```
/*******************************************************************************
*  Project Name: Eg3_Mem_DMA_DAC
*  Device Tested: CY8C3866AXI, CY8C5868AXI
*  Software Version: PSoC Creator 3.0 SP1
*  Complier tested: Keil(C51) and GCC
*  Related Hardware: CY8CKIT-001, CY8CKIT-030 and CY8CKIT-050
********************************************************************************
********************************************************************************
*
* Copyright (2014)), Cypress Semiconductor Corporation. All Rights Reserved.
********************************************************************************
*
* This software is owned by Cypress Semiconductor Corporation (Cypress)
* and is protected by and subject to worldwide patent protection (United
* States and foreign), United States copyright laws and international treaty
* provisions. Cypress hereby grants to licensee a personal, non-exclusive,
* non-transferable license to copy, use, modify, create derivative works of,
* and compile the Cypress Source Code and derivative works for the sole
* purpose of creating custom software in support of licensee product to be
* used only in conjunction with a Cypress integrated circuit as specified in
* the applicable agreement. Any reproduction, modification, translation,
* compilation, or representation of this software except as specified above
* is prohibited without the express written permission of Cypress.
*
* Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH
* REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
* Cypress reserves the right to make changes without further notice to the
* materials described herein. Cypress does not assume any liability arising
out
* of the application or use of any product or circuit described herein.
Cypress
* does not authorize its products for use as critical components in life
-support
* systems where a malfunction or failure may reasonably be expected to result
in
* significant injury to the user. The inclusion of Cypress' product in a life-
* support systems application implies that the manufacturer assumes all risk
of
* such use and in doing so indemnifies Cypress against all charges.
*
* Use of this Software may be limited by and subject to the applicable Cypress
* software license agreement.
********************************************************************************
*/

/
********************************************************************************
********************************
                    PROJECT DESCRIPTION
********************************************************************************
********************************
```

```
* Sine Lookup of length 128 is created in a flash and these values are updated to a DAC
* at regular intervals, using DMA, in order to generate a sine wave.
* The update rate and the number of points in the sine look-up table determine the frequency
* of the output sine wave.
* The values from the look-up table are updated into the DAC using a DMA.
* The DMA is set to update values on a hardware trigger.
* The hardware trigger is given by clock component.
* The output frequency of the sine wave equals update rate/number of points in the sine look-up table.
* The example project generates a 7.8125 KHz sine wave with 128 points in the sine look-up table with an
* update rate of  1MHz.
* Oscilloscope is connected to P0[0] to see the DAC output
********************************************************************************
******************************** */

#include <device.h>
#include <cytypes.h>

#define TABLE_LENGTH 128
#define DMA_BYTES_PER_BURST 1
#define DMA_REQUEST_PER_BURST 1
int32 Shift;
int32 degree;
/* This table stores the 128 points in Flash for smoother sine wave generation */
CYCODE const uint8 sineTable[TABLE_LENGTH*2] =
{
    128, 134, 140, 147, 153, 159, 165, 171,
    177, 183, 188, 194, 199, 204, 209, 214,
    218, 223, 227, 231, 234, 238, 241, 244,
    246, 248, 250, 252, 253, 254, 255, 255,
    255, 255, 255, 254, 253, 252, 250, 248,
    246, 244, 241, 238, 234, 231, 227, 223,
    218, 214, 209, 204, 199, 194, 188, 183,
    177, 171, 165, 159, 153, 147, 140, 134,
    128, 122, 115, 109, 103,  97,  91,  85,
     79,  73,  68,  62,  57,  52,  47,  42,
     37,  33,  29,  25,  22,  18,  15,  12,
     10,   7,   6,   4,   2,   1,   1,   0,
      0,   0,   1,   1,   2,   4,   6,   7,
     10,  12,  15,  18,  22,  25,  29,  33,
     37,  42,  47,  52,  57,  62,  68,  73,
     79,  85,  91,  97, 103, 109, 115, 122,
    128, 134, 140, 147, 153, 159, 165, 171,
    177, 183, 188, 194, 199, 204, 209, 214,
    218, 223, 227, 231, 234, 238, 241, 244,
    246, 248, 250, 252, 253, 254, 255, 255,
    255, 255, 255, 254, 253, 252, 250, 248,
    246, 244, 241, 238, 234, 231, 227, 223,
```

```
    218, 214, 209, 204, 199, 194, 188, 183,
    177, 171, 165, 159, 153, 147, 140, 134,
    128, 122, 115, 109, 103,  97,  91,  85,
     79,  73,  68,  62,  57,  52,  47,  42,
     37,  33,  29,  25,  22,  18,  15,  12,
     10,   7,   6,   4,   2,   1,   1,   0,
      0,   0,   1,   1,   2,   4,   6,   7,
     10,  12,  15,  18,  22,  25,  29,  33,
     37,  42,  47,  52,  57,  62,  68,  73,
     79,  85,  91,  97, 103, 109, 115, 122
};

/* Variable declarations for DMA .
* These variables are defined as global variables to avoid "may be used before being set" warning
* issued by the PSoC 5 compilers  MDK/RVDS.In this case these variables are automatically initialized to zero */
uint8 DMA_1_Chan;               /* The DMA Channel */
uint8 DMA_2_Chan;

uint8 DMA_1_TD[1];            /* The DMA Transaction Descriptor (TD) */
uint8 DMA_2_TD[1];

void main()
{
    ADC_Start();
    ADC_StartConvert();
    LCD_Start();
    VDAC8_Start();
    VDAC8_Shift_Start();

    CYGlobalIntEnable;

    #if (defined(__C51__))  /* Source base address when PSoC3 is used */
        #define DMA_SRC_BASE (CYDEV_FLS_BASE)
    #else                   /* Source base address when PSoC5 is used */
        #define DMA_SRC_BASE (&sineTable[0])
    #endif

    #define DMA_DST_BASE (CYDEV_PERIPH_BASE)  /* Destination base address */

    /* Step1 : DmaInitialize - Initialize the DMA channel
     * Bytes per burst = 1, (8 bit data transferred to VDAC one at a time)
     * Request per burst = 1 (this will cause transfer of the bytes only with every new request)
     * High byte of source address = Upper 16 bits of Flash Base address for PSoC 3,
                                     = HI16(&sineTable) for PSoC 5
     * High byte of destination address =  Upper 16 bits of peripheral base address */
    DMA_1_Chan = DMA_1_DmaInitialize(DMA_BYTES_PER_BURST, DMA_REQUEST_PER_BURST, HI16(DMA_SRC_BASE), HI16(DMA_DST_BASE) );
```

```
    DMA_2_Chan = DMA_2_DmaInitialize(DMA_BYTES_PER_BURST, DMA_REQUEST_PER_BURST, HI16(DMA_SRC_BASE), HI16(DMA_DST_BASE) );


    /* Step2 :CyDmaTdAllocate - Allocate TD */
    DMA_1_TD[0] = CyDmaTdAllocate();
    DMA_2_TD[0] = CyDmaTdAllocate();

    /* Step3 :CyDmaTdSetConfiguration - Configures the TD:
     * tdHandle = DMA_TD[0] - TD handle previously returned by CyDmaTdAlloc()
     * Transfer count = table_length (number of bytes to transfer for a sine wave)
     * Next Td = DMA_TD[0] ; loop back to the same TD to generate a continous sien wave
     * Configuration = The source address is incremented after every burst transfer
     */
    CyDmaTdSetConfiguration(DMA_1_TD[0], 128, DMA_1_TD[0], TD_INC_SRC_ADR);
    CyDmaTdSetConfiguration(DMA_2_TD[0], 128, DMA_2_TD[0], TD_INC_SRC_ADR);


    /* Step 4 : CyDmaTdSetAddress - Configure the lower 16 bit source and destination addresses
     * Source address = Lower 16 bits of sineTable array
     * Destination address = Lower 16 bits of VDAC8_Data_PTR register */
    CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)sineTable), LO16((uint32)VDAC8_Data_PTR) );
    CyDmaTdSetAddress(DMA_2_TD[0], LO16((uint32)sineTable), LO16((uint32)VDAC8_Shift_Data_PTR) );

    /* Step 5: Map the TD to the DMA Channel */
    CyDmaChSetInitialTd(DMA_1_Chan, DMA_1_TD[0]);
    CyDmaChSetInitialTd(DMA_2_Chan, DMA_2_TD[0]);

    /* Step 6: Enable the DMA channel */
    CyDmaChEnable(DMA_1_Chan, 1);
    CyDmaChEnable(DMA_2_Chan, 1);

    for(;;)
    {
        /* Your code here. */
         if(ADC_IsEndConversion(ADC_RETURN_STATUS))
        {
        Shift = ADC_GetResult32();

        if(Shift<0x0000){
                Shift = 0x0000;
        }
        if(Shift > 0xFDD1){
                Shift = 0xFFFF;
        }
        Shift  = Shift / 511;
```

```
        // if(Shift<0xf)
          //   Shift = 0xf;
         LCD_Position(0u,0u);
         degree = (Shift*360)/128;
         if(degree <=0x2A)
            degree=0;
         LCD_PrintString("Degree Shifted:");
         LCD_Position(1u,0u);
         LCD_PrintInt16(degree);
         CyDmaTdSetAddress(DMA_1_TD[0], LO16((uint32)&sineTable[Shift]), LO16((uint32)VDAC8_Data_PTR) );
    }
    }
}
```