NXP Semiconductors
Application Note

Document Number: AN5413
Rev.5, 12/2020

S32K1xx系列使用指南

软件示例和启动代码来行使微控制器的功能

作者：恩智浦半导体

URL: https://www.nxp.com.cn/docs/en/application-note/AN5413.pdf

1 简介

本应用笔记提供了软件示例，并描述了帮助用户开始使用S32K1xx系列MCU所需的必要启动步骤。

本文档描述了提供的一些示例，完整的源代码、项目和文档可在网页https://source.codeaurora.org/?q=s32support查找， 项目使用NXP的S32 Design Studio v 3.3实施，并在S32K144评估板版本S32K144EVB-Q100上进行测试。要访问项目及其相应文档，请执行以下操作：

- 请参阅第6节以导入此处提到的示例，以及S32K14x和S32K11x的其他示例。
- 每个项目的文件向客户提供模块的说明以及所用功能和或驱动程序的摘要。更多详细信息，请参阅第6节。

目录

这些示例和其他示例也被移植到ARM®KeilTM MDK工具。参见ARMKeil应用笔记304，https://developer.arm.com/documentation/kan304/latest/ 以获取更多信息。

2 软件示例

下表列出了本应用笔记中的示例。这三个Hello World示例旨在作为基础项目，可以复制并添加代码以创建新项目。

表1.示例清单

Example	Programs	Summary
Hello World	hello	Simplest project: • Configure GPIO • Output to LED follows switch input
Hello World + Clocks	hello_clocks	Perform common initialization for clocks and LPIT: • Initialize System Oscillator (SOSC) for 8 MHz crystal • Initialize SPLL with 8 MHz SOSC input to provide 80 MHz clock • Change Normal RUN mode clock from default FIRC to 160 MHz SPLL (before dividers) • Initialize LPIT channel to count 40M clocks (1 second timeout) • Toggle output to LED every LPIT timeout
Hello World + Clocks + Interrupts	hello_clocks_interrupts	The Hello World + Clock example is modified to service the PIT channel timeout with an interrupt service route: • Initialize system clock to 80 MHz • Initialize an LPIT channel for 1 second timeout and enable its interrupt • Wait forever • At LPIT timeout interrupt, toggle output to LED
DMA	eDMA	Transfer a string of bytes to a single byte location: • Initialize a Transfer Control Descriptor (TCD) • Use software (instead of peripheral DMA requests) to initiate transfers
Timed I/O (FTM)	FTM	Perform common timed I/O functions with FTM: • Module counter initialization • Pulse Width MOdulation • Output Compare • Input Capture
ADC - SW Trigger	ADC	Perform simple analog to digital conversions using software trigger: • Initialize ADC for SW trigger, continuous mode • Loop: - Convert channel connected to pot on evaluation board - Scale result to 0 to 5000 mV - Light evaluation board LEDs to reflect voltage level - Convert channel connected to the ADC high reference voltage
UART	UART	Transmit and receive characters: • Initialize UART for 9600 baud, 1 stop, no parity • Loop: - Transmit string, then a prompt character on new line - When character is received, echo it back

Table 1. List of examples

Example	Programs	Summary
SPI	LPSPI	Transmit and receive a SPI frame: • Initialize LPSPI for 1M Baud, PCS3 which is connected to SPC on EVB • Wait for Tx FIFO to have at least one available slot then issue transmit • Wait for Rx FIFO to have at least one received frame then read data
CAN 2.0	FlexCAN	Transmit and receive an eight byte CAN 2.0 message at 500 KHz: • Initialize FlexCAN and Message Buffer 4 to receive a message • Transmit one frame using Message Buffer 0 • Loop: - If Message Buffer 4 received message flag is set, read message - If Message Buffer 0 transmit done flag is set, transmit another message
CAN FD	FlexCAN_FD	Transmit and receive a 64 byte CAN FD message at 500 KHz and 1 or 2 MHz: • Initialize FlexCAN and Message Buffer 4 to receive a message • Transmit one frame using Message Buffer 0 Loop: - If Message Buffer 4 received message flag is set, read message - If Message Buffer 0 transmit done flag is set, transmit another message

如果使用S32K148EVB，则应用笔记示例中使用的许多I/O端口是不同的。汇总见表2。

表 2. EVB的应用笔记示例I/O端口差异摘要

I/O	S32K144 EVB	S32K148 EVB
Blue LED	PTD0	***PTE21***
Green LED	PTD16	***PTE22***
Red LED	PTD15	***PTE23***
BTN 0	PTC12 (SW2)	PTC12 (SW3)
FTM0 Channel 0[1]	PTD15 (Red LED)	PTD15 ***(no LED)***
FTM0 Channel 1	PTD16 (Green LED)	PTD16 ***(no LED)***
FTM0 Channel 6	PTE8	PTE8
Potentiometer to ADC	PTC14 (ADC Channel 12)	***PTC28 (ADC channel 28)***
UART1_RX	PTC6	PTC6
UART1_TX	PTC7	PTC7
LPSPI1_SOUT	PTB16	***PTA27***
LPSPI1_SIN	PTB15	***PTA29***
LPSPI1_PCSx	PTB17 PCS3 (to UJA1169)	***PTA26 PCS0 (to UJA1132)***
LPSPI1_SCK	PTB14	***PTA28***
CAN0_TX	PTE5	PTE5
CAN0_RX	PTE4	PTE4

[1] For LED connections to FTM example with PWM outputs, use FTM4 instead of FTM0.

2.1 Hello World

2.1.1 描述

总结： 这个简短的项目是学习GPIO的起点。轮询输入以检测高电平或低电平。根据输入状态设置输出。如果在S32K14x评估板上运行代码，按下按钮0将点亮蓝色LED，如下图所示。

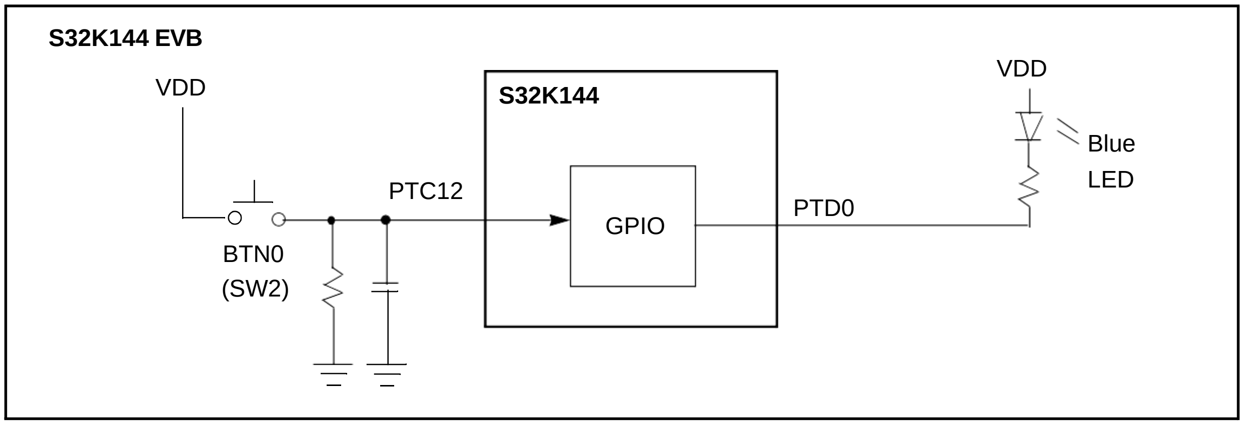


图 1. Hello World 方框图

2.1.2 设计

- 主函数之前的初始化：
  - — 定义中断地址（如复位处理器）和闪存配置
  - — 初始化堆栈指针、寄存器
  - — 禁用看门狗（如果已配置）
  - — 初始化向量表
  - — 将变量从ROM复制到RAM，并将数据段（.bss）归零
  - — 解除屏蔽中断
- 禁用看门狗
- 启用GPIO模块时钟并配置GPIO端口：
  - — PTC12:GPIO输入（转到评估板上的BTN 0）
  - — PTD0:GPIO输出（转到蓝色LED）
- 环路：
  - — 如果按下BTN0（输入=1）
    - – 打开LED（输出=0）
  - — 其他（输入=0）
    - – 关闭LED（输出=1）

2.1.3 代码

2.1.3.1 hello.c

```
#include "S32K144.h"    /* include peripheral declarations S32K144 */

#define PTD0  0         /* Port PTD0, bit 0: FRDM EVB output to blue LED */
#define PTC12 12        /* Port PTC12, bit 12: FRDM EVB input from BTN0 [SW2] */

void WDOG_disable (void){
  WDOG->CNT=0xD928C520;    /*Unlock watchdog*/
  WDOG->TOVAL=0x0000FFFF;  /*Maximum timeout value*/
  WDOG->CS = 0x00002100;   /*Disable watchdog*/
}

int main(void)
  { int counter =
  0;
  WDOG_disable();
                              /* Enable clocks to peripherals (PORT modules) */
  PCC-> PCCn[PCC_PORTC_INDEX] = PCC_PCCn_CGC_MASK; /* Enable clock to PORT C */
  PCC-> PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK; /* Enable clock to PORT D */
                               /* Configure port C12 as GPIO input (BTN 0 [SW2] on EVB) */
  PTC->PDDR &= ~(1<<PTC12);    /* Port C12: Data Direction= input (default) */
  PORTC->PCR[12] = 0x00000110; /* Port C12: MUX = GPIO, input filter enabled */
                               /* Configure port D0 as GPIO output (LED on EVB) */
  PTD->PDDR |= 1<<PTD0;        /* Port D0: Data Direction= output */
  PORTD->PCR[0] = 0x00000100;  /* Port D0: MUX = GPIO */

  for(;;) {
    if (PTC->PDIR & (1<<PTC12)) {   /* If Pad Data Input = 1 (BTN0 [SW2] pushed) */
      PTD-> PCOR |= 1<<PTD0;        /* Clear Output on port D0 (LED on) */
    }
    else {                          /* If BTN0 was not pushed */
      PTD-> PSOR |= 1<<PTD0;        /* Set Output on port D0 (LED off) */
    }
    counter++;
}
```

2.2 Hello World + 时钟

2.2.1 描述

概要：该项目提供了时钟的通用初始化和LPIT通道计数器功能。核心时钟设置为80 MHz。LPIT0通道0被配置为计数一秒的SPLL时钟。软件轮询通道的超时标志，并在标志设置时将GPIO输出切换到LED。

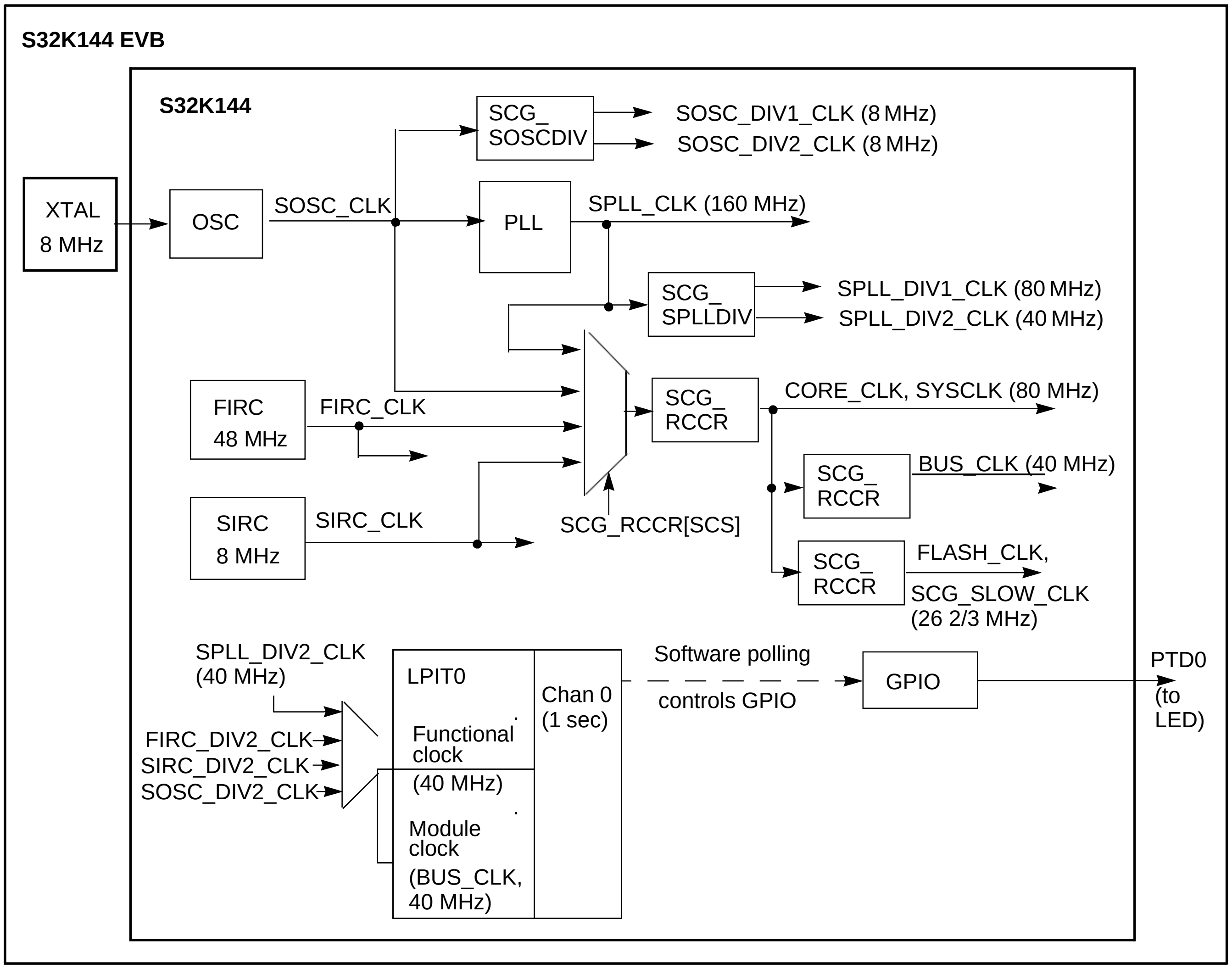


图 2. Hello World + Clocks 方框图

表 3.时钟、分频器要求和常见配置

Clock	Divider Requirements	Slow RUN (typically with FIRC)	Normal RUN (with SPLL)	High Speed RUN (with SPLL)	Very Low Power RUN, VLPR (with SIRC or SOSC)
CORE_CLK, SYS_CLK	DIVCORE: • >= BUS_CLK	48 MHz	80 MHz (max)	112 MHz (max)	4 MHz
BUS_CLK	DIVBUS: • CORE_CLK divided by integer	48 MHz	40 MHz (max)	56 MHz (max)	4 MHz
FLASH_CLK, SCG_SLOW_CLK	DIVSLOW: • CORE_CLK divided by integer • CORE_CLK / FLASH_CLK= 8 max.	24 MHz	26 2/3 MHz (max)	28 MHz (max)	1 MHz

表2总结了时钟频率和关系。在切换新时钟之前，必须初始化核心时钟、系统时钟、总线时钟和闪存时钟的分频器。在时钟开关接通之前，分频器不会有新的值。

2.2.2 设计

- 初始化端口引脚：
  — 启用端口D模块的时钟
  — PTD0:GPIO输出（转到蓝色LED）
- 为8 MHz晶体初始化系统振荡器（SOSC）
  — 初始化所需的SOSC分配器
  — 配置范围、高增益、参考
  — 确保SOSC控制和状态寄存器已解锁
  — 在SOSC控制和状态寄存器中启用SOSC
  — 等待SOSC生效
- 使用8 MHz SOSC将系统PLL（SPLL）初始化为160 MHz
  — 确保禁用SPLL以允许配置
  — 初始化所需的SPLL除法器
  — 初始化PLL参考时钟分频器和系统PLL乘法器[1]
    – Fpll=Fosc/PLL参考时钟分频器x系统PLL乘法器/2=8 MHz/1 x 20/2=160 MHz
  — 确保SPLL控制和状态寄存器已解锁
  — 在SPLL控制和状态寄存器中启用SPLL
  — 等待SPLL生效
- 初始化LPIT0通道0：
  — 启用SPLL_DIV2_时钟的时钟源
  — 启用时钟至LPIT0寄存器

1. VCO output frequency, has a minimum of 180 MHz and maximum of 320 MHz per S32K1xx Data Sheet rev. 1 0

— 启用LPIT0模块
— 初始化通道0：
  – 超时=1秒时钟
  – 将模式设置为32位计数器并启用通道0
- 将正常运行模式时钟更改为SPLL
— 为新目标时钟频率初始化内核、总线和闪存的时钟分频器
— 开关系统时钟输入至SPLL（分频器前160 MHz）
- 环路：
— 等待LPIT0通道0标志
— 增量计数器，切换PTD0 GPIO输出并清除通道标志

2.2.3 代码

2.2.3.1 hello_clocks.c

```
#include "S32K144.h"            /* include peripheral declarations S32K144 */
#include "clocks_and_modes.h"
int lpit0_ch0_flag_counter = 0; /* LPIT0 timeout counter */

void PORT_init (void) {
  PCC-> PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK; /* Enable clock for PORT D */
  PTD->PDDR |= 1<<0;            /* Port D0:  Data Direction= output */
  PORTD->PCR[0] =  0x00000100;  /* Port D0:  MUX = ALT1, GPIO (to blue LED on EVB) */
}
void LPIT0_init (void) {
  PCC->PCCn[PCC_LPIT_INDEX] = PCC_PCCn_PCS(6);    /* Clock Src = 6 (SPLL2_DIV2_CLK)*/
  PCC->PCCn[PCC_LPIT_INDEX] |= PCC_PCCn_CGC_MASK; /* Enable clk to LPIT0 regs */
  LPIT0->MCR = 0x00000001;    /* DBG_EN-0: Timer chans stop in Debug mode */
                              /* DOZE_EN=0: Timer chans are stopped in DOZE mode */
                              /* SW_RST=0: SW reset does not reset timer chans, regs */
                           /* M_CEN=1: enable module clk (allows writing other LPIT0 regs)*/
  LPIT0->TMR[0].TVAL = 40000000;    /* Chan 0 Timeout period: 40M clocks */
  LPIT0->TMR[0].TCTRL = 0x00000001; /* T_EN=1: Timer channel is enabled */
                              /* CHAIN=0: channel chaining is disabled */
                              /* MODE=0: 32 periodic counter mode */
                              /* TSOT=0: Timer decrements immediately based on restart */
                              /* TSOI=0: Timer does not stop after timeout */
                              /* TROT=0 Timer will not reload on trigger */
                              /* TRG_SRC=0: External trigger source */
                              /* TRG_SEL=0: Timer chan 0 trigger source is selected*/
}
```

```
void WDOG_disable (void){
  WDOG->CNT=0xD928C520;    /*Unlock watchdog*/
  WDOG->TOVAL=0x0000FFFF;  /*Maximum timeout value*/
  WDOG->CS = 0x00002100;   /*Disable watchdog*/
}

int main(void)
  {  WDOG_disable();
  PORT_init();             /* Configure ports */
  SOSC_init_8MHz();        /* Initialize system oscillator for 8 MHz xtal */
  SPLL_init_160MHz();       /* Initialize sysclk to 160 MHz with 8 MHz SOSC */
  NormalRUNmode_80MHz();   /* Init clocks: 80 MHz sysclk & core, 40 MHz bus, 20 MHz flash */
  LPIT0_init();            /* Initialize PIT0 for 1 second timeout  */

  for (;;) {                        /* Toggle output to LED every LPIT0 timeout */
    while (0 == (LPIT0->MSR & LPIT_MSR_TIF0_MASK)) {} /* Wait for LPIT0 CH0 Flag */
    lpit0_ch0_flag_counter++;         /* Increment LPIT0 timeout counter */
    PTD->PTOR |= 1<<0;                /* Toggle output on port D0 (blue LED) */
    LPIT0->MSR |= LPIT_MSR_TIF0_MASK; /* Clear LPIT0 timer flag 0 */
}
```

2.2.3.2 clocks_and_modes.c

```
#include "S32K144.h"           /* include peripheral declarations S32K144 */
#include "clocks_and_modes.h"

void SOSC_init_8MHz(void) {
  SCG->SOSCDIV=0x00000101;  /* SOSCDIV1 & SOSCDIV2 =1: divide by 1 */
  SCG->SOSCCFG=0x00000024;  /* Range=2: Medium freq (SOSC between 1MHz-8MHz)*/
                            /* HGO=0:   Config xtal osc for low power */
                            /* EREFS=1: Input is external XTAL */
  while(SCG->SOSCCSR & SCG_SOSCCSR_LK_MASK); /* Ensure SOSCCSR unlocked */
  SCG->SOSCCSR=0x00000001;  /* LK=0:          SOSCCSR can be written */
                            /* SOSCCMRE=0:    OSC CLK monitor IRQ if enabled */
                            /* SOSCCM=0:      OSC CLK monitor disabled */
                            /* SOSCERCLKEN=0: Sys OSC 3V ERCLK output clk disabled */
                            /* SOSCLPEN=0:    Sys OSC disabled in VLP modes */
                            /* SOSCSTEN=0:    Sys OSC disabled in Stop modes */
                            /* SOSCEN=1:      Enable oscillator */
  while(!(SCG->SOSCCSR & SCG_SOSCCSR_SOSCVLD_MASK)); /* Wait for sys OSC clk valid */
}
void SPLL_init_160MHz(void) {
  while(SCG->SPLLCSR & SCG_SPLLCSR_LK_MASK); /* Ensure SPLLCSR unlocked */
  SCG->SPLLCSR = 0x00000000;  /* SPLLEN=0: SPLL is disabled (default) */
  SCG->SPLLDIV = 0x00000302;  /* SPLLDIV1 divide by 2; SPLLDIV2 divide by 4 */
  SCG->SPLLCFG = 0x00180000;  /* PREDIV=0: Divide SOSC_CLK by 0+1=1 */
                              /* MULT=24:  Multiply sys pll by 4+24=40 */
                              /* SPLL_CLK = 8MHz / 1 * 40 / 2 = 160 MHz */
  while(SCG->SPLLCSR & SCG_SPLLCSR_LK_MASK); /* Ensure SPLLCSR unlocked */
  SCG->SPLLCSR = 0x00000001; /* LK=0: SPLLCSR can be written */
                             /* SPLLCMRE=0:  SPLL CLK monitor IRQ if enabled */
                             /* SPLLCM=0:    SPLL CLK monitor disabled */
                             /* SPLLSTEN=0:  SPLL disabled in Stop modes */
                             /* SPLLEN=1:    Enable SPLL */
  while(!(SCG->SPLLCSR & SCG_SPLLCSR_SPLLVLD_MASK)); /* Wait for SPLL valid */
}
void NormalRUNmode_80MHz (void) {  /* Change to normal RUN mode with 8MHz SOSC, 80 MHz PLL*/
  SCG->RCCR=SCG_RCCR_SCS(6)      /* PLL as clock source*/
    |SCG_RCCR_DIVCORE(0b01)      /* DIVCORE=1, div. by 2: Core clock = 160/2 MHz = 80 MHz*/
    |SCG_RCCR_DIVBUS(0b01)       /* DIVBUS=1, div. by 2: bus clock = 40 MHz*/
    |SCG_RCCR_DIVSLOW(0b10);     /* DIVSLOW=2, div. by 3: SCG slow, flash clock= 26 2/3 MHz*/
  while (((SCG->CSR & SCG_CSR_SCS_MASK) >> SCG_CSR_SCS_SHIFT ) != 6) {}
                                 /* Wait for sys clk src = SPLL */
}
```

2.3 Hello World +中断

2.3.1 描述

小结：这个项目与前一个项目相同，只是实现了一个中断来处理计时器标志。中断处理程序清除该标志并切换输出，而不是软件轮询计时器标志。对于LPIT0计时器时钟，使用SPLL_DIV2_CLK再次超时1秒。

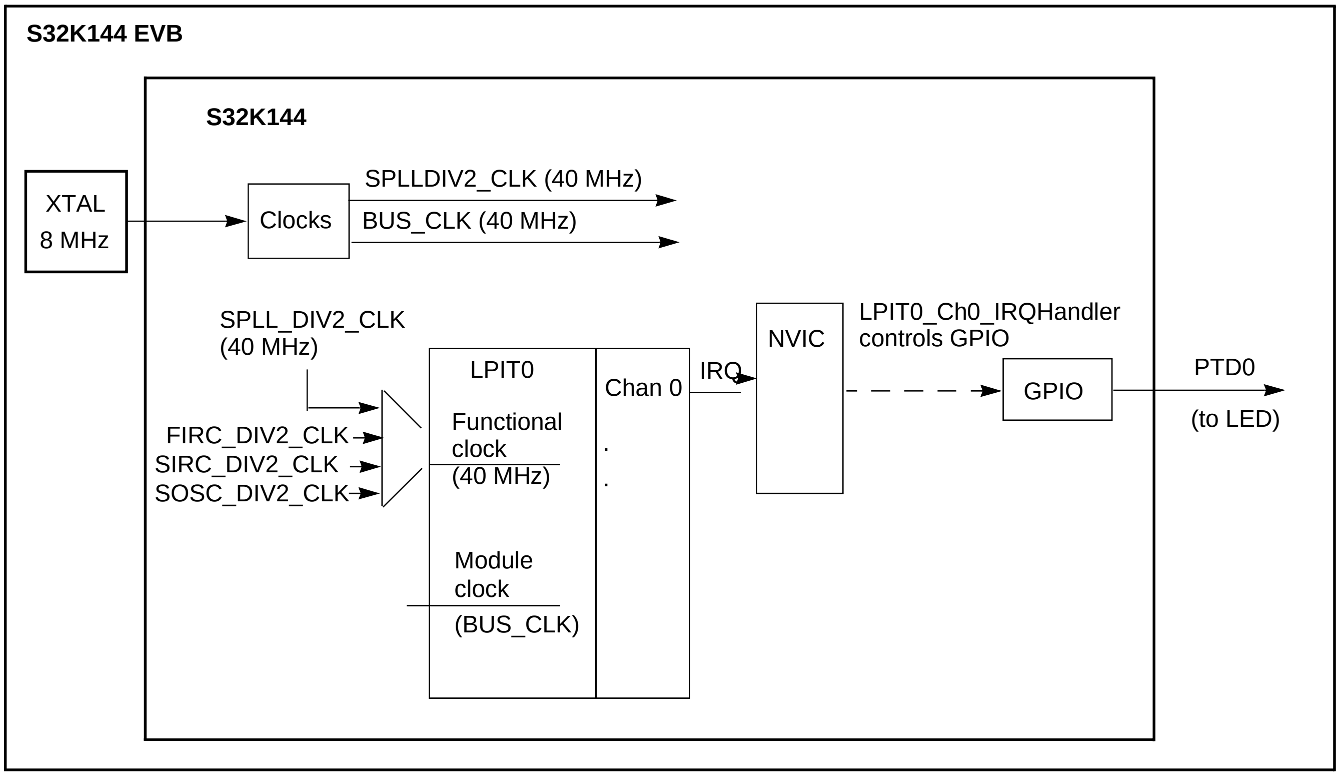


图3.Hello World+中断框图

在本例中，要初始化中断，需要对NVIC进行三次写入：

- 清除任何先前挂起的中断（如果有）
  — 向中断清除挂起寄存器（ICPR）中的中断位写入1
- 启用所需的中断#
  — 将1写入中断集启用寄存器（ISER）中的中断位
- 设置中断的优先级
  — 将优先级从0到15写入相应的中断优先级寄存器（IP）

下表显示了复位时内核使用的向量。1至15是内核使用。15及以上是外围设备、DMA和软件中断使用。

表4.S32K144矢量表（使用S32 Design Studio的示例）

Address	**Contents in Big Endian memory view**	**Contents as Little Endian format addresses**[1]	**Vector #**	**IRQ # (NVIC interrupt source)**	**Symbol in file startup_S32K144.S, section .isr_vector, __isr_vector**	**Description**
0x0000 0000	0070 0020	2000 7000	0		__StackTop	Initial stack pointer
0x0000 0004	1104 0000	0000 4010	1		Reset_Handler	Initial Program Counter
0x0000 0008	4D04 0000	0000 044C	2		NMI_Handler	Non-maskable IRQ (NMI) Vector
0x0000 000C	4D04 0000	0000 044C	3		HardFault_Handler	Hard Fault Vector
0x0000 0010	4D04 0000	0000 044C	4		MemManage_Handler	MemManage Fault Vector
0x0000 0014	4D04 0000	0000 044C	5		BusFault_Handler	Bus Fault Vector
0x0000 0018	4D04 0000	0000 044C	6		UsageFault_Handler	Usage Fault Vector
0x0000 001C	0000 0000	0000 0000	7		0	- Not Used -
0x0000 0020	0000 0000	0000 0000	8		0	- Not Used -
0x0000 0024	0000 0000	0000 0000	9		0	- Not Used -
0x0000 0028	0000 0000	0000 0000	10		0	- Not Used -
0x0000 002C	4D04 0000	0000 044C	11		SVC_Handler	Supervisor call (SVC) Vector
0x0000 0030	4D04 0000	0000 044C	12		DebugMon_Handler	Debug Monitor
0x0000 0034	0000 0000	0000 0000	13		0	- Not Used -
0x0000 0038	4D04 0000	0000 044C	14		PendSV_Handler	Pendable request for system service (PendableSrvReq) Vector
0x0000 003C	4D04 0000	0000 044C	15		SysTick_Handler	Sys tick timer (Sys Tick) Vector
0x0000 0040	4D04 0000	0000 044C	16	0	DMA0_IRQHandler	Interrupt # 0 Vector: DMA Channel 0 transfer complete
0x0000 0044	4D04 0000	0000 044C	17	1	DMA1_IRQHandler	Interrupt # 1 Vector DMA Channel 0 transfer complete
0x0000 0048	4D04 0000	0000 044C	18	2	DMA2_IRQHandler	Interrupt # 2 Vector DMA Channel 0 transfer complete
etc.						
0X0000 0100	5906 0000	0000 0658	64	48	`LPIT0_Ch0_IRQHandler`	Interrupt #48 Vector: LPIT0 Ch. 0
etc. for the rest of the vectors.						

[1] Big Endian addresses have the least significant bit =1 for ARM™ Thumb architecture. For ease of reading vector addresses, this bit has been set to zero in this column.

2.3.2 设计

- 初始化端口引脚：
  - — 启用端口D模块的时钟
  - — PTD0:GPIO输出（转到蓝色LED）
- 为8 MHz晶体初始化系统振荡器（SOSC）：
  - — 初始化所需的SOSC分配器
  - — 配置范围、高增益、参考
  - — 确保SOSC控制和状态寄存器已解锁
  - — 在SOSC控制和状态寄存器中启用SOSC
  - — 等待SOSC生效
- 使用8 MHz SOSC将系统PLL（SPLL）初始化为160 MHz：
  - — 确保禁用SPLL以允许配置
  - — 初始化所需的SPLL除法器
  - — 初始化PLL参考时钟分频器和系统PLL乘法器1
    - – Fpll=Fosc/PLL参考时钟分频器x系统PLL乘法器/2=8 MHz/1 x 40/2=160 MHz
  - — 确保SPLL控制和状态寄存器已解锁
  - — 在SPLL控制和状态寄存器中启用SPLL
  - — 等待SPLL生效
- 初始化LPIT0通道0：
  - — 启用SPLL_DIV2_时钟的时钟源
  - — 启用时钟至LPIT0寄存器
  - — 启用LPIT0模块
  - — 初始化通道0：
    - – 启用通道的中断
    - – 超时=1秒时钟
    - – 将模式设置为32位计数器并启用通道0
- 将正常运行模式时钟更改为SPLL：
  - — 为新目标时钟频率初始化内核、总线和闪存的时钟分频器
  - — 开关系统时钟输入至SPLL（分频器前160 MHz）
- 循环：永远等待
- LPIT_0通道0中断处理器：
  - — 清除通道标志2
  - — 增量计数器
  - — 切换PTD0 GPIO输出

1.VCO output frequency, has a minimum of 180 MHz and maximum of 320 MHz per S32K1xx Data Sheet rev. 1 0

2.To ensure interrupt flag clears before routine exit, perform a memory read-after-write such as ctr++. Refererence: S32K14x Series Reference Manual, Rev. 1, 08/2016, section 3.4.1 and http://www.keil.com/support/docs/3928.htm

2.3.3 代码

2.3.3.1 hello_interrupts.c

```
#include "S32K144.h"          /* include peripheral declarations S32K144 */
#include "clocks_and_modes.h"

int idle_counter = 0;           /* main loop idle counter */
int lpit0_ch0_flag_counter = 0; /* LPIT0 chan 0 timeout counter */

void NVIC_init_IRQs (void) {
  FSL_NVIC->ICPR[1] = 1 << (48 % 32);  /* IRQ48-LPIT0 ch0: clr any pending IRQ*/
  FSL_NVIC->ISER[1] = 1 << (48 % 32);  /* IRQ48-LPIT0 ch0: enable IRQ */
  FSL_NVIC->IP[48] =0x0A;              /* IRQ48-LPIT0 ch0: priority 10 of 0-15*/
}
void PORT_init (void) {
  PCC-> PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK; /* Enable clock for PORT D */
  PTD->PDDR |= 1<<0;            /* Port D0:  Data Direction= output */
  PORTD->PCR[0] =  0x00000100;  /* Port D0:  MUX = ALT1, GPIO (to blue LED on EVB) */
}
void LPIT0_init (void) {
  PCC->PCCn[PCC_LPIT0_INDEX] = PCC_PCCn_PCS(6);    /* Clock Src = 6 (SPLL2_DIV2_CLK)*/
  PCC->PCCn[PCC_LPIT0_INDEX] |= PCC_PCCn_CGC_MASK; /* Enable clk to LPIT0 regs */
  LPIT0->MCR = 0x00000001;    /* DBG_EN-0: Timer chans stop in Debug mode */
                              /* DOZE_EN=0: Timer chans are stopped in DOZE mode */
                              /* SW_RST=0: SW reset does not reset timer chans, regs */
                            /* M_CEN=1: enable module clk (allow writing other LPIT0 regs) */
  LPIT0->MIER = 0x00000001;   /* TIE0=1: Timer Interrupt Enabled fot Chan 0 */
  LPIT0->TVAL0 = 80000000;    /* Chan 0 Timeout period: 80M clocks */
  LPIT0->TCTRL0 = 0x00000001; /* T_EN=1: Timer channel is enabled */
                              /* CHAIN=0: channel chaining is disabled */
                              /* MODE=0: 32 periodic counter mode */
                              /* TSOT=0: Timer decrements immediately based on restart */
                              /* TSOI=0: Timer does not stop after timeout */
                              /* TROT=0 Timer will not reload on trigger */
                              /* TRG_SRC=0: External trigger source */
                              /* TRG_SEL=0: Timer chan 0 trigger source is selected*/
}
void WDOG_disable (void){
  WDOG->CNT=0xD928C520;    /*Unlock watchdog*/
  WDOG->TOVAL=0x0000FFFF; /*Maximum timeout value*/
  WDOG->CS = 0x00002100;    /*Disable watchdog*/
}
```

```
int main(void)
  {  WDOG_disable();
  PORT_init();              /* Configure ports */
  SOSC_init_8MHz();         /* Initialize system oscillator for 8 MHz xtal */
  SPLL_init_160MHz();        /* Initialize SPLL to 160 MHz with 8 MHz SOSC */
  NormalRUNmode_80MHz();    /* Init clocks: 80 MHz sysclk & core, 40 MHz bus, 20 MHz flash */
  NVIC_init_IRQs();         /* Enable desired interrupts and priorities */
  LPIT0_init();             /* Initialize PIT0 for 1 second timeout   */

  for (;;)
    { idle_counter++;
  }
}

void LPIT0_Ch0_IRQHandler (void) {
  LPIT0->MSR |= LPIT_MSR_TIF0_MASK; /* Clear LPIT0 timer flag 0 */
          /* Perform read-after-write to ensure flag clears before ISR exit */
  lpit0_ch0_flag_counter++;         /* Increment LPIT0 timeout counter */
  PTD->PTOR |= 1<<0;                /* Toggle output on port D0 (blue LED) */
}
```

2.3.3.2 clocks_and_modes.c

See code in clocks_and_modes.c of the Hello World + Clock example.

2.4 DMA

2.4.1 描述

摘要：初始化eDMA通道的传输控制描述符（TCD），以将字节字符串（“Hello world”）从SRAM中的阵列传输到单个SRAM字节位置。这模拟了DMA的一种常见用法，即在DMA控制下，一串数据或命令自动传输到外围设备的输入寄存器。本示例旨在说明如何设置DMA传输。

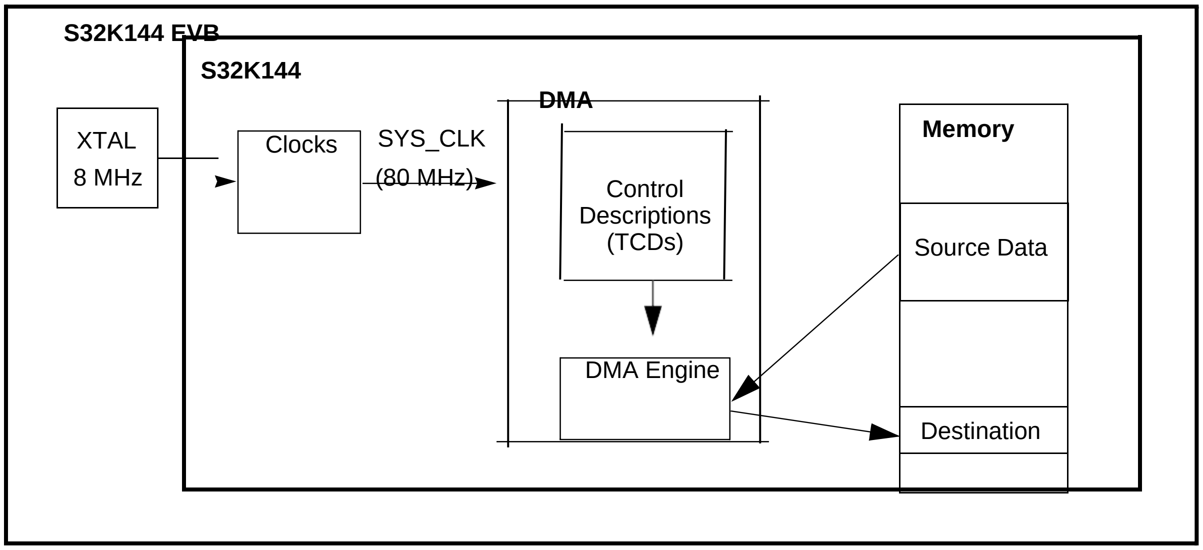


图4.DMA示例框图

将DMA与外围设备一起使用时，需要DMA MUX。参考手册附带的电子表格S32K1xx_DMA_Interrupt_mapping，DMA_Chu MUX表中列出了映射到DMA信道的外围设备分配号。对于本例中的软件触发传输，不需要DMA MUX。

本教程级示例中未使用的中断，在所有所需传输或其中一半传输完成时非常有用。一种使用情形是在转换完成后让类似于外围设备的ADC生成DMA请求。DMA控制器可以自动将转换结果传输到SRAM。经过所需数量的转换后，DMA控制器可以生成该通道的中断请求。

信道链接和散射-聚集（SGA）是高级功能，可使DMA请求允许在每个DMA请求上进行多个不同的传输，或为每个DMA请求使用不同的TCD。这些强大的功能可与外围设备一起使用，以实现状态机类型的子系统。示例：输入信号生成DMA请求，该请求传输数据以初始化多个外围设备。微循环映射通常不在MCU级应用程序中使用，但对于图形而言，它可以以45度为增量旋转图像。

因为本例中不涉及外设，所以不会发生自动DMA握手。相反，这里给出的软件握手必须针对每个DMA请求实施（小循环传输）：

- 启动DMA服务请求（为所需通道设置启动位）。

- 通过轮询启动和活动状态，等待小回路传输完成。
- 重复上述两个步骤，直到主回路完成，如完成位所示

对于每次传输，这些步骤都显得“复杂”，在本例中，这只是一个字节。然而，当使用实际的外围设备时，软件永远不必执行这些步骤；它们是由硬件自动完成的。

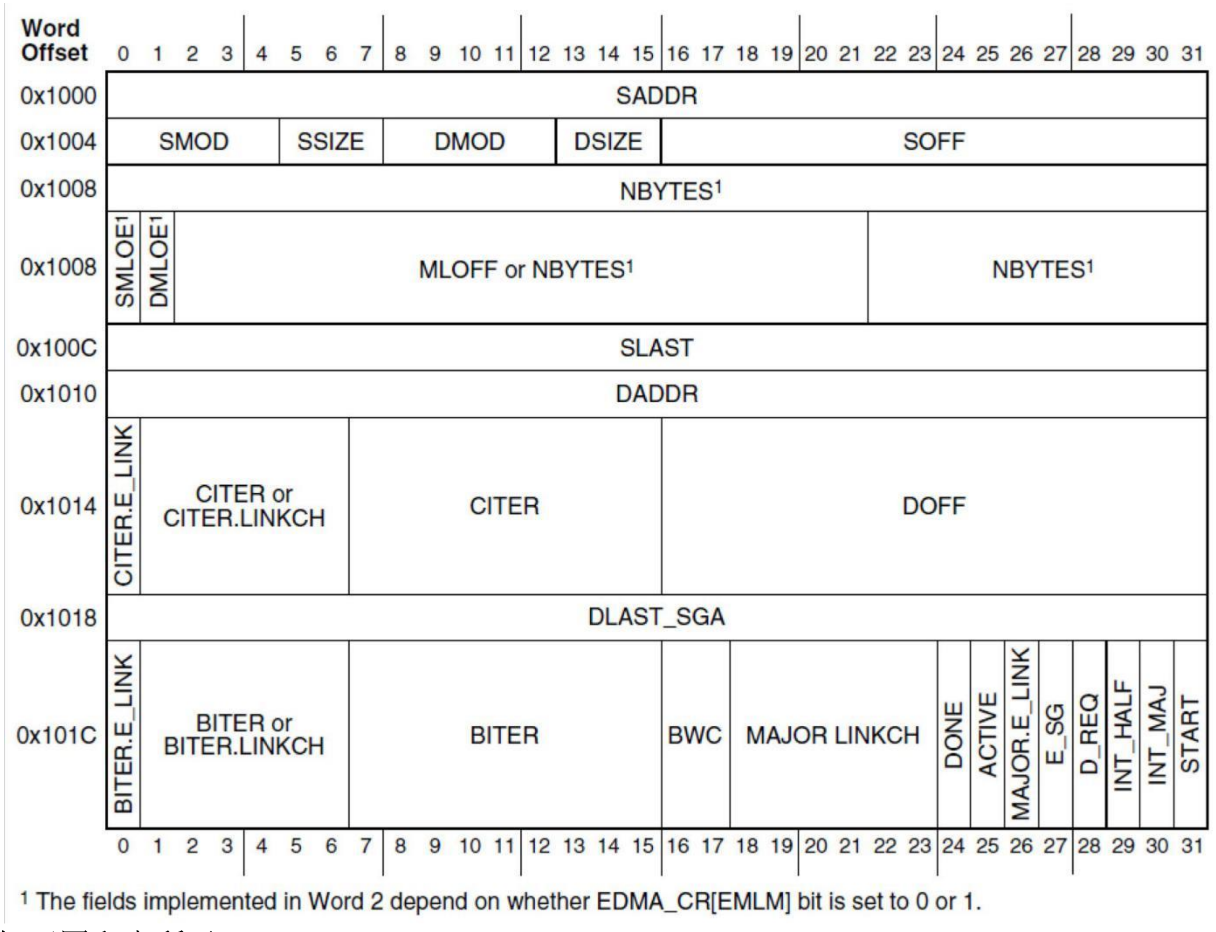
启动位通常由请求服务的外围设备通过硬件设置。DMA处理引擎激活通道后，设置活动位。如果DMA引擎忙于维护其他通道，可以通过清除起始位来取消传输。然后需要检查活动位，以确保服务未在该通道上启动。

作为练习，可以修改TCD，使目标是数组而不是单字节位置(提示：将目标声明为字符串并更改DOFF=1。）

2.4.2 设计

2.4.2.1 DMA传输控制描述符（TCD）

TCD描述要传输的数据以及如何控制传输。每个TCD占用eDMA结构内部RAM中的八个32位



字，如下图和表所示。

图5.传输控制描述符

Table 5. NXP S32K144头文件的TCD初始化

TCD Field	Option	Initialization for TCD*n* with field value *x*		
SADDR	—	DMA->TCD[*n*].SADDR	=	DMA_TCD_SADDR_SADDR(x)
SOFF	—	DMA->TCD[*n*].SOFF	=	DMA_TCD_SOFF_SOFF(x)

SMOD
SSIZE
DMOD
DSIZE
	—	DMA->TCD[*n*].ATTR.SMOD
		DMA->TCD[*n*].ATTR.SSIZE
		DMA->TCD[*n*].ATTR.DMOD
		DMA->TCD[*n*].ATTR.DSIZE
			=
			=
			=
			=
				DMA_TCD_ATTR_SMOD(x)
				DMA_TCD_ATTR_SSIZE(x)
				DMA_TCD_ATTR_DMOD(x)
				DMA_TCD_ATTR_DSIZE(x)

SMLOE DMLOE MLOFF NBYTES	Minor Loop mapping disabled	DMA->TCD[*n*].NBYTES.MLNO	=	DMA_TCD_NBYTES_MLNO_NBYTES(*x*)

	Minor Loop mapping enabled and Offset disabled	DMA->TCD[*n*].NBYTES.MLOFFNO
		DMA->TCD[*n*].NBYTES.MLOFFNO
		DMA->TCD[*n*].NBYTES.MLOFFNO
			=
			=
			=
				DMA_TCD_NBYTES_MLOFFNO_SMLOE(x)
				DMA_TCD_NBYTES_MLOFFNO_DMLOE(x)
				DMA_TCD_NBYTES_MLOFFNO_NBYTES(x)

	Minor Loop mapping enabled and Offset enabled	DMA->TCD[*n*].NBYTES.MLOFFYES
		DMA->TCD[*n*].NBYTES.MLOFFYES
		DMA->TCD[*n*].NBYTES.MLOFFYES
		DMA->TCD[*n*].NBYTES.MLOFFYES
			=
			=
			=
			=
				DMA_TCD_NBYTES_MLOFFYES_SMLOE(x)
				DMA_TCD_NBYTES_MLOFFYES_DMLOE(x)
				DMA_TCD_NBYTES_MLOFFYES_MLOFF(x)
				DMA_TCD_NBYTES_MLOFFYES_NBYTES(x)

SLAST	—	DMA->TCD[*n*].SLAST	=	DMA_TCD_SLAST_SLAST(x)
DADDR	—	DMA->TCD[*n*].DADDR	=	DMA_TCD_DADDR_DADDR(x)
DOFF	—	DMA->TCD[*n*].DOFF	=	DMA_TCD_DOFF_DOFF(x)

CITER
CITER.LINKCH
CITER.E.LINK
	Channel Linking disabled	DMA->TCD[*n*].CITER.ELINKNO
		DMA->TCD[*n*].CITER.ELINKNO
			=
			=
				DMA_TCD_CITER_ELINKNO_CITER(x)
				DMA_TCD_CITER_ELINKNO_ELINK(x)

	Channel Linking enabled	DMA->TCD[*n*].CITER.ELINKYES
		DMA->TCD[*n*].CITER.ELINKYES
		DMA->TCD[*n*].CITER.ELINKYES
			=
			=
			=
				DMA_TCD_CITER_ELINKYES_CITER(x)
				DMA_TCD_CITER_ELINKYES_LINKCH(x)
				DMA_TCD_CITER_ELINKYES_ELINK(x)

DLAST_SGA	—	DMA->TCD[*n*].DLASTSGA	=	DMA_TCD_DLASTSGA_DLASTSGA(x)

START
INT_MAJ
INT_HALF
D_REQ
E_SG
MAJOR.E.LINK
ACTIVE
DONE
MAJOR.LINKCH
BWC
	—	DMA->TCD[*n*].CSR
		DMA->TCD[*n*].CSR
		DMA->TCD[*n*].CSR
		DMA->TCD[*n*].CSR
		DMA->TCD[*n*].CSR
		DMA->TCD[*n*].CSR
		DMA->TCD[*n*].CSR
		DMA->TCD[*n*].CSR
		DMA->TCD[*n*].CSR
		DMA->TCD[*n*].CSR
			=
			=
			=
			=
			=
			=
			=
			=
			=
			=
				DMA_TCD_CSR_START(x)
				DMA_TCD_CSR_INTMAJOR(x)
				DMA_TCD_CSR_INTHALF(x)
				DMA_TCD_CSR_DREQ(x)
				DMA_TCD_CSR_ESG(x)
				DMA_TCD_CSR_MAJORELINK(x)
				DMA_TCD_CSR_ACTIVE(x)
				DMA_TCD_CSR_DONE(x)
				DMA_TCD_CSR_MAJORLINKCH(x)
				DMA_TCD_CSR_BWC(x)

BITER
BITER.LINKCH
BITER.E_LINK
	Channel Linking disabled	DMA->TCD[*n*].BITER.ELINKNO
		DMA->TCD[*n*].BITER.ELINKNO
			=
			=
				DMA_TCD_BITER_ELINKNO_BITER(x)
				DMA_TCD_BITER_ELINKNO_ELINK(x)

	Channel Linking enabled	DMA->TCD[*n*].BITER.ELINKYES
		DMA->TCD[*n*].BITER.ELINKYES
		DMA->TCD[*n*].BITER.ELINKYES
			=
			=
			=
				DMA_TCD_BITER_ELINKYES_BITER(x)
				DMA_TCD_BITER_ELINKYES_LINKCH(x)
				DMA_TCD_BITER_ELINKYES_ELINK(x)

2.4.2.2 设计步骤

- 禁用看门狗
- 系统时钟：初始化SOSC为8 MHz，sysclk为80 MHz，运行模式为80 MHz
- 初始化DMA控制器：
  -启用DMA MUX模块的时钟（如果软件使用起始位启动DMA，则不需要）
  -启用所需的频道(如果软件使用起始位启动DMA，则不需要。）
- 初始化DMA传输控制描述符（此处仅使用TCD0）：
  -来源
  –源地址（SADD）：使用字符串“Hello World”的地址
  –源偏移量（SOFF）：每次传输将源地址增加1字节
  –源模（SMOD）：此处未使用的功能
  –源大小（SSIZE）：一次读取1字节
  –源上次地址调整（SLAST）：在主循环后将源地址减少11
  -目的地
  –目标地址（DADDR）：使用单个字节的地址
  –目的地偏移量（DOFF）：不要在小循环后向目的地地址添加偏移量
  –目的地模数（DMOD）：此处未使用功能
  –目标大小（DSIZE）：一次写入1字节
  –目的地最后地址调整（DLAST）：不在主循环后调整地址
  -每个DMA请求的字节数和迭代次数（小循环）
  –每个DMA请求要传输的字节数（N字节）：一个字节
  –主循环（CITER和BITER）中的迭代次数/次循环次数：11
  –用于小循环后额外迭代的通道到通道链接（BITER ELINK和CITER ELINK）：禁用
  -控制和状态
  –主循环完成后禁用通道（DREQ）：禁用通道
  –在主循环（INTHALF）中途生成中断请求：禁用
  –完成主循环后生成中断请求（INTMARY）：禁用
  –启用散射聚集（ESG）：禁用。没有其他TCD加载到通道
  –在主环路（主链路）后启用信道链路：禁用
  –主回路（主回路链接）后的通道链接编号：空-功能禁用
  –带宽控制（BWC）：设置为0，以便在R/W后没有暂停
  –清除状态标志的初始值（开始、活动、完成）：设置为零
- 开始第一次传输（设置Start=1）并等待传输完成（Start=0，ACTIVE=0）
- 循环：未设置通道的完成状态时：
  -开始下一次传输（设置Start=1）并等待传输完成（Start=0，ACTIVE=0）
- 清除通道的完成状态位

2.4.3 代码

2.4.3.1 main.c

```c
#include "S32K144.h" /* Include peripheral declarations S32K144 */
#include "dma.h"
#include "clocks_and_modes.h"

void WDOG_disable (void){
  WDOG->CNT=0xD928C520;    /* Unlock watchdog */
  WDOG->TOVAL=0x0000FFFF; /* Maximum timeout value */
  WDOG->CS = 0x00002100;     /* Disable watchdog */
}

int main(void)

  {  WDOG_disable();
  SOSC_init_8MHz();         /* Initialize system oscillator for 8 MHz xtal */
  SPLL_init_160MHz();        /* Initialize SPLL to 160 MHz with 8 MHz SOSC */
  NormalRUNmode_80MHz();   /* Init clocks: 80 MHz SPLL & core, 40 MHz bus, 20 MHz flash */

  DMA_init();               /* Init DMA controller */
  DMA_TCD_init();           /* Init DMA Transfer Control Descriptor(s) */

  DMA->SSRT = 0;            /* Set chan 0 START bit to initiate first minor loop */
  while (((DMA->TCD[0].CSR >> DMA_TCD_CSR_START_SHIFT) & 1) |     /* Wait for START = 0 */
        ((DMA->TCD[0].CSR >> DMA_TCD_CSR_ACTIVE_SHIFT)  & 1))  {} /* and ACTIVE = 0 */
                                        /* Now minor loop has completed */

  while (!((DMA->TCD[0].CSR >> DMA_TCD_CSR_DONE_SHIFT) & 1) ) {     /* Loop till DONE = 1 */
    /* Place breakpoint at next instruction & observe expressions TCD0_Source, TCD0_Dest */
    DMA->SSRT = 0;                      /* Set chan 0 START bit to initiate next minor loop */
    while (((DMA->TCD[0].CSR >> DMA_TCD_CSR_START_SHIFT) & 1) |     /* Wait for START = 0 */
          ((DMA->TCD[0].CSR >> DMA_TCD_CSR_ACTIVE_SHIFT)  & 1))  {} /* and ACTIVE = 0 */
                                        /* Now minor loop has completed */
  }

  DMA->TCD[0].CSR &= ~(DMA_TCD_CSR_DONE_MASK);    /* Clear DONE bit */

  while (1) {}                                    /* Wait forever */
}
```

2.4.3.2 DMA.c

```
#include "S32K144.h" /* include peripheral declarations S32K144 */
#include "dma.h"
uint8_t TCD0_Source[] = {"Hello World"};          /* TCD 0 source (11 byte string) */
uint8_t volatile TCD0_Dest = 0;                   /* TCD 0 destination (1 byte) */
void DMA_init(void) {
 /* This is an initialization place holder for:    */
 /* 1. Enabling DMA MUX clock PCC_PCCn[PCC_DMAMUX_INDEX] (not needed when START bit used) */
 /* 2. Enabling desired channels by setting ERQ bit (not needed when START bit used) */
}
void DMA_TCD_init(void) {
                                /* TCD0: Transfers string to a single memory location */
  DMA->TCD[0].SADDR = DMA_TCD_SADDR_SADDR((uint32_t volatile) &TCD0_Source); /* Src */ DMA-
  >TCD[0].SOFF  =  DMA_TCD_SOFF_SOFF(1);  /*  Src  addr  add  1  byte  after  transfer*/  DMA-
  >TCD[0].ATTR = DMA_TCD_ATTR_SMOD(0) | /* Src modulo feature not used */
                             DMA_TCD_ATTR_SSIZE(0) | /* Src read 2**0 =1 byte per transfer*/
                             DMA_TCD_ATTR_DMOD(0)  | /* Dest modulo feature not used */
                             DMA_TCD_ATTR_DSIZE(0); /* Dest write 2**0 =1 byte per trans.*/
   DMA->TCD[0].NBYTES.MLNO  = DMA_TCD_NBYTES_MLNO_NBYTES(1); /* Transfer 1 byte /minor loop*/
   DMA->TCD[0].SLAST        = DMA_TCD_SLAST_SLAST(-11); /* Src addr change after major loop*/
   DMA->TCD[0].DADDR        = DMA_TCD_DADDR_DADDR((uint32_t volatile) &TCD0_Dest);/* Dest. */
  DMA->TCD[0].DOFF         = DMA_TCD_DOFF_DOFF(0);     /* No dest adr offset after transfer*/
   DMA->TCD[0].CITER.ELINKNO= DMA_TCD_CITER_ELINKNO_CITER(11) |/* 11 minor loop iterations*/
                             DMA_TCD_CITER_ELINKNO_ELINK(0);    /* No minor loop chan link */
  DMA->TCD[0].DLASTSGA  =  DMA_TCD_DLASTSGA_DLASTSGA(0);  /*  No  dest  chg  after  major  loop*/
  DMA->TCD[0].CSR = DMA_TCD_CSR_START(0) | /* Clear START status flag */
                             DMA_TCD_CSR_INTMAJOR(0)     |  /* No IRQ after major loop */
                            DMA_TCD_CSR_INTHALF(0)      |/* No IRQ after 1/2 major loop */
                            DMA_TCD_CSR_DREQ(1)         |  /* Disable chan after major loop*/
                            DMA_TCD_CSR_ESG(0)          |  /* Disable Scatter Gather */
                            DMA_TCD_CSR_MAJORELINK(0)   |  /* No major loop chan link */
                            DMA_TCD_CSR_ACTIVE(0)       |  /* Clear ACTIVE status flag */
                            DMA_TCD_CSR_DONE(0)         |  /* Clear DONE status flag */
                            DMA_TCD_CSR_MAJORLINKCH(0) |/* Chan # if major loop ch link */
                            DMA_TCD_CSR_BWC(0);            /* No eDMA stalls after R/W */
  DMA->TCD[0].BITER.ELINKNO= DMA_TCD_BITER_ELINKNO_BITER(11) |  /* Initial iteration count*/
                            DMA_TCD_BITER_ELINKNO_ELINK(0);    /* No minor loop chan link */
}
```

2.4.3.3 clocks_and_modes.c

See code in clocks_and_modes.c of the Hello World + Clock example.

2.5 Timed I/O (FTM)

2.5.1 描述

摘要：本示例使用Flex定时器模块（FTM）执行以下常见数字I/O功能：

- 边缘对齐脉冲宽度调制（EPWM）：1 Hz，低25%，高75%
- 输出比较（OC）：每100毫秒切换一次输出（10 MHz切换产生5 MHz频率）
- 输入捕获（IC）：捕获输入的上升或下降沿时间

FTM中的所有通道共享一个用于I/O功能的通用16位计数器。

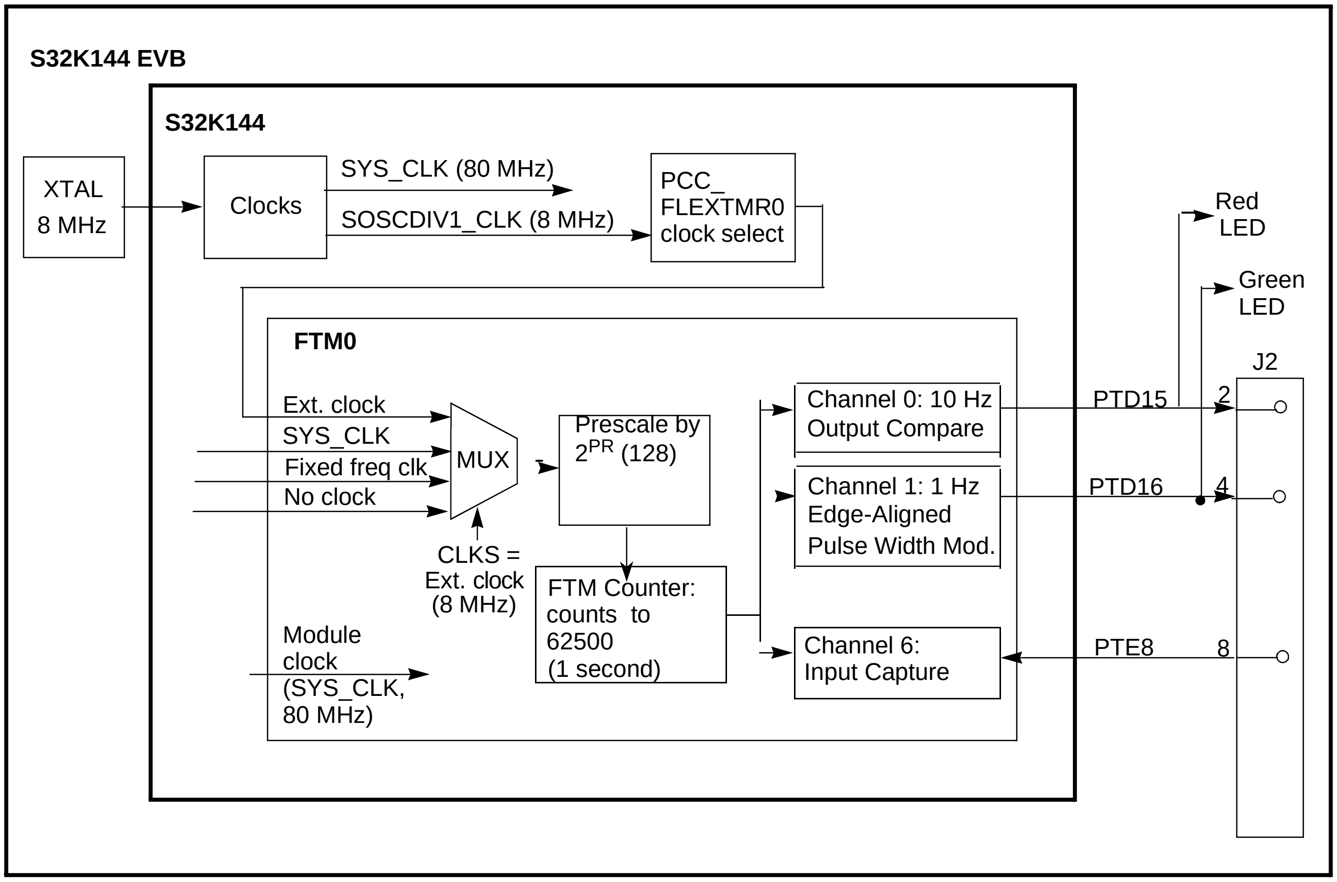


图6.定时I/O示例框图

要测量输入捕获时间，请将导线从J2引脚8连接到引脚4或引脚2。

2.5.2 设计

2.5.2.1 信道模式选择

通道模式通过整个FTM模块和单个通道的寄存器中的设置进行配置。下表显示了用于本示例中实现的通道模式的设置。

表6.FTM信道模式的定时I/O示例所需寄存器位字段设置

Module Registers Settings[1]				Channel Registers Settings		Mode	Configuration
SC [CPWMS]	COMBINE [DECAPENx]	COMBINE [MCOMBINEx]	COMBINE [COMBINEx]	C*n*SC [MS*n*B:MS*n*B]	C*n*SC [ELS*n*B:ELS*n*A]		
0	0	0	0	00	11	Input Capture	Capture on rising or falling edge
	0	0	0	01	01	Output Compare	Toggle on output match
	0	0	0	1X	X1	Edge-Aligned PWM	Low-true pulses (set on output match)

[1] “x” in the bitfield is a number that maps to a pair of channels. If x=0, it applies to channels 0, 1; x=2, channels 2, 3; etc.

2.5.2.2 结果输出波形

下面的波形显示了1 Hz时的PWM，占空比为75%，输出比较为10 Hz。

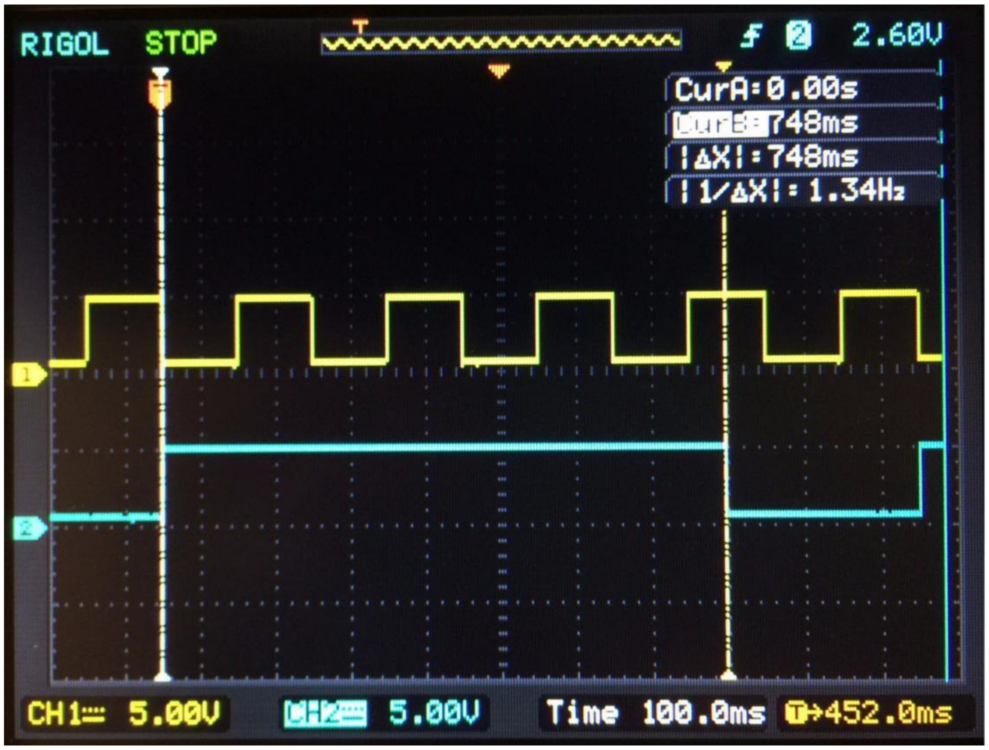


Figure 7. FTM example output waveforms

2.5.2.3 设计步骤

•禁用看门狗

•系统时钟：初始化8MHz的SOSC，初始化80MHz的sysclk，初始化80MHz的运行模式

•初始化FTM0。输入时钟源应为8 MHz SOSCDIV1_时钟：

— 禁用对FTM0寄存器的写保护以允许配置

— 控制：

— 按128（8 MHz/128=62500 Hz）预刻度时钟源

— 启用通道0、1作为输出

— 不使用过滤或中断

— PWM配置为递增计数（CPWMS=0）

— 时钟源=无（初始化后将启动时钟/计数器）

— 初始化模式，极性设置：CPWMS、DECAPEN、MCOMBINE、联合收割机，极性=0

— 计数器计数值=62500（1秒周期）

•将FTM0通道0初始化为输出比较，每100毫秒切换一次匹配：

— 为输出比较模式配置MSB:MSA、ELSB:ELSA

— 将初始比较值设置为6250（100毫秒）

— 将极性设置为活动高

•将FTM0通道1初始化为EPWM，1 Hz，75%占空比：

— 为PWM模式配置MSB:MSA、ELSB:ELSA

— 设置75%占空比的初始比较值

•将FTM0通道6初始化为输入捕获，任一边缘：

— 为输入捕获模式配置MSB:MSA、ELSB:ELSA

•初始化FTM0的端口引脚：

— 启用端口D和端口E模块的时钟

— PTD15:FTM0通道0-输出比较-连接至红色LED

— PTD16:FRM0通道1-PWM-连接到绿色LED

— PTE8:FTM0通道6-输入捕获

•启动FTM0计数器

•环路：

— 如果设置了输出比较匹配标志：

— 输出引脚切换（硬件自动）

— 更新下一个100毫秒的比较值（将6250添加到当前计数）

— 如果设置了输入捕获标志，则清除标志和读取计时器：

— 清晰的旗帜

— 存储以前的捕获值

— 读取当前捕获值

— 计算先前和当前捕获值的增量

软件示例

2.5.3 代码

2.5.3.1 main.c

```c
#include "S32K144.h"          /* include peripheral declarations S32K144 */
#include "clocks_and_modes.h"
#include "FTM.h"

void PORT_init (void) {
  PCC->PCCn[PCC_PORTD_INDEX ]|=PCC_PCCn_CGC_MASK;   /* Enable clock for PORTD */
  PCC->PCCn[PCC_PORTE_INDEX ]|=PCC_PCCn_CGC_MASK;   /* Enable clock for PORTE */
  PORTE->PCR[8]|=PORT_PCR_MUX(2);                    /* Port E8: MUX = ALT2, FTM0CH6 */
  PORTD->PCR[15]|=PORT_PCR_MUX(2);                   /* Port D15: MUX = ALT2, FTM0CH0 */
  PORTD->PCR[16]|=PORT_PCR_MUX(2);                   /* Port D16: MUX = ALT2, FTM0CH1 */
}

void WDOG_disable (void) {
  WDOG->CNT=0xD928C520;     /* Unlock watchdog */
  WDOG->TOVAL=0x0000FFFF;   /* Maximum timeout value */
  WDOG->CS = 0x00002100;    /* Disable watchdog */
}

int main(void) {
  WDOG_disable();        /* Disable WDOG*/
  SOSC_init_8MHz();      /* Initialize system oscillator for 8 MHz xtal */
  SPLL_init_160MHz();     /* Initialize SPLL to 160 MHz with 8 MHz SOSC */
  NormalRUNmode_80MHz(); /* Init clocks: 80 MHz SPLL & core, 40 MHz bus, 20 MHz flash */
  FTM0_init();           /* Init FTM0 */
  FTM0_CH0_OC_init();    /* Init FTM0 CH0, red LED */
  FTM0_CH1_PWM_init();   /* Init FTM0 CH1, green LED */
  FTM0_CH6_IC_init();    /* Init FTM0 CH6, j2-8 */
  PORT_init();           /* Configure ports */
  start_FTM0_counter();  /* Start FTM0 counter */
  for(;;) {
       FTM0_CH0_output_compare();/* If output compare match: */
                               /* Pin toggles (automatically by hardware) */
                               /* Clear flag 8 */
                               /* Reload timer */
       FTM0_CH6_input_capture(); /* If input captured: clear flag, read timer */
}
```

2.5.3.2 FTM.c

```
#include "S32K144.h" /* include peripheral declarations S32K144 */
#include "FTM.h"
uint16_t CurrentCaptureVal = 0;
uint16_t PriorCaptureVal = 0;
uint16_t DeltaCapture = 0;

void FTM0_init(void) {
  PCC->PCCn[PCC_FLEXTMR0_INDEX] &= ~PCC_PCCn_CGC_MASK; /* Ensure clk disabled for config */
  PCC->PCCn[PCC_FLEXTMR0_INDEX] |= PCC_PCCn_PCS(0b001)/* Clock Src=1, 8 MHz SOSCDIV1_CLK */
                                |  PCC_PCCn_CGC_MASK;   /* Enable clock for FTM regs */
  FTM0->MODE |= FTM_MODE_WPDIS_MASK;  /* Write protect to registers disabled (default) */
  FTM0->SC = 0x00030007;      /* Enable PWM channel 0 output*/
                              /* Enable PWM channel 1 output*/
                              /* TOIE (Timer Overflow Interrupt Ena) = 0 (default) */
                             /* CPWMS (Center aligned PWM Select) = 0 (default, up count) */
                             /* CLKS (Clock source) = 0 (default, no clock; FTM disabled) */
                              /* PS (Prescaler factor) = 7. Prescaler = 128 */
  FTM0->COMBINE = 0x00000000;/* FTM mode settings used: DECAPENx, MCOMBINEx, COMBINEx=0   */
  FTM0->POL = 0x00000000;     /* Polarity for all channels is active high (default) */
  FTM0->MOD = 62500 -1 ;      /* FTM1 counter final value (used for PWM mode) */
                              /* FTM1 Period = MOD-CNTIN+0x0001 ~= 62500 ctr clks   */
                              /* 8MHz /128 = 62.5kHz ->  ticks -> 1Hz */
}

void FTM0_CH0_OC_init(void) {
  FTM0->CONTROLS[0].CnSC = 0x00000014; /* FTM0 ch0: Output Compare, toggle output on match */
                                       /* CHIE (Chan Interrupt Ena)= 0 (default) */
                                      /* MSB:MSA (chan Mode Select)= 0b01, Output Compare */
                                    /* ELSB:ELSA (chan Edge or Level Select)= 0b01, toggle*/
  FTM0->CONTROLS[0].CnV= 6250; /* FTM0 ch 0 Compare Value= 6250 clks, 100ms toggle*/ FTM0-
  >POL &= ~FTM_POL_POL0_MASK; /* FTM0 ch 0 polarity = 0 (Default, active high) */
}

void FTM0_CH1_PWM_init(void) {
  FTM0->CONTROLS[1].CnSC = 0x00000028;   /* FTM0 ch1: edge-aligned PWM, low true pulses */
                                         /* CHIE (Chan Interrupt Ena) = 0 (default) */
                                        /* MSB:MSA (chan Mode Select)=0b10, Edge Align PWM*/
                                     /* ELSB:ELSA (chan Edge/Level Select)=0b10, low true */
  FTM0->CONTROLS[1].CnV =  46875;        /* FTM0 ch1 compare value (~75% duty cycle) */
}
```

```
void FTM0_CH6_IC_init(void) {
  FTM0->CONTROLS[6].CnSC = 0x0000000C;  /* FTM0 ch6: Input Capture rising or falling edge */
                                        /* CHIE (Chan Interrupt Ena) = 0 (default) */
                                        /* MSB:MSA (chan Mode Select)=0b00, Input Capture */
                                    /* ELSB:ELSA (ch Edge/Level Select)=0b11, rise or fall*/
}

void FTM0_CH0_output_compare(void) {
  if (1==((FTM0->CONTROLS[0].CnSC & FTM_CnSC_CHF_MASK)>>FTM_CnSC_CHF_SHIFT)) {
                                                     /* If chan flag is set */
  FTM0->CONTROLS[0].CnSC &= ~FTM_CnSC_CHF_MASK;  /* Clear flag: read reg then set CHF=0 */
    if(  FTM0->CONTROLS[0].CnV==56250)  {         /* If count at last value before end, */
      FTM0->CONTROLS[0].CnV= 0 ;                 /* Update compare value: to 0*/
    }
    else {
      FTM0->CONTROLS[0].CnV= FTM0->CONTROLS[0].CnV + 6250 ;
                                     /* Update compare value: add 6250 to current value*/
      }
  }
}

void FTM0_CH6_input_capture(void) {
  if (1==((FTM0->CONTROLS[6].CnSC & FTM_CnSC_CHF_MASK)>>FTM_CnSC_CHF_SHIFT)) {
                                                       /* If chan flag is set */
  FTM0->CONTROLS[6].CnSC &= ~FTM_CnSC_CHF_MASK;    /* Clear flag: read reg then set CHF=0 */
  PriorCaptureVal = CurrentCaptureVal;             /* Record value of prior capture */
  CurrentCaptureVal = FTM0->CONTROLS[6].CnV;       /* Record value of current capture */
  DeltaCapture = CurrentCaptureVal - PriorCaptureVal;
                              /* Will be 6250 clocks (100 msec) if connected to FTM0 CH0 */
  }
}

void start_FTM0_counter (void)
  { FTM0->SC |= FTM_SC_CLKS(3);
                    /* Start FTM0 counter with clk source = external clock (SOSCDIV1_CLK)*/
}
```

2.5.3.3 clocks_and_modes.c

See code in clocks_and_modes.c of the Hello World + Clock example.

2.6 ADC - SW 触发器

2.6.1 描述

摘要： ADC初始化为一次性转换的软件触发器转换两个通道。每个转换都需要自己的软件触发器。一个通道（AD12）连接到S32K144评估板上的电位计另一个与VREFSH相连。结果按0至5000 mV的比例缩放。

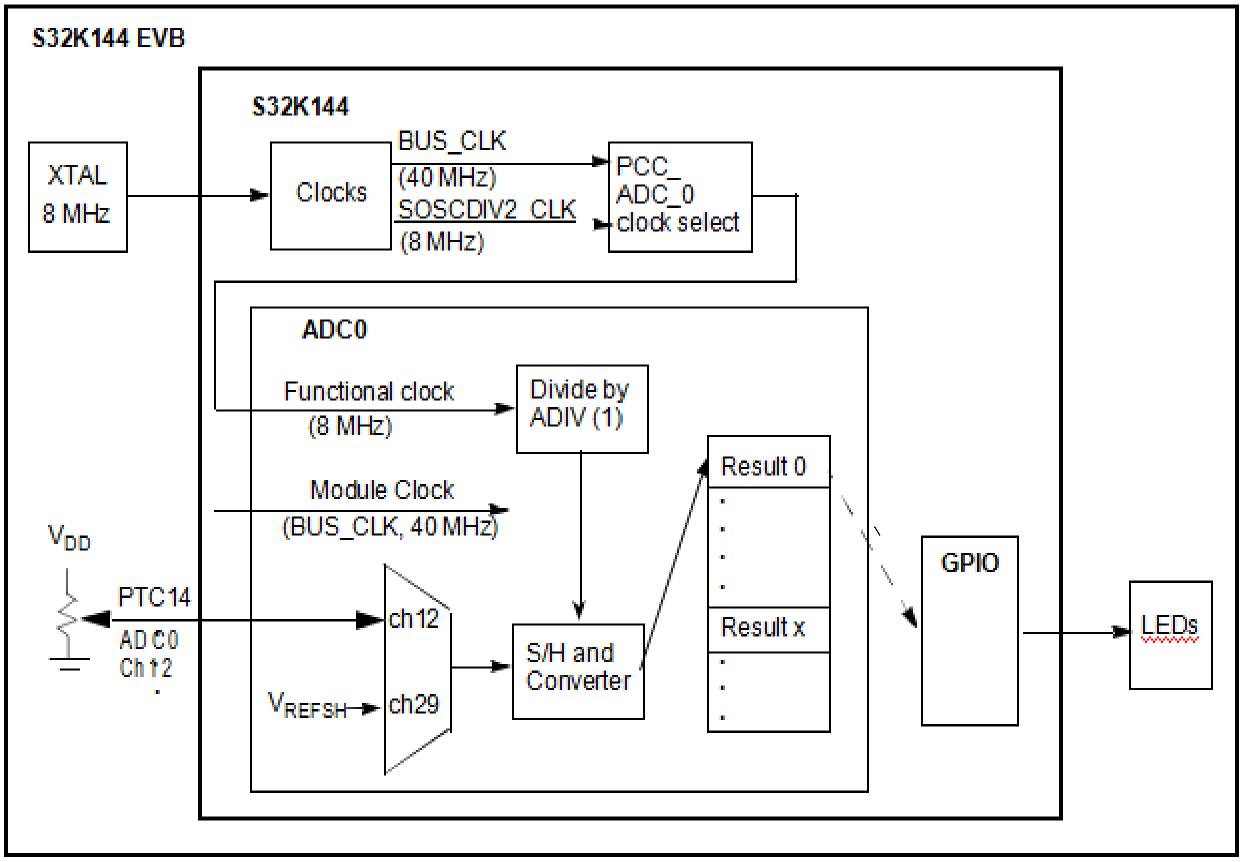


图8.ADC示例框图

在评估板上，根据下表使用三个LED指示转换结果范围。

表7.来自电位计的ADC示例输入电压的LED颜色

Scaled conversion result	LED illuminated
3750 - 5000 mV	Red
2500 - 3750 mV	Green
1250 - 2500 mV	Blue
0 to 1250 mV	None

2.6.2 设计

此简单示例中不包括ADC校准。因此，结果可能低于规定的精度。初始化校准机构的步骤见参考手册ADC章节的校准功能部分。

•禁用看门狗

•系统时钟：初始化8MHz的SOSC，初始化80MHz的sysclk，初始化80MHz的运行模式

•初始化端口引脚：

— 启用端口D的时钟

— PTD0:GPIO输出-连接到蓝色LED

— PTD15:GPIO输出-连接至红色LED

— PTD16:GPIO输出-连接到绿色LED

— （不复位，模拟管脚无需配置。）

•初始化ADC：

— 选择SOSCDIV2_CLK作为功能时钟，并将其启用到模块

— 禁用模块并禁用来自模块的中断请求（重置默认状态）

— 使用SOSCDIV2_CLK（除以1）为12位转换配置ADC

— 配置13个ADCK时钟周期的采样时间（重置默认值）

— 选择软件触发器进行转换，无比较功能，无DMA，并使用默认电压参考引脚-外部引脚VREFH和VREFL(重置默认值）

— 禁用连续转换（因此每个软件触发器有一个转换）、禁用硬件平均、禁用校准序列启动

•环路：

— 为通道12发出ADC转换命令，该通道连接到NXP评估板上的电位计(将ADC_SC1[0]用于软件触发器。）

— 等待转换完成标志。转换完成后：

— 读取结果并缩放至0至5000 mV（所有软件触发器的结果为ADC_R[0]）

— 根据电压范围点亮LED

— 发出ADC转换命令以读取通道29、ADC高参考电压（使用ADC_SC1[0]作为软件触发器。）

— 等待转换完成标志。转换完成后：

— 读取结果并缩放至0至5000 mV（所有软件触发器的结果为ADC_R[0]）

2.6.3 代码

2.6.3.1 main.c

```
#include "S32K144.h" /* include peripheral declarations S32K144 */
#include "clocks_and_modes.h"
#include "ADC.h"

#define PTD15 15 /* RED LED*/
#define PTD16 16 /* GREEN LED*/
#define PTD0 0   /* BLUE LED */

  uint32_t adcResultInMv_pot = 0;
  uint32_t adcResultInMv_Vrefsh = 0;

void PORT_init (void) {
  PCC->PCCn[PCC_PORTD_INDEX ]|=PCC_PCCn_CGC_MASK;   /* Enable clock for PORTD */
  PORTD->PCR[PTD0]  =  0x00000100;  /* Port D0: MUX = GPIO */
  PORTD->PCR[PTD15] =  0x00000100;  /* Port D15: MUX = GPIO */
  PORTD->PCR[PTD16] =  0x00000100;  /* Port D16: MUX = GPIO */

  PTD->PDDR |= 1<<PTD0;            /* Port D0:  Data Direction= output */
  PTD->PDDR |= 1<<PTD15;          /* Port D15: Data Direction= output */
  PTD->PDDR |= 1<<PTD16;          /* Port D16: Data Direction= output */
}

void WDOG_disable (void){
  WDOG->CNT=0xD928C520;     /* Unlock watchdog */
  WDOG->TOVAL=0x0000FFFF;   /* Maximum timeout value */
  WDOG->CS = 0x00002100;    /* Disable watchdog */
}

int main(void)
{
  WDOG_disable();        /* Disable WDOG*/
  SOSC_init_8MHz();      /* Initialize system oscillator for 8 MHz xtal */
  SPLL_init_160MHz();    /* Initialize SPLL to 160 MHz with 8 MHz SOSC */
  NormalRUNmode_80MHz(); /* Init clocks: 80 MHz sysclk & core, 40 MHz bus, 20 MHz flash */
  PORT_init();          /* Init  port clocks and gpio outputs */
  ADC_init();            /* Init ADC resolution 12 bit*/
  for(;;) {
    convertAdcChan(12);                    /* Convert Channel AD12 to pot on EVB */
    while(adc_complete()==0){}             /* Wait for conversion complete flag */
    adcResultInMv_pot = read_adc_chx();        /* Get channel's conversion results in mv */

    if (adcResultInMv_pot > 3750) {        /* If result > 3.75V */
      PTD->PSOR |= 1<<PTD0 | 1<<PTD16;     /* turn off blue, green LEDs */
      PTD->PCOR |= 1<<PTD15;               /* turn on red LED */
    }
    else if (adcResultInMv_pot > 2500) {   /* If result > 3.75V */
      PTD->PSOR |= 1<<PTD0 | 1<<PTD15;     /* turn off blue, red LEDs */
      PTD->PCOR |= 1<<PTD16;            /* turn on green LED */
    }
    else if (adcResultInMv_pot >1250) {        /* If result > 3.75V */
      PTD->PSOR |= 1<<PTD15 | 1<<PTD16;    /* turn off red, green LEDs */
      PTD->PCOR |= 1<<PTD0;             /* turn on blue LED */
    }
    else {
      PTD->PSOR |= 1<<PTD0 | 1<< PTD15 | 1<<PTD16; /* Turn off all LEDs */
    }

    convertAdcChan(29);                    /* Convert chan 29, Vrefsh */
   while(adc_complete()==0){}              /* Wait for conversion complete flag */
   adcResultInMv_Vrefsh = read_adc_chx();       /* Get channel's conversion results in mv*/
  }
}
```

2.6.3.2 ADC.c

```c
#include "ADC.h"

void ADC_init(void)  {
  PCC->PCCn[PCC_ADC0_INDEX] &=~ PCC_PCCn_CGC_MASK;  /* Disable clock to change PCS */
  PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_PCS(1);     /* PCS=1: Select SOSCDIV2 */
  PCC->PCCn[PCC_ADC0_INDEX] |= PCC_PCCn_CGC_MASK;   /* Enable bus clock in ADC */

  ADC0->SC1[0] =0x00001F;          /* ADCH=1F: Module is disabled for conversions*/
                                   /* AIEN=0: Interrupts are disabled */
  ADC0->CFG1 = 0x000000004;        /* ADICLK=0: Input clk=ALTCLK1=SOSCDIV2 */
                                   /* ADIV=0: Prescaler=1 */
                                   /* MODE=1: 12-bit conversion */
  ADC0->CFG2 = 0x00000000C;        /* SMPLTS=12(default): sample time is 13 ADC clks */
  ADC0->SC2 = 0x00000000;          /* ADTRG=0: SW trigger */
                                   /* ACFE,ACFGT,ACREN=0: Compare func disabled */
                                   /* DMAEN=0: DMA disabled */
                                   /* REFSEL=0: Voltage reference pins= VREFH, VREEFL */
  ADC0->SC3 = 0x00000000;          /* CAL=0: Do not start calibration sequence */
                                   /* ADCO=0: One conversion performed */
                                   /* AVGE,AVGS=0: HW average function disabled */
}

void convertAdcChan(uint16_t adcChan) {   /* For SW trigger mode, SC1[0] is used */
  ADC0->SC1[0]&=~ADC_SC1_ADCH_MASK;      /* Clear prior ADCH bits */
  ADC0->SC1[0] = ADC_SC1_ADCH(adcChan);   /* Initiate Conversion*/
}

uint8_t adc_complete(void)  {
  return ((ADC0->SC1[0] & ADC_SC1_COCO_MASK)>>ADC_SC1_COCO_SHIFT); /* Wait for completion*/
}

uint32_t read_adc_chx(void)
                               {
   uint16_t adc_result=0;
  adc_result=ADC0->R[0];      /* For SW trigger mode, R[0] is used */
  return  (uint32_t) ((5000*adc_result)/0xFFF); /* Convert result to mv for 0-5V range */
}
```

2.6.3.3 clocks_and_modes.c

See code in clocks_and_modes.c of the Hello World + Clock example.

2.7 UART

2.7.1 描述

摘要：此示例执行一个简单的UART 9600波特传输到PC上的COM端口。未实现FIFO、中断和DMA。

开放式SDA接口可在评估板上使用，其中UART信号传输至USB接口，USB接口可连接至具有终端仿真程序（如PUTTY、TeraTerm或其他软件）的PC。

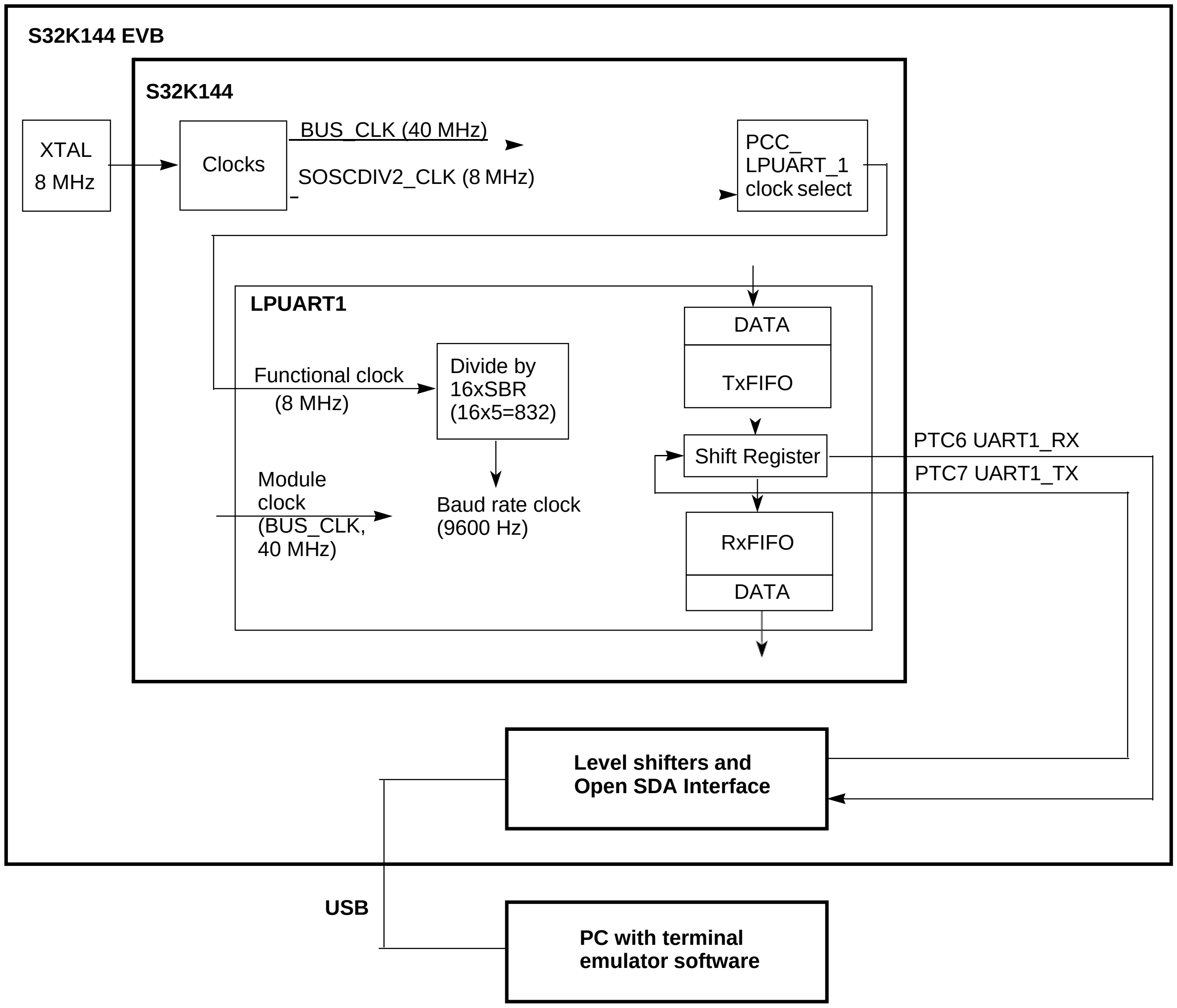


图9.UART示例框图

可以使用PC上的串口调试助手，如PuTTY。要配置PuTTY，请执行以下操作：

1.确定要使用的COM端口：
  -打开Windows设备管理器
  -扩展端口（COM和LPT）
  -注意哪个COM端口用于OpenSDA(示例：COM3）
2.设置putty。
3.配置会话对话框：
  -会话-连接类型：单击串行按钮
  -会话-串行线：输入COM端口（示例：COM3）
  -会话-速度：输入所需的波特率（本例中为9600）
4.展开连接组（如果未展开）并选择串行对话框：
  -连接-串行：验证COM端口号是否正确
  -SONECTION-串行：在本例中设置速度（波特率）：9600
  -连接-串行：根据需要，设置数据位数、奇偶校验、停止位、流量控制
5.点击“打开”按钮打开串行调试窗口。

2.7.2 设计

此基本示例中不包括溢出处理。因此，如果接收数据的速度比软件处理数据的速度快，则会丢失一些数据。如果这是一个问题，可以将溢出处理逻辑添加到应用程序中。

•禁用看门狗

•系统时钟：初始化8MHz的SOSC，初始化80MHz的sysclk，初始化80MHz的运行模式

•初始化端口引脚：

— 启用端口C模块的时钟

— PTC6、PTC7：配置LPUART1\U RX、LPUART1\U TX

•初始化LPUART1：

— 启用SOSC_DIV2_时钟的时钟源

— 配置波特率：9600波特，一个停止位，8位字符

— 未启用中断、DMA或匹配功能

— 配置LPUART1控件：启用发射机、接收机、无奇偶校验、8位字符

•传输两个字符串：

— 要发送的每个字符的循环：如果设置了传输数据就绪状态位，则将字符写入数据寄存器

•循环回显接收到的字符：

— 传输提示字符（“>”）

— 等待设置RDRF标志，然后读取字符

— 传回读字符

2.7.3 代码

2.7.3.1 main.c

```c
#include "S32K144.h" /* include peripheral declarations S32K144 */
#include "clocks_and_modes.h"
#include "LPUART.h"

char data=0;
void PORT_init (void) {
  PCC->PCCn[PCC_PORTC_INDEX ]|=PCC_PCCn_CGC_MASK; /* Enable clock for PORTC */
  PORTC->PCR[6]|=PORT_PCR_MUX(2);           /* Port C6: MUX = ALT2,UART1 TX */
  PORTC->PCR[7]|=PORT_PCR_MUX(2);           /* Port C7: MUX = ALT2,UART1 RX */
}

void WDOG_disable (void){
  WDOG->CNT=0xD928C520;     /* Unlock watchdog */
  WDOG->TOVAL=0x0000FFFF;   /* Maximum timeout value */
  WDOG->CS = 0x00002100;    /* Disable watchdog */
}

int main(void)
{
  WDOG_disable();        /* Disable WDGO*/
  SOSC_init_8MHz();      /* Initialize system oscillator for 8 MHz xtal */
  SPLL_init_160MHz();     /* Initialize SPLL to 160 MHz with 8 MHz SOSC */
  RUNmode_80MHz();       /* Init clocks: 80 MHz SPLL & core, 40 MHz bus, 20 MHz flash */
  PORT_init();           /* Configure ports */

  LPUART1_init();        /* Initialize LPUART @ 9600*/
  LPUART1_transmit_string("Running LPUART example\n\r");     /* Transmit char string */
  LPUART1_transmit_string("Input character to echo...\n\r"); /* Transmit char string */

  for(;;) {
        LPUART1_transmit_char('>');           /* Transmit prompt character*/
        LPUART1_receive_and_echo_char();/* Wait for input char, receive & echo it*/
  }
}
```

2.7.3.2 LPUART.c

```
#include "S32K144.h" /* include peripheral declarations S32K144 */
#include "LPUART.h"

void LPUART1_init(void)  /* Init. summary: 9600 baud, 1 stop bit, 8 bit format, no parity */
{
  PCC->PCCn[PCC_LPUART1_INDEX] &= ~PCC_PCCn_CGC_MASK;    /* Ensure clk disabled for config */
  PCC->PCCn[PCC_LPUART1_INDEX] |= PCC_PCCn_PCS(0b001)    /* Clock Src= 1 (SOSCDIV2_CLK) */
                               |  PCC_PCCn_CGC_MASK;     /* Enable clock for LPUART1 regs */

  LPUART1->BAUD = 0x0F000034;  /* Initialize for 9600 baud, 1 stop: */
                               /* SBR=52 (0x34): baud divisor = 8M/9600/16 = ~52 */
                               /* OSR=15: Over sampling ratio = 15+1=16 */
                               /* SBNS=0: One stop bit */
                               /* BOTHEDGE=0: receiver samples only on rising edge */
                               /* M10=0: Rx and Tx use 7 to 9 bit data characters */
                               /* RESYNCDIS=0: Resync during rec'd data word supported */
                               /* LBKDIE, RXEDGIE=0: interrupts disable */
                               /* TDMAE, RDMAE, TDMAE=0: DMA requests disabled */
                               /* MAEN1, MAEN2,  MATCFG=0: Match disabled */

  LPUART1->CTRL=0x000C0000;    /* Enable transmitter & receiver, no parity, 8 bit char: */
                               /* RE=1: Receiver enabled */
                               /* TE=1: Transmitter enabled */
                               /* PE,PT=0: No hw parity generation or checking */
                               /* M7,M,R8T9,R9T8=0: 8-bit data characters*/
                               /* DOZEEN=0: LPUART enabled in Doze mode */
                            /* ORIE,NEIE,FEIE,PEIE,TIE,TCIE,RIE,ILIE,MA1IE,MA2IE=0: no IRQ*/
                               /* TxDIR=0: TxD pin is input if in single-wire mode */
                               /* TXINV=0: TRansmit data not inverted */
                               /* RWU,WAKE=0: normal operation; rcvr not in statndby */
                               /* IDLCFG=0: one idle character */
                               /* ILT=0: Idle char bit count starts after start bit */
                               /* SBK=0: Normal transmitter operation - no break char */
                               /* LOOPS,RSRC=0: no loop back */
}
```

```
void LPUART1_transmit_char(char send) {    /* Function to Transmit single Char */
  while((LPUART1->STAT & LPUART_STAT_TDRE_MASK)>>LPUART_STAT_TDRE_SHIFT==0);
                                /* Wait for transmit buffer to be empty */
  LPUART1->DATA=send;           /* Send data */
}

void LPUART1_transmit_string(char data_string[])  {  /* Function to Transmit whole string */
  uint32_t i=0;
  while(data_string[i] != '\0')  {          /* Send chars one at a time */
    LPUART1_transmit_char(data_string[i]);
    i++;
  }
}

char LPUART1_receive_char(void) {    /* Function to Receive single Char */
  char receive;
  while((LPUART1->STAT & LPUART_STAT_RDRF_MASK)>>LPUART_STAT_RDRF_SHIFT==0);
                                  /* Wait for received buffer to be full */
  receive= LPUART1->DATA;         /* Read received data*/
  return receive;
}

void LPUART1_receive_and_echo_char(void)  {  /* Function to echo received char back */
  char send = LPUART1_receive_char();        /* Receive Char */
  LPUART1_transmit_char(send);               /* Transmit same char back to the sender */
  LPUART1_transmit_char('\n');               /* New line */
}
```

2.7.3.3 clocks_and_modes.c

See code in clocks_and_modes.c of the Hello World + Clock example.

2.8 SPI

2.8.1 描述

摘要：使用FIFO执行简单的LPSPI传输，可以提高吞吐量。初始化后，以1 Mbps的速率传输16位帧。软件将轮询标志，而不是使用中断和DMA。SBC的状态寄存器被读取为变量LPSPI1_16bits_read。对于UJA1169，通常读取的值为0xFDEF。

S32K144 EVB注：该示例使用LPSP1和外围芯片Select 3，该芯片连接到收发器UGA1169TK/F1。要为SBC供电，请将外部12V电源连接至EVB，并连接上的针脚1-2跳线J107。USB电缆仍然可以连接到EVB，以便继续调试。如果SBC未通电，LPSPI数据将全部为零。

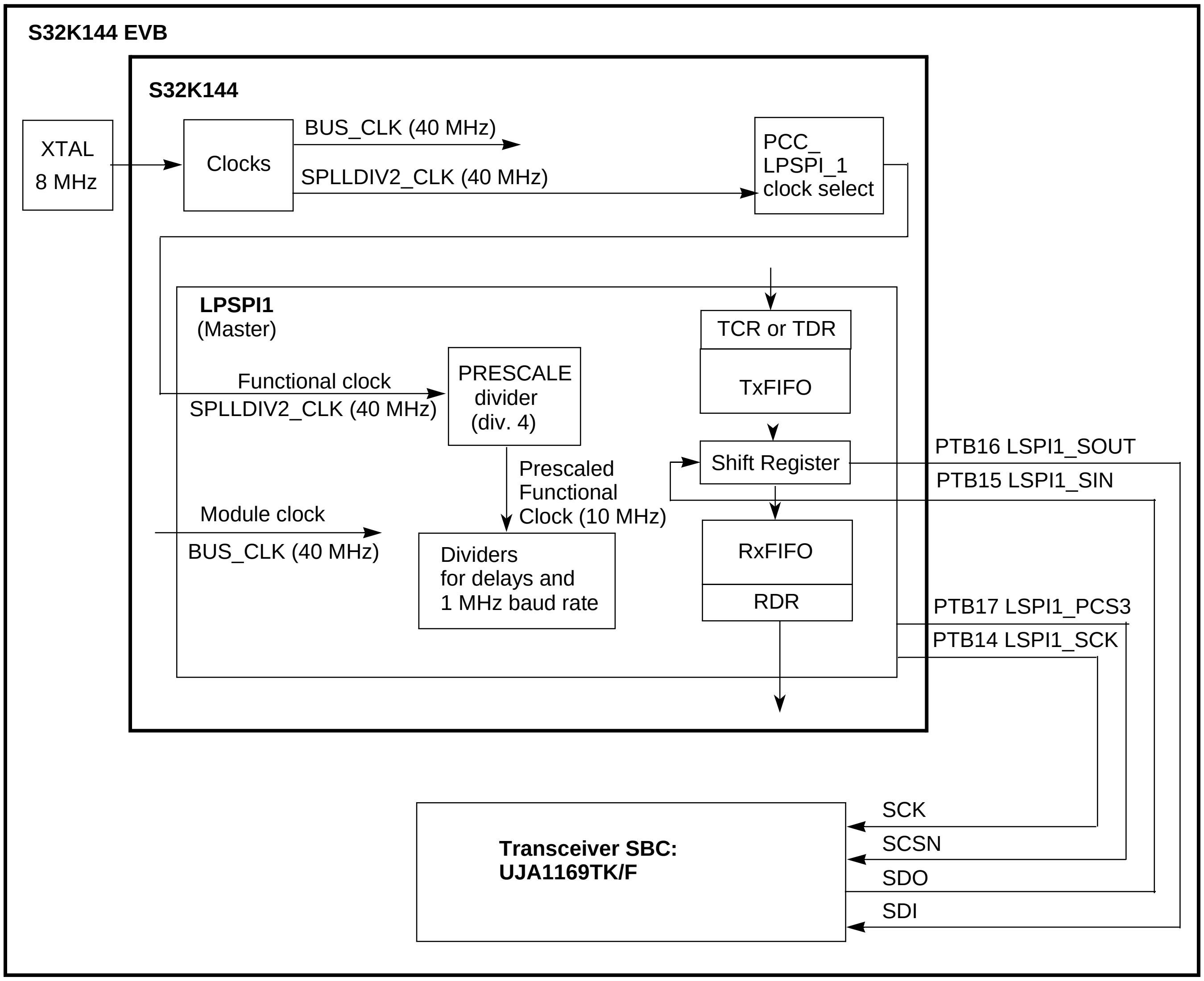


图10.LPSPI示例框图

1.Initial S32K144 EVBs, currently obsolete, use MC33903 transceiver, have different SPI commands.

2.8.2 设计

•禁用看门狗

•系统时钟：初始化8MHz的SOSC，初始化80MHz的sysclk，初始化80MHz的运行模式

•初始化LPSPI1：

— 模块控制：

— 禁用模块以允许配置

— 将LPSPI配置为主机

— 初始化10 MHz预刻度功能时钟（100 usec周期）的所需时钟配置：

— 预刻度功能时钟频率=功能时钟/预刻度=40 MHz/4=10 MHz

— SCK波特率=（功能时钟/预刻度）/（SCKDIV+2）
=（40 MHz/4）/（8+2）=10 MHz/10=1 MHz

— SCK至PCS延迟=5个预定功能时钟=50 nesc

— PCS到CSK延迟=10个预定功能时钟=1个usec

— 传输之间的延迟=10个预定功能时钟=1 usec。

— 先进先出控制：

— RxFIFO：当FIFO中的字>0时设置接收数据标志（RDF）

— TxFIFO：当FIFO中的字数小于等于3时设置传输数据标志（TDF）

— 配置传输命令（稍后可以应用其他配置，例如，对于使用不同芯片选择和帧大小的数据）：

— 按8预分频功能时钟（80 MHz/8=10 MHz分频时钟）

— 帧大小=16位

— 用于外围芯片选择的PCS3

— SCK极性激活低

— 阶段：数据在SCK前缘更改，在SCK后缘捕获

— MSB优先，字节交换禁用，连续传输禁用

— 用于传输的单位宽度

— 正常FIFO使用：Rx FIFO中存储的Rx数据，从Tx FIFO加载的Tx数据

— 模块控制：

— 启用模块，包括调试和打瞌睡模式

•初始化LPSPI1的端口引脚

•环路：

— 等待设置传输数据标志（TDF）（指示传输FIFO可用性），然后写入一个SPI帧以传输FIFO并清除标志

— 等待设置接收数据标志（RDF）（表示接收FIFO有数据要读取），然后读取接收到的SPI帧并清除标志。预期数据：来自UJA1169TK/F的0xFDEF

— 递增计数器

2.8.3 代码

2.8.3.1 main.c

```c
#include "S32K144.h"           /* include peripheral declarations S32K144 */
#include "LPSPI.h"
#include "clocks_and_modes.h"

  uint16_t tx_16bits = 0xFD00;  /* SBC UJA1169: read Dev ID Reg @ 0x7E (expect non-zero)*/
                                /* Note: Obsolete EVB with MC33903 example used 0c2580 */
                                /*       to read SAFE reg flags (expect nonzero result).*/
  uint16_t LPSPI1_16bits_read;  /* Returned data in to SPI */

void WDOG_disable (void){
  WDOG->CNT=0xD928C520;     /*Unlock watchdog*/
  WDOG->TOVAL=0x0000FFFF;   /*Maximum timeout value*/
  WDOG->CS = 0x00002100;    /*Disable watchdog*/
}

void PORT_init (void) {
  PCC->PCCn[PCC_PORTB_INDEX ]|=PCC_PCCn_CGC_MASK; /* Enable clock for PORTB */
  PORTB->PCR[14]|=PORT_PCR_MUX(3); /* Port B14: MUX = ALT3, LPSPI1_SCK */
  PORTB->PCR[15]|=PORT_PCR_MUX(3); /* Port B15: MUX = ALT3, LPSPI1_SIN */
  PORTB->PCR[16]|=PORT_PCR_MUX(3); /* Port B16: MUX = ALT3, LPSPI1_SOUT */
  PORTB->PCR[17]|=PORT_PCR_MUX(3); /* Port B17: MUX = ALT3, LPSPI1_PCS3 */
}

int main(void)
  { uint32_t counter =
  0; WDOG_disable();
  SOSC_init_8MHz();        /* Initialize system oscillator for 8 MHz xtal */
  SPLL_init_160MHz();      /* Initialize SPLL to 160 MHz with 8 MHz SOSC */
  NormalRUNmode_80MHz();   /* Init clocks: 80 MHz sysclk & core, 40 MHz bus, 20 MHz flash */
  LPSPI1_init_master();    /* Initialize LPSPI 1 as master */
  PORT_init();             /* Configure ports */
  for(;;) {
    LPSPI1_transmit_16bits(tx_16bits);       /* Transmit half word (16 bits) on LPSPI1 */
    LPSPI1_16bits_read = LPSPI1_receive_16bits(); /* Receive half word on LSPI1 */
    counter++;
  }
```

2.8.3.2 LPSPI.c

```
#include "S32K144.h" /* include peripheral declarations S32K144 */
#include "LPSPI.h"
void LPSPI1_init_master(void) {
  PCC->PCCn[PCC_LPSPI1_INDEX] = 0;          /* Disable clocks to modify PCS ( default) */
  PCC->PCCn[PCC_LPSPI1_INDEX] = 0xC6000000; /* Enable PCS=SPLL_DIV2 (40 MHz func'l clock) */
  LPSPI1->CR    = 0x00000000;   /* Disable module for configuration */
  LPSPI1->IER   = 0x00000000;   /* Interrupts not used */
  LPSPI1->DER   = 0x00000000;   /* DMA not used */
  LPSPI1->CFGR0 = 0x00000000;   /* Defaults: */
                                /* RDM0=0: rec'd data to FIFO as normal */
                                /* CIRFIFO=0; Circular FIFO is disabled */
                                /* HRSEL, HRPOL, HREN=0: Host request disabled */
  LPSPI1->CFGR1 = 0x00000001;   /* Configurations: master mode*/
                                /* PCSCFG=0: PCS[3:2] are enabled */
                              /* OUTCFG=0: Output data retains last value when CS negated */
                                /* PINCFG=0: SIN is input, SOUT is output */
                                /* MATCFG=0: Match disabled */
                                /* PCSPOL=0: PCS is active low */
                                /* NOSTALL=0: Stall if Tx FIFO empty or Rx FIFO full */
                                /* AUTOPCS=0: does not apply for master mode */
                                /* SAMPLE=0: input data sampled on SCK edge */
                                /* MASTER=1: Master mode */
  LPSPI1->TCR   = 0x5300000F;   /* Transmit cmd: PCS3, 16bits, prescale func'l clk by 4. */
                                /* CPOL=0: SCK inactive state is low */
                             /* CPHA=1: Change data on SCK lead'g, capture on trail'g edge*/
                                /* PRESCALE=2: Functional clock divided by 2**2 = 4 */
                                /* PCS=3: Transfer using PCS3 */
                                /* LSBF=0: Data is transferred MSB first */
                                /* BYSW=0: Byte swap disabled */
                                /* CONT, CONTC=0: Continuous transfer disabled */
                                /* RXMSK=0: Normal transfer: rx data stored in rx FIFO */
                                /* TXMSK=0: Normal transfer: data loaded from tx FIFO */
                                /* WIDTH=0: Single bit transfer */
                                /* FRAMESZ=15: # bits in frame = 15+1=16 */
  LPSPI1->CCR   = 0x04090808;   /* Clk dividers based on prescaled func'l clk of 100 nsec */
                                /* SCKPCS=4: SCK to PCS delay = 4+1 = 5 (500 nsec) */
                                /* PCSSCK=4: PCS to SCK delay = 9+1 = 10 (1 usec) */
                                /* DBT=8: Delay between Transfers = 8+2 = 10 (1 usec) */
                               /* SCKDIV=8: SCK divider =8+2 = 10 (1 usec: 1 MHz baud rate) */
  LPSPI1->FCR   = 0x00000003;   /* RXWATER=0: Rx flags set when Rx FIFO >0 */
                                /* TXWATER=3: Tx flags set when Tx FIFO <= 3 */
  LPSPI1->CR    = 0x00000009;   /* Enable module for operation */
                                /* DBGEN=1: module enabled in debug mode */
                                /* DOZEN=0: module enabled in Doze mode */
                                /* RST=0: Master logic not reset */
                                /* MEN=1: Module is enabled */
}

void LPSPI1_transmit_16bits (uint16_t send) {
  while((LPSPI1->SR & LPSPI_SR_TDF_MASK)>>LPSPI_SR_TDF_SHIFT==0);
                                   /* Wait for Tx FIFO available */
  LPSPI1->TDR = send;              /* Transmit data */
  LPSPI1->SR |= LPSPI_SR_TDF_MASK; /* Clear TDF flag */
}

uint16_t LPSPI1_receive_16bits (void)
  { uint16_t receive = 0;
  while((LPSPI1->SR & LPSPI_SR_RDF_MASK)>>LPSPI_SR_RDF_SHIFT==0);
                                   /* Wait at least one RxFIFO entry */
  receive= LPSPI1->RDR;            /* Read received data */
  LPSPI1->SR |= LPSPI_SR_RDF_MASK; /* Clear RDF flag */
  return receive;                  /* Return received data */
}
```

2.8.3.3 clocks_and_modes.c

See code in clocks_and_modes.c of the Hello World + Clock example.

2.9 CAN 2.0

2.9.1 描述

摘要：基于8 MHz晶体，FlexCAN模块初始化为500 KHz（2 usec周期）位时间。消息缓冲区0发送8字节消息，消息缓冲区4可以接收8字节消息。

本示例旨在将两个EVB连接在一起，“节点A”和“节点B”。初始化节点A后，它将发送一条初始消息。节点A然后循环：等待从节点B接收消息，然后将一条消息发送回。在节点B初始化之后，它将等待从节点a接收消息，然后将消息发送回。

最初的电动车辆总线（现已过时）使用收发器MC33903。规范和SBC操作，以供设计部分参考。

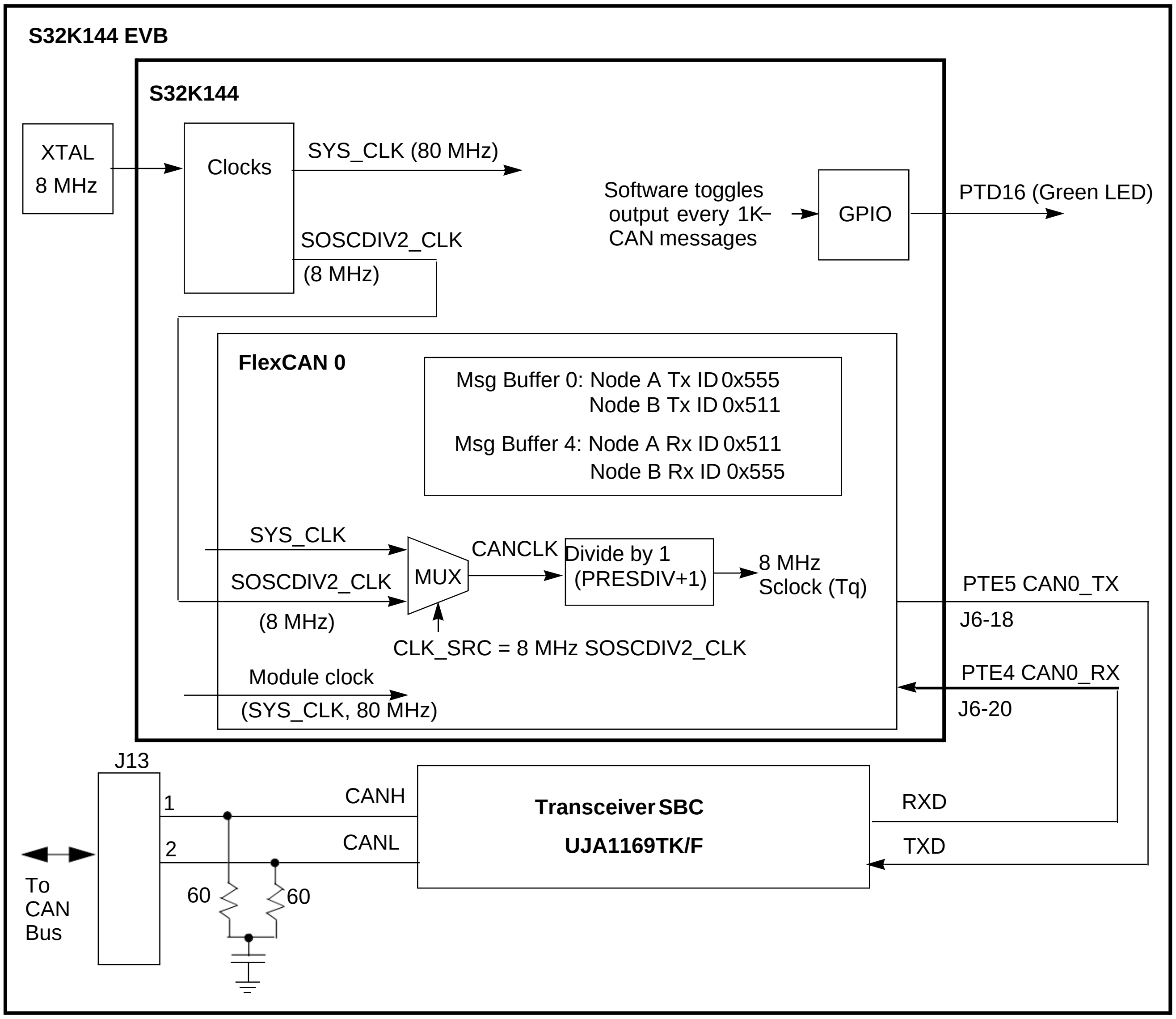


图11.CAN示例框图。如果使用两块板，节点A和B发送/接收不同的ID

2.9.2 设计

2.9.2.1 CAN 2.0时序计算

这些通用准则在CAN 2.0示例中用于满足500 KHz的比特率：

•CAN比特率周期通常细分为12－20个时间量程1。

•在整个比特率周期内，采样点通常选择在75%－80%左右。

•剩余的20－25%将是相位Seg2的值。

•相位Seg1的值将与相位Seg2相同。

•同步Seg为1倍量程。

•再同步跳转宽度（RJW+1）=相位Seg2（如果相位Seg2<4；否则（RJW+1）=4。对于本例和上述指南，这些是为CAN模块选择的值：

•每比特率周期的时间量子数=16

•采样点=75%，即16个时间量子周期中的12个时间量子，

阶段Seg2=（100%－75%） 16次量子=25% 16次量子=4次量子；PSEG2=3

相位Seg1=相位Seg2=4时间量子；PSEG1=3

项目Seg=16－阶段Seg1－阶段Seg2－同步Seg=16－4－4－1=7；PROPSEG=6再同步跳转宽度（RJW+1）=4

同样在本例中，以下内容适用于8 MHz晶体。fCANCLK=8 MHz（EVB振荡器）

因此,，

fSclock（时间量子频率）=（16时间量子/比特率周期） (500 K比特率周期/秒）=8MHz预分频器值（PRESDIV+1）=fCANCLK/fSclock=8 MHz/8 MHz=1

PRESDIV=1－1=0

表8.CAN 2FD示例定时段摘要。Sclock=8 MHz，比特率=500 KHz。

	SYNCSEG Time Quanta	PROP_SEG Time Quanta	PHASE_SEG1 Time Quanta	PHASE_SEG2 Time Quanta	Number of Time Quanta per bit time
CAN 2.0 Time Quanta	1	7	4	4	16
CAN_CTRL1 register bit fields	–	PROPSEG = 6	PSEG1 = 3	PSEG 2 = 3	-

1.“quantum” is the singular term; “quanta” is the plural term.

2.9.2.2 CAN 2.0消息缓冲区结构

下面是CAN 2.0消息缓冲区。NXP FlexCAN头文件实现了一种字而不是字节的结构，其中一个字是四个字节。

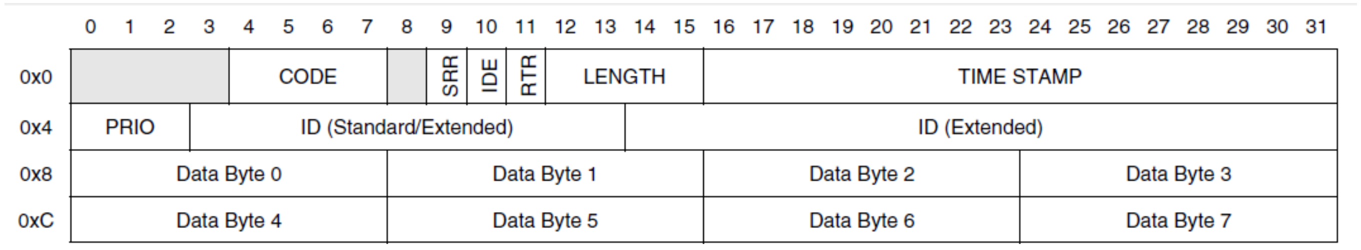


图12.CAN 2.0消息缓冲区结构（版本2.0 B部分）

2.9.2.3 设计步骤

•禁用看门狗

•将SOSC初始化为8 MHz，将sysclk初始化为80 MHz，并将正常运行模式时钟切换为SPLL

•初始化FlexCAN 0：

— 启用时钟到模块

— 选择OSC作为时钟源

— 配置500 KHz发送/接收的位定时

— 使所有消息缓冲区无效

— 设置传入掩码和全局掩码位以检查接收消息的所有ID位

— 为接收配置消息缓冲区4，ID 0x556，标准ID

— 否定32个消息缓冲区的模块暂停状态

•初始化端口引脚：

— 启用端口E的时钟

— PTE4、PTE5：配置为CAN0\U接收、CAN0\U发送

— 如果SBC为MC33903，则启用端口B的时钟，并为LSPI1配置端口引脚PTB14:PTB17

•如果收发器SBC为MC33903，则初始化LPSPI和SBC以进行CAN收发器操作

仅节点A：使用消息缓冲区0（标准ID 0x555）传输一条消息

注意

如果CAN收发器未通电或需要初始化，则不会看到传输的CAN帧，因为CAN_RX不会看到CAN_TX信号。如果不使用收发器，CAN0_-TX和CAN0_-RX可以跨接在一起。如果没有响应，节点将继续传输。

- 环路：

— 如果设置了消息缓冲区4接收消息标志，则读取消息

— 如果设置了消息缓冲区0传输完成标志，则发送另一条消息

2.9.2.4 屏幕截图

下面的屏幕截图显示了使用ID=0x555的CAN传输帧的开始，如下所示：

- 在游标之前：1个同步位（0）、11个标准ID位（0x555=0b101 0101 0101）、RTR、IDE、r0（3个0）
- 游标之间（8个usec=每个游标2个usec的4位时间）：数据长度控制：（0x8或0b1000）
- 第二个光标后：数据（0xA，···）或0xb1010...，依此类推。）

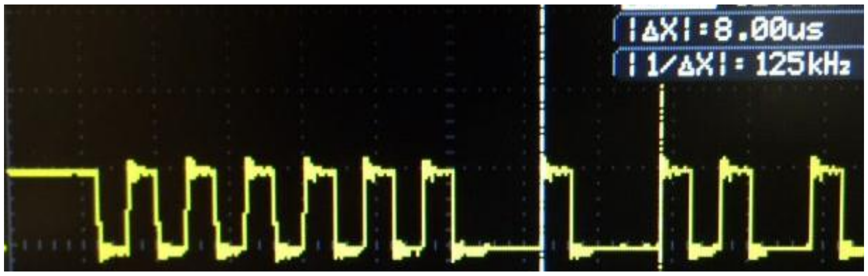


图13.标准ID为0x555的CAN 2.0传输帧开始处的TxD引脚

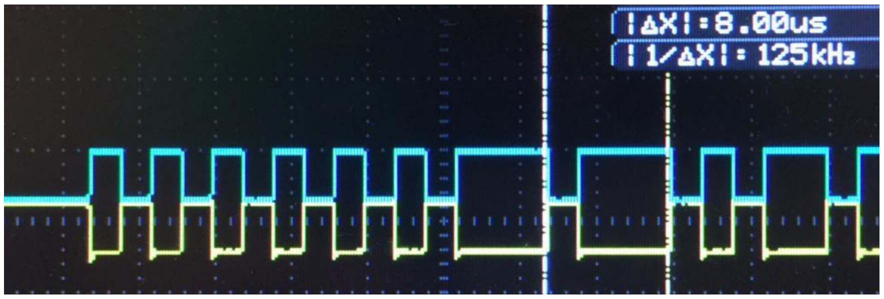
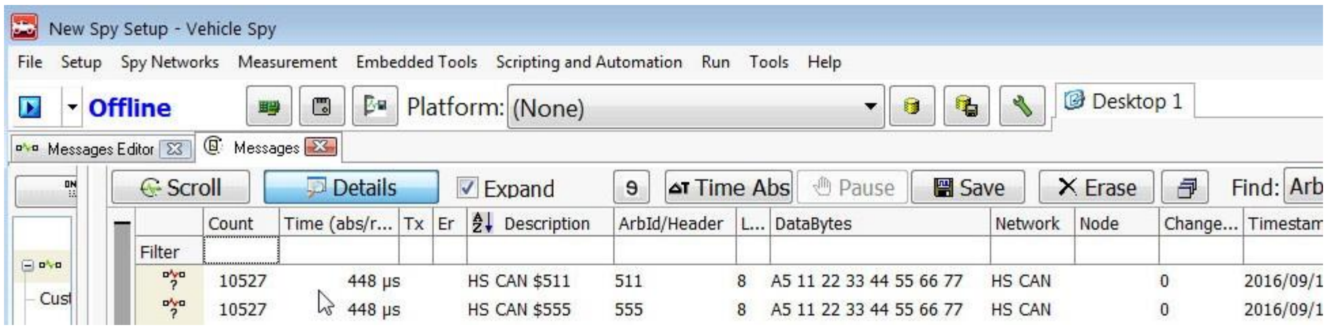


图14.标准ID为0x555的CAN 2.0传输帧开始时的CANH、CANL

以下屏幕截图显示了使用Intrepid控制系统的Vehicle Spy CAN工具在两个EVB节点之间的CAN帧结果。


New Spy Setup - Vehicle Spy
File Setup Spy Networks Measurement Embedded Tools Scripting and Automation Run Tools Help
Offline
Platform: (None)
Desktop 1
Messages Editor
Messages
Scroll
Details
Expand
Time Abs
Pause
Save
Erase
Find: Arb
Filter
Cust

Count	Time (abs/r...	Tx	Er	Description	ArbId/Header	L...	DataBytes	Network	Node	Change...	Timestam
10527	448 µs			HS CAN $511	511	8	A5 11 22 33 44 55 66 77	HS CAN		0	2016/09/1
10527	448 µs			HS CAN $555	555	8	A5 11 22 33 44 55 66 77	HS CAN		0	2016/09/1

图15.发送和接收的CAN帧

2.9.2.5 示例操作

要使用两块板运行示例，请执行以下步骤：

1. 如图所示，用电缆连接两块板之间的CAN高、CAN低和接地
2. 将12 V电源连接到两个板上
3. 根据下面的箭头，移动电源选择跳线以使用外部12 V（远离CAN接头）。

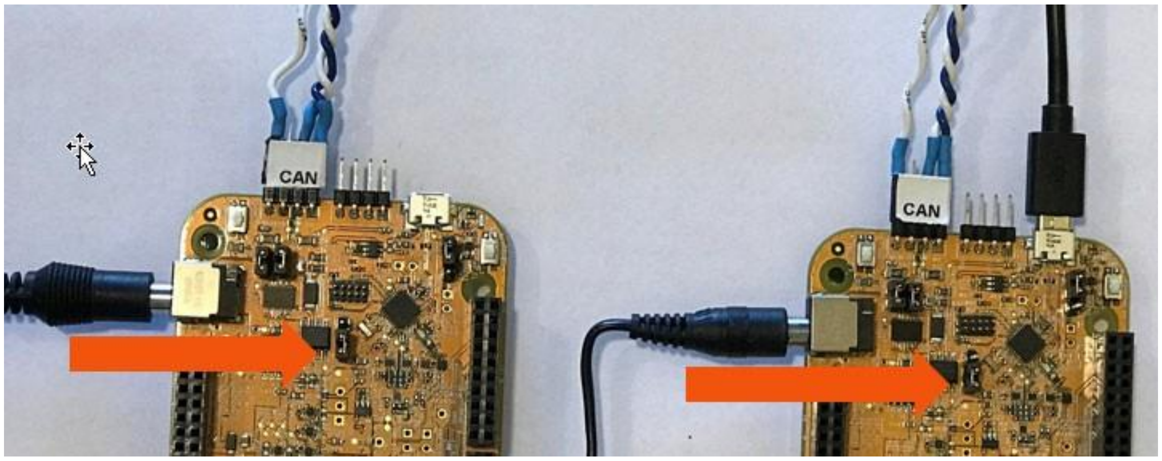


图16.CAN示例的连接和跳线设置

4. 通过注释FlexCAN.h中的行来配置节点B的代码，如下所示：
   //#定义节点A
5. 制定计划
6. “节点B”EVB的闪存程序
7. 通过取消对FlexCAN.h中的行的注释来配置节点A的代码，如下所示：#定义节点A
8. 制定计划
9. “节点A”EVB的闪存程序
10. 在两块板都通电的情况下，启动节点A上的程序，启动传输序列。

2.9.2.6 参考: MC33903 SBC

初始评估板（板背面的示意图SCH-28810标签）使用SBC MC33903，需要通过SPI配置CAN和LIN接口。可以使用以下最小简单序列。

表9.SBC MC33903C通过SPI发送的命令和预期接收状态

Step	Transmit Command	Transmitted Description	Expected Receive Status	Received Description
Verify SPI communication (by reading device ID[1])	0x2580	Read SAFE register: • Lower 5 bits of returned data *Reference: p. 85 & 90, MC33903_4_5 Data Sheet, Rev 12.0, 8/2016*	Non zero. (Example: 0x00F4)	Read command returned flags: • Upper byte (Fixed status): • Lower byte (Register status): - bit 7 (0b1):Vdd is 5 V - bits 6:5 (0b11): part # - bits 4:0 (0b10100):device ID
Read possible reset causes	0xDF80	Read REG H (Regulator High) register to read upper 8 status bits • Assume BATFAIL is high, so device requires initialization • Reading register clears flags *Reference: p. 85, MC33903_4_5 Data Sheet, Rev 12.0, 8/2016*	0x0000	Read command returned flags: • Upper byte (Fixed status): • Lower byte (Upper 8 bits of register status): (If bit BATFAIL is low, then the previous initialization would still be intact.)
Enable configuring CAN and LIN (by transitioning to normal mode)	0x5A00	Write to Watchdog Refresh register: • Transitions from init to normal mode (Note: EVB HW disables SBC watchdog) *Reference: p. 75, MC33903_4_5 Data Sheet, Rev 12.0, 8/2016*	0x0000	Write command returned flags: • Upper byte (Fixed status): • Lower byte (Extended status):
Enable CAN voltage regulator	0x5E10	Write to REG (Regulator) register: • Turn on 5 V-CAN regulator *Reference: p. 79, MC33903_4_5 Data Sheet, Rev 12.0, 8/2016*	0x0000	Write command returned flags: • Upper byte (Fixed status): • Lower byte (Extended status):
Enable CAN Tx & Rx	0x60C0	Write to CAN register: • Enable Tx & Rx modes, • Fast CAN slew rate • 3 dominant pulses wake up, INT generation after 5 dominant pulses *Reference: p80. , MC33903_4_5 Data Sheet, Rev 12.0, 8/2016*	0x0000	Write command returned flags: • Upper byte (Fixed status): • Lower byte (Extended status):
Enable LIN for Tx & Rx	0x66C4	Write to LIN/1 register • Tx/Rx mode • Slew rate for 20 Kbits/s • Termination on • Recessive when Vsup2 type < 6 V *Reference: p. 83, MC33903_4_5 Data Sheet, Rev 12.0, 8/2016*	0x0000	Write command returned flags: • Upper byte (Fixed status): • Lower byte (Extended status):

[1] NOTE: If the received status is 0x0000, then likely the device is not powered. Ensure 12 V is connected to the EVB and the power source selection jumper J107 has pins 1-2 connected (default is 2-3).

2.9.3 代码

2.9.3.1 main.c

```
#include "S32K144.h" /* include peripheral declarations S32K144 */
#include  "FlexCAN.h"
#include "clocks_and_modes.h"
void WDOG_disable (void){
  WDOG->CNT=0xD928C520;    /* Unlock watchdog */
  WDOG->TOVAL=0x0000FFFF; /* Maximum timeout value */
  WDOG->CS = 0x00002100;    /* Disable watchdog */
}
void PORT_init (void) {
  PCC->PCCn[PCC_PORTE_INDEX] |= PCC_PCCn_CGC_MASK; /* Enable clock for PORTE */
  PORTE->PCR[4] |= PORT_PCR_MUX(5); /* Port E4: MUX = ALT5, CAN0_RX */
  PORTE->PCR[5] |= PORT_PCR_MUX(5); /* Port E5: MUX = ALT5, CAN0_TX */
  PCC->PCCn[PCC_PORTD_INDEX ]|=PCC_PCCn_CGC_MASK;   /* Enable clock for PORTD */
  PORTD->PCR[16] =  0x00000100;      /* Port D16: MUX = GPIO (to green LED) */
  PTD->PDDR |= 1<<16;                /* Port D16: Data direction = output */
}
int main(void) {
  uint32_t rx_msg_count = 0;
  WDOG_disable();
  SOSC_init_8MHz();         /* Initialize system oscillator for 8 MHz xtal */
  SPLL_init_80MHz();        /* Initialize SPLL to 80 MHz with 8 MHz SOSC */
  NormalRUNmode_80MHz();    /* Init clocks: 80 MHz SPLL & core, 40 MHz bus, 20 MHz flash */
  FLEXCAN0_init();          /* Init FlexCAN0 */
  PORT_init();              /* Configure ports */
#ifdef NODE_A               /* Node A transmits first; Node B transmits after reception */
  FLEXCAN0_transmit_msg(); /* Transmit initial message from Node A to Node B */
#endif
  for (;;) {                         /* Loop: if a msg is received, transmit a msg */
    if ((CAN0->IFLAG1 >> 4) & 1) {  /* If CAN 0 MB 4 flag is set (received msg), read MB4 */
      FLEXCAN0_receive_msg ();       /* Read message */
      rx_msg_count++;                /* Increment receive msg counter */
      if (rx_msg_count == 1000) { /* If 1000 messages have been received, */
        PTD->PTOR  |=  1<<16;  /*  toggle  output  port  D16  (Green  LED)  */
        rx_msg_count = 0;            /* and reset message counter */
      }
      FLEXCAN0_transmit_msg ();      /* Transmit message using MB0 */
    }
}
```

2.9.3.2 FlexCAN.c

```
#include "S32K144.h" /* include peripheral declarations S32K144 */
#include "FlexCAN.h"
uint32_t  RxCODE;              /* Received message buffer code */
uint32_t  RxID;                /* Received message ID */
uint32_t  RxLENGTH;            /* Received message number of data bytes */
uint32_t  RxDATA[2];           /* Received message data (2 words) */
uint32_t  RxTIMESTAMP;         /* Received message time */

void FLEXCAN0_init(void) {
#define MSG_BUF_SIZE  4    /* Msg Buffer Size. (CAN 2.0AB: 2 hdr +  2 data= 4 words) */
  uint32_t   i=0;
  PCC->PCCn[PCC_FLEXCAN0_INDEX] |= PCC_PCCn_CGC_MASK; /* CGC=1: enable clock to FlexCAN0 */
  CAN0->MCR |= CAN_MCR_MDIS_MASK;         /*MDIS=1: Disable module before selecting clock */
  CAN0->CTRL1 &= ~CAN_CTRL1_CLKSRC_MASK;  /* CLKSRC=0: Clock Source = oscillator (8 MHz) */
  CAN0->MCR &= ~CAN_MCR_MDIS_MASK;        /* MDIS=0; Enable module config. (Sets FRZ, HALT)*/
  while (!((CAN0->MCR & CAN_MCR_FRZACK_MASK) >> CAN_MCR_FRZACK_SHIFT))  {}
                 /* Good practice: wait for FRZACK=1 on freeze mode entry/exit */
  CAN0->CTRL1 = 0x00DB0006; /* Configure for 500 KHz bit time */
                            /* Time quanta freq = 16 time quanta x 500 KHz bit time= 8MHz */
                            /* PRESDIV+1 = Fclksrc/Ftq = 8 MHz/8 MHz = 1 */
                            /*    so PRESDIV = 0 */
                            /* PSEG2 = Phase_Seg2 - 1 = 4 - 1 = 3 */
                            /* PSEG1 = PSEG2 = 3 */
                            /* PROPSEG= Prop_Seg - 1 = 7 - 1 = 6 */
                            /* RJW: since Phase_Seg2 >=4, RJW+1=4 so RJW=3. */
                            /* SMP = 1: use 3 bits per CAN sample */
                            /* CLKSRC=0 (unchanged): Fcanclk= Fosc= 8 MHz */
  for(i=0; i<128; i++ ) {   /* CAN0: clear 32 msg bufs x 4 words/msg buf = 128 words*/
    CAN0->RAMn[i] = 0;      /* Clear msg buf word */
  }
  for(i=0; i<16; i++ ) {          /* In FRZ mode, init CAN0 16 msg buf filters */
    CAN0->RXIMR[i] = 0xFFFFFFFF;  /* Check all ID bits for incoming messages */
  }
  CAN0->RXMGMASK = 0x1FFFFFFF;  /* Global acceptance mask: check all ID bits */
  CAN0->RAMn[ 4*MSG_BUF_SIZE + 0] = 0x04000000; /* Msg Buf 4, word 0: Enable for reception */
                                                /* EDL,BRS,ESI=0: CANFD not used */
                                                /* CODE=4: MB set to RX inactive */
                                                /* IDE=0: Standard ID */
                                              /* SRR, RTR, TIME STAMP = 0: not applicable */
#ifdef NODE_A                                   /* Node A receives msg with std ID 0x511 */
  CAN0->RAMn[ 4*MSG_BUF_SIZE + 1] = 0x14440000; /* Msg Buf 4, word 1: Standard ID = 0x111 */
#else                                             /* Node B to receive msg with std ID 0x555 */
  CAN0->RAMn[ 4*MSG_BUF_SIZE + 1] = 0x15540000; /* Msg Buf 4, word 1: Standard ID = 0x555 */
#endif
                                                   /* PRIO = 0: CANFD not used */
  CAN0->MCR = 0x0000001F;       /* Negate FlexCAN 1 halt state for 32 MBs */
  while ((CAN0->MCR && CAN_MCR_FRZACK_MASK) >> CAN_MCR_FRZACK_SHIFT)  {}
                 /* Good practice: wait for FRZACK to clear (not in freeze mode) */
  while ((CAN0->MCR && CAN_MCR_NOTRDY_MASK) >> CAN_MCR_NOTRDY_SHIFT)  {}
                 /* Good practice: wait for NOTRDY to clear (module ready)  */
}

void FLEXCAN0_transmit_msg(void) { /* Assumption: Message buffer CODE is INACTIVE */
  CAN0->IFLAG1 = 0x00000001; /* Clear CAN 0 MB 0 flag without clearing others*/ CAN0-
  >RAMn[ 0*MSG_BUF_SIZE + 2] = 0xA5112233; /* MB0 word 2: data word 0 */
  CAN0->RAMn[ 0*MSG_BUF_SIZE + 3] = 0x44556677; /* MB0 word 3: data word 1 */
#ifdef NODE_A
  CAN0->RAMn[ 0*MSG_BUF_SIZE + 1] = 0x15540000; /* MB0 word 1: Tx msg with STD ID 0x555 */
#else
  CAN0->RAMn[ 0*MSG_BUF_SIZE + 1] = 0x14440000; /* MB0 word 1: Tx msg with STD ID 0x511 */
#endif
  CAN0->RAMn[ 0*MSG_BUF_SIZE + 0] = 0x0C400000 | 8 <<CAN_WMBn_CS_DLC_SHIFT; /* MB0 word 0: */
                                                   /* EDL,BRS,ESI=0: CANFD not used */
                                                  /* CODE=0xC: Activate msg buf to transmit */
                                                   /* IDE=0: Standard ID */
                                                   /* SRR=1 Tx frame (not req'd for std ID) */
                                                /* RTR = 0: data, not remote tx request frame*/
                                                   /* DLC = 8 bytes */
}
void FLEXCAN0_receive_msg(void) {  /* Receive msg from ID 0x556 using msg buffer 4 */
  uint8_t j;
  uint32_t dummy;

  RxCODE   = (CAN0->RAMn[ 4*MSG_BUF_SIZE + 0] & 0x07000000) >> 24;  /* Read CODE field */
  RxID     = (CAN0->RAMn[ 4*MSG_BUF_SIZE + 1] & CAN_WMBn_ID_ID_MASK) >>CAN_WMBn_ID_ID_SHIFT ;
  RxLENGTH =(CAN0->RAMn[ 4*MSG_BUF_SIZE + 0] & CAN_WMBn_CS_DLC_MASK)>>CAN_WMBn_CS_DLC_SHIFT;
  for (j=0; j<2; j++) {  /* Read two words of data (8 bytes) */
    RxDATA[j] = CAN0->RAMn[ 4*MSG_BUF_SIZE + 2 + j];
  }
  RxTIMESTAMP = (CAN0->RAMn[ 0*MSG_BUF_SIZE + 0] & 0x000FFFF);
  dummy = CAN0->TIMER;            /* Read TIMER to unlock message buffers */
  CAN0->IFLAG1 = 0x00000010;      /* Clear CAN 0 MB 4 flag without clearing others*/
}
```

2.9.3.3 FlexCAN.h (Partial listing)

```
#define NODE_A        /* If using 2 boards as 2 nodes, NODE A & B use different CAN IDs */
```

2.9.3.4 clocks_and_modes.c

See code in clocks_and_modes.c of the Hello World + Clock example.

2.9.3.5 Reference: MC33903 code for obsolete EVB (not included in project)

LPSPI1 initialization, transmit and receiver functions: See code for functions in SPI example.

Port pin initialization:

```
#ifdef SBC_MC33903  /* If board has MC33904, SPI pin config. is required */
  PCC->PCCn[PCC_PORTB_INDEX] |= PCC_PCCn_CGC_MASK; /* Enable clock for PORTB */
  PORTB->PCR[14] |= PORT_PCR_MUX(3);  /* Port B14: MUX = ALT3, LPSPI1_SCK */
  PORTB->PCR[15] |= PORT_PCR_MUX(3);  /* Port B15: MUX = ALT3, LPSPI1_SIN */
  PORTB->PCR[16] |= PORT_PCR_MUX(3);  /* Port B16: MUX = ALT3, LPSPI1_SOUT */
  PORTB->PCR[17] |= PORT_PCR_MUX(3);  /* Port B17: MUX = ALT3, LPSPI1_PCS3 */
#endif
```

MC33903 initialization using LPSPI1:

```
void LPSPI1_init_MC33903(void) {
  uint32_t i = 0;                  /* Loop counter */
  uint16_t MC33903_spi_init[] = { /* SPI commands and data to initialize MC33903C */ 0x2580,
                                /* Read SAFE register flags: bits 4:0 contain nonzero ID */
    0xDF80,                     /* Read Vreg High flags:  */
    0x5A00,                     /* Write Watchdog reg.: Enter NORMAL mode*/
    0x5E10,                     /* Write Regulator reg.: Enable 5V CAN regulator */
    0x60C0,                     /* Write CAN reg.: CAN in Tx & Rx modes, fast slew */
    0x66C4};                    /* Write LIN/1 reg.: Tx/Rx mode, 20 Kbps slew, term. on */
  uint16_t spi_result = 0;      /* Result received SPI data from SBC */
                             /* Note: MC33904 DBG input on EVB is tied to 9V nominal, */
                             /*       which puts device in a debug state */
                             /*       which disables the SBC's watchdog. */
  for (i=0; i< sizeof (MC33903_spi_init)/2; i++) {
    LPSPI1_transmit_16bits (MC33903_spi_init[i]);   /* Transmit to MC33904 */
    spi_result =  LPSPI1_receive_16bits();          /* Read result */
                             /* Note: It is good practice to verify SPI configuration by */
                             /*       reading appropriate flags/registers, especially */
                             /*       fault flags, after configuration routines. */
  }
}
```

2.10 CAN FD

2.10.1 描述

摘要：该示例将先前的经典CAN示例移植到CAN FD，数据有效负载增加，CAN FD数据相位位时间为2 MHz[1]。消息大小增加到64字节。

本示例旨在将两个EVB连接在一起，“节点A”和“节点B”。初始化节点A后，它将发送一条初始消息。节点A然后循环：等待从节点B接收消息，然后将一条消息发送回。在节点B初始化之后，它将等待从节点a接收消息，然后将消息发送回。

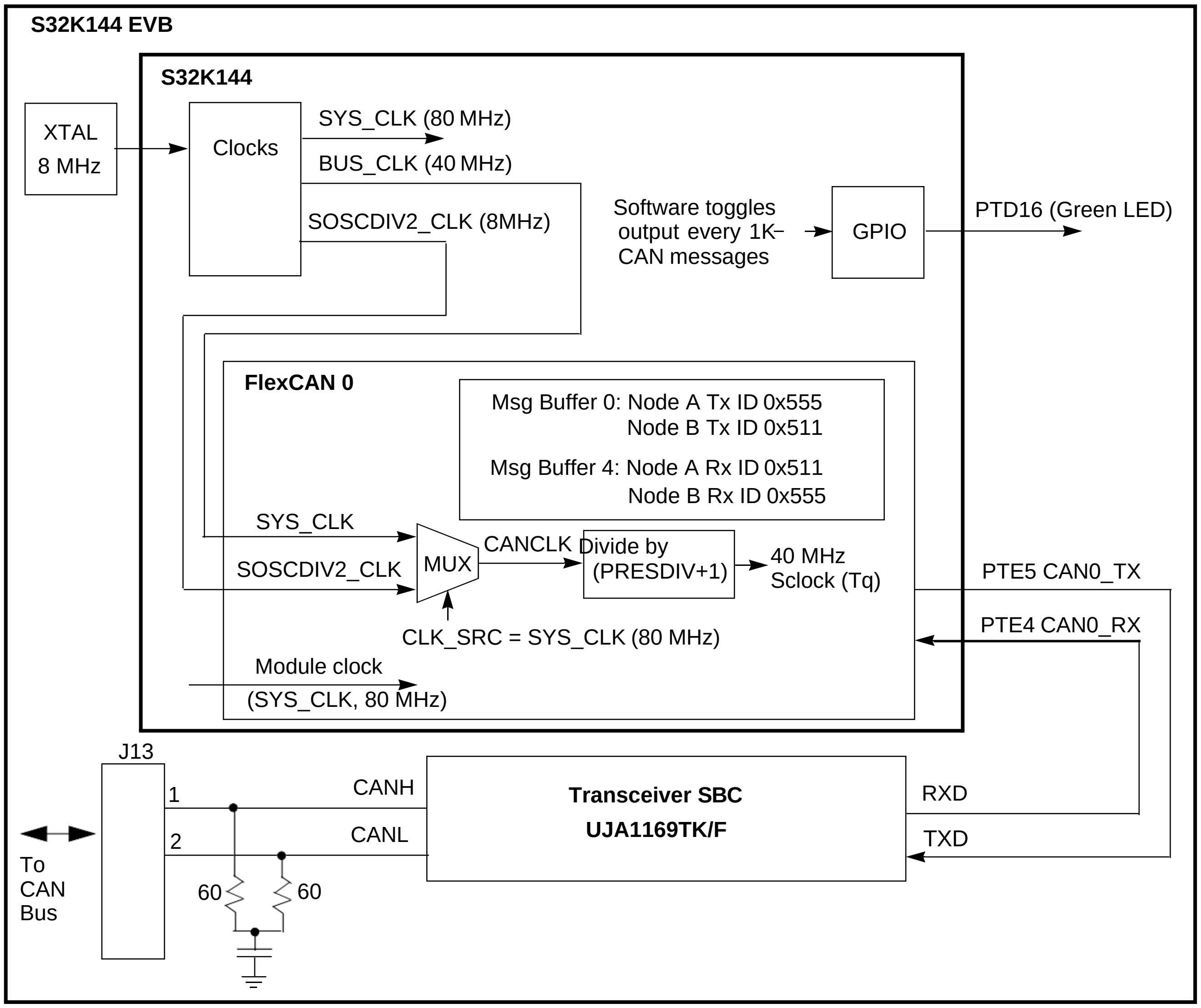


图17.CAN FD示例框图。节点A和B发送/接收不同的ID

1. Initial S32K144 EVBs use MC33903C transceiver, which is not rated for High Speed CAN bit times, not the faster CAN FD bit times. If using this transceiver, data phase timing is limited to 1 MHz. See code in Design section for reference.

2.10.2 设计

2.10.2.1 CAN 2.0与CAN FD初始化摘要

下面是用于这些示例的Flextime模块初始化的简要比较。一目了然，不同之处在于不同的位位时序和增加了用于收发器延迟补偿、有效负载大小、CAN FD和CAN FD ISO fix的控制。

表10.经典CAN与CAN FD示例的FlexCAN模块初始化摘要

Initialization	Classic CAN registers, memory	CAN FD registers, memory
Clock source selection	CAN_CTRL1, CAN_MCR	CAN_CTRL1, CAN_MCR
CAN bit timing	CAN_CTRL1	Nominal phase: CAN_CBT Data phase: CAN_FDCBT
Transceiver Delay Compensation	—	CAN_FDCTRL
Payload (message buffer data) size	—	CAN_FDCTRL
Message buffers inactivation	RAMn	RAMn (optional larger sizes)
Tx and Rx message buffer configuration	RAMn	RAMn (optional larger sizes)
Enable ISO version CAN FD (CRC fix)	—	CAN_CTRL2 [STFCNTEN]
Enable CAN FD	—	CAN_MCR
Negate halt state	CAN_MCR	CAN_MCR

2.10.2.2 CAN FD位时序

启用CAN FD时（CAN_MDR[FDEN]=1），必须使用CAN_CBT中的扩展位定时变量替换CAN_CTRL1中的位定时变量。此示例为500 KHz标称位时间速率配置扩展位定时变量（在标称相位）。

如果将使用CAN FD比特率开关（BPS）功能来提高数据阶段的比特率，则还必须使用CAN_FDCBT中的快速比特定时变量。此示例为2 MHz数据比特率配置快速定时变量（数据相位）。

时钟源选择：

- CAN比特率周期细分为称为时间量程1的单位
- 每个CAN比特率的最小时间量子数为8
- CANCLK源：对于2 Mhz比特时间，最小fCANCLK为8 x 2 Mhz=16 Mhz
  - — CAN_-CTRL1[CLKSRC]=SYS_-CLK（因为SOSC只有8 MHz）

标称相位和数据相位的预分频器:

- 使用的预分频器值为2：
  - — Sclock=CAN_时钟/预分频器=80 MHz/2=40 MHz
  - — CAN_CBT[EPRESDIV]=CAN_FDCBT[FDPRESDIV]=预分频器-1=1

表11.CAN FD示例定时段摘要。Sclock=40 MHz，预分频器=除以1。

Phase and Applicable Register	Bit Rate	Time Quanta per bit time (Sclock / Bit Rate)	SYNCSEG Time Quanta	PROP_SEG Time Quanta	PHASE_SEG1 Time Quanta	PHASE_SEG2 Time Quanta	Resync Jump Width Time Quanta
Nominal Phase CAN_CBT	500 KHz	40MHz / 500KHz = 80 tq	1	47	16	16	16
		—	—	EPROPSEG = 46	EPSEG1 = 15	EPSEG2 = 15	ERJW = 15
Data Phase CAN_FDCBT	2 MHz	40 Mhz / 2 MHz = 20 tq	1	7	8	4	4
		—	—	FPROPSEG = 7	FPSEG1 = 7	FPSEG 2 = 3	FRJW = 3

2.10.2.3 传输延迟补偿（TDC）

通过验证传输模块CAN接收引脚上的传输位是否“听到”，检查传输位是否存在错误。在发送比特时间期间，使用允许收发器中的正常延迟的采样点对接收状态进行采样，以将发送比特“循环”回到接收输入。

在CAN FD帧的数据阶段，如果比特率较高，实际一位的时间可能比收发器循环时间短，从而导致检测到意外错误。为了补偿这一点，可以通过启用FDCTRL[TDCEN]使用辅助采样点来代替正常采样点。这个新的采样点是硬件测量的环路延迟和FDCTRL[TDCOFF]中定义的软件定义的偏移时间之和。

例如，可以在大约50%的位时间选择采样点。可以从收发器数据表中得出环路时间的估计值。TDC基于CANCLK的数量，在本例中，40 MHz CANCLK为25纳秒（如果CANCLK为8 MHz OSC，则为125纳秒）。

根据MC33903，本例的TDC偏移确定如下：

50%位时间=收发器中测量环路延迟的估计值+软件定义的偏移量

=MC33903 tLDR（典型值，传播环路延迟，快速转换）+TDCOFF

=120纳秒+TDCOFF

对于1 MHz数据相位，偏移量为：

TDCOFF=（50%x1000纳秒）-120秒=500纳秒-120纳秒=380纳秒

~= 15 CANCLKs (@25 纳秒 per CANCLK)

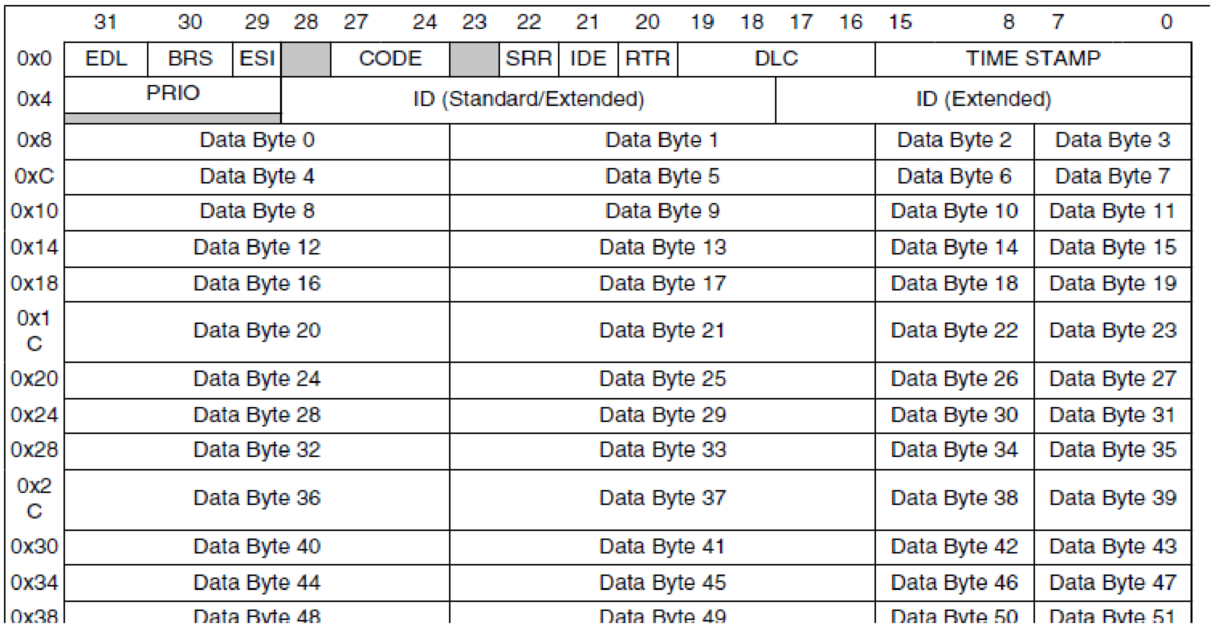
对于2 MHz数据相位，偏移量为：

TDCOFF = (50% x 500 纳秒) - 120 纳秒 = 250 纳秒 - 120 纳秒 = 130 纳秒

~= 5 CANCLKs (@25 nsec per CANCLK)

2.10.2.4 CAN FD消息缓冲区结构

FD帧可以具有附加控件和可选附加数据。这些附加内容反映在下面的CAN FD消息缓冲区结构中。



	31	30	29	28	27 24	23	22	21	20	19	18	17	16	15 8	7 0
0x0	EDL	BRS	ESI	▨	CODE	▨	SRR	IDE	RTR	DLC				TIME STAMP	
0x4	PRIO			ID (Standard/Extended)								ID (Extended)			
0x8	Data Byte 0					Data Byte 1								Data Byte 2	Data Byte 3
0xC	Data Byte 4					Data Byte 5								Data Byte 6	Data Byte 7
0x10	Data Byte 8					Data Byte 9								Data Byte 10	Data Byte 11
0x14	Data Byte 12					Data Byte 13								Data Byte 14	Data Byte 15
0x18	Data Byte 16					Data Byte 17								Data Byte 18	Data Byte 19
0x1C	Data Byte 20					Data Byte 21								Data Byte 22	Data Byte 23
0x20	Data Byte 24					Data Byte 25								Data Byte 26	Data Byte 27
0x24	Data Byte 28					Data Byte 29								Data Byte 30	Data Byte 31
0x28	Data Byte 32					Data Byte 33								Data Byte 34	Data Byte 35
0x2C	Data Byte 36					Data Byte 37								Data Byte 38	Data Byte 39
0x30	Data Byte 40					Data Byte 41								Data Byte 42	Data Byte 43
0x34	Data Byte 44					Data Byte 45								Data Byte 46	Data Byte 47
0x38	Data Byte 48					Data Byte 49								Data Byte 50	Data Byte 51

图18.CAN FD消息缓冲区结构（64字节有效负载的部分列表)

2.10.2.5 访问消息缓冲区

消息缓冲区在RAM阵列中实现。对于CAN FD，RAM阵列到消息缓冲区开始的索引取决于消息缓冲区数据大小的大小，如在寄存器CAN_FDCTRL[MBDSRx]位字段中配置的。

NXP提供的头文件将消息缓冲RAM定义为4字节字数组。下表说明了各种数据有效负载大小的消息缓冲区开始索引以及S32K144 FlexCAN 0的该有效负载的最大消息缓冲区数量。

表12.S32K144 FlexCAN0:RAMn字索引到消息缓冲区的开始（MB）。

RAM Index (words, 4B each)	8 Bytes Data (MSBDRx = 0)		16 Bytes Data (MSBDRx = 1)		32 Bytes Data (MSBDRx = 2)		64 Bytes Data (MSBDRx = 3)	
	MB # (32 max.)	Frame	MB # (20 max.)	Frame	MB # (11 max.)	Frame	MB # (6 max.)	Frame
0x0	MB0	8B Hdr.	MB0	8B Hdr.	MB0	8B Hdr.	MB0	8B Hdr.
0x2		8B Data		16B Data		32B Data		64B Data
0x4	MB1	8B Hdr.						
0x8		8B Data	MB1	8B Hdr.				
0xA	MB2	8B Hdr.		16B Data				
0xC		8B Data			MB1	8B Hdr.		
0xE	MB3	8B Hdr.	MB2	8B Hdr.		32B Data		
0x10		8B Data		16B Data				
0x12	MB4	8B Hdr.						
0x14		8B Data	MB4	8B Hdr.			MB1	8B Hdr.
0x16	MB5	8B Hdr.		16B Data	MB2	8B Hdr.		64B Data
0x18		8B Data				32B Data		
0x1A	MB6	8B Hdr.	MB5	8B Hdr.				
0x1C		8B Data		16B Data				
etc.								

表13.S32K144 FlexCAN 0:CAN FD最大消息缓冲区数量

8 Bytes Data (MSBDRx = 0)	16 Bytes Data (MSBDRx = 1)	32 Bytes Data (MSBDRx = 2)	64 Bytes Data (MSBDRx = 3)
32 Message Buffers	21 Message Buffers	12 Message Buffers	7 Message Buffers

要访问消息缓冲区中的一个字，请添加消息缓冲区开头的索引和消息缓冲区中的单词编号索引：

```
CANx->RAMn[(index of start of message buffer) + (index of word inside message buffer)]
```

示例：对于64字节有效负载，初始化消息缓冲区2的第二个数据字（索引值为3，因为头字是索引0、1，数据字是索引2）：

```
CANx->RAMn[(0x24 + 0x3)] = 0x12345678; /* MB2, 2nd data word */
```

可以使用MBDSRx位字段对代码进行泛化，以访问消息缓冲区的字。例子

```
#define MBDSR0 3 /* Msg Buffer Data Size for Region 0: 3 (64 bytes or 16 words) */
uint32_t msg_buf_size_r0; /* Message Buffer size, region 0, in words */

msg_buf_size_r0 = 2 + exp2(MBDSR0+1); /* Msg Buf Size Reg 0 = 2 hdr + 2**4 data words */

CAN0->RAMn[4*msg_buf_size_r0 + 0] |= 1<<31; /* MB4 word 0: set bit 31, EDL = 1 */
```

2.10.2.6 设计步骤

•禁用看门狗

•将SOSC初始化为8 MHz，将sysclk初始化为80 MHz，并将正常运行模式时钟切换为SPLL

•初始化FlexCAN 0：

— 启用时钟到模块

— 选择系统时钟（80 MHz）作为时钟源

— 为500 KHz标称相位和2 MHz数据相位的比特率配置比特定时

— 配置收发器延迟补偿

— 配置有效负载大小

— 禁用所有消息缓冲区

— 设置所需的发送和接收缓冲区

— 为ISO CAN FD启用CRC修复

— 为32个消息缓冲区启用CAN FD并取消模块暂停状态

•初始化端口引脚：

— 启用端口E的时钟，并将PTE4、PTE5配置为CAN0_RX、CAN0_TX

— 如果SBC为MC33903，则启用端口B的时钟，并为LSPI1配置端口引脚PTB14:PTB17

•如果收发器SBC为MC33903，则初始化LPSPI和SBC以进行CAN收发器操作

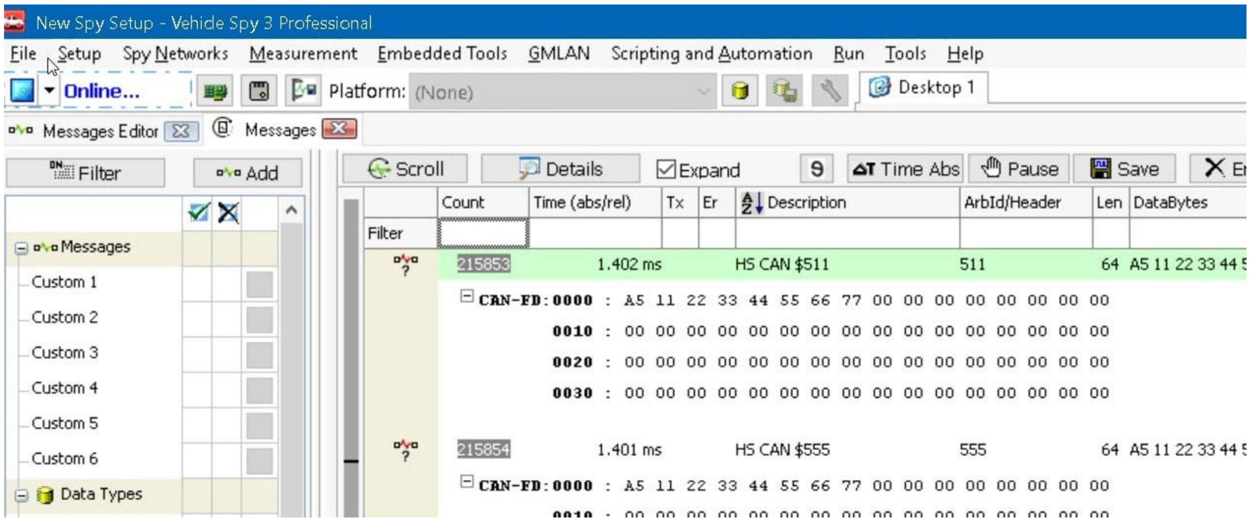
•仅节点A：使用消息缓冲区0（标准ID 0x555）传输一条消息

•环路：

— 如果设置了消息缓冲区4接收消息标志，则读取消息

— 如果设置了消息缓冲区0传输完成标志，则发送另一条消息

2.10.2.7 截图



	Count	Time (abs/rel)	Tx	Er	Description	ArbId/Header	Len	DataBytes
Filter								
	215853	1.402 ms			HS CAN $511	511	64	A5 11 22 33 44 5
	215854	1.401 ms			HS CAN $555	555	64	A5 11 22 33 44 5

```
CAN-FD:0000 : A5 11 22 33 44 55 66 77 00 00 00 00 00 00 00 00
       0010 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
       0020 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
       0030 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

CAN-FD:0000 : A5 11 22 33 44 55 66 77 00 00 00 00 00 00 00 00
```

图19.发送和接收的CAN FD帧（显示两条消息的部分屏幕截图）

2.10.2.8 示例操作

要使用两块板运行示例，请执行以下步骤：

1. 如图所示，用电缆连接两块板之间的CAN高、CAN低和接地
2. 将12V电源连接到两个板上
3. 移动电源选择跳线以使用外部12V（根据下面的箭头远离CAN接头）

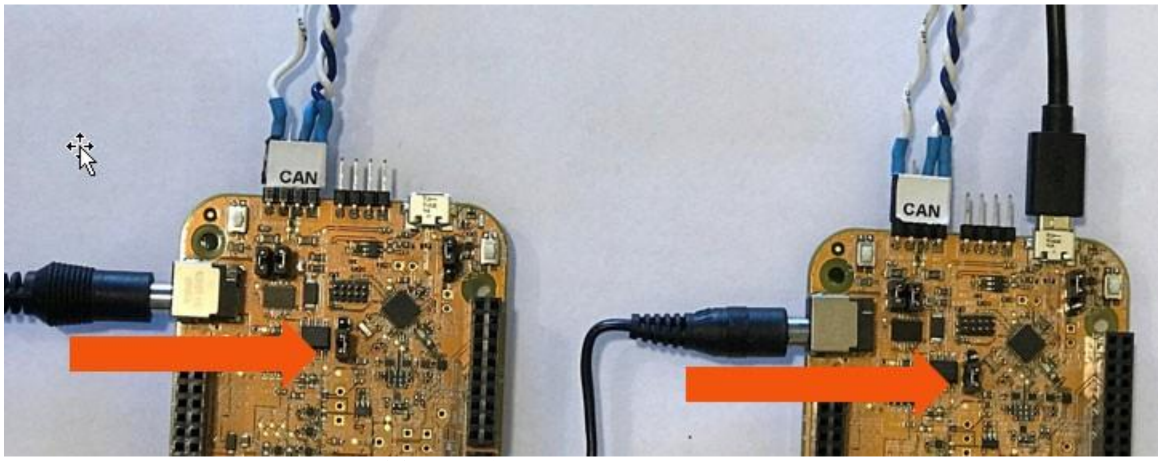


图20.CAN示例的连接和跳线设置

4. 通过注释FlexCAN.h中的行来配置节点B的代码，如下所示：

   //#define Node A

5. 编译工程
6. 烧写固件到“节点B”开发板
7. 通过取消对FlexCAN.h中的行的注释来配置节点A的代码，如下所示：

   #define Node A

8. 编译工程
9. 烧写固件到节点A的开发板

在两块板都通电的情况下，启动节点A上的程序，启动传输序列。板上的绿色LED将每发送和接收1000 个CAN数据包，就会闪烁一次。

2.10.3 代码

2.10.3.1 main.c

```
#include "S32K144.h" /* include peripheral declarations S32K144 */
#include "FlexCAN_FD.h"
#include "clocks_and_modes.h"
void WDOG_disable (void){
  WDOG->CNT=0xD928C520;    /* Unlock watchdog */
  WDOG->TOVAL=0x0000FFFF; /* Maximum timeout value */
  WDOG->CS = 0x00002100;    /* Disable watchdog */
}
void PORT_init (void) {
  PCC->PCCn[PCC_PORTE_INDEX] |= PCC_PCCn_CGC_MASK; /* Enable clock for PORTE */
  PORTE->PCR[4] |= PORT_PCR_MUX(5); /* Port E4: MUX = ALT5, CAN0_RX */
  PORTE->PCR[5] |= PORT_PCR_MUX(5); /* Port E5: MUX = ALT5, CAN0_TX */
  PCC->PCCn[PCC_PORTD_INDEX ]|=PCC_PCCn_CGC_MASK;   /* Enable clock for PORTD */
  PORTD->PCR[16] =  0x00000100;  /* Port D16: MUX = GPIO (to green LED) */
  PTD->PDDR |= 1<<16;            /* Port D16: Data direction = output */
}
int main(void) {
  uint32_t rx_msg_count = 0;
  uint32_t i = 0;             /* dummy delay variable */
  WDOG_disable();
  SOSC_init_8MHz();         /* Initialize system oscillator for 8 MHz xtal */
  SPLL_init_160MHz();        /* Initialize SPLL to 160 MHz with 8 MHz SOSC */
  NormalRUNmode_80MHz();   /* Init clocks: 80 MHz sysclk & core, 40 MHz bus, 20 MHz flash*/
  FLEXCAN0_init();          /* Init FlexCAN0 */
  PORT_init();              /* Configure ports */
#ifdef NODE_A               /* Node A transmits first; Node B transmits after reception */
  FLEXCAN0_transmit_msg(); /* Transmit initial message from Node A to Node B */
#endif
  for (;;) {                        /* Loop: if a msg is received, transmit a msg */
    if ((CAN0->IFLAG1 >> 4) & 1) {  /* If CAN 0 MB 4 flag is set (received msg), read MB4 */
      FLEXCAN0_receive_msg ();      /* Read message */
      rx_msg_count++;               /* Increment receive msg counter */
      if (rx_msg_count == 1000) {  /* If 1000 messages have been received, */
        PTD->PTOR  |=  1<<16;  /*  toggle  output  port  D16  (Green  LED)  */
        rx_msg_count = 0;           /* and reset message counter */
      }
      FLEXCAN0_transmit_msg ();     /* Transmit message using MB0 */
    }
  }
```

2.10.3.2 FlexCAN_FD.c

```
#include "S32K144.h" /* include peripheral declarations S32K144 */
#include "FlexCAN_FD.h"

uint32_t  RxCODE;              /* Received message buffer code */
uint32_t  RxID;                /* Received message ID */
uint32_t  RxLENGTH;            /* Received message number of data bytes */
uint32_t  RxDATA[2];           /* Received message data (2 words) */
uint32_t  RxTIMESTAMP;         /* Received message time */

void FLEXCAN0_init(void) {
#define MSG_BUF_SIZE  18    /* Msg Buffer Size. (2 words hdr + 16 words data  = 18 words) */
  uint32_t   i=0;

  PCC->PCCn[PCC_FLEXCAN0_INDEX] |= PCC_PCCn_CGC_MASK; /* CGC=1: enable clock to FlexCAN0 */
  CAN0->MCR |= CAN_MCR_MDIS_MASK;          /* MDIS=1: Disable module before selecting clock */
  CAN0->CTRL1 |= CAN_CTRL1_CLKSRC_MASK;   /* CLKSRC=1: Clock Source = BUSCLK (40 MHz) */
  CAN0->MCR &= ~CAN_MCR_MDIS_MASK;         /* MDIS=0; Enable module config. (Sets FRZ,HALT)*/
  while (!((CAN0->MCR & CAN_MCR_FRZACK_MASK) >> CAN_MCR_FRZACK_SHIFT))  {}
               /* Good practice: wait for FRZACK=1 on freeze mode entry/exit */
  CAN0->CBT = 0x802FB9EF;   /* Configure nominal phase: 500 KHz bit time, 40 MHz Sclock */
                            /* Prescaler = CANCLK / Sclock = 80 MHz / 40 MHz = 2 */
                            /* EPRESDIV = Prescaler - 1 = 2 - 1 = 1 */
                            /* EPSEG2 = 15 */
                            /* EPSEG1 = 15 */
                            /* EPROPSEG = 46 */
                            /* ERJW = 15 */
  /* BITRATEn =Fcanclk /( [(1 + (EPSEG1+1) + (EPSEG2+1) + (EPROPSEG + 1)] x (EPRESDIV+1)) */
  /*          = 80 MHz /( [(1 + ( 15   +1) + ( 15   +1) + (   46    + 1)] x (    1   +1)) */
  /*          = 80 MHz /( [1+16+16+47] x 2) = 80 MHz /(80x2) = 500 Kz */

CAN0->FDCBT = 0x00131CE3;  /* Configure data phase: 2 MHz bit time, 40 MHz Sclock */
                             /* Prescaler = CANCLK / Sclock = 80 MHz / 40 MHz = 2 */
                             /* FPRESDIV = Prescaler - 1 = = 2 - 1 = 1 */
                             /* FPSEG2 = 3 */
                             /* FPSEG1 = 7 */
                             /* FPROPSEG = 7 */
                             /* FRJW = 3 */
  /* BITRATEf =Fcanclk /( [(1 + (FPSEG1+1) + (FPSEG2+1) + (FPROPSEG)] x (FPRESDIV+!)) */
  /*          = 80 MHz /( [(1 + (  7   +1) + (  3   +1) + (   7    )] x (    1   +1)) */
  /*          = 80 MHz /( [1+8+4+7] x 2) = 80 MHz /(20x2) = 80 MHz / 40 = 2 MHz   */
  CAN0->FDCTRL =0x80038500;  /* Configure bit rate switch, data size, transcv'r delay   */
                             /* BRS=1: enable Bit Rate Swtich in frame's header */
                             /* MBDSR1: Not applicable */
                             /* MBDSR0=3: Region 0 has 64 bytes data in frame's payload */
                             /* TDCEN=1: enable Transceiver Delay Compensation */
                             /* TDCOFF=5: 5 CAN clocks (300us) offset used */
  for(i=0; i<128; i++ ) {    /* CAN0: clear 128 words RAM in FlexCAN 0 */
    CAN0->RAMn[i] = 0;       /* Clear msg buf words. All buffers CODE=0 (inactive) */
  }
  for(i=0; i<16; i++ ) {          /* In FRZ mode, init CAN0 16 msg buf filters */
    CAN0->RXIMR[i] = 0xFFFFFFFF;  /* Check all ID bits for incoming messages */
  }
  CAN0->RXMGMASK = 0x1FFFFFFF;  /* Global acceptance mask: check all ID bits */
                                               /* Message Buffer 4 - receive setup: */
  CAN0->RAMn[ 4*MSG_BUF_SIZE + 0] = 0xC4000000; /* Msg Buf 4, word 0: Enable for reception */
                                               /* EDL=1: Extended Data Length for CAN FD */
                                               /* BRS = 1: Bit Rate Switch enabled */
                                               /* ESI = 0: Error state */
                                               /* CODE=4: MB set to RX inactive */
                                               /* IDE=0: Standard ID */
                                             /* SRR, RTR, TIME STAMP = 0: not applicable*/
#ifdef NODE_A                                  /* Node A receives msg with std ID 0x511 */
  CAN0->RAMn[ 4*MSG_BUF_SIZE + 1] = 0x14440000; /* Msg Buf 4, word 1: Standard ID = 0x111 */
#else                                         /* Node B to receive msg with std ID 0x555 */
  CAN0->RAMn[ 4*MSG_BUF_SIZE + 1] = 0x15540000; /* Msg Buf 4, word 1: Standard ID = 0x555 */
#endif
                                               /* PRIO = 0: CANFD not used */
  CAN0->CTRL2 |= CAN_CTRL2_STFCNTEN_MASK;      /* Enable CRC fix for ISO CAN FD */
  CAN0->MCR = 0x0000081F;       /* Negate FlexCAN 1 halt state & enable CAN FD for 32 MBs */
  while ((CAN0->MCR && CAN_MCR_FRZACK_MASK) >> CAN_MCR_FRZACK_SHIFT)  {}
                /* Good practice: wait for FRZACK to clear (not in freeze mode) */
  while ((CAN0->MCR && CAN_MCR_NOTRDY_MASK) >> CAN_MCR_NOTRDY_SHIFT)  {}
                /* Good practice: wait for NOTRDY to clear (module ready)   */
}
```

```
void FLEXCAN0_transmit_msg(void) { /* Assumption: Message buffer CODE is INACTIVE */
  CAN0->IFLAG1 = 0x00000001; /* Clear CAN 0 MB 0 flag without clearing others*/ CAN0-
  >RAMn[ 0*MSG_BUF_SIZE + 2] = 0xA5112233; /* MB0 word 2: data word 0 */
  CAN0->RAMn[ 0*MSG_BUF_SIZE + 3] = 0x44556677; /* MB0 word 3: data word 1 */
#ifdef NODE_A
  CAN0->RAMn[ 0*MSG_BUF_SIZE + 1] = 0x15540000; /* MB0 word 1: Tx msg with STD ID 0x555 */
#else
  CAN0->RAMn[ 0*MSG_BUF_SIZE + 1] = 0x14440000; /* MB0 word 1: Tx msg with STD ID 0x511 */
#endif
  CAN0->RAMn[ 0*MSG_BUF_SIZE + 0] = 0xCC4F0000 | 8 <<CAN_WMBn_CS_DLC_SHIFT; /* MB0 word 0: */
                                                /* EDL=1 CAN FD format frame*/
                                               /* BRS=1: Bit rate is switched inside msg */
                                                /* ESI=0: ??? */
                                               /* CODE=0xC: Activate msg buf to transmit */
                                                /* IDE=0: Standard ID */
                                                /* SRR=1 Tx frame (not req'd for std ID) */
                                             /* RTR = 0: data, not remote tx request frame*/
                                                /* DLC=15; 64 bytes */
}

void FLEXCAN0_receive_msg(void) {  /* Receive msg from ID 0x556 using msg buffer 4 */
  uint8_t j;
  uint32_t dummy;

  RxCODE   = (CAN0->RAMn[ 4*MSG_BUF_SIZE + 0] & 0x07000000) >> 24;  /* Read CODE field */
  RxID     = (CAN0->RAMn[ 4*MSG_BUF_SIZE + 1] & CAN_WMBn_ID_ID_MASK) >>CAN_WMBn_ID_ID_SHIFT ;
  RxLENGTH =(CAN0->RAMn[ 4*MSG_BUF_SIZE + 0] & CAN_WMBn_CS_DLC_MASK)>>CAN_WMBn_CS_DLC_SHIFT;
  for (j=0; j<2; j++) {  /* Read two words of data (8 bytes) */
    RxDATA[j] = CAN0->RAMn[ 4*MSG_BUF_SIZE + 2 + j];
  }
  RxTIMESTAMP = (CAN0->RAMn[ 0*MSG_BUF_SIZE + 0] & 0x000FFFF);
  dummy = CAN0->TIMER;             /* Read TIMER to unlock message buffers */
  CAN0->IFLAG1 = 0x00000010;       /* Clear CAN 0 MB 4 flag without clearing others*/
}
```

2.10.3.3 FlexCAN_FD.h (Partial listing)

```
#define NODE_A        /* If using 2 boards as 2 nodes, NODE A & B use different CAN IDs */
```

2.10.3.4 clocks_and_modes.c

See code in clocks_and_modes.c of the Hello World + Clock example.

2.10.3.5 Reference: MC33903 code for obsolete EVB (not included in project)

LPSPI1 initialization, transmit and receiver functions: See code for functions in SPI example.

Port pin and MC33803 initialization: See code for functions in CAN 2.0 example.

CAN initialization for 1 MHz data phase (MC33903 limit):

```
/* Use 1 MHz data phase bit rate for MC33903:*/
  CAN0->FDCBT = 0x00135CE7;  /* Configure data phase: 1 MHz bit time, 40 MHz Sclock */
                             /* Prescaler = CANCLK / Sclock = 80 MHz / 40 MHz = 2 */
                             /* FPRESDIV = Prescaler - 1 = 2 - 1 = 1 */
                             /* FPSEG2 = 7 */
                             /* FPSEG1 = 7 */
                             /* FPROPSEG = 23 */
                             /* FRJW = 3 */
  /* BITRATEf =Fcanclk /( [(1 + (FPSEG1+1) + (FPSEG2+1) + (FPROPSEG)] x (FPRESDIV+!)) */
  /*          = 80 MHz /( [(1 + (  7   +1) + (  7   +1) + (   23   )] x (    1   +1)) */
  /*          = 80 MHz /( [1+8+8+23] x 2) = 80 MHz /(40x2) = 80 MHz / 80 = 1 MHz   */
  CAN0->FDCTRL = 0x8003F300;  /* Configure bit rate switch, data size, transcv'r delay */
                             /* BRS=1: enable Bit Rate Swtich in frame's header */
                             /* MBDSR1: Not applicable */
                             /* MBDSR0=3: Region 0 has 64 bytes data in frame's payload */
                             /* TDCEN=1: enable Transceiver Delay Compensation */
                             /* TDCOFF=15: 15 CAN clocks (375us) offset used */
```

2.11 其他示例和文档

有关更多示例，请参考nxp.com中的相关文件，该文件中有两个文件夹：“doc”和“examples”。examples文件夹是每个S32K14x EVB的每个cookbook项目的源代码，包括本AN中已经提到的示例以及本AN中未涵盖的一些外围设备的示例。

“doc”文件夹包含示例文件夹中所有示例的必要文档。双击名为“此处开始”的快捷方式阅读文档。

3 Startup 代码

3.1 S32 Design Studio, S32K14x flash target

编译器通常有向导或示例启动代码，其中包括如下所列的初始化。通常，应将此代码视为起点。用户应检查初始化，以查看是否缺少任何初始化或是否需要更改

表14.S32 Design Studio v1.2、S32K144和flash target的启动代码摘要

Step	startup_S32K144.s	system_S32K144.h	system_S32K144.c	startup.c
1	__isr_vector table, per link file starts at 0x0, defines: - 0x0 StackTop address, - 0x4 Reset_Handler address, - other exception addresses, - interrupt vector addresses			
2	Reset_Handler: - Masks interrupts - Initializes regs, SP, - SystemInit			
3		DISABLE_WDOG = 1	SystemInit: - disable watchdog	
4	init_data_bss			
5				init_data_bss: - Init vector table - Init data pointers, .data etc. - Copy init data ROM to RAM - Init .bss
6	Branch to main			

其他常见的启动功能包括：

- 缓存
  - — S32K144有一个4KB指令缓存、2路集合关联、四字行
  - — 软件可以配置MPU以定义闪存的可缓存和不可缓存区域。
  - — 软件必须使缓存失效并启用缓存。

```
LMEM->PCCCR = 0x85000001; /* Invalidate cache & enable write buffer, cache */
```

- 闪存控制器
  - — 预取缓冲区执行推测性读取以提高顺序访问的性能
  - — 软件必须启用预取缓冲区。

```
MSCM->OCMDR0 = 0x00000020; /* Bit 5 = 1: Enable program flash prefetch buffer */
MSCM->OCMDR1 = 0x00000020; /* Bit 5 = 1: Enable data flash prefetch buffer */
```

1.S32K1xx Series Reference Manual, Rev 11, 06,2019, section 33.4.4.1 Cache set commands.
2.S32K1xx Series Reference Manual, Rev 11, 06,2019, section 35.5.2 Speculative reads.

4 头文件备忘单

表 15. Header files cheat sheet头文件备忘单

Action	Family	Syntax	Examples
Initialize Register	S12	MODULEREG = value;	CPMUPOSTDIV = 0;
	MPC5xxx	MODULE.REG.R = value;	SIUL.PCR[40].R = 0x1234;
	KEA	MODULE_REG = value;	FTM2_C0SC = 0X68;
	S32K	MODULE->REG = value;	PORTD->PCR[10] = 0X00000200;
Initialize Bit Field	S12	MODULEREG_FIELD = value;	CPMUPLL_FM = 2;
	MPC5xxx	MODULE.REG.B.FIELD = value;	SIUL.PCR[4].B.PA = 3;

	KEA	MODULE*n*-_REG &= ~MOD_REG_FIELD_MASK;
		and MODULE*n*-_REG |= MOD_REG_FIELD(value);
			ADC_SC1 &= ~ADC_SC1_ADCH_MASK; // clear field
			ADC_SC1 |= ADC_SC1_ADCH(6); // initialize field

	S32K	MODULE*n*-->REG &= MODULE_REG_FIELD_MASK;
		MODULE*n*-->REG = MODULE_REG_FIELD(value);
			PORTE->PCR[4] &= ~PORT_PCR_MUX_MASK;
			PORTE->PCR[4] |= PORT_PCR_MUX(0b010);

Set Bit	S12	MODULEREG_FIELD = 1;	CPMURTI_RTDEC = 1;
	MPC5xxx	MODULE*[n]*.REG.B.FIELD = 1;	SIUL.PCR[40].B.OBE = 1;

	KEA	MODULE*n*-_REG |= MODULE_REG_FIELD_MASK;
		or MODULE*n*-_REG |= 1<<CONSTANT;
			SIM_SCGC |= SIM_SCGC_FM2_MASK;
			#define PTC0 9;
			GPIOA_PDDR |= 1<<PTC0;

	S32K	MODULE*n*-->REG |= MODULE_REG_FIELD_MASK;
		or MODULE*n*-->REG |= 1<<CONSTANT;
			LPIT0->MCR |= LPIT_MCR_CEN_MASK;
			PTD->PDDR |= 1<<10;

Clear Bit	S12	MODULEREG_FIELD = 0;	CPMURTI_RTDEC = 0;
	MPC5xxx	MODULE.REG.B.FIELD = 0;	SIUL.PCR[40].B.OBE = 0;

	KEA	MODULE*n*-_REG &= ~MODULE_REG_FIELD_MASK;
		or MODULE*n*-_REG &= ~(1<<CONSTANT);
			I2C_C1 &= ~I2C_C1_TX_MASK;
			GPIOA_PDDR &= ~(1<<12);

	S32K	MODULE*n*->REG &= ~MODULE_REG_FIELD_MASK;
		or MODULE*n*-->REG &= ~(1<<CONSTANT);
			PTC->PDDR &= ~(1<<12);

Read Bit	S12	x = MODULEREG_FIELD;	x = CPMURTI_RTDEC;
	MPC5xxx	x = MODULE.REG.B.FIELD;	x = SIUL.PCR[5].B.OBE;
	KEA	x = (MODULE*n*-_REG>>CONSTANT) & 1;	x = (GPIOA_PIDR>>PTD0) & 1;
	S32K	x = (MODULE*n*-->REG>>CONSTANT) & 1;	x = `(LPSPI1->SR & LPSPI_SR_TDF_MASK)>>LPSPI_SR_TDF_SHIFT`
Read Bit Field	S12	x = MODULEREG_FIELD;	x = CPMUSYNR_SYNDIV;
	MPC5xxx	x = MODULE.REG.B.FIELD;	x = SIUL.PCR[5].B.PA;
	KEA	x = (MODULE*n*-_REG & MODULE_REG_FIELD_MASK) >> MODULE_REG_FIELD_SHIFT;	x = (I2C_A1 & I2C_A1_AD_MASK) >> I2C_A1_AD_MASK;
	S32K	x = (MODULE*n*-->REG & MODULE_REG_FIELD_MASK) >> MODULE_REG_FIELD_SHIFT;	x = `(LPSPI1->SR & LPSPI_SR_TDF_MASK)>>LPSPI_SR_TDF_SHIFT`

5 新增项目

示例项目是使用ARM版本1.1的S32 Design Studio实现的。创建自己的项目的一个简单方法是从现有项目开始。下表列出了示例步骤。

表16.创建新S32 Design Studio项目的示例步骤

	Step	Description
1	Start application	• Start S32 Design Studio with desired workspace
2	Add blank project	• File - New - New S32DS Project • Enter project name. Example: FTM • Select processor. Example: S32K144 • Click Next • Review cores and parameters and change if desired • Click Finish
3	Add a new empty source file to project	• Right click on “src” folder • Select New - File • If needed, change parent folder • Enter File name. Example: FTM.c • Click Finish Note: if the file name already exists in the folder, an error message is displayed but the file still is added to the project.
4.	Copy an existing source file from a different project in the workspace	• Select file(s) from src folder of other project in workspace • With the selected files highlighted: right click - Copy • Select new project src folder • Right click - Paste
5	Make any file name adjustments	• Right click, and rename • Change #include if needed
6	Build project, edit as needed and rebuild	• Build by clicking on the hammer icon (any files added or deleted in the project’s src folder will be included/deleted in the build.and then appear in an updated list in the src folder)
7	Tip: Close unrelated projects	• In the Project Explorer window, right click on the project name • Select “Close unrelated projects” (Can speed up debug, etc.)

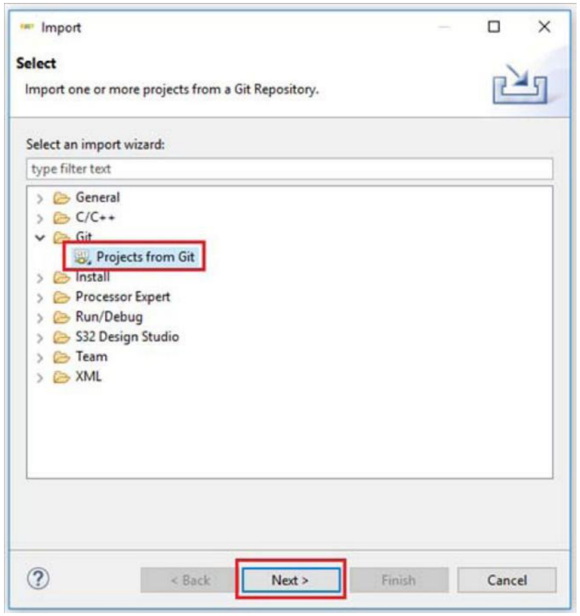
6 66S32K1xx系列使用说明存储库

除了本应用笔记中提到的项目外，还有一个可用的存储库，您可以在其中找到更多S32K1xx外围设备的初始化示例。请参见以下步骤将这些项目导入到S32DS工作区。

6.1 从S32K1xx系列使用说明存储库导入项目

1. 先决条件：安装和配置gits
   —https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
   —https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup
2. 为具有所需工作空间的ARM启动S32DS
3. 克隆存储库并首次导入

— Go to: File -> Import…
— Select: Git -> Projects from Git, and click Next.



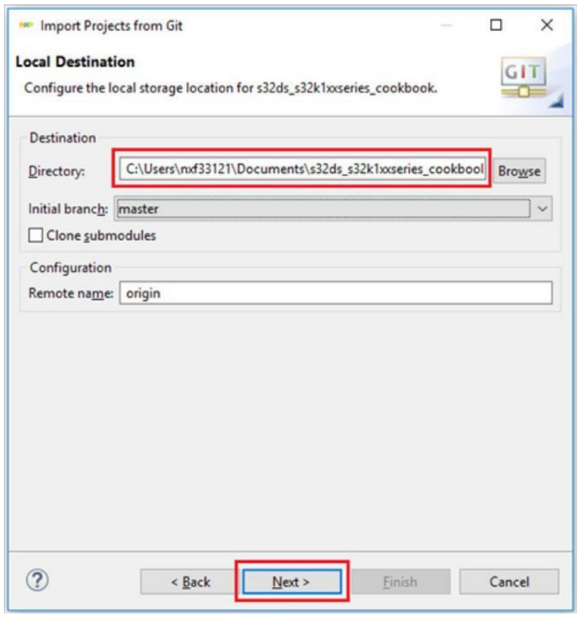
— 选择：克隆URL，然后单击下一步。
— 在URL字段中，根据下表粘贴所需的存储库URL
然后单击下一步。

表17.克隆可用示例代码的S32K1xx系列存储库的URL

Device URL	URL
S32K116	https://source.codeaurora.org/external/s32support/s32k116_cookbook
S32K118	https://source.codeaurora.org/external/s32support/s32k118_cookbook
S32K142	https://source.codeaurora.org/external/s32support/s32k142_cookbook
S32K144	https://source.codeaurora.org/external/s32support/s32k144_cookbook
S32K144W	https://source.codeaurora.org/external/s32support/s32k144w_cookbook
S32K146	https://source.codeaurora.org/external/s32support/s32k146_cookbook
S32K148	https://source.codeaurora.org/external/s32support/s32k148_cookbook

— 选中“主分支”，然后单击“下一步”。

— 在目标目录中，输入要克隆存储库的本地路径，然后单击


Import Projects from Git

Local Destination

Configure the local storage location for s32ds_s32k1xxseries_cookbook.

Destination

Directory: C:\Users\nxf33121\Documents\s32ds_s32k1xxseries_cookbool

Initial branch: master

☐ Clone submodules

Configuration

Remote name: origin

— 选择：工作树-目录，然后单击下一步。

— -选中要导入的项目，然后单击“完成”。

4. 从现有存储库导入项目

— 转到：文件导入…

— 选择：Git中的Git项目，然后单击Next。

— 选择：现有本地存储库，然后单击下一步。

— 根据表18选择：s32k1xx_cookbook，然后单击Next(请记住，存储库应该已经存在，请确保您已经遵循了步骤3）

表18.可用示例代码的S32K1xx系列存储库名称

Device	Name
S32K116	s32K116_cookbook
S32K118	s32k118_cookbook
S32K142	s32k142_cookbook
S32K144	s32k144_cookbook
S32K144W	s32144w_cookbook
S32K146	s32k146_cookbook
S32K148	s32k148_cookbook

— 选中要导入的项目，然后单击“完成”。

5. Build and debug
6. S32K1xx系列食谱文档
   — 转到刚刚克隆存储库的本地目录。
   — 打开“文档”文件夹
   — 双击“此处开始”。

7 修改历史

表19.修改历史

Rev. No.	Date	Substantive Change(s)
0	03/2017	Initial release
1	07/2017	• Code clarifications/improvements made to Hello+Interrrupts, ADC, UART, SPI, • CAN 2.0 and CAN FD examples. See text at start of source code files for details. • Code for obsolete S32K144 EVB board with MC33903 CAN PHY removed from • Section 2.8, Section 2.9 and Section 2.10 projects and put in application note text. • Added Example Operation to Section 2.9 and Section2.10 examples • Other minor modifications • Editorial updates
2	08/2018	• Added Section 2.11
3	01/2019	• Added Section 6
4	09/2019	• Updated footnotes in Section 3.1.
5	12/2020	• Updated S32K1xx Cookbook introduction and Section 6 with the new available repository structure for the example codes.

Document Number AN5413
Revision 5 12/2020