

Add Duty Cycle To PWM Driver

The modification I made was to add 1 additional variable for duty cycle at the device driver:


```
~/linux-2.6.38/drivers/char$ ls mini6410_pwmHarry.c
```

```
//static void PWM_Set_Freq( unsigned long freq )  
static void PWM_Set_Freq( unsigned long freq, duty ) //Harry: add "duty"
```



User App Program:

```
while( 1 )  
{  
    int key;  
    set_buzzer_freq(freq);  
    printf( "\tFreq = %d\n", freq );  
    key = getch();  
    switch(key) {  
    case '+':  
        if( freq < 20000 )  
            freq += 10;  
        break;
```



static void PWM_Set_Freq(unsigned long freq, duty)

```
static void PWM_Set_Freq( unsigned long freq, duty ) //Harry: add "duty"
```

```
{
```

```
    unsigned long tcon;  
    unsigned long tcnt;  
    unsigned long tduty; //Harry  
    unsigned long tcfg1;  
    unsigned long tcfg0;  
    struct clk *clk_p;  
    unsigned long pclk;  
    unsigned tmp;
```

```
TCON  
TCNT  
TCFG0  
TCFG1
```

```
    tmp = readl(S3C64XX_GPFCON);  
    tmp &= ~(0x3U << 28);  
    tmp |= (0x2U << 28);  
    writel(tmp, S3C64XX_GPFCON);
```

```
    tcon = __raw_readl(S3C_TCON);  
    tcfg1 = __raw_readl(S3C_TCFG1);  
    tcfg0 = __raw_readl(S3C_TCFG0);  
    tcfg0 &= ~S3C_TCFG_PRESCALER0_MASK;  
    tcfg0 |= (50 - 1); //prescaler = 50
```

```
    tcfg1 &= ~S3C_TCFG1_MUX0_MASK;  
    tcfg1 |= S3C_TCFG1_MUX0_DIV16;    //mux = 1/16
```

pp. 322, CPU Datasheet

GPF14	[29:28]	00 = Input 10 = PWM TOUT[0]
GPF15	[31:30]	00 = Input 10 = PWM TOUT[1]

static void PWM_Set_Freq(unsigned long freq, duty)

```
__raw_writel(tcfg1, S3C_TCFG1);  
__raw_writel(tcfg0, S3C_TCFG0);
```

```
clk_p = clk_get(NULL, "pclk");  
pclk = clk_get_rate(clk_p);  
tcnt = (pclk/50/16)/freq;  
tduty = tcnt*duty;    //Harry: duty is x% of the period, duty cycle
```

```
/*-----
```

Harry: Feb 2016

S3C_TCNTB: Timer Counter Buffer Register

S3C_TCMPB: Timer Compare Buffer Register

```
-----*/
```

```
__raw_writel(tcnt, S3C_TCNTB(0));  
//__raw_writel(tcnt/2, S3C_TCMPB(0));  
__raw_writel(tduty, S3C_TCMPB(0));
```

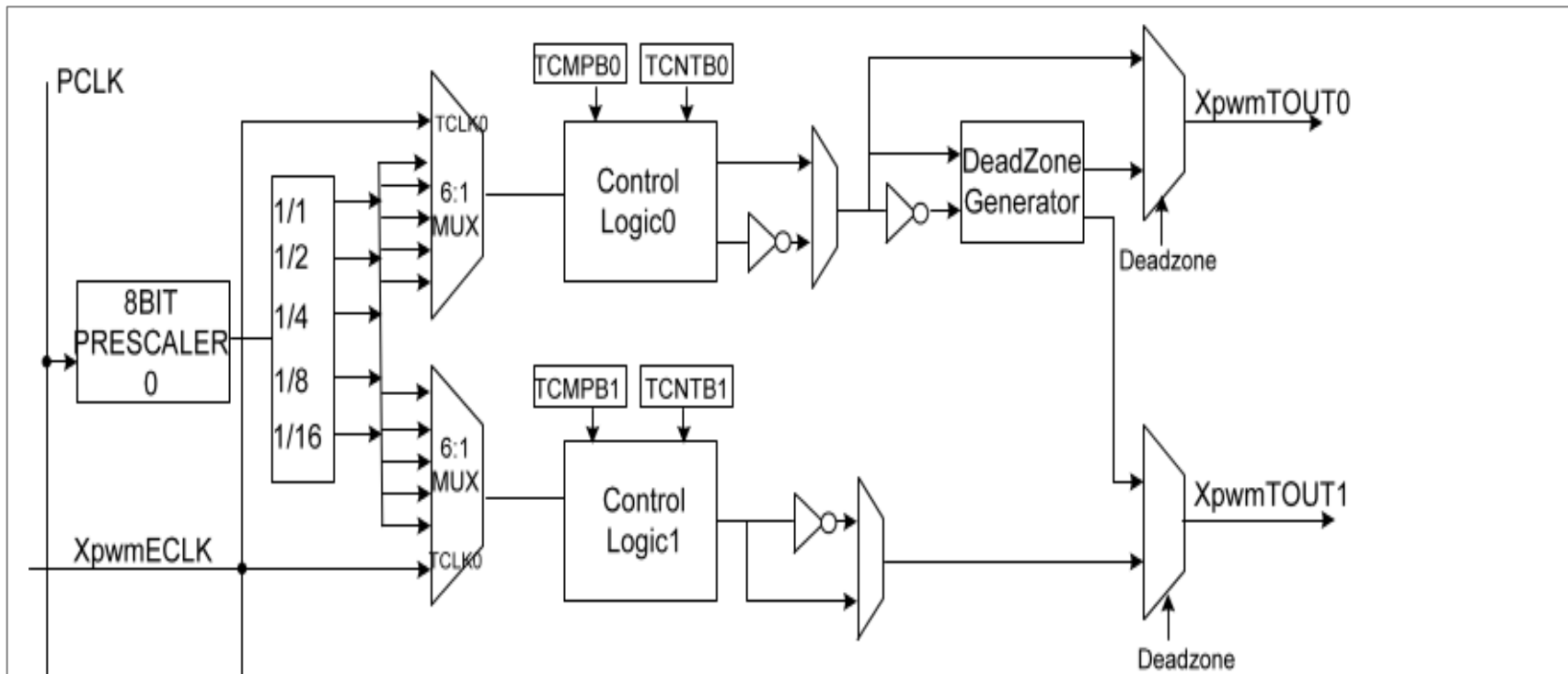
```
tcon &= ~0x1f;  
tcon |= 0xb;    //disable deadzone, auto-reload, inv-off,  
                //update TCNTB0&TCMPB0, start timer 0  
__raw_writel(tcon, S3C_TCON);
```

```
tcon &= ~2;    //clear manual update bit  
__raw_writel(tcon, S3C_TCON);
```

```
}
```

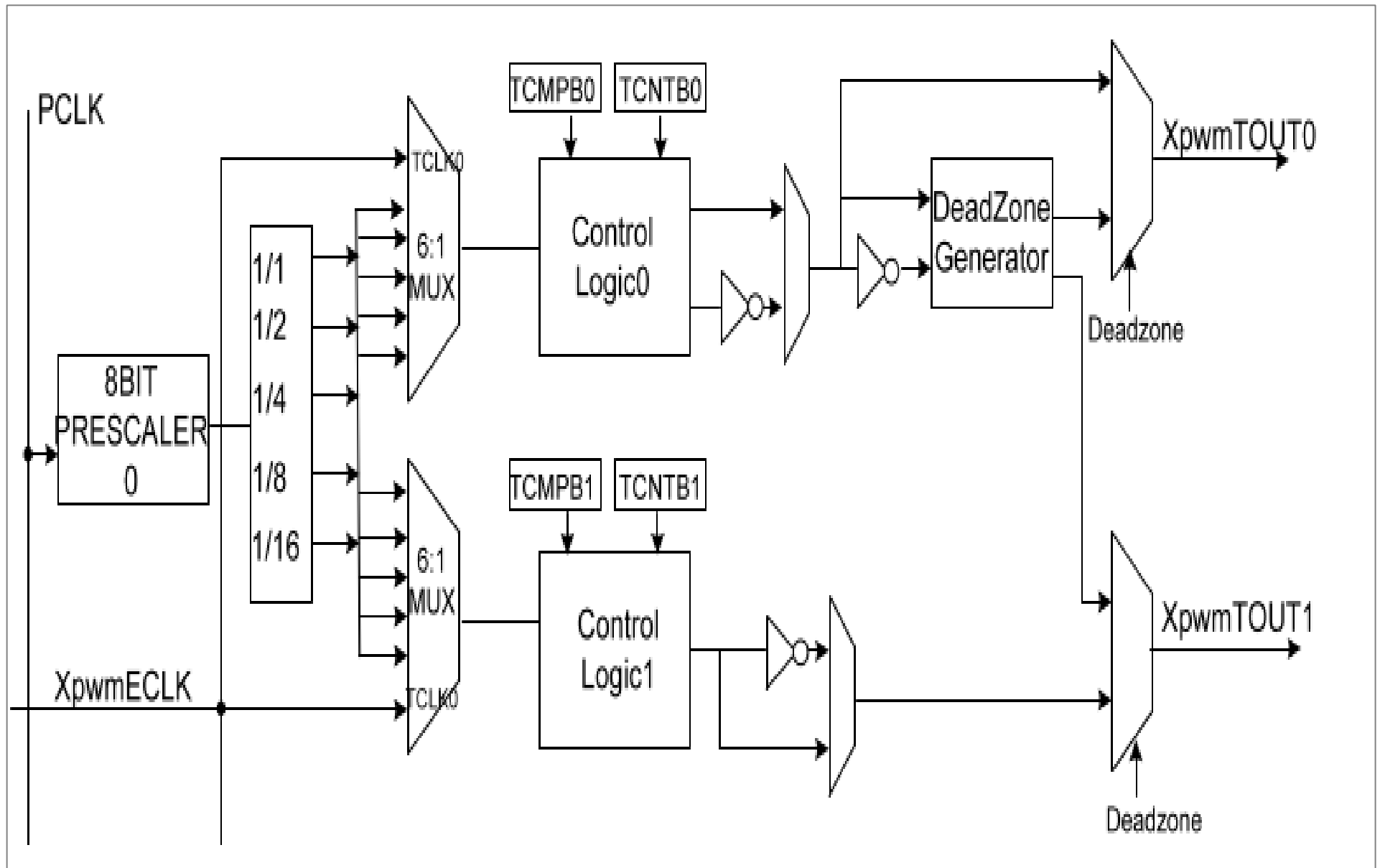
PWM 2 Output Timers

- (1) The S3C6410X CPU has five 32-bit timers to generate internal INT, Timers 0 and 1 has output pins and include a PWM function.
- (2) Two 8-bit Prescalers as 1th level division for the PCLK, Five Clock Dividers and Multiplexers for 2nd level of division for the Prescaler.
- (3) Supports Auto-Reload Mode and One-Shot Pulse Mode.



pp 1107 CPU
Datasheet

PWM SPRs



The Algorithm

Algorithm:

Step 1. The down-counter is initially loaded from the Timer Count Buffer register (TCNTBn).

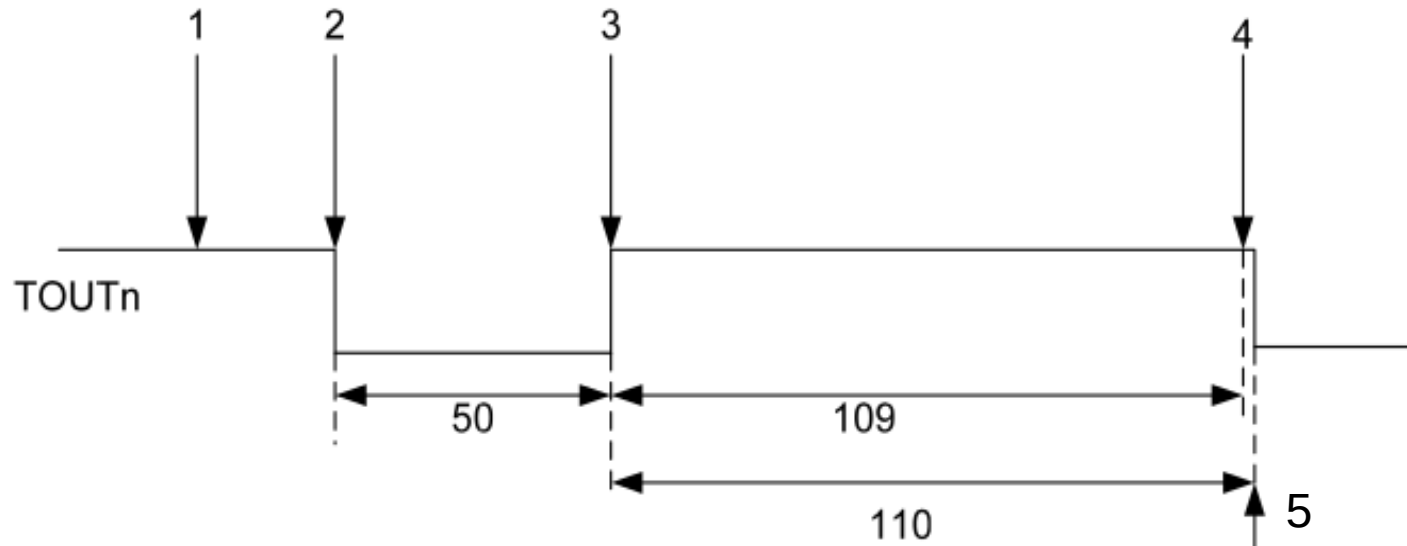
Step 2. For each clk, it will counts down, when the down count value in TCNTBn matches TCMPBn (Timer Compare Buffer) register. The output level changes. (e.g. the compare register determines the turn-on time of a PWM output.the

Step 3. When down-counter TCNTBn reaches 0, the interrupt is generated, one cycle is finished.

Step 4. TCNTBn can be automatically reloaded to start the next cycle.

Note: The TCNTBn and TCMPBn registers are double buffered to allow the timer parameters to be updated in the middle of a cycle. The new values will not take effect until the current timer cycle completes.

PWM Operation

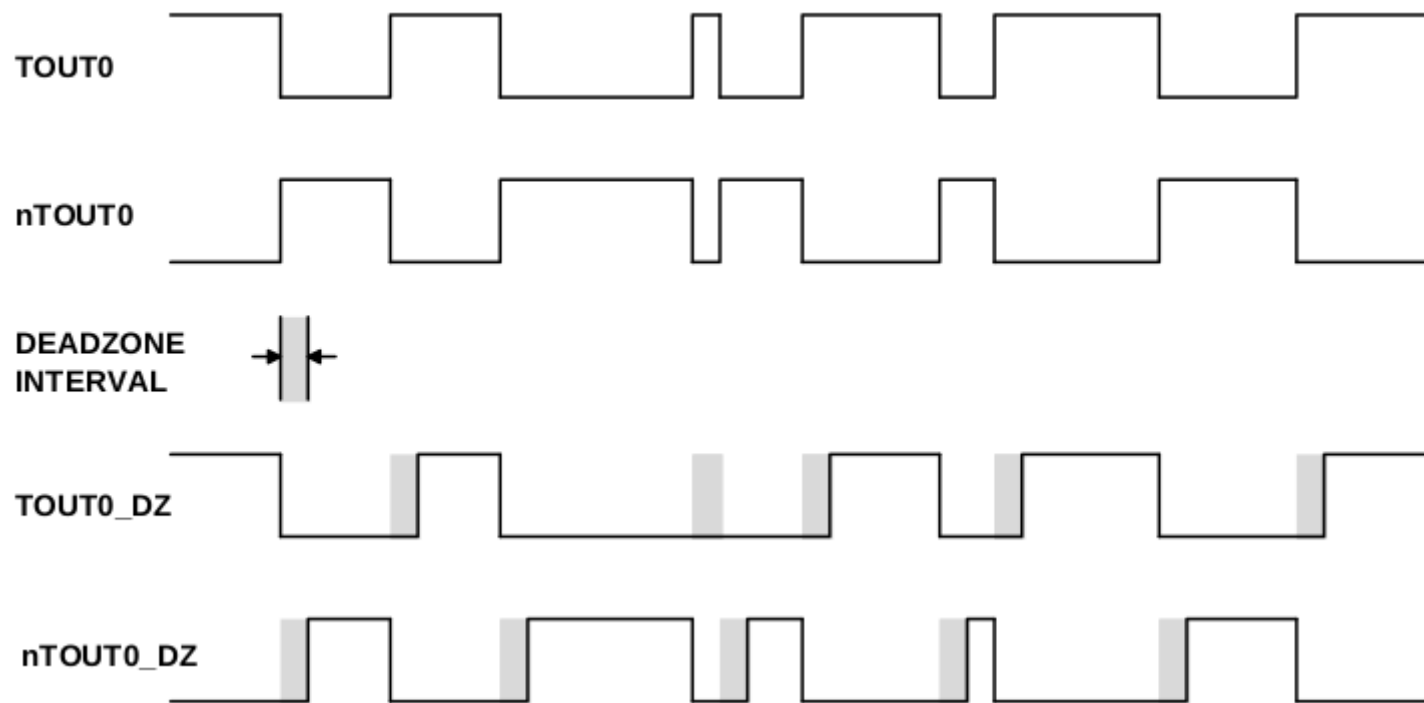


Example:

1. Initialize the TCNTBn with 159(50+109) and the TCMPBn with 109.
2. Start Timer by setting the start bit and manual update bit off.
The TCNTBn value of 159 is loaded into the down-counter, the output is driven low.
3. When down-counter counts down to the value in the TCMPBn register 109, the output is changed from low to high.
4. When the down-counter reaches 0, the interrupt request is generated.
5. The down-counter is automatically reloaded with TCNTBn, which restarts the cycle.

PWM Dead Zone Generator

The deadzone is for the control of power devices, which is used to insert the time gap between a turn-off of one device and a turn-on of the other device. This time gap prohibits the two switching devices turning on simultaneously. TOUT0 is the PWM output. nTOUT0 is the inversion of the TOUT0. If the dead-zone is enabled, the output waveform of TOUT0, nTOUT0 will be TOUT0_DZ and nTOUT0_DZ. TOUT0_DZ and nTOUT0_DZ never can be turned on simultaneously by the dead zone interval. For functional correctness, the deadzone length must be set smaller than compare counter value.



From CPU
Datasheet
pp. 1115

TCFG0, TCNTBn and TCMPBn

32.4.1.1 TCFG0 (Timer Configuration Register) , pp 1118

Register	Offset	R/W	Description
TCFG0	0x7F006000	R/W	Timer Configuration Register 0 that configures the two 8-bit Prescaler and DeadZone Length

Timer input clock Frequency = $PCLK / (\{\text{prescaler value} + 1\} / \{\text{divider value}\})$
{prescaler value} = 1~255
{divider value} = 1, 2, 4, 8, 16, TCLK

32.4.1.2 TCFG1 (Timer Configuration Register)

Register	Offset	R/W	Description
TCFG1	0x7F006004	R/W	Timer Configuration Register 1 that controls 5 MUX and DMA Mode Select Bit

32.4.1.3 TCON (Timer Control Register)

Register	Offset	R/W	Description
TCON	0x7F006008	R/W	Timer Control Register

32.4.1.5 TCMPB0 (Timer0 Compare Register)

Register	Offset	R/W	Description
TCMPB0	0x7F006010	R/W	Timer 0 Compare Buffer Register

