AN4300
Application note

Advanced light control and diagnostics using SPC56xBx/RPC56xBx and SPC56xCx microcontrollers

Introduction

The members of the SPC56xBx/RPC56xBx and SPC56xCx families are high performance microcontrollers developed for car body application needs. These devices are especially useful in automotive light management applications and offer features to assist the car body system developer in writing optimized firmware with a very low CPU load.

This document offers an interesting example about the interconnection between the eMIOS, CTU, and ADC modules of the SPC56EC74 (one member of the SPC56xB/RPC56xBx/Cx family) in a lighting application, including the support of circuitry diagnostics, without placing an additional load on the MCU's CPU. This example is easily applied to each device of the SPC56xB/RPC56xBx/Cx family.

Contents

List of tables

List of figures

1 Lighting application design challenge

1.1 General light application aspects

In an automotive lighting application it is necessary to drive two types of lights:

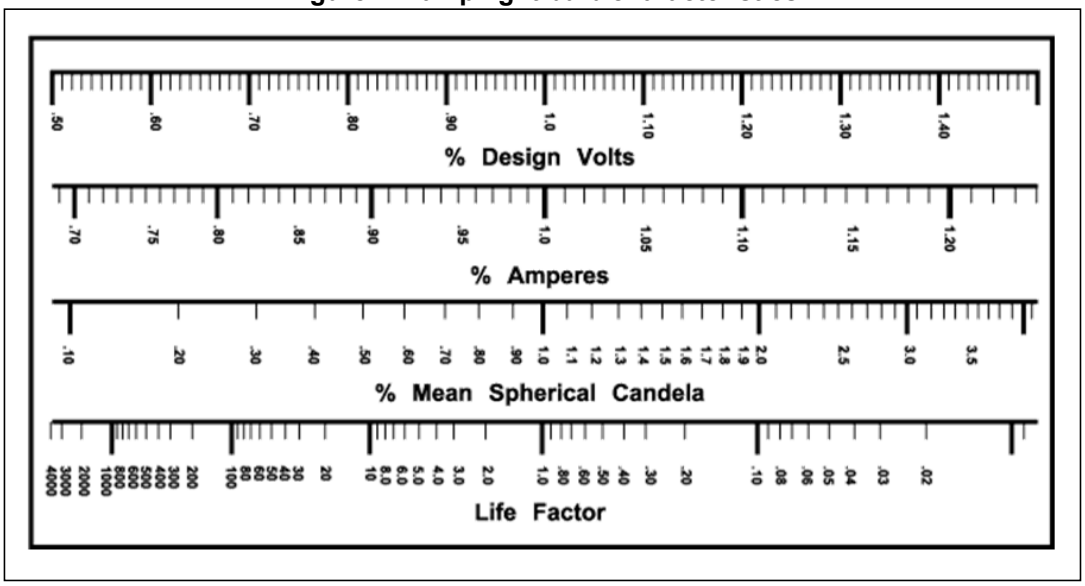
1. Bulb lamp
2. LED

When controlling the lamp light bulb, two points must be addressed:

- Lamp power supply variations
- Lamp lifetime

Lamp power supply voltage may differ when compared to its nominal value. *Figure 1* displays how lamp lifetime depends on power supply voltage. The impact of 5 % overvoltage (630 mV at 12.6 V) on light bulb performance is significant, as indicated by the vertical line in the figure. The light bulb luminous flux grows up to 120 % and the light bulb lifetime is reduced to less than 60 %, when compared to nominal values. Therefore, the lamp power supply voltage should be regulated to the nominal value to maintain lamp lifetime. The most common control technique is the PWM, which can also be used for lamp light beam intensity control.

Figure 1. Lamp light bulb characteristics



A lamp's lifetime is limited and it can be shortened if the lamp is connected in a faulty environment. If the design includes lamp diagnostics and protection, the lamp's nominal lifetime can be preserved. Usually the load drivers report various diagnosis information and the light application has to collect and check this information. These factors must taken into account when designing a lighting module.

1.2 Driver signals characteristics

The signals that drive the light loads must satisfy the following requirements:

- Capability to drive up to 60 channels with PWM signals
- Capability to drive different load types using PWM signals with different characteristics:
  - Bulb lamp
    PWM frequency: ~100 Hz
    Duty cycle: 50 to 100 %
    Duty cycle adjustment precision: usually 5 %
  - LED
    PWM frequency: 150 to 200 Hz
    Duty cycle: 10 % to 100 % (as much as 2 % to 100 % for high efficiency LED)
    Duty cycle adjustment precision: usually 1–2 %

The variation of the duty cycle is performed to maintain constant power on the lamps when the battery voltage changes.

The duty cycle variation is based on the following expression:

Equation 1

$$\delta = \frac{V_{ref}^{2}}{V_{bat}^{2}} \qquad If\ Vbat > Vref$$

where

V_{ref} is the nominal value of the battery voltage

V_{bat} is the actual battery voltage

If $V_{bat} \leq V_{ref}$, the duty cycle is equal to 100 %, else the expression is applied. The adaptation of the duty cycle is performed one time per period.

A typical battery voltage value for automotive applications is 13.2 V, so if, for example, the application reads a V_{bat} equal to 16 V, a duty cycle of 68 % applies.

When turning the lamp light bulb on, the bulb current applied to a cold filament can reach up to 10 times the rated current. The total current of several lamps causes a significant power supply voltage drop and an increase in the electromagnetic interference (or EMI) emissions. Therefore, it is essential not to drive all lamps together and it is necessary to introduce a delay between each pair of rising edges of the PWM signals (see *Figure 3*).

1.3 System proposal

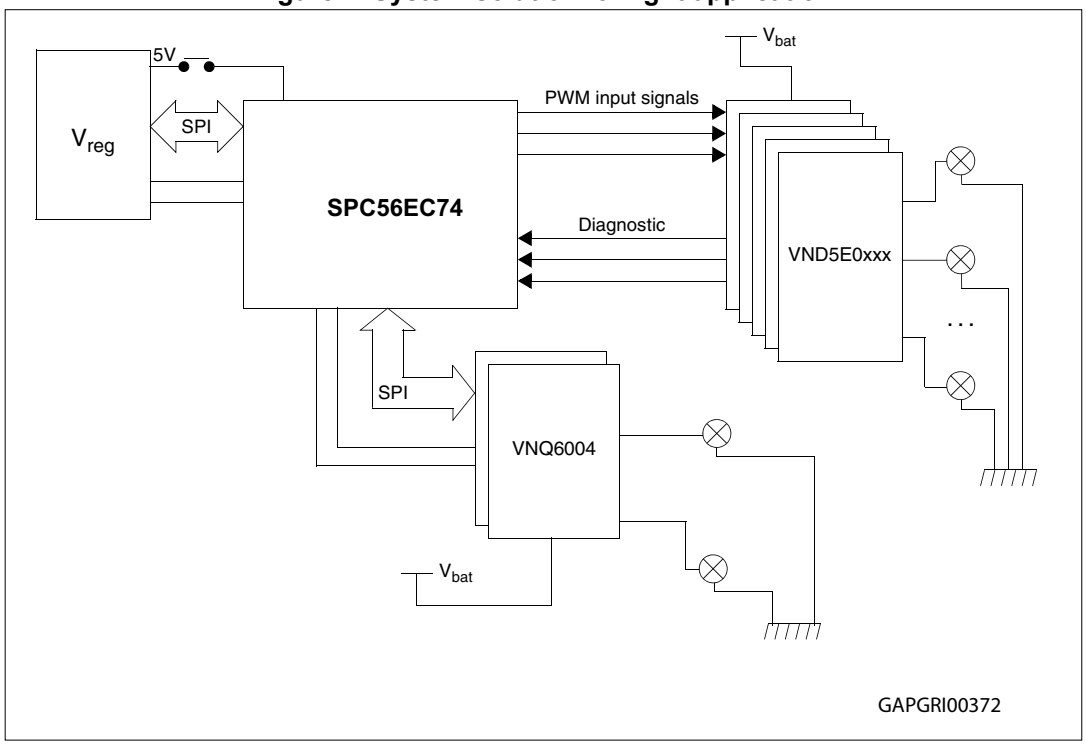
In a typical system solution for a light application, the SPC56xB/SPC56xCx/RPC56xB microcontroller family can manage several power drivers, depending on how the power is managed and the number of channels to be driven.

Some devices are driven by an SPI link. Using this link, the application can configure the signals, that are generated inside the driver device, to drive the loads.

Other devices need an input signal (PWM signals) for each load to generate the output signal to drive the loads.

It is possible to use both the microcontroller and the power devices for the solution shown in *Figure 2*. This solution is configured with an STMicroelectronics power driver and the SPC56EC74 microcontroller.

Figure 2. System solution for light application



This system uses devices that are driven by PWM input signals and other devices that use an SPI connection to configure the drivers.

1.4 Diagnosis check

Today's requests for automotive systems are based not only upon technical features and reliability, but also upon increasingly important safety concepts. To fulfill these requirements, a system must provide and communicate feedback on its behavior to external devices. This, in turn, makes it possible to detect faults and react to them in a specified time.

The power drivers used in light applications provide diagnostic signals that give information about the status of loads driven. These signals must be read and interpreted to give feedback to the application. It is important to understand if there are faults and to execute protective action to obtain a safe and robust system.

Three different diagnosis types exist:

- Digital
- Analog
- Serial

The diagnosis types are explained in the following sections.

1.4.1 Digital diagnosis

To read the digital diagnosis it is necessary to read an I/O pin of the driver for each load driven. To read this type of diagnosis the microcontroller has to read the digital pin in polling mode or by connecting this signal to an external interrupt line.

1.4.2 Analog diagnosis

To read the analog diagnosis a current sense pin is available for each load driven. This pin is used to read a proportional current (voltage using a resistor) that flows in the load. Comparing this current with certain thresholds can determine if problems on the load exist. To read the voltage of the current sense pin, an analog-to-digital conversion must be performed.

The analog diagnosis has to be read in synchronous mode to the input (PWM) signal which is available after a delay from the rising edge of the input signals. The analog-to-digital conversion has to be triggered with a delay from the rising edge of the input signals to allow the load current to stabilize.

1.4.3 Serial diagnosis

When the power devices provide serial diagnosis, they usually save the type of fault in internal registers. The microcontroller, using the SPI link, then performs a query on the specified register to examine the fault status.

The application can perform the following types of diagnosis inspection:

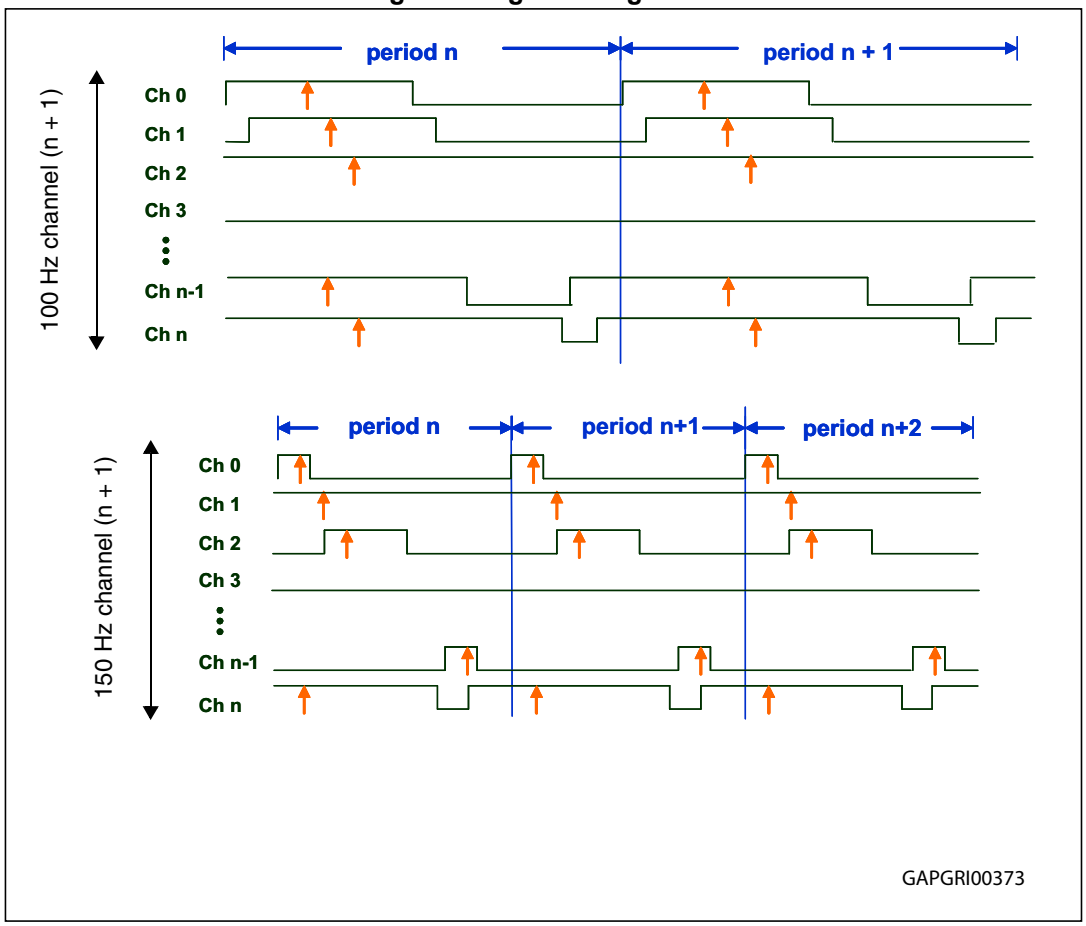
- During T_ON (time when the PWM input signal is switched on) of the input signal
  - Open-load
  - Overtemperature
  - Undervoltage
  - Short-circuit to GND
  - Short-circuit to V_{CC}
- During T_OFF (time when the PWM signal is switched off) of the input signal
  - Open-load
  - Short-circuit to V_{CC}

The digital and serial diagnosis can usually be read asynchronously with respect to the PWM input signals and the diagnosis action can be performed during a noncritical instant of the system cycle. The CPU load used to collect this information is not an important aspect.

The analog diagnosis consists of checking the current that flows in the loads. The input signal status must be known to collect diagnosis information in a specific instant of the loads' activity. The reading of the diagnosis and the input signal must be synchronized to determine the status of the input signal.

The input signal is a PWM signal, making it necessary to select if the diagnosis is occurred in the T_ON or T_OFF phase of the input signal. Furthermore, a delay exists between the rising (falling) edge and the instant when the diagnosis read occurs because the current sense signal is stabilized only after a transitory time (about 300 µs).

Figure 3. Light management



The SPC56xBx/RPC56xBx and SPC56xCx devices offer dedicated features to cover the light application requirements with very low CPU load, especially when the analog diagnosis has to be managed. *Section 2: SPC56xBx/RPC56xBx features to cover lighting needs* describes the device features that are used to manage the analog diagnosis.

2 SPC56xBx/RPC56xBx features to cover lighting needs

The SPC56EC74 is a member of the SPC56xBx/RPC56xBx and SPC56xCx 32-bit microcontroller unit family. It operates at speeds of up to 120 MHz and provides a variety of peripherals (DSPI, IIC, ADC, CTU, eMIOS, FlexCAN, LINFlex) balanced with relatively large Flash, RAM, and emulated EEPROM memory.

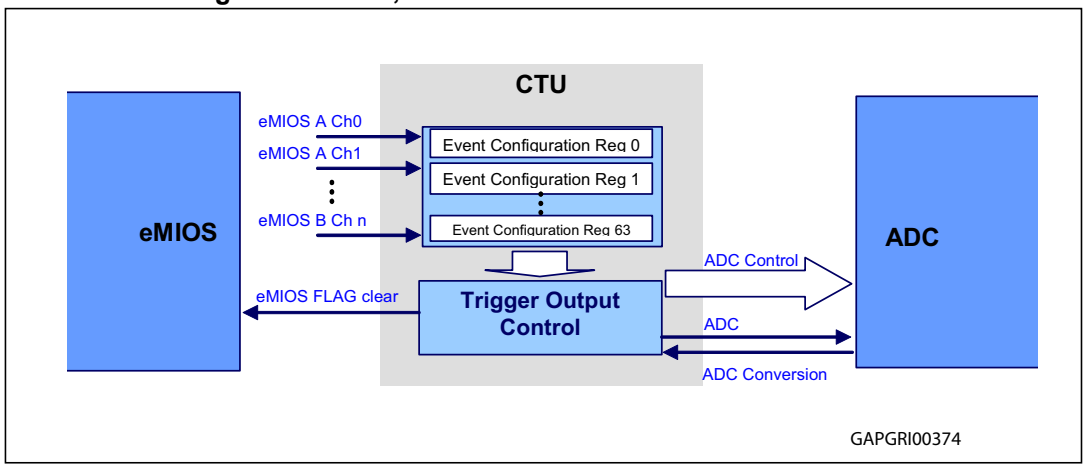
The following modules are essential parts of the lighting application suitable for lamp bulb as well as for LED control:

- eMIOS (enhanced Modular I/O Subsystem)—provides functionality to generate or measure time events and then to generate PWM signals
- CTU (Cross Trigger Unit)—allows triggering analog-to-digital conversion synchronized to a PWM signal
- ADC—allows converting analog signals synchronized to a PWM signal and allows holding the converted value in different register for each ADC channel

The three module descriptions are covered in *Section 2.1: eMIOS (enhanced Modular Input/Output Subsystem) Section 2.2: CTU (Cross Triggering Unit)* and *Section 2.3: ADC (Analog-to-Digital Converter)*. It is also recommended to read the SPC56xBx/RPC56xBx and SPC56xCx device reference manual see *Section Appendix A: Reference documents*.

The eMIOS channels are connected to event configuration registers of the CTU, where it links an eMIOS trigger to a specified ADC channel conversion (see *Figure 4*). Inside each event configuration, that is connected in a fixed way to an eMIOS channel, it is possible to write an ADC channel that is triggered when an event from the eMIOS channel arrives.

Figure 4. eMIOS, ADC and CTU interconnection scheme



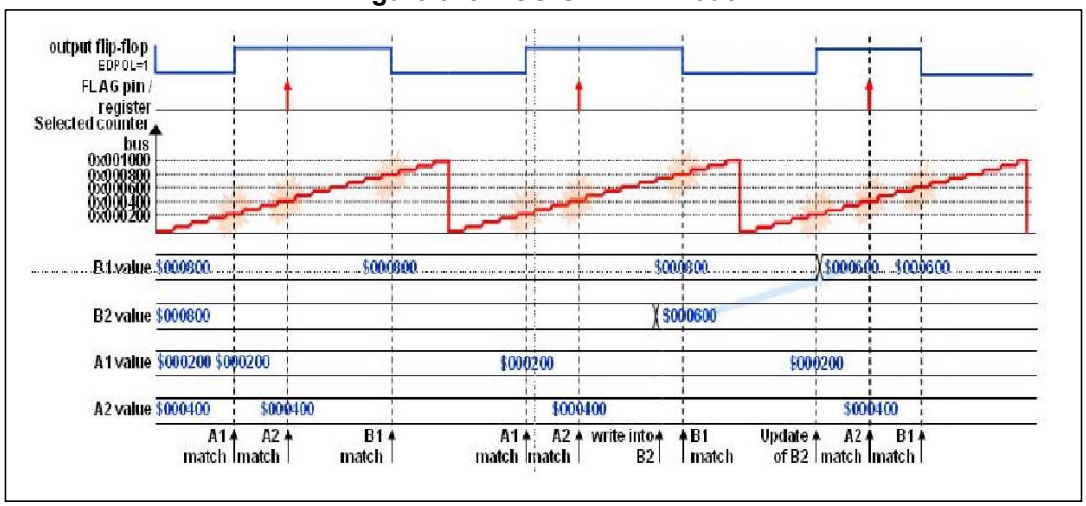
2.1 eMIOS (enhanced Modular Input/Output Subsystem)

The SPC56EC74 includes two identical eMIOS modules that together provide up to 64 channels with input and output capability. Each eMIOS module provides 32 channels with OPWMT mode, especially suited for the lighting application. In OPWMT mode, a channel is in PWM mode, making it possible to configure a compare match, anywhere in the period, to

trigger—using the CTU—an analog-to-digital conversion at a specified point of the period. This is especially suited for lighting applications.

When configuring channels UC[0], UC[8], UC[16], UC[23], and UC[24] in modulus Counter Mode (MC), the A1 register determines the counting period.

Figure 5. eMIOS-OPWMT mode



When configuring a channel in OPWMT mode, the channel can be linked to:

- Internal counter (only for the channels that have this feature)
- Counter bus A driven by UC[23]
- Counter buses B, C, D, E driven by UC[0], UC[8], UC[16], UC[24]

In a channel configured in OPWMT mode, the A1 register value defines the PWM leading-edge and the B1 register value defines the PWM trailing edge (duty cycle). The A2 register is the sampling point for diagnostic purposes (the ADC channel trigger using the CTU), and in B2, the buffered value of the trailing edge, which is, on A1 match, loaded into the B1 register (see the eMIOS chapter of the device reference manual see *Section Appendix A: Reference documents*). When the B2 value equals the A1 value, the PWM duty cycle is 0 %. When a B2 value is higher than the channel period, the PWM duty cycle reaches 100 %. The PWM output signal polarity is dependent on the user's choice.

When configuring channels UC[0], UC[8], UC[16], UC[23] and UC[24] in Modulus Counter (MC) mode, the A1 register determines the counting period.

2.2 CTU (Cross Triggering Unit)

When the eMIOS OPWMT channel counter reaches the A2 value, the specific ADC channel conversion is initiated by the CTU module. The ADC channel number is specified in the CTU event configuration register by the user. When configuring the event configuration register, be aware, that:

- The CTU channel 0–15 event can trigger any of the ADC channels 0–15
- The CTU channel 32-47 event can trigger any of the ADC channels 32–47
- The CTU channel 48–95 event can trigger any of ADC channels 48–95

For more details, see the Cross Triggering Unit chapter in the device reference manual (see *Section Appendix A: Reference documents*).

2.3 ADC (Analog-to-Digital Converter)

The 2 ADC module convert analog signals into digital form; ADC_0 with a 10-bit resolution and ADC_1 with a 12-bit resolution offering normal, injected and trigger-injected modes. The ADC module can be configured to provide up to 62 single-ended input channels with no additional external logic, expandable to 90 channels using an external multiplexer, all driven by the ADC module logic.

When using the SPC56EC74:

- The ADC channels 0–15 correspond to ANP pins 0–15 with a resolution of 10-bit or 12-bit (depending on the choice of ADC_0 (10-bit resolution) or ADC_1 (12-bit resolution); in both cases the used pins are the same (precision channels)
- The ADC_0 channels 32–60 (28 channels) correspond to ANS pins 0–27 (extended channel, medium occurrence) while the ADC_1 channels 32-44 correspond to ANS pins 0-12
- The ADC channels 64–95 correspond to ANX pins 0–3 (external multiplexer driven by MA0–2 ADC pins; external channel, medium accuracy)

For the family SPC56xBx/RPC56xBx/SPC56xCx, 6 + 3 analog watchdogs (6 on 10-bit ADC, 3 on 12-bit ADC) are present on the ADC peripheral.

The analog watchdogs provide the capability to generate an interrupt for continuous hardware monitoring to determine if the converted channel value lies within a given guarded area defined by user-specified threshold values. If the converted value lies outside the guarded area, an interrupt is generated. The analog watchdog interrupts are well suited for light bulb diagnostic purposes.

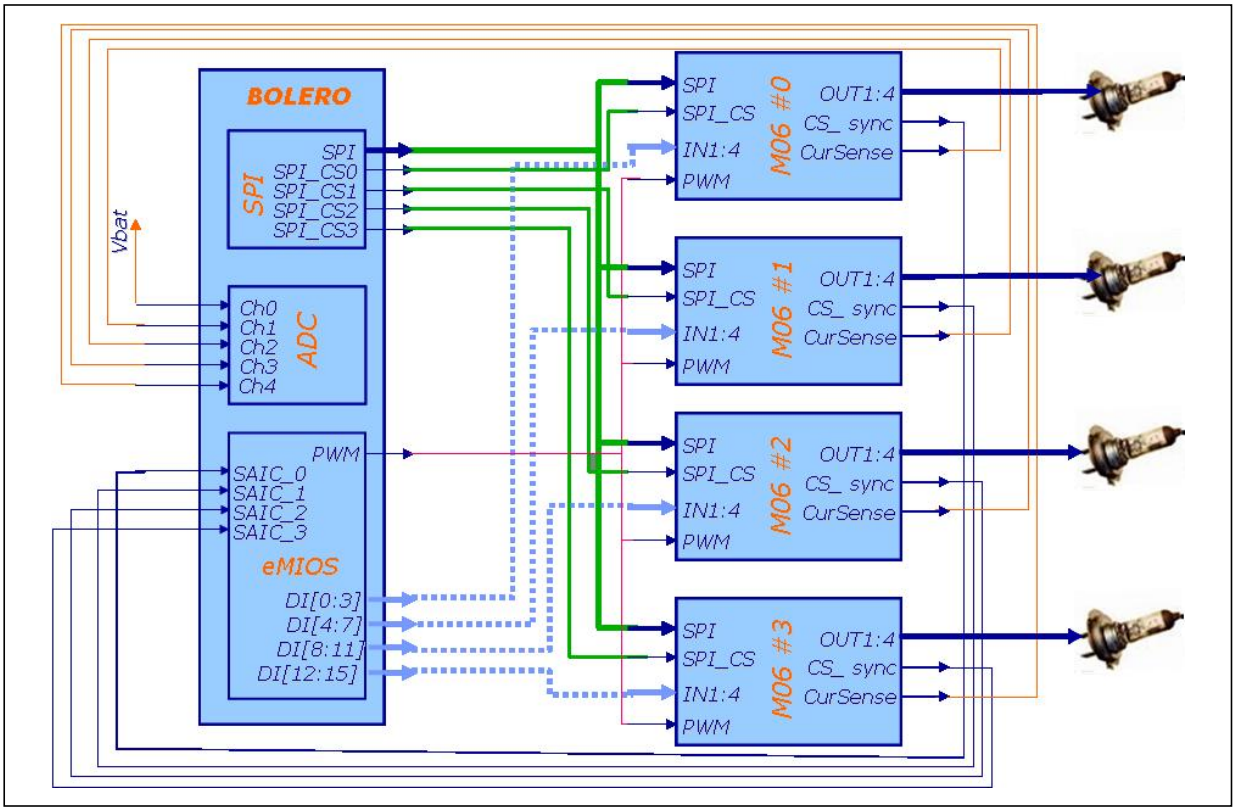
With SPC56xBx/RPC56xBx/SPC56xCx devices, the diagnosis value can be collected in hardware mode, thus reducing the load on the CPU for this task. Moreover, one time within an input signal period, a load is placed on the CPU when these values collected in hardware mode are checked and when they are compared to certain thresholds.

The analog watchdog feature can reduce this CPU load by triggering an interrupt only when a fault occurs, instead of one time for the period, because the diagnosis check is performed in automatic mode.

3 Lighting application example architecture

The lighting application example code introduces essential SPC56EC74 initialization and configuration for demonstrating eMIOS and ADC synchronization using the CTU. An example is based on the LA system study depicted in *Figure 6* that is suitable for lamp bulb as well as for LED control.

Figure 6. Lighting application—example system concept



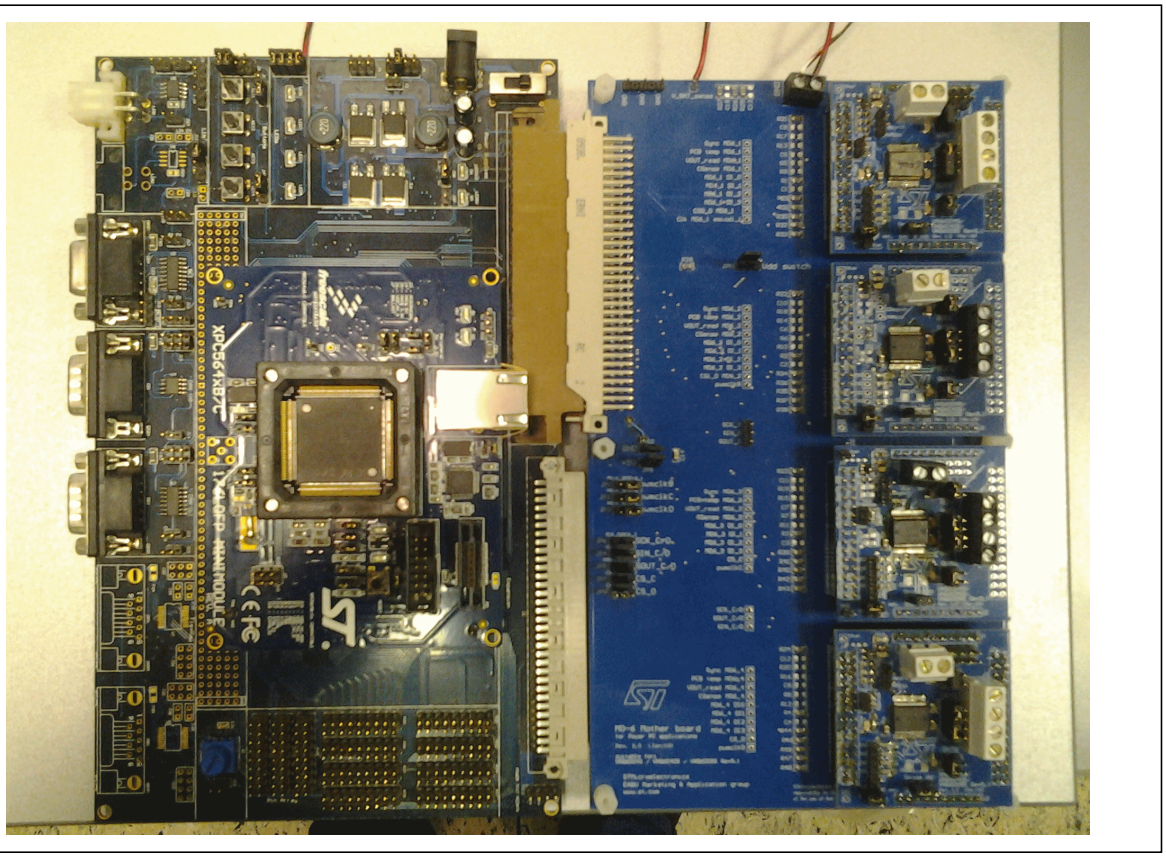
The demonstration is based on the SPC56EC74L7 176LQFP EVB, and the application code is written in C language using the Green Hills compiler.

The purpose of the application code is to show how the external triggering capability of SPC56xBx/RPC56xBx/SPC56xCx device brings “system” advantage in highly complex body application such as:

- Minimal microcontroller resources required
- Low Interrupt and CPU load
- Minimized usage of Memory resources
- Few ADC channels and pins needed

The example application code has been implemented using the M0-6 Mother board for Power PC applications connected with the SPC56EC74L7 minimodule and the EVB using the J1 connector on the board following the connection reported in *Figure 6* and in *Figure 7*.

Figure 7. SPC56EC74L7 Minimodule + M0-6 Board



3.1 Hardware architecture

The hardware system as described in the *Figure 6* uses:

- The same DSPI channel (DSPI_0) to drive all the M0-6 devices
- A dedicated chip selected for each M0-6 device
- A PWM channel (EMIOS Channel) shared for all M0-6 devices to provide a common time base (M0-6 Clock)
- Four eMIOS channels configured in SAIC to manage the trigger signals
- An ADC channel for each M0-6 device shared for the four channels inside the same M0-6 device.

3.1.1 Generalities on M0-6

As power devices we have used the STMicroelectronics M0-6 devices:

- VNQ6040S
- VNQ6004S

These devices are made using STMicroelectronics VIPower technology. They are intended for driving resistive or inductive loads directly connected to ground. Devices are protected against voltage transient on Vcc pin. Programming, control and diagnostics are

implemented via the SPI Bus through the dedicated pins. Each device offers four channels that can drive different loads in term of power.

3.2 Software Implementation

The software implements the following features:

Support of the SPI configuration of 16 channel of four M0-6

Adapt the PWM signal to drive the loads to the V_bat variations

Collect the current sense values for each channel and for each period

Low CPU load to collect the Current Sense

The code allows to initialize the configuration of M0-6 devices and at the beginning of each period (10 msec) performs the following steps:

Vbat Measurement via ADC

Duty cycle update for all channels of M0-6 devices

SPI frame for new information configuration

Digital diagnosis on all M0-6 devices

3.2.1 Synchronous diagnosis

Figure 8 and *Figure 9* represent the initial configuration of the Microcontroller and the Vbat measurement strategy in order to update the duty-cycle of the PWM signal.

Figure 8. Initial Configuration flowchart

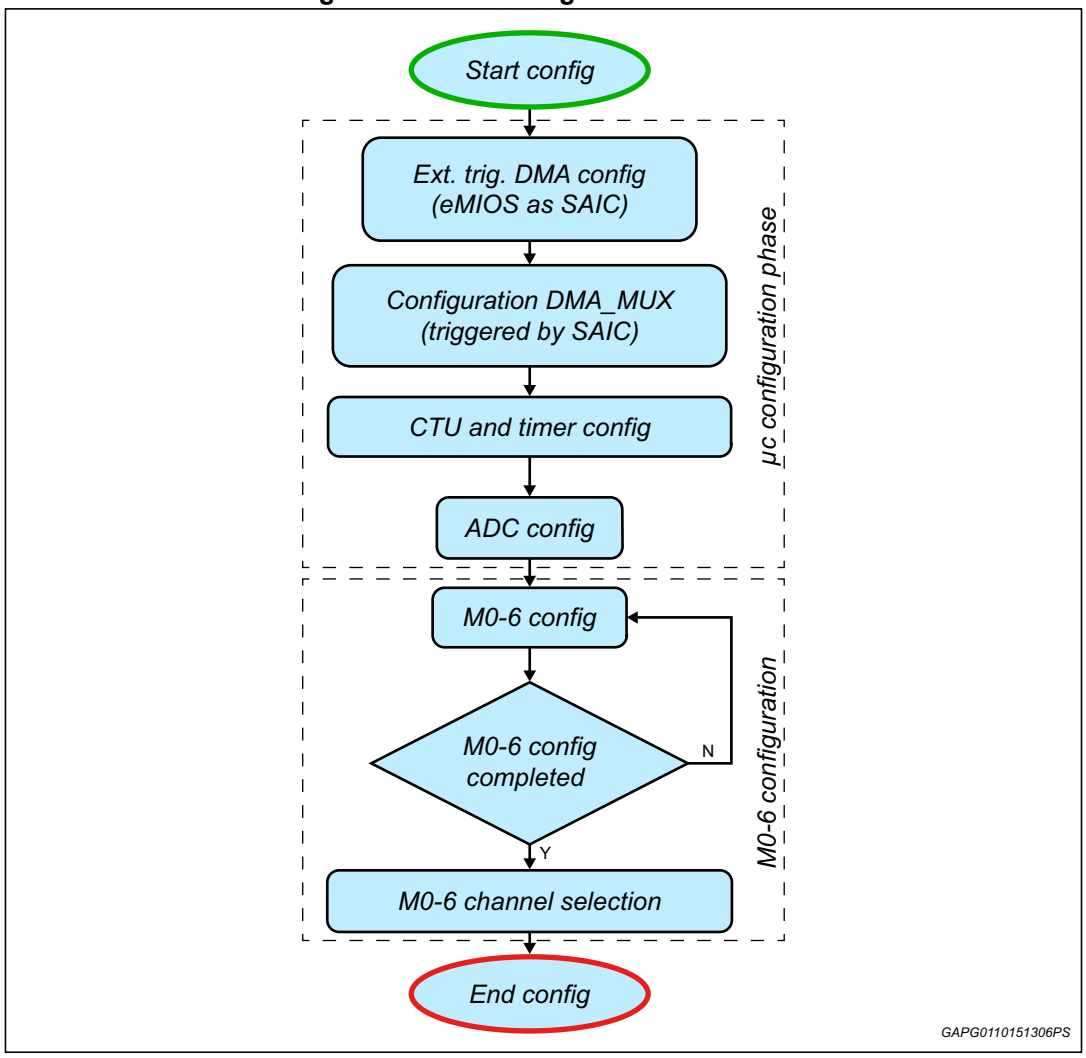
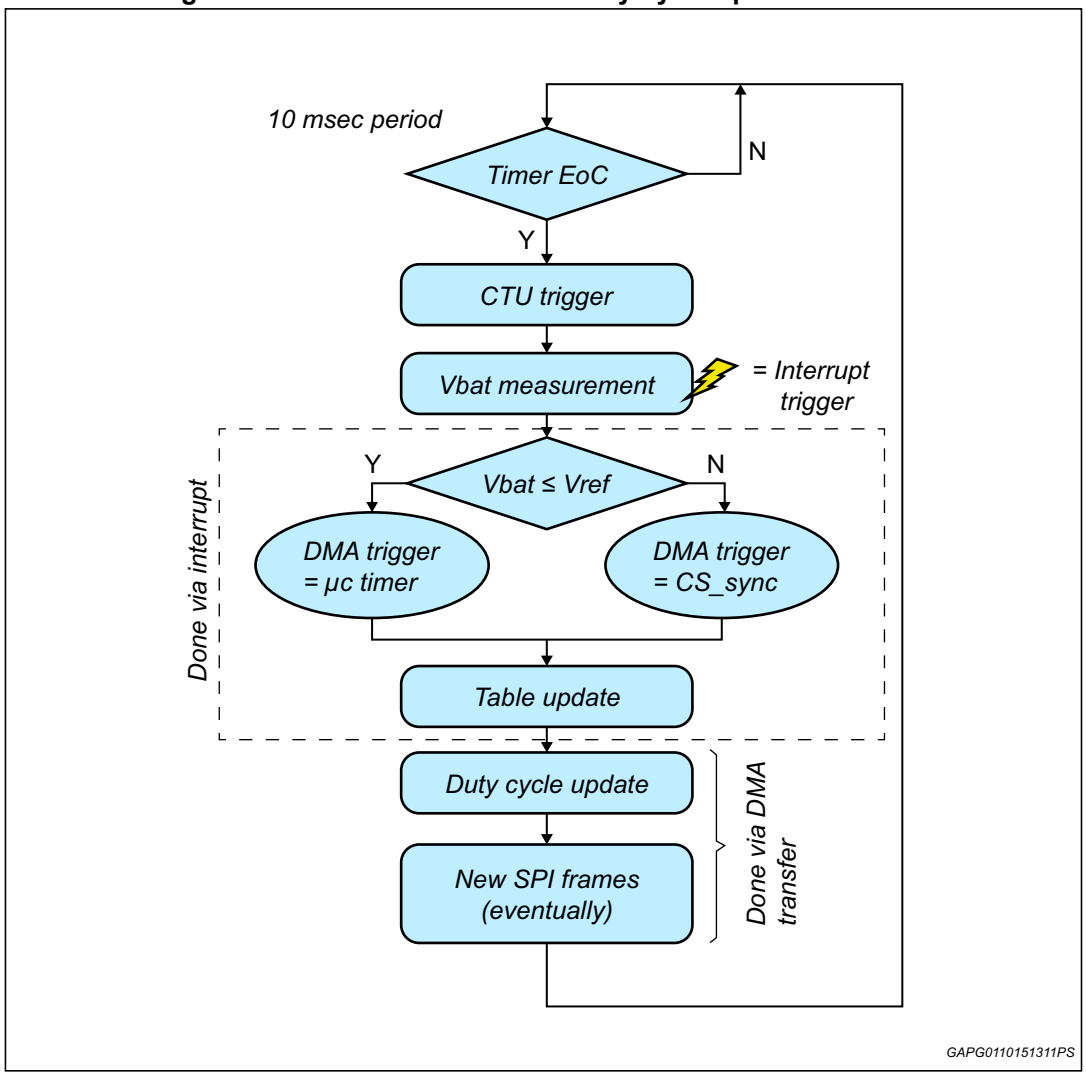


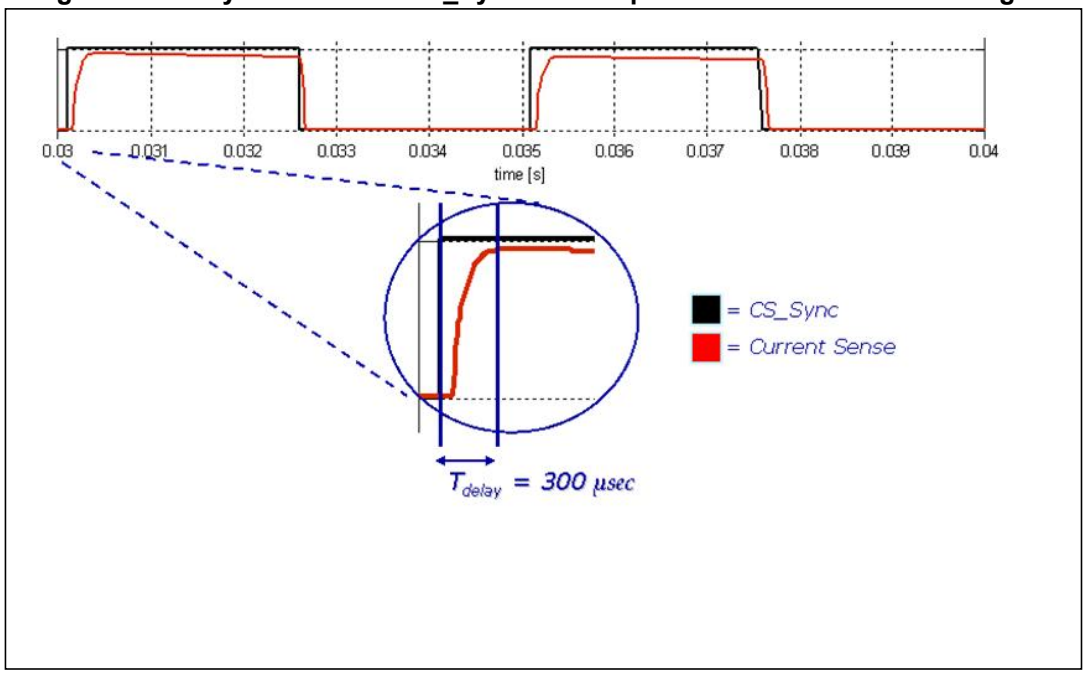
Figure 9. Vbat measurement and duty-cycle update flowchart



3.3 Synchronous Analog Diagnosis

The reading of the current sense is done in a synchronous way with the M0-6 output signals. In the following application code, current sense is read during TON of the period, 300 µs after CS_sync rising edge. It should be noticed that the microcontroller has to be synchronized with the outputs of the M0-6 device.

Figure 10. Delay between the CS_Sync and Output channel/Current Sense signal



Depending on Vbat values, M0-6 can be either in PWM mode or in DC mode. In the former case synchronous diagnosis is triggered by the rising edge of the CS_Sync signal for each active channel, by an internal timer expiration (e.g. 10 ms) of the micro in the latter.

- When in PWM mode, the CS_Sync signals is used to trigger the ADC conversion of the Current Sense of a channel after a 300 µs delay from its rising edge.
- The End of Conversion of the ADC activates a DMA channel that transfers the ADC value from the peripheral register to a SRAM location.
- At the end of a period all Current Sense are available to the same memory structure.

3.4 Implementation of the synchronous diagnosis

In order to trigger ADC conversions to perform analog diagnosis synchronous with external events (CS_sync arrival) the following resources of the microcontroller have been used:

- A Channel of the eMIOS is configured as Input Capture
- The arrival of the input signal on the Input Capture triggers a DMA transfer intended to trigger a second eMIOS channel previously configured as SAOC (Single Action Output Compare mode) counting 300 µs
- The SAOC End of Counting event triggers another eDMA transfer triggering the ADC conversion.

After the conversion the following action are performed:

- The end of ADC conversion triggers a DMA transfer moving the ADC values to a buffer in RAM.
- Via channel linking, a DMA transfer sends an SPI frame to a given M0-6 in order to link the multiplexer to the next current sense/CS_Sync.

4 MCU initialization

In this section are described the most significant functions used in the application example code.

4.1 Setting up Mode Entry and clocks

This module enables clocks to all peripherals and sets the 12 MHz bus clock using the PLL.

Figure 11. MC_Mode_Init function7

```
void MC_MODE_INIT(void)
{
   /* [RM] system clock driven by FIRC */
   CGM.SC_SS.R = 0x0;

   /* keep the z0 divider for division of 2*/
   CGM_Z0_DCR = 0x01000000;

   /* keep the FEC divider for division of 2*/
   CGM_FEC_DCR = 0x01000000;

   /* keep the FLASH divider for division of 2*/
   CGM_FLASH_DCR = 0x01000000;

   /* Enable system clock dividers with value =1*/
   CGM.SC_DC[0].R = 0x80;
   CGM.SC_DC[1].R = 0x80;
   CGM.SC_DC[2].R = 0x80;

   //Modes Configurations
   ME.MER.R = 0x000005FF;               // enable all modes

   //Setting RUN Configuration Register ME_RUN_PC[0]
   ME.RUNPC[0].R=0x000000FE; // Peripheral ON in every modes


   //re-enter in DRUN mode to update the DRUN config
   ME.MCTL.R = 0x30005AF0;         // Mode & Key
   ME.MCTL.R = 0x3000A50F;         // Mode & Key

   while(ME.GS.B.S_MTRANS==1);              /* Wait for mode entry to complete */
   while(ME.GS.B.S_CURRENTMODE!=0x3);       /* Check DRUN mode has been entered */
}
```

Figure 12. PLL initialization function

```
void PLL_12MHZ()
{
 /* Enable CLKOUT pin so clock frequency can be verified */
 /* Configure CLKOUT on PA0 */
  SIU.PCR[0].R         = 0x0A00;    /* PA0 clkout                */
  CGM.OC_EN.R          = 1;         /* Enable Output Clock       */
  CGM.OCDS_SC.B.SELDIV = 2;         /* Divide Output Clock by 4 */
  CGM.OCDS_SC.B.SELCTL = 2;       /* PLL is Output Clock       */

  ME.RUN[0].B.FXOSCOON = 1;         /* Enable external osc  */
  ME.RUN[0].B.FMPLLON  = 1;         /* Enable PLL           */
  ME.RUN[0].B.SYSCLK   = 0x4;     /* System clock is PLL */

  CGM.FMPLL_CR.B.IDF  = 0x9;        /* 10 */
  CGM.FMPLL_CR.B.ODF  = 0x3;        /* 16 */
  CGM.FMPLL_CR.B.NDIV =  48;        /* 48 */

  ME.MCTL.R = 0x40005AF0;           /* Mode & Key */
  ME.MCTL.R = 0x4000A50F;           /* Mode & Key inverted */
  while(ME.GS.B.S_MTRANS==1) {};    /* Wait for mode entry to complete */
}
```

The PA[0] pin is configured as Clock output in order to verify the clock bus frequency provided by the PLL. The CGM module allows to tune the dividers in order to provide a clock source of 12MHz starting from an external oscillator of 40 MHz.

4.2 DSPI configuration

The following function allows to initialize the DSPI module used for the SPI transmission.

It configures the DSPI_0 as master and the following pins as:

- Pad PA[12] as DSPI_0 SIN (Serial input)
- Pad PA[13] as DSPI_0 SOUT (Serial output)
- Pad PA[14] as DSPI_0 SCK (Serial Communication Clock)
- Pad PA[15] as DSPI_0 PCS0 (Peripheral Chip Select_0)
- Pad PF[10] as DSPI_0 PCS1 (Peripheral Chip Select_1)
- Pad PF[11] as DSPI_0 PCS2 (Peripheral Chip Select_2)
- Pad PE[06] as DSPI_0 PCS3 (Peripheral Chip Select_3)

The CTAR register defines the following parameters:

- Set Double Baud rate disabled
- Frame size equal to 16
- Baud Rate Prescaler value equal to 3
- Baud Rate Scaler equal to 4
- Delay after Transfer Scaler equal to 32
- PCS to SCK delay Prescaler equal to 7

Figure 13. DSPI master init function

```
void InitDSPI_master(void)
{
  SIU.PCR[12].R = 0x0103;         /* Config pad PA12 as DSPI_0 SIN, OBE = 0 */
  SIU.PCR[13].R = 0x0603;         /* Config pad PA13 as DSPI_0 SOUT */
  SIU.PCR[14].R = 0x0603;         /* Config pad PA14 as DSPI_0 SCK, output */
  SIU.PCR[15].R = 0x0603;         /* Config pad PA15 as DSPI_0 PCS0 */
  SIU.PCR[90].R = 0x0603;         /* Config pad PF10 as DSPI_0 PCS1 */
  SIU.PCR[91].R = 0x0603;         /* Config pad PF11 as DSPI_0 PCS2 */
  SIU.PCR[70].R = 0x0A03;         /* Config pad PE06 as DSPI_0 PCS3 */

  DSPI_0.MCR.R = 0x80000000;      /* Configure DSPI_0 as master */
  DSPI_0.MCR.B.PCSIS0 = 1;        /* CS: idle = high level*/
  DSPI_0.MCR.B.PCSIS1 = 1;        /* CS: idle = high level*/
  DSPI_0.MCR.B.PCSIS2 = 1;        /* CS: idle = high level*/
  DSPI_0.MCR.B.PCSIS3 = 1;        /* CS: idle = high level*/

  DSPI_0.CTAR[0].R = 0x78010001; /* Set Double Baud rate disabled, Frame size 16
                                    Baud Rate Prescaler value is 3,
                                    Baud Rate Scaler is 4 */
  DSPI_0.CTAR[0].B.DT = 0x4;      /* Delay after Transfer Scaler is 32*/

  DSPI_0.CTAR[0].B.PCSSCK = 3;    /* PCS to SCK delay prescaler=7:
                                    it's necessary to communicate with TC */
}
```

4.3 eMIOS_0 and EMIOS_1 modules

4.3.1 eMIOS_0 and eMIOS_1 module initialization

The following function allows to configure the two eMIOS and the following pins as:

- Set PCR[1] as E0UC[1] using PA[1]
- Set PCR[89] as E1UC[1] using PF[9]
- Set PCR[112] as E1UC[2] using PH[0]
- Set PCR[92] as E1UC[25] using PF[12]
- Set PCR[93] as E1UC[26] using PF[13]
- Set PCR[104] as E1UC[17] using PG[8]
- Set PCR[105] as E1UC[18] using PG[9]

Figure 14. eMIOS init function

```
void eMIOS_Config_loc(void)
{
    EMIOS_0.MCR.R   = 0;
    EMIOS_1.MCR.R   = 0;
    SIU.PCR[1].R    = 0x0600;
    SIU.PCR[89].R   = 0x0600;
    SIU.PCR[112].R  = 0x0600;
    SIU.PCR[92].R   = 0x0500;
    SIU.PCR[93].R   = 0x0500;
    SIU.PCR[104].R  = 0x0500;
    SIU.PCR[105].R  = 0x0500;
}
```

Figure 15. eMIOS clock init function

```
void ExeTest(void){

    EMIOS_1.MCR.R = (EMIOS_GPREN      /* Global Prescaler Enable */
                    |EMIOS_GPRE(0x0));/* Module Prescaler div 1  */

    EMIOS_0.MCR.R = (EMIOS_GTBE       /* Global Time Base Enable */
                    |EMIOS_GPREN      /* Global Prescaler Enable */
                    |EMIOS_GPRE(0xB));/* Module Prescaler div 12  */

}
```

For eMIOS_1, the bus clock is divided by one in the system clock Prescaler, while for eMIOS_0 the Prescaler is set to 12.

EMIOS_0 Ch23 is configured in MC to drives the Bus A with a period of 1 msec, due to the value of 1000 used as PWM_PERIOD.

eMIOS_0 CH1 is configured in SAOC mode, with an internal counter period of 300 µsec.

Figure 16. eMIOS_0 configuration code

```
* EMIOS_0 Ch23 is configured in Modulus Counter to drives the Bus A
  with a period of 1 msec( because the prescaler is 12 and the Sys Clock is 12 MHZ ---> 1MHz to the eMIOS0)*/
  EMIOS_0.CH[23].CADR.R       = PWM_PERIOD;       /* Period will be 1000 clocks (1msec) Frequency 1KHz*/
  EMIOS_0.CH[23].CCR.B.MODE   = 0x10;             /* Mode is Modulus Counter, internal clock */
  EMIOS_0.CH[23].CCR.B.BSL    = 0x3;              /* Bus Selected: all channels: internal counter */
  EMIOS_0.CH[23].CCR.B.UCPRE  = 0x0;              /* Set channel prescaler to divide by 1 */
  EMIOS_0.CH[23].CCR.B.FREN   = 0;                /* Normal Operation, no freeze channel counting when in debug mode */
  EMIOS_0.CH[23].CCR.B.UCPEN  = 1;                /* Enabled prescaler; uses default divide by 1 */
  EMIOS_0.CH[23].CCR.B.EDPOL  = 1;                /* Edge Polarity bit: trigger on a rising edge */
  EMIOS_0.CH[23].CCR.B.FEN    = 1;                /* Enable interrupt on FLAG */
 //Channel used as counter after the trigger

* EMIOS_0 Ch1 is configured in SAOC (Single Action Output Compare), with internal counter, period of 300 usec*/
  EMIOS_0.CH[1].CCR.R         = 0;                /* Initialization of the register */
  EMIOS_0.CH[1].CADR.R        = 0;
  EMIOS_0.CH[1].CBDR.R        = 0;
  EMIOS_0.CH[1].CCNTR.R       = 0;

 //*emios0_siu[1]= 0x0600; /* Set the eMIOS1 pad for eMIOS o/p SIU.PCR[1].R = 0x0600;*/
  EMIOS_0.CH[1].CADR.R        = 300;              /* Set period   300 clocks (300usec) with a internal freq of 3.3KHz */
  EMIOS_0.CH[1].CCR.B.MODE    = 0x3;              /* Mode is SAOC */
```

eMIOS_1 CH23, configured in MC, drives bus A with a period of 19.417 µsec (51.5 KHz)

eMIOS_1 CH0, configured in MC, drives bus B with a period of 10 msec (100 Hz)

eMIOS_1 CH1 is configured in OPWMT driven by bus A (51.5 KHz) with a duty cycle of 50% to give the PWM clock to M0-6 devices.

eMIOS_1 CH2 is configured in OPWMT driven by bus B, with the same period of the PWM generated by M0-6 devices (100 Hz) and it is used to give an internal counter to check the Vbat once for each period or to trigger the collection of the diagnosis collection when the M0-6 devices work in DC mode.

eMIOS_1 CH17 is configured in SAIC mode and it is connected to the Sync_mux of the M0-6_3. Internally it triggers DMA_MUX 21.

eMIOS_1 CH18 is configured in SAIC mode and it is connected to the Sync_mux of the M0-6_4. Internally it triggers DMA_MUX 22.

eMIOS_1 CH25 is configured in SAIC mode and it is connected to the Sync_mux of the M0-6_1. Internally it triggers DMA_MUX 23.

eMIOS_1 CH26 is configured in SAIC mode and it is connected to the Sync_mux of the M0-6_2. Internally it triggers DMA_MUX 24.

Figure 17. eMIOS_1 Ch23 and Ch0 configuration code

```
/* Configure the PWM Clock MO6_1 */
/* EMIOS_1 Ch23 is configured in Modulus Counter drives
    busA with a period of 19,417 usec(51.5 KHz)
   (because the prescaler is 1 and the Sys Clock is 12MHz
    --> 12 MHz to the eMIOS1*/
    EMIOS_1.CH[23].CADR.R      = COUNTER_PERIOD;/* Period will be 233 clocks (19,417 usec) Frequency 51.5 KHz*/
    EMIOS_1.CH[23].CCR.B.MODE  = 0x10;         /* Mode is Modulus Counter, internal clock */
    EMIOS_1.CH[23].CCR.B.BSL   = 0x0;          /* Use internal counter bus[A] */
    EMIOS_1.CH[23].CCR.B.UCPRE = 0x0;          /* Set channel prescaler to divide by 1 */
    EMIOS_1.CH[23].CCR.B.FREN  = 0;            /* Normal Operation, no freeze channel counting when in debug mode */
    EMIOS_1.CH[23].CCR.B.EDPOL = 1;            /* Edge Polarity bit: trigger on a rising edge */
    EMIOS_1.CH[23].CCR.B.FEN   = 1;            /* Enable interrupt on FLAG */

/* EMIOS_1 ch0 is configured in Modulus Counter drives
    busB with a period of 10 msec (100 Hz)
    ( because the Bus[A] has internal frequency of 51.5 KHz*/
    EMIOS_1.CH[0].CADR.R       = LAMP_PERIOD; /* Period will be 515 clocks (10 msec) Frequency 100 Hz*/
    EMIOS_1.CH[0].CCR.B.MODE   = 0x10;        /* Mode is Modulus Counter, internal clock */
    EMIOS_1.CH[0].CCR.B.BSL    = 0x0;         /* Use internal counter */
    EMIOS_1.CH[0].CCR.B.UCPRE  = 0x3;         /* Set channel prescaler to divide by 4 */
    EMIOS_1.CH[0].CCR.B.FREN   = 0;           /* Freeze channel counting when in debug mode */
    EMIOS_1.CH[0].CCR.B.EDPOL  = 1;           /* Edge Polarity bit: trigger on a rising edge */
    EMIOS_1.CH[0].CCR.B.FEN    = 1;           /* Enable interrupt on FLAG */
```

Figure 18. eMIOS_1 Ch1 and Ch2 configuration code

```
   // PWM Clock
/* EMIOS_1 ch1 is configured in OPWMT driven by busA (51.5 KHz)
   with a duty cycle of 50% to give the PWM clock to MO6 devices*/
  //*emios1_siu[1] = 0x0600; /* Set the eMIOS1 pad for eMIOS o/p */ //SIU.PCR[89].R = 0x600//
    EMIOS_1.CH[1].CADR.R       = 1;                    /* Set Initial Duty Cycle  */
    EMIOS_1.CH[1].CBDR.R       = COUNTER_PERIOD/2;     /* Set Initial Duty Cycle  */
    EMIOS_1.CH[1].CCR.R        = (EMIOS_UCPRE(0)       /* Channel Prescaler div 1 */
                                 |EMIOS_EDPOL
                                 |EMIOS_BSL(0x0)       /* Select Bus A */
                                 |0xA6);               /* Set mode OPWMT and EDPOL 1 to trigger on a rising edge */

    // Timer for VBat read
/*  EMIOS_1 ch2 is configured in OPWMT driven by busB (100Hz)
    and it is used to give internal counter to check the Vbat
    once for each period or to trigger the collection of the
    diagnosis when the MO6 devices work in DC mode.*/
  //*emios1_siu[2] = 0x0600; /* Set the eMIOS1 pad for eMIOS o/p SIU.PCR[112].R = 0x0600*/
    EMIOS_1.CH[2].CADR.R       = 1;                   /* Set Initial Duty Cycle   */
    EMIOS_1.CH[2].CBDR.R       = LAMP_PERIOD/2;       /* Set Initial Duty Cycle   */
    EMIOS_1.CH[2].ALTCADR.R    = 1;                   /* Set period */
    EMIOS_1.CH[2].CCR.R        = (EMIOS_UCPRE(0x0)    /* Channel Prescaler div 1 */
                                 |EMIOS_EDPOL
                                 |EMIOS_BSL(0x1)      /* Select Bus B */
                                 |EMIOS_FEN
                                 |EMIOS_DMA
                                 |0xA6);              /* Set mode OPWMT and EDPOL 1 to trigger on a rising edge */
```

Figure 19. eMIOS_1 Ch25, Ch26 and Ch 17 configuration code

```
/* EMIOS_1 ch 25 is configured in SAIC mode and it is connected
  to the Sync_mux of the MO6_1. Internally it triggers DMA_MUX23*/
   //*emios1_siu[25] = 0x0500; /* Set the eMIOS2 pad for eMIOS input SIU.PCR[92].R = 0x0500*/
    EMIOS_1.CH[25].CCR.R        = (EMIOS_UCPRE(0)       /* Channel Prescaler div 1 */
                                  |EMIOS_UCPREN
                                  |EMIOS_EDPOL
                                  |EMIOS_FEN
                                  |EMIOS_DMA
                                  |EMIOS_BSL(0x0)       /* Select Bus A  */
                                  |0x2);                /* Set mode SAIC */

/* EMIOS_1 ch 26 is configured in SAIC mode and it is connected
  to the Sync_mux of the MO6_2. Internally it triggers DMA_MUX24*/
    //*emios1_siu[26] = 0x0500; /* Set the eMIOS2 pad for eMIOS input SIU.PCR[93].R = 0x0500*/
    EMIOS_1.CH[26].CCR.R        = (EMIOS_UCPRE(0)       /* Channel Prescaler div 1 */
                                  |EMIOS_UCPREN         /* Enable ch Prescaler     */
                                  |EMIOS_EDPOL
                                  |EMIOS_FEN
                                  |EMIOS_DMA
                                  |EMIOS_BSL(0x0)       /* Select Bus A            */
                                  |0x2);                /* Set mode SAIC           */
/* EMIOS_1 ch 17 is configured in SAIC mode and it is connected
  to the Sync_mux of the MO6_3. Internally it triggers DMA_MUX21*/
 //*emios1_siu[17] = 0x0500; /* Set the eMIOS2 pad for eMIOS input SIU.PCR[104].R = 0x0500*/
    EMIOS_1.CH[17].CCR.R        = (EMIOS_UCPRE(0)       /* Channel Prescaler div 1 */
                                  |EMIOS_UCPREN         /* Enable ch Prescaler     */
                                  |EMIOS_EDPOL
                                  |EMIOS_FEN
                                  |EMIOS_DMA
                                  |EMIOS_BSL(0x0)       /* Select Bus A            */
                                  |0x2);                /* Set mode SAIC           */
```

Figure 20. eMIOS_1 Ch18 configuration code

```
/* EMIOS_1 ch 18 is configured in SAIC mode and it is connected
   to the Sync_mux of the MO6_4. Internally it triggers DMA_MUX22*/
//*emios1_siu[18] = 0x0500; /* Set the eMIOS2 pad for eMIOS input SIU.PCR[105].R = 0x0500*/
    EMIOS_1.CH[18].CCR.R        = (EMIOS_UCPRE(0)       /* Channel Prescaler div 1 */
                                  |EMIOS_UCPREN         /* Enable ch Prescaler     */
                                  |EMIOS_EDPOL
                                  |EMIOS_FEN
                                  |EMIOS_DMA
                                  |EMIOS_BSL(0x0)       /* Select Bus A            */
                                  |0x2);                /* Set mode SAIC           */
    EMIOS_1.CH[18].CSR.R    = 1;
    INTC.PSR[166].B.PRI     = 1;              /* Set to 1 the priority of the EMIOSFLAG register [F18,F19] */
    count = 0;

    EMIOS_1.CH[23].CCR.B.UCPEN  = 1;          /* Enable prescaler; uses default divide by 1 */
    EMIOS_1.CH[0].CCR.B.UCPEN   = 1;          /* Enable prescaler; uses default divide by 1 */
}
```

4.4 DMA module

The initDMA_Vbat function permits to configure DMA channel number 6 to be used to transfer, by SPI, the configuration to M0-6 devices about the PWM waves generation.

This DMA channel is configured to be triggered by DSPI0_TX signal each time when Vbat is read and PWM configuration is needed.

The scatter gather feature allows to change the TCD descriptor to send the duty-cycle configuration for 16 channels, for all M0-6 (4 channels for 4 M0-6) and it allows to change the TCD descriptor to wait the new manual trigger to send M06_PWM control register for the next period.

Figure 21. Init DMA Vbat function

```
tU16 initDMA_Vbat (void)
{
    tU32 fail  = 0;
    //SPI_VBat_SetTCD(*tcd,         ch,    start,         dest,      inSize, outLoop, interrupt);
    SPI_VBat_SetTCD  (&tcdSG_PWM[0], 0, (tU32)msg_PWM, (tU32)0xFFF90034, 4, M06_NUMBER, 0);     //Create a the TCD structure for the SG_PWM vector[0]
    tcdSG_PWM[0].D_REQ      = 1;                    // The channel's DMAERG bit is cleared when the outer major loop is complete
    tcdSG_PWM[0].DLAST_SGA  = (tU32) &tcdSG_PWM[1];     // point to address of TCD for scatter gather
    tcdSG_PWM[0].E_SG       = 1;                    // Enable Scatter/Gather
    tcdSG_PWM[0].D_REQ      = 0;                    // first transfer will not clear ERQR, 2nd one will


    SPI_VBat_SetTCD  (&tcdSG_PWM[1], 0, (tU32)DMA_M06_VBAT, (tU32)0xFFF90034, 4, M06_NUMBER*NCHANNEL, 0); //Create a the TCD structure for the SG_PWM vector[1]
    tcdSG_PWM[1].DLAST_SGA  = (tU32) &tcdSG_PWM[0];     // point to address of TCD for scatter gather
    tcdSG_PWM[1].E_SG       = 1;                    // Enable Scatter/Gather
    tcdSG_PWM[1].D_REQ      = 1;                    // this transfer will clear ERQR

    SPI_VBat_SetTCD  (0, 6, (tU32)msg_PWM, (tU32)0xFFF90034, 4, M06_NUMBER, 0); //Set the TCD for the SPI communication of the msg_PWM using the EDMA ch6
    EDMA.TCD[6].DLAST_SGA   = (tU32) &tcdSG_PWM[1];     // point to address of TCD for scatter gather
    EDMA.TCD[6].E_SG        = 1;                    // Enable Scatter/Gather
    EDMA.TCD[6].D_REQ       = 0;                    // this transfer will not clear ERQR


    DMAMUX.CHCONFIG[6].B.SOURCE = 1;                // DMA Channel Source, specifies which DMA source is routed to a particular DMA Channel
    DMAMUX.CHCONFIG[6].B.ENBL   = 0x1;              // DMA Channel enabled

    if (fail)
      return (FAIL);
    else
      return (PASS);
}
```

The function SPI_Vbat_SetTCD allows to create a TCD structure for the SG_PWM vectors 0 and 1 and for the DMA channel 6.

Figure 22. SPI_Vbat_SetTCD function

```
void SPI_VBat_SetTCD  (volatile struct EDMA_TCD_STD_tag *tcd, tU16 ch, tU32 start, tU32 dest, tU32 inSize,
{

  if (tcd == 0) {   tcd = &EDMA.TCD[ch];  }

/* Set up TCD */
    tcd->SADDR = start;             //  Start address
    tcd->DADDR = dest;              //  Destination Address
    tcd->SMOD = 0x00;               //  Source address modulo
    tcd->DMOD = 0x00;               //  Destination address modulo
    tcd->DSIZE = 2;                 //  Destination transfer size : 32 Bits
    tcd->SSIZE = 2;                 //  Source transfer size : 8 Bits
    tcd->SOFF = 4;                  //  Signed source address offset
    tcd->NBYTES = inSize;           //  Inner "minor" byte count
    tcd->SLAST = 0;                 // last Signed source address adjust
    tcd->DOFF = 0;                  //  Signed destination address offset
    tcd->DLAST_SGA = 0;             //  Signed destination address adjust
    tcd->BITERE_LINK = 0x0;
    tcd->BITER = outLoop;           //  begining "major" iteration count
    tcd->CITERE_LINK = 0x0;
    tcd->CITER= outLoop;            //  Current "major" iteration count Disabled
    tcd->BWC = 0x0;                 //  Bandwidth control :  No DMA Stalls
    tcd->MAJORLINKCH = 0x00;        //  Major Channel number
    tcd->MAJORE_LINK = 0;           //  Major Channel Link : Disabled
    tcd->DONE = 0;                  //  Channel Done
    tcd->ACTIVE = 0;                //  Channel ACtive
    tcd->E_SG = 0;                  //  Enable Scatter/Gather : Disabled
    tcd->D_REQ = 0;                 //  Disable TCD When done
    tcd->INT_HALF = 0x0;            //  Interrupt on minor loop count : Disabled
    tcd->INT_MAJ = interr;          //  Interrupt on major loop completion

}   /* end of SPI SetTCD  */
```

The function InitDMA_1 configures the eDMA channels to collect the diagnosis automatically. For this reason the following channels are configured as described below:

- DMA CH0, CH7, CH8 and CH9 are configured to trigger the start counter of the eMIOS0_CH1 when the SYNC_MUX comes from M0-6_1, M0-6_2, M0-6_3 and M0-6_4 respectively.
- DMA CH1 is configured to trigger the stop of counter of the eMIOS0_CH1
- DMA CH1 is linked to DMA CH2. In this way when the CH1 finishes it starts the DMA CH2 transmission which allows the start of the ADC conversion.
- DMA CH5 is configured to send, by SPI, the configuration to the MUX for the active M0-6 to select the right channel connected to the CS and with CS_SYNC.
- DMA CH4 is configured to transfer the value of the 4 Data Registers relative to the 4 ADC channels linked to the 4 M0-6 to a buffer in the memory, this is done at each trigger event because it is necessary to read all ADC channels that can trigger DMA transfers.
- DMA CH10 is configured to transfer the value converted by ADC to the final RAM structure. The scatter gather feature is used to change the TDC to adapt the ADC Data Register and Memory location relative to the M0-6 Channel selected.

Figure 23. initDMA_1 function part 1

```
tU16 initDMA_1 (void)
{
    tU32 fail  = 0;
    DMA_STEP1[0] = (EMIOS_UCPRE(0)       /* Channel Prescaler div 1 */
                   |EMIOS_UCPREN         /* Enable ch Prescaler     */
                   |EMIOS_EDPOL
                   |EMIOS_FEN
                   |EMIOS_DMA
                   |EMIOS_BSL(0x3)       /* Select Internal Bus     */
                   |0x3);                /* Set mode SAOC           */

    /* Ch0 of DMA is configured to trigger the start counter of the eMIOS0 ch1 when arrives the SYNC_MUX from M06_1*/
    SetTCD  (0, 0, (tU32)DMA_STEP1, (tU32)&(EMIOS_0.CH[1].CCR.R), 4, 1, 0);   //Start SAOC M06_1
    /* Ch7 of DMA is configured to trigger the start counter of the eMIOS0 ch1 when arrives the SYNC_MUX from M06_2*/
    SetTCD  (0, 7, (tU32)DMA_STEP1, (tU32)&(EMIOS_0.CH[1].CCR.R), 4, 1, 0);   //Start SAOC M06_2
    /* Ch8 of DMA is configured to trigger the start counter of the eMIOS0 ch1 when arrives the SYNC_MUX from M06_3*/
    SetTCD  (0, 8, (tU32)DMA_STEP1, (tU32)&(EMIOS_0.CH[1].CCR.R), 4, 1, 0);   //Start SAOC M06_3
    /* Ch9 of DMA is configured to trigger the start counter of the eMIOS0 ch1 when arrives the SYNC_MUX from M06_4*/
    SetTCD  (0, 9, (tU32)DMA_STEP1, (tU32)&(EMIOS_0.CH[1].CCR.R), 4, 1, 0);   //Start SAOC M06_4
    /* Ch1 of DMA is configured to trigger the stop of counter of the eMIOS0 ch1 */
    SetTCD  (0, 1, (tU32)DMA_STEP2, (tU32)&(EMIOS_0.CH[1].CCR.R), 4, 1, 0);   //STOP SAOC M01

    EDMA.TCD[1].MAJORE_LINK = 1;      /* enable link on major loop */
    /* Ch1 is linked to Ch2 so when it finish its transfer start ch2 and allows to start the ADC conversion*/
    EDMA.TCD[1].MAJORLINKCH = 2;      /* link to next channel */
```

Figure 24. initDMA_1 function part 2

```
/* Ch2 of DMA is configured to trigger the start of conversion of the ADC*/
SetTCD  (0, 2, (tU32)DMA_STEP3, (tU32)0xFFE00000, 4, 1, 0);              //Start ADC conversion

SetTCD  (&tcdSG[0], 0, (tU32)ADC_DR_MO6[1], (tU32)&CS_MO6[1], 4, 4, 0);   //Moves read CS in the memory
tcdSG[0].DOFF           = 4;              /* Signed destination address offset */
tcdSG[0].CITERE_LINK    = 1;              /* enable link on minor loop */
tcdSG[0].BITERE_LINK    = 1;              /* enable link on minor loop */
tcdSG[0].CITER      |= 5<<9;              /* link to next channel */
tcdSG[0].BITER      |= 5<<9;              /* link to next channel */
tcdSG[0].MAJORE_LINK    = 1;              /* enable link on major loop */
tcdSG[0].MAJORLINKCH    = 5;              /* link to next channel */
tcdSG[0].DLAST_SGA      = (tU32) &tcdSG[1]; /* point to address of TCD for scatter gather */
tcdSG[0].E_SG           = 1;              /* Enable Scatter/Gather */
tcdSG[0].D_REQ          = 0;              /* first transfer will not clear ERQR, 2nd one will */

SetTCD  (&tcdSG[1], 0, (tU32)ADC_DR_MO6[2], (tU32)&CS_MO6[2], 4, 4, 0);   //Moves read CS in the memory
tcdSG[1].DOFF           = 4;              /* Signed destination address offset */
tcdSG[1].CITERE_LINK    = 1;              /* enable link on minor loop */
tcdSG[1].BITERE_LINK    = 1;              /* enable link on minor loop */
tcdSG[1].CITER      |= 5<<9;              /* link to next channel */
tcdSG[1].BITER      |= 5<<9;              /* link to next channel */
tcdSG[1].MAJORE_LINK    = 1;              /* enable link on major loop */
tcdSG[1].MAJORLINKCH    = 5;              /* link to next channel */
tcdSG[1].DLAST_SGA      = (tU32) &tcdSG[2]; /* point to address of TCD for scatter gather */
tcdSG[1].E_SG           = 1;              /* Enable Scatter/Gather */
tcdSG[1].D_REQ          = 0;              /* first transfer will not clear ERQR, 2nd one will */

SetTCD  (&tcdSG[2], 0, (tU32)ADC_DR_MO6[3], (tU32)&CS_MO6[3], 4, 4, 0);   //Moves read CS in the memory
tcdSG[2].DOFF           = 4;              /* Signed destination address offset */
tcdSG[2].CITERE_LINK    = 1;              /* enable link on minor loop */
tcdSG[2].BITERE_LINK    = 1;              /* enable link on minor loop */
tcdSG[2].CITER      |= 5<<9;              /* link to next channel */
tcdSG[2].BITER      |= 5<<9;              /* link to next channel */
tcdSG[2].MAJORE_LINK    = 1;              /* enable link on major loop */
tcdSG[2].MAJORLINKCH    = 5;              /* link to next channel */
tcdSG[2].DLAST_SGA      = (tU32) &tcdSG[3]; /* point to address of TCD for scatter gather */
tcdSG[2].E_SG           = 1;              /* Enable Scatter/Gather */
tcdSG[2].D_REQ          = 0;              /* first transfer will not clear ERQR, 2nd one will */
```

Figure 25. init DMA_1 function part 3

```
SetTCD  (&tcdSG[3], 0, (tU32)ADC_DR_MO6[0], (tU32)&CS_MO6[0], 4, 4, 0);   //Moves read CS in the memory
tcdSG[3].DOFF           = 4;                /* Signed destination address offset */
tcdSG[3].CITERE_LINK    = 1;                /* enable link on minor loop */
tcdSG[3].BITERE_LINK    = 1;                /* enable link on minor loop */
tcdSG[3].CITER      |= 5<<9;                /* link to next channel */
tcdSG[3].BITER      |= 5<<9;                /* link to next channel */
tcdSG[3].MAJORE_LINK    = 1;                /* enable link on major loop */
tcdSG[3].MAJORLINKCH    = 5;                /* link to next channel */
tcdSG[3].DLAST_SGA      = (tU32) &tcdSG[0]; /* point to address of TCD for scatter gather */
tcdSG[3].E_SG           = 1;                /* Enable Scatter/Gather */
tcdSG[3].D_REQ          = 0;                /* first transfer will not clear ERQR, 2nd one will */

/* Ch4 of DMA is configured to transfer the Value converted by ADC to a RAM structure. The scatter gater
SetTCD  (0, 4, (tU32)ADC_DR_MO6[0], (tU32)CS_MO6, 4, 4, 0);          //Moves read CS in the memory
EDMA.TCD[4].DOFF        = 4;                /* Signed destination address offset */
EDMA.TCD[4].DLAST_SGA   = (tU32) &tcdSG[0]; /* point to address of TCD for scatter gather */
EDMA.TCD[4].CITERE_LINK = 1;                /* enable link on minor loop */
EDMA.TCD[4].BITERE_LINK = 1;                /* enable link on minor loop */
EDMA.TCD[4].CITER   |= 5<<9;                /* link to next channel */
EDMA.TCD[4].BITER   |= 5<<9;                /* link to next channel */
EDMA.TCD[4].MAJORE_LINK = 1;                /* enable link on major loop */
EDMA.TCD[4].MAJORLINKCH = 5;                /* link to next channel */
EDMA.TCD[4].E_SG        = 1;                /* Enable Scatter/Gather */
EDMA.TCD[4].D_REQ       = 0;                /* first transfer will not clear ERQR, 2nd one will */

/* Ch5 pf DMA is configured to send, by SPI, the configuration to the MUX for the active MO6 to select th
SetTCD  (0, 5, (tU32)SEQ_MO6, (tU32)0xFFF90034, 4, 16, 0);           //Configures the next CS
EDMA.TCD[5].SOFF        = 4;                /* Signed source address offset */
EDMA.TCD[5].SLAST = -(4*16);                /* last Signed source address adjust */

EDMA.SERQ.R = 1; //Set enable request and the corresponding bit in EMDA_ERQRL
EDMA.SERQ.R = 4; //Set enable request and the corresponding bit in EMDA_ERQRL
```

4.5 ADC module

The dadc_init function initializes ADC module enabling the CTU trigger. It selects the ADC channels used for the conversion. ADC_0 CH0 is used to convert the VBat and a End of Conversion interrupt is set on this channel.

EMIOS1_CH2 is used as counter to read Vbat. When this counter elapses, the CTU_CH34 (connected to eMIOS1_CH2) will trigger the ADC_0 CH0 to convert the Vbat.

At the EoC an interrupt will be raised to adapt the Duty Cycle of the PWM signals to the Vbat.

Figure 26. ADC init function

```
void dadc_init(void)
{
//ADC_0 ch0 is used to convert the VBat and it is configured an
//interrupt at the EOC of this channel. eMIOS1 ch2 is used as counter to read VBat.
//When the counter elapses, the CTU Ch34 (connected to the EMIOS1 ch2)
// will trigger the ADC0 ch0 to convert the Vbat.
//At the EOC an interrupt will be raisen to adapt the Duty Cycle of the
// PWM signals to the VBat
    vuint8_t i, j;
    for (i=20; i < 26; i++) //28
    {
        SIU.PCR[i].B.APC = 1;
    }

    for (i=48; i < 60; i++)
    {
        SIU.PCR[i].B.APC = 1;
    }

    ADC_0.MCR.B.PWDN   = 0;      // exit from power down state (offset cancellation i
    ADC_0.MCR.B.CTUEN  = 1;      // Enable the CTU
    ADC_0.MCR.B.MODE   = 0;      // Configure One shot conversion

    ADC_0.NCMR0.R = 0x1E;        // Normal Conversion Mask Register Enables bits
                                 //  of normal sampling for channel 1, 2, 3, 4

    ADC_0.IMR.B.MSKEOCTU = 1;    // Interrupt Mask Register Mask Enabled End of CTU C
    ADC_0.CIMR0.B.CIM0   = 1;    // Channel Interrupt Mask Register Enables Interrupt
    INTC.PSR[62].B.PRI   = 1;    // Set priority for the ADC_EOC end of conversion

    ADC_0.DMAR0.B.DMA4 = 0x1;    // DMA Channel Select Register enable channel 4 to c
    ADC_0.DMAE.B.DMAEN   = 1;    // DMA enable register: DMA feature enabled.

    for (i=0;i<NCHANNEL; i++)
    {
      for (j = 0; j < MD6_NUMBER; j++)
      {
        DMA_MD6_VBAT[j][i]=(0x1<< (16+j)) +msg[i];
      }
    }
```

Figure 27. ADC_0 CH0 EoC interrupt service routine

```
void ADC0_EOC_Isr(void)
{
    if (ADC_0.ISR.B.EOCTU && ADC_0.CEOCFR0.B.EOC_CH0)
    {
        SIU.GPDO[2].B.PDO ^= 1;
        ADC_0.ISR.B.EOCTU =1;
        ADC_0.CEOCFR0.B.EOC_CH0 = 1;
        checkVoltageBattery();
        SIU.GPDO[2].B.PDO ^= 1;
    }
}
```

4.5.1 Check Voltage Battery function

This function is called inside the EOC interrupt of the ADC, when the conversion is triggered by the CTU and only for the ADC0_CH0.

It performs a reading on the Vbat value and carries out the duty cycle of the PWM signals update.

The voltage read with a nominal value of the Battery voltage (13.2 V) is equal to:

13,2*4,7/(4,7+22) =2,32V with a resistor divider of 22KOhm and 4.7 KOhm

The resolution of the ADC is:

5V/1024=4,88mV

Reporting this resolution to the value of the Battery using the resistor divider:

4.88*(4,7+22)/4,7 = 28,404 mV

that is the resolution of voltage of the battery (V_BAT_RESOLUTION)

When the value is read, following the relation $\delta = \frac{Vref^2}{Vbat^2}$:

if Vref ≥ Vbat then the battery voltage is under than nominal (13,2 V) value and the output channels of the M0-6 devices are configured in DC mode (100% Duty Cycle) and the trigger to read the diagnosis is given manually with the same period used to read the Vbat.

If Vref < Vbat the battery voltage is upper than the nominal value and the output channels of the M0-6 devices are configured in PWM mode with Duty Cycle based on the above relation. The trigger, in order to read the diagnosis, is given by the Sync_Mux signal from each M0-6 device.

After that the table of PWM signals are updated and these values are sent to each M0-6 devices using the SPI connection.

Figure 28. Check Voltage Battery function part1

```
void checkVoltageBattery (void)
{
    uint8_t i, j;
    valore_app = ADC_0.CDR[0].B.CDATA;
    valore3 = V_BAT_NOM * 1000;
    valore4 = V_BAT_RESOLUTION;
    valore2 = V_BAT_NOM_ADC;
    if (valore_app > 0)
    {
        valore = ((V_BAT_NOM_ADC * V_BAT_NOM_ADC) * 100) / ((valore_app * valore_app));
        if (valore >= 100)
        {
            valore = 100;
            EDMA.CERQ.R = 0; //Clear enable Request, clear corresponding bit in EDMA_ERQRL
            EDMA.CERQ.R = 7; //Clear enable Request, clear corresponding bit in EDMA_ERQRL
            EDMA.CERQ.R = 8; //Clear enable Request, clear corresponding bit in EDMA_ERQRL
            EDMA.CERQ.R = 9; //Clear enable Request, clear corresponding bit in EDMA_ERQRL
            for (i = 0; i < MC6_NUMBER; i++)
            {
              txSpi(&msg_PWM_0[i]); //SPI transmission
            }
            if (MC6_config == 1)
            {
                for (i = 0; i < MC6_NUMBER*NCHANNEL; i++)
                {
                    EDMA.TCD[2].START = 1;
                    for (j = 0; j < 100; j++);
                }
            }
        }else
            {
                msg_PWM[0] = 0x00010000 + ((WRITE_MODE|MC6_PWMCR)<<8) | (vuint16_t) 0xF;
                msg_PWM[1] = 0x00020000 + ((WRITE_MODE|MC6_PWMCR)<<8) | (vuint16_t) 0xF;
                msg_PWM[2] = 0x00040000 + ((WRITE_MODE|MC6_PWMCR)<<8) | (vuint16_t) 0xF;
                msg_PWM[3] = 0x00080000 + ((WRITE_MODE|MC6_PWMCR)<<8) | (vuint16_t) 0xF;

                EDMA.SERQ.R = 0; //Set enable Request, set corresponding bit in EDMA_ERQRL
                EDMA.SERQ.R = 7; //Set enable Request, set corresponding bit in EDMA_ERQRL
                EDMA.SERQ.R = 8; //Set enable Request, set corresponding bit in EDMA_ERQRL
                EDMA.SERQ.R = 9; //Set enable Request, set corresponding bit in EDMA_ERQRL
```

Figure 29. Check Voltage Battery function part2

```
            valore = valore * 255 / 100;

            msg[0] = ((WRITE_MODE|MD6_DUTYC0CR)<<8) | (vuint16_t) valore;
            msg[1] = ((WRITE_MODE|MD6_DUTYC1CR)<<8) | (vuint16_t) valore;
            msg[2] = ((WRITE_MODE|MD6_DUTYC2CR)<<8) | (vuint16_t) valore;
            msg[3] = ((WRITE_MODE|MD6_DUTYC3CR)<<8) | (vuint16_t) valore;

            for (i=0;i<NCHANNEL; i++)
            {
                for (j = 0; j < MD6_NUMBER; j++)
                {
                    DMA_MD6_VBAT[j][i]=(0x1<< (16+j)) + ((vuint16_t) msg[i]);
                }
            }

            EDMA.SERQ.R = 6; //Set enable Request, set corresponding bit in EI
            DSPI_0.RSER.B.TFFFDIRS = 1; //DMA/Interrupt Request Select and ena
            DSPI_0.RSER.B.TFFFRE   = 1; //Transmit FIFO Fill Request enabled.
        }
}else
    {
        valore = 100;
        EDMA.CERQ.R = 0; //Clear enable Request, clear corresponding bit in EI
        EDMA.CERQ.R = 7; //Clear enable Request, clear corresponding bit in EI
        EDMA.CERQ.R = 8; //Clear enable Request, clear corresponding bit in EI
        EDMA.CERQ.R = 9; //Clear enable Request, clear corresponding bit in EI
        initDMA_1 (); /* Configure the eDMA channels to collect the diagnosis
        for (i = 0; i < MD6_NUMBER; i++)
        {
            txSpi(&msg_PWM_0[1]); // Transmit the PWM via SPI
        }

        if (MD6_config == 1)
        {
            for (i = 0; i < MD6_NUMBER*NCHANNEL; i++)
            {
                EDMA.TCD[2].START = 1;
                for (j = 0; j < 100; j++);
            }
        }
```

4.6 CTU module

The bctu_init function configures the CTU channels as follows; CTU CH34 is connected to the eMIOS1_CH2 and it is configured to trigger the ADC conversion of the ADC0_CH0.

Figure 30. bctu_init function

```
void bctu_init(void)
{
    CTU.EVTCFGR[34].B.TM            = 1;        // Trigger Enabled
    CTU.EVTCFGR[34].B.CHANNEL_VALUE = 0;        // Selected channel 0
}
```

4.7 Micro Resources summary

The Lighting Application example allocates the several resources inside the micro:

Table 1. Resources allocated on the microcontroller

Peripheral name	Resource	Notes
DSPI	DSPI_0	7 pins
INTC	IRQ 62 IRQ 0	ADC_EOC Software interrupt
eDMA	CH6-Vbat measure CH0 - Start counter M0-6-1 CH7-Start counter M0-6-2 CH8 - Start counter M0-6-3 CH9 - Start counter M0-6-4 CH1 Start ADC conversion CH4 Transfer data register of ADC in RAM CH10 Transfer value in final array CH5 Trigger SPI communication to select the M0-6 channel	10 DMA channels used
eMIOS	eMIOS0 - 2 Channels eMIOS1 - 8 Channels	10 eMIOS channels used
DADC	CH0 for Vbat CH1, CH2, CH3, CH4 to convert the CS	5 ADC channels used
BCTU	CTU_CH34 eMIOS1_CH2 vs ADC0_CH0	1 CTU channel used

5 Conclusion

The SPC56EC74 lighting application example introduces MCU eMIOS, CTU, and ADC modules usage within a lighting application, which is in this case well suited for the lamp light bulbs control as well as for the lamp LEDs control with no major change required in the peripherals software.

The benefit of using the CTU module is significant as opposed to using the timer to trigger ADC at a certain point within a PWM channel cycle. The timer produces frequent interrupts that need to be processed by the MCU CPU, causing an increase in the CPU load and suspending run time processes. Using the CTU gives the application designer a powerful tool to avoid diagnostic timer-configuration challenges and makes the lighting application simpler to design and test.

The ADC module gives the designer the opportunity to use watchdog threshold interrupts to increase MCU performance, and enables fast responses in case of lamp light bulb failures. If required, the number of the ADC channels can be increased from 64 up to 95, using simple external logic circuitry fully controlled by the ADC module.

The eMIOS module offers up to 64 PWM channels suitable for a lighting application as well as for other PWM-based applications. The eMIOS module provides PWM channels with high resolution and flexibility.

The lighting application example consists of an example description and complete software demonstration based on the SPC56EC74L7 LQFP176 minimodule and EVB.

This example is easily portable to each device of the SPC56xBx/RPC56xBx/SPC56xCx family; for devices which have DMA module, no particular modifications are required. For devices where DMA module is missing, the DMA feature can be replaced with interrupt routines but with a higher CPU load.

Appendix A Reference documents

A.1 Reference documents

- SPC560B4x, SPC560B5x, SPC560C4x, SPC560C5x, 32-bit MCU family built on the embedded Power Architecture® (RM0017), available at www.st.com
- SPC564Bxx, SPC56ECxx 32-bit MCU family built on the embedded Power Architecture® (RM0070 DOC ID18196)

A.2 Acronyms

Table 2. Acronyms

Acronym	Name
ADC	Analog to Digital Converter
ALTCADR	eMIOS UC A2 Register
CADR	eMIOS UC A1 Register
CBDR	eMIOS UC B2 Register
CTU	Cross Triggering Unit
EMC	Electromagnetic Compatibility
eMIOS	Configurable enhanced Modular I/O Subsystem
EVB	Evaluation Board
LA	Lighting Application
MC	eMIOS channel Modulus Counter mode
MCU	Microcontroller Unit (SPC560B50)
OPWMT	eMIOS channel Output PWM mode with Trigger
PCR	Pad Configuration Register
PWM	Pulse Width Modulation
SIUL	System Integration Unit Lite
SAIC	Single Action Input Capture
SAOC	Single Action Output Compare
DMA	Direct Memory Access
UC	eMIOS Universal Counter channel

Revision history

Table 3. Document revision history

Date	Revision	Changes
13-Jun-2013	1	Initial release
17-Sep-2013	2	Updated Disclaimer.
08-Oct-2015	3	Robust root part numbers added.