6 Timing Generation and Measurements

Contents

6.2

Introduction

The K70 has several timer modules:

Module	Description
Programmable delay block (PDB)	The PDB provides controllable delays from either an internal or an external trigger, or a programmable interval tick, to the hardware trigger inputs of ADCs and/or generates the interval triggers to DACs, so that the precise timing between ADC conversions and/or DAC updates can be achieved.
FlexTimer modules (FTM)	The FTM is an eight channel timer that supports input capture, output compare, and the generation of PWM signals.
Periodic interrupt timers (PIT)	The PIT module is an array of 4 timers that can be used to raise interrupts and trigger DMA channels.
Low-power timer (LPTMR)	The LPTMR can be configured to operate as a time counter with optional prescaler, or as a pulse counter with optional glitch filter, across all power modes, including the low-leakage modes. It can also continue operating through most system reset events, allowing it to be used as a time of day counter.
Carrier modulator timer (CMT)	The CMT module provides a means to generate the protocol timing and carrier signals for a variety of encoding schemes used in infrared remote controls.
Real-time clock (RTC)	The RTC operates off an independent power supply and 32 kHz crystal oscillator and has a 32-bit seconds counter with a 32-bit alarm.
IEEE 1588 timers	The IEEE 1588 standard provides accurate clock synchronization for distributed control nodes for industrial automation applications.

6.1 FlexTimer Module

The FlexTimer Module (FTM) has the capability of capturing events and time-stamping them, and of generating events at certain times. It also has logic to generate PWM waveforms without the need for software intervention. A simplified block diagram of the FlexTimer module is shown below:

A simplified FTM block diagram

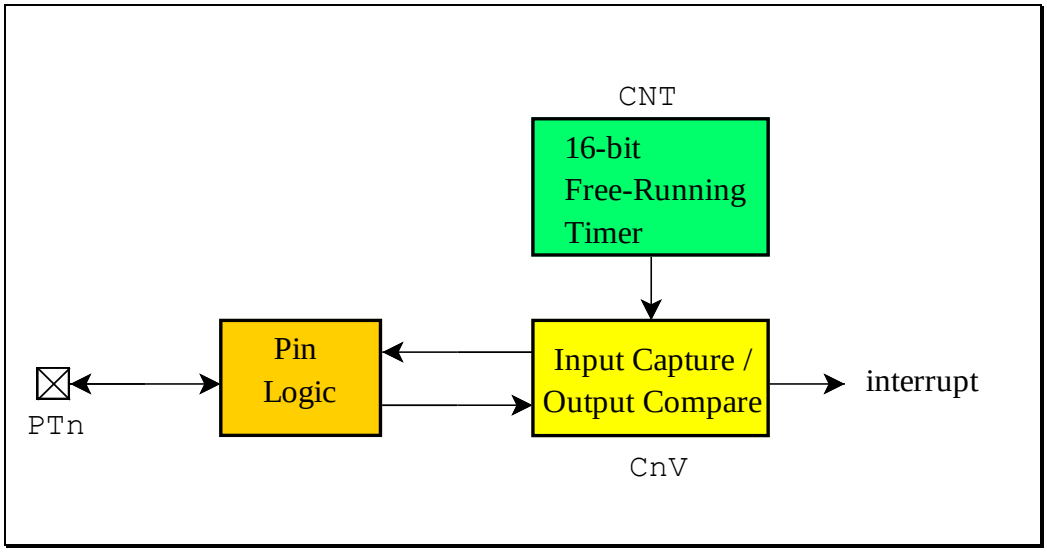


Figure 6.1

There is a 16-bit free-running timer called `CNT`. This is used to time-stamp an input event (an *input capture*) or to trigger an output event (an *output compare*).

The timer has a free-running counter and a value counter for each channel

The "input capture / output compare" block is just a register called `CnV`, where `n` is the channel number, that gets loaded with the current value of `CNT` for an input capture event, and which holds a desired value of `CNT` to trigger an output compare event.

The types of events to capture, or to initiate on a successful compare, are setup through various control registers. For inputs, it is possible to capture rising and falling edges. Outputs can be made to toggle, clear or be set.

There are numerous control registers used to set up the FTM module. Only a few are needed to interact with the FTM once it has been set up for a particular application. A complete description of the FTM can be found in Chapter 43 of NXP's *K70 Sub-Family Reference Manual*.

6.1.1 Output Compare

Output compare can be used to create square waves, generate pulses, implement time delays, and execute periodic interrupts. You can use output compare together with input capture to measure period and frequency over a wide range and with varying resolution.

A channel set up as an *output compare* channel will trigger an output action when the output compare register is equal to the free-running timer. A block diagram of the output compare action is shown below:

A simplified output compare block diagram

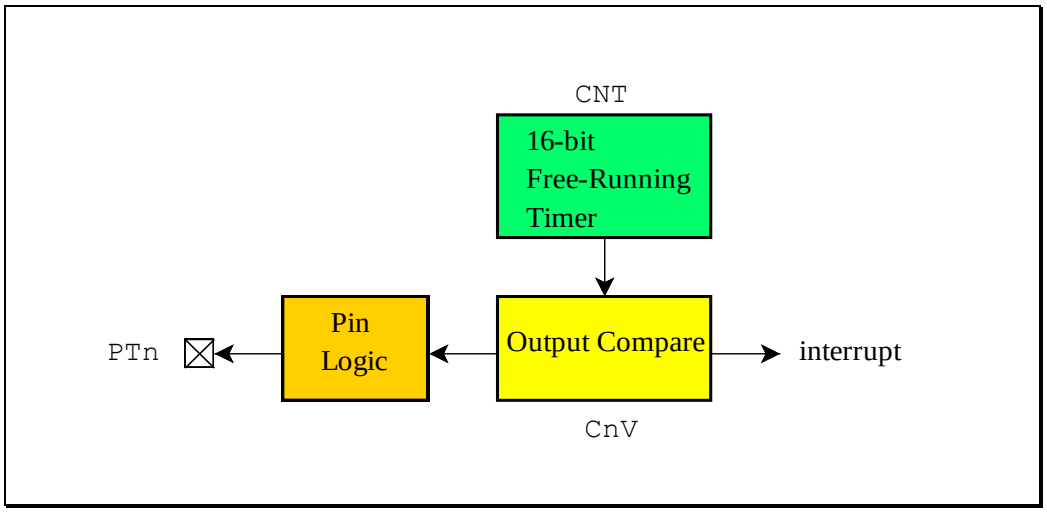


Figure 6.2

A compare result output action can be set up using the Channel Status and Control Register, `CnSC`, for the relevant channel. The options are:

Output compare actions

Action
Timer disconnected from output pin logic
Toggle output on match
Clear output on match
Set output on match

6.1.2 Time Delay Using Output Compare

One simple application of the output compare feature is to create a fixed timer. Timers are useful in situations where you start an operation, wait a certain amount of time, and then stop the operation. Usually the process looks like this:

1. Start an operation (turn on or turn off an output device).
2. Start the timer.
3. When the timer expires, stop the operation (turn off or turn on the output device).

You can also use timers to detect timeout conditions. For example, you turn on a motor and then start a timer. You expect the speed of the motor to increase, and if the speed doesn't exceed a threshold before a timer times out, then you might turn the motor off and notify an operator. In these cases, you start an operation then monitor the process to see if conditions are met before the timer expires:

1. Start an operation (turn on or turn off an output device).
2. Start the timer.
3. Monitor for desired conditions. If conditions are met, stop the timer.
4. If the timer times out, stop the operation and notify the operator.

Let `delay` be the number of cycles you wish to wait. The steps to start a timer are:

1. Read the current 16-bit `CNT`.
2. Set the 16-bit output compare register to `CNT + delay;`
3. Clear the output compare flag.
4. The output compare flag will set and trigger an interrupt after the required delay.

This method will only work for values of `delay` that fall between a minimum value (the time it takes to implement steps 1 to 3) and 65536. It will function properly even if `CNT` rolls over from `0xFFFF` to `0`, since the 16-bit addition is really a modulo `0x10000` addition.

EXAMPLE 6.1 Creating a Timer Using Output Compare

The output compare feature is a convenient mechanism to create a simple timer. We will turn an LED on when the timer on channel 3 of FTM0 times out, after 1 second.

```
// Timer value for 1 second
const uint16_t RATE = 24414;

void FTM0_Init(void)
{
  // Set up FTM0
  ...
  // Initialize NVIC
  ...
}

void __attribute__ ((interrupt)) FTM0_ISR(void)
{
  // Clear interrupt flag
  FTM0_CnSC(3) &= ~FTM_CnSC_CHF_MASK;

  // Call user function
  LED_On();
}

void TOC3_Init(void)
{
  // Ensure interrupts are disabled
  __DI();

  // Initialise the FTM0 module
  FTM0_Init();

  // Enable Ch 3 as output compare with interrupts enabled
  FTM0_CnSC(3) = (FTM_CnSC_MSA_MASK | FTM_CnSC_CHIE_MASK);

  // Time out for 1 second
  FTM0_C3V = FTM0_CNT + RATE;

  // Enable interrupts
  __EI();
}
```

In the vector table in `vectors.c`, we put:

```
  ...
  (tIsrFunc)&Cpu_Interrupt,          /* 0x4D  CMP2        */
  (tIsrFunc)&FTM0_ISR,               /* 0x4E  FTM0        */
  (tIsrFunc)&Cpu_Interrupt,          /* 0x4F  FTM1        */
  ...
```

6.1.3 Input Capture

A channel can be set up as an *input capture* channel. We can use input capture to measure the period or pulse width of 3.3V CMOS signals. The input capture system can also be used to trigger interrupts on rising or falling transitions of external signals. A simplified block diagram of a channel set up for input capture is shown below:

A simplified input capture block diagram

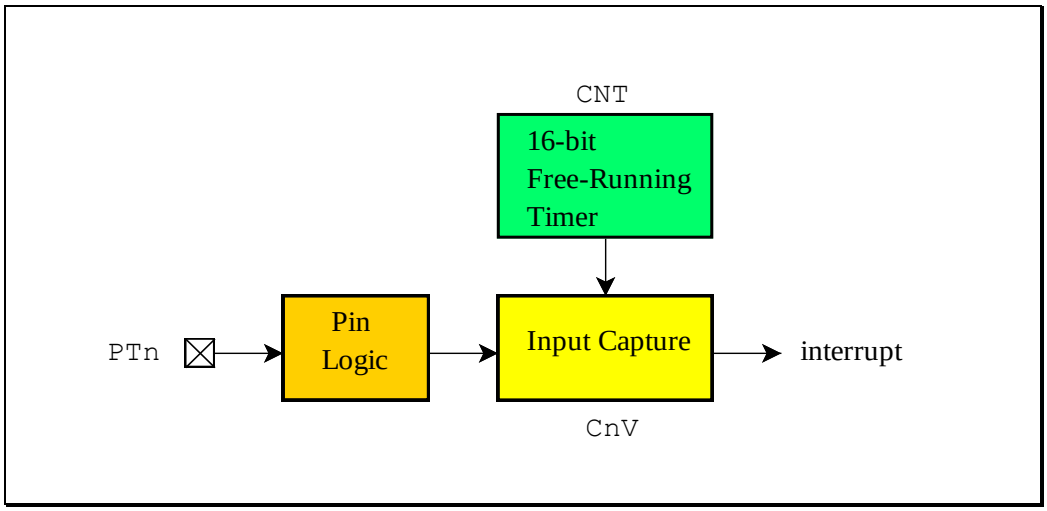


Figure 6.3

The input capture edge detection circuits can be set up using the CnSC register. The options are:

Input compare configurations

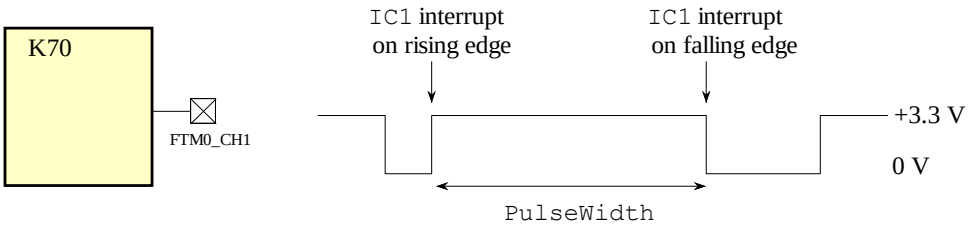
Configuration
Capture disabled
Capture on rising edges only
Capture on falling edges only
Capture on any edge (rising or falling)

Two or three actions result from a capture event:

1. The current 16-bit CNT value is copied into the input capture register, CnV.
2. The input capture flag is set in CnSC.
3. An interrupt is requested when the CHIE bit is 1 in CnSC.

EXAMPLE 6.2 Pulse-Width Measurement

The basic idea of pulse-width measurement is to cause an input capture event on first the rising edge and then the falling edge of an input signal. The difference between these two times will be the pulse width.



The resolution of the measurement is determined by the rate at which CNT is incremented. We will use Channel 1 of the timer for the implementation.

```
uint16_t PulseWidth;     // Pulse width in CNT units
BOOL Done;               // True when pulse width is measured

void FTM0_Init(void)
{
  // Set up FTM0
  ...
  // Initialize NVIC
  ...
}

void __attribute__ ((interrupt)) FTM0_ISR(void)
{
  // Value of CNT at rising edge
  static uint16_t rising;

  // Clear interrupt flag
  FTM0_CnSC(1) &= ~FTM_CnSC_CHF_MASK;

  // See if a rising edge is detected
  if (FTM0_CH1)
  {
    // Record time of rising edge
    rising = FTM0_C1V;
    // Set edge detection to falling edge only
    FTM0_CnSC(1) = (FTM_CnSC_ELSB_MASK | FTM_CnSC_CHIE_MASK);
  }
  else
  {
    // Falling edge detected – calculate the pulse width
    PulseWidth = FTM0_C1V – rising;
    Done = bTRUE;
  }
```

```
}

void TIC1_Init(void)
{
  // Ensure interrupts are disabled
  __DI();

  // Initialise the FTM0 module
  FTM0_Init();

  // Enable Ch1 as input capture on rising edge
  FTM0_CnSC(1) = (FTM_CnSC_ELSA_MASK | FTM_CnSC_CHIE_MASK);

  // No measurement yet
  Done = bFALSE;

  // Enable interrupts
  __EI();
}
```

In the vector table in `vectors.c`, we put:

```
  ...
  (tIsrFunc)&Cpu_Interrupt,          /* 0x4D  CMP2        */
  (tIsrFunc)&FTM0_ISR,               /* 0x4E  FTM0        */
  (tIsrFunc)&Cpu_Interrupt,          /* 0x4F  FTM1        */
  ...
```

6.2 Periodic Interrupt Timer

A block diagram of one of the periodic interrupt timer (PIT) is shown below:

A simplified periodic interrupt timer block diagram

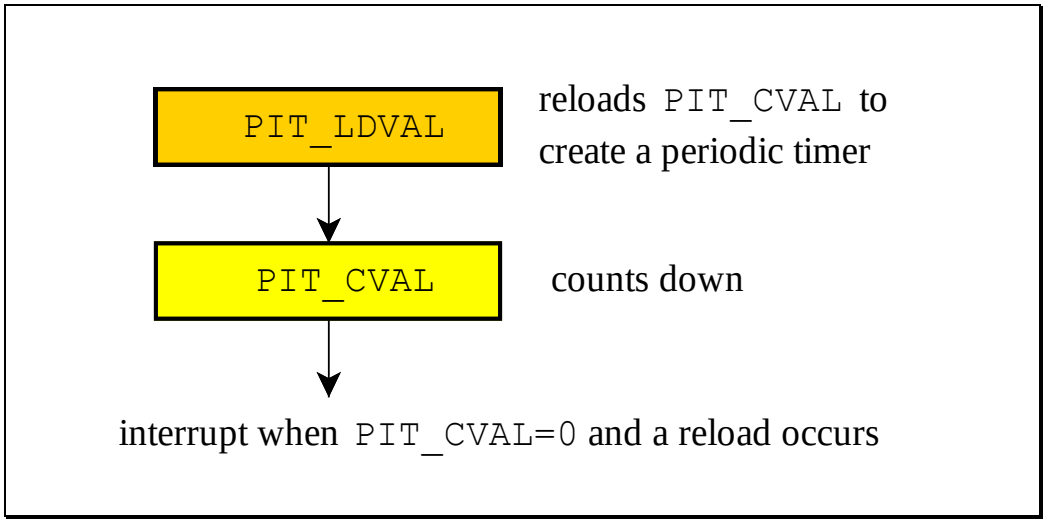


Figure 6.4

A PIT generates triggers at periodic intervals, when enabled. The timer loads the start value as specified in the LDVAL register, counts down to 0 and then loads the respective start value again. Each time the timer reaches 0, it will generate a trigger pulse and set the interrupt flag.

Note that an interrupt will occur (if enabled) when the CVAL register reaches zero and the next clock "tick" reloads the start value as specified in the LDVAL register. For example, to create a timer with a period of 1000 "ticks" of the module clock, the LDVAL register needs to be loaded with 999.

A complete description of the PIT can be found in Chapter 44 of NXP's *K70 Sub-Family Reference Manual*.

6.3 Real Time Clock

The management of time is important in many embedded systems. The real time clock (RTC) unit in the K70 operates in a separate power domain – it operates from *Vbat*, which is connected to a battery. It can therefore keep track of the time while the main power is off.

The RTC unit relies on an external 32.768 kHz crystal for its timekeeping. The crystal must have "load" capacitors connected to it which are internal to the K70 – the selection of the load capacitors is accomplished by bits in the RTC Control register, `RTC_CR`.

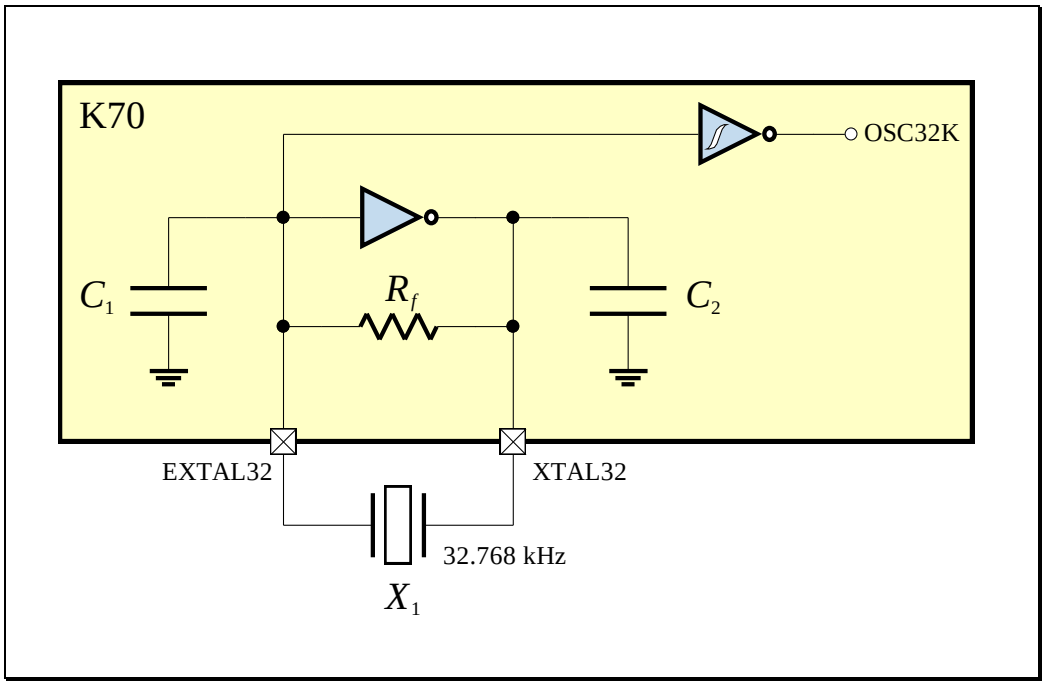


Figure 6.5

The RTC has the ability to generate an interrupt every second.

A complete description of the RTC can be found in Chapter 47 of NXP's *K70 Sub-Family Reference Manual*.