AN10917

Memory to DAC data transfers using the LPC1700's DMA

Rev. 01 — 8 March 2010 Application note

Document information

Info	Content
Keywords	LPC1700, DMA, DAC, ADC, Timer 0, Memory-to-Peripheral
Abstract	This application note covers application details on performing memory-to-DAC DMA transfers. The application examples generate sine and triangular waveforms onto the DAC.

Revision history

Rev	Date	Description
01	20100308	Initial version.

Contact information

For additional information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

This application note serves as a quick introduction to the LPC1700's memory-to-DAC DMA transfers functionality. It will provide a simple description on how the DMA can be used to generate an output onto the DAC without utilizing the CPU.

A detailed feature description of the peripherals listed below can be found in the LPC1700 user manual.

1.1 DMA

Data transfers can require a lot of CPU time performing *load* and *store* instructions. When using a Direct Memory Access (DMA) controller, the CPU is freed up to execute other instructions.

The DMA controller in the LPC1700 supports memory-to-memory, memory-to-peripheral, peripheral-to-peripheral, and peripheral-to-memory transfers. With the use of linked lists, it also supports scatter or gather modes. These modes enable data transfers from non-contiguous areas of memory. Since the clock is functioning during sleep mode, it can also perform DMA transfers to/from AHB SRAM blocks.

1.2 Timer 0

The LPC1700 features four configurable 32-bit timer/counters.

1.3 ADC

An Analog-to-Digital Converter (ADC) captures an external voltage and represents it as a digital value. The LPC1700 contains an 8-channel multiplexed 12-bit successive approximation ADC.

1.4 DAC

The Digital-to-Analog Converter (DAC) creates an output voltage from a corresponding digital input value. The LPC1700 features a double buffered 10-bit DAC that can generate repetitive DMA requests using an internal countdown counter.

2. Application demo

2.1 Requirements

The MCB1700 can be programmed by using:

- Keil's uVision4 with the ULINK JTAG module

-OR-

- IARs Embedded Workbench 5.4 with the J-Link module

-OR-

- FlashMagic's free ISP programming tool with a RS-232 serial cable.

2.1.1 Software

Either Keil's uVision4 or IAR's Workbench development tools can be used to compile the application demos. Throughout this application note, only screenshots from uVision are shown.

The evaluation versions of Keil or IAR will work for the demo. If you are using uVision version 4.00 or earlier or Workbench 5.4 or earlier, please see the Known issues section.

Optional: FlashMagic's ISP utility can be used to flash the LPC1700 without using the ULINK JTAG module. It is available for free at:
http://www.nxp.com/redirect/flashmagictool.com/

2.1.2 Hardware

Keil MCB1700 Development Board

Keil ULINK JTAG module -OR- Optional RS-232 Serial cable

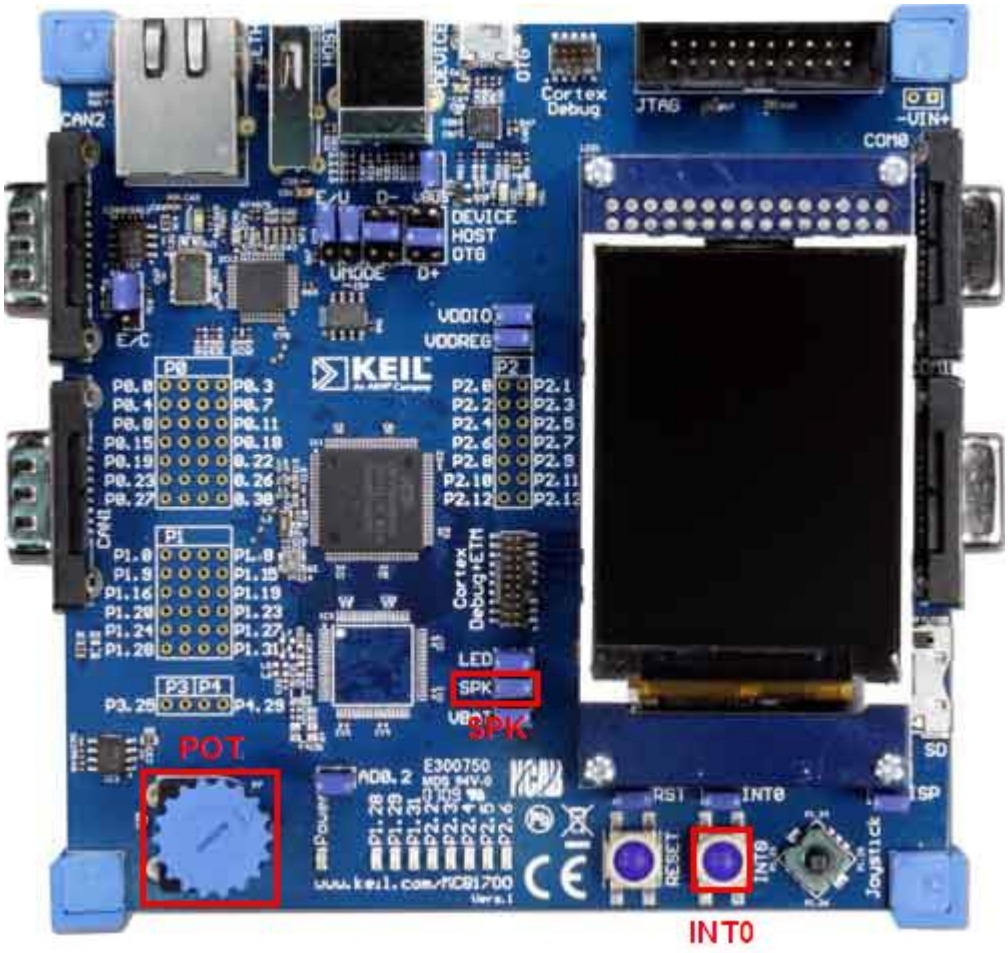


Fig 1. Keil MCB1700 featuring NXP's LPC1768

2.2 Objective

The objective of this application demo is to demonstrate how the DMA performs a memory-to-DAC (peripheral) data transfer. By using the DMA to perform the transfers directly from memory to the DAC frees up the CPU to perform other tasks (or no tasks).

The application note briefly covers three sample projects. All of the three sample projects utilize the DMA. These three application examples produce different output waveforms on the DAC's output pin (P0.26).

The first two examples do not require the CPU for the memory-to-DAC data transfers. To show this we will let the CPU simply toggle two LEDs indefinitely until "INT0" button is pressed. Once "INT0" is pressed, it will turn off the LEDs and put the LPC1700 into sleep mode. During the LPC1700's sleep mode the DMA will continue with its data transfers.

The last (3rd) example utilizes Timer 0 to trigger the memory-to-DAC DMA transfers. Unlike the previous examples, this project uses some CPU time to generate a delay between the actual transfers. The length of this delay is adjusted by using a potentiometer connected to the ADC (AD0.2) on the LPC1700. As an additional indicator, the LED connected onto P1.28 is toggled with the use of Timer 0's match register 0 (MAT0.0).

A summary of the projects with their descriptions are shown in Table 1.

Table 1. Project Summary

Project Example	Project Description
Sine wave	Outputs a sine wave onto the DAC using the DMA. CPU toggles P1.28 and P1.29. INT0 puts the LPC1700 into sleep mode. Once started, will loop indefinitely.
Triangular wave	Outputs a triangular wave onto the DAC using the DMA. CPU toggles P1.28 and P1.29. INT0 puts the LPC1700 into sleep mode. Once started, will loop indefinitely.
Timer 0 enabled Triangular wave	Outputs a triangular wave onto the DAC using the DMA. Timer 0 initiates the memory-to-DAC DMA data transfers. DMA needs to be reconfigured after each complete (32 word) DMA transfer. Match register MAT0.0 toggles P1.28 at each match. ADC input AD0.2 is used to determine the delay time between transfers.

It is encouraged to observe the output of the DAC on an oscilloscope. The "SPK" jumper on the MCB1700 (Fig 1) can be removed so that the DAC isn't connected to the speaker's amplifier.

2.3 Project design

2.3.1 Data location

To reduce power consumption, the DMA controller on the LPC1700 supports DMA transfers while the CPU is in sleep mode. In sleep mode, the main SRAM block is powered down to conserve power. The remaining (up to 32 kB) AHB SRAM remains powered during sleep mode.

It is crucial that any components needed by the DMA are located in the AHB SRAM in order for the DMA to function properly while in sleep mode.

This applies to the source/destination DMA transfer memory locations and to any linked lists that may be used in the DMA's scatter feature.

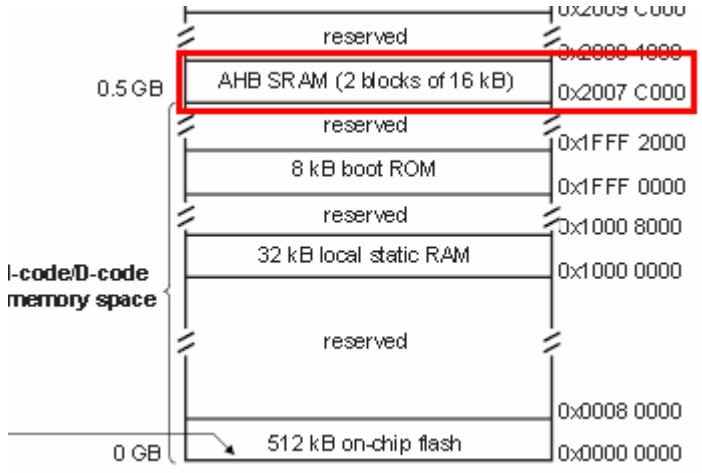


Fig 2. LPC1700 system memory map (from the LPC1700 user manual)

2.3.2 Data preparation

In order to output a sine wave (Fig 26) or a triangular wave (Fig 27) using the DMA, we need to generate the proper data content in memory. This content is then simply transferred to the DAC using the DMA. The differences in these two waveforms only differ in their memory content. The rest of the data preparation remains the same.

Fig 3 shows how the sine wave data is calculated and formatted for the DAC. All of the data needs to be increased (by 512) and shifted (left by 6) so that we have only positive data values that fit in the DACR register (see Fig 6).

The sine wave data block consists of 64 words (256 bytes) formatted for the DAC.

```
for(i=0; i < WAVE_SAMPLE_NUM; i++)
    sinusoide[i] = 512*sin(2*PI*i/WAVE_SAMPLE_NUM);

for(i=0; i < WAVE_SAMPLE_NUM; i++)
    sinusoide[i] = ((sinusoide[i] + 512) << 6) // DACR bit 6-15, VALUE
                 | 1 << 16;                    // DACR bit 16, BIAS = 1
```

Fig 3. Sine wave

This code portion fills in a block of memory with the values that are generated by the sine function. Afterwards, each element is:

1. Increased by 512 so that the peak-to-peak values are all within the DAC's 10-bit resolution (0 to 1023).
2. Shifted to the left by 6 with the BIAS enabled due to the DACR specs.

The Triangular pattern (Fig 4) consists of 32 words (128 bytes) and is already formatted for the DAC so no further manipulation in required.

```
for(i=0;i!=16;i++) triangle[i] = (((i+1)<<6) - 1)<<6;
for(i=0;i!=16;i++) triangle[16+i] = (((16-i)<<6) - 64)<<6;
```

Fig 4. Triangular wave data block

2.3.2.1 DAC's data format

The DMA will perform 32-bit wide data transfers; therefore, it is important to have the data preformatted so that it can be used by the DAC.

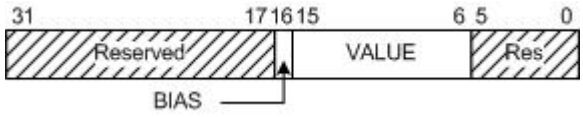


Fig 5. DACR

The VALUE field contains the value that the DAC will output.

4.1 D/A Converter Register (DACR - 0x4008 C000)

This read/write register includes the digital value to be converted to analog, and a bit that trades off performance vs. power. Bits 5:0 are reserved for future, higher-resolution D/A converters.

Table 541: D/A Converter Register (DACR - address 0x4008 C000) bit description

Bit	Symbol	Value	Description	Reset Value
5:0	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:6	VALUE		After the selected settling time after this field is written with a new VALUE, the voltage on the AOUT pin (with respect to V_{SSA}) is VALUE × ((V_{REFP} - V_{REFN})/1024) + V_{REFN}.	0
16	BIAS[1]	0	The settling time of the DAC is 1 µs max, and the maximum current is 700 µA. This allows a maximum update rate of 1 MHz.	0
		1	The settling time of the DAC is 2.5 µs and the maximum current is 350 µA. This allows a maximum update rate of 400 kHz.	
31:17	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

[1] The settling times noted in the description of the BIAS bit are valid for a capacitance load on the AOUT pin not exceeding 100 pF. A load impedance value greater than that value will cause settling time longer than the specified time. One or more graph(s) of load impedance vs. settling time will be included in the final data sheet.

Fig 6. D/A Converter Register (DACR)

2.3.3 DMA configuration

The purpose of having the DMA is to offload the CPU from intensive overhead to perform the bulk of data transfers. The following sections give a brief description of the DMA's components.

2.3.3.1 Linked lists

The DMA on the LPC1700 has the ability to use a linked list structure so that non-contiguous blocks of memory can be transferred. Each individual DMA channel can support an independently configured data transfer.

Each channel contains a few set of registers that pertain to its configuration/operation. They include the following registers:

a. **Channel Source address**

b. **Channel Destination address**

c. **Channel Linked List Item**

d. **Channel Control**

e. Channel Configure

From these five registers, the first four (bolded) can be combined into a Linked List node/structure. If the Linked List Item register is 0, the DMA will disable the channel after all the data has been transferred. If the LLI register is not cleared, the DMA will load new source, destination, LLI, and control values from the address pointed by the LLI.

```
struct {
  uint32_t  source;      // start of source area
  uint32_t  destination; // start of destination area
  uint32_t  next;        // address of next strLLI in chain
  uint32_t  control;     // DMACCxControl register
} LLIO;
```

Fig 7. Linked List Node structure

It is important to keep the link list node structure as shown in Fig 7.

Fig 8 shows a simple illustration on how the DMA channel can be configured with or without Linked Lists.

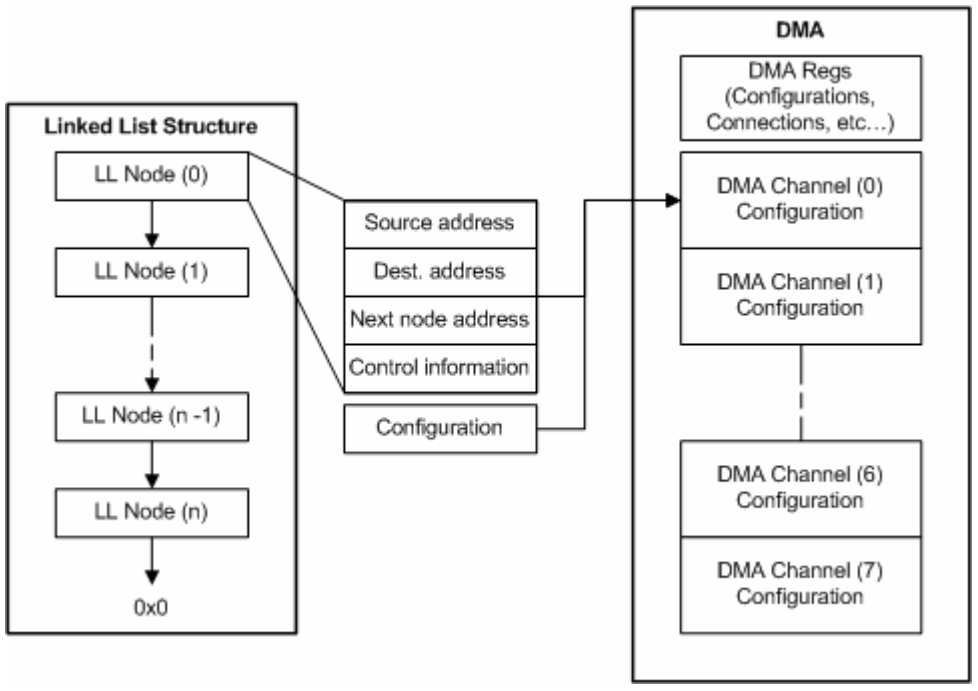
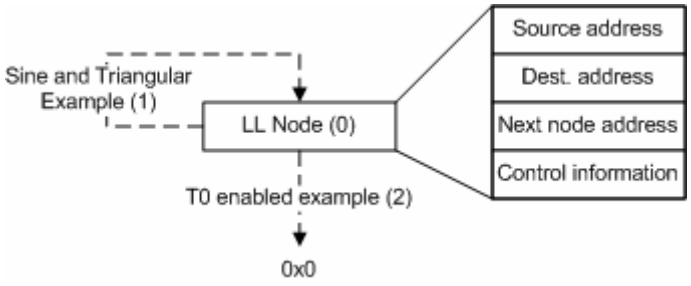


Fig 8. DMA Configuration structure

In the sine and triangular wave examples, the memory block that contains the data values are contiguous (sinusoid[ ] and triangle[ ]). The linked list structure is utilized to have the DMA continuously transfer the same block of data to the DAC.



(1) The Sine and Triangular examples will continue indefinitely by having a linked list node pointing to itself

(2) The T0 enabled example will perform just one transfer

Fig 9. Linked list configuration for the sine, triangular, and T0 enabled example projects

The Timer 0 enabled example project only performs one complete data transfer and then disables its channel.

```
LLI0.source      = (uint32_t) &sinusoide[0];
LLI0.destination = (uint32_t) &(LPC_DAC->DACR);
LLI0.next        = (uint32_t) &LLI0;
LLI0.control     = 1<<26 | 2<<21 | 2<<18 | WAVE_SAMPLE_NUM;
```

Fig 10. Linked list node configuration for the sine wave example

Sine wave example linked list node configuration.

Note that with this configuration, the DMA will perform the same data transfer repetitively.

2.3.3.2 DMA triggering

To start a memory-to-memory transfer a DMA channel is configured and then enabled. However, when a DMA channel is transferring to or from (or even both) a peripheral, DMA requests from the peripheral(s) are needed. These requests are essentially generated when the peripheral(s) is(are) ready.

Transfer direction	Request generator	Flow controller
Memory-to-peripheral	Peripheral	DMA Controller
Peripheral-to-memory	Peripheral	DMA Controller
Memory-to-memory	DMA Controller	DMA Controller
Source peripheral to destination peripheral	Source peripheral and destination peripheral	DMA Controller

Fig 11. DMA request signals

Available sources

Only the sources shown in Fig 11 are able to generate DMA requests on the LPC1700.

4.2.3 DMA request connections

The connection of the GPDMA to the supported peripheral devices depends on the DMA functions implemented in those peripherals. Table 31–545 shows the DMA Request numbers used by the supported peripherals. UART and timer DMA requests on channels 8 through 15 are chosen via the DMAREQSEL register, see Section 31–5.15.

Table 545. DMA Connections

Peripheral Function	DMA Single Request Input (DMACSREQ)	DMA Burst Request Input (DMACBREQ)	DMA Request Signal
SSP0 Tx	0	0	Dedicated DMA requests
SSP0 Rx	1	1	Dedicated DMA requests
SSP1 Tx	2	2	Dedicated DMA requests
SSP1 Rx	3	3	Dedicated DMA requests
ADC	4	4	ADC interrupt request [1]
I²S channel 0	-	5	Dedicated DMA request
I²S channel 1	-	6	Dedicated DMA request
DAC	-	7	Dedicated DMA request
UART0 Tx / MAT0.0	-	8	Dedicated DMA requests
UART0 Rx / MAT0.1	-	9	Dedicated DMA requests
UART1 Tx / MAT1.0	-	10	Dedicated DMA requests
UART1 Rx / MAT1.1	-	11	Dedicated DMA requests
UART2 Tx / MAT2.0	-	12	Dedicated DMA requests
UART2 Rx / MAT2.1	-	13	Dedicated DMA requests
UART3 Tx / MAT3.0	-	14	Dedicated DMA requests
UART3 Rx / MAT3.1	-	15	Dedicated DMA requests

[1] Generates an interrupt and/or DMA request depending on software setup.

Fig 12. DMA request connections available on the LPC1700

Note that MAT0.0 is multiplexed with UART0 Tx in the DMA Request Select register RMAReqSel[0].

In the sine wave and triangular wave example projects, we opt to use the DAC's countdown counter to generate the DMA request. In the Timer 0 enabled example project match register MAT0.0 is used instead.

Because MAT0.0 is multiplexed with UART0 Tx, we need to make sure to "route" this source to the DMA.

Table 561. DMA Request Select register (DMAReqSel - 0x400F C1C4)

Bit	Name	Function
0	DMASEL08	Selects the DMA request for GPDMA input 8: 0 - UART0 TX is selected. 1 - Timer 0 match 0 is selected.

a. UM's description of DMAREQSEL

```
/* Select MAT0.0 instead of UART0 Tx */
LPC_SC->DMAREQSEL = 0x01;
```

b. Setting the DMAREQSEL register

Fig 13. Selecting MAT0.0 as an DMA request source

Specifying the sources

To configure the DMA channel so that a peripheral triggers a transfer, we need to specify that particular peripheral in the channel's configuration register.

SrcPeripheral	Source peripheral. This value selects the DMA source request peripheral. This field is ignored if the source of the transfer is from memory. See Table 31–545 for peripheral identification.
DestPeripheral	Destination peripheral. This value selects the DMA destination request peripheral. This field is ignored if the destination of the transfer is to memory. See Table 31–545 for peripheral identification.

Fig 14. Description of "SrcPeripheral" and "DestPeripheral" from the LPC1700's user manual

In the sine wave and triangular example projects we specify the DAC as the peripheral to trigger the DMA request. Since this source is not multiplexed with any other request line, we can disregard the DMAREQSEL register in this project.

```
LPC_GPDMACH0->DMACCConfig    =  1          // channel enabled (0)
                             | (0 << 1)    // source peripheral (1 - 5) = none
                             | (7 << 6)    // destination peripheral (6 - 10) = DAC
                             | (1 << 11)   // flow control (11 - 13) = mem to per
                             | (0 << 14)   // (14) = mask out error interrupt
                             | (0 << 15)   // (15) = mask out terminal count interrupt
                             | (0 << 16)   // (16) = no locked transfers
                             | (0 << 18);  // (27) = no HALT
```

Fig 15. Selecting the DAC as a triggering source

In the Timer 0 enabled example project we are now using MAT0.0 as the DMA request source. Note that we need to configure DMAREQSEL in this project because it is multiplexed with UART0 Tx.

```
LPC_GPDMACH0->DMACCConfig    =  1          // channel enabled (0)
                             | (0 << 1)    // source peripheral (1 - 5) = none
                             | (8 << 6)    // destination request peripheral (6 - 10) = MAT0.0
                             | (1 << 11)   // flow control (11 - 13) = mem to per
                             | (0 << 14)   // (14) = mask out error interrupt
                             | (0 << 15)   // (15) = mask out terminal count interrupt
                             | (0 << 16)   // (16) = no locked transfers
                             | (0 << 18);  // (27) = no HALT
```

Fig 16. Selecting match register MAT0.0 as the triggering source

2.3.3.3 Channel control

For all three sample projects we know that the sine wave or triangular wave data is located in a contiguous block of memory. The sine wave data block contains 64 words and the triangle data block contains 32 words.

For each time that a DMA request is received, the DMA will transfer a 1-word burst from the source to the destination. For it to complete the entire transfer, it needs to complete 64 transfers for the sine wave data and 32 transfers for the triangular wave data.

```
LPC_GPDMACH0->DMACCSrcAddr  = (uint32_t) &sinusoide[0];
LPC_GPDMACH0->DMACCDestAddr = (uint32_t) &(LPC_DAC->DACR);
LPC_GPDMACH0->DMACCLLI      = (uint32_t) &LLI0; // linked lists for ch0
LPC_GPDMACH0->DMACCControl  = WAVE_SAMPLE_NUM    // transfer size (0 - 11) = 64
                        | (0 << 12)              // source burst size (12 - 14) = 1
                        | (0 << 15)              // destination burst size (15 - 17) = 1
                        | (2 << 18)              // source width (18 - 20) = 32 bit
                        | (2 << 21)              // destination width (21 - 23) = 32 bit
                        | (0 << 24)              // source AHB select (24) = AHB 0
                        | (0 << 25)              // destination AHB select (25) = AHB 0
                        | (1 << 26)              // source increment (26) = increment
                        | (0 << 27)              // destination increment (27) = no increment
                        | (0 << 28)              // mode select (28) = access in user mode
                        | (0 << 29)              // (29) = access not bufferable
                        | (0 << 30)              // (30) = access not cacheable
                        | (0 << 31);             // terminal count interrupt disabled
```

Fig 17. DMA channel control for the sine wave sample project

Note that “transfer size” does not necessarily correspond to the number of bytes.

For the sine wave and triangular wave project we want to transfer the entire block of memory to the DAC, while each 1-word burst request is generated from the DAC. Once all of the 32 words have been transferred, the DMA will continue on to the next linked list node. In these two sample projects the DMA channel’s LLI register is always pointing to the same node (itself). Because in this situation the LLI node will never be equal to 0, the DMA will never disable the channel.

However, if the channel’s LLIO register had been set to 0, this would have meant that the DMA will transfer the entire memory block (32 transfers in this case) and then disable the channel. Once the channel has been disabled, it will then have to be reconfigured again before it can be enabled.

This is exactly the case with the Timer 0 enabled example project.

```
LPC_GPDMACH0->DMACCSrcAddr  = (uint32_t) &triangle[0];
LPC_GPDMACH0->DMACCDestAddr = (uint32_t) &(LPC_DAC->DACR);
LPC_GPDMACH0->DMACCLLI      = 0;            // linked lists for ch0
LPC_GPDMACH0->DMACCControl  = 32            // transfer size (0 - 11) = 32
                            | (0 << 12)     // source burst size (12 - 14) = 1
                            | (0 << 15)     // destination burst size (15 - 17) = 1
                            | (2 << 18)     // source width (18 - 20) = 32 bit
                            | (2 << 21)     // destination width (21 - 23) = 32 bit
                            | (0 << 24)     // source AHB select (24) = AHB 0
                            | (0 << 25)     // destination AHB select (25) = AHB 0
                            | (1 << 26)     // source increment (26) = increment
                            | (0 << 27)     // destination increment (27) = no increment
                            | (0 << 28)     // mode select (28) = access in user mode
                            | (0 << 29)     // (29) = access not bufferable
                            | (0 << 30)     // (30) = access not cacheable
                            | (0 << 31);    // terminal count interrupt disabled
```

Fig 18. DMA channel control for the Timer 0 enabled sample project

Here, we want to transfer the block of data containing the triangle data only once. After the transfer has been completed, the DMA will disable the channel allowing us to use this as an indicator that the entire data transfer has been completed.

```
LPC_TIM0->TCR = 1;                          // enable timer
while (LPC_GPDMACH0->DMACCConfig & 1) ;     // wait for the DMA to finish
LPC_TIM0->TCR = 0;                          // disable timer
```

Fig 19. Polling DMA channel 0

2.3.4 DAC configuration

With the data prepared for transfer and the DMA ready, we can now focus on the DAC. The DAC is fairly simple to configure. The key point to understand is that the DAC has a countdown counter that will generate a DMA request when it reaches 0. Once the counter reaches 0, it will automatically reload with the value specified in DACCNTVAL.

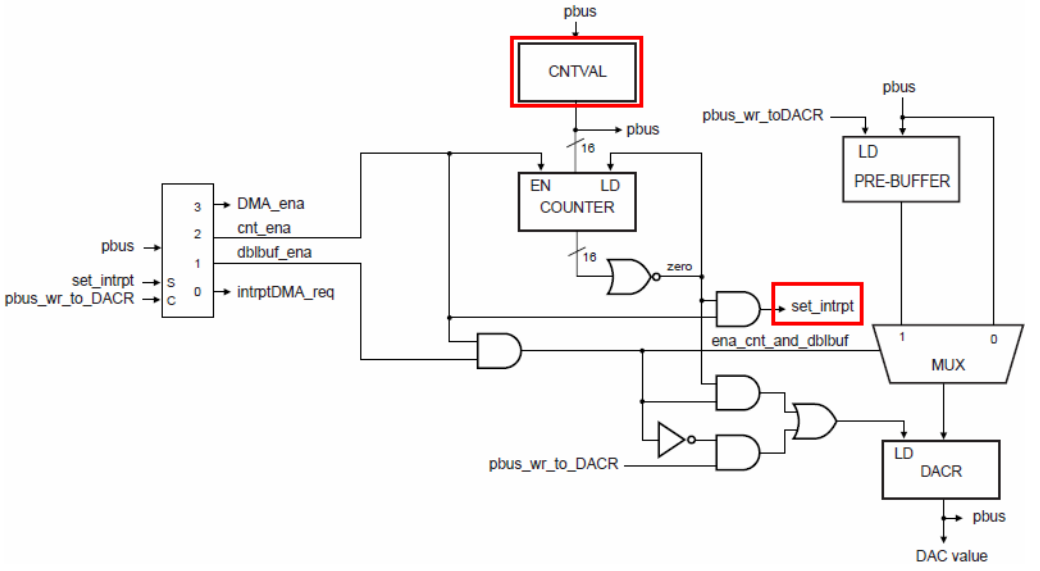


Fig 20. DAC on the LPC1700

The CNTVAL register contains a value which is compared to the DAC's counter. Once it reaches 0 it will generate a "set_intrpt" which will actually request the DMA transfer. The counter on the other hand is incremented at the PCLK_DAC rate.

```
/* DACclk = 25 MHz, so 10 usec interval */
LPC_DAC->DACCNTVAL = 250;
/* DMA, timer running, dbuff */
LPC_DAC->DACCTRL    = 1<<3 | 1<<2 | 1<<1;
```

Fig 21. DAC Configuration

2.3.5 Timer 0 and ADC configuration

Timer 0 and the ADC are only used in the Timer 0 enabled sample project.

Timer 0's match compare register 0 performs the same function as the DAC's count down counter. When a match condition exists, it triggers a 1-word memory-to-peripheral burst transfer. By modifying the interval at which Timer 0's counter matches MAT0.0 we can change the triangle waveform's period.

```
static void T0_Init(void)
{
    /* select MAT0.0 at P1.28 */
    LPC_PINCON->PINSEL3 |= (3<<24);

    /* 100 usec @ 25 Mhz */
    LPC_TIM0->MR0 = 2500;

    /* interrupt on MR0, reset timer on match 0 */
    LPC_TIM0->MCR = 0x0003;

    /* toggle MAT0.0 pin on match */
    LPC_TIM0->EMR = 0x0031;

    /* Reset Timer */
    LPC_TIM0->TCR  = 2;
}
```

Fig 22. Timer 0 Initialization

Once the DMA has been configured and enabled, Timer 0 is then enabled to create the 1-word burst transfers.

When all 32 transfers have been completed Timer 0 can then be disabled. At this point the DMA needs to be re-configured for the next transfer. Now we can introduce a software delay by the CPU before the DMA channel and Timer 0 are reconfigured and enabled. The length of the delay is determined using the ADC.

```
t =  Get_ADC_value();
for (i=0; i<(1000 * t); i++);// Delay
```

Fig 23. ADC controlled delay

The ADC has been initialized to use AD0.2, which is connected to a potentiometer (POT) on the MCB1700 (Fig 1). To get a value from the ADC, we first need to start a conversion and wait for it to complete.

```
void ADC_Init(void)
{
    /* Turn on the ADC */
    LPC_SC->PCONP    |= (1<<12);
    LPC_ADC->ADCR    =    (1<<2) | (0x4<<8) | (1<<21);
    LPC_SC->PCLKSEL0 &= ~(0x3<<24); // PCLK = clk/4  100MHz / 4 = 25MHz

    /* Select P0.25 as AD0.2 for input */
    LPC_PINCON->PINSEL1  = (LPC_PINCON->PINSEL1  & ~(0x3<<18)) | (1<<18);

    /* Disable Pullup and Pulldown resistors */
    LPC_PINCON->PINMODE1 = (LPC_PINCON->PINMODE1 & ~(0x3<<18)) | (0x2<<18);

    return;
}
```

Fig 24. ADC Initialization

2.3.6 Sleep mode

The sleep mode feature has only been added to the sine wave and triangular wave examples since these are the only projects that don't require the CPU once the memory-to-DAC transfer has started.

When the DMA transfers have started, the CPU will toggle the LEDs on P1.28 and P1.29. Once the "INT0" button is pressed, it will turn off the LEDs and put the LPC1700 to sleep. While in sleep mode, the DMA will continue performing the data transfers.

```
while(1)                                          // Loop forever
{
    /* Toggle LED to indicate that CPU is running */
    if(i == 0xF0000){
        LPC_GPIO1->FIOPIN    ^=  (1<<28) | (1<<29);
        i = 0;
        /* If button INT0 is pressed, turn off LEDs and go to sleep */
        if(!(LPC_GPIO2->FIOPIN & (1<<10))){
            LPC_GPIO1->FIOCLR = (1<<28) | (1<<29);
            __WFI();
        }
    }else
        i++;

}
```

Fig 25. Toggling the LEDs and polling to enter Sleep mode

2.3.7 Wave diagrams

The following diagrams depict the different waveforms generated by the sample projects. These waveforms are actual waveform patterned captured using an oscilloscope.

2.3.7.1 Sine wave

Note that in the sine wave diagram, the entire waveform is represented in a positive voltage.

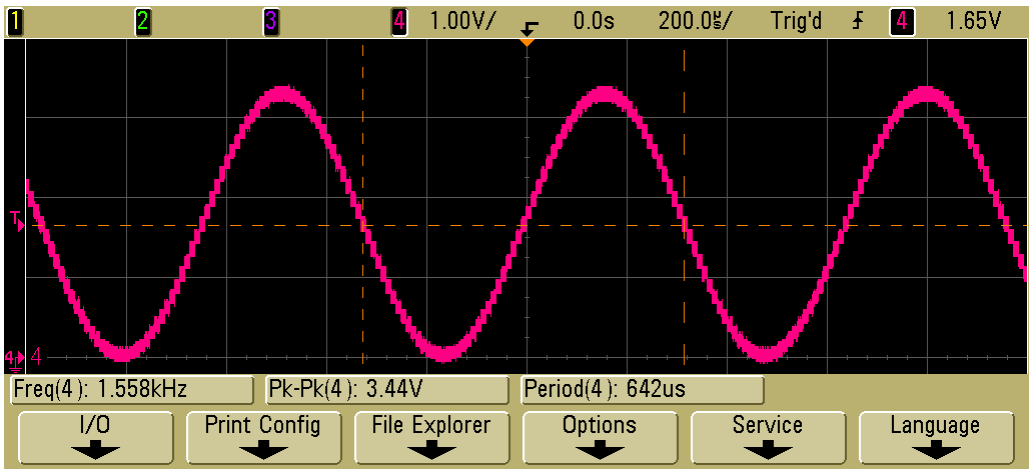


Fig 26. Sine wave output from the DAC

2.3.7.2 Triangle wave

The triangular waveform is outputted in a similar DMA configuration as with the sine wave. Here only the waveform pattern is different and it also only needs 32 1-word burst transfers instead the 64 transfers.

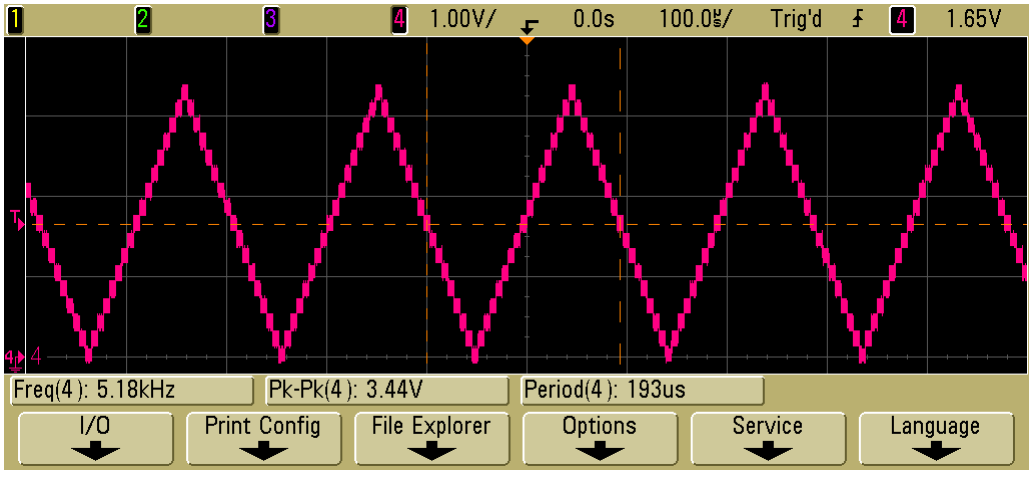


Fig 27. Triangular wave output from the DAC

2.3.7.3 Timer 0 enabled (triangular) wave

This waveform is the same as the triangular wave; however, a small delay is introduced by the CPU (using the ADC). By increasing the value captured by the ADC, the delays between the triangle periods are also increased. The delays are shown as the longer 0 V outputs by the DAC.

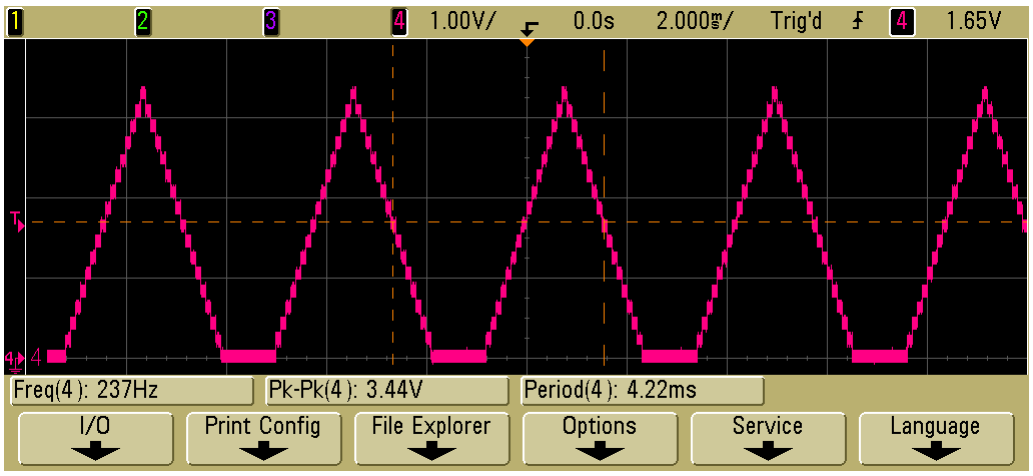
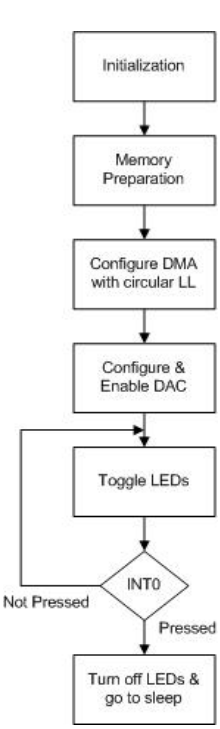


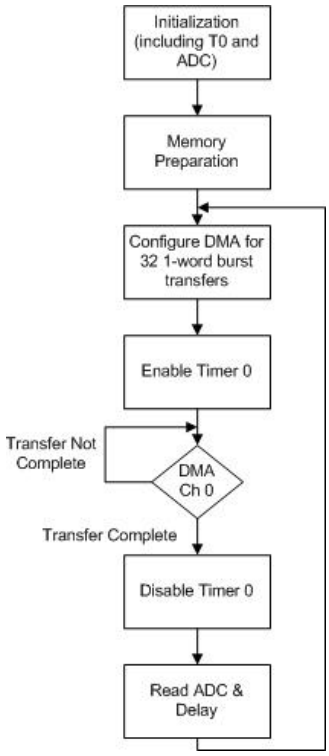
Fig 28. Timer 0 enabled triangular wave output from the DAC

2.4 Project structure

The diagrams in Fig 29 illustrate the basic project flow structure for the three sample projects.



a. Sine wave and Triangular wave sample projects



b. Timer 0 enabled sample project

Fig 29. Flow diagram

2.5 Known issues

2.5.1 Keil uVision

This correction applies to Keil's uVision version 4.00 or earlier.

The LPC17xx.h header file may not specify the "DMAREQSEL" register. To fix this, open the LPC17xx.h header file that comes with the uVision tool and ensure that DMAREQSEL is put in the place of "RESERVED9 as shown below.

```
  __IO uint32_t USBIntSt;                /* USB Device/OTG Interrupt Register  */
  __IO uint32_t DMAREQSEL;
  __IO uint32_t CLKOUTCFG;               /* Clock Output Configuration         */
} LPC_SC_TypeDef;
```

If this line contains "RESERVED9", it must be replaced with

Fig 30. "LPC17xx.h" header file placeholder

2.5.2 IAR workbench

This correction applies to IAR's Workbench 5.4 or earlier.

The "iolpc1766.h" header file needs modification. The address assigned to "DMAREQSEL" needs to be modified so it reads 0x400FC1C4.

```
__IO_REG32_BIT(DMACSYNC,          0x50004034,__READ_WRITE,__dmacsync_bits);
__IO_REG32_BIT(DMAREQSEL,         0x400FC1C4,__READ_WRITE,__dmareqsel_bits);
__IO_REG32(    DMACC0SRCADDR,     0x50004100,__READ_WRITE);
```

Fig 31. "iolpc1766.h" header file correction

3. Legal information

3.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

3.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on a weakness or default in the customer application/use or the application/use of customer's third party customer(s) (hereinafter both referred to as "Application"). It is customer's sole responsibility to check whether the NXP Semiconductors product is suitable and fit for the Application planned. Customer has to do all necessary testing for the Application in order to avoid a default of the Application and the product. NXP Semiconductors does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from national authorities.

3.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

4. Contents

1. Introduction 3
1.1 DMA 3
1.2 Timer 0 3
1.3 ADC 3
1.4 DAC 3
2. Application demo 3
2.1 Requirements 3
2.1.1 Software 3
2.1.2 Hardware 4
2.2 Objective 5
2.3 Project design 6
2.3.1 Data location 6
2.3.2 Data preparation 6
2.3.2.1 DAC's data format 7
2.3.3 DMA configuration 8
2.3.3.1 Linked lists 8
2.3.3.2 DMA triggering 10
2.3.3.3 Channel control 13
2.3.4 DAC configuration 14
2.3.5 Timer 0 and ADC configuration 15
2.3.6 Sleep mode 17
2.3.7 Wave diagrams 18
2.3.7.1 Sine wave 18
2.3.7.2 Triangle wave 18
2.3.7.3 Timer 0 enabled (triangular) wave 19
2.4 Project structure 20
2.5 Known issues 20
2.5.1 Keil uVision 20
2.5.2 IAR workbench 21
3. Legal information 22
3.1 Definitions 22
3.2 Disclaimers 22
3.3 Trademarks 22
4. Contents 23

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an please send an email to:
salesaddresses@nxp.com

Date of release: 8 March 2010
Document identifier: AN10917_1