**COP 5615: Distributed Operating Systems Principles**

**Internet of Things Support in Xinu**

**Fall 2016**

**Term Project Report**

**Group 15**

**Pushparaj Britto, Harsh Shah, Parth Rampal, Prashant Mishra, Rohan Patil**

[**brittopush@ufl.edu**](mailto:brittopush@ufl.edu)**,** [**harshyshah@ufl.edu**](mailto:harshyshah@ufl.edu)**,** [**parth.rampal@ufl.edu**](mailto:parth.rampal@ufl.edu)**,** [**jeeves901@ufl.edu**](mailto:jeeves901@ufl.edu)**,** [**rohan.patil@ufl.edu**](mailto:rohan.patil@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  XINU driver abstraction paradigm of read, write, open, close, init is used as basis for device driver functions.  LED Driver is divided into upper half and lower half. The upper half is implemented as ledinit and ledwrite. The lower half is implemented as output interrupt handler - ledhandler.  Temperature driver is divided into upper half and lower half. The upper half is implemented as adcinit, adcopen, adcread and adcclose.  The lower half is implemented as an input interrupt handler – adchandler. |
| IoT-specific concerns your design addressed, including but not limited to Energy | Energy Efficiency -   1. To lower energy consumption, the temperature sensor is powered on just before ADC is going to start sampling and powered off when sampling is done. 2. ADC is programmed to generate only one sampling value so that the sensor can be turned off as soon as a value is obtained. 3. Averaging modes can be configured to have a higher level of accuracy or save energy by sampling fewer times.   Latency – WRITE about how we resolved the latency problem. |
| Xinu I/O Interface implementation and testing | **LED driver implementation:**   1. Ledinit – a. This driver function is called during XINU initialization.  b. Ledinit sets the direction of the GPIO pins used for the output. c. Interrupt handler ledhandler is added to the interrupt vector array. 2. Ledwrite –  a. This driver function is used to turn the led on or off. It also sets the color of the led device.  b. Ledwrite sets an interrupt bit to start the output interrupt handler. Ledwrite then waits for space in the output buffer.  c. When the ledhandler signals that space in available in the output buffer, ledwrite is resumed and it sets the output data. 3. Ledhandler - 4. This is the low level device driver function. 5. The ledhandler is invoked by the interrupt dispatcher when the function ledwrite sets the selected output interrupt bit to 1. 6. The interrupt bit is cleared. 7. The ledhandler signals to ledwrite that buffer space is available for writing. 8. When the led handler returns after signal, it uses the value in the output buffer to turn the LED’s on or off.   **Temperature Driver implementation:**   1. adcinit –  a. adcinit does the initial setup for configuring the ADC. The clocks required for ADC are setup.  b. The step register in the ADC is configured to enable step 1 to sample the temperature sensor. All other steps are not enabled. c. The step1 is configured to store data in FIFO0 registers. d. Interrupt FIFO Threshold is enabled. The threshold value is set to 1. i.e. After the ADC generates one sampled value and moves it to FIFO0, an FIFO Threshold Interrupt is generated. e. The GPIO pin that is used to power on the Temperature sensor is enabled. f. The interrupt handler adchandler is added to interrupt vector table. 2. adcopen – a. The clock is checked if it is running and turned on if it is not running. b. The GPIO pin which is used to power the temperature sensor is turned on. c. The ADC is enabled which starts the sampling process. 3. adcclose –  a. The ADC is disabled. b. The GPIO pin is powered down to turn the temperature sensor off. 4. adcread – a. Before reading the value set by the interrupt handler, adcread waits for the interrupt handler to set the temperature value. b. The temperature is calculated using the returned analog voltage reading. 5. adchandler - a. adchandler is the low level input interrupt handler. b. adchandler is invoked when the ADC triggers the interrupt after FIFO0 queue has generated one value. c. The interrupt clear bit for FIFO Threshold c. The analog voltage value is read from the FIFO0 queue. d. The interrupt handler signals the adcread function.   **Common Driver Functions:**   1. ADC  The constants and variables required for ADC driver are included in adc.h and adc.c. 2. GPIO   The common constants and GPIO functions to set pin direction, set pin high and set pin low are included in gpio.h and gpio.c respectively. |
| Design of IoT Description Language, Language processing and code generation | **Indicate**: JSON  **Source**: None.  **Design:**  DDL Specification  The DDL for this application is written in json. Following is the object hierarchy and their properties. All fields with values in quotes are of type string, otherwise integer. Any value can also be "null", without the quotes.  Not all objects will have all possible fields defined for them. This means that those particular fields are not required to describe that particular object.  The numbering in this specific defines the nesting of the objects and their properties.  The DDL parser outputs a set of \*ddl-out files in the /include folder of xinu and then these are directly included in the driver code. |
| Implementation and testing of IoT Description Language, Language processing and code generation | The DDL parser is implemented using python.The DDL file is in json format, and we used json library in python to parse and generate a set of files ending with “ddl-out”. These files are copied in the /include folder and are used in driver code using #include statement.’  Note: the ddl-parser is run by the make rebuild command.  For testing we tested the generated code with C code we had already written beforehand. |
| Implementation and testing of overall on-board driver code (upper- and lower-level drivers, including generated code) | 1. The drivers were unit tested by using print statements and checking the output and register contents of every step. 2. After unit testing was completed, the driver code was tested using the Edge interface and then the cloud interface. All possible combinations of input were sent from the edge to the beagle bone to test the driver 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 | 70%. |
|  |  |
| 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. | Device Externalization is done using RESTful API. |
| 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 address of the Edge is stored in a config file hosted in the cloud.  The address of the Cloud is coded in the edge. Device address of the devices and port are setup using a configuration file.  Device configuration and Initialization are done by the device when the device is started. The cloud requests contain the configuration that is required for the temperature sensor. |
| Give the details of the externalization abstractions design. | Details of the externalization abstractions:-  Cloud sends request to the edge using the following format:-   1. For Temperature: http://[AddressOfEdge]/TempEdge.php?frequency=10&timeout=60&precision=3 2. **For LED:** <http://[AddressOfEdge]/LEDEdge.php?LED=1> 3. Where:    1. [AddressOfEdge] is the IP address or DNS lookup name of the Edge Server.    2. frequency is the frequency in seconds that the temperature should be retreived with.    3. timeout is the number of seconds till which the edge will retreive the temperature.    4. "The Edge will retreive the temperature from the device every 'frequency' seconds for 'timeout' seconds"    5. precision is the sampling rate of the temperature sensor and can have the integer values 1 (lowest) to 3 (highest).    6. LED is defined as 1 for Blue, 2 for Red and 0 for off. |
| 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. | 1. The cloud layer exposes RESTful API through which the edge POSTs the temperature for the edge. 2. The Cloud sends a GET request to Edge layer containing either the temperature request or led request. 3. After processing the request, Edge sends the result/acknowledge back to the cloud through POST request. 4. UDP packets are used for communications between edge layer and device. 5. Edge communicates with the BBB using UDP packets. We chose UDP packets as it is simpler than TCP and does not require a lengthy handshake. 6. The Edge communicates with the BBB with the following 3 types of messages:    1. "GetTemp1": This message asks the BBB to return the temperature value with no averaging.    2. "GetTemp2": This message asks the BBB to return the temperature value with average of 8 values.    3. "GetTemp3": This message asks the BBB to return the temperature value with average of 16 values.    4. "led0": To turn the LED off    5. "led1": To turn on Red light on BBB    6. "led2": To turn on Blue light on BBB |
| Describe your on-board IoT devices Demo App. | **Devices**: Temperature Sensor, RGB LED.  **App:**   1. LED can be turned off, LED color can be set to Red or set to Blue. 2. Temperature Sensor – Temperature is read using three possible sensing modes.  Modes:- Mode1(Low Accuracy, High Energy Efficiency), Mode2 (Intermediate Accuracy and Efficiency) and Mode3 (High Accuracy and Low Energy Efficiency). |
| Describe your web-based IoT devices Demo App. | Our web-based demo app is called Temp Monitor.   1. All temperature values received from the temperature sensor through the edge are displayed on a real-time graph that plots temperature against time. 2. The app allows the user to choose sensing frequency of the temperature sensor. The user can also change the sensor mode to enable power saving features. 3. The app also allows the user to connect an actuator like a LED that is triggered based on threshold temperature. If the temperature goes above the threshold, LED output will be red. Otherwise it will be blue. 4. The user can also directly control the LED output (Red, Blue, Off). |

**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)?

1. The most time consuming part for device drivers was finding out the structure of the ADC status word. Initially, we had no idea that we had to turn the ADC clocks on and the clock structure can change by itself.
2. The most challenging part for edge coding is how to handle multiple requests from cloud for temperature so that only one instance of the edge code is running and communicating with the device. It caused a lot of synchronization issues for us.
3. The most challenging part for cloud layer was to ensure that we get realtime temperature.

**3. Overall Experience**

Overall experience. Describe your overall experience good or bad.

Our experience was good. We spent a lot of time working together and learnt a lot.

**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.