UNIVERSIDADE FEDERAL DE UBERLÂNDIA - UFU

FACULDADE DE ENGENHARIA ELÉTRICA – FEELT

SINAIS E SISTEMAS EM ENGENHARIA BIOMÉDICA

**Acionamento de Motor de Passo Bipolar**

**Alunos:**

1. Ítalo Gustavo Sampaio Fernandes - 11511EBI004
2. Nathalia Rodrigues - 11411EBI018
3. Paulo Camargos Silva - 11611EBI023

**Prof.** Sérgio Ricardo de Jesus Oliveira

Uberlândia, **01** de **novembro** de 2017

**1 – Introdução**

.

Os Motores de Passo são dispositivos eletromecânicos que convertem pulsos elétricos em movimentos mecânicos que geram variações angulares discretas. O rotor ou eixo de um motor de passo é rotacionado em pequenos incrementos angulares, denominados “passos”, quando pulsos elétricos são aplicados em uma determinada sequência nos terminais deste. A rotação de tais motores é diretamente relacionada aos impulsos elétricos que são recebidos, bem como a sequência a qual tais pulsos são aplicados refletem diretamente na direção a qual o motor gira. A velocidade que o rotor gira é dada pela frequência de pulsos recebidos e o tamanho do ângulo rotacionado é diretamente relacionado com o número de pulsos aplicados. Um motor de passo pode ser usado em aplicações onde é necessário controlar vários fatores tais como: ângulo de rotação, velocidade, posição e sincronismo. O ponto forte de um motor de passo é a possibilidade de controlar seus movimentos de forma precisa.

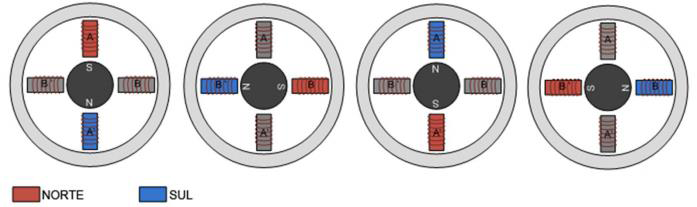


Figura 1 - Princípio de funcionamento do motor

O Motor de Passo Bipolar usa uma ligação por polo e necessita que o circuito de controle possa reverter o sentido da corrente para acionar as bobinas de forma correta Fig.2. Os motores bipolares são constituídos por bobinas sem derivação central. Por este fato, estas bobinas devem ser energizadas de tal forma que a corrente elétrica flua na direção inversa a cada dois passos para permitir o movimento contínuo do rotor, ou seja, a polaridade deve ser invertida durante o funcionamento do motor.

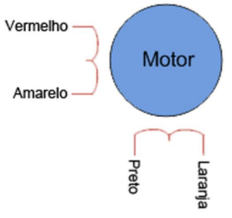


Figura 2 - Motor de Passo Bipolar

Os motores de passo bipolares são conhecidos por sua excelente relação tamanho/torque: eles proporcionam um maior torque, cerca de 40% a mais, comparativamente a um motor unipolar do mesmo tamanho. Isto se deve ao fato de que quando se energiza uma fase, se magnetiza ambos os polos em que a fase (ou bobina) está instalada. Assim, o rotor sofre a ação de forças magnéticas de ambos os polos, ao invés de apenas um, como acontece no motor unipolar.

**2 – Materiais e Métodos**

Os materiais utilizados neste experimento foram:

* Arduino;
* Módulo L298;
* 1 Motor de Passo Bipolar;
* Fonte de Alimentação 5v.

**2.1 – Esquema do circuito**

A figura abaixo exibe o esquema do circuito montado na *protoboard* neste experimento.

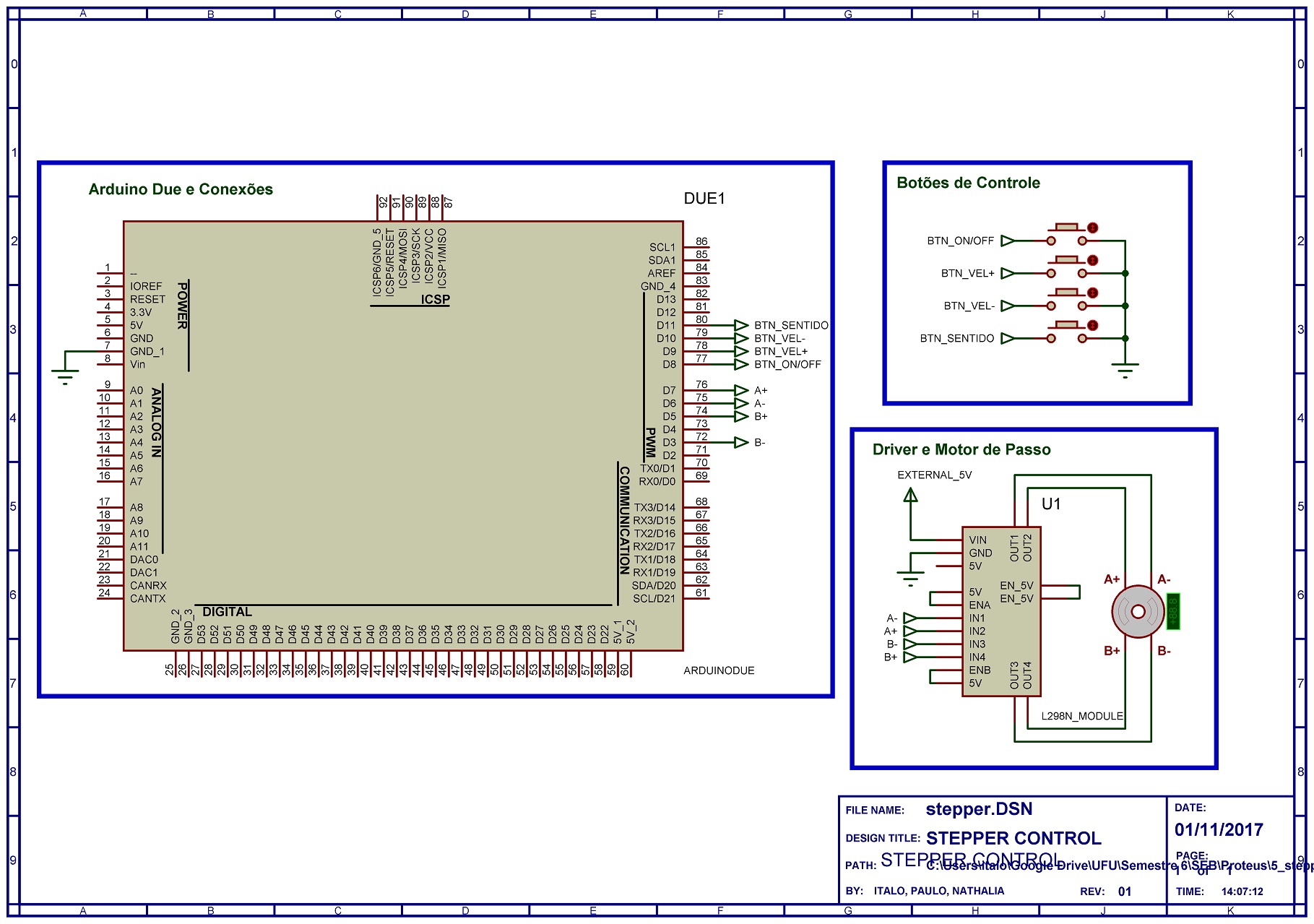


Figura 3: Esquema do circuito montado para o controle do motor bipolar com o módulo L298N.

**2.2 – Código**

O código desenvolvido para este experimento se encontra no Anexo I e no link GitHub exibido ao final do relatório. O diagrama abaixo descreve o funcionamento do código.

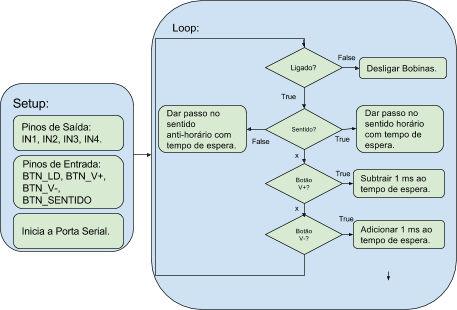
****

Figura 4 - Diagrama do Código

No método *void setup()* são definidos os pinos de saída do motor IN1, IN2, IN3 e IN4 e pinos de controle, dos botões de on-off, velocidade e sentido. No interior do ocorre a verificação se o sistema está ligado, caso sim, é verificado o sentido (estado do pino). Caso o pino de sentido leia valor *true*, o código chama a função para dar os passos no sentido horário com respectivo tempo de espera. Caso o pino leia *false*, o motor dá os passos no sentido anti-horário. Após a verificação e excução do sentido, é verificado o pino que aumenta a velocidade. Caso seja lido *true,* a velocidade é aumentada, ou seja, o tempo de *delay* é decrescido. Caso seja lido *false*, o motor diminui a velocidade, aumentando o tempo de *delay*. A Figura 5 exibe o diagrama UML do código escrito.

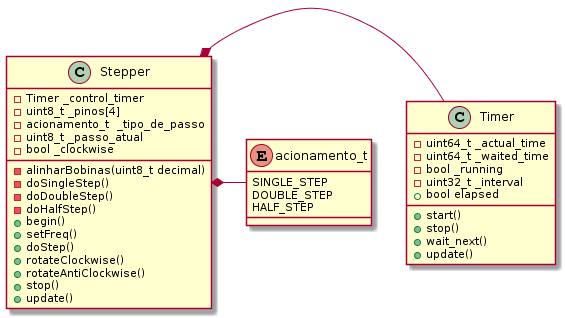


Figura 5 - Diagrama de Classes UML das classes implementadas

**3 – Resultados e discussões**

Foi possível observar o funcionamento do motor de passo bipolar, a partir de um sistema que era capaz de aumentar ou diminuir sua velocidade, mudar o sentido de rotação (horário/anti-horário), assim como ligar e desligar, todos por meio de uma chave.

Pela chave L-D controlamos se o motor estava ligado ou não. Quando a chave estava em “HIGH”, nível lógico alto, o motor funcionava, quando recebia “LOW” ele parava. Pela chave VEL+ controlamos a aceleração do motor, quando recebia “LOW”, aumentava a velocidade do motor, ou seja, diminuímos o tempo entre um passo e outro para 5ms. Pela chave VEL- tínhamos o mesmo funcionamento da anterior, mas agora diminuímos a velocidade, podendo ver que aumentamos *delay* entre cada passo. Já pela chave A-H controlamos o sentido de rotação, no qual o motor funcionava normalmente no sentido horário e quando conectamos a chave ele girava no sentido contrário.

**4 – Conclusão**

O motor de passo é um dispositivo eletromecânico de extrema importância e diversas aplicações. A sua utilização associada ao Arduino proporciona uma infinidade de possibilidades de configuração de rotação. O experimento realizado neste Laboratório possibilitou nosso aprendizado sobre o funcionamento do motor e seu controle. Com os conhecimentos adquiridos, poderemos realizar projetos que envolvam o controle preciso de posição, por exemplo, um robô “*self balancing”.*

**5 – Referências**

1. http://labdegaragem.com/profiles/blogs/tutorial-sobre-motor-de-passo
2. http://www.feis.unesp.br/Home/departamentos/engenhariaeletrica/aula3-motor-de-passo-2013-1-13-03-2013-final.pdf

**ANEXO I - CÓDIGO .ino PARA MOTOR - Disponível também no seguinte link:** [**https://github.com/italogfernandes/SEB/blob/master/projeto\_5\_stepper/src/main.ino**](https://github.com/italogfernandes/SEB/blob/master/projeto_5_stepper/src/main.ino)

1. /\* UNIVERSIDADE FEDERAL DE UBERLANDIA
2. Biomedical Engineering
3. Autors: Ítalo G S Fernandes
4. Paulo Camargos Silva
5. Nathalia Rodrigues
6. contact: italogsfernandes@gmail.com
7. URLs: https://github.com/italogfernandes/SEB
8. Este codigo faz parte da disciplina de sinais e sistemas
9. para engenhara biomedica e visa realizar o controle de
10. um motor de passo.
11. Nele existem 2 implementacoes, uma mais simples com delay
12. e sem preocupacao com possiveis erros.
13. E outra mais elaborada, onde as principais funcoes foram
14. encapsuladas como objetos e podem ser reutilizadas com
15. maior facilidade.
16. Esquema de montagem:
17. Arduino - Dispositivo
18. 8       -   Chave Liga Desliga
19. 9       -   Chave de Aumentar a Velocidade
20. 10      -   Chave de Diminuir a Velocidade
21. 11      -   Chave para Alterar Sentido de Rotacao
22. 7       -   Terminal A+ (Positivo da Bobina A) do Motor de Passo
23. 6       -   Terminal A- (Negativo da Bobina A) do Motor de Passo
24. 5       -   Terminal B+ (Positivo da Bobina B) do Motor de Passo
25. 3       -   Terminal B- (Negativo da Bobina B) do Motor de Passo
26. Obs: As chaves nao sao usadas no codigo mais elaborado,
27. este se comunica via comunicao Serial com Baudrate de 115200
28. Menu:
29. +: Mais Velocidade
30. -: Menos Velocidade
31. p: Parar
32. r: Rodar Sentido Horário
33. a: Rodar Sentido Anti-Horário
34. s: Single-Step
35. d: Double-Step
36. h: Half-Step
38. Para selecionar qual tipo de codigo voce deseja executar use:
39. Para selecionar a versao com delay:
40. #define VERSAO\_COM\_DELAY
41. //#define VERSAO\_SEM\_DELAY
42. Para selecionar a versao sem delay:
43. //#define VERSAO\_COM\_DELAY
44. #define VERSAO\_SEM\_DELAY
45. \*/
47. //Descomente e comente qual versao você deseja executar:
48. #define VERSAO\_COM\_DELAY
49. //#define VERSAO\_SEM\_DELAY

52. /////////////////////////////////////
53. //IMPLEMENTACAO DA VERSAO COM DELAY//
54. /////////////////////////////////////
55. #ifdef VERSAO\_COM\_DELAY
56. // Definição dos pinos
57. #define pinLD 8
58. #define pinVelmais 9
59. #define pinVelmenos 10
60. #define pinSentido 11
61. #define pinAmais 7
62. #define pinAmenos 6
63. #define pinBmais 5
64. #define pinBmenos 3
65. int tempoespera = 20;
67. void setup() {
68. pinMode(pinLD, INPUT\_PULLUP);
69. pinMode(pinVelmais, INPUT\_PULLUP);
70. pinMode(pinVelmenos, INPUT\_PULLUP);
71. pinMode(pinSentido, INPUT\_PULLUP);
72. pinMode(pinAmais, OUTPUT);
73. pinMode(pinAmenos, OUTPUT);
74. pinMode(pinBmais, OUTPUT);
75. pinMode(pinBmenos, OUTPUT);
76. Serial.begin(9600);
77. }
79. void loop() {
80. if (digitalRead(pinLD) == HIGH) {
82. if (digitalRead(pinSentido)) {
83. darpassodouble(tempoespera);
84. }
85. else {
86. darpassodoubleanti(tempoespera);
87. }
89. if (digitalRead(pinVelmais) == LOW) {
90. tempoespera = tempoespera - 1;
91. if (tempoespera < 5) {
92. tempoespera = 5; // velocidade maxima
93. }
94. Serial.print("Novo Tempo = ");
95. Serial.print(tempoespera);
96. Serial.println(" ms");
97. }
98. if (digitalRead(pinVelmenos) == LOW) {
99. tempoespera = tempoespera + 1;
100. Serial.print("Novo Tempo = ");
101. Serial.print(tempoespera);
102. Serial.println(" ms");
103. }
104. } else {
105. digitalWrite(pinAmais, 0);
106. digitalWrite(pinAmenos, 0);
107. digitalWrite(pinBmais, 0);
108. digitalWrite(pinBmenos, 0);
109. }
110. }
111. void darpassosingle(int tempo) {
112. digitalWrite(pinAmais, 1);
113. digitalWrite(pinAmenos, 0);
114. digitalWrite(pinBmais, 0);
115. digitalWrite(pinBmenos, 0);
116. Serial.println(" Passo 1");
117. delay(tempo);
118. digitalWrite(pinAmais, 0);
119. digitalWrite(pinAmenos, 0);
120. digitalWrite(pinBmais, 0);
121. digitalWrite(pinBmenos, 1);
122. //Serial.println(" Passo 2");
123. delay(tempo);
124. digitalWrite(pinAmais, 0);
125. digitalWrite(pinAmenos, 1);
126. digitalWrite(pinBmais, 0);
127. digitalWrite(pinBmenos, 0);
128. Serial.println(" Passo 3");
129. delay(tempo);
130. digitalWrite(pinAmais, 0);
131. digitalWrite(pinAmenos, 0);
132. digitalWrite(pinBmais, 1);
133. digitalWrite(pinBmenos, 0);
134. Serial.println(" Passo 4");
135. delay(tempo);
136. }
138. void darpassodouble(int tempo) {
140. digitalWrite(pinAmais, 1);
141. digitalWrite(pinAmenos, 0);
142. digitalWrite(pinBmais, 0);
143. digitalWrite(pinBmenos, 1);
144. // Serial.println(" Passo 1");
145. delay(tempo);
146. digitalWrite(pinAmais, 0);
147. digitalWrite(pinAmenos, 1);
148. digitalWrite(pinBmais, 0);
149. digitalWrite(pinBmenos, 1);
150. //  Serial.println(" Passo 2");
151. delay(tempo);
152. digitalWrite(pinAmais, 0);
153. digitalWrite(pinAmenos, 1);
154. digitalWrite(pinBmais, 1);
155. digitalWrite(pinBmenos, 0);
156. // Serial.println(" Passo 3");
157. delay(tempo);
158. digitalWrite(pinAmais, 1);
159. digitalWrite(pinAmenos, 0);
160. digitalWrite(pinBmais, 1);
161. digitalWrite(pinBmenos, 0);
162. //Serial.println(" Passo 4");
163. delay(tempo);
164. }
165. void darpassodoubleanti(int tempo) {
166. //Ta bugada essa
167. digitalWrite(pinAmais, 1);
168. digitalWrite(pinAmenos, 0);
169. digitalWrite(pinBmais, 0);
170. digitalWrite(pinBmenos, 1);
171. Serial.println(" Passo 1");
172. delay(tempo);
173. digitalWrite(pinAmais, 1);
174. digitalWrite(pinAmenos, 0);
175. digitalWrite(pinBmais, 1);
176. digitalWrite(pinBmenos, 0);
177. Serial.println(" Passo 2");
178. delay(tempo);
179. digitalWrite(pinAmais, 0);
180. digitalWrite(pinAmenos, 1);
181. digitalWrite(pinBmais, 1);
182. digitalWrite(pinBmenos, 0);
183. Serial.println(" Passo 3");
184. delay(tempo);
185. digitalWrite(pinAmais, 0);
186. digitalWrite(pinAmenos, 1);
187. digitalWrite(pinBmais, 0);
188. digitalWrite(pinBmenos, 1);
189. Serial.println(" Passo 4");
190. delay(tempo);
191. }
192. #endif /\*VERSAO\_COM\_DELAY\*/
194. /////////////////////////////////////
195. //IMPLEMENTACAO DA VERSAO SEM DELAY//
196. /////////////////////////////////////
197. #ifdef VERSAO\_SEM\_DELAY
198. #define USING\_ARDUINO\_DUE
199. //#define USING\_ARUDINO\_UNO
201. ////////////
202. //Defines //
203. ////////////
204. #define UART\_BAUDRATE 115200

207. ///////////
208. //Clases //
209. ///////////
211. /////////////////////////////////////////////////
212. // Implementacao de um timer atraves do millis //
213. /////////////////////////////////////////////////
215. class Timer {
216. private:
217. unsigned long \_actual\_time;
218. unsigned long \_waited\_time;
219. bool \_running;
220. uint32\_t \_interval;
222. public:
223. bool elapsed;
225. Timer(uint32\_t interval = 1000) {
226. \_interval = interval;
227. elapsed = false;
228. \_actual\_time = micros();
229. \_waited\_time = \_actual\_time + interval;
230. }
232. void setInterval(uint32\_t interval) {
233. \_interval = interval;
234. }
236. uint32\_t getInterval() {
237. return \_interval;
238. }
240. void start() {
241. \_running = true;
242. \_actual\_time = millis();
243. \_waited\_time = \_actual\_time + \_interval;
244. }
246. void stop() {
247. \_running = false;
248. }
250. void wait\_next() {
251. elapsed = false;
252. }
254. void update() {
255. \_actual\_time = micros();
256. if (\_running) {
257. if (\_actual\_time >= \_waited\_time) {
258. \_waited\_time = \_actual\_time + \_interval;
259. elapsed = true;
260. }
261. }
262. }
264. };


268. ///////////////////////////////////////////
269. //Classe para controle do motor de passo //
270. ///////////////////////////////////////////
272. //Necessarios para a Classe Stepper:
273. typedef enum {
274. SINGLE\_STEP,
275. DOUBLE\_STEP,
276. HALF\_STEP
277. } acionamento\_t;
278. //Sequencias de passos
279. const uint8\_t single\_step\_seq[4] = {8, 4, 2, 1};
280. const uint8\_t double\_step\_seq[4] = {12, 6, 3, 9};
281. const uint8\_t half\_step\_seq[8] = {8, 12, 4, 6, 2, 3, 1, 9};
283. class Stepper {
284. private:
285. Timer \_control\_timer;
286. uint8\_t \_pin\_phase\_A[2]; //Pinos da Fase A [Positivo] [Negativo]
287. uint8\_t \_pin\_phase\_B[2]; //Pinos da Fase B [Positivo] [Negativo]
288. acionamento\_t \_tipo\_de\_passo;
289. uint8\_t \_passo\_atual;
290. bool \_clockwise;
292. void alinharBobinas(uint8\_t decimal) {
293. digitalWrite(\_pin\_phase\_A[0], (decimal >> 3) & 0x01); //Bit 3
294. digitalWrite(\_pin\_phase\_B[0], (decimal >> 2) & 0x01); //Bit 2
295. digitalWrite(\_pin\_phase\_A[1], (decimal >> 1) & 0x01); //Bit 1
296. digitalWrite(\_pin\_phase\_B[1], (decimal >> 0) & 0x01); //Bit 0
297. }
299. void doSingleStep() {
300. alinharBobinas(single\_step\_seq[\_passo\_atual / 2]); //Utiliza apenas os indices e 0,2,4,6 para dar o passo
301. }
303. void doDoubleStep() {
304. alinharBobinas(double\_step\_seq[\_passo\_atual / 2]);
305. }
307. void doHalfStep() {
308. alinharBobinas(half\_step\_seq[\_passo\_atual]);
309. }
311. public:
312. /\*\*
313. [Stepper description]
314. @param pinAPositive  [description]
315. @param pinANegative  [description]
316. @param pinBPositive  [description]
317. @param pinBNegative  [description]
318. @param tipo\_de\_passo [description]
319. \*/
320. Stepper(uint8\_t pinAPositive, uint8\_t pinANegative,
321. uint8\_t pinBPositive, uint8\_t pinBNegative,
322. acionamento\_t tipo\_de\_passo = SINGLE\_STEP) {
323. \_pin\_phase\_A[0] = pinAPositive;
324. \_pin\_phase\_A[1] = pinANegative;
325. \_pin\_phase\_B[0] = pinBPositive;
326. \_pin\_phase\_B[1] = pinBNegative;
327. \_tipo\_de\_passo = tipo\_de\_passo;
328. \_passo\_atual = 0;
329. \_control\_timer.setInterval(20000);
330. \_clockwise = true;
331. }
333. void begin() {
334. pinMode(\_pin\_phase\_A[0], OUTPUT);
335. pinMode(\_pin\_phase\_B[0], OUTPUT);
336. pinMode(\_pin\_phase\_A[1], OUTPUT);
337. pinMode(\_pin\_phase\_B[1], OUTPUT);
338. }
340. void setAcionamento(acionamento\_t tipo\_de\_passo) {
341. \_tipo\_de\_passo = tipo\_de\_passo;
342. }
344. void setStepInterval(uint32\_t interval) {
345. \_control\_timer.setInterval(interval);
346. }
348. uint32\_t getStepInterval() {
349. return \_control\_timer.getInterval();
350. }
352. void setFreq(uint32\_t frequency) {
353. setStepInterval(1000000 / frequency);
354. }
356. uint32\_t getFreq() {
357. return 1000000 / getStepInterval();
358. }
360. void doStep() {
361. switch (\_tipo\_de\_passo) {
362. case SINGLE\_STEP:
363. doSingleStep();
364. break;
365. case DOUBLE\_STEP:
366. doDoubleStep();
367. break;
368. case HALF\_STEP:
369. doHalfStep();
370. break;
371. }
372. if (\_clockwise) {
373. ++\_passo\_atual = \_passo\_atual % 8; //Incremento circular
374. } else { //Decemento circular
375. if (\_passo\_atual == 0) {
376. \_passo\_atual = 7;
377. } else {
378. \_passo\_atual = \_passo\_atual - 1;
379. }
381. }
382. }
384. void rotateClockwise() {
385. \_clockwise = true;
386. \_control\_timer.start();
387. }
389. void rotateAntiClockwise() {
390. \_clockwise = false;
391. \_control\_timer.start();
392. }
394. void stop() {
395. \_control\_timer.stop();
396. }
398. void update() {
399. \_control\_timer.update();
400. if (\_control\_timer.elapsed) {
401. doStep();
402. \_control\_timer.wait\_next();
403. }
404. }
405. };

408. //////////////////////
409. //Variaveis globais //
410. //////////////////////
411. Stepper motor(7, 6, 5, 3); ///A+, A-, B+ , B-
412. bool status\_led = false;
413. char serialOp;
414. //////////////////
415. //Main Function //
416. //////////////////
417. void setup() {
418. pinMode(LED\_BUILTIN, OUTPUT);
419. motor.begin();
420. Serial.begin(UART\_BAUDRATE);
421. Serial.println("Hora do show porra!");
422. Serial.println("Envie um comando:");
423. Serial.println("+: Mais Velocidade");
424. Serial.println("-: Menos Velocidade");
425. Serial.println("p: Parar");
426. Serial.println("r: Rodar Sentido Horário");
427. Serial.println("a: Rodar Sentido Anti-Horário");
428. Serial.println("s: Single-Step");
429. Serial.println("d: Double-Step");
430. Serial.println("h: Half-Step");
431. motor.setFreq(100);
432. //Biiirl
433. motor.rotateClockwise();
434. }
436. void loop() {
437. motor.update();
438. if (Serial.available()) {
439. serialOp = Serial.read();
440. switch (serialOp) {
441. case '+':
442. motor.setFreq(motor.getFreq() + 1);
443. Serial.println("Step Freq: " + String(motor.getFreq()) + " Hz");
444. Serial.println("Step Interval: " + String(motor.getStepInterval()) + " us");
445. break;
446. case '1':
447. motor.setFreq(motor.getFreq() + 10);
448. Serial.println("Step Freq: " + String(motor.getFreq()) + " Hz");
449. Serial.println("Step Interval: " + String(motor.getStepInterval()) + " us");
450. break;
451. case '2':
452. motor.setFreq(motor.getFreq() - 10);
453. Serial.println("Step Freq: " + String(motor.getFreq()) + " Hz");
454. Serial.println("Step Interval: " + String(motor.getStepInterval()) + " us");
455. break;
456. case '3':
457. motor.setFreq(motor.getFreq() + 100);
458. Serial.println("Step Freq: " + String(motor.getFreq()) + " Hz");
459. Serial.println("Step Interval: " + String(motor.getStepInterval()) + " us");
460. break;
461. case '4':
462. motor.setFreq(motor.getFreq() - 100);
463. Serial.println("Step Freq: " + String(motor.getFreq()) + " Hz");
464. Serial.println("Step Interval: " + String(motor.getStepInterval()) + " us");
465. break;
466. case '-':
467. motor.setFreq(motor.getFreq() - 1);
468. Serial.println("Step Freq: " + String(motor.getFreq()) + " Hz");
469. Serial.println("Step Interval: " + String(motor.getStepInterval()) + " us");
470. break;
472. case 'p':
473. motor.stop();
474. Serial.println("Stopped");
475. break;
476. case 'r':
477. motor.rotateClockwise();
478. Serial.println("Started Clockwise");
479. break;
480. case 'a':
481. motor.rotateAntiClockwise();
482. Serial.println("Started AntiClockwise");
483. break;
484. case 's':
485. motor.setAcionamento(SINGLE\_STEP);
486. Serial.println("Mode: Single-Step");
487. break;
488. case 'd':
489. motor.setAcionamento(DOUBLE\_STEP);
490. Serial.println("Mode: Double-Step");
491. break;
492. case 'h':
493. motor.setAcionamento(HALF\_STEP);
494. Serial.println("Mode: Half-Step");
495. break;
496. default:
497. break;
498. }
499. }
500. }
501. /////////////////////////
502. //End of Main Function //
503. /////////////////////////
504. #endif /\*VERSAO\_SEM\_DELAY\*/