```c
// Paul Valenzuela
// Anh Nguyen
// John Salcedo
// PWMtest.c
// Runs on TM4C123
// Use PWM0/PB6 and PWM1/PB7 to generate pulse-width modulated outputs
// Use PA4 and PA3 to Control the Direction of a Car
// PF3 PF2 PF1 are LEDs used to indicate what direction car is going


#include "PLL.h"
#include "PWM.h"
#include "tm4c123gh6pm.h"
#include <stdint.h>       // C99 data types


// Basic functions defined at end of startup.s
void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
void WaitForInterrupt(void);  // Low power mode

// Global Variables that are manipulated throughout program
uint32_t currentState;        // State of FSM
uint32_t duty;                // Duty cyle of PB6 PB7

// Pre Defined States
#define stop 0
#define forward 1
#define reverse 2

// State Machine Structure
struct State {
  uint32_t pb6;               // PWM behavior for PB6
  uint32_t pb7;               // PWM behavior for PB7
  uint32_t direction;         // Direction (High or Low) PA4 and PA3
  uint32_t light;             // Light Indicating Direction
};

typedef const struct State STyp;

//State Machine
STyp FSM[3] = {
  {0x00, 0x080C, 0x00, 0x02},
  {0x8C, 0x080C, 0x00, 0x08},
  {0xC8 , 0x0C08, 0x18, 0x04},
};

unsigned long H,L;

// Initialize Directional Pins
void Motor_Init(void){

  SYSCTL_RCGC2_R |= 0x00000001;     // activate clock for port A

  GPIO_PORTA_AMSEL_R &= ~0x18;      // disable analog functionality on PA3 PA4
  GPIO_PORTA_PCTL_R &= ~0x000FF000; // configure PA3 PA4 as GPIO
  GPIO_PORTA_DIR_R |= 0x18;         // make PA3 PA4 out
  GPIO_PORTA_AFSEL_R &= ~0x18;      // Disable alt funct on PA3 PA4
  GPIO_PORTA_DEN_R |= 0x18;         // Enable digital I/O on PA3 PA4
  GPIO_PORTA_DATA_R = 0x00;         // Make PA4 low PA3 low


}

// Initialize Switches
void Switch_Init(void){  unsigned long volatile delay;
  SYSCTL_RCGC2_R |= 0x00000020;     // Activate clock for port F
  delay = SYSCTL_RCGC2_R;

  H = L = 8000;
  GPIO_PORTF_LOCK_R = 0x4C4F434B;   // Unlock GPIO Port F
  GPIO_PORTF_CR_R = 0x11;           // Allow changes to PF4,0
  GPIO_PORTF_DIR_R &= ~0x11;        // Make PF4,0 in (built-in button)
  GPIO_PORTF_DIR_R |= 0x0E;         // PF4 PF0 Out, PF3 PF2 PF1 In
  GPIO_PORTF_AFSEL_R &= ~0x11;      // Disable alt funct on PF4,0
  GPIO_PORTF_DEN_R |= 0x1F;         // 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;         // 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;          // Clear flags 4,0
  GPIO_PORTF_IM_R |= 0x11;          // Arm interrupt on PF4,PF0
  NVIC_PRI7_R = (NVIC_PRI7_R&0xFF00FFFF)|0x00200000; // Priority 1
  NVIC_EN0_R = 0x40000000;          // Enable interrupt 30 in NVIC

}

// Interrupt Handler for Port F Interrupts
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(currentState == 0) currentState = 1;       // From Stop to Forward
    else if(currentState == 1) currentState = 2;  // From Forward to Reverse
    else if(currentState == 2) currentState = 1;  // From Reverse to Forward
  }

  // The light is configured to show which state the car is in
  GPIO_PORTF_DATA_R = FSM[currentState].light;
  GPIO_PORTA_DATA_R = FSM[currentState].direction;



  if(GPIO_PORTF_RIS_R&0x10){    // SW1 touch
    GPIO_PORTF_ICR_R = 0x10;    // Acknowledge flag4

    if(duty<300 ) L = 300;          // 0% to 30% or 30% to 60%
    else if(duty < 600) L = 600;
    else if(duty < 800) L = 800;    // 60% to 80%
    else if(duty < 980) L = 980;    // 80% to 98%
    else
    { duty = 1;
      GPIO_PORTF_DATA_R = 0x02;     // Red light to indicate off
    }
  }

  // Configure New Duty Cycles and Behavior of PWM
  PWM0A_Duty(duty, FSM[currentState].pb6);  // PB6 PWM
  PWM0B_Duty(duty, FSM[currentState].pb7);  // PB7 PWM
}


int main(void){
  DisableInterrupts();          // Disable interrupts while initializing
  PLL_Init();                   // Bus clock at 80 MHz
  Motor_Init();                 // Output from PA4, PA3
  PWM0A_Init(1000, 100);        // Initialize PWM0, 40000 Hz, 10% duty

  Switch_Init();                // Arm PF4, PF0 for falling edge interrupts
                                // Also arm switches
  EnableInterrupts();           // Enable after all initialization are done
  currentState = 0;             // Currently Stopped
  duty = 300;                   // 30% Duty Cycle
  GPIO_PORTF_DATA_R = FSM[currentState].light;  // Light of current state
  while(1){
    WaitForInterrupt();         // Wait for Intterrupts
}

}
```

```c
// Paul Valenzuela
// Anh Nguyen
// John Salcedo
// PWM.c
// Runs on TM4C123
// Use PWM0/PB6 and PWM1/PB7 to generate pulse-width modulated outputs.

#include <stdint.h>
#include "tm4c123gh6pm.h"


#define PWM_0_GENA_ACTCMPAD_ONE 0x000000C0  // Set the output signal to 1
#define PWM_0_GENA_ACTLOAD_ZERO 0x00000008  // Set the output signal to 0
#define PWM_0_GENB_ACTCMPBD_ONE 0x00000C00  // Set the output signal to 1
#define PWM_0_GENB_ACTLOAD_ZERO 0x00000008  // Set the output signal to 0

#define SYSCTL_RCC_USEPWMDIV    0x00100000  // Enable PWM Clock Divisor
#define SYSCTL_RCC_PWMDIV_M     0x000E0000  // PWM Unit Clock Divisor
#define SYSCTL_RCC_PWMDIV_2     0x00000000  // /2


// period is 16-bit number of PWM clock cycles in one period (3<=period)
// period for PB6 and PB7 must be the same
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
// PWM clock rate = processor clock rate/SYSCTL_RCC_PWMDIV
//                  = BusClock/2
//                  = 80 MHz/2 = 40 MHz (in this example)
// Output on PB6/M0PWM0
void PWM0A_Init(uint16_t period, uint16_t duty){

  SYSCTL_RCGCPWM_R |= 0x01;             // Activate PWM0
  SYSCTL_RCGCGPIO_R |= 0x02;            // Activate port B
  while((SYSCTL_PRGPIO_R&0x02) == 0){}; // Wait for clock to initialize
  GPIO_PORTB_AFSEL_R |= 0x40;           // Enable alt funct on PB6

  GPIO_PORTB_PCTL_R &= ~0x0F000000;     // Configure PB6 as PWM0
  GPIO_PORTB_PCTL_R |= 0x0C000000;
  GPIO_PORTB_AMSEL_R &= ~0x40;          // Disable analog functionality on PB6
  GPIO_PORTB_DEN_R |= 0x40;             // Enable digital I/O on PB6
  SYSCTL_RCC_R = 0x00100000 |           // Use PWM divider
      (SYSCTL_RCC_R & (~0x000E0000));   // Configure for /2 divider
  PWM0_0_CTL_R = 0;                     // Re-loading down-counting mode
  PWM0_0_GENA_R = 0x8C;                 // High on LOAD, Low on CMPA down

  PWM0_0_LOAD_R = period - 1;           // Cycles needed to count down to 0
  PWM0_0_CMPA_R = duty - 1;             // Count value when output rises

  PWM0_0_CTL_R |= 0x00000001;           // Start PWM0
  PWM0_ENABLE_R |= 0x00000001;          // Enable PB6/M0PWM0
}


// change duty cycle of PB6
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
void PWM0A_Duty(uint16_t duty, uint16_t output){
  PWM0_0_CMPA_R = duty - 1;             // Set New Duty Cycle
  PWM0_0_GENA_R = output;               // Set New Output
}



// period is 16-bit number of PWM clock cycles in one period (3<=period)
// period for PB6 and PB7 must be the same
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
// PWM clock rate = processor clock rate/SYSCTL_RCC_PWMDIV
//                  = BusClock/2
//                  = 80 MHz/2 = 40 MHz (in this example)
// Output on PB7/M0PWM1
void PWM0B_Init(uint16_t period, uint16_t duty){
  volatile unsigned long delay;
  SYSCTL_RCGCPWM_R |= 0x01;             // Activate PWM0
  SYSCTL_RCGCGPIO_R |= 0x02;            // Activate port B
  delay = SYSCTL_RCGCGPIO_R;            // Allow time to finish activating
  GPIO_PORTB_AFSEL_R |= 0x80;           // Enable alt funct on PB7
  GPIO_PORTB_PCTL_R &= ~0xF0000000;     // Configure PB7 as M0PWM1
  GPIO_PORTB_PCTL_R |= 0x40000000;
  GPIO_PORTB_AMSEL_R &= ~0x80;          // Disable analog functionality on PB7
  GPIO_PORTB_DEN_R |= 0x80;             // Enable digital I/O on PB7
  SYSCTL_RCC_R |= SYSCTL_RCC_USEPWMDIV; // Use PWM divider
  SYSCTL_RCC_R &= ~SYSCTL_RCC_PWMDIV_M; // Clear PWM divider field
  SYSCTL_RCC_R += SYSCTL_RCC_PWMDIV_2;  // Configure for /2 divider
  PWM0_0_CTL_R = 0;                     // Re-loading down-counting mode
  PWM0_0_GENB_R = 0x80C;                // PB7 goes high on LOAD, low on CMPB down

  PWM0_0_LOAD_R = period - 1;           // Cycles needed to count down to 0
  PWM0_0_CMPB_R = duty - 1;             // Count value when output rises
  PWM0_0_CTL_R |= 0x00000001;           // Start PWM0
  PWM0_ENABLE_R |= 0x00000002;          // Enable PB7/M0PWM1
}


// change duty cycle of PB7
// duty is number of PWM clock cycles output is high  (2<=duty<=period-1)
void PWM0B_Duty(uint16_t duty, uint32_t output){
  PWM0_0_CMPB_R = PWM0_0_CMPA_R;        // Set New Duty Cycle
  PWM0_0_GENB_R = output;               // Set New Output
}
```