CG2271 Real Time Operating Systems

Lab Lecture 2.1
Timer Programming
(Reference Guide Chapter 29)

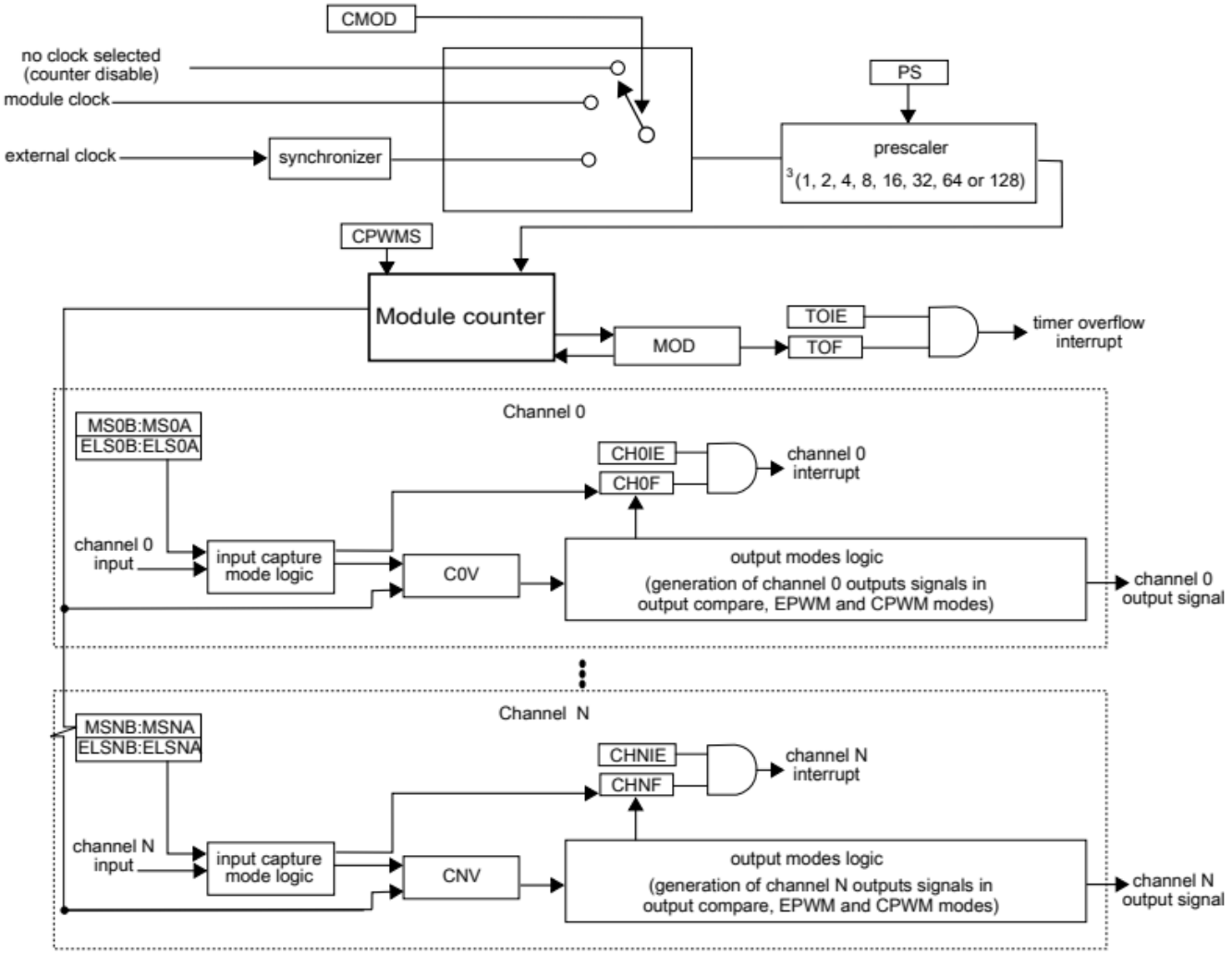
Lab Lecture 2.1 – Timer Programming

INTRODUCTION

Introduction

- The MCXC444 features three low power Timer/PWM Modules (TPMs).
  - TPM0, 1 and 2 can be clocked through various sources.
  - Each features a 16-bit counter that can count upwards or downwards.
  - Each TPM has 6 external channels that can function as:
    - Input capture – count number of times an input is triggered.
    - PWM – Phase correct (centre-aligned) or fast (edge-aligned) PWM.
    - Compare – Triggered when counter matches a value.

Introduction



Introduction

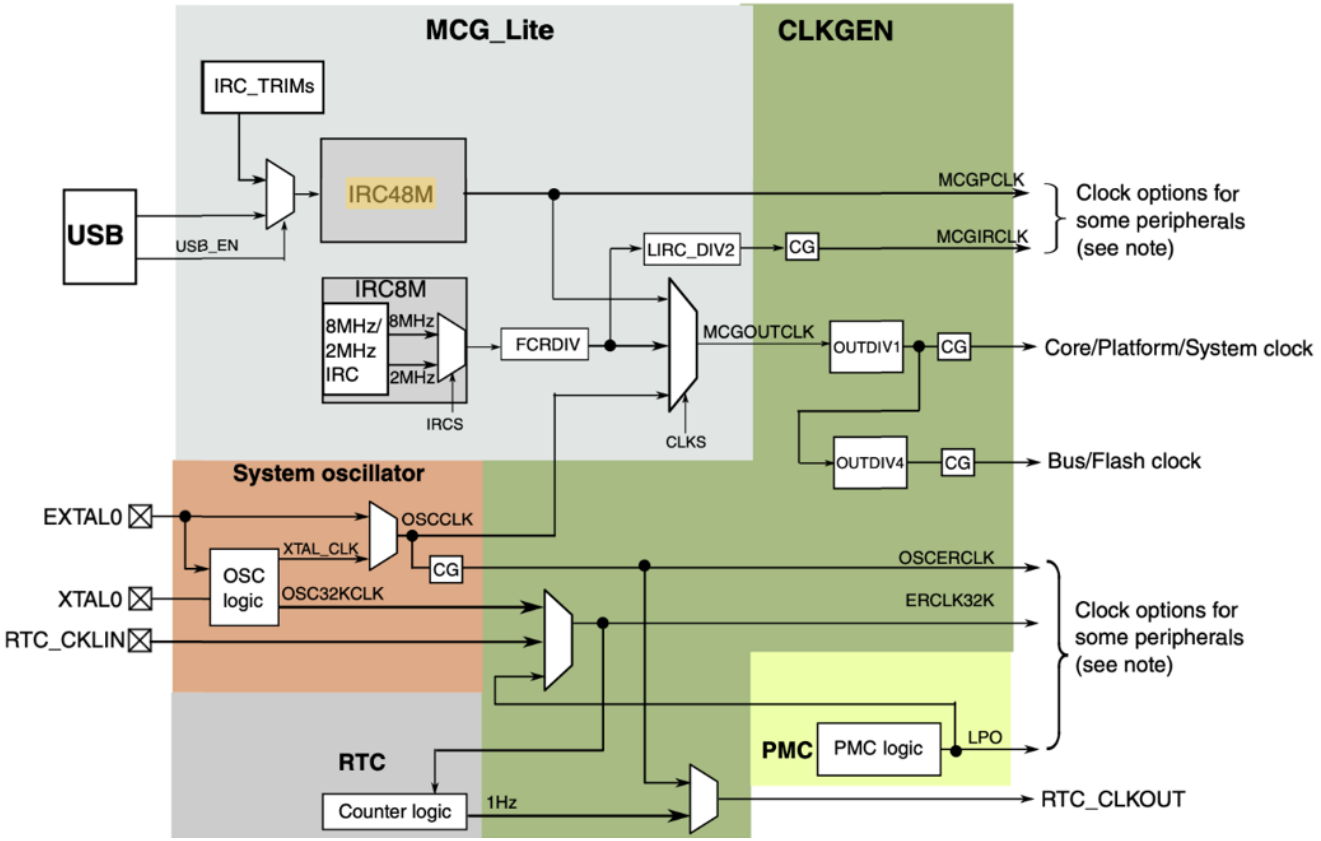
- The TPMs have many features but in this lecture we will only focus on generating interrupts at fixed intervals.
  - Useful for timing events (e.g. interval between a GPIO input being triggered), for creating a pre-emptive OS kernel, etc.
- In the next lecture we will look at using the TPMs to generate PWM signals.
- Once you understand the basics, you can reconfigure the TPMs to do other things.
  - E.g. generate waveforms when a value is matched.

Lab Lecture 2.1 – Timer Programming

CLOCK SOURCE SETUP

Clock Source Setup

- The TPMs can be driven by several internal and external clock sources.



Clock Source Setup

- We will use the Multipurpose Clock Generator Lite (MCG-Lite) to drive the TPMs.
  - Features a high frequency internal reference clock (HIRC) of 48 MHz and low frequency internal reference clock (LIRC) source of 8 MHz or 2 MHz.

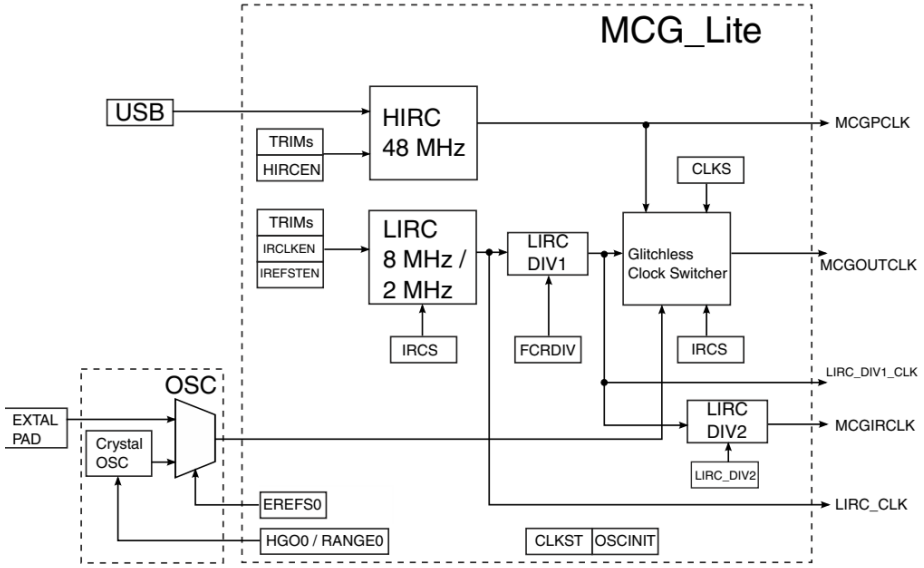


Figure 27-1. MCG_Lite block diagram

Choosing the MCG Clock Source

- The MCG Control Register 1 (MCG_C1) lets you choose the clock source in the CLKS bits:

Address: 4006_4000h base + 0h offset = 4006_4000h

Bit	7	6	5	4	3	2	1	0
Read	CLKS		0				IRCLKEN	IREFSTEN
Write								
Reset	0	1	0	0	0	0	0	0

Field	Description
7–6 CLKS	Clock Source Select Selects the clock source for MCGOUTCLK. 00 Selects HIRC clock as the main clock source. This is HIRC mode. 01 Selects LIRC clock as the main clock source. This is LIRC2M or LIRC8M mode. 10 Selects external clock as the main clock source. This is EXT mode. 11 Reserved. Writing 11 takes no effect.
5–2 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
1 IRCLKEN	Internal Reference Clock Enable Enables the IRC source. 0 LIRC is disabled. 1 LIRC is enabled.
0 IREFSTEN	Internal Reference Stop Enable Controls whether the IRC source remains enabled when the MCG_Lite enters Stop mode. 0 LIRC is disabled in Stop mode. 1 LIRC is enabled in Stop mode, if IRCLKEN is set.

Choosing the MCG Clock Source

- Which clock source you choose depends on the time interval you wish to program.
  - In our example we will generate an interrupt once per second.
  - Using a 48MHz clock will require a very large prescalar, so we will use the 2/8 MHz clock instead.
- We also need to set IRCLKEN to 1 to enable the low frequency internal reference clock.

Choosing the MCG Clock Source

- MCG lets you further divide the clock frequency two more times (circled) to generate the final output frequency at MCGIRCLK

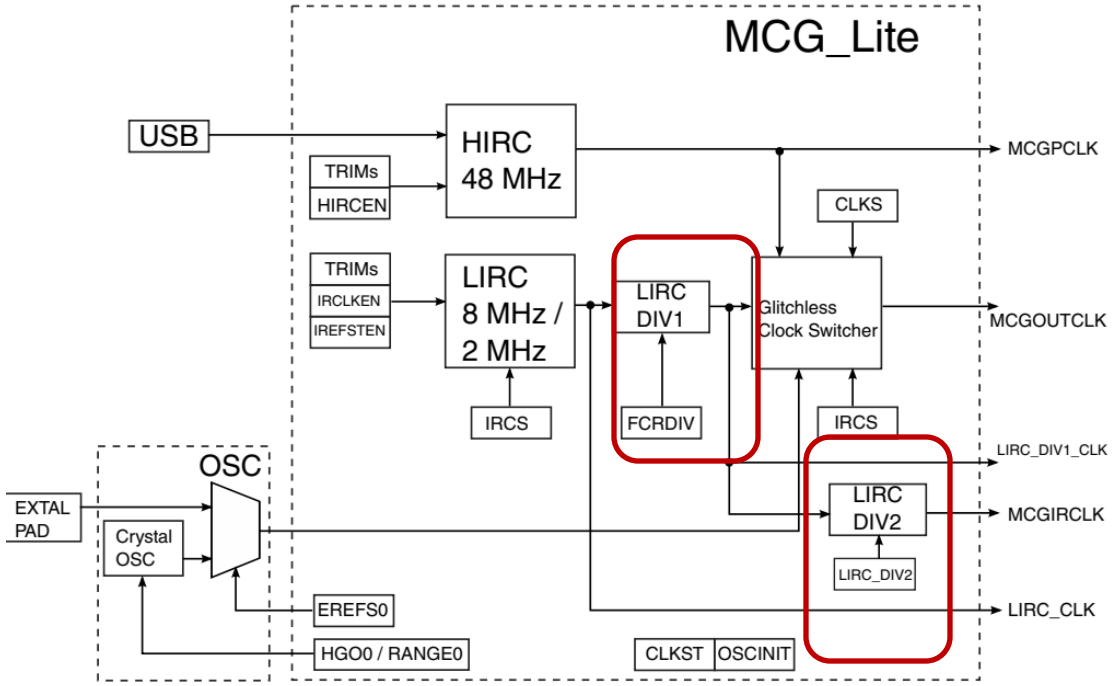


Figure 27-1. MCG_Lite block diagram

Choosing the MCG Clock Source

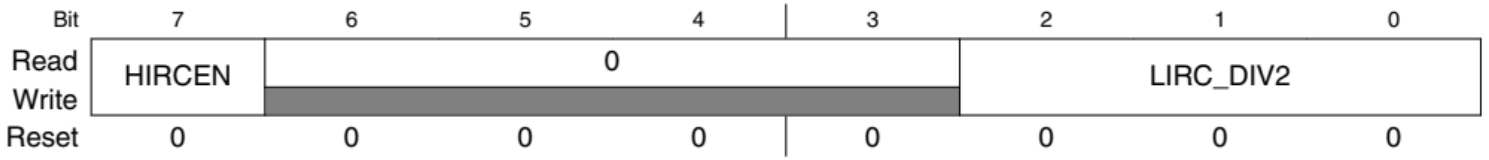
- We do this through the FRCDIV bits of the MCG Status and Control Register (MCG_SC) and LIRC_DIV2 bits of the MCG Miscellaneous Control Register (MCG_MC)

Address: 4006_4000h base + 8h offset = 4006_4008h

Bit	7	6	5	4	3	2	1	0
Read	0				FCRDIV			0
Write								
Reset	0	0	0	0	0	0	0	0

Field	Description
3–1 FCRDIV	Low-frequency Internal Reference Clock Divider Selects the factor value to divide the LIRC source. 000 Division factor is 1. 001 Division factor is 2. 010 Division factor is 4. 011 Division factor is 8. 100 Division factor is 16. 101 Division factor is 32. 110 Division factor is 64. 111 Division factor is 128.
0 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.

Choosing the MCG Clock Source



Field	Description
7 HIRCEN	High-frequency IRC Enable Enables the HIRC, even when MCG_Lite is not working at HIRC mode. 0 HIRC source is not enabled. 1 HIRC source is enabled.
6–3 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
LIRC_DIV2	Second Low-frequency Internal Reference Clock Divider Selects the factor value to further divide the LIRC source. 000 Division factor is 1. 001 Division factor is 2. 010 Division factor is 4. 011 Division factor is 8. 100 Division factor is 16. 101 Division factor is 32. 110 Division factor is 64. 111 Division factor is 128.

Choosing the MCG Clock Source

- If we want to keep the 8 MHz frequency, we choose divisors of 1 for FCRDIV and LIRC_DIV2:

```
// Set IRCS to 1 to choose 8 MHz clock
MCG->C2 |= MCG_C2_IRCS_MASK;

// Choose FCRDIV of 0 for divisor of 1
MCG->SC &= ~MCG_SC_FCRDIV_MASK;
MCG->SC |= MCG_SC_FCRDIV(0b0);

// Choose LIRC_DIV2 of 0 for divisor of 1
MCG->MC &= ~MCG_MC_LIRC_DIV2_MASK;
MCG->MC |= MCG_MC_LIRC_DIV2(0b0);
```

Clock Source Setup Full Code

```
// Configure the MCG Internal Reference Clock
void setMCGIRClk() {

   MCG->C1 &= ~MCG_C1_CLKS_MASK;
   // Choose MCG clock source of 01 for LIRC
   // and set IRCLKEN to 1 to enable LIRC
   MCG->C1 |= ((MCG_C1_CLKS(0b01) | MCG_C1_IRCLKEN_MASK));

   // Set IRCS to 1 to choose 8 MHz clock
   MCG->C2 |= MCG_C2_IRCS_MASK;

   // Choose FCRDIV of 0 for divisor of 1
   MCG->SC &= ~MCG_SC_FCRDIV_MASK;
   MCG->SC |= MCG_SC_FCRDIV(0b0);

   // Choose LIRC_DIV2 of 0 for divisor of 1
   MCG->MC &= ~MCG_MC_LIRC_DIV2_MASK;
   MCG->MC |= MCG_MC_LIRC_DIV2(0b0);

}
```

Lab Lecture 2.1 – Timer Programming

TPM SETUP

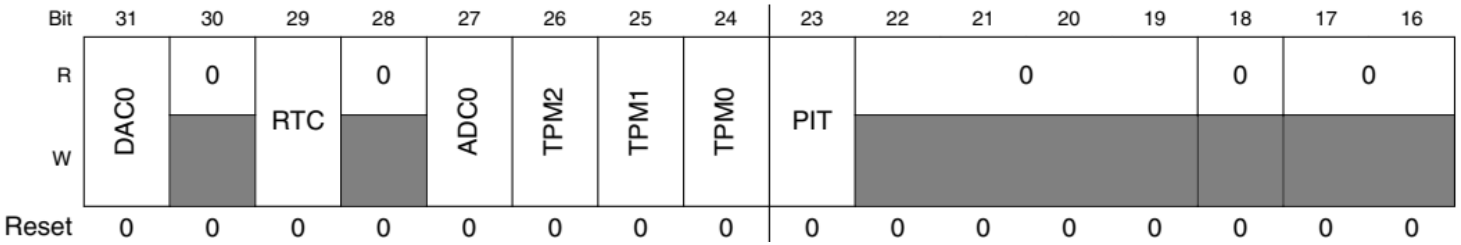
Preliminary setup

- It is always good practice to disable interrupts while setting up a timer.
  - We are going to use TPM1, so we disable TPM1_IRQn
  - We also call setMCGIRClk which sets up the MCG internal reference clock, as shown in the previous section:

```
// Disable TPM1 interrupt
NVIC_DisableIRQ( TPM1_IRQn );

// Initialize the MCG Internal Reference Clock
setMCGIRClk();
```

TPM Clock Gating

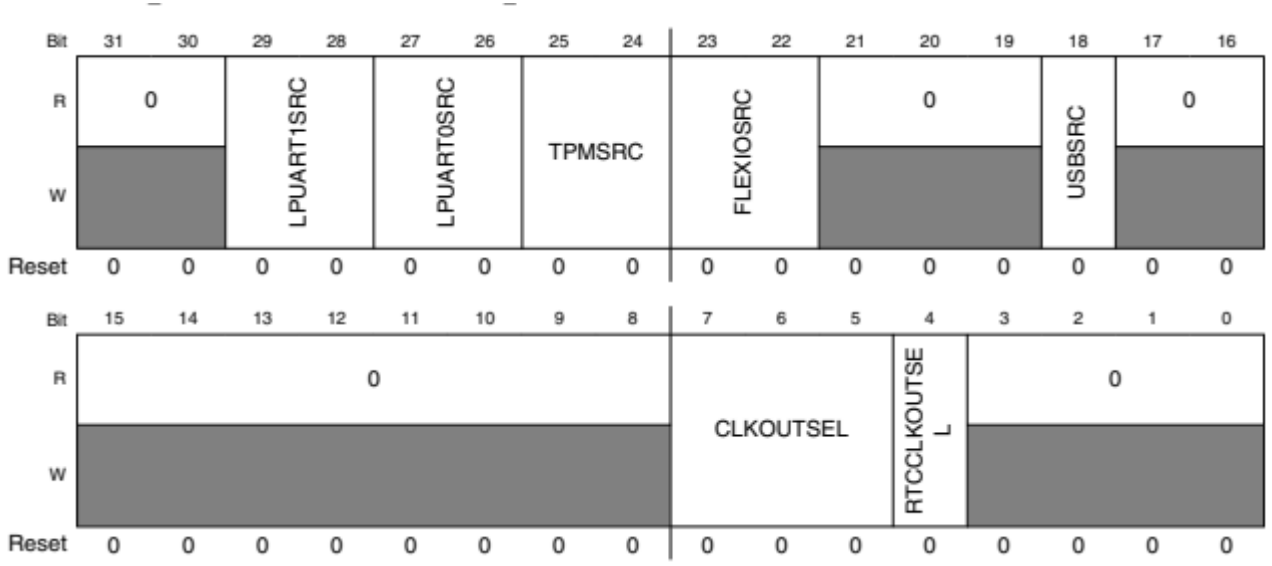
- We will now set up the TPM. We begin first by turning on the clocking to the correct TPM in SIM_SCGC6:
  - In our example we will use TPM1, so we set the TPM1 bit to 1.




```
// Turn on the clock gating
SIM->SCGC6 |= SIM_SCGC6_TPM1_MASK;
```

TPM Clock Selection

- We select which clock we want to use for the TPM through the TPMSRC bits in the System Options Register 2 (SIM_SOPT2).
  - SIM_SOPT2 also controls the clock selection for the USB, low power UART, etc.



TPM Clock Selection

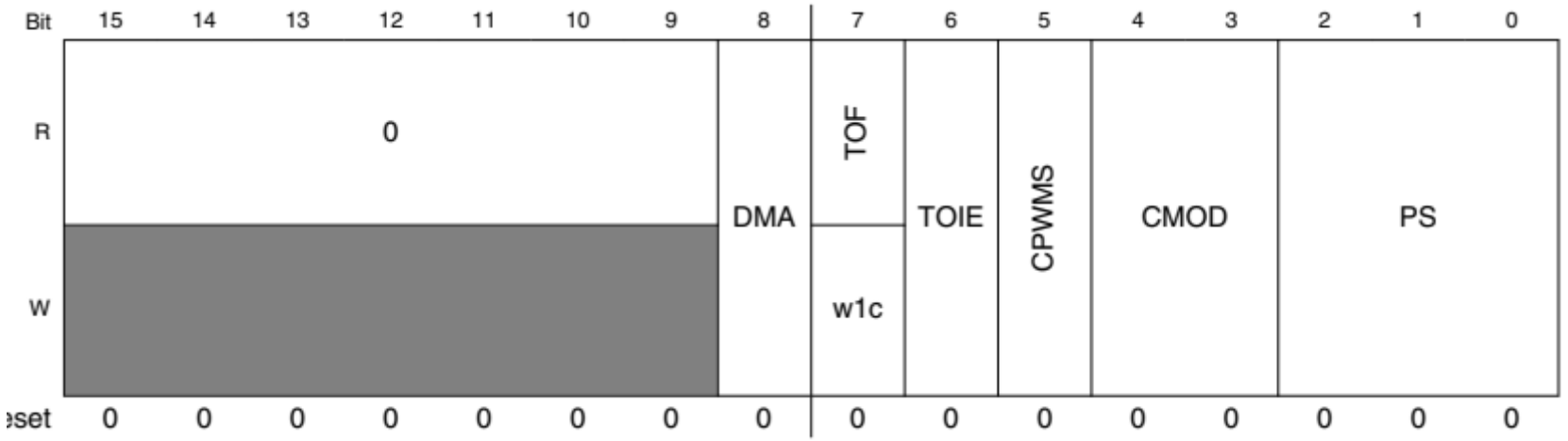
Field	Description
25–24 TPMSRC	TPM Clock Source Select Selects the clock source for the TPM counter clock 00 Clock disabled 01 IRC48M clock 10 OSCERCLK clock 11 MCGIRCLK clock

- Since we have set up the MCG Internal Reference Clock (MCGIRCLK), we will use that (0b11):

```
// Set clock source
SIM->SOPT2 &= ~SIM_SOPT2_TPMSRC_MASK;

// Use MCGIRCLK
SIM->SOPT2 |= SIM_SOPT2_TPMSRC(0b11);
```

TPM Core Setup

- We can now begin setting up the TPM itself.
  - We begin by turning off the TPM by setting CMOD (counter mode) to 0.
  - We clear the prescalar bits PS.
  - Both of these are in the TPM Status and Control Register TPMx_SC:



TPM Core Setup

Field	Description
31–9 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
8 DMA	DMA Enable Enables DMA transfers for the overflow flag. 0 Disables DMA transfers. 1 Enables DMA transfers.
7 TOF	Timer Overflow Flag Set by hardware when the TPM counter equals the value in the MOD register and increments. Writing a 1 to TOF clears it. Writing a 0 to TOF has no effect. If another TPM overflow occurs between the flag setting and the flag clearing, the write operation has no effect; therefore, TOF remains set indicating another overflow has occurred. In this case a TOF interrupt request is not lost due to a delay in clearing the previous TOF.

TPM Core Setup

Field	Description
	0 TPM counter has not overflowed. 1 TPM counter has overflowed.
6 TOIE	Timer Overflow Interrupt Enable Enables TPM overflow interrupts. 0 Disable TOF interrupts. Use software polling or DMA request. 1 Enable TOF interrupts. An interrupt is generated when TOF equals one.
5 CPWMS	Center-Aligned PWM Select Selects CPWM mode. This mode configures the TPM to operate in up-down counting mode. This field is write protected. It can be written only when the counter is disabled. 0 TPM counter operates in up counting mode. 1 TPM counter operates in up-down counting mode.
4–3 CMOD	Clock Mode Selection Selects the TPM counter clock modes. When disabling the counter, this field remain set until acknolwedged in the TPM clock domain. 00 TPM counter is disabled 01 TPM counter increments on every TPM counter clock 10 TPM counter increments on rising edge of TPM_EXTCLK synchronized to the TPM counter clock 11 Reserved.
PS	Prescale Factor Selection Selects one of 8 division factors for the clock mode selected by CMOD. This field is write protected. It can be written only when the counter is disabled. 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

TPM Core Setup

- We turn off the timer and clear the prescalar:

```
// Use MCGIRCLK
SIM->SOPT2 |= SIM_SOPT2_TPMSRC(0b11);

// Turn off TPM1 and clear the prescalar mask
TPM1->SC &= ~(TPM_SC_CMOD_MASK | TPM_SC_PS_MASK);
```

- The TPM maintains its count using the TPMx_CNT register, which we initialize to 0.

Address: Base address + 4h 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	0																COUNT															
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

```
// Initialize the count to 0
TPM1->CNT = 0;
```

TPM Prescalar and Modulo Selection

- We must now pick the prescalar *ps*; as with the ATMega328P, the timer counter (TPMx_CNT) is updated (incremented or decremented) every *ps* clock cycles.
  - **Note: ALWAYS CLEAR TPMx_CNT BEFORE SETTING TPMx_MOD.**
  - We selected the 8MHz LIRC, table on the next slide shows the interval (in ms) between increments to TPMx_CNT.
  - Formula used:

$$MOD = \frac{T}{P} = T \div \frac{PS}{F} = T \times \frac{F}{PS}$$

where T is the trigger interval, and P is the period of the counter.

TPM Prescalar and Modulo Selection

- If we wish to trigger an interval every T ms, and TPMx_CNT is updated every P, then:

  TPMx_MOD = T/P, rounded to the nearest integer.

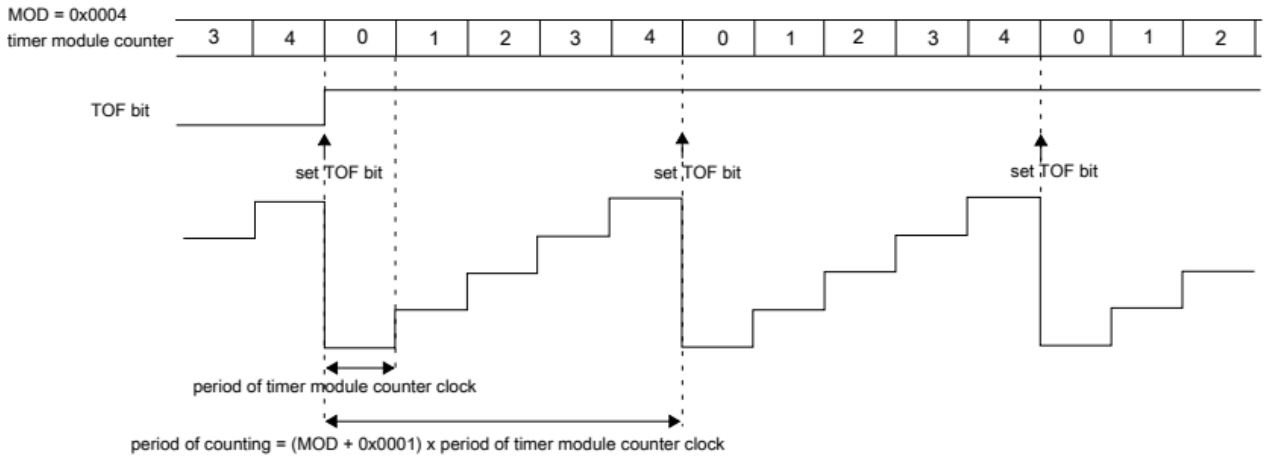
  - Where possible choose a prescalar so that TPMx_MOD is an integer.
  - Otherwise choose a prescalar where the rounding is as small as possible.

TPM Prescalar and Modulo Selection

Prescalar Value	TPMx_CNT Update Intervals (ms)
1	0.000125
2	0.00025
4	0.0005
8	0.001
16	0.002
32	0.004
64	0.008
128	0.016

TPM Prescalar and Modulo Selection

- The selection of the prescalar is related to the selection of the modulo (TPMx_MOD) value.
  - When TPMx_CNT reaches the value in TPMx_MOD, the timer overflow flag (TOF) is set, and if the timer overflow interrupt is enabled (through the TOIE bit in TPMx_SC). TPMx_CNT is cleared if in up-counting mode.



TPM Prescalar and Modulo Selection

- In our example we will choose a prescalar of 128.
  - We wish to trigger an interrupt every second = every 1000 ms.
  - Thus $P = \frac{128}{8000} = 0.016\,ms$, and TPM1_MOD = $\frac{1000}{0.016} = 62500$

TPM Prescalar and Modulo Selection

- So:
  - We set PS in TPMx_SC to 7.
  - We also set the TOIE bit to 1.
  - We set the modulo to 62500

```
// Set a prescalar of 128, and the TOIE mask
TPM1->SC |= ((TPM_SC_TOIE_MASK) | TPM_SC_PS(0x7));

// Initialize modulo.
// For a PS of 128, time is 128/8000 ms = 0.016 ms
// For 1 second = 1000 ms, MOD = 1000/0.016 = 62500
TPM1->MOD = 62500;
```

Setting Interrupt Priority and Enabling

- Finally we set the interrupt priority and enable the TPM_IRQn interrupt.

```
// Set priority to high
NVIC_SetPriority( TPM1_IRQn, 0);

NVIC_EnableIRQ( TPM1_IRQn);
```

- Full TPM setup code is on the next slide.

Full TPM Setup Code

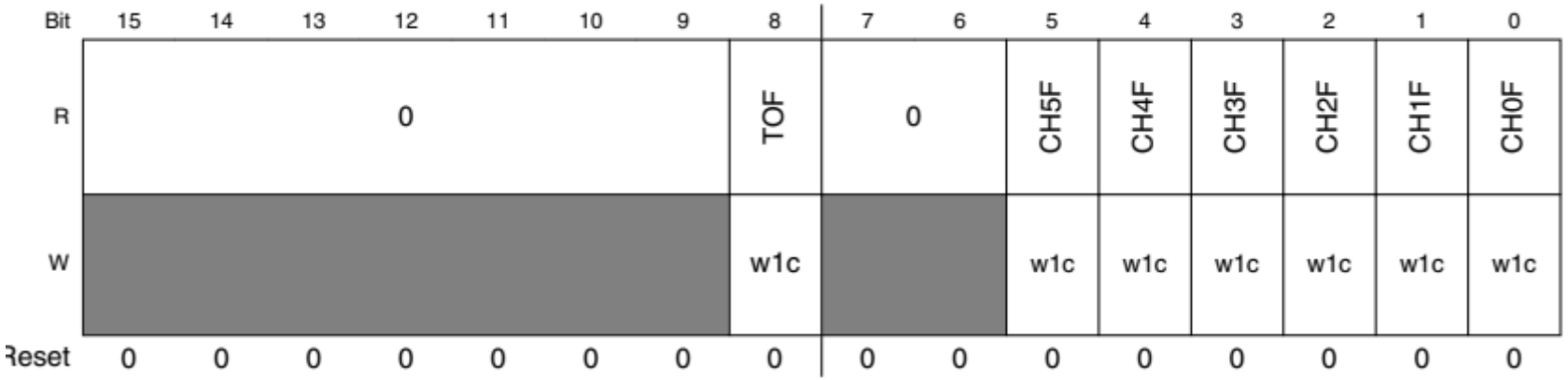
```
void initTimer() {

  // Disable TPM1 interrupt
  NVIC_DisableIRQ( TPM1_IRQn );

  // Initialize the MCG Internal Reference Clock
  setMCGIRClk();

  // Turn on the clock gating
  SIM->SCGC6 |= SIM_SCGC6_TPM1_MASK;

  // Set clock source
  SIM->SOPT2 &= ~SIM_SOPT2_TPMSRC_MASK;

  // Use MCGIRCLK
  SIM->SOPT2 |= SIM_SOPT2_TPMSRC(0b11);

  // Turn off TPM1 and clear the prescalar mask
  TPM1->SC &= ~(TPM_SC_CMOD_MASK | TPM_SC_PS_MASK);

  // Initialize the count to 0
  TPM1->CNT = 0;

  // Set a prescalar of 128, and the TOIE mask
  TPM1->SC |= ((TPM_SC_TOIE_MASK) | TPM_SC_PS(0x7));

  // Initialize modulo.
  // For a PS of 128, time is 128/8000 ms = 0.016 ms
  // For 1 second = 1000 ms, MOD = 1000/0.016 = 62500
  TPM1->MOD = 62500;

  // Set priority to high
  NVIC_SetPriority( TPM1_IRQn, 0);
  NVIC_EnableIRQ( TPM1_IRQn );

}
```

Writing the ISR

- The various modules in the MCXC444 are each given only a single ISR vector.
  - In the case of TPMx, it will be TPMx_IRQn (i.e. TPM0_IRQn, TPM1_IRQn, TPM2_IRQn).
  - The corresponding ISRs are called TPMx_IRQHandler.
- This means that an ISR can be triggered for various reasons.
  - For TPMx, we need to check the TPMx_STATUS register for the reason.

Writing the ISR

- We are only interested in the TOF (timer overflow flag) in TPMx_STATUS:



Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0							TOF	0		CH5F	CH4F	CH3F	CH2F	CH1F	CH0F
W								w1c			w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

  - If TOF=1, then the timer has overflowed and the period we want has passed.
  - Write a 1 to TOF to clear it.

Writing the ISR

```
void TPM1_IRQHandler(){
  NVIC_ClearPendingIRQ( TPM1_IRQn);

  if(TPM1->STATUS & TPM_STATUS_TOF_MASK) {

    // ... Perform some action ...

    // Clear the TOF flag
      TPM1->STATUS |= TPM_STATUS_TOF_MASK;
    }
}
```

Starting the Timer

- We finally start the timer by setting the CMOD bits in TPMx_SC to 0b01:

```
// Use TPM counter clock to increment.
TPM1->SC |= TPM_SC_CMOD(0b1);
```

	1 TPM counter operates in up-down counting mode.
4–3 CMOD	Clock Mode Selection Selects the TPM counter clock modes. When disabling the counter, this field remain set until acknolwedged in the TPM clock domain. 00 TPM counter is disabled 01 TPM counter increments on every TPM counter clock 10 TPM counter increments on rising edge of TPM_EXTCLK synchronized to the TPM counter clock 11 Reserved.
PS	Prescale Factor Selection

Conclusion

- This lecture shows how to set up the Timer/PWM Module (TPM) to trigger an interrupt at fixed intervals.
- In the next lecture we will see how to use the TPM to generate PWM signals.

CG2271 Real Time Operating Systems

Lab Lecture 2.2

PWM Programming

(Reference Guide Chapter 29)

Lab Lecture 2.2 – PWM Programming

INTRODUCTION

Introduction

- The three Timer/PWM Modules (TPMs) produce a total of 10 PWM outputs (See next slide)
  - PWM signals can either be edge-aligned or centre aligned.
  - Equivalent to the ATMega328P's fast PWM and phase-correct PWM respectively.
- Most of what we need to know about the TPMs was already covered in Lab Lecture 2.1.
  - Please view that lecture first if you've not already done so.

PWM Pin Outputs

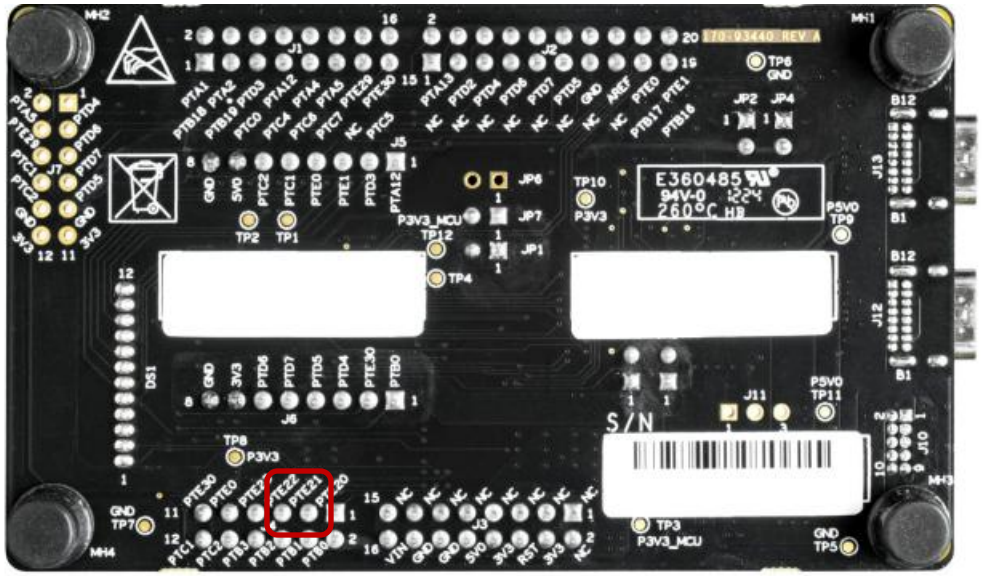
Channel Name	Pin Name	MUX ALT Value
TPM0-CH0	PTE24/PTA3	3
TPM0-CH1	PTE25/PTA4	3
TPM0-CH2	PTE29/PTA5	3
TPM0-CH3	PTE30	3
TPM0-CH4	PTE31/PTD4	3/4
TPM0-CH5	PTA0/PTD5	3/4
TPM1-CH0	PTE20/PTA12/PTB0	3
TPM1-CH1	PTE21/PTA13/PTB1	3
TPM1-CH2	-	=
TPM1-CH3	-	-
TPM1-CH4	-	-
TPM1-CH5	-	-
TPM2-CH0	PTE22PTA1/PTB2/PTB18	3
TPM2-CH1	PTE23/PTA2/PTB3/PTB19	3
TPM2-CH2	-	-
TPM2-CH3	-	-
TPM2-CH4	-	-
TPM2-CH5	-	-

Lab Lecture 2.2 – PWM Programming

SETTING UP THE PWM

Basic TPM Setup

- We will now look at how to drive PTE21.
  - This is available on the FRDM-MCXC444 board header.
    - Corresponds to TPM1_CH1



Basic TPM Setup

- Steps from Lab Lecture 4:
  - 1. Set up Multipurpose Clock Generator (MCG) to produce an 8 MHz clock rate.
  - 2. Set up the TPM clock source in SIM_SOPT2.
  - 3. Clear the PS bits.
  - 4. Set the PS bits to 0x7 (128) and the TOIE bit if necessary.
  - 5. Clear TPMx_CNT.
  - 6. Set TPMx_MOD to determine PWM frequency. (see next page)

PWM Frequency

- To simplify matters we will set a prescalar of 128.
- We use the TPMx_MOD modulo to set the PWM frequency:
  - What happens when the frequency is too low/high?
- We will not worry too much about switching losses, so we choose a PWM frequency of 250 Hz.
  - This will give us very nice smooth operations.

PWM Frequency

- Settings:
  - TPM Clock Rate: 8 MHz (from MCG)
  - Prescalar Setting: 128
  - TPMx_CNT update period (ms): $\frac{128}{8000} = 0.016$
  - PWM period (ms) = $\frac{1}{250} = 4$
- Note that we are using centre aligned PWM (see later), which gives us half the frequency.
  - Recall that $MOD = T \times \frac{F}{PS} = \frac{T}{P}$

    PS is prescalar, F is clock frequency & T is PWM Period.
  - So TPMx_MOD = $4 \div 0.016 \div 2 = 125$

Full Basic Setup Code

- Set up MCG and TPM Clock Source
  - The suggested template code combines `setTPMClock()` into `initTimer()`, but we keep it separate here to keep it self-contained.

```
// Configure the MCG Internal Reference Clock
void setMCGIRClk() {

  MCG->C1 &= ~MCG_C1_CLKS_MASK;
  // Choose MCG clock source of 01 for LIRC
  // and set IRCLKEN to 1 to enable LIRC
  MCG->C1 |= ((MCG_C1_CLKS(0b01) | MCG_C1_IRCLKEN_MASK));

  // Set IRCS to 1 to choose 8 MHz clock
  MCG->C2 |= MCG_C2_IRCS_MASK;

  // Choose FCRDIV of 0 for divisor of 1
  MCG->SC &= ~MCG_SC_FCRDIV_MASK;
  MCG->SC |= MCG_SC_FCRDIV(0b0);

  // Choose LIRC_DIV2 of 0 for divisor of 1
  MCG->MC &= ~MCG_MC_LIRC_DIV2_MASK;
  MCG->MC |= MCG_MC_LIRC_DIV2(0b0);

}
```

```
void setTPMClock(){

  // Set MCGIRCLK
  setMCGIRClk();

  // Choose MCGIRCLK (8 MHz)
  SIM->SOPT2 &= ~SIM_SOPT2_TPMSRC_MASK;
  SIM->SOPT2 |= SIM_SOPT2_TPMSRC(0b11);

}
```

Full Basic Setup Code

- Set up TPM1 (since PTE21 is connected to it)

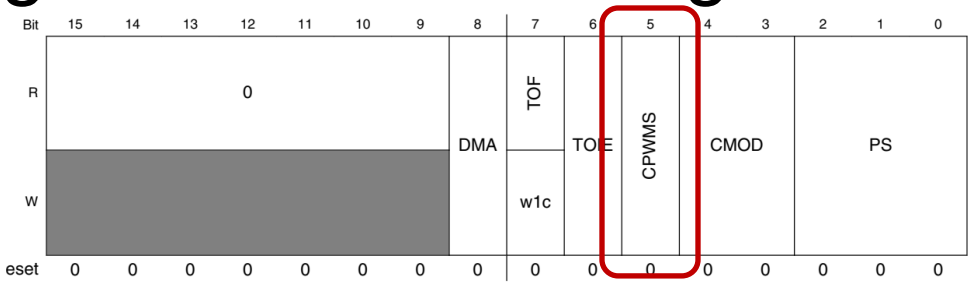
```
// Turn on clock gating to TPM1
SIM->SCGC6 |= SIM_SCGC6_TPM1_MASK;
// Set up TPM1
// Turn off TPM1 and clear the prescalar field
TPM1->SC &= ~(TPM_SC_CMOD_MASK | TPM_SC_PS_MASK);

// Set prescalar of 128
TPM1->SC |= TPM_SC_PS(0b111);

// Select centre-aligned PWM mode
TPM1->SC |= TPM_SC_CPWMS_MASK;

// Initialize count to 0
TPM1->CNT = 0;

// We nominally choose a PWM frequency of 250 Hz
// TPM period = 128 / 8000 =0.016 ms
// Period for 250Hz = 4 ms
// For 250Hz, our mod value = 4 / (0.016 * 2) = 125
// Note we must x2 in the denominator because the output
// frequency is 1/2 of the normal frequency in centre-aligned PWM
TPM1->MOD=125;
```

Full Basic Setup Code

- Notice there is one extra line in the setup code:

```
// Select centre-aligned PWM mode
TPM1->SC |= TPM_SC_CPWMS_MASK;
```

- This sets up the CPWMS bit in the TPM status and control register to choose between edge-aligned and centre-aligned PWM.



5 CPWMS	Center-Aligned PWM Select Selects CPWM mode. This mode configures the TPM to operate in up-down counting mode. This field is write protected. It can be written only when the counter is disabled. 0 TPM counter operates in up counting mode. 1 TPM counter operates in up-down counting mode.

Setting Up the PWM Outputs

- We can now set up the PWM on PTE21, which is ALT3.

```
// Turn on clock gating to Port E
// Set up the clock gating
SIM->SCGC5 |=  SIM_SCGC5_PORTE_MASK;


// Set the pin multiplexors
PORTE->PCR[PWM_PIN] &= ~PORT_PCR_MUX_MASK;

// PWM
PORTE->PCR[PWM_PIN] |= PORT_PCR_MUX(0b11);

// Set pins to output
GPIOE->PDDR |= (1 << PWM_PIN);
```

  - Not strictly necessary to set the respective pins as output, but we do so for clarity:

Setting Up the PWM Outputs

- Finally we configure the PWM channel"
  - Table below shows the channel number for PTE21:

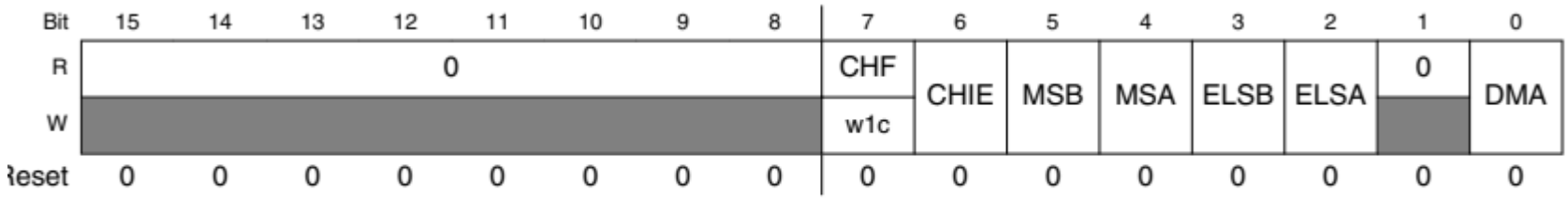
Channel Name	Pin Name	MUX ALT Value
TPM0-CH0	PTE24/PTA3	3
TPM0-CH1	PTE25/PTA4	3
TPM0-CH2	PTE29/PTA5	3
TPM0-CH3	PTE30	3
TPM0-CH4	PTE31/PTD4	3/4
TPM0-CH5	PTA0/PTD5	3/4
TPM1-CH0	PTE20/PTA12/PTB0	3
TPM1-CH1	PTE21/PTA13/PTB1	3
TPM1-CH2	-	=
TPM1-CH3	-	-
TPM1-CH4	-	-
TPM1-CH5	-	-
TPM2-CH0	PTE22PTA1/PTB2/PTB18	3
TPM2-CH1	PTE23/PTA2/PTB3/PTB19	3
TPM2-CH2	-	-
TPM2-CH3	-	-
TPM2-CH4	-	-
TPM2-CH5	-	-

Setting Up the PWM Outputs

- From the table we see we need to set up TPM1 Channel 1.
  - We configure through the channel status and control register1 TPM1_C1SC
  - Programmatically this is accessible via:
    - TPM1->CONTROLS[1].CnSC
  - PWM Duty Cycle is set using the TPM1_C1V, Corresponds to TPM1->CONTROL[1].CnV

Setting Up the PWM Outputs

- We set the CPWMS bit in TPM1_SC to 1, using centre-aligned PWM.
- We can refer to the Section 29.4.4 to see how to configure the PWM output in the TPMx_CnSC register:
  - MSnB:MSnA bits set the pin mode.
  - ELSnB:ELSnA bits set the pin polarity.



Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0								CHF	CHIE	MSB	MSA	ELSB	ELSA	0	DMA
W									w1c							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TPMx_C*n*SC field descriptions

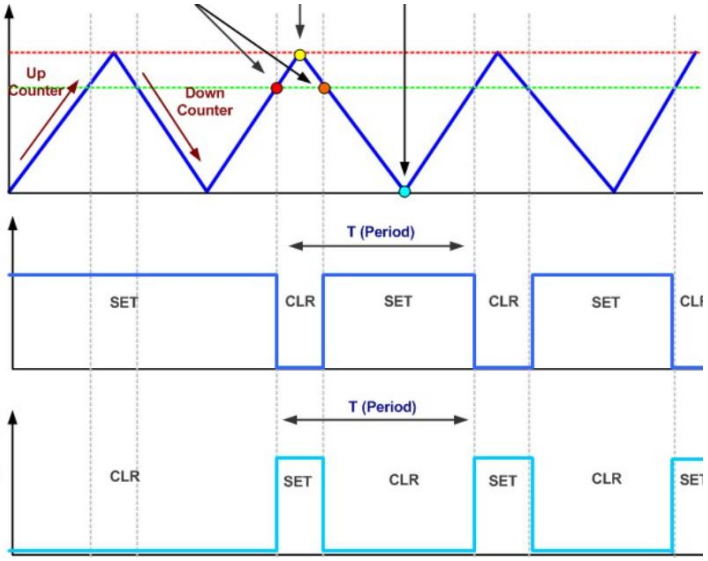
Setting Up the PWM Outputs

CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
X	00	00	None	Channel disabled
X	01	00	Software compare	Pin not used for TPM
0	00	01	Input capture	Capture on Rising Edge Only
		10		Capture on Falling Edge Only
		11		Capture on Rising or Falling Edge
	01	01	Output compare	Toggle Output on match
		10		Clear Output on match
		11		Set Output on match
	10	10	Edge-aligned PWM	High-true pulses (clear Output on match, set Output on reload)
		X1		Low-true pulses (set Output on match, clear Output on reload)
	11	10	Output compare	Pulse Output low on match
		01		Pulse Output high on match
1	10	10	Center-aligned PWM	High-true pulses (clear Output on match-up, set Output on match-down)
		01		Low-true pulses (set Output on match-up, clear Output on match-down)

- With CPWMS=1, MSnB:MSnA = 0b10 sets centre PWM output.
- Setting ELSnB:ELSnA controls the polarity of the PWM signal.

Choosing PWM Polarity

- ELSnB:ELSnA = 0b10
  - High True: Clear pin matching CnV counting up, set pin matching CnV counting down.
- ELSnB: ELSnA = 0b01
  - Low True: Set pin matching CnV counting up, clear pin matching CnV counting down.



ELSnB:ELSnA = 0b10

ELSnB:ELSnA = 0b01

Choosing PWM Polarity

- We are going to generate a conventional PWM signal (High-True Pulse), so:
  - We clear MSB:MSA and ELSB:ELSA as usual using inversions of the masks.
  - We set MSB=1 to give MS=10, and ELSB to 1 to give ELS=10.

Choosing PWM Polarity

```
// Configure channel 1
// MS=10, ELS=10.
// So set MSB=1, ELSB=1
// MSA=0 and ELSA=0 because of our prior clearing.
// Note that this configures a REVERSE PWM signal
// I.e. it sets when counting up and clears when counting
// down. This is because the LEDs are active low.

TPM1->CONTROLS[1].CnSC &= ~(TPM_CnSC_MSA_MASK | TPM_CnSC_ELSA_MASK);
TPM1->CONTROLS[1].CnSC |= (TPM_CnSC_MSB(1) | TPM_CnSC_ELSB(1));
```

Set the PWM Value and Turning On

- We set the PWM value for a channel n by writing to TPM1->CONTROL[1]->CnV

```
void setPWM(int percent) {

   int value = (int)((percent / 100.0) * (double) TPM1->MOD);
   TPM1->CONTROLS[1].CnV=value;
}
```

- As before with the timer, we turn on the PWM signal by writing a 1 to the CMOD bit in TPM0_SC:

```
void startPWM() {
   TPM1->SC |= TPM_SC_CMOD(0b01);
}
```

Conclusion

- In Lab Lecture 2.1 we looked at the TPM module and learnt how to set up a timer.
- In this lecture we extended the idea to generate PWM signals.
- In the next lecture we will look at how to use the Analog-Digital Converter (ADC) on the MCXC444.

CG2271 Real Time Operating Systems

Lab Lecture 2.3

Analog – Digital programming

(Reference Guide Chapter 23)

Lab Lecture 2.3 – Analog to Digital Programming

INTRODUCTION

Introduction

- In Lab Lecture 2.2 we saw how to generate “analog” output using the Timer/PWM Modules (TPMs).
- In this lecture we will look at the Analog-Digital Converter (ADC) on the MCXC444.
  - We will read the visible light sensor on the FRDM-MCXC444 board and use this to control the brightness of the RGB LED.
  - Please see Lab Lectures 4 and 5 on how to configure the TPMs to generate PWM to control the LEDs.

ADC Features on the MCXC444

- Successive Approximation.
- Up to 24 channels, with up to 16 bits of resolution per channel.
  - Analog inputs 0 to 3 can operate in “differential mode”.
    - Measures voltages relative between two pins, rather than relative to ground.
    - Provides more accurate measurements
    - Using differential mode reduces the total number of ADC channels available.
  - Other inputs operate in single ended mode.

ADC Features on the MCXC444

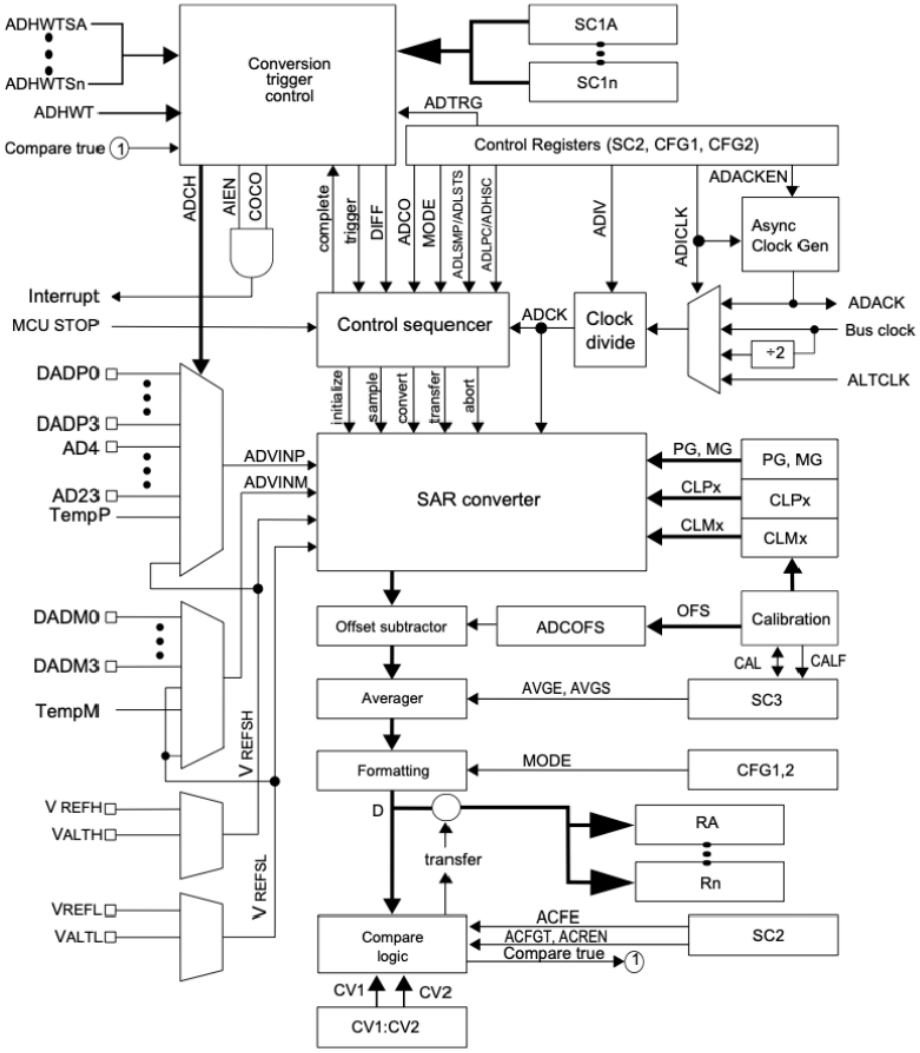
- The ADC input pins are spread around the MCXC444 chip – Note that not all 24 inputs are available (e.g. ADC0_SE10 does not exist)

Pin	ADC Function	ALT
PTE20	ADC0_DP0 / ADC0_SE0	ALT0
PTE21	ADC0_DM0 / ADC0_SE4a	ALT0
PTE22	ADC0_DP3 / ADC0_SE3	ALT0
PTE23	ADC0_DM3 / ADC0_SE7a	ALT0
PTE29	ADC0_SE4b	ALT0
PTE30	ADC0_SE23	ALT0
PTB0	ADC0_SE8	ALT0
PTB1	ADC0_SE9	ALT0
PTB2	ADC0_SE12	ALT0
PTB3	ADC0_SE13	ALT0
PTC0	ADC0_SE14	ALT0
PTC1	ADC0_SE15	ALT0
PTC2	ADC0_SE11	ALT0
PTD1	ADC0_SE5b	ALT0
PTD5	ADC0_SE6b	ALT0
PTD6	ADC0_SE7b	ALT0

ADC Features on the MCXC444

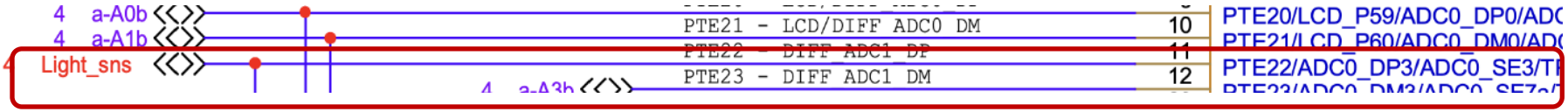
- Note the existence of “a” and “b” versions of some pins, e.g. ADC0_SE7a and ADC0_SE7b.
  - “b” pins are controlled using the ADC0_SC1b status and control register (see later), while all other pins are controlled using ADC0_SC1a.
  - ADC conversion on the “b” pins can only be triggered by internal hardware.
    - I.e. We cannot start conversion on the “b” pins through programming.
    - Conversion on the ”a” and other pins can be triggered by both internal hardware and software.

ADC Features on the MCXC444



Configuring the ADC Module

- The FRDM-MCXC444 board has a light sensor connected to ADC0_SE3 on PTE22.
  - Our examples will assume we are using this pin.



ADC Clock Gating

- Clock gating is controlled by SIM_SCGC6; we need to turn this on:

```
// Enable clock gating to ADC0
SIM->SCGC6 |= SIM_SCGC6_ADC0_MASK;
```

- We are accessing PTE22, so we turn on the clock to PORTE on SIM_SCGC5

```
// Enable clock gating to PTE
// This is done when we initialize the PWM
// but we want to make our function self-contained
// so we do it again
SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;
```

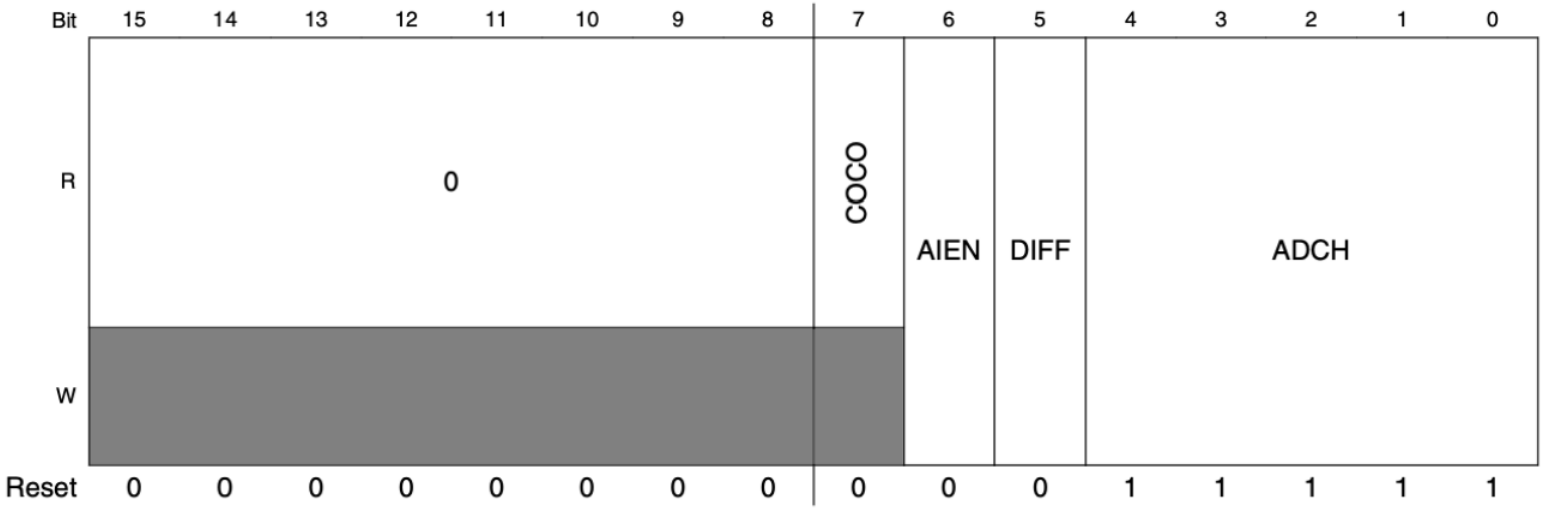
ADC Pin Configuration

- Select the correct ALT value (ALT0) for the ADC pin you want to use (PTE22 in our example)

```
#define ADC_PIN     22  // ADC0_SE3 is on PTE22

// Set PTE22 (ADC0_DP3) to ADC
PORTE->PCR[ADC_PIN] &= ~PORT_PCR_MUX_MASK;
PORTE->PCR[ADC_PIN] |= PORT_PCR_MUX(0);
```

ADC0_SC1a

- We now need to configure the ADC first. We begin with Status and Control Register 1.
  - ADC0_SE3 is not a “b” pin, so we will configure ADC0_SC1a. In CMSIS it is accessible via: ADC0->SC1[0].



ADC0_SC1a

ADCx_SC1*n* field descriptions

Field	Description
31–8 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
7 COCO	Conversion Complete Flag This is a read-only field that is set each time a conversion is completed when the compare function is disabled, or SC2[ACFE]=0 and the hardware average function is disabled, or SC3[AVGE]=0. When the compare function is enabled, or SC2[ACFE]=1, COCO is set upon completion of a conversion only if the compare result is true. When the hardware average function is enabled, or SC3[AVGE]=1, COCO is set upon completion of the selected number of conversions (determined by AVGS). COCO in SC1A is also set at the completion of a calibration sequence. COCO is cleared when the respective SC1n register is written or when the respective Rn register is read. 0 Conversion is not completed. 1 Conversion is completed.
6 AIEN	Interrupt Enable Enables conversion complete interrupts. When COCO becomes set while the respective AIEN is high, an interrupt is asserted. 0 Conversion complete interrupt is disabled. 1 Conversion complete interrupt is enabled.
5 DIFF	Differential Mode Enable Configures the ADC to operate in differential mode. When enabled, this mode automatically selects from the differential channels, and changes the conversion algorithm and the number of cycles to complete a conversion. 0 Single-ended conversions and input channels are selected. 1 Differential conversions and input channels are selected.
ADCH	Input channel select Selects one of the input channels. The input channel decode depends on the value of DIFF. DAD0-DAD3 are associated with the input pin pairs DADPx and DADMx. **NOTE:** Some of the input channel options in the bitfield-setting descriptions might not be available for your device. For the actual ADC channel assignments for your device, see the Chip Configuration details.

Table continues on the next page...

ADC0_SC1a

Field	Description
	The successive approximation converter subsystem is turned off when the channel select bits are all set, that is, ADCH = 11111. This feature allows explicit disabling of the ADC and isolation of the input channel from all sources. Terminating continuous conversions this way prevents an additional single conversion from being performed. It is not necessary to set ADCH to all 1s to place the ADC in a low-power state when continuous conversions are not enabled because the module automatically enters a low-power state when a conversion completes. 00000 When DIFF=0, DADP0 is selected as input; when DIFF=1, DAD0 is selected as input. 00001 When DIFF=0, DADP1 is selected as input; when DIFF=1, DAD1 is selected as input. 00010 When DIFF=0, DADP2 is selected as input; when DIFF=1, DAD2 is selected as input. 00011 When DIFF=0, DADP3 is selected as input; when DIFF=1, DAD3 is selected as input. 00100 When DIFF=0, AD4 is selected as input; when DIFF=1, it is reserved. 00101 When DIFF=0, AD5 is selected as input; when DIFF=1, it is reserved. 00110 When DIFF=0, AD6 is selected as input; when DIFF=1, it is reserved. 00111 When DIFF=0, AD7 is selected as input; when DIFF=1, it is reserved. 01000 When DIFF=0, AD8 is selected as input; when DIFF=1, it is reserved. 01001 When DIFF=0, AD9 is selected as input; when DIFF=1, it is reserved. 01010 When DIFF=0, AD10 is selected as input; when DIFF=1, it is reserved. 01011 When DIFF=0, AD11 is selected as input; when DIFF=1, it is reserved. 01100 When DIFF=0, AD12 is selected as input; when DIFF=1, it is reserved. 01101 When DIFF=0, AD13 is selected as input; when DIFF=1, it is reserved. 01110 When DIFF=0, AD14 is selected as input; when DIFF=1, it is reserved. 01111 When DIFF=0, AD15 is selected as input; when DIFF=1, it is reserved. 10000 When DIFF=0, AD16 is selected as input; when DIFF=1, it is reserved. 10001 When DIFF=0, AD17 is selected as input; when DIFF=1, it is reserved. 10010 When DIFF=0, AD18 is selected as input; when DIFF=1, it is reserved. 10011 When DIFF=0, AD19 is selected as input; when DIFF=1, it is reserved. 10100 When DIFF=0, AD20 is selected as input; when DIFF=1, it is reserved. 10101 When DIFF=0, AD21 is selected as input; when DIFF=1, it is reserved. 10110 When DIFF=0, AD22 is selected as input; when DIFF=1, it is reserved. 10111 When DIFF=0, AD23 is selected as input; when DIFF=1, it is reserved. 11000 Reserved. 11001 Reserved. 11010 When DIFF=0, Temp Sensor (single-ended) is selected as input; when DIFF=1, Temp Sensor (differential) is selected as input. 11011 When DIFF=0,Bandgap (single-ended) is selected as input; when DIFF=1, Bandgap (differential) is selected as input. 11100 Reserved. 11101 When DIFF=0,V_{REFSH} is selected as input; when DIFF=1, -V_{REFSH} (differential) is selected as input. Voltage reference selected is determined by SC2[REFSEL]. 11110 When DIFF=0,V_{REFSL} is selected as input; when DIFF=1, it is reserved. Voltage reference selected is determined by SC2[REFSEL]. 11111 Module is disabled.

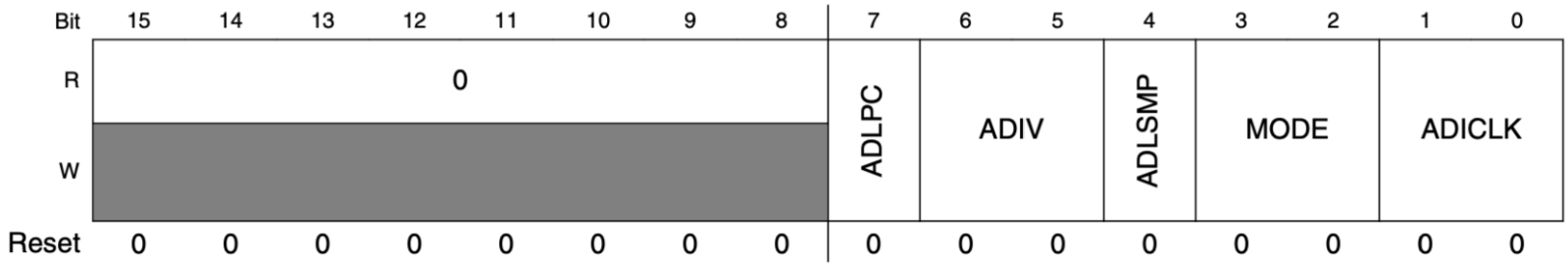
ADC0_SC1a

- We will choose the ADC channel later; for now, we want to enable interrupts, and disable differential mode by setting AIEN to 1 and DIFF to 0:

```
// Configure the ADC
// Enable ADC interrupt
ADC0->SC1[0] |= ADC_SC1_AIEN_MASK;

// Select single-ended ADC
ADC0->SC1[0] &= ~ADC_SC1_DIFF_MASK;
ADC0->SC1[0] |= ADC_SC1_DIFF(0b0);
```

ADC0_CFG1

- We will now look at the ADC Configuration Register 1, which lets us choose the resolution, low power/normal power mode, long sample mode, etc.



ADC0_CFG1

ADCx_CFG1 field descriptions

Field	Description
31–8 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
7 ADLPC	Low-Power Configuration Controls the power configuration of the successive approximation converter. This optimizes power consumption when higher sample rates are not required. 0 Normal power configuration. 1 Low-power configuration. The power is reduced at the expense of maximum clock speed.
6–5 ADIV	Clock Divide Select Selects the divide ratio used by the ADC to generate the internal clock ADCK. 00 The divide ratio is 1 and the clock rate is input clock. 01 The divide ratio is 2 and the clock rate is (input clock)/2. 10 The divide ratio is 4 and the clock rate is (input clock)/4. 11 The divide ratio is 8 and the clock rate is (input clock)/8.
4 ADLSMP	Sample Time Configuration Selects between different sample times based on the conversion mode selected. This field adjusts the sample period to allow higher impedance inputs to be accurately sampled or to maximize conversion speed for lower impedance inputs. Longer sample times can also be used to lower overall power consumption if continuous conversions are enabled and high conversion rates are not required. When ADLSMP=1, the long sample time select bits, (ADLSTS[1:0]), can select the extent of the long sample time.

Table continues on the next page...

ADC0_CFG1

Field	Description
	0 Short sample time. 1 Long sample time.
3–2 MODE	Conversion mode selection Selects the ADC resolution mode. 00 When DIFF=0:It is single-ended 8-bit conversion; when DIFF=1, it is differential 9-bit conversion with 2's complement output. 01 When DIFF=0:It is single-ended 12-bit conversion ; when DIFF=1, it is differential 13-bit conversion with 2's complement output. 10 When DIFF=0:It is single-ended 10-bit conversion. ; when DIFF=1, it is differential 11-bit conversion with 2's complement output 11 When DIFF=0:It is single-ended 16-bit conversion..; when DIFF=1, it is differential 16-bit conversion with 2's complement output
ADICLK	Input Clock Select Selects the input clock source to generate the internal clock, ADCK. Note that when the ADACK clock source is selected, it is not required to be active prior to conversion start. When it is selected and it is not active prior to a conversion start, when CFG2[ADACKEN]=0, the asynchronous clock is activated at the start of a conversion and deactivated when conversions are terminated. In this case, there is an associated clock startup delay each time the clock source is re-activated. 00 Bus clock 01 Bus clock divided by 2(BUSCLK/2) 10 Alternate clock (ALTCLK) 11 Asynchronous clock (ADACK)

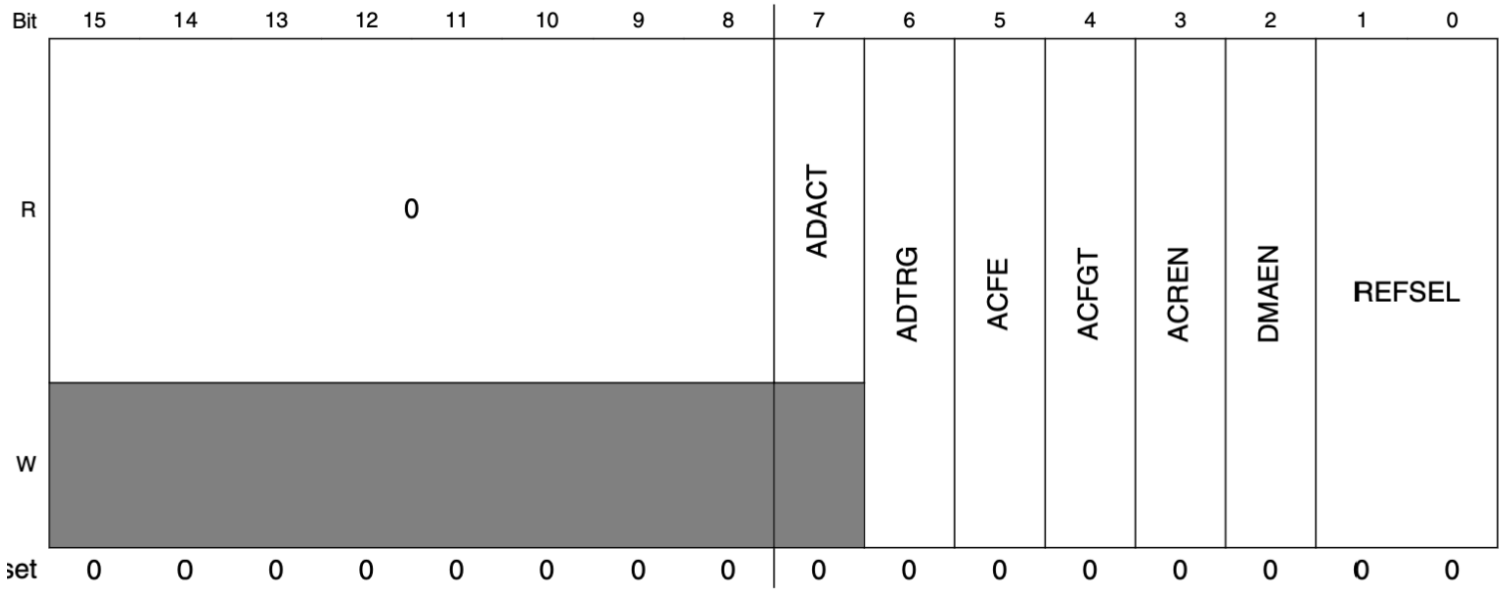
ADC0_CFG1

- We will keep everything as default, but use a 12-bit resolution (0b01):

```
// Set 12 bit conversion
ADC0->CFG1 &= ~ADC_CFG1_MODE_MASK;
ADC0->CFG1 |= ADC_CFG1_MODE(0b01);
```

ADC0_SC2

- The Status and Control Register (ADC0_SC2) lets us choose whether to trigger the conversion in software or via internal hardware, and choose the reference voltage source, amongst other things.



ADC0_SC2

ADCx_SC2 field descriptions

Field	Description
31–8 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
7 ADACT	Conversion Active Indicates that a conversion or hardware averaging is in progress. ADACT is set when a conversion is initiated and cleared when a conversion is completed or aborted. 0 Conversion not in progress. 1 Conversion in progress.
6 ADTRG	Conversion Trigger Select Selects the type of trigger used for initiating a conversion. Two types of trigger are selectable:

Table continues on the next page...

ADC0_SC2

Field	Description
	• Software trigger: When software trigger is selected, a conversion is initiated following a write to SC1A. • Hardware trigger: When hardware trigger is selected, a conversion is initiated following the assertion of the ADHWT input after a pulse of the ADHWTSn input. 0 Software trigger selected. 1 Hardware trigger selected.
5 ACFE	Compare Function Enable Enables the compare function. 0 Compare function disabled. 1 Compare function enabled.
4 ACFGT	Compare Function Greater Than Enable Configures the compare function to check the conversion result relative to the CV1 and CV2 based upon the value of ACREN. ACFE must be set for ACFGT to have any effect. 0 Configures less than threshold, outside range not inclusive and inside range not inclusive; functionality based on the values placed in CV1 and CV2. 1 Configures greater than or equal to threshold, outside and inside ranges inclusive; functionality based on the values placed in CV1 and CV2.
3 ACREN	Compare Function Range Enable Configures the compare function to check if the conversion result of the input being monitored is either between or outside the range formed by CV1 and CV2 determined by the value of ACFGT. ACFE must be set for ACFGT to have any effect. 0 Range function disabled. Only CV1 is compared. 1 Range function enabled. Both CV1 and CV2 are compared.

ADC0_SC2

2 DMAEN	DMA Enable 0 DMA is disabled. 1 DMA is enabled and will assert the ADC DMA request during an ADC conversion complete event noted when any of the SC1n[COCO] flags is asserted.
REFSEL	Voltage Reference Selection Selects the voltage reference source used for conversions. 00 Default voltage reference pin pair, that is, external pins V_{REFH} and V_{REFL} 01 Alternate reference pair, that is, V_{ALTH} and V_{ALTL} . This pair may be additional external pins or internal sources depending on the MCU configuration. See the chip configuration information for details specific to this MCU 10 Reserved 11 Reserved

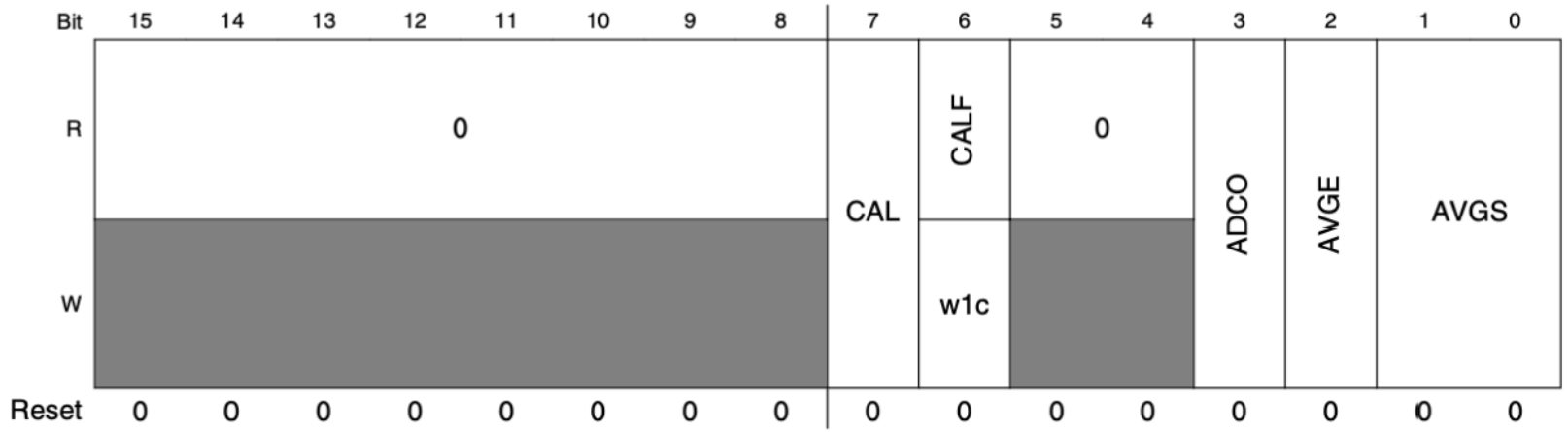
ADC_SC2

- We are interested in two fields in ADC0_SC2:
  - ADTRG: Select between software or hardware to trigger conversion.
    - We want to trigger in software by writing to ADC0_SC1.
  - REFSEL: Choose reference voltage.
    - The FRDM-MCXC444 board uses the alternate voltage reference VALTH and VALTL rather than the default voltage reference VREFH and VREFL.

```
// Use software trigger
ADC0->SC2 &= ~ADC_SC2_ADTRG_MASK;

// Use VALTH and VALTL
ADC0->SC2 &= ~ADC_SC2_REFSEL_MASK;
ADC0->SC2 |= ADC_SC2_REFSEL(0b01);
```

ADC0_SC3

- The Status and Control Register 3 (ADC0_SC3) lets us control calibration, averaging and continuous conversion.



ADC0_SC3

Field	Description
31–8 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
7 CAL	Calibration Begins the calibration sequence when set. This field stays set while the calibration is in progress and is cleared when the calibration sequence is completed. CALF must be checked to determine the result of the calibration sequence. Once started, the calibration routine cannot be interrupted by writes to the ADC registers or the results will be invalid and CALF will set. Setting CAL will abort any current conversion.
6 CALF	Calibration Failed Flag Displays the result of the calibration sequence. The calibration sequence will fail if SC2[ADTRG] = 1, any ADC register is written, or any stop mode is entered before the calibration sequence completes. Writing 1 to CALF clears it. 0 Calibration completed normally. 1 Calibration failed. ADC accuracy specifications are not guaranteed.

ADC0_SC3

Field	Description
5–4 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
3 ADCO	Continuous Conversion Enable Enables continuous conversions. 0 One conversion or one set of conversions if the hardware average function is enabled, that is, AVGE=1, after initiating a conversion. 1 Continuous conversions or sets of conversions if the hardware average function is enabled, that is, AVGE=1, after initiating a conversion.
2 AVGE	Hardware Average Enable Enables the hardware average function of the ADC. 0 Hardware average function disabled. 1 Hardware average function enabled.
AVGS	Hardware Average Select Determines how many ADC conversions will be averaged to create the ADC average result. 00 4 samples averaged. 01 8 samples averaged. 10 16 samples averaged. 11 32 samples averaged.

ADC0_SC3

- ADC0 can average across 4, 8, 16 or 32 samples to produce its value.
  - We aren't interested in this, so we are going to turn off averaging.
- We can do continuous conversion:
  - A new conversion begins after the previous ends.

```
// Don't use averaging
ADC0->SC3 &= ~ADC_SC3_AVGE_MASK;
ADC0->SC3 |= ADC_SC3_AVGE(0);

// Use continuous conversion
ADC0->SC3 &= ~ADC_SC3_ADCO_MASK;
ADC0->SC3 |= ADC_SC3_ADCO(1);
```

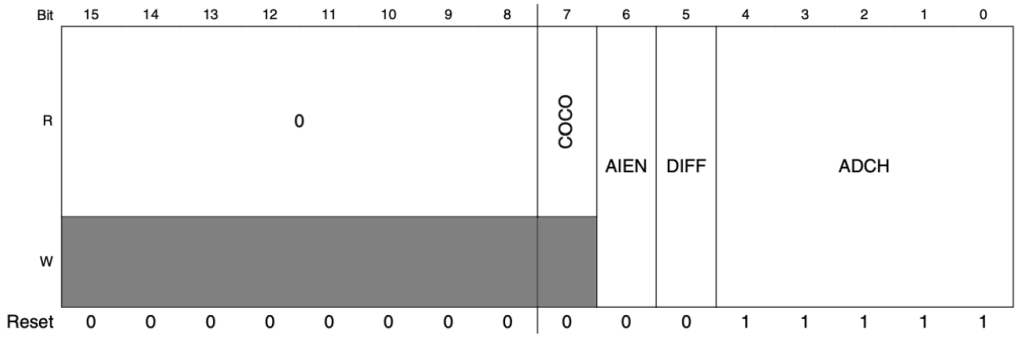
Setting up IRQ

- ADC0 triggers the ADC0_IRQn interrupt if AIEN in ADC0_SC1 is set.
  - We set the priority number and enable interrupts:

```
// Configure interrupt
NVIC_DisableIRQ(ADC0_IRQn);
NVIC_ClearPendingIRQ(ADC0_IRQn);

// Lowest priority
NVIC_SetPriority(ADC0_IRQn, 192);
NVIC_EnableIRQ(ADC0_IRQn);
```

Starting Conversion

- The ADCH field in ADC0_SC1 lets us choose which channel to convert:
  - Set to 0b11111 to stop conversions

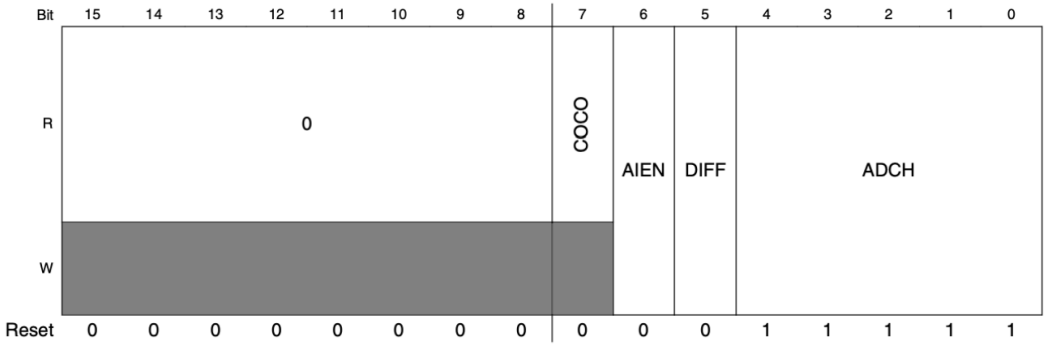


  - Set to any other channel to start conversion.

```
ADC0->SC1[0] &= ~ADC_SC1_ADCH_MASK;
ADC0->SC1[0] |= ADC_SC1_ADCH(channel);
```

Getting Results

- We have configured our ADC to trigger ADC0_IRQn each time conversion is completed.
  - Caught by the ADC0_IRQHandler ISR.
- Must check the COCO (Conversion Complete) flag in ADC0_SC1a.
  - Actual result can be read from ADC0_Ra (ADC0->R[0])



Getting Results

```
void ADC0_IRQHandler() {
    NVIC_ClearPendingIRQ(ADC0_IRQn);

    if(ADC0->SC1[0] & ADC_SC1_COCO_MASK){
        int result = ADC0->R[0];
        PRINTF("IRQ: Value = %d\r\n", result);
        // Use results
    }
}
```

Conclusion

- In this chapter we looked at how to program the Analog to Digital Convertor (ADC) on the MCXC444.
- In the next chapter we will look at how to do serial communications on the MCXC444.