Class D Audio Amplifier

James Carter, Electrical Engineering

Project Advisor: Dr. Mitchell

April 25, 2016

Evansville, IN
Table of Contents

I. Introduction

II. Problem Definition
   a. Amplifier Classes
   b. 2.2 Class D Amplifier
   c. 2.3 Design Requirements

III. Solution

IV. Design
   a. Safety
   b. Verification
   c. Equalizer
   d. Protection
   e. Low Pass Filter
   f. Power Losses

V. Budget

VI. Results

VII. Conclusion

List of Figures

Figure 1: 555 Timer and Comparator

Figure 2: Class D Amplifier concepts
Figure 3: Impulse response

Figure 4: Magnitude and phase plot

Figure 5: Potentiometers

Figure 6: DSP Board

Figure 7: Low pass Filter Equations

List of Tables

Table 1: Power loss Equations

Table 2: Budget

I. Introduction
The audio amplifier is used in many devices that we all use every day. Audio signals are very low and need to be amplified to increase part or all of the signal. For the amplifier that is used with subwoofers in cars, the audio amplifier is mainly used to amplify the lower frequencies which create the bass. The more power that is produced by the amplifier means that the speaker receives more power, and the more power that the speaker receives the louder the speaker can get before distortion is noticeable by the listener.

The Class D amplifier is designed to be low cost, smaller, and more efficient than most other amplifier designs. The only issue to deal with is distortion due to the switching of the signal during amplification. The amplifier can create some clicking sounds because the signal is being switched from high to low rapidly. To avoid those sounds a filter is used to ensure better quality of sound for the listener.

II. Problem Definition

a. Amplifier Classes

There are many different types of amplifier designs each with its own advantages and disadvantages. The classifications of amplifiers are Class A, Class B, Class AB, Class G, Class H, Class C, Class D, Class E, Class G2, Class G3, Class F, Class I, Class K, Class S, Class T, and Doherty Amplifier [1], [6].

Class A amplifiers are very inefficient, with the efficiency being less than 20%, however, they do have the advantage of having the least amount of distortion. Class B is the opposite because it is much more efficient but has an extremely high amount of distortion due to the time between transistors on and off state. Class B is most often used in low power applications. The
most popular type of amplifier is class AB. This class takes the best of class A and class B and combines them into one package. This resulting amplifier has a max efficiency of about 78% and low distortion. Class G and class H are both similar in design. The design shifts the voltage up or down to the peaks of the audio signal. The difference between them being that class G uses a square wave of voltage around the audio signal and class H follows the sine curve of the signal. Both are very efficient and are often used in high performance and professional audio amplifier designs. The class D amplifier is the most efficient and is much smaller in physical than other efficient amplifiers such as class AB. The other amplification classes listed are less common designs [1].

b. Class D Amplifier

Power output is important when designing an audio amplifier. The output of the amplifier design will be 1200 watts power. The amplification of the audio signal will be achieved using the class D amplifier model, also known as the switching amplifier. Class D amplifiers were first patented in 1954, and they have a theoretical efficiency of 100%, however in reality they can get just over 90% [1]. The basic idea of the class D amplifier starts with two inputs. The inputs are from the audio signal and usually a triangle wave signal. Both signals are sent into a comparator and then the output is a PWM signal. The signal is then amplified before it is sent through a low pass filter to rid the signal of any high frequencies which are not needed in the audio range. Then finally the audio signal is sent to the speaker to produce an amplified sound wave.

c. Design Requirements
The amplifier is designed with the following criteria:

* Class D amplifier design

  * 1200 watts of power

  * 4 Ohm load

* Digital filter equalizer on output

  * Butterworth design

  * 5 bands

  * 10\textsuperscript{th} order filters

The amplifier is designed using the class D amplification model. The design will be
should be able to produce 1200 watts of continuous power to power 2 subwoofers that are wired
up as a 4 ohm load. The amplifier is also designed with a digital filter equalizer to be set by the
user. Using the filter will allow the user to adjust the music to their liking.

III. Solution

a. Design

The audio amplifier is designed using class D amplification. This process is more
efficient than most other amplification methods. A triangle wave is generated using an LM555
and sent into a comparator with the audio signal and outputs a square wave, which is then sent
through gate drivers. The gate drivers output to transistors, the purpose of these gate driver and
transistors are for switching [1]. The gate drivers are also used to increase the voltage that is sent into the MOSFET’s because the output of the modulation circuit will produce low voltages [3]. The audio signal is switched at high frequencies, usually 250 kHz for the full audio bandwidth [1]. The next stage of the amplifier uses a transformer to increase the PWM signal even more. The single is boosted to 100 volts to achieve the correct power output for the design. The audio signal is then sent through a low pass filter which returns the signal back into an audio file. The

![Figure 1: This shows the triangle wave generator from the 555 Timer](image1)

![Figure 2: This shows the concept of a Class D amplifier](image2)
The picture below shows an example of the input and output of the modulation stage of a class D amplifier [4].

The outside design is just as important as the inside of the amplifier. The amplifier must function, however it must also have its own appearance to make it unique. The case is compact and able to fit inside the trunk of a car. It would be very impractical if the case were to be larger for a few reasons. It would be hard to put it inside smaller cars and it does not need to take up that much space.

b. Safety

Safety is important in any design. Many machines or electronic devices focus on functionality and cost first. Safety is the last thing considered in most cases. For this design the first step is safety since the design involves high voltage and high current. When connecting to the battery supply in the car for the amplifier it is always important to remember that there is a fuse on the power line before the capacitor. This fuse is to limit the amount of current that can go into the amplifier and also to protect the user, too much current could melt the wires. The next thing is to use correct wiring for the load. The input needs to have 4 gauge wire or bigger. The amount of current is high enough that when the amplifier operates at peak capacity it may melt the wire is it is smaller. Since the input is high in current and the output is much lower in current the output wiring does not need to be as big as the input wiring. Most speaker output wires use 14 gauge wire, however for this design 10 gauge wire is used also, for safety the ground must be stable and secured to the metal of the car somewhere. In most cases the user must use sandpaper to scrape off the paint inside the trunk of their car first and then drill a hole to mount the ground
wire. In many newer cars that have the battery located in the trunk, the battery is grounded to the car, so if would be fine to mount the ground to the battery. Double check everything before applying any power if something is wrong the components could get hot and possibly catch fire. Before the amplifier is ready for use I will test it to ensure that it is safe to use.

c. Verification

To verify that it works and is reaching the power desired an oscilloscope is used to show the output voltage. With the output voltage the output power can be calculated with this equation \( P = \frac{V^2}{R} \), where \( V \) is the voltage, \( P \) is the output power and \( R \) is the load which will be 4 ohms. Using this equation it can be shown that the amplifier works as designed.

d. Equalizer

The audio signal is sent through a digital filter equalizer. The digital filter is programmed using C and loaded onto a microcontroller. The filter can be set by the user to achieve the sound that is desired. The filter design is designed using the Butterworth method. The equalizer has 5 bands and each band has an adjustable gain. The equalizer is split up into high and low. The first 2 bands on the low side are combined and sent out as one output to the

Figure 3: Impulse response of First filter in the Equalizer
subwoofers. The other 3 bands are all combined as the high output and sent out to a different output which can be connected to any other speakers in the car.

The equalizer was designed in Matlab using the fdatool function. This tool simulates the filter, checks stability of the filter, and generates graphs and other data about the filter. This data is used to ensure that the filter meets all requirements. From this data the coefficients that were needed to write the program in C were given. The equalizer bands are made up of a low pass filter up to 80 Hz, a band pass filter from 80Hz to 400 Hz, another band pass filter from 400Hz to 3 kHz, a 3rd band pass filter from 3kHz to 10kHz, and finally a high pass filter from 10kHz up. The fdatool showed the filter in a graph, Figure 3 graph of the impulse response for the first band of the filter, and Figure 4 shows the magnitude and phase plot. The equalizer was implemented on an ARM processor DSP board. Figure 6 is a picture of the board used. Figure 5 is the 5 potentiometers connected to the input of the equalizer.
e. **Protection**

To avoid any damage to the components while the amplifier is in operation, the design will also include some protection circuits. A sensor is used to ensure that the circuit does not overheat, and current protection to ensure that the transistors are not damaged due to a high current [4]. The current protection circuits are necessary because fuses that block high AC currents would still allow high DC currents through which could damage the circuitry. With the protection the design will also need some LED indicators that blink when something is faulting out. The amplifier will be designed with 2 types of protection. The protection types are DC and transient speaker protection and soft-start circuit protection. The DC and transient protection will use capacitors and diodes to block the DC current which will protect the speaker from damage. The soft-start protection will be used to limit high current surges by using a combinations of diodes, resistors and capacitors [7].

f. **Low Pass Filter**
The low pass filter design is a 3rd order Butterworth filter designed using the equations below [2]. The results of these equations are

\[ L_1 = 3.8197 \times 10^{-5} \text{H}, \]

\[ C_1 = 2.1221 \times 10^{-6} \text{F}, \]

and \[ L_2 = 1.2732 \times 10^{-5} \text{H}. \] These were all calculated using a value of \( f_c \) equal to 25 kHz, and a value of 4 for \( R_L \). This filter design should filter out the high carrier frequencies from the modulation section of the amplifier.

### g. Power Loss

The power loss should be low in this amplifier design. The theoretical power loss is 0%, however the practical design there is some loss. The power losses can be calculated with the following equations: [5]

\[ \text{P}_{\text{Total Loss}} = \text{P}_{\text{Switching}} + \text{P}_{\text{GATE}} + \text{P}_{\text{COND}} \]

<table>
<thead>
<tr>
<th># of Order</th>
<th>( L_1 )</th>
<th>( C_1 )</th>
<th>( L_2 )</th>
<th>( C_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.414514</td>
<td>0.707107</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1.5000</td>
<td>1.3333</td>
<td>0.5000</td>
<td>0.3333</td>
</tr>
<tr>
<td>4</td>
<td>1.530734</td>
<td>1.577161</td>
<td>1.082392</td>
<td>0.332583</td>
</tr>
</tbody>
</table>

Power losses are equal to the power from the switching transistors and the drivers.
Switching power is the switching energy multiplied by the switching frequency

\[ P_{\text{switching}} = E_{\text{sw}} \times f_{\text{sw}} \]

\[ E_{\text{sw}} \int_0^T V_{DS}(t) \times I_D(t) \, dt \]

\( T \) is the length of the switching pulse

\[ P_{\text{GATE}} = 2 \times Q_g \times V_{\text{driver}} \times f_{\text{sw}} \]

The power loss from the transistor

\[ P_{\text{COND}} = R_{DS(ON)} / R_L \times P_O \]

\( P_{\text{Total Loss}} \) based on values from the data sheets of components is 100 watts

### IV. Budget

All the components needed for this amplifier design were listed below in the table.

<table>
<thead>
<tr>
<th>Part</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCB Board</td>
<td>$50</td>
</tr>
<tr>
<td>Comparator</td>
<td>$1</td>
</tr>
<tr>
<td>Transformer</td>
<td>$50</td>
</tr>
<tr>
<td>MOSFET’s</td>
<td>$5</td>
</tr>
<tr>
<td>MOSFET drivers</td>
<td>$2</td>
</tr>
<tr>
<td>LM555 Timer</td>
<td>$1</td>
</tr>
<tr>
<td>Microcontroller for the digital filter</td>
<td>$20</td>
</tr>
<tr>
<td>LED’s</td>
<td>$5</td>
</tr>
<tr>
<td>Component</td>
<td>Cost</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2F Capacitor</td>
<td>$30</td>
</tr>
<tr>
<td>Case</td>
<td>$10</td>
</tr>
<tr>
<td>4 Gauge wire kit with fuse 60 Amp fuse</td>
<td>$35</td>
</tr>
<tr>
<td>Toroidal Core inductors</td>
<td>$2</td>
</tr>
<tr>
<td>Other Components (resistors, capacitors, diodes, other parts)</td>
<td>$15</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$226</strong></td>
</tr>
</tbody>
</table>

Many of the components were purchased online from Mouser. Some parts were cheaper than expected. For example, the case was much cheaper than expected because it was not built in house. It was designed to be clear to see everything inside, with Plexiglas sheets on the top and bottom. Other parts were already owned such as the wires and 2F capacitor.

V. Results

The Project was expected to be able to amplify an audio signal to 1200 watts of output power, however due to some hardware issues it did not. Somewhere in the circuit too much current was being drawn from the 555 timer. Because of this, the timer fried every time the next stage of amplification was connected. Although the amplifier part of the project did not work, the equalizer did. There were 5 sliding potentiometers which were used to change the audio signal on each input. All 5 of the inputs changed the 2 outputs as designed.
VI. Conclusion

Although this did completely do what it was designed to do, there was a lot of information gained from working on this design. There are many different types of amplifiers and each of them has different advantages and disadvantages. The main advantage that class D has is power efficiency. With high power efficiency, class D amplifiers work well in any mobile application. Other Classes of amplifiers are also used in cars for subwoofers, but for the cost and size class D works well. The other part of the design which was the equalizer came with some challenges but they were not too difficult because the digital filter equalizer worked.

Appendix A

Equalizer C Code

//Implements 5 10th order Butterworth filter using second order sections.

//The coefficients were generated with the fdatool in MATLAB.

#include "stm32f407vg.h"

// first filter LP 80 Hz

//First Section

const float bA0 = 1;

const float bA1 = 2;

const float bA2 = 1;
const float aA0 = 1;
const float aA1 = -1.996310639060943;
const float aA2 = 0.99644032264008231;

//Second Section

const float bB0 = 1;
const float bB1 = 2;
const float bB2 = 1;
const float aB0 = 1;
const float aB1 = -1.9895750238886925;
const float aB2 = -0.98970426966699832;

//Third Section

const float bC0 = 1;
const float bC1 = 2;
const float bC2 = 1;
const float aC0 = 1;
const float aC1 = -1.9838810416608386;
const float aC2 = 0.98400991754951672;

//Fourth Section
const float bD0 = 1;
const float bD1 = 2;
const float bD2 = 1;
const float aD0 = 1;
const float aD1 = -1.979764523832248;
const float aD2 = 0.97989312885650526;

//Fifth Section
const float bE0 = 1;
const float bE1 = 2;
const float bE2 = 1;
const float aE0 = 1;
const float aE1 = -1.977607181125502;
const float aE2 = 0.97773564945478786;

// Second filter 80Hz to 400 Hz

//First Section
const float bA01 = 1;
const float bA11 = 0;
const float bA21 = -1;
const float aA01 = 1;
const float aA11 = -1.9738502793249773;
const float aA21 = 0.97694345800874494;

//Second Section
const float bB01 = 1;
const float bB11 = 0;
const float bB21 = -1;
const float aB01 = 1;
const float aB11 = -1.9950340521100296;
const float aB21 = 0.99516853403613015;

//Third Section
const float bC01 = 1;
const float bC11 = 0;
const float bC21 = -1;
const float aC01 = 1;
const float aC11 = -1.9833741294109437;
const float aC21 = 0.98356081013827135;
//Fourth Section
const float bD01 = 1;
const float bD11 = 0;
const float bD21 = -1;
const float aD01 = 1;
const float aD11 = -1.9422240416353387;
const float aD21 = 0.94440334719672225;

//Fifth Section
const float bE01 = 1;
const float bE11 = 0;
const float bE21 = -1;
const float aE01 = 1;
const float aE11 = -1.9547811626698761;
const float aE21 = 0.9554163378333872;

// Third filter 400 Hz to 3000 Hz

//First Section
const float bA011 = 1;
const float bA111 = 0;
const float bA211 = -1;
const float aA011 = 1;
const float aA111 = -1.9738502793249773;
const float aA211 = 0.97694345800874494;

//Second Section
const float bB011 = 1;
const float bB111 = 0;
const float bB211 = -1;
const float aB011 = 1;
const float aB111 = -1.9950340521100296;
const float aB211 = 0.99516853403613015;

//Third Section
const float bC011 = 1;
const float bC111 = 0;
const float bC211 = -1;
const float aC011 = 1;
const float aC111 = -1.9833741294109437;
const float aC211 = 0.98356081013827135;

//Fourth Section
const float bD011 = 1;
const float bD111 = 0;
const float bD211 = -1;
const float aD011 = 1;
const float aD111 = -1.9422240416353387;
const float aD211 = 0.94440334719672225;

//Fifth Section
const float bE011 = 1;
const float bE111 = 0;
const float bE211 = -1;
const float aE011 = 1;
const float aE111 = -1.9547811626698761;
const float aE211 = 0.9554163378333872;

// Fourth filter 3000 Hz to 10000 Hz

//First Section
const float bA0111 = 1;
const float bA1111 = 0;
const float bA2111 = -1;
const float aA0111 = 1;
const float aA1111 = -0.277309245545412;
const float aA2111 = 0.68851232922403383;

//Second Section
const float bB0111 = 1;
const float bB1111 = 0;
const float bB2111 = -1;
const float aB0111 = 1;
const float aB1111 = -1.6807760516117929;
const float aB2111 = 0.85376536862250596;

//Third Section
const float bC0111 = 1;
const float bC1111 = 0;
const float bC2111 = -1;
const float aC0111 = 1;
const float aC1111 = -1.3782068079539149;
const float aC2111 = 0.36902541140163841;

//Fourth Section
const float bD0111 = 1;
const float bD1111 = 0;
const float bD2111 = -1;
const float aD0111 = 1;
const float aD1111 = -0.43246384576206498;
const float aD2111 = 0.32732608547737435;

//Fifth Section
const float bE0111 = 1;
const float bE1111 = 0;
const float bE2111 = -1;
const float aE0111 = 1;
const float aE1111 = -0.88606282914838375;
const float aE2111 = 0.29485794577107349;

// Fifth filter 10000 Hz to 20000 Hz
//First Section

const float bA01111 = 1;
const float bA11111 = -2;
const float bA21111 = 1;
const float aA01111 = 1;
const float aA11111 = -0.25203166874535599;
const float aA21111 = 0.73194769118719849;

//Second Section

const float bB01111 = 1;
const float bB11111 = -2;
const float bB21111 = 1;
const float aB01111 = 1;
const float aB11111 = -0.20083278536138557;
const float aB21111 = 0.38011179568383174;

//Third Section

const float bC01111 = 1;
const float bC11111 = -2;
const float bC21111 = 1;
const float aC01111 = 1;
const float aC11111 = -0.17124139038429187;
const float aC21111 = 0.17676136569732132;

//Fourth Section
const float bD01111 = 1;
const float bD11111 = -2;
const float bD21111 = 1;
const float aD01111 = 1;
const float aD11111 = -0.15468243733888809;
const float aD21111 = 0.062969156018938363;

//Fifth Section
const float bE01111 = 1;
const float bE11111 = -2;
const float bE21111 = 1;
const float aE01111 = 1;
const float aE11111 = -0.14719913870392951;
const float aE21111 = 0.011544341598589122;

int main()
{

int uInt, yInt, yout1, yout2, yout3, yout4, yout5, out1, out2;

float u, yA, yB, yC, yD, yE;

float u1, u2;

float yA1, yA2, yB1, yB2, yC1, yC2, yD1, yD2, yE1, yE2;

int uInt1, yInt1;

float u11, yA11, yB11, yC11, yD11, yE11;

float u111, u21;

float yA111, yA21, yB111, yB21, yC111, yC21, yD111, yD21, yE111, yE21;

int uInt11, yInt11, uInt111, yInt111, uInt1111, yInt1111;

float u1111, u211;

float u11111, u2111;

float yA11111, yA211, yB11111, yB2111, yC11111, yC2111, yD11111, yD2111, yE11111, yE2111;

float yA111111, yA21111, yB111111, yB21111, yC111111, yC21111, yD111111, yD21111, yE111111, yE21111;

float u111111, u21111;

float yA1111111, yA211111, yB1111111, yB211111, yC1111111, yC211111, yD1111111, yD211111, yE1111111, yE211111;
//Clock bits

RCC_AHB1ENR |= 1; //Bit 0 is GPIOA clock enable bit

RCC_APB1ENR |= (1 << 29); //Bit 29 is DAC clock enable bit

RCC_APB2ENR |= 0x100; //Bit 8 is ADC 1 clock enable bit

RCC_APB1ENR |= (1 << 4); //Enable peripheral timer for timer 6

//I/O bits

GPIOA_MODER |= 0x4000; //Bits 15-14 = 01 for digital output on PA7

//OTYPER register resets to 0 so it is push/pull by default

GPIOA_OSPEEDER |= 0xC000; //Bits 15-14 = 11 for high speed on PA7

//PUPDR defaults to no pull up no pull down

GPIOA_MODER |= 0xFF00; //PA4-PA5 are analog

GPIOB_MODER = 0x0F; //PB0-1 are analog

GPIOC_MODER = 0x3; //PC0 is analog

GPIOA_PUPDR &= 0xFFFFF0FF; //Pins PA4 PA5 are no pull up and no pull down

GPIOB_PUPDR &= 0xFFFFFFFF0; //Pins PB0-1 are no pull up and no pull down

GPIOC_PUPDR &= 0xFFFFFFFFFC; //Pin PC0 is no pull up or pull down

//DAC bits

DAC_CR |= 0x3E; //Bits 3, 4, 5 = 111 for software trigger ch1
// Bit 2 = 1 for Ch 1 trigger enabled

// Bit 1 = 1 for Ch 1 output buffer enabled

DAC_CR |= 1; // Bit 0 = 1 for Ch 1 enabled

DAC_CR |= 0x3E0000; // Same bits for ch 2

DAC_CR |= 0x10000;

// ADC bits

ADC_SQR3 |= 0xFFFFFFFFE0; // Bits 4:0 to 0

ADC_SQR3 |= 0x6; // Bits 4:0 are channel number for conversion

ADC_SQR3 |= 0x7;

ADC_SQR3 |= 0x8;

ADC_SQR3 |= 0x9;

ADC_SQR3 |= 0x10;

ADC_CR2 |= 1; // Bit 0 turn ADC on

ADC_CR2 |= 0x400; // Bit 10 allows EOC to be set after conversion

ADC_CCR |= 0x30000; // Bits 16 and 17 = 11 so clock divided by 8

// Channel is set to 5 which corresponds to PA5

// Timer 6 bits
TIM6_CR1 |= (1 << 7); //Auto reload is buffered

TIM6_CR1 |= (1 << 3); //One pulse mode is on.

TIM6_PSC = 0; //Don't use prescaling

TIM6_ARR = (363); //(16 MHz)/(363) = 44.1kHz

TIM6_CR1 |= 1; //Enable Timer 6

//Main program loop

while(1)
{

    // First filter

    ADC_CR2 |= 0x40000000; //Bit 30 does software start of A/D conversion

    while((ADC_SR & 0x2) == 0); //Bit 1 is End of Conversion

    uInt = ADC_DR;

    u = ((float)(uInt & 0xFFF))/(float)4095.0;

    yA = (bA0 * u) + (bA1 * u1) + (bA2 * u2) - (aA1 * yA1) - (aA2 * yA2); yA /= aA0; //First section

    yB = (bB0 * yA) + (bB1 * yA1) + (bB2 * yA2) - (aB1 * yB1) - (aB2 * yB2); yB /= aB0; //Second section

    yC = (bC0 * yB) + (bC1 * yB1) + (bC2 * yB2) - (aC1 * yC1) - (aC2 * yC2); yC /= bB0; //Third section
yD = (bD0 * yC) + (bD1 * yC1) + (bD2 * yC2) - (aD1 * yD1) - (aD2 * yD2); yD /= bC0; //Fourth section

yE = (bE0 * yD) + (bE1 * yD1) + (bE2 * yD2) - (aE1 * yE1) - (aE2 * yE2); yE /= bD0; //Fifth section

//yInt = (int)(2048*y); //Data to D/A

yInt = (int)(2900*(yE+1));
yout1 = yInt;
u2 = u1; u1 = u;
yA2 = yA1; yA1 = yA;
yB2 = yB1; yB1 = yB;
yC2 = yC1; yC1 = yC;
yD2 = yD1; yD1 = yD;
yE2 = yE1; yE1 = yE;

// second filter

ADC_CR2 |= 0x40000000; //Bit 30 does software start of A/D conversion

while((ADC_SR & 0x2) == 0); //Bit 1 is End of Conversion
uInt1 = ADC_DR;

u11 = ((float)(uInt1 & 0xFFF))/(float)4095.0;

yA11 = (bA01 * u11) + (bA11 * u111) + (bA21 * u21) - (aA11 * yA111) - (aA21 * yA21); yA1 /= aA01;  //First section

yB11 = (bB01 * yA1) + (bB11 * yA11) + (bB21 * yA21) - (aB11 * yB111) - (aB21 * yB21); yB1 /= aB01; //Second section

yC11 = (bC01 * yB1) + (bC11 * yB11) + (bC21 * yB21) - (aC11 * yC111) - (aC21 * yC21); yC1 /= bB01; //Third section

yD11 = (bD01 * yC1) + (bD11 * yC11) + (bD21 * yC21) - (aD11 * yD111) - (aD21 * yD21); yD1 /= bC01; //Fourth section

yE11 = (bE01 * yD1) + (bE11 * yD11) + (bE21 * yD21) - (aE11 * yE111) - (aE21 * yE21); yE1 /= bD01; //Fifth section

//yInt = (int)(2048*y); //Data to D/A

yInt1 = (int)(2900*(yE1+1));

yout2 =yInt1;

out1 = yout1 +yout2; // add filters together

DAC_DHR12R1 = out1 & 0xFFF; //Converted number to D/A

DAC_SWTRIGR |= 0x1; //Start the D/A conversion
u21 = u11; u11 = u1;

yA21 = yA11; yA11 = yA1;

yB21 = yB11; yB11 = yB1;

yC21 = yC11; yC11 = yC1;

yD21 = yD11; yD11 = yD1;

yE21 = yE11; yE11 = yE1;

// Third filter

ADC_CR2 |= 0x40000000; //Bit 30 does software start of A/D conversion

while((ADC_SR & 0x2) == 0); //Bit 1 is End of Conversion

uInt11 = ADC_DR;

u111 = ((float)(uInt11 & 0xFFF))/(float)4095.0;

yA1111 = (bA011 * u11) + (bA111 * u111) + (bA211 * u211) - (aA111 * yA1111) - (aA211 * yA211); yA11 /= aA011;    //First section

yB111 = (bB011 * yA11) + (bB111 * yA111) + (bB211 * yA211) - (aB111 * yB1111) - (aB211 * yB211); yB11 /= aB011; //Second section

yC111 = (bC011 * yB11) + (bC111 * yB111) + (bC211 * yB211) - (aC111 * yC1111) - (aC211 * yC211); yC11 /= bB011; //Third section
\[ y_{D111} = (b_{D011} \cdot y_{C11}) + (b_{D111} \cdot y_{C111}) + (b_{D211} \cdot y_{C211}) - (a_{D111} \cdot y_{D1111}) - (a_{D211} \cdot y_{D211}); y_{D11} /= b_{C011}; //Fourth section \]

\[ y_{E111} = (b_{E011} \cdot y_{D11}) + (b_{E111} \cdot y_{D111}) + (b_{E211} \cdot y_{D211}) - (a_{E111} \cdot y_{E1111}) - (a_{E211} \cdot y_{E211}); y_{E11} /= b_{D011}; //Fifth section \]

//yInt = (int)(2048*y); //Data to D/A

\[ y\text{Int}11 = (\text{int})(2900*(y_{E11}+1)); \]

\[ \text{yout3} = y\text{Int}11; \]

u211 = u111; u111 = u11;

yA211 = yA111; yA111 = yA11;

yB211 = yB111; yB111 = yB11;

yC211 = yC111; yC111 = yC11;

yD211 = yD111; yD111 = yD11;

yE211 = yE111; yE111 = yE11;

// Fourth filter

ADC_CR2 |= 0x40000000; //Bit 30 does software start of A/D conversion

while((ADC_SR & 0x2) == 0); //Bit 1 is End of Conversion
uInt111 = ADC_DR;

u1111 = ((float)(uInt111 & 0xFFF))/(float)4095.0;

yA11111 = (bA0111 * u1111) + (bA1111 * u11111) + (bA2111 * u21111) -
(aA1111 * yA11111) - (aA2111 * yA2111); yA1111 /= aA0111;   //First section

yB11111 = (bB0111 * yA111) + (bB1111 * yA1111) + (bB2111 * yA2111) -
(aB1111 * yB11111) - (aB2111 * yB2111); yB1111 /= aB0111;  //Second section

yC11111 = (bC0111 * yB111) + (bC1111 * yB1111) + (bC2111 * yB2111) -
(aC1111 * yC11111) - (aC2111 * yC2111); yC1111 /= bB0111;  //Third section

yD11111 = (bD0111 * yC111) + (bD1111 * yC1111) + (bD2111 * yC2111) -
(aD1111 * yD11111) - (aD2111 * yD2111); yD1111 /= bC0111;  //Fourth section

yE11111 = (bE0111 * yD111) + (bE1111 * yD1111) + (bE2111 * yD2111) -
(aE1111 * yE11111) - (aE2111 * yE2111); yE1111 /= bD0111;  //Fifth section

//yInt = (int)(2048*y); //Data to D/A

yInt111 = (int)(2900*(yE1111+1));

yout4 = yInt111;

u2111 = u111; u1111 = u111;

yA2111 = yA1111; yA1111 = yA111;
yB2111 = yB1111; yB1111 = yB111;
yC2111 = yC1111; yC1111 = yC111;
yD2111 = yD1111; yD1111 = yD111;
yE2111 = yE1111; yE1111 = yE111;

// Fifth Filter
ADC_CR2 |= 0x40000000; //Bit 30 does software start of A/D conversion
while((ADC_SR & 0x2) == 0); //Bit 1 is End of Conversion
uInt1111 = ADC_DR;
u111111 = ((float)(uInt1111 & 0xFFF))/(float)4095.0;
yA111111 = (bA01111 * u111111) + (bA11111 * u111111) + (bA21111 * u211111) - (aA11111 * yA111111) - (aA21111 * yA211111); yA111111 /= aA01111; //First section

yB111111 = (bB01111 * yA111111) + (bB11111 * yA111111) + (bB21111 * yA211111) - (aB11111 * yB111111) - (aB21111 * yB211111); yB111111 /= aB01111; //Second section

yC111111 = (bC01111 * yB111111) + (bC11111 * yB111111) + (bC21111 * yB211111) - (aC11111 * yC111111) - (aC21111 * yC211111); yC111111 /= bB01111; //Third section
yD11111 = (bD01111 * yC11111) + (bD11111 * yC11111) + (bD21111 * yC21111) - (aD11111 * yD11111) - (aD21111 * yD21111); yD11111 /= bC01111; //Fourth section

yE11111 = (bE01111 * yD11111) + (bE11111 * yD11111) + (bE21111 * yD21111) - (aE11111 * yE11111) - (aE21111 * yE21111); yE11111 /= bD01111; //Fifth section

//yInt = (int)(2048*y); //Data to D/A

yInt1111 = (int)(2900*(yE11111+1));

yout5 = yInt1111;

out2 = yout3 + yout4 + yout5; //adding filters together

u21111 = u1111; u11111 = u1111;

yA21111 = yA11111; yA11111 = yA11111;

yB21111 = yB11111; yB11111 = yB11111;

yC21111 = yC11111; yC11111 = yC11111;

yD21111 = yD11111; yD11111 = yD11111;

yE21111 = yE11111; yE11111 = yE11111;

DAC_DHR12R2 = out2 & 0xFFF; //Converted number to D/A

DAC_SWTRIGR |= 0x2; //Start the D/A conversion
while((TIM6_CR1 & 1) != 0); //Wait here until timer runs out

    TIM6_CR1 |= 1; //Restart timer

}
#define GPIOA_OSPEEDER (*((volatile unsigned long *) 0x40020008))  //GPIO A Speed register

#define GPIOA_OTYPER (*((volatile unsigned long *) 0x40020004))  //GPIO A Output type register

#define GPIOA_IDR (*((volatile unsigned long *) 0x40020010))  //GPIO A Input Data register

#define GPIOA_ODR (*((volatile unsigned long *) 0x40020014))  //GPIO A Output Data register

#define GPIOA_BSRR (*((volatile unsigned long *) 0x40020018))  //GPIO A Output Bit set/reset reg

#define GPIOA_AFRL (*((volatile unsigned long *) 0x40020020))  //GPIO A Alt Funct reg bits 0-7

#define GPIOA_AFRH (*((volatile unsigned long *) 0x40020024))  //GPIO A Alt Funct reg bits 8-15

//GPIO Port B Addresses

#define GPIOB_MODER (*((volatile unsigned long *) 0x40020400))  //GPIO B Mode Reg

#define GPIOB_PUPDR (*((volatile unsigned long *) 0x4002040C))  //GPIO B Pull Up/Pull Dn Reg

#define GPIOB_OSPEEDER (*((volatile unsigned long *) 0x40020408))  //GPIO B Speed register
#define GPIOB_OTYPER (*((volatile unsigned long *) 0x40020404)) //GPIO B Output type register

#define GPIOB_IDR (*((volatile unsigned long *) 0x40020410)) //GPIO B Input Data register

#define GPIOB_ODR (*((volatile unsigned long *) 0x40020414)) //GPIO B Output Data register

#define GPIOB_BSRR (*((volatile unsigned long *) 0x40020418)) //GPIO B Output Bit set/reset reg

#define GPIOB_AFRL (*((volatile unsigned long *) 0x40020420)) //GPIO B Alt Funct reg bits 0-7

#define GPIOB_AFRH (*((volatile unsigned long *) 0x40020424)) //GPIO B Alt Funct reg bits 8-15

//GPIO Port C Addresses

#define GPIOC_MODER (*((volatile unsigned long *) 0x40020800)) //GPIO C Mode Reg

#define GPIOC_PUPDR (*((volatile unsigned long *) 0x4002080C)) //GPIO C Pull Up/Pull Dn Reg

#define GPIOC_OSPEEDER (*((volatile unsigned long *) 0x40020808)) //GPIO C Speed register

#define GPIOC_OTYPER (*((volatile unsigned long *) 0x40020804)) //GPIO C Output type register
#define GPIOC_IDR (*((volatile unsigned long *) 0x40020810))   //GPIO C Input Data register

#define GPIOC_ODR (*((volatile unsigned long *) 0x40020814))   //GPIO C Output Data register

#define GPIOC_BSRR (*((volatile unsigned long *) 0x40020818))   //GPIO C Output Bit set/reset reg

#define GPIOC_AFRL (*((volatile unsigned long *) 0x40020820))   //GPIO C Alt Funct reg bits 0-7

#define GPIOC_AFRH (*((volatile unsigned long *) 0x40020824))   //GPIO C Alt Funct reg bits 8-15

//Clock addresses

#define RCC_CR (*((volatile unsigned long *) 0x40023800))   //Clock Control Register

#define RCC_PLLCFGR (*((volatile unsigned long *) 0x40023804))   //PLL Config Register

#define RCC_CFGR (*((volatile unsigned long *) 0x40023808))   //Clock Config Register

#define RCC_CIR (*((volatile unsigned long *) 0x4002380C))   //Clock Interrupt Register

#define RCC_APB1ENR (*((volatile unsigned long *) 0x40023840))   //DAC Periph Clock Enable Reg
#define RCC_AHB1ENR (*((volatile unsigned long *) 0x40023830))   //GPIO Enable Reg

#define RCC_APB2ENR (*((volatile unsigned long *) 0x40023844))   //ADC Periph Clock Enable Reg

//Flash Memory

#define FLASH_ACR (*((volatile unsigned long *) 0x40023C00))   //Flash Access Control Register

//A to D Addresses

#define ADC_SR (*((volatile unsigned long *) 0x40012000))      //ADC Status Register

#define ADC_CR1 (*((volatile unsigned long *) 0x40012004))     //ADC Control Register 1

#define ADC_CR2 (*((volatile unsigned long *) 0x40012008))     //ADC Control Register 2

#define ADC_SQR1 (*((volatile unsigned long *) 0x4001202C))     //ADC Regular Seq Register 1

#define ADC_SQR2 (*((volatile unsigned long *) 0x40012030))     //ADC Regular Seq Register 2

#define ADC_SQR3 (*((volatile unsigned long *) 0x40012034))     //ADC Regular Seq Register 3

#define ADC_DR (*((volatile unsigned long *) 0x4001204C))      //ADC Data Register

#define ADC_CCR (*((volatile unsigned long *) 0x40012304))     //ADC Common Control Register
//Timer 2 addresses

#define TIM2_CR1 (*((volatile unsigned long *) 0x40000000))   //Timer 2 Control Reg 1
#define TIM2_CR2 (*((volatile unsigned long *) 0x40000004))   //Timer 2 Control Reg 2
#define TIM2_SR (*((volatile unsigned long *) 0x40000010))   //Timer 2 Status Reg
#define TIM2_DIER (*((volatile unsigned long *) 0x4000000C))   //Timer 2 Interrupt Enable
#define TIM2_EGR (*((volatile unsigned long *) 0x40000014))   //Timer 2 Event Generation Reg
#define TIM2_CNT (*((volatile unsigned long *) 0x40000024))   //Timer 2 Count Reg
#define TIM2_PSC (*((volatile unsigned long *) 0x40000028))   //Timer 2 Prescale Reg
#define TIM2_ARR (*((volatile unsigned long *) 0x4000002C))   //Timer 2 Auto-Reload Reg

//Timer 3 addresses

#define TIM3_CR1 (*((volatile unsigned long *) 0x40000400))   //Timer 3 Control Reg 1
#define TIM3_CR2 (*((volatile unsigned long *) 0x40000404))   //Timer 3 Control Reg 2
#define TIM3_SR (*((volatile unsigned long *) 0x40000410))   //Timer 3 Status Reg
#define TIM3_DIER (*((volatile unsigned long *) 0x4000040C))   //Timer 3 Interrupt Enable
//Timer 3 Event Generation Reg

#define TIM3_EGR (*((volatile unsigned long *) 0x40000414))   //Timer 3 Event Generation Reg

#define TIM3_CNT (*((volatile unsigned long *) 0x40000424))   //Timer 3 Count Reg

#define TIM3_PSC (*((volatile unsigned long *) 0x40000428))   //Timer 3 Prescale Reg

#define TIM3_ARR (*((volatile unsigned long *) 0x4000042C))   //Timer 3 Auto-Reload Reg

//Timer 6 addresses

#define TIM6_CR1 (*((volatile unsigned long *) 0x40001000))   //Timer 6 Control Reg 1

#define TIM6_CR2 (*((volatile unsigned long *) 0x40001004))   //Timer 6 Control Reg 2

#define TIM6_SR (*((volatile unsigned long *) 0x40001010))    //Timer 6 Status Reg

#define TIM6_DIER (*((volatile unsigned long *) 0x4000100C))  //Timer 6 Interrupt Enable

#define TIM6_CNT (*((volatile unsigned long *) 0x40001024))   //Timer 6 Count Reg

#define TIM6_PSC (*((volatile unsigned long *) 0x40001028))   //Timer 6 Prescale Reg

#define TIM6_ARR (*((volatile unsigned long *) 0x4000102C))   //Timer 6 Auto-Reload Reg

#define TIM6_EGR (*((volatile unsigned long *) 0x40001014))   //Timer 6 Event Generation Reg

//Interrupt controller addresses
#define NVICISER0 (*((volatile unsigned long *) 0xE000E100))   //Interrupt Enable 0-31
#define NVICISER1 (*((volatile unsigned long *) 0xE000E104))   //Interrupt Enable 32-63
#define NVICICER0 (*((volatile unsigned long *) 0xE000E180))   //Interrupt Clear Enable 0-31
#define NVICICER1 (*((volatile unsigned long *) 0xE000E184))   //Interrupt Clear Enable 32-63

//USART1 Registers

#define USART1_SR   (*((volatile unsigned long *) 0x40011000))   //USART 1 Status Register
#define USART1_DR   (*((volatile unsigned long *) 0x40011004))   //USART 1 Data Register
#define USART1_BRR  (*((volatile unsigned long *) 0x40011008))  //USART 1 Baud Rate Register
#define USART1_CR1  (*((volatile unsigned long *) 0x4001100C))  //USART 1 Control Register 1
#define USART1_CR2  (*((volatile unsigned long *) 0x40011010))  //USART 1 Control Register 2
#define USART1_CR3  (*((volatile unsigned long *) 0x40011014))  //USART 1 Control Register 3
#define USART1_GTPR (*((volatile unsigned long *) 0x40011018)) //USART 1 Gd Time & Prescale Register

//USART4 Registers
#define USART4_SR (*((volatile unsigned long *) 0x40004C00))   //USART 4 Status Register
#define USART4_DR (*((volatile unsigned long *) 0x40004C04))   //USART 4 Data Register
#define USART4_BRR (*((volatile unsigned long *) 0x40004C08))  //USART 4 Baud Rate Register
#define USART4_CR1 (*((volatile unsigned long *) 0x40004C0C))  //USART 4 Control Register 1
#define USART4_CR2 (*((volatile unsigned long *) 0x40004C10))  //USART 4 Control Register 2
#define USART4_CR3 (*((volatile unsigned long *) 0x40004C14))  //USART 4 Control Register 3
#define USART4_GTPR (*((volatile unsigned long *) 0x40004C18)) //USART 4 Gd Time & Prescale Register

//USART6 Registers
#define USART6_SR (*((volatile unsigned long *) 0x40011400))   //USART 6 Status Register
#define USART6_DR  (*((volatile unsigned long *) 0x40011404))  //USART 6 Data Register

#define USART6_BRR  (*((volatile unsigned long *) 0x40011408))  //USART 6 Baud Rate Register

#define USART6_CR1  (*((volatile unsigned long *) 0x4001140C))  //USART 6 Control Register 1

#define USART6_CR2  (*((volatile unsigned long *) 0x40011410))  //USART 6 Control Register 2

#define USART6_CR3  (*((volatile unsigned long *) 0x40011414))  //USART 6 Control Register 3

#define USART6_GTPR (*((volatile unsigned long *) 0x40011418)) //USART 6 Gd Time & Prescale Register

References


