**COP 5615: Distributed Operating Systems Principles**

**Internet of Things Support in Xinu**

**Fall 2016**

**Term Project Report**

**Group 9**

**Chang Xu,** [**theoneneo@ufl.edu**](mailto:theoneneo@ufl.edu)**,**

**Maoyao Yuan,** [**yuanmaoyao1@ufl.edu**](mailto:yuanmaoyao1@ufl.edu)**,**

**Haoran** [**Cui, cuihaoran@ufl.edu**](mailto:Cui,%20cuihaoran@ufl.edu)**,**

**Tianyu Ma, tianym94@ufl.edu,**

**Yinan Zhang, zyn1018@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 | Our project is building a fire detection system with flame sensor, button, temperature sensor, buzzer, and lights. We use 3 GPIO pins as output, 2 GPIO pins as input, and 1 AIN pin as input.   1. Device Buzzer: BBB P8 pin 8; AM335x gpio2[3], GPIO No 67; output mode. 2. Device Red Light: BBB P8 pin 9; AM335x gpio2[5], GPIO No 69; output mode. 3. Device Green Light: BBB P8 pin 10; AM335x gpio2[4], GPIO No 68; output mode. 4. Device Temperature Sensor: BBB P9 pin 36; AM335x AIN5; input mode; 5. Device Flame Sensor: BBB P8 pin 25; AM335x gpio0[26], GPIO No 26; input mode; interrupt IRQ 96, rising detection. 6. Device Button: BBB P8 pin 12; AM335x gpio1[12], GPIO No 44; input mode; interrupt IRQ 98, Rising detection. |
| IoT-specific concerns your design addressed, including but not limited to Energy | Considering the number of IoT devices could be tremendous, if devices send data packages to Cloud directly, the workload of Cloud would be extremely high. Therefore, we use Edge to reduce the number of packages sent to Cloud. Devices send data packages to Edge, and Edge filters duplicating information. Streamlined and accurate data packages will be sent to Cloud after the processing of Edge. |
| Xinu I/O Interface implementation and testing | Implementation:  Since Xinu only supports UDP, we use UDP to communicate between Device and Edge. Device uses port 35271; Edge uses port 35272. And we use http to communicate between Edge and Cloud.  Testing:  For circuit part, we experimented and tested the circuits with breadboards. For network communication, we did comprehensive integration testing with Devices, Edge and Cloud. |
| Design of IoT Description Language, Language processing and code generation | **Indicate**: XML  **Source**: JDOM xml parser  **Design:**  To generate the appropriate drivers, we have defined devices description language which contains GPIO pin, AIN pin, modes(input or output), type of input and output data(digital or analog), devices’ default value and addresses of related register. We also have created a DDL parser(JAVA) to parse the DDL and generate corresponding device drivers. |
| Implementation and testing of IoT Description Language, Language processing and code generation | We edited DDL using XML and the parser is written in Java taking JDOM xml parser as reference. To test the correctness of the code generated by parser, We load the code generated by our DDL parser into BBB comparing to the code written by ourselves. It turns out that the system works successfully in both ways. |
| Implementation and testing of overall on-board driver code (upper- and lower-level drivers, including generated code) | Basically, the drivers are used to initialize devices and contains functions that may be called by devices. We have written several functions gpiowrite(), gpioinit(), aininit(), ainrea(), devinit() which can initialize corresponding registers’ value and define functions will be called later. We conduct several simulation tests such as using lighter to trigger the alarm, pushing button to trigger the alarm to find out whether drivers work properly or not. |
| Did you use the same existing device driver structure and mechanisms in Xinu? | No. Xinu will generate config.c and config.h automatically in its own way. We decide not to contain Xinu’s building function in our DDL file due to it requires a lot of workload. |
| Approximate % driver code generated with respect to overall on-board driver code | 100%. Besides the automatically generated driver code, we also integrated static low-level driver implementation in our generator. In that way, we are able to make our driver code 100% generated. |
|  |  |
| 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 use each Device to represent a room in the Cloud to monitor the fire situation and temperature of that room. |
| 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. | Device can automatically retrieve IP address of Edge when it’s connected to the local network. Edge can register new Device and send Device status information to Cloud. Could can monitor and manage Devices by sending commands to Edge, then Edge forwards the commands to each Device. |
| Give the details of the externalization abstractions design. | Device broadcasts a specific message to find Edge when it’s the first time to connect to local network. Edge responses to such a specific message so the connection between Device and Edge is built. To maintain the connection, Device periodically sends heartbeat message to Edge. Therefore, Edge determines the Device’s status and sends the information to Cloud. |
| 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. | A web-based interface is implemented on Cloud side to monitor and manage Devices. In the control panel, administrator can know alarm status, temperature, and online/offline of all Devices. Specifically, the administrator can know the alarm is triggered by sensor or by pushing button.  The system consumes very little computing power of Device. It only collects data from input sources, communicates with Edge, and executes commands from Cloud.  The consuming of network bandwidth is also very low. Since IoT Devices may have to work in a limited bandwidth environment, only necessary information is communicated with Edge.  Besides above, the system is energy-saving. We use Keye Flame Sensor to detect fire which is sensitive enough but consumes low energy. |
| Describe your on-board IoT devices Demo App. | **App:**  Fire detection by flame sensor and alert raising;  Raising alert by pushing button;  Green light on normal mode, and red light on alarm mode.  Device online/offline status checking and updating.  Cloud can dismiss fire alarm (in web-based monitoring interface). |
| Describe your web-based IoT devices Demo App. | Web-based IoT is composed by two modules: Edge and Cloud. We implemented a web-based Cloud and an Edge to manage the status of Devices. Edge is implemented by Java and Cloud is implemented by Ruby on Rails. MySQL DB is utilized in the backend of Cloud to store Device data. Edge and Cloud can be run on different laptops. |

**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)?

Challenges we faced include:

1. Determination of what we should build for the project, what sensor should be used. After a long period of meditation and discussion, we finally decided to make a fire detection system with flame sensor, temperature sensor and push button.
2. Study on AM335x. In order to know which registers should be operated and in what sequence, we spent days of time on the study of AM335x TRM and on the experiment with BBB.
3. Technical solution of Edge and Cloud. We tried different ways and languages about building Edge and Cloud. With many experiments and setbacks, we had the Edge built by Java, the Cloud built by Ruby, and the DB powered by MySQL.
4. The study and application of DDL. Since DDL is a new technology used specifically for IoT, we spent a lot effort to understand its principles and to apply it in our project implementation.

We consider point 2, study on AM335x, and point 3, study on Edge and Cloud, to be equally the most time-consuming parts of the project. And we think the total effort could be more manageable if some hardware introduction about BBB and AM335x could be provided.

**3. Overall Experience**

Overall experience. Describe your overall experience good or bad.

* **Bad**: Figuring out what AM335x registers should be operated and in what sequence is not a good experience. The AM335x TRM contains only details of every module and registers themselves, no any example or use case about how you should use these registers. Days of time were spent on trying operating different registers. And the final solution looks relatively simple, only several registers need to be controlled, compared with the progress to figure it out.
* **Good**: We are all excited to see the Devices can detect fire and report to Cloud, and Cloud can monitor and manage the situation. The collaboration of Devices, Edge and Cloud rewards our efforts. This project is very interesting.

**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.

* We think our efforts spent are quite even.