MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF MECHATRONICS ENGINEERING

A COST EFFECTIVE SOLUTION FOR THE REAL TIME SIMULATION OF MERCEDES-BENZ'S KEYLESS-GO SYSTEM'S DOOR HANDLE ON A HARDWARE-IN-THE-LOOP-PLATFORM

UNDERGRADUATE PROJECT DISSERTATION

Submitted by

Siddhant Mahapatra 140929350
siddhant.mahapatra@learner.manipal.edu +91 9986945444

Under the guidance of
Mr. Adit Gupta
RDI/CEE
Mercedes-Benz Research and Development India Pvt. Ltd.

Mercedes-Benz
The best or nothing.

Acknowledgements

The internship opportunity I had with **Mercedes-Benz Research and Development India Pvt. Ltd.** was a great chance for my learning and professional development. Therefore, I consider myself as a very fortunate individual as I was provided with an opportunity to be a part of it. I am also grateful for having a chance to meet so many wonderful people and professionals who led me though this internship period. I have taken efforts in this project. However, it would not have been possible without the kind support and help I received from many individuals in this organization. I would like to extend my sincere thanks to all of them.

Bearing in mind previous I am using this opportunity to express my deepest gratitude and special thanks to my Mentor and Guide, **Mr. Adit Gupta**, KGCU Testing Team Lead, RDI/CEE, who in spite of being extraordinarily busy with his duties, gave me such attention and time to hear, guide and keep me on the correct path and allowing me to carry out my project at the organization and extending during the training.

I express my deepest thanks to my Manager, **Mrs. Sindhu Murthy**, for taking part in useful decision & giving necessary advices and guidance and arranged all facilities to make life easier. I choose this moment to acknowledge her contribution gratefully.

My thanks and appreciations also go to my colleagues who have willingly helped me out with their abilities.

I perceive this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain desired career objectives.

Abstract

*This work utilizes a state-of-the-art and a new-age FPGA technology developed by Intel-Altera, to develop a cost effective solution for the real time simulation of Mercedes-Benz's Keyless-Go system's Door Handle on a Hardware-in-the-Loop Platform. Unlike most car manufacturers, Mercedes-Benz uses automotive technology and protocols unique to itself. This work revolves around one such protocol used by the door handle to communicate with the Keyless-Go ECU. In a Hardware-in-the-Loop testing environment, the Keyless-Go ECU receives the protocol generated by a simulation card, instead of a physical door handle. The simulation card generates the protocol based on the test inputs provided by the test engineer, to test the response of the ECU when exposed to a plethora of complex scenarios based on the test requirements and specifications. In this work a FPGA based board is used as a simulation card, due to its ability to produce highly precise signals with frequencies in the sub-*µs *range.*

In a gist, a FPGA based development board acts as a real time interface between the HIL system and the Keyless-Go ECU, to simulate the functionalities of a door handle. This document craftily and subtly unfolds from giving an account of Hardware-in-the-Loop testing systems to Intel's MAX 10 FPGA to Mercedes-Benz's Keyless-Go system to RS232 UART based FPGA design and implementation, and finally to the Single-Port Triple Speed Gigabit Ethernet Communication aspect.

In this work, RS232 UART communication based MAX 10 FPGA design has been developed as a part of the solution to generate the door handle telegram protocol with controllable parameters. A more efficient Single-Port Triple Speed Gigabit Ethernet Communication aspect has been tested for full duplex communication between the board and the PC, but has not been integrated with the telegram generation design model as part of this work.

Contents

List of figures

List of tables

A COST EFFECTIVE SOLUTION FOR THE REAL TIME SIMULATION OF MERCEDES-BENZ'S KEYLESS-GO SYSTEM'S DOOR HANDLE ON A HARDWARE-IN-THE-LOOP-PLATFORM

Chapter 1. INTRODUCTION

1.1 *Area of work* :

This work revolves around HIL systems designed custom to the requirements of Mercedes Benz. The Keyless Go ECU, one of the several ECU's installed in a Mercedes-Benz vehicle, is required to be tested in a variety of scenarios on a Hardware-in-the-loop platform, to attest it's functionalities and it's robustness. The door handle is one of the components that communicates with the KG-ECU. This work attempts to come up with a cost effective solution for the real time simulation of the Door Handle on a KG-ECU installed HIL platform.

Hardware-in-the-loop simulation (HIL) has become an integral component in the electronic development process for testing control functions in the automotive industry. HIL simulation involves operating mechatronic systems, particularly electronic control units (ECUs), in a closed loop with components that are simulated in real time to test them intensively in this virtual environment.

The importance of electronics and software has grown considerably in all areas of everyday life over the last several years. This trend can clearly be seen in mechanical engineering (mechatronics), as well as in automotive, aerospace, and home entertainment, where today's televisions and games consoles have more computing power than the Apollo rockets 40 years ago. Virtually all innovations in the field of automotive engineering are now based on new or further developed electronics. This applies to the latest engine technologies such as start-stop and hybrid drives, as well as passive and active safety systems (airbags, ESP, pre-crash), driver assistance systems (ACC, lane keeping systems, night vision), and infotainment systems (telephone, Internet, DVD player, TV, etc.). Today's top-of-the-range cars include up to 70 ECUs that are connected with one another via various bus systems. The complexity of this networked structure of functions, combined with an enormous number of

different vehicle versions (different engines and transmissions, different options, country-specific versions, etc.), is a major challenge in terms of developing and testing vehicle ECUs. Hence, intensive work is currently being done to define manufacturer-independent standards for function modules and architectures, with the goal of separating the hardware, the operating system, and the application layers (AUTOSAR [AS09], JASPAR, etc.). *Hardware-in-the-loop technology (HIL)* has become an integral part of the electronics development process of car manufacturers and automotive suppliers for testing single ECUs as well as ECU networks.

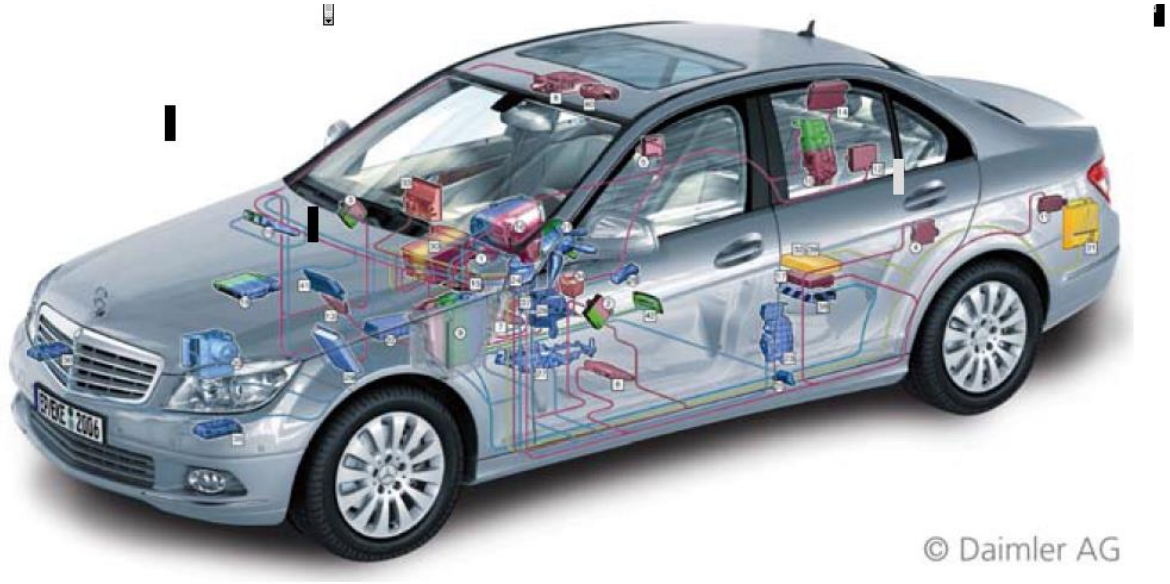


Figure 1.1: ECU network in a typical Mercedes-Benz vehicle

1.2 Present day scenario:

Hardware-in-the-loop simulation is a method used in the product development cycle in which one or more real components interact with components that are simulated in real time (dynamic models). The part of the system that is not simulated can consist of real devices, machines, or mechanical test benches. Nowadays, however, the term is mainly used to refer to a real system that consists of one or more ECUs, controllers, or intelligent mechatronic components for which a virtual environment is *simulated* electrically and

dynamically. The interactions between the real and the simulated subsystems make tough realtime demands on the latter. The simulated subsystem has to perform the following actions within one simulation step:

- Read in the measurement signals (actuator control by the ECU)
- Calculate and perform numeric integration (simulate the entire dynamic model of a real system)
- Output the results (sensor simulation for the ECU).

The result is a *closed loop* between the real controller and the simulated plant. Failure to meet real-time conditions can result in unstable simulation and even damage to the real technical device.

Figure 2 shows a signal flow that illustrates this structure. In the real system, the real ECU is integrated in the real vehicle (left); in HIL simulation, it is connected to the HIL simulator via electrical interfaces (center and right).

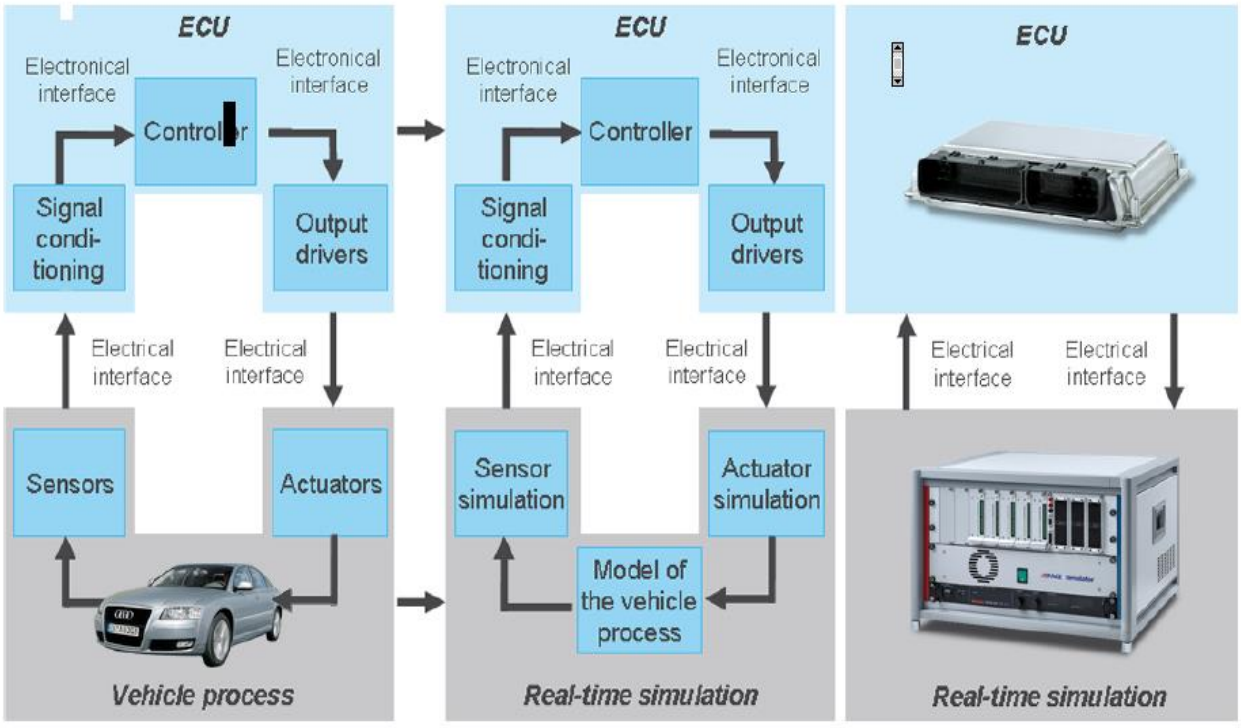


Figure 1.2: *Signal flows in a real system and in HIL simulation.*

1.3 Motivation :

- The major aims of HIL simulation are to enhance product quality, to master complexity, and to reduce development costs. This last aim is achieved by moving function tests and diagnostics tests from tests drives or test bench experiments to the HIL laboratory. The result is that the number of expensive prototype vehicles and time spent at the test bench are considerably reduced. HIL tests can be reproduced as often as required and also automated. It is now common practice to completely automatically run, evaluate, and document tests overnight or on weekends. This allows the test operators to concentrate on assessing tests, implementing new tests or test scripts, and adjusting tests that failed (for example, due to ECU errors). Automation provides far greater test coverage than manual tests, and this enhances quality and maturity.

- Frequently, HIL simulators are also used to perform tests that would not be possible at all on a real system (for example, because no prototype vehicles are available yet), or that would be critical for the components (danger of damaging or destroying prototypes) or even dangerous for the test driver (for example, if there is a sensor error at top speed on a sharp bend). Thus, HIL simulation makes it possible to give an ECU an environment that is so realistic that the ECU's diagnostics system does not detect any inconsistencies in the electrical signal quality or the dynamic behavior of the plant (no fault memory entry by the diagnostics). In this situation, any tests that would typically be performed in or with the vehicle (or on a test bench) can also be performed on the HIL simulator. This is the basis for inducing error situations intentionally. Diagnostic functions (with short circuits, functional faults or communication errors), fail-safe levels and limp-home mode can be tested in this way.

- The door handle simulation will be conducted on the HIL platform using a special purpose interface card. This card will be designed and

programmed to act as an interface between the HIL system and the ECU under test. The entire project revolves around understanding the HIL system and the Keyless-Go system design of Mercedes Benz and the subsequent low cost solution to design and implement an interface card.

Chapter 2: LITERATURE REVIEW

2.1 HIL Systems: Basics and Components

2.1.1 Hardware Components of the HIL System

- **Host PC**: An off-the-shelf Windows® PC is generally used as the user interface to the real-time processor (experiment software) and for executing the modeling and implementation software. To ensure high data rates and low latencies in communication with the real-time hardware, specially optimized interface cards are used, and sometimes also Gigabit Ethernet. It is the development of this interface card that is the primary focus of this project.

- **Real-time processor system:** Standard server PCs with a real-time operating system have now almost caught up with special real-time processor boards in terms of pure processor performance, which is particularly important for model calculation However, connection to the I/O cards is problematic, as server PCs are frequently not optimized for this. Thus, processor boards with optimized I/O interfaces are used for high-end applications to achieve an optimum overall system consisting of a processor and I/O.

- **I/O boards and signal conditioning:** Automotive HIL applications require boards for simple analog and digital signals, and also special boards for fast, engine- angle-synchronous signals (see section 3.1.1) for fast sensor simulation (wheel speeds, knock signals) or

actuator measurement. Some of these I/O boards already possess the necessary protection circuits and signal adaptations to specific automotive electrical system voltages (12 V, 24 V, etc.); otherwise, separate signal conditioning is used (e.g. for current interfaces, lambda probes, etc.).

- **Bus systems:** The ECUs form networks and communicate via various bus systems in the vehicle (CAN, LIN, FlexRay, MOST). When parts of the network are operated in HIL, the bus behavior of all the missing ECUs has to be appropriately simulated. Thus, the HIL simulator must be able to *generate* messages (this is called *restbus simulation*) and to *read* all the messages coming from the ECU. Here too, special I/O boards that this project is revolving around, are used, frequently with intelligent subprocessors or FPGAs and suitable bus transceivers.

- **Electrical loads and load simulation:** ECUs control electrical actuators (called *loads*), e.g. valves, electric motors, relays, current-controlled actuators and piezo injectors. Either the real loads or electrically equivalent circuits can be used in an HIL system. The ECU's diagnostic system monitors these actuators so that it can take appropriate action if a fault occurs (short circuit, open circuit) or at least inform the driver. Quite often, it is sufficient to connect a substitute resistance to the ECU as a load. However, if the load itself has dynamic behavior (variable resistance such as in a headlamp) and the ECU performs diagnostics on this, either a real load is integrated into the HIL system or an electronic (= dynamic) load simulation controlled by the real-time system is used.

- **Electrical fault simulation:** To generate the electrical fault states mentioned above, failure simulation units are integrated into the HIL system. These can simulate hard short circuits and open circuts, and also leakage resistance and loose contacts. Relays or semiconductor

switches are used for this depending on requirements. The failure system is programmed and activated either interactively or via test automation.

- **Real components:** To treat ECUs realistically, real components are necessary. These might be loads that are not easy to simulate, or that can only be simulated with a lot of effort, and sometimes smaller setups or even complex test benches are used.

- **Power supply:** Simulating the vehicle electrical system and the battery vehicle also requires power supplies whose voltages can be specified dynamically by the simulator. This is especially important for undervoltage and overvoltage tests (e.g. in jumpstarts by trucks) and for simulating voltage drops when starting the engine.

2.1.2 Software Components of the HIL System

The software components for an HIL system are subdivided into the operating software (PC), the real-time software (real-time system) and the dynamic plant model.

- *Operating Software (Implementation and Experiment Software):*

MATLAB®/Simulink® (ML/SL) and Dymola have become established as a quasi standard for function development in the automobile industry. It therefore makes sense to use ML/SL/DM to describe both the dynamic behavior of the plant, and the definition and configuration of the I/O of a simulator system.
The entire function and instrumentation code for the real-time system is then generated by autocoding.
Interactive operation of the HIL system requires configurable GUIs with a wide range of different views, which can be adapted flexibly to specific projects. In addition, 3-D visualization is indispensable,

especially for vehicle dynamics applications, as complex driving maneuvers cannot be suitably evaluated by recorded time behaviors alone.

However, the greatest benefit is derived from HIL systems if test runs are not interactive but automated. HIL tests frequently run overnight or on weekends, and all that the operators need to do during the day is to examine the generated test reports, repeat failed tests, and implement new tests. The time saved by automation means that when used systematically, HIL systems pay for themselves in only a short time, typically less than a year.

Script languages such as VBA, MATLAB and Python are frequently used for automation. However, there are also test tools on the market that allow test implementation to be performed in graphical form. As in Simulink, the test creator can put together a test from single test steps or whole test sequences from libraries, and "program" parallel as well as serial test sequences.

- *Real-Time Software:*

  To meet HIL simulation's tough real-time requirements, real-time operating systems (or operating system kernels) designed for maximum I/O throughput and minimum I/O latency are used. Multitasking, multirate simulation (different step sizes within the model) and multiprocessor systems must also be supported. These requirements are fulfilled only by special real-time kernels (e.g. [DS09]).

  However, it is now also possible to set up simple HIL systems based on standard real-time operating systems such as QNX and LINUX-RT.

- *Dynamic Models:*

  To close the control loop between the ECU and simulator inputs/outputs, dynamic plant models are needed. These must provide a sufficiently good representation of the system to be controlled. The model quality must be so good that the ECU does not detect any inconsistencies or implausibilities. The real-time capability of the model is another decisive criterion for HIL use. Very detailed models (including FEM approaches and 3-D flow simulation) are used in engine and chassis development, and such models are far from being real-time-capable. For HIL operation, therefore, specific models and model approaches are used that are real-time-capable and sufficiently precise with regard to the ECUs' sensors.

2.2 HIL Application Areas

HIL systems are employed for testing *single* ECUs and ECU *networks*. Singl ECUs are frequently tested thoroughly by their supplier befor they are delivered to the car maker - the original equiment manufacturer (OEM) - for integration. The OEM then performs comprehensive network tests on all the ECUs together.

2.3 Interface Card for Simulation

A basic difference between electric machines and other vehicle components is that the machine is controlled at a very high clock rate (typ. 10-20 KHz). This makes high demands on the HIL simulation. I/O sampling rates in the millisecond range and mean value models, which are used for combustion engines, are not sufficient for fast current and position control. Special FPGA-based hardware is needed for this. Figure below shows a generic FPGA board that can be equipped with application- specific modules, e.g. for measuring the PWM control signal or for simulating incremental encoders or resolvers (clock

rate 40 MHz). Moreover, small model parts can be simulated directly on the FPGA to achieve simulation step sizes in the sub-µs range.

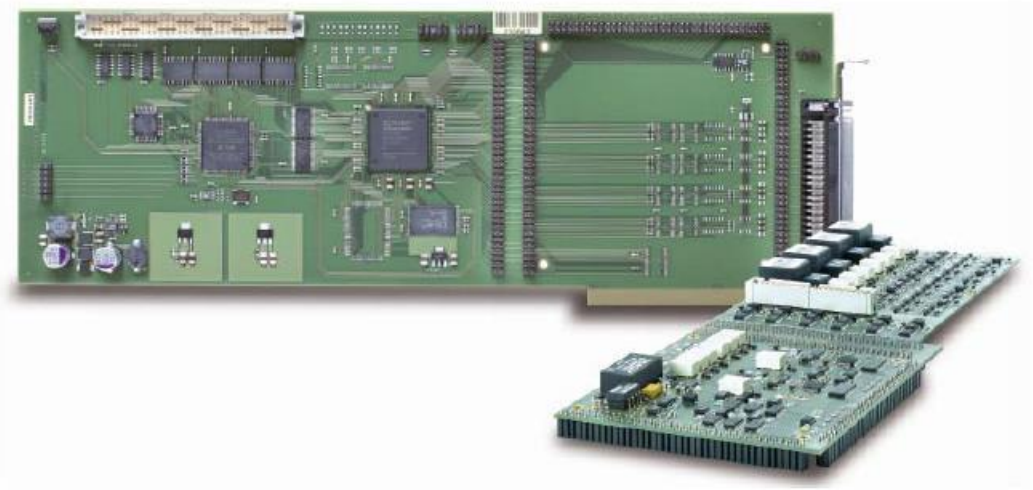


Figure 2.1: *Generic FPGA board for the measurement, simulation and generation of signals in the sub-µs range*

2.4 Testing the ECU Network

The vehicle manufacturers are responsible for the *vehicle* as an overall system, and for ensuring that ECUs from different suppliers work together. Thus, network testing on all the ECUs is of major importance in vehicle development.

Figure 2.2: *Network simulator for testing a vehicle*

2.5 Special HIL Solutions

2.5.1 Integrated Sensors and Actuators

There is generally an electrical interface between the ECU and the (simulated) actuators and sensors. In this case the wiring harness can simply be opened up and connected to the simulator. However, if the ECU has *integrated* sensors or actuators, the interfaces between the ECU and the HIL system are at different locations.

Integrated sensors and actuators are frequently used in transmission ECUs that are installed directly in the automatic transmission. For HIL simulation, the integrated sensors must be stimulated physically, e.g. inductive speed sensors are stimulated controlled coils, pressure membranes by strong lifting magnets, and temperature sensors by controlled hot plates.

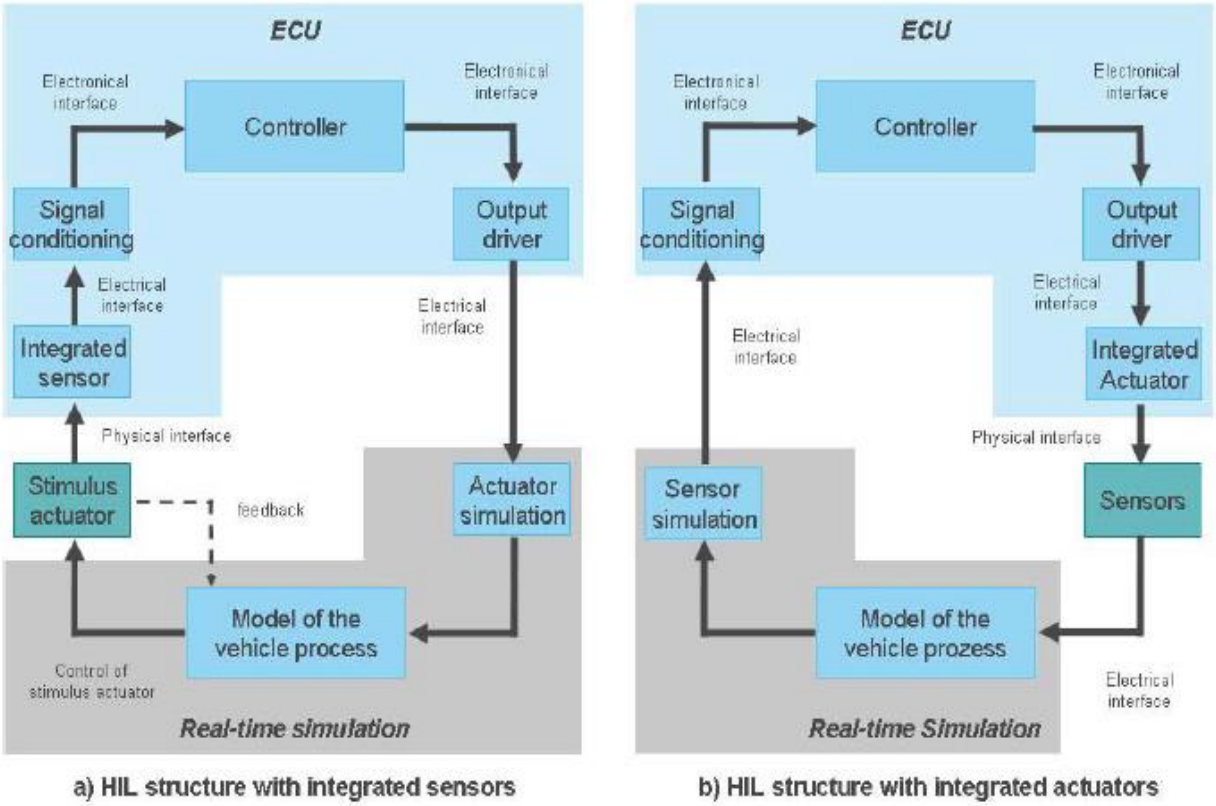


Figure 2.3: *Integrated sensors and actuators: electrical interfaces are replaced by physical interfaces*

2.5.2 Test Bench Integration

It is important to test an entire mechatronic subsystem such as a steering system or a chassis on the HIL simulator in addition to performing pure electronics tests. Thus, fatigue tests have to be performed in addition to function testing. Classic mechanical test benches are coupled with HIL simulators for this. Figure below shows such a test bench for a steering system. Test benches have also been coupled for testing a part of the drivetrain; for example, a real Formula One transmission was connected to a simulated engine and a simulated drivetrain. The coupling of HIL simulation with mechanical axle test benches for function development and verification for active chassis has also been published.

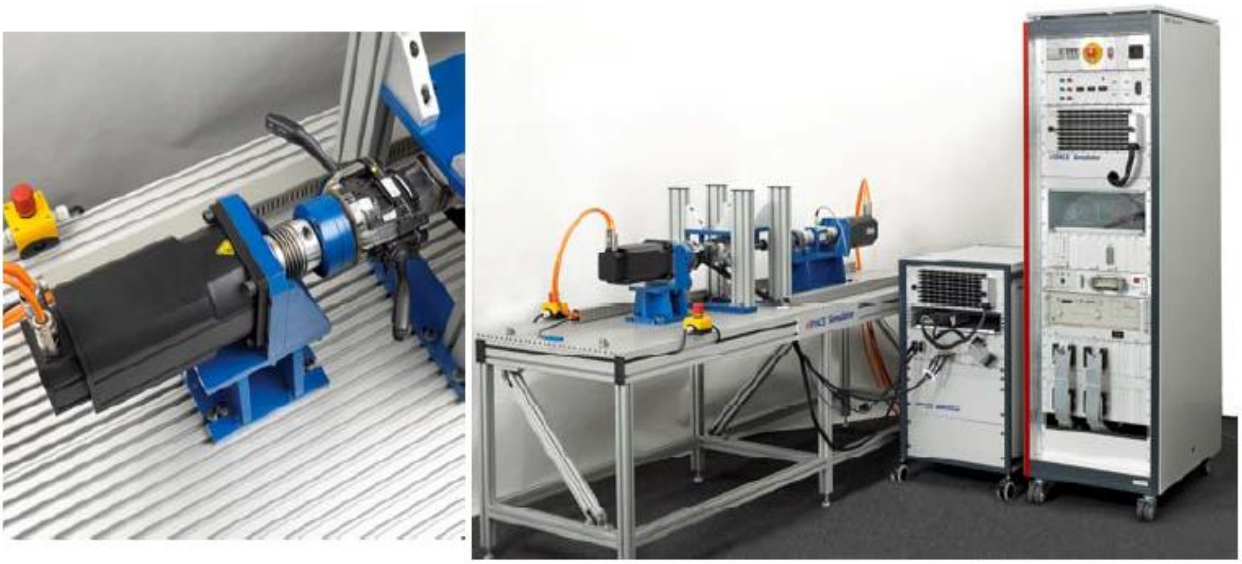


Figure 2.4: *Example of an HIL simulator for testing electrical steering systems. The mechanical loads that are applied to the steering system on the test bench are the same as those that occur in the vehicle.*

2.6 HIL test platform

- There is an increasing requirement to develop vehicles within a shorter time frame. Achieving a rapid vehicle development programme reduces cost and maximizes the impact of a new vehicle in an extremely competitive market place. The increased use of

electronic control systems requires that a smarter and more efficient means of development and testing be deployed. Hardware in the Loop (HIL) systems are an important technology enabling integrated electronic control systems to be developed within the time frame defined by the vehicle development process and with a reduced reliance on expensive prototype vehicles. Such HIL systems provide a flexible test environment that can be updated and modified as the vehicle and corresponding systems develop throughout the duration of a vehicle programme.

- An HIL test platform allows the integration of real control system hardware, a controller under test, with a simulation model representing the real environment in which the controller under test operates. The simulation model, commonly called a plant model, is designed to be executed as compiled code on a target microprocessor that forms a part of specially designed HIL hardware. The HIL system also includes signal conditioning hardware that provides an interface between the controller under test and the virtual world of the plant model. Figure below shows the HIL system used for testing control system software. The system shows a PC that has a hardware communication device linking it to the HIL hardware. An interface between a development tool and the system under test is also installed to enable calibration and system data analysis to be carried out. Also required is HIL user interface software, test automation software and model development software, such as Dymola/Matlab/Simulink, for plant model construction. The use of automated test software promotes the efficient use of HIL systems. It enables repeatable tests, normally carried out manually by test engineers, to be executed on the HIL system. Typical tests performed on an automotive ECU are short- and open-circuits on I/O that exercise ECU diagnostic software. In addition, automated tests are developed to verify the base functionality of the ECU. An important feature of test automation software is the ability to store

test results and present them in a readable format that is easily analyzed.

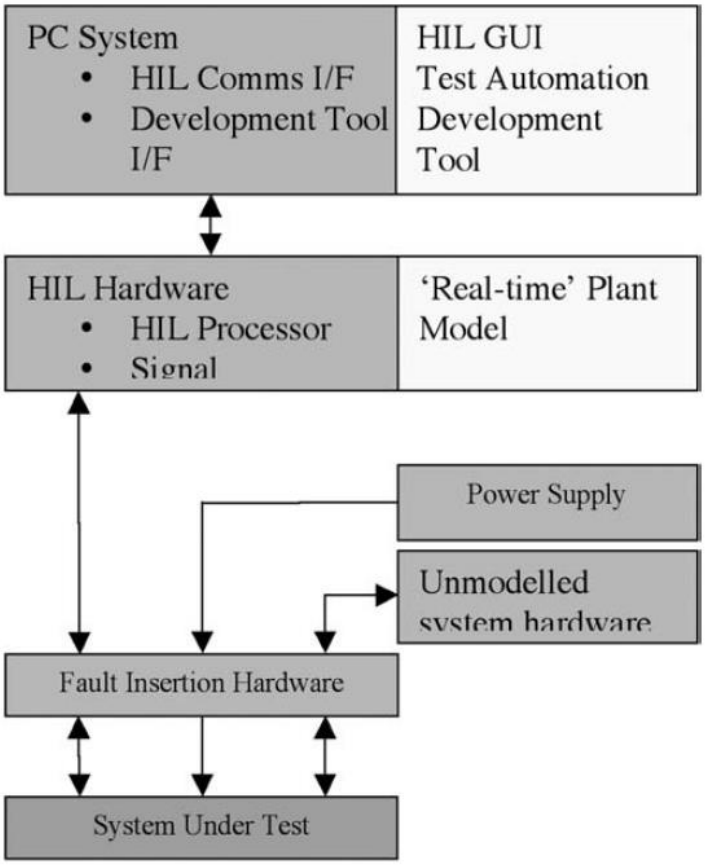


Figure 2.5: HIL test platform for ECU testing

2.6.1 V-cycle controller development process

- The approach largely adopted for control system development follows the stages defined by the V-cycle shown in the Figure below. This common approach places HIL testing on the right-hand side of the V and is a distinct phase of the control system development process.
-

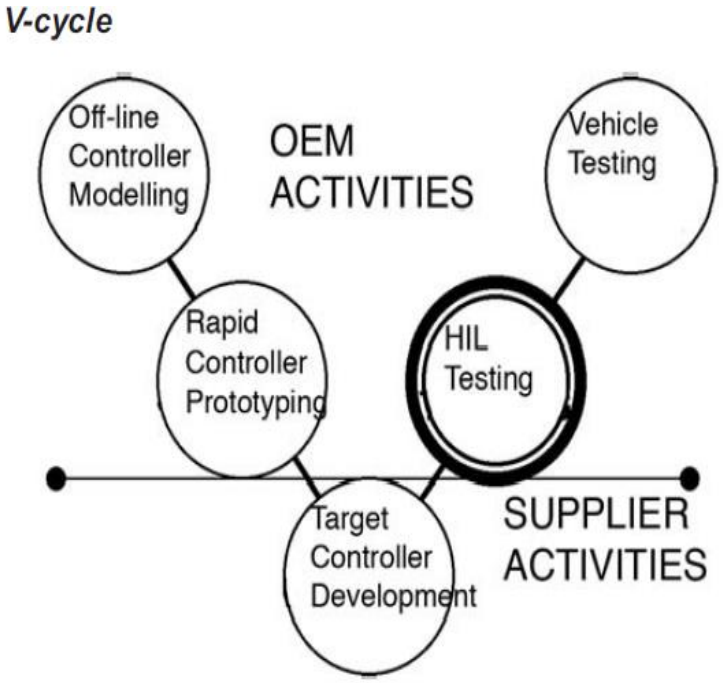


Figure 2.6: V-cycle controller development process

- Figure above illustrates the V-Cycle that defines a control system development process. The process begins by defining a set of functional requirements for a system from which a simulation model of the control system is developed. Following the development and testing of a simulation model, a rapid prototype phase may follow whereby the control system can be embedded into a prototype platform in a vehicle. In addition to providing an in-vehicle test platform, this phase also allows new systems to be demonstrated and evaluated. The construction and verification of a controller model is followed by the development of the embedded code and the target ECU. This phase is typically carried out by a supplier with specialist knowledge of a particular system. Following delivery of the ECU and software, HIL systems are used to carry out various forms of functional testing of the strategy and associated diagnostic routines. The types of testing well suited to being carried out using

HIL systems are those tests that are repetitive, such as testing for system failure modes. The final phase of the V-cycle is the use of vehicle testing for final verification of the software, hardware and its calibration.

2.6.2 Extended use of HIL for vehicle control system development and test

- The V-cycle described in the previous section shows the approach adopted for control system development that emphasizes the use of HIL testing on the right hand side of the V. This approach remains valid for testing many of the control systems fitted in vehicles today. However, the use of HIL is being extended to support other elements of the control system development cycle. In particular, this is the case with an increased effort to take a more holistic approach to engineering integrated electrical systems for vehicles. There is a benefit for vehicle manufacturers such as Mercedes-Benz to develop control systems independently of third-party suppliers. In addition to having ownership of the intellectual property that corresponds to a particular system, developing systems that differentiates their cars from those of competitors is important. Systems or features developed 'in-house', by engineers of the company, help to define the brand and this is important in a very competitive market. That is another reason behind the plan to design the interface card by me. To help achieve brand differentiation, a larger part of the control system software development is being undertaken by Mercedes Benz. With this effort being made to take a more holistic approach to engineering integrated electrical systems for vehicles, HIL systems are being used increasingly as a tool throughout the

development cycle. The natural first step for extending the use of HIL systems is to combine a number so-called ‘stand-alone’ HIL test platforms to create an integrated test environment.

2.7 Combined HIL-virtual test and development platform

- A combined HIL-virtual test platform has been built to support the development of an integrated body electronics system for a new vehicle. Taking a model-based approach to the design of control system software has allowed HIL technology to be used to test system integration prior to having physical ECU hardware available. The use of HIL systems for testing a combination of virtual and physical controllers allows more complete testing to be carried out earlier in the life of the vehicle development programme. Figure below illustrates the use of a combined HIL and virtual development and test platform applied to the phases of a model-based controller development process.

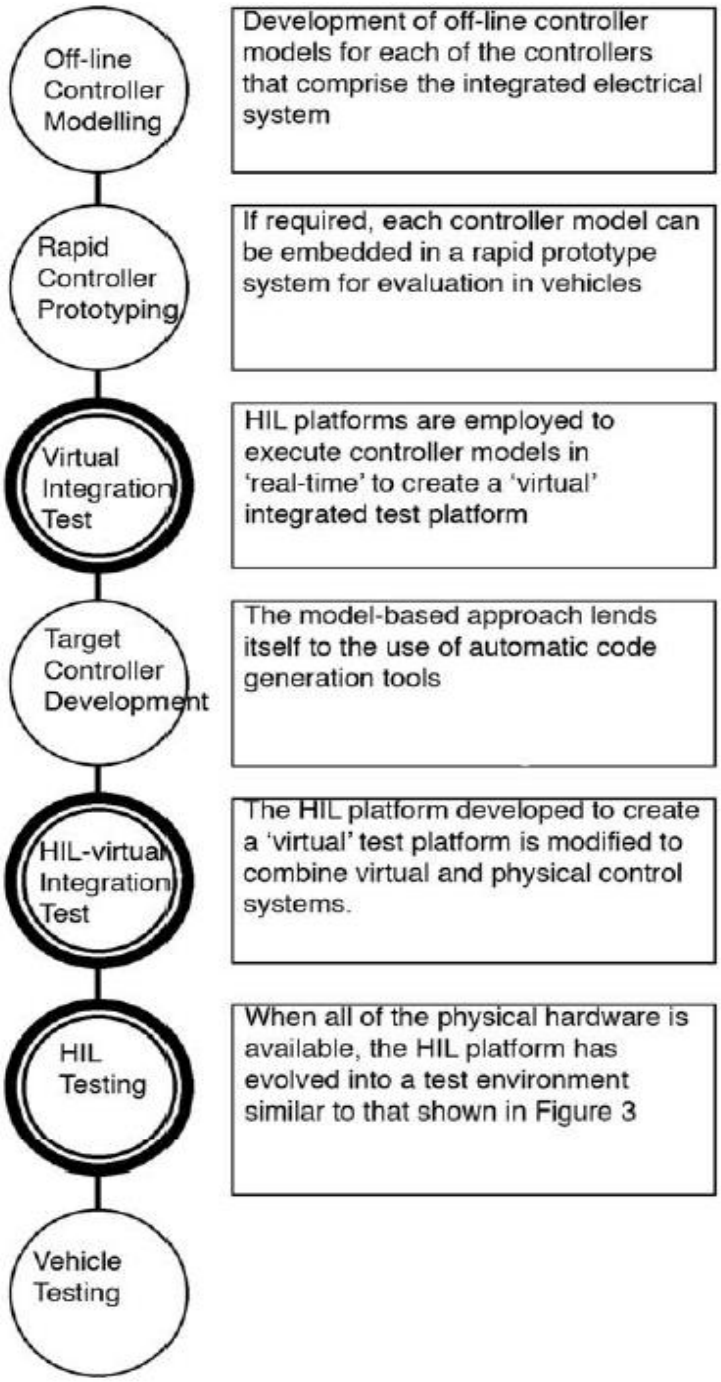


Figure 2.7: Phases of model-based controller development process

- The *first phase* is 'off-line controller modelling' in which a comprehensive set of control system requirements are translated into a control system functional model. In order to construct and test a control system model, it is useful to employ a suitable vehicle plant model to close

the control loop. Giving careful consideration to the specification and design of such a plant model during this initial phase helps to extend its use to form the basis of a plant model to support subsequent HIL testing phases.

- The *second phase* is 'rapid controller prototyping' during which the functionality of control system models is evaluated in engineering vehicles using specialist rapid prototyping hardware. This phase allows candidate control systems to be demonstrated to vehicle evaluation teams prior to production software being written.

- The *third phase* is 'virtual integration test'. This phase uses HIL platforms as a means of executing a combination of control system models in 'real-time' by linking them to a single vehicle plant model.

- The *fourth phase* is 'target controller development' during which control system models are translated into production ready software either by hand-coding, auto-code generation or a combination of both.

- The *fifth phase* is 'combined HIL-virtual integration test'. This phase requires the model that supports the virtual test platform to be modified to allow a combination of actual and virtual control systems to be linked. This enables real control systems to be tested as they become available.

- The *final phase* prior to vehicle testing is 'HIL test' that relies on having available all of the real control systems. The development of a HIL system to support the complete model-based controller development process

also allows test scripts to be developed for the 'virtual' controllers and then re-used when physical hardware becomes available.

Chapter 3. PROBLEM DEFINITION

- **Cost Factor**: Interface cards supplied by HIL system manufacturers such as National Instruments, DSpace, etc, are very expensive and delicate. This work focusses on trying to derive an alternative cost effective solution to RTPC - ECU interfacing.

- **Ownership of IP**: There is a benefit for vehicle manufacturers such as Mercedes-Benz to develop control systems independently of third-party suppliers. In addition to having ownership of the intellectual property that corresponds to a particular system, developing systems that differentiates their cars from those of competitors is important. Systems or features developed 'in-house', by engineers of the company, such as me, help to define the brand and this is important in a very competitive market. This is another reason behind the plan to design the interface card, in-house by me.

- **Flexibility**: The design choices and customizability will be in Mercedes-Benz's control. Modifications in design due to future development in the particular system can be brought about relatively quickly and with more flexibility in design.

Chapter 4. RESEARCH METHODOLOGY

4.1 Primary Layer Selection:

(DSP vs CPLD vs FPGA vs SOC vs MCU)

- ➔ DSP vs CPLD vs FPGA vs SOC vs MCU: This forms the primary layer of the interface card. Factors like speed, complexity, integration capability, etc, has to be compared and judged to choose one of these technologies to form the base layer for the card. A detailed comparison and the subsequent selection of one of them will be covered in a comprehensive manner, in this section.
- ➔ After intense scrutiny of boards available in the market, **INTEL Altera MAX 10 Development Board** is selected as our primary option as an interface card with Intel Altera MAX 10 FPGA as the primary layer. The remaining content in this section will elaborate on the key elements involved in the selection process.

4.1.1 Intel - Altera MAX 10 Dev

(FPGA + CPLD + NIOS II Soft Core)

As an interface board for Door Handle Simulation on HIL Platform

4.1.1.1 Revolutionizing Non-Volatile Integration

Intel® MAX® 10s revolutionize non-volatile integration by delivering advanced processing capabilities in a low-cost, single chip small form factor programmable logic device. Building upon the single chip heritage of previous MAX device families, densities range from 2K – 50K LEs, using either single or dual-core voltage supplies. The MAX 10 FPGA family encompasses

both advanced small wafer scale packaging (3 mm x 3 mm) and high I/O pin count packages offerings.

MAX 10 FPGAs are built on TSMC's 55 nm embedded NOR flash technology, enabling instant-on functionality.

MAX 10s offer system-level cost savings through increased integration of system component functions:

- **Dual configuration flash**—A single, on-die flash memory supports dual configuration, for true fail-safe upgrades with thousands of possible reprogram cycles.
- **Analog blocks**—integrated analog blocks with ADCs and temperature sensor provide lower latency and reduced board space with more flexible sample-sequencing.
- **Instant on**—MAX 10 FPGAs can be the first usable device on a system board to control bring-up of other components such as high density FPGAs, ASICs, ASSPs, and processors.
- **Nios® II soft core embedded processor**—MAX 10 FPGAs support the integration of the soft core Nios II embedded processors, providing embedded developers a single-chip, fully configurable, instant-on processor subsystem.
- **DSP blocks**—As the first non-volatile FPGA with DSP, MAX 10 FPGAs are ideal for high-performance, high-precision applications using integrated 18x18 multipliers.
- **DDR3 external memory interfaces**—MAX 10 FPGAs support DDR3 SDRAM and LPDDR2 interfaces through soft intellectual property (IP) memory controllers, optimal for video, datapath, and embedded applications.
- **User flash**—With up to 736 KB of on-die user flash code storage, MAX 10 FPGAs enable advanced single chip Nios II embedded applications. The amount of user flash available depends on configuration options.

The MAX 10 FPGA Development Board includes the following:

- CE-compliant MAX 10 FPGA development board
    - Featured devices
        - MAX 10 FPGA (10M50D, dual supply, F484 package)
        - Intel Enpirion® EN2342QI 4A voltage-mode synchronous step-down converter with integrated inductor

        - Enpirion EN6337QA 3A high-efficiency DC-DC step-down converters with integrated inductor
        - Enpirion EP5358xUI 600 mA DC-DC step-down converters with integrated inductor
        - MAX II CPLD – EPM1270M256C4N
    - Programming and Configuration
        - Embedded Intel FPGA Download Cable II (JTAG)
        - Optional JTAG direct via 10-pin header
    - Memory devices
        - 64Mx16 1 Gb DDR3 SDRAM with soft memory controller
        - 128Mx8 1 Gb DDR3 SDRAM with soft memory controller
        - 512Mb quad serial peripheral interface (quad SPI) flash memory
    - Communication ports
        - Two Gigabit Ethernet (GbE) RJ-45 ports
        - One mini-USB 2.0 UART
        - One HDMI video output
        - One universal HSMC connector (see HSMC expansion cards)
        - Two 12-pin Digilent Pmod* Compatible connectors
    - Analog
        - Two MAX 10 FPGA ADC SMA inputs
        - 2x10 ADC header
        - Potentiometer input to ADC
        - One external 16 bit digital-to-analog converter (DAC) device with SMA output
    - Clocking
        - 25 MHz single-ended, external oscillator clock source
        - Silicon labs clock generator with programmable frequency GUI
    - Switches, push buttons, jumpers, and status LEDs

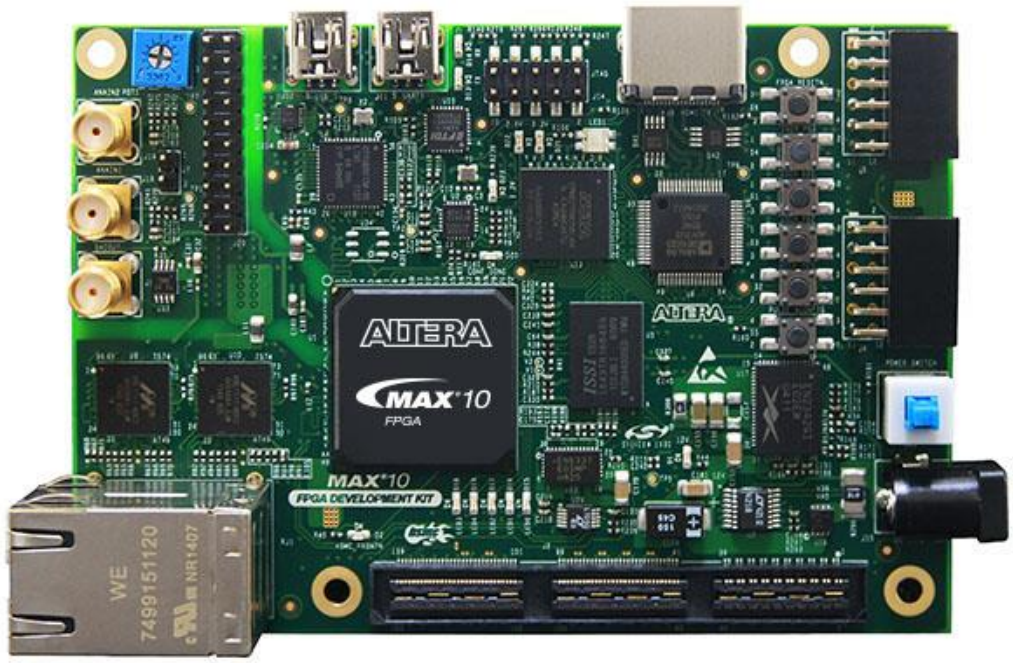


Figure 4.1: MAX 10 Development Board

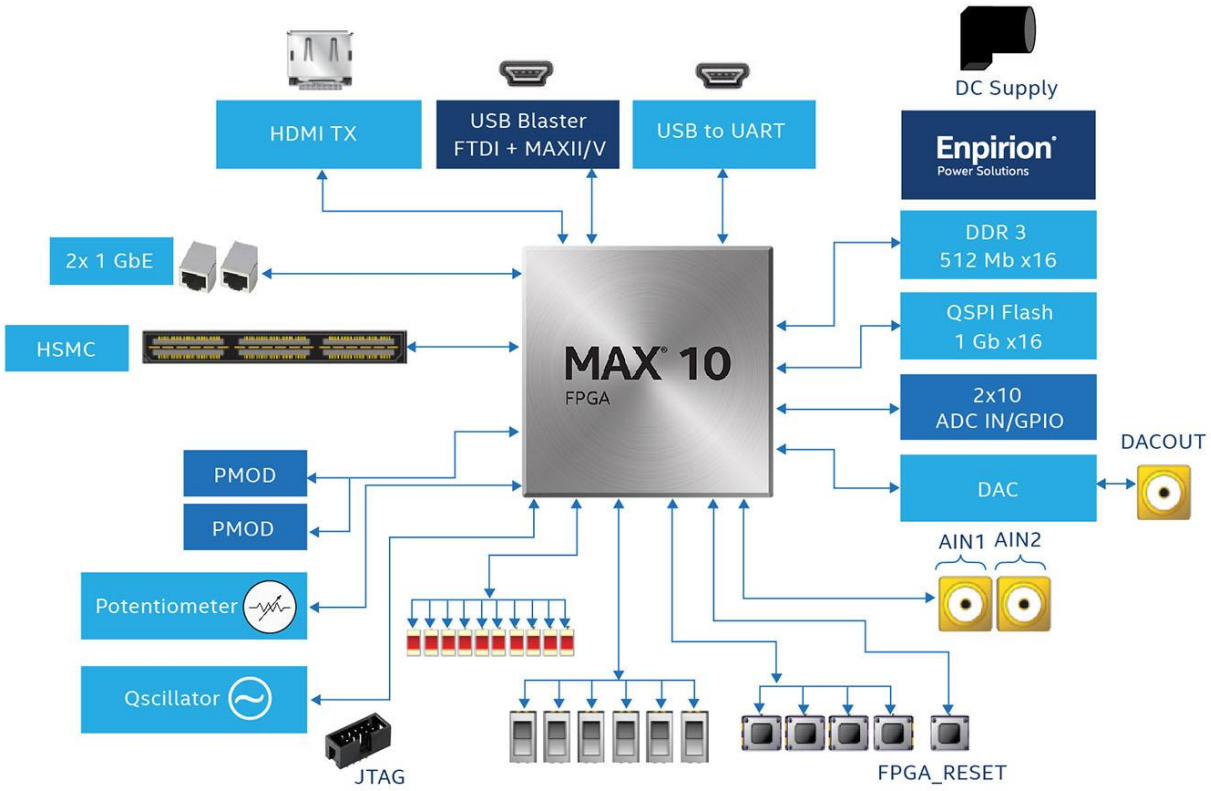


Figure 4.2: MAX 10 Development Board Block Diagram

4.1.1.2 Intel MAX 10 Architecture

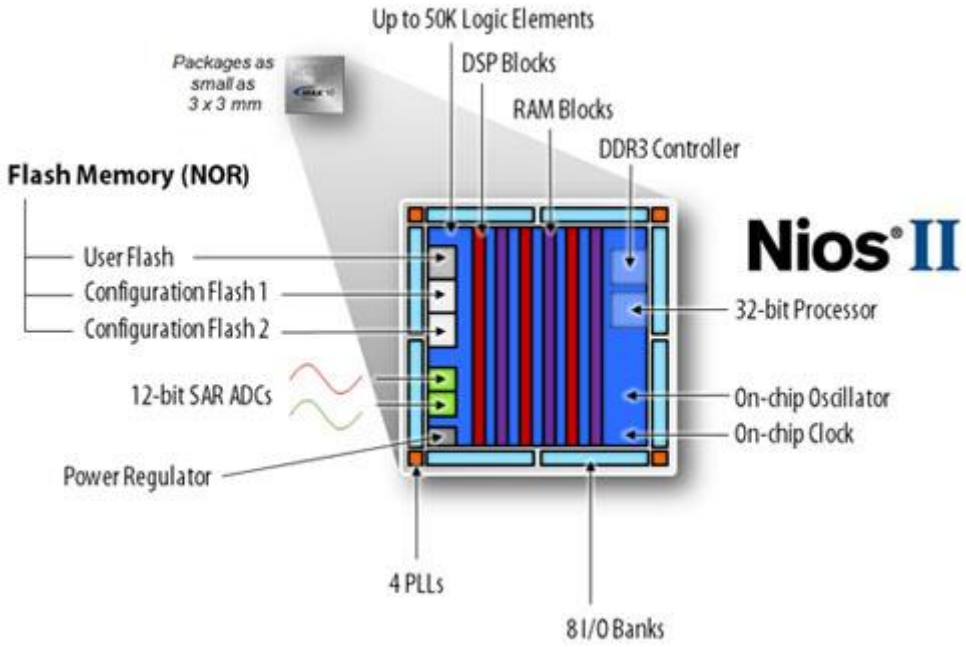


Figure 4.3: Intel MAX 10 Architecture

- Up to 50,000 logic elements (LEs)
- Up to 500 user I/O pins
- Non-volatile instant-on architecture
- Single chip
- Packages as small as 3x3 mm^2
- Embedded SRAM
- DSP blocks
- High-performance phase-locked loops (PLLs) and low-skew global clocks
- External memory interface (DDR3 SDRAM/DDR3L SDRAM/DDR2 SDRAM/LPDDR2)
- Nios® II soft core embedded processor support
- 3.3 V, LVDS, PCI*, and 30+ other I/O standards supported
- Embedded SAR ADCs – 12 bit, 1 Msps
    - Up to 18 analog input channels
    - Temperature sensor
- Single or dual-core voltage supply offering

- Embedded flash
    - Dual configuration flash
    - User flash memory
- Internal oscillator
- Power-saving features
    - Sleep mode to reduce dynamic power by up to 95%
    - Input buffer power-down
- 128 bit Advanced Encryption Standard (AES) and other design security features

4.1.1.3 MAX 10 vs CPLD

If one visits the Wikipedia page for "CPLD" he/she will find a picture of an Altera MAX device (EPM7128), a 2,500 gate-equivalent, 128 macrocell "second generation" CPLD (or "EPLD" as the company was spinning it in those days) which, according to the datasheet, was capable of implementing "complete system-level designs."

MAX 10 is a highly capable FPGA with an on-chip flash configuration memory. For the first time, the on-chip flash is capable of dual configuration. That is, it can hold more than one configuration of the FPGA. Dual configuration brings a host of new usage models to the table that we haven't yet seen on a non-volatile FPGA, such as fail-safe upgrades, where the "factory" configuration can be retained in one memory partition while a new upgrade is loaded and verified, without risk of corrupting or replacing the previous configuration. There is also a third, "user" flash memory block, which brings up an interesting set of additional ideas for HIL applications.

MAX 10 brings densities up to 50K logic elements, a quantum leap for SRAM-based non-volatile FPGAs. Lattice's largest MACH XO3 device, by comparison, weighs in at 6.9K logic elements. But MAX 10 brings a lot more than LUTs to the party. The new family includes analog blocks (ADCs and temperature-sensing diodes), which enable the devices to be used in system monitoring applications. And, with that kind of density, MAX 10 can easily integrate an Altera Nios II soft-core embedded processor, turning it into a non-volatile (almost) instant-on high-performance SoC.

Because MAX 10 is "instant on" (and in this case, "instant" means less than 10ms for the chip to configure itself from on-chip flash), a wide range of system startup, system monitoring, power sequencing, and system management applications are within easy reach – in markets

like from automotive communications. The fast boot time operation implies that it is best suited to the task at hand of deploying it as an interface between the HIL system and the ECU.

On the IO front, MAX 10 delivers up to 500 user IOs, and it includes a DDR3 external memory interface for high-performance commodity external storage. This is a considerable amount of IO for a CPLD-class device. All that IO would also make MAX 10 great for bridging applications such as this one.

Since the logic cells (LUTs) are built with SRAM-style logic and configured from flash, we don't have to pay the "flash penalty" of performance of the FPGA fabric. We'll get the performance we would expect in a conventional SRAM FPGA in a device that behaves like a non-volatile one.

4.1.1.4 MAX 10 vs MCU

4.1.1.4.1 Introduction

Processors, whether microprocessors or microcontrollers, are one of the most universal components in digital electronic systems. With the recent rapid growth of Internet of Things (IoT) solutions this prevalence continues to increase. Whether designing for the IoT revolution or bleeding edge products that require novel solutions, a commercial off-the-shelf (COTS) processor cannot provide the optimal combination of performance, peripherals, form factor, scalability, or life cycle persistence to deliver competitive differentiation and time-to-market needs. With COTS processors, designers must compromise on microcontrollers with fixed functionality by choosing to either overpay for features they don't need or over-designing their system to add features the COTS lack. FPGA-based solutions, with their integrated fully customizable soft processing capability, enable true flexibility and scalability—addressing custom needs with programmable hardware and software in a single chip.

The Nios® II processor is a FPGA-optimized 32 bit RISC Harvard architecture processor (soft) core. As the FPGA device is programmable, the Nios II processor is also configurable—enabling it to meet the exact requirements of any application; in addition the processor peripheral set is also easily configured to meet any application-specific requirement. Intel® MAX® 10 FPGA-based solutions based on soft-core Nios II processors, overcome the limitations of COTS by enabling a unique single-chip embedded system that can be customized

for product differentiation and optimization. MAX 10 FPGA's innovative architecture and Nios II processor's flexibility provide an unmatched alternative solution for today's embedded designers.000

4.1.1.4.2 Single-chip, instant-on capability

By integrating 55 nm embedded flash onto the MAX 10 FPGA die, the capability for a true single-chip embedded system with both hardware and software customization is fully realized. Imagine a CPU with customizable hardware features that can change real-time in the field to match the Quality of Service (QoS) or feature packages licensed by the end customer. Better yet, the ability to upgrade a system's microcontroller hardware for changes due to emerging standards, functionality missing in initial release due to time-to-market pressures, or product upgrades purchased after the initial installation. These scenarios are not physically possible with COTS processors, but are possible when utilizing Nios II processors with MAX 10 FPGA's embedded flash and remote update capability. System boot and management also benefit from the on-die
flash. In traditional FPGA systems with embedded processor technology, whether hard or soft, the FPGA takes a "noninstant" amount of time to power-up and configure. With
MAX 10 FPGA's on-die flash, the FPGA powers-up instantly (in a few milliseconds) as the first component in the system, allowing the custom FPGA logic to not only completely manage system bring-up but also allowing the Nios II system to be available at system power-on for software diagnostics or prognostics.

4.1.1.4.3 Small form factor, integrated devices

With packaging as small as 3 x 3 mm2, MAX 10 FPGA's single-chip solution is the smallest configurable FPGA footprint in the industry. These small package sizes allow the MAX 10 FPGA to replace or augment ASICs, ASSPs, and microcontroller units (MCUs) in portable or space constrained applications.

4.1.1.4.4 Customizable peripheral sets

Customized peripherals in the FPGA can be an embedded system's "secret sauce" allowing any number of general purpose I/Os (GPIOs), Ethernet MACs, serial interfaces, multiple CPUs,

and so on. A large library of embedded peripherals are available to plug into the custom system or designers have the option of creating their entirely unique and custom hardware
Peripheral blocks using Verilog HDL or VHDL. Assembling the exact set of peripherals necessary for an end system, which are not available in COTS products, gives embedded designers a differentiating advantage.

4.1.1.4.5 Real-time processing optimization

Traditionally, embedded developers have had limited options for accelerating performance near the end of a design cycle, including buying a faster processor or last minute hand tuning of assembly subroutines. While both options can be effective, the trade-offs they bring are too large to ignore.
MAX 10 FPGAs and Nios II processors provide an entirely new toolbox of performance-enhancing features. Using custom hardware accelerators, designers can optimize their system performance in a way not possible with traditional off-the-shelf processors. The configurable nature of Nios II processor systems allow designers to create custom components in FPGA logic that can run as a co-processor unit for complex algorithms. These accelerator or coprocessing units can run in parallel to the Nios II processor, executing functions orders of magnitude faster than software execution.

4.1.1.5 MAX 10 vs DSP

DSP is basically processor that is running some code from the memory that is converting some data in the memory and it can also interface some peripherals to interface to the outside world. There is very large variety of DSP processors from small, general purpose to some very specialized and sophisticated models. This processors are not very different from general purpose processors or microcontrollers and line between this groups of processors are not very clear specially with addition of specialized peripherals to available SoC designs. MAX 10 in other hand is not processor in the basic concept, it is simple digital logic circuit that can be configured to run different operations from logic, signal processing and one can configure FPGA as DSP or general purpose processor. Basic difference from running the code like in DSP to configuring FPGA for some operation is that FPGA can adjust architecture to

predefined process and is not limited to simple process some instructions. Of course one can always combine this two approaches in MAX 10.

4.1.1.6 Why MAX 10 is ideal for this particular application?

- MAX 10 combines the better of all the two worlds, the worlds of a FPGA and a CPLD. And when a soft core processor is thrown into the package, it sweetens the deal. MAX 10 can be programmed using either VHDL or Verilog, although myHDL provides the option of some python productivity.

- While zeroing in on the optimal board, MAX 10 in it's price range, faced fierce competition from its Xilinx counterpart, ZYBO-Z7: ZYNQ-7000. ZYBO-Z7 was complemented by MAX 10's second rival, the PYNQ-Z1 board by Xilinx, a board specially designed for Python productivity. The primary difference between the Xilinx pair and MAX 10 is that, both the boards from Xilinx are APSoCs (All Programmable Systems on Chips). It primarily features an ARM Cortex – A9 processor, responsible to carry out high performance serial processes, while being tightly integrated with a Xilinx FPGA, sufficing the high computational needs.

  For this application, SoC is not really recommended as there is no requirement of intensive signal processing like tasks in this project. Speed is of the essence in this project and I feel this board is more than enough in catering to that, especially due to its parallel process execution nature. A heavy processor is unnecessary and keeping it as light as possible, is the way to go, in my perspective.

- MAX 10 was also compared to CPLD's, namely Coolrunner II , by Xilinx, and MAX V, by Intel-Altera. CPLD's on their own might just be capable of serving this project's purpose, but are outrun by MAX 10, while looking at a long run perspective. MAX 10, as stated earlier has a CPLD component in it, which makes MAX 10 far superior to a CPLD. As much as I would have loved to implement a CPLD board in this project, one cannot ignore the fact that regardless of their availability and capability, they are becoming a thing of the past with the emergence and rise of MAX 10 - like class of devices or ApSoC class of devices, and a decrease in support, rendering them a prospect of becoming obsolete in the near future.

➔ MAX 10 has been tried and tested for real time applications and the combination of the NIOS II core and FPGA outperform off-the-shelf processors in system performance. On a closing remark, good old DSP boards did not really stand a chance in comparison to MAX 10, not only because MAX 10 has DSP blocks built in it, but also because DSP's serial execution nature and its importance in the complex signal processing use cases, where obtaining precision and accuracy from complex processing is of more importance than agility and speed, stands inferior to MAX 10's capabilities, as far as this project is concerned.

➔ The MAX 10 dev board has two ethernet ports that will be used to communicate with the RTPC. The board also has two 12-pin PMOD ports, one of which will be used to interface an external PMOD RS232 UART module to enable serial communication between the board and the RTPC. A CAN PMOD module can also be used for CAN communication in the future. Other boards do not provide this flexibility is design.

4.2 THE KEYLESS – GO SYSTEM

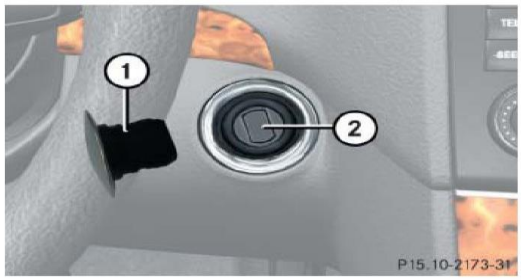
4.2.1 General Description of Keyless Go

- Keyless-Go is an electronic access and authorization system, in which driver doesn't have to take the key in hand before boarding or starting the vehicle.
- KG system allows to use the vehicle without active operation of KEY.
- In KG system key is called as ID-Encoder or ID-Transponder.
- In KG system it is enough to carry the key to Lock, Unlock, and to start or stop engine.
- All the above operations can be performed after authentication of key. If key authentication is success then user can have access to vehicle otherwise not possible to access the vehicle.

Figure 4.4: The key used in a Keyless-Go system

4.2.2 General Description of Keyless Start

- The system allows the use of the vehicle without any active operation of the Key. It extends the carrying of Key to perform engine start.
- The number of connected LF antennas is limited in keyless start system (Unlike in Keyless Go 4-LF and 1-HF) and is coded with SCN (software calibration number). Accordingly the coded antennas are selected and these outputs in the system diagnostics are considered.
- The ZB-functions (unlocking, security, trunk internal search) are not available in Keyless start.
- If keyless go is not coded in global variant coding then keyless start is available.



① KEYLESS-GO start/stop button
② Starter switch

Figure 4.5: Keyless Start visual imagery

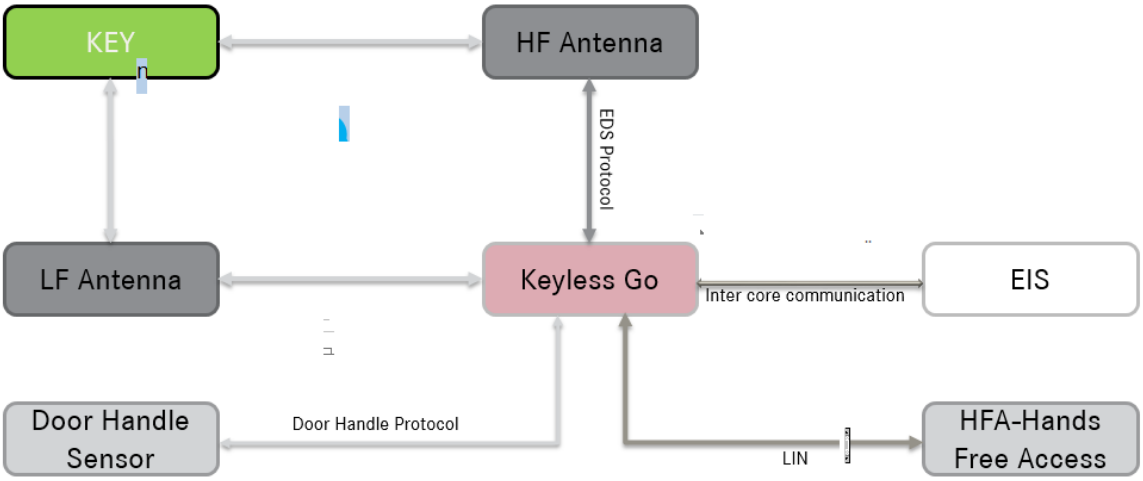
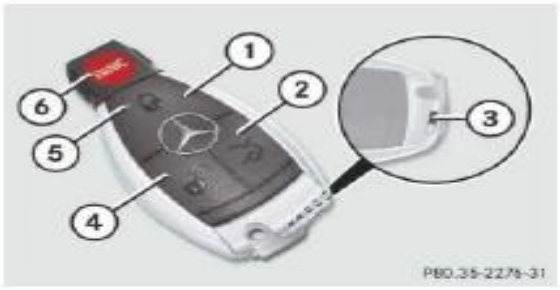


Figure 4.6: KG SYSTEM BLOCK DIAGRAM

4.2.3 Key

- Key communication with KG system is wireless communication via LF and HF antennas.

- Key should be valid as per Keyline (Set of keys designed for respective KG ECU) to communicate with KG system.
- Position of Key is required to perform certain functions and the position of the key is recognized by NF antennas.
- For location determination of the Key, data is sent via arranged NF/LF antennas to the KG system in the vehicle. Depending on inductive antenna's reception level of the Key data determines the position of the Key.
- If no key is detected on a valid position the query is repeated a certain number by attempts. If no key is detected even after repeated request the action is cancelled with display "ID-transponder not recognized".
- The evaluation of the authentication codes and the control of actual access or driving authorization function is carried out by the EIS.



SmartKey with KEYLESS-GO
① Lock button
② Opening button for trunk/tailgate (▷ page 90)
③ Mechanical key locking tab
④ Unlock button

Figure 4.7: SmartKey with KEYLESS-GO

4.2.4 Outer Door Handle

- Secure Sensor is used to secure (lock) the car door.
- Spot sensor is used for comfort functions. After closing the doors vehicle can be secured by spot sensor.
- Unlock sensor is used to unlock the vehicle.
- To open the doors Hall sensor is pulled.

- All the above operations can be performed if and only if valid key is present with in the range of the vehicle.

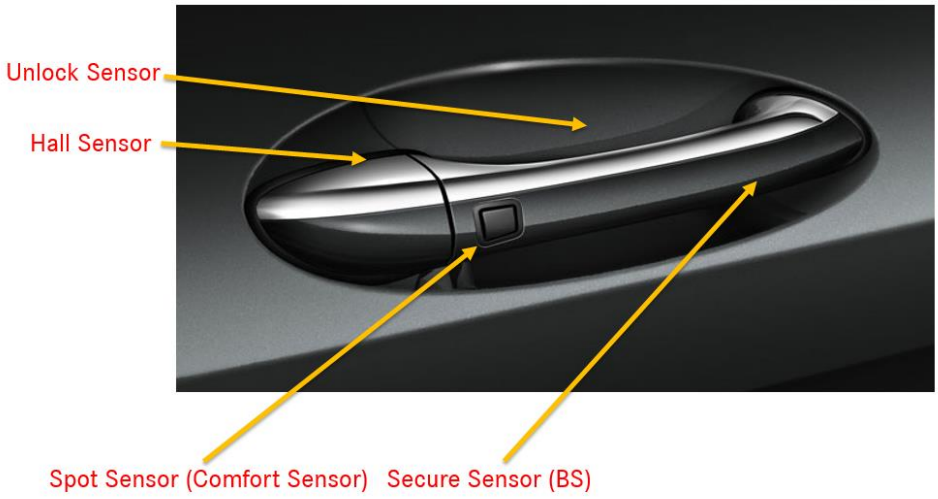


Figure 4.8: Door Handle Sensors and their Positions

4.2.4.1 NS – Sensor

- The NS-sensor is located inside the handle and has a high sensitivity. It initiates the process unlocking vehicle.
- Capacitive proximity sensor is activated in handle bar for every unlock request. The unlock request is transmitted through a defined protocol to KG control unit. This signal is triggered only when reaching into the door handle but not by pulling out of handle.

4.2.4.2 BS – Sensor

- BS-sensor is located on the outside of the handle and is designed insensitive. so the only direct contact leads to a trigger. It initiates the process of securing vehicle.
- A capacitive sensor is attached to the exterior side of each door handle bar. If this surfaces receives secure actuation from outside then KG unit performs an identification check with the help of corresponding antenna and verifies the existence of an internal key with in the indoor antennas. At least one key should be found outside of the vehicle in this case to secure the vehicle.

4.2.4.3 Spot – Sensor

- The spot sensor is present on the outside of the handle by and is designed insensitive, so the only direct contact leads to a trigger. It initiates the process of vehicle securing and comfort closing in additional.
- A securing of vehicle by spot sensor is carried out by actuation of the 3 capacitive sensor on the outer side of the handlebar. If the spot sensor is pressed for more than 1.2 S a comfort closing will be executed. The maximum duration of comfort function is 90 S. An emergency opening of door handle is triggered up to 2 S after the end of comfort closure.

4.2.4.4 Hall – Sensor

- The Hall-sensor detects a pull of the handle from its zero position. Here, the Hall sensor is pulled out of the magnetic field of a permanent magnet.
- After activation of the Hall sensor and a subsequent recognized deactivation should be detected before the protocol is switched off. The sensor is additionally 30 times scanned and examined for a possible toggle operation.

4.2.5 Outer Door Handle Operations

4.2.5.1 Unlock over door handle

- Person with a valid key should be present on the side of the vehicle on which the door handle has been operated.
- Door will be unlocked if person with valid key touches the unlock sensor. To open the door handle the Hall sensor should be pulled.
- If the actuation takes place in the driver's door, a request for selectively unlocking has to be started else request for global unlocking has to be started.
- If the actuation takes place on the front passenger seat or the rear doors, a request for global unlocking must be started.

Figure 4.9: Door Handle Unlocking;

Clockwise: NS Sensor Trigger, Pull action, Hall Sensor Trigger, With Key Near-By

4.2.5.2 After unlocking via door handle

- If vehicle door is unlocked by rotary latch from inside of the car and door is closed then for 1.5 seconds car door doesn't open if hall sensor is pulled.
- Outer door handle on the respective vehicle door detects the control desire to open and these signals via the door handle interface to the keyless-go.
- Keyless-go must evaluate the received protocol of the outer door handle according to closing logic.
- Person with a valid ID encoder should be present on the side of the vehicle on which the door handle has been actuated.
- The corresponding Rotary latches are not recognized as open after identifying the Hall-signal within 1.5 seconds.
- The door cannot be opened with in 1.5 seconds in this case. After 1.5 seconds if the valid key is present the vehicle door can be open.

Figure 4.10: After Unlocking of Door Handle;

Clockwise: Locked , Unlocked , Pull Hall Sensor , Key nearby

4.2.5.3 Locking over door handle

- Vehicle can be locked by secure sensor or comfort sensor. Locking via secure sensor is only allowed if the vehicle is not globally locked.
- One actuates secure sensor for securing vehicle or securing vehicle via comfort in the outer door handle. Outer door handle detects the operating demand to secure and signals this via the door handle interface to the keyless-go.
- Keyless-go must evaluate the received protocol of the outer door handle according to closing logic.
- Person with valid ID transmitter should be located on the vehicle side where the door handle is pressed.

Figure 4.11: Locking of the Door Handle;

Clockwise: Locked , Touch BS Sensor , Touch Comfort Sensor , Key nearby

4.2.5.4 Comfort functions via outer door handle

- When the comfort sensor on the outer door handle is pressed the door will be locked. If the comfort sensor on the outer door handle remains still pressed for 1.5 seconds then comfort functionalities will start.
- Those includes closing of roof top, closing of windows and other comfort functionalities.
- Person with a valid ID encoder should be present on the side of the vehicle on which the door handle is actuated. Comfort functions will be started after successful authentication of key.
- Comfort functionalities will be terminated if the comfort sensor is released on the outer door handle and when terminal/ locking status changes.
- Comfort functionalities can also be terminated if any other sensor on the outer door handle is activated.

Figure 4.12: Comfort Functions of the Door Handle

Clockwise: Touch Comfort Sensor , Window closed 50% , 75% , 90%

4.2.5.5 Emergency opening via outer door handle

- Emergency opening via outer door handle is a scenario of interrupting comfort functions by pulling hall sensor of outer door handle.
- When comfort function is active and if a person tries to open outer door handle, the comfort functions will be halted.

4.3 Door Handle Telegram Protocol

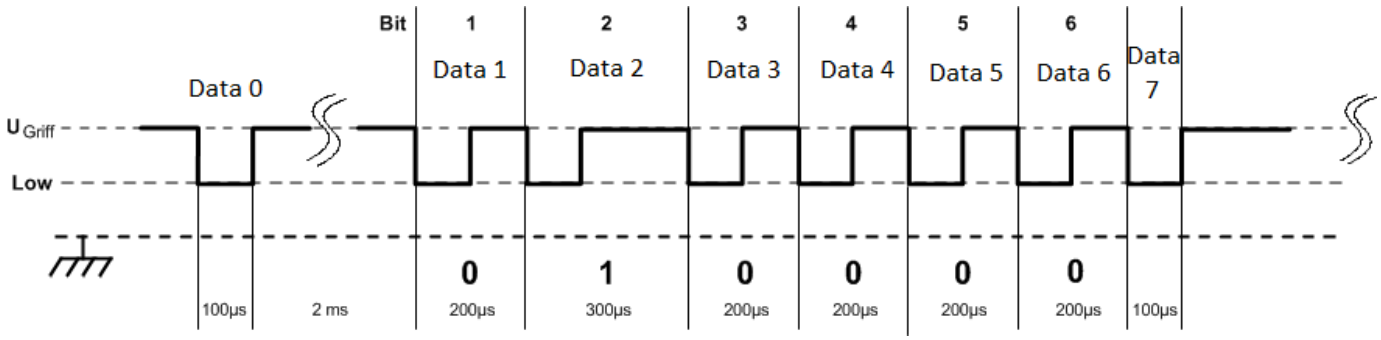


Figure 4.13: Door Handle Telegram

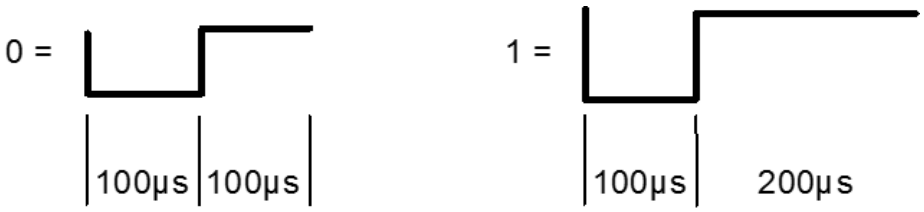


Figure 4.14: Bit Coding-Low and High pulse values for door handle protocol

- *The door handle sends a pre-pulse before the first telegram. This serves to awaken the current receiver as well as activate the receive mode, so that already the first door handle telegram can be received and evaluated. This first impulse is present only in the first protocol. All other telegrams which are cyclically sent wont have preimpulse but will have only an "intermission".*
- *Error Bit will be set when there is any invalid length of door handle protocol is detected or any incorrect message is present in any other bit fields of door handle protocol.*
- *The Protocol of the door handle is repeated cyclically for minimum 7ms and maximum of 20 ms.*
- *The KG+NFC door handle transmits the information "NFC signal". With this, the request for NFC communication is communicated from the door handle to the KG control unit. This information is only evaluated in the KG control unit if the relevant SA code is coded in the KG control unit, otherwise it is ignored. The hall signal is evaluated according to the content of the first bit, without redundancy*

4.4 FBS-mobile - System overview

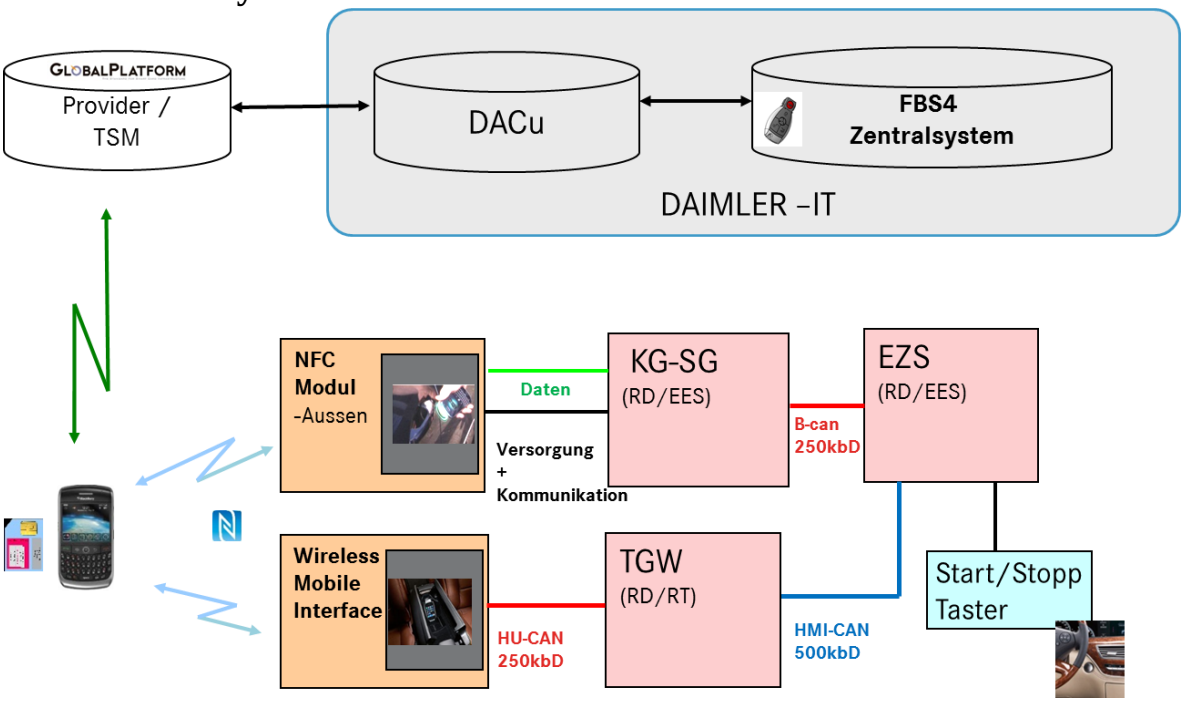


Figure 4.15: System Overview of FBS-Mobile

With the special equipment "FBS-mobile", the driver's door handle gets an NFC interface in addition to the existing sensors.

4.5 Maximum equipment Keyless Go with FBS-mobile

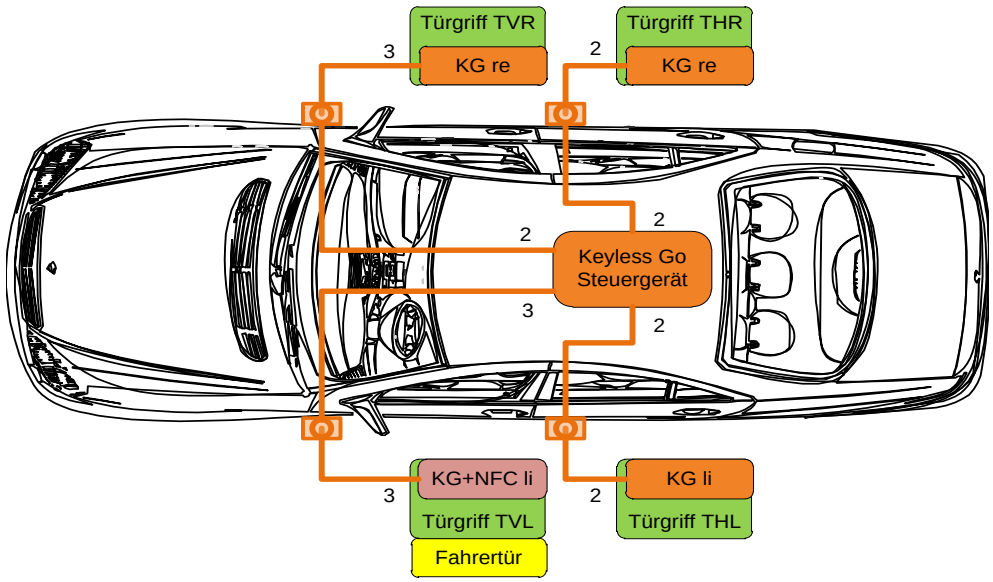


Figure 4.16: Maximum equipment Keyless Go with FBS mobile

Source	Target
Türgriff	Door handle
Fahrertür	Driver’s door
KG li	KG left
Steuergerät	Control unit
KG re	KG right

4.6 Variant overview

Table 4.1: Variant Overview

	Description	Door handle	Electronics
Series	Keyless Start	Standard	without
SA	Keyless Go	Keyless Go	Keyless Go
SA	FBS mobile	Keyless Go(1)	Keyless Go + NFC(2)
SA	Keyless Go and FBS mobile	Keyless Go	Keyless Go + NFC

(1) KG door handle on FT always with comfort spot, without function in case of code 896.

(2) Only FBS-mobile function via NFC.

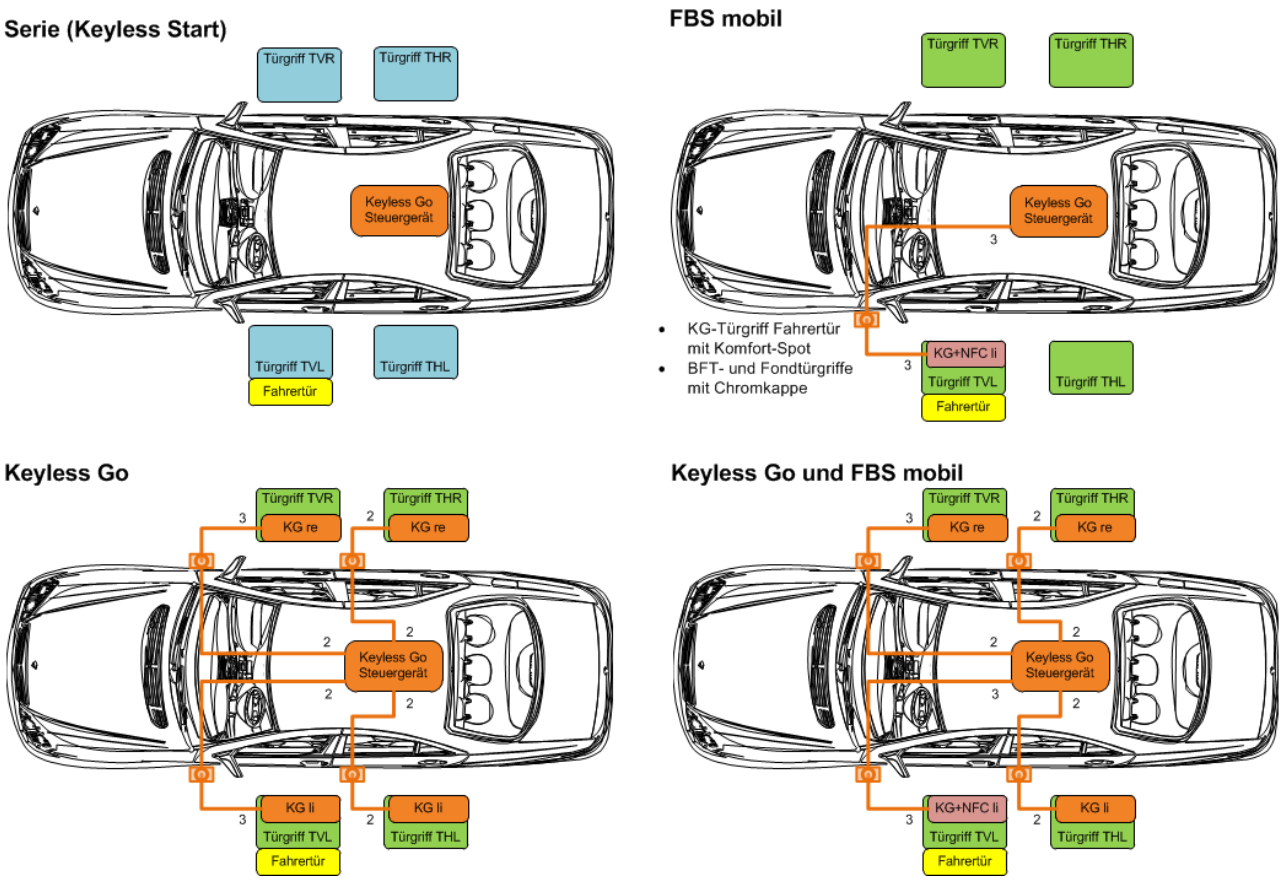


Figure 4.17: Variant Overview

Source	Target
FBS mobil	FBS mobile
Serie	Series
Und	And
Türgriff	Door handle
KG li	KG left
KG re	KG right
KG-Türgriff Fahrertür mit Komfort-Spot	KG door handle for driver‘s door with comfort-spot
BFT- und Fondtürgriffe mit Chromkappe	BFT and rear door handles with chrome lid
Fahrertür	Driver’s door

4.7 Interfaces for FBS mobile

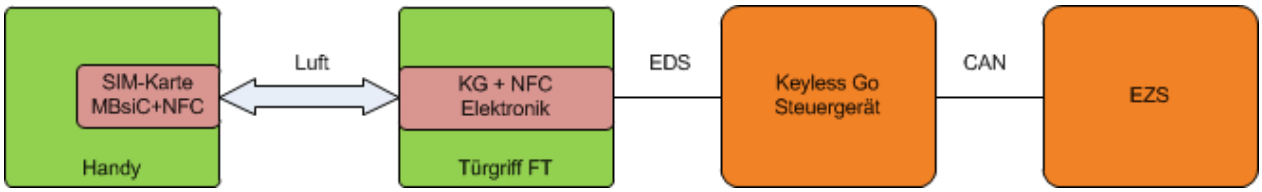


Figure 4.18: Interfaces for FBS Mobile

Source	Target
SIM-Karte	SIM card
Elektronik	Electronics
Türgriff	Door handle
Steuergerät	Control unit
Luft	Air

4.8 Door handle sensor position and functions

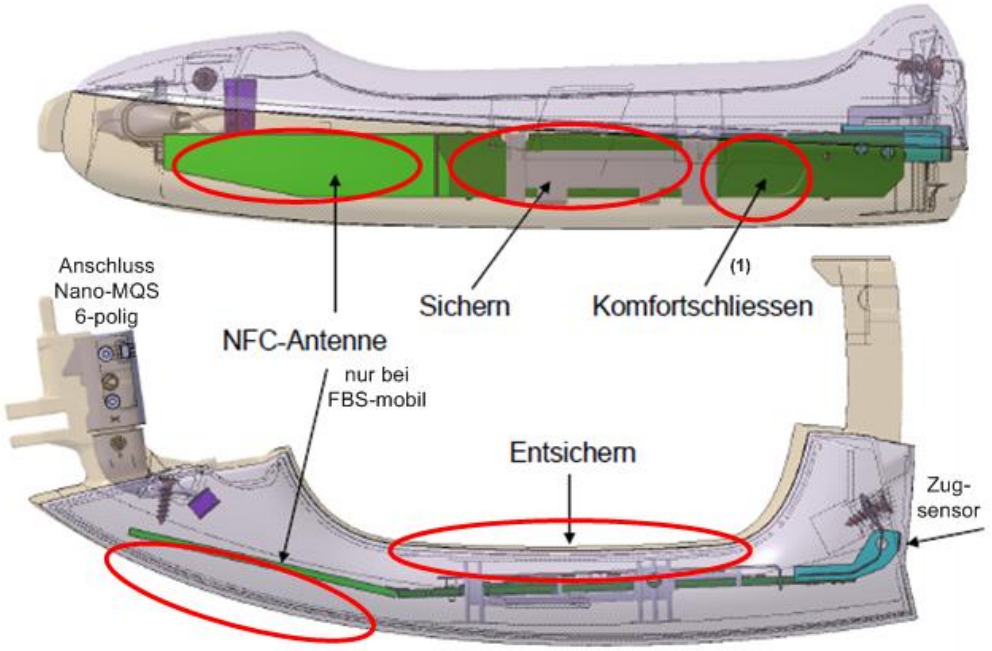


Figure 4.19: Door Handle Sensor Position and Functions

Source	Target
Anschluss Nano-MQS 6-polig	Connection 6-pin
Entsichern	Unlock
Sichern	Lock/Secure
NFC-Antenne	NFC antenna
Zugsensor	Tension sensor
Komfortschliessen	Comfort closure
Nur bei FBS-mobil	Only in case of FBS mobile

The NFC door handle is connected to the Keyless Go (KG) control unit.
This connection is always assigned to the driver's door.

Chapter 5: Interface Card Design and Implementation

RS232 UART based design and implementation

This section will cover the design and implementation aspect of the real time simulation of a door handle on a Hardware-in-the-Loop platform. Communication protocols such as RS232 UART and Gigabit Ethernet will be implemented for seamless integration with the existing Keyless-Go HIL platform.

RS232 based implementation will be looked into in greater detail than the Ethernet based implementation.

5.1 Salient Elements

- Using Intel max 10 to act as an interface between the Keyless go ECU and the RTPC, TO SIMULATE THE FUNCTIONALITIES OF THE DOOR HANDLE.
- RS-232 COMMUNICATION AS THE SERIAL INTERFACE BETWEEN TRANSMITTING SYSTEM AND THE MAX 10 DEV BOARD.
- Using Verilog as the HDL of choice to construct a digital design in the MAX 10 FPGA.

5.2 Digilent PmodRS232™ Converter Module Board

➔ The PmodRS232 Converter Module Board (the RS232 module) translates voltage from the logic levels used by Digilent system boards to the RS232 voltage used for serial communications.

➔ The RS232 module creates a two-way I/O exchange by converting RS232 voltage to logic level voltage and converting logic voltage to RS232 voltage. RS-232 voltage levels are -3 to -12V for a logic '1', and +3 to +12 for a logic '0'.

➔ The RS232 module is configured as a data communications equipment (DCE) device. It connects to data terminal equipment (DTE) devices, such as the serial port on a PC, using a straight-through cable.

➔ Features include:

- A Maxim Integrated™ MAX3232CSE RS232 transceiver
- A DB9 connector and 6-pin header
- Transmit and receive data functions
- optional RTS and CTS handshaking functions

5.2.1 Functional Description

➔ The RS232 module provides two transmit buffers and two receive buffers. A transmit buffer converts a logic-level signal on its input to an RS232 voltage-level signal on its output.

A receive buffer converts an RS232 voltage level signal on its input to a logic-level signal on its output. The RS232 module can be configured as either a 3-wire DTE serial port (with one transmitter for transmit-data signals, one receiver for receive-data signals, and a signal ground connection), or as a 5-wire DTE serial port with an additional transmitter and receiver for RTS and CTS handshaking signals, respectively.

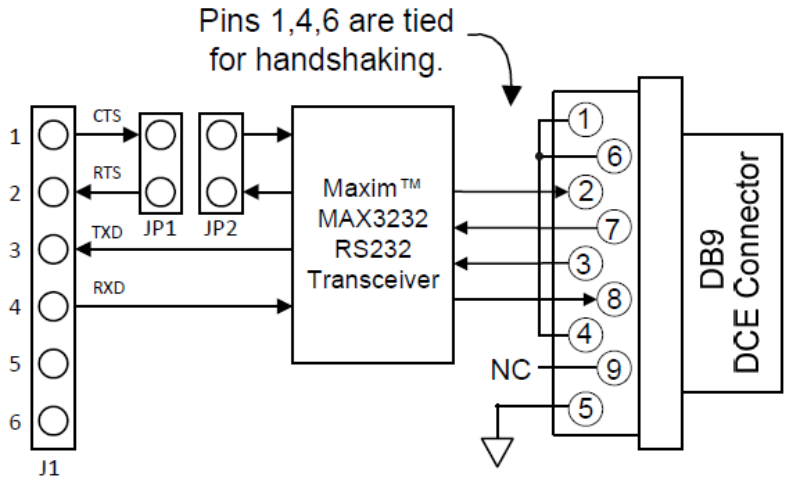


Figure 5.1: RS232 Module Block Diagram

➔ The RS232 module is wired as a DTE device. RS232 signals are named from the perspective of the DCE. The TXD signal carries data from the DCE to the DTE, therefore, the TXD signal on pin 3 is connected to the output of a receiver and is connected to the receive input of a UART on the system board. Similarly, the RXD signal carries data from the DTE to the DCE and is connected to the input of a transmitter on the RS232 module and is connected to the output of the UART on the system board. The CTS signal on pin 1 can be connected to the input of a transmitter and the RTS signal on pin 2 can be connected to the output of a receiver.

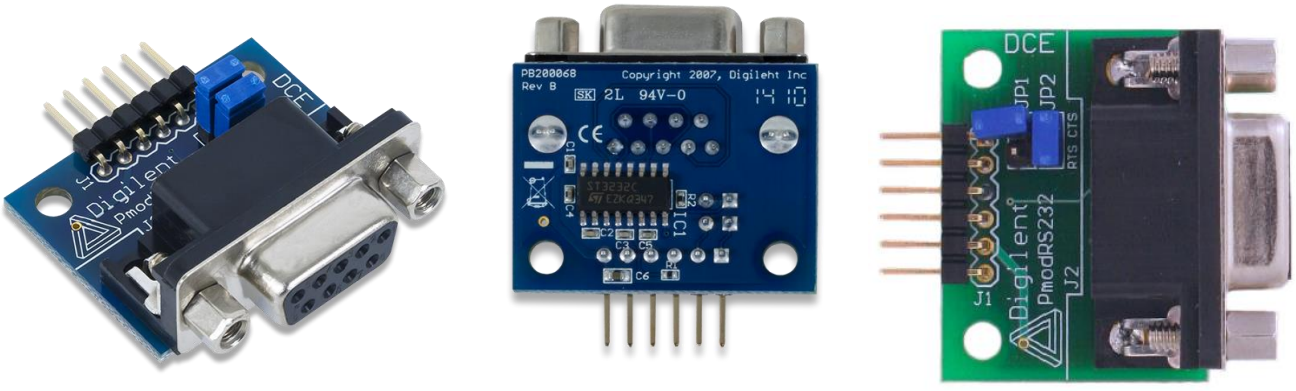


Figure 5.2: Digilent PmodRS232™ Converter Module Board

5.2.2 Using Jumper Blocks JP1 and JP2

- Jumper blocks JP1 and JP2 are used to configure the RS232 module for either 3-wire or 5-wire operation. The 3-wire operation has been implemented and tested in this project. Pins 1 and 2 of JP1 are connected to pins 1 and 2 of connector J1, respectively. Pins 1 and 2 of JP2 are connected to the CTS transmitter and the RTS receiver, respectively.
- To configure the RS232 module as a 3-wire DTE with no handshaking, we have placed a shorting block across the two pins of JP2 and ensure there's no shorting block on JP1. This loops RTS back to CTS on the RS232 side of the module and leaves pins 1 and 2 unconnected on J1.
- To configure the RS232 module as a 5-wire DTE with RTS/CTS handshaking, we can place a shorting block across pin 1 of JP1 and pin 1 of JP2, and place another shorting block across pin 2 of JP1 and pin 2 of JP2. This connects the CTS transmitter to pin 1 of J1 and the RTS receiver to pin 2 of J1.

5.3 Transmission side: PC / RTPC

- A LABVIEW based UI or a custom Hyperterminal / Teraterm based UI can be used to transmit data from a PC or a RTPC.
- Baud rate has to be the same as of the FPGA Receiver module, 115200 in this case.
- If the module uses a 10 MHz clock and has to interface to 115200 baud UART :

10000000 / 115200 = 87 Clocks Per Bit.

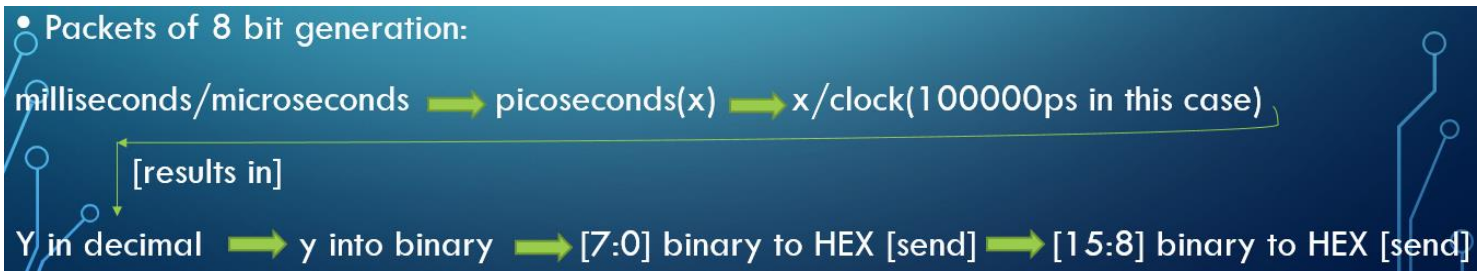


Figure 5.3: Method for generation of 8 bit long packets

5.4 Verilog Module Control Flow Diagram

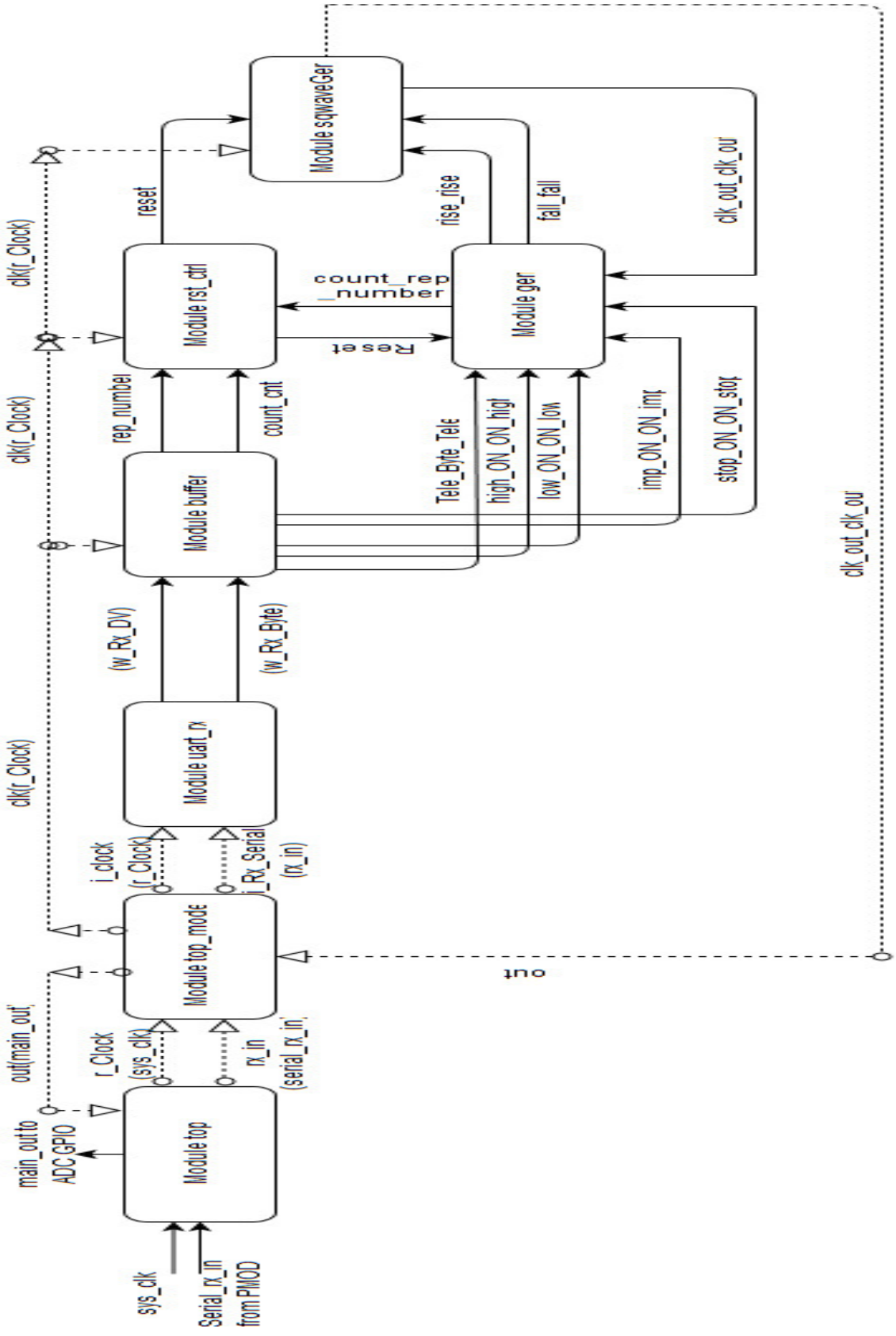


Figure 5.4: Verilog Module Control Flow Diagram

The diagram above and the following section elucidate the Verilog modules designed and also provide an overview of how the control flows through the modules to produce the desired output signal using the serial inputs fed into it.

5.5 Verilog Design Components

➔ ***Module top***

- This module is the top module of the project that connects physical pins on the board to virtual pins of the model. Represents the inputs and outputs of the entire design. System clock and the serial input are the two inputs to the module, and the only output is the telegram. This module initializes the module top_model, and passes on it's inputs to the inputs of the module top_model. The output of top_model module is passed on to the output of the top module. The inputs of the top module are connected to the on board oscillator chip or a clock modified PLL, and the Rx receive pin of the external UART RS232 port, via the PMOD port, respectively. The output of the module is connected to one of the ADC GPIO pin.

➔ ***Module top_model:***

- This module initializes all the remaining modules, and makes interconnections between the modules it initializes. We assume that the module uses a 10 MHz clock and want to interface to 115200 baud UART. Based on these assumptions, the clocks to bit ratio aggregates to 87. It also passes on the output of the sqwaveGen module to the top module.

➔ ***Module uart_rx:***

- This module contains the UART Receiver. This receiver is able to receive 8 bits of serial data, one start bit, one stop bit, and no parity bit, from rx_in (connected to the serial receive pin of the external UART port, via the onboard PMOD port). When receive is complete o_rx_dv will be driven high for one clock cycle. We set the parameter CLKS_PER_BIT as follows:

- CLKS_PER_BIT = (Frequency of i_Clock)/(Frequency of UART) Example: 10 MHz Clock, 115200 baud UART (10000000)/(115200) = 87
- We Double-register the incoming data because it allows the data to be used in the UART RX Clock Domain. It also removes problems caused by metastability
- The FPGA is continuously sampling the line. Once it sees the line transition from high to low, it knows that a UART data word is coming. This first transition indicates the start bit. Once the beginning of the start bit is found, the FPGA waits for one half of a bit period. This ensures that the middle of the data bit gets sampled. From then on, the FPGA just needs to wait one bit period (as specified by the baud rate) and sample the rest of the data. The figure below shows how the UART receiver works inside of the FPGA. First a falling edge is detected on the serial data line. This represents the start bit. The FPGA then waits until the middle of the first data bit and samples the data. It does this for all eight data bits.

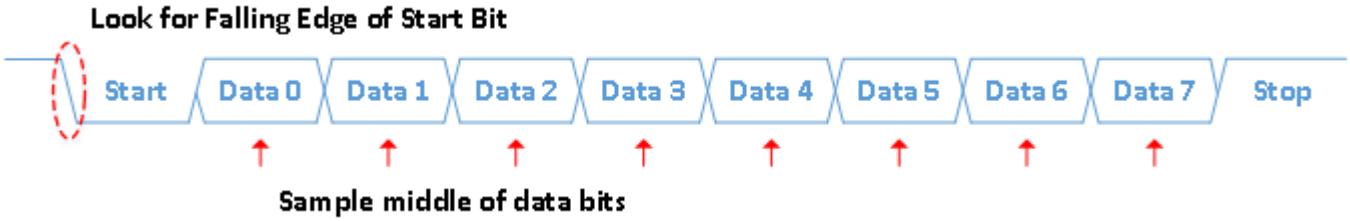


Figure 5.5: UART Serial Data Stream

- The above data stream shows how the module is structured. The 8 bit data, w_Rx_Byte and w_Rx_DV(o_rx_dv) are the outputs of this module.

➔ ***Module buffer:***

- The outputs of the uart_rx module is fed as inputs to this module. This module takes 8 bits of data, ten times, and feeds the incoming data into the various parameter variables, that controls the telegram properties such as the ON duration of the HIGH bit, ON duration of the LOW bit, ON duration of the HIGH signal between the impulse and the first telegram bit, ON duration of the HIGH signal between

two consecutive telegrams, the number of telegram repetitions, and lastly the bit status of the Telegram bits. The w_Rx_DV input from the uart_rx buffer acts as the primary clock of this module, with the system clock as its secondary clock.

➔ ***Module gen and Module sqwaveGen*:**

- The gen module deploys a state machine that reads each telegram bit status and calls the sqwaveGen module to generate a LOW-HIGH of variable HIGH duration signal based on the contents of the various parameter variables, obtained from the module buffer. The impulse followed by a HIGH of variable duration is triggered only before the first telegram cycle. A HIGH signal of variable duration exists between consecutive telegram cycles. Module gen operates on the clock provided by the output of the module sqwaveGen, and this output of swaveGen module is fed back into the top_model module which inturn feeds it back to the top model's output. The top model's output is connected to the physical ADC GPIO pin. The sqwaveGen module operates on system clock and the output of the rst_ctrl module. Module gen also outputs the number of times the telegram has been repeated, at the end of every telegram cycle, and sends it to the module rst_ctrl.

➔ ***Module rst_ctrl:***

- This module uses the input from the module gen to keep a track of the number of telegram repetitions. Once the number of telegram repetitions matches with the maximum number of repetitions, obtained from the buffer module, the reset is set to LOW, which means that the telegram repetition would cease and a steady HIGH signal would be witnessed at the ADC GPIO pin. The reset is set to HIGH, upon the completion of receiving all of the bits (80 bits is this case) received at the receive pin of the external UART part, via the PMOD port. Once the receive is complete, telegram generation repetitive cycle can begin.

5.6 Verilog codebase configuration

➔ **Controllable Parameters with specifications:** Below is the table listing the parameters that can be controlled by the user and the specifications linked to its control.

Table 5.1: Configuration, Controllable Parameters with specifications

Parameters	Specifications
Telegram + NFC Presence + Reserved	6 bits + 1 bit + 1 bit respectively
Number of telegram repetition	8 bits value
ON duration of Telegram's HIGH bit	Clock * 16 bits
ON duration of Telegram's LOW bit	Clock * 16 bits
Duration of 'after impulse HIGH' region	Clock * 16 bits
Duration of HIGH between successive telegrams	Clock * 16 bits

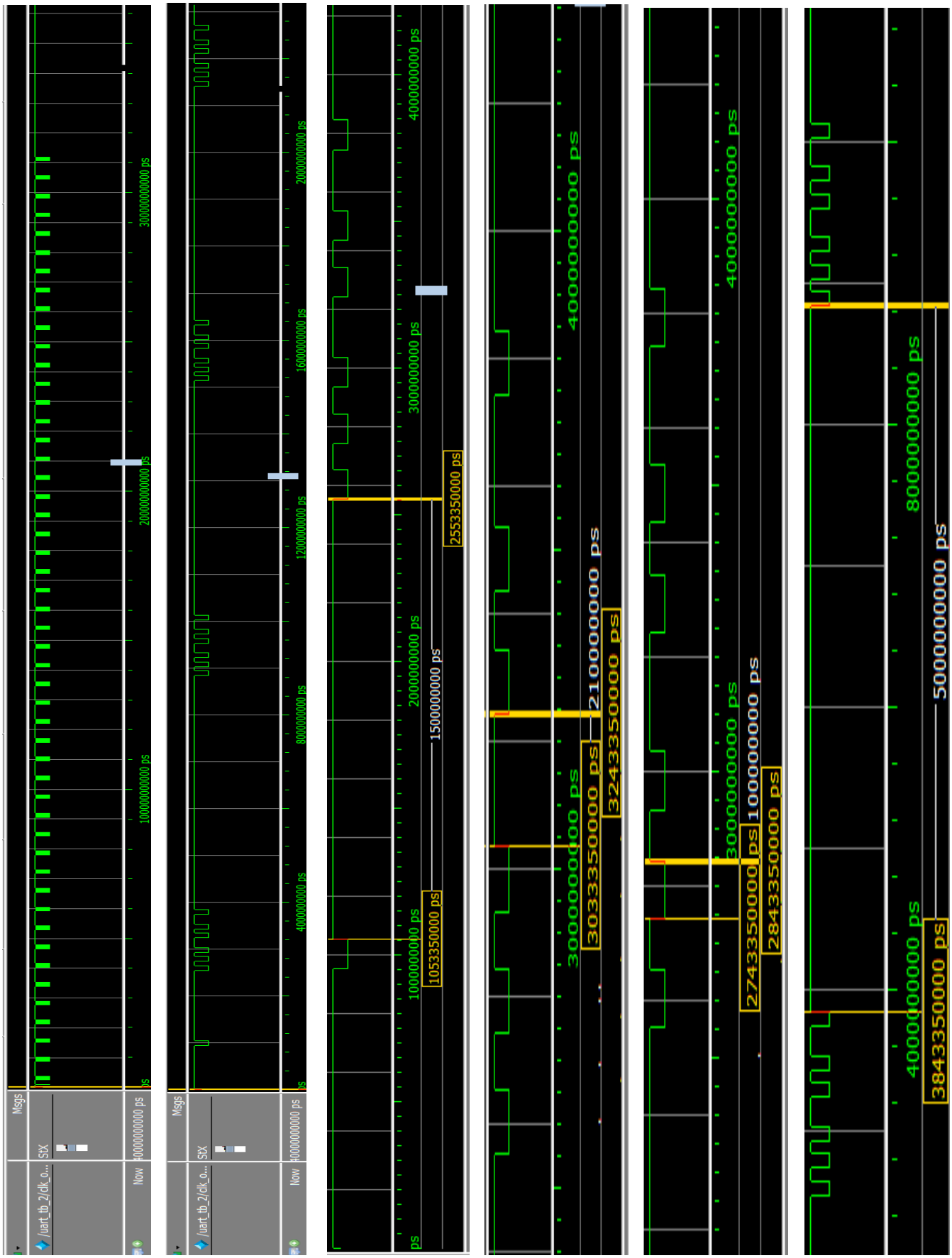
5.7 Test Case

➔ The following table elucidates the test case deployed to test the Verilog model design.

Table 5.2: Test case Panout

Parameters	Specifications	Requirements	Serial Inputs 10MHz Clock
Telegram + NFC Presence + Reserved	6 bits + 1 bit + 1 bit respectively	HIGH – Data 3 and Data 6, NO NFC	HEX 64
Number of telegram repetition	8 bits value	50 TIMES	HEX 32
ON duration of Telegram's HIGH bit	Clock * 16 bits	210 microseconds	7-0 : HEX 34 15-8 : HEX 08
ON duration of Telegram's LOW bit	Clock * 16 bits	90 microseconds	7-0 : HEX 84 15-8 : HEX 03
Duration of 'after impulse HIGH' region	Clock * 16 bits	1.5 milliseconds	7-0 : HEX 98 15-8 : HEX 3A
Duration of HIGH between successive telegrams	Clock * 16 bits	5 milliseconds	7-0 : HEX 50 15-8 : HEX C3

5.7.1 Test Case Results



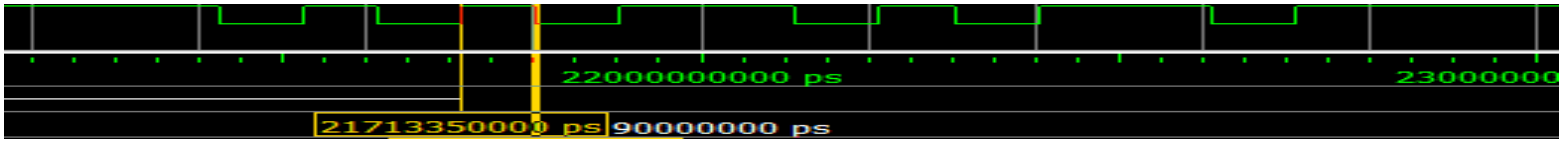


Figure 5.6: Test Case Results; Elucidated below in Left to Right manner

- The following is the elucidation of the above graphs, listed out in a Left to Right fashion, along with the measured reading of the parameters. Each graph is linked to one of the telegram's variable parameters as shown in section 5.1.3.
    - **Zoomed Out View : 50 Number of Repetitions**
    - **Zoomed IN View : HIGH – Data 3 and Data 6 , NO NFC**
    - **Region of HIGH between Impulse and First Telegram : 1.5 milliseconds**
    - **ON duration of Telegram's HIGH bit : 210 microseconds**
    - **OFF duration of Telegram's LOW bit : 100 microseconds**
    - **Duration of HIGH between successive telegrams: 5 milliseconds**
    - **ON duration of Telegram's LOW bit : 90 microseconds**
- The output at the clk_out_clk_out wire, connected to the board's ADC GPIO pin, yields the above displayed signals based on the serial inputs as shown in Table 5.2. The output obtained matches precisely with the expected output, thus validating the design logic.

5.8 Verilog implementation in MAX10

- Top model of the custom designed Verilog library will be flashed into the MAX 10 board via a INTEL compiler and blaster that coverts the Verilog code into a Digital Logic Design and blasts it onto the FPGA chip.
- The top module has 2 inputs and 1 output.

  Inputs: 1. Clock

  2. Serial receive line

  Output: Telegram
- The compiled Digital Circuit will have the above inputs and outputs.

- The clock input to the chip will either flow in directly from the 50Mhz oscillator chip onboard the MAX 10 board, or a PLL modified clock. Thus, the clock input parameter will be connected to the system clock.
- The serial receive INPUT line, represented by the Rx_Serial parameter of the top model, will be connected to the receive pin of the UART RS232 module.
- The telegram OUTPUT line of the Verilog model will be connected to an ADC-GPIO pin.

5.9 Hardware Implementation of MAX 10

- The UART RS232 external module will be connected to the PMOD outlet of the MAX 10 board.
- The ADC-GPIO pin will be connected to the ECU.
- The PMOD UART RS232 external module supplied by Digilent has 6 pins. Namely, Power, ground, Receive, transmit, Clear to Send and Ready to Send .
- It easily fits into the PMOD port (12 pin) of the MAX 10 board.
- The PMOD port onboard MAX 10 board, has 10 I/O pins and a pair of ground-power pins.
- The Digilent Pmod RS232 (Revision B) converts between digital logic voltage levels to RS232 voltage levels. The RS232 module is configured as a data communications equipment (DCE) device. It connects to data terminal equipment (DTE) devices, such as the serial port on a PC, using a straight-through cable.
- The Pmod RS232 utilizes the Maxim Integrated MAX3232 transceiver to allow the system board to communicate with UART compatible PMOD port onboard MAX 10 dev board.

Chapter 6: Single-Port Triple Speed Gigabit Ethernet Communication [Full Duplex]

6.1 Introduction

This design demonstrates Triple Speed Ethernet solution for MAX 10® device family using Altera® Triple Speed Ethernet MAC and Marvell 88E1111 PHY chip on MAX 10 FPGA Development Kit. In this design, the Single-Port Triple-Speed Ethernet MAC connects to the on-board PHY chip through the Reduce Gigabit Media Independent Interface (RGMII).

- The design offers the following features:
  - Supports programmable test parameters such as number of packets, packet length, source and destination MAC addresses, and payload-data type.
  - Supports sequential random burst test, which enables the configuration of each burst for the number of packet, payload-data type, and payload size. A pseudo-random binary sequence (PRBS) generator generates the payload data type in fixed incremental values or in a random sequence.
  - Demonstrates transmission and reception of Ethernet packets through internal loopback path at the maximum theoretical data rates without errors.

6.2 System Architecture

The design demonstrates fully operational subsystems that integrate the Triple-Speed Ethernet Core for Ethernet applications. Figure below shows a top-level block diagram of the design implementing the RGMII interface.

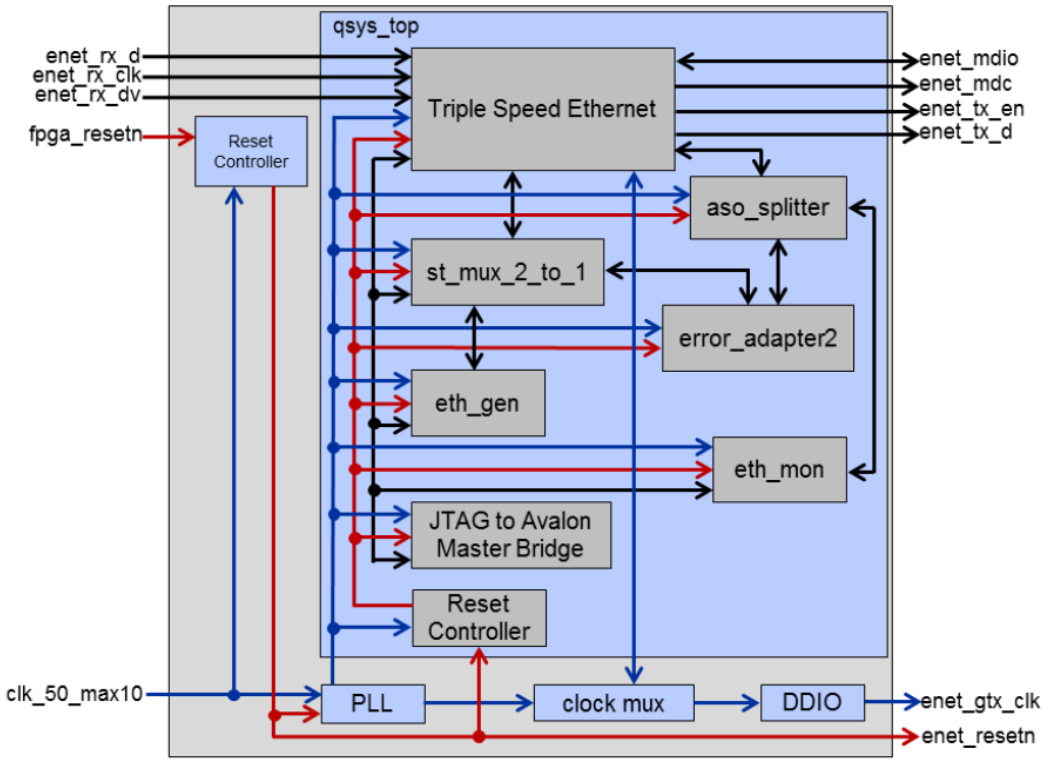


Figure 6.1: Design Simplified Block Diagram

6.3 Design Components

Table 6.1: Design Components deployed and their Description

Component	Description
Triple Speed Ethernet	**Triple-Speed Ethernet MegaCore Function** This IP core is configured for 10/100/1000 Ethernet MAC function to handle the flow of data between user applications and Ethernet network through an external Ethernet PHY.
st_mux_2_to_1	**Avalon-ST Multiplexer** This Qsys custom component accepts data on its two Avalon-ST sink interfaces and multiplexes the data for transmission on its Avalon-ST source interface. One Avalon-ST sink interface connects to the source of the Ethernet Packet Generator for forward loopback while the other sink interface connects to the source of the Error Adapter for reverse loopback. The Avalon-ST source interface sends Ethernet packets to the Triple-Speed Ethernet IP Core.

aso_splitter	**Avalon-ST Splitter** This Qsys custom component accepts data on its Avalon-ST sink interface and splits the data for transmission on its two Avalon-ST source interfaces. The Avalon-ST sink interface receives Ethernet packets from the Triple-Speed Ethernet IP Core. One Avalon-ST source interface connects to the sink of the Ethernet Packet Generator for forward loopback while the other source interface connects to the sink of the Error Adapter for reverse loopback.
error_adapter2	**Error Adapter** This Qsys custom component connects mismatched Avalon-ST source and sink interfaces. The adapter allows the developer to connect a data source to a data sink of differing byte sizes. For TX-to-RX Avalon-ST reverse loopback in this design, ff_tx_err is a 1-bit error signal while rx_err is a 6-bit error signal. The adapter ensures that the per-bit error information provided by ff_tx_err at the source interface connects correctly to the rx_err signal. The adapter connects matching error conditions that are handled by the source and the sink.
eth_gen	**Ethernet Packet Generator** This Qsys custom component generates Ethernet packets. Figure 3 shows a high-level block diagram of the Ethernet Packet Generator module. This module contains the following components: Avalon-MM registers, Ethernet packet generation block, CRC Generator, and Shift Register (RAM-based) megafunction.
eth_mon	**Ethernet Packet Monitor** This Qsys custom component verifies the payload of all receive packets, indicates the validity of the packets, and collects statistics about each packet, such as the number of bytes received. This module contains the following components: CRC Checker and Avalon-MM registers.
JTAG to Avalon Master Bridge	**JTAG to Avalon Master Bridge Core** This IP core provides a connection between the System Console and Qsys system through the physical interfaces. The System Console can initiate Avalon Memory-Mapped (Avalon-MM) transactions by sending encoded streams of bytes through the bridge's physical interfaces
Reset Controller	**Reset Controller** This module is used to synchronize and generate signals as per design requirements
PLL	**Phase-Locked Loop (PLL) Core** This IP core takes an input clock from a 50-MHz crystal on the development board and generates a 125MHz, 25MHz and 2.5MHz PLL output clock (clk_125M). The 125MHz output clock is the system-wide clock source for the Qsys system. All the components in the design use the 125-MHz clock from the PLL core.

clock mux	**Clock Multiplexer** This module is driven by the clock selector signals from Triple Speed Ethernet IP Core to select the 125MHz/25MHz/2.5MHz transmit reference clock for RGMII
DDIO	**Double data rate input/output** This digital component doubles the data-rate of a communication channel. This module is inserted to minimize the clock skew.

6.4 Ethernet Packet Generator

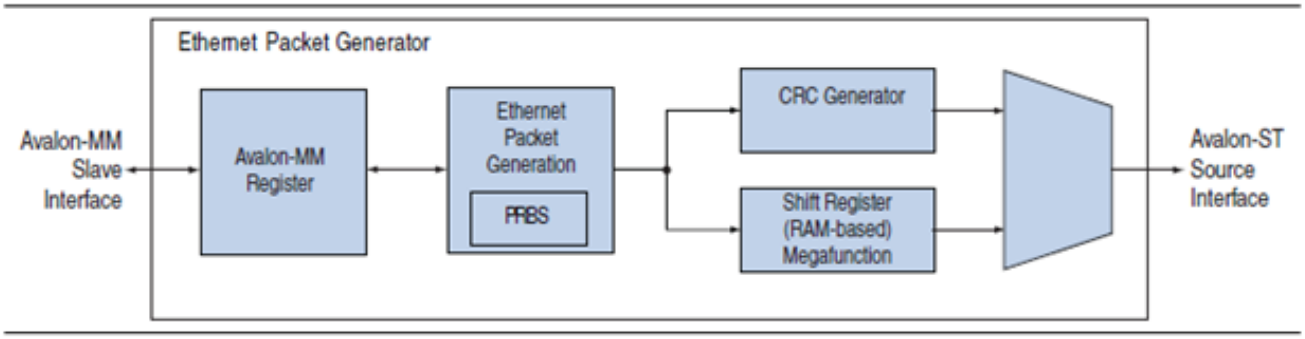


Figure 6.2: Ethernet Packet Generator Block Diagram

The Avalon-MM slave interface provides access to the Avalon-MM register interface. Using a Tcl script, a user can configure the Avalon-MM configuration registers to specify the following:

 Number of packets to generate.

 MAC source and destination addresses—can be unicast, multicast, or broadcast addresses. If the user chooses to use unicast addresses, set the source address to the address of the transmitting MAC and the destination address to the address of the receiving MAC. This setting ensures that the receiving Ethernet port in the design receives valid packets.

The Avalon-ST source interface streams Ethernet packets in the format shown in Figure below. The generated packets do not include the 7-byte preamble, 1-byte start frame delimiter (SFD) and 4-byte MAC-calculated Frame Check Sequence (FCS) fields.

 Packet length—we can specify a fixed or random packet length. A fixed length ranges between 24 to 9,600 bytes; a random length ranges between 24 and 1,518. The Triple-Speed Ethernet IP core pads undersized packets to meet the minimum required length, 64 bytes.

 Payload—we can specify the data type to be incremental or pseudo-random. Incremental data starts from zero and gets incremented by one in subsequent packets. The PRBS block generates the pseudo-random data.

 Random seed—specify the random seed used by the PRBS block to generate the pseudo-random data.

The Avalon-MM status registers provide the status of the transmit operation and report the number of packets that were successfully transmitted.

The Ethernet packet generation block generates an Ethernet packet header, data payload and running sequence number for each packet. The Ethernet packet generation block sends the packets to the CRC Generator and the RAM-based shift register mega function.

The CRC Generator calculates the CRC-32 checksum for the packet and the

RAM-based shift register mega function stores the packet until the checksum is available. After the generator merges the valid CRC-32 checksum with the packet stream, it sends the complete packet to the Avalon-ST source interface.

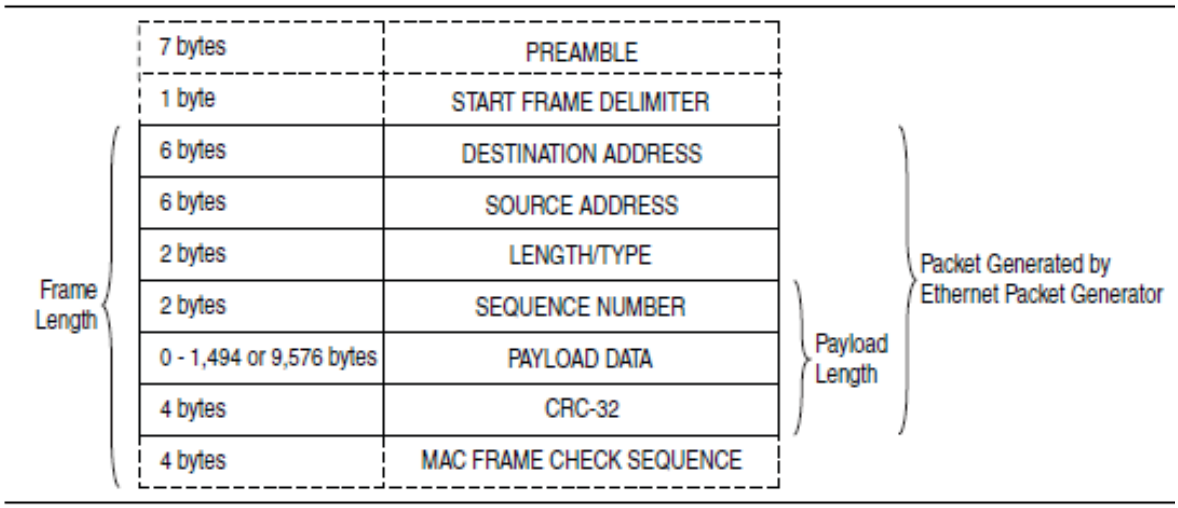


Figure 6.3: Ethernet Packet Generator Output Frame Format

6.5 Ethernet Packet Monitor

This Qsys custom component verifies the payload of all receive packets, indicates the validity of the packets, and collects statistics about each packet, such as the number of bytes received. Figure below shows a high-level block diagram of the Ethernet

Packet Monitor module. This module contains the following components: CRC Checker and Avalon-MM registers.

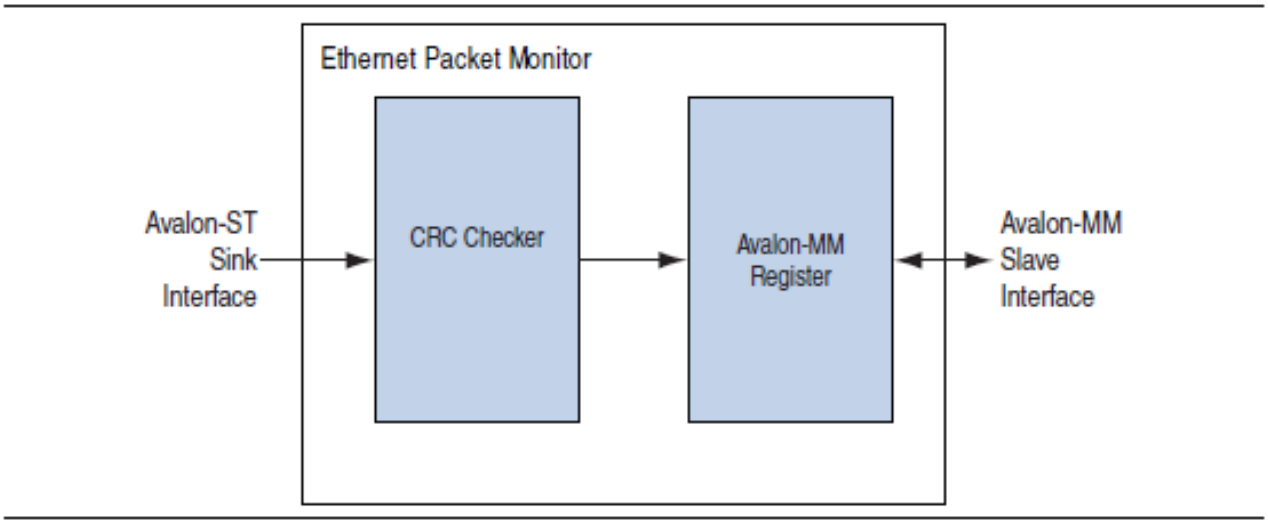


Figure 6.4: Ethernet Monitor Block Diagram

The Avalon-ST sink interface accepts Ethernet packets and sends the packets to the CRC Checker.

The CRC Checker computes the CRC-32 checksum of the receive packet and verifies it against the CRC-32 checksum field in the packet. It then outputs a status signal that identifies whether the packet received is good or corrupted, and updates the statistics counters accordingly.

The Avalon-MM slave interface provides access to the Avalon-MM register interface.

Using a Tcl script, one can configure the Avalon-MM configuration registers to specify the number of packets the monitor expects to receive. The Avalon-MM status registers provide the status of the receive operation and report the number of good and bad packets received, the number of bytes received, and the number of clock cycles. This information is used to calculate the performance and throughput rate of the design.

Chapter 7: Conclusion and Future Scope

Specificity of the objective of this project with respect to the domain of Hardware-in-the-Loop testing, has been narrated at the beginning of this document. This document craftily and subtly unfolds from giving an account of Hardware-in-the-Loop testing systems to Intel's MAX 10 FPGA to Mercedes-Benz's Keyless-Go system to RS232 UART based design and implementation and finally to the Single-Port Triple Speed Gigabit Ethernet Communication aspect. Please refer to section 5.6.1 to look through the results pertaining to the test case deployed to test the functioning of the RS232 UART based Verilog FPGA design that generates the door handle telegram with user controllable parameters in order to simulate and test the functionalities of the door handle.

Ethernet communication has been tested to its theoretical speed limits, but unfortunately, the integration of the protocol with the telegram generation model could not be included in the scope of this project. In the future, a full-fledged gigabit Ethernet communication can be deployed to generate and transmit the telegram signals to the Keyless-Go ECU, in order to achieve a high speed simulation of the door handle on a Hardware-in-the-Loop platform.

In the scope of this project, a RS232 based UART communication model was successfully deployed and tested on a MAX 10 FPGA based board simulation, to achieve an optimal simulation of the door handle, with inconsequential limitations on the speed, efficiency and the baud rate. In the future, the Verilog model can be improved upon to accommodate additional controllable parameters like the OFF time of a HIGH bit or a LOW bit, amongst many others.

But this project acts as a PoC for the company to further investigate the realm of cost effective FPGA based custom Hardware-In-The-Loop solution development.

REFERENCES

Mercedes-Benz Internal Documents [Confidential]

Journal / Conference Papers

[1] **N. Alachiotis, S. A. Berger and A. Stamatakis, "Efficient PC-FPGA Communication over Gigabit Ethernet," *2010 10th IEEE International Conference on Computer and Information Technology*, Bradford, 2010, pp. 1727-1734. doi: 10.1109/CIT.2010.302**

[2] M. Bacic, "On hardware-in-the-loop simulation," *Proceedings of the 44th IEEE Conference on Decision and Control*, 2005, pp. 3194-3198. doi: 10.1109/CDC.2005.1582653

[3] W. Grega, "Hardware-in-the-loop simulation and its application in control education," *Frontiers in Education Conference, 1999. FIE '99. 29th Annual*, San Juan, PuertoRico, 1999, pp. 12B6/7-12B612 vol.2. doi: 10.1109/FIE.1999.841594

[4] K. Fang, Y. Zhou, P. Ma and M. Yang, "Credibility evaluation of hardware-in-the-loop simulation systems," *2018 Chinese Control And Decision Conference (CCDC)*, Shenyang, 2018, pp. 3794-3799. doi: 10.1109/CCDC.2018.8407782

[5] C. Liu, R. Ma, B. Hao, H. Luo, F. Gao and F. Gechter, "FPGA based hardware in the loop test of railway traction system," *2018 IEEE International Conference on Industrial Electronics for Sustainable Energy Systems (IESES)*, Hamilton, 2018, pp. 206-211. doi: 10.1109/IESES.2018.8349875

[6] T. Le, F. M. Renner and M. Glesner, "Hardware in-the-loop simulation-a rapid prototyping approach for designing mechatronics systems," *Proceedings 8th IEEE International Workshop on Rapid System Prototyping Shortening the Path from Specification to Prototype*, Chapel Hill, NC, 1997, pp. 116-121. doi: 10.1109/IWRSP.1997.618886

[7] S. Wakitani and T. Yamamoto, "Practice of model-based development for automotive engineers," *2017 IEEE Frontiers in Education Conference (FIE)*, Indianapolis, IN, 2017, pp. 1-4. doi: 10.1109/FIE.2017.8190678

[8] S. U. Jonwal and P. P. Shingare, "Advanced Encryption Standard (AES) implementation on FPGA with hardware in loop," *2017 International Conference on Trends in Electronics and Informatics (ICEI)*, Tirunelveli, 2017, pp. 64-67. doi: 10.1109/ICOEI.2017.8300776

[9] C. Washington and J. Dolman, "Creating next generation HIL simulators with FPGA technology," *2010 IEEE AUTOTESTCON*, Orlando, FL, 2010, pp. 1-6. doi: 10.1109/AUTEST.2010.5613618

[10] M. Ruba, R. Nemes and C. Martis, "FPGA Based Real-Time Electric Power Assisted Steering Motor-Drive Simulator Designed for HiL Testing in the Automotive Industry," *2017 IEEE Vehicle Power and Propulsion Conference (VPPC)*, Belfort, 2017, pp. 1-6. doi: 10.1109/VPPC.2017.8331024