LOONGSON 2F

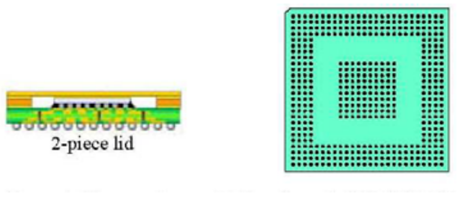
High performance 64bit superscalar MIPS microprocessor

PRELIMINARY DATA

FEATURES

批注 [S1]: MAIN Features, Review and add/remove if needed.

- 64-bit MIPS III Instruction Set and Extended Set compatible, 64-bit word length;
- Quad-issue dynamic superscalar with support for 2 fix point units, 2 fully pipelined floating point multipliers / adders and a load / store unit.
- 9-10 stage super pipelining with support for register renaming, dynamic scheduling, branch prediction and other out-of-order execution.
- IEEE 754-compatible floating-point units enable fully pipelined multiplication and addition arithmetic operation, and hardware division and extraction, with support for media acceleration.
- The joint TLB has 64 entries each of which contains 2 pages ranging from 4KB-4MB; the independent 16-32 entry instruction TLBs support executable bits and prevents buffer overflow attack.
- On-chip separate our-way set associative L1 caches offer 64 KB capacity and 62-byte block for code and data application respectively.
- 512KB of on-chip four-way set associative L2 cache (32-byte block) can be turned on/off by software.
- Integrated 64-bit DDR3 memory controller I/F at 333MHz max.
- Integrated 133MHz PCIX bus controller, with support for PCIX bus, and compatible with PCI.



Package: 452 HFC BGA   27mmx27mm

- 1-GHz main frequency allows for dynamic frequency conversion and shutdown of the core clock for dynamic power management.
- Lower power < 5W at 1GHz.
- Video accelerate module in its write data path to PCI/PCI-X controller. With software driver, the video accelerate module can transfer YUV format video data to RGB format and do zoom action automatically.

OVERVIEW

批注 [S2]: Overview summary

The Loongson2F is an evolution of the Loongson2E with the enhanced I/O and memory accessing bandwidth and a software work frequency changing scheme and compatibility to MIPS64.

The Loongson2F integrates a high performance Loongson2 CPU core, DDR2 memory controller, PCI/PCI-X interface, Local bus, interrupt controller and video acceleration unit.

The Loongson2F will be manufactured in CMOS090 90nm technology using standard rules.

1. Product Overview

批注 [S3]: Detail overview of the G2F

The Loongson2F is an evolution of the Loongson2E with the enhanced I/O and memory accessing bandwidth and a software work frequency changing scheme and compatibility to MIPS64.

The Loongson2F has a standard 32-bit PCI/PCI-X interface, a standard 64-bit DDR2 interface, an 8/16-bit Local IO interface, a 4-bit GPIO interface and an enhanced processor core developed from Loongson2E.

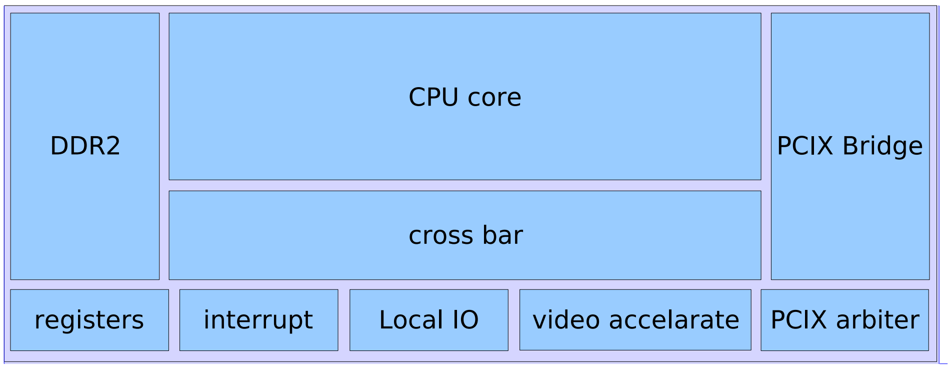
The Loongson2F is intended to be build into a system via standard PCI/PCI-X bus. It can both act as a master or a target PCI/PCI-X as device/host.

The Loongson2F is also intended to achieve higher memory accessing bandwidth by utilizing a 64-bit DDR2 memory controller.

The Loongson2F is expected to have better power management ability by using a software manageable work frequency changing scheme. The operating system can utilize this feature to change the processor frequency according to the workload.

The Loongson2F integrates a video accelerate module in its write data path to PCI/PCI-X controller. Accompany with software driver, the video accelerate module can transfer YUV format video data to RGB format and do zoom action automatically. This can greatly reduce the processor's work load when the system utilizes a simple VGA controller.

The cores are centered on 2x2 AXI cross bar with 128-bit width data bus. The CPU core and PCI/PCI-X slave takes up two master ports, DDR2 controller one slave port, and all other modules including the PCI/PCI-X master share one slave port.



批注 [S4]: This block diagram is not details enough.

1.1 CPU Core

Loongson2F CPU core is a four-issue nine-stage general-purpose RISC microprocessor that implements the 64-bit MIPS instruction set.

The features of this CPU core include:

- Four issue
- Nine stage
- Out of order executing

- Two fix-point units
- Two float-point units
- 64-entry reorder buffer
- 24-entry load-store queue
- 16-entry fix-point issue queue
- 16-entry float-point issue queue
- 64-entry data TLB
- 8-entry instruction TLB
- 64KB instruction L1 cache, 4way set associative
- 64KB data L1 cache, 4way set associative
- 512KB L2 cache, 4way set associative
- 2K-entry BHT
- 32-entry BTB

1.2 AXI Crossbar

The AXI Crossbar provides a 2x2 interconnection compatible with the AMBA AXI protocol. The features of the crossbar include:

- 128bit, full-pipelined data-path.
- Up-to-4 asynchronous FIFOs to transfer signals between different clock domains.
- Up-to-4 programmable address windows for individual AXI Master.

1.3 DDR2 SDRAM Controller

The built-in memory controller of Loongson2F processor fully conforms to DDR SDRAM industry standard JESD79C. All memory read/writes are implemented according to JESD79C specification.

The features of this memory controller include:

- Fully pipelined command read and writes data interfaces to the memory controller.
- Advanced bank look-ahead features for high memory throughput.
- Interface to a standard AXI port.
- A programmable register interface to control memory device parameters and protocols including auto pre-charge.
- Full initialization of memory on memory controller reset.
- Built-in adjustable Delay Compensation Circuitry (DCC) for reliable data sends and captures timing.
- ECC functionality with single-bit and double-bit error reporting and automatic correction of single-bit

error events. Programmable reporting and correction. Programmable removal of ECC storage.

- Programmable memory data path size of full memory data width or half memory data width.
- Clock frequencies form 113MHZ to 333MHZ supported.

1.4 Video Accelerator

The Video Accelerator supports the standard yuv444 and yuv422 video format. With the cooperation of software, the data of YUV format can be translate to RGB format. Besides, Video Accelerator can perform auto-zoom with little software intervention.

1.5 PCI/PCI-X Controller

The PCI/PCI-X controller conforms to both PCI 2.3 and PCI-X 1.0 specification. Its features include:

- Pin selectable host or satellite mode
- 32-bit bus width
- Support fast back to back as a target
- Support dual address cycle
- 8 outstanding master request
- 4 delay-split read request
- Support PCI-X 133

1.6 PCI/PCI-X Arbiter

The PCI/PCI-X arbiter follows both PCI and PCI-X specification. Including following features:

- Two-level round-robin arbitration theme
- Bus parking
- Bad device detect and isolation

1.7 Local Bus

The local bus provides a simple bus interface for system boot ROM and IO device. The interface is designed for chip-connect simplicity.

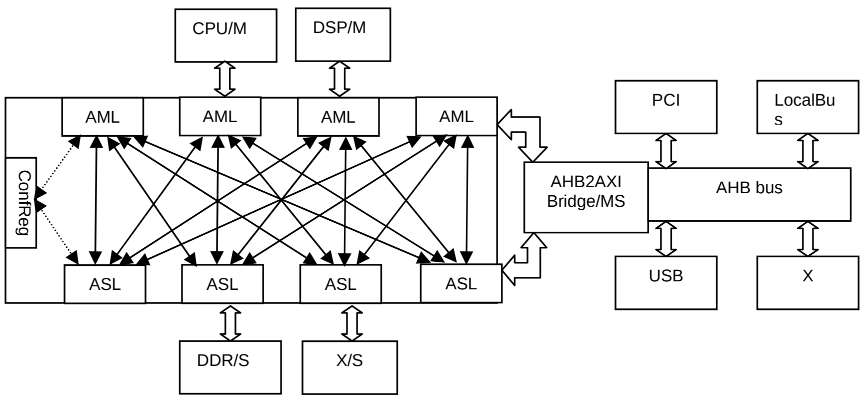
1.8 Interrupt Controller

12 pins are available for chip interrupt. Including 4 dedicated interrupt pin, 4 PCI interrupts and 4 GPIO capable of interrupt triggering. Each pin can be individually configured as level/edge sensitive and masked out.

2. SoC Architecture

批注 [S5]: From your doc More details about the architecture. I think it is good to keep the comparison to L2E. But we need to have a identical L2E document .

The expandable Loongson SoC architecture consists of two stages: the crossbar switch which interfaced with high-performance IP blocks such as Loongson 2x, DSP, DDR2, PCI Express; and the share bus which is connected with low-speed IP blocks such as Loongson 1x, PCI and Local Bus. The Loongson SoC adopts the AXI interface specification for its crossbar switch ports and AHB standard for its share bus interfaces. The AHB bus is connected with the AHB2AXI Bridge, as a standard AXI interface, as shown in the figure below.



This architecture offers scalability and configurability. Fist of all, the number of ports is configurable for both the AXI crossbar switch and the AHB bus. The AXI stage supports up to 4 master AXH devices which can be a Loongson 2x, a DSP (codec IP), a AHB master port and a high-speed I/O such as PCE Express or Hyper Transport; and up to 4 slave AXI devices which can be a AHB slave port, a DDR2 port, a high-speed I/O port such as PCI Express or Hyper Transport and other high-speed ports. Second, individual ports are configurable in synchronous or asynchronous transmission mode, thus eliminating the problem on signal transmission due to the clock domain crossing (CDC), to meet the requirements of other IP blocks. Further more, parameters including FIFO items and arbitration logic are dynamically configurable according to the specific features of other IP blocks.

Its core architecture includes three sections: the crossbar switch in the AXI interface (AXI switch module), the bridge between AXI and AHB (AXI2AHB) and the AHB bus and its arbiter (AHB module). Only AXI switch module is available in the L2F.

2.1 AXI Switch Module

The AXI switch module is connected to IP blocks via AXI interfaces, with built in arbitration and routing logic. Seeing the diversity trend of IP blocks, special attention is paid to the design of the arbitration and addressing logic in terms of customization of hardware specification and software programmability. One AXI master may access multiple slaves. For example, a CPU core needs to access DDR, PCI and other interfaces. In turn, a slave may be accessed by multiple masters. For example, a DDR interface can be accessed by a CPU core, PCI master and other IP blocks. Typically, Master IP blocks have well-established specifications for interfaces. However, some slave IP blocks have some certainties in interface specifications, like the configurable number of AXI interfaces, built-in arbitration logic, etc.

The AXI Switch Module consists of AML (AXI-Master-Link) and ASL (AXI-Slave-Link). The module also includes a register module used for software-management/-configurability of AML and ASL sumodules. The AML is connected with the Master using AXI Slave specifications, which provides buffer and addressing. It requires a corresponding software-configurable register to configure address windows that match accessible AXI Slaves in one-to-one and multiple-to-one mode. The AML will decode and select based on the address windows. The address space is determined by the base address, size and local base address. With 4 segments of address pace at most, one AML needs 12 registers. The ASL is connected with the Slave through the AXI Master specifications, providing arbitration and Slave service requests. The register module receives write and read requests on configuration of the registers of the AXI Switch Module, such configuration of address windows in the AML module. The AXI specifications define that every AXI interface has five channels: a Read Address, a Read Data, a Write Address, a Write Date and a Write Response. All the channels have the same 128-bit data width, 64-bit address width, and 8-bit ID width. The ID includes global ID which is determined by structure and interconnected logic, and local ID which is internally decided by the master IP. While allowing for multiple masters and requests, and out-of-order completion, the ASL uses ID to determine the address route along which data will return. The ID itself contains logic and topology information that is transparent to slave IPs. How to assign and manage ID:

1. Master IP starts a new transaction, when the AXI interface ID contains only local ID information. The AML sends the local ID domain to the ASL.
2. After receiving the transaction, the ASL uses the logic and topology to determine the global ID and put it together with the local ID. Then the resulting ID is transmitted to the

slave IP.

3. The slave IP returns the data with an ID to ASL. In turn, the ASL sends the ID to the AML that corresponds to the global ID domain.
4. The AML masks and returns the global ID to the Master IP。

In the above process, the master adopts the address mode. The assignment of global ID is transparent to the master.

The relationship between the AML and ASL is asker and server. The interface signaling still uses the AXI interface specifications. The AML buffers a request from the AXI master until next data beat comes when it determines the destination ASL port based on the access address. The ASL responds to a read/write request from a AML through the arbitration. The communication between AML and ASL is consistent with the interface in terms of data, address, and ID bit-width. Because the AML and ASL are interconnected with a crossbar switch, the signals from all the AML ports are delivered to four ASL ports simultaneously. For convenience of identification, the signals between the four AXL ports and the four master devices are prefixed with m0_, m1_, m2_, m3_; these between the four slave devices and AXI switch, with s0_, s1_, s2_, s3_; these from the AML to ASL, with aml0_, aml1_, aml2_, aml2_; these from the four ASL to AML, with asl0_, asl1_, asl2_, asl3_.The Loongson SoC architecture AXI bus signals are named in small letters.

2.1.1 AML Design

The AML buffers a write\read request from the AXI master and then decides the corresponding ASL module based on the configurable address windows. Each AML contains 4 window addresses each of which are software-configured to correspond to a certain AXI slave. The address decoding logic transmits the request to the corresponding ASL. The window address is configurable as low as 1MB, for a minimum number of address bits to compare. The Loongson SoC architecture has a 128-bit AXI data width (max), resulting in 128-bit of write data buffer.

To decouple the AXI master and AXI switch pipelines, the AML provides a two-item buffer request queues for read and write requests respectively. A request from the AXI crossbar switch pipeline first joins the request queue, and then one of the items is transmitted to the corresponding ASL after windows comparison.

2.1.2 ASL Design

Similarly, the interface between the ASL and AML implements AXI protocols, with separate channels and arbitration logic for read and write requests. After the arbitration process, the ASL buffers and transmits a request from various AML modules to the AXI slave.

The response from the AXI slave is buffered at the ASL for beat before going to the AML for arbitration. The corresponding AML port is identified immediately since the response signal contains the requesting AXI master ID. Similarly, the there are two items in the buffer when the ASL receives a response from the AXI slave.

2.1.3 Asynchronous Interface

For an asynchronous interface, the clock signal converts between the AML and ASL. Multiple asynchronous request sources are synchronized to the slave clock domain for arbitration. So the ASL and AXI slave share the same clock source. The request from different clock domains are processed at the AML module which share the same clock source with the master. Before interoperating with the ASL, the AML signal has to be synchronized with the ASL by an asynchronous FIFO. Repeat the same process when data returns. With the overhead of asynchronous signal conversion, a higher FIFO depth might be needed to minimize the loss in performance. The FIFO is configurable in bit width and depth at RTL level.

2.1.4 Access Sequence

The AXI switch has no constraints on read sequence and enables out-of-order execution. However, because the write address and data are traveling along different paths, the some of the buffer channels could be locked up without limitations. Therefore some limitations are imposed on the master write operation. According to the AXI protocol, the slave supports the master out-of-order write by limiting the depth of out-of-order write or the number of out-of-order write operations. The master out-of-order write is forbidden when there is no available slave information. In other words, the master is required to transfer the write address in the same sequence as the write data, of course, without need for simultaneity. Now our masters are designed for the same sequence. Actually, both the write addresses and the write data are put in position beforehand, even at the same time, thus eliminating the need for out-of-order transfer. However, in our crossbar switch, the slave port receives and transmits write from 4 masters and transfers to the slave IP blocks. For the slave, this is a type of out-of-order write that takes place among multiple masters. Some support is needed to avoid the lockup effect. For example, when Master 1’s write address arrives earlier than its write data, it seizes one of the buffer entries in the slave port. If only one entry is available, the slave port will not receive other write address but wait for the corresponding write data. Just then Master 2's write data arrives and enters the slave port. Now the crossbar switch cannot transmit to the slave IP block. It has no more available buffer entry to receive Master 1's write data it is awaiting, either. The lockup takes place.

Solving this problem, the Loongson SoC builds a write operation list at the slave port of the AXI crossbar switch, which is used to control the reception and transmission of write data, preventing the buffer in the slave side from seizing by the write data that the slave IP block is not ready to receive. This is done by recording all the write address requests that the slave IP block have received to this write operation list and allowing the write data matching these recorded requests to enter the slave section of the crossbar switch, ensuring all these write data are received by the slave IP block without taking up the buffer in the slave section of the switch. Once the last write data request arrives, these write operations are deleted from the write operation list, thus overcoming the lockup due to out-of-order writes among different masters.

2.1.5 Bridge Module for Data Width Conversion

The crossbar switch reaches 128 bits in both the AXI interface and internal data width in consideration of: a) today, all the memory modules sold in the market are 64-bit DDR/DDR2, so 128-bit data is returned per beat. The adoption of 128-bit data width makes it possible to prevent the delay occurred when the CPU accesses the memory because of the data width conversion; b) So far the L2x provides 64 bit of access channels for (L1 and registers as well as L1 and L2), which is scalable to 128 bits at most. For this reason, 128-bit bandwidth is sufficient; c) a 128-bit crossbar switch offers moderate performance and high cost-effectiveness, compared with lower 64-bit and expensive 256-bit bandwidth.

An IP block with data width higher or lower than 128 bits has to connect to the AXI crossbar switch module through a bridge module where its data width is converted to standard 128 bits required by the AXI switch. The bridge module, which provides data width conversion and protocol translation, is available in master-link interfacing to AXI master and slave link interfacing to AXI slave. The data width conversion scenarios include:

- A request higher than 128 bits from the AXI Master can be transferred in the Burst mode including multiple Transfers as defined in the AXI specifications.
- A request lower than 128 bits from the AXI Master can be assigned a transfer size using the SIZE Zone of the AXI protocol.
- The return data higher than 128 bits from the AXI Slave can be transferred in the Burst

mode including multiple Transfers as defined in the AXI specifications.

- When return data from the AXI Slave is lower than 128 bits, the allocation of the data on the 128-bit data bus is determined by the transfer address.

When the AXI Master (L2x) has to convert its data path width from 128 bits at the AXI Switch to 32 bits at the AXI Slave (PCI), LEN and SIZE conversions are needed at the AXI Slave. Typically, in order to link modules with different data path widths, the bridge module needs a specific queue for data width conversion.

2.1.6 Configuration Register Module

This module allows for configuration of address window registers. These windows each include three 64-bit registers: BASE which is aligned to MB (megabyte), MASK which adopts a format similar to the netmask with a high level of 1, and MMAP where the two low levels represent the number of the appropriate ASL. The assignment of these configuration registers is done using 64-bit two-word write operation.

Window Hit Formula：(IN_ADDR & MASK) == BASE

Target Port Number：MMAP [1:0] in hit window

New Address Conversion：OUT_ADDR = (IN_ADDR & ~MASK) | {MMAP [63:20], 20’h0}

Address	Name	Description
3ff0 0000	M0_WIN0_BASE	base address of Window 0 of Mater 0
3ff0 0008	M0_WIN1_BASE	base address of Window 1 of Mater 0
3ff0 0010	M0_WIN2_BASE	base address of Window 2 of Mater 0
3ff0 0018	M0_WIN3_BASE	base address of Window 3 of Mater 0
3ff0 0020	M0_WIN0_SIZE	Mask of Window 0 of Mater 0
3ff0 0028	M0_WIN1_SIZE	Mask of Window 1 of Mater 0
3ff0 0030	M0_WIN2_SIZE	Mask of Window 2 of Mater 0
3ff0 0038	M0_WIN3_SIZE	Mask of Window 3 of Mater 0
3ff0 0040	M0_WIN0_MMAP	Mapped base address of Window 0 of Master0
3ff0 0048	M0_WIN1_MMAP	Mapped base address of Window 1 of Master0
3ff0 0050	M0_WIN2_MMAP	Mapped base address of Window 2 of Master0
3ff0 0058	M0_WIN3_MMAP	Mapped base address of Window 3 of Master0
3ff0 0060	M1_WIN0_BASE	base address of Window 0 of Mater 1
3ff0 0068	M1_WIN1_BASE	base address of Window 1 of Mater 1
3ff0 0070	M1_WIN2_BASE	base address of Window 2 of Mater 1
3ff0 0078	M1_WIN3_BASE	base address of Window 3 of Mater 1
3ff0 0080	M1_WIN0_SIZE	Mask of Window 0 of Mater 1

3ff0 0088	M1_WIN1_SIZE	Mask of Window 1 of Mater 0
3ff0 0090	M1_WIN2_SIZE	Mask of Window 2 of Mater 1
3ff0 0098	M1_WIN3_SIZE	Mask of Window 3 of Mater 1
3ff0 00a0	M1_WIN0_MMAP	Mapped base address of Window 0 of Master1
3ff0 00a8	M1_WIN1_MMAP	Mapped base address of Window 1 of Master1
3ff0 00b0	M1_WIN2_MMAP	Mapped base address of Window 2 of Master1
3ff0 00b8	M1_WIN3_MMAP	Mapped base address of Window 3 of Master1
3ff0 00c0	M2_WIN0_BASE	base address of Window 0 of Mater 2
3ff0 00c8	M2_WIN1_BASE	base address of Window 1 of Mater 2
3ff0 00d0	M2_WIN2_BASE	base address of Window 2 of Mater 2
3ff0 00d8	M2_WIN3_BASE	base address of Window 3 of Mater 2
3ff0 00e0	M2_WIN0_MASK	Mask of Window 0 of Mater 2
3ff0 00e8	M2_WIN1_MASK	Mask of Window 1 of Mater 2
3ff0 00f0	M2_WIN2_MASK	Mask of Window 2 of Mater 2
3ff0 00f8	M2_WIN3_MASK	Mask of Window 3 of Mater 2
3ff0 0100	M2_WIN0_MMAP	Mapped base address of Window 0 of Master2
3ff0 0108	M2_WIN1_MMAP	Mapped base address of Window 1 of Master2
3ff0 0110	M2_WIN2_MMAP	Mapped base address of Window 2 of Master2
3ff0 0118	M2_WIN3_MMAP	Mapped base address of Window 3 of Master2
3ff0 0120	M3_WIN0_BASE	base address of Window 0 of Mater 3
3ff0 0128	M3_WIN1_BASE	base address of Window 1 of Mater 3
3ff0 0130	M3_WIN2_BASE	base address of Window 2 of Mater 3
3ff0 0138	M3_WIN3_BASE	base address of Window 3 of Mater 3
3ff0 0140	M3_WIN0_MASK	Mask of Window 0 of Mater 3
3ff0 0148	M3_WIN1_MASK	Mask of Window 1 of Mater 3
3ff0 0150	M3_WIN2_MASK	Mask of Window 2 of Mater 3
3ff0 0158	M3_WIN3_MASK	Mask of Window 3 of Mater 3
3ff0 0160	M3_WIN0_MMAP	Mapped base address of Window 0 of Master 3
3ff0 0168	M3_WIN1_MMAP	Mapped base address of Window 1 of Master 3
3ff0 0170	M3_WIN2_MMAP	Mapped base address of Window 2 of Master 3
3ff0 0178	M3_WIN3_MMAP	Mapped base address of Window 3 of Master 3

In addition, when none of the four address windows is hit because of reading an invalid address caused by CPU’s speculation execution, to prevent the system from crashing, the configuration register module return all-zero data to the CPU.

2.2. AXI2AHB Bridge

Not available on L2F.

2.3. AHB Bus Module

Not available on L2F.

3. Processor Core Improvement

批注 [S6]: Samecomments. From your doc. Improvement vs L2E.
Details comments on the 4 main improvments

The L2F's processor core features great architecture improvement, including:

a) Optimized power consumption by minimizing unwanted bit flipping of register stacks, L1 and L2 Caches and other flip-flops and by using dynamic frequency conversion logic to reduce the power consumption when the CPU is free;

b) MIPS compatible design. Compatible with MIPS III, the L2F has to move some of custom operations to SPECIAL 2 and COP2's custom operation code space;

c) AXI compatible processor interfaces.

d) Performance improvement through allowing cache block replacement command to be executable in user mode, as well as partial delay optimization.

3.1 Optimized Power

The L2F delivers lower power by minimizing unwanted bit flipping of register stacks, L1and L2 Caches, and using dynamic frequency conversion logic designs.

The reduced bit flipping is achieved by refusing to read on the register stacks when the instruction operand is 0 or immediate. In the L2E, two operands are read from the register stacks for each instruction. They are src1 and scr2. Our study shows about 70% of fixpoint instruction src2 uses immediate or 0 for operand and this is true of 10-30% of src1. It is necessary to read on the register stacks at the operand of immediate or zero. In this way, the L2F determines whether Read Enable of the register stack should be activated by judging the operand immediate or zero.

The same way is applied in L1 Cache to reduce the undesired read operations to the cache. In the L2E, if the load/store bus（dmemref）is valid, the CPU can read from 16 blocks of RAM of 4 cache banks simultaneously. Each RAM is 512×64 in size. In contrast, L2F allows for more refined access to reduce unnecessary read to reduce power consumption, including:

- Typical store is done in dmemref through simple access to the tag section, without need of access to the data section.
- Typical load is done in dmemref through simple access to the appropriate 64 bits instead of the whole 256 bits of each bank, or by access to only four blocks of cache RAM per cycle.

- CACHE25 is done in dmemref by accessing only four blocks of cache RAM per cycle.
- CACHE1 and CACHE21 are done in dmemref through reading all the 16 blocks of cache RAM per cycle.
- REPLACE and LOOKU are done in drefill through access to all the 16 blocks of cache RAM per cycle.

This supplicated control of RAM Read Enable provides around 80% power reduction for L1 cache RAM.

For reducing the power consumption of L2 cache, L2f includes an additional software-controlled L2 cache Read Enable/Disable, which is implemented through adapting scache and cache2mem, the two modules of the processor core: a) scache: when L2 cache is available, the inclusion relation between L2 and L1 caches ensures the cache block written back from L1 is hit in L2 cache. When L2 cache is disabled, a replacement from L1 cache and a writeback incurred due to L1 cache instruction have to be copied out through the arbiter module. When L2 cache is disabled, if the bit of dmemwrite_w is dirty, then after one beat, the data and address are transferred from the dmemwrite bus to the smemwrite bus which, the write request bus of the cache RAM, provides replacement/writeback for the L2, and provides the same function for the L1 when the L2 is disabled. b) cache2mem: when L2 is disabled, the L1 internal request is put into SCMISS mode immediately after going into the missqueue. Without the need to issue a find (LOOKUP) or writeback and invalid （WTBKINV）request to the L1, diwtbkcntis set at 11. Similarly, without the need to refill the L2, srefillcnt is set at 11.

For software compatibility, when the L2 is disabled, the Loognson 2F support the L2 cache instructions, which, however is not executable. After entering the missqueue, the L2 cache instructions are set in the RDY, with the srefillcnt set at 11. In the RDY state, the chip sends a message along the refill bus to the CP0 queue that appropriate instructions has been executed. If the L2 cache is disabled, it is necessary to determine whether the replacement in the data cache can be written back to the arbiter module (without write buffer) or the buffer (with the buffer). The signal wait_cache_yes is generated depending on whether the writeback incurred by the L1 instruction can be copied to the arbiter module (without write buffer) or the buffer (with the buffer). This signal will be zero only if the queue is empty and write is allowed without any ongoing writeback.

Furthermore, a thorough study shows the availability of flip-flops that can use pipeline enable signals for input enable control in the Loongson 2F. So thousands of such flip-flops are optimized for pipeline enable signal application.

The Loongson 2F is still designed with dynamic frequency conversion logic, making it possible to reduce power consumption by scaling the main frequency when the OS is not in use. Controlled by software, the processor core runs at 0/8, 2/8, 3/8, 4/8, 5/8, 6/8, 7/8 and 8/8 of operating frequency. It is achieved by maintaining div_count, a 8-bit shift counter at the PLL output clock which can dynamically adjust the number of 1 within the device based on a 3-bit control signal freq_scale. When freq_scale=011, for example, the processor core clock runs at 4/8 of the PLL clock frequency, and there are four shifting“1s” in the 8-bit div_count. The core clock is the result of an individual bit of div_count NAND the PLL clock phase.

3.2 MIPS Compatible Design

MIPS only defined the user mode instructions in its early MIPS Architectures such as MIPS I, MIPS II, MIPS III, MIPS IV and MIPS V. Later the company provided both user mode and kernel mode in their new specifications such as MIPS32 and MIPS64. Now MIPS only licenses and supports MIPS32 and MIPS64.

MIPS compatibility is a main goal of the Loongson 2F design. It is designed to compatible with MIPS III. Implementing all the features described in the MIPSIII, Loongson 2E has some of custom instructions occupied the MIPS-reserved instruction slot. So in the 2F, these custom-tailored instruction opcodes need to remove to the user instruction slot (COP2 or Special2). They include instructions as follows:

- The 2E implemented MOVZ and MOVN, the instructions designed for MIPS IV instead of MIPS III. Compatible with MIPS64 specifications, eliminating the need of modifications.
- The media instructions take the COP2 slot reserved for user, compared with the 2E whose COP1 instruction slot is taken up.
- The MIPS architecture implements floating-point multiply-add instructions in MIPS IV and later versions, while the 2E realizes the floating-point multiply-accumulate instructions because the multiply-add is not available in MIPS III, resulting in incompatibility. For the 2F, the floating-point multiply-accumulate opcode need to remove to Special 2, the customer-reserved slot.
- In the 2E, the fmt domain for fixpoint instruction paired-single is used by media instructions, leading to incompatibility with MIPS. Align the 2F.
- Considering the fixpoint multiply-division instructions use HI and LO registers in the previous MIPS instruction set, the 2E implements the instruction using general-purpose registers for target registers. Remove these opcode to Special 2 instruction slot in the 2F.

Finally, these changes in instruction format do not impact the 2F’s internal opcode and data paths. For example, the 2F uses COP2 slot for media instruction, but internally continues to use COP1 opcode slot and data paths.

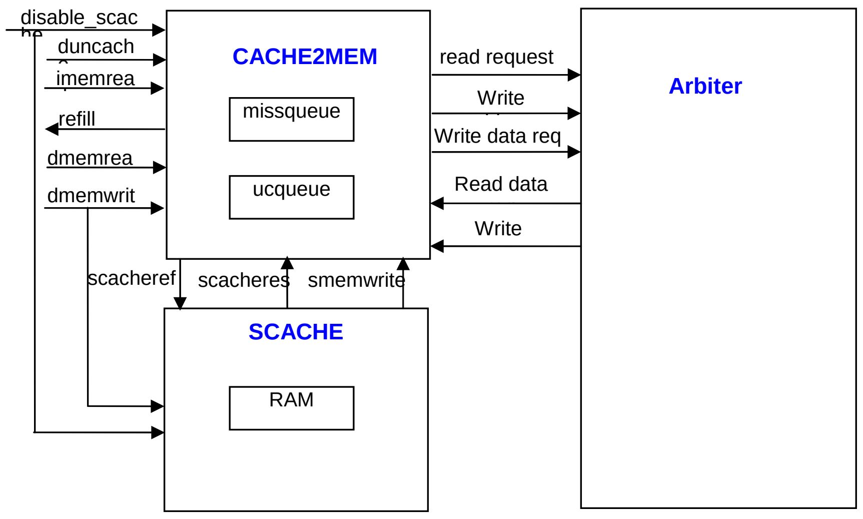
3.3. AXI-Compatible Interface Design

The Loongson 2F implements extendable SoC interconnection architecture and L2 cache with configurable Enable/Disable using the AXI interface specifications. To achieve these features, we have to modify cache2mem, a module of the processor core.

The configurable Enable/Disable is realized adding a disable_scache signal to the L2 cache. If the value of the signal is 1, the L2 cache is disabled; otherwise, the L2 is available. When the initial value of disable_scache equals 0, this feature can be dynamically configured through OS.

In the Loongson SoC architecture, the processor core operates as a master and implements the AXI interface specifications. The AXI interface protocols include five channels: read address, read data, write address, write data and write response. The 2E incorporates the suncache, smemread, smemwrite, srefill0 and srefill1 buses into the five AXI channels.

In the 2F, suncache’s read and smemread are merged into the read address channel; suncache’s write and smemwrite are merged, with write address separate from write data, into write address channel and write data channel; srefill0 and srefill1 combined into read data channel. In the Loongson SoC architecture, all the channels share a uniform data width of 128 bits, the same address width of 64 bits and ID length of 8 bits. The processor core is also modified in terms of domain widths. The diagram below provides the connection between CACHE2MEM, SCACHE and Arbiter.



3.4. Performance Optimization

During the implementation of Loongson 2E's Java virtual machine, it was found that great time was needed in refreshing the cache, and much of the time was consumed during system call (because cache instruction belongs to kernel instruction set, user code has to read the cache through system call). Therefore, a bit called user_mode_cacheop (DIAG [8]) is included in the DIAG control register of the 2F. When the bit is set to 1, the user mode program is allowed to use the cache refresh instructions (CACHE1、CACHE17、CACHE21…).

In addition, some of the delays are optimized for the 2F's physical paths. The delay-optimized section is mainly included in the CPU0 and fixpoint queues, dealing with forward speculation cancel.

4. DDR2 SDRAM Memory Controller Design

批注 [S7]: This chapter will not be present in L2E documents.

4.1. Overview

The 2F integrates a built-in memory controller fully compatible with DDR2 SDRAM standard (JESD79-2B). Its main features as follows:

- Fully pipelined instruction, read/write on the interface
- Fusion and sort of memory instructions for higher overall bandwidth.
- Standard AXI Interface
- Configuration register read/write ports enable changes in basic memory parameters
- Built-in dynamic delay compensation (DCC) circuit for higher data transmission and reception reliability.
- ECC that detects 1-bit error and 2-bit error on the data path and automatically corrects 1-bit error.
- Operation frequency range: 133MHz-333MHz

4.2. Memory Control ASIC Interface and Memory Interface Signal

The memory control ASIC interface signals include AXI-certified interface signal, control signal and DDR2 SDRAM memory interface signal. Two additional pins are included for higher read accuracy. For signal definitions, see the table below:

Name	I/O	Description
Memory Controller ASIC Interface Signals（AXI Section）		
aclk	I	AXI interface clock
areset	I	AXI interface reset
arid[7:0]	I	Read request ID
araddr[39:0]	I	Read request address
arlen[3:0]	I	Read request length
arsize[3:0]	I	Read request size
arburst[1:0]	I	Read request burst type
arvalid	I	Read request valid
arready	O	Read request ready
rid[7:0]	O	Read data ID
rdata[127:0]	O	Read data
rresp[1:0]	O	Read data response
rlast	O	Read last data indication
rvalid	O	Read data valid
rready	I	Read data reception ready
awid[7:0]	I	write request ID
awaddr[39:0]	I	write request address
awlen[3:0]	I	write request length

awsize[3:0]	I	write request size
awburst[1:0]	I	write request burst type
awcache[3:0]	I	write request cache mode
awvalid	I	write request valid
awready	O	write request reception ready
wid[7:0]	I	write data ID
wdata[127:0]	I	write data
wstrb[3:0]	I	write data strobe
wlast	I	Write last data indication
wvalid	I	write data valid
wready	O	write data ready
bid[7:0]	O	Write response ID
bresp[1:0]	O	Write response type
bvalid	O	Write response valid
bready	I	Write response reception ready
Memory Controller ASIC Interface Signals (Control Signal Section)		
controller_int	O	Memory controller interrupt output
dll_lock	O	Memory controller DLL lock indication
ecc_dataout_corrected	O	ECC 1-bit error indication (corrected)
ecc_dataout_uncorrected	O	ECC 2-bit error indication (uncorrected)
q_almost_full	O	Command queue full indication
refresh_in_process	O	Memory module in process of refresh indication
srefresh_enter	I	Memory module enters self-refresh control
scanen	I	Test mode enable
scanin	I	Test input
scanout	O	Test output
scanmode	I	Test mode select
test_rd_clk	I	Test read clock input
test_wr_clk	I	Test write clock input
param_75_ohm_sel	O	Controller Pad termination resistor select
tsel	O	Controller termination resistor enable
config_reg_enable	I	configuration register access control

Memory Controller DDR Interface Signals		
dq[72:0]	IO	DDR2 SDRAM data bus
dqs[8:0]	IO	DDR2 SDRAM data strobe
dqs_n[8:0]	IO	DDR2 SDRAM reverse data strobe
dqm[8:0]	O	DDR2 SDRAM data mask
addr[14:0]	O	DDR2 SDRAM address bus
ba[2:0]	O	DDR2 SDRAM Bank address
we#	O	DDR2 SDRAM write enable
cas#	O	DDR2 SDRAM column select enable
ras#	O	DDR2 SDRAM row select enable
cs[3:0]#	O	DDR2 SDRAM chip select
cke[1:0]	O	DDR2 SDRAM clock enable
clk[5:0]	O	DDR2 SDRAM noninvert clock output

clk_n[5:0]	O	DDR2 SDRAM invert clock output
odt[3:0]	O	DDR2 SDRAM onchip terminating resistor select
gatein	I	Data reception indication
gateout	O	Data reception indication

4.3. Memory Controller Configuration Registers

The memory controller includes 26 64-bit configuration registers. One register contains data that may form multiple parameters, a single parameter or only partial parameter. The table below provides these configuration registers and their parameter information (all the unused bits are reserved).

Parameter Name	**Bit**	**Default**	**Range**	**Description**
CONF_CTL_00[31:0] Offset: 0x00				
AREFRESH	24	0x0	0x0-0x1	Issue an auto refresh command to memory based on the auto_refresh_mode parameter setup
AP	16	0x0	0x0-0x1	Enable memory controller auto-refresh?
ADDR_CMP_EN	8	0x0	0x0-0x1	Allow command requeueing logic to check address conflict?
ACTIVE_AGING	0	0x0	0x0-0x1	Record the ageing commands in the queue avoiding starvation of low priorities?
CONF_CTL_00[63:32] Offset: 0x00				
DDR2_SDRAM_MODE	56	0x0	0x0-0x1	Memory Controller DDRI and DDRII mode setup
CONCURRENTAP	48	0x0	0x0-0x1	Allow controller auto precharge one bank while issuing a command to another bank?
BANK_SPLIT_EN	40	0x0	0x0-0x1	Allow command requeueing logic to split banks?
AUTO_REFRESH_MODE	32	0x0	0x0-0x1	Select when to issue an auto-precharge, next burst or command boundary?
CONF_CTL_01[31:0] Offset: 0x10				
ECC_DISBALE_W_UC_ERR	24	0x0	0x0-0x1	Disable ECC when an unrecoverable error is found in R/M/W?
DQS_N_EN	16	0x0	0x0-0x1	Enable DQA difference?
DLL_BYPASS_MODE	8	0x0	0x0-0x1	Enable DLL BYPASS Mode?
DLLLOCKREG	0	0x0	0x0-0x1	Indicate if DLL locks? (Read only)
CONF_CTL_01[63:32] Offset: 0x10				
FWC	56	0x0	0x0-0x1	Is a mandatory write check required? If yes, memory controller will store in memory XOR of value and data specified in the xor_check_bits.

FAST_WRITE	48	0x0	0x0-0x1	Allow controller to enable fast write? If yes, controller will issue a write command to memory module before receiving complete write data.
ENABLE_QUICK_SREFRESH	40	0x0	0x0-0x1	Enable quick self-refresh? If yes, memory will do self-refresh even if initialization does not end.
EIGHT_BANK_MODE	32	0x0	0x0-0x1	Indicate if memory module has 8 banks.
CONF_CTL_02[31:0] Offset: 0x20				
NO_CMD_INIT	24	0x0	0x0-0x1	Issue other commands over tDLL time during memory initialization?
INTRPTWRITENA	16	0x0	0x0-0x1	Interrupt previous read command with autoprechagre plus other read/write commands to the same bank?
INTRPTREADA	8	0x0	0x0-0x1	Interrupt previous read command with autoprechagre plus other read commands to the same bank?
INTRPTAPBURST	0	0x0	0x0-0x1	Interrupt current autoprecharge with other commands to another bank?
CONF_CTL_02[63:32] Offset: 0x20				
PRIORITY_EN	56	0x0	0x0-0x1	Enable command requeue logic to prioritize?
POWER_DOWN	48	0x0	0x0-0x1	If yes, memory controller will close all the memory pages with precharge command to set clock enable low without transmitting all the received commands until this value is set back to 0.
PLACEMENT_EN	40	0x0	0x0-0x1	Enable command requeue logic?
ODT_ADD_TURN_CLK_EN	32	0x0	0x0-0x1	Insert a turn-around clock between the fast back-to-back read or write commands to different chip selects?
CONF_CTL_03[31:0] Offset: 0x30				
RW_SAME_EN	24	0x0	0x0-0x1	Allow command requeue logic to requeue write/read commands to the same bank?
REG_DIMM_EN	16	0x0	0x0-0x1	Enable registered DIMM memory module?
REDUC	8	0x0	0x0-0x1	Only use 32-bit data path?
PWRUP_SREFRESH_EXIT	0	0x0	0x0-0x1	Use self-refresh to exit power-down mode rather than memory initialization command?
CONF_CTL_03[63:32] Offset: 0x30				
SWAP_PORT_RW_SAME_EN	56	0x0	0x0-0x1	Determine if the similar commands are swapped on the same port when swap_en is enabled?
SWAP_EN	48	0x0	0x0-0x1	Swap a new highly prioritized command for an ongoing one when command

				requeue logic is enabled?
START	40	0x0	0x0-0x1	Initialize memory?
SREFRESH	32	0x0	0x0-0x1	Self-refresh memory?
CONF_CTL_04[31:0] Offset: 0x40				
WRITE_MODEREG	24	0x0	0x0-0x1	Write EMRS register of memory module?
WRITEINTERP	16	0x0	0x0-0x1	Define if a write burst can be interrupted by a read command.
TREF_ENABLE	8	0x0	0x0-0x1	Enable self refresh inside the controller?
TRAS_LOCKOUT	0	0x0	0x0-0x1	Issue auto-prechareg before the expiration of tRAS?
CONF_CTL_04[63:32] Offset: 0x40				
RTT_0	57:24	0x0	0x0-0x3	Define the on-chip termination resistance of memory module.
CTRL_RAW	49:48	0x0	0x0-0x3	Set ECC detect and correction mode.
AXI0_W_PRIORITY	41:40	0x0	0x0-0x3	Set priority for AXI0 port write command.
AXI0_R_PRIORITY	33:32	0x0	0x0-0x3	Set priority for AXI0 port read command.
CONF_CTL_05[31:0] Offset: 0x50				
COLUMN_SIZE	26:24	0x0	0x0-0x7	Set margin between actual and max (14) numbers of column addresses.
CASLAT	18:16	0x0	0x0-0x7	Set CAS latency value.
ADDR_PINS	10:8	0x0	0x0-0x7	Set margin between actual and max (15) numbers of address pins.
RTT_PAD_TERMINATIO N	1:0	0x0	0x0-0x3	Set termination resistance for memory controller pad
CONF_CTL_05[63:32] Offset: 0x50				
Q_FULLNESS	58:56	0x0	0x0-0x7	Define a limit to the number of commands beyond which memory controller queue is considered full.
PORT_DATA_ERROR_TY PE	50:48	0x0	0x0-0x7	Define data error type on the memory controller ports.
OUT_OF_RANGE_TYPE	42:40	0x0	0x0-0x7	Define the type of out-of-range access errors.
MAX_CS_REG	34:32	0x4	0x0-0x4	Define the number of chip-selects used in controller.
CONF_CTL_06[31:0] Offset: 0x60				
TRTP	26:24	0x0	0x0-0x7	Define the number of cycles from memory read command to precharge.
TRRD	18:16	0x0	0x0-0x7	Define the interval of the Active command to different banks
TEMRS	10:8	0x0	0x0-0x7	Define emrs during memory initialization.
TCKE	2:0	0x0	0x0-0x7	Define minimum CKE pulse width.
CONF_CTL_06[63:32] Offset: 0x60				
APREBIT	59:56	0x0	0x0-0xf	Define which bit of address line is

				selected to send autoprecharge to memory.
WRLAT	50:48	0x0	0x0-0x7	Define the time (based on clock cycles) from transmission of write command to reception of first data.
TWTR	42:40	0x0	0x0-0x7	Define the required clock cycles from write to read command.
TWR_INT	34:32	0x0	0x0-0x7	Define write recovery time for memory module.
CONF_CTL_07[31:0] Offset: 0x70				
ECC_C_ID	27:24	0x0	0x0-0xf	Define source ID number for 1-bit ECC error.
CS_MAP	19:16	0x0	0x0-0xf	Define available chip-selects
CASLAT_LIN_GATE	11:8	0x0	0x0-0xf	Define the time (measured in a half of a cycle) for which gate open is active when the read command returns data.
CASLAT_LIN	3:0	0x0	0x0-0xf	Define CAS Latency for memory module.
CONF_CTL_07[63:32] Offset: 0x70				
MAX_ROW_REG	59:56	0xf	0x0-0xf	Define the actual number of row addresses.
MAX_COL_REG	51:48	0xe	0x0-0xe	Define the actual number of column addresses.
INITAREF	43:40	0x0	0x0-0xf	Define the number of autorefresh commands required for system initialization.
ECC_U_ID	35:32	0x0	0x0-0xf	Define the source ID for unrecoverable dual-byte errors. (Translator's Note: 2-bit errors?)
CONF_CTL_08[31:0] Offset: 0x80				
ODT_RD_MAP_CS3	27:24	0x0	0x0-0xf	Make its ODT termination resistance available in defining CS3 read command?
ODT_RD_MAP_CS2	19:16	0x0	0x0-0xf	Make its ODT termination resistance available in defining CS2 read command?
ODT_RD_MAP_CS1	11:8	0x0	0x0-0xf	Make its ODT termination resistance available in defining CS1 read command?
ODT_RD_MAP_CS0	3:0	0x0	0x0-0xf	Make its ODT termination resistance available in defining CS0 read command?
CONF_CTL_08[63:32] Offset: 0x80				
ODT_WR_MAP_CS3	59:56	0x0	0x0-0xf	Make its ODT termination resistance available in defining CS3 write command?
ODT_WR_MAP_CS2	51:48	0x0	0x0-0xf	Make its ODT termination resistance available in defining CS2 write

				command?
ODT_WR_MAP_CS1	43:40	0x0	0x0-0xf	Make its ODT termination resistance available in defining CS1 write command?效
ODT_WR_MAP_CS0	35:32	0x0	0x0-0xf	Make its ODT termination resistance available in defining CS0 write command?
CONF_CTL_09[31:0] Offset: 0x90				
PORT_DATA_ERROR_ID	27:24	0x0	0x0-0xf	ID number of data error on the port.
PORT_CMD_ERROR_TYPE	19:16	0x0	0x0-0xf	Type of command errors on port.
PORT_CMD_ERROR_ID	11:8	0x0	0x0-0xf	ID number of command errors on port.
OUT_OF_RANGE_SOURCE_ID	3:0	0x0	0x0-0xf	ID number of out-of-range access errors.
CONF_CTL_09[63:32] Offset: 0x90				
OCD_ADJUST_PUP_CS0	60:56	0x0	0x0-0x1f	Set OCD pull-up value for memory chip-select 0. Memory controller will issue an OCD adjustment command to memory module according this parameter value during initialization process.
OCD_ADJUST_PDN_CS0	52:48	0x0	0x0-0x1f	Set OCD pull-down value for memory chip-select 0. Memory controller will issue an OCD adjustment command to memory module according this parameter value during initialization process.
TRP	43:40	0x0	0x0-0xf	Define the number of clock cycles required for memory precharge.
TDAL	35:32	0x0	0x0-0xf	If auto-precharge is set, this parameter defines the number of clock cycles for auto-precharge and write recovery.
CONF_CTL_10[31:0] Offset: 0xa0				
AGE_COUNT	29:24	0x0	0x0-0x3f	Define initial aging value for the command requeue logic using ageing algorithm.
TRC	20:16	0x0	0x0-0x1f	Define the number of clock cycles between Active commands to the same bank.
TMRD	12:8	0x0	0x0-0x1f	Define the number of clock cycles required to configure a mode register inside memory modules.
TFAW	4:0	0x0	0x0-0x1f	Define tFAW for memory modules.
CONF_CTL_10[63:32] Offset: 0xa0				
DLL_DQS_DELAY_2	62:56	0x0	0x0-0x7f	Define % of delay for DQS2 when reading data, with an increase of 1/128 of a clock cycle per time.

DLL_DQS_DELAY_1	54:48	0x0	0x0-0x7f	Define % of delay for DQS1 when reading data, with an increase of 1/128 of a clock cycle per time.
DLL_DQS_DELAY_0	46:40	0x0	0x0-0x7f	Define % of delay for DQS0 when reading data, with an increase of 1/128 of a clock cycle per time.
COMMAND_AGE_COUNT	37:32	0x0	0x0-0x3f	Define initial ageing value for each command when command requeue logic using ageing algorithm.
CONF_CTL_11[31:0] Offset: 0xb0				
DLL_DQS_DELAY_6	30:24	0x0	0x0-0x7f	1/128 Define % of delay for DQS6 when reading data, with an increase of 1/128 of a clock cycle per time.
DLL_DQS_DELAY_5	22:16	0x0	0x0-0x7f	Define % of delay for DQS5 when reading data, with an increase of 1/128 of a clock cycle per time.
DLL_DQS_DELAY_4	14:8	0x0	0x0-0x7f	Define % of delay for DQS4 when reading data, with an increase of 1/128 of a clock cycle per time.
DLL_DQS_DELAY_3	6:0	0x0	0x0-0x7f	Define % of delay for DQS3 when reading data, with an increase of 1/128 of a clock cycle per time.
CONF_CTL_11[63:32] Offset: 0xb0				
WR_DQS_SHIFT	62:56	0x0	0x0-0x7f	Define % of delay for clk_wr when writing data, with an increase of 1/128 of a clock cycle per time.
DQS_OUT_SHIFT	54:48	0x0	0x0-0x7f	Define % of delay for DQS when writing data, with an increase of 1/128 of a clock cycle per time.
DLL_DQS_DELAY_8	46:40	0x0	0x0-0x7f	Define % of delay for DQS8 when reading data, with an increase of 1/128 of a clock cycle per time.
DLL_DQS_DELAY_7	38:32	0x0	0x0-0x7f	Define % of delay for DQS7 when reading data, with an increase of 1/128 of a clock cycle per time.
CONF_CTL_12[31:0] Offset: 0xc0				
TRAS_MIN	31:24	0x0	0x0-0xff	Define the minimum number of clock cycles for valid row address commands in memory modules.
OUT_OF_RANGE_LENGTH	23:16	0x0	0x0-0xff	Define command length for out-of-range access.
ECC_U_SYND	15:8	0x0	0x0-0xff	Define the reasons for unrecoverable 2-bit errors (read only).
ECC_C_SYND	7:0	0x0	0x0-0xff	Define the reasons for recoverable 1-bit errors (read only).
CONF_CTL_12[63:32] Offset: 0xc0				
DLL_DQS_DELAY_BYPASS_0	56:48	0x0	0x0-0x1ff	Define the number of dqs0 delay lines in DLL bypass mode.

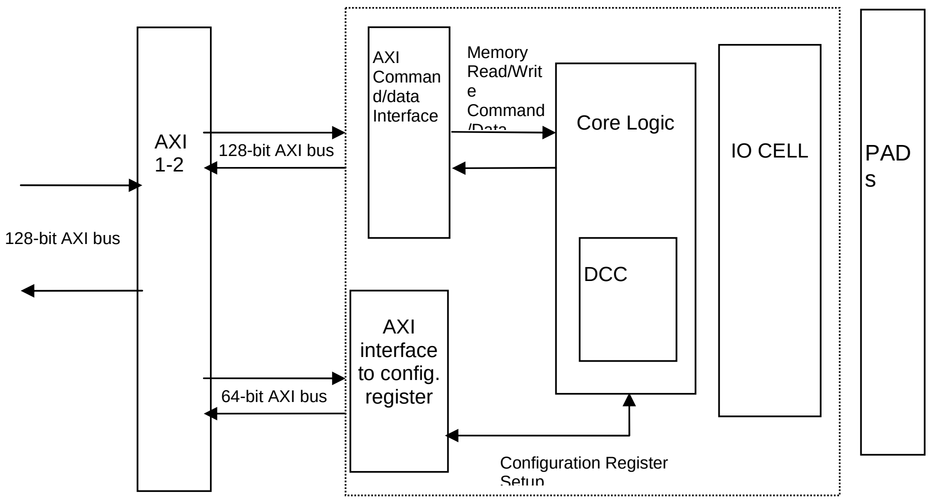
TRFC	47:40	0x0	0x0-0xff	Define the number of clock cycles required for memory module refresh.
TRCD_INT	39:32	0x0	0x0-0xff	Define the number of clock cycles from RAS to CAS.
CONF_CTL_13[31:0] Offset: 0xd0				
DLL_DQS_DELAY_BYPASS_2	24:16	0x0	0x0-0x1	Define the number of dqs2 delay lines in DLL bypass mode.
DLL_DQS_DELAY_BYPASS_1	8:0	0x0	0x0-0x1	Define the number of dqs1 delay lines in DLL bypass mode.
CONF_CTL_13[63:32] Offset: 0xd0				
DLL_DQS_DELAY_BYPASS_4	56:48	0x0	0x0-0x1ff	Define the number of dqs4 delay lines in DLL bypass mode.
DLL_DQS_DELAY_BYPASS_3	40:32	0x0	0x0-0x1ff	Define the number of dqs3 delay lines in DLL bypass mode.
CONF_CTL_14[31:0] Offset: 0xe0				
DLL_DQS_DELAY_BYPASS_6	24:16	0x0	0x0-0x1ff	Define the number of dqs6 delay lines in DLL bypass mode.
DLL_DQS_DELAY_BYPASS_5	8:0	0x0	0x0-0x1ff	Define the number of dqs5 delay lines in DLL bypass mode.
CONF_CTL_14[63:32] Offset: 0xe0				
DLL_DQS_DELAY_BYPASS_8	56:48	0x0	0x0-0x1ff	Define the number of dqs8 delay lines in DLL bypass mode.
DLL_DQS_DELAY_BYPASS_7	40:32	0x0	0x0-0x1ff	Define the number of dqs7 delay lines in DLL bypass mode.
CONF_CTL_15[31:0] Offset: 0xf0				
DLL_LOCK	24:16	0x0	0x0-0x1ff	When indicating DLL lock, delay the number of delay units required over the whole clock cycle.
DLL_INCREMENT	8:0	0x0	0x0-0x1ff	Define the number of additional delay units per time when DLL locks.
CONF_CTL_15[63:32] Offset: 0xf0				
DQS_OUT_SHIFT_BYPASS	56:48	0x0	0x0-0x1ff	Define the number of delay units for wr_dqs in dqs out bypass mode.
DLL_START_POINT	40:32	0x0	0x0-0x1ff	Define the number of initial delay units when DLL locks.
CONF_CTL_16[31:0] Offset: 0x100				
INT_ACK	25:16	0x0	0x0-0x3ff	Set the bit at "1", this parameter will have the bit-related interrupt cleared.
WR_DQS_SHIFT_BYPASS	8:0	0x0	0x0-0x1ff	Define the number of delay units in wr dqs bypass mode.
CONF_CTL_16[63:32] Offset: 0x100				
INT_STATUS	58:48	0x0	0x0-0x7ff	Define the interrupt reasons for memory controller.
INT_MASK	42:32	0x0	0x0-0x7ff	Define interrupt mask bit for memory controllers.
CONF_CTL_17[31:0] Offset: 0x110				
EMRS1_DATA	30:16	0x0	0x0-0x7ff	Define data stored in the memory

				module EMRS1 register when controller initializes memory modules.
TREF	13:0	0x0	0x0-0x3ff	Define clock interval between two memory refresh operations.
CONF_CTL_17[63:32] Offset: 0x110				
EMRS2_DATA_1	62:48	0x0000	0x0-0x7fff	Define EMRS2 data corresponding to Chipslect 1 during memory initialization.
EMRS2_DATA_0	46:32	0x0000	0x0-0x7fff	Define EMRS2 data corresponding to Chipslect 0 during memory initialization.
CONF_CTL_18[31:0] Offset: 0x120				
EMRS2_DATA_3	30:16	0x0000	0x0-0x7fff	Define EMRS2 data corresponding to Chipslect 3 during memory initialization.
EMRS2_DATA_2	14:0	0x0000	0x0-0x7fff	Define EMRS2 data corresponding to Chipslect 2 during memory initialization.
CONF_CTL_18[63:32] Offset: 0x120				
AXI0_EN_LT_WIDTH_INSTR	63:48	0x0000	0x0-0xffff	Define if AXI0 port receives a memory access of less than 64 bits.
EMRS3_DATA	46:32	0x0000	0x0-0x7fff	Define data corresponding to EMRS 3 during memory initialization.
CONF_CTL_19[31:0] Offset: 0x130				
TDLL	31:16	0x0000	0x0-0xffff	Define the number of clock cycles required for memory module DLL lock.
TCPD	15:0	0x0000	0x0-0xffff	Define the number of clock cycles from valid clock to precharge.
CONF_CTL_19[63:32] Offset: 0x130				
TRAS_MAX	63:48	0x0000	0x0-0xffff	Define the max number of clock cycles for valid row commands in memory modules.
TPDEX	47:32	0x0000	0x0-0xffff	Define the number of clock cycles for the command Exit when Power Fails.
CONF_CTL_20[31:0] Offset: 0x140				
TXSR	31:16	0x0000	0x0-0xffff	Define the number of clock cycles for memory selfrefresh exit.
TXSNR	15:0	0x0000	0x0-0xffff	Define tXSNR for memory module.
CONF_CTL_20[63:32] Offset: 0x140				
XOR_CHECK_BITS	63:48	0x0000	0x0-0xffff	When fwc is set up, store in the memory the value of this parameter xor the check bit for next write.
VERSION	47:32	0x2041	0x2041	Define memory controller version #
CONF_CTL_21[31:0] Offset: 0x150				
ECC_C_ADDR[7:0]	31:24	0x000	0x0-	Record address information about 1-bit

		0	0x1ffffffff	ECC errors.
TINIT	23:0	0x000 0	0x0- 0xfffff	Define number of clock cycles required for memory module initialization.
CONF_CTL_21[63:32] Offset: 0x150				
ECC_C_ADDR[36:8]	60:32	0x0	0x0- 0x1ffffffff	Record addresss informaton about 1-bit ECC errors.
CONF_CTL_22[31:0] Offset: 0x160				
ECC_U_ADDR[31:0]	31:0	0x0	0x0- 0x1ffffffff	Record addresss informaton about 2-bit ECC errors.
CONF_CTL_22[63:32] Offset: 0x160				
ECC_U_ADDR[36:32]	36:32	0x0	0x0- 0x1ffffffff	Record addresss informaton about 2-bit ECC errors.
CONF_CTL_23[31:0] Offset: 0x170				
OUT_OF_RANGE_ADDR[31:0]	31:0	0x0	0x0- 0x1ffffffff	Record addresss informaton about out-rang-accesses.
CONF_CTL_23[63:32] Offset: 0x170				
OUT_OF_RANGE_ADDR[36:32]	36:32	0x0	0x0- 0x1ffffffff	Record addresss informaton about out-rang-accesses.
CONF_CTL_24[31:0] Offset: 0x180				
PORT_CMD_ERROR_ADDR[31:0]	31:0	0x0	0x0- 0x1ffffffff	Record addresss informaton command errors on ports.
CONF_CTL_24[63:32] Offset: 0x180				
PORT_CMD_ERROR_ADDR[36:32]	36:32	0x0	0x0- 0x1ffffffff	Record addresss informaton command errors on port.
CONF_CTL_25[31:0] Offset: 0x190				
ECC_C_DATA[31:0]	31:0	0x0	0x0- 0x1ffffffff	Record data informaton about 1-bit ECC errors.
CONF_CTL_25[63:32] Offset: 0x190				
ECC_C_DATA[63:32]	63:32	0x0	0x0- 0x1ffffffff	Record data informaton about 1-bit ECC errors.
CONF_CTL_26[31:0] Offset: 0x1a0				
ECC_U_DATA[31:0]	31:0	0x0	0x0- 0x1ffffffff	Record data informaton about 2-bit ECC errors.
CONF_CTL_26[63:32] Offset: 0x1a0				
ECC_U_DATA[63:32]	63:32	0x0	0x0- 0x1ffffffff	Record data informaton about 2-bit ECC errors.

4.4. Loongson 2F Memory Control Module Design

Functionally DDR2 SDRAM controller includes a one-to-two AXI interface, a command/data AXI interface, a AXI interface to the configuration register, core logic and a I/O Cell.


128-bit AXI bus
AXI 1-2
128-bit AXI bus
64-bit AXI bus
AXI Command/data Interface
Memory Read/Write Command
AXI interface to config. register
Core Logic
DCC
IO CELL
PADs
Configuration Register Setup

4.4.1 AXI One-to-Two Module

The denali ddr2 module includes a command/data AXI interface via which memory read/write information is transferred from and to the CPU core and PCI interface and a AXI interface to the configuration register which is used for configuration through the CPU core of initialization of the memory controller and management of error information and other tasks. For hardware cost reduction, a one-to-two AXI module is placed between the AXI crossbar switch and the AXI interface to the memory controller, shown in above diagram. For the 2F architecture, this module is designed with the following in mind:

（1）**Channel Selection.** Here Channel 0 is specified as the main path via which nearly all the data moves. In order to avoid memory holes in normal operation, a configuration of disable_conf_spaces is introduced. If this signal is active, all the transactions will be transferred to Channel 0. If not, all the transactions from CPU with length of 1(a?len=0) and a physical address at 0xfff_fe00~0xfff_ffff will be sent to Channel 1.

（2）**Write Channel Algorithm.** Since the AXI write address cycle corresponds to a number of data cycles, the transmission direction of a certain transaction must recorded in the module. Here, Channel 1 is designed to only accept access from CPU, so it is enough to record the ID from CPU. Two 16-bit registers（wc_l0_idmap, wc_l1_idmap）record the transmission direction of all IDs: write data will be transmitted in one single direction if (in and only if) they see a record on the idmap in the same direction or that the same ID will almost complete its transfer in this direction on the present write address channel.

IDs are inserted into the two idmaps when the write address channel succeeds in shaking hands, while they are deleted when the write response channel succeeds in shaking hands. Deletion superior to insertion. An ID can not and should not be inserted into the idmap when its write address and the last write data have completed in one single cycle.

The write of the same IDs is not supported in this module. A write request will be kept from transmitting to any downstream channel if the ID of the write address channel is recorded in the idmap. (the cache2mem of the L2F ensures the write of the same IDs won't appear before the completion of one write ID, without the need for a mechanism preventing the same IDs)

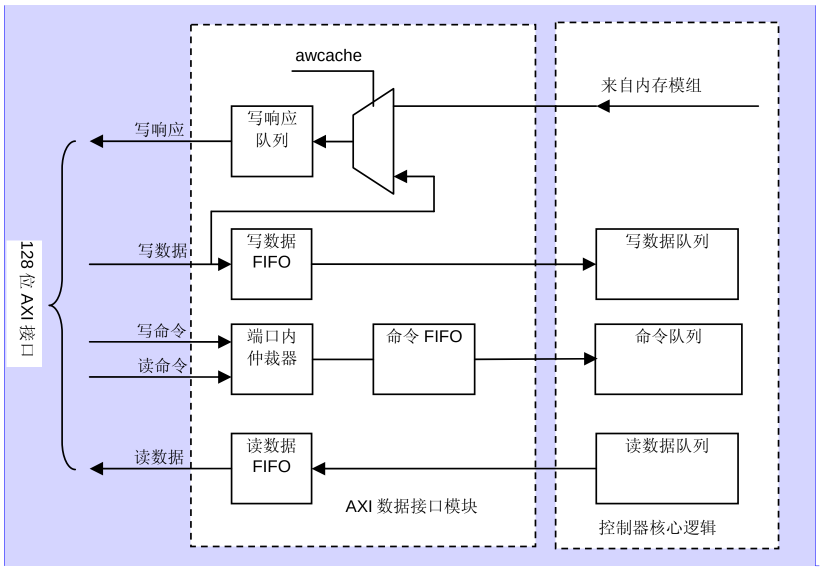
（3）Return Channel. Both write response and read return channels transfer data back to the master. Because the AXI slave cannot start a transmission by itself and a continuous feedback is impossible, a fixed-priority method (Channel 1 is superior to Channel 0) is adopted. Seldom accessed, the use of Channel 1 at the control side of the 1-of-2 data selector can reduce the switch operations, too.

4.4.2. AXI Interface Module

The AXI interface includes data and register interfaces.

The data interface module, which uses the AXI protocol, provides an interface with external memory read and write commands. As an AXI slave, this AXI data port accepts access from the external AXI master CPU and PCIs. It stores in its FIFO the AXI transaction addresses and the size, length and ID of access data, and then translates these requests into internal DDR2 controller commands before sending them to the core control of DDR2 controller that will optimize and transfer these commands to memory modules. Shown in the block diagram, the AXI interface includes 5 separate channels from/to memory: write command, write data, write response, read command, and read data. For optimized performance, the read command to the memory controller can be requeued and the read data from the AXI data port of memory controller can be disordered and interleaved. To keep the return read data on the AXI bus from disordering or interleaving, it is possible to transmit read command on the AXI interface using the same ID. Write data interleaving is not allowed at the AXI data port.

The data interface contains 3 FIFOs for command, write and read data. All the FIFOs share the same depth of 2.



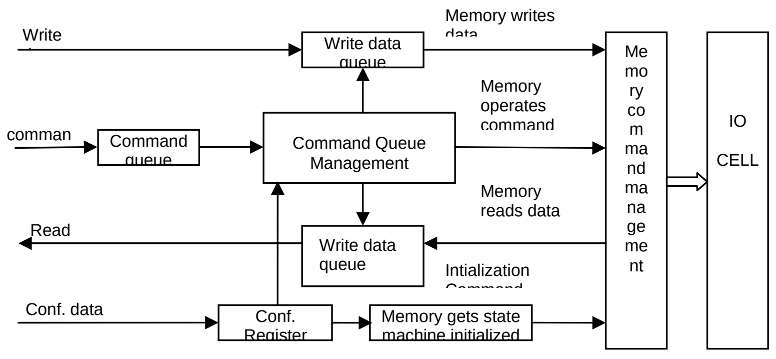
批注 [S8]: To be in ENGLISH

According to the AXI protocols, the read and write are separate channels so that it is possible for read and write commands to arrive at the same time. Therefore a read/write arbiter is placed before the command FIFO. The read/write arbiter algorithm as follows: a) the command that comes in first is stored first in the FIFO. The first-come command means that both awvalid and awready are valid for write and that both arvalid and arready are valid for read. b) If both read and write are active, then read command is received first. The command interface FIFO stores what is described in the AXI protocol as follows: i) address (araddr or awaddr) ii) command type (arvalid or awvalid) iii) read/write length (awlen or arlen) iv) read/write data width (awasize or arsize) v) bufferable/cacheable tag (awcache) vi) command ID (arid or awid). When receiving a write command, the AXI data interface returns the response signal to the command initiator using bresp and the corresponding valid signal bvalid. Masters have different requirements for response time. Some masters want the bus to be released immediately, hoping the memory controller responds quickly after receiving the command. Some masters do not want the controller to respond until the data is really stored in the memory. These response requirements are provided using awcache. When the signal is b0001, the memory controller will transmit a response upon reception of the last data of the write request. When awcache is b0000, indicating this is a non-cacheable request, the memory controller will not return the response until all the data of the write request are transmitted to memory module. When the core logic of memory controller returns read data, the read data FIFO stores information as follows: a) read data (rdata); b) read command ID (rid); c) last data tag (rlast); d) read data response (rresp) which identifies the return status of the AXI read command transaction. The write data FIFO stores the write data from the AXI bus, including the following information: a) data to memory (wdata); b) write command ID (wid); c) data mask (wstrb).

The AXI interface of configuration registers simply translates an access command from the 1-to-2 AXI interface into an internal access to configuration registers. A total of 26 64-bit configuration registers are included, each containing one or more configuration parameters. For more see Section 3.

4.4.3. Memory Controller Core Logic

The Core Control Logic includes Command, Write Data and Read Data Queues, Memory Initialization State Machine, Command Queue Management with Command Requeueing, Memory Module Command Management Queue, Delay Compensation (DCC) circuit, etc. The Core Logic receives and puts the internal translated command from the AXI interface module in the command queue of the Core Logic, and places the write data in the write data queue. At the same time it stores the read data from memory module in the Read Data Queue and then transfers them to the external master through the AXI Interface Module. At the Command Queue Management, the commands in the queue are translated into standard memory read/write through a wide range of optimization methods, before being transferred to Memory Module Command Management. The Memory Initialization State Machine reads initial configuration information in the configuration registers and transfers the initialization command to the Memory Module Command Management following the standard DDR2 SDRAM initialization process. The Memory Module Command Management receives the memory command from the Command Queue Management and Memory Initialization State Machine. It also transfers the resulting memory module command to the IO Cell by scheduling commands according to memory specific commands, such as refresh, precharge and others. At the core of the DDR2 memory controller PHY, the Delay Compensation (DCC) will be presented in a separate section. The block diagram below shows the complete core logic of memory controller:



Each of Write Data and Read Data Queues contains eight 128-bit entries, while the Command Queue includes eight entries. It records the address, data size and ID of a memory access from the port. When the parameter placement_en is set 0 at the configuration registers, the command queue is equivalent to a simple internal queue. Before being sent to the memory controller, the commands will be placed in the queue in time order, and will generate and transmit memory commands to the Memory Command Management in the order in which they are received. When the parameter placement_en is set 1, the Command Queue Management with Command Requeueing will use the following methods to optimize the queuing of the commands in the queue in order to increase the overall bandwidth of memory controller:

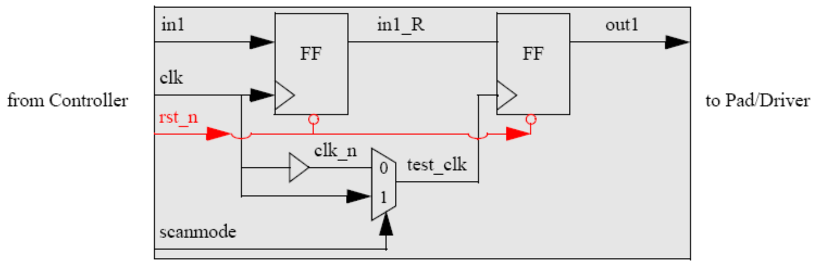
a) For read and write access to the same chip-select, bank and page (the same row address), the memory read/write command will be initiated in the order in which the commands are received. This feature could be disabled through the parameter addr_cmp_en only if the consistency is ensured in the overall system.
b) To access the same ID, the memory read/write command will be initiated in the order in which the commands are received. The only exception is that a read of different addresses can be initiated before a write command even if they share the same ID.
c) The 2 read commands or a read command plus a write command on the same port in the queue can be initiated out of order. Note that the 2 write commands on the same port must be sent in the order.
d) When priority_en is set, the command requeueing logic will get the high-priority commands initiated first before these with lower priority regardless of the order in which it receives commands.
e) When two commands access different row addresses on the same bank, the last access must close the current page using the precharge command before the next command can access a different page, resulting in a lot of overhead. For this reason, the command requeueing logic will combine the commands of access to the same pages without conflicts of addresses and IDs, thus reducing the cycle of close/open and increasing the overall system bandwidth.
f) Some overhead also occurs when memory switches from read commands into write commands. So when the parameter rw_same_en is enabled, the command requeueing logic will try to combine read commands that do not have conflicts of addresses and IDs with other read commands, and it will try to combine write that do not have conflicts of addresses and IDs with other write commands, too.

4.4.4. I/O Cell

The I/O Cell module receives and translates the commands and data from the Memory Command Management into DDR2-compatible timing signals. It also controls data transmission of the L2F's DDR2 Pad. A memory controller requires four types of I/O cells for control command, data mask, transmission and reception of data and data strobe.

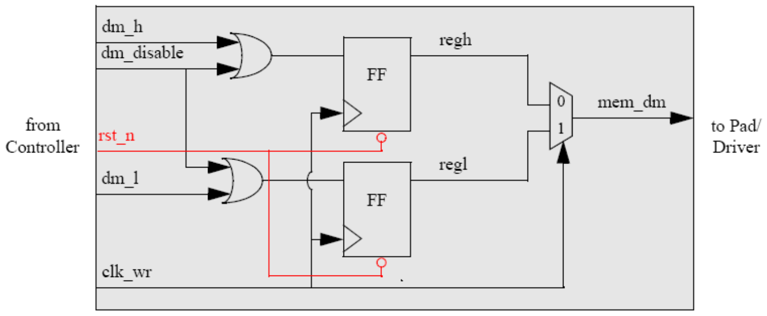
1）I/O Cell for control command output

This type of I/O Cell transmits memory control commands including memory address, bank, chip-selects, CKE, RAS#, CAS#, and WE#. These one-direction output signals eliminate the need for Pad enable signals. And they are transmitted on clock falling edge for maximum sample window for memory grains. This type of I/O cell is shown in the block diagram below.



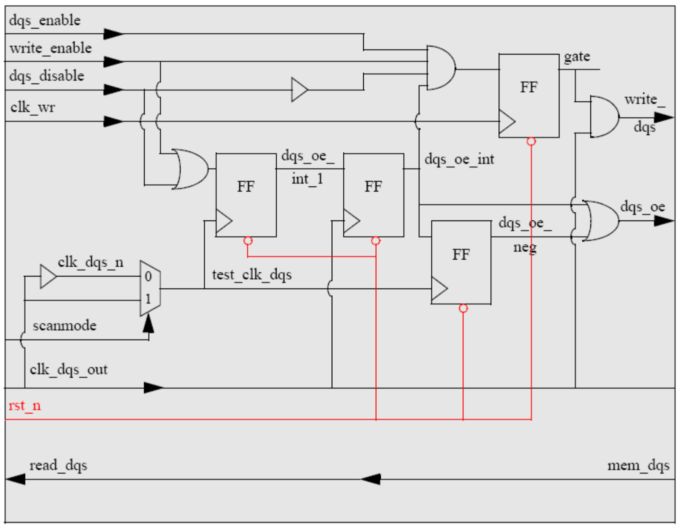
2）I/O Cell for data mask output

The DDR2 needs to transmit write data and mask simultaneously during one single write operation. Each of DDR2 transactions transmits 256-bit data every two clock cycles, with every 8-bit data corresponding to 1-bit mask. That means 16-bit data are transmitted in one clock cycle. Unlike the control signal, the data mask output needs to transmit data twice on one clock edge, one on a write clock (clk_wr) rising edge, the other on its falling edge. The data mask output I/O Cell is shown in the block diagram below.



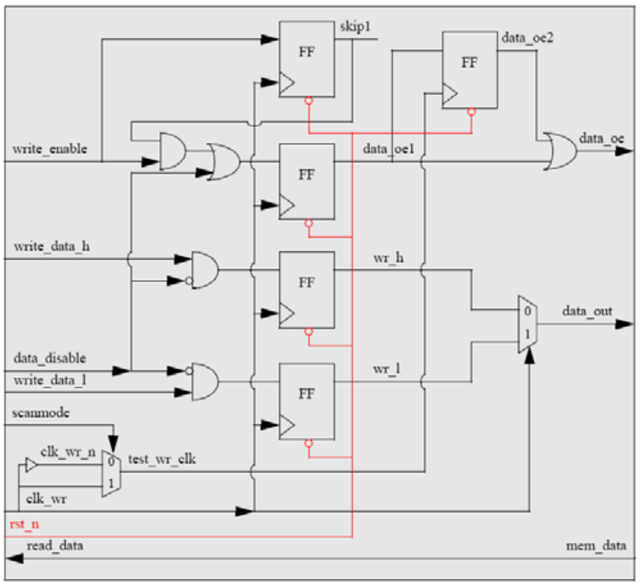
3）I/O Cell for data strobe

A data strobe (DQS) is a bi-directional signal. So it is necessary to generate output enable signals. The input dqs signal is not handled in the I/O cell but is directly transmitted to the Delay Compensation Circuit (DCC) of the Core Control Logic Module. This type of I/C Cell is showed in the block diagram below:



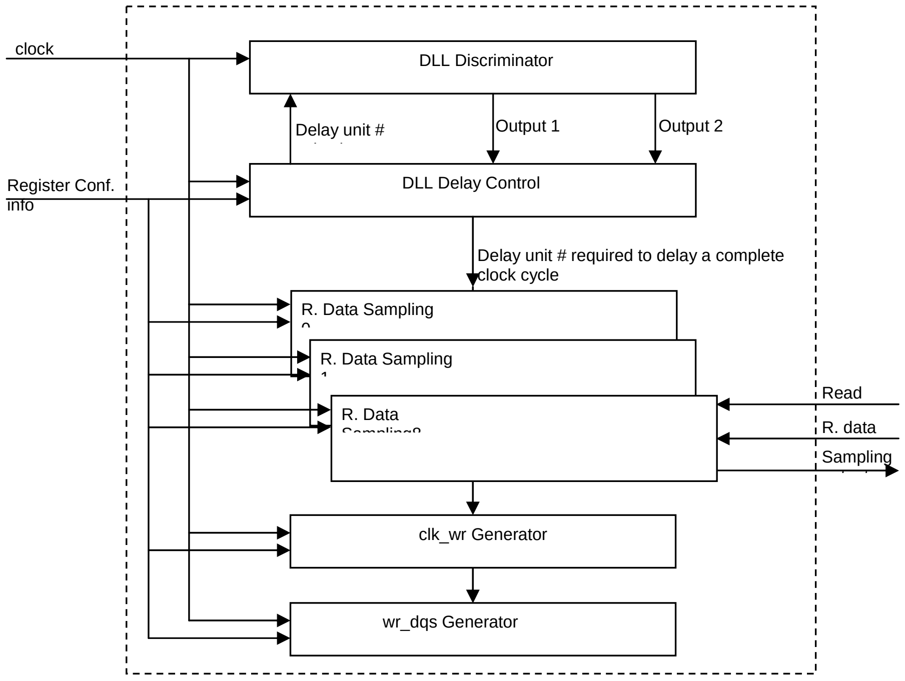
4）Data I/O Cell

DDR2 data (DQ) are bi-directional signals. So it is necessary to generate output enable signals. The input dq signal is not handled in the I/O cell but is directly transmitted to the Delay Compensation Circuit (DCC) of the Core Control Logic Module. This type of I/C Cell is showed in the block diagram below:

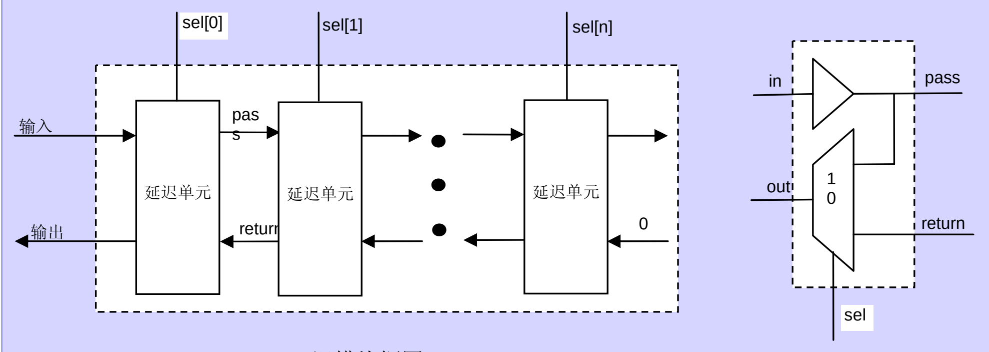


4.4.5. Delay Compensation Circuit (DCC)

The DCC is part of the core control logic of the memory controller. It is the core of the DDR2 memory controller PHY module. It plays a crucial role in ensuring that the memory module is capable of reliably sampling the control commands and write data from the memory controller and that the memory controller is able to sample the read data from the memory module. How to work: to start with measuring the number of delay units needed to delay one memory clock cycle; then find the number of delay units needed to delay each read/write data strobe (dqs) referring to the configuration register parameters (In the bypass mode, the bypass register provides the absolute number of the delay units. In the normal mode, the configuration register provides % of delay write and read data strobes.); finally, add the delay units to all the signals to be delayed, including 9 read data strobes, 1 write data strobe and 1 write data signal. The block diagram below shows the complete DDC schematic.



A delay line module is included in the DLL Discriminator and each of the Read Data Sample Modules (x 9) as well as the clk_wr and wr_dqs generation modules. Each delay line module consists of 400 tightly arranged identical delay units. The signal to be delayed is input through the module input. A selection signal determines how many units should be delayed. The block diagram below shows the complete delay line module as well as the logic of each delay unit. The actual delay of each unit is the sum of the delays of a buffer and a mux.



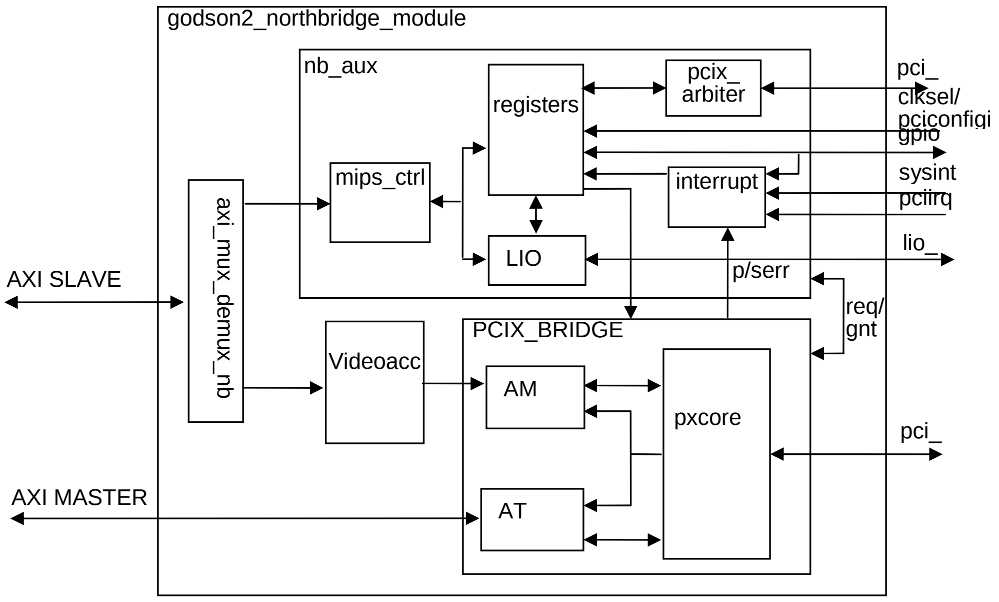
延迟模块框图　　延迟单元

批注 [S9]: In ENGLISH

5. Northbridge Module Design

批注 [S10]: Will not be in L2E document

Shown in the diagram below, the NB module of the L2F includes the axi_mux_demux_nb, which, a AXI 1-to-2 interface, dispatches CPU requests; the Video accelerator which, a video accelerator, includes zoom, yuv2rgb, etc; the PCIX_BRIDGE which, the AXI PCIX bridge, includes the AXI Master and Target as well as the PCIX IP; and the nb_aux which, an AXI slave, includes NB registers, Local IO, interrupt controllers and the PCIX arbiter and other sub modules.



5.1. axi_mux_demux_nb Module

This module transmits a signal from the AX master to either of the downstream ports according to rules and returns signals from the Slave to Master. Its path consists of combinational logic, without any flip-flops. It contains only two 16-bit registers used for recording the direction in which CPU write data is transmitted.

The Naming Rules: The signals from AX master are prefixed with h_, while 10_ and 11_ are prefixed to the AXI signals from downstream Channels 0 and 1, respectively. The write address (AW) channel related signals contain wac string; the write data (W) channel, wdc string; the write response channel, wrc; the read channel, rac and rdc.

- **Channel Selection**

Here Channel 0 is specified as the main channel where the majority of data will move through. With a phyfisical address of 512M or ranging from 256 to 512M, all the transactions will be transmitted to Channel 0 when they hit the PCI header, configuration space, PCI IO and PCI MEM. Since prefetch is not allowed in parts of the PCI space, a special care has been dedicated to the module read operation, preventing the CPU speculation from changing device status. A memory window (mem_win) is defined in the prefetch CPU physical address on the PCI channel. A module read request to the PCI will be transferred to Channel 1 if it falls out of the window. The Channel 1 will make a default response to the undefined address space (Full zero for read; reject for write.).

- **Write Channel Algorithm**

Since the AXI write address cycle corresponds to a number of data cycles, the transmission direction of a certain transaction must recorded in the module. Here, Channel 2 is designed to only accept access from CPU, so it is enough to record the ID from CPU. Two 16-bit registers （wc_l0_idmap, wc_l1_idmap） record the remission direction of all the IDs: write data will be transmitted in one single direction iff (in and only if) they see a record on the idmap in the same direction or that the same ID will almost complete its transfer in this direction on the present write address channel.

IDs are inserted into the two idmaps when the write address channel succeeds in shaking hands, while they are deleted when last data transfer has completed on the write data channel. Deletion superior to insertion. An ID can not and should not be inserted into a idmap when its write address and the last write data has completed in one single cycle.

The write of the same IDs is not supported in this module. A write request will be kept from transmitting to any downstream channel if the ID of this write address channel is recorded in the idmap. (the cache2mem of the L2F ensures the write of the same IDs won't appear before the completion of one write ID, without the need for a mechanism preventing the same IDs)

- **Return Channel**

Both write response and read return channels transfer data back to the master. Because the AXI slave cannot start a transmission by itself and a continuous feedback is impossible, a fixed-priority method (Channel 1 is superior to Channel 0) is adopted. Seldom accessed, the use of Channel 1 at the control side of the 1-of-2 data selector can reduce the switch operations, too.

5.2. PCIX_BRIDGE Module

5.2.1 Description

The L2F's PCIX/PCI controller includes two parts: the PCIX/PCI controller, which functions as a complete PCIX/PCI state machine for the AXI PCIX/PCI controller; and the interface converter from the AXI interface to the controller application interface of the Synopsis's PCIX/PCI controller, which provides data and request flow and buffer control, and converts data/requests from AXI to the controller or controller to AXI.

Our converter module includes two sub modules: AM which provides a channel for the processor core to access the PCI bus and the controller's internal registers through the AXI interface (including PCI header and the controller's internal configuration registers.); and AT which converts dma requests and data received by the controller on the PCI bus into the AXI bus requests and data, and then feeds back the return data from the AXI bus that are converted at the controller, through the PCI bus to the device that sends the dma request. The description of the L2F's PCI/PCIX controller architecture will focus on these sub modudes.   .

5.2.2 AM Submodule

This module includes a state machine which controls the conversion from the AXI interface to the controller interface, a 8-item table recording the rid of a transmitted request and the seq num returned by the controller, a one-cache-row-sized read data buffer which is used for buffer/conversion of data formats, and a set of address decoding/translation modules.

- **AM State Machine**

The AM State Machine operates in two states: idle and busy. It receives a request from the AXI port in the state idle, and then enters the state busy where it needs to record translated/decoded addresses, pci/pcix commands used by requests, and request types (read/write, whether to enable the module operation, keyword priority, access to the controller's internal registers, and the rid of read requests.). In the state busy , if the received request is not an access to the controller's internal registers, the state machine will initiate a request to the Synopsis IP's AM port. Upon the completion of requests/data transmission and shaking-hands operation on the AXI port, the state machine switches to the idle state. When it is an access to the controller's internal registers, the state machine will initiate an access to the register port. After completing the access and shaking-hands on the AXI port, the state machine goes back into the state idle. It does not receive a new AXI request in busy state. The read request will be processed first when read/write quests from the AXI port arrive at the same time.

- **AM Read Request Table**

The table records the type and rid of the read request as well as the seq num returned from the IP when the IP receives a read request to the PCI bus from the state machine. The rid and read type can be found in the table referencing to the seq num when the IP returns read data.

- **Read Data Buffer**

The buffer stores single read (non-module read) and module read data when the AXI interface does not receive read data.

- **Address Decoding/Translation**

This module translates the AXI address into the PCI address, and creates commands and type information for requests according to the AXI port address related window.

Now next paragraphs describe different AXI request processes.

- **AXI Read:**

a) The State Machine (SM) receives a read request from the AXI port and enters the state busy .

b) If it is a request to access the internal registers, the SM makes a request to the internal register port and waits for its response. Upon receiving the response, the SM releases read data and shaking-hands signals on the AXI port. When the shaking-hands process ends at the AXI port, the SM goes back into the idle state and the AXI read request has completed. If the request is to access the PCI bus, the SM initiates a request to the IP. The Read Request Table will record the related contents when the IP responds to the request. Finally the SM enters the state idle.

c) When the IP receives the returned read data on the PCI bus, the related rid and type are found in the Read Request Record Table and are fed back with related data on the AXI port. If the AXI port fails to complete shaking-hands for receiving data, the read data and shaking-hands signals will remain active, with the SM disabled to receive and send requests. When the shaking-hands is established and the data is transmitted, this record will be deleted from the table. Now, the AXI has completed one cycle of read and transmission. If the IP receives a master abort or other error status, the SM will respond with full F and complete shaking-hands at the AXI port.

- **AXI Write:**

a) The SM enters the state busy after receiving a write address request from the AXI port.

b) If it is a request to access the internal registers, the SM makes a request to the internal register port and waits for its response. Upon receiving the response, the SM releases shaking-hands signals on the AXI port. When the shaking-hands process ends at the AXI port, the SM goes back into the idle state and the AXI write request has completed. If the request is to access the PCI bus, the SM initiates a request to the IP's AM port, and waits for it to read the data on the AXI port. When the data are transmitted, the SM establishes shaking-hands for bresp at the AXI port. The SM rejects new requests during the shaking-hands. When this process ends, the SM goes back into the idle state . Now, this cycle of write and transmission ends. If the IP provides error status in the cycle of write and transmission, the SM will respond by rejecting the data in this cycle and completing shaking-hands at the AXI port. Now this AXI cycle of write and transmission has completed.

5.2.3 AT Module

The AT module includes a 8-entry request queue, a 20x64-bit write data queue, a 24x64-bit read data queue, three state machines which initiates a write operation on the AXI bus, transmits read data in the PCI and PCIX modes, respectively, a 4-entry seq num queue, a 8-engry rlen queue, and a 4-entry read data return status queue.

- **Request Queue**

The Request Queue records write and read requests transmitted by the IP at the AT port in the PCI/PCIX mode and PCIX respectively. Each of the queue entries contains the 64-bit addr, we which indicates write when high (indicates read when low), byte count which indicates the byte count in the pcix transmission, processed which indicates the current entry is processed, valid which indicates the current entry is valid, and order which indicates the current entry must be executed in a given order.

The maintenance of the queue uses three pointers: wptr which indicates the current write position, rptr which current read position, and cptr which indicates the position of the current processed entry. If the valid of all the entries has a valid bit, the queue is full.

Before a request from the AT port joins a queue, the addr, we, byte count and order are set to values, respectively, with the valid set and the processed cleared. When a state machine reads the request from a queue, the processed is set. When the request has completed on the AXI bus, the valid is cleared. When a request has a valid value for the order, the state machine cannot read the entry from the queue unless all other previous requests have been processed on the AXI bus.

- **Write Data Queue**

Each of the write data queue entries contains 64-bit data and one tag which identifies whether the current entry is the last data of one cycle of PCI/PCIX write and transmission.

- **Read Data Queue**

Each of the read data queue entries contains 64-bit data and two tags which identify whether the current entries are the last item of one adb and the last data of one cycle PCIX read and transmission, respectively.

- **AXI Bus Write State Machine**

This module initiates a write request on the AXI bus. When it finds a valid and unprocessed write request from the read port of the request queue, the SM will start to work to set the current write request as processed. When it finds write data from the write data queue, the SM will transmit data on the AXI bus. This is a fixed-length write operation with len=1 and size=128 bits. If the start or end data of a write request is not aligned with the cache row, the SM fills in the invalid strb. The write address is transmitted with the last data of each cycle of write and transmission on the AXI bus. The SM will not send new data until the write address is received. It reads the last write data from the queue and then clears the valid of the current write request.

- **Read State Machine in PCI Mode**

This state machine initiates a read request on the AXI bus in the PCI mode. When it receives a read request from the AT port, the SM will start to work indicating that it will not receive a new request until the current request is processed. The SM starts with a read request with len=7 on the AXI bus. Then it sends a read request with len=3 when the first AXI read request data is returned. So the SM will not issue the next read request until the data of the processed request is returned. This process continues until all the data on the PCI bus are transmitted. The SM will reset the read and write pointers and counter of the read data queue when the all the data on the PCI bus side are transmitted or an error status occurs. If some of read data still do not return from the AXI side at the moment, the SM will enter wait state , waiting for the missing data to come back. When the data on the AXI come back, the SM operates in the state idle, ready for new read requests.

- **Read State Machine in PCIX Mode**

This state machine initiates a read request on the AXI bus in the PCIX mode. The L2F has up to 32x64bit data buffer entries on the dma read data channel (24 entries on the AT interface).The number of remaining buffer entries on the current dma read data channel will get updated once whenever the SM transmits a new read request. The first read request on the AXI bus has len of 7, the subsequent requests, len of 3. If the number of remaining buffer entries is greater than or equal to its threshold (configured through the Northbridge configuration registers, the default threshold=8), the SM will initiate a new read request on the AXI bus. It goes back into the idle state when the last request is transmitted.

- **Rlen Queue**

The AXI Protocol requires that a single transmission should not cross the range of 4K. In this case, the read SM needs to split one request out of range into two. So we use the rlen queue to store the len domain of the AXI read request transmitted each time. The len domain of the current return read data is used to update the entry which corresponds to the ongoing PCIX read request in the read data return status queue. When the last data of one cycle of the AXI read data returns, the entry occupied by the previous return data will be released. The PCIX read SM is not allowed to initiate a new read request when the rlen queue is full.

- **Read Data Return Status Queue**

Each entry of this queue records the states of a PCIX read request which is enqueued but has not received the last data from the AXI bus. These states include calculating whether the current return data is the low level address of the adb boundary, and calculation of the number of the remaining data of current request that have not come back yet. The read SM will, according to the number of remaining unreturned data calculate whether the current return data is the last data of one cycle of PCIX read and transmission. Because what is initiated is only the read request aligned with the cache address, each entry of this queue still contains the domain which records whether the current return data should be rejected.

- **seq num Queue**

The AT port initiates a read request with a seq num. The returned data needs to provide the seq num of the corresponding request at AT port. When a read request at the AT port enqueues, the seq num queue records the seq num corresponding to the request. The 4-entry seq num queue means that up to 4 read requests can be processed simultaneously. The AT module will not receive any new read requests when the seq num queue is full.
The next paragraph will describe different PCI/PCIX request processes.

- **PCI/PCIX Write**

The AT module can receive a write request when the request queue has empty entries and the write data queue is ready (the write data queue has empty entries in the PCI mode; it contains empty entries at least that hold one adb in the PCIX). The received write request first enqueues to the request queue, while its corresponding data joins the write data queue. Once seeing this request at the queue read port, the write SM will initiate and maintain a write request with len=1 on the AXI bus until the last data of this write request is transmitted on the AXI bus. Because it is a fixed-length AXI write request, the write SM may insert invalid strb between the first AXI write request and the last AXI write transmission.

- **PCI Read**

Only one PCI read request is processed in the AT module. That means it does not receive a new read request when a previous request is being processed. The PCI read state machine operates in several states: idle which indicates the SM is free or idle, rfirst which indicates the SM is sending the first read request, rlast indicating the SM is waiting for the last read data to come back, rdata, next indicating the SM is sending a read request, and rhold indicating the SM is holding the last read request. During a PCI operation, because the SM cannot predict when the transmission of the last data cycle ends, a signal “flush” is generated at the end of the PCI bus operation, rejecting the unreturned data and those in the read data queue.

1) If AT receives a new read request, idle—>rfirst；
2) If the current request is in the range of 4K, rfirst—>rlast；
   If not, rfirst—>rdata；
3) When last AXI read data returns, if SM is in rlast or receives the signal flush, it goes back to idle state; if not, it enters rnext.
4) If SM receives flush in rnext, it enters the state rhold, holding the unfinished AXI request signal; if not, it goes into rdata after a request transmission.

- **PCIX Read**

The AT module can receive up to 4 PCIX read requests. The PCIX read state machine only transmits a read request. The AXI return data is maintained by the read data return status queue. The PCIX read SM has three states: idle which indicates the SM is free or idle, rfirst which indicates the SM is sending the first read request, and tran which indicates the SM is transmitting the remaining requests. The AXI Protocol requires that a single transmission should not cross the range of 4K. In this case, the read SM needs to split one request out of range into two.

In order to improve the efficiency on the ddr side, the first request within the range of 4K is assigned len=7, the subsequent requests within the same range are set to len=3. When recognizing the read request from the request queue, the SM will first calculate how many cache rows of data are fetched in total according to the address of the read request and byte count and then enter the state first. The SM switches to the idle state when the first AXI read request is finished. Aligned with the cache row, our proposed operation fetches more data than required (less than one cache row). Once the PCIX read data is transmitted on the PCI bus, the SM will have to eliminate the remaining data of this cycle in the read data queue.

5.3. nb_aux Module

This module converts an AXI signal into an internal bus signal(WishBone) at the module mips_ctrl, and uses the address decoded at the AdrDecode to read and write the slave modules including Northbridge configuration registers and LocalIO.

5.3.1. mips_ctrl Module

This module is an AXI-to-WISHBONE interface converter. It performs conversion of a request from the 128-bit AXI interface to the 64-bit WISHBONE interface. The WISHBONE is an easy bus for interconnecting internal modules, much simpler than the AXI bus. In the L2F's I/O modules, the WISHONE connects the Northbridge 's former Local IO and internal control register module. Because these are low-speed modules which have lower demand for performance, we designed the AXI-to-WISHBONE interface converter module with in mind higher conversion efficiency with lower resource consumption. For this reason, this module only processes one AXI request at a time and does not receive a new request until the current AXI request is finished (The transmission of all the signals and the shaking-hands of the AXI bus signals have completed.). The read request is first accepted when the AXI read and write requests arrive at the same time.

5.3.2. AdrDecode Module

This module outputs a set of hit signals according to an access address, using four signals: cpu_rom, cpu_localio, cpu_boot, cpu_nb_config. The address space assignment is shown the table below:

Address Space	Size	Location
0x00000000~0x07ffffff	128M	LocalIO – IO
0x08000000~0x0fffffff	128M	LocalIO – ROM
0x10000000~0x13ffffff	64M	PCIX_BRIDGE PCI lo0
0x14000000~0x17ffffff	64M	PCIX_BRIDGE PCI lo1
0x18000000~0x1bffffff	64M	PCIX_BRIDGE PCI lo2
0x1c000000~0x1dffffff	32M	LocalIO – ROM
0x1e000000~0x1fbfffff	28M	LocalIO – IO
0x1fc00000~0x1fcfffff	1M	LocalIO – ROM
0x1fe00000~0x1fe000ff	256B	PCIX_BRIDGE PCI Header

0x1fe00100~0x1fe001ff	256B	Nbcfg
0x1fe80000~0x1feffff	512K	PCIX_BRIDGE PCI config space
0x1ff00000~0x1fffffff	1M	LocalIO – IO
Else	N/A	PCIX_BRIDGE PCI memory space

5.3.3. Register Module

The Northbridge configuration registers

The NB configuration registers include internal module configuration, GPIO, interrupt status and CPU configuration and sampling. Those who perform the same functions as the registers of Bonito share the same addresses, minimizing software modifications.

All the registers are 32 bits, and their names are suffixed with_r. Their output uses their names with the suffix –r removed. The registers' unwritable bit should not be connected to the output, thus simplifying the code. The flip-flop with output floated are removed using a comprehensive tool.

Both read and write operations are performed in two cycles: the address-matched register output is transferred to the output register in the first cycle; read or write operation is performed in the second cycle. It is a write operation, a new value is generated by operation of the output register's value and the value to write together with the write byte enable. The new value is written in the second cycle.

The external signal is sampled at each beat.

Address	Register	Description
00	poncfg	Power on
04	gencfg	General
08	liocfg	LocalIO
0C	reserved	
10	pcimap	PCI mapping
14	pcix_bridge_cfg	PCI/X bridge
18	pcimap_cfg	PCI read/write device address
1C	gpio_data	GPIO data
20	gpio_en	GPIO direction
24	intedge	interrupt pulse trigger
28	reserved	
2C	intpol	interrupt valid level
30	intenset	interrupt enable set
34	intenclr	interrupt enable clear
38	inten	interrupt enable
3C	intisr	interrupt request vectors
40	mem_win_base_l	memory window base address low 32 bits

44	mem_win_base_h	memory window base address high 32 bits
48	mem_win_mask_l	memory window mask low 32 bits
4C	mem_win_mask_h	memory window mask high 32 bits
50	pci_hit0_sel_l	PCI window 0 control low 32 bits
54	pci_hit0_sel_h	PCI window 0 control high 32 bits
58	pci_hit1_sel_l	PCI window 1 control low 32 bits
5C	pci_hit1_sel_h	PCI window 1 control high 32 bits
60	pci_hit2_sel_l	PCI window 2 control low 32 bits
64	pci_hit2_sel_h	PCI window 2 control high 32 bits
68	pxarb_config	PCIX arbiter configuration
6C	pxarb_status	PCIX arbiter status
70	reserved	
74	reserved	
8	reserved	
7C	reserved	
80	chip_config0	Chip configuration
84	chip_config1	Chip configuration
88	chip_config2	Chip configuration
8C	chip_config3	Chip configuration
90	chip_sample0	
94	chip_sample1	
98	chip_sample2	
9C	chip_sample3	
A0	ov_ctrl	Video accelerator control register
A4	ov_ori_size	Original image size
A8	ov_zoom_size	Zoom image size
AC	ov_fb_base	frame buffer base address of the current image on screen
B0	ov_fb_stride	frame buffer width (stride) for the current image on screen
B4	ov_hor_zoom1	Horizontal zoom control 1
B8	ov_hor_zoom2	Horizontal zoom control 2
BC	ov_ver_zoom	Vertical zoom control
C0	ov_x_pos	X coordinate of the current image position on screen
C4	ov_x_width	Screen width
C8	ov_fb_base	frame buffer base address
CC	ov_fb_mask	frame buffer range mask

These tables below provide register specifications.

Bit-field	**Field Name**	**Acces s**	**Reset Value**	**Description**
CR00: poncfg				

15:0	pcix_bus_dev	Read Only	lio_ad[7:0]	Bus and device # required for CPU fetch in the PCIX Agent mode.
15:8	reserved	Read Only	lio_ad[15:8]	
23:16	pon_pci_configi	Read Only	pci_configi	pci_configi value
31:24	reserved	Read Only		
CR04: gencfg				
0	ov_en	Read/Write	0	video accelerator enable
31:1	reserved	Read Only	0	
CR08: liocfg				
1:0	reserved	Read Only	0	
6:2	rom_wait	Read/Write	5'b11111	Rom data read delay (cycle)
7	rom_width	Read/Write	pci_config[0]	Rom data width
12:8	io_wait	Read/Write	5'b11111	iO data read delay（cycle）
13	io_width	Read/Write	1'b0	io data width
14	iopf_en	Read/Write	1'b0	Io device prefetch enable
31:15	reserved	Read Only	0	
CR10: pcimap				
5:0	trans_lo0	Read/Write	0	pci_lo0 window mapping address high 6 bits
11:6	trans_lo1	Read/Write	0	pci_lo1 window mapping address high 6 bits
17:12	trans_lo2	Read/Write	0	pci_lo2 window mapping address high 6 bits
31:18	reserved	Read Only	0	
CR14: pcix_bridge_cfg				
5:0	pcix_rgate	Read/Write	6'h18	Readout threshold to ddr in the PCIX mode
6	pcix_ro_en	Read/Write	0	Does PCIX bridge allow write to overtake read?
31:18	reserved	Read Only	0	
CR18: pcimap_cfg				
15:0	dev_addr	Read/Write	0	AD bus is high 16 bits during PCI read/write configuration

16	conf_type	Read/ Write	0	Configures read/write type
31:17	reserved	Read Only	0	
CR1C: gpio_data				
3:0	gpio_out	Read/ Write	0	GPIO output data
15:4	reserved	Read Only	0	
19:16	gpio_in	Read/ Write	0	GPIO input data
31:20	reserved	Read Only	0	
CR20: gpio_en				
3:0	gpio_en	Read/ Write	F	input at high; output at low
31:4	reserved	Read Only	0	
CR50,54/58,5C/60,64: pci_hit*_sel				
0	reserved	Read Only	0	
2:1	pci_img_size	Read/ Write	2'b11	00: 32 bit；10: 64 bit；others: invalid
3	pref_en	Read/ Write	0	Prefetch enable
11:4	reserved	Read Only	0	
62:12	bar_mask	Read/ Write	0	Window size mask
63	burst_cap	Read/ Write	1	Enable burst transfer?
CR68: pxarb_config				
0	device_en	Read/ Write	1	Enable external devices
1	disable_broken	Read/ Write	0	Disable broken master devices
2	default_mas_en	Read/ Write	1	Park bus to default master device.
5:3	default_master	Read/ Write	0	Park a bus to the default device #
7:6	park_delay	Read/ Write	0	Delay from the time when no device requests for the bus to the time when the bus is parked to a default device. 00: 0 cycle 01: 8 cycle 10: 32 cycle 11: 128 cycle

15:8	level	Read/ Write	8'h01	Devices at Level 1
23:16	rude_dev	Read/ Write	0	Support for some specific devices
31:13	reserved	Read Only	0	
CR6C: pxarb_status				
7:0	broken_master	Read Only	0	Broken master devices (cleared to zero when the disable strategy is changed)
10:8	last_master	Read Only	0	Master device that uses the bus last
31:11	reserved	Read Only	0	
CR80: core_config				
2:0	freq_scale	Read/ Write	3'b111	Software-controlled frequency scaling
3	disable_scache	Read/ Write	0	Disable L2 Cache
4	imp_first	Read/ Write	1	Keyword priority
7:5	reserved	Read/ Write	0	
8	disable_ddr_conf	Read/ Write	0	Disable DDR2 configuration port
9	ddr_buffer_cpu	Read/ Write	1	Allow the data not to enter the memory when the write operation has completed?
10	ddr_buffer_pci	Read/ Write	1	Allow the data not to enter the memory when the write operation has completed?
31:11	reserved	Read Only	0	

5.3.4. Interrupt Module

This module sets the interrupt line polarity, enables and transfers interrupt signals. It sets all the external interrupts as active and resets them as active low. The INT0-3 has an unchangeable interrupt enable bit of 1, which is controlled by the state machine cp0. Other external interrupts have their respective enable bit. The pulse mode interrupts (e.g. PCI_SERR) is selected by the configuration register intedge. The interrupt handler uses intenclr to clear pulse records.

The table below provides interrupt line connections:

Bit field	Control Register			Interrupt Source
	intpol(acc/def)	intedge(acc/def)	inten(acc/def)	
3 : 0	RW / 0	RW / 0	RW / 0	GPIO

7 : 4	RO / 0	RO / 0	RW / 0	PCI_INTn
8	RO / 1	RO / 0	RW / 0	PCI_PERR
9	RO / 1	RO / 1	RW / 0	PCI_SERR
10	RO / 1	RO / 1	RW / 0	denali
14 : 11	RW / 0	RW / 0	RESERVED	INTn
31 : 15				reserved

5.3.5. LocalIO Module

This module connects simple ROM and IO devices. These devices feature separate addresses, data, control lines and no shaking-hands.

The configuration of the module parameter liocfg is described in the table below. The fields bus_period and big_mem are retained in the register. The bus_period is not available in the L2F for lack of access to power-on configuration. The big_mem is used for output 32 bit address support.

Bit Field	Field	Description
1:0	bus_period	Available in three values representing the PCI speed: 33MHz, 66MHz and 133MHz, keeping the value of “wait”the same at different speeds. The frequency is detected by software in the L2F. This field is retained in the register.
6:2	rom_wait	Rom data read delay (cycle)
7	rom_width	Rom data width（0: 8bit, 1: 16bit）
12:8	io_wait	io data read delay (cycle)
13	io_width	io data width（0: 8bit, 1: 16bit）
14	iopf_en	io prefetch enable
15	big_mem	lio_addr address output control. This value is set to zero in the L2F.

For lower counts of pins, the address and data bus are multiplexed, with high level addresses held by the Adlock controlled transparent latch.

big_mem	Address Structure
0	{ADLock(lio_ad[15:0]), lio_addr[7:0]}
1	{ADLock({lio_addr[7:0], lio_ad[15:0]}), lio_addr[7:0]}

The L2F’s LocalIO has a designated capacity of only 32M max. The 24-bit address data bus and 16-bit data width are enough, so the configuration of big_mem is reserved. (Notes: the undefined low 256M of the physical address to the Northbridge is cut into two halves so that the ROM and IO hold one half each.)

5.3.6. Pcix_arbiter Module

This module operates in the PCIX and PCI modes, implementing a dual round-robin matching algorithm, bus parking and broken device isolation. For its configuration and state registers, see CR68 and CR6C.

This module includes a bus monitor, precedence logic and an arbitration state machine. The bus monitor outputs the information of the last owner to the precedence logic, which ranks and encodes the input requests to create the highest priority request that is decoded to generate the winning request in current precedence status, according to which, the arbitration state machine generates the signal GNT# at the right time.

Preventing the master that has gained an access to the bus from delay in initializing an operation, the bus monitor includes additional logic that automatically sees the delaying mast as the current master device. This delaying device might still be added to the broken device vectors. The isolation of broke devices are implemented in the pretreatment process where a request signal must be handled to enter the precedence logic.

For PCI devices using the signal hold like 82371, the active hold will prevent the arbitration state machine from switching to the GNT state.

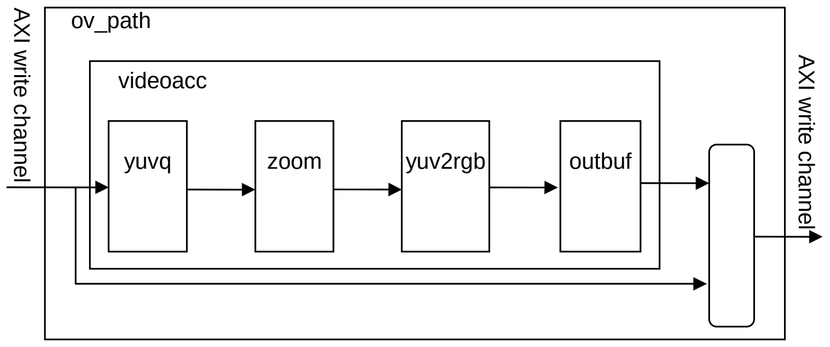
Note: the PCIX requires the arbiter:

a) When the bus is idle, the GNT# must be maintained for 5 cycles; this requirement is unnecessary in other cases.
b) To convert GNT#, an idle cycle has to be inserted.
c) Fair arbitration algorithm
d) All the I/O signals are directly connected with registers.

5.4 Video Acc Module

5.4.1 Video Accelerator Module Structure

This function is achieved by implementing image scaling and YUV/RGB format conversion, enabling the L2x processor to play mpeg2/mpeg4 movies without video card support for these functions. The L2E implements the video accelerator in the FPGA of the Northbridge, while the L2F integrates it on the chip, placed between the AXI crossbar switch and PCI controller. This module includes yuvq, zoom, yuv2rg band outbuf, around which the module ov_path is built for bypass selection. See the block diagram below.



- Ov_path Module

The Northbridge video acceleration is archived by inserting the module ov_path into the path from AXI write channel to PCIX_BRIDGE. The Northbridge configuration with additional video acceleration is shown in the diagram above.

The write requests that fall in the address range of 0x13f00000 ~ 0x13ffffff will be sent to the video module and then enter the yuvq module. The AXI one-to-two logic is the same as other parts (except the slight changes in address decoding), while the two -to- one logic is similar to the slave_link of the crossbar switch, except that the number of write requests that are allowed to exceed is 1 max. A round-robin arbitration scheme is established between two competing channels, reducing the priority of the party who just used the bus. It works in the way that the highest priority request is transmitted to the PCIX bridge when the bus is free and that the bus will get locked after shaking-hands has completed on the address bus. The signals valid and ready on the data channel are selected according to the current owner of the bus. The bus will unlock when the transmission has completed.

To avoid unexpected errors, exceptions or failures, the acceleration module can be completely shut down by the signal ov_en, making the module ov_path logically equal to a straight through connection. Note that it would be disastrous to remove the signal ov_en when the acceleration is working.

- YUVQ module

The video acceleration module supports YUV 422 and 444. The current data format is selected by software-configured registers. When ov_yuvfmt = 1, the current format is YUV 444; if not, YUV 422.

	L2E Address	L2F Address	Remarks
Y Add	0x13f0_0000	0x13f0_0000	Size: a row
U Add	0x13f0_0020	0x13f0_0020	Size: a row
V Add	0x13f0_0040	0x13f0_0040	Size: a row
Y1 Add	0x13f0_0040	0x13f0_0060	Valid in 422 format, a row size.

The Yuvq module includes 3 queues (yq/uq/vq) which are built of registers or multi-port RAM to store Y/U/V data respectively. It is supported with two 1-byte read ports and one write port whose data width ranging from 1 to 16 bytes, depended upon the coming data to write. These queues stores data in YUV 422 format, with Queue Y being 128 bytes, Queues U and V being 64 bytes, respectively. In YUV 444 mode, the reception logic will merge and compress data in such way that adjacent U and V share the value of UV at an even address while removing the value of UV at an odd address. When receiving write data, each of the queues is required to receive next category of data requests every time the queue reaches a full row. The queue identifies the type of current data according to the locally maintained inside addresses instead of outside ones, reducing the risk of dead lock due to software neglects, thus increasing the system robustness. In YUV 444 format, the module receives data in the Y-U-V order, one cache row for each category, corresponding to the one row space starting with the Y/U/V addresses 0x0/0x20/0x40 respectively.

The YUVQ module's communications signals with the Zoom module include pointers of head, tail, read and write as well as the corresponding YUV data. These modules exchange data in standard YUV 444 format. This means the YUVQ module has to convert data from YUV 422 to 444 before communicating with the Zoom. They select the V queue's pointers as their pointers for communication because the data V arrives later than the others during operation. Since the pointers remain at the same level, whether the queue is full or not, a high level of 8 bits is added to the pointers for easy indication of full and empty queues. If the 8-bit head and tail pointers are identical, the queue is empty; if they are same at low 7 bits except the higher bit, the queue is full. The YUVQ module accepts the two read pointers from the Zoom module to provide data needed by the Zoom. Because the data is saved in the YUV 422, the Queue Y indexes using the pointer's low 7 bits. Sharing the same UV value at two adjacent points, the U and V queues index using [6:1] bits.

- Zoom Module

This module scales and transfers the raw YUV data from the YUVQ module to the module va_outbuf. It also requires each transmitted segment to include the number of dots after zoom-in and addresses in the frame buffer for the use of the va_outbuf.

Main I/O Description:

input zoom_allow; //Allow data transfer?

input ov_zoom_en; // Enable zoom?

input ov_sync; //reset signal, which, if set up, enables next cycle to enter frame_start.

input ov_rgb_24; // rgb output format, which influence fb_offset computing.

input ov_rgb_32; // the same as above

input[7:0] qtail; // the pointers of acquisition queue in the module yuvq

input[23:0] ori_yuv0; // raw YUV data obtained from the YUVQ's acquisition queue according the qread 0

input[23:0] ori_yuv1; // raw YUV data obtained from the YUVQ's acquisition queue according the qread 1

input[16:0] ov_stepx; // Horizontal scaling

input[16:0] ov_stepy; // Vertical scaling

input[10:0] ov_segment_size; // how many dots are in each segment (64 original dots) when zoomed (estimated value)?.

input[18:0] ov_size_mul_step; // ov_segment_size multiplying ov_stepx

input[10:0] ov_last_segment_size; // how many original dots are in the last segment.

input[10:0] ov_ori_win_width; // Original window width

input[10:0] ov_ori_win_height; // Original window height

input[10:0] ov_zoom_win_width; // Zoomed window width

input[10:0] ov_zoom_win_height; //zoomed window height

input[12:0] ov_zoom_win_x0; // on-screen X coordinates of the dot in the left top corner of an image

input[10:0] ov_scr_width; // screen resolution (horizontal lines)

input[31:0] ov_win_fb_base; // base address of an image in the frame_buffer

input[31:0] ov_fb_stride; // address incrimination between lines on the screen in the frame buffer

output[7:0] qhead; // head pointer to YUVQ

output[6:0] qread0; //read pointer to YUVQ

output[6:0] qread1; // read pointer to YUVQ

output[23:0] zoom_yuv0; // Zoomed data that is transmitted

output[23:0] zoom_yuv1; // Zoomed data that is transmitted

output zoom_yuv0_valid; // Is the transmitted yuv0 valid?

output   zoom_yuv1_valid;   // Is the transmitted yuv1 valid?

output   zoom_yuv0_strb;// Is the dot corresponding to yuv0 on screen?

output   zoom_yuv1_strb;// Is the dot corresponding to yuv1 on screen?

output[10:0]   zoom_segment_size;   // how many dots are in the current segment if zoomed?

output   zoom_addr_valid;   // valid address signal

output[31:0]   zoom_fb_offset; // current segment address in the frame buffer.

This module scales raw images based on the stepx and stepy. A zoomed dot has the same yuv value as an original dot closest to it. Horizontal scaling is completely performed by hardware. while vertical scale-up is implemented by hardware, scale-down is achieved by removing unwanted rows through software.

This module divides data in the original Row 1 into multiple segments each of which typically corresponds to original 64 dots except for last segment. When a segment starts, the module calculates how many zoomed dots the segment will deliver and the address in the frame buffer, according to the zoom rate, while transmitting the signal addr_valid signaling the module va_outbuf a new cycle.

The state machine enters the process status if the difference between qhenad and qtail is greater than segment_width (typically 64). In this state, it will continue to deliver all the segment data only if the zoom_allow is enabled. After zooming, if the posx_old of the dot original coordinates is greater than the end address of the segment, the current segment transmission will complete, and a new segment starts in the next beat. For width enlargement, the original segment might need to correspond to multiple horizontally parallel segments after zooming. The qhead will not move ahead until these zoomed segments are created and delivered. This means one segment has completed and the module YUVQ can drop the used data. The state machine will enter the state line_start if the last segment of one original row (indicated by the internal signal, zoom_last_segment) has completed. It will go into the state frame_start if all the lines of one original frame have completed. The state machine will switch to the state frame_start whether by synac or by reset.

- Yuv2rgb Module

This module converts original pixel color format from YUV to RGB. It only performs YUV444 to RGB24 conversion, whatever YUV format the original image is. Other YUV format conversion is performed by software in combination with the module YUVQ. Other RGB format conversion is done the module outbuf. Two pixels are converted in each beat. The input YUV data is 8 bits each. The output RGB data is 8 bits each, too. The conversion uses a fixpoint algorithm described below. For operation accuracy, temporary 16-bit and 10-bit results are retained for multiplication and addition, respectively. The output RGB is checked for saturation. When the value is higher than 255, select 255; when it is lower than 0, select 0.

```
u = YUVdata[UPOS] - 128;
v = YUVdata[VPOS] - 128;
rdif = v + ((v*103)>>8);
invgdif = ((u*88)>>8) + ((v*183)>>8);
bdif = u + ((u*198)>>8);
r = YUVdata[YPOS] + rdif;
g = YUVdata[YPOS] - invgdif;
b = YUVdata[YPOS] + bdif;
```

- Outbuf Module

This module acquires and transmits the RGB data of zoomed dots according to the given format via the AXI bus. Its input signals include rgb_addr_valid which, the module triggering pulse, indicates the current segment_size and fb_offset are valid and the data transmission can start next beat; rgb0; rgb1; rgb0_valid; rgb1_valid; rgb0_strb; rgb1_strb; rgb_segment_size; and rgb_fb_offset. The rgb? represents 24-bit GRB data, while the rgb?_strb indicates now the dot is on the screen and output is needed. The rgb?_valid used to mean the data validity. But now this signal is only used with strb to generate byte enable because the module zoom will always remain active except for sync. The output interfaces are standard AXI master and rgb_allow.

This module has two separate parts: input and output. The input state machine will jump to read_n_place from idle at addr_valid, unless the last segment transmission has completed the output of the AXI address cycle. The segment_size and fb_offset will remain unchanged until the first RGB data transmission has completed. So it is possible that the state machine does not acquire related data until it decides to jump to rand_n_place. The rgb_allow will not be provided before this.

The input 24-bit RGB data will, before entering the buffer, be saved in the required format:

```
rgb16 = {rgb[23:16+3], rgb[15:8+2], rgb[7:0+3]}
rgb24 = {rgb[23:16], rgb[15:8], rgb[7:0]}
rgb32 = {8'b0, rgb[23:16], rgb[15:8], rgb[7:0]}
```

The current collecting buffer can receive RGB data whenever it is free. If the current RGB data is higher than 128 bits, the section out of range can be stored temporally in the cb_cookie before written in the buffer in the next cycle. When it is the last cycle of RGB data transmission for itself, the signal allow will remove the next beat to synchronize with the module zoom.

The output part sees a new segment coming in with new_xfer_valid, new_xfer_addr, and new_xfer_length. It indicates it is ready to receive new segments using ob_accept_xfer. The awlength records the total required number of AXI transmission cycles. It will initiate next AXI transmission cycle if its value does not equal to -1 after the current transmission has completed.

5.4.2 Video Acceleration Register Description

This table below provides descriptions to the video acceleration registers each of which has 32 bits of capacity except ov_pci_base_mem.

Address	Registers	Description
0x1fe001a0	ov_ctrl	Control register for video acceleration
0x1fe001a4	ov_ori_size	Original image size
0x1fe001a8	ov_zoom_size	Zoomed image size
0x1fe001ac	ov_fb_base	The base address of current display image frame buffer
0x1fe001b0	ov_fb_stride	Horizontal width of current display image frame buffer
0x1fe001b4	ov_hor_zoom1	Horizontal zoom control 1
0x1fe001b8	ov_hor_zoom2	Horizontal zoom control 2
0x1fe001bc	ov_ver_zoom	Vertical zoom
0x1fe001c0	ov_x_pos	X coordinates of a display image
0x1fe001c4	ov_x_width	Horizontal screen width
0x1fe001c8	ov_fb_base	frame buffer base address
0x1fe001cc	ov_fb_mask	frame buffer range mask
0x13f00000	ov_pci_base_mem	32*4 byte display data buffer

Bit field	Field Name	Access	Reset Value	Description
ov_ctrl：Control register for video acceleration				
0	reset			
1	Y2R_EN			
2	ZoomEn			
4:3	inFMT			The image format from mplayer to the module 00: YV12 01: YUV422 10: YUV444
6:5	outFMT			The image format to display controller from this module 00: RGB16 01: RGB24 10: RGB32

10:7	resolution			The current resolution of display controller 0000: 320x200 0001: 320x350 0010: 360x400 0011: 640x200 0100: 640x350 0101: 640x480 0110: 720x350 0111: 720x400 1000: 800x600 1001:1024x768 1010:1280x1024 1011:1600x1200
ov_ori_size：Original Image Size				
10:0	X			
21:11	Y			
ov_zoom_size：Zoom Image Size				
10:0	X			
21:11	Y			
ov_fb_base：The base address of current display image in the frame buffer				
31:0	addr			The base address of current display image in the frame buffer can be calculated according to the zoom rate and dest_x and dest_y (start position of current window)in the software driver as well as the base address of the video adapter frame buffer.
ov_fb_stride：Horizontal width of current display image frame buffer				
31:0	stride			Horizontal width of current display image frame buffer
ov_hor_zoom1：Horizontal Zoom Control 1				
10:0	ov_stepx			Zoom rate, which is the result of horizontal original size divided by zoomed size and is stored as decimals in 5.12 format.
27:11	ov_seg_size			The number of dots contained in each segment: the number of 32÷ zoom rate +1 is rounded down.
ov_hor_zoom2：Horizontal Zoom Control 2				
10:0	ov_size_mul_step			Zoom rate x dots in each segment, that is, the product of above two registers.
28:11	ov_last_seg_size			Dots contained in last segment. The remainder of the dot counts from -0.5 to +0.5 divided by 32. Also 5.12 format.

ov_ver_zoom：Vertical Zoom Control				
16:0	ov_stepy			See the definition of ov_stepx.
ov_x_pos：X coordinates of a display image				
12:0	ov_x_pos			X coordinates of a display image is a signed number.
ov_x_width：Horizontal screen width				
10:0	ov_x_width			Horizontal screen width
ov_fb_base：frame buffer base address				
31:0	ov_fb_base			frame buffer base address
ov_fb_mask: frame buffer range mask				
31:0	ov_fb_mask			frame buffer range mask
ov_pci_base_mem：Display data buffer				
32 bytes	ov_pci_base_mem			Data that writes 64 dots each time: First 32 bytes: data y of first 32 dots; Second 32 bytes: data u of 64 dots; Third 32 bytes: data V of 64 dots; Fourth 32 bytes: data y of last 32 dots;

6. Reset and Clock Domain

The L2F integrates commercial DDR2 and PCIX IPs, and provides support for dynamic frequency conversion, so the L2F has a more complex clock design than L2E, as are the L2F's interrupt and reset designs.

6.1 Clock Domain

The L2F features 4 external input clocks: SYSCLK which generates a processor internal clock, coreclock by doubling frequency through a PLL; MEMCLK which generates a DD2 clock, memclock, by doubling frequency through a PLL; PCICLK which directly adopts input as a main clock of the Northbridge; and TESTCLK clock used for test purpose.

The L2F's internal main clock domains include coreclock, PCICLOCK and DDR2 memclock. The PCICLOCK is provided by the motherboard through the PCICLOCK pin, while the coreclock and memclock are generated by separate PLLs using the pin-provided clocks SYSCLK and MEMCLK.

The PLLs generating coreclock and memclock provide two output channels: PHI which operates in the frequency range of 1.6GHz-3.2GHz; and PHI2, the dividing frequency of PHI (division factor selectable), which is used by the L2F's coreclock and memclock. Alternatively, they can directly use the clocks provided by pin input clocks SYSCLC and MEMCLK, respectively. The generation of coreclock and memclock is controlled by the L2F's external CLKSEL[9:0] pin. See the table below.

PLL clock		**coreclock**		
clksel[2:0]	**PHI**	**clksel[4:3]**	**PHI2**	**coreclock**
000	SYSCLK*18	00	PHI/2/1	PHI2
001	SYSCLK*20	01	PHI/2/2	PHI2
010	SYSCLK*22	10	PHI/2/4	PHI2
011	SYSCLK*24	11	PHI/2/8	SYSCLK
100	SYSCLK*26			
101	SYSCLK*28			
110	SYSCLK*30			
111	SYSCLK*32			
PLL clock		**memclock**		
clksel[7:5]	**PHI**	**clksel[9:8]**	**PHI2**	**coreclock**
000	MEMCLK*18	00	PHI/2/1	MEMCLK
001	MEMCLK*20	01	PHI/2/2	PHI2
010	MEMCLK*22	10	PHI/2/4	PHI2
011	MEMCLK*24	11	PHI/2/8	PHI2
100	MEMCLK*26			
101	MEMCLK*28			
110	MEMCLK*30			

111	MEMCLK*32			

Among the three clocks, the roots of corclock and memclock are in the related select output of the module clock_control on the mid layer, while the PCICLOCK's root lies in the chip's pin input.
Data exchange takes place between coreclock, memclock and picklock. The clock domain crossing signals between them include:

- The main address and data paths between these domains are established by the asynchronous FIFO of the AXI switch module godson2f_arbiter_module.
- In the godson2f_arbiter_module's confreg submodule, 12*64-bit registers are assigned values by the coreclock, which are used in the PCICLOCK. Configured by software, the register settings are unchangeable.
- The DDR2 error signal controller_in transfers from DDR2 memclock to the PCICLOCK.
- Interrupts (external INT and GPIO, PCI_IRQn or the PCI exception, controller_int or the DDR2 error signal） are sampled and combined by the PCICLOCK before transferring to the coreclock module. A total of 6 levels of interrupt and one NMI signals in the IO module transfers from the PCICLOCK to the coreclock.
- The RESET signal travels from the SYSCLK to the PCICLOCK, coreclock and memclock; the softreset goes from the SYSCLK to the coreclock.
- The chip_sample signal is generated by three Compensation Cells, and sampled by the PCICLOCK.
- The chip_config[] is generated by the PCICLOCK, among which the freq_ctrl （chip_config[2:0]） transfers from the PCICLOCK to the PLL output domain, the pclock; the disable_scache (chip_config[3]) and imp_first (chip_config[4]) to the coreclock; the chip_config[9:8] to the memclock, for control of DDR2 register configuration; other signals to directly control the PAD.
- In the modules CLOCK_CTRL and FREQ_SCALE, the RESET for frequency conversion control moves from the PCICLOCK to the PLL output domain, the PCLOCK.

Special attention must be paid to the physical design of the above CCD transmission.
In addition to the three main domains, the L2F uses the rising edge of a few other clocks to trigger flip-flops. They include external input clock SYSCLK, PLL output clocks, etc. all of them are used in the modules CLOCK_CTRL or RESET_CTRL on the MID layer.
The SYSCLK in the RESET_CTRL module generates the reset signal VCOK, COLDRESET, HARDRESET, SOFTRESET and others, which will be used by various parts of the L2F one after

another. The PLL_CTRL in the CLOCK_CTRL module generates a PLL control clock, whose root is input via the processor PIN.

PCLOCK0 and PCLOCK1, the output clocks of the two PLLs, which are used in the FREQ_SCALE and TEST_CLOCK of the CLOCK_CTRL, control dynamic frequency conversion and generate a clock for real speed test. They are also transmitted via the CLOCKMODE_ Gate Control to an output pin for the PLL test. Their roots are in the PLL output.

The physical design should try to constrain the above signals to a limited zone. If designed by default, the double space is required. No double space must be approved by architecture designers.

6.2 Reset and Interrupt Signals

The reset signal SYSRESET is input to the L2F processor through its RESET pin, and are sampled and counted by the SYSCLK in the module RESET_CTRL on the MID layer, generating VCCOK, COLDRESET, HARDRESET and SOFTRESET, whose timing relation and functions as follows:

- VCCOK continues from SYSRESET for 2**8 beats, for use in controlling the PLL reset.
- COLDRESET replaces VCCOK and stays for 2**16 beats, to reset the processor core and NB.
- HARDRESET continues from COLDRESET for 2**8 beats, unused now.
- SOFTRESET lasts for 2**8 beats after HARDRESET, for initialization of BHT Table in the Fetch Module.

Note that the four reset signals follow a timing sequence. Working in combination with the processor core clock, the COLDRESET cannot be withdrawn until the PLLs get stable.

Further more, both COLDRESET and SOFTRESET connect with the negated SYSRESET in the test mode, allowing the test bench to control related reset signals.

The L2F’s interrupt signals include:

- NMI, externally nonmaskable interrupts；
- INT[3:0], external interrupts；
- GPIO[3:0], internal interrupts；
- PCI_IRQn[A, B, C, D], PCI interrupts；
- controller_int, DDR2 Controller error interrupts.

Combined and processed by the domain PCICLOCK, these interrupts generate 6 levels of typical interrupts and one NMI interrupt for the processor core with the CORECLOCK domain.

7. Pin-out and Signal descriptions

批注 [S11]: This table will be updated, with additional information as ball number, max freq, max load, ..etc…

4.1 Processor Pin Assignments

	Pin Name	Pin Type	Buffer Type	PU/PD	Function
001	DDR2_A0	O	SSTL_18 （1.8V）	No	DDR2 Address Bus
002	DDR2_A1	O	SSTL_18 （1.8V）	No	'
003	DDR2_A2	O	SSTL_18 （1.8V）	No	'
004	DDR2_A3	O	SSTL_18 （1.8V）	No	'
005	DDR2_A4	O	SSTL_18 （1.8V）	No	'
006	DDR2_A5	O	SSTL_18 （1.8V）	No	'
007	DDR2_A6	O	SSTL_18 （1.8V）	No	'
008	DDR2_A7	O	SSTL_18 （1.8V）	No	'
009	DDR2_A8	O	SSTL_18 （1.8V）	No	'
010	DDR2_A9	O	SSTL_18 （1.8V）	No	'
011	DDR2_A10	O	SSTL_18 （1.8V）	No	'
012	DDR2_A11	O	SSTL_18 （1.8V）	No	'
013	DDR2_A12	O	SSTL_18 （1.8V）	No	'
014	DDR2_A13	O	SSTL_18 （1.8V）	No	'
015	DDR2_A14	O	SSTL_18 （1.8V）	No	'
016	DDR2_CKp0	O	SSTL_18 （1.8V）	No	DDR2 CLK INPUT
017	DDR2_CKn0	O	SSTL_18 （1.8V）	No	'
018	DDR2_CKp1	O	SSTL_18 （1.8V）	No	'

019	DDR2_CKn1	O	SSTL_18 （1.8V）	No	'
020	DDR2_CKp2	O	SSTL_18 （1.8V）	No	'
021	DDR2_CKn2	O	SSTL_18 （1.8V）	No	'
022	DDR2_CKp3	O	SSTL_18 （1.8V）	No	'
023	DDR2_CKn3	O	SSTL_18 （1.8V）	No	'
024	DDR2_CKp4	O	SSTL_18 （1.8V）	No	'
025	DDR2_CKn4	O	SSTL_18 （1.8V）	No	'
026	DDR2_CKp5	O	SSTL_18 （1.8V）	No	'
027	DDR2_CKn5	O	SSTL_18 （1.8V）	No	'
028	DDR2_CKE0	O	SSTL_18 （1.8V）	No	DDR2 CLK ENABLE
029	DDR2_CKE1	O	SSTL_18 （1.8V）	No	'
030	DDR2_CKE2	O	SSTL_18 （1.8V）	No	'
031	DDR2_CKE3	O	SSTL_18 （1.8V）	No	'
032	DDR2_ODT0	O	SSTL_18 （1.8V）	No	DDR2 On Die Termination Control
033	DDR2_ODT1	O	SSTL_18 （1.8V）	No	'
034	DDR2_ODT2	O	SSTL_18 （1.8V）	No	'
035	DDR2_ODT3	O	SSTL_18 （1.8V）	No	'
036	DDR2_SCSn0	O	SSTL_18 （1.8V）	No	DDR2 Chip Select
037	DDR2_SCSn1	O	SSTL_18 （1.8V）	No	'
038	DDR2_SCSn2	O	SSTL_18 （1.8V）	No	'
039	DDR2_SCSn3	O	SSTL_18 （1.8V）	No	'
040	DDR2_BA0	O	SSTL_18 （1.8V）	No	DDR2 Bank Address Bus
041	DDR2_BA1	O	SSTL_18	No	'

			（1.8V）		
042	DDR2_BA2	O	SSTL_18（1.8V）	No	'
043	DDR2_WEn	O	SSTL_18（1.8V）	No	DDR2 Write Enable
044	DDR2_CASn	O	SSTL_18（1.8V）	No	DDR2 Column Address Strobe
045	DDR2_RASn	O	SSTL_18（1.8V）	No	DDR2 Row Address Strobe
046	DDR2_DQ0	I/O	SSTL_18（1.8V）	No	DDR2 Data Bus
047	DDR2_DQ1	I/O	SSTL_18（1.8V）	No	'
048	DDR2_DQ2	I/O	SSTL_18（1.8V）	No	'
049	DDR2_DQ3	I/O	SSTL_18（1.8V）	No	'
050	DDR2_DQ4	I/O	SSTL_18（1.8V）	No	'
051	DDR2_DQ5	I/O	SSTL_18（1.8V）	No	'
052	DDR2_DQ6	I/O	SSTL_18（1.8V）	No	'
053	DDR2_DQ7	I/O	SSTL_18（1.8V）	No	'
054	DDR2_DQ8	I/O	SSTL_18（1.8V）	No	'
055	DDR2_DQ9	I/O	SSTL_18（1.8V）	No	'
056	DDR2_DQ10	I/O	SSTL_18（1.8V）	No	'
057	DDR2_DQ11	I/O	SSTL_18（1.8V）	No	'
058	DDR2_DQ12	I/O	SSTL_18（1.8V）	No	'
059	DDR2_DQ13	I/O	SSTL_18（1.8V）	No	'
060	DDR2_DQ14	I/O	SSTL_18（1.8V）	No	'
061	DDR2_DQ15	I/O	SSTL_18（1.8V）	No	'
062	DDR2_DQ16	I/O	SSTL_18（1.8V）	No	'
063	DDR2_DQ17	I/O	SSTL_18（1.8V）	No	'

064	DDR2_DQ18	I/O	SSTL_18（1.8V）	No	'
065	DDR2_DQ19	I/O	SSTL_18（1.8V）	No	'
066	DDR2_DQ20	I/O	SSTL_18（1.8V）	No	'
067	DDR2_DQ21	I/O	SSTL_18（1.8V）	No	'
068	DDR2_DQ22	I/O	SSTL_18（1.8V）	No	'
069	DDR2_DQ23	I/O	SSTL_18（1.8V）	No	'
070	DDR2_DQ24	I/O	SSTL_18（1.8V）	No	'
071	DDR2_DQ25	I/O	SSTL_18（1.8V）	No	'
072	DDR2_DQ26	I/O	SSTL_18（1.8V）	No	'
073	DDR2_DQ27	I/O	SSTL_18（1.8V）	No	'
074	DDR2_DQ28	I/O	SSTL_18（1.8V）	No	'
075	DDR2_DQ29	I/O	SSTL_18（1.8V）	No	'
076	DDR2_DQ30	I/O	SSTL_18（1.8V）	No	'
077	DDR2_DQ31	I/O	SSTL_18（1.8V）	No	'
078	DDR2_DQ32	I/O	SSTL_18（1.8V）	No	'
079	DDR2_DQ33	I/O	SSTL_18（1.8V）	No	'
080	DDR2_DQ34	I/O	SSTL_18（1.8V）	No	'
081	DDR2_DQ35	I/O	SSTL_18（1.8V）	No	'
082	DDR2_DQ36	I/O	SSTL_18（1.8V）	No	'
083	DDR2_DQ37	I/O	SSTL_18（1.8V）	No	'
084	DDR2_DQ38	I/O	SSTL_18（1.8V）	No	'
085	DDR2_DQ39	I/O	SSTL_18（1.8V）	No	'
086	DDR2_DQ40	I/O	SSTL_18	No	'

			（1.8V）		
087	DDR2_DQ41	I/O	SSTL_18（1.8V）	No	'
088	DDR2_DQ42	I/O	SSTL_18（1.8V）	No	'
089	DDR2_DQ43	I/O	SSTL_18（1.8V）	No	'
090	DDR2_DQ44	I/O	SSTL_18（1.8V）	No	'
091	DDR2_DQ45	I/O	SSTL_18（1.8V）	No	'
092	DDR2_DQ46	I/O	SSTL_18（1.8V）	No	'
093	DDR2_DQ47	I/O	SSTL_18（1.8V）	No	'
094	DDR2_DQ48	I/O	SSTL_18（1.8V）	No	'
095	DDR2_DQ49	I/O	SSTL_18（1.8V）	No	'
096	DDR2_DQ50	I/O	SSTL_18（1.8V）	No	'
097	DDR2_DQ51	I/O	SSTL_18（1.8V）	No	'
098	DDR2_DQ52	I/O	SSTL_18（1.8V）	No	'
099	DDR2_DQ53	I/O	SSTL_18（1.8V）	No	'
100	DDR2_DQ54	I/O	SSTL_18（1.8V）	No	'
101	DDR2_DQ55	I/O	SSTL_18（1.8V）	No	'
102	DDR2_DQ56	I/O	SSTL_18（1.8V）	No	'
103	DDR2_DQ57	I/O	SSTL_18（1.8V）	No	'
104	DDR2_DQ58	I/O	SSTL_18（1.8V）	No	'
105	DDR2_DQ59	I/O	SSTL_18（1.8V）	No	'
106	DDR2_DQ60	I/O	SSTL_18（1.8V）	No	'
107	DDR2_DQ61	I/O	SSTL_18（1.8V）	No	'
108	DDR2_DQ62	I/O	SSTL_18（1.8V）	No	'

109	DDR2_DQ63	I/O	SSTL_18（1.8V）	No	'
110	DDR2_DQS0	I/O	SSTL_18（1.8V）	No	DDR2 Data Strobe Bus
111	DDR2_DQS1	I/O	SSTL_18（1.8V）	No	'
112	DDR2_DQS2	I/O	SSTL_18（1.8V）	No	'
113	DDR2_DQS3	I/O	SSTL_18（1.8V）	No	'
114	DDR2_DQS4	I/O	SSTL_18（1.8V）	No	'
115	DDR2_DQS5	I/O	SSTL_18（1.8V）	No	'
116	DDR2_DQS6	I/O	SSTL_18（1.8V）	No	'
117	DDR2_DQS7	I/O	SSTL_18（1.8V）	No	'
118	DDR2_DQS8	I/O	SSTL_18（1.8V）	No	For ECC
119	DDR2_DQSn0	I/O	SSTL_18（1.8V）	No	'
120	DDR2_DQSn1	I/O	SSTL_18（1.8V）	No	'
121	DDR2_DQSn2	I/O	SSTL_18（1.8V）	No	'
122	DDR2_DQSn3	I/O	SSTL_18（1.8V）	No	'
123	DDR2_DQSn4	I/O	SSTL_18（1.8V）	No	'
124	DDR2_DQSn5	I/O	SSTL_18（1.8V）	No	'
125	DDR2_DQSn6	I/O	SSTL_18（1.8V）	No	'
126	DDR2_DQSn7	I/O	SSTL_18（1.8V）	No	'
127	DDR2_DQSn8	I/O	SSTL_18（1.8V）	No	For ECC
128	DDR2_DQM0	O	SSTL_18（1.8V）	No	DDR2 Data Mask Bus
129	DDR2_DQM1	O	SSTL_18（1.8V）	No	'
130	DDR2_DQM2	O	SSTL_18（1.8V）	No	'
131	DDR2_DQM3	O	SSTL_18	No	'

			（1.8V）		
132	DDR2_DQM4	O	SSTL_18（1.8V）	No	'
133	DDR2_DQM5	O	SSTL_18（1.8V）	No	'
134	DDR2_DQM6	O	SSTL_18（1.8V）	No	'
135	DDR2_DQM7	O	SSTL_18（1.8V）	No	'
136	DDR2_DQM8	O	SSTL_18（1.8V）	No	For ECC
137	DDR2_CB0	I/O	SSTL_18（1.8V）	No	DDR2 ECC
138	DDR2_CB1	I/O	SSTL_18（1.8V）	No	'
139	DDR2_CB2	I/O	SSTL_18（1.8V）	No	'
140	DDR2_CB3	I/O	SSTL_18（1.8V）	No	'
141	DDR2_CB4	I/O	SSTL_18（1.8V）	No	'
142	DDR2_CB5	I/O	SSTL_18（1.8V）	No	'
143	DDR2_CB6	I/O	SSTL_18（1.8V）	No	'
144	DDR2_CB7	I/O	SSTL_18（1.8V）	No	'
145	DDR2_GATEO0	O	SSTL_18（1.8V）	No	DDR2 GATE for DLL
146	DDR2_GATEO1	O	SSTL_18（1.8V）	No	DDR2 GATE for DLL
147	DDR2_GATEO2	O	SSTL_18（1.8V）	No	DDR2 GATE for DLL
148	DDR2_GATEI0	I	SSTL_18（1.8V）	No	'
149	DDR2_GATEI1	I	SSTL_18（1.8V）	No	'
150	DDR2_GATEI2	I	SSTL_18（1.8V）	No	'
151	DDR2_VREF	PWR	0.9V	No	DDR2 VREF, 4 pads required
152	DDR2_VREF	PWR	0.9V	No	DDR2 VREF, 4 pads required
153	DDR2_VREF	PWR	0.9V	No	DDR2 VREF, 4 pads required

154	DDR2_VREF	PWR	0.9V	No	DDR2 VREF, 4 pads required
155	PCI_AD0	I/O	3.3V PCI	No	PCI Address and Data Bus
156	PCI_AD1	I/O	3.3V PCI	No	'
157	PCI_AD2	I/O	3.3V PCI	No	'
158	PCI_AD3	I/O	3.3V PCI	No	'
159	PCI_AD4	I/O	3.3V PCI	No	'
160	PCI_AD5	I/O	3.3V PCI	No	'
161	PCI_AD6	I/O	3.3V PCI	No	'
162	PCI_AD7	I/O	3.3V PCI	No	'
163	PCI_AD8	I/O	3.3V PCI	No	'
164	PCI_AD9	I/O	3.3V PCI	No	'
165	PCI_AD10	I/O	3.3V PCI	No	'
166	PCI_AD11	I/O	3.3V PCI	No	'
167	PCI_AD12	I/O	3.3V PCI	No	'
168	PCI_AD13	I/O	3.3V PCI	No	'
169	PCI_AD14	I/O	3.3V PCI	No	'
170	PCI_AD15	I/O	3.3V PCI	No	'
171	PCI_AD16	I/O	3.3V PCI	No	'
172	PCI_AD17	I/O	3.3V PCI	No	'
173	PCI_AD18	I/O	3.3V PCI	No	'
174	PCI_AD19	I/O	3.3V PCI	No	'
175	PCI_AD20	I/O	3.3V PCI	No	'
176	PCI_AD21	I/O	3.3V PCI	No	'
177	PCI_AD22	I/O	3.3V PCI	No	'
178	PCI_AD23	I/O	3.3V PCI	No	'
179	PCI_AD24	I/O	3.3V PCI	No	'
180	PCI_AD25	I/O	3.3V PCI	No	'
181	PCI_AD26	I/O	3.3V PCI	No	'
182	PCI_AD27	I/O	3.3V PCI	No	'
183	PCI_AD28	I/O	3.3V PCI	No	'
184	PCI_AD29	I/O	3.3V PCI	No	'
185	PCI_AD30	I/O	3.3V PCI	No	'
186	PCI_AD31	I/O	3.3V PCI	No	'
187	PCI_CBEn0	I/O	3.3V PCI	No	PCI Command/Byte Enable
188	PCI_CBEn1	I/O	3.3V PCI	No	'
189	PCI_CBEn2	I/O	3.3V PCI	No	'
190	PCI_CBEn3	I/O	3.3V PCI	No	'
191	PCI_IRQnA	I	3.3V PCI	No	PCI Interrupt Request
192	PCI_IRQnB	I	3.3V PCI	No	'
193	PCI_IRQnC	I	3.3V PCI	No	'

194	PCI_IRQnD	I	3.3V PCI	No	'
195	PCI_REQn0	I/O	3.3V PCI	No	PCI Bus Request
196	PCI_REQn1	I	3.3V PCI	No	'
197	PCI_REQn2	I	3.3V PCI	No	'
198	PCI_REQn3	I	3.3V PCI	No	'
199	PCI_REQn4	I	3.3V PCI	No	'
200	PCI_REQn5	I	3.3V PCI	No	'
201	PCI_REQn6	I	3.3V PCI	No	'
202	PCI_GNTn0	I/O	3.3V PCI	No	PCI Bus Grant
203	PCI_GNTn1	O	3.3V PCI	No	'
204	PCI_GNTn2	O	3.3V PCI	No	'
205	PCI_GNTn3	O	3.3V PCI	No	'
206	PCI_GNTn4	O	3.3V PCI	No	'
207	PCI_GNTn5	O	3.3V PCI	No	'
208	PCI_GNTn6	O	3.3V PCI	No	'
209	PCI_IDSEL	I	3.3V PCI	No	PCI Initialization Device Select
210	PCI_RESETn	I/O	3.3V PCI	No	PCI Reset
211	PCI_FRAMEn	I/O	3.3V PCI	No	PCI Cycle Frame
212	PCI_IRDYn	I/O	3.3V PCI	No	PCI Initiator Ready
213	PCI_TRDYn	I/O	3.3V PCI	No	PCI Target Ready
214	PCI_DEVSELn	I/O	3.3V PCI	No	PCI Device Select
215	PCI_STOPn	I/O	3.3V PCI	No	PCI Stop
216	PCI_PAR	I/O	3.3V PCI	No	PCI Parity
217	PCI_PERR	I/O	3.3V PCI	No	PCI Data Parity Error
218	PCI_SERR	I/O	3.3V PCI	No	PCI System Error
219	PCI_CLK	I	3.3V PCI	No	PCI Clock
220	PCI_CONFIG0	I	3.3V PCI	No	PCI Config
221	PCI_CONFIG1	I	3.3V PCI	No	PCI Config
222	PCI_CONFIG2	I	3.3V PCI	No	PCI Config
223	PCI_CONFIG3	I	3.3V PCI	No	PCI Config
224	PCI_CONFIG4	I	3.3V PCI	No	PCI Config
225	PCI_CONFIG5	I	3.3V PCI	No	PCI Config
226	PCI_CONFIG6	I	3.3V PCI	No	PCI Config
227	PCI_CONFIG7	I	3.3V PCI	No	PCI Config
228	LIO_AD0	I/O	3.3V	No	Local IO Address and Data Bus
229	LIO_AD1	I/O	3.3V	No	'
230	LIO_AD2	I/O	3.3V	No	'
231	LIO_AD3	I/O	3.3V	No	'
232	LIO_AD4	I/O	3.3V	No	'

233	LIO_AD5	I/O	3.3V	No	'
234	LIO_AD6	I/O	3.3V	No	'
235	LIO_AD7	I/O	3.3V	No	'
236	LIO_AD8	I/O	3.3V	No	'
237	LIO_AD9	I/O	3.3V	No	'
238	LIO_AD10	I/O	3.3V	No	'
239	LIO_AD11	I/O	3.3V	No	'
240	LIO_AD12	I/O	3.3V	No	'
241	LIO_AD13	I/O	3.3V	No	'
242	LIO_AD14	I/O	3.3V	No	'
243	LIO_AD15	I/O	3.3V	No	'
244	LIO_CSn	O	3.3V	No	LOcal IO Chip Select
245	LIO_ROMCSn	O	3.3V	No	Local IO Rom chip select
246	LIO_WRn	O	3.3V	No	Local IO Write Enable
247	LIO_RDn	O	3.3V	No	Local IO Read Enable
248	LIO_ADLOCK	O	3.3V	No	Local IO Address Lock
249	LIO_DIR	O	3.3V	No	Local IO DIR
250	LIO_DEN	O	3.3V	No	Local IO Device Enable
251	LIO_A0	O	3.3V	No	Local IO Address Bus
252	LIO_A1	O	3.3V	No	'
253	LIO_A2	O	3.3V	No	'
254	LIO_A3	O	3.3V	No	'
255	LIO_A4	O	3.3V	No	'
256	LIO_A5	O	3.3V	No	'
257	LIO_A6	O	3.3V	No	'
258	LIO_A7	O	3.3V	No	'
259	SYSRESET	I	3.3V	Iddqhold	System Reset
260	SYSCLK	I	3.3V	No	System Clock
261	MEMCLK	I	3.3V	No	DDR2 Option Clock
262	GPIO0	I/O	3.3V	No	GPIO
263	GPIO1	I/O	3.3V	No	'
264	GPIO2	I/O	3.3V	No	'
265	GPIO3	I/O	3.3V	No	'
266	INTn0	I	3.3V	Iddqhold	Interrupt
267	INTn1	I	3.3V	Iddqhold	'
268	INTn2	I	3.3V	Iddqhold	'
269	INTn3	I	3.3V	Iddqhold	'
270	NMIn	I	3.3V	Iddqhold	'
271	CLKSEL0	I	3.3V	Iddqhold	Core Clock Select
272	CLKSEL1	I	3.3V	Iddqhold	'
273	CLKSEL2	I	3.3V	Iddqhold	'

274	CLKSEL3	I	3.3V	Iddqhold	'
275	CLKSEL4	I	3.3V	Iddqhold	'
276	CLKSEL5	I	3.3V	Iddqhold	Memory Clock Select
277	CLKSEL6	I	3.3V	Iddqhold	'
278	CLKSEL7	I	3.3V	Iddqhold	'
279	CLKSEL8	I	3.3V	Iddqhold	'
280	CLKSEL9	I	3.3V	Iddqhold	'
281	TEST_CTRL0	I	3.3V	Pull-up	Test Control
282	TEST_CTRL1	I	3.3V	Pull-up	Test Control
283	TEST_CTRL2	I	3.3V	Pull-up	Test Control
284	TEST_CTRL3	I	3.3V	Pull-up	Test Control
285	TEST_CTRL4	I	3.3V	Pull-up	Test Control
286	TEST_CTRL5	I	3.3V	Pull-up	Test Control
287	TEST_CTRL6	I	3.3V	Pull-up	Test Control
288	TEST_CTRL7	I	3.3V	Pull-up	Test Control
289	TESTCLK	I	3.3V	No	Test clock
290	PLLCLOCK0	O	3.3V	No	PLL0 clock out for test
291	PLLCLOCK1	O	3.3V	No	PLL1 clock out for test
292	TRST	I	3.3V	Pull-up	JTAG signal
293	TMS	I	3.3V	Pull-up	JTAG signal
294	TDI	I	3.3V	Pull-up	JTAG signal
295	TCK	I	3.3V	No	JTAG signal
296	TDO	O	3.3V	No	JTAG signal
Note：Iddqhold menas no pull-up during a leakage test but pull-up in other casese.					

4.2 Alphabetical Signals Reference

5. Memory Map

批注 **[S12]:** Need input from ICT

6. Power ON Sequence (Boot)

批注 **[S13]:** Need inputs from ICT

7. Electrical characteristics

批注 **[S14]:** To be updated

8. Thermal Specifications and Design considerations

批注 [S15]: To be updated. We need to decide wich Power consumption value to publish (for estimated value perhaps it is better just to wait silicon results for 2F

For 2E, we need to have the silison measurements

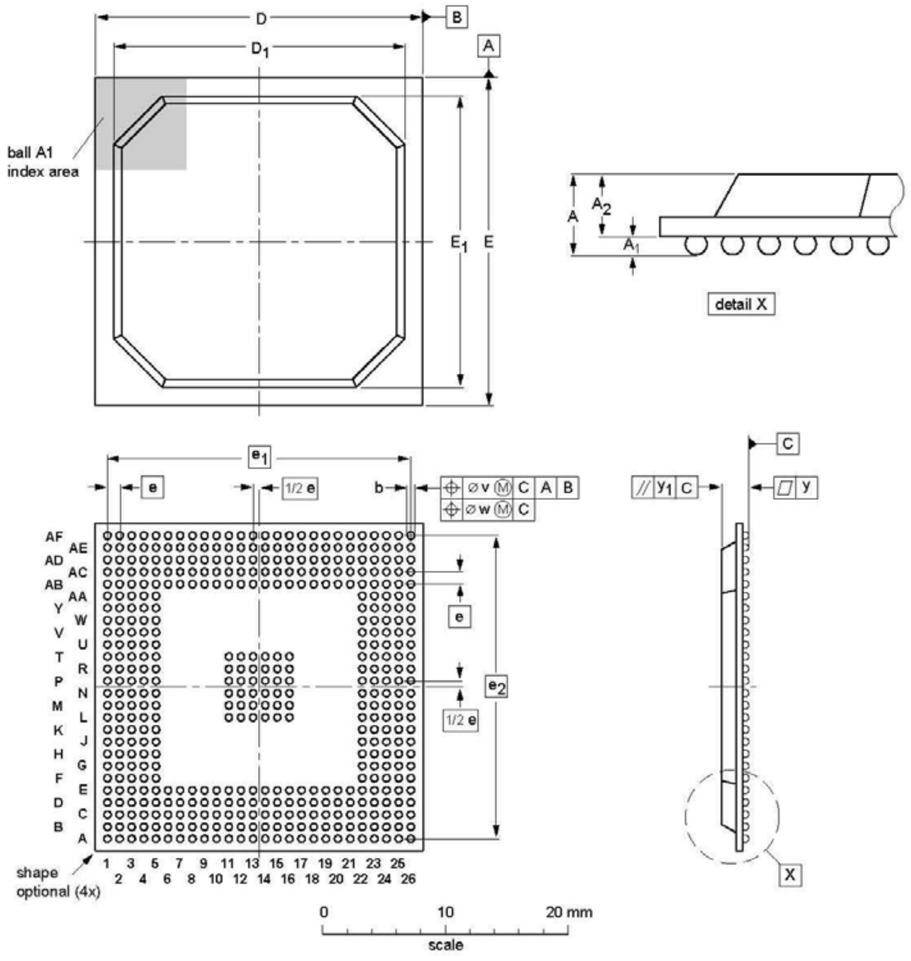
9.1 Desktop 100% CPU utilization

Unit:W	BC_1V26_m40C 1.3GHz	TC_1V2_25C 1GHz	TC_1V26_125C 1GHz	TC_1V0_25C	WC_1V08_125C 600MHz	WC_0V9_125C
Dyn.	5.171	3.559	4.6267		2.019	
Leak.	0.9295	0.6313	1.7937		0.8501	
Total	6.104	4.190	6.4204		2.8691	

9. Package Mechanical Specifications

批注 [S16]: To be updated by david Kaire

10.1 Package Mechanical Drawing

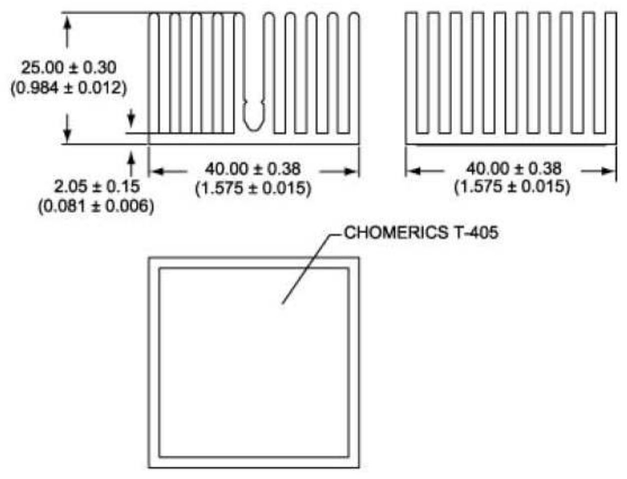


DIMENSIONS (mm are the original dimensions)

UNIT	A max.	A_1	A_2	b	D	D_1	E	E_1	e	e_1	e_2	v	w	y	y_1
mm	2.45	0.6 0.4	1.85 1.60	0.7 0.5	27.2 26.8	24.75 23.75	27.2 26.8	24.75 23.75	1	25	25	0.3	0.15	0.2	0.35

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	JEITA			
SOT795-1	144E	MS-034	- - -			02-11-18 05-04-26

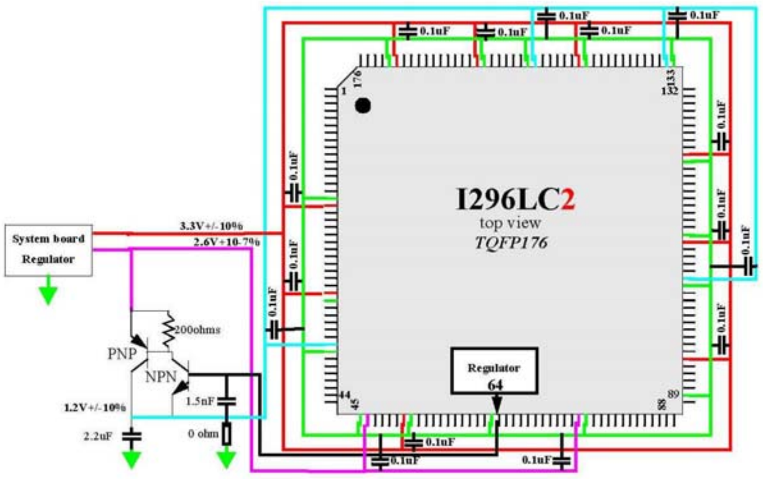
10.2 Heat Sink mechanical Drawing


25.00 ± 0.30
(0.984 ± 0.012)
2.05 ± 0.15
(0.081 ± 0.006)
40.00 ± 0.38
(1.575 ± 0.015)
40.00 ± 0.38
(1.575 ± 0.015)
CHOMERICS T-405

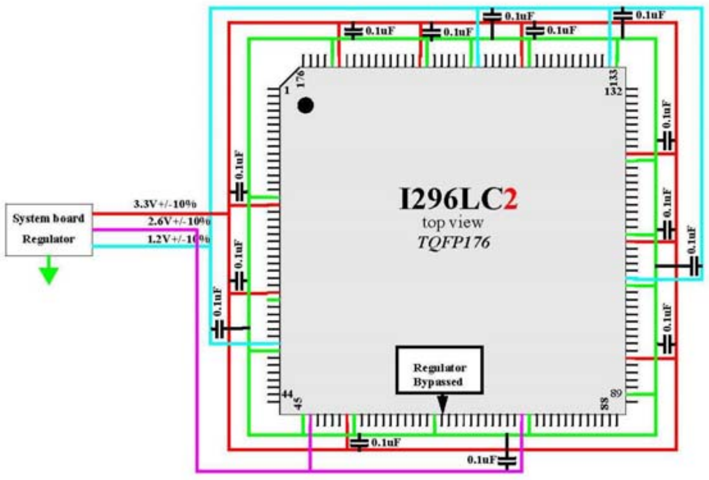
10. Application Environment

- Proposed options to supply Core suppply:

OPTION 1: Internal Regulator On, External Bipolar used



OPTION 2: Internal Regulator Bypassed, Core supply 1.2V directly provided on leads 36,106,134,152 from On-board regulators.



批注 [S17]: We need to show the application board proposal, with all external components: voltage regulator, decoupling capacitance …DDR, ….etc …. Compensation cell in accurate mode …

批注 [S18]: This is just an example , for power core supply