The capture feature of eFlexPWM module

The eFlexPWM module is integrated in DSC MC56F84xxx and Mc56F82xxx family, Kinetis KV family, i.mxrt family and mpc56xx family, it is widely used in motor control, energy conversion applications…

As the name implies that eFlexPWM is flexible to generate PWM signals and output the PWM signals to PWM_nA, PWM_nB and PWM_nX pins, here "n" means the sub-module index of eFlexPWM module. One eFlexPWM module can generate 12 PWM signals. The capture feature of the eFlexPWM module is a secondary feature for the module, like all the capture feature, the eFlexPWM can accept the external captured signal, can detect the predefined edge of the captured signal and trigger an interrupt, can figure out the duty cycle and cycle time of the external captured signal. As you know that the eFlexPWM module has 4 sub-modules: SM0/SM1/SM2/SM3, each sub-module has three signals: PWM_nA, PWM_nB and PWM_nX, each signal of PWM_nA, PWM_nB and PWM_nX can be configured as captured pin and can be connected to external captured signal.

1)pin assignment of the captured signal.

As you know that the eFlexPWM signal pins PWM_nA, PWM_nB and PWM_nX can OUTPUT PWM signals, when you do not need them to output PWM signals, they can be configured as captured signal input pin.

Different processor has different ways to configure the pad as eFlexPWM pins PWM_nA, PWM_nB and PWM_nX, anyway, after the pads are configured as PWM signals pins, if you set the corresponding bits in PWM_OUTEN register, the pad are PWM output pins, if you clear the corresponding bits in PWM_OUTEN register, the pad are captured signal input pins.

2)the capture function mechanism of eFlexPWM module

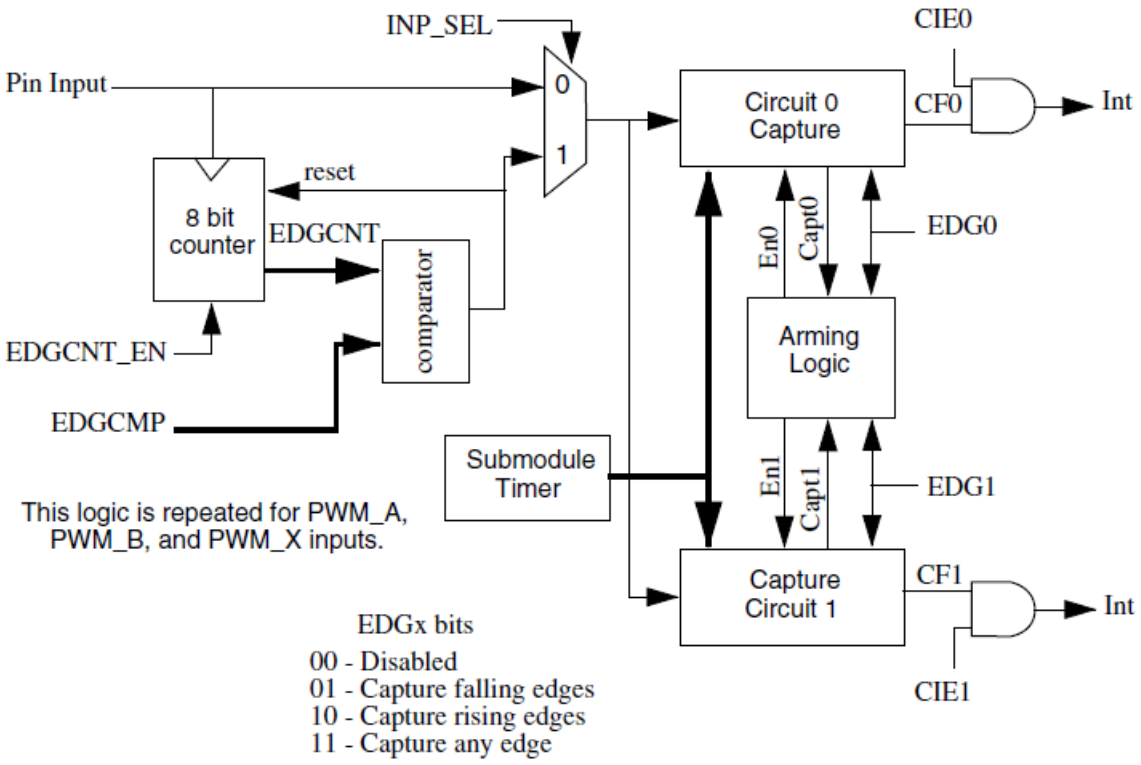


Figure 28-263. Enhanced Capture (E-Capture) Logic

From above figure, we can see that the captured signal can be from two paths, one is from the pad directly, another is from the output of the internal comparator, which can be a divider of the input captured signal so that it can accept higher frequency signal.

From the above figure, there are two capture circuit: capture circuit 0 and capture circuit 1, Each circuit can be configured independently for example, triggering edge mode: rising/falling edge, one-shot mode or continuous mode, whether an interrupt is triggered. When the edge of the captured signal which is predefined by capture circuit 0 is detected, the current value of sub-module counter PWM_SMnCNT is copied to PWM_SMn_CVAL0 automatically, When the edge of the captured signal which is predefined by capture circuit 1 is detected, the current value of sub-module counter PWM_SMnCNT is copied to PWM_SMn_CVAL1 automatically. User can set the corresponding bits in PWMA_SM*n*INTEN to determine whether an interrupt is triggered once an edge of the capture signal is detected.

The following figure is the interrupt enable register, the CA1IE means the circuit 1 capture interrupt for all PWM_nA pin, in detail for PWM_0A, PWM_1A, PWM_2A and PWM_3A. The CA0IE means the capture circuit 0 interrupt for PWM_nA pin, in detail for PWM_0A, PWM_1A,PWM_2A and PWM_3A. The CX1IE means the capture circuit 1 interrupt for all PWM_nX pins of the eFlexPWM module. The CX0IE means the capture circuit 0 interrupt for all PWM_nX pins of the eFlexPWM module.

Because one interrupt can be triggered by multiple sources for example SM0, SM1, SM2 or SM3, so in the ISR, user has to check the PWM_SMnSTS to determine the interrupt source.

28.3.19 Interrupt Enable Register (PWMA_SM*n*INTEN)

Address: E600h base + 13h offset + (48d × i), where i=0d to 3d

Bit	15	14	13	12	11	10	9	8
Read	0		REIE	RIE	CA1IE	CA0IE	CB1IE	CB0IE
Write								
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
Read	CX1IE	CX0IE	CMPIE					
Write								
Reset	0	0	0	0	0	0	0	0

PWMA_SM*n*INTEN field descriptions

3)figure out the duty cycle and cycle time of the captured signal
First of all, let's describe the capture register,
If the capture pin is PWM_nA, this is the register group:
Control register: PWPWMA_SM*n*CAPTCTRLA
Compare register: PWMA_SM*n*CAPTCOMPA
Capture Value0 register for capture circuit 0: PWMA_SM*n*CVAL2
Capture value0 cycle register for capture circuit 0: PWMA_SM*n*CVAL2CYC

Capture Value1 register for capture circuit 1: PWMA_SM*n*CVAL3
Capture value1 cycle register for capture circuit 1: PWMA_SM*n*CVAL3CYC

If the capture pin is PWM_nB, this is the register group:
Control register: PWMA_SM*n*CAPTCTRLB
Compare register: PWMA_SM*n*CAPTCOMPB
Capture Value0 register for capture circuit 0: PWMA_SM*n*CVAL4
Capture value0 cycle register for capture circuit 0: PWMA_SM*n*CVAL4CYC

Capture Value1 register for capture circuit 1: PWMA_SM*n*CVAL5
Capture value1 cycle register for capture circuit 1: PWMA_SM*n*CVAL5CYC

If the capture pin is PWM_nX, this is the register group:
Control register: PWPWMA_SM*n*CAPTCTRLX
Compare register: PWMA_SM*n*CAPTCOMPX
Capture Value0 register for capture circuit 0: PWMA_SM*n*CVAL0
Capture value0 cycle register for capture circuit 0: PWMA_SM*n*CVAL0CYC

Capture Value1 register for capture circuit 1: PWMA_SM*n*CVAL1
Capture value1 cycle register for capture circuit 1: PWMA_SM*n*CVAL1CYC

The control register sets up the edge mode for the capture circuit 0 and capture circuit 1, triggering source, counting mode. The compare register can set up the divider value if the triggering source is from compare output. The Capture Value0 register is the register which stores the value from the current counter value of sub-module automatically when the capture circuit0 detects the capture edge. The Capture Value1 register is the register which store the value from the current counter value of sub-module automatically when the capture circuit1 detects the capture edge. The Capture value0 cycle register is incremented each time the counter is loaded with the INIT value at the end of a PWM modulo cycle. The Capture value1 cycle register is incremented each time the counter is loaded with the INIT value at the end of a PWM modulo cycle.

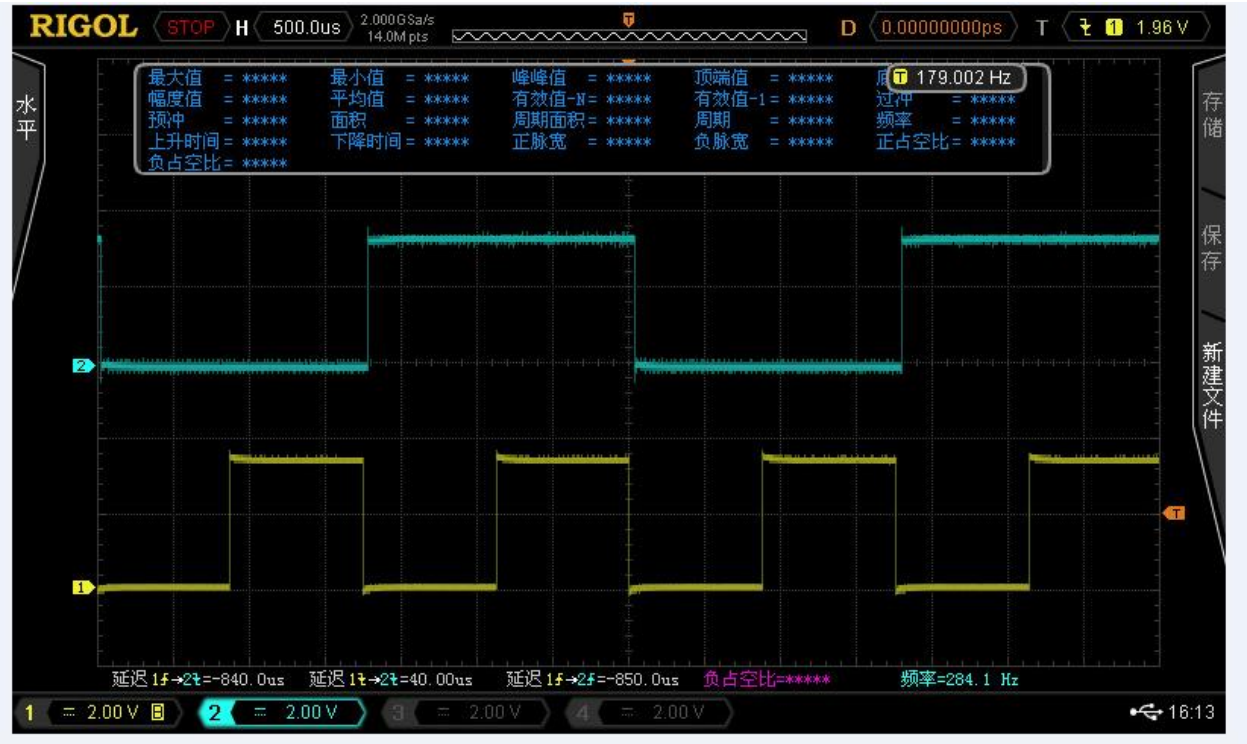
For duty cycle time test, user can set up that the triggering mode of the capture circuit 0 is rising edge, the triggering mode of the capture circuit 1 is falling edge.

the duty cycle time is [PWMA_SM*n*CVAL1+PWMA_SM*n*CVAL1CYC_LOCKED*(PWMA_SMnVAL1-PWMA_SMnINIT)]- [PWMA_SM*n*CVAL0+PWMA_SM*n*CVAL0CYC_LOCKED*(PWMA_SMnVAL1-PWMA_SMnINIT)]

For cycle time test, user can set up the triggering mode of the capture circuit 0 and the capture circuit 1 in the same mode, it is okay.

the cycle time is [PWMA_SM*n*CVAL1+PWMA_SM*n*CVAL1CYC_LOCKED*(PWMA_SMnVAL1-PWMA_SMnINIT)]- [PWMA_SM*n*CVAL0+PWMA_SM*n*CVAL0CYC_LOCKED*(PWMA_SMnVAL1-PWMA_SMnINIT)]

If the captured signal is asynchronous with the PWM signal and it's cycle time is larger than the PWM cycle time, I suggest that both the capture circuit 0 and capture circuit 1 interrupts are enabled, in the ISR of capture, check the capture source, if the capture circuit 0 generate the interrupt, copy the PWMA_SM*n*CVAL0CYC register value to a variable in memory, let's called for example PWMA_SM*n*CVAL0CYC_LOCKED, if the capture circuit 1 generate the interrupt, copy the PWMA_SM*n*CVAL1CYC register value to a variable in memory for example PWMA_SM*n*CVAL1CYC_LOCKED.



For above screenshot of oscilloscope, the channel 1 signal is the captured signal, in my test it is from PWMA_0A(GPIOE1, pin A40 of primary connector) which output PWM signal, the channel 2 signal is GPIOA8(pin 1 of J503) on TWR-8400 board which is toggled in capture ISR, user can see that falling edge of captured signal triggers interrupt.
The project is developed under CodeWarrior for mcu ver10.6 tools and TWR-8400 board.
Note: in the doc, the eFlexPWM, PWM module and PWMA module are used indiscriminatingly, they are the same, mean eFlexPWM module.
code snippet:

```
void PWMA_init(void);
void GPIOE_init();
void CLOCK_init(void);
void testGPIO_A8(void);
void testCrossBarFun(void);
void pinPWMA_1X(void);
void captureFuncPWMA_1A(void);
void PWM_ISR_SETTING();
void captureISR(void);
unsigned int changeFlag;


unsigned int sample[8];
```

```
unsigned int flag=0;
unsigned int varEdge0[100],varEdge1[100],varDiff[100];
unsigned int varsm0cnt[100], varsm1cnt[100];
unsigned int timeOverFlow[100];

int main(void)
{
	DisableWatchdog;
	CLOCK_init();
//	testCrossBarFun();
	PWMA_init();
	GPIOE_init();
	testGPIO_A8();
	pinPWMA_1X();
	PWM_ISR_SETTING();
	captureFuncPWMA_1A();
	PWMA_MCTRL|=0x0100;  //enable the PWM module
	PWMA_MCTRL|=0x0200;  //enable the PWM module
	PWMA_MCTRL|=0x0400;  //enable the PWM module
	PWMA_MCTRL|=0x0800;  //enable the PWM module
	__asm(bfclr #$300, SR);

	for(;;)
	{
		asm(nop);
	}
	return(0);

}


void PWMA_init(void)
{
	//SM0 initialization

	PWMA_SM0INIT=0xF000;
	PWMA_SM0VAL0=0x0000;
	PWMA_SM0VAL1=0x1000;	//the modulo is 256
	PWMA_SM0VAL2=0xF800;  //75% duty cycle
	PWMA_SM0VAL3=0x800;
	PWMA_SM0VAL4=0xFC00;  //25% duty cycle
	PWMA_SM0VAL5=0x300;
	PWMA_SM0CTRL2=0x0000; //complementary mode for PWM0A and PWM0B, IP bus clock
	PWMA_SM0CTRL=0x3400; //4 PWM opportunity, PWM clock=Fclk
	PWMA_SM0OCTRL=0x0000; //PWM does not inverter, PWM forced to logic 0 in fault
state
	PWMA_SM0TCTRL=0x0000;
	PWMA_SM0INTEN=0x0000; //disable all interrupt
	PWMA_SM0DISMAP0=0x0000; //Disable fault mask
	PWMA_SM0DISMAP1=0x0000; //Disable fault mask
	PWMA_SM0DTCNT0=0x0000; //dead time is set to 0
	PWMA_SM0DTCNT1=0x0000;
	PWMA_SM0CTRL|=0x04;

	//SM1 module initialization
```

```
       PWMA_SM1INIT=0xF000;
       PWMA_SM1VAL0=0xFC00;
       PWMA_SM1VAL1=0x300;        //the modulo is 256
       PWMA_SM1VAL2=0xFC00;  //75% duty cycle
       PWMA_SM1VAL3=0x300;
       PWMA_SM1VAL4=0xFC00;  //25% duty cycle
       PWMA_SM1VAL5=0x300;
       PWMA_SM1CTRL2=0x2200; //independent mode, IP bus clock, the INIT_SEL should be
10, which means
       //that the SM0 synchronize thw SM1, PWMX_INIT is set as a test
       PWMA_SM1CTRL=0x3400; //4 PWM opportunity, PWM clock=Fclk
       PWMA_SM1OCTRL=0x0000; //PWM does not inverter, PWM forced to logic 0
       PWMA_SM1TCTRL=0x0000;
       PWMA_SM1INTEN=0x0000;
       PWMA_SM1DISMAP0=0x0000; //Disable fault mask
       PWMA_SM1DISMAP1=0x0000; //Disable fault mask
       PWMA_SM1DTCNT0=0x0000; //dead time is set to 0
       PWMA_SM1DTCNT1=0x0000;
       PWMA_SM1CAPTCTRLX|=0x40; //PWMA1_X output
       PWMA_OUTEN|=0x0FF0;  //enable PWM output
//     PWMA_FCTRL)=0xF000; //fault logic setting
       PWMA_SM1CTRL|=0x04;

             //test PWM2_X output signal
             //when the SM2 module of PWMA is synchronized by SM0 sync signal, the
PWM1_X/PWM2_X/PWM3_X can output arbitary
             //waveform without any restriction because of PWMA_SMxVAL1 register,
only PWMA_SM0VAL1 register is used to control the
             //PWMA frequency
             PWMA_SM2INIT=0xF000;
             PWMA_SM2VAL0=0xF800;
             PWMA_SM2VAL1=0x800;        //the modulo is 256
             PWMA_SM2VAL2=0xF800;  //75% duty cycle
             PWMA_SM2VAL3=0x800;
             PWMA_SM2VAL4=0xFC00;  //25% duty cycle
             PWMA_SM2VAL5=0x300;
             PWMA_SM2CTRL2=0x2200; //independent mode, IP bus clock, the INIT_SEL
should be 10, which means
                   //that the SM0 synchronize the SM2 module
             PWMA_SM2CTRL=0x3400; //4 PWM opportunity, PWM clock=Fclk
             PWMA_SM2OCTRL=0x0000; //PWM does not inverter, PWM forced to logic 0
             PWMA_SM2TCTRL=0x0000;
             PWMA_SM2INTEN=0x0000;
             PWMA_SM2DISMAP0=0x0000; //Disable fault mask
             PWMA_SM2DISMAP1=0x0000; //Disable fault mask
             PWMA_SM2DTCNT0=0x0000; //dead time is set to 0
             PWMA_SM2DTCNT1=0x0000;
             PWMA_SM2CTRL|=0x04;
             //PWMA global register setting
             PWMA_OUTEN|=0x0FF0;  //enable PWM output
             PWMA_MASK=0x0000;  //disable PWM mask
             PWMA_SWCOUT=0x0000;  //determine dead time logic
             //     PWMA_FCTRL)=0xF000; //fault logic setting
```

```
            PWMA_MCTRL|=0x0007; //must use the instruction, otherwise, the counter
will disorder, IPOL is cleared, PWM23 manipulate the duty cycle

      return;
}


void GPIOE_init()
{
      //GPIOE port are multiplexed with PWMA
      GPIOE_PER|=0x00FF;  //Enable PWM0A&PWM0B output from GPIOE0 and E1.
      SIM_GPSEL=0x00;
      SIM_GPSEH=0x00;
      return;
}


void CLOCK_init(void)
{
      SIM_PCE3|=0x00FF;  //enable PWMA and PWMB all channels
      SIM_PCE0|=0xFF7F;  //enable all Timer and GPIO A/B/C/D/E/G/F
      SIM_PCE2|=0x180; //enable SAR ADC clock and Cyclic ADC clock
      return;
}


void testCrossBarFun(void)
{
      GPIOC_PER|=0x4000; //writting the GPIOC peripheral enable reg
//    XB_XBC0)&=0xFF00; //select GND output, logic 0, pin 23 in J501
      asm(nop);
      asm(nop);
      asm(nop);
      asm(nop);
      asm(nop);
//    SIM_GPS1)|=0x1000; //select XB_OUT0 output, set the C14 bit in SIM_GPS1 reg
      asm(nop);
      asm(nop);
      asm(nop);
      asm(nop);
      asm(nop);
//    XB_XBC0)|=0x0001; //select VDD output, GPIOC14 output logic 1
      asm(nop);
      asm(nop);
      asm(nop);
      asm(nop);
      asm(nop);
      return;
}

void testGPIO_A8(void)
{
      GPIOA_PER&=~(0x100);
      GPIOA_DDR=0x100; //set GPIO A8 output
      GPIOA_DR=0x100; //set the A8 bit
```

```
	GPIOA_DR^=0x100; //toggle the A8 bit
	GPIOA_DR^=0x100; //toggle the A8 bit
	asm(nop);
	GPIOA_DR^=0x100; //toggle the A8 bit
	GPIOA_DR^=0x100; //toggle the A8 bit
	GPIOA_DR^=0x100; //toggle the A8 bit

	return;
}
//GPIOG9 PWMB_1X PWMA_1X TA3 XB_OUT11, pin 30 on J503 on TWR-8400 board
//connect the measured signal to PWMA_1A pin GPIOG9, test the duty cycle of the
signal
void pinPWMA_1X(void)
{
	//let PWMA_1X signal output
	GPIOG_PER|=0x200;
	SIM_GPSGH&=~(0x30);
	SIM_GPSGH|=0x01<<2; //set G9 bits to be 01 in binary:01 Function = PWMA_1X;
}

void captureFuncPWMA_1A(void)
{
	//bit setting:
	//EDGCNTX_EN=1 Edge counter enabled
	//INP_SELX=0 Raw PWM_X input signal selected as source.
	//EDGX1=01 Capture falling edges
	//EDGX0=10 Capture rising edges
	//ONESHOTX=0 Free running mode is selected.
	PWMA_SM1CAPTCTRLX&=0x00;
	PWMA_SM1CAPTCTRLX=0x98;
	PWMA_SM1CAPTCTRLX|=0x01; //start capture function
	PWMA_OUTEN&=~(0x0F); //disable all PWMA_X pins
}

void PWM_ISR_SETTING()
{
	//interrupt priority setting
	INTC_IPR8|=0xC000; //set PWMA_CAP interrupt source priority
	//enable the PWM_SM0 reload interrupt
	PWMA_SM1INTEN|=0x80; //edge1(falling edge) trigger interrupt

	return;

}

#pragma interrupt on
void captureISR(void)
{
	static unsigned int i=0;
	if((PWMA_SM1STS)&0x80)
	{
		varEdge0[i]=PWMA_SM1CVAL0;
		varEdge1[i]=PWMA_SM1CVAL1;
		varDiff[i]=varEdge1-varEdge0;
		varsm0cnt[i]=PWMA_SM0CNT;
```

```
                varsm1cnt[i]=PWMA_SM1CNT;
                timeOverFlow[i]=PWMA_SM1CVAL1CYC;
                i++;
                if(i>=50)
                        {
                        asm(debughlt);
                        }
        //clear the interrupt flag
                PWMA_SM1STS|=0xC0; //clear the CFX1 and CFX0 bits

                asm(nop);
                GPIOA_DR^=0x100; //toggle the A8 bit
        }
}
```