

Table of Contents

1. CORE -DETAILS.....	4
1. CortexA9 - Overview.....	4
2.CortexA9DualCore Processor configuration.....	4
3. L1 RAMs Configuration.....	5
4. Level 1 memory configuration in macrocell.....	5
5. L2 Cache Controller Configuration in Macrocell.....	5
6. Level 2 memory Configuration.....	6
7. Macrocell Functional Components.....	6
8. Different Voltage Domains of Macrocell.....	6
9. Internal Block Diagram of CortexA9DualCore Processor.....	8
2. CORESIGHT – JTAG.....	9
Buses.....	10
AMBA Trace Bus (ATB).....	10
AMBA 3 APB.....	11
Advanced High-performance Bus (AHB).....	11
AMBA Advanced eXtensible Interface (AXI).....	11
Control and access components.....	11
Embedded Cross Trigger.....	12
Trace sources.....	12
Instrumentation Trace Macrocell and System Trace Macrocell.....	12
Embedded Trace Macrocells (ETMs) and Program Trace Macrocells (PTMs).....	13
Trace links.....	13
Trace funnel.....	13
Replicator.....	13
Synchronous 1:1 ATB Bridge.....	13
Embedded Trace FIFO (ETF).....	13
Trace sink components.....	14
Embedded Trace Buffer (ETB) : On-Chip Trace Memory ETB.....	14
Trace Port Interface Unit (TPIU).....	14
JTAG.....	14
Test Setup / Connecting the DSTREAM unit.....	15
3. ARM DS-5 Development Studio.....	17
1.Creating a new platform configuration.....	17
2.Core Connection Establishment.....	19
3.Creating Project in Eclipse.....	20
4. To debug a project:.....	22
4.BRING -UP.....	23
1.UART CONTROLLER: UART (PL011).....	23
UART Controller.....	23
UART CONTROLLER: BARE METAL CODE.....	27
Test Procedure:.....	28
2. GPIO Controller - PL061.....	29
Overview.....	29
Features of the GPIO Controller.....	30
Block Diagram.....	31

Test Setup / Board Connections.....	31
Test Procedure.....	33
5.Static Memory Controller (PL353).....	33
1.Overview.....	33
2.Features of SMC (PL353):.....	34
3.Operating Description.....	35
4. NOR.....	36
5.NOR FLASH Commands.....	37
6. SRAM.....	40
6.SPI FLASH.....	43
SPI Modes.....	44
SPI NOR FLASH COMMANDS.....	45
7.SD/MMC (Secure Digital / Multi Media Card) Controller.....	48
Overview.....	48
Features.....	48
Block diagram.....	50
DWC Mobile Storage host controller Description.....	50
SD/MMC:.....	51
SD protocol:.....	53
8.LPDDR2/DDR3 Controller.....	59
Overview.....	59
Features of LPDDR2/DDR3 Controller.....	60
Block Diagram of LPDDR2/DDR3 Controller.....	64
Description of the Block Diagram.....	64
LPDDR2/DDR3 Memory Controller (uMCTL2).....	64
AXI Port Interface(XPI).....	65
Port Arbiter(PA).....	65
DDR Controller (DDRC).....	65
DDR Multi-PHY.....	65
DDR INTERFACE.....	66
Understanding RAM Timings.....	68
CAS Latency (CL) Impact on RAM Speed.....	70
RAS to CAS Delay (tRCD) Impact on RAM Speed.....	71
RAS Precharged (tRP) Impact on RAM Speed.....	72
9. MIPI DSI HOST CONTROLLER.....	73
Overview.....	73
Features of DSI:.....	74
Block Diagram of DSI.....	75
DesignWare Cores MIPI DSI Host Controller.....	76
Operational Model Overview.....	77
DPI Interface.....	78
Configuration Example.....	79
Burst Mode.....	81
Guidelines for Selecting the Burst or Non-Burst Mode.....	81
APB Slave Generic Interface.....	81
There are two signaling modes in the DSI physical layer:.....	86
Data lane states.....	87
Data lane operating modes.....	87
Control mode.....	87

High-speed transmission mode.....	87
Start-of-transmission (SoT) procedure.....	88
End-of-transmission (EoT) procedure.....	88
Escape mode.....	88
Ultra low-power state.....	90
Bidirectional lanes and bus turnaround procedure.....	90
Clock-lane power modes.....	91
Low-power mode.....	91
High-speed mode.....	91
Ultra-low-power state (ULPS).....	93
10.DSI protocol.....	94
Packet structure.....	94
Long packet.....	95
Short packet.....	96
Data identifier byte.....	96
Packet transmission modes.....	97
Host to display data types.....	98
Shut down and color modes.....	100
Synchronization events.....	100
Packed pixel streams.....	100
Command mode data types.....	100
11. MIPI CSI-2 Host Controller.....	102
Overview.....	102
Features of MIPI CSI-2 Host Controller.....	103
Block Diagram:.....	103
D - PHY.....	105
MIPI-CSI2 and D-PHY configuration example sequence.....	105
MIPI D-PHY clock.....	107
D-PHY registers.....	108

1. CORE -DETAILS

1. CortexA9 - Overview

Cortex-A9 MPCore speed-optimized macrocell (referred as macrocell) is developed by TSMC. There are two cores in macrocell, so called CortexA9 Dual Core Processor. The two CortexA9 processors use the same individual hardware configuration. Each CortexA9 processor incorporates an **integer unit that implements the ARM architecture v7-A**. Each processor in DualCore supports the **ARM, Thumb instruction sets and Jazelle technology to enable direct execution of Java byte codes and a range of Advanced SIMD instructions**, also **each processor supports security extensions and multiprocessing extensions**.

2.CortexA9DualCore Processor configuration

Configuration option	Implementation
Cortex-A9 processors	Two
Instruction cache size per Cortex-A9 processor	32KB
Data cache size per Cortex-A9 processor	32KB
TLB size per Cortex-A9 processor	128 entries
BTAC size per Cortex-A9 processor	1024 entries
GHB size	2048 entries
Instruction micro TLB per CortexA9 Processor	32 entries
Media Processing Engine with NEON technology per Cortex-A9 processor	Included
FPU per Cortex-A9 processor	Not Included
Preload Engine per Cortex-A9 processor	Not Included
Jazelle DBX extension per Cortex-A9 processor: Full support	Included
Program Trace Macrocell (PTM) interface per Cortex-A9 processor	Included
Power off and dormant mode wrappers	Not Included*
Support for parity error detection	Not Included
ARM_BIST	Included
AXI Master ports	Two Master ports
AXI ACP Slave port	One, Included
Shared Peripheral Interrupts (SPIs)	128

3. L1 RAMs Configuration

The Internal RAMs present in each CortexA9 of DualCore Processor are described below.

- Instruction Cache Data RAM
- Data Cache Data RAM
- Instruction Cache Tag RAM
- Data Cache Tag RAM
- Instruction Cache Valid RAM
- Data Cache Valid RAM
- Data Cache Dirty RAM
- BTAC RAM
- GHB RAM
- TLB RAM
- Instruction Micro TLB RAM

4. Level 1 memory configuration in macrocell

The total instances are doubled as macrocell is configured for two processors.

MEMORY	SIZE
Instruction Cache(I-Cache)	32 Kbytes (8 instances of 1024x32-bit) word-write
Instruction Tag(I-Tag)	4 instances of 256 x 22-bit word-write
Data Cache (D-Cache)	32 Kbytes (8 instances of 1024x32-bit) byte-write
Data Tag(D-Tag)	4 instances of 256x25-bit bit-write
Data Cache Outer	1 instance of 256x12-bit bit-write
SCU tag RAM	4 instances of 256 x 23-bit bit-write
TLB RAM 128-entry	2 instances of 64x32-bit bit-write and, 2 instances of 64x29-bit bit-write
BTAC RAM target array	2 instances of 256x32-bit word-write
BTAC RAM control array	2 instances of 256x28 bit word-write
Global History Buffer	4 instances of 512x 4-bit bit-write

5. L2 Cache Controller Configuration in Macrocell

Configuration option	Implementation
Number of master ports	2
Number of slave ports	2
Support for Lockdown by master	Included
Support for lockdown by line	Included
Support for Address Filtering	Included

Parity support	Included
Default Latencies	3'b111
Data_banking support	Not Included

6. Level 2 memory Configuration

L2 cache is unified, physically indexed and physically tagged Cache.

Configuration option	Implementation
Cache Size	512KB
No of Ways	8way (assumed)
Way Size	64KB(assumed)
16 ways	This to be configured if 16 ways is supported. Not supported.

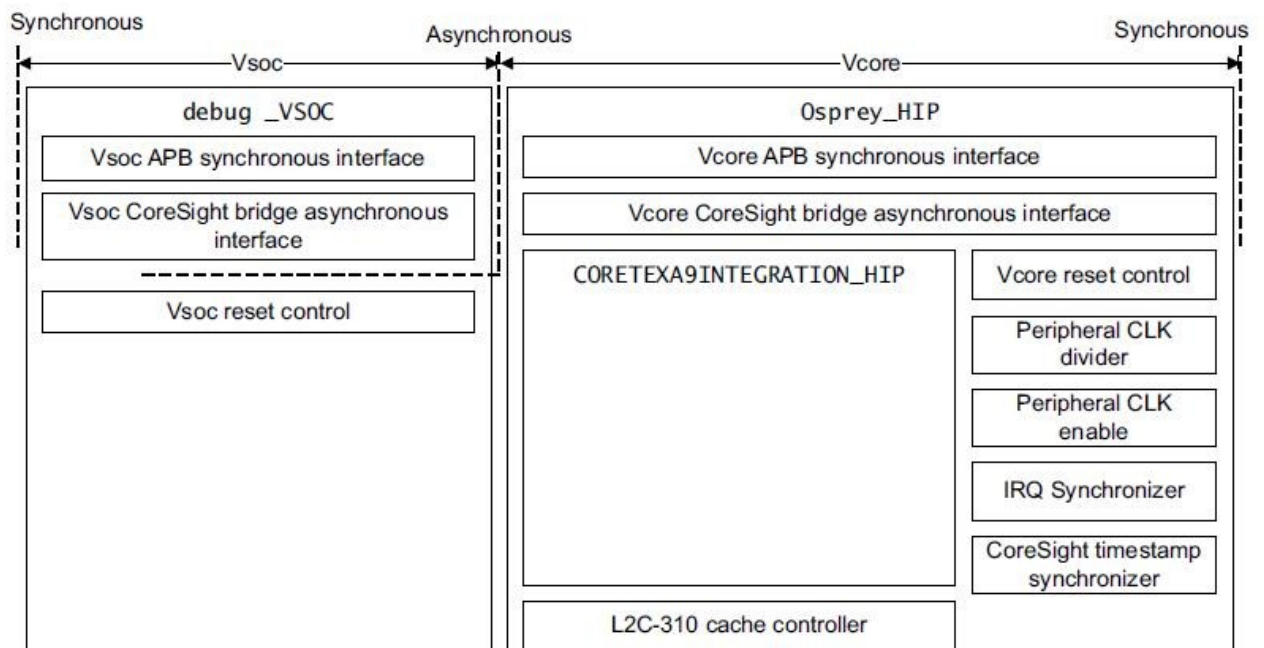
7. Macrocell Functional Components

- Two Cortex-A9 processors, CPU0 and CPU1, **each containing a private timer and watchdog unit, 32KB Instruction cache, 32KB Data cache and 128 TLB entries**
- Two media processing engines, NEON0 and NEON1, that support NEON technology and floating-point operations
- **A Snoop Control Unit (SCU) with an Accelerator Coherency Port (ACP) for coherent memory transfers to ensure coherency within the cluster**
- A global timer
- An integrated Interrupt Controller that is an implementation of the Generic Interrupt Controller architecture with 128 interrupt lines.
- An integrated L1 ARM MBIST Controller.
- Level 2 (L2) cache controller, PL310 has:

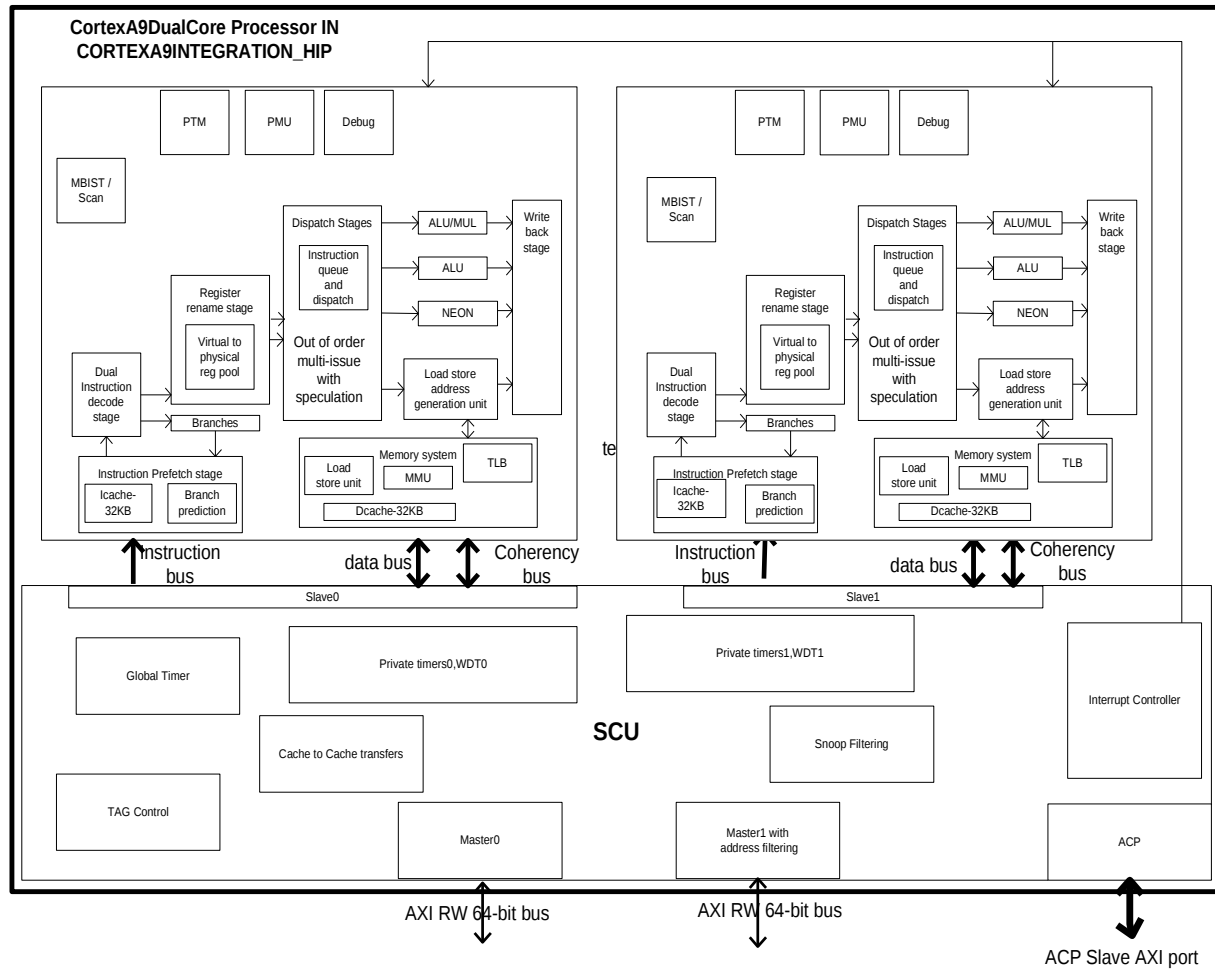
8. Different Voltage Domains of Macrocell

The macrocell has two Voltage Domains, Vcore and Vsoc:

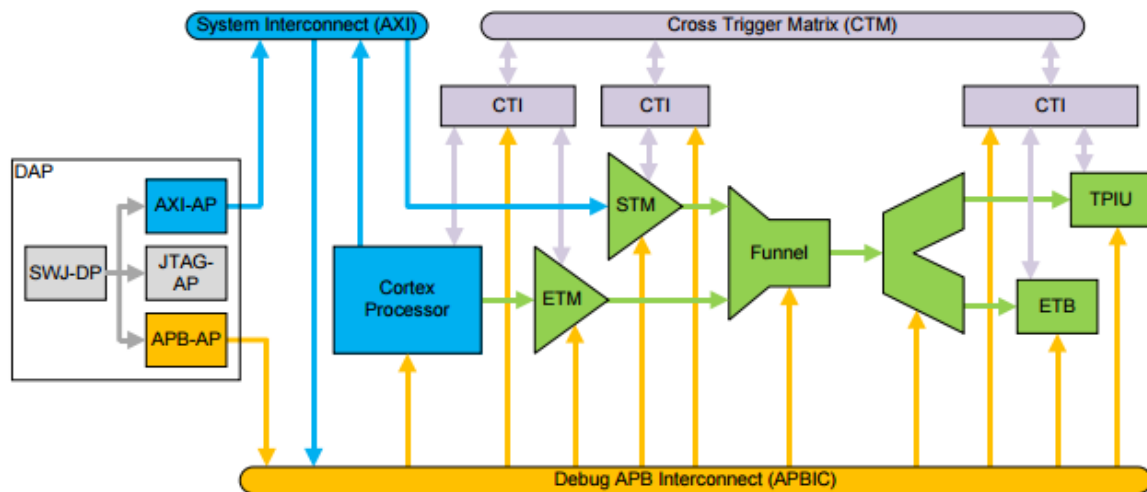
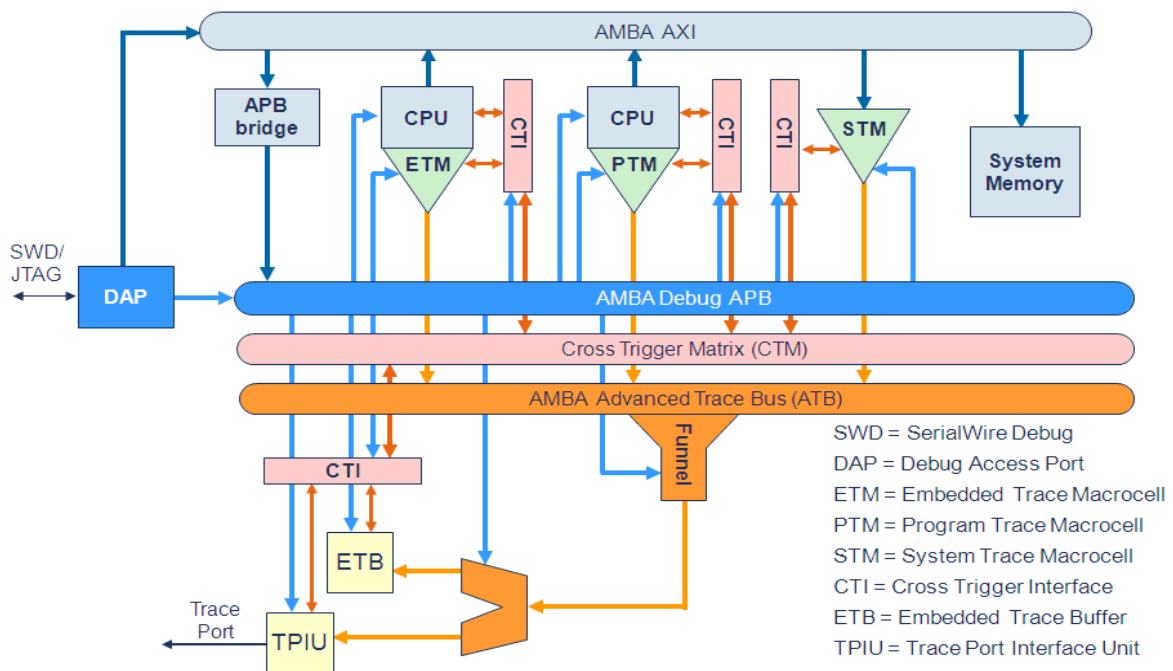
- The Vcore domain can be a variable voltage domain if integration supports Dynamic Voltage and Frequency Scaling (DVFS). It powers the majority of logic including the CPUs, NEON, SCU, and Level 2 Cache Controller L2C-310.
- The Vsoc domain is a fixed voltage domain and is used to power the debug APB and ATB interfaces in the macrocell.
- When Vcore and Vsoc are driven at different voltages, level shifters are required on all macrocell inputs and outputs, except for the pins that only connect to the Vsoc domain in the macrocell.



9. Internal Block Diagram of CortexA9DualCore Processor



2. CORESIGHT – JTAG



Buses

The CoreSight systems use the following bus protocols to connect components together, and to enable integration in a SoC:

- *AMBA Trace Bus (ATB)*
- *AMBA 3 Advanced Peripheral Bus (AMBA 3 APB)*
- *Advanced High-performance Bus (AHB)*
- *AMBA Advanced eXtensible Interface (AXI).*

AMBA Trace Bus (ATB)

The ATB transfers trace data through the CoreSight infrastructure in a SoC. Trace sources are ATB masters, and sinks are ATB slaves. Link components provide both master and slave interfaces.

The ATB protocol supports:

- Stalling of trace sources to enable the CoreSight components to funnel and combine sources into a single trace stream.
- Association of trace data with the generating source using trace source IDs. A CoreSight system can trace up to 111 different items at any one time.
- Capture and transfer of multiple byte bus widths, currently to 32-bits.
- A flushing mechanism to force historic trace to drain from any sources, links, or sinks up to the point that the request was initiated.

For more information about ATB, see the *AMBA ATB Protocol Specification*.

AMBA 3 APB

CoreSight supports the AMBA 3 APB protocol to enable transfer extension using wait states.

The Debug APB bus uses the AMBA 3 APB protocol within a CoreSight system. The Debug APB is a bus dedicated to the connection of debug and trace components in a CoreSight-compliant SoC. All CoreSight components are configured and accessed over this bus through the APB-Mux in the DAP.

For more information about AMBA 3 APB, see the *AMBA 3 APB Protocol Specification*.

Advanced High-performance Bus (AHB)

CoreSight supports access to a system bus infrastructure using the *AHB Access Port* (AHB-AP) in the DAP. The AHB-AP provides an AHB master port for direct access to system memory.

CoreSight also supports AHB bus tracing using an *AHB Trace Macrocell* (HTM) that provides non-invasive debug visibility to any bus transactions on AHB connections.

For more information on AHB, see the *AMBA Specification*.

AMBA Advanced eXtensible Interface (AXI)

CoreSight supports the use of AXI in the system interconnect. Direct access to the AXI system can be provided through a Cortex core as an AXI bus master, or through the use of an AHB to AXI bridge on the AHB Access Port in the DAP. CoreSight also supports trace generation from bus masters on the AXI through the use of the STM that converts stimulus writes to the device into a trace data stream. For more information on AXI,

Control and access components

Control and access components configure, provide access to, and control debug logic and the generation of trace. They do not generate trace, or process the trace data. The CoreSight control and

access components are:

- the *Debug Access Port* (DAP)
- the *Embedded Cross Trigger* (ECT) that includes the *Cross Trigger Matrix* (CTM) and the *Cross Trigger Interface* (CTI).

Embedded Cross Trigger

The ECT is a modular component that supports the interaction and synchronization of multiple triggering events within a SoC.

The ECT consists of the following types of module:

- A CTI. The CTI provides the interface between a component or subsystem and the *Cross Trigger Matrix* (CTM). The system requires a CTI for each subsystem that supports cross triggering.
- A CTM. The CTM combines the trigger requests generated from CTIs and broadcasts them to all CTIs as channel triggers. This enables subsystems to interact, cross trigger, with one another. You can connect CTMs together to increase the number of CTIs.

Trace sources

Sources generate trace data and provide master ports to the AMBA Trace Bus. Depending on the licensed CoreSight components, the following trace sources can be provided:

- CoreSight ETMs and PTMs for CPU trace:
- the *Instrumentation Trace Macrocell* (ITM)
- the *System Trace Macrocell* (STM).

Instrumentation Trace Macrocell and System Trace Macrocell

The *Instrumentation Trace Macrocell* (ITM) and *System Trace Macrocell* (STM) are application-driven trace sources that generate trace based on software written to the program interface. The ITM presents 32 APB registers, and the STM provides a set of 64K AXI registers that, on a write transaction, generate corresponding trace that indicates the register and value written.

Embedded Trace Macrocells (ETMs) and Program Trace Macrocells (PTMs)

The ETMs provide processor-driven trace through an ATB-compliant trace port. You can configure the ETM through the CoreSight APB programming interface. ETM9CS and ETM11CS both provide an asynchronous ATB master port that transfers trace data onto the CoreSight infrastructure.

Trace links

Links provide connection, triggering, and flow of traced data. The following sections describe the links:

- *Trace funnel*
- *Replicator*
- *Synchronous 1:1 ATB Bridge*
- *Embedded Trace FIFO (ETF).*

Trace funnel

The Trace funnel combines up to eight trace sources on a single funnel. A static arbitration scheme selects the input trace stream to pass at any instant. The static arbitration permits reorganization of the slave port priorities between trace sessions. You can chain funnels together, with the ATB output from one funnel connected to an ATB input port of another. This enables you to both increase the number of inputs, and to connect independent systems together.

Replicator

The Replicator enables you to wire two trace sinks together and operate them on the same incoming trace stream. The input trace stream is output on two ATB ports that can then operate independently.

Synchronous 1:1 ATB Bridge

The Synchronous ATB Bridge provides a register slice that enables timing closure through the addition of a pipeline stage. It also provides a unidirectional link between two synchronous ATB domains. The bridge is 1:1 because both the input and output interfaces exist in the same clock domain. Because the bridge is a single register slice over the ATB interface, it temporarily holds one cycle of trace data within the register bank.

Embedded Trace FIFO (ETF)

The Embedded Trace FIFO is a trace buffer that uses a dedicated SRAM as either a circular capture

buffer, or as a FIFO. The trace stream is captured by an ATB input that can then be output over an ATB output or the Debug APB interface.

The ETF is a configuration option of the *Trace Memory Controller* (TMC).

Trace sink components

CoreSight SoC contains the following components for receiving debug information and looking after its transmission onto the main debug infrastructure. The trace sink components are:

- *Trace Port Interface Unit.*
- *Embedded Trace Buffer.*

Embedded Trace Buffer (ETB) : On-Chip Trace Memory ETB

A logic block that extends the information capture functionality of a trace macrocell.

on-chip trace memory known as the CoreSight **Embedded Trace Buffer (ETB)**. However, its capacity is much smaller than an external trace tool – normally only 2 to 8 KB. If the trace data is saved in the ETB and then read over the JTAG interface,

The ETB accepts trace data from CoreSight trace source components through an *AMBA Trace Bus* (ATB).

Trace Port Interface Unit (TPIU)

Drains trace data and acts as a bridge between the on-chip trace data and the data stream captured by a trace port analyzer.

The TPIU acts as a bridge between the on-chip trace data, with separate IDs, to a data stream, encapsulating IDs where required, that is then captured by a *Trace Port Analyzer* (TPA).

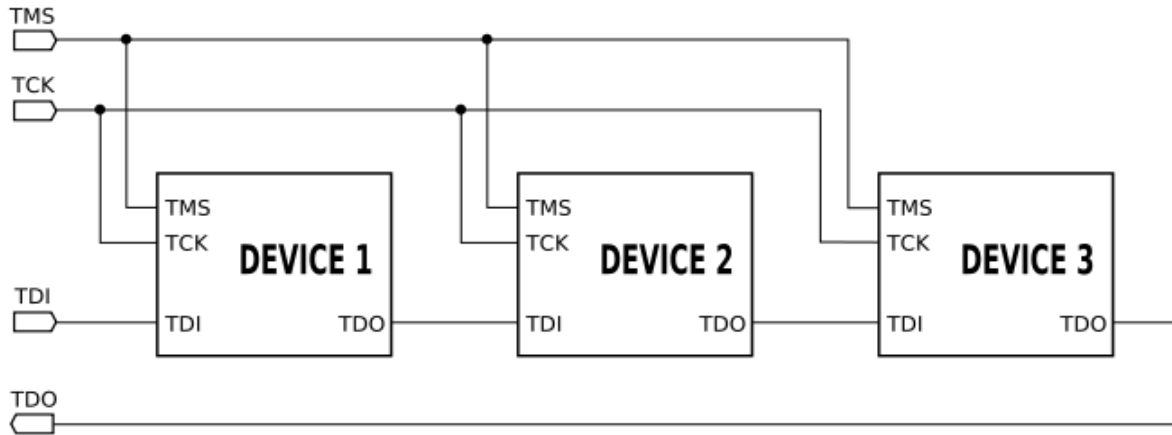
JTAG

JTAG (named after the **Joint Test Action Group** which codified it) is an industry standard for verifying designs and testing printed circuit boards after manufacture.

The connector pins are:

1. **TDI** (Test Data In)
2. **TDO** (Test Data Out)

3. **TCK** (Test Clock)
4. **TMS** (Test Mode Select)
5. **TRST** (Test Reset) optional.



The maximum operating frequency of TCK varies depending on all chips in the chain (the lowest speed must be used), **but it is typically 10-100 MHz**

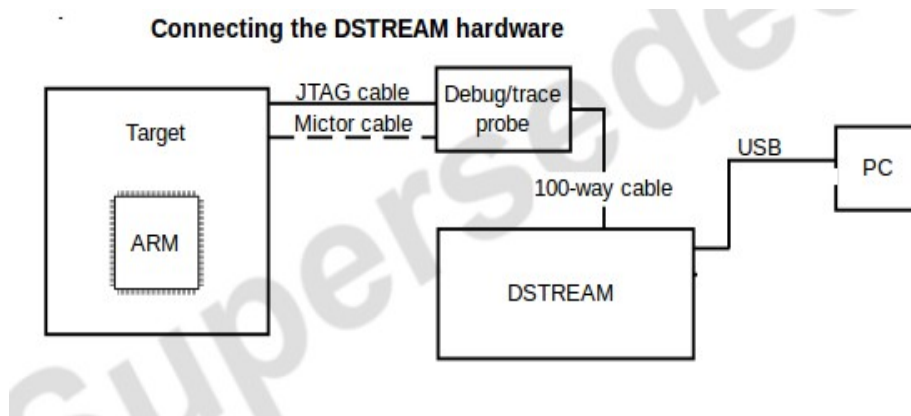
One bit of data is transferred in from TDI, and out to TDO per TCK rising clock edge. Different instructions can be loaded.

Test Setup / Connecting the DSTREAM unit

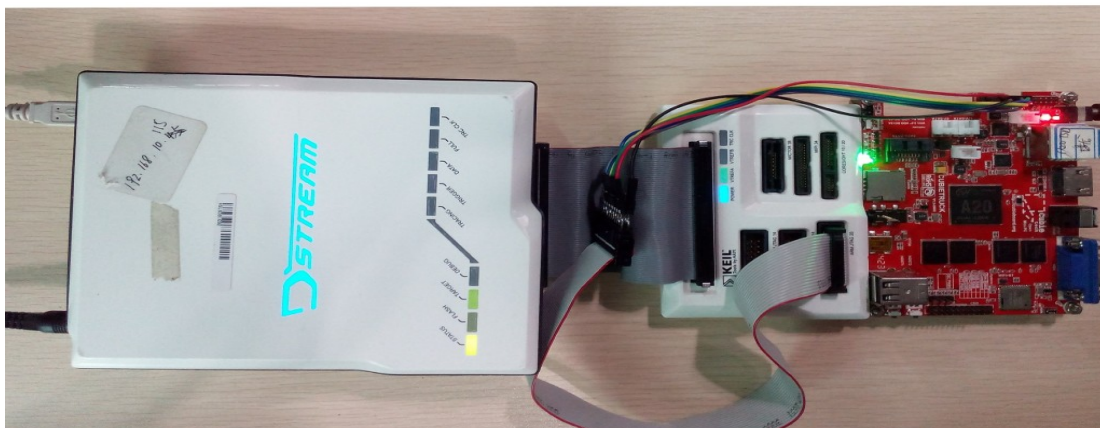
To connect the DSTREAM unit to host computer and to the target hardware, carry out the following:

1. Have the Host PC running Ubuntu 14.04 LTS [64-bit], with DS-5 installed, having the trial 60 day license
2. Connect the host computer to the DSTREAM unit as shown in the following figure, using the USB port
 - If you are connecting using the USB port, connect one end of the supplied USB cable to a USB port on the host computer, and the other end of the cable to the USB port on the DSTREAM unit.

Note: The USB drivers are installed with the debug host software.



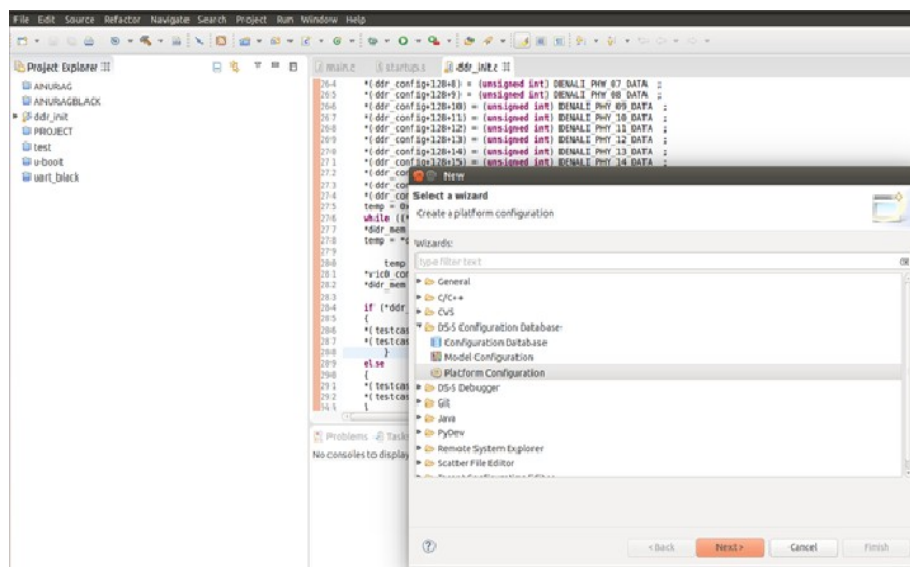
3. Connect the DSTREAM unit to the target hardware, using the appropriate debug or trace cables:
 - Connect one end of the supplied 100-way cable to the DSTREAM unit, and connect the other end of the cable to the probe unit.
 - Connect the target hardware to the probe using the appropriate cables and connector.
 - ARM JTAG 20 ,This is the most commonly-used debug connector standard for ARM-based target boards
4. Power-up the target hardware [Connect 12V DC Input] and power-up the DSTREAM unit.
5. Typical Test Setup looks like this



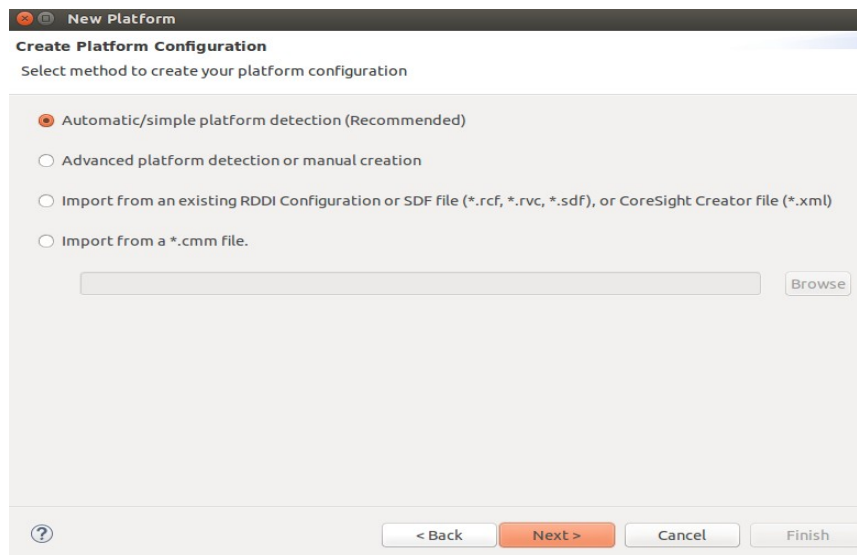
3. ARM DS-5 Development Studio

1. Creating a new platform configuration

- You can use the Platform Configuration Editor (PCE) within DS-5 to create debug configurations for new platforms. Creating a new platform configuration in DS-5 requires a new Platform Configuration project in Eclipse
- From the main menu in DS-5, select **File > New > Other** to open the new project dialog.
- Select **DS-5 Configuration Database > Platform Configuration** and then click **Next**.

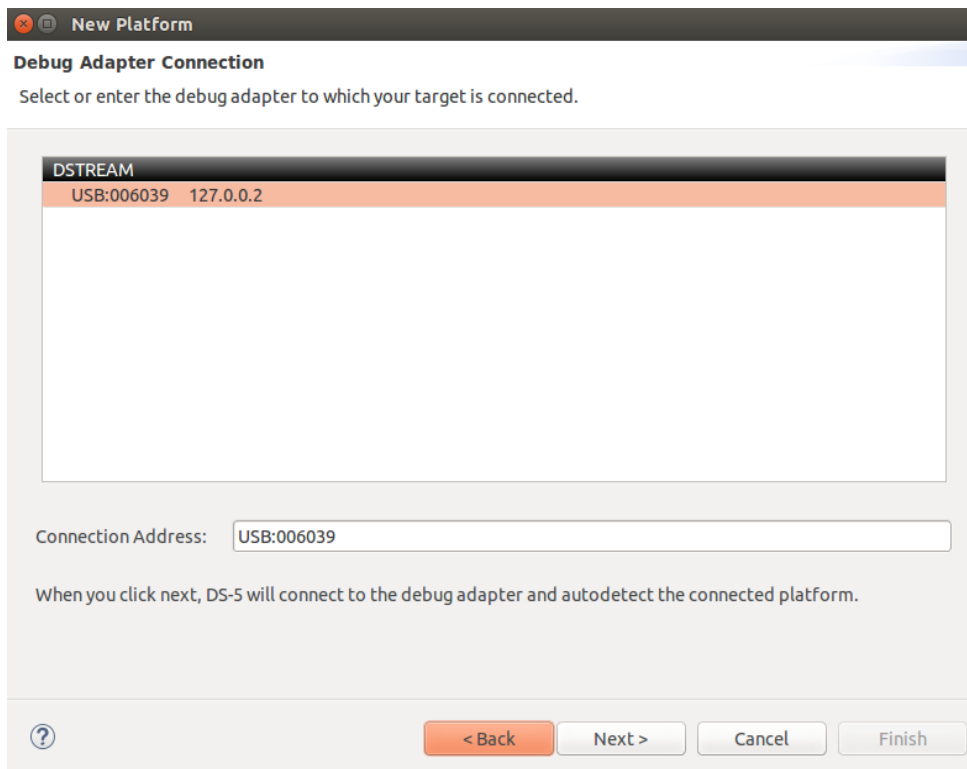


- This shows the **Create Platform Configuration dialog box**. Select the required method to create the configuration for your platform and click **Next**.
- Select **DS-5 Configuration Database > Platform Configuration** and then click **Next**



- Automatic/simple platform detection

This is the recommended option. DS-5 automatically detects the devices that are present on your platform. It then provides you the opportunity to add more devices if needed and to specify how the devices are interconnected.



1. Once we have selected the DSTREAM unit, DS-5 will connect to the SoC and try to read all the information it needs for debug and trace

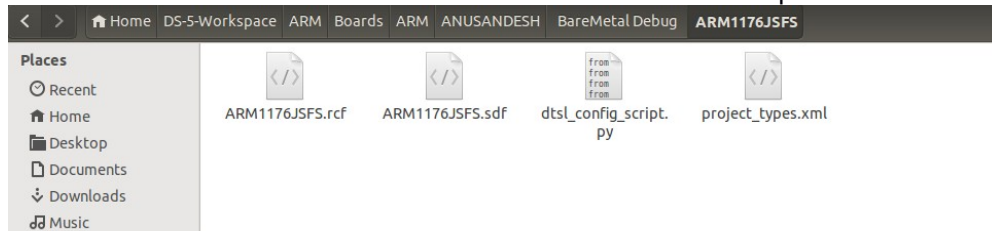
Note: Manual Platform Configuration

We can manually create the platform configuration if DS-5 does not automatically detect the platform configuration. DS-5 uses all the information that it reads from the platform to create a custom platform configuration

2. Core Connection Establishment

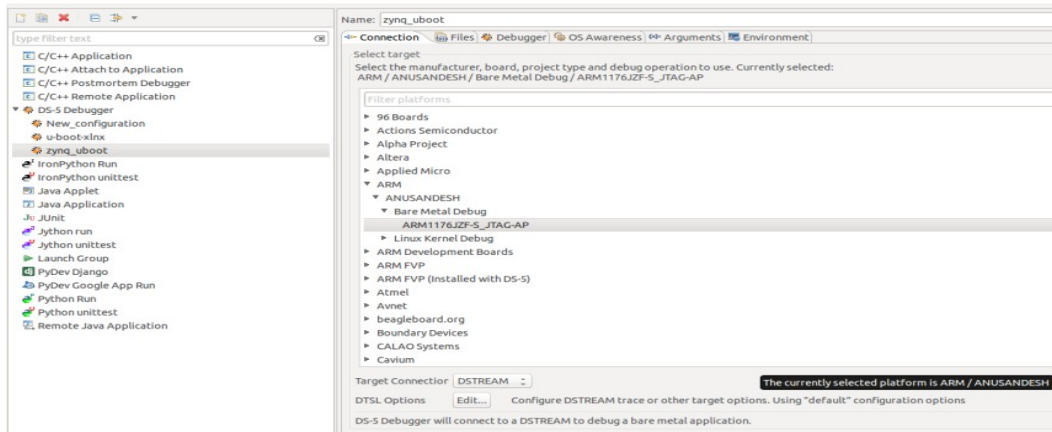
- **Procedure:**
 - From the main menu, select **Window--> Open Perspective--> DS-5 Debug**.
 - From the main menu, select **Run--> Debug Configurations** to open the Debug Configuration dialog.
 - In the configuration tree, **select DS-5 Debugger and then click New** to create a new configuration.
 - Enter a **suitable name for the new configuration**, in the Name field. For example, zynq_uboot.
 - Use the **Connection tab to specify the target and connection settings**.
 - In the Select target area of the dialog, browse and select the platform you require. For example, to connect to the ARM1176JZF-S , Browse and select **home--> DS-5-Workspace-->Boards-->ARM --> ANUSANDESH--> BareMetal Debug--> ARM1176JZF-S_JTAG-AP**.
 - Select your debug hardware unit in the Target Connection list. For example, DSTREAM.

The location of the folder and the files are shown in below picture



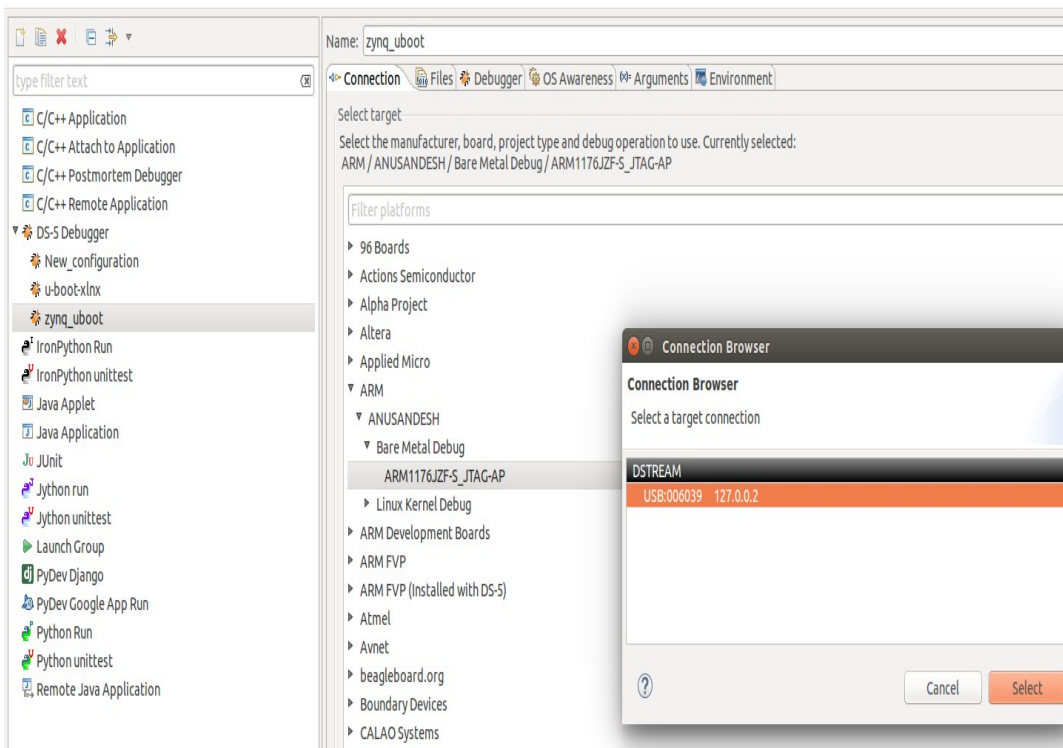
- To start debugging using D-Stream, we need to make sure that D-Stream connection has been made with the DS-5. To do this there is a Debug Configuration tab click on that one and select Connection from that. After that a dialogue box will appear. In that select the bare metal debug. Then we can see the USB id of the D-Stream connected. Now the DS-5 is connected to the D-Stream. It's shown in below diagrams.

Create, manage, and run configurations



Debug Configurations

Create, manage, and run configurations

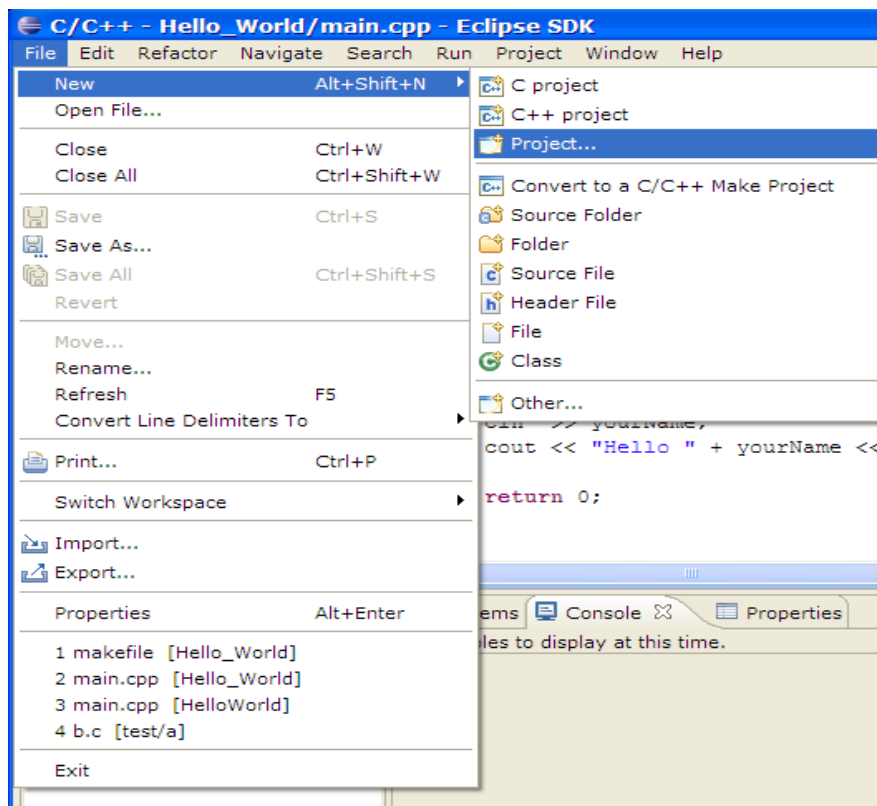


3. Creating Project in Eclipse

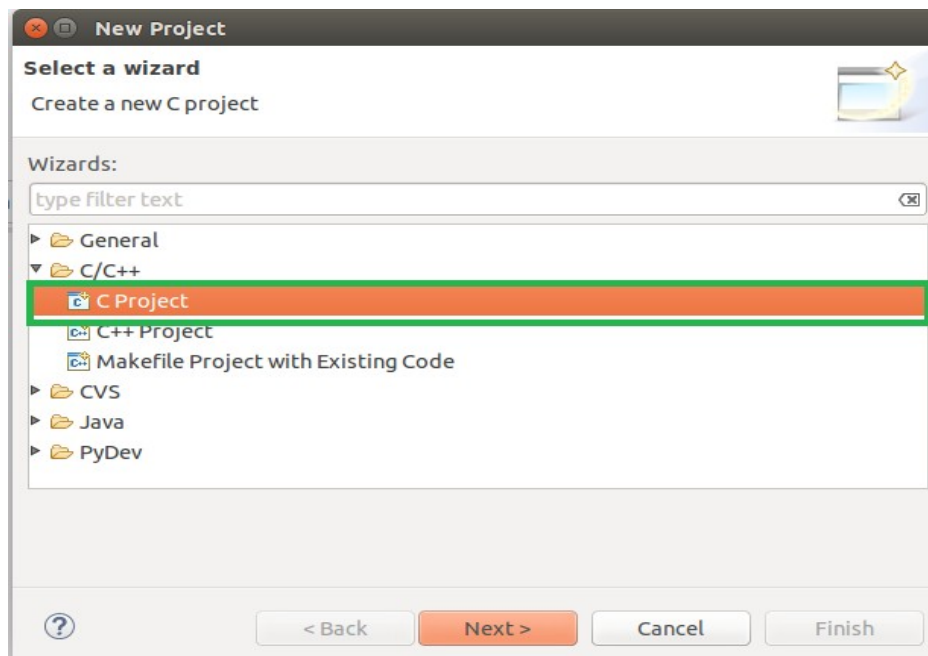
You can create a standard make or managed make C or C++ project.

To create a project:

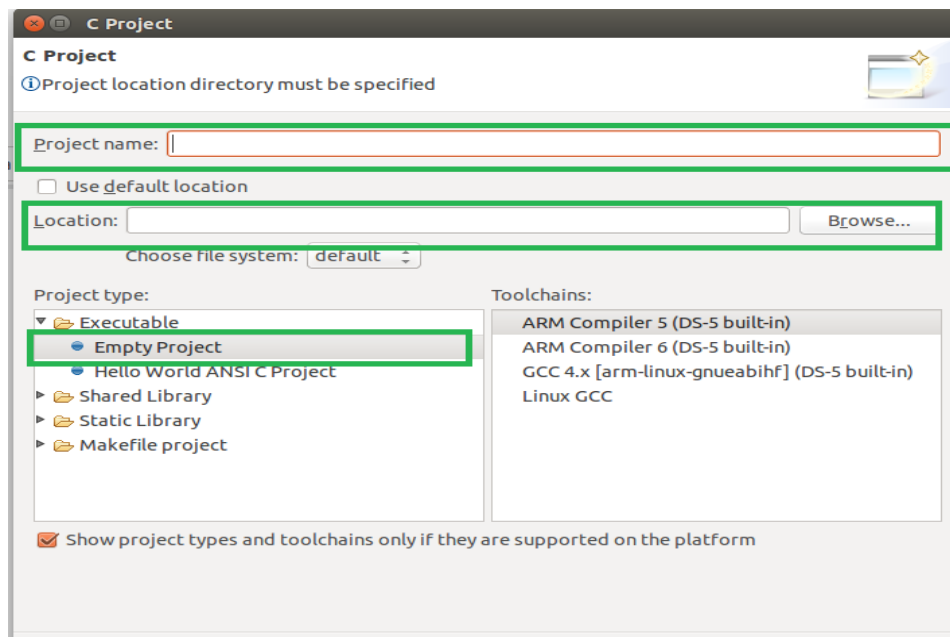
1. Click **File > New > Project**.



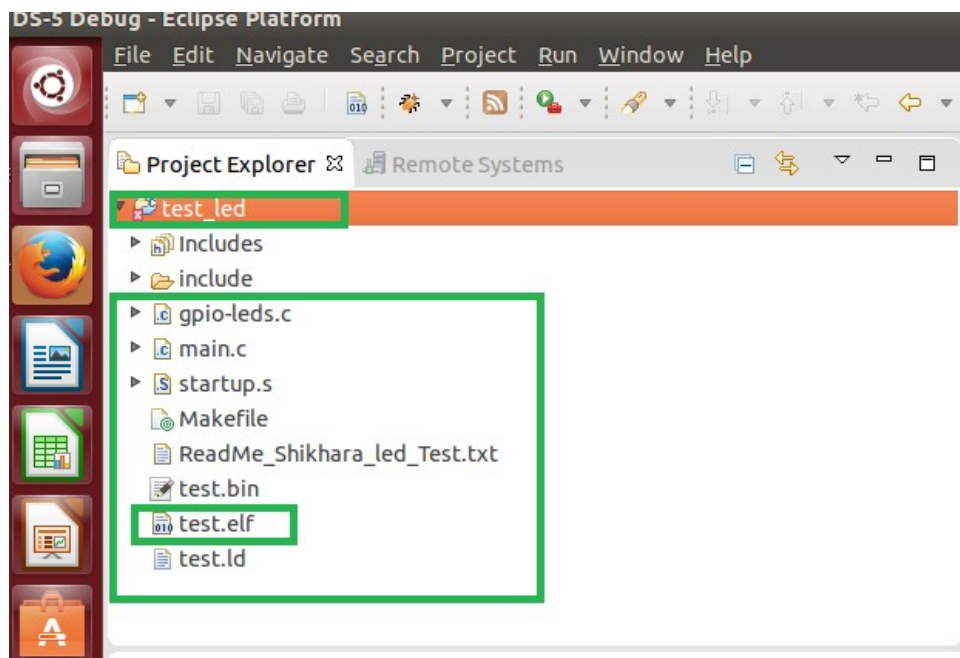
2. Select **C Project** [Select the type of project to create. Expand the **C/C++** folder and select **C Project**]



3. In the **Project name** field, type test_led
 Leave the **Use Default Location** option selected.
Empty Project provides a single source project folder that contains files

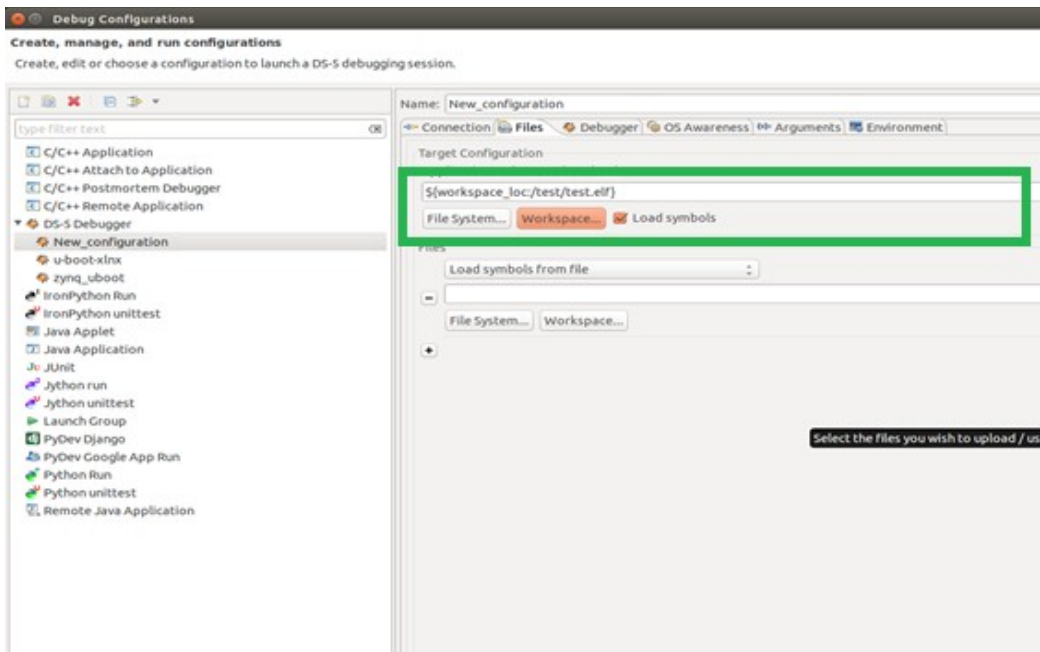


4. Project Explorer, will show the **project name**, with source and executable [elf] file



4. To debug a project:

- Click the **Run > Debug Configurations...** menu option. The **Debug Configurations** dialog opens.
- From the new configuration , workspace , select the elf file location
- From the new configuration ,Debugger, select entry point



4.BRING -UP

1.UART CONTROLLER: UART (PL011)

UART Controller

Universal Asynchronous Receiver/Transmitter, is a type of "**asynchronous receiver/transmitter**", a piece of computer hardware that translates data between parallel and serial forms. The UART is an AMBA slave module that connects to the Advanced Peripheral Bus (APB). The UART includes an Infrared Data Association (IrDA) Serial InfraRed (SIR) protocol ENcoder/DECoder (ENDEC).

Features of PrimeCell UART Controller:

The UART provides:

- Compliance to the AMBA Specification (Rev 2.0) onwards for easy integration into SoC implementation.
- Programmable use of UART or IrDA SIR input/output.
- **Separate 32×8 transmit and 32×12 receive *First-In, First-Out* (FIFO) memory buffers to reduce CPU interrupts.**

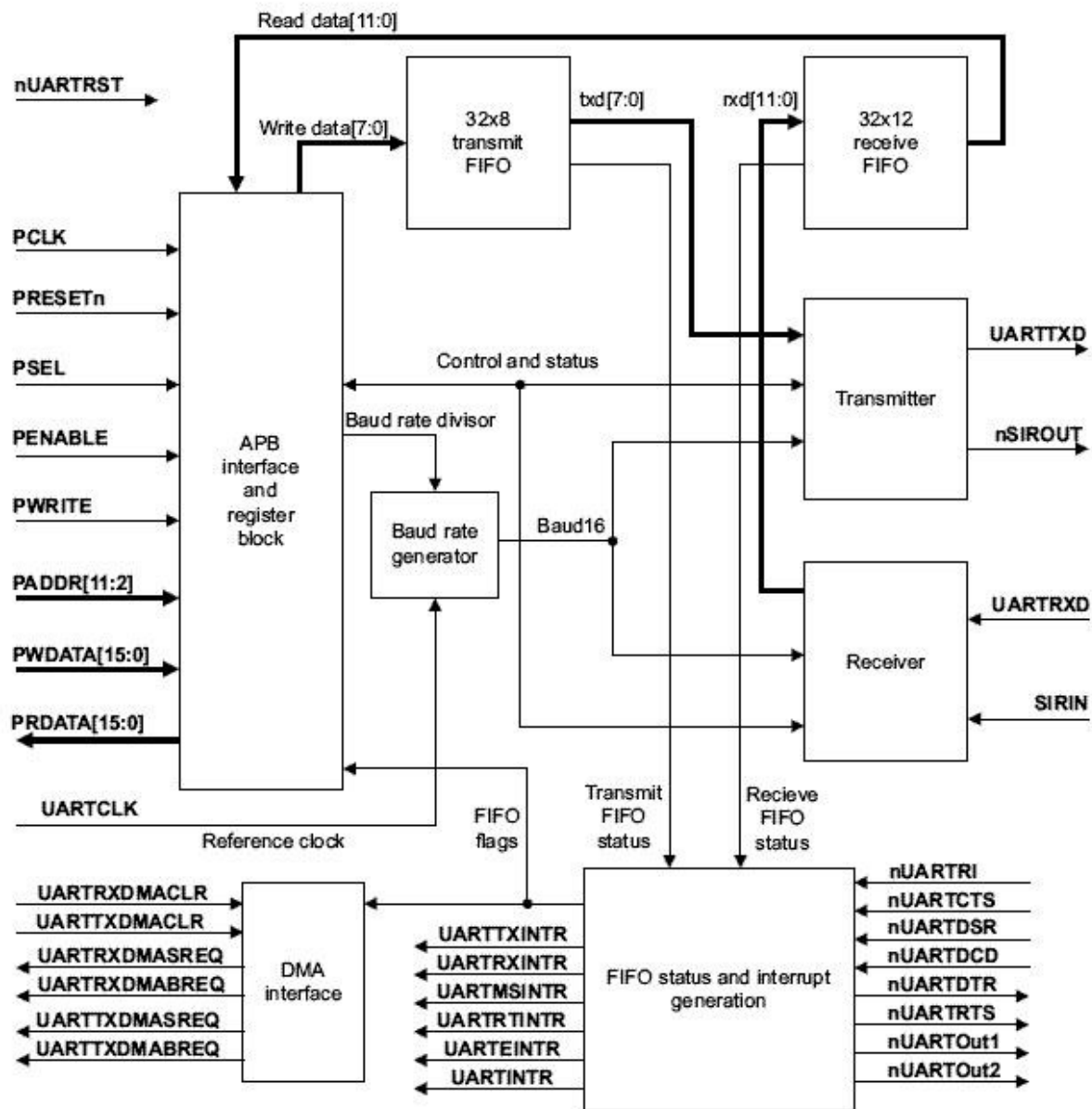
- **Programmable FIFO disabling for 1-byte depth.**
- **Programmable baud rate generator.**
- Standard asynchronous communication bits (start, stop and parity). These are added prior to transmission and removed on reception.
- Independent masking of transmit FIFO, receive FIFO, receive timeout, modem status, and error condition interrupts.
- Support for *Direct Memory Access* (DMA).
- False start bit detection.
- Line break generation and detection.
- Support of the modem control functions CTS, DCD, DSR, RTS, DTR, and RI.
- Programmable hardware flow control.

Fully-programmable serial interface characteristics:

- **data can be 5, 6, 7, or 8 bits**
- **even, odd, stick, or no-parity bit generation and detection**
- **1 or 2 stop bit generation**
- **baud rate generation, dc up to UARTCLK/16**

IrDA SIR ENDEC block providing:

- programmable use of IrDA SIR or UART input/output
- support of IrDA SIR ENDEC functions for data rates up to 115200 bps half-duplex
- programmable division of the **UARTCLK** reference clock to generate the appropriate bit duration for low-power IrDA mode.
- Identification registers that uniquely identify the UART. These can be used by an operating system to automatically configure itself.



UARTC Block Diagram

AMBA APB interface:

The AMBA APB interface generates read and write decodes for accesses to status/control registers, and the transmit and receive FIFOs.

Register block:

The register block stores data written, or to be read across the AMBA APB interface.

Baud rate generator

The baud rate generator contains free-running counters that generate the internal \tilde{A} —16 clocks, Baud16 and IrLPBaud16 signals. Baud16 provides timing information for UART transmit and receive control. Baud16 is a stream of pulses with a width of one UARTCLK clock period and a frequency of 16 times the baud rate. IrLPBaud16 provides timing information to generate the pulse width of the IrDA encoded transmit bit stream when in low-power IrDA mode.

Transmit FIFO

The transmit FIFO is an 8-bit wide, 32 location deep, FIFO memory buffer. CPU data written across the APB interface is stored in the FIFO until read out by the transmit logic. The transmit FIFO can be disabled to act like a one-byte holding register.

Receive FIFO

The receive FIFO is a 12-bit wide, 32 location deep, FIFO memory buffer. Received data and corresponding error bits, are stored in the receive FIFO by the receive logic until read out by the CPU across the APB interface. The receive FIFO can be disabled to act like a one-byte holding register.

Transmit logic

The transmit logic performs parallel-to-serial conversion on the data read from the transmit FIFO. Control logic outputs the serial bit stream beginning with a start bit, data bits with the Least Significant Bit (LSB) first, followed by the parity bit, and then the stop bits according to the programmed configuration in control registers.

Receive logic

The receive logic performs serial-to-parallel conversion on the received bit stream after a valid start pulse has been detected. Overrun, parity, frame error checking, and line break detection are also performed, and their status accompanies the data that is written to the receive FIFO.

Interrupt generation logic

Individual maskable active HIGH interrupts are generated by the UART. A combined interrupt output is also generated as an OR function of the individual interrupt requests. Single combined interrupt can be used with a system interrupt controller that provides another level of masking on a per-peripheral basis.

DMA interface

The UART provides an interface to connect to the DMA controller as UART DMA interface

UART CONTROLLER: BARE METAL CODE

Based on Baud rate calculate : Control Values

```
#include<stdio.h>
#define CONFIG_PL011_CLOCK    24000000

#define baudrate 115200
int main()
{
    unsigned int temp;
    unsigned int divider;
    unsigned int remainder;
    unsigned int fraction;

    temp = 16 * baudrate;
    divider = CONFIG_PL011_CLOCK / temp;
    remainder = CONFIG_PL011_CLOCK % temp;
    temp = (8 * remainder) / baudrate;
    fraction = (temp >> 1) + (temp & 1);

    printf("value of divider decimal=%d hexa=0x%x\n",divider,divider);
    printf("value of fraction decimal=%d hexa=0x%x\n",fraction,fraction);
    return 0;
}

enum
{
    // The offsets for reach register for the UART.
    UART0_DR    = (UART0_BASE + 0x00),
    UART0_RSRECR = (UART0_BASE + 0x04),
    UART0_FR    = (UART0_BASE + 0x18),
    UART0_ILPR  = (UART0_BASE + 0x20),
    UART0_IBRD  = (UART0_BASE + 0x24),
    UART0_FBRD  = (UART0_BASE + 0x28),
    UART0_LCRH  = (UART0_BASE + 0x2C),
    UART0_CR    = (UART0_BASE + 0x30),
    UART0_IFLS  = (UART0_BASE + 0x34),
    UART0_IMSC  = (UART0_BASE + 0x38),
    UART0_RIS   = (UART0_BASE + 0x3C),
    UART0_MIS   = (UART0_BASE + 0x40),
    UART0_ICR   = (UART0_BASE + 0x44),
    UART0_DMACR = (UART0_BASE + 0x48),
    UART0_ITCR  = (UART0_BASE + 0x80),
    UART0_ITIP  = (UART0_BASE + 0x84),
    UART0_ITOP  = (UART0_BASE + 0x88),
    UART0_TDR   = (UART0_BASE + 0x8C),
};
```

```

void uart_init(void)
{
    // Disable UART0.
    writel(0x00000000,UART0_CR);

    // Clear pending interrupts.
    writel(0x7FF,UART0_ICR);

    //refer to baudrate_calc.c code to generate divisor and fraction value,24000000
    writel(0x0d,UART0_IBRD); //write divider value in hexa
    writel(0x01,UART0_FBRD); //write fraction value in hexa

    // Enable FIFO & 8 bit data transmissio (1 stop bit, no parity).
    writel(UART_PL011_LCRH_FEN | UART_PL011_LCRH_WLEN_8, UART0_LCRH);

    // Enable UART0, receive & transfer enable of UART.
    writel(UART_PL011_CR_UARTEN | UART_PL011_CR_TXE |
    UART_PL011_CR_RXE,UART0_CR);
}

void uart_putc(unsigned char byte)
{
    // Wait for UART to become ready to transmit.
    while ( readl(UART0_FR) & UART_PL01x_FR_TXFF ) { }
    writel(byte,UART0_DR);
}

unsigned char uart_getc()
{
    // Wait for UART to have recieved something.
    while ( readl(UART0_FR) & UART_PL01x_FR_RXFE ) { }
    return readl(UART0_DR);
}

```

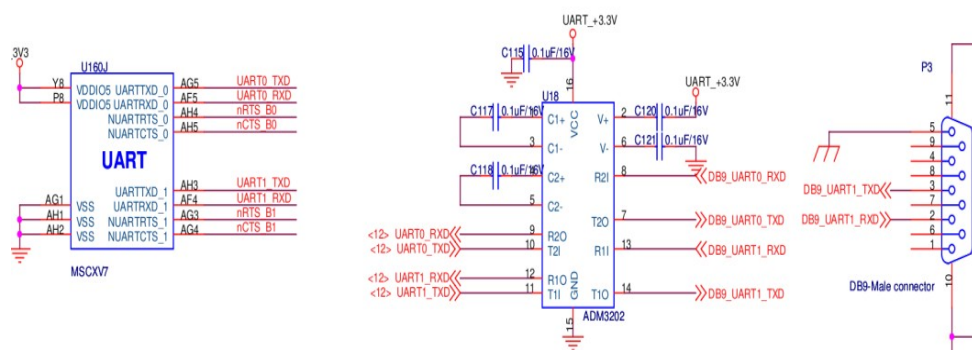
Test Procedure:

Open the code with Eclipse:

1. Check the linked script file especially "On-Chip RAM Memory Address" , Stack size (4KB)
2. Create work space , copy the "UART Test" code from the Repository to here
3. Check the memory map of the following

a. SHIKHARA_UART0_BASE	0xD457B000
b. SHIKHARA_UART1_BASE	0xD457C000
c. SHIKHARA_UART2_BASE	0xD457D000

- d. **SHIKHARA_UART3_BASE** **0xD457E000**
 - e. **CONFIG_PL011_CLOCK** **24000000 //24MHZ**
 - f. **CONFIGURE_BAUDRATE** **115200**
4. Compile the code , before should have the cross-compiler installed on "Host PC running Ubuntu 14.04 LTS[64-bit]"
 - a. `arm-none-eabi-gcc -nostdlib -mcpu=cortex-a9 -o test.elf -T test.ld main.c shikhara-uart.c startup.s -I ./include`
 - b. `arm-none-eabi-objcopy -O binary test.elf test.bin`
 5. After creating the elf file, when we connect the target we can see in the command console which elf file is loading
 6. Put the Breakpoints, see the expected output



2. GPIO Controller - PL061

Overview

General Purpose Input/Output (GPIO) is a generic pin on a chip whose behavior (including whether it is an input or output pin) can be controlled (programmed) by the user at run time. The GPIO is an AMBA slave module that connects to the Advanced Peripheral Bus (APB).

The GPIO provides eight programmable inputs or outputs that can be controlled in two modes:

- Software mode through an APB bus interface
- Hardware mode through a hardware control interface.

Ports of different widths can be created by multiple instantiation. An interrupt interface is provided to configure any number of pins as interrupt sources. Interrupts can be generated depending on a level, or a transitional value of a pin. At system reset, GPIO lines default to inputs. The GPIO interfaces with input and output pad cells using a data input, data output, and output enable per pad.

Features of the GPIO Controller

- Compliance to the AMBA Specification (Rev 2.0) onwards for easy integration into SoC implementation.
- Eight individually programmable input/output pins, default to input at reset.
- Scalability by multiple instantiation to 16, 24, 32, 40, or more bits.
- Programmable interrupt generation capability, from a transition or a level condition, on any number of pins.
- Hardware control capability of GPIO lines for different system configurations.
- Bit masking in both read and write operations through address lines.
- Identification registers that uniquely identify the GPIO.

AMBA APB interface

The AMBA APB interface generates read and write decodes for accesses to control, interrupt, and data registers within the GPIO.

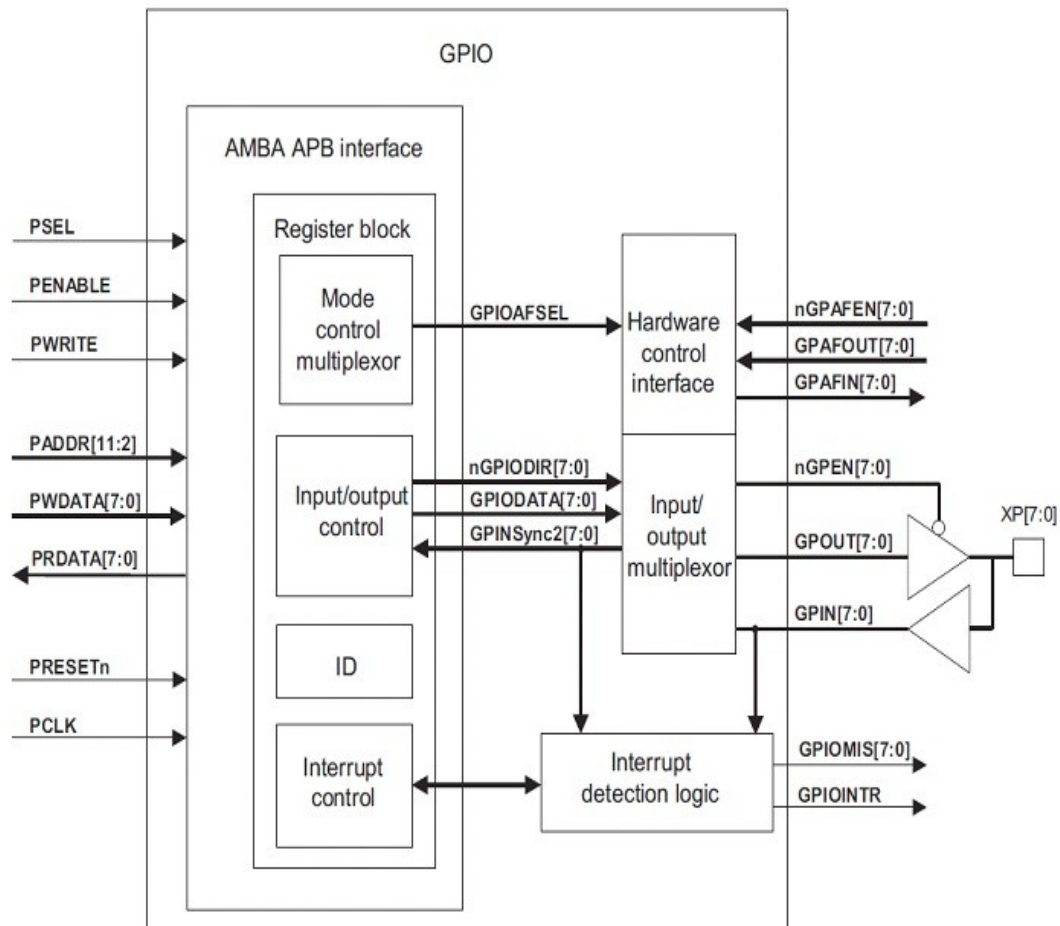
Interrupt detection logic

The GPIO has the ability to generate mask-programmable interrupts based on the level, or transitional value of any of its GPIO lines.

Mode control

GPIO lines can be controlled by software through the APB bus, or by hardware through the hardware control interface.

Block Diagram



Test Setup / Board Connections

```

/* Input port 0 , SHIKHARA_GPIO_BANK_A
   Input port 1 , SHIKHARA_GPIO_BANK_B */
int gpio_port_configure_output(int port)
{
    if(port == 0)
        writel(0xFF,SHIKHARA_GPIO_BANK_A + PL061_GPIODIR); // port direction is output
    if (port == 1)
        writel(0xFF,SHIKHARA_GPIO_BANK_B + PL061_GPIODIR); // port direction is output

    return 0;
}
/* Input port 0 , SHIKHARA_GPIO_BANK_A

```

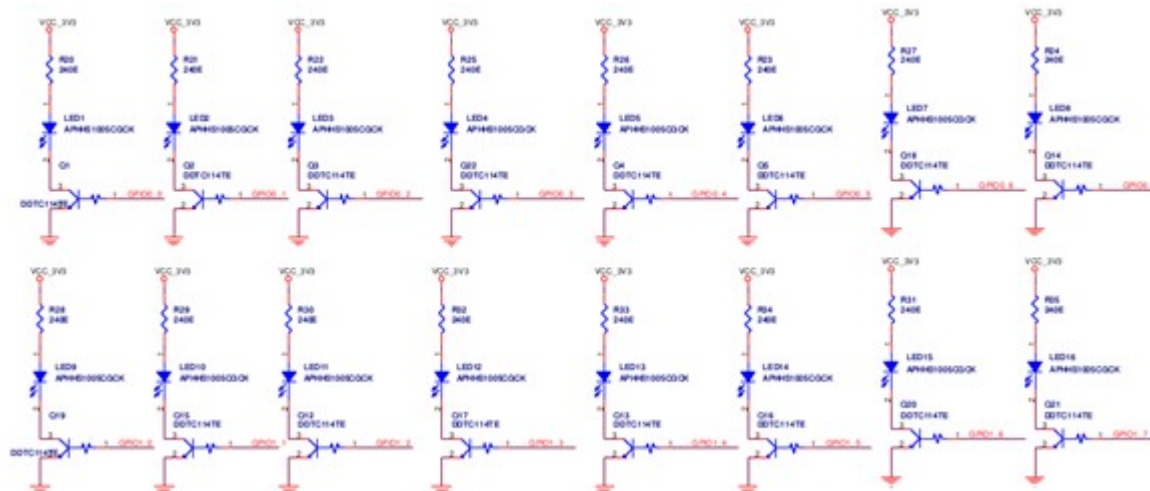
```

Input port 1 , SHIKHARA_GPIO_BANK_B */
int gpio_port_drive_high(int port)
{
    unsigned char value=1;
    int pin=0;

    if(port == 0) {
        for(pin=0;pin<8;pin++)
            writeb(value << pin, SHIKHARA_GPIO_BANK_A + (1 << (pin + 2)));
    }
    if(port == 1){
        for(pin=0;pin<8;pin++)
            writeb(value << pin, SHIKHARA_GPIO_BANK_B + (1 << (pin + 2)));
    }
    return 0;
}
/* Input port 0 , SHIKHARA_GPIO_BANK_A
Input port 1 , SHIKHARA_GPIO_BANK_B */
int gpio_port_drive_low(int port)
{
    unsigned char value=0;
    int pin=0;

    if(port == 0) {
        for(pin=0;pin<8;pin++)
            writeb(value << pin, SHIKHARA_GPIO_BANK_A + (1 << (pin + 2)));
    }
    if(port == 1){
        for(pin=0;pin<8;pin++)
            writeb(value << pin, SHIKHARA_GPIO_BANK_B + (1 << (pin + 2)));
    }
    return 0;
}

```



Test Procedure

Open the code with Eclipse:

1. Create work space , copy the "LED Blink Test" code from the Repository to work space
2. Check the linked script file especially "On-Chip RAM Memory Address" , Stack size (4KB) **0xD45B0000** ; On-Chip Memory Address 64KB
3. Check the memory-map is correct or not
 - a. **SHIKHARA_GPIO_BANK_A** **0xD4576000**
 - b. **SHIKHARA_GPIO_BANK_B** **0xD4577000**
4. Compile the code , before should have the cross-compiler installed on "Host PC running Ubuntu 14.04 LTS[64-bit]"
 - a. `arm-none-eabi-gcc -nostdlib -mcpu=cortex-a9 -o test.elf -T test.ld main.c gpio-leds.c startup.s -I ./include`
 - b. `arm-none-eabi-objcopy -O binary test.elf test.bin`
5. After creating the elf file, when we connect the target we can see in the command console which elf file is loading
6. Put the Breakpoints, see the expected output

5.Static Memory Controller (PL353)

1.Overview

The Prime cell SMC is an Advanced Microcontroller Bus Architecture (AMBA) compliant System-on-Chip (SoC) peripheral. The prime cell consists of high-performance, area-optimized SRAM and NAND memory controllers with on-chip bus interfaces that conform to the AMBA Advanced eXtensible Interface (AXI3) protocol and APB3 protocol.

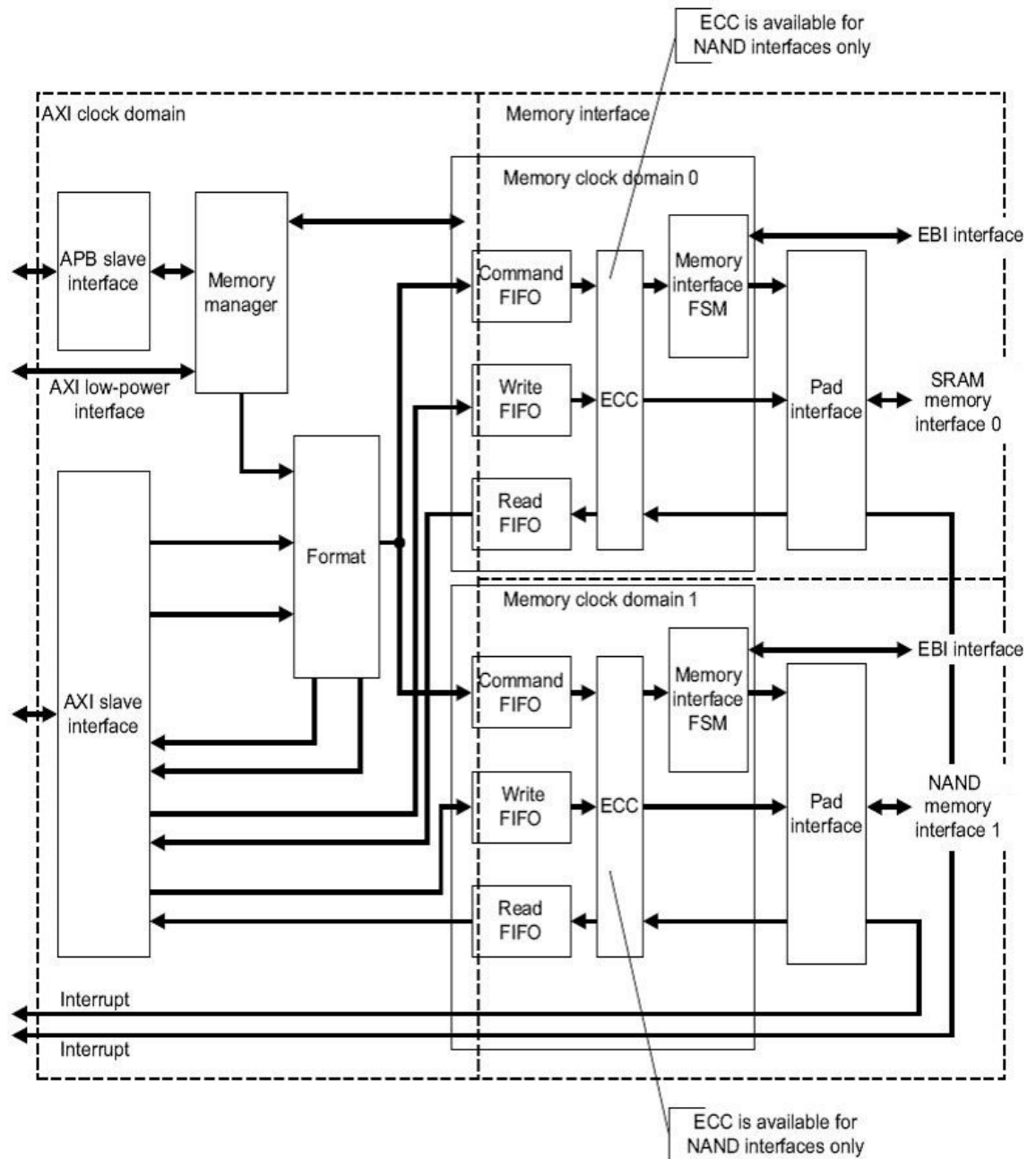
The NAND memory interface type is defined as supporting NAND flash with multiplexed Address/Data (A/D) buses.

The SRAM memory interface type is defined as supporting:

1. Synchronous or asynchronous SRAM
2. Pseudo Static Random Access Memory (PSRAM)
3. NOR flash
4. NAND flash devices with an SRAM interface.

2.Features of SMC (PL353):

- The SMC (PL353) supports two memory interfaces:
 1. Interface 0 type SRAM.
 2. Interface 1 type NAND.
- This configuration supports the following configurable parameters:
 - o 64-bit AXI data width
 - o 16-bit memory data width for interface 0. SRAM Interface will not support 8 bit data transfers because SMC has 64 bit AXI data width. Please refer to TRM page no 1-5.
 - o 16-bit memory data width for interface 1. But it supports 8 and 16 bit data transfers.
 - o 4 chip selects on each interface
 - o Command FIFO depth of 8
 - o Read data FIFO depth of 16
 - o Write data FIFO depth of 16
 - o Number of exclusive monitors - 4
 - o SLC ECC block for interface 1
 - o Entry block pipeline.



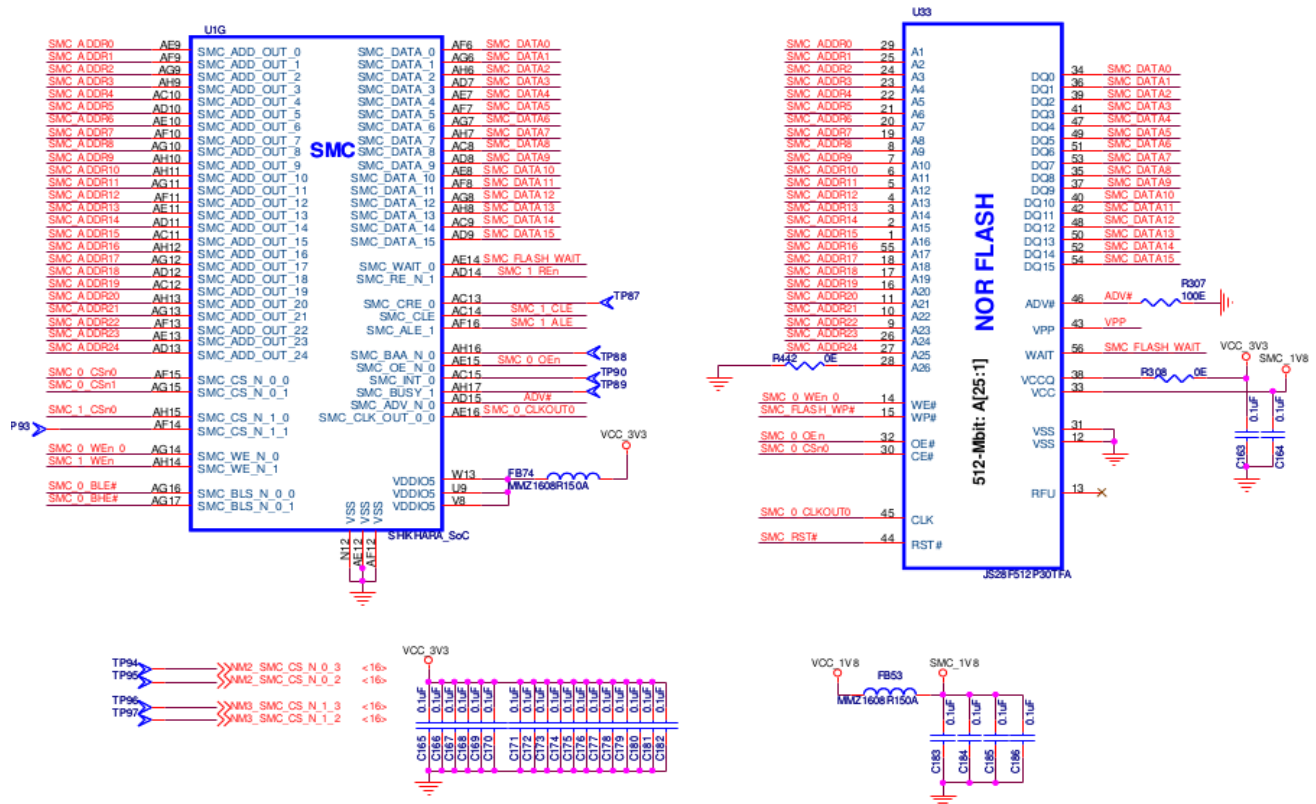
3. Operating Description

The AXI Slave interface of the SMC block takes the signals from AXI Interconnect Master port and feeds them to the Format block. The format block takes the decision between memory manager requests (RD/WR) and AXI slave interface requests (RD/WR) as per the predefined round robin arbitration scheme and maps AXI transfers on to the appropriated memory transfers and passes these to the memory interface through the command FIFO.

Memory manager block controls the direct commands issued to memory and also low power mode entry-to and exit-from through APB interface. The memory interface block mainly comprises of **CMD FIFO, WRITE FIFO, READ FIFO, Control FSM and EBI FSM** through which SMC carries out the **WR/RD operations with External NAND/NOR/SRAM memories**.

The Pad interface module provides a register I/O interface for data and control signals as the configured registers timing parameters. It also contains interrupt generation logic to support long wait states associated memories like NAND.

4. NOR



Bus Operation		RST#	CLK	ADV#	CE#	OE#	WE#	WAIT	DQ[15:0]	Notes
READ	Asynchronous	H	X	L	L	L	H	De-asserted	Output	-
	Synchronous	H	Running	L	L	L	H	Driven	Output	-
WRITE		H	X	L	L	H	L	High-Z	Input	1
OUTPUT DISABLE		H	X	X	L	H	H	High-Z	High-Z	2
STANDBY		H	X	X	H	X	X	High-Z	High-Z	2
RESET		L	X	X	X	X	X	High-Z	High-Z	2, 3

Mode	Command	Bus	First Bus Cycle			Second Bus Cycle		
		Cycles	Op	Addr ¹	Data ²	Op	Addr ¹	Data ²
Read	READ ARRAY	1	WRITE	DnA	0xFF	–	–	–
	READ DEVICE IDENTIFIER	≥2	WRITE	DnA	0x90	READ	DBA + IA	ID
	READ CFI	≥2	WRITE	DnA	0x98	READ	DBA + CFI-A	CFI-D
	READ STATUS REGISTER	2	WRITE	DnA	0x70	READ	DnA	SRD
	CLEAR STATUS REGISTER	1	WRITE	DnA	0x50	–	–	–
Program	WORD PROGRAM	2	WRITE	WA	0x40	WRITE	WA	WD
	BUFFERED PROGRAM ³	>2	WRITE	WA	0xE8	WRITE	WA	N - 1
	BUFFERED ENHANCED FACTORY PROGRAM (BEFP) ⁴	>2	WRITE	WA	0x80	WRITE	WA	0xD0
Erase	BLOCK ERASE	2	WRITE	BA	0x20	WRITE	BA	0xD0
Suspend	PROGRAM/ERASE SUSPEND	1	WRITE	DnA	0xB0	–	–	–
	PROGRAM/ERASE RESUME	1	WRITE	DnA	0xD0	–	–	–
Protection	BLOCK LOCK	2	WRITE	BA	0x60	WRITE	BA	0x01
	BLOCK UNLOCK	2	WRITE	BA	0x60	WRITE	BA	0xD0
	BLOCK LOCK DOWN	2	WRITE	BA	0x60	WRITE	BA	0x2F
	PROGRAM OTP REGISTER	2	WRITE	PRA	0xC0	WRITE	OTP-RA	OTP-D
	PROGRAM LOCK REGISTER	2	WRITE	LRA	0xC0	WRITE	LRA	LRD
Configuration	CONFIGURE READ CONFIGURATION REGISTER	2	WRITE	RCD	0x60	WRITE	RCD	0x03
Blank Check	BLOCK BLANK CHECK	2	WRITE	BA	0xBC	WRITE	BA	D0
EFI	EXTENDED FUNCTION INTERFACE ⁵	>2	WRITE	WA	0xEB	Write	WA	Sub-Op code

5.NOR FLASH Commands

CFI_CMD_READ_ID = 0x90,

CFI_CMD_CLEAR_STATUS = 0x50, CFI_CMD_READ_STATUS = 0x70,

```
CFI_CMD_RESET = 0xFF , CFI_CMD_READ=0xFF
```

```
CFI_CMD_BLOCK_ERASE = 0x20, CFI_CMD_ERASE_CONFIRM = 0xD0
```

```
(CFI_CMD_BLK_UNLCK_CONFIRM=0xD0
```

```
CFI_CMD_BLK_LOCK_STOP=0x0060, CFI_CMD_BLK_UNLCK_CONFIRM=0xD0
```

```
CFI_CMD_WRITE= 0x0040
```

```
/*Reading from NOR Flash*/
```

```
void smc_nor_read_id(void)
```

```
{  
uint16_t manufacture_id;  
    uint16_t device_id;  
    writew(CFI_CMD_READ_ID,SMC_NORFLASH_BASE_ADDR);  
    manufacture_id =  
    readw(SMC_NORFLASH_BASE_ADDR+CFI_OFFSET_MANUFACTURER_ID);  
    device_id = readw(SMC_NORFLASH_BASE_ADDR+2);//CFI_OFFSET_DEVICE_ID);  
    writew(CFI_CMD_RESET,SMC_NORFLASH_BASE_ADDR);  
    printf("Manufacturer Code=0x%2x\r\n",manufacture_id);  
    printf("Device ID Code=0x%4x\r\n",device_id);  
    top_boot=1;  
}
```

```
/*Reading from NOR Flash*/
```

```
void smc_nor_read(void)
```

```
{  
    unsigned int i,j=0;  
    for(i= 0; i < NOR_DEPTH ;i=i+2)  
    {
```

```

        writew(CFI_CMD_READ,((SMC_NORFLASH_BASE_ADDR+i)));
        read_data[j] = readw(((SMC_NORFLASH_BASE_ADDR+i)));
        j++;
    }
    for(j=0;j<60000;j++);
}

```

int smc_nor_blk_erase(unsigned int addr)

```

{
    uint16_t status;
    int res = 0;

    writew(CFI_CMD_CLEAR_STATUS,SMC_NORFLASH_BASE_ADDR+addr);
    writew(CFI_CMD_BLK_ERASE, SMC_NORFLASH_BASE_ADDR+addr);
    writew(CFI_CMD_BLK_ERASE_CNFM, SMC_NORFLASH_BASE_ADDR+addr);
}

```

/*Unlock the Block */

void smc_nor_blk_unlock(unsigned int addr)

```

{
    writew(CFI_CMD_BLK_LCK_STP, SMC_NORFLASH_BASE_ADDR+addr);
    writew(CFI_CMD_BLK_UNLCK, SMC_NORFLASH_BASE_ADDR+addr);
    writew(CFI_CMD_RESET,SMC_NORFLASH_BASE_ADDR);

}

```

/*writing to the NOR Flash */

int smc_nor_write(void)

```

{
    unsigned int i,j,res=0;
    for (i=0;i< NOR_DEPTH;i++)
        write_data[i]=0x1592;
    for(j= 0,i=0; j< NOR_DEPTH ;j=j+2)

```

```

{
    smc_nor_blk_unlock(j);

    smc_nor_blk_erase(j);

    writew(CFI_CMD_CLEAR_STATUS,SMC_NORFLASH_BASE_ADDR);

    writew(CFI_CMD_WRITE,((SMC_NORFLASH_BASE_ADDR+j)));

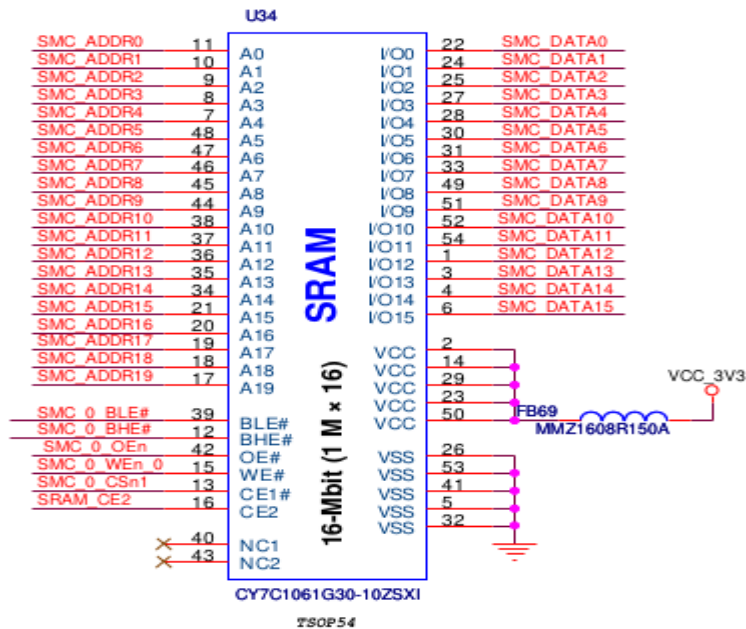
    writew(write_data[i],((SMC_NORFLASH_BASE_ADDR+j)));

    i++;
}

writew(CFI_CMD_RESET,SMC_NORFLASH_BASE_ADDR);
}

```

6. SRAM



CY7C1061G [1] is a high-performance CMOS fast static RAM automotive part with embedded ECC. ECC logic can detect and correct single-bit error in read data word during read cycles.

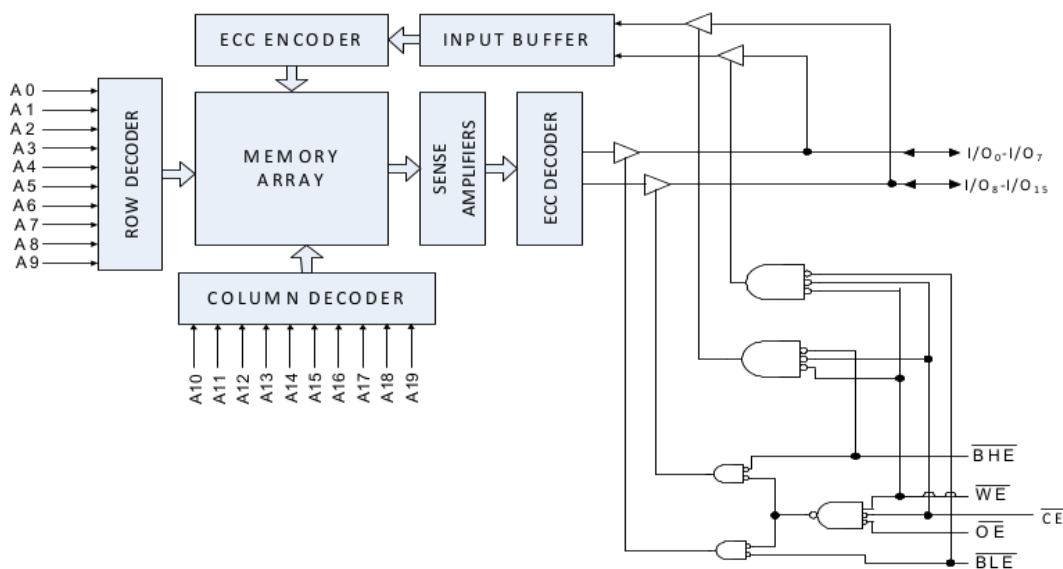
This device has single chip enable input and is accessed by asserting the chip enable input (CE) LOW.

To perform data writes, assert the Write Enable (WE) input LOW and provide the data and address on the device data pins (I/O 0 through I/O 15) and address pins (A 0 through A 19) respectively.

The Byte High Enable (BHE) and Byte Low Enable (BLE), inputs control byte writes and write data on the corresponding I/O lines to the memory location specified. BHE controls I/O 8 through I/O 15 and BLE controls I/O 0 through I/O 7 .

To perform data reads, assert the Output Enable (OE) input and provide the required address on the address lines. Read data is accessible on I/O lines (I/O 0 through I/O 15). You can perform byte accesses by asserting the required byte enable signal (BHE or BLE) to read either the upper byte or the lower byte of data from the specified address location.

Logic Block Diagram – CY7C1061G



Truth Table

$\overline{\text{CE}}$	$\overline{\text{OE}}$	$\overline{\text{WE}}$	$\overline{\text{BLE}}$	$\overline{\text{BHE}}$	I/O ₀ –I/O ₇	I/O ₈ –I/O ₁₅	Mode	Power
H	X ^[30]	X ^[30]	X ^[30]	X ^[30]	High Z	High Z	Power down	Standby (I _{SB})
L	L	H	L	L	Data out	Data out	Read all bits	Active (I _{CC})
L	L	H	L	H	Data out	High Z	Read lower bits only	Active (I _{CC})
L	L	H	H	L	High Z	Data out	Read upper bits only	Active (I _{CC})
L	X	L	L	L	Data in	Data in	Write all bits	Active (I _{CC})
L	X	L	L	H	Data in	High Z	Write lower bits only	Active (I _{CC})
L	X	L	H	L	High Z	Data in	Write upper bits only	Active (I _{CC})
L	H	H	X	X	High Z	High Z	Selected, outputs disabled	Active (I _{CC})
L	X	X	H	H	High Z	High Z	Selected, outputs disabled	Active (I _{CC})

/*Reading from SRAM */

void smc_sram_read(void)

```
{
    unsigned int i;
    for(i= 0; i < SRAM_DEPTH ;i=i+1 )
    {
        read_data[i] = readw(SMC_SRAM_BASE_ADDR+(2*i));
    }
}
```

/*writing to the SRAM */

void smc_sram_write(void)

```
{
    unsigned int i,j;
    for (i=0;i< SRAM_DEPTH;i++)
        write_data[i]=0x1592;
    for(j= 0,i=0; j< SRAM_DEPTH ;j=j+1,i++)
    {
        writew(write_data[j],SMC_SRAM_BASE_ADDR+(2*i));
    }
}
```

6.SPI FLASH

1. SPI bus-compatible serial interface
2. **128Mb Flash memory**
3. **54 MHz clock frequency (maximum)**
4. **2.7V to 3.6V single supply voltage**
5. **V PP = 9V for fast program/erase mode (optional)**
6. Page program (up to 256 bytes) in
 - 0.5ms (TYP)
 - 0.4ms (TYP with V PP = 9V)
7. Erase capability
 - **Sector erase: 2Mb**
 - **Bulk erase: 128Mb**

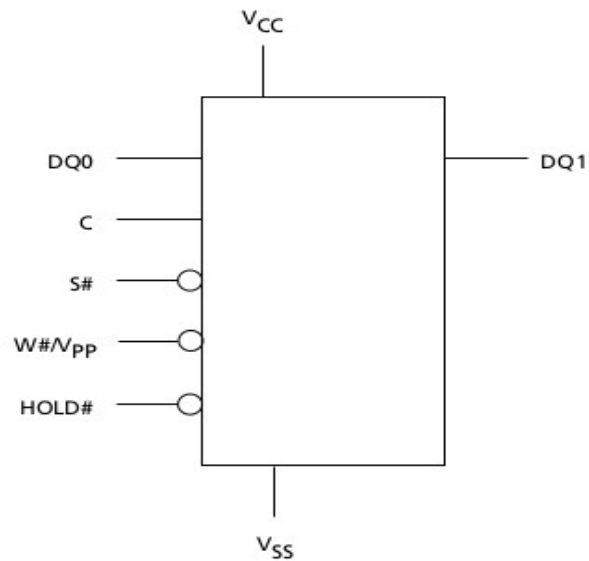
The **M25P128** is a **128Mb (16Mb x 8)** serial Flash memory device with advanced write protection mechanisms accessed by a high speed SPI-compatible bus. The device supports high-performance commands for clock frequency up to 54 MHz.

The memory can be programmed **1 to 256 bytes** at a time using the **PAGE PROGRAM** command. It is organized as 64 sectors, each containing 1024 pages. Each page is 256 bytes wide. Memory can be viewed either as 65,536 pages or as 16,777,216 bytes.

An enhanced fast program/erase mode is available to speed up operations in factory environment. The device enters this mode whenever the V PPH voltage is applied to the W#/V PP pin.

The entire memory can be erased using the **BULK ERASE** command, or it can be erased one sector at a time using the **SECTOR ERASE** command.

PAGE SIZE	256 (PAGE PROGRAM)
Number of Pages for Sector	1024 (SECTOR ERASE)
Total number of sectors	64 (BULK ERASE)
Total Size	64 * 1024* 256 = 16 MB(Bytes)



SPI Modes

These devices can be driven by a microcontroller with its serial peripheral interface (SPI) running in either of the following two SPI modes:

- CPOL = 0, CPHA = 0
- CPOL = 1, CPHA = 1

For these two modes, input data is latched in on the rising edge of serial clock (C), and output data is available from the falling edge of C.

The difference between the two modes is the clock polarity when the bus master is in standby mode and not transferring data:

- C remains at 0 for (CPOL = 0, CPHA = 0)
- C remains at 1 for (CPOL = 1, CPHA = 1)

: SPI Modes Supported

CPOL CPHA

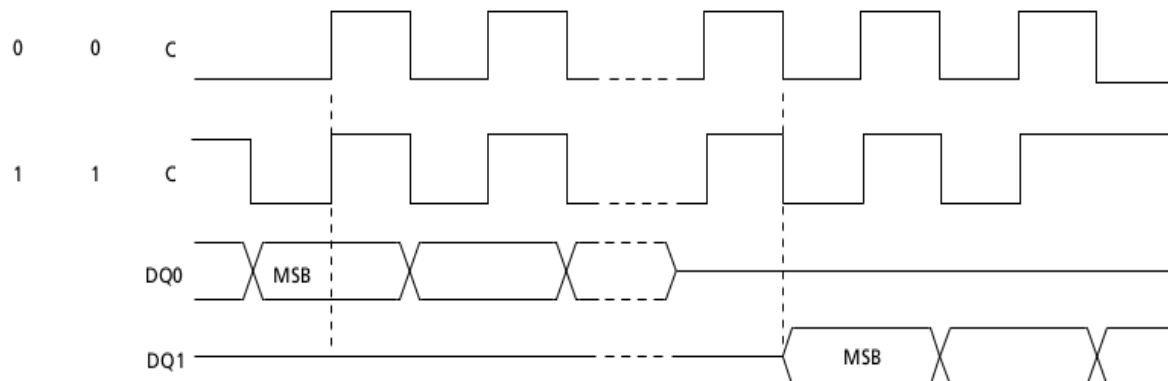


Figure 1. SPI Modes Supported

SPI NOR FLASH COMMANDS

Table 7. Command Set Codes

Command Name	One-Byte Command Code		Bytes		
			Address	Dummy	Data
WRITE ENABLE	0000 0110	06h	0	0	0
WRITE DISABLE	0000 0100	04h	0	0	0
READ IDENTIFICATION	1001 1111	9Fh	0	0	1 to 20
	1001 1110	9Eh			
READ STATUS REGISTER	0000 0101	05h	0	0	1 to ∞
WRITE STATUS REGISTER	0000 0001	01h	0	0	1
READ DATA BYTES	0000 0011	03h	3	0	1 to ∞
READ DATA BYTES at HIGHER SPEED	0000 1011	0Bh	3	1	1 to ∞
PAGE PROGRAM	0000 0010	02h	3	0	1 to 256
SECTOR ERASE	1101 1000	D8h	3	0	0
BULK ERASE	1100 0111	C7h	0	0	0

WRITE ENABLE

The WRITE ENABLE command sets the write enable latch (WEL) bit. The WEL bit must be set before execution of every **PROGRAM, ERASE, and WRITE command**.

The **WRITE ENABLE command** is entered by driving chip select (S#) LOW, sending the command code, and then driving S# HIGH.

WRITE DISABLE

The **WRITE DISABLE command** resets the write enable latch (WEL) bit. The WRITE DISABLE command is entered by driving chip select (S#) LOW, sending the command code, and then driving S# HIGH.

The WEL bit is reset under the following conditions:

1. Power-up
2. Completion of any ERASE operation
3. Completion of any PROGRAM operation
4. Completion of any WRITE STATUS REGISTER operation
5. Completion of WRITE DISABLE operation

READ IDENTIFICATION

The READ IDENTIFICATION command reads the following device identification data:

- **Manufacturer identification (1 byte): This is assigned by JEDEC.**
- **Device identification (2 bytes): This is assigned by device manufacturer; the first byte indicates memory type and the second byte indicates device memory capacity.**

READ STATUS REGISTER

The **READ STATUS REGISTER** command allows the status register to be read. The status register may be read at any time, even while a **PROGRAM, ERASE, or WRITE STATUS REGISTER** cycle is in progress. When one of these cycles is in progress, it is recommended to check the write in progress (WIP) bit before sending a new command to the device. It is also possible to read the status register continuously.

WRITE STATUS REGISTER

The WRITE STATUS REGISTER command allows new values to be written to the status register. Before the WRITE STATUS REGISTER command can be accepted, a WRITE ENABLE command must have been executed previously. After the WRITE ENABLE command has been decoded and

executed, the device sets the write enable latch (WEL) bit.

READ DATA BYTES

The device is first selected by driving chip select (S#) LOW. The command code for **READ DATA BYTES** is followed by a 3-byte address (A23-A0), each bit being latched-in during the rising edge of serial clock (C). Then the memory contents at that address is shifted out on serial data output (DQ1), each bit being shifted out at a maximum frequency f_R during the falling edge of C.

The first byte addressed can be at any location. The address is automatically incremented to the next higher address after each byte of data is shifted out. Therefore, the entire memory can be read with a single **READ DATA BYTES** command. When the highest address is reached, the address counter rolls over to 000000h, allowing the read sequence to be continued indefinitely.

PAGE PROGRAM

The **PAGE PROGRAM** command allows bytes in the memory to be programmed, which means the bits are changed from 1 to 0. Before a PAGE PROGRAM command can be accepted a WRITE ENABLE command must be executed. After the WRITE ENABLE command has been decoded, the device sets the write enable latch (WEL) bit.

The PAGE PROGRAM command is entered by driving chip select (S#) LOW, followed by the command code, three address bytes, and at least one data byte on serial data input (DQ0).

SECTOR ERASE

The **SECTOR ERASE** command sets to 1 (FFh) all bits inside the chosen sector. Before the SECTOR ERASE command can be accepted, a WRITE ENABLE command must have been executed previously. After the WRITE ENABLE command has been decoded, the device sets the write enable latch (WEL) bit.

The SECTOR ERASE command is entered by driving chip select (S#) LOW, followed by the command code, and three address bytes on serial data input (DQ0). Any address inside the sector is a valid address for the SECTOR ERASE command. S# must be driven LOW for the entire duration of the sequence.

BULK ERASE

The **BULK ERASE** command sets all bits to 1 (FFh). Before the **BULK ERASE** command can be accepted, a **WRITE ENABLE** command must have been executed previously. After the **WRITE ENABLE** command has been decoded, the device sets the write enable latch (WEL) bit.

The **BULK ERASE** command is entered by driving chip select (S#) LOW, followed by the command code on serial data input (DQ0). S# must be driven LOW for the entire duration of the sequence.

7.SD/MMC (Secure Digital / Multi Media Card) Controller

Overview

This chapter describes the on-chip SD/MMC controller (DWC_mobile_storage from Synopsys) details for the Shikhara SoC. Design Ware Configurable host controller interfaces SD, MMC, CE-ATA and SDIO cards to Shikhara SoC with generic interface. The interface towards the SD card is realized by the SD protocol with security implemented in the controller. The DWC_mobile_storage can be configured either as a Multimedia Card-only controller or as a Secure Digital_Multimedia Card controller, which simultaneously supports Secure Digital memory (SD Mem), Secure Digital I/O (SDIO), Multimedia Cards (MMC), and Consumer Electronics Advanced Transport Architecture (CE-ATA).

Features

The DWC Mobile Storage Host is a Secure Digital (SD), Multimedia Card (MMC), and CE-ATA host controller that can be configured and synthesized in order to control:

- o Secure Digital memory (SD mem – version 3.0)
 - o Secure Digital I/O (SDIO - version 3.0)
 - o Consumer Electronics Advanced Transport Architecture (CE-ATA – version 1.1)
 - o Multimedia Cards (MMC - version 4.41 and eMMC4.5)
-
- Supports Secure Digital memory protocol commands
 - Supports Secure Digital I/O protocol commands
 - Supports Multimedia Card protocol commands
 - Supports CE-ATA digital protocol commands
 - Supports Command Completion signal and interrupt to host processor

- Command Completion Signal disable feature

The following features of MMC4.41 are supported:

- DDR in 4 bit mode and 8 bit mode
- GO_PRE_IDLE_STATE command (CMD0 with argument F0F0F0F0)
- New EXTCSO registers

The following features of MMC4.4 are not supported:

- DDR in 8 bit mode
- Boot in DDR mode
- Hardware Reset Pin

- **Bus Interface Features**

The following are features for the Bus Interface Unit (BIU):

- o Supports AMBA AHB interface
- o Supports data width of 32 bits
- o In addition to AMBA slave interface, supports external DMA controllers for data transfers.
- o The external DMA interface is present when the internal DMAC is not present.
- o Interface to DW_ahb_dmac DMA controller, which shares AMBA host bus for DMA transfers
- o Does not generate split, retry, or error responses on the AMBA Slave AHB bus
- o Supports pin-based little-endian or big-endian modes of AHB operation
- o Supports separate clocks for bus interface and card interface for ease of integration
- o Supports combined single FIFO for both transmit and receive operations, which saves area and power
- o Supports FIFO depths of 128
- o FIFO controller shipped with a register-based synchronous RAM for area-sensitive applications
- o Supports FIFO over-run and under-run prevention by stopping card clock

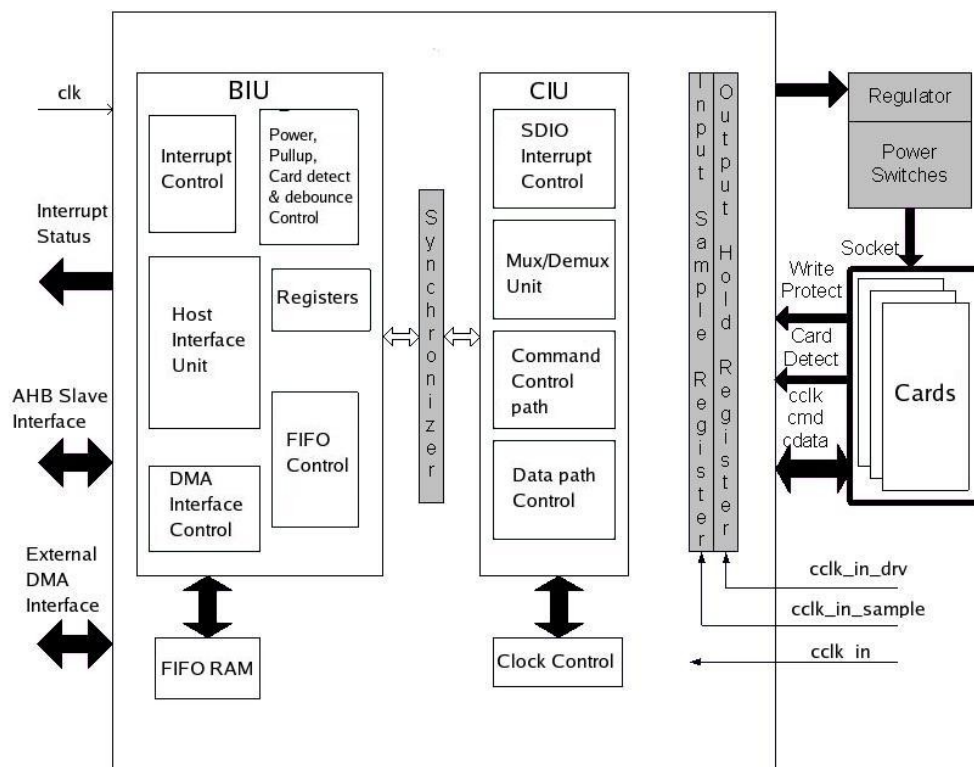
- **Card Interface Features**

The following are features for the Card Interface Unit (CIU):

- o It is configured as SD_MMC controller
- o Supports 1 to 16 SD or MMC (3.3 or 4.0) or CE-ATA devices in SD_MMC_CE-ATA mode.
- o Supports Command Completion Signal and interrupts to host
- o Supports Command Completion Signal disable
- o Supports CRC generation and error detection
- o Supports programmable baud rate. Supports up to 4 clock dividers to support simultaneous operation of multiple cards with different clock speed requirements

- o Provides individual clock control to selectively turn ON or OFF clock to a card
- o Supports power management and power switch. Provides individual power control to selectively turn ON or OFF power to a card
- o Supports host pull-up control
- o Supports card detection and initialization
- o Supports write protection
- o Supports SDIO interrupts in 1-bit and 4-bit modes
- o Supports SDIO suspend and resume operation
- o Supports SDIO read wait
- o Supports block size of 1 to 65,535 bytes
- o Supports 4-bit DDR, as defined by SD3.0 and MMC4.4

Block diagram



DWC Mobile Storage host controller Description

- An SD_MMC memory card is typically a device for FLASH mass storage.
- The SDIO card usually functions in I/O applications, which can also have optional FLASH

memory.

- The CE-ATA card functions in mobile device applications.

Internal Block Diagram of SD/MMC controller

- The DWC_mobile_storage consists of the following main functional blocks,
- Bus Interface Unit (BIU) – Provides AMBA AHB and DMA interfaces for register and data read/writes.
- Card Interface Unit (CIU) – Takes care of the SD_MMC_CEATA protocols and provides clock management.

Operating Description

This section provides an overview of the DWC_mobile_storage architecture.

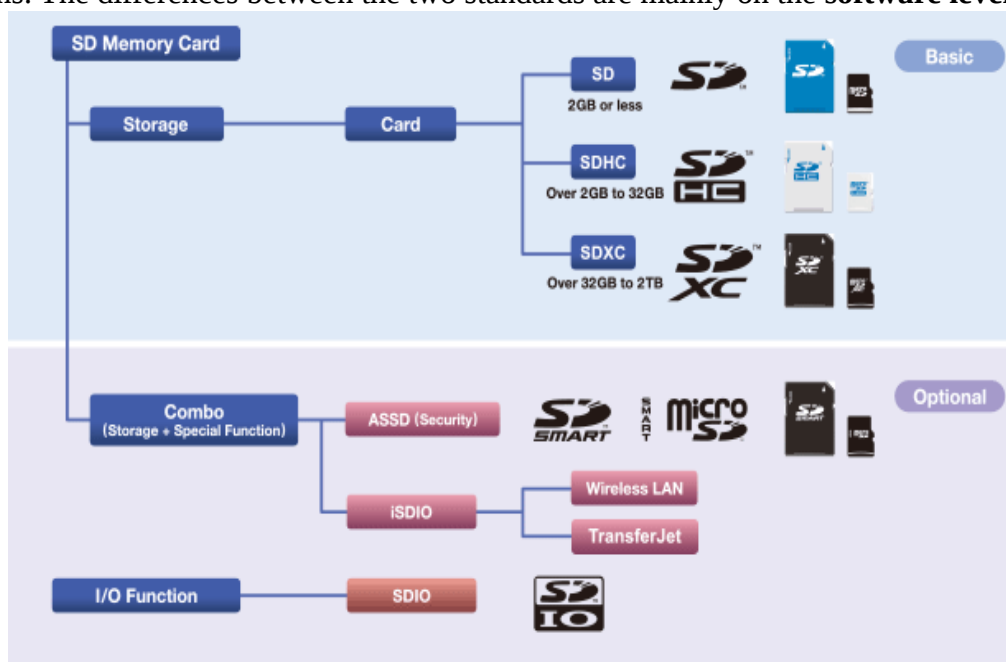
The blocks are:

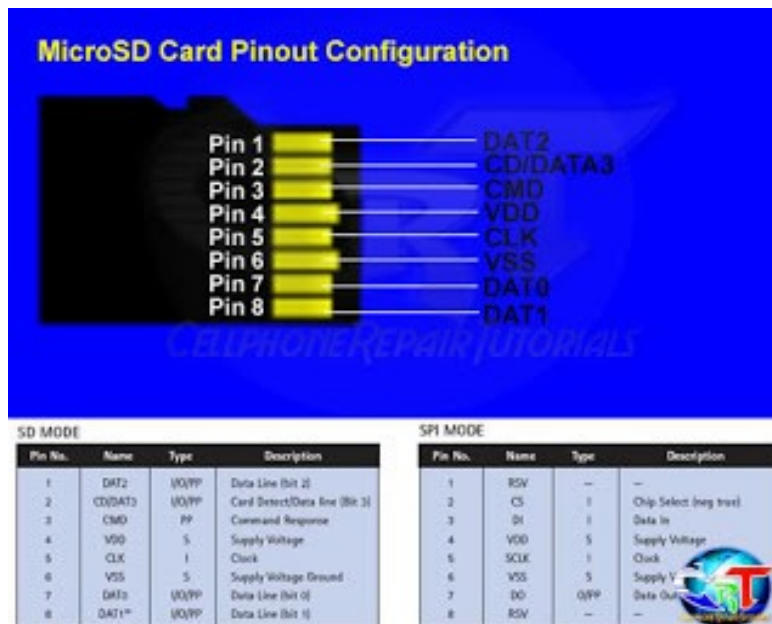
- “Bus Interface Unit (BIU)”
- “Card Interface Unit (CIU)”

The BIU provides the host interface to the registers and the data FIFO through the Host Interface Unit (HIU). Additionally, it provides independent data FIFO access through a DMA interface. The DWC_mobile_storage CIU controls the card-specific protocols. Within the CIU, the command path control unit and data path control unit interface the controller to the command and data ports of the SD_MMC_CEATA cards. The CIU also provides clock control.

SD/MMC:

MMC and SD-card share many common features and have the same physical and electrical specifications. The differences between the two standards are mainly on the **software level commands**.





microSD Pinout, SD Mode





Pin #	Pin Name	Signal Function
1	DAT2	Data Bit 2
2	CD/DAT3	Card Detect / Data Bit 3
3	CMD	Command Line
4	Vdd	Supply Voltage 2.7v / 3.6v

5	CLK	Clock
6	Vss	Ground
7	DAT0	Data Bit 0
8	DAT1	Data Bit 1

SD protocol:

Command is transferred to and fro via bidirectional CMD in form of discrete packets of 48 bits. These command packets include command index, argument and CRC bits. Command is always sent via host and received via sd card. Response packets are also of 48 bits. During data transfer, basic unit of transfer is called block, which is usually of 512 bytes and is transferred via all 4 data pins. Also, after each block transfer, 16 bit of CRC data is also sent.

Comparison of SD card capacity standards[\[64\]](#)

	SD	SDHC	SDXC	SDUC
Logo				
Capacity	Min 128MiB	2GiB	32GiB	2TiB
	Max 2GiB	32GiB	2TiB	128TiB
Typical FS	FAT16	FAT32	FAT32/exFAT	exFAT

All commands have a fixed code length of 48 bits

Bit position	47	46	[45:40]	[39:8]	[7:1]	0
Width (bits)	1	1	6	32	7	1
Value	'0'	'1'	x	x	x	'1'
Description	start bit	transmission bit	command index	argument	CRC7	end bit

Table 4-16: Command Format

CMD INDEX	type	argument	resp	abbreviation	command description
CMD0	bc	[31:0] stuff bits	-	GO_IDLE_STATE	Resets all cards to idle state
CMD1	reserved				
CMD2	bcr	[31:0] stuff bits	R2	ALL_SEND_CID	Asks any card to send the CID numbers on the CMD line (any card that is connected to the host will respond)
CMD3	bcr	[31:0] stuff bits	R6	SEND_RELATIVE_ADDR	Ask the card to publish a new relative address (RCA)
CMD4	bc	[31:16] DSR [15:0] stuff bits	-	SET_DSR	Programs the DSR of all cards
CMD5	reserved for I/O cards (refer to the "SDIO Card Specification")				
CMD7	ac	[31:16] RCA [15:0] stuff bits	R1b (only from the selected card)	SELECT/DESELECT_CARD	Command toggles a card between the stand-by and transfer states or between the programming and disconnect states. In both cases, the card is selected by its own relative address and gets deselected by any other address; address 0 deselects all. In the case that the RCA equals 0, then the host may do one of the following: <ul style="list-style-type: none"> - Use other RCA number to perform card de-selection. - Re-send CMD3 to change its RCA number to other than 0 and then use CMD7 with RCA=0 for card de-selection.
CMD8	bcr	[31:12]reserved bits [11:8]supply voltage(VHS) [7:0]check pattern	R7	SEND_IF_COND	Sends SD Memory Card interface condition, which includes host supply voltage information and asks the card whether card supports voltage. Reserved bits shall be set to '0'.
CMD9	ac	[31:16] RCA [15:0] stuff bits	R2	SEND_CSD	Addressed card sends its card-specific data (CSD) on the CMD line.
CMD10	ac	[31:16] RCA [15:0] stuff bits	R2	SEND_CID	Addressed card sends its card identification (CID) on CMD the line.
CMD11	reserved				
CMD12	ac	[31:0] stuff bits	R1b	STOP_TRANSMISSION	Forces the card to stop transmission
CMD13	ac	[31:16] RCA [15:0] stuff bits	R1	SEND_STATUS	Addressed card sends its status register.
CMD14	reserved				

CMD INDEX	type	argument	resp	abbreviation	command description
CMD15	ac	[31:16] RCA [15:0] reserved bits	-	GO_INACTIVE_STATE	Sends an addressed card into the <i>Inactive State</i> . This command is used when the host explicitly wants to deactivate a card. Reserved bits shall be set to '0'.

Table 4-18: Basic Commands (class 0)

CMD INDEX	type	argument	resp	abbreviation	command description
CMD16	ac	[31:0] block length	R1	SET_BLOCKLEN	In the case of a Standard Capacity SD Memory Card, this command sets the block length (in bytes) for all following block commands (read, write, lock). Default block length is fixed to 512 Bytes. Set length is valid for memory access commands only if partial block read operation are allowed in CSD. In the case of a High Capacity SD Memory Card, block length set by CMD16 command does not affect the memory read and write commands. Always 512 Bytes fixed block length is used. This command is effective for LOCK_UNLOCK command. In both cases, if block length is set larger than 512Bytes, the card sets the BLOCK_LEN_ERROR bit.
CMD17	adtc	[31:0] data address ²	R1	READ_SINGLE_BLOCK	In the case of a Standard Capacity SD Memory Card, this command, this command reads a block of the size selected by the SET_BLOCKLEN command. ¹ In the case of a High Capacity Card, block length is fixed 512 Bytes regardless of the SET_BLOCKLEN command.
CMD18	adtc	[31:0] data address ²	R1	READ_MULTIPLE_BLOCK	Continuously transfers data blocks from card to host until interrupted by a STOP_TRANSMISSION command. Block length is specified the same as READ_SINGLE_BLOCK command.
CMD19 ... CMD23	reserved				

1) The data transferred shall not cross a physical block boundary unless READ_BLK_MISALIGN is set in the CSD.

2) Data address is in byte units in a Standard Capacity SD Memory Card and in block (512 Byte) units in a High Capacity SD Memory Card.

Table 4-19: Block-Oriented Read Commands (class 2)

CMD INDEX	type	argument	resp	abbreviation	command description
CMD16	ac	[31:0] block length	R1	SET_BLOCKLEN	See description in Table 4-19
CMD24	adtc	[31:0] data address ²	R1	WRITE_BLOCK	In the case of a Standard Capacity SD Memory Card, this command writes a block of the size selected by the ¹ SET_BLOCKLEN command. In the case of a High Capacity Card, block length is fixed 512 Bytes regardless of the SET_BLOCKLEN command.
CMD25	adtc	[31:0] data address ²	R1	WRITE_MULTIPLE_BLOCK	Continuously writes blocks of data until a STOP_TRANSMISSION follows. Block length is specified the same as WRITE_BLOCK command.
CMD26	Reserved For Manufacturer				
CMD27	adtc	[31:0] stuff bits	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.

- 1) The data transferred shall not cross a physical block boundary unless WRITE_BLK_MISALIGN is set in the CSD. In the case that write partial blocks is not supported, then the block length=default block length (given in CSD).
- 2) Data address is in byte units in a Standard Capacity SD Memory Card and in block (512 Byte) units in a High Capacity SD Memory Card.

Table 4-20: Block-Oriented Write Commands (class 4)

R1 (normal response command):

Code length is 48 bits. The bits 45:40 indicate the index of the command to be responded to, this value being interpreted as a binary coded number (between 0 and 63). The status of the card is coded in 32 bits.

Bit position	47	46	[45:40]	[39:8]	[7:1]	0
Width (bits)	1	1	6	32	7	1
Value	'0'	'0'	x	x	x	'1'
Description	start bit	transmission bit	command index	card status	CRC7	end bit

Table 4-29: Response R1

R2 (CID, CSD register)

Code length is 136 bits. The contents of the CID register are sent as a response to the commands

CMD2 and CMD10. The contents of the CSD register are sent as a response to CMD9. Only the bits [127...1] of the CID and CSD are transferred, the reserved bit ¹ of these registers is replaced by the end bit of the response.

Bit position	135	134	[133:128]	[127:1]	0
Width (bits)	1	1	6	127	1
Value	'0'	'0'	'111111'	x	'1'
Description	start bit	transmission bit	reserved	CID or CSD register incl. internal CRC7	end bit

Table 4-30: Response R2

R3 (OCR register)

Code length is 48 bits. The contents of the OCR register are sent as a response to ACMD41.

Bit position	47	46	[45:40]	[39:8]	[7:1]	0
Width (bits)	1	1	6	32	7	1
Value	'0'	'0'	'111111'	x	'1111111'	'1'
Description	start bit	transmission bit	reserved	OCR register	reserved	end bit

Table 4-31: Response R3

R6 (Published RCA response)

Code length is 48 bit. The bits 45:40 indicate the index of the Command to be responded to - in that case, it will be '000011'(together with bit 5 in the status bits it means = CMD3). The 16 MSB bits of the argument field are used for the Published RCA number.

Bit position	47	46	[45:40]	[39:8] Argument field		[7:1]	0
Width (bits)	1	1	6	16	16	7	1
Value	'0'	'0'	x	x	x	x	'1'
Description	start bit	transmission bit	command index ('000011')	New published RCA [31:16] of the card	[15:0] card status bits: 23,22,19,12:0 (see Table 4-35)	CRC7	end bit

Table 4-32: Response R6

R7 (Card interface condition)

Code length is 48 bits. The card support voltage information is sent by the response of CMD8. Bits 19-16 indicate the voltage range that the card supports. The card that accepted the supplied voltage returns R7 response. In the response, the card echoes back both the voltage range and check pattern set in the argument.

Bit position	47	46	[45:40]	[39:20]	[19:16]	[15:8]	[7:1]	0
Width (bits)	1	1	6	20	4	8	7	1
Value	'0'	'0'	'001000'	'00000h'	x	x	x	'1'
Description	start bit	transmission bit	command index	reserved bits	voltage accepted	echo-back of check pattern	CRC7	end bit

Table 4-33: Response R7

Card Initialization and Identification

The initialization process starts with SD_{SENDOPCOND} (ACMD41) by setting its operational conditions and the HCS bit in the OCR. The HCS (Host Capacity Support) bit set to 1 indicates that the host supports High Capacity SD Memory card. The HCS (Host Capacity Support) bit set to 0 indicates that the host does not support High Capacity SD Memory card.

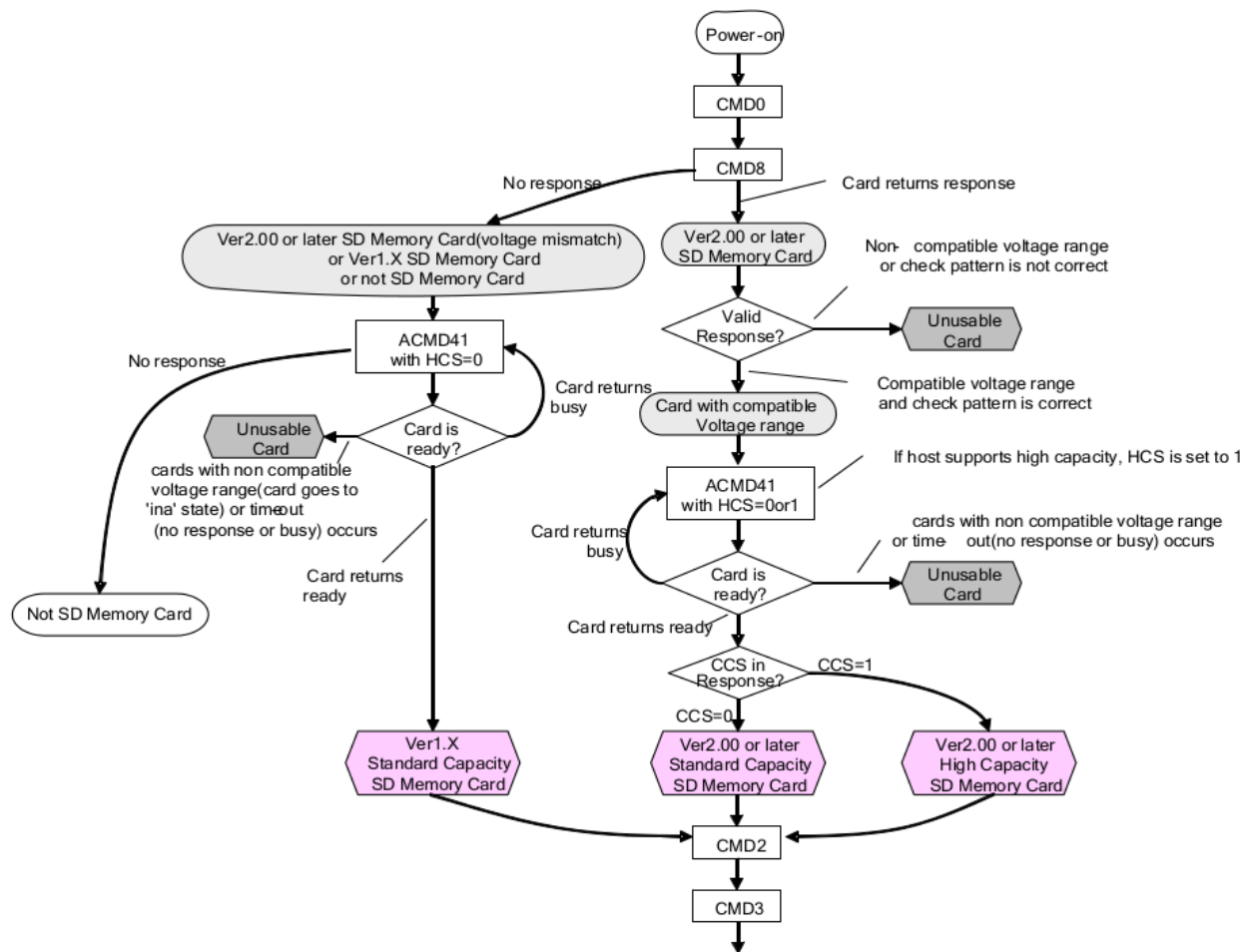


Figure 4-2: Card Initialization and Identification Flow (SD mode)

8.LPDDR2/DDR3 Controller

Overview

The **Design Ware Cores Enhanced Universal DDR Memory Controller (uMCTL2)** combined with **DWC DDR PHY** forms a complete memory interface solution for **DDR memory subsystems**. The Controller can accept memory access requests from up to 16 application side host ports (only 3 application side host ports are used). **The Controller receives read requests, write requests and data through a single/multi- port host interface(AMBA AXI 3 Interface)**. Write requests are sent to the memory controller followed by the associated write data from the SoC core. Read requests are made with a tag field. The memory controller returns the tag field with the data when it returns read data. The memory controller receives the transactions

from the SoC core which are queued internally and scheduled for access to the SDRAM while satisfying SDRAM protocol timing requirements, transaction priorities and dependencies between the transactions. The memory controller in turn issues commands on the DFI interface to the PHY module, which launches and captures data to and from the SDRAM.

The memory controller provides access to the external LPDDR2/DDR3 -SDRAM memory devices and is shared between the Cortex-A9 CPU and other peripherals

Features of LPDDR2/DDR3 Controller

- 1:2 frequency ratio architecture:
 - 1:2 frequency ratio architecture uses a 4:1 data width conversion from the HIF bus to the DDR memory data bus
- For LPDDR2/DDR3 configuration, direct external control or programmable internal control for ZQ short calibration cycles
- For LPDDR2/DDR3 configuration, support for ZQ long calibration after self-refresh exit
- For LPDDR2/DDR3 configuration, support for ZQ Reset feature through software
- Dynamic scheduling to optimize bandwidth and latency
- Read and write buffers in fully associative CAMs, configurable in powers of two, up to 32 reads and 32 writes
- Delayed writes for optimum performance on SDRAM data bus
- or maximum SDRAM efficiency, commands are executed out-of-order:
 - Read requests accompanied by a unique token (tag) from HIF
 - Read data returned with token (tag) for SoC core to associate read data with correct read request
- Programmable SDRAM parameters
- Software programmable Quality of Service (QoS) support:
 - Support for three traffic classes on read commands—high priority reads, variable priority reads and low priority reads

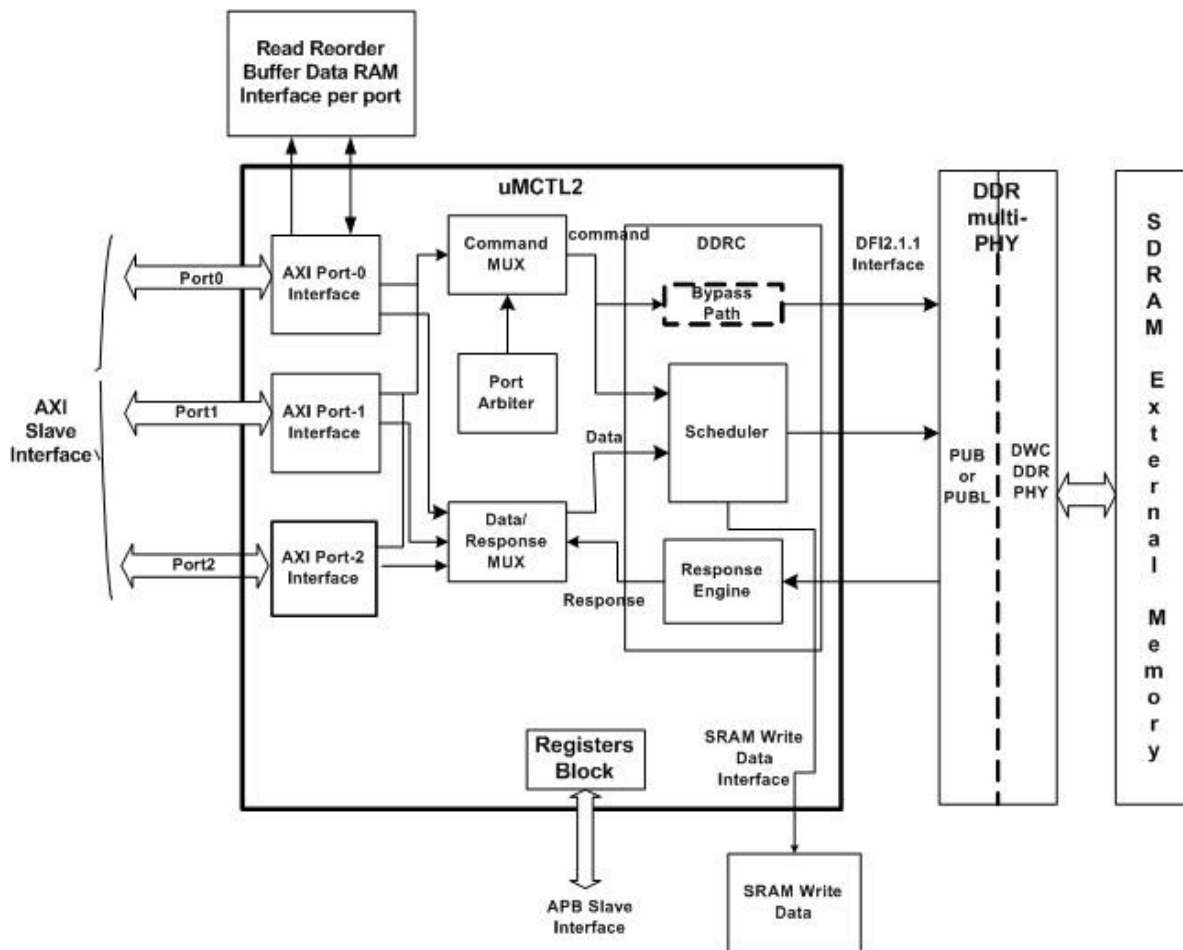
- Support for two traffic classes on write commands—normal priority writes and variable priority writes
- Support for port urgent and port throttling control
- Configurable maximum SDRAM data bus-width (full data bus-width)
- Programmable support for all of the following SDRAM data bus-widths:
 - The full data bus-width or
 - Half of the full data bus-width or
 - One quarter of the full data bus-width (if appropriate hardware configuration option is selected)
This applies to the width of the data excluding the ECC byte (if configured for ECC)
- **Supports the following burst length options if configured for BL8 operation**
(MEMC_BURST_LENGTH=8):
 - SDRAM burst length of 2, 4 or 8 in full bus-width mode
 - SDRAM burst length of 2, 4, 8, or 16 in half or quarter bus-width mode
- Two priorities for read transactions, and one for writes
- **Supports 1 or 2 memory ranks**
- Control options to avoid starvation of lower priorities
- Guaranteed **coherency for write-after-read (WAR) and read-after-write (RAW) hazards (HIF interface only)**
- Write combine to allow multiple writes to the same address to be combined into a single write to SDRAM; supported for same starting address
- Paging policy selectable by configuration inputs as any of the following:
 - Leave pages open after accesses, or
 - Close page when there are no further accesses available in the controller for that page, or
 - Auto-precharge with each access, with an optimization for page-close mode which leaves the page open after a flush for read-write and write-read collision cases
- Supports automatic SDRAM power-down entry and exit caused by lack of transaction arrival for

programmable time

- Supports automatic Clock Stop, power-down entry and exit caused by lack of transaction arrival for programmable time
- Support for self-refresh entry and exit under software control
- Support for deep power-down entry and exit under software control
- Support for explicit SDRAM mode register updates under software control
- Flexible address mapper logic to allow application specific mapping of row, column, bank, and rank bits
- Programmable support for 1T for LPDDR2/DDR3 and 2T timing for DDR3
- User-selectable refresh control options:
 - Controller-generated auto-refreshes at programmable average intervals
 - In multi-rank designs, an offset can be applied to each rank's refresh timer to allow rank refreshes to expire at different times (this may increase efficiency by allowing traffic to continue to other ranks while a given rank is being refreshed)
 - Ability to group up to 8 controller-generated refreshes together to be issued consecutively (this reduces the frequency of page closings, increasing overall efficiency)
 - When controller-generated refreshes are grouped, some refreshes can be issued speculatively when the controller is idle for a programmable period of time
 - Ability to disable controller-generated auto-refreshes
 - Input to explicitly request a refresh
 - When LPDDR2 is used, user-selectable ability to perform per-bank refreshes rather than all-banks refreshes
- Advanced power-saving design includes no unnecessary toggling of command, address, and data pins (**RAS/CAS/WE/BA/A hold** last state after each command; DQ does not transition on writes when bytes are disabled)
- **Optional ECC support, when controller is configured for 32-bit SDRAM data width, including the following:**
 - Support for single error correction / double error detection (SEC/DED):

- 32 data bits + 7 check bits in full bus width mode
- 16 data bits + 6 check bits in half bus width mode
- 8 data bits + 5 check bits in quarter bus width mode
- Direct access to full DDR bus, including ECC bits (HIF interface configurations only), for testability mode support
- Automatic data scrubbing of correctable errors
- Support for highly efficient read-modify-write when byte enables are used with ECC enabled
- Automatic logging of both correctable and uncorrectable errors
- Ability to “poison” the write data by adding uncorrectable errors, for use in testing ECC error handling
- **Support for industry standard UDIMMs (Unbuffered DIMMs) and RDIMMs (Registered DIMMs)**
- Low-area, low-power architecture
- 5-clock-cycle typical command latency through the uMCTL2 (HIF interface)
 - Can be reduced to 4 cycles by choosing not to register DFI outputs (configuration parameter)
 - 3 clock cycles for high priority read
- Leverages out of order requests with CAM to maximize throughput
- APB interface for the uMCTL2 software accessible registers
- **3 host ports using AMBA AXI3**
- For host ports with the AXI interface:
 - Compatibility with the AMBA 3 AXI protocol
 - Support for AXI burst types: incremental and wrap
 - AXI clock asynchronous/synchronous to the controller clock
 - Exclusive access support
 - Read reorder buffer with reduced latency options (for example bypass)

Block Diagram of LPDDR2/DDR3 Controller



Description of the Block Diagram

LPDDR2/DDR3 Memory Controller (uMCTL2)

The Memory Controller has one APB slave interface to configure the Registers. It also have 3 AXI (Application host ports)slave port interfaces to connect to the SoC core. **The memory controller receives read requests, write requests, and data through a single/multi- port host interface.** The Controller consists of major components such as AXI Port Interface (XPI), Port Arbiter(PA) and DDRC.

AXI Port Interface(XPI)

This block provides the interface to the application ports. It provides bus protocol handling, data buffering, data bus size conversion(upsizing or downsizing), memory burst address alignment and memory page boundary crossover functionality.

Port Arbiter(PA)

This block provides the latency sensitive, priority based arbitration between the addresses issued by the XPIs (by the ports).

DDR Controller (DDRC)

The DDR Controller has a logical CAM which holds the information on the commands, which is used by the scheduling algorithms to optimally schedule commands to be sent to the PHY based on priority, bank/rank status and DDR timing constraints. A bypass path is optionally provided to bypass the commands. The Response engine is used to send the read data in the order of scheduled read commands on the HIF.

The two SRAMs : Write Data SRAM which is connected outside the controller is used to store the data until its associated command is issued to the PHY.

Read Reorder Buffer is also an SRAM for storing the read data and returned in the order, to the AXI ports.

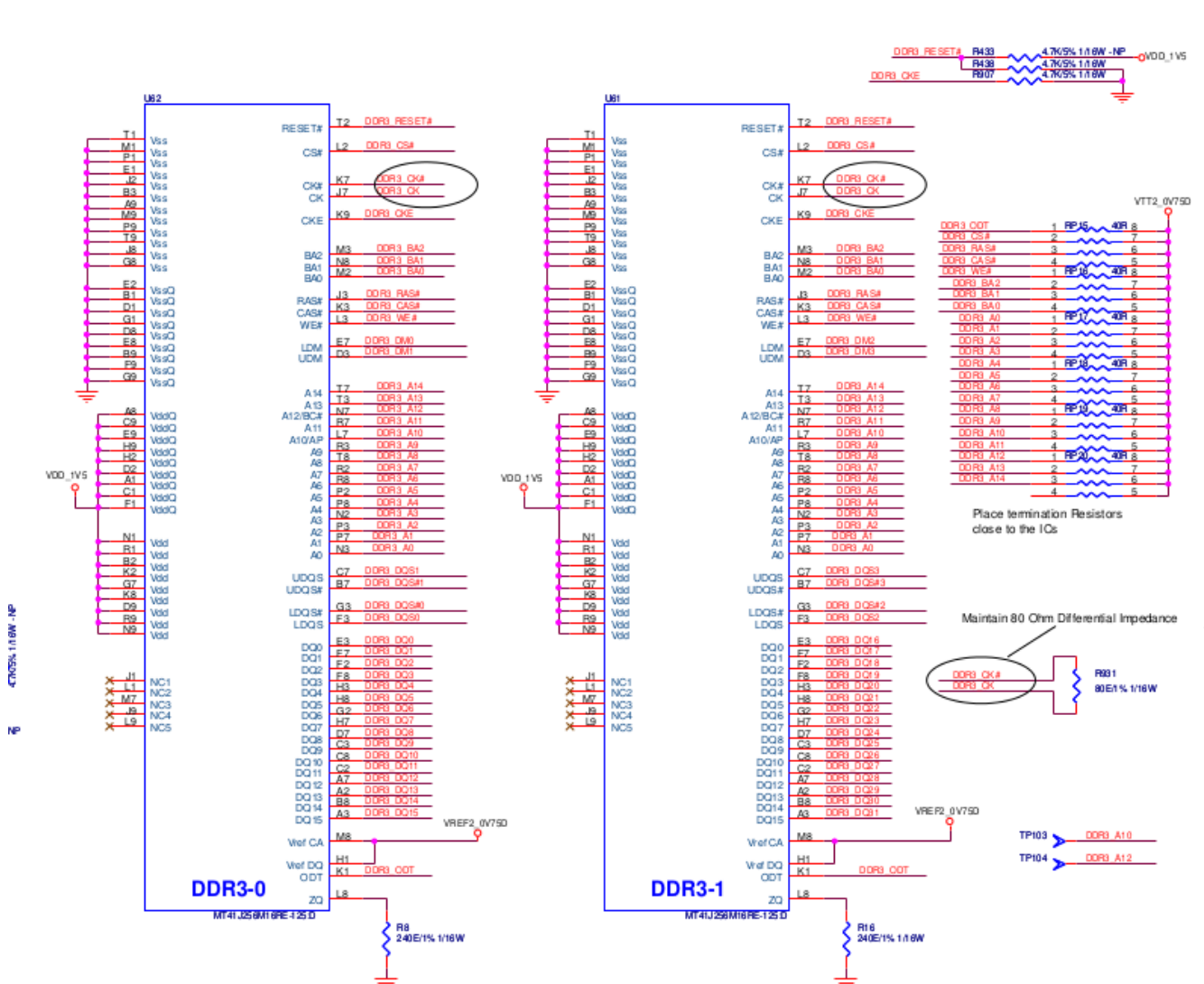
DDR Multi-PHY

The memory controller and DDR multi-PHY blocks combine to create a complete solution for connecting an SoC application bus to DDR2 memory devices. The combined solution enables the highest DDR performance and bandwidth, with the memory controller initiating DDR commands to transfer data with the maximum efficiency.

The PHY consists of Interface Timing Modules, DLLs and DDR-specific SSTL I/Os. It is a mixture of **Soft-IP and Hard-IP design elements**. The multi-PHY is lane based architecture (Byte Lane, Command Lane). The data bus interface to the external memory is organized into self-contained units

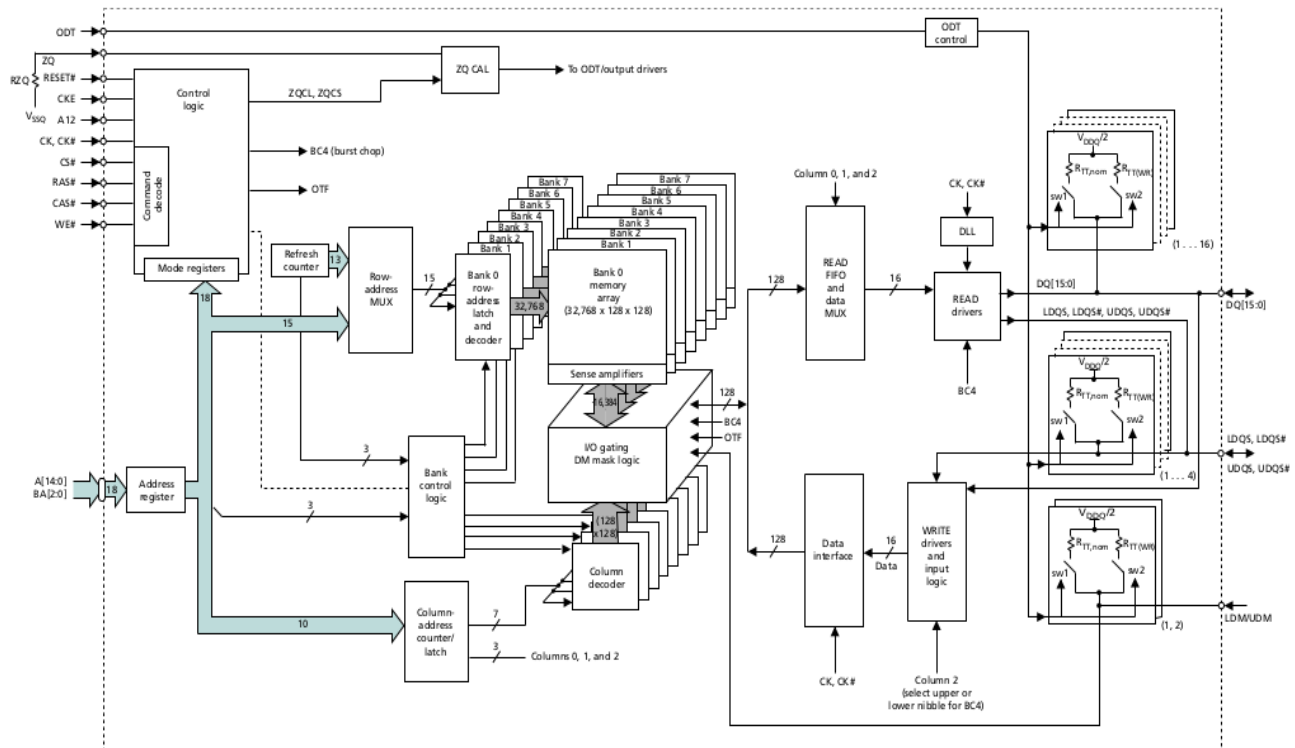
known as Byte Lanes. The number of Byte-Lanes in the PHY is dependent on the data bus width of SDRAM ie., one Byte Lane for every 8-bits of data. The Byte Lane comprises of 8-bit data and its corresponding DQS signals. The Command Lane comprises of Address, Clocks and Control signals.

DDR INTERFACE



DDR3	$V_{DD} = V_{DDQ} = 1.5V \pm 0.075V$
CLOCK	Differential clock inputs (CK, CK#)
BANKS	8 internal banks
ODT	Nominal and dynamic on-die termination (ODT) for data, strobe, and mask signals
Prefetch	8n-bit prefetch architecture
Parameter	256 Meg x 16
Configuration	32 Meg x 16 x 8 banks
Refresh count	8
Row addressing	32K (A[14:0])
Column addressing	1K (A[9:0])
Page size	2KB
Bank addressing	8 (BA[2:0])

Figure 5: 256 Meg x 16 Functional Block Diagram

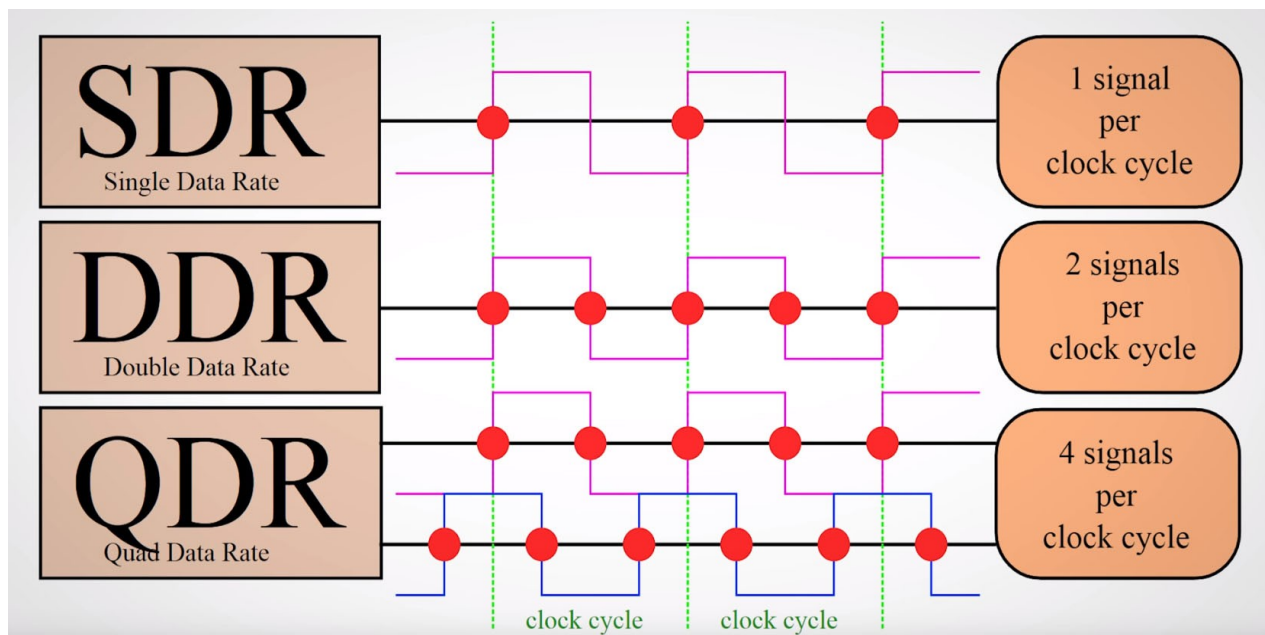


DQS, DQS#	I/O	Data strobe: Output with read data. Edge-aligned with read data. Input with write data. Center-aligned to write data.
RAS#, CAS#, WE#	I	Command inputs: RAS#, CAS#, and WE# (along with CS#) define the command being entered
CK, CK#	I	Clock: CK and CK# are differential clock inputs. All control and address input signals are sampled on the crossing of the positive edge of CK and the negative edge of CK#.
CKE	I	Clock enable: CKE enables (registered HIGH) and disables (registered LOW) internal circuitry and clocks on the DRAM
CS#	I	Chip select: CS# enables (registered LOW) and disables (registered HIGH) the command decoder
DM	I	Input data mask: DM is an input mask signal for write data. Input data is masked when DM is sampled HIGH along with the input data during a write access.

Understanding RAM Timings

DDR, DDR2, and DDR3 memories are classified according to the maximum speed at which they can work, as well as their timings. **RAM Timings** are numbers such as **3-4-4-8, 5-5-5-15, 7-7-7-21, or 9-9-9-24**, the lower the better

For instance, DDR400 memories work at 400 MHz at the most, DDR2-800 can work up to 800 MHz, and DDR3-1333 can work up to 1,333 MHz. It is important to note that this is not the real clock speed of the memory. The real clock of the DDR, DDR2, and DDR3 memories is half of the labeled clock speed. Therefore **DDR400 memories work at 200 MHz, DDR2-800 memories work at 400 MHz, and DDR3-1333 memories work at 666 MHz.**



The operations that these numbers indicate are the following: CL-tRCD-tRP-tRAS-CMD. To understand them, bear in mind that the memory is internally organized as a matrix, where the data are stored at the intersection of the lines and columns.

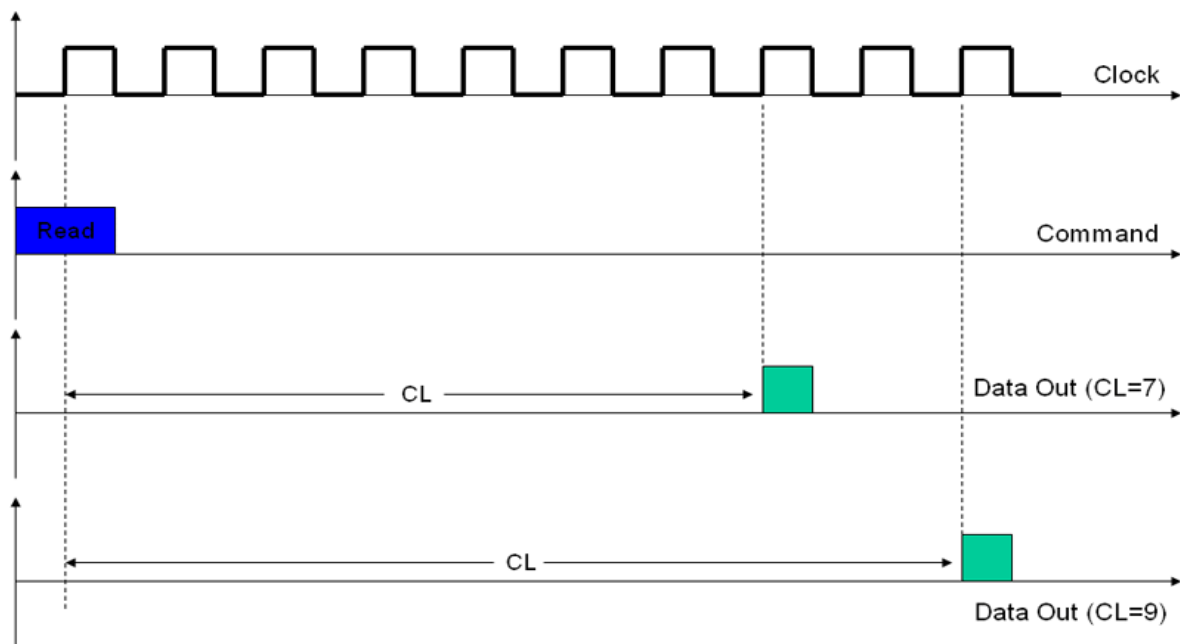
- **CL: CAS Latency.** The time it takes between a command having been sent to the memory and when it begins to reply to it. It is the time it takes between the processor asking for some data from the memory and then returning it.
- **tRCD: RAS to CAS Delay.** The time it takes between the activation of the line (RAS) and the column (CAS) where the data are stored in the matrix.
- **tRP: RAS Precharge.** The time it takes between disabling the access to a line of data and the beginning of the access to another line of data.
- **tRAS: Active to Precharge Delay.** How long the memory has to wait until the next access to the memory can be initiated.
- **CMD: Command Rate.** The time it takes between the memory chip having been activated and when the first command may be sent to the memory. Sometimes this value is not announced. It usually is T1 (1 clock cycle) or T2 (2 clock cycles).

CAS Latency (CL) Impact on RAM Speed

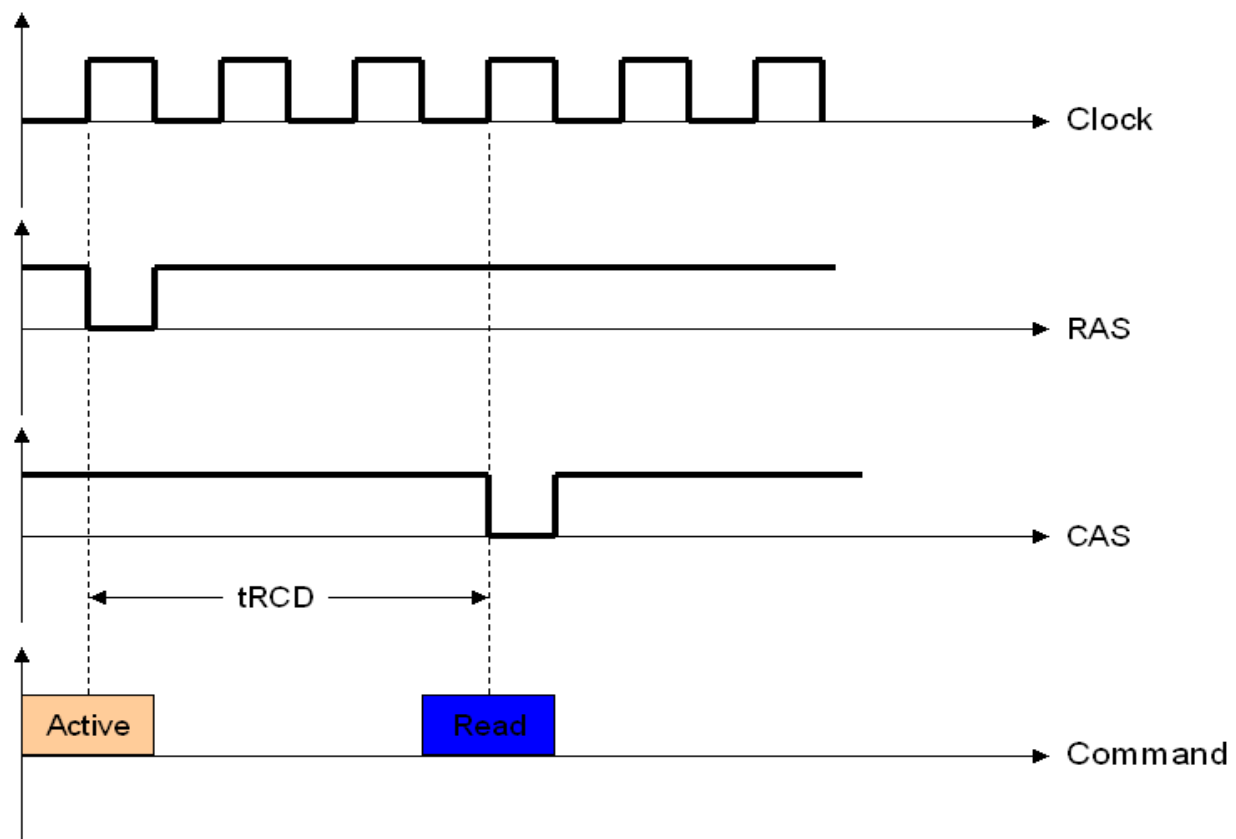
As previously mentioned, CAS Latency (CL) is the best known memory parameter. It tells us how many clock cycles the memory will delay to return requested data. A memory with CL = 7 will delay seven clock cycles to deliver data, while a memory with CL = 9 will delay nine clock cycles to perform the same operation. Thus, for two memory modules running at the same clock rate, the one with the lowest CL will be faster.

Notice that the clock rate here is the real clock rate under which the memory module is running – i.e., half the rated clock rate. As DDR, DDR2, and DDR3 memories can deliver two data per clock cycle, they are rated with double their real clock rate.

you can see how CL works. We gave two examples, a memory module with CL = 7 and a memory module with CL = 9. The command in blue would be a “read” command.



RAS to CAS Delay (tRCD) Impact on RAM Speed

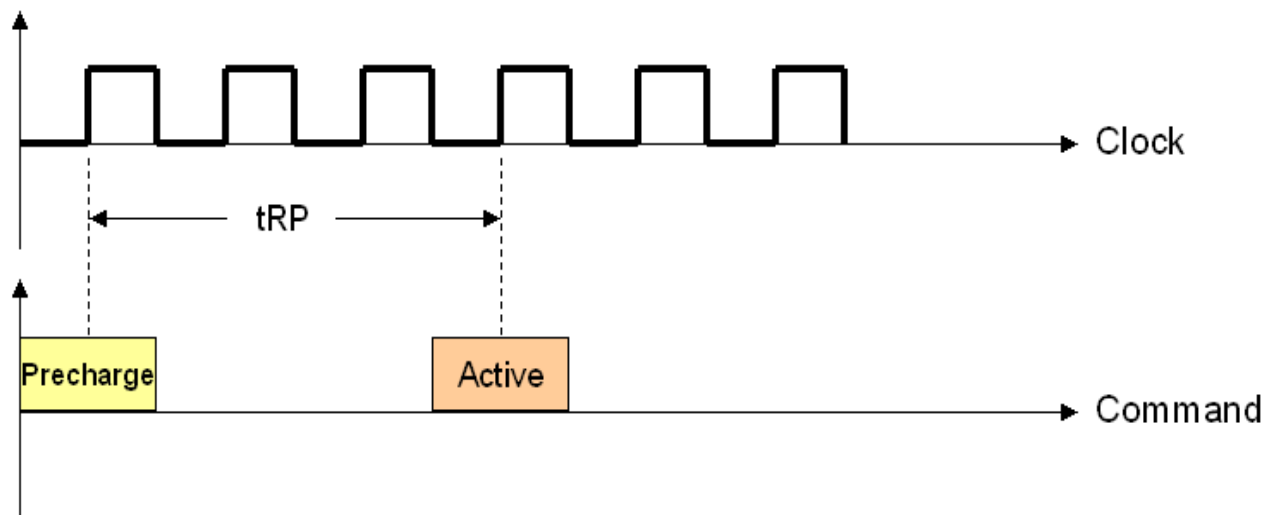


Each memory chip is organized internally as a matrix. At the intersection of each row and column we have a small capacitor that is in charge of storing a “0” or a “1” – the data. Inside the memory, the process of accessing the stored data is accomplished by first activating the row then the column where it is located. This activation is done by two control signals called RAS (Row Address Strobe) and CAS (Column Address Strobe). The less time there is between these two signals the better, as the data will be read sooner. RAS to CAS Delay or tRCD measures this time. In Figure we illustrate this, showing a memory with tRCD = 3.

As you can see, RAS to CAS Delay is also the number of clock cycles taken between the “Active” command and a “read” or “write” command.

As with CAS Latency, RAS to CAS Delay works with the memory real clock (which is half of the labeled clock). The lower this parameter, the faster the memory will be, as it will start reading or writing data earlier

RAS Precharged (tRP) Impact on RAM Speed



After data is gathered from the memory, a command called Precharge needs to be issued, closing the memory row that was being used and allowing a new row to be activated. RAS Precharge time (tRP) is the time taken between when the Precharge command and the next Active command can be issued. As we learned from the previous page, the Active command starts a read or write cycle.

In Figure , we are giving an example of a memory with $tRP = 3$.

As with the other parameters, RAS Precharge works with the memory real clock (which is half of the labeled clock). The lower this parameter, the faster the memory will be, as it will issue the Active command earlier.

Adding everything we've seen, the time elapsed between issuing the Precharge command and actually getting the data will be $tRP + tRCD + CL$.

Name	Symbol	Definition
CAS latency	CL	The number of cycles between sending a column address to the memory and the beginning of the data in response. This is the number of cycles it takes to read the first bit of memory from a DRAM with the correct row already open. Unlike the other numbers, this is not a maximum, but an exact number that must be agreed on between the memory controller and the memory.
Row Address to Column Address Delay	T_{RCD}	The minimum number of clock cycles required between opening a row of memory and accessing columns within it. The time to read the first bit of memory from a DRAM without an active row is $T_{RCD} + CL$.
Row Precharge Time	T_{RP}	The minimum number of clock cycles required between issuing the precharge command and opening the next row. The time to read the first bit of memory from a DRAM with the wrong row open is $T_{RP} + T_{RCD} + CL$.
Row Active Time	T_{RAS}	The minimum number of clock cycles required between a row active command and issuing the precharge command. This is the time needed to internally refresh the row, and overlaps with T_{RCD} . In SDRAM modules, it is simply $T_{RCD} + CL$. Otherwise, approximately equal to $T_{RCD} + 2 \times CL$.

ACT:	Activate the row and place it into row buffer
READ/WRITE:	Reads or writes a column
PRECHARGE:	Close the row

9. MIPI DSI HOST CONTROLLER

Overview

The Display Serial Interface (DSI) is part of a group of communication protocols defined by the MIPI Alliance. The Display Serial Interface (DSI) specification defines protocols between a host processor and peripheral such as a mobile device. The DSI specification builds on existing standards by adopting pixel formats and command set defined in MIPI Alliance standards for DBI-2 [2], DPI-2 [3], and DCS [1]. By standardizing this interface, components may be developed that provide higher performance, lower power and fewer pins than current devices.

1. DBI – NOT USING IN CONFIGURATION
2. DPI
3. DCS – NOT INCLUDED IN THE DESIGN

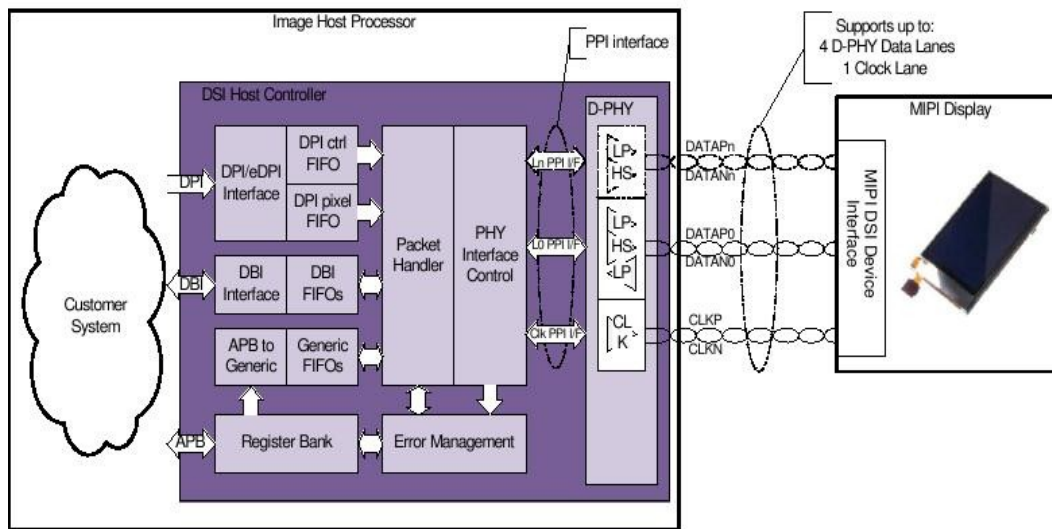
Features of DSI:

- Compliant with AMBA 2.0 Specification (APB slave) from ARM.
- DBI interface types:
 - o Type A Interface including Tearing Effect and Fixed E mode.
 - o Type A Interface including Tearing Effect and Clocked E mode.
 - o Type B Interface including Tearing Effect.
- Extended pixel clock speed beyond the DBI standard maximum clock of 20 Mhz.

DBI interface color coding mappings:

- o 8-bit Interface: 8, 12, 16, 18, and 24 bits per pixel.
 - o 9-bit Interface: 18 bits per pixel.
 - o 16-bit Interface: 8, 12, 16, 18 (options 1 and 2), and 24 (options 1 and 2) bits per pixel.
- DPI interface color coding mappings into 24-bit Interface:
 - o 16 bits per pixel, configurations 1, 2, and 3.
 - o 18 bits per pixel, configurations 1 and 2.
 - o 24 bits per pixel.
- Supports resolution up to 1920x1200.
- Co-existence of DBI and DPI interfaces with only one being operational.
- Up to four D-PHY Data Lanes.
- Bidirectional communication and escape mode support through data lane 0.
- Transmission of commands in Low-Power in Video mode.
- ECC and Checksum capabilities.
- Ultra Low-Power mode.
- Fault recovery schemes.

Block Diagram of DSI



Block Diagram of DSI

DPI-Display Pixel Interface is used to send real time pixel information from the host processor to the peripheral. It allows pixel data bus width upto 24 bits. The DPI interface captures the data and control signals and conveys them to the FIFO interfaces that transmit them to the DSI link. DPI interface does not operate concurrently with DBI interface.

DBI-The DBI interface follows the MIPI DBI specification. It is used to transmit the information in Command mode. Here, the transactions primarily take the form of sending the commands defined in the DCS specification to a peripheral, such as a display module that incorporates a display controller. The DBI interface supports the Command mode devices where the transactions are carried out through the commands as defined in DCS specification.

APB- APB slave interface is used for the register configuration. The APB Slave interface allows the transmission of generic information in Command mode, and follows the Synopsys proprietary register interface.

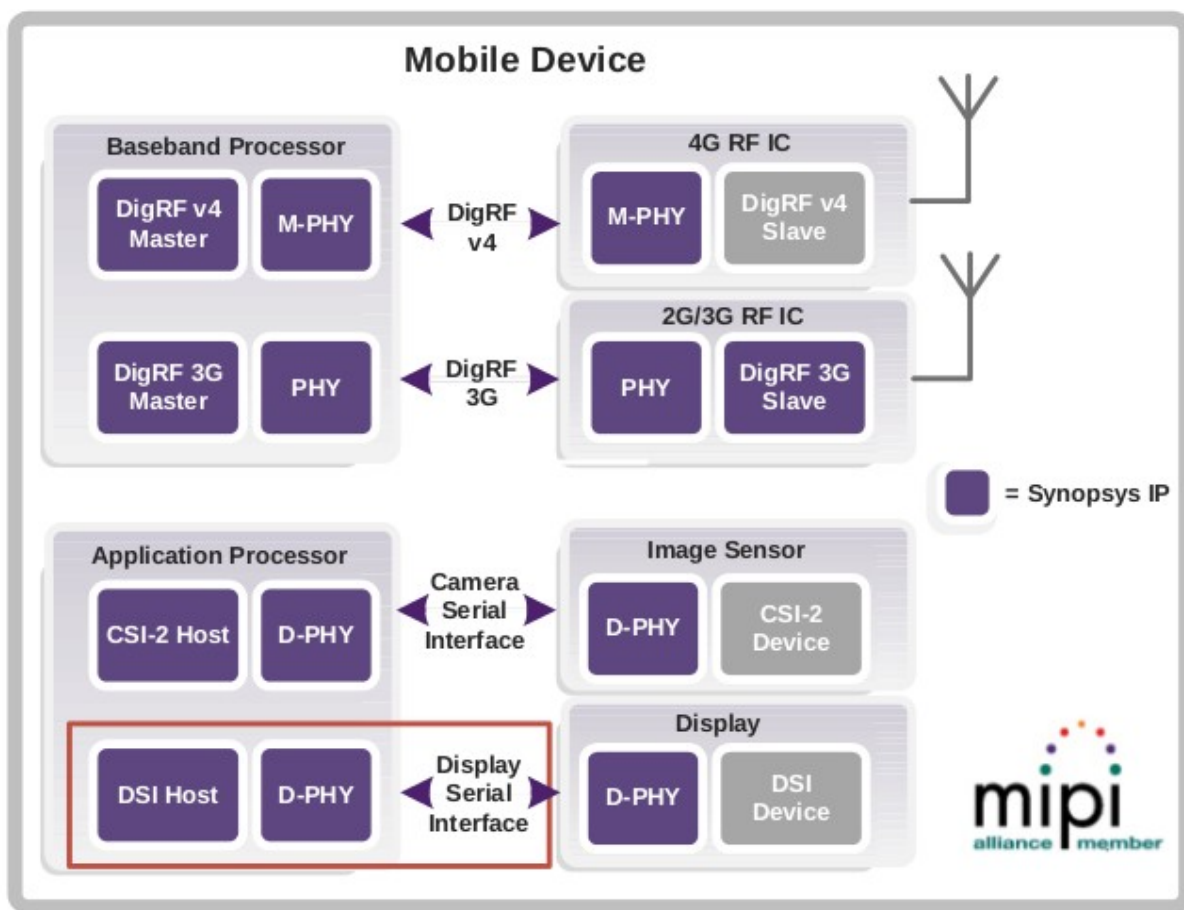
The DPI interface captures the data and control signals and conveys them to the FIFO interfaces that transmit them to the DSI link. Packet handler schedules the activities inside the link. Error management notifies and monitors error conditions on the DSI link. The PHY interface control manages the D-PHY PPI Interface. PPI is the interface between the host controller and the D-PHY. The D-PHY is from

DesignWare Cores MIPI DSI Host Controller

The Display Serial Interface (DSI) is part of a group of communication protocols defined by the MIPI.

The DWC_mipi_dsi_host provides an interface between the system and the MIPI D-PHY, allowing the communication with a DSI-compliant display.

DWC_mipi_dsi_host in System-on-Chip Example



The DWC_mipi_dsi_host conforms to the following standards:

- MIPI ® Alliance Specification for Display Serial Interface (DSI) v1.1 - 14 March 2012
- MIPI ® Alliance Standard for Display Bus Interface v2.00 (DBI-2) - 16 Nov 2005

- MIPI ® Alliance Standard for Display Pixel Interface v2.00 (DPI-2) - 23 Jan 2006
- MIPI ® Alliance Specification for D-PHY v1.1 - 16 Dec 2011
- AMBA 2.0 Specification (APB) from ARM

Operational Model Overview

The DWC_mipi_dsi_host provides means for seamless integration with Synopsys D-PHYs through coreConsultant. Optionally, the DWC_mipi_dsi_host can be configured for a non-Synopsys D-PHY. In such a configuration, it exhibits a PPI-compliant interface to connect to a D-PHY.

The DWC_mipi_dsi_host includes dedicated video interfaces and a generic APB interface that can be used to transmit information to the display. These interfaces are as follows:

- **DPI interface:** This interface follows the MIPI DPI specification. It is used to transmit information in **Video mode, in which the transfers from the host processor to the peripheral take the form of a real-time pixel stream.** The DPI interface does not operate concurrently with the DBI interface.

- **eDPI interface:** This is an enhanced version of the DPI interface, customized for command mode. It is used to transmit information in full bandwidth. It can be used as a fully compliant DPI interface or as Synopsys proprietary interface. It provides support for the standard Video mode and the adapted Command mode and also for the tearing effect.

- **DBI interface:** This interface follows the MIPI DBI specification. It is used to transmit information in Command mode, where the transactions are commands defined in the DCS specification. The DBI interface does not operate concurrently with DPI interface.

- **APB slave interface:** This interface allows the **transmission of generic information in Command mode, and follows the Synopsys proprietary register interface.** This interface can operate concurrently with either DPI or DBI interfaces.

DSI-specific device. These attributes can be: display input buffering capabilities, video transmission mode (Burst or Non-Burst), Bus Turn-Around time, concurrent command mode traffic in a video mode transmission, or display device specifics

Clock Lane frequency is 500 MHz that results in a bandwidth of 1 Gbps for each data lane.

DPI Interface

The DPI interface follows the MIPI DPI-2 specification with pixel data bus width up to 24 bits. It is used to transmit the information in Video mode in which the transfers from the host processor to the peripheral take the form of a real-time pixel stream. This interface allows sending ShutDown (SD) and ColorMode (CM) commands, which are triggered directly by the pins at the interface. To transfer additional commands (for example, to initialize the display), use another interface such as APB Slave Generic Interface to complement the DPI interface.

The DPI interface captures the data and control signals and conveys them to the FIFO interfaces that transmit them to the DSI link. Two different streams of data are presented at the interface; video control signals and pixel data. Depending on the interface color coding, the pixel data is disposed differently throughout the dpixdata bus. Interface pixel color coding is shown in Figure 4-3.

The DPI interface can be configured to increase flexibility and promote correct usage of this interface for several systems. These configuration options are as follows:

Polarity control: All the control signals are programmable to change the polarity depending on system requirements.

- **After the core reset**, DPI waits for the first VSYNC active transition to start signal sampling, including pixel data, thus avoiding starting the transmission of the image data in the middle of a frame.

- **If interface pixel color coding is 18 bits and the 18-bit loosely packed stream is disabled, the number of pixels programmed in the vid_pkt_size field must be a multiple of four.** This means that in this mode, the two LSBs in the configuration are always inferred as zero. The specification states that in this mode, the pixel line size should be a multiple of four.

- **To avoid FIFO underflows and overflows**, the configured number of pixels is assumed to be received at all times. This happens even if the dpidataen pin is active for more or less time than necessary.

- **To keep the memory organized with respect to the packet scheduling**, the number of pixels per packet parameter is used to separate the memory space of different video packets.

Configuration Example

Video input timing

Resolution		800RGBx1280			768RGBx1024			Unit
Input Timing	Symbol	Min.	Typ.	Max.	Min.	Typ.	Max.	
PCLK frequency	-	-	71.9	80	-	55.8	80	MHz
Horizontal total	THT	880	920	1600	848	888	1536	DCLK
Horizontal synchronization	THS	10	24	-	10	24	-	DCLK
Horizontal back porch	THB	10	24	-	10	24	-	DCLK
Horizontal address	THA	-	800	-	-	768	-	DCLK
Horizontal front porch	THF	20	72	-	20	72	-	DCLK
Vertical frequency	-	-	60	-	-	60	-	Hz
Vertical total	TVT	1300	1304	2047	1044	1048	2047	THT
Vertical synchronization	TVS	1	2	-	1	2	-	THT
Vertical back porch	TVB	8	10	-	8	10	-	THT
Vertical address	TVA	-	1280	-	-	1024	-	THT
Vertical front porch	TVF	8	12	-	8	12	-	THT

The following is an example of DPI packet transmission configuration: DPI video resolution:

- PCLK period = 1/71.9 MHz
- HSA = 24 PCLK
- HBP = 24 PCLK
- HACT = 800 PCLK
- HFP = 72 PCLK
- VSA = 2 Line
- VBP = 10 Line
- VACT = 1280 Line
- VFP = 12 Line

Configuration steps:

1. Global configuration

n_lanes (PHY_IF_CFG)= 3, that is, Four lanes available for HS transmission.

2. DPI interface configuration

□ dpi_vcid (DPI_VCID) = 0

- ☐ dpi_color_coding (DPI_COLOR_CODING) = 7, that is, 24 bpp
- ☐ dataen_active_low (DPI_CFG_POL) = 1, that is, the signal is active low
- ☐ vsync_active_low (DPI_CFG_POL) = 1, that is, the signal is active low
- ☐ hsync_active_low (DPI_CFG_POL) = 1, that is, the signal is active low
- ☐ shutd_active_low (DPI_CFG_POL) = 0, that is, the signal is active high
- ☐ colorm_active_low (DPI_CFG_POL) = 0, that is, the signal is active high
- ☐ loosely18_en (DPI_COLOR_CODING) is irrelevant since 18 bpp color mode is not selected

3.Video transmission mode configuration:

a. Configure the low-power transitions:

VID_MODE_CFG[13:8] = 6'b111111, that is, enable LP in all video period.

b. Enable frame_bta_ack_en field (VID_MODE_CFG), that is, the DWC_mipi_dsi_host requests an acknowledge response message from the peripheral at the end of each frame.

c. If you want to use the Burst mode, follow these steps:

- vid_mode_type (VID_MODE_CFG) = 2'b1x
- vid_pkt_size (VID_PKT_SIZE) = 800

4.Horizontal timing configuration:

- ☐ vid_hline_time (VID_HLINE_TIME) = (HSA+HBP+HACT+HFP)*(PCLK period/Clk Lane Byte Period)

$$\text{vid_hline_time} = (8+8+480+24) \cdot (50/8) = 3250$$
- vid_hsa_time (VID_HSA_TIME) = HSA*(PCLK period/Clk Lane Byte Period);

$$\text{hsa_time} = 8 \cdot (50/8) = 50$$
- vid_hbp_time (VID_HBP_TIME) = HBP*(PCLK period/Clk Lane Byte Period);
- hbp_time = 8*(50/8) = 50

5.Vertical line configuration:

- ☐ vid_vsa_lines (VID_VSA_LINES) = 2
- ☐ vid_vbp_lines (VID_VBP_LINES) = 2
- ☐ vid_vfp_lines (VID_VFP_LINES) = 4
- ☐ v_active_lines (VID_VACTIVE_LINES) = 1280

Burst Mode

In this mode, the entire active pixel line is buffered into a FIFO and transmitted in a single packet with no interruptions. This transmission mode requires that the DPI Pixel FIFO has the capacity to store a full line of active pixel data inside it. This mode is optimally used if the difference between the pixel required bandwidth and DSI link bandwidth is very different. This enables the `DWC_mipi_dsi_host` to quickly dispatch the entire active video line in a single burst of data and then return to low-power mode

Guidelines for Selecting the Burst or Non-Burst Mode

Selecting the Burst and Non-Burst mode is mainly dependent on the system configuration and the device requirements. Choose the video transmission mode that suits the application scenario. The Burst mode is more beneficial because it increases the probability of the link spending more time in the low-power mode,

decreasing power consumption. However, the following conditions should be met for availing the maximum benefits from the Burst mode of operation:

- The `DWC_mipi_dsi_host` core should have sufficient pixel memory to store an entire pixel line to avoid the overflow of the internal FIFOs.
- The display device should support receiving a full pixel line in a single packet burst to avoid the overflow on the reception buffer.
- The DSI output bandwidth should be higher than the DPI system interface input bandwidth in a relation that enables the link to go to low-power once per line.

If the system cannot meet these requirements, it is likely that the pixel data will be lost causing the malfunctioning of the display device while using the Burst mode. These errors are related to the capabilities of the system to store the temporary pixel data.

APB Slave Generic Interface

The APB Slave interface allows the transmission of generic information in Command mode, and follows the Synopsys proprietary register interface. Commands sent through this interface are not constrained to comply with the DCS specification, and can include generic commands described in the DSI specification as manufacturer-specific.

The `DWC_mipi_dsi_host` supports the transmission of write and read command mode packets as described in the DSI specification. These packets are built using the APB register access. The **GEN_PLD_DATA** register has two distinct functions based on the operation. Writing to this register sends the data as payload when sending a Command mode packet. Reading this register returns the payload of a read back operation. The **GEN_HDR** register contains the Command mode packet header type and header data. Writing to this register triggers the transmission of the packet implying that for a long Command mode packet, the packet's payload needs to be written in advance in the **GEN_PLD_DATA** register.

The valid packets available to be transmitted through the Generic interface are as follows:

- Generic Write Short Packet 0 Parameters
- Generic Write Short Packet 1 Parameters
- Generic Write Short Packet 2 Parameters
- Generic Read Short Packet 0 Parameters
- Generic Read Short Packet 1 Parameters
- Generic Read Short Packet 2 Parameters
- Maximum Read Packet Configuration
- Generic Long Write Packet

A set of bits in the **CMD_PKT_STATUS** register report the status of the FIFOs associated with APB interface support. Generic interface packets are always transported using one of the DSI transmission modes; Video mode or Command mode. If neither of these modes is selected, the packets are not transmitted through the link and the related FIFOs eventually get overflowed

void TC358768_DCS_write_1A_1P(unsigned char address, unsigned char para)

```
{
    val=(MIPI_DSI_GENERIC_SHORT_WRITE_2_PARAM | address << 8 | para << 16);

    while (readl(DSI_BASE_ADDR + DSI_CMD_PKT_STATUS) & DSI_CMD_PKT_STATUS_GEN_CMD_FULL);
//wait, if it is full

    writel(val,(DSI_BASE_ADDR + DSI_GEN_HDR));

    while (((!readl(DSI_BASE_ADDR + DSI_CMD_PKT_STATUS)& DSI_CMD_PKT_STATUS_GEN_CMD_EMPTY))
&& (!readl(DSI_BASE_ADDR + DSI_CMD_PKT_STATUS)&DSI_CMD_PKT_STATUS_GEN_PLD_W_EMPTY)));
}
```

void TC358768_DCS_write_1A_0P(unsigned char address)

```
{  
    val=(MIPI_DSI_GENERIC_SHORT_WRITE_1_PARAM | address << 8);  
    while (readl(DSI_BASE_ADDR + DSI_CMD_PKT_STATUS) & DSI_CMD_PKT_STATUS_GEN_CMD_FULL);  
    //wait, if it is full  
  
    writel(val,(DSI_BASE_ADDR + DSI_GEN_HDR));  
  
    while ((!(readl(DSI_BASE_ADDR + DSI_CMD_PKT_STATUS)& DSI_CMD_PKT_STATUS_GEN_CMD_EMPTY))  
&&      (!(readl(DSI_BASE_ADDR + DSI_CMD_PKT_STATUS)&  
DSI_CMD_PKT_STATUS_GEN_PLD_W_EMPTY)))) ;  
  
}
```

void mipi_lcd_Panel_init()

```
{  
  
    TC358768_DCS_write_1A_1P(0xB0,0x00);//PAGE 0  
    TC358768_DCS_write_1A_1P(0xB1,0x67);  
  
    #if 0 //BIST Enable  
        TC358768_DCS_write_1A_1P(0xB2,0x4A); //Added for testing BIST  
        //TC358768_DCS_write_1A_1P(0xB2,0x49); //Added for testing BIST  
    #endif  
  
    TC358768_DCS_write_1A_1P(0xB3,0x48);  
    //TC358768_DCS_write_1A_1P(0xB3,0x4B);  
    TC358768_DCS_write_1A_1P(0xBB,0xE8); //lane 0  
    //TC358768_DCS_write_1A_1P(0xBB,0xEE); //NUMBER OF LANES 4  
    TC358768_DCS_write_1A_1P(0xC4,0x09);  
    TC358768_DCS_write_1A_1P(0xC6,0x09);  
    TC358768_DCS_write_1A_1P(0xCB,0x3F);  
    TC358768_DCS_write_1A_1P(0xCC,0x2D);  
    TC358768_DCS_write_1A_1P(0xCD,0x20);  
    TC358768_DCS_write_1A_1P(0xCE,0x1D);  
    TC358768_DCS_write_1A_1P(0xCF,0x0A);  
    TC358768_DCS_write_1A_1P(0xD0,0x00);  
    TC358768_DCS_write_1A_1P(0xD1,0x02);  
    TC358768_DCS_write_1A_1P(0xD2,0x0D);  
    TC358768_DCS_write_1A_1P(0xD3,0x10);  
}
```

TC358768_DCS_write_1A_1P(0xD4,0x10);
TC358768_DCS_write_1A_1P(0xD5,0x0E);
TC358768_DCS_write_1A_1P(0xD6,0x3F);
TC358768_DCS_write_1A_1P(0xD7,0x2D);
TC358768_DCS_write_1A_1P(0xD8,0x20);
TC358768_DCS_write_1A_1P(0xD9,0x1D);
TC358768_DCS_write_1A_1P(0xDA,0x0A);
TC358768_DCS_write_1A_1P(0xDB,0x00);
TC358768_DCS_write_1A_1P(0xDC,0x01);
TC358768_DCS_write_1A_1P(0xDD,0x0E);
TC358768_DCS_write_1A_1P(0xDE,0x10);
TC358768_DCS_write_1A_1P(0xDF,0x10);
TC358768_DCS_write_1A_1P(0xE0,0x0E);

TC358768_DCS_write_1A_1P(0xB0,0x02);//PAGE 2

TC358768_DCS_write_1A_1P(0xB1,0x00);
TC358768_DCS_write_1A_1P(0xB2,0x00);
TC358768_DCS_write_1A_1P(0xB3,0x10);
TC358768_DCS_write_1A_1P(0xB4,0x0E);
TC358768_DCS_write_1A_1P(0xB5,0x0C);
TC358768_DCS_write_1A_1P(0xB6,0x0C);
TC358768_DCS_write_1A_1P(0xB7,0x0A);
TC358768_DCS_write_1A_1P(0xB8,0x0A);
TC358768_DCS_write_1A_1P(0xB9,0x08);
TC358768_DCS_write_1A_1P(0xBA,0x00);
TC358768_DCS_write_1A_1P(0xBB,0x00);
TC358768_DCS_write_1A_1P(0xBC,0x00);
TC358768_DCS_write_1A_1P(0xBD,0x00);
TC358768_DCS_write_1A_1P(0xBE,0x00);
TC358768_DCS_write_1A_1P(0xBF,0x00);

TC358768_DCS_write_1A_1P(0xC0,0x08);
TC358768_DCS_write_1A_1P(0xC1,0x06);

TC358768_DCS_write_1A_1P(0xC2,0x06);
TC358768_DCS_write_1A_1P(0xC3,0x02);
TC358768_DCS_write_1A_1P(0xC4,0x04);
TC358768_DCS_write_1A_1P(0xC5,0x00);
TC358768_DCS_write_1A_1P(0xC6,0x00);
TC358768_DCS_write_1A_1P(0xC7,0x00);
TC358768_DCS_write_1A_1P(0xC8,0x00);
TC358768_DCS_write_1A_1P(0xC9,0x0F);
TC358768_DCS_write_1A_1P(0xCA,0x0D);
TC358768_DCS_write_1A_1P(0xCB,0x0B);
TC358768_DCS_write_1A_1P(0xCC,0x0B);
TC358768_DCS_write_1A_1P(0xCD,0x09);
TC358768_DCS_write_1A_1P(0xCE,0x09);
TC358768_DCS_write_1A_1P(0xCF,0x07);

TC358768_DCS_write_1A_1P(0xD0,0x00);
TC358768_DCS_write_1A_1P(0xD1,0x00);
TC358768_DCS_write_1A_1P(0xD2,0x00);
TC358768_DCS_write_1A_1P(0xD3,0x00);
TC358768_DCS_write_1A_1P(0xD4,0x00);
TC358768_DCS_write_1A_1P(0xD5,0x00);
TC358768_DCS_write_1A_1P(0xD6,0x07);
TC358768_DCS_write_1A_1P(0xD7,0x05);
TC358768_DCS_write_1A_1P(0xD8,0x05);
TC358768_DCS_write_1A_1P(0xD9,0x01);
TC358768_DCS_write_1A_1P(0xDA,0x03);
TC358768_DCS_write_1A_1P(0xDB,0x00);
TC358768_DCS_write_1A_1P(0xDC,0x00);

TC358768_DCS_write_1A_1P(0xB0,0x03); //PAGE 3

TC358768_DCS_write_1A_1P(0xC4,0x07);
TC358768_DCS_write_1A_1P(0xC6,0x05);
TC358768_DCS_write_1A_1P(0xC7,0x01);

```

TC358768_DCS_write_1A_1P(0xD0,0x00);
TC358768_DCS_write_1A_1P(0xD1,0x04);
TC358768_DCS_write_1A_1P(0xD2,0x04);
TC358768_DCS_write_1A_1P(0xD3,0x01);
TC358768_DCS_write_1A_1P(0xD4,0x02);
TC358768_DCS_write_1A_1P(0xD5,0x03);
TC358768_DCS_write_1A_1P(0xD6,0x04);

//TC358768_DCS_write_1A_1P(0xB0,0x06);//PAGE 6
//TC358768_DCS_write_1A_1P(0xB8,0x85);//BTA Enable : BTA->TX procedure function enable.
TC358768_DCS_write_1A_1P(MIPI_DCS_SET_PIXEL_FORMAT,0x70); //RGB24 bit format

TC358768_DCS_write_1A_0P(0x11);
udelay(200000);
TC358768_DCS_write_1A_0P(0x32);
TC358768_DCS_write_1A_0P(0x29);
udelay(200000);
printf("TRULY LCD Panel initialization is done properly\n");
writel(0x1000,DSI_CLKMGR_CFG + DSI_BASE_ADDR);
}

```

There are two signaling modes in the DSI physical layer:

- **High-speed (HS) mode:** used for fast data transmission where the lane signals are used in differential mode with speeds up to 1.5 Gbit/s.
- **Low-power (LP) mode:** used mainly for control purposes. The pair of signals of the lane may be driven independently in a single ended mode with a maximum transfer rate of 10 Mbit/s.

Data lane states

The DSI transmitter has one HS transmitter that drives the lines differentially to **HS-0 or HS-1** and **two LP transmitters that drive each line of the differential pair (DP and DN) independently in single ended mode**. This results in two possible lane states for the HS transmitter (**HS-0 and HS-1**) and four lanes states for the LP transmitters (**LP-00, LP-10, LP-01, LP-11**)

Table 4. Data lane states and operating mode

State code	D _P line level	D _N line level	High-speed burst mode	Low-power control mode	Low-power escape mode
HS-0	HS low	HS high	Differential 0	x	x
HS-1	HS high	HS low	Differential 1	x	x
LP-00	LP low	LP low	x	Bridge	Space
LP-01	LP low	LP high	x	HS-Rqst	Mark-0
LP-10	LP high	LP low	x	LP-Rqst	Mark-1
LP-11	LP high	LP high	x	STOP	(return to stop)

Data lane operating modes

There are three operating modes for data lanes: **control, high-speed transmission and escape**.

Control mode

After reset, the data lanes are in **control mode (LP-11 stop state)**. **All other modes start and end to control mode**

High-speed transmission mode

A **high-speed transmission starts with and ends to stop state (LP-11)**.

To enable synchronization between host and display, a **leader and trailer sequences are added**. **They are removed on the receiver side since they are not part of the actual payload data**.

Start-of-transmission (SoT) procedure

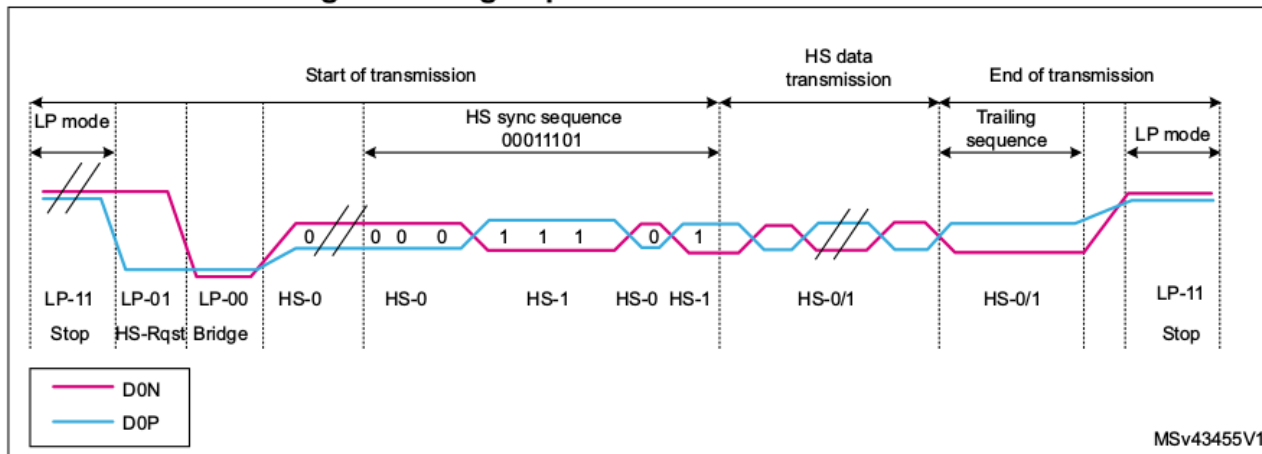
Upon reception of the HS request (LP-11, LP-01, LP-00) the data lane enters the HS mode. The host starts by driving HS-0, then drives HS sync sequence (00011101) to allow the slave synchronization. Then the host continues the transmission of the HS data.

End-of-transmission (EoT) procedure

After the end of the HS burst, the host sends a trailing sequence.

The trailing sequence is the opposite of the last data bit transmitted: if the last payload bit is HS-0 then the transmitter sends HS-1 as trailing sequence, otherwise it sends HS-0. The data lane returns to control mode via the stop state LP-11.

Figure 11. High-speed data transmission mode



Escape mode

The data lane may enter the escape mode via the escape mode request procedure (LP- 11,LP-10,LP-00,LP-01,LP-00).

After entering the escape mode, the transmitter sends an 8-bit entry command to indicate the requested action.

Escape entry command may be:

- Low-power data transmission (LPDT)

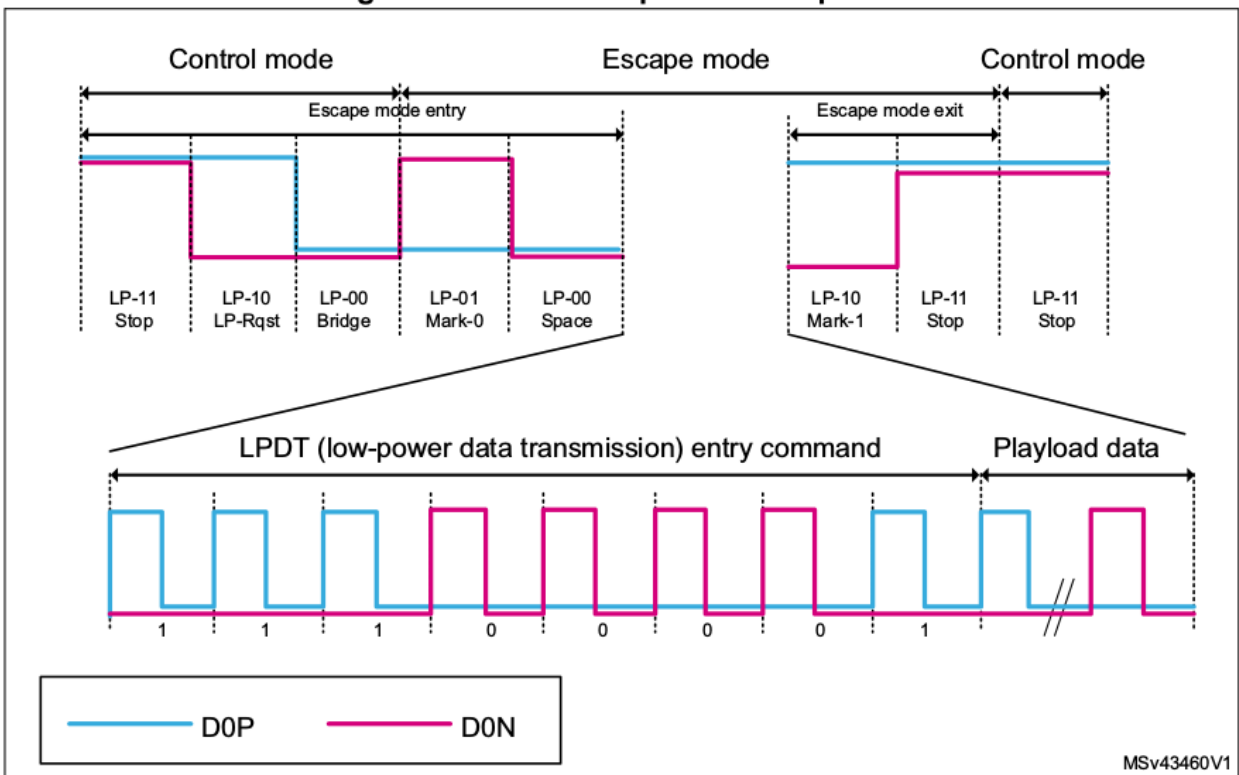
- Ultra low-power state (ULPS)
- Remote triggers

Table 5. Escape mode commands

Escape mode action	Command type	Entry command pattern
Low-power data transmission	Mode	11100001
Ultra-low-power state	Mode	00011110

The escape mode is exit through the escape mode exit procedure (**LP-10, LP-11**).

Figure 16. LPDT escape mode sequence



Data is sent in LSB (least significant bit) first, and for multibyte payload, the least significant byte is transferred first.

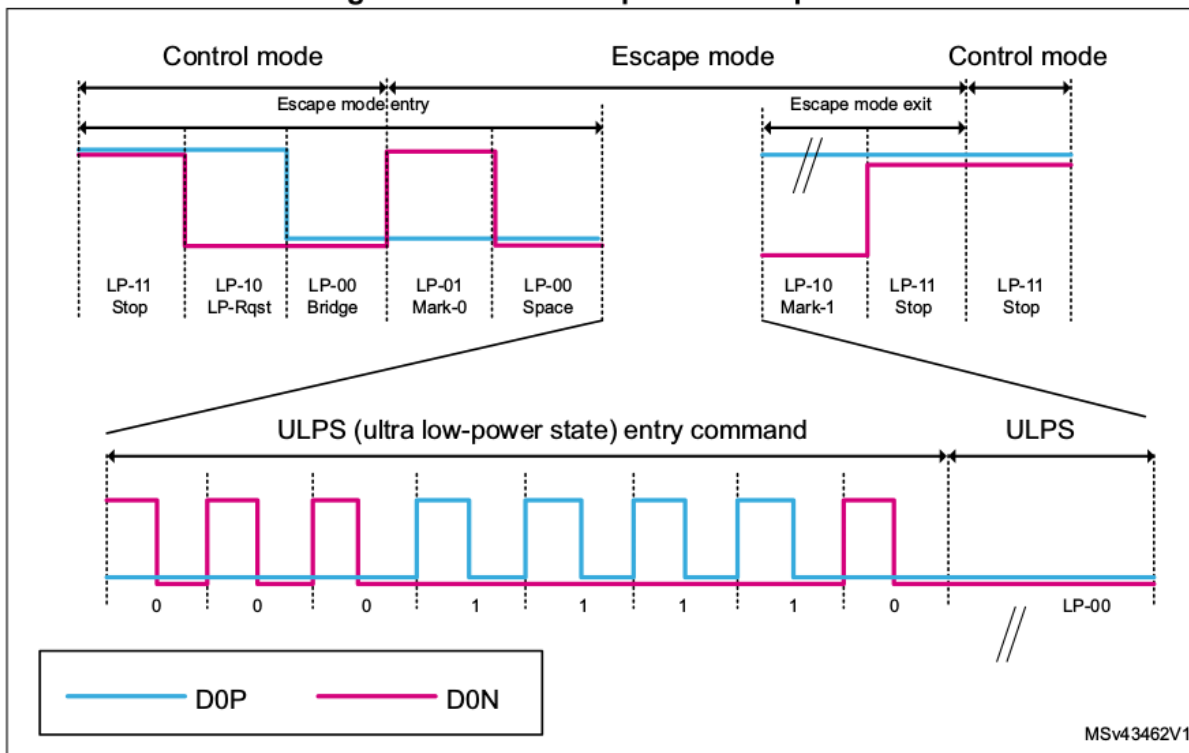
Ultra low-power state

In order to reduce the power consumption, the DSI Host may put the data lanes into the ultra low-power state (ULPS). This is achieved by the following procedure:

- Enter escape mode (LP-11,LP-10,LP-00,LP-01,LP-00).
- Send ULPS entry command (00011110).
- Keep lane signals into LP-00 state.

The ULPS state is exited with Mark-1 (LP-10) followed by stop state (LP-11).

Figure 18. ULPS escape mode sequence



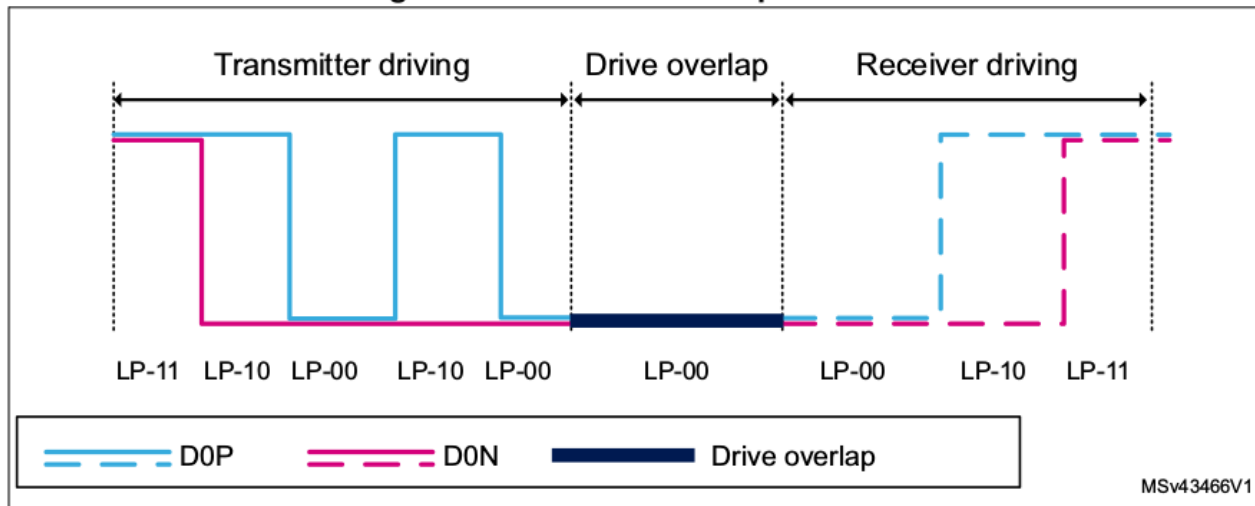
Bidirectional lanes and bus turnaround procedure

The DSI supports bidirectional data links only on data lane 0. To allow a reverse transmission, the data lane direction can be swapped using the bus turnaround (BTA) procedure.

The BTA is started from the stop state (LP-11). After the receiver has the bus ownership, the reverse transmission may begin. Then the receiver must give back the bus ownership to the

host through the same turnaround procedure

Figure 22. Bus turnaround procedure



Clock-lane power modes

The DSI CLK lanes can be driven into three different power modes: **low-power mode**, **ultra low-power state** and **high-speed clock mode**.

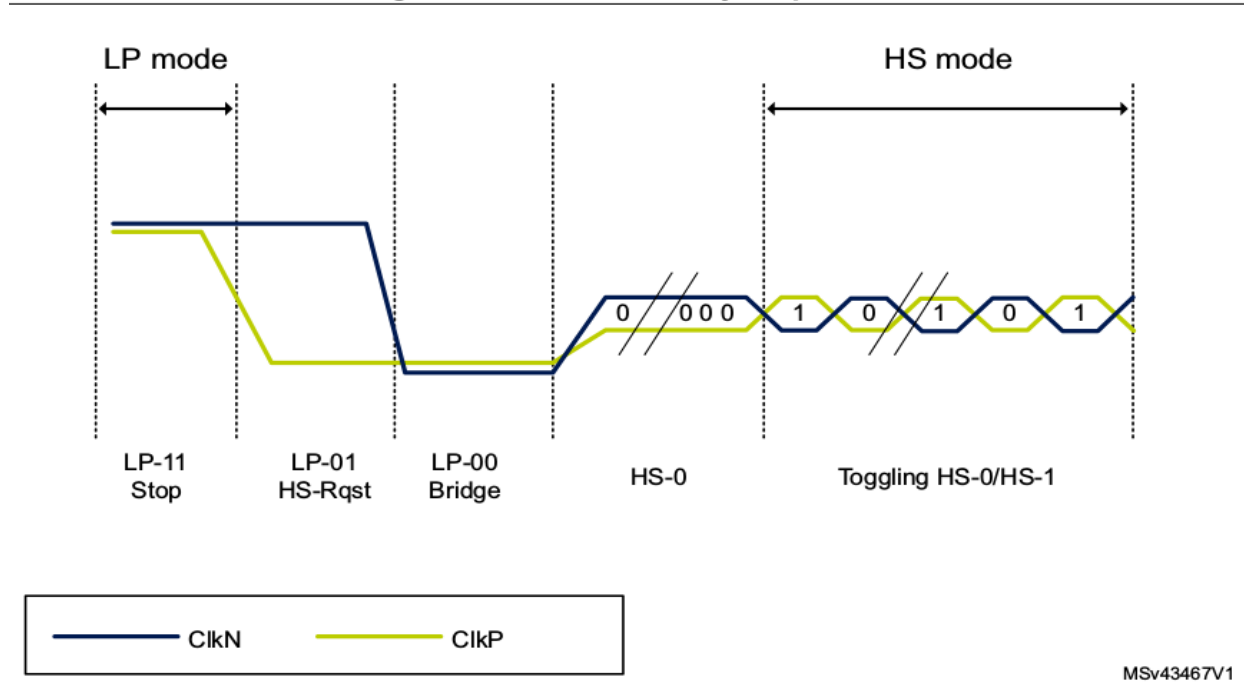
Low-power mode

During the low-power mode, the clock lane is in the LP-11 stop state. All other modes start from and ends to the LP mode.

High-speed mode

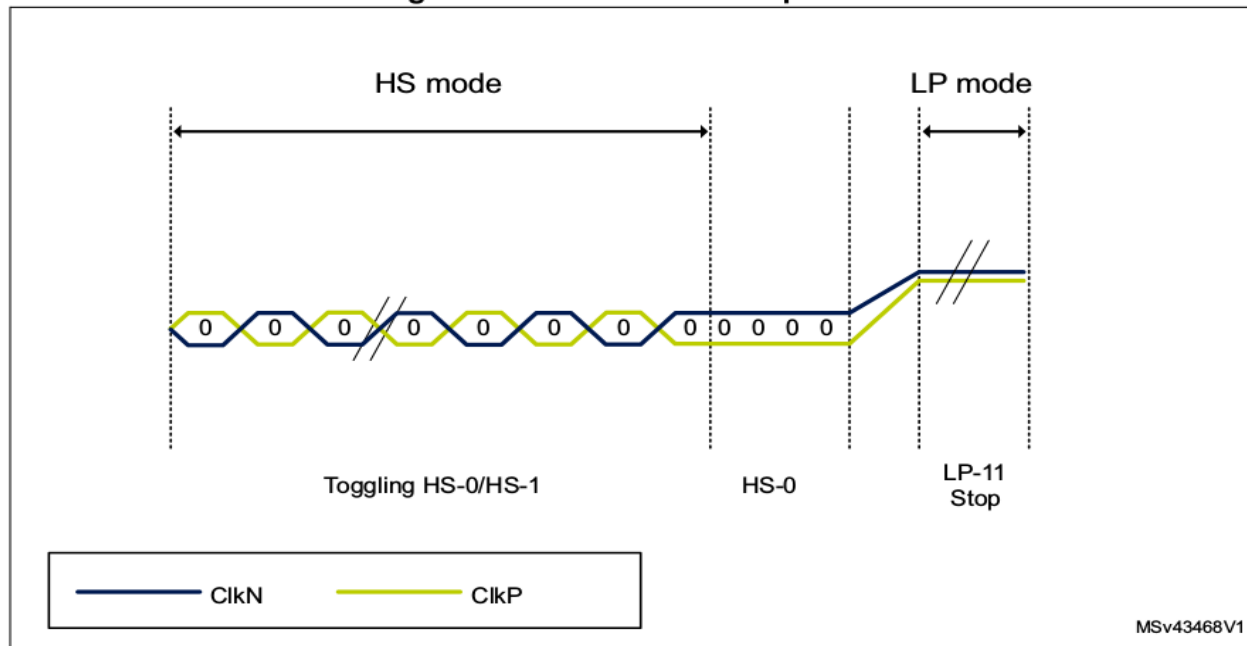
The clock lane enters the HS mode starting from the LP mode by driving a HS entry sequence (LP-11,LP-01,LP-00,HS-0). After that, the clock lane enters the HS mode and starts toggling HS-0,HS-1.

Figure 23. Clock HS entry sequence



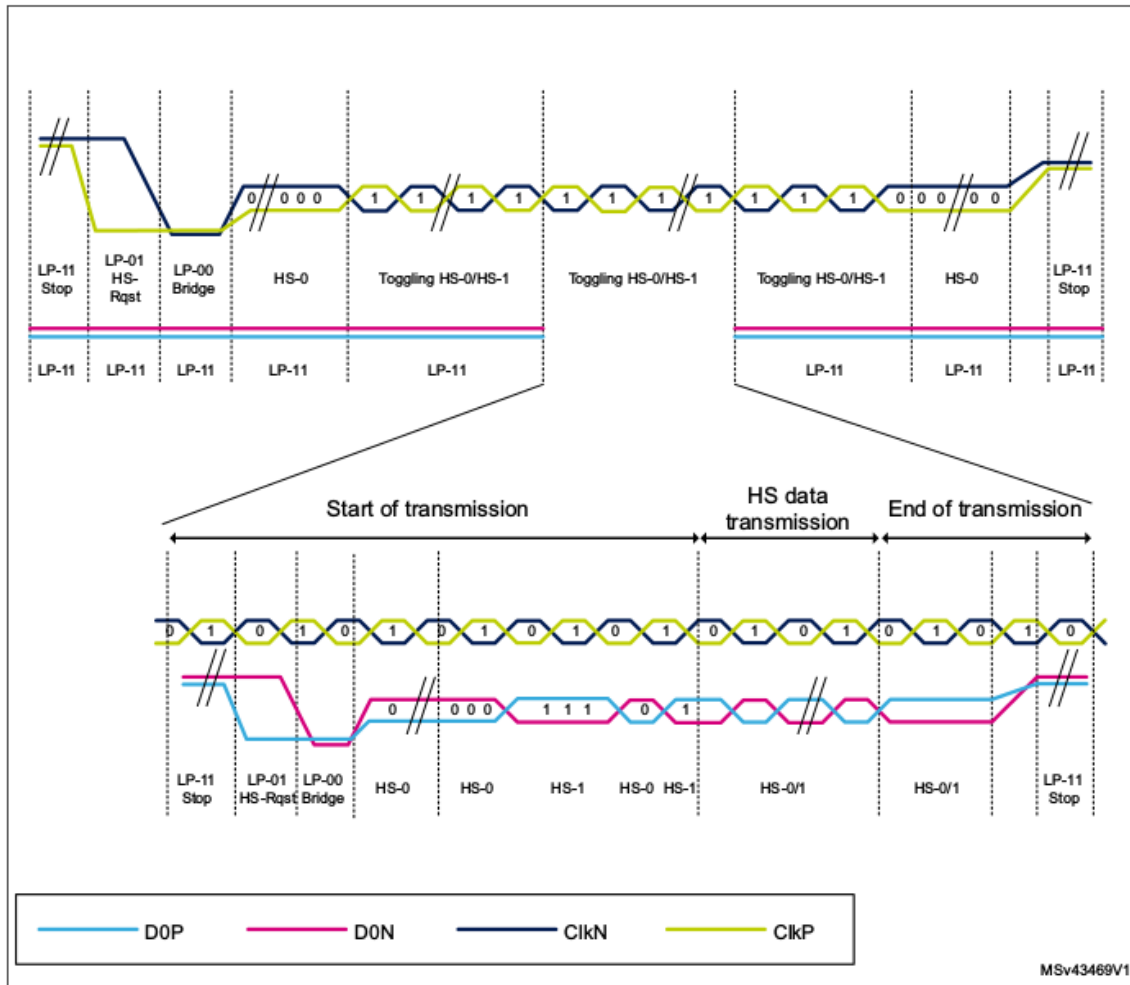
The clock lane leaves the HS mode through the exit sequence (HS-0,LP-11)

Figure 24. Clock HS exit sequence



The high-speed clock is started before that the high-speed data is sent via the data lanes. The high-speed clock continues clocking after the high-speed data lane has stopped.

Figure 25. Clock and data lanes relationship in HS mode

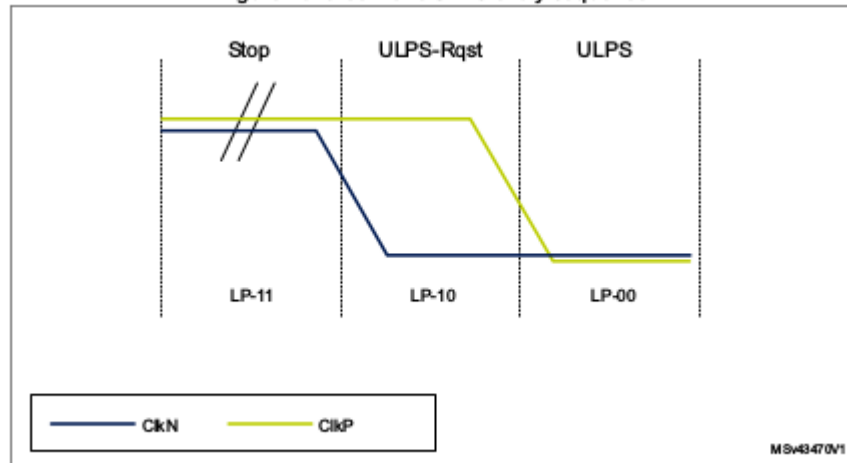


MSv43469V1

Ultra-low-power state (ULPS)

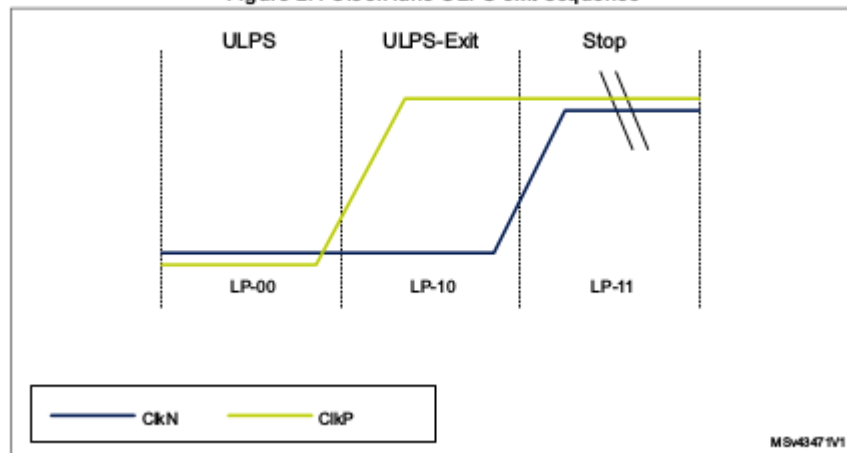
The data lane supports the escape mode while the clock lane does not support it. The clock lane supports however the ULPS (which is a subset of the escape mode). The clock lane may enter the ULPS starting from stop state using the ULPS entry sequence.

Figure 26. Clock lane ULPS entry sequence



The clock lane leaves the ULPS state towards the LPM using the ULPS exit sequence (LP-00, LP-10, LP-11) as shown in [Figure 27](#).

Figure 27. Clock lane ULPS exit sequence



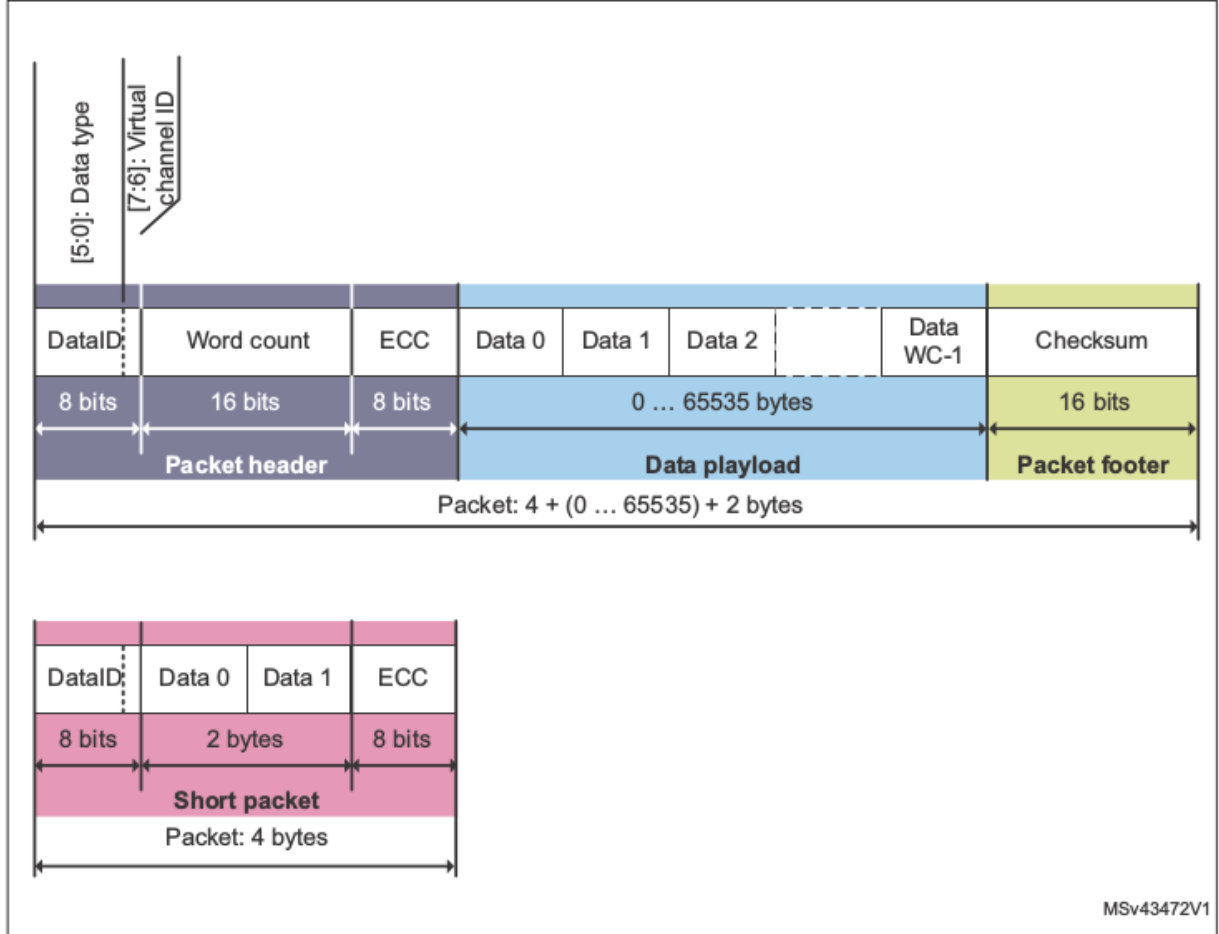
10.DSI protocol

The DSI is a packet based protocol. The parallel data and the commands are encapsulated into packets and upended with packet-protocol information and headers.

Packet structure

Two packet structures are defined for the low-level protocol communication: **long packets and short packets**

Figure 28. Short and long packet structures



Long packet

The long packets are mainly used for large blocks of data transmission such as pixel data. They are composed of three parts: packet header (PH), payload data, and packet footer (PF).

The 32-bit packet header contains:

- 8-bit data ID.
- 16-bit word count which defines the length of the payload data in bytes.
- 8-bit ECC (error-correcting code) to protect the packet header.

The payload data contains application specific data. It is mainly used to convey pixel data, or command parameters. Its length is defined by the word count. It may be between 0 and 65535 bytes in length.

The packet footer consists of a 16-bit checksum (CS). It is calculated by the transmitter and used by

the receiver to check whether the data has been received with no errors.

The minimum length of a long packet is 6 bytes with 0 payload data

- Four PH bytes.
- Two PF bytes.

The maximum length is 65541 bytes

- Four PH bytes
- 65535 payload data
- Two PF bytes.

Short packet

Short packets are formed of four bytes:

- One byte for data ID.
- Two bytes for command or payload data.
- One byte for ECC.

They are mainly used for short command transmission and for timing sensitive information like **video synchronization events**.

Data identifier byte

The first byte of any packet is the DI (data identifier) byte. The DI byte is composed of a virtual channel (VC) identifier and a data type (DT).

1. Virtual channel identifier

A DSI Host may serve up to four peripherals with tagged commands or blocks of data, using the virtual channel ID field of the header. The VC identifies the peripheral to which the data is directed to.

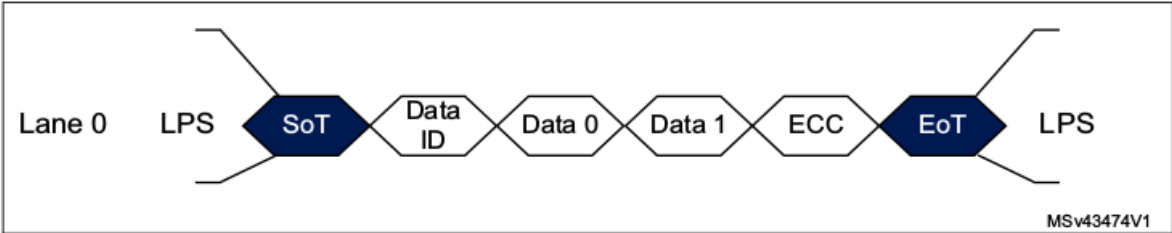
2. Data type field DT[5:0]

The data type field specifies if the packet is a long or short packet type. It also specifies the packet format and content of the payload data.

Packet transmission modes

Short and long packets may be transmitted either in **HS** or **LP mode**. Also packets sent in **HS mode** may be split between available data lanes.

Figure 32. Short packet transmission in HS mode using one data lane



In LP mode, only the data lane 0 is used for transmission.

Figure 34. Short packet transmission in low-power mode

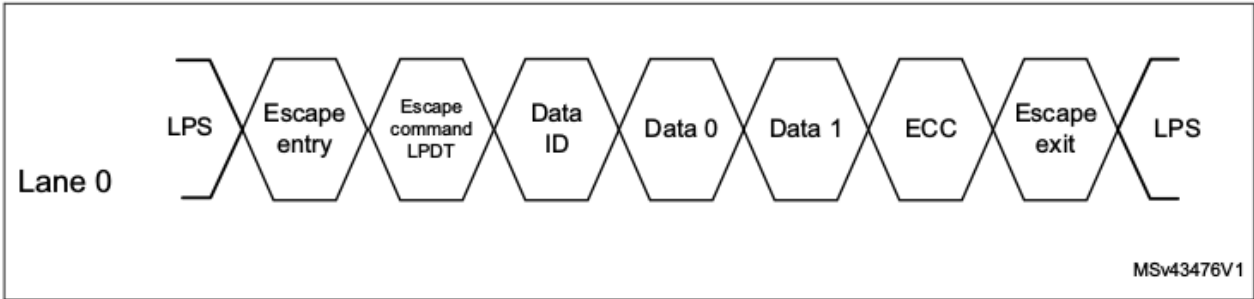
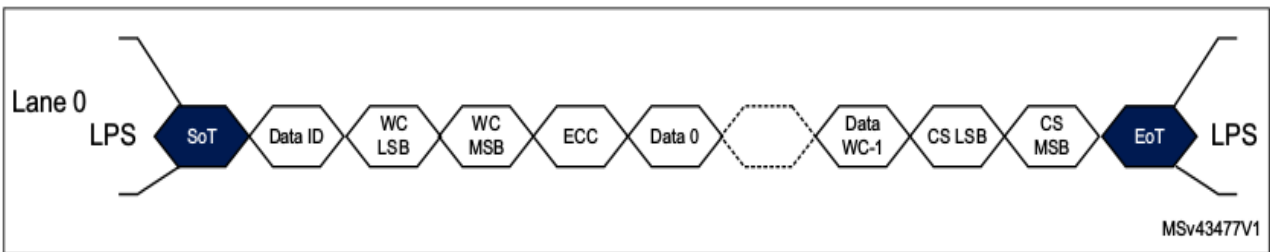
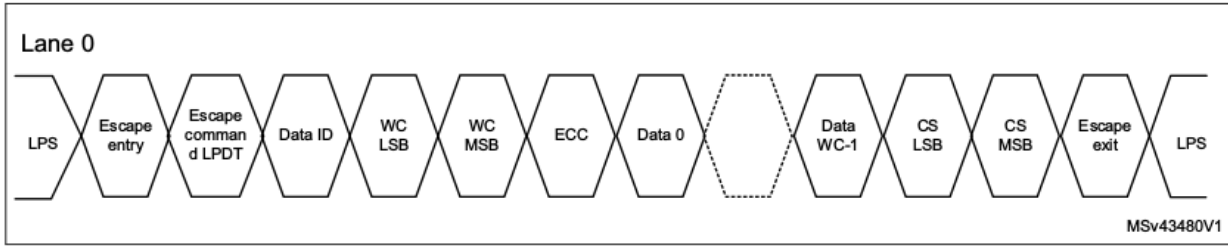


Figure 35. Long packet transmission in HS mode using one data lane



In HS mode, the data transmission can be done using multiple lanes.

Figure 38. Long packet transmission in low-power mode



Host to display data types

The host to display data types can be short packets or long packets. They can be either video or command data types. The host to display data types are shown in.

Table 6. Host to display data types

Data type	Description	Packet	DSI mode
0x01	Sync event, V sync start	Short	Video
0x11	Sync event, V sync end	Short	Video
0x21	Sync event, H sync start	Short	Video
0x31	Sync event, H sync end	Short	Video
0x08	End of transmission packet (EoTp)	Short	Both
0x02	Color mode (CM) OFF command	Short	Video
0x12	Color mode (CM) ON command	Short	Video
0x22	Shut down peripheral command	Short	Video
0x32	Turn ON peripheral command	Short	Video

Table 6. Host to display data types (continued)

Data type	Description	Packet	DSI mode
0x03	Generic short write, no parameters	Short	Command
0x13	Generic short write, 1 parameter	Short	Command
0x23	Generic short write, 2 parameters	Short	Command
0x04	Generic read, no parameters	Short	Command
0x14	Generic read, 1 parameter	Short	Command
0x24	Generic read, 2 parameters	Short	Command
0x05	DCS short write, no parameters	Short	Command
0x15	DCS short write, 1 parameter	Short	Command
0x06	DCS read, no parameters	Short	Command
0x37	Set maximum return packet size	Short	Command
0x09	Null packet, no data	Long	Video
0x19	Blanking packet, no data	Long	Video

0x19	Blanking packet, no data	Long	Video
0x29	Generic long write	Long	Command
0x39	DCS long write/Write_LUT	Long	Command
0x0C	Loosely packed pixel stream 20-bit YCbCr, 4:2:2 Format	Long	Video
0x1C	Packed pixel stream 24-bit YCbCr, 4:2:2 Format	Long	Video
0x2C	Packet pixel stream 16-bit YCbCr, 4:2:2 Format	Long	Video
0x0D	Packet pixel stream 30-bit RGB, 10-10-10 Format	Long	Video
0x1D	Packet pixel stream 36-bit RGB, 12-12-12 Format	Long	Video
0x3D	Packet pixel stream 12-bit YCbCr, 4:2:0 Format	Long	Video
0x0E	Packet pixel stream 16-bit RGB, 5-6-5 Format	Long	Video
0x1E	Packet pixel stream 18-bit RGB, 6-6-6 Format	Long	Video
0x2E	Loosely packed pixel stream 18-bit RGB, 6-6-6 Format	Long	Video
0x3E	Packed pixel stream 24-bit RGB, 8-8-8 Format	Long	Video

Shut down and color modes

The shutdown and the turn on commands are short packets used to switch on or off the display module

The color mode commands are short packets that allow to switch the display module between normal mode and low-color mode. The low-color mode is used for power saving purposes

Synchronization events

The synchronization events are sent through short packets since short packets are more suitable to convey accurate timing information

Packed pixel streams

The packed pixel stream (PPS) packets are long packets used to transmit RGB image data formatted as pixels to a video mode display module. The packet consists of the DI byte, a two-byte WC, an ECC byte, a payload of length WC bytes and a two-byte checksum.

Command mode data types

The command mode data types may be sent in HS or LP. They are used to write to and read from the display registers and from the frame buffer.

Generic commands

There are three types of generic commands:

- Generic short write.
- Generic long write.
- Generic read

Figure 46. Generic short write commands

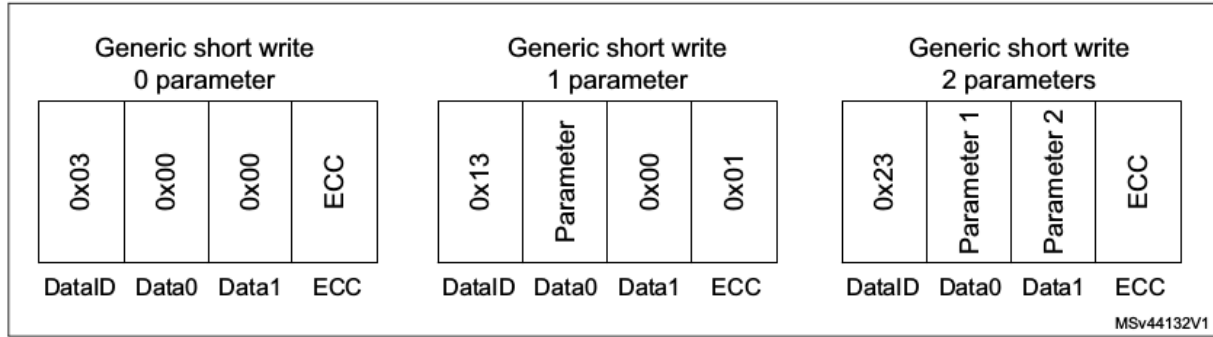


Figure 47. Generic long write commands

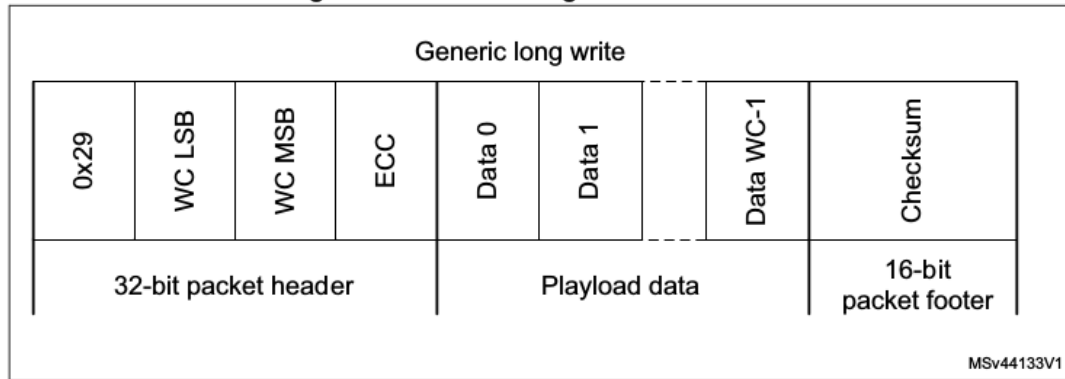
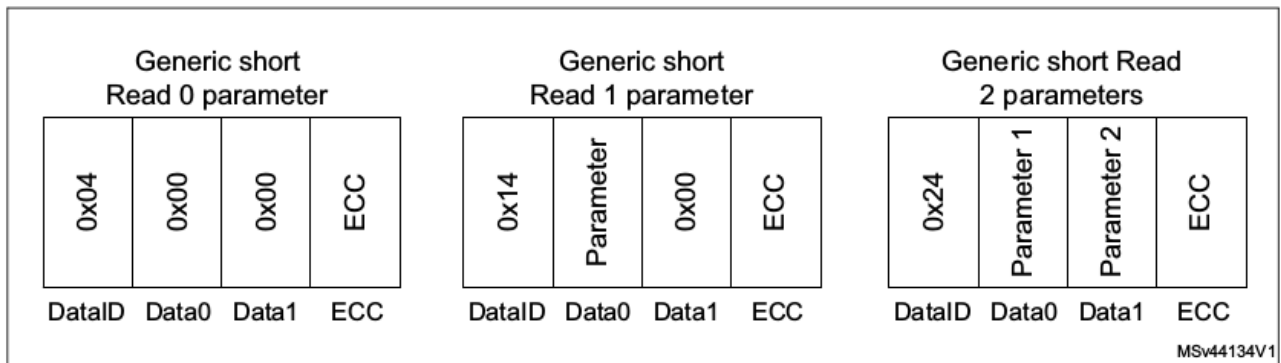
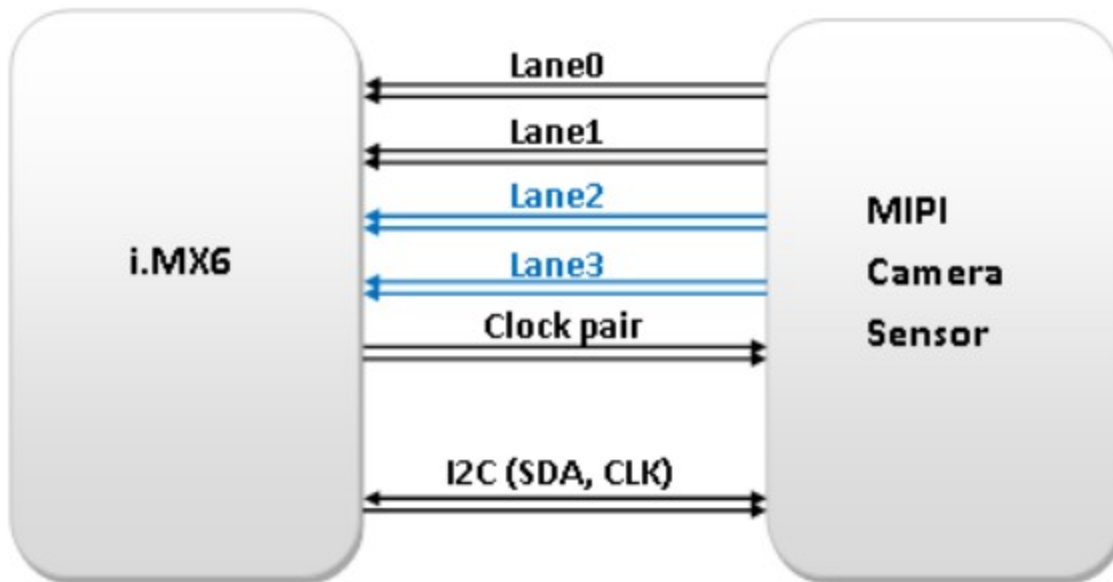


Figure 48. Generic read commands



11. MIPI CSI-2 Host Controller

Overview



■ NOTE: i.MX6DL and 6S do not have lanes 2 and 3

MIPI CSI-2 Host Controller Is used to receive a Data from a CSI-2 Complaint Camera Sensor. The CSI-2(Camera Serial Interface) Protocol specification is a part of Communication protocols defined by MIPI (Mobile Industry Processor Interface Standard) Alliance Standards intended for mobile system chip-to-chip Communications. CSI-2 Specification is used for the Image application Processor Communication in Cameras.

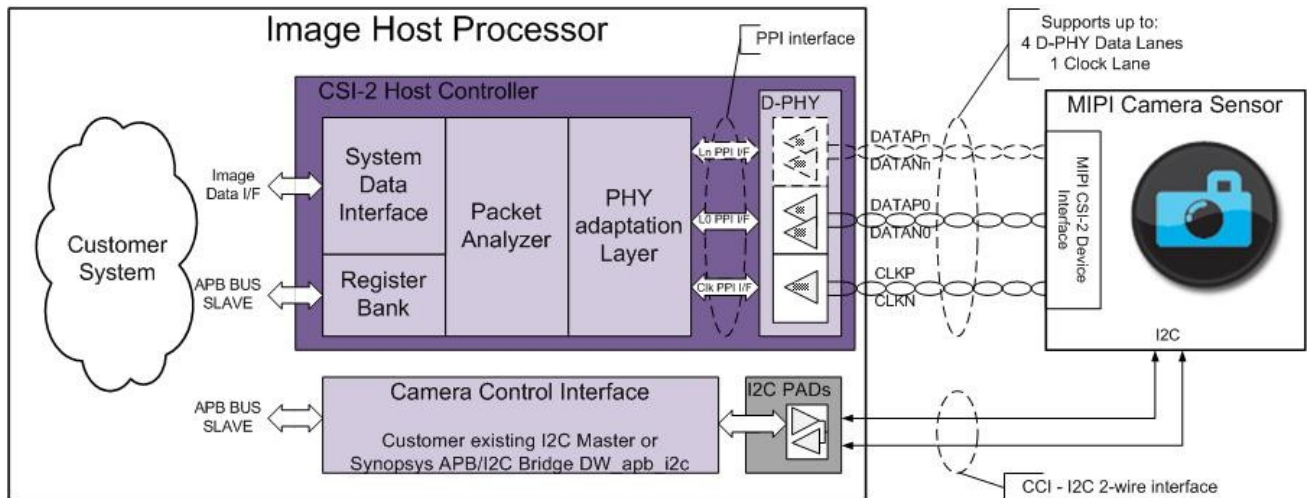
The MIPI CSI-2 Host Controller implements all the Protocol Functions .The Image Data Interface Provides Image data formatted for Memory storage and timing accurate video synchronization signals. The D-PHY acts as Physical layer Configured in one Clock lane and in between 1 to 4 Data lanes.

For Camera Configuration, In the Camera Sensor Side Camera Control Interface (CCI) is used. CCI is a two-wire, bi-directional, half_duplex, serial interface for controlling the Transmitter.

Features of MIPI CSI-2 Host Controller

- Dynamically configurable Multi-Lane Merging
- Supports Long and Short Packet Decoding.
- Supports upto 4 Data Lanes.
- Timing Accurate Signaling of Frame and Line Synchronization Packets.
- Supports Several Frame Formats Like General Frame & Digital Interlaced Video.
- Supports Virtual Channel Interleaving.
- Supports All Primary and Secondary Data Formats (RGB, YUV and RAW Formats, 24-bits to 6 bits per pixel and User-Defined Data Types).
- Error Detection and Correction at PHY, Packet, Line and Frame Levels.
- Support Camera Control Interface (CCI) Through I2C Interface.
- Interface with MIPI D-PHY following PHY Protocol Interface (PPI).

Block Diagram:



MIPI CSI-2 Host Controller consists of **PHY Adaptation Layer**, **Packet Analyzer**, **Image Data Interface** blocks and **D-PHY** block as specified in MIPI D-PHY Specification.

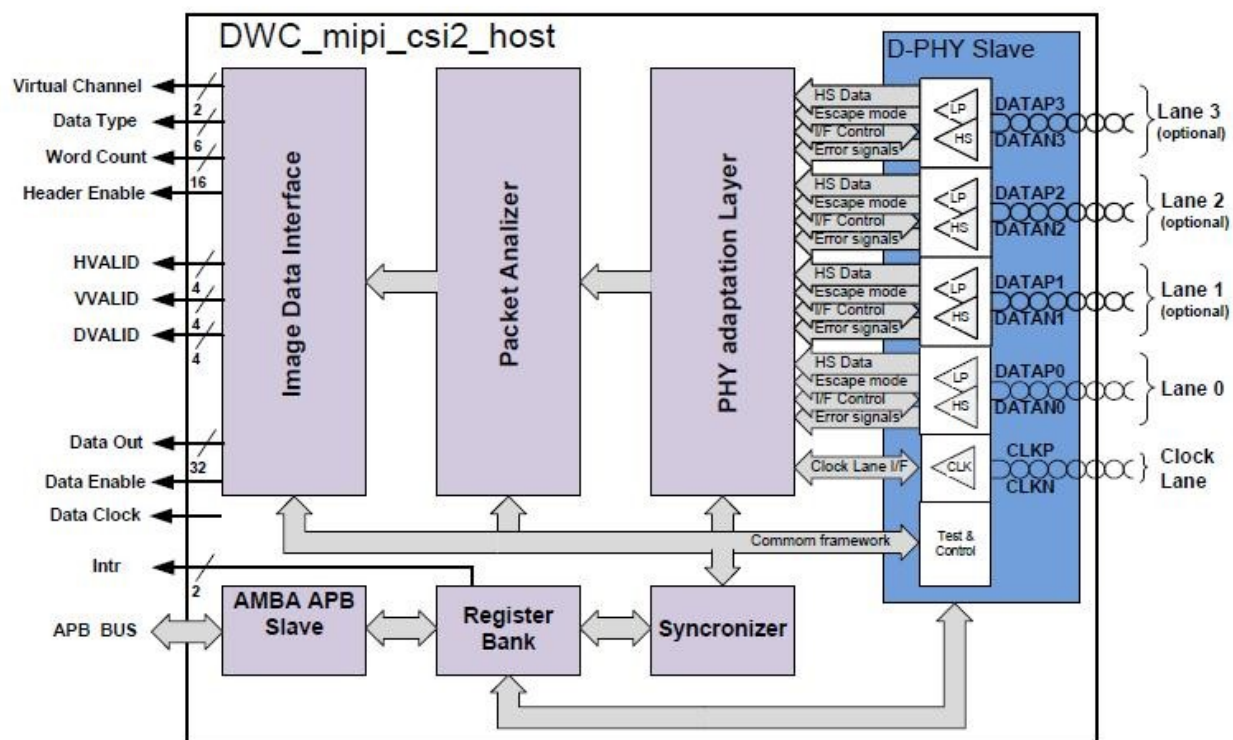
PHY Adaptation Layer is responsible for managing the **D-PHY** Interface, including **D-PHY Error Handling**. **Packet Analyzer** is used for Data lane merging from different Data Lanes together with

Header Decoding, Error Detection and Correction, Frame Size Error Detection and CRC Error Detection. Image Data Interface block Separates CSI-2 Packet Header Information and Reorders the data according to Memory Storage Format and generates the Timing Accurate Video Synchronization Signals.

AMBA - APB Slave interface providing Access to MIPI CSI-2 Host Registers for Configuration & Control. D-PHY Macro implements the Physical link Layer and is responsible for maintaining different Lanes.

CCI interface is used for Camera Configuration and Control. CCI is used as fast mode Variant of the I2C Interface and Supports 7-bit addressing.

*****A bridge has to be build to CSI Controller which is compliant to AMBA interface. Schedule will be impacted.***



D - PHY

D-PHY serves as physical layer for both MIPI DSI and CSI-2 interfaces.

D-PHY Transceiver unit is responsible for transmission and reception of data in High speed (HS) or Low Power (LP) mode. High speed mode is used for high-speed data transmission while the low power mode used for the control purpose. Point-to-point lane interconnect can be used for either data or clock signal transmission. High speed receiver is a differential line receiver while low- Power receiver is an un-terminated, single-ended receiver circuit.

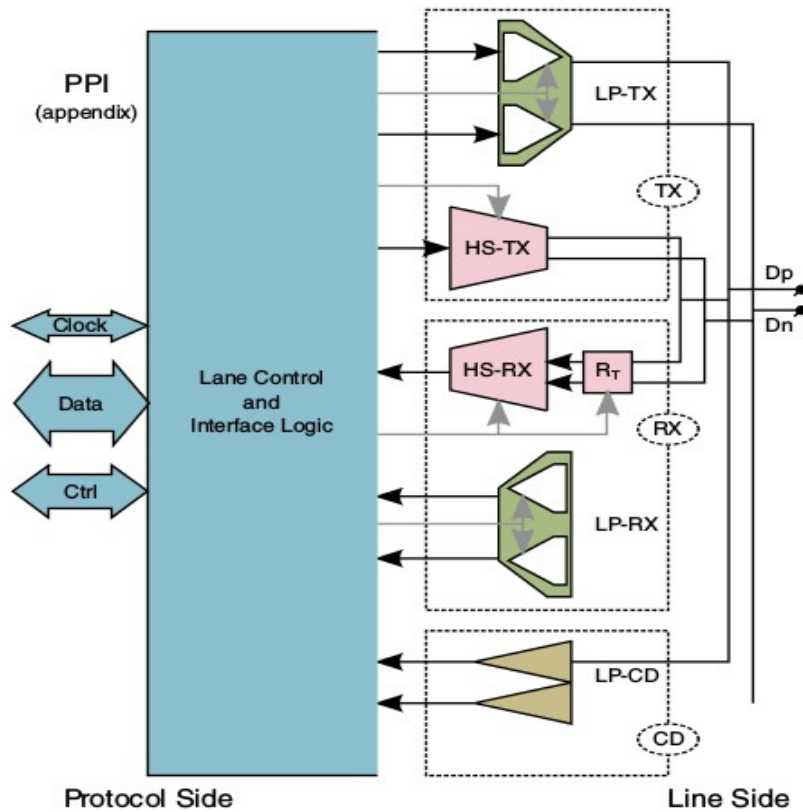


Figure 9-11. Single lane D-PHY- Block Diagram.

MIPI-CSI2 and D-PHY configuration example sequence

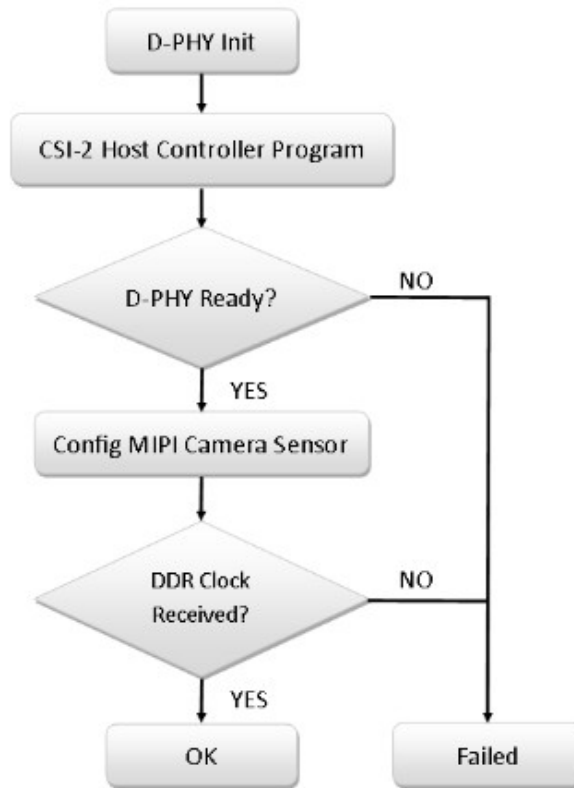


Figure 12. MIPI-CSI2 and D-PHY configuration sequence

- **Program the CSI2 host controller registers according to the operating mode's requirements**
 - Number of lanes (register N_LANES)
 - Deassert PHY shutdown (register PHY_SHUTDOWNZ)
 - Deassert PHY reset (register PHY_RSTZ)
 - Deassert CSI reset (register CSI2_RESETN)
 - (Optional) program data IDs for matching of error reporting (DATA_IDS_1 and DATA_IDS_2 registers)
 - (Optional) program the interrupt masks (MASK1 and MASK2 registers)
- Read the PHY status register (PHY_STATE) to confirm that all **data and clock lanes of the D-PHY are in the stop state**, which

means that they are ready to receive data.

- Access the camera sensor using the CCI interface to initialize and configure the camera sensor to transmit a clock on the D-PHY clock lane.
- Read the PHY status register (PHY_STATE) to confirm that the D-PHY is receiving a clock on the D-PHY clock lane.

Note:

1.if required, configure the MIPI camera sensor to have all Tx lanes in the LP-11 state (STOPSTATE).

2.The D-PHY specification states that the D-PHY master must be initialized in the LP-11 state (STOPSTATE). However, a CCI command may be required to switch the MIPI interface on

MIPI D-PHY clock

The camera sensor (the sensor output differential clock) drives and controls the MIPI D-PHY clock. The MIPI D-PHY clock must be calibrated to the actual clock range of the camera sensor's D-PHY clock and the calibrated value must be equal to or greater than the camera sensor clock. This frequency **ranges from 80 MHz to 1000 MHz**.

The MIPI D-PHY clock must be set according to a known value of the camera sensor's pixel clock. This must be a known value or a value measured with an oscilloscope during a high-speed burst.

To calculate the MIPI data rate, use these equations:

$$\text{MIPI data rate} = (\text{MIPI clock} * 2) * \text{Number of lanes} \geq \text{Pixel clock} * \text{Bits-per-pixel}$$

$$\text{MIPI clock} = (\text{Pixel clock} * \text{Bits-per-pixel}) / (\text{Number of lanes}) / 2$$

For example, a video input of 720p, 59.94 fps, and YUV422 is calculated as follows:

$$\text{Pixel clock} = 1280 * 720 * 59.94 \text{ fps} * 1 \text{ cycle/pixel} * 1.35 \text{ blank interval} = 74.57 \text{ MHz}$$

Total MIPI data rate is 74.25 M * 16 bits = 1193 Mb/s.

The frame blank intervals and the interface packaging overhead were added as the 1.35 factor in the pixel clock equation above.

For a 2-lane interface:

MIPI clock = $1193 / 2 / 2 = 298.25$ MHz

MIPI_CSI2_PHY_TST_CTRL1 setting = $298.25 \text{ MHz} * 2$ (DDR mode) = 596.5 MHz

For a 4-lane interface:

MIPI clock = $1193 / 4 / 2 = 149.12$ MHz

MIPI_CSI2_PHY_TST_CTRL1 setting = $149.12 \text{ MHz} * 2$ (DDR mode) = 298.24 MHz According to Table 2, **MIPI_CSI2_PHY_TST_CTRL1 = 0x28.**

This table shows the frequency range and the exact register value when ref_clock is set to 27 MHz:

Parameters	Values
Width	640
Height	480
FPS	30
BITS/PIXEL	24
Blank interval	1.35
Pixel clock	298598400
Total MIPI data rate	298598400
For a 2-lane interface	74649600

D-PHY registers

FW Functions	Register	Bits: Explanation	Default Value
csi_enable	PHY shutdown control (MIPI_CSI_PHY_SHUTDOWNZ)	Shutdown input. This line is used to place the complete module into power down. All analog blocks are in the power down mode and the digital logic is cleared. Active-low. Default value: 0	Default value: 0
csi_enable	PHY reset control (MIPI_CSI_DPHY_RS)	D-PHY reset output. Active-low.	Default value: 0

	TZ)	Default value: 0	
csi_enable	CSI2 controller reset (MIPI_CSI_CSI2_RES ETN)	CSI-2 controller reset output. Active-low. Default value: 0	Default value: 0
mipi_csi_set_on_lanes	Number of active data lanes (MIPI_CSI_N_LANES)	Number of active data lanes Can be updated only when the PHY lane is in the stop state Default value: CSI_N_LANES 00—one data lane (lane 0) 01—two data lanes (lanes 0 and 1) 10—three data lanes (lanes 0, 1, and 2) 11—four data lanes (all lanes)	00—one data lane (lane 0)
mipi_csi_read(CSI2_VERSION)	Controller version identification register (MIPI_CSI_VERSION)	Version of the CSI-2 host controller Default value: CSI_VERSION_ID	
mipi_csi_write(CSI2_PHY_TEST_CTRL0, 1);	D-PHY test interface control 0 (MIPI_CSI_PHY_TST_CTRL0)	The PHY test interface strobe signal which is used to clock the TESTDIN bus into the D-PHY. It controls the operation selection in conjunction with the TESTEN signal. Default value: 0 PHY test interface clear. When active, it performs the vendor-specific interface initialization (active-high). Default value: 0	
mipi_csi_write(CSI2_PHY_TEST_CTRL1, 0);	D-PHY test interface control 1 (MIPI_CSI_PHY_TST_CTRL1)	PHY test interface operation selector: 1—configures the address write operation on the falling edge of	

		<p>TESTCLK 0—configures the data write operation on the rising edge of TESTCLK</p> <p>PHY output 8-bit data bus for read-back and internal probing functionalities Default value: 0</p> <p>PHY test interface input 8-bit data bus for internal register programming and test functionalities access Default value: 0</p>	
	General settings for all blocks (MIPI_CSI_PHY_STAT E)		
	Error state register 1 (MIPI_CSI_ERR1) Table 35. Erro		
	Error state register 2 (MIPI_CSI_ERR2)		

/*

* This table is based on the table documented at

* <https://community.nxp.com/docs/DOC-94312>. It assumes

* **a 27MHz D-PHY pll reference clock.**

*/

static const struct {

u32 max_mbps;

u32 hsfreqrange_sel;

} hsfreq_map[] = {

{ 90, 0x00}, {100, 0x20}, {110, 0x40}, {125, 0x02},

```

        {140, 0x22}, {150, 0x42}, {160, 0x04}, {180, 0x24},
        {200, 0x44}, {210, 0x06}, {240, 0x26}, {250, 0x46},
        {270, 0x08}, {300, 0x28}, {330, 0x48}, {360, 0x2a},
        {400, 0x4a}, {450, 0x0c}, {500, 0x2c}, {550, 0x0e},
        {600, 0x2e}, {650, 0x10}, {700, 0x30}, {750, 0x12},
        {800, 0x32}, {850, 0x14}, {900, 0x34}, {950, 0x54},
        {1000, 0x74},
};

static int max_mbps_to_hsfreqrange_sel(u32 max_mbps)
{
    int i;

    for (i = 0; i < ARRAY_SIZE(hsfreq_map); i++)
        if (hsfreq_map[i].max_mbps > max_mbps)
            return hsfreq_map[i].hsfreqrange_sel;

    return -1;
}

static void dw_mipi_csi2_phy_write(u32 test_code, u32 test_data)
{
    /* Latest code provided by Synopsys */
    writel(0x00000001, CSI2_PHY_TEST_CTRL0);
    // TestCLR = '1'

    writel(0x00000000, CSI2_PHY_TEST_CTRL0);
    // TestCLR = '0'

    writel(0x00000044, CSI2_PHY_TEST_CTRL1);
    // TestDIN = '44'

    writel(0x00010000, CSI2_PHY_TEST_CTRL1);
    // TestEN = '1'

    writel(0x00000002, CSI2_PHY_TEST_CTRL0);

```

```
// TestCLK = '1'

writel(0x00010000, CSI2_PHY_TEST_CTRL1);

// TestEN = '1'

writel(0x00000000, CSI2_PHY_TEST_CTRL0);

// TestCLK = '0'

writel(0x00010000, CSI2_PHY_TEST_CTRL1);

// TestEN = '1'

writel(0x00000000, CSI2_PHY_TEST_CTRL1);

// TestEN = '0'

writel(test_data, CSI2_PHY_TEST_CTRL1);

// TestDIN = '08'

writel(0x00000002, CSI2_PHY_TEST_CTRL0);

// TestCLK = '1'

writel(0x00000000, CSI2_PHY_TEST_CTRL0);

// TestCLK = '0'
```

```
}
```