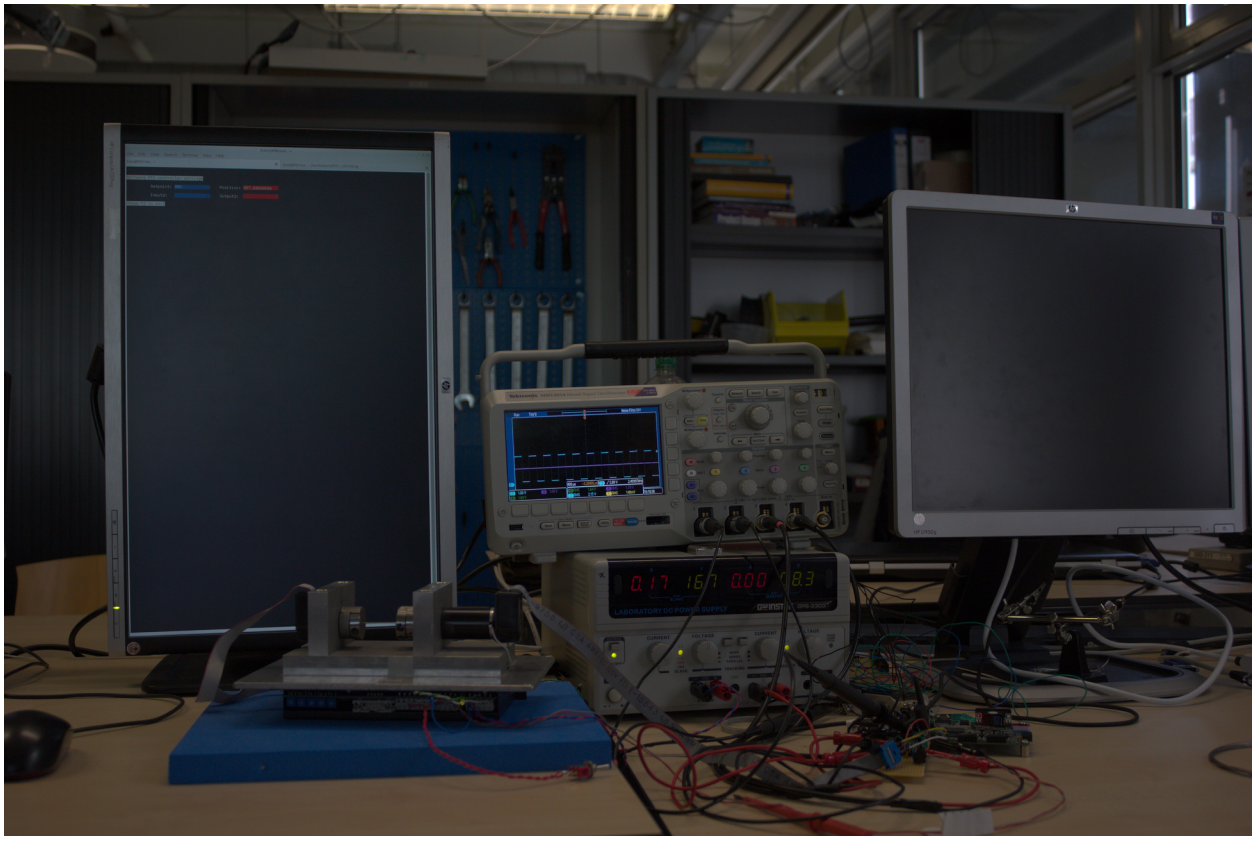
Distributed Control System

FPGA accelerated PID controller

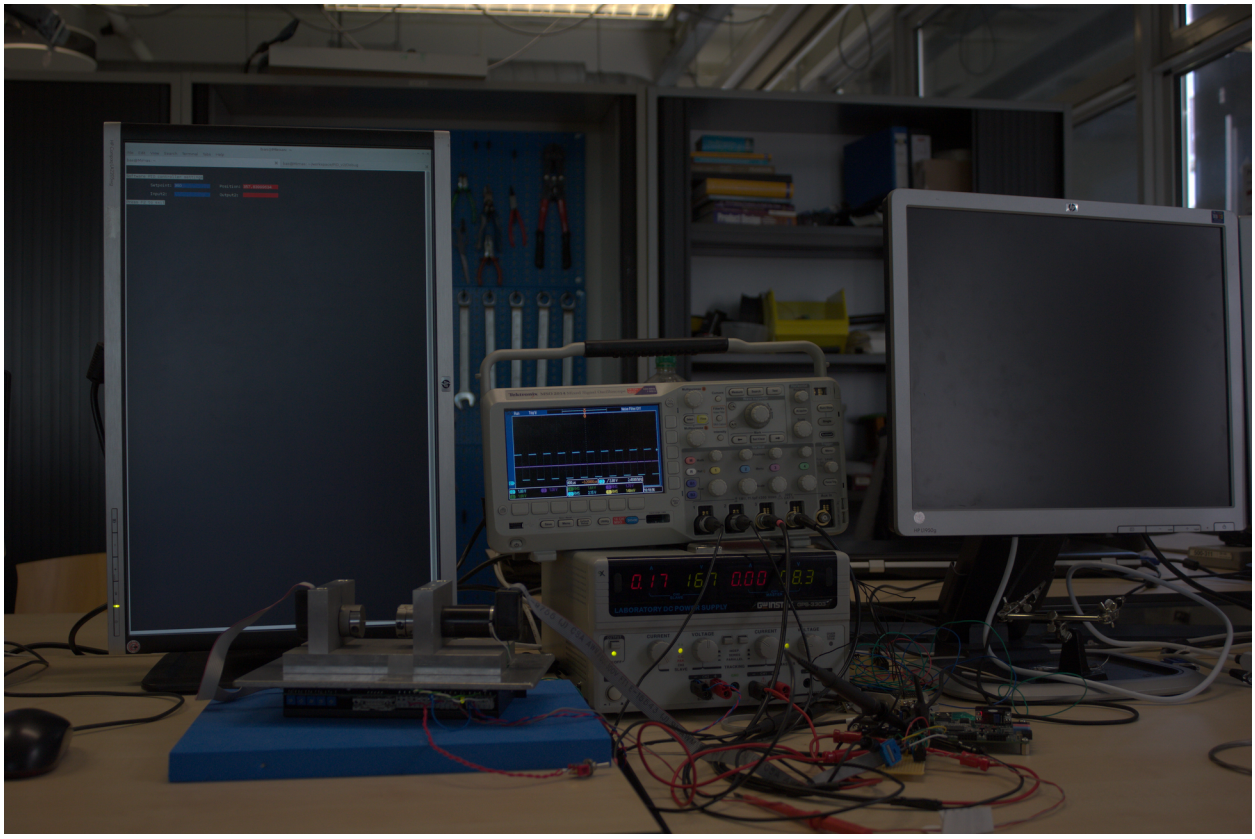


Bas Janssen
2206481
Fontys Hogeschool Engineering
Lectoraat Robotics & High Tech Mechatronics
2016
Verion: 1.5

PAGE INTENTIONALY LEFT BLANK

Distributed Control System

FPGA accelerated PID controller



Bas Janssen
2206481
Fontys Hogeschool Engineering
Lectoraat Robotics & High Tech Mechatronics
Eindhoven
Cohort M12
Mentor: P. Jacobs
17-06-2016
Version 1.5

Contact information

Fontys Hogescholen Engineering
Kenniscentrum Robotics & High Tech Mechatronics
Rachelsmolen 1
5612MA Eindhoven
Building R1, room 0.205

Client
Jeedella Jeedella
j.jeedella@fontys.nl

Content Supervisor
Mark Stappers
m.stappers@fontys.nl

Author
Bas Janssen
2206481
Cohort 12
0652021237
smh.janssen@student.fontys.nl

Preface

This report is the final report for the FPGA accelerated PID controller, part of the Distributed Control Systems project. This project runs within the Lectoraat Robotics and High Tech Mechatronics of Fontys Hogeschool Engineering Eindhoven. The Lectoraat has the goal to develop applicable knowledge to support education and industry. This knowledge is acquired with projects run in conjunction with the industry. The report will go into detail for the software designed for this project, not the hardware design.

This report is intended for follow up students working on the Distributed Control Systems project. Within this report the assumption is made that the reader is at least familiar with the terms EtherCAT, FPGA, Linux and PID controllers. However for each part a small basic introduction is included. For readers looking for the accomplishments in this project, the results are in chapter six. Following are short descriptions of the chapters in this report.

The first chapter will give a short introduction to the project. It talks about why the project was conceived, where the project was done and what the expected end result is. The second chapter, the problem definition, talks about how the project has been defined, what is included and what is not and how the customer expects the final product to function and look like. The third chapter details the methodology used during this project.

All the research preformed for this project will be described in the forth chapter. This chapter goes into the research into the Xilinx Zynq 7000 chip, Beckhoff's EtherCAT system, how the Serial Peripheral Interface works and how a PID controller functions.

Following in chapter five the design is expanded upon. First the toolchain for building for the Zynq chip is explained. This is followed by and explanation of the different software parts that have been designed.

Finally chapters six and seven provide the results and the conclusions and recommendations for this project.

I would like to thank mister J. Jeedella, mister M. Stappers and mister P. Jacobs for their assistance during this project.

Eindhoven, 02-06-2016
Bas Janssen

Summary

Within the Fontys Lectoraat Robotics and High Tech Mechatronics the Distributed Control Systems project runs. This project consists of multiple parts. A previous internship developed a system identification tool. This tool determines the transfer function of the forth order system, and computes a PID controller. The output from the identification tool project is used in this project to configure the PID controller.

For the FPGA accelerated PID controller project, the final goal is to build a PID controller in an FPGA. This controller will then be used to verify the output of the identification tool. For this goal, device drivers have to be designed and written.

By splitting the controller into separate hardware blocks, respectively the encoder module, the PWM driver and the PID controller itself, all of these parts can be reused for different projects. This also opens up the ability to test the modules, and their respective drivers, before the final integrated PID controller.

Testing proved both the PWM driver and module and the encoder driver and module function as required. Using these proven parts a software PID controller running on the Linux environment has been build, showing the functionality of these system parts.

Recommendation for possible follow up projects are:

- Look into getting Ubuntu with ROS running on the Zynq chips. A possible environment can be acquired from the Snickerdoodle project.
- Collaborate with the Fontys ICT studies on the device drivers for custom hardware.
- Start a repository with pre build hardware and corresponding drivers and software to encourage reuse.
- Test the PID controller driver with the actual hardware when the hardware is finished.

Table of Contents

List of figures

List of tables

1 Introduction

This report has been written to detail the design and implementation of the FPGA accelerated PID controller for the distributed control system project. More specifically this report elaborates on the Linux part of this system.

The distributed control system project runs within the Kenniscentrum Robotics and high tech Mechatronics, the research center of Fontys Hogeschool Engineering. Within this center applied research is done to innovate in the areas of education and professionalism, and looks to provide additions to the curricula of the different programs.

Within the larger distributed control system project a system identification tool has been designed, for which a demonstrator is desired. The system identification tool has been designed to compute the transfer function of the entire model including the motor driver. Using this transfer function the system then computes the factors for a PID controller.

The demonstrator should be usable to determine if the controller determined by the identification was successful. The demonstrator has to be build using a System on Chip containing an FPGA and ARM processor cores and should be configurable using the EtherCAT field bus.

2 Problem definition

In a previous project, a system identification tool has been designed to compute the transfer function of a fourth order model (Figure 1). This transfer function can then be used to compute a PID controller to control the system. The output from this previous project are the values used to tune the PID controller.

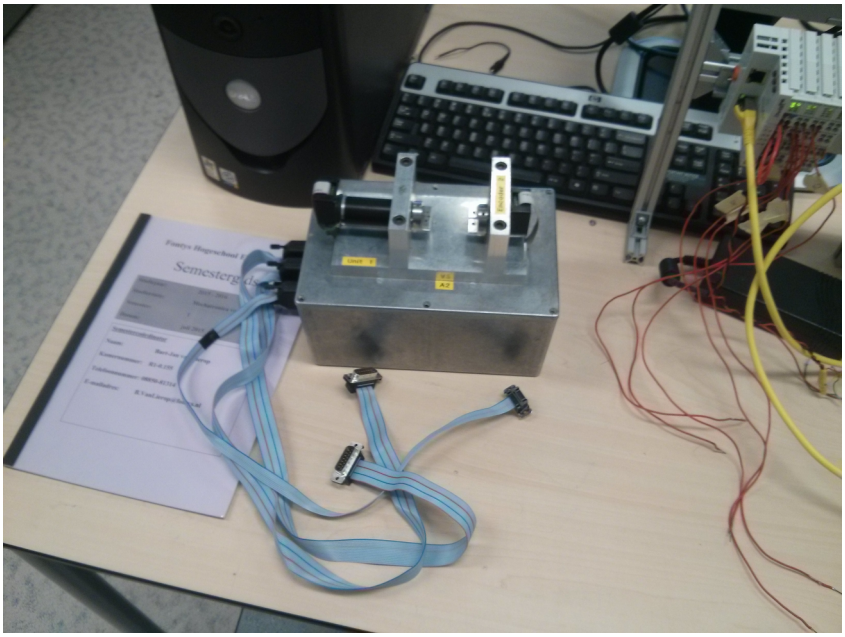


Figure 1: Fourth Order Model
Source: Bas Janssen

This model can be represented schematically as depicted in Figure 2. Here we can see that the system consists of a motor with an encoder and a small mass, connected to a second encoder and small mass with a torsion bar. This causes the input and output, respectively the motor and the second encoder, to not have a linear relation. As the torsion bar can twist the second encoder can lag behind when accelerating or lead when decelerating.

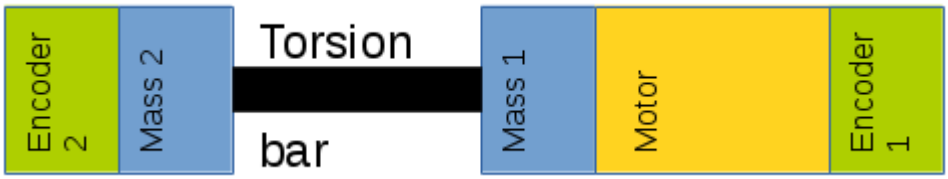


Figure 2: Schematic representation of the forth order model
Source: Bas Janssen

To confirm that the computed controller is correct, a controller has to be build. This verification controller is based around the Xilinx Zynq 7000 SoC and is controllable using EtherCAT. The

PID loop will run in the FPGA of the Zynq and the control software and handling of the EtherCAT network will be done using Linux on the ARM cores of the Zynq.

Figure 3 shows a schematic representation of the entire verification controller. The dotted line shows the part of the system that this report will detail. This includes the control network, EtherCAT, and everything that runs on Linux for the controller.

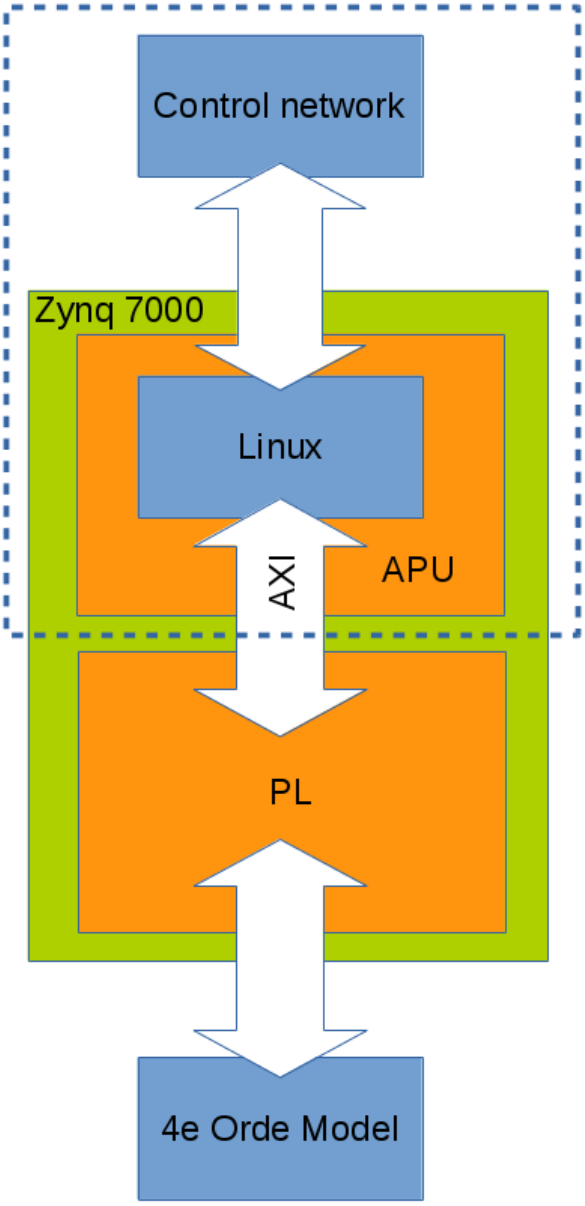


Figure 3: Schematic overview of the system
Source: Bas Janssen

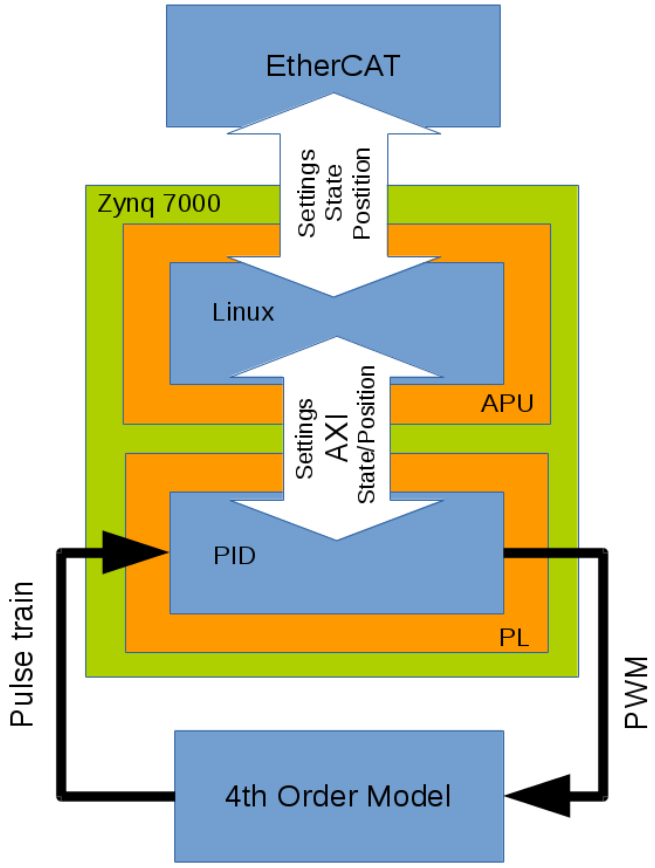


Figure 4: Schematic representation of the entire control loop
Source: Bas Janssen

Figure 4 show a schematic representation of the control loop. The PID controller, running in the FPGA reads the pulse train coming from the encoder and adds up a number indicating the position from the position on power up. To drive the model, the controller outputs a PWM pulse to the motor controller. This controller is part of the model and its black box representation. The Linux application provides the settings and the setpoint for the controller and reads back the state and position data. It also handles the communication with the EtherCAT network, which provides the settings and setpoint, and requests the data read back from the controller.

The final controller build for this project will be used for didactic purposes, to teach students how to tune a PID controller and the effects of using different values for the factors. It will also be used in the system identification course to verify that the controller based on the system verification is correct. The PID controller could also be used to build a self balancing robot, to drive the motors for this robot.

At the end of this project the following items have to be delivered:

- PID driver
- PWM driver
- Encoder driver
- EtherCAT Slave driver
- Final report
- Functional demonstrator

3 Methodology

Since this project is based around software development, the scrum method has been used. During weekly meetings with the tutors, the progress of the week leading up to the meeting has been discussed and new goals for the following week are set. These week long development cycles are equivalent to the sprint in the scrum method. The goals set during the meetings can be represented as the sprint backlog. This system does result in a incremental addition of functionality to the end product.

During the research phase the scrum process was used to set the research goal for the following week. When a new subject to research was found, this was added to the backlog. This allowed for picking new research subjects for the following week.

In the design phase scrum was used to set targets for the functionality in the drivers. After each sprint new functionality should be available in the driver that was being worked on. When implementing a new function took longer than expected, the task is kept in the sprint for the following week.

All of these targets are based upon the planning, added in Appendix 10: Planning.

4 Research

This chapter describes the research into the Xilinx Zynq chip and the EtherCAT network. First the research into the Zynq is described, and afterwards the research for the EtherCAT network. This is followed by a section describing how SPI functions and ends with a basic introduction to PID controllers.

The research into the Zynq chip and the EtherCAT network was preformed to expand the understanding of the technologies used by these systems. This research has also been used to make the choices for SPI and the register layouts for the hardware designs. The research for the SPI interface and the PID controller has been preformed to update the knowledge of both areas.

4.1 Xilinx Zynq 7000

The following paragraphs detail the research into the Xilinx Zynq chip. First the system overview will be described, followed by the ARM-FPGA interconnect. Then the SPI master will be reviewed and lastly the Linux OS on the Zynq will be detailed.

4.1.1 Xilinx Zynq System on Chip

The Xilinx Zynq-7000 System on Chip, abbreviated to SoC, is a chip containing two ARM cores and an Xilinx FPGA in a single package. Xilinx designates the ARM cores as the Application Processing Unit (APU) and the FPGA as Programmable Logic (PL). The APU and PL are interconnected on the chip. The APU and PL also contain interfaces for communication with the outside world. The APU has access to two Gigabit Ethernet interfaces, two SPI controllers, USB, two serial UARTS and general GPIO.

The PL has one hundred I/Os available to connect to peripherals. The Zynq chip also contains an I/O multiplexer which allows the user to route different interfaces to available I/O pins. Figure 5 shows a schematic overview of the SoC with its ARM cores, the FPGA and the available interfaces and interconnects.

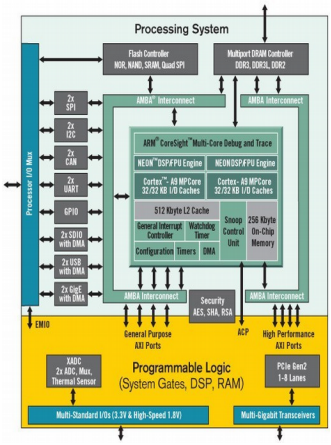


Figure 5: Xilinx Zynq All Programmable SoC
Source: Xilinx.com

4.1.2 ARM-FPGA interconnect

The Zynq chip contains an interconnect between the ARM cores and the FPGA. This interconnect is designated as AXI Ports (Advanced eXtensible Interface) in Figure 5. The AXI interconnect is connected to the AMBA interconnect (Advanced Microcontroller Bus Architecture) for connectivity with the ARM cores. The AXI bus allows for three different interfaces, respectively AXI Lite, AXI (Full) and AXI-Stream.

According to Xilinx [1] the three interfaces have the following specifications:

- The AXI Lite interface has been designed to function in an control register style. This means that all transactions are handled using a burst with a length of one, all registers have the same data width as the bus uses, and exclusive access is not available.
- The AXI interface however allows for a multi master setup and allows for transmissions with a burst length of up to 256 registers.
- The AXI-Stream interface has been designed as a one-way bus from master to slave. This implementation of the bus allows for multiple data streams of different widths over the same interconnect.

4.1.3 SPI

The Zynq chip has access to two SPI controllers based on the Cadence SPI core. These controllers are available as peripherals of the ARM cores. The connections for the controllers, MISO; MOSI; SCK and one or more SS, can be routed by the I/O-mux. This means the can be routed to I/O pins available to the APU using the MIO, or any of the pins of the FPGA using the EMIO[2]

The chips that have been produced, contain a mistake that resets the controller when Slave Select 0 is pulled low. To avoid this problem, Xilinx advises to pull the pin high. To pull the pin high, a

pull-up resistor can be used on the external pins, or when the pins are routed through the FPGA, it can be pulled high within the FPGA[3].

4.1.4 Linux

The following paragraphs detail the boot process for Linux on the Zynq chip, the device tree and Linux device drivers.

4.1.4.1 Boot process

The boot process on the Zynq chips consist of four distinct parts. Figure 6 shows this process in a flowchart. First there is the First Stage Boot Loader (FSBL), which in turn loads the boot loader. The boot loader then loads the Linux kernel and the ramdisk. The FSBL is build using the hardware configuration exported from Vivado. This makes it possible to have the FSBL configure the FPGA using the bitstream and enable the level shifters between the APU and the PL. The FSBL also configures the I/O multiplexer to enable access to the peripherals. When it has completed this step it loads U-Boot as the boot loader for the Linux environment. U-Boot then takes over the boot process and loads the ramdisk and Linux kernel image in to memory. It also hands the device tree blob to the kernel so that the kernel knows where the peripheral are located and can load the drivers for the peripherals. When the kernel has finished booting the Linux environment is available for use on the system.[2]

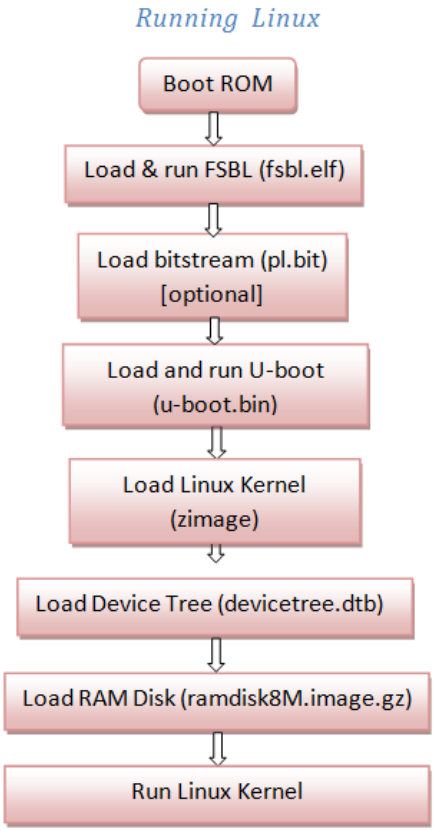


Figure 6: Linux Boot process on Zynq
Modified from source: wdfiles.com

4.1.4.2 Device tree

On ARM based systems the system wide configuration, what device is where, is defined in a device tree file. The device tree contains the memory locations, memory size, timers and device specific variables. The device tree also defines which driver the kernel should use for each device by using a compatible-id string. The device tree is build using a human readable file, which gets compiled in to a binary version, the device tree blob. This blob is usable by the boot loader and the kernel.

4.1.4.3 Device drivers

To be able to use a custom peripheral a device driver is required. As a proof of concept a program can be written to memory map the device registers into the memory space of the program. This program needs to be run as the root user, as no other user is allowed to preform a memory map. This program can be used to verify that the device is functioning correctly. However this approach is not feasible in the long term as it can only be used by the root user. This presents a problem as this approach might compromise security. It is also a problem because a normal user can not use this approach. To solve both of these problems a device driver can be written as a kernel module. This allows for the use of the device by normal users which have been granted the appropriate rights to access the device nodes. A Linux kernel module is written in C and is compiled to an object file. This resulting object file can be dynamically linked into the running kernel and will than make the device available for use.

4.1.4.4 File systems

Within Linux there exist only files and folders. Devices or special locations do not get a different type, they are always a file or a folder. The total file system is made up out of multiple parts, which may have special functions. Everything ends up under the “/” mount point, this is the root file system, or rootfs. In this rootfs there are multiple mount points and folders. The folders may contain user data, in “/home”, executable binaries in “/bin” or shared libraries in “/lib”. Mount points include “/dev” and “/sys”. These mount point are used to mount special file systems. The file system under “/dev” contains all the device nodes for the devices in the system. This includes the hard drives, denoted as “/dev/sdX”, serial ports designated with “/dev/tty...” and other devices. All of these entries are created dynamically by the udev daemon.[4]

The “/sys” mount point also houses a special file system, the sysfs, this file system houses all special files for the kernel objects and their relationships. In this file system a USB device would be located under the “/sys/bus/usb” folder. The udev daemon for the “/dev” file system uses this file system to create its device entries.[4][5]

4.2 Beckhoff EtherCAT

The following paragraphs detail the research into the Beckhoff EtherCAT control network. The first paragraph will talk about how the network functions, the second paragraph details the

available interfaces and the third paragraph will explain the workings of and the choice for the Beckhoff ET1100 slave ASIC.

4.2.1 Function

The EtherCAT protocol is based on the Ethernet protocol. However in contrast to Ethernet, EtherCAT allows for building a network without switches. The topology of this network can be build using a line, tree or star topology or any combination of the topologies, in contrast to Ethernet which only allows the use of tree topology using switches. Figure 7 shows a possible network layout for EtherCAT.

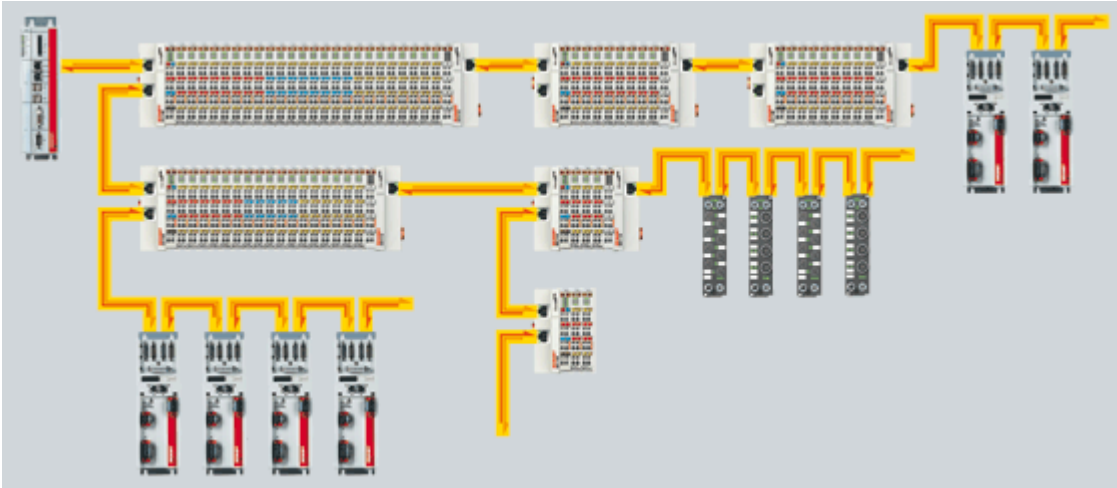


Figure 7: EtherCAT example network
Source: Beckhoff.com

EtherCAT uses a single master and multiple slaves. Masters are available as dedicated hardware masters, which can be used with an existing PLC, or as software masters. Beckhoff provides a software master called TwinCAT, which can be used on any PC running Windows if said PC can use virtualization and has been equipped with a compatible network interface card. When a compatible 100Mbit network adapter is installed, TwinCAT has drivers to turn them into an EtherCAT master or slave. There are also a few open source EtherCAT projects, namely Open EtherCAT Master/Slave[1] and IgH EtherCAT. However these open source packages are not supported by Beckhoff. Both the Simple Open EtherCAT tool and IgH EtherCAT[2] by EtherLab are available for Linux, where IgH's solution runs as a Linux kernel module and is capable of real time operation. Both a system with TwinCAT and a system using IgH EtherCAT are available in the lab. However, at the time of writing the TwinCAT system can only start into the configuration mode as the PC does not have it's virtualization enabled. This means that all slave development has to be done using the IgH master.

The EtherCAT protocol is capable of addressing a maximum of 65535 devices on a single bus.

1 https://github.com/OpenEtherCATsociety/SOEM
2 http://www.etherlab.org/en/ethercat/

Since the protocol is based upon the 100BASE-TX Ethernet protocol, EtherCAT has the same physical limitation of a maximum cable length of one hundred meters when using copper cable. If a greater distance between nodes needs to be bridged, fiber optics can be used. When using fiber optics the maximum length is increased to two kilometers.

The datagrams transmitted over the network, are not copied into RAM in the slaves. Instead each slave pulls the data it needs from the packet on the fly. In a datagram the minimum payload per device is sixty four kilobyte, with a maximum of four gigabytes per update for the entire network.

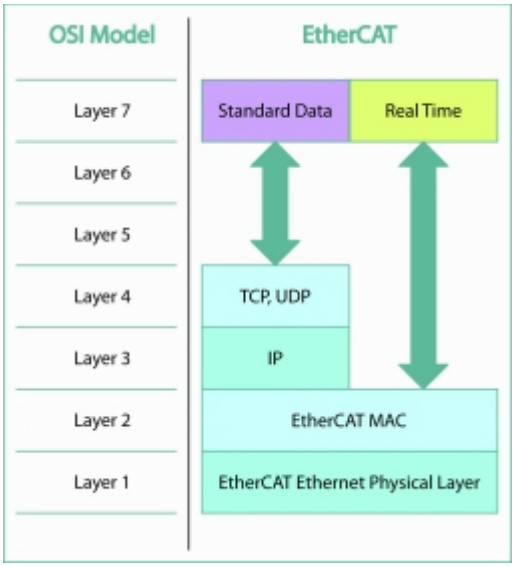


Figure 8: OSI model for EtherCAT
Source: processonline.com.au

Figure 8 shows the OSI model for EtherCAT. This shows that EtherCAT and Ethernet share the same layer 1 and layer 2, as in, they can both use the same medium for the physical layer and both use MAC addresses for the client addressing. The EtherCAT MAC uses an extra field as opposed to a standard Ethernet MAC. Layers 3 to 6 are not used with EtherCAT, as it supports only one protocol, and does not use individual slave addressing. The EtherCAT protocol uses the application layer, layer 7, for it's data transmissions.

The network is also capable of handling a so called “Hot connect/disconnect”[6]. More common terms for this functionality are “Hot pluging” or “Hot swapping”. This means that the network will not become unavailable when a new segment is connected to the network, or a segment is disconnected. The network will also detect a break point when a fault or disconnect occurs and route the datagrams in a suitable manner.

The network is also fully compatible with one hundred megabit Ethernet. This grants the ability to connect a normal PC to the network and be able to use for example HTTP for web browsing. The reverse is also possible, as in, it is possible to connect two segments of an EtherCAT

network over a normal Ethernet network. Both these options require the use of an EtherCAT gateway or gateways when connecting multiple segments over an Ethernet network. In this setup the EtherCAT datagrams are encapsulated into UDP packets[7].

4.2.2 Interfaces

For the EtherCAT network different interfaces are available. These interfaces can be separated into two main categories, namely I/O modules or fieldbus connectors. The I/O modules provide direct I/O operations or motion control capabilities. For example digital I/O modules with 16 inputs or outputs are available. The motion controllers provide servo drives or motor controllers directly connected to the EtherCAT network

The fieldbus connectors provide the ability to connect different fieldbusses, for example PROFIBUS or RS-485, to the EtherCAT network. The slaves connected to these bussen can than be transparently controlled by the EtherCAT master. This allows for the ability to update or upgrade a control system with EtherCAT without having to replace existing systems.

4.2.3 Beckhoff ET1100

To ease the development of new or custom slave, Beckhoff designed two EtherCAT slave ASICs. These ASICs are designated as the ET1100 and ET1200, which are depicted in Figure 9. Both of these chips are available on carrier boards for use with the EtherCAT development board.

Figure 9: Beckhoff ET1100 & ET1200 slave ASICs
Source: Beckhoff.com

Since the Xilinx Zynq contains SPI controllers, which are capable of functioning as SPI masters, connecting to a slave ASIC is simple. Both ASICs are capable of functioning as an SPI slave or can use a sixteen bit digital I/O bus. The ET1100 also contains a thirty two bit digital I/O bus and can use an eight or sixteen bit parallel data interface for connectivity with a microcontroller[8].

Figure 10: Beckhoff FB1111-0141 SPI slave board
Source: Beckhoff.com

In this project development for the EtherCAT network has been done using an Beckhoff ET1100 slave on the FB1111-0141 carrier board (Figure 10). This board gives access to the SPI interface and has been placed on the EL9800 EtherCAT development board (Figure 11). The EL9800 contains a power supply and breaks out the usable interfaces.

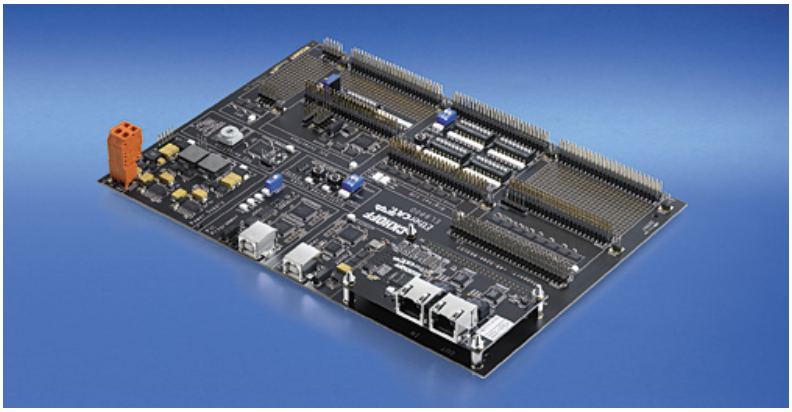


Figure 11: Beckhoff EL9800 development board with FB1111 board
Source: Beckhoff.com

4.2.4 Slave Stack Code

When using an ASIC with an external controller, for example a microcontroller, the controller needs to respond to the masters requests. The ASIC handles decoding the EtherCAT messages, but the Slave Stack runs on the controller. The Slave Stack Code has been provided by Beckhoff via the EtherCAT Technology Group.

4.3 Serial Peripheral Interface

As both the Zynq and the ET1100 have the ability to use the Serial Peripheral Interface bus (SPI), this interface has been chosen. SPI is a synchronous communication protocol using a master/slave topology. For each bus there can only be one master, however each master can address many slaves. The Zynq chip is designated as the master, where the ET1100 functions as a slave. The two chips are connected using 4 wires, namely MISO, MOSI, SCK and SS. MOSI is the signal wire that transfers data from the master (Master Out) to the slave (Slave In). MISO functions as the opposite, transferring data from the slave to the master (Master In Slave Out). Each bit is transferred on a clock edge. The clock signal is provided by the master on the SCK line. Finally the Slave Select (SS) line is asserted by the master to select which slave it is communicating with. When only a single slave is used, the SS line can be omitted by connecting the SS pin on the slave to ground, as SS is usually inverted. Figure 12 shows a typical SPI bus with each slave connected to SCLK or SCK, MOSI and MISO. Each slave is addressed by its own SS line.

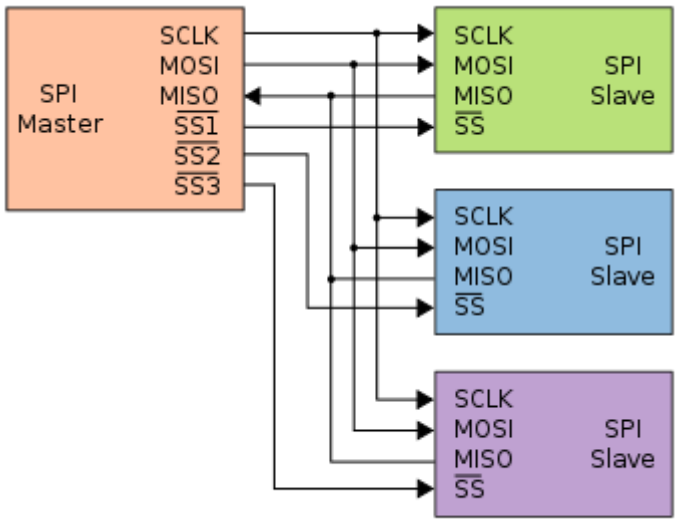


Figure 12: Typical SPI bus topology utilizing multiple slaves
Source: wikimedia.org

The SPI bus can operate in four modes. These modes determine whether a high signal or a low signal is the active state for the clock signal, designated by CPOL or Clock POLarity, and on which edge the data is sampled, designated by CPHA or Clock PHAse. When CPOL is 0, the active state for the clock line is high, and the clock rests on a low state. With CPOL equal to 1, the clock signal is inverted resulting in the active state being low and the rest state being a high

value. The CPHA is independent from the clock polarity, this results in two options per clock polarity. Namely sample data on the first edge and transmit data on the second edge with CPHA equal to 0, or with CPHA equal to 1 transmit on the first edge of the clock and sample on the second edge. Table 1 shows the four possibility for the SPI modes. The selected mode is configured on the master and has to match with the slaves.

Table 1: SPI modes with CPOL and CPHA

SPI Mode	Clock Polarity (CPOL))	Clock Phase (CPHA)
0	0	0
1	0	1
2	1	0
3	1	1

Source: Bas Janssen

Figure 13 shows an example SPI transmission from the slaves point of view. To initiate the transmission the master asserts the CS, or SS, line and starts generating clock pulses on the CLK line. The first eight bits are transferred to the slave on the MOSI line with the MISO line as undefined. After this first byte the bus becomes full duplex and the slave transfers its output at the same time as it receives data from the master. To receive the last byte of data from the slave the master sends a dummy byte and terminates the transmission by releasing the SS line.

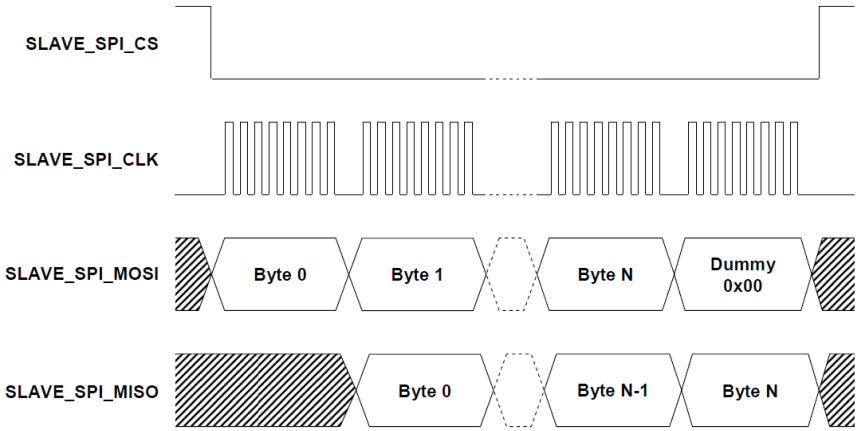


Figure 13: Example SPI data transmission
Source: ti.com

4.4 PID controller

Figure 14 shows a schematic representation of a PID controller in a feedback loop. This schematic shows a process with an input u(t) and an output y(t). This process does not conform to the required response, so the process has to be controlled. This can be done using a PID controller, which is depicted as the sum of the three factors, Proportional, Integral and Differential. The controller receives the error, the difference between the requested value, also knows as the setpoint, and the value the process returns, and calculates the new value to sent to the process. In the time domain, the P factor uses the current value of the error. If the error is large, then its output will be large as well. And if the error is small, the output from the P factor will be small. The I factor uses the previous values of the error and tries to correct if the controller output has a small offset over time. This factor will eventually get the system stable on its desired value. The D factor attempts to predict the future by calculating the current rate of change in the error value. If the error changes rapidly it will counteract the P factors output, but if it changes slowly it will boost the output of the controller. The individual outputs of the P, I and D functions are added to create the output value for the controller.

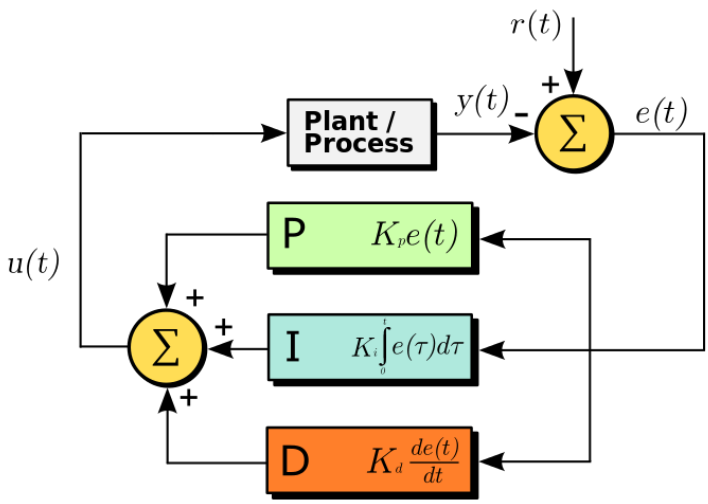


Figure 14: Block diagram of a PID controller in a feedback loop
Source:wikimedia.org

5 Design

The following paragraphs will elaborate on the setup of the tool chain, including the selection of cross compilers and software packages, and the software which has been written for the system, or existing software which has been cross compiled to the target platform.

5.1 Tool chain

To assist in developing software for the Zynq APU, Xilinx has created the Xilinx SDK. This SDK enables development for bare-metal applications as well as targeting the Linux environment on the Zynq. The SDK includes the cross compiler, Xilinx ARM GCC, and tools to create the First Stage Boot Loader, a tool to create the device tree file and a packager to build the RAM disk for the Zynq system.

5.1.1 Linux kernel

The available Linux kernel images for the Xilinx Zynq chips do not contain all drivers. This means that when an extra driver is needed, the kernel will have to be recompiled with the required driver enabled. To make sure the correct drivers are used, Xilinx offers a branch of the main line kernel through their git repository[3]. When the cross compiler included in the SDK has been installed, it can be used to compile the kernel. The resulting kernel image can be used to boot the kernel on the Zynq.

5.1.2 RAMdisk

The file system for the Linux environment is build as a RAMdisk image. The RAMdisk image is decompressed and loaded into RAM and becomes available as the root file system for the kernel.

To be able to create the RAMdisk image, a folder is created on the development machine. Within this folder, the entire structure of the file system is recreated. Then the different configuration files, binaries and links are placed into the required locations. To build the image of this folder, the folder is than packaged into a RAMdisk and a header is added to it so that U-boot will recognize it as a valid RAMdisk.

5.1.3 Boot image

The boot image for the Zynq contains the First Stage Boot Loader, the bitstream for the FPGA and points to the U-boot boot loader. The SDK can than build this binary image so that it can be used on the target device.

3 https://github.com/Xilinx/linux-xlnx

5.2 Software

The following sections details the different pieces of software that have been developed for the project or have been cross compiled to assist with the development of the software.

The first paragraphs will expand on the reason to cross compile ncurses to the Zynq system. The second section details the test program developed to test the AXI bus interface and verify that the custom hardware is accessible. The next three sections explain the functionality of the driver written for the encoder readout, the PWM generator and the PID driver. The last section of this chapter details the software developed for the EtherCAT slave.

5.2.1 ncurses

To provide a graphical user interface (GUI) to the user when the system is used in the standalone mode, when no data is provided by the control network, ncurses can be used to build a GUI in a terminal screen. This allows for a GUI that can be used over SSH, on a monitor running only a terminal interface or on top of a graphical environment in a terminal emulator. This interface can be build using the ncurses package. To be able to use this package on the Zynq system, the package has been cross compiled to the ARM architecture and placed in the appropriate locations of the file system.

Figure 15 shows the interface designed as a test for the package over SSH. This interface allows the user to enter values into the blue fields and outputs the results from the AXI slave in the red fields. This interface can be used to replicate the test results from Table 2.

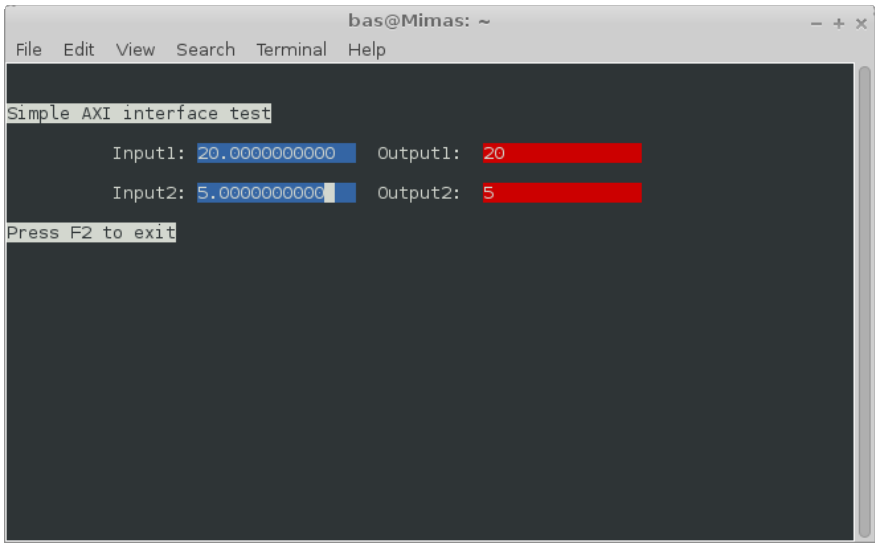


Figure 15: ncurses example interface
Source: Bas Janssen

5.2.2 AXI

As a proof of concept for the device driver, a program has been written that, when running under the root user, memory maps the registers of the AXI slave into it's personal RAM. The source code for this proof of concept has been added in Appendix 2: AXI test C code.This program is capable of manipulating the registers to control the peripheral. As a test, a simple slave containing four registers has been generated using Vivado and has been configured to forward data received on the first to registers to the second two registers. The VHDL code for this slave is available in Appendix 3: AXI test VHDL. The test program then places data in the first two registers and reads the second two registers. Both the input data and the output data are then displayed on the screen. This process of writing and reading is then repeated ten times with integers for the first data set and floats for the second data set. The results of this test have been added in Table 2.

Table 2: AXI bus test results

AXI bus communication test.			
Device base address: 43c00000			
Requesting base vaddress			
Base address: b6f85000			
Address input_0 b6f85000			
Address input_1 b6f85004			
Address output_0 b6f85008			
Address output_1 b6f8500c			
Testing bus using 10 integers and 10 floats.			
I0:1	O0:1	I1:10.010000	O1:10.010000
I0:5	O0:5	I1:-5.070000	O1:-5.070000
I0:200	O0:200	I1:0.000000	O1:0.000000
I0:8080	O0:8080	I1:1000.101990	O1:1000.101990
I0:6	O0:6	I1:77.769997	O1:77.769997
I0:1000	O0:1000	I1:-203801.781250	O1:-203801.781250
I0:480323	O0:480323	I1:10000.000000	O1:10000.000000
I0:1000000	O0:1000000	I1:54358.000000	O1:54358.000000
I0:-500	O0:-500	I1:-22.000000	O1:-22.000000
I0:1993	O0:1993	I1:4853.000000	O1:4853.000000

Source: Bas Janssen

5.2.3 Encoder module driver

This section describes the register map for the encoder block, the functionality of the device node and the test to verify that the driver works. For the generic functions in the driver that enable this functionality, see Driver init() and exit() functions, Driver probe() and remove() functions, Device node open() and close() functions and Device node read() and write() functions under the PID controller driver section (5.2.5). The source code for this driver has been appended in Appendix 8: Encoder driver C code.

5.2.3.1 Register map

The registers in Table 3 are both 32bit registers, where the first register uses the full size of the register to report the current position relative to the starting position of the system. The direction register uses only a single bit to relay the direction information. This map has been devised in conjunction with the hardware designer.

Table 3: Encoder readout register map

Register map for Encoder readout						
Register	Location	Function	Description	Type	Value range	Value type
Position	0x00	R	Read the current position	u_int32	0-4.294.967.295	Counts
Direction	0x04	R	Read the direction	u_int32	0-4.294.967.295	Unsigned integer

Source: Bas Janssen

5.2.3.2 Device node

For the encoder block, the device node is created by the driver when the driver is loaded and a compatible entry is found. This device node is read only and returns the current position in encoder counts.

5.2.3.3 Sysfs node

Since the encoder block not only returns the current position in counts, but also returns the direction the system is traveling, a read only sysfs node is created to enable reading the direction from the encoder block. This direction is represented with the least significant bit in the direction register.

5.2.3.4 Test

Testing the encoder driver was done by configuring six encoder modules in the programmable logic. Loading the driver created the six corresponding devices under Linux. By hooking up an encoder to the inputs, each module could be tested by turning the encoder and reading the device node for the current position, and the sysfs node to get the direction of travel. This tested showed the driver functions as expected.

5.2.4 PWM module driver

This section describes the register map for the PWM block, the functionality of the device node and the test to verify that the driver works. For the generic functions in the driver that enable this functionality, see Driver init() and exit() functions, Driver probe() and remove() functions, Device node open() and close() functions and Device node read() and write() functions under the PID controller driver section (5.2.5). The source code for this driver has been appended in Appendix 6: PWM driver C code.

5.2.4.1 Register map

As for the encoder readout, a register map has been devised for the PWM module. This module only uses a single register to set the pulse width. The pulse width is a thirteen bit value. The location and specifications for this register are depicted in Table 4. Currently the PWM frequency is fixed at 2.5 kHz.

Table 4: PWM driver register map

Register map for PWM driver						
Register	Location	Function	Description	Type	Value range	Value type
Pulse Width	0x00	W	Set new pulse width	u_int32	0-8192	pulse width

Source: Bas Janssen

5.2.4.2 Device node

For the PWM block, the driver creates a new entry in the /dev filesystem for each device specified in the devicetree. This device node can be read to get the current pulse width, or can be written to to set the pulse width. Since this device only needs a single register to function, no sysfs nodes a created. The device node write() function differs from the function described in 5.2.5.5 by a check for the value written to the device register. If this value is greater than 8192 the driver will limit the value to 8192.

5.2.4.3 Test

To test the driver an FPGA design has been created with four PWM blocks. Based on this design a new devicetree has been created. Using these parts, the driver can be tested. With the new devicetree in place, when the driver is loaded new devices should be created. For each successful device creation, the driver outputs a “New device created” message to the kernel log. After this has been confirmed, the new bitstream with the PWM blocks is loaded into the FPGA. Writing a value to the first device node should now result in a PWM output from the first block. Figure 16 shows the oscilloscope screen for this test. This screen shot displays that the driver works, and the PWM signals are generated. For this test, channel one has been set to “1024”, channel two to “2048”, channel three to “4096” and channel five to “6144”.

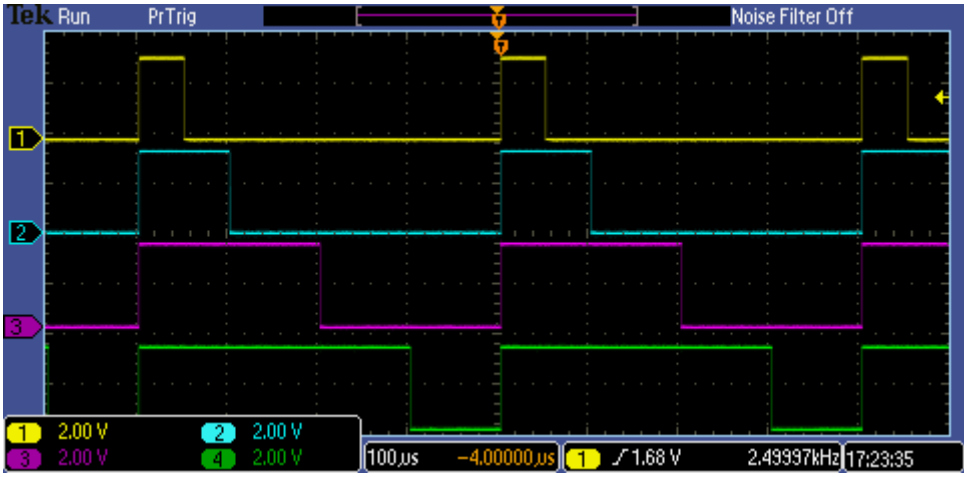


Figure 16: Oscilloscope display showing multiple PWM drivers output
Source: Bas Janssen

5.2.5 PID controller driver

After the proof of concept driver has been proven, a proper driver has been written for the PID controller using the Linux Device Drivers book[9] and the source code of the Spidev driver[10]. The source code for the driver has been appended in Appendix 4: PID driver C code. This driver takes ownership of the memory area where the registers for the controller are located. This memory area is then mapped into the kernel space memory the driver has available, and becomes readable and writable. After testing if the driver can claim this memory area, which the kernel showed to work, the driver has been expanded with the ability to read from the registers and write to these registers through read and write operations on the device node and the sysfs node. After testing these features using the four register slave, the driver has been modified to dynamically create devices and device nodes using the device tree definitions. To be able to do this the driver registers itself as a platform device driver and tells the kernel which device id's are compatible with this driver. The definition in the device tree contains the base address of the controller and the size of the memory chunk it uses. This allows the driver to calculate the addresses for the different settings for the controller and makes it possible to create a device node and the sysfs nodes for each controller.

5.2.5.1 Register map

To ease development a register map for the PID controller has been devised. This map is specified in Table 5 and has been devised in conjunction with the hardware developer.

Table 5: PID controller register map

Register map for PID controller						
Register	Location	Function	Description	Type	Value range	Value type
Setpoint	0x00	W	Set new setpoint	u_int32	0-4.294.967.295	Counts
P	0x04	R/W	Read/Set P factor	u_int32	0-4.294.967.295	Unsigned integer
I	0x08	R/W	Read/Set I factor	u_int32	0-4.294.967.295	Unsigned integer
D	0x0C	R/W	Read/Set D factor	u_int32	0-4.294.967.295	Unsigned integer
Update	0x10	W	Write update flag for PID factors	u_int32	0-4.294.967.295	bitwise flag
State	0x14	R	Read system state	u_int32	0-4.294.967.295	bitwise flag
Emerg	0x18	R	Read emergency state	u_int32	0-4.294.967.295	bitwise flag
Position	0x1C	R	Read current possition	u_int32	0-4.294.967.295	Counts

Source: Bas Janssen

Each register is 32 bits wide, this allows for the use of 32 bit variables on the application processor. The register locations are stored in a struct. The Setpoint and Position registers are mapped to the device node, as respectively the write function and the read function. All the other registers are mapped to the sysfs nodes for the controller.

5.2.5.2 Driver init() and exit() functions

The init() function handles the basic functionality of the driver. First the function requests a major device number from the kernel. If the kernel assigns a number to the driver, the driver creates a class for the devices. When the driver has been able to create the class, it will register itself as a platform device driver. This makes it possible to automatically create devices.

In the exit() function everything the driver did in the init() function will be undone. First it will unregister itself as a platform driver, then destroy the device class and finally release the major device number to the kernel.

5.2.5.3 Driver probe() and remove() functions

After the driver has been registered as a device driver the kernel can call the probe() function when the kernel reads the device tree and finds a matching device. The probe() function then creates a device using the first available minor device number. The driver than retrieves the base address and the size of the memory area the device uses from the device tree. The driver will than request this memory area to be assigned to the driver. Then the driver will memory map each set of registers for the controllers into it's own memory so that they will be readable and writable. Finally it will calculate the mapped addresses for each register, e.g. the address for the set point register, or the I factor. The addresses are stored in a struct which is passed to the different functions of the driver. If however the request for a specific memory region fails, the device will not be created.

When the OS shuts down, or the driver is manually unloaded from the kernel, the kernel will call the remove() function. This function removes the devices and their device nodes and releases the minor numbers. When the last device is removed, it will also release the memory area and

destroy the struct.

5.2.5.4 Device node open() and close() functions

The open() function for the device node activates the mutex lock and retrieves the struct with the address information from the driver data. This struct is then stored in the file struct so the read() and write() function for the device node can use the data in the struct. The mutex is used to control access to the device node, as in it is mutually exclusive thus prevents concurrent opening of the device node from different processes.

The close() function disables, or unlocks, the mutex so a other process can access the device node.

5.2.5.5 Device node read() and write() functions

Within the read() and write() functions for the device node the respective addresses are used to write to the set point for the controller or read the momentary angular position from the controller. The read() function also uses a one-shot function to allow for the read operation to be executed by the “cat” tool where after the first read it returns with a “return 0” state to indicate the last of the available data has been passed. This allows for the request of a single momentary position by issuing “cat /dev/PIDx”.

5.2.5.6 Sysfs nodes

For the configuration of the controllers the driver creates multiple nodes in the sysfs file system. The nodes are, depending on the function, read-only, write-only, or both readable and writable. These nodes do not have open() or close() functions, they only use read() and write() functions. Both of these function use the struct created in the probe() function to determine the addresses they should use.

The nodes created in the sysfs are the following, but are not limited to, the P, I and D settings, the run/stop setting, and an error status register.

5.2.5.7 Test

A test checklist for the PID driver has been added in Appendix 5: PID driver test plan. This list has not been tested with the actual hardware PID controller. However, this checklist still proves the driver works. Testing with the hardware has to be done when the hardware is finished.

5.2.6 EtherCAT Slave

The following sections will detail the different parts used to build the driver for the EtherCAT Slave. The first section details the testing of the Zynq SPI interface, the second section describes the implementation of the Slave Stack Code and the third section explains the reason no driver has been build.

5.2.6.1 Spidev

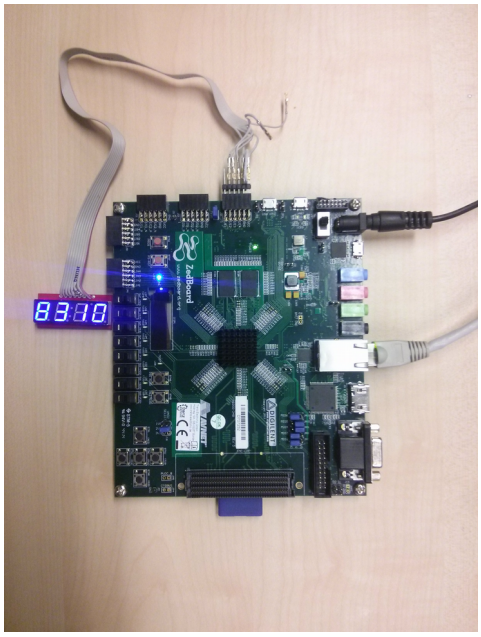


Figure 17: Test setup for the Proof of Concept
Source:Bas Janssen

To demonstrate that the SPI bus can be used from within Linux on the AP, the SPIdev driver has been used to drive a Sparkfun Serial 7-segment Display. This display is build using four 7-segment displays and can be controlled using SPI. Figure 17 show the setup used during this demonstration.

5.2.6.2 Slave Stack Code

The Slave Stack Code (SSC), as provided by Beckhoff, forms the basis for the EtherCAT driver. The SSC provides the required functions to build the software layer for the slave. The device specific method for communication with the slave ASIC must be provided by the developer.

5.2.6.3 Driver

For this project a driver for the EtherCAT slave should have been designed. However, due to time constraints this driver has not been developed. Delivery of the slave ASIC and the Slave Stack Code caused a delay to great to compensate. The development of this driver will be done outside of the internship detailed in this report.

6 Results

This project has resulted in a functional driver for the PID controller, together with drivers for the PWM and encoder hardware. These drivers are stable and usable, and are able to dynamically handle multiple devices in the programmable logic.

With these blocks a working PID driver can be build using a software PID controller running on the Linux environment, with the hardware abstracted by the drivers. This driver has been build, and a video showing the driver in operation has been added in Appendix 11: Digital files.

The other part of this project, interfacing with the EtherCAT control network has not been completed due to time constraints. All the required parts have been delivered, and are ready to be used to develop the driver.

7 Conclusion and recommendation

The project has been completed with the EtherCAT slave not implemented. The other three drivers, the PWM, Encoder and PID drivers, have been build and tested. A demonstrator has been build using the PWM and Encoder parts and a software controller.

For a possible follow up project, the following recommendations can be made:

- Look into getting Ubuntu with ROS running on the Zynq chips. A possible environment can be acquired from the Snickerdoodle project.
- Collaborate with the Fontys ICT studies on the device drivers for custom hardware.
- Start a repository with pre build hardware and corresponding drivers and software to encourage reuse.
- Test the PID controller driver with the actual hardware when the hardware is finished.

List of abbreviations

AMBA	Advanced Microcontroller Bus Architecture
AP	Application Processor
ARM	Advanced RISC Machine
AXI	Advanced eXtensible Interface
EMIO	Extended Muxed Input/Output
FPGA	Field Programmable Gate Array
FSBL	First Stage Boot Loader
GUI	Graphical User Interface
MIO	Muxed Input/Output
SoC	System on Chip
OS	Operating System
PL	Programmable Logic
SPI	Serial Peripheral Interface
SSH	Secure SHell
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

Bibliography

1: Xilinx, AMBA AXI4 Interface Protocol, 2016, http://www.xilinx.com/ipcenter/axi4.htm
2: Xilinx, Zynq-7000All Programmable SoC Technical Reference Manual, 2015
3: Xilinx, Zynq-7000 AP SoC, SPI - In Master Mode on MIO, the SPI Controller Resets Itself when the SS0 Signal Asserts, 2012
4: Greg Kroah-Hartman, udev– A Userspace Implementation of devfs, 2003
5: Patrick Mochel, The sysfs Filesystem, 2005
6: Beckhoff, Hot Connect, Diagnostic, , https://www.beckhoff.com/english/ethercat/aufbau3.htm?id=20433492043364
7: Beckhoff, EtherCAT – Ultra high-speed for automation, 2016, https://www.beckhoff.com/english/ethercat/highlights.htm?id=20433492043386
8: Beckhoff, ET1100, ET1200 | EtherCAT ASICs, 2016
9: Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman , Linux Device Drivers, 3rd Edition, 2005
10: Andrea Pateriani, David Brownell, Simple synchronous userspace interface to SPI devices, 2006

Appendix 1: Internship description document

Graduation Project Assignment

Real-Time Distributed Control System at Mechatronics Lab, Fontys

Abstract

The main goal of this project is to build a demonstrator for demonstrating the capabilities an Ethercat motion controller for the fourth order model for educational purposes.

Introduction

Closed loop control systems are used in many application areas such as industrial automation, avionics, and automotive. These closed loop control systems are often implemented in either as a standalone system or as distributed embedded control units. In this project we will investigate the basics of distributed Control System (DCS). The main goal of this project is to build a demonstrator for demonstrating the capabilities an Ethercat motion controller for the fourth order model for educational purposes.

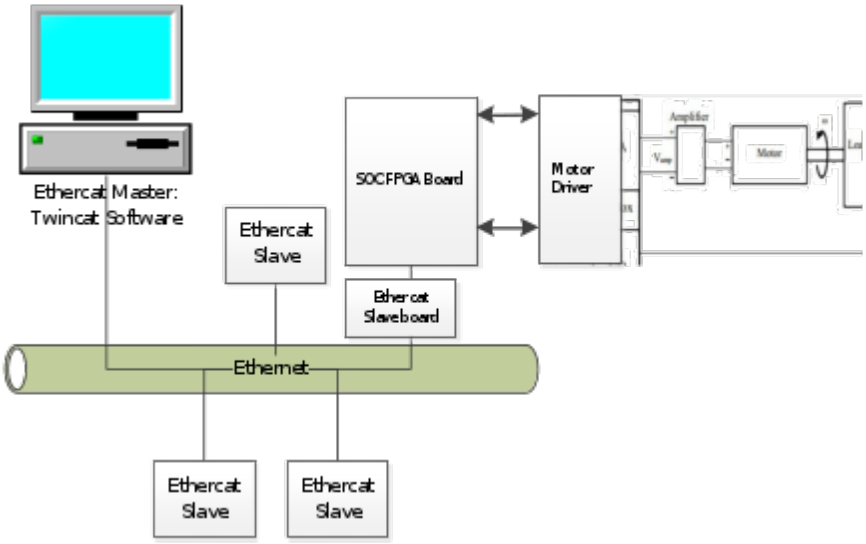


Figure 1 DCS global architecture -Phase-1

The implementation will have to be built such that it can be used as demonstrator for students.

Proposed hardware

It is estimated that this demonstrator will require new hardware support. Some of newly available SOC board can be used. For example Zybo or Snikerdoodle (see reference list) can be used since

they have a lot of processing power and can easily interface with other devices.

Proposed tasks

The task can be split between two students, while both are responsible for the total system. One will concentrate on the FPGA part while the other will focus on the software control.

1. First step in this assignment would be to investigate the working of system. In this step all the hardware and software that enables the use of Ethercat will be studied. Ethercat control loop system will also be studied. Based on these acquired knowledge a new model will be created, programmed and compared with existing model in order to demonstrate sufficient skills
2. Next step would be to explore different possibilities in order to replace some of the Ethercat modules with our own Modules, for example a micro-controller (ARM/Arduino). The main goal is exploring how a control system can be implemented locally on a microcontroller using an external controller and also how a less intelligent device could be used locally with main control being implemented by a more intelligent central device.
3. To implement the above system with the use of a system on chip (SOC) board such as Zybo or Snickerdoodle board. The use of a SOC board with Ethercat networking device will enable us to implement a fast PID controller that will be partly implemented in FPGA and partly in the Linux OS running on the arm processor.
4. In case time allows the use of Ethercat slave IP in the FPGA will be investigated.

End products

A working demonstrator of a fourth order model with the use of Ethercat.

With the use of a slave/master system, a control loop implemented on an ARM processor or an Arduino. Preferred hardware will be Zybo or Snickerdoodle.

References:

1. https://www.crowdsupply.com/krtkl/snickerdoodle
2. http://store.digilentinc.com/zybo-zynq-7000-arm-fpga-soc-trainer-board/
3. http://www.beckhoff.com/english.asp?highlights/ethercat/default.htm?pk_campaign=AdWords-AdWordsSearch-EtherCatEN&pk_kwd=ethercat&gclid=CIy7waiy28oCFSfkwgodkeIENA

Appendix 2: AXI test C code

```
/*
 * Copyright (c) 2016 Bas Janssen
 *
 */

#include <stdio.h>
#include <dirent.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdlib.h>
#include <string.h>
#include <stddef.h>
#include <stdbool.h>

#define MAP_SIZE 4096L
#define MAP_MASK (MAP_SIZE - 1)

#define ADDR_BASE 0x43C00000

static int inputs0[] = {1, 5, 200, 8080, 6, 1000, 480323, 1000000, -500, 1993};
static float inputs1[] = {10.01, -5.07, 0.0000001, 1000.1020, 77.77, -203801.7804, 9999.9999, 54358.0, -22,4853,
-300.0003};


void *getvaddr(int phys_addr)
{
	void *mapped_base;
	int memfd;

	void *mapped_dev_base;
	off_t dev_base = phys_addr;

	memfd = open("/dev/mem", O_RDWR | O_SYNC); //To open this the program needs to run as root
		if(memfd == -1)
		{
			printf("Can't open /dev/mem. \n");
			exit(0);
		}

		mapped_base = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, memfd,
dev_base & ~MAP_MASK);
			if(mapped_base == (void *) -1)
			{
				printf("Can't map the memory to user space.\n");
				exit(0);
			}

			mapped_dev_base = mapped_base + (dev_base & MAP_MASK);
			return mapped_dev_base;
}
int main(void)
{
	int i = 0;
	printf("AXI bus communication test.\n");
```

```
	printf("Device base address: %x\n", ADDR_BASE);
	printf("Requesting base vaddress\n");

	int * dev_base_vaddr = getvaddr(ADDR_BASE);
	printf("Base address: %x\n", dev_base_vaddr);
	int * input_0 = dev_base_vaddr;
	printf("Address input_0 %x\n", input_0);
	float * input_1 = dev_base_vaddr + 1;
	printf("Address input_1 %x\n", input_1);
	int * output_0 = dev_base_vaddr + 2;
	printf("Address output_0 %x\n", output_0);
	float * output_1 = dev_base_vaddr + 3;
	printf("Address output_1 %x\n", output_1);

	printf("Testing bus using 10 integers and 10 floats.\n");
	for(i=0; i<10; i++)
	{
		*input_0 = inputs0[i];
		*input_1 = inputs1[i];
		printf("I0:%i\tO0:%i\tI1:%f\tO1:%f\n", inputs0[i], *output_0, inputs1[i], *output_1);
	}

	//Unmap the virtual address space
	munmap(dev_base_vaddr, MAP_SIZE);

	return(0);
}
```

Appendix 3: AXI test VHDL

```
--Copyright 1986-2015 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------
--Tool Version: Vivado v.2015.4 (lin64) Build 1412921 Wed Nov 18 09:44:32 MST 2015
--Date        : Wed Feb 24 14:29:24 2016
--Host        : Mimas running 64-bit Debian GNU/Linux 8.3 (jessie)
--Command     : generate_target simple_axi_interface.bd
--Design      : simple_axi_interface
--Purpose     : IP block netlist
------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity s00_couplers_imp_O2VN7B is
  port (
    M_ACLK : in STD_LOGIC;
    M_ARESETN : in STD_LOGIC_VECTOR ( 0 to 0 );
    M_AXI_araddr : out STD_LOGIC_VECTOR ( 31 downto 0 );
    M_AXI_arprot : out STD_LOGIC_VECTOR ( 2 downto 0 );
    M_AXI_arready : in STD_LOGIC;
    M_AXI_arvalid : out STD_LOGIC;
    M_AXI_awaddr : out STD_LOGIC_VECTOR ( 31 downto 0 );
    M_AXI_awprot : out STD_LOGIC_VECTOR ( 2 downto 0 );
    M_AXI_awready : in STD_LOGIC;
    M_AXI_awvalid : out STD_LOGIC;
    M_AXI_bready : out STD_LOGIC;
    M_AXI_bresp : in STD_LOGIC_VECTOR ( 1 downto 0 );
    M_AXI_bvalid : in STD_LOGIC;
    M_AXI_rdata : in STD_LOGIC_VECTOR ( 31 downto 0 );
    M_AXI_rready : out STD_LOGIC;
    M_AXI_rresp : in STD_LOGIC_VECTOR ( 1 downto 0 );
    M_AXI_rvalid : in STD_LOGIC;
    M_AXI_wdata : out STD_LOGIC_VECTOR ( 31 downto 0 );
    M_AXI_wready : in STD_LOGIC;
    M_AXI_wstrb : out STD_LOGIC_VECTOR ( 3 downto 0 );
    M_AXI_wvalid : out STD_LOGIC;
    S_ACLK : in STD_LOGIC;
    S_ARESETN : in STD_LOGIC_VECTOR ( 0 to 0 );
    S_AXI_araddr : in STD_LOGIC_VECTOR ( 31 downto 0 );
    S_AXI_arburst : in STD_LOGIC_VECTOR ( 1 downto 0 );
    S_AXI_arcache : in STD_LOGIC_VECTOR ( 3 downto 0 );
    S_AXI_arid : in STD_LOGIC_VECTOR ( 11 downto 0 );
    S_AXI_arlen : in STD_LOGIC_VECTOR ( 3 downto 0 );
    S_AXI_arlock : in STD_LOGIC_VECTOR ( 1 downto 0 );
    S_AXI_arprot : in STD_LOGIC_VECTOR ( 2 downto 0 );
    S_AXI_arqos : in STD_LOGIC_VECTOR ( 3 downto 0 );
    S_AXI_arready : out STD_LOGIC;
    S_AXI_arsize : in STD_LOGIC_VECTOR ( 2 downto 0 );
    S_AXI_arvalid : in STD_LOGIC;
    S_AXI_awaddr : in STD_LOGIC_VECTOR ( 31 downto 0 );
    S_AXI_awburst : in STD_LOGIC_VECTOR ( 1 downto 0 );
    S_AXI_awcache : in STD_LOGIC_VECTOR ( 3 downto 0 );
    S_AXI_awid : in STD_LOGIC_VECTOR ( 11 downto 0 );
    S_AXI_awlen : in STD_LOGIC_VECTOR ( 3 downto 0 );
```

```
    S_AXI_awlock : in STD_LOGIC_VECTOR ( 1 downto 0 );
    S_AXI_awprot : in STD_LOGIC_VECTOR ( 2 downto 0 );
    S_AXI_awqos : in STD_LOGIC_VECTOR ( 3 downto 0 );
    S_AXI_awready : out STD_LOGIC;
    S_AXI_awsize : in STD_LOGIC_VECTOR ( 2 downto 0 );
    S_AXI_awvalid : in STD_LOGIC;
    S_AXI_bid : out STD_LOGIC_VECTOR ( 11 downto 0 );
    S_AXI_bready : in STD_LOGIC;
    S_AXI_bresp : out STD_LOGIC_VECTOR ( 1 downto 0 );
    S_AXI_bvalid : out STD_LOGIC;
    S_AXI_rdata : out STD_LOGIC_VECTOR ( 31 downto 0 );
    S_AXI_rid : out STD_LOGIC_VECTOR ( 11 downto 0 );
    S_AXI_rlast : out STD_LOGIC;
    S_AXI_rready : in STD_LOGIC;
    S_AXI_rresp : out STD_LOGIC_VECTOR ( 1 downto 0 );
    S_AXI_rvalid : out STD_LOGIC;
    S_AXI_wdata : in STD_LOGIC_VECTOR ( 31 downto 0 );
    S_AXI_wid : in STD_LOGIC_VECTOR ( 11 downto 0 );
    S_AXI_wlast : in STD_LOGIC;
    S_AXI_wready : out STD_LOGIC;
    S_AXI_wstrb : in STD_LOGIC_VECTOR ( 3 downto 0 );
    S_AXI_wvalid : in STD_LOGIC
  );
end s00_couplers_imp_O2VN7B;

architecture STRUCTURE of s00_couplers_imp_O2VN7B is
  component simple_axi_interface_auto_pc_0 is
  port (
    aclk : in STD_LOGIC;
    aresetn : in STD_LOGIC;
    s_axi_awid : in STD_LOGIC_VECTOR ( 11 downto 0 );
    s_axi_awaddr : in STD_LOGIC_VECTOR ( 31 downto 0 );
    s_axi_awlen : in STD_LOGIC_VECTOR ( 3 downto 0 );
    s_axi_awsize : in STD_LOGIC_VECTOR ( 2 downto 0 );
    s_axi_awburst : in STD_LOGIC_VECTOR ( 1 downto 0 );
    s_axi_awlock : in STD_LOGIC_VECTOR ( 1 downto 0 );
    s_axi_awcache : in STD_LOGIC_VECTOR ( 3 downto 0 );
    s_axi_awprot : in STD_LOGIC_VECTOR ( 2 downto 0 );
    s_axi_awqos : in STD_LOGIC_VECTOR ( 3 downto 0 );
    s_axi_awvalid : in STD_LOGIC;
    s_axi_awready : out STD_LOGIC;
    s_axi_wid : in STD_LOGIC_VECTOR ( 11 downto 0 );
    s_axi_wdata : in STD_LOGIC_VECTOR ( 31 downto 0 );
    s_axi_wstrb : in STD_LOGIC_VECTOR ( 3 downto 0 );
    s_axi_wlast : in STD_LOGIC;
    s_axi_wvalid : in STD_LOGIC;
    s_axi_wready : out STD_LOGIC;
    s_axi_bid : out STD_LOGIC_VECTOR ( 11 downto 0 );
    s_axi_bresp : out STD_LOGIC_VECTOR ( 1 downto 0 );
    s_axi_bvalid : out STD_LOGIC;
    s_axi_bready : in STD_LOGIC;
    s_axi_arid : in STD_LOGIC_VECTOR ( 11 downto 0 );
    s_axi_araddr : in STD_LOGIC_VECTOR ( 31 downto 0 );
    s_axi_arlen : in STD_LOGIC_VECTOR ( 3 downto 0 );
    s_axi_arsize : in STD_LOGIC_VECTOR ( 2 downto 0 );
    s_axi_arburst : in STD_LOGIC_VECTOR ( 1 downto 0 );
    s_axi_arlock : in STD_LOGIC_VECTOR ( 1 downto 0 );
    s_axi_arcache : in STD_LOGIC_VECTOR ( 3 downto 0 );
```

```
    s_axi_arprot : in STD_LOGIC_VECTOR ( 2 downto 0 );
    s_axi_arqos : in STD_LOGIC_VECTOR ( 3 downto 0 );
    s_axi_arvalid : in STD_LOGIC;
    s_axi_arready : out STD_LOGIC;
    s_axi_rid : out STD_LOGIC_VECTOR ( 11 downto 0 );
    s_axi_rdata : out STD_LOGIC_VECTOR ( 31 downto 0 );
    s_axi_rresp : out STD_LOGIC_VECTOR ( 1 downto 0 );
    s_axi_rlast : out STD_LOGIC;
    s_axi_rvalid : out STD_LOGIC;
    s_axi_rready : in STD_LOGIC;
    m_axi_awaddr : out STD_LOGIC_VECTOR ( 31 downto 0 );
    m_axi_awprot : out STD_LOGIC_VECTOR ( 2 downto 0 );
    m_axi_awvalid : out STD_LOGIC;
    m_axi_awready : in STD_LOGIC;
    m_axi_wdata : out STD_LOGIC_VECTOR ( 31 downto 0 );
    m_axi_wstrb : out STD_LOGIC_VECTOR ( 3 downto 0 );
    m_axi_wvalid : out STD_LOGIC;
    m_axi_wready : in STD_LOGIC;
    m_axi_bresp : in STD_LOGIC_VECTOR ( 1 downto 0 );
    m_axi_bvalid : in STD_LOGIC;
    m_axi_bready : out STD_LOGIC;
    m_axi_araddr : out STD_LOGIC_VECTOR ( 31 downto 0 );
    m_axi_arprot : out STD_LOGIC_VECTOR ( 2 downto 0 );
    m_axi_arvalid : out STD_LOGIC;
    m_axi_arready : in STD_LOGIC;
    m_axi_rdata : in STD_LOGIC_VECTOR ( 31 downto 0 );
    m_axi_rresp : in STD_LOGIC_VECTOR ( 1 downto 0 );
    m_axi_rvalid : in STD_LOGIC;
    m_axi_rready : out STD_LOGIC
  );
  end component simple_axi_interface_auto_pc_0;
  signal S_ACLK_1 : STD_LOGIC;
  signal S_ARESETN_1 : STD_LOGIC_VECTOR ( 0 to 0 );
  signal auto_pc_to_s00_couplers_ARADDR : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal auto_pc_to_s00_couplers_ARPROT : STD_LOGIC_VECTOR ( 2 downto 0 );
  signal auto_pc_to_s00_couplers_ARREADY : STD_LOGIC;
  signal auto_pc_to_s00_couplers_ARVALID : STD_LOGIC;
  signal auto_pc_to_s00_couplers_AWADDR : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal auto_pc_to_s00_couplers_AWPROT : STD_LOGIC_VECTOR ( 2 downto 0 );
  signal auto_pc_to_s00_couplers_AWREADY : STD_LOGIC;
  signal auto_pc_to_s00_couplers_AWVALID : STD_LOGIC;
  signal auto_pc_to_s00_couplers_BREADY : STD_LOGIC;
  signal auto_pc_to_s00_couplers_BRESP : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal auto_pc_to_s00_couplers_BVALID : STD_LOGIC;
  signal auto_pc_to_s00_couplers_RDATA : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal auto_pc_to_s00_couplers_RREADY : STD_LOGIC;
  signal auto_pc_to_s00_couplers_RRESP : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal auto_pc_to_s00_couplers_RVALID : STD_LOGIC;
  signal auto_pc_to_s00_couplers_WDATA : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal auto_pc_to_s00_couplers_WREADY : STD_LOGIC;
  signal auto_pc_to_s00_couplers_WSTRB : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal auto_pc_to_s00_couplers_WVALID : STD_LOGIC;
  signal s00_couplers_to_auto_pc_ARADDR : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal s00_couplers_to_auto_pc_ARBURST : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal s00_couplers_to_auto_pc_ARCACHE : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal s00_couplers_to_auto_pc_ARID : STD_LOGIC_VECTOR ( 11 downto 0 );
  signal s00_couplers_to_auto_pc_ARLEN : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal s00_couplers_to_auto_pc_ARLOCK : STD_LOGIC_VECTOR ( 1 downto 0 );
```

```
  signal s00_couplers_to_auto_pc_ARPROT : STD_LOGIC_VECTOR ( 2 downto 0 );
  signal s00_couplers_to_auto_pc_ARQOS : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal s00_couplers_to_auto_pc_ARREADY : STD_LOGIC;
  signal s00_couplers_to_auto_pc_ARSIZE : STD_LOGIC_VECTOR ( 2 downto 0 );
  signal s00_couplers_to_auto_pc_ARVALID : STD_LOGIC;
  signal s00_couplers_to_auto_pc_AWADDR : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal s00_couplers_to_auto_pc_AWBURST : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal s00_couplers_to_auto_pc_AWCACHE : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal s00_couplers_to_auto_pc_AWID : STD_LOGIC_VECTOR ( 11 downto 0 );
  signal s00_couplers_to_auto_pc_AWLEN : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal s00_couplers_to_auto_pc_AWLOCK : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal s00_couplers_to_auto_pc_AWPROT : STD_LOGIC_VECTOR ( 2 downto 0 );
  signal s00_couplers_to_auto_pc_AWQOS : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal s00_couplers_to_auto_pc_AWREADY : STD_LOGIC;
  signal s00_couplers_to_auto_pc_AWSIZE : STD_LOGIC_VECTOR ( 2 downto 0 );
  signal s00_couplers_to_auto_pc_AWVALID : STD_LOGIC;
  signal s00_couplers_to_auto_pc_BID : STD_LOGIC_VECTOR ( 11 downto 0 );
  signal s00_couplers_to_auto_pc_BREADY : STD_LOGIC;
  signal s00_couplers_to_auto_pc_BRESP : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal s00_couplers_to_auto_pc_BVALID : STD_LOGIC;
  signal s00_couplers_to_auto_pc_RDATA : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal s00_couplers_to_auto_pc_RID : STD_LOGIC_VECTOR ( 11 downto 0 );
  signal s00_couplers_to_auto_pc_RLAST : STD_LOGIC;
  signal s00_couplers_to_auto_pc_RREADY : STD_LOGIC;
  signal s00_couplers_to_auto_pc_RRESP : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal s00_couplers_to_auto_pc_RVALID : STD_LOGIC;
  signal s00_couplers_to_auto_pc_WDATA : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal s00_couplers_to_auto_pc_WID : STD_LOGIC_VECTOR ( 11 downto 0 );
  signal s00_couplers_to_auto_pc_WLAST : STD_LOGIC;
  signal s00_couplers_to_auto_pc_WREADY : STD_LOGIC;
  signal s00_couplers_to_auto_pc_WSTRB : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal s00_couplers_to_auto_pc_WVALID : STD_LOGIC;
begin
  M_AXI_araddr(31 downto 0) <= auto_pc_to_s00_couplers_ARADDR(31 downto 0);
  M_AXI_arprot(2 downto 0) <= auto_pc_to_s00_couplers_ARPROT(2 downto 0);
  M_AXI_arvalid <= auto_pc_to_s00_couplers_ARVALID;
  M_AXI_awaddr(31 downto 0) <= auto_pc_to_s00_couplers_AWADDR(31 downto 0);
  M_AXI_awprot(2 downto 0) <= auto_pc_to_s00_couplers_AWPROT(2 downto 0);
  M_AXI_awvalid <= auto_pc_to_s00_couplers_AWVALID;
  M_AXI_bready <= auto_pc_to_s00_couplers_BREADY;
  M_AXI_rready <= auto_pc_to_s00_couplers_RREADY;
  M_AXI_wdata(31 downto 0) <= auto_pc_to_s00_couplers_WDATA(31 downto 0);
  M_AXI_wstrb(3 downto 0) <= auto_pc_to_s00_couplers_WSTRB(3 downto 0);
  M_AXI_wvalid <= auto_pc_to_s00_couplers_WVALID;
  S_ACLK_1 <= S_ACLK;
  S_ARESETN_1(0) <= S_ARESETN(0);
  S_AXI_arready <= s00_couplers_to_auto_pc_ARREADY;
  S_AXI_awready <= s00_couplers_to_auto_pc_AWREADY;
  S_AXI_bid(11 downto 0) <= s00_couplers_to_auto_pc_BID(11 downto 0);
  S_AXI_bresp(1 downto 0) <= s00_couplers_to_auto_pc_BRESP(1 downto 0);
  S_AXI_bvalid <= s00_couplers_to_auto_pc_BVALID;
  S_AXI_rdata(31 downto 0) <= s00_couplers_to_auto_pc_RDATA(31 downto 0);
  S_AXI_rid(11 downto 0) <= s00_couplers_to_auto_pc_RID(11 downto 0);
  S_AXI_rlast <= s00_couplers_to_auto_pc_RLAST;
  S_AXI_rresp(1 downto 0) <= s00_couplers_to_auto_pc_RRESP(1 downto 0);
  S_AXI_rvalid <= s00_couplers_to_auto_pc_RVALID;
  S_AXI_wready <= s00_couplers_to_auto_pc_WREADY;
  auto_pc_to_s00_couplers_ARREADY <= M_AXI_arready;
```

```
  auto_pc_to_s00_couplers_AWREADY <= M_AXI_awready;
  auto_pc_to_s00_couplers_BRESP(1 downto 0) <= M_AXI_bresp(1 downto 0);
  auto_pc_to_s00_couplers_BVALID <= M_AXI_bvalid;
  auto_pc_to_s00_couplers_RDATA(31 downto 0) <= M_AXI_rdata(31 downto 0);
  auto_pc_to_s00_couplers_RRESP(1 downto 0) <= M_AXI_rresp(1 downto 0);
  auto_pc_to_s00_couplers_RVALID <= M_AXI_rvalid;
  auto_pc_to_s00_couplers_WREADY <= M_AXI_wready;
  s00_couplers_to_auto_pc_ARADDR(31 downto 0) <= S_AXI_araddr(31 downto 0);
  s00_couplers_to_auto_pc_ARBURST(1 downto 0) <= S_AXI_arburst(1 downto 0);
  s00_couplers_to_auto_pc_ARCACHE(3 downto 0) <= S_AXI_arcache(3 downto 0);
  s00_couplers_to_auto_pc_ARID(11 downto 0) <= S_AXI_arid(11 downto 0);
  s00_couplers_to_auto_pc_ARLEN(3 downto 0) <= S_AXI_arlen(3 downto 0);
  s00_couplers_to_auto_pc_ARLOCK(1 downto 0) <= S_AXI_arlock(1 downto 0);
  s00_couplers_to_auto_pc_ARPROT(2 downto 0) <= S_AXI_arprot(2 downto 0);
  s00_couplers_to_auto_pc_ARQOS(3 downto 0) <= S_AXI_arqos(3 downto 0);
  s00_couplers_to_auto_pc_ARSIZE(2 downto 0) <= S_AXI_arsize(2 downto 0);
  s00_couplers_to_auto_pc_ARVALID <= S_AXI_arvalid;
  s00_couplers_to_auto_pc_AWADDR(31 downto 0) <= S_AXI_awaddr(31 downto 0);
  s00_couplers_to_auto_pc_AWBURST(1 downto 0) <= S_AXI_awburst(1 downto 0);
  s00_couplers_to_auto_pc_AWCACHE(3 downto 0) <= S_AXI_awcache(3 downto 0);
  s00_couplers_to_auto_pc_AWID(11 downto 0) <= S_AXI_awid(11 downto 0);
  s00_couplers_to_auto_pc_AWLEN(3 downto 0) <= S_AXI_awlen(3 downto 0);
  s00_couplers_to_auto_pc_AWLOCK(1 downto 0) <= S_AXI_awlock(1 downto 0);
  s00_couplers_to_auto_pc_AWPROT(2 downto 0) <= S_AXI_awprot(2 downto 0);
  s00_couplers_to_auto_pc_AWQOS(3 downto 0) <= S_AXI_awqos(3 downto 0);
  s00_couplers_to_auto_pc_AWSIZE(2 downto 0) <= S_AXI_awsize(2 downto 0);
  s00_couplers_to_auto_pc_AWVALID <= S_AXI_awvalid;
  s00_couplers_to_auto_pc_BREADY <= S_AXI_bready;
  s00_couplers_to_auto_pc_RREADY <= S_AXI_rready;
  s00_couplers_to_auto_pc_WDATA(31 downto 0) <= S_AXI_wdata(31 downto 0);
  s00_couplers_to_auto_pc_WID(11 downto 0) <= S_AXI_wid(11 downto 0);
  s00_couplers_to_auto_pc_WLAST <= S_AXI_wlast;
  s00_couplers_to_auto_pc_WSTRB(3 downto 0) <= S_AXI_wstrb(3 downto 0);
  s00_couplers_to_auto_pc_WVALID <= S_AXI_wvalid;
auto_pc: component simple_axi_interface_auto_pc_0
    port map (
     aclk => S_ACLK_1,
     aresetn => S_ARESETN_1(0),
     m_axi_araddr(31 downto 0) => auto_pc_to_s00_couplers_ARADDR(31 downto 0),
     m_axi_arprot(2 downto 0) => auto_pc_to_s00_couplers_ARPROT(2 downto 0),
     m_axi_arready => auto_pc_to_s00_couplers_ARREADY,
     m_axi_arvalid => auto_pc_to_s00_couplers_ARVALID,
     m_axi_awaddr(31 downto 0) => auto_pc_to_s00_couplers_AWADDR(31 downto 0),
     m_axi_awprot(2 downto 0) => auto_pc_to_s00_couplers_AWPROT(2 downto 0),
     m_axi_awready => auto_pc_to_s00_couplers_AWREADY,
     m_axi_awvalid => auto_pc_to_s00_couplers_AWVALID,
     m_axi_bready => auto_pc_to_s00_couplers_BREADY,
     m_axi_bresp(1 downto 0) => auto_pc_to_s00_couplers_BRESP(1 downto 0),
     m_axi_bvalid => auto_pc_to_s00_couplers_BVALID,
     m_axi_rdata(31 downto 0) => auto_pc_to_s00_couplers_RDATA(31 downto 0),
     m_axi_rready => auto_pc_to_s00_couplers_RREADY,
     m_axi_rresp(1 downto 0) => auto_pc_to_s00_couplers_RRESP(1 downto 0),
     m_axi_rvalid => auto_pc_to_s00_couplers_RVALID,
     m_axi_wdata(31 downto 0) => auto_pc_to_s00_couplers_WDATA(31 downto 0),
     m_axi_wready => auto_pc_to_s00_couplers_WREADY,
     m_axi_wstrb(3 downto 0) => auto_pc_to_s00_couplers_WSTRB(3 downto 0),
     m_axi_wvalid => auto_pc_to_s00_couplers_WVALID,
     s_axi_araddr(31 downto 0) => s00_couplers_to_auto_pc_ARADDR(31 downto 0),
```

```
      s_axi_arburst(1 downto 0) => s00_couplers_to_auto_pc_ARBURST(1 downto 0),
      s_axi_arcache(3 downto 0) => s00_couplers_to_auto_pc_ARCACHE(3 downto 0),
      s_axi_arid(11 downto 0) => s00_couplers_to_auto_pc_ARID(11 downto 0),
      s_axi_arlen(3 downto 0) => s00_couplers_to_auto_pc_ARLEN(3 downto 0),
      s_axi_arlock(1 downto 0) => s00_couplers_to_auto_pc_ARLOCK(1 downto 0),
      s_axi_arprot(2 downto 0) => s00_couplers_to_auto_pc_ARPROT(2 downto 0),
      s_axi_arqos(3 downto 0) => s00_couplers_to_auto_pc_ARQOS(3 downto 0),
      s_axi_arready => s00_couplers_to_auto_pc_ARREADY,
      s_axi_arsize(2 downto 0) => s00_couplers_to_auto_pc_ARSIZE(2 downto 0),
      s_axi_arvalid => s00_couplers_to_auto_pc_ARVALID,
      s_axi_awaddr(31 downto 0) => s00_couplers_to_auto_pc_AWADDR(31 downto 0),
      s_axi_awburst(1 downto 0) => s00_couplers_to_auto_pc_AWBURST(1 downto 0),
      s_axi_awcache(3 downto 0) => s00_couplers_to_auto_pc_AWCACHE(3 downto 0),
      s_axi_awid(11 downto 0) => s00_couplers_to_auto_pc_AWID(11 downto 0),
      s_axi_awlen(3 downto 0) => s00_couplers_to_auto_pc_AWLEN(3 downto 0),
      s_axi_awlock(1 downto 0) => s00_couplers_to_auto_pc_AWLOCK(1 downto 0),
      s_axi_awprot(2 downto 0) => s00_couplers_to_auto_pc_AWPROT(2 downto 0),
      s_axi_awqos(3 downto 0) => s00_couplers_to_auto_pc_AWQOS(3 downto 0),
      s_axi_awready => s00_couplers_to_auto_pc_AWREADY,
      s_axi_awsize(2 downto 0) => s00_couplers_to_auto_pc_AWSIZE(2 downto 0),
      s_axi_awvalid => s00_couplers_to_auto_pc_AWVALID,
      s_axi_bid(11 downto 0) => s00_couplers_to_auto_pc_BID(11 downto 0),
      s_axi_bready => s00_couplers_to_auto_pc_BREADY,
      s_axi_bresp(1 downto 0) => s00_couplers_to_auto_pc_BRESP(1 downto 0),
      s_axi_bvalid => s00_couplers_to_auto_pc_BVALID,
      s_axi_rdata(31 downto 0) => s00_couplers_to_auto_pc_RDATA(31 downto 0),
      s_axi_rid(11 downto 0) => s00_couplers_to_auto_pc_RID(11 downto 0),
      s_axi_rlast => s00_couplers_to_auto_pc_RLAST,
      s_axi_rready => s00_couplers_to_auto_pc_RREADY,
      s_axi_rresp(1 downto 0) => s00_couplers_to_auto_pc_RRESP(1 downto 0),
      s_axi_rvalid => s00_couplers_to_auto_pc_RVALID,
      s_axi_wdata(31 downto 0) => s00_couplers_to_auto_pc_WDATA(31 downto 0),
      s_axi_wid(11 downto 0) => s00_couplers_to_auto_pc_WID(11 downto 0),
      s_axi_wlast => s00_couplers_to_auto_pc_WLAST,
      s_axi_wready => s00_couplers_to_auto_pc_WREADY,
      s_axi_wstrb(3 downto 0) => s00_couplers_to_auto_pc_WSTRB(3 downto 0),
      s_axi_wvalid => s00_couplers_to_auto_pc_WVALID
    );
end STRUCTURE;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity simple_axi_interface_processing_system7_0_axi_periph_0 is
  port (
    ACLK : in STD_LOGIC;
    ARESETN : in STD_LOGIC_VECTOR ( 0 to 0 );
    M00_ACLK : in STD_LOGIC;
    M00_ARESETN : in STD_LOGIC_VECTOR ( 0 to 0 );
    M00_AXI_araddr : out STD_LOGIC_VECTOR ( 31 downto 0 );
    M00_AXI_arprot : out STD_LOGIC_VECTOR ( 2 downto 0 );
    M00_AXI_arready : in STD_LOGIC;
    M00_AXI_arvalid : out STD_LOGIC;
    M00_AXI_awaddr : out STD_LOGIC_VECTOR ( 31 downto 0 );
    M00_AXI_awprot : out STD_LOGIC_VECTOR ( 2 downto 0 );
    M00_AXI_awready : in STD_LOGIC;
    M00_AXI_awvalid : out STD_LOGIC;
    M00_AXI_bready : out STD_LOGIC;
```

```
    M00_AXI_bresp : in STD_LOGIC_VECTOR ( 1 downto 0 );
    M00_AXI_bvalid : in STD_LOGIC;
    M00_AXI_rdata : in STD_LOGIC_VECTOR ( 31 downto 0 );
    M00_AXI_rready : out STD_LOGIC;
    M00_AXI_rresp : in STD_LOGIC_VECTOR ( 1 downto 0 );
    M00_AXI_rvalid : in STD_LOGIC;
    M00_AXI_wdata : out STD_LOGIC_VECTOR ( 31 downto 0 );
    M00_AXI_wready : in STD_LOGIC;
    M00_AXI_wstrb : out STD_LOGIC_VECTOR ( 3 downto 0 );
    M00_AXI_wvalid : out STD_LOGIC;
    S00_ACLK : in STD_LOGIC;
    S00_ARESETN : in STD_LOGIC_VECTOR ( 0 to 0 );
    S00_AXI_araddr : in STD_LOGIC_VECTOR ( 31 downto 0 );
    S00_AXI_arburst : in STD_LOGIC_VECTOR ( 1 downto 0 );
    S00_AXI_arcache : in STD_LOGIC_VECTOR ( 3 downto 0 );
    S00_AXI_arid : in STD_LOGIC_VECTOR ( 11 downto 0 );
    S00_AXI_arlen : in STD_LOGIC_VECTOR ( 3 downto 0 );
    S00_AXI_arlock : in STD_LOGIC_VECTOR ( 1 downto 0 );
    S00_AXI_arprot : in STD_LOGIC_VECTOR ( 2 downto 0 );
    S00_AXI_arqos : in STD_LOGIC_VECTOR ( 3 downto 0 );
    S00_AXI_arready : out STD_LOGIC;
    S00_AXI_arsize : in STD_LOGIC_VECTOR ( 2 downto 0 );
    S00_AXI_arvalid : in STD_LOGIC;
    S00_AXI_awaddr : in STD_LOGIC_VECTOR ( 31 downto 0 );
    S00_AXI_awburst : in STD_LOGIC_VECTOR ( 1 downto 0 );
    S00_AXI_awcache : in STD_LOGIC_VECTOR ( 3 downto 0 );
    S00_AXI_awid : in STD_LOGIC_VECTOR ( 11 downto 0 );
    S00_AXI_awlen : in STD_LOGIC_VECTOR ( 3 downto 0 );
    S00_AXI_awlock : in STD_LOGIC_VECTOR ( 1 downto 0 );
    S00_AXI_awprot : in STD_LOGIC_VECTOR ( 2 downto 0 );
    S00_AXI_awqos : in STD_LOGIC_VECTOR ( 3 downto 0 );
    S00_AXI_awready : out STD_LOGIC;
    S00_AXI_awsize : in STD_LOGIC_VECTOR ( 2 downto 0 );
    S00_AXI_awvalid : in STD_LOGIC;
    S00_AXI_bid : out STD_LOGIC_VECTOR ( 11 downto 0 );
    S00_AXI_bready : in STD_LOGIC;
    S00_AXI_bresp : out STD_LOGIC_VECTOR ( 1 downto 0 );
    S00_AXI_bvalid : out STD_LOGIC;
    S00_AXI_rdata : out STD_LOGIC_VECTOR ( 31 downto 0 );
    S00_AXI_rid : out STD_LOGIC_VECTOR ( 11 downto 0 );
    S00_AXI_rlast : out STD_LOGIC;
    S00_AXI_rready : in STD_LOGIC;
    S00_AXI_rresp : out STD_LOGIC_VECTOR ( 1 downto 0 );
    S00_AXI_rvalid : out STD_LOGIC;
    S00_AXI_wdata : in STD_LOGIC_VECTOR ( 31 downto 0 );
    S00_AXI_wid : in STD_LOGIC_VECTOR ( 11 downto 0 );
    S00_AXI_wlast : in STD_LOGIC;
    S00_AXI_wready : out STD_LOGIC;
    S00_AXI_wstrb : in STD_LOGIC_VECTOR ( 3 downto 0 );
    S00_AXI_wvalid : in STD_LOGIC
  );
end simple_axi_interface_processing_system7_0_axi_periph_0;

architecture STRUCTURE of simple_axi_interface_processing_system7_0_axi_periph_0 is
  signal S00_ACLK_1 : STD_LOGIC;
  signal S00_ARESETN_1 : STD_LOGIC_VECTOR ( 0 to 0 );
  signal processing_system7_0_axi_periph_ACLK_net : STD_LOGIC;
  signal processing_system7_0_axi_periph_ARESETN_net : STD_LOGIC_VECTOR ( 0 to 0 );
```

```
  signal processing_system7_0_axi_periph_to_s00_couplers_ARADDR : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_ARBURST : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_ARCACHE : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_ARID : STD_LOGIC_VECTOR ( 11 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_ARLEN : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_ARLOCK : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_ARPROT : STD_LOGIC_VECTOR ( 2 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_ARQOS : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_ARREADY : STD_LOGIC;
  signal processing_system7_0_axi_periph_to_s00_couplers_ARSIZE : STD_LOGIC_VECTOR ( 2 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_ARVALID : STD_LOGIC;
  signal processing_system7_0_axi_periph_to_s00_couplers_AWADDR : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_AWBURST : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_AWCACHE : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_AWID : STD_LOGIC_VECTOR ( 11 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_AWLEN : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_AWLOCK : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_AWPROT : STD_LOGIC_VECTOR ( 2 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_AWQOS : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_AWREADY : STD_LOGIC;
  signal processing_system7_0_axi_periph_to_s00_couplers_AWSIZE : STD_LOGIC_VECTOR ( 2 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_AWVALID : STD_LOGIC;
  signal processing_system7_0_axi_periph_to_s00_couplers_BID : STD_LOGIC_VECTOR ( 11 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_BREADY : STD_LOGIC;
  signal processing_system7_0_axi_periph_to_s00_couplers_BRESP : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_BVALID : STD_LOGIC;
  signal processing_system7_0_axi_periph_to_s00_couplers_RDATA : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_RID : STD_LOGIC_VECTOR ( 11 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_RLAST : STD_LOGIC;
  signal processing_system7_0_axi_periph_to_s00_couplers_RREADY : STD_LOGIC;
  signal processing_system7_0_axi_periph_to_s00_couplers_RRESP : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_RVALID : STD_LOGIC;
  signal processing_system7_0_axi_periph_to_s00_couplers_WDATA : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_WID : STD_LOGIC_VECTOR ( 11 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_WLAST : STD_LOGIC;
  signal processing_system7_0_axi_periph_to_s00_couplers_WREADY : STD_LOGIC;
  signal processing_system7_0_axi_periph_to_s00_couplers_WSTRB : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal processing_system7_0_axi_periph_to_s00_couplers_WVALID : STD_LOGIC;
  signal s00_couplers_to_processing_system7_0_axi_periph_ARADDR : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal s00_couplers_to_processing_system7_0_axi_periph_ARPROT : STD_LOGIC_VECTOR ( 2 downto 0 );
  signal s00_couplers_to_processing_system7_0_axi_periph_ARREADY : STD_LOGIC;
  signal s00_couplers_to_processing_system7_0_axi_periph_ARVALID : STD_LOGIC;
  signal s00_couplers_to_processing_system7_0_axi_periph_AWADDR : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal s00_couplers_to_processing_system7_0_axi_periph_AWPROT : STD_LOGIC_VECTOR ( 2 downto 0 );
  signal s00_couplers_to_processing_system7_0_axi_periph_AWREADY : STD_LOGIC;
  signal s00_couplers_to_processing_system7_0_axi_periph_AWVALID : STD_LOGIC;
  signal s00_couplers_to_processing_system7_0_axi_periph_BREADY : STD_LOGIC;
  signal s00_couplers_to_processing_system7_0_axi_periph_BRESP : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal s00_couplers_to_processing_system7_0_axi_periph_BVALID : STD_LOGIC;
  signal s00_couplers_to_processing_system7_0_axi_periph_RDATA : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal s00_couplers_to_processing_system7_0_axi_periph_RREADY : STD_LOGIC;
  signal s00_couplers_to_processing_system7_0_axi_periph_RRESP : STD_LOGIC_VECTOR ( 1 downto 0 );
  signal s00_couplers_to_processing_system7_0_axi_periph_RVALID : STD_LOGIC;
  signal s00_couplers_to_processing_system7_0_axi_periph_WDATA : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal s00_couplers_to_processing_system7_0_axi_periph_WREADY : STD_LOGIC;
  signal s00_couplers_to_processing_system7_0_axi_periph_WSTRB : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal s00_couplers_to_processing_system7_0_axi_periph_WVALID : STD_LOGIC;
begin
```

```
M00_AXI_araddr(31 downto 0) <= s00_couplers_to_processing_system7_0_axi_periph_ARADDR(31 downto 0);
M00_AXI_arprot(2 downto 0) <= s00_couplers_to_processing_system7_0_axi_periph_ARPROT(2 downto 0);
M00_AXI_arvalid <= s00_couplers_to_processing_system7_0_axi_periph_ARVALID;
M00_AXI_awaddr(31 downto 0) <= s00_couplers_to_processing_system7_0_axi_periph_AWADDR(31 downto 0);
M00_AXI_awprot(2 downto 0) <= s00_couplers_to_processing_system7_0_axi_periph_AWPROT(2 downto 0);
M00_AXI_awvalid <= s00_couplers_to_processing_system7_0_axi_periph_AWVALID;
M00_AXI_bready <= s00_couplers_to_processing_system7_0_axi_periph_BREADY;
M00_AXI_rready <= s00_couplers_to_processing_system7_0_axi_periph_RREADY;
M00_AXI_wdata(31 downto 0) <= s00_couplers_to_processing_system7_0_axi_periph_WDATA(31 downto 0);
M00_AXI_wstrb(3 downto 0) <= s00_couplers_to_processing_system7_0_axi_periph_WSTRB(3 downto 0);
M00_AXI_wvalid <= s00_couplers_to_processing_system7_0_axi_periph_WVALID;
S00_ACLK_1 <= S00_ACLK;
S00_ARESETN_1(0) <= S00_ARESETN(0);
S00_AXI_arready <= processing_system7_0_axi_periph_to_s00_couplers_ARREADY;
S00_AXI_awready <= processing_system7_0_axi_periph_to_s00_couplers_AWREADY;
S00_AXI_bid(11 downto 0) <= processing_system7_0_axi_periph_to_s00_couplers_BID(11 downto 0);
S00_AXI_bresp(1 downto 0) <= processing_system7_0_axi_periph_to_s00_couplers_BRESP(1 downto 0);
S00_AXI_bvalid <= processing_system7_0_axi_periph_to_s00_couplers_BVALID;
S00_AXI_rdata(31 downto 0) <= processing_system7_0_axi_periph_to_s00_couplers_RDATA(31 downto 0);
S00_AXI_rid(11 downto 0) <= processing_system7_0_axi_periph_to_s00_couplers_RID(11 downto 0);
S00_AXI_rlast <= processing_system7_0_axi_periph_to_s00_couplers_RLAST;
S00_AXI_rresp(1 downto 0) <= processing_system7_0_axi_periph_to_s00_couplers_RRESP(1 downto 0);
S00_AXI_rvalid <= processing_system7_0_axi_periph_to_s00_couplers_RVALID;
S00_AXI_wready <= processing_system7_0_axi_periph_to_s00_couplers_WREADY;
processing_system7_0_axi_periph_ACLK_net <= M00_ACLK;
processing_system7_0_axi_periph_ARESETN_net(0) <= M00_ARESETN(0);
processing_system7_0_axi_periph_to_s00_couplers_ARADDR(31 downto 0) <= S00_AXI_araddr(31 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_ARBURST(1 downto 0) <= S00_AXI_arburst(1 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_ARCACHE(3 downto 0) <= S00_AXI_arcache(3 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_ARID(11 downto 0) <= S00_AXI_arid(11 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_ARLEN(3 downto 0) <= S00_AXI_arlen(3 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_ARLOCK(1 downto 0) <= S00_AXI_arlock(1 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_ARPROT(2 downto 0) <= S00_AXI_arprot(2 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_ARQOS(3 downto 0) <= S00_AXI_arqos(3 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_ARSIZE(2 downto 0) <= S00_AXI_arsize(2 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_ARVALID <= S00_AXI_arvalid;
processing_system7_0_axi_periph_to_s00_couplers_AWADDR(31 downto 0) <= S00_AXI_awaddr(31 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_AWBURST(1 downto 0) <= S00_AXI_awburst(1 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_AWCACHE(3 downto 0) <= S00_AXI_awcache(3 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_AWID(11 downto 0) <= S00_AXI_awid(11 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_AWLEN(3 downto 0) <= S00_AXI_awlen(3 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_AWLOCK(1 downto 0) <= S00_AXI_awlock(1 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_AWPROT(2 downto 0) <= S00_AXI_awprot(2 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_AWQOS(3 downto 0) <= S00_AXI_awqos(3 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_AWSIZE(2 downto 0) <= S00_AXI_awsize(2 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_AWVALID <= S00_AXI_awvalid;
processing_system7_0_axi_periph_to_s00_couplers_BREADY <= S00_AXI_bready;
processing_system7_0_axi_periph_to_s00_couplers_RREADY <= S00_AXI_rready;
processing_system7_0_axi_periph_to_s00_couplers_WDATA(31 downto 0) <= S00_AXI_wdata(31 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_WID(11 downto 0) <= S00_AXI_wid(11 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_WLAST <= S00_AXI_wlast;
processing_system7_0_axi_periph_to_s00_couplers_WSTRB(3 downto 0) <= S00_AXI_wstrb(3 downto 0);
processing_system7_0_axi_periph_to_s00_couplers_WVALID <= S00_AXI_wvalid;
s00_couplers_to_processing_system7_0_axi_periph_ARREADY <= M00_AXI_arready;
s00_couplers_to_processing_system7_0_axi_periph_AWREADY <= M00_AXI_awready;
s00_couplers_to_processing_system7_0_axi_periph_BRESP(1 downto 0) <= M00_AXI_bresp(1 downto 0);
s00_couplers_to_processing_system7_0_axi_periph_BVALID <= M00_AXI_bvalid;
s00_couplers_to_processing_system7_0_axi_periph_RDATA(31 downto 0) <= M00_AXI_rdata(31 downto 0);
```

```
  s00_couplers_to_processing_system7_0_axi_periph_RRESP(1 downto 0) <= M00_AXI_rresp(1 downto 0);
  s00_couplers_to_processing_system7_0_axi_periph_RVALID <= M00_AXI_rvalid;
  s00_couplers_to_processing_system7_0_axi_periph_WREADY <= M00_AXI_wready;
s00_couplers: entity work.s00_couplers_imp_O2VN7B
    port map (
     M_ACLK => processing_system7_0_axi_periph_ACLK_net,
     M_ARESETN(0) => processing_system7_0_axi_periph_ARESETN_net(0),
     M_AXI_araddr(31 downto 0) => s00_couplers_to_processing_system7_0_axi_periph_ARADDR(31 downto 0),
     M_AXI_arprot(2 downto 0) => s00_couplers_to_processing_system7_0_axi_periph_ARPROT(2 downto 0),
     M_AXI_arready => s00_couplers_to_processing_system7_0_axi_periph_ARREADY,
     M_AXI_arvalid => s00_couplers_to_processing_system7_0_axi_periph_ARVALID,
     M_AXI_awaddr(31 downto 0) => s00_couplers_to_processing_system7_0_axi_periph_AWADDR(31 downto 0),
     M_AXI_awprot(2 downto 0) => s00_couplers_to_processing_system7_0_axi_periph_AWPROT(2 downto 0),
     M_AXI_awready => s00_couplers_to_processing_system7_0_axi_periph_AWREADY,
     M_AXI_awvalid => s00_couplers_to_processing_system7_0_axi_periph_AWVALID,
     M_AXI_bready => s00_couplers_to_processing_system7_0_axi_periph_BREADY,
     M_AXI_bresp(1 downto 0) => s00_couplers_to_processing_system7_0_axi_periph_BRESP(1 downto 0),
     M_AXI_bvalid => s00_couplers_to_processing_system7_0_axi_periph_BVALID,
     M_AXI_rdata(31 downto 0) => s00_couplers_to_processing_system7_0_axi_periph_RDATA(31 downto 0),
     M_AXI_rready => s00_couplers_to_processing_system7_0_axi_periph_RREADY,
     M_AXI_rresp(1 downto 0) => s00_couplers_to_processing_system7_0_axi_periph_RRESP(1 downto 0),
     M_AXI_rvalid => s00_couplers_to_processing_system7_0_axi_periph_RVALID,
     M_AXI_wdata(31 downto 0) => s00_couplers_to_processing_system7_0_axi_periph_WDATA(31 downto 0),
     M_AXI_wready => s00_couplers_to_processing_system7_0_axi_periph_WREADY,
     M_AXI_wstrb(3 downto 0) => s00_couplers_to_processing_system7_0_axi_periph_WSTRB(3 downto 0),
     M_AXI_wvalid => s00_couplers_to_processing_system7_0_axi_periph_WVALID,
     S_ACLK => S00_ACLK_1,
     S_ARESETN(0) => S00_ARESETN_1(0),
     S_AXI_araddr(31 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_ARADDR(31 downto 0),
     S_AXI_arburst(1 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_ARBURST(1 downto 0),
     S_AXI_arcache(3 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_ARCACHE(3 downto 0),
     S_AXI_arid(11 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_ARID(11 downto 0),
     S_AXI_arlen(3 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_ARLEN(3 downto 0),
     S_AXI_arlock(1 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_ARLOCK(1 downto 0),
     S_AXI_arprot(2 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_ARPROT(2 downto 0),
     S_AXI_arqos(3 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_ARQOS(3 downto 0),
     S_AXI_arready => processing_system7_0_axi_periph_to_s00_couplers_ARREADY,
     S_AXI_arsize(2 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_ARSIZE(2 downto 0),
     S_AXI_arvalid => processing_system7_0_axi_periph_to_s00_couplers_ARVALID,
     S_AXI_awaddr(31 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_AWADDR(31 downto 0),
     S_AXI_awburst(1 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_AWBURST(1 downto 0),
     S_AXI_awcache(3 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_AWCACHE(3 downto 0),
     S_AXI_awid(11 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_AWID(11 downto 0),
     S_AXI_awlen(3 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_AWLEN(3 downto 0),
     S_AXI_awlock(1 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_AWLOCK(1 downto 0),
     S_AXI_awprot(2 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_AWPROT(2 downto 0),
     S_AXI_awqos(3 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_AWQOS(3 downto 0),
     S_AXI_awready => processing_system7_0_axi_periph_to_s00_couplers_AWREADY,
     S_AXI_awsize(2 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_AWSIZE(2 downto 0),
     S_AXI_awvalid => processing_system7_0_axi_periph_to_s00_couplers_AWVALID,
     S_AXI_bid(11 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_BID(11 downto 0),
     S_AXI_bready => processing_system7_0_axi_periph_to_s00_couplers_BREADY,
     S_AXI_bresp(1 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_BRESP(1 downto 0),
     S_AXI_bvalid => processing_system7_0_axi_periph_to_s00_couplers_BVALID,
     S_AXI_rdata(31 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_RDATA(31 downto 0),
     S_AXI_rid(11 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_RID(11 downto 0),
     S_AXI_rlast => processing_system7_0_axi_periph_to_s00_couplers_RLAST,
     S_AXI_rready => processing_system7_0_axi_periph_to_s00_couplers_RREADY,
```

```
      S_AXI_rresp(1 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_RRESP(1 downto 0),
      S_AXI_rvalid => processing_system7_0_axi_periph_to_s00_couplers_RVALID,
      S_AXI_wdata(31 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_WDATA(31 downto 0),
      S_AXI_wid(11 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_WID(11 downto 0),
      S_AXI_wlast => processing_system7_0_axi_periph_to_s00_couplers_WLAST,
      S_AXI_wready => processing_system7_0_axi_periph_to_s00_couplers_WREADY,
      S_AXI_wstrb(3 downto 0) => processing_system7_0_axi_periph_to_s00_couplers_WSTRB(3 downto 0),
      S_AXI_wvalid => processing_system7_0_axi_periph_to_s00_couplers_WVALID
    );
end STRUCTURE;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity simple_axi_interface is
  port (
    DDR_addr : inout STD_LOGIC_VECTOR ( 14 downto 0 );
    DDR_ba : inout STD_LOGIC_VECTOR ( 2 downto 0 );
    DDR_cas_n : inout STD_LOGIC;
    DDR_ck_n : inout STD_LOGIC;
    DDR_ck_p : inout STD_LOGIC;
    DDR_cke : inout STD_LOGIC;
    DDR_cs_n : inout STD_LOGIC;
    DDR_dm : inout STD_LOGIC_VECTOR ( 3 downto 0 );
    DDR_dq : inout STD_LOGIC_VECTOR ( 31 downto 0 );
    DDR_dqs_n : inout STD_LOGIC_VECTOR ( 3 downto 0 );
    DDR_dqs_p : inout STD_LOGIC_VECTOR ( 3 downto 0 );
    DDR_odt : inout STD_LOGIC;
    DDR_ras_n : inout STD_LOGIC;
    DDR_reset_n : inout STD_LOGIC;
    DDR_we_n : inout STD_LOGIC;
    FIXED_IO_ddr_vrn : inout STD_LOGIC;
    FIXED_IO_ddr_vrp : inout STD_LOGIC;
    FIXED_IO_mio : inout STD_LOGIC_VECTOR ( 53 downto 0 );
    FIXED_IO_ps_clk : inout STD_LOGIC;
    FIXED_IO_ps_porb : inout STD_LOGIC;
    FIXED_IO_ps_srstb : inout STD_LOGIC
  );
  attribute CORE_GENERATION_INFO : string;
  attribute CORE_GENERATION_INFO of simple_axi_interface : entity is "simple_axi_interface,IP_Integrator,
{x_ipVendor=xilinx.com,x_ipLibrary=BlockDiagram,x_ipName=simple_axi_interface,x_ipVersion=1.00.a,x_ipLangua
ge=VHDL,numBlks=6,numReposBlks=4,numNonXlnxBlks=1,numHierBlks=2,maxHierDepth=0,da_axi4_cnt=1,da_ps
7_cnt=1,synth_mode=Global}";
  attribute HW_HANDOFF : string;
  attribute HW_HANDOFF of simple_axi_interface : entity is "simple_axi_interface.hwdef";
end simple_axi_interface;

architecture STRUCTURE of simple_axi_interface is
  component simple_axi_interface_processing_system7_0_0 is
  port (
    M_AXI_GP0_ARVALID : out STD_LOGIC;
    M_AXI_GP0_AWVALID : out STD_LOGIC;
    M_AXI_GP0_BREADY : out STD_LOGIC;
    M_AXI_GP0_RREADY : out STD_LOGIC;
    M_AXI_GP0_WLAST : out STD_LOGIC;
    M_AXI_GP0_WVALID : out STD_LOGIC;
    M_AXI_GP0_ARID : out STD_LOGIC_VECTOR ( 11 downto 0 );
    M_AXI_GP0_AWID : out STD_LOGIC_VECTOR ( 11 downto 0 );
```

```
  M_AXI_GP0_WID : out STD_LOGIC_VECTOR ( 11 downto 0 );
  M_AXI_GP0_ARBURST : out STD_LOGIC_VECTOR ( 1 downto 0 );
  M_AXI_GP0_ARLOCK : out STD_LOGIC_VECTOR ( 1 downto 0 );
  M_AXI_GP0_ARSIZE : out STD_LOGIC_VECTOR ( 2 downto 0 );
  M_AXI_GP0_AWBURST : out STD_LOGIC_VECTOR ( 1 downto 0 );
  M_AXI_GP0_AWLOCK : out STD_LOGIC_VECTOR ( 1 downto 0 );
  M_AXI_GP0_AWSIZE : out STD_LOGIC_VECTOR ( 2 downto 0 );
  M_AXI_GP0_ARPROT : out STD_LOGIC_VECTOR ( 2 downto 0 );
  M_AXI_GP0_AWPROT : out STD_LOGIC_VECTOR ( 2 downto 0 );
  M_AXI_GP0_ARADDR : out STD_LOGIC_VECTOR ( 31 downto 0 );
  M_AXI_GP0_AWADDR : out STD_LOGIC_VECTOR ( 31 downto 0 );
  M_AXI_GP0_WDATA : out STD_LOGIC_VECTOR ( 31 downto 0 );
  M_AXI_GP0_ARCACHE : out STD_LOGIC_VECTOR ( 3 downto 0 );
  M_AXI_GP0_ARLEN : out STD_LOGIC_VECTOR ( 3 downto 0 );
  M_AXI_GP0_ARQOS : out STD_LOGIC_VECTOR ( 3 downto 0 );
  M_AXI_GP0_AWCACHE : out STD_LOGIC_VECTOR ( 3 downto 0 );
  M_AXI_GP0_AWLEN : out STD_LOGIC_VECTOR ( 3 downto 0 );
  M_AXI_GP0_AWQOS : out STD_LOGIC_VECTOR ( 3 downto 0 );
  M_AXI_GP0_WSTRB : out STD_LOGIC_VECTOR ( 3 downto 0 );
  M_AXI_GP0_ACLK : in STD_LOGIC;
  M_AXI_GP0_ARREADY : in STD_LOGIC;
  M_AXI_GP0_AWREADY : in STD_LOGIC;
  M_AXI_GP0_BVALID : in STD_LOGIC;
  M_AXI_GP0_RLAST : in STD_LOGIC;
  M_AXI_GP0_RVALID : in STD_LOGIC;
  M_AXI_GP0_WREADY : in STD_LOGIC;
  M_AXI_GP0_BID : in STD_LOGIC_VECTOR ( 11 downto 0 );
  M_AXI_GP0_RID : in STD_LOGIC_VECTOR ( 11 downto 0 );
  M_AXI_GP0_BRESP : in STD_LOGIC_VECTOR ( 1 downto 0 );
  M_AXI_GP0_RRESP : in STD_LOGIC_VECTOR ( 1 downto 0 );
  M_AXI_GP0_RDATA : in STD_LOGIC_VECTOR ( 31 downto 0 );
  FCLK_CLK0 : out STD_LOGIC;
  FCLK_RESET0_N : out STD_LOGIC;
  MIO : inout STD_LOGIC_VECTOR ( 53 downto 0 );
  DDR_CAS_n : inout STD_LOGIC;
  DDR_CKE : inout STD_LOGIC;
  DDR_Clk_n : inout STD_LOGIC;
  DDR_Clk : inout STD_LOGIC;
  DDR_CS_n : inout STD_LOGIC;
  DDR_DRSTB : inout STD_LOGIC;
  DDR_ODT : inout STD_LOGIC;
  DDR_RAS_n : inout STD_LOGIC;
  DDR_WEB : inout STD_LOGIC;
  DDR_BankAddr : inout STD_LOGIC_VECTOR ( 2 downto 0 );
  DDR_Addr : inout STD_LOGIC_VECTOR ( 14 downto 0 );
  DDR_VRN : inout STD_LOGIC;
  DDR_VRP : inout STD_LOGIC;
  DDR_DM : inout STD_LOGIC_VECTOR ( 3 downto 0 );
  DDR_DQ : inout STD_LOGIC_VECTOR ( 31 downto 0 );
  DDR_DQS_n : inout STD_LOGIC_VECTOR ( 3 downto 0 );
  DDR_DQS : inout STD_LOGIC_VECTOR ( 3 downto 0 );
  PS_SRSTB : inout STD_LOGIC;
  PS_CLK : inout STD_LOGIC;
  PS_PORB : inout STD_LOGIC
 );
 end component simple_axi_interface_processing_system7_0_0;
 component simple_axi_interface_rst_processing_system7_0_50M_0 is
 port (
```

```
    slowest_sync_clk : in STD_LOGIC;
    ext_reset_in : in STD_LOGIC;
    aux_reset_in : in STD_LOGIC;
    mb_debug_sys_rst : in STD_LOGIC;
    dcm_locked : in STD_LOGIC;
    mb_reset : out STD_LOGIC;
    bus_struct_reset : out STD_LOGIC_VECTOR ( 0 to 0 );
    peripheral_reset : out STD_LOGIC_VECTOR ( 0 to 0 );
    interconnect_aresetn : out STD_LOGIC_VECTOR ( 0 to 0 );
    peripheral_aresetn : out STD_LOGIC_VECTOR ( 0 to 0 )
  );
  end component simple_axi_interface_rst_processing_system7_0_50M_0;
  component simple_axi_interface_simple_axi_interface_0_0 is
  port (
    s00_axi_awaddr : in STD_LOGIC_VECTOR ( 3 downto 0 );
    s00_axi_awprot : in STD_LOGIC_VECTOR ( 2 downto 0 );
    s00_axi_awvalid : in STD_LOGIC;
    s00_axi_awready : out STD_LOGIC;
    s00_axi_wdata : in STD_LOGIC_VECTOR ( 31 downto 0 );
    s00_axi_wstrb : in STD_LOGIC_VECTOR ( 3 downto 0 );
    s00_axi_wvalid : in STD_LOGIC;
    s00_axi_wready : out STD_LOGIC;
    s00_axi_bresp : out STD_LOGIC_VECTOR ( 1 downto 0 );
    s00_axi_bvalid : out STD_LOGIC;
    s00_axi_bready : in STD_LOGIC;
    s00_axi_araddr : in STD_LOGIC_VECTOR ( 3 downto 0 );
    s00_axi_arprot : in STD_LOGIC_VECTOR ( 2 downto 0 );
    s00_axi_arvalid : in STD_LOGIC;
    s00_axi_arready : out STD_LOGIC;
    s00_axi_rdata : out STD_LOGIC_VECTOR ( 31 downto 0 );
    s00_axi_rresp : out STD_LOGIC_VECTOR ( 1 downto 0 );
    s00_axi_rvalid : out STD_LOGIC;
    s00_axi_rready : in STD_LOGIC;
    s00_axi_aclk : in STD_LOGIC;
    s00_axi_aresetn : in STD_LOGIC
  );
  end component simple_axi_interface_simple_axi_interface_0_0;
  signal processing_system7_0_DDR_ADDR : STD_LOGIC_VECTOR ( 14 downto 0 );
  signal processing_system7_0_DDR_BA : STD_LOGIC_VECTOR ( 2 downto 0 );
  signal processing_system7_0_DDR_CAS_N : STD_LOGIC;
  signal processing_system7_0_DDR_CKE : STD_LOGIC;
  signal processing_system7_0_DDR_CK_N : STD_LOGIC;
  signal processing_system7_0_DDR_CK_P : STD_LOGIC;
  signal processing_system7_0_DDR_CS_N : STD_LOGIC;
  signal processing_system7_0_DDR_DM : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal processing_system7_0_DDR_DQ : STD_LOGIC_VECTOR ( 31 downto 0 );
  signal processing_system7_0_DDR_DQS_N : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal processing_system7_0_DDR_DQS_P : STD_LOGIC_VECTOR ( 3 downto 0 );
  signal processing_system7_0_DDR_ODT : STD_LOGIC;
  signal processing_system7_0_DDR_RAS_N : STD_LOGIC;
  signal processing_system7_0_DDR_RESET_N : STD_LOGIC;
  signal processing_system7_0_DDR_WE_N : STD_LOGIC;
  signal processing_system7_0_FCLK_CLK0 : STD_LOGIC;
  signal processing_system7_0_FCLK_RESET0_N : STD_LOGIC;
  signal processing_system7_0_FIXED_IO_DDR_VRN : STD_LOGIC;
  signal processing_system7_0_FIXED_IO_DDR_VRP : STD_LOGIC;
  signal processing_system7_0_FIXED_IO_MIO : STD_LOGIC_VECTOR ( 53 downto 0 );
  signal processing_system7_0_FIXED_IO_PS_CLK : STD_LOGIC;
```

```
signal processing_system7_0_FIXED_IO_PS_PORB : STD_LOGIC;
signal processing_system7_0_FIXED_IO_PS_SRSTB : STD_LOGIC;
signal processing_system7_0_M_AXI_GP0_ARADDR : STD_LOGIC_VECTOR ( 31 downto 0 );
signal processing_system7_0_M_AXI_GP0_ARBURST : STD_LOGIC_VECTOR ( 1 downto 0 );
signal processing_system7_0_M_AXI_GP0_ARCACHE : STD_LOGIC_VECTOR ( 3 downto 0 );
signal processing_system7_0_M_AXI_GP0_ARID : STD_LOGIC_VECTOR ( 11 downto 0 );
signal processing_system7_0_M_AXI_GP0_ARLEN : STD_LOGIC_VECTOR ( 3 downto 0 );
signal processing_system7_0_M_AXI_GP0_ARLOCK : STD_LOGIC_VECTOR ( 1 downto 0 );
signal processing_system7_0_M_AXI_GP0_ARPROT : STD_LOGIC_VECTOR ( 2 downto 0 );
signal processing_system7_0_M_AXI_GP0_ARQOS : STD_LOGIC_VECTOR ( 3 downto 0 );
signal processing_system7_0_M_AXI_GP0_ARREADY : STD_LOGIC;
signal processing_system7_0_M_AXI_GP0_ARSIZE : STD_LOGIC_VECTOR ( 2 downto 0 );
signal processing_system7_0_M_AXI_GP0_ARVALID : STD_LOGIC;
signal processing_system7_0_M_AXI_GP0_AWADDR : STD_LOGIC_VECTOR ( 31 downto 0 );
signal processing_system7_0_M_AXI_GP0_AWBURST : STD_LOGIC_VECTOR ( 1 downto 0 );
signal processing_system7_0_M_AXI_GP0_AWCACHE : STD_LOGIC_VECTOR ( 3 downto 0 );
signal processing_system7_0_M_AXI_GP0_AWID : STD_LOGIC_VECTOR ( 11 downto 0 );
signal processing_system7_0_M_AXI_GP0_AWLEN : STD_LOGIC_VECTOR ( 3 downto 0 );
signal processing_system7_0_M_AXI_GP0_AWLOCK : STD_LOGIC_VECTOR ( 1 downto 0 );
signal processing_system7_0_M_AXI_GP0_AWPROT : STD_LOGIC_VECTOR ( 2 downto 0 );
signal processing_system7_0_M_AXI_GP0_AWQOS : STD_LOGIC_VECTOR ( 3 downto 0 );
signal processing_system7_0_M_AXI_GP0_AWREADY : STD_LOGIC;
signal processing_system7_0_M_AXI_GP0_AWSIZE : STD_LOGIC_VECTOR ( 2 downto 0 );
signal processing_system7_0_M_AXI_GP0_AWVALID : STD_LOGIC;
signal processing_system7_0_M_AXI_GP0_BID : STD_LOGIC_VECTOR ( 11 downto 0 );
signal processing_system7_0_M_AXI_GP0_BREADY : STD_LOGIC;
signal processing_system7_0_M_AXI_GP0_BRESP : STD_LOGIC_VECTOR ( 1 downto 0 );
signal processing_system7_0_M_AXI_GP0_BVALID : STD_LOGIC;
signal processing_system7_0_M_AXI_GP0_RDATA : STD_LOGIC_VECTOR ( 31 downto 0 );
signal processing_system7_0_M_AXI_GP0_RID : STD_LOGIC_VECTOR ( 11 downto 0 );
signal processing_system7_0_M_AXI_GP0_RLAST : STD_LOGIC;
signal processing_system7_0_M_AXI_GP0_RREADY : STD_LOGIC;
signal processing_system7_0_M_AXI_GP0_RRESP : STD_LOGIC_VECTOR ( 1 downto 0 );
signal processing_system7_0_M_AXI_GP0_RVALID : STD_LOGIC;
signal processing_system7_0_M_AXI_GP0_WDATA : STD_LOGIC_VECTOR ( 31 downto 0 );
signal processing_system7_0_M_AXI_GP0_WID : STD_LOGIC_VECTOR ( 11 downto 0 );
signal processing_system7_0_M_AXI_GP0_WLAST : STD_LOGIC;
signal processing_system7_0_M_AXI_GP0_WREADY : STD_LOGIC;
signal processing_system7_0_M_AXI_GP0_WSTRB : STD_LOGIC_VECTOR ( 3 downto 0 );
signal processing_system7_0_M_AXI_GP0_WVALID : STD_LOGIC;
signal processing_system7_0_axi_periph_M00_AXI_ARADDR : STD_LOGIC_VECTOR ( 31 downto 0 );
signal processing_system7_0_axi_periph_M00_AXI_ARPROT : STD_LOGIC_VECTOR ( 2 downto 0 );
signal processing_system7_0_axi_periph_M00_AXI_ARREADY : STD_LOGIC;
signal processing_system7_0_axi_periph_M00_AXI_ARVALID : STD_LOGIC;
signal processing_system7_0_axi_periph_M00_AXI_AWADDR : STD_LOGIC_VECTOR ( 31 downto 0 );
signal processing_system7_0_axi_periph_M00_AXI_AWPROT : STD_LOGIC_VECTOR ( 2 downto 0 );
signal processing_system7_0_axi_periph_M00_AXI_AWREADY : STD_LOGIC;
signal processing_system7_0_axi_periph_M00_AXI_AWVALID : STD_LOGIC;
signal processing_system7_0_axi_periph_M00_AXI_BREADY : STD_LOGIC;
signal processing_system7_0_axi_periph_M00_AXI_BRESP : STD_LOGIC_VECTOR ( 1 downto 0 );
signal processing_system7_0_axi_periph_M00_AXI_BVALID : STD_LOGIC;
signal processing_system7_0_axi_periph_M00_AXI_RDATA : STD_LOGIC_VECTOR ( 31 downto 0 );
signal processing_system7_0_axi_periph_M00_AXI_RREADY : STD_LOGIC;
signal processing_system7_0_axi_periph_M00_AXI_RRESP : STD_LOGIC_VECTOR ( 1 downto 0 );
signal processing_system7_0_axi_periph_M00_AXI_RVALID : STD_LOGIC;
signal processing_system7_0_axi_periph_M00_AXI_WDATA : STD_LOGIC_VECTOR ( 31 downto 0 );
signal processing_system7_0_axi_periph_M00_AXI_WREADY : STD_LOGIC;
signal processing_system7_0_axi_periph_M00_AXI_WSTRB : STD_LOGIC_VECTOR ( 3 downto 0 );
```

```
  signal processing_system7_0_axi_periph_M00_AXI_WVALID : STD_LOGIC;
  signal rst_processing_system7_0_50M_interconnect_aresetn : STD_LOGIC_VECTOR ( 0 to 0 );
  signal rst_processing_system7_0_50M_peripheral_aresetn : STD_LOGIC_VECTOR ( 0 to 0 );
  signal NLW_rst_processing_system7_0_50M_mb_reset_UNCONNECTED : STD_LOGIC;
  signal NLW_rst_processing_system7_0_50M_bus_struct_reset_UNCONNECTED : STD_LOGIC_VECTOR ( 0 to
0 );
  signal NLW_rst_processing_system7_0_50M_peripheral_reset_UNCONNECTED : STD_LOGIC_VECTOR ( 0 to
0 );
begin
processing_system7_0: component simple_axi_interface_processing_system7_0_0
    port map (
     DDR_Addr(14 downto 0) => DDR_addr(14 downto 0),
     DDR_BankAddr(2 downto 0) => DDR_ba(2 downto 0),
     DDR_CAS_n => DDR_cas_n,
     DDR_CKE => DDR_cke,
     DDR_CS_n => DDR_cs_n,
     DDR_Clk => DDR_ck_p,
     DDR_Clk_n => DDR_ck_n,
     DDR_DM(3 downto 0) => DDR_dm(3 downto 0),
     DDR_DQ(31 downto 0) => DDR_dq(31 downto 0),
     DDR_DQS(3 downto 0) => DDR_dqs_p(3 downto 0),
     DDR_DQS_n(3 downto 0) => DDR_dqs_n(3 downto 0),
     DDR_DRSTB => DDR_reset_n,
     DDR_ODT => DDR_odt,
     DDR_RAS_n => DDR_ras_n,
     DDR_VRN => FIXED_IO_ddr_vrn,
     DDR_VRP => FIXED_IO_ddr_vrp,
     DDR_WEB => DDR_we_n,
     FCLK_CLK0 => processing_system7_0_FCLK_CLK0,
     FCLK_RESET0_N => processing_system7_0_FCLK_RESET0_N,
     MIO(53 downto 0) => FIXED_IO_mio(53 downto 0),
     M_AXI_GP0_ACLK => processing_system7_0_FCLK_CLK0,
     M_AXI_GP0_ARADDR(31 downto 0) => processing_system7_0_M_AXI_GP0_ARADDR(31 downto 0),
     M_AXI_GP0_ARBURST(1 downto 0) => processing_system7_0_M_AXI_GP0_ARBURST(1 downto 0),
     M_AXI_GP0_ARCACHE(3 downto 0) => processing_system7_0_M_AXI_GP0_ARCACHE(3 downto 0),
     M_AXI_GP0_ARID(11 downto 0) => processing_system7_0_M_AXI_GP0_ARID(11 downto 0),
     M_AXI_GP0_ARLEN(3 downto 0) => processing_system7_0_M_AXI_GP0_ARLEN(3 downto 0),
     M_AXI_GP0_ARLOCK(1 downto 0) => processing_system7_0_M_AXI_GP0_ARLOCK(1 downto 0),
     M_AXI_GP0_ARPROT(2 downto 0) => processing_system7_0_M_AXI_GP0_ARPROT(2 downto 0),
     M_AXI_GP0_ARQOS(3 downto 0) => processing_system7_0_M_AXI_GP0_ARQOS(3 downto 0),
     M_AXI_GP0_ARREADY => processing_system7_0_M_AXI_GP0_ARREADY,
     M_AXI_GP0_ARSIZE(2 downto 0) => processing_system7_0_M_AXI_GP0_ARSIZE(2 downto 0),
     M_AXI_GP0_ARVALID => processing_system7_0_M_AXI_GP0_ARVALID,
     M_AXI_GP0_AWADDR(31 downto 0) => processing_system7_0_M_AXI_GP0_AWADDR(31 downto 0),
     M_AXI_GP0_AWBURST(1 downto 0) => processing_system7_0_M_AXI_GP0_AWBURST(1 downto 0),
     M_AXI_GP0_AWCACHE(3 downto 0) => processing_system7_0_M_AXI_GP0_AWCACHE(3 downto 0),
     M_AXI_GP0_AWID(11 downto 0) => processing_system7_0_M_AXI_GP0_AWID(11 downto 0),
     M_AXI_GP0_AWLEN(3 downto 0) => processing_system7_0_M_AXI_GP0_AWLEN(3 downto 0),
     M_AXI_GP0_AWLOCK(1 downto 0) => processing_system7_0_M_AXI_GP0_AWLOCK(1 downto 0),
     M_AXI_GP0_AWPROT(2 downto 0) => processing_system7_0_M_AXI_GP0_AWPROT(2 downto 0),
     M_AXI_GP0_AWQOS(3 downto 0) => processing_system7_0_M_AXI_GP0_AWQOS(3 downto 0),
     M_AXI_GP0_AWREADY => processing_system7_0_M_AXI_GP0_AWREADY,
     M_AXI_GP0_AWSIZE(2 downto 0) => processing_system7_0_M_AXI_GP0_AWSIZE(2 downto 0),
     M_AXI_GP0_AWVALID => processing_system7_0_M_AXI_GP0_AWVALID,
     M_AXI_GP0_BID(11 downto 0) => processing_system7_0_M_AXI_GP0_BID(11 downto 0),
     M_AXI_GP0_BREADY => processing_system7_0_M_AXI_GP0_BREADY,
     M_AXI_GP0_BRESP(1 downto 0) => processing_system7_0_M_AXI_GP0_BRESP(1 downto 0),
     M_AXI_GP0_BVALID => processing_system7_0_M_AXI_GP0_BVALID,
```

```
      M_AXI_GP0_RDATA(31 downto 0) => processing_system7_0_M_AXI_GP0_RDATA(31 downto 0),
      M_AXI_GP0_RID(11 downto 0) => processing_system7_0_M_AXI_GP0_RID(11 downto 0),
      M_AXI_GP0_RLAST => processing_system7_0_M_AXI_GP0_RLAST,
      M_AXI_GP0_RREADY => processing_system7_0_M_AXI_GP0_RREADY,
      M_AXI_GP0_RRESP(1 downto 0) => processing_system7_0_M_AXI_GP0_RRESP(1 downto 0),
      M_AXI_GP0_RVALID => processing_system7_0_M_AXI_GP0_RVALID,
      M_AXI_GP0_WDATA(31 downto 0) => processing_system7_0_M_AXI_GP0_WDATA(31 downto 0),
      M_AXI_GP0_WID(11 downto 0) => processing_system7_0_M_AXI_GP0_WID(11 downto 0),
      M_AXI_GP0_WLAST => processing_system7_0_M_AXI_GP0_WLAST,
      M_AXI_GP0_WREADY => processing_system7_0_M_AXI_GP0_WREADY,
      M_AXI_GP0_WSTRB(3 downto 0) => processing_system7_0_M_AXI_GP0_WSTRB(3 downto 0),
      M_AXI_GP0_WVALID => processing_system7_0_M_AXI_GP0_WVALID,
      PS_CLK => FIXED_IO_ps_clk,
      PS_PORB => FIXED_IO_ps_porb,
      PS_SRSTB => FIXED_IO_ps_srstb
    );
processing_system7_0_axi_periph: entity work.simple_axi_interface_processing_system7_0_axi_periph_0
    port map (
      ACLK => processing_system7_0_FCLK_CLK0,
      ARESETN(0) => rst_processing_system7_0_50M_interconnect_aresetn(0),
      M00_ACLK => processing_system7_0_FCLK_CLK0,
      M00_ARESETN(0) => rst_processing_system7_0_50M_peripheral_aresetn(0),
      M00_AXI_araddr(31 downto 0) => processing_system7_0_axi_periph_M00_AXI_ARADDR(31 downto 0),
      M00_AXI_arprot(2 downto 0) => processing_system7_0_axi_periph_M00_AXI_ARPROT(2 downto 0),
      M00_AXI_arready => processing_system7_0_axi_periph_M00_AXI_ARREADY,
      M00_AXI_arvalid => processing_system7_0_axi_periph_M00_AXI_ARVALID,
      M00_AXI_awaddr(31 downto 0) => processing_system7_0_axi_periph_M00_AXI_AWADDR(31 downto 0),
      M00_AXI_awprot(2 downto 0) => processing_system7_0_axi_periph_M00_AXI_AWPROT(2 downto 0),
      M00_AXI_awready => processing_system7_0_axi_periph_M00_AXI_AWREADY,
      M00_AXI_awvalid => processing_system7_0_axi_periph_M00_AXI_AWVALID,
      M00_AXI_bready => processing_system7_0_axi_periph_M00_AXI_BREADY,
      M00_AXI_bresp(1 downto 0) => processing_system7_0_axi_periph_M00_AXI_BRESP(1 downto 0),
      M00_AXI_bvalid => processing_system7_0_axi_periph_M00_AXI_BVALID,
      M00_AXI_rdata(31 downto 0) => processing_system7_0_axi_periph_M00_AXI_RDATA(31 downto 0),
      M00_AXI_rready => processing_system7_0_axi_periph_M00_AXI_RREADY,
      M00_AXI_rresp(1 downto 0) => processing_system7_0_axi_periph_M00_AXI_RRESP(1 downto 0),
      M00_AXI_rvalid => processing_system7_0_axi_periph_M00_AXI_RVALID,
      M00_AXI_wdata(31 downto 0) => processing_system7_0_axi_periph_M00_AXI_WDATA(31 downto 0),
      M00_AXI_wready => processing_system7_0_axi_periph_M00_AXI_WREADY,
      M00_AXI_wstrb(3 downto 0) => processing_system7_0_axi_periph_M00_AXI_WSTRB(3 downto 0),
      M00_AXI_wvalid => processing_system7_0_axi_periph_M00_AXI_WVALID,
      S00_ACLK => processing_system7_0_FCLK_CLK0,
      S00_ARESETN(0) => rst_processing_system7_0_50M_peripheral_aresetn(0),
      S00_AXI_araddr(31 downto 0) => processing_system7_0_M_AXI_GP0_ARADDR(31 downto 0),
      S00_AXI_arburst(1 downto 0) => processing_system7_0_M_AXI_GP0_ARBURST(1 downto 0),
      S00_AXI_arcache(3 downto 0) => processing_system7_0_M_AXI_GP0_ARCACHE(3 downto 0),
      S00_AXI_arid(11 downto 0) => processing_system7_0_M_AXI_GP0_ARID(11 downto 0),
      S00_AXI_arlen(3 downto 0) => processing_system7_0_M_AXI_GP0_ARLEN(3 downto 0),
      S00_AXI_arlock(1 downto 0) => processing_system7_0_M_AXI_GP0_ARLOCK(1 downto 0),
      S00_AXI_arprot(2 downto 0) => processing_system7_0_M_AXI_GP0_ARPROT(2 downto 0),
      S00_AXI_arqos(3 downto 0) => processing_system7_0_M_AXI_GP0_ARQOS(3 downto 0),
      S00_AXI_arready => processing_system7_0_M_AXI_GP0_ARREADY,
      S00_AXI_arsize(2 downto 0) => processing_system7_0_M_AXI_GP0_ARSIZE(2 downto 0),
      S00_AXI_arvalid => processing_system7_0_M_AXI_GP0_ARVALID,
      S00_AXI_awaddr(31 downto 0) => processing_system7_0_M_AXI_GP0_AWADDR(31 downto 0),
      S00_AXI_awburst(1 downto 0) => processing_system7_0_M_AXI_GP0_AWBURST(1 downto 0),
      S00_AXI_awcache(3 downto 0) => processing_system7_0_M_AXI_GP0_AWCACHE(3 downto 0),
      S00_AXI_awid(11 downto 0) => processing_system7_0_M_AXI_GP0_AWID(11 downto 0),
```

```
      S00_AXI_awlen(3 downto 0) => processing_system7_0_M_AXI_GP0_AWLEN(3 downto 0),
      S00_AXI_awlock(1 downto 0) => processing_system7_0_M_AXI_GP0_AWLOCK(1 downto 0),
      S00_AXI_awprot(2 downto 0) => processing_system7_0_M_AXI_GP0_AWPROT(2 downto 0),
      S00_AXI_awqos(3 downto 0) => processing_system7_0_M_AXI_GP0_AWQOS(3 downto 0),
      S00_AXI_awready => processing_system7_0_M_AXI_GP0_AWREADY,
      S00_AXI_awsize(2 downto 0) => processing_system7_0_M_AXI_GP0_AWSIZE(2 downto 0),
      S00_AXI_awvalid => processing_system7_0_M_AXI_GP0_AWVALID,
      S00_AXI_bid(11 downto 0) => processing_system7_0_M_AXI_GP0_BID(11 downto 0),
      S00_AXI_bready => processing_system7_0_M_AXI_GP0_BREADY,
      S00_AXI_bresp(1 downto 0) => processing_system7_0_M_AXI_GP0_BRESP(1 downto 0),
      S00_AXI_bvalid => processing_system7_0_M_AXI_GP0_BVALID,
      S00_AXI_rdata(31 downto 0) => processing_system7_0_M_AXI_GP0_RDATA(31 downto 0),
      S00_AXI_rid(11 downto 0) => processing_system7_0_M_AXI_GP0_RID(11 downto 0),
      S00_AXI_rlast => processing_system7_0_M_AXI_GP0_RLAST,
      S00_AXI_rready => processing_system7_0_M_AXI_GP0_RREADY,
      S00_AXI_rresp(1 downto 0) => processing_system7_0_M_AXI_GP0_RRESP(1 downto 0),
      S00_AXI_rvalid => processing_system7_0_M_AXI_GP0_RVALID,
      S00_AXI_wdata(31 downto 0) => processing_system7_0_M_AXI_GP0_WDATA(31 downto 0),
      S00_AXI_wid(11 downto 0) => processing_system7_0_M_AXI_GP0_WID(11 downto 0),
      S00_AXI_wlast => processing_system7_0_M_AXI_GP0_WLAST,
      S00_AXI_wready => processing_system7_0_M_AXI_GP0_WREADY,
      S00_AXI_wstrb(3 downto 0) => processing_system7_0_M_AXI_GP0_WSTRB(3 downto 0),
      S00_AXI_wvalid => processing_system7_0_M_AXI_GP0_WVALID
    );
rst_processing_system7_0_50M: component simple_axi_interface_rst_processing_system7_0_50M_0
    port map (
      aux_reset_in => '1',
      bus_struct_reset(0) => NLW_rst_processing_system7_0_50M_bus_struct_reset_UNCONNECTED(0),
      dcm_locked => '1',
      ext_reset_in => processing_system7_0_FCLK_RESET0_N,
      interconnect_aresetn(0) => rst_processing_system7_0_50M_interconnect_aresetn(0),
      mb_debug_sys_rst => '0',
      mb_reset => NLW_rst_processing_system7_0_50M_mb_reset_UNCONNECTED,
      peripheral_aresetn(0) => rst_processing_system7_0_50M_peripheral_aresetn(0),
      peripheral_reset(0) => NLW_rst_processing_system7_0_50M_peripheral_reset_UNCONNECTED(0),
      slowest_sync_clk => processing_system7_0_FCLK_CLK0
    );
simple_axi_interface_0: component simple_axi_interface_simple_axi_interface_0_0
    port map (
      s00_axi_aclk => processing_system7_0_FCLK_CLK0,
      s00_axi_araddr(3 downto 0) => processing_system7_0_axi_periph_M00_AXI_ARADDR(3 downto 0),
      s00_axi_aresetn => rst_processing_system7_0_50M_peripheral_aresetn(0),
      s00_axi_arprot(2 downto 0) => processing_system7_0_axi_periph_M00_AXI_ARPROT(2 downto 0),
      s00_axi_arready => processing_system7_0_axi_periph_M00_AXI_ARREADY,
      s00_axi_arvalid => processing_system7_0_axi_periph_M00_AXI_ARVALID,
      s00_axi_awaddr(3 downto 0) => processing_system7_0_axi_periph_M00_AXI_AWADDR(3 downto 0),
      s00_axi_awprot(2 downto 0) => processing_system7_0_axi_periph_M00_AXI_AWPROT(2 downto 0),
      s00_axi_awready => processing_system7_0_axi_periph_M00_AXI_AWREADY,
      s00_axi_awvalid => processing_system7_0_axi_periph_M00_AXI_AWVALID,
      s00_axi_bready => processing_system7_0_axi_periph_M00_AXI_BREADY,
      s00_axi_bresp(1 downto 0) => processing_system7_0_axi_periph_M00_AXI_BRESP(1 downto 0),
      s00_axi_bvalid => processing_system7_0_axi_periph_M00_AXI_BVALID,
      s00_axi_rdata(31 downto 0) => processing_system7_0_axi_periph_M00_AXI_RDATA(31 downto 0),
      s00_axi_rready => processing_system7_0_axi_periph_M00_AXI_RREADY,
      s00_axi_rresp(1 downto 0) => processing_system7_0_axi_periph_M00_AXI_RRESP(1 downto 0),
      s00_axi_rvalid => processing_system7_0_axi_periph_M00_AXI_RVALID,
      s00_axi_wdata(31 downto 0) => processing_system7_0_axi_periph_M00_AXI_WDATA(31 downto 0),
      s00_axi_wready => processing_system7_0_axi_periph_M00_AXI_WREADY,
```

```
    s00_axi_wstrb(3 downto 0) => processing_system7_0_axi_periph_M00_AXI_WSTRB(3 downto 0),
    s00_axi_wvalid => processing_system7_0_axi_periph_M00_AXI_WVALID
  );
end STRUCTURE;
```

Appendix 4: PID driver C code

```
/*****************************************************/
/*                                                   */
/*      Version 1.2                                  */
/*      Author: Bas Janssen                          */
/*      Lectoraat Robotics and High Tech Mechatronics */
/*      2016                                         */
/*                                                   */
/*****************************************************/

#include <linux/init.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/types.h>
#include <linux/kfifo.h>
#include <linux/ioport.h>
#include <linux/slab.h>
#include <linux/of.h>
#include <linux/of_device.h>
#include <linux/of_address.h>
#include <linux/platform_device.h>
#include <linux/list.h>
#include <asm/io.h>
#include <asm/uaccess.h>

MODULE_LICENSE("Dual BSD/GPL");
MODULE_AUTHOR("Bas Janssen");

#define CLASS_NAME "PID"
#define DEVICE_NAME "PID"

#define FPGA_SPACING 1

#define N_PID_MINORS 32

static struct class* PID_class = NULL;
static int PID_major = 0;

static DECLARE_BITMAP(minors, N_PID_MINORS);

static LIST_HEAD(device_list);
static DEFINE_MUTEX(device_list_lock);

//Make sure only one proccess can accessour the device
static DEFINE_MUTEX(PID_device_mutex);

static int memory_request = 0;

//Custom struct to store the data we want in the driver
struct PID_data {
        unsigned int * setpoint_address;
        unsigned int * position_address;
```

```
	unsigned int * P_address;
	unsigned int * I_address;
	unsigned int * D_address;
	unsigned int * state_address;
	unsigned int * update_address;
	unsigned int * emerg_address;
	int			message_read;
	unsigned int	base_register;
	struct list_head device_entry;
	dev_t			devt;
	};


static int PID_itoa(int value, char *buffer)
{
	char data[11];
	char temp_char;
	int i = 0;
	int j;
	int tmp;
	char int_array[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
	//As we dont have access to itoa(), we write it our selves. Convert the int to an array of chars, and flip it while
trimming leading 0's.
   tmp = value;
   for(i = 0; i<11; i++)
   {
			temp_char = tmp % 10;
      data[i] = '0' + temp_char;
      tmp   = tmp/10;
   }
	for(i = 0; i<11; i++)
	{
		if(data[9-i] != '0')
		{
			for(j = 0; j<(11-i); j++)
			{
				int_array[j] = data[9-i-j];
			}
			if(j<11)
			{
				int_array[j-1] = '\n';
				int_array[j] = '\0';
			}
			else
			{
				int_array[10] = '\n';
				int_array[11] = '\0';
			}
			break;
		}
	}
	strcpy(buffer, int_array);
	return 0;
}

/*-----------------------------------------------------------------------------------------------------------------*/
/*Device node handler functions and definition struct                                                              */
/*@PID_read the function to handle a read on the device node, returns the current position                         */
```

```
/*@PID_write the function the handle a write to the device node, sets the setpoint                    */
/*@PID_open handles the opening of the device node, gets the PID_data struct from memory and sets the device lock*/
/*@PID_release handles the closing of the device node and removes the device lock                    */
/*@PID_fops defines the handler functions for the operations                                          */
/*-----------------------------------------------------------------------------------------------------------------*/

static ssize_t PID_read(struct file* filp, char __user *buffer, size_t lenght, loff_t* offset)
{
	struct PID_data *PID;
	ssize_t retval;
	ssize_t copied = 0;
	unsigned int fpga_value;
	char int_array[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

	//Grab the PID_data struct out of the file struct.
	PID = filp->private_data;

	//cat keeps requesting new data until it receives a "return 0", so we do a one shot.
	if(PID->message_read)
		return 0;
	//Read from the I/O register
	fpga_value = ioread32(PID->position_address);

	PID_itoa(fpga_value, int_array);

	//Check how long the char array is after we build it.
	copied = sizeof(int_array);

	retval = copy_to_user(buffer, &int_array, copied);
	PID->message_read = 1;
	if(retval)
		return retval;
	return retval ? retval : copied;
}

static ssize_t PID_write(struct file* filp, const char __user *buffer, size_t lenght, loff_t* offset)
{
	struct PID_data *PID;
	ssize_t retval;
	unsigned int converted_value;
	ssize_t count = lenght;

	//Grab the PID_data struct out of the file struct.
	PID = filp->private_data;

	//Since the data we need is in userspace we need to copy it to kernel space so we can use it.
	retval = kstrtouint_from_user(buffer, count, 0, &converted_value);
	if(retval)
		return retval;
	//Write to the I/O register
	iowrite32(converted_value, PID->setpoint_address);
	return retval ? retval : count;
}

static int PID_open(struct inode* inode, struct file* filp)
{
	int status;
	struct PID_data *PID;
```

```
	mutex_lock(&device_list_lock);

	//Find the address of the struct using the device_list and the device_entry member of the PID_data struct.
	list_for_each_entry(PID, &device_list, device_entry) {
		//Check if the struct is the correct one.
		if(PID->devt == inode->i_rdev) {
			//Store the struct in the private_data member of the file struct so that it is usable in the read and
write functions of the device node.
			PID->message_read = 0;
			filp->private_data = PID;

			status = 0;
		}
	}

	//Try to lock the device, if it fails the device is already in use.
	if(!mutex_trylock(&PID_device_mutex))
	{
		printk(KERN_WARNING "Device is in use by another process\n");
		return -EBUSY;
	}

	mutex_unlock(&device_list_lock);

	return 0;
}

static int PID_release(struct inode* inode, struct file* filp)
{
	//Remove the mutex lock, so other processes can use the device.
	mutex_unlock(&PID_device_mutex);
//	printk(KERN_INFO "Unlocking mutex\n");
	return 0;
}

struct file_operations PID_fops = {
	.owner =	THIS_MODULE,
	.read =		PID_read,
	.write =	PID_write,
	.open =		PID_open,
	.release =	PID_release,
};

/*-----------------------------------------------------------------------------------------------------*/
/*Sysfs endpoint definitions and handler functions                                                     */
/*                                                                                                     */
/*                                                                                                     */
/*                                                                                                     */
/*                                                                                                     */
/*-----------------------------------------------------------------------------------------------------*/

static ssize_t sys_set_node(struct device* dev, struct device_attribute* attr, const char* buffer, size_t lenght)
{
	struct PID_data *PID;
	int retval;
	unsigned int converted_value;
	unsigned int * address;
	int count = lenght;
```

```
	//Find the address of the struct using the device_list and the device_entry member of the PID_data struct.
	list_for_each_entry(PID, &device_list, device_entry) {
		//Check if the struct is the correct one.
		if(PID->devt == dev->devt) {
			//Grab the address for the node that is being called.
			if(strcmp(attr->attr.name, "P") == 0)
				address = PID->P_address;
			else if(strcmp(attr->attr.name, "I") == 0)
				address = PID->I_address;
			else if(strcmp(attr->attr.name, "D") == 0)
				address = PID->D_address;
			else if(strcmp(attr->attr.name, "Update") == 0)
				address = PID->update_address;
		}
	}

	retval = kstrtoint(buffer, 0, &converted_value);
	if(retval)
		return retval;
	iowrite32(converted_value, address);
	return retval ? retval : count;
}

static ssize_t sys_read_node(struct device* dev, struct device_attribute* attr, char *buffer)
{
	struct PID_data *PID;
	int retval;
	int copied;
	//char data[11];
	//char temp_char;
	//int i = 0;
	//int j = 0;
	char int_array[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
	//int tmp;
	unsigned int * address;
	unsigned int fpga_value;

	//Find the address of the struct using the device_list and the device_entry member of the PID_data struct.
	list_for_each_entry(PID, &device_list, device_entry) {
		//Check if the struct is the correct one.
		if(PID->devt == dev->devt) {
			//Grab the address for the node that is being called.
			if(strcmp(attr->attr.name, "P") == 0)
				address = PID->P_address;
			else if(strcmp(attr->attr.name, "I") == 0)
				address = PID->I_address;
			else if(strcmp(attr->attr.name, "D") == 0)
				address = PID->D_address;
			else if(strcmp(attr->attr.name, "State") == 0)
				address = PID->state_address;
			else if(strcmp(attr->attr.name, "Emerg") == 0)
				address = PID->emerg_address;
		}
	}

	fpga_value = ioread32(address);
```

```
	PID_itoa(fpga_value, int_array);

	//Check how long the char array is after we build it.
	copied = sizeof(int_array);

	retval = copy_to_user(buffer, &int_array, copied);

	return retval ? retval : copied;
}

//Define the device attributes for the sysfs, and their handler functions.
static DEVICE_ATTR(P, S_IRUSR | S_IWUSR, sys_read_node, sys_set_node);
static DEVICE_ATTR(I, S_IRUSR | S_IWUSR, sys_read_node, sys_set_node);
static DEVICE_ATTR(D, S_IRUSR | S_IWUSR, sys_read_node, sys_set_node);
static DEVICE_ATTR(State, S_IRUSR, sys_read_node, NULL);
static DEVICE_ATTR(Update, S_IWUSR, NULL, sys_set_node);
static DEVICE_ATTR(Emerg, S_IRUSR, sys_read_node, NULL);

static struct attribute *PID_attrs[] = {
	&dev_attr_P.attr,
	&dev_attr_I.attr,
	&dev_attr_D.attr,
	&dev_attr_State.attr,
	&dev_attr_Update.attr,
	&dev_attr_Emerg.attr,
	NULL,
};

static struct attribute_group PID_attr_group = {
	.attrs = PID_attrs,
};

static const struct attribute_group* PID_attr_groups[] = {
	&PID_attr_group,
	NULL,
};

/*-----------------------------------------------------------------------------------------------------*/
/*Platform driver functions and struct                                                                 */
/*@PID_dt_ids[] struct to store the compatible device tree names                                       */
/*@PID_probe called when a compatible device is found in the device tree. Creates device and maps iomem      */
/*@PID_remove called when the driver is removed from the kernel, removes the device and unmaps iomem      */
/*@PID_driver struct to define the platform driver, contains the compatible ID's and the function names */
/*-----------------------------------------------------------------------------------------------------*/

static const struct of_device_id PID_dt_ids[] = {
	{ .compatible = "fontys,PID"},
	{},
};

MODULE_DEVICE_TABLE(of, PID_dt_ids);

static int PID_probe(struct platform_device *pltform_PID)
{
	int minor;
	int status;
	struct resource res;
	int rc;
```

```
	struct PID_data *PID;

	PID = kzalloc(sizeof(*PID), GFP_KERNEL);
	if(!PID)
		return -ENOMEM;

	INIT_LIST_HEAD(&PID->device_entry);

	mutex_lock(&device_list_lock);
	minor = find_first_zero_bit(minors, N_PID_MINORS);
	if (minor < N_PID_MINORS)
	{
		struct device *dev;

		PID->devt = MKDEV(PID_major, minor);
		dev = device_create_with_groups(PID_class, NULL, PID->devt, NULL, PID_attr_groups,
CLASS_NAME "%d", minor);
		status = PTR_ERR_OR_ZERO(dev);
	}
	else
	{
		printk(KERN_DEBUG "No minor number available!\n");
		status = -ENODEV;
	}
	if( status == 0)
	{
		printk(KERN_INFO "New PID controller PID%d\n", minor);
		set_bit(minor, minors);
		list_add(&PID->device_entry, &device_list);
		//Retreive the base address and request the memory region.
		rc = of_address_to_resource(pltform_PID->dev.of_node, 0, &res);
		if( request_mem_region(res.start, resource_size(&res), CLASS_NAME) == NULL)
		{
			printk(KERN_WARNING "Unable to obtain physical I/O addresses\n");
			goto failed_memregion;
		}
		PID->base_register = res.start;
		//Remap the memory region in to usable memory
		PID->setpoint_address =  of_iomap(pltform_PID->dev.of_node, 0);
		PID->P_address = PID->setpoint_address + 1;
		PID->I_address = PID->setpoint_address + 2;
		PID->D_address = PID->setpoint_address + 3;
		PID->update_address = PID->setpoint_address + 4;
		PID->state_address = PID->setpoint_address + 5;
		PID->emerg_address = PID->setpoint_address + 6;
		PID->position_address = PID->setpoint_address + 7;
	}
	mutex_unlock(&device_list_lock);

	if(status)
		kfree(PID);
	else
		platform_set_drvdata(pltform_PID, PID);

	return status;

	failed_memregion:
```

```
		device_destroy(PID_class, PID->devt);
		clear_bit(MINOR(PID->devt), minors);
	return -ENODEV;
}

static int PID_remove(struct platform_device *pltform_PID)
{
	struct PID_data *PID = platform_get_drvdata(pltform_PID);
	struct resource res;
	int rc;

	rc = of_address_to_resource(pltform_PID->dev.of_node, 0, &res);

	mutex_lock(&device_list_lock);
	//Unmap the iomem
	iounmap(PID->setpoint_address);
	//Delete the device from the list
	list_del(&PID->device_entry);
	//Destroy the device node
	device_destroy(PID_class, PID->devt);
	//Clear the minor bit
	clear_bit(MINOR(PID->devt), minors);
	if(PID->base_register)
	{
		release_mem_region(res.start, resource_size(&res));
		memory_request = 0;
	}
	//Free the kernel memory
	kfree(PID);
	mutex_unlock(&device_list_lock);

	return 0;
}

static struct platform_driver PID_driver = {
	.driver = {
		.name = "PID",
		.owner = THIS_MODULE,
		.of_match_table = of_match_ptr(PID_dt_ids),
	},
	.probe = PID_probe,
	.remove = PID_remove,
};

/*-----------------------------------------------------------------------------------------------------*/
/*Module level functions                                                                               */
/*@PID_init initialisation function for the driver                                                    */
/*@PID_exit exit function for the driver                                                              */
/*-----------------------------------------------------------------------------------------------------*/

static int PID_init(void)
{
	int retval;
	//Request a major device number from the kernel
	PID_major = register_chrdev(0, DEVICE_NAME, &PID_fops);
	//If the number is smaller than 0, it's an error. So jump to the error state.
	if(PID_major < 0)
	{
```

```
		printk(KERN_WARNING "hello: can't get major %d\n", PID_major);
		retval = PID_major;
		goto failed_chrdevreg;
	}
	//Now that we have a major number, create a device class.
	PID_class = class_create(THIS_MODULE, CLASS_NAME);
	//If there is an error, jump to the error state.
	if( IS_ERR(PID_class))
	{
		printk(KERN_NOTICE "Error while creating device class\n");
		retval =  PTR_ERR(PID_class);
		goto failed_classreg;
	}
	//Now that we have a class, we can create our device node.
	retval = platform_driver_register(&PID_driver);
	if(retval)
	{
		printk(KERN_NOTICE "Error while registering driver\n");
		goto failed_driverreg;
	}

	//Initialize the mutex lock.
	mutex_init(&PID_device_mutex);

	return 0;

	failed_driverreg:
		class_destroy(PID_class);
	failed_classreg:
		unregister_chrdev(PID_major, DEVICE_NAME);
	failed_chrdevreg:
		return -1;
}

static void PID_exit(void)
{
	//When we remove the driver, destroy it's device(s), class and major number.
	//device_destroy(PID_class, MKDEV(PID_major, 0));
	platform_driver_unregister(&PID_driver);
	class_destroy(PID_class);
	unregister_chrdev(PID_major, DEVICE_NAME);
}

module_init(PID_init);
module_exit(PID_exit);
```

Appendix 5: PID driver test plan

Onderdeel	Test	Testmethode	Verwachte resultaat	Resultaat
Inladen driver	Wordt er een major number aangemaakt?	Manueel driver inladen, dmesg output bekijken	Geen melding in dmesg	pass
	Wordt er een class aangemaakt?	Kijk of /sys/class/PID aangemaakt is	Geen melding in dmesg, /sys/class/PID bestaat	pass
	Is de driver geregistreerd?	Controleer de dmesg output.	Geen melding in dmesg, /sys/class/PID bestaat	pass
Device nodes	Wordt er een device node aangemaakt?	List de inhoudt van /dev	Een of meerdere bestanden in de vorm van PID... bestaan in /dev	pass
	Zijn er sysfs endpoints aangemaakt?	Kijk of er onder /sys/class/PID... bestanden bestaan	De bestanden P, I, D, State, Update en Emerg bestaan	pass
Lezen schrijven device node	Is de device node beschrijfbaar?	Gebruik het commando "echo ... > /dev/PID..."	Geen foutmeldingen	pass
	Is de device node leesbaar?	Gebruik het commando "cat /dev/PID..."	Geen foutmeldingen	pass
Lezen schrijven sysfs endpoints	Is het endpoint P te beschrijven?	Gebruik het commando "echo ... > /sys/class/PID/PID.../P"	Geen foutmeldingen	pass
	Is het endpoint P te lezen?	Gebruik het commando "cat /sys/class/PID/PID.../P"	De waarde die in de schrijftest is ingevoerd	pass
	Is het endpoint I te beschrijven?	Gebruik het commando "echo ... > /sys/class/PID/PID.../I"	Geen foutmeldingen	pass
	Is het endpoint I te lezen?	Gebruik het commando "cat /sys/class/PID/PID.../I"	De waarde die in de schrijftest is ingevoerd	pass
	Is het endpoint D te beschrijven?	Gebruik het commando "echo ... > /sys/class/PID/PID.../D"	Geen foutmeldingen	pass
	Is het endpoint D te lezen?	Gebruik het commando "cat /sys/class/PID/PID.../D"	De waarde die in de schrijftest is ingevoerd	pass
	Is het endpoint Update te beschrijven?	Gebruik het commando "echo ... > /sys/class/PID/PID.../Update"	Geen foutmeldingen	pass
	Is het endpoint State te lezen?	Gebruik het commando "cat /sys/class/PID/PID.../State"	Geen foutmeldingen	pass
	Is het endpoint Emerg te lezen?	Gebruik het commando "cat /sys/class/PID/PID.../Emerg"	Geen foutmeldingen	pass
Uitladen driver	Worden de device nodes verwijderd?	Manueel driver uitladen, inhoud van /dev bekijken	Geen /dev/PID... nodes meer	pass
	Zijn de sysfs endpoints verwijderd?	Inhoud van /sys/class/PID bekijken	Geen map /sys/class/PID meer omdat de driver uitgeladen is	pass
	Is de class vernietigd?	Inhoud van /sys/class bekijken	Geen map /sys/class/PID meer omdat de driver uitgeladen is	pass
Eindresultaat	Zijn alle testen met een goed resultaat afgesloten?	Controleer de resultaten van de voorgaande testen.	Alle testen zijn succesvol voltooid.	pass

Appendix 6: PWM driver C code

```
/*****************************************************/
/*                                                   */
/*      Version 1.0                                  */
/*      Author: Bas Janssen                          */
/*      Lectoraat Robotics and High Tech Mechatronics */
/*      2016                                         */
/*                                                   */
/*****************************************************/

#include <linux/init.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/types.h>
#include <linux/kfifo.h>
#include <linux/ioport.h>
#include <linux/slab.h>
#include <linux/of.h>
#include <linux/of_device.h>
#include <linux/of_address.h>
#include <linux/platform_device.h>
#include <linux/list.h>
#include <asm/io.h>
#include <asm/uaccess.h>

MODULE_LICENSE("Dual BSD/GPL");
MODULE_AUTHOR("Bas Janssen");

#define CLASS_NAME "PWM"
#define DEVICE_NAME "PWM"

#define FPGA_SPACING 1

#define N_PWM_MINORS 32

static struct class* PWM_class = NULL;
static int PWM_major = 0;

static DECLARE_BITMAP(minors, N_PWM_MINORS);

static LIST_HEAD(device_list);
static DEFINE_MUTEX(device_list_lock);

//Make sure only one proccess can accessour the device
static DEFINE_MUTEX(PWM_device_mutex);

static int memory_request = 0;

//Custom struct to store the data we want in the driver
struct PWM_data {
        unsigned int * pulsewidth_address;
        int             message_read;
```

```
	unsigned int	base_register;
	struct list_head device_entry;
	dev_t		devt;
	};


static int PWM_itoa(int value, char *buffer)
{
	char data[11];
	char temp_char;
	int i = 0;
	int j;
	int tmp;
	char int_array[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
	//As we dont have access to itoa(), we write it our selves. Convert the int to an array of chars, and flip it while
trimming leading 0's.
   tmp = value;
   for(i = 0; i<11; i++)
   {
			temp_char = tmp % 10;
      data[i] = '0' + temp_char;
      tmp   = tmp/10;
   }
	for(i = 0; i<11; i++)
	{
		if(data[9-i] != '0')
		{
			for(j = 0; j<(11-i); j++)
			{
				int_array[j] = data[9-i-j];
			}
			if(j<11)
			{
				int_array[j-1] = '\n';
				int_array[j] = '\0';
			}
			else
			{
				int_array[10] = '\n';
				int_array[11] = '\0';
			}
			break;
		}
	}
	strcpy(buffer, int_array);
	return 0;
}

/*-----------------------------------------------------------------------------------------------------------------*/
/*Device node handler functions and definition struct                                                              */
/*@PWM_read the function to handle a read on the device node, returns the current position                        */
/*@PWM_write the function the handle a write to the device node, sets the setpoint                                */
/*@PWM_open handles the opening of the device node, gets the PWM_data struct from memory and sets the device
lock	*/
/*@PWM_release handles the closing of the device node and removes the device lock                                 */
/*@PWM_fops defines the handler functions for the operations                                                      */
/*-----------------------------------------------------------------------------------------------------------------*/
```

```
static ssize_t PWM_read(struct file* filp, char __user *buffer, size_t lenght, loff_t* offset)
{
	struct PWM_data *PWM;
	ssize_t retval;
	ssize_t copied = 0;
	unsigned int fpga_value;
	char int_array[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

	//Grab the PWM_data struct out of the file struct.
	PWM = filp->private_data;

	//cat keeps requesting new data until it receives a "return 0", so we do a one shot.
	if(PWM->message_read)
		return 0;
	//Read from the I/O register
	fpga_value = ioread32(PWM->pulsewidth_address);

	PWM_itoa(fpga_value, int_array);

	//Check how long the char array is after we build it.
	copied = sizeof(int_array);

	retval = copy_to_user(buffer, &int_array, copied);
	PWM->message_read = 1;
	if(retval)
		return retval;
	return retval ? retval : copied;
}

static ssize_t PWM_write(struct file* filp, const char __user *buffer, size_t lenght, loff_t* offset)
{
	struct PWM_data *PWM;
	ssize_t retval;
	unsigned int converted_value;
	ssize_t count = lenght;

	//Grab the PWM_data struct out of the file struct.
	PWM = filp->private_data;

	//Since the data we need is in userspace we need to copy it to kernel space so we can use it.
	retval = kstrtouint_from_user(buffer, count, 0, &converted_value);
	if(retval > 8192)
		retval = 8192;
	if(retval)
		return retval;
	//Write to the I/O register
	iowrite32(converted_value, PWM->pulsewidth_address);
	return retval ? retval : count;
}

static int PWM_open(struct inode* inode, struct file* filp)
{
	int status;
	struct PWM_data *PWM;
	mutex_lock(&device_list_lock);

	//Find the address of the struct using the device_list and the device_entry member of the PWM_data struct.
	list_for_each_entry(PWM, &device_list, device_entry) {
```

```
		//Check if the struct is the correct one.
		if(PWM->devt == inode->i_rdev) {
			//Store the struct in the private_data member of the file struct so that it is usable in the read and
write functions of the device node.
			PWM->message_read = 0;
			filp->private_data = PWM;

			status = 0;
		}
	}

	//Try to lock the device, if it fails the device is already in use.
	if(!mutex_trylock(&PWM_device_mutex))
	{
		printk(KERN_WARNING "Device is in use by another process\n");
		return -EBUSY;
	}

	mutex_unlock(&device_list_lock);

	return 0;
}

static int PWM_release(struct inode* inode, struct file* filp)
{
	//Remove the mutex lock, so other processes can use the device.
	mutex_unlock(&PWM_device_mutex);
//	printk(KERN_INFO "Unlocking mutex\n");
	return 0;
}

struct file_operations PWM_fops = {
	.owner =	THIS_MODULE,
	.read =		PWM_read,
	.write =	PWM_write,
	.open =		PWM_open,
	.release =	PWM_release,
};

/*-----------------------------------------------------------------------------------------------------------*/
/*Platform driver functions and struct                                                                       */
/*@PWM_dt_ids[] struct to store the compatible device tree names                                             */
/*@PWM_probe called when a compatible device is found in the device tree. Creates device and maps iomem   */
/*@PWM_remove called when the driver is removed from the kernel, removes the device and unmaps iomem   */
/*@PWM_driver struct to define the platform driver, contains the compatible ID's and the function names */
/*-----------------------------------------------------------------------------------------------------------*/

static const struct of_device_id PWM_dt_ids[] = {
	{ .compatible = "fontys,PWM"},
	{},
};

MODULE_DEVICE_TABLE(of, PWM_dt_ids);

static int PWM_probe(struct platform_device *pltform_PWM)
{
	int minor;
	int status;
```

```
	struct resource res;
	int rc;

	struct PWM_data *PWM;

	PWM = kzalloc(sizeof(*PWM), GFP_KERNEL);
	if(!PWM)
		return -ENOMEM;

	INIT_LIST_HEAD(&PWM->device_entry);

	mutex_lock(&device_list_lock);
	minor = find_first_zero_bit(minors, N_PWM_MINORS);
	if (minor < N_PWM_MINORS)
	{
		struct device *dev;

		PWM->devt = MKDEV(PWM_major, minor);
		dev = device_create_with_groups(PWM_class, NULL, PWM->devt, NULL, NULL, CLASS_NAME
"%d", minor);
		status = PTR_ERR_OR_ZERO(dev);
	}
	else
	{
		printk(KERN_DEBUG "No minor number available!\n");
		status = -ENODEV;
	}
	if( status == 0)
	{
		printk(KERN_INFO "New PWM controller PWM%d\n", minor);
		set_bit(minor, minors);
		list_add(&PWM->device_entry, &device_list);
		//Retreive the base address and request the memory region.
		rc = of_address_to_resource(pltform_PWM->dev.of_node, 0, &res);
		if( request_mem_region(res.start, resource_size(&res), CLASS_NAME) == NULL)
		{
			printk(KERN_WARNING "Unable to obtain physical I/O addresses\n");
			goto failed_memregion;
		}
		PWM->base_register = res.start;
		//Remap the memory region in to usable memory
		PWM->pulsewidth_address =  of_iomap(pltform_PWM->dev.of_node, 0);
	}
	mutex_unlock(&device_list_lock);

	if(status)
		kfree(PWM);
	else
		platform_set_drvdata(pltform_PWM, PWM);

	return status;

	failed_memregion:
		device_destroy(PWM_class, PWM->devt);
		clear_bit(MINOR(PWM->devt), minors);
	return -ENODEV;
}
```

```
static int PWM_remove(struct platform_device *pltform_PWM)
{
	struct PWM_data *PWM = platform_get_drvdata(pltform_PWM);
	struct resource res;
	int rc;

	rc = of_address_to_resource(pltform_PWM->dev.of_node, 0, &res);

	mutex_lock(&device_list_lock);
	//Unmap the iomem
	iounmap(PWM->pulsewidth_address);
	//Delete the device from the list
	list_del(&PWM->device_entry);
	//Destroy the device node
	device_destroy(PWM_class, PWM->devt);
	//Clear the minor bit
	clear_bit(MINOR(PWM->devt), minors);
	if(PWM->base_register)
	{
		release_mem_region(res.start, resource_size(&res));
		memory_request = 0;
	}
	//Free the kernel memory
	kfree(PWM);
	mutex_unlock(&device_list_lock);

	return 0;
}

static struct platform_driver PWM_driver = {
	.driver = {
		.name = "PWM",
		.owner = THIS_MODULE,
		.of_match_table = of_match_ptr(PWM_dt_ids),
	},
	.probe = PWM_probe,
	.remove = PWM_remove,
};

/*-----------------------------------------------------------------------------------------------------------*/
/*Module level functions                                                                                     */
/*@PWM_init initialisation function for the driver                                                           */
/*@PWM_exit exit function for the driver                                                                     */
/*-----------------------------------------------------------------------------------------------------------*/

static int PWM_init(void)
{
	int retval;
	//Request a major device number from the kernel
	PWM_major = register_chrdev(0, DEVICE_NAME, &PWM_fops);
	//If the number is smaller than 0, it's an error. So jump to the error state.
	if(PWM_major < 0)
	{
		printk(KERN_WARNING "hello: can't get major %d\n", PWM_major);
		retval = PWM_major;
		goto failed_chrdevreg;
	}
	//Now that we have a major number, create a device class.
```

```
	PWM_class = class_create(THIS_MODULE, CLASS_NAME);
	//If there is an error, jump to the error state.
	if( IS_ERR(PWM_class))
	{
		printk(KERN_NOTICE "Error while creating device class\n");
		retval =  PTR_ERR(PWM_class);
		goto failed_classreg;
	}
	//Now that we have a class, we can create our device node.
	retval = platform_driver_register(&PWM_driver);
	if(retval)
	{
		printk(KERN_NOTICE "Error while registering driver\n");
		goto failed_driverreg;
	}

	//Initialize the mutex lock.
	mutex_init(&PWM_device_mutex);

	return 0;

	failed_driverreg:
		class_destroy(PWM_class);
	failed_classreg:
		unregister_chrdev(PWM_major, DEVICE_NAME);
	failed_chrdevreg:
		return -1;
}

static void PWM_exit(void)
{
	//When we remove the driver, destroy it's device(s), class and major number.
	//device_destroy(PWM_class, MKDEV(PWM_major, 0));
	platform_driver_unregister(&PWM_driver);
	class_destroy(PWM_class);
	unregister_chrdev(PWM_major, DEVICE_NAME);
}

module_init(PWM_init);
module_exit(PWM_exit);
```

Appendix 7: PWM driver test plan

Onderdeel	Test	Testmethode	Verwachte resultaat	Resultaat
Inladen driver	Wordt er een major number aangemaakt?	Manueel driver inladen, dmesg output bekijken	Geen melding in dmesg	pass
	Wordt er een class aangemaakt?	Kijk of /sys/class/PWM aangemaakt is	Geen melding in dmesg, /sys/class/PWM bestaat	pass
	Is de driver geregistreerd?	Controleer de dmesg output.	Geen melding in dmesg, /sys/class/PWM bestaat	pass
Device nodes	Wordt er een device node aangemaakt?	List de inhoudt van /dev	Een of meerdere bestanden in de vorm van PWM... bestaan in /dev	pass
Lezen device node	Is de device node leesbaar?	Gebruik het commando "cat /dev/PWM..."	De ingestelde puls breedte	pass
Schrijven device node	Is de device node schijfbaar?	Gebruik het commando "echo ... > /dev/PWM..."	Geen foutmelding	pass
Uitladen dirver	Worden de device nodes verwijderd?	Manueel driver uitladen, inhoud van /dev bekijken	Geen /dev/PWM... nodes meer	pass
	Is de class vernietigd?	Inhoud van /sys/class bekijken	Geen map /sys/class/PWM meer omdat de driver uitgeladen is	pass
Eindresultaat	Zijn alle testen met een goed resultaat afgesloten?	Controleer de resultaten van de voorgaande testen.	Alle testen zijn succesvol voltooid.	pass

Appendix 8: Encoder driver C code

```
/*****************************************************/
/*                                                   */
/*      Version 1.0                                  */
/*      Author: Bas Janssen                          */
/*      Lectoraat Robotics and High Tech Mechatronics */
/*      2016                                         */
/*                                                   */
/*****************************************************/

#include <linux/init.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/types.h>
#include <linux/kfifo.h>
#include <linux/ioport.h>
#include <linux/slab.h>
#include <linux/of.h>
#include <linux/of_device.h>
#include <linux/of_address.h>
#include <linux/platform_device.h>
#include <linux/list.h>
#include <asm/io.h>
#include <asm/uaccess.h>

MODULE_LICENSE("Dual BSD/GPL");
MODULE_AUTHOR("Bas Janssen");

#define CLASS_NAME "Encoder"
#define DEVICE_NAME "Encoder"

#define FPGA_SPACING 1

#define N_Encoder_MINORS 32

static struct class* Encoder_class = NULL;
static int Encoder_major = 0;

static DECLARE_BITMAP(minors, N_Encoder_MINORS);

static LIST_HEAD(device_list);
static DEFINE_MUTEX(device_list_lock);

//Make sure only one proccess can accessour the device
static DEFINE_MUTEX(Encoder_device_mutex);

static int memory_request = 0;

//Custom struct to store the data we want in the driver
struct Encoder_data {
        unsigned int * position_address;
        unsigned int * direction_address;
```

```
        unsigned int   base_register;
        int                    message_read;
        struct list_head device_entry;
        dev_t                  devt;
        };


static int Encoder_itoa(int value, char *buffer)
{
        char data[11];
        char temp_char;
        int i = 0;
        int j;
        int tmp;
        char int_array[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        //As we dont have access to itoa(), we write it our selves. Convert the int to an array of chars, and flip it while
trimming leading 0's.
   tmp = value;
   for(i = 0; i<11; i++)
   {
                temp_char = tmp % 10;
      data[i] = '0' + temp_char;
      tmp   = tmp/10;
   }
        for(i = 0; i<11; i++)
        {
                if(data[9-i] != '0')
                {
                        for(j = 0; j<(11-i); j++)
                        {
                                int_array[j] = data[9-i-j];
                        }
                        if(j<11)
                        {
                                int_array[j-1] = '\n';
                                int_array[j] = '\0';
                        }
                        else
                        {
                                int_array[10] = '\n';
                                int_array[11] = '\0';
                        }
                        break;
                }
        }
        strcpy(buffer, int_array);
        return 0;
}

/*-----------------------------------------------------------------------------------------------------------------*/
/*Device node handler functions and definition struct                                                              */
/*@Encoder_read the function to handle a read on the device node, returns the current position                    */
/*@Encoder_write the function the handle a write to the device node, sets the setpoint                             */
/*@Encoder_open handles the opening of the device node, gets the Encoder_data struct from memory and sets the
device lock       */
/*@Encoder_release handles the closing of the device node and removes the device lock                              */
/*@Encoder_fops defines the handler functions for the operations                                                   */
/*-----------------------------------------------------------------------------------------------------------------*/
```

```
static ssize_t Encoder_read(struct file* filp, char __user *buffer, size_t lenght, loff_t* offset)
{
	struct Encoder_data *Encoder;
	ssize_t retval;
	ssize_t copied = 0;
	unsigned int fpga_value;
	char int_array[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

	//Grab the Encoder_data struct out of the file struct.
	Encoder = filp->private_data;

	//cat keeps requesting new data until it receives a "return 0", so we do a one shot.
	if(Encoder->message_read)
		return 0;
	//Read from the I/O register
	fpga_value = ioread32(Encoder->position_address);

	Encoder_itoa(fpga_value, int_array);

	//Check how long the char array is after we build it.
	copied = sizeof(int_array);

	retval = copy_to_user(buffer, &int_array, copied);
	Encoder->message_read = 1;
	if(retval)
		return retval;
	return retval ? retval : copied;
}

static ssize_t Encoder_write(struct file* filp, const char __user *buffer, size_t lenght, loff_t* offset)
{
//	@TODO: Write is not possible
/*	struct Encoder_data *Encoder;
	ssize_t retval;
	unsigned int converted_value;
	ssize_t count = lenght;

	//Grab the Encoder_data struct out of the file struct.
	Encoder = filp->private_data;

	//Since the data we need is in userspace we need to copy it to kernel space so we can use it.
	retval = kstrtouint_from_user(buffer, count, 0, &converted_value);
	if(retval)
		return retval;
	//Write to the I/O register
	iowrite32(converted_value, Encoder->setpoint_address);
	return retval ? retval : count;*/

	return -EPERM;
}

static int Encoder_open(struct inode* inode, struct file* filp)
{
	int status;
	struct Encoder_data *Encoder;
	mutex_lock(&device_list_lock);
```

```
	//Find the address of the struct using the device_list and the device_entry member of the Encoder_data struct.
	list_for_each_entry(Encoder, &device_list, device_entry) {
		//Check if the struct is the correct one.
		if(Encoder->devt == inode->i_rdev) {
			//Store the struct in the private_data member of the file struct so that it is usable in the read and
write functions of the device node.
			Encoder->message_read = 0;
			filp->private_data = Encoder;

			status = 0;
		}
	}

	//Try to lock the device, if it fails the device is already in use.
	if(!mutex_trylock(&Encoder_device_mutex))
	{
		printk(KERN_WARNING "Device is in use by another process\n");
		return -EBUSY;
	}

	mutex_unlock(&device_list_lock);

	return 0;
}

static int Encoder_release(struct inode* inode, struct file* filp)
{
	//Remove the mutex lock, so other processes can use the device.
	mutex_unlock(&Encoder_device_mutex);
//	printk(KERN_INFO "Unlocking mutex\n");
	return 0;
}

struct file_operations Encoder_fops = {
	.owner =	THIS_MODULE,
	.read =		Encoder_read,
	.write =	Encoder_write,
	.open =		Encoder_open,
	.release =	Encoder_release,
};

/*-----------------------------------------------------------------------------------------------------*/
/*Sysfs endpoint definitions and handler functions                                                     */
/*                                                                                                     */
/*                                                                                                     */
/*                                                                                                     */
/*                                                                                                     */
/*-----------------------------------------------------------------------------------------------------*/

/*
static ssize_t sys_set_node(struct device* dev, struct device_attribute* attr, const char* buffer, size_t lenght)
{
	struct Encoder_data *Encoder;
	int retval;
	unsigned int converted_value;
	unsigned int * address;
	int count = lenght;
```

```
	//Find the address of the struct using the device_list and the device_entry member of the Encoder_data struct.
	list_for_each_entry(Encoder, &device_list, device_entry) {
		//Check if the struct is the correct one.
		if(Encoder->devt == dev->devt) {
			//Grab the address for the node that is being called.
			if(strcmp(attr->attr.name, "P") == 0)
				address = Encoder->P_address;
			else if(strcmp(attr->attr.name, "I") == 0)
				address = Encoder->I_address;
			else if(strcmp(attr->attr.name, "D") == 0)
				address = Encoder->D_address;
			else if(strcmp(attr->attr.name, "Update") == 0)
				address = Encoder->update_address;
		}
	}

	retval = kstrtoint(buffer, 0, &converted_value);
	if(retval)
		return retval;
	iowrite32(converted_value, address);
	return retval ? retval : count;
}
*/

static ssize_t sys_read_node(struct device* dev, struct device_attribute* attr, char *buffer)
{
	struct Encoder_data *Encoder;
	int retval;
	int copied;
	//char data[11];
	//char temp_char;
	//int i = 0;
	//int j = 0;
	char int_array[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
	//int tmp;
	unsigned int * address;
	unsigned int fpga_value;

	//Find the address of the struct using the device_list and the device_entry member of the Encoder_data struct.
	list_for_each_entry(Encoder, &device_list, device_entry) {
		//Check if the struct is the correct one.
		if(Encoder->devt == dev->devt) {
/*			//Grab the address for the node that is being called.
			if(strcmp(attr->attr.name, "P") == 0)
				address = Encoder->P_address;
			else if(strcmp(attr->attr.name, "I") == 0)
				address = Encoder->I_address;
			else if(strcmp(attr->attr.name, "D") == 0)
				address = Encoder->D_address;
			else if(strcmp(attr->attr.name, "State") == 0)
				address = Encoder->state_address;
			else if(strcmp(attr->attr.name, "Emerg") == 0)
				address = Encoder->emerg_address;*/
			address = Encoder->direction_address;
		}
	}

	fpga_value = ioread32(address);
```

```
	Encoder_itoa(fpga_value, int_array);

	//Check how long the char array is after we build it.
	copied = sizeof(int_array);

	retval = copy_to_user(buffer, &int_array, copied);

	return retval ? retval : copied;
}

//Define the device attributes for the sysfs, and their handler functions.
static DEVICE_ATTR(Direction, S_IRUSR, sys_read_node, NULL);

static struct attribute *Encoder_attrs[] = {
	&dev_attr_Direction.attr,
	NULL,
};

static struct attribute_group Encoder_attr_group = {
	.attrs = Encoder_attrs,
};

static const struct attribute_group* Encoder_attr_groups[] = {
	&Encoder_attr_group,
	NULL,
};

/*-----------------------------------------------------------------------------------------------------------*/
/*Platform driver functions and struct                                                                       */
/*@Encoder_dt_ids[] struct to store the compatible device tree names                                         */
/*@Encoder_probe called when a compatible device is found in the device tree. Creates device and maps iomem*/
/*@Encoder_remove called when the driver is removed from the kernel, removes the device and unmaps iomem*/
/*@Encoder_driver struct to define the platform driver, contains the compatible ID's and the function names */
/*-----------------------------------------------------------------------------------------------------------*/

static const struct of_device_id Encoder_dt_ids[] = {
	{ .compatible = "fontys,Encoder"},
	{ .compatible = "xlnx,IP-Enc-Struct-1.8"},
	{},
};

MODULE_DEVICE_TABLE(of, Encoder_dt_ids);

static int Encoder_probe(struct platform_device *pltform_Encoder)
{
	int minor;
	int status;
	struct resource res;
	int rc;

	struct Encoder_data *Encoder;

	Encoder = kzalloc(sizeof(*Encoder), GFP_KERNEL);
	if(!Encoder)
		return -ENOMEM;

	INIT_LIST_HEAD(&Encoder->device_entry);
```

```
	mutex_lock(&device_list_lock);
	minor = find_first_zero_bit(minors, N_Encoder_MINORS);
	if (minor < N_Encoder_MINORS)
	{
		struct device *dev;

		Encoder->devt = MKDEV(Encoder_major, minor);
		dev = device_create_with_groups(Encoder_class, NULL, Encoder->devt, NULL, Encoder_attr_groups,
CLASS_NAME "%d", minor);
		status = PTR_ERR_OR_ZERO(dev);
	}
	else
	{
		printk(KERN_DEBUG "No minor number available!\n");
		status = -ENODEV;
	}
	if( status == 0)
	{
		printk(KERN_INFO "New Encoder controller Encoder%d\n", minor);
		set_bit(minor, minors);
		list_add(&Encoder->device_entry, &device_list);
		//Retreive the base address and request the memory region.
		rc = of_address_to_resource(pltform_Encoder->dev.of_node, 0, &res);
		if( request_mem_region(res.start, resource_size(&res), CLASS_NAME) == NULL)
		{
			printk(KERN_WARNING "Unable to obtain physical I/O addresses\n");
			goto failed_memregion;
		}
		Encoder->base_register = res.start;
		//Remap the memory region in to usable memory
		Encoder->position_address =  of_iomap(pltform_Encoder->dev.of_node, 0);
		Encoder->direction_address = Encoder->position_address + 1;
	}
	mutex_unlock(&device_list_lock);

	if(status)
		kfree(Encoder);
	else
		platform_set_drvdata(pltform_Encoder, Encoder);

	return status;

	failed_memregion:
		device_destroy(Encoder_class, Encoder->devt);
		clear_bit(MINOR(Encoder->devt), minors);
	return -ENODEV;
}

static int Encoder_remove(struct platform_device *pltform_Encoder)
{
	struct Encoder_data *Encoder = platform_get_drvdata(pltform_Encoder);
	struct resource res;
	int rc;

	rc = of_address_to_resource(pltform_Encoder->dev.of_node, 0, &res);

	mutex_lock(&device_list_lock);
```

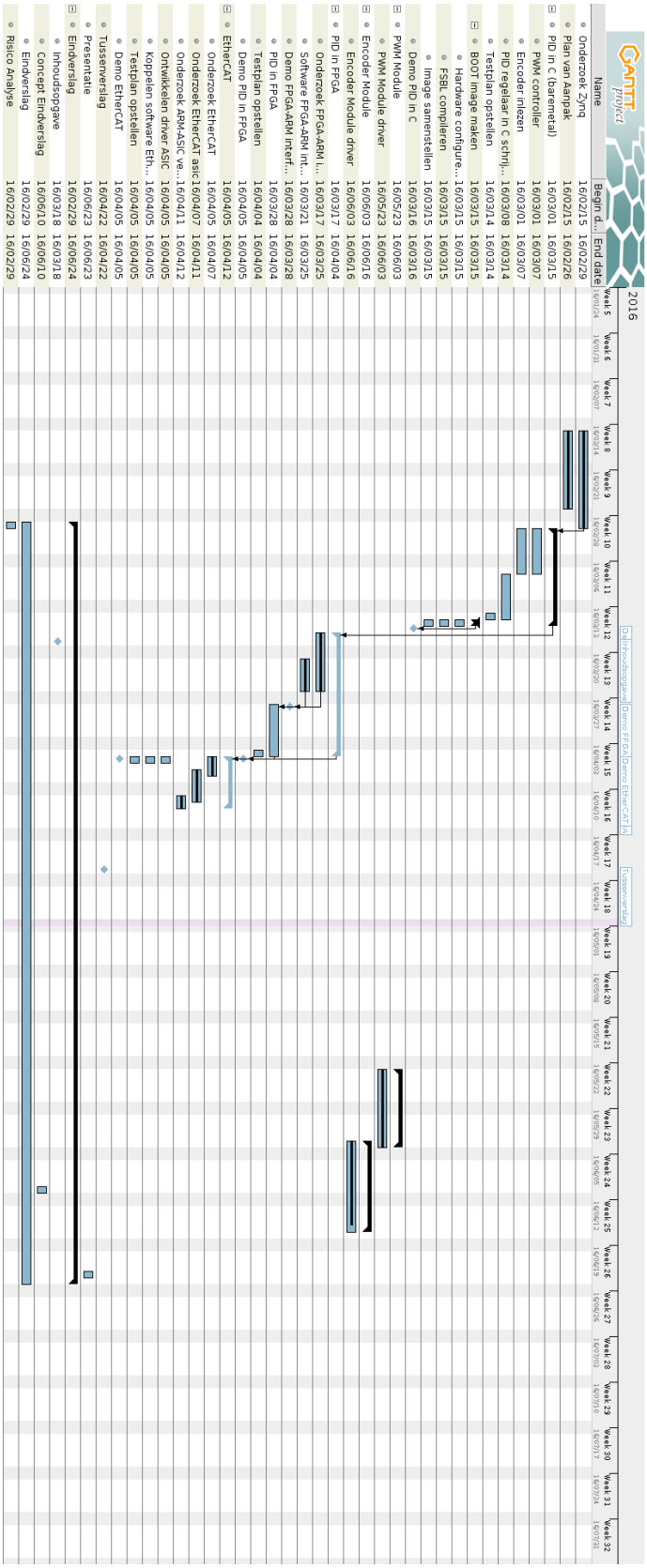
```
	//Unmap the iomem
	iounmap(Encoder->position_address);
	//Delete the device from the list
	list_del(&Encoder->device_entry);
	//Destroy the device node
	device_destroy(Encoder_class, Encoder->devt);
	//Clear the minor bit
	clear_bit(MINOR(Encoder->devt), minors);
	if(Encoder->base_register)
	{
		release_mem_region(res.start, resource_size(&res));
		memory_request = 0;
	}
	//Free the kernel memory
	kfree(Encoder);
	mutex_unlock(&device_list_lock);

	return 0;
}

static struct platform_driver Encoder_driver = {
	.driver = {
		.name = "Encoder",
		.owner = THIS_MODULE,
		.of_match_table = of_match_ptr(Encoder_dt_ids),
	},
	.probe = Encoder_probe,
	.remove = Encoder_remove,
};

/*-----------------------------------------------------------------------------------------------------------*/
/*Module level functions                                                                                     */
/*@Encoder_init initialisation function for the driver                                                       */
/*@Encoder_exit exit function for the driver                                                                 */
/*-----------------------------------------------------------------------------------------------------------*/

static int Encoder_init(void)
{
	int retval;
	//Request a major device number from the kernel
	Encoder_major = register_chrdev(0, DEVICE_NAME, &Encoder_fops);
	//If the number is smaller than 0, it's an error. So jump to the error state.
	if(Encoder_major < 0)
	{
		printk(KERN_WARNING "hello: can't get major %d\n", Encoder_major);
		retval = Encoder_major;
		goto failed_chrdevreg;
	}
	//Now that we have a major number, create a device class.
	Encoder_class = class_create(THIS_MODULE, CLASS_NAME);
	//If there is an error, jump to the error state.
	if( IS_ERR(Encoder_class))
	{
		printk(KERN_NOTICE "Error while creating device class\n");
		retval =  PTR_ERR(Encoder_class);
		goto failed_classreg;
	}
	//Now that we have a class, we can create our device node.
```

```
	retval = platform_driver_register(&Encoder_driver);
	if(retval)
	{
		printk(KERN_NOTICE "Error while registering driver\n");
		goto failed_driverreg;
	}

	//Initialize the mutex lock.
	mutex_init(&Encoder_device_mutex);

	return 0;

	failed_driverreg:
		class_destroy(Encoder_class);
	failed_classreg:
		unregister_chrdev(Encoder_major, DEVICE_NAME);
	failed_chrdevreg:
		return -1;
}

static void Encoder_exit(void)
{
	//When we remove the driver, destroy it's device(s), class and major number.
	//device_destroy(Encoder_class, MKDEV(Encoder_major, 0));
	platform_driver_unregister(&Encoder_driver);
	class_destroy(Encoder_class);
	unregister_chrdev(Encoder_major, DEVICE_NAME);
}

module_init(Encoder_init);
module_exit(Encoder_exit);
```

Appendix 9: Encoder driver test plan

Onderdeel	Test	Testmethode	Verwachte resultaat	Resultaat
Inladen driver	Wordt er een major number aangemaakt?	Manueel driver inladen, dmesg output bekijken	Geen melding in dmesg	pass
	Wordt er een class aangemaakt?	Kijk of /sys/class/Encoder aangemaakt is	Geen melding in dmesg. /sys/class/Encoder bestaat	pass
	Is de driver geregistreerd?	Controleer de dmesg output.	Geen melding in dmesg. /sys/class/Encoder bestaat	pass
Device nodes	Wordt er een device node aangemaakt?	List de inhoudt van /dev	Een of meerdere bestanden in de vorm van Encoder... bestaan in /dev	pass
	Zijn er sysfs endpoints aangemaakt?	Kijk of er onder /sys/class/Encoder... bestanden bestaan	Het bestand Direction bestaat	pass
Lezen device node	Is de device node leesbaar?	Gebruik het commando "cat /dev/Encoder..."	Een positie waarde	pass
Lezen sysfs endpoint	Is het endpoint Direction te lezen?	Gebruik het commando "cat /sys/class/Encoder/Encoder../Direction"	Een richtings waarde 1 of 0	pass
Uitladen dirver	Worden de device nodes verwijderd?	Manueel driver uitladen, inhoud van /dev bekijken	Geen /dev/Encoder... nodes meer	pass
	Zijn de sysfs endpoints verwijderd?	Inhoud van /sys/class/Encoder bekijken	Geen map /sys/class/Encoder meer omdat de driver uitgeladen is	pass
	Is de class vernietigd?	Inhoud van /sys/class bekijken	Geen map /sys/class/Encoder meer omdat de driver uitgeladen is	pass
Eindresultaat	Zijn alle testen met een goed resultaat afgesloten?	Controleer de resultaten van de voorgaande testen.	Alle testen zijn succesvol voltooid.	pass

Appendix 10: Planning



Appendix 11: Digital files