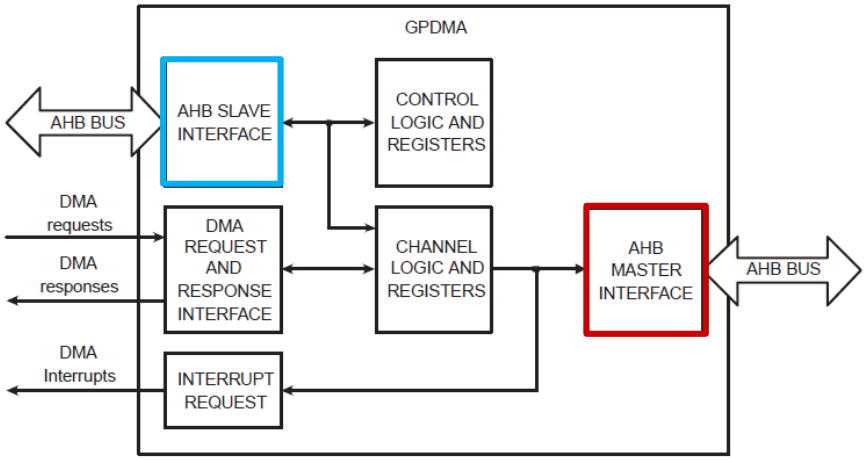
GPDMA

(**G**eneral **P**urpose **DMA** controller)

GPDMA: General Purpose DMA controller

- Direct Memory Access is a feature which allows the peripherals to access the memory directly to transfer data to other peripherals without the help of CPU.

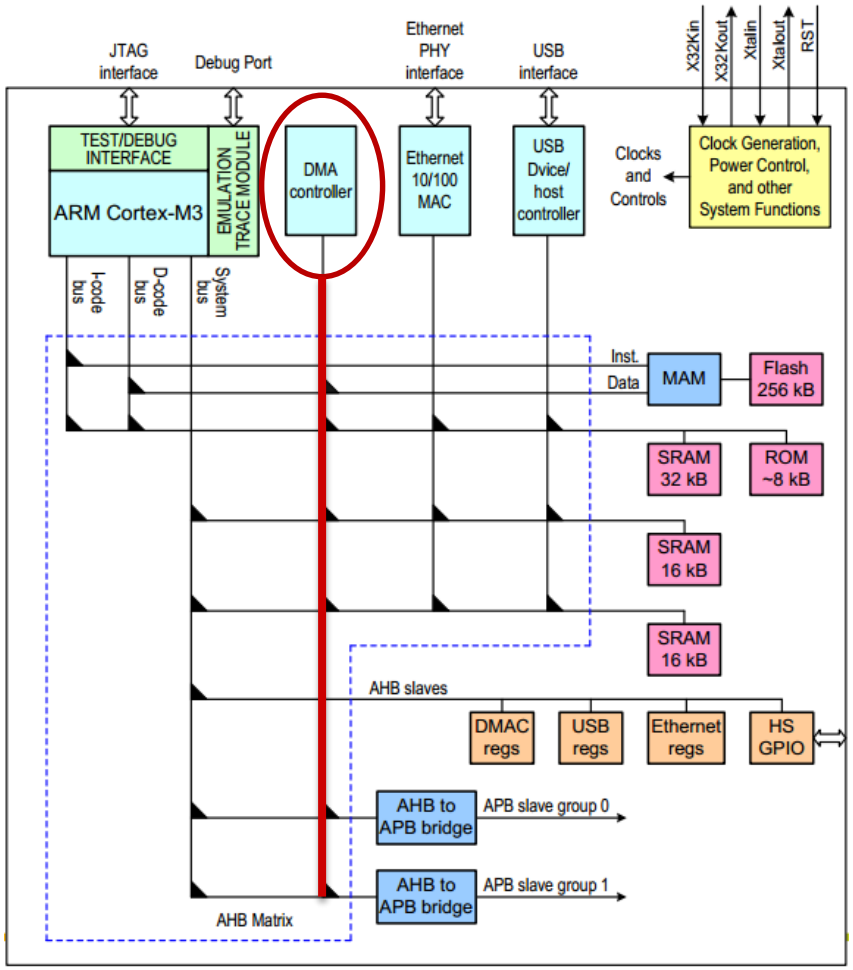


- So whenever we are using DMA, we can be doing computations on the CPU while the data transfer is taking place.

- LPC17xx has a General Purpose DMA (GPDMA) which has **8 channels** of which each can perform **unidirectional data transfer**.
  - All types of data transfers like **memory-memory**, **memory-peripheral**, **peripheral-peripheral** are supported.
  - And we can prioritize the DMA transfers as we wish. It can also perform **8-bit, 16-bit and 32-bit wide transactions**.
  - If we want to transfer a stream of data that is **not stored in contiguous** manner, then we can use the **scatter** and **gather** feature of the DMA which uses linked list.

GPDMA: General Purpose DMA controller

- Supports high-speed peripherals as well as memory-to-memory transfers.
  - 32-bit master bus width (support 8-, 16-, or 32-bit transfers).
  - Eight DMA channels, each with a four-word FIFO.
  - 16 peripheral DMA request lines.
  - Memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral-to-peripheral transfers are supported.
- DMA support for following peripherals:
  - All UARTs.
  - 12-bit ADC and 10-bit DAC.
  - Timer match conditions (trigger only).
  - SSP/ I2S.
  - GPIOs.
- Single DMA and burst DMA request signals.



GPDMA: Register Map

Table 544. GPDMA register map

Name	Description	Access	Reset state	Address
General registers				
DMACIntStat	DMA Interrupt Status Register	RO	0	0x5000 4000
DMACIntTCStat	DMA Interrupt Terminal Count Request Status Register	RO	0	0x5000 4004
DMACIntTCClear	DMA Interrupt Terminal Count Request Clear Register	WO	-	0x5000 4008
DMACIntErrStat	DMA Interrupt Error Status Register	RO	0	0x5000 400C
DMACIntErrClr	DMA Interrupt Error Clear Register	WO	-	0x5000 4010
DMACRawIntTCStat	DMA Raw Interrupt Terminal Count Status Register	RO	0	0x5000 4014
DMACRawIntErrStat	DMA Raw Error Interrupt Status Register	RO	0	0x5000 4018
DMACEnbldChns	DMA Enabled Channel Register	RO	0	0x5000 401C
DMACSoftBReq	DMA Software Burst Request Register	R/W	0	0x5000 4020
DMACSoftSReq	DMA Software Single Request Register	R/W	0	0x5000 4024
DMACSoftLBReq	DMA Software Last Burst Request Register	R/W	0	0x5000 4028
DMACSoftLSReq	DMA Software Last Single Request Register	R/W	0	0x5000 402C
DMACConfig	DMA Configuration Register	R/W	0	0x5000 4030
DMACSync	DMA Synchronization Register	R/W	0	0x5000 4034
DMAREQSEL	Selects between UART and timer DMA requests on channels 8 through 15	R/W	0	0x400F C1C4
Channel 0 registers				
DMACC0SrcAddr	DMA Channel 0 Source Address Register	R/W	0	0x5000 4100
DMACC0DestAddr	DMA Channel 0 Destination Address Register	R/W	0	0x5000 4104
DMACC0LLI	DMA Channel 0 Linked List Item Register	R/W	0	0x5000 4108
DMACC0Control	DMA Channel 0 Control Register	R/W	0	0x5000 410C
DMACC0Config	DMA Channel 0 Configuration Register	R/W	0 [1]	0x5000 4110

GPDMA: General Configuration

- The essential things that you need to do while configuring the DMA are select the channel to be used for data transfer based on the priority.
  - Set the bus **transfer width** (8-bit, 16-bit or 32-bit).
  - Set the **source** and **destination**.
  - Set the DMA **request signal** (The signal that will start the DMA transfer).
  - Set the endian behaviour (Little endian or Big endian).
  - Set the **burst size** (Number of bytes that need to be transferred continuously in one DMA transaction).
  - Set the **block transfer size** (max. 4095)
  - Set the **type of transfer** (Mem-Mem, Mem-Peripheral or Peripheral-Peripheral).

GPDMA: Memory to GPIO transfers example (I)

- Now let us go step by step and setup the GPDMA to perform a simple task of toggle a group of LEDs connected to GPIO:
  - Lets store a pattern series 1's and 0's in an array (100 bytes) and then transfer it to the GPIO port for which the LEDs are connected through channel 0 of the DMA.
  - Timer 1 **generates the DMA request signal** every 0.5 seg.

- **Setup of the timer which generates the DMA request signal:**

```
LPC_SC->PCONP |= 1 << 2;          // Power up DMA
LPC_TIM1->MR0 = F_tick/2 - 1;     // 0.5 seg. (F_tick=F_pclk, PR=0)
LPC_TIM1->MCR = 1 << 1;           // reset on Match Compare 0
LPC_TIM1->TCR = 0x01;             // start timer after DMA init.
```

- **Now lets configure the DMA:**
- 1. Power up the GPDMA (LPC17xx manual table 46, pg 64)

```
LPC_SC->PCONP |= 1 << 29;
//There is no need to setup the clock for DMA as we were doing for other peripherals.
//The system clock will be directly given to the DMA.
```

- 2. Enable the GPDMA (LPC17xx manual table 557, pg 599)

```
LPC_GPDMA->DMACConfig |= 1 << 0;
```

GPDMA: Memory to GPIO transfers example (II)

- 3. Set the Match Compare 0 (MAT1.0 signal) as the DMA request signal.
  - LPC_GPDMA->DMACSync &= ~(1 << 10); // use MAT1.0 for Sync (LPC17xx manual table 558, pg 599) Not needed becouse DMACSync register will be zero on reset.
  - LPC_SC->DMAREQSEL |= 1 << 2; // Timer1 Match Compare 0 as DMA request (LPC17xx manual table 559, pg 600)
- 4. **Clear the Interrupt Terminal Count Request and Interrupt Error Status register**. (Writing 1 to the clear registers will clear the entry in the main register)
  - LPC_GPDMA->DMACIntErrClr |= 0xff; // (LPC17xx manual table 549, pg 596)
  - LPC_GPDMA->DMACIntTCClear |= 0xff; // (LPC17xx manual table 547, pg 595)
- 5. Set the **source and destination addresses** (LPC17xx manual table 560 and 561, pg 601)
  - LPC_GPDMACH0->DMACCDestAddr = (uint32_t) &(LPC_GPIO1->FIOPIN3); // LED is connected to P1[24…31].
  - LPC_GPDMACH0->DMACCSrcAddr = (uint32_t) &data[0]; // data[] is the array where I have stored alternating 1's and 0's

GPDMA: Memory to GPIO transfers example (III)

- 6. Clear the **L**inked **L**ist **I**tem as we are not using it. (LPC17xx manual table 562, pg 602)
  - LPC_GPDMACH0->DMACCLLI = 0;
- 7. Set the burst transfer size, source burst and destination burst size, source and destination width, source increment, destination increment (LPC17xx manual table 563, pg 603)
  - In our case, let the **transfer size be 100 bytes**, source burst and destination **burst sizes are 1**, source and destination **width are 8-bits** and we need to do **source increment** and there is **no need for destination increment**.
  - LPC_GPDMACH0->DMACCControl = 100 | (1 << 26);
- 8. Set the type of transfer as **Memory to Peripheral** and the destination **request line as MAT1.0** (LPC17xx manual table 564, pg 605)
  - LPC_GPDMACH0->DMACCConfig = (10 << 6) | (1 << 11); // 10 corresponds to MAT1.0 and it is selected as the destination request peripheral (LPC17xx manual table 543, pg 592)
- 9. Enable the channel (LPC17xx manual table 564, pg 605)
  - LPC_GPDMACH0->DMACCConfig |= 1; //enable CH0
- 10. Setup an interrupt for DMA transfer completion (optional), so that after the DMA transfer is complete and interrupt request is generated, and in the interrupt service routine, you need to disable the channel by writing:
  - LPC_GPDMACH0->DMACCConfig = 0; // stop CH0 DMA
- 10. Repeat all the steps of configuration to setup another transaction.

GPDMA: Memory to GPIO transfers example (IV)

```
int main (void)
{

  uint32_t i;

  // GPIO configuracion
  //LPC_GPIO3->FIODIR|= 0xFF<<24; // 8 bits mayor peso de P3 como salidas
  LPC_GPIO3->FIODIR3 = 0xFF; // 8 bits mayor peso de P3 como salidas
  LPC_GPIO3->FIOPIN3 = 0;

  // inicializamos 100 bytes en RAM con valores de un contador de 2 bits (00,01,10,11)
  //  para sacarlo al bit 25,26 de P3 (LED1 y LED2 Tarjeta Mini-DK2)
  for(i=0;i<100;i++) data[i]=~ ((i&0x3)<<1); // Pines 25,26

  // setup Timer1
  LPC_SC->PCONP|=1<<2;       // Power up Timer 1
  LPC_TIM1->MR0= Fpclk/2-1; // 0.5 SEGUNDOS (request DMA)
  LPC_TIM1->MCR=1 << 1;      // reset on Match 0
  LPC_TIM1->TCR = 0x01;      // start timer.
  DMAInit();


  while(1);

}
```

GPDMA: Memory to GPIO transfers example (V)

```
void DMAInit(void)
{
  LPC_SC->PCONP|=1<<29;          // Power Up DMA
  LPC_GPDMACH0->DMACCConfig=0;   // Stop CH0 DMA

  LPC_GPDMA->DMACConfig|=1<<0;   // Enable DMA
  LPC_GPDMA->DMACSync|=1<<10;    // MAT1.0 for Sync
  LPC_SC->DMAREQSEL|=1<<2;       // Timer1 Match 0 solicita el DMA

  LPC_GPDMA->DMACIntErrClr|=0xff;
  LPC_GPDMA->DMACIntTCClear|=0xff;

  LPC_GPDMACH0->DMACCDestAddr = (uint32_t) &(LPC_GPIO3->FIOPIN3); // Dirección parte alta de P3 (destino)
  LPC_GPDMACH0->DMACCSrcAddr = (uint32_t) &data[0];               // Comienzo del bloque de memoria (origen)
  LPC_GPDMACH0->DMACCLLI = 0;

  LPC_GPDMACH0->DMACCControl = 100 |( 1 << 26 ); // Transfer size = 100 bytes and enable source increment.

  LPC_GPDMACH0->DMACCConfig = ( 10 << 6 ) | ( 1 << 11); // Set MAT1.0 as destination request peripheral
                                                        //  and the type of transfer as Memory to Peripheral.

  LPC_GPDMACH0->DMACCConfig |=1; //Enable CH0 DMA
}
```

GPDMA: ADC to Memory transfer example (I)

- Programa que demuestra el funcionamiento del ADC por DMA.
- El inicio de conversión del ADC mediante MAT1.0.
- Canal 0 del ADC (AD0.0) es P0.23.
- Necesario hab. la interrup. ADC canal 0 (No habilitar int. ADC en el NVIC)
- DMA en modo transferencia de 8 bits (Acceder a ADDR0[8…15])

```
#include <LPC17xx.H>
#define F_cpu 100e6        // Defecto Keil (xtal=12Mhz)
#define F_pclk F_cpu/4     // Defecto despues del reset
#define F_muestreo 8000    // Frecuencia de muestreo del ADC

 //Configuracion del DMA
#define WIDTH_VALUE 0      // Size Mode Transfer = 8 bits
#define OFFSET_ADC 0x11    // Para acceder a ADDR0[8..15]

//#define WIDTH_VALUE 1      // Size Mode Transfer = 16 bits
//#define OFFSET_ADC 0x10    // Para acceder a ADDR0[0..15]

#define SBURST 0x00
#define DBURST 0x00
#define TAM_BLOCK_DMA 1000 // Tamaño del bloque

unsigned char buffer[TAM_BLOCK_DMA]; // destino de las muestras
```

GPDMA: ADC to Memory transfer example (II)

- Configuración del Timer 0 (Trigger del DMA)

```
/*  Timer 0 en modo Output Compare (reset TOTC on Match 1)
  Counter clk: 25 MHz   MAT0.1 : On match, Toggle pin/output (P1.29)
  Cada 2 Match se provoca el INICIO DE CONVERSIÓN DEL ADC
  Habilitamos la salida (MAT0.1) para observar la frecuencia de muestreo del ADC */

void init_TIMER0(void)
{
    LPC_SC->PCONP|=(1<<1);              //
    LPC_PINCON->PINSEL3|= 0x0C000000; //
    LPC_TIM0->PR = 0x00;                //
    LPC_TIM0->MCR = 0x10;               //
    LPC_TIM0->MR1 = (F_pclk/F_muestreo/2)-1; // DOS Match para iniciar la conversión!!!!
    LPC_TIM0->EMR = 0x00C2;             //
    LPC_TIM0->TCR = 0x01;               //
}
```

GPDMA: ADC to Memory transfer example (III)

- Configuración del ADC

```
void init_ADC(void)
{
  LPC_SC->PCONP|= (1<<12);           // POwer ON
  LPC_PINCON->PINSEL1|= (1<<14);     // ADC input= P0.23 (AD0.0)
  LPC_PINCON->PINMODE1|= (2<<14);    // Deshabilita pullup/pulldown
  LPC_SC->PCLKSEL0&=~(3<<8);         // CCLK/4 (Fpclk después del reset) (100 Mhz/4 = 25Mhz)
  LPC_ADC->ADCR= 0;
  LPC_ADC->ADCR= (0x01<<0)|          // Canal 0
                 (0x01<<8)|          // CLKDIV=1   (Fclk_ADC=25Mhz /(1+1)= 12.5Mhz)
                 (4<<24)|            // Inicio de conversión con el Match 1 del Timer 0
                 (0x01<<21);         // PDN=1
  LPC_ADC->ADINTEN= (1<<0);          // Hab. interrupción fin de conversión canal 0
}
```

GPDMA: ADC to Memory transfer example (IV)

- Configuración del DMA

```
void init_DMA(void)
{

   //POWER-ON
   LPC_SC->PCONP |= 1 << 29;
   LPC_GPDMA->DMACConfig    |= (1<<0); // Hab. DMA

  //Borramos flags (interrupts. pendientes?)
   LPC_GPDMA->DMACIntTCClear = 0xff;
   LPC_GPDMA->DMACIntErrClr = 0xff;

  //Ponemos a 0 CCControl y CCConfig.
   LPC_GPDMACH0->DMACCConfig = 0;
   LPC_GPDMACH0->DMACCControl =0;

   //Origen y destino.
   LPC_GPDMACH0->DMACCDestAddr = (uint32_t)buffer;
   LPC_GPDMACH0->DMACCSrcAddr = LPC_ADC_BASE+OFFSET_ADC; // ADR0 (0x40003420)

   //Linked list item register, solo una fuente.
   LPC_GPDMACH0->DMACCLLI      = 0;

  //Periferico de origen: ADC y Transfer type - Desde periferico a memoria (P2M)

                              //| S.BURST=1  | D.Burst=1  |  S.Width        |     D.Width     |  Incr. Dest | TC interrupt enable.
   LPC_GPDMACH0->DMACCControl|= (SBURST<<12) |(DBURST<<15)|(WIDTH_VALUE<<18)|(WIDTH_VALUE<<21)| ( 1 << 27 ) | (1 << 31);
   LPC_GPDMACH0->DMACCControl|= TAM_BLOCK_DMA;//Tamaño maximo.

                              //ORIGEN: ADC|   Dest=mem. |    P2M    |    ERROR/TC MASK     | Inicio
   LPC_GPDMACH0->DMACCConfig =   ( 4 << 1 ) | (0x00 << 6) | (2 << 11) | (1 << 15) | (1 << 14)| 1;

  //Habilita la interrupcion del DMA en el NVIC
   NVIC_EnableIRQ(DMA_IRQn);
}
```

GPDMA: ADC to Memory transfer example (V)

- Función Interrupción del DMA (Error, o fin de la transferencia)

```
void DMA_IRQHandler(void)
{
static uint32_t contador;
  contador++;
  if (LPC_GPDMA->DMACIntStat & 1) //Interrupcion?
   {
        if (LPC_GPDMA->DMACIntTCStat & 1)//Terminal Count Interrupt Request?
          {
          LPC_GPDMA->DMACIntTCClear = 1; //Borramos interrupcion;
          send_buffer_N(buffer,TAM_BLOCK_DMA);
          }
       if (LPC_GPDMA->DMACIntErrStat & 1)//Error Interrupt Request?
          {
          LPC_GPDMA->DMACIntErrClr = 1; //Borramos interrupcion;
          }
   }

}
```

GPDMA: Basic functions

- LPC17xx.h

```
/*------------- General Purpose Direct Memory Access (GPDMA) ------------------*/
/** @brief  General Purpose Direct Memory Access (GPDMA) register structure definition */
typedef struct                          /* Common Registers                     */
{
  __I  uint32_t DMACIntStat;
  __I  uint32_t DMACIntTCStat;
  __O  uint32_t DMACIntTCClear;
  __I  uint32_t DMACIntErrStat;
  __O  uint32_t DMACIntErrClr;
  __I  uint32_t DMACRawIntTCStat;
  __I  uint32_t DMACRawIntErrStat;
  __I  uint32_t DMACEnbldChns;
  __IO uint32_t DMACSoftBReq;
  __IO uint32_t DMACSoftSReq;
  __IO uint32_t DMACSoftLBReq;
  __IO uint32_t DMACSoftLSReq;
  __IO uint32_t DMACConfig;
  __IO uint32_t DMACSync;
} LPC_GPDMA_TypeDef;

/** @brief  General Purpose Direct Memory Access Channel (GPDMACH) register structure definition */
typedef struct                          /* Channel Registers                    */
{
  __IO uint32_t DMACCSrcAddr;
  __IO uint32_t DMACCDestAddr;
  __IO uint32_t DMACCLLI;
  __IO uint32_t DMACCControl;
  __IO uint32_t DMACCConfig;
} LPC_GPDMACH_TypeDef;
```

GPDMA: Basic functions

- Example of structure

```
typedef union {
  struct {
    uint32_t TRANSFER_SIZE              : 12;
    uint32_t SOURCE_BURST_SIZE          : 3;
    uint32_t DESTINATION_BURST_SIZE     : 3;
    uint32_t SOURCE_TRANSFER_WIDTH      : 3;
    uint32_t DESTINATION_TRANSFER_WIDTH : 3;
    uint32_t                            : 2;  // reserved and must be written 0
    uint32_t SOURCE_INCREMENT           : 1;  // after each transfer, increment source address(1)/don't increment(0)
    uint32_t DESTINATION_INCREMENT      : 1;  // after each transfer, increment destination address(1)/don't increment(0)
    uint32_t                            : 3;  // Prot1..Prot3, not used in LPC17xx
    uint32_t TERMINAL_COUNT_INT_ENABLE  : 1;  // terminal count interrupt enable
  };
  uint32_t reg;
} LPC_DMAC_CxCONTROL_TypeDef;

typedef struct                          /* Channel Registers                */
{
  __IO uint32_t DMACCSrcAddr;
  __IO uint32_t DMACCDestAddr;
  __IO uint32_t DMACCLLI;
  __IO LPC_DMAC_CxCONTROL_TypeDef DMACCControl;
  __IO LPC_DMAC_CxCONFIG_TypeDef DMACCConfig;
       uint32_t RESERVED[3];
} LPC_GPDMACH_TypeDef;

#define LPC_GPDMACH0_BASE     (LPC_AHB_BASE  + 0x04100)
#define LPC_GPDMACH           ((LPC_GPDMACH_TypeDef    *) LPC_GPDMACH0_BASE )  // to be used as an array
```

GPDMA: Basic functions

- dma.c dma.h

```c
// initialise the DMA controller
void DMA_init(void) {

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

    LPC_SC->PCONP |= 1<<29;     //Power GPDMA module

    /* Enable the GPDMA controller */
    LPC_GPDMA->DMACConfig = 1;

    /* Enable synchro logic request 8, MAT0.0 */
    LPC_GPDMA->DMACSync   = 1 << 8;
}
```

```c
#ifndef DMA_H_
#define DMA_H_

#include "LPC17xx.h"
#include <defs.h>

void DMA_init(void);
void DMA_config(signed long* outputBuffer);
void DMA_waitForTransfer(void);

#endif /* DMA_H_ */
```

GPDMA: Basic functions

```
// Configure the DMA controller
void DMA_config(signed long* outputBuffer) {
    LPC_GPDMACH0->DMACCSrcAddr  = (uint32_t) outputBuffer;
    LPC_GPDMACH0->DMACCDestAddr = (uint32_t) &(LPC_DAC->DACR);
    LPC_GPDMACH0->DMACCLLI      = 0;            // linked lists for ch0
    LPC_GPDMACH0->DMACCControl  = bufferSize// 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

    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
}
```

```
void DMA_waitForTransfer(void) {
    // wait for the DMA to finish
    while (LPC_GPDMACH0->DMACCConfig & 1);
}
```