

# Data communication using single board computers

## A breif Review of the program

Presented by : Prakruti M Bilagi  
2014IEN36  
Central university of Karnataka

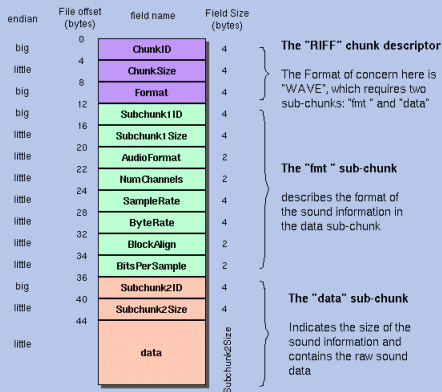
Guide : Dr. G V V Sharma  
Dept of EE  
Indian Institute of Technology Huderabad

# The code package

- *Pcm\_wave\_header.h*
- *Audioformat.h*
- *Error\_reporter.h*
- *Error\_reporter.cpp*
- *Main.cpp*
- *Transmitter.h*
- *Transmitter.cpp*
- *Wave\_reader.h*
- *Wave\_reader.cpp*
- *Makefile*
- *Acaustic\_guiter.wav*

# Pcm\_wave\_header

## *The Canonical WAVE file format*



```
struct PCMWaveHeader
{
    char chunkID[4];
    unsigned chunkSize;
    char format[4];
    char subchunk1ID[4];
    unsigned subchunk1Size;
    unsigned short audioFormat;
    unsigned short channels;
    unsigned sampleRate;
    unsigned byteRate;
    unsigned short blockAlign;
    unsigned short bitsPerSample;
    char subchunk2ID[4];
    unsigned subchunk2Size;
};
```

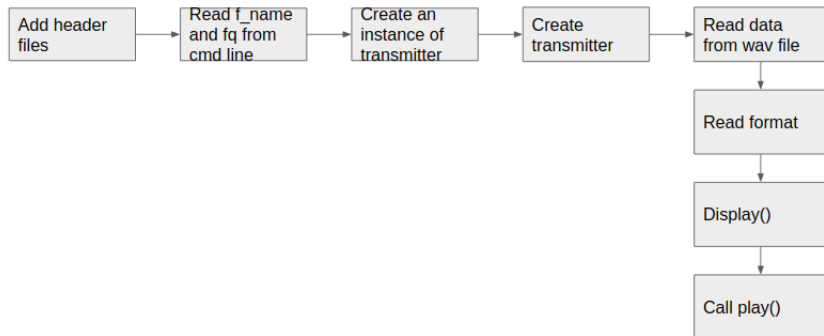
```
struct AudioFormat
{
    unsigned short channels;
    unsigned short bitsPerSample;
    unsigned sampleRate;
};
```

# ErrorReporter.h

```
using std::exception;
using std::string;
class ErrorReporter : public exception
{
    public:
        explicit ErrorReporter(string message);
        virtual ErrorReporter() throw();
        virtual const char* what() const throw();
    protected:
        string errorMessage;
};
```

```
#include "error_reporter.h"
ErrorReporter::ErrorReporter(string message) :
    errorMessage(message)
{
}
ErrorReporter::ErrorReporter() throw()
{
}
const char* ErrorReporter::what() const throw()
{
    return errorMessage.c_str();
}
```

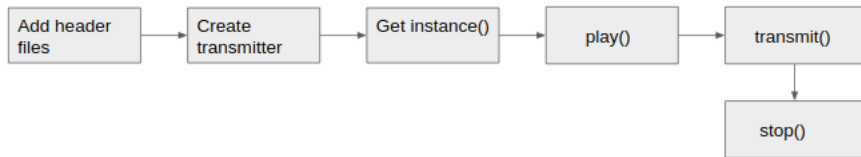
# Main.cpp



# Transmitter.h

```
class Transmitter
{
    public:
        virtual Transmitter();
        void play(string filename, double frequency, bool loop);
        void stop();
        static Transmitter* getInstance();
    private:
        Transmitter();
        bool forceStop, eof;
        static void* peripherals;
        static vectorfloat* buffer;
        static bool transmitting, restart;
        static unsigned frameOffset, clockDivisor;
        static void* transmit(void* params);
};
```





- Include header files
- Assign the base addresses for gpio base clock base clockdividor base and a counter
- Assign initial values for variables to transmitter class members
- Check for the system, type and version of the host system.
- Assign memory for the GPIO pins using memFd and mmap system calls and this memory would be the base for assessing the GPIO and clock etc.

# Transmitter conti:.

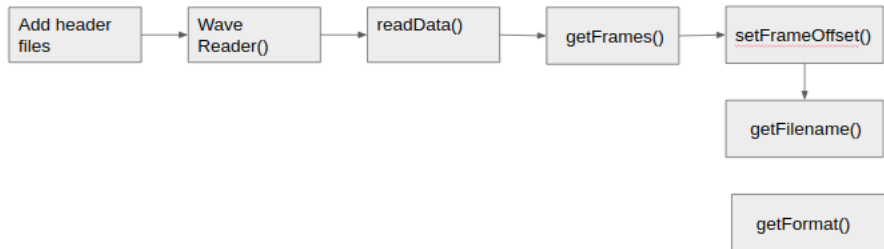
- In the function play()
  - a It checks if the transmitter is already transmitting something.
  - b Creates the objects of class Wave\_reader.(reader)
  - c Grab the format of the reader and store it in "format" variable.
  - d Set the value to the clockDivisor.
  - e Create the bufferFrames that will be stored in the vector "frames"
  - f Create a vector "Frames" and use the fuction getFrames from class reader to stack the bufferFrames into it.
  - g Create thread to start the transmission.
  - h Read the whole data frames one by one.
  - i Set "transmitting" variable "false".
  - j Join the thread.
  - k Delete the reader and format finally while exiting.

# Transmitter conti:.

- Next comes the `transmit()` function where in does the changes in the clock frequency for the transmission purpose.
- a Declares variables to store the current, start and playbackstart positions of the data.
- b Creates unsigned offset, length and temp variables.
- c Creates a vectr of floats to hold the data.
- d Loads the sample-rate into a variable.
- e Assigns the preemp value of 0.734883  $preemp = \frac{0.75 - 2500000.0}{(float)(samplerate * 75)}$
- f Sets the GPIO pin 4 on alternate function-0.
- h Enables the clock-base pin.
  - i The playback starts
  - j Assigning of value and quantize it between -1, value, 1
  - k Set the clockdividor base register.
  - l Increment the counter register.
- m Disable the clock base pin.
  - When the interrupt is generated from the keyboard the the transmission/FM is aborted.

# Wave\_reader.h

```
class WaveReader
{
public:
    WaveReader(string filename, bool forceStop);
    virtual WaveReader();
    AudioFormat* getFormat();
    vector<float>* getFrames(unsigned frameCount, bool forceStop);
    bool setFrameOffset(unsigned frameOffset);
private:
    string filename;
    PCMWaveHeader header;
    unsigned dataOffset, currentOffset;
    int fileDescriptor;
    vector<char> *readData(unsigned bytesToRead, bool
        headerBytes, bool forceStop);
    string getFilename();
};
```



- Includes header files.
- It has a function for WaveReader(), which will read the entire data in the wavefile.
- It has a dataread() which will read the specified bytes from the file and returns the data vector.
- It has a getFrames() function that will read the particular count of frames and return the vector frame.
- Setframeoffset() function will create the shift in the data frames.
- getfilename() is a function that will return the file name of the WAV

# Makefile

As the name suggests it is make file which allows for the execution on this FM on particular processor