AN204825

F²MC-16FX Family, Lamp Control and Monitor with PPG and ADC

This application note describes how to control and monitor the lights used in a dashboard or a HVAC system with the usage of PPG and ADC.

Contents

Introduction

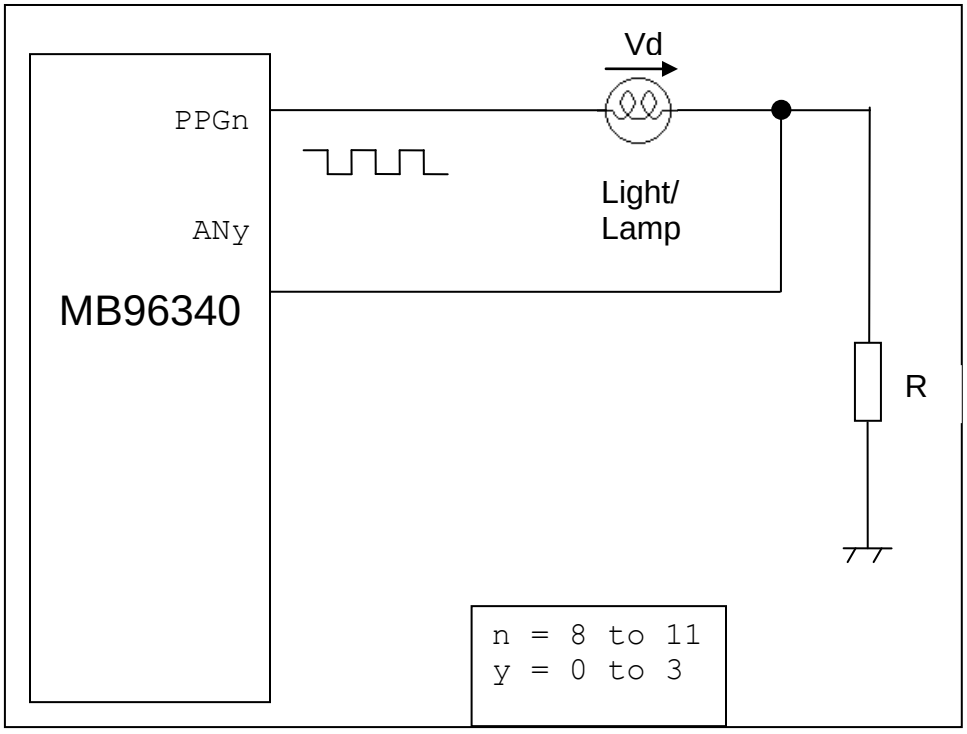
This application note describes how to control and monitor the lights used in a dashboard or a HVAC system with the usage of PPG and ADC.

In such automotive applications the light intensity is controlled by the PPG outputs using the PWM technique. However, it is also essential to monitor whether a particular light is on and functioning properly. This is achieved by measuring the voltage across the lamp with an ADC.

Connection Diagram

The following figure shows the connection diagram. The Programmable Pulse Generator (PPG) output is connected to one terminal of the light/lamp and the other terminal is pulled down via a resistor and also connected to the Analog-to-Digital Converter (ADC) input.

Figure 1. Connection Diagram - Lamp Control and Monitor



In the above diagram the PPG8 output corresponds to the AN0 input, PPG9 output corresponds to the AN1 input, so on and so forth. For the ease of understanding just one pair of PPG output and ADC input is shown.

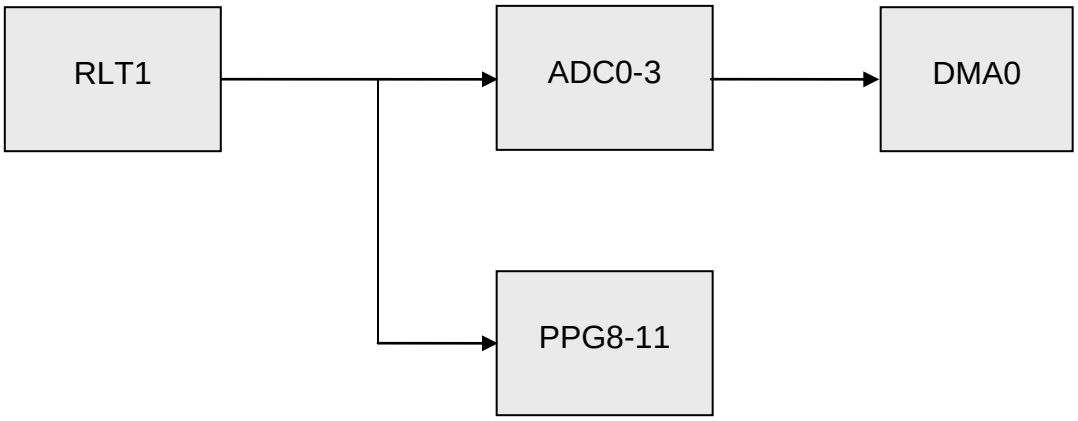
For PPG outputs, PWM frequency of 250 Hz is used. This frequency is chosen so that it is high enough for the flicker in the lamp to be invisible and it's also low enough as not to introduce EMI noise.

It should be noted that the PPG output and the ADC input may not be able to connect directly to the lamp as shown in the above figure since the power requirements of the lamp may not be compatible with that of the micro. In such scenario, there might be the need of some driver or signal conditioning circuitry.

Operation

The below block diagram depicts the peripheral operation and dependency.

Figure 2.Block Diagram - Peripheral Operation and Dependency



Reload Timer

The Reload Timer 1 (RLT1) is used as the time base for triggering the ADC conversion as well as starting the PPGs. This means the ADC conversion on channels AN0-3 are triggered by RLT1 internal output.

RLT1 is required to issue an interrupt at an interval of 0.5 ms at 16 MHz CLKP1. Hence the RLT1 prescaler is configured to divide the CLKP1 by 16 and the reload register of RLT1 is loaded with 499. This is because the underflow would happen after value loaded with reload register + 1 cycle, i.e. after 500 clock cycles of RLT1. The following formula explains the same:

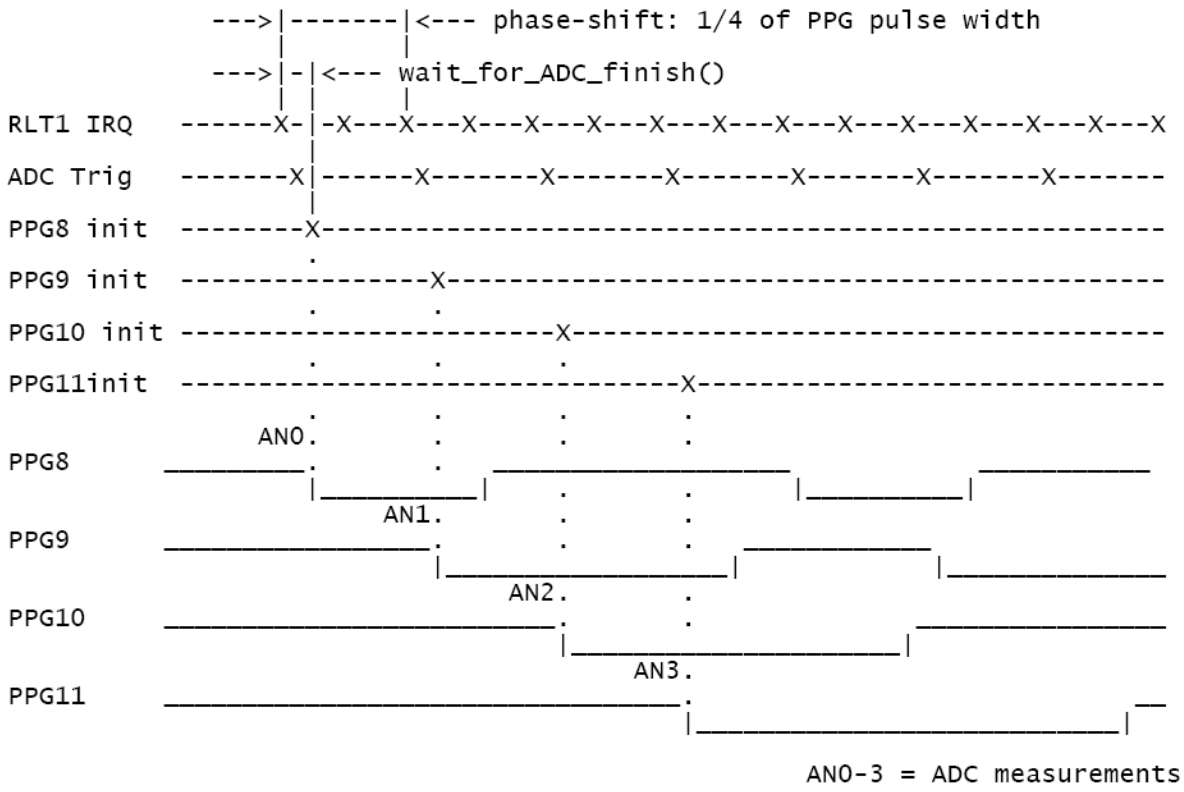
$$\text{Underflow Interrupt Interval} = \frac{1}{16MHz/16} \cdot (499+1) = 0.5ms$$

Hence the each ADC channel will start conversion after every 1 ms (after every complete RLT1 cycle that is after 2 underflows).

In the RLT1 interrupt service routine (ISR) the PPG8-11 are started after every 1 ms (i.e. after 2 interrupts). The PPG8 is started at the very first interrupt; PPG9 is started at the third interrupt, so on and so forth. This ensures the phase shift of ¼ (1 ms) of the total PPG pulse width (4 ms). It should be noted that the RLT1 ISR is only used during initial setup of PPGs and it is not required for normal operation.

The below figure describes the timing behavior described above.

Figure 3. Timing Diagram



PPG

All the PPGs (PPG8-11) are fed by 16 MHz of CLKP1 clock with prescaler of 4, hence the clock of 1 MHz. In the RLT1 ISR the PPGs are started as below:

- The PPG8 is started at the first interrupt with period of 4 ms and a duty of 75%.
- The PPG9 is started at the third interrupt with period of 4 ms and a duty of 62.5%.
- The PPG10 is started at the fifth interrupt with period of 4 ms and a duty of 50%.
- The PPG11 is started at the seventh interrupt with period of 4 ms and a duty of 37.5%.

The actual period is the value loaded in period register (`PCSR`) +1 and the actual duty is the value loaded in the duty register (`PDUT`) + 1. Hence for the period of 4 ms and a duty of 75% the value required to be loaded in `PCSR8` is 15999 and the value required to be loaded in `PDUT8` is 11999. Period and duty register of the remaining PPGs are configured considering the same.

ADC and DMA

The ADC is used in STOP mode with 10 bit conversion resolution. Channel 0 to 3 will be converted one after another at every trigger from RLT1 i.e. after every 1 ms. The sampling time is configured as 8 cycles[1] (this is only valid if the supply voltage is between range - 4.5V ≤ AV_{CC} ≤ 5.5V) where as the compare time is configured as 22 cycles[2]. Hence at 16 MHz of CLKP1, the total time taken for each channel conversion would be 1.875 µs. The conversion will go on in cyclic manner (i.e. channel 0-1-2-3-0-1-2-3).

In the RLT1 ISR, there is a wait loop of about 6 µs before starting each of the PPGs. This wait time should be at least equal to the conversion time ADC (1.875 µs, in this case). This makes sure that the PPG output is ALWAYS HIGH when the ADC conversion happens.

The following logic can be applied to determine the status of the lamp:

- If the lamp is operational, then the ADC would also see the voltage equivalent to $AV_{CC} – V_d$ (voltage drop across the lamp).
- If the lamp is non-operational (open), then the ADC would see voltage equivalent to V_{SS} (ground potential).
- If the lamp is non-operational (short), then the ADC would see voltage equivalent to V_{CC} (no voltage drop across the lamp).

In order to accommodate the tolerance in the supply voltage and the voltage drop across the lamp, the ADC equivalent of lamp voltage is not compared with the above mentioned fixed voltages but range of voltages as below:

Sr. No	Condition	Voltage Across the Lamp in Ideal Condition	ADC Equivalent of Compared Voltage Range	
			Minimum	Maximum
1	Lamp Operational	$AV_{CC} – V_d$	`AVCC_Vd_MIN`	`AVCC_Vd_MAX`
2	Lamp Non-operational : Open	V_{SS}	`AVSS_CNT`	`AVSS_MAX`
3	Lamp Non-operational : Short	V_{CC}	`AVCC_MIN`	`AVCC_CNT`

After every conversion the DMA transfers the converted ADC value to `MeasBuf[]` array. After 4 such transfers the ADC ISR would be executed, which re-initializes the DMA. In the main routine the ADC data can be analyzed with the above discussed logic to monitor the lamp. Nonetheless, the logic discussed above is just an example. In an application this logic may not be used as is and may need enhancements.

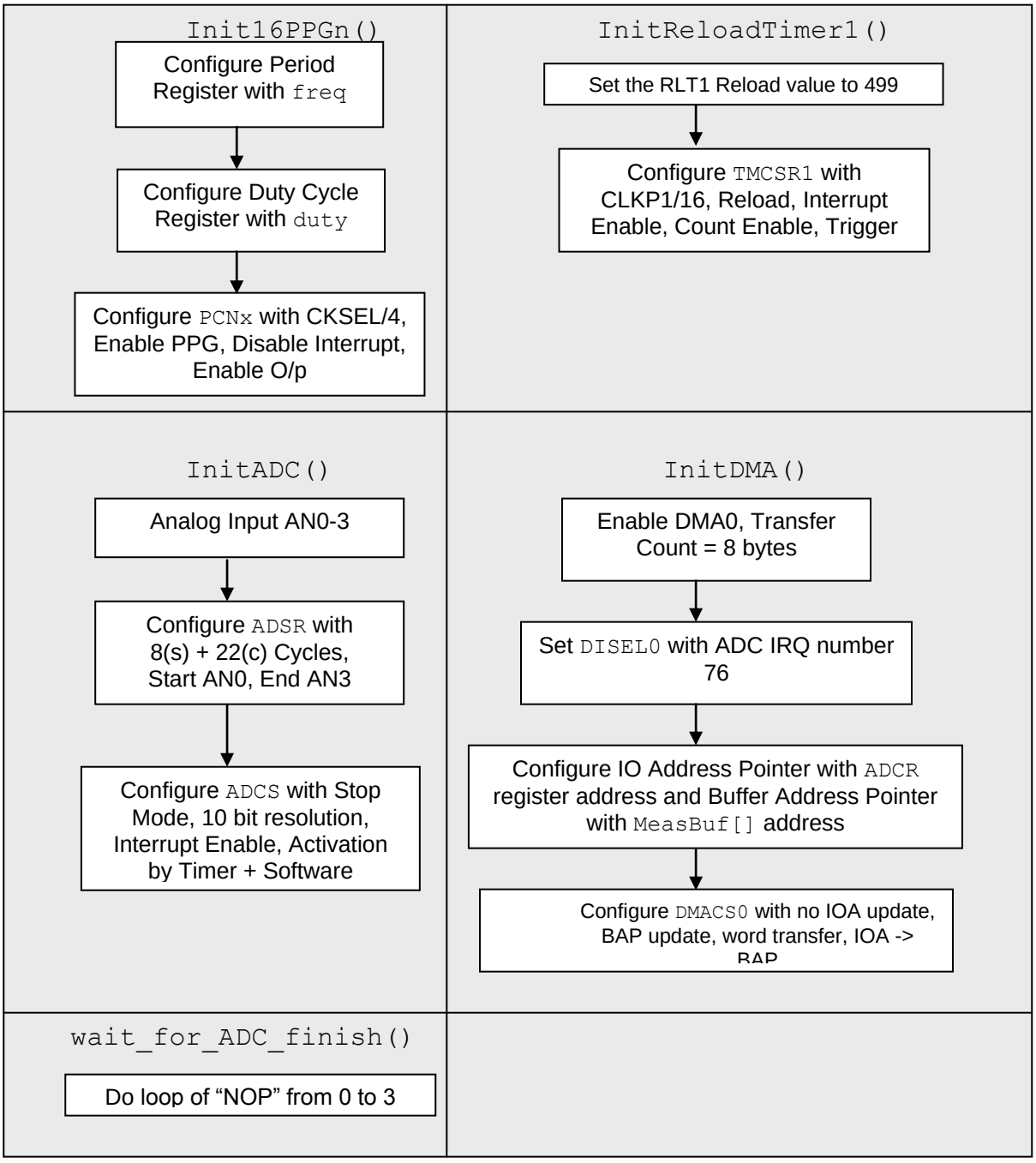
[1] The cycle setting for the Sample Time depends on the driving impedance and the supply voltage AV_{CC}.

[2] The cycle setting for Compare Time depends on the supply voltage AV_{CC}.

Example Code

Initialization Functions

3.0.1 Flowchart



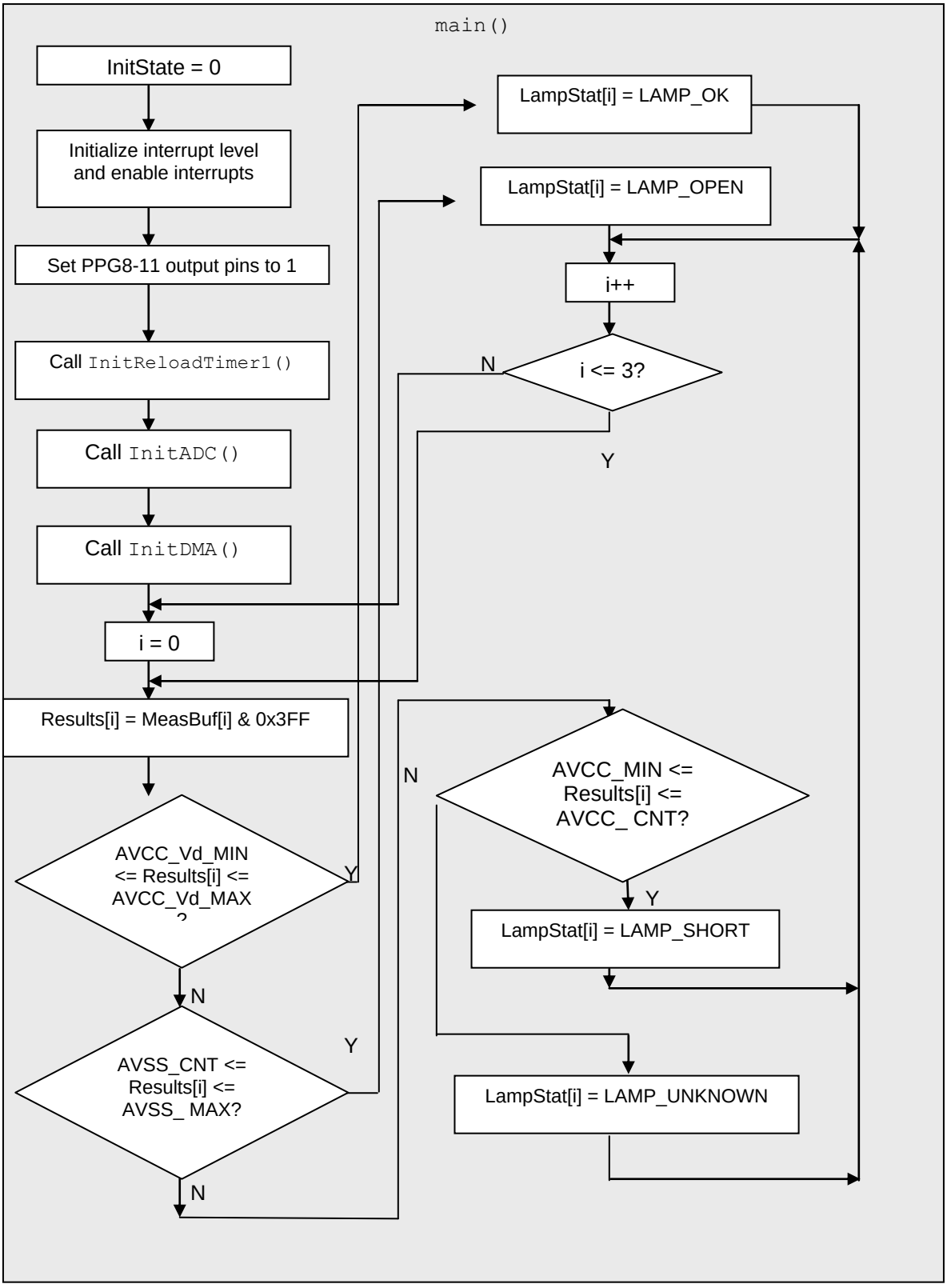
3.0.2 C Code

```
char InitState;           // State Flag for sequential PPG init and ADC init
volatile int  MeasBuf[4];     // ADC Measurement Buffer
int  Results[4];          // Result Buffer: Takes data from (MeasBuf[] & 0x3FF)

unsigned char LampStat[4];        // Status of the Lamp

/*---------------------------------------------------------------------------*/
/*                          Initialize PPGs                                  */
/*---------------------------------------------------------------------------*/

void Init16PPG8 (unsigned int freq, unsigned int duty)
{
  PCSR8 = freq;  // always set cycle value PERIOD 1st
  PDUT8 = duty;  // set duty value DUTY CYCLE
  PCN8 = 0xD402; // clk/4, enable PPG8, disable interrupts, enable output
}

void Init16PPG9 (unsigned int freq, unsigned int duty)
{
  PCSR9 = freq;  // always set cycle value PERIOD 1st
  PDUT9 = duty;  // set duty value DUTY CYCLE
  PCN9 = 0xD402; // clk/4, enable PPG9, disable interrupts, enable output
}

void Init16PPG10 (unsigned int freq, unsigned int duty)
{
  PCSR10 = freq;  // always set cycle value PERIOD 1st
  PDUT10 = duty;  // set duty value DUTY CYCLE
  PCN10 = 0xD402; // clk/4, enable PPG10, disable interrupts, enable output
}

void Init16PPG11 (unsigned int freq, unsigned int duty)
{
  PCSR11 = freq;  // always set cycle value PERIOD 1st
  PDUT11 = duty;  // set duty value DUTY CYCLE
  PCN11 = 0xD402; // clk/4, enable PPG11, disable interrupts, enable output
}

/*---------------------------------------------------------------------------*/
/*                          Initialize RLT1                                  */
/*---------------------------------------------------------------------------*/

void InitReloadTimer1 (void)
{
  TMRLR1 = 499;       // set reload value 499
                      // One Cycle: 500 ticks * 1 µS = 0.5 ms
   TMCSR1 = 0x041B;   // prescaler CLKP1/2^4, reload, interrupt enable, count enable,
                      // trigger
}

/*---------------------------------------------------------------------------*/
/*                          Initialize ADC                                   */
/*---------------------------------------------------------------------------*/

void InitADC (void)
{
  ADER0 = 0x0F;       // ADC Input 0-3

  ADSR = 0x4003;      // 8 cyc. sample, 22 cyc. conversion, Start AN0, End AN3
  ADCSL = 0xC0;       // Stop Mode, 10 Bit
  ADCSH = 0xA8;       // Interrupts, Activation by Timer and Software
}
```

```
/*                          (Re-)Initialize DMA                               */
/*----------------------------------------------------------------------------*/

void InitDMA (void)
{
  DER = 0x0001;   // DMA 0 enable

  DCTH0 = 0;      // 8 Bytes = 4 words
  DCTL0 = 8;
  DISEL0  = 76;         // ADC irq number
  IOAH0 = (unsigned char) &ADCR >> 8;  // I/O Bank 00
  IOAL0 = (unsigned char) &ADCR & 0xFF;
  DMACS0 = 0x18;   // no IOA update, BAP update, word transfer, IOA -> BAP
  BAPH0 = (__far unsigned long) &MeasBuf[0] >> 16;
  BAPM0 = (__far unsigned long) &MeasBuf[0] >> 8;
  BAPL0 = (__far unsigned long) &MeasBuf[0] & 0xFF;
}

/*----------------------------------------------------------------------------*/
/*                             Wait delay                                     */
/*----------------------------------------------------------------------------*/

void wait_for_ADC_finish(void)  // inclusive CALLP and RETP about 6 us
{
  volatile unsigned char i;

  for (i = 0; i < 4; i++)
    __wait_nop();
}
```

Main Function

3.1.1 Flowchart



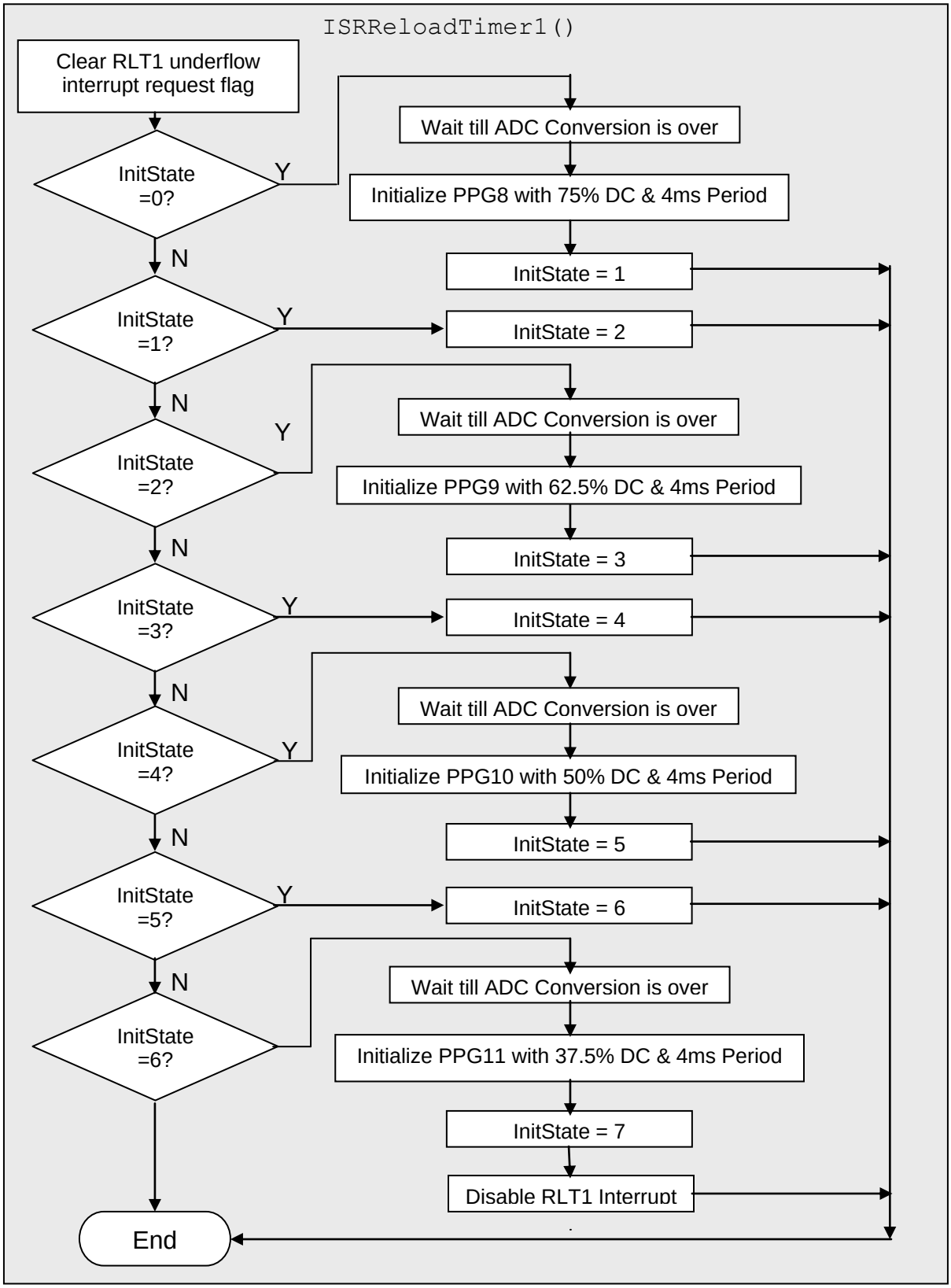
3.1.2 C Code

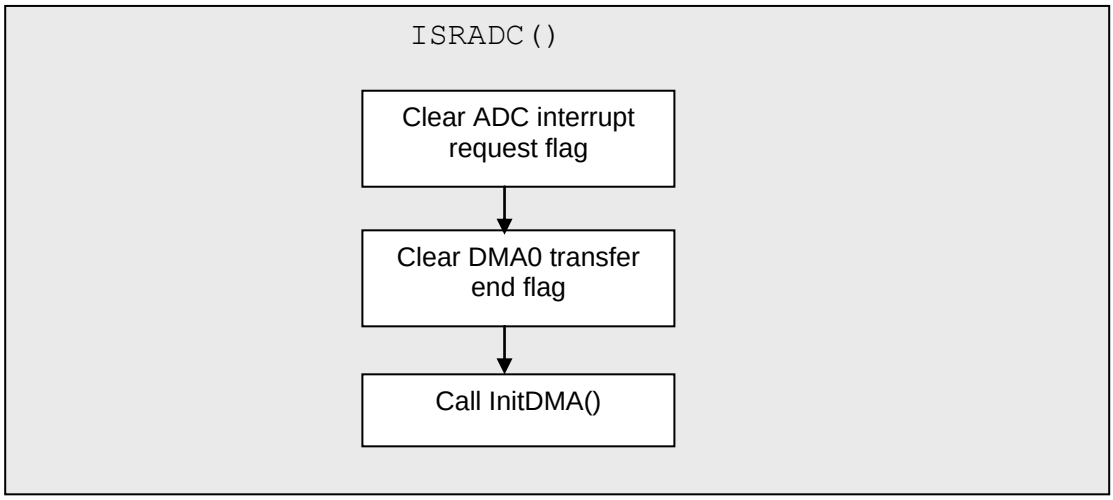
The Following section shows sample code for configurable parameters, which can be edited by the user depending upon the application requirement. It should be noted that the lamp voltage is compared to range of voltage rather than a fixed value so as to accommodate the tolerance in the supply voltage and the voltage drop across the lamp.

```
// Configurable Parameters
#define AVCC      5        // ADC Supply Voltage in volts
#define AVSS      0        // ADC Ground Voltage in volts
#define Vd        0.1      // voltage drop across lamp in volts
#define Margin    2        // Tolerance in voltage in terms of percentage

// Derived Parameters
#define AVCC_CNT  (1024)               // ADC Count equi. to Vcc at 10 bit resolution
#define AVSS_CNT  ((AVSS/AVCC)*1024)   // ADC Count equi. to Vss at 10 bit resolution
#define Vd_CNT        ((Vd/AVCC)*1024) // ADC Count equi. to Vd at 10 bit resolution
#define AVCC_MIN  (AVCC_CNT-((AVCC_CNT*Margin)/100))
#define AVSS_MAX  (AVSS_CNT+((AVSS_CNT*Margin)/100))
#define AVCC_Vd   (AVCC_CNT-Vd_CNT)
#define AVCC_Vd_MAX   (AVCC_Vd+((AVCC_Vd*Margin)/100))
#define AVCC_Vd_MIN   (AVCC_Vd-((AVCC_Vd*Margin)/100))
#define LAMP_DEFAULT  0
#define LAMP_OK   1
#define LAMP_OPEN 2
#define LAMP_SHORT    3
#define LAMP_UNKNOWN  4

char InitState;         // State Flag for sequential PPG init and ADC init
int  MeasBuf[4];        // ADC Measurement Buffer
int  Results[4];        // Result Buffer: Takes data from (MeasBuf[] & 0x3FF)
unsigned char LampStat[4];        // Status of the Lamp

void main(void)
{
  unsigned int i;
  InitState = 0;
  InitIrqLevels();
  __set_il(7);          // allow all levels
  __EI();               // globaly enable interrupts

  PDR09 = 0x0F;         // Set PPG11-8 output pins to "1" for 1st ADC measurement
  DDR09 = 0x0F;         // -> output "1"
  InitReloadTimer1();   // Start RLT1/ADC and PPG sequence
  InitADC();           // Init ADC
  InitDMA();           // Init ADC-DMA transfer

  while(1)
  {
    for (i=0; i<=3 ; i++)
    {
        // Mask results from buffer and update array
            Results[i] = MeasBuf[i] & 0x3FF;
        //determine the lamp status
        if (Results [i] >= AVCC_Vd_MIN && Results [i] <= AVCC_Vd_MAX)
            LampStat[i] = LAMP_OK;
        else if (Results [i] >= AVSS_CNT && Results [i] <= AVSS_MAX)
            LampStat[i] = LAMP_OPEN;
        else if (Results [i] >= AVCC_MIN && Results [i] <= AVCC_CNT)
            LampStat[i] = LAMP_SHORT;
        else
        LampStat[i] = LAMP_UNKNOWN;
     }
    }
}
```

Interrupt Service Routines

3.2.1 Flowchart




ISRADC()
Clear ADC interrupt request flag
Clear DMA0 transfer end flag
Call InitDMA()

3.2.2 C Code

```
/*                          Reload 1 Timer ISR                                */
/*----------------------------------------------------------------------------*/

__interrupt void ISRReloadTimer1(void)
{
  TMCSR1_UF = 0;                // reset underflow interrupt request flag

  // Sequential Initialization of PPG11 - PPG8
  switch (InitState)
  {
    case 0:
      wait_for_ADC_finish();    // wait for about 6us
      Init16PPG8(15999, 11999);   // 1 ms + 3 ms = 4 ms
      InitState = 1;
      break;

    case 1:
      InitState = 2;
      break;

    case 2:
      wait_for_ADC_finish();    // wait for about 6us
      Init16PPG9(15999, 9999);   // 1.5 ms + 2.5 ms = 4 ms
      InitState = 3;
      break;

    case 3:
      InitState = 4;
      break;

    case 4:
      wait_for_ADC_finish();    // wait for about 6us
      Init16PPG10(15999, 7999);  // 2 ms + 2 ms = 4 ms
      InitState = 5;
      break;

    case 5:
      InitState = 6;
      break;

    case 6:
      wait_for_ADC_finish();    // wait for about 6us
      Init16PPG11(15999, 5999);  // 2.5 ms + 1.5 ms = 4 ms
      InitState = 7;
      TMCSR1 = 0x0412;          // disable RLT1 interrupt (ADC/PPG configuration done)
      break;

    case default:
    break;
   }
}

/*----------------------------------------------------------------------------*/
/*                                ADC ISR                                     */
/*----------------------------------------------------------------------------*/

__interrupt void ISRADC(void)
{
  ADCSH = 0xA8; // clear INT
  DSR = 0x0000; // Clear DMA request

  InitDMA();   // Reinitialize DMA
}
```

Interrupt Vectoror

3.3.1 Code

```
ICR = (52 << 8) | 2;        /* Priority Level 2 for RLT1 of MB9634x Series */
ICR = (76 << 8) | 2;        /* Priority Level 2 for ADC of MB9634x Series */

    . . .

/* ISR prototype */
__interrupt void ISRReloadTimer1(void);
__interrupt void ISRADC (void);

    . . .
#pragma intvect ISRReloadTimer1   52          /* RLT1 of MB9634x Series */
#pragma intvect ISR_ADC           76          /* ADC of MB9634x Series */
    . . .
```

Additional Information

Information about Cypress Microcontrollers can be found on the following Internet page:

http://www.cypress.com/cypress-microcontrollers

The software example related to this application note is:

96340_ppg_rlt_adc_dma

It can be found on the following Internet page:

http://www.cypress.com/cypress-mcu-product-softwareexamples

Document History

Document Title: AN204825 - F²MC-16FX Family, Lamp Control and Monitor with PPG and ADC

Document Number:002-04825

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	NOFL	03/28/2007	Initial release
			07/24/2007	Updated with re-review findings from PHu
*A	5097379	NOFL	05/11/2016	Migrated Spansion Application Note MCU-AN-300237-E-V11 to Cypress format
*B	5865605	AESATP12	08/30/2017	Updated logo and copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709