Freescale Semiconductor
Application Note

Document Number: AN3737
Rev. 0, 12/2008

Emulating a PWM Module Using the IO Processor (IOP) on the MPC5510 Family

by: Oscar Luna
RTAC Latin America

1 Introduction

This application note provides software framework to emulate a pulse width modulated (PWM) driver by taking advantage of the MPC5510 second core (Z0).

There are many techniques available to generate pulse width modulated signals. This document describes software emulation of PWM signals using the IOP.

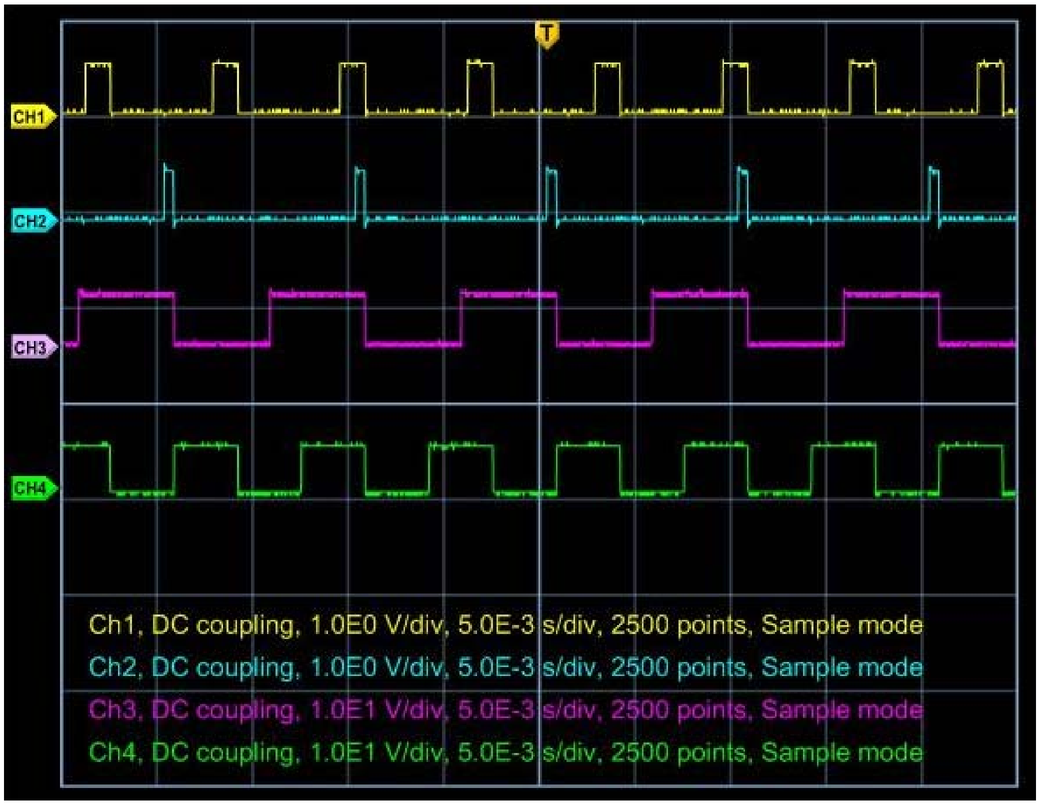
Contents

2 Emulated PWM Module Principles

An emulated PWM module is implemented by setting up one or several GPIO pins as output and by configuring one periodic interrupt timer (PIT) channel as a time base reference for generating all PWM signals.

The PIT channel is initialized with a predefined interrupt time base to generate each of the PWM channel signals configured by the user. The PIT provides the number of time ticks generated for each PWM channel. The PIT also determines the resolution of all the generated PWM signals.

The IOP core services the PIT timer interrupt. When a service routine occurs the IOP core decreases a counter variable providing the exact position within the period. Depending on this position and the desired duty cycle the generated PWM algorithm decides whether to set, clear, or leave unchanged the signal on an output pin. Several PWM channels with different duty cycles can be executed by using one PIT as a time base reference.



```
Ch1, DC coupling, 1.0E0 V/div, 5.0E-3 s/div, 2500 points, Sample mode
Ch2, DC coupling, 1.0E0 V/div, 5.0E-3 s/div, 2500 points, Sample mode
Ch3, DC coupling, 1.0E1 V/div, 5.0E-3 s/div, 2500 points, Sample mode
Ch4, DC coupling, 1.0E1 V/div, 5.0E-3 s/div, 2500 points, Sample mode
```

Figure 1. Generated PWM Signal Principle

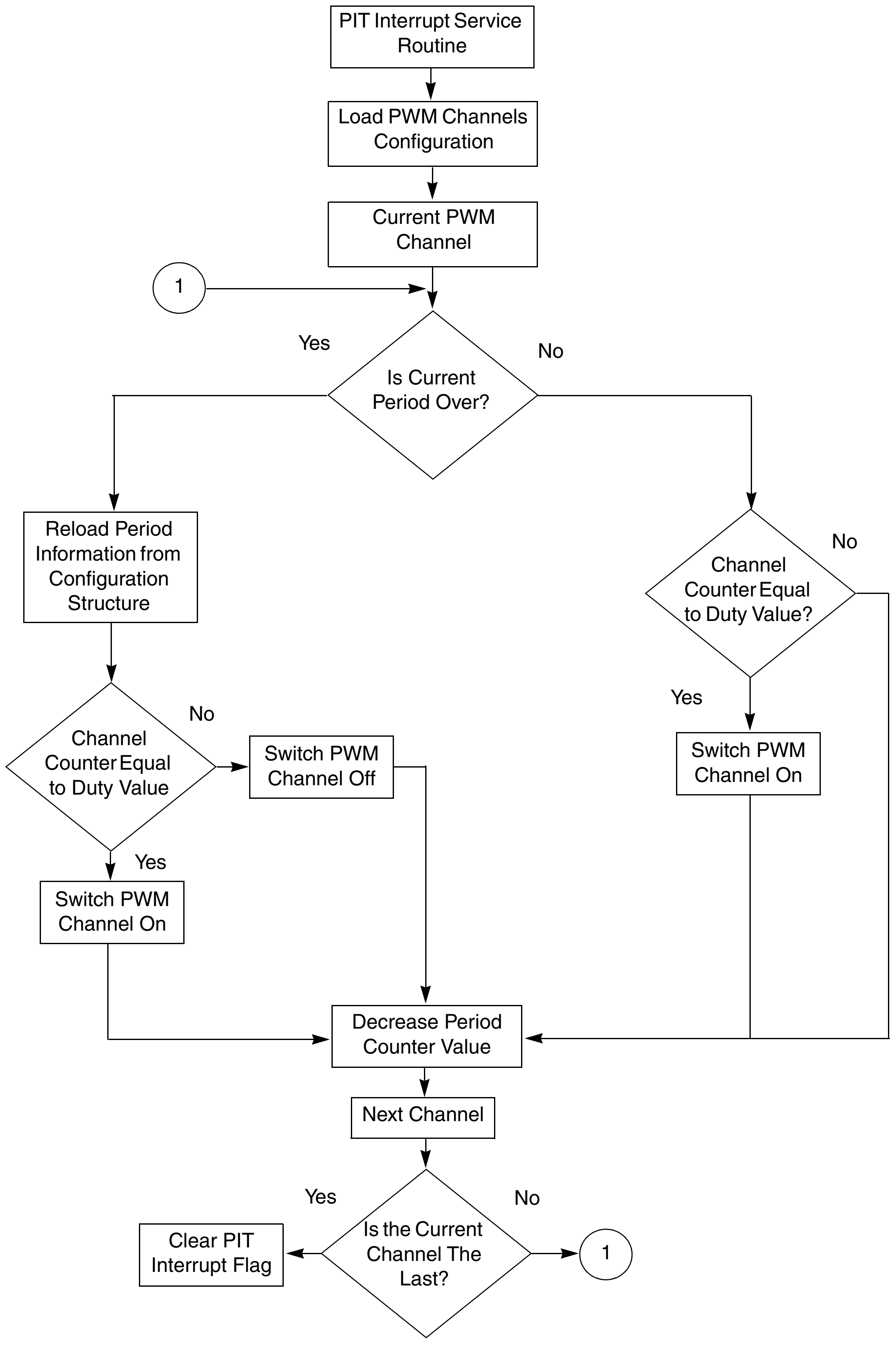

PIT Interrupt Service Routine
Load PWM Channels Configuration
Current PWM Channel
1
Is Current Period Over?
Yes
No
Reload Period Information from Configuration Structure
Channel Counter Equal to Duty Value?
No
Yes
Switch PWM Channel On
Channel Counter Equal to Duty Value
No
Switch PWM Channel Off
Yes
Switch PWM Channel On
Decrease Period Counter Value
Next Channel
Yes
Is the Current Channel The Last?
No
Clear PIT Interrupt Flag
1

Figure 2. PWM Generating Algorithm

3 PWM Driver Configuration

A configuration structure is used to operate the PWM driver during initialization and run-time operation. The configuration holds all the information related with each configured PWM channel. All configuration parameters are located in Pwm_Cfg.c. Detailed information to configure the emulated PWM driver is further explained in this section.

3.1 Channel Configuration

Each PWM channel needs five different configuration parameters to operate correctly. Figure 3 demonstrates each of the five parameters.

```
/** Pwm Channel Configuration Parameters */
typedef struct
{
    uint8_t  u8Pwm_Channel_Id;        /* Pwm channel name              */
    uint8_t  u8Pwm_Port_Channel;      /* Assigned HW channel           */
    uint32_t u32Period;               /* Pwm Period Value              */
    uint32_t u32Duty;                 /* Pwm duty Cycle                */
    uint32_t u32Cntr;                 /* Duty Cycle Counter            */
} Pwm_ChannelConfigType;
```

Figure 3. PWM Channel Configuration Parameters

3.2 Channel Parameters Explanation

- u32Pwm_Channel_Id — Symbolic identifier used as a reference for vfnSetPeriodAndDuty and vfnSetDuty_Cycle functions.
- u32Pwm_Port_Channel — This parameter indicates the microcontroller pin number to configure as a PWM channel. This field contains only the offset value of the microcontroller pin configurated as a PWM signal. For example, pin sixteen describes the offset value of PortB pin 0, pin seventeen describes the offset value of PortB pin 1, and so on.
- u32Period — Holds the number of time ticks to generate a desired period.
- u32Duty — This parameter contains the number of time ticks that reach a specific duty cycle within the limits of the period value.
- u32Cntr — Run-time counter used to match the end of a duty cycle signal and final edge of the period.

3.3 Configuring PWM Channels

These are the general configuration parameters needed to configure each PWM channel. Configuration parameters are located in Pwm_Cfg.c.

```
Pwm_ChannelConfigType Pwm_ChannelConfig[] =
{
   {
            (uint8_t)(PWM_CHANNEL_0),      /* Pwm Channel Id                    */
            (uint8_t)(PORTB_PIN_0),        /* Port B, Pin 0                     */
            (uint32_t)SET_PERIOD_AT_150HZ, /* Period running at 150Hz           */
            (uint32_t)DTY_AT_5P_WITH_150HZ,/* 5% Duty Cycle                     */
            (uint32_t)0x00                 /* Duty Counter Initialized          */
    },
         {
        (uint8_t)(PWM_CHANNEL_1),           /* Pwm Channel Id                  */
        (uint8_t)(PORTB_PIN_1),             /* Port B, Pin 1                   */
            (uint32_t)SET_PERIOD_AT_100HZ,  /* Period running at 100Hz         */
            (uint32_t)DTY_AT_20P_WITH_100HZ,/* 20% Duty Cycle                  */
            (uint32_t)0x00                  /* Duty Counter Initialized        */
    }
};
```

3.4 PWM General Configuration Definitions

There are general parameters mandatory to correctly operate the PWM driver. These parameters are:

- PWM_MAX_CHANNELS — The total number of configured PWM channels. Maximum number of allowed PWM channels are thirty-two and only if enough port pin resources are available.
- PWM_ERROR_DETECT — Enables or disables the error detection layer.
- Z0_CORE — When configured to on state, the IOP processor manages the PIT interrupt service routine to generate all PWM signals.
- PWM_INIT_DELAY — Enables or disables the switching delays between PWM signals to reduce EMI. Enabling this feature allows the driver to start each of the configured PWM signals with a predefined delay between signals to reduce EMI.
- DIAGNOSTICS — When configured to on state, this allows the ADC to be triggered at any point within a period.

NOTE

User must set the total amount of configured PWM channels in the PWM_MAX_CHANNELS to avoid unexpected behaviors in the driver operation.

4 PWM Driver Initialization

Three different sections complement the PWM driver initialization. These sections are further detailed in this application note. Figure 2 shows the complete initialization sequence the driver performs.

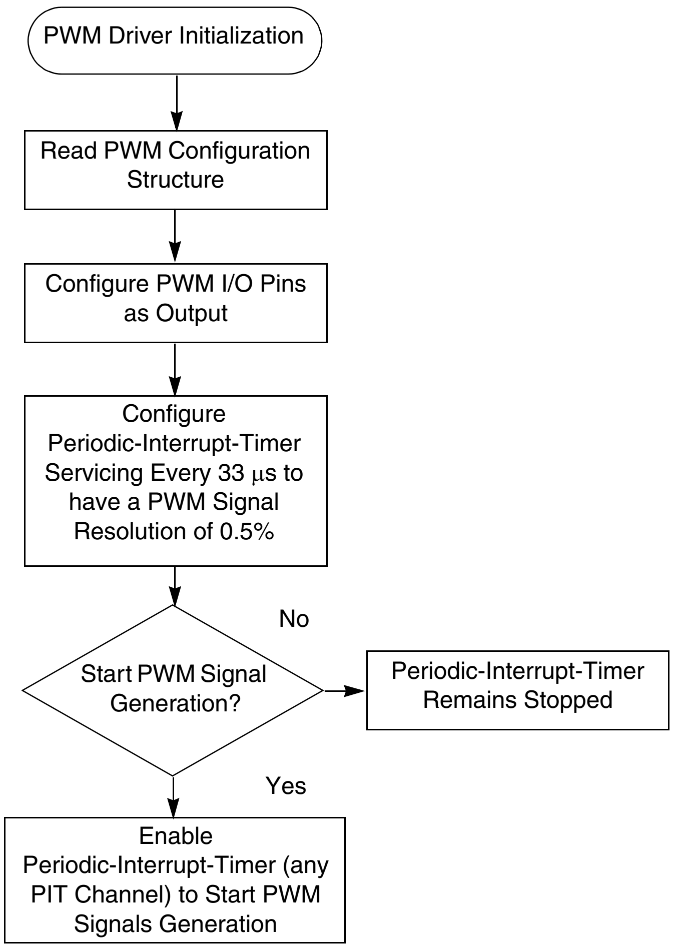

PWM Driver Initialization
Read PWM Configuration Structure
Configure PWM I/O Pins as Output
Configure Periodic-Interrupt-Timer Servicing Every 33 µs to have a PWM Signal Resolution of 0.5%
Start PWM Signal Generation?
No
Periodic-Interrupt-Timer Remains Stopped
Yes
Enable Periodic-Interrupt-Timer (any PIT Channel) to Start PWM Signals Generation

Figure 4. PWM Driver Initialization Procedure

4.1 Configuring I/O Pins as PWM Channels

During the PWM initialization sequence the driver first configures all I/O pins described in the configuration as outputs. No other pins are configured as PWM channels other than the pins stated in the channel configuration.

The I/O pin configuration algorithm goes through each of the PWM channels in the channel configuration to initialize every pin as output. The initialization algorithm takes the pin number value from the parameter, configures the pin as output, and sets the output state as high.

The algorithm configures only the I/O pins described in the configuration:

```
for(u32PortCntr = PWM_MAX_CHANNELS; u32PortCntr > (uint32_t)0 ; u32PortCntr--)
{
/* Obtain current Port Pin Offset to calculate physical address */
u8PortPinOffset = (&PortCfgPtr[(uint32_t)u32PortCntr - (uint32_t)1])->u8Pwm_Port_Channel;
/* Configure current Pin Pad as Output */
SIU_PCR_ADDR(u8PortPinOffset);
/* Set current pin state as High or Low */
SIU_GPDO_ADDR(u8PortPinOffset,PWM_HIGH);
}
```

NOTE

Make sure that pins configured as PWM channels are not used by another task in the application.

4.2 Periodic-Interrupt-Timer (PIT) Initialization

The PIT provides the time base for generating the PWM signals. The PWM driver makes use of PIT channel one as a time base reference and is configured to generate a periodic interrupt of 33 µs. This periodic interrupt is chosen to provide the driver to generate frequencies of 100 and 150 Hz with a duty cycle resolution of 0.5%.

The PWM frequency can easily be changed by setting a different counter value invoking the function shown below:

```
vfnPit_Init(uint8_t u8PitChannel, uint32_t u32Period)
```

To change the frequency operation of the PWM driver it is required to specify the PIT channel to use as a base timer (u8PitChannel) and also the PIT counter value used to generate the periodic interrupt intervals (u32Period) to generate PWM signals.

The example code provided within this application note generates PWM frequencies of 100 and 150 Hz for a high quality lighting application.

To generate a 33 µs time base using PIT channel one, the initialization code start enabling the PIT module. After enabling the module, channel one is selected and preloaded with a counter value of 2112 based on a system frequency of 64 MHz. This counter value generates an interrupt service of 33 µs.

$$PitInterruptCounter = \frac{Period}{ClockPeriod} - 1 \qquad \textbf{Eqn. 1}$$

Period=33 µs,Clock_Period(64MHz)=15.63 ns

$$PitInterruptCounter = \frac{33\mu s}{15.62ns} - 1 = 2112s \qquad \textbf{Eqn. 2}$$

After selecting and loading channel one counter, the next step the code executes is to rise the interrupt priority level of PIT channel one. It also assigns the core that will take control of channel one.

The PWM driver can take control of any PIT channel and assign the PIT interrupt service routine to Z1 (main processor) or Z0 (second processor) core. By default the Z0 core takes control of PIT channel one.

The PWM driver then clears the PIT interrupt flag from channel one to avoid any false triggering before enabling the channel. After clearing the channel flag the channel is enabled to operate and generate an interrupt every 33 µs.

Figure 4 shows the complete initialization of PIT channel one that generates a time base of 33 µs.

```
void vfnPit_Init(uint8_t u8PitChannel, uint32_t u32Period)
{

PIT_CTRL = (uint32_t)0x00;
PIT_LOAD_VALUE(u8PitChannel, u32Period);
#if Z0_CORE == ON
INT_PSR((uint8_t)((uint8_t)PIT_BASE_OFFSET + (uint8_t)u8PitChannel)
,(uint8_t)0xC1);
#else
INT_PSR((uint8_t)((uint8_t)PIT_BASE_OFFSET + (uint8_t)u8PitChannel)
,(uint8_t)1);
#endif
PIT_FLG         = (PIT_FLG    | ((uint32_t)1<< (uint32_t)u8PitChannel));
PIT_INTEN     = (PIT_INTEN  | ((uint32_t)1<< (uint32_t)u8PitChannel));
PIT_INTSEL   = (PIT_INTSEL | ((uint32_t)1<< (uint32_t)u8PitChannel));
vfnPit_Set_Callback_Fnc(Pwm_Generation_Fnc);
}
```

Figure 5. PIT Channel One Initialization Procedure

NOTE

The user must decide which core takes control over the PIT channel interrupt and define the interrupt vector in the correct core vector table.

5 Error Detection Layer

An error detection layer has been implemented in the PWM driver to catch any wrong parameters that pass through each of the implemented functions in the driver.

The error detection layer can be enabled or disabled as desired. This option reduces the size of the driver if no error detection is needed during run-time. The error detection layer is enabled by setting the PWM_ERROR_DETECT value to on state.

```
#define PWM_ERROR_DETECT  ON
```

The error detection layer avoids writing wrong configuration values to PWM functions and invalid register values to the periodic-interrupt-timer. When the error detection layer validates the parameters through the PWM functions it configures the registers and continues with the function flow.

If the error detection layer does not validate a correct parameter, the function reports where the error is detected and the type of error. These report parameters are stored in a RAM location and it is up to the user to take action if an error occurs.

```
if(Pwm_Cfg_Ptr == NULL_PTR)  /* Pwm module already initilized?         */
{
   vfnPwm_Report_Error(PWM_SETDUTYCYCLE_ID,PWM_E_UNINIT);
}
```

6 PWM Implementation Functions

6.1 Changing the Duty Cycle During Run-Time

Changing the duty cycle is allowed during run-time by invoking the function vfnSetDuty_Cycle. A new duty cycle value takes place after the period in progress reaches the end.

The duty cycle function is configured by passing the PWM channel id and the new duty cycle value. This function validates whether the new duty cycle value is within the allowed maximum range.

```
void vfnSetDuty_Cycle(uint8_t u8PwmChannel, uint32_t u32DutyCyle)
```

To configure the maximum range of duty cycle permitted, it is mandatory to set the macro MAX_DUTY_CYCLE with the desired maximum value.

```
#define MAX_DUTY_CYCLE     (uint32_t) (DUTY_CYCLE_100)
```

The maximum duty cycle is not restricted to any value. The user can set the maximum value needed.

Figure 6 shows how the change duty cycle function works.

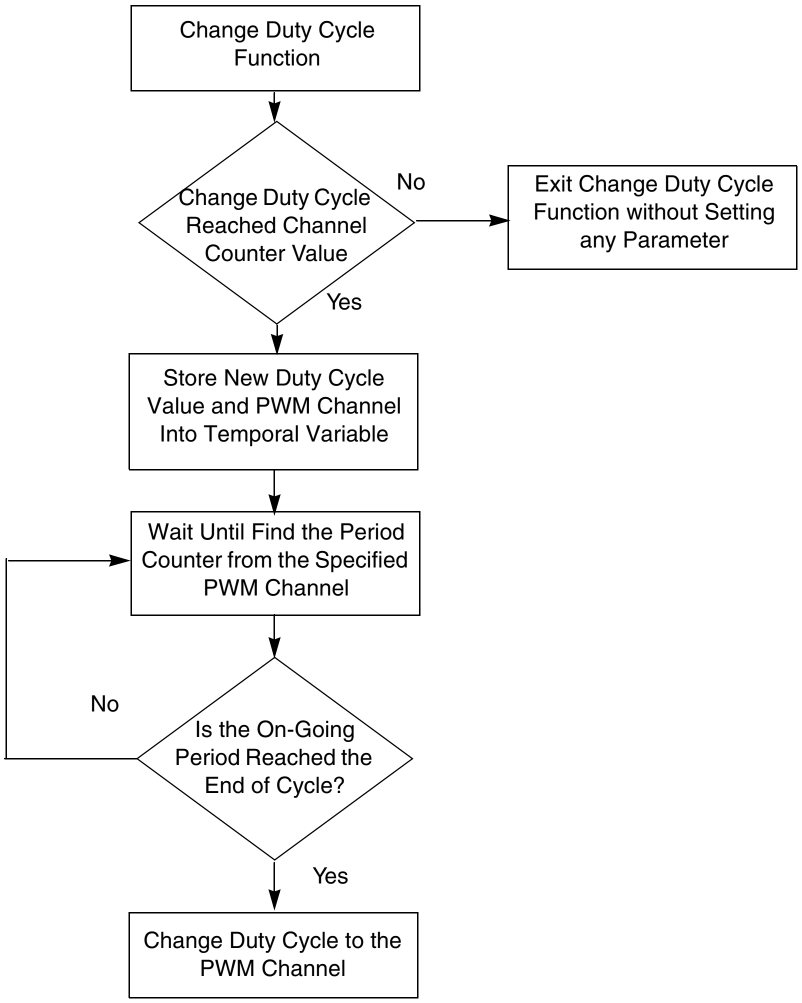

Change Duty Cycle Function
Change Duty Cycle Reached Channel Counter Value
No
Exit Change Duty Cycle Function without Setting any Parameter
Yes
Store New Duty Cycle Value and PWM Channel Into Temporal Variable
Wait Until Find the Period Counter from the Specified PWM Channel
No
Is the On-Going Period Reached the End of Cycle?
Yes
Change Duty Cycle to the PWM Channel

Figure 6. Procedure to Change Duty Cycle During Run-Time

6.2 Changing Period and Duty Cycle During Run-Time

Changing period and duty cycle in run-time is possible by calling vfnSetPeriodAndDuty. This function needs three different parameters to change the period and duty cycle of a specific PWM channel.

These three parameters are the PWM channel id, the new period value, and the new duty cycle value. The vfnSetPeriodAndDuty function validates the values of the period and duty cycle to avoid any malfunctions in the PWM signal generation.

6.3 Initializing the Z0 Core and Transferring Control Over PIT Channel One

After the microcontroller exits the reset state, the Z1 core is activated. The IOP core remains in reset after the device comes out from power reset. To release the IOP core from the reset state, the Z1 core can then release the IOP core from reset.

A macro definition is used to release the IOP core from reset. The IOP core is brought out of reset by writing the Z0VEC register in the clock, reset, and power control (CRP) module with the start address.

```
#define RUN_PWM_ON_Z0     CRP.Z0VEC.R = (unsigned long)__start_p1
```

7 PWM Driver Features

The PWM driver includes a channel offset feature that reduces EMI and a diagnostic feature that allows an ADC channel to be triggered at various points during the PWM period. These features can be enabled at a pre-compiled time and are found at Pwm_Cfg.h file.

All PWM features are described below:

```
#define PWM_INIT_DELAY   OFF
#define DIAGNOSTICS      OFF
```

Both features can be enabled/disabled by setting the predefined values of on/off to their respective feature.

7.1 Enabling Switching Offsets to Reduce EMI

Switching on all PWM channels and subsequent loads at the same time causes an increase in EMI. To reduce this effect the PWM driver can be configured to offset the switching of each channel by a programmable delay.

To enable the selectable switching delay feature it is mandatory to configure the next two parameters:

- Set PWM_INIT_DELAY definition located at Pwm_Cfg.h file with the precompiled definition on.
  ```
  #define PWM_INIT_DELAY  ON
  ```
- Configure the initialization counter value (u32Cntr) of each PWM channel to allow switching time delay on that particular channel. The PIT timeout determines the smallest possible EMC delay allowed.

The channel switching time delay counters are located at Pwm_Cfg.c file. The PWM software provides flexibility during the EMC evaluation of the electronic module. The switching delay can be programmed by enabling the precompiled definition PWM_INIT_DELAY and configuring each PWM channel with different initialization values on their counter variables:

1. `#define PWM_INIT_DELAY  ON   (Pwm_Cfg.h file)`
2. `On Pwm_Cfg.c file:`

```
#if (PWM_INIT_DELAY == ON)
         Pwm_ChannelConfigType Pwm_ChannelConfig[PWM_MAX_CHANNELS] =
         {
            {
                  (uint32_t)(PWM_CHANNEL_0),
                  (uint32_t)(PORTB_PIN_0),
                  (uint32_t)SET_PERIOD_AT_150HZ,
                  (uint32_t)DTY_AT_50P_WITH_150HZ,
                  (uint32_t)0x08, /* Switching delay counter          */
                   #if (DIAGNOSTICS == ON)
                  (uint32_t)ADC_CHANNEL_28,
                  (uint32_t)0x10,
                  (uint32_t)0x00
                  #endif
             },
            {
                  (uint32_t)(PWM_CHANNEL_1),
                  (uint32_t)(PORTB_PIN_1),
                  (uint32_t)SET_PERIOD_AT_150HZ,
                  (uint32_t)DTY_AT_50P_WITH_150HZ,
                  (uint32_t)0x10, /* Switching delay counter          */
                  #if (DIAGNOSTICS == ON)
                  (uint32_t)ADC_CHANNEL_29,
                  (uint32_t)0x10,
                  (uint32_t)0x00
                  #endif
                  }
         }
#endif
```

The fifth parameter of each PWM channel structure is the initialization counter value that produces the switching delay offset on that specific channel.

7.2 Measuring Loads on PWM Channels Using the Diagnostic Feature

A diagnostic implementation function allows the driver to measure loads during the PWM signals generation. To obtain an accurate measure of the load, this runtime feature has been implemented to trigger as near as possible to the end of the active phase of the PWM period.

The following code shows an example of configuring ADC channel 28 to trigger every time the duty cycle reaches 50%.

```
#if (DIAGNOSTICS == ON)
 (uint32_t)ADC_CHANNEL_28,  /* ADC Channel 28 assigned              */
 (uint32_t)0x60,            /* Triggers ADC when 50% duty is reached  */
 (uint32_t)0x00             /* Reset ADC buffer                     */
#endif
```

An ADC trigger value is compared against the current counter value of a specific channel when the diagnostic feature is enabled while the PWM signals are generated. After both values match, an ADC conversion is initiated based on the ADC channel number taken from the PWM channel configuration (Pwm_Cfg.c file).

To enable the diagnostic feature it is necessary to perform the following steps:

1. Enable the precompiled diagnostics definition by changing the state from off to on.

   ```
   #define DIAGNOSTICS  ON   (Pwm_Cfg.h file)
   ```
2. Configure the ADC channel to be sampled and the ADC triggering value used to start a conversion at a certain position within the generation of the PWM signal.

The example code shown below uses ADC channel 28 to measure the load on PWM channel 0. The seventh parameter (from top to bottom) holds the ADC trigger value indicating the value within the period where the ADC conversion starts.

```
Pwm_ChannelConfigType Pwm_ChannelConfig[PWM_MAX_CHANNELS] =
{
   {
(uint32_t)(PWM_CHANNEL_0),
       (uint32_t)(PORTB_PIN_0),
       (uint32_t)SET_PERIOD_AT_150HZ,
       (uint32_t)DTY_AT_50P_WITH_150HZ,
       (uint32_t)0x08,
        #if (DIAGNOSTICS == ON)
       (uint32_t)ADC_CHANNEL_28,
       (uint32_t)0x10,
       (uint32_t)0x00
     #endif
   }
}
```

The diagnostic feature implemented during runtime only starts a conversion on a particular channel based on the PWM configuration parameter. It is mandatory the user read the converted value using the function described below at any given time within the user's code:

ADC read function:

```
uint16_t u16Read_Adc_Result(uint8_t u8AdcFifoBuffer)
```

To use the ADC read function it is necessary to indicate which RFIFO to use to gather the result from the last conversion. For practical purposes, the diagnostic feature uses RFIFO 0 to read any result from all ADC channels.

8 PWM Sample Application

To demonstrate the use of the emulated PWM driver, two PWM channels are configured to generate frequencies of 150 Hz on channel one and 100 Hz on channel two to drive a high quality lighting application.

Each PWM channel has a resolution frequency of 0.5%. To achieve a period resolution of 0.5% at 150 Hz, channel one of the periodic-interrupt-timer must be configured to service the interrupt routine every 33 µs based on the 64 MHz system frequency. The interrupt base time used in this example has been calculated by dividing 6.6 ms/ 200 to represent the resolution period proposed of 0.5%. The complete sample application is mounted into a multi-thread scheduler programmed with four tasks to be executed at different time intervals from 100 ms to 800 ms.

- The first task executes vfnSetDuty_Cycle after 100 milliseconds. This task changes the channel 0 duty cycle from 5% to 50% without modifying the 150 Hz period.
- The second task executes after 200 ms and changes the channel 1 duty cycle from 20% to 5% keeping the same 100 Hz period.
- The third task is executed after 400 ms changing channel two parameters. The PWM period changes from 150 Hz to 100 Hz and the duty cycle from 50% to 5%.
- The fourth task is executed after 800 ms changing the same parameters as the third task but with different values. The PWM period changes from 100 to 150 Hz and duty cycle value from 65 to 20%.

9 References

AN3225 — XGATE Library: PWM Driver.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN3737
Rev. 0
12/2008