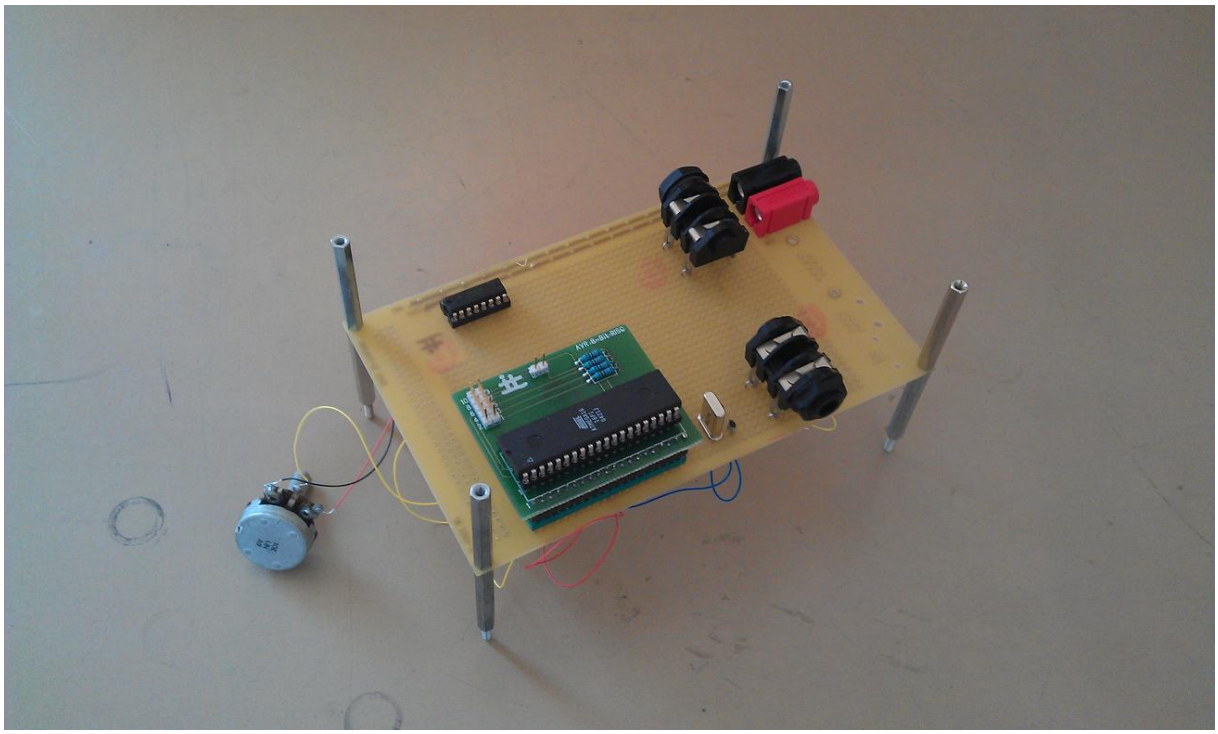
Effektpedal för elgitarr



Handledare: Bertil Lindvall

Ivan Rimac (I05)
Jimmy Lundberg (I08)

2011-05-10

Contents

Bakgrund

Detta projekt ingår i kursen EITF11 ”Digitala projekt”. Under sju veckors tid ska vi jobba med ett digitalt projekt som vi själva väljer.

Vi har valt ett projekt som går ut på att ta fram en prototyp av en effektpedal för elgitarr. Pedalen ska ta in analogt ljud och behandla den digitalt för att sedan skicka ut analogt ljud med en förhöjd frekvens. Frekvenshöjningens storlek beror på läget hos pedalen. Frekvensen på insignalen från gitarren är ca 80-1300 Hz.

I detta projekt kommer vi endast att kunna ta fram en väldigt primitiv prototyp. Tid, pengar och kunskap är begränsad och detta kommer i sin tur att påverka kvalitén på slutresultatet. Syftet med projektet är därför i huvudsak att vi ska lära oss hur man genomför ett mindre digitalt projekt och få ökad kunskap om de tekniker som används i denna typ av projekt.

Kravspecifikation

Kravspecifikation – ”Effektpedal”

Pedalen ska ta en analog insignal från en elgitarr och beroende på pedalens läge ska pedalen skicka ut samma signal eller en med högre frekvens.

Utsignalen ska höja frekvensen på insignalen med 0 – 4 ggr.

För att göra detta behövs A/D-omvandlare för insignalen från gitarren, D/A-omvandlare för utsignalen från pedalen till en förstärkare. Ett sätt att mäta pedalens läge, samt omvandla detta värde till en digital signal (då den inte redan är det).

Systemet innehåller:

1 mikroprocessor (AVR atmega16)

2 insignaler, gitarrens ljud, samt pedalens läge

1 utsignal, ljud som ska skickas till en förstärkare

2 kontakter från gitarr input och output till förstärkare

1 JTAG för att föra över vårt program till mikroprocessorn

Outputens kvalitet avgörs av frekvensen samt upplösningen på A/D- och D/A-omvandlarna.

För att testa systemet behöver vi någon form av tongenerator.

Vi måste även kunna mäta utsignalen från systemet.

Mål

Avläsning i 5 kHz (sampling 10 kHz), i mån av tid bättre.

Användning

Systemet är tänkt att användas som en gitarr-effektpedal där den utgör en tredje komponent mellan en elgitarr och dess förstärkare. För att använda pedalen kopplar man in gitarrsladden i gitarren och i input-kontakten i pedalkomponenten. En likadan sladd kopplas mellan förstärkaren och pedalkomponentens output-kontakt. Sedan ska förstärkaren sättas på så att man kan höra ljudet som genereras. Systemet kan nu användas.

Nu kan man ändra ljudets frekvens genom att ändra ”pedalens” läge, d.v.s. skruva på potentiometern. Vid noll-läget (potentiometern är skruvad så långt som möjligt åt ena hållet) så ger pedalen ingen frekvenshöjning alls, utan ljudet skickas bara vidare in i förstärkaren (men med sämre kvalitet p.g.a. samplingen). När man sedan skruvar på potentiometern så ökar ljudets frekvens, ju mer man skruvar desto högre frekvens. När pedalen når sitt max-läge (potentiometern är skruvad så långt som går åt andra hållet) så ökas frekvensen maximalt, vilket innebär att output-ljudet har ca 4 ggr högre frekvens än input-ljudet. I max-läget är också ljudkvalitén som sämst, vilket beror på begränsningar i hårdvaran.

Hårdvara

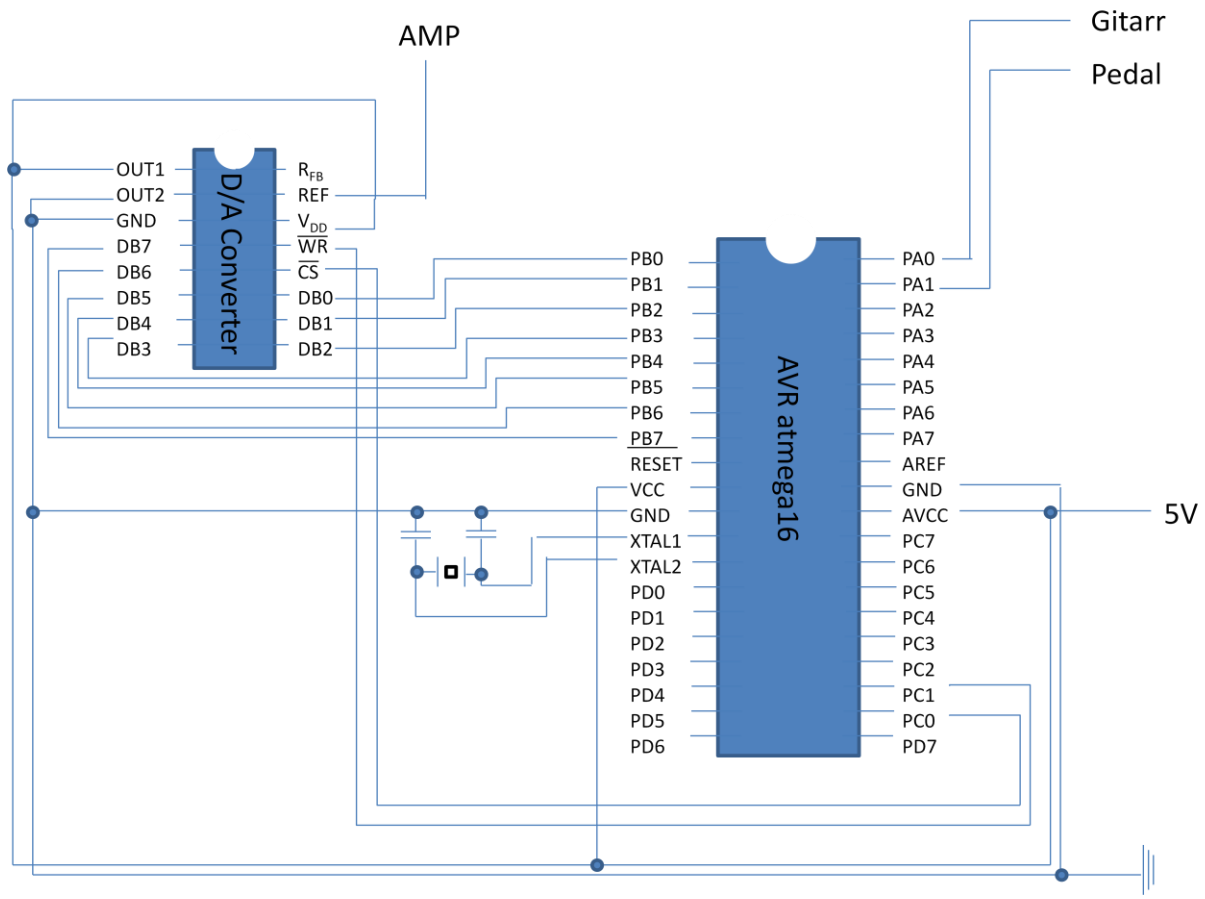
Komponentlista:

- Microcontroller Atmel AVR Atmega16
- TLC7524 8-bit D/A Converter
- Potentiometer (linjär, 10k)
- Atmel JTAG ICE
- Kristall
- Kontakter, TS hona (6.3 mm), 2 st.
- Två kapacitatorer

Vi använder oss utav processorn ”Atmel AVR Atmega16”, eftersom den har många funktioner färdiga i systemet som gör den relativt lättanvänd för en nybörjare. Denna processor har en inbyggd ADC och en klockfrekvens på upp till 16 MHz. Till denna kopplades också en DAC, ”TLC7524 8-bit D/A Converter”. Vi kopplade sedan en potentiometer (linjär, 10k) till AVR:n för att kunna skicka insignaler till den inbyggda ADC:n. För att kunna få upp processorns klockfrekvens till 16 MHz använde vi en extern klocka som består av en kristall och två kapacitatorer. Vi använde oss utav ”Atmel JTAG ICE” för att kunna föra över vårt c-program från datorn till AVR:n. Systemet använder också två kontakter av typen ”TS hona, 6.3 mm” där analog input från gitarren skickas in och analog output skickas vidare till gitarrens förstärkare.

Kopplingsschemat nedan visar precis hur alla komponenter är kopplade till varandra.

Figur 1. Kopplingsschema



Mjukvara

Här följer en beskrivning av mjukvaran som används av systemet. Den fullständiga källkoden finns i appendix A.

Nedanstående konstanter och variabler är tillgängliga för alla funktioner i programmet:

```
#define sampSize 256
unsigned char samples[sampSize];
unsigned char sampPos = 0;
unsigned long outPos = 0;
unsigned char pedalTimer = 0;
unsigned char pedalRatio = 10;
unsigned char ADCchan = 0;
unsigned char pedal = 0;
```

Programmet utgörs av följande funktioner:

```
void startADCconversion();
void portInit();
void ADCInit();
void ADCRun();
void selectADCChanel(unsigned char chan);
void sendOutput();
void saveSample(unsigned char s);
ISR(ADC_vect);
void guitarInputInit();
```

```
int main();
```

Programmet gör analog till digital omvandling av värden från gitarren och lägger resultatet i en ringbuffert, samples. Vi representerar ringbufferten med en vanlig array, de funktioner som använder bufferten måste själv hålla koll på buffertens storlek och vilken possition man försöker läsa ifrån. sampSize anger buffertens storlek.

```
#define sampSize 256
unsigned char samples[sampSize];
```

Då D/A omvandlaren omvandlar maximalt 8-bitars tal omvandlar vi endast till 8 bitar, för att spara värdena används datatypen unsigned char.

sampPos är den plats som nästa lästa värde från gitarren ska läggas på. outPos är den plats som nästa ut-värde till D/A omvandlaren ska hämtas ifrån.

```
unsigned char sampPos = 0;
unsigned long outPos = 0;
```

Pedalens läge läses av efter pedalRatio antal värden har lästs till bufferten och skrivits ut till D/A omvandlaren. pedalTimer används för att avgöra om det är dags att uppdatera pedalens värde som sparas i pedal. ADCchan används för att indikera vilken kanal som A/D omvandlaren läser ifrån.

```
unsigned char pedalTimer = 0;
unsigned char pedalRatio = 10;
unsigned char pedal = 0;
unsigned char ADCchan = 0;
```

portInit() initierar vilka ben på portarna som ska användas. Alla ben på port B kopplas till D/A omvandlaren DB0-DB7. PC0 är kopplad till CS och PC1 till WR på D/A omvandlaren; och initieras båda till 1. PA0 är kopplad till gitarren, och PA1 till pedalen.

```
void portInit();
```

ADCInit() och ADCRun initierar A/D omvandlaren, att endast 8 bitar ska sparas, att hela värdet ska sparas i ADCH, vilken klockhastighet som ska användas, att det genereras avbrott när en omvandling är klar, och att globala avbrott är tillåtna. Avslutningsvis anropar funktionen startADCconversion() som startar den första omvandlingen.

```
void ADCInit();
void ADCRun();
void startADCconversion();
```

guitarInputInit() nollställer bufferten.

```
void guitarInputInit();
```

selectADCChanel() byter vilken kanal A/D omvandlaren läser ifrån. Kanal 0 är kopplad till gitarren, och kanal 1 till pedalen. Då det endast finns 8 kanaler på ADC'n ignoreras värden större än 7.

```
void selectADCChanel(unsigned char chan);
```

sendOutput() beräknar utifrån pedal vilket som är nästa element i bufferten som ska skrivas ut. Detta värde läggs på port B. För att utföra D/A omvandlingen skickas värdet noll till CS på DAC'n via PC0, sedan noll till WR (via PC1) dessa värden återställs sedan i omvänd ordning. På WR's uppåtflank utförs omvandlingen. sendOutput anropas från ISR(ADC_vect) när ADCchan är noll.

```
void sendOutput();
```

saveSample() anropas från ISR(ADC_vect) när ADCchan är noll. Till funktionen skickas ADCH, som läggs i bufferten.

```
void saveSample(unsigned char s);
```

ISR(ADC_vect) är en avbrottsrutin som anropas när en A/D omvandling är klar. Om ADCchan är noll var omvandlingen från gitarren och saveSample() och sendOutput ska skickas. Annars om ADCchan är ett har pedalens läge avlästs och pedal ska uppdateras.

```
ISR(ADC_vect);
```

main() kör init funktionerna och lägger sig sedan i en tom oändlig loop.

```
int main();
```

Resultat

Resultatet av vårt projekt blev en effektpedal som fungerar som vi tänkt oss, dock med en del brus i ljudsignalen.

Målet med sampling i 10 kHz uppnåddes precis, enligt en grov bedömning av outputen med hjälp av ett oscilloskop. För att ta reda på samplingsfrekvensen kopplade vi inputen till en tongenerator och outputen till ett oscilloskop. På oscilloskopet observerade vi att outputen hade en period på 100 mikrosekunder vilket innebär en samplingsfrekvens på 10 kHz.

Om man har en bättre ADC, som är snabbare och framför allt har mindre brus i sin signal kan outputen förbättras. I nuläget utnyttjas ADC på AVR:en maximalt, och på grund av tidsbrist har vi inte testat en separat ADC kopplad till AVR:en.

Med tanke på den hårdvara som användes och den begränsade tid vi lade på projektet, är kvalitén på slutresultatet bättre än vi förväntade oss. Vi är alltså mycket nöjda över resultatet.

Appendix A – Källkod

```
#include <avr/io.h>
#include <avr/interrupt.h>
/*
PA0 är input från gitarr
PA1 är pedalens läge
PC0 är kopplad till CS på DAC'n
PC1 är kopplad till WR på DAC'n
PB0-PB7 är kopplade till DB0-DB7 på DAC'n
PD0 används av catch-all interrupt rutinen för felmedelande
*/

#define sampSize 256
unsigned char samples[sampSize];
unsigned char sampPos = 0;
unsigned long outPos = 0;
unsigned char pedalTimer = 0;
unsigned char pedalRatio = 10;
unsigned char ADCchan = 0;
unsigned char pedal = 0;

void startADCconversion(){
		ADCSRA |= (1 << ADSC);  // Start A2D Conversions
}
void portInit(){
		DDRB = 0xFF;
		DDRC = 0x3;
		PORTC = 3;	//00000011 "nollställer" de inverterade CS och WR benen på DAC'en

}
void ADCInit(){
		//ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // Set ADC prescaler to 128 -
125KHz sample rate @ 16MHz
		ADCSRA &= 0xF8; //ADCSRA &= 11111000

		ADMUX |= (1 << REFS0); // Set ADC reference to AVCC
		ADMUX |= (1 << ADLAR); // Left adjust ADC result to allow easy 8 bit reading
		ADMUX |= 0; // välj kanal 0 på ADC'n detta är default
}
void ADCRun(){
		//ADCSRA |= 0x20;  // Set ADC to Free-Running Mode
		//SFIOR &= ~(0xE0);		// också free-running mode
		ADCSRA |= (1 << ADEN);  // Enable ADC
		ADCSRA |= (1 << ADIE);  // Enable ADC Interrupt
		sei();   // Enable Global Interrupts

		startADCconversion();
}
```

```
void selectADCChanel(unsigned char chan){
		if(chan < 8){
				ADMUX &= ~(7);
				ADMUX |= chan;
		}
}
void sendOutput(){
		PORTB = samples[outPos/100];
		outPos = (outPos + 100 + 10*pedal/7) % (sampSize*100);
		PORTC = 2;	//00000010 => noll skickas till CS på DAC'en
		PORTC = 0;	//00000000 => noll skickas till WR på DAC'en
		//sleep(?);	//Det verkar inte som om man behörver vänta på DAC'en
		PORTC = 2;	//Uppåtflank på WR
		PORTC = 3;	//Uppåtflank på CS
}
void saveSample(unsigned char samp){
		samples[sampPos] = samp;
		sampPos = (++sampPos) % sampSize;
}
ISR(ADC_vect)
{
		if(ADCchan == 0){
				saveSample(ADCH);
				sendOutput();
				pedalTimer = (++pedalTimer) % pedalRatio;
				if(pedalTimer == 0){
						ADCchan = 1;
						selectADCChanel(ADCchan);
				}
		}else{
				pedal = ADCH;
				ADCchan = 0;
				selectADCChanel(ADCchan);
		}
		startADCconversion();
}
void guitarInputInit(){
		for(unsigned char i = 0; i < sampSize; ++i){
				samples[i] = 0;
		}
}
int main(){
		portInit();
		ADCInit();
		ADCRun();

		for(;;){}

		return 0;
}
```