HUNT ENGINEERING
Chestnut Court, Burton Row,
Brent Knoll, Somerset, TA9 4BP, UK
Tel: (+44) (0)1278 760188,
Fax: (+44) (0)1278 760199,
Email: sales@hunteng.co.uk
http://www.hunteng.co.uk
http://www.hunt-dsp.com

Data transfers to a Host PC, or a processor based module.

Rev 1.0 P.Warnes 24-10-03

The HERON and HEART product range is designed for use in real time systems. HUNT ENGINEERING provides the hardware and the tools *you* need to be able to develop *your* application with them.

While each application is actually trying to solve a different problem, each application will be attempting to use the same data flow model with the HERON and HEART hardware. As such they will all have similar requirements, and the techniques that can be used in developing the system will be similar.

This discusses the issues involved when the data transfer is controlled by a processor running software and using interrupts, DMA etc.

History

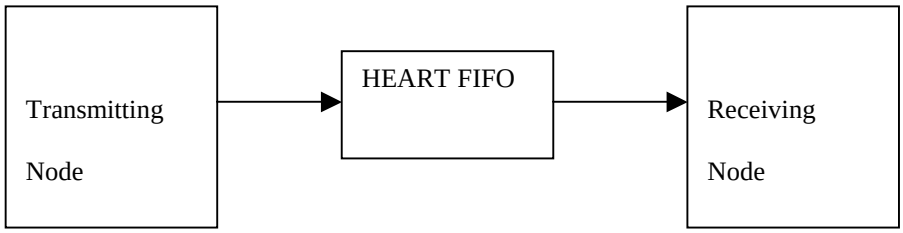
Rev 1.0 First written

Background

Real Time systems are often built with processing elements like DSPs in them. Often there is also a Host computer involved that is used to stream data to, or from. This document highlights the issues that you must be aware of when building such a system. If you don't understand these issues then you will almost certainly fail to make your system run reliably in real time.

System architecture

HERON systems are based upon nodes in the system communicating using FIFO links.



The FIFO is intended as way of de-coupling the clock rates of the two nodes, and not as a storage element to be used in the system design. They are synchronous, which allows each node to drive the clock of it's end of the FIFO with a frequency that is convenient to that node.

The FIFOs provide some flags that indicate "states". Typically there are flags that indicate the end state, i.e. the FIFO is empty or full, and also different flags that indicate a "block" state, i.e. there is space to write at least "n" or at least "n" items to read.

HEART systems for example have the end state flags, and also "almost" flags that indicate a programmable block, and also Block flags that indicate on blocks of 64 words.

Sending data to a PC

A 32bit PCI bus has a maximum data rate of 132Mbytes/sec. This is the rate that is achieved when 32 bits are transferred in every 33Mhz clock cycle, but the bus protocol demands that the address of each burst is transferred first. So it can be seen that you need to use large bursts to achieve high data rates.

The HEPC9 uses what is called a "Master Mode engine" to make the PCI transfers. That will be programmed to use the block flags of the FIFOs so that it knows how much data can be transferred before starting the burst. It tries to continue bursting on the PCI bus as long as there is data or space in the FIFO that is being used. In practice the PC motherboard chipset will restrict the length of the burst that we are allowed. This could be a maximum length, or could be that another device is trying to use the PCI bus.

So we will see some variation between different PCs, simply because the chipsets behave differently.

On top of the PCI transaction level, there will exist an operating system, and a device driver. When the Master Mode engine completes the current transfer, it will use an interrupt to inform the device driver that it needs new instructions. It is the operating system that detects that interrupt and schedules the correct device driver routine to be run. Possibly that device driver routine will then inform the user application and wait for a response from that user application before it can continue.

When a PC is running Windows there is no guaranteed interrupt response time. Microsoft documentation indicates that interrupt response times can be the order of 10 to 100ms. Clearly this is not a real time operating system.

Many documents that discuss these issues can be found by searching the internet – often in documents that are trying to compare a real time extension to that of "vanilla" windows. What is happening is that the operating system is running several tasks and sometimes several interrupts might be received at the same time. The list of things to do is simply worked through in order, until you are serviced.

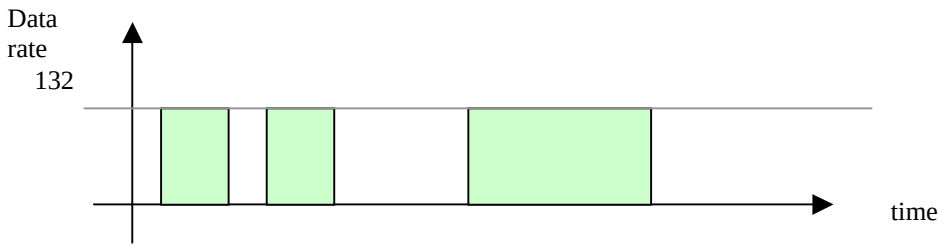
Similarly there are no guarantees given about the time taken to schedule the user code after the device driver indicates that it can be continued.

Even so-called Real Time Operating systems cannot give absolute guarantees on these times. Typically they state items like context switching times and interrupt latencies – when certain conditions apply, i.e. only a single interrupt, or only a single task can continue etc.

This means that regardless of the operating system running on the PC, you cannot ever calculate the maximum time to service an interrupt, or switch to the user program.

Sustainable data rates

In fact the combination of the PCI chipset and the necessity to run an operating system means that the PCI transfers will always be made up of bursts of 132Mbytes/second and 0Mbytes/second :-



What we cannot predict for a given system are the length of the bursts, nor the length of the gaps. All we can do is to say that the average rate will be between 0 and 132Mbytes/second, and measure it over a long period for a given system.

Instantaneous data rates

Given this behaviour we need to be concerned about the average throughput that we can achieve, but we also need to be concerned at a much lower level. If we don't design our systems properly we can have issues with the fact that often the instantaneous data rate is 0! If we have a real time system we have data flowing through it all of the time. If the transfer to the PC stops for an indeterminable time, and it is linked directly to our real time data flow, then we will certainly lose data when the PCI bus is not transferring. The HERON-FIFOs help with this, in that they provide a small buffer that can absorb data while the PC does not respond. However the size of these buffers is around 500Words, i.e. 2000Bytes. If the "gap" is as high as 100ms, then this buffer can solve the problem for us only if the data rate is less than 20,000 bytes/second!

If we look at it the other way around, if we have a data rate of 50Mbytes/second we need a buffer of at least 5Mbytes!

So it is really important that we use some data buffer to de-couple transfers to the host from the real time part of our system. For that we can implement a circular buffer (in effect a large FIFO) using either a C6000 based module, or an FPGA+memory module.

C6000 Transfers

The C6000 also uses a bus to access the HERON FIFOs. This bus might also be shared with other things like the off chip memory etc. The C6000 buses are optimised for burst transfers, so just like the with PC the transfer is better made using the Block flags. Actually HERON-API also uses the on chip DMA engines to make the transfers, and it is them that are triggered from the block flags.

The user application will request larger blocks from HERON-API and will receive indication of when the transfer is complete. At that time the User program will need to respond and start the next transfer, so there are still interrupts and context switches involved.

DSP/BIOS is used to service the interrupts and to schedule the tasks, and is talked about as real time. Although there are some better measurements and guarantees that for windows (indeed the times are several orders of magnitude lower than those experienced when using windows) the benchmarks are still made under very controlled conditions.

In a real system, where several sources of interrupts exist, and several tasks/threads etc are running, you still cannot predict the latencies that will occur.

Again the data flow will be bursty, but the “dead” times will be both shorter and less varying than for the PC.

Here the HEART FIFOs might be enough to absorb the “dead” times for quite reasonable data rates (10s of Mbytes/sec) but it is still necessary to check if that is the case for your particular system implementation. When using direct I/O modules like those in the GDIO range, this simply defines the maximum sample rates for your system. If you have a module that also has an FPGA, then it is quite easy to implement a FIFO inside that FPGA to buffer data when the C6000 is not responding, and we’d recommend doing that whenever you have the resources free in your FPGA to do it.

Double buffering

When you are using the PC or a processor module, to achieve the highest sustained transfer rates you really want to be transferring all of the time. I.e. you want to minimise the periods of 0Mbytes/second. An important part of this is to keep the time that your program spends between receiving an indication that a transfer is complete, and starting the next transfer, as short as possible.

Both the Host API and HERON-API use an asynchronous model that allows you to be processing while the transfer is taking place.

So you shouldn’t use the technique :-

```
Start transfer
Wait for end
Process data
```

But rather:-

```
Start transfer
Process data
Wait for end
```

Which means that you need (at least) two data buffers, so that you can process one while transferring data into the other.

HEART timeslots

The peak bandwidth of a HEART FIFO connection is guaranteed by design. When hardware such as FPGA modules are used to transfer data, you can clearly demonstrate the bandwidths that are guaranteed.

However it is not as simple when using a processor based node. The bursty nature of the data will almost always result in a lower bandwidth than that guaranteed by the HEART connection. This is because the bandwidth that is guaranteed by the connection will actually be limiting the peak rate, but can do nothing about the times when there is no data transferred.

The way that HEART is constructed helps with this, because the HEART FIFO is actually made from 2 FIFOs, with the HEART connection in the middle. So actually the processor can burst into and out of the FIFO at it's own end at the full module speed. It is only when that FIFO is used up that the HEART limit applies.

For this reason it is difficult to predict exactly what bandwidth will be sustainable when using a processor node. You need to carefully design your system to continue to operate with the effects that will be introduced.