Internet of Things Platform on ARM/FPGA Using Embedded Linux

Ahmad Safwan Haron, Mohamad Sofian Abu Talip*,
Anis Salwa Mohd Khairuddin, Tengku Faiz Tengku Mohmed Noor Izam
Department of Electrical Engineering, Faculty of Engineering
University of Malaya
50603 Kuala Lumpur, Malaysia
*sofian_abutalip@um.edu.my

***Abstract*— As the demand for smarter gadget in small form factor and the prediction of Internet of Things as the next big thing, the role of embedded systems in our daily lives become more prominent. In embedded system, Yocto project has started to become synonymous with embedded Linux due to its multi architecture support. As for the hardware, the flexibility and low cost of Field Programmable Gate Array (FPGA) make it as an attractive choice for implementation of embedded Linux. This project is to show how the flexibility of the FPGA can be used as hardware for booting up an embedded operating system. The chosen board, Altera DE1-SoC has an onboard hard ARM Cortex-A9, which is needed to drive the Yocto Linux. The Yocto Linux is stored in Secure Digital (SD) card and can be accessed via the onboard SD card reader. Finally, the FPGA-based system is able to access the hardware on the Linux-system using device drivers for the common parallel I/O hardware and user applications to read/write or from the devices.**

Keywords; Reconfigurable System; System on Chip; FPGA; ARM; Yocto Linux

I. INTRODUCTION

Embedded system is a combination of logical circuitry and software that is built into a product for specific purposes such as managing various input and output without human intervention. It is the crucial core of every modern electronic product, ranging to simple children toys to complex aircraft control systems [1]. Typically, embedded system is designed to perform a narrow range of pre-defined tasks. Thus, they do not have any of the typical general-purpose peripheral devices or any kind of user interface software, unless it is designed to do so. This leads to reduction in terms of complexity, size and cost and increase in the robustness of the embedded systems as compared to it general-purpose counterpart. Some embedded system includes an operating system, which is referred to as an embedded operating system.

Due to the pressure to minimize cost for embedded systems in order to minimize the prices of the final products, free software is chosen as its operating systems. Such operating systems also can be freely modifying according the requirements of the specific applications. Linux is the most widely used embedded operating system worldwide. According to the developers of proprietary embedded systems, the downsides of using Linux are its insufficient real time capabilities, large size, lack of good development tools and possible licensing issues [6]. However, these disadvantages are not important and they are diminishing, due to improvements in Linux's real time capabilities with newer kernels, small size, improved development tools and licensing simplicity [8].

Moreover, Yocto Project is distribution environment and focus on improving the software development process for embedded Linux distribution. It has the aim and objective of attempting to improve the lives of developers of customized Linux systems supporting the ARM, MIPS, PowerPC and x86 (32 & 64 bit) architecture. A key part of this is an open source build system with strong community, based on the OpenEmbedded architecture, enables developers to create their own Linux distribution to their specific environment.

The rest of this paper are organized as follows. Section II presented the related work of this project. Section III gives an overview of the design and implementation. Section IV discussed about the results and analysis as well as the performance evaluation. Finally, Section V summarized this work with a conclusion.

II. RELATED WORK

Embedded Linux on FPGA has attracted many researchers in this field. According to Jasinski *et al.* [2], they proposed a case study on the impact of system adoption in an embedded system. The study was done under three different scenarios, using the μC/OS-II real-time OS, and using the μClinux general-purpose OS. A standard development kit containing Altera Cyclone II FPGA was used to provide a flexible hardware platform to accommodate all three configurations. The paper hypothesizes that the adoption of an OS reduced the development time by 48%. However, the tradeoff is the increased in program memory requirements at least by 71%.

Embedded system, mostly on FPGA has to be more secure. Its mobile characteristic has make it exposed to hostile environment. Due to large storage required by embedded OS, its kernel usually stored in an external memory. This off-chip memory allowed an attacker to temper the kernel and installing malicious code. The FPGA embedded processor is capable of updating the OS, which is usually through an insecure network. However, the usage of the unsecured

978-1-5386-0607-0/17 $31.00 © 2017 IEEE
DOI 10.1109/ACOMP.2017.26

network leads to security flaws in terms of the system integrity or freshness. In a proposed work related to secure booting, Devic *et al.* [3] proposed the solution by introducing a trusted computing mechanism that consists of the whole security chain from bitstream-to-kernel-boot. This is to ensure that both hardware and software remains their integrity while preventing replay attacks. The proposed security mechanism has its performance improved due the FPGA's hardware acceleration capability.

In another work by Santini *et al.* [4], studied the reliability analysis of OS for Embedded SoC by measuring the difference in the neutron-induced error rate when executing the application bare to the metal and on top of the Linux Kernel. The error that may occurred are the differences in the produced results compared to the expected one, which is called Silent Data Corruption (SDC) and Single Event Functional Interruption (SEFI), where it system either hangs, crashes, or has to be reboot in order the return to the normal operation.

In [5], the work explained how Microblaze, a 32-bit soft-core processor by Xilinx is used in a series of design projects. A commercially available FPGA based single board computer is configured with the Microblaze. The pre-configured FPGA is then configured to run uCLinux, a forked version of Linux for embedded systems. By utilizing this platform, they are able to build custom hardware that can utilize the implemented Microblaze processor via the On-chip Peripheral Bus (OPB), and custom software that runs as thread on the operating system. The implementation of Xilinx's Microblaze processor and the uCLinux operating system has tested to be successful to date. The Microblaze processor with its 50 MHz clock frequency are sufficient to provide the needed processing power and a significant amount of user-logic can be provided by the Spartan-3 FPGA. As for the uCLinux, it is a good platform for software development and debugging. However, there are several cons on using this method, especially from educational point of view such as lack of simulation capability, lack of hardware observability, higher software complexity and steeper learning curve.

Then, the work by Cvek *et al.* [7] describes the design of a network monitoring system running on FPGA-based Linux that uses the SNMPv2 protocol to manage the devices that connected to the network. The SNMP agent are executed just like a normal Linux applications and all the related information in the database are managed by SQLite manager. However, it is really important to remember that the tradeoff for a more flexible system is its performance will be reduced.

Klenke [9] developed a full boot-to-runtime protection flow of an embedded Linux kernel during boot and confidentiality/integrity protection of the external memory containing the kernel and the main application code/data. This is done using the hardware components that induced 10% occupancy area of Virtex-6 FPGA while improving the throughout for Linux booting and low latency security for runtime protection.

According to Monson *et al.* [11] describe that FPGA-based Linux test-bed was constructed for the purpose of measuring its sensitivity to single-event upsets. The test-bed consists of two ML410 Xilinx development boards connected using a 124-pin custom connector board. The Design Under Test (DUT) consists of the "hard core" PowerPC, running the Linux OS and several peripherals implemented in "soft" (programmable) logic. Faults were injected via the Internal Configuration Access Port (ICAP). The experiments performed demonstrate that the Linux-based system was sensitive to 92,542 upsets-less than 0.7 percent of all tested bits.

In this work we booted Yocto Linux in Altera DE1-SoC board. We design and implemented Linux utilities, kernel configuration and compilation. We also set the configuration of the interaction between the FPGA and the onboard ARM Cortex A9 processor.

III. DESIGN AND IMPLEMENTATION

In this project, Altera DE1-SoC Development Kit is chosen as the hardware for the implementation of the embedded Linux. The DE1-SoC presents a robust hardware design platform built around the Altera SoC FPGA, which combines the latest dual-core Cortex embedded cores with industry-leading programmable logic. The ARM-based hard processor system (HPS) consists of processor, peripheral, and memory interfaces tied with the FPGA fabric using high-bandwidth interconnect backbone. The FPGA available on the board is Cyclone V SoC, Altera's latest processor from Cyclone series. The board has 85000 programmable logic elements and 4450 kilobits of embedded memory. The six fractional phase locked loops (PLLs) are used to divide, multiply, and shift the phase of clock signals.

A. Building the System Using Qsys

In this project, the successor for SOPC Builder, Qsys is used to configure the connection between the FPGA, the HPS and the needed peripheral. The Qsys system integration tools in Quartus Prime software saves time and effort in the FPGA design process by enabling faster system development and design reuse. It automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems. By using Qsys, the time consuming task of writing HDL code to specify system-level connections can be minimized. Furthermore, bus functional model (BFMs), monitors and other IP verification can be used for design's verification.

B. Configuring the Hard Processor System (HPS)

First and foremost, the HPS has to be configured so that it suits the needs for this project. The HPS are important since its will handle all the processing power required by Yocto. The FPGA's clock, CLK are connected to the HPS-to-FPGA (H2F), FPGA-to-HPS (F2H) and H2F-Lightweight Bridges. The H2F-Bridge allows the master in the HPS to communicate with slaves in the FPGA as shown in Figure 1.

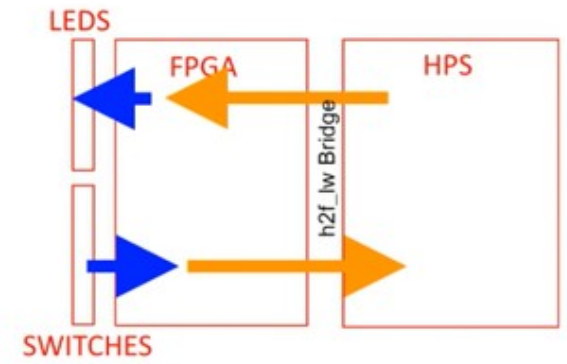

Figure 1 : The communication between FPGA and HPS.

Since the HPS will read input and give output from and to the peripherals, they are connected to the FPGA via H2F-LW. The read/write operation will happen here. The H2F-LW bridge provides a lower-performance interface to the FPGA fabric. This is useful for accessing the control and status register of the soft peripheral.

C. Adding Parallel I/O Port

To use the soft peripherals as an input or output, the processor requires a certain amount of general-purpose I/O (GPIO) pins. These pins are connected to the FPGA and HPS. The GPIO component has a number of option for customizing GPIO interfaces. It can be set as input only, output only, or bidirectional. In bidirectional mode, the direction of each pin must be set in the direction register at run-time via software. In input only mode, the interface has various interrupt and edge capture capabilities including the capturing of either or both edge and edge or level sensitive interrupt triggers. The input only GPIO component is configured to capture edges on its ports. It can capture low-to-high transitions, high-to-low transitions, or both. Whenever an edge is detected, the condition is indicated in the edge capture register.

D. Configuring the Peripherals Pin

The controller for various peripheral for Yocto are configured in the HPS setting, under peripheral pins. The peripheral needed for this project are Ethernet, SD card reader, Universal Serial Bus (USB) and UART.

Ethernet, which usually used for Internet connection is used for transferring the program for Yocto from the computer to the FPGA. The controller for the Ethernet media access is set to HPS I/O Set 0 with the mode of reduced gigabit media-independent interface (RGMII). Since the kernel of the Linux is stored in external non-volatile memory, the SD card reader controller has to be enabled. The pin setting is HPS I/O Set 0 and the mode is 4-bit data. The USB is used to program the FPGA using Quartus Programmer via the USB type B port on the board. Its controller is set to HPS I/O Set 0 with the mode of SDR with PHY clock output mode. The last peripheral is the UART, which is needed in order to use Yocto on the computer via the onboard mini USB port. The setting is HPS I/O Set 0 with no flow control mode. From this setting interface, we were able to see the Peripheral Mux table, which is used to check for pins with invalid multiple assignments in order to avoid pin multiplexing conflicts.

E. Configuring the SDRAM

The SDRAM controller behave likes simple memory. It is connecting to one or more SDRAM chips, and handles all SDRAM protocol requirements. Internal to the device, the core presents an Avalon-MM slave port that appears as linear memory to Avalon-MM master peripherals. The core can access SDRAM subsystems with various data widths (8, 16, 32, or 64 bits), various memory sizes, and multiple chip selects. The SDRAM controller must be configured for the timing requirements of the specific SDRAM brand and model being used.

In order to use the HPS properly, its SDRAM controller has to be configured. The SDRAM controller, which provides data access and run-time programmability, used to arrange the data to reduce row conflicts and bus turn-around time. The data is read by group and the transactions is written together, thus leads to more efficient traffic patterns and latency reduction. The controller made up of a multiport front end (MPFE) and a single-port controller. Multiple independent interfaces to the single-port controller is provided by the MPFE while the single-port controller is used to communicate and manage each external memory device.

F. Integrating Qsys Design with Quartus Prime

To integrate a Qsys system into a Quartus Prime project, Quartus Prime IP file, *.qip* or *.qsys* is needed:

a. *.qip* is generated by Qsys. This file is used to integrate Qsys design with Quartus Prime project where full control is preferable

b. *.qsys* is a system file saved by Qsys. When there is no customization or scripts in the design flow, this method is more convenient to be implemented

To ensure a consistent integration flow throughout the development, only one type of IP file is needed. If both file is used, two sets of output files will be generated for the design. Thus, an error will have occurred during compilation of the project files due to the presence of two different output files.

Qsys can be used to generate Verilog or VHDL files for all of the modules in the system, instantiation template for the system, block symbol files, Synopsis Design Constraints file and Peripheral Template File. For this project, it was used to generate the instantiation template needed for the FPGA design in the Quartus Prime. This instantiation template contains all the component of the HPS and all the soft peripheral used in this project. Then, the instantiation part component will be connected to the nodes of the FPGA in order to establish communication between the FPGA, HPS and the soft peripheral.

G. The HDL Coding for the FPGA Design

For the FPGA design, the coding is done using VHDL. The function of the program is to manage all the signals and communication from various components of the board needed to run Yocto. The components that are used for this project are the clock, switches, LED, the HPS SDRAM, Ethernet port, SD card reader, UART and USB terminal. In the Entity of the program, all the nodes of the FPGA need to manage are initialized. Here all the nodes are defined as input, output or bidirectional. Their respective size bits are specified too. In the Architecture part, all of the components that has to be connected to FPGA are listed. Then the components are connected to the nodes of FPGA in port map by matching the node of FPGA to its respectively components. Both of the list of the component and instantiation part are generated by Qsys.

H. TCL Scripts

Tool Control Language (Tcl) is used to automate as many steps as possible. In Quartus Prime, developing and running Tcl scripts allowed us to perform a wide range of functions, such as compiling a design or writing procedures to automate common tasks.

The Tcl scripts can be used to manage a Quartus Prime project, make assignments, define design constraints, make device assignments, compile the design, perform timing analysis and access report. Tcl scripts also facilitate project or assignment migration. For example, when designing in different projects with the same prototype or development board, we can automate reassignment of pin locations in each new project. The Quartus Prime software can also generate a

Tcl script based on all the current assignments in the project, which aids in switching assignments to another project.

In this project, two Tcl scripts, *hps_sdram_p0_parameter* and *hps_sdram_p0_pin_assignments* are used. Both script was generated by Qsys. The *hps_sdram_p0_parameter* used to set the all the values of the SDRAM used. As for configuring the pin assignments on the FPGA, *hps_sdram_p0_pin_assigments* is used instead. These scripts are run as the final step before compiling the project.

I. Load the Programming File

After the compilation is done, the design can be load to the DE1-SoC board using Quartus Programmer. This is done by Active Serial (AS) programming which is performed by downloading a configuration bit stream directly into the Cyclone V FPGA through a USB cable. Then the downloaded stream bit is stored into the serial EEPROM chip in the board. Since EEPROM is a non-volatile storage, the information is retained even though the power supply is turned off. This method allowed the developers to use the same design without the need to reload it after the power is switched off. The Quartus Prime programmer connects to the USB blaster circuit on the DE1-SoC board and when the MSEL switch on the board is configure to "PROG", the configuration signals are sent to the FPGA.

J. Yocto Kernel Image

The build system for the OS can be obtained from the Yocto Project official website. This build system requires the developer to build the OS from basic. Since this is aim for experienced embedded Linux developer, we use the image files provided by Terasic Technologies, which manufacture the development board. The image file is downloaded from their website and the image file is put in a Secure Digital (SD) card. Since the kernel is stored in the SD card, no storage in the SDRAM has to be used and the kernel is retained after the power to the board is switched off.

Since Yocto Project is embedded operating system, it run any program created by developer, just like a normal operating system. The most essential component for writing the program is the Embedded Command Shell, which is provided by Altera as shown in Figure 2.

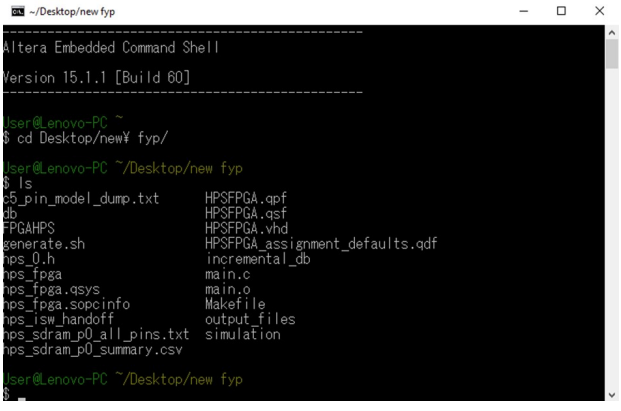

Figure 2: Altera Command Shell

Before writing the code for the program, an important header file has to be created. A generate.sh script is used to create *hps_0.h*, which is the header file needed to enable the program to run on Yocto. This header file contains all information about Qsys component, which in this project are the LEDs and the switches. To create *hps_0.h*, generate.sh is executed using the command shell. With the presence of the header file, the coding for the program can be proceeded. The program was developed in C language.

Since the interaction between HPS and FPGA happen through H2F-LW, it has to be define in the program. The base address of H2F-LW is FF200000 and its address region is 2MB long. Some pointers, *virtual_base, led_addr* and *sw_addr* has to be defined because we want to write and read the value directly from the memory address. The function of *mmap* is to create a new mapping in the virtual address space of the process. The size of this space corresponds to the H2F-LW address region. The last parameter in the line defines the offset, which is the base address of the H2F-LW. The address of the new mapping is returned as "virtual base" pointer. In order to the switches or LEDs, the base offset of the corresponding PIO is added. This offset is set in Qsys, and now is defined in the header file. When the value from the switch register is read, the program will assign it to LEDs. Since the pointer is defined as void, the variables switches is defined as unsigned integer. By doing so, the program will interpret the value in the pointer memory address, as unsigned 32-bit integer. To compensate the propagation delay in board, one second delay is added between the read and write operation.

Due to the small storage of a typically embedded system, embedded operating system does not have built-in GNU Compiler Collection (GCC). GCC is important to compile a C program. Because of that, the program created for Yocto has to be ported using *makefile*. The *makefile* will port the program by cross compiling the program with the GNU for Linux and GCC. After running the *makefile* script, a file will be created that contains the program. The file is transferred to the board using a secure shell client (*ssh*) via Ethernet.

IV. Results and Analysis

After the Yocto is properly implemented and successfully booted, it can be used by connecting it to a VGA monitor for displayed. A keyboard is connected by either USB or PS/2 for inputting the command for the Linux. The Yocto also can be used by connecting it to any computer using a secure shell client via USB or Ethernet connection. In the secure shell client, the monitor and keyboard of the connected computer is used as the display and input for Yocto. Since Yocto is a fork version of Linux, any standard UNIX commands such as *ls, cd* and *free* can be executed within it. Table II list some example of the supported command. The list of files and directories of the root file system can be obtained by executing *ls –al* command. Figure 3 shows the output of *ps* command.

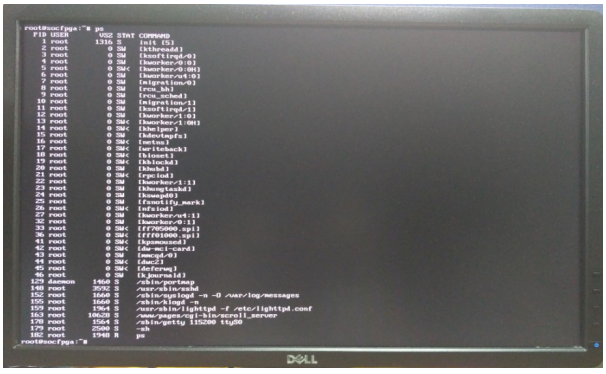

Figure 3: The output of *ps* command.

Unlike uClinux where the kernel is stored in the SDRAM, no space is wasted to store Yocto's kernel. In this experiment, the memory used for Yocto is 18 MB out of 1 GB of SDRAM available.

A. Resources Utilization

After the design is done, the resources utilization used in the project can be analyzed using the built-in tool in Quartus Prime. The tool needed is run automatically at the Analyze and Synthesis stage during the project compilation process. This logic and resources utilization is one of the factor when choosing the programmable logic device (PLD) which the design will be implemented. From the value obtained from the evaluation, we are able to determine the amount of logic design needed, or the size of the device.

In Quartus Prime, the place-and-route tools will utilize its advanced algorithms and optimization techniques to increase the performance and reduce the amount of the logic needed for a given project. In some scenario, the design may contain megafunctions and only the place-and-route tool is the common source for accurate logic utilization reports. The report provided the tool is consisting of three main areas, Quartus Prime register and memory packing, including register-packing into logic elements, DSP blocks, and I/O elements and others; Quartus Prime netlist optimizations; and IP megafunctions. Table I and Figure 4 summarize overall resource utilization for this project.

Table I: Resources Utilization.

Logic Block	Available	Used	Utilization
Logic Elements	32,070	299	<1%
Registers	32,070	596	2%
I/O	457	130	28%
DLLs	4	1	25%

Table II: Example of the supported command.

Command	Process
`ls`	List the files and folders in the current directory
`ps`	List information about the currently running processes including the PID number for each process
`kill`	Stops a running process. The PID for the process that will be halted must be specified.
`free`	Lists the amount of used and available memory
`ping`	Starts a network utility that can be used to test network communication
`ifconfig`	Displays and sets network configuration settings
`dhcpcd`	Starts a DHCP client daemon that will automatically acquire an IP address and other the network settings from a DHCP server
`ftpd`	Starts an FTP server daemon that can be used to access files on the uClinux system from over the network
`telnetd`	Starts a telnet server daemon that can be used to access a µClinux console terminal
	over the network
`boa`	Starts a lightweight web server on the µClinux system. The root web page is stored in /home/httpd/.
`mount`	Used to mount/access different filesystems including USB memory sticks

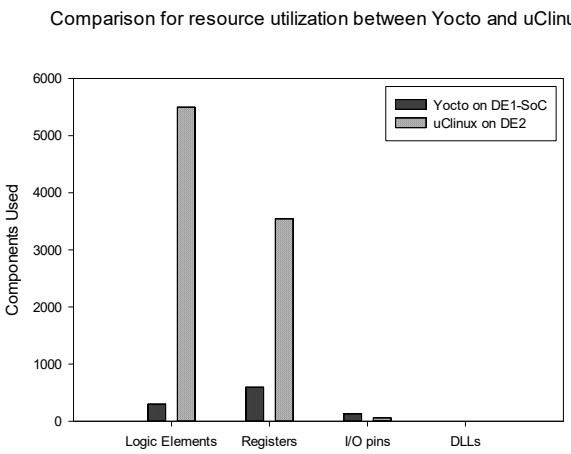

Figure 4: Comparison of resources utilization between Yocto and uClinux.

B. Power Consumption

Nowadays, the design has become larger and more complex while the process technology become smaller. This leads to the importance of power consumption analysis on the design. We evaluated power consumption of the design using PowerPlay Power Tool. The tool produced the evaluation for various type of power dissipation as in Figure 6. The total power consumed by the components used is 1636.64 mW. The power consumption of the FPGA, which consisted of the core dynamic, the core static and the I/O has the total of 558.68 mW.

It can be seen that power consumed by I/O parts is the lowest. When a design is synthesized and fitted to the target device, the electrical standard used by each I/O cell and the capacitance load value is specified in order to obtain the accurate I/O power estimates. The I/O power consumption is effected by the number of elements used in the design. The operating mode of each circuit also affects the power consumption.

The core dynamic, which is the additional power required for signal activity or toggling, is evaluated at 96.56 mW. This power dissipation occurred because there is power loss when the signal travel between FPGA, HPS and the PIO. This loss can be reduced by using smaller device since the distance between the components is reduced.

The core static has largest power consumer for the FPGA. Static power is the power consumed by the logic elements in the design. Another factor that affect the core static power consumption is the temperature.

As for the power consumption for the HPS, it consumed 1077.97 mW if it designs use two processor core and 1005.13 mW for single processor core used. The power consumption for the HPS does not depend on the design since the onboard processor cannot be configure in terms of its logic elements. Figure 5 and Figure 6 gives the total indication of the overall power consumption.

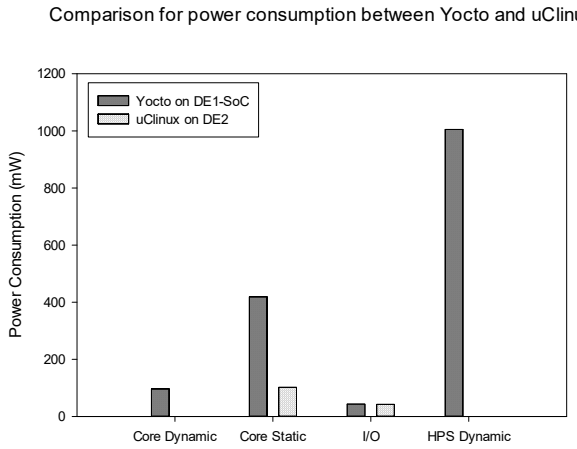

Figure 5: Comparison of Power Consumption between Yocto and uClinux.

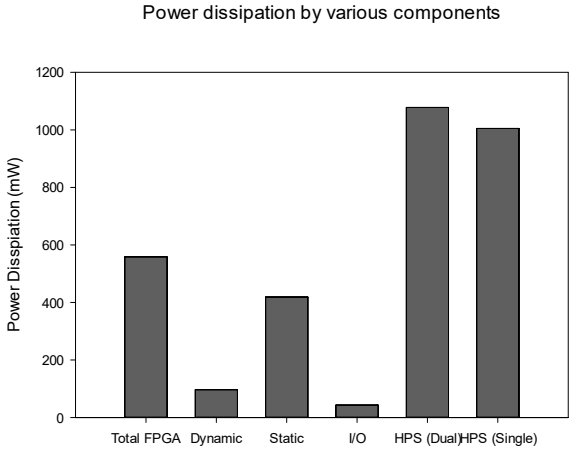

Figure 6: Power dissipation by various components.

V. CONCLUSION

The main objective of the project is to implement an embedded OS, Yocto project on the Altera DE1-SoC FPGA board. For this, the connection between FPGA, HPS and the PIO involved in the design must be properly configured to ensure proper communications between these components. After carrying out this project, we believe that embedded Linux will play a big role in technology as we can see from the adoption rate of Android, a Linux based operating system. The increased in development of embedded Linux is also catalyzed by the introduction of Internet of Things. The ability of Yocto to run on many of the major processors' architecture such as Intel x86 and the flexibility of FPGA are the main attraction for embedded system developer.

ACKNOWLEDGMENT

This work was supported by the Fundamental Research Grant Scheme (FRGS), Grant No. FP042-2014B.

REFERENCES

[1] T. S. Hall and J. O. Hamblen, "Using an FPGA Processor Core and Embedded Linux for Senior Design Projects," *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*, San Diego, CA, 2007, pp. 33-34.

[2] R. P. Jasinski, M. R. Moroz and V. A. Pedroni, "The impact of operating system adoption in an embedded project: A case study," *Programmable Logic (SPL), 2012 VIII Southern Conference on*, Bento Goncalves, 2012, pp. 1-7.

[3] F. Devic, L. Torres and B. Badrignans, "Securing Boot of an Embedded Linux on FPGA," *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, Shanghai, 2011, pp. 189-195.

[4] T. Santini, L. Carro, F. R. Wagner and P. Rech, "Reliability Analysis of Operating Systems for Embedded SoC," *2015 15th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Moscow, 2015, pp. 1-5.

[5] P. Cotret, F. Devic, G. Gogniat, B. Badrignans and L. Torres, "Security enhancements for FPGA-based MPSoCs: A boot-to-runtime protection flow for an embedded Linux-based system," *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*, York, 2012, pp. 1-8.

[6] F. Devic, L. Torres and B. Badrignans, "Securing Boot of an Embedded Linux on FPGA," *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, Shanghai, 2011, pp. 189-195.

[7] P. Cvek, T. Drahoňovský and M. Rozkovec, "GNU/Linux and reconfigurable multiprocessor FPGA platform," *Electronics, Control, Measurement, Signals and their application to Mechatronics (ECMSM), 2013 IEEE 11th International Workshop of*, Toulouse, 2013, pp. 1-5

[8] H. Guanglin, G. Rujun and Y. Shi, "Application of FPGA in Small UAV Autopilot Based on Embedded Linux System," *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, Taipei, 2007, pp. 731-734.

[9] R. H. Klenke, "Experiences Using the Xilinx Microblaze Softcore Processor and uCLinux in Computer Engineering Capstone Senior Design Projects," *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*, San Diego, CA, 2007, pp. 123-124.

[10] A. R. Sánchez, O. Alvarado-Nava and F. J. Z. Martínez, "Network monitoring system based on an FPGA with Linux," *Technologies Applied to Electronics Teaching (TAEE), 2012*, Vigo, 2012, pp. 232-236.

[11] J. S. Monson, M. Wirthlin and B. Hutchings, "Fault Injection Results of Linux Operating on an FPGA Embedded Platform," *2010 International Conference on Reconfigurable Computing and FPGAs*, Quintana Roo, 2010, pp. 37-42.