PWM(FTM) example in Kinetis Design Studio with FDRM-K64F

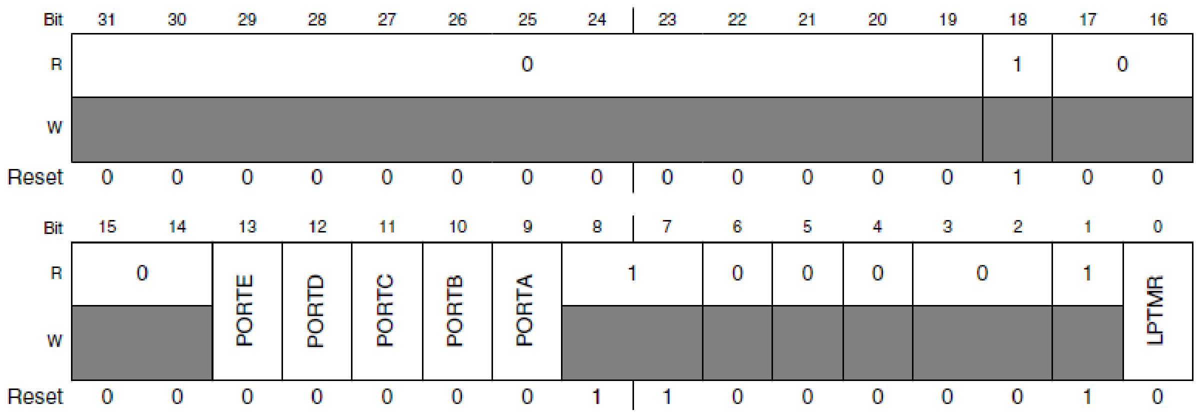
By:
Paul Garate
Augusto Panecatl

Description

In this document you will find a detailed step by step guide of how to configure FTM as PWM on Kinetis K devices using Kinetis Design Studio, using the FRDM-K64F120 evaluation board.

1. Clock Gating

First of all, we need to enable the clock gate corresponding to the modules we will use. The clocks that we need to enable are those of FTM0 and PortD.



Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0													1	0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0		PORTE	PORTD	PORTC	PORTB	PORTA	1		0	0	0	0		1	LPTMR
W																
Reset	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0

13 PORTE	Port E Clock Gate Control This bit controls the clock gate to the Port E module. 0 Clock disabled 1 Clock enabled
12 PORTD	Port D Clock Gate Control This bit controls the clock gate to the Port D module. 0 Clock disabled 1 Clock enabled

SIM_SCGC5_PORTn_MASK are defined as masks to enable the module's clock, where "n" corresponds to the specific GPIO PORT we want to activate, i.e:

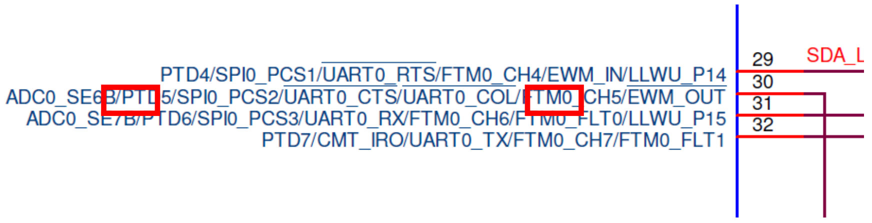
SIM_SCGC5 = SIM_SCGC5_PORTD_MASK;

By declaring the mask we are writing 0x400 to the SIM_SCGC5 register, setting up the 10th bit of the System Clock Gating Control Register 5 which enables Port D; the FTM0 is assigned in the PIN5 of the port D, so we need to enable both clocks, one of the FTM0 and the PORTD.

```
SIM_SCGC6 |= SIM_SCGC6_FTM0_MASK;      /*Enable the FTM0 clock*/
SIM_SCGC5 |= SIM_SCGC5_PORTD_MASK;     /*Enable the PORTD clock*/
```

2. Pin Control Register configuration

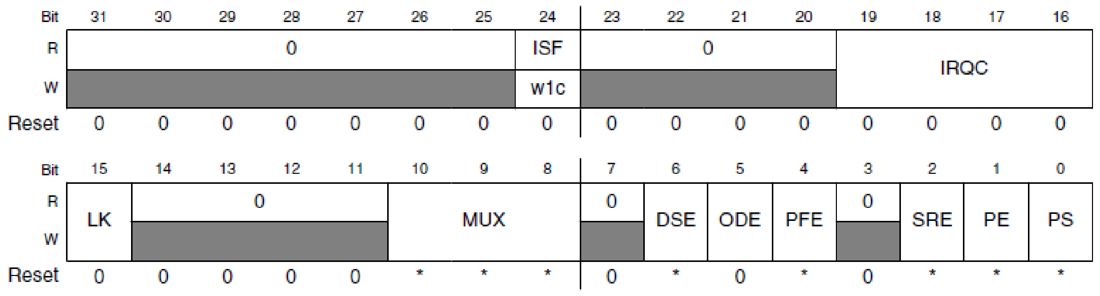
Once the clock gating has been setup we need to configure the pin function using the multiplexor, according to the FRDM-K64's schematic the FTM0 is assigned to several pins, but we will select PTD5.



PTD4/SPI0_PCS1/UART0_RTS/FTM0_CH4/EWM_IN/LLWU_P14
ADC0_SE6B/PTD5/SPI0_PCS2/UART0_CTS/UART0_COL/FTM0_CH5/EWM_OUT
ADC0_SE7B/PTD6/SPI0_PCS3/UART0_RX/FTM0_CH6/FTM0_FLT0/LLWU_P15
PTD7/CMT_IRO/UART0_TX/FTM0_CH7/FTM0_FLT1
29
30
31
32
SDA_L

The next step is to configure the Pin Control Register to define the pin function:

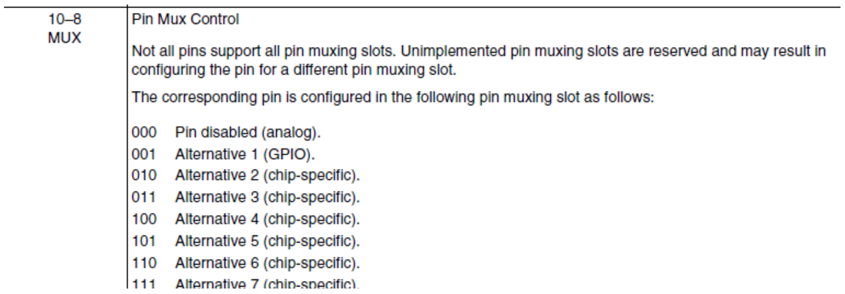
Pin Control Register n (PORTx_PCRn)



Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0							ISF	0				IRQC			
W								w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

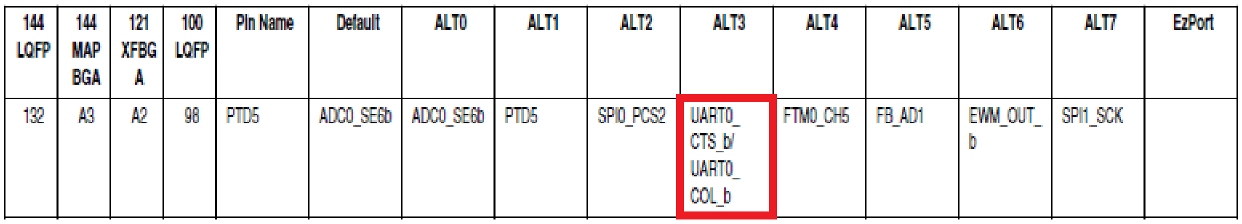
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	LK	0				MUX			0	DSE	ODE	PFE	0	SRE	PE	PS
W																
Reset	0	0	0	0	0	*	*	*	0	*	0	*	0	*	*	*

The bits we need to configure in are those in the MUX field.



10–8 MUX	Pin Mux Control Not all pins support all pin muxing slots. Unimplemented pin muxing slots are reserved and may result in configuring the pin for a different pin muxing slot. The corresponding pin is configured in the following pin muxing slot as follows: 000 Pin disabled (analog). 001 Alternative 1 (GPIO). 010 Alternative 2 (chip-specific). 011 Alternative 3 (chip-specific). 100 Alternative 4 (chip-specific). 101 Alternative 5 (chip-specific). 110 Alternative 6 (chip-specific). 111 Alternative 7 (chip-specific).

According to the board's schematic we need to configure PTD5, according to the multiplexing chart we need to set the pin as Alternative 4:



144 LQFP	144 MAP BGA	121 XFBGA	100 LQFP	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	EzPort
132	A3	A2	98	PTD5	ADC0_SE6b	ADC0_SE6b	PTD5	SPI0_PCS2	UART0_CTS_b/ UART0_COL_b	FTM0_CH5	FB_AD1	EWM_OUT_b	SPI1_SCK	

In configuration register ***PORTx_PCRn*** "**x**" corresponds to the port whilst "***n***" corresponds to the pin

```
        PORTD_PCR5 = PORT_PCR_MUX(4);        /*MUX = ALT 4*/
```

*PORT_PCR_MUX(4) = 1024

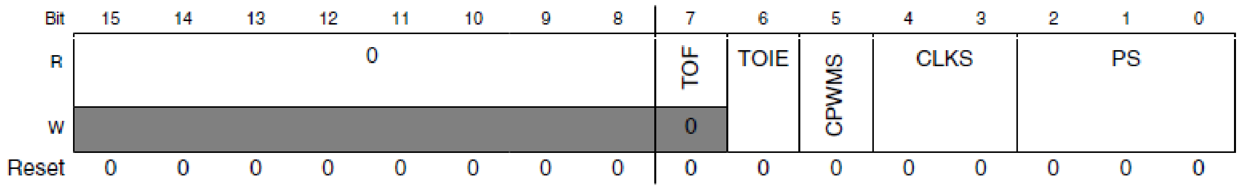
We also to enable the IRQ of the FTM0 using the next line:

```
   NVIC_EnableIRQ(FTM0_IRQn);            /*Enable the FTM Interrupt*/
```

3. Configuring the FTM

The Status and control register contains the overflow status flag and control bits used to configure the interrupt enable, FTM configuration, clock source, and prescaler factor.

Status And Control (FTMx_SC)



Field	Description
6 TOIE	Timer Overflow Interrupt Enable Enables FTM overflow interrupts. 0 Disable TOF interrupts. Use software polling. 1 Enable TOF interrupts. An interrupt is generated when TOF equals one.
4–3 CLKS	Clock Source Selection Selects one of the three FTM counter clock sources. This field is write protected. It can be written only when MODE[WPDIS] = 1. 00 No clock selected. This in effect disables the FTM counter. 01 System clock 10 Fixed frequency clock 11 External clock
2–0 PS	Prescale Factor Selection Selects one of 8 division factors for the clock source selected by CLKS. The new prescaler factor affects the clock source on the next system clock cycle after the new value is updated into the register bits. This field is write protected. It can be written only when MODE[WPDIS] = 1. 000 Divide by 1 001 Divide by 2 010 Divide by 4 011 Divide by 8 100 Divide by 16 101 Divide by 32 110 Divide by 64 111 Divide by 128

We will enable the TOEI (Time Overflow Enable Interrupt), select the System Clock as our source clock in CLKS, as well as set The Prescale Factor Selection dividing by 128 with the next line:

```
        FTM0_SC |= 0x004F;        /*Setting TOIE = 1,CLKS =  01, PS = 111*/
```

The next thing to do is Configure the ***FTMx_MOD*** The Modulo register contains the modulo value for the FTM counter. After the FTM counter reaches the modulo value, the overflow flag (TOF) becomes set at the next clock.

Modulo (FTM*x*_MOD)

Address: Base address + 8h offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved																MOD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FTMx_MOD field descriptions

Field	Description
31–16 Reserved	This field is reserved.
15–0 MOD	Modulo Value

 We are Setting this value in 32000, we must be careful while the value, since the FTM is a 16-bit counter, the ***FTM_MOD*** value should never exceed ***32768***.

```
FTM0_MOD = 32000;          /*Setting the Modulo register = 32000*/
```

Next thing ot configure is ***FTMx_CnSC***, this register contains the channel-interrupt-status flag and the control bits used to configure the interrupt enable, channel configuration and pin function.

Table 40-67. Mode, edge, and level selection

1X	10	Edge-Aligned PWM	High-true pulses (clear Output on match)
	X1		Low-true pulses (set Output on match)

We need only to configure bits ***MSnB:MSnA*** and ***ELSnB:ELSnA*** to set the FTM to work as a center aligned PWM:

```
FTM0_C5SC |= 0x0028;             /*Setting MSB = 1, ELSnB = 1*/
```

The last thing to do is setting the signal's duty cycle/channel value, using the ***FTMx_CnV*** register.

Channel (n) Value (FTM*x*_C*n*V)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0																VAL															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FTMx_C*n*V field descriptions

Field	Description
31–16 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
15–0 VAL	Channel Value Captured FTM counter value of the input modes or the match value for the output modes

We will assign the channel an initial value of 500.

```
FTM0_C5V = 500;             /*Value of the Channel*/
```

4. Code

Our main code consists in a simple counter just to have the microcontroller doing something.

```
int i;
for (;;)
  {
   i++;                     /*Just a Count*/
  }
```

Here is the important thing, all the PWM handling is performed on an interrupt routine, every time the interrupt triggers the channel value will change as well. When the interrupt is triggered the code will save the channel value in our variable ***ChannelValue***, after that the interrupt flag must be cleared so the next interrupt can be detected, after that we compare the current channel value stored in ***ChannelValue*** to the value of the ***TMP0_MOD*** (***32000***), if the channel value is bigger than the Modulo Value; ***FTM0_C5V*** will be reset, if the channel value is smaller the code will add ***500*** to the ***FTM0_C5V***.

```
void FTM0_IRQHandler (void)
       {
       unsigned long ChannelValue = FTM0_C5V; /*Take the value of the Channel to
                                              compare it*/
       (void)FTM0_SC;
       FTM0_SC |= 0x0080;                     /*FTM counter has overflow*/
       if(ChannelValue < 32000)               /*Channel Value > Modulo Value*/
              {
              FTM0_C5V += 500;                /*Add 500 to Channel*/
              }
       else
              {
               FTM0_C5V = 0;                  /*Set Channel in 0*/
              }
       }
```

Now, what is the frequency and period of our PWM signal?. Taking into consideration the microcontroller is running at 21MHz (MCG= FEI) and the fact that we selected a prescale divide of 128 in the ***FTM0_SC***, it means we have a clock frequency of 21MHz/128 = 164KHz; if we set the modulo value ***FTM0_MOD= 32000*** we have a PWM frequency of 164KHz/32000 = 5Hz and a frequency of 1/5Hz = 200ms.