**CHAPTER 1**

**INTRODUCTION**

The project is aimed at evaluating the performance of an operating system on an embedded system. Before delving into its implementation, an introduction is needed to the parts involved in the project. The whole report is centered around the field of embedded systems and the use of Linux to run applications on them. Hence an introduction to Embedded Systems and using Linux as an OS in them is provided.

**1.1 Embedded Systems**

An embedded system is a special purpose computer system that is designed to perform very small sets of designated activities. Embedded systems date back as early as the late 1960s where they used to control electromechanical telephone switches. The first recognizable embedded system was the Apollo Guidance Computer developed by Charles Draper and his team. Later they found their way into the military, medical sciences and the aerospace and automobile industries.

Today they are widely used to serve various purposes like:

* Network equipment such as firewall, router, switch, and so on.
* Consumer equipment such as MP3 players, cell phones, PDAs, digital cameras, camcorders, home entertainment systems and so on.
* Household appliances such as microwaves, washing machines, televisions and so on.
* Mission-critical systems such as satellites and flight control.
* The key factors that differentiate an embedded system from a desktop computer:
* They are cost sensitive.
* Most embedded systems have real time constraints.
* There are multitudes of CPU architectures such as ARM, MIPS, PowerPC that are used in embedded systems. Application-specific processors are employed in embedded systems.
* Embedded Systems have and require very few resources in terms of ROM or other I/O devices as compared to a desktop computer.

**1.1.1 Types of Setup**

Embedded systems generally have a setup that includes a host which is generally a personal computer, and a target that actually executes all the embedded applications. The various types of host/ desktop architectures that are used in embedded systems are:

Linked Setup:

In this setup, the target and the host are permanently linked together using a physical cable. This link is typically a serial cable or an Ethernet link. The main property of this setup is that no physical hardware storage device is being transferred between the target and the host. The host contains the cross-platform development environment while the target contains an appropriate boot loader, a functional kernel, and a minimal root file system.

Removable Storage Setup:

In the removable setup, there are no direct physical links between the host and the target. Instead, a storage device is written by the host, is then transferred into the target, and is used to boot the device. The host contains the cross-platform development environment. The target, however, contains only a minimal boot loader. The rest of the components are stored on a removable storage media, such as a Compact Flash IDE device, MMC Card, or any other type of removable storage device.

Standalone Setup:

The target is a self-contained development system and includes all the required software to boot, operate, and develop additional software. In essence, this setup is similar to an actual workstation, except the underlying hardware is not a conventional workstation but rather the embedded system itself. This one does not require any cross-platform development environment, since all development tools run in their native environments. Furthermore, it does not require any transfer between the target and the host, because all the required storage is local to the target.

**1.1.2 Operating Systems**

In an embedded system, when there is only a single task that is to be performed, then only a binary is too loaded into the target controller and is to be executed. However, when there are multiple tasks to be executed or multiple events to be handled, then there has to be a program that handles and prioritizes these events. This program is the Operating System (OS), which one is very familiar with, in desktop PCs.

Various Operating Systems:

Embedded Operating Systems are classified into two categories:

**1. Real-time Operating Systems (RTOS):**

Real Time Operating Systems are those which guarantee responses to each event within a defined amount of time. This type of operating system is mainly used by time-critical applications such as measurement and control systems. Some commonly used RTOS for embedded systems are: VxWorks, OS-9, Symbian, and RTLinux.

**2. Non-Real-time Operating Systems:**

Non-Real Time Operating Systems do not guarantee defined response times. These systems are mostly used if multiple applications are needed. Windows CE and PalmOS are examples for such embedded operating systems.

Why Linux?

There are a wide range of motivations for choosing Linux over a traditional embedded OS.

The following are the criteria due to which Linux is preferred:

**1. Quality and Reliability of Code:**

Quality and reliability are subjective measures of the level of confidence in the code that comprises software such as the kernel and the applications that are provided by distributions. Some properties that professional programmers expect from a “quality” code are modularity and structure, readability, extensibility and configurability. “Reliable” code should have features like predictability, error recovery and longevity. Most programmers agree that the Linux kernel and other projects used in a Linux system fit this description of quality and reliability. The reason is the open source development model, which invites many parties to contribute to projects, identify existing problems, debate possible solutions, and fix problems effectively.

**2. Availability of Code:**

Code availability relates to the fact that the Linux source code and all build tools are available without any access restrictions. The most important Linux components, including the kernel itself, are distributed under the GNU General Public License (GPL).Access to these components’ source code is therefore compulsory (at least to those users who have purchased any system running GPL-based software, and they have the right to redistribute once they obtain the source in any case). Code availability has implications for standardization and commoditization of components, too. Since it is possible to build Linux systems based entirely upon software for which source is available, there is a lot to be gained from adopting standardized embedded software platforms.

**3. Hardware Support:**

Broad hardware support means that Linux supports different types of hardware platforms and devices. Although a number of vendors still do not provide Linux drivers, considerable progress has been made and more is expected. Because a large number of drivers are maintained by the Linux community itself, you can confidently use hardware components without fear that the vendor may one day discontinue driver support for that product line. Linux also provides support for dozens of hardware architectures. No other OS provides this level of portability.

Typical architecture of an Embedded Linux System



Fig: 1.1 Typical Architecture Of An Embedded Linux System

**1. Hardware**

Linux normally requires at least a 32-bit CPU containing a memory management unit (MMU).A sufficient amount of RAM must be available to accommodate the system. Minimal I/O capabilities are required if any development is to be carried out on the target with reasonable debugging facilities. The kernel must be able to load a root file system through some form of permanent storage, or access it over a network.

**2. Linux Kernel**

Immediately above the hardware sits the kernel, the core component of the operating system. Its purpose is to manage the hardware in a coherent manner while providing familiar high-level abstractions to user-level software. It is expected that applications using the APIs provided by a kernel will be portable among the various architectures supported by this kernel with little or no changes. The low-level interfaces are specific to the hardware configuration on which the kernel runs and provide for the direct control of hardware resources using a hardware-independent API. Higher-level components provide the abstractions common to all UNIX systems, including processes, files, sockets, and signals. Since the low-level APIs provided by the kernel are common among different architectures, the code implementing the higher-level abstractions is almost constant, regardless of the underlying architecture. Between these two levels of abstraction, the kernel sometimes needs what could be called interpretation components to understand and interact with structured data coming from or going to certain devices. File system types and networking protocols are prime examples of sources of structured data the kernel needs to understand and interact with in order to provide access to data going to and coming from these sources.

**3. Applications and Libraries**

Applications do not directly operate above the kernel, but rely on libraries and special system daemons to provide familiar APIs and abstract services that interact with the kernel on the application’s behalf to obtain the desired functionality. The main library used by most Linux applications is the GNU C library, glibc. For Embedded Linux systems, substitutes to this library that are much less in size than glibc are preferred.

**1.3 REASONS TO SWITCH TO EMBEDDED LINUX**

Embedded Linux offers relief from an unstable and unfocused business model, and there are other strong reasons to consider using Linux:

* Complete source code availability makes it easier to fix items yourself, rather than being dependent on black boxes that are under someone else’s control.
* Development tools to support different processors are available as free downloads, or with a support offering that comes with a cost.
* There are many Linux distributions available with companies and organizations that support them, creating a lack of dependence on the whims of a single company.
* Many universities offer Linux courses, thus future engineers will be available with basic knowledge.
* Availability of different programming environments such as Java, Qt, and .NET also means a company can port existing Windows applications over the embedded Linux.
* BSP support is available for a variety of processors: ARM, x86, PowerPC, MIPS, etc. as well as driver support for many peripherals. Linux drivers are as readily available as Windows drivers.

**1.4 COMPARISON OF WINDOWS EMBEDDED & EMBEDDED LINUX**

A direct comparison between Windows Embedded and embedded Linux is not feasible. Embedded Linux is scalable from small devices without an MMU to large-scale server systems. Windows Embedded is not one product, but several products, each designed to address a specific market segment or device.

Windows Embedded Compact formally known as Windows CE) was developed in the 1990’s as a small ROM able Windows operating system for PDA and smart phones. Windows Embedded Compact has been replaced by Windows 8 for ARM as the current Windows Phone operating system.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ATTRIBUTE | WINDOWS CE 5.0 | WINDOWS EMBEDDED CE 6.0 | WINDOWS EMBEDDED COMPACT 7 | WINDOWS EMBEDDED COMPACT 2013 | EMBEDDED LINUX |
| PROCESSORS | X86, ARM, SH, MIPS | X86, ARM,  SH, MIPS | X86, ARM, MIPS | X86, ARM | X86, ARM, Power pc,MIPs |
| SUPPORT | Ye s | Ye s | Ye s | Ye s | Real-time support option available in the kernel |
| TOOLS | Platform Builder, Windows Embedded CE Test Kit (CETK) | Visual Studio with Platform Builder PlugIn, Windows Embedded CE Test Kit (CETK | Visual Studio with Platform Builder PlugIn, Compact Test Kit (CTK) | Visual Studio with Platform Builder PlugIn, Compact Test Kit (CTK) | Yocto Project |
| KERNEL PROGRAMMING LANGUAGES | C/C++ | C/C++ | C/C++ | C/C++ | C |
| HOST DEVELOPMENT SYSTEM | Windows | Windows | Windows | Windows | Windows, Linux |
| COST OF DEVELOPMENT SEAT | $995 | $995 | $995 | ~$1,000 | Free |
| LIFECYCLE SUPPORT END DATE | 1/14/2014 | 10/13/2020 | 2023 (estimated) | 2023 (estimated) | None |

Table1.1 COMPARISON OF WINDOWS EMBEDDED & EMBEDDED LINUX

**1.5 EMBEDDED LINUX OPTIONS**

The tasks for an embedded Linux project are as follows:

* Choosing the development tools to build the kernel – The cross compiler for the kernel might be different from the compiler for the applications, since the kernel cannot be linked to user space libraries.
* Choosing the development tools to build the user space applications.
* Building the boot loader.
* Configuring and porting the kernel.
* Creating the file system.

**CHAPTER 2**

**LITERATURE SURVEY**

With embedded systems fast expanding its reach, subject matter related to this field is available in abundance. While working on this project we have studied matter from various sources such as books, online articles and reference manuals. The knowledge gained from this activity has been of great help to us in understanding the basic concepts related to our project and has ignited further interest in this topic.

“Linux for Embedded and Real time Applications”, by Doug Abbott has been of great help in providing an introduction to the process of building embedded systems in Linux. It has helped us understand the process of configuring and building the Linux kernel and installing tool chains.

We understood the preponderance of the ARM processors in the field of embedded systems and the features of ARM processors from the document “The ARM Architecture” by Leonid Ryzhyk. The ARM architecture is a confluence of many useful features that makes it better than other peer processors. Being small in size and requiring less power, they prove useful in providing an efficient performance in embedded applications.

**CHAPTER 3**

**AIM AND OBJECTIVE OF PROJECT**

The objective of the project is to design **“**Embedded Web Server for Real-time Remote Control and Monitoring of an FPGA-based On-Board Computer System**”** The aim of this project is to get a fully functional single board computer (SBC) working with custom built monitoring software that communicates with all modern devices. It should be capable of extracting the necessary data from the control unit in order to use it in a meaningful and useful way. Communication to and from the CU will be done using the Onboard Diagnostics.

* **The objectives of this project are as follows:**
* To get the QT for Embedded Linux C++ framework successfully cross compiled for the ARM architecture so that it will run on the single board computer.
* To configure the embedded Linux distribution, Debian Etch that comes with the SBC in such a way that X11 GUI service will be removed and the bare minimum services started.

**Minimum Requirements**

The following are the minimum requirements for this project:

* Software requirements
* Linux OS (ubuntu-12.2)
* QT creator( 4.8 )
* Hardware requirements
* ARM/ZYNQ
* Cables
* Sensors
* Ethernet

**3.1 OBJECTIVE:**

In comparison with PC, the embedded system is greatly improved in stability, reliability and safety etc. The embedded system transplanted web server can be called embedded web server. Through web page posted by embedded web server, remote users can obtain the real-time status information and control remote equipment’s without time and space restriction. In this project the monitoring and controlling of devices is achieved using TCP/IP protocol and the Raspberry Pi. The TCP/IP protocol is the vital part, as it is used for the implementation of Embedded Web server.

This project consists of two parts, a server module which is connected to internet through RJ45 interface and a sensor module which always reads values from the sensors and is published on the internet. So the user can monitor and control the devices through Browser.

When the client types the correct IP address in the browser, the webpage displays the sensors values to the user and if there are any discrepancies in the values, then the user can control the device through a relay connected to the sensor module. Apart from controlling the devices remotely, automatic controlling procedure is also provided to control the devices.

**3.2 Block diagram:**

**3.2.1 TRANSMITTER:**

**REGLATED POWER SUPPLY**

**DEVICE1**

**DEVICE2**

**RELAY2**

**RELAY1**

**GAS SENSOR**

**MICROCONTROLLER**

**LDR SENSOR**

**TEMPERATURE SENSOR**

**ADC**

**ZIGBEE**

Fig: 3.1 Transmitter

**3.2.2 RECIEVER:**

**ZIGBEE**

**ARM (Xilinx FPGA)**

**TOUCH SCREEN DISPLAY UNIT**

**REGULATED POWER SUPPLY**

**INTERNET**

**SERVER**

HTTP

**TEMP**

**GAS**

**LDR**

**DEVICE1**

**ON**

**OFF**

**DEVICE2**

**AT REMOTE PC ON WEB PAGE**

**ON**

**OFF**

Fig: 3.2 Receiver

**CHAPTER 4**

**ARM ARCHITECTURE: AN OVERVIEW**

**4.1 Introduction**

ARM is a 32-bit RISC processor architecture developed by the ARM Corporation. ARM processors possess a unique combination of features that makes ARM the most popular embedded architecture today. First, ARM cores are very simple compared to most other general-purpose processors, which means that they can be manufactured using a comparatively small number of transistors, leaving plenty of space on the chip for application specific macro cells. A typical ARM chip can contain several peripheral controllers, a digital signal processor, and some amount of on-chip memory, along with an ARM core. Second, both ARM ISA and pipeline design are aimed at minimizing energy consumption — a critical requirement in mobile embedded systems. Third, the ARM architecture is highly modular: the only mandatory component of an ARM processor is the integer pipeline; all other components, including caches, MMU, floating point and other co-processors are optional, which gives a lot of flexibility in building application-specific ARM-based processors. Finally, while being small and low-power, ARM processors provide high performance for embedded applications. For example, the PXA255 XScale processor running at 400MHz provides performance comparable to Pentium 2 at 300MHz, while using fifty times less energy.

**4.2 ARM vs RISC**

In most respects, ARM is RISC architecture. Like all RISC architectures, the ARM ISA is a load-store one, that is, instructions that process data operate only on registers and are separate from instructions that access memory. All ARM instructions are 32-bit long and most of them have a regular three-operand encoding. Finally, the ARM architecture features a large register file with 16 general-purpose registers. All of the above features facilitate pipelining of the ARM architecture. However, the ARM architecture deviated from the RISC architecture in some respects to improve its performance. The ARM did not include register windows that were used by original RISC architectures to reduce complexity. The ARM architecture introduced an auto-indexing addressing mode, where the value of an index register is incremented or decremented while a load or store is in progress. ARM supports multiple register- transfer instructions that allow loading or storing up to 16 registers at once.

**4.3 Thumb instruction set extension**

The Thumb instruction set was introduced in the fourth version of the ARM architecture in order to achieve higher code density for embedded applications. Thumb provides a subset of the most commonly used 32-bit ARM instructions which have been compressed into 16-bit wide opcodes. On execution, these 16-bit instructions can be either decompressed to full 32- bit ARM instructions or executed directly using a dedicated Thumb decoding unit. Although Thumb code uses 40% more instructions than equivalent 32-bit ARM code, it typically requires 30% less space. Thumb code is 40% slower than ARM code; therefore Thumb is usually used only in non-performance-critical routines in order to reduce memory and power consumption of the system.

**4.4 Pipeline Design in ARM**

**

Fig. 4.1.Pipeline architecture in ARM7 and ARM9 cores

**4.4.1 The 3-stage pipeline**

It is a classical fetch-decode-execute pipeline, which, in the absence of pipeline hazards and memory accesses, completes one instruction per cycle. The first pipeline stage reads an instruction from memory and increments the value of the instruction address register, which stores the value of the next instruction to be fetched. This value is also stored in the PC register. The next stage decodes the instruction and prepares control signals required to execute it on. The third stage does all the actual work: it reads operands from the register file, performs ALU operations, reads or writes memory, if necessary, and finally writes back modified register values. In case the instruction being executed is a data processing instruction, the result generated by the ALU is written directly to the register file and the execution stage completes in one cycle. If it is a load or store instruction, the memory address computed by the ALU is placed on the address bus and the actual memory access is performed during the second cycle of the execute stage. This pipeline remained unchanged from the first ARM processor to the ARM7TDMI core.

**4.4.2 The 5 stage pipeline**

The 3-stage pipeline has the problem of pipeline stall when a memory read or writes operation is going on, and the next instruction is to be fetched. The solution to this problem was to use a separate instruction and data cache. First, to make the pipeline more balanced, ARM9TDMI moved the register read step to the decode stage, since instruction decode stage was much shorter than the execute stage. Second, the execute stage was split into 3 stages. The first stage performs arithmetic computations, the second stage performs memory accesses (this stage remains idle when executing data processing instructions) and the third stage writes the results back to the register file. This results in a much better balanced pipeline, which can run at faster clock rate, but there is one new complication — the need to forward data among pipeline stages to resolve data dependencies between stages without stalling the pipeline. The ARM10 and ARM11 came up with the 6-stage and the 8-stage pipeline.

**4.5 Differences**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Features | 8051 | ARM7 | ARM9 | ARM11 | ARM Cortex |
| Architecture | CISC | RISC-Vonneumann | RISC-Harvard | RISC-Harvard | RISC |
| Memory(ROM) | 4KB | 512Kb | up to 2MB | Up to 1gb | 1gb and above |
| Memory(RAM) | 128 bytes | 40Kb | 64MB | 512MB | 512MB |
| Frequency | Up to 24 MHZ | Up to 60MHz | 400 MHz | 700Mhz | Up to 1GHZ |
| OS | Windows | Windows | Wince,linux,android | Windows, Linux | Windows,Linux |
| Language | Embedded c, C/C++ | Embedded c, c/c++ | C/C++ | C/C++ | C/C++ |
| Timers | 2 | 2 | 3 | 8 | 8 |
| RTC | No | yes | Yes | Yes | YES |
| UART | 1 | 2 | 3 | 4 | Up to 6 |
| USB | No | yes | YES | YES | yes |
| CAMERA | No | No | YES | YES | yes |
| ETHERNET | No | No | YES | YES | yes |
| PROTOCOLS | No | I2C,SPI | SPI,I2C,RDP,HTTP,FTP | SPI,I2C,RDP,HTTP,FTP | SPI,I2C, |
| TYPE | CONTROLLER | PROCESSOR | PROCESSOR | PROCESSOR | PROCESSOR |
| GPIO PINS | 32 | 65 | 34 | 26 |  |
| DATABUS | 8 bit | 32 bit | 32 bit | 32 bit |  |
| PIPELINE | No | 3stage | 5 stage | 8 stage | 13 stage |
| ADC | NO | YES | YES | NO | YES |

Table 4.1 Differences

**CHAPTER 5**

**HARDWARE USED**

**5.1 ZYBO(ZYnq Board)**

**Overview**

The ZYBO (ZYnq BOard) is a feature-rich, ready-to-use, entry-level embedded software and digital circuit development platform built around the smallest member of the Xilinx Zynq-7000 family, the Z-7010. The Z-7010 is based on the Xilinx All Programmable System-on-Chip (AP SoC) architecture, which tightly integrates a dual-core ARM Cortex-A9 processor with Xilinx 7-series Field Programmable Gate Array (FPGA) logic. When coupled with the rich set of multimedia and connectivity peripherals available on the ZYBO, the Zynq Z-7010 can host a whole system design. The on-board memories, video and audio I/O, dual-role USB, Ethernet, and SD slot will have your design up-and-ready with no additional hardware needed. Additionally, six Pmod connectors are available to put any design on an easy growth path.

The Zynq 7010 AP SoC offers the following features:

* 650Mhz dual-core Cortex-A9 processor
* DDR3 memory controller with 8 DMA channels
* High-bandwidth peripheral controllers: 1G Ethernet, USB 2.0, SDIO
* Low-bandwidth peripheral controller: SPI, UART, CAN, I2C
* Reprogrammable logic equivalent to Artix-7 FPGA
  + 4,400 logic slices, each with four 6-input LUTs and 8 flip-flops o 240 KB of fast block RAM
* Two clock management tiles, each with a phase-locked loop (PLL) and mixed-mode clock manager (MMCM)
  + 80 DSP slices
  + Internal clock speeds exceeding 450MHz o On-chip analog-to digital converter (XADC)
* ZYNQ XC7Z010-1CLG400C
  + 512MB x32 DDR3 w/ 1050Mbps bandwidth
  + Dual-role (Source/Sink) HDMI port
* 16-bits per pixel VGA source port
  + Trimode (1Gbit/100Mbit/10Mbit) Ethernet PHY
  + MicroSD slot (supports Linux file system)
* OTG USB 2.0 PHY (supports host and device)
* External EEPROM (programmed with 48-bit globally unique EUI-48/64™ compatible identifier)
* Audio codec with headphone out, microphone and line in jacks
* 128Mb Serial Flash w/ QSPI interface
* On-board JTAG programming and UART to USB converter
* GPIO: 6 pushbuttons, 4 slide switches, 5 LEDs

The ZYBO is compatible with Xilinx’s new high-performance Vivado Design Suite as well as the ISE/EDK toolset. These toolsets meld FPGA logic design with embedded ARM software development into an easy to use, intuitive design flow. They can be used for designing systems of any complexity, from a complete operating system running multiple server applications in tandem, down to a simple bare-metal program that controls some LEDs.

An accessory kit that contains a 5V/2.5A power adapter, a USB A to Micro B cable, an 8GB speed class 10 microSD card, and a Xilinx voucher for a Vivado Design Suite license will be available to purchase separately off of the Digilent website in the near future. The Vivado Design Suite license enables use of the Logic Analyzer tools, which accelerate debug and testing. This license is a 1 year, device-locked license, and will only work with the Z-7010 Zynq part on the ZYBO. After the license expires, any version of Vivado Design Suite that was released during this 1 year period can continue to be used indefinitely.

**5.1.1 Power Supplies**

The ZYBO can be powered from the Digilent USB-JTAG-UART port (J11), or from an external power supply. Jumper JP7 (near the power switch) determines which power source is used. There are three valid configurations for this jumper corresponding to the three powering options: USB, wall wart with barrel jack, and battery pack. There is a diagram on the board silkscreen indicating all three.

All on-board power supplies are enabled or disabled by the power switch SW4. The power indicator LED (LD11) is on when all the supply rails reach their nominal voltage. An overview of the power circuit is shown in Fig.5.1

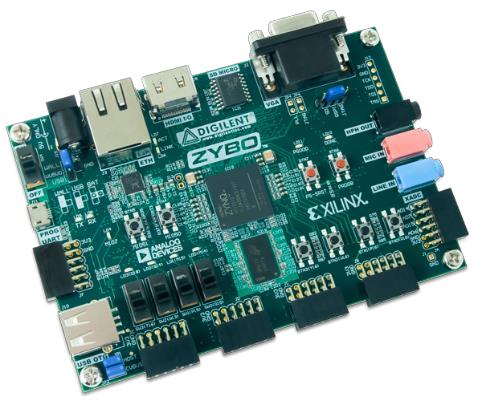


Fig 5.1:ZYBO(ZYnq

A USB 2.0 port can deliver maximum 0.5A of current according to the specifications. This should provide enough power for lower complexity designs. An idling blank board consumes around 0.2A from the 5V input supply. As an example, the standalone lwIP echo server sample project replying to ping requests consumes 0.38A on average. More demanding applications, including any that drive multiple peripheral boards or other USB devices, might require more power than the USB port can provide. In this case, power consumption will increase until it’s limited by the USB host. This limit varies a lot between manufacturers and depends on many factors. When in current limit, once the voltage rails dip below their nominal value, the Zynq is reset by the Power-on Reset signal and power consumption returns to its idle value. Also, some applications may need to run without being connected to a PC’s USB port. In these instances an external power supply or battery pack can be used.

An external power supply (wall wart) can be used by plugging into to the power jack (J15) and setting jumper JP7 to “wall”. The supply must use a coax, center-positive 2.1mm internal-diameter plug, and deliver 4.5VDC to 5.5VDC and at least 2.5A of current (i.e., at least 12.5W of power). Suitable supplies can be purchased from the Digilent website or through catalog vendors like DigiKey. Power supply voltages above 6VDC might cause permanent damage.

An external battery pack can be used by connecting the battery’s positive terminal to the center pin of JP7 and the negative terminal to the pin labeled J14 next to JP7. The external battery pack must be limited to 5.5VDC. The

minimum voltage of the battery pack depends on the application: if the USB Host (J10) or HDMI Source (J8) function is used, at least 4.6V need to be provided. In other cases the minimum voltage is 3.6V.

Voltage regulator circuits from Analog Devices create the required 3.3V, 1.8V, 1.5V, and 1.0V supplies from the main power input. Table 2 provides additional information (typical currents depend strongly on FPGA configuration and the values provided are typical of medium size/speed designs).

The supply rails are daisy-chained to follow the Xilinx-recommended start-up sequence. Flicking the power switch (SW4) will enable the 1.0V rail, which enables the 1.8V digital supply rail, which in turn enables the I/O supply rails 3.3V and 1.5V. The 1.25V reference and 1.8V analog supply ramp together with the 3.3V rail. Once all the channels of the ADP5052 (IC26) supply reach regulation, the PGOOD signal will assert, enabling the 3.3V audio supply, lighting up the power LED (LD11), and de-asserting the Power-On Reset signal (PS\_POR\_B) of the Zynq.

Each power supply uses a soft-start ramp of 1-10ms to limit in-rush current. There is an additional delay of at least 130ms after the power rails reach regulation and before the Power-On Reset signal de-assert to allow for the PS\_CLK (IC22) to stabilize.

**5.1.2 Zynq AP SoC Architecture**

The Zynq AP SoC is divided into two distinct subsystems: The Processing System (PS), and the Programmable Logic (PL). Figure 3 shows an overview of the Zynq AP SoC architecture, with the PS colored light green and the PL in yellow. Note that the PCIe Gen2 controller and Multi-gigabit transceivers are not available on the Zynq-7010 device.

The PL is nearly identical to a Xilinx 7-series Artix FPGA, except that it contains several dedicated ports and buses that tightly couple it to the PS. The PL also does not contain the same configuration hardware as a typical 7-series FPGA, and it must be configured either directly by the processor or via the JTAG port.

The PS consists of many components, including the Application Processing Unit (APU, which includes 2 Cortex-A9 processors), Advanced Microcontroller Bus Architecture (AMBA) Interconnect, DDR3 Memory controller, and various peripheral controllers with their inputs and outputs multiplexed to 54 dedicated pins (called MultiplexedI/O, or MIO pins). Peripheral controllers that do not have their inputs and outputs connected to MIO pins can instead route their I/O through the PL, via the Extended-MIO (EMIO) interface. The peripheral controllers are connected to the processors as slaves via the AMBA interconnect, and contain readable/writable control registers that are addressable in the processors’ memory space. The programmable logic is also connected to the interconnect as a slave, and designs can implement multiple cores in the FPGA fabric that each also contain addressable control registers. Furthermore, cores implemented in the PL can trigger interrupts to the processors (connections not shown in Fig. 3) and perform DMA accesses to DDR3 memory

There are many aspects of the Zynq AP SoC architecture that are beyond the scope of this document. For a complete and thorough description, refer to the Zynq Technical Reference Manual, available at www.xilinx.com.

Figure 5.3 depicts the external components connected to the MIO pins of the ZYBO. The Zynq Board Definition File found on the Digilent ZYBO product page can be imported into EDK and Vivado Designs to properly configure the PS to work with these peripherals.

Two on-board status LEDs provide visual feedback on traffic flowing through the port: the transmit LED (LD9) and the receive LED (LD8). Signal names that imply direction are from the point-of-view of the DTE (Data Terminal Equipment), in this case the PC.

The FT2232HQ is also used as the controller for the Digilent USB-JTAG circuitry, but the USB-UART and USB-JTAG functions behave entirely independent of one another. Programmers interested in using the UART functionality of the FT2232 within their design do not need to worry about the JTAG circuitry interfering with the UART data transfers, and vice-versa. The combination of these two features into a single device allows the ZYBO to be programmed, communicated with via UART, and powered from a computer attached with a single Micro USB cable.

**3 Clock Sources**

The ZYBO provides a 50 MHz clock to the Zynq PS\_CLK input, which is used to generate the clocks for each of the PS subsystems. The 50 MHz input allows the processor to operate at a maximum frequency of 650 MHz and the DDR3 memory controller to operate at a maximum of 525 MHz (1050 Mbps). The ZYBO Base System Design configures the PS to work properly with this input clock, and should be used as a reference when creating custom designs.

The PS has a dedicated PLL capable of generating up to four reference clocks, each with settable frequencies, that can be used to clock custom logic implemented in the PL. Additionally, The ZYBO provides an external 125 MHz reference clock directly to pin L16 of the PL. The external reference clock allows the PL to be used completely independently of the PS, which can be useful for simple applications that do not require the processor.

The PL of the Zynq-Z7010 also includes two MMCM’s and two PLL’s that can be used to generate clocks with precise frequencies and phase relationships. Any of the four PS reference clocks or the 125 MHz external reference clock can be used as an input to the MMCMs and PLLs. For a full description of the capabilities of the Zynq PL clocking resources, refer to the “7 Series FPGAs Clocking Resources User Guide” available from Xilinx.

Figure 13 outlines the clocking scheme used on the ZYBO. Note that the reference clock output from the Ethernet PHY is used as the 125 MHz reference clock to the PL, in order to cut the cost of including a dedicated oscillator for this purpose. Keep in mind that CLK125 will be disabled when the Ethernet PHY (IC1) is held in hardware reset by driving the PHYRSTB signal low.

**Basic I/O**

The ZYBO board includes four slide switches, four push buttons, and four individual LEDs connected to the Zynq PL, as shown in Fig. 14. There are also two additional pushbuttons and one LED connected directly to the PS via MIO pins, also shown in Fig. 14. The pushbuttons and slide switches are connected to the Zynq via series resistors to prevent damage from inadvertent short circuits (a short circuit could occur if a pin assigned to a pushbutton or slide switch was inadvertently defined as an output). The pushbuttons are "momentary" switches that normally generate a low output when they are at rest, and a high output only when they are pressed. Slide switches generate constant high or low inputs depending on their position.

The high-efficiency LEDs are anode-connected to the Zynq via 330-ohm resistors, so they will turn on when a logic high voltage is applied to their respective I/O pin. Additional LEDs that are not user-accessible indicate power-on, FPGA programming status, and USB and Ethernet port status.

The LED and two pushbuttons attached directly to the PS are accessed using the Zynq GPIO controller. This core is described in full in Chapter 14 of the Zynq Technical Reference Manual.

**Reset Sources**

**Power-on Reset**

The Zynq PS supports external power-on reset signals. The power-on reset is the master reset of the entire chip. This signal resets every register in the device capable of being reset. The ZYBO drives this signal from a comparator that holds the system in reset until all power supplies are valid. Several other IC’s on the ZYBO are reset by this signal as well.

**Program Push Button Switch**

A PROG push switch, BTN6, toggles Zynq PROG\_B. This resets the PL and causes DONE to be de-asserted. The PL will remain unconfigured until it is reprogrammed by the processor or via JTAG.

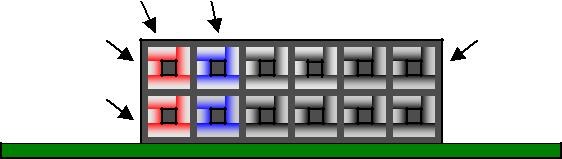
**Processor Subsystem Reset**

The external system reset, labeled PS\_SRST/BTN7, resets the Zynq device without disturbing the debug environment. For example, the previous break points set by the user remain valid after system reset. Due to security concerns, system reset erases all memory content within the PS, including the OCM. The PL is also cleared during a system reset. System reset does not cause the boot mode strapping pins to be re-sampled.

**Pmod Connectors**

Pmod connectors are 2x6, right-angle, 100-mil spaced female connectors that mate with standard 2x6 pin headers. Each 12-pin Pmod connector provides two 3.3V VCC signals (pins 6 and 12), two Ground signals (pins 5 and 11), and eight logic signals, as shown in Fig. 16. The VCC and Ground pins can deliver up to 1A of current, but care must be taken not to exceed any of the power budgets of the onboard regulators or the external power supply.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| VCC GND |  | 8 signals | | |  |
| Pin 6 |  |  |  | Pin 1 |  |
|  |  |  |  |
|  |  |  |



Pin 12

Figure 5.3: Pmod diagram.

Digilent produces a large collection of Pmod accessory boards that can attach to the Pmod expansion connectors to add ready-made functions like A/D’s, D/A’s, motor drivers, sensors, and other functions. See www.digilentinc.com for more information.

The ZYBO has six Pmod connectors, some of which behave differently than others. Each Pmod connector falls into one of four categories: standard, MIO connected, XADC, or high-speed Table 9 specifies which category each Pmod falls into, and also lists the Zynq pins they are connected to. The following sections describe the different types of Pmods.

**Dual Analog/Digital Pmod (XADC Pmod)**

The on-board Pmod expansion connector labeled “JA” is wired to the auxiliary analog input pins of the PL. Depending on the configuration, this connector can be used to input differential analog signals to the analog-to-digital converter inside the Zynq (XADC). Any or all pairs in the connector can be configured either as analog input or digital input-output.

In analog input mode, the voltage on these pins must be limited to 1V peak-to-peak. In digital mode, the regular VCCO-dependent limits apply. See Xilinx datasheets for more information.

The Dual Analog/Digital Pmod on the ZYBO differs from the rest in the routing of its traces. The eight data signals are grouped into four pairs, with the pairs routed closely coupled for better analog noise immunity. Pins 1 and 7, pins 2 and 8, pins 3 and 9, and pins 4 and 10 are paired up. Furthermore, each pair has a partially loaded anti-alias filter laid out on the PCB. The filter does not have capacitors C94-C97. In designs where such filters are desired, the capacitors can be manually loaded by the user. NOTE: The coupled routing and the anti-alias filters might limit the data speeds when used for digital signals.

The XADC core within the Zynq is a dual channel 12-bit analog-to-digital converter capable of operating at 1 MSPS. Either channel can be driven by any of the auxiliary analog input pairs connected to the JXADC header. The XADC core is controlled and accessed from the PL via the Dynamic Reconfiguration Port (DRP). The DRP also provides access to voltage monitors that are present on each of the FPGA’s power rails, and a temperature sensor that is internal to the FPGA. For more information on using the XADC core, refer to the Xilinx document titled “7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter.” It is also possible to access the XADC core directly using the PS, via the “PS-XADC” interface. This interface is described in full in chapter 30 of the Zynq Technical Reference Manual.

**High-Speed Pmod**

The High-speed Pmods use the standard Pmod connector, but have their data signals routed as impedance matched differential pairs for maximum switching speeds. They have pads for loading resistors for added protection, but the ZYBO ships with these loaded as 0-Ohm shunts. With the series resistors shunted, these Pmods offer no protection against short circuits, but allow for much faster switching speeds. The signals are paired to the adjacent signals in the same row: pins 1 and 2, pins 3 and 4, pins 7 and 8, and pins 9 and 10. Traces are routed 100 ohm (+/- 10%) differential.

These connectors should be used only when high speed differential signaling is required or the other Pmods are all occupied. If used as single-ended, coupled pairs will have significant crosstalk. In applications where this is a concern, the standard Pmod connector shall be used. Another option would be to ground one of the signals (drive it low from the FPGA) and use its pair for the signal-ended signal.

Since the High-Speed Pmods have 0-ohm shunts instead of protection resistors, the operator must take precaution to ensure that they do not cause any shorts.

**5.2** **REGULATED POWER SUPPLY**

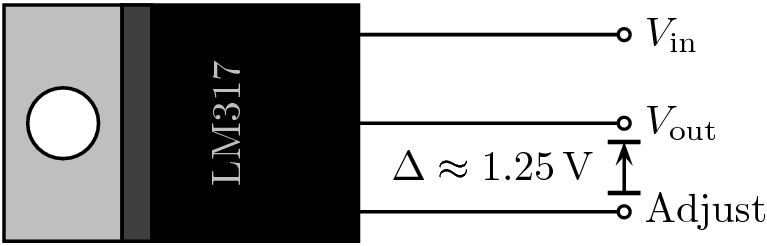
A variable regulated power supply, also called a variable bench power supply, is one where you can continuously adjust the output voltage to your requirements. Varying the output of the power supply is the recommended way to test a project after having double checked parts placement against circuit drawings and the parts placement guide.

This type of regulation is ideal for having a simple variable bench power supply. Actually this is quite important because one of the first projects a hobbyist should undertake is the construction of a variable regulated power supply. While a dedicated supply is quite handy, it’s much handier to have a variable supply on hand, especially for testing. Mainly the ARM controller needs 3.3 volt power supply. To use these parts we need to build a regulated 3.3 volt source. Usually you start with an unregulated power To make a 3.3 volt power supply, we use a LM317 voltage regulator IC (Integrated Circuit). The IC is shown below.

**CIRCUIT FEATURES**

|  |  |
| --- | --- |
| Vout range | 1.25V - 37V |
| Vin - Vout difference | 3V - 40V |
| Operation ambient temperature | 0 - 125°C |
| Output Imax | <1.5A |
| Minimum Load Currentmax | 10mA |

A current-limiting circuit constructed with LM317



Part pinout of LM317 showing its constant voltage reference

**LM317** is the standard part number for an integrated three-terminal adjustable linear voltage regulator. LM317 is a positive voltage regulator supporting input voltage of 3V to 40V and output voltage between 1.25V and 37V. A typical current rating is 1.5A although several lower and higher current models are available. Variable output voltage is achieved by using a potentiometer or a variable voltage from another source to apply a control voltage to the control terminal. LM317 also has a built-in current limiter to prevent the output current from exceeding the rated current, and LM317 will automatically reduce its output current if an overheat condition occurs under load. LM317 is manufactured by many companies, including National Semiconductor, Fairchild Semiconductor, and STMicroelectronics.

Although LM317 is an adjustable regulator, it is sometimes preferred for high-precision fixed voltage applications instead of the similar LM78xx devices because the LM317 is designed with superior output tolerances. For a fixed voltage application, the control pin will typically be biased with a fixed resistor network, a Zener diode network, or a fixed control voltage from another source. Manufacturer datasheets provide standard configurations for achieving various design applications, including the use of a pass transistor to achieve regulated output currents in excess of what the LM317 alone can provide.

LM317 is available in a wide range of package forms for different applications including heat sink mounting and surface-mount applications. Common form factors for high-current applications include TO-220 and TO-3. LM317 is capable of dissipating a large amount of heat at medium to high current loads and the use of a heat sink is recommended to maximize the lifespan and power-handling capability.

LM337 is the negative voltage complement to LM317 and the specifications and function are essentially identical, except that the regulator must receive a control voltage and act on an input voltage that are below the ground reference point instead of above it.

**BLOCK DIAGRAM**

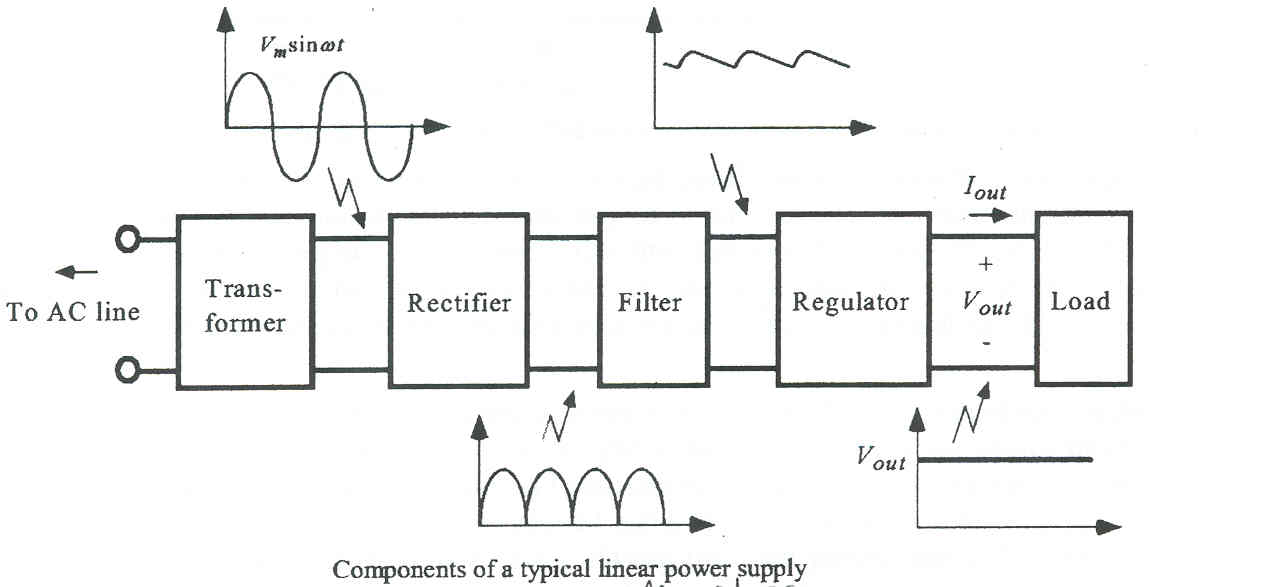


Fig Block Diagram

WE CAN EVEN USE A USB CONNECTOR FOR THE REQUIRED SUPPLY INSTEAD OF THE ABOVE CIRCUIT

**5.3. HDMI CABLE**

**5.3.1DESCRIPTION**

Tyco Electronics’ connector incorporating High-Definition Multimedia Interface (HDMI™) technology is an interface for consumer electronics products such as digital television, set-top boxes and DVD players.HDMI supports standard, enhanced, or high-definition video, plus multi-channel audio on a single cable.

****