PROJECT MOTOR INTERFACING USING THE FTM MODULE

Course: ENGG*3640 Microcomputer Interfacing

Instructor: Radu Muresan

Student Names/Numbers:

Bilal Ayyache (0988616)

Emeka Madubuike (0948959)

Mohammed Al-Fakhri (0982745)

Date: 28/11/2018

Contents

1. Introduction

The main objective of lab 6 is to get familiar with the FTM Module and learn how to use this module to control a DC Motor. To be able to use this module, one should understand what the flex timer module is. The flex timer module is a two-to-eight channel timer that supports input capture, output compare, and the generation of PWM signals. PWM signals can be used to control electric motors/ power management applications. The flex timer module is built upon the HCS08 timer PWM module. The FTM time reference is a 16-bit counter that can be used as an unsigned or signed counter.

FTM Features include:

- FTM source clock is selectable
- 16-Bit Counter
- Each channel can be configured for input capture, output compare, or edge-aligned PWM mode
- Each pair of channels can be combined to generate a PWM signal with independent control of both edges of PWM signal
- Software control of PWM outputs

To communicate between the microcontroller and DC motor, the Inter-integrated Circuit was used (I2C). the I2C module provides a method of communication between various devices. The interface is designated to operate up to 100 kbits/s with maximum bus loading and timing. The I2C device is capable of operating at higher baud rates, up to a maximum of clock/20, with reduced bus loading. The maximum communication length and the number of devices that can be connected are limited by a maximum bus capacitance of 400 pF. The I2C module also complies with the System Management Bus (SMBus) Specification.

1.1. Lab Description

The main task of lab 6 was to control a DC Motor and to input a desired duty cycle for the DC motor through the putty terminal using the modules explained in the introduction section, the FTM and I2C modules. The FTM registers was used to update the frequency of the motor. These registers include configuration (4003_8084), fault mode status (4003_8074), features mode selection (4003_8054), channel status and Control(4003_800C), Channel Value (4003_8010), Counter Initial Value (4003_804C), Status and Control(4003_8000). Refer to the software implementation section to learn more about how these registers function.

1.2. System Requirements

During this lab, the tools and equipment that were used are enumerated below:

- Kiel Uvision Program was used to create instructions to the K60 Microcontroller
- Freescale TWR-K60D100M Microcontroller was used to implement instructions sent
- A Hi Speed USB 2.0 Connection cable between PC and board was used to connect the Microcontroller to the PC.
- Putty displayed the results by receiving commands through the COM port in which the USB 2.0 was connected to. Kiel Uvision software receives hardware input using the serial window. The Serial window accepts serial input and output data streams. The window displays serial output data received from a simulated CPU, while characters typed into a serial window are input to the simulated CPU. This allows testing a UART interface prior to having the target hardware.
- To implement the circuit design, a DC Motor, transistors, Diode, and a breadboard was used. (More information of equipment can be found in section 2.1).

2. Background

2.1. Equipment

- **Kiel Uvision Program**: The Kiel Uvision program is an IDE that combines project management, source code editing, program debugging, and run-time environment into a single powerful environment. Using this environment, the user is able to easily and efficiently test, verify, debug, and optimize the code developed. During the third lab, the debug functionality helped in understanding how the code is acting.

Figure 2.1.1 K60 Microcontroller

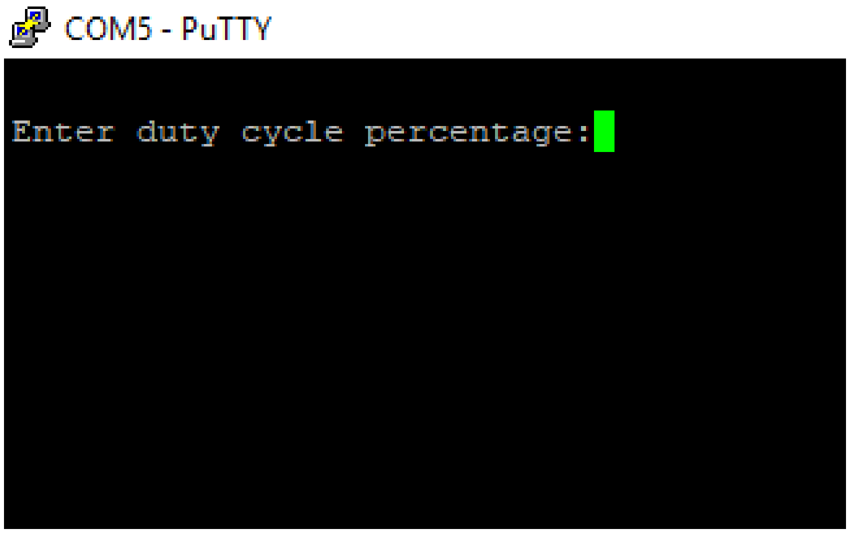
- **5V DC Motor:** A DC motor is a type of rotary electrical load that converts electrical energy to mechanical energy. The motor used in this lab uses an electromechanical mechanism that changes the direction of current flow periodically.

- **K60 Microcontroller**: The K60 Microcontroller (Figure2.1) from NXP contains a low power MCU core ARM Cortex-M4 that features an analog integration, serial communication, USB 2.0 full-speed OTG controller and 10/100 Mbps Ethernet MAC. These characteristics makes this Microcontroller suitable to preform task in a very efficient and fast manner. A USB connection was used to sync the microcontrollerwith the Keil uVision software that that was provided by the teaching assistant. Through Lab 3, the main feature that was used was the NVIC component.

- **BreadBoard:** A breadboard is a construction base for prototyping of electronics. In this lab we used a breadboard to implement the logic functions by connecting our IC Chips' pins to satisfy the objective of this lab. IC Chips were placed between the board's valley and pins were connected using jumper wires.
- **Resistors:** A resistor is a circuit element used to model the current resisting behavior of a material. For the purpose of constructing circuits, resistors are usually made from metallic alloys and carbon compounds. The resistor used was a 330Ohm resistor
- **Transistors:** a transistor is a "nonlinear" component that has three leads. Transistors can be used for switching and amplifying signals. The transistor used was a 2N222A Transistor
- **DC Power Supply:** A DC power supply is an electronic device that supplies electricity to a circuit. The power supply job is to convert electric current from the source to the voltage and current required.
- **Oscilloscope:** An Oscilloscope is an electronic device that is used to view oscillation of current or voltage, by displaying the signal on a digital screen.
- **Function Generator:** A function generator is an electronic device that produces different types of electric waveforms over large variety of frequencies. It can generate a sine wave, square wave, or a triangular wave.

3. Implementation

As an implementation overview of this lab, it was required a control a DC Motor and to input a desired duty cycle for the DC motor through the putty terminal using the FTM and I2C modules. The implementation of this lab was approached from two different engineering perspectives. Software implementation shows how the implementation was executed including the configuration of all the registers. Many obstacles were encountered in this part and to overcome these, an Engineering approach was considered. The code outline was discussed by team members and debugged before typing into uVision Keil. The second part was to discuss and determine what hardware components should be used to implement the circuit design and complete the main task. Hardware parts include transistors, resistors, jumper wires, oscilloscope, DC output generator, and DC Motor. This approach to lab 6 ensured success of experiment.



```
Enter duty cycle percentage:
```

Figure 3.1 Running the Code

User is prompted to enter the desired duty cycle to run the motor.

3.1.1. Software Implementation

Software implementation was executed using C Programming. Software was used to control a DC Motor by inputting a desired duty cycle for the DC motor through the putty terminal. This was performed using the FTM and I2C modules. The FTM registers was used to update the frequency of the motor. These registers include:

- **Configuration (Line 21):** absolute address location: 4003_8084, This register selects the number of times that the FTM counter overflow should occur before the TOF bit to be set, the FTM behavior in BDM modes, the use of an external global time base, and the global time base signal generation.
- **fault mode status (Line 22):** absolute address location: 4003_8074, This register selects the filter value for the fault inputs, enables the fault inputs and the fault inputs filter
- **features mode selection (Line 23):** absolute address location: 4003_8054, This register contains the global enable bit for FTM-specific features and the control bits used to configure
  - Fault control mode and interrupt
  - Capture Test mode
  - PWM synchronization

  - Write protection
  - Channel output initialization These controls relate to all channels within this module.
- **channel status and Control (Line 25):** absolute address location: 4003_800C, CnSC contains the channel-interrupt-status flag and control bits used to configure the interrupt enable, channel configuration, and pin function.
- **Channel Value (Line 26):** absolute address location: 4003_8010, this register contains the captured FTM counter value for the input modes or the match value for the output modes. In this lab the value was calculated and was multiplied by a percentage to control the frequency inputted. Refer to Figure A. 1 in list of figures section.
- **Counter Initial Value (Line 27):** absolute address location: 4003_804C, The Counter Initial Value register contains the initial value for the FTM counter. Writing to the CNTIN register latches the value into a buffer. The CNTIN register is updated with the value of its write buffer according to Registers updated from write buffers. When the FTM clock is initially selected, by writing a non-zero value to the CLKS bits, the FTM counter starts with the value 0x0000. To avoid this behavior, before the first write to select the FTM clock, write the new value to the the CNTIN register and then initialize the FTM counter by writing any value to the CNT register.
- **Status and Control (Line 28):** absolute address location: 4003_8000, SC contains the overflow status flag and control bits used to configure the interrupt enable, FTM configuration, clock source, and prescaler factor. These controls relate to all channels within this module.

3.1.2. Hardware Implementation

As shown in the circuit, the circuit was implemented using a MOSFET transistor, diode, and a motor. Voltage was supplied by the DC Power generator. The microcontroller was connected to the first pin in the transistor, the motor was connected to the second pin and lastly the power supply was connected to the motor. Third pin in the transistor was connected to ground.

When the user resets the microcomputer, the program runs. The motor controller simply works when voltage is applied, so if the power source applies high and the microcontroller applies high the motor will not run because the flow is the same on both side. When the user inputs a percentage on the PuTTY terminal, as the user inputs different percentages the speed of the motor will change. This value changes the amount of time for the flow of voltage through circuit, whenever the percentage increase the amount of time for the flow increase which causes the motor to run faster.

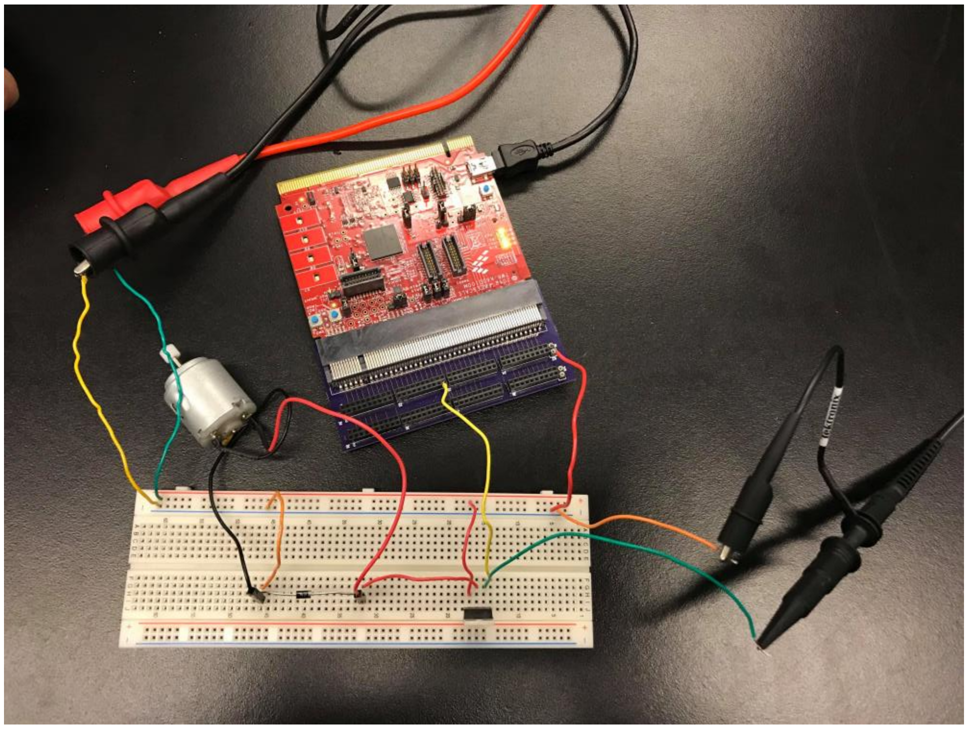
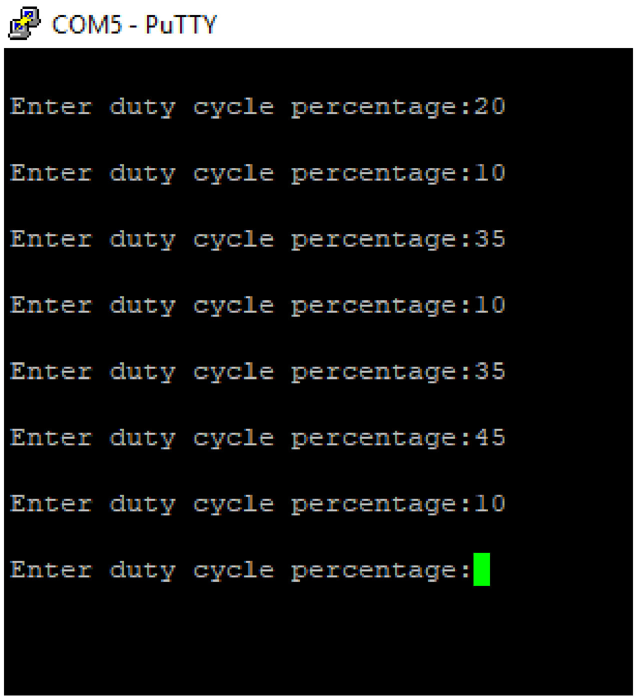


Figure 3.1.2.1 Circuit Connection

3.2. Simulation Results



COM5 - PuTTY

```
Enter duty cycle percentage:20

Enter duty cycle percentage:10

Enter duty cycle percentage:35

Enter duty cycle percentage:10

Enter duty cycle percentage:35

Enter duty cycle percentage:45

Enter duty cycle percentage:10

Enter duty cycle percentage:
```

Figure 3.2.1. Results of running code with inputs

3.3. Lab Requirements

3.3.1. Lab Requirement 1

For circuit design, refer to hardware implementation. To understand how the PMOS Transistor acts in this circuit refer to lab requirement 2 in section 3.3.2.

3.3.2. Lab Requirement 2

The FTM module implemented to control the 5V DC motor can be seen in appendix Figure A.

Through exhaustive testing, a trend was discovered. As the duty cycle increase, the lower edge of the signal decreased. This is as a result of our use of edge alignment, as we want our rising edges on the same spot always. Due to this, when the duty cycle is increased and the gate is on for longer our signals top and bottom edges get longer. However due to the placement of the rising edge as a result of edge alignment the lower edge is cut shorter and shorter the more we increase our duty cycle.

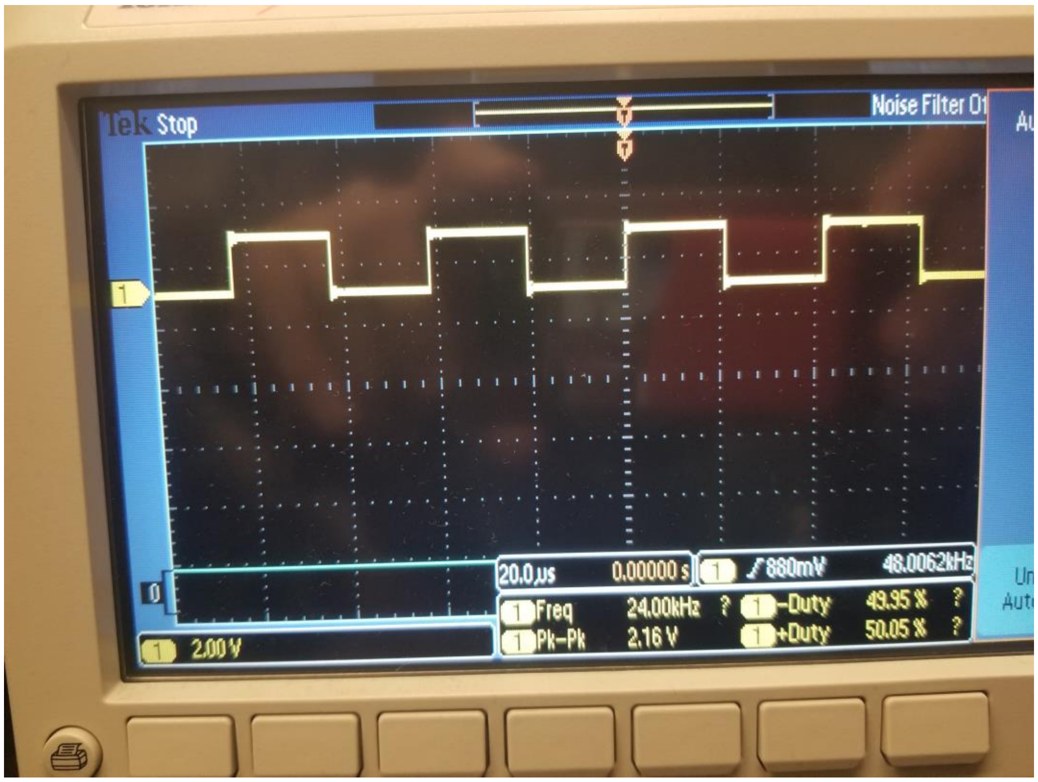


Figure 3.3.2.1. Oscilloscope screenshot at 50%

4. Conclusion

In Conclusion, the K60 Microcomputer was able to successfully control the DC Motor using the flex Timer module. This was completed using Pulse width Modulation. Results of this lab proves that the functionality of the experiment is evident and shows that the software and hardware implemented works properly. Results of lab 5 can be found in the simulation results section (section 3.3).

The user for the program was prompted to enter a duty cycle through the putty terminal which was then retrieved and used to update the frequency of the motor. The user was able to change the speed of the motor by inputting a percentage value in the putty terminal. User was allowed to continuously alter the duty cycles inputted to the program. A while loop was used to make this function possible as it allowed the user to control the speed of the motor.

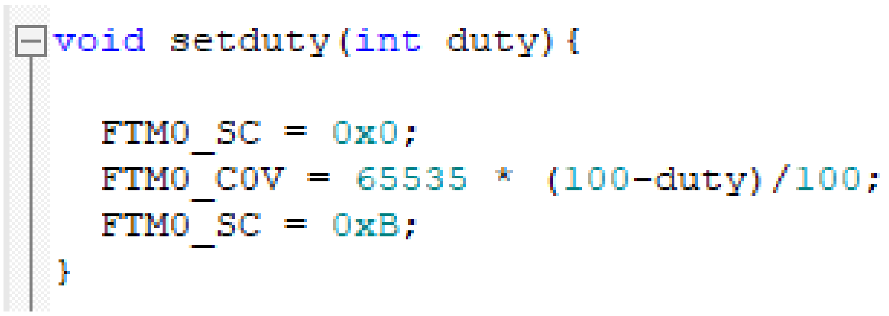
Through lab 6, the main challenge helped in understanding how FTM modules work and learn how to use this module to control a DC Motor. Such module is very useful in the engineering industry as DC Motors are used in a lot of different technologies. This includes automation and industrial engineering.

References

[1] Radu Muresan, "ENGG3640: Microcomputer Interfacing Laboratory Manual, Version 2", University of Guelph, July 2016.

List Of Figures

1. Main.c



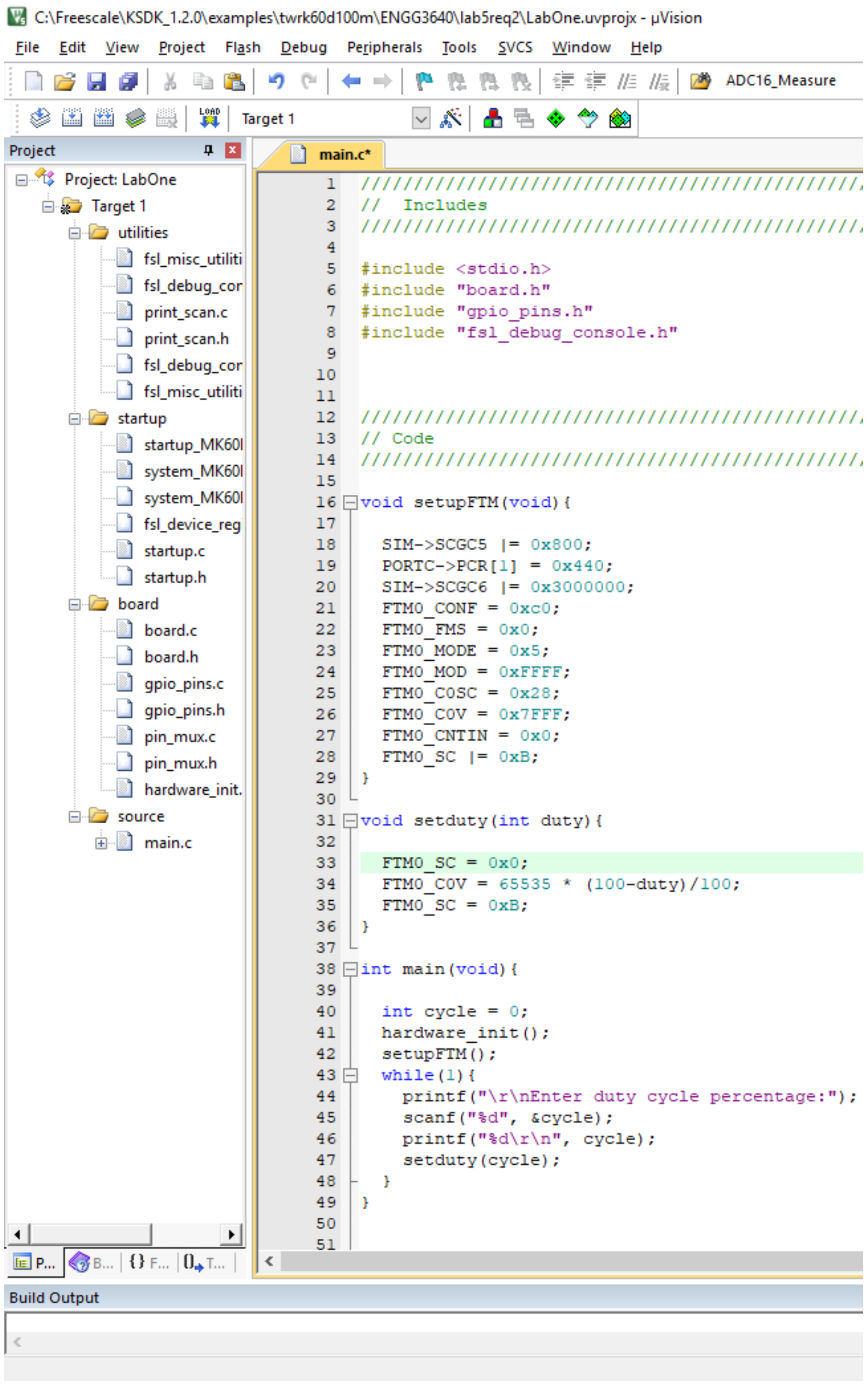
```
void setduty(int duty){

   FTM0_SC = 0x0;
   FTM0_C0V = 65535 * (100-duty)/100;
   FTM0_SC = 0xB;
}
```

Figure A. 2 Set Duty function

Appendices

A. Main.c (requirement 2)



```
///////////////////////////////////////////////
//  Includes
///////////////////////////////////////////////

#include <stdio.h>
#include "board.h"
#include "gpio_pins.h"
#include "fsl_debug_console.h"



///////////////////////////////////////////////
// Code
///////////////////////////////////////////////

void setupFTM(void){

  SIM->SCGC5 |= 0x800;
  PORTC->PCR[1] = 0x440;
  SIM->SCGC6 |= 0x3000000;
  FTM0_CONF = 0xc0;
  FTM0_FMS = 0x0;
  FTM0_MODE = 0x5;
  FTM0_MOD = 0xFFFF;
  FTM0_C0SC = 0x28;
  FTM0_C0V = 0x7FFF;
  FTM0_CNTIN = 0x0;
  FTM0_SC |= 0xB;
}

void setduty(int duty){

  FTM0_SC = 0x0;
  FTM0_C0V = 65535 * (100-duty)/100;
  FTM0_SC = 0xB;
}

int main(void){

  int cycle = 0;
  hardware_init();
  setupFTM();
  while(1){
    printf("\r\nEnter duty cycle percentage:");
    scanf("%d", &cycle);
    printf("%d\r\n", cycle);
    setduty(cycle);
  }
}
```

Figure A. 1 Main.c