Hochschule RheinMain
Fachbereich Design Informatik Medien
Prof. Dr. Robert Kaiser

Echtzeitverarbeitung

SS 2022
LV 4511 / LV 8481

Übungsblatt 6
Laborversuch
Abgabe: 13. Woche (06.07.2022)

Aufgabe 6.1. (Praktikumsumgebung):

In den folgenden Aufgaben soll ein Schrittmotor über das EchtzeitHat betrieben werden. Ein Schrittmotor ist ein Synchronmotor und unterscheidet sich von normalen Gleichstrommotoren, wie Sie sie bisher eingesetzt haben. Der Rotor (drehendes mittleres Teil) besteht aus Permanentmagneten und es gibt mehrere Spulen die außen herum angeordnet sind. Über die Schaltung dieser (Strator-) Spulen wird das Magnetfeld verändert und der Rotor dreht sich weiter. Es gibt bipolare und unipolare Motoren. Die bipolaren Motoren haben vier Anschlüsse über die die Polarität von zwei Spulen gesteuert werden kann. Bei unipolaren Motoren ist in der Mitte der Spule noch jeweils eine Erdungsleitung ausgeführt. Das Prinzip für einen bipolaren Motor ist in Abbildung 1 dargestellt. Je nach Schaltung der Spule ist das Magnetfeld an einer anderen Stelle. Der Magnet des Rotors folgt diesem und bewegt sich weiter. Er macht einen „Schritt“ (daher der Name …).

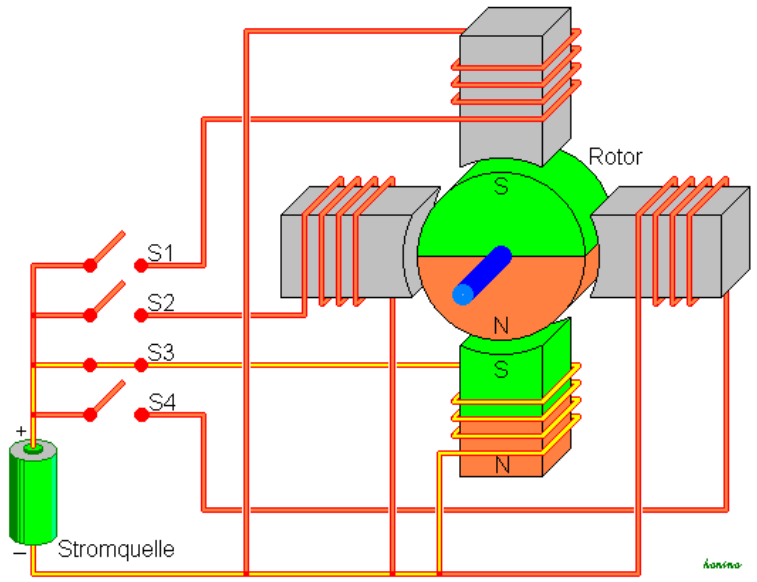


Abbildung 1: Schema eines Schrittmotors mit vier Schritten (unipolar) (Quelle: wikipedia [4])

Die Spulen können nach verschiedenen Verfahren angesteuert werden. Wird immer nur eine Spule mit Strom versorgt, heißt dies „WaveDrive“-Betrieb. Dies ist relativ einfach umzusetzen. Wenn

hingegen immer zwei benachbarte Spulen angesteuert werden, heißt das „Vollschrittbetrieb“. Hierbei ist das mögliche Drehmoment größer, daher wird diese Betriebsart in der Praxis häufiger verwendet. Die Position des Rotors liegt dann immer zwischen den Spulen. Im sogenannten „Halbschrittbetrieb“ werden beide Ansteuerungen kombiniert. Eine Umdrehung hat damit acht Schritte.

Nach dem Schema in Abbildung 1 hätte der Motor nur vier (Voll-) Schritte pro Umdrehung. In der Praxis haben Schrittmotoren mehr Schritte. 200 Schritte pro Umdrehung (1,8°) sind gebräuchlich. Hierzu werden mehrere Stratorspulen um den Rotor und Magnete am Rotor verwendet. Im Halbschrittbetrieb verdoppelt sich weiterhin die Anzahl an möglichen Schritten.

Es gibt eine Reihe von Vorteilen von Schrittmotoren: Über die explizite Schaltung der Elektromagnete kann der Rotor, ohne Positionsrückmeldung von Sensoren, exakt zu einer bestimmten Position gedreht werden. Des Weiteren hält der Motor eine bestimmte Position, auch gegen größere Kräfte, wenn die Spulen mit Energie versorgt werden.

Der Nachteil von Schrittmotoren ist die aufwändigere Ansteuerung, und dass sie prinzipbedingt nicht richtig „rund laufen“. Des Weiteren ist die maximale Drehgeschwindigkeit begrenzt. Bei hohen Geschwindigkeiten kann es zudem vorkommen, dass Schritte verloren gehen.

Es gibt spezielle Treiberbauteile für Schrittmotoren, die die Ansteuerung übernehmen. Hier muss nur die Richtung und der einzelne Schritt signalisiert werden. Für diese Aufgabe werden jedoch die beiden H-Brücken des EchtzeitHat verwendet. Es ist daher notwendig die Spulen in der richtigen Reihenfolge in der gewünschten Frequenz an- und auszuschalten, bzw. die Polarität über die Stromrichtung zu steuern.

Wenn die H-Brücke geschaltet ist, dann fließt kontinuierlich Strom durch den Motor, nur begrenzt durch den Innenwiderstand der Statorwicklung. Im Vollschritt auch durch mehrere Spulen. Wenn der Motor sich dreht, ist das nicht weiter schlimm, denn je nach Geschwindigkeit sind die Spulen ja zeitweise auch stromlos. Wenn allerdings nach Programmende die H-Brücke im geschalteten Zustand verbleibt, führt das dazu, dass diese und der Spanungswandler auf dem EchtzeitHat recht schnell sehr heiß werden (Verbrennungsgefahr!). Es ist daher wichtig, dass nach Programmende die GPIOs der H-Brücke auf 0 (LOW) gesetzt werden. Für den Fall, dass das Programm über ein Signal abgebrochen wird, ist es sinnvoll dies in einem Signal-Handler zu machen! (LMGTFY: [1] [2])

Die notwendige Ansteuerung für die Spulen finden Sie in den folgenden zwei Tabellen für Voll- und für Halbschritte (entnommen von [3]). Da die in dieser Aufgabe verwendeten Schrittmotoren 200 Schritte für eine Umdrehung benötigen, kommt es nicht so darauf an, wo die Sequenz begonnen wird (der erste Schritt ist ggf. etwas größer). Danach sollte allerdings immer in der richtigen Sequenz vor- und zurückgegangen werden.

In der Tabelle ist die notwendige Polarität für die jeweilige Spule angegeben. Wie weiter unten beschrieben wird, können Sie die Richtung der H-Brücke mit drei GPIOs steuern oder den Motor

anhalten. Für den „Unbelegt“-Fall setzen Sie am besten die jeweiligen GPIOs auf LOW (0).

Bipolare Ansteuerung Vollschritt

	Phase Spule 1a	Phase Spule 1b	Phase Spule 2a	Phase Spule 2b
Schritt 1	+	-	+	-
Schritt 2	+	-	-	+
Schritt 3	-	+	-	+
Schritt 4	-	+	+	-

- steht für Minus | + steht für Plus | 0 steht für Unbelegt

Bipolare Ansteuerung Halbschritt

	Phase Spule 1a	Phase Spule 1b	Phase Spule 2a	Phase Spule 2b
Schritt 1	+	-	+	-
Schritt 2	+	-	0	0
Schritt 3	+	-	-	+
Schritt 4	0	0	-	+
Schritt 5	-	+	-	+
Schritt 6	-	+	0	0
Schritt 7	-	+	+	-
Schritt 8	0	0	+	-

- steht für Minus | + steht für Plus | 0 steht für Unbelegt

Hinweise:

(a) Es wird der H-Brücken-IC TB6612FNG [7] verwendet. Der Schaltplan des EchtzeitHat ist in 2 gegeben.

(b) Es können zwei Motoren angeschlossen werden. Für die Ansteuerung eines Motors werden drei! GPIOs benötigt. Am Beispiel vom “DRV_A”:

- GPIO5: “Enable” schaltet den Strom für die H-Brücke ein
- GPIO6: “IN1” steuert die Richtung des Motors
- GPIO12: “IN2” steuert die Richtung des Motors

(c) Die Logik für die Ansteuerung ist:

- EN = High, IN1 = High, IN2 = Low => Vorwärts
- EN = High, IN1 = Low, IN2 = High => Rückwerts
- EN = High, IN1 = IN2 = High => Schneller Motor-Stop
- EN = High, IN1 = IN2 = Low => Motor-Freilauf, Motor-Stop

(d) Siehe auch Ausschnitt des Datenblatts des TB6612FNG 3

Aufgabe 6.2. (Stepper ansteuern):

Schreiben Sie ein C-Programm mit dem Sie den Schrittmotor 1x um die Achse drehen lassen. Verwenden Sie zuerst Vollschritte und erweitern Sie danach das Programm auf eine Halbschritt-ansteuerung.

Denken Sie daran, sicherzustellen, dass nach dem Beenden Ihres Programms die H-Brücke deaktiviert wird (alle GPIOs auf 0 gesetzt). Ansonsten werden H-Brücke und Spannungsregler recht schnell, recht heiß.

Wählen Sie für den Anfang eine Zeitspanne von 80 ms zwischen den Schritten. Sie können folgenden Code für das Schlafenlegen des Threads bzw. die Verwendung im Programm nutzen:

```
static void sleep_util(struct timespec* ts, int delay) {
    ts->tv_nsec += (delay * 1000);
    if (ts->tv_nsec >= 1000 * 1000 * 1000) {
        ts->tv_nsec -= 1000 * 1000 * 1000;
        ts->tv_sec++;
    }
    clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, ts, NULL);
}

// im Programm 1x anlegen und zwischen den Schritten referenzieren
struct timespec ts;
clock_gettime(CLOCK_MONOTONIC, &ts);

// delay zwischen den Schritten
int delay = <>;

// zwischen den Schritten aufrufen
sleep_util(&ts, delay);
```

(Hinweis: Wenn Sie den Programmcode aus dem PDF kopieren, überprüfen Sie die Zeichen genau. Latex ersetzt häufig die normalen Zeichen durch „besser aussehende“, die allerdings kein ASCII sind …)

Reduzieren Sie langsam die Zeitabstände zwischen den einzelnen Schritten. Ab wann gibt es Probleme? Lassen Sie ggf. auch hochpriore Programme im Hintergrund laufen. Welche Auswirkungen hat dies?

Aufgabe 6.3. (RT-Erweiterung für Stepper):

Wenden Sie Ihre Kenntnisse aus den vorigen Übungsblättern bezüglich RT-POSIX auf das Programm an. Wählen Sie eine hohe Priorität für ihren Thread. Wie klein können Sie den Pausen zwischen den Schritten wählen, ohne dass Schritte übersprungen werden?

Aufgabe 6.4. (Tacho für Systemauslastung):

Schrittmotoren werden inzwischen häufig auch im Armaturenbrett von Autos zur Anzeige von Geschwindigkeit und Drehzahl verwendet (i.d.R. mit noch mehr Schritten).

Schreiben Sie ein Programm, das die Systemauslastung des Raspberry Pi mit dem Schrittmotor im Winkel zwischen 0 und 180 Grad darstellt.

Verwenden Sie zwei Threads für die Steuerung des Schrittmotors und die sekündliche Ermittlung der Systemauslastung.

Sie können folgendes Snippet für die Ermittlung der Systemauslastung verwenden:

```
FILE* procFile = NULL;
unsigned long long  last_jiffies_work = 0;
unsigned long long last_jiffies_idle = 0;

// TODO: Beginn schleife

procFile = fopen("/proc/stat", "r");
if (procFile == NULL) {
    // fehlerbehandlung
}
unsigned long long jiffies_user, jiffies_nice, jiffies_system, jiffies_idle;

int ret = fscanf(procFile, "%*s %llu %llu %llu %llu", &jiffies_user,
    &jiffies_nice, &jiffies_system, &jiffies_idle );

// TODO: fehlerbehandlung

fclose(procFile);

unsigned long long jiffies_sum = jiffies_user + jiffies_nice + jiffies_system;

printf("Gesamt auslastung: %llu \n",
    jiffies_sum *100 /(jiffies_sum+jiffies_idle));

unsigned long long diff_jiffies_active = jiffies_sum - last_jiffies_work;
unsigned long long diff_jiffies_idle = jiffies_idle - last_jiffies_idle;

printf("Auslastung letzte Sekunde: %llu \n",
    diff_jiffies_active*100/(diff_jiffies_active + diff_jiffies_idle));

last_jiffies_work = jiffies_sum;
last_jiffies_idle = jiffies_idle;

// TODO: 1 Sekunde schlafen
```

Stellen Sie sicher, dass keine Schritte verloren gehen.

A. (*):

Literatur

[1] http://openbook.rheinwerk-verlag.de/c_von_a_bis_z/020_c_headerdateien_007.htm

[2] https://airtower.wordpress.com/2010/06/16/catch-sigterm-exit-gracefully/

[3] http://rn-wissen.de/wiki/index.php/Schrittmotoren

[4] http://commons.wikimedia.org/wiki/File:Schrittmotor.PNG

[5] http://wiringpi.com/

[6] http://wiringpi.com/reference/

[7] https://toshiba.semicon-storage.com/us/product/linear/motordriver/detail.TB6612FNG.html

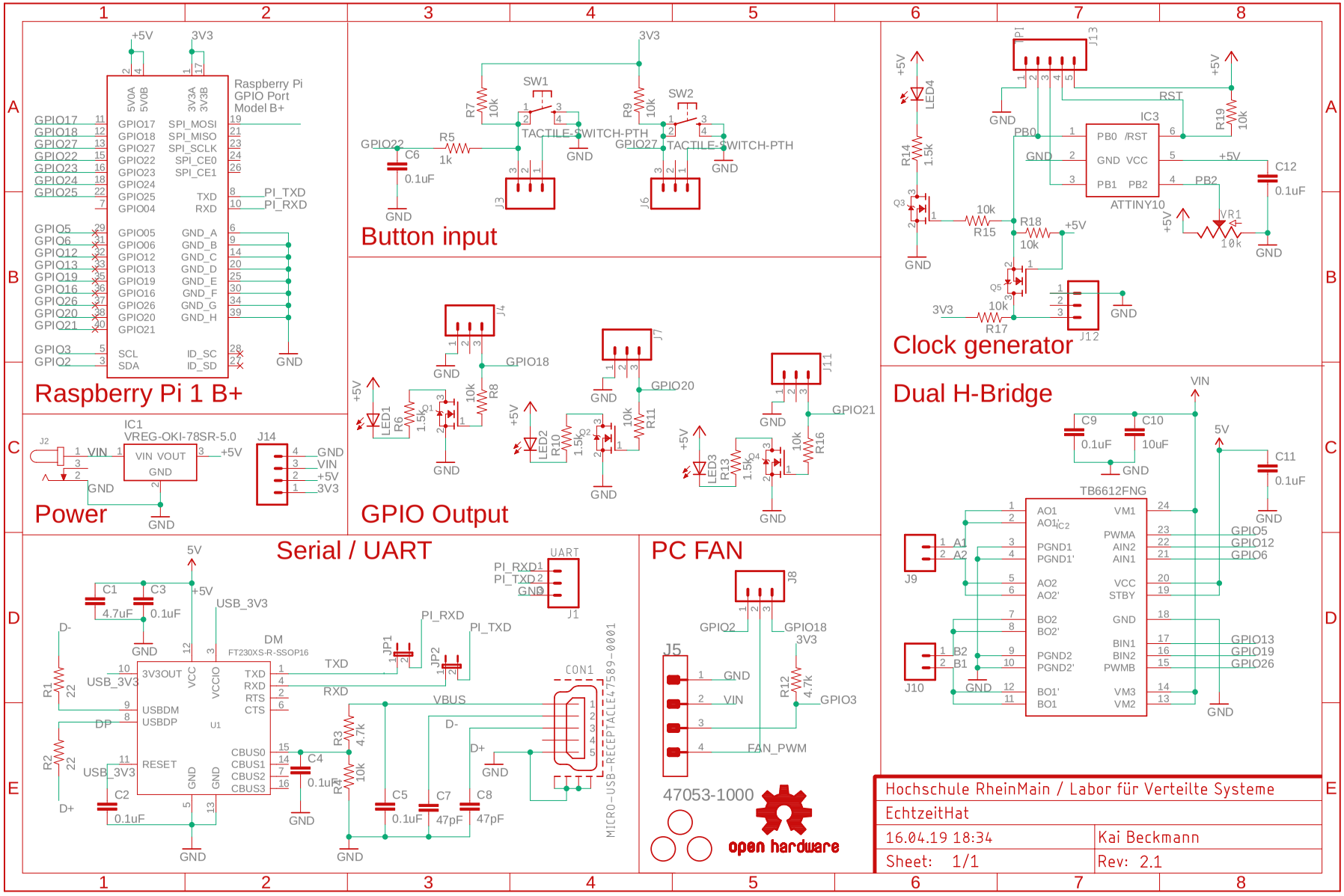
B. (Raspberry Pi GPIO Pins):

Raspberry Pi – GPIO-connector

HAT	WiringPi	GPIO	Name	Header		Name	GPIO	WiringPi	HAT
			3.3V	1	2	5V			
FanSoftPWM	8	2	SDA	3	4	5V			
Fan Tacho	9	3	SCL	5	6	GND			
	7	4		7	8	TxD	14	15	
			GND	9	10	RxD	15	16	
	0	17		11	12		18	1	PWM/GPIO18
SW2	2	27		13	14	GND			
SW1	3	22		15	16		23	4	
			3.3V	17	18		24	5	
	12	10	MOSI	19	20	GND			
	13	9	MISO	21	22		25	6	
	14	11	SCLK	23	24	CE0	8	10	
			GND	25	26	CE1	7	11	
				27	28				
DRV_A_en	21	5		29	30				
DRV_A_in1		6		31	32		12	26	DRV_A_in2
DRV_B_in1	23	13		33	34				
DRV_B_in2	24	19	MISO	35	36				
DRV_B_en	25	26		37	38	MOSI	20	28	GPIO20
			GND	39	40	SCL	21	29	GPIO21

C. (EchtzeitHat Schaltplan):

D. (TB6612FNG Logik):



16.04.19 18:35 f=0.96 /users/k_beckma/ownCloud/EAGLE/projects/EchtzeitHat/EchtzeitHat.sch (Sheet: 1/1)

Abbildung 2: RPiRTHat Schaltplan

H-SW Control Function

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

H-SW Operating Description

・To prevent penetrating current, dead time t2 and t4 is provided in switching to each mode in the IC.

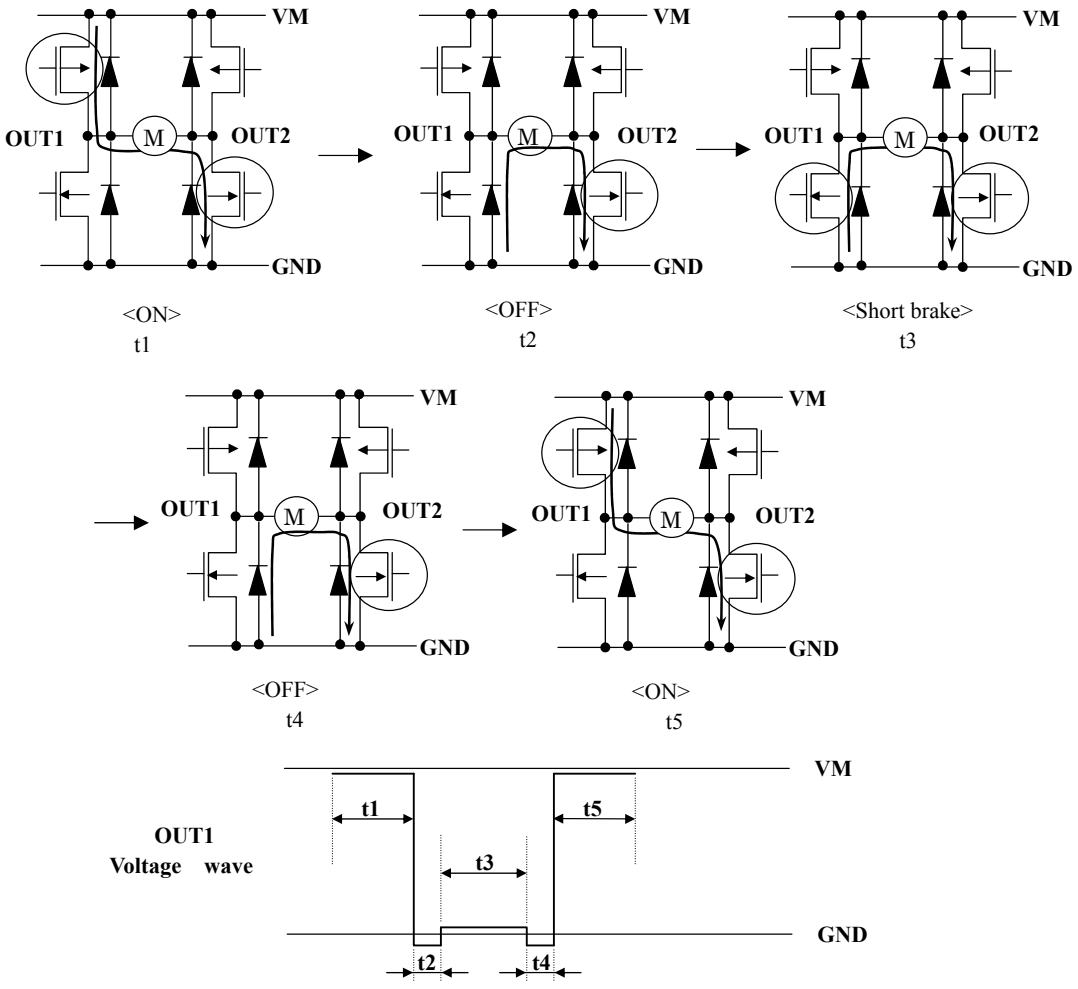


Abbildung 3: TB6612FNG Logik