COP 5615: Distributed Operating Systems Principles
Internet of Things Support in Xinu
Fall 2016
Term Project Report

Group 20

Smart Lights

Ajay Kumar, Anuja Tole, Jaikrishna Sukumar, Manisha Dewal,

Pratik Karambelkar

ajaykr009@ufl.edu, atole@ufl.edu, jaikrishna@ufl.edu, manisha.dewal@ufl.edu,

pratikaditi1169@ufl.edu

1. Describe your project using this table

Part	Indicate Completeness (give a no. from 1-10), followed by Description
Xinu I/O Interface design	**Completeness: 10** Our I/O interface includes GPIO, ADC and PWM. 1) **GPIO**: We used ARM335x technical reference manual to familiarize ourselves with these modules. BeagleBone consists of 4 memory mapped ports with 32 GPIO pins each, i.e. a total of 128 GPIO pins. Four banks consisting of various control registers are used to control the behavior of the ports. The control registers are at a fixed offset from the base of each bank (OE, DATAIN and DATAOUT). These control register are 32 bit registers and each bit in the respective register controls one pin in the bank according to the bit position. Most of the GPIO pins are available for interface via the expansion header pins in the BeagleBone. We can interface our devices by setting the appropriate header pins to the GPIO mode and use the corresponding GPIO bank registers to enable and write values. **2) PWM:** Pulse Width Modulation subsystem in BeagleBone is used by our application to control the intensity of LEDS by varying the duty cycle of the output wave. We can vary the color displayed by the RGB LED by varying the duty cycle and hence the light intensity of each component channel R, G and B. There are three PWM subsystems available in BeagleBone whose interface is exposed by the expansion header pins. Each subsystem consists of two pins and so there are 6 pins in total for collecting the PWM output. The expansion header pins are multiplexed to provide several functionalities. So, the pin mode must be changed to that of PWM before using it. Each PWM subsystem consists of control registers such as TBPRD and CMPA in the Enhanced PWM mode which are used to set the frequency and duty cycle respectively. Other PWM registers and subsystems used are Clock configure registers and Action configure subsystems. 3) **ADC**: The BBB ADC is used by our application to read the LDR(Light Dependent Resistor) values which indicate

	ambient light. These values are used to control the LED intensities using PWM. The ADC is managed using device table and an interrupt handler. The adcinit() devcall initializes the ADC Clock and sets the interrupt handler using set_evec() function. A semaphore is waited on in the adcread() function which will be signalled in the ADC interrupt handler. The ADC and the GPIO subsystems have been implemented as Xinu devices and the high-level I/O calls such as read & write has been used to invoke their operations.
IoT-specific concerns your design addressed, including but not limited to Energy	**Completeness: 8** a) **Communication protocol**: We have used UDP to communicate between devices which is one of the standard protocols. We used the industry standard protocol so that sharing data and communicating with other devices will be easier and we can expand on the functionality as required. b) **Inter device compatibility**: Since we've used standard communication protocols and command mechanisms, it should be easy to integrate with other things/devices. c) **Ease of use**: We've exposed the user interface via an HTML page and deployed it using a web server which can be accessed from any PC or mobile device. d) **Energy**: We have addressed the energy constraints in most of the aspects. We have used push/pull model for communication between devices by using UDP packets. UDP is a connectionless protocol which means it doesn't require active refreshing mechanism to keep the connection alive. In our design, every smart light device registers on specific UDP port and puts the application to sleep and waits for commands. Whenever there is a packet received, the application wakes up and processes the request. e) **User Experience**: User experience is an important topic in IoT. We focused on keeping a simple user interface for the user to easily communicate with our application. The user interface has been made very simple and an about page has been made to explain the features of the system
Xinu I/O Interface implementation and testing	**Completeness: 8**

	GPIO and ADC were configured as devices in Xinu with a few high level IO operations such as read, write, control, init and interrupt handler. The device table configured contains the low level driver implementations generated by the DDL document and parser at compile time. The interactive shell was used to invoke and test the high level IO operations through exposed system calls. Within the low level drivers, GPIO, PWM and ADC low level registers were addressed using the memory map specifications and bit manipulation was used to control the functionality of pins.
Design of IoT Description Language, Language processing and code generation	**Completeness: 10** **Indicate**: XML-, JSON, Other-based - **XML** **Source**: any open source used? Indicate the Github or other s/w package name. <include the library used> - **Python XML library** **Design:** A DDL parser was developed using Python and its XML parsing library. The parser takes in two arguments in the format <device driver folder> <Xinu codebase folder>. It looks for a specific file called devconf.xml which contains the information relating the specific device such as the device name and associated functions. It then works on configuring that device to work with Xinu operating system by plugging in the appropriate functions into the correct place of Xinu code. It also modifies the device configuration file (config/Configuration) and the Makefile to include the device folder for rebuilding. It should be possible to write the device operations for any device in C and plug it in Xinu by invoking the DDL parser. The user then just has to make again to get the binary with the device driver included.
Implementation and testing of IoT Description Language, Language processing and code generation	**Completeness: 8** The Device Description Language consists of an XML file named devconf.xml which includes the device name, the actual name of the device to be plugged in and the device operations such as read, write etc. The XML fields describe the specific functions to call for the operations that are defined. By looking at these fields, the DDL parser should be able to configure any device with the Xinu source code. We tested the code generation by the DDL parser by defining our PWM and ADC submodules as Device Drivers implemented in C. The code generation script was able to successfully install the device driver code inside of Xinu and was able to make, build and run successfully.
Implementation and testing of overall on-board driver code	**Completeness: 8**

(upper- and lower-level drivers, including generated code)	The device driver code was completely tested on the device by forcing all corner cases and high stress conditions. The lower-level device driver code was abstracted into higher-level I/O operations and the resulting device driver code was generated and configured with the help of the DDL. The modified Xinu source code was successfully built and tested by running it on the board. Reference: TI Codebase
Did you use the same existing device driver structure and mechanisms in Xinu?	**Completeness: 9** Yes, we've used the same generalized format suggested by Xinu source code and have written the appropriate functions that interact with the device. This has been followed for the ADC and GPIO.
Approximate % driver code generated with respect to overall on-board driver code	15%
Which device externalization abstraction have you chosen (which existing technology or any new ideas)? You may, or may not explain the reason for your choice.	We have used an UDP command socket as a mechanism of device externalization. Xinu registers a process at a particular UDP port and puts the application to sleep until there is a command/message to process. When the network protocol receives a particular packet for this port, the message is constructed by the operating system and the process is woken up.
How, where, and when do you specify the edge and cloud addresses of the device? Explain how device configuration and initialization are done including device externalization.	The cloud and the edge addresses are specified via a configuration file for the edge node. The edge device parses the values in the configuration files and communicates with the things according to the specifications mentioned. The device initialization is made by the edge node once it starts up. The things start by default in the shell mode and take up their port numbers for the command sockets by means of an argument.
Give the details of the externalization abstractions design.	The things that have been designed in the project externalize themselves to the edge by means of a command mode. The things act as a slave to the edge node and just work according to the commands received by it.
Describe the implementation of the abstractions (how they connect to the actual device), and discuss any IoT-specific concern (including energy) that may have been addressed by your implementation.	The things (implemented on BBB) expose an UDP command socket registered by the Xinu OS which links it to a particular process/application serving the edge commands. The 'iotled' shell application that has been developed registers a particular UDP socket for it to use and requests the OS (Xinu) to wake it up when there's any message for itself. This way, the application doesn't have to continuously maintain an active connection to the edge which would waste a lot of CPU cycles, network packets and also drain the power. This way, the IoT application has been made extremely energy efficient. The application running on the things have been designed to be very efficient and lightweight. Most of the processing is being done in the edge node. Also, the high level features of the application such as the LDR activated mode have been designed by keeping the energy constraints in mind. For example, activating the LDR mode will only

	trigger the thing to change the color intensity when there's a significant change in the ambient light conditions. There's no use in letting the 'iotled' application of every update in the ambient light conditions; also that would waste a lot of energy.
Describe your on-board IoT devices Demo App.	**Devices**: Low level devices: GPIO, ADC, PWM High level devices: - Pyroelectric Infrared Sensor - for motion detection - Light Dependent Resistor - for measuring ambient light conditions - RGB LED - to generate light with wide range of colors and intensities **App:** The onboard demonstration application is developed as a shell command in Xinu namely 'obdemo'. When executed, a program will start running locally on the board that will detect the presence of a person in the room and change the color of the RGB LED accordingly.The LED emits red color when there is no one in the room and green otherwise. We have also developed a DDL parser for this application so that the user can control and configure the parameters of this application. This application can be used by the user to run the device/thing in the standalone mode for testing the hardware without having to depend on the edge commands.
Describe your web-based IoT devices Demo App.	Smart lights: We have implemented a webapp - 'SMART LIGHT' that can control the entire system and make adjustments automatically depending on other factors like occupancy of the room and ambient light. It can also be used for energy efficiency and for aesthetic reasons. Additionally, the web application has a color palette which gives the user wide range of colors to select from. We have 3 modes: 1) Manual mode: This mode sets the color of the LEDs as manually set by the user. The color will remain on in the room till another change is made. 2) Scheduled mode : This can be used to set a start and a stop time for the light to be on. The light will automatically switch on and off with a particular color as specified in the webapp. 3) LDR based mode: In this mode, the color of the light will be input from the user and the intensity will be varied according to the ambient light conditions as detected by the outdoor sensor. Along with each of these modes, the user can select an **energy saving option** which will automatically switch off the light to save power when there's no occupancy in the room.

2. Challenges

Challenges your group faced. What was the most time consuming parts of the project? what piece(s) would you have really liked to have us provide to you so the total effort is more manageable (again, if any)?

We faced a lot of challenges during the implementation of the project.
They are:

1) Figuring out the registers of the low level GPIO pins, ADC and PWM and understanding their functionality.
2) Integration of all the different components into a single application. This was one of the most time consuming parts of our project, but this also taught us a lot of stuff. The complexity of making several things/devices work at the same time together required us to look into issues that we never faced in an software engineering using sequential programming.
3) Configuring all the registers for the PWM subsystem in the right order with the right counter values was a difficult task. It required a lot of effort to read through the entire section of the technical reference manual before PWM code could be written.
4) One another challenge was deploying the dashboard application. Choosing a universally compatible application (HTML) required us to use a web server along with a Websocket over TCP connection. It took a considerable effort learning technologies such as Javascript which were not required for an operating systems course.
5) Opening and reading/writing from/to the UDP socket on the PC to communicate with Xinu process.
6) Working from the lowest level to the highest level of code. We had to look into registers for GPIOs, device driver , UDP IP protocol and web application. We had to understand and work from Assembly level, C, Python, HTML and Javascript.
7) Testing the LDR and PIR sensor in different light conditions.
8) Time constraint especially because the help document and the report format was posted late.
9) Less resources available online for XINU specific limitations and issues.
10) To make the project better manageable with respect to the time constraints, it'd have been better if the following reference implementations were provided to the students:
    a) Example usage of the GPIO with the memory mapped registers
    b) ADC usage on ARM 3358 chip
    c) Registers to handle and set to get the PWM working on the chip

3. Overall Experience

Overall it was a great learning experience implementing a project using the latest technologies. It was very challenging to build the low level interface along with high level application and integrate each component to work together in such a limited time. Writing the device drivers and emulating the cloud gave us hands-on experience and exposed us to the nitty gritties of these technologies. Overall we learned a lot about the IoT and IoT specific concerns.

4. Effort Distribution

The effort was completely even among all the team members.