Freescale Semiconductor, Inc.
Application Note

Document Number: AN5258
Rev. 0, 02/2016

Using DMA for Pulse Counting on S32K

1. Introduction

This application note describes pulse counting on the S32K product series of 32-bit automotive MCUs using the Enhanced Direct Memory Access (eDMA) and PORT modules. The number of the signal pulses can be counted using the Low-Power Timer (LPTMR) or the FlexTimer (FTM). However, the number of pulse inputs is limited, because the FTM module is usually used for PWM generation and LPTMR has only one counter inside.

In this application note, the PORT module is utilized to capture multiple pulse inputs and the eDMA channel's Current Major Iteration Counter (CITER) register is used as a simple pulse counter.

This S32K chip configuration offers up to five pulse input channels that can be evaluated simultaneously. This way, the CPU load is also rapidly reduced.

This application note addresses primarily the S32K product series of 32-bit automotive MCUs.

Contents

2. Functional Description

The eDMA is used primarily for data transfers between the main memory and another peripheral register space without CPU interference. CITER is a part of the eDMA structure and it can be also used for pulse counting.

After receiving the peripheral request, the minor loop of the eDMA module starts to perform a basic data transfer. After the minor loop completes, CITER decrements. To capture the active edge from a port pin, set the major loop counter of the eDMA module to the maximum value. The CITER register value then reflects the actual pulse count.

The S32K device includes DMA Multiplexer (DMAMUX) that enables mapping up to 63 eDMA request signals to any of the 16 eDMA channels. To capture multiple pulse inputs, PORTA–PORTE can be set as sources for channels 0–5 of the eDMA module.

Table 1. Channel request source

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

This code shows how to configure the eDMA channel request source to the corresponding PORT request:

```
DMAMUX->CHCFG[0] = DMAMUX_CHCFG_SOURCE(49) | DMAMUX_CHCFG_ENBL_MASK;     // PORTA request
DMAMUX->CHCFG[1] = DMAMUX_CHCFG_SOURCE(50) | DMAMUX_CHCFG_ENBL_MASK;     // PORTB request
DMAMUX->CHCFG[2] = DMAMUX_CHCFG_SOURCE(51) | DMAMUX_CHCFG_ENBL_MASK;     // PORTC request
DMAMUX->CHCFG[3] = DMAMUX_CHCFG_SOURCE(52) | DMAMUX_CHCFG_ENBL_MASK;     // PORTD request
DMAMUX->CHCFG[4] = DMAMUX_CHCFG_SOURCE(53) | DMAMUX_CHCFG_ENBL_MASK;     // PORTE request
```

NOTE

Because a single PORT instance can generate a single eDMA request signal, only a single pin within the PORT instance has to be configured as a pulse input. Therefore, the maximum number of the pulse inputs equals the number of chip's PORT module instances. Each PORT instance must have only a single pin configured as a pulse input.

3. Initialization and Application

To capture multiple pulse inputs, follow these eDMA and PORT initialization steps:

PORT module:

1. Enable the PORT clock.
2. Set the PORT pin multiplexer to the GPIO mode.
3. Enable the eDMA request on the rising/falling edge.

eDMA module:

1. Enable the DMAMUX and eDMA clocks.
2. Configure the DMA channel request source in DMAMUX.
3. Set the eDMA channel source and destination addresses to dummy variable addresses.
4. Set the eDMA channel source and destination minor and major address adjustment and offset to 0.
5. Set each eDMA channel CITER and BITER registers to the maximum value.
6. Enable the eDMA channel requests.

Application:

1. Initialize the eDMA and PORT modules.
2. Read the eDMA channel CITER register value to determine the actual pulse count.

This is the example code:

```
/* Variables declaration */
uint32_t dummy1;
uint32_t frequency_ch1;

void PORT_Init()
{
/* Enable clock for PORTA-PORTE */
PCC->PCCn[PCC_PORTA_INDEX] = PCC_PCCn_CGC_MASK;
PCC->PCCn[PCC_PORTB_INDEX] = PCC_PCCn_CGC_MASK;
PCC->PCCn[PCC_PORTC_INDEX] = PCC_PCCn_CGC_MASK;
PCC->PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK;
PCC->PCCn[PCC_PORTE_INDEX] = PCC_PCCn_CGC_MASK;

/* Configure PORT pin to GPIO mode and enable eDMA request on rising/falling edge */
PORTA->PCR[8] = PORT_PCR_MUX(1) | PORT_PCR_IRQC(1);
PORTB->PCR[8] = PORT_PCR_MUX(1) | PORT_PCR_IRQC(1);
PORTC->PCR[8] = PORT_PCR_MUX(1) | PORT_PCR_IRQC(1);
PORTD->PCR[8] = PORT_PCR_MUX(1) | PORT_PCR_IRQC(1);
PORTE->PCR[8] = PORT_PCR_MUX(1) | PORT_PCR_IRQC(1);
}

void DMAMUX_Init()
{
/* Enable clock for DMAMUX */
PCC->PCCn[PCC_DMAMUX0_INDEX] = PCC_PCCn_CGC_MASK;

/* Select an enable eDMA channel trigger source */
DMAMUX->CHCFG[0] = DMAMUX_CHCFG_SOURCE(49) | DMAMUX_CHCFG_ENBL_MASK;
DMAMUX->CHCFG[1] = DMAMUX_CHCFG_SOURCE(50) | DMAMUX_CHCFG_ENBL_MASK;
DMAMUX->CHCFG[2] = DMAMUX_CHCFG_SOURCE(51) | DMAMUX_CHCFG_ENBL_MASK;
DMAMUX->CHCFG[3] = DMAMUX_CHCFG_SOURCE(52) | DMAMUX_CHCFG_ENBL_MASK;
DMAMUX->CHCFG[4] = DMAMUX_CHCFG_SOURCE(53) | DMAMUX_CHCFG_ENBL_MASK;
}
```

Before initializing eDMA register, it is necessary to fill in eDMA source and destination address into a dummy variable's address value. eDMA module initialization for one channel is performed as follows:

```
void DMA_Init()
{
/* Enable clock for eDMA */
PCC->PCCn[PCC_DMA0_INDEX] = PCC_PCCn_CGC_MASK;

/* Source Configuration */
DMA->TCD0_SADDR = &dummy1;
DMA->TCD0_ATTR = DMA_TCD0_ATTR_SSIZE(0);
DMA->TCD0_SOFF = 0; // no address shift after each transfer
DMA->TCD0_SLAST = 0;

/* Destination Configuration */
DMA->TCD0_DADDR = &dummy1;
DMA->TCD0_ATTR = DMA_TCD0_ATTR_DSIZE(0);
DMA->TCD0_DOFF = 0; // no address shift after each transfer
DMA->TCD0_DLASTSGA = 0;

/* Set Citer and Biter to Maximum Value */
DMA->TCD0_CITER.TCD0_CITER_ELINKNO = DMA_TCD0_CITER_ELINKNO_CITER_MASK;
DMA->TCD0_BITER.TCD0_BITER_ELINKNO = DMA_TCD0_BITER_ELINKNO_BITER_MASK;
DMA->TCD0_NBYTES.TCD0_NBYTES_MLNO = 1; // transfer one byte on each trigger arrived

/* Start Transfer for Channel0 */
DMA->SERQ = DMA_SERQ_SERQ(0);
}
```

Read eDMA channel counter in your application. Pulse counting value equals to BITTER minus CITTER register value:

```
frequency_ch1 = (DMA->TCD0_BITER.TCD0_BITER_ELINKNO & DMA_TCD0_BITER_ELINKNO_BITER_MASK)-
                (DMA->TCD0_CITER.TCD0_CITER_ELINKNO & DMA_TCD0_CITER_ELINKNO_CITER_MASK);
```

NOTE

You can read the eDMA channel counter at any time in your application before the major iteration count is exhausted (when CITER reaches 0). When CITER reaches 0, it is user's responsibility to reset the CITER register to BITER and restart the counting operation. Configure the eDMA channel to generate an interrupt when CITER reaches 0 by setting the INTMAJOR bit in the TCDn_CSR register.

4. Functional Limitations

Consider the limitations of the implementation demonstrated in this application note.

4.1. Maximum eDMA transfer

The maximum frequency of the measured signal is limited by the maximum transfer rate of the eDMA engine and the number of signals measured simultaneously using the method described in this application note. Consider the other eDMA transfers enabled in the final application as a limiting factor.

4.2. eDMA major loop counter

CITER is limited to 15 bits. Check the CITER register and make sure it does not reach zero. When the CITER register reaches zero, CITER reloads the value from the BITER register and continues to decrement.

5. Conclusion

The eDMA multiple pulse counter implementation can be completed using the internal hardware logic, without the CPU intervention. The CPU only reads the eDMA channel CITER register to get the actual pulse count value.

The figure below shows three test square signals with different frequencies (20 kHz, 10 kHz, and 1 kHz). The frequency of the particular signal was determined using the eDMA module.

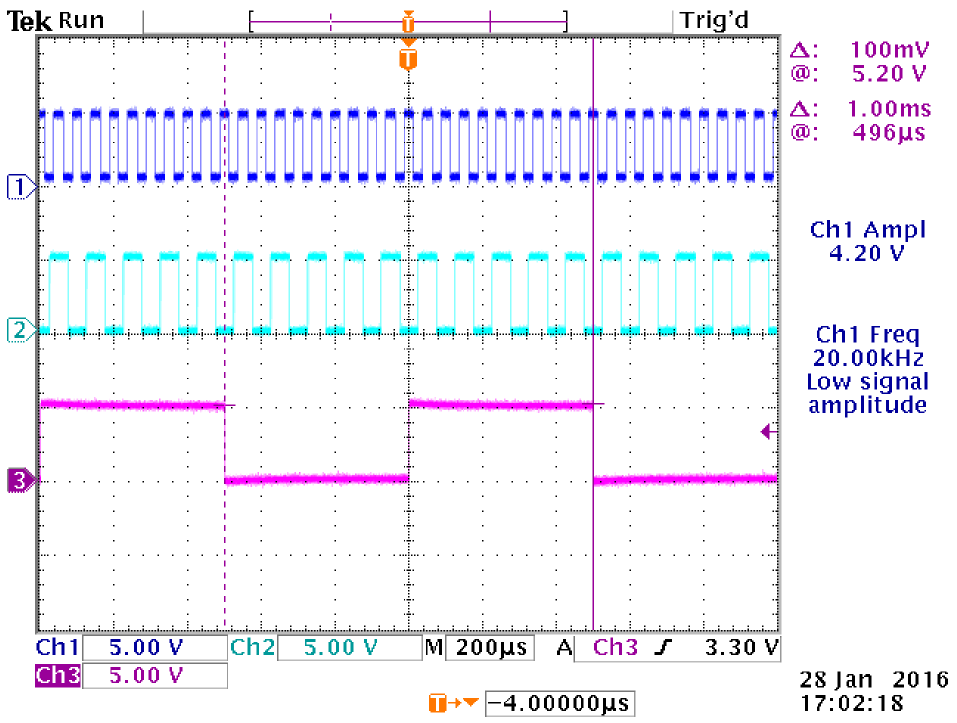
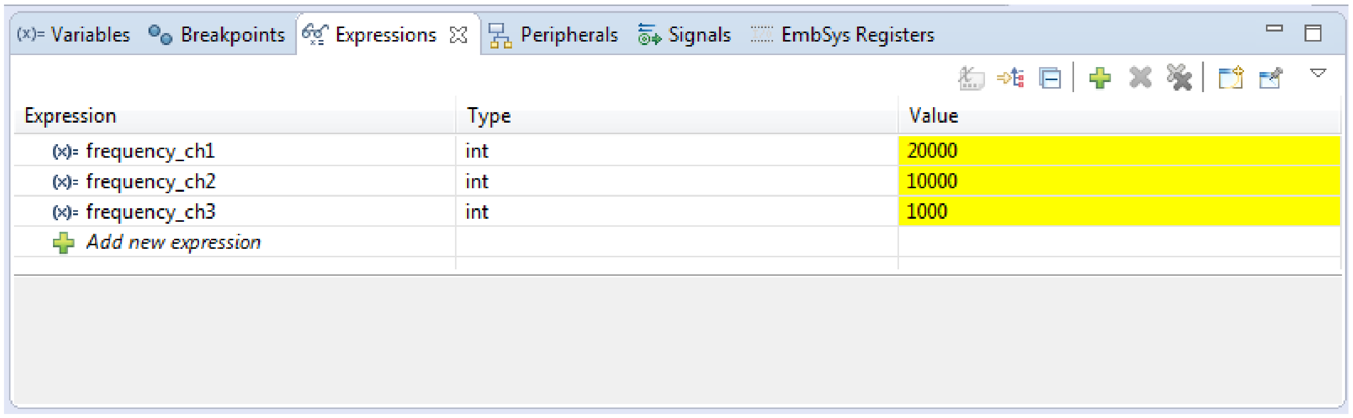


Figure 1. Input square waves

To measure the frequency of the input signals, follow these steps:

1. Enable the PIT timer to generate a one-second interrupt.
2. In the interrupt routine, read the difference between the BITER and CITER registers and reset the eDMA counter.
3. Check the values in the “Expression” tab in S32 Design Studio.



Expression	Type	Value
frequency_ch1	int	20000
frequency_ch2	int	10000
frequency_ch3	int	1000

Figure 2. Frequency of testing square waves in S32 Design Studio

6. Revision History

This table summarizes the changes done to this document since the initial release.

Table 2. Revision history

Revision number	Date	Substantive changes
0	02/2016	Initial release.

Document Number: AN5258
Rev. 0
02/2016