

Equipe de Robótica da FURG  
FBot/FURGBOL

Centro de Ciências Computacionais - C3  
Universidade Federal do Rio Grande

## **Conserto dos Motores Brushless EC 45 - 30 Watt MAXON**

Autor	Julia
	Lohanna
	Mariana
	Ysabella
	Vivian Misaki Aoki

Rio Grande, 8 de janeiro de 2019

# 1 Objetivo

Recuperar máximo de motores velhos.

## 2 Desenvolvimento

### 2.1 Partes do motor

- Base do motor;
- Placa do circuito dos 3 Hall's;
- Proteção de plástico para o circuito da placa;
- Enrolamento de cobre;
- conjunto Bucha de latão;
- Coisa que fecha o conjunto de bucha de latão;
- Imã.



Figura 1: Partes do motor

Fonte: O autor

- Bucha de Latão com Rolamento no fundo;
- Mola com suporte de plástico;
- Outro Rolamento;
- Coisa que fecha a bucha.



Figura 2: Partes internas do latão (peças 5 e 6 da Figura 1)

Fonte: O autor

### 3 Desmontar o motor

- Apoiar na fita isolante ou em algo com o mesmo diâmetro (Figura 3)



Figura 3: Uma das partes apoiada para ser desmontada

Fonte: O autor

- Usar um pino, inclinado, quando dá para sentir que está pegando no segundo rolamento, bater com martelo para abrir (Figura 4), as partes internas vão se separar (Figuras 5 e 6)



Figura 4: Primeiro passo para desmontar o motor  
Fonte: O autor



Figura 5: Parte interna que será retirada  
Fonte: O autor



Figura 6: Parte que se mantém junta  
Fonte: O autor

- Com o ferro de solda tirar a solda dos fios das bobinas, ir batendo, com um prego, pra tirar os três pinos do centro e separar as bobinas da placa/flat (Figura3)
- Separar todas as partes



Figura 7: Placa interna  
Fonte: O autor

## 4 Identificação e solução de problemas

- Testar os Hall
  - Pinos utilizados: o 5V, GND e as leituras dos Hall
  - Colocar a fonte em 5V, ligar o GND antes de qualquer coisa e depois os 5V
  - Ler os Hall com o osciloscópio, precisa ter movimento no motor, gera uma onda quadrada

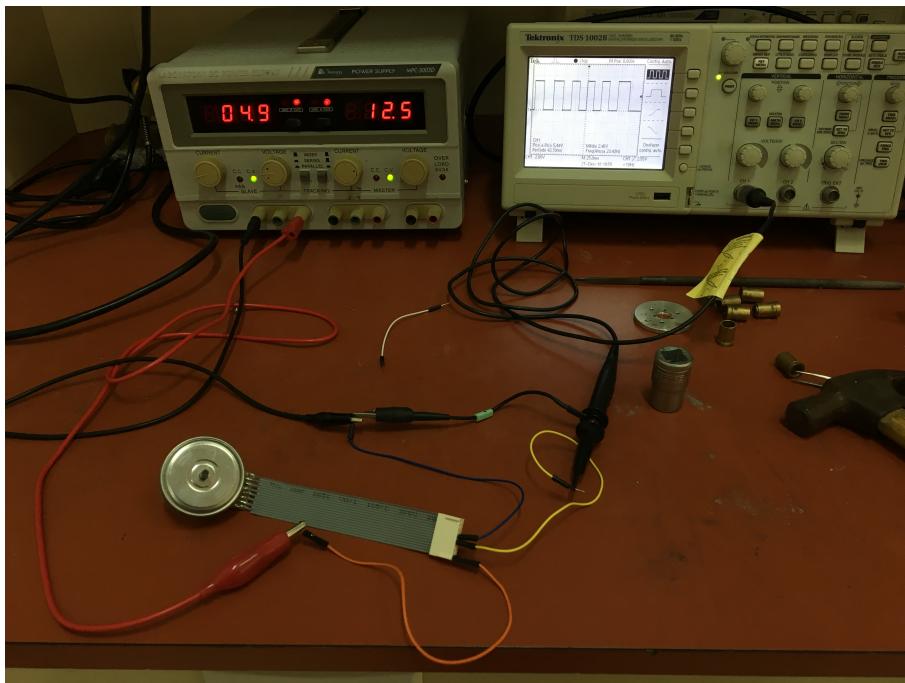


Figura 8: Teste do hall

Fonte: O autor

- Testar se há curto entre os pinos e a borda dos mesmo. Se houver:
  - Borda dos furos centrais pode estar colocando o motor inteiro em curto
  - Observar se os pinos estão encostando na borda, se sim, usar uma broca maior que o furo e raspar a borda (três furos centrais e o latão)
- Verificar se os pinos do plug estão corretos e bem encaixados
- Verificar a resistência entre as bobinas

## 5 Placa interna do motor

- Aumentar os três furos centrais com uma broca de 2mm
- Prender a placa e o disco externo em uma madeira, com pregos, com a parte metálica para cima

- Utilizando o furo central da parte externa como base, acertar o furo da placa com uma broca de 11mm
- Desmontar para fazer as soldas

## 6 Montar o motor

- Soldar as bobinas de volta na placa, tomar cuidado para não fechar curto
- inserir a bucha com rolamento → suporte de plástico → mola → suporte de plástico → rolamento → o item que fecha a bucha ao conjunto de bobina.



Figura 9: Conjunto da bobina  
Fonte: O autor



Figura 10: Inserindo bucha no conjunto de bobina  
Fonte: O autor



Figura 11: Conjunto de mola  
Fonte: O autor



Figura 12: Procedimento1  
Fonte: O autor



Figura 13: Procedimento2

Fonte: O autor



Figura 14: Conjunto da bobina

Fonte: O autor

- Usar uma ferramenta redonda, soquete de chave estrela, (que não encoste nos fios das bobinas) e um martelo para fechar as partes do meio



Figura 15: Fechando o conjunto  
Fonte: O autor

- Fechar o motor com cuidado

## 7 Driver's do motor

- Foi feito teste de acionamento de todos os motores disponíveis nos driver's com o código de teste dos motores encontrado no google drive (A) usando a biblioteca para modificar a frequência do PWM (B).
- Nos driver's sem números não funcionou, e no número 11 necessita de impulso externo a cada parada do motor.
- Testes de continuidade e curto são feitos e faz a troca dos CI's que continuam não funcionando.

## Apêndice A Código Teste dos Motores

```
1 //#include ".\Pwm01\pwm01.h"
2 #include "pwm01.h"
3
4 #define dir_m1 9
5 #define en_m1 8
6 //#define pwm_m1 3
7 #define hall_1 10
8 #define hall_2 12
9 #define hall_3 11
10
11 uint32_t pwm_duty = 32767;
12 uint32_t pwm_freq1 = 20000;
13 uint32_t valPWM = 0;
14
15
16 void setup()
17 {
18     // Set PWM Resolution
19     //    pwm_set_resolution(16);
20
21     // Setup PWM Once (Up to two unique frequencies allowed
22     //-----
23     pwm_setup( dir_m1, pwm_freq1, 1 );
24
25
26     // Write PWM Duty Cycle Anytime After PWM Setup
27     //-----
28
29     pinMode(en_m1, OUTPUT);
30     pinMode(hall_1, INPUT);
31     pinMode(hall_2, INPUT);
32     pinMode(hall_3, INPUT);
33
34     Serial.begin(115200);
35
36     digitalWrite(en_m1, HIGH);
37 }
38
39 void loop()
40 {
41
42     while (valPWM <= 255) {
43         pwm_write_duty( dir_m1, valPWM );
44         delay(50);
45         valPWM++;
46
47         Serial.println(valPWM);
48     }
49     delay(2000);
50
51     while (valPWM > 0) {
52         valPWM--;
53         pwm_write_duty( dir_m1, valPWM );
54         delay(50);
```

```
55     Serial.println(valPWM);  
56 }  
57 delay(2000);  
58 }
```

## Apêndice B Biblioteca Teste dos Motores

```

1  /*
2
3 Library:      pwm01.h (version 01)
4 Date:        2/11/2013
5 Written By:   randomvibe
6
7 Purpose:
8     Setup one or two unique PWM frequencies directly in sketch
9     program ,
10    set PWM duty cycle , and stop PWM function .
11
12 User Functions:
13     pwm_set_resolution( int res ) ~ setup PWM resolution; up to 16
14     bit
15     pwm_setup( uint32_t pwm_pin, uint32_t pwm_freq, int iclock ) ~
16     Setup PWM on clock-A (iclock=1) or clock-B (iclock=2)
17     pwm_write_duty( uint32_t pwm_pin, uint32_t pwm_duty ) ~ Write
18     PWM duty cycle
19     pwm_stop( uint32_t pwm_pin ) ~ Force PWM stop
20
21 Notes:
22     - Applies to Arduino-Due board , PWM pins 6, 7, 8 & 9.
23     - Libary Does not operate on the TIO pins.
24     - Unique frequencies set via PWM Clock-A ("CLKA") and Clock-B
25     ("CLKB")
26     Therefore , up to two unique frequencies allowed .
27     - Set max duty cycle counts (pwm_max_duty_Ncount) equal to 255
28     per Arduino approach. This value is best SUITED for low
29     frequency
30     applications (2hz to 40,000hz) such as PWM motor drivers ,
31     38khz infrared transmitters , etc .
32     - Future library versions will address high frequency
33     applications .
34     - Arduino's "wiring_analog.c" function was very helpful in this
35     effort .
36
37 */
38
39 #include "Arduino.h"
40
41
42 static int      pwm_resolution_nbit = 8;
43 static uint32_t  pwm_clockA_freq = 0;
44 static uint32_t  pwm_clockB_freq = 0;
45 static uint32_t  pwm_max_duty_Ncount = 255;
46
47 void pwm_set_resolution( int res )
48 {
49     pwm_resolution_nbit = res ;
50 }
51
52 void pwm_set_clockA_freq( uint32_t val )
53 {
54     pwm_clockA_freq = val;
55 }
56
57 void pwm_set_clockB_freq( uint32_t val )
58 {
59     pwm_clockB_freq = val;
60 }
61
62 void pwm_stop( uint32_t pin )
63 {
64     analogWrite( pin, 0 );
65 }
66
67 void pwm_start( uint32_t pin )
68 {
69     analogWrite( pin, 255 );
70 }
71
72 void pwm_set_duty( uint32_t pin, uint32_t duty )
73 {
74     analogWrite( pin, duty );
75 }
76
77 void pwm_get_duty( uint32_t pin, uint32_t *duty )
78 {
79     *duty = analogRead( pin );
80 }
81
82 void pwm_set_resolution( int res )
83 {
84     pwm_resolution_nbit = res;
85 }
86
87 void pwm_set_clockA_freq( uint32_t val )
88 {
89     pwm_clockA_freq = val;
90 }
91
92 void pwm_set_clockB_freq( uint32_t val )
93 {
94     pwm_clockB_freq = val;
95 }
96
97 void pwm_stop( uint32_t pin )
98 {
99     analogWrite( pin, 0 );
100 }
101
102 void pwm_start( uint32_t pin )
103 {
104     analogWrite( pin, 255 );
105 }
106
107 void pwm_set_duty( uint32_t pin, uint32_t duty )
108 {
109     analogWrite( pin, duty );
110 }
111
112 void pwm_get_duty( uint32_t pin, uint32_t *duty )
113 {
114     *duty = analogRead( pin );
115 }
116
117 void pwm_set_resolution( int res )
118 {
119     pwm_resolution_nbit = res;
120 }
121
122 void pwm_set_clockA_freq( uint32_t val )
123 {
124     pwm_clockA_freq = val;
125 }
126
127 void pwm_set_clockB_freq( uint32_t val )
128 {
129     pwm_clockB_freq = val;
130 }
131
132 void pwm_stop( uint32_t pin )
133 {
134     analogWrite( pin, 0 );
135 }
136
137 void pwm_start( uint32_t pin )
138 {
139     analogWrite( pin, 255 );
140 }
141
142 void pwm_set_duty( uint32_t pin, uint32_t duty )
143 {
144     analogWrite( pin, duty );
145 }
146
147 void pwm_get_duty( uint32_t pin, uint32_t *duty )
148 {
149     *duty = analogRead( pin );
150 }
151
152 void pwm_set_resolution( int res )
153 {
154     pwm_resolution_nbit = res;
155 }
156
157 void pwm_set_clockA_freq( uint32_t val )
158 {
159     pwm_clockA_freq = val;
160 }
161
162 void pwm_set_clockB_freq( uint32_t val )
163 {
164     pwm_clockB_freq = val;
165 }
166
167 void pwm_stop( uint32_t pin )
168 {
169     analogWrite( pin, 0 );
170 }
171
172 void pwm_start( uint32_t pin )
173 {
174     analogWrite( pin, 255 );
175 }
176
177 void pwm_set_duty( uint32_t pin, uint32_t duty )
178 {
179     analogWrite( pin, duty );
180 }
181
182 void pwm_get_duty( uint32_t pin, uint32_t *duty )
183 {
184     *duty = analogRead( pin );
185 }
186
187 void pwm_set_resolution( int res )
188 {
189     pwm_resolution_nbit = res;
190 }
191
192 void pwm_set_clockA_freq( uint32_t val )
193 {
194     pwm_clockA_freq = val;
195 }
196
197 void pwm_set_clockB_freq( uint32_t val )
198 {
199     pwm_clockB_freq = val;
200 }
201
202 void pwm_stop( uint32_t pin )
203 {
204     analogWrite( pin, 0 );
205 }
206
207 void pwm_start( uint32_t pin )
208 {
209     analogWrite( pin, 255 );
210 }
211
212 void pwm_set_duty( uint32_t pin, uint32_t duty )
213 {
214     analogWrite( pin, duty );
215 }
216
217 void pwm_get_duty( uint32_t pin, uint32_t *duty )
218 {
219     *duty = analogRead( pin );
220 }
221
222 void pwm_set_resolution( int res )
223 {
224     pwm_resolution_nbit = res;
225 }
226
227 void pwm_set_clockA_freq( uint32_t val )
228 {
229     pwm_clockA_freq = val;
230 }
231
232 void pwm_set_clockB_freq( uint32_t val )
233 {
234     pwm_clockB_freq = val;
235 }
236
237 void pwm_stop( uint32_t pin )
238 {
239     analogWrite( pin, 0 );
240 }
241
242 void pwm_start( uint32_t pin )
243 {
244     analogWrite( pin, 255 );
245 }
246
247 void pwm_set_duty( uint32_t pin, uint32_t duty )
248 {
249     analogWrite( pin, duty );
250 }
251
252 void pwm_get_duty( uint32_t pin, uint32_t *duty )
253 {
254     *duty = analogRead( pin );
255 }
256
257 void pwm_set_resolution( int res )
258 {
259     pwm_resolution_nbit = res;
260 }
261
262 void pwm_set_clockA_freq( uint32_t val )
263 {
264     pwm_clockA_freq = val;
265 }
266
267 void pwm_set_clockB_freq( uint32_t val )
268 {
269     pwm_clockB_freq = val;
270 }
271
272 void pwm_stop( uint32_t pin )
273 {
274     analogWrite( pin, 0 );
275 }
276
277 void pwm_start( uint32_t pin )
278 {
279     analogWrite( pin, 255 );
280 }
281
282 void pwm_set_duty( uint32_t pin, uint32_t duty )
283 {
284     analogWrite( pin, duty );
285 }
286
287 void pwm_get_duty( uint32_t pin, uint32_t *duty )
288 {
289     *duty = analogRead( pin );
290 }
291
292 void pwm_set_resolution( int res )
293 {
294     pwm_resolution_nbit = res;
295 }
296
297 void pwm_set_clockA_freq( uint32_t val )
298 {
299     pwm_clockA_freq = val;
300 }
301
302 void pwm_set_clockB_freq( uint32_t val )
303 {
304     pwm_clockB_freq = val;
305 }
306
307 void pwm_stop( uint32_t pin )
308 {
309     analogWrite( pin, 0 );
310 }
311
312 void pwm_start( uint32_t pin )
313 {
314     analogWrite( pin, 255 );
315 }
316
317 void pwm_set_duty( uint32_t pin, uint32_t duty )
318 {
319     analogWrite( pin, duty );
320 }
321
322 void pwm_get_duty( uint32_t pin, uint32_t *duty )
323 {
324     *duty = analogRead( pin );
325 }
326
327 void pwm_set_resolution( int res )
328 {
329     pwm_resolution_nbit = res;
330 }
331
332 void pwm_set_clockA_freq( uint32_t val )
333 {
334     pwm_clockA_freq = val;
335 }
336
337 void pwm_set_clockB_freq( uint32_t val )
338 {
339     pwm_clockB_freq = val;
340 }
341
342 void pwm_stop( uint32_t pin )
343 {
344     analogWrite( pin, 0 );
345 }
346
347 void pwm_start( uint32_t pin )
348 {
349     analogWrite( pin, 255 );
350 }
351
352 void pwm_set_duty( uint32_t pin, uint32_t duty )
353 {
354     analogWrite( pin, duty );
355 }
356
357 void pwm_get_duty( uint32_t pin, uint32_t *duty )
358 {
359     *duty = analogRead( pin );
360 }
361
362 void pwm_set_resolution( int res )
363 {
364     pwm_resolution_nbit = res;
365 }
366
367 void pwm_set_clockA_freq( uint32_t val )
368 {
369     pwm_clockA_freq = val;
370 }
371
372 void pwm_set_clockB_freq( uint32_t val )
373 {
374     pwm_clockB_freq = val;
375 }
376
377 void pwm_stop( uint32_t pin )
378 {
379     analogWrite( pin, 0 );
380 }
381
382 void pwm_start( uint32_t pin )
383 {
384     analogWrite( pin, 255 );
385 }
386
387 void pwm_set_duty( uint32_t pin, uint32_t duty )
388 {
389     analogWrite( pin, duty );
390 }
391
392 void pwm_get_duty( uint32_t pin, uint32_t *duty )
393 {
394     *duty = analogRead( pin );
395 }
396
397 void pwm_set_resolution( int res )
398 {
399     pwm_resolution_nbit = res;
400 }
401
402 void pwm_set_clockA_freq( uint32_t val )
403 {
404     pwm_clockA_freq = val;
405 }
406
407 void pwm_set_clockB_freq( uint32_t val )
408 {
409     pwm_clockB_freq = val;
410 }
411
412 void pwm_stop( uint32_t pin )
413 {
414     analogWrite( pin, 0 );
415 }
416
417 void pwm_start( uint32_t pin )
418 {
419     analogWrite( pin, 255 );
420 }
421
422 void pwm_set_duty( uint32_t pin, uint32_t duty )
423 {
424     analogWrite( pin, duty );
425 }
426
427 void pwm_get_duty( uint32_t pin, uint32_t *duty )
428 {
429     *duty = analogRead( pin );
430 }
431
432 void pwm_set_resolution( int res )
433 {
434     pwm_resolution_nbit = res;
435 }
436
437 void pwm_set_clockA_freq( uint32_t val )
438 {
439     pwm_clockA_freq = val;
440 }
441
442 void pwm_set_clockB_freq( uint32_t val )
443 {
444     pwm_clockB_freq = val;
445 }
446
447 void pwm_stop( uint32_t pin )
448 {
449     analogWrite( pin, 0 );
450 }
451
452 void pwm_start( uint32_t pin )
453 {
454     analogWrite( pin, 255 );
455 }
456
457 void pwm_set_duty( uint32_t pin, uint32_t duty )
458 {
459     analogWrite( pin, duty );
460 }
461
462 void pwm_get_duty( uint32_t pin, uint32_t *duty )
463 {
464     *duty = analogRead( pin );
465 }
466
467 void pwm_set_resolution( int res )
468 {
469     pwm_resolution_nbit = res;
470 }
471
472 void pwm_set_clockA_freq( uint32_t val )
473 {
474     pwm_clockA_freq = val;
475 }
476
477 void pwm_set_clockB_freq( uint32_t val )
478 {
479     pwm_clockB_freq = val;
480 }
481
482 void pwm_stop( uint32_t pin )
483 {
484     analogWrite( pin, 0 );
485 }
486
487 void pwm_start( uint32_t pin )
488 {
489     analogWrite( pin, 255 );
490 }
491
492 void pwm_set_duty( uint32_t pin, uint32_t duty )
493 {
494     analogWrite( pin, duty );
495 }
496
497 void pwm_get_duty( uint32_t pin, uint32_t *duty )
498 {
499     *duty = analogRead( pin );
500 }
501
502 void pwm_set_resolution( int res )
503 {
504     pwm_resolution_nbit = res;
505 }
506
507 void pwm_set_clockA_freq( uint32_t val )
508 {
509     pwm_clockA_freq = val;
510 }
511
512 void pwm_set_clockB_freq( uint32_t val )
513 {
514     pwm_clockB_freq = val;
515 }
516
517 void pwm_stop( uint32_t pin )
518 {
519     analogWrite( pin, 0 );
520 }
521
522 void pwm_start( uint32_t pin )
523 {
524     analogWrite( pin, 255 );
525 }
526
527 void pwm_set_duty( uint32_t pin, uint32_t duty )
528 {
529     analogWrite( pin, duty );
530 }
531
532 void pwm_get_duty( uint32_t pin, uint32_t *duty )
533 {
534     *duty = analogRead( pin );
535 }
536
537 void pwm_set_resolution( int res )
538 {
539     pwm_resolution_nbit = res;
540 }
541
542 void pwm_set_clockA_freq( uint32_t val )
543 {
544     pwm_clockA_freq = val;
545 }
546
547 void pwm_set_clockB_freq( uint32_t val )
548 {
549     pwm_clockB_freq = val;
550 }
551
552 void pwm_stop( uint32_t pin )
553 {
554     analogWrite( pin, 0 );
555 }
556
557 void pwm_start( uint32_t pin )
558 {
559     analogWrite( pin, 255 );
560 }
561
562 void pwm_set_duty( uint32_t pin, uint32_t duty )
563 {
564     analogWrite( pin, duty );
565 }
566
567 void pwm_get_duty( uint32_t pin, uint32_t *duty )
568 {
569     *duty = analogRead( pin );
570 }
571
572 void pwm_set_resolution( int res )
573 {
574     pwm_resolution_nbit = res;
575 }
576
577 void pwm_set_clockA_freq( uint32_t val )
578 {
579     pwm_clockA_freq = val;
580 }
581
582 void pwm_set_clockB_freq( uint32_t val )
583 {
584     pwm_clockB_freq = val;
585 }
586
587 void pwm_stop( uint32_t pin )
588 {
589     analogWrite( pin, 0 );
590 }
591
592 void pwm_start( uint32_t pin )
593 {
594     analogWrite( pin, 255 );
595 }
596
597 void pwm_set_duty( uint32_t pin, uint32_t duty )
598 {
599     analogWrite( pin, duty );
600 }
601
602 void pwm_get_duty( uint32_t pin, uint32_t *duty )
603 {
604     *duty = analogRead( pin );
605 }
606
607 void pwm_set_resolution( int res )
608 {
609     pwm_resolution_nbit = res;
610 }
611
612 void pwm_set_clockA_freq( uint32_t val )
613 {
614     pwm_clockA_freq = val;
615 }
616
617 void pwm_set_clockB_freq( uint32_t val )
618 {
619     pwm_clockB_freq = val;
620 }
621
622 void pwm_stop( uint32_t pin )
623 {
624     analogWrite( pin, 0 );
625 }
626
627 void pwm_start( uint32_t pin )
628 {
629     analogWrite( pin, 255 );
630 }
631
632 void pwm_set_duty( uint32_t pin, uint32_t duty )
633 {
634     analogWrite( pin, duty );
635 }
636
637 void pwm_get_duty( uint32_t pin, uint32_t *duty )
638 {
639     *duty = analogRead( pin );
640 }
641
642 void pwm_set_resolution( int res )
643 {
644     pwm_resolution_nbit = res;
645 }
646
647 void pwm_set_clockA_freq( uint32_t val )
648 {
649     pwm_clockA_freq = val;
650 }
651
652 void pwm_set_clockB_freq( uint32_t val )
653 {
654     pwm_clockB_freq = val;
655 }
656
657 void pwm_stop( uint32_t pin )
658 {
659     analogWrite( pin, 0 );
660 }
661
662 void pwm_start( uint32_t pin )
663 {
664     analogWrite( pin, 255 );
665 }
666
667 void pwm_set_duty( uint32_t pin, uint32_t duty )
668 {
669     analogWrite( pin, duty );
670 }
671
672 void pwm_get_duty( uint32_t pin, uint32_t *duty )
673 {
674     *duty = analogRead( pin );
675 }
676
677 void pwm_set_resolution( int res )
678 {
679     pwm_resolution_nbit = res;
680 }
681
682 void pwm_set_clockA_freq( uint32_t val )
683 {
684     pwm_clockA_freq = val;
685 }
686
687 void pwm_set_clockB_freq( uint32_t val )
688 {
689     pwm_clockB_freq = val;
690 }
691
692 void pwm_stop( uint32_t pin )
693 {
694     analogWrite( pin, 0 );
695 }
696
697 void pwm_start( uint32_t pin )
698 {
699     analogWrite( pin, 255 );
700 }
701
702 void pwm_set_duty( uint32_t pin, uint32_t duty )
703 {
704     analogWrite( pin, duty );
705 }
706
707 void pwm_get_duty( uint32_t pin, uint32_t *duty )
708 {
709     *duty = analogRead( pin );
710 }
711
712 void pwm_set_resolution( int res )
713 {
714     pwm_resolution_nbit = res;
715 }
716
717 void pwm_set_clockA_freq( uint32_t val )
718 {
719     pwm_clockA_freq = val;
720 }
721
722 void pwm_set_clockB_freq( uint32_t val )
723 {
724     pwm_clockB_freq = val;
725 }
726
727 void pwm_stop( uint32_t pin )
728 {
729     analogWrite( pin, 0 );
730 }
731
732 void pwm_start( uint32_t pin )
733 {
734     analogWrite( pin, 255 );
735 }
736
737 void pwm_set_duty( uint32_t pin, uint32_t duty )
738 {
739     analogWrite( pin, duty );
740 }
741
742 void pwm_get_duty( uint32_t pin, uint32_t *duty )
743 {
744     *duty = analogRead( pin );
745 }
746
747 void pwm_set_resolution( int res )
748 {
749     pwm_resolution_nbit = res;
750 }
751
752 void pwm_set_clockA_freq( uint32_t val )
753 {
754     pwm_clockA_freq = val;
755 }
756
757 void pwm_set_clockB_freq( uint32_t val )
758 {
759     pwm_clockB_freq = val;
760 }
761
762 void pwm_stop( uint32_t pin )
763 {
764     analogWrite( pin, 0 );
765 }
766
767 void pwm_start( uint32_t pin )
768 {
769     analogWrite( pin, 255 );
770 }
771
772 void pwm_set_duty( uint32_t pin, uint32_t duty )
773 {
774     analogWrite( pin, duty );
775 }
776
777 void pwm_get_duty( uint32_t pin, uint32_t *duty )
778 {
779     *duty = analogRead( pin );
780 }
781
782 void pwm_set_resolution( int res )
783 {
784     pwm_resolution_nbit = res;
785 }
786
787 void pwm_set_clockA_freq( uint32_t val )
788 {
789     pwm_clockA_freq = val;
790 }
791
792 void pwm_set_clockB_freq( uint32_t val )
793 {
794     pwm_clockB_freq = val;
795 }
796
797 void pwm_stop( uint32_t pin )
798 {
799     analogWrite( pin, 0 );
800 }
801
802 void pwm_start( uint32_t pin )
803 {
804     analogWrite( pin, 255 );
805 }
806
807 void pwm_set_duty( uint32_t pin, uint32_t duty )
808 {
809     analogWrite( pin, duty );
810 }
811
812 void pwm_get_duty( uint32_t pin, uint32_t *duty )
813 {
814     *duty = analogRead( pin );
815 }
816
817 void pwm_set_resolution( int res )
818 {
819     pwm_resolution_nbit = res;
820 }
821
822 void pwm_set_clockA_freq( uint32_t val )
823 {
824     pwm_clockA_freq = val;
825 }
826
827 void pwm_set_clockB_freq( uint32_t val )
828 {
829     pwm_clockB_freq = val;
830 }
831
832 void pwm_stop( uint32_t pin )
833 {
834     analogWrite( pin, 0 );
835 }
836
837 void pwm_start( uint32_t pin )
838 {
839     analogWrite( pin, 255 );
840 }
841
842 void pwm_set_duty( uint32_t pin, uint32_t duty )
843 {
844     analogWrite( pin, duty );
845 }
846
847 void pwm_get_duty( uint32_t pin, uint32_t *duty )
848 {
849     *duty = analogRead( pin );
850 }
851
852 void pwm_set_resolution( int res )
853 {
854     pwm_resolution_nbit = res;
855 }
856
857 void pwm_set_clockA_freq( uint32_t val )
858 {
859     pwm_clockA_freq = val;
860 }
861
862 void pwm_set_clockB_freq( uint32_t val )
863 {
864     pwm_clockB_freq = val;
865 }
866
867 void pwm_stop( uint32_t pin )
868 {
869     analogWrite( pin, 0 );
870 }
871
872 void pwm_start( uint32_t pin )
873 {
874     analogWrite( pin, 255 );
875 }
876
877 void pwm_set_duty( uint32_t pin, uint32_t duty )
878 {
879     analogWrite( pin, duty );
880 }
881
882 void pwm_get_duty( uint32_t pin, uint32_t *duty )
883 {
884     *duty = analogRead( pin );
885 }
886
887 void pwm_set_resolution( int res )
888 {
889     pwm_resolution_nbit = res;
890 }
891
892 void pwm_set_clockA_freq( uint32_t val )
893 {
894     pwm_clockA_freq = val;
895 }
896
897 void pwm_set_clockB_freq( uint32_t val )
898 {
899     pwm_clockB_freq = val;
900 }
901
902 void pwm_stop( uint32_t pin )
903 {
904     analogWrite( pin, 0 );
905 }
906
907 void pwm_start( uint32_t pin )
908 {
909     analogWrite( pin, 255 );
910 }
911
912 void pwm_set_duty( uint32_t pin, uint32_t duty )
913 {
914     analogWrite( pin, duty );
915 }
916
917 void pwm_get_duty( uint32_t pin, uint32_t *duty )
918 {
919     *duty = analogRead( pin );
920 }
921
922 void pwm_set_resolution( int res )
923 {
924     pwm_resolution_nbit = res;
925 }
926
927 void pwm_set_clockA_freq( uint32_t val )
928 {
929     pwm_clockA_freq = val;
930 }
931
932 void pwm_set_clockB_freq( uint32_t val )
933 {
934     pwm_clockB_freq = val;
935 }
936
937 void pwm_stop( uint32_t pin )
938 {
939     analogWrite( pin, 0 );
940 }
941
942 void pwm_start( uint32_t pin )
943 {
944     analogWrite( pin, 255 );
945 }
946
947 void pwm_set_duty( uint32_t pin, uint32_t duty )
948 {
949     analogWrite( pin, duty );
950 }
951
952 void pwm_get_duty( uint32_t pin, uint32_t *duty )
953 {
954     *duty = analogRead( pin );
955 }
956
957 void pwm_set_resolution( int res )
958 {
959     pwm_resolution_nbit = res;
960 }
961
962 void pwm_set_clockA_freq( uint32_t val )
963 {
964     pwm_clockA_freq = val;
965 }
966
967 void pwm_set_clockB_freq( uint32_t val )
968 {
969     pwm_clockB_freq = val;
970 }
971
972 void pwm_stop( uint32_t pin )
973 {
974     analogWrite( pin, 0 );
975 }
976
977 void pwm_start( uint32_t pin )
978 {
979     analogWrite( pin, 255 );
980 }
981
982 void pwm_set_duty( uint32_t pin, uint32_t duty )
983 {
984     analogWrite( pin, duty );
985 }
986
987 void pwm_get_duty( uint32_t pin, uint32_t *duty )
988 {
989     *duty = analogRead( pin );
990 }
991
992 void pwm_set_resolution( int res )
993 {
994     pwm_resolution_nbit = res;
995 }
996
997 void pwm_set_clockA_freq( uint32_t val )
998 {
999     pwm_clockA_freq = val;
1000 }
1001
1002 void pwm_set_clockB_freq( uint32_t val )
1003 {
1004     pwm_clockB_freq = val;
1005 }
1006
1007 void pwm_stop( uint32_t pin )
1008 {
1009     analogWrite( pin, 0 );
1010 }
1011
1012 void pwm_start( uint32_t pin )
1013 {
1014     analogWrite( pin, 255 );
1015 }
1016
1017 void pwm_set_duty( uint32_t pin, uint32_t duty )
1018 {
1019     analogWrite( pin, duty );
1020 }
1021
1022 void pwm_get_duty( uint32_t pin, uint32_t *duty )
1023 {
1024     *duty = analogRead( pin );
1025 }
1026
1027 void pwm_set_resolution( int res )
1028 {
1029     pwm_resolution_nbit = res;
1030 }
1031
1032 void pwm_set_clockA_freq( uint32_t val )
1033 {
1034     pwm_clockA_freq = val;
1035 }
1036
1037 void pwm_set_clockB_freq( uint32_t val )
1038 {
1039     pwm_clockB_freq = val;
1040 }
1041
1042 void pwm_stop( uint32_t pin )
1043 {
1044     analogWrite( pin, 0 );
1045 }
1046
1047 void pwm_start( uint32_t pin )
1048 {
1049     analogWrite( pin, 255 );
1050 }
1051
1052 void pwm_set_duty( uint32_t pin, uint32_t duty )
1053 {
1054     analogWrite( pin, duty );
1055 }
1056
1057 void pwm_get_duty( uint32_t pin, uint32_t *duty )
1058 {
1059     *duty = analogRead( pin );
1060 }
1061
1062 void pwm_set_resolution( int res )
1063 {
1064     pwm_resolution_nbit = res;
1065 }
1066
1067 void pwm_set_clockA_freq( uint32_t val )
1068 {
1069     pwm_clockA_freq = val;
1070 }
1071
1072 void pwm_set_clockB_freq( uint32_t val )
1073 {
1074     pwm_clockB_freq = val;
1075 }
1076
1077 void pwm_stop( uint32_t pin )
1078 {
1079     analogWrite( pin, 0 );
1080 }
1081
1082 void pwm_start( uint32_t pin )
1083 {
1084     analogWrite( pin, 255 );
1085 }
1086
1087 void pwm_set_duty( uint32_t pin, uint32_t duty )
1088 {
1089     analogWrite( pin, duty );
1090 }
1091
1092 void pwm_get_duty( uint32_t pin, uint32_t *duty )
1093 {
1094     *duty = analogRead( pin );
1095 }
1096
1097 void pwm_set_resolution( int res )
1098 {
1099     pwm_resolution_nbit = res;
1100 }
1101
1102 void pwm_set_clockA_freq( uint32_t val )
1103 {
1104     pwm_clockA_freq = val;
1105 }
1106
1107 void pwm_set_clockB_freq( uint32_t val )
1108 {
1109     pwm_clockB_freq = val;
1110 }
1111
1112 void pwm_stop( uint32_t pin )
1113 {
1114     analogWrite( pin, 0 );
1115 }
1116
1117 void pwm_start( uint32_t pin )
1118 {
1119     analogWrite( pin, 255 );
1120 }
1121
1122 void pwm_set_duty( uint32_t pin, uint32_t duty )
1123 {
1124     analogWrite( pin, duty );
1125 }
1126
1127 void pwm_get_duty( uint32_t pin, uint32_t *duty )
1128 {
1129     *duty = analogRead( pin );
1130 }
1131
1132 void pwm_set_resolution( int res )
1133 {
1134     pwm_resolution_nbit = res;
1135 }
1136
1137 void pwm_set_clockA_freq( uint32_t val )
1138 {
1139     pwm_clockA_freq = val;
1140 }
1141
1142 void pwm_set_clockB_freq( uint32_t val )
1143 {
1144     pwm_clockB_freq = val;
1145 }
1146
1147 void pwm_stop( uint32_t pin )
1148 {
1149     analogWrite( pin, 0 );
1150 }
1151
1152 void pwm_start( uint32_t pin )
1153 {
1154     analogWrite( pin, 255 );
1155 }
1156
1157 void pwm_set_duty( uint32_t pin, uint32_t duty )
1158 {
1159     analogWrite( pin, duty );
1160 }
1161
1162 void pwm_get_duty( uint32_t pin, uint32_t *duty )
1163 {
1164     *duty = analogRead( pin );
1165 }
1166
1167 void pwm_set_resolution( int res )
1168 {
1169     pwm_resolution_nbit = res;
1170 }
1171
1172 void pwm_set_clockA_freq( uint32_t val )
1173 {
1174     pwm_clockA_freq = val;
1175 }
1176
1177 void pwm_set_clockB_freq( uint32_t val )
1178 {
1179     pwm_clockB_freq = val;
1180 }
1181
1182 void pwm_stop( uint32_t pin )
1183 {
1184     analogWrite( pin, 0 );
1185 }
1186
1187 void pwm_start( uint32_t pin )
1188 {
1189     analogWrite( pin, 255 );
1190 }
1191
1192 void pwm_set_duty( uint32_t pin, uint32_t duty )
1193 {
1194     analogWrite( pin, duty );
1195 }
1196
1197 void pwm_get_duty( uint32_t pin, uint32_t *duty )
1198 {
1199     *duty = analogRead( pin );
1200 }
1201
1202 void pwm_set_resolution( int res )
1203 {
1204     pwm_resolution_nbit = res;
1205 }
1206
1207 void pwm_set_clockA_freq( uint32_t val )
1208 {
1209     pwm_clockA_freq = val;
1210 }
1211
1212 void pwm_set_clockB_freq( uint32_t val )
1213 {
1214     pwm_clockB_freq = val;
1215 }
1216
1217 void pwm_stop( uint32_t pin )
1218 {
1219     analogWrite( pin, 0 );
1220 }
1221
1222 void pwm_start( uint32_t pin )
1223 {
1224     analogWrite( pin, 255 );
1225 }
1226
1227 void pwm_set_duty( uint32_t pin, uint32_t duty )
1228 {
1229     analogWrite( pin, duty );
1230 }
1231
1232 void pwm_get_duty( uint32_t pin, uint32_t *duty )
1233 {
1234     *duty = analogRead( pin );
1235 }
1236
1237 void pwm_set_resolution( int res )
1238 {
1239     pwm_resolution_nbit = res;
1240 }
1241
1242 void pwm_set_clockA_freq( uint32_t val )
1243 {
1244     pwm_clockA_freq = val;
1245 }
1246
1247 void pwm_set_clockB_freq( uint32_t val )
1248 {
1249     pwm_clockB_freq = val;
1250 }
1251
1252 void pwm_stop( uint32_t pin )
1253 {
1254     analogWrite( pin, 0 );
1255 }
1256
1257 void pwm_start( uint32_t pin )
1258 {
1259     analogWrite( pin, 255 );
1260 }
1261
1262 void pwm_set_duty( uint32_t pin, uint32_t duty )
1263 {
1264     analogWrite( pin, duty );
1265 }
1266
1267 void pwm_get_duty( uint32_t pin, uint32_t *duty )
1268 {
1269     *duty = analogRead( pin );
1270 }
1271
1272 void pwm_set_resolution( int res )
1273 {
1274     pwm_resolution_nbit = res;
1275 }
1276
1277 void pwm_set_clockA_freq( uint32_t val )
1278 {
1279     pwm_clockA_freq = val;
1280 }
1281
1282 void pwm_set_clockB_freq( uint32_t val )
1283 {
1284     pwm_clockB_freq = val;
1285 }
1286
1287 void pwm_stop( uint32_t pin )
1288 {
1289     analogWrite( pin, 0 );
1290 }
1291
1292 void pwm_start( uint32_t pin )
1293 {
1294     analogWrite( pin, 255 );
1295 }
1296
1297 void pwm_set_duty( uint32_t pin, uint32_t duty )
1298 {
1299     analogWrite( pin, duty );
1300 }
1301
1302 void pwm_get_duty( uint32_t pin, uint32_t *duty )
1303 {
1304     *duty = analogRead( pin );
1305 }
1306
1307 void pwm_set_resolution( int res )
1308 {
1309     pwm_resolution_nbit = res;
1310 }
1311
1312 void pwm_set_clockA_freq( uint32_t val )
1313 {
1314     pwm_clockA_freq = val;
1315 }
1316
1317 void pwm_set_clockB_freq( uint32_t val )
1318 {
1319     pwm_clockB_freq = val;
1320 }
1321
1322 void pwm_stop( uint32_t pin )
1323 {
1324     analogWrite( pin, 0 );
1325 }
1326
1327 void pwm_start( uint32_t pin )
1328 {
1329     analogWrite( pin, 255 );
1330 }
1331
1332 void pwm_set_duty( uint32_t pin, uint32_t duty )
1333 {
1334     analogWrite( pin, duty );
1335 }
1336
1337 void pwm_get_duty( uint32_t pin, uint32_t *duty )
1338 {
1339     *duty = analogRead( pin );
1340 }
1341
1342 void pwm_set_resolution( int res )
1343 {
1344     pwm_resolution_nbit = res;
1345 }
1346
1347 void pwm_set_clockA_freq( uint32_t val )
1348 {
1349     pwm_clockA_freq = val;
1350 }
1351
1352 void pwm_set_clockB_freq( uint32_t val )
1353 {
1354     pwm_clockB_freq = val;
1355 }
1356
1357 void pwm_stop( uint32_t pin )
1358 {
1359     analogWrite( pin, 0 );
1360 }
1361
1362 void pwm_start( uint32_t pin )
1363 {
1364     analogWrite( pin, 255 );
1365 }
1366
1367 void pwm_set_duty( uint32_t pin, uint32_t duty )
1368 {
1369     analogWrite( pin, duty );
1370 }
1371
1372 void pwm_get_duty( uint32_t pin, uint32_t *duty )
1373 {
1374     *duty = analogRead( pin );
1375 }
1376
1377 void pwm_set_resolution( int res )
1378 {
1379     pwm_resolution_nbit = res;
1380 }
1381
1382 void pwm_set_clockA_freq( uint32_t val )
1383 {
1384     pwm_clockA_freq = val;
1385 }
1386
1387 void pwm_set_clockB_freq( uint32_t val )
1388 {
1389     pwm_clockB_freq = val;
1390 }
1391
1392 void pwm_stop( uint32_t pin )
1393 {
1394     analogWrite( pin, 0 );
1395 }
1396
1397 void pwm_start( uint32_t pin )
1398 {
1399     analogWrite( pin, 255 );
1400 }
1401
1402 void pwm_set_duty( uint32_t pin, uint32_t duty )
1403 {
1404     analogWrite( pin, duty );
1405 }
1406
1407 void pwm_get_duty( uint32_t pin, uint32_t *duty )
1408 {
1409     *duty = analogRead( pin );
1410 }
1411
1412 void pwm_set_resolution( int res )
1413 {
1414     pwm_resolution_nbit = res;
1415 }
1416
1417 void pwm_set_clockA_freq( uint32_t val )
1418 {
1419     pwm_clockA_freq = val;
1420 }
1421
1422 void pwm_set_clockB_freq( uint32_t val )
1423 {
1424     pwm_clockB_freq = val;
1425 }
1426
1427 void pwm_stop( uint32_t pin )
1428 {
1429     analogWrite( pin, 0 );
1430 }
1431
1432 void pwm_start( uint32_t pin )
1433 {
1434     analogWrite( pin, 255 );
1435 }
1436
1437 void pwm_set_duty( uint32_t pin, uint32_t duty )
1438 {
1439     analogWrite( pin, duty );
1440 }
1441
1442 void pwm_get_duty( uint32_t pin, uint32_t *duty )
1443 {
1444     *duty = analogRead( pin );
1445 }
1446
1447 void pwm_set_resolution( int res )
1448 {
1449     pwm_resolution_nbit = res;
1450 }
1451
1452 void pwm_set_clockA_freq( uint32_t val )
1453 {
1454     pwm_clockA_freq = val;
1455 }
1456
1457 void pwm_set_clockB_freq( uint32_t val )
1458 {
1459     pwm_clockB_freq = val;
1460 }
1461
1462 void pwm_stop( uint32_t pin )
1463 {
1464     analogWrite( pin, 0 );
1465 }
1466
1467 void pwm_start( uint32_t pin )
1468 {
1469     analogWrite( pin, 255 );
1470 }
1471
1472 void pwm_set_duty( uint32_t pin, uint32_t duty )
1473 {
1474     analogWrite
```

```

47 {
48     pwm_clockA_freq = val;
49 }
50
51
52 void pwm_set_clockB_freq(uint32_t val)
53 {
54     pwm_clockB_freq = val;
55 }
56
57
58 static inline uint32_t mapResolution(uint32_t value, uint32_t from
59 , uint32_t to)
60 {
61     if (from == to)
62         return value;
63     if (from > to)
64         return value >> (from-to);
65     else
66         return value << (to-from);
67
68
69
70 // MAIN PWM INITIALIZATION
71 //_____
72 void pwm_setup( uint32_t pwm_pin, uint32_t pwm_freq, int
73 iclock )
74 {
75     uint32_t pwm_duty = 0;
76     uint32_t chan = g_APinDescription[pwm_pin].ulPWMChannel;
77
78     // SET CLOCK FREQUENCY
79     //_____
80     if (iclock==1) pwm_set_clockA_freq( pwm_max_duty_Ncount*
pwm_freq );
81     if (iclock==2) pwm_set_clockB_freq( pwm_max_duty_Ncount*
pwm_freq );
82
83     if (pwm_pin>=6 && pwm_pin<=9)
84     {
85         // SET PWM RESOLUTION
86         //_____
87         pwm_duty = mapResolution( pwm_duty, pwm_resolution_nbit ,
PWMRESOLUTION );
88
89         // PWM STARTUP AND SETUP CLOCK
90         //_____
91         pmc_enable_periph_clk( PWM_INTERFACE_ID );
92         PWMC_ConfigureClocks( pwm_clockA_freq , pwm_clockB_freq ,
VARIANT_MCK );
93
94         // SETUP PWM FOR pwm_pin
95         //_____

```

```

96     PIO_Configure( g_APinDescription[pwm_pin].pPort ,
97     g_APinDescription[pwm_pin].ulPinType ,   g_APinDescription[
98     pwm_pin ].ulPin ,   g_APinDescription[pwm_pin ].ulPinConfiguration
99     );
100    if (iclock==1)  PWMC_ConfigureChannel(PWM_INTERFACE, chan ,
101        PWM_CMR_CPRE_CLKA, 0, 0);
102    if (iclock==2)  PWMC_ConfigureChannel(PWM_INTERFACE, chan ,
103        PWM_CMR_CPRE_CLKB, 0, 0);
104    PWMC_SetPeriod(PWM_INTERFACE, chan , pwm_max_duty_Ncount);
105    PWMC_SetDutyCycle(PWM_INTERFACE, chan , pwm_duty);
106    PWMC_EnableChannel(PWM_INTERFACE, chan );
107 }
108 // WRITE DUTY CYCLE
109 //_____
110 void  pwm_write_duty( uint32_t  pwm_pin ,  uint32_t  pwm_duty )
111 {
112     if (pwm_pin>=6 && pwm_pin<=9)
113     {
114         pwm_duty = mapResolution( pwm_duty ,  pwm_resolution_nbit ,
115             PWM_RESOLUTION );
116         uint32_t  chan = g_APinDescription[pwm_pin].ulPWMChannel;
117         PWMC_SetDutyCycle(PWM_INTERFACE, chan ,  pwm_duty );
118     }
119 }
120 };
121
122
123 // FORCE PWM STOP
124 //_____
125
126 void  pwm_stop( uint32_t  pwm_pin )
127 {
128     pinMode(pwm_pin , OUTPUT);           // sets the digital pin as
129     output
130     digitalWrite(pwm_pin , LOW);        // sets the LED off
131 };

```