ESEMPIO DI IMPIEGO DELLA PIATTAFORMA ARDUINO IN AMBITO SONORO-MUSICALE

Stefano Silvestri

Il seguente progetto consiste nella realizzazione di un “sintetizzatore caotico” per scopi musicali. Per la realizzazione è stata impiegata una tradizionale scheda Arduino UNO con microcontrollore Atmel Atmega328P-PU per costruire un’interfaccia di controllo digitale-analogico che interviene su una coppia di circuiti non-lineari. Le condizioni di generazione delle oscillazioni caotiche vengono regolate attraverso la modifica dei parametri del circuito e per mezzo di un software scritto in linguaggio Processing che si interfaccia con la scheda Arduino via comunicazione seriale.

Segue lo schema teorico degli oscillatori caotici accoppiati inoltre da una resistenza utile per verificare l’eventualità della comparsa di sincronizzazione dei sistemi complessi (acusticamente oscillazioni in accordo di fase/frequenza/ampiezza o attrattori reciprocamente influenzati):

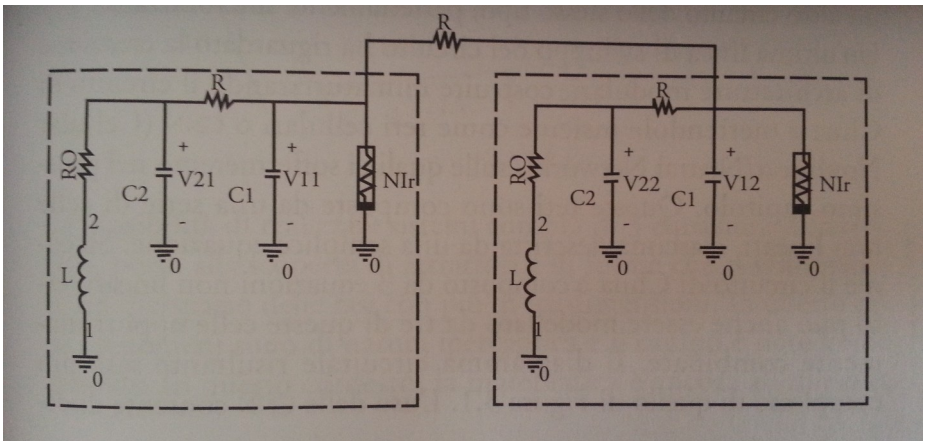
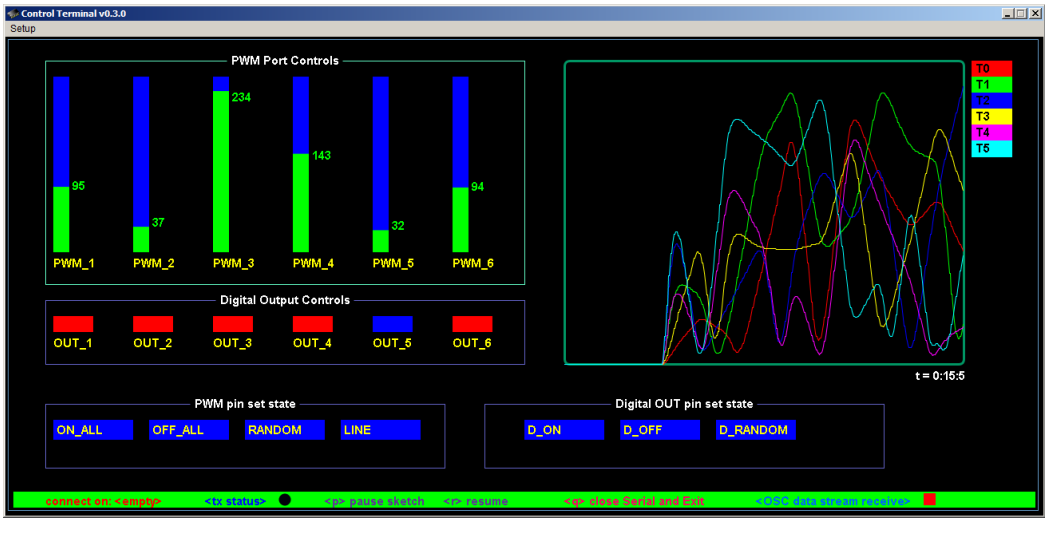


Fig.1

Per il controllo del circuito analogico è stato ideato un apposito driver basato su optoisolatori vactrol e switch digitali IC4N35 che intervengono sui componenti RC (Fig.1) rispettivamente regolandone la resistività mediante segnale PWM o interrompendo/cortocircuitando capacità in parallelo con semplici segnali digitali.

L’interfaccia software mostrata di seguito consente quindi di inviare valori di LOW/HIGH o di PWM (da 0 a 5V) per controllare le correnti nei punti dei parametri resistivi e capacitivi del circuito applicativo reale:



Il sistema è così interfacciato al computer per il controllo digitale mediante USB e nel suo complesso risulta il seguente:

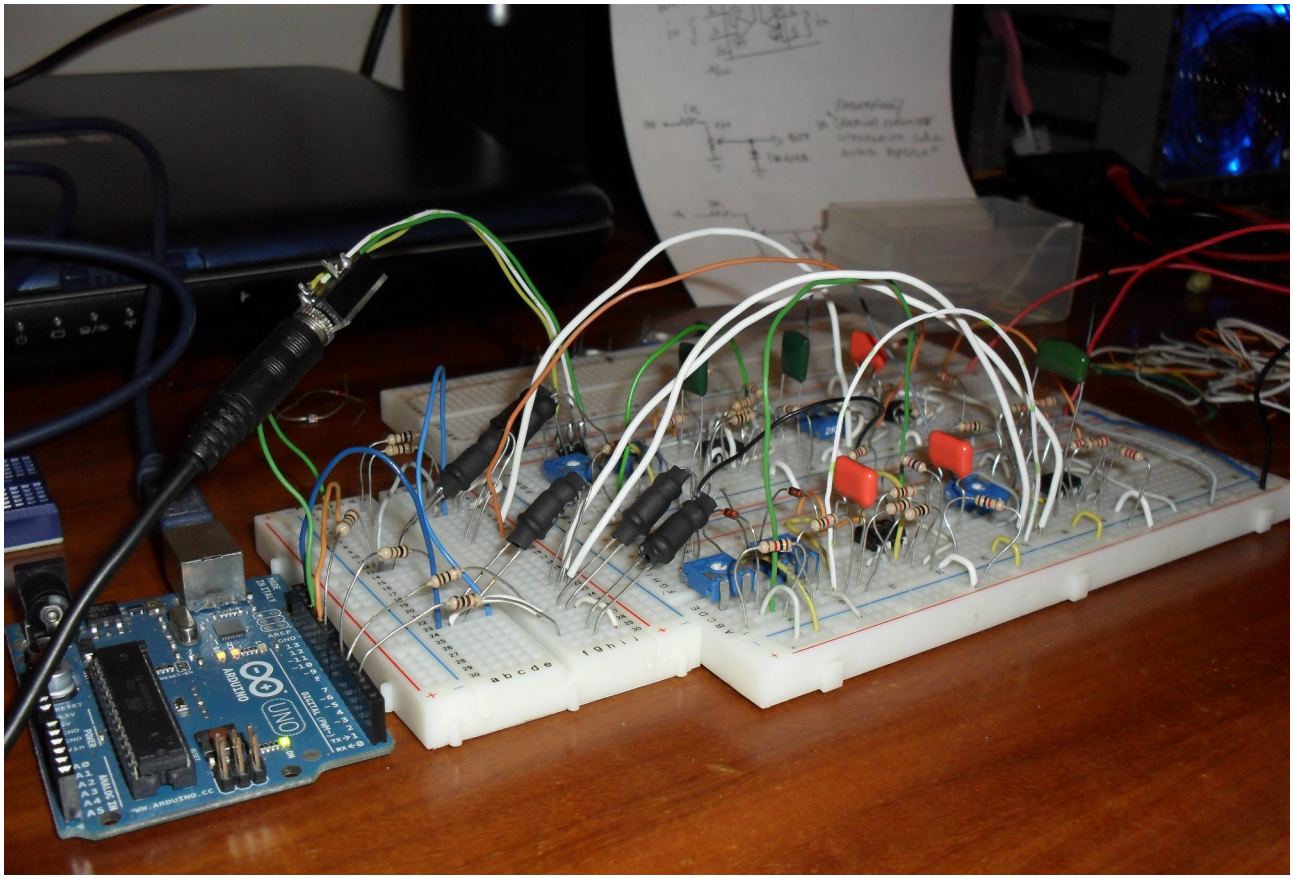


Fig.2 – A sinistra la Board Arduino per l'interfacciamento al PC; sezione optoisolatori e interruttori digitali (breadboard centrale); a destra la sezione analogica dei circuiti caotici di Chua

Di seguito si mostra un'analisi della forma d'onda del segnale nel tempo e nello spettro di ampiezza in banda udibile:

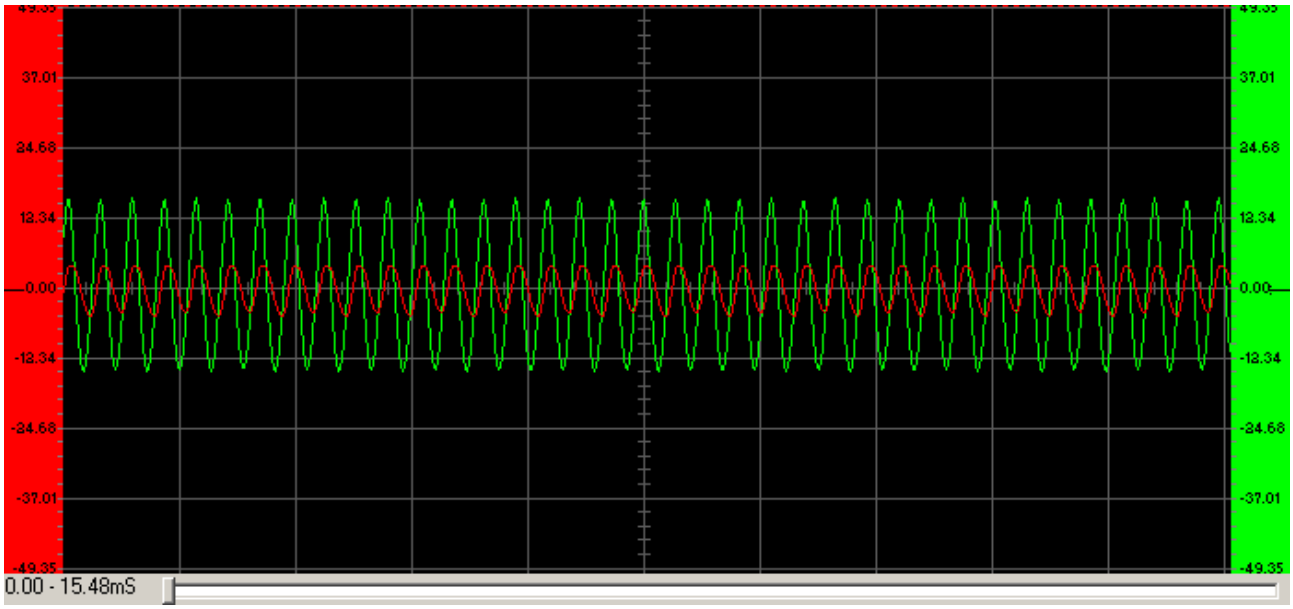


Fig.1a – Oscillazione orbita circolare

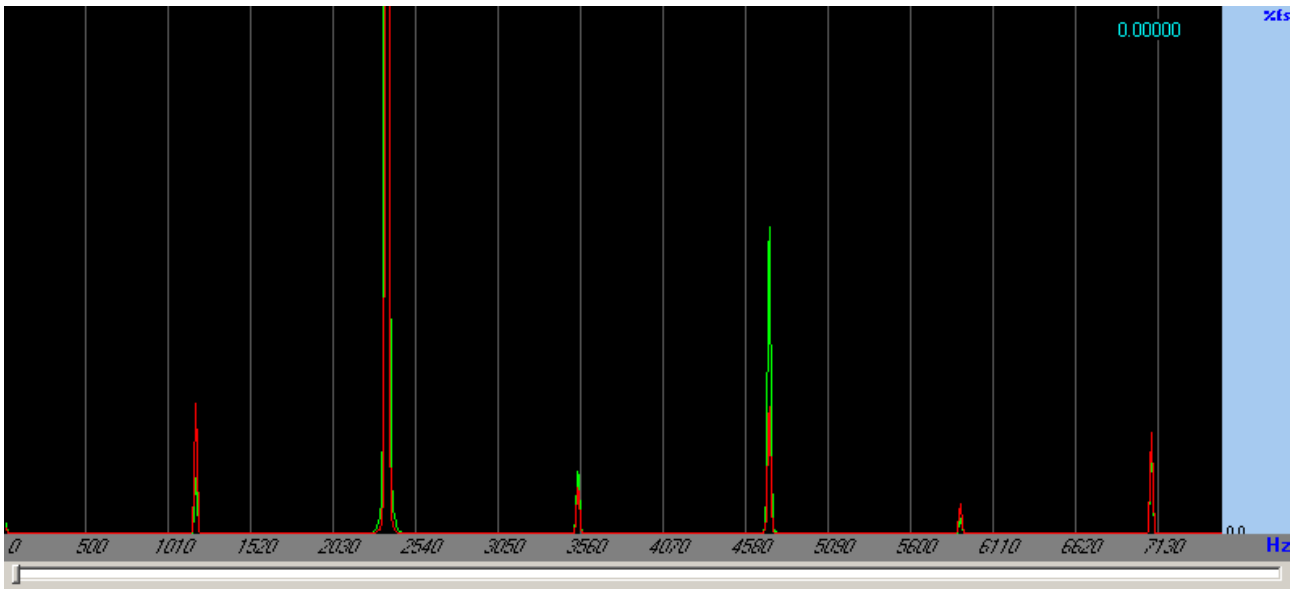


Fig.2a – Spettro orbita circolare

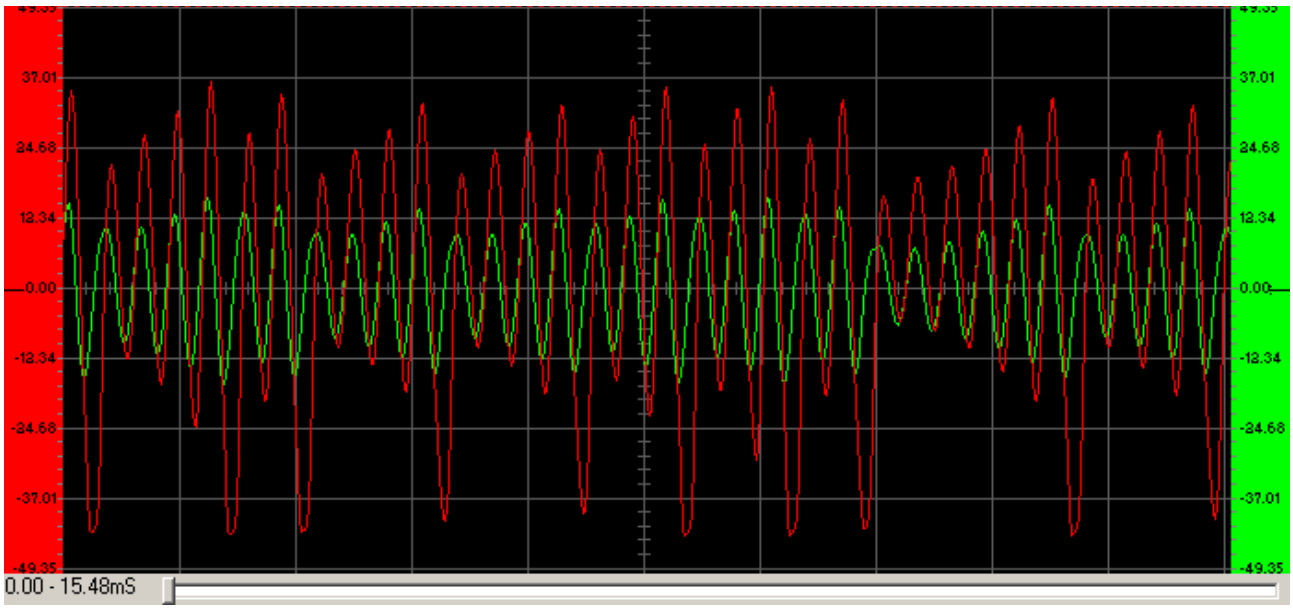


Fig.1b – Oscillazione caotica

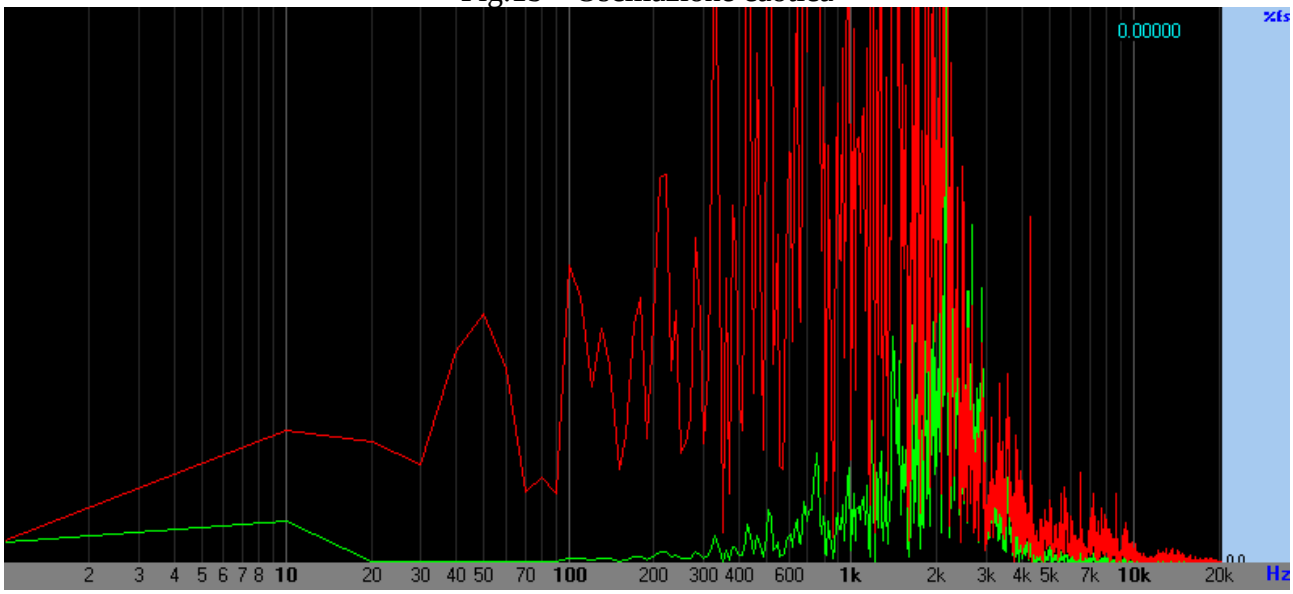
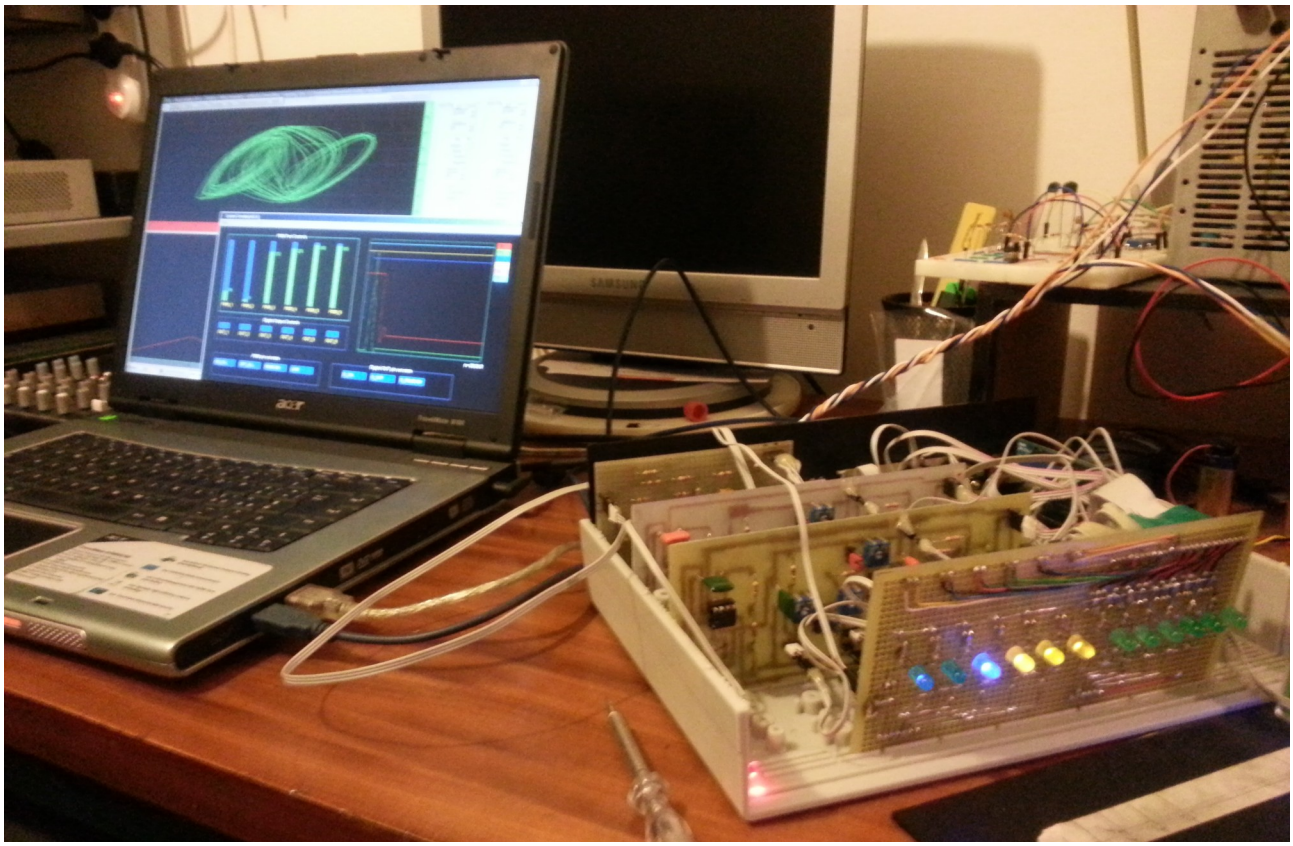
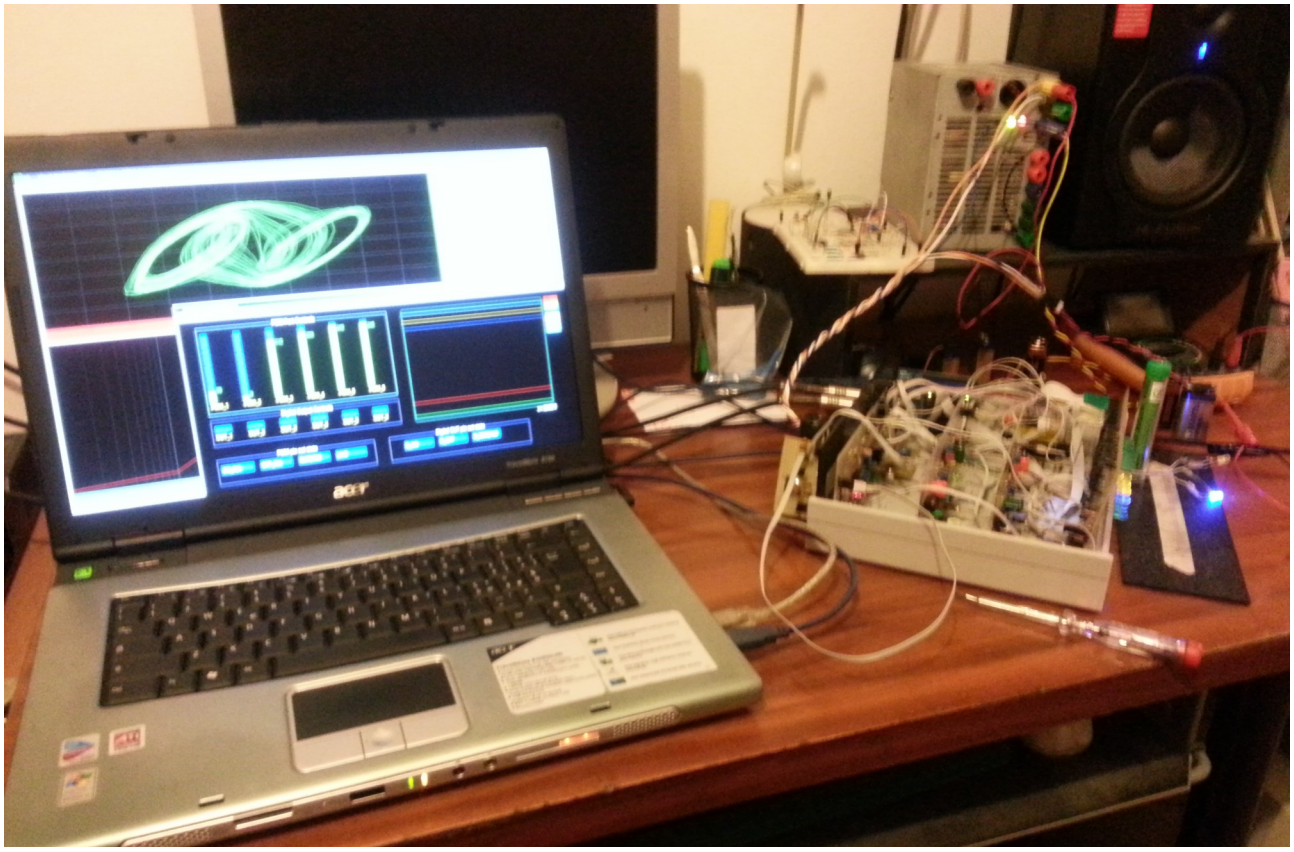


Fig.2b – Spettro dell'oscillazione caotica

Foto del prototipo completo:





Di seguito lo Sketch per la programmazione dell'interfaccia di controllo in Arduino:

```
/*---------------------------------------------------------------------------------------------
ricezione dati seriali da serial monitor o programma per invio
dati seriali:

byte di PWM (0-255), pin digital out (HIGH-LOW)

                          byte header   byte tag      byte val
protocollo:      /                      @                    0 - 255

Rispetto alla vecchia versione è stato scritto un protocollo di comunicazione dedicato, evitando
l'uso di firmata e la libreria Arduino per processing

vantaggi: migliore risposta e stabilità dei controlli delle porte

Stefano Silvestri
11/08/2013 18:54:54
----------------------------------------------------------------------------------------------*/

#define HEADER      '/'                                     //campo di header char (1Byte)
//#define TAG_PWM_1        ':'                              //campi di tag char
#define BYTE_MESSAGGIO          14                          //header+tag+6pwm   val+6digital
out val
//byte vals[6];                                             //campi variabile da (1-byte)

//char TAGS_PWM[6] = {':', '$', '%', '&', '@', '#'};
//char TAGS_DIGTALOUT[6] = {'£', '!', '?', '^', '*', '~'};

const int PWM_PINS[6]={3, 5, 6, 9, 10, 11};
const int DIGITAL_OUT_PINS[6]={2, 4, 7, 8, 12, 13};

const int power_status= A0;
//const int rx_monitor= 0;


void setup()
{
        Serial.begin(9600);

        for (int i=0; i<sizeof(DIGITAL_OUT_PINS); i++) //dichiarazione pin digitali come uscita
        {
                pinMode(DIGITAL_OUT_PINS[i], OUTPUT);
        }

        for (int i=0; i<sizeof(PWM_PINS); i++)//inizializzazione pwm a low, non c'è bisogno di
dichiararli come output
        {
                digitalWrite(PWM_PINS[i], LOW);
        }
```

```
	pinMode(power_status, OUTPUT);
	digitalWrite(power_status, LOW);
	//pinMode(rx_monitor, OUTPUT);
	//digitalWrite(rx_monitor, LOW);

}


void loop()
{
	char tag;

	if (Serial.available() >= BYTE_MESSAGGIO) //se arriva almeno la lunghezza minima del messaggio
	{
		digitalWrite(power_status, HIGH);

		if (Serial.read() == HEADER)//lettura primo Byte HEADER e verifica correttezza
		{
			//Serial.print(HEADER);	//stampa su seriale per lettura da processing

			tag=Serial.read();

			switch(tag)
			{
				case ':':
					//vals[0]=Serial.read();
	//--------------------lettura valore pwm ricevuto
					analogWrite(PWM_PINS[0], Serial.read());
					//Serial.print(tag); Serial.println(vals[0]);
					break;
				case '$':
					analogWrite(PWM_PINS[1], Serial.read());
					break;
				case '%':
					analogWrite(PWM_PINS[2], Serial.read());
					break;
				case '&':
					analogWrite(PWM_PINS[3], Serial.read());
					break;
				case '@':
					analogWrite(PWM_PINS[4], Serial.read());
					break;
				case '#':
					analogWrite(PWM_PINS[5], Serial.read());
					break;

				case '£':
					if (Serial.read()==1)
		//---------------lettura tag pin digitali
					{
						digitalWrite(DIGITAL_OUT_PINS[0], HIGH);
```

```
		}
		else {
			digitalWrite(DIGITAL_OUT_PINS[0], LOW);
		}
		break;
case '!':
		if (Serial.read()==1)

		{
			digitalWrite(DIGITAL_OUT_PINS[1], HIGH);
		}
		else {
			digitalWrite(DIGITAL_OUT_PINS[1], LOW);
		}
		break;
case '?':
		if (Serial.read()==1)

		{
			digitalWrite(DIGITAL_OUT_PINS[2], HIGH);
		}
		else {
			digitalWrite(DIGITAL_OUT_PINS[2], LOW);
		}
		break;
case '^':
		if (Serial.read()==1)

		{
			digitalWrite(DIGITAL_OUT_PINS[3], HIGH);
		}
		else {
			digitalWrite(DIGITAL_OUT_PINS[3], LOW);
		}
		break;
case '*':
		if (Serial.read()==1)

		{
			digitalWrite(DIGITAL_OUT_PINS[4], HIGH);
		}
		else {
			digitalWrite(DIGITAL_OUT_PINS[4], LOW);
		}
		break;
case '~':
		if (Serial.read()==1)

		{
			digitalWrite(DIGITAL_OUT_PINS[5], HIGH);
		}
		else {
```

```
                                        digitalWrite(DIGITAL_OUT_PINS[5], LOW);
                                    }
                                    break;
                                default:
                                    Serial.print("uncorrisponding TAG!!!\t");
                                    Serial.println("tag");
                                    break;
                            }


                        }

                }
                Serial.flush();

}
```

BIBLIOGRAFIA

Silvestri S. (2013), *Studio sulle applicazioni compositive di sistemi complessi: esperimento su interazioni caotiche - parte 1*

Silvestri S. (2013), *Una performance in tempo reale basata su un sistema caotico controllato digitalmente: esperimento delle interazioni caotiche - parte 2*

Progetto presentato nell'ambito del Corso CODING – I.S.I.S. Europa
Docente formatore Avanguardie Educative Prof. Angelo Giordano

Napoli, 15/12/2021 – *Stefano Silvestri*