Failsafe ECU through dynamic partial-reconfiguration in FPGAs

Constantin Schieber, 01228774
Rupert Schorn, 01325700
Andreas Hirtenlehner, 01327273
Peter Schober, XXXXXX

March 3, 2019

1 Introduction

In this lab, fail-safe mechanisms for Electronic Control Units (ECUs) are explored on the basis of Partial Reconfiguration (PR) in Field Programmable Gate Arrays (FPGAs).

The introduced design contains typical characteristics of an automotive system. Communication between the different sub-systems is realized by a specific link (including protocol) and connects the FPGA to the ECUs that handle operations during normal conditions. ECUs are monitored by a bus monitor in the FPGA for nominal operation characteristics and can be replaced by the instantiation of the very same ECU within the FPGA in case of failure.

Our ECUs are based on ARM Cortex-M1/M3 controllers. Both of them became recently available as an intellectual property (IP)-package for Xilinx based FPGAs. By using a common hardware base and a common software middleware we were able to streamline the development of the control-software which enables a faster development process for new applications and reduces the overhead of their functional verification.

Brief overview, problem statement and final outcome.

1.1 Design Overview

The assumed system consists of three regular separate control units that are connected through a communication link. Figure 1 shows the basic components of the assumed design.

The realized scenario reads a throttle position and controls an engine after some data conversion. The ECU is responsible for gathering the throttle position data, measured and provided by the throttle sensor (THS). After the ECU converts the throttle position data into engine control data, it forwards these data to the motor control unit (MCU), which is responsible for controlling the engine. A fourth unit acts as a fallback unit in case of a failure of one of the regular control units. This fallback unit is realized by using PR in an FPGA. Figure 2 displays a sequence diagram assuming normal operation. In this case, the fallback unit doesn't have to perform any active functionality, it just has to passively monitor the data transfer on the communication link to detect possible errors.

The fallback unit has to monitor all transferred data on the bus and detect any failures (e.g. timeout, error flags, ...). Once an error is detected, a certain partial reconfiguration is triggered by the bus monitor module, which is part of the fallback unit. After the reconfiguration is finished, the fallback unit completely takes over the functionality of the faulty component. Due to the fail silent assumption, the faulty device will not affect the behaviour of the system. Figure 3 shows a sequence diagram including a faulty MCU. The bus monitor detects that the MCU is running into a timeout and triggers a PR to take over

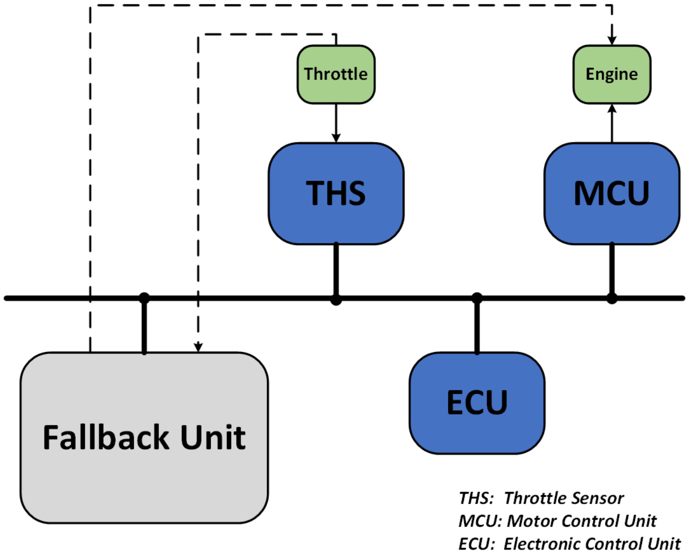

Figure 1: Basic concept for Failsafe ECU

its functionality. After finishing the reconfiguration, normal operation takes over again (see figure 2), but the fallback unit serves as MCU now.

1.2 Required tools, intellectual properties (IPs) and packages

TODO: licenses

- Vivado 2018.3
- Vivado 2018.2
- uVision 5 (Windows only)
- ARM Keil (Windows only)
- ARM Cortex M1 IP for Vivado
- UART IP by Martin Mosbeck
- ARM Mbed OS

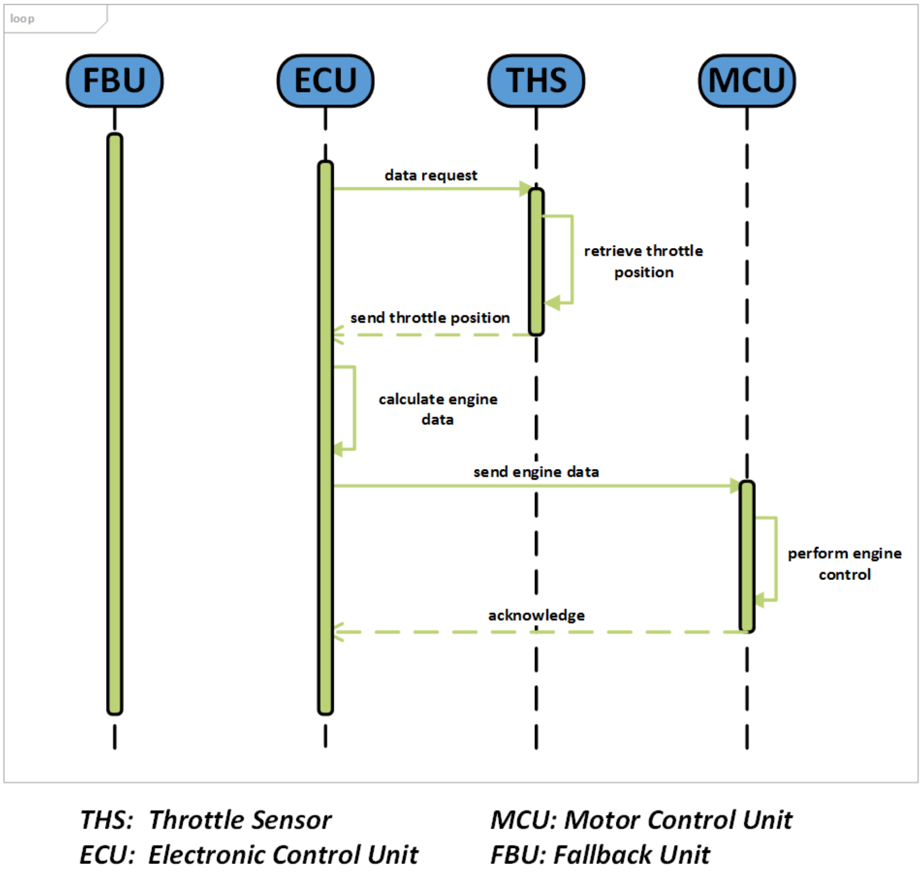

Figure 2: Sequence diagram for normal operation

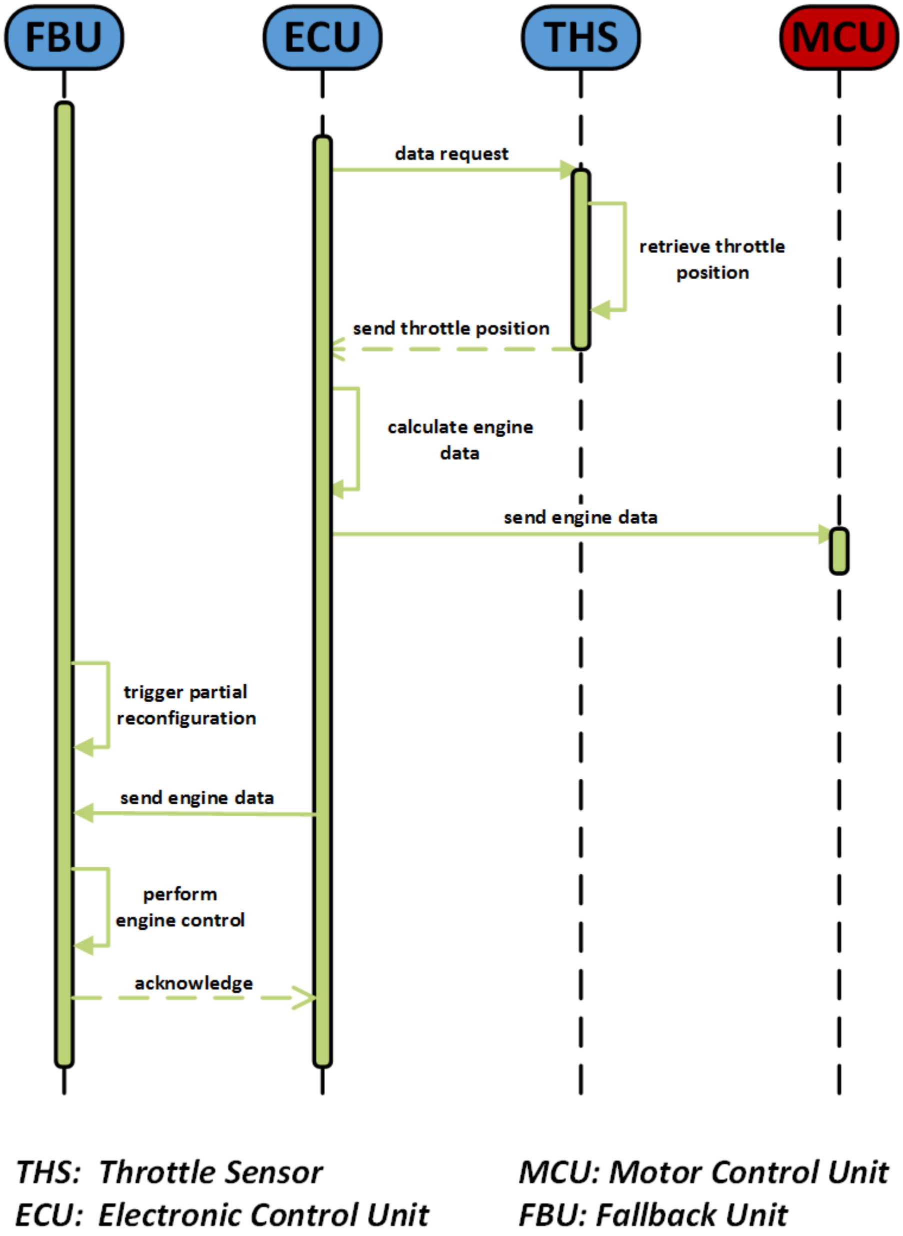

Figure 3: Sequence diagram for MCU failure

2 Cortex M1 / M3

Description of the setup of Cortex M1 and Cortex M3 Cores. [1] [2]

2.1 Usage of the Keil Toolchain

Link to tutorial enough? Maybe most basic steps (e.g. adaption of arty project for our needs). [3]

2.2 Usage of Cortex M1 in Vivado

Only global implementation / synthesis runs are permitted to obtain a working bitstream. If OOC (Out of Context) runs are used, everything except for the Cortex M1 will work fine. The Cortex M1 will enter a hard-fault state which does not allow recovery. This is indicated by a high bit on the *Lockup* port of the processor.

Also trivial, follow tutorial that is provided on ARM Website and adapt to own needs.

2.2.1 Code via Memory Initialization File

File is bound to synthesis process, how to change it...

2.3 How to mbed OS

How was the Cortex M1 Project adapted to support the Mbed OS? What is gained through the usage of mbed OS?

2.4 Implemented Functionality on the Cortex M1

How are the UART and IIC peripherals supported in the source code, which libraries are used, interrupt based or not, performance?

Show highlights of the code?

3 Bus and Peripherals

This section describes the used evaluation boards and bus transceivers for the test setup. Further, the used communication protocol is explained.

3.1 Used Peripherals

What Peripherals were used (Cortex M1/M3 Boards, which ones).

How to program / use them (only brief).

How are they connected to the Bus?

IIC and UART here or in next section.

3.2 Bus

How is it set up?

Made assumptions?

Document used / invented protocol.

4 Partial Reconfiguration

This section reasons about design choices and encountered obstacles during the development process.

4.1 Limitations imposed by partial reconfiguration

PR does impose some limitations on the design process, a brief description of the encountered limitations and how they were handled is given in the following.

4.1.1 No block diagram support

The PR workflow as implemented by Xilinx in Vivado does not allow the Reconfigurable Partition (RP) to be present in a block diagram. Only hdl files are eligible for the PR process.

To solve this problem without the loss of comfort that is provided by the usage of block diagrams (mainly the connection of different signals between modules) we decided to transfer our Cortex Module into an IP-Package. This IP-Package can then be instantiated as a register-transfer-level (RTL) module alongside an existing block diagram. Only the signal connections between the IP and the block diagram have to be declared manually then.

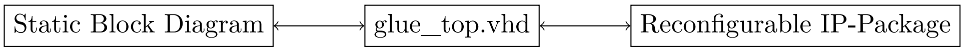

Figure 4: Enable PR by the separation of the block diagram and the RP

Due to this, the originally planned AXI-Bus interface for the RP needs to be connected manually. This has proven to be very error-prone in earlier evaluation steps and was then replaced by an interface that only exposed required signals. An AXI-Bus interface may very well work though, as it probably failed due to other, unrelated errors.

4.1.2 Global synthesis / implementation runs are mandatory

PR works with global synthesis and implementation runs only. This imposes restrictions on the naming of files within packages. While out-of-context (OOC) runs allow the same instance names for different IPs a global run requires distinct file and instance names for everything that is included in the project.

Global synthesis and implementation runs are also necessary to include the memory initialization file for the Cortex processor.

4.1.3 processor configuration access port (PCAP) / internal configuration access port (ICAP) on the Zynq-7000

As the Zynq-7000 was used for the prototyping process our first choice was the usage of the PCAP for writing partial bitstreams to the FPGA as it is the most straight forward (and well documented) way for this platform.

Due to design considerations (scaling well for production vs fast prototyping) a switch to using the ICAP was made. For this, the PCAP needs to be actively deactivated after the boot of the processing system ([4], page 218).

4.1.4 ICAP primitive instantiation

Instantiation of the ICAP as a hardware primitive in VHDL is documented in the 7series libraries guide ([5], page 178). For completeness sake, the component definition is listed below as it does not exist directly in the documentation. The comments the generic parameters that were used in this project.

```
component ICAPE2 is
generic (
    DEVICE_ID    : std_logic_vector(31 downto 0); -- X"23727093"
    ICAP_WIDTH   : string; --"X32"
    SIM_CFG_FILE_NAME    : string -- "None"
);
port (
    O            : out std_logic_vector(31 downto 0);
    CLK          : in std_logic;
    CSIB         : in std_logic;
    I            : in std_logic_vector(31 downto 0);
    RDWRB        : inout std_logic
);
end component ICAPE2;
```

Listing 1: Component definition of the ICAPE2 hardware primtive

4.1.5 Read from SD-Card

The SD-Card was used to provide a bootable image with the default configuration and all partial bitstreams. Reads from the SD-Card may fail, resulting in a non-functioning or only partially functioning system. The following steps should be performed to trouble-shoot the problem.

- If the system is working in its original configuration and only fails the partial reconfiguration, the file names of the bitstreams should be checked. Only a maximum of 8 characters (+3 for the extension) for the file name are permitted by default.
- Slow (e.g. overwriting everything with zeros) reformat of the SD-Card may be tried.
- Smaller SD-Card should be tried.

4.2 Integration Overview

Due to our problems with the AXI-Bus it was decided to put all relevant peripheral communication into the RP. This eliminates the need of implementing an AXI-Interface at the cost of an increased size of the RP.

The following modules are included in the RP:

- AXI-GPIO
- AXI-UartLite
- AXI-Timer
- AXI-IIC
- Cortex M1

[6], [7] Usage of ICAP. How is the Zynq still used - Loading images and binary blobs from sd card into DDR. What is partially reconfigured - Cortex, uart, IIC. Why not use AXI?

4.2.1 Packaged IP

The PR IP was packaged according to the guide in [8]. To avoid naming conflicts, all block designs of the different IPs need to be named differently before this process.

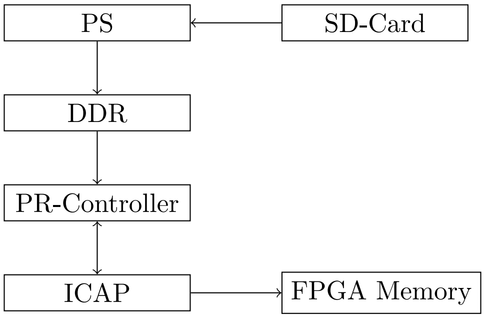

Figure 5: Data path for loading a partial bitstream

4.2.2 Loading the PR module

The module is loaded through the PR-Controller IP that is provided by Vivado. This module is connected via the AXI memory interface and loads the partial bitstream from DDR. At start-up, the partial bitstreams are loaded form the SD-Card into the DDR through the Zynq Processing System (PS) as it already needs to be used for the deactivation of the PCAP. Figure 5 shows how the different modules and memory entities are related to each other.

5 Results

6 Conclusion

This work has explored the possibility of using partial reconfiguration in FPGAs to provide efficient redundancy in an automotive system. Instead of providing a redundant hardware entity for every critical module, one single FPGA provides dynamic redundancy for each of these modules. To avoid over-commitment with regards to resource usage (space and power) partial reconfiguration is used. By using the newly available Cortex M1 IPs, a streamlined software development process is possible. The same software can be executed on the cores in the FPGA as well as on the actual hardware with minimal adaptions in the build-process. This reduces the amount of testing and tool-chain adaptions that need to be performed. We demonstrated these concepts on the Zynq-7000 and with three Cortex M1 Central Processing Units (CPUs) that were connected with a bus.

6.1 Future Work

Based on this work a more heterogeneous set of critical hardware could be provided with redundancy. A good first step would be to include the Cortex M3 CPU that couldn't be included in this project due to time constraints. The bus monitor could be extended with a more sophisticated fault detection algorithm, which could also mean to employ a more sophisticated bus protocol. Measurements with regards to the system performance (e.g. time to reconfigure, time to detect fault, time to mitigate fault, power usage ...) should be considered also.

References

[1] ARM, "Arm® Cortex®-M1 DesignStart™ FPGA-Xilinx edition User Guide," 2018. [Online]. Available: https://static.docs.arm.com/100211/0001/arm_cortex_m1_designstart_fpga_xilinx_edition_ug_100211_0001_00_en.pdf?_ga=2.121336076.523725493.1550477241-1711067005.1549732081

[2] ——, "Cortex-M1 Technical Reference Manual." [Online]. Available: https://static.docs.arm.com/ddi0413/d/DDI0413D_cortexm1_r1p0_trm.pdf

[3] "Getting Started." [Online]. Available: http://www2.keil.com/mdk5/install

[4] Xilinx, "Zynq 7000 Technical Manual," 2018. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf

[5] ——, "7series libraries guide," 2018. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_2/ug953-vivado-7series-libraries.pdf

[6] ——, "Vivado Design Suite User Guide: Partial Reconfiguration (UG909)," 2018. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug909-vivado-partial-reconfiguration.pdf

[7] ——, "Vivado Design Suite Tutorial: Partial Reconfiguration (UG947)," 2018. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documentation/sw_manuals/xilinx2018_3/ug947-vivado-partial-reconfiguration-tutorial.pdf

[8] ——, "ug1118-vivado-creating-packaging-custom-ip.pdf." [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug1118-vivado-creating-packaging-custom-ip.pdf