Lectures 17: Point-to-Point Interconnect, PCI Express, and Interrupts

1. Point-to-Point Interconnect using the Intel Quick Path Interconnect (QPI) as an example.
   1.1. Needed because faster data rates do not allow enough time for the synchronization and arbitration functions of shared buses. Exacerbated on multicore chips.
   1.2. QPI introduced in 2008.
       1.2.1. In single core, replaces the front side bus between the processor and the north bridge.
       1.2.2. In multi-core, connect processors to processors and I/O hub.
1.3. Three characteristics of point-to-point interconnect schemes. Very similar to network connections.
   1.3.1. Multiple direct connections between pairs of components. So no need for arbitration. Like routers of the Internet.
   1.3.2. Packetized data transfer. Rather than raw data, packets have control headers and error control codes.
   1.3.3. Layered protocol architecture
       1.3.3.1. Physical Layer = actual wires carrying the signals, and circuitry to support transmission of 1’s and 0’s.
           1.3.3.1.1. In QPI, the unit of transfer is 20 bits.
           1.3.3.1.1.1. Each bit takes a pair of wires called a lane. Each component has a set of 20 lanes for data transmission, 20 lanes for data receiving, as well as one clock lane for each direction, so 42 lanes, and thus 84 wires.
           1.3.3.1.1.2. Transfer speed of 6.4G transfers per second, so 6.4 * 40 bits = 32G Bytes/s
           1.3.3.1.1.3. Uses differential signaling, where one wire has a higher voltage that returns through the other, lower (zero) voltage wire of the lane. Which of the two wires has a high voltage determines whether the bit is a zero or a one.
           1.3.3.1.1.4. This layer also manages the job of collecting together the data of four transmission cycles into a 80 bit units used by the Link Layer.
       1.3.3.2. Link Layer = responsible for reliable transmission and flow control. Think Transfer Control Protocol (TCP) of Internet.
           1.3.3.2.1. In QPI, 72-bit message payload, 8-bit error control code, called a flow control unit, (Flit).
           1.3.3.2.1.1. Sender can only send a certain number of Flit before waiting for acknowledgement of success by receiver. Based on buffer in receiver.
           1.3.3.2.1.2. Uses using cyclic redundancy check (similar to the SECDED we learned in class) to detect errors. If error detected, then sender must resend that Flit, and all subsequent Flits. Note that this is done at the Link Layer, not the Physical Layer.
       1.3.3.3. Routing = provides the framework for directing packets through the fabric of the connected components. Think Internet Protocol (IP).
           1.3.3.3.1. For single core, this is simple. Just indicate which I/O device, memory address, port.
           1.3.3.4. Protocol Layer = provides for the needs of particular type of transfer, e.g. cache coherence.
2. PCI Express = Peripheral Component Interconnect Express, a point-to-point interconnect scheme that supports time-dependent data streams by tagging data.
   2.1. Physical Layer = 1, 4, 6, 16, or 32 bidirectional lanes.
       2.1.1. 128 bits of data prefixed by a 2-bit header block for a total of 130 bit codeword.
       2.1.2. Doesn’t rely on clock for synchronization so uses transitions in data. Long strings of 0’s or 1’s in data would cause problems. Uses scrambling and 2-bit header block of 10 for data block or 01 for link-level information block.
   2.2. Transaction Layer
       2.2.1. Transaction Types
           2.2.1.1. Memory = RAM, and PCIe I/O devices
           2.2.1.2. I/O = transfer in system memory map reserved for legacy PCI devices.
           2.2.1.3. Configuration = read/write configuration registers associated with I/O devices. May be locked.
           2.2.1.4. Message = address space for control signals related to interrupts, error handling, and power management.
       2.2.2. Transaction Layer Packets consist of: framing header (physical link layer), Sequence number (data link layer), header, data, TL error correcting code, data link error correcting code, framing footer (physical layer).
   2.3. Data Link Layer = ensure reliable delivery of packets.
3. Interrupts
   3.1. Why do we need them? OS handles the interface between the internals and externals of the machine, and there is a great speed disparity between CPU and I/O devices. Keyboard 100ms, disk drive 10 ms, CPU 1 ns.
   3.2. Possible methods of dealing with I/O
       3.2.1. Busy waiting = OS sits in a loop that waits for a key to be pressed.
       3.2.2. Polling = OS checks with device now and then. Less waste of CPU time, but less responsive.
       3.2.3. Interrupt = a change in program flow generated by some (external or internal event). Like a phone ringing.
   3.3. What do we need for the interrupt process to work properly?
       3.3.1. Preserve current state.
3.3.2. JSR to a preset location based on the event.
3.3.3. Should be completely invisible, so the current state should be able to be restored perfectly.
3.4. What do we need to add to the machine to support interrupts?
3.4.1. The basic FDE cycle will have to be modified to become CFDE or FDEC.
3.4.2. Need to add PSW to stack when calling.
3.4.3. We will have to add an RTI instruction to get back.
3.4.4. Need a way to enable and disable interrupts.
3.4.5. Need to have different classes of interrupts (maskable, non-maskable).
3.4.6. Need to know what to load into PC.
3.4.7. Need to know what goes into the Interrupt Service Routine.
3.5. What about multiple classes of interrupts?
3.5.1. Need to define an Interrupt Hierarchy - each needs service.
3.5.2. Need a mix of polling and interrupt vectors.
3.5.3. Use jump tables to allow flexibility and modification.
3.6. Classification of interrupts
3.6.1. Examples: Power failure, arithmetic overflow, I/O device request, OS call, page fault.
3.6.2. By timing:
   3.6.2.1. Synchronous (deterministic) = function of program and memory state (overflow, page fault).
   3.6.2.2. Asynchronous (nondeterministic) = External device or hardware malfunction (printer ready, bus error).
3.6.3. Type of user
   3.6.3.1. Normal User Request = from user program (OS Call).
   3.6.3.2. Coerced = from operating system or hardware (page fault, protection violation).
3.6.4. Masking
   3.6.4.1. User maskable = can be temporarily ignored (overflow, breakpoint).
   3.6.4.2. Nonmaskable = Must be handled (power failure, page fault, reset button).
3.6.5. Location in instructions
   3.6.5.1. Within an instruction = must be dealt with to complete instruction (page fault).
   3.6.5.2. Between instructions = not part of an instruction (I/O device request, OS Call).
3.6.6. Result
   3.6.6.1. Resume = transparently return to user process (page fault, I/O request)
   3.6.6.2. Terminate = give up and die (protection violation, power failure)