Freescale Semiconductor
Application Note

Document Number: AN5083
Rev 0, 01/2015

Using DMA for pulse counting on Kinetis

Contents

1 Introduction

This application note explains how to use DMA and GPIO modules for pulse counting. Many Kinetis series microcontrollers have a Low-Power Timer and FlexTimer that can be used as pulse input capture. However, when applications need to count multiple pulse inputs, Low Power Timer cannot meet the requirements. For example, LPTMR (Low-Power Timer Module) have only one counter inside. It can only be used as a single pulse counter.

This application aims to use GPIO to capture multiple pulse input and use the DMA module channel countdown register to count pulses.

By using this method Kinetis chips can count up to 5 channels pulse inputs simultaneously without CPU interference .

The compatibility of Kinetis MCUs allows this application to be implemented on different sub-families of Kinetis besides K60. The fact all Kinetis K series shares the same modules (IP blacks) enable easy reuse of the code across the entire K sub-families.

2 Functional description

A DMA request is generated when the peripheral request and trigger are effective simultaneously, When DMA finishes a minor loop transfer, the DMA channel counter counts down from the reload value until it reaches zero. If we set the DMA major loop counter to be the a proper value and set the DMA trigger source to be external edge sensitive capture from a port, we can do pulse input capture from this port by reading DMA channel's countdown register.

Kinetis K series includes a DMA request MUX that allows up to 64 DMA request signals to be mapped to any of the DMA channels. Thus, we arrange PORTA-PORTE's rising/falling trigger source to DMA channel 0-5.

Table 1. Channel request source

DMA channel number	DMA request source number	Source description
DMA channel 0	49	PORTA request
DMA channel 1	50	PORTB request
DMA channel 2	51	PORTC request
DMA channel 3	52	PORTD request
DMA channel 4	53	PORTE request

The following code snippet shows how to configure DMA channel request source to the corresponding PORT request.

```
DMAMUX0_CHCFG0 = DMA MUX_CHCFG_ENBL_MASK| DMA MUX_CHCFG_SOURCE(49); //PORTA request
DMAMUX0_CHCFG1 = DMA MUX_CHCFG_ENBL_MASK| DMA MUX_CHCFG_SOURCE(50); //PORTB request
DMAMUX0_CHCFG2 = DMA MUX_CHCFG_ENBL_MASK| DMA MUX_CHCFG_SOURCE(51); //PORTC request
DMAMUX0_CHCFG3 = DMA MUX_CHCFG_ENBL_MASK| DMA MUX_CHCFG_SOURCE(52); //PORTD request
DMAMUX0_CHCFG4 = DMA MUX_CHCFG_ENBL_MASK| DMA MUX_CHCFG_SOURCE(53); //PORTE request
```

NOTE

One PORT instance can only has one pin to be used as DMA pulse input, because DMA trigger source is based on PORT. So, the maximum channel number for pulse input equals to chip's PORT module instance.

3 Initialization and application

To initialize DMA multiple pulse counting capability, perform the following steps:

For DMA module:

1. Enable DMAMUX and DMA clock and configure DMAMUX request source.
2. Set DMA source and destination address to a dummy variable address.
3. Set DMA source and destination minor and major address adjustment and offset to zero.
4. Set each DMA channel's countdown counter(CITER, BITER) to proper value.
5. Enable DMA channel request to wait peripheral trigger signal.

For GPIO and PORT module:

1. Set PORT pin mux to GPIO.
2. Enable DMA rising/falling request on PORT_MUX register.

For application:

1. Initializing DMA and GPIO module.

2. Read DMA channel counters value in application to get pulse count value.

The example code is shown below:

```
/* clock gate initialization */
SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK,
SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK,
SIM->SCGC5 |= SIM_SCGC5_PORTC_MASK,
SIM->SCGC5 |= SIM_SCGC5_PORTD_MASK,
SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK,

/* set pin to GPIO function and enable DMA request source */
PORTA->PCR[0] |= PORT_PCR_MUX(1)| PORT_PCR_IRQC(1);
PORTA->PCR[0] |= PORT_PCR_MUX(1)| PORT_PCR_IRQC(1);
PORTA->PCR[0] |= PORT_PCR_MUX(1)| PORT_PCR_IRQC(1);
PORTA->PCR[0] |= PORT_PCR_MUX(1)| PORT_PCR_IRQC(1);
PORTA->PCR[0] |= PORT_PCR_MUX(1)| PORT_PCR_IRQC(1);

/* enable DMA and DMAMUX clock */
SIM->SCGC6 |= SIM_SCGC6_DMAMUX_MASK;
SIM->SCGC7 |= SIM_SCGC7_DMA_MASK;
```

Before configuring DMA MUX trigger and source number, disablethe DMAMUX->CFG[ENBL] bit.

```
      /* clears register for changing source and trigger */
DMAMUX->CHCFG[1] = 0;
DMAMUX->CHCFG[2] = 0;
DMAMUX->CHCFG[3] = 0;
DMAMUX->CHCFG[4] = 0;

/* set DMA channel request source */
DMAMUX->CHCFG[0]= DMA MUX_CHCFG_ENBL_MASK| DMA MUX_CHCFG_SOURCE(49);
DMAMUX->CHCFG[1]= DMA MUX_CHCFG_ENBL_MASK| DMA MUX_CHCFG_SOURCE(50);
DMAMUX->CHCFG[2]= DMA MUX_CHCFG_ENBL_MASK| DMA MUX_CHCFG_SOURCE(51);
DMAMUX->CHCFG[3]= DMA MUX_CHCFG_ENBL_MASK| DMA MUX_CHCFG_SOURCE(52);
DMAMUX->CHCFG[4]= DMA MUX_CHCFG_ENBL_MASK| DMA MUX_CHCFG_SOURCE(53);
```

Before initializing DMA register, you need to fill the DMA source and destination address register into a dummy variable's address value.

DMA module initialization. For an example, only take the DMA channel 0.

```
/* source configuration */
DMA0->TCD[0].SADDR = &dummy;
DMA0->TCD[0].ATTR = DMA_ATTR_SSIZE(0);
DMA0->TCD[0].SOFF= 0; /* no address shift after each transfer */
DMA0->TCD[0].SLAST = 0;

/* destination configuration */
DMA0->TCD[0].DADDR = &dummy;
DMA0->TCD[0].ATTR = DMA_ATTR_DSIZE(0);
DMA0->TCD[0].DOFF= 0;
DMA0->TCD[0].DLAST_SGA= 0;

/* set CITER and BITER to maximum value */
DMA0->TCD[0].CITER_ELINKNO = DMA_CITER_ELINKNO_CITER_MASK;
DMA0->TCD[0].BITER_ELINKNO = DMA_CITER_ELINKNO_BITER_MASK;
DMA0->TCD[0].NBYTES_MLNO = 1; /* transfer one byte on each trigger arrived */

/* enable auto close request */
DMA0->TCD[0].CSR |= DMA_CSR_DREQ_MASK;

/* start transfer */
DMA0->SERQ = DMA_SERQ_SERQ(0);
```

Read DMA channel counter in your application. Pulse counting value is BITER minus CITER.

NOTE

You can read DMA channel counter at any time in your application BEFORE major iteration count is exhausted, which means when CITER reaches zero. Once CITER reaches zero, its user's responsibility to reset CITER register to BITER and restart counting operation.

```
Uint32_t value;
 value = DMA0->TCD[0]. BITER_ELINKNO - DMA0->TCD[0].CITER_ELINKNO;
```

4 Conclusion

The DMA multiple pulse counter implementation totally completed by internal hardware logic. The user only needs to read DMA channel countdown register to get pulse counting value. The whole process will not occupy any CPU resource.

Figure 1 shows the result of using DMA to capture three different square waves with different frequency (10 KHz, 20 KHz, and 1 KHz). Input square waves:

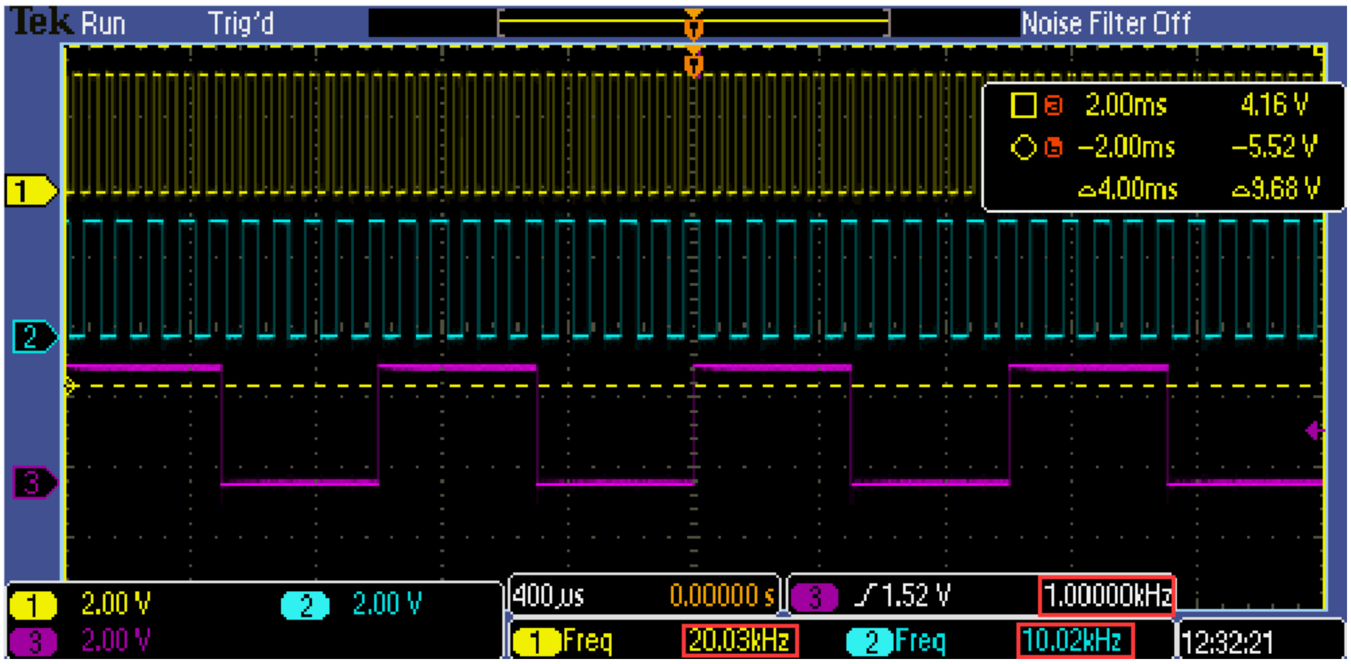
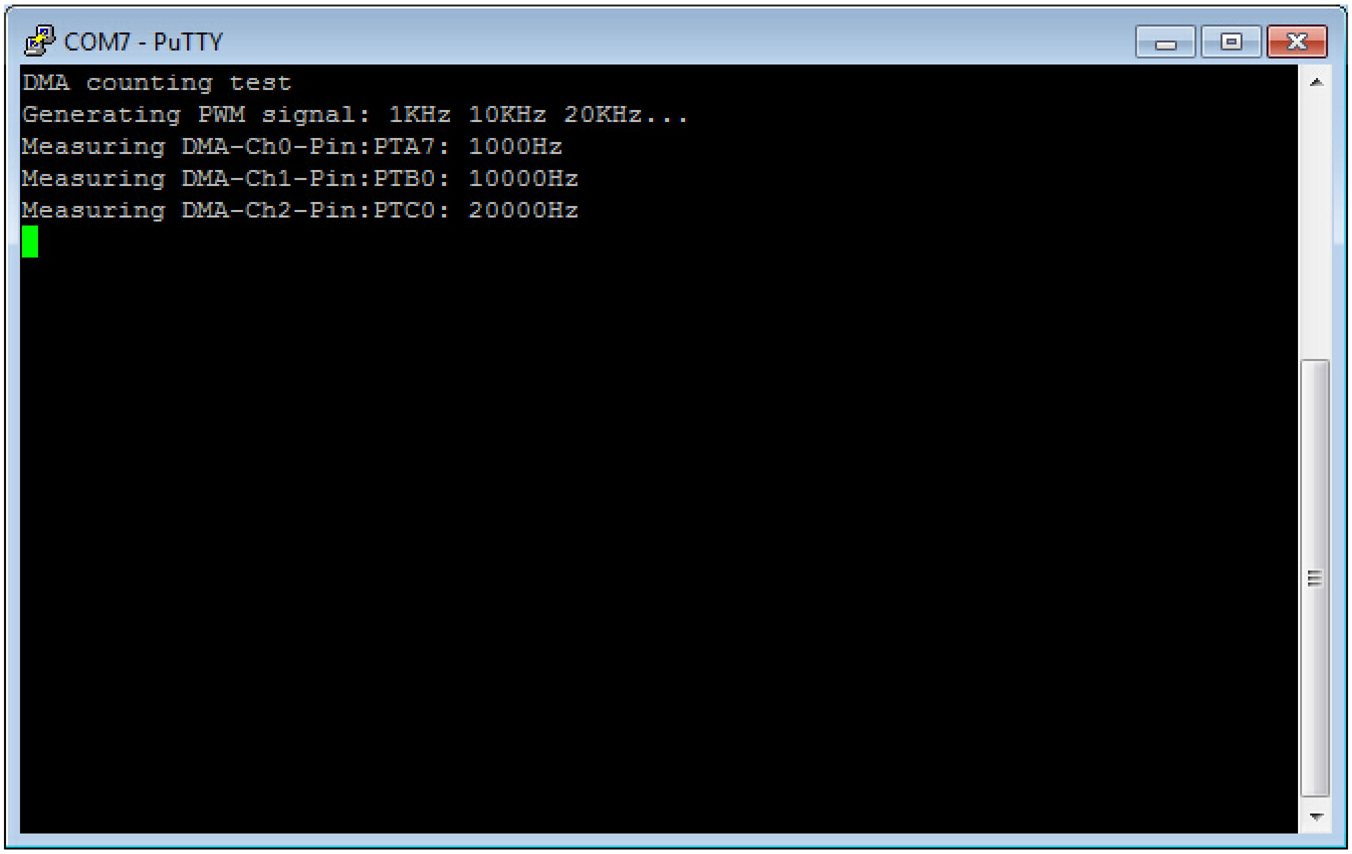


Figure 1. Input square waves

To measure square wave's frequency:

1. Enable a PIT timer to generator second interrupt.
2. Read each DMA's channel counter in interrupt services and reset counter.
3. Display counting result on serial terminal.



```
DMA counting test
Generating PWM signal: 1KHz 10KHz 20KHz...
Measuring DMA-Ch0-Pin:PTA7: 1000Hz
Measuring DMA-Ch1-Pin:PTB0: 10000Hz
Measuring DMA-Ch2-Pin:PTC0: 20000Hz
```

Figure 2. Counting result on serial terminal

5 Functional limitations

There are several limitations that must be considered during implementation. Solutions are suggested for some of these limitations.

5.1 Maximum DMA rate

The maximum transfer rate is limited for DMA hardware requests, especially for single byte transfers. The maximum achievable rate for 100MHz device is 8.7MHz, Thus, the maximum input frequency cannot exceed that speed.

5.2 DMA major loop counter

The major loop counter of the DMA register DMA_TCDn_CITER_ELINKNO is limited to 15 bits, in the case where the channel linking feature of the DMA is not in use (refer to DMA chapter in the Reference Manual). Therefore, there is a limitation of 32K major loops count. So, user must check CITER register and make sure it will not reach to zero. If, CITER reaches zero, BITER will reload the value from BITER register.

Document Number AN5083
Revision 0, 01/2015