AN205382

F²MC-16FX, Motor Driver With IRF7389

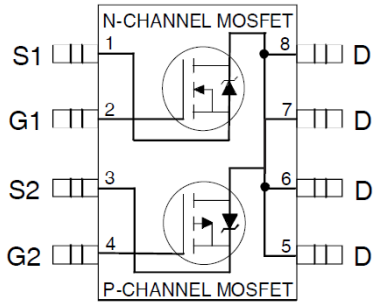
This application note gives an idea how to simply drive a DC motor with two IRF7389 ICs as motor bridge. Goals are cheap small motor drivers that support left/right turn of DC motors and speed control via PWM.

Contents

1 Introduction

1.1 Overview

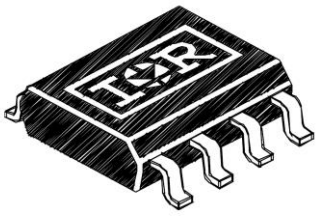
This application note gives an idea how to simply drive a DC motor with two IRF7389 ICs as motor bridge. Goals are cheap small motor drivers that support left/right turn of DC motors and speed control via PWM. Following schematic view is given by the datasheet of the IRF7389, which shows two MOSFETS in one package:



Top View

For detailed information see the datasheet (see chapter 4)

Note:
It has to be considered that a motor is an inductive load that produces high voltage- and current-peaks depends on the motor voltage.



The IRF7389 is available as SMD SO-8 package and saves space on a PCB. Heating sinks can be directly designed on the PCB layout

For implementing a PWM signal with a Cypress 16FX MCU there are different possibilities. Some typical hardware macros which are used for PWM generation are the Output Capture Unit (OCU) and the Programmable Pulse Generator (PPG). In this example, the PPG is used to generate a PWM signal.

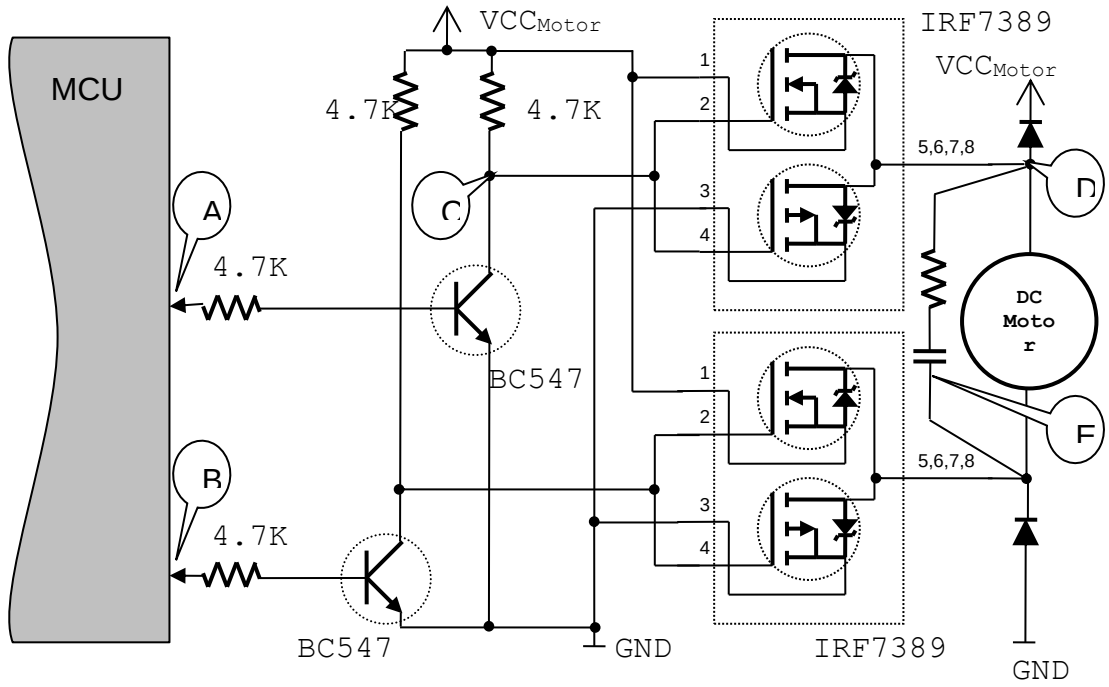
This example should be also easily ported to other Cypress MCU families.

2 Hardware Implementation

2.1 Example Circuit

For a complete motor bridge, two IRF7389 ICs are required. The following circuit gives an idea how to implement a very simple motor bridge. This bridge uses less I/O-pins to drive the DC motor. This has disadvantages while using a PWM to control the motor speed: For every channel or half-bridge, two capacitors have to be charged or discharged while changing the logic value. This means, that only slow PWMs are supported, because higher frequencies increase the dynamic power dissipation, which decreases the efficiency. The datasheet describes values about 1MHz for a MOSFET. In inverter mode, this frequency should be divided by two, so the maximum would be under 500 KHz. With this circuit, the motor can be driven left or right direction. Example:

A=0; B=0	=>	Motor is in stop mode
A=1; B=0	=>	Motor rotates left (A can be used also in slow PWM mode)
A=0; B=1	=>	Motor rotates right (B can be used also in slow PWM mode)
A=1; B=1	=>	Motor is in stop mode

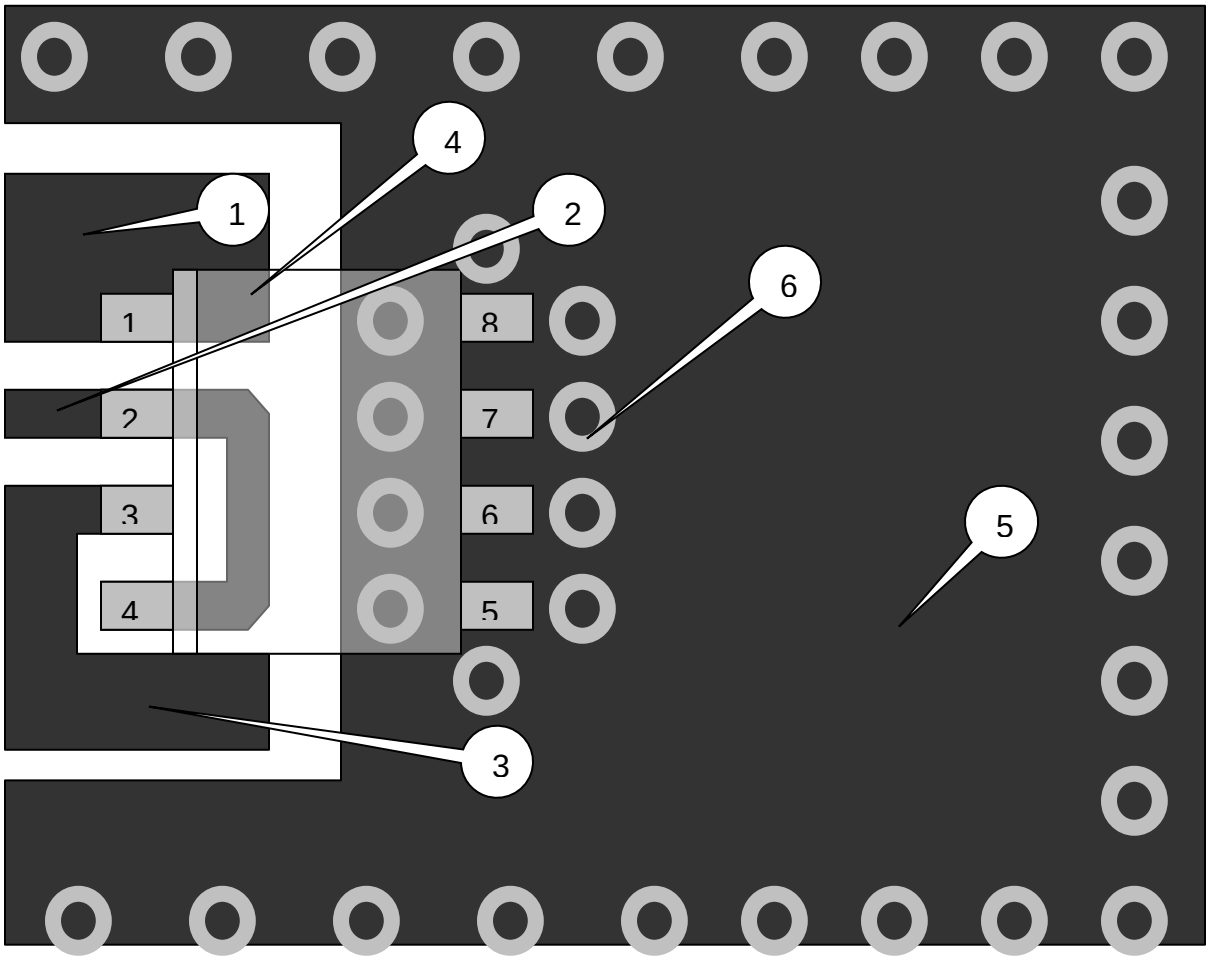


To reduce noise in the motor supply voltage, a RC filter depending of the PWM frequency (E) is added to the Motor. Diodes reminding the risk of destroying the MOSFETs or other semiconductors in the whole circuit including the MCU! In the circuit both IRF7389 are connected as an inverter. If one of the two IRF7389 ICs is considered, every signal that is connected to (C) will be inverted and amplified at (D). To have a good V_{GS} (C), the control pins from the MCU (A) and (B) are preamplifier and inverted by the npn-transistor, pulled up by a 4.7K Ohm resistor to VCC_{Motor}.

2.2 Example PCB Layout

In the following view an aperture of an example PCB layout is shown in top view and shows only one driver IRF7389 (4). The critical components in PCB layout are the heat sink (5) and the voltage wires (1)/(3) to the MOSFET drivers. If possible, the heat sink (5) has to be as big as possible and should use more than one layer. To connect the different heating sink layers, interlayer connections (6) are used.

Pin 2 of the IC (2) is connected to a NPN transistor pulled up by a resistor to VCC-Motor which is not shown in this aperture ((C) in schematic). To have low resistance and afford high currents from power supply, wires from VCC-Motor to Pin 1 (1) should be wide as possible. The same rule should be used while designing wires for the ground Pin 3 (3) of the IC.



1. Motor VCC
2. (C) in schematic
3. GND
4. IC IRF7389 top view
5. Heat sink / (D) in schematic
6. Interlayer connections

3 Software Example

This example will describe how to realize two driven motors controlled by using the PPG macros in a Cypress 16FX MCU. The motors can be controlled by speed and direction.

3.1 Initializations

The PPGs will be configured to be driven by a 48MHz clock. The frequency in this example is about 730Hz and is calculated as followed:

f_{PWM} = (f_{CKSEL} / divider) / (Cycle_Value_Period)

with f_{CKSEL} = 48MHz, divider = 1, Cycle_Value_Period = 0xFFFF

First all required PPGs has to be initialized. Following Registers are used:

- PCSR<n>: used to set the cycle value period (value: 0xFFFF)
- PDUT<n>: used to set the initial duty cycle (value: 0x0000)
- PCNL<n>: PPG Control Status register (Lower Byte) (value: 0x00)
- PCNH<n>: PPG Control Status register (Higher Byte) (value: 0xD0):

CNTE	STGR	MDSE	RTRG	CKS1	CKS0	PGMS	-
1	1	0	1	0	0	0	0
Enable	S-Trigger		Retrigger	divider			

```
void InitPPG0(void)
{
  PCSR0 = 0xFFFF;       // always set cycle value PERIOD 1st
  PDUT0 = 0x0000;       // setduty value DUTY CYCLE
  PCNL0 = 0x00;        // Output disable
  PCNH0 = 0xD0;        // Count enable, S-Trigger, Retrigger, CKSEL divided by 1
}
void InitPPG1(void)
{
  PCSR1 = 0xFFFF;      // always set cycle value PERIOD 1st
  PDUT1 = 0x0000;      // setduty value DUTY CYCLE
  PCNL1 = 0x00;       // Output disable
  PCNH1 = 0xD0;       // Count enable, S-Trigger, Retrigger, CKSEL divided by 1
}
void InitPPG2(void)
{
  PCSR2 = 0xFFFF;      // always set cycle value PERIOD 1st
  PDUT2 = 0x0000;      // setduty value DUTY CYCLE
  PCNL2 = 0x00;       // Output disable
  PCNH2 = 0xD0;       // Count enable, S-Trigger, Retrigger, CKSEL divided by 1
}
void InitPPG3(void)
{
  PCSR3 = 0xFFFF;     // always set cycle value PERIOD 1st
  PDUT3 = 0x0000;    // setduty value DUTY CYCLE
  PCNL3 = 0x00;     // Output disable
  PCNH3 = 0xD0;     // Count enable, S-Trigger, Retrigger, CKSEL divided by 1
}

void InitMotorDrivers(void)
{
  InitPPG0();       //Init PPG0
  InitPPG1();       //Init PPG1
  InitPPG2();       //Init PPG2
  InitPPG3();       //Init PPG3
}
```

3.2 Motor Functions

```
/* setSpeed is used to set duty of the PWM.                                  */
/* - PPGno <1..n>: 1 means PPG0                                              */
/* - level: 0…65535                                                          */
void setSpeed(unsigned char PPGno, unsigned int level)
{
  unsigned char onoff = 0;
  if (level) {            //check if level is 0… if it is 0, disable output
    onoff = 1;
  }
  switch(PPGno) {
    case 1:
      PDUT0 = level;    //PPG0 was chosen, set duty cycle
      PCNL0_OE = onoff;//switch PPG on/off (if level=0 switch it off)
      break;
    case 2:
      PDUT1 = level;   //PPG1 was chosen, set duty cycle
      PCNL1_OE = onoff;//switch PPG on/off (if level=0 switch it off)
      break;
    case 3:
      PDUT2 = level;  //PPG2 was chosen, set duty cycle
      PCNL2_OE = onoff;//switch PPG on/off (if level=0 switch it off)
      break;
    case 4:
      PDUT3 = level;   //PPG3 was chosen, set duty cycle
      PCNL3_OE = onoff;//switch PPG on/off (if level=0 switch it off)
      break;
  }
}

/* setMotor is used to set speed and direction of one of <n> dc motors.    */
/* - motorno <1..n>                                                          */
/* - speed: -32768…+32767 (+/- for direction)                                */

void setMotor (unsigned char motorno, signed short speed) {
    if (speed >= 0) {                              //direction depending
      setSpeed((motorno)*2,(((unsigned int)speed)<<1));//motor<n> = {PPG<n*2-1>,PPG<n*2>}
      setSpeed ((motorno)*2-1,(unsigned int)0);
    } else {
      setSpeed ((motorno)*2,(unsigned int)0);              //motor<n> = {PPG<n*2-1>,PPG<n*2>}
      setSpeed ((motorno)*2-1,(((unsigned int)((-speed))<<1));
    }
}
```

3.3 Sample Program

The following sample program starts the motors in full speed while decreasing the speed of the motors until total stop, changes the direction and increases the speed to full speed and repeats this procedure.

```
void main(void)
{
   signed short int speed = 0;
   InitMotorDrivers();
   for(;;)
   {
     for(speed=-32768;speed < 32768;speed++)
     {
        setMotor(1,speed);
        setMotor(2,speed);
     }
     for(speed=-32768;speed < 32768;speed++)
     {
       setMotor(1,-speed);
       setMotor(2,-speed);
     }
    }
}
```

4 Appendix

4.1 International Rectifier

IRF7389:

http://www.irf.com/product-info/datasheets/data/irf7389.pdf

4.2 References

Cypress Microelectronics

http://www.cypress.com/cypress-microcontrollers

16 FX Series:

http://www.cypress.com/16fx

Additional Application Notes can be found in:

http://www.cypress.com/cypress-mcu-product-softwareexamples

Document History

Document Title: AN205382 - F²MC-16FX, Motor Driver with IRF7389

Document Number: 002-05382

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	09/07/2009	V1.0, MSc, First version
*A	5067999	MKEA	03/29/2016	Converted Spansion Application Note “MCU-AN-300098-E-V10” to Cypress format
*B	5847709	AESATP12	08/24/2017	Updated logo and copyright.
*C	6070279	NOFL	02/18/2018	Sunset Review Migrated to new template Updated links

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709