Freescale Semiconductor
Application Note

Document Number: AN4251
Rev. 0, 01/2011

Controlling DC motors and Servo motors

Example code based on the MPC5604B MCU

by: Francisco Ramirez Fuentes, Marco Trujillo, Cuauhtli Padilla, Rodrigo Mendoza

Contents

1 Introduction

The first section of this application note provides the basics of DC (Direct Current) and Servo motors.The successive sections explain the implementation of code drivers using the MPC5604B MCU. The electronic circuits created to control these motors and schematics for PCBs, tips to reduce noise over important signals can also be found in this application note.

A DC motor is an electrical device that converts energy into rotational movement. The motor moves a gear in one direction if current flows through the terminals (clockwise or counterclockwise), and in the opposite direction if current flows backwards through the same terminals. If there is a force opposing the motor, then the terminals are short circuited and the current through the terminals can go as high as 5 A or more. The voltage or current that must be delivered to the motor to work is too much for a microcontroller output port so an intermediary device must be used. In this application note, the Freescale H Bridge – MC33931 is used.

Table 1. Advantages and Disadvantages of using an IC H-bridge (MC33931)

Advantages	Disadvantages
Senses current flowing through output and temperature.	With a voltage lower than 8 V the device is functional, which increases the output resistance, thus the power consumed too.
If the current is higher than 6.5 A then it uses PWM (pulse width modulation) to regulate it.	
Compatible with TTL/CMOS logic inputs.	
Sleep mode available (low current consuming).	

The DC motor used for the example implemented is a 24 V motor. As current can go very high, the H-Bridge will have to tri-state the outputs in order to prevent heat or electric damage to any component. The servo motor is an electric actuator that can be positioned in a desired angle from 0° to 180°. The operation of a typical servo motor is explained in Figure 1; depending on the duty cycle of the control signal, the servo motor will rotate to a specific position. In this application note, Section 3 onwards it is explained how to control the hardware and motors with the MCU. A detailed explanation of each function of the medium and low level drivers is given.

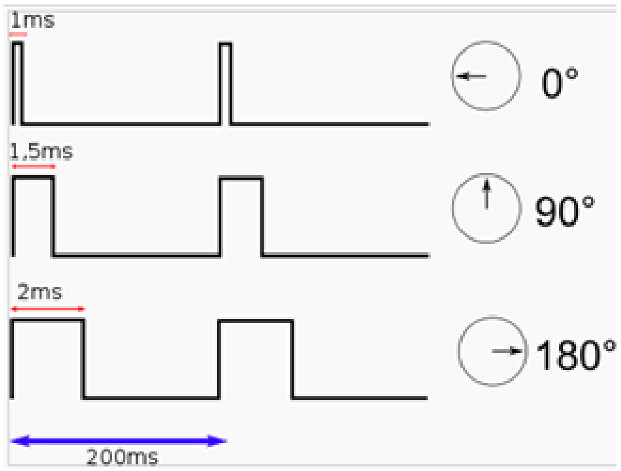

Figure 1. Controlling a Servo motor

NOTE

Rev 0 of this application note provides only the drivers as explained in the example code, which can be downloaded as AN4245SW from https://www.freescale.com. A complete example using these drivers will be provided in the next revision of the application note.

2 DC Motor Circuitry

The circuit to control the motor uses three inputs (Table 2): IN1, IN2 and EN where IN1 and IN2 are inputs that define the direction of the motor, and EN defines if the motor is in Sleep mode. The outputs are three (Table 2): OUT1, OUT2 and FB, where OUT1 and OUT2 are the control of the motor (the two terminals of the motor are connected to these outputs), and FB is the feedback current connected to a small resistor (smaller than 300Ω) then it can be read by an ADC channel on the microcontroller to know how much current is driven to the motor, so this way, the user can know if the motor is ramping up or stalled for some reason.

Table 2. Truth table for inputs and outputs

EN	IN1	IN2	OUT1	OUT2
HIGH	LOW	LOW	LOW	LOW
HIGH	HIGH	LOW	HIGH	LOW
HIGH	LOW	HIGH	LOW	HIGH
HIGH	HIGH	HIGH	HIGH	HIGH
LOW	HIGH OR LOW	HIGH OR LOW	TRISTATED	TRISTATED

Table 2 reflects that the outputs have the same value as the inputs if the enable is HIGH (VDD) and are tri-stated if the enable is LOW (0V).

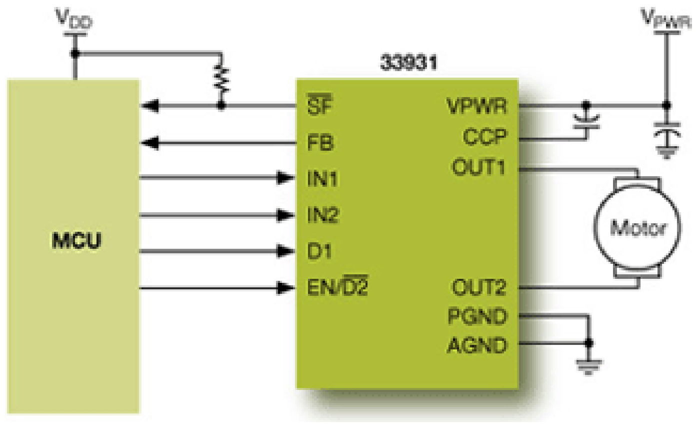

Figure 2. Schematic for a DC Motor control

In the schematic (Figure 1), some pins such as CCP (Charge Pump), D1 (Disable 1), and SF (status flag) can be seen (not shown at the truth table) that must be connected as recommended so that the H Bridge can operate properly. For more information, refer to the MC33931 datasheet from https://www.freescale.com.

3 Speed control on the DC motor

To control the speed of the DC motor, a proportional integral algorithm in C language has been created. It uses a variable that controls the speed and is increased or decreased if it is not running at a pre-selected speed.

To do this implementation, some hardware is required to measure the speed of the motor. It can be done with an infrared LED and a opto-electronic sensor, with a Hall Effect sensor and magnets, or many other ways. A square train of pulses is obtained as the output of this sensing implementation. This is called the “velocimeter”. Once the “velocimeter” is implemented, the software is developed to translate the output into measured speed. The driver to measure this speed input uses SAIC (Single Action Input Capture – eMIOS functionality; see MPC5604BCRM Reference Manual from https://www.freescale.com for a better reference) to calculate the period (Figure 2), and this uses a modulus counter (MCB) on the MPC5604B – it also can use some other eMIOS functionalities; such as IPWM (Input Pulse Width Measurement). Once the period is obtained, the measured speed is equal to the circumference of the wheel of the car divided by the time of a complete revolution. Then, a control function calculates the difference of the desired speed and the measured speed, increasing or decreasing the signal output proportionally to that speed, by the difference already multiplied by a constant:

where Kc is a constant (correction factor).

Figure 3. Velocity equation

$$\text{VelocityControl} = \text{VelocityControl} + -(\text{Difference} \times \text{Kc})$$

In the code, Kc is 0,0000083 because the velocities are calculated in mm/s and the VelocityControl variable is a PWM duty that goes from 0 to 100, so the variable has to be very small. This may vary depending on the algorithm and the units chosen.

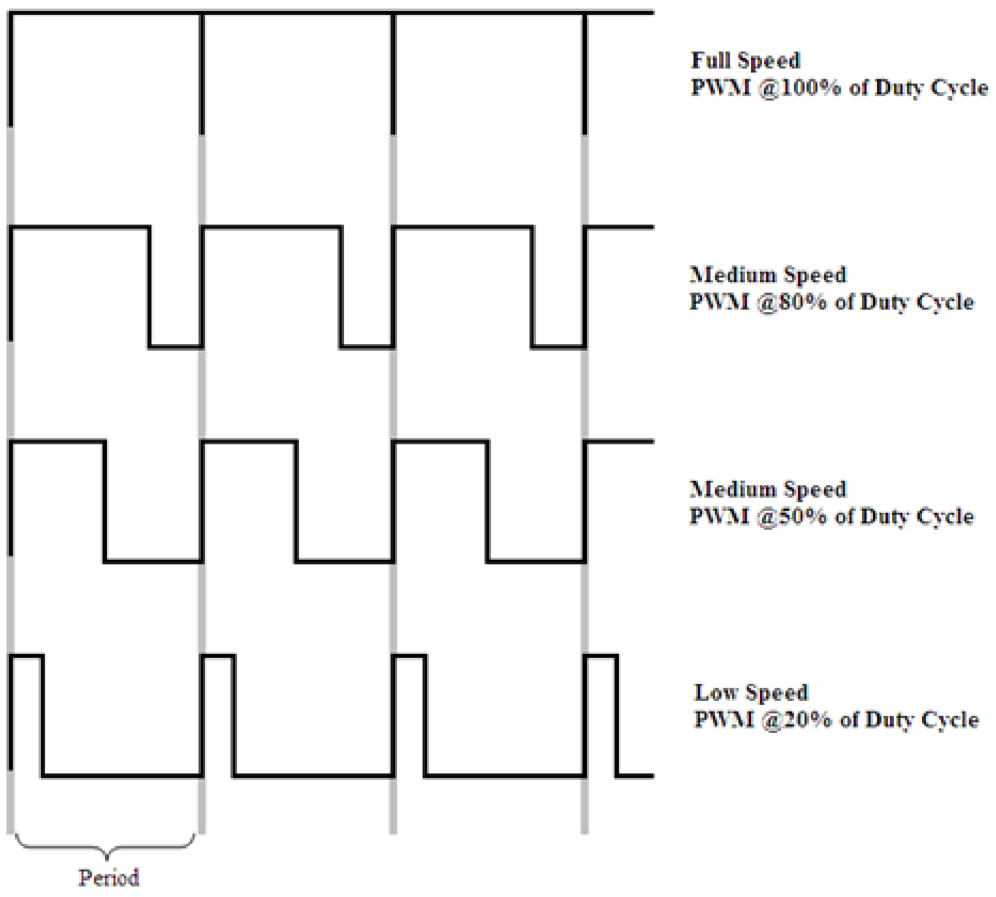

Figure 4. Measuring the speed

4 Controlling the Servo motor

The signal that controls the servo motor in this example is given by series of pulses (Figure 1) with a period of 200 ms or 5 Hz and the duration depends on the required direction. For most servo motors, pulses of 1 ms represent a 0 degree turn, and 2 ms represent a 180 degree turn, but it can vary. This information shall be specified in the data sheet of each servo motor and can be tested by trying different PWM signals. To send the signal by software, the MCU must generate a PWM signal with a duty cycle of 0.5% to 1% and a period of 200 ms. It is recommended to elaborate a function that has a position parameter and sets the servo motor to the desired position to simplify the code (as implemented in the example code for this document).

5 Suggestion to reduce noise in signals

When elaborating PCBs or designing circuits involving motors that consume large amounts of current, it is useful to separate completely signals with high and low current within the hardware. For PCBs, signals such as motor outputs, analog ground and power supply voltage should be with wide area and low resistance and it is recommended to use heat sinks to prevent some components from melting. Another useful tip is to capacitate as much as possible important analog signals, in order to reduce noise caused by the motor or long wires. It is essential to connect capacitors to all steady state pins or signals vulnerable to noise (and power supplies).

6 Sample code for Servo and Motor control

System Architecture

The drivers (Figure 5) are implemented for the MPC560xB to control a servo motor and a DC motor. In the following sections, a detailed explanation of each function on the medium and low level drivers is given:

- High level
    - Main motor control algorithm (high level program done by user)

- Medium level
    - Servo motor driver (Driver_Servo.c and Driver_Servo.h)
    - Main motor driver (Driver_Motor.c and Driver_Motor.h)

- Low level
    - ADC driver (Driver_ADC.c and Driver_ADC.h)
    - eMIOS driver (Driver_EMIOS.c and Driver_EMIOS.h)
    - SIU driver (Driver_SIU.c and Driver_SIU.h)
    - Initialization:
        - Driver_MPC5604B.h

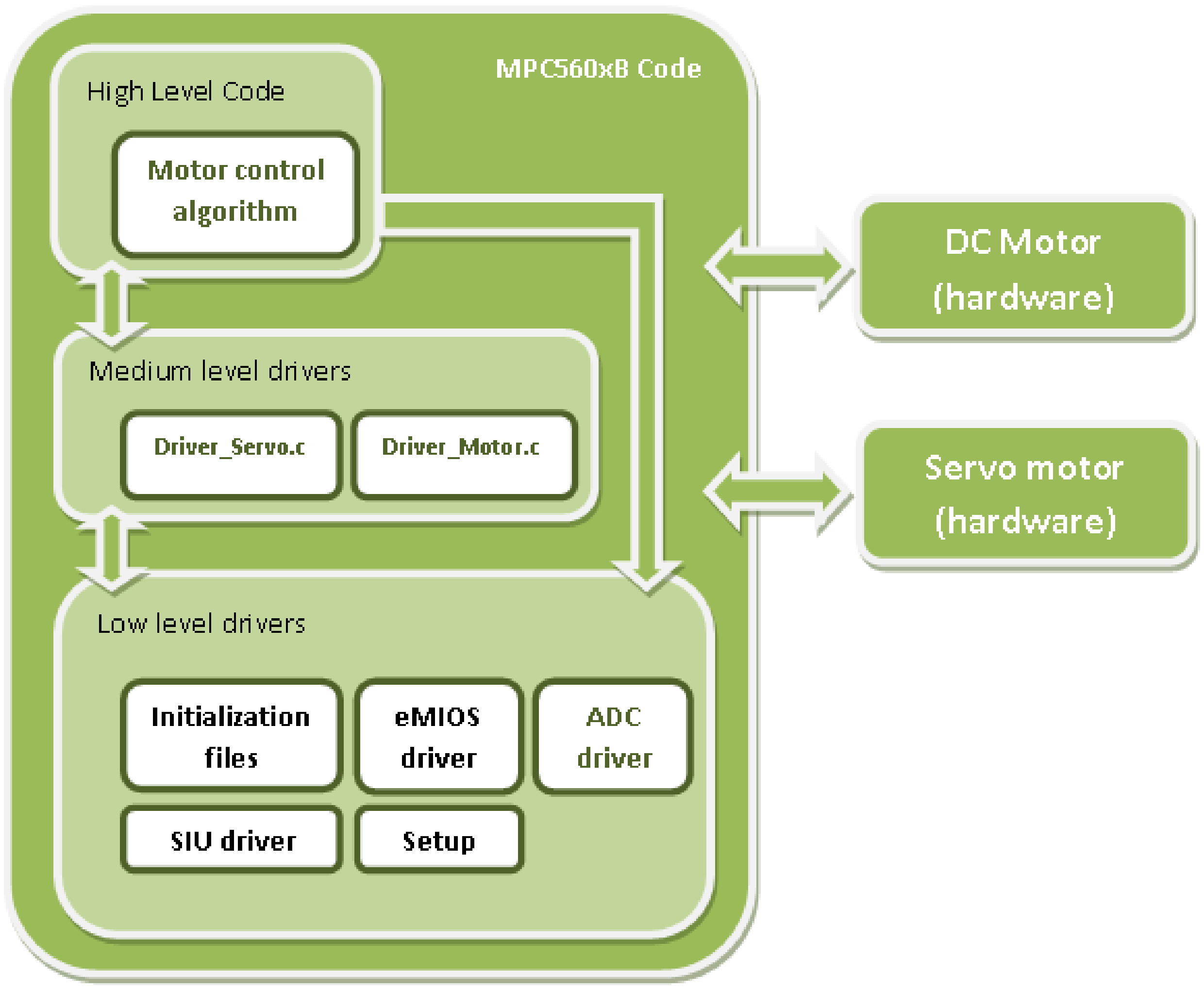

Figure 5. Software Architecture-APIs distribution

7 Driver explanation

Medium level drivers

All the drivers are explained in detail in this section.

Table 3. Driver_Servo.c

Function	*vfnSet_Servo*	Sets the servo motor to a position relative to the maximum and minimum values established. For example, if a reading in SAIC has a value of 10 as a minimum because less is invalid, and a maximum of 510, then a call to vfnSet_Servo(260,10,510) sets the servo motor exactly at the middle, and vfnSet_Servo(10,10,510) sets the servo motor to the left (all left possible).
Parameters	• u16Position: The desired position of the servo motor, with relative values to the minimum and maximum values. • u16MinVal: The minimum value possible for u16Position (if u16Position is equal to u16MinVal then the servo motor turns all the way left). • u16MaxVal: The maximum value possible for u16Position (if u16Position is equal to u16MaxVal then the servo motor turns all the way right).	
Return	Null	
Function	*vfnInit_Servo*	Initializes the MCB counters for both the servo motor and the main motor modules and the OPWM channel for the direction pin of the servo motor (and its respective SIU pin initialization).
Definitions involved for both functions	SERVO_CTRL	The eMIOS channel used for the signal for servo control.
	SERVO_CTRL_PCR	The pad number of the signal for servo control used for pad initialization in SIU.
	SERVO_MIN_US	The value representing a 0 degree turn in the wheels controlled by the servo motor. This value is in microseconds, and is the width of the 20ms-period signal pulse which determines direction.
	SERVO_MAX_US	The value representing a 180 degree turn in the wheels controlled by the servo motor. This value is in microseconds, and is the width of the 20ms-period signal pulse which determines direction.
	SERVO_MCB_CHANNEL	The eMIOS channel used as Modulus Up Counter Buffered which is the counter for the servo signal. This means that the counter must have a 20ms period.

Table 4. Functions on Driver_Motor.c

Function	*vfnSet_Motor_Forward* *vfnSet_Motor_Reverse* *vfnSet_Motor_Stop*	Sends pulses to the motor equivalent to the percentage parameter u16Perc. It is polarized normal (forward) or reverse depending on the function, and for motor stop, IN1 and IN2 control outputs are turned to 5V to stop the motor from the H Bridge.
Parameters	• u16Perc; with a 100 being 100% power (Vcc), 0% is GND, and 50% is equivalent to a clock signal with 50% duty	
Return	• u16Get_Feedback_Current; Returns the current consumed by the motor by reading an ADC port.	
Function	*vfnSet_Motor_Velocity*	Sets the motor forward to a desired velocity by increasing the PWM signal (more duty) to the motor if the measured velocity is too slow or by decreasing the PWM signal if the measured velocity is too fast. It works with a proportional integral algorithm.
Function	*vfnRead_Period*	If a SAIC scan is done, this function replaces the last value of *i32Period* with the period measured, else it does nothing. *i32Period* is in micro seconds and represents the time it takes to the back wheels to give one revolution because the signal the SAIC scans has a rising edge each time the wheels give a complete revolution. From this variable velocity can be measured: Velocity = Circumference of wheel / i32Period This function must be called frequently in case the SAIC scan is done, and if it is called long after the SAIC scan is done, it will lose precision. It also checks if an overflow occurs in the MCB so that the i32Period variable is increased by the time to overflow (period of MCB). Also, to measure the period accurately, it has to know if the MCB had an overflow, to sum the corresponding value to the i32Period counter, so each time it has an overflow, this function takes care of adding the variable.
Function	*vfnInit_Motor*	Initializes the GPIO pins for the direction of the motor, the OPWM channel for the enable pin of the motor (and its respective SIU pin initialization), the analog pin for the ADC scan of the current, and the SAIC channel for the velocimeter (and its respective SIU pin initialization).

Table continues on the next page...

Table 4. Functions on Driver_Motor.c (continued)

Definitions involved	MOTOR_MCB_CHANNEL	The eMIOS channel used as Modulus Up Counter Buffered which is the counter for the motor signal. This means that the period of the counter will be the period of the motor PWM signal.
	MOTOR_IN_1_PIN	The GPIO pin used for logic output IN1, which is a control signal for the H Bridge controlling the motor.
	MOTOR_IN_2_PIN	The GPIO pin used for logic output IN2, which is a control signal for the H Bridge controlling the motor.
	MOTOR_EN	The eMIOS channel used for the signal of Motor Enable for the H Bridge controlling the motor.
	MOTOR_EN_PCR	The pad number of the signal of Motor Enable used for pad initialization in SIU.
	MOTOR_SENSE_CH	The ADC channel used for current feedback of the motor.
	VELOCIMETER	The eMIOS channel used for the signal of the Velocimeter, input that measures the speed of the wheels.
	VELOCIMETER_PCR	The pad number of the signal of Velocimeter used for pad initialization in SIU.
	MAX_CURRENT	Number equivalent to the current consumed when feedback pin returns 5V.
	CIRCLE	The measure of the circumference of the wheels in nanometers.
	CONTROL	Control and delay are constants used in the proportional integral control unit of speed. The constant Kc is 1/(CONTROL*DELAY), where CONTROL is the proportional constant, and DELAY is the number of times the program waits to execute the code (executes 1 of DELAY times).
	DELAY	
	THRES and THRES2	These threshold constants define a threshold of maximum and minimum speed where the motor starts up fast or stops to get to the desired speed quickly.

Low level drivers

Table 5. Driver_ADC.c

Function	*vfnInit_NormalConversion_Adc*	Initialize ADC in scan mode, Configure ADC clock to 32 MHz, set an ADC Channel from a channel type as a Normal Conversion, and start conversions by setting NSTART to 1.
Parameters	• u8ChannelType: The type of the channel (Precision, extended or external) defined in Driver_MPC5604B.h as ADC_CHANNEL_TYPE_tag. • u32Channel: The ADC channel used for convertions defined in Driver_MPC5604B.h as ADC_CHANNEL_tag.	
Return	Null	
Function	*u16Read_Adc*	Checks for the last ADC conversion to be complete, reads the value of the conversion, scale the read value in a range from 0 to Maximum Value and returns the scaled value.
Parameters	• u8Channel: The channel used for ADC. Channels appear in Driver_MPC5604B.h.	
Return	• u8MaximumValue: The maximum possible value for the result. It is the value returned if a 10 bit conversion returns 1023.	

Table 6. Driver_eMIOS.c

vfnSetup_Emios_0 and vfnInit_Emios_0	Enables eMIOS clock, configure prescaler to generate 4 MHz eMIOS clock, enables global time base, enables Freezing channel to freeze them when in debug mode. Also enable eMIOS counters to start pulse generation and processing.
vfnInit_Emios_0_Mcb	Defines eMIOS channel as Modulus up counter buffered with the selected period, configure prescaler to produce 1 MHz time base. The parameters of the function are the following: *u8Channel*: eMIOS channel to be configured as Counter (0,8,16,23,24). *u16Period*: Sets A register to establish period in clock pulses.
vfnInit_Emios_0_Opwm	Defines eMIOS channel as positive OPWM with time base corresponding to the counter bus B, C, D or E and establish its raising and falling edge. The parameters are the following: • u8Channel: eMIOS channel to be configured as OPWM. • u16A: Sets A register to establish leading edge. • u16B: Sets B register to establish trailing edge.

Table continues on the next page...

Table 6. Driver_eMIOS.c (continued)

vfnInit_Emios_0_Saic	Defines eMIOS channel as SAIC with time base corresponding to the counter bus B, C, D or E. Allow channel freezing and set required polarity.
vfnSet_Duty_Opwm	Establish Duty Cycle in counter pulses for an eMIOS channel. The parameters are the following: • u8Channel: eMIOS channel configured as OPWM. • u16Duty: Sets B register to establish trailing edge (register A is set to zero).
vfnSet_Duty_Perc_Opwm	Establish Duty Cycle in a percentage for an eMIOS channel. The parameters are the following: • u8Channel: eMIOS channel configured as OPWM. • u16DutyPerc: Duty cycle in a percentage (0-100). • u16McbChannel: eMIOS channel used as counter bus time base.
u16Read_Saic	Returns the value of counter when SAIC flag occur for a channel of EMIOS module, if the reading is not ready of between minimum and maximum parameters, then it returns 0. The parameters are the following: • u8Channel: eMIOS channel configured as SAIC. • u16MinVal: Minimum accepted read value. • u16MaxVal: Maximum accepted read value.
u16Get_Counter	Returns the counter value of the MCB of the channel selected when the function is executed, it has a channel parameter which determines the MCB channel.
u16Get_Period_Mcb	Returns the fixed period value of the MCB of the channel selected. It has a channel parameter which determines the MCB channel.

SIU is the module that assigns functions to the physical pins. These drivers were implemented because there are many functions each pin can have, so there are pad configuration register values for each pin (refer to MPC5604BCRM Reference Manual available from https://www.freescale.com).

Table 7. Driver_SIU.c

vfnInit_Emios_Output_Pad	Initializes and assigns a pin as eMIOS, and for output purpose.
vfnInit_Emios_Input_Pad	Initializes and assigns a pin as eMIOS, and for input purpose.
vfnInit_Adc_Pad	Initializes and assigns a pin as ADC for input purpose.
The following functions are not part of the SIU but GPIO, yet both are used to control pad configuration.	
vfnInit_Gpio_Out	Initializes and assigns a pin as GPIO for output purpose.
vfnSet_Gpio	Sets the value of the pin initialized as GPIO to the parameter value u8Val which can be 0 or 1 (0V or 5V).
Parameter	*u8PcrVal* which is the SIU pad configuration register value for a pin selected

Table 8. Setup.c

vfnDisable_Watchdog	Disables the watchdog by clearing the watchdog enable.
vfnInit_Peri_Clk_Gen	Enables peripheral set 3 and divides by 1 the system clock.
vfnInit_Modes_And_Clock	Initializes the general modes RUN0, and for ADC, SIU, EMIOS, and clock, with PLL to 64 MHz.
vfnInit_All	Calls all the functions in this file. This is the only function of Setup.c required to be called in the main file.

Driver_MPC5604B.h

This file has no functions, and contains only useful definitions that relate modules and make easier the programming. Includes an ADC channel selection masks, pad configuration register values for ADC channels, pad configuration register values for eMIOS channels and pad configuration register values for GPIO.

An example to know when to use them is the following:

```
#define PCR_EMIOS_0_1345 /*PC13*/
```

This means that Channel 13 of eMIOS 0 can be assigned to port C13, and the value that the SIU uses to assign that pin is 45 (which is the pad configuration register for that pin). For a complete reference on how these registers are defined, please refer to the MPC5604BCRM available from https://www.freescale.com.

Document Number: AN4251
Rev. 0, 01/2011