i.MX25 Basic SDMA activities

By Padykov Igor, TIC,
Freescale Semiconductor, Inc.
Novosibirsk

1.1 Definitions, Acronyms, and Abbreviations

BD: Buffer Descriptor. Data structure located in Host or dedicated host memory space and used for point-to-point data transfer with the SDMA.

CCB: Control Channel Block. Data structure located in Host or dedicated host memory space, each SDMA channel has a dedicated CCB that points to the array of buffer descriptors

SDMA: Smart Direct Memory Access module

Scripts: SDMA program executed on a channel.

WML: Watermark level, lower or upper threshold that triggers a DMA request to the SDMA.

1.2 Overview

The SDMA module is responsible to perform data transfer inside a multi-core platform. Scripts, written in SDMA assembly, have been developed to cover many kinds of data transfer.
I.MX25 processor has two cores : ARM core and SDMA core. Peripherals connected through SDMA SPBA on Figure 1 are considered to belong to Shared Domain, while other belong to Application Processor Domain (arm core or AP), see Figure 1.

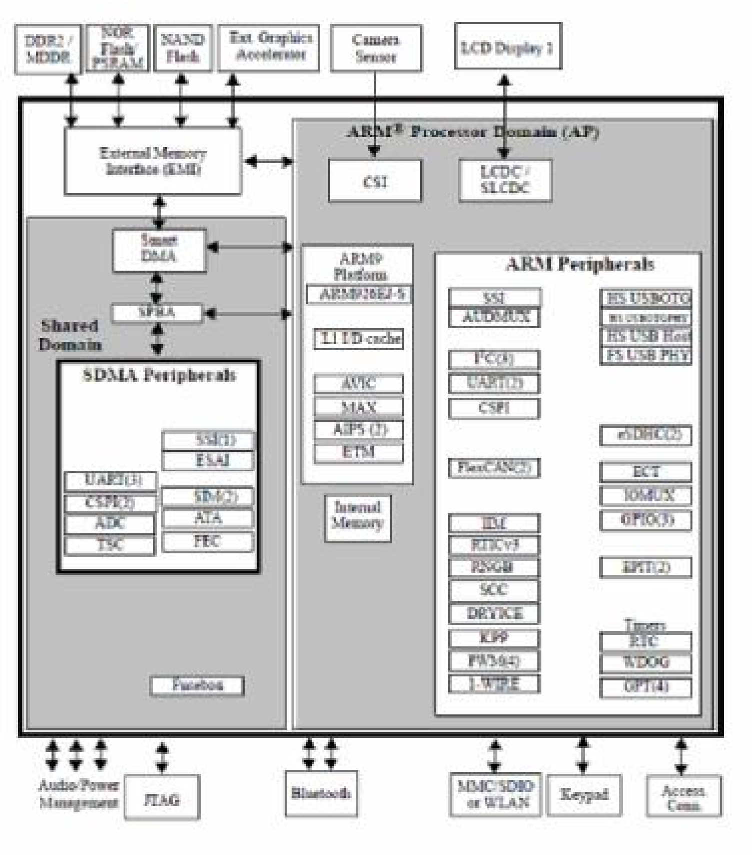


Figure 1 i.MX25 Block Diagram

Host memory - mean the memory space accessible through the MAX of the ARM platform

- accessible by the Peripheral DMA.
-

EMI or External Memory - mean the external memories connected to the External Memory Interface - accessible by the Burst DMA

Shared Peripheral: A same peripheral can be connected to the shared peripheral bus (output of SDMA SPBA module) and to the ARM platform; it is the case for UART, CSPI, and SSI and other . Therefore *shared UART* indicates the UART connected to the shared peripheral, whereas UART means the UART connected to the ARM platform

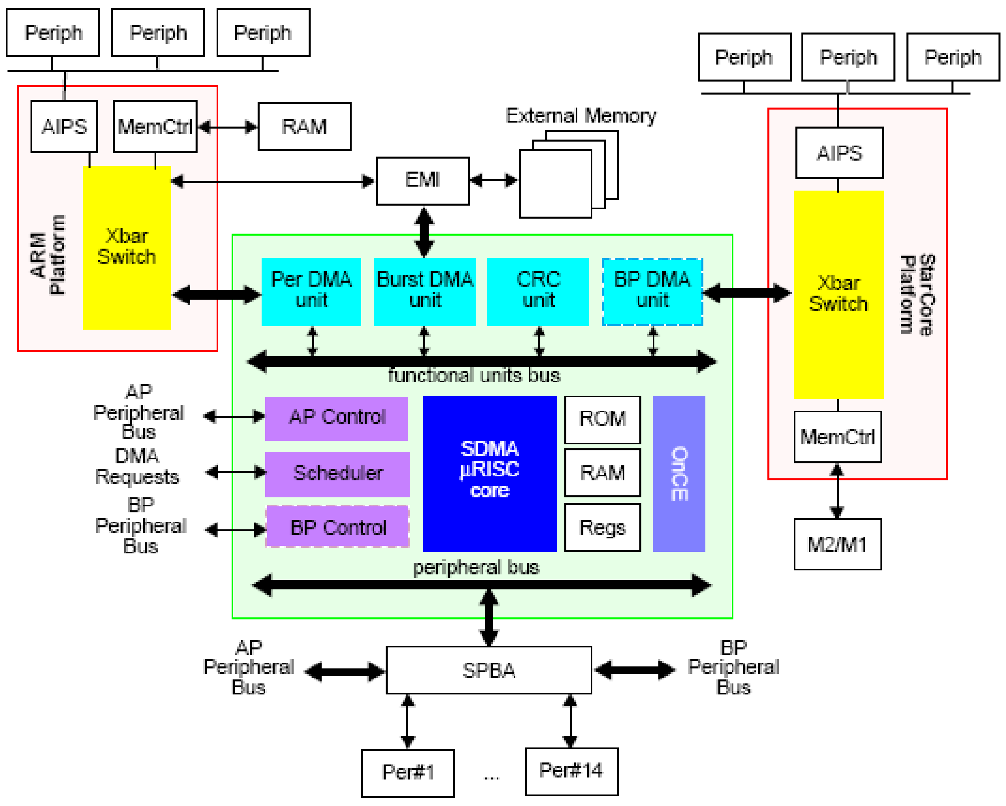


Figure 2 SDMA Module Block diagram

The Smart DMA has two blocks of internal memories: 4 Kbytes of ROM and 8 Kbytes of RAM. SDMA ROM has some number of scripts.

Basically SDMA script library is organized in three sections:

1. memory to memory scripts
2. memory to peripheral scripts
3. peripheral to memory scripts

Script can reside in ROM or in RAM.

2. Parameters required by data transfer script

2.1 Watermark level (WML)

The WML determines the data transfer loop size, meaning the number of bytes that will be read/write from/to the receive/transmit FIFO. This parameter must be a **multiple** of the peripheral FIFO data size.

2.2 Event mask

1-bit high vlaue, which means that if the script attached to the channel must be triggered by DMA request number I, event_mask[I] must be set to 1.

2.2 Peripheral address

Base address or the FIFO address of the peripheral.

2.3 Data length

parameter is passed through the command field of the buffer descriptor and is coded on bits 25, 24 :

00:	32-bit data transfer.
01:	8-bit data transfer.
10:	16-bit data transfer.
11:	24-bit data transfer.

2.4 Parameters required by generic memory to memory (with ap) scripts

The memory to memory (ap source or/and destination) scripts need to have the following parameter set:

M3 Start Address : required to determine which dma port (peripheral/burst DMA) has to be used to do the transfer.

Count: This parameter specifies the total number of bytes that must be transferred

Memory addresses: These 2 word parameters of the Buffer Descriptor structure are both decoded by all the scripts even if only one is useful for the transfer.

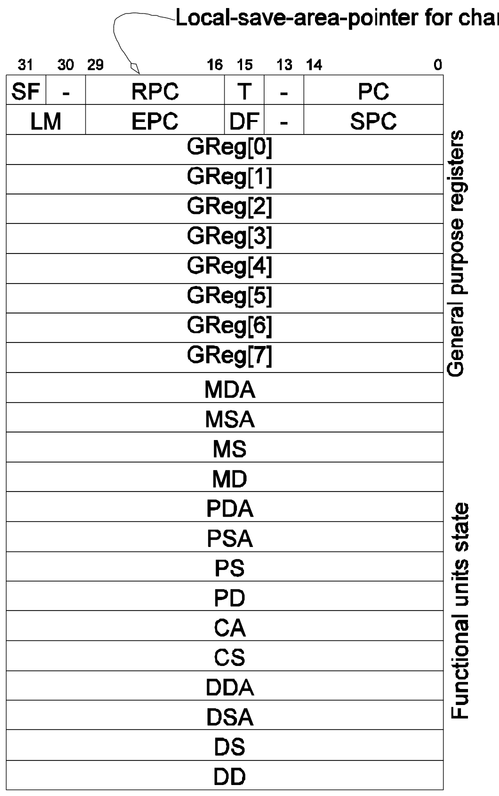
For memory to memory transfers, the first one specifies the source address of the transfer, the second one the destination address.

Command: Some scripts need some specific information passed through this Command parameter.

3. Parameter passing mechanism

Parameters are passed to the script according two mechanisms: by channel context and by buffer descriptor as showed in following diagrams.

3.1 Parameters transmit through the Context

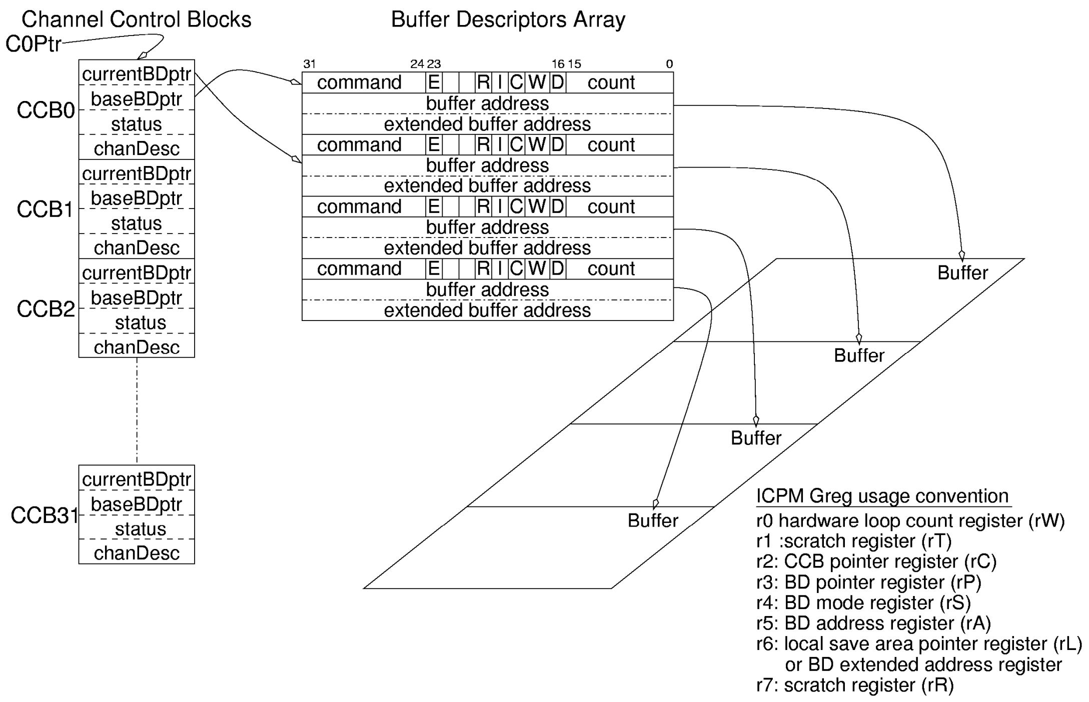


MDA: Host burst DMA address destination register
MSA: Host burst DMA address source register
MS: Host burst DMA status register
MD: Host burst DMA data register
PDA: Host peripheral DMA address destination register
PSA: Host peripheral DMA address source register
PS: Host peripheral DMA status register
PD: Host peripheral DMA data register
CA: CRC unit address register
CS: CRC unit status register
DDA: Dedicated burst DMA destination address register
DSA: Dedicated burst DMA source address register
DS: Dedicated burst DMA status register
DD: Dedicated burst DMA data register
Total 24 registers

There are 32 channel context memory structures pointed to by the local save area pointer. These channel context memory structures are fixed. The script in the SDMA computes the memory offset for a given channel based on the structure length and channel number. The figure above shows the structure of the channel context as it is saved in the SDMA local memory.

The PC field of the first register must point to the SDMA RAM address where the script that will be executed.

3.2 Parameters transmit through the Buffer Descriptor


Channel Control Blocks
C0Ptr
CCB0
currentBDptr
baseBDptr
status
chanDesc
CCB1
CCB2
CCB31
Buffer Descriptors Array
31
24 23
16 15
0
command
E
R
I
C
W
D
count
buffer address
extended buffer address
Buffer
ICPM Greg usage convention
r0 hardware loop count register (rW)
r1 :scratch register (rT)
r2: CCB pointer register (rC)
r3: BD pointer register (rP)
r4: BD mode register (rS)
r5: BD address register (rA)
r6: local save area pointer register (rL)
or BD extended address register
r7: scratch register (rR)

For each channel, the supporting memory structures in the dedicated processor are defined by:

- The Channel Control Blocks (CCB): one for each channel.
- The Buffer Descriptor array (BD): a buffer descriptor points to the “real” data buffer of the data to be transferred from source to destination and defines its properties and state.

Typically, in the BD data structure the first 32 bit word is called mode word; the next two words are base and extended buffer address. Table below shows the field layout:

31 30 29 28 27 26 25 24	23	22	21	20	19	18	17	16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Command	-	-	L	R	I	C	W	D	Count
Buffer Address									
Extended Buffer Address									

Table Buffer Descriptor Format

Description for some of the fields:

- Count: Number of bytes for this transfer
- D: D=0 means SDMA has done the transfer for this BD while D=1 means not
- W: Wrap. If W=1, after current BD is done, will wrap to the base BD(pointed by basdBDptr in CCB)
- C: Continuous. If C=1, after current BD is done, will move to the next BD
- I: Interrupt. If I=1, after current BD is done, will set the corresponding bit(according to the channel number) in SDMA interrupt register
- R: Error. If R=1, there’s error happened during current BD transfer

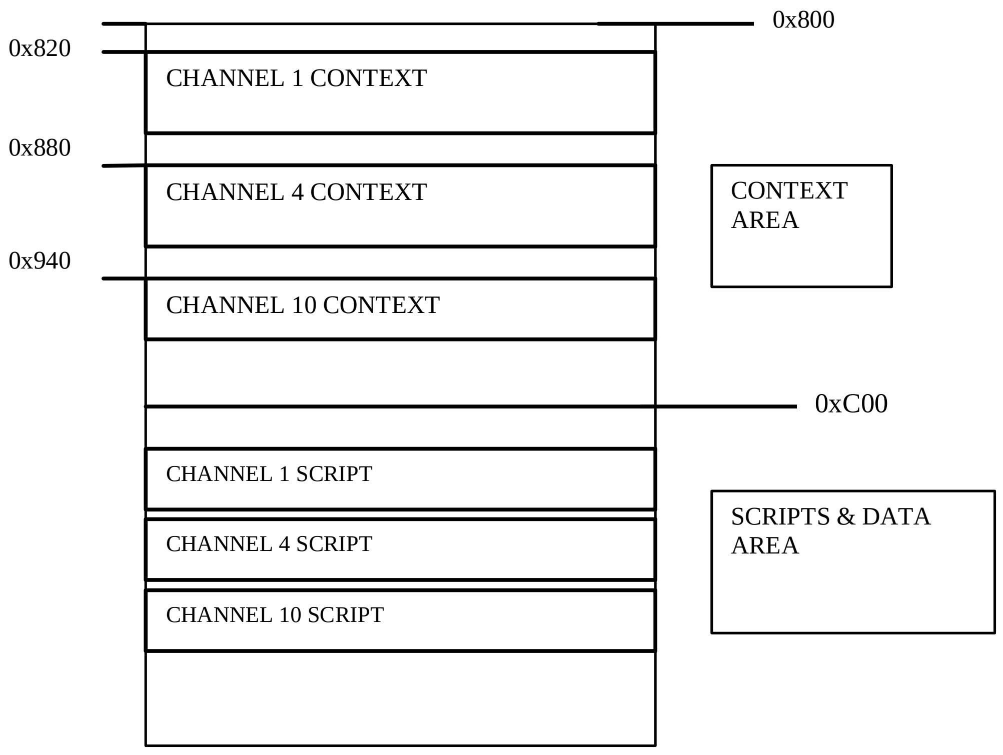
- L: Last buffer descriptor. This bit is set in SDMA IPC scripts to indicate to the receiving Core that the transfer has ended
- Command: This field is used to differentiate operations performed in the script. Usage of this field varies from script to script. Typically, bit 24 and 25 are used to indicate the bus width for many scripts.

If Continuous bit is set, the next BD right behind the current one will be processed after current one is finished. So with the Continuous bit set, BDs can constitute a BD chain. For one channel, up to 64 BDs can be supported in the chain. The Continuous bit of the last BD in the chain should be cleared.

4.0 Loading SDMA scripts

The SDMA channel 0 is dedicated to the boot session, its goal is to download into the SDMA RAM the code and the context of the different SDMA scripts that will be later used during the application. In described below example Channel 10 will be used for loading mcu_2_app script for SDRAM memory to CSPI1 DMA data transfer.

After boot code execution, SDMA memory will be populated with the contexts and scripts as presented in next diagram:



The Channel 0 control block, which is located at address pointed by MC0Ptr register, holds a pointer to the array of buffer descriptors. The buffer descriptors are used to tell the channel 0 (boot channel) what to do. Boot code first read the MC0Ptr pointer to know where the CCB is located, then it reads it to catch the pointer to the array of buffer descriptors.

5.0 Writing simple SDMA application

mcu_2_app script will be used, script file is sdma_script_code_ROMv2.h .

Brief description of script is given below.

5.1 mcu_2_app

This generic script is used to transfer data from memories accessed by the BurstDMA (External memories) to a 8/16/24 or 32 bits peripheral connected to the AIPS. It can be used for SSI(8/16/24 or 32bits data size), for CSPI(32bits data size), for SIM (16bits data size) or for UART1,2 (8bits data size), these peripherals being connected to AIPS.

Parameters transmit through the context

r0: mask to check events2 – If script is triggered by event 32+I, r0[I] must be set to 1. (Project dependent)

r1: mask to check events – If script is triggered by event I, r1[I] must be set to 1. (Project dependent)

r6: address of the peripheral Tx fifo (project dependent)

r7: Watermark level – Used to determine the maximum of data that can be retrieved from the peripheral each time the channel is started.

Parameters transmit trough the Buffer descriptor

The peripheral size/data length is set in the command field of the first Buffer descriptor word, specially bits 24,25.

The number of bytes to transmit is stored in the first Buffer descriptor word (count field)

The source address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used.

Step1 . Prepare structures for Channel 0 – it is used for laoding scripts to RAM
and Channel 10 - used channel for data transfer (SDRAM to CSPI1)

```
#define SRC           0x82000000  //   source
#define DEST          0x82200000  //   destination in SDRAM
#define BUFF_SIZE     0x00002000    / /   buffer size

#define BD_DONE  0x010000     // Buffer Descriptor constants
#define BD_WRAP  0x020000
#define BD_CONT  0x040000
#define BD_INTR  0x080000
#define BD_RROR  0x100000
#define BD_LAST  0x200000
#define BD_EXTD  0x800000

//  Channel Control Block //

typedef struct dummyCCB {
  unsigned long baseBDptr;
  unsigned long currentBDptr;
  unsigned long status;
  unsigned long channelDescriptor;
} channelControlBlock;

channelControlBlock CCB[32];
```

```
unsigned long CTXT_CH10_PTR[32];     // Context for Channel 10

unsigned long BDCh0[3];
unsigned long BDCh10[3];
```

Step2 .   Fill structures

```
  // **** CHANNEL CONTROL BLOCK *** //
  // connect buffer descriptors with channel control block
  CCB[0].baseBDptr         = (unsigned long)&BDCh0;
  CCB[0].currentBDptr      = 0x00000000;
  CCB[0].status            = 0x00000000;
  CCB[0].channelDescriptor = 0x00000000;

  CCB[10].baseBDptr         = (unsigned long)&BDCh10;
  CCB[10].currentBDptr      = 0x00000000;
  CCB[10].status            = 0x00000000;
  CCB[10].channelDescriptor = 0x00000000;




  // setup Buffer Descriptor for channel 0  for loading context for channel 10 to SDMA SRAM

  BDCh0[0] = 0x01810020;                                   //SET DM - Extended - INT - CONT - DONE

  // buffer address
  BDCh0[1] = (unsigned long)&CTXT_CH10_PTR;  // pointer to context of channel 10
                                                             // see channel 10 context
                                                              // extended buffer address
  BDCh0[2] = 0x00000940;                                   // where in SDMA we want to put context
  // size of each context is 32-words.  Starting address of RAM in SDMA
  // is 0x800 (which is start of channel 0 context). So chan10 context is loaded
  // at offset of 10 x 32-words from 0x800, or 0x940.

    //Channel 10 Context
  // Context[0] is the PC - program counter
  CTXT_CH10_PTR[0] = mcu_2_app_ADDR;          // program counter, address offset in SDMA ROM
                                                             // which points to start of script, refer to SDMA
                                                             // script header file.
  // Initialize the other context registers to zero

  for (i=1;i<=31;i++)    CTXT_CH10_PTR[i] = 0x0;


            CTXT_CH10_PTR[3] = 0x00000200;   //R1= event mask  [I]
            CTXT_CH10_PTR[8] = 0x43FA4004;   // R6= SPI_TXFIFO address base+4
            CTXT_CH10_PTR[9] = 0x00000008;   // R7 = watermark – 8 bytes

  // Fill ch10 Buffer Descriptor, set the bits in the parameters for the buffer descriptor
  BDCh10[0] = 0x00810000 | SDMA_SIZE;  //SET DM - Extended - INT - CONT - DONE (need to
                                                  //validate this descriptor
                                                  // SDMA_SIZE is the count, which is number bytes
                                                  // to transfer (bytes total)
                                                  // burst size not set, b/c for peripheral
                                                  // this is defined by the water mark for the periph
                                                   // FIFO
  // set up the buffer and extended buffer descriptor

  BDCh10[1] = SRC;                               // memory source

  // Set priority
  // CHNPRI_0: channel 0 is of pty 7
  reg32_write(SDMA_CHNPRI_0,0x00000007);
  // CHNPRI_1: channel 10 is of pty 1
  reg32_write(SDMA_CHNPRI_10,0x00000001);

  // sets up so that the context is 32 words not 24.
    // Set bit for Scratch RAM
  sdma_data_temp = reg32_read(SDMA_CHN0ADDR);
  sdma_data_temp = sdma_data_temp | 0x00004000;
  reg32_write(SDMA_CHN0ADDR,sdma_data_temp);

  // Event override register.  Since ch0 SDMA not started by peripheral signal
  //  EO for channel 0 = 1 (started by software)
  //  EO for channel 10 = 0 (started by event – CSPI TXFIFO DMA Req)


  reg32_write(SDMA_EVTOVR,0x00000001);                  // channel 0 -via start bit
  reg32_write(SDMA_HOSTOVR,0x00000400);                 // channel 10 -via ext_req

  reg32_write(SDMA_CHENBL_9, 0x400 );     // SPI1_TX event (9 event) maps to 10 chnl

  // Start channel 0 to load to SDMA SRAM the context and buffer descriptor for channel 10

  // write to HSTART register to start channel 0

  //  HE for channel 0
  reg32_write(SDMA_START,0x00000001);

// polling on the done bit in the buffer descriptor parameter
// normally use the interrupt to indicate end of transfer

            while(BDCh0[0]&BD_DONE);

// now that transfer is done, check to see if error
// check to see if error bit is set
            if ( BDCh0[0]&BD_RROR)
            {
                        printf("DMA error detected on channel 0\n");
            }

// start of channel 10 transfer

            // write to CSPI DMA EN register to start SDMA channel 10
            enable_cspi(cspi1);
            cspi_setup_transfer(cspi1, 45, CSPI_CH0, CSPI_MASTER_MODE);
            cspi1->ctrl |= CSPI_CTRL_SMC;
            cspi1->dma =0x2; //enable SPI1 TXFIFO DMA THDEN=1

   // wait till done (poll done bit)
            while(BDCh10[0]&BD_DONE);

            // check for errors that may have occurred during transfer
            if ( BDCh10[0]&BD_RROR)
            {
                        printf("DMA error detected on channel 10\n");
            }
```

I.MX25 PDK SPI Interface

The i.MX25 CPU board does not access the debug board through WEIM interface, but uses the CSPI1 interface instead. The read and write operations are 46 bits in length and are described in the tables below. Reads from the CPLD to the CPU are occurring on the negative edge CSPI1_SCLK signal and writes from the CPU to the CPLD occurring on the positive edge.

CPLD Memory Map

Function	Software Address Offset	Physical Address Line Values (i.mx 31 and 32)							
		CS5_B	A16	A15	A14	A5	A4	A3	A2
R/W, SMSC LAN9217 Ethernet 10/100 BT	0x00000	0	0	0	0	X	X	X	X
R/W, External UART A	0x08000	0	0	0	1	X	X	X	X
R/W, External UART B	0x10000	0	0	1	0	X	X	X	X
R/W, LEDs	0x20000	0	1	0	0	0	0	0	0
Read Only, Status of Switches and Buttons	0x20008	0	1	0	0	0	0	0	1
Read Only, Status of Interrupts	0x20010	0	1	0	0	0	0	1	0
Write Only, Interrupt Reset	0x20020	0	1	0	0	0	1	0	0
R/W, Software Override: UART Routing	0x20028	0	1	0	0	0	1	0	1
R/W, Software Override: Debug Flash Access	0x20030	0	1	0	0	0	1	1	0
R/W, Interrupt Mask	0x20038	0	1	0	0	0	1	1	1
Read Only, Returns AAAA	0x20040	0	1	0	0	1	0	0	0
Read Only, Returns 5555	0x20048	0	1	0	0	1	0	0	1
Read Only, Returns CPLD Code Version	0x20050	0	1	0	0	1	0	1	0
Read Only, Returns CAFÉ	0x20058	0	1	0	0	1	0	1	1
Write Only, Software Reset	0x20060	0	1	0	0	1	1	0	0
Read Only, Returns CPU and Personality IDs	0x20068	0	1	0	0	1	1	0	1

In described example LEDs on i.MX25 Debug board are used for indication of CSPI DMA transfer.
Basically next functions are used for turning on/off LEDs :

```
cspi_CPLD_write(0x1800,0x03C00027);  // all led Off

cspi_CPLD_write(0x1800,0x03FFFFE7);  //  all led On
```

1800 – is data with counter 0-18 (CPLD register address)
0x03C00027 – data with counter 19-45 (see tables below) (CPLD register data)

For register “LEDS” 0x2000 write 0x1800 in data with counter 0-18.
There is shift of 1 between the CPLD register software address and the physical line bits.
Since there are 8 LEDS only data with counter 32-39 (mask 0x03C03FE7) will take effect.

Fill data with patterns:

```
for (i = 0; i <= 0x4000; i++) *(unsigned int *)(SRC+i*4)=0xFFFFFFFF;
sdma_size=0x4000;

            for (i = 0; i <= 0x4000; i++) *(unsigned int *)(SRC+i*4)=0xFFFFFFFF;
                   for (i = 0; i <= sdma_size/16; i=i+4)
         {
                   *(unsigned int *)(SRC+i*4) = 0x1800;
                   *(unsigned int *)(SRC+(i+1)*4) = 0x03C00027 | (i&0xFC0);
                   *(unsigned int *)(SRC+(i+2)*4) = 0xFFFFFFFF;
                   *(unsigned int *)(SRC+(i+3)*4) = 0xFFFFFFFF;
         }

sdma_size=0x1000;
```

Start SDMA transfers:

```
while (1)
{
   sdma_start(sdma_size);
   sdma_size>>=1; if (sdma_size==0x80) sdma_size=0x1000;

                printf("BDCh0[0] = 0x%x  BDCh10[0] = 0x%x  \n", BDCh0[0],BDCh10[0]);

}
```

SDMA writes data from memory to CSPI1 à to CPLD LED register
Four left leds will blink.

SPI Write Operation

Counter	Memory Signal	Input	Output	Description
0	NA	0	NA	Designates operation as a write
1	CS5_B	1	NA	Memory map initinally not selected
2	A[16]	Address	NA	Input 17 bit address
3	A[15]	Address	NA	Input 17 bit address
4	A[14]	Address	NA	Input 17 bit address
5	A[13]	Address	NA	Input 17 bit address
6	A[12]	Address	NA	Input 17 bit address
7	A[11]	Address	NA	Input 17 bit address
8	A[10]	Address	NA	Input 17 bit address
9	A[9]	Address	NA	Input 17 bit address
10	A[8]	Address	NA	Input 17 bit address
11	A[7]	Address	NA	Input 17 bit address
12	A[6]	Address	NA	Input 17 bit address
13	A[5]	Address	NA	Input 17 bit address
14	A[4]	Address	NA	Input 17 bit address
15	A[3]	Address	NA	Input 17 bit address
16	A[2]	Address	NA	Input 17 bit address
17	A[1]	Address	NA	Input 17 bit address
18	A[0]	Address	NA	Input 17 bit address
19	CS5_B	0	NA	Select memory map
20	WR_B	1	NA	Write not enabled until data is input
21	WR_B	1	NA	Write not enabled until data is input
22	WR_B	1	NA	Write not enabled until data is input
23	WR_B	1	NA	Write not enabled until data is input
24	D[15]	Data	NA	Input 16 bit data
25	D[14]	Data	NA	Input 16 bit data
26	D[13]	Data	NA	Input 16 bit data
27	D[12]	Data	NA	Input 16 bit data
28	D[11]	Data	NA	Input 16 bit data
29	D[10]	Data	NA	Input 16 bit data
30	D[9]	Data	NA	Input 16 bit data
31	D[8]	Data	NA	Input 16 bit data
32	D[7]	Data	NA	Input 16 bit data
33	D[6]	Data	NA	Input 16 bit data
34	D[5]	Data	NA	Input 16 bit data
35	D[4]	Data	NA	Input 16 bit data
36	D[3]	Data	NA	Input 16 bit data
37	D[2]	Data	NA	Input 16 bit data
38	D[1]	Data	NA	Input 16 bit data
39	D[0]	Data	NA	Input 16 bit data
40	WR_B	1	NA	Write not enabled
41	WR_B	0	NA	Write enabled, load data to registers
42	WR_B	0	NA	Write enabled, load data to registers
43	WR_B	1	NA	Write not enabled, data loading done
44	WR_B	1	NA	Write not enabled, data loading done
45	CS5_B	1	NA	Deselect memory map, write is done

SPI Read Operation

Counter	Memory Signal	Input	Output	Description
0	NA	1	NA	Designates operation as a read
1	CS5_B	1	NA	Memory map initially not selected
2	A[16]	Address	NA	Input 17 bit address
3	A[15]	Address	NA	Input 17 bit address
4	A[14]	Address	NA	Input 17 bit address
5	A[13]	Address	NA	Input 17 bit address
6	A[12]	Address	NA	Input 17 bit address
7	A[11]	Address	NA	Input 17 bit address
8	A[10]	Address	NA	Input 17 bit address
9	A[9]	Address	NA	Input 17 bit address
10	A[8]	Address	NA	Input 17 bit address
11	A[7]	Address	NA	Input 17 bit address
12	A[6]	Address	NA	Input 17 bit address
13	A[5]	Address	NA	Input 17 bit address
14	A[4]	Address	NA	Input 17 bit address
15	A[3]	Address	NA	Input 17 bit address
16	A[2]	Address	NA	Input 17 bit address
17	A[1]	Address	NA	Input 17 bit address
18	A[0]	Address	NA	Input 17 bit address
19	CS5_B	0	NA	Select memory map
20	OE_B	1	NA	Write not enabled
21	OE_B	0	NA	Read enabled prior to data output
22	OE_B	0	NA	Read enabled prior to data output
23	OE_B	0	NA	Read enabled prior to data output
24	D[15]	NA	Data	output 16 bit data
25	D[14]	NA	Data	output 16 bit data
26	D[13]	NA	Data	output 16 bit data
27	D[12]	NA	Data	output 16 bit data
28	D[11]	NA	Data	output 16 bit data
29	D[10]	NA	Data	output 16 bit data
30	D[9]	NA	Data	output 16 bit data
31	D[8]	NA	Data	output 16 bit data
32	D[7]	NA	Data	output 16 bit data
33	D[6]	NA	Data	output 16 bit data
34	D[5]	NA	Data	output 16 bit data
35	D[4]	NA	Data	output 16 bit data
36	D[3]	NA	Data	output 16 bit data
37	D[2]	NA	Data	output 16 bit data
38	D[1]	NA	Data	output 16 bit data
39	D[0]	NA	Data	output 16 bit data
40	OE_B	0	NA	Read enabled after data output
41	OE_B	1	NA	Read not enabled, data output done
42	OE_B	1	NA	Read not enabled, data output done
43	OE_B	1	NA	Read not enabled, data output done
44	OE_B	1	NA	Read not enabled, data output done
45	CS5_B	1	NA	Deselect memory map, read is done