1. Introduction

Digital Signal Processors (DSPs) are commonly used in the Telecommunications market for applications such as modems, voice and data compression, and Echo Cancellation. A DSP is commonly used with a communication controller. An incoming stream of voice is processed by the DSP, and the data is forwarded to a host CPU (communication controller), which routes the data stream to another voice channel, LAN, or remote target over the WAN cloud.

This type of application requires a high performance CPU, which has sufficient power for both processing and distribution of data. The GT-64240/260 is a high performance System Controller, which provides the CPU with DRAM, PCI, Ethernet, WAN and Device bus ports. Thus it provides an excellent solution for routing data received on the DSP interface to other channels such as Telecommunications (T1, Frame Relay, etc.), Ethernet 10/100 ports and other communication controllers interfaced over the PCI of the GT-64240/260.

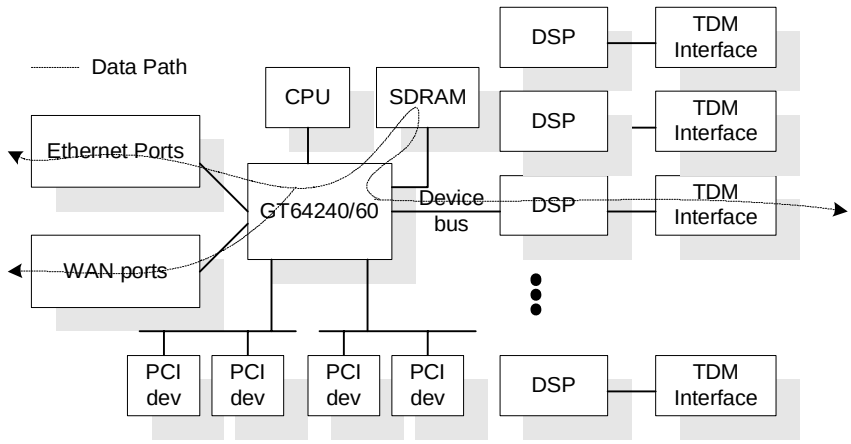
This Application Note focuses on the interface between the GT-64240/260 System Controller and DSPs. Specifically it focuses on the interface to Texas Instruments DSPs. For further information, refer to the GT-64240/260 datasheet.

1.1 Basic Configuration

There are many ways to connect DSPs to the GT-64240/260, and there are many possible data flow routes. In this document we assume a specific hardware configuration and data flow.

Figure 1 shows a basic system with a GT-64240/260 and Texas Instruments DSPs.

Figure 1: Basic System with GT-64240/260 and Texas Instruments DSPs



The Data flow is as follows:

1. Data is received on the TDM channel on the DSP.
2. The DSP processes the data, and interrupts the host CPU (or the CPU polls the DSP).
3. The CPU transfers the data to its local memory (SDRAM), processes the data and makes the routing decisions either to forward to another port on its own device or to forward to another port on another DSP.

Data can also flow from the local ports on the host CPU to the DSPs. In this case there is no interrupt from the DSP.

1.2 Interface Buses

1.2.1 DSP Interface Bus

Two buses can be used to interface with Texas Instrument DSPs—the Expansion bus and the Host Port Interface (HPI) bus. Some DSPs have an Expansion bus, while others have an HPI bus. The HPI bus can be considered as a derivative of the Expansion bus. (An HPI bus is an Expansion bus with a lower bit count in Asynchronous mode.) Therefore in this document we focus on the Expansion bus in Slave mode, assuming that all transactions are initiated by the host. The DSP may interrupt the host for a service request, but it is the host that initiates transactions on the bus.

1.2.2 GT-64240/260 Interface Bus

The Device bus is a 32 bit multiplexed bus that functions at a speed of up to 100 MHz. The Device bus functions as the Interface bus on the GT-64240/260.

The DSP is mapped on the GT-64240/260 Device bus. The DSP memory is accessed via two registers—an Address register and a Data register. Both are mapped on the GT-64240/260 Device bus.

To access any register/memory in the DSP address space, perform the following two transactions:

1. **Write to the DSP XBISA register**. This sets the address to be accessed and the type of access—incremental or hold. On incremental access, the address field in the DSP XBISA register is incremented for every data transfer.
2. **Write to/ Read from the DSP XBD register.** This performs the actual data transfer. If the AINC bit on the DSP XBISA register is set, the address that is accessed will be incremented every time the DSP XBD register is accessed.

These are considered as separate transactions. Normally the host CPU performs the first Write to the DSP XBISA register, while the GT-64240/260's DMAs perform the data transfer to the DSP XBD register.

1.3 Design Considerations

1.3.1 Host CPU Bandwidth Requirements

One of our basic assumptions is that data is transferred by a DMA on the GT-64240/260, otherwise the CPU would have to spend a large percentage of its bandwidth on data transfer. The host CPU is left mainly with the task of initiating the DMAs and handling the interrupts for service requests, which are received from the DSP.

Assuming that the DSP mainly handles voice applications, the data rate of any incoming voice channel is 64 Kb per second (and double that for simultaneous Receive and Transmit). To reduce overhead, data is transferred to and from the host CPU in cells. For this discussion we assume a cell of 20 µsec (which is equivalent to 160 bytes). We also assume that the data is forwarded to the host CPU without compression (worst case).

When the DSP has finished processing a cell (160 bytes), the host CPU must transfer the data to its own memory. This can be done by interrupting the CPU for each cell of data, or the host CPU can poll the DSPs and query them for service requests.

If Interrupts are used, the number of interrupts per second is:

*1/cell time * Number of DSPs * Number of Voice Channels per DSP*

For example:

Number of DSPs = 16

Number of Voice Channels per DSP = 12

Number of Interrupts: ~10000 interrupts per second.

Software that is running under VxWorks may spend up to 8 µsec overhead on each Interrupt. In the above example the CPU spends 8% of the CPU bandwidth on overhead.

The above example was simulated for a MIPS R7000 CPU, running at 200 MHz (100 MHz on the CPU bus). The 8 µsec is only the overhead. It includes the response time of the operating system, and the time needed by the Interrupt handler to verify the interrupt cause. It does not include the time necessary to activate the DMAs.

If the number of DSPs or the number of channels is doubled, the overhead reaches 16% or even 32% (without any data processing).

1.3.2 Data Transfer Rate

The data transfer rate is:

Number of Voice channels per DSP x Number of DSPs x 64Kb x 2 (both ways)

Using the same example as in Section 1.3.1, the data transfer rate is about 3 MB per second. Doubling the number of DSPs or the number of voice channels per DSP increases the data transfer rate to 6 MB, or even 12 MB per second.

1.3.3 Multiple DSPs on the GT-64240/260 Device Bus

Normally one host CPU should serve multiple DSPs. Common configurations are 8,16, 32 or 64 DSPs per host CPU.

As part of the hardware bus protocol, the electrical problem of driving so many devices on the same Device bus must be solved.

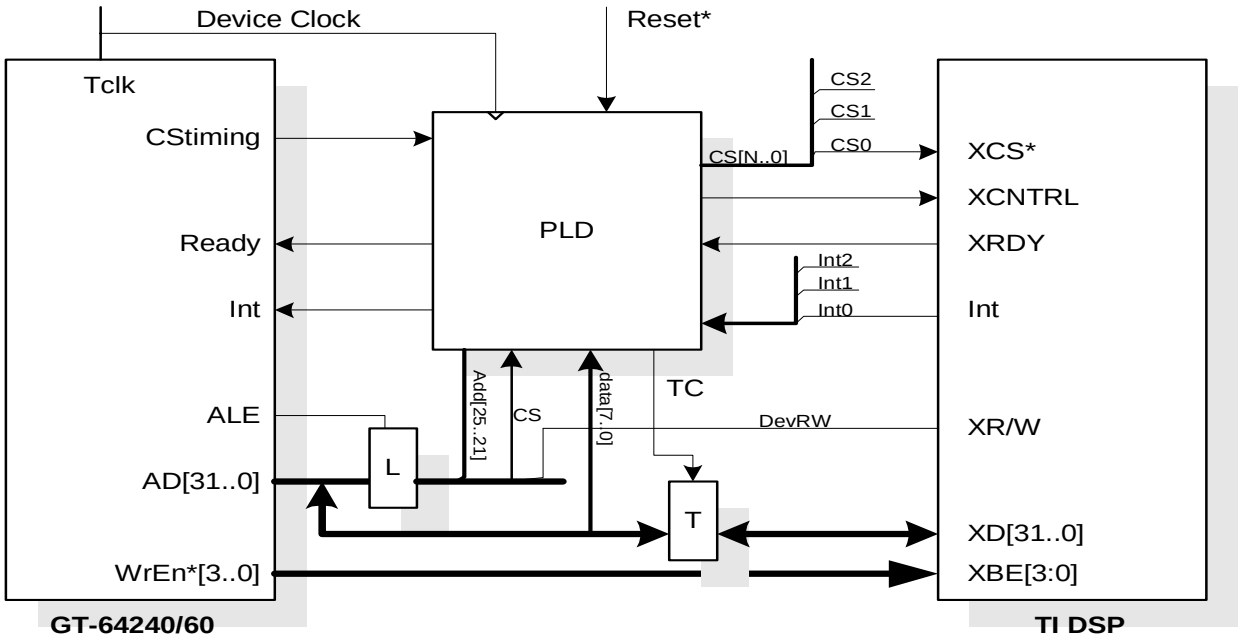
2. Interface

The Expansion bus may operate in Asynchronous or Synchronous mode. In Asynchronous mode, the interface is simpler to implement (see Section 2.1) and the transfer rate is high enough to support heavy load configurations. The bandwidth calculation is described in Section 1.3.1.

2.1 Asynchronous Expansion Bus

2.1.1 Hardware Interface

Figure 2: Hardware Interface

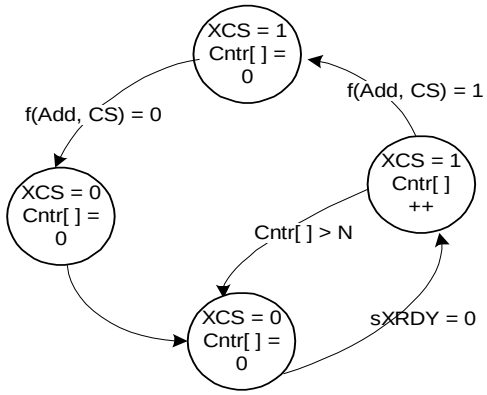


The GT-64240/260 may initiate by single or burst transactions. In Asynchronous mode the Expansion bus supports only single transactions, so the PLD has to convert burst transactions into single transactions.

The Int signal on the GT-64240/260 interface comes from the PLD. One of the problems which is discussed in Section 2.1.4 is the Interrupt mechanism. The PLD is used as an Interrupt Controller. All the interrupts from the DSP go to the PLD, which generates interrupts to the GT64260/40. Once an interrupt is received on the GT64260/40, the host CPU reads a Cause register on the PLD, to verify the source of the Interrupt. It then activates the DMA to perform the transfer to/from the right DSP.

Figure 3 illustrates the State Machine which is used to generate the XCS signal. Table 1 describes the signal interface.

Figure 3: State Machine Used to Generate the XCS Signal



The Value N in the Sate Machine shown in Figure 3 defines the delay between the deassertion of the XCS* signal and the next assertion. According to the GT-64240/260 datasheet, this time should be at least 4 x P, where P is the DSP clock cycle.

Assuming a DSP clock cycle of 4 ns (250 MHz) and a GT-64240/260 device bus cycle of 10 ns (100 MHz). The value N should be at least 2. In general, N should be configured such that:

$$N \times T > 4 \times P$$

Where:

T is the GT-64240/260 Device Bus Clock cycle

P is the DSP clock cycle

The signal names in Table 1 are taken from Figure 3.

Table 1: Signal Names

Expansion Bus Pin	GT-64240/260	Notes
XCS*	Combination of: CS, CStiming, XRDY, Addr[23..21] (see Figure 3)	See State machine in Figure 3.
XCNTRL	Addr[25]	Can Also be any other Addr signal.
XR/W	DevRdWr	
XD[31..0]	Driven by AD[31..0] through bus transceiver.	The bus transceiver is used to reduce load on the Device bus, and also to enable configuration of the DSP during reset.
XBE[3:0]	Driven directly from GT-64240/260 WrEn signals	The signals need to be buffered.
XRDY	Ready	Ready is asserted on XRDY assertion and deasserted one clock later.

2.1.2 Implementation for up to Eight DSPs

The GT-64240/260 has eight independent DMAs, which may be used to transfer data from the host CPU local memory to the DSP. The DMA may be triggered in block mode, by the CPU or by hardware using trigger mode.

The CPU may spend up to 32% of its time handling the overhead on the interrupts from the DSPs (see Section 1.3.1). In order to prevent this, the Interrupt of the DSP can be connected as a DMA request to the GT-64240/260, so it does not require any CPU bandwidth. The IDMA of the GT-64240/260 will act according to the DMA request.

2.1.3 Timing Considerations

The frequency on the GT-64240/260 Device bus can reach 100 MHz. The DSP Asynchronous bus runs more slowly, depending on the CPU clock speed (200/250/300 MHz).

In this section we assume that the GT-64240/260 runs at a frequency of 100 MHz, and we use a DSP running at 250 MHz. If you work with different frequencies, you should change the calculations in this section accordingly.

Figure 4 shows the waveform for a burst write. The Write parameters of the GT-64240/260 should be configured as follows:

Table 2: GT-64240/260 Device Parameters for a Burst Write

Parameters	Value	Comments
ALE2Wr	2	Shortest time.
WrLow	1	Shortest time. The actual time depends on the Ready signal
WrHigh	1	Shortest time.

Figure 4: Write Transaction to the DSP

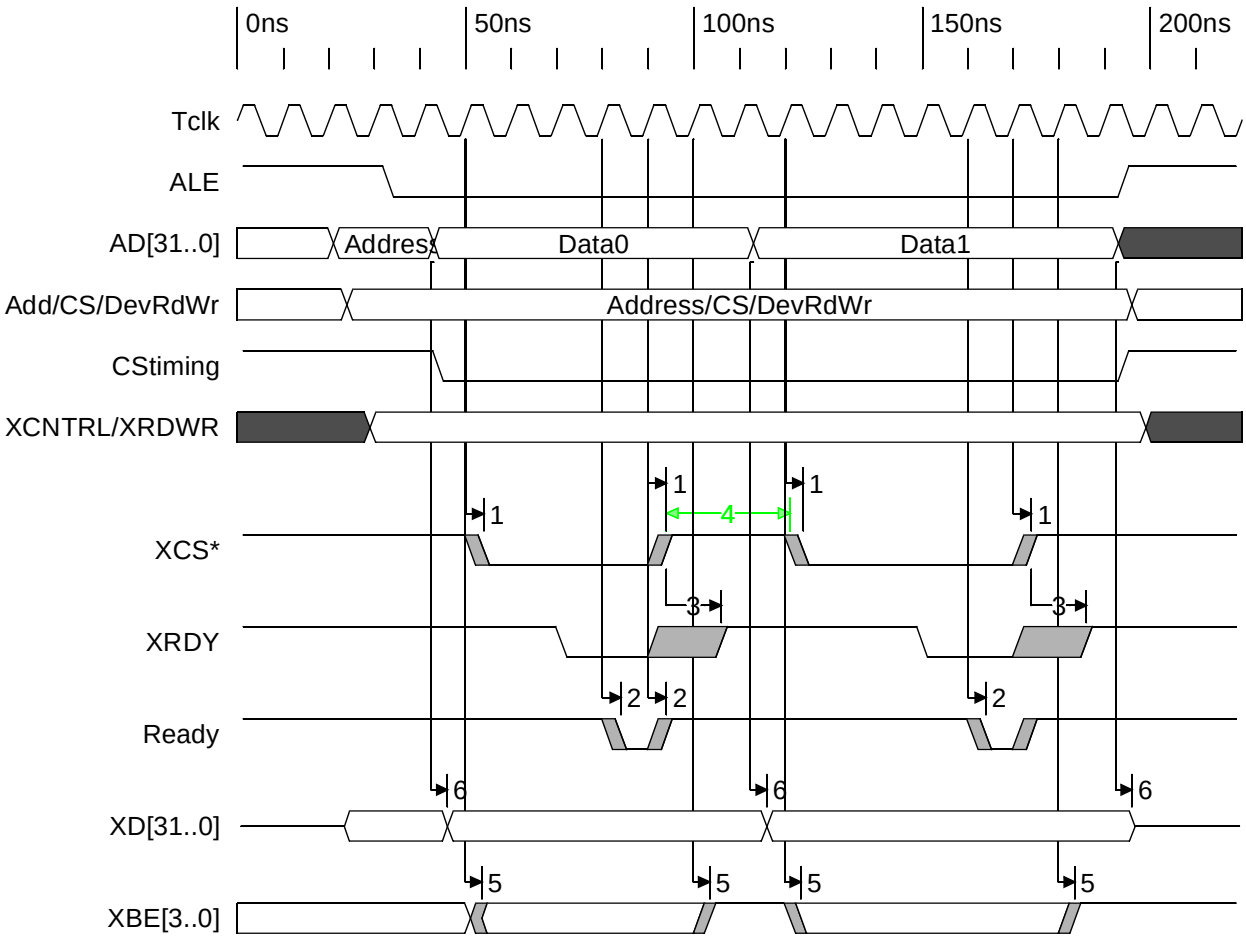


Table 3: DSP Write AC Timing

No	Parameter	Min [ns]	Max [ns]
1	XCS* clock delay	2	6
2	Ready clock delay	2	6
3	Delay XCS* high to XRDY high	0	12
4	XCS* high	4p	
5	XBE clock delay	1	3.5
6	AD to XD delay		4

p = 1/DSP clock frequency. For example, for a 250 MHz CPU clock, p = 4ns.

Figure 5 illustrates a Burst Read initiated from the GT-64240/260. The device parameters are as follows:

Table 4: GT-64240/260 Device Parameters for a Burst Read

Parameter	Value	Comments
Acc2First	1	Shortest time. The actual time depends on the Ready signal
Acc2Next	2	Shortest time. The actual time depends on the Ready signal
TurnOff	2	Shortest time.

Figure 5: Read Transaction to the DSP

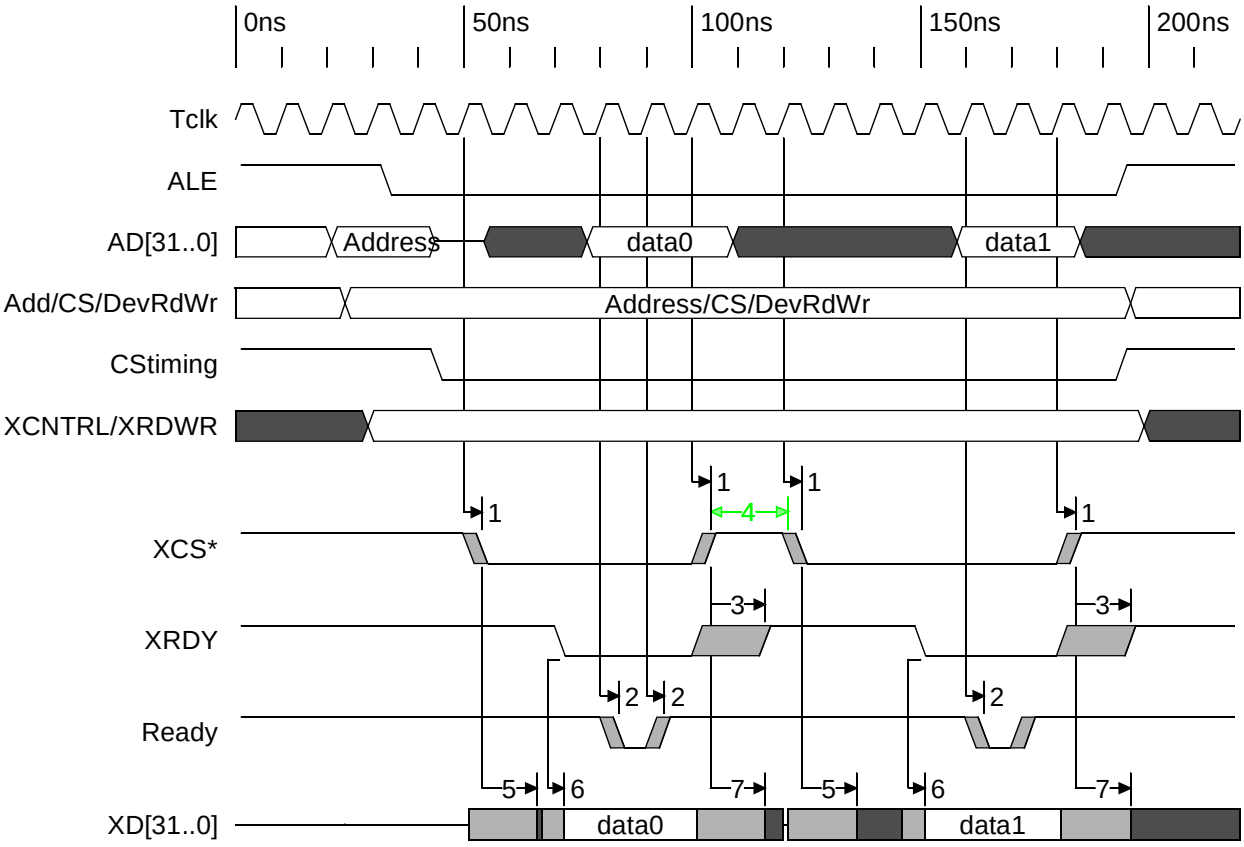


Table 5: DSP Read AC Timing

No	Parameter	Min [ns]	Max [ns]
1	XCS* clock delay	2	6
2	Ready clock delay	2	6
3	Delay XCS* high to XRDY high	0	12
4	XCS* high	4p	
5	XCS* low to XD low impedence	0	
6	XRDY* low to data valid	-4	1
7	XCS* high to XD invalid	0	12

p = 1/DSP clock frequency. For example, for a 250 MHz CPU clock, p = 4ns.

2.1.4 CPU Bandwidth: Interrupt vs Polling

Using the data in Section 1.3.1, there may be 10000, 20000 or even 40000 interrupts per second, depending on the DSP configuration.

To minimize the overhead on Interrupts, we can implement a polling mechanism. The Interrupts from the DSP are sent to the PLD, which includes a register with one bit per DSP. This register is mapped on the Device bus of the GT-64240/260. The CPU may poll the register and serve the Interrupt in Polling mode. The restriction here is that the CPU should be able to visit each DSP every 20 µsec for Read and Write.

The actual data transfer is performed by the GT-64240/260 IDMA engines. In its local memory, the CPU prepares descriptors which direct the IDMAs to perform the data transfer.

As soon as the CPU realizes the source of the DSP, it activates one of the DMAs to perform the transfer. The CPU does not perform the data transfer itself.

2.1.5 Data Transfer Rate

Using the example in Section 1.3.1:

Number of DSPs = 16

Number of Voice Channels per DSP = 12

Thus a transfer rate of 3 MB per second. is needed. This may increase up to 6 or 12 MB per second, if we double the number of DSPs and the number of channels per DSP.

Looking at the waveforms in Section 2.1.3, we see that the transfer rate is ~50 MB per second.

This transfer rate is well above any potential bandwidth requirements. Thus the Asynchronous interface is good enough, and it does not require the use of Synchronous mode.

2.1.6 Multiple DSPs on a Single Device Bus

As mentioned in previous sections, the basic topology is multiple DSPs on a single Device bus. Figure 6 shows the interface between the DSP and the GT-64240/260. In this section we focus on the electrical problems which are expected due to the extreme load on the Device bus.

The AC timing of the GT-64240/260 and the DSP assumes a total load of 30pf.

The loading of each pin on both buses is about 10pf.

When an IBIS simulation is run, with 16 DSPs hanging on the Device bus, it shows that there is an extra delay of 12 nsec on the nominal timing numbers. Adding the extra delay still does not violate the timing, but it creates a very slow slew rate, which may lead to high current consumption in the chip.

To be on the safe side, we recommend dividing the DSPs into four groups. On the Data bus each group is driven by a transceiver, and on the Control bus each group is driven by a buffer.

The Control (Dir, OE) on the transceiver is generated by the PLD. Figure 6 shows the recommended load balancing for multiple DSP configuration.

Figure 6: Recommended Load Balancing for the Multiple DSP Configuration