Time: Its Measurement and Use

Implementing Real Time Software for
Control of Mechanical Systems

What’s Time?

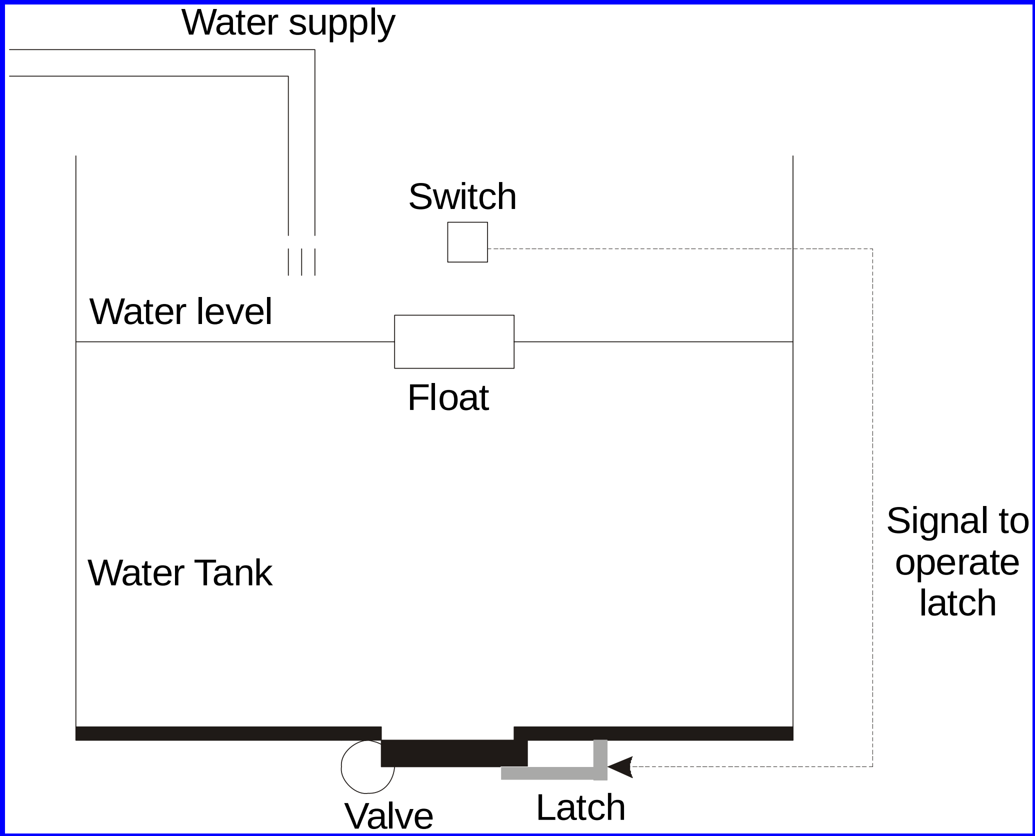
- A constant frequency oscillator
  - Don’t ask what “constant frequency” means – the discussion gets circular!
- Small amplitude pendulum, mass-spring are best known
  - Read *Longitude* by Sobel
- Two key design problems:
  - Keeping the frequency constant (age, temperature)
  - Getting energy to the oscillator (escapement)

Limit Cycle Oscillators

- Harmonic oscillators come to mind to get constant frequency (pendulum, mass-spring, inductor-capacitor)
  - However, even if "perfect" (*i.e.,* no friction) no energy can be removed for observation!
- Limit cycle oscillators are needed for timekeeping
  - Constant input (power source)
  - Oscillatory output, amplitude independent of initial condition
  - Must be nonlinear

Limit Cycle Examples

- Constant inflow
- Valve is spring-loaded
- Float touches switch
- Valve opens
- Tank empties
- Valve closes
- Cycle repeats

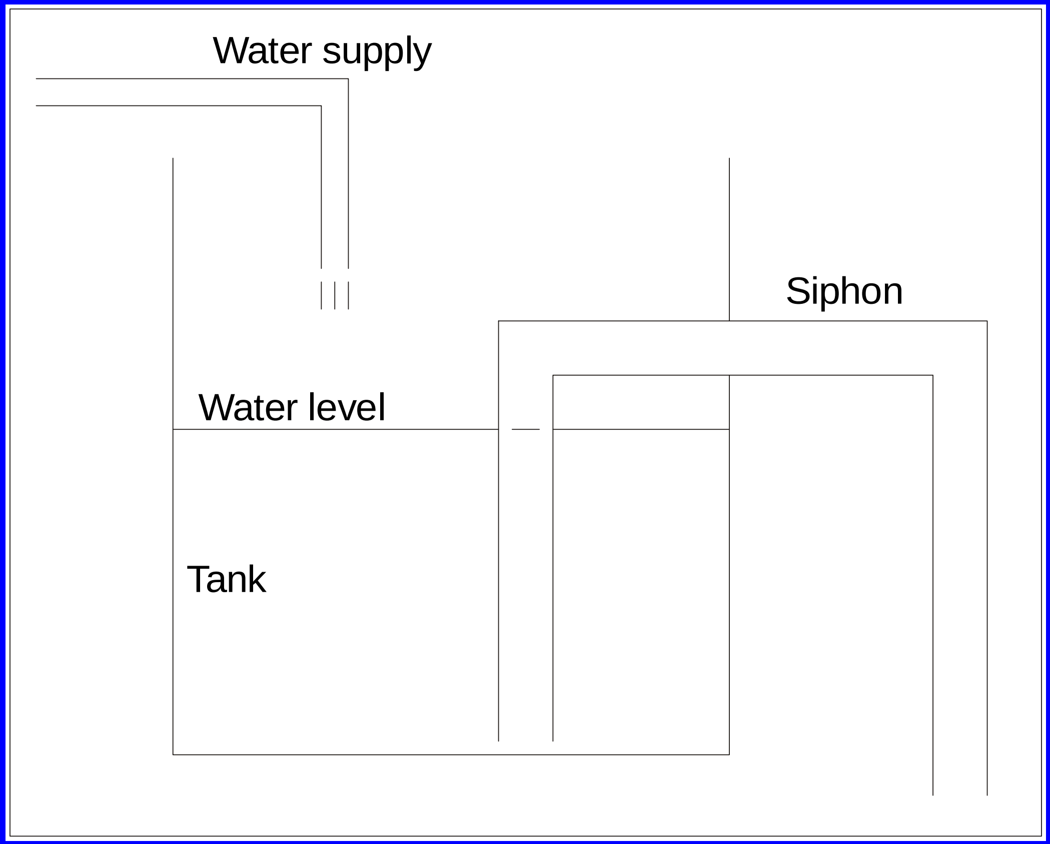


Relaxation Oscillator

- This is a relaxation oscillator
- Each cycle is a “tick”
- Timing consistency depends on water supply being constant
- Emptying must be fast relative to cycle time to not affect timing

Another Version

- Water level rises
- Floods siphon
- Tank empties
- Cycle repeats



Simpler, But Less Control

- Emptying speed of siphon is slower
- Still requires constant inflow rate for accuracy
- Resemblance to the common household appliance "invented" by the late 19th century Englishman Thomas Crapper is not a coincidence (see http://www2.exnet.com/1995/11/01/science/science.html for an interesting discussion of the surrounding history)

Electrical Version

- Analogous circuit can be constructed using capacitor
- It depends on constant voltage of source

Better – Use Natural Frequency

- Mass-spring (very low friction)
- Apply force to mass when it is near its rest position
  - But, only when it is moving in positive direction
  - Force accelerates it in the same direction
- Turns harmonic oscillator into limit cycle
- Timing depends on mass-spring combination
- Almost independent of strength of force

Escapement

- A mechanical equivalent to the scheme above
- Applied to either pendulum or mass-spring
- Also used to operate the hands of the clock in classical applications
- Mechatronics style can just take off a signal

Quartz Crystal

- Electronic equivalent
- Very stable
  - But needs elaborate compensation to meet real clock accuracy – this computer is off by about a minute a week!
- Quartz is piezo-electric
  - Strain <-> voltage
- Has mechanical natural frequency
- Use piezo effect to insert energy

Real Time Software

- Time is most commonly used incrementally
  - Do something every XX time units
  - Do something XX time units after some event
- Sometimes it's absolute
  - Time-of-day
- Time coordination with multiple processors at same hierarchy level (see the movie *Gallipoli*)
  - Avoiding time jumps (missed events, events happen twice)

Time: Real and Otherwise

- User level: time is always available
- *GetTimeNow() or similar function*
- Result depends on context:
  - Simulated time
  - Calibrated time
  - Real time based on several different measurement methods

Using Time

- Simple Dispatcher (Group-priority software)
  - Distributes scans to tasks
  - Does not know anything about task context
  - Internal task structure determines actions base on events (time is an event)
- Scheduler (TranRun4 software)
  - Knows when a task wants to run
  - Only gives it scans on schedule
  - Time can be implicit in the task

MeasuringTime

- Simulation
  - Internal time increment
  - Incremented with each scan
  - flat model: all scans are equal
  - Δ t small to simulate a fast computer (simulate slowly)
  - Δ t large to simulate a slow computer (simulate fast)
- “Time” is a number in memory
- No relation to reality

Calibrated Time

- Calibrated time
  - Same code as simulated time
  - Calibrate program with real clock
  - Easy to implement
  - Adjust Δ t until running time matches clock time
  - Not very accurate for short or long periods
  - OK in some cases where mid-length intervals are needed
  - Must be recalibrated with change in code or computer

Free Running Clock

- Use the oscillator/counter combination
- Free running clock
  - Clock, counter, register to read counter
  - Read register to find out current time
  - PC: clock at about 1.18 MHz
  - Time resolution is ~1 microsecond
  - Easy to use and accurate enough for mechanical system control

Free Running Clock: Rollover

- PC clock/counter is 16-bit
- Counts to limit in about 60 milliseconds
- Counter rolls over and keeps counting
- Two problems:
  - Handling rollover arithmetically (free running means rollover is not predictable)
  - Not loosing track of a complete rollover (inaccurate time) – the used car syndrome

Rollover Arithmetic

- Get time from differences
- In 2's complement arithmetic, differences are correct across a rollover
  - counter is unsigned
  - difference is 2's complement (signed)
  - Example (4-bit counter): 0010 - 1101 (curr - past)

```
  0010
- 1101
  ----
  0101  (through away the final borrow)
```

Rollover . . .

- 2 – (-3) = 5; trick for 2's comp negative #s:
  - Bitwise complement, add 1; 1101->0010->0011 (-3)
- Use the resulting difference to get running time
- Add it to current time using wide word
- How wide: 32 bit, unsigned gives 4x10^9 counts
  - 4x10^9 counts / (~3.6x10^9 counts/hr) = ~1 hour
- Not nearly enough for industrial applications
- Reference: the year 2,000 problem!

Missing Rollovers

- Arithmetic handles one rollover
- Two (or more) rollovers spell trouble!
- No way to know about it
- Time becomes inaccurate
- Solutions:
  - Use “guard” interval
  - Use interrupts

Guard Interval

- Software clock is serviced whenever *GetTimeNow()* is called
- Difference from last service is computed
- If difference > max-interval / K => error
- Statistical - will not guarantee to catch errors
- In practice - very good, K: 2 to 4
- Balance worst case latency with how fast error is found (false positive vs. false negative)

Service Clock with Interrupt

- Interrupt: hardware-based preemptive mechanism
- More details later
- Simple usage: preempt user software to service clock
- Guarantees accurate free-running time regardless of task execution time
- Simplest case: count interrupts (but cruder time granulation)

PC Clocks

- Windows-95, Windows-NT
  - Interrupt serviced free-running clock
  - OS does interrupts
  - ~1 microsecond resolution
- DOS (not in Windows)
  - Free-running clock with guard interval
  - Or user-written interrupt service
  - ~1 microsecond resolution
  - with K=2, ~30 ms maximum execution time

Testing the Time Environment

```
while(Time < EndTime)
    {  // Produces Histograms of run time
    Time = GetTimeNow();
    delt = Time - LastTime;
    LastTime = Time;
    for(j = 0; j < nbin; j++)
      {
      if(delt <= values[j])
        {
        occur[j]++;
        break;
        }
      }
    IncrementTime();  // For internal time mode
    }
```

Testing

- Run this in several environments to see differences in time behavior
- Win-95 w/IDE, Win-95 File manager, DOS
- All done on same Pentium-133 portable
- Total run time is 10 seconds

Result:: Win32/console/Win-95

0.000005 0 --- 1st column: time in seconds; 2nd number of scans
0.000010 1026347
0.000019 2228
0.000037 1252
0.000073 585
0.000142 420
0.000277 118
0.000541 370
0.001057 18
0.002064 9
0.004029 3
0.007868 4
0.015364 20
0.030000 9
> 0.030000 4

Consistency of Results

- This is real time!
- Simple, but still real time
- Results depend on asynchronous events
- Program code is the same every iteration
- However, there is a broad distribution of execution times!
- Here are two more runs done using identical procedures:

Two More Runs . . .

0.000005 0	0.000005 0
0.000010 3509	0.000010 903226
0.000019 839434	0.000019 2509
0.000037 1423	0.000037 1000
0.000073 459	0.000073 389
0.000142 446	0.000142 462
0.000277 126	0.000277 125
0.000541 350	0.000541 361
0.001057 35	0.001057 40
0.002064 6	0.002064 6
0.004029 5	0.004029 4
0.007868 4	0.007868 5
0.015364 9	0.015364 8
0.030000 21	0.030000 28
> 0.030000 16	> 0.030000 14

Further Environmental Effect

- The previous were run from the Borland IDE (integrated develpment environment)
- Here are two runs that are done from the file manager . . .

Run From File Manager

0.000005	11498	0.000005	0
0.000010	1221676	0.000010	1245707
0.000019	1098	0.000019	1118
0.000037	672	0.000037	666
0.000073	520	0.000073	364
0.000142	330	0.000142	392
0.000277	55	0.000277	57
0.000541	345	0.000541	337
0.001057	145	0.001057	175
0.002064	54	0.002064	29
0.004029	22	0.004029	3
0.007868	3	0.007868	1
0.015364	7	0.015364	7
0.030000	20	0.030000	22
> 0.030000	18	> 0.030000	16

From File Manager

- These are significantly faster
- Still broadly distributed
- Still different from run-to-run

Real Time in DOS

- Now try DOS (reboot, not from Windows)
- Uses similar timer, but maintained by program instead of OS
- DOS remains popular for home-grown real time systems
- Embedded PCs popular and easy to work with
- DOS is cheap
- Slowly disappearing

DOS Results

0.000005 16378
0.000010 1971682
0.000019 1
0.000037 0
0.000073 0
0.000142 0
0.000277 0
0.000541 0
0.001057 0
0.002064 0
0.004029 0
0.007868 0
0.015364 0
0.030000 0
> 0.030000 0

Time Performance Conclusion

- Environment has major effect on performance
- DOS is fastest
- DOS interferes least
- Win-95 has relatively long OS events that preempt user program (Win-NT/2000 is similar)
- Win-95 or NT good for prototype debugging or slow processes, not for production

Time Synchronization

- Cooperative activities can be synchronous or asynchronous
- If they are to be coordinated, asynchronous activities require a “handshake”
- If handshake is across a network it causes timing errors
- Activities can be active (do something) or passive (record something)
- In either case, time synchronization can give closer coordination than asynchronous handshake

Network Time Protocols

- Time stamping and its uses
- Network Time Protocol (NTP), Simple NTP (SNTP) and IEEE-1588
- SNTP is used for Windows time synchronization
- Accuracy depends on network delays
- IEEE-1588 uses time stamping at lowest possible level to eliminate software errors
- Accuracy around 1 micro-sec or better