TAMPERE UNIVERSITY OF TECHNOLOGY

JUKKA SADEHARJU

DESIGNING EXERCISE WORK FOR SYSTEM-ON-CHIP COURSE

Master of Science Thesis

Examiners: Professor Timo D. Hämäläinen, PhD Erno Salminen

Examiners and topic approved in the Faculty of Computer and Electrical Engineering council meeting on 6.6.2012

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Sähkötekniikan koulutusohjelma
SADEHARJU, JUKKA: Harjoitustyön suunnittelu SoC-kurssille
Diplomityö
2012
Pääaine: Sulautetut järjestelmät
Tarkastajat: Timo D. Hämäläinen ja Erno Salminen
Avainsanat: Moniprosessorijärjestelmät, Sulautetut järjestelmät, SoC alustat, Harjoitustyö

Usean prosessorin mikroprosessorijärjestelmät ovat kasvattaneet suosiotaan jatkuvasti viime vuosien aikana. Tähän on johtanut ensisijaisesti tarve tehokkaille ja monimutkaisille sulautetuille järjestelmille joiden fyysinen koko on rajoitettu. Tällaiset järjestelmät vaativat monenlaisia järjestelmä- ja suunnitteluteknikoita mahdollistamaan tuotteiden tehokas suunnittelu ja kohtuulliset tuotantokustannukset.

Tässä työssä pohditaan ”TKT-3541 SoC Alustat”-kurssiin liittyvän harjoitustyön sisältöä ja rakennetta. Koko projektin tarkoituksena on luoda kurssille uusi harjoitustyö, joka aiempaa työtä paremmin kuvaa käytettyjä tekniikoita ja menetelmiä. Näihin nojautuen pyritään löytämään ratkaisut jotka parhaiten soveltuvat ajallisesti rajatun kurssin tarkoituksiin.

Harjoitustyön lähtökohtana ovat tavoitteet jotka Tampereen teknillisen yliopiston tietokonetekniikan laitos on kurssille asettanut. Näihin tavoitteisiin pyritään vastaamaan harjoitustyötä rakennettaessa. Kurssin yleisten vaatimusten lisäksi harjoitustyön luomisessa on pyritty koostamaan harjoitustyön vaiheet mahdollisimman hyvin nykyaikaisia järjestelmien vaatimuksia kuvaavaksi.

Työssä pyritään koostamaan harjoitustyö, joka opettaa MPSoC -järjestelmien kokonaisvaltaisen rakenteen. Järjestelmiin liittyviä ratkaisuja opetetaan mahdollisimman syvällisesti yhden lukukauden puitteissa. Järjestelmien rakenteen ja osien lisäksi pyritään käyttämään nykyaikaisia suunnittelumalleja, kuten uudelleenkäyttö (engl. reuse) ja alustaperustainen suunnittelu (engl. platform based design). Harjoitustyö tehdään ohjelmoitavalle logiikkapiirille (FPGA), joka mahdollistaa erilaisten järjestelmäteknisten ratkaisujen käytön ilman laitteeseen tehtäviä fyysisiä muutoksia.

Työssä pohditaan miten harjoitustyö muodostuu ja miltä osin pystyimme parantamaan harjoitustyötä suhteessa aiempaan harjoitustyöhön. Harjoitustyöstä saatiin edellistä paremmin esiin järjestelmän alustat sekä niiden osat. Järjestelmän rakenteeseen tehtiin muutoksia jotka mahdollistavat paremmin todellisissa järjestelmissä käytettävien tekniikoiden käytön. Lisäksi järjestelmän alustan ja sovellustason välinen rajapinta muodostettiin järkevämmäksi. Samalla opetetaan standardien mukaisten rajapintafunktioiden ja laiteajurien hyötyjä.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Electrical Engineering
SADEHARJU, JUKKA: Designing Exercise Work for System-on-Chip Course
Master of Science Thesis
2012
Major: Embedded systems
Examiners: Timo D. Hämäläinen ja Erno Salminen
Keywords: Multiprocessor systems, Embedded systems, SoC Platforms, Exercise work

Multiprocessor systems have become very popular system design during last decades. That is consequence of need for more and more efficient systems with low physical space. This kind of systems need several different system and design methods to give efficient system design and reasonable production expenses.

In this thesis I consider the contents and structure of the exercise work of the course "TKT-3541 SoC Platforms". The purpose of the project is creating the exercise work that improves from the former exercise work of the same course. Different techniques and approaches are considered to give the best solutions for the work with certain time limitations.

This work tries to find the topics and content that fulfils the requirements which are set to it by the department of the computer sciences. Among general requirements for the course, the system of the exercise work tries to describe the real life systems-on-chip.

This work describes the phases and content of the exercise work. That exercise work teaches the principal structure of multiprocessor systems-on-chip and different approaches related to those systems. The exercise work is implemented to the field programmable gate array logic which makes possible to create different platform structures without physical changes.

The improvements to the former exercise work were numerous. Different parts of the platforms are more visible in new exercise work and design approaches used were clearer. For example, reuse is significant approach that is used better in the new exercise work. Also the abstractions and the interfaces are examples of the major improvements of the work.

PREFACE

This Thesis is written for the Department of the Computer systems of the Tampere University of Technology. This work is part of the project to create exercise work for cource TKT-3541 SoC Platforms. It is written during years 2011 and 2012.

I want to thank my examiners Timo D. Hämäläinen and Erno Salminen for the comments and guidance during the work. I also want to thank Jussi Raasakka for the implementation of the exercise instructions and student guidance during the spring 2012. Great thanks for my family for their patience and support during this work.

TABLE OF CONTENTS

ABBREVIATIONS AND ACRONYMS

ADD	Area-Driven Design
API	Application Program Interface
BDD	Block-Based Design
BDF	Block Design File
DRAM	Dynamic RAM
DSP	Digital Signal Processor
eCos	Embedded Configurable Operating System
FPGA	Field Programmable Gate Array
HAL	Hardware Abstraction Layer
HD	High Definition
HDL	Hardware Description Language
HW	Hardware
IC	Integrated Chip
IP	Intellectual Property
IPC	Inter Processor Communication
IP-XACT	Component metadata description standard in XML format
IRQ	Interrupt Request Query
JTAG	Joint Test Action Group
LAB	Logic Array Blocks
LE	Logic Element
LED	Light Emitting Diode
LUT	Lookup Table
MCAPI	Multicore Communication API
MPSoC	Multiprocessor System-on-Chip
NoC	Network-on-Chip
PBD	Platform Based Design
PCB	Product Circuit Board
RAM	Random Access Memory
RTL	Register Transfer Level
RTOS	Real-Time Operating System
Rx	Receive
SoC	System-on-Chip
SBT	Software Build Tool
SRAM	Static RAM
SW	Software
TDD	Time-Driven Design
Tx	Transmission
UML	Unified Markup Language
VLNV	Identity data including Vendor, Library, Name, and Version

VLSI	Very Large Scale Integration
XML	Extensible Mark-up Language
µC/OS-II	Micrium Microcontroller Operating Systems Version 2

1 INTRODUCTION

Multiprocessor systems-on-chip (MPSoC) are getting more and more important design technology when high performance is required in embedded systems [1][2]. So, MPSoCs are taking foothold from general purpose architectures when designing high performance systems. They are used in systems that require multiple processing elements. MPSoCs are more specific for the designed system than general purpose processor systems. That makes possible to create more efficient and low energy systems that meet the certain performance requirements.

This thesis describes the design of an MPSoC exercise work created for the course TKT-3541 SoC Platforms [3]. SoC Platforms is the course of the department of Computer Systems in Tampere University of Technology. The exercise work teaches the latest design approaches and technologies, for example reuse of SoC designs and inter component communication with network-on-chip. The exercise work includes the layout design flow of the MPSoC system: the design of system platform, its components, application layer, and HW/SW co-design. The hardware platform is implemented on FPGA. Hardware platform includes microprocessors and intellectual property blocks (IP) communicating with a network-on-chip.

This Thesis also describes the challenges and prospects of the design of MPSoC system for giving the overall picture of the SoC platforms. Biggest challenge is limited time when going through this large subject. It prevents us from creating the most illustrative multiprocessor system. Still, the MPSoC platforms are quite easy to create with the tools used in this exercise work. That helps students to understand the structure of modern SoC platforms. So, the fundamental goal in this exercise work is to give the overall picture of SoC designs and abilities to understand more specific functions of the SoC and its components.

Finally the thesis explains why the exercise work is constructed to the form it has. It explains why the selected methods are used and why structure of the system is like it is.

2 MULTIPROCESSOR SYSTEM-ON-CHIP

System-on-Chip (SoC) is a complex integrated chip (IC) that integrates different functional elements into a single chip. The multiprocessor SoC (MPSoC) is a SoC that uses multiple programmable processors in its design. MPSoC is a very large scale integration (VLSI) system that incorporates most of the components used. In last decade it has became more general and entered the marketplace [1][2]. The recent progress in the microelectronics has given the possibility to build processors, digital hardware and mixed-signal circuits integrated into a single chip. However, putting multiple processors, memories, buses and peripherals into a single chip reduces energy consumption and space of the chip but brings on challenges for system designers. In Figure 1 there is an example composition of MPSoC system. It includes three processing units, two external communication components, memory controller, and embedded memory.

MPSoC are used in several different applications. High end mobile phones are products where the high efficiency is essential. Mobile phones should be able to use relatively long time with one charge. For example Exynos 4210 [4] MPSoC is used in several Samsung mobile phones. It has CortexA9 dual core with 1.0GHz processors and 45nm line width. For example, it has performance to run several hours high definition (HD) videos with a 1650mAh battery.

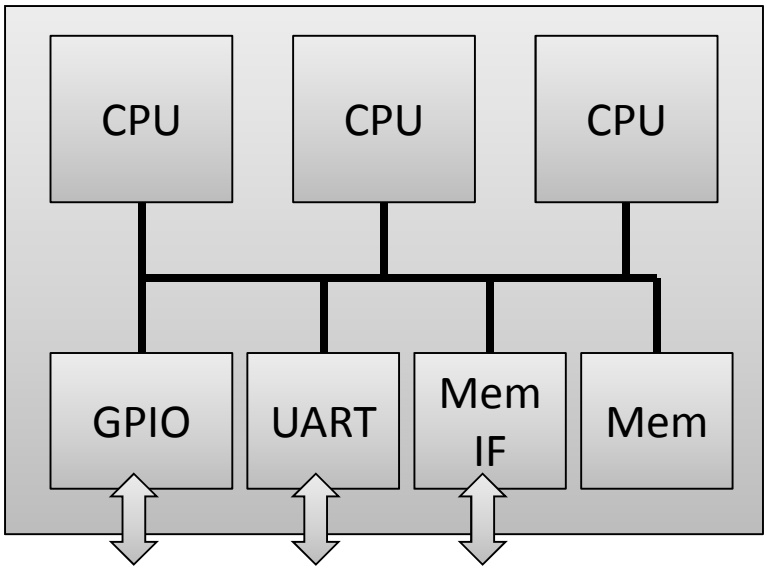

Figure 1: Basic example of MPSoC system. This system includes three processing units (GPUs), general purpose I/O port (GPIO), universal asynchronous receiver transmitter (UART), memory interface for external memory, and internal memory chip.

As well as the consumer electronics the product requirements are the same in other application fields. Portable medical devices require reliable electronics with low energy consumption, for example, electrocardiogram analysis, hearing aids, data compression or encryption [5]. For example, Icycom is a low power radio frequency DSP SoC for wireless sensor and body networks [6]. It has adjustable clock frequency and 180nm line width. It runs at an average of 150µA/MHz.

2.1 Design Methodology

The Mix of different technologies in one chip is a great challenge for a designer, due to heterogeneous components in one design [7]. There are different design methodologies to use in a SoC design. For example, Soo Hoo Chang et al. divide these methodologies to area-driven design (ADD), block-based design (BBD), timing-driven design (TDD), and platform-based design (PBD) [8]. ADD focuses on the area of the chip. The limited area of the chip often leads to problems with performance. BBD is a design approach when the components are reused which saves work when the system design gets more complex. TDD is an approach to concentrate timings of signals. MPSoC systems are getting more and more complex besides improving technology so the BDD becomes very reasonable approach. PBD includes the cumulative capabilities of the timing driven development (TDD) and BBD and benefits of design reuse and design hierarchy [8]. Platform in BDD is a reusable design that can be used with several different systems.

PBD is the design method used in the exercise work described in this Thesis. The system platform includes hardware and software platforms. The hardware platform is a family of architectures that satisfy a set of constraints imposed to allow the reuse of hardware and software components [9]. The software platform is on top of the hardware layer and makes it possible to create more abstract and reusable application. Platform-based design divides the system design into two phases: the platform design and the function design. Figure 2 shows the division to the platform and function. These two design parts are mapped to one design and turned into logic gates design in synthesis. The platform design is generated for a class of applications and the platform is adapted for a particular product in that application space [1].

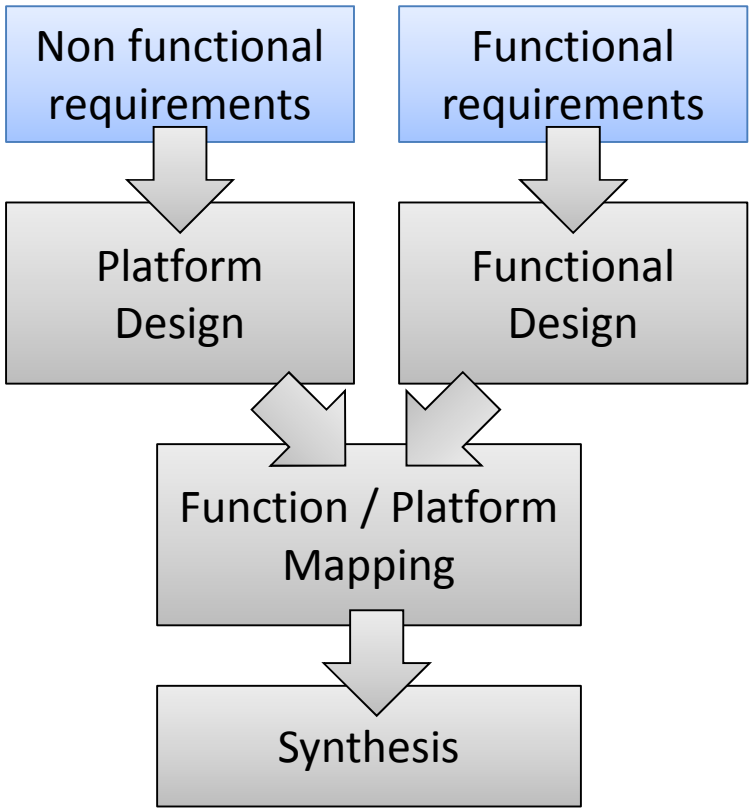

Figure 2: Platform based design include the division to platform design and functional design. These two design parts are mapped together for a synthesis.

The platform design and the functional design are two different designs flows that implement one common design. The PBD design process is meet-in-the-middle process (Figure 3). The functional design is a top-down approach where a function instance is mapped from the function space into an instance of the platform. The function space includes all possible functions that can be implemented and desired functions are selected from that space. These selected functions are mapped into a platform instance. The platform design is a bottom-up approach where the platform instance is built by choosing the components from the platform space that characterizes it [10]. The platform space includes all possible platforms and the desired platform is restricted from that space. This means that every feature in our platform is specified from the platform space. This composition of the designs makes the HW/SW co-design one of the most important technology in PBD.

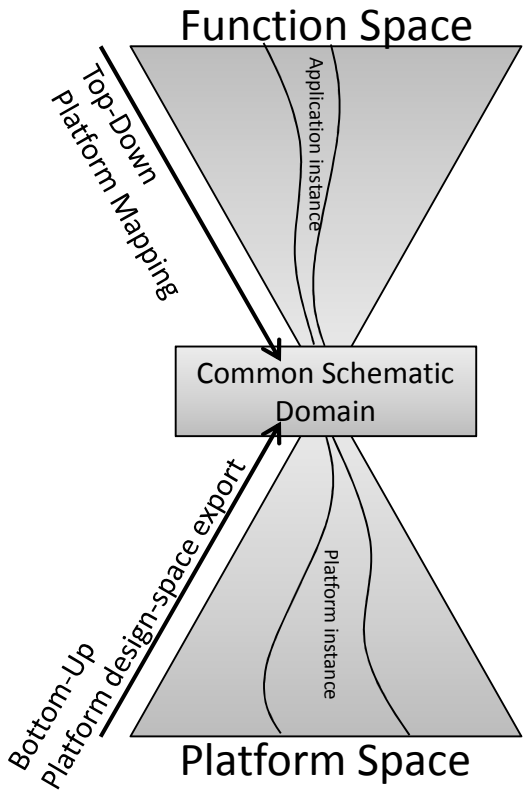

Figure 3: PBD Triangles shows the meet-in-the-middle design approach. A function instance from the function space is mapped to a platform. A platform instance is exported from the components selected from the platform space.

If non functional requirements of different system variations are similar, it is profitable to create platform that can be reused in different cases. The application design enables the multiple system variations even when the designed hardware is same. That division enables a large amount of hardware production and leaves the possible product variations for the functional specification. This decreases cost of the hardware.

2.2 System Specification

In platform based design the application layer runs on a system platform. The system platform is divided to the hardware (HW) platform, the software (SW) platform. The Application layer is the functional layer on the system platform. This is shown in Figure 4. The hardware platform includes the physical device and generated architectures which are implemented on that device. When designing large embedded systems the cost of the design is high. It is essential to reuse the system components to reduce the redundant work and design costs of similar projects. The purpose of the software platform is to ease the reuse of software in the application layer [11]. The software platform makes abstractions of the hardware platform for the application layer. For example, the software layer includes an operating system and API. Usually the software platform and

the application layer include all of the product specialisation and the hardware platform is the same in several projects.

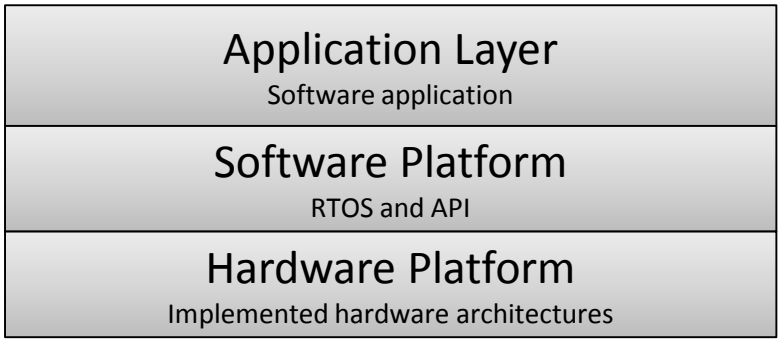

Figure 4: Hardware platform is the physical base of a product. The application layer is always modifiable and includes the functionality of the product. A software platform is between these two layers and makes possible to create reusable applications onto the hardware platform.

The system platform of MPSoC has different levels of functionality and abstractions. Petkov et al. [7] divide the MPSoC system to three abstraction levels; register transfer level (RTL); bus functional level; and system level (see Figure 5). These levels raise the level of abstraction, so that the application can be created without knowing the lower level functionality. RTL is the physical connection and design that connects the processors, memories and IP blocks together. Bus functional level includes a connection for software components and hardware IPs. System level includes the execution environment that uses abstract models for software and hardware components. The software model is for example real time operating system (RTOS) that include application program interface (API) for higher level software components.

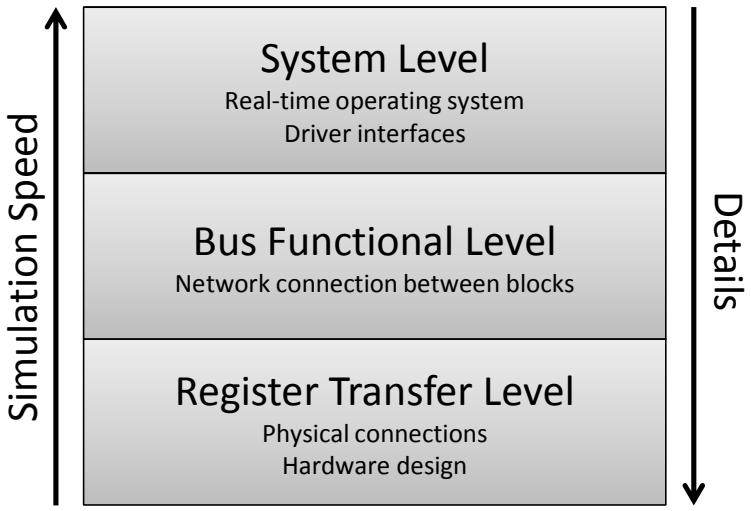

Figure 5: MPSoC system platform hierarchy levels. Register transfer level is the lowest level and is most complex in large systems. Bus functional level includes the system design components and connections between them. System level is the highest level and it includes the driver function interfaces for application development. Simulation speed of the system is higher when moving towards the system level and decreasing the details of the system.

2.3 System-on-Chip Reuse

Reuse of IP blocks is one of the most important design techniques to get high quality system with good productivity and low time-to-market. Whenever some functionality may be needed again after the use in first target design, the reuse of a component design should be considered. In SoC systems, nearly all designed IP blocks should be designed reusable. Because the system designers use more and more software to implement their products the design methods have to allow the reuse of software. In platform based design the interface of hardware platform have to be abstracted so that application software uses the higher level interface. That is called application program interface.

The design of reusable blocks is more difficult in SoCs than generally in software technology because of variety of technologies to do the design [2]. Reuse of components increases the efficiency of system design by reducing redundant work. The designer can use an IP without having to worry about internal details. A system developer that uses IP blocks has to configure them and connect the interfaces of the components in own designs. In Figure 6 there is an example of two implementation variations of the same system. That demonstrates the advantages to use the reusable components whenever it is possible.

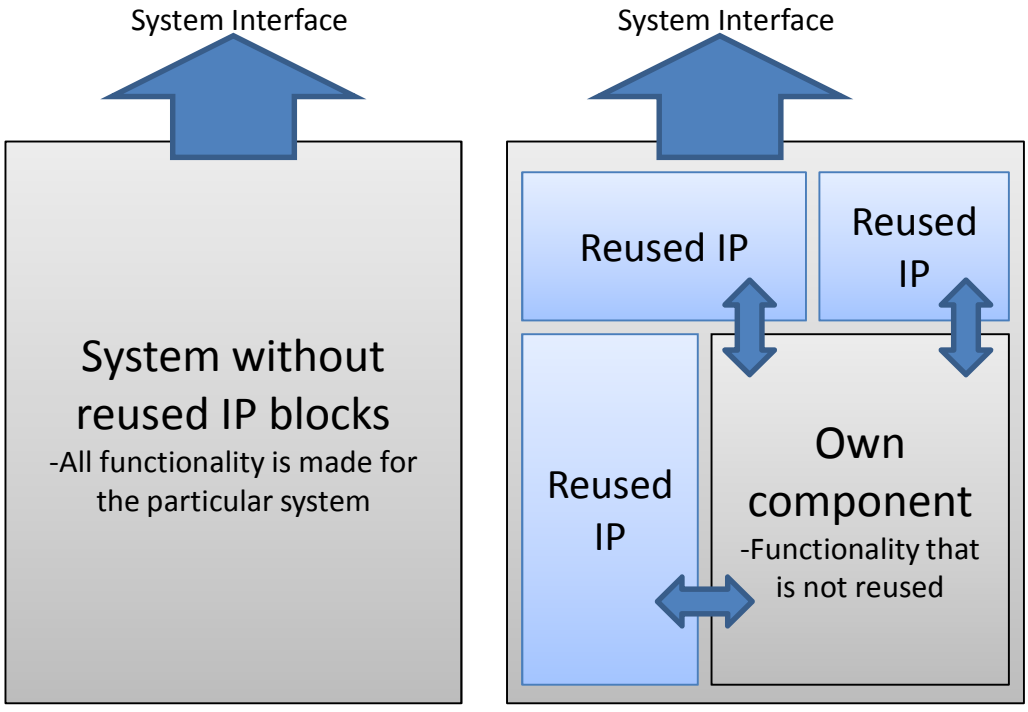

Figure 6: Example of two system implementations. In the left-hand system all functionality is self made. On the right-hand side, reused components are used as parts of the system. That left less work for the system designer, if we assume that the functionalities and interfaces of the blocks are adequately designed.

There are two possible ways to use the reusable block in the larger system. Components can be used as it is distributed or adapted to meet the desired SoC requirements. The former means that the component is fixed as it is and all system specific functionality should be added elsewhere in the system. In the latter, the IP include some functionality that can be configured when including it to a system either by setting parameters or modifying the source code of the IP [8].

Reused IP blocks can either be soft or hard IPs. Soft-IP is used as a model and hard-IP is used as a closed hardware chip. Soft-IP can be used as it is or modified. Hard-IPs cannot be modified. Modifications to the hard-IP functioning has to be done externally with an adapter block. For example, the HIBI bus that is shared as HDL files is a soft-IP that can be fully tailored to its target. Nios II processor is considered as firm IP as it works only on Altera FPGA's. Firm IP include the characteristics from both hard-IP and soft-IPs. The functioning of Nios II cannot actually be modified, but there are several different configurations that can be used, and system can be altered that way.

This kind of reuse is called internal reuse. There the IP blocks are used as a part of the system. Whole SoC system can also be reused externally as a part of a more complex system. That is more abstract reuse than IP reuse mentioned earlier [8]. There the idea of reuse is exactly the same, but abstraction level is different. For example the digital tuner could be the IP that is used in a set-top box and that can be used as a part of a media centre.

2.4 Field Programmable Gate Array

Hardware of the exercise work is based on the field-programmable gate array (FPGA). FPGA is a semiconductor device that can be configured after manufacturing. FPGA consist of logic elements (LE) that can perform logic operations, and connections between LEs. LE can perform complex combinatorial functions or simple logic operations. Usually logic blocks also include memory elements, such as flip-flops.

The FPGA structure described in this work is Altera Cyclone II[12], because DE2 education board we use includes Cyclone II chip. LE is a smallest unit of logic in FPGA and it can be used to implement custom logic. Each LE includes a programmable register that can be configured. In normal mode, four inputs from the local interconnect are inputs for four-input look up table (LUT).

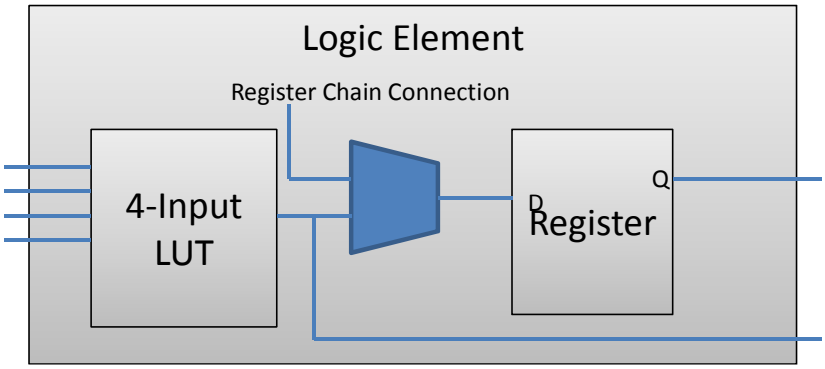

Figure 7: Simple figure of logic element. Main parts for the LE logic are lookup table and programmable register.

LEs are grouped in Logic array blocks (LAB) which is connected to hierarchical interconnects (see Figure 8). Each LAB is connected to local interconnects which is connected to row and column interconnects. Each LAB also have local interconnect to neighbour LABs and every LE is chained together inside each LAB as a register chain connections (see Figure 7). These local connections saves a capacity of local interconnects, because adjacent logic can be fitted into adjacent logic elements. Interconnects are connected together with switches.

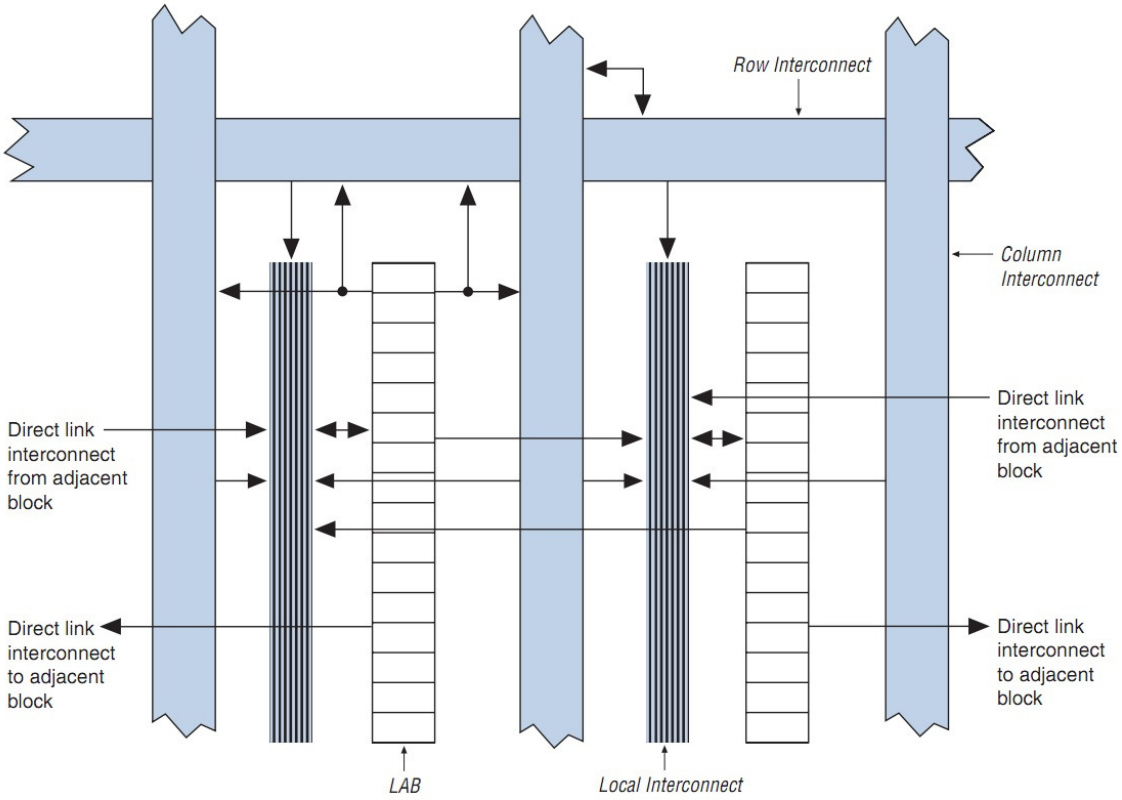

Figure 8: Cyclone II LAB Structure [12]. LABs are connected together with register chain connections, local interconnections, and row and column interconnections.

2.4.1 Altera Development and Education board DE2

Altera DE2 development and education board (Figure 9) is used in the exercise work described [13]. The board is used for teaching in several cources of the Department of Computer systems, so the students are probably already familiar with the board [14]. It includes several microchips and interface connections. The most important chip is the Cyclone II FPGA that is used to implement logic of the SoC. Other components that are used in the exercise work are buttons, LEDs and 7-segment displays for user interface and random access memories (RAM) for NIOS II processors that are synthesized in Cyclone II.

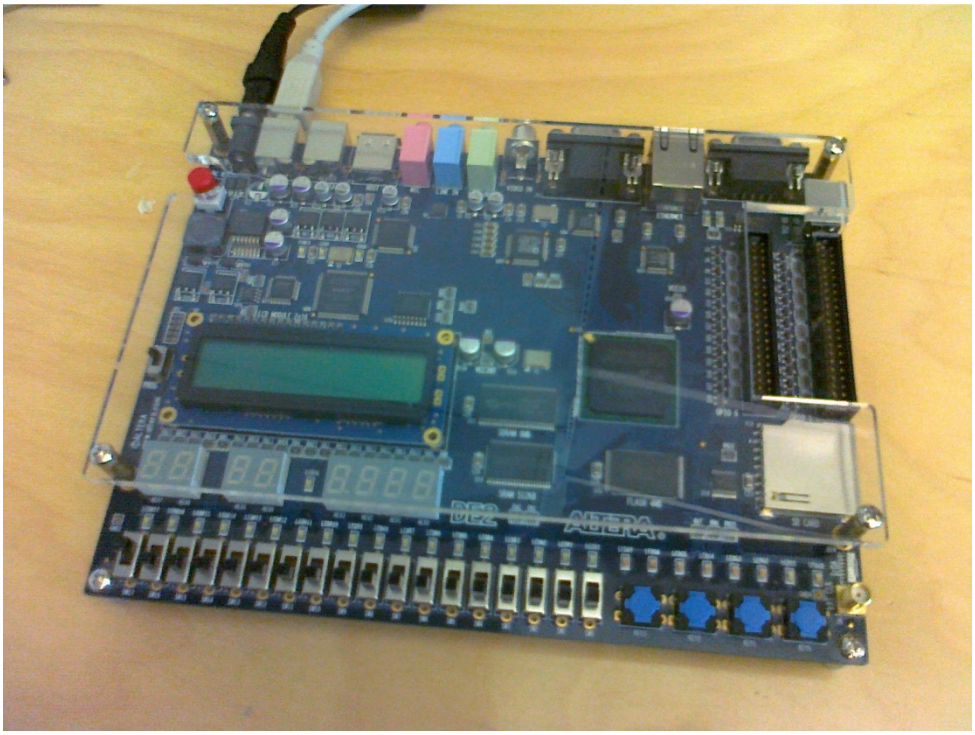

Figure 9: Altera DE2 development and education board.

2.4.2 Altera Quartus II

Altera distributes Quartus II software as FPGA designing and programming software. Quartus II includes solutions for all phases of FPGA design flow (see Figure 11). The first phase of the Quartus II design flow is the design entry that includes design of the implemented system. The design entry includes Hardware description language (HDL) and block design (BDF) files that defines the system. Quartus II makes the analysis and synthesis of the design. Quartus II software includes block designer that can be used to connect blocks and simple logic to larger designs (see Figure 10). Example shows how phase-locked loop (PLL) is connected to the clock input. It generates two output clocks,

one for DRAM and the other for the unnamed block. Quartus II also includes tools for component creation, like SOPC builder.

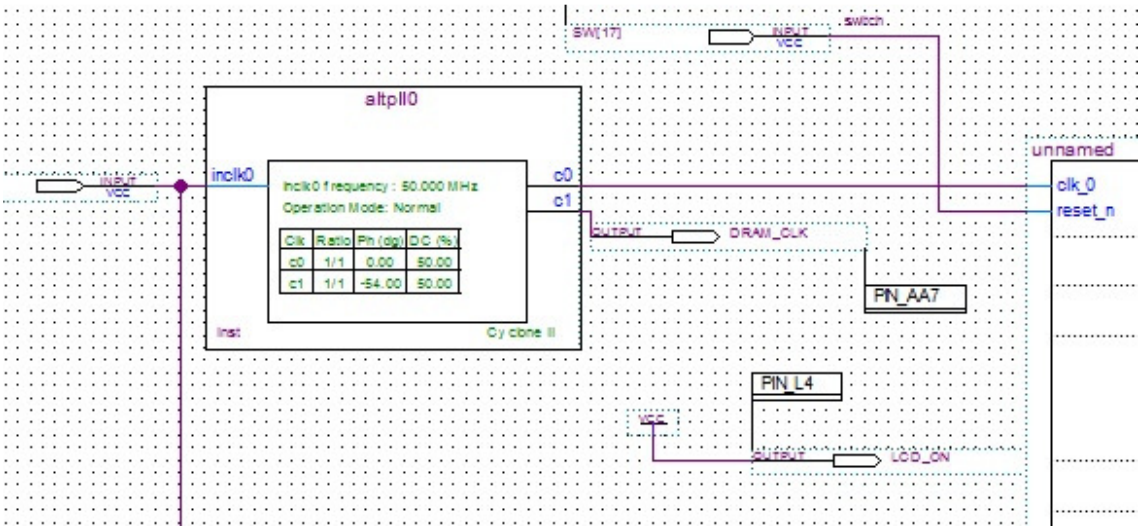

Figure 10: Screen capture of Quartus II block designer. It includes two inputs and two outputs that are connected to blocks with wires.

Analysis and synthesis examines the logical completeness and consistency of the project. Analysis and synthesis phase also synthesizes and performs technology mapping on the logic in the design. Synthesis minimizes gate count, removes redundant logic and utilizes the device architecture as efficiently as possible. [15]

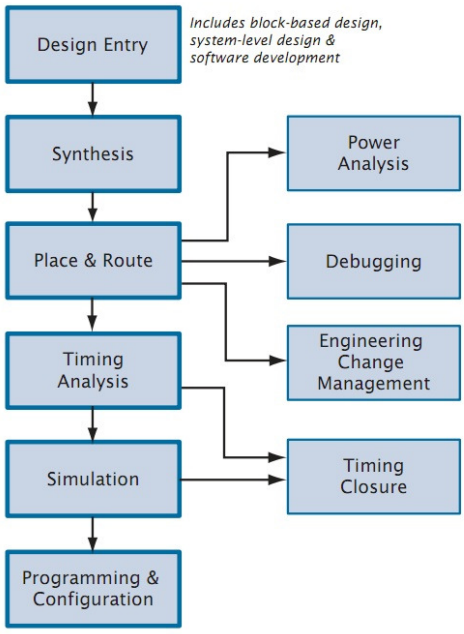

Figure 11: Quartus II Design Flow [15]. Design entry includes the design of the system. Quartus II delivers several different tools for system designing. Synthesis include technology mappings for the place and route. Place and route fits the design to timing analyzer, simulator, net list writer, or assembler.

The fitter matches the logic and timing requirements with the available resources of the FPGA target device. That is the place and route. Fitter optimizes the logic functions to the best physical locations for routing and timing. Fitter places the associated logic within an LAB or adjacent LABs allowing the local and register chain connections [12]. After fitting, the assembler module generates the programming files that can be programmed to a device. [15]

2.4.3 Altera Nios II

Altera offers a general-purpose RISC processor core for Altera's FPGA devices. Nios II is a very versatile processor. It can be as small as 600 logic elements but with different constitutions the performance can be over 300 MIPS [16]. The versatility of Nios II processors is useful for our MPSoC system. Nios II allows a designer to create own components to the processor system, so all own components do not have to use general input and output pins (cp. general-purpose microcontrollers). In the exercise work described in this Thesis, an example of own component is a processor element for HIBI that connects processor to a HIBI network. Nios II system saves logic elements in FPGA chip, because it can be constructed as needed, so unneeded features can be left out. One example of Nios II processor design is in Figure 12. There is Nios II processor core connected to four different memories, universal asynchronous receiver/transmitter (UART), two timers, and interfaces to other components.

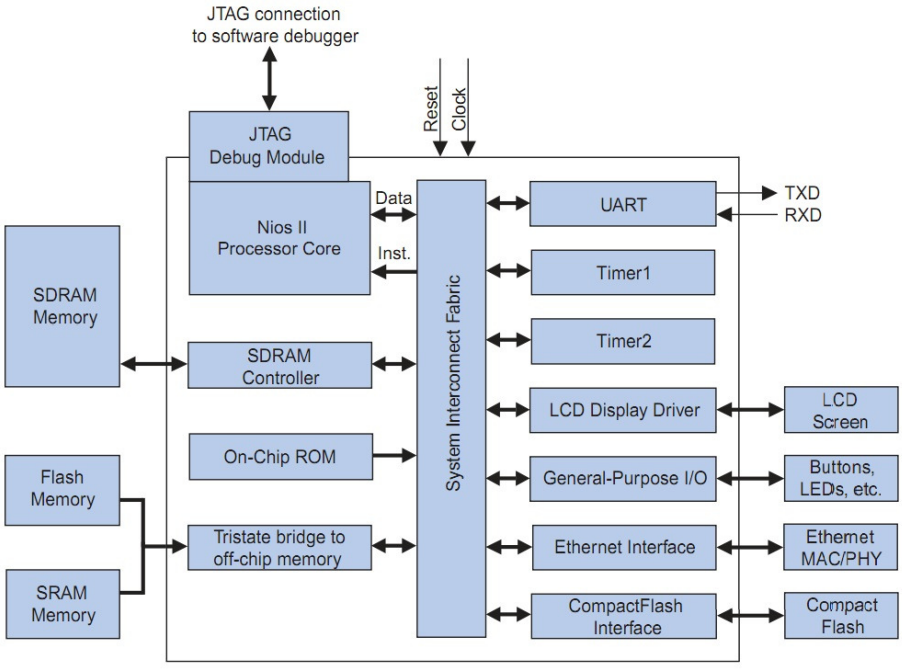

Figure 12: Example of Nios II processor system [17].

3 OTHER SOC COURCES AND EXERCISE WORKS

There is a wide range of MPSoC courses in different universities. The topics of the courses vary between different schools, but some of the topics are focused in almost every related course. Courses which are taken along in this investigation are listed in Table 1.

Table 1: List of the System-on-Chip courses.

#	Course	University	Course nr.	Year	Material	Ref
1	SOC Design Lab	NTHU, Taiwan	EE5255	2004	Lecture notes	[18]
2	System-on-Chip Design	University of Texas	EE382V	2010	Lecture notes and books for reading	[19]
3	SoC Design	University of Turku	ETT_2014	2010	Lecture notes and extras	[20]
4	System-on-Chip Design	University of Cambridge		2011	Lecture notes	[21]
5	SoC Design and Verification with System Verilog	San Jose State University	EE272	2011	Lecture notes	[22]
6	System-on-Chip	University of Southampton			N/A	[23]
7	System-on-Chip Design for DSP and Communications	University of Westminster			N/A	[24]
8	System-on-Chip (SoC) Design	University of Twente	121075	2011	Website material	[25]
9	Computer Hardware - a System on Chip	Linköping Institute of Technology	TSEA44	2011	N/A	[26]
10	System-on-Chip Design	University of Illinois	ECE 527	2010	Lecture notes	[27]
11	SoC-design	Tampere University of Technology	TKT-2431	2012	Lecture notes and books	[28]
12	**SoC-platforms** (Course of this work)	Tampere University of Technology	TKT-3547	2012	Lecture notes	[3]

Topics of these courses are listed in Table 2. Almost all of those topics are fundamental in every MPSoC device but are not always needed to show the clear picture of the MPSoC.

The most common topic is the concept of SoC platforms. That is the main thing that separates the MPSoC from general purpose computers. This makes the teaching of platform the most significant topic in most of these courses and it is the main topic in this very course as well. SoC platforms are related in almost all of the other topics in the list. For example, SoC reuse and Network-on-Chip (NoC) cannot be fully handled without considering platforms. Another important topic is SoC reuse. Reuse is design approach that aims to lower the cost of SoC design. With reuse the IP blocks and software implementations can be reused in different applications. It is more and more valuable approach when sizes of the designs get larger. That is why the topic is also an important topic in courses.

Despite layered approach, there are always some dependencies between HW and embedded SW. That means that the HW/SW co-design cannot be bypassed when designing MPSoC products. Hardware and software have to be designed to function together in a product so that the requirements and limitations of hardware have to be taken into account when designing software and vice versa.

Table 2: Topics of the System-on-Chip courses.

	Course	1	2	3	4	5	6	7	8	9	10	11	12	#
Topics	**Platform**		x	x	x	x	x	x	x	x	x	x	x	11
	Reuse		x	x		x		x		x	x	x	x	9
	HW/SW Co-design		x	x		x			x	x	x	x	x	9
	Debug/Verification	x	x		x	x				x	x			6
	NoC			x	x							x	x	4
	Low-Power Design			x	x		x		x					4
	RTOS	x	x										x	3
	#	3	5	5	4	4	2	2	3	4	4	4	5	

Debug and verification are obviously present in every real world SoC project. There are plenty of different tools for debug and verification that can be used when developing SoC. For example, logic analysers, waveform simulators, and debug systems for software. Low-Power design is one of the most important reasons to use SoC instead of general purpose computers and controllers. The low-power design is still separated from the platform design. Most issues of low-power design are not directly depending of the structure of the platform.

Network-on-Chip is also in important role in SoC. The structure of communication between components is important when the efficiency of the system is important. When there is heavy traffic in communication between components the structure of NoC is in important role. Real time operating system is in major role when building a reusable functions in MPSoC system. RTOS makes higher abstraction level for application layer. This abstraction makes possible to create hardware independent applications. RTOS is not necessary for the MPSoC function but without it the reuse would be very difficult.

The exercises are in big role when teaching the SoC. Most of the courses include some kind of exercise work but implementations are usually hidden behind a password. At least two other exercise works include the SoC system implementation into the FPGA chip. These courses are number 8 and number 11 in Table 1. Course number 8 includes the audio system project that contains the use of two different processors, one for control and another for hardware acceleration. The exercise work of course number 11 is a video encoder with one CPU and one HW accelerator.

4 KACTUS DESIGN ENVIRONMENT

Kactus2 is a computer program used to design embedded products. The main purpose for Kactus2 is to make FPGA easier for SW engineers. It also helps to packetize IPs for reuse and exchange [29]. In addition to SoC, It can be used to create hierarchical description of the whole product. The design created with Kactus2 is based on IP-XACT XML metadata that is used to ensure the unambiguous interoperability between different partners and tools [29]. Metadata is a formal description of the design. It includes the reference to component design files, but does not include the actual functionality. Kactus2 cannot be used to create IP blocks. Design blocks have to be written with HDL editors and software tools.

4.1 IP-XACT

IP-XACT is a standard for documenting of the metadata IP components for SoC in an extensible mark-up language (XML) format [30]. IP-XACT is standardized by IEEE and created by SPIRIT Consortium. An IP-XACT description is a set of XML documents referring to one another. IP-XACT includes a schema that is the core of IP-XACT specification. That schema defines a number of document types and semantic rules that describes the relationship of different documents. The most important document types are *design* document, *component* document and *bus definition* document [31].

Component document describes an IP component that can be instantiated in the design document. Components have a bus interfaces that are described in a separate bus definition document. Detailed interface description enables design automation, for example detecting and preventing illegal connections. There can also be hierarchical collection of IP blocks as a design for bigger IP components. IP-XACT component include following [29]:

- Identification and general information
- Views
- Associated files, tools and languages
- Ports and bus interfaces
- Parameters and configuration
- Addressing information.

Identification and general information is the VLNV identity that includes *vendor*, *library*, *name* and *version*. For example (TUT, ip.hwp.storage, fifo, 1.0). It is unique for all IP components. Views are used to represent different roles of the component, for example RTL implementation for synthesis and behavioural model for simulation. Files, tools and languages are associated for different views. For example, addresses may be associated for bus interfaces.

Design document represents the block diagram of the system. It includes component instances and bus connections between the components. It is like normal schematic of components [29].

The buses connect different IP components. Bus definition document describes the type of the bus that connects different components. Bus definition contains a signal interface and constrains for those signals. This includes signal names, directions, widths and types of signals [31].

In Figure 13 is a screen capture of an example design in Kactus2. There are interfaces and components connected with buses. Kactus2 helps with the connections denying wrong connections. The design looks clear and all multi bit bus connections are shown with single lines.

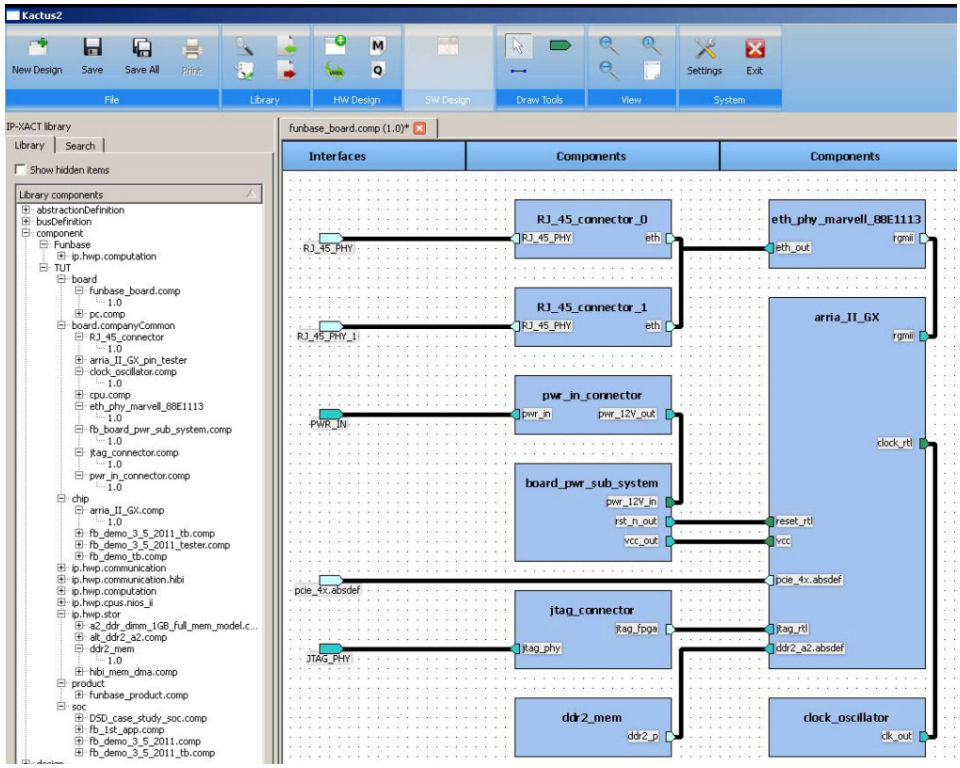

Figure 13: Screen capture of Kactus2. There are five inputs in interface connections and two levels of components. Interface connections are on the left side of designer area.

4.2 MCAPI

MCAPI provides a standardized API for communication and synchronization between closely distributed cores and processors in embedded systems [32]. The purpose of MCAPI is to capture the basic elements that are required for closely distributed embedded systems. It is both an API and communication semantic specification. MCAPI communication is based on *node* and *endpoint* abstractions. Node is a logical concept that can be a process, a thread, a HW acclerator, or a processor core. Each node can have multiple endpoints that are socket-like communication termination points. Endpoints are defined with a tuple <*domain*, *node_id*, *endpoint_id*>. Endpoints may have attributes, e.g. Quality of Service (QoS), buffers, and timeouts. In Figure 14 is an example of MCAPI structure.

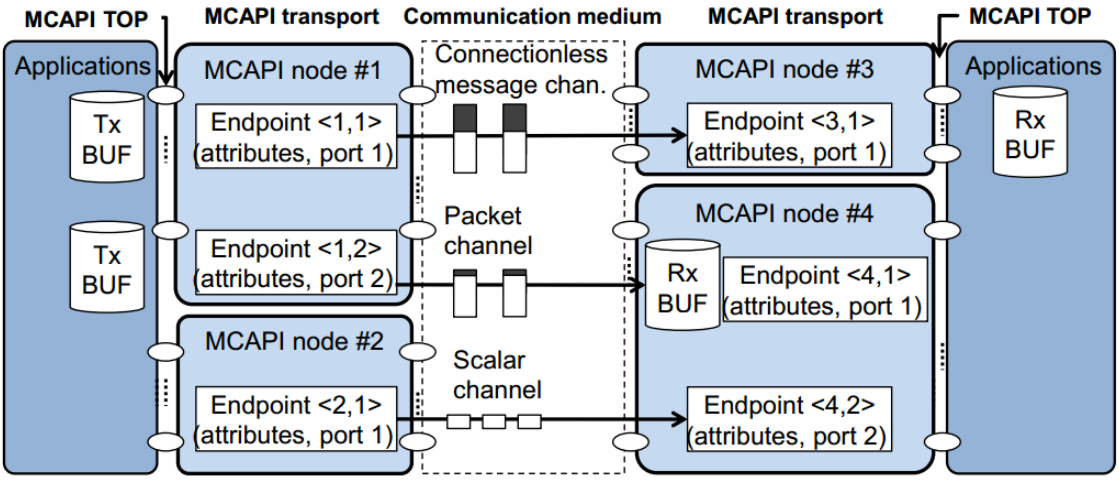

Figure 14: Example structure of MCAPI. Application APIs are in both sides of the figure and are connected to MCAPI. Nodes are on the MCAPI top interface, and endpoints define the connection between nodes. [29]

4.3 Kactus2 Design Flow

There are three main purposes to use Kactus2. First, it can be used to draft and specify product, printed circuit boards (PCB), chips, SoCs and IPs. Kactus2 stores created specifications in IP-XACT format. Second, it can be used to create MPSoC from created components. Third, it can be used to packetize IPs for reuse and exchange. These IPs can be imported to any IP-XACT standard compatible product. So, Kactus2 can be used to create templates and blocks from your IP components for library. [29]

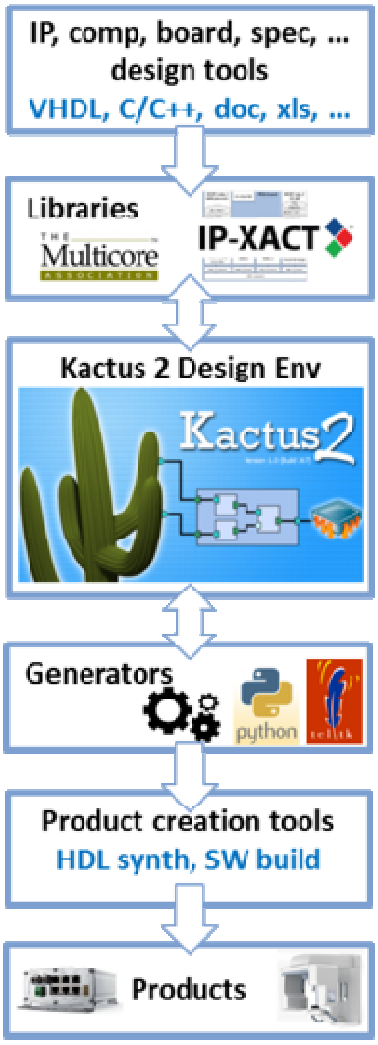

Figure 15: Place of Kactus2 in an MPSoC system design flow. Kactus2 is used to generate IP-XACT documented components and system design from different kind of IP components. These components can be used to create SoC products. [29].

Using different components and specification files Kactus2 is used to construct a system design for HDL synthesis and software build. This means that together with several other tools Kactus2 is used to create a final product. On this design flow Kactus2 replaces other design tools that could do the same design, for example block design tool of Quartus II. The part of Kactus2 in MPSoC design flow is shown in Figure 15. Kactus2 uses IP-XACT and Multicore communications API (MCAPI) libraries to create design of a system. With these standard specifications the system design can be constructed from different kinds of components and there are high abstractions from connections between components. Kactus2 cannot be used to create functionality of IP components but only standard interfaces of those and packetizing. The functionality has to be created with different design tools and is not used when designing with Kactus2. The final product can be generated from the design created with Kactus2.

5 SOC PLATFORMS EXERCISE WORK

The exercise work of the course TKT-3547 SoC Platforms introduces the concepts and design phases of an MPSoC platforms and applications. Realistic platforms and applications are very complex and require long time and large development team. Therefore, the course work must be lighter version that still highlights the most important concepts and design phases. The product of the work is the reaction game. It is played with buttons and LEDs. The game is played with four push buttons, which player tries to press in correct order in increasing game speed. The order to press buttons is indicated with LEDs. The amount of correctly pressed buttons equals the score of the game. Such a simple function is desired because the focus of the exercise work is in the platform and design layers, not in some specific application.

The topics of the course are listed in Table 3. The content of the lectures and the exercises are similar, but in different form and order. In the lectures, the topics are concerned wider but not as extensive as in the exercises. More of the topics of the exercises are discussed in chapter 6.2.

Table 3: Topics of the lectures and the exercises in the cource. [33]

Lectures	Lecture topics	Exercises	Exercise topic
1	SoC architectures	1	SoC specification
2	SoC design	2	Altera SOPC design
3	Parallel computing	3	IP-Block HW design
4	SoC interconnections	4	Driver design
5	HW dependent SW	5	Tasks and synchronization
6	RTOS and multitasking	6	IPC and messaging
7	Task scheduling and synchronization	7	Game design 1
8	Task communication	8	IP-XACT basic HW design
9	IP-XACT part 1	9	IP-XACT game HW design
10	IP-XACT part 2	10	IP-XACT SW design
11	Multiprocessing API	11	MCAPI design
12	Review	12	Game design 2

The exercise work is divided into two phases. In the first phase students get familiar with three layers of the system and the basic system design flow with the Alteras tools. The second phase teaches how to create reusable hardware and software components.

That is done with Kactus2. The multiprocessor platform gives experience for concurrent software developing beside the main objectives.

5.1 Objectives for the Exercise Work

The exercise work of the course TKT-3547 SoC Platforms gives the practical view from the main topics of the course and supports learning objectives of the course. The study guide of the Tampere University of Technology describes the learning outcomes of the course [34].

> "The student learns basic concepts of System-on-Chip and its division to hardware platform, software platform and application layers. Logical layers, standards and implementation of layers and interfaces are studied in detail. A practical view is given by exercises, in which a multiprocessor system is created on FPGA and used as platform for an example real-time application."

This means that students learn the structure and design flow of SoC. They learn different design layers of the MPSoC system and how different layers are actually connected. Students familiarize with the MPSoC levels from RTL to application level. The system of the exercise work shows the basic structure of MPSoC. It includes multiple processors and IP components that students have to create. Students also create software drivers for those own created IPs. The major objectives to teach in this exercise work are structure of MPSoC, abstraction layers, design reuse and HW/SW co-design.

5.2 Notes from the Former Exercise Work

The course had also an exercise work before this version [35]. The former exercise work had the same DE2 development board to implement the system. That system had Nios II processors connected to each other with HIBI network and the Avalon interconnect fabric. HIBI network was used for inter processor communication (IPC) and all peripherals were on the Avalon switch fabric. Figure 16 shows the basic composition of platform of the former exercise work. That system was quite good for teaching MPSoC platform. There were two processors communicating with each other via network. However, IP reuse was not covered. One important thing that this structure does not show for students is the reuse of IPs. All peripherals were by the system interconnect fabric and that design is not reusable on a system without system interconnect fabric.

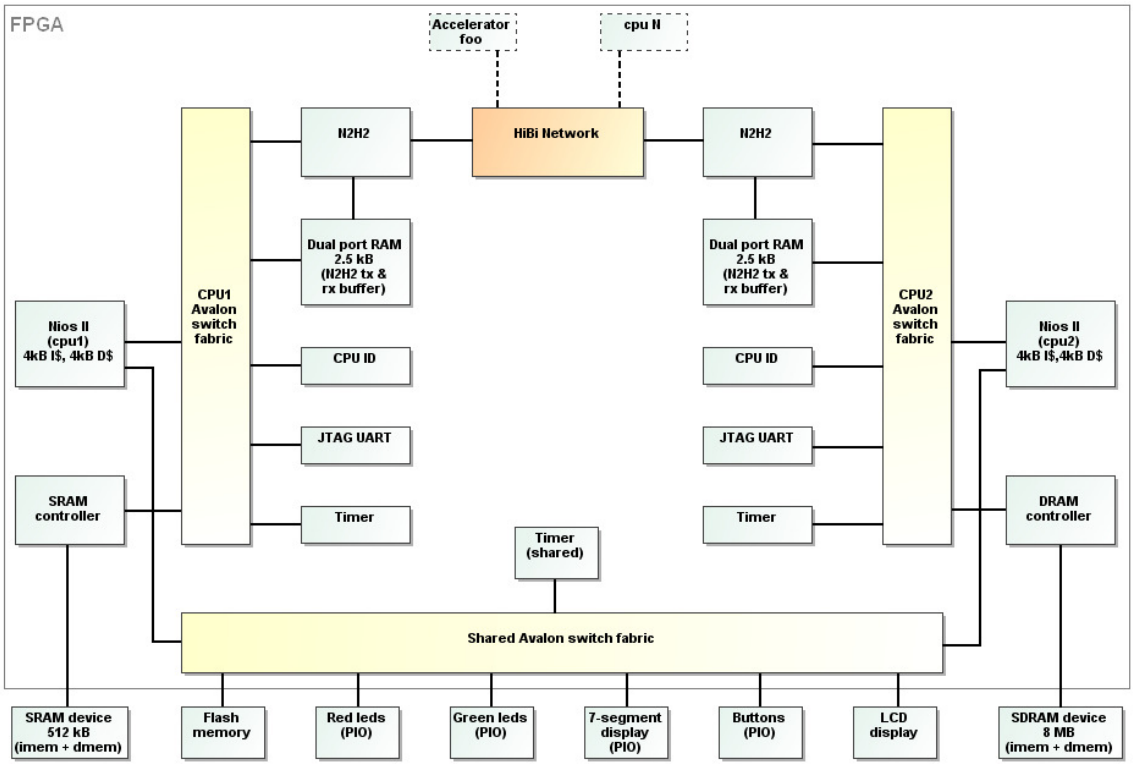

Figure 16: Platform of the former exercise work taken from first part of the exercise. It included most of the components that is also in the new exercise work. But as a hint of the future, there is IP blocks drawn connected to the HIBI network. Those components weren't the part of the actual former exercise work.

As described, the basis of the exercise work was quite good. The major problems were on the implementation of the work. Students didn't actually construct the platform of the system. It was created by the course personnel and given to the students. The construction of a hardware platforms wasn't practical because students studied it with documents and questions. That moved the focus of the exercise work towards the application. So, the biggest thing to change for the new work is to move the focus more to the HW platform and its composition. After this improvement the IP reuse and wider vision to the NoC can be handled properly in exercises. This makes possible for students to create their own reusable IPs.

Software platform used eCos operating system [36]. That was also given to the students by the course personnel. The students didn't configure the SW platform and didn't touch the application program interface. The SW platform was another black box component in addition to the platform.

Biggest things to improve from the former exercise work are shown in Figure 17. HW platform should be implemented on the exercises; functions for hardware should be put to the API; peripheral components should be done by using reusable design; and network functions should be in API as well.

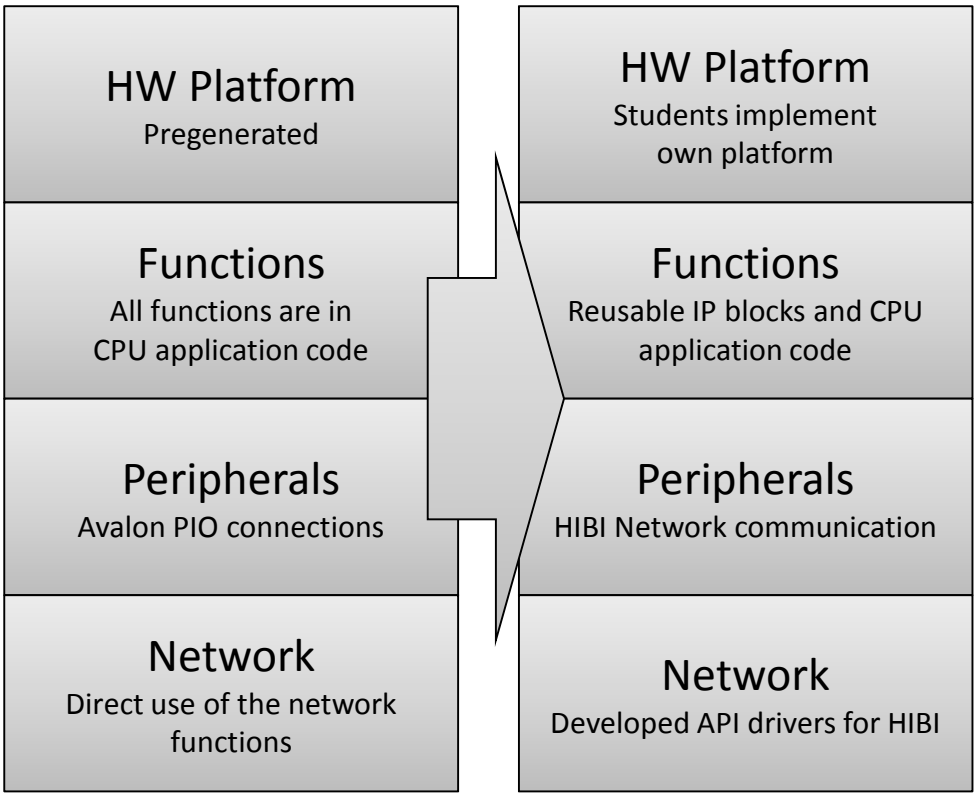

Figure 17: Main things to improve for the new exercise work. HW platform should be implemented on the exercises; functions for hardware should be put to the API; peripheral components should be done by using reusable design; and network functions should be in API as well. Avalon PIO means the peripheral I/O connections of the Avalon bridge which are replaced with HIBI and HW components.

5.3 Structure of the New System

During the exercise work, students learn the three layers of the SoC design described in the study guide: hardware platform, software platform and application layers. The hardware platform is a multiprocessor system, which is used to the base for the software platform. The application layers are the top layers of the system. These are the three layers of the system explained in the study guide (shown in Figure 18).

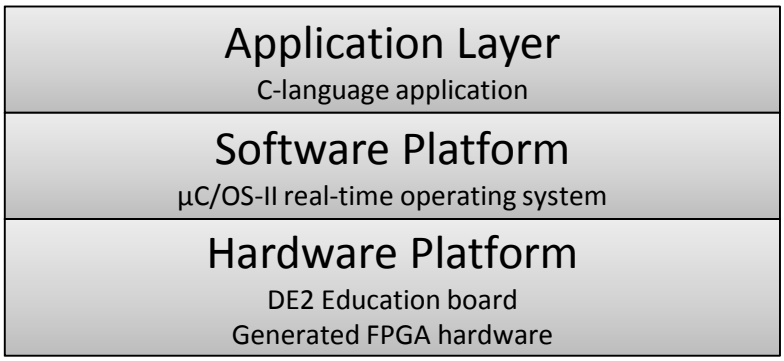

Figure 18: The three layers of the system implemented in the exercise work.

The hardware platform includes the physical components of the DE2 education board and hardware implementation that is synthesized for FPGA chip. Physical peripherals are also on the DE2 education board. This platform is selected to the exercise work, because the FPGA is fast and easy technology to create the system from a scratch. Because the Department of Computer Systems has already DE2 boards for education there is not easier way to implement MPSoC system on this course. FPGA chip of the DE2 board is Altera Corporations Cyclone II Altera distributes several designing tools and with the board. There are tools for all phases in this system design. All this makes easy to select DE2 and FPGA for the platform of the exercise work. The system that is implemented for FPGA chip includes the main functionality of the product. The main parts are microprocessors and IP blocks.

The software platform of this exercise work is the Micrium µC/OS-II real time operating system [37]. It includes all that we need for operating systems. There are several other operating systems that we could have chosen. µC/OS-II was selected for our RTOS because its version for Nios II is distributed with the Nios II Embedded Design Suite (EDS).

Application layer is software to be run in microprocessors. With custom made IP blocks these processors include the actual functionality of the product. IP blocks takes care for the driving of peripheral components, all other application functionality is in the codes for processors.

5.4 Hardware Platform

The hardware platform in the exercise work is generated into the FPGA circuit of DE2 education board. DE2 provides FPGA and the user interface needed to the system i.e. LEDs, buttons and 7-segment displays. The hardware platform is the set of architectures that makes possible to use these peripherals as a part of the system application. Nios II

Soft-core processors, IP blocks, and HIBI network are the components which are used to construct our hardware platform.

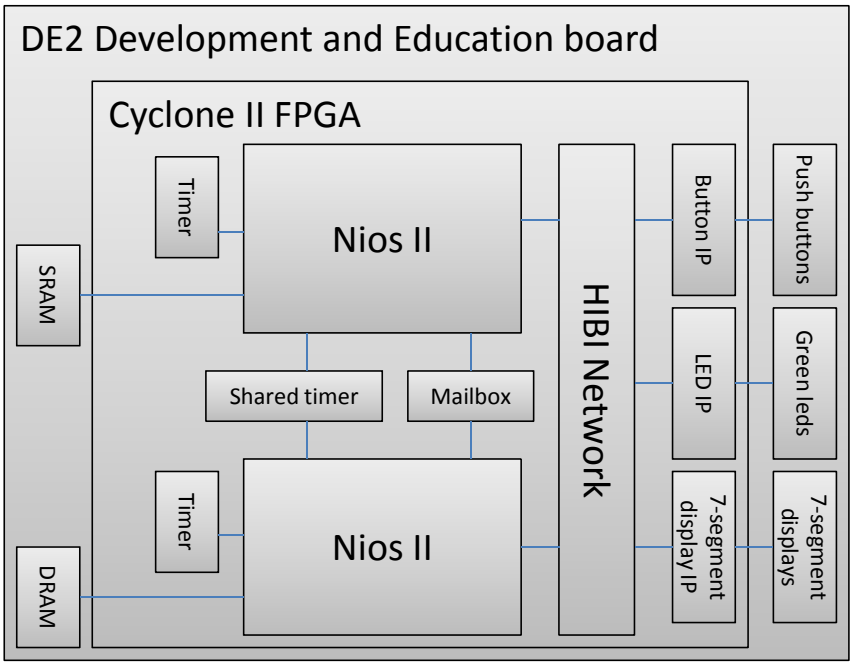

Figure 19: Structure of the hardware platform. Two Nios II processors and IP blocks for systems user interface is connected to the HIBI network. In the implementation made with kactus2, Nios II processors are different blocks and are not connected to each other via Avalon system interconnect fabric. That disables also shared timer that is drawn in the picture.

All of the components of the hardware platform, expect the external memories, are synthesized to the FPGA as well. The designs are written in hardware description language (HDL). Peripheral interface is connected to IP blocks. These IP blocks are the interface blocks that connects used peripheral components to the HIBI network. Processing units are used to create the functions of the system and use peripheral components via HIBI network. HIBI is used to communication of the final product, but in exercise work also Avalon mailboxes are implemented to give a comparative technique for IPC. Composition of the components used in hardware platform is shown in Figure 19.

There are shown three hardwired IP components, Nios II CPUs, HIBI network, memories for CPUs, timers and mailboxes. Components are reusable HDL designs that are included into the system. In the exercise work this system is constructed two times with different designer tools, first with Altera Quartus II block designer and later with Kactus2. Both designers are used to connect various reusable parts together to construct the complete HW platform. Quartus II block designer II is used in the work because it is simple tool and easy to use beside other Quartus II tools i.e. SOPC builder and programmer. Kactus2 is used because of its good reuse properties. It has better support for hierarchy and allows multiple interconnection types. This leads to easier and clearer

designing of the system, especially connections. Kactus2 is also used to packetize IP components for sharing and reuse. Both designers generate the same functionality and are used in different phases of the exercise work.

5.5 Nios II processor

The processors of the system are Nios II processors [16]. Nios II is used because of the ease of the design with free SOPC builder or Qsys software. The processors use static RAM (SRAM) and dynamic RAM (DRAM) chips as a program and data memory. Both of these memory chips are on the DE2 education board. All other components of the processors are synthesized in FPGA. There are three processor core configurations that can be used in Nios II. These are economy (/e core), standard (/s core) and fast (/f core). It is not necessary to force students to use some specific core, because all of them include all needed properties.

Both processors have their own memories for data and instruction memory. Processors have also own interval timers as system timers. There is also a shared timer for processors that is used as timestamp timer. Timestamp timer is used as a timing device when comparing time usage of different design solutions.

Processors are connected to HIBI network with DMA processing [38] for data transmission. There are also Avalon mailboxes for IPC. In this work, mailboxes are used only to compare the data transmission performance against HIBI. This shows the superiority of the DMA transmission against mailbox when transmitting larger data amounts. Students have to use HIBI to all data transmission of the game software in the exercise work.

5.6 IP Components

There are three peripheral devices that are used in the exercise work; four push-buttons, eight LEDs and two 7-segment displays. The components are connected also to physical components on DE2 and to HIBI Network. All of these would be easy to connect the processors as the parallel input/output (PIO) signals, but the designs would not be highly reusable. The better solution to include these peripheral devices into the system is make IP blocks to control them. IP blocks controls the peripherals and are connected to the HIBI. Blocks can be reused in the systems that include the HIBI. Moreover, any processor connected to HIBI can access them. Figure 20 shows the connections of IP block between HIBI network and peripherals. Students create packetize these components in the exercise phase 9 described in section 6.2.

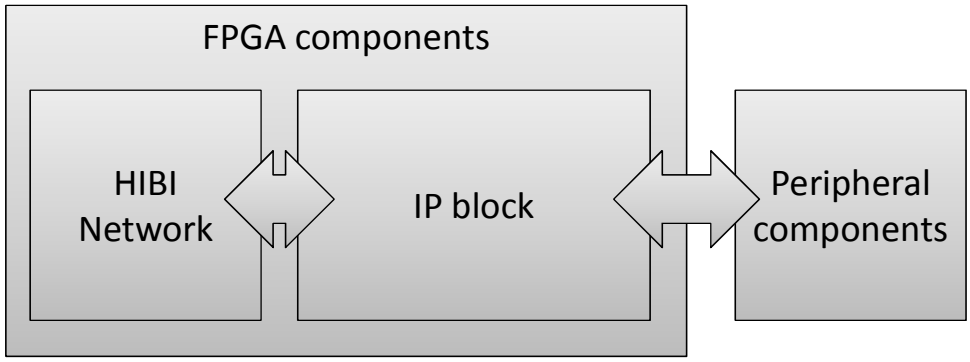

Figure 20: IP block connects the system to the peripheral components. For example IP block gets messages from network and controls the LEDs.

7-segment and LED components are similar. They can be used by sending them packet via HIBI network. For example: to light numbers in the 7-segment display would be as easy as sending one word to the component. IP for buttons functions to other direction. When a button is pressed the IP sends a packet to the destination component.

IP blocks are HDL components that can be used in Quartus II and Kactus2 as simple components that have interface to a HIBI wrapper. Students create drivers for HIBI PE DMA to Nios II hardware abstraction layer (HAL). Then the HIBI Network and IP blocks can be used as a file mode device.

5.7 Heterogeneous IP Block Interconnection

Heterogeneous IP Block Interconnection (HIBI) is a communication network for SoC. HIBI can be used to connect processors and IP blocks in the SoCs. It has an application independent interface to allow reuse components [39]. In this exercise HIBI connects all IPs and processors because this allows us to design reusable components for LED and button interfaces.

5.7.1 Network Topology

The network topology of the HIBI is not fixed. It can be built with the wrappers, bus segments and bridges. The topology of our system is shown in Figure 21. Our implementation of HIBI network includes only one bus segment. That is the simplest form of the bus and is good in our case. More segments could be connected to the bus with bridge components to construct a hierarchical bus. Use of multiple bridges increase latency, but multiple segments allows parallel transmissions in different segments. HIBI network includes a wrapper for every IP block. Wrapper connects the IP to the network. Wrapper follows the traffic of the segment and can act either as a slave or as a master. Masters can initiate transfers and slaves can only response to the transfer. Each wrapper has an address region that can be used to receive data. Different addresses in that region

can be used to separate different kind of transmission. Each address can be used as a channel. For example, data from buttons pressed can be sent to one channel and data from a processor can be sent to another channel.

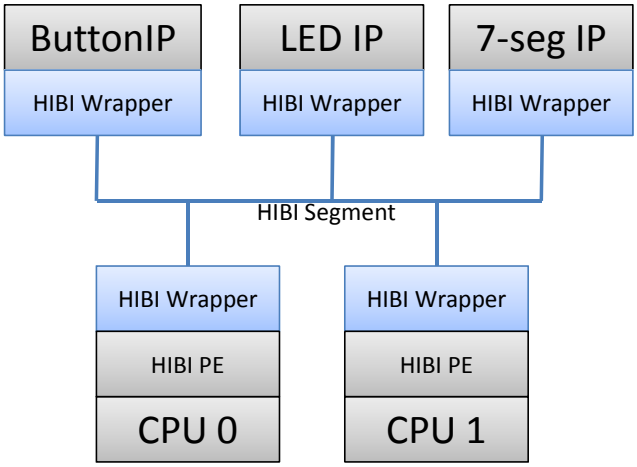

Figure 21: IP blocks and processing units connected to HIBI network.

5.7.2 IP interface

There are two FIFO buffer memories in each wrapper, one for data to transmit and another for received data. Each IP component controls the data transmission by reading and writing those two FIFO buffers. Figure 22 illustrates the logical steps of the transmission procedures. Before sending word to network, IP have to be sure that the Tx FIFO is not full in the connected wrapper and before reading data from network, IP have to be sure that Rx FIFO is not empty.

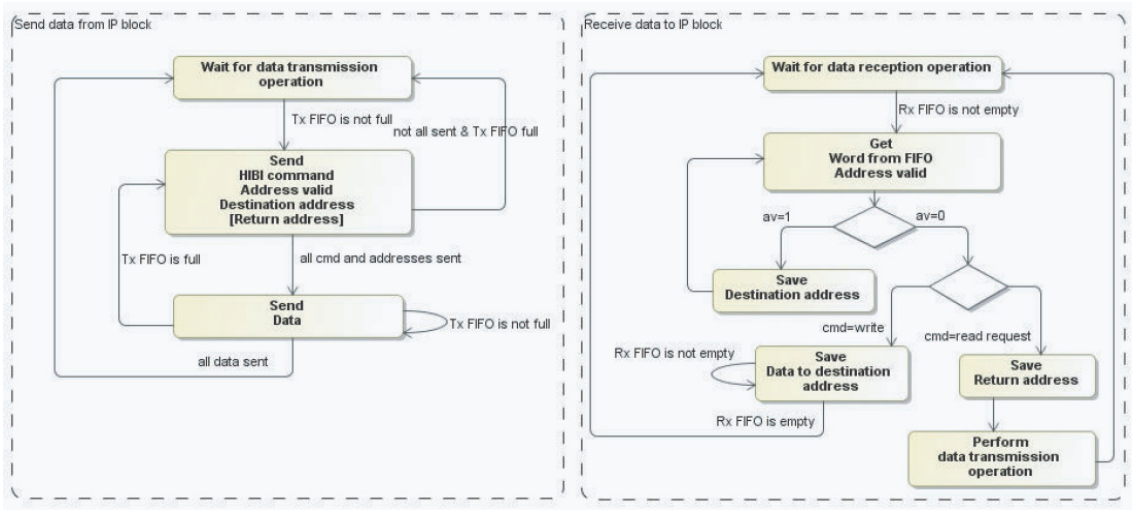

Figure 22: Logical flow of sending and receiving data to HIBI. [39]

Figure 23 and Figure 24 shows example timing of the signals used in transmitting data between an IP and a HIBI network.
When sending data to network, IP block has to wait that buffer is not full. If that buffer is full IP block have to wait that the wrapper sends earlier words to the network. When buffer is not full the IP can write a word by setting command and data signals correct and raising WE signal high as a start of transmission.

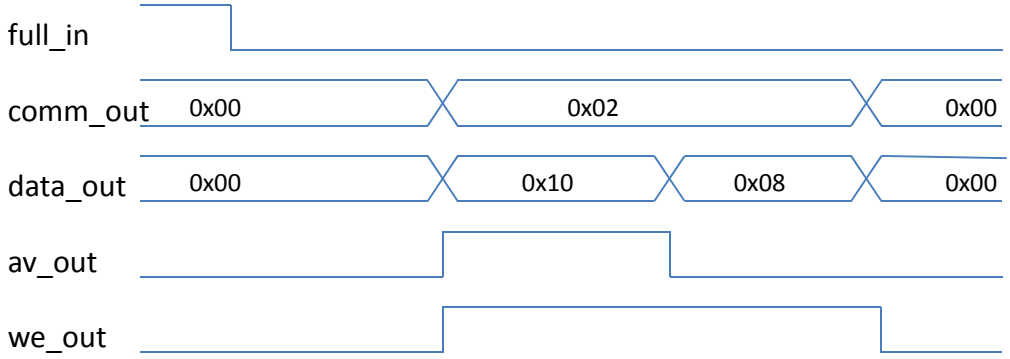

Figure 23: Needed signals between IP and wrapper in send operation and example timing of sending word 0x08 from IP to HIBI address 0x10. Full_in shows if the buffer is full and cannot accept new data. Comm_out is the operation command (0x02 is command for send). Data_out is the data to be sent. Av_out is address valid signal that shows it the data to be sent is an address. We_out will be set to start the transmission. Wrapper controls the full_in signal and IP controls all other signals.

The reception of data is also controlled with the signals between IP and wrapper. IP have to wait that there is a data to receive. Then IP checks that the command and data is correct and receives the data by setting read enable (RE) signal high. The wrapper receives to the FIFO only the data for correct HIBI address space. IP have to check that the HIBI address is correct before reading data from wrapper.

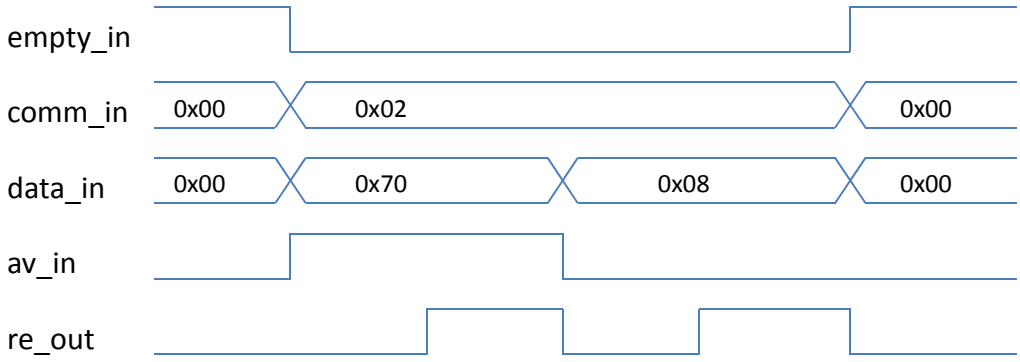

Figure 24: Needed signals between IP and wrapper in receive operation and example of timing in receive of word 0x08 from HIBI network. When empty_in, comm_in, data_in and av_in are desired IP can receive the data by setting re_out signal high.

5.7.3 HIBI Processing Element DMA

HIBI PE DMA allows connecting processor systems compatible interface to the HIBI network. Processing element is connected to HIBI PE DMA with Avalon memory mapped interface (Avalon-MM). Also a dual port memory is needed between processor and DMA. DMA is connected to the HIBI network with a wrapper. Processing element is connected directly to the HIBI PE DMA to access its registers. Data is transmitted via dual port memory which is used as a buffer. The structure of the HIBI PE DMA and connections to processor and HIBI is shown in Figure 25.

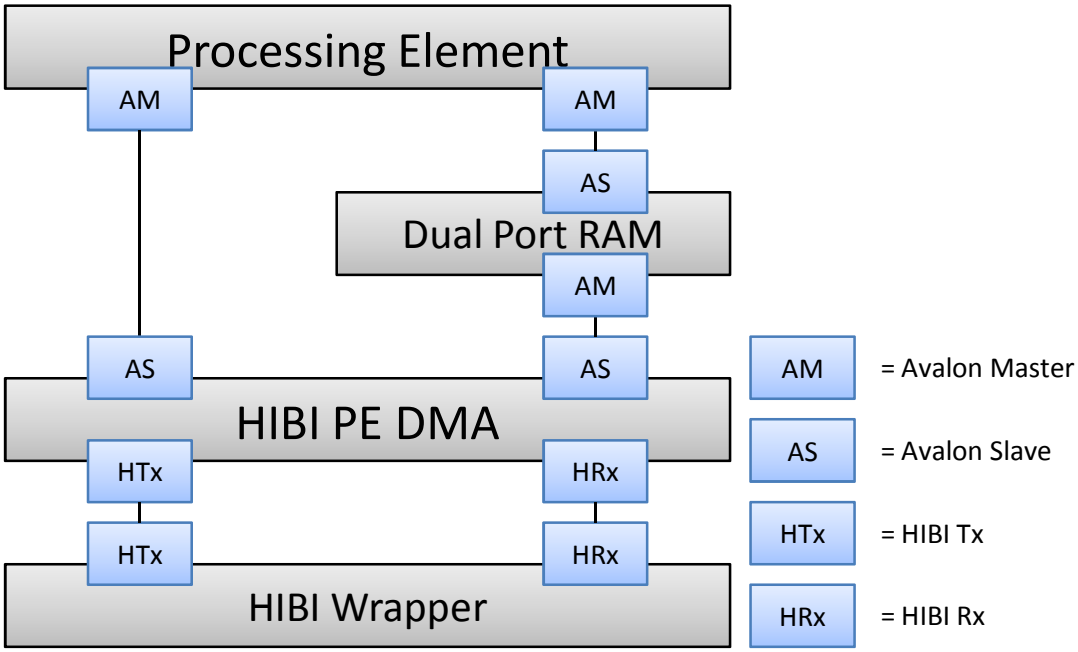

Figure 25: HIBI PE DMA connects the processing element to the HIBI wrapper. Dual port memory is used to store the transmitted data.

HIBI PE DMA uses either packet or stream channels for transmission. In this exercise work we use it with packet mode. There is a C language drivers designed for the processor. Drivers are the set of pre-processor macros which are used for the HIBI PE DMA and HIBI wrapper configurations and commands. With these macros the desired amount of packets can be sent to another IP.

Multiple channels can be used to receive data with HIBI PE DMA. All channels have to be initialized to receive the packets. Amount of incoming data have to be known when initializing a channel to receive. Data reception can be done either polling the registers or using interrupts.

5.8 Processor Design

Processors of the exercise work are processing elements which are designed with the Altera SOPC Builder tool (Figure 27). Both two processors contain the components shown in Figure 26. Processors in the system are Nios II processors. External SRAM and DRAM are used as data and instruction memories. These two memories are located on the DE2 education board and Altera offers the memory controllers for both types of the memories. Use of external memories leaves more FPGA logic elements for other functionality. JTAG UART is used to ease the software development. It is used as a character device to make possible to send text to connected terminal.

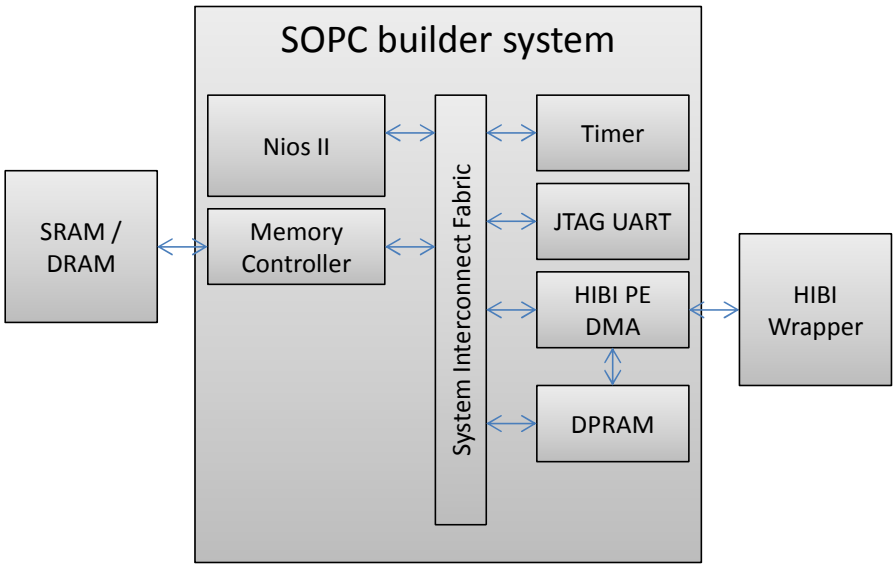

Figure 26: Composition of the SOPC builder system in the exercise work.

Use	Connections	Module	Description	Clock	Base	End	IRQ	Tags
☑		⊟ **cpu_0**	Nios II Processor	[clk]				
		instruction_master	Avalon Memory Mapped Master	**clk_0**				
		data_master	Avalon Memory Mapped Master	[clk]			IRQ 31	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	**0x00000800**	0x00000fff		
☑		⊟ **sdram_0**	SDRAM Controller	[clk]				
		s1	Avalon Memory Mapped Slave	**clk_0**	**0x02000000**	0x027fffff		
☑		⊟ **timer_0**	Interval Timer	[clk]				
		s1	Avalon Memory Mapped Slave	**clk_0**	**0x00000020**	0x0000003f	0	
☑		⊟ **jtag_uart_0**	JTAG UART	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	**clk_0**	**0x00000010**	0x00000017	1	
☑		⊟ **sysid**	System ID Peripheral	[clk]				
		control_slave	Avalon Memory Mapped Slave	**clk_0**	**0x00000000**	0x00000007		
☑		⊟ **mailbox_0**	Mailbox					
		s1	Avalon Memory Mapped Slave	**clk_0**	**0x00000060**	0x0000006f		
☑		⊟ **mailbox_memory_0**	On-Chip Memory (RAM or ROM)	[clk1]				
		s1	Avalon Memory Mapped Slave	**clk_0**	**0x00002000**	0x00002fff		
☑		⊟ **mailbox_1**	Mailbox					
		s1	Avalon Memory Mapped Slave	**clk_0**	**0x00000070**	0x0000007f		
☑		⊟ **mailbox_memory_1**	On-Chip Memory (RAM or ROM)	[clk1]				
		s1	Avalon Memory Mapped Slave	**clk_0**	**0x00003000**	0x00003fff		
☑		⊟ **cpu_1**	Nios II Processor	[clk]				
		instruction_master	Avalon Memory Mapped Master	**clk_0**				
		data_master	Avalon Memory Mapped Master	[clk]			IRQ 31	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	**0x00000800**	0x00000fff		
☑		⊟ **sram_1**	SRAM/SSRAM Controller	[clock_reset]				
		avalon_sram_slave	Avalon Memory Mapped Slave	**clk_0**	**0x00200000**	0x0027ffff		
☑		⊟ **timer_1**	Interval Timer	[clk]				
		s1	Avalon Memory Mapped Slave	**clk_0**	**0x00000020**	0x0000003f	0	
☑		⊟ **jtag_uart_1**	JTAG UART	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	**clk_0**	**0x00000010**	0x00000017	1	
☑		⊟ **common_timer**	Interval Timer	[clk]				
		s1	Avalon Memory Mapped Slave	**clk_0**	**0x00000040**	0x0000005f	2	2

Figure 27: Screen capture from SOPC builder tool. Components are connected with Avalon bridge.

Avalon system interconnect fabric connects the read, write and data signals. Each connection to interconnection fabric is either master or slave. Each bus master requests control from an arbiter. The arbiter controls that two bus masters do not drive the bus simultaneously.

5.9 Design with Quartus II

Design of the hardware platform of the exercise work is done with two system designing tools. First of the designs is done with Quartus II design tool. Quartus II gives the design tools to implement the HDL system into FPGA chip. The components are in the design as blocks and are connected to each other with signals (as in Figure 28).

Nios II processors can be in same SOPC component or different SOPC components. The design is little clearer if processors are in different components and it makes system simulation simpler. On the other hand the Avalon mailboxes can be used only if the processors are in common SOPC component. This makes possible to compare the performance of HIBI bus and Avalon mailboxes, so we leave both processors in same SOPC component. Also button and LED IPs can be together or separated. There is no difference between these implementations because button IP only sends data and LED IP only receives data from HIBI bus. Again, it might be clearer if we separate these IP blocks.

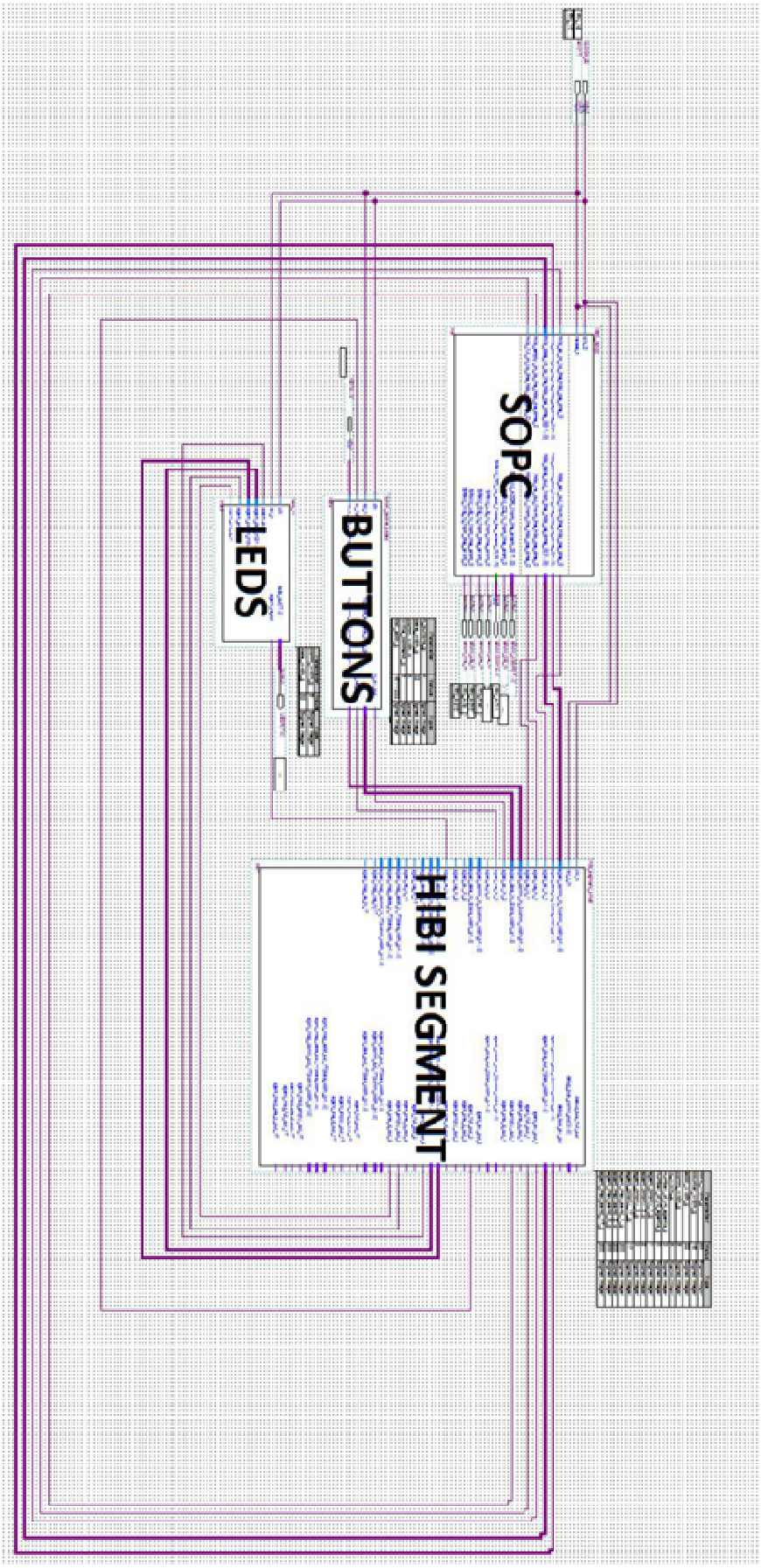

Figure 28: Screen capture from the Quartus II block designer. As can be seen, the signals are complicated when there are lots of components.

5.10 Design with Kactus2

Hardware platform will be also designing with Kactus2. In Kactus2 the designing of system is easier if all components and buses are already defined. The components are simple to connect and whole system looks clear, as can be seen from Figure 29. It shows an example design in Kactus2 designer window. Due to detailed interface descriptions the connections are shown with single lines that represent predefined buses. Interface descriptions also prevents from making illegal connections. The function of the system is same as made earlier with Quartus II block designer.

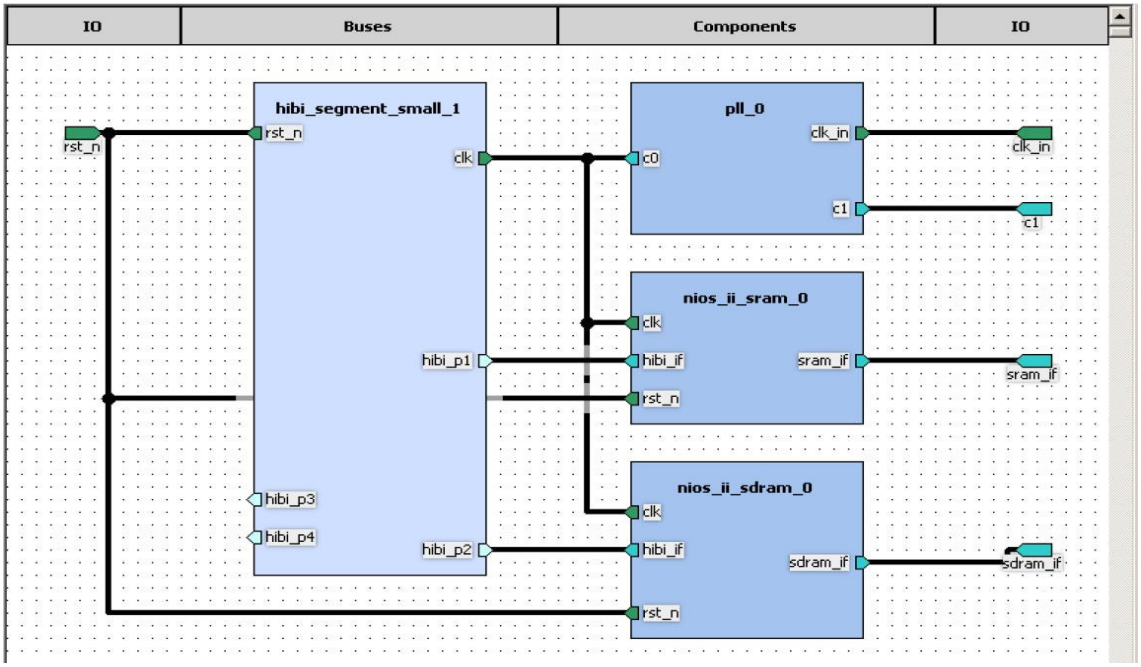

Figure 29: Screen capture of Kactus2 design layout. There are five I/O pins, HIBI network component, PLL, and two Nios II processors.

5.11 Micrium µC/OS-II

The software platform of the system is the Micrium µC/OS-II real time operating system which is delivered with complete ANSI C source code. It was chosen as it is delivered with the Nios II software build tool (SBT), because it is easy to configure, and Micrium also offers free licensing for universities. µC/OS-II includes tasks that we use in our software. It also allows multi-threaded programming. µC/OS-II has also interrupt requests (IRQ) but we use the interrupts which are in Nios II HAL. The memory footprint of the µC/OS-II can be scaled between 5 Kbytes and 24 Kbytes.

5.12 Driver Development

Device driver development is an important thing in HW/SW interface design. Driver is the software between hardware platform and application layer. That makes possible to generalize the application software for different kind of hardware implementations and makes possible to create software without knowing the structure of hardware. So, it increases the abstraction level of the device usage and makes possible to create hardware independent software applications.

In this exercise work the drivers are developed so that they utilize HIBI PE DMA. The drivers to be developed are HAL API device drivers and function as a file mode device. The driver functions are embedded to the Nios II HAL and driver interface functions that are not implemented are inherited from HAL API. Driver interface is shown in Figure 30. There are seven function prototypes for different API functions. The functions that should at least be implemented for IP components are *open(), close()*, *read()* and *write()*. These are the UNIX-style POSIX API functions. The open-function is called when the HW is wanted to use. A call to open() creates a new open file description. Read() and write() functions are used to the data transmission to the file device.

```
typedef struct {
   alt_llist    llist;     /* for internal use */
   const char*  name;
   int (*open)  (alt_fd* fd, const char* name, int flags, int mode);
   int (*close) (alt_fd* fd);
   int (*read)  (alt_fd* fd, char* ptr, int len);
   int (*write) (alt_fd* fd, const char* ptr, int len);
   int (*lseek) (alt_fd* fd, int ptr, int dir);
   int (*fstat) (alt_fd* fd, struct stat* buf);
   int (*ioctl) (alt_fd* fd, int req, void* arg);
} alt_dev;
```

Figure 30: Nios II HAL driver functions for file mode device.

The drivers are automatically initialized to Nios II HAL with the *alt_sys_init()* function during boot. It is an automatically generated function that calls INIT and INSTANCE macros for each component found in the hardware design [40]. Automatic generation requires that the driver implementation is in correctly named files that are in certain file structure.

Custom device driver needs hardware access macros and HAL function files included to the component file system. Hardware access macros are in a *_regs.h*-ending file. That file includes the entire hardware interface used in the driver codes.

HAL functions are implemented to the HAL files *.h* and *.c* files. HAL functions for file device are listed in Figure 30. These are the files where the POSIX API functions are implemented.

The easiest option for the directory structure for the driver is following:

The IP is located <*my_design*>/IP/<*component_folder*>.
The device driver files are placed to the component folder as follows:

- /inc/<*component*>_regs.h
- /HAL/inc/<*component*>.h
- /HAL/src/<*component*>.c.

5.13 Application Layer

Application layer is the software that is written in C and runs on Nios II processors. In this exercise the main application is the reaction game. The application is divided for multiple processors and multiple tasks. The game is so simple that it does not actually require the performance of multiple processors, but simplicity is advantageous when multiprocessor programming is taught in short time.

In the exercise work the application has to be programmed with two processors. One that runs the game logic and another that uses the HW components. µC/OS-II tasks are free to be used in both processors but are not required. HIBI is used for all communication between processors and hardware components. HIBI network is used with functions in the self developed driver.

The software is easiest to create with Nios II software build tool (SBT) that is distributed by Altera Corporation. µC/OS-II for Nios II is also distributed with Nios II SBT. Application programs can be debugged with Nios II terminals (see Figure 31) that are connected to the board via Joint Test Action Group (JTAG) test and development connection. In the figure, the application of the right-hand side terminal sends eight data values (from 20 to 27) to the left-hand side application. These are received to the interrupt channel 0. The left-hand side application also receives the messages from button IP that are received to interrupt channel 1.

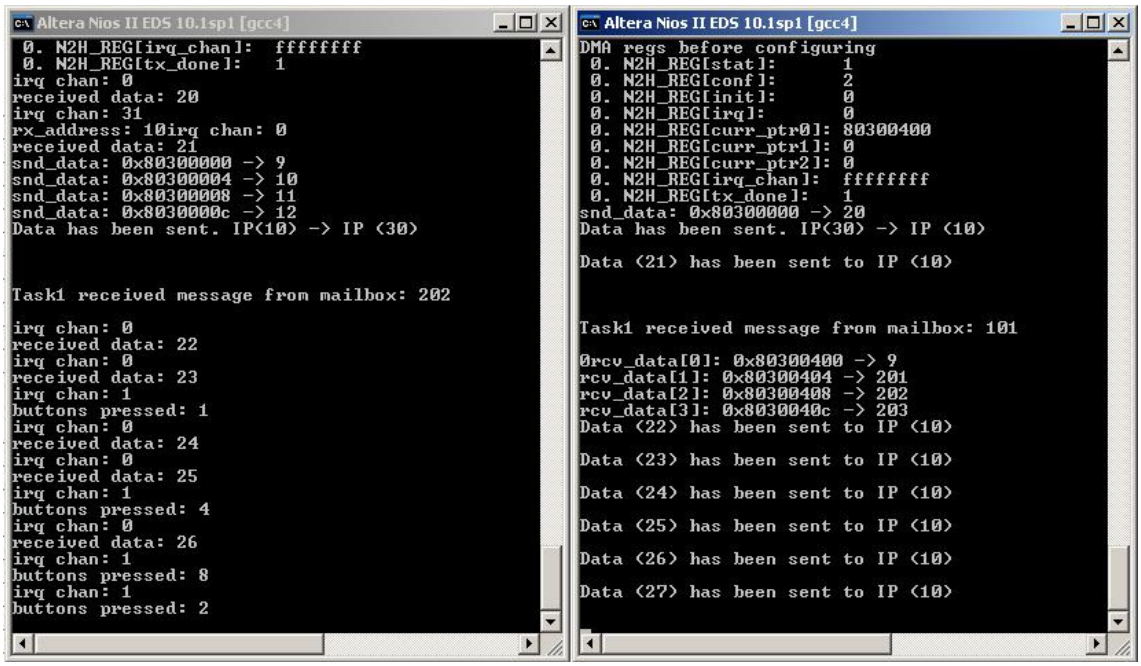

Figure 31: Screen capture from two Nios II terminals. Nios II terminals can be used as a character devices from the applications. It can be used for debug and development. In this view right-hand side process sends data to another process. These are received to IRQ channel 0. Left-hand side process also receives two messages from button IP (IRQ channel 1). Both processes also send and receive a mailbox messages (data values 101 and 202).

6 RESULTS AND ANALYSIS

The result of this Thesis project is not directly the MPSoC product but the specifications for exercise work to create MPSoC. The most important topics are considered in this chapter. Here is also discussed the phases of the exercise work.

6.1 Topic comparison for the Exercise Work

To decide what to include to the exercise work it is important to consider worth of the different phases and possible topics. Biggest topics and alternatives are listed in Table 4. The topics listed are either the topics included to work or alternative solutions. The figures of benefits and workloads are for the students.

Table 4: Benefits and workload of different topics. Topics are divided to platform structural topics and topcis related to multiprocessor programming. Table includes the benefit, workload, and their ratio of all topics. Column "used" shows if the topic is taken with to the exercise work.

Topic	Benefit	Workload	Benefit/Workload	Used
Structure of MPSoC system				
Quartus II design	3	2	1.5	x
Kactus2 design	4	3	1.3	x
Communication via NoC	5	3	1.7	x
Communication via mailboxes and direct peripheral connections	1	1	1	
Processing unit design	3	1	3.0	x
Pre-generated processing units	1	0.5	2.0	
IPs for peripherals	5	3	1.7	x
Avalon peripherals	2	2	1.0	
Multiprocessor programming				
Compare of communication techniques	2	1	2.0	x
Threads	2	2	1.0	x
µC/OS-II implementation	2	0.5	4.0	x
eCos implementation	2	3	0.7	
Driver to API	4	2	2.0	x
Normal functions for HW	2	1.5	1.3	
Real-life application	7	5	1.4	
Reaction game application	3	2	1.5	x
Average	3.0	2.0	1.69	10/16

The benefits are estimated to show the significance for the students learning. The figures of workload are also for students. The benefits of the topic usually grow when workload grows. For example, the workload is high if students create some part of the system completely but then they also learn it completely. The benefit/workload –ratio tries to make easier to compare this kind of problems. The topics taken to the exercise work are considered to give the best benefit with the workload that can be bear. For example, some real-life application would give much more benefit than reaction game application, but the workload would be too high for the time limits of exercise work. So, the selected application is the reaction game.

The amounts of worth and workload are approximated analytically. So, comparison of all figures is difficult because of the difference of their nature. E.g. the benefits of µC/OS-II are hard to compare against the benefits of Quartus II design. The different topics should be compared to similar solution. E.g. reaction game application should be compared to other applications. Another way to think the worth and workload is to analyze if the topic is even worth teaching.

The benefits and workloads of designs with Quartus II and with Kactus2 have to be evaluated. Which one is more valuable and why should we use both. Design with Quartus II is easier than Kactus2 when the components of the whole system are not already created. Most of the students have also some experience of Quartus II from prerequisite courses. The workload of Quartus II design should be six hours at maximum. That means that it should easily fit in the time reserved for one week. The benefits in using Quartus II are easy design method and fast implementation of the system top level.

The workload of Kactus2 is bigger than Quartus II when designing a system this small. Different component and bus definitions take time and are used only in this one system. Use of Kactus2 is still reasonable, because its benefits are bigger than in Quartus II as well. Main benefit is in the design of reusable and hierarchical systems. These reasons show that even these design tools result the same system, they have different reasons to take along to the exercise work. Both of these design tools have much worth and could be both used in this exercise work.

In large real-life products the communication is done via network on chip. That is the reason we want to use it in this work as well. NoC as IPC can be compared to Avalon mailboxes. Network is much faster than mailboxes when transmitting large amounts of data. When we include NoC to the system we get also the benefit of NoC reuse properties. IPC and peripheral communication can be done via NoC. If IPC would be done via mailboxes the peripheral connections have to be created separately. That would lead to single-use design. So, even if NoC takes little more time to implement, the benefits lead us to select it as the method for communication.

Processing units are essential components of the MPSoC. In former exercise work the processor design was pre-generated and given to students. That saved little time but left hidden important part of the system. The implementation of the processors in SOPC builder system should take four hours at maximum, so there are really no reasons to leave that out of the work. In this case students can reproduce and modify the whole design in their other projects as well.

The RTOS to be used is also considered. The features we need from RTOS are included in almost all operating systems. So, the requirements of the RTOS are not in the major role. The easiest RTOS to include to the system is µC/OS-II as it is delivered with Nios II SBT. The reference system we get from the former exercise work, which was eCos. Because both of these operating systems include all we need, the µC/OS-II is chosen to get less workload.

The selection of the application is probably the most difficult part to compare. The best application to multiprocessor systems would be the application that actually requires the multiple processing units. That kind of application would be best to motivate the MPSoC platform. The problem of this kind of application in exercise work is the workload. It would need much familiarization and work. In the short exercise work the solution has to be light but still include the purpose for multiprocessor programming. This dilemma leads to think possible applications that require reasonable amount of work but would still motivate the MPSoC. The reaction game is the application selected to the work, because it is easy to understand but still can be divided for multiple processors. This is the part of exercise work that can be improved in future.

6.2 The Phases of the Exercise Work

The exercise work is divided to twelve different parts that are listed in Table 5. The instructions of all exercises are in appendix 1. These exercises have all own deadlines. One exercise has at least one week time to be finished. The exercise work is divided by the topics that takes different time to do. Some of the exercises need more work than others, which is the downside of the splitting. Sometimes weekly workload is high and sometimes low. The reason why the exercise is split is still the spreading of workload. Without several deadlines some of the students would leave the work to the last weeks. So, ultimately this arrangement serves the students. The exercise work will be done in groups of two students to spread the workload.

Table 5: The parts of the exercise work.

#	Topic	Tasks	Time / hours
1	SoC Specification	SoC specification writing; Introduction to topics of the exercise work	14.1
2	Altera SOPC Design	Get familiar with Altera design flow	3.3
3	IP-Block HW Design	Creation of IP blocks; vhdl block verification	6.0
4	Driver Design	Drivers of HIBI network to HAL	15.3
5	Tasks and Synchronization	Using multiple tasks; application logic implementation	6.5
6	IPC and Messaging	Comparing usage and performance of different messaging methods	6.8
7	Game Design 1	Implementation of game design specified in exercise 1	11.6
8	IP-XACT Basic HW design	Getting familiar with Kactus2	2.0
9	IP-XACT Game HW Design	IP packetizing; HW platform implementation with kactus2	3.8
10	IP-XACT SW design	SW packetizing; Getting familiar with Kactus2 SW design features	2.1
11	MCAPI Design	Getting familiar with Kactus2 MCAPI design features	1.5
12	Game design 2	Final game design with the platform created with Kactus2	3.5

First exercise is to generate the specification of the SoC system. Students create the specification of their plan for the product. This exercise gives big picture of the upcoming tasks of the exercise. Students create their plan according to a given user requirements. Students are encouraged to create unified markup language (UML) diagrams to the specification. This exercise gives already some understanding of the parts of MPSoC.

Main purpose of the exercise two is to get familiar of the Altera design flow. Students generate two Nios II processors and create a hello world program for them. In this exercise, students use different tools to create the system and get familiar with the design flow described in section 2.4.2. They create CPUs with NIOS II and create the design of the system with Quartus II block design tool. This design is compiled to the DE2 board and hello world software is implemented to the CPUs with the Nios II embedded design suite. It is important that students get a working template system running on FPGA already in the beginning.

In exercise three, students create their own IP components in HDL to use the peripherals via HIBI. This structure increases the amount of reuse in this exercise work and refreshes students HDL design skills. All of these components can be used also in different system that is implemented in HDL. System generated in exercises two and three is shown in Figure 19 on page 25.

In the exercise four, the hardware component abstractions are made for the CPU applications. Students write the driver functions for the HIBI network and IP components. The functions are written as the file mode devices that can be used to control IP blocks over HIBI network. The functions use PE DMA interface. The development of the drivers is handled more in section 5.12.

In exercises five, the game is created with one processor and several tasks. These tasks implements different functions that have to be synchronized to work together as a game. Four tasks is minimum in the exercise and students can implement the communication between the tasks as they want. The students are advised to create four tasks, one for game logic and one for each peripheral device. Students learn some of the features of µC/OS-II such as tasks, mailboxes, and message queues. This exercise sets the stage for the multiprocessor programming. The final game application is done with two processors in exercise seven.

Exercise six contains the inter-processor communication with HIBI network and with Avalon mailboxes. These two methods are compared with different amount of data and different packet sizes. The difference in performance between HIBI and Avalon mailboxes can be seen in reference implementation of the system in section 6.4.

In exercises from eight to twelve the system is created again, but now with Kactus2 design tool. In this phase the platform is done with Kactus2 software and it teaches mainly reuse. The design created with Kactus2 is more abstract than the design created with Quartus II. The signals between components are also abstracted. So, the composition of system couldn't be easier. In exercise eight the SoC is designed again. Its purpose is to get familiar with the Katcus2 design software with the basic design (see Figure 13).

In exercise nine, students create IP-XACT components for their own HDL designs. These components are packetized with the component editor. These components are integrated to the design created in exercise eight. In exercise ten, the software is packetized with Kactus2. Applications and software platform is packetized into IP-XACT metadata objects and SW are mapped to the underlying HW platform. This way also two SW is better reusable and documented.

The exercise eleven handles the MCAPI and Kactus2 MCAPI design features. MCAPI is an alternative way to implement communication between processing elements via HIBI. Students create MCAPI endpoints for processing elements and map them to the HW IP blocks.

Finally in exercise twelve, students create the whole system with hardware implemented with Kactus2. The hardware and software is already created in previous exercises, so the task of this exercise is simply put them together.

6.3 Time Consuming of the Exercise Work

The reported total time to complete this exercise work was approximately 75 hours. It is the time used by the groups of two, so the work of one student is smaller. The workload is little too big because it is planned to be approximately 60 hours on a 5 credit unit course. That is 5 hours/exercise for 12 exercises. Completion times of different exercises are listed in Table 5 and in Figure 32. There are average times spent from groups in first implementation of the exercise work in spring 2012. Figure 32 shows that exercises two, four and seven are significantly more time-consuming than other exercises. That imbalance can be easily equalized by dividing these exercises for two exercise weeks.

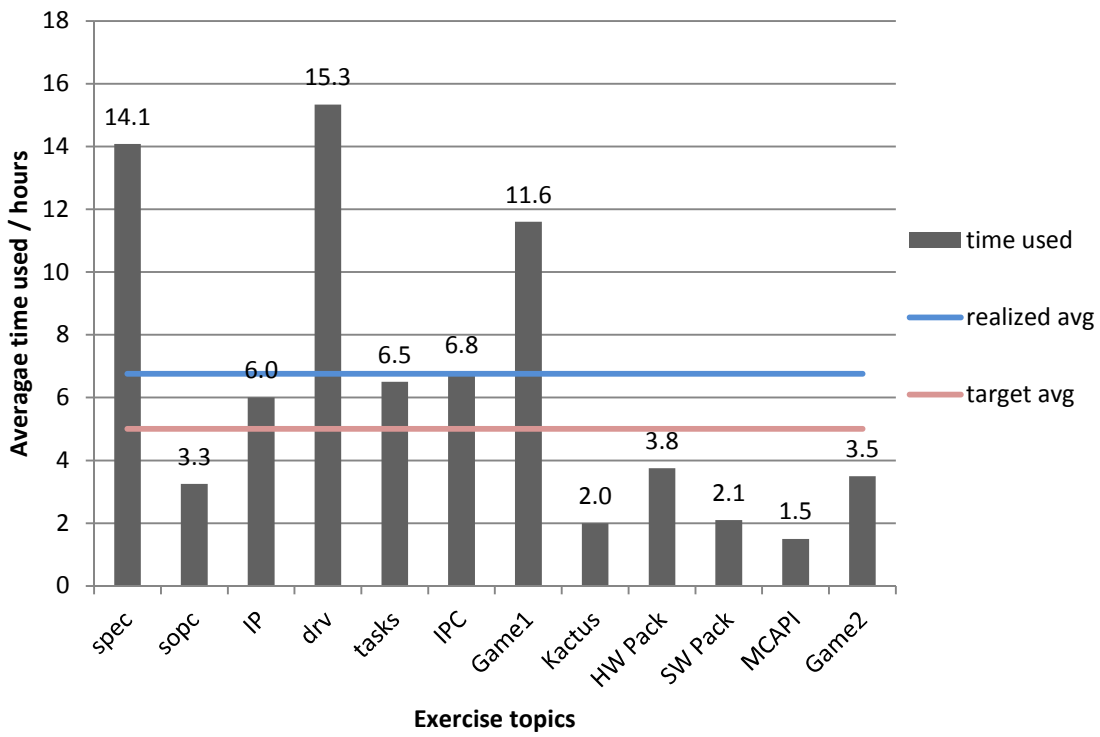

Figure 32: Time usage of different phases of the exercise work. Realized average time for exercise was 6.8 hours and targeted average is 5 hours.

In Figure 33 is weekly average time used for the exercises. The planned amount is 5 hours per exercise, so two groups managed to complete the exercises faster than that. Other groups used time over the planned time and one group used double time compared to 5 hours average. To decrease the average time on future course implementa-

tions, the exercises should be changed slightly. The options are to modify tools, exercise instructions, or contents of the exercises.

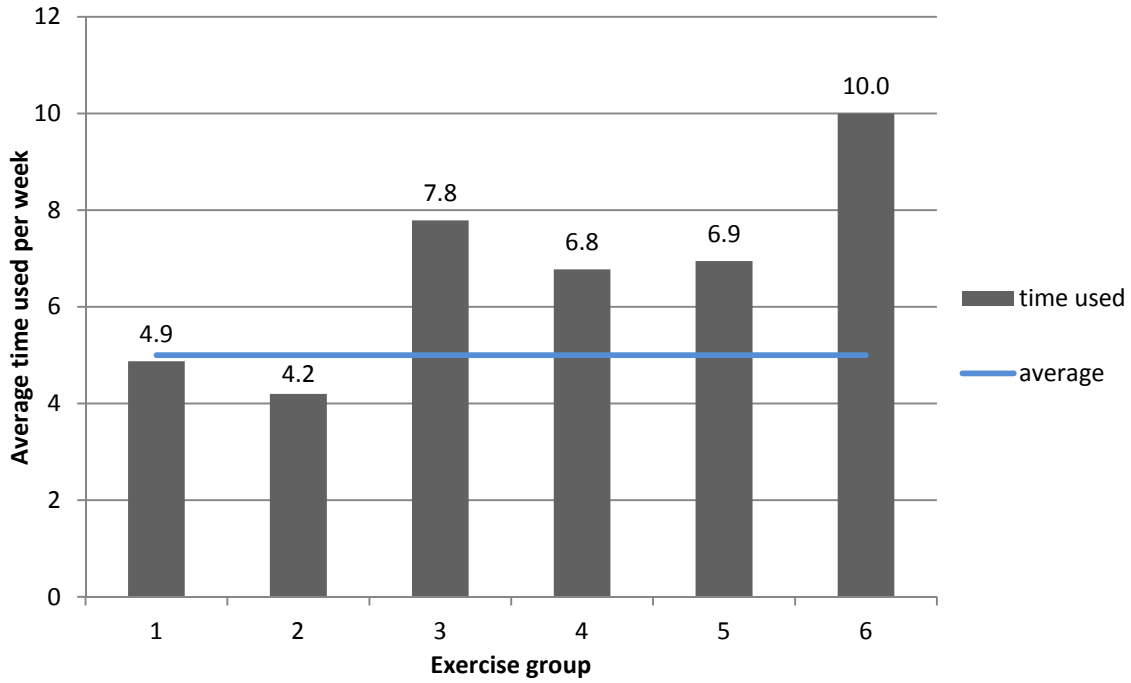

Figure 33: Time usage of exercise groups. Times are average times of all finished exercises. Planned average time for one exercise is 5 hours.

6.4 Reference Implementation

Reference implementation of the system created with Quartus II used 16,736 / 33,216 (45%) logic elements from the FPGA chip. The amount of logic elements used for combinational logic was 12,741 elements and for dedicated logic registers 9,279 elements. Total number of registers was 9380 registers. That includes two processing units, two IP blocks, and HIBI network. The frequency of the system was 50 MHz.

The size of IP block designs, written in VHDL, was 350 lines of code. The size of application codes, written in C, was approximately 4000 lines.

The reference implementation was used to compare the performances of two different IPC methods: HIBI and Avalon mailbox. Measured time was time usage of data sending to other processor and back. Size of one mailbox message is one word. That leads to several messages when sending more than one word. Sending with HIBI was always with one sending. Figure 34 shows the performance difference when amount of sending data is large. For example, mailboxes require 40X time when transfer size is 100 words. Mailboxes uses approximately 1150 clock cycles for one word transmission and DMA

uses 26 clock cycles. This shows the advantage of DMA especially with very high amount of data.

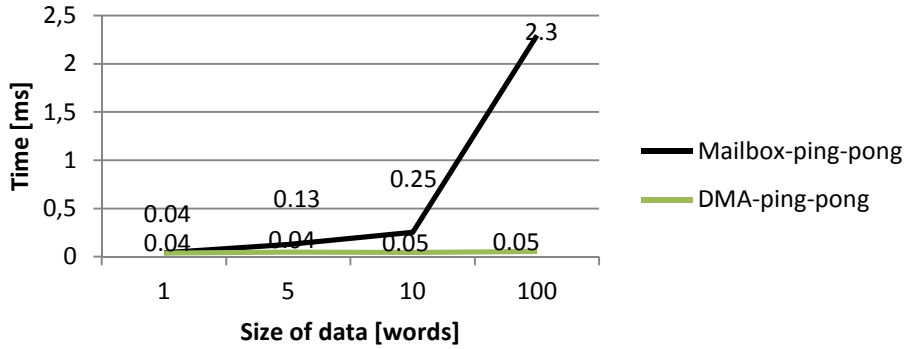

Figure 34: Used time to sending data between processors. The transmission time of mailboxes grows linearly when size of data grows, because each word is always sent separately. The transmission time of DMA grows just a slighly when data amount is incresing, because DMA sends all data at once and the spent time was due the memory operations.

7 CONCLUSION

This Thesis presented design of an exercise work related to system-on-chip platforms.

The real life MPSoCs are big and complex systems that have strong requirements. The real life projects lasts much longer than this exercise, and have much more people involved. When deciding the most important topics for the exercise work we have to keep an eye on the trends of embedded systems. One of the trends is that embedded systems have gained the market share from general purpose computers. This have became possible because of the increased performance of embedded systems during last few years. Following this trend the MPSoCs are getting more complex, and design solutions have to be considered carefully. The increased complexity of MPSoC designs is increasing the requirements of design as well. Very large systems cannot be designed from scratch because the cost and time-to-market of the product would rise too high. The reuse has become one of the most important techniques to help with this difficulty. Techniques to increase the amount of reuse are considered as a side of the each phase of the MPSoC development.

The exercise work demonstrates different phases of MPSoC design flow. Students learn the abstractions of MPSoC from HW platform to applications, from RTL, through abstractions, to application level. This Thesis handles the topics related to the exercise work. Value of different topics was considered and different design methods were compared. This thesis explains why the exercise work is constructed to the form it has. It also describes how different phases and parts of the exercise work supports the objectives described in study guide.

The implementation of the new exercise work succeeded. We got improvements to most of the topics handled. We managed to remove the black box parts of the former work, so the whole design flow is now visible for students.

REFERENCES

[1] W. Wolf, A. A. Jerraya and G. Martin, “Multiprocessor System-on-Chip Technology,” in *Transactions On Computer-Aided Design of Integrated Circuits and Systems*, IEEE, 2008, pp. 1701-1713.

[2] W. Wolf, “The future of multiprocessor systems-on-chips,” in *DAC '04 Proceedings of the 41st annual Design Automation Conference*, New York, 2004.

[3] Tampere University of Technology, “TKT3541/3547 SoC platforms,” 2012. [Online]. Available: http://www.tkt.cs.tut.fi/kurssit/3541/. [Accessed 26 4 2012].

[4] Samsung Corporation, “Samsung Exynos 4 dual 45nm brochure,” Samsung Corporation, [Online]. Available: http://www.samsung.com/global/business/semiconductor/minisite/Exynos/data/exynos4_dual_45nm.pdf. [Accessed 26 4 2012].

[5] D. Manic, D. Severac, M. Morgan and J.-P. Dan, “System-On-Chip Solutions For Portable Medical Devices,” 1 3 2008. [Online]. Available: http://www.emdt.co.uk/article/system-chip-solutions-portable-medical-devices. [Accessed 26 4 2012].

[6] E. Le Roux, N. Scolari, B. Banerjee, C. Arm, P. Volet, D. Sigg, P. Heim, J. Perotto, F. Kaess, N. Raemy, A. Vouilloz and D. Ruffieux, “icycom characteristics,” CSEM, [Online]. Available: http://www.csem.ch/docs/Show.aspx?id=12168. [Accessed 14 3 2012].

[7] I. Petkov, P. Amblard, M. Hristov ja A. Jerraya, ”Systematic Design Flow For Fast Hardware/Software Prototype Generation From Bus Functional Model For MPSoC,” tekijä: *RSP '05 Proceedings of the 16th IEEE International Workshop on Rapid System*, Washington, 2005.

[8] S. H. Chang and S. D. Kim, “Reuse-based Methodology in Developing System-on-Chip (SoC),” in *Fourth International Conference on Software Engineering Research,Management and Applications (SERA’06)*, 2006.

[9] A. Nandi ja R. Marculescu, ”System-Level Power/Performance Analysis for

Embedded Systems Design," Department of Electrical and Computer Engineering, Pittsuburgh, 2001.

[10] A. Sangiovanni-Vincentelli, "Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design," in *Proceedings of the IEEE (2007)*, vol. 95, IEEE, 2007, pp. 467-506.

[11] K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey and A. Sangiovanni-Vincentelli, "System-level design: Orthogonalization of Concerns and Platform-Based Design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 19, no. 12, p. 29, 12 2000.

[12] Altera Corporation, "2. Cyclone II Architecture," in *Cyclone II Design Handbook*, 2007, pp. 2.1-2.62.

[13] Altera Corporation, "DE2 Development and Education Board," Altera Corporation, [Online]. Available: http://www.altera.com/education/univ/materials/boards/de2/unv-de2-board.html. [Accessed 29 3 2012].

[14] O. Vainio, E. Salminen ja J. Takala, Teaching Digital Systems Using a Unified FPGA Platform, Tampere: TUT, 2010.

[15] Altera Corporation, Introduction to the Quartus II Software, San Jose: Altera Corporation, 2010, p. 126.

[16] Altera Corporation, "Nios II Processor: The World's Most Versatile Embedded Processor," Altera Corporation, [Online]. Available: http://www.altera.com/devices/processor/nios2/ni2-index.html. [Haettu 10 2 2012].

[17] Altera Corporation, Nios II Processor Referenece Handbook, San Jose: Altera Corporation, 2011.

[18] National Tsing Hua University, "EE5255 SOC Design Lab," 2004. [Online]. Available: http://larc.ee.nthu.edu.tw/~hp/EE5255/. [Accessed 26 4 2012].

[19] University of Texas, "EE382V: SYSTEM-ON-CHIP DESIGN," 2010. [Online]. Available: http://www.ece.utexas.edu/~gerstl/ee382v_s10/. [Accessed 26 4 2012].

[20] University of Turku, "SoC Design, 5 ECTS," 2010-2011. [Online]. Available: https://nettiopsu.utu.fi/opas/opintojakso.htm?id=6284&lang=en&uiLang=fi. [Accessed 26 4 2012].

[21] University of Cambridge, “Computer Laboratory,” 2010-2011. [Online]. Available: http://www.cl.cam.ac.uk/teaching/1011/CST/node79.html. [Accessed 26 4 2012].

[22] San Jose State University, “EE272 - SoC Design and Verification with SystemVerilog,” 2011. [Online]. Available: http://www.engr.sjsu.edu/tle/272syl.pdf. [Accessed 26 4 2012].

[23] University of Southampton, "System on Chip: University of Southampton," [Online]. Available: http://www.enqa.net/electrical-engineering/system-on-chip-university-of-southampton/view-details.html. [Accessed 26 4 2012].

[24] University of Westminster, “System-on-Chip Design for DSP and Communications,” [Online]. Available: http://www.mastersportal.eu/students/browse/programme/6912/system-on-chip-design-for-dsp-and-communications.html. [Accessed 1 2 2012].

[25] University of Twente, “System-on-Chip Design (121075),” 2011. [Online]. Available: http://wwwhome.cs.utwente.nl/~gerezsh/soc/index.html. [Accessed 26 4 2012].

[26] Linköping Institute of Technology, “Computer Hardware - a System on Chip,” 2011. [Online]. Available: http://kdb-5.liu.se/liu/lith/studiehandboken/action.lasso?&-response=enkursplan.lasso&op=eq&k_budget_year=2011&op=eq&k_kurskod=TSEA44. [Accessed 26 4 2012].

[27] University of Illionois, “ECE 527 SoC Design,” 2011. [Online]. Available: http://courses.engr.illinois.edu/ece527/. [Accessed 26 4 2012].

[28] Tampere University of Technology, “TKT-2431 SoC-suunnittelu,” 2012. [Online]. Available: http://www.tkt.cs.tut.fi/kurssit/2431/. [Accessed 26 4 2012].

[29] L. Matilainen, A. Kamppi, J.-M. Määttä, E. Salminen and T. D. Hämäläinen, “KACTUS2: IP-XACT/IEEE1685 Compatible Design Environment for Embedded Multiprocessor System-on-Chip Products,” TUT, Tampere, 2011.

[30] IEEE, IEEE 1685: Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP Within Tool Flows, New York: IEEE, 2009, p. 360.

[31] ARM Limited, IP-XACT Components reference manual, 2007.

[32] Multicore Association, “MULTICORE COMMUNICATIONS API WORKING

GROUP," [Online]. Available: http://www.multicore-association.org/workgroup/comapi.php. [Accessed 24 4 2012].

[33] Tampere University of Technology, "POP portal," [Online]. Available: http://www.tut.fi/pop. [Haettu 9 5 2012].

[34] Tampere University of Technology, "Study guide of TUT - POP portal," 2011-2012. [Online]. Available: https://pop-portal.tut.fi/portal/page/portal/POP-portaali/20Opinnot/23Opinto-opas. [Accessed 25 1 2011].

[35] Department of Copmuter Systems, TUT, "TKT-3541/TKT-3547 SoC platforms Exercises," 2011. [Online]. Available: http://www.tkt.cs.tut.fi/kurssit/3541/K11/Ex/. [Haettu 14 5 2012].

[36] eCos Community, "eCos," [Online]. Available: http://ecos.sourceware.org/. [Haettu 9 5 2012].

[37] Micrium, "Micrium - µC/OS-II Kernel," Micrium, [Online]. Available: http://micrium.com/page/products/rtos/os-ii. [Accessed 8 5 2012].

[38] E. Salminen, L. Matilainen, J. Arvio, A. Alhonen and L. Lehtonen, "Funbase IP libarary," OpenCores, [Online]. Available: http://opencores.org/project,funbase_ip_library. [Accessed 26 4 2012].

[39] E. Salminen and T. Hämäläinen, Heterogenerous IP Block Interconnection (HIBI) Reference Manual, Tampere: TUT, 2011, p. 41.

[40] Altera Corporation, Guidelines for Developing a Nios II HAL Device Driver, San Jose: Altera Corporation, 2011.

[41] Altera Corporation, "FPGAs," Altera Corporation, [Online]. Available: http://www.altera.com/products/fpga.html. [Haettu 10 2 2012].

APPENDIX 1 – EXERCISE WORK INSTRUCTIONS

Exercise 1: SoC Specification

The purpose of this exercise if to learn how to write specifications for a System on a Chip.
You're given a ready made specification, which is missing parts that you need to fill. The parts that you need to modify are marked with yellow color. For the diagrams you can use Microsoft Visio 2010 which is installed in the Windows class (TC419).

We recommend you to use many UML diagrams in the specifications. There is a requirement that you use at least three UML diagrams in the specification. Some hints/requirements of the UML diagrams are presented in the specification template.

What to specify

During the exercise sessions a game is developed for the Altera DE2 board. Basic idea of the game is to press buttons according to flashing LEDs. The game will eventually get harder as the flashing rate will increase. Points are being awarded for each correct press. Incorrect press will end the game. To get more familiar with the game logic you can try to play web version of the game found in the links at the bottom of this page. There is also link to a Youtube video of gameplay if you are unfamiliar with the subject.

SoC specification must comply with given user requirements to be acecptable. Before returning the SoC specification double check that your specification complies with these requirements or it will be rejected.
It is required that you create documentation of your HW architecture using Kactus2 software (Kactus2 pictures of the HW architecture are required in the SoC specification). For this exercise Kactus2 software is used only for documentation purposes. During the later exercises we'll make use of the Kactus2 software more extensively so it's very important that you learn the basics during the first exercise. In the materials section you'll find tutorial howto create hierarchical systems using the Kactus2 software, which introduces the necessary features required to create documentation for the exercise 1.

Material for the exercise

- SoC Specification Template(16 pages)
- User Requirements (5 pages)
- Kactus2 hierarchical design tutorial (15 pages)

What to return

Compress the following files into zip file named TKT-3541-Ggg-E01.zip, where gg is your group number.

1. Completed SoC system specification document

2. Kactus2 created documentation files (html+pngs) in compressed zip file named kactus2_document.zip
3. Your time usage on this exercise written in the body section of the return email

Exercise 2: Altera SoPC Design

The purpose of this exercise is to get familiar with the Altera design flow. After this exercise you should be familiar with Quartus2, SoPC-builder, and Nios2 EDS tools.

To complete this exercise, complete the Altera SoPC-builder tutorial and return the required files.

Make sure that you are confident in using these tools, as they are used extensively in later exercises.

Material for the exercise

- Altera SoPC-builder tutorial (8 pages)
- SRAM Controller IP (5 files)
- DE2 pin assignments (1 file)
- SDC file for Timequest timing analyzer (1 file)

What to return

Compress the following files into zip file named TKT-3541-Ggg-E02.zip, where gg is your group number.

1. Fitter Summary (Located in Quartus project folder with file extension .fit.summary)
2. System.h files (Located in Software BSP project folder)
3. terminals.png, which is a screenshot of the two nios2-terminals showing hello world programs running on nios2 terminals
4. Your time usage on this exercise written in the body section of the return email

Exercise 3: IP-Block HW Design

The purpose of this exercise is to create own custom IP-block that can read pushbuttons and drive leds/7-segment display. IP-block is connected to HiBi network so it can communicate with Nios2 processors. Detailed instructions can be found from the Exercise instructions pdf.

Material for the exercise

- HIBI.zip (several files)
- Exercise Instructions (3 pages)
- HiBi Datasheet (42 pages)

What to return

Compress the following files into zip file named TKT-3541-Ggg-E03.zip, where gg is your group number.

1. IP-block VHDL code(s)
2. IP-block test bench VHDL code(s)
3. Your time usage on this exercise written in the body section of the return email

Exercise 4: Driver Design

The purpose of this exercise is to create driver for the HIBI_PE_DMA block and integrate it to Altera HAL. You are given a skeleton which already contains most of the stuff needed to implement the driver functionality. Detailed instructions on how to do this exercise can be found from the exercise instructions pdf.

Material for the exercise

- Exercise Instructions (8 pages)
- HIBI_PE_DMA.zip (several files)
- HIBI.zip (several files)
- Simple Test Program (1 file)
- HIBI_PE_DMA Reference Documentation (13 pages)
- HIBI_PE_DMA Introduction (23 slides)
- Building HIBI_PE_DMA System (17 slides)

What to return

Compress the following files into zip file named TKT-3541-Ggg-E04.zip, where gg is your group number.

1. Screen capture of your top level design, or top level vhdl code (if you are using vhdl as a top level)
2. hibi_pe_dma_read.c
3. hibi_pe_dma_write.c
4. hibi_pe_dma_ioctl.c
5. hibi_pe_dma_lseek.c
6. hibi_pe_dma_close.c
7. All other files that you have modified/added to the driver
8. Your custom test program (if you have used custom test program and not the given one)
9. Your time usage on this exercise written in the body section of the return email

Exercise 5: Tasks and Synchronization

In this exercise we're going to make the game logic working on a single processor. Buttons and other peripherals are emulated using dedicated tasks so we can easily verify the correctness of the game logic itself. After this exercise you should be quite familiar with uCos-II features such as tasks, semaphores, mailboxes, and message queues. See more detailed exercise instructions from the link below.

Material for the exercise

- Exercise Instructions (3 pages)

What to return

Compress the following files into zip file named TKT-3541-Ggg-E05.zip, where gg is your group number.

1. Code file(s) of the test program and tasks
2. Your time usage on this exercise written in the body section of the return email

Exercise 6: IPC and Messaging

In this exercise we're going try two different methods on transferring messages/data between two processors. First method is to use the HiBi network to transfer the data. Second method is Alteras mailbox core. We are going to use performance counter to measure the time between the transfers. See more detailed exercise instructions from the link below.

Material for the exercise

- Exercise Instructions (2 pages)
- Measurement Table (1 sheet)
- HIBI PE DMA HW + DRIVERS (several files)

What to return

Compress the following files into zip file named TKT-3541-Ggg-E06.zip, where gg is your group number.

1. Filled measurement table (doc/pdf)
2. Your time usage on this exercise written in the body section of the return email

Exercise 7: Game Design 1

In this exercise you need to implement the whole game functionality using the specification that you created in the exercise 1.
You need to demonstrate working game to the assistant!

What to return

Compress the following files into zip file named TKT-3541-Ggg-E07.zip, where gg is your group number.

1. Hardware configuration file (.sof)
2. All software files to build the game application
3. Demonstration of the working game to the assistant
4. Your time usage on this exercise written in the body section of the return email

Exercise 8: IP-XACT Basic HW design

In this exercise we're going to implement again simple SoC design using Kactus2 software. The goal of this exercise is just to get more familiar to Kactus2 software. In later exercises we're going to implement the full SoC game design.

Material for the exercise

- Exercise Instructions (4 pages)
- Library Files (several files)

What to return

Compress the following files into zip file named TKT-3541-Ggg-E08.zip, where gg is your group number.

1. Your vendor library (Gxx folder)
2. Generated Top Level VHDL code
3. Generated Quartus Project file
4. Your time usage on this exercise written in the body section of the return email

Exercise 9: IP-XACT Game HW Design

In this exercise you packetize your own IPs (led and button) to IP-XACT format used by the Kactus2 tool. After this exercise you should have the game HW implemented using Kactus2 software. See exercise instruction for further details. Download new library files (these ones have the timer included for uCos-II). Replace the previous exercise library contents with these so you can use uCos-II OS.

You must also edit nios__ii_sram component. It is missing its hibi interface port maps. Double click component to open it. Edit hibi_if Port maps so that assignments are correct.
Hibi segment component has bad default value which you must edit for it to work correctly. Click on the component on your soc design view and on the right hand pane you should see Configurable element values list. Double click on the list and select number_of_r4_agents_g. Assign value 4 to it. In case you can't see the configurable element values list you can also edit the component itself. In this case you can find this option under Model Parameters.
After this you are missing two source files, named fifo.vhd and multiclk_fifo.vhd. You can manually add these files to the Quartus project or edit hibi component to include these files. These source files can be found from your previous exercises or download from HIBI.zip found in exercise 3.

Material for the exercise

- Exercise Instructions (5 pages)
- Library Files (several files)

What to return

Compress the following files into zip file named TKT-3541-Ggg-E09zip, where gg is your group number.

1. Your vendor library (Gxx folder)
2. Your time usage on this exercise written in the body section of the return email

Exercise 10: IP-XACT SW design

In this exercise you packetize your SW using Kactus2 software. Purpose of this exercise is to get familiar with Kactus 2 SW design features. Packetize application codes and SW platform into IP-XACT metadata objects and map SW components to the underlying HW platform. Current version of Kactus does not support autogeneration of makefile i.e. it is used for documentation and project source code management purposes.

Material for the exercise

- Exercise Instructions (4 pages)
- Library Files (several files)

What to return

Compress the following files into zip file named TKT-3541-Ggg-E10zip, where gg is your group number.

1. Your vendor library (Gxx folder)
2. Your time usage on this exercise written in the body section of the return email

Exercise 11: MCAPI Design

Purpose of this exercise is to get familiar with Kactus 2 MCAPI design features and MCAPI itself. MCAPI is alternative way to implement communication between processing elements via HIBI. In this exercise you design MCAPI communication between all PEs in your HW platform including HW accelerators. Current version of Kactus does not support autogeneration of makefile i.e. it is used for documentation and project source code management purposes. See exercise instructions for further details.

Material for the exercise

- Exercise Instructions (3 pages)
- Library Files (several files)

What to return

Compress the following files into zip file named TKT-3541-Ggg-E11.zip, where gg is your group number.

1. Your vendor library (Gxx folder)
2. Your time usage on this exercise written in the body section of the return email

Exercise 12: Game design 2

The purpose of this exercise is to create the game application using Kactus2 created HW. By now you should already have the hardware (exercise 9) and software (exercise 7) ready so the only thing left to do is to simple put everything together once more and fix any remaining bugs.

What to return

Compress the following files into zip file named TKT-3541-Ggg-E12.zip, where gg is your group number.

1. Your vendor library (Gxx folder)
2. Your top level vhdl code (Created with Kactus2)
3. Demonstration of the working game
4. Your time usage on this exercise written in the body section of the return email.