Real-Time USB Communication in the Quest Operating System

Eric Missimer, Ye Li, Richard West

Table of contents

Quest USB Subsystem

Quest:

- Boston University's in house operating system
- Developed for real-time and high-confidence systems

Major Contributions:

- Bandwidth allocation guarantees when opening connections with endpoints
- Dynamic reordering of transfer requests to avoid unnecessary rejections
- Real-time guarantees for both asynchronous and periodic transfers

Motivation - Why USB?

- Precise timing: 125 μs time granularity
- To date, USB 2.0 is the highest bandwidth I/O on SBC (BeagleBoard, PandaBoard, Raspberry Pi)
- USB 3.0 is just starting to appear on such SBC
- One common bus for I/O devices and communication
    - Towards a real-time interconnect for distributed embedded systems

USB 2.0

- Master-Slave Protocol
- Time is broken up by 125 μs chunks called micro-frames
- 8 micro-frames make up one frame (1 millisecond)
- Transactions cannot cross micro-frames
- Four types of USB transactions:
  - (1) Bulk and (2) Control - Asynchronous
  - (3) Isochronous and (4) Interrupt - Periodic
- For each micro-frame at most 80% can be used for periodic transactions

USB Transaction Types

Endpoint Type	Data Integrity Ensured	Guaranteed Transfer Rate	Example
Control	Yes	No	All Devices
Bulk	Yes	No	Mass Storage
Isochronous	No	Yes	Camera
Interrupt	Yes	Yes	Keyboard

USB 2.0 Host and Device

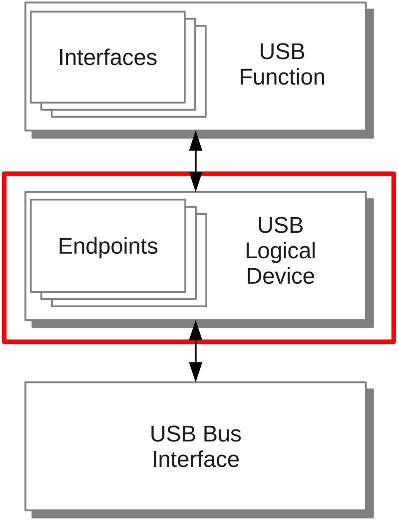
USB Host:

- Host controller makes all scheduling decisions
- More complex circuitry than a device

USB Device:

- Device contains one or more endpoints
- Each endpoint has a:
    - Number
    - Direction
    - **Transaction Type**
    - **Packet Size**
    - **Interval** (for periodic transactions, $\times 2^n$ micro-frames)

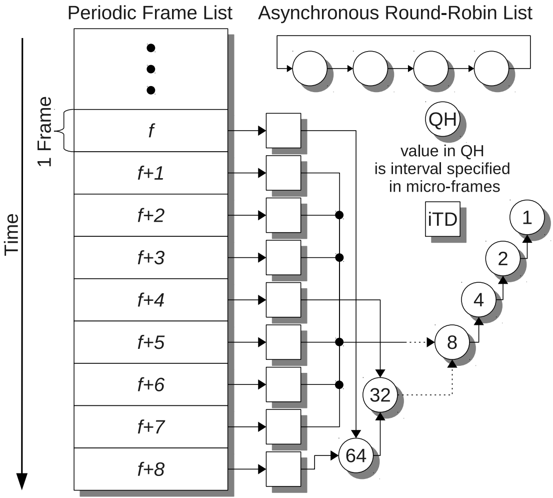
USB Device Stack



EHCI - Enhanced Host Controller Interface

- Information for performing USB transactions are contained in transaction descriptors (TDs)
- Handles asynchronous and periodic transactions differently
  - Separate scheduling data structures for each class of transactions
- Isochronous TDs (iTDs) and Queue Head (QH) plus queue TDs (qTDs)

EHCI Scheduling

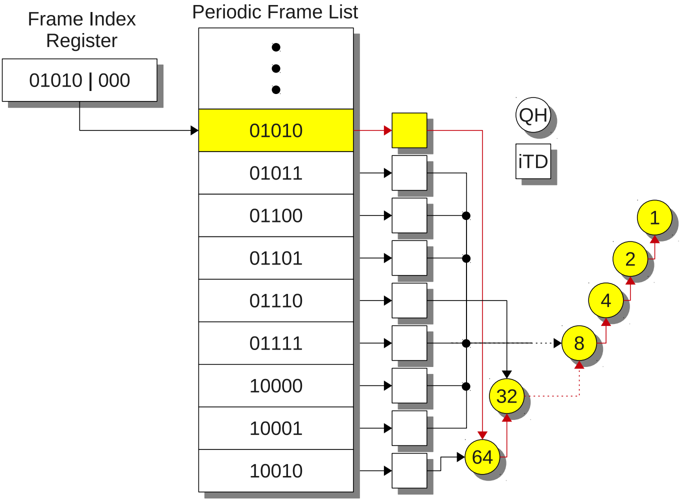


Processing TDs

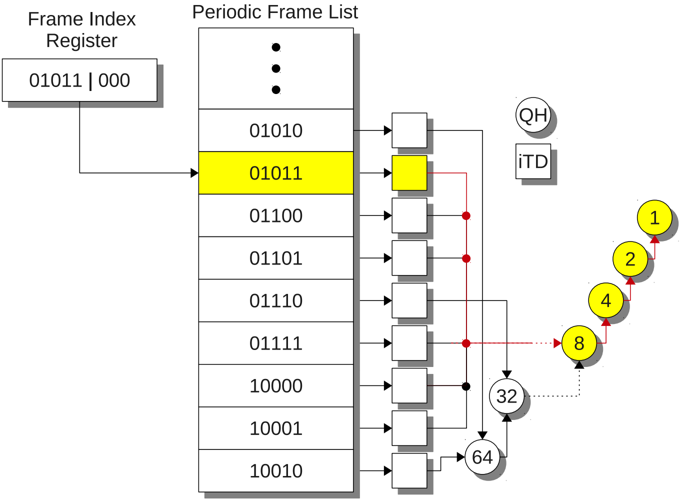
At the start of the micro-frame the host controller does the following:

- Use the frame index register to index into the periodic list
- Process the iTDs and QHs/qTDs in the periodic list pointed to by the periodic list entry
- Process the next QH in the asynchronous circular list until the end of the micro-frame

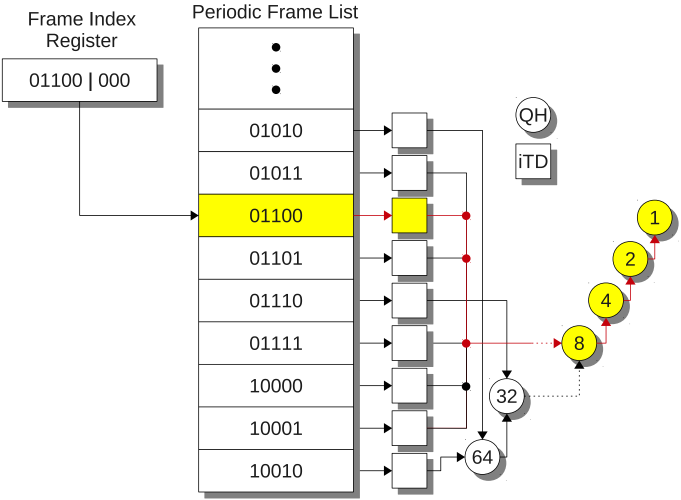
EHCI Scheduling: Example



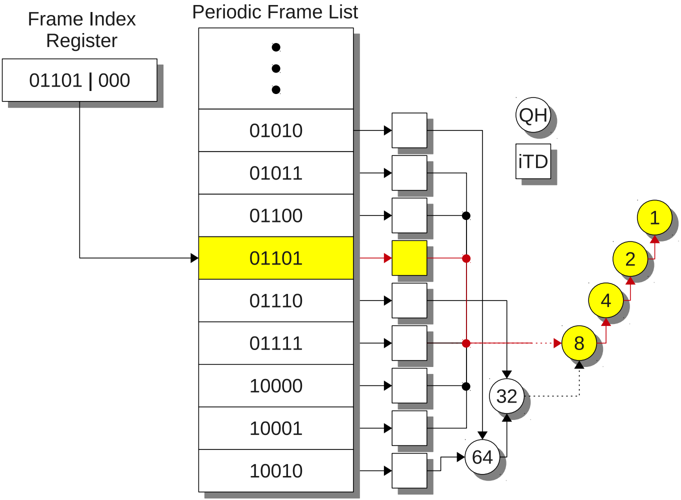
EHCI Scheduling: Example



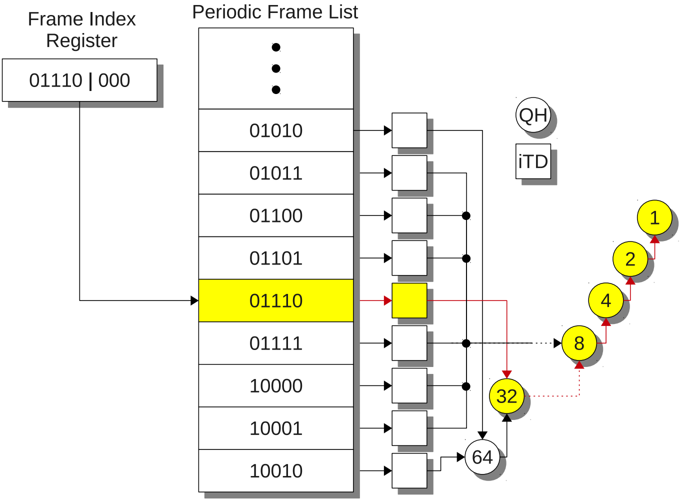
EHCI Scheduling: Example



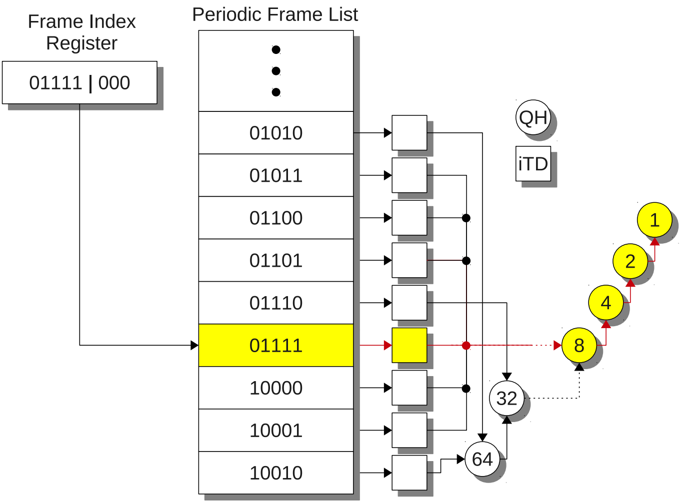
EHCI Scheduling: Example



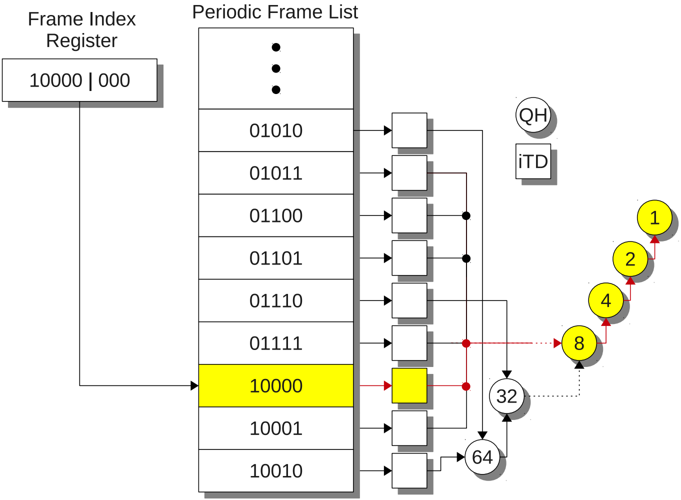
EHCI Scheduling: Example



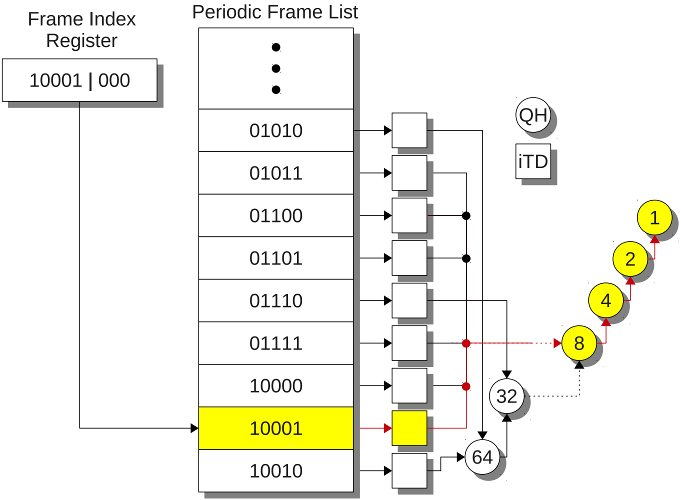
EHCI Scheduling: Example



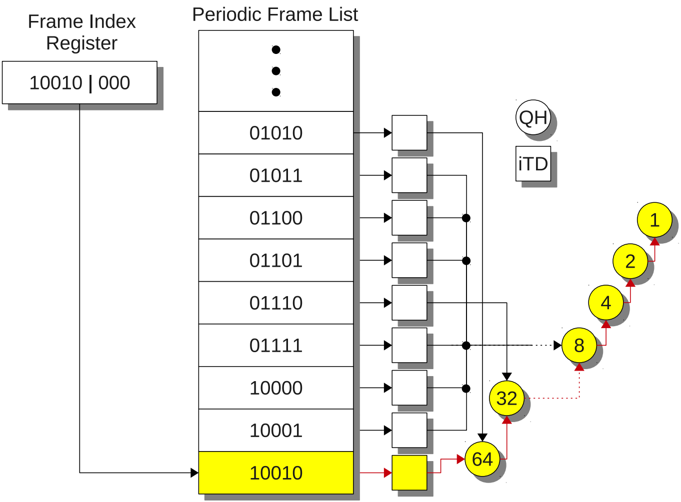
EHCI Scheduling: Example



EHCI Scheduling: Example



EHCI Scheduling: Example



USB Scheduling Problem

- Periodic USB requests can be represented as a tuple (w_i, t_i).
  - w_i time to send transaction i
  - t_i interval of transaction i
- Scheduling problem is given a set of tuples $\{(w_1, t_1), (w_2, t_2), \ldots, (w_n, t_n)\}$ is there an assignment of USB transactions to micro-frames such that no micro-frame is over-committed.
- If a request is assigned to micro-frame f it is also assigned to micro-frames $f + n * t_i, n \in \mathbb{N}$

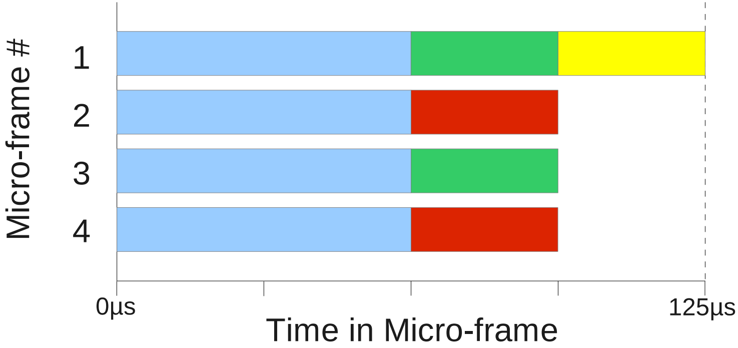
Scheduling Example

Request 1 (Tx Delay = 25%, Interval = 2)

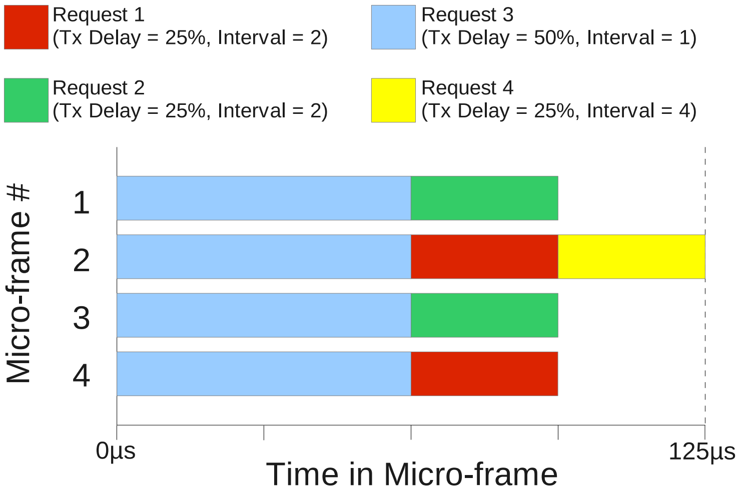
Request 2 (Tx Delay = 25%, Interval = 2)

Request 3 (Tx Delay = 50%, Interval = 1)

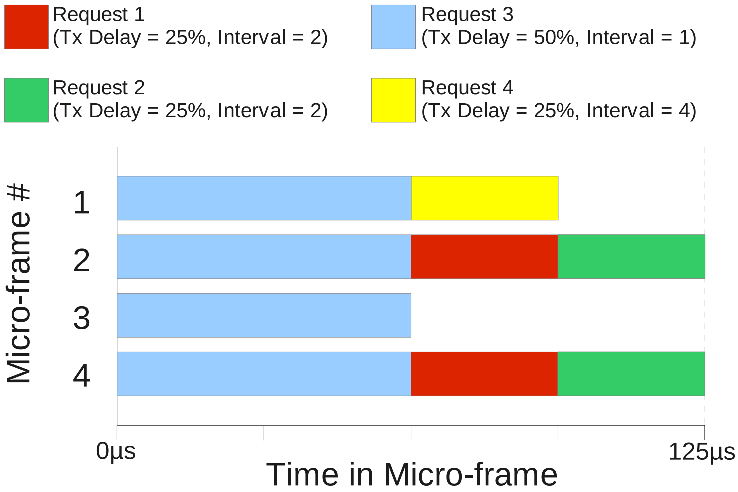
Request 4 (Tx Delay = 25%, Interval = 4)



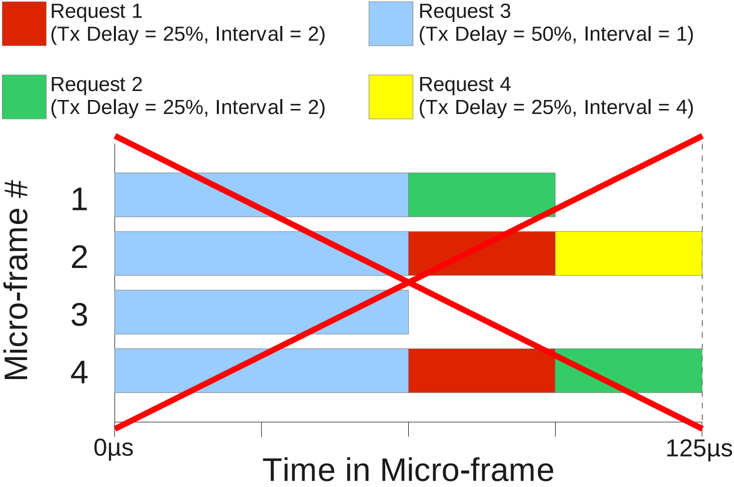
Scheduling Example



Scheduling Example



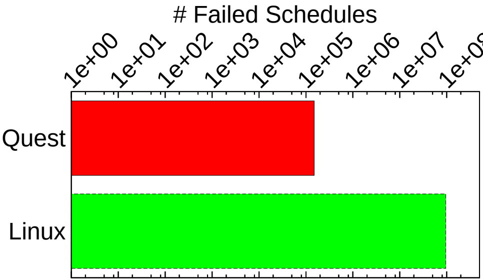
Invalid Scheduling Example



Quest USB Scheduling

Heuristic that parallels first fit decreasing algorithm for bin-packing and rate monotonic scheduling

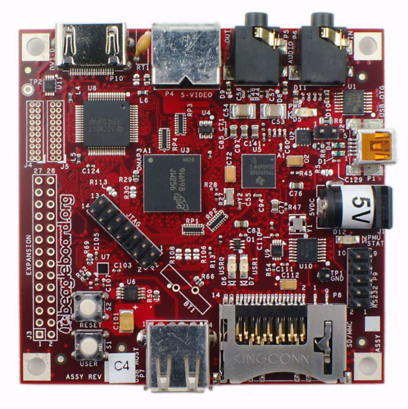
- Sort requests by interval from smallest to largest, breaking ties by sorting transmission delay from largest to smallest
- Assign sorted requests to first available micro-frame



- 1 to 5 requests
- Intervals: 2, 4, 8, 16
- Packet Sizes: 32, 64, . . . , 1024
- Quest $\approx 150,000$
- Linux $\approx 95,000,000$

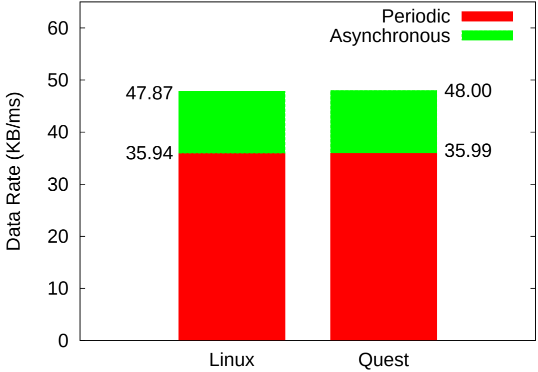
Experimental Evaluation

- Beagleboard:
  - Ångström Linux
  - Custom USB-Gadget Device Driver
- Functionally equivalent Linux and Quest device drivers



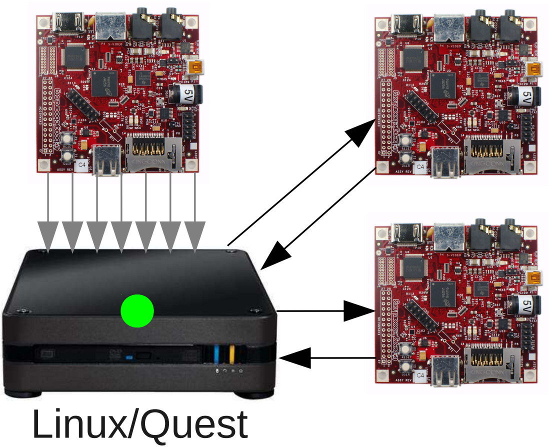
Maximum Throughput

- 9 isochronous endpoints with an interval of 1 and 9 bulk endpoints, packet size for all is 512-bytes

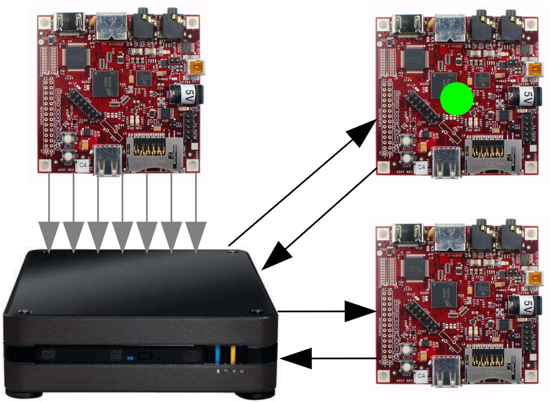


- The design of the Quest USB subsystem is not detrimental to overall throughput

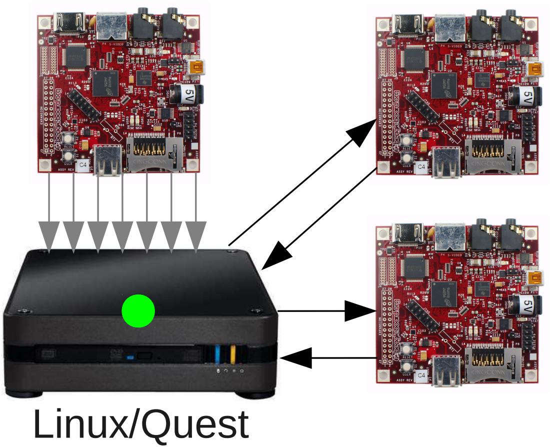
Bandwidth Allocation Experiment



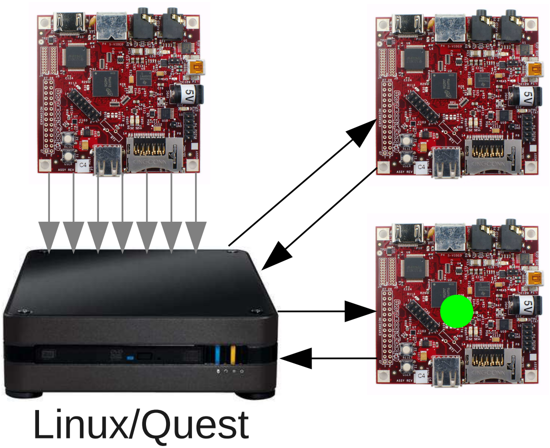
Bandwidth Allocation Experiment



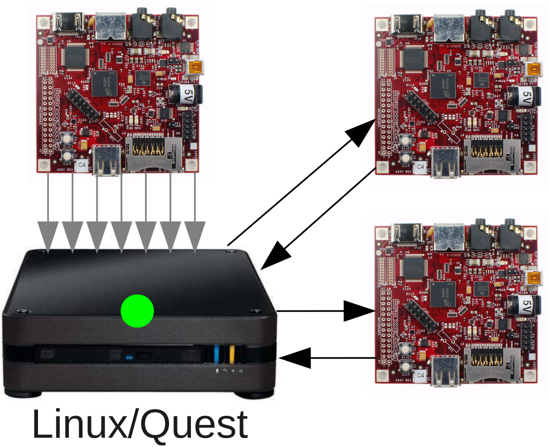
Bandwidth Allocation Experiment



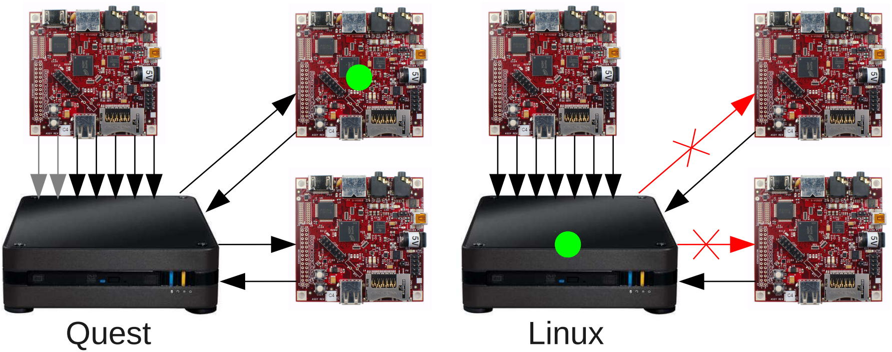
Bandwidth Allocation Experiment



Bandwidth Allocation Experiment



Bandwidth Allocation Experiment



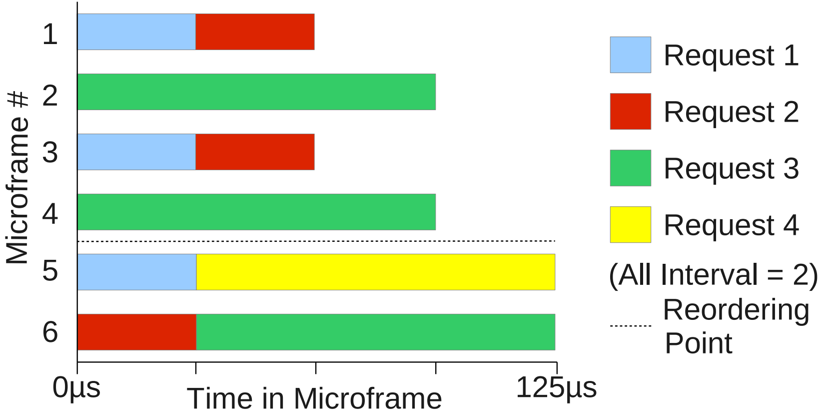
Bandwidth Allocation: Results

OS	Tokens Passed Without Third Beagleboard	Tokens Passed With Third Beagleboard	Third Beagleboard Isoc Endpoints Opened
Quest	1499	1499	5
Linux	1499	300	7

Dynamic Reordering of Transactions

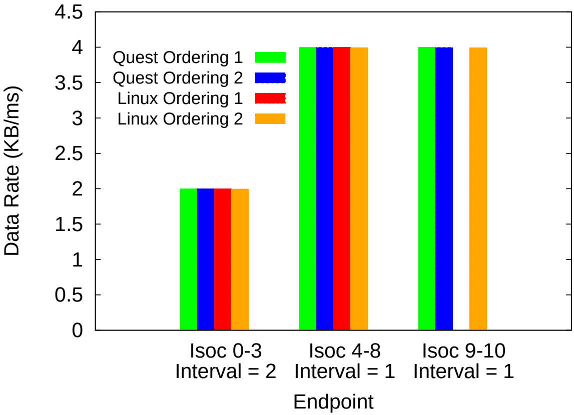
- System can reorder transactions in the periodic list to admit new USB requests that would otherwise be rejected
  - Reordering algorithm is uses heuristic for USB scheduling problem
- Transactions are reordered while maintaining endpoint constraints
- Endpoints violate their poll/push rate for one interval at most

Dynamic Reordered of Transactions: Example



Dynamic Reordered of Transactions

- First Ordering: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- Second Ordering: 4, 5, 6, 7, 8, 9, 10, 0, 1, 2, 3

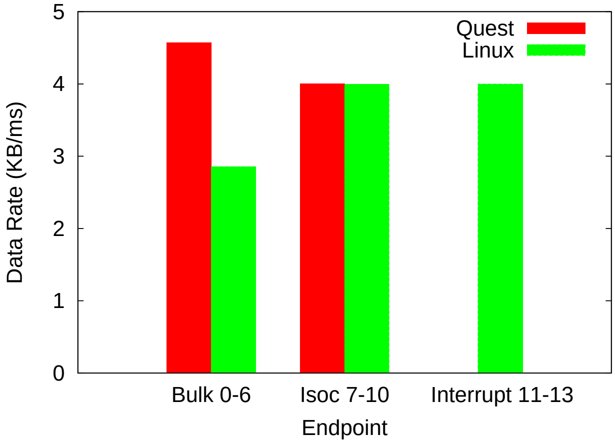


Real-Time Asynchronous

- In most systems, asynchronous transactions (Bulk & Control) are typically handled in a best effort manner
- Guaranteed at least 20% of each micro-frame
- In Quest, asynchronous transactions can be assigned an interval value just as periodic (specified by the device driver instead of the endpoint)

Real-Time Asynchronous: Experiment and Results

- Seven bulk, four isochronous and three interrupt endpoints
- All have an interval of one and a packet size of 512-bytes



Conclusions

- Heuristic algorithm that outperforms Linux's first fit approach for the USB scheduling problem
- Bandwidth allocation guarantees when opening connections with endpoints
- Dynamic reordering of transfer requests to avoid unnecessary rejections
- Real-time guarantees for both asynchronous and periodic transfers

Thank You

- Thank you for your time
- More information can be found at:
  `http://www.cs.bu.edu/~richwest/quest.html`

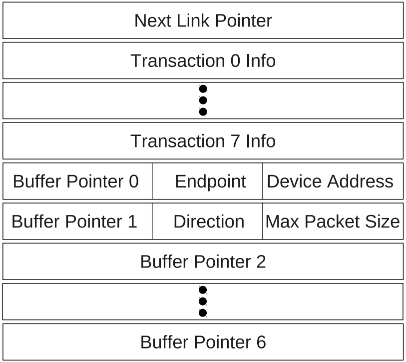
Questions?

Questions?

EHCI Transfer Descriptors

Isochronous Transfer Descriptor (iTD)

- Only used for Isochronous transactions
- Contains the information for at most 8 isochronous transactions
- Multiple iTDs for a single endpoint in the periodic list



EHCI Transfer Descriptors cont.

Queue Head (QH) and Queue Element Transfer Descriptor (qTD)

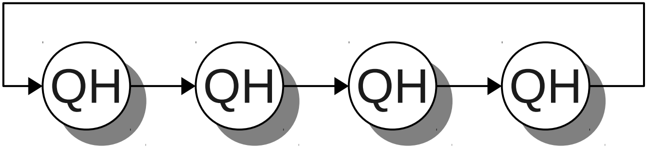
- Used for Control, Bulk and Interrupt transactions
- Contains the information for multiple transactions.
- Forms a linked list with other qTDs
- Head of the list is a QH

QH		
Next Link Pointer		
Max Packet Length	Endpoint Device	Address
Port Number	Hub Address	μFrame S-mask
Current qTD Pointer		

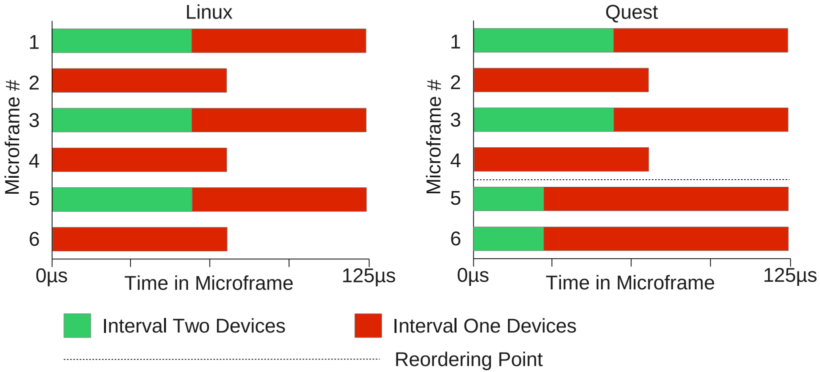
qTD	
Next qTD Pointer	
Alternate Next qTD Pointer	
Total Bytes to Transfer	Status
Buffer Pointer 0	
⋮	
Buffer Pointer 4	

Real-Time Asynchronous cont.

- One caveat: Asynchronous transactions are in a circular linked list
- We have to schedule all asynchronous transactions as having an interval equal to the smallest asynchronous interval
- Can then leave room in periodic schedule for asynchronous transfers



Dynamic Reordered of Transactions cont.



Future Directions

- USB OTG and USB device to PCI (e.g. Net2280) support
- USB 3.0 - XHCI host controller driver
- USB FPGA-based router for P2P communication over USB