Project #2 – Audio Visualizer

Phillip VanOss

Nick Schrock

July 7, 2016

Dr. Parikh

**Introduction**

Embedded software is often written using the C programming language. C provides a number of basic functions that can be manipulated and targeted towards an embedded system. However, the standard C libraries do not contain every possible driver for every possible peripheral system. To remedy this, custom drivers can be written and are often supplied by an embedded systems provider, such as Texas Instruments. These drivers allow for easy use of the device functionality. The purpose of this project is to write new drivers that build upon the lower level drivers provided. These custom drivers were designed to enable an OLED screen to show the time averaged power of a signal taken in on an analog to digital converter pin.

**Custom Files and Functions**

A number of custom files were written for this application. Each file contains functions that were specifically written to help the system perform as a sound visualizer. In this section each file will be detailed, starting with the drivers and ending with the main file. Within each file description, a short synopsis of each custom function will provided. In this way the entire system can be understood in intricate detail.

**adcdriver.c and adcdriver.h**

These files set up the analog to digital converter peripheral and contain all function definitions pertaining to that peripheral.

* void ADC\_Init(void)
  + This function is to be called when the ADC0 peripheral is set to be initialized. It sets the system clock, enables the ADC port and pin, sets up the ADC sequence, and enables the ADC interrupt.
* void ADC\_IntClear(void)
  + This function clears the interrupt associated with the ADC0 peripheral.
* void ADC\_IntRegister(void (\*pfnHandler)(void))
  + This function takes in a pointer to the function that is to be set as the interrupt handler for the ADC interrupt.
* void ADC\_StartSample(void)
  + This function starts an ADC read event.
* void ADC\_GetSample(unsigned long \*value)
  + This function takes in a pointer to the variable where the result of an ADC read is to be written.

**timerdriver.c and timerdriver.h**

These files set up the timer peripheral and contain all function definitions pertaining to that peripheral.

* void Timer\_Init(void)
  + This function is to be called when the timer peripheral is set to be initialized. It enables the peripheral, configures the peripheral, sets the timer to trigger an interrupt at a frequency of 16 KHz. It then enables the timer interrupts.
* void Timer\_IntRegister(void (\*pfnHandler)(void))
  + This function takes in a pointer to the function that is to be set as the interrupt handler for the timer interrupt.
* void Timer\_IntClear(void)
  + This function clears the interrupt associated with the timer peripheral.

**project2.c**

This file is the “main” file of the project. It utilizes the two drivers described above along with defining a few non-driver related functions.

This section will describe a number of the important variables and then describe the functions included within the file.

* Variables
  + unsigned uAdcFlag
    - This variable is used to let the main program know when an ADC interrupt has occurred.
  + unsigned uTimerFlag
    - This variable is used to let the main program know when a Timer interrupt has occurred.
  + unsigned long ulBuffer1[TWO\_HUNDRED\_MS]
  + unsigned long ulBuffer2[TWO\_HUNDRED\_MS]
    - These variables are arrays capable of holding 200 milliseconds worth of ADC read values. These buffers will be analyzed and the results displayed throughout the program.
  + unsigned uBufferSelect
    - This variable helps the system to determine which of the buffers is to be averaged, and which is to be used for ADC storage.
  + double amplification
    - This variable is the software amplification factor. It allows a weak analog signal to be boosted up to higher levels for easier display purposes.
  + unsigned char array[Y\_LIM\*X\_LIM]
    - This variable is an array of characters the size of the max power bar that holds intensity of the pixels in the power bar.
  + unsigned uTimerCount
    - This variable holds the number of timer interrupt occurrences and is used to determine when 200 milliseconds have passed.
  + unsigned long ulAdcVal
    - This variable holds the ADC value returned after an ADC read event. It is stored in a selected buffer for future manipulation.
  + double dAvgAdcVal
    - This variable holds the averaged value of a buffer for display.
* Functions
  + void ADC\_IsrHandler()
    - This function is called when an ADC interrupt is triggered. It clears the interrupt and sets a flag to let the system know that an interrupt has occurred.
  + void Timer\_IsrHandler()
    - This function is called when a Timer interrupt is triggered. It clears the interrupt and sets a flag to let the system know that an interrupt has occurred. It also starts and ADC read event.
  + double Average\_Buffer(unsigned long\* buffer)
    - This function takes in a pointer to a buffer that is to be averaged. It sums all the elements in the buffer and then divides the sum by the buffer length. It then returns that value.
  + void OLED\_DisplayPowerLvl(double power)
    - This function takes in the averaged power of a buffer and converts it an amplified character array for display on the OLED screen.
  + int main(void)
    - This function is the main function of the program. It runs for the duration of the program and makes calls to the other functions described. It initializes the peripherals along with setting up the appropriate interrupts. It then goes into a continuous loop where the interrupt flags are check. If the uAdcFlag is set, it lowers the flag and gets the sampled ADC value, storing it in ulAdcVal. This value is then stored in a buffer that is selected by the uBufferSelect variable. If the uTimerFlag variable is set, it lowers the flag and adds a value of one to the uTimerCount variable. When the uTimerCount variable has a value that indicates that two hundred milliseconds has passed, its value is reset to zero and the selected buffer is averaged and displayed. After averaging a buffer, the uBufferSelect variable is toggled to set up the sequence to utilize the opposite buffer for the next time.