The designs of the master and slave CCB FPGAs

[Document number: A48001N004, revision 1]

Martin Shepherd, California Institute of Technology

April 5, 2004

This page intentionally left blank.

Abstract

The aim of this document is to detail the design of the CCB FPGA firmware, and define its interfaces to the rest of the CCB hardware. The design will be presented in a hierarchical manner, starting with block diagrams of major components and their interconnections, and ending with the VHDL code that synthesizes the lowest level components displayed, and connects them together.

[This is still a work in progress]

Contents

List of Figures

Chapter 1

Introduction

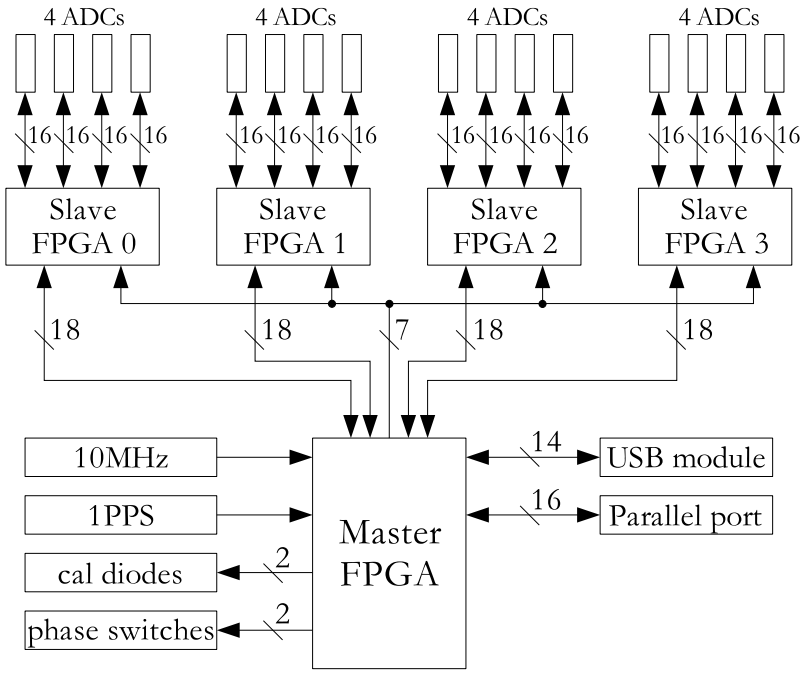


Figure 1.1: An overall summary of the FPGA connections

Figure 1.1 shows the overall architecture of the FPGAs with respect to the rest of the CCB. At the heart of the system, the Master FPGA controls 4 slave FPGAs, receives commands from the computer's EPP parallel port, delivers interrupts to the computer, via the same link, dispatches observed data to the computer over a USB link, and controls calibration diodes and phase switches in the receiver. All of its timing signals are derived from the Green Bank 10MHz and 1PPS reference signals.

Under the direction of the Master FPGA, each of the slave FPGAs continuously reads 14-bit data samples from 4 ADCs at 10MSPS, and integrates these samples until told to deliver them to the Master FPGA, or, when in dump mode, delivers them un-integrated to the Master FPGA.

The following two chapters detail the internal logic and external interconnections of the Slave and Master FPGAs, respectively.

Chapter 2

The slave FPGAs

There will be 4 slave FPGAs controlled by one master FPGA. All of the slave FPGAs will be identical, so this chapter documents the internal components, and external I/O connections of a single slave FPGA. Figure 2.1 shows the layout of a slave FPGA, showing the major logic components within the FPGA, the internal interconnections between these components, and all of the external I/O-pin connections to the 4 ADCs to the left, and to the master FPGA, shown at the bottom of the diagram.

2.1 An overview of the internals of a slave FPGA

Starting from the left hand side of the diagram, the 14 bit data-signals and 1-bit overflow signal of each of the 4 ADCs, are simultaneously latched into a corresponding 15-bit register at the start of every 10MHz clock cycle. Simultaneously, the previous values of these registers are read by integrator components. When in dump mode, the output of one of these 15-bit outputs is also siphoned off, to be read directly by the master FPGA. The outputs of the integrators are subsequently flash loaded into a PISO, at the end of each integration, ready to be read out by the master FPGA.

2.1.1 Normal integration mode

When not in dump mode, the output of each ADC, delayed by one clock cycle by the input register, and by another 4 clock cycles by the pipeline within the ADC, is read into an integrator component.

The integrator component, which will be detailed shortly, either ignores the new sample, if the `drop` signal from the master FPGA is asserted, or adds it to the accumulation bin specified by the 2-bit `phase` signal, as received from the master FPGA.

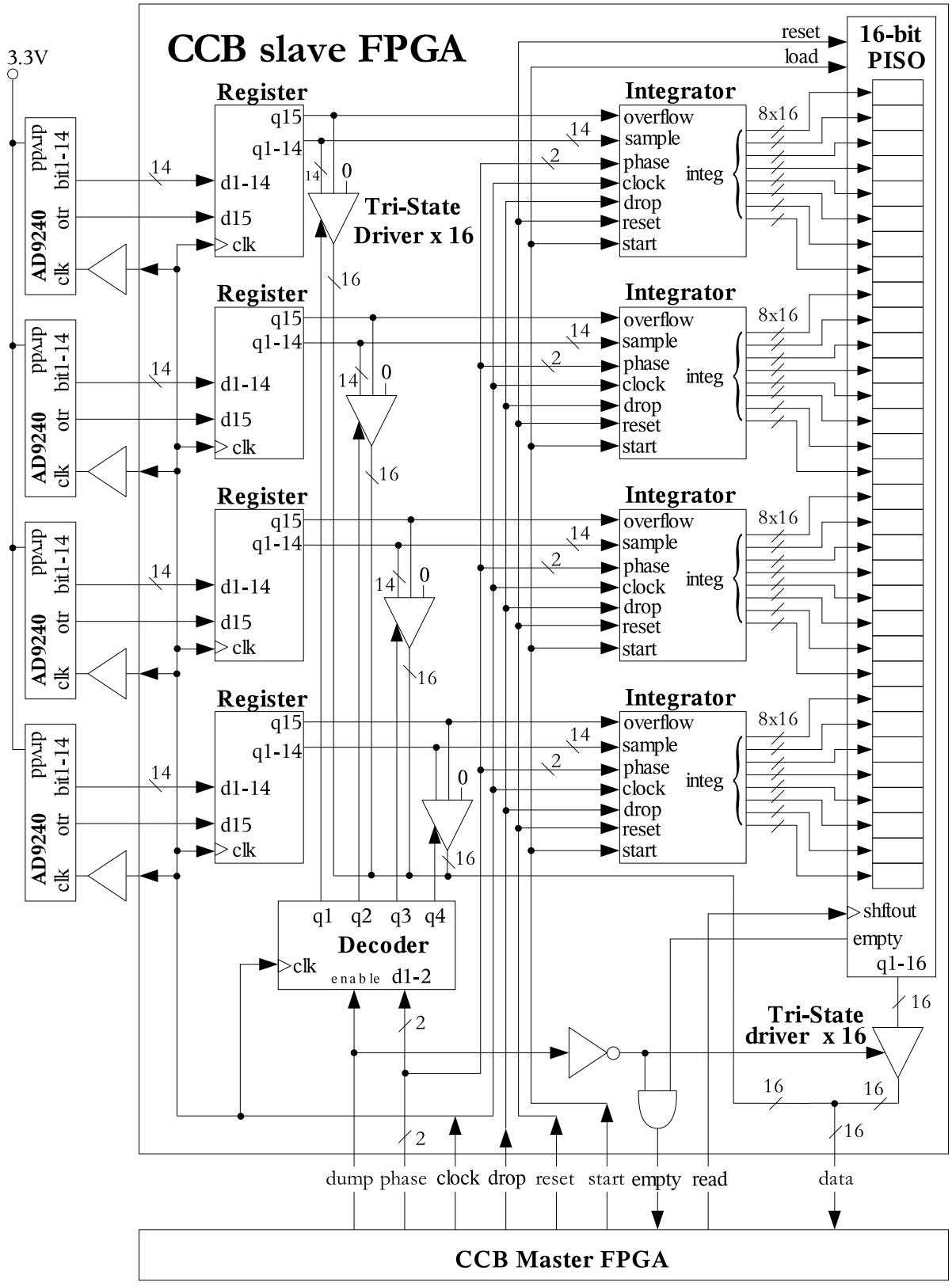


Figure 2.1: The top-level design of the slave FPGA

At the end of each integration period, the master FPGA asserts the `start` signal for one clock cycle. This causes the output PISO, on the right of the diagram, to flash-load the contents of all of the 32-bit accumulator output bins of the integrators, and at the same time, the integrator components clear all of their accumulators, and add in the latest sample to the appropriate empty bin, unless the `drop` signal is asserted.

The accumulator bins are 32-bits wide, but there are only 16 pins available for clocking the data out of the slave FPGA. Therefore a 16-bit wide FIFO is used, with two entries assigned to each accumulator bin. Since there are 4 accumulator bins per integrator, 4 integrators, and 2 PISO entries per accumulator, the 16-bit wide PISO will be 32 entries in length.

Once the PISO has been flash loaded with the accumulator values, its `empty` signal becomes deasserted, which tells the master FPGA that there are new integrations waiting to be read into its output PISO. When the master FPGA is ready to do this, it clocks the `read` line of the slave FPGA, in sync with the global 10MHz clock, to read out the PISO. Once all data have been read from the PISO, its `empty` signal is asserted, telling the master FPGA to direct its attention to reading data from another slave FPGA, if any.

2.1.2 Dump mode

In dump mode, although the integrators still integrate data, the tri-state output driver following the PISO, is placed into a high impedance state, to enable one of the 16-bit tri-state drivers at the outputs of the ADC input registers, to instead direct the output of a chosen ADC, directly to the `data` output of the slave FPGA. In this mode, the 2-bit `phase` signal is re-interpreted as the address of the ADC to be dumped, so an address decoder, which is only enabled when the `dump` signal is asserted, is used to enable the chosen tri-state buffer, according to this address.

At the end of dump mode, the master FPGA briefly asserts the slave FPGA's `reset` signal, to discard the garbage contents of the integrators and the output PISO, before normal integration mode resumes.

2.1.3 External connections

The FPGA I/O pins should be configured for 3.3V logic levels.

Provided that the `DRVDD` pins of the ADCs are connected to a 3.3V power-supply, then the output data pins of the ADCs can directly drive the corresponding input pins of the FPGA. This may not be the case if there are cables or filters in between. Unfortunately, the ADC clock input appears to require higher CMOS levels, regardless of the voltage at the ADC `DRVDD` pin. Hence the individual 3.3V to CMOS buffer amplifiers between the FPGA clock output pins and the ADC clock input pins. If the clock output pins of the FPGA are close

together, then presumably the number of FPGA clock output pins could be reduced; as could the number of separate buffer chips.

Note that an assumption in this design, which may not be correct, is that the interconnections between master and slave FPGAs, and between the slave FPGAs and the analog board, will be via daughter-board connectors, rather than via cables and EMI filters. The scenario in mind has a central board housing the master FPGA, the USB interface and the parallel port interface, then on the component side of this board, would be 4 daughter cards holding the slave FPGAs, and on the other side of the board, shielded by the ground plane of the master FPGA board, would be the daughter board housing the analog electronics. The connections from the ADCs to the slave FPGAs would thus go via two levels of daughter card connectors. If a significantly different scheme is adopted, which includes inter-board cabling and/or EMI filtering, then extra buffer amplifiers may need to be added for the data lines, along with some means to compensate for any rise time delays added by EMI filters.

Chapter 3

The master FPGA

Figure 3.1 shows the layout of the master FPGA, showing its major internal components, along with their interconnections, and all of the external I/O-pin connections to external chips. The central brain of this design, is the *State Generator* component, which orchestrates the timing and values of all control signals going to the other components and the slave FPGAs. The *State Generator* is in turn told what to do by the computer, via the *Control Gateway* component, which handles all interactions with the parallel port interface. The *Data Dispatcher* component is responsible for sending integrated and dump-mode data to the computer, via the USB interface.

3.1 The Control Gateway

The *Control Gateway* handles all interactions with the CCB computer's EPP parallel port interface. It places commands and configuration data that it receives from the computer, within an array of registers, which are always visible to the *State Generator*. It also raises parallel-port interrupts, when so directed by the *State Generator*. Note that apart from the interrupt mask, which is read by the computer from its interrupt handler, all data is directed from the computer to the FPGA.

The reset signal of the EPP parallel port is used to reset the firmware and the USB chip. This can be asserted at any time by the device driver in the CCB computer, and this will automatically be done every time that the device driver is loaded or reloaded.

The *Control Gateway* component will be designed to present an 8-bit register based interface to the computer. This is simplified by built-in support for separate address and data cycles in standard EPP hardware. In particular, the receipt of an address-byte by the *Control Gateway* will be interpreted as the address of the 8-bit register into which to record subsequently received data-bytes.

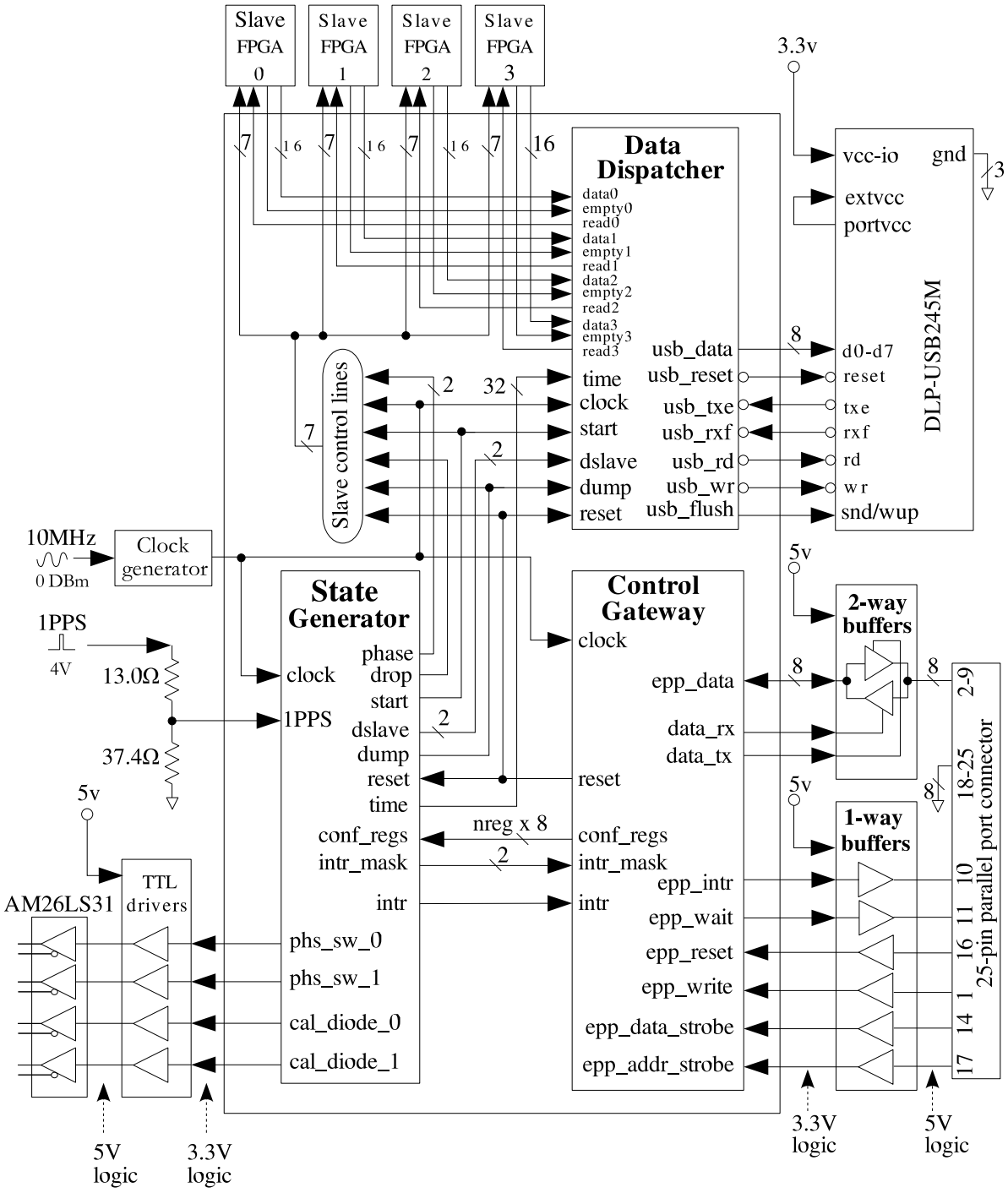


Figure 3.1: The top-level design of the master FPGA

Since no read-back of the internal registers is planned, any read request from the computer will be responded to with the interrupt mask, which will say which interrupts have been generated since the last time that this mask was read. This mask can thus be read from an interrupt handler with a single EPP read, without having to worry about other incomplete multi-byte read transactions.

There are only two times when data will be sent to the master FPGA by the computer.

1. When starting a new scan, a write to the control register will be used to prepare the *State Generator* for reconfiguration. This will be followed by multiple writes, to send the configuration data of the new scan, and terminated by the command which instructs the *State Generator* to start the new scan.

   Since the FPGA does nothing with the configuration data that it is sent, until it is told to start the next scan, it is safe to send the values of multi-byte configuration registers, a byte at a time.

2. During a scan, the configuration of the calibration diodes (for the integration after the next integration), will be sent, on receipt of each end-of-integration interrupt.

   Since between scans, only the calibration diode configuration register is written to, the device driver will send the address of this register once per scan, just after starting each new scan, such that the interrupt handler needn't specify the register to write its cal-diode reconfiguration info into, before writing said info. Thus at the end of each integration period, there will be one EPP read to get the interrupt mask, plus one EPP write to reconfigure the calibration diodes.

3.2 The Data Dispatcher

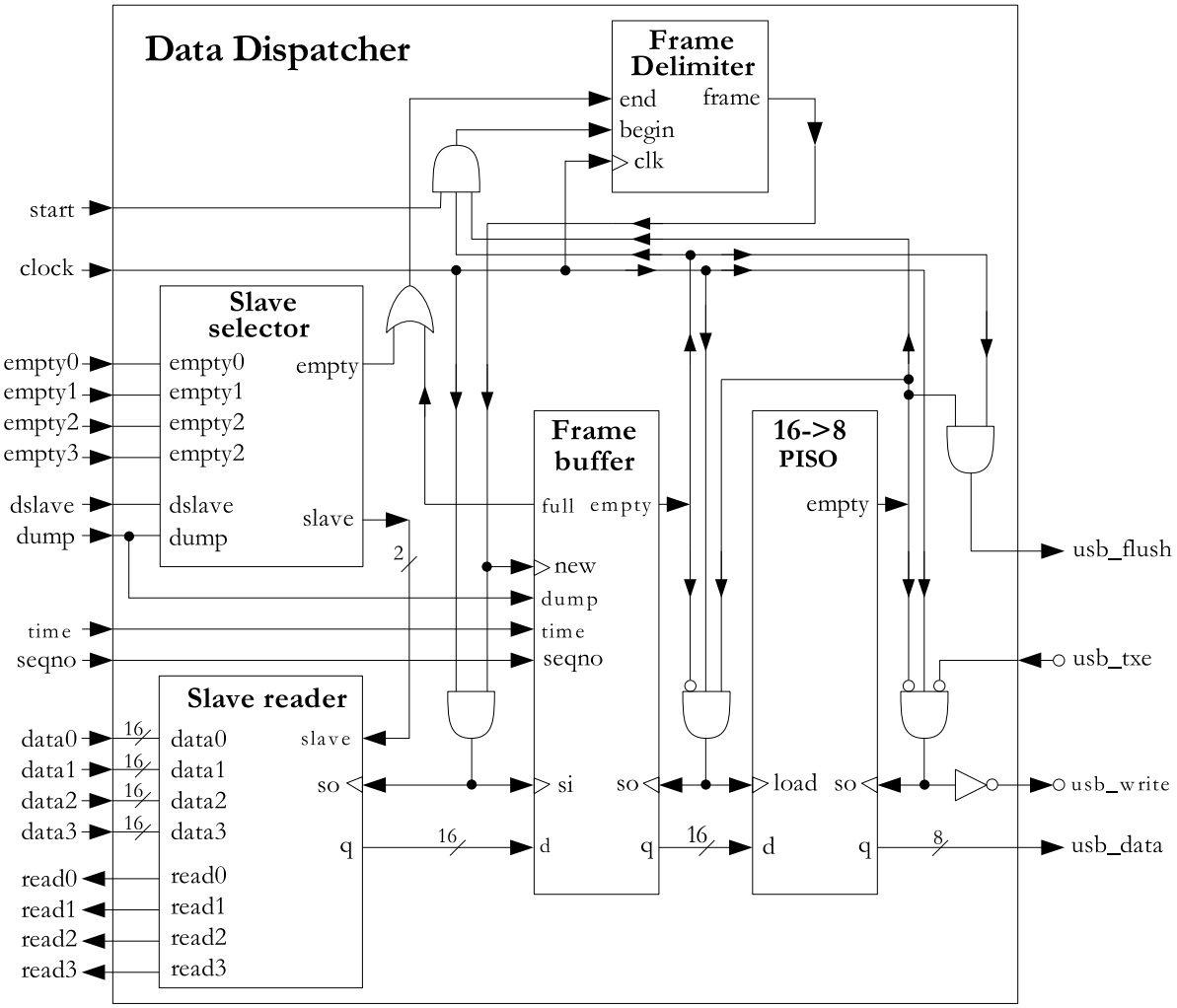
At the end of each integration period, and at the start of dump mode, the *Data Dispatcher* component will read integrated or dump-mode data from the slave FPGAs into a large FIFO, then stream the contents of this FIFO, preceded by a header, to the computer, via the USB link. Note that all communications over the USB bus will be directed from the FPGA to the computer. Thus, although the read (rd) and read-enable (rxf) pins of the USB interface are shown as inputs to the *Data Dispatcher*, there are no plans to use them at the moment.

Note the use of the DLP-USB245M module. This is a tiny PCB module containing a 6MHz crystal, a surface-mount FT245BM USB1.1 chip, a USB connector and all the interconnections needed between these parts. The PCB is just 1.5×0.7 inches in size, and the USB connector sticks out a further third of an inch from one end. The module can be soldered onto the CCB PCB, via 24 dual in-line pins. Its data-sheet can be downloaded from:

`http://www.dlpdesign.com/usb/dlp-usb245m12.pdf`

The two of these modules that I bought for testing the FT245BM, I got from a company called Saelig (`www.saelig.com`), which is an official US distributor for the FT245BM. The modules arrived overnight. Since then, I have noticed that Mouser Electronics carries them as well. Their catalog number at Mouser is 626-DLP-USB245M, and they cost $25.

3.2.1 The internals of the Data Dispatcher



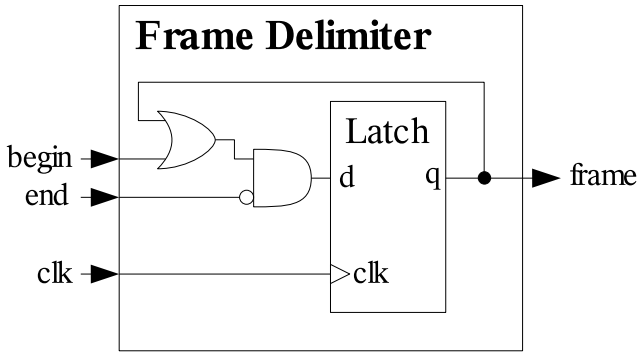
The above figure shows the building blocks of the Data Dispatcher, and how they are interconnected. All data available from one slave at a time, are read by the Slave Reader and passed on to the Frame Buffer. The current slave to read from, is selected by the Slave Selector, which cycles through the slaves in reverse numerical order, when reading out integrated data, or settles on the slave that is specified by its `dslave` input signal, when in dump mode. The `frame` output signal of the Frame Delimiter controls the packaging of output

data frames within the Frame Buffer. When the `frame` signal goes high, a new output data frame is initialized, and data from the slaves start to be transfered to the Frame Buffer. When all data from the slaves have been transfered, or the frame-buffer becomes full, the `frame` signal goes low, to terminate the frame. The `frame` signal doesn't go high again until the next rising edge of the `start` signal from the State Generator, and even then, it only goes high if the previous contents of the Frame Buffer have been completely transfered to the CPU over the USB bus. These measures prevent a new frame from trampling on an incompletely sent frame, and prevent temporary buffer-full conditions in dump mode, from creating unpredictable sampling gaps within a frame.

The Frame Buffer contains a large FIFO for the slave data, plus a small PISO which is initialized to contain a frame header, consisting of a time-stamp, an indication of whether dump mode was in effect, and a scan sequence number.

Once a new frame has been started, the contents of the Frame Buffer are clocked out, starting with the frame header, and followed by the slave data, in 16-bit chunks. Each chunk is loaded into a 2 entry, 8-bit wide PISO, such that 8-bits at a time can be clocked out to the USB chip, whenever the USB chip has room in its own internal FIFO, for more data. When both the Frame Buffer and the 8-bit PISO have been emptied of all data, the `usb_flush` signal is asserted, to tell the USB chip to send all remaining data to the CPU, as soon as possible, without waiting for enough data to precisely fill up the final USB block.

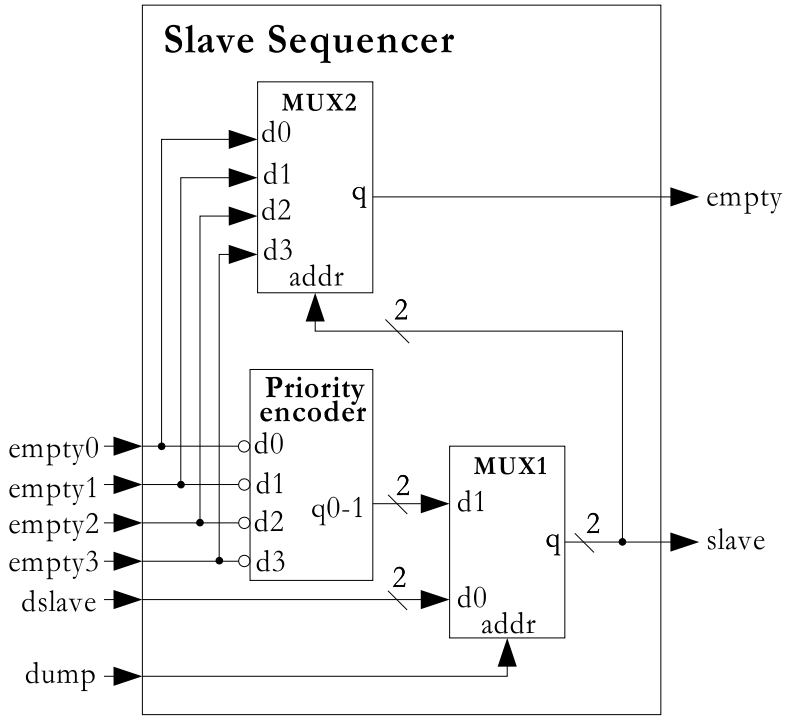
The internals of the Frame Delimiter



When the `end` input signal is asserted at the start of a clock cycle, the `frame` output becomes deasserted, regardless of the state of the `begin` input signal. This terminates the assembly of an output frame. The `frame` signal does not go high again, until the start of a clock cycle when the `begin` signal is newly asserted, after the `end` signal has been deasserted.

This means that a new frame will not begin if the `begin` signal is asserted while the `end` signal is still asserted, and that if the `end` signal becomes asserted during a frame, the frame is terminated, regardless of the state of the `begin` input.

The internals of the Slave Selector



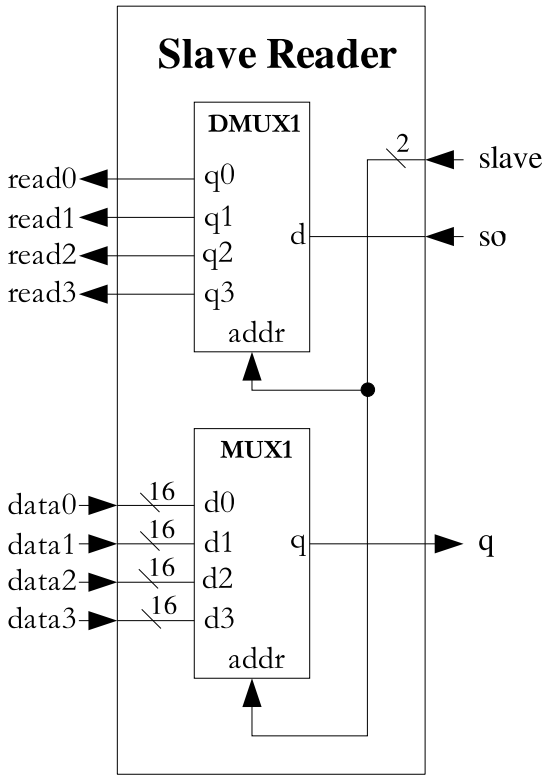
The purpose of the Slave Selector is to tell the Slave Reader which slave FPGA to read data from, and to provide an indication of whether any data remain to be read, both from the slave that is being read from, and from any slaves that remain to be selected. The `slave` output signal specifies the selected slave FPGA as a 2-bit slave number, and the `empty` output specifies when the slaves have been exhausted of data to be read. These outputs are only valid on the rising edges of the clock.

In normal integration mode, all of the slave FPGAs have data available in their PISOs at the end of each integration period. So all of their `empty` lines become deasserted. A priority encoder in the Slave Selector looks at logically inverted versions of these `empty` signals, to select the highest numbered slave that still has data available to be read. Thus as soon as the highest numbered slave has had all of its data read out by the Slave Reader, the Slave Selector detects this, and selects the next highest slave that has its `empty` signal deasserted. In this way, it causes the Slave Reader to read-out the slave FPGAs in the order 3,2,1,0. At the same time, it routes the `empty` input signal of the selected slave to the `empty` output signal of the Slave Selector. Thus at the start of each new clock cycle, the output `empty` signal of the Slave Selector is high until all of the slaves have been exhausted of integrated data.

In dump mode, as specified by the `dump` input signal being asserted, the slave selection of the priority encoder is overridden by the slave number specified by the 2-bit `dslave` input

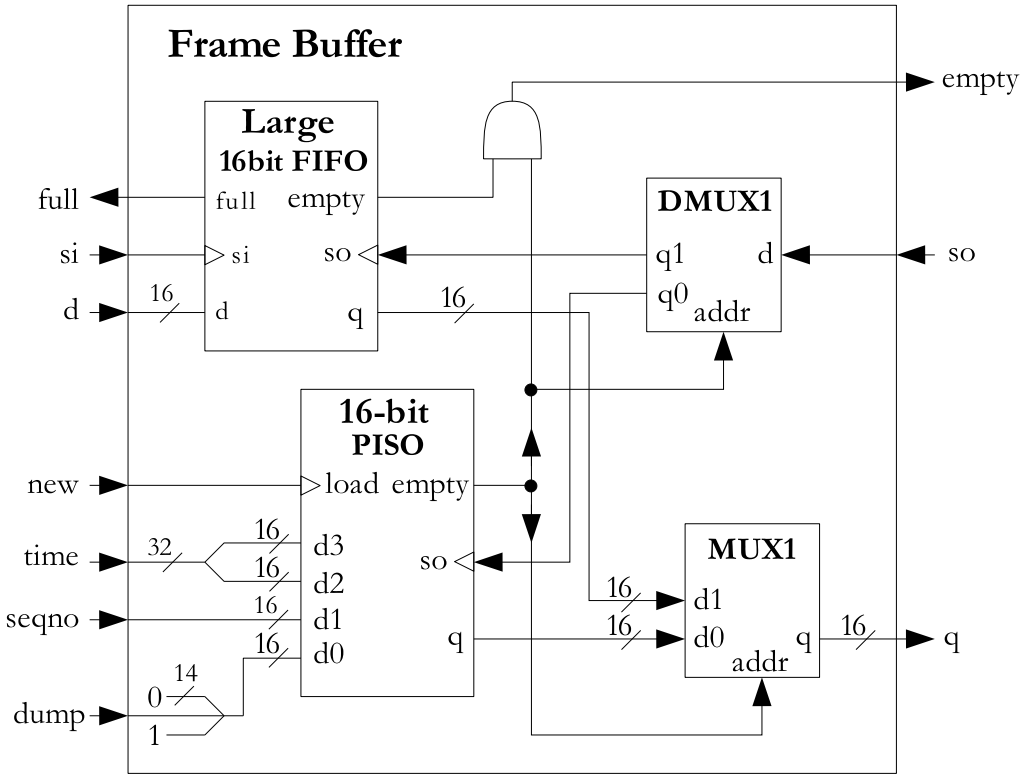
signal, and the output `empty` signal reflects the `empty` input signal from this slave. Since in dump mode, slaves can indefinitely deliver data at 10MSPS, the `empty` signal remains deasserted in this mode, until the State Generator deasserts the `dump` signal.

The internals of the Slave Reader



The Slave Reader uses its `slave` input signal to connect the `data` and `read` signals of the correspondingly numbered slave FPGA, to the `so` input and `q` output signals of the Slave Reader. Thus a read-strobe on the `so` input of the Slave Reader, is routed through demultiplexer, DMUX1, to the `read` output going to the selected slave, while the data that this returns from the selected slave is routed through the multiplexer, MUX1, to the `q` output of the Slave Reader.

The internals of the Frame Buffer



The Frame Buffer has two major parts, a 16-bit wide PISO containing a frame header, and a large 16-bit wide FIFO containing integrated or dump-mode data. The outputs of the Frame Buffer simulate the output of a virtual 16-bit wide FIFO, formed from the serialized concatenation of the contents of these two parts, with the header coming out first, followed by the data.

The glue logic with which the Data Dispatcher embeds the Frame Buffer, ensures that a new frame can not be started unless the Frame Buffer is empty. Thus at the start of a new frame, both the data FIFO and the header PISO of the Frame Buffer, are guaranteed to be empty. The start of a new frame is signaled by a rising edge on the `new` input, which is externally connected to the `frame` output of the Frame Delimiter in the Data Dispatcher.

At the start of a new frame, the rising edge of the `new` input signal causes the header PISO to load the contents of the header, as derived from input signals received from the State Generator. The header currently consists of 4 16-bit words, which are as follows.

- The first of the 16-bit header words identifies the type of frame that is being packaged, and since it has a value that doesn't look like a data value, the CPU can use it as the

indication of the start of a new frame, in case other frame separation measures don't work.

Note that since a normal data value will either be zero, in the case of a missing ADC board, or a significantly non-zero number, in the presence of sampled noise, a small non-zero 16-bit number, is a good choice for something that shouldn't look like a data byte.

In practice, the least significant bit is hard-wired to be 1, to ensure that the header is always zero, the second bit is 0 in normal integration mode, and 1 in dump mode, and the remaining 14 most significant bits are always 0. Thus this 16-bit integer will have the value 1, in normal integration mode, and 3 in dump mode.

- The second of the header words is a 16-bit scan sequence number. This reflects the value of a counter in the State Generator, which is reset to zero, whenever the FPGA firmware is reset, and incremented by 1 whenever a parallel port command to start a scan or intra-scan is received. The CPU will use this both to watch for the first integration of a newly requested scan, and potentially to watch for missing scans.

- The 3rd and 4th words are the least and most significant 16 bits of a 32-bit time-stamp. This is the value of a counter in the State Generator which is reset to zero at the start of each new scan, and incremented by 1 every one-thousand 100ns clock cycles. Thus the time-stamp measures the time elapsed since the start of the second on which the last scan started, has a resolution of 100μs, and wraps around every 5 days.

  Note that the resolution is one tenth of the minimum required integration period. Greater resolution would be overkill, and would unnecessarily cause the time-stamp clock to wrap-around more frequently. Lower resolution could lead to two integration periods getting the same time-stamp. On the real-time computer, the sum of the absolute time of the 1PPS edge on which the scan was started, and the FPGA relative time-stamp, will form the high-resolution time-stamp that is sent with the data, to the manager.

On the same clock edge at which the `new` signal goes high, data become available from the slaves. These data are synchronously clocked into the FIFO, 16-bits at a time, by the Data Dispatcher, using the `si`, shift-in, input and the `d`, data inputs. This continues until there are no data left to be read from the slaves, or the FIFO becomes full. In either case, the Frame Delimiter disables further input to the Frame Buffer, until the next time that the State Generator asserts the `start` signal, after the contents of the Frame Buffer have all been sent to the real-time CPU.

3.3 The State Generator

TBD

[Note that the reset lines aren't currently shown as being distributed to the slaves by the State Generator*, whereas they must be, to allow the* State Generator *to reset and halt the slave FPGAs, while it is in the process of receiving reconfiguration information for a new scan. Also I need to add a scan sequence number output, and route this to the Data Dispatcher. I'll fix this when I document the design of the State Generator]*