Institutionen för systemteknik
Department of Electrical Engineering

Examensarbete

FPGA Implementation of a UPnP Media Renderer

Examensarbete utfört i Elektroniksystem
vid Tekniska högskolan i Linköping
av

Karl-Rikard Ländell
Axel Wiksten Färnström

LiTH-ISY-EX--11/4368--SE

Linköping 2011

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Electrical Engineering	Linköpings tekniska högskola
Linköpings universitet	Linköpings universitet
SE-581 83 Linköping, Sweden	581 83 Linköping

FPGA Implementation of a UPnP Media Renderer

Examensarbete utfört i Elektroniksystem
vid Tekniska högskolan i Linköping
av

Karl-Rikard Ländell
Axel Wiksten Färnström

LiTH-ISY-EX--11/4368--SE

Handledare:	**Kent Palmkvist**
	ISY, Linköpings universitet
	Pär Gunnars Risberg
	Actiwave AB
Examinator:	**Kent Palmkvist**
	ISY, Linköpings universitet

Linköping, 8 November, 2011

Avdelning, Institution Division, Department Division of Electronics Systems Department of Electrical Engineering Linköpings universitet SE-581 83 Linköping, Sweden	**Datum** Date 2011-11-08

Språk Language ☐ Svenska/Swedish ☒ Engelska/English ☐ ____________	**Rapporttyp** Report category ☐ Licentiatavhandling ☒ Examensarbete ☐ C-uppsats ☐ D-uppsats ☐ Övrig rapport ☐ ____________	**ISBN** — **ISRN** LiTH-ISY-EX--11/4368--SE **Serietitel och serienummer** Title of series, numbering **ISSN** —

URL för elektronisk version

http://www.es.isy.liu.se
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-4368

Titel Title	Implementation av en UPnP Media Renderer på en FPGA FPGA Implementation of a UPnP Media Renderer
Författare Authors	Karl-Rikard Ländell, Axel Wiksten Färnström

Sammanfattning
Abstract

Actiwave AB delivers audio solutions for active speakers. One of the features is that audio can be streamed to the speakers over a local network connection. The module that provides this functionality is expensive. This thesis investigates if this can instead be achieved by taking advantage of the Spartan-6 FPGA on their platform, using part of it as a MicroBlaze soft processor on which a rendering device can be implemented.

The thesis discusses design decisions such as selection and integration of operating system, UPnP framework and media decoder.

A fully functional prototype application for a desktop computer was implemented, with the intention of porting it to the FPGA platform. There turned out to be too many compability issues though, so instead, a simpler renderer was implemented on the FPGA.

Mp3 music files were successfully streamed to and decoded on the soft processor, but without fulfilling real-time constraints.

The conclusion is that it is reasonable to implement a UPnP Media Renderer on the FPGA. Decoding in real-time can be an issue due to insufficient performance of the soft processor, but several possible solutions exist.

Nyckelord
Keywords FPGA, MicroBlaze, UPnP, embedded linux, PetaLinux

Abstract

Actiwave AB delivers audio solutions for active speakers. One of the features is that audio can be streamed to the speakers over a local network connection. The module that provides this functionality is expensive. This thesis investigates if this can instead be achieved by taking advantage of the Spartan-6 FPGA on their platform, using part of it as a MicroBlaze soft processor on which a rendering device can be implemented.

The thesis discusses design decisions such as selection and integration of operating system, UPnP framework and media decoder.

A fully functional prototype application for a desktop computer was implemented, with the intention of porting it to the FPGA platform. There turned out to be too many compability issues though, so instead, a simpler renderer was implemented on the FPGA.

Mp3 music files were successfully streamed to and decoded on the soft processor, but without fulfilling real-time constraints.

The conclusion is that it is reasonable to implement a UPnP Media Renderer on the FPGA. Decoding in real-time can be an issue due to insufficient performance of the soft processor, but several possible solutions exist.

Acknowledgments

We would like to thank Kent Palmkvist for providing us with computers, tools and somewhere to work. We would also like to thank Pär Gunnars Risberg for giving us the opportunity to work on an interesting subject, letting us be a part of the Actiwave team and fueling us with new energy when progress was slow.

Contents

Abbreviations

ASIC	Application Specific Integrated Circuit
DDR	Double Data Rate
DHCP	Dynamic Host Configuration Protocol
DLNA	Digital Living Network Alliance
EDK	Embedded Developers Kit
FPGA	Field-Programmable Gate Array
FPU	Floating Point Unit
GPIO	General Purpose Input/Output
IP	Intellectual Property
IP	Internet Protocol
ISE	Integrated Software Environment
JTAG	Joint Test Action Group
LED	Light-Emitting Diode
LUT	Look-Up Table
MMU	Memory Management Unit
RAM	Random Access Memory
RF	Radio Frequency
RISC	Reduced Instruction Set Computer
SDIO	Secure Digital Input/Output
SPI	Serial Peripheral Interface
TFTP	Trivial File Transfer Protocol
UART	Universal Asynchronous Receiver/Transmitter

UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
USB	Universal Serial Bus
WLAN	Wireless Local Area Network
XML	eXtensible Markup Language

Chapter 1

Introduction

This chapter is an introduction to this thesis and describes why and how the work was carried out. It also contains an outline of the thesis.

1.1 Background

Actiwave AB delivers innovative solutions for the audio industry using digital filters and advanced algorithms. Their aim is to allow their customers to produce small speakers with focus on design without compromising the sound quality.

One of the key features of Actiwave products is the ability to recieve audio through a wireless network connection. To achieve this, they use a rather expensive hardware module, that supports the UPnP standard. It allows the device to connect to a wireless network and act as a playback device, called a Media Renderer.

The aim of this thesis is to provide the same functionality, by taking advantage of the FPGA that is already present on their platform. This FPGA contains the main functionality of filtering and sound altering, but there is a lot of unutilized logic left that can be used for other purposes. The advantage of this solution is that it will greatly reduce the production cost, and it will also result in a physically smaller product, which is preferable from a design perspective.

1.1.1 FPGA

A Field-Programmable Gate Array (FPGA) is a device that can be programmed to perform any logical function. The fact that it is reprogrammable makes it useful for developing new products, which are then often realized as ASICs. However, FPGA prices have dropped in the recent years. They are now affordable enough to be used in commercial products.

The obvious advantage of having an FPGA in a product is that new features can be added without altering the hardware platform, which reduces the time-to-market and developing costs.

Disadvantages are that the unit cost is higher and the performance is not as good as a corresponding ASIC. In this case, an FPGA is already included in the target platform. If a larger FPGA is needed though, that could lead to extra cost per unit.

1.1.2 Soft Processor

Soft processors are IP cores that can be implemented on an FPGA. This means that part of an FPGA can act as a processor, configured for its specific task.

While using a soft processor increases flexibility and shortens time-to-market, the performance, as in the case with FPGAs in general, can not be expected to match that of a fabricated processor.

The architecture of a soft processor is usually quite simple. Typically there is no FPU or MMU. This is due to the fact that complex constructions take up a large portion of the area available on the FPGA. As FPGAs have become more and more popular in recent years, prices have dropped and the FPGAs have grown bigger. This has led to the development of more complex soft processors, with FPU and MMU as optional features.

1.2 Objectives

The main objective of this thesis is to develop a working prototype of a media renderer. This is to be achieved by implementing a soft processor on the given FPGA. A WLAN chip for communication with other network devices should also be selected. It will need to be interfaced to the FPGA.

Software needs to be written, implementing the Media Renderer functionality. If possible, this could be achieved by modifying existing open source software.

Playback of mp3 files is mandatory and requires a decoder. If there is time, support for other file formats (e. g. WMA, FLAC, Ogg Vorbis) is to be implemented as well.

Basic control functionality is also to be implemented if time is available. This includes being able to skip to the next track or pause playback.

1.3 Scope

The desired functionality is to be implemented using a soft processor on the FPGA. The main advantage of using a soft processor is that many parameters can be adjusted and tested, to find a setup that is ideal for the specific purpose. In the case of this thesis however, it is important to be careful not to take up too much space on the FPGA.

An operating system that can run on the soft processor is needed. Renderer software compatible with that operating system must be written.

1.4 Method

Already existing open source software should be used when possible, to shorten the time needed to be spent on software development.

The first phase will be to collect facts and gather understanding on the DLNA and UPnP standards. The official documentation will be the starting point.

Next thing to do is to research on available operating systems, software libraries and WLAN chips. Regarding operating systems and software libraries, the search will have to be very general. The best place to look for WLAN chips would be the major vendors.

When the research is done, the next step is to configure the soft processor and get the operating system of choice up and running on the FPGA, to have a platform to implement the renderer on.

With the platform available, the rendering application can be developed.

Once the whole system is working satisfactory on the test board, it is time to interface the WLAN chip to the FPGA.

If there is still time available after that, additional features can be implemented, for example extra control functionality.

1.5 Outline

This report is meant to be readable for anyone with a basic understanding of computer architecture and electronic systems. Concepts that may be unknown to the reader are briefly explained, so that the contents of the report can be understood.

The initial chapters introduces the reader to UPnP, the available hardware, and PetaLinux. The Implementation chapter describes how the work was carried out and the design decisions made during it. The Results chapter lists what has been achieved during the project. The concluding chapters discusses issues that occured during the project, the results and the possibilities for future development.

Chapter 2

Overview of UPnP

This chapter briefly describes the concepts of UPnP technology, focusing on its media functionality.

2.1 Introduction

UPnP is the name of a number of networking protocols, with the purpose of simplifying implementation and usage of network devices. The protocols are defined by the UPnP Forum, formed in 1999 by a coalition of computer industry companies. The different usage scenarios of UPnP devices range from watching videos streamed over a local network to switching on or off a light by a network command.[5]

The protocols are built on open standards commonly used for internet communication, such as TCP/IP, UDP, HTTP and XML. The implementation of the protocols is independent of media, platform and programming language and offers functionality for addressing, discovery, description, control, event notification and presentation.[5]

2.2 Media Devices

Media products that have UPnP functionality can be categorized as either Control Points, Devices, or both. Control Points coordinate the other Devices, usually in response to input from the user.[5] A common example of a Control Point could be a computer running a program which can control media playback on Devices connected to the local network. One Device could be a WLAN-connected mobile phone, acting as a server from which the media can be streamed. The renderer device could then be an ethernet-connected TV playing the streamed media.

The above example involves the two categories of devices handling audio and video; MediaServer and MediaRenderer. The communication between them transfers the actual media, but this transfer does actually not involve UPnP protocols.

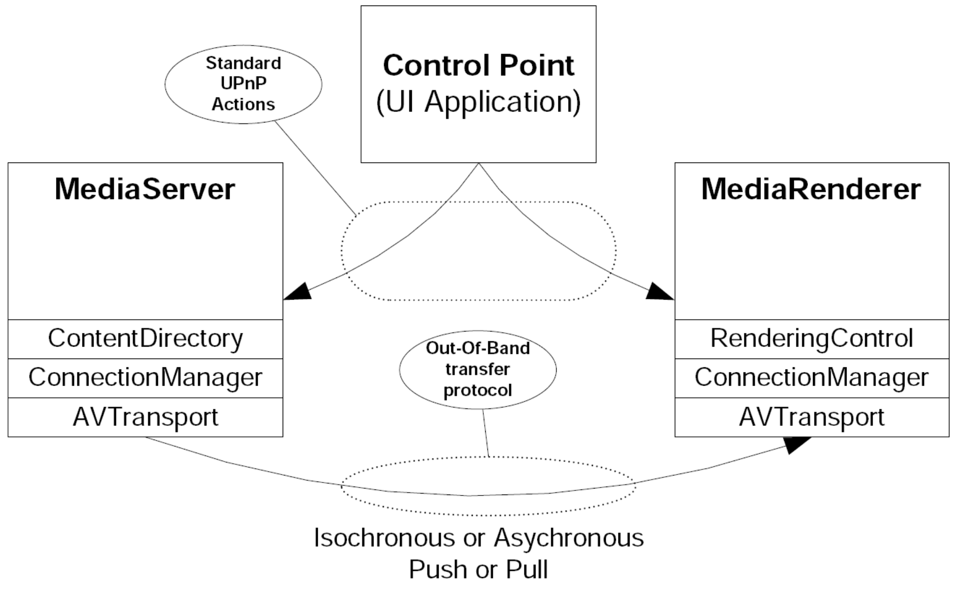

Figure 2.1. *Communication between UPnP Media Devices[3].*

However, the communication between the Control Point and the MediaServer device, as well as between the Control Point and the MediaRenderer device, to set up and control the media transfer is dictated by the UPnP standard.[3] This example is illustrated by figure 2.1.

2.3 Services

Also seen in figure 2.1 are the services of MediaServer and MediaRenderer devices respectively. The UPnP services are described in the UPnP AV Architecture document[3], which is the source for all statements in this section, unless something else is specified. A UPnP service provides a device with functionality and has a set of state variables associated with it.

The functionality is implemented with UPnP actions, which can be described as commands a device can recieve. Depending on the action, the device may or may not return the value of a state variable as a response.

The following sections will briefly discuss the different services of UPnP Media Devices.

2.3.1 ConnectionManager

The ConnectionManager service handles the connections of a device and is mandatory for both MediaRenderer devices and MediaServer devices. It allows the Control Point to setup and close connections between devices and to find information about ongoing tranfers in the network[2].

This service also makes it possible to distinguish between several MediaRenderer devices by assigning them unique device identifiers. This makes it possible for a single MediaServer device to support multiple MediaRenderer devices at the same time. It can for example be useful if the user want to stream the same audio content to more than one room.

2.3.2 ContentDirectory

ContentDirectory is a service that allows a Control Point to enumerate and browse the data available on the MediaServer device. The server returns information (meta-data) such as the name of a music track, the name of an artist, etc. The meta-data also includes what kind of data formats and transfer protocols are available for the particular media content, so that this can be matched with what the MediaRenderer is able to show or play.

2.3.3 AVTransport

Depending on the transfer protocol used (push- or pull-based), the AVTransport service may be available either in the MediaServer device or in the MediaRenderer device. It offers the ability to control the flow of the streamed content, with actions such as play, pause, stop, seek, etc.

2.3.4 RenderingControl

The RenderingControl service is available in the MediaRenderer device. It allows the Control Point to control how the content is rendered, by adjusting volume, brightness etc.

2.4 A Typical Playback Scenario

The following scenario illustrates the mechanisms involved and actions invoked, when playing content over network on UPnP media devices. This particular scenario uses a pull-based transfer protocol, which means that the AVTransport service is available in the MediaRenderer device. For further reference, see the UPnP AV Architecture document[3], unless something else is specified.

At the start of the scenario, a Control Point is already connected to the network, as the user adds a MediaRenderer device and a Mediaserver device to the setup.

Figure 2.2 illustrates the actions invoked by the Control Point, as well as their responses. AVT is short for AVTransport Service, CM is for ConnectionManager Service, RCS is for RenderingControl Service and CDS is for ContentDirectory Service.

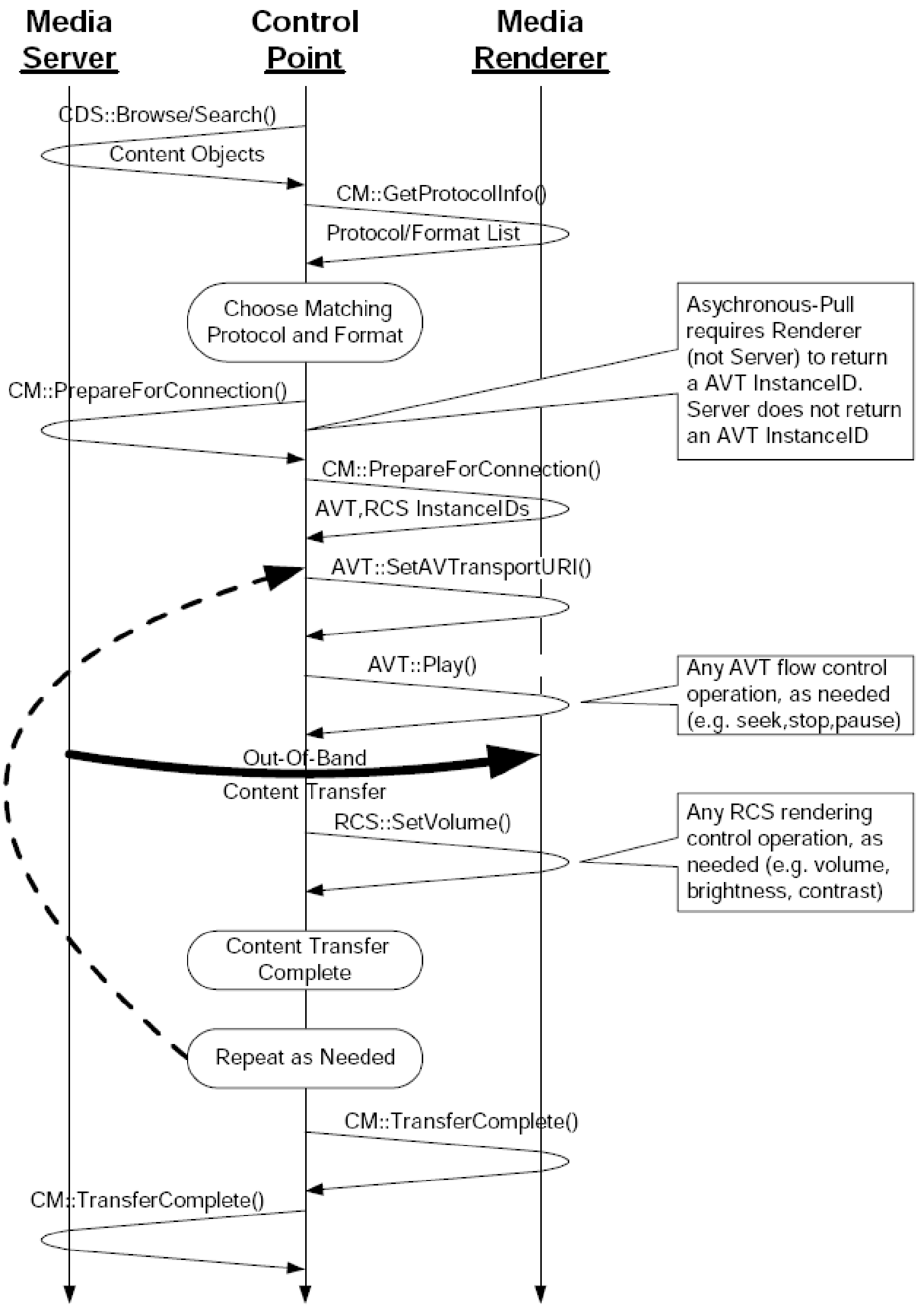
Media Server
Control Point
Media Renderer
CDS::Browse/Search()
Content Objects
CM::GetProtocolInfo()
Protocol/Format List
Choose Matching Protocol and Format
Asychronous-Pull requires Renderer (not Server) to return a AVT InstanceID. Server does not return an AVT InstanceID
CM::PrepareForConnection()
CM::PrepareForConnection()
AVT,RCS InstanceIDs
AVT::SetAVTransportURI()
AVT::Play()
Any AVT flow control operation, as needed (e.g. seek,stop,pause)
Out-Of-Band Content Transfer
RCS::SetVolume()
Any RCS rendering control operation, as needed (e.g. volume, brightness, contrast)
Content Transfer Complete
Repeat as Needed
CM::TransferComplete()
CM::TransferComplete()

Figure 2.2. *UPnP actions in a playback scenario[3].*

2.4.1 Addressing

When a UPnP device is connected to the network, it is given an IP address. In most cases this is handled by the DHCP client, which is a required component in each UPnP device.[4]

2.4.2 Discovery

When an IP address has been assigned to each device, the next step is to advertise its existence and basic information about which services it can provide to the Control Points in the network. The Control Point may then request an XML description of the device, which contains more details about the actions and state variables that are available.[4]

2.4.3 Content Selection

Since the Control Point now has discovered that a MediaServer device with a ContentDirectory service has been connected to the network, it allows the user to browse the content on the server, using the user interface of the Control Point.

The content information retrieved from the MediaServer device also contains information about the contents available formats and by which protocols it can be transfered.

A MediaRenderer device has been discovered as well, which the user selects for playback of the desired content.

2.4.4 Match Protocols and Formats

The Control Point now uses the ConnectionManager service of the MediaRenderer device to find out which transfer protocols and formats are supported by it. The Control Point then matches this information with that of the MediaServer content that previously was selected, to choose a format and protocol that are supported by both of the devices.

2.4.5 Configure Content Transfer

The next step is to communicate the decisions that were made to the media devices, and allow them to prepare for the connection. The Control Point also invokes an action that gives the MediaRenderer device the URI of the media content selected by the user.

2.4.6 Start Transfer

Now the playback can be started, by invoking the Play action in the AVTransport service of the MediaRenderer device.

2.4.7 Control Playback

While playing the content, the user can pause or stop the playback, as well as adjust the volume or mute the audio, through the user interface of the Control Point. Any command given by the user, has a corresponding action either in the AVTransport service or the RenderingControl service, which is invoked by the control point.

2.4.8 Select Additional Content

Depending on the user interface, the user might be able to create a playlist in the Control Point. In this case, the URI of the next content item is given to the AVTransport service at the end of playback of the first item, and playback is started again.

2.4.9 End Transfer Session

When the playback session is ended, the Control Point invokes an action that terminates the connection between them.

Chapter 3

The Hardware Platform

This chapter presents the MicroBlaze soft processor and the FPGA board. It also mentions the main components of the SP601 board and how they relate to this thesis.

3.1 Introduction

The core of Actiwave's platform is an FPGA, which is used for filtering and audio manipulation. Only a small part of it is utilized though, which leaves room to exploit for the rendering functionality.

Actiwave provided a Xilinx SP601 board, containing the FPGA they use in their products, so that the work on the FPGA could be started.

3.2 Xilinx SP601 Evaluation Board

The board and its components are depicted in figure 3.1. The components relevant for this thesis are described in the subsections below.

3.2.1 FPGA

The main component of the board is the FPGA. The SP601 board carries a Xilinx Spartan-6 FPGA called XC6LX16. It has 2278 logic slices with four look-up tables and eight flip-flops in each of them.[28] This is where the synthesized MicroBlaze soft processor is implemented.

The Spartan-6 FPGA also contains a total of 72 KB of Block RAM[28], which for example can be used as cache memory for the MicroBlaze processor.

3.2.2 Memory

There is 128 MB of DDR2 memory available on the board[27]. In this thesis, it is used as the primary memory for the MicroBlaze soft processor.

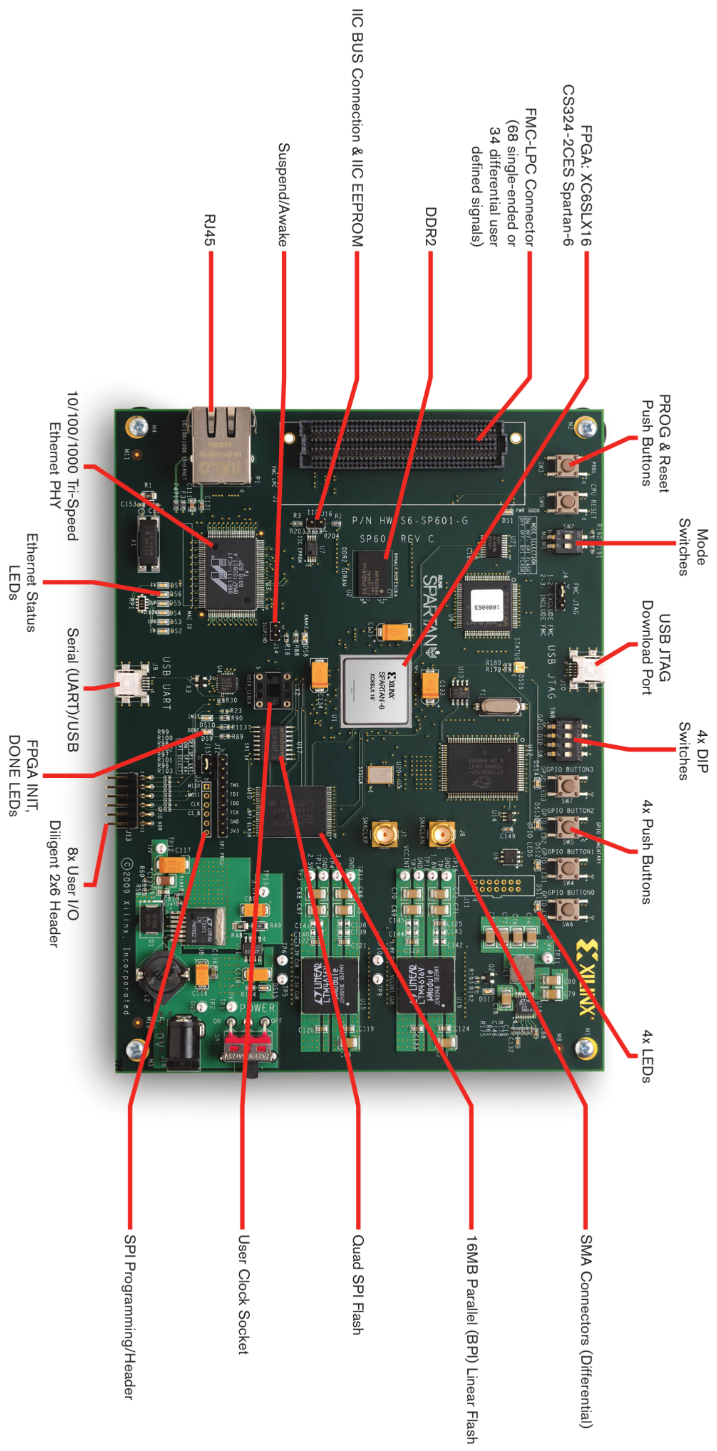
FPGA: XC6SLX16 CS324-2CES Spartan-6
FMC-LPC Connector (68 single-ended or 34 differential user defined signals)
DDR2
IIC BUS Connection & IIC EEPROM
Suspend/Awake
RJ45
10/100/1000 Tri-Speed Ethernet PHY
Ethernet Status LEDs
Serial (UART)/USB
FPGA INIT, DONE LEDs
8x User I/O Diligent 2x6 Header
PROG & Reset Push Buttons
Mode Switches
USB JTAG Download Port
4x DIP Switches
4x Push Buttons
4x LEDs
SPI Programming/Header
User Clock Socket
Quad SPI Flash
16MB Parallel (BPI) Linear Flash
SMA Connectors (Differential)

Figure 3.1. *The Xilinx SP601 Evaluation Board*[26].

3.2.3 Connectivity

There are two USB connections on the SP601 board. The USB-to-UART interface is used as a communications link between a PC and the board, so that the user can use a terminal interface on the PC to control and monitor the implementation on the board. The USB JTAG interface is used to program the FPGA.

There is also an ethernet connection, which makes it possible to connect the board to a local network[27].

3.2.4 Flash

There is a 16 MB Flash memory that can be used as secondary memory[27]. A final implementation could be stored here, so that it can be loaded into the primary memory on startup.

3.2.5 Peripherals

There are 4 LEDs, 4 push buttons and 4 switches availible on the board[27].

3.3 Xilinx ISE Design Suite

ISE is a set of design tools for Xilinx FPGAs. The Embedded Edition of ISE includes tools and IP blocks for development of embedded systems as well as tools for developing software for those systems[29]. It includes an application for hardware development, called Xilinx Platform Studio (XPS). XPS allows the user to create customized processor systems on FPGAs using graphical design views.[30]

3.4 The MicroBlaze Soft Processor

Xilinx offers a soft processor solution for their FPGAs, called the MicroBlaze. It is a 32-bit RISC architecture core that comes in two versions. One is area-optimized and has a 3-stage pipeline, while the other is performance-optimized and has a 5-stage pipeline.[25]

The MicroBlaze core is included in the Embedded Edition of ISE Design Suite and can easily be configureded and generated using the configuration wizard. The size of cache memory and local memory, the clock frequency and which peripherals to include are some of the parameters that can be adjusted. It is also possible to decide whether the processor will or will not include an FPU and/or an MMU.[25]

Chapter 4

Overview of PetaLinux

This chapter introduces PetaLinux and describes the general steps of building and running it on a MicroBlaze soft processor.

4.1 Introduction

PetaLinux is the name of a linux distribution targeting FPGA-based designs[12]. It started as a port of uClinux for the MicroBlaze soft processor but has recently become commercialized and further developed under its new name. PetaLinux is developed and maintained by PetaLogix.[10]

In this thesis, PetaLinux version 0.40 Final has been utilized. It is the version that PetaLogix supplies to students through their university program[15].

4.1.1 uClinux

uClinux is a branch of Linux intended for MMU-less systems. It was released for the Motorola 68000 microprocessor in 1998 and has been ported to many different architectures since then. The name uClinux is short for Micro (greek letter μ) Controller Linux.[7]

Early versions were based on the 2.0 Linux kernel, while later versions are based on the 2.4 or 2.6 kernel and include support for devices with MMUs[7].

4.2 Host PC

PetaLinux is built and downloaded to the target FPGA board from a desktop PC running Linux. It needs to have the Xilinx ISE Suite and EDK tools installed as well as ports to connect to the JTAG and serial interfaces of the target board (which in this specific case is done through two USB ports).[13]

Another requirement to be able to communicate with the target device, is to have a console program installed. The PetaLinux documentation recommends Kermit for this purpose[13].

4.3 The Board Bring-up Process

The board bring-up process is described in the PetaLinux user guide[14]. The key steps in this process are shown in figure 4.1 and will be outlined in this section.

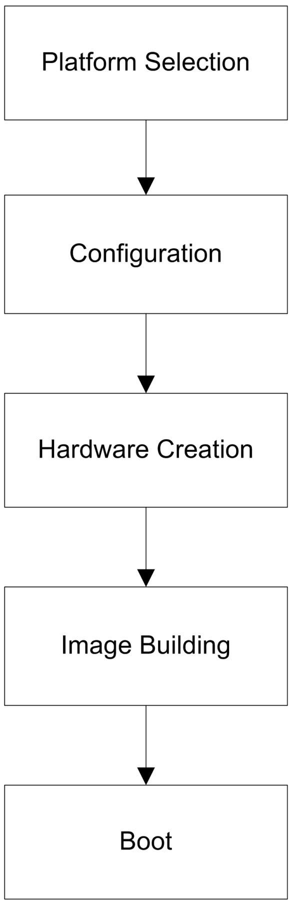

Figure 4.1. *The process of setting up the PetaLinux platform*

4.3.1 Platform Selection

When the necessary tools have been installed, PetaLinux can be configured for the build process. The first step is to select an existing platform or to add a new one.

There are reference designs for some of the most common boards, but not for the SP601 board, which is used in this thesis. Adding a new platform can easily be done using a script that comes with the PetaLinux tools.

4.3.2 Configuration

After selecting the correct platform, it is possible to access the configuration menu, where kernel settings and user settings can be adjusted. User settings includes for example the option to add and configure libraries and applications included in the PetaLinux distribution.

4.3.3 Hardware Creation

At this point, the EDK hardware project containing the MicroBlaze processor needs to be included in the PetaLinux source tree. There are also some configuration files in the project folder that needs manual editing, and FS-Boot needs to be included in the hardware project.

FS-Boot is a small, first stage bootloader for the MicroBlaze processor and is used for serial downloading and booting of a larger bootloader with more features.

When the hardware project has been properly configurated, the user can generate and download the configuration bitstream to the FPGA through EDK.

4.3.4 Building PetaLinux

PetaLinux includes a script called AutoConfig, which is used for propagating the settings from a hardware project to the PetaLinux configuration. After running the script, the build process can be started by issuing a command.

The build process will result in Linux and U-Boot images in a few different formats.

U-Boot is an open source bootloader with support for multiple platforms, including the MicroBlaze processor. It can be used for downloading the linux image through serial or ethernet connection, and for programming the flash memory.

4.3.5 Booting PetaLinux

As soon as the bitstream has been downloaded to the FPGA board, it will boot into FS-Boot. It is then possible to interact with the board through the serial connection between it and the host PC.

The next step is to download the U-Boot image through the USB connection to the board. When this download is complete, U-Boot can be started, allowing download of the complete PetaLinux image to the primary memory or flash.

Once this download is also completed, PetaLinux can be run on the board, using the terminal program running on the host PC as a command prompt.

Chapter 5

Implementation

This chapter describes the course of work and the decisions that were made during it.

5.1 Research

The research phase began with learning more about the DLNA standard, since the intentions initially were to implement a DLNA Media Renderer. There turned out to be a lot of licensing costs connected to DLNA. After discussing the issue with Pär Gunnars Risberg, the supervisor for this thesis, the descision was made to implement the renderer according to the UPnP standard instead.

The following weeks were spent reading up on the UPnP standard and examining the different operating systems available for the MicroBlaze architecture. There these two main options were considered; PetaLinux and BlueCat Linux. PetaLinux turned out to be available as a student version, free of charge, which was more or less of a necessity[15]. Since PetaLinux comes with a script for automatic configuration, it would most likely be the easier of the two to get it up and running on the MicroBlaze[14].

During this time, WLAN chips were also looked into. In order to establish a wireless network connection, other components than just the chip are needed. The W2SW0001 module includes all RF circuits needed[23], which would save a great deal of design time.

The plan was to find and use open source software and frameworks as much as possible to reduce the need to implement protocols and standards. Thus, the next step was to look into existing applications that resembles the needs for this thesis, fucntionality wise. The problem was divided into two areas; UPnP communication and audio playback.

5.2 Software Development Kits (SDK) for UPnP

A UPnP library to base the renderer on was needed, and the first thing to look for. There are quite a few different SDKs available. Some of them are commercial and/or focused on a specific platform, which makes them less suitable for this project. The following subsections will describe the SDKs which seemed worth considering.

5.2.1 Libupnp

The first library that was considered was libupnp. It is used in a lot of different applications, both for media servers, but also in other types of applications where port forwarding or other non-media functionality is required[9].

Although there are a few media server implementations based on libupnp, no examples or samples of media renderer implementations could be found. This, combined with that it is poorly documented, are the major disadvantages with libupnp.

5.2.2 Intel Developer Tools for UPnP

The Developer Tools for UPnP is, as the name strongly suggests, a collection of tools that are useful when creating UPnP compatible software. In addition to these tools, which are useful for testing and debugging, a UPnP stack and sample devices can be created.[20]

This framework has few dependencies[21] and seemed like a good choice. A sample device was generated and compiled, intended to act as an empty shell of a media renderer. The sample device offers very minimal functionality, but it has the ability to receive and send replies to UPnP commands.

Unluckily, none of the control points that were available for testing with were able to detect this device, although it could be detected by the UPnP Developer Tools. The renderer responded to UPnP commands sent to it using the same tools.

5.2.3 Platinum UPnP

Platinum UPnP is a library that is written in C++ which can be compiled and run on a variety of platforms, for example Windows, Linux, iOS and Android. It is used by applications such as XBMC, doubletwist, Asset UPnP and Boxee. Some of these applications have media server functionality, some have media renderer functionality and some have both.[17]

There are sample applications available, providing a way to get started with building a UPnP device. No useful documentation on how to use the library could be found, but by examining the samples and looking at the source code for XBMC, a fairly good understanding of the SDK could be reached.

A sample Media Renderer was compiled and run on the desktop computer, with good results. It was detected by all control points that were tested and could recieve commands that were sent to it using the debugging tools from Intel.

5.3 Media Decoding

Since the UPnP standard does not include decoding, the UPnP device needed to be interfaced with an audio player device. The media playback alternatives that were considered are presented below.

5.3.1 Libmad

Libmad is a very commonly used decoding library for MPEG audio streams[8]. Even though it is a good library, the documentation turned out to be lacking. Because of that, the descision was made to move on to other alternatives.

5.3.2 GStreamer

GStreamer is a multimedia framework that is built in a modular fashion, which gives the advantage of being able to add whatever decoding formats one could need as different modules[19]. It is used by a wide variety of applications and is well documented[18].

5.4 Desktop Renderer Application

When it had been decided that PetaLinux would be used as the operating system, PetaLogix was enquired for a student version to use in this thesis. It normally takes 3 to 4 weeks to get a response[15], and it took over a month before it acually did arrive. While waiting for the operating system to arrive, a prototype renderer was developed for another platfrom, which was intended to be ported to PetaLinux once it would be up and running. A PC supplied by Actiwave AB, running Ubuntu 9.10, was instead used as the development platform.

As a starting point, the functionality-lacking sample media renderer based on Platinum UPnP was used.

Gstreamer was used as a media framework because of its modularity and documentation. Since GStreamer is written in C and Platinum UPnP in C++ it initially seemed like there would be problems integrating them, but it could be solved by adding a header to the C files that needed to include from the C++ files.

For starters, the renderer would take a URI given by the control point and start playback of it through a command line call. This simple test program had one problem though; the whole program ran as one single process, so when the renderer sent the command to the player to start playing a track, the program got stuck and could not receive any further commands until the track was finished.

After studying the GStreamer Application Development Manual[24], threading was introduced, which solved the problem of the program getting stuck. More functionality was also added to the renderer, for example volume control, pausing, resuming and skipping to a specific point in a track.

A state were reached where the application worked really well with all the available control points; a laptop using the PlayTo functionality in Windows 7,

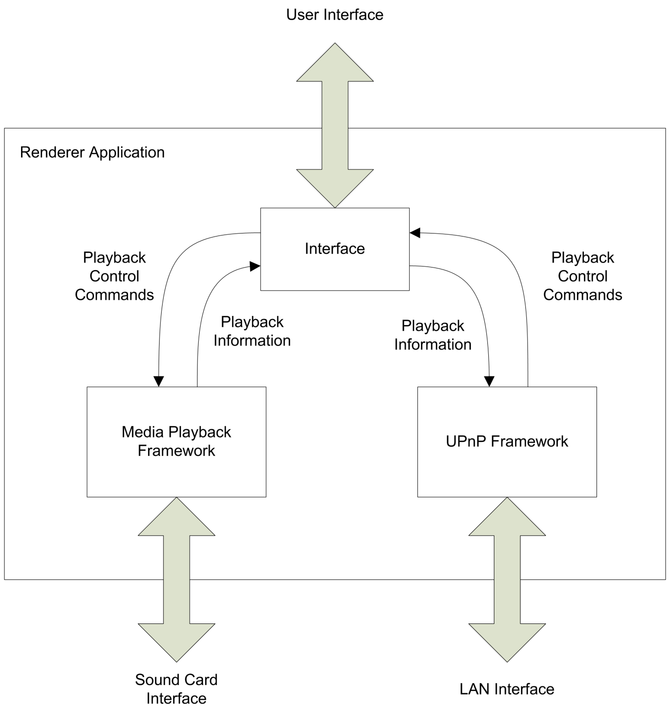

Figure 5.1. *Structure of the renderer application*

an android phone using an application called Andromote and an Iphone using an application called PlugPlayer. The renderer responed to commands as it should, which could be seen using the Device Sniffer, which is one of the UPnP Developer Tools.

The structure of the desktop application is shown in figure 5.1.

Since the renderer was intended for a commercial product in the long run, the way it appears to other devices needed to change; for example name, model, and icon. Most of these changes were easily done by changing some text strings in the renderer implementation. Changing the icon was more tedious. There is a function in Platinum UPnP that is supposed to set an image file as icon for a UPnP device. Unfortunately it did not work. Instead, the icon was converted to byte format and included in a data file.

The size of the desktop application executable file is 2.6 MB, which is a bit big for an embedded system. The smaller size, the smaller flash memory can be used in the end-product. Since the target platform board has a 16 MB flash memory, it is not big enough to be a problem in the scope of this thesis, though.

5.5 MicroBlaze Configuration

As already mentioned, the desktop application was developed while waiting for the operating system to arrive. When it finally did, the desktop application was almost finished. The work was split up, and the bring-up process was started in parallel with finishing the desktop version of the renderer.

The first step in the bring-up process was to make a basic MicroBlaze design by simply stepping through the Base System Builder Wizard in XPS, selecting all the default settings. Unfortunately this turned out to be an invalid processor for running PetaLinux, because of system requirements that are not mentioned in the PetaLinux User Guide. A timer had to be added to the MicroBlaze processor. The timer also had to be configured in a specific way. On top of that, the ethernet controller and the serial port had to be configured to use interrupts. It was a hard to find fault, because it resulted i build errors and not just in PetaLinux malfunctioning, as expected. Figuring out what was wrong was achieved by many hours of browsing and searching the PetaLinux mailing list for topics where other users had experienced similar problems.

The rest of the hardware configuration was more straightforward. By following the steps in the PetaLinux User Guide[14] FS-Boot was added to the hardware project the MicroBlaze configuration propagated to the PetaLinux build, using the Autoconfig script. For more details on the bringup process, see section 4.3.

Changes were made to the MicroBlaze during the course of the project. They will be discussed later in this chapter. The final version of the MicroBlaze includes an MMU and a LED control interface. The LEDs are for testing purposes and can be removed if not needed, to reduce the utilization of the FPGA slightly. No FPU was used, because it used up far too much logic, and was discarded early on. This MicroBlaze configuration includes 16 KB instruction cache memory, and 16 KB of data cache memory. The clock frequency is 75 MHz. Table 5.1 shows the utilization of FPGA logic for the final version of the MicroBlaze.

Hardware	Used	Available	Utilization [%]
Slice registers	4624	18224	25
Slice LUTs	5938	9112	65
Occupied slices	2016	2278	88
Block RAMs	26	32	81

Table 5.1. FPGA logic utilization in the MicroBlaze.

5.6 Configuring, Building and Booting PetaLinux

Once the hardware platform was configured, getting PetaLinux up and running on the board went fairly smooth. Starting out with a basic configuration of PetaLinux, the biggest problem was installing and configuring Kermit, which is recommended by PetaLogix for the communication over the serial port[14]. Kermit is a file transfer and terminal emulation program developed at the Columbia University[22]. What needed to be done was to change the read/write permissions for the USB port, which had strange default values.

Once Kermit was configured, the PetaLinux image was downloaded to the board through the serial connection. When trying to boot it though, only the first few characters appeared, and then the booting crashed. It turned out that part of the MicroBlaze configuration needed to be changed. Settings like baud rate and parity for the ethernet IP block needed to match the ones in Kermit.

PetaLinux was then rebuilt and downloaded to the board again. This time it booted correctly, and Kermit could be used to send system commands.

5.7 PetaLinux Renderer Application

With PetaLinux up and running, it was time to start working out how to port the desktop application to the intended platform. The first step was to include the libraries that had beem used (Platinum UPnP and GStreamer) in the PetaLinux image. This proved more of a challange than what was anticipated.

Platinum UPnP is built using Scons, a Python-based software construction tool, while PetaLinux uses a different tool, make. Adding libraries to PetaLinux means creating a make-file that builds the library, then adding the library to the configuration menu and the PetaLinux make-file, so that it is included when the image is built[6]. Also, to add dynamic libraries, it turned out that an MMU is needed. The first basic design did not include one, since the hardware utilization needed to be kept as low ass possible. Another problem was that Platinum UPnP is written in C++, which is supposed to be supported by PetaLinux. This support is poorly documented though. If there is a way to compile Platinum UPnP for the target system, it was never found.

Gstreamer has dependencies on two other libraries, Liboil and Glib. Before Gstreamer could be built, they needed to be added to PetaLinux. Adding Liboil worked out, but Glib turned out to be trouble. There were not any configuration that supported the MicroBlaze architecture. Perhaps the library could have been built with some changes to the code, but it would have taken a lot of time, and still would not guarantee any success. This led to the conclusion that another UPnP framework was needed, and another media decoder as well, preferably with as few dependencies to other libraries as possible.

The solution was to give the UPnP tools from Intel another chance. They can generate code for a small UPnP stack and a sample renderer, written in C. It had been tested earlier on the desktop PC, when comparing different UPnP frameworks. This approach removed the need to go through the tedious process of adding a UPnP library, since the whole functionality could be achieved by adding a few C-files and including them in the renderer application. Initially there were problems compiling the renderer, because it required OpenSSL, which would not compile for PetaLinux. This problem was solved by reconfiguring the UPnP stack to not use encryption, which is what OpenSSL is used for. The resulting sample renderer could be detected by another computer on the network, using the Intel UPnP debug tools. It could also recieve and respond to UPnP commands sent using the same tools. The size of the renderer executable file is 259.1 KB, which is almost 10 times less than the desktop application executable. It does not make much of a difference for the result of this thesis, but it is better suited for embedded systems in more general terms.

As for the decoding, it turned out there is an mp3 playback application included in PetaLinux, called mp3play. This opened the opportunity to play mp3 files in the PetaLinux file system, but also those shared by a media server, as mp3play can take an URI as its input.

Since the platform did not have any sound card or other means of outputting the audio, the output was instead displayed on LEDs on the FPGA board. This was made possible by modifying the mp3play application to use an output mapped to the LEDs, instead of the default linux audio output. The size of the modified version executable is 271.7 KB, which is in the same range as the previous version.

To find out which bitrates the processor could to decode in real-time, a few tracks with different bitrates were tested, and the decoding time was measured. Regular music files were used. Because the files had to be bundled with the PetaLinux image, only small files were used for playing locally, to save time. By experimenting the best possible cache configuration was found. The more cache memory in the MicroBlaze configuration, the better performance, but it did not make a major impact. The tables 5.2 and 5.3 shows the results from the measurements with the best cache configuration. They clearly show that this configuration of the MicroBlaze processor, and thus any configuration, does not deliver enough performance to decode high bitrate mp3 files in real-time, using mp3play for decoding.

At this stage, the time had come to terminate the project.

File Description	Media Length [s]	Decoding Time [s]
test1.mp3, 1.2 MB, 22050 Hz, 62 kbps, Joint Stereo	155	86
test2.mp3, 1.3 MB, 44100 Hz, 192 kbps, Stereo	56	61

Table 5.2. Decoding times for local files.

File Description	Media Length [s]	Decoding Time [s]
test1.mp3, 1.2 MB, 22050 Hz, 62 kbps, Joint Stereo	155	90
test2.mp3, 1.3 MB, 44100 Hz, 192 kbps, Stereo	56	81
test3.mp3, 3,9 MB, 44100 Hz, 160 kbps, Stereo	202	238
test4.mp3, 5,6 MB, 44100 Hz, 128 kbps, Joint Stereo	366	390

Table 5.3. Decoding times for files streamed via network.

Chapter 6

Results

This chapter presents the results that have been achieved during the thesis.

6.1 Prototype Desktop Application

A media renderer application for desktop Linux computers was made, which worked satisfactorily well. All desired functionality was successfully implemented, including skipping to a specific point in a track and handling different audio formats. The desktop renderer application is compatible with different control point implementations in Windows 7, Android and iOS.

6.2 PetaLinux Bring-up

A hardware platform based on MicroBlaze was configured and synthesized on the FPGA. PetaLinux was successfully configured, built and run on that hardware platform.

6.3 PetaLinux Application

A simple form of media renderer was implemented for the PetaLinux platform, that can be discovered on the network and receive UPnP commands.

The possibility for audio playback was also evaluated. Media files in mp3 format were successfully streamed and decoded. Due to the lack of an audio card, the output was displayed on the LEDs available on the target board. The performance was not good enough to decode high quality mp3 files. Measurements of the performance can be found in tables 5.2 and 5.3.

Chapter 7

Discussion

This chapter analyzes the results and discusses the biggest problems that occured during the project, as well as the reasons why certain goals were not reached.

7.1 Problems

There were a lot of issues that cost a lot of time during this project, and these were some of the major ones.

7.1.1 Resource Availability

PetaLinux was not available from the beginning, which meant that the application had to be developed for another platform to begin with. When PetaLinux finally was working on the FPGA, the UPnP application was already implemented, with only minor tweaks left to do. It turned out though, that the application was not fit to be ported to PetaLinux, both because of dependency issues, and because of compiling problems. It ended up in a situation where there was no point to continue on, but instead made more sense to start over with a new application based on a completely different framework. With PetaLinux available from the start, this problem would have been discovered far earlier. If so, a lot of time could have been spent on the UPnP Tools based solution instead.

7.1.2 Lacking Documentation

Setting up PetaLinux was a challenge of its own, since the manual was incomplete at the time. For example, PetaLinux needs certain timers and interrupt flags to be included in the design of the MicroBlaze soft processor, but it was not mentioned in the manual. It took several weeks to figure that out the hard way. This is partly the reason why PetaLinux was not up and running until very late in the project, which in its turn was an issue, as mentioned above.

Since June 16th 2010, a bring-up guide is available[16]. This guide includes the requirements on timers and interrupt flags[11], and besides it is much easier to follow than the PetaLinux user guide. If that would have been available during the bring-up process, a lot of time and effort could have been saved.

7.1.3 Slow File Transfer

We did not have full network administrative rights on the work station that were used for programming and communicating with the FPGA. Due to this, TFTP could not be used for file transfer to the board. Instead the PetaLinux image file had to be downloaded through a standard UART interface, which made the process unnecessarily slow. One transfer could take anything from 25 minutes to 45 minutes, depending on the size of the image. Since there were a lot of experimenting with different settings and different content in the image file, the progress was hampered greatly by constant stops.

7.2 Objective Fullfilment

This section will discuss the objectives, and why some of them were not fulfilled.

7.2.1 Rendering Application

The rendering application was what took the most time. There were lots of different options to evaluate, and there were not time for more than scratching on the surface and pick the one that seemed to be the best. That early on in the project, it was hard to tell if one framework would be better than any other, so the idea was to go for something with plenty of coding examples, for as quick and easy understanding as possible. Platinum UPnP matched this attribute very well, so we decided to use it as a starting point for development of the renderer.

Since PetaLinux was not available to begin with, developing for the target platform was simply not an option. This meant that the application had to be developed on an alternative platform. Ubuntu 9.10 was selected as a target platform, because we though that porting an application from one linux based system to another would not be so hard. Also, a station with that operating system was available. The Ubuntu application, when finished, worked really well.

Getting PetaLinux up and running was not a trivial matter. When the Ubuntu application was done, PetaLinux had been successfully booted on the desired platform, the MicroBlaze soft processor, so it was time to port the application. We were aware that the performance could be an issue, since the MicroBlaze is so much slower than a full-blown desktop computer, but we never anticipated that compatability would be such an issue as it would turn out. The reason was a misconception that PetaLinux would have all basic system components available that Ubuntu has. It turned out though, that some very important things were missing, or existed in slim versions, without full functionality. A lot of different approaches were tried, like reconfiguring libraries, and adding new libraries to the PetaLinux structure, but nothing of it gave any results.

Another framework was needed. One that was less dependent on other libraries and system components. A skeleton renderer application was generated with UPnP Developer Tools. Due to lack of time left, building a rendering application with full functionality was not an option. However, compiling a basic renderer was possible to do fairly quick. It was successfully run on PetaLinux, and it could be detected with other devices on a local network. It was not a fully working application, but it was at least a proof of concept. Time was running out, there was not that much to be learned from implementing a UPnP and media decoder interface again.

7.2.2 Playback

The Ubuntu application has full playback functionality. It was never implemented into the PetaLinux version, but it was possible to stream mp3 media via a local network to the decoding application mp3play, which was running on the FPGA. The performance was however far from acceptable, which was yet another reason not to finish the PetaLinux application. Selection of decoder is a design choice that affects the performace, together with a lot of other variables. Improving the performance to good enough levels is a thesis work of its own. (Such a thesis was even initiated in January 2011 by two other students.) Because of this, it was better to leave the choice of decoder open, and thus, terminate the project at this stage.

7.2.3 Control Functionality

The Ubuntu application has a rich control functionality including commands like volume control, pause and play. The same would be possible to implement into the renderer for PetaLinux, but just as with the playback, the decoder needs to be selected in order to start this work.

7.2.4 WLAN Chip and Interface

Some initial search for WLAN chips was done. Since time ran out before a fully working renderer could be implemented on the FPGA, the stage of interfacing it to the FPGA was never reached.

Chapter 8

Conclusions and Future Work

This chapter summarizes the conclusions drawn from this thesis work and presents the ways its results can be developed in the future.

8.1 Conclusions

When implementing a UPnP Media Renderer on a small operating system such as PetaLinux, it is desirable to use a framework that is as complete as possible. Dependencies on system functions and other libraries should be avoided if possible. For a desktop computer application, GStreamer and Platinum UPnP are well suited to use as a basis, as they offer many nice features. They are not well suited to run on PetaLinux though. The UPnP stack that can be generated with UPnP Developers tools only depends on OpenSSL, and that dependency is possible to remove. It is a much better choice for embedded applications such as the one discussed in this thesis.

To decode audio in real-time on a Spartan-6 MicroBlaze soft processor is not trivial. When using mp3play as a decoder, the performance is insufficient for decoding high quality mp3 music files, even with an optimal cache configuration.

8.2 Future Work

For the device to be a finished product that can be used commercially, more work needs to be done.

8.2.1 Decoding

The performance needs a major boost. There are quite a few routes one could take, but all of them have different advantages and disadvantages.

The first and most obvious way would be to hardware accelerate critical parts of the decoding, for example the discrete cosine transform when it comes to mp3 decoding. Doing so gives a major performance boost to a specific part of the decoding, but it comes at the cost of extra hardware. This could mean that a larger FPGA needs to be used, or that other parts of the hardware would have to be sacrificed, which could make other parts of the decoding slower. Another downside of hardware acceleration is that it is not as flexible as software. Different sound formats may use different transforms, and thus, an accelerator may only be used for some, or even one sound format. If all commonly used formats are to be supported, then it will most likely not be feasible to hardware accelerate them all.

Another solution would be to only play unencoded audio on the FPGA, for example only stream .wav files. Instead the user would have to have a device from which he or she would stream all audio data, that would use a special software to decode all data to an uncompressed audio format. Such programs already exist, for example Mediatomb, which offers transcoding via plugins and scripts[1]. It is not certain if the standard ethernet interface in the MicroBlaze can handle high enough data rates for this solution, though. The major downside with transcoding is a big loss of flexibility, as the media server has to have transcoding capability, which means that some devices will not work at all, and that the user might have to install extra software on those that works at all.

The decoding software is of course something that one should consider. There are probably decoders available on the market that are more efficient than mp3play. To get the best performance on the software side, one could even write a decoder that is optimized for the MicroBlaze running PetaLinux, or alternatively optimize an already existing decoder for the target platform.

It is also worth to consider using a DSP processor instead of the MicroBlaze, since DSP processors are generally good at handling heavy computational loads. The downside with this is that the intended platform already has an FPGA on it, which then would not be fully utilized. Using a DSP processor means adding cost compared to using the MicroBlaze processor.

8.2.2 UPnP Interface

There are different ways to attack the problem, but the best way is most likely to build the renderer based on the framework from UPnP tools, since it has been confirmed to be compatible with PetaLinux. It would most likely just mean straight forward implementation, merging it with an audio decoder of choice, in the same manner as that has already been done with Platinum-UPnP and GStreamer.

8.2.3 WLAN Interface

A WLAN chip still needs to be interfaced to the MicroBlaze processor. It can be done in many ways, but the easiest would be to go for the Wi2wi solution, since all necessary RF circuits are included. Both SDIO, GPIO and SPI interfaces are available.[23]

8.2.4 Adapting to Actiwave's Platform

Before the renderer can be commercialized, it needs to be integrated into Actiwave's platform. Since there already is a Spartan-6-FPGA on it, a lot of the work is already done. The WLAN chip would need to be integrated into the platform though, and the MicroBlaze would need to be interfaced to Actiwave's sound interface. There may also be licensing issues to consider, since a lot of open source software has been used, for which special terms may apply when used in a commercial product. A license for PetaLinux would also be needed, since a university version was used during this thesis work. It is not allowed to use it commercially.

Bibliography

[1] *MediaTomb Website*. `http://mediatomb.cc/`, February 2011.

[2] UPnP Forum. *ConnectionManager:1*. `http://upnp.org/specs/av/UPnP-av-ConnectionManager-v1-Service.pdf`, June 2002.

[3] UPnP Forum. *UPnP AV Architecture:1*. `http://upnp.org/specs/av/UPnP-av-AVArchitecture-v1-20020625.pdf`, June 2002.

[4] UPnP Forum. *UPnP Device Architecture version 1.0*. `http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf`, October 2008.

[5] UPnP Forum. *What is UPnP?* `http://upnp.org/about/what-is-upnp/`, June 2010.

[6] Analog Devices Inc. *Adding Libraries*. `http://docs.blackfin.uclinux.org/doku.php?id=uclinux-dist:libs:adding_libraries`, August 2010.

[7] Arcturus Networks Inc. *What is uClinux?* `http://www.uclinux.org/description/`, August 2010.

[8] Underbit Technologies Inc. *Applications using MAD*. `http://www.underbit.com/products/mad/#using`, June 2010.

[9] The libupnp project. *Projects Using libupnp*. `http://pupnp.sourceforge.net/#projects`, June 2010.

[10] PetaLogix. *About PetaLogix*. `http://petalogix.com/about`, June 2010.

[11] PetaLogix. *PetaLinux Bringup Guide ver 1.3*. `http://www.petalogix.com/resources/documentation/petalinux_sdk/PetaLinux-BoardBringup.pdf`, November 2010.

[12] PetaLogix. *PetaLinux Development Kit*. `http://petalogix.com/products/petalinux`, June 2010.

[13] PetaLogix. *PetaLinux Installation Guide*. `http://www.petalogix.com/resources/documentation/petalinux/petalinux-installation-guide`, June 2010.

[14] PetaLogix. *PetaLinux User Guide.* `http://www.petalogix.com/resources/documentation/petalinux/userguide`, June 2010.

[15] PetaLogix. *PetaLogix University Program.* `http://petalogix.com/univ`, June 2010.

[16] PetaLogix. *PetaLogix News.* `http://www.petalogix.com/news`, February 2011.

[17] Plutinosoft. *Platinum UPnP.* `http://www.plutinosoft.com/platinum`, 2006.

[18] GStreamer project. *GStreamer Applications.* `http://gstreamer.freedesktop.org/apps/`, June 2010.

[19] GStreamer project. *What is GStreamer?* `http://gstreamer.freedesktop.org`, June 2010.

[20] UPnP Developer Tools Project. *UPnP Developer Tools.* `http://opentools.homeip.net/dev-tools-for-upnp`, June 2010.

[21] UPnP Developer Tools Project. *UPnP Stacks.* `http://opentools.homeip.net/dev-tools-for-upnp/stacks`, June 2010.

[22] Columbia University. *The Kermit Project.* `http://www.columbia.edu/kermit/`, August 2010.

[23] Wi2Wi. *W2SW0001 Product Brief.* `http://www.wi2wi.com/products/datasheets/W2SW0001%20PB%20rev1.2.pdf`, 2008.

[24] AndyWingo Ronald S. Bultje Wim Taymans, Steve Baker and Stefan Kost. *GStreamer Application Development Manual.* `http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/index.html`, June 2010.

[25] Xilinx. *MicroBlaze FAQ.* `http://www.xilinx.com/products/design_resources/proc_central/microblaze_faq.pdf`, April 2009.

[26] Xilinx. *SP601 Product Brief.* `http://www.xilinx.com/publications/prod_mktg/sp601_product_brief.pdf`, 2009.

[27] Xilinx. *SP601 User Guide.* `http://www.xilinx.com/support/documentation/boards_and_kits/ug518.pdf`, June 2010.

[28] Xilinx. *Spartan-6 Family Overview.* `http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf`, August 2010.

[29] Xilinx. *ISE Design Suite: Embedded Edition.* `http://www.xilinx.com/tools/embedded.htm`, February 2011.

[30] Xilinx. *Xilinx Platform Studio (XPS).* `http://www.xilinx.com/tools/xps.htm`, February 2011.