Example 12.2. Design an interface for an 8-V 200-mA geared DC motor. SW1 will make the motor spin faster and SW2 will make it spin slower.

Solution: We will use the TIP120 circuit as shown in Figure 12.8 because the TIP120 can sink many times the current needed for this motor. We select a +8.4V battery supply and connect it to the positive side of the motor. Just like the stepper motor, when designing with motors we will not use the 3.3V or +5V from the LaunchPad. Although the current is only 200 mA when spinning unloaded, it will increase to 1 A if the motor experiences mechanical friction. The needed base current is

$I_b = I_{coil} / h_{fe}$ = 200mA/900 = 0.22mA

The desired interface resistor.

$R_b \leq (V_{OH} - V_{be})/I_b$ = (3.3-1.3)/0.22mA = 9 kΩ

To cover the variability in h_{fe}, we will use a 1 kΩ resistor instead of the 9 kΩ. The actual voltage across the motor when active will be +8.4-0.7 = 7.7V. Motors and transistors vary a lot, so it is appropriate to experimentally verify the design by measuring the voltages and currents.

Program 12.7 is used to control the motor. Interrupts are triggered on the falling edges of PF4 and PF0. The period of the PWM output is chosen to be about 10 times shorter than the time constant of the motor. The electronic driver will turn on and off every 1ms, but the motor only responds to the average level. The software sets the duty cycle of the PWM to adjust the delivered power. When active, the interface will drive +7.65 V across the motor. The actual current will be a function of the friction applied to the shaft. The software maintains **`H+L`** to be 80,000 (Figure 12.9); this will set the period on PA5 to be a fixed value of 1ms. The duty cycle, **`H/(H+L)`**, will specify the amount of electrical power delivered to the motor.

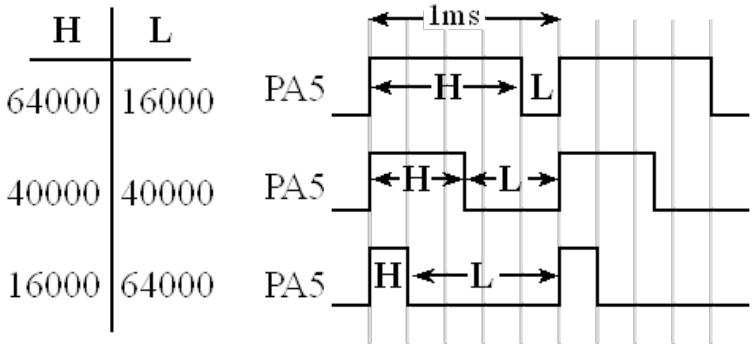

Figure 12.9. The duty cycle of the PWM output sets the power to the motor.

VIDEO 12.4B. DUTY CYCLE FLOWCHART AND TIMING

C12 4b PWM flowcharts YouTube

Help

5:13 / 5:13 1.0x

DR. JONATHAN VALVANO: There are two interrupt service

routines to run on a robot.

The first is a periodic interrupt called SysTick,

and it will generate the pulse width modulated output on PA5.

So we are going to see PA5 look like this, where the time when it's high

is an H and the time when it's low is an L.

The bus clock is running at 80 megahertz.

And this time here is going to be one millisecond.

And so these H plus L, which we saw was a constant,

is going to have to be 80,000 to get the period to be one millisecond.

So what I'm going to do in my SysTick interrupt

is I'm going to look and see what happens.

So in other words, if PA5 is already equal to a 1,

I'm going to set PA5 equal to 0.

Then if PA5 is equal to a 0, I will set PA5 equal to a 1.

That will cause the output to toggle on every interrupt.

And we're going to get an interrupt on each of these times.

The interrupt's not entirely regular, but we're

going to get an interrupt on each of those times.

And the way I'm going to make this work is if the output is a low,

I'm going to set the reload value equal to L minus 1.

And if the output is a high, I'll set the reload value equal to H minus 1.

And the minus 1 is because, as you know, it counts down to 0.

And that is my flow chart for the output.

And this will generate a pulse-width modulated signal on PA5.

I'm going to use the two switches on the LaunchPad,

switch 1 and switch 2, to allow the operator

to select how much power is delivered to the motor.

And I'm going to create an edge-triggered interrupt such

that I get an interrupt on the falling edge of either of these two buttons.

And in order to communicate between this interrupt service routine, which

is edge-triggered, to this one, which is periodic,

I'm going to pass data through global variables H and L. And so if switch 2

has been pressed-- and I will look at its raw interrupt status

to see if that one caused the interrupt.

So what I'm going to do is speed it up by subtracting 8,000 from L.

That's going to change it by 10%.

Down here there will be a calculation of H which is equal to 80,000 minus L.

But in order to make sure it doesn't wrap around,

I'll make sure L is larger than 8,000, in order

to test so it doesn't go all the way to 0.

Help

If it's not switch 2, we'll look at switch 1.

And if its raw interrupt status is set, that

means I got it edge-triggered on switch 1.

And if its H is small enough, then I will add 8,000 to the L value.

So again, we'll look at the value of L. And if it's small enough,

then we will add 8,000-- which if I add 8,000 to L and then I calculate H,

this will slow it down or supply less power.

And there's one last thing I have to do is acknowledge the interrupt.

And I acknowledge the interrupt by setting the interrupt clear register

and then return.

So this is a flow chart of the system.

The SysTick will produce the pulse-width modulated output,

Help

The interface for one motor is shown in Figure 12.10. If the robot has two motors then a second copy of the TIP120 circuit will be needed. The Port F edge-triggered interrupt will change the duty cycle by ±10% depending on whether the operator touches SW1 or SW2. Since the two ISRs pass data (**`H`**,**`L`** passed from **`GPIOPortF_Handler`** to **`SysTick_Handler`**), we will set the interrupt priorities to be equal. With equal priorities neither ISR will interrupt the other. This way, the global variables **`H`**,**`L`** will always be in a consistent state.

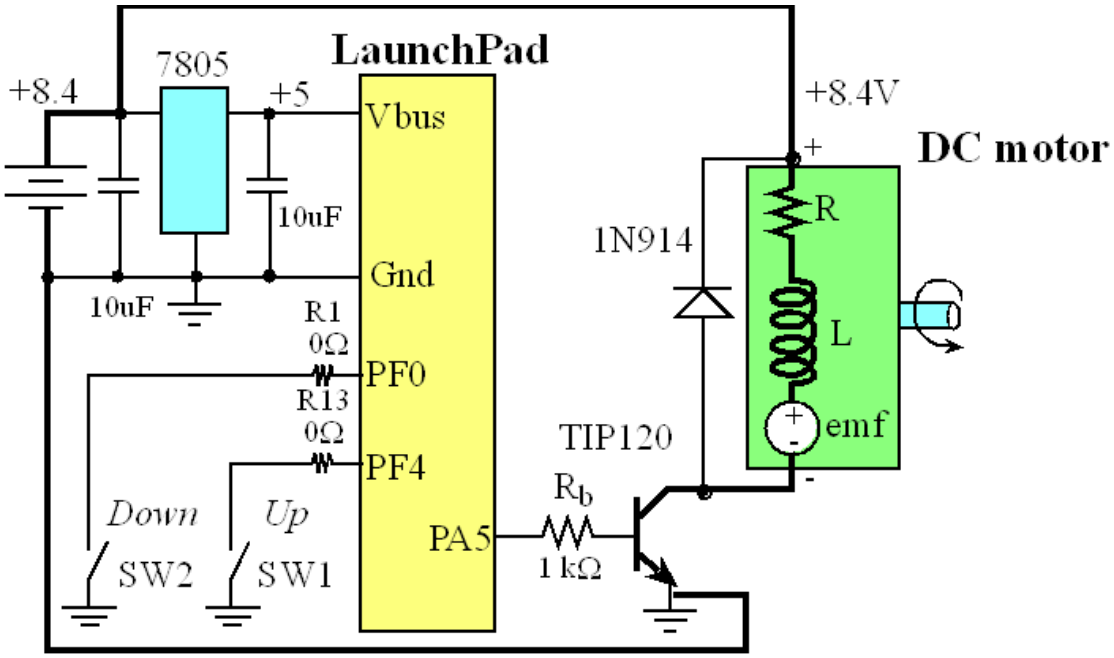

Figure 12.10. DC motor interface (bold lines show path of the large current).

```
unsigned long H,L;

void Motor_Init(void){
  SYSCTL_RCGC2_R |= 0x00000001; // activate clock for port A
  L = H = 40000;               // 50%
  GPIO_PORTA_AMSEL_R &= ~0x20;      // disable analog functionality on PA5
  GPIO_PORTA_PCTL_R &= ~0x00F00000; // configure PA5 as GPIO
  GPIO_PORTA_DIR_R |= 0x20;     // make PA5 out
  GPIO_PORTA_DR8R_R |= 0x20;    // enable 8 mA drive on PA5
  GPIO_PORTA_AFSEL_R &= ~0x20;  // disable alt funct on PA5
  GPIO_PORTA_DEN_R |= 0x20;     // enable digital I/O on PA5
  GPIO_PORTA_DATA_R &= ~0x20;   // make PA5 low
  NVIC_ST_CTRL_R = 0;           // disable SysTick during setup
  NVIC_ST_RELOAD_R = L-1;       // reload value for 500us
  NVIC_ST_CURRENT_R = 0;        // any write to current clears it
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000; // priority 2
  NVIC_ST_CTRL_R = 0x00000007;  // enable with core clock and interrupts
}

void SysTick_Handler(void){
  if(GPIO_PORTA_DATA_R&0x20){   // toggle PA5
    GPIO_PORTA_DATA_R &= ~0x20; // make PA5 low
    NVIC_ST_RELOAD_R = L-1;     // reload value for low phase
  } else{
    GPIO_PORTA_DATA_R |= 0x20;  // make PA5 high
    NVIC_ST_RELOAD_R = H-1;     // reload value for high phase
  }
}

void Switch_Init(void){  unsigned long volatile delay;
  SYSCTL_RCGC2_R |= 0x00000020; // (a) activate clock for port F
  delay = SYSCTL_RCGC2_R;
  GPIO_PORTF_LOCK_R = 0x4C4F434B; // unlock GPIO Port F
  GPIO_PORTF_CR_R = 0x11;         // allow changes to PF4,0
  GPIO_PORTF_DIR_R &= ~0x11;    // (c) make PF4,0 in (built-in button)
  GPIO_PORTF_AFSEL_R &= ~0x11;  //     disable alt funct on PF4,0
  GPIO_PORTF_DEN_R |= 0x11;     //     enable digital I/O on PF4,0
  GPIO_PORTF_PCTL_R &= ~0x000F000F; //  configure PF4,0 as GPIO
  GPIO_PORTF_AMSEL_R &= ~0x11;  //     disable analog functionality on PF4,0
  GPIO_PORTF_PUR_R |= 0x11;     //     enable weak pull-up on PF4,0
  GPIO_PORTF_IS_R &= ~0x11;     // (d) PF4,PF0 is edge-sensitive
  GPIO_PORTF_IBE_R &= ~0x11;    //     PF4,PF0 is not both edges
  GPIO_PORTF_IEV_R &= ~0x11;    //     PF4,PF0 falling edge event
  GPIO_PORTF_ICR_R = 0x11;      // (e) clear flags 4,0
  GPIO_PORTF_IM_R |= 0x11;      // (f) arm interrupt on PF4,PF0
  NVIC_PRI7_R = (NVIC_PRI7_R&0xFF00FFFF)|0x00400000; // (g) priority 2
  NVIC_EN0_R = 0x40000000;      // (h) enable interrupt 30 in NVIC
```

```
}
// L range: 8000,16000,24000,32000,40000,48000,56000,64000,72000
// power:   10%   20%   30%   40%   50%   60%   70%   80%   90%

void GPIOPortF_Handler(void){ // called on touch of either SW1 or SW2
  if(GPIO_PORTF_RIS_R&0x01){  // SW2 touch
    GPIO_PORTF_ICR_R = 0x01;  // acknowledge flag0
    if(L>8000) L = L-8000;    // slow down
  }
  if(GPIO_PORTF_RIS_R&0x10){  // SW1 touch
    GPIO_PORTF_ICR_R = 0x10;  // acknowledge flag4
    if(L<72000) L = L+8000;   // speed up
  }
  H = 80000-L; // constant period of 1ms, variable duty cycle
}

int main(void){
  DisableInterrupts();  // disable interrupts while initializing
  PLL_Init();           // bus clock at 80 MHz
  Motor_Init();         // output from PA5, SysTick interrupts
  Switch_Init();        // arm PF4, PF0 for falling edge interrupts
  EnableInterrupts();   // enable after all initialization are done
  while(1){ // main program is free to perform other tasks
    WaitForInterrupt(); // low power mode
  }
}
```

Program 12.7. Motor output using a periodic interrupt. Switches are used to adjust the power (C12_DCMotor).

VIDEO 12.4C. PWM SOFTWARE CONTROL OF DC MOTOR

C12 4c PWM software YouTube

PROFESSOR JONATHAN VALVANO: All right, let's look

at the software used to control the motor.

Let's begin with the initializations.

We're going to use port A for the output.

So I'll activate its clock.

	9:00 / 9:00	1.0x			

We have two global variables, H and L. And they

are going to be initialized to some value.

This value doesn't matter.

But they're initialized.

We're going to turn on port A bit 5 in the usual way-- no analog, direction

registers, and output.

I am going to set the 8 milliamp output current

because this has to drive the base of the TIP120.

So I'll enable 8 milliamps out of PA5.

And I'll enable it, and then initially make PA5 low.

I want to run SysTick.

So I'm going to set the reload value equal to L,

and so it will run for this initial value

Help