Nicolae-Vlad ILIE (78639) - Procesor de efecte

Autorul poate fi contactat la adresa: **Login pentru adresa**

Introducere

Ce face?

Microcontroller-ul va prelua semnalul de la chitara, il va modifica/memora daca este cazul si il va da mai departe la iesire.

Care este scopul lui?

Scopul proiectului este realizarea unui de procesor de efecte. Functionalitatea principala este implementarea unui looper. Un looper inregistreaza semnalul generat de chitara intre doua momente de timp si apoi il reda la nesfarsit. Peste aceasta secventa se pot suprapune in continuare altele. (mai multe detalii aici : https://www.quora.com/What-exactly-is-a-looping-pedal-and-what-does-it-do + un exemplu practic : https://www.youtube.com/watch?v=W1_N1aJwzck). Pe langa functionalitatea de baza, procesorul va putea modifica semnalul primit de la chitara generand urmatoare tipuri de efecte : distortion, tremolo, delay, etc..

Ideea de baza de la care am pornit si utilitatea proiectului

Necesitatea unui looper si a unor efecte destul de comune la un pret accesibil care pot fi reprogramate de catre utilizator.

Descriere generală

Schema bloc

Hardware Design

Lista de piese

Nume piesa	Numar piese	Obs
Placa de bază cu ATMega324A-PU	1	
Led-uri	5	Folosite pentru identificare efectului curent
Resistente	3 x 500k, 1 x 2k	Folosite pentru implementarea filtrelor trece sus respectiv trece jos
Condensatoare	2 x 470nF	Folosite pentru implementarea filtrelor pentru excluderea frecventelor nedorite
Mufa Jack 6.3mm Mama	1	https://electroniclight.ro/mf1401-jack-mama-63mm-mono/2907.htm
DAC MCP4725 cu interfata I2C (TWI)	1	https://www.optimusdigital.ro/ro/altele/1327-modul-dac-mcp4725-cu-interfaa-i2c.html
Butoane	4	https://electroniclight.ro/mf4308-push-buton-6x6x7mm/1722.htm
Potentiometru 10k	1	https://electroniclight.ro/potentiometru-mono-10k/2412.htm
Convertor USB la UART	1	https://www.optimusdigital.ro/ro/interfata-convertoare-usb-la-serial/591-convertor-ch340g-la-uart.html (debug)

Scheme electrice

Vor exista 4 butoane : accentuare, diminuare efect curent, down si up. Cu down si cu up se va umbla prin meniul de efecte vizibil prin intermediul celor 5 leduri, iar cu butoanele de accentuare si diminuare efect se va modifica forma efectului curent.

Obtinerea semnalului: am folosit ADC-ul integrat in AtMega324 in modul diferential deoarece semnalul electric generat de dozele chitarii electrice se afla in intervalul (-300mV, 300mV). Pentru excluderea frecventelor nedorite am adaugat un filtru trece sus ce va blocheaza frecventele care sunt sub ~80Hz.

Comunicarea dintre ATMega324 si convertorul digital analog se realizeaza folosind protocolul Two-Wire Interface (TWI), standardul I2C. In acest two-wire interface masterul este ATMega324 care

prin pinul PC0 (SCL) va impune bitrate-ul iar prin pinul PC1 (SDA) sunt trimise datele catre slave (DAC).

Semnalul analogic dat de DAC este trecut printr-un filtru trece jos ce exclude frecventele ce trec de pragul de ~11kHz. Potentiometrul formeaza un divizor rezistiv intre GND si OUT pentru a controla volumul.

Software Design

Platform : Linux

Env : Sublime

Setup : pm-2018-setup

Control butoane

```
  Pentru identificarea input-ului de la utilizator folosesc intreruperile
PCINT1, mai precis PCINT8, PCINT9, PCINT10, PCINT11 pentru pinii PB0, PB1,
PB2 si PB3.
  In rutina pentru tratarea intreruperii verific ce buton a fost folosit si
modific efectul curent folosit si amplificarea lui.
  /* button interrupts */
  if ((PINB & _BV(PB0)) == 0)
  {
      /* move to next efect */
      effect_type = (effect_type + 1) % MAX_EFFECTS;
  }
  if ((PINB & _BV(PB1)) == 0)
  {
      /* move to previous effect */
      effect_type =
          (effect_type + MAX_EFFECTS - 1) % MAX_EFFECTS;
  }
  if ((PINB & _BV(PB2)) == 0)
  {
      /* increase amplif */
      if(effect_type == 1){
          es_distortion_depth += es_distortion_step;
      }else if(effect_type == 2){
          /* compression */
          es_compression_alpha_release += es_compression_step;
          es_compression_alpha_attack += es_compression_step;
      }else if(effect_type == 3){
```

```
          /* tremolo */
          es_tremolo_depth += es_tremolo_step;
      }else if(effect_type == 4){
          /* echo */
          es_echo_depth += es_echo_step;
      }
  }
  if ((PINB & _BV(PB3)) == 0)
  {
      /* decrease amplif */
      if(effect_type == 1){
          /* distortion */
          es_distortion_depth -= es_distortion_step;
      }else if(effect_type == 2){
          /* compression */
          es_compression_alpha_release -= es_compression_step;
          es_compression_alpha_attack -= es_compression_step;
      }else if(effect_type == 3){
          /* tremolo */
          es_tremolo_depth -= es_tremolo_step;
      }else if(effect_type == 4){
          /* echo */
          es_echo_depth -= es_echo_step;
      }
  }
```

Configurare ADC

```
  /* Initialize ADC */
  ADMUX = 0
        /* Measure between ADC1 - ADC0
         * If differential channels are selected,
         * only 2.56V should be used as Internal Voltage Reference
       */
      /* Set internal 2.56V Voltage Reference with external capacitor at
AREF pin */
      | _BV(REFS1)
      | _BV(REFS0)
      /* Measure ADC1 - ADC0 with 10x gain */
      /* try more settings - 200x gain */
      | 0b01001;
  ADCSRA = 0
      /* Activate ADC interrupt */
      | _BV(ADIE)
      /* Activate ADC Auto Triggering */
      | _BV(ADATE)
      /* Set ADC Prescaler to 64 */
      /* try more settigs - 128 or 32 */
```

```
    /* 128 - 0b111, 32 - 0b101 */
    | 0b110;
/* ADC Free Running Mode - Auto Trigger
 * you don't need to set ADSC for the next conversion
 */
ADCSRB = 0;
/* Disable digital input buffer for ADC pins (ADC0, ADC1)
 * to reduce power consumption in the digital input buffer
 */
DIDR0 = 0b00000011;
/* Enable ADC */
ADCSRA |= _BV(ADEN);
/* Start first ADC conversion */
ADCSRA |= _BV(ADSC);
```

TWI

```
Implementarea protocolului din documentatia MCP4725.
/* fast mode I2C
 * writing uint16_t data to dac reg
 */
uint8_t write_DAC(uint16_t data)
{
    /* init */
    TWI_start();
    if (TWI_status() != 0x08)
        return 1;

    /* send first byte */
    TWI_write(TWI_1st_BYTE);
    if (TWI_status() != 0x18)
        return 1;

    /* send first 8 bytes */
    TWI_write((uint8_t)((data >> 8) & 0x0F));
    if (TWI_status() != 0x28)
        return 1;

    /* send the other half */
    TWI_write((uint8_t)data);
    if(TWI_status() != 0x28)
        return 1;

    /* end */
    TWI_stop();

    return 0;
```

```
  }
```

Procesarea semnalului

Pentru procesarea audio mi-a fost mult mai la indemana sa folosesc float-uri. Semnalul primit din adc a fost convertit la float si apoi este normalizat la intervalul [-1,1]. Am ales fac asta deoarece majoritatea documentatiei legata de procesare audio foloseste sample-uri din intervalul respectiv. Am implementat urmatoarele efecte : Distortion, Compression, Tremolo si Echo. La compilare am folosit flag-urile -O3 -ffast-math.

```
  /* ADC conversion complete handler */
  ISR(ADC_vect)
  {
      /* gather from ADC */
      read_adc = ADC;

      /* Twos Complement converter */
      if (read_adc & _BV(9)) {
          //negative number
          read_adc = read_adc | ~(_BV(10) - 1);
      }

      /* process normalized signal [-1, 1] */
      output_dac = (int16_t)(process_signal(((float)read_adc) / SGN_MAX) *
SGN_MAX);
      /* used for delay and echo */
      OUTPUT_HISTORY_ADD(output_dac);

      /* amplif signal in his range */
      output_dac = output_dac + SGN_MAX;

      /* send to output through dac */
      DAC_output(output_dac);
  }
```

In functia process_signal aplic efectul dorit si modific starea led-urilor. Dupa aplicarea efectului dorit sample-ul curent este limitat la intervalul [-1,1].

```
  float process_signal(float x)
  {
      /* set the desire effect */

      /* clear */
      if(effect_type == 0){
          CLEAR_LEDS();
          LED1_UP();
      }
```

```
    /* compression */
    if (effect_type == 1){
        CLEAR_LEDS();
        LED2_UP();
        x = apply_compression(x);
    }
    /* distortion */
    if (effect_type == 2){
        CLEAR_LEDS();
        LED3_UP();
        x = apply_distortion(x);
    }
    /* tremolo */
    if (effect_type == 3){
        CLEAR_LEDS();
        LED4_UP();
        x = apply_tremolo(x);
    }
    /* echo */
    if (effect_type == 4){
        CLEAR_LEDS();
        LED5_UP();
        x = apply_echo(x);
    }

    /* adjust signal, keep it in range */
    return adjust_amplitude(x);
}
```

Pentru controlul led-urilor am folosit urmatoarele:

```
/* Clear all leds */
#define CLEAR_LEDS() do{ \
    PORTC &= ~_BV(PC2); \
    PORTC &= ~_BV(PC3); \
    PORTC &= ~_BV(PC4); \
    PORTC &= ~_BV(PC5); \
    PORTC &= ~_BV(PC6); \
    } while(0);

/* light up led1 */
#define LED1_UP() do { \
    PORTC |= _BV(PC6);  \
    } while(0);
```

Distortion

```
float apply_distortion(float x)
```

```
 {
     return (1-es_distortion_depth) * x + es_distortion_depth *
tanh(es_distortion_gain * x);
 }
```

Compression

```
  float apply_compression(float x)
  {
      const float slope = 1.0f / 5.0f;
      static float compressor_gain = 1.0f;
      static float state = 0.0f;

      const float max_abs = fmax(x, es_compression_kmin);
      const float max_abs_dB = log(max_abs);
      const float overshoot = max_abs_dB - es_compression_knee_treshold;
      const float rect = fmax(overshoot, 0.0f);
      const float cv = rect * slope;
      const float prev_state = state;

      if (cv <= state)
          state = es_compression_alpha_attack * state + (1 -
es_compression_alpha_attack) * cv;
      else
          state = es_compression_alpha_release * state + (1 -
es_compression_alpha_release) * cv;

      compressor_gain *= exp(state - prev_state);

      return x * compressor_gain;
  }
```

Tremolo

Implica combinarea semnalului de la chitara cu semnalul unui oscilator sinusoidal.M-am folosit de timer1 initilizat asftel :

```
  /* initialize timer1 to be used with the tremolo effect */
  TCCR1A = 0;
  TCCR1B = 0
      | _BV(WGM12)
      | _BV(CS12)
      | _BV(CS10);

  /* settings */
```

```
  OCR1A = TIMER_1HZ;
  float apply_tremolo(float x)
  {
      float modulation = sin((2 * M_PI / OCR1A) * TCNT1);
      modulation = (1 - es_tremolo_depth) + es_tremolo_depth * modulation *
modulation;
      return x * modulation;
  }
```

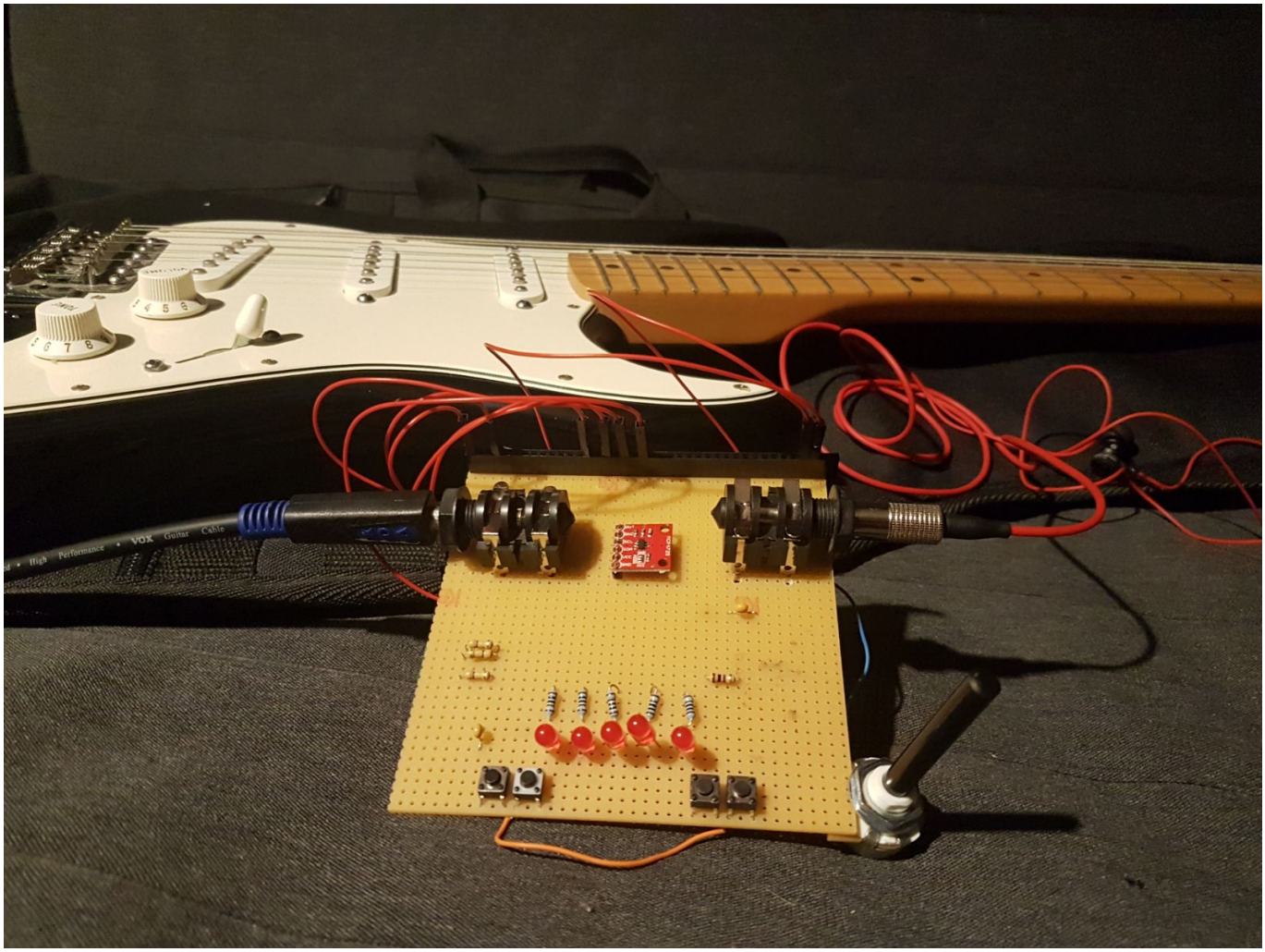
Echo

Ma folosesc de un buffer circular pentru a retine sample-urile procesate(output_history).

```
  float apply_echo(float x)
  {
       float out_delayed = (float)output_history[(output_history_last +
(MAX_OUTPUT_HISTORY - 1) - es_echo_delay) % MAX_OUTPUT_HISTORY];
      out_delayed /= SGN_MAX;

      return (1 - es_echo_depth) * x + es_echo_depth * out_delayed;
  }
```

Rezultate Obţinute


VOX Guitar Cable

Concluzii

Pentru implementarea looper-ului dorit la inceputul acestui proiect aveam nevoie de mufe jack 6.3mm mama Stereo. Pentru excluderea frecventelor nedorite cred ca ar fi fost mai benefica folosirea unui amplificator diferential care ar reduce zgomotele si ar pastra informatia utila pentru adc-ul din ATMega324. De asemenea, lipsa memoriei m-a impiedicat sa memorez un interval de timp semnificativ. La final, semnalul inregistrat impreuna cu semnalul curent vor fi trecute printr-un sumator operational catre output daca se folosesc tot mufe mono, daca se folosesc mufe stereo se pot pune pe canale diferite. Probabil la urmatoarea incercare voi opta pentru un DAC cu o rezolutie mai mare.

Proiectul si-a atins scopul, am invatat foarte multe in implementarea acestui procesor de efecte.

Download

ilienicolaevlad.zip

Bibliografie/Resurse

Digital Audio Effects 10_CM0268_Audio_FX.pdf

Distortion distortion

Wah-Wah 213.php

Tremolo coding-some-tremolo

Phase Shifter 053675.html

Mono vs Stereo Jack stereo-and-mono-cables-and-jacks-what-happens-when-you-cross-them

Arduino Guitar Pedal Arduino-Guitar-Pedal

Digital Delay arduino-based-low-bit-stand-alone.html

- Documentația în format PDF