było to pewnego rodzaju obejście problemu. W przypadku częstotliwości trzydziestu klatek na sekundę, z jaką pracowała kamera, problem ten był niezauważalny, a wręcz pomijalny.

Moduł dołączał również do ramki inne wymagane informacje. Ostatecznie nie wykorzystano jednak standardu I^2C do komunikacji z rejestrami kamery, gdyż wartości rejestrów nie były potrzebne jednostce analizującej obraz w takim stopniu, jak pierwotnie zakładano. Informacje na temat wykrytych obiektów wciąż były dostarczane prawidłowo przy wstrzykiwaniu nieco uboższego obrazu. Początkowe wiersze każdej ramki uzupełniono wartościami domyślnymi dla każdego z rejestrów kamery [9].

2.7. Schemat pełnego środowiska testowego

Po wykonaniu wstrzykiwania ramki wideo do kamery, możliwe było zastąpienie sensora wizyjnego w kamerze. Środowisko testowe umożliwiało wykonanie ewaluacji rozpatrywanej kamery z wykorzystaniem wbudowanego sensora wizyjnego (rys. 2.21) lub z jego pominięciem (rys. 2.25).



Rysunek 2.25. *Schemat stworzonego środowiska testowego z uwzględnieniem możliwości zastąpienia sensora wizyjnego*, źródło: opracowanie własne

3. Testy

W niniejszym rozdziale zwrócono uwagę na elementy potrzebne do przeprowadzenia ewaluacji stworzonego środowiska. W dalszej części przeanalizowano zebrane dane testowe, porównano je z dostępnymi danymi rzeczywistymi oraz sporządzono tabele zestawiające informacje o testowanym systemie.

3.1. Wstrzykiwanie ramki obrazu do kamery

3.1.1. Przeprowadzenie testów

Po stworzeniu opisu sprzętu przy pomocy języka Verilog, wygenerowano plik bitowy z konfiguracją sprzętu. Plik załadowano na kartę ZC706, podłączono do niej karty rozszerzeń, czyli Digilent FMC-HDMI oraz moduł umożliwiający podłączenie poprzez standard FMC bezpośrednio z kamerą. Głównym zadaniem tego modułu [3] była serializacja danych i przesłanie ich w takiej postaci do jednostki obliczajacej w kamerze. Karta produkcji Digilentu została wcześniej skonfigurowana przy pomocy programu programu MATLAB, według informacji z dokumentacji [16]. Konfiguracja odbywała się za pomocą komunikacji z wykorzystaniem I^2C. Do pamięci karty zostały wczytane następujące ustawienia: rozdzielczość 1284x968, częstotliwość odświeżania 30Hz, 12-bitowa przestrzeń kolorów.

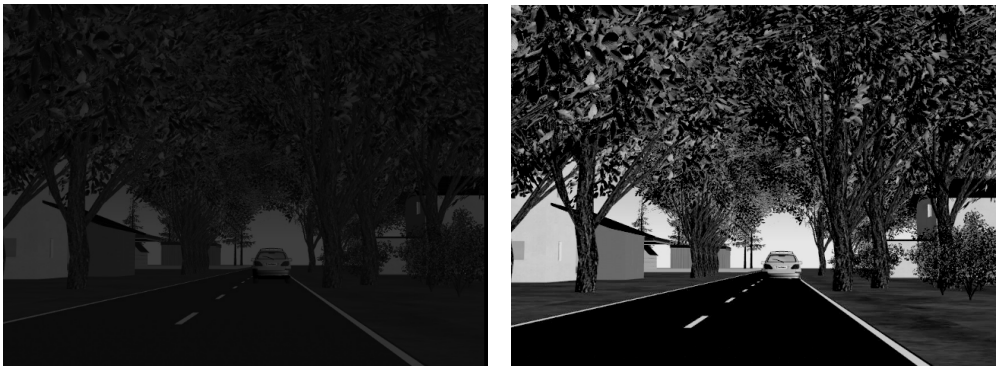
Po podłączeniu karty do komputera za pośrednictwem HDMI została ona rozpoznana jako standardowy monitor. W ustawieniach systemu operacyjnego Windows 7, dotyczących wyświetlania obrazu na wielu monitorach, wybrano rozszerzanie obrazu – pulpit systemowy został rozciągnięty na dwa wyświetlacze. Po przeciągnięciu okna z wyświetlaną wizualizacją poza ramę monitora i po włączeniu trybu zmaksymalizowanego dla okna (rys. 3.1), rozpoczęto sprawdzenie poprawności wstrzykiwania obrazu.

Przy pomocy programu Code Composer Studio firmy Texas Instruments oraz protokołu JTAG (ang. *Joint Test Action Group*) dokonano debugowania całego procesu. Kamera została połączona z komputerem za pomocą USB (ang. *Universal Serial Bus*). Pakiet Code Composer

Rysunek 3.1. *Zrzut ekranu z otwartą wizualizacją środowiska CarMaker w celu wstrzyknięcia jej do kamery*, źródło: opracowanie własne

Studio, oprócz dużych możliwości w zakresie debugowania aplikacji zaimplementowanych na mikrokontrolerach, umożliwia też programowanie takich układów.

Co ważne z punktu widzenia powyższej implementacji, Code Composer Studio dawał możliwość zatrzaśnięcia ramki przed konwersją dokonywaną przez jednostkę obliczeniową (rys. 3.2a), kiedy obraz był 12-bitowy oraz po konwersji (rys. 3.2b), gdy przy pomocy dołączanego histogramu otrzymano obraz 8-bitowy.



(a) *Obraz 12-bitowy* (b) *Obraz 8-bitowy*

Rysunek 3.2. *Wstrzykiwana ramka obrazu przed i po mapowaniu wartości 12-bitowych na wartości 8-bitowe, źródło: opracowanie własne*

3.1.2. Wyniki testów

Jak można zauważyć na zrzutach ekranu, załączonych we wcześniejszej sekcji, wstrzykiwanie obrazu zostało wykonane poprawnie. Karta FMC-HDMI dobrze odbiera obraz przesyłany z komputera i przesyła go do FPGA. Tam zostaje obliczony i dołączony histogram, na podstawie którego jednostka obliczeniowa dokonuje konwersji wartości pikseli 12-bitowych na wartości 8-bitowe.

3.2. Testowanie kamery ADAS

W celu przetestowania kamery zaawansowanego systemu wspomagania kierowcy wykorzystano wcześniej zaimplementowaną logikę autonomicznego hamowania pojazdu. Podczas ewaluacji skorzystano ze wszystkich elementów stworzonego systemu.

3.2.1. Przeprowadzenie testów

Po połączeniu ze sobą wszystkich elementów sprzętowych, uruchomiono platformę dSPACE oraz program Control Desk, w celu przeprowadzenia symulacji. Po wykonaniu tego zadania uruchomiono program CarMaker i połączono go z symulacją na dSPACE. Po ustanowieniu komunikacji, odtwarzano kolejno filmy z wcześniej przygotowanymi jazdy testowymi i notowano uzyskane wyniki. Za zaliczone uznawano jedynie te sytuacje, kiedy auto nie wjechało w przeszkodę, bądź znacząco zredukowało swoją prędkość.

Testy systemu przygotowano w oparciu o protokół ewaluacyjny agencji Euro NCAP [23]. Wykonano wizualizacje odwzorowujące zarówno miasto (rys. 3.3a), jaki i tereny podmiejskie (rys. 3.3b). Dla terytorium miejskiego rozpatrzono scenariusz, w którym samochód poprzedzający testowany pojazd stoi w miejscu, a testowany pojazd porusza się z prędkością 10 – 50 km/h. Dla terenów podmiejskich rozpatrzono poniższe sytuacje:

– test pierwszy – samochód poprzedzający testowany pojazd stoi w miejscu, testowany pojazd porusza się z prędkością 30 – 80 km/h,

– test drugi – samochód poprzedzający testowany pojazd porusza się z prędkością 20 km/h, testowany pojazd porusza się z prędkością 20 – 50 km/h,

– test trzeci – samochód poprzedzający testowany pojazd porusza się z prędkością 50 km/h, po czym zwalnia z przyspieszeniem wynoszącym -2 lub -6 m/s^2, testowany pojazd porusza się z prędkością 50 km/h, a odległość początkowa pomiędzy samochodami wynosi 12 m,

– test czwarty – samochód poprzedzający testowany pojazd porusza się z prędkością 50 km/h, po czym zwalnia z przyspieszeniem wynoszącym -2 lub –6 m/s^2, testowany pojazd porusza się z prędkością 50 km/h, a odległość początkowa pomiędzy samochodami wynosi 40 m.

Dla każdego z powyższych testów, ze zmiennymi prędkościami symulowanego samochodu, należy przyjąć rozdzielczość prędkości tego auta wynoszącą 5 km/h.

(a) *Miasto* (b) *Tereny podmiejskie*

Rysunek 3.3. *Wizualizacje wykonane w programie CarMaker*, źródło: opracowanie własne

3.2.2. Wyniki testów

Wyniki funkcjonalności testowanej zgodnie z normami określonymi w [23] zostały zebrane dla wszystkich rodzajów testów (Tabele 3.1 – 3.5).

Tabela 3.1. *Teren miejski*

prędkość samochodu [km/h]	**wynik testu**
10 – 45	zaliczony
45 – 50	zredukowano prędkość

Tabela 3.2. *Teren podmiejski, test pierwszy*

prędkość samochodu [km/h]	**wynik testu**
30 – 50	zaliczony
50 – 65	zredukowano prędkość
65 – 80	niezaliczony

Tabela 3.3. *Teren podmiejski, test drugi*

prędkość samochodu [km/h]	**wynik testu**
20 – 45	zaliczony
45 – 50	niezaliczony

Tabela 3.4. *Teren podmiejski, test trzeci*

przyspieszenie samochodu [m/s^2]	**wynik testu**
-2	redukcja prędkości
-6	niezaliczony

Tabela 3.5. *Teren podmiejski, test czwarty*

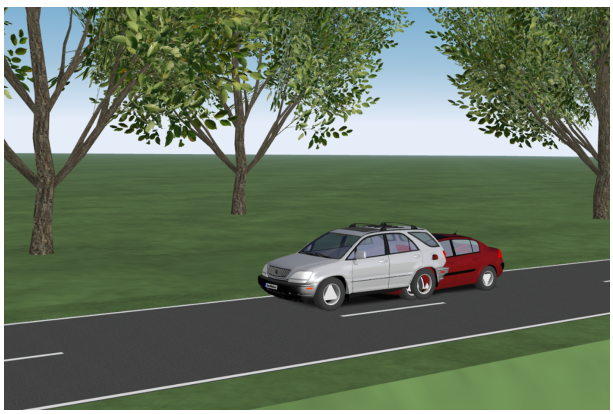
przyspieszenie samochodu [m/s^2]	**wynik testu**
-2	zaliczony
-6	zaliczony

Powyższe dane zestawiono z rezultatami odtworzonych scenariuszy testowych zarejestrowanych w warunkach rzeczywistych z wykorzystaniem takiej samej kamery. Można było zauważyć, że wyniki otrzymane w ramach symulacji *Hardware-in-the-Loop* z zamkniętym sprzężeniem zwrotnym (rys. 3.4) nie odbiegają od wyników referencyjnych, różnice sięgają zaś $5km/h$.

Jak można zauważyć na podstawie tych danych, sprzężenie zwrotne zostało poprawnie zamknięte, stworzony system reaguje poprawnie, ze sporą dozą realizmu.

(a) $30km/h$

(b) $45km/h$



(c) $80km/h$

Rysunek 3.4. *Zaliczone i niezaliczone scenariusze ewaluacyjne na drogach podmiejskich, test pierwszy*, źródło: opracowanie własne

4. Podsumowanie

W niniejszej pracy autor podjął się stworzenia systemu testowego dla kamery zaawansowanego wspomagania kierowcy z użyciem metodologii *Hardware-in-the-Loop*. Z całą pewnością można stwierdzić, że zadanie to zostało zrealizowane, a wszystkie założenia spełnione.

Dokonano przeglądu aktualnej literatury oraz rozwiązań związanych z tematyką testowania systemów wizyjnych z wykorzystaniem metodologii *Hardware-in-the-Loop*. Następnie stworzono model docelowego systemu w środowisku MATLAB oraz Simulink. Wygenerowano kod, po czym wgrano go na platformę DS1006. Stworzono aplikację do komunikacji z kamerą, wykorzystując magistralę CAN oraz protokół sieciowy UDP. Wykonano wizualizację drogi z wykorzystaniem środowiska CarMaker, która była przesyłana do kamery. Stworzono pełne środowisko testowe, umożliwiające przetestowanie w kamerze funkcjonalności automatycznego hamowania (AEB) zgodnie z normami NCAP. Wykonanano testy. Wykorzystano układ FPGA w celu zastąpienia sensora wizyjnego w kamerze, tak aby możliwe było wstrzykiwanie generowanego obrazu.

Odnosząc się do otrzymanych wyników można zauważyć, że zaprezentowany w tej pracy system działa poprawnie. Zostało to potwierdzone przez porównanie otrzymanych danych z danymi referencyjnymi. Jest to o tyle cenne, że system może być rozpatrywany jako podstawa do testowania czynnych układów samochodowych przed zamontowaniem ich w pojazdach.

Jak zostało to wiele razy podkreślone w niniejszej pracy, współczesne systemy bezpieczeństwa czynnego montowane w samochodach opierają swoje działanie nie tylko na danych otrzymanych z kamer, ale także o informacje wysyłane przez urządzenia takie jak radar oraz LIDAR. Aby stworzony system jeszcze skuteczniej odwzorowywał warunki rzeczywiste, należy dołączyć te urządzenia do układu, czyniąc go w ten sposób niezaprzeczalnie bardziej kompleksowym. Wydaje się, że jest to najbardziej słuszna droga do rozwoju dalszych badań. Tym bardziej, że oba wymienione wyżej urządzenia używają podobnych standardów komunikacji jak kamery – mowa tu przede wszystkim o magistrali CAN.

Nie oznacza to jednak wcale, że system w obecnym kształcie jest gotowy na współdziałanie z innymi urządzeniami. Aktualnie duży nacisk kładzie się na implementację nowszych standardów komunikacji do pojazdów, między innymi protokołu *FlexRay*. Ze względu choćby na fakt braku realizacji obsługi tej technologii w stworzonym środowisku testowym, dołączanie innych układów do projektu musi zostać poprzedzone odpowiednimi badaniami, a następnie implementacją takich standardów.

Bibliografia

[1] J. M. Borrmann, F. Haxel; D. Nienhüser, A. Viehl, J. M. Zöllner, O.Bringmann i W. Rosenstiel *"STELLaR - A case-study on SysTEmaticaLLy embedding a Traffic Light Recognition"* w *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, strony 1258–1265, Qingdao, 2014

[2] C. Galko, R. Rossi, X. Savatier *"Vehicle-Hardware-In-The-Loop system for ADAS prototyping and validation"* w *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, strony 329–334, Samos, 2014

[3] M. Gorgoń, M. Komorkiewicz, T. Kryjak, P. Skruch i K. Turek *"FPGA-Based Hardware-in-the-Loop Environment Using Video Injection Concept for Camera-Based Systems in Automotive Application"* w *The 2016 conference on Design & Architectures for Signal & Image Processing*, strony 179—185, Rennes, 2016

[4] M. Nentwig i M. Stamminger *"A method for the reproduction of vehicle test drives for the simulation based evaluation of image processing algorithms"* w *13th International IEEE Conference on Intelligent Transportation Systems*, strony 1307–1312, Funchal, 2010

[5] M. Nentwig i M. Stamminger *"Hardware-in-the-loop testing of computer vision based driver assistance systems"* w *2011 IEEE Intelligent Vehicles Symposium (IV)*, strony 339–344, Baden-Baden, 2011

[6] Ö. Tunçer, L. Güvenç, F. Coşkun i E. Karsligil *"Vision Based Lane Keeping Assistance Control Triggered by a Driver Inattention Monitor"* w *2010 IEEE International Conference on Systems, Man and Cybernetics*, strony 289–297, Istanbuł, 2010

[7] Dane firmy Bosch, zaawansowane systemy wspomagania kierowcy, *http://products.bosch-mobility-solutions.com/en/de/driving_comfort/*

[8] Dane firmy Nvidia, zaawansowane systemy wspomagania kierowcy, *http://www.nvidia.com/object/advanced-driver-assistance-systems.html*

[9] Dane firmy ON Semiconductor, nota katalogowa sensora wizyjnego AR0132AT, *http://www.mouser.com/ds/2/308/AR0132AT-D-888230.pdf*

[10] Dane firmy Analog Devices, nota katalogowa procesora ADSP-BF609, *http://www.analog.com/media/en/technical-documentation/application-notes/EE358.pdf*

[11] Dane firmy dSPACE GmbH, nota katalogowa procesora DS1006, *DS1006 Processor Board Features*, 2014

[12] Dane firmy dSPACE GmbH, opis obsługi CAN z wykorzystaniem karty DS4302 i Simulinka, *DS4302 CAN Interface Board RTI Reference*, 2014

[13] Dane firmy dSPACE GmbH, opis obsługi UDP z wykorzystaniem karty DS4121 i Simulinka, *RTI Ethernet (UDP) Blockset Reference*, 2014

[14] Dane firmy dSPACE GmbH, opis wykorzystywania oprogramowania Simulink w celu modelowania układów na platformę dSPACE, *Simulink Modeling Guide*, 2014

[15] Dane firmy Xilinx, nota katalogowa produktu System-on-a-chip ZC706, *https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html*

[16] Dane firmy Digilent, nota katalogowa produktu FMC-HDMI, *http://store.digilentinc.com/fmc-hdmi-dual-hdmi-input-expansion-card/*

[17] Dane Międzynarodowej Organizacji Normalizacyjnej, ISO 11898-2:2016, *https://www.iso.org/obp/ui/#iso:std:iso:11898:-2:ed-2:v1:en*

[18] Dane firmy IPG Automotive GmbH, *CarMaker Programmer's Guide Version 5.1*, 2016

[19] Dane firmy IPG Automotive GmbH, *CarMaker User's Guide Version 5.1*, 2016

[20] Dane organizacji Wikipedia, opis protokołu UDP, *https://en.wikipedia.org/wiki/UDP*

[21] Dane Międzynarodowej Organizacji Normalizacyjnej, ISO 7498-1:1994, *http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip*

[22] Dane firmy Mathworks, informacje na temat *solverów* pakietu Simulink, *https://www.mathworks.com/help/simulink/ug/types-of-solvers.html*

[23] Dane organizacji Euro NCAP, sposób przeprowadzania testów układów AEB, *TEST PROTOCOL – AEB systems*, *http://www.euroncap.com/en/for-engineers/protocols/safety-assist/*, 2015

Dodatki

Dodatek A – spis zawartości płyty CD

Na płycie CD, która jest dołączona do niniejszej pracy znajduje się:

– tekst pracy inżynierskiej w formacie PDF.

Dodatek B – wykaz elementów stworzonego systemu

Elementy omawianego systemu stworzone przy pomocy pakietu MATLAB i Simulink:

– *main_program.slx* – główny model środowiska ewaluacyjnego komunikujący się z testowaną kamerą, platformą dSPACE i aplikacją CarMaker,

– *generate_frames.m* – skrypt generujący 12-bitowe ramki obrazu do pliku w formacie RAW,

– *calculate_histogram.m* – skrypt wczytujący wcześniej wygenerowane ramki obrazu i obliczający dla nich histogram,

– *generate_coe.m* – skrypt obliczający odpowiadającą wartość z zakresu 0 – 244 dla liczb $0 – 2^{12}$ i zapisujący je w kolejności rosnącej do pliku COE.

Elementy omawianego systemu stworzone przy pomocy programu Vivado:

– *get_frame.v* – moduł wczytujący przygotowane ramki z pliku RAW i generujący sygnały wideo,

– *save_histogram.v* – moduł odczytujący obliczony histogram z pamięci i zapisujący go do pliku tekstowego,

– *calculate_histogram.v* – moduł obliczający histogram dla wczytanej ramki obrazu,

– *pixel_position.m* – moduł obliczający aktualną pozycję piksela we wczytanej ramce obrazu.

Pozostałe elementy omawianego systemu:

– panel operatorski programu Control Desk kontrolujący zachowanie opracowanego systemu,

– symulacje wykonane w programie CarMaker, odpowiadające zróżnicowanym scenariuszom testowym.