UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Departamento de Ingeniería Electrónica

Técnicas Digitales III

Guía de Trabajos Prácticos

Segundo cuatrimestre

Ciclo lectivo 2021 **Plan 95A**

TABLA DE CONTENIDO

1. INTERACCIÓN CON EL KIT "BEAGLEBONE BLACK"

a) Familiarícese con el hardware de la ***BeagleBone Black (BBB)***. Siga los pasos documentados en la Wiki de la cátedra http://wiki.electron.frba.utn.edu.ar/doku.php?id=td3:bbx

b) Pruebe los ejemplos de código provistos por la cátedra. Interactúe con la misma mediante SSH y SCP.

2. EJERCICIO INTEGRADOR

Objetivo

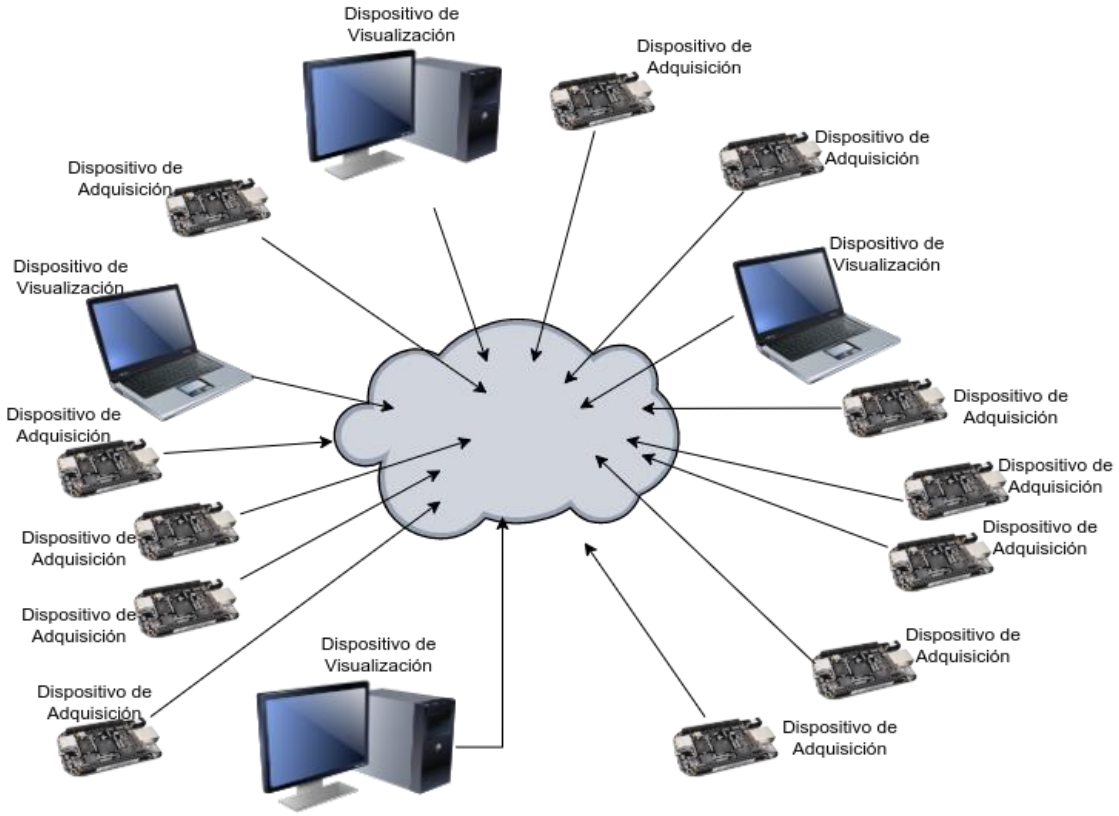
El presente desarrollo tiene por objeto obtener, por encuesta y en forma remota, lecturas de un sensor y visualizarlas en el navegador de una PC.

Requisitos

Se debe adquirir un dispositivo (sensor), acelerómetro y giroscopio, MPU6050. En caso de poseer otro módulo, puede utilizarlo pero la cátedra concentrará su trabajo en el módulo anteriormente citado, con lo cual resultará más rápida la resolución de inconvenientes que puedan producirse en la aplicación del mismo.

Arquitectura

El sistema a implementar debe estar compuesto por una estación de visualización (PC), un dispositivo de adquisición (BeagleBone Black) y el sensor elegido, según se detalla en la siguiente imagen:



Estación de visualización **Dispositivo de adquisición**

Estación de visualización

Esta parte del proyecto solo se limita a hacer solicitudes periódicas al servidor y presentarlo en pantalla. A tal fin se utilizará en una PC una de estas dos soluciones a elección:

a) Una aplicación escrita en Python provista por la cátedra encargada de la visualización de datos y presentación de gráficos en tiempo real, permitiendo seleccionar a cada dispositivo de adquisición a interrogar, mediante su dirección IP.

b) Un navegador de Internet para lo cual se empaquetará la información de acuerdo con el protocolo HTML 1.1. de acuerdo con lo estipulado en el Apéndice I

Dispositivo de adquisición

El dispositivo de adquisición deberá satisfacer las siguientes funcionalidades y requerimientos:

1. Gestionar el dispositivo de medición (sensor), mediante un controlador (*driver*) implementado como un módulo de kernel (*.ko*), acorde con la especificación del Linux Kernel Driver Model (LKDM).
2. El driver deberá implementar la API POSIX (open, close, read, write, ioctl) de un *char device*, gestionar el acceso y configuración del sensor elegido mediante la interfaz serie disponible en el módulo de hardware, configurándolo en modo maestro e inicializar el periférico correspondiente, empleando parámetros de configuración establecidos en el *device tree*.
3. El módulo deberá poder instalarse y desinstalarse en forma dinámica
4. Una vez instalado, el módulo deberá recolectar, procesar y almacenar los datos del dispositivo de medición.
5. Se deberá mantener activo un servidor TCP concurrente que recibirá solicitudes periódicas desde las estaciones de visualización a las que enviará los datos. El servidor deberá cumplir los siguientes requisitos / funcionalidades:

   5.1. ***Archivo de configuración***:
   Archivo de texto plano que contiene las variables de estado del servidor, a razón de una línea por variable cada una de las cuales es un valor numérico en representación ASCII:

   [Cantidad Máxima de Conexiones]. Valor defaut = 100.
   [Cantidad de pedidos de conexión que bufferea (backlog)]. Valor defaut = 20.
   [Tamaño de la ventana del filtro]. Valor default = 2.

   En caso de no existir el archivo de configuración, el servidor utilizará como valor inicial los tres valores señalados como defaut.

   5.2. Cada vez que el servidor concurrente recibe la señal ***SIGUSR2***, relee el archivo de configuración y recarga sus variables de estado con los parámetros allí encontrados de cantidad de conexiones activas, y tamaño de la ventana del filtro. Esta actividad no debe interrumpir la labor de los procesos child, ni afectar el normal funcionamiento del proceso padre. Por tal motivo el parámetro ***backlog*** solo se aplica cuando inicia el server. Nunca se recarga con ***SIGUSR2*** ya que implica reiniciar el server

   5.3. ***Control de conexiones y pedidos de conexión***:
   Limitar la cantidad de pedidos de conexiones simultáneos (en cola o ***backlog***), así como la cantidad máxima de procesos que se creen para atender a los clientes que requieran datos (o sea la cantidad de conexiones activas), de acuerdo con las dos primeras variables en el archivo de configuración.

   5.4. ***Adquisición de datos***
   El manejo del dispopsitivo debe ser por Interrupción de hardware en el módulo de driver. El dispositivo seleccionado (MPU6050) interrumpe cada vez que los sensores entregan su/s valor/es. Tiene además FIFO interno de 1024 bytes en el que puede copiar los valores de los sensores tomados en el mismo instante de tiempo, para que puedan ser leídos en modo burst.

Los datos también pueden ser leídos desde los registros. Pero si se habilita el FIFO interno de 1024 bytes programando al dispositivo durante su inicialización éste trabaja de manera mas eficiente. La organización de los valores dentro del FIFO puede representarse de acuerdo con la documentación mediante una estructura del tipo:

```
struct MPU6050_REGS {
        uint16_t accel_xout;
        uint16_t accel_yout;
        uint16_t accel_zout;
        int16_t temp_out;
        uint16_t gyro_xout;
        uint16_t gyro_yout;
        uint16_t gyro_zout;
};
```

Lst 1. Estructura de datos leídos por los tres sensores

La documentación del dispositivo que se sugiere leer en detalle es: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf , hojas de datos, y https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf , documentación de registros internos.

Siendo que se debe satisfacer las syscalls definidas en el estándar POSIX, donde las funciones ***read ()*** y ***write ()*** reciben un argumento entero en el que se especifica la cantidad de bytes a leer o a escribir, y cuyo rango de valores es cualquier entero positivo, el driver debe manejar esta situación, sin afectar el comportamiento de las syscalls.

De modo que cada lectura del dispositivo físico se tratará como un paquete de 14 bytes, y se almacenará en la FIFO del dispositivo, generando seguidamente una interrupción.

Cada vez que recibe un ***read ()***, el driver deberá determinar si la cantidad de bytes requerida en el argumento de la función, puede satisfacerse con la cantidad de datos disponibles en el FIFO. *El dispositivo tiene la capacidad de leer la cantidad de datos válidos presentes en el FIFO*.

En caso de haber suficientes datos en el FIFO, lee en modo Burst la cantidad requerida en la syscall ***read ()***, y responde el pedido normalmente. En caso contrario, pone el proceso a dormir. Cada vez que recibe una interrupción agreg los datos sensados en el FIFO y verifica si la cantidad requerida se satisface en cuyo caso despierta al proceso para que se proceda a completar el requerimiento.

Se evaluará el criterio de diseño del módulo para resolver como y desde que punto del driver se despierte al proceso para completar la respuesta a ***read ()*** al momento de haberse completado la cantidad suficiente de datos en el FIFO.

Se considerará favorablemente los trabajos que resuelvan la situación planteada en el eventual caso en que la cantidad de datos requerida en el argumento de ***read ()*** supere el tamaño del FIFO. En tal caso el módulo debería alojar esa cantidad de memoria en el Kernel Space y mantener al proceso

dormido hasta que se complete la cantidad de datos solicitada, para recién en ese momento despertar al proceso y enviarle los datos como respuesta a la syscall.

Los datos se interpretarán en el orden definido en el listado Lst 1, sin ningún tipo de separador

5.5. ***Procesamiento digital.***

Aplicar un filtro de media con el tamaño de ventana establecido en el archivo de configuración. Este tamaño se interpreta como la cantidad de muestras anteriores y posteriores a la muestra actual (considere que los tres sensores devuelven por cada lectura 7 muestras de acuerdo con el listado Lst 1).

La función que implementa el filtro de media se escribirá en un archivo fuente separado del resto.

Compilar con -OFast de modo de forzar al compilador a una optimización más agresiva respecto del procesador. De este modo es esperable que el compilador utilice instrucciones SIMD disponibles en el procesador CORTEX-A8.

Además de los programas fuente se pide un documento con el análisis de las instrucciones SIMD (NEON) empleadas por el compilador en esta función. Para ello compilar este archivo fuente con la opción -OFast y -S, esta última generará el código en lenguaje ensamblador. El set de instrucciones NEON se describe en el apéndice C del documento *NEON Programmer's Guide*, versión 1.0, disponible en: https://documentation-service.arm.com/static/5f731b591b758617cd95559c?token=

Analizando la descripción de las instrucciones SIMD involucradas, se pide la descripción del algoritmo empleado con la mirada puesta en las instrucciones vectoriales y como estas eventualmente inciden en la aceleración del mismo.

5.6. Transmisión de datos

Por cada solicitud de los clientes, el servidor transmite a las estaciones de monitoreo el vector de los 7 valores adquiridos por los tres sensores de acuerdo con el orden mostrado en Lst 1, seguido por el vector de los respectivos valores de media, sin ningún separador.

3. PRIMER RECUPERATORIO

Agregar al código del driver un contador del número de lecturas del parámetro medido realizadas desde que se instaló el módulo. El driver deberá retornar el valor del contador a través de ***ioctl***. Al filtro de media, se agrega un algoritmo para estimación de la frecuencia de vibración aplicando una FFT (Ver https://www.scitepress.org/Papers/2017/63409/63409.pdf)

4. SEGUNDO RECUPERATORIO

En caso que un proceso intente abrir el dispositivo cuando éste ya está siendo utilizado, el mismo deberá dormir hasta que se libere el recurso.

Apéndice I

Noción de la comunicación HTTP:

En el marco del protocolo HTTP, el navegador (cliente) enviará una de las siguientes solicitudes GET (***request*** en términos HTTP), cuyas respuestas (***status*** en términos HTTP) por parte del server se especifican en cada caso:

1. Solicitud de contenido:
   Request:
   GET / HTTP/1.1

   Respuesta:
   Si el método es correcto:
   HTTP/1.1 200 Ok
   Content-Type: text/html; charset=UTF-8\n
   Content-Length: %d\n El número es: strlen(HTML)
   \n
   … ***acá escribir el código HTML …***

   Si el método solicitado no es GET:
   HTTP/1.1 400 Bad method
   Si el recurso no es /
   HTTP/1.1 404 Bad resource

2. Solicitud de elemento gráfico:
   Request:
   GET /Archivo-grafico HTTP/1.1

   Respuesta:
   Si el método es correcto:
   HTTP/1.1 200 Ok
   Content-Type: image/png\n
   Content-Length: %d\n El número es: sizeof(buffer_imagen)
   \n
   … acá copiar el gráfico en binario …

   Si el método solicitado no es GET:
   HTTP/1.1 400 Bad method
   Si el recurso no es /Archivo-grafico
   HTTP/1.1 404 Bad resource

3. Para errores 400 o 404:

Content-Type: text/html; charset=UTF-8\n
Content-Length: %d\n El número es: strlen(HTML)
\n
… acá escribir el código HTML que muestre el error en pantalla …