RC32355 Design Guide

Notes

This document is intended to reflect some of the design considerations that need to be applied when designing a system based on the RC32355. This document is intended to document other, subtle behaviors of the RC32355 than the ones listed in the errata sheet for the device. These issues should be considered early in the design process to avoid lengthy debug time.

Document Revision History

April 18, 2001: Initial publication.

July 3, 2001: Added further explanation in TDM Data Transfer section.

July 9, 2001: Added Disabling DMA Channels section.

August 13, 2001: Added Connecting 8-bit, 16-bit, and 32-bit Devices to the RC32355 Memory Controller section.

October 8, 2001: Added DP Package for RC32355 section.

June 6, 2003: Added Ethernet Controller Software Reset section.

November 17, 2003: Added UART Mode section.

DPTR and NDPTR Registers

The following procedure is suggested to maximize the efficiency of DMA processing.

1. Write the first descriptor chain to either the DPTR or the NDPTR register. However, if you intend to write to DPTR first, it is recommended that you first clear the NDPTR register (NDPTR = = 0). Data sent to the NDPTR register will automatically load to DPTR when DTPR has finished processing.

2. While the first chain of descriptors is being processed, a new chain is formed in memory. Write subsequent descriptor chains to the NDPTR register.

3. When the current chain in DPTR completes, the DMA will interrupt the CPU (Interrupt on Done, Finished, etc. as configured in the DMAxSM and IPEND registers), causing the chain residing in NDPTR to load to DPTR. When the interrupt occurs, write the chain being formed in memory to NDPTR. In this manner, while one chain is being processed in the DPTR register, there is always one chain residing in NDPTR and one chain being formed in memory.

RC32355 SDRAM Design Considerations

The RC32355 supports a maximum system speed of 75 MHz in the PQFP-208 package.

For the SDRAM to operate at maximum efficiency for these configurations, the board designers using the RC32355 must keep the following within certain limits:

- *SDRAM board architecture*
- *signal routing*
- *system loading.*

SDRAM Board Architecture

For maximum SDRAM speed, IDT recommends that all SDRAM control and data signals be directly connected to the RC32355. All other memory devices should be placed behind buffers. The SYSCLKP pin should be used to drive only the clocks on the SDRAM chips. It should not be used to drive any other device on the board.

The SDCLKINP signal should be tapped off from the SYSCLKP at the SDRAM. The length of the resulting SDCLKINP trace should match the SDRAM data trace length as much as possible. In no case should the length of the SDCLKINP trace exceed the length of the SDRAM data traces.

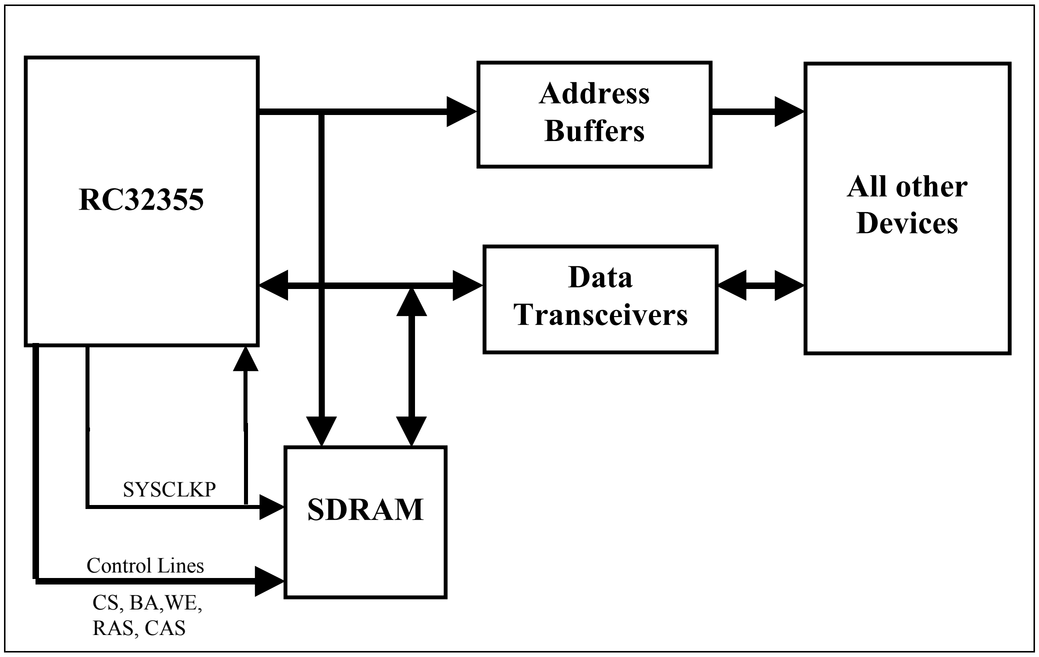


Figure 1

Signal Routing

To provide the most margin for SDRAM hold times:

- Keep all signals going to the SDRAM, address, data, and control signals as short as possible
- Match trace lengths as close as possible
- Minimize as much as possible the trace length for SYSCLKP to the SDRAM (this is very important)
- Make the SYSCLKP trace the shortest SDRAM trace on the board
- Match the length of the SDCLKINP to the SDRAM data lines.

Although this optimum configuration may not be achievable, the further that the layout deviates from it, the less margin there will be for SDRAM hold times.

System Loading

To support maximum speeds, reasonable SDRAM loading constraints must be followed.

For 75MHz operation, IDT recommends that no more than eight discrete SDRAM chips be connected. SODIMMs with up to eight chips may also be used. IDT does not recommend DIMMs at this speed. However, if customers wish to use DIMMs, they should limit themselves to some combination with no more than eight loads. If two DIMMs are used, then each DIMM should have no more than four chips on it. In addition, any unused clock lines should be left unconnected; otherwise, the loading on SYSCLKP will exceed the drive capability of the RC32355 SYSCLKP pin.

It is possible to use more than eight SDRAM chips or very large DIMMs. However, if the loading exceeds that recommended above, a customer should buffer all of the control, address, data, and clock lines accordingly. Because of the extra delay associated with this, it will not be possible to achieve higher system speeds.

Board Reset from EJTAG Probe

The EJTAG specification requires the RST* pin on the EJTAG header to reset the board. Many ICE probes use the RST* pin (which is open drain output) to sense whether power is applied to the probe. When the EJTAG probe initiates a board reset by bringing RST* low, the signal is driven low solidly.

However, at the end of the reset procedure the EJTAG probe cannot drive the RST* signal back high because this pin is an open drain output. The EJTAG probe must rely on a pull-up resistor supplied by the board.

Even with a resistor as small as 1000 ohms, it takes a full 100ns to pull this signal high even when it is only driving one load. If that single load is an EPLD, the signal tends to oscillate a bit as it crosses the Vih threshold for the EPLD. This causes the EPLD to register the deassertion and reassertion of the RST* signal multiple times. Because all of the IDT parts require continuous reset pulses in the order of 100ms or greater, this oscillation causes the part to fail reset in an unpredictable way.

Listed below are two methods to avoid this problem.

1. The reset signal from the ICE probe can be routed through the power management IC. Generally, these ICs contain an input which initiates a reset of the circuit. The pulse width of this reset depends on the resistor/capacitor combination attached to the IC.

2. If the signal goes into an EPLD, the assertion of RST* can be used to initiate a cold reset of the processor and board circuit. But when RST* deasserts, a counter with a count length of greater than two microseconds should be used before deasserting the cold reset to the processor and board.

The IDT79S355 evaluation board uses the second method.

Operation of UART in Polled Transmit Mode

The RC32355 provides 2 UARTs which are designed to be compatible with both the 16450 and the 16550. The 16550 is identical to the 16450 except that the 16550 provides a 16 byte FIFO on both receive and transmit sides.

FIFOs for the 16550 are enabled by setting bit 0 of the Buffer Control Register, BCR[0].

There are two modes in which users can program the UARTs:

- *Interrupt driven mode, when it is supported by the UART Controller and the board design*
- *Polled mode.*

The RC32334 UARTs behave in compliance with the 16550 specifications in the interrupt driven mode.

FIFOs were introduced in the 16550 to enable the 16 bytes of FIFOs to be filled (transmit) or emptied (receive) in a single execution of the interrupt handler, thereby reducing the load on the CPU. However, these FIFOs can be used in polled mode, although the benefit is less compared to the interrupt driven mode.

Typically, in the polled transmit mode, the Line Status Register (LSR) checks to see if it is appropriate for the software to write the next byte to the UART or the FIFO. The 16550 specifications state that LSR[5] can be guaranteed to be 1 when the Transmit Holding Register (THR) is empty, and LSR[5] can be guaranteed to be a 0 when THR is not empty. A non-empty THR implies at least one byte in the FIFO buffer. Therefore, writing a single byte to the transmit FIFO ought to result in LSR[5] returning a 0. This does not happen on the RC32334.

IDT recommends two possible procedures for ensuring the current operation of the RC32355 when it is used in polled transmit mode:

- Test LSR[6] bit instead of LSR[5] bit. LSR[6] bit, when set to 1 by the controller, indicates to the user that the transmit buffer as well as the THR is empty. This will allow the user to transmit one byte at a time in the polled mode.
- Set the DMA mode in the Buffer Control Register, BCR[3], to 1. This will activate the TXRDY interrupt signal when the transmit FIFO buffer is completely empty and will deactivate the TXRDY signal when the buffer is completely full. The state of the TXRDY signal can be probed by software through the Expansion Interrupt Controller. Register Group 5 deals with UART channel 0 and Group 6 deals with UART channel 1. TXRDY Interrupt State can be read through the “Interrupt Pending Register” corresponding to the UART channel under consideration. Once a completely empty buffer condition is sensed by polling the Interrupt Pending Register, up to 16 bytes can be written to the transmit FIFO in a single attempt without worrying about the level of fullness of the buffer, etc.

Synchronization Problems for Chip Modules with Multiple Clock Domains

There is a potential synchronization / timing issue when resetting various on-chip modules using the EN bit in each module (e.g. Ethernet, TDM, and USB) that have multiple clock domains within the RC32355. For example, the IPBus^{TM} portion of the Ethernet module runs at the system clock frequency (say 75 MHz), while the MAC side of the Ethernet module runs at the MII clock rates (2.5 or 25 MHz). When the EN bit in the Ethernet module register has been cleared to reset the module, one expects the module to be reset immediately (within the next system-clock cycle) and then can re-enable the module by setting the EN bit on the next instruction. However, there is a possibility that the MAC portion of the module did not reset because the internal EN bit did not have enough time to cross clock domains and get sampled at the rising edge of the slower MII clock. As a result, the Ethernet module may continually request DMA transfers and write unspecified data to memory. This result also applies to the TDM and USB interfaces.

Follow these steps to ensure the modules are reset:

1. Clear the EN bit
2. Wait for a period of time (several interface clock cycles) to make sure the EN bit was sampled by the other clock domain and that the module has been completely reset
3. Re-enable the module by setting the EN bit.

CPU Interrupts

When the RISCore 32300 CPU core processes an interrupt, there are two code components: a generic interrupt handler function, and a specific ISR (Interrupt Service Routine). The generic handler must do the following:

1. Save the current state of the CPU (all registers)
2. Determine the pending interrupt with the highest priority
3. Call the appropriate ISR (Somewhere in here, the interrupt source should be cleared)
4. [Optional] return to step 2 if other interrupts are still pending
5. Restore the state of the CPU (all registers)
6. Execute ERET instruction to return to the original PC location.

In the ISR, the code should clear the source of the interrupt. For example, if you are processing a DMA interrupt for TDM Input, the ISR should clear the DMA11S register bits before exiting the ISR. If this is not done, we will get the same interrupt as soon as we are done (i.e. go from step 6 directly back to step 1). A problem occurs in that even though the source of the interrupt in the ISR is cleared, the system still gives an additional interrupt after completing step 6. When handling this interrupt (in step 2), we cannot find ANY interrupts pending. As a result, we display an error an exit.

The system gives an additional interrupt because the register write to clear the source of the interrupt gets delayed going from the YY Bus to the IP Bus. The reason for the delay is that an external master has been granted the bus, so the IP Bus cannot accept the transaction from the BIU until the external arbiter has completed. Although the delay here occurred due to the external arbiter, it could just as easily have occurred if the DMA had been granted the bus. In either case, during this delay the CPU continues onto steps 5 and 6. The instructions AND the data for these steps are already in the cache. As a result, the CPU is able to complete these steps very quickly with no need to perform a transaction over the YY Bus (Remember, the YY Bus is tied up now with our pending register write transaction that cannot complete until the external arbiter has released the bus). Immediately after completing step 6, the CPU branches to the interrupt vector and begins step 1. By the time it gets to step 2, the write transaction has completed, and the interrupt source has been cleared. Since no pending interrupts are active, an error is displayed.

To avoid this problem, make sure that the transaction in the ISR that clears the interrupt source completes before exiting the ISR. Since the RC32355 can only queue up one external write at a time, one could simply perform the write twice in a row before exiting the ISR. If the first one is delayed (due to the external arbiter or DMA being granted the bus), the CPU will have to stall on the second write.

TDM Data Transfer

Using the RISCore 32300 CPU to read and write TDM data is highly inefficient and will only work with certain system configurations and is not really recommended. Even if it does work, the RISCore 32300 CPU may not be able to keep the TDM continuously supplied with data when active time slots occur. OVR and UND must be closely monitored for error conditions.The DMA Controller must be used to receive and transmit data on the TDM port. The DMA Controller is recommended because it is a much more efficient way to read and write data from the TDM interface and will not cause any over-run or under-run errors to occur.

In general, it is recommended to use the DMA Controller to transfer and receive data on all 4 communications modules (ATM, TDM, Ethernet, and USB).

The following is a special consideration when transmitting data on the TDM port. When transferring data on the TDM port and using the DMA Controller, it is very possible to find discrepancies in the number of “Tx Done” events reported in the DMA channel and the expected number of “Tx Done” events based on the number of bytes transferred.

The reason for this discrepancy is as follows: The TDM interface has a 32 byte transmit FIFO, and when the transmit DMA is started, it runs as fast as the bus will allow until the FIFO is full. As an example, for the one byte burst size, 3 DMA accesses are needed for each byte, a descriptor read, a data read, and a descriptor update. This translates into 96 DMA cycles in the first 46 μsec after starting the transmit DMA. This uses up about 50% of the bus bandwidth. This means that there should be 32 done events in the first 46 μsec or one every 1.4 μsec. This is too fast for the software to keep up with, and so many of the first 32 done events are missed. After the FIFO is full, it waits for the TDM bus to transmit a byte before fetching another one. This slows down the DMA accesses and “Tx Done” events to a rate that the software can keep up with. The discrepancy in the number of “Tx Done” events reported in the DMA Controller depends on the burst size used (1, 4, 8, or 16 bytes).

Disabling DMA Channels

The RC32355 enables the users to disable the DMA channels at will > depending on the need of the applications. Any of the DMA channels can be disabled by writing a zero into DMAxC[RUN] bit. Usually it is recommended to disable the DMA Channel when there is no more activity on the interface the DMA channel is servicing. However, special sequence must be used when disabling certain DMA channels (see below) and their corresponding peripherals.

Ethernet Output DMA (DMA Channel 10),

USB Output DMA (DMA channel 14),

ATM VC DMA (DMA Channels 1-8),

Specifically, when disabling the DMA channels (and their corresponding peripherals) listed above, the following sequence must be used:

1. Disable the DMA channel first
2. Wait for the DMAxS[H] bit to get set
3. Then disable the on-chip peripheral.

This limitation doesn't apply for the other DMA channels.

For example, when an Ethernet overrun condition occurs, the following sequence must be used:

1. Disable the Ethernet output DMA channels first
2. Wait for both DMAxS[H] bits to set
3. Then disable the Ethernet interface.

Note: The Ethernet Input DMA channel can be disabled any time, i.e., before or after disabling the Ethernet interface.

Re-enable both the DMA channels and Ethernet interface to recover from the overrun condition.

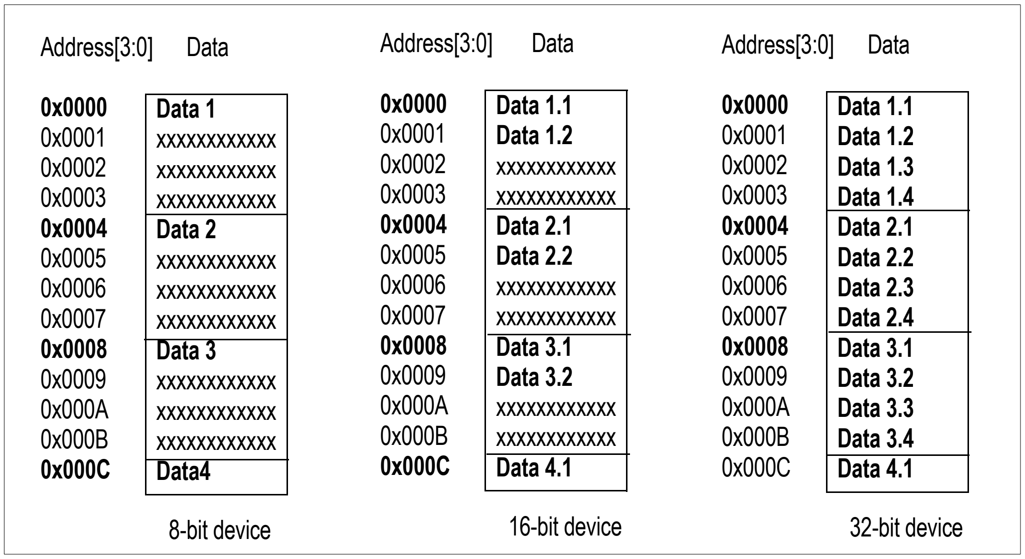
Connecting 8-bit, 16-bit, and 32-bit Devices to the RC32355 Memory Controller

When accessing 8-bit or 16-bit devices, the device controller **always reads a word** (32 bits) from the device regardless of the device size setting specified in the device control register. For example, if the RISCore 32300 CPU core reads one byte from an 8-bit device, the device controller will perform four single byte reads. This can cause problems in certain applications.

The width of a device—8-bit, 16-bit, or 32-bit—is configured in the device size field of the device control register. 8-bit devices connect to *memadd[25:0]*, 16-bit devices connect to *memadd[25:1]*, and 32-bit devices connect to *memadd[25:2]*. During write transactions to 16-bit and 32-bit devices, the byte write enable (BWEN[3:0]) signals are used to select the number of bytes to be written.

This tendency to read multiple bytes during byte accesses might conflict with devices which contain 8- or 16-bit flag registers. When theses types of flag registers are inadvertently read by the four byte access created by the attempt to read only one location, it modifies the contents of the registers. These types of interfaces are often referred to as CTRL-E, IDMA, or “mailbox” interfaces. One example of a CTRL-E implementation that is affected by this problem is the Alcatel ADSL MTK20150 chipset (see Application Note AN-304) which is included in the 79RP355 Reference Platform. This interface is seen by the device controller as an 8-bit device, and it contains several 8-bit flag registers used for operations on the ADSL line. If these registers are mapped as 8-bit registers, then accessing any register located within the same word of address space (0-3,4-7,8-B,C-F) will cause the contents of these registers to be corrupted.

In order to prevent register corruption, 8-bit/16-bit devices must be connected to the device controller as 32-bit devices (using *memadd[25:2]*), and the associated software must access the devices as 32-bit devices. The code should then use the “cast” command to convert the word to byte or byte back to word for reads and writes respectively.



Address[3:0]	Data	Address[3:0]	Data	Address[3:0]	Data
0x0000	**Data 1**	**0x0000**	**Data 1.1**	**0x0000**	**Data 1.1**
0x0001	xxxxxxxxxxxx	0x0001	**Data 1.2**	0x0001	**Data 1.2**
0x0002	xxxxxxxxxxxx	0x0002	xxxxxxxxxxxx	0x0002	**Data 1.3**
0x0003	xxxxxxxxxxxx	0x0003	xxxxxxxxxxxx	0x0003	**Data 1.4**
0x0004	**Data 2**	**0x0004**	**Data 2.1**	**0x0004**	**Data 2.1**
0x0005	xxxxxxxxxxxx	0x0005	**Data 2.2**	0x0005	**Data 2.2**
0x0006	xxxxxxxxxxxx	0x0006	xxxxxxxxxxxx	0x0006	**Data 2.3**
0x0007	xxxxxxxxxxxx	0x0007	xxxxxxxxxxxx	0x0007	**Data 2.4**
0x0008	**Data 3**	**0x0008**	**Data 3.1**	**0x0008**	**Data 3.1**
0x0009	xxxxxxxxxxxx	0x0009	**Data 3.2**	0x0009	**Data 3.2**
0x000A	xxxxxxxxxxxx	0x000A	xxxxxxxxxxxx	0x000A	**Data 3.3**
0x000B	xxxxxxxxxxxx	0x000B	xxxxxxxxxxxx	0x000B	**Data 3.4**
0x000C	**Data4**	**0x000C**	**Data 4.1**	0x000C	**Data 4.1**
	8-bit device		16-bit device		32-bit device

Figure 1 8-bit, 16-bit, and 32-bit Devices Mapped as x32

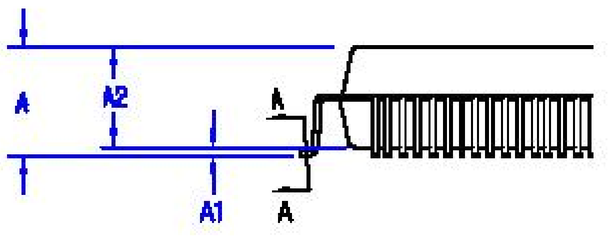
Figure 1 gives a pictorial example of how data are physically stored (byte mapping). The data in bold type refers to relevant data while the addresses in bold type account for word-based mapping (the addresses really accessed by the Device Controller if it is connected as a 32-bit device).

DP Package for RC32355

IDT recently modified its Plastic Quad Flat Pack 208 pins (PQFP208) for the RC32355, switching from DP to DH packaging. This change occurred because of mechanical weaknesses observed in the DP part.

The DP package incorporates a built-in copper heatslug to facilitate heat transfer. The heatslug weighs 9.7 grams. This weight, under heavy vibration tests, puts an abnormal strain on the solder joint, resulting in possible failure. To prevent failure on the RC32355 DP package, IDT recommends mounting the board with an adhesive attachment under the body to relieve any stress on the joint.

The figure below displays a vertical view of the DP package outline. A1 accounts for the void, approximately 0.3mm, between the package bottom and the lead bottom. This void should be filled with either glue or other adhesive attachment to compensate for the extra strain.



RC32355 DP Package Outline

Because the DH package fixes the abnormal strain problem, this recommendation applies **only** to RC32355 DP packages. Nevertheless, all evaluation boards need to be handled carefully to avoid potential damage.

Ethernet Controller Software Reset

The RC32355 Ethernet controller can be reset manually by clearing the EN bit in the corresponding ETHxINTFC register. When the EN bit is cleared, an Ethernet interface reset is generated and the RIP bit is set to indicate that an Ethernet reset is in progress. The reset may take several clock cycles to complete due to the crossing of multiple clock domains. When the reset has completed, the RIP is cleared and the Ethernet interface may be re-enabled by setting the EN bit.

However, if the MII clocks disappear or are not present during the Ethernet interface reset, the RIP bit will remain set, preventing the Ethernet interface from being reset until an MII clock is provided. The only other way to clear the RIP bit is to generate a cold_reset. The Ethernet PHY normally drives the MII clocks. Consequently, the behavior described above can be avoided by ensuring that any software which resets the RC32355 Ethernet controller does not reset the Ethernet PHY at the same time.

UART Mode

Do not change between UART 16450 and 16550 modes (bit [0] of the FIFO control register) and then flush the FIFOs while there are characters still in the FIFO TX buffer waiting to be sent. This may result in a UART transmit status malfunction. If this happens, the TE and THR bits in the UARTxLS register may be permanently cleared until the interface is reset. Characters can otherwise still be transmitted and the other status signals will continue to function.