ESE5320: System-on-a-Chip Architecture

Day 27: December 6, 2023
Real Time
(Optional)

Penn

Today

Real Time

- Part 1: Demands
- Part 2: Challenges
  - Algorithms
  - Architecture
- Part 3: Disciplines to achieve

Message

- Real-Time applications demand different discipline from best-effort tasks
- Look more like synchronous circuits
- Can sequentialize, like processor
  - But must avoid/rethink typical general-purpose processor common-case optimizations

Real Time

- “Real” – refers to physical time
  - Connection to Real or Physical World
- Contrast with “virtual” or “variable” time
- Handles events with absolute guarantees on timing

Real-Time Tasks

- What timing guarantees might you like for the following tasks?
  - Turn steering wheel on a drive-by-wire car
    - Delay to recognized and car turns
  - Self-driving car detects an object in its path
    - Delay from object appearing to detection
  - Pacemaker stimulates your heart
  - Video playback (frame to frame delay)

Real-Time Guarantees

- Attention/processing within fixed interval
  - Sample new value every XX ms
  - Produce new frame every 30 ms
  - Both: schedule to act and complete action
- Bounded response time
  - Respond to keypress within 20 ms
  - Detect object within 100 ms
  - Return search results within 200 ms

Computer Response

- What do these things indicate?
  - When will the computer complete the task?


https://en.wikipedia.org/wiki/File:Windows_8_%2B_10_wait_cursor.gif
https://en.wikipedia.org/wiki/File:WaitCursor-300p.gif

7

Real-Time Response

- What if your car gave you a spinning wait wheel for 5 seconds when you
  - Turned the wheel?
  - Stepped on the brakes?

8

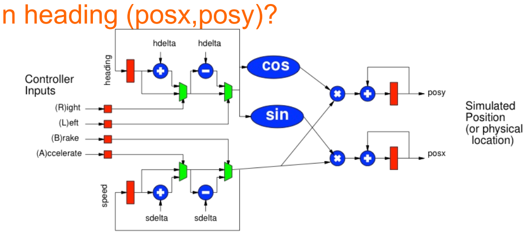
Synchronous Circuit Model

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

9

Preclass 2

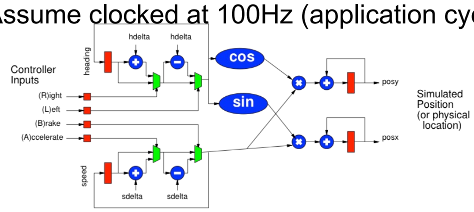
- (Circuit) Cycle time could operate?
- Assume clocked at 100Hz (application cycle)
- Worst-case delay from (L)eft press to change in heading (posx,posy)?

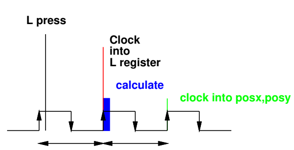


10

Preclass 2

- Assume clocked at 100Hz (application cycle)





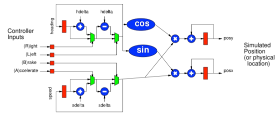
11

Historically

- Real-Time concerns grew up in EE
  - Because an analog circuit was the only way could meet frequency demands
  - …later a dedicated digital circuit…
- Applications
  - Signal processing, video, control, …

12

Technological Change



- Area units for spatial design shown (preclass 2c)
- Fraction of processor capacity required (Prelcass 2d)
- Why might prefer using a processor to using the spatial circuit?
  - Hint: What does preclass 2c,d suggest?

13

Performance Scaling

- As circuit speeds increased
  - Can meet real-time performance demands with heavy sequentialization
- Circuit and processor clocks
  - from MHz to GHz
- Many real-time task rates unchanged
  - 44KHz audio, 33 frames/second video
- Even 100MHz processor
  - Can implement audio in a small fraction of its computational throughput capacity

14

HW/SW Co-Design

- Computer Engineers – know can implement anything as hardware or software
- Want freedom to move between hardware and software to meet requirements
  - Performance, costs, energy

15

Real-Time Challenge

- Meet real-time demands / guarantees
  - Economically using programmable architectures
- Sequentialize and share resources with deterministic, guaranteed timing
- Spatial (all hardware, HLS synthesized) implementations are good at meeting real-time guarantees, but may be bigger than necessary

16

Part 2

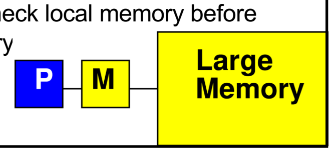
CHALLENGES

17

Day 3

Processor Data Caches

- Traditional Processor Data Caches are a heuristic instance of this
  - Add a small memory local to the processor
    - It is fast, low latency
  - Store anything fetched from large/remote memory in local memory
    - Hoping for reuse in near future
  - On every fetch, check local memory before go to large memory



18

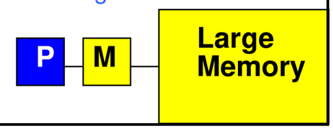
Day 3

Processor Data Caches

- Demands more than a small memory
  - Need to sparsely store address/data mappings from large memory
  - Makes more area/delay/energy expensive than just a simple memory of capacity
- Don't need explicit data movement
- Cannot control when data moved/saved
  - Bad for determinism
- Limited ability to control what stays in small memory simultaneously

Processor Data Caches

- Traditional Processor Data Caches are a heuristic instance of this
  - Store anything fetched from large/remote memory in local memory
    - Hoping for reuse in near future
  - On every fetch, check local memory before go to large memory
  - Stall processor while waiting for data



Preclass 3: Processor Cache Timing

- Assume
  - cache miss (go to large memory) takes 10 cycles
  - Cache hit (small memory) takes 1
  - Start with empty cache
- Due to memory delay, how long to execute:

```
b=a[0]+a[1];
c=a[1]+a[2];
d=a[2]+a[0];
```

```
b=a[i]+a[j];
c=a[k]+a[l];
d=a[m]+a[n];
```


Scratchpad

- Recall, scratchpad memory
  - Small
  - Explicitly managed (not dynamic like cache)
- If move (DMA) data to scratchpad memory, would be deterministic

```
b=a[0]+a[1];
c=a[1]+a[2];
d=a[2]+a[0];
```

```
b=a[i]+a[j];
c=a[k]+a[l];
d=a[m]+a[n];
```


Observe

- Instructions on "General Purpose" processors take variable number of cycles

Preclass 4

- How many cycles?
  - sin, cos 100 cycles each
  - Assignments take 1 cycle

```
old_sh=sh; old_ch=ch;
  if (!left || !right)
       {sh=old_sh;ch=old_ch;}
  else
       {sh=sin(heading);
        ch=cos(heading);}
```

Preclass 5

- How many cycles?

```
sum=0;
for (i=0;i<32;i++) {
    sum+=(0-(b%2)) & a;
    b=b>>1;
    a=a<<1;
    }
```


Preclass 5

- How many cycles?

```
sum=0;
for (;b!=0;b=b>>1)  {
     if (b%2==1)
        sum+=a;
     a=a<<1;
}
```


Observe

- Data-dependent branching, looping
  - Means variable time for operations

Two Challenges

1. Architecture – Hardware have variable (data-dependent) delay
   - Esp. for General-Purpose processors
     - Instructions take different number of cycles
2. Algorithm – computational specification have variable (data-dependent) operations
   - Different number of instructions

$$Time = \sum_i Cycles(i)$$

Algorithm

- What programming constructs are data-dependent (variable delay)?

Programming Constructs

- Conditionals: if/then/else
- Loops without compile-time determined bounds
  - While with termination expressions
  - For with data-dependent bounds
- Data-dependent recursion
- Interrupts
  - I/O events, time-slice
- Note: 1^{st} three were issue for HLS
  - For same reason – how did we address?

Hardware Architecture

- Some typical (4710,5710) processor "optimizations" can cause variable delay
  - Caches
  - Branch prediction
  - Common-case optimizations
  - Pipeline stalls
  - Speculative issue

Part 3

DISCIPLINES TO ACHIEVE REAL-TIME

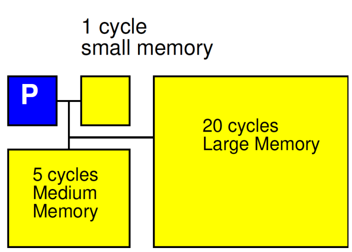
What can we do to make architecture more deterministic?

- Explicitly managed memory
- Eliminate Branching (too severe?)
- Unpipelined processors
- Fixed-delay pipelines
  - Offline-scheduled resource sharing
  - Multi-threaded
- Deadlines

Explicitly Managed Memory

- Make memory hierarchy visible
  - Use Scratchpad memories instead of caches
- Explicitly move data between memories
  - E.g. movement into local memory
- Already do for Register File in Processor
  - Load/store between memory and RF slot
  - …but don't do for memory hierarchy

Explicitly Managed Memory



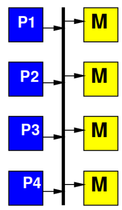
Offline Schedule Resource Sharing

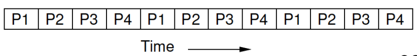
- Don't arbitrate
- Decide up-front when each shared resource can be used by each thread or processor
  - Simple fixed schedule
  - Detailed Schedule
- What
  - Memory bank, bus, I/O, network link, …

Time-Multiplexed Bus

Fixed by hardware master

- 4 masters share a bus
- Each master gets to make a request on the bus every 4th cycle
  - If doesn't use it, goes idle



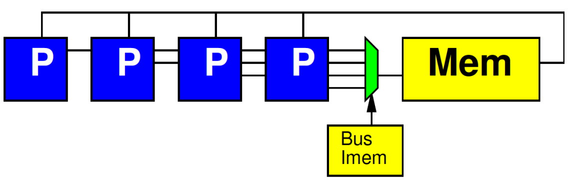


Time-Multiplexed Bus

- Regular schedule
- Fixed bus slot schedule of length N > masters
  - (probably a multiple)
- Assign owner for each slot
  - Can assign more slots to one
- E.g. N=8, for 4 masters
  - Schedule (1 2 1 3 1 2 1 4)

Fully Scheduled

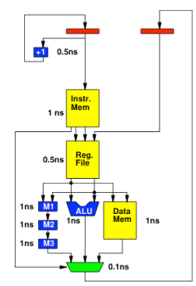
- At extreme, fully schedule which tasks gets resource on each cycle



Simple Deterministic Processor with Multiplier

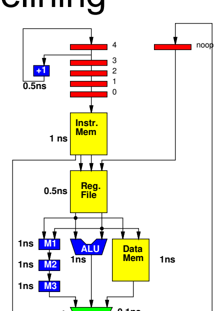
- No branching
- Unpipelined
- Every operation completes in fixed time

- Cycle time as shown?
- Retimed cycle time?
- What's inefficient about this design?



Simple Deterministic Processor with some Pipelining

- No branching
- Every operation completes in fixed time
- Retimed cycle time?
  - Hint what are cycles?
- How pipelines added change behavior?
  - Hint: what is sequence of addresses into Instr. Mem?



Deadline Instruction

- Deal with algorithmic (branching) variability
- Set a hardware counter for thread
- Decrement counter on each cycle
- Demand counter reach 0 before thread allowed to continue at deadline instruction
  - Stall if get there early
  - Similar to flip-flop on a logic path
    - Wait for clock edge to change or sample value
- Model: fixed execution time

WCET

- WCET – Worst-Case Execution Time
- Analysis when working with algorithms and architectures with data-dependent delay
  - Need to meet real time
  - Calculate the worst-case runtime of a task
    - Like calculating the critical path (but harder)
    - Worst-case delay of instructions
    - Worst-case path through code
    - Worst-case # loop iterations
  - Rationale for setting Deadlines
    - (like a cycle time)

64

Deterministic Pipelines

- Not how ARM, Intel (4710, 5710) processor are pipelined
- Those include operations that make timing variable
  - dynamic data hazards, branch speculation
- Here, data becomes available after a predictable time
- Branches take effect at a fixed time
  - Likely delayed
- Schedule to delays to get correct data

65

Different Goals

Real-Time

- Willing to recompile to new hardware
- Want time on hardware predictable
- Willing to schedule for delays in particular hardware

General Purpose/Best Effort

- ISA fixed
- Want to run same assembly on different implementations
- Tolerate different delays for different hardware
- Run faster on newer, larger implementations

66

SoC Opportunity

- Can choose which resources are shared
- Can dedicate resources to tasks
- Isolate real-time tasks/portions of tasks from best-effort
  - Separate hardware/processors
  - Separate memories, network

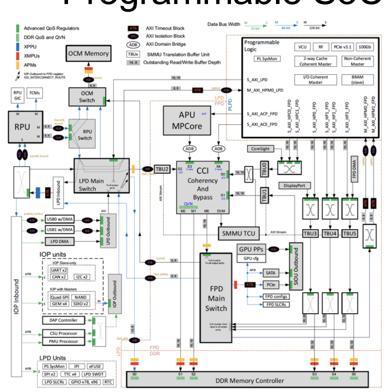
67

UltraScale+ Zynq

- Has 2 "Real-Time Processor"
  - ARM Cortex-R5
    - 32b (vs. 64b for A53 APU processor)
    - ARMv7-R (vs. ARMv8)
    - Single ALU, dual issue
    - Branch prediction
- Explicitly managed scratchpads
    - Tightly-Coupled Memories
    - On-Chip Memory (OCM)

68

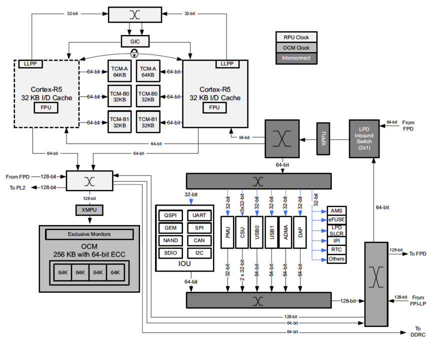
Programmable SoC



UG1085 Xilinx UltraScale Zynq TRM (p27)

69

RPU Subsystem



UG1085
Xilinx
UltraScale
Zynq
TRM
(p65)

Big Ideas:

- Real-Time applications demand different discipline from best-effort tasks
- Look more like synchronous circuits and hardware discipline
- Avoid or use care with variable delay programming constructs
- Can sequentialize, like processor
  - But must avoid/rethink typical processor common-case optimizations
  - Offline calculate static schedule for computation and sharing
    - Instead of dynamic arbitration, stalls

Admin

- Feedback
- Project due Monday 12/11
- Signup for Demo Day
  - Will invite signup on Ed Discuss
- Final: Tuesay, Dec. 19, 3pm, Moore 216
- Review: watch Ed Discuss
- No DQ for today