ESE5320: System-on-a-Chip Architecture

Day 25: November 30, 2022

Real-Time Scheduling

Today

Real Time
- Part 1: Synchronous Reactive Model
- Part 2: Interrupts and IO
  - Polling alternative
  - Timer?
- Part 3: Resource Scheduling Graphs

Message

- Scheduling is key to real time
  - Analysis
  - Guarantees

Synchronous Circuit Model

- A simple synchronous circuit is a good "model" for real-time task
  - Run at fixed clock rate
  - Take input every cycle
  - Produce output every cycle
  - Complete computation between input and output
  - Designed to run at fixed-frequency
    - Critical path meets frequency requirement

Synchronous Reactive Model

- Discipline for Real-Time tasks
- Embodies "synchronous circuit model"

Synchronous Reactive

- There is a rate for interaction with external world (like the clock)
- Computation scheduled around these clock ticks (or time-slices)
  - Continuously running threads
  - Each thread performs action per tick
- Inputs and outputs processed at this rate
- Computation can "react" to events
  - Reactions finite and processed before next tick

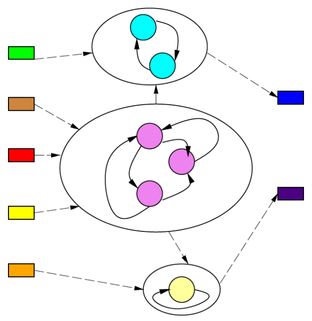
Thread Form

while (1) { tick(); }

- tick() -- yields after doing its work
  - Until next master cycle
  - May be state machine
    - May change state and have different behavior based on state
  - May trigger actions to respond to events (inputs)

7

Thread Model



8

Tick Rate

- Driven by application – demands of external control
  - Control loop 100 Hz
    - Robot, airplane, car, manufacturing plant
  - Video at 33 fps
  - Game with 20ms response
  - Router with 1ms packet latency
    - 12μs

9

Tick Rate

- Multiple rates
  - May need master tick as least-common multiple of set of interaction rates
    - …and lower freq. events scheduled less frequently
  - E.g. 100Hz control loop and 33Hz video
    - Master at 10ms
    - Schedule video over 3 10ms time-slots
      - May force decompose into tasks fit into smaller time window since must schedule events at highest frequency

10

Synchronous Reactive

- Ideal model
  - Per tick reaction (task processing) instantaneous
- Separate function from compute time
- Separate function from technology
  - Feature size, processor mapped to
- Like synchronous circuit
  - If logic correct, works when run clock slow enough
  - Works functionally when change technology
  - Then focus on reducing critical path
    - → making timing work

11

Timing and Function

- Why want to separate function from technology and timing?
- Move to slower processor(s):
  - What would happen if just moved?
  - What needs to happen?
- Move to faster processor(s):
  - What would happen if just moved?
  - What want to happen?

12

Synchronous Reactive Timing

- Once functional,
  - need to guarantee all tasks (in all states)
    - Can complete in tick time-slot
    - On particular target architecture
- Identify WCET (worst-case execution time)
  - Like critical path in FSM circuit
  - Time of task on processor target

13

Preclass 1

- Time available to process objects?

```
tick() {
   for(i=0;i<MAX_OBJECTS;i++)  {
        obj[i].inputs(); // see below
        obj[i].updatePositionState(); // 1,000 cycles
        obj[i].collide(); // 9,000 cycles
        obj[i].render();  // 1,000 cycles
      }
   updateScreen(); // takes 10 ms
}
```

14

Preclass 1

- Worst-case object processing time?

```
tick() {
   for(i=0;i<MAX_OBJECTS;i++)  {
        obj[i].inputs(); // see below
        obj[i].updatePositionState(); // 1,000 cycles
        obj[i].collide(); // 9,000 cycles
        obj[i].render();  // 1,000 cycles
      }
   updateScreen(); // takes 10 ms
}
// for object class
inputs() {
   int move=getMoveInput(); // 10
   int fire=getFireInput(); // 10
   switch (move){
     case LEFT: moveLeft(); break; // 10
     case RIGHT: moveLeft(); break; // 10
     case FORWARD: thrustIncrease(); break; // 5,000
     case BACK: thrustDecrease(); break; // 4,000
     default:
     }
   if (fire) processFire(); // 10,000
}
```

15

Preclass 1

- Maximum number of objects on single GHz processor?

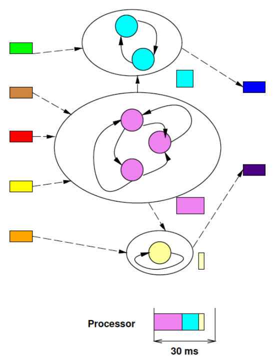
16

Synchronous Reactive Timing

- Once functional,
  - need to guarantee all tasks (in all states) can complete in tick time-slot
  - On particular target architecture
- Identify WCET
  - Like critical path in FSM circuit
  - Time of task on processor target
- Schedule onto platform
  - Threads onto processor(s)

17

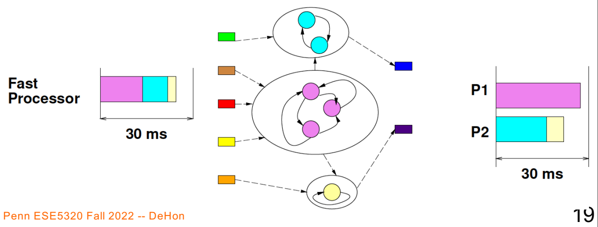
Threads Mapped to Processor



18

Platforms

- Platform 1: fast processor
- Platform 2: many slow processors



Synchronous Reactive Model

- Discipline for Real-time tasks
- Embodies the “synchronous circuit model”
  - Master clock rate
  - Computation decomposed per clock
  - Functionality assuming instantaneous compute
  - On platform, guarantee runs fast enough to complete critical path at “clock” rate

Interrupts and IO

Part 2

Interrupt

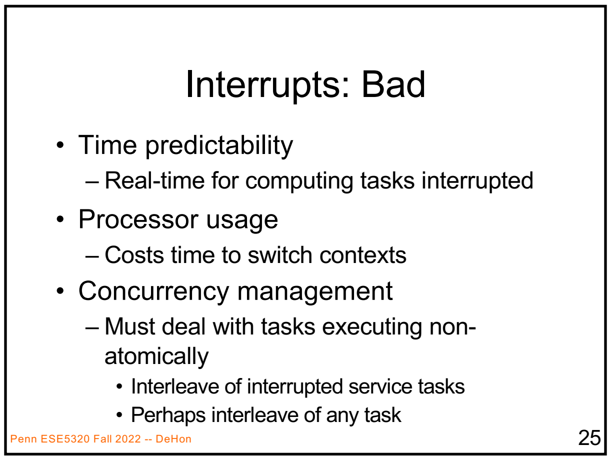
- External event that redirects processor flow of control
- Typically forces a thread switch
- Common for I/O, Timers
  - Indicate a need for attention

Interrupts

- Why would we use interrupts for I/O?

Interrupts: Good

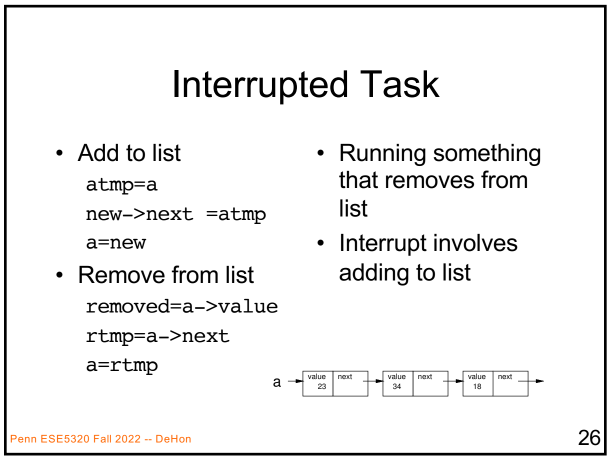
- Allow processor to run some other work
- Infrequent, irregular task service with low response service latency
  - Low latency
  - Ok when low throughput inputs
    - So infrequent interrupts…



Interrupts: Bad

- Time predictability
  - Real-time for computing tasks interrupted
- Processor usage
  - Costs time to switch contexts
- Concurrency management
  - Must deal with tasks executing non-atomically
    - Interleave of interrupted service tasks
    - Perhaps interleave of any task

25



Interrupted Task

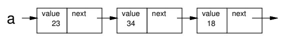
- Add to list

```
atmp=a
new->next =atmp
a=new
```

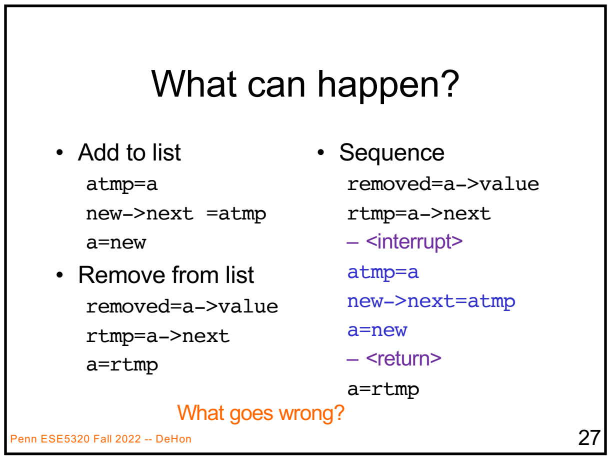
- Remove from list

```
removed=a->value
rtmp=a->next
a=rtmp
```

- Running something that removes from list
- Interrupt involves adding to list



26



What can happen?

- Add to list

```
atmp=a
new->next =atmp
a=new
```

- Remove from list

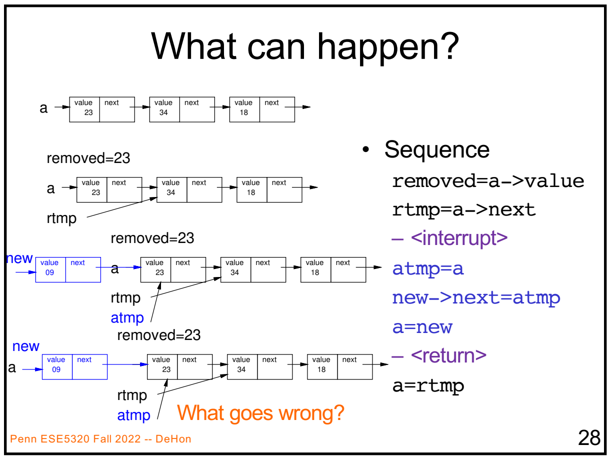
```
removed=a->value
rtmp=a->next
a=rtmp
```

- Sequence

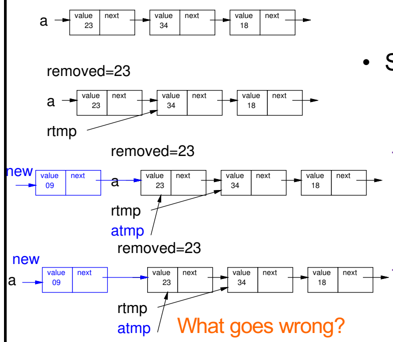
```
removed=a->value
rtmp=a->next
– <interrupt>
atmp=a
new->next=atmp
a=new
– <return>
a=rtmp
```

What goes wrong?

27



What can happen?

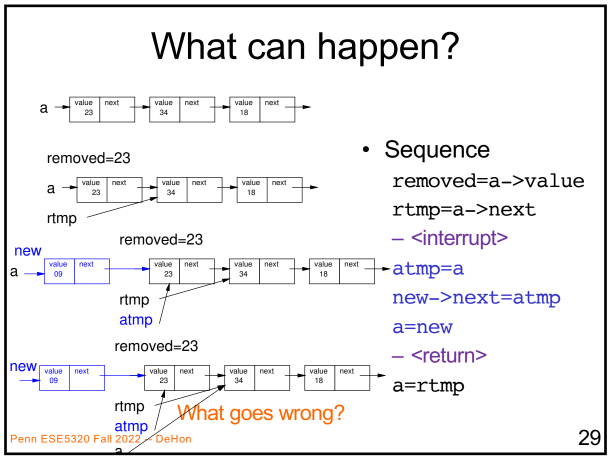


- Sequence

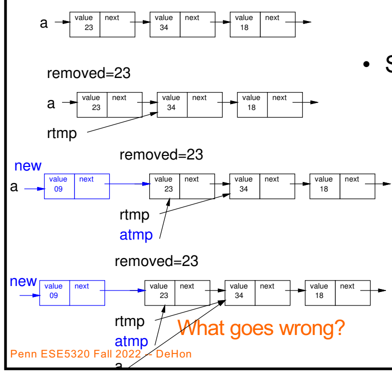
```
removed=a->value
rtmp=a->next
– <interrupt>
atmp=a
new->next=atmp
a=new
– <return>
a=rtmp
```

What goes wrong?

28



What can happen?

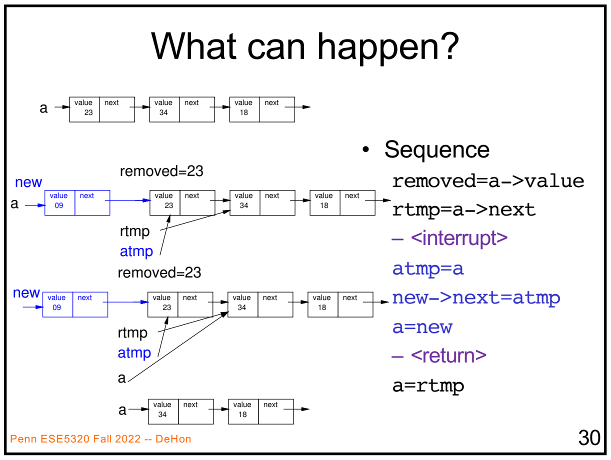


- Sequence

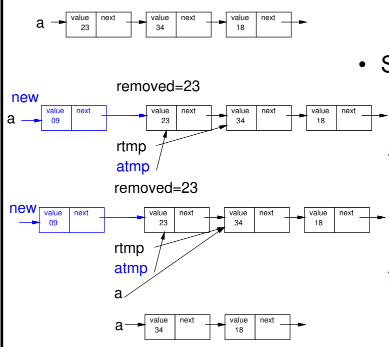
```
removed=a->value
rtmp=a->next
– <interrupt>
atmp=a
new->next=atmp
a=new
– <return>
a=rtmp
```

What goes wrong?

29



What can happen?

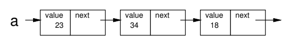


- Sequence

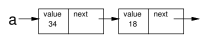
```
removed=a->value
rtmp=a->next
– <interrupt>
atmp=a
new->next=atmp
a=new
– <return>
a=rtmp
```

30

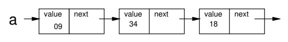
What can happen?



Got:



Intended:



- Sequence

```
removed=a->value
rtmp=a->next
– <interrupt>
atmp=a
new->next=atmp
a=new
– <return>
a=rtmp
```

31

Interrupts: Bad

- Time predictability
  - Real-time for computing tasks interrupted
- Processor usage
  - Costs time to switch contexts
- Concurrency management
  - Must deal with tasks executing non-atomically
    - Interleave of interrupted service tasks
    - Perhaps interleave of any task

32

Polling Discipline

- Alternate to I/O interrupts
- Every I/O task is a thread
- Budget time and rate it needs to run
  - E.g. 10,000 cycles every 5ms
  - Likely tied to
    - Buffer sizes
    - Response latency
- Schedule I/O threads as real-time tasks
  - Some can be DMA channels

33

IO Thread

while (1) { process_input(); }

- Like tick() -- yields after doing its work
  - Wait for next master cycle

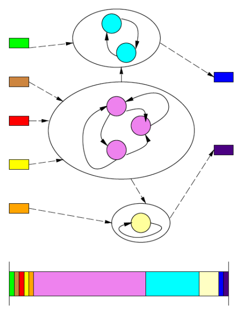
34

Preclass 2

- Input at 100KB/s
- 30ms time-slot window
- Size of buffer?
- 100 cycles/byte, GHz processor – runtime of service routine?
  - Fraction of processor capacity?

35

Scheduling I/O Tasks



36

Timer Interrupts

- Why do we have timer interrupts in conventional operating systems?
  - E.g. in linux?

Timer Interrupts

- Best effort tasks (i.e. non-real-time tasks)
  - Have no guarantee to finish in bounded time
  - Timer interrupts necessary
    - to allow other threads to run
    - fairness
    - to switch to real-time service tasks
- Need timer interrupts if need to share processor with best-effort and real-time threads
  - **Alternate:** Easier to segregate real-time and best-effort threads onto different processors

Timer Interrupts?

- Bounded-time tasks
  - E.g. reactive tasks in real-time
  - Task has guarantee to release processor within time window
  - **Not** need timer interrupts to regain control from task
  - (Maybe use deadline operations [Day24] for timer)

Greedy Strategy

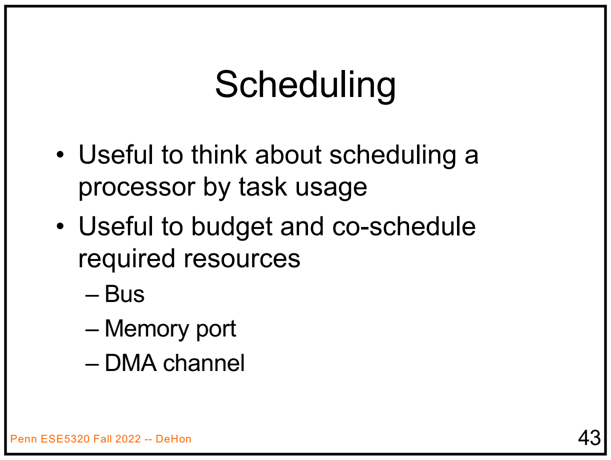
- Schedule real-time tasks
  - Scheduled based on worst-case, so may not use all time allocated
- Run best-effort tasks at end of time-slice after complete real-time tasks
  - Timer-interrupt to recover processor in time for start of next scheduling time slot
- (adds complexity)

Real-Time Tasks

- Interrupts less attractive
  - More disruptive
- Scheduled polling better predictability
- Fits with Synchronous Reactive Model

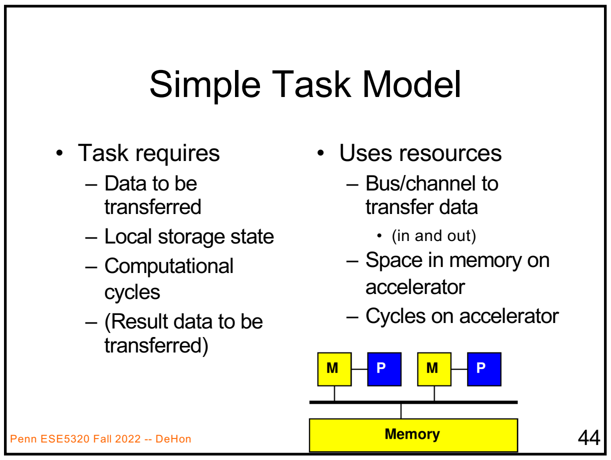
Resource Scheduling Graphs

Part 3



Scheduling

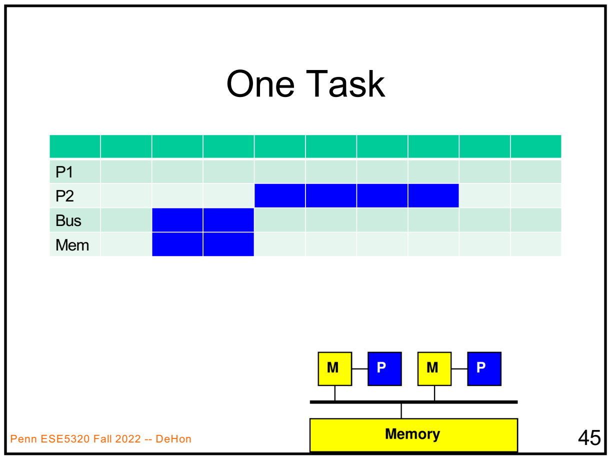
- Useful to think about scheduling a processor by task usage
- Useful to budget and co-schedule required resources
  - Bus
  - Memory port
  - DMA channel



Simple Task Model

- Task requires
  - Data to be transferred
  - Local storage state
  - Computational cycles
  - (Result data to be transferred)
- Uses resources
  - Bus/channel to transfer data
    - (in and out)
  - Space in memory on accelerator
  - Cycles on accelerator

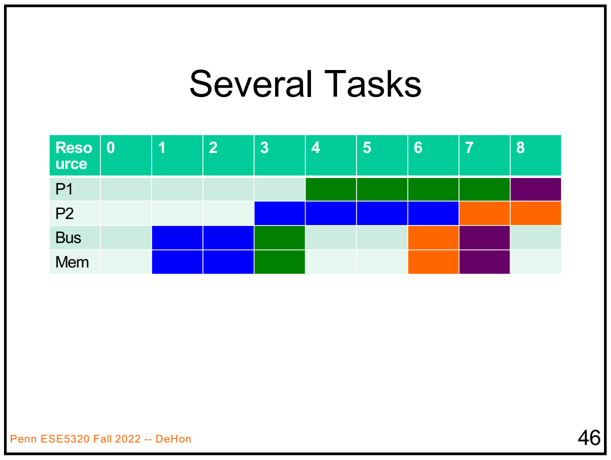
M
P
M
P
Memory



One Task

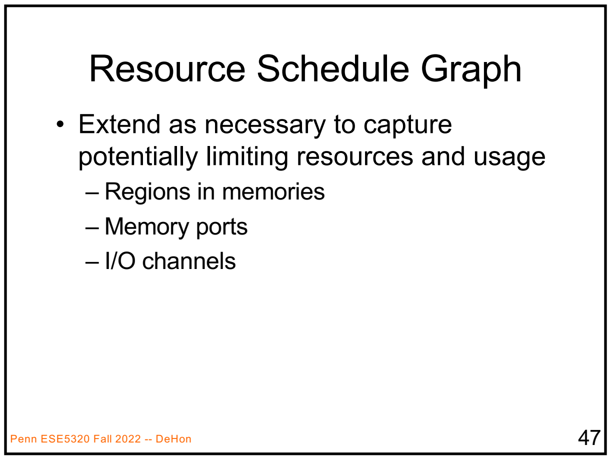
P1									
P2				🟦	🟦	🟦	🟦		
Bus		🟦	🟦						
Mem		🟦	🟦						

M
P
M
P
Memory



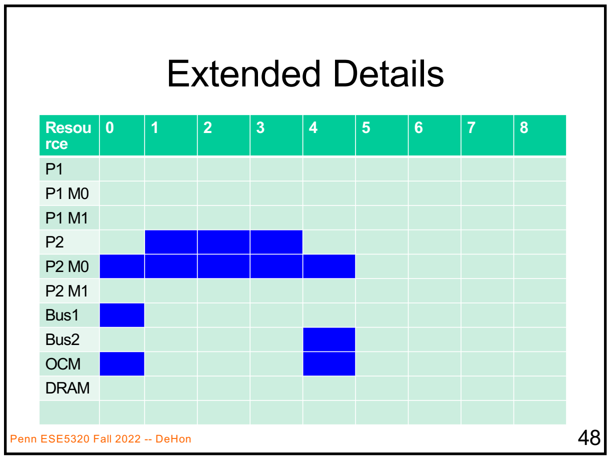
Several Tasks

Resource	0	1	2	3	4	5	6	7	8
P1					🟩	🟩	🟩	🟩	🟪
P2				🟦	🟦	🟦	🟦	🟧	🟧
Bus		🟦	🟦	🟩			🟧	🟪	
Mem		🟦	🟦	🟩			🟧	🟪	



Resource Schedule Graph

- Extend as necessary to capture potentially limiting resources and usage
  - Regions in memories
  - Memory ports
  - I/O channels



Extended Details

Resource	0	1	2	3	4	5	6	7	8
P1									
P1 M0									
P1 M1									
P2		🟦	🟦	🟦					
P2 M0	🟦	🟦	🟦	🟦	🟦				
P2 M1									
Bus1	🟦								
Bus2					🟦				
OCM	🟦				🟦				
DRAM									

Several Tasks

Resource	0	1	2	3	4	5	6	7	8
P1									
P1 M0									
P1 M1									
P2		🟦	🟦	🟦	🟧	🟧	🟧		
P2 M0	🟦	🟦	🟦	🟦	🟦				
P2 M1		🟧	🟧	🟧	🟧	🟧	🟧	🟧	
Bus1	🟦	🟧	🟧						
Bus2					🟦				
OCM	🟦				🟦				
DRAM		🟧	🟧					🟧	

Approach

- Ideal/initial – look at processing requirements
  – Resource bound on processing
- Look for bottlenecks / limits with Resource Bounds independently
  – Add buses, memories, etc.
- Plan/schedule with Resource Schedule Graph

Preclass 3a

- Resource Bound
  – Data movement over bus?
  – Compute on 2 processors?
  – Compute on 2 processors when processor must wait while local memory is written?

Task	Data (bytes)	Compute cycles	Data+Compute Work
A	1600	1600	
B	200	600	
C	800	3200	
D	200	600	
E	400	400	

Resource Bound wait Transfer

- Total processor cycles when processor must idle during transfer
  – $\text{Cycles}_{\text{proc}} = \sum(Comp[i] + Bytes[i])$
- $\text{RB}_{\text{proc}} = (\text{Cycles}_{\text{proc}})/2$
- $\text{RB}_{\text{bus}} = \sum(Bytes[i])$
- $\text{RB} = \max(\text{Rb}_{\text{bus}}, \text{RB}_{\text{proc}})$

Preclass 3b Schedule

- Processor wait for data load

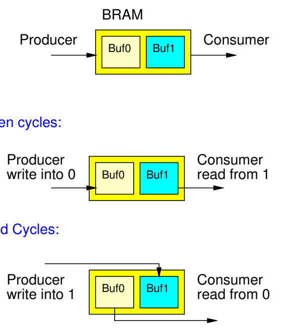
200 cycle intervals

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	3
P1																																
P2																																
B																																

Double Buffering

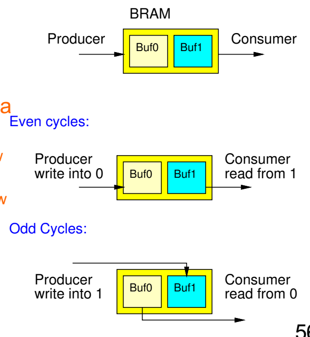
- Common trick to overlap compute and communication
- Reserve two buffers input (output)
- Alternate buffer use for input
- Producer fills one buffer while consumer working from the other
- Swap between tasks
- Tradeoff memory for concurrency
- Sub-buffers in Vitis clEnqueueMigrateObjects

Double Buffer



Double Buffer Schedule

- How impact schedule?
  - When can move data into buffer?
    - Hint: think about how impact preclass 3b schedule? What new freedom have?
  - Impact on use of processor?



Preclass 3c Schedule

- Double Buffer

200 cycle intervals

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
P1																																
P2																																
B																																

Minimum local memory space required?

Resource Schedule Graphs

- Useful to plan/visualize resource sharing and bottlenecks in SoC
- Supports scheduling
- Necessary for real-time scheduling

Big Ideas:

- Scheduling is key to real time
  - Analysis, Guarantees
- Synchronous Reactive
  - Scheduling worst-case tasks “reactions” into master time-slice matching rate
  - Separate function from timing
- Schedule I/O with polling threads
  - Avoid interrupts
- Schedule dependent resources
  - Buses, memory ports, memory regions…

Admin

- Feedback
- Wolf Lecture Wednesday at 3pm
  - Tsu-Jae King Liu
  - Sustaining the Semiconductor Revolution
- Reading for Monday online
- P4 due Friday