CTU

CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Department of Cybernetics

Bachelor's thesis

FPGA-based Control of Multi-Legged Walking Robot

Miroslav Tržil

May 2019

Supervisor: Doc. Ing. Jan Faigl, Ph.D.

Supervisor specialist: Ing. Petr Čížek

Declaration

I declare that the presented work was developed independently ant that I have listed all sources of the information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 24, 2019

. . . . . .

Miroslav Tržil

Acknowledgement

I would like to express thanks to my supervisor doc. Ing, Jan Faigl. Ph.D. and my supervisor specialist Ing. Petr Čížek for their guidance, valuable advice and patience with me. I am profoundly grateful to all academical workers, who have given me all important and useful knowledge. I would like to thank my loved ones for their support through my whole studies.

Abstrakt

Tato bakalářská práce se věnuje návrhu paralelní architektury pro hardwarovou akceleraci řízení pohybu šestinohého kráčejícího robotu. Akcelerace je dosaženo paralelizací komunikačního rozhraní k jednotlivým servomotorům Dynamixel AX-12A, která jsou v základním návrhu robotu spojena v jednom komunikačním řetězci, pomocí paralelní architektury hradlového pole (FPGA). Hlavním benefitem použití FPGA architektury je, že umožňuje vysokorychlostní, synchronní, paralelní řízení jednotlivých nohou robotu. Při použití navržené architektury bylo docíleno zrychlení komunikace 13x oproti standardnímu řízení přes ovládací PC. Práce také předkládá návrh desky plošného spoje, která zprostředkovává rozhraní mezi servomotory a vývojovou deskou FPGA. Navržená deska navíc integruje rozhraní k dalším senzorům, ochranné prvky zamezující podbití napájecí Li-poly baterie robotu pod bezpečnou úroveň a zdroj napájení pro použitou elektroniku.

Klíčová slova: FPGA, Hexapod, UART, Dynamixel AX-12A

Abstract

In this thesis, a Field-Programmable Gate Array (FPGA) architecture for hardware accelerated control of a multi-legged walking robot is presented. The core of the presented work is the development of the FPGA architecture to parallelize and speed-up the communication with the robot's intelligent Dynamiel AX12-A servomotors. The presented architecture enables synchronous, truly parallel, highspeed control of the individual legs of the robot. The experimental results show, that the proposed architecture achieves 13 times speedup of the communication in comparison to the PC implementation of the robot controller. Further, a custom designed printed circuit board (PCB) that interfaces the FPGA development board to the robot is presented. Further, the custom designed PCB allows interfacing of other sensors, it integrates a power converters for the FPGA board, and a Li-poly battery watchdog to cut of the electric current to prevent depleting, and thus damaging, of the batteries.

Keywords: FPGA, Hexapod, UART, Dynamixel AX-12A

Contents

List of Figures

List of Tables

Abbreviations

ADC	Analog-to-digital converter
AvalonMM interface	Avalon memory mapped interface
CMOS	Complementary metal–oxide–semiconductor
ComRob	Computational Robotics
DC	Direct current
DLL	Delay-locked loop
DoF	Degree of freedom
EPROM	Erasable programmable read-only memory
FIFO	First in first out
FPGA	Field programmable gate array
FTP	Force threshold-based position
GPIO	General-purpose input/output
HPS	Hard Processor system
IC	Integrated circuit
ID	Identifier
I/O	Input / output
JTAG	Join Test Action Group
LAB	Logic array block
LED	Light emitting diode
Li-poly	Lithium polymer
LUT	Look-up table
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
PC	Personal computer
PCB	Printed circuit board
PLL	Phase-locked loops
RAM	Random-access-memory
SD	Secure digital
SRAM	Static random access memory
SPI	Serial peripheral interface
SPS	Soft processor systems
TTL	Transistor transistor logic
UART	Universal asynchronous receiver - transmitter
USB	Universal Serial Bus
V	Voltage
VHSIC	Very high speed integrated circuit
VHDL	VHSIC hardware description language

Chapter 1
Introduction

Modern society develops robots to make the life more comfortable. The robots are replacing humans in physically demanding and dangerous jobs. In industry, the robots become essential parts of the process and nowadays, the mobile robots are gaining on importance. Those robots can be used in many applications such as search and rescue, or data-collection missions, exploration of unknown environments, or even in military applications.

Many mobile robots exist nowadays. Those robots are designed to move in various environments (such as water, air, and the surface). This thesis is focused on terrestrial moving robots. Those robots can be divided into groups by way of their moving to wheel robots, robots with tracks, and walking robots. The walking robots have an advantage in rough terrains, where they are able to move without any destruction of the environment.

Hexapod robot platforms are used in the Computational Robotics Laboratory [1] for research projects such as exploration [1] or terrain characterization [2]. The currently used hexapods have three degrees of freedom (DoF) per leg. However, Martin Zoula presents a hexapod platform with 4 DoF per leg in his thesis [3]. Each degree of freedom is enabled by one servomotor, which means that the currently used robot consists of eighteen servomotors (Dynamixel AX-12A). All those actuators onboard of the currently used robots have been connected by the daisy chain, i. e., the servomotors have to be read sequentially. All the servomotors communicate via Half-duplex Universal asynchronous receiver-transmitter (UART), the protocol si further described in section 2.1.

To move the robot the Gait (described in Section 2.1) have to be applied. In ComRob Laboratory adaptive gate was developed [4] and the communication speed is critical for this gate. Therefore, to speed it up, we propose a parallel architecture to be used for the communication with the servomotors. In such a way, each servo can be connected directly to the control unit. The disadvantage of it is an amount of wires going along the leg, increasing the probability of defect issue on them. Thus, the whole leg is connected in a daisy chain and the legs are connected to the control unit (legs are parallel) in this thesis.

When the robot was controlled by a single bus, the connection with the onboard computer was easy. However, when six buses have to be controlled the additional device is needed. It can be implemented by the microcontroller (e.g., Teensy 3.6 or STM32) which enables the use of six UART interfaces or by the Field programmable gate array (FPGA). FPGA allows the programmer to describe the exact hardware requirements, which are then implemented by connecting individual logical elements. FPGA perfectly fits the problems which are easy to parallelise or pipelined. The FPGA is preferred in this thesis because it enables creating the packet simultaneously and it allows for easy synchronization of the legs, which is favourable for the locomotion control.

Other renowned institutions uses FPGA to control their robots proving that this is a good way. First example is MIT Cheetah [5], having 4kHz control loop. Another examples are [6], [7] and [8]. The usage of FPGA has some disadvantages as well. The main one is difficult and time-consuming programming. However, there is an opportunity to combine the power of FPGA with the speediness of programming standard CPUs by using the system on chip (SoC) or soft-core processors. Then, methodologies for the design of the SoC architectures [9] can be applied to speed up the development process significantly.

[1] part of Ai Center, Faculty of Electronic Engineering, Czech Technical University in Prague

In this thesis, an FPGA architecture is presented that reports more than thirteen times bigger speed of communication with using FPGA in comparison to the laptop with the USB converter.

The thesis starts with the description of the robot (Section 2.1), where all important information about the hexapod itself, gate and used actuators are briefly described and provide background information for understanding the importance of the communication speed. The basic principles of FPGA technology are described in the next Section, 2.2. This Section also contains information about the FPGA development board which is planned to be used on the robot. To use the FPGA on a hexapod, the prototype board has to be designed. This board is described in Chapter 3. In Chapter 3.3 the software solution is presented. It consists of the FPGA architecture, and the CPU program that enables to test the architecture. The results of the experimental validation of the architecture are reported in Chapter 5. Concluding remarks are dedicated to Chapter 6.

Chapter 2
Problem Statement

This thesis is focused on the implementation of the FPGA architecture for hardware accelerated control of a multi-legged walking robot, that is achieved by parallelization of the communication with the individual servomotors of the robot. The FPGA communication module is used on board of hexapod robot, and provides the connection to servomotors. The locomotion controller (described in the next section) needs to write and read data to a servomotor with a high frequency. Before this thesis, all servos have been connected using a single bus. by using the FPGA it is possible to communicate with each leg separately. The locomotion controller [10] sets positions of selected servos and then immediately reads their actual positions. For the correct movement of the robot, the time-determinism is critical.

The main hypothesis is, that the parallel approach should make the communication with the servos at least six time faster as we are communicating with each leg on a dedicated line, than when all servos are connected on a single daisy chain and are read sequentially.

2.1 Hexapod Description

As was written in the introduction, hexapods are used in the Computational Robotics Laboratory as multi-legged robots. Up to this time the PhantomX AX Mark II is used(see Figure 1). However, two new robots are currently developed [3] [11]. The PhantomX is an open source robotic platform, produced by Interbotix. As you can see in Figure 1, each PhantomX leg is an open chain with 3 Degrees of Freedom. Each joint is active and controlled by the Dynamixel AX-12A servomotor. The names of the joints are Coxa, Femur and Tibia. The Coxa is connected to the torso and Tibia is a part of the foot tip. The names of the individual joints are inspired by the insect.

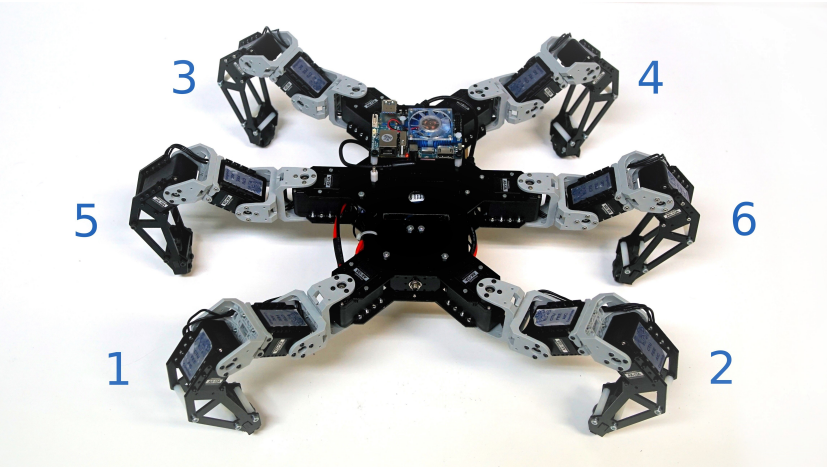

Figure 1: PhantomX AX Mark II.

Hexapod can locomote using different Gaits [12]. The first and the fastest is the tripod gait. Assume that the legs are numbered, as shown in Figure 1. The Tripod Gait starts moving legs 3,6 and 1 together, while legs 4, 5 and 2 are supporting the robot. In the next step, the supporting and swinging legs are swapped. Another gate is Quadruped. Two legs are in swing phase and four in the stance phase in this gate. If the robot travels in rough terrain the pentapod gait can be used. Despite

being slow, the robot using pentapod gait is very stable. Only one leg is in swing phase at a time, the other five are supporting the robot.

When the robot is moving on a flat surface the legs can be controlled in open-loop. However, in the terrain, the leg has to stop when hitting the ground. If the leg does not stop and continue exerting the force, the torso of the robot might be pulled up and other legs might lose their support. On the other hand, when the leg does not reach the surface, the robot might fall in the next swing phase.

The ground detection can be approached in different ways, in ComRob laboratory the force threshold-based position (FTP) controller is used [4] [10] . The ground is detected by measuring an increasing the torque in a servo motor. Unfortunately, the Dynamixel AX-12A cannot measure the torque directly. It uses a P-type controller, where the error angle is considered as proportional to the torque which, unfortunately, holds only when the servo is not moving. Therefore, the locomotion control is based on monitoring of the position error of the servomotor. The main idea is to stop the leg movement when the difference between the set and read position of the servo is above a predefined threshold value. Therefore, the swing-down phase is interpolated by small steps to decrease the position error when the leg starts its movement. These small steps reduce the ground reaction force as well.

Dynamixel AX-12A

This Section is based on [13]. Dynamixel AX-12A, the consumer smart servo motor is used as an actuator. Despite its compact size, it can produce high torque, which is produced by DC motor with a gear reducer. The servo enables feedback control for the angular position, angular velocity and torque. This servo motor has its operation range limited to $300°$, as shown in Figure 2. Position can be controlled with a resolution of 1024 steps, which means that the angle can be read with the resolution of $0.29°$.

As you can see in Figure 3 this servo motor has two Molex3P connectors. All three pins in one connector are directly connected to the second connector. Thus the AX-12A can be operated with only one connector attached or can be easily used in a daisy chain.

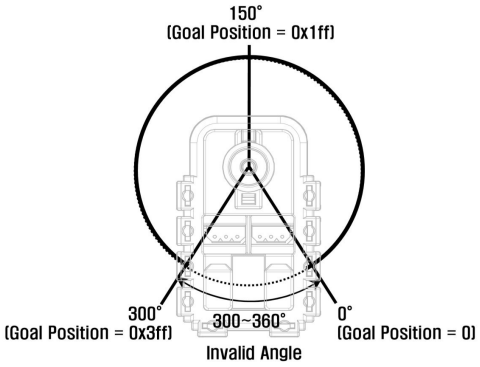

Figure 2: AX-12A reachable position. Courtesy of [13]

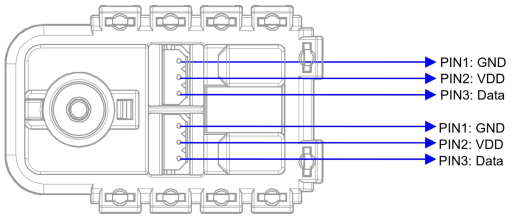

Figure 3: AX-12A pin assigment. Courtesy of [13]

The power supply is connected to the second pin. The input voltage should be between 9 to 12 Volts, which perfectly fits a 3-cell Li-poly battery.

The servo can be controlled via the pin3. It supports Transistor-transistor logic (TTL) logic which is a two-state logic, that is typically used by the integrated circuits built from the bipolar transistors. The logic zero (LOW) is represented by 0 to 0.8 V. The 2.7 to 5 V will be read as the logic one (HIGH). Between those two values is an invalid metastable state.

The selected properties of AX-12A are summarized in Table 1.

Table 1: Selected properties of AX-12A.

Weight:	54.6 g
Dimension:	32 mm · 50 mm · 40 mm
Resolution:	0.29°
Gear reduction ratio:	254 : 1
Stall Torque:	1.5 Nm (at 12.0 V, 1.5 A)
No load speed:	59 rpm(at 12.0 V)
Running temperature:	-5 – 70° C
Voltage:	9 – 12 V

The servos communicate via the half-duplex UART[2] interface. The half-duplex UART is a serial communication which connects the transmitter and the receiver using a single wire. In the case of AX-12A, UART is used as a multi-drop bus. If a node is not transmitting, it has high impedance on its port. Pull up resistor is connected to the bus. Therefore 5 V can be measured on the bus when all devices have their pins in high impedance. When the device transmits logical zero, it switches the pin to the ground and on the bus 0 V can be measured. When the node transmits logical one, its pin will be in the high impedance. Despite, collision can't cause a short-circuit, it can be recognized (node which transmits logical 1 will read logical 0).

The speed of the communication can be changed in range 9,600 to 1 M bauds. Baud is a unit, for symbol rate per second. In this case, when there are only two symbols, the baud rate is the same as bit rate. The AX-12A is set to 1 M bauds by default. The aim is to have communication as fast as possible, so this is a good choice. The lower speed is needed when using long wires to detect collisions.

To ensure working communication, even the last bit has to be sampled before the transmission of the bit ends. Therefore the timing mistake should not be larger than one half of single bit duration. The possible error, which will not affect the communication, is calculated in equation $\frac{100}{bit\ quantity \cdot 4} = 2.5\,\%$. The previous equation does not consider the transient response. The bit quantity is multiplied by four because both receiver and transmitter have this tolerance.

In this section communication protocol is described. The controller communicates with the Dynamixel servo motors by sending them packets (formatted block of data) that comply to the Dynamixel Protocol 1.0 [13]. Two packet types can be distinguished. First one sends the controller to the servomotor. This packet is called an instruction packet and consists of:

- **Header 1** has value 0XFF.
- **Header 2** is the same as header 1.
- **ID** unequivocally determines each servo. The value of servo ID should be in the interval 0 to 253. When the ID is 254, the packet is considered as a broadcast [3].
- **Length** of the packet is needed because it is not fixed. This number should be computed as an amount of parameters + 2. For example, the shortest packet has no parameters, the length is 2, and this packet contains six bytes.
- An **instruction** contains the command for the servomotor. The possible Values are listed in Table 2.
- **Parameters** are used when instruction requires additional data.

[2] universal asynchronous receiver-transmitter
[3] All servomotors reads broadcast packets.

- A **checksum** is used to detect any damage of the packet. If the sum of all bytes is divisible by 255, the packet is considered as correct.

Table 2: List of packet instructions.

Value	Instruction	Function	Number of parameters
0x01	Ping	Used for obtaining a status packet	0
0x02	Read	Reading data from the servomotor	2
0x03	Write	Writing values to the servomotor	2 +
0x04	Reg Write	Similar to Write, but it is executed through Action instruction	2 +
0x05	Action	Triggers the previous instruction	0
0x06	Factory reset	Set the control table of the servomotor to the default value settings	0
0x83	Sync write	Write data on the same address at once	4 +

The servo motor replies by sending a Status Packet, which consists of:

- **Header 1** has value 0XFF.
- **Header 2** is the same as header 1.
- The **ID** of the servo which sends the status packet.
- **Length**.
- The **Error** byte displays the error status sent from the Dynamixel. Each bit represents specific problem as shown in Table 3.
- Optional **parameters**.
- **Checksum**.

The proposed architecture is supposed to implement the protocol to relieve the computational burden from the CPU and allow synchronous sending and reading of the packet communication content.

Table 3: Meaning of each bit in error byte.

Bit	Name	Description
Bit 0	Input voltage error	Set to 1 if the voltage is out of the operating voltage range as defined in the control table
Bit 1	Angle limit error	Set to 1 if the goal position is out of the range of the angle limits.
Bit 2	Overheating error	Set to 1 if the inside temperature of Dynamixel higher then value in the control table.
Bit 3	Range error	Set to 1 if the instruction is out of range for the use.
Bit 4	Checksum error	Set to 1 if the checksum of the instruction packet is incorrect.
Bit 5	Overload error	Set to 1 if the set torque is low to control the current load.
Bit 6	Instruction error	Set to 1 if the instruction is undefined or need more parameters.
Bit 7		Unused

2.2 Field Programmable Gate Array

The Field Programmable Gate Arrays (FPGAs) are digital integrated circuits (ICs). The "Field Programmable" signifies that the customer can configure the design after manufacturing. The "Gate" stands for a logic gate, which implements some Boolean function. Finally, the word "Arrays" indicate the large quantity of the gates.

FPGA was invented in the mid-1980's by Xilinx [14]. Nowadays, two companies are leading the production of FPGAs. First one, is Xilinx, the inventor and the second one is Intel, who bought this division from Aletra. In this thesis, Intel FPGA is used.

Each manufacturer, and even every device family from one manufacturer uses different technologies. In this section, the common principle is described.

The FPGA consists of logic blocks, routing and others resources (see Figure 8). This section is based on [15], [16] and [17].

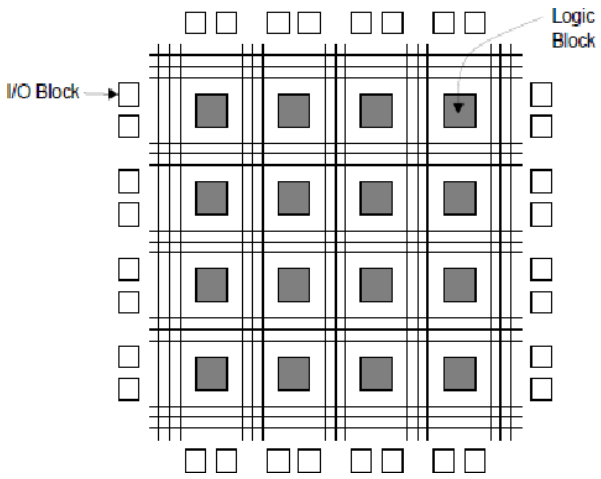

Figure 4: FPGA structure. Courtesy of [18].

Logic element

In this thesis, the Intel (Altera) terminology is used, because their FPGA board is applied. Intel call the smallest logic block as a logic element, other manufacturers may call them differently (for example logic cell from Xilinx [9]). A typical logic element (as shown in Figure 5) is composed of N-input Look-up table(LUT), multiplexor and flip-flop. The LUT is a memory, which represents a constant Boolean function of N variables. The LUT can be used as a memory element, as well. Multiplexor switches the LUT's output to a flip flop or directly to the output from the logic element. The Flip-flop sample and hold the value until the next clock signal is activated. The LUTs are connected to bigger parts, which Intel call logic array block (LAB).

Routing

The LABs are connected to make complex architecture via switches and wires. The wires are both horizontal and vertical and create an interconnected matrix. The two most used technology of the programmable switches are described below.

The majority of FPGAs are based on SRAM (Static Random Access Memory) technology, which is based on the flip-flops and can be implemented on CMOS as the rest of the FPGA parts. The SRAM is volatile [4], thus they have to be reconfigured each time the system is powered up. Consequently,

[4] Volatile memories drop their information after power lost.

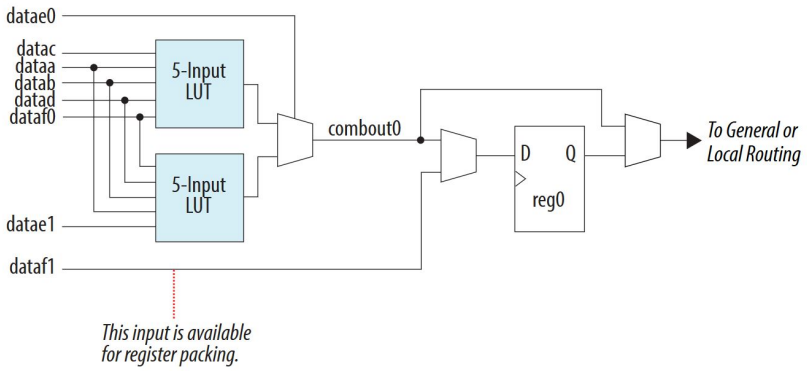

Figure 5: LUT used in Cyclon V. Courtesy of [19].

external memories (such as EPROM) is needed. However, external memory causes problems with the protection of the intellectual property of the design.

In contrast, with EPROM, antifuse technologies retains all date even without power. However, it can be programmed only once. When the chip is manufactured all the links are disconnected, and during programming, high voltage makes them conductive. The advantages of antifuse are immunity to radiation and reverse-engineering safety.

Other Resources

The FPGA contains additional components such as embedded processors or dedicated blocks for communication support. Further, clock resources are described in the next paragraph.

The clock source can be both external or from an internal oscillator. Then FPGA can change the clock frequency by PLL and DLL. The clock is then distributed to the desired elements via a clock tree. One FPGA contains more clock trees. Therefore more clock domains can be used in the design.

Other useful resources are memories. Memories can be implemented by the LAB or by the dedicated memory blocks. Memory blocks can be used in many applications such as cache for an embedded processor, FIFO, single or dual port RAM.

Input/output (I/O) blocks are used to interface the FPGA to the peripheral components. The signals can be registered or unregistered. The pins can be configured as input, output, or as bidirectional. The voltage level should be set typically from 1.2 V to 3.3 V.

As the FPGAs are very versatile, they can implement even a full scale processors in the FPGA fabric. Such processors are called soft processor systems (SPS). On the other hand, FPGA fabric can be directly interfaced with an actual processor creating a hard processor system (HPS). The main advantage of the SPS system is, that the instruction set of the processor can be easily appended with new, accelerated instructions, at the cost the maximum clock frequency of such processor is limited by the FPGA fabric to approximately hundreds of MHz. The HPS systems are full scale processors that can have frequency up to units of GHz.

Utilized Field Programmable Gate Array

The DE10-Nano development board from Intel is used in this thesis (see Figure 6). This board combines the dual-core ARM Cortex-A9 embedded processor (HPS) (its features are listed in Table 5) with the Cyclone V SE programmable logic (its features are listed in Table 4). The DE10-Nano is powered by 5 V.

The FPGA can be configured by Embedded USB-Blaster II (JTAG) cable (external JTAG is possible but not necessary) by active serial programming, which is non-volatile. In addition, the HPS can

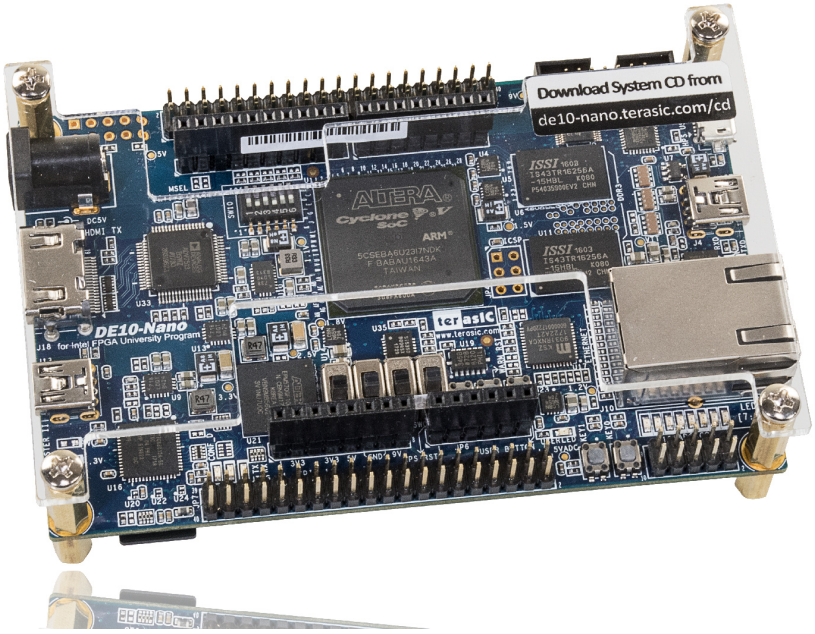

Figure 6: Used FPGA device, DE10-Nano. Courtesy of www.terasic.com (cited on 19.5.2019).

configure the FPGA and it can even reconfigure parts of the FPGA during the runtime.

The DE10-Nano is equipped with LTC2308, 12-bit analog-to-digital convertor (ADC). The ADC has eight channels, which are connected to the 2x5 header and six of them are shared with the Arduino Analog input. Measured voltage has to be in range 0 – 4.096 V. The ADC is connected to the FPGA fabric via SPI.

Table 4: Features of the FPGA chip.

FPGA chip	Cyclone V	5CSEBA6U23I7
	Number of logic elements:	110 k
	Memory	5,570 kilobits
	Number of PLLs	6
	Number of user defined I/Os	145
GPIO	Number of push buttons	2
	Number of slide switches	4
	Number of LEDs	8
Expansion	two 40 pin headers	72 GPIO
	Arduino header	16 GPIO
Other	A/D convertor	LTC2308
	HDMI TX interface	ADV7513

Table 5: Features of the Hard Processor system.

Processor	Dual-core ARM	Cortex - A9
	L1 instruction cache	32 kB
	L1 data cache	32 kB
	L2 shared cache	512 kB
Memory	On-chip SRAM	64 kB
	DDR3 SDRAM	1 GB
	micro SD	8 GB
Others	Gigabit Ethernet	KSZ9031RN
	USB On-The-Go	2.0
	Accelerometer	ADXL345
	User button	1
	User LED	1
	Linear Technology connector	14 pin

Chapter 3
Hardware Design

Aa prototype printed circuit board (PCB) has to be designed to enable the FPGA to control the robot. The main reason is, that the servomotors communicate using the 5V logic, while the FPGA operates on 3.3V logic, and therefore direct interfacing would damage the FPGA board. In addition to the interfacing, the PCB provides power supply for electronics which is used on the robot, level shifting, and some other features. The PCB design is shown in Figure 7.

In Figure 8 the block diagram of the PCB board is shown. The red colour connects the power supply by the voltage of the battery. The orange colour represents the 5V power supply. The digital data is drawn in the blue colour, and the analogue signals are green. The switches in the ellipse represent a MOS transistor, controlled by the blue signal. The individual modules are described in the following sections.

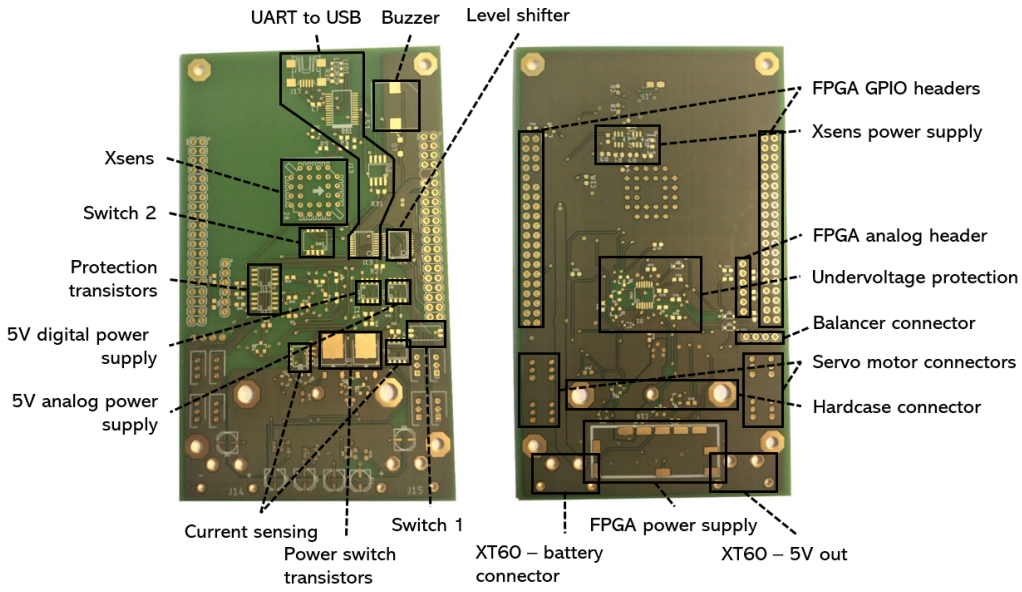

Figure 7: Prototype board.

3.1 Battery Protection

The entire robot is powered by three-cell lithium polymer battery (Li-poly) battery. The maximum voltage of a single cell is 4.2 V when fully charged. The discharged cut-off voltage is 3 V. When the voltage drops below this level, the battery may be irreversibly damaged. Before this happens the power has to be switched off. In this work two transistors, P-channel power MOSFET is used (in Figure 7 marked as power switch transistors). Each transistor controls one of the two power supply branches. One powers the servomotors, and the second one is supplying all other electronics. Those transistors can be turned on only if the voltage has the correct polarity.

The secondary reverse voltage protection is mechanically assured by the used connectors, which

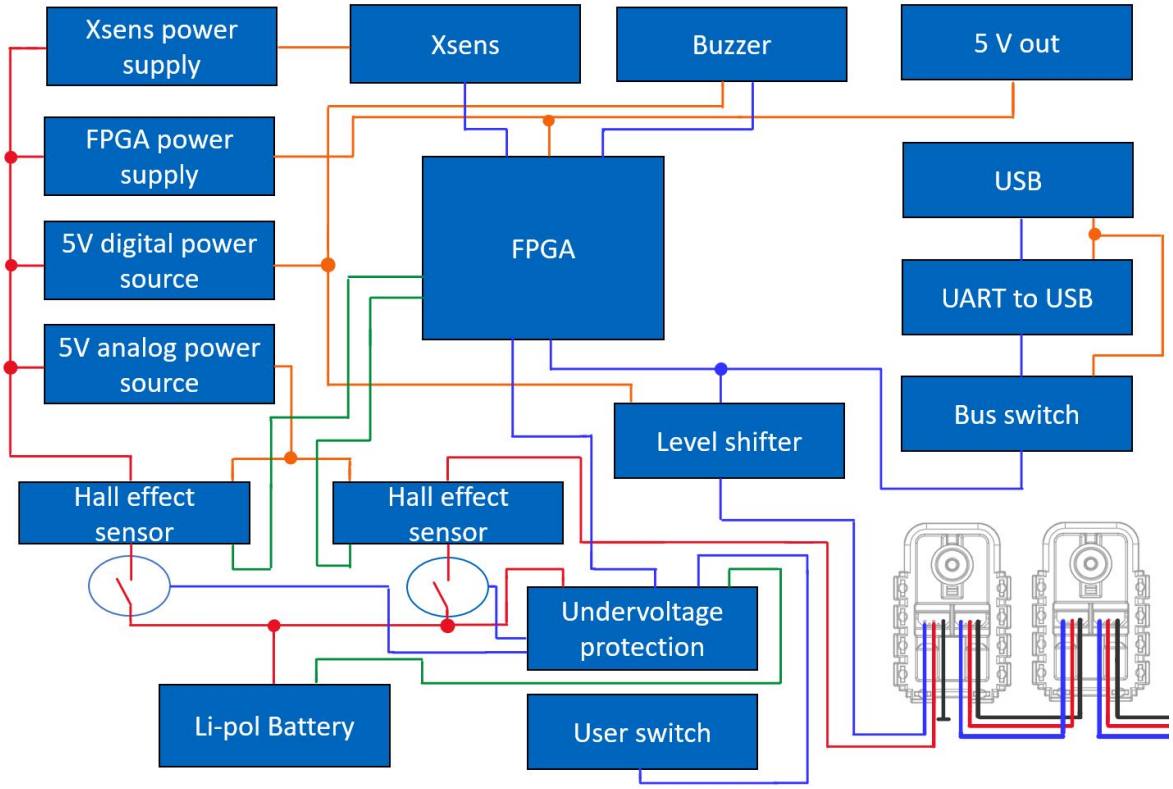

Figure 8: Block diagram of the prototype board.

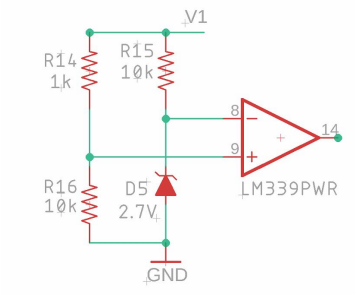

Figure 9: Principle of the under-voltage protection consists of a comparator.

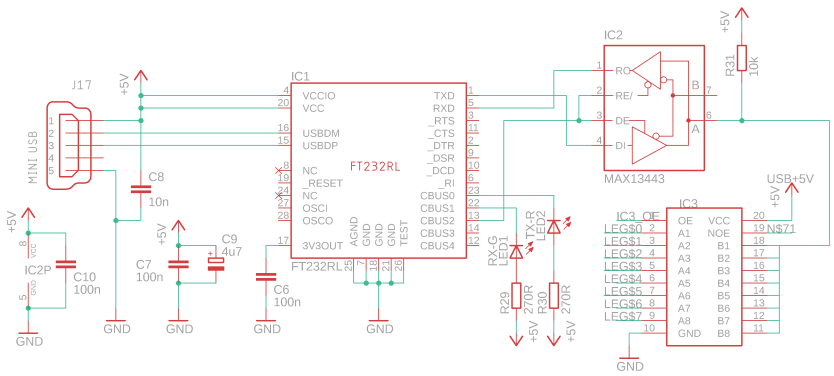

Figure 10: schema of UART USB.

cannot be plugged in the wrong direction. There are two connectors. One is male XT60 (J14) and the second one is constructed from the banana plugs, which should fit the hard case Li-poly battery(Hardcase connector in Figure 7).

In this thesis, all voltages from all cells are monitored by the ADC. Because the maximum voltage of the battery is 12.6 V it is decreased by the voltage divider. In addition, the hexapod robot can be run without the FPGA. For that purpose there is an analog solution (shown in figure 9). The Zener diode is a source of the reference voltage(2.7 V). When the voltage drops below 2.97 V the comparator turns off both P-mos power transistors. As the Li-poly battery has three cells, three circuits are used. The integrated circuit consists of four comparators, and the last one is used to turn on the power supply when the battery is firstly connected. The idea of this is simple. The voltage on a resistor-capacitor circuit slowly increases when firstly connect. When there is an under-voltage issue, the voltage is high enough to not influence the power transistors. The comparator is powered by the battery voltage therefore Schottky diode is used as reverse voltage protection.

The power transistors can be turned on only when right polarity is applied (as was mentioned before) and when Switch 1 is active. When this is done, the battery can be connected, and the resistor-capacitor circuit will open the transistor. Afterwards, the following condition has to be satisfied to keep the transistor conductive: The FPGA PIN has to be in high impedance or high level and the the voltage of the battery has to be above the 2.97V threshold.

3.2 Interface

The prototype board supports eight Molex connectors for servo motors. As was mentioned in subsection 2.1 the Dynamixel use TTL logic. However, the highest voltage which can be reached on the FPGA IO pins is 3.3 V. For this reason level shifter(TXS0108E) is used. The pins on FPGA were chosen to be close together to minimize the used area in the FPGA.

Block UART to USB (shown in Figure 10) provides the option to use the board without the FPGA. This block is based on the original dynamixel USB to half-duplex UART transceiver that is used when the robot is controlled from the PC. All eight data wires from Molex connectors are connected to the bus switch, the other ends of the bus switch are connected with the MAX13443 fault-protected RS-485 transceiver. This IC change the half-duplex UART to full-duplex UART which is then connected to FT232 that provides the USB connection.

The prototype board is equipped with six switches. SW1 consists of two switches for the two power circuits. The SW2 consist of four switches. The first one (in Figure 7 the most right one) has to be switched on if the UART to USB block is used. The second and the third ones can turn off the under-voltage analog protection. The last switch can shortcut the third balancer pin with the battery plus contact. It should be turned on when the hard case battery is used.

Table 6: Meaning of used LEDs.

Board reference	Colour	Meaning
LED1	Green	Illuminate when data is send from UART to USB.
LED2	Red	Illuminate when data is send from USB to UART.
D2	Green	Illuminate when power for the servomotors is active.
D6	Green	Illuminate when power for the electronics is active.
D9	Green	Illuminate when power for Xsens is active.
D10	Green	Illuminate when power for FPGA is active.

Table 7: Used FPGA pins.

FPGA pin number	Description
AF28	Xsens UART TX pin
AF27	Xsens UART RX pin
AA13	UART for leg 0
AA11	UART for leg 1
AA26	UART for leg 2
AB25	UART for leg 3
AB26	UART for leg 4
AA19	UART for leg 5
AA18	UART for leg 6
AB23	UART for leg 7
ADC_IN1	output of current sensor for servomotors
ADC_IN2	output of current sensor for electronic
ADC_IN3	voltage of the first cell multiply by 0.6
ADC_IN4	voltage of the second cell multiply by 0.37
ADC_IN5	voltage of the third cell multiply by 0.25

Meaning of all LEDs used on the prototype board is described in Table 6.

The prototyping board is connected with the FPGA by two headers. One is Arduino analog expansion header and the second one is GPIO0. Despite the GPIO1 does not contain any used pin, it can be connected to increase the rigidity of the construction. All used pins and their purpose are shown in Table 7.

3.3 Additional Features

The prototype board enables some additional features. One is current sensing. A typical ways of current sensing are by using current transformer or shunt resistor. However, in our application the usage of current transformer is limited as it cannot measure the direct current and the disadvantage of the shunt resistor is the power drop on it. For these reasons, a hall effect sensor is used in this work. This sensor measures the magnetic field, which is generated when current flows. The output is in Volts, and it is directly proportional to the current. The output voltage is multiplied by 0.6 (via the resistor divider) to ensure that the voltage will be in the range of the FPGAs ADC.

The second feature of the prototype board is to provide connectivity to of an inertial measurement unit, Xsens. The Xsens IMU needs a stable power supply, therefore a dedicated 5V power supply was designed, which is inspired by [20]

Chapter 4
Software Solution

4.1 FPGA Architecture

In this section, the proposed FPGA architecture is described. The whole project was programmed in VHSIC [5] hardware description language(VHDL) via Quartus prime, the programmable logic device design software produced by Intel. The FPGA architecture is designed as the System on Chip (SoC) with a custom designed Servo Control core. Therefore it benefits of both the general purpose CPU and the custom designed Servo Control core The main benefit of the CPU is, that it is easily programmable in C programming language and allows for rapid prototyping of the code. On the other hand, Servo control core uses the parallelism of the FPGA fabric for the communication with the individual servomotors. But the design in VHDL is time consuming.

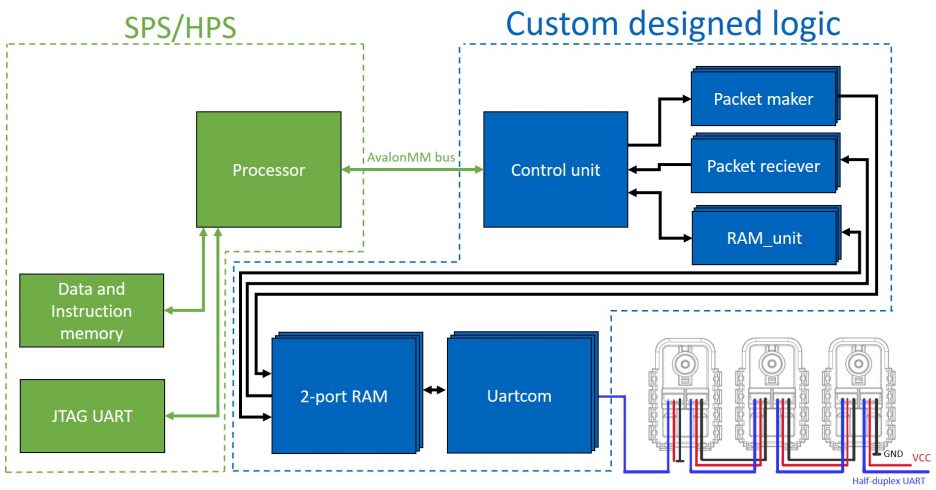

Figure 11: Diagram showing the buildings blocks of the FPGA architecture.

Figure 11 shows the overall structure of the FPGA architecture. The custom designed Servo Control Core can be easily added to project in a platform designer software because it is implemented as a QSYS component. Further, it can be connected to the processor using the Avalon memory mapped interface (AvalonMM) bus. The Servo Control Core comprises from the control unit and individual communication pipelines to the connected legs, hence, all the blocks except the Control unit are generated as many times as there are connected servo communication channels [6]. The black arrows represent the direction of data-flows and are implemented by a various number of signals. The important part of the design is the 2-port RAM, that is used for buffering of the packets that are to be send to the servos and for the servos responses. Each communication channel has its own 2-port RAM. The RAM can be written with new data from processor directly through the Control unit. This allows the processor to send an arbitrary packet to the servos. However, as the main motivation is to offload the computations of the processor, there are the Packet maker and the Packet Receiver modules that so far

[5] Very high speed integrated circuit
[6] This parameter can be changed in control unit

Table 8: List of signals connecting the ”Control unit”.

Signal name	Type	Direction	Description
Avs_s0_address	16 bits long logic vector	In	AvalonMM address
Avs_s0_read	Logic signal	In	Read enable
Avs_s0_readdata	16 bits long logic vector	Out	Red data
Avs_s0_write	Logic signal	In	Write enable
Avs_s0_readdata	16 bits long logic vector	In	Write data
Avs_s0_waitrequest	logic signal	In	Busy indicator
Clock_clk	Logic signal	In	Clock signal
Reset_reset	Logic signal	In	Reset signal
UART_out	8 bits long logic vector	Bidirectional	Half-duplex UART

implement an automated packet creation for the set position packet and the get position packet, and parsing of the servo response. The individual blcoks are described in more detail below.

Control unit

This core implements the interface between the FPGA and the SPS or HPS processor. In the control unit, AvalonMM interface is implemented by registers. All signals are listed in Table 8. There are control registers to control the overall behaviour of the core. Next, there are ranges of registers that are for direct write and read from and to the RAM. More information on the processor interface is written in Section 4.2.

In this unit, all other components are imported and the bidirectional UART pin splits here to Tx and Rx. It is working as follows, the UART signal is in high impedance when Tx is in logic one. When Tx is in the logical zero, the UART signal is in logic zero as well. The Rx signal sniffs (reads) the UART signal.

UARTcom

The Rx and Tx signals are connected (see Table 9) to the UARTcom unit that is based on the full-duplex UART solution presented in [21]. This unit is designed as a simple state automata and it has three states. Besides it has a byte count signal, that is used for counting the received bytes.

In the first state, the data is sending from the RAM to UART. The RAM has to contain the whole packet because neither data nor the checksum is modified by the ”UARTcom” unit. This unit only adds the start and stop bit to each byte. This state is started by a rising edge on ”send_signal” continuously.

In the second state, the UARTcom core is reading data from the RX signal. It is started when the unit is not sending any data and a falling edge appears on RX signal. This edge starts the clock process, which counts half of the baud rate period and samples the RX signal. From this moment the clock process counts the whole period. It means that the RX signal is sampled only once per period in the middle of it. In our experience, the signal is smooth and there is no significant noise. If there will be any problems in the future, sampling the signal for more times per period can be the solution.

The last state is the idle state. The system can get to this state when transmitting or receiving ends. Receiving can end in two ways, one way is that byte is received correctly. In such a case, the byte counter is incremented. The second way is that after the starting falling edge the bit is not in logic zero or if the stop bit is not logic one. After receiving the correct bit, the process waits 250ms for the next byte. If the next byte does not arrive during that time, the byte counter is restarted.

Table 9: List of signals connecting the ”UARTcom” .

Signal name	Type	Direction	Description
Clk_50Mhz	In	Logic signal	Clock source
Clk_out	Out	Logic signal	Clock for the RAM
Tx	Out	Logic signal	UART transmit signal
Rx	In	Logic signal	UART receive signal
RAM_address	Out	8 bits long logic vector	RAM address
RAM_write	Out	Logic signal	Write enable for the RAM
RAM_readdata	In	8 bits long logic vector	Data red from RAM
RAM_writedata	Out	8 bits long logic vector	Data to be stored in the RAM
Send_signal	In	Logic signal	Starts the sending process
DataReady_signal	Out	Logic signal	High when data can be write to the RAM

Table 10: List of signals connecting the Dual-port ram.

Signal name	Type	Direction	Description
Address_a	8 bits long logic vector	In	Address of the first port
Address_b	8 bits long logic vector	In	Address of the second port
Clock_a	logic signal	In	Clock of the first port
Clock_b	logic signal	In	Clock of the second port
Data_a	8 bits long logic vector	In	Data for writing to the first port
Data_b	8 bits long logic vector	In	Data for writing to the second port
Wren_a	logic signal	In	Write enable for the first port
Wren_b	logic signal	In	Write enable for the second port
Q_a	8 bits long logic vector	Out	Data red from the first port
Q_b	8 bits long logic vector	Out	Data red from the second port

Dual-port ram

The 2-port RAM was generated using the IP catalog. The RAM contains 256 8-bit words. Both ports are bidirectional and have a separate clock, all signals are listed in Table 10. All packets stored in the RAM starts from the zero address. The RAM uses one M9K embedded memory block. If data is read during the write operation, new data is read. All signals except the output data are registered.

The next three components share the second port of the RAM. The highest priority to access the RAM has the RAM_unit. If the RAM_unit is not used, the Packet maker can be used. The lowest priority has the packet receiver core.

RAM_unit

Firs unit to share the RAM is the RAM_unit. This unit enables reading and writing data from the processor to the RAM. The core is connected by the signals listed in Table 11. The process can be started by the rising edge of “read_RAM_ON” or “writeRAM_ON”. By this moment, the data and the address should be valid. When the process ends, “read_RAM_finished” or “write_RAM_finished” is at high level. This allows the processor to construct a custom packet to be send via the half-duplex UART interface. It is also used to read the response.

Table 11: List of signals connecting the ”RAM_unit”.

Signal name	Type	Direction	Description
Clk	Logic signal	In	Clock source
Asc_address	8 bits long logic vector	In	Address from HPS/SPS
Asc_write_data	8 bits long logic vector	In	Write data from HPS/SPS
Read_RAM_on	Logic signal	In	Reading from RAM enable
Write_RAM_on	Logic signal	In	Writing to RAM enable
RAM_q	8 bits long logic vector	In	RAM output
RAM_data_in	8 bits long logic vector	Out	Write data to RAM
RAM_adr	8 bits long logic vector	Out	Address for the RAM
RAM_write_enable	Logic signal	Out	Write enable
Read_RAM_finished	Logic signal	Out	When 0 is busy
Write_RAM_finished	Logic signal	Out	When 0 is busy
Data_out	8 bits long logic vector	Out	Exporting the red data

Table 12: List of signals connecting the ”Packet_maker”.

Signal name	Type	Direction	Description
Clk	Logic signal	In	Clock source
ServoID	8 bits long logic vector	In	ID of the servomotor
avs_data	16 bits long logic vector	In	Required angle or starting register
SetAngle_on	Logic signal	In	Creating write angle packet enable
ReadAngle_on	Logic signal	In	Creating read packet enable
RAM_write_enable	Logic signal	Out	Write enable for the RAM
RAM_write_data	8 bits long logic vector	Out	Write data to the RAM
RAM_adr	8 bits long logic vector	Out	Address for writing in RAM
Is_finished	Logic signal	Out	Busy indicator

Packet maker

The second core which uses the shared port is the packet_maker. This unit can only write to the RAM. This unit is able to create a set angle and read angle packets. The inputs are the servo ID and the set angle, provided by the processor through the control unit. The process will construct the rest of the packet. All signals connection this unit si listed in Table 12 The unit can be started by applying logic one to “SetAngle_on” or “ReadAngle_on”. When the process ends the signal “is_finished” is high.

Packet receiver

The last core which shares the port is the packet receiver. This unit read the data from RAM and check the information in it (such as the start bytes and checksum). If there is an error in the packet, the signal “errorDetected” goes high. See Table 13 for more information about the signals.

This process exports ID and status byte. If the signal “ReadAngle_en” is hight, the data of the packet will be exported in signal “ReadAngle”.

This process can be turned on when the “enable” signal is high. When the process ends, the falling edge will appear on “packet_receiver_on”.

Table 13: List of signals connecting the ”Packet_receiver”.

Signal name	Type	Direction	Description
Clk	Logic signal	In	Clock source
Data_ready	Logic signal	In	It is switching the core on
RAM_data	8 bits long logic vector	In	Red data
packet_receiver_on	Logic signal	In	Enable the core
ErrorDetected_on	Logic signal	Out	High when error is detected
RAM_adr	8 bits long logic vector	Out	RAM address

4.2 CPU Program

To run and debug the FPGA architecture, the soft-core microprocessor (NIOS II) is used. The software for the NIOS processor is written in C language.

The FPGA architecture is controlled by storing data to the registers. The FPGA core has a memory space which is addressable by 16 bits. In this QSYS system, the address space starts on 0x08000000. The address written in the ARM processor is four-time larger than it shows in the FPGA architecture. To write or read data from the registers HAL library is used. The first address byte determine the function and the second byte can be used in the function.

When the first byte is 0x00, the behaviour of the whole core can be changed. When a function writes 0x00 on a second byte, the send signal is sent to the Uartcom core [7]. There is the limitation that 32 separated UARTs is the maximum number this architecture can control as each bit in data signal stands for a single Uartcom core. When the processor reads this register, the data contains value data_ready which is active during the time the new data is receiving. The address 0x08000040 is reserved for the mode of the protocol unit. The address 0x8000080 enables to change the RAM for direct access. The direct access to a RAM is available when the first byte is 0x01. The Entire RAM can be addressed by the second byte.

The next Address space is dedicated to the information of the servo positions. Two approaches have been considered for organisation of the memmory mapped addresses.

The First one is to have for each servo ID one register block. The block would contain the information of the actual angle, the set angle, the status register and the leg number, where the servomotor is placed in. The disadvantage is that there could not be two servo motors with the same IDs on the hexapod.

The second way is to have one block for each possible leg (totally 32). This blocks will contains a smaller block for each servo. The one servo needs three registers, one for the ID, one for the status and one for the set and the read angle. The disadvantage is henceforth a larger memory space. However, there is the opportunity to have the same ID on each leg, so in case one leg has a defect, the whole leg can be quickly replaced with another.

In the herein presented architecture, we have selected the first way, as there is the need to maintain the ability to control the robot through the USB. This disallows usage of the same servo IDs per legs, although it would be beneficial regarding the possible repair time of the platform. A couple of RAM should implement the interface. One RAM available for the FPGA fabric and the second for the CPU. This ensures that no data cannot be change during the reading operation.

[7] This can be done by the SendToServo function

Chapter 5
Results

Because the material for soldering the prototyping board was not delivered on time, the experiments were done using prototype board equipped with the level shifter (TXS0104EDR). As there is no need in the dimension compatibility between the FPGA and the prototype board, a smaller FPGA, DE0-Nano, was used. This development kit has the advantage of smaller FPGA chip Cyclon IV family, meaning the compilation takes less time.

Firstly, the FPGA architecture was created, compiled, and uploaded to the DE0-Nano. Then the softcore program was compiled and uploaded to the chip. The programming was done via the Nios II Software Build Tools for Eclipse.

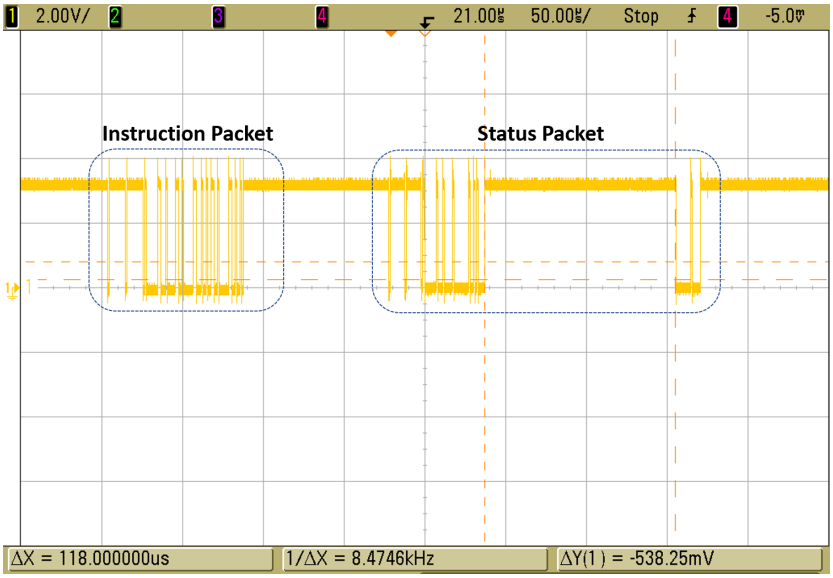

Figure 12: Screenshot from oscilloscope of UART communication contains Status Packet sent in two parts.

When we have started the communication with the servos, we have discovered an interesting behaviour of the servomotors that was previously unknown to us. Some packets were not received correctly. Oscilloscope [8] shows the record which is depicted in Figure 12. The Instruction packet is correctly transmitted, however the status packet (Servomotor response) pause in the middle of the transmitting. As further measurements show, this pause can be between arbitrary two bytes. Its length was measured in a range from 90 μs to 150 μs. Which is most likely caused by the servomotor control loop.

5.1 Communication Speed

After the ”UARTcom” core was modified, to consider the byte pause (up to 200 μ) as correct packet the comparison with the PC communication can be measured. The servomotor was set to send the

[8] Agilent Technologies DSO6104A

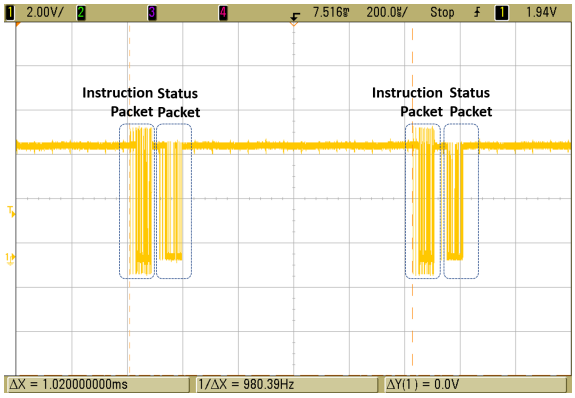

Figure 13: Oscilloscope screenshot of PC communication with a single servomotor.

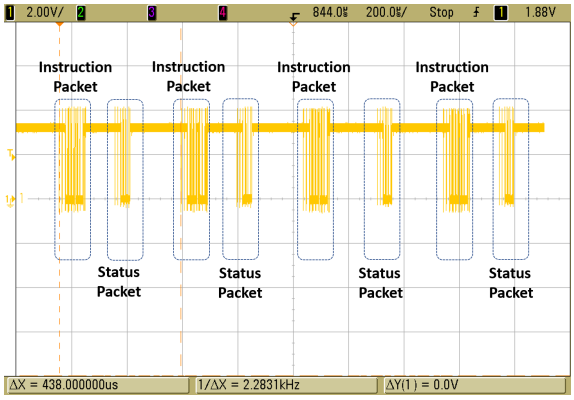

Figure 14: Oscilloscope screenshot of FPGA communication with a single servomotor.

Figure 15: Oscilloscope screenshot of PC communication with all servomotors.

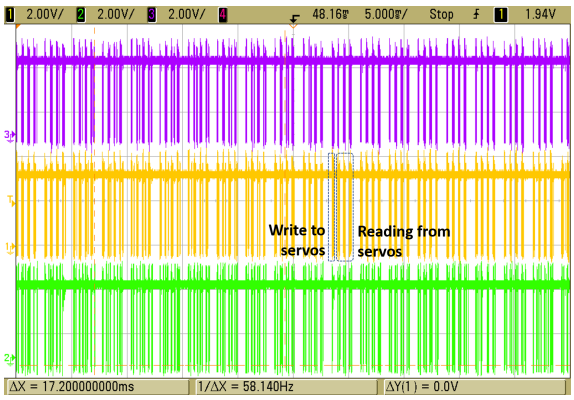

Figure 16: Oscilloscope screenshot of FPGA communication with all servomotors.

data immediately when it is asked. The PC communicates with the servomotor via the "Robotics USB2Dynamixel adapter" (all servomotors shared a single UART BUS).

The results are shown in Figures 13 and 14 the oscilloscope was set to 200 μs per division and 2 V per division for both figures. The signal in Figure 14 has a lower amplitude because the oscilloscope probe was placed to FPGA pin (3.3 V), whilst in the Figure 13 the probe measure the bus (5 V). As you can see, the FPGA can communicate more than two times faster than the PC. The reason for that is that the PC can access the UART only once per one millisecond which is caused by the operating system, which transfers content of the serial buffers to the running application only once per millisecond, unlike the FPGA which has no communication limits. FPGA needs only around 250 μs for reading a single register, but to remain on the safe side concerning the split packet problems, we have selected 440 μs as the base packet communication speed.On the other hand, PC needs 1 ms to read a single register from the servomotors.

5.2 Reading from All Servomotors

The next experiment was proving the ability to communicate with a multiple servos at the time. The results are shown in Figure 16. In the picture, only one-half of all signals are shown because the oscilloscope does not support six channels. As you can see, all instruction packets are synchronized (when we zoom in on the start of the instruction packet, the time shift is less than one nanosecond, see Figure 17). Therefore the read angles of all the joints are synchronized.

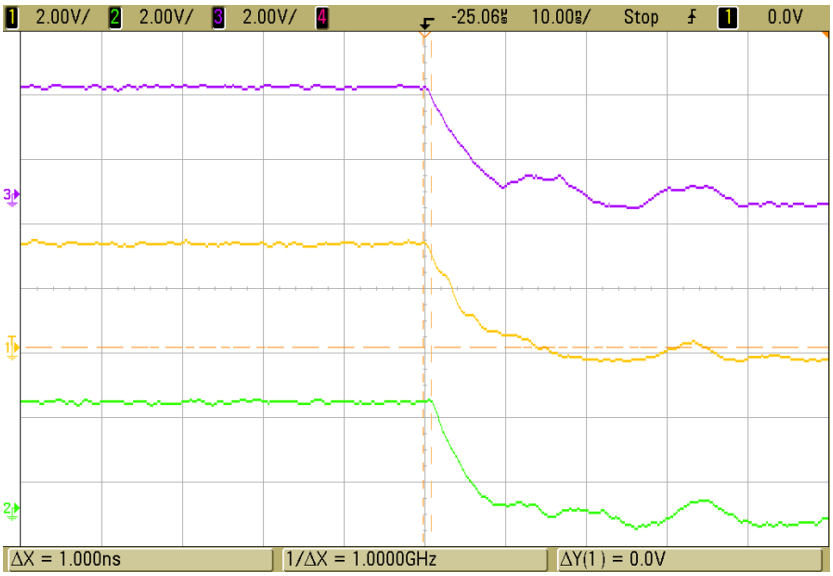

Figure 17: Time shift between the legs.

The motivation for this work is to speed the communication With this experiment we have proven the feasibility of the proposed approach as the FPGA is capable of commanding all the servomotors synchronously. All servos were set up as in the previous experiment and all servomotors connected to PC were connected in a single daisy chain. After running the code the waveform was as visualized in Figure 15. The first packet is longer than the others and it is bulk write after the reading phase starts. As there are eighteen servomotors eighteen packets have to be created and send. Using the PC, all the servomotors are read in 17.2 ms. The FPGA reads all the values in 1.3 ms which is 13 times faster in comparison to the PC implementation.

Chapter 6
Conclusion

In this work, a possible advantage of using FPGA for communication via half-duplex UART, with the Dynamixel AX-12A, servomotors of the hexapod walking robot has been examined and compared with the single bus communication. The time needed for reading angles from all hexapod servomotors was reduced from 17.2 ms to 1.3 ms. This significant improvement has been achieved thanks to the parallel connection of the legs, and the ability of the FPGA to read the UART anytime.

To enable FPGA to control the robot, the prototype board was designed, and it is described in Chapter 3. Unfortunately, the components for the prototype board has not arrived in time. Thus the prototype board was not experimentally verified. However, when the board is soldered up, it will provide the power supply for used electronics, protect the battery from under voltage, and provide other functions such as current sensing. Moreover, the board is prepared for using the magnetic buzzer indicator, which will warn before the low battery has a low level. This functionality was planned to be controlled by the FPGA, and it was not implemented yet.

The FPGA architecture was implemented as a smaller block, which might be possibly overused in other applications and on other FPGA devices. The architecture is described in Section 4.1 and it is built as the SoC design combining benefits of both the general purpose processor that is easy to programm and the high performance of the FPGA fabric. The CPU program was compiled for the softcore processor which is part of the FPGA. Despite hardware implementation onboard of the robot was not finished and the robot was not able to locomote, it has been proved that the communication speed significantly increases by using FPGA. Hence, we have verified the concept and the crucial component of the adaptive locomotion control (described in Chapter 2), which is the communication with the servomotors. The actual locomotion control of the robot can be built easily on the herein proposed hardware and software components as it will be deployed in software of the SoC system only.

References

[1] J. Bayer and J. Faigl, "Localization Fusion for Aerial Vehicles in Partially GNSS Denied Environments," in *Modelling and Simulation for Autonomous Systems (MESAS)*, 2019, pp. 251–262.

[2] M. Prágr, P. Čížek, and J. Faigl, "Cost of Transport Estimation for Legged Robot Based on Terrain Features Inference from Aerial Scan," in *IEEE/RSJ International Conference on Robots and Systems (IROS)*, 2018, pp. 1745–1750.

[3] M. Zoula, "Locomotion control of hexapod walking robot with four degrees of freedom per leg," Bachelor's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, 2019.

[4] J. Faigl and P. Čížek, "Adaptive locomotion control of hexapod walking robot for traversing rough terrains with position feedback only," *Robotics and Autonomous Systems*, vol. 116, pp. 136 – 147, 2019.

[5] S. Seok, A. Wang, Meng Yee Chuah, D. Otten, J. Lang, and S. Kim, "Design principles for highly efficient quadrupeds and implementation on the MIT Cheetah robot," in *IEEE International Conference on Robotics and Automation*, 2013, pp. 3307–3312.

[6] M. C. e. a. A. Espinal, H. Rostro-Gonzalez, "Quadrupedal Robot Locomotion: A Biologically Inspired Approach and Its Hardware Implementation," in *Computational Intelligence and Neuroscience*, vol. 2016, 2016, pp. 240–245.

[7] M. Henrey, S. Edmond, L. Shannon, and C. Menon, "Bio-inspired walking: A FPGA multicore system for a legged robot," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 105–111.

[8] S. Bartsch, M. Manz, P. Kampmann, A. Dettmann, H. Hanff, M. Langosz, K. v. Szadkowski, J. Hilljegerdes, M. Simnofske, P. Kloss, M. Meder, and F. Kirchner, "Development and Control of the Multi-Legged Robot MANTIS," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*, 2016, pp. 1–8.

[9] S. Pedre, "A new co-design methodology for processor-centric embedded systems in fpga-based chips," Doctoral thesis, Universidad de Buenos Aires. Facultad de Ciencias Exactas y Naturales Departamento de Computación, 2013.

[10] J. Mrva and J. Faigl, "Tactile sensing with servo drives feedback only for blind hexapod walking robot," in *10th International Workshop on Robot Motion and Control (RoMoCo)*, 2015, pp. 240–245.

[11] M. T. Nguyen, "High-fidelity modeling of hexapod walking robot locomotion," Bachelor's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, 2019.

[12] N. Porcino, "Hexapod Gait Control by a Neural Network," in *IEEE International Joint Conference on Neural Networks (IJCNN)*, 1990, pp. 189–194.

[13] *Dynamixel AX - 12*, ROBOTIS, 2006.

[14] S. Trimberger, “Three ages of FPGAs: A retrospective on the first thirty years of FPGAtechnology,” 2015.

[15] R. Cofer and B. Harding, *Rapid System Prototyping with FPGAs*. Newnes, 2006.

[16] C. Maxfield, *The Design Warrior’s Guide to FPGAs*. Newnes, 2004.

[17] E. Monmasson and M. Cirstea, “FPGA Design Methodology for Industrial Control Systems—A Review,” *Industrial Electronics, IEEE Transactions on*, vol. 54, pp. 1824 – 1842, 2007.

[18] S. D. Pandit and V. N. Shet, “Review of FPGA based control for switch mode converters,” *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pp. 1–5, 2017.

[19] *Cyclone V Device Handbook*, cited on 24-05-2019.

[20] *MTi 1-series Data Sheet*, cited on 24-05-2019.

[21] Y. Fang and X. Chen, “Design and Simulation of UART Serial Communication Module Based on VHDL,” *3rd International Workshop on Intelligent Systems and Applications*, pp. 1–4, 2011.