Customizing And Building The Linux Kernel To Control Appliance Actuators And Sensors In Domotics

Steven M. Hannah

North Carolina Agricultural and Technical State University

CUSTOMIZING AND BUILDING THE LINUX KERNEL TO CONTROL APPLIANCE ACTUATORS AND SENSORS IN DOMOTICS

by

Steven M. Hannah

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Department: Electrical and Computer Engineering
Major: Electrical Engineering
Major Professor: Dr. Christopher Doss

North Carolina A&T State University
Greensboro, North Carolina
2011

School of Graduate Studies
North Carolina Agricultural and Technical State University

This is to certify that the Master's Thesis of

Steven M. Hannah

has met the thesis requirements of
North Carolina Agricultural and Technical State University

Greensboro, North Carolina
2011

Approved by:

Dr. Christopher Doss Major Professor	Dr. Corey Graves Committee Member
Dr. Alvernon Walker Committee Member	Dr. John Kelly Department Chairperson

Dr. Sanjiv Sarin
Interim Dean of Graduate Studies

BIOGRAPHICAL SKETCH

Steven M. Hannah was born on June 27th, 1986 in Richmond Virginia. He received his Bachelor of Science in Electrical Engineering from North Carolina Agricultural and Technical State University in May of 2009. He is a currently a candidate for the Master of Science in Electrical Engineering. He has presented research and had publications in the conferences including the 2007 and 2008 Opt-Ed Alliance day in Greensboro, North Carolina, NCUR conference 22 in Salisbury, Maryland, and the 2009 FCCM IEEE Symposium in San Francisco, California.

ACKNOWLEDGEMENTS

I would like to sincerely thank my advisor Dr. Christopher Doss and my committee members Dr. Corey Graves and Dr. Alvernon Walker for keeping me focused and motivating me when I thought all was lost. I would also like to thank my family for their everlasting and undying love and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

1. American With Disabilities Act – ADA
2. Socially Assistive Agent and Domotics – SAAD
3. Field Programmable Gate Array - FPGA
4. Xilinx Platform Studio - XPS
5. Intellectual Property - IP
6. (Very High Speed Integrated Circuit) Hardware Description Language - VHDL
7. Virtual Machine - VM
8. GNU Compiler Collection - GCC
9. Configurable Logic Block - CLB
10. Programmable Logic Device - PLD
11. Programmable Logic Array - PLA
12. Programmable AND-array Logic - PLA
13. Simple Programmable Logic Device - SPLD
14. Complex Programmable Logic Device - CPLD
15. Very-Large-Scale Integration - VLSI
16. Application Specific Integrated Circuit - ASIC
17. Input/Output Block - IOB
18. International Organization for Standardization - ISO
19. Embedded Linux Development Kit - ELDK
20. Open Source Linux - OSL

21. Board Support Package - BSP

22. Least Significant Bit - LSB

23. Light Emitting Diode - LED

24. Random Access Memory - RAM

25. General Purpose Input/Output -GPIO

ABSTRACT

Hannah, Steven M. CUSTOMIZING AND BUILDING THE LINUX KERNEL TO CONTROL APPLIANCE ACTUATORS AND SENSORS IN DOMOTICS. **(Advisor: Christopher Doss)**, North Carolina Agricultural and Technical State University.

The human population has been increasing ever since the end of the Bubonic Plague in the 1300's. The global human population is getting older as well as increasing. This is because on average people are living longer. Among the reasons for the drastically improved quality of human life are advances in technology and medicine. The possibility of developing a disability increases along with a person's age. There are numerous available options for the disabled elderly, including nursing homes, in-home care and living with other disabled people. A person may decide not to exploit any of these options and "age in place". This thesis will explain the use of currently existing technology in a "smart house" subproject that may provide a possible alternative to the current options that the elderly disabled have, and how the Linux kernel is modified, built and downloaded onto the Ml405 Evaluation Platform to control appliance actuators and sensors.

CHAPTER 1

INTRODUCTION

As the human population continues to grow and age, the question arises of how the disabled elderly will be cared for. Among the current options available for this growing age group are nursing homes and in-home care. These options may not be affordable to an elderly disabled person or his or her family. A possible low-cost alternative to the currently available options may be home automation.

Smart house technology involves the concept of automating a home. This idea has been explored for many years. There are some requirements for a home to be labeled “smart”. A smart home can range from having those basic requirements to having much more sophisticated abilities.

The Socially Assistive Agent and Domotics (SAAD) project is a prototype smart home proposed by the Electrical and Computer Engineering, Computer Science, Nursing, and Psychology departments at North Carolina A&T State University. The proposed project involves meeting a subject’s physical needs as well as engaging the subject emotionally. SAAD is composed of many subprojects, including wireless communication to control actuators and sensors. This subproject was named WireAct. The primary goal of WireAct was to successfully customize, build, and download the Linux 2.6 kernel to an FPGA board to interface with its hardware peripherals.

Linux is a versatile, open source operating system. Originally created to be a terminal emulator, Linux has expanded to being used for all types of applications including games and web servers. The Linux directory structure is fairly organized, containing a root directory and numerous subdirectories under root.

FPGAs are considered programmable logic. They are capable of being configured to perform virtually any task. There are several vendors as well as several families of FPGAs, including the Virtex 4 ML405 which was used for the purpose of performing the experiments involved in WireAct.

The chosen method for conducting WireAct's experiment involved obtaining the Linux 2.6 kernel from http://xilinx.wikidot.com. The kernel was then built, downloaded, and run on the ML405 board. The simplest and most straightforward way to build the Linux kernel was to create a Linux environment, so a virtual computer was created to run a Linux operating system. The Linux kernel could then be built inside the virtual computer and downloaded to the FPGA board by the host computer.

The default Linux kernel wasn't sufficient to perform the task of controlling any external hardware. Creating custom executable programs and adding them to the kernel's ramdisk was a feasible way to customize the Linux kernel. Additions to the Linux kernel include programs to print messages on a console application, activate LEDs on the ML405 board, and enable or disable external motors connected indirectly to the board.

Chapter two discusses the human U.S. and world populations. Chapter three covers the basics and different levels of smart house technology. The SAAD project is discussed in chapter four. Chapters five and six cover the history and an overview of

Linux and FPGAs. Chapters seven and eight cover the procedures of the experimental setup and results. Chapter 9 covers the conclusion.

CHAPTER 2

THE HUMAN POPULATION

The global human population is steadily increasing. According to anthropologist studies, the human species dates back at least 3 million years. Humans lived as hunters and gatherers for most of our history. Due to this dubious way of life, the total population remained relatively small, probably numbering no more than 10 million. An increase in human population occurred when agriculture was introduced, which enabled communities to evolve and sustain more people. Due to the steady rate of growth, the world population increased to about 300 million by 1 A.D. As recently as the last 50 years, the world population has multiplied more hastily than ever before, and growth is only expected to expedite in the future. A graph showing the world population's growth can be seen in Figure 1 [1]. As shown in Figure 1, the human population has risen exponentially ever since the Bubonic Plague which killed roughly 25 million people (1/3 of Europe's population) in the 1300's. Contributions to the substantial increase in population growth following the Bubonic Plague include the Industrial Revolution in the 18th century, and even World War II, when less advanced countries' populations began to increase dramatically [2, 3].

2.1 The U.S. Population

In 1790, which was the year of the first census of the U.S. population, there were roughly over 3.9 million people in the United States. By 1900, the U.S. population had jumped to almost 76 million people. Just 100 years later, the 2000 census counted 281.4 million people in the United States.

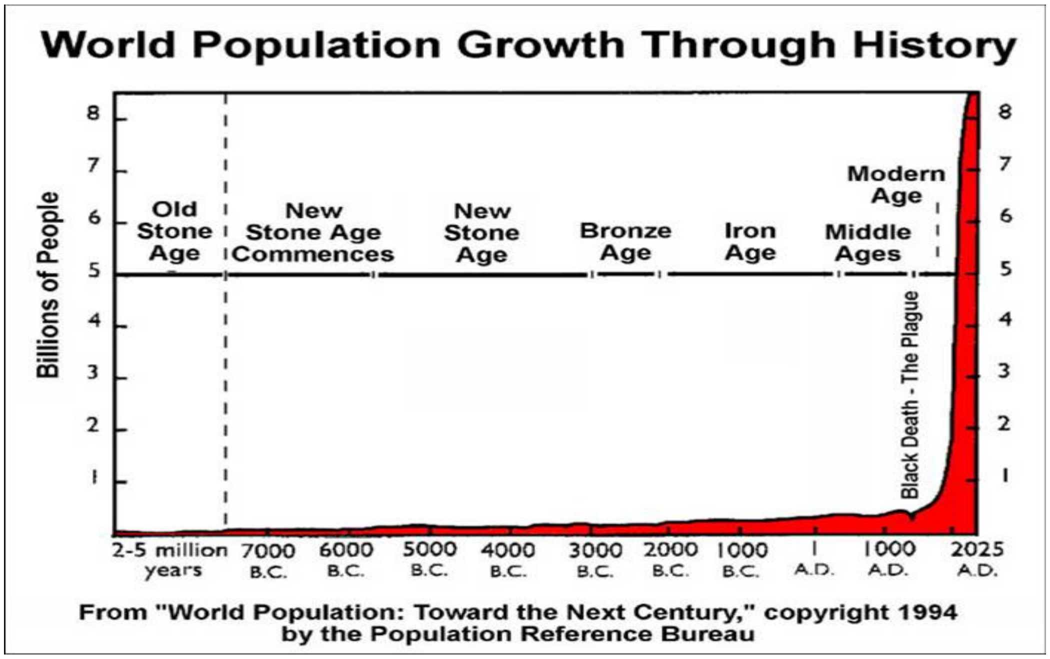


Figure 1. World Population Growth Throughout History

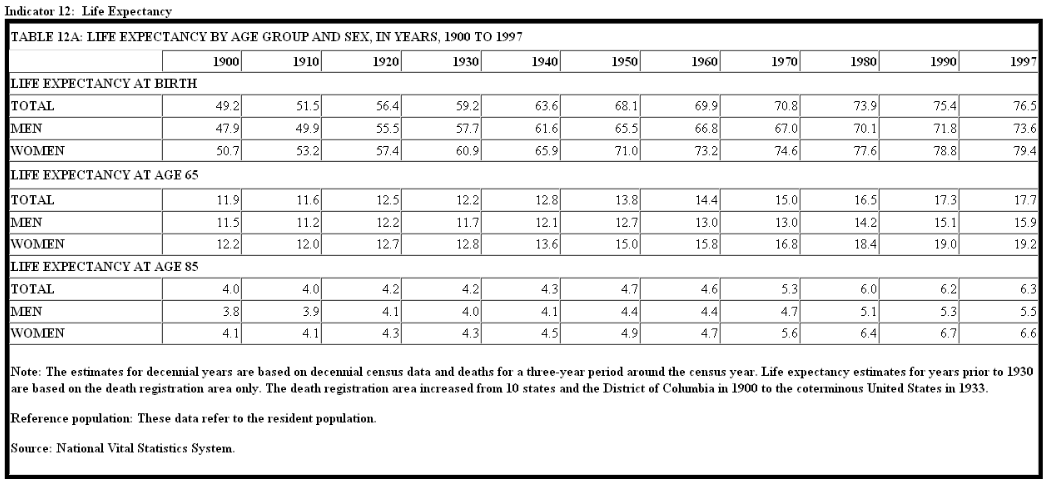
As of January 16, 2010, the U.S. population is projected to be roughly 311.9 million, with a net gain of one person every 15 seconds. Of the world population, the U.S. is ranked third in population, only behind China and India [4, 5, 6].

2.2 The World Population

The global population has skyrocketed as well. For a long time the population didn't grow significantly, as a result of times of growth followed by times of decline. By 1750, the world population is estimated to be 791 million, with 64% in Asia, 21% in Europe, and 13% in Africa. By the beginning of the 20th century, the global population had more than doubled to 1.7 billion. An increase of 53% was seen as the world population increased to 2.5 billion in just 50 years. Over the next half a century the global population increased even quicker. By 2000, an estimated 6.1 billion people populated the world. As of January 16, 2011, the world population is approximated at about 6.9 billion [5]. Based on these facts, the world population is approximately 9 times larger than it was 250 years ago [5, 7].

2.3 Average Life Expectancies

An increase of average life expectancies can be recognized as a direct contribution to the exploding population in the United States as well as the rest of the world. At the beginning of the 20th century, the average life expectancy for a U.S. male citizen was 47.9 years and 50.7 years for women. Figure 2 shows the average life expectancies from 1900 to 1997 for newborns as well as at ages 65 and 85. Figure 2 illustrates that both genders in every age group have seen an increase in life expectancy. For example, a man who was 65 in 1997 could expect to live approximately another 16 years, instead of another 11.5 at the turn of the 20th century. The life expectancy of some age groups had increased by nearly 60 percent during the 20th century. Females have a higher average life expectancy than males for several reasons. Males have a higher mortality rate than females in every age group. Men generally consume more tobacco, alcohol and drugs than females which make them more susceptible to diseases such as lung cancer, tuberculosis, and cirrhosis.



Indicator 12: Life Expectancy

TABLE 12A: LIFE EXPECTANCY BY AGE GROUP AND SEX, IN YEARS, 1900 TO 1997

	1900	1910	1920	1930	1940	1950	1960	1970	1980	1990	1997
LIFE EXPECTANCY AT BIRTH											
TOTAL	49.2	51.5	56.4	59.2	63.6	68.1	69.9	70.8	73.9	75.4	76.5
MEN	47.9	49.9	55.5	57.7	61.6	65.5	66.8	67.0	70.1	71.8	73.6
WOMEN	50.7	53.2	57.4	60.9	65.9	71.0	73.2	74.6	77.6	78.8	79.4
LIFE EXPECTANCY AT AGE 65											
TOTAL	11.9	11.6	12.5	12.2	12.8	13.8	14.4	15.0	16.5	17.3	17.7
MEN	11.5	11.2	12.2	11.7	12.1	12.7	13.0	13.0	14.2	15.1	15.9
WOMEN	12.2	12.0	12.7	12.8	13.6	15.0	15.8	16.8	18.4	19.0	19.2
LIFE EXPECTANCY AT AGE 85											
TOTAL	4.0	4.0	4.2	4.2	4.3	4.7	4.6	5.3	6.0	6.2	6.3
MEN	3.8	3.9	4.1	4.0	4.1	4.4	4.4	4.7	5.1	5.3	5.5
WOMEN	4.1	4.1	4.3	4.3	4.5	4.9	4.7	5.6	6.4	6.7	6.6

Note: The estimates for decennial years are based on decennial census data and deaths for a three-year period around the census year. Life expectancy estimates for years prior to 1930 are based on the death registration area only. The death registration area increased from 10 states and the District of Columbia in 1900 to the coterminous United States in 1933.

Reference population: These data refer to the resident population.

Source: National Vital Statistics System.

Figure 2. U.S. Average Life Expectancy (1900 – 1997)

Males also tend to participate in riskier behavior since they are more aggressive and competitive in nature. Men also have a disadvantage from the very beginning. Male fetuses have a higher mortality rate. Babies are conceived at a ratio of about 124 males to 100 females. However, the ratio of those surviving birth is only 105 males to 100 females. At this point, the higher mortality rate of males as opposed to females is somewhat of a balance measure, so that the population of males is about the same as females around the mating age. As of 2010, the average life expectancy is projected to be

78.3 years. Females are still outliving their male counterparts though. The average life expectancy for a man in 2010 is 75.7 years while it is 80.8 for women [8, 9, 10].

2.4 Reasons for Improved Human Life

One reason why human life has improved so dramatically is the advancement of medicine. Medicine has improved more in the past 200 years than in all of its prior history. A major advancement in medicine is the discovery of penicillin in 1928. Penicillin is a group of antibiotics derived from the Penicillium fungi. Penicillin was significant because they were the first drugs that were effective against many diseases that were serious prior to their discovery. Such diseases include syphilis and Staphylococcus infections. Such diseases used to kill thousands of people each year until these antibiotics were found. Other milestones in medicine include the polio vaccine of the 1950's, the accomplishments of organ transplants and heart surgery, and the eradication of smallpox [11, 12].

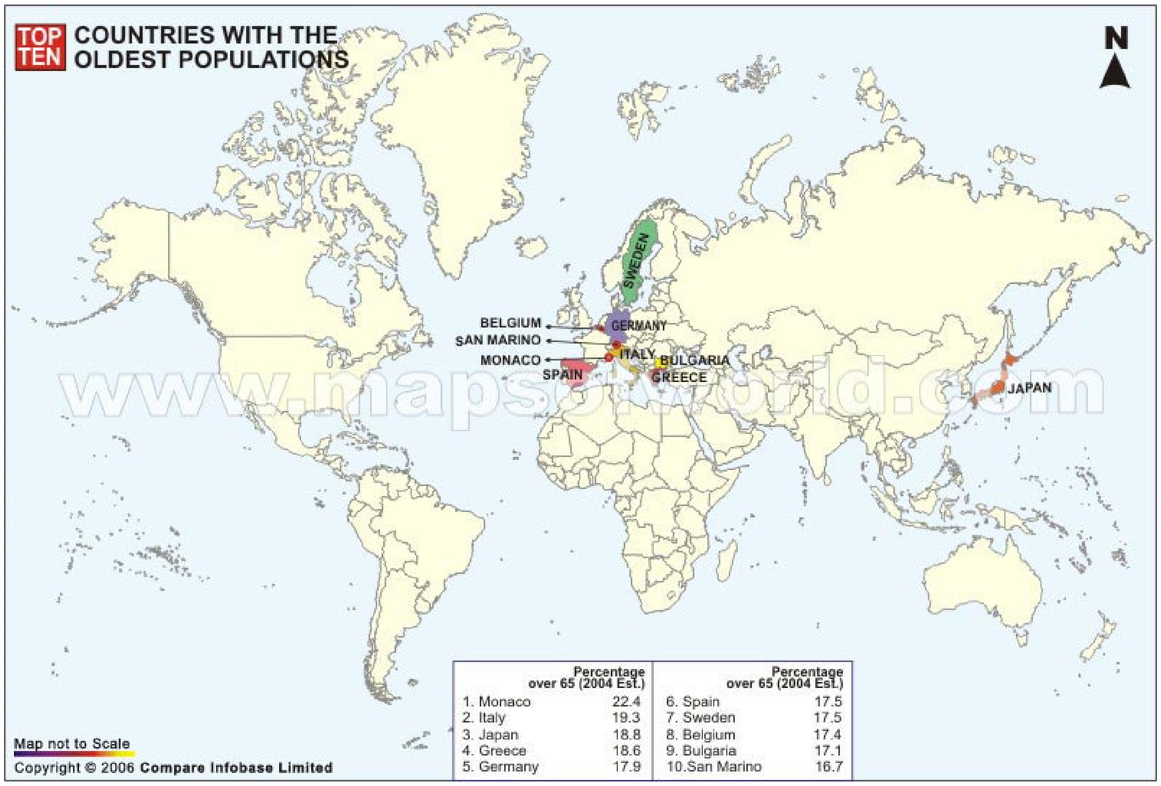
Technology can also be credited as a contribution to the improvement of human life and average life expectancy. Technology helps people in so many ways that are completely subconscious to them. Cooling systems such as freezers and refrigerators allow people to preserve food for longer periods of time. Water can be filtered and is easily accessible virtually anywhere in the U.S. as well as many places around the world. People are able to maintain a comfortable climate in their homes during the cold months of the year thanks to gas or electric heat. Technology in the medical field such as wheelchairs, prosthetic limbs, stethoscopes, CAT scanners and other devices used to diagnose or remedy a condition. Today people may take these types of instruments for granted, but they have greatly improved the overall quality of human life [13].

2.5 The Senior Population

There are many different definitions for “the elderly”. Some definitions include people as young as 50, as does the American Association of Retired Persons. Others may extend as far as the age of 70 (past mandatory retirement age for professors in the United States). Most definitions would consider age 60 or 65 as becoming an “elderly person”. No matter which definition is used, the senior population is increasing not only in the United States but in many other countries around the world. In 2000, approximately 605 million people of the world’s population were 60 years or older. By 2050, that number is expected to be close to 2 billion. If this expectation is met, seniors will outnumber children 14 and under for the first time in history. Figure 3 shows the top ten countries with the oldest populations. The United States has a senior population percentage of about 12 percent as of 2011. The U.S. Census Bureau reported that the dependency ratio (the number of people 65 and older to every 100 people of traditional working ages) is expected to rise from 22 in 2010 to 25 in 2030. That’s a rise from 13 percent of the U.S. population to 19 percent. After 2030 the dependency ratio will rise more slowly to 37 by 2050 [14, 15, 16, 17].

2.6 People With Disabilities

Just like there are several definitions for the “elderly”, there are several definitions of the word disability according to different organizations. The most commonly cited definition is that of the World Health Organization (1976), which actually distinguishes between impairment, disability and handicap. The World Health Organization defines a disability as “any restriction or lack (resulting from an impairment) of ability to perform an activity in the manner or within the range considered



	Percentage over 65 (2004 Est.)		Percentage over 65 (2004 Est.)
1. Monaco	22.4	6. Spain	17.5
2. Italy	19.3	7. Sweden	17.5
3. Japan	18.8	8. Belgium	17.4
4. Greece	18.6	9. Bulgaria	17.1
5. Germany	17.9	10.San Marino	16.7

Figure 3. Top Ten Countries With the Oldest Populations

normal for a human being". There are other definitions, such as the one used by the American with Disabilities Act (ADA), which defines a disability as "a physical or mental impairment that substantially limits one or more major life activities of such individual". Over 54 million people (19 percent of the U.S. population) reported at least some level of disability in 2005. These disabilities included difficulties with mental functioning, emotional functioning and cognitive functioning as well as physical disabilities [18, 19, 20].

As a person's age increases, the probability that he or she will develop at least one disability increases as well. Over 18 million U.S. adults over the age of 64 were disabled in 2005. 13 million of those were labeled as severely disabled. Seventy-one percent of people over 80 were disabled, and 56 percent of those were severely disabled [20].

2.7 Current Options for the Disabled

A nursing home is one of the first options that a person may think of. A nursing home used to be the only cost effective method of assisting the elderly. However, not everyone can afford so send someone to a nursing home. As of 2008, the national average cost of a private room in a nursing home was $212 per day ($77,380 a year). Even a semi-private room was too costly for most people, averaging $191 per day ($69,715 a year). This could be a burden on a working family or even on national healthcare costs. Other choices for the elderly include in-home care or living with friends, relatives or even other disabled people. Households with more than one person can share daily tasks of living by handling the tasks that meet their abilities [21, 22].

2.8 A Possible Alternative in the Future

An elderly person may decide that he or she doesn't want to exploit any of these options but to "age in place". Aging in place is defined as the ability to live in one's own home –wherever that might be – for as long as confidently and comfortably possible [23]. The number of older people who live alone is also on the rise [22]. In 2000, roughly 1/3 of all non-institutionalized elders lived alone [22]. If an elderly person who lives alone happens to be disabled in one or more ways, that person would possibly not have regular assistance in tasks that are difficult or impossible to perform. For those seniors who would like to age in place in spite of his or her possible limitations on daily living, a possible solution may be found in the concept of domotics, or home automation.

Domotics has been explored for decades. For example, the "House of Tomorrow" was featured in the 1933-1934 Chicago World's Fair which was designed with 12 sides, 3 stories, and a wedding-cake shape. The technology depicted in this exhibit included a built-in dishwasher, electric lights with dimmer switches, central air conditioning, an electric garage door opener, and passive solar heating. The House of Tomorrow was unique in design and technology for its time. Most of the technology displayed in this exhibit would be considered ordinary everyday appliances taken for granted in today's world. If this kind of technology was conceived nearly 80 years ago, it should also be conceivable to develop technology that will allow the elderly -especially the elderly disabled- to continue to live in their homes [24].

CHAPTER 3

SMART HOUSE TECHNOLOGY

A smart house is a house that has highly advanced automatic systems for lighting, temperature control, multi-media, security, and many other functions. It appears intelligent because of the computer systems that can monitor many aspects of daily living. For example, a resident may be alerted when his or her favorite TV show is about to start. There may be systems advanced enough to notify a resident that his or her back door is unlocked and lock it if the resident authorizes the system to do so. There are different levels of smart house functions which are organized based on complexity and how long the function has been available [25].

Level one smart house technology offers basic communications. This level technology is necessary for a smart house, but simply having level one technology doesn't solely make it a smart house. Basic communications include the means to communicate with and receive communications from others beyond the home, such as telephones. The internet is also a communication means, and is essential if a home is to be considered "smart" [24].

Level two technology involves responding to simple control commands from within or outside the home. At this level, anything in the house that is electrically powered can be operated using voice commands. Such household items include lights, electric door locks, thermostats, small appliances, and mechanically controlled curtains and windows [24].

Level three involves automating household functions. It is possible to automate certain aspects of one’s home such as when lights go off, when music or TV is turned on, and when security systems are armed or disarmed. Today, products are available that offer more flexibility on this level of automation. Computer-based smart home products allow easier setup for the on-off cycle, and different scenarios can be programmed, such as “weekends at home”, “vacation mode” or “work-week mode”. In some systems, regular cycles can be broken or interrupted easily in the home or through a phone call [24].

Level four technology tracks the location in the home, behavior, and health indicators. If a smart home knows where a person is, it can take appropriate measures in just that room or area instead of the entire house. For example, if the smart house is going to issue a reminder to the resident that his or her favorite television show is about to start, it can do so in the exact room that the resident is sitting or standing. Behaviors that can be tracked include trips to the bathroom or kitchen, sleeping habits and exercising habits [24].

A level five smart home can analyze data that make decisions. This means that a smart home can learn a resident’s normal life patterns. If there is a deviation from these patterns, it could be a sign that something may be wrong and the system can check with the resident or a family member to be sure that the person is well. This level smart house can also learn the resident’s preferences such as light, music and temperature in a specific area of the house. Adjustments can then be made or the smart house can then ask the resident if he or she would like an adjustment to be made [24].

Level six technology involves providing information, reminders, and prompts for daily tasks. Examples of such information are when mail has been delivered, when someone is at the front door, when the stove has been on for too long, or if a resident has forgotten to take medications. For someone with a more severe disability and has difficulty even with tasks such as dressing and grooming, the smart house can prompt the person with voice and visual cues to help him or her with each step of the activity [24].

A level seven smart home has the ability to answer questions. The internet can be used to assist with this level of technology. For questions that a resident cannot answer him or herself, the smart home can accept questions with a voice recognition interface. The interface then goes on the web to seek the answers for the resident. For more personal questions, like “Have I taken my medication this morning?”, the smart house can search its own database for the answer [24].

Level eight smart homes can make household arrangements. Household arrangements can include household repairs or making food shopping lists. Smart homes at this level can arrange such tasks because they have the sophistication to track activities such as a resident's eating and exercising habits [24].

CHAPTER 4

THE SAAD PROJECT

4.1 Overview

Faculty members in the Department of Electrical and Computer Engineering (along with computer science, psychology, and nursing) have proposed the "SAAD" (Socially Assistive Agent and Domotics) project as a response to the interest in finding more feasible alternatives to assist the elderly with aging in place. The main purpose of SAAD is to develop a socially assistive agent to facilitate self-care for an elderly subject in his or her own home, as well as to engage the subject socially and emotionally. Social goals of the SAAD project include:

- Allowing the subject to deal with daily activities and routines,
- Providing companionship,
- Eliciting positive emotions on a regular basis,
- Keeping relevant parties informed of the subject's state,
- Having the system manage itself and be easily modified by the subject if wanted, and
- Implementing policies that relate to the agent.

The research involved in SAAD uses proven technology which in turn avoids the pitfalls and costs of other methods (such as robotics), and provides the foundation to replicate systems with reasonable cost and effort. Among the technology used for this project is smart-house technology. Research goals of the proposed project include:

- Providing common ground and context, retain an accessible and repairable record of the interaction between the subject, the agent, and other parties that covers the entire period in which the system has been in use,
- Implementing SAAD so that it is autonomic,
- Displaying information provided by the fusion of domotic and biophysical data and deemed relevant by the agent in a readily understood form on the monitor along with the avatar,
- Allowing the agent to interact via the avatar with the subject to assist in ADL's (Activities of Daily Living), and
- Providing web access that keeps significant third parties aware of the subject's status and allows them to interact with the subject audibly and via text-based devices.

The SAAD system, once materialized, will be installed throughout the home of an elderly person as shown in Figure 4. The layout of the SAAD system includes a media room that holds the central computer. This central computer will handle most of the data processing. The family, kitchen, and bedrooms will each have a monitor mounted on a wall along with speakers and a local processor for audio and video processing. The local processor will also be responsible for collecting and processing sensor data collected from inconspicuous sensors such as wristbands or small clip-on devices. The avatar will appear on the appropriate room monitor and speak to the subject. As the subject moves from room to room, the avatar will follow him or her by appearing on the corresponding monitor in that room. According to proposals submitted by North Carolina A & T State University faculty, the SAAD project is composed of several independent subprojects intended for either one or a small group of students to work on.

These subprojects may be independent, but many of them can be integrated into pairs or even groups of three or more once they are completed. Some of the suggested prototype projects include:

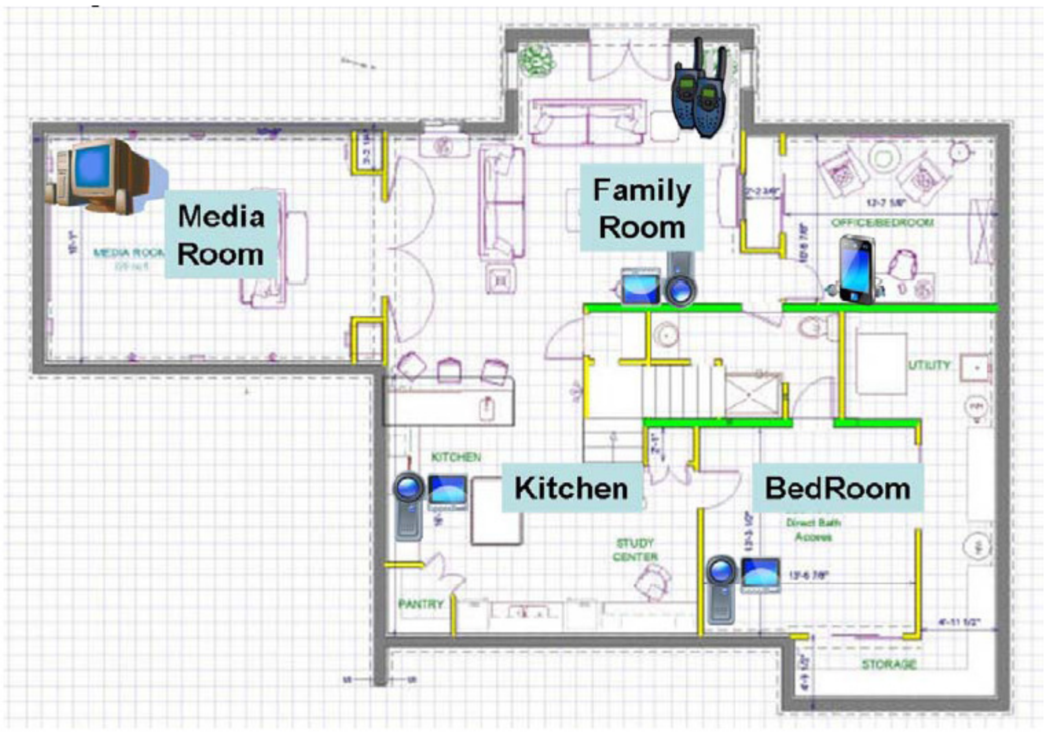


Figure 4. Physical Layout of SAAD System

1. **The agent** – This subproject actually has three sections. The first section involves implementing a program that executes a script via keyboard input and text output. This will use Jess for the scripts. The system will keep track of the common ground as it evolves so that it interacts in a believable way. The next step is to use speech synthesis (without prosody) for output. The last section is largely an integration of the first two.
2. **Speech recognition –** The system will recognize individual words with a microphone.
3. **Speech synthesis –** The system will be able to produce human speech with some degree of prosody (expressing emotion)
4. **Avatar –** The avatar is what the socially assistive agent will be embodied as. It will be displayed on one or several monitors throughout the house, particularly the monitor closest to the subject.
5. **Web access –** SAAD will have a web presence that can be accessed at predetermined times or when desired. The server can bring up a browser window as well. Web access also makes it possible for interested parties to track and communicate with the subject
6. **Wireless, Bluetooth, or Wi-Fi communication –** Some form of wireless communication is needed. Cost and necessity (possibly among others) will be determining factors in determining which protocol will be used. Wireless communication will be used for the purpose of this subproject to handle sensor data and actuator commands

7. **Locating the subject –** This will be done by utilizing a simple motion sensor or some other similar technology

4.2 WireAct

This thesis focuses on the subproject that deals with the handling of sensor data and actuator commands via wireless communication, specifically on configuring a field programmable gate array (FPGA) to control representations of actuators and sensors. This subproject is referred to as WireAct. There were numerous hardware and software programming methods used to address the problem at hand. The FPGA of choice was the Xilinx ML405 Evaluation Board, which also includes a PowerPC embedded processor. Xilinx Platform Studio (XPS) was used as the programming environment. Various custom intellectual properties (IP's) were created to control the different actuators and sensors. For example, a custom IP was created to control an actuator to turn on the lights and another IP will be created to turn on a television. VHDL was used to create and configure the custom IP's and C language was used to program the embedded processor of the ML405 to interact between IP's. Version 2.6 of the Linux kernel was also utilized for this research. In order to use the Linux kernel, a virtual computer was created inside a host computer to run Ubuntu, which is a Linux-based operation system. Programs were written in C language and compiled into static files, then added to the Linux kernel on the virtual machine. The kernel was then built and downloaded onto the ML405 board. The custom hardware IP's were programmed to be controlled through the serial port of the ML405 using a HyperTerminal connection with a baud rate of 9600 bits per second. For example, if it was desired to turn a light on, a user would type “lights on” through the HyperTerminal connection. To turn the lights off, the user would simply type “lights off”. WireAct was broken up into several milestones which were accomplished and are discussed in the experimental setup and experimental results chapters.

CHAPTER 5

HISTORY AND OVERVIEW OF LINUX

5.1 History

Linux is an open-source version and redistributable clone of the UNIX operating system which was released in 1991 by Finnish computer science student Linus Torvalds. While studying at the university, Torvalds used MINIX, which is another UNIX-like system, to write his own kernel. He started by writing device drivers and hard-drive access and had a basic design by September of 1991 which he called Version 0.01. This kernel was combined with the GNU system to form a free operating system which would be known as Linux [26, 27, 28].

Linux was initially a terminal emulator, which Torvalds used to access the UNIX servers at the University of Helsinki in Finland. However, with time Linux has grown into a well-respected system that is widely used in educational and corporate networks. By 2000, most computer companies supported Linux in some manner. It is used for Web servers, file servers, and can even be used on TV receivers and recorders such as TiVo, cell phones and game systems including the Playstation 3. Linux is also considered free software, which means that anyone can download the source from the Internet or buy it on an inexpensive CD-ROM [26, 27, 29].

5.2 Overview

A kernel can be classified as being one of four types:

- **Monolithic** – This is the type traditionally used by most UNIX-based operating systems. A monolithic kernel contains all of the system core functions and device drivers such as disk drives and printers.
- **Microkernel** – A microkernel only offers minimum services such as defining memory address spaces and process management. All other functions are implemented independent of the kernel.
- **Hybrid kernel** – A hybrid kernel is similar to a microkernel. However, it includes additional code in the kernel space so that such code can execute more swiftly.
- **Exo kernel** – This type of kernel is still in the design and research stage. The user will be able to create processes that can access kernel resources directly in an Exo kernel.

As most UNIX-based systems are, the Linux kernel is monolithic. It is written in a GNU compiler collection (GCC) supported version of C programming language along with short sections of code written in assembly language. Linux is not only portable, but also very versatile. It is one of the most commonly used system kernels, capable of running on a number of systems including the IPAQ and Apple's iPod and iPhone. Linux can run on many virtual machine architectures as a host operating system or as a guest operating system. There are a number of different versions of the Linux operating system as well. Among the different versions are Ubuntu, Red Hat, Fedora, Oracle, and Xandros [30, 31].

Figure 5 shows the basic directory structure for the Linux architecture. The starting point of the Linux directory structure is the root directory, denoted by “/”. Every other directory in the system is under this directory and is considered to be a subdirectory. The root directory usually only contains subdirectories and it’s not sensible to store single files directly under it [32].

The /bin directory is a common subdirectory of root and holds the essential user executable programs that are needed to have minimal functionality for booting and repairing the system. Some of the executables that are stored in the /bin directory include:

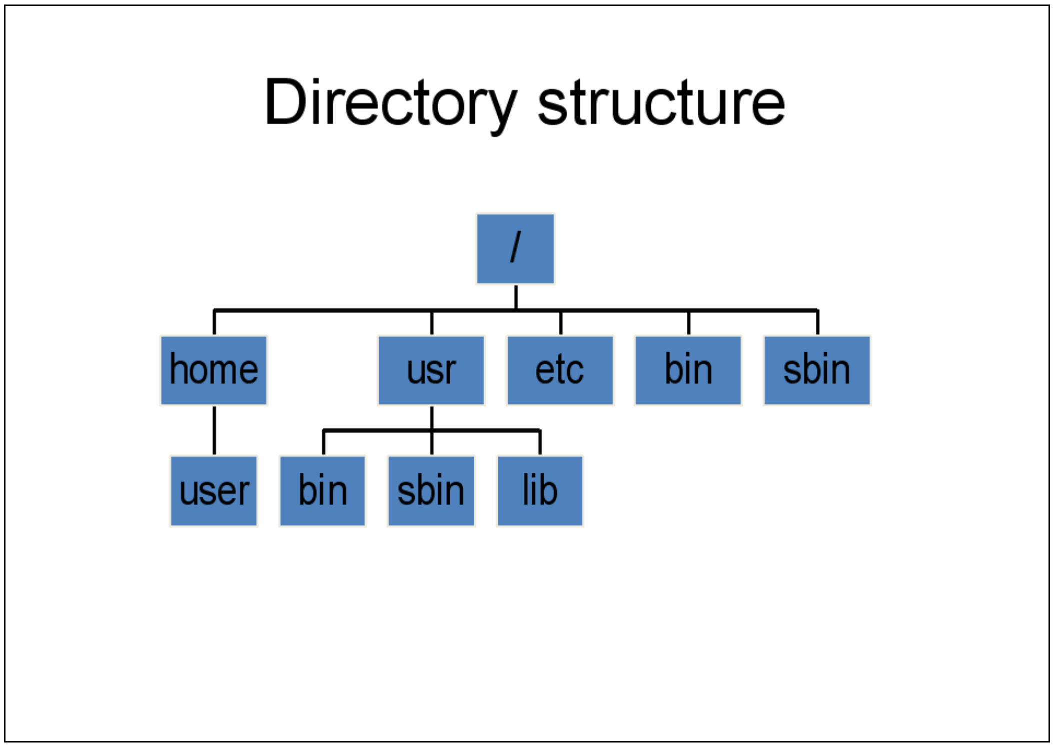


Figure 5. Basic Linux Directory Structure

- **cd** – Changes the current directory that a user is in,
- **ls** – Lists all of the programs and files in a directory,
- **pwd** – Acronym for print working directory. Displays the name of the current working directory, and
- **kill** – Terminates a process.

The /bin directory also contains the shells which execute commands that are issued from a standard input device (keyboard) or from a file [33].

The /sbin directory is similar to the /bin directory in the fact that it holds executables that are needed to boot the system. However, the /sbin programs are system binaries and are usually only carried out by the root user. As a result, /sbin is not initially included in the "PATH" environment variable for regular users. There are more than 250 programs that are typically found in the /sbin directory. Among the most commonly used are:

- **fastboot** – Restarts the system without rechecking disks,
- **fsck** – A filesystem check and repair utility,
- **halt** – Stops the system,
- **reboo**t – Restarts the system, and
- **update** – Updates an application [34].

The /lib directory contains the libraries needed for the executables in the /bin and /sbin directories. Most modern machines share libraries, so very little would work without this directory. Among the libraries that are found in /lib should be the libraries that are needed to start the system, according to the Filesystem Hierarchy Standard (FHS) [35].

The /usr directory usually holds the most data on a system. This directory is home to three subdirectories with the same names as the three subdirectories to root: /usr/bin, /usr/sbin and /usr/lib. These three subdirectories hold the nonessential user commands, system commands and libraries which are not needed in single user mode [35, 36, 37].

Any other directories that a user creates typically are stored under the /home directory. Every user on the system has his or her own directory under /home. This directory also holds anything from music and videos, to configuration files or preferred settings for software a user uses. A user's home directory is by default protected by file system permissions and only that user or an administrator can access it [37]. If a user wants to specify who has access to the home directory (or any other directory), the "chmod" command can be used to change the permissions of the home directory. The may be other directories under "/" such as

- **/mnt –** used to mount filesystems or devices,
- **/opt –** holds software and add-ons, and
- **/tmp –** holds temporary files [38].

CHAPTER 6

FIELD PROGRAMMABLE GATE ARRAYS (FPGAs)

6.1 History

The FPGA industry emerged from programmable read-only memory (PROM) along with programmable logic devices (PLDs). The first FPGA, the XC2064, was developed by Xilinx in 1985. The chip consisted of only 64 configurable logic blocks (CLBs) with two 3-input lookup tables. At that time, chips like FPGAs had realizable limitations, so the US Naval Surface Warfare Department funded a project to develop a computer that had the capacity to contain 600,000 reprogrammable gates. This technology was patented in 1992. After this, FPGAs started to improve drastically, both in sophistication and level of production. During the early 1990s, FPGAs were mostly utilized in telecommunications and networking. Within a few years, they had expanded to other industries such as consumer and automotive applications. Today, some of the applications that utilize FPGAs are digital signal processing, aerospace, and defense systems. They are also used more in high performance computing such as Fast Fourier Transforms and Convolution. As versatile as FPGAs are, they still possess some limitations due to their potentially complex design [39, 40].

6.2 Overview

A PLD is a chip that is manufactured at a factory and then customized by a programmer to create different logic circuits. There are a number of different types of PLDs that have been introduced over the years. Among them are programmable logic arrays (PLAs), programmable AND-array logic (PAL), simple programmable logic devices (SPLD) and complex programmable logic devices (CPLDs). Some designers classify the FPGA as a type of PLD as well, since it is also a device that is ultimately programmed by an end-user [41].

The FPGA is the most complex of the PLD group. They now have the potential of containing millions of transistors and can compete with very-large-scale integration (VLSI) chips and application-specific integrated circuits (ASICs) in size, circuit density, and switching speed. Predesigned cells and routing wires mainly compose FPGAs. They are customized by creating or destroying connections within the cells or between cells and wires [41].

Figure 6 displays the basic structure of an FPGA. There are three basic components to every FPGA:

- Configurable logic blocks (CLBS)
- Input/Output blocks (IOBs)
- Interconnect matrix

The large blocks on the inside of the FPGA structure are the CLBs. CLBs are the basic logic units of an FPGA. More advanced FPGAs such as Xilinx's Virtex 5 can contain an array of as many as 17,000 to 18,000 CLBs. Every CLB has a configurable switch matrix with 4 or 6 inputs, a selector circuit (e.g. a multiplexer), flip flops for synchronous storage elements, and even full adders. The design of a CLB would be similar to the one shown in Figure 7. A CLB has the capability of implementing a simple logic function [42, 43].

The smaller blocks on the outside of the FPGA structure are input/output blocks. The internal structure of an IOB can be seen in Figure 8.

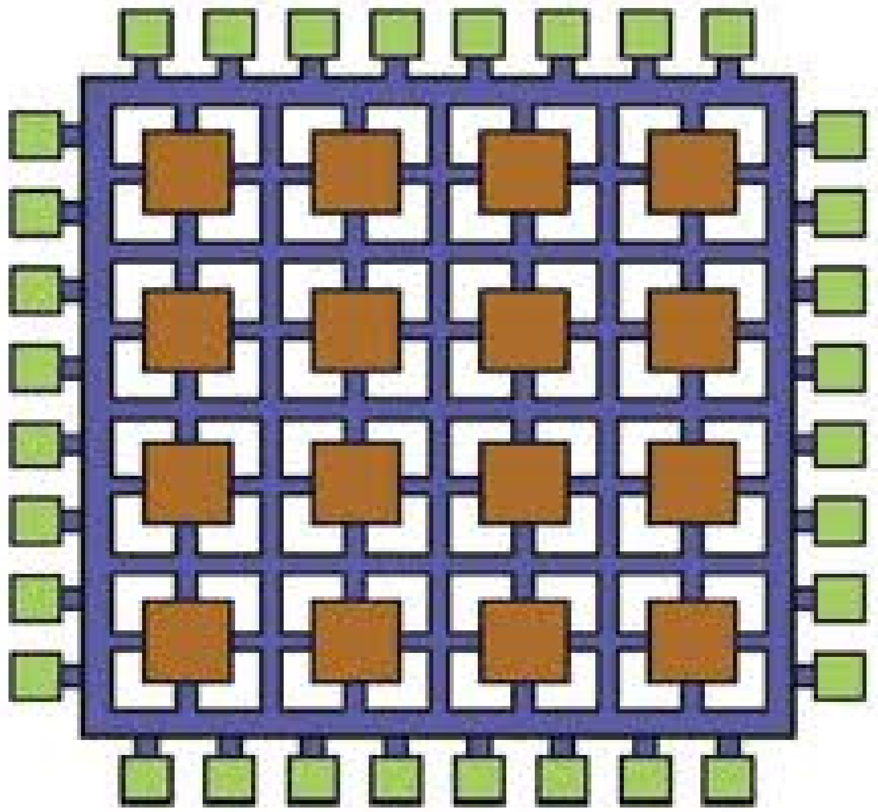


Figure 6. Basic FPGA structure

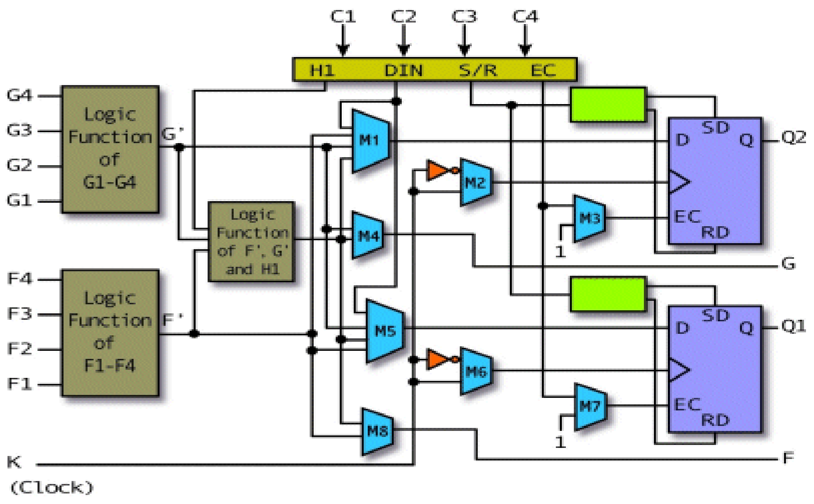


Figure 7. FPGA Configurable Logic Block (CLB) (courtesy of Xilinx)

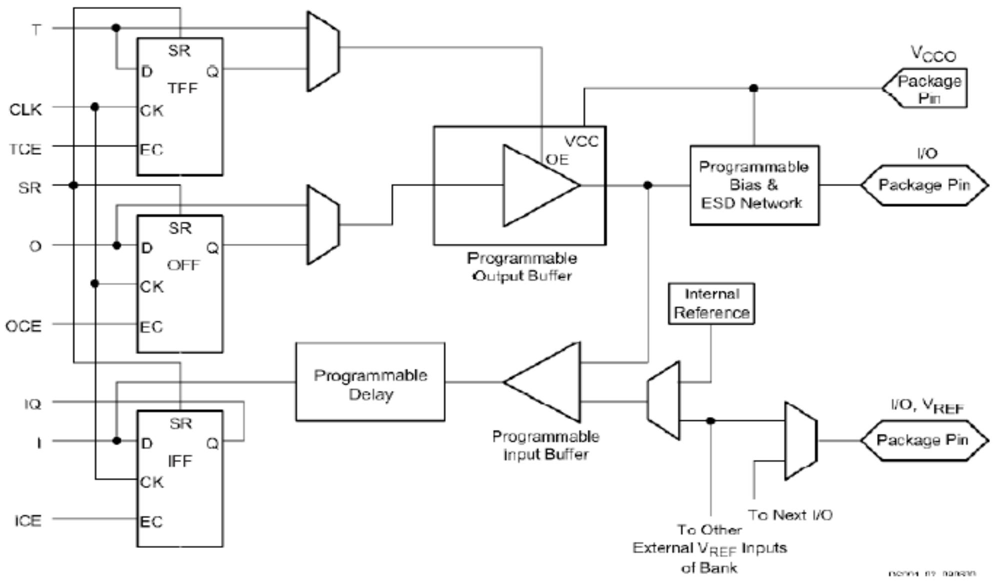


Figure 8. Input/Output Block (IOB) structure

Sometimes called input/output pads, these are responsible for providing the interface between the internal logic and the FPGA's package pins. Each pad also possesses optional pull-up and pull-down resistors and a weak-keeper circuit. Before an FPGA is configured, all outputs that aren't involved in the configuration are forced to high-impedance and the weak-keeper circuits are inactive, but inputs have the option of being pulled up. In the input path of an IOB, a buffer can route the input signal directly to internal logic or to an input flip-flop, which eliminates pad-to-pad hold time. In the output path, there is a 3-state output buffer that sends the output signal to the pad. Just as with an input signal, an output signal can be sent directly to the buffer or to an output flip-flop [44].

CLBs are connected with other CLBs and IOBs by the interconnect matrix (also called the routing matrix). An FPGA contains different types of connection lines:

- Long lines – used to connect CLBs that are far apart
- Short lines – used to connect neighboring CLBs
- Dedicated clock trees – used to synchronize CLBs
- Dedicated set/reset lines – used to set or reset all flip-flops in the FPGA

Using the interconnect matrix, numerous CLBs can be combined to form more advanced circuitry. Every FPGA has a number of CLBs, IOBs, and an interconnect matrix. More advanced FPGAs can include additional components such embedded microprocessors and dedicated multipliers [45].

6.3 The FPGA of Choice

The FPGA chosen for the purpose of subproject WireAct is the Xilinx ML405 Evaluation Platform. This board is one of the Virtex 4 FPGAs. The ML405 consists of its specific Virtex 4 FPGA package (XC4VFX20) along with a number of external hardware peripherals that can be configured to interface with the FPGA chip. Among these peripherals are pushbuttons, LED lights, an LCD screen, a RS232 serial port, and a series of expansion headers. More can be read about each of the ML405's peripherals in the "Detailed Description" section of the "ML405 Evaluation Platform User Guide" which is available on www.xilinx.com.

The Virtex 4 on the ML405 board also has a PowerPC 405 processor block. The PowerPC is responsible for handling any potential communication between peripherals using data buses and is programmed using C language. Features of the PowerPC 405 include:

- Up to 450 MHz operation,
- 16 KB instruction cache,
- 16 KB data cache, and
- On-chip memory.

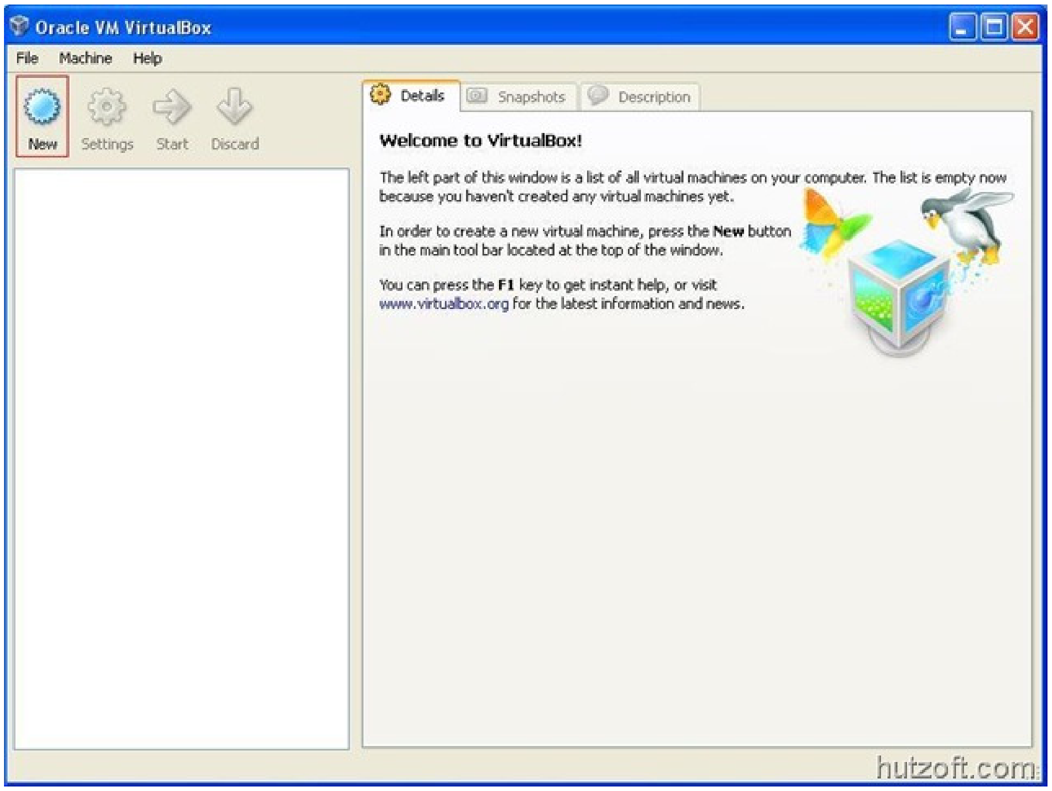
Aside from all of the ML405's capabilities, this board was chosen because http://xilinx.wikidot.com has provided a reference design bit stream for the ML405 (as well as the ML507) that can simply be downloaded and executed on the board. This reference design provided a good starting point for the research pertaining to WireAct.

CHAPTER 7

EXPERIMENTAL SETUP

7.1 The Virtual Computer

One of the goals for this subproject was to customize and build the Linux kernel and run it on the ML405. This required the creation of an environment that can run the Linux operating system. A virtual computer was the simplest and easiest approach. "VirtualBox" is an x86 and AMD64/Intel64 virtualization product than can be used in commercial as well as home applications. It can be currently run on Windows, Linux, Macintosh and OpenSolaris hosts and can run a vast range of guest operating systems. With the help of VirtualBox, multiple operating systems (inside multiple virtual machines) can be run at the same time. The only limitation that is associated with how many virtual computers can be simultaneously run is how much space and memory is available on the disk. VirtualBox is currently available in many different packages and the host operating system determines installation. The VirtualBox software was easily found and downloaded from the Internet. Figure 9 displays the startup window once the software was downloaded. This window is known as the "VirtualBox Manager". The left features a pane that lists all of the virtual machines that the user has created. This list was empty after installation since no virtual machines were created yet. The pane on the right shows the currently selected virtual machine's properties, if any. Once again, no properties were displayed due to the fact that no virtual machines were created at that point [46,47].



Welcome to VirtualBox!

The left part of this window is a list of all virtual machines on your computer. The list is empty now because you haven't created any virtual machines yet.

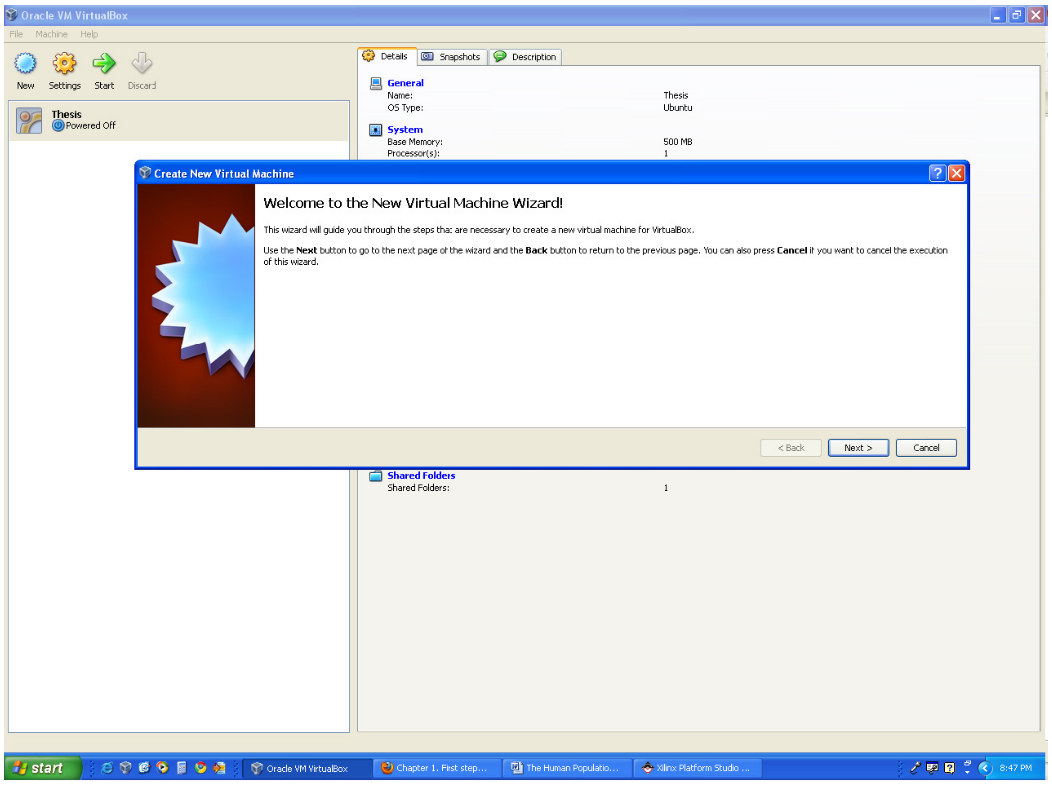
In order to create a new virtual machine, press the **New** button in the main tool bar located at the top of the window.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for the latest information and news.

Figure 9. VirtualBox Startup Window

The virtual machine runs off an ISO (International Organization for Standardization) file, also known as an ISO image. It is an archive file of an optical disc which includes the contents of every written section of the disc, including the file system. Every version of Linux operating systems has a corresponding ISO file. Ubuntu 10.10 was the chosen version of Linux. The respective ISO file was found and downloaded from http://www.ubuntu.com/desktop/get-ubuntu/download [48].

Now that a suitable version of Linux was found, the next step was to actually create a virtual machine. From the VirtualBox manager, the "New" button at the top of the window started a wizard to guide the setup of the new virtual machine [47]. An alternate method of starting the wizard is under the "Machine" menu in the toolbar. Figure 10 shows the wizard to create a new virtual machine.



General
Name: Thesis
OS Type: Ubuntu

System
Base Memory: 500 MB
Processor(s): 1

Welcome to the New Virtual Machine Wizard!

This wizard will guide you through the steps tha: are necessary to create a new virtual machine for VirtualBox.

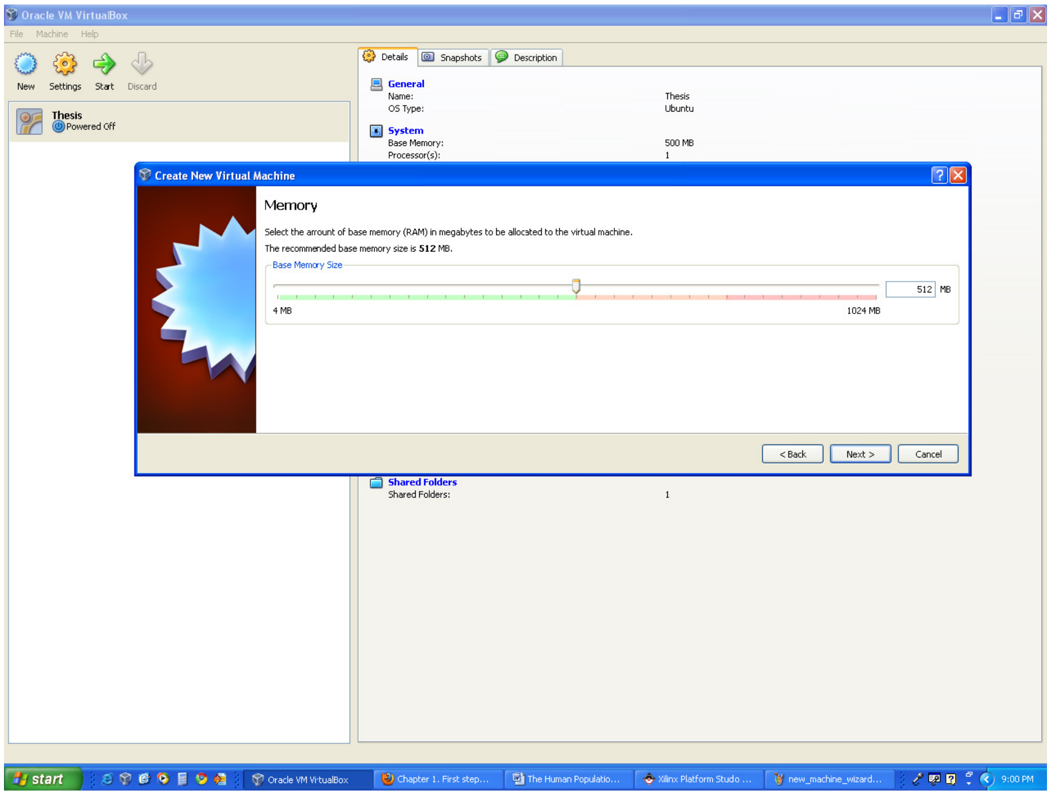
Use the **Next** button to go to the next page of the wizard and the **Back** button to return to the previous page. You can also press **Cancel** if you want to cancel the execution of this wizard.

Shared Folders
Shared Folders: 1

Figure 10. "Create New Virtual Machine" Wizard: Welcome Page

The next page of the wizard asked for the name, operating system, and operating system version of the new VM. There were a number of operating systems that could be chosen including Microsoft Windows, Linux, Solaris, BSD, IBM OS/2, and Mac OS X. The new virtual machine was named “Thesis”. The preferred operating system was Linux. Among the different versions of Linux were Linux 2.2, Linux 2.4, Linux 2.6, Debian, Fedora, Red Hat and Ubuntu, among others. Since the Ubuntu 10.10 ISO file was already downloaded, the version of choice was Ubuntu.

Following the basic information page was the page to specify how much memory the VM will allocate when it is running. Figure 11 displays the memory page.



Thesis
Powered Off

General
Name: Thesis
OS Type: Ubuntu

System
Base Memory: 500 MB
Processor(s): 1

Memory

Select the amount of base memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended base memory size is **512** MB.

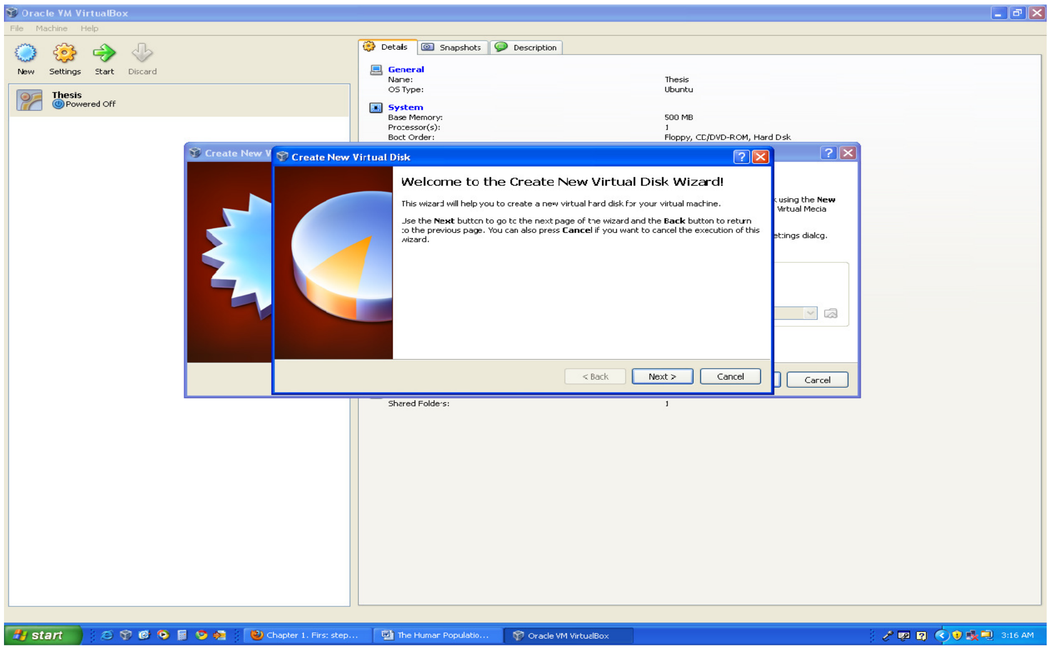
Base Memory Size
512 MB
4 MB
1024 MB

Shared Folders
Shared Folders: 1

Figure 11. “Create New Virtual Machine” Wizard: Memory Page

This was a setting that needed to be chosen carefully. The memory that the VM used while it was running was not available to the host operating system [47]. The wizard stated that the recommended base memory size was 512 MB. The host computer had a total of 1 GB of RAM, so 512 MB was the most memory that could be reserved for the VM without compromising the operability of the host operating system. If a host computer had more RAM at its disposal, more memory could have been specified for the VM.

After specifying how much memory was reserved for the new VM, the virtual hard disk needed to be specified. The new VM could either use a newly created virtual hard disk or use an existing hard disk image file. A new hard disk was created for this new VM. Electing to create a new disk image file launched another wizard which aided in creating a new virtual disk. The welcome page of the new wizard can be seen in Figure 12. After the welcome page, the wizard asked which type the new hard disk would be: dynamically expanding or fixed-sized. If the disk was dynamically expanding storage, it would start out small in size and grow as the guest operating system stores more data onto it. A fixed-sized hard disk would initially occupy its specified size. Despite the fact that a fixed-sized disk initially occupies more space, it is still faster than a dynamically expanding disk because it requires less overhead [47]. For this reason, the new VM was given a fixed-sized disk. The disk's location and size was determined on the next page of the wizard which is shown in Figure 13.



General
Name: Thesis
OS Type: Ubuntu

System
Base Memory: 500 MB
Processor(s): 1
Boot Order: Floppy, CD/DVD-ROM, Hard Disk

Shared Folders: 1

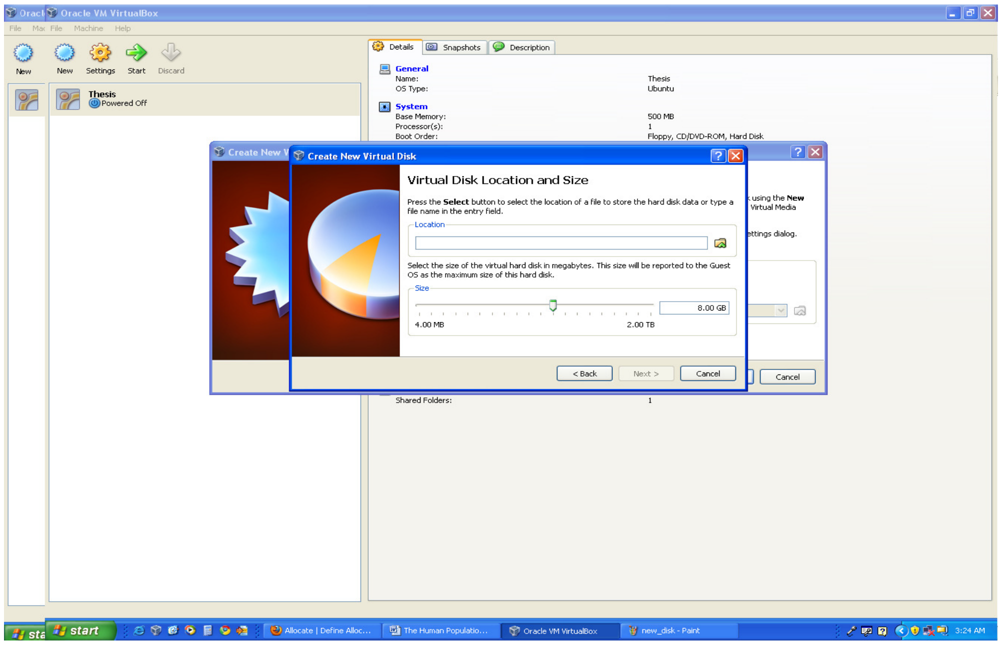
Thesis
Powered Off

Welcome to the Create New Virtual Disk Wizard!

This wizard will help you to create a new virtual hard disk for your virtual machine.

Use the **Next** button to go to the next page of the wizard and the **Back** button to return to the previous page. You can also press **Cancel** if you want to cancel the execution of this wizard.

Figure 12. "Create New Virtual Disk" Wizard: Welcome Page



General
Name: Thesis
OS Type: Ubuntu

System
Base Memory: 500 MB
Processor(s): 1
Boot Order: Floppy, CD/DVD-ROM, Hard Disk

Shared Folders: 1

Thesis
Powered Off

Virtual Disk Location and Size

Press the **Select** button to select the location of a file to store the hard disk data or type a file name in the entry field.

Location

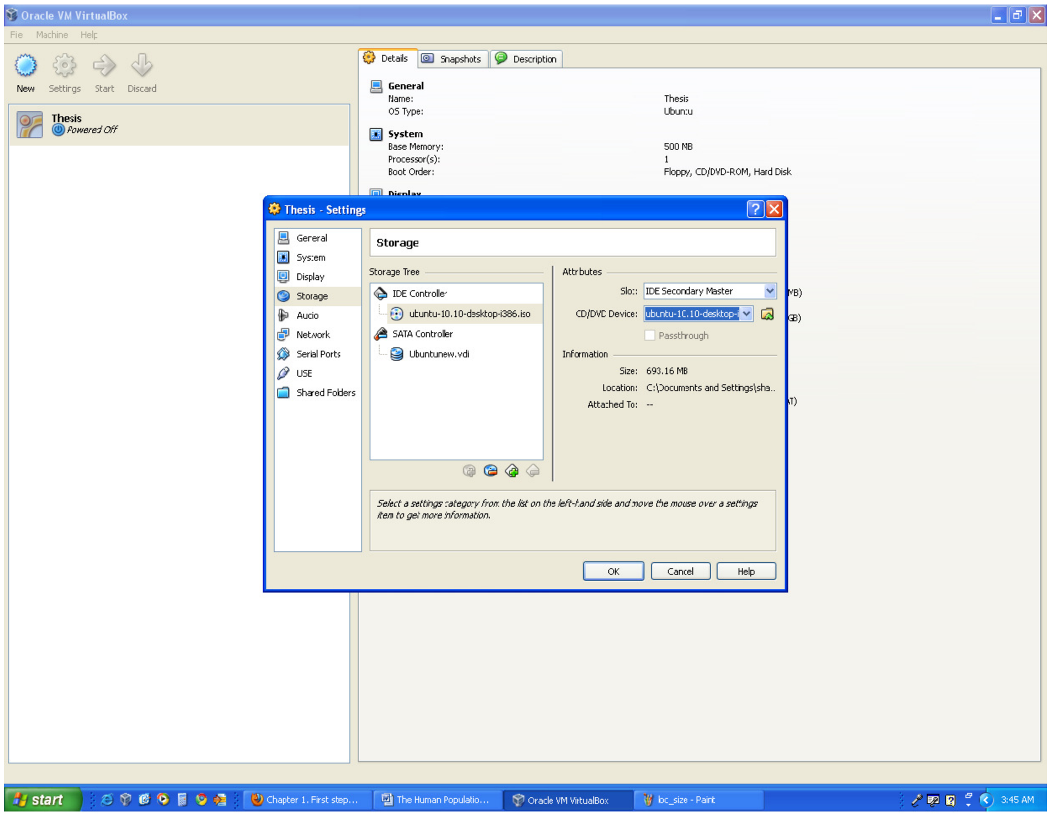
Select the size of the virtual hard disk in megabytes. This size will be reported to the Guest OS as the maximum size of this hard disk.

Size: 8.00 GB (4.00 MB – 2.00 TB)

Figure 13. "Create New Virtual Disk" Wizard: Virtual Disk Location and Size Page

The hard disk was also given the name "Ubuntunew" at this step and was stored in the "HardDisks" folder under the ".VirtualBox" directory. Roughly 11 GB was reserved for this hard disk to compensate for any additional files that the VM may need to store. Before the hard disk was created a final summary page displayed the user-defined specifications for the new hard disk. Clicking finish created the new hard disk [47].

With the virtual disk created, the VM was ready to run and have Ubuntu 10.10 installed onto it. To start the VM with the running Ubuntu 10.10 operating system, the Ubuntu ISO file was added to the IDE controller as shown in Figure 14.



General
Name: Thesis
OS Type: Ubuntu

System
Base Memory: 500 MB
Processor(s): 1
Boot Order: Floppy, CD/DVD-ROM, Hard Disk

Storage

Storage Tree
IDE Controller
ubuntu-10.10-desktop-i386.iso
SATA Controller
Ubuntunew.vdi

Attributes
Slot: IDE Secondary Master
CD/DVD Device: ubuntu-10.10-desktop-i
Passthrough

Information
Size: 693.16 MB
Location: C:\Documents and Settings\sha..
Attached To: --

Select a settings category from the list on the left-hand side and move the mouse over a settings item to get more information.

Figure 14. VM Storage Settings Page

With the ISO file attached to the secondary master of the IDE controller, the VM booted as if there was an Ubuntu 10.10 CD in its imaginary CD drive. Once the VM was up and running, the Ubuntu 10.10 operating system was installed onto the VM. Restarting the VM finalized the installation. However before restarting the VM, the secondary master of the IDE controller was set back to "Empty". With Ubuntu actually installed onto the VM, there was no need to leave the ISO file attached to the IDE controller [47].

7.2 The Default Linux Kernel

With the VM up and running, the next task was to build and download the default Linux kernel onto the ML405 board. Several preparations were made before downloading the kernel from the website. The PowerPC GNU tools were acquired by following a series of steps found under the "Tools" tab of the open source Xilinx website http://xilinx.wikidot.com. First, an "Embedded Linux Development Kit (ELDK)" ISO file needed to be downloaded to the VM. This was a fairly large file (roughly 1.9 GB) and took approximate three and a half hours to download to the VM. The necessary directories such as /media/cdrom, /opt/ELDK and /opt/ELDK/4.2 were created in the terminal application of the VM. However, no further steps could be taken until this ISO file completed downloading [49].

The open source website (http://xilinx.wikidot.com) described how to mount the ISO file, in this case to the VM. The following command mounted the ISO image to the /media/cdrom directory

- **mount –o loop –t iso9660 ppc-2008-04-01.iso /media/cdrom**.

All directions on xilinx.wikidot.com assume that the user has root privileges. To ensure that the mount command is not denied due to permission issues, the word "sudo" was used at the beginning of such commands. The word sudo allowed the user to issue a command as a superuser or another user. Executing a sudo command required that the user verify his or her identity by entering the password of the current user [50]. As a result, the mount command was issued as

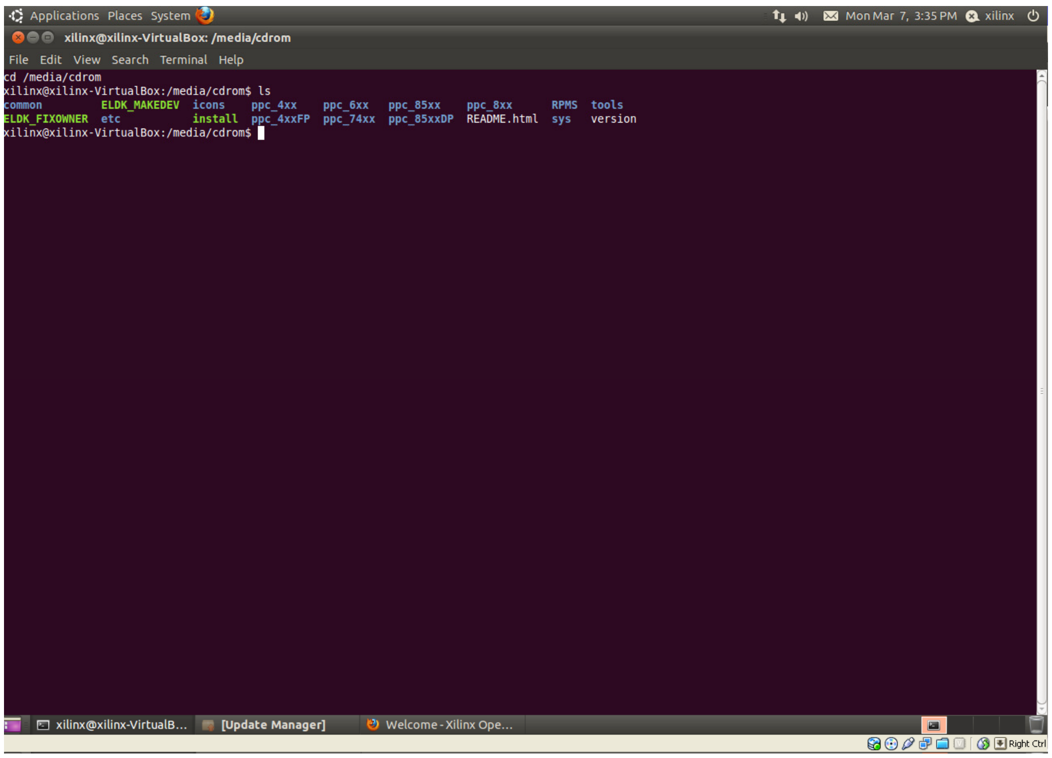
- **sudo mount –o loop –t iso9660 ppc-2008-04-01.iso /media/cdrom.**

The /media/cdrom directory now contained the contents of the ppc ISO file and could be displayed using the "ls" command. The contents of the ISO image are shown in Figure 15. Among the contents of the ISO file was the "ppc_4xx" directory, which contained a large number of RPM files.

The following commands installed the ppc_4xx directory to the newly created /opt/ELDK/4.2 directory:

1. **./install -d /opt/ELDK/4.2 ppc_4xx**
2. **cd /opt/ELDK/4.2** – Changes the current directory to /opt/ELDK/4.2
3. **source eldk_init 4xx** – Initializes the CROSS_COMPILE and setup paths to uses the ELDK tools
4. **/media/cdrom/ELDK_FIXOWNER -a ppc_4xx** – Changes owners of the files of the ELDK installation to root
5. **/media/cdrom/ELDK_MAKEDEV -a ppc_4xx** – Creates the device nodes.

The website stated that before anything is compiled for the PowerPC, the "eldk_init" script, located in the /opt/ELDK/4.2 directory, needed be sourced [49].


Applications Places System
Mon Mar 7, 3:35 PM xilinx
xilinx@xilinx-VirtualBox: /media/cdrom
File Edit View Search Terminal Help
xilinx@xilinx-VirtualB...
[Update Manager]
Welcome - Xilinx Ope...
Right Ctrl

```
cd /media/cdrom
xilinx@xilinx-VirtualBox:/media/cdrom$ ls
common       ELDK_MAKEDEV  icons    ppc_4xx    ppc_6xx   ppc_85xx    ppc_8xx      RPMS  tools
ELDK_FIXOWNER  etc           install  ppc_4xxFP  ppc_74xx  ppc_85xxDP  README.html  sys   version
xilinx@xilinx-VirtualBox:/media/cdrom$
```

Figure 15. Contents of /media/cdrom Directory in Linux Terminal

With the installation complete, the ELDK tools could now be used. The next procedures were found on the "PowerPC Linux" page under the "Open Source Linux (OSL)" tab. This page provided an ML405 reference system and a prebuilt ramdisk image to download to the host computer. Before the PowerPC Linux kernel could be obtained from the Xilinx Git server, Git needed to be downloaded to the VM. The following command obtained Git from the Git server:

- **sudo apt-get install git**

After Git was installed on the VM, the command

- **git clone git://git.xilinx.com/linux-2.6-xlnx.git** created a “linux-2.6-xlnx” directory in the current working directory, which was a new directory called “xilinx”. The contents of the “linux-2.6-xlnx” directory are displayed in Figure 16. The next task was to configure the newly obtained Linux kernel. A default configuration, provided by Xilinx, was used by issuing the following command in Terminal:

- **make ARCH=powerpc 40x/virtex4_defconfig.**

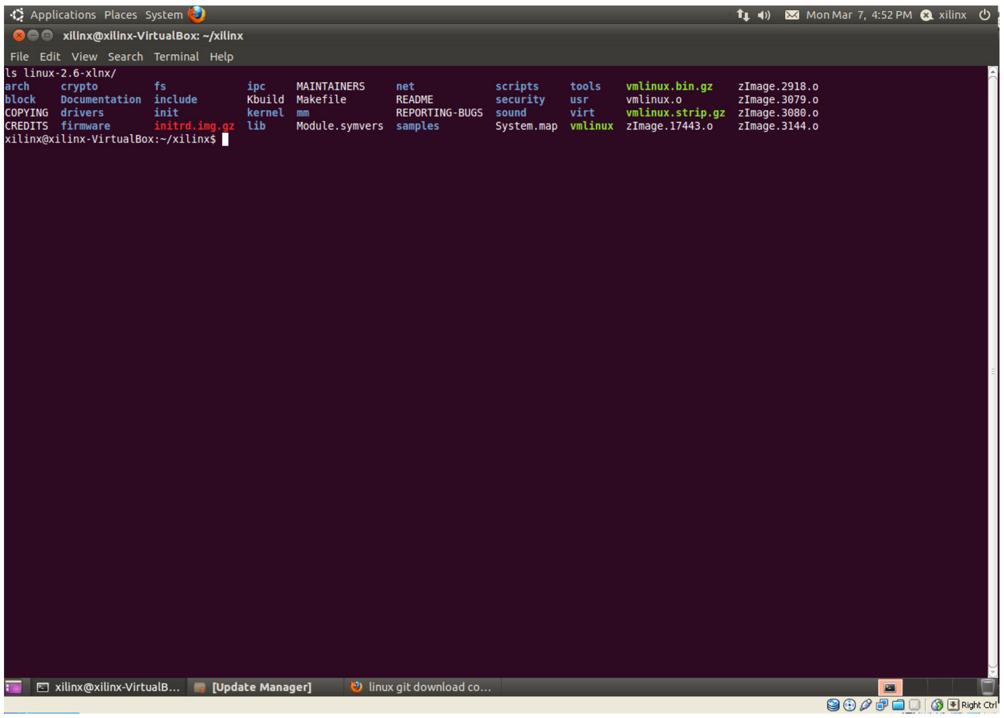
Any Linux commands involving “make ARCH=” had to be issued in the Linux kernel’s top directory. Once the kernel was configured to its default specifications, the command

- **make ARCH=powerpc simpleImage.virtex405-ml405**

built the kernel and created an elf file named “simpleImage.virtex405-ml405.elf” in the arch/powerpc/boot directory of the Linux tree. This command built the kernel without a ramdisk. The following command built the kernel with the ramdisk present:

- **make ARCH=powerpc simpleImage.initrd.virtex405-ml405**

This command will be referred to as simply “create.ram.kernel-<custom name>” with custom name is this case being “ml405”. The downloaded prebuilt ramdisk had to be saved in the arch/powerpc/boot directory of the kernel tree or the command to build the kernel with the ramdisk failed [49]. The ramdisk is important because it is what is used to boot the kernel once it is downloaded onto the board. It contains a "linuxrc" executable and the necessary directories to have the basic functionality of a Linux OS.


Applications Places System
Mon Mar 7, 4:52 PM xilinx
xilinx@xilinx-VirtualBox: ~/xilinx
File Edit View Search Terminal Help
xilinx@xilinx-VirtualB...
[Update Manager]
linux git download co...
Right Ctrl

```
ls linux-2.6-xlnx/
arch     crypto         fs            ipc     MAINTAINERS     net            scripts     tools    vmlinux.bin.gz    zImage.2918.o
block    Documentation  include       Kbuild  Makefile        README         security    usr      vmlinux.o         zImage.3079.o
COPYING  drivers        init          kernel  mm              REPORTING-BUGS sound       virt     vmlinux.strip.gz  zImage.3080.o
CREDITS  firmware       initrd.img.gz lib     Module.symvers  samples        System.map  vmlinux  zImage.17443.o    zImage.3144.o
xilinx@xilinx-VirtualBox:~/xilinx$ 
```

Figure 16. Contents of Linux Tree in Linux Terminal

CHAPTER 8

EXPERIMENTAL RESULTS

The built Linux kernel was ready to be loaded and run on the board. At this point, the rest of the tasks were completed using the host operating system. The newly created elf files needed to be copied to the folder of a new XPS project or an existing XPS project on the host computer. A new XPS project called "simpleimage" was created for the purpose of testing the default Linux kernel. It wasn't necessary to configure the PowerPC or any IPs correctly at this point. This project was only needed to launch XMD because the reference bit stream was being downloaded onto the board using IMPACT. XMD was used to download and run the elf file on the ML405 using the following commands [49]

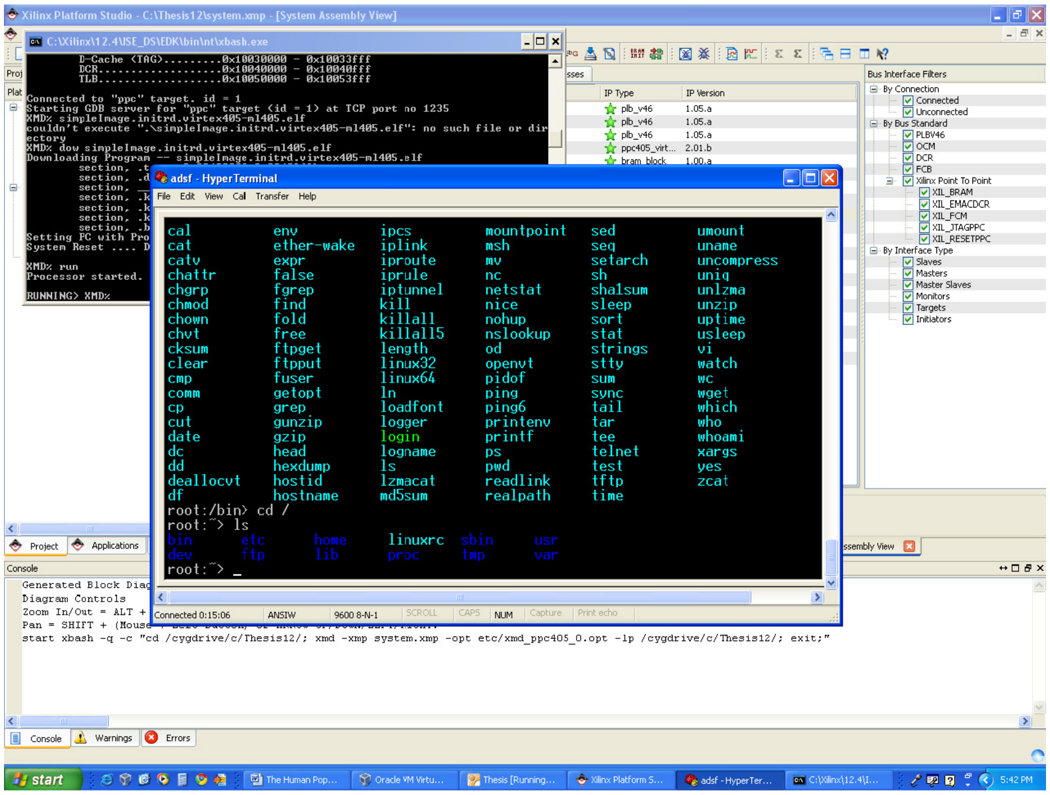
1. connect ppc hw -debugdevice deviceNr 3 cpunr 1
2. dow simpleImage.initrd.virtex405-ml405.elf
3. run

The default Linux kernel was successfully run and viewed though a HyperTerminal connection at 9600 bps. Figure 17 shows the running default kernel.

8.1 Creating a New Project to Run the Linux Kernel

Since the Linux kernel was successfully downloaded and tested on the ML405 using the reference bitstream, it was time to test the kernel using a new bitstream from a new project. This was done by creating new project in XPS with as many IP specifications matched to the referenced design as possible. After many attempts, the project named “Thesis 12” was created with the following specifications:

- PLB system
- Single Processor System


Xilinx Platform Studio - C:\Thesis12\system.xmp - [System Assembly View]
C:\Xilinx\12.4\ISE_DS\EDK\bin\nt\xbash.exe
adsf - HyperTerminal
File Edit View Cal Transfer Help
Connected 0:15:06 ANSIW 9600 8-N-1 SCROLL CAPS NUM Capture Print echo
Project
Applications
Console
Warnings
Errors

```
        D-Cache (TAG).........0x10030000 - 0x10033fff
        DCR...................0x10040000 - 0x10040fff
        TLB...................0x10050000 - 0x10053fff

Connected to "ppc" target. id = 1
Starting GDB server for "ppc" target (id = 1) at TCP port no 1235
XMD% simpleImage.initrd.virtex405-ml405.elf
couldn't execute ".\simpleImage.initrd.virtex405-ml405.elf": no such file or dir
ectory
XMD% dow simpleImage.initrd.virtex405-ml405.elf
Downloading Program -- simpleImage.initrd.virtex405-ml405.elf
        section, .t
        section, .d
        section, _
        section, .k
        section, .k
        section, .k
        section, .b
Setting PC with Pro
System Reset .... D

XMD% run
Processor started.

RUNNING> XMD%
```

```
cal         env         ipcs        mountpoint  sed         umount
cat         ether-wake  iplink      msh         seq         uname
catv        expr        iproute     mv          setarch     uncompress
chattr      false       iprule      nc          sh          uniq
chgrp       fgrep       iptunnel    netstat     sha1sum     unlzma
chmod       find        kill        nice        sleep       unzip
chown       fold        killall     nohup       sort        uptime
chvt        free        killall5    nslookup    stat        usleep
cksum       ftpget      length      od          strings     vi
clear       ftpput      linux32     openvt      stty        watch
cmp         fuser       linux64     pidof       sum         wc
comm        getopt      ln          ping        sync        wget
cp          grep        loadfont    ping6       tail        which
cut         gunzip      logger      printenv    tar         who
date        gzip        login       printf      tee         whoami
dc          head        logname     ps          telnet      xargs
dd          hexdump     ls          pwd         test        yes
deallocvt   hostid      lzmacat     readlink    tftp        zcat
df          hostname    md5sum      realpath    time
root:/bin> cd /
root:~> ls
bin      etc      home     linuxrc  sbin     usr
dev      ftp      lib      proc     tmp      var
root:~> _
```

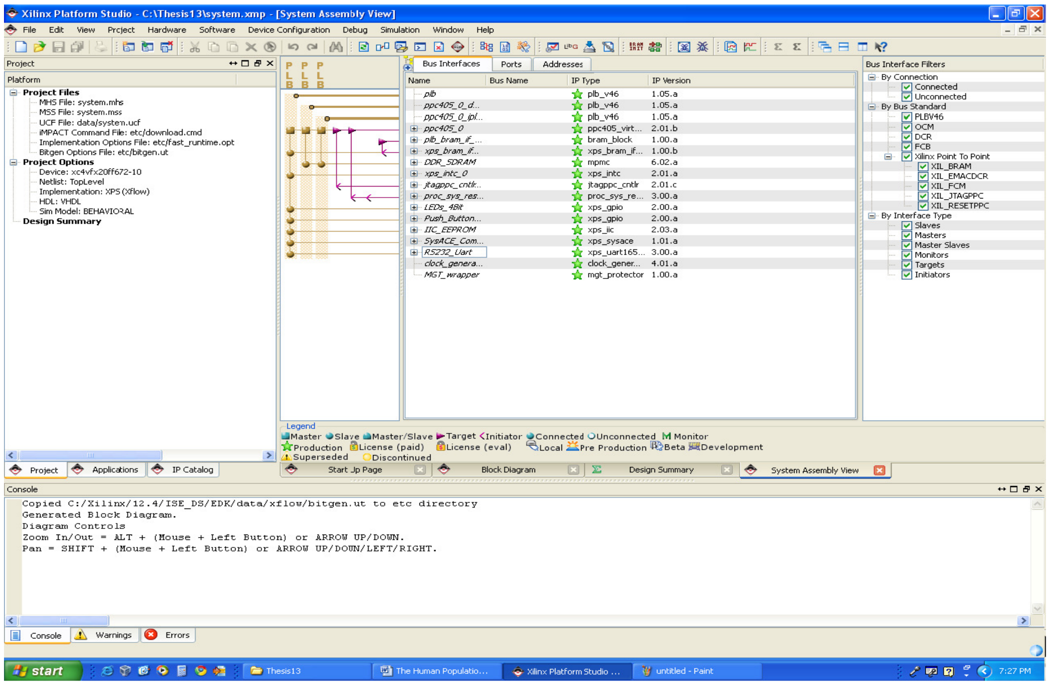
```
Generated Block Diag
Diagram Controls
Zoom In/Out = ALT +
Pan = SHIFT + (Mouse
start xbash -q -c "cd /cygdrive/c/Thesis12/; xmd -xmp system.xmp -opt etc/xmd_ppc405_0.opt -lp /cygdrive/c/Thesis12/; exit;"
```

Figure 17. Linux Kernel Running in HyperTerminal

- 300 MHz Processor Clock Frequency
- 100 MHz Bus Clock Frequency

- No On-chip Memory
- LEDs_Positions, Ethernet_MAC, TriMode_MAC_GMII, SRAM, and FLASH disabled
- Use Interrupts on IIC_EEPROM, LEDs_4Bit, RS232_Uart, PushButton_Position, and SysACE_CompactFlash
- Change the RS232_Uart to xps_uart16550 and Configure as UART 16550
- Change xps_bram_if_cntlr_1 size to 64 KB
- Enable Instruction and Data Cache – DDR SDRAM should hold both instruction and data cache

Figure 18 shows the new XPS project system assembly view.


Xilinx Platform Studio - C:\Thesis13\system.xmp - [System Assembly View]

Project Files
- MHS File: system.mhs
- MSS File: system.mss
- UCF File: data/system.ucf
- iMPACT Command File: etc/download.cmd
- Implementation Options File: etc/fast_runtime.opt
- Bitgen Options File: etc/bitgen.ut

Project Options
- Device: xc4vfx20ff672-10
- Netlist: TopLevel
- Implementation: XPS (Xflow)
- HDL: VHDL
- Sim Model: BEHAVIORAL

Design Summary

Name	Bus Name	IP Type	IP Version
plb		plb_v46	1.05.a
ppc405_0_d...		plb_v46	1.05.a
ppc405_0_ipl...		plb_v46	1.05.a
ppc405_0		ppc405_virt...	2.01.b
plb_bram_if_...		bram_block	1.00.a
xps_bram_if...		xps_bram_if...	1.00.b
DDR_SDRAM		mpmc	6.02.a
xps_intc_0		xps_intc	2.01.a
jtagppc_cntlr...		jtagppc_cntlr	2.01.c
proc_sys_res...		proc_sys_re...	3.00.a
LEDs_4Bit		xps_gpio	2.00.a
Push_Button...		xps_gpio	2.00.a
IIC_EEPROM		xps_iic	2.03.a
SysACE_Com...		xps_sysace	1.01.a
RS232_Uart		xps_uart165...	3.00.a
clock_genera...		clock_gener...	4.01.a
MGT_wrapper		mgt_protector	1.00.a

Bus Interface Filters
- By Connection: ✓ Connected, ✓ Unconnected
- By Bus Standard: ✓ PLBV46, ✓ OCM, ✓ DCR, ✓ FCB, ✓ Xilinx Point To Point (✓ XIL_BRAM, ✓ XIL_EMACDCR, ✓ XIL_FCM, ✓ XIL_JTAGPPC, ✓ XIL_RESETPPC)
- By Interface Type: ✓ Slaves, ✓ Masters, ✓ Master Slaves, ✓ Monitors, ✓ Targets, ✓ Initiators

Console
```
Copied C:/Xilinx/12.4/ISE_DS/EDK/data/xflow/bitgen.ut to etc directory
Generated Block Diagram.
Diagram Controls
Zoom In/Out = ALT + (Mouse + Left Button) or ARROW UP/DOWN.
Pan = SHIFT + (Mouse + Left Button) or ARROW UP/DOWN/LEFT/RIGHT.
```

Figure 18. XPS Project System Assembly View

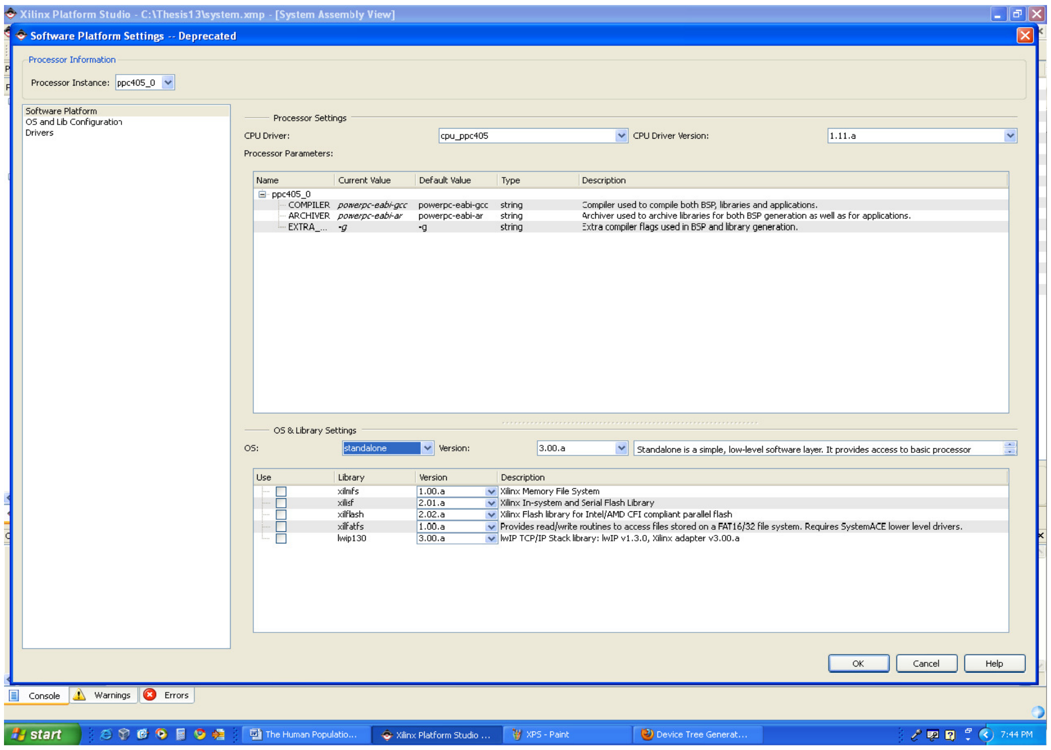
The system assembly view shows all of the intellectual properties that are being used and their bus connections.

The device tree generator produces a device tree file (xilinx.dts) that contains information about the current project's hardware design. It has information like hardware peripherals' base addresses, bus compatibility, and information on peripheral interrupts. The Linux kernel uses the generated device tree as a window to communicate with the peripherals in the XPS design. The VM was used to obtain the device tree generator using the following command:

- **git clone git://git.xilinx.com/devict-tree.git**

This command first created an empty repository in the current directory and downloads the device tree into it. Inside the device tree directory there was a subdirectory called "bsp". This directory was copied to the top of the "Thesis12" project directory. The user repositories were then rescanned so that the project could use the device tree generator. This was done by selecting "Rescan User Repositories" under the XPS Project tab [49].

Once the "bsp" directory was recognized by the project, the software settings were configured by selecting "Software Platform Settings" under the Software menu. The page shown in Figure 19 appeared after selecting this option. With the device tree generator now being recognized, a new option called "device-tree" was available when the "OS" window was expanded. Once the "device-tree" OS was selected, the configuration for the OS was set under the "OS and LIB Configuration" tab. Expanding "device-tree" invoked two configuration settings: "bootargs" and "console device". Each setting had a current value which was initially blank, a default value, a type, and a description. For “bootargs”, the current value was set to **console=ttyS0 ip=on root=/dev/ram.** The current value for “console device” was set to **RS232_Uart.**



Processor Instance: ppc405_0

CPU Driver: cpu_ppc405 CPU Driver Version: 1.11.a

Processor Parameters:

Name	Current Value	Default Value	Type	Description
ppc405_0				
COMPILER	*powerpc-eabi-gcc*	powerpc-eabi-gcc	string	Compiler used to compile both BSP, libraries and applications.
ARCHIVER	*powerpc-eabi-ar*	powerpc-eabi-ar	string	Archiver used to archive libraries for both BSP generation as well as for applications.
EXTRA_...	*-g*	-g	string	Extra compiler flags used in BSP and library generation.

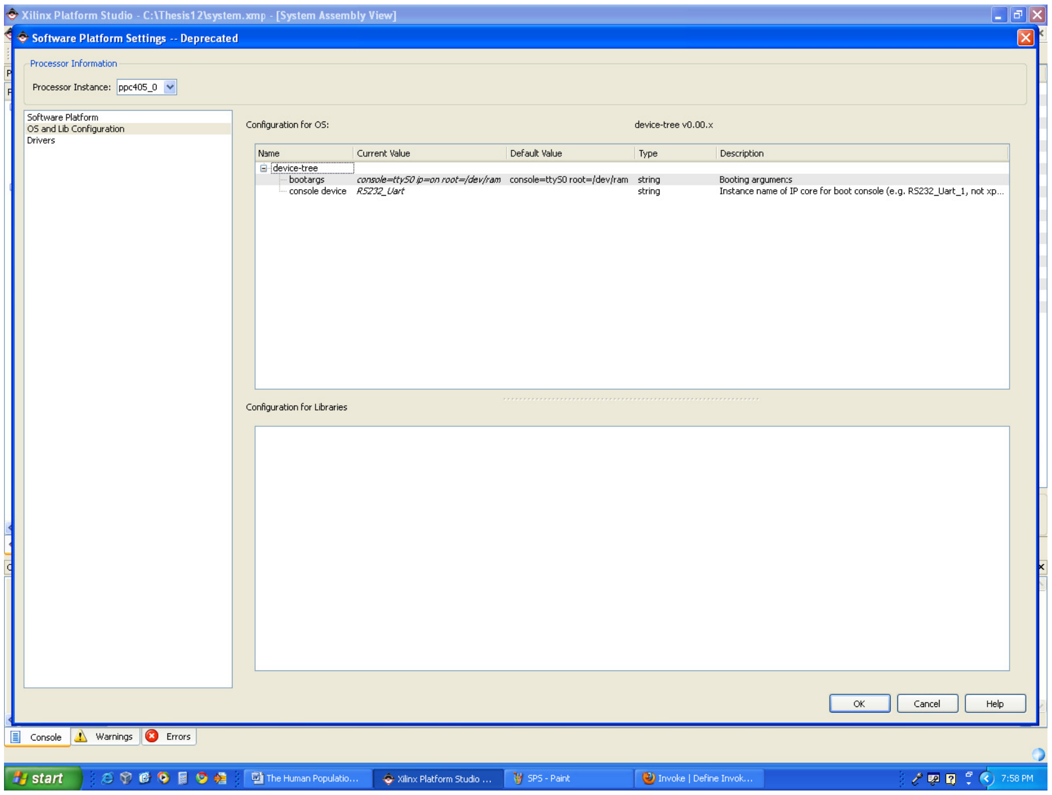
OS: standalone Version: 3.00.a Standalone is a simple, low-level software layer. It provides access to basic processor

Use	Library	Version	Description
☐	xilmfs	1.00.a	Xilinx Memory File System
☐	xilisf	2.01.a	Xilinx In-system and Serial Flash Library
☐	xilflash	2.02.a	Xilinx Flash library for Intel/AMD CFI compliant parallel flash
☐	xilfatfs	1.00.a	Provides read/write routines to access files stored on a FAT16/32 file system. Requires SystemACE lower level drivers.
☐	lwip130	3.00.a	lwIP TCP/IP Stack library: lwIP v1.3.0, Xilinx adapter v3.00.a

Figure 19. XPS Software Platform Settings Page

Figure 20 displays the “OS and Lib Configuration” tab with completed settings. Nothing else needed to be done in this window, so the software platform settings window was closed and the settings were saved by pressing the “OK” button at the bottom of the window. The device tree was then created by selecting “Generate Libraries and BSPs” option in the Software menu. At this point, XPS created and saved the device tree “xilinx.dts” in the \ppc405_0\libsrc\device-tree_v0_00_x directory of the current project.

The name of the device tree was changed from “xilinx.dts” to “virtex405-thesis12”. It was still recognized as a dts file even without the dts extension in the name.


Xilinx Platform Studio - C:\Thesis12\system.xmp - [System Assembly View]

Software Platform Settings -- Deprecated

Processor Information

Processor Instance: ppc405_0

Software Platform
OS and Lib Configuration
Drivers

Configuration for OS: device-tree v0.00.x

Name	Current Value	Default Value	Type	Description
device-tree				
bootargs	*console=ttyS0 ip=on root=/dev/ram*	console=ttyS0 root=/dev/ram	string	Booting argumen:s
console device	*RS232_Uart*		string	Instance name of IP core for boot console (e.g. RS232_Uart_1, not xp...

Configuration for Libraries

Figure 20. XPS Software Platform Settings “OS and Lib Configuration” Page with Complete Settings

In order for the VM to use this device tree to create another elf file for the board, the device tree first needed to be copied into the arch/powerpc/boot/dts directory of the Linux tree on the VM. The new elf file was created using the following command:

- **create.ram.kernel-thesis12.**

The resulting elf file was “simpleImage.initrd.virtex405-thesis12.elf”, which was copied from the VM to the top of the “Thesis12” directory. A new bitstream for this project was generated by selecting “Generate Bitstream” under the Hardware menu. Once the bistream was complete, it was downloaded to the board by selecting “Download Bitstream” under the “Device Configuration” menu. As before with the reference design, the new elf file was downloaded by launching and using XMD. The only difference was the elf file that was downloaded. Instead of downloading the original elf file with the original device tree, the new elf file was downloaded by issuing the command:

- **dow simpleImage.initrd.virtex405-thesis12.elf.**

The running Linux kernel behaved exactly like the kernel running with the reference design [49].

8.2 Customizing the Linux Kernel

With the default kernel working on the newly created XPS project, the next step was to customize the Linux kernel, rebuild it, and download it back onto the ML405 board. Before the Linux kernel could be customized, the ramdisk on the kernel needed to be expanded. The default ramdisk from xilinx.wikidot.com was only 4 MB. An 8 MB ramdisk needed to be created to ensure that there was enough space to store the “Hello World” program. The only feasible way to expand the ramdisk was to create a new larger ramdisk and copy the contents of the default ramdisk over to it. The “Expanding File System” page of http://xilinx.wikidot.com listed all of the steps in order to expand the ramdisk. A new subdirectory (“ramdisks”) was created under the “xilinx” subdirectory in the VM to hold all of the newly created ramdisks. Since larger ramdisks were going to be needed as more programs were added to the ramdisk, several subdirectories (based on ramdisk size) were created under the "ramdisk" subdirectory as well. This just made it easier to keep track of the different sized ramdisks. Since creating an 8 MB ramdisk was the topic of interest, the current directory was changed to **~/xilinx/ramdisks/8M** so that the new 8 MB ramdisk would be created inside the 8M subdirectory. The following command actually created a ramdisk called "ramdisk.image".

- **sudo dd if=/dev/zero of=ramdisk.image bs=1M count=8**

This command created a set of 8 1 MB blocks initially valued at zero. This command is a permission restricted command, hence the sudo authorization at the beginning of the command [49].

Next, "ramdisk.image" had to be designated a certain filesystem type. There are several different filesystem types supported by Linux. Some of the more common filesystem types are ext2, ext3, FAT32, and iso9660. The new ramdisk was designated an ext2 filesystem using the command

- **sudo mke2fs –F –v –m0 ramdisk.image.**

Declaring the ramdisk's filesystem type required root privileges as well. A new directory, "/mnt/new-disk" was created to mount the new ramdisk, and "/mnt/old-disk" was created to mount the default ramdisk. The contents of the default ramdisk were copied to the new ramdisk using the command

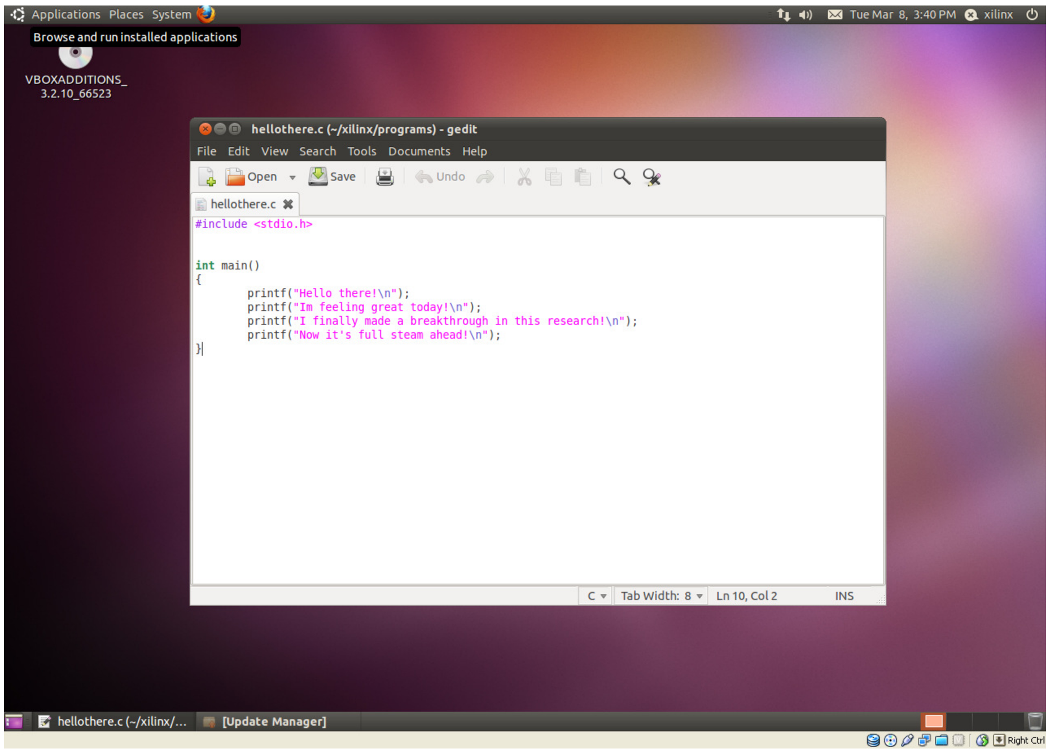
- **sudo cp /mnt/old-disk/* /mnt/new-disk**.

The asterisk indicated that all of the contents of that directory are to be copied. The default ramdisk was no longer needed at this point and was unmounted. The new ramdisk was left mounted because files were added into it later [49].

The next step was to modify the Linux kernel. Writing a simple C file, compiling it to a binary file, and adding it the /bin directory of the ramdisk were the procedures involved in modifying the kernel. Figure 21 shows the “hellothere.c” file in the VM's “gedit” text editor. This program simply prints the designated message to the HyperTerminal console. The messages and keywords only appeared in color after the file was saved as a C file. The Linux kernel couldn’t use the actual C file, so the C file was compiled into an executable file called “hellothere” using the following command

- **ppc_4xx-gcc –o hellothere –static hellothere.c**

The ramdisk’s /bin directory received a copy of the executable since the ramdisk was still mounted to the /mnt/new-disk directory. The addition of the new custom binary file made this ramdisk complete. The ramdisk was then unmounted, compressed using the “gzip” command, and copied to the arch/powerpc/boot directory of the Linux tree. Since the name of the new ramdisk was the same as the old ramdisk, Terminal issued a prompt to enter “y” to overwrite the old disk or “n” not to. The new 8 MB ramdisk took the place of the old 4 MB ramdisk. The **create.ram.kernel-thesis12** command was then issued to create and overwrite the previously existing elf file [49]. Once the bitstream and elf file were downloaded onto the board, the Linux kernel was able to call the “hellothere” binary program as a Linux command.

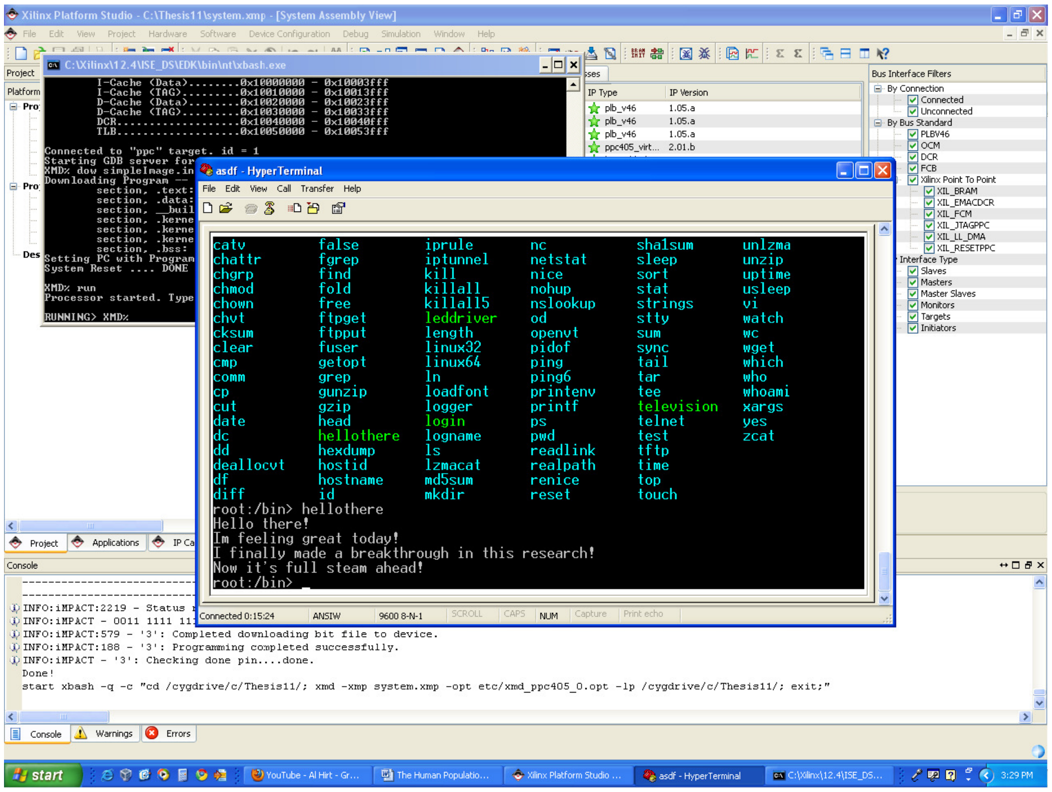

Applications Places System
Tue Mar 8, 3:40 PM xilinx
Browse and run installed applications
VBOXADDITIONS_ 3.2.10_66523
hellothere.c (~/xilinx/programs) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
hellothere.c
C Tab Width: 8 Ln 10, Col 2 INS
hellothere.c (~/xilinx/... [Update Manager]
Right Ctrl

```c
#include <stdio.h>


int main()
{
	printf("Hello there!\n");
	printf("Im feeling great today!\n");
	printf("I finally made a breakthrough in this research!\n");
	printf("Now it's full steam ahead!\n");
}
```

Figure 21. “hellothere” C file in VM Text Editor

The executed command through HyperTerminal is shown in Figure 22. The name of the binary file is shown in the /bin directory in green letters as well as the printed message when the program was called. The “hellothere.c” code is included in the Appendix section. Adding custom binary files made the Linux kernel more robust and the kernel could be modified to perform certain tasks. Next was a test to see if the kernel could actually interact with the hardware on the ML405 board. An LED driver C file, found on the internet, mapped the base address of the LED peripheral into the kernel memory and toggled the least significant bit (LSB) light whenever a number was entered followed by the enter key.



```
         I-Cache (Data)........0x10000000 - 0x10003fff
         I-Cache (TAG).........0x10010000 - 0x10013fff
         D-Cache (Data)........0x10020000 - 0x10023fff
         D-Cache (TAG).........0x10030000 - 0x10033fff
         DCR...................0x10040000 - 0x10040fff
         TLB...................0x10050000 - 0x10053fff

Connected to "ppc" target. id = 1
Starting GDB server for
XMD% dow simpleImage.in
Downloading Program --
         section, .text:
         section, .data:
         section, __buil
         section, .kerne
         section, .kerne
         section, .kerne
         section, .bss:
Setting PC with Program
System Reset .... DONE

XMD% run
Processor started. Type

RUNNING> XMD%
```

```
catv        false       iprule      nc          sha1sum     unlzma
chattr      fgrep       iptunnel    netstat     sleep       unzip
chgrp       find        kill        nice        sort        uptime
chmod       fold        killall     nohup       stat        usleep
chown       free        killall5    nslookup    strings     vi
chvt        ftpget      leddriver   od          stty        watch
cksum       ftpput      length      openvt      sum         wc
clear       fuser       linux32     pidof       sync        wget
cmp         getopt      linux64     ping        tail        which
comm        grep        ln          ping6       tar         who
cp          gunzip      loadfont    printenv    tee         whoami
cut         gzip        logger      printf      television  xargs
date        head        login       ps          telnet      yes
dc          hellothere  logname     pwd         test        zcat
dd          hexdump     ls          readlink    tftp
deallocvt   hostid      lzmacat     realpath    time
df          hostname    md5sum      renice      top
diff        id          mkdir       reset       touch
root:/bin> hellothere
Hello there!
Im feeling great today!
I finally made a breakthrough in this research!
Now it's full steam ahead!
root:/bin> _
```

```
-----------------------------
-----------------------------
INFO:iMPACT:2219 - Status r
INFO:iMPACT - 0011 1111 11
INFO:iMPACT:579 - '3': Completed downloading bit file to device.
INFO:iMPACT:188 - '3': Programming completed successfully.
INFO:iMPACT - '3': Checking done pin....done.
Done!
start xbash -q -c "cd /cygdrive/c/Thesis11/; xmd -xmp system.xmp -opt etc/xmd_ppc405_0.opt -lp /cygdrive/c/Thesis11/; exit;"
```

Xilinx Platform Studio - C:\Thesis11\system.xmp - [System Assembly View]
C:\Xilinx\12.4\ISE_DS\EDK\bin\nt\xbash.exe
asdf - HyperTerminal
Connected 0:15:24
ANSIW
9600 8-N-1

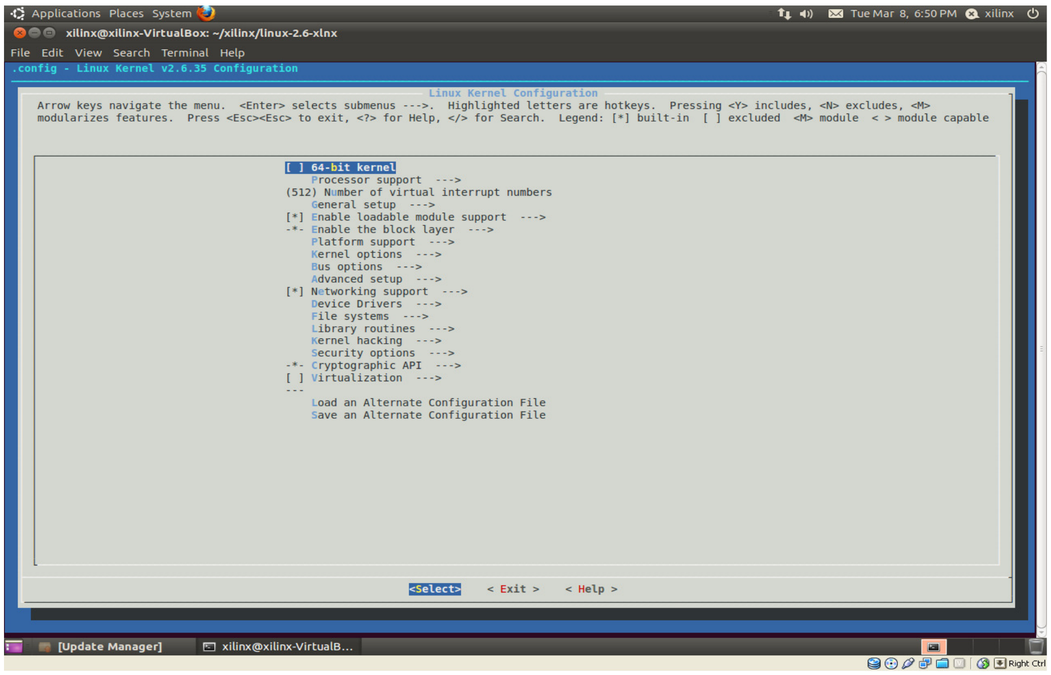
Figure 22. Linux Kernel Executing "hellothere" Binary File

The binary file for the LED driver is also shown in the ramdisk's /bin directory in Figure 22. The LED peripheral contained at least one software accessible register so once the LED driver's functionality was verified, the code was copied to another C file and modified to read the data from the register, write data to the register, and display the least significant hexadecimal digit of the first software accessible register on the four LEDs. The new C file was named "lightcontrol.c" and is included in the Appendix section. The binary file "lightcontrol" was created by compiling the "lightcontrol.c" file. However, the ramdisk was too small to store any more binary files. The following steps show how to create the 16 MB ramdisk, add the new static files to the new ramdisk, and configure the Linux kernel to handle the larger ramdisk:

1. Create the 16 MB ramdisk following the same procedures used to create the 8 MB ramdisk.
    i. Kept in the newly created ~/xilinx/ramdisks/16M directory
2. Copy the contents of the 8 MB ramdisk, including the custom binary files, to the 16 MB ramdisk.
3. Add the "lightcontrol" binary file to the /bin directory of the new ramdisk.
4. Unmount, zip, and copy the new ramdisk to the /arch/powerpc/boot directory of the Linux kernel tree.
5. Change the default ramdisk size for the Linux kernel
    i. Launch the kernel's configuration menu (Figure 23) by issuing the command **make ARCH=powerpc menuconfig**
    ii. Navigate to the "Block devices" submenu which is located under the "Device Drivers" menu.
    iii. Change the "Default RAM disk size (kbytes)" value from "8192" to "16384".
    iv. Exit the kernel configuration

Adding a 16 MB ramdisk no longer caused any problems.

The newly added programs required a new hardware peripheral. Since the LED peripheral was an XPS general purpose input/output (GPIO) peripheral, it couldn't be modified. A new peripheral called "simple_register" was created to test the "lightcontrol" binary file. The behavior of the new peripheral was edited using Xilinx



Applications Places System
Tue Mar 8, 6:50 PM xilinx
xilinx@xilinx-VirtualBox: ~/xilinx/linux-2.6-xlnx
File Edit View Search Terminal Help
[Update Manager]
xilinx@xilinx-VirtualB...
Right Ctrl

```
.config - Linux Kernel v2.6.35 Configuration
                          Linux Kernel Configuration
 Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M>
 modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                       [ ] 64-bit kernel
                           Processor support  --->
                       (512) Number of virtual interrupt numbers
                           General setup  --->
                       [*] Enable loadable module support  --->
                       -*- Enable the block layer  --->
                           Platform support  --->
                           Kernel options  --->
                           Bus options  --->
                           Advanced setup  --->
                       [*] Networking support  --->
                           Device Drivers  --->
                           File systems  --->
                           Library routines  --->
                           Kernel hacking  --->
                           Security options  --->
                       -*- Cryptographic API  --->
                       [ ] Virtualization  --->
                       ---
                           Load an Alternate Configuration File
                           Save an Alternate Configuration File

                          <Select>    < Exit >    < Help >
```

Figure 23. Linux Kernel Configuration Menu

Project Navigator to map the last hexadecimal digit of the software accessible register to the LEDs. The peripheral was then synthesized and imported back to XPS where it was added to the project. The new peripheral was then given an address, and its output ports were tied to the LED pins in the ucf file. Changing the hardware meant that a new device tree and bitstream needed to be generated. The new device tree was name **virtex405-thesis12-1.**

While XPS was generating a new bitstream to accommodate the new "simple_register" peripheral, the **create.ram.kernel-thesis12-1** command was issued in the VM to create a new elf file containing the Linux kernel with the 16 MB ramdisk. The new bitstream was downloaded onto the board once it was generated followed by the new "simpleImage.initrd.virtex405-thesis12-1.elf" file using XMD. The hellothere, leddriver, and lightcontrol binary files functioned as expected when the kernel was loaded and run on the ML405 board. The hexadecimal value shown on the LED lights changed whenever the software accessible register received a new value.

Similar C programs, including a "television", "lights", "tvon", "tvoff", "lightson", "lightsoff", "doorlock" and "doorunlock" were written in the VM text editor, compiled into binary files, and tested in the Linux kernel. These programs were written to turn on an LED when a certain value was written to the peripheral's software accessible register and turn the LED off when another value was written to the register. "Leddriver.c" was used as a reference to write these programs in terms of mapping the hardware peripherals' base address to the kernel's memory. One peripheral was created for the programs involved with lights (lights, lightson, and lightsoff) to control. Another peripheral was created for the programs involved with television (television, tvon, tvoff) to control. A third peripheral was created for the "doorlock" and "doorunlock" programs to control. Each one of these peripherals was tied to a separate LED light. Whenever the kernel was running and the "tvon" command was issued, LED1 would turn on. "lightson" turned on LED0. Both "television" and "lights" asked how long the lights of television stayed on and turned off after the specified time interval. The "doorlock" and "doorunlock" programs powered the LED light for 0.5 seconds and turned off to simulate a door being locked or unlocked.

The next task was to configure the television and light peripherals to control something that represented an appliance or an appliance actuator or sensor. The Tamiya

FA-130 motor was a suitable representation. The Tamiya FA-130 is a small DC motor that the Tamiya company uses in some of its gearboxes. The basic specifications are listed below:

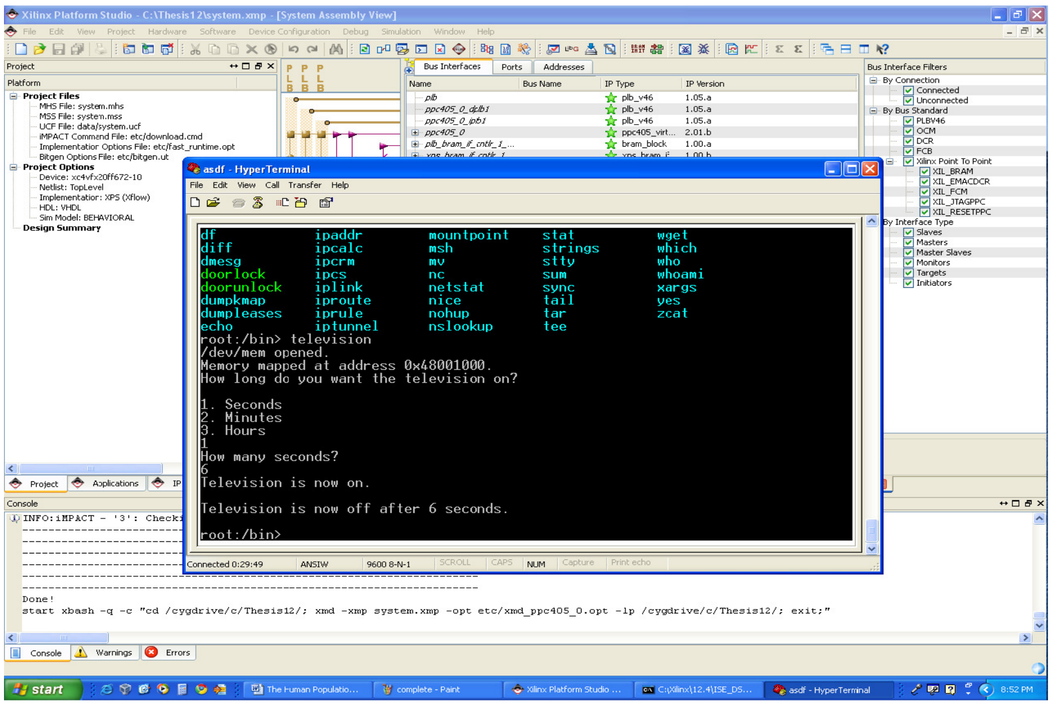
- RPM: 6990-9100 (6990 Max. Efficiency)
- Voltage: 1.5-3V (1.5V Recommended) (4.5 V max)
- Amperage: .66A
- Stall torque: 4.6 gcm [51]

An H bridge was used to power the motors since the motors were too small to be powered by the ML405 board directly. The H bridge used an external power supply to power the motors and data signals from the expansion headers of the ML405 to enable and disable the motors. One H bridge could be used to control two FA-130 motors.

Slight changes were made to the behavior of the custom hardware peripherals and the ucf file to power the motors. Instead of each peripheral having one output tied to an LED light, the peripherals were modified to have two outputs tied to two expansion header pins to send data signals to the H bridge. A separate peripheral was created to send the ground and voltage signals to the H bridge since these two signals don't change. Since the door peripheral would power one motor for "doorlock" and another motor for "doorunlock", this peripheral was edited to have four outputs tied to four expansion header pins. Another peripheral was created to control the ground and voltage signals going to the H bridge. To include the updated peripherals, a new bitstream and device tree (named "virtex405-thesis12-5") were generated.

In order for the Linux kernel to have sufficient storage space to hold all of the binary files, a 32 MB ramdisk was created under the **~/xilinx/ramdisks/32M** directory. The ramdisk was then mounted to the **/mnt/new-disk** directory and the contents of the 16 MB ramdisk were copied to the larger ramdisk. Any custom binary files not included in the new ramdisk's /bin directory were copied from the **~/xilinx/programs** directory. The kernel configuration menu was launched to change the default RAM disk size from "16384" to "32768" to support a 32 MB ramdisk. The new ramdisk was then unmounted, compressed and copied to the Linux kernel's arch/powerpc/boot directory in the VM's Terminal application. A new elf file "simpleImage.initrd.virtex405-thesis12-5" was created by issuing the **create.ram.kernel-thesis12-5** command. The new elf file was then copied to the XPS "Thesis12" project directory. The newly generated bitstream along with the new elf file were then ready to be downloaded onto the ML405 board.

The final project, which included peripherals and custom programs to control light, television, and door actuators, was successfully completed and functioned as expected. The "lightson" and "tvon" programs activated their respective peripheral's Tamiya motor and printed a message to HyperTerminal that the lights or the television is now on. The "lightsoff" and "tvoff" programs deactivated their peripheral's motor and printed a message to HyperTerminal that the lights or the television is now off. The "lights" and "television" programs first asked if the lights of television wanted to be turned on for a period of seconds, minutes or hours. Once an option was selected, the program prompted for an interval of the selected time period. The respective motor was then activated and deactivated after the specified time period. HyperTerminal printed a message that the appliance was now on and off after designated time period. For example, if the television was designated to stay on for 5 seconds, HyperTerminal printed the message “Television is now on”, and the television motor turned off after 5 seconds. The HyperTerminal console then printed the message “Television is now off after 5 seconds. The “doorlock” program activated its motor for 0.5 seconds to simulate a door automatically being locked. The message “The door is now locked” was printed onto the console window while the motor was activated. The “doorunlock” program worked in a similar way as the “doorlock” program. The only difference was that the “doorunlock” program activated a different motor and displayed “The door is now unlocked” in the HyperTerminal window. The executed “television” program is displayed in the HyperTerminal console in Figure 24. A 32 MB ramdisk was more than enough space to contain the extra binary files. If a programmer wanted to create programs to control other appliances such as a ceiling fan and add them the Linux kernel, there is room on the ramdisk to do it.



```
df              ipaddr          mountpoint      stat            wget
diff            ipcalc          msh             strings         which
dmesg           ipcrm           mv              stty            who
doorlock        ipcs            nc              sum             whoami
doorunlock      iplink          netstat         sync            xargs
dumpkmap        iproute         nice            tail            yes
dumpleases      iprule          nohup           tar             zcat
echo            iptunnel        nslookup        tee
root:/bin> television
/dev/mem opened.
Memory mapped at address 0x48001000.
How long do you want the television on?

1. Seconds
2. Minutes
3. Hours
1
How many seconds?
6
Television is now on.

Television is now off after 6 seconds.

root:/bin>
```

```
INFO:iMPACT - '3': Checki
-------------------------
-------------------------
-------------------------
-------------------------
--------------------------------------------------------------
--------------------------------------------------------------
Done!
start xbash -q -c "cd /cygdrive/c/Thesis12/; xmd -xmp system.xmp -opt etc/xmd_ppc405_0.opt -lp /cygdrive/c/Thesis12/; exit;"
```

Figure 24. Complete Modified Linux Kernel Executing “Television” Program

CHAPTER 9

CONCLUSION

The U.S. and world populations are growing and getting older simply because more people are living longer. This is due to breakthroughs in science, medicine and technology that many people take for granted today. The probability of developing a disability increases as a person gets older. Not everybody can afford the current options that are available to assist the disabled elderly. However, a possible cost friendly solution may be found in automated homes, which is the motivation for the SAAD project. The SAAD project is composed of several subprojects, including the subproject that was named WireAct.

One of the goals of WireAct was to customize the Linux kernel to interface with custom hardware peripherals on an FPGA board. Several experiments were conducted to achieve this goal including:

1. Building, downloading and running the default kernel
2. Customizing the kernel
3. Creating a new project to run the kernel
4. Customizing the kernel to interface with an FPGA's hardware
5. Customizing the kernel to interface with representations of appliance actuators in the form of DC motors

This was a tremendous milestone in WireAct. The results at hand show that an FPGA can be configured to control appliance actuators and sensors, which also contributes to the overall progress of the SAAD project. Future research may include developing a graphical user interface (GUI) using C# object oriented programming to interact with the ML405 board through serial communication. As of right now, actuator timing is handled by the Power PC microprocessor. Eventually each hardware peripheral will have its own timer so that the microprocessor can tend to other program executions. Wireless USB communication between the ML405 board and appliance actuators and sensors, as well as the ML405 and a host computer, is another subject of future research.

BIBLIOGRAPHY

1. Oberlin College Presentation. (2010, February 16). Retrieved January 15, 2011 from http://www.vhemt.org/oar.htm

2. Population Reference Bureau. (n.d.). Retrieved January 20, 2011 from http://www.prb.org/Educators/TeachersGuides/HumanPopulation/PopulationGrowth.aspx

3. What is the bubonic plague?. (n.d.). Retrieved January 15, 2011 from http://www.essortment.com/all/bubonicplague_rmhr.htm

4. USA Population. (n.d.). Retrieved January 15, 2011 from http://geography.about.com/od/obtainpopulationdata/a/uspopulation.htm

5. US & World Population Clock. (n.d.). Retrieved January 15, 2011 from http://www.census.gov/population/www/popclockus.html

6. CIA – The World Factbook. (n.d.). Retrieved January 19, 2011 from https://www.cia.gov/library/publications/the-world-factbook/rankorder/2119rank.html?countryName=United%20States&countryCode=us&regionCode=na&rank=3#us

7. sixbilpart1.pdf (n.d.). Retrieved January 15, 2011 from http://www.un.org/esa/population/publications/sixbillion/sixbilpart1.pdf

8. Life Expectancy. (n.d.). Retrieved January 17,2011 from http://www.efmoody.com/estate/lifeexpectancy.html

9. Life Expectancy. (n.d.). Retrieved January 17, 2011 from http://en.wikipedia.org/wiki/Life_expectancy#Human_life_expectancy_patterns

10. 2010 Average Life Expectancy by Gender, Race and Country. (2010, October 20). Retrieved January 17, 2011 from http://www.suite101.com/content/2010-life-expectancy-longevity-factors-and-the-latest-news-a284558

11. Penicillin. (n.d.). Retrieved January 17, 2011 from http://en.wikipedia.org/wiki/Penicillin

12. Milestones in Medicine. (n.d.). Retrieved January 17, 2011 from http://www.stjohnprovidence.org/HealthInfoLib/swArticle.aspx?1,1015

13. Medical Technology Instruments | eHow.com (2010, July 22). Retrieved January 19, 2011 from http://www.ehow.com/list_6764123_medical-technology-instruments.html

14. Aging Population – Seniors Are Fastest Growing Population Worldwide. (n.d.). Retrieved January 19, 2011 from http://seniorliving.about.com/od/lifetransitionsaging/a/seniorpop.htm

15. Senior Population in the U.S. 2010 to 2050. (2010, May 20). Retrieved January 19, 2011 from http://www.disabled-world.com/disability/statistics/senior-population.php

16. AMDA: About AMDA – US Senior Population Demographics. (n.d.). Retrieved January 19, 2011 from http://www.amda.com/about/seniordemographics.cfm

17. Mann, W. (2005). Smart Technology for Aging, Disability, and Independence: The State of the Science p. 2

18. What is a Disability? . (n.d.). Retrieved January 19, 2011 from http://hcdg.org/definition.htm

19. The ADA Amendments Act of 2008. (2008, September 25). Retrieved January 19, 2011 from http://www.access-board.gov/about/laws/ada-amendments.htm

20. How Many Disabled Americans Are There?. (n.d.). Retrieved January 19, 2011 from http://ezinearticles.com/?How-Many-Disabled-Americans-AreThere?&id=1918829

21. About Living Trust Plus™. (n.d.). Retrieved January 19, 2011 from http://www.livingtrustplus.com/about/

22. Mann, W. (2005).Smart Technology for Aging, Disability, and Independence – The State of the Science p. 5

23. Aging in place. (n.d.). Retrieved January 24, 2011 from http://en.wikipedia.org/wiki/Aging_in_place

24. Mann, W. (2005).Smart Technology for Aging, Disability, and Independence – The State of the Science p. 33 – 38

25. Smart House – Definition of Smart House. (n.d.). Retrieved January 24, 2011 from - http://architecture.about.com/od/buildyourhous1/g/smarthouse.htm

26. Yaghmour, K. (2008). Building Embedded Linux Systems. Sebastopol, CA. O'Reilly Media, Inc.

27. History of Linux. (n.d.). Retrieved February 15, 2011 from http://en.wikipedia.org/wiki/History_of_Linux

28. History of Linux, Who Invented Linux, How Was Linux Invented. (n.d.). Retrieved February 15, 2011 from http://www.livinginternet.com/i/iw_unix_gnulinux.htm

29. Siever, E. (1999). Linux in a Nutshell. Sebastopol, CA. O'Reilly & Associates, Inc.

30. Naveenraja, S. Overview of Linux Kernel Structure. (n.d.). Retrieved February 15, 2011 from http://www.scribd.com/doc/19462050/Overview-of-Linux-Kernel-Structure

31. Linux Kernel. (n.d.). Retrieved February 24, 2011 from http://en.wikipedia.org/wiki/Linux_kernel

32. Langstedt, N (2005, September 22). Linux's directory structure. Retrieved February 24, 2011 from http://www.tuxfiles.org/linuxhelp/linuxdir.html

33. /bin definition by the Linux Information Project (LINFO). (2005, June 5). Retrieved February 24, 2011 from - http://www.linfo.org/bin.html

34. /sbin definition by the Linux Information Project (LINFO). (2007, July 19). Retrieved February 24, 2011 from - http://www.linfo.org/sbin.html

35. The Linux filesystem explained | FreeOS, free operating system. (2001, January 3). Retrieved March 8, 2011 from - http://www.freeos.com/articles/3102

36. /usr. (n.d.). Retrieved February 24, 2011 from http://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/usr.html

37. Linux Architecture – Overview – (Powerpoint Presentation)

38. Home directory. (n.d.). Retrieved February 24, 2011 from http://en.wikipedia.org/wiki/Home_directory

39. FPGA Design – A Brief History. (2010, July 18.). Retrieved March 3, 2011 from http://www.enventureonline.com/engineering-blog/fgpa-design/fpga-design-%E2%80%93-a-brief-history.html

40. Field-programmable gate array. (n.d.). Retrieved March 3, 2011 from http://en.wikipedia.org/wiki/Field-programmable_gate_array#History

41. Lee, S. (2006). Advanced Digital Logic Design Using VHDL, State Machines, and Synthesis for FPGAs. Toronto, Ontario. Nelson.

42. Xilinx : About Xilinx : Getting Started. (n.d.). Retrieved March 3, 2011 from http://www.xilinx.com/company/gettingstarted/index.htm

43. All about FPGAs. (2006, March 21). Retrieved March 3, 2011 from http://www.eetimes.com/design/programmable-logic/4014815/All-about-FPGAs

44. week-2.pdf. (n.d.). Retrieved March 3, 2011 from http://bertram-family.com/felix/soc_course_files/week-2.pdf

45. FPGA routing matrix | FPGA Central. (2009, February 11). Retrieved March 3, 2011 from http://www.fpgacentral.com/docs/fpga-tutorial/fpga-routing-matrix

46. VirtualBox. (n.d.). Retrieved March 7, 2011 from http://www.virtualbox.org/

47. Chapter 1. First steps (n.d.). Retrieved March 7, 2011 from http://www.virtualbox.org/manual/ch01.html

48. ISO image (n.d.). Retrieved March 3, 2011 from http://en.wikipedia.org/wiki/ISO_image

49. Welcome – Xilinx Open Source Wiki. (n.d.). Retrieved March 3, 2011 from http://xilinx.wikidot.com

50. sudo - Linux Command – Unix Command. (n.d.). Retrieved March 7, 2011 from http://linux.about.com/od/commands/l/blcmdl8_sudo.htm

51. Tamiya Motors. (n.d.). Retrieved February 28, 2011 from http://www.superdroidrobots.com/product_info/tamiya_motors.htm

APPENDIX A

```
//"hellothere.c" Code

#include <stdio.h>


int main()
{
	printf("Hello there!\n");
	printf("Im feeling great today!\n");
	printf("I finally made a breakthrough in this research!\n");
	printf("Now it's full steam ahead!\n");
}
```

APPENDIX B

```
//"leddrivever.c" Code

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>

// The purpose this test is to show that users can get to devices in user
// mode for simple things like GPIO. This is not to say this should replace
// a kernel driver, but does provide some short term solutions sometimes
// or a debug solution that can be helpful.

// This test maps a GPIO in the hardware into the user space such that a
// GPIO signal can be toggled fast. On the ML507 reference system, the
// signal could be toggled about every 50 ns which is pretty fast.

// This test was derived from devmem2.c.

#define GPIO_BASE_ADDRESS     0x81400000
#define GPIO_DATA_OFFSET     0
#define GPIO_DIRECTION_OFFSET     4
#define MAP_SIZE 4096UL
#define MAP_MASK (MAP_SIZE - 1)

int main()
{
   int memfd;
   int n;
   void *mapped_base, *mapped_dev_base;
   off_t dev_base = GPIO_BASE_ADDRESS;

   memfd = open("/dev/mem", O_RDWR | O_SYNC);
      if (memfd == -1) {
      printf("Can't open /dev/mem.\n");
      exit(0);
   }
   printf("/dev/mem opened.\n");
   // Map one page of memory into user space such that the device is in that page, but it
may not
   // be at the start of the page

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

   // get the address of the device in user space which will be an offset from the base
   // that was mapped as memory is mapped at the start of a page

   mapped_dev_base = mapped_base + (dev_base & MAP_MASK);

   // write to the direction register so all the GPIOs are on output to drive LEDs

   *((unsigned long *) (mapped_dev_base + GPIO_DIRECTION_OFFSET)) = 0;

   // toggle the output as fast as possible just to see how fast it works

   printf("enter 1\n");
   scanf ("%d", &n);
   printf("n is %d1\n",n);

   while (1) {
         printf("enter 1\n");
         scanf ("%d", &n);
         printf("n is %d1\n",n);
      *((unsigned long *) (mapped_dev_base + GPIO_DATA_OFFSET)) = 0;
         printf("enter 1\n");
         scanf ("%d", &n);
         printf("n is %d1\n",n);
      *((unsigned long *) (mapped_dev_base + GPIO_DATA_OFFSET)) = 1;
   }

   // unmap the memory before exiting

   if (munmap(mapped_base, MAP_SIZE) == -1) {
      printf("Can't unmap memory from user space.\n");
      exit(0);
   }
```

```
    close(memfd);
    return 0;
}
```

APPENDIX C

```
//C code to operate the television IP.  Simple modifications can be made to operate other
//IPs

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>

#define GPIO_BASE_ADDRESS     0XC2400000 //change the base address according to
                                          //the address of your
                                         //IP in your XPS project
#define REG_OFFSET          0
#define GPIO_DATA_OFFSET     0
#define GPIO_DIRECTION_OFFSET     4

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

int main ()
{
   int period;
   int period2;
   int time;
   int memfd;
   int n;
   char mode;
   void *mapped_base, *mapped_dev_base;
   off_t dev_base = GPIO_BASE_ADDRESS;

   memfd = open("/dev/mem", O_RDWR | O_SYNC);
      if (memfd == -1) {
      printf("Can't open /dev/mem.\n");
      exit(0);
   }
   printf("/dev/mem opened.\n");

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

          mapped_dev_base = mapped_base + (dev_base & MAP_MASK);

    // write to the direction register so all the GPIOs are on output to drive LEDs

    *((unsigned long *) (mapped_dev_base + GPIO_DIRECTION_OFFSET)) = 0;

//Skip this section to simply turn the IP on or off
          printf("How long do you want the television on?\n\n");
          printf("1. Seconds\n2. Minutes\n3. Hours\n");
          scanf("%d",&time);
          if (time == 1)
          {
                  printf("How many seconds?\n");
          }
          else if (time == 2)
          {
                  printf("How many minutes?\n");
          }
          else if (time == 3)
          {
                  printf("How many hours?\n");
          }
          scanf("%d",&period);
          if (time == 1)
          {
                  period2 = period;
          }
          else if (time == 2)
          {
                  period2 = period * 60;
          }
          else if (time == 3)
          {
                  period2 = period * 3600;
          }



// To turn IP on
```

```
        *((unsigned long *) (mapped_dev_base + GPIO_DATA_OFFSET)) =
0X0000000F; //Value can be changed
                //according to the
                //value that the IP is
                //looking for
        printf("Television is now on.\n\n");
        sleep(period2); //To keep IP on for specified time interval
//To turn IP off
        *((unsigned long *) (mapped_dev_base + GPIO_DATA_OFFSET)) =
0X0000000E; //Value can be changed
                //according to the
                //value that the IP is
                //looking for
//Prints to the console that the IP is now off after specified time period
        if (time == 1)
        {
                printf("Television is now off after %d seconds.\n\n",period);
        }
        else if (time == 2)
        {
                printf("Television is now off after %d minutes.\n\n",period);
        }
        else if (time == 3)
        {
                printf("Television is now off after %d hours.\n\n",period);
        }
}
```

APPENDIX D

```
--Light Controller USER logic VHDL code

--USER logic implementation added here
  process(slv_reg0)
          begin
                    if (slv_reg0 = X"0000000F") then
                              lights <= "10";
                    elsif (slv_reg0 = X"0000000E") then
                              lights <= "11";
                    end if;
          end process;
```

APPENDIX E

-- Television Controller USER logic VHDL code

```
--USER logic implementation added here
  process(slv_reg0)
        begin
                if (slv_reg0 = X"0000000F") then
                        television <= "10";
                elsif (slv_reg0 = X"0000000E") then
                        television <= "11";
                end if;
        end process;
```

APPENDIX F

```
-- Door lock Controller USER logic VHDL code

--USER logic implementation added here
  voltage <= '0';
  ground <= '0';
  process(slv_reg0)
  begin
        if (slv_reg0 = X"0000000F") then
                lock <= "10";
                unlock <= "11";
        elsif (slv_reg0 = X"0000000D") then
                lock <= "11";
                unlock <= "10";
        elsif (slv_reg0 = X"0000000E") then
                lock <= "11";
                unlock <= "11";
        end if;
  end process;
```