The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

**How to Check the Ordering Part Number**

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

**For More Information**

Please contact your local sales office for additional information about Cypress products and solutions.

**About Cypress**

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

AN205266

# FR, MB91460, Direct Memory Access

This application note describes the functionality of the Direct Memory Access Controller (DMAC) and gives some examples. The DMAC is mainly used to transfer data between a source and a destination memory location without any CPU load.

## Contents

1 Introduction....1
1.1 Key Features....1
2 Direct Memory Access....2
2.1 Block Diagram....2
2.2 Outline....3
2.3 Registers....5
2.4 Operation....8
3 DMAC Examples....12
3.4 DMAC Example with External Bus....12
3.5 DMAC Example with UART....17
4 Additional Information....20
Document History....21

## 1 Introduction

This application note describes the functionality of the Direct Memory Access Controller (DMAC) and gives some examples.

The DMAC is mainly used to transfer data between a source and a destination memory location without any CPU load. The direction can be from resource (peripheral) to memory and vice versa. This transfer can be a single transfer or multiple transfers from single address or an address area. DMA is always triggered by a resource interrupt and does not interrupt the CPU until the transfer has ended.

### 1.1 Key Features

- Byte / Half-word / Word Transfer
- Transfer from External Area / Built-in I/O / Internal RAM to External Area / Built-in I/O / Internal RAM
- Entire space of 32-bit is addressable
- Step / Block / Bulk / Demand Transfers with possibility of update (increment or decrement) of source and destination address with Reload
- Block Count and Transfer Count (with reload) registers available
- External input pin or built-in peripherals or software requests as DMAC transfer request sources
- External Transfer requests available on DMAC channels 0 to 3
- DMA can be stopped by STOP request (supported by UART-Receive interrupt, in case of receive error)

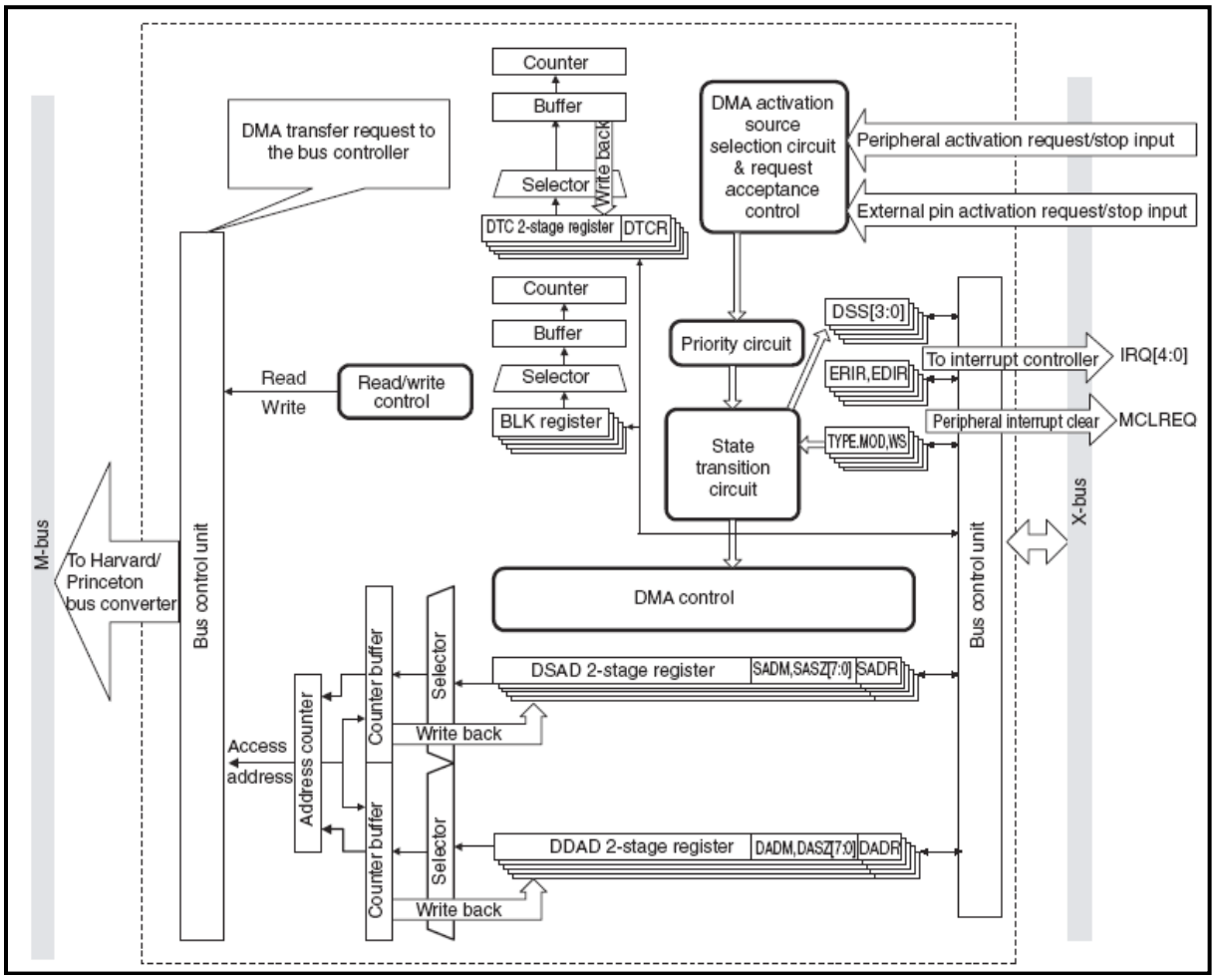
# 2 Direct Memory Access

The basic functionality of the DMAC

## 2.1 Block Diagram

The below figure shows the block diagram of DMAC of FR

Figure 1. Block Diagram of DMAC



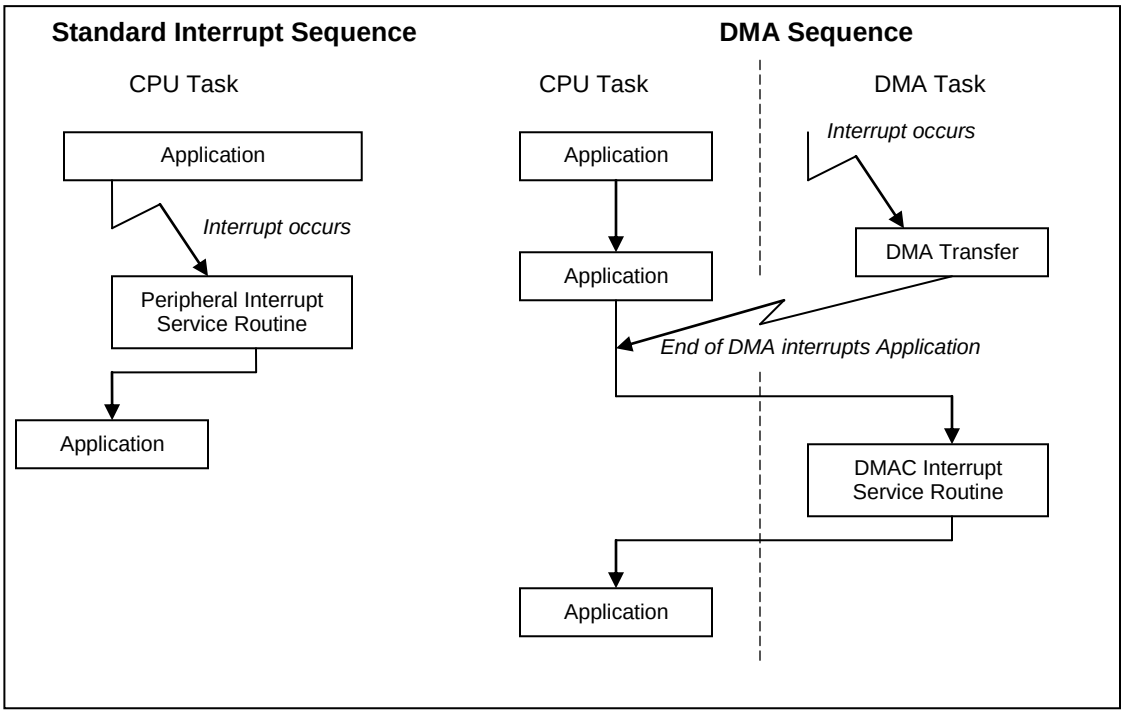
## 2.2 Outline

The DMAC is used to transfer data from one address (space) to another without involving the CPU. The address space includes RAM, ROM, resource (except DMAC) registers or external area.

### 2.2.1 Single Transfer

The following diagram shows a standard interrupt sequence and a single transfer DMA sequence:

Figure 2. Single Transfer

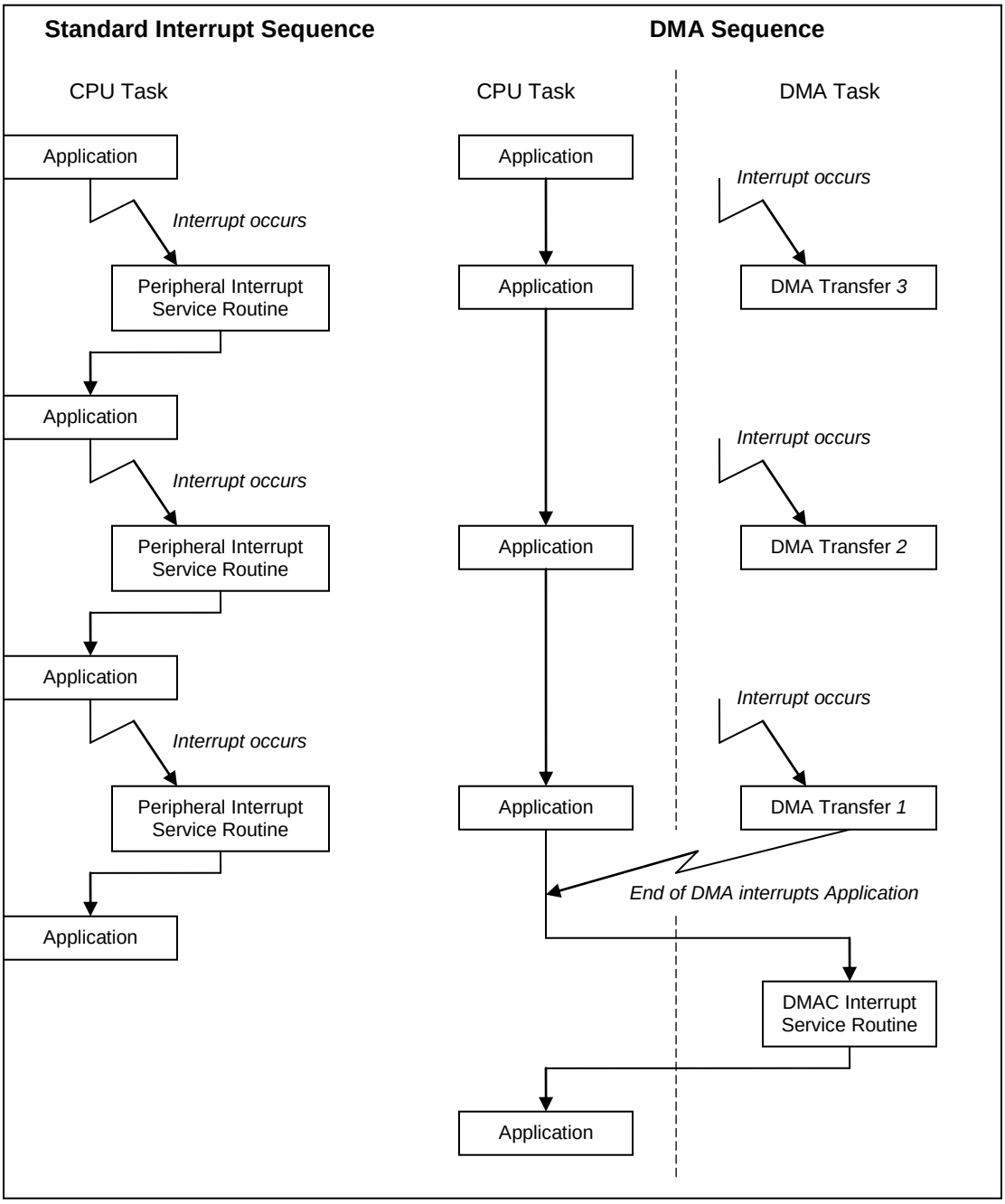



For a single transfer the DMA does not offer a real advantage against a standard interrupt.

### 2.2.2 Multiple Transfers

The following diagram shows a standard interrupt sequence and a multiple transfer DMA sequence:

Figure 3. Multiple Transfers




For a multiple transfer the DMAC advantage is obvious. During Transfer *3* and *2* the application is not interrupted as in the standard interrupt sequence. Only after the Transfer *1*, the interrupt service routine of the DMAC is executed.

However, it should be also noted that the DMAC shares the same bus as of CPU, hence during the time DMA is engaged in the data transfer the CPU is either waiting for the DMA to finish the transfer or it is executing the code from the pre-fetch queue.

## 2.3 Registers

### 2.3.1 Control/Status Registers A (DMACA0-4)

These registers control the operation of the corresponding DMAC channel.

Table 1. DMACAn

| Bit No. | Name | Explanation | Initial Value | Value | Operation |
|---|---|---|---|---|---|
| 31 | DENB | DMA Enable[1] | 0 | 0 | Disables DMA operation on the corresponding channel |
| | | | | 1 | Enables DMA operation on the corresponding channel |
| 30 | PAUS | Pause | 0 | 0 | Enables DMA operation on the corresponding channel |
| | | | | 1 | Temporarily stops DMA operation on the corresponding channel |
| 29 | STRG | Software Trigger[2] | 0 | 0 | Disabled |
| | | | | 1 | DMA transfer request |
| 28 … 24 | IS4 … IS0 | Input Select[3] | 0,0,0,0,0 | 01110 - 10010 | Selects the source for the transfer request |
| 23 … 20 | EIS3 … EIS0 | Extended Input Select[3] | 0,0,0,0 | 0000 - 0111 | Selects the source for the transfer request along with IS4-0 bits |
| 19 … 16 | BLK3 … BLK0 | Block Size[4] | x | 0000 - 1111 | Selects the block size for corresponding DMAC channel |
| 15 … 0 | DTC15 … DTC00 | DMA Terminal Count | x | 0x0000 – 0xFFFF | Stores the transfer count for corresponding DMAC channel |

[1] When the transfer on the corresponding channel reached the specified count, this bit gets cleared to 0.

[2] The software transfer request by the `STRG` bit function is always valid regardless of the setting of these bits `IS` and `EIS` bits.

[3] Please refer the hardware manual for the transfer source selection.

[4] If all the bits of block size are 0, then the block size is 16 (times one transfer unit).

### 2.3.2 Control/Status Registers B (DMACB0-4)

These registers control the operation of the corresponding DMAC channel.

Table 2. DMACBn

| Bit No. | Name | Explanation | Initial Value | Value | Operation |
|---|---|---|---|---|---|
| 31 … 30 | TYPE | Transfer Type | 0,0 | 0,0 | 2-cycle transfer |
| | | | | 0,1 | Fly-by: External Memory --> external I/O transfer |
| | | | | 1,0 | Fly-by: external I/O --> external memory transfer |
| | | | | 1,1 | Setting disabled |
| 29 … 28 | MOD | Transfer Mode | 0,0 | 0,0 | Block/Step transfer |
| | | | | 0,1 | Burst transfer |
| | | | | 1,0 | Demand transfer |
| | | | | 1,1 | Setting disabled |
| 27 … 26 | WS | Data Width | 0,0 | 0,0 | Byte transfer |
| | | | | 0,1 | Half-word transfer |
| | | | | 1,0 | Word transfer |
| | | | | 1,1 | Setting disabled |
| 25 | SADM | Source Address Count Mode Select | 0 | 0 | Increment transfer source address |
| | | | | 1 | Decrement transfer source address |
| 24 | DADM | Destination Address Count Mode Select | 0 | 0 | Increment transfer destination address |
| | | | | 1 | Decrement transfer destination address |
| 23 | DTCR | Transfer Count Register Reload | 0 | 0 | Transfer count register reloading disabled |
| | | | | 1 | Transfer count register reloading enabled |
| 22 | SADR | Source Address Register Reload[5] | 0 | 0 | Source address register reloading disabled |
| | | | | 1 | Source address register reloading enabled |
| 21 | DADR | Destination Address Register Reload[5] | 0 | 0 | Destination address register reloading disabled |
| | | | | 1 | Destination address register reloading enabled |
| 20 | ERIE | Error Interrupt Request Enable | 0 | 0 | Error Interrupt Request enabled |
| | | | | 1 | Error Interrupt Request disabled |
| 19 | EDIE | End Interrupt Request Enable | 0 | 0 | End Interrupt Request enabled |
| | | | | 1 | End Interrupt Request disabled |

[5] Source and destination address register reload feature is only valid if the Transfer count register reload feature is enabled (DTCR = 1).

Table 3. DMACBn-II

| Bit No. | Name | Explanation | Initial Value | Value | Operation |
|---|---|---|---|---|---|
| 18 | DSS2 | End Code-1 | 0 | 0 | Initial Value |
| | | | | 1 | DMA stopped temporarily due to DMAH bit or PAUS bit or an interrupt) |
| 17 … 16 | DSS1 … DSS0 | End Code-2 | 0,0 | 0,0 | Initial Value |
| | | | | 0,1 | Address error (underflow/overflow) – ERIE |
| | | | | 1,0 | Transfer stop request – ERIE |
| | | | | 1,1 | Transfer ended normally – EDIE |
| 15 … 8 | SASZ7 … SASZ0 | Source Address Count Size | x | - | Specifies the increment or decrement width for the transfer source address (DMASA) of the corresponding channel in each transfer operation. |
| 7 … 0 | DASZ7 … DASZ0 | Destination Address Count Size | x | - | Specifies the increment or decrement width for the transfer destination address (DMADA) of the corresponding channel in each transfer operation. |

### 2.3.3 Transfer Source Address Setting Register (DMASA0-4)

These registers select transfer source address.

### 2.3.4 Transfer Destination Address Setting Register (DMADA0-4)

These registers select transfer destination address. Transfer source and destination address setting registers can't be read when the transfer is going on. If one tries to read them while the ongoing transfer, the address before the transfer is read and if read after the transfer then the next access address is read.

### 2.3.5 DMAC All-Channel Control Register (DMACR)

This register offers the master control for all five DMAC channels.

Table 4. DMACR

| Bit No. | Name | Explanation | Initial Value | Value | Operation |
|---|---|---|---|---|---|
| 31 | DMAE | DMA Enable | 0 | 0 | DMAC transfer on all channels disabled |
| | | | | 1 | DMAC transfer on all channels enabled |
| 30 … 29 | - | - | x | x | - |
| 28 | PM01 | Priority Mode | 0 | 0 | Fixed priority (ch0 > ch1) |
| | | | | 1 | Alternating priority (ch1 > ch0) |
| 27 … 24 | DMAH3 … DMAH0 | DMA Halt | 0 | 0,0,0,0 | DMAC operation on all channels enabled |
| | | | | 0,0,0,1 - 1,1,1,1 | DMAC operation on all channels temporarily stopped |
| 23 … 0 | - | - | x | x | - |

## 2.4 Operation

The DMAC has 5 channels and all these 5 channel functions can be set independently of each other.

### 2.4.1 Transfer Mode

Each DMAC channel performs transfer according to the transfer mode set by the MOD[1:0] bits of its DMACB register.

#### 2.4.1.1 Step Transfer

Step transfer is performed when the block size (BLK[3:0] of DMACA) is specified as 1 in the block transfer mode (MOD[1:0] = 0 of DMABA).

1. In case of step transfer after the transfer request is received, one transfer operation (of single byte/half-word/word as specified by WS[1:0] bits of DMABA) is performed.
2. The peripheral interrupt, if any, which had initiated the transfer request gets cleared by DMAC
3. Then the transfer is stopped with DSS[1:0] of DMABA register gets set to B'11 indicating transfer ended normally.
4. After the transfer end the DMASA, DMADA registers and BLK[1:0] and DTC[15:00] bits of DMACA register gets updated accordingly.
5. If there is one more transfer request during the transfer is happening then it would be ignored.

#### 2.4.1.2 Block Transfer

Block transfer is performed when the block size (BLK[3:0] of DMACA) is specified as any value greater than 1 in the block transfer mode (MOD[1:0] = B'00 of DMABA).

1. In case of block transfer after the transfer request is received, a single block transfer unit is transferred and then the transfer is stopped. If WS[1:0] of DMABA is B'10 and BLK[3:0] of DMACA is B'1010, then the single block transfer unit would be 10 (32-bit) words.
2. After this the DMASA, DMADA registers gets updated accordingly.
3. The BLK[1:0] of DMACA register become 0 and DTC[15:0] bits of DMACA register gets decremented by 1.
4. The peripheral interrupt, if any, which had initiated the transfer request, gets cleared by DMAC.
5. Then the DMAC waits for next transfer request. During this time the CPU can get access of the buses until the DMAC gets next transfer request.
6. Upon getting the next such transfer requests steps 1 to 4 are performed unless and until the DTC[15:0] becomes 0.
7. Once the DTC[15:0] becomes 0, the transfer is stopped with DSS[1:0] of DMABA register gets set to B'11 indicating transfer ended normally.
8. If the reload is enabled then the DMAC waits for next transfer request.

#### 2.4.1.3 Burst Transfer

For the burst transfer mode, MOD[1:0] bits of DMABA needs to be set as B'01.

1. In case of block transfer after the transfer request is received, transfer is carried out continuously as specified by the transfer count and then the transfer is stopped. If WS[1:0] of DMABA is B'10, BLK[3:0] of DMACA is B'1010 and DTC[15:0] is 0x10 then transfer would be carried out continuously until 160 (32-bit)words are transferred.
2. DTC[15:0] bits of DMACA register would get decremented to 0.
3. After this the DMASA, DMADA registers gets updated accordingly.
4. Once the DTC[15:0] becomes 0, the transfer is stopped with DSS[1:0] of DMABA register gets set to B'11 indicating transfer ended normally.
5. The peripheral interrupt, if any, which had initiated the transfer request, gets cleared by DMAC.
6. If the reload is enabled then the DMAC waits for next transfer request.
7. During this time the CPU can get access of the buses until the DMAC gets next transfer request.

#### 2.4.1.4 Demand Transfer

For the demand transfer mode, MOD[1:0] bits of DMABA needs to be set as B'10. The number of block remain always 1 and the BLK[3:0] bit setting of DMACA is ignored. The transfer source should be selected as high level or low level at DMAC external pin (DREQ) by configuring IS[4:0] bits of DMACA register as B'01110 or B'01111 respectively. As long as the level at the DREQ pin is maintained as per the configuration of IS[4:0] bits, the transfer request is considered to be active.

1. In case of demand transfer after the transfer request is received (that is the signal level at the DREQ pin is as per as per the configuration of IS[4:0] bits), a single transfer is performed. If WS[1:0] of DMABA is B'10 then one (32-bit) word would be transferred.
2. DTC[15:0] bits of DMACA register gets decremented by 1.
3. After this the DMASA, DMADA registers gets updated accordingly.
4. If the transfer request is not active anymore then the DMAC waits for the transfer request (configured level at DREQ). During this time the CPU can get access of the buses until the DMAC gets next transfer request.
5. If the transfer request is still active steps 1 to 3 are repeated until the DTC[15:0] becomes 0.
6. Once the DTC[15:0] becomes 0, the transfer is stopped with DSS[1:0] of DMABA register gets set to B'11 indicating transfer ended normally.
7. The peripheral interrupt, if any, which had initiated the transfer request, gets cleared by DMAC.
8. If the reload is enabled then the DMAC waits for next transfer request.

### 2.4.2 Transfer Type

#### 2.4.2.1 2-cycle Transfer

In 2-cycle transfer the DMAC performs 2 operations sequentially. Upon receiving the transfer request it reads the data from an address specified in the transfer source register DMASA and then writes it to the address as specified by transfer destination register DMADA.

In case of block/step 2-cycle transfer all areas accessible by 32-bit addressing are specifiable, however for demand 2-cycle transfer make sure to set an external area address as transfer source or transfer destination or both.

#### 2.4.2.2 Fly-by Transfer

Fly-by transfer is used when the data needs to be transferred from one external peripheral to the other or vice a versa such as external memory, external I/O etc.

External Memory to External I/O:

In this case the DMAC operates using a read operation as its unit of operation. DMAC issues a fly-by transfer (read) request to the bus controller and the bus controller in turn issues the read command to the external memory and at the same time it also issues write command to the external I/O. Simultaneously the bus controller floats the address and data bus with the appropriate content. Hence two sequential operations (read followed by write) are replaced by two simultaneous operations (read with write).

External I/O to External Memory:

In this case the DMAC operates using a write operation as its unit of operation. DMAC issues a fly-by transfer (read) request to the bus controller and the bus controller in turn issues the write command to the external memory and at the same time it also issues read command to the external I/O. Simultaneously the bus controller floats the address and data bus with the appropriate content. Hence two sequential operations (read followed by write) are replaced by two simultaneous operations (read with write).

For Fly-by transfer the value configured in the transfer destination address register DMADA is used as the address for access and it should point to the external area where as the transfer destination address register DMASA is ignored.

### 2.4.3 DMA Transfer and Interrupts

By default the DMAC has the access of D-Bus. Whether the CPU is executing any function or interrupt service routine the DMAC can request the CPU for the D-Bus by asserting D-Bus hold request (DHREQ), the CPU in turn would respond to this DMA request by D-Bus hold acknowledge (DHACK) granting the access of the D-Bus to the DMA.In such situation NMI or peripheral interrupt can cancel the hold request (depending upon configuration of LVL bits if HRCL register) and gain the access of the bus while corresponding ISR executes. Upon execution of RETI instruction the DMA would gain access to the D-Bus again (provided the MHALTI flag and the corresponding peripheral interrupt flag is cleared in the ISR).

During DMA transfer, interrupts are generally not accepted until the transfer ends.

- • If a DMA transfer request occurs during interrupt processing, the transfer request is accepted and interrupt processing is stopped until the transfer is completed.
- • If, as an exception, an NMI request or an interrupt request with a higher level than the hold suppress level set by the interrupt controller occurs, DMAC temporarily cancels the transfer request via the bus controller at a transfer unit boundary (one block) to temporarily stop the transfer until the interrupt request is cleared. In the meantime, the transfer request is retained internally. After the interrupt request is cleared, DMAC reissues a transfer request to the bus controller to acquire the right to use the bus and then restarts DMA transfer.

When the high priority interrupt occurs (NMI or interrupt priority higher than that specified by LVL bits if HRCL register) during the DMAC transfer, the DMAC transfer gets temporarily stopped by the hold request cancel request from NMI or by the peripheral. The corresponding ISR starts getting executed. Now within this ISR if the corresponding interrupt flag is cleared then DMAC transfer is restarted. In this case to continue suppressing the DMAC transfer, the DMAH[3:0] bits of DMACR register should be incremented by 1 before clearing the interrupt flag and it should be decremented by 1 before leaving the ISR. This is how the DMAC transfer would be suppressed until the execution of ISR. In such case the DMAC transfer would be continued before the execution of RETI instruction. In case of nested interrupts the DMAC transfer would be only continued after the lowest priority interrupt (interrupt with the lowest priority among the pending interrupt but priority higher than that specified by LVL bits if HRCL register) finishes execution.

### 2.4.4 External Transfer Requests

Table 5. External Transfer Request

| Sr. No | Pin Name | Description | Functionality |
|---|---|---|---|
| 1 | DREQ | External Transfer Request Input Pin | External transfer request can be issued over DREQ pin. The configurable levels are low level or high level and the configurable edges are rising edge or falling edge. It should be noted that the minimum width of the signal appearing at DREQ should be 5 system clock cycles. |
| 2 | DACKX | External Transfer Request Acknowledgement Output Pin | This is an acknowledgement signal with reference to DREQ. It indicates that the external transfer with respect to the transfer request is performed and usually being asserted during the external bus access. In the 2-cycle transfer it is asserted either while the external source or while the external destination is being accessed.<br>In the fly-by transfer it is continuous asserted since the source as well as the destination is external. |
| 3 | DEOP | DMA End Output Pin | Signal on this pin indicates that the DMAC transfer is performed for specified number of times.<br>DEOP is asserted while the last data item of the last block is being transferred<br>If the transfer source and destination is internal then the DEOP is not asserted. |
| 4 | DEOTX | External Transfer Stop Pin | Using this pin the ongoing DMAC transfer can be forcibly stopped.<br>If the transfer is stopped using this pin then the DMAH[2:0] bits of the DMACB register would transfer stop request. Interrupt may be generated if it is enabled.It should be noted that this pin shared with DEOP.<br>The minimum width of the signal appearing at DEOTX pin should be 5 system clock cycles. |

### 2.4.5 External Bus Request Arbitration (BRQ)

When a device is operating in external bus extended mode, an external bus request (BRQ) function can be used. Using BRQ pin any external peripheral can request to use the external bus for the configured chip select area.

While the BRQ is asserted by any external device and the access is granted. Then such external device controls the external bus and performs the data transfer. During such transfer, if DMAC requires to use the external bus for the transfer, the DMAC has to wait until the BRQ is active. Once BRQ is released the DMAC can use external bus.

If the DMAC is already using the external bus and the BRQ is asserted DMAC transfer is temporarily stopped. Once BRQ is released the DMAC can use external bus.

If the DMAC transfer through the external bus and BRQ is simultaneous then DMAC waits till the external device finishes transfer and then DMAC takes over

# 3 DMAC Examples

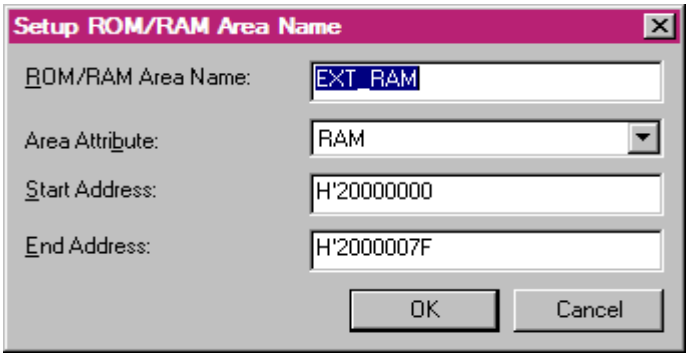
Examples for Direct Memory Access Controller

## 3.4 DMAC Example with External Bus

The following example demonstrates how to configure the DMAC channel 0 with external transfer requests. Here the falling edge on DREQ0 pin triggers the DMAC transfer. Upon such trigger the DMA transfers 1 block of data (which comprises of 4 words) from the internal RAM to the external ram connected via external bus starting from address 0x20000000. The DTC is set as 4, hence after transferring such 4 blocks of data (in response to 4 falling edges), the DMAC0 interrupt is generated. In the DMAC0 ISR the source and destination array is compared. If there is any mismatch then 0xAA is output on LEDs connected to Port 16, otherwise 0xFF is output on the LEDs.

In order to place the dst_buff[] at the external area starting from address 0x20000000, the ROM/RAM area needs to be set up as below:

Figure 4. Setup ROM/RAM Area




The above dialog box can be reached as mentioned below…

*Project -> Setup Project -> Linker -> Disposition Connection -> ROM/RAM Area List -> Set*

After doing this, the external bus should be configured using the *Start91460.asm* as below:

Enabling External Bus interface→

```
;=============================================================================
; 4.8 External Bus Interface
;The rest of the configuration is only applicable for devices with an external bus
;interface.
;If the device does not offer an external bus interface, the configuration can be
;stopped at this point.
;=============================================================================
 #set    EXTBUS          ON                ; <<< Ext. Bus on/off
 ; Note: This feature is not supported by every device. Please check the data sheet.
 The Following devices for example do not offer an external bus interface:
 MB91464A,MB91467C, MB91465K, MB91463N, MB91465X.
 ;============================================================================
```

Enabling chip select →

```
; 4.8.1 Select Chip select (Only EXTBUS == ON)
;================================================================================
;
#set    CS0             OFF                     ; <<< select CS (ON/OFF)
#set    CS1             ON                      ; <<< select CS (ON/OFF)
#set    CS2             OFF                     ; <<< select CS (ON/OFF)
#set    CS3             OFF                     ; <<< select CS (ON/OFF)
#set    CS4             OFF                     ; <<< select CS (ON/OFF)
#set    CS5             OFF                     ; <<< select CS (ON/OFF)
#set    CS6             OFF                     ; <<< select CS (ON/OFF)
#set    CS7             OFF                     ; <<< select CS (ON/OFF)
#set    SDRAM           OFF                     ; <<< select if a SDRAM is connected
;
#set    ENACSX          B'00000010              ; <<< set CS, ENACSX
;                         ||||||||
;                         ||||||||__ CS0 bit, enable/disable CS0 (1/0)
;                         |||||||___ CS1 bit, enable/disable CS1 (1/0)
;                         ||||||____ CS2 bit, enable/disable CS2 (1/0)
;                         |||||_____ CS3 bit, enable/disable CS3 (1/0)
;                         ||||______ CS4 bit, enable/disable CS4 (1/0)
;                         |||_______ CS5 bit, enable/disable CS5 (1/0)
;                         ||________ CS6 bit, enable/disable CS6 (1/0)
;                         |_________ CS7 bit, enable/disable CS7 (1/0)
;
; Note: If the SWB Monitor Debugger is used, set the CS1 (external RAM only) or CS0
 and CS 1 (external RAM and flash) to off.
; Note: Not all Chip selects are supported by the different devices.  Please check
 the data sheet.
```

Set momory addressing →

```
;================================================================================
; 4.8.2 Set memory addressing for Chip selects (only EXTBUS == ON)
;================================================================================
 #set    AREASEL0        0x0020                  ; <<< set start add. for CS0, ASR0
#set    AREASEL1        0x2000                  ; <<< set start add. for CS1, ASR1
#set    AREASEL2        0x0000                  ; <<< set start add. for CS2, ASR2
#set    AREASEL3        0x0000                  ; <<< set start add. for CS3, ASR3
#set    AREASEL4        0x0000                  ; <<< set start add. for CS4, ASR4
#set    AREASEL5        0x0000                  ; <<< set start add. for CS5, ASR5
#set    AREASEL6        0x3000                  ; <<< set start add. for CS6, ASR6
#set    AREASEL7        0x0000                  ; <<< set start add. for CS7, ASR7

; Configure the starting address of each used Chip select. Chip selects which are not
; used (not set to ON in "Select Chip select") need not be set (setting ignored).
;
; NOTE: Just the upper 16-bit of the start address must be set, e.g. when using start
 ;address 0x00080000 set 0x0008.
```

Configure chip select area →

```
;=============================================================================
; 4.8.3 Configure Chip select Area (only EXTBUS == ON)
;=============================================================================
#set    CONFIGCS0       B'1000100000100010      ; <<< Config. CS0, ACR0
#set    CONFIGCS1       B'0110100000100010      ; <<< Config. CS1, ACR1
#set    CONFIGCS2       B'0000000000000000      ; <<< Config. CS2, ACR2
#set    CONFIGCS3       B'0000000000000000      ; <<< Config. CS3, ACR3
#set    CONFIGCS4       B'0000000000000000      ; <<< Config. CS4, ACR4
#set    CONFIGCS5       B'0000000000000000      ; <<< Config. CS5, ACR5
#set    CONFIGCS6       B'0111100001101000      ; <<< Config. CS6, ACR6
#set    CONFIGCS7       B'0000000000000000      ; <<< Config. CS7, ACR7
;                         ||||||||||||||||
;                         ||||||||||||||||__ TYP0 bit, TYP0-4 bits select access
type
;                         |||||||||||||___ TYP1 bit
;                         ||||||||||||____ TYP2 bit
;                         |||||||||||_____ TYP3 bit
;                         ||||||||||______ LEND bit, select little '1' or big
endian '0'
;                         |||||||||_______ WREN bit, en-/disable (1/0) Write
access
;                         ||||||||________ PFEN bit, en-/disable (1/0) pre-fetch
;                         |||||||_________ SREN bit, en-/disable (1/0)
;                         ||||||__________ BST0 bit, BSTx bits select burst size
;                         |||||___________ BST1 bit
;                         ||||____________ DBW0 bit, DBWx select data bus width
;                         |||_____________ DBW1 bit
;                         ||______________ ASZ0 bit, ASZx bits select address size
of CS
;                         |_______________ ASZ1 bit
;                                          ASZ2 bit
;                                          ASZ3 bit
```

**Main.c**

```
/*                              SAMPLE CODE                                   */
/*----------------------------------------------------------------------------*/

#define COUNT 16
#define MEM_START 0x20000000

int src_buff[COUNT] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};

#pragma segment DATA=_EXT_RAM, attr=DATA
int dst_buff[COUNT];
#pragma segment DATA

void Setup_DMA0(void)
{
  PFR13_D0 = 1;                     // Enable DREQ0
  PFR13_D1 = 1;                     // Enable DACK0

  DMACR_DMAE = 1;                   // Enable all DMAC channels
  DMACA0_DENB = 0;                  // Disable DMAC channel 0

  DMASA0 = (IO_LWORD)&src_buff;     // Source Start Address
  DMADA0 = (IO_LWORD)&dst_buff;     // Destination Start Address

  DMACA0_IS = 0xF;                 // External DMA-pin low level or falling edge

  DMACB0_WS1 = 1; DMACB0_WS0 = 0; // Word-width

  DMACB0_EDIE = 1;                  // IRQ when transfer finished

  DMACB0_SADM = 0;                  // Source Address increment
  DMACB0_DADM = 0;                  // Destination Address increment
  DMACB0_SASZ = 0x04;               // Source Address increment by 4
  DMACB0_DASZ = 0x04;               // destination Address increment by 4

  DMACB0_DTCR = 0 ;                 // Reload DTC disable
  DMACA0_BLK = 4;                   // Block_Value 4 words
  DMACA0_DTC = 4;                   // DTC set
  DMACB0_MD1 = 0;
  DMACB0_MD0 = 0;                   // Block Mode

  DMACA0_DENB = 1;                  // Enable DMAC channel 0
}

void Setup_Leds(void)
{
DDR16 = 0xff;                       // port as output
PFR16 = 0x00;                       // use GPIO function
PDR16 = 0x00;                       // all LEDs off
}
```

```
void main(void)
{
__EI();                    // Enable interrupts
__set_il(31);              // Allow all levels
InitIrqLevels();           // Init interrupts

PORTEN = 0x3;              // Enable I/O Ports
                           // This feature is not supported by MB91V460A
                           // For all other devices the I/O Ports must be
     enabled

Setup_Leds();              // Initialize led port
Setup_DMA0();              // Initialize DMAC

while(1)                  // endless loop
{
         HWWD_CL = 0;
       }
}

__interrupt void DMA_IRQ(void)
{
  unsigned char i, err = 0;

  if(DMACB0_DSS1 == 1 && DMACB0_DSS0 == 1 )   // Normal End
  {
      DMACB0_DSS = 0;                          // clear DMA channel status

      for (i=0; i<COUNT; i++)
      {
          if (src_buff[i] != dst_buff[i])      // is source diff than dest?
          {
              err = 1;                        // set err variable
                break;
          }
      }

      if (1 == err)
      {
          PDR16 = 0xAA;                     // Glow alternate LEDs connected at
                                            // Port16 to indicate error
      }
      else
      {
          PDR16 = 0xFF;                    // Glow all LEDs connected at Port16
      }
  }
   }
```

**Vectors.c**

Please note, that the corresponding interrupt vector and level has to be defined in the *vectors.c* module of our standard template project.

```
/*                          SAMPLE CODE                                        */
/*-------------------------------------------------------------------------*/
void InitIrqLevels(void)
{
. . .

ICR63 = 20;                             // DMA Controller
                                        // Main/Sub OSC stability wait
. . .
}

__interrupt void DMA_IRQ (void);        // Prototype

   . . .

#pragma intvect DMA_IRQ          76     // DMA Controller
   . . .
```

## 3.5 DMAC Example with UART

The following example code shows how to configure the DMAC with the UART2. The DMAC channel 0 is configured in the block transfer 2-cycle mode. Each block transfer would have 1-byte transferred. If the data is received on UART2 receive buffer RDR02 then the DMA channel 0 transfers the same data to the UART2 transmit buffer TDR02. The transfer takes places unless 50 receptions (since the DTC is set as 0x32) or if there is any error in the reception. After DMAC interrupt gets generated and in the ISR DMAC interrupt flag is cleared with 0xFF output to port 16.

**Main.c**

```
/*                              SAMPLE CODE                                    */
/*-----------------------------------------------------------------------------*/
void Init_UART2(void)
{
  BGR02 = 1666;             // 9600 baud at 16MHz CLKP1
  SCR02 = 0x17;            // 8 bit, clear reception errors, Tx & Rx enabled
  SSR02 = 0x02;            // LSB First, enable receive interrupts
  SMR02 = 0x0D;            // Mode 0, Reset Counter, Reset UART, SOT0 enabled
  PFR20_D0 = 1;           // enable UART
  PFR20_D1 = 1;           // enable UART
  EPFR20_D1 = 0;          // enable UART
}
void Init_DMAC(void)
{
DMACR_DMAE = 1;                    // Enable all DMAC channels
DMACA0_DENB = 0;                   // Disable DMAC channel 0

DMASA0 = (IO_LWORD)&RDR02;         // Source Start Address
DMADA0 = (IO_LWORD)&TDR02;         // Destination Start Address

DMACA0_IS = 0x14;                 // UART02 receive
DMACA0_EIS = 0x3

DMACB0_WS1 = 0; DMACB0_WS0 = 0;   // Byte-width

DMACB0_EDIE = 1;                   // IRQ when transfer finished

DMACB0_SASZ = 0;                   // Source Address Same
DMACB0_DASZ = 0;                   // Destination Address Same
DMACB0_DTCR = 0;                   // Reload DTC Disable
DMACA0_BLK = 1;                    // Block Value 1 byte
DMACA0_DTC = 0x32;                 // DTC set as 50
DMACB0_MD1 = 0;                   // Block Mode
DMACB0_MD0 = 0;
DMACA0_DENB = 1;                  // Enable DMAC channel 0
}
```

### Vectors.c

Please note, that the corresponding interrupt vector and level has to be defined in the vectors.c module of our standard template project.

```
/*                              SAMPLE CODE                                    */
/*-----------------------------------------------------------------------------*/
void InitIrqLevels(void)
{
. . .

ICR63 = 20;           // DMA Controller
                     // Main/Sub OSC stability wait
   . . .
}

__interrupt void DMAIRQHandler (void); // Prototype

   . . .

#pragma intvect DMAIRQHandler    142    // DMA Controller
   . . .
```

# 4 Additional Information

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

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

The software examples related to this application note is:

*91460_dma_uart0*

*91460_dma_extbus*

It can be found on the following Internet page:

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

## Document History

Document Title: AN205266 - FR, MB91460, Direct Memory Access

Document Number: 002-05266

| Revision | ECN | Orig. of Change | Submission Date | Description of Change |
|---|---|---|---|---|
| ** | -- | NOFL | 04/22/2008 | First Version; MPi |
| | | | 06/29/2009 | V1.1; MSt<br>Completed chapter 2.4.3 DMA Transfer and Interrupts |
| *A | 5088825 | NOFL | 04/06/2016 | Converted Spansion Application Note “MCU-AN-300059-E-V11” to Cypress format |
| *B | 5869161 | AESATMP9 | 08/31/2017 | 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

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

© Cypress Semiconductor Corporation, 2008-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.