**COP 5615: Distributed Operating Systems Principles**

**Internet of Things Support in Xinu**

**Fall 2016**

**Term Project Report**

**Group 03**

**Madhav Agarwal, Nidhi Makhijani, Dhiraj Borade, Srikrishna Iyer and Digvijay Kulkarni**

[madhavagrawal17@ufl.edu](mailto:madhavagrawal17@ufl.edu)**,** [nidhimj22@ufl.edu](mailto:nidhimj22@ufl.edu)**,** [kris21592@ufl.edu](mailto:kris21592@ufl.edu)**,** [dkulkarni@ufl.edu](mailto:dkulkarni@ufl.edu)**,** [dhirajborade@ufl.edu](mailto:dhirajborade@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 | 10  Xinu I/O design is Implementation as two level of driver, lower level driver abstraction and higher level driver abstraction.  The higher side of device driver implementation is consist of code generation using GSL library.The gpioinit.c and adcinit.c are generated through the gsl files by taking input from the DDL file. This file is the higher level of device driver code.The gpioinit initializes the pin and port for output/input as per the values in the DDL file. It also handles the interrupt on detection of rising and falling edges.The adcinit handles the interrupt whenever the buffer gets full. The other higher level driver are adcread.c (read the adc value in a buffer and in a circular buffer as a producer consumer problem), gpiopush.c (to write the digital output in a circular buffer of size one similar to putc implementation ), gpiopull.c (to read the digital input similar got getc higher level call)  For the lower level device driver we have implemented adchandler.c (Interrupt is triggered when a adc FIFO buffer register is full so that all the value can be read from the adc data register ), gpiohandle\_in.c (read the value from the gpio datain register and update the circular buffer and signal the semaphore), gpiohandle\_out.c (write the value of pin into the gpio dataout register ), gpiohandler.c(Interupt is trgiggered for digital rising and falling edge detection.). |
| IoT-specific concerns your design addressed, including but not limited to Energy | 10  We are not continuously reading the data from the device by polling for GPIO Digital and ADC. For Digital Input, we will read the input only when interrupt is triggered, otherwise the process will wait.  For ADC Read, we are reading the temperature in an interval of 10s and rest of the time the process is in sleep mode. In this way, the CPU of Xinu is not being used continuously, thus saving a lot of energy.  To decrease the latency of I/O operations, we are using interrupt based drivers, where interrupt will be triggered when the status of the interrupt changes.  For implementing the drivers, we use the Xinu structure of dividing the driver into a high-level device abstraction and a low-level implementation. This high level of device driver abstraction is quite powerful because it gives the flexibility to configure the same for multiple devices using the same GPIO. |
| Xinu I/O Interface implementation and testing | 10  The Xinu I/o interface is implemented using the functions gpioinit, gpiopush, gpiopull, adcinit, adcread, gpiohandler and adchandler.   * We tested the devices by connecting to the pins and checking the output through manipulating the values of the pins. * We tested the devices through main() function * We tested the devices through the higher-level driver codes for initialization, reading and writing into the buffer. * We tested the devices through taking the values from the DDL file on compile time which generated the codes for the higher-level functions. |
| Design of IoT Description Language, Language processing and code generation | **Indicate**: XML-, JSON, Other-based  We have used XML file for implementing the DDL Logic.  **Source**: any open source used? Indicate the Github or another s/w package name.  Yes. We use GNU Specific Library (<https://github.com/imatix/gsl>).  **Design:** The DDL is used to generate the high-level device driver C codes at compile time. A code snippet is shown below. The DDL contains information about all the BBB’s connected and all the devices connected to each BBB. Also, the status of each device, whether high or low, the mode of the device and the pin/port number of the device is shown through the DDL. The DDL code generation is implemented using GSL (GNU Scientific Library) where an XML file and a GSL file is used to generate a C file. A snippet of the .gsl file is shown below. The file takes the values from the xml file above and generates a .c file |
| Implementation and testing of IoT Description Language, Language processing and code generation | 10  The DDL file used is an XML file whose parameters are used by the GSL file to generate the C code on compilation. The port and pin numbers are initialized in the XML file that are used to generate the higher-level C functions for the devices connected. |
| Implementation and testing of overall on-board driver code (upper- and lower-level drivers, including generated code) | 10  Testing of the overall on-board driver was done in various steps.   * We tested the devices by connecting to the pins and checking the output through manipulating the values of the pins. * We tested the devices through main() function * We tested the devices through the higher-level driver codes for initialization, reading and writing into the buffer. * We tested the devices through taking the values from the DDL file on compile time which generated the codes for the higher-level functions. |
| Did you use the same existing device driver structure and mechanisms in Xinu? | Yes. |
| Approximate % driver code generated with respect to overall on-board driver code | 47% |
| 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. | 10  We are using adcread and adchandler, gpiopull and gpiopush as the device externalization abstraction. Adcread because the adc has to read 4 bytes at a time from the buffer.  Gpiopull and gpiopush acts as getc and putc abstractions which read and write a character to the buffer. |
| 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. | 10  The edge and cloud addresses of the device are described in the DDL xml file. The EDGE application uses these values for initialization. The edge layer is an interface between the low level devices/platforms and the outer world.  -Edge layer exposes different functionalities to get the information about the devices and their states as well as actuation of devices through RESTful web service APIs.  -Edge gathers and updates itself with information about the connected devices (e.g. devices connected to BBB). To do so edge communicates with BBBs over UDP.  -During initialization Edge parses device DDL xml file to know the connected BBBs and the devices those are connected to corresponding BBBs. Also it acquires and stores their corresponding IP addresses and ports for future communication.  -Edge listens on messages from connected BBBs and updates the states of the devices based on the messages received. BBBs will send message to edge only when there is a change in the state of any of the devices connected to it.  -Edge can also send instruction to BBB to actuate a device over UDP.  -Edge takes care of any conversions, e.g. converting/calculation temperature values from the readings received from sensor based on an expression predefined in DDL file. |
| Give the details of the externalization abstractions design. | 10  In our design at the lowest level, we have two BBB’s one connected to a temperature sensor and an LED, and the other connected to an LED, a Button and an LDR. The BBB’s communicate with an EDGE application through UDP where the EDGE uses the values from an XML file to initialize and gives the appropriate values to a Web Based Cloud app on request. |
| 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. | 10  The EDGE and BBB’s communicate to each other via UDP protocol. The EDGE initializes itself using the values from the DDL file. From the same file, it understands which devices are connected to which BBB.  We are not reading the data continuously from the device by polling for GPIO Digital and ADC. For Digital Input, we will read the input only when interrupt is triggered, otherwise the process will wait. For ADC Read, we are reading the temperature in an interval of 10 sec and rest of the time the process is in sleep mode. In this way, the CPU of Xinu is not being used continuously, thus saving a lot of energy. |
| Describe your on-board IoT devices Demo App. | 10  **Devices**: describe any of the basic, composite and virtual devices used.  **App:** The devices used are 2 LED’s, a button /LDR, and a Temperature Sensor (TMP36)  The combination is used to implement a smart home. The LED represents the lights and Air Conditioning. On the digital input from the LDR/button, we detect if it is Daytime or Nighttime, and accordingly turn the lights on and off. The temperature sensor measures the temperature and accordingly the Air Conditioning is turned on or off.  So, our edge has two nodes BBB1 and BBB2 (two beagleboard are connected with the Edge). In one Beagleboard we a LED and a digital input (ldr/button) as device and in the other beagleboard we have LED and temperature sensor as devices. We are monitoring the input signal from both the beagleboard (TEMP sensor and digital Input), and eventually updating the edge if any changes occurs like change of digital input (high/low) or temperature measurement. For the output devices like in this case two LEDs in two beagleboard we are operating the LEDs from the web application so that a External user can control the devices using web application or from his browser.  In case when a button is turn on one beagleboard we are communicating through Edge to control the LED on the other board. In this case Edge is also a medium of communication between two nodes(beagleboards). |
| Describe your web-based IoT devices Demo App. | 10  The web-app is developed using HTML, CSS and Jquery. These are supported by every modern browser today.  Every time we toggle the switch of the two LEDs, an AJAX request is sent to the Rest services that further prompt the UDP server which then interacts with the BBB. The web-app polls for the status of the two LED’s and the temperature every 10 seconds. The polling is done via Long polling using Jquery AJAX requests. AJAX requests are non-blocking and asynchronous.  The REST services return the HTTP response which is then parsed and displayed on the dashboard. |

**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)?

Many challenges were face during implementation of this project.

* We had to read extensively about the device driver implementation in Xinu and the interfacing of devices with the BBB.
* We had to research about the DDL Code Generation Logic, went through a lot of parsers and tested extensively with each.
* We faced challenges with communication between the interfaces and had to go through a lot of material to finally overcome the challenges.
* Integration and Testing of the complete project was also a major part, where every part was tested individually at first and then by combining all the devices together.

**3. Overall Experience**

Overall experience. Describe your overall experience good or bad.

Overall experience was good. The group got to learn the basic functionality of Xinu and the implementation of device drivers and the implementation of IoT. Everything from the lowest level (pin configuration) to the highest level (Web interface) was involved.

We learnt about how different components interact with each other to create real-world products. We also learnt how to collaborate and work in a team to develop an end-to-end product.

**4. Effort Distribution**

Report only if effort was considered by any member of the group to not be even. In this case a table showing the names, ID’s, and percentage of effort should be provided.

Everyone has put in equal efforts in all the sections (Device Drivers/EDGE/Cloud) of the project. The Ownership and Major Contribution of members towards the part they were assigned is mentioned below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Email ID** | **UF ID** | **Percentage** | **Ownership (Major Contribution)** |
| Madhav Agrawal | m[adhavagrawal17@ufl.edu](mailto:Madhavagrawal17@ufl.edu) | ⁠⁠⁠2975-4919 | 30 | * Implementation of low level GPIO and ADC device drivers and communication between the BBB and EDGE application. * Implementation and testing of code generation using DDL. |
| Nidhi Makhijani | n | 3987-5129 | 17.5 | * Implementation of Cloud Based Web App. |
| Srikrishna Iyer | k[ris21592@ufl.edu](mailto:Kris21592@ufl.edu) | 8259-8779 | 17.5 | * Implementation of higher level GPIO functions. * Implementation of code generation using DDL. |
| Dhiraj Borade | [dhirajborade@ufl.edu](mailto:dhirajborade@ufl.edu) | 4595-8142 | 17.5 | * Implementation of higher level ADC functions. * Hardware configuration and testing of the devices. |
| Digvijay Kulkarni | [dkulkarni@ufl.edu](mailto:digvijaykulkarni@ufl.edu) | 4824-1954 | 17.5 | * Implementation of the EDGE application. * Communication between EDGE and BBB over UDP. |